Package 'furniture'

July 4, 2016

Type Package

Title Table1 and TableX for Social Scientists

Version 0.2.0

Date 2016-06-28

Author Tyson Barrett [aut, cre],
Emily Brignone [aut]

Maintainer Tyson Barrett <t.barrett@aggiemail.usu.edu>

Description Social Science functions: creates tables for academic articles, runs two-part models, and produces marginal effects for GLM and Two-Part Models. table1() produces the common Table 1 in research articles with descriptive statistics. tableX() creates a summary table of ``lm" or ``glm" objects for model summaries. tp() runs two part models. frames() produces average marginal effects. For use in "#Rstats for Social Sciences".

Depends car

Suggests MASS, dplyr

Encoding UTF-8

License GPL-2

NeedsCompilation no

R topics documented:

`furniture-package` *Table1 and TableX for Social Scientists*

Description

Social Science functions: creates tables for academic articles, runs two-part models, and produces marginal effects for GLM and Two-Part Models. table1() produces the common Table 1 in research articles with descriptive statistics. tableX() creates a summary table of "lm" or "glm" objects for model summaries. tp() runs two part models. frames() produces average marginal effects. For use in "#Rstats for Social Sciences".

Details

The DESCRIPTION file:

Package:	furniture
Type:	Package
Title:	Table1 and TableX for Social Scientists
Version:	0.2.0
Date:	2016-06-28
Authors@R:	c(person("Tyson", "Barrett", role=c("aut","cre"), email = "t.barrett@aggiemail.usu.edu"), person("Emily",
Author:	Tyson Barrett [aut, cre], Emily Brignone [aut]
Maintainer:	Tyson Barrett <t.barrett@aggiemail.usu.edu>
Description:	Social Science functions: creates tables for academic articles, runs two-part models, and produces margina
Depends:	car
Suggests:	MASS, dplyr
Encoding:	UTF-8
License:	GPL-2

Index of help topics:

The furniture package provides two functions, both used to create tables for academic reporting. `table1()` creates a descriptive statistics table in a format that is ready to export. `tableX()` summarizes linear model output.

Author(s)

Tyson Barrett [aut, cre], Emily Brignone [aut]

Maintainer: Tyson Barrett <t.barrett@aggiemail.usu.edu>

Examples

```
## Data from MASS package ##
library(MASS)
data("birthwt")

## Using dplyr ##
library(dplyr)
b = mutate(.data=birthwt,
           smoke = as.factor(smoke),
           race  = as.factor(race),
           ht    = as.factor(ht),
           ui    = as.factor(ui))
levels(b$race) = c("white", "black", "other")

## Table1 examples##

table1(b, c("age", "race", "smoke", "ptl", "ht", "ui", "ftv"),
       splitby="low",
       test=TRUE,
      var.names = c("Age", "Race", "Smoking Status", "Previous Premature Labors", "Hypertension",
                    "Uterine Irratibility", "Physician Visits"),
       splitby_labels = c("Regular Birthweight", "Low Birthweight"))

table1(b, c("age", "race", "smoke", "ptl", "ht", "ui", "ftv"),
       splitby="low",
       test=TRUE,
      var.names = c("Age", "Race", "Smoking Status", "Previous Premature Labors", "Hypertension",
                    "Uterine Irratibility", "Physician Visits"),
       splitby_labels = c("Regular Birthweight", "Low Birthweight"),
       test.type = "or",
       format.output = "stars")

## TableX examples ##

fit1 = lm(bwt ~ age + race, data=birthwt)
fit2 = lm(bwt ~ age + race + ht + ui + smoke + ptl, data=birthwt)
fit3 = lm(bwt ~ age + race + ht + ui + smoke + ptl + ftv, data=birthwt)

models = list(fit1, fit2, fit3)
tableX(models=models, model.names = c("1", "2", "3"))
```

`frames` *Average Marginal Effects for Generalized Linear Models*

Description

Produces the average marginal effects for generalized linear models with links of "probit", "logit", "log", and "identity". Useful to produce the marginal effects of variables in the outcome's original metric (e.g., dollars, counts, etc.). The function can take a model already run (as a "glm" object) or can run the model with the formula and family.

Usage

```
frames(model = NULL, formula = NULL, family = NULL, data = NULL, bootsize = 1000, ci = 0.95)
```

Arguments

model	(optional if formula, family, and data are not NULL) the model object (i.e., "glm")
formula	(optional if model is not NULL) the formula for the GLM modeling procedure
family	(optional if model is not NULL) the distribution and link of the GLM modeling procedure
data	(optional if model is not NULL) the data that the formula will be applied to in the modeling procedure
bootsize	(default = 1,000) the number of bootstrap samples used for the confidence intervals
ci	(default = 0.95) the confidence level for the confidence intervals

Details

Uses bootstrapping to get the confidence intervals. Marginal effects are produced using the derivatives of the variables.

Value

A list is returned of the type "frames" with:

AME	a data.frame of the average marginal effects
Model	the output of summary.glm(model) (e.g., coefficients, standard errors, p-values, call, etc.)
Variables	A vector of the variables in the model. Includes the intercept term.
Family	The family distribution and link function
Bootsize	the number of bootstrap samples used for the confidence intervals
Alpha	the alpha level for the confidence intervals (1 - ci)
Data	the data.frame

Note

The frames() function is still in early development. The results will be similar to Stata's margin command but the exact values and confidence intervals will likely be somewhat different.

Author(s)

Tyson S. Barrett

References

http://www.stata-journal.com/sjpdf.html?articlenum=st0086

See Also

tp2frames

Examples

```
library(MASS)
data(birthwt)

for (i in c(1,4:5,7:8)){
  birthwt[,i] = as.factor(birthwt[,i])
}

logglm   = glm(bwt ~ smoke + age + lwt + race + ht + ui, data=birthwt, family = Gamma(link = "log"))
lgtglm   = glm(low ~ smoke + age + lwt + race + ht + ui, data=birthwt, family = "binomial")
myglm    = glm(bwt ~ smoke + age + lwt + race + ht + ui, data=birthwt)

log = frames(logglm, bootsize = 100)
lgt = frames(lgtglm, bootsize = 100)
lin = frames(myglm, bootsize = 100)
```

---

print.frames *Print Method for the Frames Object*

Description

The print method for the frames object. By default, only the Average Marginal Effects are shown in the output from the frames object.

Usage

```
print.frames(x, ...)
```

Arguments

x	the "frames" object
...	other arguments that can be applied to the print function

Value

The data.frame of AME from the `frames()` function is printed.

Author(s)

Tyson S. Barrett

See Also

`frames`, `tp2frames`, `summary.frames`

print.tp *Print Method for Two Part Models*

Description

The print method for two part model objects of class "tp".

Usage

```
print.tp(x, ...)
```

Arguments

x	the "tp" object from the tp() function.
...	other arguments that can be applied to the print function

Value

Prints the coefficient's table from summary.glm() for both parts of the model.

Author(s)

Tyson S. Barrett

See Also

tp, summary.tp

summary.frames *Summary Method for Frames Objects*

Description

The summary method for objects produced via frames().

Usage

```
summary.frames(object, ...)
```

Arguments

object	the "frames" object
...	other arguments that can be applied to the summary method

Value

Prints the summary.glm(framesobj) and the average marginal effects

Author(s)

Tyson S. Barrett

See Also

frames, print.frames, tp2frames

Examples

```
library(MASS)
data(birthwt)

for (i in c(1,4:5,7:8)){
  birthwt[,i] = as.factor(birthwt[,i])
}

logglm   = glm(bwt ~ smoke + age + lwt + race + ht + ui, data=birthwt, family = Gamma(link = "log"))
lgtglm   = glm(low ~ smoke + age + lwt + race + ht + ui, data=birthwt, family = "binomial")
myglm    = glm(bwt ~ smoke + age + lwt + race + ht + ui, data=birthwt)

log = frames(logglm, bootsize = 100)
lgt = frames(lgtglm, bootsize = 100)
lin = frames(myglm,bootsize = 100)

summary(log)
summary(lgt)
summary(lin)
```

---

summary.tp *Summary Method for tp Objects*

Description

The summary method for objects produced via the `tp()` function. Prints the model summaries and the combined average marginal effects.

Usage

```
summary.tp(object, ...)
```

Arguments

`object`	the "tp" object
`...`	other arguments that can be applied to the summary function

Value

Prints the model summaries from `summary.glm()` for both parts of the model and the combined average marginal effects.

Author(s)

Tyson S. Barrett

See Also

tp

Examples

```
library(MASS)
library(dplyr)
data(bacteria)
d = transmute(bacteria,
         outcome = ifelse(y == "y", sample(1:100, replace=TRUE, size = length(bacteria[bacteria$y=="y",1])),
         gender  = as.factor(sample(c("Male", "Female"), replace=TRUE, size = length(bacteria[,1]))),
           age     = as.numeric(sample(1:18, replace=TRUE, size = length(bacteria[,1]))),
         race    = as.factor(sample(c("White", "Black", "Other"), replace=TRUE, size = length(bacteria[,1]))),
          weight  = as.numeric(sample(30:90, replace=TRUE, size = length(bacteria[,1]))))

btp = tp(outcome ~ gender + age + race + weight, data=d)
summary(btp)
```

---

table1 *Table 1 for Social Scientists*

Description

Produces a descriptive table, stratified by an optional categorical variable, providing means/frequencies and standard deviations/percentages. It is well-formatted for easy transition to academic article or report.

Usage

```
table1(data, vars, splitby = NULL, splitby_labels = NULL, test = FALSE, test.type = "default", rour
```

Arguments

data	the data.frame that is to be summarized
vars	a vector of variables to be summarized; can be the indices or the column names
splitby	the categorical variable to stratify by; requires `levels(splitby)>0`
splitby_labels	allows for custom labels of the splitby levels; must match the number of levels of the splitby variable
test	logical; if set to TRUE then bivariate tests of significance are performed; requires `levels(splitby)>1`; automatically performs the correct test based on variable types and number of levels
test.type	has two options: "default" performs the default tests of significance only; "or" gives unadjusted odds ratios as well based on logistic regression
rounding	the number of digits after the decimal; default is 3
var.names	custom variable names to be printed in the table
format.output	has three options: 1) "full" provides the table with the type of test, test statistic, and the p-value for each variable; 2) "pvalues" provides the table with the p-values; 3) "stars" provides the table with stars indicating significance
NAkeep	when set to TRUE it also shows how many missing values are in the data by variable
m_label	when `NAkeep = TRUE` this provides a label for the missing values in the table

Value

A list is returned with potentially two items:

`Table1`	A data.frame object with the means/frequencies and standard deviations/percentages.
`Note`	A simple string vector indicating what the stars in the output signify (only available when `format.output = "stars"`)

Note

When using `dplyr` and other piping packages, the function works best when the data object is a `data.frame`. If it is not, use `data = as.data.frame(data)` before using this function.

Author(s)

Tyson Barrett and Emily Brignone

See Also

`tableX`

Examples

```
## Data from MASS package ##
library(MASS)
data("birthwt")

## Using dplyr
library(dplyr)
b = mutate(.data=birthwt,
           smoke = as.factor(smoke),
           race  = as.factor(race),
           ht    = as.factor(ht),
           ui    = as.factor(ui))
levels(b$race) = c("white", "black", "other")

## Table1 examples
table1(b, c("age", "race", "smoke", "ptl", "ht", "ui", "ftv"),
       NAkeep=TRUE)

table1(b, c("age", "race", "smoke", "ptl", "ht", "ui", "ftv"),
       splitby="low",
       NAkeep=TRUE)

table1(b, c("age", "race", "smoke", "ptl", "ht", "ui", "ftv"),
       splitby="low",
       test=TRUE)

table1(b, c("age", "race", "smoke", "ptl", "ht", "ui", "ftv"),
       splitby="low",
       test=TRUE,
     var.names = c("Age", "Race", "Smoking Status", "Previous Premature Labors", "Hypertension",
                   "Uterine Irratibility", "Physician Visits"),
       splitby_labels = c("Regular Birthweight", "Low Birthweight"))

table1(b, c("age", "race", "smoke", "ptl", "ht", "ui", "ftv"),
```

```
  splitby="low",
  test=TRUE,
 var.names = c("Age", "Race", "Smoking Status", "Previous Premature Labors", "Hypertension",
                "Uterine Irratibility", "Physician Visits"),
  splitby_labels = c("Regular Birthweight", "Low Birthweight"),
  test.type = "or",
  format.output = "stars")
```

`tableX` *Table X for Social Scientists*

Description

Provides a summary of n > 1 linear models in a well-formatted table, ready for export into a social science academic article or report.

Usage

```
tableX(models, model.names, type = "default", level = 0.95)
```

Arguments

`models`	a list of the model objects; can be "lm" or "glm" objects
`model.names`	a vector of character strings that will name each model
`type`	has two options: 1) "default" produces the coefficients, standard errors, and stars indicating significance; 2) "exp" provides exponentiated coefficients and expontiated confidence intervals
`level`	the level of confidence in the confidence intervals when `type = "exp"`.

Value

Outputs a list with two elements:

`Table`	the table of the model summary
`Note`	indicates the level of significance that the stars signify; only useful for when `type = "default"`

Note

As of now, only "lm" and "glm" objects are able to be included. This will be updated with future releases.

Author(s)

Tyson Barrett

See Also

`table1`

Examples

```
## The function is currently defined as
function (models, model.names, type = "default", level = 0.95)
{
    if (type != "default" & type != "exp")
      stop(cat(paste("The", type, "method is not supported. Yet. Let me know you want it @ t.barrett@aggiemai
         "\n", "Types Supported: default (reports estimates and standard errors) \n                and exp (r
    if (!all(sapply(models, class) == "lm" | sapply(models, class) ==
        "glm"))
      stop(cat(paste("The models must be 'lm' or 'glm' objects. If there is another modeling type \n
    m = m2 = rows = pval = se = obs = N = list()
    for (i in seq_along(models)) {
        summed = summary(models[[i]])$coefficients
        m[[i]] = data.frame(Est = summed[, 1])
        se[[i]] = data.frame(SE = summed[, 2])
        pval[[i]] = data.frame(PValue = summed[, 4])
        obs[[i]] = data.frame(Est = sum(!is.na(models[[i]]$fitted.values)))
    }
    if (type == "default") {
        for (i in seq_along(m)) {
            m2[[i]] = data.frame(Est = round(m[[i]], 2), SE = round(se[[i]],
                2), Sig = paste(ifelse(pval[[i]] < 0.001, "***",
                ifelse(pval[[i]] < 0.01, "**", ifelse(pval[[i]] <
                  0.05, "*", ""))), names = rownames(m[[i]]))
            N[[i]] = data.frame(Est = obs[[i]], SE = NA, Sig = NA,
                names = "N")
            m2[[i]] = rbind(N[[i]], m2[[i]])
        }
    }
    if (type == "exp") {
        for (i in seq_along(m)) {
            cis = exp(confint(models[[i]], alpha = 1 - level))
            m2[[i]] = data.frame(Est = round(exp(m[[i]][, "Est"]),
                3), Lower = round(cis[, 1], 2), Upper = round(cis[,
                2], 2), names = rownames(m[[i]]))
            N[[i]] = data.frame(Est = obs[[i]], Lower = NA, Upper = NA,
                names = "N")
            m2[[i]] = rbind(N[[i]], m2[[i]])
        }
    }
    options(warn = -1)
    merged = Reduce(function(...) merge(..., by = "names", all = TRUE),
        m2)
    names(merged) = c("Variables", rep(model.names, each = 3))
    merged[, c(seq(3, length.out = length(model.names), by = 3))] <- sapply(merged[,
        c(seq(3, length.out = length(model.names), by = 3))],
        as.character)
    merged[is.na(merged)] <- ""
    Note = paste("Sig Levels: *** < 0.001, ** < 0.01, * < 0.05")
    return(list(Table = merged, Note = Note))
  }

## Data from MASS package ##
library(MASS)
data("birthwt")
```

```
## Modeling Linear Regression ##
fit1 = lm(bwt ~ age + race, data=birthwt)
fit2 = lm(bwt ~ age + race + ht + ui + smoke + ptl, data=birthwt)
fit3 = lm(bwt ~ age + race + ht + ui + smoke + ptl + ftv, data=birthwt)

## TableX ##
models = list(fit1, fit2, fit3)
tableX(models=models, model.names = c("1", "2", "3"))

## Modeling Logistic Regression ##
log1 = glm(low ~ age + race, data=birthwt, family="binomial")
log2 = glm(low ~ age + race + ht + ui + smoke + ptl, data=birthwt, family="binomial")
log3 = glm(low ~ age + race + ht + ui + smoke + ptl + ftv, data=birthwt, family="binomial")

## TableX ##
models = list(log1, log2, log3)
tableX(models=models, model.names = c("1", "2", "3"), type="exp")
```

`tp` *Two-Part Generalized Linear Models*

Description

Produces a two-part model (sometimes refered to as a "Hurdle Model"). The function is built just like regular `glm()` but needs additional specification of what link function to use for the binary part and the family and link to use for the count/OLS part. This procedure is very useful for outcome data that have a large number of zero's (more than would be expected via most distributions). Therefore, it is assumed that two mechanisms are at play: 1) that affects whether an individual has any of the outcome vs. none, and 2) for those that do have some of the outcome, that affects how much. It is often useful for cost and utilization analyses.

Usage

```
tp(formula, data, bin.link = "logit", count.family = poisson(link = "log"))
```

Arguments

`formula`	the formula in general form (i.e., y ~ x1 + x2 + ...)
`data`	the data for the modeling
`bin.link`	the binary link function. Can be either "logit" or "probit"
`count.family`	the family for the count portion of the models (e.g., `poisson(link = "log")`)

Details

The function is built on the base function `glm()` and can therefore perform the types of models inherent in that function.

Value

A list of class "tp" is returned:

`Binary`	the binary portion of the model (distribution of "binomial" with specified link) with all of the data
`Count`	the count portion of the model (specified distribution and link) fit to only the data where y > 0

Author(s)

Tyson S. Barrett

References

http://www.stata-journal.com/article.html?article=st0368

See Also

`tp2frames`

Examples

```
library(MASS)
library(dplyr)
data(bacteria)
d = transmute(bacteria,
         outcome = ifelse(y == "y", sample(1:100, replace=TRUE, size = length(bacteria[bacteria$y=="y",1])),
         gender  = as.factor(sample(c("Male", "Female"), replace=TRUE, size = length(bacteria[,1]))),
           age     = as.numeric(sample(1:18, replace=TRUE, size = length(bacteria[,1]))),
         race    = as.factor(sample(c("White", "Black", "Other"), replace=TRUE, size = length(bacteria[,1]))),
          weight  = as.numeric(sample(30:90, replace=TRUE, size = length(bacteria[,1]))))

btp = tp(outcome ~ gender + age + race + weight, data=d)
summary(btp)
```

`tp2frames` *Average Marginal Effects for Two-Part Models*

Description

Produces the average marginal effects for two-part models (sometimes refered to as "Hurdle Models"). Part 1 is a binomial GLM of "probit" or "logit" links. Part 2 is often a count model (e.g., poisson or gamma) with a "log" or "identity" link. It combines the marginal effects from both models. Useful to produce the marginal effects of variables in the outcome's original metric (e.g., dollars, counts, etc.).

Usage

```
tp2frames(model, bootsize = 1000, ci = 0.95)
```

Arguments

model	the model object of class "tp"
bootsize	(default = 1,000) the number of bootstrap samples used for the confidence intervals
ci	(default = 0.95) the confidence level for the confidence intervals

Value

A list is returned of the type "frames" with:

AME	a data.frame of the average marginal effects
Model	the output of summary.glm(model) (e.g., coefficients, standard errors, p-values, call, etc.)
Variables	A vector of the variables in the model. Includes the intercept term.
Family	The family distribution and link function
Bootsize	the number of bootstrap samples used for the confidence intervals
Alpha	the alpha level for the confidence intervals (1 - ci)
Data	the data.frame

Note

The function only works with objects produced from the tp() function in the furniture package.

Author(s)

Tyson S. Barrett

References

http://www.stata.com/help.cgi?margins

See Also

frames, tp

Examples

```
library(MASS)
library(dplyr)
data(bacteria)
d = transmute(bacteria,
         outcome = ifelse(y == "y", sample(1:100, replace=TRUE, size = length(bacteria[bacteria$y=="y",1])),
         gender  = as.factor(sample(c("Male", "Female"), replace=TRUE, size = length(bacteria[,1]))),
           age     = as.numeric(sample(1:18, replace=TRUE, size = length(bacteria[,1]))),
         race    = as.factor(sample(c("White", "Black", "Other"), replace=TRUE, size = length(bacteria[,1])))
          weight  = as.numeric(sample(30:90, replace=TRUE, size = length(bacteria[,1]))))

btp = tp(outcome ~ gender + age + race + weight, data=d)
summary(btp)
```

Index