Package 'rockchalk'

August 13, 2014

Type Package

Title Regression Estimation and Presentation

Version 1.8.86

Date 2014-08-09

Author Paul E. Johnson <pauljohn@ku.edu>

Maintainer Paul E. Johnson <pauljohn@ku.edu>

Description A collection of functions for ease of presentation and interpretation of regression analysis.
These functions are used to produce the statistics lectures
in http://pj.freefaculty.org/guides. There are functions to explore the effect (or lack thereof) from
``mean centering,'' ``residual centering'', and standardization.
The vignette ``rockchalk'' offers a fairly comprehensive overview.
The vignette ``Rstyle'' has advice about coding in R.
See the outreg function for a convenient way to generate
regression models for reports. For nice, easy to use pictures,see plotSlopes, plotCurves, plotPlane.
The package title ``rockchalk'' refers to our school motto,``Rock Chalk Jayhawk, Go K.U.''.

License GPL (>= 3.0)

LazyLoad yes

Imports grDevices, lme4, car, MASS

Depends R (>= 2.10)

NeedsCompilation no

Repository CRAN

Date/Publication 2014-08-13 16:42:17

R topics documented:

`rockchalk-package` *Miscellaneous regression functions*

Description

Package:	rockchalk
Type:	Package
Version:	1.4
Date:	2012-01-10
License:	GPL >= 3
LazyLoad:	yes

The rockchalk package includes an ever-growing collection of functions that assist in the presentation of regression models. The initial function was `outreg`, which produces LaTeX tables that summarize one or many fitted regression models. It also offers plotting conveniences like `plotPlane` and `plotSlopes`, which illustrate some of the variables from a fitted regression model. For a detailed check on multicollinearity, see `mcDiagnose`. The user should be aware of this fact: Not all of these functions lead to models or types of analysis that we endorese. Rather, they all lead to analysis that is endorsed by some scholars, and we feel it is important to facilitate the comparison of competing methods. For example, the function `standardize` will calculate standardized regression coefficients for all predictors in a regression model's design matrix in order to replicate results from other statistical frameworks, no matter how unwise the use of such coefficients might be. The function `meanCenter` will allow the user to more selectively choose variables for centering (and possibly standardization) before they are entered into the design matrix. Because of the importance of interaction variables in regression analysis, the `residualCenter` and `meanCenter` functions are offered. While mean centering does not actually help with multicollinearity of interactive terms, many scholars have argued that it does. The meanCenter function can be compared with the "residual centering" of interaction terms.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Maintainer: Paul Johnson <<pauljohn@ku.edu>>

References

http://pj.freefaculty.org/R

`addLines` *Superimpose regression lines on a plotted plane*

Description

The examples will demonstrate the intended usage.

Usage

```
addLines(to = NULL, from = NULL, col, lwd = 2, lty = 1)
```

Arguments

`to`	a 3d plot object produced by plotPlane
`from`	output from a plotSlopes or plotCurves function (class="rockchalk")
`col`	color of plotted lines (default: "red")
`lwd`	line width of added lines (default: 2)
`lty`	line type of added lines (default: 1)

Details

From an educational stand point, the objective is to assist with the student's conceptualization of the two and three dimensional regression relationships.

Value

NULL, nothing, nicht, nada.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
library(rockchalk)


set.seed(12345)

dat <- genCorrelatedData2(100, means = c(0,0,0,0), sds = 1, rho = 0,
       beta = c(0.03, 0.01, 0.1, 0.4, -0.1), stde = 2)

dat$xcat1 <- gl(2,50, labels=c("M","F"))

dat$xcat2 <- cut(rnorm(100), breaks=c(-Inf, 0, 0.4, 0.9, 1, Inf),
labels=c("R", "M", "D", "P", "G"))

dat$y2 <- with(dat, 0.03 + 0.1*x1 + 0.1*x2 + 0.25*x1*x2 + 0.4*x3 -
               0.1*x4 + 0.2 * as.numeric(xcat1) +
               contrasts(xcat2)[as.numeric(xcat2), ] %*% c(-0.1, 0.1, 0.2, 0) +
               1 * rnorm(100))

## linear ordinary regression
m1 <- lm(y ~ x1 + x2 + x3 + x4, data = dat)

## These will be parallel lines

plotSlopes(m1, plotx = "x1", modx = "x2", modxVals = "std.dev.",
           n = 5, main = "A plotSlopes result with \"std.dev.\" values of modx")


m1ps <- plotSlopes(m1, plotx = "x1", modx = "x2", modxVals = c(-2,0,2))

m1pp <- plotPlane(m1, plotx1 = "x1", plotx2 = "x2", ticktype =
"detailed", npp = 10)

addLines(from = m1ps, to = m1pp, lty = 1, lwd = 2)

m1pp <- plotPlane(m1, plotx1 = "x1", plotx2 = "x2", ticktype = "detailed", npp = 10)
addLines(from = m1ps, to = m1pp, lty = 2, lwd = 5, col = "green")

## Other approach would wrap same into the linesFrom argument in plotPlane

plotPlane(m1, plotx1 = "x1", plotx2 = "x2", ticktype = "detailed",
          npp = 10, linesFrom = m1ps)

## Need to think more on whether dotted lines from ps object should
## be converted to solid lines in plotPlane.
```

`centerNumerics` *Find numeric columns, center them, re-name them, and join them with the original data.*

Description

The meanCentered regression function requires centered-inputs when calculations are predicted. For comparison with ordinary regression, it is convenient to have both centered and the original data side-by-side. This function handles that. If the input data has columns c("x1","x2","x3"), then the centered result will have columns c("x1","x2","x3","x1c","x2c","x3c"), where "c" indicates "mean-centered". If standardize=TRUE, then the result will have columns c("x1","x2","x3","x1cs","x2cs","x3cs"), where "cs" indicate "centered and scaled".

Usage

```
centerNumerics(data, center, standardize = FALSE)
```

Arguments

`data`	Required. data frame or matrix.
`center`	Optional. If nc is NOT supplied, then all numeric columns in data will be centered (possiblly scaled). Can be specified in 2 formats. 1) Vector of column names that are to be centered, 2) Vector named elements giving values of means to be used in centering. Values must be named, as in c("x1" = 17, "x2" = 44). (possibly scaled).
`standardize`	Default FALSE. If TRUE, the variables are first mean-centered, and then divided by their standard deviations (scaled). User can supply a named vector of scale values by which to divide each variable (otherwise sd is used). Vector must have same names and length as center argument. Variables can be entered in any order (will be resorted inside function).

Value

A data frame with 1) All original columns 2) additional columns with centered/scaled data, variables renamed "c" or "cs" to indicate the data is centered or centered and scaled. Attributes "centers" and "scales" are created for "record keeping" on centering and scaling values.

Author(s)

<pauljohn@ku.edu>

Examples

```
set.seed(12345)
dat <- data.frame(x1=rnorm(100, m = 50), x2 = rnorm(100, m = 50),
    x3 = rnorm(100, m = 50), y = rnorm(100),
    x4 = gl(2, 50, labels = c("Male","Female")))
datc1 <- centerNumerics(dat)
head(datc1)
summarize(datc1)
datc2 <- centerNumerics(dat, center=c("x1", "x2"))
head(datc2)
summarize(datc2)
attributes(datc2)
```

```
datc3 <- centerNumerics(dat, center = c("x1" = 30, "x2" = 40))
head(datc3)
summarize(datc3)
attributes(datc3)
datc4 <- centerNumerics(dat, center=c("x1", "x2"), standardize = TRUE)
head(datc3)
summarize(datc4)
attributes(datc4)
datc5 <- centerNumerics(dat, center=c("x1"=30, "x2"=40),
standardize = c("x2" = 5, "x1" = 7))
head(datc5)
summarize(datc5)
attributes(datc5)
```

`centralValues` *Central Tendency estimates for variables*

Description

This is needed for the creation of summaries and predicted values of regression models. It takes a data frame and returns a new data frame with one row in which the mean or mode of the columns is reported.

Usage

```
centralValues(x)
```

Arguments

x	a data frame

Value

a data frame with the same variables and one row, the summary indicators.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
myDat <- data.frame(x=rnorm(100), y=rpois(100,l=4), z = cut(rnorm(100), c(-10,-1,0,10)))
centralValues(myDat)
```

cheating *Cheating and Looting in Japanese Electoral Politics*

Description

Extracted from the "cheating-replication.dta" data file with permission by the authors, Benjamin Nyblade and Steven Reed. The Stata data file provided by the authors included many constructed variables that have been omitted. Within R, these can be easily re-contructed by users.

Usage

```
data(cheating)
```

Format

data.frame: 16623 obs. on 27 variables

Details

Special thanks to NyBlade and Reed for permission to repackage this data. Also special thanks to them for creating an especially transparent variable naming scheme.

The data set includes many columns for variables that can easily be re-constructed from the columns that are provided here. While Stata users might need to manually create 'dummy variables' and interactions, R users generally do not do that manually.

These variables from the original data set were omitted:

Dummy variables for the year variable: c("yrd1", "yrd2", ..., "yrd17", "yrd18")

Dummy variables for the ku variable: c("ku1", "ku2", ..., "ku141", "ku142")

Constructed product variables: c("actualratiosq", "viabsq", "viab_candcamp_divm", "viab_candothercamp_divm", "viabsq_candcamp_divm", "viabsq_candothercamp_divm", "absviab_candcamp", "absviab_candothercamp", "absviab_candcamp_divm", "absviab_candothercamp_divm", "viabsq_candcamp", "viabsq_candothercamp", "viab_candcamp", "viab_candothercamp", "candothercamp_divm", "candcamp_divm", "candcamp-minusm", "candothercampminusm", "predratiosq", "absviab")

Mean centered variables: constr2 <- c("viab_candcampminusm", "viab_candothercampminusm", "viabsq_candothercampminusm", "viabsq_candcampminusm")

In the end, we are left with these variables:

[1] "ku" [2] "prefecture" [3] "dist" [4] "year" [5] "yr" [6] "cdnr" [7] "jiban" [8] "cheating" [9] "looting" [10] "actualratio" [11] "viab" [12] "inc" [13] "cons" [14] "ur" [15] "newcand" [16] "jwins" [17] "cons_cwins" [18] "oth_cwins" [19] "camp" [20] "fleader" [21] "incablast" [22] "predratio" [23] "m" [24] "candcamp" [25] "candothercamp" [26] "kunocheat" [27] "kunoloot"

Author(s)

Paul E. Johnson <pauljohn@ku.edu>, on behalf of Benjamin Nyblade and Steven Reed

Source

http://faculty.arts.ubc.ca/bnyblade/publications.html.

References

Benjamin Nyblade and Steven Reed, "Who Cheats? Who Loots? Political Competition and Corruption in Japan, 1947-1993." American Journal of Political Science 52(4): 926-41. October 2008.

Examples

```
require(rockchalk)
data(cheating)

table1model2 <- glm(cheating ~ viab + I(viab^2) + inc + cons + ur
+ newcand + jwins + cons_cwins + oth_cwins, family = binomial(link
= "logit"), data = cheating)

predictOMatic(table1model2)

predictOMatic(table1model2, interval = "confidence")

## The publication used "rare events logistic", which I'm not bothering
## with here because I don't want to invoke additional imported packages.
## But the ordinary logit results are proof of concept.
```

---

checkIntFormat *A way of checking if a string is a valid file name.*

Description

A copy from R's grDevices:::checkIntFormat because it is not exported there

Usage

```
checkIntFormat(s)
```

Arguments

s	An integer

Value

logical: TRUE or FALSE

Author(s)

R Core Development Team

checkPosDef *Check a matrix for positive definitness*

Description

Uses eigen to check positive definiteness. Follows example used in MASS package by W. N. Venables and Brian D. Ripley

Usage

```
checkPosDef(X, tol = 1e-06)
```

Arguments

X	A matrix
tol	Tolerance (closeness to 0 required to declare failure)

Value

TRUE or FALSE

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

combineLevels *recode a factor by "combining" levels*

Description

This makes it easy to put levels together and create a new factor variable. If a factor variable is currently coded with levels c("Male","Female","Man", "M"), and the user needs to combine the redundant levels for males, this is the function to use! This is a surprisingly difficult problem in R.

Usage

```
combineLevels(fac, levs, newLabel = "combinedLevels")
```

Arguments

fac	An R factor variable, either ordered or not.
levs	The levels to be combined. Users may specify either a numerical vector of level values, such as c(1,2,3), to combine the first three elements of level(fac), or they may specify level names. This can be done as a character vector of *correctly spelled* factor values, such as c("Yes","Maybe","Always") or it may be provided as a subset of the output from levels, such as levels(fac)[1:3].
newLabel	A character string that represents the label of the new level to be created when levs values are combined.

Details

If the factor is an ordinal factor, then levels may be combined only if they are adjacent. A factor with levels c("Lo","Med","Hi","Extreme") allows us to combine responses "Lo" and "Med", while it will NOT allow us to combine "Lo" with "Hi".

A non-ordered factor can be reorganized to combine any values, no matter what positions they occupy in the levels vector.

Value

A new factor variable, with unused levels removed.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
x <- c("M","A","B","C","A","B","A","M")
x <- factor(x)
levels(x)
x2a <- combineLevels(x, levs = c("M","A"), newLabel = "M_or_A")
addmargins(table(x2a, x, exclude=NULL))
x2b <- combineLevels(x, c(1,4), "M_or_A")
addmargins(table(x2b, x, exclude=NULL))
x3 <- combineLevels(x, levs = c("M","A","C"), newLabel = "MAC")
addmargins(table(x3, x, exclude=NULL))
## Now an ordinal factor
z <- c("M","A","B","C","A","B","A","M")
z <- ordered(z)
levels(z)
table(z, exclude=NULL)
z2a <-  combineLevels(z, levs = c(1,2), "Good")
addmargins(table(z2a, z, exclude = NULL))
z2b <- combineLevels(z, levs = c("A","B"), "AorB")
addmargins(table(z2b, z, exclude = NULL))
```

---

cutByQuantile *Calculates the "center" quantiles, always including the median, when n is odd.*

Description

If the numeric variable has fewer than 6 unique observed values, this will send the data to `cutByTable`. The default return will find dividing points at three quantiles: c(0.25, 0.50, 0.75) If n=4, the dividing points will be c(0.20, 0.40, 0.60, 0.80) If n=5, c(0.0, 0.25, 0.50, 0.75, 1.0) Larger n that are odd will include 0.5 and evenly spaced points out to proportions 0 and 1.0. Larger n that is even will return evenly spaced points calculated by R's `pretty` function.

Usage

```
cutByQuantile(x, n = 3)
```

Arguments

x	A numeric vector.
n	The number of quantile points. See details.

Value

A vector

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

---

cutBySD *Returns center values of x, the mean, mean-std.dev, mean+std.dev*

Description

If the numeric variable has fewer than 6 unique observed values, this will send the data to cutByTable.

Usage

```
cutBySD(x, n = 3)
```

Arguments

x	A numeric variable
n	Should be an odd number 1, 3, 5, or 7. If 2 < n < 5, values that divide the data at c(m-sd, m, m+sd) are returned. If n > 4, the returned values are c(m-2sd, m-sd, m, m+sd, m+2sd).

Value

A named vector

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
x <- rnorm(100, m = 100, s = 20)
cutBySD (x, n = 3)
cutBySD (x, n = 5)
```

cutByTable — *Select most frequently occurring values from numeric or categorical variables.*

Description

The "n" most frequently occurring values are returned, sorted by frequency of occurrence (in descending order). The names attribute includes information about the percentage of cases that have the indicated values.

Usage

```
cutByTable(x, n = 5)
```

Arguments

x	A numeric or character variable
n	The maximum number of values that may be returned.

Details

This is used by plotSlopes, plotCurves, and other "newdata" making functions.

Value

A named vector.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

focalVals — *Create a focal value vector.*

Description

This selects some values of a variable and creates a new "focal vector" from them. Can use one "divider" algorithm, to be selected by name.

Usage

```
focalVals(x, divider = "quantile", n = 3)
```

Arguments

x	The input variable may be numeric or a factor.
divider	Either a quoted string name of an algorithm or a function. Default = "quantile" for numeric variables, "table" for factors. Other valid values: "seq" for an evenly spaced sequence from minimum to maximum, "std.dev." for a sequence that has the mean at the center and values on either side that are proportional to the standard deviation.
n	Desired number of focal values.

Details

This is a "wrapper" (or convenience) function that re-directs work to other functions. The functions that do the work to select the focal values for types ("table", "quantile", "std.dev.", "seq") are (cutByTable(), cutByQuantile(), cutBySD(), and plotSeq())

The built-in R function `pretty()` works as of rockchalk 1.7.2. Any function that accepts an argument n will work, as long as it creates a vector of values.

Value

A named vector of focal values selected from a variable. The values of the names should be informative and useful for plotting or other diagnostic work.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

See Also

`predictOMatic` `newdata`

---

formatVC *formatter for merMod objects copied from lme4*

Description

R packaging started to complain about usage of non-exported functions from packages. lme4 team said they might export this function at some time in future. Until then, I need to copy it.

Usage

```
formatVC(varc, digits = max(3, getOption("digits") - 2), comp = "Std.Dev.",
  formatter = format, ...)
```

Arguments

varc	variance estimates
digits	digits desired
comp	what do you want
formatter	a format function
...	other arguments

Value

formatted text

Author(s)

Doug Bates, Martin Machler, Ben Bolker, Stephen Walker

---

genCorrelatedData *Generates a data frame for regression analysis*

Description

The output is a data frame (x1, x2, y) with user-specified correlation between x1 and x2. The y (output) variable is created according to the equation y = beta1 + beta2 * x1 + beta3 * x2 + beta4 * x1 * x2 + e. The arguments determine the scales of the X matrix, the random error, and the slope coefficients.

Usage

```
genCorrelatedData(N = 100, means = c(50, 50), sds = c(10, 10), rho = 0,
  stde = 1, beta = c(0, 0.2, 0.2, 0))
```

Arguments

N	Number of cases desired
means	2-vector of means for x1 and x2
sds	2-vector of standard deviations for x1 and x2
rho	Correlation coefficient for x1 and x2
stde	standard deviation of the error term in the data generating equation
beta	beta vector of at most 4 coefficients for intercept, slopes, and interaction

Details

The vector (x1,x2) is drawn from a multivariate normal distribution in which the expected value (argument means). The covariance matrix of X is built from the standard deviations (sds) and the specified correlation between x1 and x2 (rho). It is also necessary to specify the standard deviation of the error term (stde) and the coefficients of the regression equation (beta).

Examples

```
## 1000 observations of uncorrelated x1 and x2 with no
## interaction between x1 and x2
dat <- genCorrelatedData(N=1000, rho=0, beta=c(1, 1.0, -1.1, 0.0))
  mcGraph1(dat$x1, dat$x2, dat$y, theta=20, phi=8,
  ticktype="detailed", nticks=10)
m1 <- lm(y ~ x1 + x2, data = dat)
plotPlane(m1, plotx1 = "x1", plotx2 = "x2")
```

---

genCorrelatedData2 *Generates a data frame for regression analysis.*

Description

Unlike genCorrelatedData, this new-and-improved function can generate a data frame with as many predictors as the user requests along with an arbitrarily complicated regression formula. The result will be a data frame with columns named (y, x1, x2, ..., xp).

Usage

```
genCorrelatedData2(N = 100, means = c(50, 50, 50), sds = c(10, 10, 10),
  rho = c(0, 0, 0), stde = 100, beta = c(0, 0.15, 0.1, -0.1),
  verbose = TRUE)
```

Arguments

N	Number of cases desired
means	P-vector of means for X. Implicitly sets the dimension of the predictor matrix as N x P.
sds	Values for standard deviations for columns of X. If less than P values are supplied, they will be recycled.
rho	Correlation coefficient for X. Several input formats are allowed (see lazyCor). This can be a single number (common correlation among all variables), a full matrix of correlations among all variables, or a vector that is interpreted as the strictly lower triangle (a vech).
stde	standard deviation of the error term in the data generating equation
beta	beta vector of coefficients for intercept, slopes, and nonlinear-interaction terma. The first P+1 values are the intercept and slope coefficients for the predictors. Additional elements in beta are interpreted as coefficients for nonlinear and interaction coefficients. I have decided to treat these as a column (vech) that fills into a lower triangular matrix. See Details.
verbose	TRUE or FALSE. Should information about the data generation be reported to the terminal?

Details

Arguments supplied must have enought information so that an N x P matrix of predictors can be constructed. The matrix X is drawn from a multivariate normal distribution, the expected value vector (mu vector) is given by the `means` and the var/covar matrix (Sigma) is built from user supplied standard deviations `sds` and the correlations between the columns of X, given by `rho`. The user can also set the standard deviation of the error term (`stde`) and the coefficients of the regression equation (`beta`).

If called with no arguments, this creates a data frame with X ~ MVN(mu = c(50,50,50), Sigma = diag(c(10,10,10))). y = X is N(m = 0, sd = 200). All of these details can be changed by altering the arguments.

The y (output) variable is created according to the equation

y = b1 + b2 * x1 + ...+ bk * xk + b[k+1] * x1 * ...interactions.. + e

For shorthand, I write b1 for beta[1], b2 for beta[2], and so forth.

The first P+1 arguments in the argument beta are the coefficients for the intercept and the columns of the X matrix. Any additional elements in beta are the coefficients for nonlinear and interaction terms.

Those additional values in the beta vector are completely optional. Without them, the true model is a linear regression. However, one can incorporate the effect of squared terms (conceptualize that as x1 * x1, for example) or interactions (x1 * x2) easily. This is easier to illustrate than describe. Suppose there are 4 columns in X. Then a beta vector like beta = c(0, 1, 2, 3, 4, 5, 6, 7, 8) would amount to asking for

y = 0 + 1 x1 + 2 x2 + 3 x3 + 4 x4 + 5 *x1^2 + 6 x1 x2 + 7 x1 x3 + 8 x1 x4 + error

If beta supplies more coefficients, they are interpeted as additional interactions.

When there are a many predictors and the beta vector is long, this can become confusing. I think of this as a vech for the lower triangle of a coefficient matrix. In the example with 4 predictors, beta[1:5] are used for the intercepts and slopes. The rest of the beta elements lay in like so:

X1 X2 X3 X4 X1 b6 . . X2 b7 b10 . X3 b8 b11 b13 X4 b9 b12 b14 b15

If the user only supplies b6 and b7, the rest are assumed to be 0.

To make this clear, the formula used to calculate y is printed to the console when genCorrelated-Data2 is called.

Value

A data matrix that has columns c(y, x1, x2, ..., xP)

Examples

```
## 1000 observations of uncorrelated X with no interactions
set.seed(234234)
dat <- genCorrelatedData2(N = 10, rho = 0.0, beta = c(1, 2, 1, 1),
    means = c(0,0,0), sds = c(1,1,1), stde = 0)
summarize(dat)
## The perfect regression!
m1 <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m1)
```

```
dat <- genCorrelatedData2(N = 1000, rho = 0,
    beta = c(1, 0.2, -3.3, 1.1), stde = 50)
m1 <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m1)
predictOMatic(m1)
plotCurves(m1, plotx = "x2")

## interaction between x1 and x2
dat <- genCorrelatedData2(N = 1000, rho = 0.2,
    beta = c(1, 1.0, -1.1, 0.1, 0.0, 0.16), stde = 1)
summarize(dat)
## Fit wrong model? get "wrong" result
m2w <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m2w)
## Include interaction
m2 <- lm(y ~ x1 * x2 + x3, data = dat)
summary(m2)

dat <- genCorrelatedData2(N = 1000, rho = 0.2,
    beta = c(1, 1.0, -1.1, 0.1, 0.0, 0.16), stde = 100)
summarize(dat)
m2.2 <- lm(y ~ x1 * x2 + x3, data = dat)
summary(m2.2)

dat <- genCorrelatedData2(N = 1000, means = c(100, 200, 300, 100),
    sds = 20,  rho = c(0.2, 0.3, 0.1, 0, 0, 0),
    beta = c(1, 1.0, -1.1, 0.1, 0.0, 0.16, 0, 0, 0.2, 0, 0, 1.1, 0, 0, 0.1),
    stde = 200)
summarize(dat)
m2.3w <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m2)

m2.3 <- lm(y ~ x1 * x2 + x3, data = dat)
summary(m2.3)

predictOMatic(m2.3)
plotCurves(m2.3, plotx = "x1", modx = "x2", modxVals = "std.dev.", n = 5)

simReg <- lapply(1:100, function(x){
    dat <- genCorrelatedData2(N = 1000, rho = c(0.2),
        beta = c(1, 1.0, -1.1, 0.1, 0.0, 0.46), verbose = FALSE)
    mymod <- lm (y ~ x1 * x2 + x3, data = dat)
    summary(mymod)
})

x3est <- sapply(simReg, function(reg) {coef(reg)[4 ,1] })
summarize(x3est)
hist(x3est,  breaks = 40, prob = TRUE,
    xlab = "Estimated Coefficients for column x3")

r2est <- sapply(simReg, function(reg) {reg$r.square})
summarize(r2est)
```

```
hist(r2est,  breaks = 40, prob = TRUE, xlab = "Estimates of R-square")

## No interaction, collinearity
dat <- genCorrelatedData2(N = 1000, rho = c(0.1, 0.2, 0.7),
    beta = c(1, 1.0, -1.1, 0.1), stde = 1)
m3 <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m3)

dat <- genCorrelatedData2(N=1000, rho=c(0.1, 0.2, 0.7),
    beta = c(1, 1.0, -1.1, 0.1), stde = 200)
m3 <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m3)
mcDiagnose(m3)

dat <- genCorrelatedData2(N = 1000, rho = c(0.9, 0.5, 0.4),
    beta = c(1, 1.0, -1.1, 0.1), stde = 200)
m3b <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m3b)
mcDiagnose(m3b)
```

`getAuxRsq` *retrieves estimates of the coefficient of determination from a list of regressions*

Description

Asks each regression model in a list for a summary and then reports the R-squares.

Usage

```
getAuxRsq(auxRegs)
```

Arguments

`auxRegs`	a list of fitted regression objects

Value

a numeric vector of the same length as auxRegs.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

getDeltaRsquare *Calculates the delta R-squares, also known as squared semi-partial correlation coefficients.*

Description

The change in the R-square when a variable is removed from a regression is called delta R-square. It is sometimes suggested as a way to determine whether a variable has a substantial effect on an outcome. This is also known as the squared semi-partial correlation coefficient.

Usage

```
getDeltaRsquare(model)
```

Arguments

model	a fitted regression model

Value

a vector of estimates of the delta R-squares

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
dat1 <- genCorrelatedData(N=250, means=c(100,100),
sds=c(30,20), rho=0.0,  stde = 7, beta=c(1.1, 2.4, 4.1, 0))
m1 <- lm(y ~ x1 + x2, data=dat1)
getDeltaRsquare(m1)
## more problematic in presence of collinearity
dat2 <- genCorrelatedData(N=250, means=c(100,100),
sds=c(30,20), rho=0.6,  stde = 7, beta=c(1.1, 2.4, 4.1, 0))
m2 <- lm(y ~ x1 + x2, data=dat2)
getDeltaRsquare(m2)
```

getFocal *Select focal values from an observed variable.*

Description

This is a generic function with 2 methods, getFocal.default handles numeric variables, while getFocal.factor handles factors. No other methods have been planned for preparation.

Usage

```
getFocal(x, ...)
```

Arguments

x	Required. A variable
...	Other arguments that will be passed to the user-specified xvals function.

Details

This is used in functions like `plotSlopes` or `plotCurves`.

Value

A vector.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

`getFocal.default` *Select focal values from a numeric variable*

Description

getFocal.default

Usage

```
## Default S3 method:
getFocal(x, xvals = NULL, n = 3, ...)
```

Arguments

x	Required. A variable
xvals	If `xvals` is not provided, a default divider algorithm will be selected ("quantile"). The divider algorithms provided with rockchalk are c("quantile", "std.dev.", "table", "seq"). `xvals` can also be the name of a user-supplied function, such as R's `pretty()`. If the user supplies a vector of possible values, that selection will be checked to make sure all elements are within a slightly expanded range of `x`. If a value out of range is requested, a warning is issued. Maybe that should be an outright error?
n	Number of values to be selected.
...	Other arguments that will be passed to the user-specified xvals function.

Value

A named vector of values.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
x <- rnorm(100)
getFocal(x)
getFocal(x, xvals = "quantile")
getFocal(x, xvals = "quantile", n = 5)
getFocal(x, xvals = "std.dev")
getFocal(x, xvals = "std.dev", n = 5)
getFocal(x, xvals = c(-1000, 0.2, 0,5))
```

---

getFocal.factor *Select focal values from factor variables.*

Description

This is used by predictOMatic and plotSlopes to choose values of a categorical moderator variable.

Usage

```
## S3 method for class 'factor'
getFocal(x, xvals = NULL, n = 3, ...)
```

Arguments

x	Required. A factor variable
n	Number of values to be selected.
xvals	Optional. With factor variables, this argument is generally not used because the only implemented divider algorithm is "table" (see cutByTable), which selects the n most frequently observed values. That is the default algorithm. It is legal to specify xvals = "table", but there is no point in doing that. However, xvals may take two other formats. It may be a user-specified function that can to select levels values from x or it may be a vector of labels (or, names of levels). The purpose of the latter is to check that the requested levels are actually present in the supplied data vector x. If the levels specified are not in the observed variable, then this function stops with an error message.
...	Additional arguments passed to user-specified function in xvals.

Value

A named vector of level names (labels).

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
x <- factor(c("A","B","A","B","C","D","D","D"))
getFocal(x)
getFocal(x, n = 2)
```

`getPartialCor` *Calculates partial correlation coefficients after retrieving data matrix froma fitted regression model*

Description

The input is a fitted regression model, from which the design matrix is retrieved, along with the dependent variable. The partial correlation is calculated using matrix algebra that has not been closely inspected for numerical precision. That is to say, it is in the stats book style, rather than the numerically optimized calculating format that functions like `lm()` have adopted.

Usage

```
getPartialCor(model, dvonly = TRUE)
```

Arguments

`model`	A fitted regression model, such as output from lm(). Any object that has methods model.matrix and model.frame will be sufficient.
`dvonly`	Default = TRUE. Only show first column of the full partial correlation matrix. That corresponds to the partial correlation of each predictor with y. I mean, r[yx].[others]

Details

I often criticize partial correlations because they change in a very unstable way as terms are added or removed in regression models. Nevertheless, I teach with books that endorse them, and in order to have something to criticize, I need to have a function like this. There are other packages that offer partial correlation calculations, but they are either 1) not easy to install from CRAN because of dependencies or 2) do not directly calculate the values we want to see.

To students. 1) This gives the same result as the function `cov2pcor` in `gRbase`, so far as I can tell. Why use this? Simply for convenenience. We have found that installing gRbase is a problem because it depends on packages in Bioconductor. 2) By default, I show only one column of output, the partial correlations involving the dependent variable as something being explained. The other columns that would depict the dependent variable as a predictor of the independent variables have been omitted. You can let me know if you think that's wrong.

Please note I have not gone out of my way to make this calculation "numerically stable." It does not use any orthogonal matrix calculations; it is using the same textbook theoretical stats formula

that is used by cov2pcor in gRbase and in every other package or online source I could find. I prepared a little WorkingExample file matrix-partial-correlations-1.R that discusses this, in case you are interested (http://pj.freefaculty.org/R).

Value

A column or matrix of partial correlation coefficients

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

`getVIF` *Converts the R-square to the variance inflation factor*

Description

calculates vif = 1/(1-R-square)

Usage

```
getVIF(rsq)
```

Arguments

`rsq`	a vector of real values, presumably fitted R-squares

Value

a vector of vif estimates

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

`lazyCor` *Create correlation matrices.*

Description

Use can supply either a single value (the common correlation among all variables), a column of the lower triangular values for a correlation matrix, or a candidate matrix. The function will check X and do the right thing. If X is a matrix, check that it is a valid correlation matrix. If its a single value, use that to fill up a matrix. If itis a vector, try to use it as a vech to fill the lower triangle..

Usage

```
lazyCor(X, d)
```

Arguments

`X`	Required. May be one value, a vech, or a matrix
`d`	Optional. The number of rows in the correlation matrix to be created. lazyCor will deduce the desired size from X if possible. If X is a single value, d is a required argument.

Value

A correlation matrix.

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
lazyCor(0.5, 8)
lazyCor(c(0.1, 0.2, 0.3))
lazyCor(c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6))
```

`lazyCov` *Create covariance matrix from correlation and standard deviation information*

Description

This is a flexible function that allows lazy R programmers to create covariance matrix. The user may be lazy because the correlation and standard deviation infomation may be supplied in a variety of formats.

Usage

```
lazyCov(Rho, Sd, d)
```

Arguments

Rho	Required. May be a single value (correlation common among all variables), a vector of the lower triangular values (vech) of a correlation matrix, or a symmetric matrix of correlation coefficients.
Sd	Required. May be a single value (standard deviation common among all variables) or a vector of standard deviations, one for each variable.
d	Optional. Number of rows or columns. lazyCov may be able to deduce the required dimension of the final matrix from the input. However, when the user supplies only a single value for both Rho and Sd, d is necessary.

Value

covariance matrix.

Author(s)

<pauljohn@ku.edu>

Examples

```
##correlation 0.8 for all pairs, standard deviation 1.0 of each
lazyCov(Rho = 0.8, Sd = 1.0, d = 3)
## supply a vech (lower triangular values in a column)
lazyCov(Rho = c(0.1, 0.2, 0.3), Sd = 1.0)
## supply vech with different standard deviations
lazyCov(Rho = c(0.1, 0.2, 0.3), Sd = c(1.0, 2.2, 3.3))
newRho <- lazyCor(c(0.5, 0.6, 0.7, -0.1, 0.1, 0.2))
lazyCov(Rho = newRho, Sd = 1.0)
lazyCov(Rho = newRho, Sd = c(3, 4, 5, 6))
```

---

lmAuxiliary *Estimate leave-one-variable-out regressions*

Description

This is a convenience for analysis of multicollinearity in regression.

Usage

```
lmAuxiliary(model)
```

Arguments

model	a fitted regression model

Value

a list of fitted regressions, one for each omitted variable.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

`magRange` *magRange Magnify the range of a variable.*

Description

By default, R's range function returns the minimum and maximum values of a variable. This returns a magnified range. It is used for some plotting functions in the rockchalk package

Usage

```
magRange(x, mult = 1.25)
```

Arguments

`x`	an R vector variable
`mult`	a multiplier by which to magnify the range of the variable. A value of 1 leaves the range unchanged. May be a scalar, in which case both ends of the range are magnified by the same amount. May also be a two valued vector, such as c(minMag, maxMag), in which case the magnification applied to the minimum is minMag and the magnification of the maximum is maxMag. After version 1.5.5, mult may be smaller than 1, thus shrinking the range. Setting mult to values closer to 0 causes the range to shrink to the center point from both sides.

Examples

```
x1 <- c(0, 0.5, 1.0)
range(x1)
magRange(x1, 1.1)
magRange(x1, c(1.1, 1.4))
magRange(x1, 0.5)
magRange(x1, c(0.1, 0.1))
x1 <- rnorm(100)
range(x1)
magRange(x1)
magRange(x1, 1.5)
magRange(x1, c(1,1.5))
```

`makeSymmetric` *Create Symmetric Matrices, possibly covariance or correlation matrices, or check a matrix for symmetry and serviceability.*

Description

Check X and do the right thing. If X is a matrix, check that it is a valid for the intended purpose (symmetric or correlation or covariance). If X a single value, use that to fill up a matrix. If it is a vector, try to use it as a vech to fill the lower triangle. If d is supplied as an integer, use that as desired size.

Usage

```
makeSymmetric(X, d = NULL, diag = NULL, corr = FALSE, cov = FALSE)
```

Arguments

`X`	A single value, a vector (a vech), or a matrix
`d`	Optional. An integer, the desired number of rows (or columns). Don't specify this argument if X is already a matrix. Only required if X is an integer and diag is not supplied. Otherwise, the function tries to deduce desired size of output from X (as a vech) and diag.
`diag`	Values for the diagonal. This is important because it alters the way X is interpreted. If diag is not provided, then X is understood to include diagonal elements.
`corr`	TRUE or FALSE: Should we construct a correlation matrix
`cov`	TRUE or FALSE: Should this be a covariance matrix?

Value

A d x d matrix

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
makeSymmetric(X = 3, d = 4)
makeSymmetric(X = 3, d = 4, diag = c(99, 98, 97, 96))
makeSymmetric(c(1,2,3))
makeSymmetric(c(1,2,3), d = 5)
makeSymmetric(c(0.8,0.4, 0.2), cov = TRUE)
makeSymmetric(c(0.8,0.4, 0.2), cov = TRUE, diag = c(44, 55, 66))
```

makeVec *makeVec for checking or creating vectors*

Description

This is a convenience for handling function arguments. If x is a single value, it makes a vector of length d in which all values are equal to x. If x is a vector, check that its length is d.

Usage

```
makeVec(x = NULL, d = NULL)
```

Arguments

x	A single value or a vector
d	An integer, the desired size of the vector

Value

A vector of length d

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

mcDiagnose *Multi-collinearity diagnostics*

Description

Conducts a series of checks for multicollinearity.

Usage

```
mcDiagnose(model)
```

Arguments

model	a fitted regression model

Value

a list of the "auxiliary regressions" that were fitted during the analysis

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
library(rockchalk)
N <- 100
dat <- genCorrelatedData(N=N, means=c(100,200), sds=c(20,30), rho=0.4, stde=10)
dat$x3 <- rnorm(100, m=40, s=4)
m1 <- lm(y ~ x1 + x2 + x3, data=dat)
summary(m1)
m1d <- mcDiagnose(m1)

m2 <- lm(y ~ x1 * x2 + x3, data=dat)
summary(m2)
m2d <- mcDiagnose(m2)



m3 <- lm(y ~ log(10+x1) + x3 + poly(x2,2), data=dat)
summary(m3)
m3d <- mcDiagnose(m3)

N <- 100
x1 <- 50 + rnorm(N)
x2 <- log(rgamma(N, 2,1))
x3 <- rpois(N, lambda=17)
z1 <- gl(5, N/5)
dummies <- contrasts(z1)[ as.numeric(z1), ]
dimnames(dummies) <- NULL ## Avoids row name conflict in data.frame below
y3 <- x1  -.5 * x2 + 0.1 * x2^2 + dummies %*% c(0.1,-0.1,-0.2,0.2)+ 5 * rnorm(N)
dat <- data.frame(x1=x1, x2=x2, x3=x3,  z1=z1, y3 = y3)

m3 <- lm(y3 ~ x1 + poly(x2,2)  + log(x1) + z1, dat)
summary(m3)

mcDiagnose(m3)
```

---

mcGraph1 *Illustrate the effect of multicollinearity in regression.*

Description

This is a set of functions that faciliates the examination of multicollinearity. Suppose the "true" relationship is y[i] = 0.2 * x1[i] + 0.2 * x2[i] + e where e is Normal(0, stde^2).

Usage

```
mcGraph1(x1, x2, y, x1lab, x2lab, ylab, ...)
```

Arguments

x1	a predictor vector
x2	a predictor vector
y	the dependent variable
x1lab	label for the x1 axis, (the one called "xlab" inside persp)
x2lab	label for the x2 axis, (the one called "ylab" inside persp)
ylab	label for the y (vertical) axis (the one called "zlab" inside persp)
...	additional parameters passed to persp

Value

The perspective matrix from persp (that can be used with trans3d to add more details in the plot)

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
set.seed(12345)
## Create data with x1 and x2 correlated at 0.10
dat <- genCorrelatedData(rho=.1, stde=7)

mcGraph1(dat$x1, dat$x2, dat$y, theta=20, phi=8, ticktype="detailed", nticks=10)
```

---

mcGraph2 *mcGraph2 draws a 3-D representation of a scatterplot with shadows in the x1-x2 plane.*

Description

The observations are represented by blue points floating above the x1-x2 plane. If scaley=1, the end result is a scatterplot "cloud" of the y points above the x1-x2 plane, and gray shadows of the points are cast down from the cloud onto the x1-x2 plane itself. This uses persp to make the actual drawing.

Usage

```
mcGraph2(x1, x2, y, rescaley = 1, drawArrows = TRUE, x1lab, x2lab, ylab,
  ...)
```

Arguments

x1	a predictor vector
x2	a predictor vector
y	the dependent variable
rescaley	a single scalar value or a vector of the same length as y.
drawArrows	TRUE or FALSE, do you want arrows from the bottom up to observed y?
x1lab	label for the x1 axis, (the one called "xlab" inside persp)
x2lab	label for the x2 axis, (the one called "ylab" inside persp)
ylab	label for the y (vertical) axis (the one called "zlab" inside persp)
...	arguments passed to persp

Value

The perspective matrix from persp (that can be used with trans3d to add more details in the plot)

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
set.seed(12345)
## Create data with x1 and x2 correlated at 0.10
dat <- genCorrelatedData(rho=.1, stde=7)
## This will "grow" the "cloud" of points up from the
## x1-x2 axis
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.0, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.1, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.2, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.3, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.4, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.5, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.6, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.7, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.8, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 0.9, theta = 0)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 1, theta = 0)

##rotate this
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 1, theta = 20)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 1, theta = 40)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 1, theta = 60)
mcGraph2(dat$x1, dat$x2, dat$y, rescaley = 1, theta = 80)

## once they reach the top, make them glitter a while
for(i in 1:20){
  mcGraph2(dat$x1, dat$x2, dat$y, rescaley = runif(length(dat$x1), .9,1.1), theta = 0)
}
```

mcGraph3 *mcGraph3 draws a 3-dimensional scatter and a regression plane*

Description

The observations are scattered in 3-dimensions, the fitted values are represented by a mesh, and their shadows in the x1-x2 plane are also represented.

Usage

```
mcGraph3(x1, x2, y, interaction = FALSE, drawArrows = TRUE, x1lab, x2lab,
  ylab, ...)
```

Arguments

x1	a predictor vector
x2	a predictor vector
y	the dependent variable
interaction	a TRUE or FALSE request for inclusion of the x1-x2 interaction in the regression calculation
drawArrows	TRUE or FALSE, do you want arrows from the plane to observed y?
x1lab	label for the x1 axis, (the one called "xlab" inside persp)
x2lab	label for the x2 axis, (the one called "ylab" inside persp)
ylab	label for the y (vertical) axis (the one called "zlab" inside persp)
...	optional arguments passed to persp

Value

a list of 2 objects, the fitted regression model and the perspective matrix used with persp to draw the image.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
set.seed(12345)
## Create data with x1 and x2 correlated at 0.10
dat <- genCorrelatedData(rho=.1, stde=7)

mcGraph3(dat$x1, dat$x2, dat$y, theta = 0)

dat2 <- genCorrelatedData(rho = 0, stde = 7)

mcGraph3(dat2$x1, dat2$x2, dat2$y, theta = 0, phi = 10)
```

```
mcGraph3(dat2$x1, dat2$x2, dat2$y, theta = 30, phi = 10)
mcGraph3(dat2$x1, dat2$x2, dat2$y, theta = -30, phi = 10)
mcGraph3(dat2$x1, dat2$x2, dat2$y, theta = -30, phi = -10)
mcGraph3(dat2$x1, dat2$x2, dat2$y, theta = -30, phi = -15)

## Run regressions with not-strongly correlated data
modset1 <- list()
for(i in 1:20){
  dat2 <- genCorrelatedData(rho = .1, stde = 7)
  summary(lm( y ~ x1 + x2 , data = dat2))
  modset1[[i]] <- mcGraph3(dat2$x1, dat2$x2, dat2$y, theta = -30)
}


## Run regressions with strongly correlated data
modset2 <- list()
for(i in 1:20){
  dat2 <- genCorrelatedData(rho = .981, stde = 7)
  summary(lm( y ~ x1 + x2 , data = dat2))
  modset2[[i]] <- mcGraph3(dat2$x1, dat2$x2, dat2$y, theta = -30)
}

dat3 <- genCorrelatedData(rho = .981, stde = 100, beta=c(0.1, 0.2, 0.3, -0.1))
mcGraph3(dat3$x1, dat3$x2, dat3$y, theta=-10, interaction = TRUE)
```

`meanCenter` *meanCenter*

Description

meanCenter selectively centers or standarizes variables in a regression model.

Usage

```
meanCenter(model, centerOnlyInteractors = TRUE, centerDV = FALSE,
  standardize = FALSE, terms = NULL)

## Default S3 method:
meanCenter(model, centerOnlyInteractors = TRUE,
  centerDV = FALSE, standardize = FALSE, terms = NULL)
```

Arguments

`model`	a fitted regression model (presumably from lm)
`centerOnlyInteractors`	Default TRUE. If FALSE, all numeric predictors in the regression data frame are centered before the regression is conducted.

`centerDV` Default FALSE. Should the dependent variable be centered? Do not set this option to TRUE unless the dependent variable is a numeric variable. Otherwise, it is an error.

`standardize` Default FALSE. Instead of simply mean-centering the variables, should they also be "standardized" by first mean-centering and then dividing by the estimated standard deviation.

`terms` Optional. A vector of variable names to be centered. Supplying this argument will stop meanCenter from searching for interaction terms that might need to be centered.

Details

Works with "lm" class objects, objects estimated by `glm()`. This centers some or all of the the predictors and then re-fits the original model with the new variables. This is a convenience to researchers who are often urged to center their predictors. This is sometimes suggested as a way to ameliorate multi-collinearity in models that include interaction terms (Aiken and West, 1991; Cohen, et al 2002). Mean-centering may enhance interpretation of the regression intercept, but it actually does not help with multicollinearity. (Echambadi and Hess, 2007). This function facilitates comparison of mean-centered models with others by calculating centered variables. The defaults will cause a regression's numeric interactive variables to be mean centered. Variations on the arguments are discussed in details.

Suppose the user's formula that fits the original model is `m1 <- lm(y ~ x1*x2 + x3 + x4, data = dat)`. The fitted model will include estimates for predictors `x1`, `x2`, `x1:x2`, `x3` and `x4`. By default, `meanCenter(m1)` scans the output to see if there are interaction terms of the form `x1:x2`. If so, then x1 and x2 are replaced by centered versions (m1-mean(m1)) and (m2-mean(m2)). The model is re-estimated with those new variables. model (the main effect and the interaction). The resulting thing is "just another regression model", which can be analyzed or plotted like any R regression object.

The user can claim control over which variables are centered in several ways. Most directly, by specifying a vector of variable names, the user can claim direct control. For example, the argument `terms=c("x1","x2","x3")` would cause 3 predictors to be centered. If one wants all predictors to be centered, the argument `centerOnlyInteractors` should be set to FALSE. Please note, this WILL NOT center factor variables. But it will find all numeric predictors and center them.

The dependent variable will not be centered, unless the user explicitly requests it by setting centerDV = TRUE.

As an additional convenience to the user, the argument `standardize = TRUE` can be used. This will divide each centered variable by its observed standard deviation. For people who like standardized regression, I suggest this is a better approach than the `standardize` function (which is brain-dead in the style of SPSS). meanCenter with `standardize = TRUE` will only try to standardize the numeric predictors.

To be completely clear, I believe mean-centering is not helpful with the multicollinearity problem. It doesn't help, it doesn't hurt. Only a misunderstanding leads its proponents to claim otherwise. This is emphasized in the vignette "rockchalk" that is distributed with this package.

Value

A regression model of the same type as the input model, with attributes representing the names of the centered variables.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

References

Aiken, L. S. and West, S.G. (1991). Multiple Regression: Testing and Interpreting Interactions. Newbury Park, Calif: Sage Publications.

Cohen, J., Cohen, P., West, S. G., and Aiken, L. S. (2002). Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences (Third.). Routledge Academic.

Echambadi, R., and Hess, J. D. (2007). Mean-Centering Does Not Alleviate Collinearity Problems in Moderated Multiple Regression Models. Marketing Science, 26(3), 438-445.

See Also

lmres standardize residualCenter

Examples

```
library(rockchalk)
N <- 100
dat <- genCorrelatedData(N = N, means = c(100, 200), sds = c(20, 30),
                         rho = 0.4, stde = 10)
dat$x3 <- rnorm(100, m = 40, s = 4)

m1 <- lm(y ~ x1 * x2 + x3, data = dat)
summary(m1)
mcDiagnose(m1)

m1c <- meanCenter(m1)
summary(m1c)
mcDiagnose(m1c)

m2 <- lm(y ~ x1 * x2 + x3, data = dat)
summary(m2)
mcDiagnose(m2)

m2c <- meanCenter(m2, standardize = TRUE)
summary(m2c)
mcDiagnose(m2c)

m2c2 <- meanCenter(m2, centerOnlyInteractors = FALSE)
summary(m2c2)

m2c3 <- meanCenter(m2, centerOnlyInteractors = FALSE, centerDV = TRUE)
```

```
summary(m2c3)

dat <- genCorrelatedData(N = N, means = c(100, 200), sds = c(20, 30),
                         rho = 0.4, stde = 10)
dat$x3 <- rnorm(100, m = 40, s = 4)
dat$x3 <- gl(4, 25, labels = c("none", "some", "much", "total"))

m3 <- lm(y ~ x1 * x2 + x3, data = dat)
summary(m3)
## visualize, for fun
plotPlane(m3, "x1", "x2")

m3c1 <- meanCenter(m3)
summary(m3c1)

## Not exactly the same as a "standardized" regression because the
## interactive variables are centered in the model frame,
## and the term "x1:x2" is never centered again.
m3c2 <- meanCenter(m3, centerDV = TRUE,
                   centerOnlyInteractors = FALSE, standardize = TRUE)
summary(m3c2)

m3st <- standardize(m3)
summary(m3st)

## Make a bigger dataset to see effects better
N <- 500
dat <- genCorrelatedData(N = N, means = c(200,200), sds = c(60,30),
                         rho = 0.2, stde = 10)
dat$x3 <- rnorm(100, m = 40, s = 4)
dat$x3 <- gl(4, 25, labels = c("none", "some", "much", "total"))
dat$y2 <- with(dat,
               0.4 - 0.15 * x1 + 0.04 * x1^2 -
               drop(contrasts(dat$x3)[dat$x3, ] %*% c(-1.9, 0, 5.1))  +
               1000* rnorm(nrow(dat)))
dat$y2 <- drop(dat$y2)

m4literal <- lm(y2 ~ x1 + I(x1*x1) + x2 + x3, data = dat)
summary(m4literal)
plotCurves(m4literal, plotx="x1")
## Superficially, there is multicollinearity (omit the intercept)
cor(model.matrix(m4literal)[ -1 , -1 ])

m4literalmc <- meanCenter(m4literal, terms = "x1")
summary(m4literalmc)

m4literalmcs <- meanCenter(m4literal, terms = "x1", standardize = TRUE)
summary(m4literalmcs)

m4 <- lm(y2 ~ poly(x1, 2, raw = TRUE) + x2 + x3, data = dat)
summary(m4)
plotCurves(m4, plotx="x1")
```

```
m4mc1 <- meanCenter(m4, terms = "x1")
summary(m4mc1)

m4mc2 <- meanCenter(m4, terms = "x1", standardize = TRUE)
summary(m4mc2)

m4mc3 <- meanCenter(m4, terms = "x1", centerDV = TRUE, standardize = TRUE)
summary(m4mc3)
```

`model.data` *Create a "raw" (UNTRANSFORMED) data frame equivalent to the input data that would be required to fit the given model.*

Description

This is a generic method. Unlike model.frame and model.matrix, this does not return transformed variables. It deals with regression formulae that have functions like poly(x, d) in them. It differentiates x from d in those expressions. And it also manages log(x + 10). The default method works for standarad R regression models like lm and glm.

Usage

```
model.data(model, ...)
```

Arguments

`model`	A fitted regression model in which the data argument is specified. This function will fail if the model was not fit with the data option.
`...`	Arguments passed to implementing methods.

Value

A data frame

Author(s)

Paul Johnson <pauljohn@ku.edu>

`model.data.default` *Create a data frame suitable for estimating a model*

Description

This is the default method. Works for lm and glm fits.

Usage

```
## Default S3 method:
model.data(model, na.action = na.omit, ...)
```

Arguments

`model`	A fitted model
`na.action`	Defaults to na.omit, so model as it would appear in user workspace is re-created, except that rows with missing values are deleted. Changing this argument to na.pass will provide the data as it was in the workspace.
`...`	Place holder for other arguments, not used at present

Value

A data frame

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
library(rockchalk)
```

```
## first, check if model.data works when there is no data argument
## This used to fail, now OK

x1 <- rnorm(100, m = 100, s = 10)
x2 <- rnorm(100, m = 50, s =20)
y <- rnorm(100, m = 40, s = 3)

m0 <- lm(y ~ log(10+x1) + x2)
m0.data <- model.data(m0)
head(m0.data)
```

```
m1 <- lm(log(43 + y) ~ log(10+x1) + x2)
m1.data <- model.data(m1)
head(m1.data)

d <- 3

m2 <- lm(log(d + y) ~ log(10+x1) + x2)
m2.data <- model.data(m2)
head(m2.data)

m3 <- lm(log(y + d) ~ log(10+x1) + x2)
m3.data <- model.data(m3)
head(m3.data)



## check numeric and categorical predictors

x1 <- rpois(100, l=6)
x2 <- rnorm(100, m=50, s=10)
x3 <- rnorm(100)
xcat1 <- gl(2,50, labels=c("M","F"))
xcat2 <- cut(rnorm(100), breaks=c(-Inf, 0, 0.4, 0.9, 1, Inf),
             labels=c("R", "M", "D", "P", "G"))
dat <- data.frame(x1, x2, x3, xcat1, xcat2)
rm(x1, x2, x3, xcat1, xcat2)
dat$xcat1n <- with(dat, contrasts(xcat1)[xcat1, ,drop=FALSE])
dat$xcat2n <- with(dat, contrasts(xcat2)[xcat2, ])

STDE <- 20
dat$y <- with(dat,
              0.03 + 0.8*x1 + 0.1*x2 + 0.7*x3 +
              xcat1n %*% c(2) + xcat2n %*% c(0.1,-2,0.3, 0.1) +
              STDE*rnorm(100))



m1 <- lm(y ~ poly(x1, 2), data=dat)
m1.data <- model.data(m1)
head(m1.data)
attr(m1.data, "varNamesRHS")

## Check to make sure d is not mistaken for a data column
d <- 2
m2 <- lm(y ~ poly(x1, d), data=dat)
m2.data <- model.data(m2)
head(m2.data)
attr(m2.data, "varNamesRHS")


## Check to see how the 10 in log is handled
m3 <- lm(y ~ log(10 + x1) + poly(x1, d) + sin(x2), data=dat)
```

```
m3.data <- model.data(m3)
head(m3.data)
attr(m3.data, "varNamesRHS")


m4 <- lm(log(50+y) ~ log(d+10+x1) + poly(x1, 2), data=dat)
m4.data <- model.data(m4)
head(m4.data)
attr(m4.data, "varNamesRHS")


m5 <- lm(y ~ x1*x1, data=dat)
m5.data <- model.data(m5)
head(m5.data)
attr(m5.data, "varNamesRHS")


m6 <- lm(y ~ x1 + I(x1^2), data=dat)
m6.data <- model.data(m6)
head(m6.data)
attr(m6.data, "varNamesRHS")


## Put in some missings.
## poly doesn't work if there are missings, but
## can test with log
dat$x1[sample(100, 5)] <- NA
dat$y[sample(100, 5)] <- NA
dat$x2[sample(100, 5)] <- NA
dat$x3[sample(100,10)] <- NA

m1 <- lm(y ~ log(10 + x1), data=dat)
m1.data <- model.data(m1)
head(m1.data)
summarize(m1.data)
attr(m1.data, "varNamesRHS")



m2 <- lm(y ~ log(x1 + 10), data=dat)
m2.data <- model.data(m2)
head(m2.data)
summarize(m1.data)
attr(m1.data, "varNamesRHS")

d <- 2
m3 <- lm(log(50+y) ~ log(d+10+x1) + x2 + sin(x3), data=dat)
m3.data <- model.data(m3)
head(m3.data)
summarize(m3.data)
attr(m3.data, "varNamesRHS")
```

```
m4 <- lm(y ~ I(x1) + I(x1^2) + log(x2), data=dat)
m4.data <- model.data(m4)
summarize(m4.data)
attr(m4.data, "varNamesRHS")


m5 <- lm(y ~ x1 + I(x1^2) + cos(x2), data=dat)
m5.data <- model.data(m5)
head(m5.data)
summarize(m5.data)
attr(m5.data, "varNamesRHS")



## Now try with some variables in the dataframe, some not

x10 <- rnorm(100)
x11 <- rnorm(100)



m6 <- lm(y ~ x1 + I(x1^2) + cos(x2) + log(10 + x10) + sin(x11) +
         x10*x11, data = dat)
m6.data <- model.data(m6)
head(m6.data)
dim(m6.data)
summarize(m5.data)
attr(m6.data, "varNamesRHS")
```

`mvrnorm` *Minor revision of mvrnorm (from* `MASS`*) to facilitate replication*

Description

This is the `mvrnorm` function from the MASS package (Venables and Ripley, 2002), with one small modification to facilitate replication of random samples. The aim is to make sure that, after the seed is reset, the first rows of generated data are identical no matter what value is chosen for n. The one can draw 100 observations, reset the seed, and then draw 110 observations, and the first 100 will match exactly. This is done to prevent unexpected and peculiar patterns that are observed when n is altered with MASS package's mvrnorm.

Usage

```
mvrnorm(n = 1, mu, Sigma, tol = 1e-06, empirical = FALSE,
  EISPACK = FALSE)
```

Arguments

n	the number of samples ("rows" of data) required.
mu	a vector giving the means of the variables.
Sigma	positive-definite symmetric matrix specifying the covariance matrix of the variables.
tol	tolerance (relative to largest variance) for numerical lack of positive-definiteness in Sigma
empirical	logical. If true, mu and Sigma specify the empirical not population mean and covariance matrix.
EISPACK	logical. Set to true to reproduce results from MASS versions prior to 3.1-21.

Details

To assure replication, only a very small change is made. The code in MASS::mvrnorm draws a random sample and fills a matrix by column, and that matrix is then decomposed. The change implemented here fills that matrix by row and the problem is eliminated.

Some peculiarities are noticed when the covariance matrix changes from a diagonal matrix to a more general symmetric matrix (non-zero elements off-diagonal). When the covariance is strictly diagonal, then just one column of the simulated multivariate normal data will be replicated, but the others are not. This has very troublesome implications for simulations that draw samples of various sizes and then base calculations on the separate simulated columns (i.e., some columns are identical, others are completely uncorrelated).

Value

If n = 1 a vector of the same length as mu, otherwise an n by length(mu) matrix with one sample in each row.

Author(s)

Ripley, B.D. with revision by Paul E. Johnson

References

Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

See Also

For an alternative multivariate normal generator function, one which has had this fix applied to it, consider using the new versions of rmvnorm in the package mvtnorm.

Examples

```
library(MASS)
library(rockchalk)

set.seed(12345)
```

```
X0 <- MASS::mvrnorm(n=10, mu = c(0,0,0), Sigma = diag(3))
## create a smaller data set, starting at same position
set.seed(12345)
X1 <- MASS::mvrnorm(n=5, mu = c(0,0,0), Sigma = diag(3))
## Create a larger data set
set.seed(12345)
X2 <- MASS::mvrnorm(n=15, mu = c(0,0,0), Sigma = diag(3))
## The first 5 rows in X0, X1, and X2 are not the same
X0
X1
X2
set.seed(12345)
Y0 <- mvrnorm(n=10, mu = c(0,0,0), Sigma = diag(3))
set.seed(12345)
Y1 <- mvrnorm(n=5, mu = c(0,0,0), Sigma = diag(3))
set.seed(12345)
Y2 <- mvrnorm(n=15, mu = c(0,0,0), Sigma = diag(3))
# note results are the same in the first 5 rows:
Y0
Y1
Y2
identical(Y0[1:5, ], Y1[1:5, ])
identical(Y1[1:5, ], Y2[1:5, ])

myR <- lazyCor(X = 0.3, d = 5)
mySD <- c(0.5, 0.5, 0.5, 1.5, 1.5)
myCov <- lazyCov(Rho = myR, Sd = mySD)

set.seed(12345)
X0 <- MASS::mvrnorm(n=10, mu = rep(0, 5), Sigma = myCov)
## create a smaller data set, starting at same position
set.seed(12345)
X1 <- MASS::mvrnorm(n=5, mu = rep(0, 5), Sigma = myCov)
X0
X1
##' set.seed(12345)
Y0 <- rockchalk::mvrnorm(n=10, mu = rep(0, 5), Sigma = myCov)
## create a smaller data set, starting at same position
set.seed(12345)
Y1 <- rockchalk::mvrnorm(n=5, mu = rep(0, 5), Sigma = myCov)
Y0
Y1
```

`newdata` *Create a newdata frame for usage in predict methods*

Description

This is a generic function. The default method covers almost all regression models.

Creates a new data frame from which a model could be re-estimated.

Usage

```
newdata(model, predVals, n, ...)

## Default S3 method:
newdata(model = NULL, predVals = NULL, n = 3,
  emf = NULL, divider = "quantile", ...)
```

Arguments

model	Required. Fitted regression model
predVals	Predictor Values that deserve investigation. Previously, the argument was called "fl". This can be 1) a keyword, one of c("auto", "margins") 2) a vector of variable names, which will use default methods for all named variables and the central values for non-named variabled, 3) a named vector with predictor variables and divider algorithms, or 4) a full list that supplies variables and possible values. Please see details and examples.
n	Optional. Default = 3. How many focal values are desired? This value is used when various divider algorithms are put to use if the user has specified keywords "default", "quantile", "std.dev." "seq", and "table".
...	Other arguments.
divider	Default is "quantile". Determines the method of selection. Should be one of c("quantile", "std.dev", "seq", "table").
emf	Optional. data frame used to fit model (not a model frame, which may include transformed variables like log(x1). Instead, use output from function model.data). It is UNTRANSFORMED variables ("x" as opposed to poly(x,2).1 and poly(x,2).2).

Details

It scans the fitted model, discerns the names of the predictors, and then generates a new data frame. It can guess values of the variables that might be substantively interesting, but that depends on the user-supplied value of predVals. If not supplied with a predVals argument, newdata returns a data frame with one row – the central values (means and modes) of the variables in the data frame that was used to fit the model. The user can supply a keyword "auto" or "margins". The function will try to do the "right thing."

The predVals can be a named list that supplies specific values for particular predictors. Any legal vector of values is allowed. For example, `predVals = list(x1 = c(10, 20, 30), x2 = c(40, 50), xcat = levels(xcat)))`. That will create a newdata object that has all of the "mix and match" combinations for those values, while the other predictors are set at their central values.

If the user declares a variable with the "default" keyword, then the default divider algorithm is used to select focal values. The default divider algorithm is an optional argument of this function. If the default is not desired, the user can specify a divider algorithm by character string, either "quantile", "std.dev.", "seq", or "table". The user can mix and match algorithms along with requests for specific focal values, as in `predVals = list(x1 = "quantile", x2 = "std.dev.", x3 = c(10,20, 30), xcat1 <- levels(xca`

Value

A data frame of x values that could be used as the data = argument in the original regression model. The attribute "varNamesRHS" is a vector of the predictor variable names.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

See Also

predictOMatic

Examples

```
library(rockchalk)

## Replicate some R classics.  The budworm.lg data from predict.glm
## will work properly after re-formatting the information as a data.frame:

## example from Venables and Ripley (2002, pp. 190-2.)
df <- data.frame(ldose = rep(0:5, 2),
                 sex = factor(rep(c("M", "F"), c(6, 6))),
                 SF.numdead = c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16))
df$SF.numalive = 20 - df$SF.numdead

budworm.lg <- glm(cbind(SF.numdead, SF.numalive) ~ sex*ldose,
                  data = df,  family = binomial)

predictOMatic(budworm.lg)

predictOMatic(budworm.lg, n = 7)

predictOMatic(budworm.lg, predVals = c("ldose"), n = 7)

predictOMatic(budworm.lg, predVals = c(ldose = "std.dev.", sex = "table"))



## Now make up a data frame with several numeric and categorical predictors.

set.seed(12345)
N <- 100
x1 <- rpois(N, l = 6)
x2 <- rnorm(N, m = 50, s = 10)
x3 <- rnorm(N)
xcat1 <- gl(2,50, labels = c("M","F"))
xcat2 <- cut(rnorm(N), breaks = c(-Inf, 0, 0.4, 0.9, 1, Inf),
             labels = c("R", "M", "D", "P", "G"))
dat <- data.frame(x1, x2, x3, xcat1, xcat2)
rm(x1, x2, x3, xcat1, xcat2)
dat$xcat1n <- with(dat, contrasts(xcat1)[xcat1, , drop = FALSE])
dat$xcat2n <- with(dat, contrasts(xcat2)[xcat2, ])
```

```
STDE <- 15
dat$y <- with(dat,
              0.03 + 0.8*x1 + 0.1*x2 + 0.7*x3 + xcat1n %*% c(2) +
              xcat2n %*% c(0.1,-2,0.3, 0.1) + STDE*rnorm(N))
## Impose some random missings
dat$x1[sample(N, 5)] <- NA
dat$x2[sample(N, 5)] <- NA
dat$x3[sample(N, 5)] <- NA
dat$xcat2[sample(N, 5)] <- NA
dat$xcat1[sample(N, 5)] <- NA
dat$y[sample(N, 5)] <- NA
summarize(dat)


m0 <- lm(y ~ x1 + x2 + xcat1, data = dat)
summary(m0)
## The model.data() function in rockchalk creates as near as possible
## the input data frame.
m0.data <- model.data(m0)
summarize(m0.data)

## no predVals: analyzes each variable separately
(m0.p1 <- predictOMatic(m0))

## requests confidence intervals from the predict function
(m0.p2 <- predictOMatic(m0, interval = "confidence"))

## predVals as vector of variable names: gives "mix and match" predictions
(m0.p3 <- predictOMatic(m0, predVals = c("x1", "x2")))

## predVals as vector of variable names: gives "mix and match" predictions
(m0.p3s <- predictOMatic(m0, predVals = c("x1", "x2"), divider = "std.dev."))

## "seq" is an evenly spaced sequence across the predictor.
(m0.p3q <- predictOMatic(m0, predVals = c("x1", "x2"), divider = "seq"))

(m0.p3i <- predictOMatic(m0, predVals = c("x1", "x2"),
                         interval = "confidence", n = 3))

(m0.p3p <- predictOMatic(m0, predVals = c("x1", "x2"), divider = pretty))

## predVals as vector with named divider algorithms.
(m0.p3 <- predictOMatic(m0, predVals = c(x1 = "seq", x2 = "quantile")))
## predVals as named vector of divider algorithms

## same idea, decided to double-check
(m0.p3 <- predictOMatic(m0, predVals = c(x1 = "quantile", x2 = "std.dev.")))
getFocal(m0.data$x2, xvals =  "std.dev.", n = 5)


## Change from quantile to standard deviation divider
(m0.p5 <- predictOMatic(m0, divider = "std.dev.", n = 5))
```

```
## Still can specify particular values if desired
(m0.p6 <- predictOMatic(m0, predVals = list("x1" = c(6,7),
                            "xcat1" = levels(m0.data$xcat1))))

(m0.p7 <- predictOMatic(m0, predVals = c(x1 = "quantile", x2 = "std.dev.")))
getFocal(m0.data$x2, xvals =  "std.dev.", n = 5)

(m0.p8 <- predictOMatic(m0, predVals = list( x1 = quantile(m0.data$x1,
                        na.rm = TRUE, probs = c(0, 0.1, 0.5, 0.8,
                        1.0)), xcat1 = levels(m0.data$xcat1))))

(m0.p9 <- predictOMatic(m0, predVals = list(x1 = "seq", "xcat1" =
                                levels(m0.data$xcat1)), n = 8) )


(m0.p10 <- predictOMatic(m0, predVals = list(x1 = "quantile",
                                 "xcat1" = levels(m0.data$xcat1)), n =  5) )


(m0.p11 <- predictOMatic(m0, predVals = c(x1 = "std.dev."), n = 10))

## Previous same as

(m0.p11 <- predictOMatic(m0, predVals = c(x1 = "default"), divider =
                         "std.dev.", n = 10))

## Previous also same as

(m0.p11 <- predictOMatic(m0, predVals = c("x1"), divider = "std.dev.", n = 10))


(m0.p11 <- predictOMatic(m0,  predVals = list(x1 = c(0, 5, 8), x2 = "default"),
                         divider = "seq"))



m1 <- lm(y ~ log(10+x1) + sin(x2) + x3, data = dat)
m1.data <- model.data(m1)
summarize(m1.data)


(newdata(m1))
(newdata(m1, predVals = list(x1 = c(6, 8, 10))))
(newdata(m1, predVals = list(x1 = c(6, 8, 10), x3 = c(-1,0,1))))
(newdata(m1, predVals = list(x1 = c(6, 8, 10),
                x2 = quantile(m1.data$x2, na.rm = TRUE), x3 = c(-1,0,1))))

(m1.p1 <- predictOMatic(m1, divider = "std.dev", n = 5))
(m1.p2 <- predictOMatic(m1, divider = "quantile", n = 5))

(m1.p3 <- predictOMatic(m1, predVals = list(x1 = c(6, 8, 10),
                            x2 = median(m1.data$x2, na.rm = TRUE))))
```

```
(m1.p4 <- predictOMatic(m1, predVals = list(x1 = c(6, 8, 10),
                                x2 = quantile(m1.data$x2, na.rm = TRUE))))

(m1.p5 <- predictOMatic(m1))
(m1.p6 <- predictOMatic(m1, divider = "std.dev."))
(m1.p7 <- predictOMatic(m1, divider = "std.dev.", n = 3))
(m1.p8 <- predictOMatic(m1, divider = "std.dev.", interval = "confidence"))


m2 <- lm(y ~ x1 + x2 + x3 + xcat1 + xcat2, data = dat)
##  has only columns and rows used in model fit
m2.data <- model.data(m2)
summarize(m2.data)

## Check all the margins
(predictOMatic(m2, interval = "conf"))

## Lets construct predictions the "old fashioned way" for comparison

m2.new1 <- newdata(m2, predVals = list(xcat1 = levels(m2.data$xcat1),
                           xcat2 = levels(m2.data$xcat2)), n = 5)
predict(m2, newdata = m2.new1)


(m2.p1 <- predictOMatic(m2,
                        predVals = list(xcat1 = levels(m2.data$xcat1),
                            xcat2 = levels(m2.data$xcat2)),
                        xcat2 = c("M","D")))
## See? same!

## Pick some particular values for focus
m2.new2 <- newdata(m2, predVals = list(x1 = c(1,2,3), xcat2 = c("M","D")))
## Ask for predictions
predict(m2, newdata = m2.new2)


## Compare: predictOMatic generates a newdata frame and predictions in one step

(m2.p2 <- predictOMatic(m2, predVals = list(x1 = c(1,2,3),
                                xcat2 = c("M","D"))))

(m2.p3 <- predictOMatic(m2, predVals = list(x2 = c(0.25, 1.0),
                                xcat2 = c("M","D"))))

(m2.p4 <- predictOMatic(m2, predVals = list(x2 = plotSeq(m2.data$x2, 10),
                                xcat2 = c("M","D"))))

(m2.p5 <- predictOMatic(m2, predVals = list(x2 = c(0.25, 1.0),
                                xcat2 = c("M","D")), interval = "conf"))

(m2.p6 <- predictOMatic(m2, predVals = list(x2 = c(49, 51),
                                xcat2 = levels(m2.data$xcat2),
                                x1 = plotSeq(dat$x1))))
```

```
plot(y ~ x1, data = m2.data)
by(m2.p6, list(m2.p6$xcat2), function(x) {
    lines(x$x1, x$fit, col = x$xcat2, lty = as.numeric(x$xcat2))
})

m2.newdata <- newdata(m2, predVals = list(x2 = c(48, 50, 52),
                              xcat2 = c("M","D")))
predict(m2, newdata = m2.newdata)

(m2.p7 <- predictOMatic(m2, predVals = list(x2 = c(48, 50, 52),
                                xcat2 = c("M","D"))))

(m2.p8 <- predictOMatic(m2,
             predVals = list(x2 = range(m2.data$x2, na.rm = TRUE),
             xcat2 = c("M","D"))))

(m2.p9 <- predictOMatic(m2, predVals = list(x2 = plotSeq(m2.data$x2),
             x1 = quantile(m2.data$x1, pr =c(0.33, 0.66), na.rm = TRUE),
             xcat2 = c("M","D"))))
plot(y ~ x2 , data = m2.data)

by(m2.p9, list(m2.p9$x1, m2.p9$xcat2), function(x) {lines(x$x2, x$fit)})



(predictOMatic(m2, predVals = list(x2 = c(50, 60), xcat2 = c("M","D")),
               interval = "conf"))

## create a dichotomous dependent variable
y2 <- ifelse(rnorm(N) > 0.3, 1, 0)
dat <- cbind(dat, y2)

m3 <- glm(y2 ~ x1 + x2 + x3 + xcat1, data = dat, family = binomial(logit))
summary(m3)
m3.data <- model.data(m3)
summarize(m3.data)

(m3.p1 <- predictOMatic(m3, divider = "std.dev."))

(m3.p2 <- predictOMatic(m3, predVals = list(x2 = c(40, 50, 60),
                             xcat1 = c("M","F")),
                        divider = "std.dev.", interval = "conf"))

## Want a full accounting for each value of x2?
(m3.p3 <- predictOMatic(m3,
                predVals = list(x2 = unique(m3.data$x2),
                    xcat1 = c("M","F")), interval = "conf"))


## Would like to write a more beautiful print method
## for output object, but don't want to obscure structure from user.
## for (i in names(m3.p1)){
```

```
##     dns <- cbind(m3.p1[[i]][i], m3.p1[[i]]$fit)
##     colnames(dns) <- c(i, "predicted")
##     print(dns)
## }
```

---

outreg *Creates a publication quality result table for regression models. Works with models fitted with lm, glm, as well as lme4.*

Description

This provides "markup" that the user is will copy into a LaTeX document. As of rockchalk 1.8.4, can also create HTML markup. The rockchalk vignette demonstrates use of outreg in Sweave.

Usage

```
outreg(modelList, type = "latex", modelLabels = NULL, varLabels = NULL,
  tight = TRUE, showAIC = FALSE, float = FALSE, request, runFuns,
  digits = 3, alpha = c(0.05, 0.01, 0.001), SElist = NULL,
  PVlist = NULL, Blist = NULL, title, label, gofNames,
  browser = identical(type, "html"))
```

Arguments

modelList	A regression model or an R list of regression models. Default model names will be M1, M2, and so forth. User specified names are allowed, such as `list("My Model" = m1,"Her Model" = m2)`. This is the currently recommended way to supply model lables. This is less error prone than the use of the modelLabels argument.
type	Default = "latex". The alternative is "html"
modelLabels	This is allowed, but discouraged. A vector of character string variables, one for each element in modelList. Will override the names in modelList.
varLabels	To beautify the parameter names printed. Must be a named vector in the format c(parmname = "displayName", parmname = "displayName"). Include as many parameters as desired, it is not necessary to supply new labels for all of the parameters.
tight	Table format. If TRUE, parameter estimates and standard errors are printed in a single column. If FALSE, parameter estimates and standard errors are printed side by side.
showAIC	This is a legacy argument, before the `request` argument was created. If TRUE, the AIC estimate is included with the diagnostic values. It has the same effect as described by `request`.
float	Default = FALSE. Include boilerplate for a LaTeX table float, with the tabular markup inside it. Not relevant if type = "html".

request	Extra information to be retrieved from the summary(model) and displayed. This must be a vector of named arguments, such as c(adj.r.squared = "adj R^2", fstatistic = "F"). The name must be a valid name of the output object, the value should be the label the user wants printed in the table. See details.
runFuns	A list of functions
digits	Default = 3. How many digits after decimal sign are to be displayed.
alpha	Default = c(0.05, 0.01, 0.001). I think stars are dumb, but enough people have asked me for more stars that I'm caving in.
SElist	Optional. Replacement standard errors. Must be a list of named vectors. outreg uses the R summary to retrieve standard errors, but one might instead want to use robust or bootstrapped standard errors. This argument may supply a new SE vector for each fitted regression model, but it is also allowed to supply the SE replacement for just one of the models. The format should be list("A Model Label" = c(0.1, 0.3, 0.4 On the left, one must use the same names that are used in the modelList argument.
PVlist	Optional. A list of replacement "p values". It must be a list of named vectors, similar in format to SElist. The which the elements are the "p values" that the user wants to use for each model.
Blist	Optional. This is only needed in the rare case where a model's parameters cannot be discerned from its summary. List must have names for models, and vectors slope coefficient. See discussion of SElist and PVlist.
title	A LaTeX caption for the table. Not relevant if type = "html".
label	A string to be used as a LaTeX label in the table to be created. Not relevant if type = "html".
gofNames	Optional pretty names. R regression summaries use names like "sigma" or "r.squared" that we might want to revise for presentation. I prefer to refer to "sigma" as "RMSE", but perhaps you instead prefer something like gofnames = c("sigma" = "That Estimate I don't understand", "deviance" = "Another Mystery"). The words that you might replace are "sigma", "r.squared", "deviance", "adj.r.squared", "fstatistic".
browser	Display the regression model in a browser? Defaults to TRUE if type = "html"

Details

outreg returns a string vector. It is suggested that users should save the outreg result and then use cat to save it. That is myMod <- outreg(m1, ...) cat(myMod, file = "myMod.html") or cat(myMod, file = "myMod.tex". In version 1.8.66, we write the html file to a temporary location and display it in a web browser. Many word processors will not accept a cut-and paste transfer from the browser, they will, however, be able to open the html file itself and automatically re-format it in the native table format.

The table includes a minimally sufficient (in my opinion) model summary. It offers parameter estimates, standard errors, and minimally sufficient goodness of fit. My tastes tend toward minimal tables, but users request more features, and outreg's interface hass been generalized to allow specialized requests. See request and runFuns arguments.

I don't want to write a separate table function for every different kind of regression model that exists (how exhausting). So I've tried to revise outreg() to work with regression functions that follow

the standard R framework. It is known to work `lm` and `glm`, as well as `merMod` class from `lme4`, but it will try to interact with other kinds of regression models. Those models should have methods `summary()`, `coef()`, `vcov()` and `nobs()`. Package writes should provide those, its not my job.

Do you want "robust standard errors"? P values calculated according to some alternative logic? Go ahead, calculate them in your code, outreg will now accept them as arguments. As of Version 1.8.4, users can provide their own standard errors and/or p-values for each model. Thus, if a model answers in the usual way to the standard R request `coef(summary(model))`, outreg can work if users supply standard errors.

About the customizations `request`. The `request` argument supplies a list of names of summary output elements that are desired. The format is a pair, a value to be retrieved from `summary(model)`, and a pretty name to be printed for it. With the `lm()` regression, for example, one might want the output of the F test and the adjusted R-square: Include `request = c(adj.r.squared = "adj. $R^2$", "fstatistic" = "F")`. The value on the left is the name of the desired information in the summary object, while the value on the right is *any* valid LaTeX (or HTML) markup that the user desires to display in the table. `request` terms that generate a single numerical value will generally work fine, while requests that ask for more structured information, such as the F test (including the 2 degrees of freedom values) may work (user feedback needed).

The `runFuns` argument is inspired by a user request: could this include the BIC or other summaries that can be easily calculated? Any R function, such as `AIC` or `BIC`, should work, as long as it returns a single value. This is a two-part specification, a function name and a pretty label to be used in printing. For example, `runFuns = c("AIC" = "Akaike Criterion", "BIC" = "Schwartz Criterion", "logLik" = "LL")`.

Value

A character vector, one element per row of the regression table.

Note

There are many R packages that can be used to create LaTeX regression tables. memisc, texreg, apsrtable, xtables, and rms are some. This "outreg" version was in use in our labs before we were aware that those packages were in development. It is not intended as a competitor, it is just a slightly different version of the same that is more suited to our needs.

Author(s)

Paul E. Johnson <<pauljohn@ku.edu>>

Examples

```
set.seed(2134234)
dat <- data.frame(x1 = rnorm(100), x2 = rnorm(100))
dat$y1 <- 30 + 5 * rnorm(100) + 3 * dat$x1 + 4 * dat$x2
dat$y2 <- rnorm(100) + 5 * dat$x2
m1 <- lm(y1 ~ x1, data = dat)
m2 <- lm(y1 ~ x2, data = dat)
m3 <- lm(y1 ~ x1 + x2, data = dat)
gm1 <- glm(y1 ~ x1, family = Gamma, data = dat)
outreg(m1, title = "My One Tightly Printed Regression", float = TRUE )
```

```
outreg(list("Fingers" = m1), tight = FALSE,
    title = "My Only Spread Out Regressions", float = TRUE,
    alpha = c(0.05, 0.01, 0.001))

outreg(list("Model A" = m1, "Model B label with Spaces" = m2),
    varLabels = list(x1 = "Billie"),
    title = "My Two Linear Regressions", request = c(fstatistic = "F"))

outreg(list("Model A" = m1, "Model B" = m2),
    modelLabels = c("Overrides ModelA", "Overrides ModelB"),
    varLabels = list(x1 = "Billie"),
    title = "Note modelLabels Overrides model names")

ex5 <- outreg(list("Whichever" = m1, "Whatever" = m2),
    title = "Still have showAIC argument, as in previous versions",
    showAIC = TRUE, float = TRUE)
## make a file:
## cat(ex5, file = "some_name_you_choose.tex")


ex5html <- outreg(list("Whichever" = m1, "Whatever" = m2),
    title = "Still have showAIC argument, as in previous versions",
    showAIC = TRUE, type = "html")
## make a file:
## cat(ex5html, file = "some_name_you_choose.html")
## Open that in LibreOffice or MS Word


outreg(list("Whatever" = m1, "Whatever" =m2),
    title = "Another way to get AIC output",
    runFuns = c("AIC" = "Akaike IC"))

outreg(list("Amod" = m1, "Bmod" = m2, "Gmod" = m3),
    title = "My Three Linear Regressions", float = FALSE)

## A new feature in 1.85 is ability to provide vectors of beta estimates
## standard errors, and p values if desired.
## Suppose you have robust standard errors!
library(car)
newSE <- sqrt(diag(car::hccm(m3)))
## See 2 versions of m3 in the table?
outreg(list("Model A" = m1, "Model B" = m2, "Model C" = m3, "Model C w Robust SE" = m3),
        SElist= list("Model C w Robust SE" = newSE))

## outreg uses a t or normal approximation to calculate p values, but you can
## calculate your own.  Let's dial down those std errors but insist they are
## not significantly different from zero
newSE <- 0.3*newSE
newPvals <- rep(0.1, length(newSE))


## Pass in your own SE and P values.
outreg(list("Model A" = m1, "Model B" = m2, "Model C" = m3),
```

```
       SElist = list("Model C" = newSE),
       PVlist = list("Model C" = newPvals), alpha = c(0.05, 0.01, 0.001))
## It took me a while to realize we might as well allow the user to
## pass in a vector of Beta estimates as well. Seems obvious now, though.
outreg(list("Model C" = m3, "Model C Robust SE" = m3, "Model C MLv2" = m3),
      Blist = list("Model C MLv2" = c("(Intercept)" = 0.222, "x1" = 0.222, "x2" = 0.222)),
       SElist = list("Model C Robust SE" = newSE, "Model C MLv2" = 1.4*newSE),
       PVlist = list("Model C" = newPvals),
       alpha = c(0.05, 0.01, 0.001), type = "html")


outreg(list("I Love Long Titles" = m1,
       "Prefer Brevity" = m2,
       "Short" = m3), tight = FALSE, float = FALSE)

outreg(list("GLM" = gm1), float = TRUE)

outreg(list("OLS" = m1, "GLM" = gm1), float = TRUE,
    alpha = c(0.05, 0.01))

outreg(list(OLS = m1, GLM = gm1), float = TRUE,
    request = c(fstatistic = "F"), runFuns = c("BIC" = "BIC"))

outreg(list(OLS = m1, GLM = gm1), float = TRUE,
    request = c(fstatistic = "F"), runFuns = c("BIC" = "BIC"),
    digits = 5, alpha = c(0.01))

outreg(list("OLS 1" = m1, "OLS 2" = m2,  GLM = gm1), float = TRUE,
    request = c(fstatistic = "F"),
    runFuns = c("BIC" = "BIC", logLik = "ll"),
    digits = 5, alpha = c(0.05, 0.01, 0.001))

outreg(list("Model A" = gm1, "Model B label with Spaces" = m2),
    request = c(fstatistic = "F"),
    runFuns = c("BIC" = "Schwarz IC", "AIC" = "Akaike IC",
    "nobs" = "N Again?"))

## Here's a fit example from lme4.
if (require(lme4)){
  fm1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
  outreg(fm1)
  ## Fit same with lm for comparison
  lm1 <- lm(Reaction ~ Days, sleepstudy)
  ## Get robust standard errors
  lm1rse <- sqrt(diag(car::hccm(lm1)))

  outreg(list("Random Effects" = fm1, "OLS" = lm1, "OLS Robust SE" = lm1),
       SElist = list("OLS Robust SE" = lm1rse), type = "html")

  ## From the glmer examples
  gm2 <- glmer(cbind(incidence, size - incidence) ~ period + (1 | herd),
                  data = cbpp, family = binomial)
  lm2 <- lm(incidence/size ~ period,  data = cbpp)
```

```
  lm2rse <- sqrt(diag(hccm(lm2)))
  library(MASS)
  ## Lets see what MASS::rlm objects do? Mostly OK
  rlm2 <- MASS::rlm(incidence/size ~ period, data = cbpp)

  outreg(list("GLMER" = gm2, "lm" = lm2, "lm w/robust se" = lm2, "rlm" = rlm2),
       SElist = list("lm w/robust se" = lm2rse), type = "html")

}
```

---

outreg0 *Creates a publication quality result table for regression models. outreg0 is the last version in the last development stream.*

Description

outreg0 writes its output directly to the terminal, but does not create an output object. The new version of this function–which I wish you would try instead–does the same work, but it also creates an output object that can be transformed for other purposes.

Usage

```
outreg0(modelList, title, label, modelLabels = NULL, varLabels = NULL,
  tight = TRUE, showAIC = FALSE, float = FALSE, request, runFuns,
  digits = 3, alpha = 0.05)
```

Arguments

modelList	A regression model or an R list of regression models. If this is a named list, the names will be used as column labels, unless the argument modelLabels is supplied separately, which will override the names of modelList.
title	A title to be displayed on the top of the LaTeX regression table.
label	A string to be used as a LaTeX label in the table to be created.
modelLabels	A vector of charater string variables, one for each element in modelList. Will override the names in modelList.
varLabels	To beautify the parameter names printed. Must be a named vector in the format c(parmname = "displayName", parmname = "displayName"). Include as many parameters as desired, it is not necessary to supply new labels for all of the parameters. "displayName" must be valid laTeX. This feature is useful if your variable names have any illegal LaTeX characters, such as "$" or "_".
tight	If TRUE, parameter estimates and standard errors are printed in a single column. If FALSE, parameter estimates and standard errors are printed side by side.
showAIC	If TRUE, the AIC estimate is included with the diagnostic values
float	Include boilerplate for a table float, with the tabular markup inside it.

request	Extra information to be retrieved from the summary(model) and displayed. This must be a vector of named arguments, such as c(adj.r.squared = "adj R^2", fstatistic = "F"). The name must be a valid name of the output object, the value should be the label the user wants printed in the table. See details.
runFuns	A list of functions
digits	Default = 3. How many digits after decimal sign are to be displayed.
alpha	Default = 0.05. I think stars are dumb, but enough people have asked me for more stars that I'm caving in. Enter c(0.05, 0.01, 0.001) to see what happens.

Details

This provides "markup" that the user is will copy into a LaTeX document. The table is a LaTeX tabular environment, which can be enclosed in a table to obtain a "floating," automatically numbered LaTeX document environment. The default will create the tabular-inside-a-table, but if only the tabular is needed, use the argument float = FALSE. This can be called within Sweave documents. The markup generated by the basic usage will generally be presentable as is, while user requests for additional details may cause output that needs some hand-editing.

A LaTeX tabular object is created, with the floating table markup around it if requested (float = TRUE).

The table include a minimally sufficient (in my opinion) model summary. For any fitted model, `outreg0()` will present the parameter estimates and standard errors, and it will also scan the summary of the object for some summary values and goodness of fit indicators. Some users may want more information. lm fits will not include output for the "fstatistic" that is reported in the lm's summary object, but users can ask for it with the argument `request`. Similarly, the argument `runFuns` can ask for additional diagnostic functions to be reported.

While `outreg0()` originally designed for models fitted by `lm()` and `glm()`, it will now work for any kind of regression model that has methods `summary()`, `coef()`, `vcov()` and `nobs()`. This will not succeed, however, if the regression model's summary function does not create an object that holds values in a workable structure (in the same way that `lm()` and `glm()` do). Please pressure people who write R packages that claim to "do regression" to supply methods to do the basic work that we need to compile tables.

Two types of customization arguments were introduced with rockchalk version 1.7.3. The new arguments are `request` and `runFuns`. Beta test reports are welcome!

The `request` argument supplies a list of names of summary output elements that are desired. The format is a pair, a value to be retrieved and a name to be printed for it. With the `lm()` regression, for example, one might want the output of the F test and the adjusted R-square. An example usage might be `request = c(adj.r.squared = "adj. $R^2", fstatistic = "F")`. The names may be included in quotation marks if the user desires to do so: `request = c("adj.r.squared" = "adj. $R^2","fstatistic" = "F`
The value on the left is the name of the desired information in the summary object, while the value on the right is *any* valid LaTeX markup that the user desires to display in the first column of the table. `request` terms that generate a single numerical value will generally work fine, while requests that ask for more structured information, such as the F test (including the 2 degrees of freedom values) are still a work in progress.

The `runFuns` argument is inspired by a user request: could this include the BIC or other summaries that some models report? We have to run the BIC function, and divert the result into the correct column of the result table. Any R function, whether supplied with and R package or in the user's

own code, may be used. This is a two-part specification, one representing the function to be run, the other representing the name that is desired in the output. For example, it might be `runFuns = c("AIC" = "Akaike Criterion", "BIC" = "Schwartz Criterion","logLik" = "LL")`.

Value

None

Note

There are many R packages that can be used to create LaTeX regression tables. memisc, texreg, apsrtable, xtables, and rms are some. This "outreg" version was in use in our labs before we were aware that those packages were in development. It is not intended as a competitor, it is just a slightly different version of the same that is more suited to our needs.

Author(s)

Paul E. Johnson <<pauljohn@ku.edu>>

Examples

```
set.seed(2134234)
dat <- data.frame(x1 = rnorm(100), x2 = rnorm(100))
dat$y1 <- 30 + 5 * rnorm(100) + 3 * dat$x1 + 4 * dat$x2
dat$y2 <- rnorm(100) + 5 * dat$x2
m1 <- lm(y1 ~ x1, data = dat)
m2 <- lm(y1 ~ x2, data = dat)
m3 <- lm(y1 ~ x1 + x2, data = dat)
gm1 <- glm(y1 ~ x1, family = Gamma, data = dat)

outreg0(list("Model 1" = m1), title = "My One Tightly Printed Regression")

outreg0(list("Model 1" = m1), title = "My One Tightly Printed Regression in a
Float", float = TRUE, label = "outreg1")

outreg0(list("Fingers" = m1), tight = FALSE, title = "My Only
Spread Out Regression", float = TRUE, alpha = c(0.05, 0.01,
0.001))

outreg0(list(ModelA = m1, "Model B label with Spaces" = m2),
      varLabels = list(x1 = "Billie"),
      title = "My Two Linear Regressions",
      request = c(fstatistic = "F"))

outreg0(list(ModelA = m1, ModelB = m2),
    modelLabels = c("Overrides ModelA", "Overrides ModelB"),
    varLabels = list(x1 = "Billie"),
    title = "Note modelLabels Overrides model names")

outreg0(list(m1, m2), modelLabels = c("Whatever", "Whichever"),
    title = "Still have showAIC argument, as in previous versions",
    showAIC = TRUE, float = TRUE)
```

```
outreg0(list(m1, m2), modelLabels = c("Whatever", "Whichever"),
    title = "Another way to get AIC output",
    runFuns = c("AIC" = "Akaike IC"))

outreg0(list("Amod" = m1, "Bmod" = m2, "Gmod" = m3),
    title = "My Three Linear Regressions", float = FALSE)

outreg0(list(m1, m2, m3), tight = FALSE,
    modelLabels = c("I Love really long titles", "Hate Long", "Medium"),
    float = FALSE)

outreg0(list(gm1), modelLabels = c("GLM"), float = TRUE)

outreg0(list(m1, gm1), modelLabels = c("OLS", "GLM"),
    float = TRUE, alpha = c(0.05, 0.01))

outreg0(list(OLS = m1, GLM = gm1), float = TRUE,
    request = c(fstatistic = "F"), runFuns = c("BIC" = "BIC"))

outreg0(list(OLS = m1, GLM = gm1), float = TRUE,
    request = c(fstatistic = "F"), runFuns = c("BIC" = "BIC"),
    digits = 5, alpha = 0.01)

outreg0(list("OLS 1" = m1, "OLS 2" = m2,  GLM = gm1), float = TRUE,
    request = c(fstatistic = "F"),
    runFuns = c("BIC" = "BIC", logLik = "ll"),
    digits = 5, alpha = c(0.05, 0.01, 0.001))

outreg0(list(ModelA = gm1, "Model B label with Spaces" = m2),
    request = c(fstatistic = "F"),
    runFuns = c("BIC" = "Schwarz IC", "AIC" = "Akaike IC",
         "nobs" = "N Again?"))
```

`outreg2HTML` *Convert LaTeX output from outreg to HTML markup*

Description

This function is deprecated. Instead, please use `outreg(type = "html")`

Usage

```
outreg2HTML(outreg, filename)
```

Arguments

`outreg`	output from outreg
`filename`	A file name into which the regression markup is to be saved. Should end in .html.

Details

This will write the html on the screen, but if a filename argument is supplied, it will write a file. One can then open or insert the file into Libre Office or other popular "word processor" programs.

Value

A vector of strings

Author(s)

Paul E. Johnson <<pauljohn@ku.edu>>

Examples

```
dat <- genCorrelatedData2(means = c(50,50,50,50,50,50),
    sds = c(10,10,10,10,10,10), rho = 0.2, beta = rnorm(7), stde = 50)
m1 <- lm(y ~ x1 + x2 + x3 + x4 + x5 + x6 + x1*x2, data = dat)
summary(m1)

m1out <- outreg(list("Great Regression" = m1), alpha = c(0.05, 0.01, 0.001),
         request = c("fstatistic" = "F"), runFuns = c(AIC = "AIC"),
         float = TRUE)
##html markup will appear on screen
outreg2HTML(m1out)
## outreg2HTML(m1out, filename = "funky.html")
## I'm not running that for you because you
## need to be in the intended working directory

m2 <- lm(y ~ x1 + x2, data = dat)

m2out <- outreg(list("Great Regression" = m1, "Small Regression" = m2),
               alpha = c(0.05, 0.01, 0.01),
                request = c("fstatistic" = "F"), runFuns = c(BIC = "BIC"))
outreg2HTML(m2out)
## Run this for yourself, it will create the output file funky2.html
## outreg2HTML(m2out, filename = "funky2.html")
## Please inspect the file "funky2.html
```

---

padW0 *Pad with 0's.*

Description

Sometimes we receive this c(1, 22, 131) and we need character variables of the same size, such as c("001", "022", "131"). This happens if a user has mistakenly converted a zip code (US regional identifier) like "00231" to a number. This function converts the number back to a 0 padded string.

Usage

```
padW0(x)
```

Arguments

x	a numeric variable.

Details

This works differently if the number provided is an integer, or a character string. Integers are left padded with the character "0". A character string will be left-padded with blanks.

Value

A character string vector padded with 0's

Author(s)

Paul Johnson <pauljohn@ku.edu>

Examples

```
x <- c(1 , 11, 22, 121, 14141, 31)
(xpad <- padW0(x))
x <- rpois(7, lambda = 11)
(xpad <- padW0(x))
x <- c("Alabama", "Iowa", "Washington")
```

---

perspEmpty *perspEmpty*

Description

Creates a persp plot without drawing anything in the interior. Does equivalent of `plot( type="n")` for persp.

Usage

```
perspEmpty(x1, x2, y, x1lab = "x1", x2lab = "x2", ylab = "y", x1lim,
  x2lim, ...)
```

Arguments

x1	data for the first horizontal axis, an R vector
x2	data for the second horizontal axis, an R vector
y	data for the vertical axis, an R vector
x1lab	label for the x1 axis, (the one called "xlab" inside persp)
x2lab	label for the x2 axis, (the one called "ylab" inside persp)
ylab	label for the y (vertical) axis (the one called "zlab" inside persp)
x1lim	Optional: limits for x1 axis (should be a vector with 2 elements)

x2lim	Optional: limits for x2 axis (should be a vector with 2 elements)
...	further arguments that are passed to persp. Please note Please remember that y is the vertical axis, but for persp, that is the one I call x2. Thus dot-dot-dot arguments including xlab, ylab, zlab, xlim, ylim, and zlim are going to be ignored.

Details

Regression demonstrations require a blank slate in which points and planes can be drawn. This function creates that blank persp canvas for those projects. It is not necessary that x1, x2 and y be vectors of the same length, since this function's only purpose is to plot an empty box with ranges determined by the input variables. persp calls the 3 axes x, y, and z, but here they are called x1, x2, and y.

Value

The perspective matrix that is returned by persp

Examples

```
x1 <- 1:10
x2 <- 41:50
y <-  rnorm(10)
perspEmpty(x1, x2, y)
res <- perspEmpty(x1, x2, y, ticktype="detailed", nticks=10)
mypoints1 <- trans3d ( x1, x2, y, pmat = res )
points( mypoints1, pch = 16, col= "blue")
```

---

plot.testSlopes *Plot testSlopes objects*

Description

plot.testSlopes is a method for the generic function plot. It has been revised so that it creates a plot illustrating the marginal effect, using the Johnson-Neyman interval calculations to highlight the "statistically significantly different from zero" slopes.

Usage

```
## S3 method for class 'testSlopes'
plot(x, ..., shade = TRUE, col = rgb(1, 0, 0, 0.1))
```

Arguments

x	A testSlopes object.
...	Additional arguments that are ignored currently.
shade	Optional. Create colored polygon for significant regions.
col	Optional. Color of the shaded area. Default transparent pink.

Value

NULL

Author(s)

<pauljohn@ku.edu>

---

plotCurves *Assists creation of predicted value curves for regression models.*

Description

Creates a predicted value plot that includes a separate predicted value line for each value of a focal variable. The x axis variable is specified by the plotx argument. As of rockchalk 1.7.x, the moderator argument, modx, is optional. Think of this a new version of R's termplot, but it allows for interactions. And it handles some nonlinear transformations more gracefully than termplot.

Usage

```
plotCurves(model, plotx, modx, n, modxVals = NULL, interval = c("none",
  "confidence", "prediction"), plotPoints = TRUE, plotLegend = TRUE,
  legendTitle = NULL, col = NULL, llwd = 2, opacity = 100,
  envir = environment(formula(model)), ...)
```

Arguments

model	Required. Fitted regression object. Must have a predict method
plotx	Required. String with name of predictor for the x axis
modx	Optional. String for moderator variable name. May be either numeric or factor.
n	Optional. Number of focal values of modx, used by algorithms specified by modxVals; will be ignored if modxVals supplies a vector of focal values.
modxVals	Optional. A vector of focal values for which predicted values are to be plotted. May also be a character string to select an algorithm ("quantile","std.dev." or "table"), or a user-supplied function to select focal values (a new method similar to getFocal). If modx is a factor, currently, the only available algorithm is "table" (see getFocal.factor.
interval	Optional. Intervals provided by the predict.lm may be supplied, either "conf" (95 interval for the estimated conditional mean) or "pred" (95 interval for observed values of y given the rest of the model).
plotPoints	Optional. TRUE or FALSE: Should the plot include the scatterplot points along with the lines.
plotLegend	Optional. TRUE or FALSE: Should the default legend be included?
legendTitle	Optional. You'll get an automatically generated title, such as "Moderator: modx", but if you don't like that, specify your own string here.

col	Optional. A color vector to differentiate the moderator values in the plot. If not specified, the R's builtin palette() will be used. User may supply a vector of valid color names, either explicitly c("pink","black", "gray70") or implicitly, rainbow(10) or gray.colors(5). Color names will be recycled if there are more focal values of modx than colors provided.
envir	environment to search for variables.
llwd	Optional. Line widths for predicted values. Can be single value or a vector, which will be recycled as necessary. ##'
opacity	Optional, default = 100. A number between 1 and 255. 1 means "transparent" or invisible, 255 means very dark. the darkness of confidence interval regions
...	further arguments that are passed to plot or predict. The arguments that are monitored to be sent to predict are c("type", "se.fit", "dispersion", "terms", "na.action").

Details

This is similar to plotSlopes, but it accepts regressions in which there are transformed variables, such as "log(x1)". It creates a plot of the predicted dependent variable against one of the numeric predictors, plotx. It draws a predicted value line for each value of modx, a moderator variable. The moderator may be a numeric or categorical moderator variable.

The user may designate which particular values of the moderator are used for calculating the predicted value lines. That is, modxVals = c(12,22,37) would draw lines for values 12, 22, and 37 of the moderator. User may instead supply a character string to choose one of the built in algorithms. The default algorithm is "quantile", which will select n values that are evenly spaced along the modx axis. The algorithm "std.dev" will select the mean of modx (m) and then it will select values that step away from the mean in standard deviation sd units. For example, if n = 3, the focal values will m, m - sd, am + sd.

Value

A plot is created as a side effect, a list is returned including 1) the call, 2) a newdata object that includes information on the curves that were plotted, 3) a vector modxVals, the values for which curves were drawn.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
library(rockchalk)

## Replicate some R classics.  The budworm.lg data from predict.glm
## will work properly after re-formatting the information as a data.frame:

## example from Venables and Ripley (2002, pp. 190-2.)
df <- data.frame(ldose = rep(0:5, 2),
                 sex = factor(rep(c("M", "F"), c(6, 6))),
                 SF.numdead = c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16))
```

```
df$SF.numalive = 20 - df$SF.numdead

budworm.lg <- glm(cbind(SF.numdead, SF.numalive) ~ sex*ldose, data = df,
                  family = binomial)


plotCurves(budworm.lg, plotx = "ldose", modx = "sex", interval = "confidence",
           ylim = c(0, 1))

## See infert
model2 <- glm(case ~ age + parity + education + spontaneous + induced,
              data = infert, family = binomial())

## Curvature so slight we can barely see it
model2pc1 <- plotCurves(model2, plotx = "age", modx = "education",
                        interval = "confidence", ylim = c(0, 1))


model2pc2 <- plotCurves(model2, plotx = "age", modx = "education",
                        modxVals = levels(infert$education)[1],
                        interval = "confidence", ylim = c(0, 1))


model2pc2 <- plotCurves(model2, plotx = "age", modx = "education",
                        modxVals = levels(infert$education)[c(2,3)],
                        interval = "confidence", ylim = c(0, 1))

model2pc2 <- plotCurves(model2, plotx = "age", modx = "education",
                        modxVals = levels(infert$education)[c(2,3)],
                        ylim = c(0, 1), type = "response")




## Manufacture some data
set.seed(12345)
N <- 500
dat <- genCorrelatedData2(N = 500, means = c(5, 0, 0, 0), sds = rep(1, 4),
                          rho = 0.2, beta = rep(1, 5), stde = 20)

dat$xcat1 <- gl(2, N/2, labels = c("Monster", "Human"))
dat$xcat2 <- cut(rnorm(N), breaks = c(-Inf, 0, 0.4, 0.9, 1, Inf),
             labels = c("R", "M", "D", "P", "G"))

###The design matrix for categorical variables, xcat numeric
dat$xcat1n <- with(dat, contrasts(xcat1)[xcat1, ])
dat$xcat2n <- with(dat, contrasts(xcat2)[xcat2, ])


stde <- 2
dat$y <- with(dat, 0.03 + 11.5 * log(x1) * contrasts(dat$xcat1)[dat$xcat1] +
              0.1 * x2 + 0.04 * x2^2 + stde*rnorm(N))
```

```
stde <- 1
dat$y2 <- with(dat, 0.03 + 0.1 * x1 + 0.1 * x2 + 0.25 * x1 * x2 + 0.4 * x3 -
               0.1 * x4 + stde * rnorm(N))
stde <- 8
dat$y3 <- with(dat, 3 + 0.5 * x1 + 1.2 * (as.numeric(xcat1)-1) +
-0.8 * (as.numeric(xcat1)-1) * x1 +  stde * rnorm(N))

stde <- 8
dat$y4 <- with(dat, 3 + 0.5 * x1 +
               contrasts(dat$xcat2)[dat$xcat2, ] %*% c(0.1, -0.2, 0.3, 0.05)  +
               stde * rnorm(N))


## Curvature with interaction
m1 <- lm(y ~ log(x1)*xcat1 + x2 + I(x2^2), data=dat)
summary(m1)

## First, with no moderator
plotCurves(m1, plotx = "x1")

plotCurves(m1, plotx = "x1", modx = "xcat1")

## ## Verify that plot by comparing against a manually contructed alternative
## par(mfrow=c(1,2))
## plotCurves(m1, plotx = "x1", modx = "xcat1")
## newdat <- with(dat, expand.grid(x1 = plotSeq(x1, 30), xcat1 = levels(xcat1)))
## newdat$x2 <-  with(dat, mean(x2, na.rm = TRUE))
## newdat$m1p <- predict(m1, newdata = newdat)
## plot( y ~ x1, data = dat, type = "n", ylim = magRange(dat$y, c(1, 1.2)))
## points( y ~ x1, data = dat, col = dat$xcat1, cex = 0.4, lwd = 0.5)
## by(newdat, newdat$xcat1, function(dd) {lines(dd$x1, dd$m1p)})
## legend("topleft", legend=levels(dat$xcat1), col = as.numeric(dat$xcat1), lty = 1)
## par(mfrow = c(1,1))
## ##Close enough!


plotCurves(m1, plotx = "x2", modx = "x1")

plotCurves(m1, plotx = "x2", modx = "xcat1")

plotCurves(m1, plotx = "x2", modx = "xcat1", interval = "conf")


m2 <- lm(y ~ log(x1)*xcat1 + xcat1*(x2 + I(x2^2)), data = dat)
summary(m2)
plotCurves(m2, plotx = "x2", modx = "xcat1")

plotCurves(m2, plotx  ="x2", modx = "x1")


m3a <- lm(y ~ poly(x2, 2) + xcat1, data = dat)
```

```
plotCurves(m3a, plotx = "x2")
plotCurves(m3a, plotx = "x2", modx = "xcat1")
#OK

m4 <- lm(log(y+10) ~ poly(x2, 2)*xcat1 + x1, data = dat)
summary(m4)
plotCurves(m4, plotx = "x2")

plotCurves(m4, plotx  ="x2", modx = "xcat1")

plotCurves(m4, plotx = "x2", modx = "x1")

plotCurves(m4, plotx = "x2", modx = "xcat1")

plotCurves(m4, plotx = "x2", modx = "xcat1", modxVals = c("Monster"))


##ordinary interaction
m5 <- lm(y2 ~ x1*x2 + x3 +x4, data = dat)
summary(m5)
plotCurves(m5, plotx = "x1", modx = "x2")
plotCurves(m5, plotx = "x1", modx = "x2", modxVals = c( -2, -1, 0, 1, 2))
plotCurves(m5, plotx = "x1", modx = "x2", modxVals = c(-2))
plotCurves(m5, plotx = "x1", modx = "x2", modxVals = "std.dev.")
plotCurves(m5, plotx = "x1", modx = "x2", modxVals = "quantile")
plotCurves(m5, plotx = "x3", modx = "x2")


library(car)
mc1 <- lm(statusquo ~ income * sex, data = Chile)
summary(mc1)
plotCurves(mc1, plotx = "income")
plotCurves(mc1, modx = "sex", plotx = "income")
plotCurves(mc1, modx = "sex", plotx = "income", modxVals = "M")

mc2 <- lm(statusquo ~ region * income, data =  Chile)
summary(mc2)
plotCurves(mc2, modx = "region", plotx = "income")
plotCurves(mc2, modx = "region", plotx = "income",
           modxVals = levels(Chile$region)[c(1,4)])
plotCurves(mc2, modx = "region", plotx = "income", modxVals = c("S","M","SA"))
plotCurves(mc2, modx = "region", plotx = "income", modxVals = c("S","M","SA"),
           interval = "conf")

plotCurves(mc2, modx = "region", plotx = "income", plotPoints = FALSE)


mc3 <- lm(statusquo ~ region * income + sex + age, data =  Chile)
summary(mc3)
plotCurves(mc3, modx = "region", plotx = "income")


mc4 <- lm(statusquo ~ income * (age + I(age^2)) + education + sex + age, data = Chile)
```

```
summary(mc4)
plotCurves(mc4, plotx = "age")
plotCurves(mc4, plotx = "age", interval = "conf")


plotCurves(mc4, plotx = "age", modx = "income")
plotCurves(mc4, plotx = "age", modx = "income", plotPoints = FALSE)

plotCurves(mc4,  plotx = "income", modx = "age")
plotCurves(mc4,  plotx = "income", modx = "age", n = 8)

plotCurves(mc4,  plotx = "income", modx = "age", modxVals = "std.dev.")
plotCurves(mc4, modx = "income", plotx = "age", plotPoints = FALSE)
```

`plotFancy` *Regression plots with predicted value lines, confidence intervals, color coded interactions*

Description

This is the back-end for the functions plotSlopes and plotCurves. Don't use it directly.

Usage

```
plotFancy(newdf, olddf, plotx, modx, modxVals, interval, plotPoints, plotLegend,
  legendTitle, col = NULL, llwd = 2, opacity, ...)
```

Arguments

`newdf`	The new data frame with predictors and fit, lwr, upr variables
`olddf`	A data frame with variables(modxVar, plotxVar, depVar)
`plotx`	Character string for name of variable on horizontal axis
`modx`	Character string for name of moderator variable.
`modxVals`	Values of moderator for which lines are desired
`interval`	TRUE or FALSE: want confidence intervals?
`plotPoints`	TRUE or FALSE: want to see observed values in plot?
`plotLegend`	TRUE or FALSE: draw defaut legend
`legendTitle`	Optional. You'll get an automatically generated title, such as "Moderator: modx", but if you don't like that, specify your own string here.
`col`	requested color scheme for lines and points. One per value of modxVals.
`llwd`	requested line width, will re-cycle.
`opacity`	Value in 0, 255 for darkness of interval shading
`...`	Other arguments passed to plot function.

Value

col, lty, and lwd information

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

`plotPlane` *Draw a 3-D regression plot for two predictors from any linear or non-linear lm or glm object*

Description

This allows user to fit a regression model with many variables and then plot 2 of its predictors and the output plane for those predictors with other variables set at mean or mode (numeric or factor). This is a front-end (wrapper) for R's persp function. Persp does all of the hard work, this function reorganizes the information for the user in a more readily understood way. It intended as a convenience for students (or others) who do not want to fight their way through the details needed to use persp to plot a regression plane. The fitted model can have any number of input variables, this will display only two of them. And, at least for the moment, I insist these predictors must be numeric variables. They can be transformed in any of the usual ways, such as poly, log, and so forth.

Usage

```
plotPlane(model = NULL, plotx1 = NULL, plotx2 = NULL,
  drawArrows = FALSE, plotPoints = TRUE, npp = 20, x1lab, x2lab, ylab,
  x1lim, x2lim, x1floor = 5, x2floor = 5, pch = 1, pcol = "blue",
  plwd = 0.5, pcex = 1, llwd = 0.3, lcol = 1, llty = 1,
  acol = "red", alty = 4, alwd = 0.3, alength = 0.1, linesFrom,
  lflwd = 3, envir = environment(formula(model)), ...)

## Default S3 method:
plotPlane(model = NULL, plotx1 = NULL, plotx2 = NULL,
  drawArrows = FALSE, plotPoints = TRUE, npp = 20, x1lab, x2lab, ylab,
  x1lim, x2lim, x1floor = 5, x2floor = 5, pch = 1, pcol = "blue",
  plwd = 0.5, pcex = 1, llwd = 0.3, lcol = 1, llty = 1,
  acol = "red", alty = 4, alwd = 0.3, alength = 0.1, linesFrom,
  lflwd = 3, envir = environment(formula(model)), ...)
```

Arguments

`model`	an lm or glm fitted model object
`plotx1`	name of one variable to be used on the x1 axis
`plotx2`	name of one variable to be used on the x2 axis
`drawArrows`	draw red arrows from prediction plane toward observed values TRUE or FALSE

plotPoints	Should the plot include scatter of observed scores?
npp	number of points at which to calculate prediction
x1lab	optional label
x2lab	optional label
ylab	optional label
x1lim	optional lower and upper bounds for x1, as vector like c(0,1)
x2lim	optional lower and upper bounds for x2, as vector like c(0,1)
x1floor	Default=5. Number of "floor" lines to be drawn for variable x1
x2floor	Default=5. Number of "floor" lines to be drawn for variable x2
pch	plot character, passed on to the "points" function
pcol	color for points, col passed to "points" function
plwd	line width, lwd passed to "points" function
pcex	character expansion, cex passed to "points" function
llwd	line width, lwd passed to the "lines" function
lcol	line color, col passed to the "lines" function
llty	line type, lty passed to the "lines" function
acol	color for arrows, col passed to "arrows" function
alty	arrow line type, lty passed to the "arrows" function
alwd	arrow line width, lwd passed to the "arrows" function
alength	arrow head length, length passed to "arrows" function
linesFrom	object with information about "highlight" lines to be added to the 3d plane (output from plotCurves or plotSlopes)
lflwd	line widths for linesFrom highlight lines
envir	environment from whence to grab data
...	additional parameters that will go to persp

Details

Besides a fitted model object, plotPlane requires two additional arguments, plotx1 and plotx2. These are the names of the plotting variables. Please note, that if the term in the regression is something like poly(fish,2) or log(fish), then the argument to plotx1 should be the quoted name of the variable "fish". plotPlane will handle the work of re-organizing the information so that R's predict functions can generate the desired information. This might be thought of as a 3D version of "termplot", with a significant exception. The calculation of predicted values depends on predictors besides plotx1 and plotx2 in a different ways. The sample averages are used for numeric variables, but for factors the modal value is used.

This function creates an empty 3D drawing and then fills in the pieces. It uses the R functions `lines`, `points`, and `arrows`. To allow customization, several parameters are introduced for the users to choose colors and such. These options are prefixed by "l" for the lines that draw the plane, "p" for the points, and "a" for the arrows. Of course, if plotPoints=FALSE or drawArrows=FALSE, then these options are irrelevant.

Value

The main point is the plot that is drawn, but for record keeping the return object is a list including 1) res: the transformation matrix that was created by persp 2) the call that was issued, 3) x1seq, the "plot sequence" for the x1 dimension, 4) x2seq, the "plot sequence" for the x2 dimension, 5) zplane, the values of the plane corresponding to locations x1seq and x2seq.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

See Also

persp, scatterplot3d, regr2.plot

Examples

```
library(rockchalk)


set.seed(12345)
x1 <- rnorm(100)
x2 <- rnorm(100)
x3 <- rnorm(100)
x4 <- rnorm(100)
y <- rnorm(100)
y2 <- 0.03 + 0.1*x1 + 0.1*x2 + 0.25*x1*x2 + 0.4*x3 -0.1*x4 + 1*rnorm(100)
dat <- data.frame(x1,x2,x3,x4,y, y2)
rm(x1, x2, x3, x4, y, y2)

## linear ordinary regression
m1 <- lm(y ~ x1 + x2 +x3 + x4, data = dat)

plotPlane(m1, plotx1 = "x3", plotx2 = "x4")

plotPlane(m1, plotx1 = "x3", plotx2 = "x4", drawArrows = TRUE)

plotPlane(m1, plotx1 = "x1", plotx2 = "x4", drawArrows = TRUE)


plotPlane(m1, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE, npp = 10)
plotPlane(m1, plotx1 = "x3", plotx2 = "x2", drawArrows = TRUE, npp = 40)

plotPlane(m1, plotx1 = "x3", plotx2 = "x2", drawArrows = FALSE,
          npp = 5, ticktype = "detailed")


## regression with interaction
m2 <- lm(y ~ x1  * x2 +x3 + x4, data = dat)

plotPlane(m2, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE)
```

```
plotPlane(m2, plotx1 = "x1", plotx2 = "x4", drawArrows = TRUE)
plotPlane(m2, plotx1 = "x1", plotx2 = "x3", drawArrows = TRUE)

plotPlane(m2, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE,
          phi = 10, theta = 30)



## regression with quadratic;
## Required some fancy footwork in plotPlane, so be happy
dat$y3 <- 0 + 1 * dat$x1 + 2 * dat$x1^2 + 1 * dat$x2 +
    0.4*dat$x3 + 8 * rnorm(100)
m3 <- lm(y3 ~ poly(x1,2) + x2 +x3 + x4, data = dat)
summary(m3)

plotPlane(m3, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE,
          x1lab = "my great predictor", x2lab = "a so-so predictor",
          ylab = "Most awesomest DV ever")

plotPlane(m3, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE,
          x1lab = "my great predictor", x2lab = "a so-so predictor",
          ylab = "Most awesomest DV ever", phi = -20)

plotPlane(m3, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE,
          phi = 10, theta = 30)

plotPlane(m3, plotx1 = "x1", plotx2 = "x4", drawArrows = TRUE,
          ticktype = "detailed")
plotPlane(m3, plotx1 = "x1", plotx2 = "x3", drawArrows = TRUE)

plotPlane(m3, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE,
          phi = 10, theta = 30)

m4 <- lm(y ~ sin(x1) + x2*x3 +x3 + x4, data = dat)
summary(m4)


plotPlane(m4, plotx1 = "x1", plotx2 = "x2", drawArrows = TRUE)
plotPlane(m4, plotx1 = "x1", plotx2 = "x3", drawArrows = TRUE)



eta3 <- 1.1 + .9*dat$x1 - .6*dat$x2 + .5*dat$x3
dat$y4 <- rbinom(100, size = 1, prob = exp( eta3)/(1+exp(eta3)))
gm1 <- glm(y4 ~ x1 + x2 + x3, data = dat, family = binomial(logit))
summary(gm1)
plotPlane(gm1, plotx1 = "x1", plotx2 = "x2")
plotPlane(gm1, plotx1 = "x1", plotx2 = "x2", phi = -10)

plotPlane(gm1, plotx1 = "x1", plotx2 = "x2", ticktype = "detailed")
plotPlane(gm1, plotx1 = "x1", plotx2 = "x2", ticktype = "detailed",
          npp = 30, theta = 30)
plotPlane(gm1, plotx1 = "x1", plotx2 = "x3", ticktype = "detailed",
```

```
          npp = 70, theta = 60)

plotPlane(gm1, plotx1 = "x1", plotx2 = "x2", ticktype = c("detailed"),
          npp = 50, theta = 40)

dat$x2 <- 5 * dat$x2
dat$x4 <- 10 * dat$x4
eta4 <- 0.1 + .15*dat$x1 - 0.1*dat$x2 + .25*dat$x3 + 0.1*dat$x4
dat$y4 <- rbinom(100, size = 1, prob = exp( eta4)/(1+exp(eta4)))
gm2 <- glm(y4 ~ x1 + x2 + x3 + x4, data = dat, family = binomial(logit))
summary(gm2)
plotPlane(gm2, plotx1 = "x1", plotx2 = "x2")
plotPlane(gm2, plotx1 = "x2", plotx2 = "x1")
plotPlane(gm2, plotx1 = "x1", plotx2 = "x2", phi = -10)
plotPlane(gm2, plotx1 = "x1", plotx2 = "x2", phi = 5, theta = 70, npp = 40)

plotPlane(gm2, plotx1 = "x1", plotx2 = "x2", ticktype = "detailed")
plotPlane(gm2, plotx1 = "x1", plotx2 = "x2", ticktype = "detailed",
          npp = 30, theta = -30)
plotPlane(gm2, plotx1 = "x1", plotx2 = "x3", ticktype = "detailed",
          npp = 70, theta = 60)

plotPlane(gm2, plotx1 = "x4", plotx2 = "x3", ticktype = "detailed",
          npp = 50, theta = 10)

plotPlane(gm2, plotx1 = "x1", plotx2 = "x2", ticktype = c("detailed"))
```

`plotSeq` *Create sequences for plotting*

Description

`plotSeq` is a convenience for the creation of sequence across the range of a variable. By default, the length of the plotting sequence will be equal to the length of the original sequence. In that case, the only effect is to create an evenly-spaced set of values. If `length.out` is specified, the user determines the number of elements in plotSeq.

Usage

```
plotSeq(x, length.out = length(x))
```

Arguments

`x`	an R vector variable
`length.out`	the number of elements in the desired plotting sequence.

Details

The primary intended usage is for the creation of plotting sequences of numeric variables. It takes a variable's range and the fills in evenly spaced steps. If x is a factor variable, the levels will be returned. Uses of this functionality are planned in the future.

See Also

pretty

Examples

```
#Create a quadratic regression

stde <- 14
x <- rnorm(100, m = 50, s = 10)
y <- 0.2 - 02*x + 0.2*x^2 + stde*rnorm(100)
mod1 <- lm (y ~ poly(x, 2))

plot(x, y, main="The Quadratic Regression")
seqx <- plotSeq(x, length.out = 10)
seqy <- predict(mod1, newdata = data.frame(x = seqx))
lines(seqx, seqy, col = "red")

# Notice the bad result when a plotting sequence is
# not used.
plot(x, y, main = "Bad Plot Result")
seqy <- predict(mod1)
lines(x, seqy, col = "green")
```

---

plotSlopes *Generic function for plotting regressions and interaction effects*

Description

This is a generic function for plotting regression objects. So far, there is an implementation for `lm()` objects. This allows interaction effects, but not nonlinearities like log(x1). For that, please see `plotCurves`.

This is a "simple slope" plotter for linear regression objects that are created by `lm()`. The function `plotCurves()` can handle nonlinear predictive equations and generalized linear models. The term "simple slopes" was coined by psychologists (Aiken and West, 1991; Cohen, et al 2002) for analysis of interaction effects for particular values of a moderating variable. The moderating variable may be continuous or categorical, lines will be plotted for focal values of that variable.

Usage

```
plotSlopes(model, plotx, ...)

## S3 method for class 'lm'
plotSlopes(model, plotx, modx, n = 3, modxVals = NULL,
  interval = c("none", "confidence", "prediction"), plotPoints = TRUE,
  plotLegend = TRUE, legendTitle = NULL, col = NULL, llwd = 2,
  opacity = 100, ...)
```

Arguments

model	Required. A fitted Regression
plotx	Required. Name of one predictor from the fitted model to be plotted on horizontal axis
...	Additional arguments passed to methods. Often includes arguments that are passed to plot. Any arguments that customize plot output, such as lwd, cex, and so forth, may be supplied. These arguments intended for the predict method will be used: c("type", "se.fit", "dispersion", "terms", "na.action")
modx	Optional. String for moderator variable name. May be either numeric or factor. If omitted, a single predicted value line will be drawn.
n	Optional. Number of focal values of modx, used by algorithms specified by modxVals; will be ignored if modxVals supplies a vector of focal values.
modxVals	Optional. Focal values of modx for which lines are desired. May be a vector of values or the name of an algorithm, "quantile", "std.dev.", or "table".
interval	Optional. Intervals provided by the predict.lm may be supplied, either "confidence" (95 interval for the estimated conditional mean) or "prediction" (95 interval for observed values of y given the rest of the model).
plotPoints	Optional. TRUE or FALSE: Should the plot include the scatterplot points along with the lines.
plotLegend	Optional. TRUE or FALSE: Include a default
legendTitle	Optional. You'll get an automatically generated title, such as "Moderator: modx", but if you don't like that, specify your own string here. legend. Set to FALSE if user wants to customize a legend after the plot has been drawn.
col	Optional. A color vector for predicted value lines (and intervals if requested). If not specified, the R's builtin palette() will be used. User may supply a vector of valid color names, either explicitly c("pink","black", "gray70") or implicitly, rainbow(10) or gray.colors(5). Color names will be recycled if there are more focal values of modx than colors provided.
llwd	Optional, default = 2. Line widths for predicted values. Can be single value or a vector, which will be recycled as necessary.
opacity	Optional, default = 100. A number between 1 and 255. 1 means "transparent" or invisible, 255 means very dark. the darkness of confidence interval regions

Details

This function works well with lm models in which the predictor formula includes interactions, but it does not work well with nonlinear predictors (log(x) and poly(x)). For that, please use plotCurves. plotSlopes is needed only when one wants to create an output object that can be used as input for testSlopes().

The argument plotx is the name of the horizontal plotting variable; it must be numeric. The argument modx is the moderator variable. It may be either a numeric or a factor variable. As of version 1.7, the modx argument may be omitted. A single predicted value line will be drawn. That version also introduced the arguments interval and n.

There are many ways to specify focal values using the arguments `modxVals` and `n`. This changed in rockchalk-1.7.0. If `modxVals` is omitted, a default algorithm will be used, selecting `n` values for plotting. `modxVals` may be a vector of values (for a numeric moderator) or levels (for a factor). If modxVals is a vector of values, then the argument `n` is ignored. However, if modxVals is one of the name of one of the algorithms, "table", "quantile", or "std.dev.", then the argument `n` sets number of focal values to be selected. For numeric `modx`, `n` defaults to 3, but for factors `modx` will be the number of observed values of `modx`. If modxVals is omitted, the defaults will be used ("table" for factors, "quantile" for numeric variables).

For the predictors besides `modx` and `plotx` (the ones that are not explicitly included in the plot), predicted values are calculated with variables set to the mean and mode, for numeric or factor variables (respectively). Those values can be reviewed in the newdata object that is created as a part of the output from this function

Value

Creates a plot and an output object that summarizes it.

The return object includes the "newdata" object that was used to create the plot, along with the "modxVals" vector, the values of the moderator for which lines were drawn, and the color vector. It also includes the call that generated the plot.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

References

Aiken, L. S. and West, S.G. (1991). Multiple Regression: Testing and Interpreting Interactions. Newbury Park, Calif: Sage Publications.

Cohen, J., Cohen, P., West, S. G., and Aiken, L. S. (2002). Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences (Third.). Routledge Academic.

See Also

`testSlopes` `plotCurves`

Examples

```
## Manufacture some predictors
set.seed(12345)

dat <- genCorrelatedData2 (N = 100, means = rep(0,4), sds = 1, rho = 0.2,
                           beta = c(0.3, 0.5, -0.45, 0.5, -0.1, 0, 0.6),
                           stde = 2)

dat$xcat1 <- gl(2, 50, labels = c("M", "F"))
dat$xcat2 <- cut(rnorm(100), breaks = c(-Inf, 0, 0.4, 0.9, 1, Inf),
                 labels = c("R", "M", "D", "P", "G"))
## incorporate effect of categorical predictors
dat$y <- dat$y + 1.9 * dat$x1 * contrasts(dat$xcat1)[dat$xcat1] +
          contrasts(dat$xcat2)[dat$xcat2 , ] %*% c(0.1, -0.1, 0, 0.2)
```

```
m1 <- lm(y ~ x1 * x2 + x3 + x4 + xcat1 + xcat2, data = dat)
summary(m1)

## New in rockchalk 1.7.x. No modx required:
plotSlopes(m1, plotx = "x1")

## Confidence interval, anybody?
plotSlopes(m1, plotx = "x1", interval = "conf")

## Prediction interval.
plotSlopes(m1, plotx = "x1", interval = "pred")

## Now experiment with a moderator variable
## let default quantile algorithm do its job
plotSlopes(m1, plotx = "x1", modx = "x2")
## previous uses default equivalent to
## plotSlopes(m1, plotx = "x1", modx = "x2", modxVals = "quantile")
## Want more focal values?
plotSlopes(m1, plotx = "x1", modx = "x2", n = 5)
## Pick focal values yourself?
plotSlopes(m1, plotx = "x1", modx = "x2", modxVals = c(-2, 0, 0.5))
## Alternative algorithm?
plotSlopes(m1, plotx = "x1", modx = "x2", modxVals = "std.dev.",
           main = "Uses \"std.dev.\" Divider for the Moderator",
           xlab = "My Predictor", ylab = "Write Anything You Want for ylab")

## Will catch output object from this one
m1ps <- plotSlopes(m1, plotx = "x1", modx = "x2", modxVals = "std.dev.", n = 5,
                main = "Setting n = 5 Selects More Focal Values for Plotting")

m1ts <- testSlopes(m1ps)

plot(m1ts)

### Examples with categorical Moderator variable

m3 <- lm (y ~ x1 + xcat1, data = dat)
summary(m3)
plotSlopes(m3, modx = "xcat1", plotx = "x1")
plotSlopes(m3, modx = "xcat1", plotx = "x1", interval = "predict")


m4 <- lm (y ~ x1 * xcat1, data = dat)
summary(m4)
plotSlopes(m4, modx = "xcat1", plotx = "x1")
plotSlopes(m4, modx = "xcat1", plotx = "x1", interval = "conf")


m5 <- lm (y ~ x1 + x2 + x1 * xcat2, data = dat)
summary(m5)
plotSlopes(m5, modx = "xcat2", plotx = "x1")
m5ps <- plotSlopes(m5, modx = "xcat2", plotx = "x1", interval = "conf")
```

```
testSlopes(m5ps)



## Now examples with real data. How about Chilean voters?
library(car)
m6 <- lm(statusquo ~ income * sex, data = Chile)
summary(m6)
plotSlopes(m6, modx = "sex", plotx = "income")
plotSlopes(m6, modx = "sex", plotx = "income", col = c("yellow", "blue"))


m7 <- lm(statusquo ~ region * income, data= Chile)
summary(m7)
plotSlopes(m7, plotx = "income", modx = "region")

plotSlopes(m7, plotx = "income", modx = "region", plotPoints = FALSE)
plotSlopes(m7, plotx = "income", modx = "region", plotPoints = FALSE,
           interval = "conf")
plotSlopes(m7, plotx = "income", modx = "region", modxVals = c("SA","S", "C"),
           plotPoints = FALSE, interval = "conf")
## Same, choosing 3 most frequent values
plotSlopes(m7, plotx = "income", modx = "region", n = 3, plotPoints = FALSE,
           interval = "conf")


m8 <- lm(statusquo ~ region * income + sex + age, data= Chile)
summary(m8)
plotSlopes(m8, modx = "region", plotx = "income")


m9 <- lm(statusquo ~ income * age + education + sex + age, data = Chile)
summary(m9)
plotSlopes(m9, modx = "income", plotx = "age")

plotSlopes(m9, modx = "income", plotx = "age", plotPoints = FALSE)


## Convert education to numeric, for fun
Chile$educationn <- as.numeric(Chile$education)
m10 <- lm(statusquo ~ income * educationn + sex + age, data = Chile)
summary(m10)
plotSlopes(m10, plotx = "educationn",  modx = "income")

## Now, the occupational prestige data. Please note careful attention
## to consistency of colors selected
data(Prestige)
m11 <- lm(prestige ~ education * type, data = Prestige)

plotSlopes(m11, plotx = "education", modx = "type", interval = "conf")
dev.new()
plotSlopes(m11, plotx = "education", modx = "type",
```

```
            modxVals = c("prof"), interval = "conf")
dev.new()
plotSlopes(m11, plotx = "education", modx = "type",
            modxVals = c("bc"), interval = "conf")
dev.new()
plotSlopes(m11, plotx = "education", modx = "type",
            modxVals = c("bc", "wc"), interval = "conf")
```

`predictCI` *Calculate a predicted value matrix (fit, lwr, upr) for a regression, either lm or glm, on either link or response scale.*

Description

This adapts code from predict.glm and predict.lm. I eliminated type = "terms" from consideration.

Usage

```
predictCI(object, newdata = NULL, type = c("response", "link"),
  interval = c("none", "confidence", "prediction"), dispersion = NULL,
  scale = NULL, na.action = na.pass, level = 0.95, ...)
```

Arguments

`object`	Regression object, class must include glm or lm.
`newdata`	Data frame including focal values for predictors
`type`	One of c("response", "link"), defaults to former.
`interval`	One of c("none", "confidence", "prediction"). "prediction" is defined only for lm objects, not for glm.
`dispersion`	Will be estimated if not provided. The variance coefficient of the glm, same as scale squared. Dispersion is allowed as an argument in predict.glm.
`scale`	The square root of dispersion. In an lm, this is the RMSE, called sigma in summary.lm.
`na.action`	What to do with missing values
`level`	0.95 or whatever confidence level one desires.
`...`	Other arguments to be passed to predict

Details

R's predict.glm does not have an interval argument. There are about 50 methods to calculate CIs for predicted values of GLMs, that's a major worry. This function takes the simplest route, calculating the (fit, lwr, upr) in the linear predictor scale, and then if type= "response", those 3 columns are put through linkinv(). This is the same method that SAS manuals suggest they use, same as Ben Bolker suggests in r-help (2010). I'd rather use one of the fancy tools like Edgeworth expansion, but that R code is not available (but is promised).

Use predict.lm with se.fit = TRUE to calculate fit and se.fit. Then calculate lwr and upr as fit +/- tval * se.fit. If model is lm, the model df.residual will be used to get tval. If glm, this is a normal approximation, so we thugishly assert tval = 1.98.

There's some confusing term translation. I wish R lm and glm would be brought into line. For lm, residual.scale = sigma. For glm, residual.scale = sqrt(dispersion)

Value

c(fit, lwr, upr), and possibly more.

`predictOMatic` *Create predicted values after choosing values of predictors. Can demonstrate marginal effects of the predictor variables.*

Description

It creates "newdata" frames which are passed to predict. The key idea is that each predictor has certain focal values on which we want to concentrate. We want a more-or-less easy way to spawn complete newdata objects along with fitted values. The `newdata` function creates those objects, its documentation might be helpful in understanding some nuances.

Usage

```
predictOMatic(model = NULL, predVals = "margins", divider = "quantile",
  n = 5, ...)
```

Arguments

`model`	Required. A fitted regression model. A `predict` method must exist for that model.
`predVals`	Optional. How to choose predictor values? Can be as simple as a keyword "auto" or "margins". May also be very fine-grained detail, including 1) a vector of variable names (for which values will be automatically selected) 2) a named vector of variable names and divider functions, or 3) a list naming variables and values. See details and examples.
`divider`	An algorithm name from c("quantile", "std.dev", "seq", "table") or a user-provided function. This sets the method for selecting values of the predictor. Documentation for the rockchalk methods can be found in the functions `cutByQuantile`, `cutBySD`, `plotSeq`, and `cutByTable`,.
`n`	Default = 5. The number of values for which predictions are sought.
`...`	Optional arguments to be passed to the predict function. In particular, the arguments se.fit and interval are extracted from ... and used to control the output.

Details

If no predVals argument is supplied (same as `predVals = "margins"`, predictOMatic creates a list of new data frames, one for each predictor variable. It uses the default divider algorithm (see the divider argument) and it estimates predicted values for `n` different values of the predictor. A model with formula `y ~ x1 + x2 + x3` will cause 3 separate output data frames, one for each predictor. They will be named objects in the list.

The default approach will have marginal tables, while the setting `predVals = "auto"` will create a single large newdata frame that holds the Cartesian product of the focal values of each predictor.

`predVals` may be a vector of variable names, or it may be a list of names and particular values. Whether a vector or a list is supplied, `predVals` must name only predictors that are fitted in the model. `predictOMatic` will choose the mean or mode for variables that are not explicitly listed, and selected values of the named variables are "mixed and matched" to make a data set. There are many formats in which it can be supplied. Suppose a regression formula is `y1 ~ sex + income + health + height`. The simplest format for predVals will be a vector of variable names, leaving the selection of detailed values to the default algorithms. For example, `predVals = c("income","height")` will cause sex and health to be set at central values and income and height will have target values selected according to the divider algorithm (see the argument `divider`).

The user can spcecify divider algoriths to choose focal values, `predvals = c(income = "quantile", height = "std.dev.` The dividers provided by the rockchalk package are "quantile", "std.dev.", "seq" and "table". Those are discussed more completely in the help for `focalVals`. The appropriate algorithms will select focal values of the predictors and they will supply `n` values for each in a "mix and match" data frame. After rockchalk 1.7.2, the divider argument can also be the name of a function, such as R's pretty.

Finally, users who want very fine grained control over predictOMatic can supply a named list of predictor values. For example, `predVals = list(height = c(5.5, 6.0, 6.5),income = c(10, 20, 30, 40, 50), sex =` One can also use algorithm names, `predVals = list(height = c(5.5, 6.0, 6.5), income = "quantile")` and so forth. Examples are offered below.

The variables named in the `predVals` argument should be the names of the variables in the raw data frame, not the names that R creates when it interprets a formula. We want "x", not the transformation in the functions (not `log(x)`, or `as.factor(x)` or `as.numeric(x)`). If a formula has a predictor `poly(height, 3)`, then the predVals argument should refer to height, not `poly(height, 3)`. I've invested quite a bit of effort to make sure this "just works" (many alternative packages that calculate predicted values do not).

It it important to make sure that diagnostic plots and summaries of predictions are calculated with the exact same data that was used to fit the model. This is surprisingly difficult because formulas can include things like log(income + d) and so forth. The function `model.data` is the magic bullet for that part of the problem.

Here is one example sequence that fits a model, discerns some focal values, and then uses predictOMatic.

```
 d <- 3 alpha <- 13 m1 <- lm(yout ~ xin + xout + poly(xother,2) + log(xercise + alpha), data = dat)
m1dat <- model.data(m1)
```

Now, when you are thinking about which values you might like to specify in predVals, use m1dat to decide. Try

```
summarize(m1dat)
```

Then run something like

```
predictOMatic( m1, predVals = list(xin = median(m1dat$xin), xout = c(1,2,3), xother = quantile(m1dat$x
```

Get the idea?

Value

A data frame or a list of data frames.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
library(rockchalk)

## Replicate some R classics.  The budworm.lg data from predict.glm
## will work properly after re-formatting the information as a data.frame:

## example from Venables and Ripley (2002, pp. 190-2.)
df <- data.frame(ldose = rep(0:5, 2),
                 sex = factor(rep(c("M", "F"), c(6, 6))),
                 SF.numdead = c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16))
df$SF.numalive = 20 - df$SF.numdead

budworm.lg <- glm(cbind(SF.numdead, SF.numalive) ~ sex*ldose,
                  data = df,  family = binomial)

predictOMatic(budworm.lg)

predictOMatic(budworm.lg, n = 7)

predictOMatic(budworm.lg, predVals = c("ldose"), n = 7)

predictOMatic(budworm.lg, predVals = c(ldose = "std.dev.", sex = "table"))



## Now make up a data frame with several numeric and categorical predictors.

set.seed(12345)
N <- 100
x1 <- rpois(N, l = 6)
x2 <- rnorm(N, m = 50, s = 10)
x3 <- rnorm(N)
xcat1 <- gl(2,50, labels = c("M","F"))
xcat2 <- cut(rnorm(N), breaks = c(-Inf, 0, 0.4, 0.9, 1, Inf),
             labels = c("R", "M", "D", "P", "G"))
dat <- data.frame(x1, x2, x3, xcat1, xcat2)
rm(x1, x2, x3, xcat1, xcat2)
dat$xcat1n <- with(dat, contrasts(xcat1)[xcat1, , drop = FALSE])
dat$xcat2n <- with(dat, contrasts(xcat2)[xcat2, ])
```

```
STDE <- 15
dat$y <- with(dat,
              0.03 + 0.8*x1 + 0.1*x2 + 0.7*x3 + xcat1n %*% c(2) +
              xcat2n %*% c(0.1,-2,0.3, 0.1) + STDE*rnorm(N))
## Impose some random missings
dat$x1[sample(N, 5)] <- NA
dat$x2[sample(N, 5)] <- NA
dat$x3[sample(N, 5)] <- NA
dat$xcat2[sample(N, 5)] <- NA
dat$xcat1[sample(N, 5)] <- NA
dat$y[sample(N, 5)] <- NA
summarize(dat)


m0 <- lm(y ~ x1 + x2 + xcat1, data = dat)
summary(m0)
## The model.data() function in rockchalk creates as near as possible
## the input data frame.
m0.data <- model.data(m0)
summarize(m0.data)

## no predVals: analyzes each variable separately
(m0.p1 <- predictOMatic(m0))

## requests confidence intervals from the predict function
(m0.p2 <- predictOMatic(m0, interval = "confidence"))

## predVals as vector of variable names: gives "mix and match" predictions
(m0.p3 <- predictOMatic(m0, predVals = c("x1", "x2")))

## predVals as vector of variable names: gives "mix and match" predictions
(m0.p3s <- predictOMatic(m0, predVals = c("x1", "x2"), divider = "std.dev."))

## "seq" is an evenly spaced sequence across the predictor.
(m0.p3q <- predictOMatic(m0, predVals = c("x1", "x2"), divider = "seq"))

(m0.p3i <- predictOMatic(m0, predVals = c("x1", "x2"),
                         interval = "confidence", n = 3))

(m0.p3p <- predictOMatic(m0, predVals = c("x1", "x2"), divider = pretty))

## predVals as vector with named divider algorithms.
(m0.p3 <- predictOMatic(m0, predVals = c(x1 = "seq", x2 = "quantile")))
## predVals as named vector of divider algorithms

## same idea, decided to double-check
(m0.p3 <- predictOMatic(m0, predVals = c(x1 = "quantile", x2 = "std.dev.")))
getFocal(m0.data$x2, xvals =  "std.dev.", n = 5)


## Change from quantile to standard deviation divider
(m0.p5 <- predictOMatic(m0, divider = "std.dev.", n = 5))
```

```
## Still can specify particular values if desired
(m0.p6 <- predictOMatic(m0, predVals = list("x1" = c(6,7),
                            "xcat1" = levels(m0.data$xcat1))))

(m0.p7 <- predictOMatic(m0, predVals = c(x1 = "quantile", x2 = "std.dev.")))
getFocal(m0.data$x2, xvals =  "std.dev.", n = 5)

(m0.p8 <- predictOMatic(m0, predVals = list( x1 = quantile(m0.data$x1,
                        na.rm = TRUE, probs = c(0, 0.1, 0.5, 0.8,
                        1.0)), xcat1 = levels(m0.data$xcat1))))

(m0.p9 <- predictOMatic(m0, predVals = list(x1 = "seq", "xcat1" =
                                levels(m0.data$xcat1)), n = 8) )


(m0.p10 <- predictOMatic(m0, predVals = list(x1 = "quantile",
                                 "xcat1" = levels(m0.data$xcat1)), n =  5) )


(m0.p11 <- predictOMatic(m0, predVals = c(x1 = "std.dev."), n = 10))

## Previous same as

(m0.p11 <- predictOMatic(m0, predVals = c(x1 = "default"), divider =
                         "std.dev.", n = 10))

## Previous also same as

(m0.p11 <- predictOMatic(m0, predVals = c("x1"), divider = "std.dev.", n = 10))


(m0.p11 <- predictOMatic(m0,  predVals = list(x1 = c(0, 5, 8), x2 = "default"),
                         divider = "seq"))



m1 <- lm(y ~ log(10+x1) + sin(x2) + x3, data = dat)
m1.data <- model.data(m1)
summarize(m1.data)


(newdata(m1))
(newdata(m1, predVals = list(x1 = c(6, 8, 10))))
(newdata(m1, predVals = list(x1 = c(6, 8, 10), x3 = c(-1,0,1))))
(newdata(m1, predVals = list(x1 = c(6, 8, 10),
                x2 = quantile(m1.data$x2, na.rm = TRUE), x3 = c(-1,0,1))))

(m1.p1 <- predictOMatic(m1, divider = "std.dev", n = 5))
(m1.p2 <- predictOMatic(m1, divider = "quantile", n = 5))

(m1.p3 <- predictOMatic(m1, predVals = list(x1 = c(6, 8, 10),
                            x2 = median(m1.data$x2, na.rm = TRUE))))
```

```
(m1.p4 <- predictOMatic(m1, predVals = list(x1 = c(6, 8, 10),
                                x2 = quantile(m1.data$x2, na.rm = TRUE))))

(m1.p5 <- predictOMatic(m1))
(m1.p6 <- predictOMatic(m1, divider = "std.dev."))
(m1.p7 <- predictOMatic(m1, divider = "std.dev.", n = 3))
(m1.p8 <- predictOMatic(m1, divider = "std.dev.", interval = "confidence"))


m2 <- lm(y ~ x1 + x2 + x3 + xcat1 + xcat2, data = dat)
##  has only columns and rows used in model fit
m2.data <- model.data(m2)
summarize(m2.data)

## Check all the margins
(predictOMatic(m2, interval = "conf"))

## Lets construct predictions the "old fashioned way" for comparison

m2.new1 <- newdata(m2, predVals = list(xcat1 = levels(m2.data$xcat1),
                           xcat2 = levels(m2.data$xcat2)), n = 5)
predict(m2, newdata = m2.new1)


(m2.p1 <- predictOMatic(m2,
                        predVals = list(xcat1 = levels(m2.data$xcat1),
                            xcat2 = levels(m2.data$xcat2)),
                        xcat2 = c("M","D")))
## See? same!

## Pick some particular values for focus
m2.new2 <- newdata(m2, predVals = list(x1 = c(1,2,3), xcat2 = c("M","D")))
## Ask for predictions
predict(m2, newdata = m2.new2)


## Compare: predictOMatic generates a newdata frame and predictions in one step

(m2.p2 <- predictOMatic(m2, predVals = list(x1 = c(1,2,3),
                                xcat2 = c("M","D"))))

(m2.p3 <- predictOMatic(m2, predVals = list(x2 = c(0.25, 1.0),
                                xcat2 = c("M","D"))))

(m2.p4 <- predictOMatic(m2, predVals = list(x2 = plotSeq(m2.data$x2, 10),
                                xcat2 = c("M","D"))))

(m2.p5 <- predictOMatic(m2, predVals = list(x2 = c(0.25, 1.0),
                                xcat2 = c("M","D")), interval = "conf"))

(m2.p6 <- predictOMatic(m2, predVals = list(x2 = c(49, 51),
                                xcat2 = levels(m2.data$xcat2),
                                x1 = plotSeq(dat$x1))))
```

```
plot(y ~ x1, data = m2.data)
by(m2.p6, list(m2.p6$xcat2), function(x) {
    lines(x$x1, x$fit, col = x$xcat2, lty = as.numeric(x$xcat2))
})

m2.newdata <- newdata(m2, predVals = list(x2 = c(48, 50, 52),
                              xcat2 = c("M","D")))
predict(m2, newdata = m2.newdata)

(m2.p7 <- predictOMatic(m2, predVals = list(x2 = c(48, 50, 52),
                                xcat2 = c("M","D"))))

(m2.p8 <- predictOMatic(m2,
             predVals = list(x2 = range(m2.data$x2, na.rm = TRUE),
             xcat2 = c("M","D"))))

(m2.p9 <- predictOMatic(m2, predVals = list(x2 = plotSeq(m2.data$x2),
             x1 = quantile(m2.data$x1, pr =c(0.33, 0.66), na.rm = TRUE),
             xcat2 = c("M","D"))))
plot(y ~ x2 , data = m2.data)

by(m2.p9, list(m2.p9$x1, m2.p9$xcat2), function(x) {lines(x$x2, x$fit)})



(predictOMatic(m2, predVals = list(x2 = c(50, 60), xcat2 = c("M","D")),
               interval = "conf"))

## create a dichotomous dependent variable
y2 <- ifelse(rnorm(N) > 0.3, 1, 0)
dat <- cbind(dat, y2)

m3 <- glm(y2 ~ x1 + x2 + x3 + xcat1, data = dat, family = binomial(logit))
summary(m3)
m3.data <- model.data(m3)
summarize(m3.data)

(m3.p1 <- predictOMatic(m3, divider = "std.dev."))

(m3.p2 <- predictOMatic(m3, predVals = list(x2 = c(40, 50, 60),
                             xcat1 = c("M","F")),
                        divider = "std.dev.", interval = "conf"))

## Want a full accounting for each value of x2?
(m3.p3 <- predictOMatic(m3,
                predVals = list(x2 = unique(m3.data$x2),
                    xcat1 = c("M","F")), interval = "conf"))


## Would like to write a more beautiful print method
## for output object, but don't want to obscure structure from user.
## for (i in names(m3.p1)){
```

```
##     dns <- cbind(m3.p1[[i]][i], m3.p1[[i]]$fit)
##     colnames(dns) <- c(i, "predicted")
##     print(dns)
## }
```

`print.factorSummaries` *Prints out the contents of an object created by summarizeFactors in the style of base::summary*

Description

An object with class "factorSummaries" is the input. Such an object should be created with the function rockchalk::summarizeFactors. Each element in that list is then organized for printing in a tabular summary. This should look almost like R's own summary function, except for the additional information that these factor summaries include.

Usage

```
## S3 method for class 'factorSummaries'
print(x, ...)
```

Arguments

x	A factorSummaries object produced by summarizeFactors
...	optional arguments. Only value currently used is digits.

Value

A table of formatted output

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

See Also

summary and summarize, summarizeFactors

`religioncrime` *Religious beliefs and crime rates*

Description

The data national-level summary indicators of public opinion about the existence of heaven and hell as well as the national rate of violent crime.

Usage

```
data(religioncrime)
```

Format

data.frame: 51 obs. of 3 variables

Author(s)

Paul E. Johnson <pauljohn@ku.edu> and Anonymous

Source

Anonymous researcher who claims the data is real.

Examples

```
require(rockchalk)
data(religioncrime)
mod1 <- lm(crime ~ heaven, data=religioncrime)
mod2 <- lm(crime ~ hell, data=religioncrime)
mod3 <- lm(crime ~ heaven + hell, data=religioncrime)
with(religioncrime,
mcGraph1(heaven, hell, crime)
)
with(religioncrime,
mcGraph2(heaven, hell, crime)
)
mod1 <- with(religioncrime,
mcGraph3(heaven, hell, crime)
)
summary(mod1[[1]])
##TODO: Draw more with perspective matrix mod1[[2]]
```

`residualCenter` *Calculates a "residual-centered" interaction regression.*

Description

Given a fitted `lm`, this function scans for coefficients estimated from "interaction terms" by checking for colon symbols. The function then calculates the "residual centered" estimate of the interaction term and replaces the interaction term with that residual centered estimate. It works for any order of interaction, unlike other implementations of the same approach. See also function `lmres` in package pequod.

Calculates predicted values of residual centered interaction regressions estimated in any type of regression framework (lm, glm, etc).

Usage

```
residualCenter(model)

## Default S3 method:
residualCenter(model)

## S3 method for class 'rcreg'
predict(object, ...)
```

Arguments

`model`	A fitted lm object
`object`	Fitted residual-centered regression from residualCenter
`...`	Other named arguments. May include newdata, a dataframe of predictors. That should include values for individual predictor, need not include interactions that are constructed by residualCenter. These parameters that will be passed to the predict method of the model.

Value

a regression model of the type as the input model, with the exception that the residualCentered predictor is used in place of the original interaction. The return model includes new variable centeringRegressions: a list including each of the intermediate regressions that was calculated in order to create the residual centered interaction terms. These latter objects may be necessary for diagnostics and to calculate predicted values for hypothetical values of the inputs. If there are no interactive terms, then NULL is returned.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

References

Little, T. D., Bovaird, J. A., & Widaman, K. F. (2006). On the Merits of Orthogonalizing Powered and Product Terms: Implications for Modeling Interactions Among Latent Variables. Structural Equation Modeling, 13(4), 497-519.

See Also

lmres

Examples

```
set.seed(123)
x1 <- rnorm(100)
x2 <- rnorm(100)
x3 <- rnorm(100)
x4 <- rnorm(100)
y <- rnorm(100)
dat <- data.frame(y, x1,x2,x3,x4)
rm(x1,x2,x3,x4,y)
m1 <- lm(y~ x1*x2 + x4, data = dat)

m1RC <- residualCenter(m1)

m1RCs <- summary(m1RC)
## The stage 1 centering regressions can be viewed as well
## lapply(m1RC$rcRegressions, summary)

## Verify residualCenter() output against the manual calculation
dat$x1rcx2 <- as.numeric(resid(lm(I(x1*x2) ~ x1 + x2, data = dat)))
m1m <- lm(y ~ x1 + x2 + x4 + x1rcx2, data=dat)
summary(m1m)
cbind("residualCenter" = coef(m1RC), "manual" = coef(m1m))


m2 <- lm(y~ x1*x2*x3 + x4, data=dat)
m2RC <- residualCenter(m2)
m2RCs <- summary(m2RC)

## Verify that result manually
dat$x2rcx3 <- as.numeric(resid(lm(I(x2*x3) ~ x2 + x3, data = dat)))
dat$x1rcx3 <- as.numeric(resid(lm(I(x1*x3) ~ x1 + x3, data = dat)))
dat$x1rcx2rcx3 <- as.numeric( resid(lm(I(x1*x2*x3) ~ x1 + x2 + x3 + x1rcx2 +
                                    x1rcx3 + x2rcx3 , data=dat)))
(m2m <- lm(y ~ x1 + x2 + x3+ x4 + x1rcx2 + x1rcx3 + x2rcx3 + x1rcx2rcx3,
           data = dat))

cbind("residualCenter" = coef(m2RC), "manual" = coef(m2m))


### As good as pequod's lmres
### not run because pequod generates R warnings
###
```

```
### if (require(pequod)){
###  pequodm1 <- lmres(y ~ x1*x2*x3 + x4, data=dat)
###  pequodm1s <- summary(pequodm1)
###  coef(pequodm1s)
### }

### Works with any number of interactions. See:

m3 <- lm(y~ x1*x2*x3*x4, data=dat)
m3RC <- residualCenter(m3)
summary(m3RC)
##'
## Verify that one manually (Gosh, this is horrible to write out)
dat$x1rcx4 <- as.numeric(resid(lm(I(x1*x4) ~ x1 + x4, data=dat)))
dat$x2rcx4 <- as.numeric(resid(lm(I(x2*x4) ~ x2 + x4, data=dat)))
dat$x3rcx4 <- as.numeric(resid(lm(I(x3*x4) ~ x3 + x4, data=dat)))
dat$x1rcx2rcx4 <- as.numeric(resid(lm(I(x1*x2*x4) ~ x1 + x2 + x4 +
                                      x1rcx2 + x1rcx4 + x2rcx4, data=dat)))
dat$x1rcx3rcx4 <- as.numeric(resid(lm(I(x1*x3*x4) ~ x1 + x3 + x4 +
                                      x1rcx3 + x1rcx4 + x3rcx4, data=dat)))
dat$x2rcx3rcx4 <- as.numeric(resid(lm(I(x2*x3*x4) ~ x2 + x3 + x4 +
                                      x2rcx3 + x2rcx4 + x3rcx4, data=dat)))
dat$x1rcx2rcx3rcx4 <-
    as.numeric(resid(lm(I(x1*x2*x3*x4) ~ x1 + x2 + x3 + x4 +
                        x1rcx2 + x1rcx3 + x2rcx3 + x1rcx4  + x2rcx4 +
                        x3rcx4  + x1rcx2rcx3 + x1rcx2rcx4 + x1rcx3rcx4 +
                        x2rcx3rcx4, data=dat)))
(m3m <- lm(y ~ x1 + x2 + x3 + x4 + x1rcx2 + x1rcx3 + x2rcx3 + x1rcx4 +
           x2rcx4 + x3rcx4 + x1rcx2rcx3 + x1rcx2rcx4 + x1rcx3rcx4 +
           x2rcx3rcx4 + x1rcx2rcx3rcx4, data=dat))

cbind("residualCenter"=coef(m3RC), "manual"=coef(m3m))

### If you want to fit a sequence of models, as in pequod, can do.

tm <-terms(m2)
tmvec <- attr(terms(m2), "term.labels")
f1 <- tmvec[grep(":", tmvec, invert = TRUE)]
f2 <- tmvec[grep(":.*:", tmvec, invert = TRUE)]
f3 <- tmvec[grep(":.*:.*:", tmvec, invert = TRUE)]

## > f1
## [1] "x1" "x2" "x3" "x4"
## > f2
## [1] "x1"    "x2"    "x3"    "x4"    "x1:x2" "x1:x3" "x2:x3"
## > f3
## [1] "x1"       "x2"       "x3"       "x4"       "x1:x2"    "x1:x3"    "x2:x3"
## [8] "x1:x2:x3"

f1 <- lm(as.formula(paste("y","~", paste(f1, collapse=" + "))), data=dat)
f1RC <- residualCenter(f1)
summary(f1RC)
```

```
f2 <- lm(as.formula(paste("y","~", paste(f2, collapse=" + "))), data=dat)
f2RC <- residualCenter(f2)
summary(f2RC)

f3 <- lm(as.formula(paste("y","~", paste(f3, collapse=" + "))), data=dat)
f3RC <- residualCenter(f3)
summary(f3RC)

library(rockchalk)
dat <- genCorrelatedData(1000, stde=5)

m1 <- lm(y ~ x1 * x2, data=dat)

m1mc <- meanCenter(m1)
summary(m1mc)

m1rc <- residualCenter(m1)
summary(m1rc)


newdf <- apply(dat, 2, summary)
newdf <- as.data.frame(newdf)

predict(m1rc, newdata=newdf)
```

`standardize` *Estimate standardized regression coefficients for all variables*

Description

This is brain-dead standardization of all variables in the design matrix. It mimics the silly output of SPSS, which standardizes all regressors, even if they represent categorical variables.

Usage

```
standardize(model)

## S3 method for class 'lm'
standardize(model)
```

Arguments

`model`	a fitted lm object

Value

an lm fitted with the standardized variables

a standardized regression object

Author(s)

Paul Johnson <pauljohn@ku.edu>

See Also

`meanCenter` which will center or re-scale only numberic variables

Examples

```
library(rockchalk)
N <- 100
dat <- genCorrelatedData(N = N, means = c(100,200), sds = c(20,30), rho = 0.4, stde = 10)
dat$x3 <- rnorm(100, m = 40, s = 4)

m1 <- lm(y ~ x1 + x2 + x3, data = dat)
summary(m1)

m1s <- standardize(m1)
summary(m1s)



m2 <- lm(y ~ x1 * x2 + x3, data = dat)
summary(m2)

m2s <- standardize(m2)
summary(m2s)

m2c <- meanCenter(m2)
summary(m2c)
```

---

summarize — *Sorts numeric from factor variables and returns separate summaries for those types of variables.*

Description

The work is done by the functions `summarizeNumerics` and `summarizeFactors`. Please see the help pages for those functions for complete details.

Usage

```
summarize(dat, ...)
```

Arguments

dat	A data frame
...	Optional arguments that are passed to summarizeNumerics and summarizeFactors. These may be used: maxLevels The maximum number of levels that will be reported. alphaSort If TRUE (default), the columns are re-organized in alphabetical order. If FALSE, they are presented in the original order. digits integer, used for number formatting output.

Value

A list with 2 objects, numerics and factors. numerics is a matrix of summary information, while factors is a list of factor summaries.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

Examples

```
library(rockchalk)


set.seed(23452345)
N <- 100
x1 <- gl(12, 2, labels = LETTERS[1:12])
x2 <- gl(8, 3, labels = LETTERS[12:24])
x1 <- sample(x = x1, size=N, replace = TRUE)
x2 <- sample(x = x2, size=N, replace = TRUE)
z1 <- rnorm(N)
a1 <- rnorm(N, mean = 1.2, sd = 1.7)
a2 <- rpois(N, lambda = 10 + a1)
a3 <- rgamma(N, 0.5, 4)
b1 <- rnorm(N, mean = 1.3, sd = 1.4)
dat <- data.frame(z1, a1, x2, a2, x1, a3, b1)
summary(dat)


summarize(dat)


summarizeNumerics(dat)
summarizeFactors(dat, maxLevels = 5)

summarize(dat, alphaSort = FALSE)

summarize(dat, digits = 6, alphaSort = FALSE)

summarize(dat, digits = 22, alphaSort = FALSE)

summarize(dat, maxLevels = 2)
```

```
datsumm <- summarize(dat)

datsumm$numerics
datsumm[[1]]  ## same: gets numerics

datsumm$factors
datsumm[[2]]


## Use numerics output to make plots. First,
## transpose gives varnames x summary stat matrix
datsummNT <- t(datsumm$numerics)
datsummNT <- as.data.frame(datsummNT)

plot(datsummNT$mean, datsummNT$var, xlab = "The Means",
    ylab = "The Variances")

plot(datsummNT$mean, datsummNT$var, xlab = "The Means",
    ylab = "The Variances", type = "n")
text(datsummNT$mean, datsummNT$var, labels = rownames(datsummNT))

## Here's a little plot wrinkle.  Note variable names are 'out to the
##  edge' of the plot. If names are longer they don't stay inside
##  figure. See?

## Make the variable names longer

rownames(datsummNT)
rownames(datsummNT) <- c("boring var", "var with long name",
    "tedious name var", "stupid varname", "buffoon not baboon")
plot(datsummNT$mean, datsummNT$var, xlab = "The Means",
    ylab = "The Variances", type = "n")
text(datsummNT$mean, datsummNT$var, labels = rownames(datsummNT),
    cex = 0.8)
## That's no good. Names across the edges

## We could brute force the names outside the edges like
##  this
par(xpd = TRUE)
text(datsummNT$mean, datsummNT$var, labels = rownames(datsummNT),
    cex = 0.8)
## but that is not much better
par(xpd = FALSE)

## Here is one fix. Make the unused space inside the plot
##  larger by
## making xlim and ylim bigger.  I use the magRange
##  function from
## rockchalk to easily expand range to 1.2 times its
##  current size.
## otherwise, long variable names do not fit inside plot.
##  magRange
## could be asymmetric if we want, but this use is
```

```
##  symmetric.

rownames(datsummNT)
rownames(datsummNT) <- c("boring var", "var with long name",
    "tedious name var", "stupid varname", "buffoon not baboon")
plot(datsummNT$mean, datsummNT$var, xlab = "The Means",
    ylab = "The Variances", type = "n", xlim = magRange(datsummNT$mean,
        1.2), ylim = magRange(datsummNT$var, 1.2))
text(datsummNT$mean, datsummNT$var, labels = rownames(datsummNT),
    cex = 0.8)

## Here's another little plot wrinkle.  If we don't do that to keep
## the names in bounds, we need some fancy footwork.  Note when a
## point is near the edge, I make sure the text prints toward the
## center of the graph.
plot(datsummNT$mean, datsummNT$var, xlab = "The Means",
    ylab = "The Variances")
## calculate label positions. This is not as fancy as it could be.  If
##  there were lots of variables, we'd have to get smarter about
##  positioning labels on above, below, left, or right.
labelPos <- ifelse(datsummNT$mean - mean(datsummNT$mean,
    na.rm = TRUE) > 0, 2, 4)
text(datsummNT$mean, datsummNT$var, labels = rownames(datsummNT),
    cex = 0.8, pos = labelPos)



x <- data.frame(x = rnorm(N), y = gl(50, 2), z = rep(1:4,
    25), ab = gl(2, 50))

summarize(x)
summarize(x, maxLevels = 15)

sumry <- summarize(x)
sumry[[1]]  ##another way to get the numerics output
sumry[[2]]  ##another way to get the factors output

dat <- data.frame(x = rnorm(N), y = gl(50, 2), z = factor(rep(1:4,
    25), labels = c("A", "B", "C", "D")), animal = factor(ifelse(runif(N) <
    0.2, "cow", ifelse(runif(N) < 0.5, "pig", "duck"))))

summarize(dat)

## Run this if you have internet access

## fn <- "http://pj.freefaculty.org/guides/stat/DataSets/USNewsCollege/USNewsCollege.csv"
## dat <- read.table(url(fn), sep = ",")

## colnames(dat) <- c("fice", "name", "state", "private", "avemath",
##                    "aveverb", "avecomb", "aveact", "fstmath",
##                    "trdmath", "fstverb", "trdverb", "fstact",
##                    "trdact", "numapps", "numacc", "numenr",
##                    "pctten", "pctquart", "numfull", "numpart",
```

```
##                     "instate", "outstate", "rmbrdcst", "roomcst",
##                     "brdcst", "addfees", "bookcst", "prsnl",
##                     "pctphd", "pctterm", "stdtofac", "pctdonat",
##                     "instcst", "gradrate")

## dat$private <- factor(dat$private, labels = c("public",
##                                     "private"))
## sumry <- summarize(dat, digits = 2)
## sumry

## sumry[[1]]
## sumry[[2]]

## summarize(dat[, c("fice", "name", "private", "fstverb",
##                   "avemath")], digits = 4)
```

`summarizeFactors` *Extracts non-numeric variables, calculates summary information, including entropy as a diversity indicator.*

Description

This function finds the non- numeric variables and ignores the others. (See `summarizeNumerics` for a function that handles numeric variables.) It then treats all non-numeric variables as if they were factors, and summarizes each. The main benefits from this compared to R's default summary are 1) more summary information is returned for each variable (entropy estimates ofdispersion), 2) the columns in the output are alphabetized. To prevent alphabetization, use alphaSort = FALSE.

Usage

```
summarizeFactors(dat = NULL, maxLevels = 5, alphaSort = TRUE,
  sumstat = TRUE, digits = max(3, getOption("digits") - 3))
```

Arguments

`dat`	A data frame
`maxLevels`	The maximum number of levels that will be reported.
`alphaSort`	If TRUE (default), the columns are re-organized in alphabetical order. If FALSE, they are presented in the original order.
`sumstat`	If TRUE (default), report indicators of dispersion and the number of missing cases (NAs).
`digits`	integer, used for number formatting output.

Details

Entropy is one possible measure of diversity. If all outcomes are equally likely, the entropy is maximized, while if all outcomes fall into one possible category, entropy is at its lowest values. The lowest possible value for entropy is 0, while the maximum value is dependent on the number of categories. Entropy is also called Shannon's information index in some fields of study (Balch, 2000 ; Shannon, 1949).

Concerning the use of entropy as a diversity index, the user might consult Balch(). For each possible outcome category, let p represent the observed proportion of cases. The diversity contribution of each category is -p * log2(p). Note that if p is either 0 or 1, the diversity contribution is 0. The sum of those diversity contributions across possible outcomes is the entropy estimate. The entropy value is a lower bound of 0, but there is no upper bound that is independent of the number of possible categories. If m is the number of categories, the maximum possible value of entropy is -log2(1/m).

Because the maximum value of entropy depends on the number of possible categories, some scholars wish to re-scale so as to bring the values into a common numeric scale. The normed entropy is calculated as the observed entropy divided by the maximum possible entropy. Normed entropy takes on values between 0 and 1, so in a sense, its values are more easily comparable. However, the comparison is something of an illusion, since variables with the same number of categories will always be comparable by their entropy, whether it is normed or not.

Warning: Variables of class POSIXt will be ignored. This will be fixed in the future. The function works perfectly well with numeric, factor, or character variables. Other more elaborate structures are likely to be trouble.

Value

A list of factor summaries

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

References

Balch, T. (2000). Hierarchic Social Entropy: An Information Theoretic Measure of Robot Group Diversity. Auton. Robots, 8(3), 209-238.

Shannon, Claude. E. (1949). The Mathematical Theory of Communication. Urbana: University of Illinois Press.

See Also

summarizeFactors and summarizeNumerics

Examples

```
set.seed(21234)
x <- runif(1000)
xn <- ifelse(x < 0.2, 0, ifelse(x < 0.6, 1, 2))
xf <- factor(xn, levels=c(0,1,2), labels("A","B","C"))
dat <- data.frame(xf, xn, x)
```

```
summarizeFactors(dat)
##see help for summarize for more examples
```

`summarizeNumerics` *Extracts numeric variables and presents an alphabetized summary in a workable format.*

Description

This function finds the numeric variables and ignores the others. (See `summarizeFactors` for a function that handles non-numeric variables.). It calculates the quantiles for each variable, as well as the mean, standard deviation, and variance, and then packs those results into a matrix. The main benefits from this compared to R's default summary are 1) more summary information is returned for each variable (dispersion), 2) the results are returned in a matrix that is easy to use in further analysis, 3) the columns in the output are alphabetized. To prevent alphabetization, use alphaSort = FALSE.

Usage

```
summarizeNumerics(dat, alphaSort = TRUE, sumstat = TRUE, digits = max(3,
  getOption("digits") - 3))
```

Arguments

`dat`	a data frame or a matrix
`alphaSort`	If TRUE (default), the columns are re-organized in alphabetical order. If FALSE, they are presented in the original order.
`sumstat`	If TRUE (default), include mean, standard deviation, and count of NAs.
`digits`	integer, used for number formatting output.

Value

a matrix with one column per variable and the rows representing the quantiles as well as the mean, standard deviation, and variance.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

See Also

summarize and summarizeFactors

summary.factor *Tabulates observed values and calculates entropy*

Description

This adapts code from R base summary.factor. It adds the calculation of entropy as a measure of diversity.

Usage

```
summary.factor(y, maxLevels = 5, sumstat = TRUE)
```

Arguments

y	a factor (non-numeric variable)
maxLevels	The maximum number of levels that will be presented in the tabulation.
sumstat	If TRUE (default), entropy (diversity) estimate and the number of NAs will be returned.

Value

a vector of named elements including the summary table as well as entropy and normed entropy.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

testSlopes *Hypothesis tests for Simple Slopes Objects*

Description

Conducts t-test of the hypothesis that the "simple slope" line for one predictor is statistically significantly different from zero for each value of a moderator variable. The user must first run `plotSlopes()`, and then give the output object to `plotSlopes()`. A plot method has been implemented for testSlopes objects. It will create an interesting display, but only when the moderator is a numeric variable.

Usage

```
testSlopes(object)
```

Arguments

object	Output from the plotSlopes function

Details

This function scans the input object to detect the focal values of the moderator variable (the variable declared as `modx` in `plotSlopes`). Consider a regression with interactions

y <- b0 + b1*x1 + b2*x2 + b3*(x1*x2) + b4*x3 + ... + error

If `plotSlopes` has been run with the argument plotx="x1" and the argument modx="x2", then there will be several plotted lines, one for each of the chosen values of x2. The slope of each of these lines depends on x1's effect, b1, as well as the interactive part, b3*x2.

This function performs a test of the null hypothesis of the slope of each fitted line in a `plotSlopes` object is statistically significant from zero. A simple t-test for each line is offered. No correction for the conduct of multiple hypothesis tests (no Bonferroni correction).

When `modx` is a numeric variable, it is possible to conduct further analysis. We ask "for which values of `modx` would the effect of `plotx` be statistically significant?" This is called a Johnson-Neyman (Johnson-Neyman, 1936) approach in Preacher, Curran, and Bauer (2006). The interval is calculated here. A plot method is provided to illustrate the result.

Value

A list including 1) the hypothesis test table, 2) a copy of the plotSlopes object, and, for numeric modx variables, 3) the Johnson-Neyman (J-N) interval boundaries.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

References

Preacher, Kristopher J, Curran, Patrick J.,and Bauer, Daniel J. (2006). Computational Tools for Probing Interactions in Multiple Linear Regression, Multilevel Modeling, and Latent Curve Analysis. Journal of Educational and Behavioral Statistics. 31,4, 437-448.

Johnson, P.O. and Neyman, J. (1936). "Tests of certain linear hypotheses and their applications to some educational problems. Statistical Research Memoirs, 1, 57-93.

See Also

plotSlopes

Examples

```
library(rockchalk)
library(car)
m1 <- lm(statusquo ~ income * age + education + sex + age, data = Chile)
m1ps <- plotSlopes(m1, modx = "income", plotx = "age")
m1psts <- testSlopes(m1ps)
plot(m1psts)


dat2 <- genCorrelatedData(N = 400, rho = .1, means = c(50, -20),
                          stde = 300, beta = c(2, 0, 0.1, -0.4))
```

```
m2 <- lm(y ~ x1*x2, data = dat2)
m2ps <- plotSlopes(m2, plotx = "x1", modx = "x2")
m2psts <- testSlopes(m2ps)
plot(m2psts)
m2ps <- plotSlopes(m2, plotx = "x1", modx = "x2", modxVals = "std.dev", n = 5)
m2psts <- testSlopes(m2ps)
plot(m2psts)

## Try again with longer variable names

colnames(dat2) <- c("oxygen","hydrogen","species")
m2a <- lm(species ~ oxygen*hydrogen, data = dat2)
m2aps1 <- plotSlopes(m2a, plotx = "oxygen", modx = "hydrogen")
m2aps1ts <- testSlopes(m2aps1)
plot(m2aps1ts)
m2aps2 <- plotSlopes(m2a, plotx = "oxygen", modx = "hydrogen",
                     modxVals = "std.dev", n = 5)
m2bps2ts <- testSlopes(m2aps2)
plot(m2bps2ts)



dat3 <- genCorrelatedData(N = 400, rho = .1, stde = 300,
                          beta = c(2, 0, 0.3, 0.15),
                          means = c(50,0), sds = c(10, 40))
m3 <- lm(y ~ x1*x2, data = dat3)
m3ps <- plotSlopes(m3, plotx = "x1", modx = "x2")
m3sts <- testSlopes(m3ps)
plot(testSlopes(m3ps))
plot(testSlopes(m3ps), shade = FALSE)

## Finally, if model has no relevant interactions, testSlopes does nothing.
m9 <- lm(statusquo ~ age + income * education + sex + age, data = Chile)
m9ps <- plotSlopes(m9, modx = "education", plotx = "age", plotPoints = FALSE)
m9psts <- testSlopes(m9ps)
```

`vech2Corr` *Convert the vech (column of strictly lower trianglar values from a matrix) into a correlation matrix.*

Description

vech2Corr is a convenience function for creating correlation matrices from a vector of the lower triangular values. It checks the arguments to make sure they are consistent with the requirements of a correlation matrix. All values must be in [-1, 1], and the number of values specified must be correct for a lower triangle.

Usage

```
vech2Corr(vech)
```

Arguments

vech	A vector of values for the strictly lower triangle of a matrix. All values must be in the [0,1] interval (because they are correlations) and the matrix formed must be positive definite.

Details

Use this in combination with the lazyCov function to convert a vector of standard deviations and the correlation matrix into a covariance matrix.

Value

A symmetric correlation matrix, with 1's on the diagonal.

Author(s)

Paul E. Johnson <pauljohn@ku.edu>

See Also

Similar functions exist in many packages, see vec2sm in corpcor, xpnd in MCMCpack

Examples

```
v <- c(0.1, 0.4, -0.5)
vech2Corr(v)
v <- c(0.1, 0.4, -0.4, 0.4, 0.5, 0.1)
vech2Corr(v)
```

---

vech2mat *Convert a half-vector (vech) into a matrix.*

Description

Fills a matrix from a vector that represents the lower triangle. If user does not supply a value for diag, then the vech will fill in the diagonal as well as the strictly lower triangle. If diag is provided (either a number or a vector), then vech is for the strictly lower triangular part. The default value for lowerOnly is FALSE, which means that a symmetric matrix will be created. See examples for a demonstration of how to fill in the lower triangle and leave the diagonal and the upper triangle empty.

Usage

```
vech2mat(vech, diag = NULL, lowerOnly = FALSE)
```

Arguments

vech	A vector
diag	Optional. A single value or a vector for the diagonal. A vech is a strictly lower triangluar vech, it does not include diagonal values. diag can be either a single value (to replace all elements along the diagonal) or a vector of the correct length to replace the diagonal.
lowerOnly	Default = FALSE.

See Also

Similar functions exist in many packages, see vec2sm in corpcor, xpnd in MCMCpack

Examples

```
x <- 1:6
vech2mat(x)
vech2mat(x, diag = 7)
vech2mat(x, diag = c(99, 98, 97, 96))
vech2mat(x, diag = 0, lowerOnly = TRUE)
```

Index