Package 'cheese'

July 22, 2025

Version 0.1.2

Date 2023-01-04

Title Tools for Working with Data During Statistical Analysis

Description Contains tools for working with data during statistical analysis, promoting flexible, intuitive, and reproducible workflows. There are functions designated for specific statistical tasks such building a custom univariate descriptive table, computing pairwise association statistics, etc. These are built on a collection of data manipulation tools designed for general use that are motivated by the functional programming concept.

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

License MIT + file LICENSE

Depends R (>= 3.4.0)

Imports dplyr (>= 0.8.2), forcats (>= 0.3.0), kableExtra (>= 1.0.1), knitr (>= 1.20), magrittr (>= 1.5), methods (>= 3.4.1), purrr (>= 0.3.2), rlang (>= 0.4.3), stringr (>= 1.3.1), tibble (>= 2.1.3), tidyr (>= 0.8.1), tidyselect (>= 1.0.0)

Suggests rmarkdown (>= 1.10)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

NeedsCompilation no

Author Alex Zajichek [aut, cre]

Maintainer Alex Zajichek <alexzajichek@gmail.com>

Repository CRAN

Date/Publication 2023-01-06 19:40:02 UTC

Contents

`absorb` *Absorb values into a string containing keys*

Description

Populate string templates containing keys with their values. The keys are interpreted as regular expressions. Results can optionally be evaluated as R expressions.

Usage

```
absorb(
    key,
    value,
    text,
    sep = "_",
    trace = FALSE,
    evaluate = FALSE
)
```

Arguments

`key`	A vector that can be coerced to type `character`.
`value`	A vector with the same length as `key`.
`text`	A (optionally named) `character` vector containing patterns.
`sep`	Delimiter to separate values by in the placeholder for duplicate patterns. Defaults to `"_"`
`trace`	Should the recursion results be printed to the console each iteration? Defaults to `FALSE`.
`evaluate`	Should the result(s) be evaluated as R expressions? Defaults to `FALSE`.

Details

The inputs are iterated in sequential order to replace each pattern with its corresponding value. It is possible that a subsequent pattern could match with a prior result, 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.
- Otherwise, a `list` with the same length as `text`.

Author(s)

Alex Zajichek

Examples

```
#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"
        )
)

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

---

depths *Find the elements in a list structure that satisfy a predicate*

Description

Traverse a list of structure to find the depths and positions of its elements that satisfy a predicate.

Usage

```
depths(
    list,
    predicate,
    bare = TRUE,
    ...
)
depths_string(
    list,
    predicate,
    bare = TRUE,
    ...
)
```

Arguments

list	A list, data.frame, or vector.
predicate	A function that evaluates to TRUE or FALSE.
bare	Should algorithm only continue for bare lists? Defaults to TRUE. See rlang::`bare-type-predicates`
...	Additional arguments to pass to predicate.

Details

The input is recursively evaluated to find elements that satisfy predicate, and only proceeds where rlang::is_list when argument bare is FALSE, and rlang::is_bare_list when it is TRUE.

Value

- depths() returns an integer vector indicating the levels that contain elements satisfying the predicate.
- depths_string() returns a character representation of the traversal. Brackets {} are used to indicate the level of the tree, commas to separate element-indices within a level, and the sign of the index to indicate whether the element satisfied predicate (- = yes, + = no).

Author(s)

Alex Zajichek

Examples

```
#Find depths of data frames
df1 <-
  heart_disease %>%

    #Divide the frame into a list
    divide(
      Sex,
      HeartDisease,
      ChestPain
```

```
    )

df1 %>%

  #Get depths as an integer
  depths(
    predicate = is.data.frame
  )

df1 %>%

  #Get full structure
  depths_string(
    predicate = is.data.frame
  )

#Shallower list
df2 <-
  heart_disease %>%
    divide(
      Sex,
      HeartDisease,
      ChestPain,
      depth = 1
    )

df2 %>%
  depths(
    predicate = is.data.frame
  )

df2 %>%
  depths_string(
    predicate = is.data.frame
  )

#Allow for non-bare lists to be traversed
df1 %>%
  depths(
    predicate = is.factor,
    bare = FALSE
  )

#Make uneven list with diverse objects
my_list <-
  list(
    heart_disease,
    list(
      heart_disease
    ),
    1:10,
    list(
      heart_disease$Age,
```

```
      list(
        heart_disease
      )
    ),
    glm(
      formula = HeartDisease ~ .,
      data = heart_disease,
      family = "binomial"
    )
  )

#Find the data frames
my_list %>%
  depths(
    predicate = is.data.frame
  )

my_list %>%
  depths_string(
    predicate = is.data.frame
  )

#Go deeper by relaxing bare list argument
my_list %>%
  depths_string(
    predicate = is.data.frame,
    bare = FALSE
  )
```

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

Description

The user can specify an unlimited number of functions to evaluate and the types of data that each set of functions will be applied to (including the default; see "Details").

Usage

```
descriptives(
    data,
    f_all = NULL,
    f_numeric = NULL,
    numeric_types = "numeric",
    f_categorical = NULL,
    categorical_types = "factor",
    f_other = NULL,
    useNA = c("ifany", "no", "always"),
```

```
    round = 2,
    na_string = "(missing)"
)
```

Arguments

data	A data.frame.
f_all	A list of functions to evaluate on all columns.
f_numeric	A list of functions to evaluate on numeric_types columns.
numeric_types	Character vector of data types that should be evaluated by f_numeric.
f_categorical	A list of functions to evaluate on categorical_types columns.
categorical_types	Character vector of data types that should be evaluated by f_categorical.
f_other	A list of functions to evaluate on remaining columns.
useNA	See table for details. Defaults to "ifany".
round	Digit to round numeric data. Defaults to 2.
na_string	String to fill in NA names.

Details

The following fun_key's are available by default for the specified types:

- ALL: length, missing, available, class, unique
- Numeric: mean, sd, min, q1, median, q3, max, iqr, range
- Categorical: count, proportion, percent

Value

A tibble::tibble with the following columns:

- fun_eval: Column types function was applied to
- fun_key: Name of function that was evaluated
- col_ind: Index from input dataset
- col_lab: Label of the column
- val_ind: Index of the value within the function result
- val_lab: Label extracted from the result with names
- val_dbl: Numeric result
- val_chr: Non-numeric result
- val_cbn: Combination of (rounded) numeric and non-numeric values

Author(s)

Alex Zajichek

Examples

```
#Default
heart_disease %>%
    descriptives()

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

    #Extract info from the column
    dplyr::filter(
        col_lab == "BloodSugar"
    )

#Nothing treated as numeric
heart_disease %>%
    descriptives(
        numeric_types = NULL
    )

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

    #Extract info from the custom function
    dplyr::filter(
        fun_key == "cv"
    )
```

`dish` *Evaluate a two-argument function with combinations of columns*

Description

Split up columns into groups and apply a function to combinations of those columns with control over whether each group is entered as a single `data.frame` or individual `vector`'s.

Usage

```
dish(
    data,
```

```
    f,
    left,
    right,
    each_left = TRUE,
    each_right = TRUE,
    ...
)
```

Arguments

data	A data.frame.
f	A function that takes a vector and/or data.frame in the first two arguments.
left	A vector of quoted/unquoted columns, positions, and/or tidyselect::select_helpers to be evaluated in the first argument of f.
right	A vector of quoted/unquoted columns, positions, and/or tidyselect::select_helpers to be evaluated in the second argument of f.
each_left	Should each left variable be indivdually evaluated in f? Defaults to TRUE. If FALSE, left columns are entered into f as a single data.frame.
each_right	Should each right variable be individually evaluated in f? Defaults to TRUE. If FALSE, right columns are entered into f as a single data.frame.
...	Additional arguments to be passed to f.

Value

A list

Author(s)

Alex Zajichek

Examples

```
#All variables on both sides
heart_disease %>%
    dplyr::select(
        where(is.numeric)
    ) %>%
    dish(
        f = cor
    )

#Simple regression of each numeric variable on each other variable
heart_disease %>%
    dish(
        f =
            function(y, x) {
                mod <- lm(y ~ x)
                tibble::tibble(
                    Parameter = names(mod$coef),
```

```
                    Estimate = mod$coef
                )
            },
        left = where(is.numeric)
    ) %>%

    #Bind rows together
    fasten(
        into = c("Outcome", "Predictor")
    )

#Multiple regression of each numeric variable on all others simultaneously
heart_disease %>%
    dish(
        f =
            function(y, x) {
                mod <- lm(y ~ ., data = x)
                tibble::tibble(
                    Parameter = names(mod$coef),
                    Estimate = mod$coef
                )
            },
        left = where(is.numeric),
        each_right = FALSE
    ) %>%

    #Bind rows together
    fasten(
        into = "Outcome"
    )
```

`divide` *Divide a data frame into a list*

Description

Separate a `data.frame` into a `list` of any depth by one or more stratification columns whose levels become the names.

Usage

```
divide(
    data,
    ...,
    depth = Inf,
    remove = TRUE,
    drop = TRUE,
    sep = "|"
)
```

Arguments

data	Any data.frame.
...	Selection of columns to split by. See dplyr::select for details.
depth	Depth to split to. Defaults to Inf. See details for more information.
remove	Should the stratfication columns be removed? Defaults to TRUE.
drop	Should unused combinations of stratification variables be dropped? Defaults to TRUE.
sep	String to separate values of each stratification variable by. Defaults to "|". Only used when the number of stratification columns exceeds the desired depth.

Details

For the depth, use positive integers to move from the root and negative integers to move from the leaves. The maximum (minimum) depth will be used for integers larger (smaller) than such.

Value

A list

Author(s)

Alex Zajichek

Examples

```
#Unquoted selection
heart_disease %>%
    divide(
        Sex
    )

#Using select helpers
heart_disease %>%
    divide(
        matches("^S")
    )

#Reduced depth
heart_disease %>%
    divide(
        Sex,
        HeartDisease,
        depth = 1
    )

#Keep columns in result; change delimiter in names
heart_disease %>%
    divide(
        Sex,
```

```
        HeartDisease,
        depth = 1,
        remove = FALSE,
        sep = ","
    )

#Move inward from maximum depth
heart_disease %>%
    divide(
        Sex,
        HeartDisease,
        ChestPain,
        depth = -1
    )

#No depth returns original data (and warning)
heart_disease %>%
    divide(
        Sex,
        depth = 0
    )
heart_disease %>%
    divide(
        Sex,
        HeartDisease,
        depth = -5
    )

#Larger than maximum depth returns maximum depth (default)
heart_disease %>%
    divide(
        Sex,
        depth = 100
    )
```

`fasten` *Bind a list of data frames back together*

Description

Roll up a `list` of arbitrary depth with `data.frame`'s at the leaves row-wise.

Usage

```
fasten(
    list,
    into = NULL,
    depth = 0
)
```

Arguments

list	A list with data.frame's at the leaves.
into	A character vector of resulting column names. Defaults to NULL.
depth	Depth to bind the list to. Defaults to 0.

Details

Use empty strings "" in the into argument to omit column creation when rows are binded. Use positive integers for the depth to move from the root and negative integers to move from the leaves. The maximum (minimum) depth will be used for integers larger (smaller) than such. The leaves of the input list should be at the same depth.

Value

A tibble::tibble or reduced list

Author(s)

Alex Zajichek

Examples

```
#Make a divided data frame
list <-
  heart_disease %>%
  divide(
    Sex,
    HeartDisease,
    ChestPain
  )

#Bind without creating names
list %>%
  fasten

#Bind with names
list %>%
  fasten(
    into = c("Sex", "HeartDisease", "ChestPain")
  )

#Only retain "Sex"
list %>%
  fasten(
    into = "Sex"
  )

#Only retain "HeartDisease"
list %>%
  fasten(
    into = c("", "HeartDisease")
```

```
  )

#Bind up to Sex
list %>%
  fasten(
    into = c("HeartDisease", "ChestPain"),
    depth = 1
  )

#Same thing, but start at the leaves
list %>%
  fasten(
    into = c("HeartDisease", "ChestPain"),
    depth = -2
  )

#Too large of depth returns original list
list %>%
  fasten(
    depth = 100
  )

#Too small of depth goes to 0
list %>%
  fasten(
    depth = -100
  )
```

---

grable *Make a* kable *with a hierarchical header*

Description

Create a `knitr::kable` with a multi-layered (graded) header.

Usage

```
grable(
    data,
    at,
    sep = "_",
    reverse = FALSE,
    format = c("html", "latex"),
    caption = NULL,
    ...
)
```

Arguments

data	A data.frame.
at	A vector of quoted/unquoted columns, positions, and/or tidyselect::select_helpers. Defaults to all columns.
sep	String to separate the columns. Defaults to "_".
reverse	Should the layers be added in the opposite direction? Defaults to FALSE.
format	Format for rendering the table. Must be "html" (default) or "latex".
caption	Optional caption for the table
...	Arguments to pass to kableExtra::kable_styling

Value

A knitr::kable

Author(s)

Alex Zajichek

---

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

muddle *Randomly permute some or all columns of a data frame*

Description

Shuffle any of the columns of a `data.frame` to artificially distort relationships.

Usage

```
muddle(
    data,
    at,
    ...
)
```

Arguments

`data`	A `data.frame`.
`at`	A vector of quoted/unquoted columns, positions, and/or `tidyselect::select_helpers`. Defaults to all columns.
`...`	Additional arguments passed to `sample`.

Value

A `tibble::tibble`

Author(s)

Alex Zajichek

Examples

```
#Set a seed
set.seed(123)

#Default permutes all columns
heart_disease %>%
  muddle

#Permute select columns
heart_disease %>%
  muddle(
    at = c(Age, Sex)
  )

#Using a select helper
heart_disease %>%
  muddle(
```

```
    at = matches("^S")
  )

#Pass other arguments
heart_disease %>%
  muddle(
    size = 5,
    replace = TRUE
  )
```

---

some_type *Is an object one of the specified types?*

Description

Check if an object inherits one (or more) of a vector classes.

Usage

```
some_type(
    object,
    types
)
```

Arguments

object	Any R object.
types	A character vector of classes to test against.

Value

A logical indicator

Author(s)

Alex Zajichek

Examples

```
#Columns of a data frame
heart_disease %>%
    purrr::map_lgl(
        some_type,
        types = c("numeric", "logical")
    )
```

stratiply *Stratify a data frame and apply a function*

Description

Split a data.frame by any number of columns and apply a function to subset.

Usage

```
stratiply(
    data,
    f,
    by,
    ...
)
```

Arguments

data	A data.frame.
f	A function that takes a data.frame as an argument.
by	A vector of quoted/unquoted columns, positions, and/or tidyselect::select_helpers
...	Additional arguments passed to f.

Value

A list

Author(s)

Alex Zajichek

Examples

```
#Unquoted selection
heart_disease %>%
    stratiply(
        head,
        Sex
    )

#Select helper
heart_disease %>%
    stratiply(
        f = head,
        by = starts_with("S")
    )

#Use additional arguments for the function
```

```
heart_disease %>%
  stratiply(
        f = glm,
        by = Sex,
        formula = HeartDisease ~ .,
        family = "binomial"
  )

#Use mixed selections to split by desired columns
heart_disease %>%
  stratiply(
        f = glm,
        by = c(Sex, where(is.logical)),
        formula = HeartDisease ~ Age,
        family = "binomial"
  )
```

`stretch` *Span keys and values across the columns*

Description

Pivot one or more values across the columns by one or more keys

Usage

```
stretch(
    data,
    key,
    value,
    sep = "_"
)
```

Arguments

`data`	A `data.frame`.
`key`	A vector of quoted/unquoted columns, positions, and/or `tidyselect::select_helpers` whose values will become the column name(s).
`value`	A vector of quoted/unquoted columns, positions, and/or `tidyselect::select_helpers` whose values will be spread across the columns.
`sep`	String to separate keys/values by in the resulting column names. Defaults to `"_"`. Only used when there are more than one keys/values.

Details

In the case of multiple `value`'s, the labels are always appended to the end of the resulting columns.

Value

A `tibble::tibble`

Author(s)

Alex Zajichek

Examples

```
#Make a summary table
set.seed(123)
data <-
  heart_disease %>%
  dplyr::group_by(
    Sex,
    BloodSugar,
    HeartDisease
  ) %>%
  dplyr::summarise(
    Mean = mean(Age),
    SD = sd(Age),
    .groups = "drop"
  ) %>%
  dplyr::mutate(
    Random =
      rbinom(nrow(.), size = 1, prob = .5) %>%
      factor
  )

data %>%
  stretch(
    key = c(BloodSugar, HeartDisease),
    value = c(Mean, SD, Random)
  )

data %>%
  stretch(
    key = where(is.factor),
    value = where(is.numeric)
  )

data %>%
  stretch(
    key = c(where(is.factor), where(is.logical)),
    value = where(is.numeric)
  )
```

typly *Evaluate a function on columns conforming to one or more (or no) specified types*

Description

Apply a function to columns in a data.frame that inherit one of the specified types.

Usage

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

Arguments

data	A data.frame.
f	A function.
types	A character vector of classes to test against.
negated	Should the function be applied to columns that don't match any types? Defaults to FALSE.
...	Additional arguments to be passed to f.

Value

A list

Author(s)

Alex Zajichek

Examples

```
heart_disease %>%

    #Compute means on numeric or logical data
    typly(
        f = mean,
        types = c("numeric", "logical"),
        na.rm = TRUE
    )
```

`univariate_associations` *Compute association statistics between columns of a data frame*

Description

Evaluate a `list` of scalar functions on any number of "response" columns by any number of "predictor" columns

Usage

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

Arguments

`data`	A `data.frame`.
`f`	A function or a `list` of functions (preferably named) that take a vector as input in the first two arguments and return a scalar.
`responses`	A vector of quoted/unquoted columns, positions, and/or `tidyselect::select_helpers` to be evaluated as the first argument. See the `left` argument in `dish`.
`predictors`	A vector of quoted/unquoted columns, positions, and/or `tidyselect::select_helpers` to be evaluated as the second argument. See the `right` argument in `dish`.

Value

A `tibble::tibble` with the response/predictor columns down the rows and the results of the `f` across the columns. The names of the result columns will be the names provided in `f`.

Author(s)

Alex Zajichek

Examples

```
#Make a list of functions to evaluate
f <-
  list(

    #Compute a univariate p-value
    `P-value` =
      function(y, x) {
        if(some_type(x, c("factor", "character"))) {
```

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

          } else {

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

          }

          ifelse(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(y, x) {

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

        }
    )

  #Choose a couple binary outcomes
  heart_disease %>%
    univariate_associations(
      f = f,
      responses = c(ExerciseInducedAngina, HeartDisease)
    )

  #Use a subset of predictors
  heart_disease %>%
    univariate_associations(
      f = f,
      responses = c(ExerciseInducedAngina, HeartDisease),
      predictors = c(Age, BP)
    )

  #Numeric predictors only
  heart_disease %>%
    univariate_associations(
      f = f,
      responses = c(ExerciseInducedAngina, HeartDisease),
      predictors = is.numeric
    )
```

`univariate_table` *Create a custom descriptive table for a dataset*

Description

Produces a formatted table of univariate summary statistics with options allowing for stratification by one 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 (q1, q3)"),
    categorical_summary = c(Summary = "count (percent%)"),
    other_summary = NULL,
    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",
    sep = "_",
    fill_blanks = "",
    caption = NULL,
    ...
)
```

Arguments

data	A data.frame.
strata	An additive formula specifying stratification columns. Columns on the left side go down the rows, and columns on the right side go across the columns. Defaults to NULL.
associations	A named list of functions to evaluate with column strata and each variable. Defaults to NULL. See univariate_associations.
numeric_summary	A named vector containing string templates of how results for numeric data should be presented. See details for what is available by default. Defaults to c(Summary = "median (q1, q3)").
categorical_summary	A named vector containing string templates of how results for categorical data should be presented. See details for what is available by default. Defaults to c(Summary = "count (percent%)").
other_summary	A named character vector containing string templates of how results for non-numeric and non-categorical data should be presented. Defaults to NULL.

all_summary	A named character vector containing string templates of additional results applying to all variables. See details for 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	Arguments passed to forcats::fct_relevel for reordering the variables. Defaults to NULL
labels	A named character vector containing the new labels. Defaults to NULL
levels	A named list of named character vectors containing the new levels. Defaults to NULL
format	The format that the result should be rendered in. Must be "html", "latex", "markdown", "pandoc", or "none". 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".
sep	Delimiter to separate summary columns. 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.

Value

A table of summary statistics in the specified format. A tibble::tibble is returned if format = "none".

Author(s)

Alex Zajichek

Examples

```
#Set format
format <- "pandoc"

#Default summary
heart_disease %>%
    univariate_table(
      format = format
    )

#Stratified summary
heart_disease %>%
    univariate_table(
        strata = ~Sex,
        add_n = TRUE,
        format = format
```

```
    )

#Row strata with custom summaries with
heart_disease %>%
    univariate_table(
        strata = HeartDisease~1,
        numeric_summary = c(Mean = "mean", Median = "median"),
        categorical_summary = c(`Count (%)` = "count (percent%)"),
        categorical_types = c("factor", "logical"),
        add_n = TRUE,
        format = format
    )
```

Index