Package 'brolgar'

October 12, 2022

Title Browse Over Longitudinal Data Graphically and Analytically in R

Version 0.1.2

Description Provides a framework of tools to summarise, visualise, and explore longitudinal data. It builds upon the tidy time series data frames used in the 'tsibble' package, and is designed to integrate within the 'tidyverse', and 'tidyverts' (for time series) ecosystems. The methods implemented include calculating features for understanding longitudinal data, including calculating summary statistics such as quantiles, medians, and numeric ranges, sampling individual series, identifying individual series representative of a group, and extending the facet system in 'ggplot2' to facilitate exploration of samples of data. These methods are fully described in the paper ``brolgar: An R package to Browse Over Longitudinal Data Graphically and Analytically in R'', Nicholas Tierney, Dianne Cook, Tania Prvan (2020) <arXiv:2012.01619>.

License MIT + file LICENSE

URL https://github.com/njtierney/brolgar

BugReports https://github.com/njtierney/brolgar/issues

Depends R (>= 3.5.0)

Imports dplyr (>= 0.8.3), fabletools, ggplot2 (>= 3.2.0), glue (>= 1.3.1), magrittr (>= 1.5), purrr (>= 0.3.2), rlang (>= 0.4.0), stats, tibble (>= 2.1.3), tidyr (>= 0.8.3), tsibble (>= 0.8.2), vctrs

Suggests markdown, covr, gapminder, gghighlight (>= 0.1.0), knitr (>= 1.23), lme4, modelr, readr (>= 1.3.1), rmarkdown (>= 1.14), spelling (>= 2.1), testthat (>= 2.1.0), tsibbledata, vdiffr (>= 0.3.1)

VignetteBuilder knitr

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.1.1

Config/testthat/edition 3

NeedsCompilation no

Author Nicholas Tierney [aut, cre] (<https://orcid.org/0000-0003-1460-8722>),
Di Cook [aut] (<https://orcid.org/0000-0002-3813-7155>),
Tania Prvan [aut],
Stuart Lee [ctb],
Earo Wang [ctb]

Maintainer Nicholas Tierney <nicholas.tierney@gmail.com>

Repository CRAN

Date/Publication 2021-08-25 12:50:18 UTC

R topics documented:

add_n_obs . . . 3
brolgar . . . 3
brolgar-features . . . 4
b_min . . . 5
facet_sample . . . 6
facet_strata . . . 8
heights . . . 9
index_summary . . . 11
keys_near . . . 12
keys_near.data.frame . . . 12
keys_near.tbl_ts . . . 13
key_slope . . . 14
l_funs . . . 15
monotonic . . . 15
nearests . . . 17
near_between . . . 18
near_middle . . . 20
near_quantile . . . 20
n_obs . . . 21
pisa . . . 22
sample-n-frac-keys . . . 24
stratify_keys . . . 25
wages . . . 26

Index **28**

add_n_obs *Add the number of observations for each key in a* tsibble

Description

Here, we are not counting the number of rows in the dataset, but rather we are counting the number observations for each keys in the data.

Usage

```
add_n_obs(.data, ...)
```

Arguments

.data	tsibble
...	extra arguments

Value

tsibble with n_obs, the number of observations per key added.

Examples

```
library(dplyr)
# you can explore the data to see those cases that have exactly two
 # observations:
heights %>%
  add_n_obs() %>%
  filter(n_obs == 2)
```

brolgar *brolgar*

Description

brolgar stands for: **BR**owse over **L**ongitudinal data **G**raphically and **A**nalytically in **R**.

brolgar-features *Calculate features of a* tsibble *object in conjunction with* features()

Description

You can calculate a series of summary statistics (features) of a given variable for a dataset. For example, a three number summary, the minimum, median, and maximum, can be calculated for a given variable. This is designed to work with the features() function shown in the examples. Other available features in brolgar include:

- feat_three_num() - minimum, median, maximum
- feat_five_num() - minimum, q25, median, q75, maximum.
- feat_ranges() - min, max, range difference, interquartile range.
- feat_spread() - variance, standard deviation, median absolute distance, and interquartile range
- feat_monotonic() - is it always increasing, decreasing, or unvarying?
- feat_diff_summary() - the summary statistics of the differences amongst a value, including the five number summary, as well as the standard deviation and variance. Returns NA if there is only one observation, as we can't take the difference of one observation, and a difference of 0 in these cases would be misleading.
- feat_brolgar() all features in brolgar.

Usage

```
feat_three_num(x, ...)

feat_five_num(x, ...)

feat_ranges(x, ...)

feat_spread(x, ...)

feat_monotonic(x, ...)

feat_brolgar(x, ...)

feat_diff_summary(x, ...)
```

Arguments

x	A vector to extract features from.
...	Further arguments passed to other functions.

Examples

```
# You can use any of the features `feat_*` in conjunction with `features`
# like so:
heights %>%
  features(height_cm, # variable you want to explore
           feat_three_num) # the feature summarisation you want to perform
```

---

b_min *Brolgar summaries (b_summaries)*

Description

Customised summaries of vectors with appropriate defaults for longitudinal data. The functions are prefixed with b_ to assist with autocomplete. It uses `na.rm = TRUE` for all, and for calculations involving quantiles, `type = 8` and `names = FALSE`. Summaries include: * b_min: The minimum * b_max: The maximum * b_median: The median * b_mean: The mean * b_q25: The 25th quantile * b_q75: The 75th quantile * b_range: The range * b_range_diff: difference in range (max - min) * b_sd: The standard deviation * b_var: The variance * b_mad: The mean absolute deviation * b_iqr: The Inter-quartile range * b_diff_var: The variance diff() * b_diff_sd: The standard deviation of diff() * b_diff_mean: The mean of diff() * b_diff_median: The median of diff() * b_diff_q25: The q25 of diff() * b_diff_q75: The q75 of diff()

Usage

```
b_min(x, ...)

b_max(x, ...)

b_median(x, ...)

b_mean(x, ...)

b_q25(x, ...)

b_q75(x, ...)

b_range(x, ...)

b_range_diff(x, ...)

b_sd(x, ...)

b_var(x, ...)

b_mad(x, ...)
```

```
b_iqr(x, ...)

b_diff_var(x, ...)

b_diff_sd(x, ...)

b_diff_mean(x, ...)

b_diff_median(x, ...)

b_diff_q25(x, ...)

b_diff_q75(x, ...)

b_diff_max(x, ...)

b_diff_min(x, ...)

b_diff_iqr(x, ...)
```

Arguments

x	a vector
...	other arguments to pass

Examples

```
x <- c(1:5, NA, 5:1)
min(x)
b_min(x)
max(x)
b_max(x)
median(x)
b_median(x)
mean(x)
b_mean(x)
range(x)
b_range(x)
var(x)
b_var(x)
sd(x)
b_sd(x)
```

---

facet_sample *Facet data into groups to facilitate exploration*

Description

This function requires a `tbl_ts` object, which can be created with `tsibble::as_tsibble()`. Under the hood, `facet_strata` is powered by `stratify_keys()` and `sample_n_keys()`.

Usage

```
facet_sample(
  n_per_facet = 3,
  n_facets = 12,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  shrink = TRUE,
  strip.position = "top"
)
```

Arguments

`n_per_facet`	Number of keys per facet you want to plot. Default is 3.
`n_facets`	Number of facets to create. Default is 12
`nrow`	Number of rows and columns.
`ncol`	Number of rows and columns.
`scales`	Should scales be fixed (`"fixed"`, the default), free (`"free"`), or free in one dimension (`"free_x"`, `"free_y"`)?
`shrink`	If `TRUE`, will shrink scales to fit output of statistics, not raw data. If `FALSE`, will be range of raw data before statistical summary.
`strip.position`	By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top", "bottom", "left", "right")`

Value

a ggplot object

Examples

```
library(ggplot2)
ggplot(heights,
aes(x = year,
    y = height_cm,
    group = country)) +
  geom_line() +
  facet_sample()

ggplot(heights,
       aes(x = year,
           y = height_cm,
           group = country)) +
```

```
  geom_line() +
  facet_sample(n_per_facet = 1,
               n_facets = 12)
```

`facet_strata` *Facet data into groups to facilitate exploration*

Description

This function requires a `tbl_ts` object, which can be created with `tsibble::as_tsibble()`. Under the hood, `facet_strata` is powered by `stratify_keys()`.

Usage

```
facet_strata(
  n_strata = 12,
  along = NULL,
  fun = mean,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  shrink = TRUE,
  strip.position = "top"
)
```

Arguments

`n_strata`	number of groups to create
`along`	variable to stratify along. This groups by each `key` and then takes a summary statistic (by default, the mean). It then arranges by the mean value for each `key` and assigns the `n_strata` groups.
`fun`	summary function. Default is mean.
`nrow`	Number of rows and columns.
`ncol`	Number of rows and columns.
`scales`	Should scales be fixed (`"fixed"`, the default), free (`"free"`), or free in one dimension (`"free_x"`, `"free_y"`)?
`shrink`	If `TRUE`, will shrink scales to fit output of statistics, not raw data. If `FALSE`, will be range of raw data before statistical summary.
`strip.position`	By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top", "bottom", "left", "right")`

Value

a ggplot object

Examples

```
library(ggplot2)
ggplot(heights,
       aes(x = year,
           y = height_cm,
           group = country)) +
  geom_line() +
  facet_strata()


ggplot(heights,
       aes(x = year,
           y = height_cm,
           group = country)) +
  geom_line() +
  facet_wrap(~continent)

ggplot(heights,
       aes(x = year,
           y = height_cm,
           group = country)) +
  geom_line() +
  facet_strata(along = year)


library(dplyr)
heights %>%
  key_slope(height_cm ~ year) %>%
  right_join(heights, ., by = "country") %>%
  ggplot(aes(x = year,
             y = height_cm)) +
  geom_line(aes(group = country)) +
  geom_smooth(method = "lm") +
  facet_strata(along = .slope_year)
```

---

heights *World Height Data*

Description

Average male heights in 144 countries from 1810-1989, with a smaller number of countries from 1500-1800. Data has been filtered to only include countries with more than one observation.

Usage

```
heights
```

Format

An object of class `tbl_ts` (inherits from `tbl_df`, `tbl`, `data.frame`) with 1490 rows and 4 columns.

Details

`heights` is stored as a time series `tsibble` object. It contains the variables:

- country: The Country. This forms the identifying `key`.
- year: Year. This forms the time `index`.
- height_cm: Average male height in centimeters.
- continent: continent extracted from country name using `countrycode` package (https://joss.theoj.org/papers/10.21105/j

For more information, see the article: "Why are you tall while others are short? Agricultural production and other proximate determinants of global heights", Joerg Baten and Matthias Blum, European Review of Economic History 18 (2014), 144–165. Data available from https://datasets.iisg.amsterdam/dataset.xhtml?persistentId=hdl:10622/IAEKLA, accessed via the Clio Infra website.

Examples

```
# show the data
heights

# show the spaghetti plot (ugh!)
library(ggplot2)
ggplot(heights,
       aes(x = year,
           y = height_cm,
           group = country)) +
    geom_line()

# Explore all samples with `facet_strata()`
ggplot(heights,
       aes(x = year,
           y = height_cm,
           group = country)) +
  geom_line() +
  facet_strata()

# Explore the heights over each continent
ggplot(heights,
       aes(x = year,
           y = height_cm,
           group = country)) +
  geom_line() +
  facet_wrap(~continent)

# explore the five number summary of height_cm with `features`
heights %>%
  features(height_cm, feat_five_num)
```

index_summary *Index summaries*

Description

These functions check if the index is regular (`index_regular()`), and summarise the index variable (`index_summary()`). This can be useful to check your index variables.

Usage

```
index_regular(.data, ...)

## S3 method for class 'tbl_ts'
index_regular(.data, ...)

## S3 method for class 'data.frame'
index_regular(.data, index, ...)

index_summary(.data, ...)

## S3 method for class 'tbl_ts'
index_summary(.data, ...)

## S3 method for class 'data.frame'
index_summary(.data, index, ...)
```

Arguments

`.data`	data.frame or tsibble
`...`	extra arguments
`index`	the proposed index variable

Value

logical TRUE means it is regular, FALSE means not

Examples

```
# a tsibble
index_regular(heights)

# some data frames
index_regular(pisa, year)
index_regular(airquality, Month)

# a tsibble
index_summary(heights)
```

```
# some data frames
index_summary(pisa, year)
index_summary(airquality, Month)
index_summary(airquality, Day)
```

---

keys_near *Return keys nearest to a given statistics or summary.*

Description

Return keys nearest to a given statistics or summary.

Usage

```
keys_near(.data, ...)

## Default S3 method:
keys_near(.data, ...)
```

Arguments

.data	tsibble
...	extra arguments to pass to mutate_at when performing the summary as given by funs.

Value

data.frame containing keys closest to a given statistic.

Examples

```
keys_near(heights, height_cm)
```

---

keys_near.data.frame *Return keys nearest to a given statistics or summary.*

Description

Return keys nearest to a given statistics or summary.

Usage

```
## S3 method for class 'data.frame'
keys_near(.data, key, var, top_n = 1, funs = l_five_num, ...)
```

Arguments

.data	data.frame
key	key, which identifies unique observations.
var	variable to summarise
top_n	top number of closest observations to return - default is 1, which will also return ties.
funs	named list of functions to summarise by. Default is a given list of the five number summary, l_five_num.
...	extra arguments to pass to mutate_at when performing the summary as given by funs.

Examples

```
heights %>%
  key_slope(height_cm ~ year) %>%
  keys_near(key = country,
            var = .slope_year)
# Specify your own list of summaries
l_ranges <- list(min = b_min,
                 range_diff = b_range_diff,
                 max = b_max,
                 iqr = b_iqr)

heights %>%
  key_slope(formula = height_cm ~ year) %>%
  keys_near(key = country,
              var = .slope_year,
              funs = l_ranges)
```

---

keys_near.tbl_ts *Return keys nearest to a given statistics or summary.*

Description

Return keys nearest to a given statistics or summary.

Usage

```
## S3 method for class 'tbl_ts'
keys_near(.data, var, top_n = 1, funs = l_five_num, stat_as_factor = TRUE, ...)
```

Arguments

.data	tsibble
var	variable to summarise

top_n	top number of closest observations to return - default is 1, which will also return ties.
funs	named list of functions to summarise by. Default is a given list of the five number summary, l_five_num.
stat_as_factor	coerce stat variable into a factor? Default is TRUE.
...	extra arguments to pass to mutate_at when performing the summary as given by funs.

Examples

```
# Return observations closest to the five number summary of height_cm
heights %>%
  keys_near(var = height_cm)
```

---

key_slope *Fit linear model for each key*

Description

Using key_slope you can fit a linear model to each key in the tsibble. add_key_slope adds this slope information back to the data, and returns the full dimension tsibble.

Usage

```
key_slope(.data, formula, ...)

add_key_slope(.data, formula)

add_key_slope.default(.data, formula)
```

Arguments

.data	tsibble
formula	formula
...	extra arguments

Value

tibble with coefficient information

Examples

```
key_slope(heights, height_cm ~ year)
```

l_funs *A named list of the five number summary*

Description

Designed for use with the `keys_near()` function.

Usage

```
l_five_num

l_three_num
```

Format

An object of class `list` of length 5.

An object of class `list` of length 3.

Examples

```
# Specify your own list of summaries
l_ranges <- list(min = b_min,
                 range_diff = b_range_diff,
                 max = b_max,
                 iqr = b_iqr)

heights %>%
  key_slope(formula = height_cm ~ year) %>%
  keys_near(key = country,
              var = .slope_year,
              funs = l_ranges)
```

monotonic *Are values monotonic? Always increasing, decreasing, or unvarying?*

Description

These provides three families of functions to tell you if values are always increasing, decreasing, or unvarying, with the functions, `increasing()`, `decreasing()`, or `unvarying()`. Under the hood it uses `diff` to find differences, so if you like you can pass extra arguments to `diff`.

Usage

```
increasing(x, ...)

decreasing(x, ...)

unvarying(x, ...)

monotonic(x, ...)
```

Arguments

x	numeric or integer
...	extra arguments to pass to diff

Value

logical TRUE or FALSE

Examples

```
vec_inc <- c(1:10)
vec_dec<- c(10:1)
vec_ran <- c(sample(1:10))
vec_flat <- rep.int(1,10)

increasing(vec_inc)
increasing(vec_dec)
increasing(vec_ran)
increasing(vec_flat)

decreasing(vec_inc)
decreasing(vec_dec)
decreasing(vec_ran)
decreasing(vec_flat)

unvarying(vec_inc)
unvarying(vec_dec)
unvarying(vec_ran)
unvarying(vec_flat)

library(ggplot2)
library(gghighlight)
library(dplyr)

heights_mono <- heights %>%
  features(height_cm, feat_monotonic) %>%
  left_join(heights, by = "country")

  ggplot(heights_mono,
         aes(x = year,
             y = height_cm,
```

```
            group = country)) +
  geom_line() +
  gghighlight(increase)

 ggplot(heights_mono,
        aes(x = year,
            y = height_cm,
            group = country)) +
  geom_line() +
  gghighlight(decrease)

heights_mono %>%
filter(monotonic) %>%
  ggplot(aes(x = year,
             y = height_cm,
             group = country)) +
  geom_line()

heights_mono %>%
  filter(increase) %>%
  ggplot(aes(x = year,
             y = height_cm,
             group = country)) +
  geom_line()
```

`nearests` *Is x nearest to y?*

Description

Returns TRUE if x is nearest to y. There are two implementations. `nearest_lgl()` returns a logical vector when an element of the first argument is nearest to an element of the second argument. `nearest_qt_lgl()` is similar to `nearest_lgl()`, but instead determines if an element of the first argument is nearest to some value of the given quantile probabilities. See example for more detail.

Usage

```
nearest_lgl(x, y)

nearest_qt_lgl(y, ...)
```

Arguments

`x`	a numeric vector
`y`	a numeric vector
`...`	(if used) arguments to pass to `quantile()`.

Value

logical vector of `length(y)`

Examples

```
x <- 1:10
y <- 5:14
z <- 16:25
a <- -1:-5
b <- -1

nearest_lgl(x, y)
nearest_lgl(y, x)

nearest_lgl(x, z)
nearest_lgl(z, x)

nearest_lgl(x, a)
nearest_lgl(a, x)

nearest_lgl(x, b)
nearest_lgl(b, x)

library(dplyr)
heights_near_min <- heights %>%
  filter(nearest_lgl(min(height_cm), height_cm))

heights_near_fivenum <- heights %>%
  filter(nearest_lgl(fivenum(height_cm), height_cm))

heights_near_qt_1 <- heights %>%
  filter(nearest_qt_lgl(height_cm, c(0.5)))

heights_near_qt_3 <- heights %>%
  filter(nearest_qt_lgl(height_cm, c(0.1, 0.5, 0.9)))
```

---

near_between *Return x percent to y percent of values*

Description

Return x percent to y percent of values

Usage

```
near_between(x, from, to)
```

Arguments

x	numeric vector
from	the lower bound of percentage
to	the upper bound of percentage

Value

logical vector

Examples

```
x <- runif(20)

near_middle(x = x,
            middle = 0.5,
            within = 0.2)

library(dplyr)
heights %>% features(height_cm, list(min = min)) %>%
  filter(near_between(min, 0.1, 0.9))

near_quantile(x = x,
              probs = 0.5,
              tol = 0.01)

near_quantile(x, c(0.25, 0.5, 0.75), 0.05)

heights %>%
  features(height_cm, l_five_num) %>%
  mutate_at(vars(min:max),
            .funs = near_quantile,
            0.5,
            0.01) %>%
  filter(min)

heights %>%
  features(height_cm, list(min = min)) %>%
  mutate(min_near_q3 = near_quantile(min, c(0.25, 0.5, 0.75), 0.01)) %>%
  filter(min_near_q3)

heights %>%
  features(height_cm, list(min = min)) %>%
  filter(near_between(min, 0.1, 0.9))

heights %>%
  features(height_cm, list(min = min)) %>%
  filter(near_middle(min, 0.5, 0.1))
```

near_middle *Return the middle x percent of values*

Description

Return the middle x percent of values

Usage

```
near_middle(x, middle, within)
```

Arguments

x	numeric vector
middle	percentage you want to center around
within	percentage around center

Value

logical vector

Examples

```
x <- runif(20)
near_middle(x = x,
            middle = 0.5,
            within = 0.2)

library(dplyr)
heights %>% features(height_cm, list(min = min)) %>%
  filter(near_middle(min, 0.5, 0.1))
```

near_quantile *Which values are nearest to any given quantiles*

Description

Which values are nearest to any given quantiles

Usage

```
near_quantile(x, probs, tol = 0.01)
```

Arguments

x	vector
probs	quantiles to calculate
tol	tolerance in terms of x that you will accept near to the quantile. Default is 0.01.

Value

logical vector of TRUE/FALSE if number is close to a quantile

Examples

```
x <- runif(20)
near_quantile(x, 0.5, 0.05)
near_quantile(x, c(0.25, 0.5, 0.75), 0.05)

library(dplyr)
heights %>%
  features(height_cm, list(min = min)) %>%
  mutate(min_near_median = near_quantile(min, 0.5, 0.01)) %>%
  filter(min_near_median)
heights %>%
  features(height_cm, list(min = min)) %>%
  mutate(min_near_q3 = near_quantile(min, c(0.25, 0.5, 0.75), 0.01)) %>%
  filter(min_near_q3)
```

---

n_obs *Return the number of observations*

Description

Returns the number of observations of a vector or data.frame. It uses `vctrs::vec_size()` under the hood.

Usage

```
n_obs(x, names = TRUE)
```

Arguments

x	vector or data.frame
names	logical; If TRUE the result is a named vector named "n_obs", else it is just the number of observations.

Value

number of observations

Note

You cannot use `n_obs` with `features` counting the key variable like so - `features(heights, country, n_obs)`. Instead, use any other variable.

Examples

```
n_obs(iris)
n_obs(1:10)
add_n_obs(heights)
heights %>%
  features(height_cm, n_obs) # can be any variable except id, the key.
```

---

pisa *Student data from 2000-2018 PISA OECD data*

Description

A subset of PISA data, containing scores and other information from the triennial testing of 15 year olds around the globe. Original data available from https://www.oecd.org/pisa/data/. Data derived from https://github.com/kevinwang09/learningtower.

Usage

```
pisa
```

Format

A tibble of the following variables

- year the year of measurement
- country the three letter country code. This data contains Australia, New Zealand, and Indonesia. The full data from learningtower contains 99 countries.
- school_id The unique school identification number
- student_id The student identification number
- gender recorded gender - 1 female or 2 male or missing
- math Simulated score in mathematics
- read Simulated score in reading
- science Simulated score in science
- stu_wgt The final survey weight score for the student score

Understanding a bit more about the PISA data, the `school_id` and `student_id` are not unique across time. This means the longitudinal element is the country within a given year.

We can cast `pisa` as a `tsibble`, but we need to aggregate the data to each year and country. In doing so, it is important that we provide some summary statistics of each of the scores - we want to

include the mean, and minimum and maximum of the math, reading, and science scores, so that we do not lose the information of the individuals.

The example code below does this, first grouping by year and country, then calculating the weighted mean for math, reading, and science. This can be done using the student weight variable `stu_wgt`, to get the survey weighted mean. The minimum and maximum are then calculated.

Examples

```
pisa

library(dplyr)
# Let's identify

#1.  The **key**, the individual, who would have repeated measurements.
#2.  The **index**, the time component.
#3.  The **regularity** of the time interval (index).

# Here it looks like the key is the student_id, which is nested within
# school_id #' and country,

# And the index is year, so we would write the following

as_tsibble(pisa,
           key = country,
           index = year)

# We can assess the regularity of the year like so:

index_regular(pisa, year)
index_summary(pisa, year)

# We can now convert this into a `tsibble`:

pisa_ts <- as_tsibble(pisa,
           key = country,
           index = year,
           regular = TRUE)

pisa_ts
pisa_ts_au_nz <- pisa_ts %>% filter(country %in% c("AUS", "NZL", "QAT"))

library(ggplot2)
ggplot(pisa_ts_au_nz,
       aes(x = year,
           y = math_mean,
           group = country,
           colour = country)) +
  geom_ribbon(aes(ymin = math_min,
                  ymax = math_max),
              fill = "grey70") +
  geom_line(size = 1) +
  lims(y = c(0, 1000)) +
```

```
  labs(y = "math") +
facet_wrap(~country)
```

`sample-n-frac-keys` *Sample a number or fraction of keys to explore*

Description

Sample a number or fraction of keys to explore

Usage

```
sample_n_keys(.data, size)

sample_frac_keys(.data, size)
```

Arguments

`.data`	tsibble object
`size`	The number or fraction of observations, depending on the function used. In `sample_n_keys`, it is a number > 0, and in `sample_frac_keys` it is a fraction, between 0 and 1.

Value

tsibble with fewer observations of key

Examples

```
library(ggplot2)
sample_n_keys(heights,
              size = 10) %>%
  ggplot(aes(x = year,
             y = height_cm,
             group = country)) +
  geom_line()
library(ggplot2)
sample_frac_keys(wages,
                 0.1) %>%
  ggplot(aes(x = xp,
             y = unemploy_rate,
             group = id)) +
  geom_line()
```

stratify_keys *Stratify the keys into groups to facilitate exploration*

Description

To look at as much of the raw data as possible, it can be helpful to stratify the data into groups for plotting. You can `stratify` the `keys` using the `stratify_keys()` function, which adds the column, `.strata`. This allows the user to create facetted plots showing a more of the raw data.

Usage

```
stratify_keys(.data, n_strata, along = NULL, fun = mean, ...)
```

Arguments

`.data`	data.frame to explore
`n_strata`	number of groups to create
`along`	variable to stratify along. This groups by each `key` and then takes a summary statistic (by default, the mean). It then arranges by the mean value for each `key` and assigns the `n_strata` groups.
`fun`	summary function. Default is mean.
`...`	extra arguments

Value

data.frame with column, `.strata` containing `n_strata` groups

Examples

```
library(ggplot2)
library(brolgar)

heights %>%
  sample_frac_keys(size = 0.1) %>%
  stratify_keys(10) %>%
 ggplot(aes(x = height_cm,
            y = year,
            group = country)) +
 geom_line() +
 facet_wrap(~.strata)

 # now facet along some feature
library(dplyr)
 heights %>%
key_slope(height_cm ~ year) %>%
  right_join(heights, ., by = "country") %>%
  stratify_keys(n_strata = 12,
                along = .slope_year,
```

```
                fun = median) %>%
  ggplot(aes(x = year,
             y = height_cm,
             group = country)) +
  geom_line() +
  facet_wrap(~.strata)


heights %>%
  stratify_keys(n_strata = 12,
                along = height_cm) %>%
  ggplot(aes(x = year,
             y = height_cm,
             group = country)) +
  geom_line() +
  facet_wrap(~.strata)
```

`wages` *Wages data from National Longitudinal Survey of Youth (NLSY)*

Description

This data contains measurements on hourly wages by years in the workforce, with education and race as covariates. The population measured was male high-school dropouts, aged between 14 and 17 years when first measured. `wages` is a time series `tsibble`. It comes from J. D. Singer and J. B. Willett. Applied Longitudinal Data Analysis. Oxford University Press, Oxford, UK, 2003. https://stats.idre.ucla.edu/stat/r/examples/alda/data/wages_pp.txt

Usage

```
wages
```

Format

A `tsibble` data frame with 6402 rows and 8 variables:

id 1–888, for each subject. This forms the `key` of the data

ln_wages natural log of wages, adjusted for inflation, to 1990 dollars.

xp Experience - the length of time in the workforce (in years). This is treated as the time variable, with t0 for each subject starting on their first day at work. The number of time points and values of time points for each subject can differ. This forms the `index` of the data

ged when/if a graduate equivalency diploma is obtained.

xp_since_ged change in experience since getting a ged (if they get one)

black categorical indicator of race = black.

hispanic categorical indicator of race = hispanic.

high_grade highest grade completed

unemploy_rate unemployment rates in the local geographic region at each measurement time

Examples

```
# show the data
wages
library(ggplot2)
# set seed so that the plots stay the same
set.seed(2019-7-15-1300)
# explore a sample of five individuals
wages %>%
  sample_n_keys(size = 5) %>%
  ggplot(aes(x = xp,
             y = ln_wages,
             group = id)) +
  geom_line()

# Explore many samples with `facet_sample()`
  ggplot(wages,
         aes(x = xp,
             y = ln_wages,
             group = id)) +
  geom_line() +
  facet_sample()

# explore the five number summary of ln_wages with `features`
wages %>%
  features(ln_wages, feat_five_num)
```

Index

∗ **datasets**
  heights, 9
  l_funs, 15
  pisa, 22
  wages, 26

add_key_slope *(*key_slope*)*, 14
add_n_obs, 3

b_diff_iqr *(*b_min*)*, 5
b_diff_max *(*b_min*)*, 5
b_diff_mean *(*b_min*)*, 5
b_diff_median *(*b_min*)*, 5
b_diff_min *(*b_min*)*, 5
b_diff_q25 *(*b_min*)*, 5
b_diff_q75 *(*b_min*)*, 5
b_diff_sd *(*b_min*)*, 5
b_diff_var *(*b_min*)*, 5
b_iqr *(*b_min*)*, 5
b_mad *(*b_min*)*, 5
b_max *(*b_min*)*, 5
b_mean *(*b_min*)*, 5
b_median *(*b_min*)*, 5
b_min, 5
b_q25 *(*b_min*)*, 5
b_q75 *(*b_min*)*, 5
b_range *(*b_min*)*, 5
b_range_diff *(*b_min*)*, 5
b_sd *(*b_min*)*, 5
b_summaries *(*b_min*)*, 5
b_var *(*b_min*)*, 5
brolgar, 3
brolgar-features, 4

decreasing *(*monotonic*)*, 15

facet_sample, 6
facet_strata, 8
feat_brolgar *(*brolgar-features*)*, 4
feat_brolgar(), *4*
feat_diff_summary *(*brolgar-features*)*, 4
feat_diff_summary(), *4*
feat_five_num *(*brolgar-features*)*, 4
feat_five_num(), *4*
feat_monotonic *(*brolgar-features*)*, 4
feat_monotonic(), *4*
feat_ranges *(*brolgar-features*)*, 4
feat_ranges(), *4*
feat_spread *(*brolgar-features*)*, 4
feat_spread(), *4*
feat_three_num *(*brolgar-features*)*, 4
feat_three_num(), *4*
features(), *4*

heights, 9

increasing *(*monotonic*)*, 15
index_regular *(*index_summary*)*, 11
index_summary, 11

key_slope, 14
keys_near, 12
keys_near(), *15*
keys_near.data.frame, 12
keys_near.tbl_ts, 13

l_five_num *(*l_funs*)*, 15
l_funs, 15
l_three_num *(*l_funs*)*, 15

monotonic, 15

n_obs, 21
near_between, 18
near_middle, 20
near_quantile, 20
nearest_lgl *(*nearests*)*, 17
nearest_qt_lgl *(*nearests*)*, 17
nearests, 17

pisa, 22

sample-n-frac-keys, 24
sample_frac_keys *(*sample-n-frac-keys*)*, 24
sample_n_keys *(*sample-n-frac-keys*)*, 24
sample_n_keys(), *7*
stratify_keys, 25
stratify_keys(), *7*, *8*

unvarying *(*monotonic*)*, 15

wages, 26