Package 'tibble'

January 22, 2018

Title Simple Data Frames

Version 1.4.2

Description Provides a 'tbl_df' class (the 'tibble') that provides
stricter checking and better formatting than the traditional data frame.

License MIT + file LICENSE

URL http://tibble.tidyverse.org/, https://github.com/tidyverse/tibble

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

Depends R (>= 3.1.0)

Imports cli, crayon, methods, pillar (>= 1.1.0), rlang, utils

Suggests covr, dplyr, import, knitr (>= 1.5.32), microbenchmark,
mockr, nycflights13, rmarkdown, testthat, withr

VignetteBuilder knitr

Encoding UTF-8

LazyData yes

RoxygenNote 6.0.1

Collate 'add.R' 'as_tibble.R' 'check-names.R' 'compat-lazyeval.R'
'compat-purrr.R' 'enframe.R' 'exports.R' 'glimpse.R'
'has-name.R' 'tibble.R' 'lst.R' 'new.R' 'repair-names.R'
'rownames.R' 'strrep.R' 'tbl-df.r' 'tibble-package.R'
'tribble.R' 'type-sum.r' 'utils-format.r' 'utils.r' 'wrap.R'

NeedsCompilation yes

Author Kirill Müller [aut, cre],
Hadley Wickham [aut],
Romain Francois [ctb],
RStudio [cph]

Maintainer Kirill Müller <krlmlr+r@mailbox.org>

Repository CRAN

Date/Publication 2018-01-22 19:41:48 UTC

R topics documented:

`tibble-package` *tibble: Simple Data Frames*

Description

Provides a 'tbl_df' class (the 'tibble') that provides stricter checking and better formatting than the traditional data frame.

Details

The S3 class `tbl_df` wraps a local data frame. The main advantage to using a `tbl_df` over a regular data frame is the printing: tbl objects only print a few rows and all the columns that fit on one screen, describing the rest of it as text.

Methods

`tbl_df` implements four important base methods:

print By default only prints the first 10 rows (at most 20), and the columns that fit on screen; see `print.tbl()`

`[` Does not simplify (drop) by default, returns a data frame

`[[`, `$` Calls `.subset2()` directly, so is considerably faster. Returns `NULL` if column does not exist, `$` warns.

Important functions

`tibble()` and `tribble()` for construction, `as_tibble()` for coercion, and `print.tbl()` and `glimpse()` for display.

Author(s)

Maintainer: Kirill Müller <krlmlr+r@mailbox.org>

Authors:

- Hadley Wickham <hadley@rstudio.com>

Other contributors:

- Romain Francois <romain@r-enthusiasts.com> [contributor]
- RStudio [copyright holder]

See Also

Useful links:

- http://tibble.tidyverse.org/
- https://github.com/tidyverse/tibble
- Report bugs at https://github.com/tidyverse/tibble/issues

Examples

```
tibble(a = 1:26, b = letters)
as_tibble(iris)
```

---

add_column *Add columns to a data frame*

Description

This is a convenient way to add one or more columns to an existing data frame.

Usage

```
add_column(.data, ..., .before = NULL, .after = NULL)
```

Arguments

`.data`	Data frame to append to.
`...`	Name-value pairs, passed on to `tibble()`. All values must have one element for each row in the data frame, or be of length 1. These arguments are passed on to `tibble()`, and therefore also support unquote via `!!` and unquote-splice via `!!!`.
`.before, .after`	One-based column index or column name where to add the new columns, default: after last column.

See Also

Other addition: add_row

Examples

```
# add_column ---------------------------------
df <- tibble(x = 1:3, y = 3:1)

add_column(df, z = -1:1, w = 0)

# You can't overwrite existing columns
## Not run:
add_column(df, x = 4:6)

## End(Not run)
# You can't create new observations
## Not run:
add_column(df, z = 1:5)

## End(Not run)
```

---

add_row *Add rows to a data frame*

Description

This is a convenient way to add one or more rows of data to an existing data frame. See tribble() for an easy way to create an complete data frame row-by-row.

Usage

```
add_row(.data, ..., .before = NULL, .after = NULL)
```

Arguments

.data	Data frame to append to.
...	Name-value pairs, passed on to tibble(). Only columns that exist in .data can be used, unset columns will get an NA value. These arguments are passed on to tibble(), and therefore also support unquote via !! and unquote-splice via !!!.
.before, .after	One-based row index where to add the new rows, default: after last row.

Details

add_case() is an alias of add_row().

See Also

Other addition: add_column

Examples

```
# add_row ---------------------------------
df <- tibble(x = 1:3, y = 3:1)

add_row(df, x = 4, y = 0)

# You can specify where to add the new rows
add_row(df, x = 4, y = 0, .before = 2)

# You can supply vectors, to add multiple rows (this isn't
# recommended because it's a bit hard to read)
add_row(df, x = 4:5, y = 0:-1)

# Absent variables get missing values
add_row(df, x = 4)

# You can't create new variables
## Not run:
add_row(df, z = 10)

## End(Not run)
```

---

as_tibble *Coerce lists and matrices to data frames*

Description

as.data.frame() is effectively a thin wrapper around data.frame, and hence is rather slow (because it calls data.frame() on each element before cbinding together). as_tibble is a new S3 generic with more efficient methods for matrices and data frames.

Usage

```
as_tibble(x, ...)

## S3 method for class 'tbl_df'
as_tibble(x, ..., validate = FALSE, rownames = NULL)

## S3 method for class 'data.frame'
as_tibble(x, validate = TRUE, ..., rownames = NA)

## S3 method for class 'list'
as_tibble(x, validate = TRUE, ...)
```

```
## S3 method for class 'matrix'
as_tibble(x, ..., rownames = NULL)

## S3 method for class 'table'
as_tibble(x, n = "n", ...)

## S3 method for class 'NULL'
as_tibble(x, ...)

## Default S3 method:
as_tibble(x, ...)
```

Arguments

x	A list. Each element of the list must have the same length.
...	Other arguments passed on to individual methods.
validate	When TRUE, verifies that the input is a valid data frame (i.e. all columns are named, and are 1d vectors or lists). You may want to suppress this when you know that you already have a valid data frame and you want to save some time, or to explicitly enable it if you have a tibble that you want to re-check.
rownames	If NULL, remove row names (default for matrices, may become default for data frames in the future). If NA, keep row names (current default for data frames). Otherwise, the name of the new column that will contain the existing row names.
n	Name for count column, default: "n".

Details

This is an S3 generic. tibble includes methods for data frames (adds tbl_df classes), tibbles (returns unchanged input), lists, matrices, and tables. Other types are first coerced via `as.data.frame()` with `stringsAsFactors = FALSE`.

`as_data_frame` and `as.tibble` are aliases.

Examples

```
l <- list(x = 1:500, y = runif(500), z = 500:1)
df <- as_tibble(l)

m <- matrix(rnorm(50), ncol = 5)
colnames(m) <- c("a", "b", "c", "d", "e")
df <- as_tibble(m)

# as_tibble is considerably simpler than as.data.frame
# making it more suitable for use when you have things that are
# lists
## Not run:
if (requireNamespace("microbenchmark", quiet = TRUE)) {
  l2 <- replicate(26, sample(letters), simplify = FALSE)
  names(l2) <- letters
  microbenchmark::microbenchmark(
```

```
    as_tibble(l2, validate = FALSE),
    as_tibble(l2),
    as.data.frame(l2)
  )
}

if (requireNamespace("microbenchmark", quiet = TRUE)) {
  m <- matrix(runif(26 * 100), ncol = 26)
  colnames(m) <- letters
  microbenchmark::microbenchmark(
    as_tibble(m),
    as.data.frame(m)
  )
}

## End(Not run)
```

---

enframe *Converting atomic vectors to data frames, and vice versa*

Description

`enframe()` converts named atomic vectors or lists to two-column data frames. For unnamed vectors, the natural sequence is used as name column.

`deframe()` converts two-column data frames to a named vector or list, using the first column as name and the second column as value.

Usage

```
enframe(x, name = "name", value = "value")

deframe(x)
```

Arguments

`x`	An atomic vector (for `enframe()`) or a data frame (for `deframe()`)
`name, value`	Names of the columns that store the names and values

Value

A tibble

Examples

```
enframe(1:3)
enframe(c(a = 5, b = 7))
```

frame_matrix *Row-wise matrix creation*

Description

Create matrices laying out the data in rows, similar to `matrix(..., byrow = TRUE)`, with a nicer-to-read syntax. This is useful for small matrices, e.g. covariance matrices, where readability is important. The syntax is inspired by `tribble()`.

Usage

```
frame_matrix(...)
```

Arguments

`...`	Arguments specifying the structure of a `frame_matrix`. Column names should be formulas, and may only appear before the data.

Value

A matrix.

Examples

```
frame_matrix(
  ~col1, ~col2,
  1,     3,
  5,     2
)
```

glimpse *Get a glimpse of your data*

Description

This is like a transposed version of `print()`: columns run down the page, and data runs across. This makes it possible to see every column in a data frame. It's a little like `str()` applied to a data frame but it tries to show you as much data as possible. (And it always shows the underlying data, even when applied to a remote data source.)

Usage

```
glimpse(x, width = NULL, ...)
```

Arguments

x	An object to glimpse at.
width	Width of output: defaults to the setting of the option tibble.width (if finite) or the width of the console.
...	Other arguments passed on to individual methods.

Value

x original x is (invisibly) returned, allowing glimpse() to be used within a data pipe line.

S3 methods

glimpse is an S3 generic with a customised method for tbls and data.frames, and a default method that calls str().

Examples

```
glimpse(mtcars)

if (!requireNamespace("nycflights13", quietly = TRUE))
  stop("Please install the nycflights13 package to run the rest of this example")

glimpse(nycflights13::flights)
```

---

is_tibble *Test if the object is a tibble*

Description

This function returns FALSE for regular data frames and TRUE for tibbles.

Usage

```
is_tibble(x)
```

Arguments

x	An object

Value

TRUE if the object inherits from the tbl_df class.

`new_tibble` *Constructor*

Description

Creates a subclass of a tibble. This function is mostly useful for package authors that implement subclasses of a tibble, like **sf** or **tibbletime**.

Usage

```
new_tibble(x, ..., nrow = NULL, subclass = NULL)
```

Arguments

`x`	A tibble-like object
`...`	Passed on to `structure()`
`nrow`	The number of rows, guessed from the data by default
`subclass`	Subclasses to assign to the new object, default: none

Details

`x` must be a named (or empty) list, but the names are not currently checked for correctness.

The `...` argument allows adding more attributes to the subclass.

The `row.names` attribute will be computed from the `nrow` argument, overriding any existing attribute of this name in `x` or in the `...` arguments. If `nrow` is `NULL`, the number of rows will be guessed from the data. The `new_tibble()` constructor makes sure that the `row.names` attribute is consistent with the data before returning.

The `class` attribute of the returned object always consists of `c("tbl_df", "tbl", "data.frame")`. If the `subclass` argument is set, it will be prepended to that list of classes.

Examples

```
new_tibble(list(a = 1:3, b = 4:6))

# One particular situation where the nrow argument is essential:
new_tibble(list(), nrow = 150, subclass = "my_tibble")

# It's safest to always pass it along:
new_tibble(list(a = 1:3, b = 4:6), nrow = 3)

## Not run:
# All columns must be the same length:
new_tibble(list(a = 1:3, b = 4.6))

# The length must be consistent with the nrow argument if available:
new_tibble(list(a = 1:3, b = 4:6), nrow = 2)

## End(Not run)
```

rownames *Tools for working with row names*

Description

While a tibble can have row names (e.g., when converting from a regular data frame), they are removed when subsetting with the `[` operator. A warning will be raised when attempting to assign non-`NULL` row names to a tibble. Generally, it is best to avoid row names, because they are basically a character column with different semantics to every other column. These functions allow to you detect if a data frame has row names (`has_rownames()`), remove them (`remove_rownames()`), or convert them back-and-forth between an explicit column (`rownames_to_column()` and `column_to_rownames()`). Also included is `rowid_to_column()` which adds a column at the start of the dataframe of ascending sequential row ids starting at 1. Note that this will remove any existing row names.

Usage

```
has_rownames(df)

remove_rownames(df)

rownames_to_column(df, var = "rowname")

rowid_to_column(df, var = "rowid")

column_to_rownames(df, var = "rowname")
```

Arguments

`df`	A data frame
`var`	Name of column to use for rownames.

Details

In the printed output, the presence of row names is indicated by a star just above the row numbers.

Examples

```
has_rownames(mtcars)
has_rownames(iris)
has_rownames(remove_rownames(mtcars))

head(rownames_to_column(mtcars))

mtcars_tbl <- as_tibble(rownames_to_column(mtcars))
mtcars_tbl
column_to_rownames(as.data.frame(mtcars_tbl))
```

`set_tidy_names` *Repair object names*

Description

`set_tidy_names()` ensures its input has non-missing and unique names (duplicated names get a suffix of the format `..#` where # is the position in the vector). Valid names are left unchanged, with the exception that existing suffixes are reorganized.

`tidy_names()` is the workhorse behind `set_tidy_names()`, it treats the argument as a string to be used to name a data frame or a vector.

`repair_names()` is an older version with different renaming heuristics, kept for backward compatibility. New code should prefer `tidy_names()`.

Usage

```
set_tidy_names(x, syntactic = FALSE, quiet = FALSE)

tidy_names(name, syntactic = FALSE, quiet = FALSE)

repair_names(x, prefix = "V", sep = "")
```

Arguments

`x`	A named vector.
`syntactic`	Should all names be made syntactically valid via `make.names()`?
`quiet`	If `TRUE` suppresses output from this function
`name`	A character vector representing names.
`prefix`	A string, the prefix to use for new column names.
`sep`	A string inserted between the column name and de-duplicating number.

Value

`x` with valid names.

Examples

```
# Works for lists and vectors, too:
set_tidy_names(3:5)
set_tidy_names(list(3, 4, 5))

# Clean data frames are left unchanged:
set_tidy_names(mtcars)

# By default, all rename operations are printed to the console:
tbl <- as_tibble(structure(list(3, 4, 5), class = "data.frame"),
                 validate = FALSE)
```

```
set_tidy_names(tbl)

# Optionally, names can be made syntactic:
tidy_names("a b", syntactic = TRUE)
```

`tbl_sum` *Provide a succinct summary of an object*

Description

`tbl_sum()` gives a brief textual description of a table-like object, which should include the dimensions, the data source, and possible grouping (for `dplyr`). The default implementation forwards to `pillar::obj_sum()`.

Usage

```
tbl_sum(x)
```

Arguments

`x`	Object to summarise

See Also

`pillar::type_sum()`, `pillar::is_vector_s3()`

`tibble` *Build a data frame or list*

Description

`tibble()` is a trimmed down version of `data.frame()` that:

- Never coerces inputs (i.e. strings stay as strings!).
- Never adds `row.names`.
- Never munges column names.
- Only recycles length 1 inputs.
- Evaluates its arguments lazily and in order.
- Adds `tbl_df` class to output.
- Automatically adds column names.

`data_frame()` is an alias to `tibble()`.

`tibble_()` and its alias `data_frame_()` use lazy evaluation and are deprecated. New code should use `tibble()` or `data_frame()` with quasiquotation.

`lst()` is similar to `list()`, but like `tibble()`, it evaluates its arguments lazily and in order, and automatically adds names.

`lst_()` uses lazy evaluation and is deprecated. New code should use `lst()` with quasiquotation.

Usage

```
tibble(...)

data_frame(...)

lst(...)
```

Arguments

`...`	A set of name-value pairs. Arguments are evaluated sequentially, so you can refer to previously created variables. These arguments are processed with `rlang::quos()` and support unquote via `!!` and unquote-splice via `!!!`.

See Also

`as_tibble()` to turn an existing list into a data frame.

Examples

```
a <- 1:5
tibble(a, b = a * 2)
tibble(a, b = a * 2, c = 1)
tibble(x = runif(10), y = x * 2)

lst(n = 5, x = runif(n))

# tibble never coerces its inputs
str(tibble(letters))
str(tibble(x = list(diag(1), diag(2))))

# or munges column names
tibble(`a + b` = 1:5)

# You can splice-unquote a list of quotes and formulas
tibble(!!! list(x = rlang::quo(1:10), y = quote(x * 2)))

# data frames can only contain 1d atomic vectors and lists
# and can not contain POSIXlt
## Not run:
tibble(x = tibble(1, 2, 3))
tibble(y = strptime("2000/01/01", "%x"))

## End(Not run)
lst(n = 5, x = runif(n))

# You can splice-unquote a list of quotes and formulas
lst(!!! list(n = rlang::quo(2 + 3), y = quote(runif(n))))
```

tibble-options *Package options*

Description

Display options for `tbl_df`, used by `trunc_mat()` and (indirectly) by `print.tbl()`.

Package options

- `tibble.print_max`: Row number threshold: Maximum number of rows printed. Set to `Inf` to always print all rows. Default: 20.
- `tibble.print_min`: Number of rows printed if row number threshold is exceeded. Default: 10.
- `tibble.width`: Output width. Default: `NULL` (use `width` option).
- `tibble.max_extra_cols`: Number of extra columns printed in reduced form. Default: 100.
- `pillar.bold`: Use bold font, e.g. for column headers? This currently defaults to `FALSE`, because many terminal fonts have poor support for bold fonts.
- `pillar.subtle`: Use subtle style, e.g. for insignificant digits? Default: `TRUE`.
- `pillar.neg`: Highlight negative numbers? Default: `TRUE`.
- `pillar.sigfig`: The number of significant digits that will be printed and highlighted, default: `3`. Set the `pillar.subtle` option to `FALSE` to turn off highlighting of significant digits.
- `pillar.min_title_chars`: The minimum number of characters for the column title, default: `15`. Column titles may be truncated up to that width to save horizontal space. Set to `Inf` to turn off truncation of column titles.

tribble *Row-wise tibble creation*

Description

Create tibbles using an easier to read row-by-row layout. This is useful for small tables of data where readability is important. Please see tibble-package for a general introduction.

Usage

```
tribble(...)
```

Arguments

`...`	Arguments specifying the structure of a `tibble`. Variable names should be formulas, and may only appear before the data.

Details

`frame_data()` is an older name for `tribble()`. It will eventually be phased out.

Value

A tibble.

Examples

```
tribble(
  ~colA, ~colB,
  "a",   1,
  "b",   2,
  "c",   3
)

# tribble will create a list column if the value in any cell is
# not a scalar
tribble(
  ~x,  ~y,
  "a", 1:3,
  "b", 4:6
)
```

Index