

Package ‘mason’

July 5, 2018

Type Package

Title Build Data Structures for Common Statistical Analysis

Version 0.2.6

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

LazyData TRUE

Depends R (>= 3.4.0)

Imports dplyr, tidyr, broom, magrittr, stats, utils, lazyeval

RoxygenNote 6.0.1

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

VignetteBuilder knitr

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

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

NeedsCompilation no

Author Luke Johnston [aut, cre]

Maintainer Luke Johnston <lwjohnst@gmail.com>

Repository CRAN

Date/Publication 2018-07-05 12:20:02 UTC

R topics documented:

add_settings	2
add_variables	4
construct	5
design	6

mason	6
polish	7
scrub	8

Index	10
--------------	-----------

add_settings	<i>Add analysis settings to the blueprint</i>
--------------	---

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 <code>glm</code> this can be a character string naming a family function, a family function or the result of a call to a family function. For <code>glm.fit</code> only the third option is supported. (See family for details of family functions.)
corstr	The correlation structure. See <code>geepack::geeglm()</code> .
conf.int	whether to include a confidence interval
conf.level	confidence level of the interval, used only if <code>conf.int=TRUE</code>
method	the method to be used in fitting the model. The default method " <code>glm.fit</code> " uses iteratively reweighted least squares (IWLS); the alternative " <code>model.frame</code> " 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 <code>glm.fit</code> . If specified as a character string it is looked up from within the <code>stats</code> 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 " <code>everything</code> ", " <code>all.obs</code> ", " <code>complete.obs</code> ", " <code>na.or.complete</code> ", or " <code>pairwise.complete.obs</code> ".
hclust.order	Whether to order the correlation data based on the <code>stats::hclust()</code> 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 <code>scale</code> . If <code>scale</code> is <code>TRUE</code> , 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 <code>cv.data</code> .
paired	a logical indicating whether you want a paired t-test.

Value

Settings for the analysis are added to the blueprint

Examples

```
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)
```

add_variables	<i>Add variables to the analysis</i>
---------------	--------------------------------------

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	<i>Construct the results of the analysis</i>
-----------	--

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.

Examples

```
library(magrittr)
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	<i>Design the blueprint for an analysis.</i>
--------	--

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", "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	<i>Build a (results from analyses) structure like a mason</i>
-------	---

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::ggplot()` or created into a table using `pander::pander()`, `knitr::kable()`, or `pixiedust::dust()`. 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

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

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_at()`.
- `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 polish up the constructed analysis results.

Description

Scrub down and polish up the constructed analysis results.

Usage

```
scrub(data)
```

Arguments

`data` The blueprint data object.

Value

Outputs a cleaned up version of the constructed analysis.

Examples

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

Index

add_settings, 2
add_variables, 4

base::grepl(), 8
base::gsub(), 7, 8

construct, 5

design, 6
dplyr::dplyr(), 6
dplyr::filter(), 8
dplyr::mutate(), 8
dplyr::mutate_at(), 8

family, 3

geepack::geeglm(), 3
ggplot2::ggplot(), 6

knitr::kable(), 6

magrittr::magrittr(), 6
mason, 6
mason-package (mason), 6

pander::pander(), 6
pixiedust::dust(), 6
polish, 7
polish_adjust_pvalue (polish), 7
polish_filter (polish), 7
polish_renaming (polish), 7
polish_transform_estimates (polish), 7

scrub, 8
scrub(), 6
stats::hclust(), 3
stats::p.adjust(), 7, 8