The IPSUR Package

January 13, 2007

Version 0.1-2

Date 2007-01-11

Title Introduction to Probability and Statistics Using R

Author G. Jay Kerns <gkerns@ysu.edu> with contributions by Theophilius Boye, adapted from the work of John Fox et al.

Maintainer G. Jay Kerns <gkerns@ysu.edu>

Depends R (>= 2.1.0), car (>= 1.0-15), grDevices, tcltk, utils

Suggests abind, distr, distrEx, effects (>= 1.0-7), foreign, grid, lattice, lmtest, MASS, mgcv, multcomp (>= 0.991-2), nlme, nnet, qcc, relimp, rgl, RODBC

LazyLoad no

Description This package accompanies G. Andy Chang and G. Jay Kerns, Introduction to Probability and Statistics Using R (in progress). The package contributes functions unique to the book as well as specific configuration and selected functionality to the R Commander by John Fox.

License GPL version 2 or newer

URL http://www.r-project.org, http://www.cc.ysu.edu/~gjkerns/IPSUR/package

R topics documented:

`BloodPressure` *Blood Pressure and Heart Rate Readings*

Description

These data were collected during from 2004 through 2006 by Taoying Bian.

Usage

```
data(BloodPressure)
```

Format

A data frame with 202 observations on the following 7 variables:

year year. From 2004 through 2006

month month of the year. January = 1.

day the day of the month.

hour the 24-clock hour.

systolic systolic blood pressure reading (in mm Hg).

diastolic diastolic blood pressure reading (in mm Hg).

heart.rate heart rate reading, in beats per minute.

Details

From 2004 through 2006, Mrs. Taoying Bian regularly collected data concerning her blood pressure and heart rate.

Source

These data were collected by Taoying Bian from 2004 through 2006.

`Commander` *R Commander*

Description

Start the R Commander GUI (graphical user interface)

Usage

```
Commander()
```

Details

Getting Started

The default R Commander interface consists of (from top to bottom) a menu bar, a toolbar, a script window, an output window, and a messages window.

Commands to read, write, transform, and analyze data are entered using the menus in the menu bar at the top of the *Commander* window. Most menu items lead to dialog boxes requesting further specification. I suggest that you explore the menus to see what is available.

Below the menu bar is a toolbar with (from left to right) an information field displaying the name of the active data set; buttons for editing and displaying the active data set; and an information field showing the active statistical model. There is also a *Submit* button for re-executing commands in the script window. The information fields for the active data set and active model are actually buttons that can be used to select the active data set and model from among, respectively, data frames or suitable model objects in memory.

Almost all commands require an active data set. When the Commander starts, there is no active data set, as indicated in the data set information field. A data set becomes the active data set when it is read into memory from an R package or imported from a text file, SPSS data set, Minitab data set, or STATA data set. In addition, the active data set can be selected from among R data frames resident in memory. You can therefore switch among data sets during a session.

By default, commands are logged to the script window (the initially empty text window immediately below the toolbar); commands and output appear in the output window (the initially empty text window below the script window); and the active data set is attached to the search path. To alter these and other defaults, see the information below on configuration.

Some `Rcmdr` dialogs (those in the *Statistics -> Fit models* menu) produce linear, generalized linear, or other models. When a model is fit, it becomes the active model, as indicated in the information field in the R Commander toolbar. Items in the *Models* menu apply to the active model. Initially,

there is no active model. If there are several models in memory, you can select the active model from among them.

If command logging in turned on, R commands that are generated from the menus and dialog boxes are entered into the script window in the Commander. You can edit these commands in the normal manner and can also type new commands into the script window. Individual commands can be continued over more than one line, but each line after the first must be indented with one or more spaces or tabs. The contents of the script window can be saved during or at the end of the session, and a saved script can be loaded into the script window. The contents of the output window can also be edited or saved to a text file.

To re-execute a command or set of commands, select the lines to be executed using the mouse and press the *Submit* button at the right of the toolbar (or *Control-R*, for "run"). If no text is selected, the *Submit* button (or *Control-R*) submits the line containing the text-insertion cursor. Note that an error will be generated if the submitted command or commands are incomplete.

Pressing *Control-F* brings up a find-text dialog box (which can also be accessed via *Edit -> Find*) to search for text in the script window or the output window. Edit functions such as search are performed in the script window unless you first click in the output window to make it the active window.

Pressing *Control-S* will save the script or output window.

Pressing *Control-A* selects all of the text in the script or output window.

Right-clicking the mouse (clicking button 3 on a three-button mouse) in the script or output window brings up a "context" menu with the *Edit*-menu items, plus (in the script window) a *Submit* item.

When you execute commands from the *Commander* window, you must ensure that the sequence of commands is logical. For example, it makes no sense to fit a statistical model to a data set that has not been read into memory.

Pressing a letter key (e.g., "a") in a list box will scroll the list box to bring the next entry starting with that letter to the top of the box.

Exit from the Commander via the *File -> Exit* menu or by closing the *Commander* window.

Customization and Configuration

Configuration files reside in the `etc` subdirectory of the package, or in the locations given by the `etc` and `etcMenus` options (see below).

The `Rcmdr` menus can be customized by editing the file `Rcmdr-menus.txt`.

Some functions (e.g., `hist`) that do not normally create visible printed output when executed from the *R Console* command prompt will do so — unless prevented — when executed from the *Commander* script window. Such output can be suppressed by listing the names of these functions in the `log-exceptions.txt` file.

You can add R code to the package, e.g., for creating additional dialogs, by placing files with file type `.R` in the `etc` directory, also editing `Rcmdr-menus.txt` to provide additional menus, submenus, or menu-items. A demo addition is provided in the file `BoxCox.demo`. To activate the demo, rename this file to `BoxCox.R`, and uncomment the corresponding menu line in `Rcmdr-menus.txt`. Alternatively, you can edit the source package and recompile it.

A number of functions are provided to assist in writing dialogs, and `Rcmdr` state information is stored in a separate environment. See `help("Rcmdr.Utilities")` and the manual supplied in the `doc` directory of the `Rcmdr` package for more information.

In addition, several features are controlled by run-time options, set via the `options("Rcmdr")` command. These options should be set before the package is loaded. If the options are unset, which is the usual situation, defaults are used. Specify options as a list of *name=value* pairs. You can set none, one, several, or all options. The available options are as follows:

`attach.data.set` if `TRUE` (the default is `FALSE`), the active data set is attached to the search path.

`check.packages` if `TRUE` (the default), on start-up, the presence of all of the Rcmdr recommended packages will be checked, and if any are absent, the Rcmdr will offer to install them.

`command.text.color` Color for commands in the output window; the default is `"red"`.

`console.output` If `TRUE`, output is directed to the *R Console*, and the *R Commander* output window is not displayed. The default is `FALSE`.

`contrasts` Serves the same function as the general `contrasts` option; the default is\`c("contr.Treatment", "contr.poly")`. When the Commander exits, the `contrasts` option is returned to its pre-existing value. Note that `contr.Treatment` is from the `car` package.

`crisp.dialogs` If `TRUE`, dialogs should appear on the screen fully drawn, rather than built up widget by widget. This option should affect the Windows version of R only, but should in any event be harmless. The default is `TRUE` under Windows for R versions 2.1.1 and above, and `FALSE` otherwise. If you're working on Windows and encounter increased stability problems, trying setting this option to `FALSE`.

`default.font` The default font, as an X11 font specification, given in a character string. If specified, this value takes precedence over the default font size (below). This option is only for non-Windows systems.

`default.font.size` The size, in points, of the default font. The default is 10 for Windows systems and 12 for other systems Unless otherwise specified (see the previous item), the default font is `"*helvetica-medium-r-normal-*-xx*"`, where `xx` is the default font size. This option is only for non-Windows systems.

`double.click` Set to `TRUE` if you want a double-click of the left mouse button to press the default button in all dialogs. The default is `FALSE`.

`error.text.color` Color for error messages; the default is `"red"`.

`etc` Set to the path of the directory containing the Rcmdr configuration files; defaults to the `etc` subdirectory of the installed Rcmdr package.

`etcMenus` Set to the path of the directory containing the Rcmdr menu file `Rcmdr-menus.txt`; defaults to the value given by the `etc` option.

`grab.focus` Set to `TRUE` for the current Tk window to "grab" the focus — that is, to prevent the focus from being changed to another Tk window. On some systems, grabbing the focus in this manner apparently causes problems. The default is `TRUE`. If you experience focus problems, try setting this option to `FALSE`.

`load.at.startup` A character vector of names of packages to be loaded when the `Rcmdr` package is loaded; the default is to load only the `car` package. Other required packages will be loaded as needed. If it is available, the `car` package will be loaded at when the Commander starts in any event.

`log.commands` If `TRUE` (the default), commands are echoed to the script window; if `FALSE`, the script window is not displayed.

log.font.size The font size, in points, to be used in the script window, in the output window, in recode dialogs, and in compute expressions — that is, where a monospaced font is used. The default is 10 for Windows systems and 12 for other systems.

log.height The height of the script window, in lines. The default is 10. Setting `log.height` to 0 has the same effect as setting `log.commands` to `FALSE`.

log.text.color Color for text in the script window; the default is `"black"`.

log.width The width of the script and output windows, in characters. The default is 80.

multiple.select.mode Affects the way multiple variables are selected in variable-list boxes. If set to `"extended"` (the default), left-clicking on a variable selects it and deselects any other variables that are selected; Control-left-click toggles the selection (and may be used to select additional variables); Shift-left-click extends the selection. This is the standard Windows convention. If set to `"multiple"`, left-clicking toggles the selection of a variable and may be used to select more than one variable. This is the behaviour in the Rcmdr prior to version 1.9-10.

output.height The height of the output window, in lines. The default is twice the height of the script window, or 20 if the script window is suppressed. Setting `output.height` to 0 has the same effect as setting `console.output` to `TRUE`.

output.text.color Color for output in the output window; the default is `"blue"`.

placement Placement of the *R Commander* window, in pixels; the default is `"-40+20"`, which puts the window near the upper-right corner of the screen.

suppress.X11.warnings On (some?) Linux systems, multiple X11 warnings are generated by Rcmdr commands after a graphics-device window has been opened. Set this option to `TRUE` (the default when running interactively under X11 prior to R version 2.4.0) to suppress reporting of these warnings. An undesirable side effect is that then *all* warnings and error messages are intercepted by the Rcmdr, even those for commands entered at the R command prompt. Messages produced by such commands will be printed in the Commander Messages window after the next Rcmdr-generated command. Some X11 warnings may be printed when you exit from the Commander. This problem only applies to R versions before 2.4.0, and the default value of the option is set accordingly.

retain.messages If `TRUE` (the default is codeFALSE), the contents of the message window are not erased between messages. In any event, a "NOTE" message will not erase a preceding "WARNING" or "ERROR".

scale.factor A scaling factor to be applied to all Tk elements, such as fonts. This works well only in Windows. The default is `NULL`.

showData.threshold If the number of variables in the active data set exceeds this value (default, 100), then `edit()` rather than `showData()` is used to display the data set. A disadvantage is that control doesn't return to the Commander until the edit window is closed. The reason for the option is that `showData()` is very slow when the number of variables is large; setting the threshold to 0 suppresses the use of showData altogether.

show.edit.button Set to `TRUE` (the default) if you want an *Edit* button in the Commander window, permitting you to edit the active data set. Windows users may wish to set this option to `FALSE` to suppress the *Edit* button because changing variable names in the data editor can cause R to crash (though I believe that this problem as been solved).

sort.names Set to `TRUE` (the default) if you want variable names to be sorted alphabetically in variable lists.

tkwait This option addresses a problem that, to my knowledge, is rare, and may occur on some non-Windows systems. If the Commander causes R to hang, then set the `tkwait` option to `TRUE`; otherwise set the option to `FALSE` or ignore it. An undesirable side effect of setting the `tkwait` option to `TRUE` is that the R session command prompt is suppressed until the Commander exits. One can still enter commands via the script window, however. In particular, there is no reason to use this option under Windows, and it should not be used with the Windows R GUI with buffered output when output is directed to the R console.

use.rgl If `TRUE` (the default), the `rgl` package will be loaded if it is present in an accessible library; if `FALSE`, the `rgl` package will be ignored even if it is available. The `rgl` package can sometimes cause problems when running R under X11.

warning.text.color Color for warning messages; the default is `"darkgreen"`.

Many options can also be set via the *File -> Options* menu, which will restart the Commander after options are set.

Known Problem

Occasionally, under Windows, after typing some text into a dialog box (e.g., a subsetting expression in the Subset Data Set dialog), buttons in the dialog (e.g., the OK button) will have no effect when they are pressed. Clicking anywhere inside or outside of the dialog box should restore the function of the buttons. As far as I have been able to ascertain, this is a problem with Tcl/Tk for Windows.

Note

This version is compatible with SciViews, which currently runs only under Windows systems: http://www.sciviews.org/SciViews-R; see `Rcmdr.sciviews-specific`. Under Windows, the `Rcmdr` package can also be run under the Rgui in SDI (single-document interface) mode, or under `rterm.exe`; you might experience problems running the `Rcmdr` under `ESS` with-NTEmacs or XEmacs.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

Examples

```
options(Rcmdr=list(log.font.size=12, contrasts=c("contr.Sum", "contr.poly")))
```

`Compute` *Rcmdr Compute Dialog*

Description

The compute dialog is used to compute new variables.

Details

The name of the new variable must be a valid R object name (consisting only of upper and lower-case letters, numerals, and periods, and not starting with a numeral).

Enter an R expression in the box at the right. The expression is evaluated using the active data set. You can double-click in the variable-list box to enter variable names in the expression. The expression must evaluate to a valid variable, which is added to the active data set.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

Arithmetic

Confint *Confidence Intervals for Model Coefficients*

Description

Except for glm objects, where a method is provided that provides intervals optionally based on the Wald statistic, this generic function simply calls `confint` in the `stats` package via its default method.

Usage

```
Confint(object, parm, level = 0.95, ...)

## S3 method for class 'glm':
Confint(object, parm, level=0.95, type=c("LR", "Wald"), ...)
```

Arguments

`object`	a model object.
`parm`	which parameters to use, defaults to all.
`level`	level of confidence, defaulting to 0.95.
`type`	for a glm object, confidence interval based on the profile likelihood (the default) or the Wald statistic.
`...`	arguments to be passed down to methods.

Value

dependent upon the method called.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

confint

FeedingTimes *Feeding Times of a Newborn*

Description

These data were collected during July and August, 2006 at the request of the pediatrician concerning the feeding habits of Anna Lu Kerns.

Usage

```
data(FeedingTimes)
```

Format

A data frame with 42 observations on the following 7 variables:

age.days age in days. July 1, 2006 = 1.

clock.hours the 24-clock hour.

clock.min the clock minute.

type type of food eaten, being direct breast milk, formula, pumped breast milk, or no food (rest)

amount.oz amount of food eaten, in ounces.

duration.min duration of feeding time.

time.hours sequential time in hours. Time = 0 corresponds to 8 AM, July 9th, 2006.

Details

During July and August 2006 the author collected data concerning the feeding habits of his newborn daughter, Anna Lu Kerns. The time of feeding was recorded, along with the type of food eaten. The amount of food eaten (in oz.) was recorded except when Anna was breastfeeding, in which case the duration of feeding was recorded. Some other durations were missing and others were calculated from the clock times.

Source

These data were collected by the author during July and August 2006 during observation of his newborn daughter.

Hist *Plot a Histogram*

Description

This function is a wrapper for the hist function in the base package, permitting percentage scaling of the vertical axis in addition to frequency and density scaling.

Usage

```
Hist(x, scale = c("frequency", "percent", "density"), ...)
```

Arguments

x	a vector of values for which a histogram is to be plotted.
scale	the scaling of the vertical axis: "frequency" (the default), "percent", or "density".
...	arguments to be passed to hist.

Value

This function returns NULL, and is called for its side effect — plotting a histogram.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

hist

Examples

```
data(Prestige)
Hist(Prestige$income, scale="percent")
```

IPSUR-package *Introduction to Probability and Statistics Using R*

Description

This package accompanies G. Andy Chang and G. Jay Kerns, Introduction to Probability and Statistics Using R (in progress). The package contributes functions unique to the book as well as specific configuration and selected functionality to the R Commander by John Fox.

Details

Package:	IPSUR
Version:	0.1-1
Date:	2006-10-04
Depends:	R (>= 2.1.0), grDevices, tcltk, utils
Suggests:	abind, car (>= 1.0-15), effects (>= 1.0-7), foreign, grid, lattice, lmtest, MASS, mgcv, multcomp, nlme, nnet, qcc,
LazyLoad:	no
License:	GPL version 2 or newer
URL:	http://www.r-project.org, http://www.cc.ysu.edu/ gjkerns/IPSUR/package

Author(s)

G. Jay Kerns <gkerns@ysu.edu> with contributions by Theophilius Boye, adapted from the work of John Fox et al.

Maintainer: G. Jay Kerns <gkerns@ysu.edu>

IPSUR.Utilities *IPSUR Utility Functions*

Description

These functions support writing additions to the IPSUR package, and were patterned after their Rcmdr equivalents. Additional R code can be placed in files with file type `.R` in the `etc` subdirectory of the package. Add menus, submenus, and menu items by editing the file `Rcmdr-menus.txt` in the same directory.

Usage

```
checkMultiLevelFactors(n=1)
listMultiLevelFactors(dataSet=ActiveDataSet())
MultiLevelFactors(names)
multiLevelFactorsP(n=1)
```

Arguments

`dataSet`	the quoted name of a data frame in memory.
`names`	optional names to be stored.
`n`	number of variables to check for.

Details

There are several groups of functions exported by the `Rcmdr` package and documented briefly here. To see how these functions work, it is simplest to examine the dialog-generating functions in the `Rcmdr` package.

Checking for errors: The function `checkMultiLevelFactors` checks for the existence of objects and writes an error message to the log if it is absent (or insufficiently numerous, in the case of different kinds of variables).

Information: The following function returns vectors of object names: `listMultiLevelFactors`

Author(s)

G. Jay Kerns ⟨gkerns@ysu.edu⟩

References

T. Lumley (2001) Programmer's niche: Macros in R. *R News*, **1(3)**, 11–13.

`IPSURgetAnswer` *Display answers to selected problems*

Description

Displays answers to selected problems in the system's web browser. Adapted from a function in the UsingR package by John Verzani.

Usage

```
IPSURgetAnswer(chapter = NULL, problem = NULL)
```

Arguments

`chapter`	The chapter number
`problem`	The problems number

Not all answers are available.

Details

Some selected answers from the problems in *Introduction to Probability & Statistics Using R* are available from the webpage `http://www.cc.ysu.edu/~gjkerns/IPSUR`. This function will display them one-by-one in the browser.

Value

If available, opens web browser to the requested answer.

Author(s)

G. Jay Kerns

See Also

See AlsoIPSURweb

Examples

```
IPSURgetAnswer()
```

IPSURweb *Opens browser to IPSUR webpages*

Description

Opens the browser to webpages from the IPSUR website. Adapted from a function in the UsingR package by John Verzani.

Usage

```
IPSURweb(what = c("homepage", "errata", "changes", "exercises", "package"))
```

Arguments

what	A character string indicating what page to open. No value specified will open the IPSUR homepage. Others are "errata" for typos and errors, "changes" for changes to the text brought about by new changes to R, "exercises" to go to a page containing selected answers, and "packages" for materials related to the IPSUR package.

Value

Opens the browser to the respective web page.

Author(s)

G. Jay Kerns

References

see http://www.cc.ysu.edu/~gjkerns

See Also

See AlsoIPSURgetAnswer

Examples

```
## Not run: IPSURweb()                    # main webpage
## Not run: IPSURweb("err")                       # errata
```

`KMeans` *K-Means Clustering Using Multiple Random Seeds*

Description

Finds a number of k-means clusting solutions using R's `kmeans` function, and selects as the final solution the one that has the minimum total within-cluster sum of squared distances.

Usage

```
KMeans(x, centers, iter.max=10, num.seeds=10)
```

Arguments

`x`	A numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a dataframe with all numeric columns).
`centers`	The number of clusters in the solution.
`iter.max`	The maximum number of iterations allowed.
`num.seeds`	The number of different starting random seeds to use. Each random seed results in a different k-means solution.

Value

A list with components:

`cluster`	A vector of integers indicating the cluster to which each point is allocated.
`centers`	A matrix of cluster centres (centroids).
`withinss`	The within-cluster sum of squares for each cluster.
`tot.withinss`	The within-cluster sum of squares summed across clusters.
`betweenss`	The between-cluster sum of squared distances.
`size`	The number of points in each cluster.

Author(s)

Dan Putler

See Also

`kmeans`

Examples

```
data(USArrests)
KMeans(USArrests, centers=3, iter.max=5, num.seeds=5)
```

Rcmdr-package *R Commander*

Description

A platform-independent basic-statistics GUI (graphical user interface) for R, based on the tcltk package.

Details

Package:	Rcmdr
Version:	1.2-6
Date:	2006/12/20
Depends:	R (>= 2.1.0), tcltk, grDevices, utils
Suggests:	abind, car (>= 1.1-1), effects (>= 1.0-7), foreign, grid, lattice, lmtest, MASS, mgcv, multcomp, nlme, nnet, relimp
LazyLoad:	no
License:	GPL version 2 or newer
URL:	http://www.r-project.org, http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/

Index:

`stem.leaf` Stem-and-Leaf Display

Translations

The R Commander comes with translations from English into several other languages. I am grateful to the following individuals for preparing these translations: Catalan, Manel Salamero; Spanish, Carlos Enrique Carleos Artime; French, Philippe Grosjean; Italian, Stefano Calza; Japanese, Takaharu Araki; Brazilian Portuguese, Adriano Azevedo Filho; Romanian, Adrian Dusa; Russian, Alexey Shipunov; Slovenian, Jaro Lajovic.

Author(s)

John Fox <jfox@mcmaster.ca>, with contributions from Michael Ash, Theophilius Boye, Stefano Calza, Andy Chang, Philippe Grosjean, Richard Heiberger, G. Jay Kerns, Renaud Lancelot, Matthieu Lesnoff, Martin Maechler, Dan Putler, Miroslav Ristic, and Peter Wolf.

Maintainer: John Fox <jfox@mcmaster.ca>

`Rcmdr.Utilities` *Rcmdr Utility Functions*

Description

These functions support writing additions to the Rcmdr package. Additional R code can be placed in files with file type `.R` in the `etc` subdirectory of the package. Add menus, submenus, and menu items by editing the file `Rcmdr-menus.txt` in the same directory.

Usage

```
activateMenus()
activeDataSet(dsname, flushModel=TRUE)
ActiveDataSet(name)
activeDataSetP()
activeModel(model)
ActiveModel(name)
activeModelP()
checkActiveDataSet()
checkActiveModel()
checkBoxes(window=top, frame, boxes, initialValues=NULL, labels)  # macro
checkClass(object, class, message=NULL)  # macro
checkFactors(n=1)
checkMethod(generic, object, message=NULL, default=FALSE, strict=FALSE,
    reportError=TRUE)  # macro
checkNumeric(n=1)
checkReplace(name, type=gettextRcmdr("Variable"))
checkTwoLevelFactors(n=1)
checkVariables(n=1)
closeDialog(window, release=TRUE)  # macro
```

```
CommanderWindow()
dataSetsP()
dialogSuffix(window=top, onOK=onOK, rows=1, columns=1, focus=top, bindReturn=TRUE,
    preventGrabFocus=FALSE, preventDoubleClick=FALSE, preventCrisp=FALSE)  # macro
doItAndPrint(command, log=TRUE)
errorCondition(window=top, recall=NULL, message, model=FALSE)  # macro
exists.method(generic, object, default=TRUE, strict=FALSE)
Factors(names)
factorsP(n=1)
## S3 method for class 'listbox':
getFrame(object)
## S3 method for class 'listbox':
getSelection(object)
getRcmdr(x, mode="any")
gettextRcmdr(...)
glmP()
GrabFocus(value)
groupsBox(recall=NULL, label=gettextRcmdr("Plot by:"),
    initialLabel=gettextRcmdr("Plot by groups"),
    plotLinesByGroup=FALSE, positionLegend=FALSE,
    plotLinesByGroupsText=gettextRcmdr("Plot lines by group"))  # macro
groupsLabel(frame=top, groupsBox=groupsBox, columnspan=1)  # macro
hclustSolutionsP()
initializeDialog(window=top, title="", offset=10, preventCrisp=FALSE)  # macro
is.valid.name(x)
justDoIt(command)
listAllModels(envir=.GlobalEnv, ...)
listDataSets(envir=.GlobalEnv, ...)
listFactors(dataSet=ActiveDataSet())
listGeneralizedLinearModels(envir=.GlobalEnv, ...)
listLinearModels(envir=.GlobalEnv, ...)
listMultinomialLogitModels(envir=.GlobalEnv, ...)
listNumeric(dataSet=ActiveDataSet())
listProportionalOddsModels(envir=.GlobalEnv, ...)
listTwoLevelFactors(dataSet=ActiveDataSet())
listVariables(dataSet=ActiveDataSet())
lmP()
logger(command)
LogWindow()
Message(message, type=c("note", "error", "warning"))
MessagesWindow()
modelFormula(frame=top, hasLhs=TRUE)  # macro
modelsP(n=1)
Numeric(names)
numericP(n=1)
OKCancelHelp(window=top, helpSubject=NULL, model=FALSE)  # macro
OutputWindow()
packageAvailable(name)
```

```
putRcmdr(x, value)
radioButtons(window=top, name, buttons, values=NULL, initialValue=..values[1],
    labels, title)  # macro
RcmdrTclSet(name, value)
RcmdrTkmessageBox(message, icon=c("info", "question", "warning",
    "error"), type=c("okcancel", "yesno", "ok"), default, title="")
subOKCancelHelp(window=subdialog, helpSubject=NULL)  # macro
subsetBox(window=top, model=FALSE)  # macro
trim.blanks(text)
TwoLevelFactors(names)
twoLevelFactorsP(n=1)
UpdateModelNumber(increment=1)
variableListBox(parentWindow, variableList=Variables(), bg="white",
    selectmode="single", export="FALSE", initialSelection=NULL,
    listHeight = 4, title)
Variables(names)

# the following function is exported for technical reasons,
# but is not meant to be called directly

commanderPosition()
```

Arguments

bg	background color.
bindReturn	if TRUE, the *Return* key is bound to the onOK function in the dialog.
boxes	vector of quoted names for check boxes, used to generate each box and its associated variable.
buttons	vector of quoted names for buttons in a set of related radio buttons.
class	quoted name of class.
columnspan	number of dialog-box columns to be spanned by frame.
command	a character string that evaluates to an R command.
dataSet, dsname	the quoted name of a data frame in memory.
default	default button: if not specified, "ok" for "okcancel", "yes" for "yesno", and "ok" for "ok"; or look for a default method.
envir	the environment to be searched; should generally be left at the default.
export	export selection?
flushModel	set (or reset) the active model to NULL? Should normally be TRUE when the active data set is changed; an exception is when variables are simply added to, deleted from, or modified in the data set set.
focus	Tk window to get the focus.
frame	frame or quoted name for frame depending upon the function.
generic	quoted name of generic function.

groupsBox	listbox object for selecting groups variable.
hasLhs	does the model formula have a left-hand side?
helpSubject	the quoted name of a help subject, to be called as help(helpSubject) when the dialog *Help* button is pressed.
icon	Message-box icon.
increment	increment to model number; -1 to set back after error.
initialLabel	label for groups button before a selection is made.
initialSelection	index of item initially selected, 0-base indexing.
initialValue	for a set of related radio buttons.
initialValues	for a set of related check boxes.
label	label prefix for groups button after a selection is made.
labels	a vector of character strings to label a set of radio buttons or check boxes.
listHeight	Maximum number of elements displayed simultaneously in list box.
log	echo command to the log window, as well as executing it and printing its output.
message	error (or other) message.
mode	mode of object to retrieve.
model	the name of a model, as a character string, or TRUE or FALSE, depending upon the function.
name	quoted name.
names	optional names to be stored.
n	number of items to check for.
object	an object (depends on context).
offset	in pixels, from top-left of Commander window.
onOK	function to execute when the *OK* button is pressed.
plotLinesByGroup	include a check box for plotting lines by group?
plotLinesByGroupsText	the label for the plot-lines-by-group check box.
positionLegend	include a check box for a legend?
preventGrabFocus	prevent the dialog box from grabbing the focus.
preventDoubleClick	prevent double-clicking from pressing the OK button, even when the double.click option is set; necessary for statistical modelling dialogs, which use double-clicking to build the model formula.
preventCrisp	prevent call to tclServiceMode, which (rarely) causes problems with some dialogs.
recall	function to call after error — usually the function that initiates the dialog.

release	release the focus if the grab.focus option has been set.
reportError	if TRUE, report an error message.
rows, columns	numbers of rows and columns of widgets in the dialog box.
values	vector of quoted values associated with radio buttons or check boxes.
selectmode	"single" or "multiple".
strict	if TRUE, only use first element of class vector.
text	a text string.
title	Window or dialog-box-element title.
type	quoted type of object to check; used to generate check-replace dialog box; or type of message to print in Message window.
value	an object to be stored.
variableList	a vector of variable names.
window, parentWindow	a Tk window.
x	an R object name, as a character string.
...	For gettextRcmdr, text string or vector of text strings to translate; otherwise disregard.

Details

There are several groups of functions exported by the Rcmdr package and documented briefly here. To see how these functions work, it is simplest to examine the dialog-generating functions in the Rcmdr package.

Executing and logging commands: The functions doItAndPrint, justDoIt, and logger control the execution, logging, and printing of commands generated by menus and dialogs. logger(command) adds command to the log/script window and to the output window. justDoIt(command) causes command to be executed. doItAndPrint(command) does both of these operations, and also prints the output produced by the command.

Checking for errors: The function is.valid.name checks whether a character string specifies a valid name for an R object. The functions checkActiveDataSet, checkActiveModel, checkFactors, checkNumeric, checkTwoLevelFactors, and checkVariables check for the existence of objects and write an error message to the log if they are absent (or insufficiently numerous, in the case of different kinds of variables). The function checkReplace opens a dialog to query whether an existing object should be replaced. The function checkMethod, checks whether a method exists for a particular generic that is appropriate for a particular object. The function checkClass checks whether an object is of a specific class. Both of these functions write error messages to the log if the condition fails. The function errorCondition reports an error to the user and (optionally) re-starts a dialog.

Information: Several functions return vectors of object names: listAllModels, listDataSets, listGeneralizedLinearModels, listFactors, listLinearModels, listMultinomialLogitModels, listNumeric, listProportionalOddsModels, listTwoLevelFactors, listVariables. The functions activeDataSet and activeModel respectively report or set the active data set and model. The function packageAvailable reports whether the named package is available to be loaded (or has possibly already been loaded).

The function `exists.method` checks whether a method exists for a particular generic that is appropriate for a particular object, and returns `TRUE` or `FALSE`.

Building dialog boxes: Several functions simplify the process of constructing Tk dialogs: initializing a dialog box, `initializeDialog`, and completing the definition of a dialog box, `dialogSuffix`; a set of check boxes, `checkBoxes`; a set of radio buttons, `radioButtons`; a list box with associated scrollbars and state variable, `variableListBox` (and the associated functions `getFrame` and `getSelection`); a button and subdialog for selecting a "grouping" variable, `groupsBox`; displaying the currently defined groups in a dialog, `groupsLabel`; a dialog-box structure for entering a model formula, `modelFormula`; a text box for entering a subsetting expression, `subsetBox`; *OK*, *Cancel*, and *Help* buttons for dialogs, `OKCancelHelp`, and subdialogs, `subOKCancelHelp`.

Translating text: The `gettextRcmdr` function simply passes its argument(s) to `gettext`, adding the argument `domain="R-Rcmdr"`.

Miscelaneous: The function `trim.blanks` removes spaces from the beginning and end of a character string.

Some of these functions, marked `# macro` under *Usage*, are "macro-like" in their behaviour, in that they execute in the environment from which they are called. These were defined with an adaptation (used with permission) of Thomas Lumley's `defmacro` function, described in Lumley (2001).

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

References

T. Lumley (2001) Programmer's niche: Macros in R. *R News*, **1(3)**, 11–13.

`Rcmdr.sciviews-specific`

Rcmdr SciViews-specific Functions

Description

These functions provide compatibility with SciViews (`http://www.sciviews.org`). Thanks to them, Rcmdr is totally integrated into SciViews Insider. In this environment, the main 'R Commander' window is replaced by an 'R Commander menu' and log files are replaced by special R code editing windows with syntax highlighting. Most of these functions are not intended for direct use.

Usage

```
is.SciViews()
is.SciViews.TclTk()
svlogger(command)
optionLogCommand()
```

```
optionAttachDataSet()
optionSortVariables()
refreshStatus()
```

Arguments

command	a character string that evaluates to an R command.

Details

The functions `is.SciViews` tests if R is running under SciViews. If not, most of the other SciViews-specific functions do nothing. `is.SciViews.TclTk` test if the SciViews client communicates with R through Tcl/Tk (otherwise, it probably uses SciViews plugs). The function `svlogger` is similar to `logger`, but it records Rcmdr commands in the specific SciViews R script window and in the SciViews command history, instead of the log window and the default R command history. `optionLogCommand`, `optionAttachDataSet` and optionSortVariables allow to change the command logging, automatic attachment of the active data set and sorting of variable names (equivalent options than those accessible by check boxes in the 'R Commander' window of Rcmdr outside of SciViews, or in the Options dialog box). In SciViews insider, the state of these options, as well as the names of the active data set and model are displayed in the status bar. `refreshStatus` make sure that this information in the status bar is updated according to the current internal state of Rcmdr.

Author(s)

Philippe Grosjean ⟨phgrosjean@sciviews.org⟩

RcmdrPager *Pager for Text Files*

Description

This is a slightly modified version of the tkpager, changed to use the Rcmdr monospaced font and a white background.

Usage

```
RcmdrPager(file, header, title, delete.file)
```

Arguments

file	character vector of file(s) to be displayed.
header	for the beginning of each file.
title	for window
delete.file	delete file(s) on close.

See Also

tkpager

RcmdrTestDrive *Test-driving the R Commander*

Description

These are simulated data specifically designed to allow the inexperienced user to browse the capabilities of the R Commander.

Usage

```
data(RcmdrTestDrive)
```

Format

A data frame with 168 observations on the following 9 variables:

Order sequential order

Smoking smoking status

Gender gender of victim

Race race of victim

Before life expectancy before exposure

After life expectancy after exposure

Salary salary at retirement

Reduction potential salary reduction

Parking number of unpaid parking tickets

Details

The R Commander has extensive functionality, but many options are unavailable unless the correct types of data are loaded in the Active Data Set. This data set was randomly generated so that, when loaded, essentially all R Commander options would be available for the student to investigate. These data are entirely fictional. For an amusing contributed story tying these variables together, please visit http://www.cc.ysu.edu/~gjkerns/IPSUR/package.

Source

These data were randomly generated using the IPSUR probability menu for the R Commander.

`Recode` *Rcmdr Recode Dialog*

Description

The recode dialog is normally used to recode numeric variables and factors into factors, for example by combining values of numeric variables or levels of factors. It may also be used to produce new numeric variables. The Rcmdr recode dialog is based on the `recode` function in the `car` package.

Details

The name of each new variable must be a valid R object name (consisting only of upper and lower-case letters, numerals, and periods, and not starting with a numeral).

Enter recode directives in the box near the bottom of the dialog. Directives are normally entered one per line, but may also be separated by semicolons. Each directive is of the form `input = output` (see the examples below). If an input value satisfies more than one specification, then the first (from top to bottom, and left to right) applies. If no specification is satisfied, then the input value is carried over to the result. `NA` is allowed on input and output. Factor levels are enclosed in double-quotes on both input and output.

Several recode specifications are supported:

a single value For example, `"missing" = NA`.

several values separated by commas For example, `7,8,9 = "high"`.

a range of values indicated by a colon For example, `7:9 = "high"`. The special values `low` and `high` may appear in a range. For example, `low:10=1`. Note that these values are unquoted.

the special value `else` everything that does not fit a previous specification. For example, `else=NA`. Note that `else` matches *all* otherwise unspecified values on input, including `NA`.

If all of the output values are numeric, and the "Make new variable a factor" check box is unchecked, then a numeric result is returned.

If several variables are selected for recoding, then each is recoded using the same recode directives. In this case, the name entered in the box labelled "New variable name or prefix for multiple recodes" will be prefixed to the name of each variable being recoded. Setting an empty prefix (i.e., "") will cause the recoded variables to replace the original variables.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

`recode`

`Scatter3DDialog` *Rcmdr 3D Scatterplot Dialog*

Description

This dialog sets up a call to the `scatter3d` function to draw a three-dimensional scatterplot, and optionally to `identify3d` to label points interactively with the mouse.

Details

The explanatory variables provide the "horizontal" and "out-of-screen" axes of the scatterplot, the response variable provides the "vertical" axis.

Data points are represented as spheres or points, depending upon the number of observations.

Several regression surfaces can be plotted: a linear least-squares surface; a full quadratic least-squares surface with squared and cross-product terms; a "smooth" regression surface — either a smoothing spline, if no degrees of freedom are specified (in which case the `gam` function selects the df by generalized cross validation), or a fixed-df regression spline; an additive-regression surface (also fit by `gam`), with either smoothing spline or regression spline components (again selected according to the specification of degrees of freedom). If only one surface is fit, then residuals are plotted as red (negative) and green (positive) lines from the surface to the points.

You can specify a factor defining groups by pressing the *Plot by groups* button. A separate surface or set of surfaces is plotted for each level of the groups factor. These surfaces can be constrained to be parallel.

The completed plot can be manipulated with the mouse: Click, hold, drag the left mouse button to rotate the display; click, hold, and drag the right button (or centre button on a three-button mouse) to zoom in and out.

If the box labelled *Identify observations with mouse* is checked, you may use the mouse to identify points interactively: Press the right mouse button (or the centre button on a three-button mouse), drag a rectangle around the points to be identified, and release the button. Repeat this procedure for each point or set of "nearby" points to be identified. To exit from point-identification mode, right-click (or centre-click) in an empty region of the plot.

Points may also be identified subsequently by selecting *Identify observations with mouse* from the R Commander *3D graph* menu: As above, click and drag the left mouse button to rotate the display, and click and drag the right (or centre) button to identify points.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

`scatter3d`, `identify3d`, `rgl.open`, `gam`

`assignCluster` *Append a Cluster Membership Variable to a Dataframe*

Description

Correctly creates a cluster membership variable that can be attached to a dataframe when only a subset of the observations in that dataframe were used to create the clustering solution. NAs are assigned to the observations of the original dataframe not used in creating the clustering solution.

Usage

```
assignCluster(clusterData, origData, clusterVec)
```

Arguments

`clusterData`	The data matrix used in the clustering solution. The data matrix may have have only a subset of the observations contained in the original dataframe.
`origData`	The original dataframe from which the data used in the clustering solution were taken.
`clusterVec`	An integer variable containing the cluster membership assignments for the observations used in creating the clustering solution. This vector can be created using `cutree` for clustering solutions generated by `hclust` or the `cluster` component of a list object created by `kmeans` or `KMeans`.

Value

A factor (with integer labels) that indicate the cluster assignment for each observation, with an NA value given to observations not used in the clustering solution.

Author(s)

Dan Putler

See Also

`hclust`, `cutree`, `kmeans`, `KMeans`

Examples

```
data(USArrests)
USArrkm3 <- KMeans(USArrests[USArrests$UrbanPop<66, ], centers=3)
assignCluster(USArrests[USArrests$UrbanPop<66, ], USArrests, USArrkm3$cluster)
```

bin.var *Bin a Numeric Varisible*

Description

Create a factor dissecting the range of a numeric variable into bins of equal width, (roughly) equal frequency, or at "natural" cut points. The cut function is used to create the factor.

Usage

```
bin.var(x, bins = 4, method = c("intervals", "proportions", "natural"),
    labels = FALSE)
```

Arguments

x	numeric variable to be binned.
bins	number of bins.
method	one of "intervals" for equal-width bins; "proportions" for equal-count bins; "natural" for cut points between bins to be determined by a k-means clustering.
labels	if FALSE, numeric labels will be used for the factor levels; if NULL, the cut points are used to define labels; otherwise a character vector of level names.

Value

A factor.

Author(s)

Dan Putler, slightly modified by John Fox ⟨jfox@mcmaster.ca⟩ with the original author's permission.

See Also

cut, kmeans.

Examples

```
summary(bin.var(rnorm(100), method="prop", labels=letters[1:4]))
```

`birthday.ipsur` *Probability of coincidences for the IPSUR package*

Description

This is a modified version of the `pbirthday` and `qbirthday` functions in the `stats` package. Computes approximate answers to a generalised "birthday paradox" problem. `pbirthday.ipsur` computes the probability of a coincidence and `qbirthday.ipsur` computes the number of observations needed to have a specified probability of coincidence. The change is that precise answers are given (instead of asymptotics) in the case of exactly two coincidences.

Usage

```
qbirthday.ipsur(prob = 0.5, classes = 365, coincident = 2)
pbirthday.ipsur(n, classes = 365, coincident = 2)
```

Arguments

`classes`	How many distinct categories the people could fall into
`prob`	The desired probability of coincidence
`n`	The number of people
`coincident`	The number of people to fall in the same category

Details

The birthday paradox is that a very small number of people, 23, suffices to have a 50-50 chance that two of them have the same birthday. This function generalises the calculation to probabilities other than 0.5, numbers of coincident events other than 2, and numbers of classes other than 365.

The formula is approximate, except in the case `coincident=2`.

Value

`qbirthday.ipsur`
: Number of people needed for a probability `prob` that `k` of them have the same one out of `classes` equiprobable labels.

`pbirthday.ipsur`
: Probability of the specified coincidence.

References

Diaconis P, Mosteller F., "Methods for studying coincidences". JASA 84:853-861

Examples

```
 ## the standard version
qbirthday.ipsur()
 ## same 4-digit PIN number
qbirthday.ipsur(classes=10^4)
 ## 0.9 probability of three coincident birthdays
qbirthday.ipsur(coincident=3, prob=0.9)
## Chance of 4 coincident birthdays in 150 people
pbirthday.ipsur(150,coincident=4)
## 100 coincident birthdays in 1000 people: *very* rare:
pbirthday.ipsur(1000, coincident=100)
```

`colPercents` *Row, Column, and Total Percentage Tables*

Description

Percentage a matrix or higher-dimensional array of frequency counts by rows, columns, or total frequency.

Usage

```
colPercents(tab, digits=1)
rowPercents(tab, digits=1)
totPercents(tab, digits=1)
```

Arguments

`tab`	a matrix or higher-dimensional array of frequency counts.
`digits`	number of places to the right of the decimal place for percentages.

Value

Returns an array of the same size and shape as `tab` percentaged by rows or columns, plus rows or columns of totals and counts, or by the table total.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

`generalizedLinearModel` *Rcmdr Generalized Linear Model Dialog*

Description

This dialog is used to specify a generalized linear model to be fit by the `glm` function.

Details

The left model-formula box specifies the response variable to be used in the model; it may be a variable name or an expression evaluating to the response variable, such as `working == "Fulltime"`.

The right model-formula box specifies the right-hand (i.e., predictor) side of the model. See `glm` for details.

You can type directly in the model formula boxes. Alternatively, double-clicking the left mouse button on a variable in the variable-list transfers it to the left-hand side of the model (if it is empty) or to the right-hand side. Factors are indicated in the variable list; all other variables are numeric. You can also enter operators and parentheses using the buttons above the formula.

Double-click the left mouse button to select a family in the "Family" box and the corresponding permissible link functions appear in the "Link function" box to the right. Initially, the canonical link for the family is selected. See `family` for details.

Specifying a subset expression allows you to fit the model to a subset of observations in the active data set. For example, assuming that `gender` is a variable in the active data set, entering `gender == "Male"` would restrict the model to males.

If the active model is a generalized linear model, and the active data set has not changed, then the initial values of the left-hand-side, right-hand-side, family, link, and subset fields are retained from the active model.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

`glm`, `family`, `Comparison`

hierarchicalCluster *Rcmdr Hierarchical Clustering Dialog*

Description

This dialog is used to specify a hierarchical cluster analysis solution using `hclust`, with the distance matrix calculated using `dist`.

Details

Enter a name for the hierarchical clustering solution to be created if you want to retain more than one solution. The solution name must be a valid R object name (consisting only of upper- and lower-case letters, numerials, and periods, and not starting with a number).

Select the variables to be included in the solution using the variable selection box on the left side of the dialog box. A non-contiguous set of variables can be selected by pressing your control key (ctrl) while selecting variables.

Specifying a subset expression (the field below the variable selection box) allows you to obtain a clustering solution for a subset of observations in the active data set. For example, assuming that gener is a variable in the active data set, entering `gender == "Male"` would restrict the solution to males.

Select a clustering method and a distance measure if you are working with raw data. There is often a relationship between the selection of these two items. For example, squared-euclidian distance is appropriate for Ward's method of cluster analysis. If your data *is* a distance matrix, then select "No Transformation" as the distance measure.

The "Plot Dendrogram" option results in the dendrogram of the solution being display by using the `plot` function.

Author(s)

Dan Putler

See Also

`hclust`, `dist`

linearModel *Rcmdr Linear Model Dialog*

Description

This dialog is used to specify a linear model to be fit by the `lm` function.

Details

The left model-formula box specifies the response variable to be used in the model; it may be a variable name or an expression evaluating to the response variable, such as `log(income)`.

The right model-formula box specifies the right-hand (i.e., predictor) side of the model. See `lm` for details.

You can type directly in the model formula boxes. Alternatively, double-clicking the left mouse button on a variable in the variable-list transfers it to the left-hand side of the model (if it is empty) or to the right-hand side. You can also enter operators and parentheses using the buttons above the formula.

Specifying a subset expression allows you to fit the model to a subset of observations in the active data set. For example, assuming that `gender` is a variable in the active data set, entering `gender == "Male"` would restrict the model to males.

If the active model is a linear model and the active data set has not changed, then the initial values of the left-hand-side, right-hand-side, and subset fields are retained from the previous model.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

`lm`, `Comparison`

`numSummary` *Mean, Standard Deviation, and Quantiles for Numeric Variables*

Description

`numSummary` creates neatly formatted tables of means, standard deviations, and quantiles of numeric variables.

Usage

```
numSummary(data, statistics=c("mean", "sd", "quantiles"),
    quantiles=c(0, .25, .5, .75, 1), groups)

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

Arguments

`data`	a numeric vector, matrix, or data frame.
`statistics`	any of `"mean"`, `"sd"`, or `"quantiles"`, defaulting to all three.
`quantiles`	quantiles to report; default is `c(0, 0.25, 0.5, 0.75, 1)`.
`groups`	optional variable, typically a factor, to be used to partition the data.

x	object of class "numSummary" to print.
...	arguments to pass down from the print method.

Value

numSummary returns an object of class "numSummary" containing the table of statistics to be reported along with information on missing data, if there are any.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

mean, sd, quantile.

Examples

```
library(car)
Prestige[1, "income"] <- NA
numSummary(Prestige[,c("income", "education")])
numSummary(Prestige[,c("income", "education")], groups=Prestige$type)
remove(Prestige)
```

---

partial.cor *Partial Correlations*

Description

Computes a matrix of partial correlations between each pair of variables controlling for the others.

Usage

```
partial.cor(X, ...)
```

Arguments

X	data matrix.
...	arguments to be passed to cor.

Value

Returns a matrix of partial correlations.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

cor

Examples

```
data(DavisThin)
partial.cor(DavisThin)
```

---

plotMeans *Plot Means for One or Two-Way Layout*

Description

Plots cell means for a numeric variable in each category of a factor or in each combination of categories of two factors, optionally along with error bars based on cell standard errors or standard deviations.

Usage

```
plotMeans(response, factor1, factor2,
    error.bars = c("se", "sd", "conf.int", "none"), level=0.95,
    xlab = deparse(substitute(factor1)),
    ylab = paste("mean of", deparse(substitute(response))),
    legend.lab = deparse(substitute(factor2)), main = "Plot of Means",
    pch = 1:n.levs.2, lty = 1:n.levs.2, col = palette())
```

Arguments

response	Numeric variable for which means are to be computed.
factor1	Factor defining horizontal axis of the plot.
factor2	If present, factor defining profiles of means
error.bars	If "se", the default, error bars around means give plus or minus one standard error of the mean; if "sd", error bars give plus or minus one standard deviation; if "conf.int", error bars give a confidence interval around each mean; if "none", error bars are suppressed.
level	level of confidence for confidence intervals; default is .95
xlab	Label for horizontal axis.
ylab	Label for vertical axis.
legend.lab	Label for legend.
main	Label for the graph.
pch	Plotting characters for profiles of means.
lty	Line types for profiles of means.
col	Colours for profiles of means

Value

The function invisibly returns NULL.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

interaction.plot

---

reliability *Reliability of a Composite Scale*

Description

Calculates Cronbach's alpha and standardized alpha (lower bounds on reliability) for a composite (summated-rating) scale. Standardized alpha is for the sum of the standardized items. In addition, the function calculates alpha and standardized alpha for the scale with each item deleted in turn, and computes the correlation between each item and the sum of the other items.

Usage

```
reliability(S)

## S3 method for class 'reliability':
print(x, digits=4, ...)
```

Arguments

S	the covariance matrix of the items; normally, there should be at least 3 items and certainly no fewer than 2.
x	reliability object to be printed.
digits	number of decimal places.
...	not used: for compatibility with the print generic."

Value

an object of class reliability, which normally would be printed.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

References

N. Cliff (1986) Psychological testing theory. Pp. 343–349 in S. Kotz and N. Johnson, eds., *Encyclopedia of Statistical Sciences, Vol. 7*. Wiley.

See Also

cov

Examples

```
data(DavisThin)
reliability(cov(DavisThin))
```

---

scatter3d *Three-Dimensional Scatterplots and Point Identification*

Description

The `scatter3d` function uses the `rgl` package to draw 3D scatterplots with various regression surfaces. The function `identify3d` allows you to label points interactively with the mouse: Press the right mouse button (on a two-button mouse) or the centre button (on a three-button mouse), drag a rectangle around the points to be identified, and release the button. Repeat this procedure for each point or set of "nearby" points to be identified. To exit from point-identification mode, click the right (or centre) button an empty region of the plot.

Usage

```
scatter3d(x, y, z,
        xlab=deparse(substitute(x)), ylab=deparse(substitute(y)),
        axis.scales=TRUE,
        zlab=deparse(substitute(z)), revolutions=0, bg.col=c("white", "black"),
        axis.col=if (bg.col == "white") c("darkmagenta", "black", "darkcyan")
            else c("darkmagenta", "white", "darkcyan"),
        surface.col=c("blue", "green", "orange", "magenta", "cyan", "red", "yellow"
        neg.res.col="red", pos.res.col="green",
        square.col=if (bg.col == "white") "black" else "gray", point.col="yellow",
        text.col=axis.col, grid.col=if (bg.col == "white") "black" else "gray",
        fogtype=c("exp2", "linear", "exp", "none"),
        residuals=(length(fit) == 1), surface=TRUE, fill=TRUE, grid=TRUE, grid.line
        df.smooth=NULL, df.additive=NULL,
        sphere.size=1, threshold=0.01, speed=1, fov=60,
        fit="linear", groups=NULL, parallel=TRUE, ellipsoid=FALSE, level=0.5,
        model.summary=FALSE)

identify3d(x, y, z, axis.scales=TRUE, groups=NULL, labels=1:length(x),
    col=c("blue", "green", "orange", "magenta", "cyan", "red", "yellow", "gray"),
    offset = ((100/length(x))^(1/3)) * 0.02)
```

Arguments

x	variable for horizontal axis.
y	variable for vertical axis (response).
z	variable for out-of-screen axis.
xlab, ylab, zlab	axis labels.
axis.scales	if TRUE, label the values of the ends of the axes. *Note:* For identify3d to work properly, the value of this argument must be the same as in scatter3d.
revolutions	number of full revolutions of the display.
bg.col	background colour; one of "white", "black".
axis.col	colours for axes; if axis.scales is FALSE, then the second colour is used for all three axes.
surface.col	vector of colours for regression planes, used in the order specified by fit.
neg.res.col, pos.res.col	colours for lines representing negative and positive residuals.
square.col	colour to use to plot squared residuals.
point.col	colour of points.
text.col	colour of axis labels.
grid.col	colour of grid lines on the regression surface(s).
fogtype	type of fog effect; one of "exp2", "linear", "exp", "none".
residuals	plot residuals if TRUE; if residuals="squares", then the squared residuals are shown as squares (using code adapted from Richard Heiberger). Residuals are available only when there is one surface plotted.
surface	plot surface(s) (TRUE or FALSE).
fill	fill the plotted surface(s) with colour (TRUE or FALSE).
grid	plot grid lines on the regression surface(s) (TRUE or FALSE).
grid.lines	number of lines (default, 26) forming the grid, in each of the x and y directions.
df.smooth	degrees of freedom for the two-dimensional smooth regression surface; if NULL (the default), the gam function will select the degrees of freedom for a smoothing spline by generalized cross-validation; if a positive number, a fixed regression spline will be fit with the specified degrees of freedom.
df.additive	degrees of freedom for each explanatory variable in an additive regression; if NULL (the default), the gam function will select degrees of freedom for the smoothing splines by generalized cross-validation; if a positive number or a vector of two positive numbers, fixed regression splines will be fit with the specified degrees of freedom for each term.
sphere.size	relative sizes of spheres representing points; the actual size is dependent on the number of observations.
threshold	if the actual size of the spheres is less than the threshold, points are plotted instead.
speed	relative speed of revolution of the plot.

fov	field of view (in degrees); controls degree of perspective.
fit	one or more of "linear", "quadratic", "smooth", "additive"; to display fitted surface(s); partial matching is supported – e.g., c("lin", "quad").
groups	if NULL (the default), no groups are defined; if a factor, a different surface or set of surfaces is plotted for each level of the factor; in this event, the colours in plane.col are used successively for the points, surfaces, and residuals corresponding to each level of the factor.
parallel	when plotting surfaces by groups, should the surfaces be constrained to be parallel? A logical value, with default TRUE.
ellipsoid	plot concentration ellipsoid(s) (TRUE or FALSE).
level	expected proportion of bivariate-normal observations included in the concentration ellipsoid(s); default is 0.5.
model.summary	print summary or summaries of the model(s) fit (TRUE or FALSE). scatter3d rescales the three variables internally to fit in the unit cube; this rescaling will affect regression coefficients.
labels	text labels for the points, one for each point; defaults to the observation indices.
col	colours for the point labels, given by group. There must be at least as many colours as groups; if there are no groups, the first colour is used. Normally, the colours would correspond to the plane.col argument to scatter3d.
offset	vertical displacement for point labels (to avoid overplotting by points).

Value

scatter3d not return a useful value; it is used for its side-effect of creating a 3D scatterplot. indentify3d returns the labels of the identified points.

Note

You have to install the rgl and mgcv packages to produce 3D plots.

Author(s)

John Fox ⟨jfox@mcmaster.ca⟩

See Also

rgl.open, gam

Examples

```
    ## Not run:
State.x77 <- as.data.frame(state.x77)
with(State.x77, scatter3d(Income, Murder, Illiteracy))
with(State.x77, identify3d(Income, Murder, Illiteracy, labels=row.names(State.x77)))
with(State.x77,  scatter3d(Income, Murder, Illiteracy, fit=c("linear", "quadratic")))

## End(Not run)
```

`stem.leaf` *Stem-and-Leaf Display*

Description

Creates a classical ("Tukey-style") stem-and-leaf display.

Usage

```
stem.leaf(data, unit, m, Min, Max,
    rule.line = c("Dixon", "Velleman", "Sturges"),
    style = c("Tukey", "bare"), trim.outliers = TRUE, depths = TRUE,
    reverse.negative.leaves = TRUE)

## S3 method for class 'stem.leaf':
print(x, ...)
```

Arguments

`data`	a numeric vector.
`unit`	leaf unit, as a power of 10 (e.g., `100`, `.01`); omit to let the function choose the unit.
`m`	number of parts (1, 2, or 5) into which each stem should be divided; omit to let the function choose the number of parts/stem.
`Min`	smallest non-outlying value; omit for automatic choice.
`Max`	largest non-outlying value; omit for automatic choice.
`rule.line`	the rule to use for choosing the desired number of lines in the display; `"Dixon"` = 10*log10(n); `"Velleman"` = 2*sqrt(n); `"Sturges"` = 1 + log2(n); the default is `"Dixon"`.
`style`	`"Tukey"` (the default) for "Tukey-style" divided stems; `"bare"` for divided stems that simply repeat the stem digits.
`trim.outliers`	if `TRUE` (the default), outliers are placed on `LO` and `HI` stems.
`depths`	if `TRUE` (the default), print a column of "depths" to the left of the stems; the depth of the stem containing the median is the stem-count enclosed in parentheses.
`reverse.negative.leaves`	if `TRUE` (the default), reverse the leaves on negative stems (so, e.g., the leaf 9 comes before the leaf 8, etc.).
`x`	an object of class stem.leaf to be printed.
`...`	not used: for compatibility with the generic print function.

Details

Unlike the `stem` function in the `base` package, this function produces classic stem-and-leaf displays, as described in Tukey's *Exploratory Data Analysis*. Outliers are determined using the rule for boxplots (see `boxplot.stats`).

Value

Returns on object of class stem.leaf, which normally would be printed.

Author(s)

Peter Wolf, slightly modified by John Fox ⟨jfox@mcmaster.ca⟩ with the original author's permission.

References

Tukey, J. *Exploratory Data Analysis.* Addison-Wesley, 1977.

See Also

`stem`

Examples

```
data(Prestige)
stem.leaf(Prestige$income)
```

Index