Package 'Rcmdr'

July 27, 2010

Version 1.6-0

Date 2010/07/12

Title R Commander

Author John Fox <jfox@mcmaster.ca>, with contributions from Liviu Andronic, Michael Ash, Theophilius Boye, Stefano Calza, Andy Chang, Philippe Grosjean, Richard Heiberger, G. Jay Kerns, Renaud Lancelot, Matthieu Lesnoff, Uwe Ligges, Samir Messad, Martin Maechler, Robert Muenchen, Duncan Murdoch, Erich Neuwirth, Dan Putler, Brian Ripley, Miroslav Ristic, and Peter Wolf.

Maintainer John Fox <jfox@mcmaster.ca>

Depends R (>= 2.6.0), tcltk, grDevices, utils, car (>= 2.0-0),

Suggests abind, aplpack, colorspace, effects (>= 1.0-7), foreign, grid, Hmisc, lattice, leaps, lmtest, MASS, mgcv, multcomp (>= 0.991-2), nlme, nnet, relimp, rgl, RODBC

LazyLoad no

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

License GPL (>= 2)

URL http://www.r-project.org,
http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/

Repository CRAN

Repository/R-Forge/Project rcmdr

Repository/R-Forge/Revision 96

Date/Publication 2010-07-27 11:32:46

R topics documented:

`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:	1.6-0
Date:	2010/07/12
Depends:	R (>= 2.1.0), tcltk, grDevices, utils
Suggests:	abind, aplpack, car (>= 1.1-1), effects (>= 1.0-7), foreign, grid, lattice, lmtest, MASS, mgcv, multcomp, nlme, nn
LazyLoad:	no
License:	GPL (>= 2)

URL: http://www.r-project.org, http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/

Translations

The R Commander comes with translations from English into several other languages. I am grateful to the following individuals and groups for preparing these translations: Brazilian Portuguese, Adriano Azevedo-Filho and Marilia Sa Carvalho; Catalan, Manel Salamero; French, Philippe Grosjean; German: Gerhard Schoen; Indonesian, I Made Tirta; Italian, Stefano Calza; Japanese, Takaharu Araki; Korean, Dae-Heung Jang; Polish, Marcin Kozak; Romanian, Adrian Dusa; Russian, Alexey Shipunov; Slovenian, Jaro Lajovic; Spanish, Spanish R-UCA Project, http://knuth.uca.es/R.

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, Uwe Ligges, Samir Messad, Martin Maechler, Robert Muenchen, Duncan Murdoch, Erich Neuwirth, Dan Putler, Brian Ripley, Miroslav Ristic, and Peter Wolf.

Maintainer: John Fox <jfox@mcmaster.ca>

---

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]))
```

---

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>

`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, STATA data set, or an Excel, Access, or dBase 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), and commands and output appear in the output window (the initially empty text window below the script window). 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, the several lines of a multi-line command must be submitted simultaneously. (It is not necessary, as in earlier versions of the R Commander, to begin continuation

lines with white space.) 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. Finally, editing operations also work in the messages window.

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", or *Control-Tab*). If no text is selected, the *Submit* button (or *Control-R* or *Control-Tab*) 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, output, or messages window. Edit functions such as search are performed in the script window unless you first click in the output or messages 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, output, or messages window.

In addition, the following Control-key combinations work in the script, output, and messages windows: *Control-X*, cut; *Control-C*, copy; *Control-V*, insert; *Control-Z* or *Alt-Backspace*, undo; and *Control-W*, redo.

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.

You can cancel an R Commander dialog box by pressing the *Esc* key.

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

Customization and Configuration

The preferred way of customizing the R Commander is to write a plug-in package: see `help("Plugins")`.

Alternatively, 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`.

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. Alternatively, you can edit the source package and recompile it.

To reiterate, however, the preferred procedure is to write an R Commander plug-in package.

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:

`ask.to.exit` if `TRUE` (the default), then the user is asked whether he or she wants to exit the `Rcmdr`; if this option is set to `FALSE`, then the subsequent option is also set to `FALSE`.

`ask.on.exit` if `TRUE` (the default), then the user is asked whether to save the script file and the output file when the `Rcmdr` exits.

`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`.

`default.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. Prior to R 2.6.1, this option only works on the Windows version of R, but should in any event be harmless. The default is `TRUE`. If you encounter stability problems, try 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 for non-Windows systems is `"*helvetica-medium-r-normal-*-xx*"`, where `xx` is the default font size.

`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.

`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`.

`iconify.commander` If `TRUE`, the *Commander* window is minimized on startup; the default is `FALSE`.

`length.output.stack` The R Commander maintains a list of output objects, by default including the last several outputs; the default length of the output stack is 10. `popOutput()` "pops" (i.e., returns and removes) the first entry of the output stack. Note that, as a stack, the queue is LIFO ("last in, first out").

`length.command.stack` The R Commander also maintains a list of commands that is managed similarly; the default length of this stack is also 10.

`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.

`messages.height` The height of the messages window, in lines. The default is 3.

`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.

`number.messages` If `TRUE`, the default, messages in the messages window are numbered.

`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.

`plugins` A character vector giving the names of `Rcmdr` plug-in packages to load when the Commander starts up. Plug-in packages can also be loaded from the *Tools -> Load Rcmdr plug-in(s)* menu. See Plugins.

`prefixes` A four-item character vector to specify the prefixes used when output is directed to the R console; the default is `c("Rcmdr> ", "Rcmdr+ ", "RcmdrMsg: ", "RcmdrMsg+ ")`.

`suppress.menus` if `TRUE`, the Commander menu bar and tool bar are suppressed, allowing another program (such as Excel) to take over these functions. The default (of course) is `FALSE`.

`suppress.X11.warnings` On (some?) Linux and Mac OS X 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) 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.

`retain.messages` If `TRUE` (the default), 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".

`RExcelSupport` If `TRUE` (the default is `FALSE`), menus and output are handled by Excel.

`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 `View()` rather than `showData()` is used to display the data set. 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.

`variable.list.height` the number of items (typically variables) to display in list boxes; longer lists may be viewed by scrolling. The default is 4.

`variable.list.width` a two-item vector controlling the width of list boxes, in characters, giving the minimum and maximum width to display; the default is `c(20, Inf)`. If the widest item name falls in this range, then its number of characters determines the width of the box. *Note:* This specification works only approximately.

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

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

If you want always to launch the R Commander when R starts up, you can include the following code in one of R's start-up files (e.g., in the `Rprofile.site` file in R's `etc` subdirectory):

```
local({
old <- getOption("defaultPackages")
options(defaultPackages = c(old, "Rcmdr"))
})
```

R Commander options can also be permanently set in the same manner. For more information about R initialization, see `?Startup`.

Warning

The R Commander Script window does not provide a true console to R, and may have certain limitations. I don't recommend using the R Commander for serious programming or for data analysis that relies primarily on scripts — use a programming editor instead. If you encounter any problems with the Script window, however, I'd appreciate it if you brought them to my attention.

Known Problems

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. I have not seen this behavior in some time and the problem may have been solved.

Note

This version may be compatible with SciViews: `http://www.sciviews.org/SciViews-R`; see `Rcmdr.sciviews-specific`. This version should be compatiable with the **RExcel** package, which can use the R Commander menus.

Under Windows, the `Rcmdr` package can be run under the *Rgui* in the SDI (single-document interface) mode, or under `rterm.exe`. You might experience problems running the `Rcmdr` under `ESS` with NTEmacs or XEmacs; under other R consoles; or under the Rgui in the MDI (multiple-document interface) mode.

Author(s)

John Fox `<jfox@mcmaster.ca>`

See Also

Plugins, Rcmdr.Utilities

Examples

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

`Commander-es` *R Commander*

Description

Inicia la GUI (Interfaz Gráfica de Usuario) de R Commander

Usage

```
Commander()
```

Details

Empezando

La interfaz por defecto de R Commander consiste en (de arriba a abajo) una barra de menús, una barra de herramientas, una ventana de instrucciones, una ventana de salida y una ventana de mensajes.

Las instrucciones para leer, escribir, transformar y analizar datos se ejecutan usando la barra de menú de la parte superior de la ventana de *R Commander*. La mayor parte de los items de este menú le guiarán mediante ventanas de diálogo, preguntando más allá de la especificación. Es aconsejable explorar el menú para ver las opciones disponibles.

Bajo la barra de menú se encuentra la barra de herramientas con (de izquierda a derecha) un campo de información que muestra el nombre del conjunto de datos activos, botones para editar y mostrar el conjunto de datos activos y un campo de información mostrando el modelo estadístico activo. Bajo la ventana de instrucciones hay un botón `Ejecutar` para realizar las órdenes indicadas en la ventana de instrucciones. Los campos de información para los datos y el modelo activo son botones que pueden ser usados para seleccionar éstos entre, respectivamente, conjuntos de datos o modelos disponibles en memoria.

La mayor parte de las órdenes requiere un conjunto de datos activos. Cuando se ejecuta R Commander no hay conjunto de datos activos, como está indicado en el campo de información del conjunto de datos activos. Un conjunto de datos llega a ser un conjunto de datos activos cuando éste es leído en la memoria desde un paquete R o importado desde un archivo de texto, conjunto de datos SPSS, conjunto de datos Minitab, conjunto de datos STATA, Excel, Access o dBase. Además el conjunto de datos activos puede ser seleccionado desde conjuntos de datos R residentes en memoria. Los datos pueden ser elegidos de entre todos los conjuntos para cada sesión.

Por defecto, las órdenes son registradas en la ventana de instrucciones (la ventana de texto vacía inmediatamente después de la barra de herramientas); las órdenes y las salidas aparecen en la ventana de resultados (la ventana de texto vacía después de la ventana de instrucciones) y el conjunto de datos activos es adjuntado a la ruta de búsqueda. Para alterar éstos y otros parámetros por defecto, puede consultar la información pertinente en configuración.

Algunos diálogos de `Rcmdr` (éstos en `Estadisticos -> Ajuste de modelos`) generan el modelo lineal, modelo lineal generalizado y otros modelos. Cuando un modelo es ajustado, se convierte en el modelo activo, indicado en el campo de información de la barra de herramientas de R Commander. Los items del menú `Modelos` se aplican al modelo activo. Inicialmente, no hay modelo activo. Si hay varios modelos en memoria, puede elegir el modelo activo de entre ellos.

Si el registro de instrucciones está activo, las órdenes de R generadas desde los menús y los cuadros de diálogos, son introducidas en la ventana de instrucciones de R Comander. Se pueden editar estas órdenes de manera normal y se pueden escribir otras nuevas en la ventana de instrucciones. Las órdenes individuales pueden ser continuadas en más de una línea, pero cada línea después de la primera debe ser identada con uno o más espacios o tabuladores. El contenido de la ventana de instrucciones puede ser almacenado durante o al final de la sesión y un conjunto de instrucciones guardado puede ser cargado en la ventana de instrucciones. El contenido de la ventana de resultados puede ser editado o guardado en un archivo de texto.

Para volver a ejecutar una orden o un conjunto de ellas, se seleccionan las líneas que se desean ejecutar usando el ratón y se presiona el botón `Ejecutar`, situado a la derecha de la barra de herramientas (o `Control-R`, para ejecutarlos). Si no hay texto seleccionado el botón `Ejecutar` (o `Control-R`) envía el contenido de la línea que contiene el cursor de insercción. Observar que se generará un error si la orden o las órdenes enviadas son incompletas.

Presionando `Control-F` se abre un cuadro de diálogo de búsqueda de texto (también es accesible vía `Editar -> Buscar`) para buscar el texto en la ventana de instrucciones o la ventana de resultados. Las búsquedas son realizadas en la ventana de instrucciones a menos que primero pulse en la ventana de resultados para activarla.

Presionando `Control-S` se guardará el conjunto de instrucciones o la ventana de resultados.

Presionando `Control-A` se selecciona todo el texto del conjunto de instrucciones o de la ventana de resultados.

Pulsando el botón derecho del ratón (el tercer botón en un ratón de tres botones) en el conjunto de instrucciones o en la ventana de resultados se abre el menú contextual con los items del menú `Editar`, más un item `Ejecutar` (en la ventana de instrucciones).

Cuando ejecute órdenes en la ventana de R Commander, debe asegurarse que la sentencia sea lógica. Por ejemplo, no tiene sentido ajustar un modelo estadístico de un conjunto de datos que no ha sido leído en memoria.

Presionando una letra (ej. "a") en un cuadro con una lista se recorrerá ésta hasta la siguiente entrada que comience con esa letra desde el principio del cuadro.

Salir de R Commander se realiza mediante `Fichero -> Salir` o cerrando la ventana de R Commander.

Personalización y configuración

Los archivos de configuración están en el subdirectorio `etc` de cada paquete o en la localización dada por `etc` y en las opciones de `etcMenus` (mirar abajo).

Los menús de `Rcmdr` pueden ser personalizados editando el archivo `Rcmdr-menus.txt`.

Algunas funciones (ej. `histograma`) que normalmente no crean salida visible cuando se ejecutan desde la consola sí lo harán - a menos que se evite - cuando se ejecuten desde la ventana de instrucciones de R Commander. Tal salida puede ser suprimida listando los nombres de estas funciones en el archivo `log-exceptions.txt`.

Puede añadir código R al paquete, ej., para crear diálogos adicionales, colocando archivos con extensión `.R` en el directorio `etc`, además puede editar `Rcmdr-menus.txt` para proporcionar menús adicionales, submenús o items. Una demostración de esto se proporciona mediante el archivo `BoxCox.demo`. Para activar la demo, renombre el archivo a `BoxCox.R` y descomente la correspondiente línea del menú en `Rcmdr-menus.txt`. De forma alternativa, puede editar el código del paquete y recompilarlo.

Algunas funciones son proporcionadas para ayudar a escribir diálogos y la información del estado de `Rcmdr` en un emplazamiento separado. Mirar `help("Rcmdr.Utilities")` y el manual suministrado en el directorio `doc` del paquete de `Rcmdr` para mayor información.

Además, varias características son controladas mediante opciones, en tiempos de ejecución, establecidas por la orden `options("Rcmdr")`. Estas opciones deben ser establecidas antes de cargar el paquete. Si las opciones no están establecidas, que es la situación normal, serán usados los parámetros por defecto. Las opciones se especifican como una lista de pares `name$=$values`. Puede no establecer, establecer una, varias, o todas las opciones. Las opciones disponibles son las dadas a continuación:

`attach.data.set` Si es `TRUE` (por defecto `FALSE`), el conjunto de datos activo es fijado como la ruta de búsqueda.

`check.packages` Si es `TRUE` (por defecto), al arranque, la presencia de todos los paquetes recomendados de Rcmdr serán comprobados y si alguno no está instalado, Rcmdr preguntará si deben instalarse.

`command.text.color` El color de las órdenes en la ventana de resultados es, por defecto, `"red"`.

`console.output` Si es `TRUE` la salida será dirigida a la consola de R y la ventana de salida de R Commander no se mostrará. Por defecto es `FALSE`.

`contrasts` Ofrece la misma función que la opción general `contrasts`; por defecto es `c("contr.Treatment", "contr.poly")`. Cuando se sale de Commander la opción `contrasts` vuelve a su valor preexistente. Observe que `contr.Treatment` es del paquete `car`.

`crisp.dialogs` Si es `TRUE`, los diálogos deben aparecer en la pantalla dibujada completamente, más que acumular dispositivo a dispositivo. Esta opción debería afectar sólo a versiones Windows de R, pero debe en cualquier caso ser inofensivo. Por defecto es `TRUE` bajo versiones Windows de R 2.1.1 y superiores y `FALSE` si no. Si está trabajando en Windows y encuentra que se incrementan los problemas de estabilidad, pruebe establecer esta opción a `FALSE`.

`default.font` La fuente por defecto, como la especificación de la fuente de X11, dada en cadena de caracteres. Si está especificado, este valor toma precedencia sobre el tamaño de la fuente por defecto (abajo). Esta opción es sólo para sistemas no-Windows.

`default.font.size` Tamaño, en puntos, por defecto de la fuente. Por defecto es 10 para sistemas Windows y 12 para otro sistemas, salvo especificación de lo contrario (mirar el item anterior). La fuente por defecto es `"*helvetica-medium-r-normal-*-xx*"`, donde `xx` es por defecto el tamaño de la fuente. Esta opción es sólo para sistemas no-Windows.

`double.click` Establecer a `TRUE` si quiere que un doble click con el botón izquierdo del ratón sirva para pulsar el botón por defecto en todos los diálogos. Por defecto es `FALSE`.

`error.text.color` Color de los mensajes de error; por defecto es `"red"`.

`etc` Establece la ruta del directorio que contiene los archivos de configuración de Rcmdr; por defecto el subdirectorio `etc` del paquete Rcmdr instalado.

`grab.focus` Establecer a `TRUE` para "mantener" el enfoque en la ventana actual de Tk, esto es, para prevenir que el enfoque sea cambiado a otra ventana Tk. En algunos sistemas, mantener el enfoque de esta forma, puede causar problemas. Por defecto es TRUE. Si experimenta problemas de enfoque, intente establecer esta opción a `FALSE`.

`load.at.startup` Vector de caracteres de nombres de los paquetes que deben ser cargados cuando el paquete `Rcmdr` es cargado; por defecto se carga sólo el paquete `car`. Otros paquetes requeridos serán cargados cuando se necesiten. Si esto está disponible, el paquete `car` será cargado cuando Commander se inicie en cualquier caso.

`log.commands` Si es `TRUE` (por defecto), los comandos son repetidos en la ventana de instrucciones; si es `FALSE`, la ventana de instrucciones no se muestra.

`log.font.size` Tamaño de la fuente, en puntos, que es usado en la ventana de instrucciones, en la ventana de resultados, en diálogos recodificados y en expresiones de cáculo, esto es, donde es usada una fuente monoespacio. Por defecto es 10 para sistemas Windows y 12 para otros sistemas.

`log.height` La altura de la ventana de instrucciones, en líneas. Por defecto es 10. Estableciendo log.height a 0 tiene el mismo efecto que establer `log.commands` a `FALSE`.

`log.text.color` Color del texto de la ventana de instrucciones; por defecto es `"black"`.

`log.width` La anchura de la ventana de instrucciones y la de salida, en caracteres. Por defecto es 80.

`multiple.select.mode` Afecta a la forma en la que múltiples variables son seleccionadas en una caja de listas de variables. Si se establece a `"extended"` (por defecto), el botón izquierdo en una variable selecciona ésta y deselecciona cualquier otra variable que estuviera seleccionada; `Control+botón izquierdo` acciona la selección (y puede ser usado para seleccionar variables adicionales); `Mayúsculas+botón izquierdo` extiende la selección. éste es el convenio estándar de Windows. Si lo establece a `"multiple"`, el botón izquierdo acciona la selección de una variable y puede ser usado para seleccionar más de una variable. éste es el comportamiento de Rcmdr antes de la versión 1.9-10.

`output.height` Altura de la ventana de resultados, en líneas. Por defecto es dos veces la altura de la ventana de instrucciones o 20 si la ventana de instrucciones es suprimida. Establecer `output.height` a 0 tiene el mismo efecto que `console.output` a `TRUE`.

`output.text.color` Color de la salida en la ventana de resultados, por defecto es `"blue"`.

`placement` Emplazamiento de la ventana de R Commander, en píxeles; por defecto es `"$-40+20$"`, lo que pone la ventana cerca de la esquina superior derecha de la pantalla.

`plugins` Vector de caracteres con los nombres de paquetes de plugins de Rcmdr a cargar cuando Commander arranque. Los paquetes plugins también pueden ser cargados desde el menú `Herramientas -> Cargar paquete(s)`.

`suppress.menus` Si es `TRUE`, la barra de menús y de herramientas de R Commander son suprimidas, permitiendo que otro programa (como Excel) asuma esas funciones. Por defecto (por supuesto) es `FALSE`.

`suppress.X11.warnings` En (algunos) sistemas Linux X11 se generan múltiples advertencias por las órdenes de Rcmdr, después de abrir la ventana del dispositivo gráfico. Establecer esta opción a `TRUE` (por defecto cuando arranca interactivamente bajo X11 antes de la versión de R 2.4.0) suprime la aparición de estas advertencias. Un efecto secundario indeseable es que entonces todas las advertencias y mensajes de error son interceptados por Rcmdr, incluso para las instrucciones introducidas en los avisos de R. Los mensajes producidos por tales órdenes serán impresos en la ventana de mensajes de R Commander después de la siguiente orden generada en Rcmdr. Algunas advertencias de X11 puede ser impresas al salir de R Commander. Este problema sólo se aplica a versiones de R anteriores a 2.4.0 y el valor por defecto de la opción es establecido por consiguiente.

`retain.messages` Si es `TRUE` (por defecto `FALSE`), el contenido de la ventana de mensajes no es borrado entre mensajes. En cualquier caso, un mensaje "NOTE" no borrará un anterior "WARNING" o "ERROR".

`RExcelSupport` Establecido como `TRUE` (por defecto es `FALSE`), los menús y salidas son dirigidas a Excel.

`scale.factor` Factor de escala aplicado a todos los elementos Tk, como las fuentes. Esto funciona bien sólo en Windows. Por defecto es `NULL`.

`showData.threshold` Si el número de variables en el conjunto de datos activos excede este valor (por defecto, 100), entonces `edit()`, más que `showData()`, es utilizado para exhibir el conjunto de datos. Un inconveniente es que el control no se devuelve a Commander hasta que la ventana de edición sea cerrada. La razón de esta opción es que `showData()` es muy lento cuando el número de variables es grande; fijando el umbral a 0 suprime el uso en conjunto de showData.

`show.edit.button` Fijar a `TRUE` (por defecto) si quiere un botón `Editar` en la ventana de Commander, que permita editar el conjunto activo de datos. Los usuarios de Windows pueden desear establecer esta opción a `FALSE` para suprimir el botón `Editar` porque cambiando los nombres de las variables en el editor de datos se puede causar que R falle (aunque este problema se cree solucionado).

`sort.names` Fijar a `TRUE` (por defecto) si se quiere ordenar alfabéticamente el nombre de las variables en una lista de variables.

`tkwait` Esta opción trata un problema que, en mi conocimiento, es raro y puede ocurrir en algunos sistemas no Windows. Si R Commander causa que se cuelgue R, entonces establezca la opción `tkwait` a `TRUE`; o conserve la opción en `FALSE` e ignórelo. Un indeseable efecto secundario de establecer la opción `tkwait` a `TRUE` es que el aviso de órdenes de la sesión de R es suprimido hasta salir de R Commander. Uno sin embargo todavía puede introducir órdenes por la ventana de instrucciones. En particular, no hay razón para usar esta opción bajo Windows y no se debería usar con la GUI de R en Windows con salida protegida cuando la salida esté dirigida a la consola de R.

`use.rgl` Si es `TRUE` (por defecto), el paquete `rgl` será cargado si está presente en una librería accesible, si es `FALSE`, el paquete `rgl` será ignorado aunque esté disponible. El paquete rgl puede a veces causar problemas cuando se arranca R bajo X11.

`warning.text.color` Color de los mensajes de advertencia; por defecto es `"darkgreen"`.

Muchas opciones pueden también ser establecidas mediante el menú *Archivo* -> *Opciones*, que reiniciará R Commander después de que las opciones sean establecidas.

Si quiere lanzar R Commander cuando inicie R, puede incluir la siguiente instrucción en uno de los ficheros de inicio de R (por ejemplo, en el fichero `Rprofile.site` de la carpeta `etc` de R):

```
local({
old <- getOption("defaultPackages")
options(defaultPackages = c(old, "Rcmdr"))
})
```

Las opciones de R Commander puede ser establecidas de forma permanente de la misma forma. Para más información sobre el inicio de R, véase `?Startup`.

Avisos

La ventana de instrucciones de R Commander no proporciona una verdadera consola a R y tiene ciertas limitaciones. No se recomienda usar R Commander para la programación sería o el análisis de datos que confíe primordialmente en instrucciones - usar un editor de programación en su lugar. Por ejemplo, para declaraciones de composiciones de R incluidas entre llaves `"\{ \}"`, incluyendo definición de funciones, no serían analizadas ni ejecutadas correctamente, aunque si las líneas después de las primeras que estén identandas. Puede ejecutar declaraciones de composiciones desde la ventana de instrucciones separando los comandos dentro de las llaves por puntos y comas.

Problemas Conocidos

Ocasionalmente, bajo Windows, después de teclear algún texto en un cuadro de diálogo (ej. subconjunto de expresiones en el diálogo de subconjunto de conjunto de datos), algunos botones en el diálogo (ej. el botón Aceptar) pueden no tener efecto cuando sean presionados. Pulsando en cualquier

sitio, dentro o fuera del cuadro de diálogo, debería restaurarse las funciones de los botones. Por lo que se ha podido comprobar, éste es un problema con Tcl/Tk de Windows.

Note

Esta versión debe ser compatible con SciViews, que actualmente sólo funciona bajo sistemas Windows: `http://www.sciviews.org/SciViews-R`; mirar `Rcmdr.sciviews-specific`. Bajo Windows, el paquete `Rcmdr` puede también funcionar bajo de Rgui en modo SDI (interfaz de único documento) o bajo `rterm.exe`; puede ser que experimente problemas ejecutando `Rcmdr` bajo `ESS` con NTEmacs o XEmacs.

Author(s)

John Fox `<jfox@mcmaster.ca>` (de la versión inglesa)

Manuel González (traductor) `<gonzalezperezmanuel@gmail.com>`

Manuel Muñoz Márquez (traductor–revisor) `<manuel.munoz@uca.es>`

Véase `http://knuth.uca.es/R/doku.php?id=equipotraduccion`

La última versión de este fichero la puede encontrar en `http://knuth.uca.es/repos/R-contribuciones`

See Also

Plugins

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 lowercase 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

`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 selected) 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. If you select several variables in the variable-list box, clicking on the `+`, `*`, or `:` button will enter them into the model 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`

`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"), xlab=deparse(substitute(x)),
ylab=scale, main="", ...)
```

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".
xlab	x-axis label, defaults to name of variable.
ylab	y-axis label, defaults to value of scale.
main	main title for graph, defaults to empty.
...	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

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

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)
```

---

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 selected) or to the right-hand side. You can also enter operators and parentheses using the buttons above the formula. If you select several variables in the variable-list box, clicking on the `+`, `*`, or `:` button will enter them into the model 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`

`mergeRows` *Function to Merge Rows of Two Data Frames.*

Description

This function merges two data frames by combining their rows.

Usage

```
mergeRows(X, Y, common.only = FALSE, ...)

## S3 method for class 'data.frame':
mergeRows(X, Y, common.only = FALSE, ...)
```

Arguments

`X`	First data frame.
`Y`	Second data frame.
`common.only`	If `TRUE`, only variables (columns) common to the two data frame are included in the merged data set; the default is `FALSE`.
`...`	Not used.

Value

A data frame containing the rows from both input data frames.

Author(s)

John Fox

See Also

For column merges and more complex merges, see `merge`.

Examples

```
require(car)
D1 <- Duncan[1:20,]
D2 <- Duncan[21:45,]
D <- mergeRows(D1, D2)
dim(D)
```

---

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

```
library(car)
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

`Plugins` *R Commander Plug-in Packages*

Description

Plug-ins are R packages that extend the R Command interface.

Details

An R Commander plug-in is an ordinary R package that (1) provides extensions to the R Commander menus is a file named `menus.txt` located in the package's `etc` directory; (2) provides call-back functions required by these menus; and (3) in an `Models:` field in the package's `DESCRIPTION` file, augments the list of model objects recognized by the R Commander. The menus provided by a plug-in package are merged with the standard Commander menus. It is also possible to remove menus and menu items from the standard Commander menu file or from the files of plug-ins installed before the current one.

Plug-in packages given in the R Commander `plugins` option (see Commander) are automatically loaded when the Commander starts up. Plug-in packages may also be loaded via the Commander *Tools -> Load Rcmdr plug-in(s)* menu; a restart of the Commander is required to install the new menus. Finally, loading a plug-in package when the **Rcmdr** is not loaded will load the **Rcmdr** and activate the plug-in.

An illustrative R Commander plug-in package, **RcmdrPlugin.TeachingDemos**, is available on CRAN.

For more details, see my (slightly out-of-date) article on "Extending the Rcmdr by Plug-in Packages" in the December 2007 issue of *R News* <http://www.r-project.org/doc/Rnews/Rnews_2007-3.pdf>.

See Also

Commander

`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>

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()
aicP()
checkActiveDataSet()
checkActiveModel()
checkBoxes(window=top, frame, boxes, initialValues=NULL, labels, title=NULL)  # mac
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)
closeCommander(ask=TRUE, ask.save=ask)
closeDialog(window, release=TRUE)  # macro
CommanderWindow()
dataSetsP(n=1)
defmacro(..., expr)
dialogSuffix(window=top, onOK=onOK, onCancel=onCancel, 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)
formulaFields(model, hasLhs=TRUE, glm=FALSE)
## 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)
Library(package, pos=4)
listAllModels(envir=.GlobalEnv, ...)
listAOVModels(envir=.GlobalEnv, ...)
listDataSets(envir=.GlobalEnv, ...)
listFactors(dataSet=ActiveDataSet())
listGeneralizedLinearModels(envir=.GlobalEnv, ...)
listLinearModels(envir=.GlobalEnv, ...)
listMultinomialLogitModels(envir=.GlobalEnv, ...)
listNumeric(dataSet=ActiveDataSet())
listPlugins(loaded=FALSE)
listProportionalOddsModels(envir=.GlobalEnv, ...)
listTwoLevelFactors(dataSet=ActiveDataSet())
listVariables(dataSet=ActiveDataSet())
lmP()
logger(command)
LogWindow()
MacOSXP()
Message(message, type=c("note", "error", "warning"))
MessagesWindow()
modelFormula(frame=top, hasLhs=TRUE)  # macro
modelsP(n=1)
multinomP()
nobs(model)
Numeric(names)
numericP(n=1)
OKCancelHelp(window=top, helpSubject=NULL, model=FALSE)  # macro
OutputWindow()
packageAvailable(name)
polrP()
popCommand()
popOutput()
putRcmdr(x, value)
radioButtons(window=top, name, buttons, values=NULL,
initialValue=..values[1], labels,
```

```
title="", title.color="blue", right.buttons=TRUE)  # macro
RcmdrTclSet(name, value)
RcmdrTkmessageBox(message, icon=c("info", "question", "warning",
    "error"), type=c("okcancel", "yesno", "ok"), default, title="")
rglLoaded()
sortVarNames(x)
subOKCancelHelp(window=subdialog, helpSubject=NULL)  # macro
subsetBox(window=top, model=FALSE)  # macro
tclvalue(x)
trim.blanks(text)
TwoLevelFactors(names)
twoLevelFactorsP(n=1)
UpdateModelNumber(increment=1)
variableListBox(parentWindow, variableList=Variables(), bg="white",
    selectmode="single", export="FALSE", initialSelection=NULL,
    listHeight=getRcmdr("variable.list.height"), title)
Variables(names)

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

commanderPosition()
```

Arguments

ask	ask for confirmation.
ask.save	ask whether to save contents of script and output windows.
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?
expr	expression constituting the body of the macro; typically a compound expression.
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.
`glm`	`TRUE` if the model is a `glm` object, `FALSE` otherwise.
`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.
`loaded`	if `TRUE`, plug-in packages that are loaded are included in the vector of names returned.
`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 a model object, 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.
`onCancel`	function to execute when the *Cancel* button or *Esc* key is pressed.
`package`	quoted name of package to load.
`plotLinesByGroup`	include a check box for plotting lines by group?
`plotLinesByGroupsText`	the label for the plot-lines-by-group check box.
`pos`	position on search path at which to load package; default is `4`.

`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.

`right.buttons`
: radio button placed to right of button-labels; defaults to `TRUE`.

`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.

`title.color`
: color for title above radio buttons; defaults to `"blue"`.

`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, or a tcl variable or object, or a vector of variable names to be sorted.

`...`
: For `gettextRcmdr`, text string or vector of text strings to translate; for `defmacro`, arguments for the macro; 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. The R Commander maintains a list of output objects, by default including the last 10 outputs. `popOutput()` "pops" (i.e., returns and removes) the first entry of the output stack. Note that, as a stack, the queue is LIFO ("last in, first out"). There is also a stack of commands, which is accessed similarly by `popCommand()`.

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`, `listAOVModels`, `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`.

"Themed" Tk widgets: Tk 8.5 introduced so-called "themed" widgets, which look better than the traditional Tk widgets. Several functions, contributed by Brian Ripley, are written to access the new widgets by switching automatically between the new and old widget sets depending upon the availability of the former: `buttonRcmdr`, to access either `ttkbutton` or `tkbutton`; `labelRcmdr`, to access either `ttklabel` or `tklabel`; `ttkentry`, to access either `ttkentry` or `tkentry`; `ttkframe`, to access either `ttkframe` or `tkframe`; `ttkradiobutton`, to access either `ttkradiobutton` or `tkradiobutton`; and `ttkscrollbar`, to access either `ttkscrollbar` or `tkscrollbar`. Note that the last four functions mask functions of the same names in the **tcltk** package.

'Predicate' functions: A number of functions of the form *name*`P` are 'predicate' functions, which return `TRUE` or `FALSE` depending upon whether some condition obtains. For example, `lmP()` returns `TRUE` if there is an active model that is a linear model; and `factorsP(2)` returns `TRUE` if there are at least two factors in the active data set.

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

Miscellaneous: The function `trim.blanks` removes spaces from the beginning and end of a character string. The function `installPlugin` installs an Rcmdr plug-in from a ZIP file or directory; this function may be used to create self-installing plug-ins in the form of packages. The function `nobs` returns the number of observations on which a statistical model is based. The function `formulaFields` returns information about the left-hand side, right-hand side, data, subject, and (for GLMs) family and link, of a model object. The function `sortVarNames` sorts variable names, including those containing numerals, into a more "natural" order than does the standard `sort` function. The function `Library` may be used to load packages; it checks whether a package is already loaded, and if not by default puts it in position 4 on the search path.

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.

`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`

`rcorr.adjust` *Compute Pearson or Spearman Correlations with p-Values*

Description

This function uses the `rcorr` function in the **Hmisc** package to compute matrices of Pearson or Spearman correlations along with the pairwise p-values among the correlations. The p-values are corrected for multiple inference using Holm's method (see `p.adjust`). Observations are filtered for missing data, and only complete observations are used.

Usage

```
rcorr.adjust(x, type = c("pearson", "spearman"))

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

Arguments

`x`	a numeric matrix or data frame, or an object of class `"rcorr.adjust"` to be printed.
`type`	`"pearson"` or `"spearman"`, depending upon the type of correlations desired; the default is `"pearson"`.
`...`	not used.

Value

Returns an object of class `"rcorr.adjust"`, which is normally just printed.

Author(s)

John Fox, adapting code from Robert A. Muenchen.

See Also

`rcorr`, `p.adjust`.

Examples

```
require(car)
rcorr.adjust(Mroz[,c("k5", "k618", "age", "lwg", "inc")])
rcorr.adjust(Mroz[,c("k5", "k618", "age", "lwg", "inc")], type="spearman")
```

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 `lo` and `hi` may appear in a range. For example, `lo: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`

`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

```
library(car)
data(DavisThin)
reliability(cov(DavisThin))
```

`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-package`, `gam`

stepwise *Stepwise Model Selection*

Description

This function is a front end to the stepAIC function in the **MASS** package.

Usage

```
stepwise(mod,
    direction = c("backward/forward", "forward/backward", "backward", "forward"),
    criterion = c("BIC", "AIC"), ...)
```

Arguments

mod	a model object of a class that can be handled by stepAIC.
direction	if "backward/forward" (the default), selection starts with the full model and eliminates predictors one at a time, at each step considering whether the criterion will be improved by adding back in a variable removed at a previous step; if "forward/backwards", selection starts with a model including only a constant, and adds predictors one at a time, at each step considering whether the criterion will be improved by removing a previously added variable; "backwards" and "forward" are similar without the reconsideration at each step.
criterion	for selection. Either "BIC" (the default) or "AIC". Note that stepAIC labels the criterion in the output as "AIC" regardless of which criterion is employed.
...	arguments to be passed to stepAIC.

Value

The model selected by stepAIC.

Author(s)

John Fox <jfox@mcmaster.ca>

References

W. N. Venables and B. D. Ripley *Modern Applied Statistics Statistics with S, Fourth Edition* Springer, 2002.

See Also

stepAIC

Examples

```
# adapted from ?stepAIC in MASS
require(MASS)
example(birthwt)
birthwt.glm <- glm(low ~ ., family = binomial, data = bwt)
stepwise(birthwt.glm, trace = FALSE)
stepwise(birthwt.glm, direction="forward/backward")
```

Index