Package ‘supportR’

February 9, 2023

Type Package
Title Support Functions for Wrangling and Visualization
Version 1.0.0
Maintainer Nicholas J Lyon <njlyon@alumni.iastate.edu>
Description Suite of helper functions for data wrangling and visualization.
The only theme for these functions is that they tend towards simple, short, and narrowly-scoped.
These functions are built for tasks that often recur but are not large enough in scope to warrant an ecosystem of interdependent functions.
License MIT + file LICENSE
Language en-US
Encoding UTF-8
RoxygenNote 7.2.3
Depends R (>= 3.5)
Imports dplyr, ggplot2, googledrive, graphics, lifecycle, magrittr,
       methods, purrr, rlang, rmarkdown, stringr, tidyr, vegan
Suggests ape, knitr, palmerpenguins
VignetteBuilder knitr
NeedsCompilation no
Author Nicholas J Lyon [aut, cre] (<https://orcid.org/0000-0003-3905-1078>)
Repository CRAN
Date/Publication 2023-02-09 14:00:02 UTC

R topics documented:

array_melt ................................................................. 2
crop_tri ................................................................. 3
date_check .............................................................. 3
date_format_guess ...................................................... 4
diff_check ............................................................... 5
multi_date_check ....................................................... 6
array_melt

Melt an Array into a Dataframe

Description
Melts an array of dimensions x, y, and z into a dataframe containing columns x, y, z, and value
where value is whatever was stored in the array at those coordinates.

Usage
array_melt(array = NULL)

Arguments
array (array) array object to melt into a dataframe

Value
(dataframe) object containing the "flattened" array in dataframe format

Examples
# First we need to create an array to melt
## Make data to fill the array
vec1 <- c(5, 9, 3)
vec2 <- c(10:15)

## Create dimension names (x = col, y = row, z = which matrix)
x_vals <- c("Col_1","Col_2","Col_3")
y_vals <- c("Row_1","Row_2","Row_3")
z_vals <- c("Mat_1","Mat_2")

## Make an array from these components
g <- array(data = c(vec1, vec2), dim = c(3, 3, 2),
            dimnames = list(x_vals, y_vals, z_vals))

## "Melt" the array into a dataframe
array_melt(array = g)
**crop_tri**

_Crop a Triangle from Data Object_

**Description**

Accepts a symmetric data object and replaces the chosen triangle with NAs. Also allows user to choose whether to keep or drop the diagonal of the data object.

**Usage**

crop_tri(data = NULL, drop_tri = "upper", drop_diag = FALSE)

**Arguments**

type: (dataframe, dataframe-like, or matrix) symmetric data object to remove one of the triangles from

drop_tri: (character) which triangle to replace with NAs, either "upper" or "lower"

drop_diag: (logical) whether to drop the diagonal of the data object (defaults to FALSE)

**Value**

(dataframe or dataframe-like) data object with desired triangle removed and either with or without the diagonal

---

**date_check**

_Check a Column for Non-Dates_

**Description**

Any elements in the column that would be changed to NA if as.Date is used on the column are returned. This is useful for quickly identifying only the "problem" entries of an ostensibly date column that is read in as a character.

**Usage**

date_check(data = NULL, col = NULL)

**Arguments**

type: (dataframe) object containing at least one column of supposed dates

col: (character or numeric) name or column number of the column containing putative dates in the data object

**Value**

(character) vector of malformed dates
Examples

# Let's create some data that will be useful in demoing this function
loc <- c("LTR", "GIL", "PYN", "RIN")
time <- c('2021-01-01', '2021-01-0w', '1990', '2020-10-xx')
sites <- data.frame('site' = loc, 'visit' = time)

# Now we can use our function to identify bad dates
date_check(data = sites, col = 'visit')

---

date_format_guess   Identify Probable Format for Ambiguous Date Formats

Description

In a column containing multiple date formats (e.g., MM/DD/YYYY, "YYYY/MM/DD, etc.) identifies probable format of each date. Provision of a grouping column improves inference. Any formats that cannot be determined are flagged as "FORMAT UNCERTAIN" for human double-checking. This is useful for quickly sorting the bulk of ambiguous dates into clear categories for later conditional wrangling.

Usage

date_format_guess(
  data = NULL,
  date_col = NULL,
  groups = TRUE,
  group_col = NULL,
  return = "dataframe",
  quiet = FALSE
)

Arguments

data (dataframe) object containing at least one column of ambiguous dates
date_col (character) name of column containing ambiguous dates
groups (logical) whether groups exist in the dataframe / should be used (defaults to TRUE)
group_col (character) name of column containing grouping variable
return (character) either "dataframe" or "vector" depending on whether the user wants the date format "guesses" returned as a new column on the dataframe or a vector
quiet (logical) whether certain optional messages should be displayed (defaults to FALSE)

Value

(dataframe or character) object containing date format guesses
### Examples

# Create dataframe of example ambiguous dates & grouping variable
my_df <- data.frame('data_enterer' = c('person A', 'person B', 'person B', 'person C', 'person D', 'person E', 'person F', 'person G'),
'bad_dates' = c('2022.13.08', '2021/2/02', '2021/2/03', '2021/2/04', '1899/1/15', '10-31-1901', '26/11/1901', '08.11.2004', '6/10/02'))

# Now we can invoke the function!
date_format_guess(data = my_df, date_col = "bad_dates", group_col = "data_enterer", return = "dataframe")

# If preferred, do it without groups and return a vector
date_format_guess(data = my_df, date_col = "bad_dates", groups = FALSE, return = "vector")

---

**diff_check**

**Compare Difference Between Two Vectors**

**Description**

Reflexively compares two vectors and identifies (1) elements that are found in the first but not the second (i.e., "lost" components) and (2) elements that are found in the second but not the first (i.e., "gained" components). This is particularly helpful when manipulating a dataframe and comparing what columns are lost or gained between wrangling steps. Alternately it can compare the contents of two columns to see how two dataframes differ.

**Usage**

diff_check(old = NULL, new = NULL, sort = TRUE, return = FALSE)

**Arguments**

- **old** (vector) starting / original object
- **new** (vector) ending / modified object
- **sort** (logical) whether to sort the difference between the two vectors
- **return** (logical) whether to return the two vectors as a 2-element list

**Value**

No return value (unless return = T), called for side effects. If return = T, returns a two-element list
Examples

# Make two vectors
vec1 <- c("x", "a", "b")
vec2 <- c("y", "z", "a")

# Compare them!
diff_check(old = vec1, new = vec2, return = FALSE)

# Return the difference for later use
diff_out <- diff_check(old = vec1, new = vec2, return = TRUE)
diff_out

multi_date_check  Check Multiple Columns for Non-Dates

Description

Any elements in the columns that would be changed to NA if as.Date is used are returned. This is useful for quickly identifying only the “problem” entries of ostensibly date columns that are read in as characters. This function is an extension of the date_check function in this package that deals with only a single column.

Usage

multi_date_check(data = NULL, col_vec = NULL)

Arguments

data  (dataframe) object containing at least one column of supposed dates

col_vec  (character or numeric) vector of names or column numbers of the columns containing putative dates in the data object

Value

list of same length as col_vec with malformed dates from each column in their respective element

Examples

# Make a dataframe to test the function
loc <- c("LTR", "GIL", "PYN", "RIN")
time <- c('2021-01-01', '2021-01-0w', '1990', '2020-10-xx')
time2 <- c('1880-08-08', '2021-01-02', '1992', '2049-11-01')
time3 <- c('2022-10-31', 'tomorrow', '1993', NA)

# Assemble our vectors into a dataframe
sites <- data.frame('site' = loc, 'first_visit' = time, 'second' = time2, 'third' = time3)

# Use `multi_date_check()` to return only the entries that would be lost
multi_date_check(data = sites, col_vec = c("first_visit", "second", "third"))
**multi_num_check**

**Check Multiple Columns for Non-Numbers**

**Description**

Any elements in the columns that would be changed to NA if `as.numeric` is used are returned. This is useful for quickly identifying only the "problem" entries of ostensibly numeric columns that are read in as characters. This function is an extension of the `num_check` function in this package that deals with only a single column.

**Usage**

```r
multi_num_check(data = NULL, col_vec = NULL)
```

**Arguments**

- `data` (dataframe) object containing at least one column of supposed numbers
- `col_vec` (character or numeric) vector of names or column numbers of the columns containing putative numbers in the data object

**Value**

list of same length as `col_vec` with malformed numbers from each column in their respective element

**Examples**

```r
# Create dataframe with a numeric column where some entries would be coerced into NA
spp <- c('salmon', 'bass', 'halibut', 'eel')
ct <- c(1, '14x', '_23', 12)
ct2 <- c('a', '2', '4', '0')
ct3 <- c(NA, 'Y', 'typo', '2')
fish <- data.frame(species = spp, count = ct, num_col2 = ct2, third_count = ct3)

# Use `multi_num_check()` to return only the entries that would be lost
multi_num_check(data = fish, col_vec = c("count", "num_col2", "third_count"))
```

---

**nms_ord**

**Publication-Quality Non-metric Multi-dimensional Scaling (NMS) Ordinations**

**Description**

Produces Non-Metric Multi-dimensional Scaling (NMS) ordinations for up to 10 groups. Assigns a unique color for each group and draws an ellipse around the standard deviation of the points. Automatically adds stress (see `vegan::metaMDS` for explanation of "stress") as legend title. Because there are only five hollow shapes (see `?graphics::pch()`) all shapes are re-used a maximum of 2 times when more than 5 groups are supplied.
Usage

nms_ord(
  mod = NULL,
  groupcol = NULL,
  title = NA,
  colors = c("#41b6c4", "#c51b7d", "#7fbc41", "#d73027", "#4575b4", "#e08214", "#8073ac", "#f1b6da", "#b8e186", "#8c96c6"),
  shapes = rep(x = 21:25, times = 2),
  lines = rep(x = 1, times = 10),
  leg_pos = "bottomleft",
  leg_cont = unique(groupcol)
)

Arguments

mod Object returned by vegan::metaMDS
groupcol (dataframe) column specification in the data that includes the groups (accepts either bracket or $ notation)
title (character) string to use as title for plot
colors (character) vector of colors (as hexadecimal codes) of length >= group levels (default not colorblind safe because of need for 10 built-in unique colors)
shapes (numeric) vector of shapes (as values accepted by pch) of length >= group levels
lines (numeric) vector of line types (as integers) of length >= group levels
leg_pos (character or numeric) legend position, either numeric vector of x/y coordinates or shorthand accepted by graphics::legend
leg_cont (character) vector of desired legend entries. Defaults to unique entries in groupcol argument (this argument provided in case syntax of legend contents should differ from data contents)

Value

(base R plot) base R plot with ellipses for each group

Examples

# Use data from the vegan package
utils::data("varespec", package = ‘vegan’)  
resp <- varespec

# Make some columns of known number of groups
factor_4lvl <- c(rep.int("Trt1", (nrow(resp)/4)),
                 rep.int("Trt2", (nrow(resp)/4)),
                 rep.int("Trt3", (nrow(resp)/4)),
                 rep.int("Trt4", (nrow(resp)/4)))

# And combine them into a single data object
data <- cbind(factor_4lvl, resp)
# Actually perform multidimensional scaling
mds <- vegan::metaMDS(data[-1], autotransform = FALSE, expand = FALSE, k = 2, try = 50)

# With the scaled object and original dataframe we can use this function
nms_ord(mod = mds, groupcol = data$factor_4lvl,
    title = '4-Level NMS', leg_pos = 'topright',
    leg_cont = c('1', '2', '3', '4'))

---

### num_check

**Check a Column for Non-Numbers**

**Description**

Any elements in the column that would be changed to NA if `as.numeric` is used on the column are returned. This is useful for quickly identifying only the "problem" entries of an ostensibly numeric column that is read in as a character.

**Usage**

```r
num_check(data = NULL, col = NULL)
```

**Arguments**

- `data` (dataframe) object containing at least one column of supposed dates
- `col` (character or numeric) name or column number of the column containing putative dates in the data object

**Value**

(character) vector of malformed dates

**Examples**

```r
# Create dataframe with a numeric column where some entries would be coerced into NA
spp <- c('salmon', 'bass', 'halibut', 'eel')
ct <- c(1, '14x', '_23', 12)
fish <- data.frame('species' = spp, 'count' = ct)

# Use `num_check()` to return only the entries that would be lost
num_check(data = fish, col = "count")
```
Description

Produces Principal Coordinates Analysis (PCoA) ordinations for up to 10 groups. Assigns a unique color for each group and draws an ellipse around the standard deviation of the points. Automatically adds percent of variation explained by first two principal component axes parenthetically to axis labels. Because there are only five hollow shapes (see ?graphics::pch) all shapes are re-used a maximum of 2 times when more than 5 groups are supplied.

Usage

```r
pcoa_ord(
  mod = NULL,
  groupcol = NULL,
  title = NA,
  colors = c("#41b6c4", "#c51b7d", "#7fbc41", "#d73027", "#4575b4", "#e08214", "#8073ac", "#f1b6da", "#b8e186", "#8c96c6"),
  shapes = rep(x = 21:25, times = 2),
  lines = rep(x = 1, times = 10),
  leg_pos = "bottomleft",
  leg_cont = unique(groupcol)
)
```

Arguments

- `mod`: Object returned by `ape::pcoa`
- `groupcol`: (dataframe) column specification in the data that includes the groups (accepts either bracket or $ notation)
- `title`: (character) string to use as title for plot
- `colors`: (character) vector of colors (as hexadecimal codes) of length >= group levels (default not colorblind safe because of need for 10 built-in unique colors)
- `shapes`: (numeric) vector of shapes (as values accepted by `pch`) of length >= group levels
- `lines`: (numeric) vector of line types (as integers) of length >= group levels
- `leg_pos`: (character or numeric) legend position, either numeric vector of x/y coordinates or shorthand accepted by `graphics::legend`
- `leg_cont`: (character) vector of desired legend entries. Defaults to unique entries in `groupcol` argument (this argument provided in case syntax of legend contents should differ from data contents)

Value

(base R plot) base R plot with ellipses for each group
Examples

# Use data from the vegan package
data("varespec", package = 'vegan')
resp <- varespec

# Make some columns of known number of groups
factor_4lvl <- c(rep.int("Trt1", (nrow(resp)/4)),
                rep.int("Trt2", (nrow(resp)/4)),
                rep.int("Trt3", (nrow(resp)/4)),
                rep.int("Trt4", (nrow(resp)/4)))

# And combine them into a single data object
data <- cbind(factor_4lvl, resp)

# Get a distance matrix from the data
dist <- vegan::vegdist(resp, method = 'kulczynski')

# Perform a PCoA on the distance matrix to get points for an ordination
pnts <- ape::pcoa(dist)

# Test the function for 4 groups
pcoa_ord(mod = pnts, groupcol = data$factor_4lvl)

---

rmd_export

Knit an R Markdown File and Export to Google Drive

Description

This function allows you to knit a specified R Markdown file locally and export it to the Google Drive folder for which you provided a link. NOTE that if you have not used googledrive::drive_auth this will prompt you to authorize a Google account in a new browser tab. If you do not check the box in that screen before continuing you will not be able to use this function until you clear your browser cache and re-authenticate. I recommend invoking drive_auth beforehand to reduce the chances of this error

Usage

rmd_export(
  rmd = NULL,
  out_path = getwd(),
  out_name = NULL,
  out_type = "html",
  drive_link
)
summary_table

Generate Summary Table for Supplied Response and Grouping Variables

Description

Calculates mean, standard deviation, sample size, and standard error of a given response variable within user-defined grouping variables. This is meant as a convenience instead of doing dplyr::group_by followed by dplyr::summarize iteratively themselves.

Usage

```
summary_table(
  data = NULL,
  groups = NULL,
  response = NULL,
  drop_na = FALSE,
  round_digits = 2
)
```
theme_lyon

Arguments

data  (dataframe or dataframe-like) object with column names that match the values passed to the groups and response arguments

groups (character) vector of column names to group by

response (character) name of the column name to calculate summary statistics for (the column must be numeric)

drop_na (logical) whether to drop NAs in grouping variables. Defaults to FALSE

round_digits (numeric) number of digits to which mean, standard deviation, and standard error should be rounded

Value

(dataframe) summary table containing the mean, standard deviation, sample size, and standard error of the supplied response variable)

theme_lyon Complete ggplot2 Theme for Non-Data Aesthetics

Description

Custom alternative to the ggtheme options built into ggplot2. Removes gray boxes and grid lines from plot background. Increases font size of tick marks and axis labels. Removes gray box from legend background and legend key. Removes legend title.

Usage

theme_lyon(title_size = 16, text_size = 13)

Arguments

title_size (numeric) size of font in axis titles

text_size (numeric) size of font in tick labels

Value

(ggplot theme) list of ggplot2 theme elements
Index

array_melt, 2

crop_tri, 3

date_check, 3
date_format_guess, 4
diff_check, 5

multi_date_check, 6
multi_num_check, 7

nms_ord, 7
num_check, 9

pcoa_ord, 10

rmd_export, 11

summary_table, 12

theme_lyon, 13