Package 'epikit'

March 5, 2020

Title Miscellaneous Tools for the 'R4Epis' Project

Version 0.1.0

Description Contains tools for formatting inline code, renaming redundant columns, aggregating age categories, and calculating proportions with confidence intervals. This is part of the 'R4Epis' project <https://r4epis.netlify.com>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Imports binom, scales, dplyr (>= 0.8.0), rlang, forcats, tidyr (>= 1.0.0), tibble, glue, tidyselect

Suggests testthat (>= 2.1.0), outbreaks, epidict, covr, knitr, magrittr, rmarkdown

Additional_repositories https://r4epi.github.io/drat

URL https://github.com/R4EPI/epikit, https://r4epis.netlify.com, https://r4epi.github.io/epikit

BugReports https://github.com/R4EPI/epikit/issues

VignetteBuilder knitr

NeedsCompilation no

Author Zhian N. Kamvar [aut, cre] (<https://orcid.org/0000-0003-1458-7108>), Dirk Schumacher [aut], Alex Spina [ctb], Kate Doyle [ctb]

Maintainer Zhian N. Kamvar <zkamvar@gmail.com>

Repository CRAN

Date/Publication 2020-03-05 20:40:02 UTC

R topics documented:

`age_categories` *Create an age group variable*

Description

Create an age group variable

Usage

```
age_categories(
  x,
  breakers = NULL,
  lower = 0,
  upper = NULL,
  by = 10,
  separator = "-",
  ceiling = FALSE,
  above.char = "+"
)

group_age_categories(
  dat,
  years = NULL,
  months = NULL,
  weeks = NULL,
  days = NULL,
  one_column = TRUE,
  drop_empty_overlaps = TRUE
)
```

Arguments

`x`	Your age variable
`breakers`	A string. Age category breaks you can define within c(). Alternatively use "lower", "upper" and "by" to set these breaks based on a sequence.

lower	A number. The lowest age value you want to consider (default is 0)
upper	A number. The highest age value you want to consider
by	A number. The number of years you want between groups
separator	A character that you want to have between ages in group names. The default is "-" producing e.g. 0-10.
ceiling	A TRUE/FALSE variable. Specify whether you would like the highest value in your breakers, or alternatively the upper value specified, to be the endpoint. This would produce the highest group of "70-80" rather than "80+". The default is FALSE (to produce a group of 80+).
above.char	Only considered when ceiling == FALSE. A character that you want to have after your highest age group. The default is "+" producing e.g. 80+
dat	a data frame with at least one column defining an age category
years, months, weeks, days	the bare name of the column defining years, months, weeks, or days (or NULL if the column doesn't exist)
one_column	if TRUE (default), the categories will be joined into a single column called "age_category" that appends the type of age category used. If FALSE, there will be one column with the grouped age categories called "age_category" and a second column indicating age unit called "age_unit".
drop_empty_overlaps	if TRUE, unused levels are dropped if they have been replaced by a more fine-grained definition and are empty. Practically, this means that the first level for years, months, and weeks are in consideration for being removed via forcats::fct_drop()

Value

a factor representing age ranges, open at the upper end of the range.

a data frame

Examples

```
if (interactive() && require("dplyr") && require("epidict")) {
withAutoprint({
set.seed(50)
dat <- epidict::gen_data("Cholera", n = 100, org = "MSF")
ages <- dat %>%
  select(starts_with("age")) %>%
  mutate(age_years = age_categories(age_years, breakers = c(0, 5, 10, 15, 20))) %>%
  mutate(age_months = age_categories(age_months, breakers = c(0, 5, 10, 15, 20))) %>%
  mutate(age_days = age_categories(age_days, breakers = c(0, 5, 15)))

ages %>%
  group_age_categories(years = age_years, months = age_months, days = age_days) %>%
  pull(age_category) %>%
  table()
})
}
```

attack_rate *Rates and Ratios*

Description

Calculate attack rate, case fatality rate, and mortality rate

Usage

```
attack_rate(
  cases,
  population,
  conf_level = 0.95,
  multiplier = 100,
  mergeCI = FALSE,
  digits = 2
)

case_fatality_rate(
  deaths,
  population,
  conf_level = 0.95,
  multiplier = 100,
  mergeCI = FALSE,
  digits = 2
)

case_fatality_rate_df(
  x,
  deaths,
  group = NULL,
  conf_level = 0.95,
  multiplier = 100,
  mergeCI = FALSE,
  digits = 2,
  add_total = FALSE
)

mortality_rate(
  deaths,
  population,
  conf_level = 0.95,
  multiplier = 10^4,
  mergeCI = FALSE,
  digits = 2
)
```

Arguments

cases, deaths	number of cases or deaths in a population. For _df functions, this can be the name of a logical column OR an evaluated logical expression (see examples).
population	the number of individuals in the population.
conf_level	a number representing the confidence level for which to calculate the confidence interval. Defaults to 0.95, representing a 95% confidence interval using binom::binom.wilson()
multiplier	The base by which to multiply the output: • multiplier = 1: ratio between 0 and 1 • multiplier = 100: proportion • multiplier = 10^4: x per 10,000 people
mergeCI	Whether or not to put the confidence intervals in one column (default is FALSE)
digits	if mergeCI = TRUE, this determines how many digits are printed
x	a data frame
group	the bare name of a column to use for stratifying the output
add_total	if group is not NULL, then this will add a row containing the total value across all groups.

Value

a data frame with five columns that represent the numerator, denominator, rate, lower bound, and upper bound.

- attack_rate(): cases, population, ar, lower, upper
- case_fatality_rate(): deaths, population, cfr, lower, upper

Examples

```
# Attack rates can be calculated with just two numbers
print(ar <- attack_rate(10, 50), digits = 4) # 20% attack rate

# print them inline using `fmt_ci_df()`
fmt_ci_df(ar)

# Alternatively, if you want one column for the CI, use `mergeCI = TRUE`
attack_rate(10, 50, mergeCI = TRUE, digits = 2) # 20% attack rate

print(cfr <- case_fatality_rate(1, 100), digits = 2) # CFR of 1%
fmt_ci_df(cfr)

# using a data frame
if (require("outbreaks")) {
  withAutoprint({
  e <- outbreaks::ebola_sim$linelist
  case_fatality_rate_df(e,
    outcome == "Death",
    group = gender,
```

```
    add_total = TRUE,
    mergeCI = TRUE
  )
  })
}
```

`fac_from_num` *create factors from numbers*

Description

If the number of unique numbers is five or fewer, then they will simply be converted to factors in order, otherwise, they will be passed to cut and pretty, preserving the lowest value.

Usage

```
fac_from_num(x)
```

Arguments

`x`	a vector of integers or numerics

Value

a factor

Examples

```
fac_from_num(1:100)
fac_from_num(sample(100, 5))
```

`find_breaks` *Automatically calculate breaks for a number*

Description

Automatically calculate breaks for a number

Usage

```
find_breaks(n, breaks = 4, snap = 1, ceiling = FALSE)
```

Arguments

`n`	a number to calcluate breaks for
`breaks`	the maximum number of segements you want to have
`snap`	the number defining where to snap to the nearest factor
`ceiling`	if `TRUE`, n is included in the breaks

Value

a vector of integers

Examples

```
# find four breaks from 1 to 100
find_breaks(100)

# find four breaks from 1 to 123, rounding to the nearest 20
find_breaks(123, snap = 20)

# note that there are only three breaks here because of the rounding
find_breaks(123, snap = 25)

# Include the value itself
find_breaks(123, snap = 25, ceiling = TRUE)
```

---

fmt_ci *Helper to format confidence interval for text*

Description

This function is mainly used for placing in the text fields of Rmarkdown reports. You can use it by writing it in something like this:

```
The CFR for Bamako is `r fmt_pci(case_fatality_rate(10, 50))`
```

which will render like this: "The CFR for Bamako is 20.00\

Usage

```
fmt_ci(e = numeric(), l = numeric(), u = numeric(), digits = 2, percent = TRUE)

fmt_pci(
  e = numeric(),
  l = numeric(),
  u = numeric(),
  digits = 2,
  percent = TRUE
)

fmt_pci_df(x, e = 3, l = e + 1, u = e + 2, digits = 2, percent = TRUE)

fmt_ci_df(x, e = 3, l = e + 1, u = e + 2, digits = 2, percent = TRUE)
```

Arguments

e	the column of the estimate (defaults to the third column). Otherwise, a number
l	the column of the lower bound (defaults to the fourth column). Otherwise, a number
u	the column of the upper bound (defaults to the fifth column), otherwise, a number
digits	the number of digits to show
percent	if TRUE (default), converts the number to percent, otherwise it's treated as a raw value
x	a data frame

Value

a text string in the format of "e\

Examples

```
cfr <- data.frame(x = 1, y = 2, est = 0.5, lower = 0.25, upper = 0.75)
fmt_pci_df(cfr)

# If the data starts at a different column, specify a different number
fmt_pci_df(cfr[-1], 2, d = 1)

# It's also possible to provide numbers directly and remove the percent sign.
fmt_ci(pi, pi - runif(1), pi + runif(1), percent = FALSE)
```

---

fmt_count *Counts and proportions inline*

Description

These functions will give proportions for different variables inline.

Usage

```
fmt_count(x, ...)
```

Arguments

x	a data frame
...	an expression or series of expressions to pass to dplyr::filter()

Value

a one-element character vector of the format "n (%)"

Examples

```
fmt_count(mtcars, cyl > 3, hp < 100)
fmt_count(iris, Species == "virginica")
```

`rename_redundant` *Cosmetically relabel all columns that contains a certain pattern*

Description

These function are only to be used cosmetically before kable and will likely return a data frame with duplicate names.

Usage

```
rename_redundant(x, ...)

augment_redundant(x, ...)
```

Arguments

`x`	a data frame
`...`	a series of keys and values to replace columns that match specific patterns.

Details

- rename_redundant fully replaces any column names matching the keys
- augment_redundant will take a regular expression and rename columns via `gsub()`.

Value

a data frame.

Author(s)

Zhian N. Kamvar

Examples

```
df <- data.frame(
  x = letters[1:10],
  `a n` = 1:10,
  `a prop` = (1:10) / 10,
  `a deff` = round(pi, 2),
  `b n` = 10:1,
  `b prop` = (10:1) / 10,
  `b deff` = round(pi * 2, 2),
```

```
  check.names = FALSE
)
df
print(df <- rename_redundant(df, "%" = "prop", "Design Effect" = "deff"))
print(df <- augment_redundant(df, " (n)" = " n$"))
```

`unite_ci` *Unite estimates and confidence intervals*

Description

create a character column by combining estimate, lower and upper columns. This is similar to `tidyr::unite()`.

Usage

```
unite_ci(
  x,
  col = NULL,
  ...,
  remove = TRUE,
  digits = 2,
  m100 = TRUE,
  percent = FALSE,
  ci = FALSE
)

merge_ci_df(x, e = 3, l = e + 1, u = e + 2, digits = 2)

merge_pci_df(x, e = 3, l = e + 1, u = e + 2, digits = 2)
```

Arguments

`x`	a data frame with at least three columns defining an estimate, lower bounds, and upper bounds.
`col`	the quoted name of the replacement column to create
`...`	three columns to bind together in the order of Estimate, Lower, and Upper.
`remove`	if `TRUE` (default), the three columns in `...` will be replaced by `col`
`digits`	the number of digits to retain for the confidence interval.
`m100`	`TRUE` if the result should be multiplied by 100
`percent`	`TRUE` if the result should have a percent symbol added.
`ci`	`TRUE` if the result should include "CI" within the braces (defaults to FALSE)
`e`	the column of the estimate (defaults to the third column). Otherwise, a number
`l`	the column of the lower bound (defaults to the fourth column). Otherwise, a number
`u`	the column of the upper bound (defaults to the fifth column), otherwise, a number

Value

a modified data frame with merged columns or one additional column representing the estimate and confidence interval

Examples

```
fit <- lm(100/mpg ~ disp + hp + wt + am, data = mtcars)
df  <- data.frame(v = names(coef(fit)), e = coef(fit), confint(fit), row.names = NULL)
names(df) <- c("variable", "estimate", "lower", "upper")
print(df)
unite_ci(df, "slope (CI)", estimate, lower, upper, m100 = FALSE, percent = FALSE)
```

Index