

# Package ‘broom’

December 16, 2020

**Type** Package

**Title** Convert Statistical Objects into Tidy Tibbles

**Version** 0.7.3

**Description** Summarizes key information about statistical objects in tidy tibbles. This makes it easy to report results, create plots and consistently work with large numbers of models at once. Broom provides three verbs that each provide different types of information about a model. `tidy()` summarizes information about model components such as coefficients of a regression. `glance()` reports information about an entire model, such as goodness of fit measures like AIC and BIC. `augment()` adds information about individual observations to a dataset, such as fitted values or influence measures.

**License** MIT + file LICENSE

**URL** <https://broom.tidymodels.org/>, <https://github.com/tidymodels/broom>

**BugReports** <https://github.com/tidymodels/broom/issues>

**Depends** R (>= 3.1)

**Imports** backports, dplyr (>= 1.0.0), ellipsis, generics (>= 0.0.2), glue, methods, purrr, rlang, stringr, tibble (>= 3.0.0), tidyr (>= 1.0.0)

**Suggests** AER, akima, AUC, bbmle, betareg, biglm, binGroup, boot, btergm, car, caret, cluster, coda, covr, drc, e1071, emmeans, epiR, ergm (>= 3.10.4), fixest (>= 0.5.0), gam (>= 1.15), gamlss, gamlss.data, gamlss.dist, gee, geopack, ggplot2, glmnet, glmnetUtils, gmm, Hmisc, irlba, joineRML, Kendall, knitr, ks, Lahman, lavaan, leaps, lm.beta, lme4, lmodel2, lmtest (>= 0.9.38), lsmeans, maps, maptools, margins, MASS, Matrix, mclust, mediation, metafor, mfx, mgcv, mlogit, modeldata, modeltests, muhaz, multcomp, network, nnet, orcutt (>= 2.2), ordinal, plm, poLCA, psych, quantreg, rgeos, rmarkdown, robust, robustbase, rsample, sandwich, sp, spdep, spatialreg, speedglm, spelling, survey, survival, systemfit, testthat (>= 2.1.0), tseries, zoo

**VignetteBuilder** knitr

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**Language** en-US

**Collate** 'aaa-documentation-helper.R' 'null-and-default-tidiers.R'  
 'aer-tidiers.R' 'auc-tidiers.R' 'base-tidiers.R'  
 'bbmle-tidiers.R' 'betareg-tidiers.R' 'biglm-tidiers.R'  
 'bingroup-tidiers.R' 'boot-tidiers.R' 'broom-package.R'  
 'broom.R' 'btergm-tidiers.R' 'car-tidiers.R' 'caret-tidiers.R'  
 'data-frame-tidiers.R' 'deprecated-0-7-0.R'  
 'deprecated-0-7-1.R' 'drc-tidiers.R' 'emmeans-tidiers.R'  
 'epiR-tidiers.R' 'ergm-tidiers.R' 'fixest-tidiers.R'  
 'gam-tidiers.R' 'geepack-tidiers.R'  
 'glmnet-cv-glmnet-tidiers.R' 'glmnet-glmnet-tidiers.R'  
 'gmm-tidiers.R' 'hmisc-tidiers.R' 'joinerml-tidiers.R'  
 'kendall-tidiers.R' 'ks-tidiers.R' 'lavaan-tidiers.R' 'leaps.R'  
 'list-irlba.R' 'list-optim-tidiers.R' 'list-svd-tidiers.R'  
 'list-tidiers.R' 'list-xyz-tidiers.R' 'lm-beta-tidiers.R'  
 'lmodel2-tidiers.R' 'lmtest-tidiers.R' 'maps-tidiers.R'  
 'margins-tidiers.R' 'mass-fitdistr-tidiers.R'  
 'mass-polr-tidiers.R' 'mass-ridgelm-tidiers.R'  
 'stats-lm-tidiers.R' 'mass-rlm-tidiers.R' 'matrix-tidiers.R'  
 'mclust-tidiers.R' 'mediate-tidiers.R' 'mfx-tidiers.R'  
 'mgcv-tidiers.R' 'mlogit-tidiers.R' 'muhaz-tidiers.R'  
 'multcomp-tidiers.R' 'nnet-tidiers.R' 'nobs.R'  
 'orcutt-tidiers.R' 'ordinal-clm-tidiers.R'  
 'ordinal-clmm-tidiers.R' 'pam-tidiers.R' 'plm-tidiers.R'  
 'polca-tidiers.R' 'psych-tidiers.R' 'stats-nls-tidiers.R'  
 'quantreg-nlrq-tidiers.R' 'quantreg-rq-tidiers.R'  
 'quantreg-rqs-tidiers.R' 'rma-tidiers.R'  
 'robust-glmrob-tidiers.R' 'robust-lmrob-tidiers.R'  
 'robustbase-glmrob-tidiers.R' 'robustbase-lmrob-tidiers.R'  
 'sp-tidiers.R' 'spdep-tidiers.R' 'speedglm-speedglm-tidiers.R'  
 'speedglm-speedlm-tidiers.R' 'stats-anova-tidiers.R'  
 'stats-arima-tidiers.R' 'stats-decompose-tidiers.R'  
 'stats-factanal-tidiers.R' 'stats-glm-tidiers.R'  
 'stats-hptest-tidiers.R' 'stats-kmeans-tidiers.R'  
 'stats-loess-tidiers.R' 'stats-mlm-tidiers.R'  
 'stats-prcomp-tidiers.R' 'stats-smooth.spline-tidiers.R'  
 'stats-summary-lm-tidiers.R' 'stats-time-series-tidiers.R'  
 'survey-tidiers.R' 'survival-aareg-tidiers.R'  
 'survival-cch-tidiers.R' 'survival-coxph-tidiers.R'  
 'survival-pyears-tidiers.R' 'survival-survdiff-tidiers.R'  
 'survival-survexp-tidiers.R' 'survival-survfit-tidiers.R'  
 'survival-survreg-tidiers.R' 'systemfit-tidiers.R'  
 'tseries-tidiers.R' 'utilities.R' 'zoo-tidiers.R' 'zzz.R'

**NeedsCompilation** no

**Author** David Robinson [aut],  
Alex Hayes [aut, cre] (<<https://orcid.org/0000-0002-4985-5160>>),  
Simon Couch [aut],  
Indrajeet Patil [ctb] (<<https://orcid.org/0000-0003-1995-6531>>),  
Derek Chiu [ctb],  
Matthieu Gomez [ctb],  
Boris Demeshev [ctb],  
Dieter Menne [ctb],  
Benjamin Nutter [ctb],  
Luke Johnston [ctb],  
Ben Bolker [ctb],  
Francois Briatte [ctb],  
Jeffrey Arnold [ctb],  
Jonah Gabry [ctb],  
Luciano Selzer [ctb],  
Gavin Simpson [ctb],  
Jens Preussner [ctb],  
Jay Hesselberth [ctb],  
Hadley Wickham [ctb],  
Matthew Lincoln [ctb],  
Alessandro Gasparini [ctb],  
Lukasz Komsta [ctb],  
Frederick Novometsky [ctb],  
Wilson Freitas [ctb],  
Michelle Evans [ctb],  
Jason Cory Brunson [ctb],  
Simon Jackson [ctb],  
Ben Whalley [ctb],  
Karissa Whiting [ctb],  
Yves Rosseel [ctb],  
Michael Kuehn [ctb],  
Jorge Cimentada [ctb],  
Erle Holgersen [ctb],  
Karl Dunkle Werner [ctb] (<<https://orcid.org/0000-0003-0523-7309>>),  
Ethan Christensen [ctb],  
Steven Pav [ctb],  
Paul PJ [ctb],  
Ben Schneider [ctb],  
Patrick Kennedy [ctb],  
Lily Medina [ctb],  
Brian Fannin [ctb],  
Jason Muhlenkamp [ctb],  
Matt Lehman [ctb],  
Bill Denney [ctb] (<<https://orcid.org/0000-0002-5759-428X>>),  
Nic Crane [ctb],  
Andrew Bates [ctb],  
Vincent Arel-Bundock [ctb] (<<https://orcid.org/0000-0003-2042-7063>>),

Hideaki Hayashi [ctb],  
Luis Tobalina [ctb],  
Annie Wang [ctb],  
Wei Yang Tham [ctb],  
Clara Wang [ctb],  
Abby Smith [ctb] (<<https://orcid.org/0000-0002-3207-0375>>),  
Jasper Cooper [ctb] (<<https://orcid.org/0000-0002-8639-3188>>),  
E Auden Krauska [ctb] (<<https://orcid.org/0000-0002-1466-5850>>),  
Alex Wang [ctb],  
Malcolm Barrett [ctb] (<<https://orcid.org/0000-0003-0299-5825>>),  
Charles Gray [ctb] (<<https://orcid.org/0000-0002-9978-011X>>),  
Jared Wilber [ctb],  
Vilmantas Gegzna [ctb] (<<https://orcid.org/0000-0002-9500-5167>>),  
Eduard Szoecs [ctb],  
Frederik Aust [ctb] (<<https://orcid.org/0000-0003-4900-788X>>),  
Angus Moore [ctb],  
Nick Williams [ctb],  
Marius Barth [ctb] (<<https://orcid.org/0000-0002-3421-6665>>),  
Bruna Wundervald [ctb] (<<https://orcid.org/0000-0001-8163-220X>>),  
Joyce Cahoon [ctb] (<<https://orcid.org/0000-0001-7217-4702>>),  
Grant McDermott [ctb] (<<https://orcid.org/0000-0001-7883-8573>>),  
Kevin Zarca [ctb],  
Shiro Kuriwaki [ctb] (<<https://orcid.org/0000-0002-5687-2647>>),  
Lukas Wallrich [ctb] (<<https://orcid.org/0000-0003-2121-5177>>),  
James Martherus [ctb] (<<https://orcid.org/0000-0002-8285-3300>>),  
Chuliang Xiao [ctb] (<<https://orcid.org/0000-0002-8466-9398>>),  
Joseph Larmarange [ctb],  
Max Kuhn [ctb],  
Michal Bojanowski [ctb],  
Hakon Malmedal [ctb],  
Clara Wang [ctb],  
Sergio Oller [ctb],  
Luke Sonnet [ctb],  
Jim Hester [ctb],  
Cory Brunson [ctb],  
Ben Schneider [ctb],  
Bernie Gray [ctb] (<<https://orcid.org/0000-0001-9190-6032>>),  
Mara Averick [ctb],  
Aaron Jacobs [ctb],  
Andreas Bender [ctb],  
Sven Templer [ctb],  
Paul-Christian Buerkner [ctb],  
Matthew Kay [ctb],  
Erwan Le Pennec [ctb],  
Johan Junkka [ctb],  
Hao Zhu [ctb],  
Benjamin Soltoff [ctb],  
Zoe Wilkinson Saldana [ctb],

Tyler Littlefield [ctb],  
 Charles T. Gray [ctb],  
 Shabbh E. Banks [ctb],  
 Serina Robinson [ctb],  
 Roger Bivand [ctb],  
 Riinu Ots [ctb],  
 Nicholas Williams [ctb],  
 Nina Jakobsen [ctb],  
 Michael Weylandt [ctb],  
 Lisa Lendway [ctb],  
 Karl Hailperin [ctb],  
 Josue Rodriguez [ctb],  
 Jenny Bryan [ctb],  
 Chris Jarvis [ctb],  
 Greg Macfarlane [ctb],  
 Brian Mannakee [ctb],  
 Drew Tyre [ctb],  
 Shreyas Singh [ctb],  
 Laurens Geffert [ctb],  
 Hong Ooi [ctb],  
 Henrik Bengtsson [ctb],  
 Eduard Szocs [ctb],  
 David Hugh-Jones [ctb],  
 Matthieu Stigler [ctb],  
 Hugo Tavares [ctb] (<<https://orcid.org/0000-0001-9373-2726>>),  
 R. Willem Vervoort [ctb],  
 Brenton M. Wiernik [ctb]

**Maintainer** Alex Hayes <[alexphayes@gmail.com](mailto:alexphayes@gmail.com)>

**Repository** CRAN

**Date/Publication** 2020-12-16 19:20:07 UTC

## R topics documented:

augment.betamfx . . . . .	10
augment.betareg . . . . .	12
augment.clm . . . . .	15
augment.coxph . . . . .	17
augment.decomposed.ts . . . . .	19
augment.drc . . . . .	22
augment.factanal . . . . .	24
augment.fixest . . . . .	26
augment.glm . . . . .	28
augment.glmRob . . . . .	30
augment.glmrob . . . . .	31
augment.htest . . . . .	33
augment.ivreg . . . . .	35
augment.kmeans . . . . .	37

augment.lm	39
augment.lmRob	42
augment.lmrob	44
augment.loess	46
augment.Mclust	48
augment.mfx	50
augment.mjoint	53
augment.mlogit	55
augment.nlrq	57
augment.nls	59
augment.pam	61
augment.plm	63
augment.poLCA	64
augment.polr	67
augment.prcomp	69
augment.rlm	70
augment.rma	72
augment.rq	74
augment.rqs	76
augment.sarlm	78
augment.smooth.spline	80
augment.speedlm	81
augment.stl	83
augment.survreg	84
augment_columns	86
bootstrap	87
confint_tidy	88
data.frame_tidiers	89
durbinWatsonTest_tidiers	91
finish_glance	92
fix_data_frame	93
glance.aareg	93
glance.aov	95
glance.Arima	96
glance.betamfx	97
glance.betareg	99
glance.biglm	100
glance.binDesign	102
glance.cch	103
glance.clm	105
glance.clmm	106
glance.coefest	108
glance.coxph	110
glance.cv.glmnet	112
glance.drc	114
glance.ergm	115
glance.factanal	117
glance.ftdistr	118

glance.fixest . . . . .	120
glance.gam . . . . .	121
glance.garch . . . . .	123
glance.geeglm . . . . .	124
glance.glm . . . . .	125
glance.glmnet . . . . .	126
glance.glmRob . . . . .	128
glance.gmm . . . . .	129
glance.ivreg . . . . .	131
glance.kmeans . . . . .	133
glance.lavaan . . . . .	135
glance.lm . . . . .	137
glance.lmodel2 . . . . .	140
glance.lmRob . . . . .	141
glance.lmrob . . . . .	142
glance.margins . . . . .	144
glance.Mclust . . . . .	146
glance.mfx . . . . .	148
glance.mjoint . . . . .	150
glance.mlogit . . . . .	152
glance.muhaz . . . . .	153
glance.multinom . . . . .	154
glance.nlrq . . . . .	156
glance.nls . . . . .	157
glance.orcutt . . . . .	158
glance.pam . . . . .	160
glance.plm . . . . .	161
glance.poLCA . . . . .	163
glance.polr . . . . .	165
glance.pyyears . . . . .	166
glance.ridgelm . . . . .	168
glance.rlm . . . . .	169
glance.rma . . . . .	171
glance.rq . . . . .	172
glance.sarlm . . . . .	174
glance.smooth.spline . . . . .	175
glance.speedglm . . . . .	177
glance.speedlm . . . . .	178
glance.summary.lm . . . . .	180
glance.survdiff . . . . .	182
glance.survexp . . . . .	184
glance.survfit . . . . .	185
glance.survreg . . . . .	187
glance.svyglm . . . . .	189
glance.svyolr . . . . .	190
glance_optim . . . . .	192
list_tidiers . . . . .	193
null_tidiers . . . . .	194

<code>sparse_tidiers</code>	194
<code>sp_tidiers</code>	195
<code>summary_tidiers</code>	196
<code>tidy.aareg</code>	198
<code>tidy.acf</code>	199
<code>tidy.anova</code>	200
<code>tidy.aov</code>	201
<code>tidy.aovlist</code>	202
<code>tidy.Arima</code>	204
<code>tidy.betamfx</code>	205
<code>tidy.betareg</code>	207
<code>tidy.biglm</code>	208
<code>tidy.binDesign</code>	210
<code>tidy.binWidth</code>	211
<code>tidy.boot</code>	212
<code>tidy.btergm</code>	214
<code>tidy.cch</code>	215
<code>tidy.cld</code>	217
<code>tidy.clm</code>	218
<code>tidy.clmm</code>	220
<code>tidy.coefstest</code>	222
<code>tidy.confint.glht</code>	223
<code>tidy.confusionMatrix</code>	225
<code>tidy.coxph</code>	226
<code>tidy.cv.glmnet</code>	228
<code>tidy.density</code>	230
<code>tidy.dist</code>	231
<code>tidy.drc</code>	232
<code>tidy.emmGrid</code>	234
<code>tidy.epi.2by2</code>	236
<code>tidy.ergm</code>	237
<code>tidy.factanal</code>	239
<code>tidy.ftdistr</code>	240
<code>tidy.fixest</code>	241
<code>tidy.ftable</code>	243
<code>tidy.gam</code>	244
<code>tidy.gamlss</code>	245
<code>tidy.garch</code>	247
<code>tidy.geeglm</code>	248
<code>tidy.glht</code>	250
<code>tidy.glm</code>	251
<code>tidy.glmnet</code>	252
<code>tidy.glmRob</code>	254
<code>tidy.glmrob</code>	255
<code>tidy.gmm</code>	256
<code>tidy.htest</code>	258
<code>tidy.ivreg</code>	260
<code>tidy.kappa</code>	262



tidy.kde	263
tidy.Kendall	265
tidy.kmeans	266
tidy.lavaan	267
tidy.lm	269
tidy.lm.beta	271
tidy.lmodel2	273
tidy.lmRob	275
tidy.lmrob	276
tidy.lsmobj	277
tidy.manova	279
tidy.map	280
tidy.margins	282
tidy.Mclust	284
tidy.mediate	285
tidy.mfx	287
tidy.mjoint	289
tidy.mle2	291
tidy.mlm	292
tidy.mlogit	294
tidy.muhaaz	295
tidy.multinom	296
tidy.nlrq	298
tidy.nls	299
tidy.numeric	300
tidy.orcutt	301
tidy.pairwise.htest	303
tidy.pam	304
tidy.plm	306
tidy.poLCA	307
tidy.polr	309
tidy.power.htest	311
tidy.pcomp	312
tidy.pyears	314
tidy.rcorr	316
tidy.ref.grid	317
tidy.regsubsets	319
tidy.ridgeglm	320
tidy.rlm	321
tidy.rma	322
tidy.roc	324
tidy.rq	325
tidy.rqs	327
tidy.sarlm	328
tidy.spec	329
tidy.speedglm	330
tidy.speedlm	332
tidy.summary.glm	333

tidy.summary.lm	334
tidy.summary_emm	336
tidy.survdifff	338
tidy.survexp	339
tidy.surffit	341
tidy.survreg	342
tidy.svyglm	344
tidy.svyolr	345
tidy.systemfit	347
tidy.table	348
tidy.ts	349
tidy.TukeyHSD	350
tidy.zoo	351
tidy_irlba	353
tidy_optim	355
tidy_svd	356
tidy_xyz	358

**Index****360**


---

augment.betamfx	<i>Augment data with information from a(n) betamfx object</i>
-----------------	---

---

**Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures) that is not meaningfully defined for new observations.

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'betamfx'
augment(
  x,
  data = model.frame(x$fit),
  newdata = NULL,
  type.predict = c("response", "link", "precision", "variance", "quantile"),
  type.residuals = c("sweighted2", "deviance", "pearson", "response", "weighted",
    "sweighted"),
  ...
)
```

## Arguments

<code>x</code>	A <code>betamfx</code> object.
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>newdata</code>	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
<code>type.predict</code>	Character indicating type of prediction to use. Passed to the <code>type</code> argument of <code>betareg::predict.betareg()</code> . Defaults to "response".
<code>type.residuals</code>	Character indicating type of residuals to use. Passed to the <code>type</code> argument of <code>betareg::residuals.betareg()</code> . Defaults to "sweighted2".
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

This `augment` method wraps `augment.betareg()` for `mfx::betamfx()` objects.

## Value

A `tibble::tibble()` with columns:

.cooks	Cooks distance.
.fitted	Fitted or predicted value.
.resid	The difference between observed and fitted values.

**See Also**

[augment.betareg\(\)](#), [mfx::betamfx\(\)](#)

Other mfx tidiers: [augment.mfx\(\)](#), [glance.betamfx\(\)](#), [glance.mfx\(\)](#), [tidy.betamfx\(\)](#), [tidy.mfx\(\)](#)

**Examples**

```
## Not run:
library(mfx)

## Simulate some data
set.seed(12345)
n = 1000
x = rnorm(n)

## Beta outcome
y = rbeta(n, shape1 = plogis(1 + 0.5 * x), shape2 = (abs(0.2*x)))
## Use Smithson and Verkuilen correction
y = (y*(n-1)+0.5)/n

d = data.frame(y,x)
mod_betamfx = betamfx(y ~ x | x, data = d)

tidy(mod_betamfx, conf.int = TRUE)

## Compare with the naive model coefficients of the equivalent betareg call (not run)
# tidy(betamfx(y ~ x | x, data = d), conf.int = TRUE)

augment(mod_betamfx)
glance(mod_betamfx)

## End(Not run)
```

---

augment.betareg

*Augment data with information from a(n) betareg object*

---

**Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object.

Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures) that is not meaningfully defined for new observations.

For convenience, many augment methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'betareg'
augment(
  x,
  data = model.frame(x),
  newdata = NULL,
  type.predict,
  type.residuals,
  ...
)
```

## Arguments

<code>x</code>	A <code>betareg</code> object produced by a call to <code>betareg::betareg()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>newdata</code>	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
<code>type.predict</code>	Character indicating type of prediction to use. Passed to the <code>type</code> argument of the <code>stats::predict()</code> generic. Allowed arguments vary with model class, so be sure to read the <code>predict.my_class</code> documentation.

`type.residuals` Character indicating type of residuals to use. Passed to the `type` argument of `stats::residuals()` generic. Allowed arguments vary with model class, so be sure to read the `residuals.my_class` documentation.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Details

For additional details on Cook's distance, see `stats::cooks.distance()`.

## Value

A `tibble::tibble()` with columns:

<code>.cooks_d</code>	Cooks distance.
<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

## See Also

`augment()`, `betareg::betareg()`

## Examples

```
library(betareg)
data("GasolineYield", package = "betareg")

mod <- betareg(yield ~ batch + temp, data = GasolineYield)

mod
tidy(mod)
tidy(mod, conf.int = TRUE)
tidy(mod, conf.int = TRUE, conf.level = .99)

augment(mod)

glance(mod)
```

augment.clm

*Augment data with information from a(n) clm object***Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

**Usage**

```
## S3 method for class 'clm'
augment(
  x,
  data = model.frame(x),
  newdata = NULL,
  type.predict = c("prob", "class"),
  ...
)
```

**Arguments**

`x` A `clm` object returned from `ordinal::clm()`.

data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
type.predict	Which type of prediction to compute, either "prob" or "class", passed to <code>ordinal::predict.clm()</code> . Defaults to "prob".
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

### See Also

`tidy`, `ordinal::clm()`, `ordinal::predict.clm()`

Other ordinal tidiers: `augment.polr()`, `glance.clmm()`, `glance.clm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clmm()`, `tidy.clm()`, `tidy.polr()`, `tidy.svyolr()`

### Examples

```
library(ordinal)

fit <- clm(rating ~ temp * contact, data = wine)

tidy(fit)
tidy(fit, conf.int = TRUE, conf.level = 0.9)
tidy(fit, conf.int = TRUE, conf.type = "Wald", exponentiate = TRUE)

glance(fit)
augment(fit, type.predict = "prob")
augment(fit, type.predict = "class")

fit2 <- clm(rating ~ temp, nominal = ~contact, data = wine)
tidy(fit2)
glance(fit2)
```



augment.coxph

*Augment data with information from a(n) coxph object*

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'coxph'
augment(
  x,
  data = NULL,
  newdata = NULL,
  type.predict = "lp",
  type.residuals = "martingale",
  ...
)
```

## Arguments

`x` A coxph object returned from `survival::coxph()`.

data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
type.predict	Character indicating type of prediction to use. Passed to the <code>type</code> argument of the <code>stats::predict()</code> generic. Allowed arguments vary with model class, so be sure to read the <code>predict.my_class</code> documentation.
type.residuals	Character indicating type of residuals to use. Passed to the <code>type</code> argument of <code>stats::residuals()</code> generic. Allowed arguments vary with model class, so be sure to read the <code>residuals.my_class</code> documentation.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

When the modeling was performed with `na.action = "na.omit"` (as is the typical default), rows with NA in the initial data are omitted entirely from the augmented data frame. When the modeling was performed with `na.action = "na.exclude"`, one should provide the original data as a second argument, at which point the augmented data will contain those rows (typically with NAs in place of the new columns). If the original data is not provided to `augment()` and `na.action = "na.exclude"`, a warning is raised and the incomplete rows are dropped.

## Value

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.

## See Also

[stats::na.action](#)

[augment\(\)](#), [survival::coxph\(\)](#)

Other coxph tidiers: [glance.coxph\(\)](#), [tidy.coxph\(\)](#)

Other survival tidiers: [augment.survreg\(\)](#), [glance.aareg\(\)](#), [glance.cch\(\)](#), [glance.coxph\(\)](#), [glance.pyyears\(\)](#), [glance.survdiff\(\)](#), [glance.survexp\(\)](#), [glance.survfit\(\)](#), [glance.survreg\(\)](#),

```
tidy.aareg(), tidy.cch(), tidy.coxph(), tidy.pyears(), tidy.survdiff(), tidy.survexp(),  
tidy.survfit(), tidy.survreg()
```

## Examples

```
library(survival)  
  
cfit <- coxph(Surv(time, status) ~ age + sex, lung)  
  
tidy(cfit)  
tidy(cfit, exponentiate = TRUE)  
  
lp <- augment(cfit, lung)  
risks <- augment(cfit, lung, type.predict = "risk")  
expected <- augment(cfit, lung, type.predict = "expected")  
  
glance(cfit)  
  
# also works on clogit models  
resp <- levels(logan$occupation)  
n <- nrow(logan)  
indx <- rep(1:n, length(resp))  
logan2 <- data.frame(  
  logan[indx, ],  
  id = indx,  
  tocc = factor(rep(resp, each = n))  
)  
  
logan2$case <- (logan2$occupation == logan2$tocc)  
  
cl <- clogit(case ~ tocc + tocc:education + strata(id), logan2)  
tidy(cl)  
glance(cl)  
  
library(ggplot2)  
  
ggplot(lp, aes(age, .fitted, color = sex)) +  
  geom_point()  
  
ggplot(risks, aes(age, .fitted, color = sex)) +  
  geom_point()  
  
ggplot(expected, aes(time, .fitted, color = sex)) +  
  geom_point()
```

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'decomposed.ts'
augment(x, ...)
```

## Arguments

<code>x</code>	A <code>decomposed.ts</code> object returned from <code>stats::decompose()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble` with one row for each observation in the original times series:

<code>.seasonal</code>	The seasonal component of the decomposition.
------------------------	--

.trend            The trend component of the decomposition.  
 .remainder       The remainder, or "random" component of the decomposition.  
 .weight           The final robust weights (stl only).  
 .seasadj          The seasonally adjusted (or "deseasonalised") series.

### See Also

[augment\(\)](#), [stats::decompose\(\)](#)

Other decompose tidiers: [augment.stl\(\)](#)

### Examples

```
# Time series of temperatures in Nottingham, 1920-1939:
nottem

# Perform seasonal decomposition on the data with both decompose
# and stl:
d1 <- stats::decompose(nottem)
d2 <- stats::stl(nottem, s.window = "periodic", robust = TRUE)

# Compare the original series to its decompositions.

cbind(
  broom::tidy(nottem), broom::augment(d1),
  broom::augment(d2)
)

# Visually compare seasonal decompositions in tidy data frames.

library(tibble)
library(dplyr)
library(tidyr)
library(ggplot2)

decomps <- tibble(
  # Turn the ts objects into data frames.
  series = list(as.data.frame(nottem), as.data.frame(nottem)),
  # Add the models in, one for each row.
  decomp = c("decompose", "stl"),
  model = list(d1, d2)
) %>%
  rowwise() %>%
  # Pull out the fitted data using broom::augment.
  mutate(augment = list(broom::augment(model))) %>%
  ungroup() %>%
  # Unnest the data frames into a tidy arrangement of
  # the series next to its seasonal decomposition, grouped
  # by the method (stl or decompose).
  group_by(decomp) %>%
  unnest(c(series, augment)) %>%
```

```

mutate(index = 1:n()) %>%
ungroup() %>%
select(decomp, index, x, adjusted = .seasadj)

ggplot(decomps) +
  geom_line(aes(x = index, y = x), colour = "black") +
  geom_line(aes(
    x = index, y = adjusted, colour = decomp,
    group = decomp
  ))

```

---

augment.drc

*Augment data with information from a(n) drc object*


---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```

## S3 method for class 'drc'
augment(
  x,

```

```

  data = NULL,
  newdata = NULL,
  se_fit = FALSE,
  conf.int = FALSE,
  conf.level = 0.95,
  ...
)

```

## Arguments

x	A drc object produced by a call to <code>drc::drm()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
se_fit	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to FALSE.
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

<code>.cooksd</code>	Cooks distance.
<code>.fitted</code>	Fitted or predicted value.
<code>.lower</code>	Lower bound on interval for fitted values.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.
<code>.upper</code>	Upper bound on interval for fitted values.

**See Also**

[augment\(\)](#), [drc::drm\(\)](#)

Other drc tidiers: [glance.drc\(\)](#), [tidy.drc\(\)](#)

**Examples**

```
library(drc)

mod <- drm(dead / total ~ conc, type,
  weights = total, data = selenium, fct = LL.2(), type = "binomial"
)

tidy(mod)
tidy(mod, conf.int = TRUE)

glance(mod)

augment(mod, selenium)
```

---

augment.factanal

*Augment data with information from a(n) factanal object*

---

**Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and



`survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'factanal'
augment(x, data, ...)
```

## Arguments

<code>x</code>	A factanal object created by <code>stats::factanal()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

When data is not supplied `augment.factanal` returns one row for each observation, with a factor score column added for each factor `X`, (`.fsX`). This is because `stats::factanal()`, unlike other stats methods like `stats::lm()`, does not retain the original data.

When data is supplied, `augment.factanal` returns one row for each observation, with a factor score column added for each factor `X`, (`.fsX`).

## See Also

`augment()`, `stats::factanal()`

Other factanal tidiers: `glance.factanal()`, `tidy.factanal()`

augment.fixest

*Augment data with information from a(n) fixest object*

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'fixest'
augment(
  x,
  data = NULL,
  newdata = NULL,
  type.predict = c("link", "response"),
  type.residuals = c("response", "deviance", "pearson", "working"),
  ...
)
```

## Arguments

`x` A `fixest` object returned from any of the `fixest` estimators

data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
type.predict	Passed to <code>predict.fixest</code> type argument. Defaults to "link" (like <code>predict.glm</code> ).
type.residuals	Passed to <code>predict.fixest</code> type argument. Defaults to "response" (like <code>residuals.lm</code> , but unlike <code>residuals.glm</code> ).
...	Additional arguments passed to <code>summary</code> and <code>confint</code> . Important arguments are <code>se</code> and <code>cluster</code> . Other arguments are <code>dof</code> , <code>exact_dof</code> , <code>forceCovariance</code> , and <code>keepBounded</code> . See <code>summary.fixest</code> .

### Value

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

### Note

Important note: `fixest` models do not include a copy of the input data, so you must provide it manually.

`augment.fixest` only works for `fixest::feols()`, `fixest::feglm()`, and `fixest::femlm()` models. It does not work with results from `fixest::fenegbin()`, `fixest::feNmlm()`, or `fixest::fepois()`.

### See Also

`augment()`, `fixest::feglm()`, `fixest::femlm()`, `fixest::feols()`

Other `fixest` tidiers: `tidy.fixest()`

### Examples

```
library(fixest)
gravity <- feols(log(Euros) ~ log(dist_km) | Origin + Destination + Product + Year, trade)

tidy(gravity)
glance(gravity)
augment(gravity, trade)

## To get robust or clustered SEs, users can either:
# 1) Or, specify the arguments directly in the tidy() call
tidy(gravity, conf.int = TRUE, cluster = c("Product", "Year"))
```

```

tidy(gravity, conf.int = TRUE, se = "threeway")
# 2) Feed tidy() a summary.fixest object that has already accepted these arguments
gravity_summ <- summary(gravity, cluster = c("Product", "Year"))
tidy(gravity_summ, conf.int = TRUE)
# Approach (1) is preferred.

## The other fixest methods all work similarly. For example:
gravity_pois <- feglm(Euros ~ log(dist_km) | Origin + Destination + Product + Year, trade)
tidy(gravity_pois)
glance(gravity_pois)
augment(gravity_pois, trade)

```

---

augment.glm

---

*Augment data with information from a(n) glm object*


---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

**Usage**

```
## S3 method for class 'glm'
augment(
  x,
  data = model.frame(x),
  newdata = NULL,
  type.predict = c("link", "response", "terms"),
  type.residuals = c("deviance", "pearson"),
  se_fit = FALSE,
  ...
)
```

**Arguments**

x	A glm object returned from <code>stats::glm()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
type.predict	Passed to <code>stats::predict.glm()</code> type argument. Defaults to "link".
type.residuals	Passed to <code>stats::residuals.glm()</code> and to <code>stats::rstandard.glm()</code> type arguments. Defaults to "deviance".
se_fit	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

**Details**

If the weights for any of the observations in the model are 0, then columns ".infl" and ".hat" in the result will be 0 for those observations.

A `.resid` column is not calculated when data is specified via the newdata argument.

**Value**

A `tibble::tibble()` with columns:

<code>.cooks</code>	Cooks distance.
<code>.fitted</code>	Fitted or predicted value.
<code>.hat</code>	Diagonal of the hat matrix.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.
<code>.sigma</code>	Estimated residual standard deviation when corresponding observation is dropped from model.
<code>.std.resid</code>	Standardised residuals.

**See Also**

`stats::glm()`

Other lm tidiers: `augment.lm()`, `glance.glm()`, `glance.lm()`, `glance.summary.lm()`, `glance.svyglm()`, `tidy.glm()`, `tidy.lm.beta()`, `tidy.lm()`, `tidy.mlm()`, `tidy.summary.lm()`

---

`augment.glmRob`

*Augment data with information from a(n) glmRob object*

---

**Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and

`survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'glmRob'
augment(x, ...)
```

## Arguments

<code>x</code>	Unused.
<code>...</code>	Unused.

---

<code>augment.glmrob</code>	<i>Augment data with information from a(n) glmrob object</i>
-----------------------------	--

---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

**Usage**

```
## S3 method for class 'glmrob'
augment(
  x,
  data = model.frame(x),
  newdata = NULL,
  type.predict = c("link", "response"),
  type.residuals = c("deviance", "pearson"),
  se_fit = FALSE,
  ...
)
```

**Arguments**

x	A <code>glmrob</code> object returned from <code>robustbase::glmrob()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
type.predict	Character indicating type of prediction to use. Passed to the <code>type</code> argument of the <code>stats::predict()</code> generic. Allowed arguments vary with model class, so be sure to read the <code>predict.my_class</code> documentation.
type.residuals	Character indicating type of residuals to use. Passed to the <code>type</code> argument of <code>stats::residuals()</code> generic. Allowed arguments vary with model class, so be sure to read the <code>residuals.my_class</code> documentation.
se_fit	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to <code>FALSE</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Details**

For tidiers for robust models from the **MASS** package see `tidy.rlm()`.



**Value**

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

**See Also**

`robustbase::glmrob()`

Other robustbase tidiers: `augment.lmrob()`, `glance.lmrob()`, `tidy.glmrob()`, `tidy.lmrob()`

---

<code>augment.htest</code>	<i>Augment data with information from a(n) htest object</i>
----------------------------	---

---

**Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

**Usage**

```
## S3 method for class 'htest'
augment(x, ...)
```

**Arguments**

- `x` An htest objected, such as those created by `stats::cor.test()`, `stats::t.test()`, `stats::wilcox.test()`, `stats::chisq.test()`, etc.
- `...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Details**

See `stats::chisq.test()` for more details on how residuals are computed.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>.observed</code>	Observed count.
<code>.prop</code>	Proportion of the total.
<code>.row.prop</code>	Row proportion (2 dimensions table only).
<code>.col.prop</code>	Column proportion (2 dimensions table only).
<code>.expected</code>	Expected count under the null hypothesis.
<code>.resid</code>	Pearson residuals.
<code>.std.resid</code>	Standardized residual.

**See Also**

`augment()`, `stats::chisq.test()`

Other htest tidiers: `tidy.htest()`, `tidy.pairwise.htest()`, `tidy.power.htest()`

**Examples**

```
tt <- t.test(rnorm(10))
tidy(tt)
glance(tt) # same output for all htests

tt <- t.test(mpg ~ am, data = mtcars)
tidy(tt)

wt <- wilcox.test(mpg ~ am, data = mtcars, conf.int = TRUE, exact = FALSE)
tidy(wt)

ct <- cor.test(mtcars$wt, mtcars$mpg)
tidy(ct)
```

```
chit <- chisq.test(xtabs(Freq ~ Sex + Class, data = as.data.frame(Titanic)))
tidy(chit)
augment(chit)
```

---

augment.ivreg

*Augment data with information from a(n) ivreg object*


---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many augment methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'ivreg'
augment(x, data = model.frame(x), newdata = NULL, ...)
```

## Arguments

`x` An ivreg object created by a call to `AER::ivreg()`.

`data` A `base::data.frame` or `tibble::tibble()` containing the original data that was used to produce the object `x`. Defaults to `stats::model.frame(x)` so that `augment(my_fit)` returns the augmented original data. **Do not** pass new data

to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.

newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

### Details

This tidier currently only supports `ivreg`-classed objects outputted by the `AER` package. The `ivreg` package also outputs objects of class `ivreg`, and will be supported in a later release.

### Value

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

### See Also

`augment()`, `AER::ivreg()`

Other `ivreg` tidiers: `glance.ivreg()`, `tidy.ivreg()`

### Examples

```
library(AER)

data("CigarettesSW", package = "AER")

ivr <- ivreg(
  log(packs) ~ income | population,
  data = CigarettesSW,
  subset = year == "1995"
)

summary(ivr)

tidy(ivr)
tidy(ivr, conf.int = TRUE)
tidy(ivr, conf.int = TRUE, instruments = TRUE)
```

```
augment(ivr)
augment(ivr, data = CigarettesSW)
augment(ivr, newdata = CigarettesSW)

glance(ivr)
```

---

```
augment.kmeans
```

```
Augment data with information from a(n) kmeans object
```

---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'kmeans'
augment(x, data, ...)
```

## Arguments

`x` A `kmeans` object created by `stats::kmeans()`.

`data` A `base::data.frame` or `tibble::tibble()` containing the original data that was used to produce the object `x`. Defaults to `stats::model.frame(x)` so that `augment(my_fit)` returns the augmented original data. **Do not** pass new data to the `data` argument. Augment will report information such as influence and cooks distance for data passed to the `data` argument. These measures are only defined for the original training data.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with columns:

`.cluster` Cluster assignment.

### See Also

`augment()`, `stats::kmeans()`

Other kmeans tidiers: `glance.kmeans()`, `tidy.kmeans()`

### Examples

```
## Not run:
library(cluster)
library(dplyr)

library(modeldata)
data(hpc_data)

x <- hpc_data[, 2:5]

fit <- pam(x, k = 4)

tidy(fit)
glance(fit)
augment(fit, x)

## End(Not run)
```

augment.lm

*Augment data with information from a(n) lm object*

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'lm'
augment(
  x,
  data = model.frame(x),
  newdata = NULL,
  se_fit = FALSE,
  interval = c("none", "confidence", "prediction"),
  ...
)
```

## Arguments

`x` An `lm` object created by `stats::lm()`.

data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
se_fit	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to <code>FALSE</code> .
interval	Character indicating the type of confidence interval columns to be added to the augmented output. Passed on to <code>predict()</code> and defaults to <code>"none"</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

When the modeling was performed with `na.action = "na.omit"` (as is the typical default), rows with NA in the initial data are omitted entirely from the augmented data frame. When the modeling was performed with `na.action = "na.exclude"`, one should provide the original data as a second argument, at which point the augmented data will contain those rows (typically with NAs in place of the new columns). If the original data is not provided to `augment()` and `na.action = "na.exclude"`, a warning is raised and the incomplete rows are dropped.

Some unusual `lm` objects, such as `rlm` from MASS, may omit `.cooksd` and `.std.resid`. `gam` from `mgcv` omits `.sigma`.

When `newdata` is supplied, only returns `.fitted`, `.resid` and `.se.fit` columns.

## Value

A `tibble::tibble()` with columns:

<code>.cooksd</code>	Cooks distance.
<code>.fitted</code>	Fitted or predicted value.
<code>.hat</code>	Diagonal of the hat matrix.
<code>.lower</code>	Lower bound on interval for fitted values.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.
<code>.sigma</code>	Estimated residual standard deviation when corresponding observation is dropped from model.



.std.resid	Standardised residuals.
.upper	Upper bound on interval for fitted values.

**See Also**

[stats::na.action](#)

[augment\(\)](#), [stats::predict.lm\(\)](#)

Other lm tidiers: [augment.glm\(\)](#), [glance.glm\(\)](#), [glance.lm\(\)](#), [glance.summary.lm\(\)](#), [glance.svyglm\(\)](#), [tidy.glm\(\)](#), [tidy.lm.beta\(\)](#), [tidy.lm\(\)](#), [tidy.mlml\(\)](#), [tidy.summary.lm\(\)](#)

**Examples**

```
library(ggplot2)
library(dplyr)

mod <- lm(mpg ~ wt + qsec, data = mtcars)

tidy(mod)
glance(mod)

# coefficient plot
d <- tidy(mod, conf.int = TRUE)

ggplot(d, aes(estimate, term, xmin = conf.low, xmax = conf.high, height = 0)) +
  geom_point() +
  geom_vline(xintercept = 0, lty = 4) +
  geom_errorbarh()

# Aside: There are tidy() and glance() methods for lm.summary objects too.
# This can be useful when you want to conserve memory by converting large lm
# objects into their leaner summary.lm equivalents.
s <- summary(mod)
tidy(s, conf.int = TRUE)
glance(s)

augment(mod)
augment(mod, mtcars, interval = "confidence")

# predict on new data
newdata <- mtcars %>%
  head(6) %>%
  mutate(wt = wt + 1)
augment(mod, newdata = newdata)

# ggplot2 example where we also construct 95% prediction interval
mod2 <- lm(mpg ~ wt, data = mtcars) ## simpler bivariate model since we're plotting in 2D
au <- augment(mod2, newdata = newdata, interval = "prediction")

ggplot(au, aes(wt, mpg)) +
```

```

geom_point() +
geom_line(aes(y = .fitted)) +
geom_ribbon(aes(ymin = .lower, ymax = .upper), col = NA, alpha = 0.3)

# predict on new data without outcome variable. Output does not include .resid
newdata <- newdata %>%
  select(-mpg)
augment(mod, newdata = newdata)

au <- augment(mod, data = mtcars)

ggplot(au, aes(.hat, .std.resid)) +
  geom_vline(size = 2, colour = "white", xintercept = 0) +
  geom_hline(size = 2, colour = "white", yintercept = 0) +
  geom_point() +
  geom_smooth(se = FALSE)

plot(mod, which = 6)
ggplot(au, aes(.hat, .cooks)) +
  geom_vline(xintercept = 0, colour = NA) +
  geom_abline(slope = seq(0, 3, by = 0.5), colour = "white") +
  geom_smooth(se = FALSE) +
  geom_point()

# column-wise models
a <- matrix(rnorm(20), nrow = 10)
b <- a + rnorm(length(a))
result <- lm(b ~ a)
tidy(result)

```

---

augment.lmRob

---

*Augment data with information from a(n) lmRob object*


---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures) that is not meaningfully defined for new observations.

For convenience, many augment methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'lmRob'
augment(x, data = model.frame(x), newdata = NULL, ...)
```

## Arguments

<code>x</code>	A <code>lmRob</code> object returned from <code>robust::lmRob()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. <code>augment</code> will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>newdata</code>	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

For tidiers for robust models from the **MASS** package see `tidy.rlm()`.

## See Also

`robust::lmRob()`

Other robust tidiers: `glance.glmRob()`, `glance.lmRob()`, `tidy.glmRob()`, `tidy.lmRob()`

## Examples

```
library(robust)
m <- lmRob(mpg ~ wt, data = mtcars)

tidy(m)
augment(m)
glance(m)
```

---

augment.lmrob

*Augment data with information from a(n) lmrob object*

---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'lmrob'
augment(x, data = model.frame(x), newdata = NULL, se_fit = FALSE, ...)
```

**Arguments**

x	A <code>lmrob</code> object returned from <code>robustbase::lmrob()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to <code>NULL</code> , indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
se_fit	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to <code>FALSE</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

**Details**

For tidiers for robust models from the **MASS** package see `tidy.rlm()`.

**Value**

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

**See Also**

`robustbase::lmrob()`

Other robustbase tidiers: `augment.glmrob()`, `glance.lmrob()`, `tidy.glmrob()`, `tidy.lmrob()`

**Examples**

```
library(robustbase)
# From the robustbase::lmrob examples:
data(coleman)
set.seed(0)

m <- robustbase::lmrob(Y ~ ., data = coleman)
tidy(m)
```

```

augment(m)
glance(m)

# From the robustbase::glmrob examples:
data(carrots)
Rfit <- glmrob(cbind(success, total - success) ~ logdose + block,
  family = binomial, data = carrots, method = "Mqle",
  control = glmrobMqle.control(tcc = 1.2)
)
tidy(Rfit)
augment(Rfit)

```

---

augment.loess	<i>Tidy a(n) loess object</i>
---------------	-------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'loess'
augment(x, data = model.frame(x), newdata = NULL, se_fit = FALSE, ...)

```

## Arguments

x	A loess objects returned by <code>stats::loess()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
se_fit	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

## Details

When the modeling was performed with `na.action = "na.omit"` (as is the typical default), rows with NA in the initial data are omitted entirely from the augmented data frame. When the modeling was performed with `na.action = "na.exclude"`, one should provide the original data as a second argument, at which point the augmented data will contain those rows (typically with NAs in place of the new columns). If the original data is not provided to `augment()` and `na.action = "na.exclude"`, a warning is raised and the incomplete rows are dropped.

Note that loess objects by default will not predict on data outside of a bounding hypercube defined by the training data unless the original loess object was fit with `control = loess.control(surface = "direct")`. See `stats::predict.loess()` for details.

## Value

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.

## See Also

`stats::na.action`

`augment()`, `stats::loess()`, `stats::predict.loess()`

## Examples

```
lo <- loess(
  mpg ~ hp + wt,
  mtcars,
  control = loess.control(surface = "direct")
)

augment(lo)

# with all columns of original data
augment(lo, mtcars)

# with a new dataset
augment(lo, newdata = head(mtcars))
```

---

 augment.Mclust

 Augment data with information from a(n) Mclust object
 

---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'Mclust'
augment(x, data = NULL, ...)
```

## Arguments

<code>x</code>	An <code>Mclust</code> object return from <code>mclust::Mclust()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.



... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with columns:

<code>.class</code>	Predicted class.
<code>.uncertainty</code>	The uncertainty associated with the classification. Equal to one minus the model class probability.

## See Also

`augment()`, `mclust::Mclust()`

Other `mclust` tidiers: `tidy.Mclust()`

## Examples

```
library(dplyr)
library(mclust)
set.seed(27)

centers <- tibble::tibble(
  cluster = factor(1:3),
  num_points = c(100, 150, 50), # number points in each cluster
  x1 = c(5, 0, -3), # x1 coordinate of cluster center
  x2 = c(-1, 1, -2) # x2 coordinate of cluster center
)

points <- centers %>%
  mutate(
    x1 = purrr::map2(num_points, x1, rnorm),
    x2 = purrr::map2(num_points, x2, rnorm)
  ) %>%
  dplyr::select(-num_points, -cluster) %>%
  tidyr::unnest(c(x1, x2))

m <- mclust::Mclust(points)

tidy(m)
augment(m, points)
glance(m)
```

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'mfx'
augment(
  x,
  data = model.frame(x$fit),
  newdata = NULL,
  type.predict = c("link", "response", "terms"),
  type.residuals = c("deviance", "pearson"),
  se_fit = FALSE,
  ...
)

## S3 method for class 'logitmfx'
augment(
```

```

    x,
    data = model.frame(x$fit),
    newdata = NULL,
    type.predict = c("link", "response", "terms"),
    type.residuals = c("deviance", "pearson"),
    se_fit = FALSE,
    ...
)

## S3 method for class 'negbinmfx'
augment(
  x,
  data = model.frame(x$fit),
  newdata = NULL,
  type.predict = c("link", "response", "terms"),
  type.residuals = c("deviance", "pearson"),
  se_fit = FALSE,
  ...
)

## S3 method for class 'poissonmfx'
augment(
  x,
  data = model.frame(x$fit),
  newdata = NULL,
  type.predict = c("link", "response", "terms"),
  type.residuals = c("deviance", "pearson"),
  se_fit = FALSE,
  ...
)

## S3 method for class 'probitmfx'
augment(
  x,
  data = model.frame(x$fit),
  newdata = NULL,
  type.predict = c("link", "response", "terms"),
  type.residuals = c("deviance", "pearson"),
  se_fit = FALSE,
  ...
)

```

## Arguments

x	A <code>logitmfx</code> , <code>negbinmfx</code> , <code>poissonmfx</code> , or <code>probitmfx</code> object. (Note that <code>betamfx</code> objects receive their own set of tidiers.)
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that

augment(my\_fit) returns the augmented original data. **Do not** pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.

newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
type.predict	Passed to <code>stats::predict.glm()</code> type argument. Defaults to "link".
type.residuals	Passed to <code>stats::residuals.glm()</code> and to <code>stats::rstandard.glm()</code> type arguments. Defaults to "deviance".
se_fit	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

## Details

This generic augment method wraps `augment.glm()` for applicable objects from the mfx package.

## Value

A `tibble::tibble()` with columns:

<code>.cooks</code>	Cooks distance.
<code>.fitted</code>	Fitted or predicted value.
<code>.hat</code>	Diagonal of the hat matrix.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.
<code>.sigma</code>	Estimated residual standard deviation when corresponding observation is dropped from model.
<code>.std.resid</code>	Standardised residuals.

## See Also

`augment.glm()`, `mfx::logitmfx()`, `mfx::negbinmfx()`, `mfx::poissonmfx()`, `mfx::probitmfx()`

Other mfx tidiers: `augment.betamfx()`, `glance.betamfx()`, `glance.mfx()`, `tidy.betamfx()`, `tidy.mfx()`

## Examples

```
## Not run:
library(mfx)

## Get the marginal effects from a logit regression
mod_logmfx <- logitmfx(am ~ cyl + hp + wt, atmean = TRUE, data = mtcars)
tidy(mod_logmfx, conf.int = TRUE)

## Compare with the naive model coefficients of the same logit call (not run)
# tidy(glm(am ~ cyl + hp + wt, family = binomial, data = mtcars), conf.int = TRUE)

augment(mod_logmfx)
glance(mod_logmfx)

## Another example, this time using probit regression
mod_probmfx <- probitmfx(am ~ cyl + hp + wt, atmean = TRUE, data = mtcars)
tidy(mod_probmfx, conf.int = TRUE)
augment(mod_probmfx)
glance(mod_probmfx)

## End(Not run)
```

---

augment.mjoint

*Augment data with information from a(n) mjoint object*


---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and

`survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'mjoint'
augment(x, data = x$data, ...)
```

## Arguments

<code>x</code>	An <code>mjoint</code> object returned from <code>joineRML::mjoint()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

See `joineRML::fitted.mjoint()` and `joineRML::residuals.mjoint()` for more information on the difference between population-level and individual-level fitted values and residuals.

If fitting a joint model with a single longitudinal process, make sure you are using a named `list` to define the formula for the fixed and random effects of the longitudinal submodel.

## Value

A `tibble::tibble()` with one row for each original observation with addition columns:

<code>.fitted_j_0</code>	population-level fitted values for the <code>j</code> -th longitudinal process
<code>.fitted_j_1</code>	individuals-level fitted values for the <code>j</code> -th longitudinal process
<code>.resid_j_0</code>	population-level residuals for the <code>j</code> -th longitudinal process
<code>.resid_j_1</code>	individual-level residuals for the <code>j</code> -th longitudinal process

**Examples**

```

## Not run:
# Fit a joint model with bivariate longitudinal outcomes
library(joineRML)
data(heart.valve)
hvd <- heart.valve[!is.na(heart.valve$log.grad) &
  !is.na(heart.valve$log.lvmi) &
  heart.valve$num <= 50, ]
fit <- mjjoint(
  formLongFixed = list(
    "grad" = log.grad ~ time + sex + hs,
    "lvmi" = log.lvmi ~ time + sex
  ),
  formLongRandom = list(
    "grad" = ~ 1 | num,
    "lvmi" = ~ time | num
  ),
  formSurv = Surv(fuyrs, status) ~ age,
  data = hvd,
  inits = list("gamma" = c(0.11, 1.51, 0.80)),
  timeVar = "time"
)

# Extract the survival fixed effects
tidy(fit)

# Extract the longitudinal fixed effects
tidy(fit, component = "longitudinal")

# Extract the survival fixed effects with confidence intervals
tidy(fit, ci = TRUE)

# Extract the survival fixed effects with confidence intervals based
# on bootstrapped standard errors
bSE <- bootSE(fit, nboot = 5, safe.boot = TRUE)
tidy(fit, boot_se = bSE, ci = TRUE)

# Augment original data with fitted longitudinal values and residuals
hvd2 <- augment(fit)

# Extract model statistics
glance(fit)

## End(Not run)

```

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'mlogit'
augment(x, data = x$model, ...)
```

## Arguments

<code>x</code>	an object returned from <code>mlogit::mlogit()</code> .
<code>data</code>	Not currently used
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

At the moment this only works on the estimation dataset. Need to set it up to predict on another dataset.



**Value**

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.probability</code>	Class probability of modal class.
<code>.resid</code>	The difference between observed and fitted values.

**See Also**

[augment\(\)](#)

Other mlogit tidiers: [glance.mlogit\(\)](#), [tidy.mlogit\(\)](#)

**Examples**

```
## Not run:
library(mlogit)
data("Fishing", package = "mlogit")
Fish <- dfidx(Fishing, varying = 2:9, shape = "wide", choice = "mode")
m <- mlogit(mode ~ price + catch | income, data = Fish)

tidy(m)
augment(m)
glance(m)

## End(Not run)
```

---

augment.nlrq

*Tidy a(n) nlrq object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'nlrq'
augment(x, data = NULL, newdata = NULL, ...)
```

**Arguments**

x	A nlrq object returned from <code>quantreg::nlrq()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

**See Also**

`augment()`, `quantreg::nlrq()`

Other quantreg tidiers: `augment.rqs()`, `augment.rq()`, `glance.nlrq()`, `glance.rq()`, `tidy.nlrq()`, `tidy.rqs()`, `tidy.rq()`

**Examples**

```
n <- nls(mpg ~ k * e^wt, data = mtcars, start = list(k = 1, e = 2))

tidy(n)
augment(n)
glance(n)

library(ggplot2)
ggplot(augment(n), aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted))

newdata <- head(mtcars)
newdata$wt <- newdata$wt + 1
augment(n, newdata = newdata)
```

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'nls'
augment(x, data = NULL, newdata = NULL, ...)
```

## Arguments

<code>x</code>	An <code>nls</code> object returned from <code>stats::nls()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.

`newdata` A `base::data.frame()` or `tibble::tibble()` containing all the original predictors used to create `x`. Defaults to `NULL`, indicating that nothing has been passed to `newdata`. If `newdata` is specified, the `data` argument will be ignored.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Details

`augment.nls` does not currently support confidence intervals due to a lack of support in `stats::predict.nls()`.

### Value

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

### See Also

`tidy`, `stats::nls()`, `stats::predict.nls()`

Other nls tidiers: `glance.nls()`, `tidy.nls()`

### Examples

```
n <- nls(mpg ~ k * e^wt, data = mtcars, start = list(k = 1, e = 2))

tidy(n)
augment(n)
glance(n)

library(ggplot2)
ggplot(augment(n), aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted))

newdata <- head(mtcars)
newdata$wt <- newdata$wt + 1
augment(n, newdata = newdata)
```

augment.pam

*Augment data with information from a(n) pam object*

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'pam'
augment(x, data = NULL, ...)
```

## Arguments

<code>x</code>	An pam object returned from <code>cluster::pam()</code>
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with columns:

<code>.cluster</code>	Cluster assignment.
<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

## See Also

`augment()`, `cluster::pam()`

Other pam tidiers: `glance.pam()`, `tidy.pam()`

## Examples

```
## Not run:
library(dplyr)
library(ggplot2)
library(cluster)
library(modeldata)
data(hpc_data)

x <- hpc_data[, 2:5]
p <- pam(x, k = 4)

tidy(p)
glance(p)
augment(p, x)

augment(p, x) %>%
  ggplot(aes(compounds, input_fields)) +
  geom_point(aes(color = .cluster)) +
  geom_text(aes(label = cluster), data = tidy(p), size = 10)

## End(Not run)
```

augment.plm

*Augment data with information from a(n) plm object***Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

**Usage**

```
## S3 method for class 'plm'
augment(x, data = model.frame(x), ...)
```

**Arguments**

<code>x</code>	A <code>plm</code> object returned by <code>plm::plm()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

### See Also

`augment()`, `plm::plm()`

Other plm tidiers: `glance.plm()`, `tidy.plm()`

### Examples

```
library(plm)

data("Produc", package = "plm")
zz <- plm(log(gsp) ~ log(pcap) + log(pc) + log(emp) + unemp,
  data = Produc, index = c("state", "year")
)

summary(zz)

tidy(zz)
tidy(zz, conf.int = TRUE)
tidy(zz, conf.int = TRUE, conf.level = 0.9)

augment(zz)
glance(zz)
```

---

augment.poLCA

*Augment data with information from a(n) poLCA object*

---

### Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.



Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures) that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'poLCA'
augment(x, data = NULL, ...)
```

## Arguments

<code>x</code>	A <code>poLCA</code> object returned from <code>poLCA::poLCA()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. <code>augment</code> will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

If the `data` argument is given, those columns are included in the output (only rows for which predictions could be made). Otherwise, the `y` element of the `poLCA` object, which contains the manifest variables used to fit the model, are used, along with any covariates, if present, in `x`.

Note that while the probability of all the classes (not just the predicted modal class) can be found in the posterior element, these are not included in the augmented output.

### Value

A `tibble::tibble()` with columns:

<code>.class</code>	Predicted class.
<code>.probability</code>	Class probability of modal class.

### See Also

[augment\(\)](#), [poLCA::poLCA\(\)](#)

Other poLCA tidiers: [glance.poLCA\(\)](#), [tidy.poLCA\(\)](#)

### Examples

```
library(poLCA)
library(dplyr)

data(values)
f <- cbind(A, B, C, D) ~ 1
M1 <- poLCA(f, values, nclass = 2, verbose = FALSE)

M1
tidy(M1)
augment(M1)
glance(M1)

library(ggplot2)

ggplot(tidy(M1), aes(factor(class), estimate, fill = factor(outcome))) +
  geom_bar(stat = "identity", width = 1) +
  facet_wrap(~variable)
## Three-class model with a single covariate.

data(election)
f2a <- cbind(
  MORALG, CARESG, KNOWG, LEADG, DISHONG, INTELG,
  MORALB, CARESB, KNOWB, LEADB, DISHONB, INTELB
) ~ PARTY
nes2a <- poLCA(f2a, election, nclass = 3, nrep = 5, verbose = FALSE)

td <- tidy(nes2a)
td

# show

ggplot(td, aes(outcome, estimate, color = factor(class), group = class)) +
  geom_line() +
  facet_wrap(~variable, nrow = 2) +
```

```

  theme(axis.text.x = element_text(angle = 90, hjust = 1))

au <- augment(nes2a)
au
count(au, .class)

# if the original data is provided, it leads to NAs in new columns
# for rows that weren't predicted
au2 <- augment(nes2a, data = election)
au2
dim(au2)

```

---

augment.polr

*Augment data with information from a(n) polr object*


---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```

## S3 method for class 'polr'
augment(

```

```

x,
data = model.frame(x),
newdata = NULL,
type.predict = c("class"),
...
)

```

### Arguments

x	A polr object returned from <code>MASS::polr()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
type.predict	Which type of prediction to compute, passed to <code>MASS::predict.polr()</code> . Only supports "class" at the moment.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

### See Also

`tidy()`, `MASS::polr()`

Other ordinal tidiers: `augment.clm()`, `glance.clmm()`, `glance.clm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clmm()`, `tidy.clm()`, `tidy.polr()`, `tidy.svyolr()`

### Examples

```

library(MASS)

fit <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)

tidy(fit, exponentiate = TRUE, conf.int = TRUE)

glance(fit)
augment(fit, type.predict = "class")

fit2 <- polr(factor(gear) ~ am + mpg + qsec, data = mtcars)
tidy(fit, p.values = TRUE)

```

augment.prcomp

*Augment data with information from a(n) prcomp object*

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'prcomp'
augment(x, data = NULL, newdata, ...)
```

## Arguments

`x` A `prcomp` object returned by `stats::prcomp()`.

`data` A `base::data.frame` or `tibble::tibble()` containing the original data that was used to produce the object `x`. Defaults to `stats::model.frame(x)` so that `augment(my_fit)` returns the augmented original data. **Do not** pass new data to the `data` argument. Augment will report information such as influence and cooks distance for data passed to the `data` argument. These measures are only defined for the original training data.

`newdata` A `base::data.frame()` or `tibble::tibble()` containing all the original predictors used to create `x`. Defaults to `NULL`, indicating that nothing has been passed to `newdata`. If `newdata` is specified, the `data` argument will be ignored.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble` containing the original data along with additional columns containing each observation's projection into PCA space.

### See Also

`stats::prcomp()`, `svd_tidiers`

Other svd tidiers: `tidy.prcomp()`, `tidy_irlba()`, `tidy_svd()`

---

augment.rlm

*Augment data with information from a(n) rlm object*

---

### Description

`Augment` accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model

formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'rlm'
augment(x, data = model.frame(x), newdata = NULL, se_fit = FALSE, ...)
```

## Arguments

<code>x</code>	An <code>rlm</code> object returned by <code>MASS::rlm()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>newdata</code>	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
<code>se_fit</code>	Logical indicating whether or not a <code>.se.fit</code> column should be added to the augmented output. For some models, this calculation can be somewhat time-consuming. Defaults to <code>FALSE</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.hat</code>	Diagonal of the hat matrix.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.
<code>.sigma</code>	Estimated residual standard deviation when corresponding observation is dropped from model.

**See Also**[MASS::rlm\(\)](#)Other rlm tidiers: [glance.rlm\(\)](#), [tidy.rlm\(\)](#)**Examples**

```
library(MASS)

r <- rlm(stack.loss ~ ., stackloss)

tidy(r)
augment(r)
glance(r)
```

---

`augment.rma`*Augment data with information from a(n) rma object*

---

**Description**

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.



**Usage**

```
## S3 method for class 'rma'
augment(x, interval = c("prediction", "confidence"), ...)
```

**Arguments**

x	An rma object such as those created by <code>metafor::rma()</code> , <code>metafor::rma.uni()</code> , <code>metafor::rma.glmm()</code> , <code>metafor::rma.mh()</code> , <code>metafor::rma.mv()</code> , or <code>metafor::rma.peto()</code> .
interval	For <code>rma.mv</code> models, should prediction intervals ("prediction", default) or confidence intervals ("confidence") intervals be returned? For <code>rma.uni</code> models, prediction intervals are always returned. For <code>rma.mh</code> and <code>rma.peto</code> models, confidence intervals are always returned.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.lower</code>	Lower bound on interval for fitted values.
<code>.moderator</code>	In meta-analysis, the moderators used to calculate the predicted values.
<code>.moderator.level</code>	In meta-analysis, the level of the moderators used to calculate the predicted values.
<code>.resid</code>	The difference between observed and fitted values.
<code>.se.fit</code>	Standard errors of fitted values.
<code>.upper</code>	Upper bound on interval for fitted values.
<code>.observed</code>	The observed values for the individual studies

**Examples**

```
library(metafor)

df <-
  escalc(
    measure = "RR",
    ai = tpos,
    bi = tneg,
    ci = cpos,
    di = cneg,
```

```

      data = dat.bcg
    )

meta_analysis <- rma(yi, vi, data = df, method = "EB")

augment(meta_analysis)

```

---

 augment.rq

---

 Augment data with information from a(n) rq object
 

---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default data arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```

## S3 method for class 'rq'
augment(x, data = model.frame(x), newdata = NULL, ...)

```

**Arguments**

x	An rq object returned from <code>quantreg::rq()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
...	Arguments passed on to <code>quantreg::predict.rq</code>
object	object of class rq or rqs or rq.process produced by rq
interval	type of interval desired: default is 'none', when set to 'confidence' the function returns a matrix predictions with point predictions for each of the 'newdata' points as well as lower and upper confidence limits.
level	convergence probability for the 'confidence' intervals.
type	For <code>predict.rq</code> , the method for 'confidence' intervals, if desired. If 'percentile' then one of the bootstrap methods is used to generate percentile intervals for each prediction, if 'direct' then a version of the Portnoy and Zhou (1998) method is used, and otherwise an estimated covariance matrix for the parameter estimates is used. Further arguments to determine the choice of bootstrap method or covariance matrix estimate can be passed via the ... argument. For <code>predict.rqs</code> and <code>predict.rq.process</code> when <code>stepfun = TRUE</code> , type is "Qhat", "Fhat" or "fhat" depending on whether the user would like to have estimates of the conditional quantile, distribution or density functions respectively. As noted below the two former estimates can be monotonized with the function <code>rearrange</code> . When the "fhat" option is invoked, a list of conditional density functions is returned based on Silverman's adaptive kernel method as implemented in <code>akj</code> and <code>approxfun</code> .
na.action	function determining what should be done with missing values in 'newdata'. The default is to predict 'NA'.

**Details**

Depending on the arguments passed on to `predict.rq` via ..., a confidence interval is also calculated on the fitted values resulting in columns `.lower` and `.upper`. Does not provide confidence intervals when data is specified via the `newdata` argument.

**Value**

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.
<code>.tau</code>	Quantile.

**See Also**

`augment`, `quantreg::rq()`, `quantreg::predict.rq()`

Other quantreg tidiers: `augment.nlrq()`, `augment.rqs()`, `glance.nlrq()`, `glance.rq()`, `tidy.nlrq()`, `tidy.rqs()`, `tidy.rq()`

---

`augment.rqs`

*Augment data with information from a(n) rqs object*

---

**Description**

`Augment` accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

**Usage**

```
## S3 method for class 'rqs'
augment(x, data = model.frame(x), newdata, ...)
```

**Arguments**

x	An rqs object returned from <code>quantreg::rq()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
...	Arguments passed on to <code>quantreg::predict.rq</code>
object	object of class rq or rqs or rq.process produced by rq
interval	type of interval desired: default is 'none', when set to 'confidence' the function returns a matrix predictions with point predictions for each of the 'newdata' points as well as lower and upper confidence limits.
level	convergence probability for the 'confidence' intervals.
type	For <code>predict.rq</code> , the method for 'confidence' intervals, if desired. If 'percentile' then one of the bootstrap methods is used to generate percentile intervals for each prediction, if 'direct' then a version of the Portnoy and Zhou (1998) method is used, and otherwise an estimated covariance matrix for the parameter estimates is used. Further arguments to determine the choice of bootstrap method or covariance matrix estimate can be passed via the ...argument. For <code>predict.rqs</code> and <code>predict.rq.process</code> when <code>stepfun = TRUE</code> , type is "Qhat", "Fhat" or "fhat" depending on whether the user would like to have estimates of the conditional quantile, distribution or density functions respectively. As noted below the two former estimates can be monotonized with the function <code>rearrange</code> . When the "fhat" option is invoked, a list of conditional density functions is returned based on Silverman's adaptive kernel method as implemented in <code>akj</code> and <code>approxfun</code> .
na.action	function determining what should be done with missing values in 'newdata'. The default is to predict 'NA'.

**Details**

Depending on the arguments passed on to `predict.rq` via ..., a confidence interval is also calculated on the fitted values resulting in columns `.lower` and `.upper`. Does not provide confidence intervals when data is specified via the `newdata` argument.

**See Also**

`augment`, `quantreg::rq()`, `quantreg::predict.rqs()`

Other quantreg tidiers: `augment.nlrq()`, `augment.rq()`, `glance.nlrq()`, `glance.rq()`, `tidy.nlrq()`, `tidy.rqs()`, `tidy.rq()`

augment.sarlm

*Augment data with information from a(n) spatialreg object*

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'sarlm'
augment(x, data = x$X, ...)
```

## Arguments

<code>x</code>	An object of object returned from <code>spatialreg::lagsarlm()</code> or <code>spatialreg::errorsarlm()</code> .
<code>data</code>	Ignored, but included for internal consistency. See the details below.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Details**

The predict method for sarlm objects assumes that the response is known. See ?predict.sarlm for more discussion. As a result, since the original data can be recovered from the fit object, this method currently does not take in data or newdata arguments.

**Value**

A `tibble::tibble()` with columns:

<code>.fitted</code>	Fitted or predicted value.
<code>.resid</code>	The difference between observed and fitted values.

**See Also**

[augment\(\)](#)

Other spatialreg tidiers: [glance.sarlm\(\)](#), [tidy.sarlm\(\)](#)

**Examples**

```
## Not run:
library(spatialreg)
data(oldcol, package="spdep")
listw <- spdep::nb2listw(COL.nb, style="W")

crime_sar <- lagsarlm(CRIME ~ INC + HOVAL, data=COL.OLD,
  listw=listw, method="eigen")

tidy(crime_sar)
tidy(crime_sar, conf.int = TRUE)
glance(crime_sar)
augment(crime_sar)

crime_sem <- errorsarlm(CRIME ~ INC + HOVAL, data=COL.OLD, listw)

tidy(crime_sem)
tidy(crime_sem, conf.int = TRUE)
glance(crime_sem)
augment(crime_sem)

crime_sac <- sacsarlm(CRIME ~ INC + HOVAL, data=COL.OLD, listw)

tidy(crime_sac)
tidy(crime_sac, conf.int = TRUE)
glance(crime_sac)
augment(crime_sac)

## End(Not run)
```

---

augment.smooth.spline *Tidy a(n) smooth.spline object*

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'smooth.spline'
augment(x, data = x$data, ...)
```

## Arguments

x	A smooth.spline object returned from <code>stats::smooth.spline()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

.fitted	Fitted or predicted value.
.resid	The difference between observed and fitted values.

## See Also

`augment()`, `stats::smooth.spline()`, `stats::predict.smooth.spline()`

Other smoothing spline tidiers: `glance.smooth.spline()`



## Examples

```
spl <- smooth.spline(mtcars$wt, mtcars$mpg, df = 4)
augment(spl, mtcars)
augment(spl) # calls original columns x and y

library(ggplot2)
ggplot(augment(spl, mtcars), aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted))
```

---

augment.speedlm	<i>Augment data with information from a(n) speedlm object</i>
-----------------	---

---

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures that is not meaningfully defined for new observations).

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'speedlm'
augment(x, data = model.frame(x), newdata = NULL, ...)
```

**Arguments**

x	A speedlm object returned from <code>speedglm::speedlm()</code> .
data	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object x. Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the data argument. Augment will report information such as influence and cooks distance for data passed to the data argument. These measures are only defined for the original training data.
newdata	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create x. Defaults to NULL, indicating that nothing has been passed to newdata. If newdata is specified, the data argument will be ignored.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

.fitted	Fitted or predicted value.
.resid	The difference between observed and fitted values.

**See Also**

`speedglm::speedlm()`

Other speedlm tidiers: `glance.speedglm()`, `glance.speedlm()`, `tidy.speedglm()`, `tidy.speedlm()`

**Examples**

```
mod <- speedglm::speedlm(mpg ~ wt + qsec, data = mtcars, fitted = TRUE)

tidy(mod)
glance(mod)
augment(mod)
```

augment.stl

*Augment data with information from a(n) stl object*

## Description

Augment accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to augment via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

Augment will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related measures) that is not meaningfully defined for new observations.

For convenience, many augment methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, augment tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and `survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'stl'
augment(x, data = NULL, weights = TRUE, ...)
```

## Arguments

<code>x</code>	An <code>stl</code> object returned from <code>stats::stl()</code> .
<code>data</code>	Ignored, included for consistency with the <code>augment</code> generic signature only.
<code>weights</code>	Logical indicating whether or not to include the robust weights in the output.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed

using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble` with one row for each observation in the original times series:

<code>.seasonal</code>	The seasonal component of the decomposition.
<code>.trend</code>	The trend component of the decomposition.
<code>.remainder</code>	The remainder, or "random" component of the decomposition.
<code>.weight</code>	The final robust weights, if requested.
<code>.seasadj</code>	The seasonally adjusted (or "deseasonalised") series.

### See Also

`augment()`, `stats::stl()`

Other decompose tidiers: `augment.decomposed.ts()`

---

`augment.survreg`

*Augment data with information from a(n) survreg object*

---

### Description

`Augment` accepts a model object and a dataset and adds information about each observation in the dataset. Most commonly, this includes predicted values in the `.fitted` column, residuals in the `.resid` column, and standard errors for the fitted values in a `.se.fit` column. New columns always begin with a `.` prefix to avoid overwriting columns in the original dataset.

Users may pass data to `augment` via either the `data` argument or the `newdata` argument. If the user passes data to the `data` argument, it **must** be exactly the data that was used to fit the model object. Pass datasets to `newdata` to augment data that was not used during model fitting. This still requires that at least all predictor variable columns used to fit the model are present. If the original outcome variable used to fit the model is not included in `newdata`, then no `.resid` column will be included in the output.

`Augment` will often behave differently depending on whether `data` or `newdata` is given. This is because there is often information associated with training observations (such as influences or related) measures that is not meaningfully defined for new observations.

For convenience, many `augment` methods provide default `data` arguments, so that `augment(fit)` will return the augmented training data. In these cases, `augment` tries to reconstruct the original data based on the model object with varying degrees of success.

The augmented dataset is always returned as a `tibble::tibble` with the **same number of rows** as the passed dataset. This means that the passed data must be coercible to a tibble. At this time, tibbles do not support matrix-columns. This means you should not specify a matrix of covariates in a model formula during the original model fitting process, and that `splines::ns()`, `stats::poly()` and

`survival::Surv()` objects are not supported in input data. If you encounter errors, try explicitly passing a tibble, or fitting the original model on data in a tibble.

We are in the process of defining behaviors for models fit with various `na.action` arguments, but make no guarantees about behavior when data is missing at this time.

## Usage

```
## S3 method for class 'survreg'
augment(
  x,
  data = NULL,
  newdata = NULL,
  type.predict = "response",
  type.residuals = "response",
  ...
)
```

## Arguments

<code>x</code>	An <code>survreg</code> object returned from <code>survival::survreg()</code> .
<code>data</code>	A <code>base::data.frame</code> or <code>tibble::tibble()</code> containing the original data that was used to produce the object <code>x</code> . Defaults to <code>stats::model.frame(x)</code> so that <code>augment(my_fit)</code> returns the augmented original data. <b>Do not</b> pass new data to the <code>data</code> argument. Augment will report information such as influence and cooks distance for data passed to the <code>data</code> argument. These measures are only defined for the original training data.
<code>newdata</code>	A <code>base::data.frame()</code> or <code>tibble::tibble()</code> containing all the original predictors used to create <code>x</code> . Defaults to <code>NULL</code> , indicating that nothing has been passed to <code>newdata</code> . If <code>newdata</code> is specified, the <code>data</code> argument will be ignored.
<code>type.predict</code>	Character indicating type of prediction to use. Passed to the <code>type</code> argument of the <code>stats::predict()</code> generic. Allowed arguments vary with model class, so be sure to read the <code>predict.my_class</code> documentation.
<code>type.residuals</code>	Character indicating type of residuals to use. Passed to the <code>type</code> argument of <code>stats::residuals()</code> generic. Allowed arguments vary with model class, so be sure to read the <code>residuals.my_class</code> documentation.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with columns:

`.fitted` Fitted or predicted value.

.resid            The difference between observed and fitted values.  
 .se.fit          Standard errors of fitted values.

### See Also

[augment\(\)](#), [survival::survreg\(\)](#)

Other survreg tidiers: [glance.survreg\(\)](#), [tidy.survreg\(\)](#)

Other survival tidiers: [augment.coxph\(\)](#), [glance.aareg\(\)](#), [glance.cch\(\)](#), [glance.coxph\(\)](#),  
[glance.pyyears\(\)](#), [glance.survdiff\(\)](#), [glance.survexp\(\)](#), [glance.survfit\(\)](#), [glance.survreg\(\)](#),  
[tidy.aareg\(\)](#), [tidy.cch\(\)](#), [tidy.coxph\(\)](#), [tidy.pyyears\(\)](#), [tidy.survdiff\(\)](#), [tidy.survexp\(\)](#),  
[tidy.survfit\(\)](#), [tidy.survreg\(\)](#)

### Examples

```
library(survival)

sr <- survreg(
  Surv(futime, fustat) ~ ecog.ps + rx,
  ovarian,
  dist = "exponential"
)

tidy(sr)
augment(sr, ovarian)
glance(sr)

# coefficient plot
td <- tidy(sr, conf.int = TRUE)
library(ggplot2)
ggplot(td, aes(estimate, term)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high), height = 0) +
  geom_vline(xintercept = 0)
```

---

augment_columns	<i>Add fitted values, residuals, and other common outputs to an augment call</i>
-----------------	--

---

### Description

Add fitted values, residuals, and other common outputs to the value returned from `augment`.

### Usage

```
augment_columns(
  x,
  data,
```

```

  newdata = NULL,
  type,
  type.predict = type,
  type.residuals = type,
  se.fit = TRUE,
  ...
)

```

### Arguments

x	a model
data	original data onto which columns should be added
newdata	new data to predict on, optional
type	Type of prediction and residuals to compute
type.predict	Type of prediction to compute; by default same as type
type.residuals	Type of residuals to compute; by default same as type
se.fit	Value to pass to predict's se.fit, or NULL for no value
...	extra arguments (not used)

### Details

In the case that a residuals or influence generic is not implemented for the model, fail quietly.

---

bootstrap	<i>Set up bootstrap replicates of a dplyr operation</i>
-----------	---

---

### Description

The `bootstrap()` function is deprecated and will be removed from an upcoming release of broom. For tidy resampling, please use the `rsample` package instead. Functionality is no longer supported for this method.

### Usage

```
bootstrap(df, m, by_group = FALSE)
```

### Arguments

df	a data frame
m	number of bootstrap replicates to perform
by_group	If TRUE, then bootstrap within each group if df is a grouped tibble.

### Details

This code originates from Hadley Wickham (with a few small corrections) here: <https://github.com/tidyverse/dplyr/issues/269>

**See Also**

Other deprecated: [confint\\_tidy\(\)](#), [data.frame\\_tidiers](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#), [tidy.numeric\(\)](#)

---

`confint_tidy`*(Deprecated) Calculate confidence interval as a tidy data frame*

---

**Description**

This function is now deprecated and will be removed from a future release of broom.

**Usage**

```
confint_tidy(x, conf.level = 0.95, func = stats::confint, ...)
```

**Arguments**

<code>x</code>	a model object for which <a href="#">confint()</a> can be calculated
<code>conf.level</code>	confidence level
<code>func</code>	A function to compute a confidence interval for <code>x</code> . Calling <code>func(x, level = conf.level, ...)</code> must return an object coercible to a tibble. This dataframe like object should have to columns corresponding the lower and upper bounds on the confidence interval.
<code>...</code>	extra arguments passed on to <code>confint</code>

**Details**

Return a confidence interval as a tidy data frame. This directly wraps the [confint\(\)](#) function, but ensures it follows broom conventions: column names of `conf.low` and `conf.high`, and no row names.

`confint_tidy`

**Value**

A tibble with two columns: `conf.low` and `conf.high`.

**See Also**

Other deprecated: [bootstrap\(\)](#), [data.frame\\_tidiers](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#), [tidy.numeric\(\)](#)



---

data.frame\_tidiers      *Tidiers for data.frame objects*

---

## Description

Data frame tidiers are deprecated and will be removed from an upcoming release of broom.

## Usage

```
## S3 method for class 'data.frame'
tidy(x, ..., na.rm = TRUE, trim = 0.1)

## S3 method for class 'data.frame'
augment(x, data, ...)

## S3 method for class 'data.frame'
glance(x, ...)
```

## Arguments

x	A data.frame
...	Additional arguments for other methods.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.
trim	the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Passed to the trim argument of <a href="#">mean</a>
data	data, not used

## Details

These perform tidy summaries of data.frame objects. tidy produces summary statistics about each column, while glance simply reports the number of rows and columns. Note that augment.data.frame will throw an error.

## Value

tidy.data.frame produces a data frame with one row per original column, containing summary statistics of each:

column	name of original column
n	Number of valid (non-NA) values
mean	mean
sd	standard deviation
median	median
trimmed	trimmed mean, with trim defaulting to .1

mad	median absolute deviation (from the median)
min	minimum value
max	maximum value
range	range
skew	skew
kurtosis	kurtosis
se	standard error
glance returns a one-row data.frame with	
nrow	number of rows
ncol	number of columns
complete.obs	number of rows that have no missing values
na.fraction	fraction of values across all rows and columns that are missing

**Author(s)**

David Robinson, Benjamin Nutter

**Source**

Skew and Kurtosis functions are adapted from implementations in the moments package: Lukasz Komsta and Frederick Novomestky (2015). moments: Moments, cumulants, skewness, kurtosis and related tests. R package version 0.14.  
<https://CRAN.R-project.org/package=moments>

**See Also**

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#), [tidy.numeric\(\)](#)

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#), [tidy.numeric\(\)](#)

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#), [tidy.numeric\(\)](#)

**Examples**

```
td <- tidy(mtcars)
td

glance(mtcars)

library(ggplot2)
# compare mean and standard deviation
ggplot(td, aes(mean, sd)) + geom_point() +
  geom_text(aes(label = column), hjust = 1, vjust = 1) +
  scale_x_log10() + scale_y_log10() + geom_abline()
```

---

 durbinWatsonTest\_tidiers

*Tidy/glance a(n) durbinWatsonTest object*


---

## Description

For models that have only a single component, the `tidy()` and `glance()` methods are identical. Please see the documentation for both of those methods.

## Usage

```
## S3 method for class 'durbinWatsonTest'
tidy(x, ...)

## S3 method for class 'durbinWatsonTest'
glance(x, ...)
```

## Arguments

<code>x</code>	An object of class <code>durbinWatsonTest</code> created by a call to <code>car::durbinWatsonTest()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with columns:

<code>alternative</code>	Alternative hypothesis (character).
<code>autocorrelation</code>	Autocorrelation.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	Test statistic for Durbin-Watson test.
<code>method</code>	Always ‘Durbin-Watson Test’.

## See Also

`tidy()`, `glance()`, `car::durbinWatsonTest()`

**Examples**

```
dw <- car::durbinWatsonTest(lm(mpg ~ wt, data = mtcars))
tidy(dw)
glance(dw) # same output for all durbinWatsonTests
```

---

finish_glance	<i>(Deprecated) Add logLik, AIC, BIC, and other common measurements to a glance of a prediction</i>
---------------	---

---

**Description**

This function is now deprecated in favor of using custom logic and the appropriate `nobs()` method.

**Usage**

```
finish_glance(ret, x)
```

**Arguments**

ret	a one-row data frame (a partially complete glance)
x	the prediction model

**Value**

a one-row data frame with additional columns added, such as

logLik	log likelihoods
AIC	Akaike Information Criterion
BIC	Bayesian Information Criterion
deviance	deviance
df.residual	residual degrees of freedom

**See Also**

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [data.frame\\_tidiers](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#), [tidy.numeric\(\)](#)

---

fix_data_frame	<i>Ensure an object is a data frame, with rownames moved into a column</i>
----------------	--

---

### Description

This function is deprecated as of broom 0.7.0 and will be removed from a future release. Please see `tibble::as_tibble`.

### Usage

```
fix_data_frame(x, newnames = NULL, newcol = "term")
```

### Arguments

x	a data.frame or matrix
newnames	new column names, not including the rownames
newcol	the name of the new rownames column

### Value

a data.frame, with rownames moved into a column and new column names assigned

### See Also

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [data.frame\\_tidiers](#), [finish\\_glance\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#), [tidy.numeric\(\)](#)

---

glance.aareg	<i>Glance at a(n) aareg object</i>
--------------	------------------------------------

---

### Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'aareg'
glance(x, ...)
```

**Arguments**

`x` An aareg object returned from `survival::aareg()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>df</code>	Degrees of freedom used by the model.
<code>nobs</code>	Number of observations used.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>statistic</code>	Test statistic.

**See Also**

`glance()`, `survival::aareg()`

Other aareg tidiers: `tidy.aareg()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.cch()`, `glance.coxph()`, `glance.pyyears()`, `glance.survdiff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)

afit <- aareg(
  Surv(time, status) ~ age + sex + ph.ecog,
  data = lung,
  dfbeta = TRUE
)

tidy(afit)
```

glance.aov

*Glance at a(n) lm object***Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'aov'
glance(x, ...)
```

**Arguments**

<code>x</code>	An aov object, such as those created by <code>stats::aov()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.

**Note**

Note that `tidy.aov()` now contains the numerator and denominator degrees of freedom, which were included in the output of `glance.aov()` in some previous versions of the package.

**See Also**

[glance\(\)](#)

Other anova tidiers: [tidy.TukeyHSD\(\)](#), [tidy.anova\(\)](#), [tidy.aovlist\(\)](#), [tidy.aov\(\)](#), [tidy.manova\(\)](#)

**Examples**

```
a <- aov(mpg ~ wt + qsec + disp, mtcars)
tidy(a)
```

---

glance.Arima

*Glance at a(n) Arima object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'Arima'
glance(x, ...)
```

**Arguments**

`x` An object of class `Arima` created by `stats::arima()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.



**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
sigma	Estimated standard error of the residuals.

**See Also**

`stats::arima()`

Other Arima tidiers: `tidy.Arima()`

**Examples**

```
fit <- arima(lh, order = c(1, 0, 0))

tidy(fit)
glance(fit)
```

---

glance.betamfx

*Glance at a(n) betamfx object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'betamfx'
glance(x, ...)
```

**Arguments**

<code>x</code>	A <code>betamfx</code> object.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

This glance method wraps `glance.betareg()` for `mfx::betamfx()` objects.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>AIC</code>	Akaike's Information Criterion for the model.
<code>BIC</code>	Bayesian Information Criterion for the model.
<code>df.null</code>	Degrees of freedom used by the null model.
<code>df.residual</code>	Residual degrees of freedom.
<code>logLik</code>	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
<code>nobs</code>	Number of observations used.
<code>pseudo.r.squared</code>	Like the R squared statistic, but for situations when the R squared statistic isn't defined.

**See Also**

`glance.betareg()`, `mfx::betamfx()`

Other mfx tidiers: `augment.betamfx()`, `augment.mfx()`, `glance.mfx()`, `tidy.betamfx()`, `tidy.mfx()`

**Examples**

```
## Not run:
library(mfx)

## Simulate some data
set.seed(12345)
n = 1000
x = rnorm(n)

## Beta outcome
y = rbeta(n, shape1 = plogis(1 + 0.5 * x), shape2 = (abs(0.2*x)))
## Use Smithson and Verkuilen correction
y = (y*(n-1)+0.5)/n
```

```

d = data.frame(y,x)
mod_betamfx = betamfx(y ~ x | x, data = d)

tidy(mod_betamfx, conf.int = TRUE)

## Compare with the naive model coefficients of the equivalent betareg call (not run)
# tidy(betamfx(y ~ x | x, data = d), conf.int = TRUE)

augment(mod_betamfx)
glance(mod_betamfx)

## End(Not run)

```

---

glance.betareg	<i>Glance at a(n) betareg object</i>
----------------	--------------------------------------

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'betareg'
glance(x, ...)

```

## Arguments

x	A betareg object produced by a call to <code>betareg::betareg()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
df.null	Degrees of freedom used by the null model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
pseudo.r.squared	Like the R squared statistic, but for situations when the R squared statistic isn't defined.

**See Also**

`glance()`, `betareg::betareg()`

**Examples**

```
library(betareg)
data("GasolineYield", package = "betareg")

mod <- betareg(yield ~ batch + temp, data = GasolineYield)

mod
tidy(mod)
tidy(mod, conf.int = TRUE)
tidy(mod, conf.int = TRUE, conf.level = .99)

augment(mod)

glance(mod)
```

---

`glance.bglm`

*Glance at a(n) bglm object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

### Usage

```
## S3 method for class 'biglm'
glance(x, ...)
```

### Arguments

x	A biglm object created by a call to <code>biglm::biglm()</code> or <code>biglm::bigglm()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
deviance	Deviance of the model.
df.residual	Residual degrees of freedom.
nobs	Number of observations used.
r.squared	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.

### See Also

`glance()`, `biglm::biglm()`, `biglm::bigglm()`

Other biglm tidiers: `tidy.biglm()`

### Examples

```
## Not run:
library(biglm)

bfit <- biglm(mpg ~ wt + disp, mtcars)
tidy(bfit)
tidy(bfit, conf.int = TRUE)
tidy(bfit, conf.int = TRUE, conf.level = .9)

glance(bfit)
```

```
# bigglm: logistic regression
bgfit <- bigglm(am ~ mpg, mtcars, family = binomial())

tidy(bgfit)
tidy(bgfit, exponentiate = TRUE)
tidy(bgfit, conf.int = TRUE)
tidy(bgfit, conf.int = TRUE, conf.level = .9)
tidy(bgfit, conf.int = TRUE, conf.level = .9, exponentiate = TRUE)

glance(bgfit)

## End(Not run)
```

---

glance.binDesign      *Glance at a(n) binDesign object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'binDesign'
glance(x, ...)
```

## Arguments

x	A <code>binGroup::binDesign</code> object.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>power</code>	Power achieved by the analysis.
<code>n</code>	Sample size used to achieve this power.
<code>power.reached</code>	Whether the desired power was reached.
<code>maxit</code>	Number of iterations performed.

**See Also**

`glance()`, `binGroup::binDesign()`

Other `binGroup` tidiers: `tidy.binDesign()`, `tidy.binWidth()`

**Examples**

```
library(binGroup)
des <- binDesign(
  nmax = 300, delta = 0.06,
  p.hyp = 0.1, power = .8
)

glance(des)
tidy(des)

library(ggplot2)
ggplot(tidy(des), aes(n, power)) +
  geom_line()
```

---

`glance.cch`

*Glance at a(n) cch object*

---

**Description**

`Glance` accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

`Glance` never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

`Glance` does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

`Glance` returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'cch'
glance(x, ...)
```

**Arguments**

`x` An cch object returned from `survival::cch()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>iter</code>	Iterations of algorithm/fitting procedure completed.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>rscore</code>	Robust log-rank statistic
<code>score</code>	Score.
<code>n</code>	number of predictions
<code>nevent</code>	number of events

**See Also**

`glance()`, `survival::cch()`

Other cch tidiers: `glance.survfit()`, `tidy.cch()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.coxph()`, `glance.pyears()`, `glance.survdiff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)

# examples come from cch documentation
subcoh <- nwtco$in.subcohort
selcch <- with(nwtco, rel == 1 | subcoh == 1)
cch.data <- nwtco[selcch, ]
cch.data$subcohort <- subcoh[selcch]
## central-lab histology
cch.data$histol <- factor(cch.data$histol, labels = c("FH", "UH"))
## tumour stage
```



```

ccoh.data$stage <- factor(ccoh.data$stage, labels = c("I", "II", "III", "IV"))
ccoh.data$age <- ccoh.data$age / 12 # Age in years

fit.ccP <- cch(Surv(edrel, rel) ~ stage + histol + age,
  data = ccoh.data,
  subcoh = ~subcohort, id = ~seqno, cohort.size = 4028
)

tidy(fit.ccP)

# coefficient plot
library(ggplot2)
ggplot(tidy(fit.ccP), aes(x = estimate, y = term)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high), height = 0) +
  geom_vline(xintercept = 0)

```

glance.clm

*Glance at a(n) clm object*

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'clm'
glance(x, ...)

```

## Arguments

x	A <code>clm</code> object returned from <code>ordinal::clm()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
df.residual	Residual degrees of freedom.
edf	The effective degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.

**See Also**

`tidy`, `ordinal::clm()`

Other ordinal tidiers: `augment.clm()`, `augment.polr()`, `glance.clmm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clmm()`, `tidy.clm()`, `tidy.polr()`, `tidy.svyolr()`

**Examples**

```
library(ordinal)

fit <- clm(rating ~ temp * contact, data = wine)

tidy(fit)
tidy(fit, conf.int = TRUE, conf.level = 0.9)
tidy(fit, conf.int = TRUE, conf.type = "Wald", exponentiate = TRUE)

glance(fit)
augment(fit, type.predict = "prob")
augment(fit, type.predict = "class")

fit2 <- clm(rating ~ temp, nominal = ~contact, data = wine)
tidy(fit2)
glance(fit2)
```

---

glance.clmm

*Glance at a(n) clmm object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

### Usage

```
## S3 method for class 'clmm'
glance(x, ...)
```

### Arguments

x	A <code>clmm</code> object returned from <code>ordinal::clmm()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
edf	The effective degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.

### See Also

`tidy`, `ordinal::clmm()`

Other ordinal tidiers: `augment.clm()`, `augment.polr()`, `glance.clm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clmm()`, `tidy.clm()`, `tidy.polr()`, `tidy.svyolr()`

### Examples

```
library(ordinal)

fit <- clmm(rating ~ temp + contact + (1 | judge), data = wine)

tidy(fit)
tidy(fit, conf.int = TRUE, conf.level = 0.9)
```

```

tidy(fit, conf.int = TRUE, exponentiate = TRUE)

glance(fit)

fit2 <- c1mm(rating ~ temp + (1 | judge), nominal = ~contact, data = wine)
tidy(fit2)
glance(fit2)

```

---

glance.coefstest      *Glance at a(n) coefstest object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'coefstest'
glance(x, ...)

```

## Arguments

<code>x</code>	A <code>coefstest</code> object returned from <code>lmtest::coefstest()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
----------------------------	--

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.
df	Degrees of freedom used by the model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [stats::logLik()] may be a useful reference.
nobs	Number of observations used.
p.value	P-value corresponding to the test statistic.
r.squared	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
sigma	Estimated standard error of the residuals.
statistic	Test statistic.

### Note

Because of the way that `lmtest::coefest()` retains information about the underlying model object, the returned columns for `glance.coefest()` will vary depending on the arguments. Specifically, four columns are returned regardless: "Loglik", "AIC", "BIC", and "nobs". Users can obtain additional columns (e.g. "r.squared", "df") by invoking the "save = TRUE" argument as part of `lmtest::coefest()`. See examples.

As an aside, goodness-of-fit measures such as R-squared are unaffected by the presence of heteroskedasticity. For further discussion see, e.g. chapter 8.1 of Wooldridge (2016).

### References

Wooldridge, Jeffrey M. (2016) *Introductory econometrics: A modern approach*. (6th edition). Nelson Education.

### See Also

[glance\(\)](#), [lmtest::coefest\(\)](#)

### Examples

```
library(lmtest)

m <- lm(dist ~ speed, data = cars)

coefest(m)
tidy(coefest(m))
tidy(coefest(m, conf.int = TRUE))

# A very common workflow is to combine lmtest::coefest with alternate
# variance-covariance matrices via the sandwich package. The lmtest
# tidiers support this workflow too, enabling you to adjust the standard
# errors of your tidied models on the fly.
```

```

library(sandwich)
tidy(coeftest(m, vcov = vcovHC))           # "HC3" (default) robust SEs
tidy(coeftest(m, vcov = vcovHC, type = "HC2")) # "HC2" robust SEs
tidy(coeftest(m, vcov = NeweyWest))       # N-W HAC robust SEs

# The columns of the returned tibble for glance.coeftest() will vary
# depending on whether the coeftest object retains the underlying model.
# Users can control this with the "save = TRUE" argument of coeftest().
glance(coeftest(m))
glance(coeftest(m, save = TRUE)) # More columns

```

---

glance.coxph

*Glance at a(n) coxph object*


---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'coxph'
glance(x, ...)

```

## Arguments

x	A coxph object returned from <code>survival::coxph()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
n	The total number of observations.
nevent	Number of events.
nobs	Number of observations used.

See `survival::coxph.object` for additional column descriptions.

**See Also**

[glance\(\)](#), [survival::coxph\(\)](#)

Other coxph tidiers: [augment.coxph\(\)](#), [tidy.coxph\(\)](#)

Other survival tidiers: [augment.coxph\(\)](#), [augment.survreg\(\)](#), [glance.aareg\(\)](#), [glance.cch\(\)](#), [glance.pyears\(\)](#), [glance.survdifff\(\)](#), [glance.survexp\(\)](#), [glance.survfit\(\)](#), [glance.survreg\(\)](#), [tidy.aareg\(\)](#), [tidy.cch\(\)](#), [tidy.coxph\(\)](#), [tidy.pyears\(\)](#), [tidy.survdifff\(\)](#), [tidy.survexp\(\)](#), [tidy.survfit\(\)](#), [tidy.survreg\(\)](#)

**Examples**

```
library(survival)

cfit <- coxph(Surv(time, status) ~ age + sex, lung)

tidy(cfit)
tidy(cfit, exponentiate = TRUE)

lp <- augment(cfit, lung)
risks <- augment(cfit, lung, type.predict = "risk")
expected <- augment(cfit, lung, type.predict = "expected")

glance(cfit)

# also works on clogit models
resp <- levels(logan$occupation)
n <- nrow(logan)
indx <- rep(1:n, length(resp))
logan2 <- data.frame(
  logan[indx, ],
  id = indx,
  tocc = factor(rep(resp, each = n))
)

logan2$case <- (logan2$occupation == logan2$tocc)
```

```

cl <- clogit(case ~ tocc + tocc:education + strata(id), logan2)
tidy(cl)
glance(cl)

library(ggplot2)

ggplot(lp, aes(age, .fitted, color = sex)) +
  geom_point()

ggplot(risks, aes(age, .fitted, color = sex)) +
  geom_point()

ggplot(expected, aes(time, .fitted, color = sex)) +
  geom_point()

```

---

glance.cv.glmnet

*Glance at a(n) cv.glmnet object*


---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'cv.glmnet'
glance(x, ...)

```

## Arguments

<code>x</code>	A <code>cv.glmnet</code> object returned from <code>glmnet::cv.glmnet()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.



**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>lambda.1se</code>	The value of the penalization parameter <code>lambda</code> that results in the sparsest model while remaining within one standard error of the minimum loss.
<code>lambda.min</code>	The value of the penalization parameter <code>lambda</code> that achieved minimum loss as estimated by cross validation.
<code>nobs</code>	Number of observations used.

**See Also**

`glance()`, `glmnet::cv.glmnet()`

Other `glmnet` tidiers: `glance.glmnet()`, `tidy.cv.glmnet()`, `tidy.glmnet()`

**Examples**

```
library(glmnet)
set.seed(27)

nobs <- 100
nvar <- 50
real <- 5

x <- matrix(rnorm(nobs * nvar), nobs, nvar)
beta <- c(rnorm(real, 0, 1), rep(0, nvar - real))
y <- c(t(beta) %*% t(x)) + rnorm(nvar, sd = 3)

cvfit1 <- cv.glmnet(x, y)

tidy(cvfit1)
glance(cvfit1)

library(ggplot2)
tidied_cv <- tidy(cvfit1)
glance_cv <- glance(cvfit1)

# plot of MSE as a function of lambda
g <- ggplot(tidied_cv, aes(lambda, estimate)) +
  geom_line() +
  scale_x_log10()
g

# plot of MSE as a function of lambda with confidence ribbon
g <- g + geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .25)
g

# plot of MSE as a function of lambda with confidence ribbon and choices
# of minimum lambda marked
g <- g +
  geom_vline(xintercept = glance_cv$lambda.min) +
```

```

  geom_vline(xintercept = glance_cv$lambda.1se, lty = 2)
g

# plot of number of zeros for each choice of lambda
ggplot(tidied_cv, aes(lambda, nzero)) +
  geom_line() +
  scale_x_log10()

# coefficient plot with min lambda shown
tidied <- tidy(cvfit1$glmnet.fit)

ggplot(tidied, aes(lambda, estimate, group = term)) +
  scale_x_log10() +
  geom_line() +
  geom_vline(xintercept = glance_cv$lambda.min) +
  geom_vline(xintercept = glance_cv$lambda.1se, lty = 2)

```

glance.drc

*Glance at a(n) drc object*

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'drc'
glance(x, ...)
```

## Arguments

x	A drc object produced by a call to <code>drc::drm()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
AICc	AIC corrected for small samples

**See Also**

`glance()`, `drc::drm()`

Other drc tidiers: `augment.drc()`, `tidy.drc()`

**Examples**

```
library(drc)

mod <- drm(dead / total ~ conc, type,
  weights = total, data = selenium, fct = LL.2(), type = "binomial"
)

tidy(mod)
tidy(mod, conf.int = TRUE)

glance(mod)

augment(mod, selenium)
```

---

glance.ergm

*Glance at a(n) ergm object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'ergm'
glance(x, deviance = FALSE, mcmc = FALSE, ...)
```

**Arguments**

x	An ergm object returned from a call to <a href="#">ergm::ergm()</a> .
deviance	Logical indicating whether or not to report null and residual deviance for the model, as well as degrees of freedom. Defaults to FALSE.
mcmc	Logical indicating whether or not to report MCMC interval, burn-in and sample size used to estimate the model. Defaults to FALSE.
...	Additional arguments to pass to <a href="#">ergm::summary()</a> . <b>Cautionary note:</b> Mis-specified arguments may be silently ignored.

**Value**

`glance.ergm` returns a one-row tibble with the columns

independence	Whether the model assumed dyadic independence
iterations	The number of MCMLE iterations performed before convergence
logLik	If applicable, the log-likelihood associated with the model
AIC	The Akaike Information Criterion
BIC	The Bayesian Information Criterion

If `deviance = TRUE`, and if the model supports it, the tibble will also contain the columns

null.deviance	The null deviance of the model
df.null	The degrees of freedom of the null deviance
residual.deviance	The residual deviance of the model
df.residual	The degrees of freedom of the residual deviance

**See Also**

[glance\(\)](#), [ergm::ergm\(\)](#), [ergm::summary.ergm\(\)](#)

Other ergm tidiers: [tidy.ergm\(\)](#)

---

glance.factanal	<i>Glance at a(n) factanal object</i>
-----------------	---------------------------------------

---

### Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

### Usage

```
## S3 method for class 'factanal'
glance(x, ...)
```

### Arguments

x	A factanal object created by <code>stats::factanal()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

converged	Logical indicating if the model fitting procedure was succesful and converged.
df	Degrees of freedom used by the model.
method	Which method was used.
n	The total number of observations.
n.factors	The number of fitted factors.
nobs	Number of observations used.
p.value	P-value corresponding to the test statistic.
statistic	Test statistic.
total.variance	Total cumulative proportion of variance accounted for by all factors.

**See Also**

[glance\(\)](#), [stats::factanal\(\)](#)

Other factanal tidiers: [augment.factanal\(\)](#), [tidy.factanal\(\)](#)

**Examples**

```
set.seed(123)

# data
m1 <- dplyr::tibble(
  v1 = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 4, 5, 6),
  v2 = c(1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 3, 4, 3, 3, 3, 4, 6, 5),
  v3 = c(3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 4, 6),
  v4 = c(3, 3, 4, 3, 3, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 5, 6, 4),
  v5 = c(1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 6, 4, 5),
  v6 = c(1, 1, 1, 2, 1, 3, 3, 3, 4, 3, 1, 1, 1, 2, 1, 6, 5, 4)
)

# new data
m2 <- purrr::map_dfr(m1, rev)

# factor analysis objects
fit1 <- stats::factanal(m1, factors = 3, scores = "Bartlett")
fit2 <- stats::factanal(m1, factors = 3, scores = "regression")

# tidying the object
tidy(fit1)
tidy(fit2)

# augmented dataframe
augment(fit1)
augment(fit2)

# augmented dataframe (with new data)
augment(fit1, data = m2)
augment(fit2, data = m2)
```

---

glance.fitdistr

*Glance at a(n) fitdistr object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'fitdistr'
glance(x, ...)
```

## Arguments

x	A fitdistr object returned by <code>MASS::fitdistr()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.

## See Also

`tidy()`, `MASS::fitdistr()`  
 Other fitdistr tidiers: `tidy.fitdistr()`

## Examples

```
set.seed(2015)
x <- rnorm(100, 5, 2)

library(MASS)
fit <- fitdistr(x, dnorm, list(mean = 3, sd = 1))

tidy(fit)
glance(fit)
```

glance.fixest

*Glance at a(n) fixest object***Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'fixest'
glance(x, ...)
```

**Arguments**

<code>x</code>	A <code>fixest</code> object returned from any of the <code>fixest</code> estimators
<code>...</code>	Additional arguments passed to <code>summary</code> and <code>confint</code> . Important arguments are <code>se</code> and <code>cluster</code> . Other arguments are <code>dof</code> , <code>exact_dof</code> , <code>forceCovariance</code> , and <code>keepBounded</code> . See <a href="#">summary.fixest</a> .

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
<code>AIC</code>	Akaike's Information Criterion for the model.
<code>BIC</code>	Bayesian Information Criterion for the model.
<code>logLik</code>	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
<code>nobs</code>	Number of observations used.
<code>pseudo.r.squared</code>	Like the R squared statistic, but for situations when the R squared statistic isn't defined.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.



```
sigma           Estimated standard error of the residuals.
within.r.squared
                R squared within fixed-effect groups.
```

### Note

All columns listed below will be returned, but some will be NA, depending on the type of model estimated. `sigma`, `r.squared`, `adj.r.squared`, and `within.r.squared` will be NA for any model other than `feols`. `pseudo.r.squared` will be NA for `feols`.

### Examples

```
library(fixest)
gravity <- feols(log(Euros) ~ log(dist_km) | Origin + Destination + Product + Year, trade)

tidy(gravity)
glance(gravity)
augment(gravity, trade)

## To get robust or clustered SEs, users can either:
# 1) Or, specify the arguments directly in the tidy() call
tidy(gravity, conf.int = TRUE, cluster = c("Product", "Year"))
tidy(gravity, conf.int = TRUE, se = "threeway")
# 2) Feed tidy() a summary.fixest object that has already accepted these arguments
gravity_summ <- summary(gravity, cluster = c("Product", "Year"))
tidy(gravity_summ, conf.int = TRUE)
# Approach (1) is preferred.

## The other fixest methods all work similarly. For example:
gravity_pois <- feglm(Euros ~ log(dist_km) | Origin + Destination + Product + Year, trade)
tidy(gravity_pois)
glance(gravity_pois)
augment(gravity_pois, trade)
```

---

glance.gam

*Glance at a(n) gam object*


---

### Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

### Usage

```
## S3 method for class 'gam'
glance(x, ...)
```

### Arguments

x	A gam object returned from a call to <code>mgcv::gam()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.
df	Degrees of freedom used by the model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.

### See Also

`glance()`, `mgcv::gam()`  
 Other mgcv tidiers: `tidy.gam()`

### Examples

```
g <- mgcv::gam(mpg ~ s(hp) + am + qsec, data = mtcars)

tidy(g)
tidy(g, parametric = TRUE)
glance(g)
```

---

glance.garch	<i>Tidy a(n) garch object</i>
--------------	-------------------------------

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'garch'
glance(x, test = c("box-ljung-test", "jarque-bera-test"), ...)
```

### Arguments

x	A garch object returned by <code>tseries::garch()</code> .
test	Character specification of which hypothesis test to use. The garch function reports 2 hypothesis tests: Jarque-Bera to residuals and Box-Ljung to squared residuals.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
method	Which method was used.
nobs	Number of observations used.
p.value	P-value corresponding to the test statistic.
statistic	Test statistic.
parameter	Parameter field in the htest, typically degrees of freedom.

### See Also

`glance()`, `tseries::garch()`, []  
 Other garch tidiers: `tidy.garch()`

glance.geeglm

*Glance at a(n) geeglm object*

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'geeglm'
glance(x, ...)
```

## Arguments

<code>x</code>	A <code>geeglm</code> object returned from a call to <code>geepack::geeglm()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>alpha</code>	Estimated correlation parameter for <code>geepack::geeglm</code> .
<code>df.residual</code>	Residual degrees of freedom.
<code>gamma</code>	Estimated scale parameter for <code>geepack::geeglm</code> .
<code>max.cluster.size</code>	Max number of elements in clusters.
<code>n.clusters</code>	Number of clusters.

**See Also**

[glance\(\)](#), [geepack::geeglm\(\)](#)

**Examples**

```
library(geepack)
data(state)

ds <- data.frame(state.region, state.x77)

geefit <- geeglm(Income ~ Frost + Murder,
  id = state.region,
  data = ds, family = gaussian,
  corstr = "exchangeable"
)

tidy(geefit)
tidy(geefit, conf.int = TRUE)
```

---

glance.glm

*Glance at a(n) glm object*

---

**Description**

Glance accepts a model object and returns a [tibble::tibble\(\)](#) with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'glm'
glance(x, ...)
```

**Arguments**

x                    A glm object returned from [stats::glm\(\)](#).

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.
df.null	Degrees of freedom used by the null model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
null.deviance	Deviance of the null model.

## See Also

`stats::glm()`

Other lm tidiers: `augment.glm()`, `augment.lm()`, `glance.lm()`, `glance.summary.lm()`, `glance.svyglm()`, `tidy.glm()`, `tidy.lm.beta()`, `tidy.lm()`, `tidy.mlml()`, `tidy.summary.lm()`

## Examples

```
g <- glm(am ~ mpg, mtcars, family = "binomial")
glance(g)
```

---

glance.glmnet

*Glance at a(n) glmnet object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'glmnet'
glance(x, ...)
```

## Arguments

x	A <code>glmnet</code> object returned from <code>glmnet::glmnet()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

nobs	Number of observations used.
npasses	Total passes over the data across all lambda values.
nulldev	Null deviance.

## See Also

`glance()`, `glmnet::glmnet()`

Other `glmnet` tidiers: `glance.cv.glmnet()`, `tidy.cv.glmnet()`, `tidy.glmnet()`

## Examples

```
library(glmnet)

set.seed(2014)
x <- matrix(rnorm(100 * 20), 100, 20)
y <- rnorm(100)
fit1 <- glmnet(x, y)

tidy(fit1)
glance(fit1)
```

```

library(dplyr)
library(ggplot2)

tidied <- tidy(fit1) %>% filter(term != "(Intercept)")

ggplot(tidied, aes(step, estimate, group = term)) +
  geom_line()
ggplot(tidied, aes(lambda, estimate, group = term)) +
  geom_line() +
  scale_x_log10()

ggplot(tidied, aes(lambda, dev.ratio)) +
  geom_line()

# works for other types of regressions as well, such as logistic
g2 <- sample(1:2, 100, replace = TRUE)
fit2 <- glmnet(x, g2, family = "binomial")
tidy(fit2)

```

---

glance.glmRob

*Glance at a(n) glmRob object*


---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'glmRob'
glance(x, ...)

```

## Arguments

`x` A `glmRob` object returned from `robust::glmRob()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be



used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

<code>deviance</code>	Deviance of the model.
<code>df.residual</code>	Residual degrees of freedom.
<code>nobs</code>	Number of observations used.
<code>null.deviance</code>	Deviance of the null model.
<code>sigma</code>	Estimated standard error of the residuals.

### See Also

`robust::glmRob()`

Other robust tidiers: `augment.lmRob()`, `glance.lmRob()`, `tidy.glmRob()`, `tidy.lmRob()`

### Examples

```
library(robust)

gm <- glmRob(am ~ wt, data = mtcars, family = "binomial")

tidy(gm)
glance(gm)
```

---

`glance.gmm`

*Glance at a(n) gmm object*

---

### Description

`Glance` accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

`Glance` never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

`Glance` does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

`Glance` returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'gmm'
glance(x, ...)
```

**Arguments**

`x` A gmm object returned from `gmm::gmm()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>df</code>	Degrees of freedom used by the model.
<code>df.residual</code>	Residual degrees of freedom.
<code>nobs</code>	Number of observations used.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>statistic</code>	Test statistic.

**See Also**

`glance()`, `gmm::gmm()`

Other gmm tidiers: `tidy.gmm()`

**Examples**

```
library(gmm)

# examples come from the "gmm" package
## CAPM test with GMM
data(Finance)
r <- Finance[1:300, 1:10]
rm <- Finance[1:300, "rm"]
rf <- Finance[1:300, "rf"]

z <- as.matrix(r - rf)
t <- nrow(z)
zm <- rm - rf
h <- matrix(zm, t, 1)
res <- gmm(z ~ zm, x = h)

# tidy result
```

```

tidy(res)
tidy(res, conf.int = TRUE)
tidy(res, conf.int = TRUE, conf.level = .99)

# coefficient plot
library(ggplot2)
library(dplyr)

tidy(res, conf.int = TRUE) %>%
  mutate(variable = reorder(term, estimate)) %>%
  ggplot(aes(estimate, variable)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high)) +
  geom_vline(xintercept = 0, color = "red", lty = 2)

# from a function instead of a matrix
g <- function(theta, x) {
  e <- x[, 2:11] - theta[1] - (x[, 1] - theta[1]) %*% matrix(theta[2:11], 1, 10)
  gmat <- cbind(e, e * c(x[, 1]))
  return(gmat)
}

x <- as.matrix(cbind(rm, r))
res_black <- gmm(g, x = x, t0 = rep(0, 11))

tidy(res_black)
tidy(res_black, conf.int = TRUE)

## APT test with Fama-French factors and GMM

f1 <- zm
f2 <- Finance[1:300, "hml"] - rf
f3 <- Finance[1:300, "smb"] - rf
h <- cbind(f1, f2, f3)
res2 <- gmm(z ~ f1 + f2 + f3, x = h)

td2 <- tidy(res2, conf.int = TRUE)
td2

# coefficient plot
td2 %>%
  mutate(variable = reorder(term, estimate)) %>%
  ggplot(aes(estimate, variable)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high)) +
  geom_vline(xintercept = 0, color = "red", lty = 2)

```

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'ivreg'
glance(x, diagnostics = FALSE, ...)
```

## Arguments

<code>x</code>	An ivreg object created by a call to <code>AER::ivreg()</code> .
<code>diagnostics</code>	Logical indicating whether or not to return the Wu-Hausman and Sargan diagnostic information.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

This tidier currently only supports ivreg-classed objects outputted by the AER package. The ivreg package also outputs objects of class `ivreg`, and will be supported in a later release.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
<code>df</code>	Degrees of freedom used by the model.
<code>df.residual</code>	Residual degrees of freedom.
<code>nobs</code>	Number of observations used.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.

sigma	Estimated standard error of the residuals.
statistic	Wald test statistic.
p.value	P-value for the Wald test.

**Note**

Beginning 0.7.0, `glance.ivreg` returns statistics for the Wu-Hausman test for endogeneity and the Sargan test of overidentifying restrictions. Sargan test values are returned as NA if the number of instruments is not greater than the number of endogenous regressors.

**See Also**

[glance\(\)](#), [AER::ivreg\(\)](#)

Other ivreg tidiers: [augment.ivreg\(\)](#), [tidy.ivreg\(\)](#)

**Examples**

```
library(AER)

data("CigarettesSW", package = "AER")

ivr <- ivreg(
  log(packs) ~ income | population,
  data = CigarettesSW,
  subset = year == "1995"
)

summary(ivr)

tidy(ivr)
tidy(ivr, conf.int = TRUE)
tidy(ivr, conf.int = TRUE, instruments = TRUE)

augment(ivr)
augment(ivr, data = CigarettesSW)
augment(ivr, newdata = CigarettesSW)

glance(ivr)
```

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'kmeans'
glance(x, ...)
```

## Arguments

<code>x</code>	A kmeans object created by <code>stats::kmeans()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>betweenss</code>	The total between-cluster sum of squares.
<code>iter</code>	Iterations of algorithm/fitting procedure completed.
<code>tot.withinss</code>	The total within-cluster sum of squares.
<code>totss</code>	The total sum of squares.

## See Also

`glance()`, `stats::kmeans()`

Other kmeans tidiers: `augment.kmeans()`, `tidy.kmeans()`

## Examples

```
## Not run:
library(cluster)
library(dplyr)

library(modeldata)
data(hpc_data)

x <- hpc_data[, 2:5]

fit <- pam(x, k = 4)

tidy(fit)
glance(fit)
augment(fit, x)

## End(Not run)
```

---

glance.lavaan

*Glance at a(n) lavaan object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'lavaan'
glance(x, ...)
```

## Arguments

x A lavaan object, such as those returned from `lavaan::cfa()`, and `lavaan::sem()`.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A one-row `tibble::tibble` with columns:

<code>chisq</code>	Model chi squared
<code>npar</code>	Number of parameters in the model
<code>rmsea</code>	Root mean square error of approximation
<code>rmsea.conf.high</code>	95 percent upper bound on RMSEA
<code>srmr</code>	Standardised root mean residual
<code>agfi</code>	Adjusted goodness of fit
<code>cfi</code>	Comparative fit index
<code>tli</code>	Tucker Lewis index
<code>AIC</code>	Akaike information criterion
<code>BIC</code>	Bayesian information criterion
<code>ngroups</code>	Number of groups in model
<code>nobs</code>	Number of observations included
<code>norig</code>	Number of observation in the original dataset
<code>nexcluded</code>	Number of excluded observations
<code>converged</code>	Logical - Did the model converge
<code>estimator</code>	Estimator used
<code>missing_method</code>	Method for eliminating missing data

For further recommendations on reporting SEM and CFA models see Schreiber, J. B. (2017). Update to core reporting practices in structural equation modeling. *Research in Social and Administrative Pharmacy*, 13(3), 634-643. <https://doi.org/10.1016/j.sapharm.2016.06.006>

## See Also

`glance()`, `lavaan::cfa()`, `lavaan::sem()`, `lavaan::fitmeasures()`

Other lavaan tidiers: `tidy.lavaan()`



**Examples**

```
## Not run:
library(lavaan)

cfa.fit <- cfa(
  "F =~ x1 + x2 + x3 + x4 + x5",
  data = HolzingerSwineford1939, group = "school"
)
glance(cfa.fit)

## End(Not run)
```

---

glance.lm	<i>Glance at a(n) lm object</i>
-----------	---------------------------------

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'lm'
glance(x, ...)
```

**Arguments**

x	An <code>lm</code> object created by <code>stats::lm()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
<code>AIC</code>	Akaike's Information Criterion for the model.
<code>BIC</code>	Bayesian Information Criterion for the model.
<code>deviance</code>	Deviance of the model.
<code>df.residual</code>	Residual degrees of freedom.
<code>logLik</code>	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
<code>nobs</code>	Number of observations used.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
<code>sigma</code>	Estimated standard error of the residuals.
<code>statistic</code>	Test statistic.
<code>df</code>	The degrees for freedom from the numerator of the overall F-statistic. This is new in broom 0.7.0. Previously, this reported the rank of the design matrix, which is one more than the numerator degrees of freedom of the overall F-statistic.

**See Also**

`glance()`, `glance.summary.lm()`

Other lm tidiers: `augment.glm()`, `augment.lm()`, `glance.glm()`, `glance.summary.lm()`, `glance.svyglm()`, `tidy.glm()`, `tidy.lm.beta()`, `tidy.lm()`, `tidy.mlrm()`, `tidy.summary.lm()`

**Examples**

```
library(ggplot2)
library(dplyr)

mod <- lm(mpg ~ wt + qsec, data = mtcars)

tidy(mod)
glance(mod)

# coefficient plot
d <- tidy(mod, conf.int = TRUE)

ggplot(d, aes(estimate, term, xmin = conf.low, xmax = conf.high, height = 0)) +
  geom_point() +
  geom_vline(xintercept = 0, lty = 4) +
  geom_errorbarh()
```

```

# Aside: There are tidy() and glance() methods for lm.summary objects too.
# This can be useful when you want to conserve memory by converting large lm
# objects into their leaner summary.lm equivalents.
s <- summary(mod)
tidy(s, conf.int = TRUE)
glance(s)

augment(mod)
augment(mod, mtcars, interval = "confidence")

# predict on new data
newdata <- mtcars %>%
  head(6) %>%
  mutate(wt = wt + 1)
augment(mod, newdata = newdata)

# ggplot2 example where we also construct 95% prediction interval
mod2 <- lm(mpg ~ wt, data = mtcars) ## simpler bivariate model since we're plotting in 2D

au <- augment(mod2, newdata = newdata, interval = "prediction")

ggplot(au, aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted)) +
  geom_ribbon(aes(ymin = .lower, ymax = .upper), col = NA, alpha = 0.3)

# predict on new data without outcome variable. Output does not include .resid
newdata <- newdata %>%
  select(-mpg)
augment(mod, newdata = newdata)

au <- augment(mod, data = mtcars)

ggplot(au, aes(.hat, .std.resid)) +
  geom_vline(size = 2, colour = "white", xintercept = 0) +
  geom_hline(size = 2, colour = "white", yintercept = 0) +
  geom_point() +
  geom_smooth(se = FALSE)

plot(mod, which = 6)
ggplot(au, aes(.hat, .cooks)) +
  geom_vline(xintercept = 0, colour = NA) +
  geom_abline(slope = seq(0, 3, by = 0.5), colour = "white") +
  geom_smooth(se = FALSE) +
  geom_point()

# column-wise models
a <- matrix(rnorm(20), nrow = 10)
b <- a + rnorm(length(a))
result <- lm(b ~ a)
tidy(result)

```

glance.lmodel2

*Glance at a(n) lmodel2 object***Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'lmodel2'
glance(x, ...)
```

**Arguments**

<code>x</code>	A <code>lmodel2</code> object returned by <code>lmodel2::lmodel2()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>nobs</code>	Number of observations used.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
<code>theta</code>	Angle between OLS lines <code>'lm(y ~ x)'</code> and <code>'lm(x ~ y)'</code>
<code>H</code>	H statistic for computing confidence interval of major axis slope

**See Also**

[glance\(\)](#), [lmodel2::lmodel2\(\)](#)

Other lmodel2 tidiers: [tidy.lmodel2\(\)](#)

**Examples**

```
library(lmodel2)

data(mod2ex2)
Ex2.res <- lmodel2(Prey ~ Predators, data = mod2ex2, "relative", "relative", 99)
Ex2.res

tidy(Ex2.res)
glance(Ex2.res)

# this allows coefficient plots with ggplot2
library(ggplot2)
ggplot(tidy(Ex2.res), aes(estimate, term, color = method)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high)) +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high))
```

---

glance.lmRob

*Glance at a(n) lmRob object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'lmRob'
glance(x, ...)
```

**Arguments**

x	A <code>lmRob</code> object returned from <code>robust::lmRob()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

deviance	Deviance of the model.
df.residual	Residual degrees of freedom.
nobs	Number of observations used.
r.squared	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
sigma	Estimated standard error of the residuals.

**See Also**

`robust::lmRob()`

Other robust tidiers: `augment.lmRob()`, `glance.glmRob()`, `tidy.glmRob()`, `tidy.lmRob()`

**Examples**

```
library(robust)
m <- lmRob(mpg ~ wt, data = mtcars)

tidy(m)
augment(m)
glance(m)
```

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'lmrob'
glance(x, ...)
```

## Arguments

<code>x</code>	A <code>lmrob</code> object returned from <code>robustbase::lmrob()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

For tidiers for robust models from the **MASS** package see `tidy.rlm()`.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>df.residual</code>	Residual degrees of freedom.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
<code>sigma</code>	Estimated standard error of the residuals.

## See Also

`robustbase::lmrob()`

Other robustbase tidiers: `augment.glmrob()`, `augment.lmrob()`, `tidy.glmrob()`, `tidy.lmrob()`

## Examples

```
library(robustbase)
# From the robustbase::lmrob examples:
data(coleman)
set.seed(0)

m <- robustbase::lmrob(Y ~ ., data = coleman)
tidy(m)
augment(m)
glance(m)

# From the robustbase::glmrob examples:
data(carrots)
Rfit <- glmrob(cbind(success, total - success) ~ logdose + block,
  family = binomial, data = carrots, method = "Mqle",
  control = glmrobMqle.control(tcc = 1.2)
)
tidy(Rfit)
augment(Rfit)
```

---

glance.margins

*Glance at a(n) margins object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'margins'
glance(x, ...)
```

## Arguments

x                    A margins object returned from `margins::margins()`.



... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
<code>df</code>	Degrees of freedom used by the model.
<code>df.residual</code>	Residual degrees of freedom.
<code>nobs</code>	Number of observations used.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
<code>sigma</code>	Estimated standard error of the residuals.
<code>statistic</code>	Test statistic.

## Examples

```
library(margins)

## Example 1: Logit model ##

mod_log <- glm(am ~ cyl + hp + wt, data = mtcars, family = binomial)

# Get tidied "naive" model coefficients
tidy(mod_log)

# Convert to marginal effects with margins::margins()
marg_log <- margins(mod_log)

# Get tidied marginal effects
tidy(marg_log)
tidy(marg_log, conf.int = TRUE)
glance(marg_log) ## Requires running the underlying model again. Quick for this example.
## Not run: augment(marg_log) ## Not supported.
augment(mod_log) ## But can get the same info by running on the underlying model.

## Example 2: Threeway interaction terms ##

mod_ie <- lm(mpg ~ wt * cyl * disp, data = mtcars)

# Get tidied "naive" model coefficients
```

```

tidy(mod_ie)

# Convert to marginal effects with margins::margins()
marg_ie0 <- margins(mod_ie)
# Get tidied marginal effects
tidy(marg_ie0)
glance(marg_ie0)

# Marginal effects evaluated at specific values of a variable (here: cyl)
marg_ie1 <- margins(mod_ie, at = list(cyl = c(4,6,8)))
tidy(marg_ie1)

# Marginal effects of one interaction variable (here: wt), modulated at
# specific values of the two other interaction variables (here: cyl and drat)
marg_ie2 <- margins(mod_ie,
                    variables = "wt", ## Main var
                    at = list(cyl = c(4,6,8), drat = c(3, 3.5, 4))) ## Modulating vars
tidy(marg_ie2)

```

---

glance.Mclust

*Glance at a(n) Mclust object*


---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'Mclust'
glance(x, ...)

```

## Arguments

`x` An Mclust object return from `mclust::Mclust()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be

used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

BIC	Bayesian Information Criterion for the model.
df	Degrees of freedom used by the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
model	A string denoting the model type with optimal BIC
G	Number mixture components in optimal model
hypvol	If the other model contains a noise component, the value of the hypervolume parameter. Otherwise 'NA'.

## Examples

```
library(dplyr)
library(mclust)
set.seed(27)

centers <- tibble::tibble(
  cluster = factor(1:3),
  num_points = c(100, 150, 50), # number points in each cluster
  x1 = c(5, 0, -3), # x1 coordinate of cluster center
  x2 = c(-1, 1, -2) # x2 coordinate of cluster center
)

points <- centers %>%
  mutate(
    x1 = purrr::map2(num_points, x1, rnorm),
    x2 = purrr::map2(num_points, x2, rnorm)
  ) %>%
  dplyr::select(-num_points, -cluster) %>%
  tidyr::unnest(c(x1, x2))

m <- mclust::Mclust(points)

tidy(m)
augment(m, points)
glance(m)
```

glance.mfx

*Glance at a(n) mfx object*

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'mfx'
glance(x, ...)

## S3 method for class 'logitmfx'
glance(x, ...)

## S3 method for class 'negbinmfx'
glance(x, ...)

## S3 method for class 'poissonmfx'
glance(x, ...)

## S3 method for class 'probitmfx'
glance(x, ...)
```

## Arguments

x	A <code>logitmfx</code> , <code>negbinmfx</code> , <code>poissonmfx</code> , or <code>probitmfx</code> object. (Note that <code>betamfx</code> objects receive their own set of tidiers.)
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

This generic glance method wraps `glance.glm()` for applicable objects from the mfx package.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.
df.null	Degrees of freedom used by the null model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
null.deviance	Deviance of the null model.

**See Also**

`glance.glm()`, `mfx::logitmfx()`, `mfx::negbinmfx()`, `mfx::poissonmfx()`, `mfx::probitmfx()`

Other mfx tidiers: `augment.betamfx()`, `augment.mfx()`, `glance.betamfx()`, `tidy.betamfx()`, `tidy.mfx()`

**Examples**

```
## Not run:
library(mfx)

## Get the marginal effects from a logit regression
mod_logmfx <- logitmfx(am ~ cyl + hp + wt, atmean = TRUE, data = mtcars)
tidy(mod_logmfx, conf.int = TRUE)

## Compare with the naive model coefficients of the same logit call (not run)
# tidy(glm(am ~ cyl + hp + wt, family = binomial, data = mtcars), conf.int = TRUE)

augment(mod_logmfx)
glance(mod_logmfx)

## Another example, this time using probit regression
mod_probmfx <- probitmfx(am ~ cyl + hp + wt, atmean = TRUE, data = mtcars)
tidy(mod_probmfx, conf.int = TRUE)
augment(mod_probmfx)
glance(mod_probmfx)

## End(Not run)
```

glance.mjoint

*Glance at a(n) mjoint object*

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'mjoint'
glance(x, ...)
```

## Arguments

<code>x</code>	An mjoint object returned from <code>joineRML::mjoint()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
logLik	The log-likelihood of the model. <code>[stats::logLik()]</code> may be a useful reference.
sigma2_j	The square root of the estimated residual variance for the j-th longitudinal process

**See Also**

[glance\(\)](#), [joineRML::mjoint\(\)](#)

Other mjoint tidiers: [tidy.mjoint\(\)](#)

**Examples**

```
## Not run:
# Fit a joint model with bivariate longitudinal outcomes
library(joineRML)
data(heart.valve)
hvd <- heart.valve[!is.na(heart.valve$log.grad) &
  !is.na(heart.valve$log.lvmi) &
  heart.valve$num <= 50, ]
fit <- mjoint(
  formLongFixed = list(
    "grad" = log.grad ~ time + sex + hs,
    "lvmi" = log.lvmi ~ time + sex
  ),
  formLongRandom = list(
    "grad" = ~ 1 | num,
    "lvmi" = ~ time | num
  ),
  formSurv = Surv(fuyrs, status) ~ age,
  data = hvd,
  inits = list("gamma" = c(0.11, 1.51, 0.80)),
  timeVar = "time"
)

# Extract the survival fixed effects
tidy(fit)

# Extract the longitudinal fixed effects
tidy(fit, component = "longitudinal")

# Extract the survival fixed effects with confidence intervals
tidy(fit, ci = TRUE)

# Extract the survival fixed effects with confidence intervals based
# on bootstrapped standard errors
bSE <- bootSE(fit, nboot = 5, safe.boot = TRUE)
tidy(fit, boot_se = bSE, ci = TRUE)

# Augment original data with fitted longitudinal values and residuals
hvd2 <- augment(fit)

# Extract model statistics
glance(fit)

## End(Not run)
```

glance.mlogit

*Glance at a(n) mlogit object***Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'mlogit'
glance(x, ...)
```

**Arguments**

`x` an object returned from `mlogit::mlogit()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
rho2	McFadden's rho squared with respect to a market shares (constants-only) model.
rho20	McFadden's rho squared with respect to an equal shares (no information) model.



**See Also**

`glance()`, `mlogit::mlogit()`

Other mlogit tidiers: `augment.mlogit()`, `tidy.mlogit()`

**Examples**

```
## Not run:
library(mlogit)
data("Fishing", package = "mlogit")
Fish <- dfidx(Fishing, varying = 2:9, shape = "wide", choice = "mode")
m <- mlogit(mode ~ price + catch | income, data = Fish)

tidy(m)
augment(m)
glance(m)

## End(Not run)
```

---

glance.muhaz

*Glance at a(n) muhaz object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'muhaz'
glance(x, ...)
```

**Arguments**

x                    A muhaz object returned by `muhaz::muhaz()`.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

<code>max.hazard</code>	Maximal estimated hazard.
<code>max.time</code>	The maximum observed event or censoring time.
<code>min.hazard</code>	Minimal estimated hazard.
<code>min.time</code>	The minimum observed event or censoring time.
<code>nobs</code>	Number of observations used.

### See Also

`glance()`, `muhaz::muhaz()`

Other muhaz tidiers: `tidy.muhaz()`

### Examples

```
library(muhaz)

data(ovarian, package = "survival")
x <- muhaz::muhaz(ovarian$futime, ovarian$fustat)
tidy(x)
glance(x)
```

---

`glance.multinom`

*Glance at a(n) multinom object*

---

### Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

### Usage

```
## S3 method for class 'multinom'
glance(x, ...)
```

### Arguments

x	A multinom object returned from <code>nnet::multinom()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
deviance	Deviance of the model.
edf	The effective degrees of freedom.
nobs	Number of observations used.

### See Also

`glance()`, `nnet::multinom()`  
 Other multinom tidiers: `tidy.multinom()`

### Examples

```
library(nnet)
library(MASS)

example(birthwt)
bwt.mu <- multinom(low ~ ., bwt)
tidy(bwt.mu)
glance(bwt.mu)

## This model is a truly terrible model
## but it should show you what the output looks
## like in a multinomial logistic regression
```

```
fit.gear <- multinom(gear ~ mpg + factor(am), data = mtcars)
tidy(fit.gear)
glance(fit.gear)
```

---

glance.nlrq

*Glance at a(n) nlrq object*


---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'nlrq'
glance(x, ...)
```

## Arguments

x	A nlrq object returned from <code>quantreg::nlrq()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
tau	Quantile.

**See Also**

[glance\(\)](#), [quantreg::nlrq\(\)](#)

Other quantreg tidiers: [augment.nlrq\(\)](#), [augment.rqs\(\)](#), [augment.rq\(\)](#), [glance.rq\(\)](#), [tidy.nlrq\(\)](#), [tidy.rqs\(\)](#), [tidy.rq\(\)](#)

---

glance.nls

*Glance at a(n) nls object*

---

**Description**

Glance accepts a model object and returns a [tibble::tibble\(\)](#) with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'nls'
glance(x, ...)
```

**Arguments**

`x` An nls object returned from [stats::nls\(\)](#).

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an [augment\(\)](#) method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A [tibble::tibble\(\)](#) with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.

<code>df.residual</code>	Residual degrees of freedom.
<code>finTol</code>	The achieved convergence tolerance.
<code>isConv</code>	Whether the fit successfully converged.
<code>logLik</code>	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
<code>nobs</code>	Number of observations used.
<code>sigma</code>	Estimated standard error of the residuals.

**See Also**

[tidy](#), [stats::nls\(\)](#)

Other nls tidiers: [augment.nls\(\)](#), [tidy.nls\(\)](#)

**Examples**

```
n <- nls(mpg ~ k * e^wt, data = mtcars, start = list(k = 1, e = 2))

tidy(n)
augment(n)
glance(n)

library(ggplot2)
ggplot(augment(n), aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted))

newdata <- head(mtcars)
newdata$wt <- newdata$wt + 1
augment(n, newdata = newdata)
```

---

glance.orcutt

*Glance at a(n) orcutt object*

---

**Description**

Glance accepts a model object and returns a [tibble::tibble\(\)](#) with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'orcutt'
glance(x, ...)
```

**Arguments**

`x` An orcutt object returned from `orcutt::cochrane.orcutt()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
<code>dw.original</code>	Durbin-Watson statistic of original fit.
<code>dw.transformed</code>	Durbin-Watson statistic of transformed fit.
<code>nobs</code>	Number of observations used.
<code>number.interaction</code>	Number of interactions.
<code>p.value.original</code>	P-value of original Durbin-Watson statistic.
<code>p.value.transformed</code>	P-value of autocorrelation after transformation.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
<code>rho</code>	Spearman's rho autocorrelation

**See Also**

`glance()`, `orcutt::cochrane.orcutt()`

Other orcutt tidiers: `tidy.orcutt()`

**Examples**

```
library(orcutt)

reg <- lm(mpg ~ wt + qsec + disp, mtcars)
tidy(reg)
```

```
co <- cochrane.orcutt(reg)
co

tidy(co)
glance(co)
```

---

glance.pam

*Glance at a(n) pam object*


---

### Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

### Usage

```
## S3 method for class 'pam'
glance(x, ...)
```

### Arguments

x	An pam object returned from <code>cluster::pam()</code>
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautious note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

avg.silhouette.width	The average silhouette width for the dataset.
----------------------	---



**See Also**

[glance\(\)](#), [cluster::pam\(\)](#)

Other pam tidiers: [augment.pam\(\)](#), [tidy.pam\(\)](#)

**Examples**

```
## Not run:
library(dplyr)
library(ggplot2)
library(cluster)
library(modeldata)
data(hpc_data)

x <- hpc_data[, 2:5]
p <- pam(x, k = 4)

tidy(p)
glance(p)
augment(p, x)

augment(p, x) %>%
  ggplot(aes(compounds, input_fields)) +
  geom_point(aes(color = .cluster)) +
  geom_text(aes(label = cluster), data = tidy(p), size = 10)

## End(Not run)
```

---

glance.plm

*Glance at a(n) plm object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'plm'
glance(x, ...)
```

**Arguments**

`x` A `plm` object returned by `plm::plm()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
<code>deviance</code>	Deviance of the model.
<code>df.residual</code>	Residual degrees of freedom.
<code>nobs</code>	Number of observations used.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
<code>statistic</code>	F-statistic

**See Also**

`glance()`, `plm::plm()`

Other `plm` tidiers: `augment.plm()`, `tidy.plm()`

**Examples**

```
library(plm)

data("Produc", package = "plm")
zz <- plm(log(gsp) ~ log(pcap) + log(pc) + log(emp) + unemp,
  data = Produc, index = c("state", "year")
)

summary(zz)

tidy(zz)
tidy(zz, conf.int = TRUE)
```

```
tidy(zz, conf.int = TRUE, conf.level = 0.9)

augment(zz)
glance(zz)
```

glance.poLCA

*Glance at a(n) poLCA object***Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'poLCA'
glance(x, ...)
```

**Arguments**

x	A poLCA object returned from <code>poLCA::poLCA()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
chi.squared	The Pearson Chi-Square goodness of fit statistic for multiway tables.
df	Degrees of freedom used by the model.

df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [stats::logLik()] may be a useful reference.
nobs	Number of observations used.
g.squared	The likelihood ratio/deviance statistic

**See Also**

[glance\(\)](#), [poLCA::poLCA\(\)](#)

Other poLCA tidiers: [augment.poLCA\(\)](#), [tidy.poLCA\(\)](#)

**Examples**

```
library(poLCA)
library(dplyr)

data(values)
f <- cbind(A, B, C, D) ~ 1
M1 <- poLCA(f, values, nclass = 2, verbose = FALSE)

M1
tidy(M1)
augment(M1)
glance(M1)

library(ggplot2)

ggplot(tidy(M1), aes(factor(class), estimate, fill = factor(outcome))) +
  geom_bar(stat = "identity", width = 1) +
  facet_wrap(~variable)
## Three-class model with a single covariate.

data(election)
f2a <- cbind(
  MORALG, CARESG, KNOWG, LEADG, DISHONG, INTELG,
  MORALB, CARESB, KNOWB, LEADB, DISHONB, INTELB
) ~ PARTY
nes2a <- poLCA(f2a, election, nclass = 3, nrep = 5, verbose = FALSE)

td <- tidy(nes2a)
td

# show

ggplot(td, aes(outcome, estimate, color = factor(class), group = class)) +
  geom_line() +
  facet_wrap(~variable, nrow = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

au <- augment(nes2a)
au
```

```
count(au, .class)

# if the original data is provided, it leads to NAs in new columns
# for rows that weren't predicted
au2 <- augment(nes2a, data = election)
au2
dim(au2)
```

glance.polr

*Glance at a(n) polr object*

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'polr'
glance(x, ...)
```

## Arguments

x	A polr object returned from <code>MASS::polr()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.

deviance	Deviance of the model.
df.residual	Residual degrees of freedom.
edf	The effective degrees of freedom.
logLik	The log-likelihood of the model. [stats::logLik()] may be a useful reference.
nobs	Number of observations used.

**See Also**

[tidy, MASS::polr\(\)](#)

Other ordinal tidiers: [augment.clm\(\)](#), [augment.polr\(\)](#), [glance.clmm\(\)](#), [glance.clm\(\)](#), [glance.svyolr\(\)](#), [tidy.clmm\(\)](#), [tidy.clm\(\)](#), [tidy.polr\(\)](#), [tidy.svyolr\(\)](#)

**Examples**

```
library(MASS)

fit <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)

tidy(fit, exponentiate = TRUE, conf.int = TRUE)

glance(fit)
augment(fit, type.predict = "class")

fit2 <- polr(factor(gear) ~ am + mpg + qsec, data = mtcars)
tidy(fit, p.values = TRUE)
```

---

glance.pyyears

*Glance at a(n) pyyears object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'pyyears'
glance(x, ...)
```

**Arguments**

`x` A pyyears object returned from `survival::pyyears()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>nobs</code>	Number of observations used.
<code>total</code>	total number of person-years tabulated
<code>offtable</code>	total number of person-years off table

**See Also**

`glance()`, `survival::pyyears()`

Other pyyears tidiers: `tidy.pyyears()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.survdiff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)

temp.yr <- tcut(mgus$dxyr, 55:92, labels = as.character(55:91))
temp.age <- tcut(mgus$age, 34:101, labels = as.character(34:100))
ptime <- ifelse(is.na(mgus$pctime), mgus$futime, mgus$pctime)
pstat <- ifelse(is.na(mgus$pctime), 0, 1)
pfit <- pyyears(Surv(ptime / 365.25, pstat) ~ temp.yr + temp.age + sex, mgus,
  data.frame = TRUE
)
tidy(pfit)
glance(pfit)

# if data.frame argument is not given, different information is present in
# output
```

```
pfit2 <- pyears(Surv(ptime / 365.25, pstat) ~ temp.yr + temp.age + sex, mgus)
tidy(pfit2)
glance(pfit2)
```

---

glance.ridgelm	<i>Glance at a(n) ridgelm object</i>
----------------	--------------------------------------

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'ridgelm'
glance(x, ...)
```

## Arguments

x	A <code>ridgelm</code> object returned from <code>MASS::lm.ridge()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

This is similar to the output of `select.ridgelm`, but it is returned rather than printed.

## Value

A `tibble::tibble()` with exactly one row and columns:

kHKB	modified HKB estimate of the ridge constant
kLW	modified L-W estimate of the ridge constant
lambdaGCV	choice of lambda that minimizes GCV



**See Also**

`glance()`, `MASS::select.ridgelm()`, `MASS::lm.ridge()`

Other `ridgelm` tidiers: `tidy.ridgelm()`

**Examples**

```
names(longley)[1] <- "y"
fit1 <- MASS::lm.ridge(y ~ ., longley)
tidy(fit1)

fit2 <- MASS::lm.ridge(y ~ ., longley, lambda = seq(0.001, .05, .001))
td2 <- tidy(fit2)
g2 <- glance(fit2)

# coefficient plot
library(ggplot2)
ggplot(td2, aes(lambda, estimate, color = term)) +
  geom_line()

# GCV plot
ggplot(td2, aes(lambda, GCV)) +
  geom_line()

# add line for the GCV minimizing estimate
ggplot(td2, aes(lambda, GCV)) +
  geom_line() +
  geom_vline(xintercept = g2$lambdaGCV, col = "red", lty = 2)
```

---

`glance.rlm`

*Glance at a(n) rlm object*

---

**Description**

`Glance` accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

`Glance` never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

`Glance` does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

`Glance` returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'rlm'
glance(x, ...)
```

**Arguments**

`x` An `rlm` object returned by `MASS::rlm()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
converged	Logical indicating if the model fitting procedure was successful and converged.
deviance	Deviance of the model.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
sigma	Estimated standard error of the residuals.

**See Also**

`glance()`, `MASS::rlm()`

Other `rlm` tidiers: `augment.rlm()`, `tidy.rlm()`

**Examples**

```
library(MASS)

r <- rlm(stack.loss ~ ., stackloss)

tidy(r)
augment(r)
glance(r)
```

glance.rma

*Glance at a(n) rma object***Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'rma'
glance(x, ...)
```

**Arguments**

x	An rma object such as those created by <code>metafor::rma()</code> , <code>metafor::rma.uni()</code> , <code>metafor::rma.glmm()</code> , <code>metafor::rma.mh()</code> , <code>metafor::rma.mv()</code> , or <code>metafor::rma.peto()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

cochran.qe	In meta-analysis, test statistic for the Cochran's $Q_e$ test of residual heterogeneity.
cochran.qm	In meta-analysis, test statistic for the Cochran's $Q_m$ omnibus test of coefficients.
df.residual	Residual degrees of freedom.
h.squared	Value of the H-Squared statistic.
i.squared	Value of the I-Squared statistic.

measure	The measure used in the meta-analysis.
method	Which method was used.
nobs	Number of observations used.
p.value.cochran.qe	In meta-analysis, p-value for the Cochran's $Q_e$ test of residual heterogeneity.
p.value.cochran.qm	In meta-analysis, p-value for the Cochran's $Q_m$ omnibus test of coefficients.
tau.squared	In meta-analysis, estimated amount of residual heterogeneity.
tau.squared.se	In meta-analysis, standard error of residual heterogeneity.

### Examples

```
library(metafor)

df <-
  escalc(
    measure = "RR",
    ai = tpos,
    bi = tneg,
    ci = cpos,
    di = cneg,
    data = dat.bcg
  )

meta_analysis <- rma(yi, vi, data = df, method = "EB")

glance(meta_analysis)
```

---

glance.rq

*Glance at a(n) rq object*

---

### Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'rq'
glance(x, ...)
```

**Arguments**

`x` An `rq` object returned from `quantreg::rq()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Details**

Only models with a single tau value may be passed. For multiple values, please use a `purrr::map()` workflow instead, e.g.

```
taus %>%
  map(function(tau_val) rq(y ~ x, tau = tau_val)) %>%
  map_dfr(glance)
```

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
tau	Quantile.

**See Also**

`glance()`, `quantreg::rq()`

Other `quantreg` tidiers: `augment.nlrq()`, `augment.rqs()`, `augment.rq()`, `glance.nlrq()`, `tidy.nlrq()`, `tidy.rqs()`, `tidy.rq()`

glance.sarlm

*Glance at a(n) spatialreg object*

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'sarlm'
glance(x, ...)
```

## Arguments

<code>x</code>	An object of object returned from <code>spatialreg::lagsarlm()</code> or <code>spatialreg::errorsarlm()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.
logLik	The log-likelihood of the model. <code>[stats::logLik()]</code> may be a useful reference.
nobs	Number of observations used.

**See Also**

`glance()`, `spatialreg::lagsarlm()`, `spatialreg::errorsarlm()`, `spatialreg::sacsarlm()`  
 Other spatialreg tidiers: `augment.sarlm()`, `tidy.sarlm()`

**Examples**

```
## Not run:
library(spatialreg)
data(oldcol, package="spdep")
listw <- spdep::nb2listw(COL.nb, style="W")

crime_sar <- lagsarlm(CRIME ~ INC + HOVAL, data=COL.OLD,
  listw=listw, method="eigen")

tidy(crime_sar)
tidy(crime_sar, conf.int = TRUE)
glance(crime_sar)
augment(crime_sar)

crime_sem <- errorsarlm(CRIME ~ INC + HOVAL, data=COL.OLD, listw)

tidy(crime_sem)
tidy(crime_sem, conf.int = TRUE)
glance(crime_sem)
augment(crime_sem)

crime_sac <- sacsarlm(CRIME ~ INC + HOVAL, data=COL.OLD, listw)

tidy(crime_sac)
tidy(crime_sac, conf.int = TRUE)
glance(crime_sac)
augment(crime_sac)

## End(Not run)
```

---

`glance.smooth.spline` *Tidy a(n) smooth.spline object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'smooth.spline'
glance(x, ...)
```

**Arguments**

<code>x</code>	A <code>smooth.spline</code> object returned from <code>stats::smooth.spline()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>crit</code>	Minimized criterion
<code>cv.crit</code>	Cross-validation score
<code>df</code>	Degrees of freedom used by the model.
<code>lambda</code>	Choice of lambda corresponding to 'spar'.
<code>nobs</code>	Number of observations used.
<code>pen.crit</code>	Penalized criterion.
<code>spar</code>	Smoothing parameter.

**See Also**

`augment()`, `stats::smooth.spline()`

Other smoothing spline tidiers: `augment.smooth.spline()`

**Examples**

```
spl <- smooth.spline(mtcars$wt, mtcars$mpg, df = 4)
augment(spl, mtcars)
augment(spl) # calls original columns x and y

library(ggplot2)
ggplot(augment(spl, mtcars), aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted))
```



---

glance.speedglm      *Glance at a(n) speedglm object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'speedglm'
glance(x, ...)
```

## Arguments

<code>x</code>	A <code>speedglm</code> object returned from <code>speedglm::speedglm()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
deviance	Deviance of the model.
df.null	Degrees of freedom used by the null model.
df.residual	Residual degrees of freedom.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
null.deviance	Deviance of the null model.

**See Also**

[speedglm::speedlm\(\)](#)

Other speedlm tidiers: [augment.speedlm\(\)](#), [glance.speedlm\(\)](#), [tidy.speedglm\(\)](#), [tidy.speedlm\(\)](#)

**Examples**

```
library(speedglm)

clotting <- data.frame(
  u = c(5, 10, 15, 20, 30, 40, 60, 80, 100),
  lot1 = c(118, 58, 42, 35, 27, 25, 21, 19, 18)
)

fit <- speedglm(lot1 ~ log(u), data = clotting, family = Gamma(log))

tidy(fit)
glance(fit)
```

---

`glance.speedlm`

*Glance at a(n) speedlm object*

---

**Description**

Glance accepts a model object and returns a [tibble::tibble\(\)](#) with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'speedlm'
glance(x, ...)
```

**Arguments**

x A speedlm object returned from [speedglm::speedlm\(\)](#).

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
<code>AIC</code>	Akaike's Information Criterion for the model.
<code>BIC</code>	Bayesian Information Criterion for the model.
<code>deviance</code>	Deviance of the model.
<code>df</code>	Degrees of freedom used by the model.
<code>df.residual</code>	Residual degrees of freedom.
<code>logLik</code>	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
<code>nobs</code>	Number of observations used.
<code>p.value</code>	P-value corresponding to the test statistic.
<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
<code>statistic</code>	F-statistic.

## See Also

[speedglm::speedlm\(\)](#)

Other `speedlm` tidiers: [augment.speedlm\(\)](#), [glance.speedglm\(\)](#), [tidy.speedglm\(\)](#), [tidy.speedlm\(\)](#)

## Examples

```
mod <- speedglm::speedlm(mpg ~ wt + qsec, data = mtcars, fitted = TRUE)
```

```
tidy(mod)
glance(mod)
augment(mod)
```

---

glance.summary.lm      *Glance at a(n) summary.lm object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'summary.lm'
glance(x, ...)
```

## Arguments

<code>x</code>	An <code>lm</code> object created by <code>stats::lm()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

The `glance.summary.lm()` method is a potentially useful alternative to `glance.lm()`. For instance, if users have already converted large `lm` objects into their leaner `summary.lm` equivalents to conserve memory. Note, however, that this method does not return all of the columns of the non-summary method (e.g. AIC and BIC will be missing.)

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>adj.r.squared</code>	Adjusted R squared statistic, which is like the R squared statistic except taking degrees of freedom into account.
----------------------------	--

df.residual	Residual degrees of freedom.
nobs	Number of observations used.
p.value	P-value corresponding to the test statistic.
r.squared	R squared statistic, or the percent of variation explained by the model. Also known as the coefficient of determination.
sigma	Estimated standard error of the residuals.
statistic	Test statistic.
df	The degrees for freedom from the numerator of the overall F-statistic. This is new in broom 0.7.0. Previously, this reported the rank of the design matrix, which is one more than the numerator degrees of freedom of the overall F-statistic.

**See Also**

[glance\(\)](#), [glance.summary.lm\(\)](#)

Other lm tidiers: [augment.glm\(\)](#), [augment.lm\(\)](#), [glance.glm\(\)](#), [glance.lm\(\)](#), [glance.svyglm\(\)](#), [tidy.glm\(\)](#), [tidy.lm.beta\(\)](#), [tidy.lm\(\)](#), [tidy.mlmm\(\)](#), [tidy.summary.lm\(\)](#)

**Examples**

```
library(ggplot2)
library(dplyr)

mod <- lm(mpg ~ wt + qsec, data = mtcars)

tidy(mod)
glance(mod)

# coefficient plot
d <- tidy(mod, conf.int = TRUE)

ggplot(d, aes(estimate, term, xmin = conf.low, xmax = conf.high, height = 0)) +
  geom_point() +
  geom_vline(xintercept = 0, lty = 4) +
  geom_errorbarh()

# Aside: There are tidy() and glance() methods for lm.summary objects too.
# This can be useful when you want to conserve memory by converting large lm
# objects into their leaner summary.lm equivalents.
s <- summary(mod)
tidy(s, conf.int = TRUE)
glance(s)

augment(mod)
augment(mod, mtcars, interval = "confidence")

# predict on new data
newdata <- mtcars %>%
```

```

  head(6) %>%
  mutate(wt = wt + 1)
augment(mod, newdata = newdata)

# ggplot2 example where we also construct 95% prediction interval
mod2 <- lm(mpg ~ wt, data = mtcars) ## simpler bivariate model since we're plotting in 2D

au <- augment(mod2, newdata = newdata, interval = "prediction")

ggplot(au, aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted)) +
  geom_ribbon(aes(ymin = .lower, ymax = .upper), col = NA, alpha = 0.3)

# predict on new data without outcome variable. Output does not include .resid
newdata <- newdata %>%
  select(-mpg)
augment(mod, newdata = newdata)

au <- augment(mod, data = mtcars)

ggplot(au, aes(.hat, .std.resid)) +
  geom_vline(size = 2, colour = "white", xintercept = 0) +
  geom_hline(size = 2, colour = "white", yintercept = 0) +
  geom_point() +
  geom_smooth(se = FALSE)

plot(mod, which = 6)
ggplot(au, aes(.hat, .cooksd)) +
  geom_vline(xintercept = 0, colour = NA) +
  geom_abline(slope = seq(0, 3, by = 0.5), colour = "white") +
  geom_smooth(se = FALSE) +
  geom_point()

# column-wise models
a <- matrix(rnorm(20), nrow = 10)
b <- a + rnorm(length(a))
result <- lm(b ~ a)
tidy(result)

```

---

glance.survdiff

*Glance at a(n) survdiff object*


---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

### Usage

```
## S3 method for class 'survdiff'
glance(x, ...)
```

### Arguments

x	An <code>survdiff</code> object returned from <code>survival::survdiff()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

### Value

A `tibble::tibble()` with exactly one row and columns:

df	Degrees of freedom used by the model.
p.value	P-value corresponding to the test statistic.
statistic	Test statistic.

### See Also

`glance()`, `survival::survdiff()`

Other `survdiff` tidiers: `tidy.survdiff()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

### Examples

```
library(survival)

s <- survdiff(
  Surv(time, status) ~ pat.karno + strata(inst),
  data = lung
)
```

```
tidy(s)
glance(s)
```

---

```
glance.survexp      Glance at a(n) survexp object
```

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'survexp'
glance(x, ...)
```

## Arguments

<code>x</code>	An survexp object returned from <code>survival::survexp()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>n.max</code>	Maximum number of subjects at risk.
<code>n.start</code>	Initial number of subjects at risk.
<code>timepoints</code>	Number of timepoints.



**See Also**

`glance()`, `survival::survexp()`

Other survexp tidiers: `tidy.survexp()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdiff()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)
sexpfit <- survexp(
  fuptime ~ 1,
  rmap = list(
    sex = "male",
    year = accept.dt,
    age = (accept.dt - birth.dt)
  ),
  method = "conditional",
  data = jasa
)

tidy(sexpfit)
glance(sexpfit)
```

---

`glance.survfit`

*Glance at a(n) survfit object*

---

**Description**

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

**Usage**

```
## S3 method for class 'survfit'
glance(x, ...)
```

**Arguments**

`x` An survfit object returned from `survival::survfit()`.  
`...` Additional arguments passed to `summary.survfit()`. Important arguments include `rmean`.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>events</code>	Number of events.
<code>n.max</code>	Maximum number of subjects at risk.
<code>n.start</code>	Initial number of subjects at risk.
<code>nobs</code>	Number of observations used.
<code>records</code>	Number of observations
<code>rmean</code>	Restricted mean (see <code>[survival::print.survfit()]</code> ).
<code>rmean.std.error</code>	Restricted mean standard error.
<code>conf.low</code>	lower end of confidence interval on median
<code>conf.high</code>	upper end of confidence interval on median
<code>median</code>	median survival

**See Also**

`glance()`, `survival::survfit()`

Other cch tidiers: `glance.cch()`, `tidy.cch()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdiff()`, `glance.survexp()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)
cfits <- coxph(Surv(time, status) ~ age + sex, lung)
sfit <- survfit(cfits)

tidy(sfit)
glance(sfit)

library(ggplot2)
ggplot(tidy(sfit), aes(time, estimate)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .25)

# multi-state
fitCI <- survfit(Surv(stop, status * as.numeric(event), type = "mstate") ~ 1,
```

```

  data = mgus1, subset = (start == 0)
)
td_multi <- tidy(fitCI)
td_multi

ggplot(td_multi, aes(time, estimate, group = state)) +
  geom_line(aes(color = state)) +
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .25)

```

---

glance.survreg

*Glance at a(n) survreg object*


---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```

## S3 method for class 'survreg'
glance(x, ...)

```

## Arguments

x	An survreg object returned from <code>survival::survreg()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

AIC	Akaike's Information Criterion for the model.
BIC	Bayesian Information Criterion for the model.
df	Degrees of freedom used by the model.
df.residual	Residual degrees of freedom.
iter	Iterations of algorithm/fitting procedure completed.
logLik	The log-likelihood of the model. [ <code>stats::logLik()</code> ] may be a useful reference.
nobs	Number of observations used.
p.value	P-value corresponding to the test statistic.
statistic	Chi-squared statistic.

**See Also**

`glance()`, `survival::survreg()`

Other survreg tidiers: `augment.survreg()`, `tidy.survreg()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdif()`, `glance.survexp()`, `glance.survfit()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdif()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)

sr <- survreg(
  Surv(futime, fustat) ~ ecog.ps + rx,
  ovarian,
  dist = "exponential"
)

tidy(sr)
augment(sr, ovarian)
glance(sr)

# coefficient plot
td <- tidy(sr, conf.int = TRUE)
library(ggplot2)
ggplot(td, aes(estimate, term)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high), height = 0) +
  geom_vline(xintercept = 0)
```

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'svyglm'
glance(x, maximal = x, ...)
```

## Arguments

<code>x</code>	A <code>svyglm</code> object returned from <code>survey::svyglm()</code> .
<code>maximal</code>	A <code>svyglm</code> object corresponding to the maximal model against which to compute the BIC. See Lumley and Scott (2015) for details. Defaults to <code>x</code> , which is equivalent to not using a maximal model.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

<code>AIC</code>	Akaike's Information Criterion for the model.
<code>BIC</code>	Bayesian Information Criterion for the model.
<code>deviance</code>	Deviance of the model.
<code>df.null</code>	Degrees of freedom used by the null model.
<code>df.residual</code>	Residual degrees of freedom.
<code>null.deviance</code>	Deviance of the null model.

## References

Lumley T, Scott A (2015). AIC and BIC for modelling with complex survey data. *Journal of Survey Statistics and Methodology*, 3(1). <https://doi.org/10.1093/jssam/smu021>.

## See Also

`survey::svyglm()`, `stats::glm()`, `survey::anova.svyglm`

Other lm tidiers: `augment.glm()`, `augment.lm()`, `glance.glm()`, `glance.lm()`, `glance.summary.lm()`, `tidy.glm()`, `tidy.lm.beta()`, `tidy.lm()`, `tidy.mlm()`, `tidy.summary.lm()`

## Examples

```
library(survey)

set.seed(123)
data(api)

# survey design
dstrat <-
  svydesign(
    id = ~1,
    strata = ~stype,
    weights = ~pw,
    data = apistrat,
    fpc = ~fpc
  )

# model
m <- survey::svyglm(
  formula = sch.wide ~ ell + meals + mobility,
  design = dstrat,
  family = quasibinomial()
)

glance(m)
```

---

glance.svyolr

*Glance at a(n) svyolr object*

---

## Description

Glance accepts a model object and returns a `tibble::tibble()` with exactly one row of model summaries. The summaries are typically goodness of fit measures, p-values for hypothesis tests on residuals, or model convergence information.

Glance never returns information from the original call to the modeling function. This includes the name of the modeling function or any arguments passed to the modeling function.

Glance does not calculate summary measures. Rather, it farms out these computations to appropriate methods and gathers the results together. Sometimes a goodness of fit measure will be undefined. In these cases the measure will be reported as NA.

Glance returns the same number of columns regardless of whether the model matrix is rank-deficient or not. If so, entries in columns that no longer have a well-defined value are filled in with an NA of the appropriate type.

## Usage

```
## S3 method for class 'svyolr'
glance(x, ...)
```

## Arguments

x	A <code>svyolr</code> object returned from <code>survey::svyolr()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with exactly one row and columns:

df.residual	Residual degrees of freedom.
edf	The effective degrees of freedom.
nobs	Number of observations used.

## See Also

`tidy`, `survey::svyolr()`

Other ordinal tidiers: `augment.clm()`, `augment.polr()`, `glance.clmm()`, `glance.clm()`, `glance.polr()`, `tidy.clmm()`, `tidy.clm()`, `tidy.polr()`, `tidy.svyolr()`

## Examples

```
library(MASS)

fit <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)

tidy(fit, exponentiate = TRUE, conf.int = TRUE)
glance(fit)
```

glance\_optim

*Tidy a(n) optim object masquerading as list***Description**

Broom tidies a number of lists that are effectively S3 objects without a class attribute. For example, `stats::optim()`, `svd()` and `akima::interp()` produce consistent output, but because they do not have a class attribute, they cannot be handled by S3 dispatch.

These functions look at the elements of a list and determine if there is an appropriate tidying method to apply to the list. Those tidiers are themselves implemented as functions of the form `tidy_<function>` or `glance_<function>` and are not exported (but they are documented!).

If no appropriate tidying method is found, throws an error.

**Usage**

```
glance_optim(x, ...)
```

**Arguments**

<code>x</code>	A list returned from <code>stats::optim()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with exactly one row and columns:

<code>convergence</code>	Convergence code.
<code>function.count</code>	Number of calls to 'fn'.
<code>gradient.count</code>	Number of calls to 'gr'.
<code>value</code>	Minimized or maximized output value.

**See Also**

`glance()`, `stats::optim()`

Other list tidiers: `list_tidiers`, `tidy_irlba()`, `tidy_optim()`, `tidy_svd()`, `tidy_xyz()`



## Examples

```
f <- function(x) (x[1] - 2)^2 + (x[2] - 3)^2 + (x[3] - 8)^2
o <- optim(c(1, 1, 1), f)
```

---

list\_tidiers

*Tidying methods for lists / returned values that are not S3 objects*

---

## Description

Broom tidies a number of lists that are effectively S3 objects without a class attribute. For example, `stats::optim()`, `base::svd()` and `akima::interp()` produce consistent output, but because they do not have a class attribute, they cannot be handled by S3 dispatch.

## Usage

```
## S3 method for class 'list'
tidy(x, ...)

## S3 method for class 'list'
glance(x, ...)
```

## Arguments

x                    A list, potentially representing an object that can be tidied.  
...                    Additionally, arguments passed to the tidying function.

## Details

These functions look at the elements of a list and determine if there is an appropriate tidying method to apply to the list. Those tidiers are themselves implemented as functions of the form `tidy_<function>` or `glance_<function>` and are not exported (but they are documented!).

If no appropriate tidying method is found, throws an error.

## See Also

Other list tidiers: [glance\\_optim\(\)](#), [tidy\\_irlba\(\)](#), [tidy\\_optim\(\)](#), [tidy\\_svd\(\)](#), [tidy\\_xyz\(\)](#)

---

null\_tidiers

*Tidiers for NULL inputs*


---

### Description

tidy(NULL), glance(NULL) and augment(NULL) all return an empty `tibble::tibble`. This empty tibble can be treated a tibble with zero rows, making it convenient to combine with other tibbles using functions like `purrr::map_df()` on lists of potentially NULL objects.

### Usage

```
## S3 method for class ``NULL``
tidy(x, ...)

## S3 method for class ``NULL``
glance(x, ...)

## S3 method for class ``NULL``
augment(x, ...)
```

### Arguments

x                    The value NULL.  
 ...                  Additional arguments (not used).

### Value

An empty `tibble::tibble`.

### See Also

[tibble::tibble](#)

---

sparse\_tidiers

*Tidy a sparseMatrix object from the Matrix package*


---

### Description

Tidy a `sparseMatrix` object from the `Matrix` package into a three-column data frame, row, column, and value (with zeros missing). If there are row names or column names, use those, otherwise use indices

**Usage**

```
## S3 method for class 'dgTMatrix'
tidy(x, ...)

## S3 method for class 'dgCMatrix'
tidy(x, ...)

## S3 method for class 'sparseMatrix'
tidy(x, ...)
```

**Arguments**

x	A Matrix object
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

row	Row ID of the original observation.
value	The value/estimate of the component. Results from data reshaping.
column	Column name in the original matrix.

---

sp\_tidiers

*Tidy a(n) SpatialPolygonsDataFrame object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

Note that the `sf` package now defines tidy spatial objects and is the recommended approach to spatial data. `sp tidiers` are likely to be deprecated in the near future in favor of `sf::st_as_sf()`. Development of `sp tidiers` has halted in broom.

**Usage**

```
## S3 method for class 'SpatialPolygonsDataFrame'
tidy(x, region = NULL, ...)

## S3 method for class 'SpatialPolygons'
tidy(x, ...)

## S3 method for class 'Polygons'
tidy(x, ...)

## S3 method for class 'Polygon'
tidy(x, ...)

## S3 method for class 'SpatialLinesDataFrame'
tidy(x, ...)

## S3 method for class 'Lines'
tidy(x, ...)

## S3 method for class 'Line'
tidy(x, ...)
```

**Arguments**

x	A SpatialPolygonsDataFrame, SpatialPolygons, Polygons, Polygon, SpatialLinesDataFrame, Lines or Line object.
region	name of variable used to split up regions
...	not used by this method

---

summary_tidiers	<i>(Deprecated) Tidy summaryDefault objects</i>
-----------------	---

---

**Description**

Tidiers for summaryDefault objects have been deprecated as of broom 0.7.0 in favor of `skimr::skim()`.

**Usage**

```
## S3 method for class 'summaryDefault'
tidy(x, ...)

## S3 method for class 'summaryDefault'
glance(x, ...)
```

**Arguments**

`x` A `summaryDefault` object, created by calling `summary()` on a vector.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A one-row `tibble::tibble` with columns:

<code>minimum</code>	Minimum value in original vector.
<code>q1</code>	First quartile of original vector.
<code>median</code>	Median of original vector.
<code>mean</code>	Mean of original vector.
<code>q3</code>	Third quartile of original vector.
<code>maximum</code>	Maximum value in original vector.
<code>na</code>	Number of NA values in original vector. Column present only when original vector had at least one NA entry.

**See Also**

Other deprecated: `bootstrap()`, `confint_tidy()`, `data.frame_tidiers`, `finish_glance()`, `fix_data_frame()`, `tidy.density()`, `tidy.dist()`, `tidy.ftable()`, `tidy.gamlss()`, `tidy.numeric()`

Other deprecated: `bootstrap()`, `confint_tidy()`, `data.frame_tidiers`, `finish_glance()`, `fix_data_frame()`, `tidy.density()`, `tidy.dist()`, `tidy.ftable()`, `tidy.gamlss()`, `tidy.numeric()`

**Examples**

```
v <- rnorm(1000)
s <- summary(v)
s

tidy(s)
glance(s)

v2 <- c(v, NA)
tidy(summary(v2))
```

---

tidy.aareg	<i>Tidy a(n) aareg object</i>
------------	-------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'aareg'
tidy(x, ...)
```

## Arguments

x	An aareg object returned from <code>survival::aareg()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

`robust.se` is only present when `x` was created with `dfbeta = TRUE`.

## Value

A `tibble::tibble()` with columns:

<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>robust.se</code>	robust version of standard error estimate.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.
<code>z</code>	z score.

**See Also**

`tidy()`, `survival::aareg()`

Other aareg tidiers: `glance.aareg()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdiff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)

afit <- aareg(
  Surv(time, status) ~ age + sex + ph.ecog,
  data = lung,
  dfbeta = TRUE
)

tidy(afit)
```

---

tidy.acf

*Tidy a(n) acf object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'acf'
tidy(x, ...)
```

**Arguments**

`x` An acf object created by `stats::acf()`, `stats::pacf()` or `stats::ccf()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with columns:

acf	Autocorrelation.
lag	Lag values.

**See Also**

`tidy()`, `stats::acf()`, `stats::pacf()`, `stats::ccf()`

Other time series tidiers: `tidy.spec()`, `tidy.ts()`, `tidy.zoo()`

**Examples**

```
tidy(acf(lh, plot = FALSE))
tidy(ccf(mdeaths, fdeaths, plot = FALSE))
tidy(pacf(lh, plot = FALSE))
```

---

tidy.anova	<i>Tidy a(n) anova object</i>
------------	-------------------------------

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'anova'
tidy(x, ...)
```

**Arguments**

x	An anova objects, such as those created by <code>stats::anova()</code> or <code>car::Anova()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

The term column of an ANOVA table can come with leading or trailing whitespace, which this tidying method trims.



**Value**

A `tibble::tibble()` with columns:

<code>df</code>	Degrees of freedom used by this term in the model.
<code>meansq</code>	Mean sum of squares. Equal to total sum of squares divided by degrees of freedom.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>sumsq</code>	Sum of squares explained by this term.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `stats::anova()`, `car::Anova()`

Other anova tidiers: `glance.aov()`, `tidy.TukeyHSD()`, `tidy.aovlist()`, `tidy.aov()`, `tidy.manova()`

**Examples**

```
a <- lm(mpg ~ wt + qsec + disp, mtcars)
b <- lm(mpg ~ wt + qsec, mtcars)
tidy(anova(a, b))
```

---

`tidy.aov`

*Tidy a(n) aov object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'aov'
tidy(x, ...)
```

## Arguments

- `x` An aov object, such as those created by `stats::aov()`.
- `...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Details

The term column of an ANOVA table can come with leading or trailing whitespace, which this tidying method trims.

## See Also

`tidy()`, `stats::aov()`

Other anova tidiers: `glance.aov()`, `tidy.TukeyHSD()`, `tidy.anova()`, `tidy.aovlist()`, `tidy.manova()`

## Examples

```
a <- aov(mpg ~ wt + qsec + disp, mtcars)
tidy(a)
```

---

<code>tidy.aovlist</code>	<i>Tidy a(n) aovlist object</i>
---------------------------	---------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'aovlist'
tidy(x, ...)
```

**Arguments**

<code>x</code>	An aovlist objects, such as those created by <code>stats::aov()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Details**

The term column of an ANOVA table can come with leading or trailing whitespace, which this tidying method trims.

**Value**

A `tibble::tibble()` with columns:

<code>df</code>	Degrees of freedom used by this term in the model.
<code>meansq</code>	Mean sum of squares. Equal to total sum of squares divided by degrees of freedom.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>stratum</code>	The error stratum.
<code>sumsq</code>	Sum of squares explained by this term.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `stats::aov()`

Other anova tidiers: `glance.aov()`, `tidy.TukeyHSD()`, `tidy.anova()`, `tidy.aov()`, `tidy.manova()`

**Examples**

```
a <- aov(mpg ~ wt + qsec + Error(displ / am), mtcars)
tidy(a)
```

tidy.Arima

*Tidy a(n) Arima object*

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'Arima'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

## Arguments

x	An object of class Arima created by <code>stats::arima()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
std.error	The standard error of the regression term.
term	The name of the regression term.

## See Also

`stats::arima()`

Other Arima tidiers: `glance.Arima()`

**Examples**

```
fit <- arima(lh, order = c(1, 0, 0))

tidy(fit)
glance(fit)
```

---

tidy.betamfx	<i>Tidy a(n) betamfx object</i>
--------------	---------------------------------

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'betamfx'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	A betamfx object.
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if conf.int = TRUE. Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass conf.level = 0.9, all computation will proceed using conf.level = 0.95. Additionally, if you pass newdata = my_tibble to an <code>augment()</code> method that does not accept a newdata argument, it will use the default value for the data argument.

**Details**

The mfx package provides methods for calculating marginal effects for various generalized linear models (GLMs). Unlike standard linear models, estimated model coefficients in a GLM cannot be directly interpreted as marginal effects (i.e., the change in the response variable predicted after a one unit change in one of the regressors). This is because the estimated coefficients are multiplicative, dependent on both the link function that was used for the estimation and any other variables that were included in the model. When calculating marginal effects, users must typically choose whether they want to use i) the average observation in the data, or ii) the average of the sample marginal effects. See `vignette("mfxarticle")` from the mfx package for more details.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.
<code>atmean</code>	TRUE if the marginal effects were originally calculated as the partial effects for the average observation. If FALSE, then these were instead calculated as average partial effects.

**See Also**

`tidy.betareg()`, `mfx::betamfx()`

Other mfx tidiers: `augment.betamfx()`, `augment.mfx()`, `glance.betamfx()`, `glance.mfx()`, `tidy.mfx()`

**Examples**

```
## Not run:
library(mfx)

## Simulate some data
set.seed(12345)
n = 1000
x = rnorm(n)

## Beta outcome
y = rbeta(n, shape1 = plogis(1 + 0.5 * x), shape2 = (abs(0.2*x)))
## Use Smithson and Verkuilen correction
y = (y*(n-1)+0.5)/n

d = data.frame(y,x)
mod_betamfx = betamfx(y ~ x | x, data = d)

tidy(mod_betamfx, conf.int = TRUE)

## Compare with the naive model coefficients of the equivalent betareg call (not run)
# tidy(betamfx(y ~ x | x, data = d), conf.int = TRUE)

augment(mod_betamfx)
glance(mod_betamfx)

## End(Not run)
```

---

tidy.betareg	<i>Tidy a(n) betareg object</i>
--------------	---------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'betareg'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

## Arguments

x	A betareg object produced by a call to <code>betareg::betareg()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

The tibble has one row for each term in the regression. The `component` column indicates whether a particular term was used to model either the "mean" or "precision". Here the precision is the inverse of the variance, often referred to as  $\phi$ . At least one term will have been used to model the precision  $\phi$ .

## Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.

statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.
component	Whether a particular term was used to model the mean or the precision in the regression. See details.

**See Also**

[tidy\(\)](#), [betareg::betareg\(\)](#)

**Examples**

```
library(betareg)
data("GasolineYield", package = "betareg")

mod <- betareg(yield ~ batch + temp, data = GasolineYield)

mod
tidy(mod)
tidy(mod, conf.int = TRUE)
tidy(mod, conf.int = TRUE, conf.level = .99)

augment(mod)

glance(mod)
```

---

tidy.biglm

*Tidy a(n) biglm object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'biglm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)
```



**Arguments**

<code>x</code>	A <code>biglm</code> object created by a call to <code>biglm::biglm()</code> or <code>biglm::bigglm()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>exponentiate</code>	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to <code>FALSE</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `biglm::biglm()`, `biglm::bigglm()`

Other `biglm` tidiers: `glance.biglm()`

**Examples**

```
## Not run:
library(biglm)

bfit <- biglm(mpg ~ wt + disp, mtcars)
tidy(bfit)
tidy(bfit, conf.int = TRUE)
tidy(bfit, conf.int = TRUE, conf.level = .9)
```

```

glance(bfit)

# bigglm: logistic regression
bfit <- bigglm(am ~ mpg, mtcars, family = binomial())

tidy(bfit)
tidy(bfit, exponentiate = TRUE)
tidy(bfit, conf.int = TRUE)
tidy(bfit, conf.int = TRUE, conf.level = .9)
tidy(bfit, conf.int = TRUE, conf.level = .9, exponentiate = TRUE)

glance(bfit)

## End(Not run)

```

---

tidy.binDesign

*Tidy a(n) binDesign object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'binDesign'
tidy(x, ...)

```

## Arguments

x	A <code>binGroup::binDesign()</code> object.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

n	Number of trials in given iteration.
power	Power achieved for given value of n.

**See Also**

`tidy()`, `binGroup::binDesign()`

Other bingroup tidiers: `glance.binDesign()`, `tidy.binWidth()`

**Examples**

```
library(binGroup)
des <- binDesign(
  nmax = 300, delta = 0.06,
  p.hyp = 0.1, power = .8
)

glance(des)
tidy(des)

# the ggplot2 equivalent of plot(des)
library(ggplot2)
ggplot(tidy(des), aes(n, power)) +
  geom_line()
```

---

`tidy.binWidth`

*Tidy a(n) binWidth object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'binWidth'
tidy(x, ...)
```

**Arguments**

`x` A `binGroup::binWidth()` object.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>alternative</code>	Alternative hypothesis (character).
<code>ci.width</code>	Expected width of confidence interval.
<code>p</code>	True proportion.
<code>n</code>	Total sample size

**See Also**

`tidy()`, `binGroup::binWidth()`

Other bingroup tidiers: `glance.binDesign()`, `tidy.binDesign()`

**Examples**

```
library(binGroup)
library(dplyr)
library(ggplot2)

bw <- binWidth(100, .1)
bw
tidy(bw)
```

---

tidy.boot

*Tidy a(n) boot object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'boot'
tidy(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  conf.method = c("perc", "bca", "basic", "norm"),
  ...
)
```

**Arguments**

<code>x</code>	A <code>boot::boot()</code> object.
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>conf.method</code>	Passed to the type argument of <code>boot::boot.ci()</code> . Defaults to "perc". The allowed types are "perc", "basic", "bca", and "norm". Does not support "stud" or "all".
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

If weights were provided to the `boot` function, an `estimate` column is included showing the weighted bootstrap estimate, and the standard error is of that estimate.

If there are no original statistics in the "boot" object, such as with a call to `tsboot` with `orig.t = FALSE`, the original and `statistic` columns are omitted, and only `estimate` and `std.error` columns shown.

**Value**

A `tibble::tibble()` with columns:

<code>bias</code>	Bias of the statistic.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.
<code>statistic</code>	Original value of the statistic.

**See Also**

`tidy()`, `boot::boot()`, `boot::tsboot()`, `boot::boot.ci()`, `rsample::bootstraps()`

**Examples**

```
library(boot)

clotting <- data.frame(
  u = c(5, 10, 15, 20, 30, 40, 60, 80, 100),
  lot1 = c(118, 58, 42, 35, 27, 25, 21, 19, 18),
```

```

lot2 = c(69, 35, 26, 21, 18, 16, 13, 12, 12)
)

g1 <- glm(lot2 ~ log(u), data = clotting, family = Gamma)

bootfun <- function(d, i) {
  coef(update(g1, data = d[i, ]))
}

bootres <- boot(clotting, bootfun, R = 999)
tidy(g1, conf.int = TRUE)
tidy(bootres, conf.int = TRUE)

```

---

tidy.btergm

*Tidy a(n) btergm object*


---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

This method tidies the coefficients of a bootstrapped temporal exponential random graph model estimated with the **xergm**. It simply returns the coefficients and their confidence intervals.

### Usage

```

## S3 method for class 'btergm'
tidy(x, conf.level = 0.95, exponentiate = FALSE, ...)

```

### Arguments

x	A <code>btergm:btergm()</code> object.
conf.level	Confidence level for confidence intervals. Defaults to 0.95.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `btergm::btergm()`

---

`tidy.cch`

*Tidy a(n) cch object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'cch'
tidy(x, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	An cch object returned from <code>survival::cch()</code> .
<code>conf.level</code>	confidence level for CI
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.

p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

### See Also

[tidy\(\)](#), [survival::cch\(\)](#)

Other cch tidiers: [glance.cch\(\)](#), [glance.survfit\(\)](#)

Other survival tidiers: [augment.coxph\(\)](#), [augment.survreg\(\)](#), [glance.aareg\(\)](#), [glance.cch\(\)](#), [glance.coxph\(\)](#), [glance.pyears\(\)](#), [glance.survdiff\(\)](#), [glance.survexp\(\)](#), [glance.survfit\(\)](#), [glance.survreg\(\)](#), [tidy.aareg\(\)](#), [tidy.coxph\(\)](#), [tidy.pyears\(\)](#), [tidy.survdiff\(\)](#), [tidy.survexp\(\)](#), [tidy.survfit\(\)](#), [tidy.survreg\(\)](#)

### Examples

```
library(survival)

# examples come from cch documentation
subcoh <- nwtco$in.subcohort
selccoh <- with(nwtco, rel == 1 | subcoh == 1)
ccoh.data <- nwtco[selccoh, ]
ccoh.data$subcohort <- subcoh[selccoh]
## central-lab histology
ccoh.data$histol <- factor(ccoh.data$histol, labels = c("FH", "UH"))
## tumour stage
ccoh.data$stage <- factor(ccoh.data$stage, labels = c("I", "II", "III", "IV"))
ccoh.data$age <- ccoh.data$age / 12 # Age in years

fit.ccP <- cch(Surv(edrel, rel) ~ stage + histol + age,
  data = ccoh.data,
  subcoh = ~subcohort, id = ~seqno, cohort.size = 4028
)

tidy(fit.ccP)

# coefficient plot
library(ggplot2)
ggplot(tidy(fit.ccP), aes(x = estimate, y = term)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high), height = 0) +
  geom_vline(xintercept = 0)
```



tidy.cld

*Tidy a(n) cld object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'cld'
tidy(x, ...)
```

**Arguments**

`x` A cld object created by calling `multcomp::cld()` on a `glht`, `confint.glht()` or `summary.glht()` object.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

`contrast` Levels being compared.

`letters` Compact letter display denoting all pair-wise comparisons.

**See Also**

`tidy()`, `multcomp::cld()`, `multcomp::summary.glht()`, `multcomp::confint.glht()`, `multcomp::glht()`  
 Other multcomp tidiers: `tidy.confint.glht()`, `tidy.glht()`, `tidy.summary.glht()`

**Examples**

```
library(multcomp)
library(ggplot2)

amod <- aov(breaks ~ wool + tension, data = warpbreaks)
wht <- glht(amod, linfct = mcp(tension = "Tukey"))
```

```

tidy(wht)
ggplot(wht, aes(lhs, estimate)) +
  geom_point()

CI <- confint(wht)
tidy(CI)
ggplot(CI, aes(lhs, estimate, ymin = lwr, ymax = upr)) +
  geom_pointrange()

tidy(summary(wht))
ggplot(mapping = aes(lhs, estimate)) +
  geom_linerange(aes(ymin = lwr, ymax = upr), data = CI) +
  geom_point(aes(size = p), data = summary(wht)) +
  scale_size(trans = "reverse")

cld <- cld(wht)
tidy(cld)

```

---

tidy.clm

*Tidy a(n) clm object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'clm'
tidy(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  conf.type = c("profile", "Wald"),
  exponentiate = FALSE,
  ...
)

```

## Arguments

x	A clm object returned from <code>ordinal::clm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.

<code>conf.type</code>	Whether to use "profile" or "Wald" confidence intervals, passed to the <code>type</code> argument of <code>ordinal::confint.clm()</code> . Defaults to "profile".
<code>exponentiate</code>	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

### Details

In broom 0.7.0 the `coefficient_type` column was renamed to `coef.type`, and the contents were changed as well.

Note that `intercept` type coefficients correspond to alpha parameters, `location` type coefficients correspond to beta parameters, and `scale` type coefficients correspond to zeta parameters.

### Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

### See Also

`tidy`, `ordinal::clm()`, `ordinal::confint.clm()`

Other ordinal tidiers: `augment.clm()`, `augment.polr()`, `glance.clmm()`, `glance.clm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clmm()`, `tidy.polr()`, `tidy.svyolr()`

### Examples

```
library(ordinal)

fit <- clm(rating ~ temp * contact, data = wine)

tidy(fit)
```

```

tidy(fit, conf.int = TRUE, conf.level = 0.9)
tidy(fit, conf.int = TRUE, conf.type = "Wald", exponentiate = TRUE)

glance(fit)
augment(fit, type.predict = "prob")
augment(fit, type.predict = "class")

fit2 <- clm(rating ~ temp, nominal = ~contact, data = wine)
tidy(fit2)
glance(fit2)

```

---

tidy.clmm

*Tidy a(n) clmm object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'clmm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)

```

## Arguments

x	A clmm object returned from <code>ordinal::clmm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**Note**

In broom 0.7.0 the `coefficient_type` column was renamed to `coef.type`, and the contents were changed as well.

Note that `intercept` type coefficients correspond to alpha parameters, `location` type coefficients correspond to beta parameters, and `scale` type coefficients correspond to zeta parameters.

**See Also**

`tidy.ordinal::clmm()`, `ordinal::confint.clm()`

Other ordinal tidiers: `augment.clm()`, `augment.polr()`, `glance.clm()`, `glance.clm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clm()`, `tidy.polr()`, `tidy.svyolr()`

**Examples**

```
library(ordinal)

fit <- clmm(rating ~ temp + contact + (1 | judge), data = wine)

tidy(fit)
tidy(fit, conf.int = TRUE, conf.level = 0.9)
tidy(fit, conf.int = TRUE, exponentiate = TRUE)

glance(fit)

fit2 <- clmm(rating ~ temp + (1 | judge), nominal = ~contact, data = wine)
tidy(fit2)
glance(fit2)
```

---

tidy.coefstest	<i>Tidy a(n) coefstest object</i>
----------------	-----------------------------------

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'coefstest'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

### Arguments

x	A coefstest object returned from <code>lmtest::coefstest()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

[tidy\(\)](#), [lmtest::coefstest\(\)](#)

**Examples**

```
library(lmtest)

m <- lm(dist ~ speed, data = cars)

coefstest(m)
tidy(coefstest(m))
tidy(coefstest(m, conf.int = TRUE))

# A very common workflow is to combine lmtest::coefstest with alternate
# variance-covariance matrices via the sandwich package. The lmtest
# tidiers support this workflow too, enabling you to adjust the standard
# errors of your tidied models on the fly.
library(sandwich)
tidy(coefstest(m, vcov = vcovHC))           # "HC3" (default) robust SEs
tidy(coefstest(m, vcov = vcovHC, type = "HC2")) # "HC2" robust SEs
tidy(coefstest(m, vcov = NeweyWest))       # N-W HAC robust SEs

# The columns of the returned tibble for glance.coefstest() will vary
# depending on whether the coefstest object retains the underlying model.
# Users can control this with the "save = TRUE" argument of coefstest().
glance(coefstest(m))
glance(coefstest(m, save = TRUE)) # More columns
```

---

tidy.confint.glht	<i>Tidy a(n) confint.glht object</i>
-------------------	--------------------------------------

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'confint.glht'
tidy(x, ...)
```

**Arguments**

x                    A `confint.glht` object created by calling `multcomp::confint.glht()` on a `glht` object created with `multcomp::glht()`.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>contrast</code>	Levels being compared.
<code>estimate</code>	The estimated value of the regression term.

## See Also

`tidy()`, `multcomp::confint.glht()`, `multcomp::glht()`

Other multcomp tidiers: `tidy.cld()`, `tidy.glht()`, `tidy.summary.glht()`

## Examples

```
library(multcomp)
library(ggplot2)

amod <- aov(breaks ~ wool + tension, data = warpbreaks)
wht <- glht(amod, linfct = mcp(tension = "Tukey"))

tidy(wht)
ggplot(wht, aes(lhs, estimate)) +
  geom_point()

CI <- confint(wht)
tidy(CI)
ggplot(CI, aes(lhs, estimate, ymin = lwr, ymax = upr)) +
  geom_pointrange()

tidy(summary(wht))
ggplot(mapping = aes(lhs, estimate)) +
  geom_linerange(aes(ymin = lwr, ymax = upr), data = CI) +
  geom_point(aes(size = p), data = summary(wht)) +
  scale_size(trans = "reverse")

cld <- cld(wht)
tidy(cld)
```



---

tidy.confusionMatrix *Tidy a(n) confusionMatrix object*


---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'confusionMatrix'
tidy(x, by_class = TRUE, ...)
```

### Arguments

x	An object of class confusionMatrix created by a call to <code>caret::confusionMatrix()</code> .
by_class	Logical indicating whether or not to show performance measures broken down by class. Defaults to TRUE. When by_class = FALSE only returns a tibble with accuracy, kappa, and McNemar statistics.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with columns:

class	The class under consideration.
conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
term	The name of the regression term.
p.value	P-value for accuracy and kappa statistics.

### See Also

`tidy()`, `caret::confusionMatrix()`

## Examples

```
library(caret)

set.seed(27)

two_class_sample1 <- as.factor(sample(letters[1:2], 100, TRUE))
two_class_sample2 <- as.factor(sample(letters[1:2], 100, TRUE))

two_class_cm <- caret::confusionMatrix(
  two_class_sample1,
  two_class_sample2
)

tidy(two_class_cm)
tidy(two_class_cm, by_class = FALSE)

# multiclass example

six_class_sample1 <- as.factor(sample(letters[1:6], 100, TRUE))
six_class_sample2 <- as.factor(sample(letters[1:6], 100, TRUE))

six_class_cm <- caret::confusionMatrix(
  six_class_sample1,
  six_class_sample2
)

tidy(six_class_cm)
tidy(six_class_cm, by_class = FALSE)
```

---

tidy.coxph

*Tidy a(n) coxph object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'coxph'
tidy(x, exponentiate = FALSE, conf.int = FALSE, conf.level = 0.95, ...)
```

## Arguments

x A coxph object returned from `survival::coxph()`.

<code>exponentiate</code>	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Value

A `tibble::tibble()` with columns:

<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.

### See Also

`tidy()`, `survival::coxph()`

Other coxph tidiers: `augment.coxph()`, `glance.coxph()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdifff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.pyears()`, `tidy.survdifff()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

### Examples

```
library(survival)

cfits <- coxph(Surv(time, status) ~ age + sex, lung)

tidy(cfits)
tidy(cfits, exponentiate = TRUE)

lp <- augment(cfits, lung)
risks <- augment(cfits, lung, type.predict = "risk")
expected <- augment(cfits, lung, type.predict = "expected")
```

```

glance(cfit)

# also works on clogit models
resp <- levels(logan$occupation)
n <- nrow(logan)
indx <- rep(1:n, length(resp))
logan2 <- data.frame(
  logan[indx, ],
  id = indx,
  tocc = factor(rep(resp, each = n))
)

logan2$case <- (logan2$occupation == logan2$tocc)

cl <- clogit(case ~ tocc + tocc:education + strata(id), logan2)
tidy(cl)
glance(cl)

library(ggplot2)

ggplot(lp, aes(age, .fitted, color = sex)) +
  geom_point()

ggplot(risks, aes(age, .fitted, color = sex)) +
  geom_point()

ggplot(expected, aes(time, .fitted, color = sex)) +
  geom_point()

```

---

tidy.cv.glmnet

*Tidy a(n) cv.glmnet object*


---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'cv.glmnet'
tidy(x, ...)
```

### Arguments

x A cv.glmnet object returned from `glmnet::cv.glmnet()`.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with columns:

<code>lambda</code>	Value of penalty parameter <code>lambda</code> .
<code>nzero</code>	Number of non-zero coefficients for the given <code>lambda</code> .
<code>std.error</code>	The standard error of the regression term.
<code>conf.low</code>	lower bound on confidence interval for cross-validation estimated loss.
<code>conf.high</code>	upper bound on confidence interval for cross-validation estimated loss.
<code>estimate</code>	Median loss across all cross-validation folds for a given <code>lambda</code>

## See Also

`tidy()`, `glmnet::cv.glmnet()`

Other `glmnet` tidiers: `glance.cv.glmnet()`, `glance.glmnet()`, `tidy.glmnet()`

## Examples

```
library(glmnet)
set.seed(27)

nobs <- 100
nvar <- 50
real <- 5

x <- matrix(rnorm(nobs * nvar), nobs, nvar)
beta <- c(rnorm(real, 0, 1), rep(0, nvar - real))
y <- c(t(beta) %*% t(x)) + rnorm(nvar, sd = 3)

cvfit1 <- cv.glmnet(x, y)

tidy(cvfit1)
glance(cvfit1)

library(ggplot2)
tidied_cv <- tidy(cvfit1)
glance_cv <- glance(cvfit1)

# plot of MSE as a function of lambda
g <- ggplot(tidied_cv, aes(lambda, estimate)) +
  geom_line() +
```

```

  scale_x_log10()
g

# plot of MSE as a function of lambda with confidence ribbon
g <- g + geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .25)
g

# plot of MSE as a function of lambda with confidence ribbon and choices
# of minimum lambda marked
g <- g +
  geom_vline(xintercept = glance_cv$lambda.min) +
  geom_vline(xintercept = glance_cv$lambda.1se, lty = 2)
g

# plot of number of zeros for each choice of lambda
ggplot(tidied_cv, aes(lambda, nzero)) +
  geom_line() +
  scale_x_log10()

# coefficient plot with min lambda shown
tidied <- tidy(cvfit1$glmnet.fit)

ggplot(tidied, aes(lambda, estimate, group = term)) +
  scale_x_log10() +
  geom_line() +
  geom_vline(xintercept = glance_cv$lambda.min) +
  geom_vline(xintercept = glance_cv$lambda.1se, lty = 2)

```

---

tidy.density

*(Deprecated) Tidy density objects*


---

## Description

(Deprecated) Tidy density objects

## Usage

```

## S3 method for class 'density'
tidy(x, ...)

```

## Arguments

**x** A density object returned from `stats::density()`.

**...** Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble` with two columns: points  $x$  where the density is estimated, and estimated density  $y$ .

**See Also**

Other deprecated: `bootstrap()`, `confint_tidy()`, `data.frame_tidiers`, `finish_glance()`, `fix_data_frame()`, `summary_tidiers`, `tidy.dist()`, `tidy.ftable()`, `tidy.gamlss()`, `tidy.numeric()`

---

tidy.dist	<i>(Deprecated) Tidy dist objects</i>
-----------	---------------------------------------

---

**Description**

(Deprecated) Tidy dist objects

**Usage**

```
## S3 method for class 'dist'
tidy(x, diagonal = attr(x, "Diag"), upper = attr(x, "Upper"), ...)
```

**Arguments**

<code>x</code>	A dist object returned from <code>stats::dist()</code> .
<code>diagonal</code>	Logical indicating whether or not to tidy the diagonal elements of the distance matrix. Defaults to whatever was based to the <code>diag</code> argument of <code>stats::dist()</code> .
<code>upper</code>	Logical indicating whether or not to tidy the upper half of the distance matrix. Defaults to whatever was based to the <code>upper</code> argument of <code>stats::dist()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

If the distance matrix does not include an upper triangle and/or diagonal, the tidied version will not either.

**Value**

A `tibble::tibble` with one row for each pair of items in the distance matrix, with columns:

<code>item1</code>	First item
<code>item2</code>	Second item
<code>distance</code>	Distance between items

**See Also**

Other deprecated: `bootstrap()`, `confint_tidy()`, `data.frame_tidiers`, `finish_glance()`, `fix_data_frame()`, `summary_tidiers`, `tidy.density()`, `tidy.ftable()`, `tidy.gamlss()`, `tidy.numeric()`

**Examples**

```
cars_dist <- dist(t(mtcars[, 1:4]))
cars_dist

tidy(cars_dist)
tidy(cars_dist, upper = TRUE)
tidy(cars_dist, diagonal = TRUE)
```

---

tidy.drc

*Tidy a(n) drc object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'drc'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	A drc object produced by a call to <code>drc::drm()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.



## Details

The tibble has one row for each curve and term in the regression. The `curveid` column indicates the curve.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.
<code>curve</code>	Index identifying the curve.

## See Also

`tidy()`, `drc::drm()`

Other drc tidiers: `augment.drc()`, `glance.drc()`

## Examples

```
library(drc)

mod <- drm(dead / total ~ conc, type,
  weights = total, data = selenium, fct = LL.2(), type = "binomial"
)

tidy(mod)
tidy(mod, conf.int = TRUE)

glance(mod)

augment(mod, selenium)
```

---

tidy.emmGrid	<i>Tidy a(n) emmGrid object</i>
--------------	---------------------------------

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'emmGrid'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

### Arguments

x	An emmGrid object.
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if conf.int = TRUE. Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments passed to <a href="#">emmeans::summary.emmGrid()</a> or <a href="#">lsmeans::summary.ref.grid()</a> . <b>Cautionary note:</b> misspecified arguments may be silently ignored!

### Details

Returns a data frame with one observation for each estimated marginal mean, and one column for each combination of factors. When the input is a contrast, each row will contain one estimated contrast.

There are a large number of arguments that can be passed on to [emmeans::summary.emmGrid\(\)](#) or [lsmeans::summary.ref.grid\(\)](#).

### Value

A [tibble::tibble\(\)](#) with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
df	Degrees of freedom used by this term in the model.
p.value	The two-sided p-value associated with the observed statistic.
std.error	The standard error of the regression term.
estimate	Expected marginal mean
statistic	T-ratio statistic

**See Also**

`tidy()`, `emmeans::ref_grid()`, `emmeans::emmeans()`, `emmeans::contrast()`

Other emmeans tidiers: `tidy.lsmobj()`, `tidy.ref.grid()`, `tidy.summary.emm()`

**Examples**

```
library(emmeans)
# linear model for sales of oranges per day
oranges_lm1 <- lm(sales1 ~ price1 + price2 + day + store, data = oranges)

# reference grid; see vignette("basics", package = "emmeans")
oranges_rg1 <- ref_grid(oranges_lm1)
td <- tidy(oranges_rg1)
td

# marginal averages
marginal <- emmeans(oranges_rg1, "day")
tidy(marginal)

# contrasts
tidy(contrast(marginal))
tidy(contrast(marginal, method = "pairwise"))

# plot confidence intervals
library(ggplot2)
ggplot(tidy(marginal, conf.int = TRUE), aes(day, estimate)) +
  geom_point() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# by multiple prices
by_price <- emmeans(oranges_lm1, "day",
  by = "price2",
  at = list(
    price1 = 50, price2 = c(40, 60, 80),
    day = c("2", "3", "4")
  )
)
by_price
tidy(by_price)

ggplot(tidy(by_price, conf.int = TRUE), aes(price2, estimate, color = day)) +
  geom_line() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# joint_tests
tidy(joint_tests(oranges_lm1))
```

---

tidy.epi.2by2	<i>Tidy a(n) epi.2by2 object</i>
---------------	----------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'epi.2by2'
tidy(x, parameters = c("moa", "stat"), ...)
```

## Arguments

x	A epi.2by2 object produced by a call to <code>epiR::epi.2by2()</code>
parameters	Return measures of association (moa) or test statistics (stat), default is moa (measures of association)
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

The tibble has a column for each of the measures of association or tests contained in `massoc` when `epiR::epi.2by2()` is called.

## Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
df	Degrees of freedom used by this term in the model.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
term	The name of the regression term.
estimate	Estimated measure of association

**See Also**

[tidy\(\)](#), [epiR::epi.2by2\(\)](#)

**Examples**

```
library(epiR)
dat <- matrix(c(13, 2163, 5, 3349), nrow = 2, byrow = TRUE)
rownames(dat) <- c("DF+", "DF-")
colnames(dat) <- c("FUS+", "FUS-")
fit <- epi.2by2(
  dat = as.table(dat), method = "cross.sectional",
  conf.level = 0.95, units = 100, outcome = "as.columns"
)

tidy(fit, parameters = "moa")
```

---

tidy.ergm

*Tidy a(n) ergm object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

The methods should work with any model that conforms to the **ergm** class, such as those produced from weighted networks by the **ergm.count** package.

**Usage**

```
## S3 method for class 'ergm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)
```

**Arguments**

x	An ergm object returned from a call to <a href="#">ergm::ergm()</a> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if conf.int = TRUE. Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
...	Additional arguments to pass to <a href="#">ergm::summary()</a> . <b>Cautionary note:</b> Mis-specified arguments may be silently ignored.

**Value**

A `tibble::tibble` with one row for each coefficient in the exponential random graph model, with columns:

<code>term</code>	The term in the model being estimated and tested
<code>estimate</code>	The estimated coefficient
<code>std.error</code>	The standard error
<code>mcmc.error</code>	The MCMC error
<code>p.value</code>	The two-sided p-value

**References**

Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). **ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks. *Journal of Statistical Software*, 24(3). <https://www.jstatsoft.org/v24/i03/>.

**See Also**

`tidy()`, `ergm::ergm()`, `ergm::control.ergm()`, `ergm::summary()`

Other ergm tidiers: `glance.ergm()`

**Examples**

```
library(ergm)
# Using the same example as the ergm package
# Load the Florentine marriage network data
data(florentine)

# Fit a model where the propensity to form ties between
# families depends on the absolute difference in wealth
gest <- ergm(flomarriage ~ edges + absdiff("wealth"))

# Show terms, coefficient estimates and errors
tidy(gest)

# Show coefficients as odds ratios with a 99% CI
tidy(gest, exponentiate = TRUE, conf.int = TRUE, conf.level = 0.99)

# Take a look at likelihood measures and other
# control parameters used during MCMC estimation
glance(gest)
glance(gest, deviance = TRUE)
glance(gest, mcmc = TRUE)
```

---

tidy.factanal	<i>Tidy a(n) factanal object</i>
---------------	----------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'factanal'
tidy(x, ...)
```

## Arguments

x	A factanal object created by <code>stats::factanal()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with columns:

variable	Variable under consideration.
uniqueness	Proportion of residual, or unexplained variance
flX	Factor loading for level X.

## See Also

`tidy()`, `stats::factanal()`

Other factanal tidiers: `augment.factanal()`, `glance.factanal()`

## Examples

```
set.seed(123)

# data
m1 <- dplyr::tibble(
  v1 = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 4, 5, 6),
```

```

v2 = c(1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 3, 4, 3, 3, 3, 4, 6, 5),
v3 = c(3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 5, 4, 6),
v4 = c(3, 3, 4, 3, 3, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 5, 6, 4),
v5 = c(1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 6, 4, 5),
v6 = c(1, 1, 1, 2, 1, 3, 3, 3, 4, 3, 1, 1, 1, 2, 1, 6, 5, 4)
)

# new data
m2 <- purrr::map_dfr(m1, rev)

# factor analysis objects
fit1 <- stats::factanal(m1, factors = 3, scores = "Bartlett")
fit2 <- stats::factanal(m1, factors = 3, scores = "regression")

# tidying the object
tidy(fit1)
tidy(fit2)

# augmented dataframe
augment(fit1)
augment(fit2)

# augmented dataframe (with new data)
augment(fit1, data = m2)
augment(fit2, data = m2)

```

tidy.fitdistr

*Tidy a(n) fitdistr object*

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'fitdistr'
tidy(x, ...)
```

## Arguments

**x** A fitdistr object returned by `MASS::fitdistr()`.

**...** Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to



an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with columns:

<code>estimate</code>	The estimated value of the regression term.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

### See Also

`tidy()`, `MASS::fitdistr()`

Other `fitdistr` tidiers: `glance.fitdistr()`

### Examples

```
set.seed(2015)
x <- rnorm(100, 5, 2)

library(MASS)
fit <- fitdistr(x, dnorm, list(mean = 3, sd = 1))

tidy(fit)
glance(fit)
```

---

`tidy.fixest`

*Tidy a(n) fixest object*

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'fixest'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	A <code>fixest</code> object returned from any of the <code>fixest</code> estimators
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments passed to <code>summary</code> and <code>confint</code> . Important arguments are <code>se</code> and <code>cluster</code> . Other arguments are <code>dof</code> , <code>exact_dof</code> , <code>forceCovariance</code> , and <code>keepBounded</code> . See <a href="#">summary.fixest</a> .

**Details**

The `fixest` package provides a family of functions for estimating models with arbitrary numbers of fixed-effects, in both an OLS and a GLM context. The package also supports robust (i.e. White) and clustered standard error reporting via the generic `summary.fixest()` command. In a similar vein, the `tidy()` method for these models allows users to specify a desired standard error correction either 1) implicitly via the supplied `fixest` object, or 2) explicitly as part of the `tidy` call. See examples below.

Note that `fixest` confidence intervals are calculated assuming a normal distribution – this assumes infinite degrees of freedom for the CI. (This assumption is distinct from the degrees of freedom used to calculate the standard errors. For more on degrees of freedom with clusters and fixed effects, see <https://github.com/lrberge/fixest/issues/6> and <https://github.com/sgaure/lfe/issues/1#issuecomment-530646990>)

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

[tidy\(\)](#), [fixest::feglm\(\)](#), [fixest::fenegbin\(\)](#), [fixest::feNmlm\(\)](#), [fixest::femlm\(\)](#), [fixest::feols\(\)](#), [fixest::fepois\(\)](#)

Other `fixest` tidiers: [augment.fixest\(\)](#)

**Examples**

```

library(fixest)
gravity <- feols(log(Euros) ~ log(dist_km) | Origin + Destination + Product + Year, trade)

tidy(gravity)
glance(gravity)
augment(gravity, trade)

## To get robust or clustered SEs, users can either:
# 1) Or, specify the arguments directly in the tidy() call
tidy(gravity, conf.int = TRUE, cluster = c("Product", "Year"))
tidy(gravity, conf.int = TRUE, se = "threeway")
# 2) Feed tidy() a summary.fixest object that has already accepted these arguments
gravity_summ <- summary(gravity, cluster = c("Product", "Year"))
tidy(gravity_summ, conf.int = TRUE)
# Approach (1) is preferred.

## The other fixest methods all work similarly. For example:
gravity_pois <- feglm(Euros ~ log(dist_km) | Origin + Destination + Product + Year, trade)
tidy(gravity_pois)
glance(gravity_pois)
augment(gravity_pois, trade)

```

---

tidy.ftable

*(Deprecated) Tidy ftable objects*


---

**Description**

This function is deprecated. Please use `tibble::as_tibble()` instead.

**Usage**

```

## S3 method for class 'ftable'
tidy(x, ...)

```

**Arguments**

`x` An `ftable` object returned from `stats::ftable()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

An ftable contains a "flat" contingency table. This melts it into a `tibble::tibble` with one column for each variable, then a `Freq` column.

**See Also**

Other deprecated: `bootstrap()`, `confint_tidy()`, `data.frame_tidiers`, `finish_glance()`, `fix_data_frame()`, `summary_tidiers`, `tidy.density()`, `tidy.dist()`, `tidy.gamlss()`, `tidy.numeric()`

---

tidy.gam

*Tidy a(n) gam object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'gam'
tidy(x, parametric = FALSE, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	A gam object returned from a call to <code>mgcv::gam()</code> .
<code>parametric</code>	Logical indicating if parametric or smooth terms should be tidied. Defaults to <code>FALSE</code> , meaning that smooth terms are tidied by default.
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

When `parametric = FALSE` return columns `edf` and `ref.df` rather than `estimate` and `std.error`.

**Value**

A `tibble::tibble()` with columns:

<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.
<code>edf</code>	The effective degrees of freedom. Only reported when <code>'parametric = FALSE'</code>
<code>ref.df</code>	The reference degrees of freedom. Only reported when <code>'parametric = FALSE'</code>

**See Also**

`tidy()`, `mgcv::gam()`

Other mgcv tidiers: `glance.gam()`

**Examples**

```
g <- mgcv::gam(mpg ~ s(hp) + am + qsec, data = mtcars)

tidy(g)
tidy(g, parametric = TRUE)
glance(g)
```

---

`tidy.gamlss`

*Tidy a(n) gamlss object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'gamlss'
tidy(x, ...)
```

**Arguments**

x	A <code>gamlss</code> object returned from <code>gamlss::gamlss()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.
parameter	Type of coefficient being estimated: 'mu', 'sigma', 'nu', or 'tau'.

**See Also**

Other deprecated: `bootstrap()`, `confint_tidy()`, `data.frame_tidiers`, `finish_glance()`, `fix_data_frame()`, `summary_tidiers`, `tidy.density()`, `tidy.dist()`, `tidy.ftable()`, `tidy.numeric()`

**Examples**

```
library(gamlss)

g <- gamlss(
  y ~ pb(x),
  sigma.fo = ~ pb(x),
  family = BCT,
  data = abdom,
  method = mixed(1, 20)
)

tidy(g)
```

tidy.garch

*Tidy a(n) garch object*

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'garch'
tidy(x, ...)
```

## Arguments

x	A garch object returned by <code>tseries::garch()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

## See Also

`tidy()`, `tseries::garch()`

Other garch tidiers: `glance.garch()`

## Examples

```
library(tseries)

data(EuStockMarkets)
dax <- diff(log(EuStockMarkets))[, "DAX"]
dax.garch <- garch(dax)
dax.garch

tidy(dax.garch)
glance(dax.garch)
```

---

tidy.geeglm	<i>Tidy a(n) geeglm object</i>
-------------	--------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'geeglm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)
```

## Arguments

x	A geeglm object returned from a call to <code>geepack::geeglm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.



## Details

If `conf.int = TRUE`, the confidence interval is computed with the an internal `confint.geeglm()` function.

If you have missing values in your model data, you may need to refit the model with `na.action = na.exclude` or deal with the missingness in the data beforehand.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

## See Also

[tidy\(\)](#), [geepack::geeglm\(\)](#)

## Examples

```
library(geepack)
data(state)

ds <- data.frame(state.region, state.x77)

geefit <- geeglm(Income ~ Frost + Murder,
  id = state.region,
  data = ds, family = gaussian,
  corstr = "exchangeable"
)

tidy(geefit)
tidy(geefit, conf.int = TRUE)
```

tidy.glht

*Tidy a(n) glht object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'glht'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	A glht object returned by <code>multcomp::glht()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

contrast	Levels being compared.
estimate	The estimated value of the regression term.
null.value	Value to which the estimate is compared.

**See Also**

`tidy()`, `multcomp::glht()`

Other multcomp tidiers: `tidy.cld()`, `tidy.confint.glht()`, `tidy.summary.glht()`

**Examples**

```

library(multcomp)
library(ggplot2)

amod <- aov(breaks ~ wool + tension, data = warpbreaks)
wht <- glht(amod, linfct = mcp(tension = "Tukey"))

tidy(wht)
ggplot(wht, aes(lhs, estimate)) +
  geom_point()

CI <- confint(wht)
tidy(CI)
ggplot(CI, aes(lhs, estimate, ymin = lwr, ymax = upr)) +
  geom_pointrange()

tidy(summary(wht))
ggplot(mapping = aes(lhs, estimate)) +
  geom_linerange(aes(ymin = lwr, ymax = upr), data = CI) +
  geom_point(aes(size = p), data = summary(wht)) +
  scale_size(trans = "reverse")

cld <- cld(wht)
tidy(cld)

```

---

`tidy.glm`*Tidy a(n) glm object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```

## S3 method for class 'glm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)

```

**Arguments**

<code>x</code>	A <code>glm</code> object returned from <code>stats::glm()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.

exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### See Also

`stats::glm()`

Other lm tidiers: `augment.glm()`, `augment.lm()`, `glance.glm()`, `glance.lm()`, `glance.summary.lm()`, `glance.svyglm()`, `tidy.lm.beta()`, `tidy.lm()`, `tidy.mlml()`, `tidy.summary.lm()`

---

tidy.glmnet

*Tidy a(n) glmnet object*

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'glmnet'
tidy(x, return_zeros = FALSE, ...)
```

### Arguments

x	A glmnet object returned from <code>glmnet::glmnet()</code> .
return_zeros	Logical indicating whether coefficients with value zero should be included in the results. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

Note that while this representation of GLMs is much easier to plot and combine than the default structure, it is also much more memory-intensive. Do not use for large, sparse matrices.

No augment method is yet provided even though the model produces predictions, because the input data is not tidy (it is a matrix that may be very wide) and therefore combining predictions with it is not logical. Furthermore, predictions make sense only with a specific choice of lambda.

**Value**

A `tibble::tibble()` with columns:

<code>dev.ratio</code>	Fraction of null deviance explained at each value of lambda.
<code>estimate</code>	The estimated value of the regression term.
<code>lambda</code>	Value of penalty parameter lambda.
<code>step</code>	Which step of lambda choices was used.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `glmnet::glmnet()`

Other glmnet tidiers: `glance.cv.glmnet()`, `glance.glmnet()`, `tidy.cv.glmnet()`

**Examples**

```
library(glmnet)

set.seed(2014)
x <- matrix(rnorm(100 * 20), 100, 20)
y <- rnorm(100)
fit1 <- glmnet(x, y)

tidy(fit1)
glance(fit1)

library(dplyr)
library(ggplot2)

tidied <- tidy(fit1) %>% filter(term != "(Intercept)")

ggplot(tidied, aes(step, estimate, group = term)) +
  geom_line()
ggplot(tidied, aes(lambda, estimate, group = term)) +
  geom_line() +
  scale_x_log10()

ggplot(tidied, aes(lambda, dev.ratio)) +
  geom_line()
```

```
# works for other types of regressions as well, such as logistic
g2 <- sample(1:2, 100, replace = TRUE)
fit2 <- glmnet(x, g2, family = "binomial")
tidy(fit2)
```

---

tidy.glmRob	<i>Tidy a(n) glmRob object</i>
-------------	--------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'glmRob'
tidy(x, ...)
```

## Arguments

x	A glmRob object returned from <code>robust::glmRob()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

For tidiers for robust models from the **MASS** package see `tidy.rlm()`.

## See Also

`robust::glmRob()`

Other robust tidiers: `augment.lmRob()`, `glance.glmRob()`, `glance.lmRob()`, `tidy.lmRob()`

## Examples

```
library(robust)

gm <- glmRob(am ~ wt, data = mtcars, family = "binomial")

tidy(gm)
glance(gm)
```

---

tidy.glmrob	<i>Tidy a(n) glmrob object</i>
-------------	--------------------------------

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'glmrob'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

### Arguments

x	A <code>glmrob</code> object returned from <code>robustbase::glmrob()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

### Details

For tidiers for robust models from the **MASS** package see `tidy.rlm()`.

### Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

[robustbase::glmrob\(\)](#)

Other robustbase tidiers: [augment.glmrob\(\)](#), [augment.lmrob\(\)](#), [glance.lmrob\(\)](#), [tidy.lmrob\(\)](#)

**Examples**

```
library(robustbase)
# From the robustbase::lmrob examples:
data(coleman)
set.seed(0)

m <- robustbase::lmrob(Y ~ ., data = coleman)
tidy(m)
augment(m)
glance(m)

# From the robustbase::glmrob examples:
data(carrots)
Rfit <- glmrob(cbind(success, total - success) ~ logdose + block,
  family = binomial, data = carrots, method = "Mqle",
  control = glmrobMqle.control(tcc = 1.2)
)
tidy(Rfit)
augment(Rfit)
```

---

tidy.gmm

*Tidy a(n) gmm object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'gmm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)
```

**Arguments**

**x** A gmm object returned from [gmm::gmm\(\)](#).

**conf.int** Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.



<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>exponentiate</code>	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

## See Also

`tidy()`, `gmm::gmm()`

Other gmm tidiers: `glance.gmm()`

## Examples

```
library(gmm)

# examples come from the "gmm" package
## CAPM test with GMM
data(Finance)
r <- Finance[1:300, 1:10]
rm <- Finance[1:300, "rm"]
rf <- Finance[1:300, "rf"]

z <- as.matrix(r - rf)
t <- nrow(z)
zm <- rm - rf
h <- matrix(zm, t, 1)
```

```

res <- gmm(z ~ zm, x = h)

# tidy result
tidy(res)
tidy(res, conf.int = TRUE)
tidy(res, conf.int = TRUE, conf.level = .99)

# coefficient plot
library(ggplot2)
library(dplyr)

tidy(res, conf.int = TRUE) %>%
  mutate(variable = reorder(term, estimate)) %>%
  ggplot(aes(estimate, variable)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high)) +
  geom_vline(xintercept = 0, color = "red", lty = 2)

# from a function instead of a matrix
g <- function(theta, x) {
  e <- x[, 2:11] - theta[1] - (x[, 1] - theta[1]) %*% matrix(theta[2:11], 1, 10)
  gmat <- cbind(e, e * c(x[, 1]))
  return(gmat)
}

x <- as.matrix(cbind(rm, r))
res_black <- gmm(g, x = x, t0 = rep(0, 11))

tidy(res_black)
tidy(res_black, conf.int = TRUE)

## APT test with Fama-French factors and GMM

f1 <- zm
f2 <- Finance[1:300, "hml"] - rf
f3 <- Finance[1:300, "smb"] - rf
h <- cbind(f1, f2, f3)
res2 <- gmm(z ~ f1 + f2 + f3, x = h)

td2 <- tidy(res2, conf.int = TRUE)
td2

# coefficient plot
td2 %>%
  mutate(variable = reorder(term, estimate)) %>%
  ggplot(aes(estimate, variable)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high)) +
  geom_vline(xintercept = 0, color = "red", lty = 2)

```

**Description**

For models that have only a single component, the `tidy()` and `glance()` methods are identical. Please see the documentation for both of those methods.

**Usage**

```
## S3 method for class 'htest'
tidy(x, ...)
```

```
## S3 method for class 'htest'
glance(x, ...)
```

**Arguments**

`x` An `htest` object, such as those created by `stats::cor.test()`, `stats::t.test()`, `stats::wilcox.test()`, `stats::chisq.test()`, etc.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with columns:

<code>alternative</code>	Alternative hypothesis (character).
<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>estimate1</code>	Sometimes two estimates are computed, such as in a two-sample t-test.
<code>estimate2</code>	Sometimes two estimates are computed, such as in a two-sample t-test.
<code>method</code>	Method used.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>parameter</code>	The parameter being modeled.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.

**See Also**

`tidy()`, `stats::cor.test()`, `stats::t.test()`, `stats::wilcox.test()`, `stats::chisq.test()`  
 Other `htest` tidiers: `augment.htest()`, `tidy.pairwise.htest()`, `tidy.power.htest()`

**Examples**

```

tt <- t.test(rnorm(10))
tidy(tt)
glance(tt) # same output for all htests

tt <- t.test(mpg ~ am, data = mtcars)
tidy(tt)

wt <- wilcox.test(mpg ~ am, data = mtcars, conf.int = TRUE, exact = FALSE)
tidy(wt)

ct <- cor.test(mtcars$wt, mtcars$mpg)
tidy(ct)

chit <- chisq.test(xtabs(Freq ~ Sex + Class, data = as.data.frame(Titanic)))
tidy(chit)
augment(chit)

```

---

tidy.ivreg

*Tidy a(n) ivreg object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```

## S3 method for class 'ivreg'
tidy(x, conf.int = FALSE, conf.level = 0.95, instruments = FALSE, ...)

```

**Arguments**

<code>x</code>	An ivreg object created by a call to <code>AER::ivreg()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>instruments</code>	Logical indicating whether to return coefficients from the second-stage or diagnostics tests for each endogenous regressor (F-statistics). Defaults to FALSE.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Details

This tidier currently only supports `ivreg`-classed objects outputted by the AER package. The `ivreg` package also outputs objects of class `ivreg`, and will be supported in a later release.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>p.value.Sargan</code>	p-value for Sargan test of overidentifying restrictions.
<code>p.value.weakinst</code>	p-value for weak instruments test.
<code>p.value.Wu.Hausman</code>	p-value for Wu-Hausman weak instruments test for endogeneity.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>statistic.Sargan</code>	Statistic for Sargan test of overidentifying restrictions.
<code>statistic.weakinst</code>	Statistic for Wu-Hausman test.
<code>statistic.Wu.Hausman</code>	Statistic for Wu-Hausman weak instruments test for endogeneity.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

## See Also

`tidy()`, `AER::ivreg()`

Other `ivreg` tidiers: `augment.ivreg()`, `glance.ivreg()`

## Examples

```
library(AER)

data("CigarettesSW", package = "AER")

ivr <- ivreg(
  log(packs) ~ income | population,
  data = CigarettesSW,
  subset = year == "1995"
)

summary(ivr)

tidy(ivr)
tidy(ivr, conf.int = TRUE)
tidy(ivr, conf.int = TRUE, instruments = TRUE)

augment(ivr)
augment(ivr, data = CigarettesSW)
augment(ivr, newdata = CigarettesSW)

glance(ivr)
```

---

tidy.kappa

*Tidy a(n) kappa object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'kappa'
tidy(x, ...)
```

## Arguments

x A kappa object returned from `psych::cohen.kappa()`.

... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Details**

Note that confidence level (alpha) for the confidence interval cannot be set in `tidy`. Instead you must set the `alpha` argument to `psych::cohen.kappa()` when creating the `kappa` object.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>type</code>	Either 'weighted' or 'unweighted'.

**See Also**

`tidy()`, `psych::cohen.kappa()`

**Examples**

```
library(psych)

rater1 <- 1:9
rater2 <- c(1, 3, 1, 6, 1, 5, 5, 6, 7)
ck <- cohen.kappa(cbind(rater1, rater2))

tidy(ck)

# graph the confidence intervals
library(ggplot2)
ggplot(tidy(ck), aes(estimate, type)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high))
```

---

tidy.kde

*Tidy a(n) kde object*

---

**Description**

`Tidy` summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what `tidy` considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'kde'
tidy(x, ...)
```

**Arguments**

<code>x</code>	A kde object returned from <code>ks::kde()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

Returns a data frame in long format with four columns. Use `tidyr::pivot_wider(..., names_from = variable, values_from = value)` on the output to return to a wide format.

**Value**

A `tibble::tibble()` with columns:

<code>estimate</code>	The estimated value of the regression term.
<code>obs</code>	weighted observed number of events in each group.
<code>value</code>	The value/estimate of the component. Results from data reshaping.
<code>variable</code>	Variable under consideration.

**See Also**

`tidy()`, `ks::kde()`

**Examples**

```
library(ks)

dat <- replicate(2, rnorm(100))
k <- kde(dat)

td <- tidy(k)
td

library(ggplot2)
library(dplyr)
library(tidyr)

td %>%
  pivot_wider(c(obs, estimate),
             names_from = variable,
             values_from = value
  ) %>%
  ggplot(aes(x1, x2, fill = estimate)) +
  geom_tile() +
```



```

  theme_void()

# also works with 3 dimensions
dat3 <- replicate(3, rnorm(100))
k3 <- kde(dat3)

td3 <- tidy(k3)
td3

```

---

tidy.Kendall	<i>Tidy a(n) Kendall object</i>
--------------	---------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'Kendall'
tidy(x, ...)

```

## Arguments

x	A Kendall object returned from a call to <code>Kendall::Kendall()</code> , <code>Kendall::MannKendall()</code> , or <code>Kendall::SeasonalMannKendall()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

kendall_score	Kendall score.
p.value	The two-sided p-value associated with the observed statistic.
var_kendall_score	Variance of the kendall_score.
statistic	Kendall's tau statistic
denominator	The denominator, which is $\tau = \text{kendall\_score} / \text{denominator}$ .

**See Also**

[tidy\(\)](#), [Kendall::Kendall\(\)](#), [Kendall::MannKendall\(\)](#), [Kendall::SeasonalMannKendall\(\)](#)

**Examples**

```
library(Kendall)

A <- c(2.5, 2.5, 2.5, 2.5, 5, 6.5, 6.5, 10, 10, 10, 10, 14, 14, 14, 16, 17)
B <- c(1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 2, 2)

f_res <- Kendall(A, B)
tidy(f_res)

s_res <- MannKendall(B)
tidy(s_res)

t_res <- SeasonalMannKendall(ts(A))
tidy(t_res)
```

---

tidy.kmeans

*Tidy a(n) kmeans object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'kmeans'
tidy(x, col.names = colnames(x$centers), ...)
```

**Arguments**

x	A kmeans object created by <a href="#">stats::kmeans()</a> .
col.names	Dimension names. Defaults to the names of the variables in x. Set to NULL to get names x1, x2, ....
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <a href="#">augment()</a> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

For examples, see the kmeans vignette.

**Value**

A `tibble::tibble()` with columns:

<code>cluster</code>	A factor describing the cluster from 1:k.
<code>size</code>	Number of points assigned to cluster.
<code>withinss</code>	The within-cluster sum of squares.

**See Also**

`tidy()`, `stats::kmeans()`

Other kmeans tidiers: `augment.kmeans()`, `glance.kmeans()`

**Examples**

```
## Not run:
library(cluster)
library(dplyr)

library(modeldata)
data(hpc_data)

x <- hpc_data[, 2:5]

fit <- pam(x, k = 4)

tidy(fit)
glance(fit)
augment(fit, x)

## End(Not run)
```

---

tidy.lavaan

*Tidy a(n) lavaan object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'lavaan'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	A lavaan object, such as those returned from <code>lavaan::cfa()</code> , and <code>lavaan::sem()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments passed to <code>lavaan::parameterEstimates()</code> . <b>Cautionary note:</b> Misspecified arguments may be silently ignored.

**Value**

A `tibble::tibble()` with one row for each estimated parameter and columns:

term	The result of <code>paste(lhs, op, rhs)</code>
op	The operator in the model syntax (e.g. <code>~~</code> for covariances, or <code>~</code> for regression parameters)
group	The group (if specified) in the lavaan model
estimate	The parameter estimate (may be standardized)
std.error	
statistic	The z value returned by <code>lavaan::parameterEstimates()</code>
p.value	
conf.low	
conf.high	
std.lv	Standardized estimates based on the variances of the (continuous) latent variables only
std.all	Standardized estimates based on both the variances of both (continuous) observed and latent variables.
std.nox	Standardized estimates based on both the variances of both (continuous) observed and latent variables, but not the variances of exogenous covariates.

**See Also**

`tidy()`, `lavaan::cfa()`, `lavaan::sem()`, `lavaan::parameterEstimates()`

Other lavaan tidiers: `glance.lavaan()`

**Examples**

```
## Not run:
library(lavaan)

cfa.fit <- cfa("F =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9",
  data = HolzingerSwineford1939, group = "school"
)

tidy(cfa.fit)

## End(Not run)
```

---

tidy.lm	<i>Tidy a(n) lm object</i>
---------	----------------------------

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'lm'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	An lm object created by <code>stats::lm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

If the linear model is an `m lm` object (multiple linear model), there is an additional column response. See `tidy.mlm()`.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `stats::summary.lm()`

Other lm tidiers: `augment.glm()`, `augment.lm()`, `glance.glm()`, `glance.lm()`, `glance.summary.lm()`, `glance.svyglm()`, `tidy.glm()`, `tidy.lm.beta()`, `tidy.mlml()`, `tidy.summary.lm()`

**Examples**

```
library(ggplot2)
library(dplyr)

mod <- lm(mpg ~ wt + qsec, data = mtcars)

tidy(mod)
glance(mod)

# coefficient plot
d <- tidy(mod, conf.int = TRUE)

ggplot(d, aes(estimate, term, xmin = conf.low, xmax = conf.high, height = 0)) +
  geom_point() +
  geom_vline(xintercept = 0, lty = 4) +
  geom_errorbarh()

# Aside: There are tidy() and glance() methods for lm.summary objects too.
# This can be useful when you want to conserve memory by converting large lm
# objects into their leaner summary.lm equivalents.
s <- summary(mod)
tidy(s, conf.int = TRUE)
glance(s)

augment(mod)
augment(mod, mtcars, interval = "confidence")

# predict on new data
newdata <- mtcars %>%
```

```

  head(6) %>%
  mutate(wt = wt + 1)
augment(mod, newdata = newdata)

# ggplot2 example where we also construct 95% prediction interval
mod2 <- lm(mpg ~ wt, data = mtcars) ## simpler bivariate model since we're plotting in 2D

au <- augment(mod2, newdata = newdata, interval = "prediction")

ggplot(au, aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted)) +
  geom_ribbon(aes(ymin = .lower, ymax = .upper), col = NA, alpha = 0.3)

# predict on new data without outcome variable. Output does not include .resid
newdata <- newdata %>%
  select(-mpg)
augment(mod, newdata = newdata)

au <- augment(mod, data = mtcars)

ggplot(au, aes(.hat, .std.resid)) +
  geom_vline(size = 2, colour = "white", xintercept = 0) +
  geom_hline(size = 2, colour = "white", yintercept = 0) +
  geom_point() +
  geom_smooth(se = FALSE)

plot(mod, which = 6)
ggplot(au, aes(.hat, .cooks)) +
  geom_vline(xintercept = 0, colour = NA) +
  geom_abline(slope = seq(0, 3, by = 0.5), colour = "white") +
  geom_smooth(se = FALSE) +
  geom_point()

# column-wise models
a <- matrix(rnorm(20), nrow = 10)
b <- a + rnorm(length(a))
result <- lm(b ~ a)
tidy(result)

```

---

tidy.lm.beta

*Tidy a(n) lm.beta object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'lm.beta'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	An <code>lm.beta</code> object created by <code>lm.beta::lm.beta</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Details**

If the linear model is an `mlm` object (multiple linear model), there is an additional column response.

If you have missing values in your model data, you may need to refit the model with `na.action = na.exclude`.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

Other `lm` tidiers: `augment.glm()`, `augment.lm()`, `glance.glm()`, `glance.lm()`, `glance.summary.lm()`, `glance.svyglm()`, `tidy.glm()`, `tidy.lm()`, `tidy.mlm()`, `tidy.summary.lm()`



## Examples

```
library(lm.beta)

mod <- stats::lm(speed ~ ., data = cars)
std <- lm.beta(mod)
tidy(std, conf.int = TRUE)

ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
group <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
weight <- c(ctl, trt)

mod2 <- lm(weight ~ group)

std2 <- lm.beta(mod2)
tidy(std2, conf.int = TRUE)
```

---

tidy.lmodel2

*Tidy a(n) lmodel2 object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'lmodel2'
tidy(x, ...)
```

## Arguments

x	A lmodel2 object returned by <code>lmodel2::lmodel2()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

There are always only two terms in an `lmodel2`: "Intercept" and "Slope". These are computed by four methods: OLS (ordinary least squares), MA (major axis), SMA (standard major axis), and RMA (ranged major axis).

The returned p-value is one-tailed and calculated via a permutation test. A permutational test is used because distributional assumptions may not be valid. More information can be found in `vignette("mod2user", package = "lmodel2")`.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>term</code>	The name of the regression term.
<code>method</code>	Either OLS/MA/SMA/RMA

## See Also

`tidy()`, `lmodel2::lmodel2()`

Other `lmodel2` tidiers: `glance.lmodel2()`

## Examples

```
library(lmodel2)

data(mod2ex2)
Ex2.res <- lmodel2(Prey ~ Predators, data = mod2ex2, "relative", "relative", 99)
Ex2.res

tidy(Ex2.res)
glance(Ex2.res)

# this allows coefficient plots with ggplot2
library(ggplot2)
ggplot(tidy(Ex2.res), aes(estimate, term, color = method)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high)) +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high))
```

---

`tidy.lmRob`*Tidy a(n) lmRob object*

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'lmRob'  
tidy(x, ...)
```

### Arguments

`x` A `lmRob` object returned from `robust::lmRob()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Details

For tidiers for robust models from the **MASS** package see `tidy.rlm()`.

### See Also

`robust::lmRob()`

Other robust tidiers: `augment.lmRob()`, `glance.glmRob()`, `glance.lmRob()`, `tidy.glmRob()`

### Examples

```
library(robust)  
m <- lmRob(mpg ~ wt, data = mtcars)  
  
tidy(m)  
augment(m)  
glance(m)
```

---

tidy.lmrob	<i>Tidy a(n) lmrob object</i>
------------	-------------------------------

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'lmrob'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

### Arguments

x	A lmrob object returned from <code>robustbase::lmrob()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Details

For tidiers for robust models from the MASS package see `tidy.rlm()`.

### See Also

`robustbase::lmrob()`

Other robustbase tidiers: `augment.glmrob()`, `augment.lmrob()`, `glance.lmrob()`, `tidy.glmrob()`

### Examples

```
library(robustbase)
# From the robustbase::lmrob examples:
data(coleman)
set.seed(0)
```

```

m <- robustbase::lmrob(Y ~ ., data = coleman)
tidy(m)
augment(m)
glance(m)

# From the robustbase::glmrob examples:
data(carrots)
Rfit <- glmrob(cbind(success, total - success) ~ logdose + block,
  family = binomial, data = carrots, method = "Mqle",
  control = glmrobMqle.control(tcc = 1.2)
)
tidy(Rfit)
augment(Rfit)

```

tidy.lsmobj

*Tidy a(n) lsmobj object*

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'lsmobj'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

```

## Arguments

<code>x</code>	An lsmobj object.
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments passed to <a href="#">emmeans::summary.emmGrid()</a> or <a href="#">lsmeans::summary.ref.grid()</a> . <b>Cautionary note:</b> misspecified arguments may be silently ignored!

## Details

Returns a data frame with one observation for each estimated marginal mean, and one column for each combination of factors. When the input is a contrast, each row will contain one estimated contrast.

There are a large number of arguments that can be passed on to [emmeans::summary.emmGrid\(\)](#) or [lsmeans::summary.ref.grid\(\)](#).

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>contrast</code>	Levels being compared.
<code>df</code>	Degrees of freedom used by this term in the model.
<code>null.value</code>	Value to which the estimate is compared.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>std.error</code>	The standard error of the regression term.
<code>estimate</code>	Expected marginal mean
<code>statistic</code>	T-ratio statistic

**See Also**

`tidy()`, `emmeans::ref_grid()`, `emmeans::emmeans()`, `emmeans::contrast()`

Other emmeans tidiers: `tidy.emmGrid()`, `tidy.ref.grid()`, `tidy.summary_emm()`

**Examples**

```
library(emmeans)
# linear model for sales of oranges per day
oranges_lm1 <- lm(sales1 ~ price1 + price2 + day + store, data = oranges)

# reference grid; see vignette("basics", package = "emmeans")
oranges_rg1 <- ref_grid(oranges_lm1)
td <- tidy(oranges_rg1)
td

# marginal averages
marginal <- emmeans(oranges_rg1, "day")
tidy(marginal)

# contrasts
tidy(contrast(marginal))
tidy(contrast(marginal, method = "pairwise"))

# plot confidence intervals
library(ggplot2)
ggplot(tidy(marginal, conf.int = TRUE), aes(day, estimate)) +
  geom_point() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# by multiple prices
by_price <- emmeans(oranges_lm1, "day",
  by = "price2",
  at = list(
```

```

      price1 = 50, price2 = c(40, 60, 80),
      day = c("2", "3", "4")
    )
  )
  by_price
  tidy(by_price)

ggplot(tidy(by_price, conf.int = TRUE), aes(price2, estimate, color = day)) +
  geom_line() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# joint_tests
tidy(joint_tests(oranges_lm1))

```

---

tidy.manova

*Tidy a(n) manova object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'manova'
tidy(x, test = "Pillai", ...)
```

## Arguments

x	A manova object return from <code>stats::manova()</code> .
test	One of "Pillai" (Pillai's trace), "Wilks" (Wilk's lambda), "Hotelling-Lawley" (Hotelling-Lawley trace) or "Roy" (Roy's greatest root) indicating which test statistic should be used. Defaults to "Pillai".
...	Arguments passed on to <code>stats::summary.manova</code>
object	An object of class "manova" or an aov object with multiple responses.
intercept	logical. If TRUE, the intercept term is included in the table.
tol	tolerance to be used in deciding if the residuals are rank-deficient: see <a href="#">qr</a> .

## Details

Depending on which test statistic is specified only one of `pillai`, `wilks`, `h1` or `roy` is included.

**Value**

A `tibble::tibble()` with columns:

<code>den.df</code>	Degrees of freedom of the denominator.
<code>num.df</code>	Degrees of freedom.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>term</code>	The name of the regression term.
<code>pillai</code>	Pillai's trace.
<code>wilks</code>	Wilk's lambda.
<code>h1</code>	Hotelling-Lawley trace.
<code>roy</code>	Roy's greatest root.

**See Also**

`tidy()`, `stats::summary.manova()`

Other anova tidiers: `glance.aov()`, `tidy.TukeyHSD()`, `tidy.anova()`, `tidy.aovlist()`, `tidy.aov()`

**Examples**

```
npk2 <- within(npk, foo <- rnorm(24))
m <- manova(cbind(yield, foo) ~ block + N * P * K, npk2)
tidy(m)
```

---

tidy.map

*Tidy a(n) map object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'map'
tidy(x, ...)
```



**Arguments**

x	A map object returned from <code>maps::map()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

term	The name of the regression term.
long	Longitude.
lat	Latitude.

Remaining columns give information on geographic attributes and depend on the inputted map object. See `?maps::map` for more information.

**See Also**

`tidy()`, `maps::map()`

**Examples**

```
library(maps)
library(ggplot2)

ca <- map("county", "ca", plot = FALSE, fill = TRUE)
tidy(ca)
qplot(long, lat, data = ca, geom = "polygon", group = group)

tx <- map("county", "texas", plot = FALSE, fill = TRUE)
tidy(tx)
qplot(long, lat,
      data = tx, geom = "polygon", group = group,
      colour = I("white"))
)
```

---

tidy.margins	<i>Tidy a(n) margins object</i>
--------------	---------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'margins'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

## Arguments

x	A margins object returned from <code>margins::margins()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

The margins package provides a way to obtain coefficient marginal effects for a variety of (non-linear) models, such as logit or models with multiway interaction terms. Note that the `glance.margins()` method requires rerunning the underlying model again, which can take some time. Similarly, an `augment.margins()` method is not currently supported, but users can simply run the underlying model to obtain the same information.

## Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.

p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

[tidy\(\)](#), [margins::margins\(\)](#)

**Examples**

```
library(margins)

## Example 1: Logit model ##

mod_log <- glm(am ~ cyl + hp + wt, data = mtcars, family = binomial)

# Get tidied "naive" model coefficients
tidy(mod_log)

# Convert to marginal effects with margins::margins()
marg_log <- margins(mod_log)

# Get tidied marginal effects
tidy(marg_log)
tidy(marg_log, conf.int = TRUE)
glance(marg_log) ## Requires running the underlying model again. Quick for this example.
## Not run: augment(marg_log) ## Not supported.
augment(mod_log) ## But can get the same info by running on the underlying model.

## Example 2: Threeway interaction terms ##

mod_ie <- lm(mpg ~ wt * cyl * disp, data = mtcars)

# Get tidied "naive" model coefficients
tidy(mod_ie)

# Convert to marginal effects with margins::margins()
marg_ie0 <- margins(mod_ie)
# Get tidied marginal effects
tidy(marg_ie0)
glance(marg_ie0)

# Marginal effects evaluated at specific values of a variable (here: cyl)
marg_ie1 <- margins(mod_ie, at = list(cyl = c(4,6,8)))
tidy(marg_ie1)

# Marginal effects of one interaction variable (here: wt), modulated at
# specific values of the two other interaction variables (here: cyl and drat)
marg_ie2 <- margins(mod_ie,
```

```

variables = "wt", ## Main var
at = list(cyl = c(4,6,8), drat = c(3, 3.5, 4)) ## Modulating vars
tidy(marg_ie2)

```

---

tidy.Mclust	<i>Tidy a(n) Mclust object</i>
-------------	--------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'Mclust'
tidy(x, ...)

```

## Arguments

x	An Mclust object return from <code>mclust::Mclust()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

proportion	The mixing proportion of each component
size	Number of points assigned to cluster.
mean	The mean for each component. In case of 2+ dimensional models, a column with the mean is added for each dimension. NA for noise component
variance	In case of one-dimensional and spherical models, the variance for each component, omitted otherwise. NA for noise component
component	Cluster id as a factor.

## See Also

`tidy()`, `mclust::Mclust()`

Other mclust tidiers: `augment.Mclust()`

**Examples**

```

library(dplyr)
library(mclust)
set.seed(27)

centers <- tibble::tibble(
  cluster = factor(1:3),
  num_points = c(100, 150, 50), # number points in each cluster
  x1 = c(5, 0, -3), # x1 coordinate of cluster center
  x2 = c(-1, 1, -2) # x2 coordinate of cluster center
)

points <- centers %>%
  mutate(
    x1 = purrr::map2(num_points, x1, rnorm),
    x2 = purrr::map2(num_points, x2, rnorm)
  ) %>%
  dplyr::select(-num_points, -cluster) %>%
  tidyr::unnest(c(x1, x2))

m <- mclust::Mclust(points)

tidy(m)
augment(m, points)
glance(m)

```

tidy.mediate

*Tidy a(n) mediate object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```

## S3 method for class 'mediate'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

```

**Arguments**

x	A mediate object produced by a call to <code>mediation::mediate()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.

<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

The tibble has four rows. The first two indicate the mediated effect in the control and treatment groups, respectively. And the last two the direct effect in each group.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

## See Also

`tidy()`, `mediation::mediate()`

## Examples

```
library(mediation)
data(jobs)

b <- lm(job_seek ~ treat + econ_hard + sex + age, data = jobs)
c <- lm(depress2 ~ treat + job_seek + econ_hard + sex + age, data = jobs)
mod <- mediate(b, c, sims = 50, treat = "treat", mediator = "job_seek")

tidy(mod)
tidy(mod, conf.int = TRUE)
tidy(mod, conf.int = TRUE, conf.level = .99)
```

tidy.mfx

*Tidy a(n) mfx object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

The particular functions below provide generic tidy methods for objects returned by the `mfx` package, preserving the calculated marginal effects instead of the naive model coefficients. The returned tidy tibble will also include an additional "atmean" column indicating how the marginal effects were originally calculated (see Details below).

**Usage**

```
## S3 method for class 'mfx'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

```
## S3 method for class 'logitmfx'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

```
## S3 method for class 'negbinmfx'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

```
## S3 method for class 'poissonmfx'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

```
## S3 method for class 'probitmfx'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	A <code>logitmfx</code> , <code>negbinmfx</code> , <code>poissonmfx</code> , or <code>probitmfx</code> object. (Note that <code>betamfx</code> objects receive their own set of tidiers.)
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

The `mfx` package provides methods for calculating marginal effects for various generalized linear models (GLMs). Unlike standard linear models, estimated model coefficients in a GLM cannot be directly interpreted as marginal effects (i.e., the change in the response variable predicted after a one unit change in one of the regressors). This is because the estimated coefficients are multiplicative, dependent on both the link function that was used for the estimation and any other variables that were included in the model. When calculating marginal effects, users must typically choose whether they want to use i) the average observation in the data, or ii) the average of the sample marginal effects. See `vignette("mfxarticle")` from the `mfx` package for more details.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.
<code>atmean</code>	TRUE if the marginal effects were originally calculated as the partial effects for the average observation. If FALSE, then these were instead calculated as average partial effects.

## See Also

`tidy()`, `mfx::logitmfx()`, `mfx::negbinmfx()`, `mfx::poissonmfx()`, `mfx::probitmfx()`

Other `mfx` tidiers: `augment.betamfx()`, `augment.mfx()`, `glance.betamfx()`, `glance.mfx()`, `tidy.betamfx()`

## Examples

```
## Not run:
library(mfx)

## Get the marginal effects from a logit regression
mod_logmfx <- logitmfx(am ~ cyl + hp + wt, atmean = TRUE, data = mtcars)
tidy(mod_logmfx, conf.int = TRUE)

## Compare with the naive model coefficients of the same logit call (not run)
# tidy(glm(am ~ cyl + hp + wt, family = binomial, data = mtcars), conf.int = TRUE)

augment(mod_logmfx)
glance(mod_logmfx)

## Another example, this time using probit regression
```



```

mod_probmfx <- probitmfx(am ~ cyl + hp + wt, atmean = TRUE, data = mtcars)
tidy(mod_probmfx, conf.int = TRUE)
augment(mod_probmfx)
glance(mod_probmfx)

## End(Not run)

```

---

tidy.mjoint

*Tidy a(n) mjoint object*


---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```

## S3 method for class 'mjoint'
tidy(
  x,
  component = "survival",
  conf.int = FALSE,
  conf.level = 0.95,
  boot_se = NULL,
  ...
)

```

### Arguments

x	An mjoint object returned from <code>joineRML::mjoint()</code> .
component	Character specifying whether to tidy the survival or the longitudinal component of the model. Must be either "survival" or "longitudinal". Defaults to "survival".
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
boot_se	Optionally a bootSE object from <code>joineRML::bootSE()</code> . If specified, calculates confidence intervals via the bootstrap. Defaults to NULL, in which case standard errors are calculated from the empirical information matrix.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be

used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

### See Also

`tidy()`, `joineRML::mjoint()`, `joineRML::bootSE()`

Other mjoint tidiers: `glance.mjoint()`

### Examples

```
## Not run:
# Fit a joint model with bivariate longitudinal outcomes
library(joineRML)
data(heart.valve)
hvd <- heart.valve[!is.na(heart.valve$log.grad) &
  !is.na(heart.valve$log.lvmi) &
  heart.valve$num <= 50, ]
fit <- mjoint(
  formLongFixed = list(
    "grad" = log.grad ~ time + sex + hs,
    "lvmi" = log.lvmi ~ time + sex
  ),
  formLongRandom = list(
    "grad" = ~ 1 | num,
    "lvmi" = ~ time | num
  ),
  formSurv = Surv(fuyrs, status) ~ age,
  data = hvd,
  inits = list("gamma" = c(0.11, 1.51, 0.80)),
  timeVar = "time"
)

# Extract the survival fixed effects
tidy(fit)
```

```

# Extract the longitudinal fixed effects
tidy(fit, component = "longitudinal")

# Extract the survival fixed effects with confidence intervals
tidy(fit, ci = TRUE)

# Extract the survival fixed effects with confidence intervals based
# on bootstrapped standard errors
bSE <- bootSE(fit, nboot = 5, safe.boot = TRUE)
tidy(fit, boot_se = bSE, ci = TRUE)

# Augment original data with fitted longitudinal values and residuals
hvd2 <- augment(fit)

# Extract model statistics
glance(fit)

## End(Not run)

```

---

tidy.mle2

*Tidy a(n) mle2 object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'mle2'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)

```

## Arguments

x	An mle2 object created by a call to <code>bbmle::mle2()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed

using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

### See Also

`tidy()`, `bbmle::mle2()`, `tidy_optim()`

### Examples

```
library(bbmle)

x <- 0:10
y <- c(26, 17, 13, 12, 20, 5, 9, 8, 5, 4, 8)
d <- data.frame(x, y)

fit <- mle2(y ~ dpois(lambda = ymean),
  start = list(ymean = mean(y)), data = d
)

tidy(fit)
```

---

tidy.mlm

*Tidy a(n) mlm object*

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'mlm'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	An <code>mlm</code> object created by <code>stats::lm()</code> with a matrix as the response.
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

In contrast to `lm` object (simple linear model), tidy output for `mlm` (multiple linear model) objects contain an additional column response.

If you have missing values in your model data, you may need to refit the model with `na.action = na.exclude`.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

[tidy\(\)](#)

Other `lm` tidiers: [augment.glm\(\)](#), [augment.lm\(\)](#), [glance.glm\(\)](#), [glance.lm\(\)](#), [glance.summary.lm\(\)](#), [glance.svyglm\(\)](#), [tidy.glm\(\)](#), [tidy.lm.beta\(\)](#), [tidy.lm\(\)](#), [tidy.summary.lm\(\)](#)

**Examples**

```
mod <- lm(cbind(mpg, disp) ~ wt, mtcars)
tidy(mod, conf.int = TRUE)
```

tidy.mlogit

*Tidying methods for logit models***Description**

These methods tidy the coefficients of mnl and nl models generated by the functions of the `mlogit` package.

**Usage**

```
## S3 method for class 'mlogit'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	an object returned from <code>mlogit::mlogit()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

[tidy\(\)](#), [mlogit::mlogit\(\)](#)

Other mlogit tidiers: [augment.mlogit\(\)](#), [glance.mlogit\(\)](#)

**Examples**

```
## Not run:
library(mlogit)
data("Fishing", package = "mlogit")
Fish <- dfidx(Fishing, varying = 2:9, shape = "wide", choice = "mode")
m <- mlogit(mode ~ price + catch | income, data = Fish)

tidy(m)
augment(m)
glance(m)

## End(Not run)
```

---

tidy.muhaz

*Tidy a(n) muhaz object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'muhaz'
tidy(x, ...)
```

**Arguments**

**x** A muhaz object returned by [muhaz::muhaz\(\)](#).

**...** Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in **...**, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an [augment\(\)](#) method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with columns:

time	Point in time.
estimate	Estimated hazard rate.

**See Also**

`tidy()`, `muhaz::muhaz()`

Other muhaz tidiers: `glance.muhaz()`

**Examples**

```
library(muhaz)

data(ovarian, package = "survival")
x <- muhaz::muhaz(ovarian$futime, ovarian$fustat)
tidy(x)
glance(x)
```

---

tidy.multinom

*Tidying methods for multinomial logistic regression models*

---

**Description**

These methods tidy the coefficients of multinomial logistic regression models generated by `multinom` of the `nnet` package.

**Usage**

```
## S3 method for class 'multinom'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)
```

**Arguments**

x	A <code>multinom</code> object returned from <code>nnet::multinom()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.



... Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.
<code>y.value</code>	The response level.

## See Also

`tidy()`, `nnet::multinom()`

Other multinom tidiers: `glance.multinom()`

## Examples

```
library(nnet)
library(MASS)

example(birthwt)
bwt.mu <- multinom(low ~ ., bwt)
tidy(bwt.mu)
glance(bwt.mu)

#* This model is a truly terrible model
#* but it should show you what the output looks
#* like in a multinomial logistic regression

fit.gear <- multinom(gear ~ mpg + factor(am), data = mtcars)
tidy(fit.gear)
glance(fit.gear)
```

tidy.nlrq

*Tidy a(n) nlrq object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'nlrq'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	A nlrq object returned from <code>quantreg::nlrq()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

[tidy\(\)](#), [quantreg::nlrq\(\)](#)

Other quantreg tidiers: [augment.nlrq\(\)](#), [augment.rqs\(\)](#), [augment.rq\(\)](#), [glance.nlrq\(\)](#), [glance.rq\(\)](#), [tidy.rqs\(\)](#), [tidy.rq\(\)](#)

---

tidy.nls

*Tidy a(n) nls object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'nls'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	An nls object returned from <a href="#">stats::nls()</a> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <a href="#">augment()</a> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A [tibble::tibble\(\)](#) with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.

statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

[tidy](#), [stats::nls\(\)](#), [stats::summary.nls\(\)](#)

Other nls tidiers: [augment.nls\(\)](#), [glance.nls\(\)](#)

**Examples**

```
n <- nls(mpg ~ k * e^wt, data = mtcars, start = list(k = 1, e = 2))

tidy(n)
augment(n)
glance(n)

library(ggplot2)
ggplot(augment(n), aes(wt, mpg)) +
  geom_point() +
  geom_line(aes(y = .fitted))

newdata <- head(mtcars)
newdata$wt <- newdata$wt + 1
augment(n, newdata = newdata)
```

---

tidy.numeric

*Tidy atomic vectors*

---

**Description**

Vector tidiers are deprecated and will be removed from an upcoming release of broom.

**Usage**

```
## S3 method for class 'numeric'
tidy(x, ...)

## S3 method for class 'character'
tidy(x, ...)

## S3 method for class 'logical'
tidy(x, ...)
```

**Arguments**

x	An object of class "numeric", "integer", "character", or "logical". Most likely a named vector
...	Extra arguments (not used)

**Details**

Turn atomic vectors into data frames, where the names of the vector (if they exist) are a column and the values of the vector are a column.

**See Also**

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [data.frame\\_tidiers](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#)

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [data.frame\\_tidiers](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#)

Other deprecated: [bootstrap\(\)](#), [confint\\_tidy\(\)](#), [data.frame\\_tidiers](#), [finish\\_glance\(\)](#), [fix\\_data\\_frame\(\)](#), [summary\\_tidiers](#), [tidy.density\(\)](#), [tidy.dist\(\)](#), [tidy.ftable\(\)](#), [tidy.gamlss\(\)](#)

**Examples**

```
## Not run:
x <- 1:5
names(x) <- letters[1:5]
tidy(x)

## End(Not run)
```

---

tidy.orcutt

*Tidy a(n) orcutt object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'orcutt'
tidy(x, ...)
```

**Arguments**

x	An orcutt object returned from <code>orcutt::cochrane.orcutt()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

`orcutt::cochrane.orcutt()`

Other orcutt tidiers: `glance.orcutt()`

**Examples**

```
library(orcutt)

reg <- lm(mpg ~ wt + qsec + disp, mtcars)
tidy(reg)

co <- cochrane.orcutt(reg)
co

tidy(co)
glance(co)
```

---

tidy.pairwise.htest *Tidy a(n) pairwise.htest object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'pairwise.htest'
tidy(x, ...)
```

## Arguments

x	A pairwise.htest object such as those returned from <code>stats::pairwise.t.test()</code> or <code>stats::pairwise.wilcox.test()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

Note that in one-sided tests, the alternative hypothesis of each test can be stated as "group1 is greater/less than group2".

Note also that the columns of `group1` and `group2` will always be a factor, even if the original input is (e.g.) numeric.

## Value

A `tibble::tibble()` with columns:

group1	First group being compared.
group2	Second group being compared.
p.value	The two-sided p-value associated with the observed statistic.

## See Also

`stats::pairwise.t.test()`, `stats::pairwise.wilcox.test()`, `tidy()`

Other htest tidiers: `augment.htest()`, `tidy.htest()`, `tidy.power.htest()`

## Examples

```
attach(airquality)
Month <- factor(Month, labels = month.abb[5:9])
ptt <- pairwise.t.test(Ozone, Month)
tidy(ptt)

library(modeldata)
data(hpc_data)
attach(hpc_data)
ptt2 <- pairwise.t.test(compounds, class)
tidy(ptt2)

tidy(pairwise.t.test(compounds, class, alternative = "greater"))
tidy(pairwise.t.test(compounds, class, alternative = "less"))

tidy(pairwise.wilcox.test(compounds, class))
```

---

tidy.pam

*Tidy a(n) pam object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'pam'
tidy(x, col.names = paste0("x", 1:ncol(x$medoids)), ...)
```

## Arguments

x	An pam object returned from <code>cluster::pam()</code>
col.names	Column names in the input data frame. Defaults to the names of the variables in x.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.



## Details

For examples, see the pam vignette.

## Value

A `tibble::tibble()` with columns:

size	Size of each cluster.
max.diss	Maximal dissimilarity between the observations in the cluster and that cluster's medoid.
avg.diss	Average dissimilarity between the observations in the cluster and that cluster's medoid.
diameter	Diameter of the cluster.
separation	Separation of the cluster.
avg.width	Average silhouette width of the cluster.
cluster	A factor describing the cluster from 1:k.

## See Also

`tidy()`, `cluster::pam()`

Other pam tidiers: `augment.pam()`, `glance.pam()`

## Examples

```
## Not run:
library(dplyr)
library(ggplot2)
library(cluster)
library(modeldata)
data(hpc_data)

x <- hpc_data[, 2:5]
p <- pam(x, k = 4)

tidy(p)
glance(p)
augment(p, x)

augment(p, x) %>%
  ggplot(aes(compounds, input_fields)) +
  geom_point(aes(color = .cluster)) +
  geom_text(aes(label = cluster), data = tidy(p), size = 10)

## End(Not run)
```

tidy.plm

*Tidy a(n) plm object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'plm'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	A plm object returned by <code>plm::plm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

`tidy()`, `plm::plm()`, `tidy.lm()`

Other plm tidiers: `augment.plm()`, `glance.plm()`

**Examples**

```
library(plm)

data("Produc", package = "plm")
zz <- plm(log(gsp) ~ log(pcap) + log(pc) + log(emp) + unemp,
  data = Produc, index = c("state", "year")
)

summary(zz)

tidy(zz)
tidy(zz, conf.int = TRUE)
tidy(zz, conf.int = TRUE, conf.level = 0.9)

augment(zz)
glance(zz)
```

---

tidy.poLCA

*Tidy a(n) poLCA object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'poLCA'
tidy(x, ...)
```

**Arguments**

`x` A poLCA object returned from `poLCA::poLCA()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>class</code>	The class under consideration.
<code>outcome</code>	Outcome of manifest variable.
<code>std.error</code>	The standard error of the regression term.
<code>variable</code>	Manifest variable
<code>estimate</code>	Estimated class-conditional response probability

**See Also**

[tidy\(\)](#), [poLCA::poLCA\(\)](#)

Other poLCA tidiers: [augment.poLCA\(\)](#), [glance.poLCA\(\)](#)

**Examples**

```
library(poLCA)
library(dplyr)

data(values)
f <- cbind(A, B, C, D) ~ 1
M1 <- poLCA(f, values, nclass = 2, verbose = FALSE)

M1
tidy(M1)
augment(M1)
glance(M1)

library(ggplot2)

ggplot(tidy(M1), aes(factor(class), estimate, fill = factor(outcome))) +
  geom_bar(stat = "identity", width = 1) +
  facet_wrap(~variable)
## Three-class model with a single covariate.

data(election)
f2a <- cbind(
  MORALG, CARESG, KNOWG, LEADG, DISHONG, INTELG,
  MORALB, CARESB, KNOWB, LEADB, DISHONB, INTELB
) ~ PARTY
nes2a <- poLCA(f2a, election, nclass = 3, nrep = 5, verbose = FALSE)

td <- tidy(nes2a)
td

# show

ggplot(td, aes(outcome, estimate, color = factor(class), group = class)) +
  geom_line() +
```

```

  facet_wrap(~variable, nrow = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

au <- augment(nes2a)
au
count(au, .class)

# if the original data is provided, it leads to NAs in new columns
# for rows that weren't predicted
au2 <- augment(nes2a, data = election)
au2
dim(au2)

```

---

tidy.polr

*Tidy a(n) polr object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'polr'
tidy(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  exponentiate = FALSE,
  p.values = FALSE,
  ...
)

```

## Arguments

<code>x</code>	A polr object returned from <code>MASS::polr()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>exponentiate</code>	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.

<code>p.values</code>	Logical. Should p-values be returned, based on chi-squared tests from <code>MASS::dropterm()</code> . Defaults to FALSE.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

### Details

In broom 0.7.0 the `coefficient_type` column was renamed to `coef.type`, and the contents were changed as well. Now the contents are coefficient and scale, rather than coefficient and zeta.

Calculating p-values with the `dropterm()` function is the approach suggested by the MASS package author <https://r.789695.n4.nabble.com/p-values-of-por-td4668100.html>. This approach is computationally intensive so that p-values are only returned if requested explicitly. Additionally, it only works for models containing no variables with more than two categories. If this condition is not met, a message is shown and NA is returned instead of p-values.

### Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

### See Also

`tidy`, `MASS::polr()`

Other ordinal tidiers: `augment.clm()`, `augment.polr()`, `glance.clmm()`, `glance.clm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clmm()`, `tidy.clm()`, `tidy.svyolr()`

### Examples

```
library(MASS)
```

```
fit <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)
```

```

tidy(fit, exponentiate = TRUE, conf.int = TRUE)

glance(fit)
augment(fit, type.predict = "class")

fit2 <- polr(factor(gear) ~ am + mpg + qsec, data = mtcars)
tidy(fit, p.values = TRUE)

```

---

tidy.power.htest	<i>Tidy a(n) power.htest object</i>
------------------	-------------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'power.htest'
tidy(x, ...)

```

## Arguments

x	A power.htest object such as those returned from <code>stats::power.t.test()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with columns:

delta	True difference in means.
n	Number of observations by component.
power	Power achieved for given value of n.
sd	Standard deviation.
sig.level	Significance level (Type I error probability).

**See Also**

`stats::power.t.test()`

Other htest tidiers: `augment.htest()`, `tidy.htest()`, `tidy.pairwise.htest()`

**Examples**

```
ptt <- power.t.test(n = 2:30, delta = 1)
tidy(ptt)

library(ggplot2)

ggplot(tidy(ptt), aes(n, power)) +
  geom_line()
```

---

tidy.prcomp

*Tidy a(n) prcomp object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'prcomp'
tidy(x, matrix = "u", ...)
```

**Arguments**

<code>x</code>	A prcomp object returned by <code>stats::prcomp()</code> .
<code>matrix</code>	Character specifying which component of the PCA should be tidied. <ul style="list-style-type: none"> <li>• "u", "samples", "scores", or "x": returns information about the map from the original space into principle components space.</li> <li>• "v", "rotation", "loadings" or "variables": returns information about the map from principle components space back into the original space.</li> <li>• "d", "eigenvalues" or "pcs": returns information about the eigenvalues.</li> </ul>
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.



**Details**

See <https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca> for information on how to interpret the various tidied matrices. Note that SVD is only equivalent to PCA on centered data.

**Value**

A `tibble::tibble` with columns depending on the component of PCA being tidied.

If `matrix` is "u", "samples", "scores", or "x" each row in the tidied output corresponds to the original data in PCA space. The columns are:

row	ID of the original observation (i.e. rowname from original data).
PC	Integer indicating a principal component.
value	The score of the observation for that particular principal component. That is, the location of the observation in PCA space.

If `matrix` is "v", "rotation", "loadings" or "variables", each row in the tidied output corresponds to information about the principle components in the original space. The columns are:

row	The variable labels (colnames) of the data set on which PCA was performed
PC	An integer vector indicating the principal component
value	The value of the eigenvector (axis score) on the indicated principal component

If `matrix` is "d", "eigenvalues" or "pcs", the columns are:

PC	An integer vector indicating the principal component
std.dev	Standard deviation explained by this PC
percent	Fraction of variation explained by this component (a numeric value between 0 and 1).
cumulative	Cumulative fraction of variation explained by principle components up to this component (a numeric value between 0 and 1).

**See Also**

`stats::prcomp()`, `svd_tidiers`

Other svd tidiers: `augment.prcomp()`, `tidy_irlba()`, `tidy_svd()`

**Examples**

```
pc <- prcomp(USArrests, scale = TRUE)

# information about rotation
tidy(pc)

# information about samples (states)
tidy(pc, "samples")
```

```

# information about PCs
tidy(pc, "pcs")

# state map
library(dplyr)
library(ggplot2)

pc %>%
  tidy(matrix = "samples") %>%
  mutate(region = tolower(row)) %>%
  inner_join(map_data("state"), by = "region") %>%
  ggplot(aes(long, lat, group = group, fill = value)) +
  geom_polygon() +
  facet_wrap(~PC) +
  theme_void() +
  ggtitle("Principal components of arrest data")

au <- augment(pc, data = USArrests)
au

ggplot(au, aes(.fittedPC1, .fittedPC2)) +
  geom_point() +
  geom_text(aes(label = .rownames), vjust = 1, hjust = 1)

```

---

tidy.pyyears

*Tidy a(n) pyyears object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'pyyears'
tidy(x, ...)

```

## Arguments

x	A pyyears object returned from <code>survival::pyyears()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

expected is only present in the output when if a ratetable term is present.

If the `data.frame = TRUE` argument is supplied to `pyears`, this is simply the contents of `x$data`.

**Value**

A `tibble::tibble()` with columns:

expected	Expected number of events.
pyears	Person-years of exposure.
n	number of subjects contributing time
event	observed number of events

**See Also**

[tidy\(\)](#), [survival::pyears\(\)](#)

Other pyears tidiers: [glance.pyears\(\)](#)

Other survival tidiers: [augment.coxph\(\)](#), [augment.survreg\(\)](#), [glance.aareg\(\)](#), [glance.cch\(\)](#), [glance.coxph\(\)](#), [glance.pyears\(\)](#), [glance.survdiff\(\)](#), [glance.survexp\(\)](#), [glance.survfit\(\)](#), [glance.survreg\(\)](#), [tidy.aareg\(\)](#), [tidy.cch\(\)](#), [tidy.coxph\(\)](#), [tidy.survdiff\(\)](#), [tidy.survexp\(\)](#), [tidy.survfit\(\)](#), [tidy.survreg\(\)](#)

**Examples**

```
library(survival)

temp.yr <- tcut(mgus$dxyr, 55:92, labels = as.character(55:91))
temp.age <- tcut(mgus$age, 34:101, labels = as.character(34:100))
ptime <- ifelse(is.na(mgus$pctime), mgus$futime, mgus$pctime)
pstat <- ifelse(is.na(mgus$pctime), 0, 1)
pfit <- pyears(Surv(ptime / 365.25, pstat) ~ temp.yr + temp.age + sex, mgus,
  data.frame = TRUE
)
tidy(pfit)
glance(pfit)

# if data.frame argument is not given, different information is present in
# output
pfit2 <- pyears(Surv(ptime / 365.25, pstat) ~ temp.yr + temp.age + sex, mgus)
tidy(pfit2)
glance(pfit2)
```

tidy.rcorr

*Tidy a(n) rcorr object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'rcorr'
tidy(x, diagonal = FALSE, ...)
```

**Arguments**

x	An rcorr object returned from <code>Hmisc::rcorr()</code> .
diagonal	Logical indicating whether or not to include diagonal elements of the correlation matrix, or the correlation of a column with itself. For the elements, estimate is always 1 and p.value is always NA. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Details**

Suppose the original data has columns A and B. In the correlation matrix from `rcorr` there may be entries for both the `cor(A,B)` and `cor(B,A)`. Only one of these pairs will ever be present in the tidy output.

**Value**

A `tibble::tibble()` with columns:

column1	Name or index of the first column being described.
column2	Name or index of the second column being described.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
n	Number of observations used to compute the correlation

**See Also**

[tidy\(\)](#), [Hmisc::rcorr\(\)](#)

**Examples**

```
library(Hmisc)

mat <- replicate(52, rnorm(100))
# add some NAs
mat[sample(length(mat), 2000)] <- NA
# also column names
colnames(mat) <- c(LETTERS, letters)

rc <- rcorr(mat)

td <- tidy(rc)
td

library(ggplot2)
ggplot(td, aes(p.value)) +
  geom_histogram(binwidth = .1)

ggplot(td, aes(estimate, p.value)) +
  geom_point() +
  scale_y_log10()
```

---

tidy.ref.grid

*Tidy a(n) ref.grid object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'ref.grid'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

**x** A ref.grid object created by [emmeans::ref\\_grid\(\)](#).

**conf.int** Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.

<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments passed to <code>emmeans::summary.emmGrid()</code> or <code>lsmeans::summary.ref.grid()</code> . <b>Cautionary note:</b> misspecified arguments may be silently ignored!

### Details

Returns a data frame with one observation for each estimated marginal mean, and one column for each combination of factors. When the input is a contrast, each row will contain one estimated contrast.

There are a large number of arguments that can be passed on to `emmeans::summary.emmGrid()` or `lsmeans::summary.ref.grid()`.

### Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>df</code>	Degrees of freedom used by this term in the model.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>std.error</code>	The standard error of the regression term.
<code>estimate</code>	Expected marginal mean
<code>statistic</code>	T-ratio statistic

### See Also

`tidy()`, `emmeans::ref_grid()`, `emmeans::emmeans()`, `emmeans::contrast()`

Other emmeans tidiers: `tidy.emmGrid()`, `tidy.lsmobj()`, `tidy.summary_emm()`

### Examples

```
library(emmeans)
# linear model for sales of oranges per day
oranges_lm1 <- lm(sales1 ~ price1 + price2 + day + store, data = oranges)

# reference grid; see vignette("basics", package = "emmeans")
oranges_rg1 <- ref_grid(oranges_lm1)
td <- tidy(oranges_rg1)
td

# marginal averages
marginal <- emmeans(oranges_rg1, "day")
tidy(marginal)

# contrasts
```

```

tidy(contrast(marginal))
tidy(contrast(marginal, method = "pairwise"))

# plot confidence intervals
library(ggplot2)
ggplot(tidy(marginal, conf.int = TRUE), aes(day, estimate)) +
  geom_point() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# by multiple prices
by_price <- emmeans(oranges_lm1, "day",
  by = "price2",
  at = list(
    price1 = 50, price2 = c(40, 60, 80),
    day = c("2", "3", "4")
  )
)
by_price
tidy(by_price)

ggplot(tidy(by_price, conf.int = TRUE), aes(price2, estimate, color = day)) +
  geom_line() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# joint_tests
tidy(joint_tests(oranges_lm1))

```

---

tidy.regsubsets	<i>Tidy a(n) regsubsets object</i>
-----------------	------------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'regsubsets'
tidy(x, ...)

```

## Arguments

x	A regsubsets object created by <code>leaps::regsubsets()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed

using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble()` with columns:

<code>r.squared</code>	R squared statistic, or the percent of variation explained by the model.
<code>adj.r.squared</code>	Adjusted R squared statistic
<code>BIC</code>	Bayesian information criterion for the component.
<code>mallows_cp</code>	Mallow's Cp statistic.

### See Also

`tidy()`, `leaps::regsubsets()`

### Examples

```
all_fits <- leaps::regsubsets(hp ~ ., mtcars)
tidy(all_fits)
```

---

<code>tidy.ridgelm</code>	<i>Tidy a(n) ridgelm object</i>
---------------------------	---------------------------------

---

### Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

### Usage

```
## S3 method for class 'ridgelm'
tidy(x, ...)
```

### Arguments

<code>x</code>	A <code>ridgelm</code> object returned from <code>MASS::lm.ridge()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.



**Value**

A `tibble::tibble()` with columns:

GCV	Generalized cross validation error estimate.
lambda	Value of penalty parameter lambda.
term	The name of the regression term.
estimate	estimate of scaled coefficient using this lambda
scale	Scaling factor of estimated coefficient

**See Also**

`tidy()`, `MASS::lm.ridge()`

Other `ridgelm` tidiers: `glance.ridgelm()`

**Examples**

```
names(longley)[1] <- "y"
fit1 <- MASS::lm.ridge(y ~ ., longley)
tidy(fit1)

fit2 <- MASS::lm.ridge(y ~ ., longley, lambda = seq(0.001, .05, .001))
td2 <- tidy(fit2)
g2 <- glance(fit2)

# coefficient plot
library(ggplot2)
ggplot(td2, aes(lambda, estimate, color = term)) +
  geom_line()

# GCV plot
ggplot(td2, aes(lambda, GCV)) +
  geom_line()

# add line for the GCV minimizing estimate
ggplot(td2, aes(lambda, GCV)) +
  geom_line() +
  geom_vline(xintercept = g2$lambdaGCV, col = "red", lty = 2)
```

---

tidy.rlm

*Tidy a(n) rlm object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'rlm'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	An <code>rlm</code> object returned by <code>MASS::rlm()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**See Also**

[MASS::rlm\(\)](#)

Other `rlm` tidiers: [augment.rlm\(\)](#), [glance.rlm\(\)](#)

---

tidy.rma

*Tidy a(n) rma object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'rma'
tidy(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  exponentiate = FALSE,
  include_studies = FALSE,
  measure = "GEN",
  ...
)
```

**Arguments**

<code>x</code>	An rma object such as those created by <code>metafor::rma()</code> , <code>metafor::rma.uni()</code> , <code>metafor::rma.glmm()</code> , <code>metafor::rma.mh()</code> , <code>metafor::rma.mv()</code> , or <code>metafor::rma.peto()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>exponentiate</code>	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
<code>include_studies</code>	Logical. Should individual studies be included in the output? Defaults to FALSE.
<code>measure</code>	Measure type. See <code>metafor::escalc()</code>
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the individual study
<code>type</code>	The estimate type (summary vs individual study)

**Examples**

```
library(metafor)

df <-
  escalc(
    measure = "RR",
    ai = tpos,
```

```

    bi = tneg,
    ci = cpos,
    di = cneg,
    data = dat.bcg
  )

meta_analysis <- rma(yi, vi, data = df, method = "EB")

tidy(meta_analysis)

```

tidy.roc

*Tidy a(n) roc object***Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```

## S3 method for class 'roc'
tidy(x, ...)

```

**Arguments**

x	An roc object returned from a call to <code>AUC::roc()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

cutoff	The cutoff used for classification. Observations with predicted probabilities above this value were assigned class 1, and observations with predicted probabilities below this value were assigned class 0.
fpr	False positive rate.
tpr	The true positive rate at the given cutoff.

**See Also**

`tidy()`, `AUC::roc()`

**Examples**

```

library(AUC)
data(churn)
r <- roc(churn$predictions, churn$labels)

td <- tidy(r)
td

library(ggplot2)

ggplot(td, aes(fpr, tpr)) +
  geom_line()

# compare the ROC curves for two prediction algorithms

library(dplyr)
library(tidyr)

rocs <- churn %>%
  pivot_longer(contains("predictions"),
    names_to = "algorithm",
    values_to = "value"
  ) %>%
  nest(data = -algorithm) %>%
  mutate(tidy_roc = purrr::map(data, ~ tidy(roc(.x$value, .x$labels)))) %>%
  unnest(tidy_roc)

ggplot(rocs, aes(fpr, tpr, color = algorithm)) +
  geom_line()

```

---

tidy.rq

*Tidy a(n) rq object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```

## S3 method for class 'rq'
tidy(x, se.type = NULL, conf.int = FALSE, conf.level = 0.95, ...)

```

**Arguments**

<code>x</code>	An rq object returned from <code>quantreg::rq()</code> .
<code>se.type</code>	Character specifying the method to use to calculate standard errors. Passed to <code>quantreg::summary.rq()</code> <code>se</code> argument. Defaults to "rank" if the sample size is less than 1000, otherwise defaults to "nid".
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments passed to <code>quantreg::summary.rq()</code> .

**Details**

If `se.type = "rank"` confidence intervals are calculated by `summary.rq` and `statistic` and `p.value` values are not returned. When only a single predictor is included in the model, no confidence intervals are calculated and the confidence limits are set to NA.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `quantreg::rq()`

Other quantreg tidiers: `augment.nlrq()`, `augment.rqs()`, `augment.rq()`, `glance.nlrq()`, `glance.rq()`, `tidy.nlrq()`, `tidy.rqs()`

---

tidy.rqs	<i>Tidy a(n) rqs object</i>
----------	-----------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'rqs'
tidy(x, se.type = "rank", conf.int = FALSE, conf.level = 0.95, ...)
```

## Arguments

x	An rqs object returned from <a href="#">quantreg::rq()</a> .
se.type	Character specifying the method to use to calculate standard errors. Passed to <a href="#">quantreg::summary.rq()</a> se argument. Defaults to "rank".
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments passed to <a href="#">quantreg::summary.rqs()</a>

## Details

If `se.type = "rank"` confidence intervals are calculated by `summary.rq`. When only a single predictor is included in the model, no confidence intervals are calculated and the confidence limits are set to NA.

## Value

A [tibble::tibble\(\)](#) with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.
quantile	Linear conditional quantile.

**See Also**

[tidy\(\)](#), [quantreg::rq\(\)](#)

Other quantreg tidiers: [augment.nlrq\(\)](#), [augment.rqs\(\)](#), [augment.rq\(\)](#), [glance.nlrq\(\)](#), [glance.rq\(\)](#), [tidy.nlrq\(\)](#), [tidy.rq\(\)](#)

---

tidy.sarlm

*Tidying methods for spatially autoregressive models*

---

**Description**

These methods tidy the coefficients of spatial autoregression models generated by functions of the `spatialreg` package.

**Usage**

```
## S3 method for class 'sarlm'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

<code>x</code>	An object of object returned from <a href="#">spatialreg::lagsarlm()</a> or <a href="#">spatialreg::errorsarlm()</a> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <a href="#">augment()</a> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A [tibble::tibble\(\)](#) with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.



**See Also**

`tidy()`, `spatialreg::lagsarlm()`, `spatialreg::errorsarlm()`, `spatialreg::sacsarlm()`  
 Other spatialreg tidiers: `augment.sarlm()`, `glance.sarlm()`

**Examples**

```
## Not run:
library(spatialreg)
data(oldcol, package="spdep")
listw <- spdep::nb2listw(COL.nb, style="W")

crime_sar <- lagsarlm(CRIME ~ INC + HOVAL, data=COL.OLD,
  listw=listw, method="eigen")

tidy(crime_sar)
tidy(crime_sar, conf.int = TRUE)
glance(crime_sar)
augment(crime_sar)

crime_sem <- errorsarlm(CRIME ~ INC + HOVAL, data=COL.OLD, listw)

tidy(crime_sem)
tidy(crime_sem, conf.int = TRUE)
glance(crime_sem)
augment(crime_sem)

crime_sac <- sacsarlm(CRIME ~ INC + HOVAL, data=COL.OLD, listw)

tidy(crime_sac)
tidy(crime_sac, conf.int = TRUE)
glance(crime_sac)
augment(crime_sac)

## End(Not run)
```

---

tidy.spec

*Tidy a(n) spec object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'spec'
tidy(x, ...)
```

**Arguments**

x	A spec object created by <code>stats::spectrum()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

freq	Vector of frequencies at which the spectral density is estimated.
spec	Vector (for univariate series) or matrix (for multivariate series) of estimates of the spectral density at frequencies corresponding to freq.

**See Also**

`tidy()`, `stats::spectrum()`

Other time series tidiers: `tidy.acf()`, `tidy.ts()`, `tidy.zoo()`

**Examples**

```
spc <- spectrum(lh)
tidy(spc)

library(ggplot2)
ggplot(tidy(spc), aes(freq, spec)) +
  geom_line()
```

---

tidy.speedglm

*Tidy a(n) speedglm object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'speedglm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)
```

**Arguments**

<code>x</code>	A <code>speedglm</code> object returned from <code>speedglm::speedglm()</code> .
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>exponentiate</code>	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to <code>FALSE</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

[speedglm::speedglm\(\)](#)

Other `speedlm` tidiers: [augment.speedlm\(\)](#), [glance.speedglm\(\)](#), [glance.speedlm\(\)](#), [tidy.speedlm\(\)](#)

**Examples**

```
library(speedglm)

clotting <- data.frame(
  u = c(5, 10, 15, 20, 30, 40, 60, 80, 100),
  lot1 = c(118, 58, 42, 35, 27, 25, 21, 19, 18)
)
```

```
fit <- speedglm(lot1 ~ log(u), data = clotting, family = Gamma(log))

tidy(fit)
glance(fit)
```

---

tidy.speedlm	<i>Tidy a(n) speedlm object</i>
--------------	---------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'speedlm'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

## Arguments

x	A speedlm object returned from <code>speedglm::speedlm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

[speedglm::speedlm\(\)](#), [tidy.lm\(\)](#)

Other speedlm tidiers: [augment.speedlm\(\)](#), [glance.speedglm\(\)](#), [glance.speedlm\(\)](#), [tidy.speedglm\(\)](#)

**Examples**

```
mod <- speedglm::speedlm(mpg ~ wt + qsec, data = mtcars, fitted = TRUE)

tidy(mod)
glance(mod)
augment(mod)
```

---

`tidy.summary.glht`      *Tidy a(n) summary.glht object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'summary.glht'
tidy(x, ...)
```

**Arguments**

`x`                    A `summary.glht` object created by calling `multcomp::summary.glht()` on a `glht` object created with `multcomp::glht()`.

`...`                Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Value**

A `tibble::tibble()` with columns:

<code>contrast</code>	Levels being compared.
<code>estimate</code>	The estimated value of the regression term.
<code>null.value</code>	Value to which the estimate is compared.

p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.

**See Also**

`tidy()`, `multcomp::summary.glht()`, `multcomp::glht()`

Other multcomp tidiers: `tidy.cld()`, `tidy.confint.glht()`, `tidy.glht()`

**Examples**

```
library(multcomp)
library(ggplot2)

amod <- aov(breaks ~ wool + tension, data = warpbreaks)
wht <- glht(amod, linfct = mcp(tension = "Tukey"))

tidy(wht)
ggplot(wht, aes(lhs, estimate)) +
  geom_point()

CI <- confint(wht)
tidy(CI)
ggplot(CI, aes(lhs, estimate, ymin = lwr, ymax = upr)) +
  geom_pointrange()

tidy(summary(wht))
ggplot(mapping = aes(lhs, estimate)) +
  geom_linerange(aes(ymin = lwr, ymax = upr), data = CI) +
  geom_point(aes(size = p), data = summary(wht)) +
  scale_size(trans = "reverse")

cld <- cld(wht)
tidy(cld)
```

---

`tidy.summary.lm`

*Tidy a(n) summary.lm object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'summary.lm'
tidy(x, conf.int = FALSE, conf.level = 0.95, ...)
```

**Arguments**

x	A <code>summary.lm</code> object created by <code>stats::summary.lm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to <code>FALSE</code> .
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

The `tidy.summary.lm()` method is a potentially useful alternative to `tidy.lm()`. For instance, if users have already converted large `lm` objects into their leaner `summary.lm` equivalents to conserve memory.

**Value**

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
statistic	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

`tidy()`, `stats::summary.lm()`

Other `lm` tidiers: `augment.glm()`, `augment.lm()`, `glance.glm()`, `glance.lm()`, `glance.summary.lm()`, `glance.svyglm()`, `tidy.glm()`, `tidy.lm.beta()`, `tidy.lm()`, `tidy.mlm()`

## Examples

```
mod <- lm(mpg ~ wt + qsec, data = mtcars)
modsumm <- summary(mod)

tidy(mod, conf.int = TRUE)
tidy(modsumm, conf.int = TRUE) # same

glance(mod)
glance(modsumm) # mostly the same, except for a few missing columns
```

---

tidy.summary_emm	<i>Tidy a(n) summary_emm object</i>
------------------	-------------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'summary_emm'
tidy(x, null.value = NULL, ...)
```

## Arguments

x	A summary_emm object.
null.value	Value to which estimate is compared.
...	Additional arguments passed to <code>emmeans::summary.emmGrid()</code> or <code>lsmeans::summary.ref.grid()</code> . <b>Cautionary note:</b> misspecified arguments may be silently ignored!

## Details

Returns a data frame with one observation for each estimated marginal mean, and one column for each combination of factors. When the input is a contrast, each row will contain one estimated contrast.

There are a large number of arguments that can be passed on to `emmeans::summary.emmGrid()` or `lsmeans::summary.ref.grid()`.



**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>contrast</code>	Levels being compared.
<code>den.df</code>	Degrees of freedom of the denominator.
<code>df</code>	Degrees of freedom used by this term in the model.
<code>null.value</code>	Value to which the estimate is compared.
<code>num.df</code>	Degrees of freedom.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>std.error</code>	The standard error of the regression term.
<code>level1</code>	One level of the factor being contrasted
<code>level2</code>	The other level of the factor being contrasted
<code>term</code>	Model term in joint tests
<code>estimate</code>	Expected marginal mean
<code>statistic</code>	T-ratio statistic or F-ratio statistic

**See Also**

`tidy()`, `emmeans::ref_grid()`, `emmeans::emmeans()`, `emmeans::contrast()`

Other emmeans tidiers: `tidy.emmGrid()`, `tidy.lsmobj()`, `tidy.ref.grid()`

**Examples**

```
library(emmeans)
# linear model for sales of oranges per day
oranges_lm1 <- lm(sales1 ~ price1 + price2 + day + store, data = oranges)

# reference grid; see vignette("basics", package = "emmeans")
oranges_rg1 <- ref_grid(oranges_lm1)
td <- tidy(oranges_rg1)
td

# marginal averages
marginal <- emmeans(oranges_rg1, "day")
tidy(marginal)

# contrasts
tidy(contrast(marginal))
tidy(contrast(marginal, method = "pairwise"))

# plot confidence intervals
library(ggplot2)
ggplot(tidy(marginal, conf.int = TRUE), aes(day, estimate)) +
```

```

    geom_point() +
    geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# by multiple prices
by_price <- emmeans( oranges_lm1, "day",
  by = "price2",
  at = list(
    price1 = 50, price2 = c(40, 60, 80),
    day = c("2", "3", "4")
  )
)
by_price
tidy(by_price)

ggplot(tidy(by_price, conf.int = TRUE), aes(price2, estimate, color = day)) +
  geom_line() +
  geom_errorbar(aes(ymin = conf.low, ymax = conf.high))

# joint_tests
tidy(joint_tests( oranges_lm1 ))

```

---

tidy.survdiff

*Tidy a(n) survdiff object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'survdiff'
tidy(x, ...)
```

## Arguments

x	An survdiff object returned from <code>survival::survdiff()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

exp	Weighted expected number of events in each group.
N	Number of subjects in each group.
obs	weighted observed number of events in each group.

**See Also**

`tidy()`, `survival::survdiff()`

Other `survdiff` tidiers: `glance.survdiff()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdiff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survexp()`, `tidy.survfit()`, `tidy.survreg()`

**Examples**

```
library(survival)

s <- survdiff(
  Surv(time, status) ~ pat.karno + strata(inst),
  data = lung
)

tidy(s)
glance(s)
```

---

tidy.survexp

*Tidy a(n) survexp object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'survexp'
tidy(x, ...)
```

## Arguments

`x` An survexp object returned from `survival::survexp()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

## Value

A `tibble::tibble()` with columns:

<code>n.risk</code>	Number of individuals at risk at time zero.
<code>time</code>	Point in time.
<code>estimate</code>	Estimate survival

## See Also

`tidy()`, `survival::survexp()`

Other survexp tidiers: `glance.survexp()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdiff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdiff()`, `tidy.survfit()`, `tidy.survreg()`

## Examples

```
library(survival)
sexpfit <- survexp(
  futime ~ 1,
  rmap = list(
    sex = "male",
    year = accept.dt,
    age = (accept.dt - birth.dt)
  ),
  method = "conditional",
  data = jasa
)

tidy(sexpfit)
glance(sexpfit)
```

---

tidy.survfit	<i>Tidy a(n) survfit object</i>
--------------	---------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'survfit'
tidy(x, ...)
```

## Arguments

x	An survfit object returned from <code>survival::survfit()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>n.censor</code>	Number of censored events.
<code>n.event</code>	Number of events at time t.
<code>n.risk</code>	Number of individuals at risk at time zero.
<code>std.error</code>	The standard error of the regression term.
<code>time</code>	Point in time.
<code>estimate</code>	estimate of survival or cumulative incidence rate when multistate
<code>state</code>	state if multistate survfit object input
<code>strata</code>	strata if stratified survfit object input

**See Also**

`tidy()`, `survival::survfit()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdiff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdiff()`, `tidy.survexp()`, `tidy.survreg()`

**Examples**

```
library(survival)
cfits <- coxph(Surv(time, status) ~ age + sex, lung)
sfit <- survfit(cfits)

tidy(sfit)
glance(sfit)

library(ggplot2)
ggplot(tidy(sfit), aes(time, estimate)) +
  geom_line() +
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .25)

# multi-state
fitCI <- survfit(Surv(stop, status * as.numeric(event), type = "mstate") ~ 1,
  data = mgus1, subset = (start == 0)
)
td_multi <- tidy(fitCI)
td_multi

ggplot(td_multi, aes(time, estimate, group = state)) +
  geom_line(aes(color = state)) +
  geom_ribbon(aes(ymin = conf.low, ymax = conf.high), alpha = .25)
```

---

tidy.survreg

*Tidy a(n) survreg object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'survreg'
tidy(x, conf.level = 0.95, conf.int = FALSE, ...)
```

**Arguments**

<code>x</code>	An survreg object returned from <code>survival::survreg()</code> .
<code>conf.level</code>	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
<code>conf.int</code>	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `survival::survreg()`

Other survreg tidiers: `augment.survreg()`, `glance.survreg()`

Other survival tidiers: `augment.coxph()`, `augment.survreg()`, `glance.aareg()`, `glance.cch()`, `glance.coxph()`, `glance.pyears()`, `glance.survdifff()`, `glance.survexp()`, `glance.survfit()`, `glance.survreg()`, `tidy.aareg()`, `tidy.cch()`, `tidy.coxph()`, `tidy.pyears()`, `tidy.survdifff()`, `tidy.survexp()`, `tidy.survfit()`

**Examples**

```
library(survival)

sr <- survreg(
  Surv(futime, fustat) ~ ecog.ps + rx,
  ovarian,
  dist = "exponential"
```

```

)

tidy(sr)
augment(sr, ovarian)
glance(sr)

# coefficient plot
td <- tidy(sr, conf.int = TRUE)
library(ggplot2)
ggplot(td, aes(estimate, term)) +
  geom_point() +
  geom_errorbarh(aes(xmin = conf.low, xmax = conf.high), height = 0) +
  geom_vline(xintercept = 0)

```

---

tidy.svyglm

*Tidy a(n) svyglm object*


---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```

## S3 method for class 'svyglm'
tidy(x, conf.int = FALSE, conf.level = 0.95, exponentiate = FALSE, ...)

```

## Arguments

x	A svyglm object returned from <code>survey::svyglm()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.



**See Also**

[survey::svyglm\(\)](#), [stats::glm\(\)](#)

---

tidy.svyolr	<i>Tidy a(n) svyolr object</i>
-------------	--------------------------------

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'svyolr'
tidy(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  exponentiate = FALSE,
  p.values = FALSE,
  ...
)
```

**Arguments**

x	A svyolr object returned from <a href="#">survey::svyolr()</a> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
exponentiate	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to FALSE.
p.values	Logical. Should p-values be returned, based on chi-squared tests from <a href="#">MASS::dropterm()</a> . Defaults to FALSE.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <a href="#">augment()</a> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

## Details

In broom 0.7.0 the `coefficient_type` column was renamed to `coef.type`, and the contents were changed as well. Now the contents are `coefficient` and `scale`, rather than `coefficient` and `zeta`.

Calculating p-values with the `dropterm()` function is the approach suggested by the MASS package author <https://r.789695.n4.nabble.com/p-values-of-plor-td4668100.html>. This approach is computationally intensive so that p-values are only returned if requested explicitly. Additionally, it only works for models containing no variables with more than two categories. If this condition is not met, a message is shown and NA is returned instead of p-values.

## Value

A `tibble::tibble()` with columns:

<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>estimate</code>	The estimated value of the regression term.
<code>p.value</code>	The two-sided p-value associated with the observed statistic.
<code>statistic</code>	The value of a T-statistic to use in a hypothesis that the regression term is non-zero.
<code>std.error</code>	The standard error of the regression term.
<code>term</code>	The name of the regression term.

## See Also

`tidy`, `survey::svyolr()`

Other ordinal tidiers: `augment.clm()`, `augment.polr()`, `glance.clmm()`, `glance.clm()`, `glance.polr()`, `glance.svyolr()`, `tidy.clmm()`, `tidy.clm()`, `tidy.polr()`

## Examples

```
library(MASS)

fit <- polr(Sat ~ Infl + Type + Cont, weights = Freq, data = housing)

tidy(fit, exponentiate = TRUE, conf.int = TRUE)
glance(fit)
```

---

tidy.systemfit	<i>Tidy a(n) systemfit object</i>
----------------	-----------------------------------

---

## Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

## Usage

```
## S3 method for class 'systemfit'
tidy(x, conf.int = TRUE, conf.level = 0.95, ...)
```

## Arguments

x	A systemfit object produced by a call to <code>systemfit::systemfit()</code> .
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if <code>conf.int = TRUE</code> . Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

This tidy method works with any model objects of class `systemfit`. Default returns a tibble of six columns.

## Value

A `tibble::tibble()` with columns:

conf.high	Upper bound on the confidence interval for the estimate.
conf.low	Lower bound on the confidence interval for the estimate.
estimate	The estimated value of the regression term.
p.value	The two-sided p-value associated with the observed statistic.
std.error	The standard error of the regression term.
term	The name of the regression term.

**See Also**

`tidy()`, `systemfit::systemfit()`

**Examples**

```
set.seed(27)

library(systemfit)

df <- data.frame(
  X = rnorm(100),
  Y = rnorm(100),
  Z = rnorm(100),
  W = rnorm(100)
)

fit <- systemfit(formula = list(Y ~ Z, W ~ X), data = df, method = "SUR")
tidy(fit)

tidy(fit, conf.int = TRUE)
```

---

tidy.table

*Tidy a(n) table object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

Deprecated. Please use `tibble::as_tibble()` instead.

**Usage**

```
## S3 method for class 'table'
tidy(x, ...)
```

**Arguments**

`x` A `base::table` object.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Details**

Directly calls `tibble::as_tibble()` on a `base::table` object, which does the same things as `base::as.data.frame.table()` but also gives the returned object `tibble::tibble` class.

**Value**

A `tibble::tibble` in long-form containing frequency information for the table in a Freq column. The result is much like what you get from `tidyr::pivot_longer()`.

**See Also**

`tibble::as_tibble.table()`

---

tidy.ts

*Tidy a(n) ts object*


---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'ts'
tidy(x, ...)
```

**Arguments**

`x` A univariate or multivariate `ts` times series object.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

**Details**

`series` column is only present for multivariate `ts` objects.

**Value**

A `tibble::tibble()` with columns:

index	Index (i.e. date or time) for a 'ts' or 'zoo' object.
series	Name of the series (present only for multivariate time series).
value	The value/estimate of the component. Results from data reshaping.

**See Also**

`tidy()`, `stats::ts()`

Other time series tidiers: `tidy.acf()`, `tidy.spec()`, `tidy.zoo()`

**Examples**

```
set.seed(678)

tidy(ts(1:10, frequency = 4, start = c(1959, 2)))

z <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 1), frequency = 12)
colnames(z) <- c("Aa", "Bb", "Cc")
tidy(z)
```

---

tidy.TukeyHSD

*Tidy a(n) TukeyHSD object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'TukeyHSD'
tidy(x, ...)
```

**Arguments**

x	A TukeyHSD object return from <code>stats::TukeyHSD()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>adj.p.value</code>	P-value adjusted for multiple comparisons.
<code>conf.high</code>	Upper bound on the confidence interval for the estimate.
<code>conf.low</code>	Lower bound on the confidence interval for the estimate.
<code>contrast</code>	Levels being compared.
<code>estimate</code>	The estimated value of the regression term.
<code>null.value</code>	Value to which the estimate is compared.
<code>term</code>	The name of the regression term.

**See Also**

`tidy()`, `stats::TukeyHSD()`

Other anova tidiers: `glance.aov()`, `tidy.anova()`, `tidy.aovlist()`, `tidy.aov()`, `tidy.manova()`

**Examples**

```
fm1 <- aov(breaks ~ wool + tension, data = warpbreaks)
thsd <- TukeyHSD(fm1, "tension", ordered = TRUE)
tidy(thsd)

# may include comparisons on multiple terms
fm2 <- aov(mpg ~ as.factor(gear) * as.factor(cyl), data = mtcars)
tidy(TukeyHSD(fm2))
```

---

`tidy.zoo`

*Tidy a(n) zoo object*

---

**Description**

Tidy summarizes information about the components of a model. A model component might be a single term in a regression, a single hypothesis, a cluster, or a class. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

**Usage**

```
## S3 method for class 'zoo'
tidy(x, ...)
```

**Arguments**

`x` A zoo object such as those created by `zoo::zoo()`.

`...` Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the data argument.

**Value**

A `tibble::tibble()` with columns:

<code>index</code>	Index (i.e. date or time) for a 'ts' or 'zoo' object.
<code>series</code>	Name of the series (present only for multivariate time series).
<code>value</code>	The value/estimate of the component. Results from data reshaping.

**See Also**

`tidy()`, `zoo::zoo()`

Other time series tidiers: `tidy.acf()`, `tidy.spec()`, `tidy.ts()`

**Examples**

```
library(zoo)
library(ggplot2)

set.seed(1071)

# data generated as shown in the zoo vignette
Z.index <- as.Date(sample(12450:12500, 10))
Z.data <- matrix(rnorm(30), ncol = 3)
colnames(Z.data) <- c("Aa", "Bb", "Cc")
Z <- zoo(Z.data, Z.index)

tidy(Z)

ggplot(tidy(Z), aes(index, value, color = series)) +
  geom_line()

ggplot(tidy(Z), aes(index, value)) +
  geom_line() +
  facet_wrap(~series, ncol = 1)

Zrolled <- rollmean(Z, 5)
ggplot(tidy(Zrolled), aes(index, value, color = series)) +
  geom_line()
```



tidy\_irlba

*Tidy a(n) irlba object masquerading as list*

## Description

Broom tidies a number of lists that are effectively S3 objects without a class attribute. For example, `stats::optim()`, `svd()` and `akima::interp()` produce consistent output, but because they do not have a class attribute, they cannot be handled by S3 dispatch.

These functions look at the elements of a list and determine if there is an appropriate tidying method to apply to the list. Those tidiers are themselves implemented as functions of the form `tidy_<function>` or `glance_<function>` and are not exported (but they are documented!).

If no appropriate tidying method is found, throws an error.

## Usage

```
tidy_irlba(x, ...)
```

## Arguments

<code>x</code>	A list returned from <code>irlba::irlba()</code> .
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

## Details

A very thin wrapper around `tidy_svd()`.

## Value

A `tibble::tibble` with columns depending on the component of PCA being tidied.

If `matrix` is "u", "samples", "scores", or "x" each row in the tidied output corresponds to the original data in PCA space. The columns are:

<code>row</code>	ID of the original observation (i.e. <code>rowname</code> from original data).
<code>PC</code>	Integer indicating a principal component.
<code>value</code>	The score of the observation for that particular principal component. That is, the location of the observation in PCA space.

If `matrix` is "v", "rotation", "loadings" or "variables", each row in the tidied output corresponds to information about the principle components in the original space. The columns are:

row	The variable labels (colnames) of the data set on which PCA was performed
PC	An integer vector indicating the principal component
value	The value of the eigenvector (axis score) on the indicated principal component

If `matrix` is "d", "eigenvalues" or "pcs", the columns are:

PC	An integer vector indicating the principal component
std.dev	Standard deviation explained by this PC
percent	Fraction of variation explained by this component (a numeric value between 0 and 1).
cumulative	Cumulative fraction of variation explained by principle components up to this component (a numeric value between 0 and 1).

### See Also

[tidy\(\)](#), [irlba::irlba\(\)](#)

Other list tidiers: [glance\\_optim\(\)](#), [list\\_tidiers](#), [tidy\\_optim\(\)](#), [tidy\\_svd\(\)](#), [tidy\\_xyz\(\)](#)

Other svd tidiers: [augment.prcomp\(\)](#), [tidy.prcomp\(\)](#), [tidy\\_svd\(\)](#)

### Examples

```
library(modeldata)
data(hpc_data)

mat <- scale(as.matrix(hpc_data[, 2:5]))
s <- svd(mat)

tidy_u <- tidy(s, matrix = "u")
tidy_u

tidy_d <- tidy(s, matrix = "d")
tidy_d

tidy_v <- tidy(s, matrix = "v")
tidy_v

library(ggplot2)
library(dplyr)

ggplot(tidy_d, aes(PC, percent)) +
  geom_point() +
  ylab("% of variance explained")

tidy_u %>%
  mutate(class = hpc_data$class[row]) %>%
  ggplot(aes(class, value)) +
  geom_boxplot() +
  facet_wrap(~PC, scale = "free_y")
```

tidy\_optim

*Tidy a(n) optim object masquerading as list***Description**

Broom tidies a number of lists that are effectively S3 objects without a class attribute. For example, `stats::optim()`, `svd()` and `akima::interp()` produce consistent output, but because they do not have a class attribute, they cannot be handled by S3 dispatch.

These functions look at the elements of a list and determine if there is an appropriate tidying method to apply to the list. Those tidiers are themselves implemented as functions of the form `tidy_<function>` or `glance_<function>` and are not exported (but they are documented!).

If no appropriate tidying method is found, throws an error.

**Usage**

```
tidy_optim(x, ...)
```

**Arguments**

x	A list returned from <code>stats::optim()</code> .
...	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the <code>data</code> argument.

**Value**

A `tibble::tibble()` with columns:

parameter	The parameter being modeled.
std.error	The standard error of the regression term.
value	The value/estimate of the component. Results from data reshaping.

`std.error` is only provided as a column if the Hessian is calculated.

**Note**

This function assumes that the provided objective function is a negative log-likelihood function. Results will be invalid if an incorrect function is supplied.

`tidy(o)` `glance(o)`

**See Also**

`tidy()`, `stats::optim()`

Other list tidiers: `glance_optim()`, `list_tidiers`, `tidy_irlba()`, `tidy_svd()`, `tidy_xyz()`

**Examples**

```
f <- function(x) (x[1] - 2)^2 + (x[2] - 3)^2 + (x[3] - 8)^2
o <- optim(c(1, 1, 1), f)
```

---

tidy\_svd

*Tidy a(n) svd object masquerading as list*

---

**Description**

Broom tidies a number of lists that are effectively S3 objects without a class attribute. For example, `stats::optim()`, `svd()` and `akima::interp()` produce consistent output, but because they do not have a class attribute, they cannot be handled by S3 dispatch.

These functions look at the elements of a list and determine if there is an appropriate tidying method to apply to the list. Those tidiers are themselves implemented as functions of the form `tidy_<function>` or `glance_<function>` and are not exported (but they are documented!).

If no appropriate tidying method is found, throws an error.

**Usage**

```
tidy_svd(x, matrix = "u", ...)
```

**Arguments**

<code>x</code>	A list with components <code>u</code> , <code>d</code> , <code>v</code> returned by <code>base::svd()</code> .
<code>matrix</code>	Character specifying which component of the PCA should be tidied. <ul style="list-style-type: none"> <li>• <code>"u"</code>, <code>"samples"</code>, <code>"scores"</code>, or <code>"x"</code>: returns information about the map from the original space into principle components space.</li> <li>• <code>"v"</code>, <code>"rotation"</code>, <code>"loadings"</code> or <code>"variables"</code>: returns information about the map from principle components space back into the original space.</li> <li>• <code>"d"</code>, <code>"eigenvalues"</code> or <code>"pcs"</code>: returns information about the eigenvalues.</li> </ul>
<code>...</code>	Additional arguments. Not used. Needed to match generic signature only. <b>Cautionary note:</b> Misspelled arguments will be absorbed in <code>...</code> , where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass <code>conf.level = 0.9</code> , all computation will proceed using <code>conf.level = 0.95</code> . Additionally, if you pass <code>newdata = my_tibble</code> to an <code>augment()</code> method that does not accept a <code>newdata</code> argument, it will use the default value for the data argument.

**Details**

See <https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca> for information on how to interpret the various tidied matrices. Note that SVD is only equivalent to PCA on centered data.

**Value**

A `tibble::tibble` with columns depending on the component of PCA being tidied.

If `matrix` is "u", "samples", "scores", or "x" each row in the tidied output corresponds to the original data in PCA space. The columns are:

row	ID of the original observation (i.e. rowname from original data).
PC	Integer indicating a principal component.
value	The score of the observation for that particular principal component. That is, the location of the observation in PCA space.

If `matrix` is "v", "rotation", "loadings" or "variables", each row in the tidied output corresponds to information about the principle components in the original space. The columns are:

row	The variable labels (colnames) of the data set on which PCA was performed
PC	An integer vector indicating the principal component
value	The value of the eigenvector (axis score) on the indicated principal component

If `matrix` is "d", "eigenvalues" or "pcs", the columns are:

PC	An integer vector indicating the principal component
std.dev	Standard deviation explained by this PC
percent	Fraction of variation explained by this component (a numeric value between 0 and 1).
cumulative	Cumulative fraction of variation explained by principle components up to this component (a numeric value between 0 and 1).

**See Also**

`base::svd()`

Other svd tidiers: `augment.prcomp()`, `tidy.prcomp()`, `tidy_irlba()`

Other list tidiers: `glance_optim()`, `list_tidiers`, `tidy_irlba()`, `tidy_optim()`, `tidy_xyz()`

**Examples**

```
library(modeldata)
data(hpc_data)

mat <- scale(as.matrix(hpc_data[, 2:5]))
s <- svd(mat)
```

```

tidy_u <- tidy(s, matrix = "u")
tidy_u

tidy_d <- tidy(s, matrix = "d")
tidy_d

tidy_v <- tidy(s, matrix = "v")
tidy_v

library(ggplot2)
library(dplyr)

ggplot(tidy_d, aes(PC, percent)) +
  geom_point() +
  ylab("% of variance explained")

tidy_u %>%
  mutate(class = hpc_data$class[row]) %>%
  ggplot(aes(class, value)) +
  geom_boxplot() +
  facet_wrap(~PC, scale = "free_y")

```

---

tidy\_xyz

*Tidy a(n) xyz object masquerading as list*


---

## Description

Broom tidies a number of lists that are effectively S3 objects without a class attribute. For example, `stats::optim()`, `svd()` and `akima::interp()` produce consistent output, but because they do not have a class attribute, they cannot be handled by S3 dispatch.

These functions look at the elements of a list and determine if there is an appropriate tidying method to apply to the list. Those tidiers are themselves implemented as functions of the form `tidy_<function>` or `glance_<function>` and are not exported (but they are documented!).

If no appropriate tidying method is found, throws an error.

xyz lists (lists where x and y are vectors of coordinates and z is a matrix of values) are typically used by functions such as `graphics::persp()` or `graphics::image()` and returned by interpolation functions such as `akima::interp()`.

## Usage

```
tidy_xyz(x, ...)
```

## Arguments

x                    A list with component x, y and z, where x and y are vectors and z is a matrix. The length of x must equal the number of rows in z and the length of y must equal the number of columns in z.

...

Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in ..., where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.level = 0.9`, all computation will proceed using `conf.level = 0.95`. Additionally, if you pass `newdata = my_tibble` to an `augment()` method that does not accept a `newdata` argument, it will use the default value for the `data` argument.

### Value

A `tibble::tibble` with vector columns `x`, `y` and `z`.

### See Also

`tidy()`, `graphics::persp()`, `graphics::image()`, `akima::interp()`

Other list tidiers: `glance_optim()`, `list_tidiers`, `tidy_irlba()`, `tidy_optim()`, `tidy_svd()`

### Examples

```
A <- list(x = 1:5, y = 1:3, z = matrix(runif(5 * 3), nrow = 5))
image(A)
tidy(A)
```

# Index

- \* **Arima tidiers**
  - glance.Arima, 96
  - tidy.Arima, 204
- \* **aareg tidiers**
  - glance.aareg, 93
  - tidy.aareg, 198
- \* **anova tidiers**
  - glance.aov, 95
  - tidy.anova, 200
  - tidy.aov, 201
  - tidy.aovlist, 202
  - tidy.manova, 279
  - tidy.TukeyHSD, 350
- \* **betareg tidiers**
  - tidy.betareg, 207
- \* **biglm tidiers**
  - glance.biglm, 100
  - tidy.biglm, 208
- \* **bingroup tidiers**
  - glance.binDesign, 102
  - tidy.binDesign, 210
  - tidy.binWidth, 211
- \* **car tidiers**
  - durbinWatsonTest\_tidiers, 91
- \* **cch tidiers**
  - glance.cch, 103
  - glance.survfit, 185
  - tidy.cch, 215
- \* **coefstest tidiers**
  - tidy.coefstest, 222
- \* **coefstest\_tidiers**
  - glance.coefstest, 108
- \* **coxph tidiers**
  - augment.coxph, 17
  - glance.coxph, 110
  - tidy.coxph, 226
- \* **decompose tidiers**
  - augment.decomposed.ts, 19
  - augment.stl, 83
- \* **deprecated**
  - bootstrap, 87
  - confint\_tidy, 88
  - data.frame\_tidiers, 89
  - finish\_glance, 92
  - fix\_data\_frame, 93
  - summary\_tidiers, 196
  - tidy.density, 230
  - tidy.dist, 231
  - tidy.ftable, 243
  - tidy.gamlss, 245
  - tidy.numeric, 300
- \* **drc tidiers**
  - augment.drc, 22
  - glance.drc, 114
  - tidy.drc, 232
- \* **emmeans tidiers**
  - tidy.emmGrid, 234
  - tidy.lsmobj, 277
  - tidy.ref.grid, 317
  - tidy.summary\_emm, 336
- \* **epiR tidiers**
  - tidy.epi.2by2, 236
- \* **ergm tidiers**
  - glance.ergm, 115
  - tidy.ergm, 237
- \* **factanal tidiers**
  - augment.factanal, 24
  - glance.factanal, 117
  - tidy.factanal, 239
- \* **fitdistr tidiers**
  - glance.fitdistr, 118
  - tidy.fitdistr, 240
- \* **fixest tidiers**
  - augment.fixest, 26
  - tidy.fixest, 241
- \* **garch tidiers**
  - glance.garch, 123
  - tidy.garch, 247



- \* **geepack tidiers**
  - glance.geeglm, 124
- \* **glmnet tidiers**
  - glance.cv.glmnet, 112
  - glance.glmnet, 126
  - tidy.cv.glmnet, 228
  - tidy.glmnet, 252
- \* **gmm tidiers**
  - glance.gmm, 129
  - tidy.gmm, 256
- \* **htest tidiers**
  - augment.htest, 33
  - tidy.htest, 259
  - tidy.pairwise.htest, 303
  - tidy.power.htest, 311
- \* **ivreg tidiers**
  - augment.ivreg, 35
  - glance.ivreg, 131
  - tidy.ivreg, 260
- \* **kmeans tidiers**
  - augment.kmeans, 37
  - glance.kmeans, 133
  - tidy.kmeans, 266
- \* **lavaan tidiers**
  - glance.lavaan, 135
  - tidy.lavaan, 267
- \* **list tidiers**
  - glance\_optim, 192
  - list\_tidiers, 193
  - tidy\_irlba, 353
  - tidy\_optim, 355
  - tidy\_svd, 356
  - tidy\_xyz, 358
- \* **lm tidiers**
  - augment.glm, 28
  - augment.lm, 39
  - glance.glm, 125
  - glance.lm, 137
  - glance.summary.lm, 180
  - glance.svyglm, 189
  - tidy.glm, 251
  - tidy.lm, 269
  - tidy.lm.beta, 271
  - tidy.mlm, 292
  - tidy.summary.lm, 334
- \* **lmodel2 tidiers**
  - glance.lmodel2, 140
  - tidy.lmodel2, 273
- \* **margins tidiers**
  - tidy.margins, 282
- \* **mclust tidiers**
  - augment.Mclust, 48
  - tidy.Mclust, 284
- \* **mediate tidiers**
  - tidy.mediate, 285
- \* **mfx tidiers**
  - augment.betamfx, 10
  - augment.mfx, 50
  - glance.betamfx, 97
  - glance.mfx, 148
  - tidy.betamfx, 205
  - tidy.mfx, 287
- \* **mgev tidiers**
  - glance.gam, 121
  - tidy.gam, 244
- \* **mjoint tidiers**
  - glance.mjoint, 150
  - tidy.mjoint, 289
- \* **mlogit tidiers**
  - augment.mlogit, 55
  - glance.mlogit, 152
  - tidy.mlogit, 294
- \* **muhaz tidiers**
  - glance.muhaz, 153
  - tidy.muhaz, 295
- \* **multcomp tidiers**
  - tidy.cld, 217
  - tidy.confint.glht, 223
  - tidy.glht, 250
  - tidy.summary.glht, 333
- \* **multinom tidiers**
  - glance.multinom, 154
  - tidy.multinom, 296
- \* **nls tidiers**
  - augment.nls, 59
  - glance.nls, 157
  - tidy.nls, 299
- \* **orcutt tidiers**
  - glance.orcutt, 158
  - tidy.orcutt, 301
- \* **ordinal tidiers**
  - augment.clm, 15
  - augment.polr, 67
  - glance.clm, 105
  - glance.clmm, 106
  - glance.polr, 165

- glance.svyolr, 190
- tidy.clm, 218
- tidy.clmm, 220
- tidy.polr, 309
- tidy.svyolr, 345
- \* **pam tidiers**
  - augment.pam, 61
  - glance.pam, 160
  - tidy.pam, 304
- \* **plm tidiers**
  - augment.plm, 63
  - glance.plm, 161
  - tidy.plm, 306
- \* **poLCA tidiers**
  - augment.poLCA, 64
  - glance.poLCA, 163
  - tidy.poLCA, 307
- \* **pyears tidiers**
  - glance.pyears, 166
  - tidy.pyears, 314
- \* **quantreg tidiers**
  - augment.nlrq, 57
  - augment.rq, 74
  - augment.rqs, 76
  - glance.nlrq, 156
  - glance.rq, 172
  - tidy.nlrq, 298
  - tidy.rq, 325
  - tidy.rqs, 327
- \* **ridgelm tidiers**
  - glance.ridgelm, 168
  - tidy.ridgelm, 320
- \* **rlm tidiers**
  - augment.rlm, 70
  - glance.rlm, 169
  - tidy.rlm, 321
- \* **robust tidiers**
  - augment.lmRob, 42
  - glance.glmRob, 128
  - glance.lmRob, 141
  - tidy.glmRob, 254
  - tidy.lmRob, 275
- \* **robustbase tidiers**
  - augment.glmrob, 31
  - augment.lmrob, 44
  - glance.lmrob, 142
  - tidy.glmrob, 255
  - tidy.lmrob, 276
- \* **smoothing spline tidiers**
  - augment.smooth.spline, 80
  - glance.smooth.spline, 175
- \* **spatialreg tidiers**
  - augment.sarlm, 78
  - glance.sarlm, 174
  - tidy.sarlm, 328
- \* **speedlm tidiers**
  - augment.speedlm, 81
  - glance.speedglm, 177
  - glance.speedlm, 178
  - tidy.speedglm, 330
  - tidy.speedlm, 332
- \* **survdiff tidiers**
  - glance.survdiff, 182
  - tidy.survdiff, 338
- \* **survexp tidiers**
  - glance.survexp, 184
  - tidy.survexp, 339
- \* **survey tidiers**
  - tidy.svyglm, 344
- \* **survfit tidiers**
  - tidy.survfit, 341
- \* **survival tidiers**
  - augment.coxph, 17
  - augment.survreg, 84
  - glance.aareg, 93
  - glance.cch, 103
  - glance.coxph, 110
  - glance.pyears, 166
  - glance.survdiff, 182
  - glance.survexp, 184
  - glance.survfit, 185
  - glance.survreg, 187
  - tidy.aareg, 198
  - tidy.cch, 215
  - tidy.coxph, 226
  - tidy.pyears, 314
  - tidy.survdiff, 338
  - tidy.survexp, 339
  - tidy.survfit, 341
  - tidy.survreg, 342
- \* **survreg tidiers**
  - augment.survreg, 84
  - glance.survreg, 187
  - tidy.survreg, 342
- \* **svd tidiers**
  - augment.prcomp, 69

- tidy.prcomp, 312
- tidy\_irlba, 353
- tidy\_svd, 356
- \* **systemfit tidiers**
  - tidy.systemfit, 347
- \* **time series tidiers**
  - tidy.acf, 199
  - tidy.spec, 329
  - tidy.ts, 349
  - tidy.zoo, 351
- aareg\_tidiers(tidy.aareg), 198
- AER::ivreg(), 35, 36, 132, 133, 260, 261
- aer\_tidiers(tidy.ivreg), 260
- akima::interp(), 192, 193, 353, 355, 356, 358, 359
- Arima\_tidiers(tidy.Arima), 204
- AUC::roc(), 324
- auc\_tidiers(tidy.roc), 324
- augment, 76, 77
- augment(), 11, 14, 16, 18, 20, 21, 23–25, 27, 29, 32, 34, 36, 38, 40, 41, 43, 45–47, 49, 52, 54, 56–58, 60, 62, 64–66, 68, 70, 71, 73, 78–80, 82, 84–86, 91, 94–96, 98, 99, 101, 102, 104, 105, 107, 108, 110, 112, 114, 117, 119, 122–124, 126, 127, 129, 130, 132, 134, 136, 137, 140, 142, 143, 145, 147, 148, 150, 152, 154–157, 159, 160, 162, 163, 165, 167, 168, 170, 171, 173, 174, 176, 177, 179, 180, 183, 184, 187, 189, 191, 192, 195, 197–200, 202–205, 207, 209–211, 213–215, 217, 219, 220, 222, 224, 225, 227, 229–232, 236, 239, 241, 243, 244, 246–248, 250, 252, 254, 255, 257, 259, 261, 262, 264–266, 269, 272, 273, 275, 276, 281, 282, 284, 286, 287, 290, 292–295, 297–299, 302–304, 306, 307, 310–312, 314, 316, 320, 322–324, 328, 330–333, 335, 338, 340, 341, 343–345, 347–350, 352, 353, 355, 356, 359
- augment.betamfx, 10, 52, 98, 149, 206, 288
- augment.betareg, 12
- augment.betareg(), 11, 12
- augment.clm, 15, 68, 106, 107, 166, 191, 219, 221, 310, 346
- augment.coxph, 17, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 216, 227, 315, 339, 340, 342, 343
- augment.data.frame
  - (data.frame\_tidiers), 89
- augment.decomposed.ts, 19, 84
- augment.drc, 22, 115, 233
- augment.factanal, 24, 118, 239
- augment.fixest, 26, 242
- augment.glm, 28, 41, 126, 138, 181, 190, 252, 270, 272, 293, 335
- augment.glm(), 52
- augment.glmRob, 30
- augment.glmrob, 31, 45, 143, 256, 276
- augment.htest, 33, 259, 303, 312
- augment.ivreg, 35, 133, 261
- augment.kmeans, 37, 134, 267
- augment.lm, 30, 39, 126, 138, 181, 190, 252, 270, 272, 293, 335
- augment.lmRob, 42, 129, 142, 254, 275
- augment.lmrob, 33, 44, 143, 256, 276
- augment.loess, 46
- augment.logitmfx(augment.mfx), 50
- augment.Mclust, 48, 284
- augment.mfx, 12, 50, 98, 149, 206, 288
- augment.mjoint, 53
- augment.mlogit, 55, 153, 295
- augment.negbinmfx(augment.mfx), 50
- augment.nlrq, 57, 76, 77, 157, 173, 299, 326, 328
- augment.nls, 59, 158, 300
- augment.NULL(null\_tidiers), 194
- augment.pam, 61, 161, 305
- augment.plm, 63, 162, 307
- augment.poissonmfx(augment.mfx), 50
- augment.poLCA, 64, 164, 308
- augment.polr, 16, 67, 106, 107, 166, 191, 219, 221, 310, 346
- augment.prcomp, 69, 313, 354, 357
- augment.probitmfx(augment.mfx), 50
- augment.rlm, 70, 170, 322
- augment.rma, 72
- augment.rq, 58, 74, 77, 157, 173, 299, 326, 328
- augment.rqs, 58, 76, 76, 157, 173, 299, 326, 328
- augment.sarlm, 78, 175, 329
- augment.smooth.spline, 80, 176

- augment.speedlm, [81](#), [178](#), [179](#), [331](#), [333](#)
- augment.stl, [21](#), [83](#)
- augment.survreg, [18](#), [84](#), [94](#), [104](#), [111](#), [167](#), [183](#), [185](#), [186](#), [188](#), [199](#), [216](#), [227](#), [315](#), [339](#), [340](#), [342](#), [343](#)
- augment\_columns, [86](#)
- base::as.data.frame.table(), [349](#)
- base::data.frame, [11](#), [13](#), [16](#), [18](#), [23](#), [25](#), [27](#), [29](#), [32](#), [35](#), [38](#), [40](#), [43](#), [45](#), [46](#), [48](#), [51](#), [54](#), [58](#), [59](#), [61](#), [63](#), [65](#), [68](#), [69](#), [71](#), [75](#), [77](#), [80](#), [82](#), [85](#)
- base::data.frame(), [11](#), [13](#), [16](#), [18](#), [23](#), [27](#), [29](#), [32](#), [36](#), [40](#), [43](#), [45](#), [46](#), [52](#), [58](#), [60](#), [68](#), [70](#), [71](#), [75](#), [77](#), [82](#), [85](#)
- base::svd(), [193](#), [356](#), [357](#)
- base::table, [348](#), [349](#)
- bbmle::mle2(), [291](#), [292](#)
- bbmle\_tidiers (tidy.mle2), [291](#)
- betareg::betareg(), [13](#), [14](#), [99](#), [100](#), [207](#), [208](#)
- betareg::predict.betareg(), [11](#)
- betareg::residuals.betareg(), [11](#)
- betareg\_tidiers (tidy.betareg), [207](#)
- biglm::bigglm(), [101](#), [209](#)
- biglm::biglm(), [101](#), [209](#)
- bindesign\_tidiers (tidy.binDesign), [210](#)
- binGroup::binDesign, [102](#)
- binGroup::binDesign(), [103](#), [210](#), [211](#)
- binGroup::binWidth(), [211](#), [212](#)
- binwidth\_tidiers (tidy.binWidth), [211](#)
- boot::boot(), [213](#)
- boot::boot.ci(), [213](#)
- boot::tsboot(), [213](#)
- boot\_tidiers (tidy.boot), [212](#)
- bootstrap, [87](#), [88](#), [90](#), [92](#), [93](#), [197](#), [231](#), [232](#), [244](#), [246](#), [301](#)
- btergm::btergm(), [214](#), [215](#)
- btergm\_tidiers (tidy.btergm), [214](#)
- car::Anova(), [200](#), [201](#)
- car::durbinWatsonTest(), [91](#)
- caret::confusionMatrix(), [225](#)
- caret\_tidiers (tidy.confusionMatrix), [225](#)
- cch\_tidiers (tidy.cch), [215](#)
- cfa\_tidiers (tidy.lavaan), [267](#)
- cluster::pam(), [61](#), [62](#), [160](#), [161](#), [304](#), [305](#)
- coefstest\_tidiers (tidy.coefstest), [222](#)
- confint(), [88](#)
- confint\_tidy, [88](#), [88](#), [90](#), [92](#), [93](#), [197](#), [231](#), [232](#), [244](#), [246](#), [301](#)
- confusionMatrix\_tidiers (tidy.confusionMatrix), [225](#)
- coxph\_tidiers (tidy.coxph), [226](#)
- data.frame\_tidiers, [88](#), [89](#), [92](#), [93](#), [197](#), [231](#), [232](#), [244](#), [246](#), [301](#)
- decompose\_tidiers (augment.decomposed.ts), [19](#)
- drc::drm(), [23](#), [24](#), [114](#), [115](#), [232](#), [233](#)
- drc\_tidiers (tidy.drc), [232](#)
- durbinWatsonTest\_tidiers, [91](#)
- emmeans::contrast(), [235](#), [278](#), [318](#), [337](#)
- emmeans::emmeans(), [235](#), [278](#), [318](#), [337](#)
- emmeans::ref\_grid(), [235](#), [278](#), [317](#), [318](#), [337](#)
- emmeans::summary.emmGrid(), [234](#), [277](#), [318](#), [336](#)
- emmeans\_tidiers (tidy.lsmobj), [277](#)
- epiR::epi.2by2(), [236](#), [237](#)
- epiR\_tidiers (tidy.epi.2by2), [236](#)
- ergm::control.ergm(), [238](#)
- ergm::ergm(), [116](#), [237](#), [238](#)
- ergm::summary(), [116](#), [237](#), [238](#)
- ergm::summary.ergm(), [116](#)
- ergm\_tidiers (tidy.ergm), [237](#)
- factanal\_tidiers (tidy.factanal), [239](#)
- finish\_glance, [88](#), [90](#), [92](#), [93](#), [197](#), [231](#), [232](#), [244](#), [246](#), [301](#)
- fitdistr\_tidiers (tidy.fitdistr), [240](#)
- fix\_data\_frame, [88](#), [90](#), [92](#), [93](#), [197](#), [231](#), [232](#), [244](#), [246](#), [301](#)
- fixest::feglm(), [27](#), [242](#)
- fixest::femlm(), [27](#), [242](#)
- fixest::fenegbin(), [27](#), [242](#)
- fixest::feNmlm(), [27](#), [242](#)
- fixest::feols(), [27](#), [242](#)
- fixest::fepois(), [27](#), [242](#)
- gam\_tidiers (tidy.gam), [244](#)
- gamlss::gamlss(), [246](#)
- garch\_tidiers (tidy.garch), [247](#)
- geeglm\_tidiers (tidy.geeglm), [248](#)
- geepack::geeglm(), [124](#), [125](#), [248](#), [249](#)
- geepack\_tidiers (tidy.geeglm), [248](#)

- glance(), *91, 94, 96, 100, 101, 103, 104, 109, 111, 113, 115, 116, 118, 122, 123, 125, 127, 130, 133, 134, 136, 138, 141, 151, 153–155, 157, 159, 161, 162, 164, 167, 169, 170, 173, 175, 181, 183, 185, 186, 188, 192, 259*
- glance.aareg, *18, 86, 93, 104, 111, 167, 183, 185, 186, 188, 199, 216, 227, 315, 339, 340, 342, 343*
- glance.aov, *95, 201–203, 280, 351*
- glance.Arima, *96, 204*
- glance.betamfx, *12, 52, 97, 149, 206, 288*
- glance.betareg, *99*
- glance.betareg(), *98*
- glance.biglm, *100, 209*
- glance.binDesign, *102, 211, 212*
- glance.cch, *18, 86, 94, 103, 111, 167, 183, 185, 186, 188, 199, 216, 227, 315, 339, 340, 342, 343*
- glance.clm, *16, 68, 105, 107, 166, 191, 219, 221, 310, 346*
- glance.clm, *16, 68, 106, 106, 166, 191, 219, 221, 310, 346*
- glance.coeftest, *108*
- glance.coxph, *18, 86, 94, 104, 110, 167, 183, 185, 186, 188, 199, 216, 227, 315, 339, 340, 342, 343*
- glance.cv.glmnet, *112, 127, 229, 253*
- glance.data.frame (data.frame\_tidiers), *89*
- glance.drc, *24, 114, 233*
- glance.durbinWatsonTest (durbinWatsonTest\_tidiers), *91*
- glance.ergm, *115, 238*
- glance.factanal, *25, 117, 239*
- glance.fitdistr, *118, 241*
- glance.fixest, *120*
- glance.gam, *121, 245*
- glance.garch, *123, 247*
- glance.geeglm, *124*
- glance.glm, *30, 41, 125, 138, 181, 190, 252, 270, 272, 293, 335*
- glance.glm(), *149*
- glance.glmnet, *113, 126, 229, 253*
- glance.glmRob, *43, 128, 142, 254, 275*
- glance.gmm, *129, 257*
- glance.htest (tidy.htest), *259*
- glance.ivreg, *36, 131, 261*
- glance.kmeans, *38, 133, 267*
- glance.lavaan, *135, 268*
- glance.list (list\_tidiers), *193*
- glance.lm, *30, 41, 126, 137, 181, 190, 252, 270, 272, 293, 335*
- glance.lm(), *180*
- glance.lmodel2, *140, 274*
- glance.lmRob, *43, 129, 141, 254, 275*
- glance.lmrob, *33, 45, 142, 256, 276*
- glance.logitmfx (glance.mfx), *148*
- glance.margins, *144*
- glance.Mclust, *146*
- glance.mfx, *12, 52, 98, 148, 206, 288*
- glance.mjoint, *150, 290*
- glance.mlogit, *57, 152, 295*
- glance.muhaz, *153, 296*
- glance.multinom, *154, 297*
- glance.negbinmfx (glance.mfx), *148*
- glance.nlrq, *58, 76, 77, 156, 173, 299, 326, 328*
- glance.nls, *60, 157, 300*
- glance.NULL (null\_tidiers), *194*
- glance.optim (glance\_optim), *192*
- glance.orcutt, *158, 302*
- glance.pam, *62, 160, 305*
- glance.plm, *64, 161, 307*
- glance.poissonmfx (glance.mfx), *148*
- glance.poLCA, *66, 163, 308*
- glance.polr, *16, 68, 106, 107, 165, 191, 219, 221, 310, 346*
- glance.probitmfx (glance.mfx), *148*
- glance.pyyears, *18, 86, 94, 104, 111, 166, 183, 185, 186, 188, 199, 216, 227, 315, 339, 340, 342, 343*
- glance.ridge, *168, 321*
- glance.rlm, *72, 169, 322*
- glance.rma, *171*
- glance.rq, *58, 76, 77, 157, 172, 299, 326, 328*
- glance.sarlm, *79, 174, 329*
- glance.smooth.spline, *80, 175*
- glance.speedglm, *82, 177, 179, 331, 333*
- glance.speedlm, *82, 178, 178, 331, 333*
- glance.summary.lm, *30, 41, 126, 138, 180, 190, 252, 270, 272, 293, 335*
- glance.summary.lm(), *138, 181*
- glance.summaryDefault (summary\_tidiers), *196*
- glance.survdiff, *18, 86, 94, 104, 111, 167,*

- 182, 185, 186, 188, 199, 216, 227, 315, 339, 340, 342, 343
- glance.survexp, 18, 86, 94, 104, 111, 167, 183, 184, 186, 188, 199, 216, 227, 315, 339, 340, 342, 343
- glance.survfit, 18, 86, 94, 104, 111, 167, 183, 185, 185, 188, 199, 216, 227, 315, 339, 340, 342, 343
- glance.survreg, 18, 86, 94, 104, 111, 167, 183, 185, 186, 187, 199, 216, 227, 315, 339, 340, 342, 343
- glance.svyglm, 30, 41, 126, 138, 181, 189, 252, 270, 272, 293, 335
- glance.svyolr, 16, 68, 106, 107, 166, 190, 219, 221, 310, 346
- glance\_optim, 192, 193, 354, 356, 357, 359
- glmnet::cv.glmnet(), 112, 113, 228, 229
- glmnet::glmnet(), 127, 252, 253
- glmnet\_tidiers (tidy.glmnet), 252
- gmm::gmm(), 130, 256, 257
- gmm\_tidiers (tidy.gmm), 256
- graphics::image(), 358, 359
- graphics::persp(), 358, 359
- Hmisc::rcorr(), 316, 317
- Hmisc\_tidiers (tidy.rcorr), 316
- htest\_tidiers (tidy.htest), 259
- irlba::irlba(), 353, 354
- irlba\_tidiers (tidy\_irlba), 353
- ivreg\_tidiers (tidy.ivreg), 260
- joinerML::bootSE(), 289, 290
- joinerML::fitted.mjoint(), 54
- joinerML::mjoint(), 54, 150, 151, 289, 290
- joinerML::residuals.mjoint(), 54
- joinerml\_tidiers (tidy.mjoint), 289
- kappa\_tidiers (tidy.kappa), 262
- kde\_tidiers (tidy.kde), 263
- Kendall::Kendall(), 265, 266
- Kendall::MannKendall(), 265, 266
- Kendall::SeasonalMannKendall(), 265, 266
- Kendall\_tidiers (tidy.Kendall), 265
- kendall\_tidiers (tidy.Kendall), 265
- kmeans\_tidiers (tidy.kmeans), 266
- ks::kde(), 264
- ks\_tidiers (tidy.kde), 263
- lavaan::cfa(), 135, 136, 268
- lavaan::fitmeasures(), 136
- lavaan::parameterEstimates(), 268
- lavaan::sem(), 135, 136, 268
- lavaan\_tidiers (tidy.lavaan), 267
- leaps::regsubsets(), 319, 320
- leaps\_tidiers (tidy.regsubsets), 319
- list\_tidiers, 192, 193, 354, 356, 357, 359
- lm.beta::lm.beta, 272
- lm\_tidiers (tidy.lm), 269
- lmodel2::lmodel2(), 140, 141, 273, 274
- lmodel2\_tidiers (tidy.lmodel2), 273
- lmtest::coefstest(), 108, 109, 222, 223
- lmtest\_tidiers (tidy.coefstest), 222
- loess\_tidiers (augment.loess), 46
- lsmeans::summary.ref.grid(), 234, 277, 318, 336
- maps::map(), 281
- maps\_tidiers (tidy.map), 280
- margins::margins(), 144, 282, 283
- margins\_tidiers (tidy.margins), 282
- MASS::dropterm(), 310, 345
- MASS::fitdistr(), 119, 240, 241
- MASS::lm.ridge(), 168, 169, 320, 321
- MASS::polr(), 68, 165, 166, 309, 310
- MASS::rlm(), 71, 72, 170, 322
- MASS::select.ridgeIm(), 169
- mclust::Mclust(), 48, 49, 146, 284
- mclust\_tidiers (tidy.Mclust), 284
- mean, 89
- mediate\_tidiers (tidy.mediate), 285
- mediation::mediate(), 285, 286
- metafor::escalc(), 323
- metafor::rma(), 73, 171, 323
- metafor::rma.glmm(), 73, 171, 323
- metafor::rma.mh(), 73, 171, 323
- metafor::rma.mv(), 73, 171, 323
- metafor::rma.peto(), 73, 171, 323
- metafor::rma.uni(), 73, 171, 323
- mfx::betamfx(), 11, 12, 98, 206
- mfx::logitmfx(), 52, 149, 288
- mfx::negbinmfx(), 52, 149, 288
- mfx::poissonmfx(), 52, 149, 288
- mfx::probitmfx(), 52, 149, 288
- mgcv::gam(), 122, 244, 245
- mgcv\_tidiers (tidy.gam), 244
- mjoint\_tidiers (tidy.mjoint), 289
- mle2\_tidiers (tidy.mle2), 291

- mlogit::mlogit(), 56, 152, 153, 294, 295
- mlogit\_tidiers (tidy.mlogit), 294
- muhaz::muhaz(), 153, 154, 295, 296
- muhaz\_tidiers (tidy.muhaz), 295
- multcomp::cld(), 217
- multcomp::confint.glm(), 217, 223, 224
- multcomp::glm(), 217, 223, 224, 250, 333, 334
- multcomp::summary.glm(), 217, 333, 334
- multcomp\_tidiers (tidy.glm), 250
- multinom\_tidiers (tidy.multinom), 296
- nlrq\_tidiers (tidy.nlrq), 298
- nls\_tidiers (tidy.nls), 299
- nnet::multinom(), 155, 296, 297
- nnet\_tidiers (tidy.multinom), 296
- null\_tidiers, 194
- optim\_tidiers (tidy.optim), 355
- orcutt::cochrane.orcutt(), 159, 302
- orcutt\_tidiers (tidy.orcutt), 301
- ordinal::clm(), 15, 16, 105, 106, 218, 219
- ordinal::clmm(), 107, 220, 221
- ordinal::confint.clm(), 219, 221
- ordinal::predict.clm(), 16
- ordinal\_tidiers (tidy.clm), 218
- pam\_tidiers (tidy.pam), 304
- plm::plm(), 63, 64, 162, 306, 307
- plm\_tidiers (tidy.plm), 306
- poLCA::poLCA(), 65, 66, 163, 164, 307, 308
- poLCA\_tidiers (tidy.poLCA), 307
- polr\_tidiers (tidy.polr), 309
- prcomp\_tidiers (tidy.prcomp), 312
- predict.fixest, 27
- psych::cohen.kappa(), 262, 263
- psych\_tidiers (tidy.kappa), 262
- purrr::map(), 173
- purrr::map\_df(), 194
- pyears\_tidiers (tidy.pyears), 314
- qr, 279
- quantreg::nlrq(), 58, 156, 157, 298, 299
- quantreg::predict.rq, 75, 77
- quantreg::predict.rq(), 76
- quantreg::predict.rqs(), 77
- quantreg::rq(), 75–77, 173, 326–328
- quantreg::summary.rq(), 326, 327
- quantreg::summary.rqs(), 327
- quantreg\_tidiers (tidy.rq), 325
- rcorr\_tidiers (tidy.rcorr), 316
- ridgelm\_tidiers (tidy.ridgelm), 320
- rlm\_tidiers (glance.rlm), 169
- robust::glmRob(), 128, 129, 254
- robust::lmRob(), 43, 142, 275
- robust\_tidiers (tidy.lmRob), 275
- robustbase::glmrob(), 32, 33, 255, 256
- robustbase::lmrob(), 45, 143, 276
- robustbase\_tidiers (tidy.lmrob), 276
- roc\_tidiers (tidy.roc), 324
- rq\_tidiers (tidy.rq), 325
- rqs\_tidiers (tidy.rqs), 327
- rsample::bootstraps(), 213
- sem\_tidiers (tidy.lavaan), 267
- sexpfit\_tidiers (tidy.survexp), 339
- smooth.spline\_tidiers  
(augment.smooth.spline), 80
- sp\_tidiers, 195
- sparse\_tidiers, 194
- spatialreg::errorsarlm(), 78, 174, 175, 328, 329
- spatialreg::lagsarlm(), 78, 174, 175, 328, 329
- spatialreg::sacsarlm(), 175, 329
- spatialreg\_tidiers (tidy.sarlm), 328
- speedglm::speedglm(), 177, 331
- speedglm::speedlm(), 82, 178, 179, 332, 333
- speedglm\_tidiers (tidy.speedglm), 330
- speedlm\_tidiers (tidy.speedlm), 332
- splines::ns(), 10, 13, 15, 17, 20, 22, 24, 26, 28, 30, 31, 33, 35, 37, 39, 43, 44, 48, 50, 53, 56, 59, 61, 63, 65, 67, 69, 71, 72, 74, 76, 78, 81, 83, 84
- stats::acf(), 199, 200
- stats::anova(), 200, 201
- stats::aov(), 95, 202, 203
- stats::arima(), 96, 97, 204
- stats::ccf(), 199, 200
- stats::chisq.test(), 34, 259
- stats::cooks.distance(), 14
- stats::cor.test(), 34, 259
- stats::decompose(), 20, 21
- stats::density(), 230
- stats::dist(), 231
- stats::factanal(), 25, 117, 118, 239



- stats::ftable(), 243
- stats::glm(), 29, 30, 125, 126, 190, 251, 252, 345
- stats::kmeans(), 37, 38, 134, 266, 267
- stats::lm(), 25, 39, 137, 180, 269, 293
- stats::loess(), 46, 47
- stats::manova(), 279
- stats::na.action, 18, 41, 47
- stats::nls(), 59, 60, 157, 158, 299, 300
- stats::optim(), 192, 193, 353, 355, 356, 358
- stats::pacf(), 199, 200
- stats::pairwise.t.test(), 303
- stats::pairwise.wilcox.test(), 303
- stats::poly(), 10, 13, 15, 17, 20, 22, 24, 26, 28, 30, 31, 33, 35, 37, 39, 43, 44, 48, 50, 53, 56, 59, 61, 63, 65, 67, 69, 71, 72, 74, 76, 78, 81, 83, 84
- stats::power.t.test(), 311, 312
- stats::prcomp(), 69, 70, 312, 313
- stats::predict(), 13, 18, 32, 85
- stats::predict.glm(), 29, 52
- stats::predict.lm(), 41
- stats::predict.loess(), 47
- stats::predict.nls(), 60
- stats::predict.smooth.spline(), 80
- stats::residuals(), 14, 18, 32, 85
- stats::residuals.glm(), 29, 52
- stats::rstandard.glm(), 29, 52
- stats::smooth.spline(), 80, 176
- stats::spectrum(), 330
- stats::stl(), 83, 84
- stats::summary.lm(), 270, 335
- stats::summary.manova, 279
- stats::summary.manova(), 280
- stats::summary.nls(), 300
- stats::t.test(), 34, 259
- stats::ts(), 350
- stats::TukeyHSD(), 350, 351
- stats::wilcox.test(), 34, 259
- summary(), 197
- summary.fixest, 27, 120, 242
- summary.survfit(), 186
- summary\_tidiers, 88, 90, 92, 93, 196, 231, 232, 244, 246, 301
- survdifftidiers(tidy.survdifft), 338
- survexp\_tidiers(tidy.survexp), 339
- survey::anova.svyglm, 190
- survey::svyglm(), 189, 190, 344, 345
- survey::svyolr(), 191, 345, 346
- survfit\_tidiers(tidy.survfit), 341
- survival::aareg(), 94, 198, 199
- survival::cch(), 104, 215, 216
- survival::coxph(), 17, 18, 110, 111, 226, 227
- survival::pyears(), 167, 314, 315
- survival::Surv(), 10, 13, 15, 17, 20, 22, 25, 26, 28, 31, 33, 35, 37, 39, 43, 44, 48, 50, 54, 56, 59, 61, 63, 65, 67, 69, 71, 72, 74, 76, 78, 81, 83, 85
- survival::survdiff(), 183, 338, 339
- survival::survexp(), 184, 185, 340
- survival::survfit(), 186, 341, 342
- survival::survreg(), 85, 86, 187, 188, 343
- survreg\_tidiers(tidy.survreg), 342
- svd(), 192, 353, 355, 356, 358
- svd\_tidiers, 70, 313
- svd\_tidiers(tidy\_svd), 356
- svyolr\_tidiers(tidy.svyolr), 345
- systemfit::systemfit(), 347, 348
- systemfit\_tidiers(tidy.systemfit), 347
  
- tibble::as\_tibble(), 243, 348, 349
- tibble::as\_tibble.table(), 349
- tibble::tibble, 10, 13, 15, 17, 20, 22, 24, 26, 28, 30, 31, 33, 35, 37, 39, 43, 44, 48, 50, 53, 56, 59, 61, 63, 65, 67, 69, 70, 72, 74, 76, 78, 81, 83, 84, 136, 194, 197, 231, 238, 244, 313, 349, 353, 357, 359
- tibble::tibble(), 11, 13, 14, 16, 18, 23, 25, 27, 29, 30, 32–36, 38, 40, 43, 45–49, 51, 52, 54, 57–66, 68–71, 73, 75, 77, 79, 80, 82, 85, 91, 93–108, 110–115, 117–130, 132, 134, 135, 137, 138, 140–150, 152–163, 165–174, 176–180, 182–192, 195, 198, 200, 201, 203, 204, 206, 207, 209, 210, 212, 213, 215, 217, 219, 221, 222, 224, 225, 227, 229, 233, 234, 236, 239, 241, 242, 245–247, 249, 250, 253, 255, 257, 259, 261, 263–265, 267, 268, 270, 272, 274, 278, 280–282, 284, 286, 288, 290, 292–294, 296–299, 302, 303, 305, 306, 308, 310, 311, 315, 316, 318, 320, 321, 323, 324, 326–328,



- 330–333, 335, 337, 339–341, 343, 346, 347, 350–352, 355
- tidy, 16, 60, 106, 107, 158, 166, 191, 219, 221, 300, 310, 346
- tidy(), 68, 91, 119, 199–203, 208, 209, 211–213, 215–217, 223–225, 227, 229, 233, 235, 237–239, 241, 242, 245, 247, 249, 250, 253, 257, 259, 261, 263, 264, 266–268, 270, 274, 278, 280, 281, 283, 284, 286, 288, 290, 292, 293, 295–297, 299, 303, 305, 307, 308, 315, 317, 318, 320, 321, 324, 326, 328–330, 334, 335, 337, 339, 340, 342, 343, 348, 350–352, 354, 356, 359
- tidy.aareg, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 198, 216, 227, 315, 339, 340, 342, 343
- tidy.acf, 199, 330, 350, 352
- tidy.anova, 96, 200, 202, 203, 280, 351
- tidy.aov, 96, 201, 201, 203, 280, 351
- tidy.aovlist, 96, 201, 202, 202, 280, 351
- tidy.Arima, 97, 204
- tidy.betamfx, 12, 52, 98, 149, 205, 288
- tidy.betareg, 207
- tidy.betareg(), 206
- tidy.biglm, 101, 208
- tidy.binDesign, 103, 210, 212
- tidy.binWidth, 103, 211, 211
- tidy.boot, 212
- tidy.btergm, 214
- tidy.cch, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 215, 227, 315, 339, 340, 342, 343
- tidy.character (tidy.numeric), 300
- tidy.cld, 217, 224, 250, 334
- tidy.clm, 16, 68, 106, 107, 166, 191, 218, 221, 310, 346
- tidy.clm, 16, 68, 106, 107, 166, 191, 219, 220, 310, 346
- tidy.coeftest, 222
- tidy.confint.glm, 217, 223, 250, 334
- tidy.confusionMatrix, 225
- tidy.coxph, 18, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 216, 226, 315, 339, 340, 342, 343
- tidy.cv.glmnet, 113, 127, 228, 253
- tidy.data.frame (data.frame\_tidiers), 89
- tidy.density, 88, 90, 92, 93, 197, 230, 232, 244, 246, 301
- tidy.dgCMatrix (sparse\_tidiers), 194
- tidy.dgTMatrix (sparse\_tidiers), 194
- tidy.dist, 88, 90, 92, 93, 197, 231, 231, 244, 246, 301
- tidy.drc, 24, 115, 232
- tidy.durbinWatsonTest (durbinWatsonTest\_tidiers), 91
- tidy.emmGrid, 234, 278, 318, 337
- tidy.epi.2by2, 236
- tidy.ergm, 116, 237
- tidy.factanal, 25, 118, 239
- tidy.fitdistr, 119, 240
- tidy.fixest, 27, 241
- tidy.ftable, 88, 90, 92, 93, 197, 231, 232, 243, 246, 301
- tidy.gam, 122, 244
- tidy.gamlss, 88, 90, 92, 93, 197, 231, 232, 244, 245, 301
- tidy.garch, 123, 247
- tidy.geeglm, 248
- tidy.glm, 217, 224, 250, 334
- tidy.glm, 30, 41, 126, 138, 181, 190, 251, 270, 272, 293, 335
- tidy.glmnet, 113, 127, 229, 252
- tidy.glmRob, 43, 129, 142, 254, 275
- tidy.glmrob, 33, 45, 143, 255, 276
- tidy.gmm, 130, 256
- tidy.htest, 34, 258, 303, 312
- tidy.irlba (tidy\_irlba), 353
- tidy.ivreg, 36, 133, 260
- tidy.kappa, 262
- tidy.kde, 263
- tidy.Kendall, 265
- tidy.kmeans, 38, 134, 266
- tidy.lavaan, 136, 267
- tidy.Line (sp\_tidiers), 195
- tidy.Lines (sp\_tidiers), 195
- tidy.list (list\_tidiers), 193
- tidy.lm, 30, 41, 126, 138, 181, 190, 252, 269, 272, 293, 335
- tidy.lm(), 307, 333, 335
- tidy.lm.beta, 30, 41, 126, 138, 181, 190, 252, 270, 271, 293, 335
- tidy.lmodel2, 141, 273
- tidy.lmRob, 43, 129, 142, 254, 275
- tidy.lmrob, 33, 45, 143, 256, 276

- tidy.logical (tidy.numeric), 300
- tidy.logitmfx (tidy.mfx), 287
- tidy.lsmobj, 235, 277, 318, 337
- tidy.manova, 96, 201–203, 279, 351
- tidy.map, 280
- tidy.margins, 282
- tidy.Mclust, 49, 284
- tidy.mediate, 285
- tidy.mfx, 12, 52, 98, 149, 206, 287
- tidy.mjoint, 151, 289
- tidy.mle2, 291
- tidy.mlm, 30, 41, 126, 138, 181, 190, 252, 270, 272, 292, 335
- tidy.mlm(), 269
- tidy.mlogit, 57, 153, 294
- tidy.muhaz, 154, 295
- tidy.multinom, 155, 296
- tidy.negbinmfx (tidy.mfx), 287
- tidy.nlrq, 58, 76, 77, 157, 173, 298, 326, 328
- tidy.nls, 60, 158, 299
- tidy.NULL (null\_tidiers), 194
- tidy.numeric, 88, 90, 92, 93, 197, 231, 232, 244, 246, 300
- tidy.optim (tidy\_optim), 355
- tidy.orcutt, 159, 301
- tidy.pairwise.htest, 34, 259, 303, 312
- tidy.pam, 62, 161, 304
- tidy.plm, 64, 162, 306
- tidy.poissonmfx (tidy.mfx), 287
- tidy.poLCA, 66, 164, 307
- tidy.polr, 16, 68, 106, 107, 166, 191, 219, 221, 309, 346
- tidy.Polygon (sp\_tidiers), 195
- tidy.Polygons (sp\_tidiers), 195
- tidy.power.htest, 34, 259, 303, 311
- tidy.prcomp, 70, 312, 354, 357
- tidy.probitmfx (tidy.mfx), 287
- tidy.pyears, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 216, 227, 314, 339, 340, 342, 343
- tidy.rcorr, 316
- tidy.ref.grid, 235, 278, 317, 337
- tidy.regsubsets, 319
- tidy.ridgelm, 169, 320
- tidy.rlm, 72, 170, 321
- tidy.rlm(), 32, 43, 45, 143, 254, 255, 275, 276
- tidy.rma, 322
- tidy.roc, 324
- tidy.rq, 58, 76, 77, 157, 173, 299, 325, 328
- tidy.rqs, 58, 76, 77, 157, 173, 299, 326, 327
- tidy.sarlm, 79, 175, 328
- tidy.sparseMatrix (sparse\_tidiers), 194
- tidy.SpatialLinesDataFrame (sp\_tidiers), 195
- tidy.SpatialPolygons (sp\_tidiers), 195
- tidy.SpatialPolygonsDataFrame (sp\_tidiers), 195
- tidy.spec, 200, 329, 350, 352
- tidy.speedglm, 82, 178, 179, 330, 333
- tidy.speedlm, 82, 178, 179, 331, 332
- tidy.summary.glm, 217, 224, 250, 333
- tidy.summary.lm, 30, 41, 126, 138, 181, 190, 252, 270, 272, 293, 334
- tidy.summary\_emm, 235, 278, 318, 336
- tidy.summaryDefault (summary\_tidiers), 196
- tidy.survdif, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 216, 227, 315, 338, 340, 342, 343
- tidy.survep, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 216, 227, 315, 339, 339, 342, 343
- tidy.survfit, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 216, 227, 315, 339, 340, 341, 343
- tidy.survreg, 19, 86, 94, 104, 111, 167, 183, 185, 186, 188, 199, 216, 227, 315, 339, 340, 342, 342
- tidy.svgglm, 344
- tidy.svyolr, 16, 68, 106, 107, 166, 191, 219, 221, 310, 345
- tidy.systemfit, 347
- tidy.table, 348
- tidy.ts, 200, 330, 349, 352
- tidy.TukeyHSD, 96, 201–203, 280, 350
- tidy.zoo, 200, 330, 350, 351
- tidy\_irlba, 70, 192, 193, 313, 353, 356, 357, 359
- tidy\_optim, 192, 193, 354, 355, 357, 359
- tidy\_optim(), 292
- tidy\_svd, 70, 192, 193, 313, 354, 356, 356, 359
- tidy\_svd(), 353
- tidy\_xyz, 192, 193, 354, 356, 357, 358
- tidyr::pivot\_longer(), 349

`tseries::garch()`, [123](#), [247](#)

`xyz_tidiers (tidy_xyz)`, [358](#)

`zoo::zoo()`, [352](#)

`zoo_tidiers (tidy.zoo)`, [351](#)