Package 'rstatsToolkit'

May 11, 2015

Title Bundles up all my most used functions for doing statistical analysis

Version 0.1

Description This package is mainly my personal collection of code that I commonly use in my research and statistical analysis. Eventually I would like to develop it into a toolkit that other graduate students (I'm a graduate student right now btw) and in the future for my own graduate students to use and develop. Until that point, I am slowly developing this package into toolkit for analyzing and exploring data.

Imports gee (>= 4.13.18),
magrittr,
broom,
data.table (>= 1.9.4),
ggplot2 (>= 1.0.0),
grid (>= 3.1.1),
visreg (>= 2.0.5),
reshape2 (>= 1.4),
dplyr (>= 0.4.1)

Depends R (>= 3.1.1)

License Creative Commons 0

LazyData true

R topics documented:

`createCI` *Compute confidence interval for GEE*

Description

Generate confidence intervals (upper and lower) and p-values from a regression (e.g. GEE or GLM).

Usage

```
createCI(data, dist = test.distribution, se = standard.error,
  test.distribution = "Naive.z", standard.error = "Naive.S.E.",
  sig.level = 0.95)
```

Arguments

`data`	The results output from a GEE analysis from the gee package.
`test.distribution,dist`	The column that contains the test statistic distribution, e.g. z-score or t-score.
`standard.error,se`	The column that contains the standard error.
`sig.level`	The significance level for calculating the confidence interval.

Value

Columns with the confidence interval and p-value.

Author(s)

Luke W. Johnston

Examples

```
data(state)

## Very simple test example.  Merely to show how the function is used.
outcomes <- c('Income', 'Population')
exposures <- c('Frost', 'Illiteracy')
covariates <- c('Murder', 'LifeExp')
interaction <- 'LifeExp'
```

```
## This uses the dplyr package.
ds <- cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  rename(LifeExp = `Life Exp`,
         ## Need to rename the id variable to SID (see description
         ## above)
         SID = state.region) %>%
  arrange(SID)

## Without interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    corstr = 'exchangeable') %>%
  extractBetaFromListGEE() %>%
  unlistAndFilterIndep(., exposures) %>%
  createCI()

## With interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    interaction, corstr = 'exchangeable') %>%
  extractBetaFromListGEE() %>%
  unlistAndFilterIndep(., ':', pattern = TRUE) %>%
       ## The ':' represents an interaction
  createCI()
```

`createFormulaList` *Create a list of formulas*

Description

`createFormulaList` returns a list of formulas with all combinations of dependent and independent variables.

Usage

```
createFormulaList(dependent, independent, covariates, interactions = NULL)
```

Arguments

`dependent`	A single or a vector of variables names that the user wishes to use as dependent variables in an analysis; `dependent` must be as a string/character.
`independent`	Similar to `dependent`, except that the variables would be the independent, or exposure, variables of interest.
`covariates`	Confounders or covariates that would be included in **all** formulas and hence **all** models. Also must be a string/character.
`interactions`	Optional, include *one* variable from the `covariates` set that will be assigned as an interaction term with the `independent` variable. Must be a string/character.

Details

This function creates a list of formulas for use in a chain of processes. This function's goal is to create all combinations of a set of dependent, or outcome, variables with a set of independent, or exposure variables. Covariates and an interaction term can also be specified and included into the formula.

Value

Outputs a list of formulas.

Author(s)

Luke W. Johnston

Examples

```
outcomes <- c('Income', 'Education', 'Job')
exposures <- c('Age', 'Sex', 'Height', 'Race', 'IQ')
covariates <- c('ParentEdu', 'Country', 'City')
interactions <- 'City'

createFormulaList(outcomes, exposures, covariates)
createFormulaList(outcomes, exposures, covariates, interactions)
```

---

diagnosticPlots *Regression diagnostic plots and tests*

Description

Generate regression diagnostic plots and tests for linear regression models.

Usage

```
diagnosticPlots(data, y, x, covar)
```

Arguments

data	The dataset with the variables of interest
y	The dependent or outcome variable (that is, the y in the regression equation)
x	The independent, exposure, or predictor variable (that is, the x in the regression equation)
covar	The variables selected as to condition or adjust for the y and x relationship, also known as the confounding variables

Details

This function runs a linear regression on the specified variables and generates diagnostics based on the regression. Basic diagnostics include checking the normality of the residuals, assessing outliers, influence and Cook's D, and multicollinearity. Several tests have been commented out, though they can be uncommented if desired (edit the function to output these if desired). Some of the tests I don't fully understand how to interpret them, but as I learn more I will probably know. This function relies on **MASS** and **gplots**.

Value

Outputs multiple plots and textplots with diagnostic information

Author(s)

Luke Johnston

`extractBetaFromListGEE`

Extract beta coefficients from a GEE list object

Description

Used as part of a chain, `extractBetaFromListGEE` grabs the beta estimate from a list of GEE objects.

Usage

```
extractBetaFromListGEE(data)
```

Arguments

`data`	This is variable that contains the list of GEE objects.

Details

This is used after `loopOutputToListGEE` as part of a chain, preferably using dplyr/magrittr's pipe command (After generating the list of GEE objects, `extractBetaFromListGEE` loops through each GEE object and converts the beta estimate and associated statistics into list of dataframes.

Value

Outputs the beta coefficients.

Author(s)

Luke W. Johnston

Examples

```
data(state)

## Very simple test example.  Merely to show how the function is used.
outcomes <- c('Income', 'Population')
exposures <- c('Frost', 'Illiteracy')
covariates <- c('Murder', 'LifeExp')
interaction <- 'LifeExp'

## This uses the dplyr package.
ds <- cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  rename(LifeExp = `Life Exp`,
         ## Need to rename the id variable to SID (see description
         ## above)
         SID = state.region) %>%
  arrange(SID)

## Without interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    corstr = 'exchangeable') %>%
  extractBetaFromListGEE()

## With interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    interaction, corstr = 'exchangeable') %>%
  extractBetaFromListGEE()
```

`extractBetaGEE` *Extract beta coefficients from a GEE object*

Description

Extract the beta object from a GEE object.

Usage

```
extractBetaGEE(data)
```

Arguments

`data`	The variable that contains the GEE object.

Author(s)

Luke W. Johnston

`loopOutputToListGEE` *Loop through GEE analyses.*

Description

Run a GEE analysis on a list of formulas and create a list of GEE objects.

Usage

```
loopOutputToListGEE(data, dependent, independent, covariates,
  interactions = NULL, corstr = "exchangeable")
```

Arguments

`data`	The dataset with the variables of interest
`dependent`	A single or a vector of variables names that the user wishes to use as dependent variables in an analysis; `dependent` must be as a string/character.
`independent`	Similar to `dependent`, except that the variables would be the independent, or exposure, variables of interest.
`covariates`	Confounders or covariates that would be included in **all** formulas and hence **all** models. Also must be a string/character.
`interactions`	Optional, include *one* variable from the `covariates` set that will be assigned as an interaction term with the `independent` variable. Must be a string/character.
`corstr`	The working correlation structure to use in the `gee` function call. Options can be found in the gee package, but include 'exchangeable', 'unstructured', 'AR-M' (M =

Details

This function is merely a wrapper around `gee`, and therefore all inquiries into GEE should start there. As a note, I don't know how (yet) to provide an option to use a custom `id` variable for the GEE analysis, so **make sure to rename your `id` variable to 'SID'**. Also, according to the GEE documentations, the **ordering of the ID variables matters**! Make sure to sort the ID variables as per how you want them! For documentation on the function to create the formula list, see `createFormulaList`.

Value

Creates a list of GEE objects

Author(s)

Luke W. Johnston

Examples

```
data(state)

## Very simple test example.  Merely to show how the function is used.
outcomes <- c('Income', 'Population')
exposures <- c('Frost', 'Illiteracy')
covariates <- c('Murder', 'LifeExp')
interaction <- 'LifeExp'

## This uses the dplyr package.
ds <- cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  rename(LifeExp = `Life Exp`,
         ## Need to rename the id variable to SID (see description
         ## above)
         SID = state.region) %>%
  arrange(SID)

## Without interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    corstr = 'exchangeable')

## With interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    interaction, corstr = 'exchangeable')
```

`multiPlot` *Multiple plots on page*

Description

Lay out multiple ggplots on one frame or page.

Usage

```
multiPlot(..., plotlist = NULL, file, cols = 1, layout = NULL)
```

Arguments

`...`	Where the ggplot objects are placed to be laid out on the graph grid
`plotlist`	Can be used in place of the ... argument by specifying the ggplot objects as a list object
`file`	Not sure what this is used for
`cols`	Number of columns for the layout. For example cols=2 provides two columns and with four ggplot objects, the resulting output would be a 2 by 2 graphic
`layout`	A matrix that indicates the plot grid layout. For example, if layout = matrix(c(1, 2, 3, 3), nrow = 2, byrow = TRUE) the result would have plot 1 in the upper left, plot 2 in the upper right, and plot 3 would be go across the bottom

Details

This function, which was from http://www.cookbook-r.com/Graphs, is used to lay out several ggplot objects onto one frame or pdf page. For instance, you can have 3 plots on a page, one going vertically across the top, the other two in each corner on the bottom. This function makes up for the difficulty **ggplot2** has with outputting multiple plots on one grid. This function depends on **grid**.

Author(s)

Cookbook R

---

plotBoxWithJitter *Jittered-dot boxplot*

Description

Generates a boxplot of a factor variable on one axis with raw values "jittered" as dots underneath the box.

Usage

```
plotBoxWithJitter(data, x, y, groups = NULL)
```

Arguments

data	The dataset with the variables of interest
x	Variable for the x-axis
y	Variable for the y-axis
groups	Optional, split boxplots and jittered dots by group.

Details

This function is useful for exploring the distribution of a series of variables that share a common unit, such as kilogram. The values for each variable are plotted as jittered dots with a boxplot of the distribution layered on top of the dots. Can add axis labels, themes, etc, through ggplots interface (eg. ylab).

Author(s)

Luke W. Johnston

Examples

```
data(state)
cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  mutate(Region = as.factor(state.region),
         MedianIncome = cut(Income, breaks = c(-Inf, median(Income), Inf),
                            labels = c('Lower', 'Higher'))) %>%
  plotBoxWithJitter(., 'Region', 'Population', groups = 'MedianIncome')
```

plotForest *Forest plot*

Description

Generate a forest plot without the traditional side table .

Usage

```
plotForest(data, coeff = coefficient, yvar = y.variables.column,
  ylab = y.axis.label, xlab = x.axis.label, ci = confid.interval,
  dot.pval = pvalue.factor.column, coefficient = "Estimate",
  y.variables.column = "indep", confid.interval = c("lowerCI", "upperCI"),
  pvalue.factor.column = "NULL", groups = NULL,
  y.axis.label = "Exposures", x.axis.label = "Beta estimate")
```

Arguments

`data`	Dataset for the forest plot.
`coefficient,coeff`	The column that contains the beta estimate/coefficient.
`y.variables.column,yvar`	The column with the exposure variables that will be places on the y-axis of the forest plot.
`confid.interval,ci`	A vector that contains the lower and upper confidence interval.
`pvalue.factor.column,dot.pval`	The column that contains the p-value in the form of a factor variable (ie. with levels such as '>0.05' and '<0.05').
`groups`	The variable to split the plot up.
`y.axis.label,ylab`	The y-axis label.
`x.axis.label,xlab`	The x-axis label.

Details

Create a forest plot, with a dot and 95 without the usual side table that contains the raw data values. If the `dot.pval` argument is supplied, the dots and confidence lines increase in size and opacity as significance increases. If `groups` is also supplied, the forest plot will be split up vertically for each grouping. Thus, a large amount of information on the results can be provided in a fairly small amount of space.

Value

A forest plot

Author(s)

Luke W. Johnston

Examples

```
data(state)

## Very simple test example.  Merely to show how the function is used.
outcomes <- c('Income', 'Population')
exposures <- c('Frost', 'Illiteracy')
covariates <- c('Murder', 'LifeExp')

## This uses the dplyr package.
ds <- cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  rename(LifeExp = `Life Exp`,
         ## Need to rename the id variable to SID (see description
         ## above)
         SID = state.region) %>%
  arrange(SID)

loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    corstr = 'exchangeable') %>%
  extractBetaFromListGEE() %>%
  unlistAndFilterIndep(., exposures) %>%
  createCI() %>%
  plotForest(., dot.pval = 'f.pvalue', groups = 'dep')
```

---

plotHeatmapCorr *Correlation heatmap*

Description

Generate a matrix or non-matrix style heatmap of correlation coefficients.

Usage

```
plotHeatmapCorr(data, x, y = NULL, x.name.sub, y.name.sub,
  heat.colours = c("darkorange2", "skyblue4"), show.corr.values = TRUE,
  ylab = y.axis.label, xlab = x.axis.label, print.corr.values = FALSE,
  y.axis.label = NULL, x.axis.label = NULL)
```

Arguments

data	The dataset to plot.
x	The x axis variables.
y	The y axis variables.

`x.name.sub` The substitutions to the x axis names, incase the original variables need to be clarified (eg. 'Wgt' to 'Weight'). Use (probably) best command to use is the `gsub` command.

`y.name.sub` Same as the `x.name.sub`, but for the y axis variables.

`heat.colours` The spectrum of colours for the heat map, as a vector between the lowest (negative) correlation and the highest (positive) correlation.

`show.corr.values` Logical; add the correlation values to the heatmap.

`print.corr.values` Logical; If true, prints the correlation values.

`y.axis.label,ylab`

`x.axis.label,xlab`

Details

This function takes two arguments, the x variables and the y variables, and generates a heatmap from the variables. A correlation matrix is computed from the data, melted (reshape package), and input into ggplot2 to generate a heatmap. The output is the correlations and the plot object.

Author(s)

Luke W. Johnston

Examples

```
data(state)
plotHeatmapCorr(state.x77, c('Income', 'Population'), c('Frost', 'Murder'))

xvars <- c('Income', 'Population', 'Area', 'Frost')
## Replace Area with 'Land Area (km^2)', etc, on the axes using the gsub command
plotHeatmapCorr(state.x77, xvars, print.corr.values = TRUE,
                x.name.sub = xvars %>%
                  gsub('Area', 'Land Area (km^2)', .) %>%
                  gsub('Income', 'Income ($)', .)
                )
```

---

plotManhattanStyle *Manhattan style plot*

Description

Generates a plot similar to the GWAS Manhattan plots, which are useful to show significance across multiple significance testings.

Usage

```
plotManhattanStyle(data, x, y, groups = NULL, x.axis.label = "Exposures")
```

Arguments

data	Dataset from a regression with the p-values.
x	The column in the dataset that contains the independent variables and/or the interaction variables. Must be as a character/string.
y	The column that contains the p-value data. The argument must be a character/string.
groups	The column that splits the tests up, usually is the dependent variable if the data has been looped through a regression test (eg. see loopOutputToListGEE).
x.axis.label	The label for the x-axis.

Details

See the example for a better idea of how to use the function. This style of plot is really useful to use when you have run many eg. interaction testing in a regression analysis and you want to see which variables are barely significant vs very significant, etc. Thus, multiple comparison problems can be dealt with as the plot shows how significant a variable is compared to the rest of the significance tests. This is generally the same reason why GWAS studies use Manhattan plots.

Author(s)

Luke W. Johnston

Examples

```
data(state)

## Very simple test example.  Merely to show how the function is used.
outcomes <- c('Income', 'Population')
exposures <- c('Frost', 'Illiteracy')
covariates <- c('Murder', 'LifeExp')
interaction <- 'LifeExp'

## This uses the dplyr package.
ds <- cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  rename(LifeExp = `Life Exp`,
         ## Need to rename the id variable to SID (see description
         ## above)
         SID = state.region) %>%
  arrange(SID)

loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    interactions = interaction,
                    corstr = 'exchangeable') %>%
  extractBetaFromListGEE() %>%
```

```
  unlistAndFilterIndep(., ':', pattern = TRUE) %>%
  createCI() %>%
  plotManhattanStyle(., 'indep', 'pvalue', groups = '~ dep')
```

`plotSmoothingANOVA` *Smoothing line plot, with ANOVA*

Description

Generate a bivariate plot of a factor variable (x-axis) and a continuous variable (y-axis) that overlays a smoothing line (loess), which also prints an ANOVA p-value.

Usage

```
plotSmoothingANOVA(data, x, y, id, y.axis.limits = c(0.2, 0.8))
```

Arguments

`data`	Dataset with the variables of interest.
`x`	The factor variable on the x-axis. Generally is the time variable in a longitudinal setting. Must be a string/character.
`y`	The continuous variable on the y-axis. Must be a string/character.
`id`	The grouping variable, generally the ID for the participant in a longitudinal setting.
`y.axis.limits`	Limits of the y-axis. Must be two numbers.

Details

Generally I use this for plotting longitudinal data, where the x-axis is the timepoints and the ANOVA is testing the significance across time (which may be arguably inappropriate... FIXME).

Author(s)

Luke W. Johnston

Examples

```
data(state)
cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  mutate(f.Pop = Population %>%
      quantile(., c(0, .333, .666, 1), na.rm = TRUE) %>%
      cut(Population, ., include.lowest = TRUE)) %>%
  plotSmoothingANOVA(., 'f.Pop', 'Illiteracy', id = 'state.region')
```

plotSpaghetti *Spaghetti plot*

Description

Plot subjects in a longitudinal dataset, making a 'spaghetti' plot.

Usage

```
plotSpaghetti(data, y, x, groups = "SID")
```

Arguments

data	The dataset to plot.
y	The variable to go on the y-axis.
x	The variable to go on the x-axis.
groups	The unique ID variable to differentiate subjects in a longitudinal dataset.

Author(s)

Luke W. Johnston

Examples

```
## Not run:
## A pretend case (not a real example)
plotSpaghetti(dataset, 'Height', 'Year', 'SubjectID')

## End(Not run)
```

plotVisreg *Visualizing adjusted linear regression models*

Description

Generates plots of a linear regression model which includes confounding variables.

Usage

```
plotVisreg(data, y, x, covar, ylabel = x, xlabel = y, ...)
```

Arguments

data	Dataset with the variables of interest
y	The dependent or outcome variable in the regression equation
x	The independent or exposure variable in the regression equation
covar	The confounding variables, that is the variables being adjusted for
ylabel	The y-axis label
xlabel	The x-axis label
...	Other options. In development

Details

This function runs a linear regression on the specified variables and plots the partial residuals. This allows for visualizing the relationship between the outcome and the exposure, after adjusting for confounders. A linear slope is plotted through the partial residuals, with a confidence interval band around it. The output is a plot. This function depends on **visreg**.

Value

Outputs a plot of the regression model

Author(s)

Luke Johnston

---

rstatsToolkit *rstatsToolkit.*

Description

rstatsToolkit.

---

summarySE *Summarize means and standard errors of the mean*

Description

Calculates the sample size, mean, standard deviation, standard error of the mean, and the confidence interval of specified variables.

Usage

```
summarySE(data = NULL, measurevar, groupvars = NULL, na.rm = FALSE,
  conf.interval = 0.95, .drop = TRUE)
```

Arguments

`data`	A dataset (dataframe) that contains the values to be summarized
`measurevar`	The name of a column that contains the variable to be summarized
`groupvars`	A vector containing names of columns that contain grouping variables
`na.rm`	A binary (boolean) response that indicates whether to ignore missing (NA) data
`conf.interval`	Percent range of the confidence interval

Details

I took this function on 2014-01-21 from the website `http://www.cookbook-r.com/Graphs`. It basically summarizes the provided data by giving count, mean, standard deviation, standard error of the mean, and confidence interval (default 95 The dependencies are **plyr**

Value

Outputs a dataframe that contains the summarized statistics (means, etc.)

Author(s)

Cookbook R

`themeWhite` *Custom white ggplot theme*

Description

Creates a white, simple theme for **ggplot2** objects

Usage

```
themeWhite()
```

Details

The default **ggplot2** theme is decent for most purposes, but is visually unappealing. This function aims to correct that by setting the theme to something more similar to the default in the base R plot package. The function dependes on **ggplot2**.

Author(s)

Luke Johnston

Examples

```
## This creates a white theme
themeWhite()
```

`unlistAndFilterIndep` *Unlist a list of dataframes and filter a pattern or variable.*

Description

Unlist a list of dataframes, convert into a single dataframe, and filter out a string or pattern from the 'indep' column.

Usage

```
unlistAndFilterIndep(data, x, pattern = FALSE)
```

Arguments

`data`	The list object with the dataframes
`x`	The variables of interest (eg. exposures) that are within the 'indep' column.
`pattern`	Logical: TRUE if `x` is a pattern rather than an explicit list of variables and FALSE if `x` is an explicit list of variables (eg. exposures).

Details

This function is generally used within a chain of other commands that creates a list of dataframes, generally from a regression. The list gets unlisted and converted into a single dataset. Afterward, the dataset gets filtered by the variables of interest (eg. the exposures) that are contained within the 'indep' column.

Value

Outputs a single dataframe with only the rows with the variables from `x`.

Author(s)

Luke W. Johnston

Examples

```
data(state)

## Very simple test example.  Merely to show how the function is used.
outcomes <- c('Income', 'Population')
exposures <- c('Frost', 'Illiteracy')
covariates <- c('Murder', 'LifeExp')
interaction <- 'LifeExp'

## This uses the dplyr package.
ds <- cbind(state.region, state.x77) %>%
  as.data.frame() %>%
  rename(LifeExp = `Life Exp`,
```

```
          ## Need to rename the id variable to SID (see description
          ## above)
          SID = state.region) %>%
  arrange(SID)

## Without interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    corstr = 'exchangeable') %>%
  extractBetaFromListGEE() %>%
  unlistAndFilterIndep(., exposures)

## With interaction
loopOutputToListGEE(ds, outcomes, exposures, covariates,
                    interaction, corstr = 'exchangeable') %>%
  extractBetaFromListGEE() %>%
  unlistAndFilterIndep(., ':', pattern = TRUE)
       ## The ':' represents an interaction
```

Index