Package 'statar'

July 23, 2025

Title Tools Inspired by 'Stata' to Manipulate Tabular Data

Version 0.7.7

Description A set of tools inspired by 'Stata' to explore data.frames ('summarize', 'tabulate', 'xtile', 'pctile', 'binscatter', elapsed quarters/month, lead/lag).

License GPL-2

URL https://github.com/matthieugomez/statar

BugReports https://github.com/matthieugomez/statar/issues

Depends R (>= 3.2.0)

Imports data.table, tidyselect, dplyr (>= 1.0), ggplot2 (>= 2.0.0), lazyeval, matrixStats, methods, rlang, stringr

Suggests knitr, lubridate, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Matthieu Gomez [aut, cre]

Maintainer Matthieu Gomez <mg3901@columbia.edu>

Repository CRAN

Date/Publication 2025-07-23 14:10:02 UTC

Contents

`elapsed` *Elapsed dates (monthly, quarterly)*

Description

Elapsed dates (monthly, quarterly)

Usage

```
as.quarterly(x)

is.quarterly(x)

as.monthly(x)

is.monthly(x)
```

Arguments

x	a vector

Details

Monthly and quarterly dates are stored as integers, representing the number of elapsed calendar periods since 01/01/1970. As `yearmonth` and `yearqtr` the package `zoo`, these dates are printed in a way that fits their frequency (`YYYqq`, `YYYmMM`). The only difference is that, `monthly`, and `quarterly` are integers, which removes issues due to floating points (particularly important when merging). This also allows to use arithmetic on perios, ie `date + 1` adds one period rather than one day.

Methods to convert from and to Dates or POSIXlt are provided. In particular, you may use lubridate `week` `month` and `year` to extract information from elapsed dates.

Examples

```
library(lubridate)
library(dplyr)
date <- mdy(c("04/03/1992", "01/04/1992", "03/15/1992"))
datem <- as.monthly(date)
is.monthly(datem)
as.quarterly(date)
as.character(datem)
```

```
datem + 1
df <- tibble(datem)
# filter(df, month(datem) == 1)
seq(datem[1], datem[2])
as.Date(datem)
as.POSIXlt(datem)
as.POSIXct(datem)
week(datem)
```

`fill_gap` *Add rows corresponding to gaps in some variable*

Description

Add rows corresponding to gaps in some variable

Usage

```
fill_gap(
  x,
  ...,
  full = FALSE,
  roll = FALSE,
 rollends = if (roll == "nearest") c(TRUE, TRUE) else if (roll >= 0) c(FALSE, TRUE) else
    c(TRUE, FALSE)
)
```

Arguments

`x`	A data frame
`...`	a time variable
`full`	A boolean. When full = FALSE (default), the function creates rows corresponding to all missing times between the min and max of ... within each group. When full = TRUE, the function creates rows corresponding to all missing times between the min and max of ... in the whole dataset.
`roll`	When roll is a positive number, values are carried forward. roll=TRUE is equivalent to roll=+Inf. When roll is a negative number, values are rolled backwards; i.e., next observation carried backwards (NOCB). Use -Inf for unlimited roll back. When roll is "nearest", the nearest value is used. Default to FALSE (no rolling)
`rollends`	A logical vector length 2 (a single logical is recycled). When rolling forward (e.g. roll=TRUE) if a value is past the last observation within each group defined by the join columns, rollends[2]=TRUE will roll the last value forwards. rollends[1]=TRUE will roll the first value backwards if the value is before it. If rollends=FALSE the value of i must fall in a gap in x but not after the end or before the beginning of the data, for that group defined by all but the last join column. When roll is a finite number, that limit is also applied when rolling the end

Examples

```
library(dplyr)
library(lubridate)
df <- tibble(
    id    = c(1, 1, 1, 1),
    datem  = as.monthly(mdy(c("01/01/1992", "02/01/1992", "04/01/1992", "7/11/1992"))),
    value = c(4.1, 4.5, 3.3, 3.2)
)
df %>% group_by(id) %>% fill_gap(datem)
df %>% group_by(id) %>% fill_gap(datem, roll = 1)
df %>% group_by(id) %>% fill_gap(datem, roll = "nearest")
df %>% group_by(id) %>% fill_gap(datem, roll = "nearest", full = TRUE)
```

`is.panel` *Check whether a data.frame is a panel*

Description

Check whether a data.frame is a panel

Usage

```
is.panel(x, ...)
```

Arguments

`x`	a data frame
`...`	a time variable

Value

The function `is.panel` check that there are no duplicate combinations of the variables in ... and that no observation is missing for the last variable in ... (the time variable).

Examples

```
library(dplyr)
df <- tibble(
    id1    = c(1, 1, 1, 2, 2),
    id2   = 1:5,
    year  = c(1991, 1993, NA, 1992, 1992),
    value = c(4.1, 4.5, 3.3, 3.2, 5.2)
)
df %>% group_by(id1) %>% is.panel(year)
df1 <- df %>% filter(!is.na(year))
df1 %>% is.panel(year)
df1 %>% group_by(id1) %>% is.panel(year)
df1 %>% group_by(id1, id2) %>% is.panel(year)
```

`join` *Join two data frames together*

Description

Join two data frames together

Usage

```
join(
  x,
  y,
  kind,
  on = intersect(names(x), names(y)),
  suffixes = c(".x", ".y"),
  check = m ~ m,
  gen = FALSE,
  inplace = FALSE,
  update = FALSE,
  type
)
```

Arguments

`x`	The master data.frame
`y`	The using data.frame
`kind`	The kind of (SQL) join among "full" (default), "left", "right", "inner", "semi", "anti" and "cross".
`on`	Character vectors specifying variables to match on. Default to common names between x and y.
`suffixes`	A character vector of length 2 specifying suffix of overlapping columns. Defaut to ".x" and ".y".
`check`	A formula checking for the presence of duplicates. Specifying 1~m (resp m~1, 1~1) checks that joined variables uniquely identify observations in x (resp y, both).
`gen`	Name of new variable to mark result, or the boolean FALSE (default) if no such variable should be created. The variable equals 1 for rows in master only, 2 for rows in using only, 3 for matched rows.
`inplace`	A boolean. In case "kind"= "left" and RHS of check is 1, the merge can be one in-place.
`update`	A boolean. For common variables in x and y not specified in "on", replace missing observations by the non missing observations in y.
`type`	Deprecated

Value

A data.frame that joins rows in master and using datases. Importantly, if x or y are not keyed, the join may change their row orders.

Examples

```
library(dplyr)
x <- data.frame(a = rep(1:2, each = 3), b=1:6)
y <- data.frame(a = 0:1, bb = 10:11)
join(x, y, kind = "full")
join(x, y, kind = "left", gen = "_merge")
join(x, y, kind = "right", gen = "_merge")
join(x, y, kind = "inner", check = m~1)
join(x, y, kind = "semi")
join(x, y, kind = "anti")
y <- rename(y, b = bb)
join(x, y, kind = "full", on = "a")
join(x, y, kind = "full", on = "a", suffixes = c("",".i"))
y <- data.frame(a = 0:1, bb = 10:11)
join(x, y, kind = "left", check = m~1)
x <- data.frame(a = c(1,2), b=c(NA, 2))
y <- data.frame(a = c(1,2), b = 10:11)
join(x, y, kind = "left", on = "a",  update = TRUE)
join(x, y, kind = "left", on = "a", check = m~1,  update = TRUE)
```

---

n_narm *Count number of non missing observations*

Description

Count number of non missing observations

Usage

```
n_narm(...)
```

Arguments

...	a sequence of vectors, matrices and data frames.

Examples

```
n_narm(1:100, c(NA, 1:99))
```

`pctile` *Weighted quantile of type 2 (similar to Stata _pctile)*

Description

Weighted quantile of type 2 (similar to Stata _pctile)

Usage

```
pctile(x, probs = c(0.25, 0.5, 0.75), wt = NULL, na.rm = FALSE)
```

Arguments

`x`	A vector
`probs`	A vector of probabilities
`wt`	A weight vector
`na.rm`	Should missing values be returned?

`statar` *A package for applied research*

Description

A package for applied research

Author(s)

Maintainer: Matthieu Gomez <mg3901@columbia.edu>

See Also

Useful links:

- https://github.com/matthieugomez/statar
- Report bugs at https://github.com/matthieugomez/statar/issues

stat_binmean *Plot the mean of y over the mean of x within bins of x.*

Description

Plot the mean of y over the mean of x within bins of x.

Usage

```
stat_binmean(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  show.legend = NA,
  inherit.aes = TRUE,
  na.rm = FALSE,
  n = 20,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

`na.rm` If `FALSE` (the default), removes missing values with a warning. If `TRUE` silently removes missing values.

`n` number of x-bins. Default to 20. Set to zero if you want to use distinct value of x for grouping.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

Value

a data.frame with additional columns:

xtile	bins for x
x	mean of x
y	mean of y

Examples

```
library(ggplot2)
g <- ggplot(iris, aes(x = Sepal.Width , y = Sepal.Length)) + stat_binmean(n = 10)
g + stat_smooth(method = "lm", se = FALSE)
ggplot(iris, aes(x = Sepal.Width , y = Sepal.Length, color = Species)) + stat_binmean(n = 10)
ggplot(iris, aes(x = Sepal.Width, y = Sepal.Length, weight = Petal.Length)) + stat_binmean(n = 10)
```

---

sum_up *Gives summary statistics (corresponds to Stata command summarize)*

Description

Gives summary statistics (corresponds to Stata command summarize)

Usage

```
sum_up(df, ..., d = FALSE, wt = NULL)
```

Arguments

df	a data.frame
...	Variables to include. Defaults to all non-grouping variables. See the select documentation.
d	Should detailed summary statistics be printed?
wt	Weights. Default to NULL.

Value

a data.frame

Examples

```
library(dplyr)
N <- 100
df <- tibble(
  id = 1:N,
  v1 = sample(5, N, TRUE),
  v2 = sample(1e6, N, TRUE)
)
sum_up(df)
sum_up(df, v2, d = TRUE)
sum_up(df, v2, wt = v1)
df %>% group_by(v1) %>% sum_up(starts_with("v"))
```

tab *Returns cross tabulation*

Description

Returns cross tabulation

Usage

```
tab(x, ..., wt = NULL, na.rm = FALSE, sort = TRUE)
```

Arguments

x	a vector or a data.frame
...	Variable(s) to include. If length is two, a special cross tabulation table is printed although a long data.frame is always (invisibly) returned.
wt	Frequency weights. Default to NULL.
na.rm	Remove missing values. Default to FALSE
sort	Boolean. Default to TRUE

Value

a data.frame sorted by variables in ..., and with columns "Freq.", "Percent", and "Cum." for counts.

Examples

```
# setup
library(dplyr)
N <- 1e2 ; K = 10
df <- tibble(
  id = sample(c(NA,1:5), N/K, TRUE),
  v1 =  sample(1:5, N/K, TRUE)
)
# one-way tabulation
df %>% tab(id)
df %>% tab(id, wt = v1)
# two-way tabulation
df %>% tab(id, v1)
df %>% filter(id >= 3) %>% tab(id)
```

tempname *Create unique names within a list, a data.frame, or an environment*

Description

Create unique names within a list, a data.frame, or an environment

Usage

```
tempname(where = globalenv(), n = 1, prefix = ".temp", inherits = TRUE)
```

Arguments

where	A chracter vector, list or an environment
n	An integar that specifies length of the output
prefix	A character vector that specifies prefix for new name
inherits	Should the name unique also in the enclosing frames of the environment?

Examples

```
tempname(c("temp1", "temp3"), 4)
tempname(globalenv())
tempname(data.frame(temp = 1), n = 3)
```

tlead-tlag *lead and lag with respect to a time variable*

Description

lead and lag with respect to a time variable

Usage

```
tlead(x, n = 1L, time, default = NA)

tlag(x, n = 1L, time, default = NA)
```

Arguments

x	a vector of values
n	a positive integer of length 1, giving the number of positions to lead or lag by. When the package lubridate is loaded, it can be a period when using with time (see the lubridate function minutes, hours, days, weeks, months and years)
time	time variable
default	value used for non-existant rows. Defaults to NA.

Examples

```
date <- c(1989, 1991, 1992)
value <- c(4.1, 4.5, 3.3)
tlag(value, 1, time = date) #  returns value in year - 1
library(lubridate)
date <- as.monthly(mdy(c("01/04/1992", "03/15/1992", "04/03/1992")))
tlag(value, time = date)
library(dplyr)
df <- tibble(
   id    = c(1, 2, 2),
   date  = date,
   value = value
)
df %>% group_by(id) %>% mutate(valuel = tlag(value, n = 1, time = date))
```

---

winsorize *Winsorize a numeric vector*

Description

Winsorize a numeric vector

Usage

```
winsorize(
  x,
  probs = NULL,
  cutpoints = NULL,
  replace = c(cutpoints[1], cutpoints[2]),
  verbose = TRUE
)

winsorise(
  x,
  probs = NULL,
  cutpoints = NULL,
  replace = c(cutpoints[1], cutpoints[2]),
  verbose = TRUE
)
```

Arguments

x	A vector of values
probs	A vector of probabilities that can be used instead of cutpoints. Quantiles are computed as the inverse of the empirical distribution function (type = 1)

cutpoints	Cutpoints under and above which are defined outliers. Default is (median - five times interquartile range, median + five times interquartile range). Compared to bottom and top percentile, this takes into account the whole distribution of the vector.
replace	Values by which outliers are replaced. Default to cutpoints. A frequent alternative is NA.
verbose	Boolean. Should the percentage of replaced values printed?

Examples

```
v <- c(1:4, 99)
winsorize(v)
winsorize(v, replace = NA)
winsorize(v, probs = c(0.01, 0.99))
winsorize(v, cutpoints = c(1, 50))
```

---

xtile *Bin variable in groups (similar to Stata xtile)*

Description

Bin variable in groups (similar to Stata xtile)

Usage

```
xtile(x, n = NULL, probs = NULL, cutpoints = NULL, wt = NULL)
```

Arguments

x	A vector
n	A numeric specifying number of quantiles. Can be used instead of cutpoints
probs	A vector of probabilities that an be used instead of cutpoints. Quantiles are computed as the inverse of the empirical distribution function (type = 1)
cutpoints	Cutpoints to use when nq is not specified. For instance `cutpoints = 0.4` creates two groups, one for observations equal or below 0.4, one for observations superior to 0.4.
wt	A variable specifying weight in case the option n_quantiles is specified.

Value

An integer vector representing groups corresponding to cutpoints. Includes missing values when present in the original vector.

Examples

```
x <- c(NA, 1:10)
xtile(x, n = 3) # 3 groups based on terciles
xtile(x, probs = c(0.3, 0.7)) # 3 groups based on two quantiles
xtile(x, cutpoints = c(2, 3)) # 3 groups based on two cutpoints
```

Index