Package ‘modelr’

February 22, 2020

Title Modelling Functions that Work with the Pipe

Version 0.1.6

Description Functions for modelling that help you seamlessly integrate modelling into a pipeline of data manipulation and visualisation.

License GPL-3


BugReports https://github.com/tidyverse/modelr/issues

Depends R (>= 3.2)

Imports broom, dplyr, magrittr, purrr (>= 0.2.2), rlang (>= 0.2.0), tibble, tidyr (>= 0.8.0), tidyselect

Suggests compiler, covr, ggplot2, testthat

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

NeedsCompilation no

Author Hadley Wickham [aut, cre], RStudio [cph]

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2020-02-22 21:50:02 UTC

R topics documented:

- add_predictions .......................................................... 2
- add_predictors ........................................................... 3
- add_residuals ............................................................. 4
- bootstrap ................................................................. 5
- crossv_mc ................................................................. 5
- data_grid ................................................................. 6
Add predictions to a data frame

Description

Add predictions to a data frame

Usage

add_predictions(data, model, var = "pred", type = NULL)

spread_predictions(data, ..., type = NULL)

gather_predictions(data, ..., .pred = "pred", .model = "model", type = NULL)

Arguments

data A data frame used to generate the predictions.
model add_predictions takes a single model:
var The name of the output column, default value is pred
type Prediction type, passed on to stats::predict(). Consult predict() documentation for given model to determine valid values.
... gather_predictions and spread_predictions take multiple models. The name will be taken from either the argument name of the name of the model.
.pred, .model The variable names used by gather_predictions.
Value

A data frame. add_prediction adds a single new column, with default name pred, to the input data. spread_predictions adds one column for each model. gather_predictions adds two columns .model and .pred, and repeats the input rows for each model.

Examples

df <- tibble::data_frame(
    x = sort(runif(100)),
    y = 5 * x + 0.5 * x ^ 2 + 3 + rnorm(length(x))
)
plot(df)

m1 <- lm(y ~ x, data = df)
grid <- data.frame(x = seq(0, 1, length = 10))
grid %>% add_predictions(m1)

m2 <- lm(y ~ poly(x, 2), data = df)
grid %>% spread_predictions(m1, m2)
grid %>% gather_predictions(m1, m2)

add_predictors

Add predictors to a formula

Description

This merges a one- or two-sided formula f with the right-hand sides of all formulas supplied in ....

Usage

add_predictors(f, ..., fun = "+")

Arguments

f
A formula.

... Formulas whose right-hand sides will be merged to f.

fun A function name indicating how to merge the right-hand sides.

Examples

f <- lhs ~ rhs
add_predictors(f, ~var1, ~var2)

# Left-hand sides are ignored:
add_predictors(f, lhs1 ~ var1, lhs2 ~ var2)

# fun can also be set to a function like "+":
add_predictors(f, ~var1, ~var2, fun = "+")
add_residuals

Add residuals to a data frame

Description

Add residuals to a data frame

Usage

```r
add_residuals(data, model, var = "resid")
spread_residuals(data, ...)
gather_residuals(data, ..., .resid = "resid", .model = "model")
```

Arguments

- `data`: A data frame used to generate the residuals
- `model, var`: `add_residuals` takes a single model; the output column will be called `resid`
- `...`: `gather_residuals` and `spread_residuals` take multiple models. The name will be taken from either the argument name of the name of the model.
- `.resid, .model`: The variable names used by `gather_residuals`.

Value

A data frame. `add_residuals` adds a single new column, `.resid`, to the input data. `spread_residuals` adds one column for each model. `gather_predictions` adds two columns `.model` and `.resid`, and repeats the input rows for each model.

Examples

```r
df <- tibble::data_frame(
  x = sort(runif(100)),
  y = 5 * x + 0.5 * x ^ 2 + 3 + rnorm(length(x))
)
plot(df)

m1 <- lm(y ~ x, data = df)
df %>% add_residuals(m1)

m2 <- lm(y ~ poly(x, 2), data = df)
df %>% spread_residuals(m1, m2)
df %>% gather_residuals(m1, m2)
```
**bootstrap**

Generate \( n \) bootstrap replicates.

**Usage**

```r
bootstrap(data, n, id = ".id")
```

**Arguments**

- `data`: A data frame
- `n`: Number of bootstrap replicates to generate
- `id`: Name of variable that gives each model a unique integer id.

**Value**

A data frame with \( n \) rows and one column: `strap`

**See Also**

Other resampling techniques: `resample_bootstrap()`, `resample_partition()`, `resample()`

**Examples**

```r
library(purrr)
boot <- bootstrap(mtcars, 100)
models <- map(boot$strap, ~ lm(mpg ~ wt, data = .))
tidied <- map_df(models, broom::tidy, .id = "id")

hist(subset(tidied, term == "wt")$estimate)
hist(subset(tidied, term == "(Intercept)")$estimate)
```

**crossv_mc**

Generate test-training pairs for cross-validation

**Description**

crossv_kfold splits the data into \( k \) exclusive partitions, and uses each partition for a test-training split. `crossv_mc` generates \( n \) random partitions, holding out test of the data for training. `crossv_loo` performs leave-one-out cross-validation, i.e., \( n = nrow(data) \) training partitions containing \( n-1 \) rows each.
Usage

crossv_mc(data, n, test = 0.2, id = ".id")
crossv_kfold(data, k = 5, id = ".id")
crossv_loo(data, id = ".id")

Arguments

data A data frame
n Number of test-training pairs to generate (an integer).
test Proportion of observations that should be held out for testing (a double).
id Name of variable that gives each model a unique integer id.
k Number of folds (an integer).

Value

A data frame with columns test, train, and .id. test and train are list-columns containing resample() objects. The number of rows is n for crossv_mc(), k for crossv_kfold() and nrow(data) for crossv_loo().

Examples

cv1 <- crossv_kfold(mtcars, 5)
cv1

library(purrr)
cv2 <- crossv_mc(mtcars, 100)
models <- map(cv2$train, ~ lm(mpg ~ wt, data = .))
errs <- map2_dbl(models, cv2$test, rmse)
hist(errs)

data_grid Generate a data grid.

Description

To visualise a model, it is very useful to be able to generate an evenly spaced grid of points from the data. data_grid helps you do this by wrapping around tidyr::expand().

Usage

data_grid(data, ..., .model = NULL)
Arguments

- `data` A data frame
- `...` Variables passed on to `tidyr::expand()`
- `.model` A model. If supplied, any predictors needed for the model not present in ... will be filled in with "typical" values.

See Also

- `seq_range()` for generating ranges from continuous variables.

Examples

```r
data_grid(mtcars, vs, am)

# For continuous variables, seq_range is useful
data_grid(mtcars, mpg = mpg)
data_grid(mtcars, mpg = seq_range(mpg, 10))

# If you supply a model, missing predictors will be filled in with
# typical values
mod <- lm(mpg ~ wt + cyl + vs, data = mtcars)
data_grid(mtcars, .model = mod)
data_grid(mtcars, cyl = seq_range(cyl, 9), .model = mod)
```

Description

`fit_with()` is a pipe-friendly tool that applies a list of formulas to a fitting function such as `stats::lm()`. The list of formulas is typically created with `formulas()`.

Usage

```r
fit_with(data, .f, .formulas, ...)
```

Arguments

- `data` A dataset used to fit the models.
- `.f` A fitting function such as `stats::lm()`, `lme4::lmer()` or `rstanarm::stan_glmer()`.
- `.formulas` A list of formulas specifying a model.
- `...` Additional arguments passed on to `.f`

Details

Assumes that `.f` takes the formula either as first argument or as second argument if the first argument is `data`. Most fitting functions should fit these requirements.
See Also

formulas()

Examples

# fit_with() is typically used with formulas().
disp_fits <- mtcars %>% fit_with(lm, formulas(~disp,
   additive = ~drat + cyl,
   interaction = ~drat * cyl,
   full = add_predictors(interaction, ~am, ~vs)
))

# The list of fitted models is named after the names of the list of
# formulas:
disp_fits$full

# Additional arguments are passed on to .f
mtcars %>% fit_with(glm, list(am ~ disp), family = binomial)

formulas

Create a list of formulas

Description

formulas() creates a list of two-sided formulas by merging a unique left-hand side to a list of right-hand sides.

Usage

formulas(.response, ...)
formulae(.response, ...)

Arguments

.response A one-sided formula used as the left-hand side of all resulting formulas.
...
List of formulas whose right-hand sides will be merged to .response.

Examples

# Provide named arguments to create a named list of formulas:
models <- formulas(~lhs,
   additive = ~var1 + var2,
   interaction = ~var1 * var2
)
models$additive

# The formulas are created sequentially, so that you can refer to
# previously created formulas:
Add a reference line (ggplot2).

**Usage**

```r
geom_ref_line(h, v, size = 2, colour = "white")
```

**Arguments**

- `h, v` Position of horizontal or vertical reference line
- `size` Line size
- `colour` Line colour

**Description**

You might have heard that taller people earn more. Is it true? You can try and answer the question by exploring this dataset extracted from the National Longitudinal Study, which is sponsored by the U.S. Bureau of Labor Statistics.

**Format**

- `income` Yearly income. The top two percent of values were averaged and that average was used to replace all values in the top range.
- `height` Height, in feet
- `weight` Weight, in pounds
- `age` Age, in years, between 47 and 56.
- `marital` Marital status
- `sex` Sex
- `education` Years of education
- `afqt` Percentile score on Armed Forces Qualification Test.
Details

This contains data as at 2012.

---

**model-quality**

*Compute model quality for a given dataset*

Description

Three summaries are immediately interpretable on the scale of the response variable:

- `rmse()` is the root-mean-squared-error
- `mae()` is the mean absolute error
- `qae()` is quantiles of absolute error.

Other summaries have varying scales and interpretations:

- `mape()` mean absolute percentage error.
- `rsae()` is the relative sum of absolute errors.
- `mse()` is the mean-squared-error.
- `rsquare()` is the variance of the predictions divided by the variance of the response.

Usage

```r
mse(model, data)
rmse(model, data)
mae(model, data)
rsquare(model, data)
qae(model, data, probs = c(0.05, 0.25, 0.5, 0.75, 0.95))
mape(model, data)
rsae(model, data)
```

Arguments

- **model** A model
- **data** The dataset
- **probs** Numeric vector of probabilities
model_matrix

Examples

```r
mod <- lm(mpg ~ wt, data = mtcars)
mse(mod, mtcars)
rmse(mod, mtcars)
rsquare(mod, mtcars)
mae(mod, mtcars)
qae(mod, mtcars)
mape(mod, mtcars)
rsae(mod, mtcars)
```

---

**model_matrix**

Construct a design matrix

Description

This is a thin wrapper around `stats::model.matrix()` which returns a tibble. Use it to determine how your modelling formula is translated into a matrix, thence into an equation.

Usage

```r
model_matrix(data, formula, ...)
```

Arguments

- `data` A data frame
- `formula` A modelling formula
- `...` Other arguments passed onto `stats::model.matrix()`

Value

A tibble.

Examples

```r
model_matrix(mtcars, mpg ~ cyl)
model_matrix(iris, Sepal.Length ~ Species)
model_matrix(iris, Sepal.Length ~ Species - 1)
```
na.warn  
*Handle missing values with a warning*

**Description**

This NA handler ensures that those models that support the `na.action` parameter do not silently drop missing values. It wraps around `stats::na.exclude()` so that there is one prediction/residual for input row. To apply it globally, run `options(na.action = na.warn)`.

**Usage**

```r
na.warn(object)
```

**Arguments**

- `object`  
  A data frame

**Examples**

```r
df <- tibble::tibble(
x = 1:10,
y = c(5.1, 9.7, NA, 17.4, 21.2, 26.6, 27.9, NA, 36.3, 40.4)
)
# Default behaviour is to silently drop
m1 <- lm(y ~ x, data = df)
resid(m1)

# Use na.action = na.warn to warn
m2 <- lm(y ~ x, data = df, na.action = na.warn)
resid(m2)
```

**permute**  
*Generate n permutation replicates.*

**Description**

A permutation test involves permuting one or more variables in a data set before performing the test, in order to break any existing relationships and simulate the null hypothesis. One can then compare the true statistic to the generated distribution of null statistics.

**Usage**

```r
permute(data, n, ..., .id = ".id")
permute_(data, n, columns, .id = ".id")
```
Arguments

- **data**: A data frame
- **n**: Number of permutations to generate.
- **...**: Columns to permute. This supports bare column names or dplyr dplyr::select_helpers
- **.id**: Name of variable that gives each model a unique integer id.
- **columns**: In permute_, vector of column names to permute.

Value

A data frame with n rows and one column: `perm`

Examples

```r
library(purrr)
perms <- permute(mtcars, 1000, mpg)

models <- map(perms$perm, ~ lm(mpg ~ wt, data = .))
glanced <- map_df(models, broom::glance, .id = "id")

# distribution of null permutation statistics
hist(glanced$statistic)
# confirm these are roughly uniform p-values
hist(glanced$p.value)

# test against the unpermuted model to get a permutation p-value
mod <- lm(mpg ~ wt, mtcars)
mean(glanced$statistic > broom::glance(mod)$statistic)
```

---

**resample**

*A "lazy" resample.*

Description

Often you will resample a dataset hundreds or thousands of times. Storing the complete resample each time would be very inefficient so this class instead stores a "pointer" to the original dataset, and a vector of row indexes. To turn this into a regular data frame, call `as.data.frame`, to extract the indices, use `as.integer`.

Usage

`resample(data, idx)`
resample_bootstrap

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>The data frame</td>
</tr>
<tr>
<td>idx</td>
<td>A vector of integer indexes indicating which rows have been selected. These values should lie between 1 and nrow(data) but they are not checked by this function in the interests of performance.</td>
</tr>
</tbody>
</table>

See Also

Other resampling techniques: bootstrap(), resample_bootstrap(), resample_partition()

Examples

resample(mtcars, 1:10)

b <- resample_bootstrap(mtcars)
b
as.integer(b)
as.data.frame(b)

# Many modelling functions will do the coercion for you, so you can
# use a resample object directly in the data argument
lm(mpg ~ wt, data = b)

resample_bootstrap Generate a bootstrap replicate

Description

Generate a bootstrap replicate

Usage

resample_bootstrap(data)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>A data frame</td>
</tr>
</tbody>
</table>

See Also

Other resampling techniques: bootstrap(), resample_partition(), resample()

Examples

coef(lm(mpg ~ wt, data = resample_bootstrap(mtcars)))

coef(lm(mpg ~ wt, data = resample_bootstrap(mtcars)))

coef(lm(mpg ~ wt, data = resample_bootstrap(mtcars)))

coef(lm(mpg ~ wt, data = resample_bootstrap(mtcars)))
resample_partition

Generate an exclusive partitioning of a data frame

Description

Generate an exclusive partitioning of a data frame

Usage

resample_partition(data, p)

Arguments

data: A data frame
p: A named numeric vector giving where the value is the probability that an observation will be assigned to that group.

See Also

Other resampling techniques: bootstrap(), resample_bootstrap(), resample()

Examples

ex <- resample_partition(mtcars, c(test = 0.3, train = 0.7))
mod <- lm(mpg ~ wt, data = ex$train)
rmse(mod, ex$test)
rmse(mod, ex$train)

resample_permutation

Create a resampled permutation of a data frame

Description

Create a resampled permutation of a data frame

Usage

resample_permutation(data, columns, idx = NULL)

Arguments

data: A data frame
columns: Columns to be permuted
idx: Indices to permute by. If not given, generates them randomly

Value

A permutation object; use as.data.frame to convert to a permuted data frame
seq_range

Generate a sequence over the range of a vector

Description
Generate a sequence over the range of a vector

Usage
seq_range(x, n, by, trim = NULL, expand = NULL, pretty = FALSE)

Arguments
x
A numeric vector

n, by
Specify the output sequence either by supplying the length of the sequence with
n, or the spacing between value with by. Specifying both is an error.
I recommend that you name these arguments in order to make it clear to the reader.

trim
Optionally, trim values off the tails. \( \text{trim} / 2 \times \text{length}(x) \) values are removed from each tail.

expand
Optionally, expand the range by \( \text{expand} \times (1 + \text{range}(x)) \) (computed after trimming).

pretty
If TRUE, will generate a pretty sequence. If n is supplied, this will use \text{pretty()} instead of \text{seq()}. If by is supplied, it will round the first value to a multiple of by.

Examples
x <- rcauchy(100)
seq_range(x, n = 10)
seq_range(x, n = 10, trim = 0.1)
seq_range(x, by = 1, trim = 0.1)

# Make pretty sequences
y <- runif(100)
seq_range(y, n = 10)
seq_range(y, n = 10, pretty = TRUE)
seq_range(y, n = 10, expand = 0.5, pretty = TRUE)

seq_range(y, by = 0.1)
seq_range(y, by = 0.1, pretty = TRUE)
**Description**

These simple simulated datasets are useful for teaching modelling basics.

**Usage**

```r
sim1
sim2
sim3
sim4
```

**Format**

An object of class tbl_df (inherits from tbl.data.frame) with 30 rows and 2 columns.

---

**typical**

*Find the typical value*

---

**Description**

For numeric, integer, and ordered factor vectors, it returns the median. For factors, characters, and logical vectors, it returns the most frequent value. If multiple values are tied for most frequent, it returns them all. NA missing values are always silently dropped.

**Usage**

```r
typical(x, ...)
```

**Arguments**

- `x` A vector
- `...` Arguments used by methods
Examples

# median of numeric vector
typical(rpois(100, lambda = 10))

# most frequent value of character or factor
x <- sample(c("a", "b", "c"), 100, prob = c(0.6, 0.2, 0.2), replace = TRUE)
typical(x)
typical(factor(x))

# if tied, returns them all
x <- c("a", "a", "b", "b", "c")
typical(x)

# median of an ordered factor
typical(ordered(c("a", "a", "b", "c", "d"))))
Index

*Topic datasets
  heights, 9
  sim, 17
add_predictions, 2
add_predictors, 3
add_residuals, 4
bootstrap, 5, 14, 15
crossv_kfold(crossv_mc), 5
crossv_loo(crossv_mc), 5
crossv_mc, 5
data_grid, 6
dplyr::select_helpers, 13
fit_with, 7
formulæ(formulas), 8
formulas, 8
formulas(), 7, 8
gather_predictions(add_predictions), 2
gather_residuals(add_residuals), 4
geom_ref_line, 9
heights, 9
lme4::lmer(), 7
mae(model-quality), 10
mape(model-quality), 10
model-quality, 10
model_matrix, 11
mse(model-quality), 10
na.warn, 12
permute, 12
permute_(permute), 12
pretty(), 16
qae(model-quality), 10
resample, 5, 13, 14, 15
resample(), 6
resample_bootstrap, 5, 14, 14, 15
resample_partition, 5, 14, 15
resample_permutation, 15
rmse(model-quality), 10
rsae(model-quality), 10
rsquare(model-quality), 10
rstanarm::stan_glmer(), 7
seq(), 16
seq_range, 16
seq_range(), 7
sim, 17
sim1(sim), 17
sim2(sim), 17
sim3(sim), 17
sim4(sim), 17
spread_predictions(add_predictions), 2
spread_residuals(add_residuals), 4
stats::lm(), 7
stats::model.matrix(), 11
stats::na.exclude(), 12
tidyr::expand(), 6, 7
typical, 7, 17