Package 'mason'

June 4, 2020

Type Package

Title Build Data Structures for Common Statistical Analysis

Version 0.3.0

Description Use a consistent syntax to create data structures of common statistical techniques that can be continued in a pipe chain. Design the analysis, add settings and variables, construct the results, and polish the final structure. Rinse and repeat for any number of statistical techniques.

License MIT + file LICENSE

URL https://github.com/lwjohnst86/mason

BugReports https://github.com/lwjohnst86/mason/issues

Depends R (>= 3.6.0)

Imports broom, dplyr, magrittr, purrr, rlang, stats, tibble, tidyr, tidyselect, utils

Suggests covr, geepack, knitr, MASS, pls, rmarkdown, roxygen2, testthat

VignetteBuilder knitr

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.1.0

NeedsCompilation no

Author Luke Johnston [aut, cre] (<https://orcid.org/0000-0003-4169-2616>)

Maintainer Luke Johnston <lwjohnst@gmail.com>

Repository CRAN

Date/Publication 2020-06-04 16:10:05 UTC

R topics documented:

`add_settings` *Add analysis settings to the blueprint*

Description

Most statistical techniques need to specify some settings for them to run. This function sets those settings in the blueprint, before the statistical method is used at the construction phase.

Usage

```
add_settings(data, ...)

## S3 method for class 'gee_bp'
add_settings(
  data,
  cluster.id,
  family,
  corstr = c("independence", "exchangeable", "ar1"),
  conf.int = TRUE,
  conf.level = 0.95,
  ...
)

## S3 method for class 'cor_bp'
add_settings(
  data,
  method = c("pearson", "kendall", "spearman"),
  use = c("complete.obs", "all.obs", "pairwise.complete.obs", "everything",
    "na.or.complete"),
  hclust.order = FALSE,
  ...
)

## S3 method for class 'glm_bp'
add_settings(data, family, conf.int = TRUE, conf.level = 0.95, ...)
```

```
## S3 method for class 'pls_bp'
add_settings(
  data,
  ncomp = NULL,
  scale = TRUE,
  validation = c("none", "CV", "LOO"),
  cv.data = TRUE,
  cv.seed = 1234,
  ...
)

## S3 method for class 't.test_bp'
add_settings(data, paired = FALSE, ...)
```

Arguments

data	The blueprint data object.
...	Additional args.
cluster.id	Variable that represents the cluster for GEE.
family	a description of the error distribution and link function to be used in the model. For glm this can be a character string naming a family function, a family function or the result of a call to a family function. For glm.fit only the third option is supported. (See family for details of family functions.)
corstr	The correlation structure. See geepack::geeglm().
conf.int	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to FALSE.
conf.level	The confidence level to use for the confidence interval if conf.int = TRUE. Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
method	the method to be used in fitting the model. The default method "glm.fit" uses iteratively reweighted least squares (IWLS): the alternative "model.frame" returns the model frame and does no fitting. User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as glm.fit. If specified as a character string it is looked up from within the **stats** namespace.
use	an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".
hclust.order	Whether to order the correlation data based on the stats::hclust() algorithm.
ncomp	the number of components to include in the model (see below).
scale	numeric vector, or logical. If numeric vector, X is scaled by dividing each variable with the corresponding element of scale. If scale is TRUE, X is scaled by dividing each variable by its sample standard deviation. If cross-validation is selected, scaling by the standard deviation is done for every segment.

validation	character. What kind of (internal) validation to use. See below.
cv.data	Whether to cross-validate the dataset into training and testing sets.
cv.seed	Seed to set for cv.data.
paired	a logical indicating whether you want a paired t-test.

Value

Settings for the analysis are added to the blueprint

Examples

```
## Not run:
design(iris, 'gee') %>%
 add_settings('Species', family = binomial('logit'), conf.int = FALSE)

ds <- design(iris, 'cor')
ds <- add_settings(ds, method = 'spearman')

ds <- design(iris, 't.test')
add_settings(ds, paired = TRUE)
add_settings(ds)

## End(Not run)
```

---

add_variables *Add variables to the analysis*

Description

While different analyses use different types of variables, in general they can be classified as in the 'y' or 'x' position of a statistical equation. They can further be classified as covariates and as an interaction term.

Usage

```
add_variables(
  data,
  type = c("yvars", "xvars", "covariates", "interaction"),
  variables
)
```

Arguments

data	The blueprint data object.
type	The variable type, i.e. where it is located on the equation (y position, x, as a covariate, etc.)
variables	Variables to use for the type specified

Value

Adds variables to the blueprint

Examples

```
library(magrittr)
ds <- design(iris, 'cor') %>%
 add_settings()
add_variables(ds, 'xvar', 'Sepal.Length')
add_variables(ds, 'yvar', 'Petal.Length')

ds <- design(iris, 't.test')
ds <- add_variables(ds, 'yvar', c('Sepal.Length', 'Sepal.Width'))
ds <- add_variables(ds, 'xvar', 'Petal.Length')
```

---

construct *Construct the results of the analysis*

Description

Construct the results of the analysis

Usage

```
construct(data, ..., na.rm = TRUE)
```

Arguments

data	The blueprint data object.
...	Additional args.
na.rm	Whether to remove missing values.

Value

Uses the blueprint to construct the results of the statistical analysis. Outputs a tibble.

Examples

```
design(iris, 'cor') %>%
 add_settings() %>%
 add_variables('xvars', c('Sepal.Length', 'Sepal.Width')) %>%
 construct()

design(iris, 't.test') %>%
 add_settings() %>%
```

```
 add_variables('yvars', c('Sepal.Length', 'Sepal.Width')) %>%
 add_variables('xvars', c('Petal.Length', 'Petal.Width')) %>%
 construct()

design(iris, 'glm') %>%
 add_settings() %>%
 add_variables('yvars', c('Sepal.Length', 'Sepal.Width')) %>%
 add_variables('xvars', c('Petal.Length', 'Petal.Width')) %>%
 construct()

design(iris, 'gee') %>%
 add_settings('Species') %>%
 add_variables('yvars', c('Sepal.Length', 'Sepal.Width')) %>%
 add_variables('xvars', c('Petal.Length', 'Petal.Width')) %>%
 construct()

design(iris, 'pls') %>%
 add_settings() %>%
 add_variables('yvars', c('Sepal.Length', 'Sepal.Width')) %>%
 add_variables('xvars', c('Petal.Length', 'Petal.Width')) %>%
 construct()
```

`design` *Design the blueprint for an analysis.*

Description

Sets up the initial design (i.e. the blueprint) of a statistical analysis to use on the data. As in creating a building or structure, a blueprint is first needed to guide the construction. This function *only* creates that blueprint, but does not do any construction (e.g. actually running statistics).

Usage

```
design(data, statistic = c("gee", "cor", "glm", "pls", "t.test"))
```

Arguments

`data`	The dataset you want to analyze
`statistic`	The type of statistical test to use

Value

Creates a blueprint object that will be used to construct the analysis in a later phase.

Examples

```
design(iris, 'gee')
design(iris, 'cor')
design(iris, 'glm')
design(iris, 't.test')
```

`mason` *Build a (results from analyses) structure like a mason*

Description

Easily run common statistical analyses and build them into a form that can easily be plotting or made into a table. Many parts of `mason` use `dplyr::dplyr()` functions, which makes the analysis fast and allows it to be put into a `magrittr::magrittr()` pipe chain.

The final, `scrub()`'ed version of the analysis is in a 'tidy' format, meaning it is already in a form to send to ggplot2 or created into a table using the pander package or with `knitr::kable()`. It also allows further processing with `dplyr` and `tidyr`.

Details

One of the main goals of `mason` is to make it easy to implement other analyses in a consistent syntax and structure. Like in architecture, construction, and engineering, data analysis projects follow a similar workflow, where there is a design phase, a construction phase, and a final scrubbing/cleaning/polishing phase, with some back and forth as construction continues. `mason` tries to emulate this pattern.

See Also

For more documentation, see `vignette("mason",package = "mason")`.

`polish` *Do some final polishing of the scrubbed mason analysis data.*

Description

Do some final polishing of the scrubbed mason analysis data.

Usage

```
polish_renaming(data, renaming.fun, columns = NULL)

polish_filter(data, keep.pattern, column)

polish_transform_estimates(data, transform.fun)

polish_adjust_pvalue(data, method = "BH")
```

Arguments

data	The scrubbed object.
renaming.fun	A function, typically with base::gsub(), that searches and replaces strings.
columns	The columns to apply the renaming function to. Defaults to columns that are a factor or character vectors.
keep.pattern	Rows to keep based on a regular expression pattern.
column	The column to apply the filtering to.
transform.fun	A function to modify continuous variable columns.
method	Correction method for the p-value adjustment (stats::p.adjust()).

Functions

- polish_renaming: polish_renaming simply takes a function, most likely one that uses base::gsub(), and uses that to search and replace words, etc, in the specified columns.
- polish_filter: polish_filter is basically a thin wrapper around dplyr::filter(), but using base::grepl() for the pattern searching.
- polish_transform_estimates: polish_transform_estimates is simply a thin wrapper around dplyr::mutate().
- polish_adjust_pvalue: polish_adjust_pvalue is a thin wrapper around dplyr::mutate() and stats::p.adjust()

Examples

```
library(magrittr)
ds <- swiss %>%
 design('glm') %>%
 add_settings() %>%
 add_variables('yvar', c('Fertility', 'Education')) %>%
 add_variables('xvar', c('Agriculture', 'Catholic')) %>%
 add_variables('covariates', 'Examination') %>%
 construct() %>%
 scrub()
polish_renaming(ds, function(x) gsub('Education', 'Schooling', x))
polish_filter(ds, 'Xterm', 'term')
polish_adjust_pvalue(ds)[c('p.value', 'adj.p.value')]
polish_transform_estimates(ds, function(x) exp(x))
```

---

scrub *Scrub down and tidy up the constructed analysis results.*

Description

Scrub down and tidy up the constructed analysis results.

Usage

```
scrub(data, ...)

## S3 method for class 'pls_bp'
scrub(
  data,
  output = c("mvr_object", "default", "scores", "loadings", "score_corr",
    "explained_var"),
  ...
)
```

Arguments

data	The blueprint data object.
...	Other arguments passed to methods.
output	Selecting what to output from model.

Value

Outputs a cleaned up version of the constructed analysis.

See Also

See also tidy_up() for pls tidying.

Examples

```
ds <- design(iris, 'cor')
ds <- add_settings(ds)
ds <- add_variables(ds, 'xvars', c('Sepal.Length', 'Sepal.Width'))
ds <- construct(ds)
scrub(ds)
```

---

tidy_up *Convert model output to tidy tibble/dataframe.*

Description

Currently this only tidies up PLS model objects. The main important output objects from the PLS model are:

Usage

```
tidy_up(model, output, ...)
```

Arguments

model	The model object.
output	Which output to choose from model.
...	Not currently used. For later method additions.

Details

- Scores: These are the individual scores calculated from the model for each observation. Use these to look for patterns between components or between X or Y variables.
- Loadings: These are the combined weights in the model (including both X and Y). Strongly correlated X variables that underlie Y will have similar loadings.
- Explained variance: This is the amount of variance that an individual component explains within X. This is useful to use to see which components to keep.

Value

Tibble object with tidied model output. There are several output options:

- default: Tibble with five columns for the x variables, components, loadings, scores to variable correlations, and explained variance for each component and x variable combination.
- loadings, score_corr: Tibble with three columns for x variables, components, and either loadings or score to variable correlations.
- explained_var: Tibble with two columns for component and it's explained variance.
- scores: Tibble with one column for each component, with values for the scores for each observation.

See Also

See this website for more details on how to interpret the results of PLS.

Examples

```
library(pls)
data(yarn)

NIR <- yarn$NIR
density <- yarn$density
model <- plsr(density ~ NIR)
tidy_up(model)
tidy_up(model, "loadings")
tidy_up(model, "scores")
tidy_up(model, "score_cor")
tidy_up(model, "explained_var")
```

Index