pmg

John Verzani
pmgRgui@gmail.com

April 10, 2007

Abstract:

The `pmg` package provides a simple GUI for R using the GTK2 toolkit. the `pmg` GUI is similar in intent to the more mature `Rcmdr` GUI which uses the `tcltk` toolkit. The package was designed to lessen R's learning curve for students in introductory course. The GTK libraries are accessed through the `RGtk2` package of Michael Lawrence. This package is available for all three major platforms: windows, Mac OS X, and Unix/X11. the `gWidgets` interface is used to simplify the development of interactive GUIS, this in turn requires the `gWidgetsRGtk2` package. [1]

1 Installing pmg

If the requisite packages are available on CRAN, then installation can be done with a single command such as

```
> install.packages("pmg", dependencies=TRUE)
```

or from the toolbar. Mac users may need to add `source=TRUE`.

The above assumes that both the GTK libraries, the `RGtk2` package, and the `cairoDevice` package already exist on your system. If not, the home page for `RGtk2` has some directions (`http://www.ggobi.org/rgtk2`).

For users of windows, a script is available to download the gtk libraries and install `pmg`. Run the command

[1] In theory, but not practice, the usage of `gWidgets` would allow `pmg` to use a different GUI toolkit, the `gWidgetsrJava` package does not have enough features in it currently to do so.

```
> source("http://www.math.csi.cuny.edu/pmg/installpmg.R")
```

to initiate the script.

For users with the `iplots` or `reshape` packages installed, there are extra dialogs appearing under the menu bar.

2 Starting

The GUI uses the **gWidgets** API. The **gWidgets** package offers the promise of a toolkit-independent API for writing interactive GUIs. If more than one toolkit implementation is installed, a menu for selection will be shown. It is likely that **pmg** will only work well using the **gWidgetsRGtk2** implementation

The GUI is started as follows.

```
> library(pmg)
```

To restart the GUI, the command `pmg()` can be issued. With the Windows operating system it is best to minimize the parent R GUI as it covers up the pmg windows.

3 Using the GUI

The GUI basically consists of a variable browser and a collection of dialogs that are accessed through a menubar. The values in the variable browser may be dragged and dropped into the entry areas of the dialogs.

Most of the dialogs have a common format familiar to web users: you enter in the values in a form and click "OK." The output is then sent to the console and the "Commands" tab.

Besides the menubar, the GUI has a few key features. From left to right these are:

Some drop targets on the left hand side These are for editing the dropped object, for plotting the dropped object, for summarizing the drop object, or for deleting the dropped object.

The variable browser This shows the variables in the global environment using a familiar tree interface. The filter tab allows you to display just some desired types. The definitions of which may not be as inclusive as you would like. The

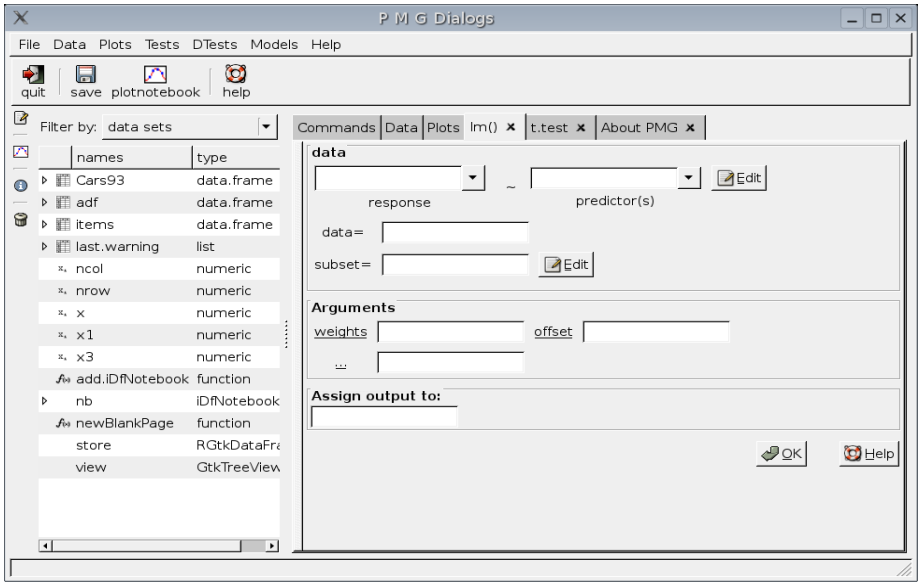

Figure 1: The pmg gui with sample dialog

base objects are refreshed every few seconds. Unfortunately, their subobjects are not. To refresh the subobjects, close then open the sub display.

These variables may be dragged and dropped. For instance, dropping on the left sidebar produces the actions described previously.

If the variables are double-clicked a summary of the object appears in a tab in the dialog notebook on the right.

The dialog notebook This notebook (a widget with tabbed pages) has two permanent tabs: a command line and a data frame viewer/editor. Other dialogs have a close button in the tab that removes that dialog.

The tabs have a drag motion set so that if an object is dragged over the tab, the corresponding page opens.

The "Commands" tab The command line has two states: one for editing the current command, and one for viewing the command with its output. Toggling between is done with the buttons immediately above the display. The command line processes chunks of R code, in the style of sourcing a .R file. A history of the last 25 commands issued is available.

Dragging variables to the command line drops their name into the window.

The dataframe viewer/editor The dataframe viewer (Figure 2) allows one to view and edit data frames. It should look more spreadsheet like, but no such widget is available in GTK. This implementation allows you to edit single cells in the table, and edit row and column names.

Editing is a little clunky. Double clicking on a cell should allow you to edit it. Hitting the enter key, jumps to the next row done and opens the edit area. To move to another column use the mouse or the arrows and the enter key.

A right-mouse popup menu allows for sorting by column values, applying a function to a column, or changing the column name. This popup is not bound to the column header, but rather the column contents. Click once in a column, then right click to see it.

A right-mouse popup on the notebook tab holding the name allows you to save, close, or rename the sheet.

The sheets with the names `*scratch:#*` are treated differently. The non-empty individual columns are saved as data vectors. Otherwise a saved sheet is saved as a data frame.

Dropping an appropriate variable on the open button will open a new tab with that variable's contents displayed. Whereas, clicking the open button opens a dialog allowing the selection of a data frame.

Dialogs Most of the menu items create dialogs that are attached as notebook pages to the right of the data frame viewer/editor.

There are primarily two types of dialogs in pmg: static ones and "dynamic" ones. The latter have menu items prefaced with an "execute" icon, and appear in separate windows. These dynamic dialogs were inspired by the Fathom software package (`http://www.keypress.com/fathom/`).

Dynamic dialogs respond to drag and drop actions. By dragging variables onto the appropriate area, the values in the widget get updated. Additionally, the bold face values in the widget can be clicked to effect changes. Again, the widget refreshes instantly. These dynamic dialogs give immediate feedback, but do not show or allow one to edit the R commands that are used.

There are currently 4 main dynamic dialogs: one for exploring graphically (the Lattice Explorer), one for finding numeric summaries, one for significance tests, and one for linear models.

The static dialogs are illustrated by the sample dialog appearing in Figure 1. Most, but not all, of the dialogs have the same behavior described below.

There are areas to enter information, and when enough values are specified, the "ok" button sends the appropriate command to the "Commands" tab area to be evaluated. Unlike the dynamic dialogs, the value is only updated when the "ok" button is clicked. Empty values are ignored. Values may need to be quoted.

This particular dialog in Figure 1 expects a model formula, in this case for calling the `plot()` function.

There are some conveniences for editing the information. When a data set is dropped onto the `data=` area the names of the data frame are added to the dropdown values for the response and predictor. Additionally, the "edit" button opens a dialog for editing the formula. A similar button allows one to subset the data frame.

The "arguments" section of the dialog has underlined labels. When these are clicked the appropriate help page section for this argument is looked for and displayed if found (this involves a grep for the colon following the argument and may miss). In this dialog, the help page refers to that for `plot()`, but the arguments are not found. The "help" button will show the full help page.

When the "ok" button is pressed, the arguments are collected and pasted together to make a function call. This is then sent to the command area and evaluated. To see the output, you need to click the command tab. As well, the command and output is printed into the console.

Dialogs are accessed through the menubar. In addition to several of the "generic" dialogs described above, there are some that have their own window. For example to load a data set so that it is visible in the variable browser the "Load data set..." menu item opens up a dialog for this. Other dialogs are available for loading an installed package, installing a package from CRAN, sourcing a file, saving a file, etc.

The dialog from "Data::Univariate Summaries::quantile" is new as of version 0.9-24. It is a possible prototype for some student-friendly dialogs along the lines of the dynamic summaries dialog. For students there is also a collection of teaching demos under the "Plots" menu.

Graphs can be handled by the default device, through the plot notebook (Figure 3) which is opened when the "plotnotebook" toolbar item is pressed, or using the Lattice Explorer. The notebook allows one to store several different plots at once. The currently selected tab, should also be the current device. (Using both the plot notebook and the Lattice Explorer can lead to confusion, as the current device is where the new figure is drawn, and this may not be the desired device.)

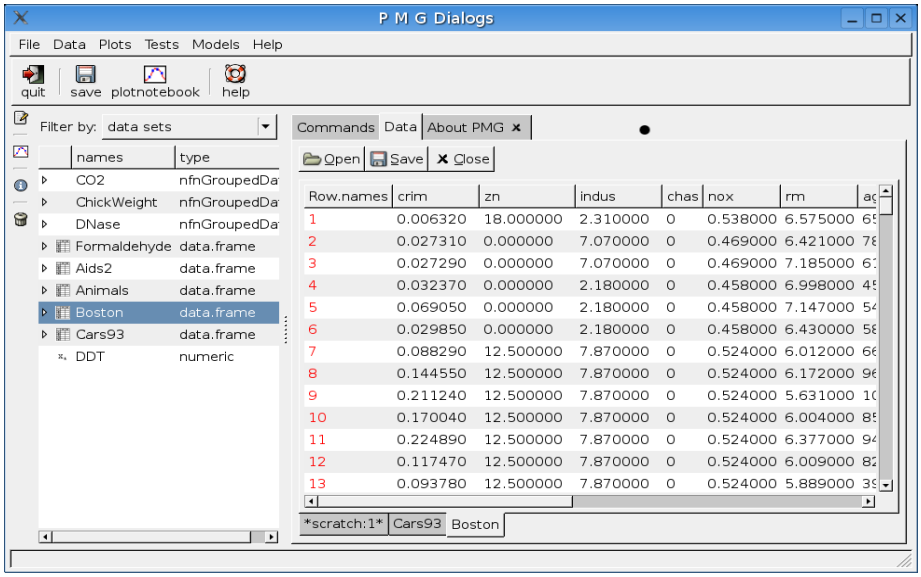

Figure 2: The data viewer and editor. The columns may be dropped onto the dynamic dialogs.

The "help" toolbar item opens up a help page browser, allowing access to R's help system.

4 A few sample tasks

We describe how to perform a few common tasks from an introductory statistics course.

4.1 Loading a data set and using a boxplot to explore a variable

Let's see how to make boxplots of the `MPG.highway` variable broken up by the number of `Cylinders` in the data set `Cars93` of the `MASS` package. The first task is to load the data set. Of course, you might think:

```
> library(MASS); boxplot(MPG.highway ~ Cylinders, Cars93)
```

but we are interested in doing this through the GUI.

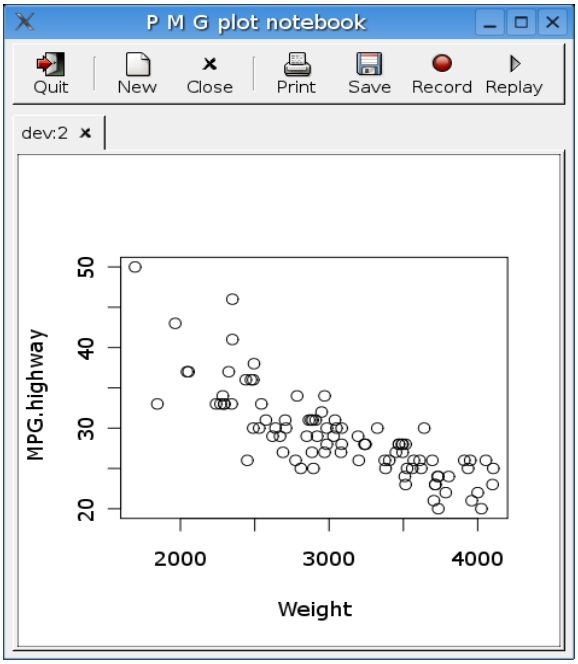

Figure 3: A notebook for holding multiple plot devices

First we load the `MASS` package using the "Load package" dialog under the File menu. Open the dialog, and then double click on MASS until the second column says TRUE.

Next, we call `data` on the data set so that it appears in the variable browser. Open the "Load data set..." dialog under the Data menu. Find the data set and then double click. The data set should appear in the variable browser shortly.

Now open the Lattice Explorer dialog under the Plots menu. Change the plot selection to `bwplot`. Then in this order drag the `MPG.highway` variable and then the `Cylinders` variable onto the main explorer area. The appropriate graph should be drawn. The basic idea is that the first variable is split up by the second, and in this case a third is possible. (Drop `Origin` to see.)

To look at new variables clear out the old ones with the clear button. To make a different graphic with the same variables, you only need to change the graph selection popup.

4.2 Using the subset feature of the data viewer

An alternative approach to using lattice graphics to break up the data by levels of some factor can be to use the `subset=` dialog for the data viewer. Changing the values shown in the data viewer, can dynamically update the graphic.

To illustrate, open the Lattice Explorer as before and change the plot selection to `bwplot`. Now drag the variable `Cars93` from the variable browser area *over* the Data tab of the dialogs area (to open up that tab) and drop the `Cars93` variable onto the area where a data frame is shown. This should cause the data viewer to open and display the data frame. Now drag the `MPG.highway` variable name (the column header) onto the Lattice Explorer dialog and drop it in the graphing area. A boxplot of the variable is drawn. This boxplot is linked to the values displayed in the `MPG.highway` column. Changing a value causes the graphic to be redisplayed taking into account the new values. We will change the values with the `subset=` dialog. At the bottom of the data viewer is an expander group that must be clicked to show the dialog. After this appears, you can select a variable and a subsetting argument to filter the displayed values. Figure 4 shows the dialog with only the 3 cylinder cars shown.

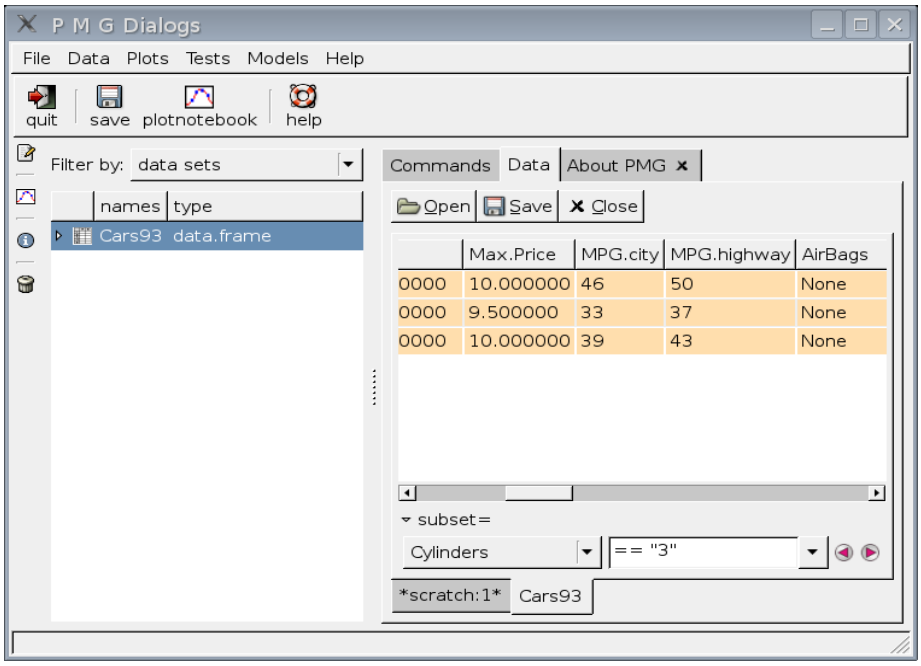

Figure 4: Data viewer showing `Cars93` data set subsetted by the number of `Cylinders` being 3.

If you scroll through the levels of `Cylinder` using the arrow keys, the boxplot in the Lattice Explorer will be redrawn each time using only the visible data.

Dragging a column header onto a widget only works for the dynamic widgets and only with `gWidgetsRGtk2`. There are currently two other dynamic widgets: one for numeric summaries of a data set and another for linear models.

4.3 using iplots

If the **iplots** package is installed the dialog found under `Plots::iplots` can also be used to dynamically adjust a graphic to a mouse click. To see how the highway mileage varies with the number of cylinders involves the following steps: open the dialog under `Plots::iplots`. Then add the data frame `Cars93`. Then select the variable `MPG.highway` on the left and then under `New plot` select the `ihist` choice. The histogram is created. Now select the `Cylinders` variable, but this time make a bar plot using `ibar`. Another graphic is opened. Clicking on one of the bars in the barplot causes those cases to highlight in the histogram. One can also select cases by dragging over multiple values of the bar plot.

There are plans for a similar interface to **rggobi**.

4.4 Finding numeric summaries of a data set

The Dynamic Summaries dialog found under the Data menu allows one to easily find some common numeric summaries. The interface is intended to make the simplest cases easy, if more control over the arguments is needed then the respective functions have an alternative dialog.

We consider again the `Cars93` data set and its `MPG.highway` variable. To find the mean of all the data, drag this to the bold-faced area "Drop variable here." The mean should be calculated and appear below. To break up the data by the levels of some other variable drag that to the "group by" area. Figure 5 shows the result (after resizing the window) of breaking the data up by the `Cylinders` variable.

Changing the summary using the popup will compute the new numeric summary. If the variables need to be changed, one can drag a new variable directly, or if the size is not compatible, clear the variables first then drag a new variable.

The drop areas can be edited by clicking on the bold text (familiar to `gmail` users). Press enter when done editing. The variable name can be directly typed in relative to the global environment. Even simple functions can be applied. One caveat, if the dropped variable comes from the data frame viewer then editing won't

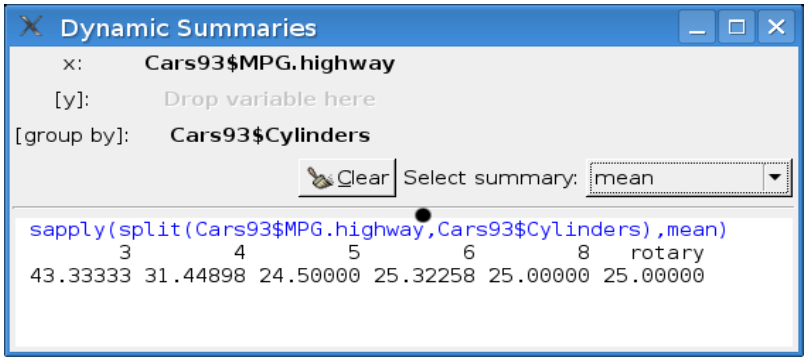

Figure 5: A dialog for computing summaries of numeric data

work. However, editing the data within the data frame viewer should have its changes instantly propogate.

5 Adding to the GUI

Currently there are only a few ways to add to the GUI.

5.1 Adding a dialog

The `pmg.add(widget, label)` function is used to add a dialog to the main notebook. It will add to the third tab. The widget is some `gWidget` instance. It should not already be attached to some container. For example, this will add a "hello world" tab

```
> pmg.add(glabel("Hello world"), label="Hi")
```

5.2 Creating a dialog to collect a function's arguments

The `pmg.gw(lst,title=NULL)` function is used to add a dialog to collect a function's argument. The function calls `ggenericwidget` to create the dialog. The `lst` argument can be a list, a function name or a function. In the last two cases the dialog is formed from looking at the values returned from `formals()`. The title can be set or will be guessed if not present.

5.3 Adding to the menubar

The `pmg.addMenubar(menulist,...)` function can be used to add new top-level entries to the menu bar. It takes a list that gets passed to `gmenu` as its argument.

For example, a new entry can be defined as follows

```
> lst = list()
> lst$MenuEntry$hi$handler = function(h,...) pmg.add(glabel("HelloWorld"), label="H
> lst$MenuEntry$mean$handler = function(h,...) pmg.gw("mean")
> pmg.addMenubar(lst)
```

User-defined menus can be added when the GUI is started. At startup pmg looks for a variable `pmg.user.menu` which if present should be a list with named components, each again a list as above. See the help page for `gmenu()` of `gWidgets` for more details on its structure.

6 TODO

The pmg GUI could be improved in many ways. A few obvious ones are

- More ability to customize the GUI. For instance, there is no way to change the fonts, there is only a primitive way to change the menubar, ...
- The bulk of the dialogs are pretty basic – they just collect arguments to pass to a function call. Some more dynamic widgets would be useful. The dialog for quantiles is meant to be a template for some student-friendly dialogs.
- It would be great to have some sort of report writing tool. The new Maple interface or the Mathematica notebook interface might be a good choice for interacting with R.