Package 'cheese'

April 1, 2019

Version 0.0.1

Date 2019-03-12

Title Tools for Intuitive and Flexible Statistical Analysis Workflows

Description Contains flexible and intuitive functions to assist in carrying out tasks in a statistical analysis and to get from the raw data to presentation-ready results. A user-friendly interface is used in specialized functions that are aimed at common tasks such as building a univariate descriptive table for variables in a dataset. These high-level functions are built on a collection of low(er)-level functions that may be useful for aspects of a custom statistical analysis workflow or for general programming use.

URL https://github.com/zajichek/cheese

License MIT + file LICENSE

Depends R (>= 3.4.0)

Imports dplyr (>= 0.7.7), forcats (>= 0.3.0), kableExtra (>= 1.0.1), knitr (>= 1.20), magrittr (>= 1.5), methods (>= 3.4.1), purrr (>= 0.2.4), rlang (>= 0.3.0.1), stringr (>= 1.3.1), tibble (>= 1.4.2), tidyr (>= 0.8.1), tidyselect (>= 0.2.4)

Suggests rmarkdown (>= 1.10), tidyverse (>= 1.2.1)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

NeedsCompilation no

Author Alex Zajichek [aut, cre]

Maintainer Alex Zajichek <alexzajichek@gmail.com>

Repository CRAN

Date/Publication 2019-04-01 08:10:03 UTC

R topics documented:

`absorb` *Absorb values of a key-value pair into an arbitrary character string containing keys*

Description

Populates user-specified string templates containing keys as placeholders, with the values. The keys are interpreted as regular expressions. Results can optionally be evaluated as R expressions.

Usage

```
absorb(
    key,
    value,
    text,
    sep = "|",
    print = FALSE,
    evaluate = FALSE
)
```

Arguments

`key`	Vector of keys that can be coerced to type `character`.
`value`	Vector of values with positions corresponding to the `key`.
`text`	Vector of character strings containing sequences of characters and keys/patterns where values should be filled.
`sep`	Character to separate values by in the placeholder in the event of duplicate keys (patterns). Defaults to `"|"`
`print`	Should the recursion results be printed to the console each iteration? Defaults to `FALSE`.
`evaluate`	Should the resulting strings be evaluated as R expressions? Defaults to `FALSE`.

Details

The algorithm iterates the provided `value` vector (in sequential order) and recursivley replaces substrings where there is a matching pattern of a `key`. Thus, it is possible that a subsequent key could match with a previous value, and hence be replaced more than once. If duplicate keys exist, the placeholder will be filled with a collapsed string of all the values for that key.

Value

If `evaluate = FALSE` (default), a character vector the same length as `text` with all matching patterns replaced by their value. If `evaluate = TRUE`, a list with the same length as `text` is returned with the result of the evaluation of each string.

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#1) Simple example
absorb(
    key = c("mean", "sd", "var"),
    value = c("10", "2", "4"),
    text =
        c("MEAN: mean, SD: sd",
          "VAR: var = sd^2",
          MEAN = "mean"
        )
)

#2) Evaluating results
absorb(
    key = c("mean", "mean", "sd", "var"),
    value = c("10", "20", "2", "4"),
    text = c("(mean)/2", "sd^2"),
    sep = "+",
    evaluate = TRUE
) %>%
    flatten_dbl()
```

`descriptives` *Compute descriptive statistics on columns of a data frame.*

Description

Computes a number of common descriptive statistics for different types of data. The user can specify an unlimited number of additional functions to compute and the types of data that each set (including the default) of functions will be applied to.

Usage

```
descriptives(
    data,
    f_all = NULL,
    f_numeric = NULL,
    numeric_types = "numeric",
    f_categorical = NULL,
    categorical_types = "factor",
    f_other = NULL,
    na.rm = TRUE,
    useNA = c("ifany", "no", "always"),
    round = 2
)
```

Arguments

data	A data.frame. Could also be a list.
f_all	Functions to apply to all columns. Should return a scalar. See "Details" for information computed by default.
f_numeric	Functions to apply to columns conforming to numeric_types. Should return a scalar. See "Details" for information computed by default.
numeric_types	Character vector of data types that should be evaluated with f_numeric.
f_categorical	Functions to apply to columns conforming to categorical_types. Should return a named vector where the names correpond to the levels. See "Details" for information computed by default.
categorical_types	Character vector of data types that should be evaluated with f_categorical.
f_other	Functions to apply to remaining columns.
na.rm	Logical argument supplied to f_numeric. Defaults to TRUE.
useNA	Supplied to f_categorical. See ?base::table for details. Defaults to "ifany".
round	Digit to round numeric data. Defaults to 2.

Details

The min, max, median, iqr, mean, sd are automatically computed for numeric data and table, prop.table*100 for categorical data. The sample size, number of missing values, number of nonmissing values, the number of unique values, the class are automatically computed on all columns.

Value

A long `tibble` with columns `.variable` (for the variable name), `.key` (for the statistic or attribute), `.value` (for numeric results), `.label` (for non-numeric results), and `.combo` (convenient combination of .value and .label coerced to a `character` vector). If categorical variables exist, additional columns `.level` (for the factor levels) and `.order` (to retain order of the levels).

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#1) Default
heart_disease %>%
    descriptives()

#2) Allow logicals as categorical
heart_disease %>%
    descriptives(
        categorical_types = c("logical", "factor")
    )

#3) Only apply "other" functions to numeric types
heart_disease %>%
    descriptives(
        numeric_types = NULL
    )

#4) Compute a custom function
heart_disease %>%
    descriptives(
        f_numeric =
            list(
                cv = function(x, na.rm) sd(x, na.rm = na.rm)/mean(x, na.rm = na.rm)
            )
    )
```

`dish` *Dish out a function to combinations of variables*

Description

Evaluates a two argument function on subsets of a data frame by evaluating each combination of columns or subsets.

Usage

```
dish(
    data,
    f,
    left = NULL,
    right = NULL,
    each_left = TRUE,
    each_right = TRUE,
    bind = FALSE,
    ...
)
```

Arguments

data	A data.frame.
f	Any function that takes a vector and/or data.frame in the first two arguments.
left	Variables to be used in the first argument of f. If NULL (default), all variables except those in right are used. Has tidyselect::vars_select capabilities.
right	Variables to be used in the second argument of f. If NULL (default), all variables except those in left are used. Has tidyselect::vars_select capabilities.
each_left	Should each left variable be separately evaluated in f? Defaults to TRUE. If FALSE, left is entered into f as a data.frame.
each_right	Should each right variable be separately evaluated in f? Defaults to TRUE. If FALSE, right is entered into f as a data.frame.
bind	Should results be binded into a single data.frame? Defaults to FALSE.
...	Additional arguments to be passed to f.

Value

A list (if bind = FALSE) or a tibble (if bind = TRUE) with the results of f evaluated on data subsets.

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#1) Default uses every variable on both sides
heart_disease %>%
    select_if(
        is.numeric
    ) %>%
    dish(
```

```
        f = cor
    )

#2) Simple regression of Age and BP on each variable
heart_disease %>%
    dish(
        f =
            function(y, x) {
                mod <- lm(y ~ x)
                tibble(
                    Parameter = names(mod$coef),
                    Estimate = mod$coef
                )
            },
        left = c("Age", "BP"),
        bind = TRUE
    )

#3) Multiple regression with Age, BP on all variables simultaneously
heart_disease %>%
    dish(
        f =
            function(y, x) {
                mod <- lm(y ~ ., data = x)
                tibble(
                    Parameter = names(mod$coef),
                    Estimate = mod$coef
                )
            },
        left = c("Age", "BP"),
        each_right = FALSE,
        bind = TRUE
    )
```

`divide` *Divide a data frame into a list by any number of variables.*

Description

Separates a data frame by one or more stratification variables into a list of data frames whose names are the strata. Removes stratification variables from each frame.

Usage

```
divide(
    data,
    by,
    sep = "|"
)
```

Arguments

data	Any data.frame.
by	Character vector of variable names or a tidyselect::select_helper.
sep	String to separate values of each stratification variable by. Defaults to "|".

Value

A named list of data.frame objects

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#1) Exact match
heart_disease %>%
    divide(
        by = "Sex"
    )

#2) Regular expression
heart_disease %>%
    divide(
        by = matches("^(S|H)")
    )

#3) Beginning character
heart_disease %>%
    divide(
        by = starts_with("S")
    )

#4) Negation
heart_disease %>%
    select(
        Sex,
        HeartDisease,
        BloodSugar
    ) %>%
    divide(
        by = -matches("^(S|H)")
)
```

`heart_disease` *Heart Disease*

Description

This is a cleaned up version of the "heart disease data set" found in the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Heart+Disease), containing a subset of the default variables.

Usage

```
heart_disease
```

Format

See "Source" for link to dataset home page

Source

https://archive.ics.uci.edu/ml/datasets/Heart+Disease

`stratiply` *Stratify a data frame, apply a function to each strata, and collect results*

Description

Calls `"divide"` to separate a data frame into one or more stratified frames, applies a function to each strata, and optionally collects the results and returns strata to their original columns.

Usage

```
stratiply(
    data,
    strata,
    f,
    delimiter = "|",
    bind = FALSE,
    separate = FALSE,
    ...
)
```

Arguments

data	Any data.frame.
strata	See the by argument in "divide".
f	Any function that takes a data.frame as an argument.
delimiter	See the sep argument in "divide". Also used for salvaging original columns.
bind	Should the results be binded back to a single data.frame? Defaults to FALSE.
separate	If bind = TRUE, should the stratification variables be separated back to their original columns? Defaults to FALSE.
...	Additional arguments passed to f.

Value

If bind = FALSE, a named list with the results of f for each strata. Otherwise the results are binded to a single frame.

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#1) Return a list
heart_disease %>%
    stratiply(
        strata = c("Sex", "HeartDisease"),
        f = head
    )

#2) Unseparated strata column
heart_disease %>%
    stratiply(
        strata = c("Sex", "HeartDisease"),
        f = head,
        bind = TRUE
    )

#3) Separated strata column
heart_disease %>%
    stratiply(
        strata = c("Sex", "HeartDisease"),
        f = head,
        bind = TRUE,
        separate = TRUE
    )

#4) Custom function
heart_disease %>%
```

```
    stratiply(
        strata = c("Sex", "HeartDisease"),
        f = function(x)
            x %>%
            select_if(is.numeric) %>%
            map(mean, na.rm = TRUE),
        bind = TRUE,
        separate = TRUE
    )

#5) Regular expression
heart_disease %>%
    stratiply(
        strata = matches("^(S|H)"),
        f = function(x)
            x %>%
            select_if(is.numeric) %>%
            map(mean, na.rm = TRUE),
        bind = TRUE,
        separate = TRUE
    )
```

`stretch` *Stretch one or variables over many columns by one or more keys*

Description

Provides similar functionality to `tidyr::spread` or `reshape2::dcast` but allows for an unlimited number of variables to be spanned across the columns.

Usage

```
stretch(
    data,
    keys,
    keep = NULL,
    send = NULL,
    join = dplyr::full_join,
    .sep = "_",
    extract_keys_as_header = FALSE,
    keep_keys_in_header = TRUE,
    ...
)
```

Arguments

`data`	A `data.frame` object.
`keys`	Variables to be used as keys. See the `by` argument in "`divide`".

keep — Variables to remain as their own columns. If NULL (default), all variables except those in send are chosen, excluding the keys. Has tidyselect::vars_select capabilities.

send — Variables to send into new columns for each key. If NULL (default), all variables except those in keep are chosen, excluding the keys. Has tidyselect::vars_select capabilities.

join — Function that joins on keep across the keys. Defaults to dplyr::full_join. See ?dplyr::join for choices.

.sep — Character to separate and identify keys values over the columns. Defaults to "_".

extract_keys_as_header — Should the keys labels be returned as a separate character vector? Defaults to FALSE. Has no effect when there is only 1 send column.

keep_keys_in_header — If extract_keys_as_header = TRUE, should the keys be left in the result columns? Defaults to TRUE. Useful to set to FALSE if a call to knitr::kable follows.

... — Additional arguments to be passed to join.

Value

A wide data.frame with variables spread over the columns. If extract_keys_as_header = TRUE, the result is a two-element list with the transformed data in 1 element and the top-level header in the other. If only 1 send variable exists, the original names are removed.

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#Make data frame with multiple summary columns
temp_summary <-

    heart_disease %>%
    group_by(
        Sex,
        HeartDisease,
        BloodSugar
    ) %>%
    summarise(
        Mean = mean(Age, na.rm = TRUE),
        SD = sd(Age, na.rm = TRUE),
        Median = median(Age, na.rm = TRUE)
    ) %>%
    ungroup()
```

```
#1) Span summaries for each combination of Sex and BloodSugar
temp_summary %>%
    stretch(
        keys = c("Sex", "BloodSugar"),
        keep = "HeartDisease"
    )

#2) If "HeartDisease" wasn't fully crossed, use different joining to get only matching groups
temp_summary %>%
    stretch(
        keys = c("Sex", "BloodSugar"),
        keep = "HeartDisease",
        join = inner_join
    )

#3) Only send two of the summaries
temp_summary %>%
    stretch(
        keys = c("Sex", "BloodSugar"),
        keep = "HeartDisease",
        send = c("Mean", "Median")
    )

#4) Clean HTML table with keys spanned over columns
result <-
    temp_summary %>%
    stretch(
        keys = c("Sex", "BloodSugar"),
        keep = "HeartDisease",
        extract_keys_as_header = TRUE,
        keep_keys_in_header = FALSE
    )
result$.result %>%
    knitr::kable(format = "html") %>%
    kableExtra::kable_styling() %>%
    kableExtra::add_header_above(
        kableExtra::auto_index(result$.header)
    )
```

`type_match` *Check if an object is at least one (or none) of the specified types*

Description

This is a utility function to assess whether an object inherits at least one (or none) of the user-specified classes.

Usage

```
type_match(
    object,
    types,
    negated = FALSE
)
```

Arguments

object	Any R object.
types	A character vector of classes to check against object.
negated	Should TRUE be returned if object inherits no types? Defaults to FALSE.

Value

A logical indicator

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#1) Default
heart_disease %>%
    map_lgl(
        type_match,
        types = "numeric"
    )

#2) Use negation
heart_disease %>%
    map_lgl(
        type_match,
        types = "numeric",
        negated = TRUE
    )

#3) Multiple types
heart_disease %>%
    map_lgl(
        type_match,
        types = c("numeric", "factor")
    )

#4) Check other objects
heart_disease %>%
    glm(Sex ~ Age, data = ., family = "binomial") %>%
    type_match(
```

```
    types = "glm"
)
```

---

typly *Apply function(s) to elements conforming to specified type(s)*

Description

Evaluates a function or a list of function on all elements of `list` or `data.frame` that inherit at least one of the allowable types specified by the user. An option is available to evaluate the function(s) on all elements that do not match.

Usage

```
typly(
    data,
    types,
    f,
    negated = FALSE,
    keep = FALSE,
    ...
)
```

Arguments

`data`	A `data.frame` or `list`.
`types`	A character vector of allowable data types identifying columns in which function(s) should be applied.
`f`	A `function` or `list` of `functions`.
`negated`	Should the function(s) be applied to columns that don't match any `types`? Defaults to `FALSE`.
`keep`	Should the non-matching columns be kept as is? Defaults to `FALSE`.
`...`	Additional arguments to be passed to `f`.

Value

A `list` with the result(s) of `f` for each applicable column.

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

heart_disease %>%

    #Compute means and medians on numeric data
    typly(
        c("numeric", "logical"),
        list(
            mean = mean,
            median = median
        ),
        keep = TRUE,
        na.rm = TRUE
    ) %>%

    #Compute table
    typly(
        "factor",
        table,
        keep = TRUE
    )
```

`univariate_associations` *Compute association metrics between any number of variables*

Description

Evaluates a list of scalar functions on any number of "responses" for each individual variable and gathers results.

Usage

```
univariate_associations(
    data,
    f,
    responses = NULL,
    predictors = NULL
)
```

Arguments

`data`	Any `data.frame`.
`f`	A `function` or a `list` of functions (prefereably named) that take a vector as input in the first two arguments and return an `atomic` scalar.
`responses`	Response or outcome variables. See "`dish`".
`predictors`	Predictors or covariates. See "`dish`".

Value

A `tibble` with the variables in the rows and the results of the functions in the columns. The names of the columns will be the names provided in `f`.

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#Make a list of functions
f <-
    list(

        #Compute a univariate p-value
        `P-value` =

            function(x, y) {

                if(type_match(y, c("factor", "character"))) {

                  p <- fisher.test(factor(x), factor(y), simulate.p.value = TRUE)$p.value


                } else {

                    p <- kruskal.test(y, factor(x))$p.value

                }
                if_else(
                    p < 0.001, "<0.001", as.character(round(p, 2))
                )
            },

        #Compute difference in AIC model between null model and one predictor model
        `AIC Difference` =
            function(x, y) {

                glm(factor(x)~1, family = "binomial")$aic -
                glm(factor(x)~y, family = "binomial")$aic

            }
    )

#1) Apply functions to Sex/HeartDisease by all other variables
heart_disease %>%
    univariate_associations(
        f = f,
        responses = c("Sex", "HeartDisease")
    )
```

```
#2) Only use two variables on RHS
heart_disease %>%
    univariate_associations(
        f = f,
        responses = c("Sex", "HeartDisease"),
        predictors = c("Age", "ChestPain")
    )

#3) Use all combinations
heart_disease %>%
    univariate_associations(
        f = f
    )
```

`univariate_table` *Create a custom univariate summary for a dataset*

Description

Produces a formatted table of univariate summary statistics with options allowing for stratification by 1 or more variables, computing of custom summary/association statistics, custom string templates for results, etc.

Usage

```
univariate_table(
    data,
    strata = NULL,
    associations = NULL,
    numeric_summary = c(Summary = "median (iqr)"),
    categorical_summary = c(Summary = "count (percent%)"),
    other_summary = c(Summary = "unique"),
    all_summary = NULL,
    evaluate = FALSE,
    add_n = FALSE,
    order = NULL,
    labels = NULL,
    levels = NULL,
    format = c("html", "latex", "markdown", "pandoc", "none"),
    variableName = "Variable",
    levelName = "Level",
    na_string = "(missing)",
    strata_sep = "/",
    summary_strata_sep = "_",
    fill_blanks = "",
    caption = "",
```

```
    ...
)
```

Arguments

data	A data.frame to summarise.
strata	A formula specifying one or more stratification variables. LHS variables go to rows, RHS variables go to columns. Defaults to NULL.
associations	A named list of functions to evaluate with column strata and each variable. Defaults to NULL.
numeric_summary	A (preferably named) character vector containing string templates of how results for numeric data should be presented. See details for a list of what is available by default. Defaults to c(Summary = "median (iqr)").
categorical_summary	A (preferably named) character vector containing string templates of how results for categorical data should be presented. See details for a list of what is available by default. Defaults to c(Summary = "count (percent%)").
other_summary	A (preferably named) character vector containing string templates of how results for non-numeric and non-categorical data should be presented. See details for a list of what is available by default. Defaults to c(Summary = "unique").
all_summary	A (preferably named) character vector containing string templates of additional results for all variables should be presented. See details for a list of what is available by default. Defaults to NULL.
evaluate	Should the results of the string templates be evaluated as an R expression after filled with their values? See "absorb" for details. Defaults to FALSE.
add_n	Should the sample size for each stratfication level be added to the result? Defaults to FALSE.
order	Character vector of 1 or more variables to reorder the result by from top to bottom. If NULL (default), the result is sorted according to names(data).
labels	Named character vector for re-labeling variables in the result. Defaults to NULL.
levels	Named list of character vectors for re-labeling factor levels in the result. Defaults to NULL.
format	The format that the result should be rendered as. Must be one of c("html", "latex", "markdown", "pa Defaults to "html".
variableName	Header for the variable column in the result. Defaults to "Variable".
levelName	Header for the factor level column in the result. Defaults to "Level".
na_string	String for NA factor levels in the result. Defaults to "(missing)".
strata_sep	Delimiter to separate stratification levels by in the result. Defaults to "/".
summary_strata_sep	Delimiter to separate summary column names with the strata groups. Defaults to "_".
fill_blanks	String to fill in blank spaces in the result. Defaults to "".
caption	Caption for resulting table passed to knitr::kable. Defaults to NULL.
...	Additional arguments to pass to "descriptives".

Details

The following statistics are available by default for each data type:

Numeric: "min", "max", "median", "iqr", "mean", "sd"

Categorical: "count", "percent"

All variables: "length", "missing", "available", "class", "unique"

These strings are typed explicitly in the `._summary` arguments and serve as placeholders for where the actual value will appear. Custom functions can be entered in a named `list`, where the names are what provide access to the values in string templates. See `"descriptives"` and `"absorb"`.

The names of the `._summary` arguments are what become the column headers in the result. If unnamed, an arbitrary name (i.e. "VX") will appear in the column header.

Value

A table of summary statistics according to the specified `format`. A `tibble` is returned if `format = "none"`.

Author(s)

Alex Zajichek

Examples

```
require(tidyverse)

#1) Default summary
heart_disease %>%
    univariate_table()

#2) Stratified summary
heart_disease %>%
    univariate_table(
        strata = ~Sex,
        add_n = TRUE
    )

#See vignette("cheese") for more examples
```

Index