Interactive Scatterplots

Elizabeth Whalen

May 30, 2006

1 Overview

1.1 Goals

The *iSPlot* package has three goals: to create views that are linked, views that are interactive, and a design that is extensible. Each goal is discussed in more detail in the following paragraphs.

Linked views mean that displays based on the same data must be simultaneously updated when the underlying data change. For example, consider two views where one view displays height versus weight and a second view displays age versus cholesterol level. Suppose that these four variables (height, weight, age, and cholesterol level) all come from the same data set and that one person, John Doe, has his height, weight, age, and cholesterol level stored in the data set. Then John Doe has one point in the view of height versus weight and also has one point in the view of age versus cholesterol level. Now for these views to be linked, these points on different views that correspond to the same entity (John Doe) must have a similar appearance or some type of visual indication that these points refer to the same entity. One example is to have the points that represent John Doe's values both appear as red circles. This visual similarity of the points lets users quickly see on the two views, which points correspond to John Doe.

Thus, for the views to be linked, certain view information, such as the color of the data point, whether the point is hidden or highlighted, and the plotting character of the point, needs to be stored in a location where all views can access it. The view information that needs to be stored in this location pertains to the visual representation of a data point, which allows all views that display this data point to render it in the same fashion.

Having interactive views means that when the user interacts with a view, a response occurs. As an example of interactivity, suppose that when the user clicked on a point in a view, the point was colored red. The interactivity is richer and more flexible if multiple events can be noticed (such as a mouse click, a mouse movement, or a key press event) and if the response to those events can be modified. To provide this interactivity, the *iSPlot* package requires the *RGtk* and *gtkDevice* packages, which let the users interact with Gtk+ functions from the R interface. Gtk+ is an open-source window toolkit for creating user interfaces. More information about

these packages is given in Section 1.2.

Finally, an extensible design is imperative so that future users can make additions based on their needs. The design for creating linked, interactive views is based on the model-view-controller paradigm. The *MVCClass* package, which is required by the *iSPlot* package, defines classes and generic functions that implement the model-view-controller (MVC) paradigm. The MVC paradigm is a design that consists of three types of objects: the controller, which defines what actions occur in response to user input; the view, which consists of displays of the data; and the model, which manages the data. The power of the MVC design is that it decouples the views of the model from the model by creating a subscribe/notify procedure between them. In other words, the views subscribe to a particular model and the model must notify the views when a change occurs so that the views are updated. By separating the model from its views, only one copy of the model needs to be stored. Please see the Vignette for the *MVCClass* package for more information about the classes that are defined in that package. Note that the use of these classes allows an extensible design because the inheritance structure of the classes is set up so that users can create new model and view classes that inherit from already implemented classes.

Another place that has been designed for extensibility is the graphical user interface (GUI) that is defined in the *iSPlot* package. Here, the GUI has built-in menus for the existing functionality that is available. However, if the user wants to access some new functionality, then the user can add new menu items to the GUI through R command line functions. This addition of menus is discussed in Section 11.3.

One key thing to notice about the extensibility of the *iSPlot* package is that any additions the user may want to make can all be performed in the R language. If the user wants to define new model or view classes, this would be done in R and similarly if the user wants to add new menus, this would also be done in the R language. Thus, even though the *iSPlot* packages requires other packages that use other languages, such as C, making additions to the *iSPlot* package only requires that the user can program in R. More information about the extensibility of the *iSPlot* package can be found in Section 11.

1.2 Required Packages

The required packages for *iSPlot* are *RGtk*, *gtkDevice*, and *MVCClass*. A short description of why these packages are needed by *iSPlot* is given here. The *RGtk* package is used to create a GUI through which the user can load, view, plot and interact with the data. The *gtkDevice* package creates a device of type Gtk that looks like a X11 device, but a Gtk device can respond to events, such as a button press or a mouse over event. By generating a function call in response to a mouse or button press event, interactive views can be created. Finally, the *MVCClass* package defines the classes and generic functions that are used in the *iSPlot* package.

1.3 GUI versus Command Line Functions

The functionality that is implemented in the *iSPlot* package can be accessed either through the GUI or through command line functions. Both methods for accessing the functionality are explained in this vignette.

2 Getting Started

```
> library(iSPlot)
```

After loading the library, the user can either begin accessing the *iSPlot* functionality through the command line functions or if the user wants to use the GUI, then continue with Section 2.1 to learn how to open the GUI.

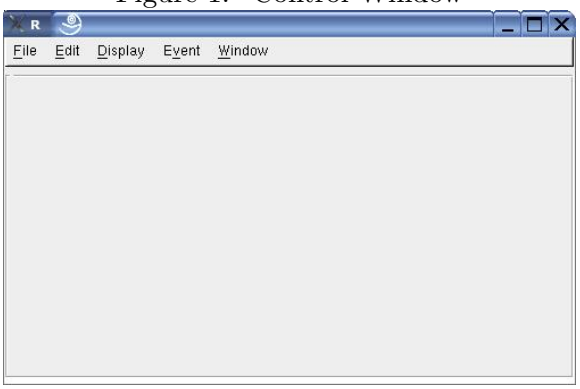
2.1 Opening the GUI

After loading the library, the first command is to open the control window, using `createControlWindow`, if the user intends to use the GUI. Data can be loaded, viewed and plotted by selecting menu items on the control window.

```
> if (interactive()) {
+     createControlWindow()
+ }
```

The control window is shown in Figure 1.

Figure 1: Control Window



3 Loading Data

Currently, this package can only load and create views of either data frames or matrices (i.e. only two-dimensional data). No other class of data can be loaded. If the user tries to load data that is not of type data frame or matrix, the user is notified that the data could not be loaded.

Before creating views of the data, it first must be loaded through the GUI or through the command line functions. The user can continue to load as many data sets as he/she would like through the GUI or through the command line functions, but only one data set can be considered the active data set. This concept is discussed in Section 4.

When a data set is loaded, an object of class `MVC` is created that stores the data set and all views of that data. The `MVC` class is defined in the *MVCClass* package and it is used to tie the model, view, and controller objects together into one object that revolves around one model (one data set). Since there is a one-to-one relationship between the model and the MVC object, they both are referred to by the same name and when the active data set is discussed, this also refers to the active MVC object.

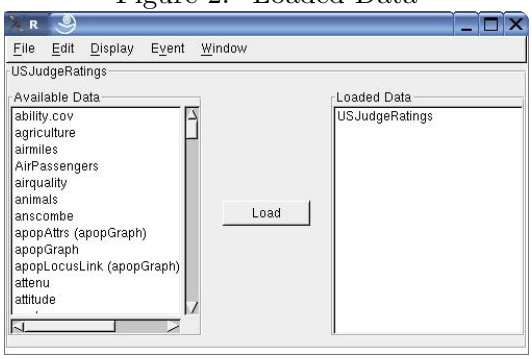
3.1 Loading Data Through the GUI

The two ways to load data through the GUI are to use either the Open Data menu item or the Open File menu item under the File menu. Either option loads the data into an environment where the data and any updates to the data are stored.

If the user wants to load data that is available from the function `data`, then highlight the File menu and select the Open Data menu item. Alternatively, the user can press Ctrl-D to activate the Open Data menu item.

To make the control window look as it is shown in Figure 2, complete the following steps. Choose Open Data under the File menu, highlight the list item, "USJudgeRatings", by scrolling down the Available Data list, and then click the Load button.

Figure 2: Loaded Data

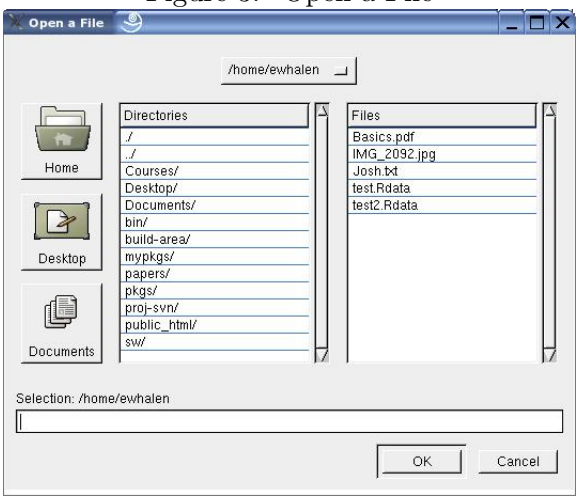


Note that once the "USJudgeRatings" data has been loaded, it no longer appears in the Available Data list on the left hand side of the control window. Also notice that the frame around the control window that is underneath the main menu now says "USJudgeRatings".

Instead, if the data the user wants to load is stored in a file, then highlight the File menu and select the Open File menu item. Similarly, the user can press Ctrl-O to activate the Open File menu item. A file explorer window opens allowing the user to choose which file they would like to load.

The file explorer window is shown in Figure 3.

Figure 3: Open a File



3.2 Loading Data Through the Command Line

The user can either load data that is available through the `data` function or that is stored in a file through the command line functions. The `loadData` function loads data that is available through the `data` function and the `loadFile` function loads data that is stored in a file. The example code below loads the "USArrests" and the "USJudgeRatings" data through the `loadData` function. Please see the man pages for these functions for more information.

```
> if (interactive()) {
+     loadData("USArrests")
+     loadData("USJudgeRatings")
+ }
```

4 Setting the Active MVC

As mentioned in Section 3, whenever a data set is loaded a MVC object is created that binds the data and its views into one unit. There is a one-to-one relationship between the MVC object and its data set so they are referred to by the same name. In the *iSPlot* package, the MVC object is considered the unit we work with. Though multiple MVC objects can be created (because multiple data sets can be loaded), only one MVC object can be active at one time.

The first data set loaded and thus, the first MVC object created automatically becomes the active MVC object. Thus, if the user initially loaded the "USJudgeRatings" data set, then the MVC object called "USJudgeRatings" is the active MVC object. Any other data sets loaded after this first data set are not the active MVC object. The user can tell what the active MVC object is by looking at the name shown in the frame around the control window, underneath the main menu, or by calling the function, `getActiveMVC`.

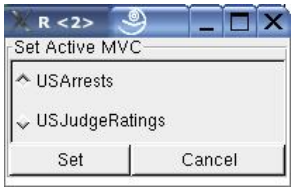
Being the active MVC object means that all of the menu items or the command line functions refer to operations performed on this MVC object. Thus, if the user loads "USArrests" after loading "USJudgeRatings", "USJudgeRatings" is the active MVC object and the user can create views of the "USJudgeRatings" data set. If the user would instead like to create views of "USArrests", then the user has to set "USArrests" to be the active data set. Setting the active data set can be performed through the Set Active MVC menu item under the Edit menu or by calling the function, `setActiveMVC`.

Once the interesting MVC object is the active MVC, the user can either view the data in a spreadsheet or go directly to plotting the data. Viewing the data is discussed in Section 5 and plotting the data is discussed in Section 6.

4.1 Setting the Active MVC Through the GUI

Suppose that the user loaded both "USJudgeRatings" and "USArrests" through the Open Data menu item as described in Section 3.1. Now to set "USJudgeRatings" as the active MVC through the GUI, highlight the Edit menu and select the Set Active MVC menu item. Similarly, the user can press Ctrl-A to activate the Set Active MVC menu item. A new window appears with radio buttons to allow the user to choose which of the MVC objects will be the active MVC. This new window appears as shown in Figure 4.

Figure 4: Set the Active MVC



Now if "USJudgeRatings" is the MVC object that the user is interested in, choose the "USJudgeRatings" radio button and click the Set button. Now the active MVC is "USJudgeRatings".

4.2 Setting the Active MVC Through the Command Line

To set the active MVC object through the command line, call the function, `setActiveMVC`, with the name of the MVC object that will become active. The example code below sets the "USJudgeRatings" MVC to be the active MVC. Please see the man page for this function for more information.

```
> if (interactive()) {
+     setActiveMVC("USJudgeRatings")
+ }
```

5 Viewing Data

After loading data using the methods described in section 3, the data can be viewed in an Excel-type format. Currently, each data set can be viewed in only one window. For example, if the "USJudgeRatings" data frame is being shown in a window, it can not be shown in a different window.

Even though each data set can be shown in only one window, there is no limit on how many data sets can be shown in separate windows. That means you can have one window showing "USJudgeRatings" and another window showing "USArrests", and so on. However, to create a view of the "USArrests" data, the user must first change the active data set. This operation was discussed in Section 4.

One last note about viewing the data: the data can not be changed through the view data windows.

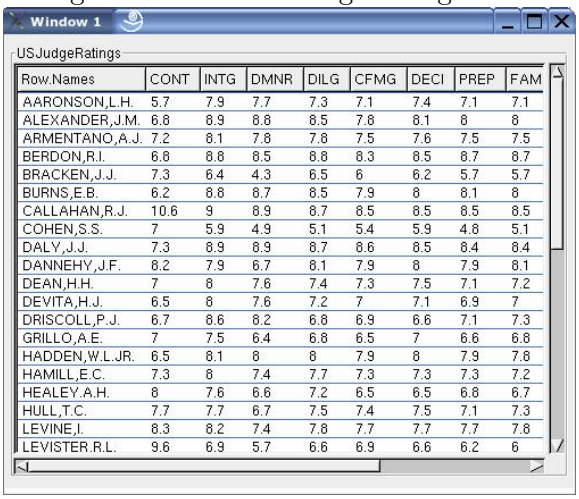
5.1 Viewing Data Through the GUI

To view the data, highlight the Display menu and select the View Data menu item. Alternatively, typing Ctrl-V activates the View Data menu item. The control window now appears as shown in Figure 5 after selecting the View Data menu item.

Now click the Show USJudgeRatings button. Another window appears that shows the data frame data. The new window showing the "USJudgeRatings" data frame is shown in Figure 6.

Figure 5: View the Data

Figure 6: View USJudgeRatings Data



Row.Names	CONT	INTG	DMNR	DILG	CFMG	DECI	PREP	FAM
AARONSON,L.H.	5.7	7.9	7.7	7.3	7.1	7.4	7.1	7.1
ALEXANDER,J.M.	6.8	8.9	8.8	8.5	7.8	8.1	8	8
ARMENTANO,A.J.	7.2	8.1	7.8	7.8	7.5	7.6	7.5	7.5
BERDON,R.I.	6.8	8.8	8.5	8.8	8.3	8.5	8.7	8.7
BRACKEN,J.J.	7.3	6.4	4.3	6.5	6	6.2	5.7	5.7
BURNS,E.B.	6.2	8.8	8.7	8.5	7.9	8	8.1	8
CALLAHAN,R.J.	10.6	9	8.9	8.7	8.5	8.5	8.5	8.5
COHEN,S.S.	7	5.9	4.9	5.1	5.4	5.9	4.8	5.1
DALY,J.J.	7.3	8.9	8.9	8.7	8.6	8.5	8.4	8.4
DANNEHY,J.F.	8.2	7.9	6.7	8.1	7.9	8	7.9	8.1
DEAN,H.H.	7	8	7.6	7.4	7.3	7.5	7.1	7.2
DEVITA,H.J.	6.5	8	7.6	7.2	7	7.1	6.9	7
DRISCOLL,P.J.	6.7	8.6	8.2	6.8	6.9	6.6	7.1	7.3
GRILLO,A.E.	7	7.5	6.4	6.8	6.5	7	6.6	6.8
HADDEN,W.L.JR.	6.5	8.1	8	8	7.9	8	7.9	7.8
HAMILL,E.C.	7.3	8	7.4	7.7	7.3	7.3	7.3	7.2
HEALEY,A.H.	8	7.6	6.6	7.2	6.5	6.5	6.8	6.7
HULL,T.C.	7.7	7.7	6.7	7.5	7.4	7.5	7.1	7.3
LEVINE,I.	8.3	8.2	7.4	7.8	7.7	7.7	7.7	7.8
LEVISTER,R.L.	9.6	6.9	5.7	6.6	6.9	6.6	6.2	6

5.2 Viewing Data Through the Command Line

To create a spreadsheet of the active MVC's data set through the command line, call the function, `createSpreadsheet`. Example code using this function is shown below. It takes no parameters since it only creates a spreadsheet view of the active MVC's data. Please see the man page for this function for more information.

```
> if (interactive()) {
+     createSpreadsheet()
+ }
```

6 Plotting Data

Once a data set is loaded, the user can create scatterplots of the data through the GUI or through the command line function, `createSPlot`. Remember that only the active MVC object's data can be plotted.

Unlike viewing the data where each data set can only appear in one spreadsheet window, users can create as many plots of each data set as they like. Each new scatterplot is presented in a new window.

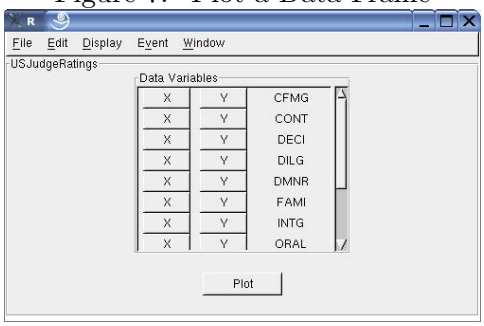
Recall from Section 3 that each MVC object contains one data set and all views of this data set. Thus, plots that are based on the same data are stored in the same MVC object and they are linked. This idea is discussed further in Section 7.

6.1 Plotting Data Through the GUI

To plot a data frame, highlight the Display menu and select the Plot Data menu item. Alternatively, the user can press Ctrl-P to activate the Plot Data menu item.

After activating the Plot Data menu item, the control window shows a frame called Data Variables, which contains toggle buttons to set the X and Y variables for the scatterplot using the column names from the active MVC object's data set. The control window now appears as shown in Figure 7.

Figure 7: Plot a Data Frame

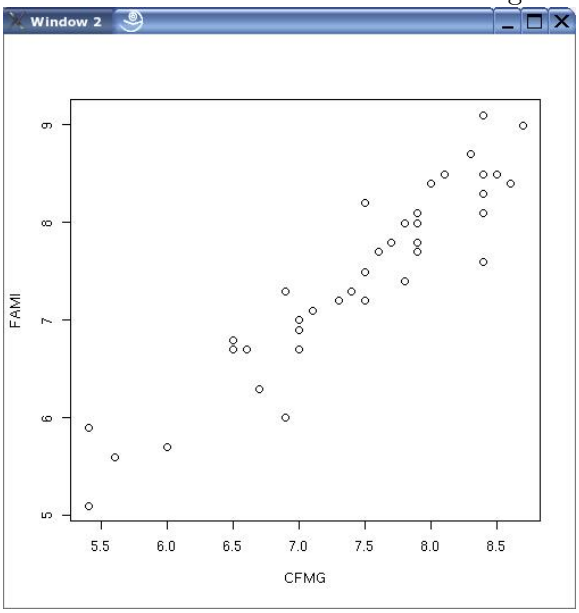


The variables available for plotting "USJudgeRatings" data are CFMG, CONT, DECI, DILG, DMNR, FAMI, INTG, ORAL, PHYS, PREP, RTEN, and WRIT. To learn more about these variables, the user can type help("USJudgeRatings") at the R prompt. As an example, click the X button for CFMG and click the Y button for FAMI, and then click the Plot button. A scatterplot of FAMI vs. CFMG appears in a new window, as shown in Figure 8.

Only one X button and Y button can be highlighted at a time because only two-way scatterplots can be made at this time.

To create another plot of "USJudgeRatings" data, click the X button for CONT and the Y

Figure 8: Plot FAMI vs. CFMG for USJudgeRatings



button for ORAL, and then click the Plot button. Now there are two plots of "USJudgeRatings" data. One of ORAL vs. CONT and one of FAMI vs. CFMG.

6.2 Plotting Data Through the Command Line

To create a scatterplot of the active MVC object's data set through the command line, call the function, `createSPlot`. The example code below creates a scatterplot with CFMG as the X variable and FAMI as the y variable. CFMG and FAMI are the names of columns from the "USJudgeRatings" data. Please see the man page for this function for more information.

```
> if (interactive()) {
+     createSPlot(varx = "CFMG", vary = "FAMI")
+ }
```

7 Setting Callback Functions to Events

Having interactive views was one of the goals discussed in Section 1.1. Creating interactive applications requires three things: some user action or input, which will be referred to as an event; a response to that action, which is executed by a callback function; and a method that connects the event to the callback function, which will be referred to as the signal handler. These three steps are ordered as follows: an event occurs, the event causes a signal to be emitted that is caught by the signal handler, and then the signal handler calls the callback

function, which results in the response to the user action. A flowchart of these steps in shown below in Figure 9.

Figure 9: Flow Chart of the Response to an Event

In an example given in Section 1.1, a point was colored red when the user clicked on that point in the view. In this example of interactivity, the event is the mouse button click and the response, which is caused by a callback function, is to color a point red. The signal handler, which is the method that connects the event to the callback function, is not noticed by the user.

Now in the *iSPlot* package, another goal is to have a flexible design. Thus, we did not want the response to an event to be fixed. We wanted to allow the user to decide what the response to an event should be. For example, the user may initially want a point to be colored when a mouse click occurs over that point and later the user may decide that a mouse click event over a point should cause the point to be hidden. Recall that the response to an event is determined by the callback function. Thus, for flexibility in the *iSPlot* package, the user can set the callback function for certain events. This lets the user decide what the response to an event is.

Currently, the four events that the user can set callback functions to are mouse over, left button click, middle button click, and right button click. This means that the user can decide what happens in response to any of these four events. Each event is discussed in more detail in the following sections.

One important thing to notice is that these callback functions can only be set for views of the active MVC object. Thus, when setting the callback function to an event through either the GUI or through command line functions, these responses are only set for views of the active MVC. However, if views of different MVCs have been created and callback functions were previously set for these views, then when the user interacts with these views of the non-active MVC, the previously set callback function is called. Thus, even views in non-active MVCs are interactive.

An example helps explain these concepts. Suppose a user had two loaded MVC objects, one was called "USArrests" and one was called "USJudgeRatings". Next suppose that the active MVC object is "USArrests", two scatterplots of this model are created, and the callback function for the left button click event is to color a point blue. Then the user sets the active MVC to be "USJudgeRatings", creates two new scatterplots of this model (so there are now four scatterplots: two of "USArrests" and two of "USJudgeRatings"), and sets the callback function for the left button click event to highlight a point. If the user now clicks the left button over a point in one of the views of "USArrests", then this point is colored blue (even though this view is not depicting the active MVC). If the user clicks the left button over a point in one of the

views of "USJudgeRatings", then this point is highlighted (recall this is a view of the active MVC).

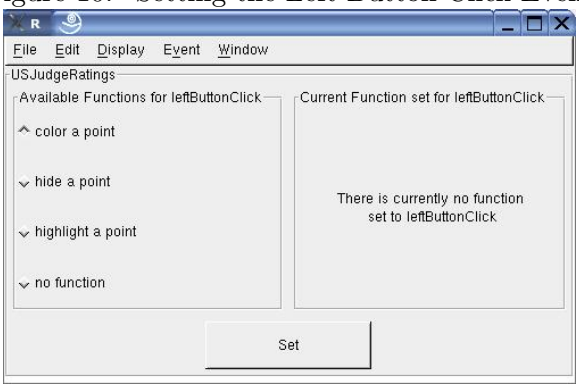
7.1 Setting the Left Button Click

Currently, the user has four options for the callback function for a left button click event. These four options are color a point, hide a point, highlight a point, or no action. Thus, when the user clicks the left button with the cursor over a point in a scatterplot, the point can either be colored, hidden, highlighted, or nothing will happen. The default is for nothing to happen.

Using the GUI

To see the callback functions that can be set to the left button click event, highlight the Event menu and select the Set Left Button Click menu item. Alternatively, the user can press Ctrl-B to activate the Set Left Button Click menu item. The control window now looks as shown in Figure 10.

Figure 10: Setting the Left Button Click Event



Using the Command Line

To see the descriptions of the callback functions that can be set to the left button click event, call the function, `getDescForEvent`, with a parameter of "leftButtonClick". This function returns the descriptions of the potential callback functions (these are the values you use as the second parameter in the call to `setCallFunc`, which is discussed in the following paragraphs).

```
> if (interactive()) {
+     getDescForEvent("leftButtonClick")
+ }
```

7.1.1 Coloring a Point

We will look at each callback function one at a time. First, let's set the left button click event to cause a point to be colored.

Using the GUI

To do this using the GUI, choose the "color a point" radio button and click the Set button (after choosing the Set Left Button Click menu item under the Event menu). A new window appears that has a color browser to allow the user to choose a color that will be used to color the points. Pick a color on the color wheel and click the Ok button. The window with the color wheel is shown in Figure 11.

Figure 11: Color Browser

Using the Command Line

To set the left button click event to color a point using the command line, call the function, `setCallFunc`, with "leftButtonClick" as the first parameter and "color a point" as the second parameter. The default value for the color is black. If you would like a different color, then call the function, `setColor`. To see the current color, call the function, `getColor`.

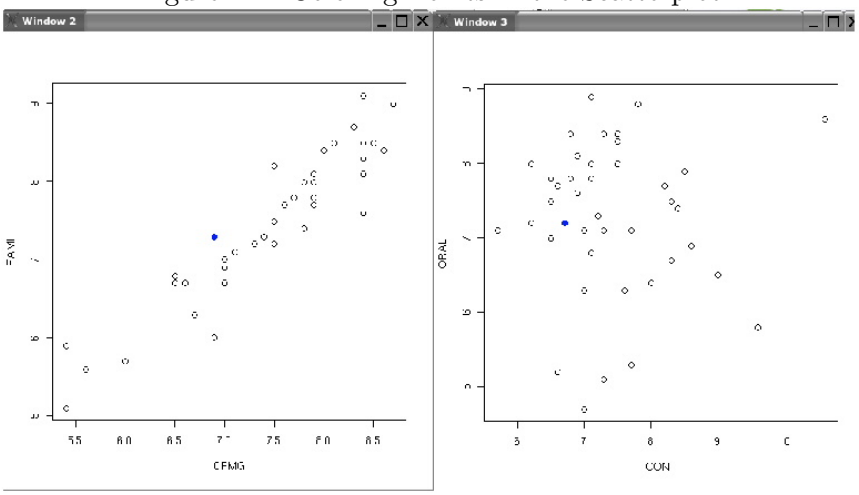
```
> if (interactive()) {
+     setCallFunc("leftButtonClick", "color a point")
+     setColor("blue")
+ }
```

Coloring a Point

Now click with the left button on a point in the scatterplot of FAMI vs. CFMG. Note that this plot was created in Section 6.1. The point that was just clicked will be filled in with the new color (I have used blue). Also, a point on the scatterplot of ORAL vs. CONT will also be filled in with the new color because these two points are from the same row of the data frame. This can be seen by viewing the "USJudgeRatings" data in a spreadsheet as was shown in Section 5. If the spreadsheet window for "USJudgeRatings" is open, the user will see that one row is selected. This selected row corresponds with the point that was just clicked with the

left button. Note that selecting a row in a spreadsheet view is considered the same as clicking a point with the left button over a scatterplot. Please see Section 7.1.5 for more information about selecting a row in a spreadsheet.

Figure 12: Coloring Points in the Scatterplot



This behavior shows how the scatterplots are linked and is shown in Figure 12. Even though a point on the FAMI vs. CFMG plot was clicked, all plots that are based on the "USJudgeRatings" data frame are updated. All points that correspond to that row in the data frame now have the new color. To keep the plots synchronized, some plotting information is stored with the data.

Now, if the user clicks on a point in the scatterplot of ORAL vs. CONT, both plots will show two blue points each. To choose a different color using the GUI, the user needs to again choose the "color a point" radio button and click the Set button, which will cause the window with the color wheel to reappear. To choose a different color using the command line, call the function, `setColor`.

7.1.2 Hiding a Point

Next, let's set the left button click event to cause a point to be hidden.

Using the GUI

To do this choose the "hide a point" radio button and click the Set button on the control window (after choosing the Set Left Button Click menu item under the Event menu).

Using the Command Line

To set the left button click event to hide a point using the command line, call the function, `setCallFunc`, with "leftButtonClick" as the first parameter and "hide a point" as the second parameter.

Hiding a Point

Now click with the left button on a point in the scatterplot of FAMI vs. CFMG. The point that was just clicked will disappear. Also, a point on the scatterplot of ORAL vs. CONT will also disappear because these two points are from the same row of the data frame. If a spreadsheet of the "USJudgeRatings" data is open, then the row corresponding to the point that was just hidden will be selected. To make the point reappear, unselect that row in the spreadsheet. The hidden point will now reappear on the scatterplots. Thus, hiding a point is a property that can be toggled on and off. The user can also click with the left button over the spot where the point used to be and the point will reappear.

7.1.3 Highlighting a Point

Next, let's set the left button click event to cause a point to be highlighted.

Using the GUI

To do this choose the "highlight a point" radio button and click the Set button on the control window (after choosing the Set Left Button Click menu item under the Event menu).

Using the Command Line

To set the left button click event to hide a point using the command line, call the function, `setCallFunc`, with "leftButtonClick" as the first parameter and "highlight a point" as the second parameter.

Highlighting a Point

Now click with the left button on a point in the scatterplot of FAMI vs. CFMG. The point that was just clicked will be highlighted (i.e. a red circle will appear around that point on the scatterplot). Also, a point on the scatterplot of ORAL vs. CONT will also be highlighted because these two points are from the same row of the data frame. If a spreadsheet of the "USJudgeRatings" data is open, then the row corresponding to the point that was just highlighted will be selected. To turn off the highlighting, unselect that row in the spreadsheet or click with the left button again over that point. Now the highlighting will disappear. Thus, highlighting a point is also a property that can be toggled on and off.

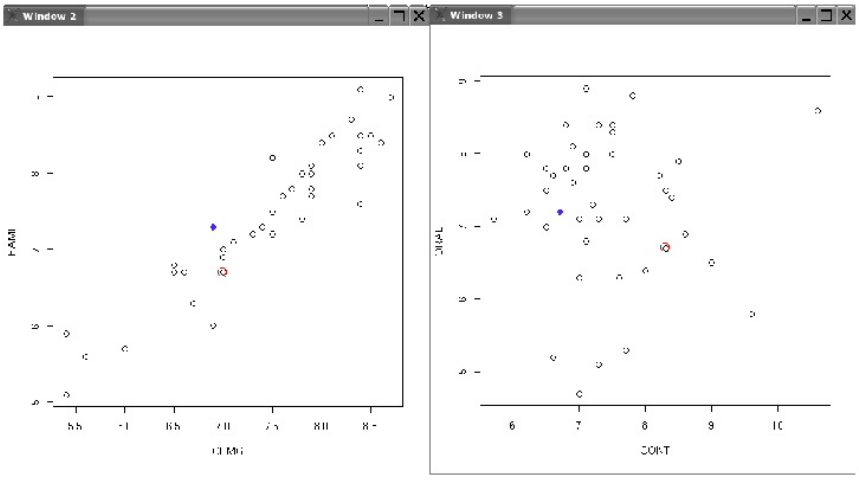
Figure 13 shows a point that has been highlighted on the scatterplots.

7.1.4 No Action

The last option and the default is to have no response to a left button click event. When the first view of the active data set is created, there is no response to a left button click event. If the user decides to color a point, hide a point, or highlight a point in response to a left button click, the user can always reset the response to be nothing when a left button click happens.

Notice also that when there is no response to a left button click this means that there is also

Figure 13: Highlighting Points in the Scatterplot



no response to selecting or unselecting a row in the spreadsheet view.

Using the GUI

To do this choose the "no action" radio button and click the Set button on the control window (after choosing the Set Left Button Click menu item under the Event menu).

Using the Command Line

To set the left button click event to have no response using the command line, call the function, `setCallFunc`, with "leftButtonClick" as the first parameter and "" as the second parameter.

7.1.5 Selecting a Row in a Spreadsheet

In the previous paragraphs, it has been mentioned that selecting a row in a spreadsheet view is equivalent to the left button click event over a scatterplot. Thus, if the user wants something to happen in response to selecting a row in a spreadsheet, then the user needs to set a callback function for the left button click event.

For coloring a point, selecting a row will color all points that correspond to that row's data while unselecting a row will not make any changes to the color of those points. Thus, coloring a point is a property that cannot be toggled on and off. In contrast, highlighting and hiding points are both properties that can be toggled. Thus, if the response to the left button click is to highlight a point, then selecting a row will highlight all points that correspond to that row's data and unselecting a row will un-highlight all points that correspond to that row's data. Similarly, if the response to the left button click is to hide a point, then selecting a row will hide all points that correspond to that row's data and unselecting a row will make those points reappear.

7.2 Setting the Middle Button Click

As with the left button click event, the user currently has four options for the callback function for a middle button click event. These four options are color a point, hide a point, highlight a point, or no action. Thus, when the user clicks the middle button with the cursor over a point, the point can either be colored, hidden, highlighted, or nothing will happen. The default is for nothing to happen. These callback functions have been described in Subsection 7.1 so please reference that section for more information on the callback functions.

The major difference between the left button click event and the middle button click event is the response to selecting or unselecting a row in a spreadsheet view. For the middle button click event, when a point is colored, hidden, or highlighted, the corresponding row in the spreadsheet view will be selected. However, if a user directly selects a row in the spreadsheet, the callback function for a middle button click event will not occur. A user selecting a row in a spreadsheet always causes the left button click event to occur. When the user colors, highlights, or hides a point using the middle button click event, the corresponding row in the spreadsheet is selected only to keep the views synchronized. Similarly, unselecting a row will not cause the callback function for a middle button click event to be called. However, if a user removes the highlighting or hiding of a point through the middle button click event over a scatterplot, the corresponding row on the spreadsheet will be unselected.

Using the GUI

To find out what callback functions can be set to the middle button click event, highlight the Event menu and select the Set Middle Button Click menu item. Alternatively, the user can press Ctrl-M to activate the Set Middle Button Click menu item.

Using the Command Line

To see the callback functions that can be set to the middle button click event, call the function, `getDescForEvent`, with a parameter of "middleButtonClick". This function returns the descriptions of the potential callback functions (these are the values you use as the second parameter in the call to `setCallFunc`, which was discussed above).

7.3 Setting the Right Button Click

As with the left and middle button click events, the user currently has four options for the callback function for a right button click event. These four options are color a point, hide a point, highlight a point, or no action. Thus, when the user clicks the right button with the cursor over a point, the point can either be colored, hidden, highlighted, or nothing will happen. The default is for nothing to happen. These callback functions have been described in Section 7.1 so please reference that section for more information on the callback functions.

The right button click event is identical to the middle button click event in how it responds to selecting and unselecting a row in a spreadsheet. Please see Section 7.2 to get more information about how a spreadsheet view is kept synchronized with scatterplot views when the user is

changing the appearance of points using the right button click event.

Using the GUI

To find out what callback functions can be set to the right button click event, highlight the Event menu and select the Set Right Button Click menu item. Alternatively, the user can press Ctrl-I to activate the Set Right Button Click menu item.

Using the Command Line

To see the callback functions that can be set to the right button click event, call the function, `getDescForEvent`, with a parameter of "rightButtonClick". This function returns the descriptions of the potential callback functions (these are the values you use as the second parameter in the call to `setCallFunc`, which was discussed in Section 7.1).

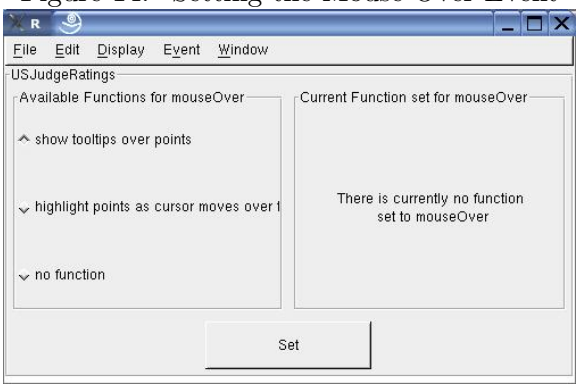
7.4 Setting the Mouse Over Event

Currently, the user has three options for the callback function for a mouse over event. These three options are show tooltips over points, highlight points as cursor moves over them, or no action. Thus, when the user moves the cursor over a point, tooltips can appear over the point, the point can be highlighted, or nothing will happen. The default is for nothing to happen.

Using the GUI

To find out what callback functions can be set to the mouse over event, highlight the Event menu and select the Set Mouse Over menu item. Alternatively, the user can press Ctrl-S to activate the Set Mouse Over menu item. The control window now looks as shown in Figure 14.

Figure 14: Setting the Mouse Over Event



Using the Command Line

To see the callback functions that can be set to the mouse over event, call the function, `getDescForEvent`, with a parameter of "mouseOver". This function returns the descriptions

of the potential callback functions (these are the values you use as the second parameter in the call to `setCallFunc`, which is discussed below).

```
> if (interactive()) {
+     getDescForEvent("mouseOver")
+ }
```

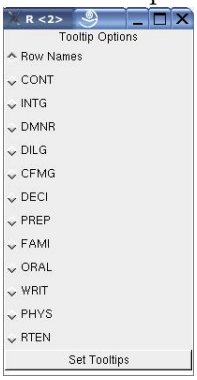
7.4.1 Showing Tooltips over Points

We will look at each callback function one at a time. First, let's set the mouse over event to cause tooltips to appear over a point.

Using the GUI

To do this choose the "show tooltips over points" radio button and click the Set button (after choosing the Set Mouse Over menu item from the Event menu). A new window appears that allows the user to choose what tooltips will appear over the points. The user can choose either any of the variables (column names) from the data frame or the row names as the tooltips. The window that appears for the "USJudgeRatings" data frame is shown in Figure 15.

Figure 15: Tooltip Options



For now, choose the "Row Names" radio button and click the Set Tooltips button.

Using the Command Line

To set the mouse over event to show tooltips using the command line, call the function, `setCallFunc`, with "mouseOver" as the first parameter and "show tooltips over points" as the second parameter. The default value for the tooltips is to show the row names of the data frame. If you would like to use different values for the tooltips, then call the function, `setTooltips`. To see the current values for the tooltips, call the function, `getTooltips`.

```
> if (interactive()) {
+     setCallFunc("mouseOver", "show tooltips over points")
```

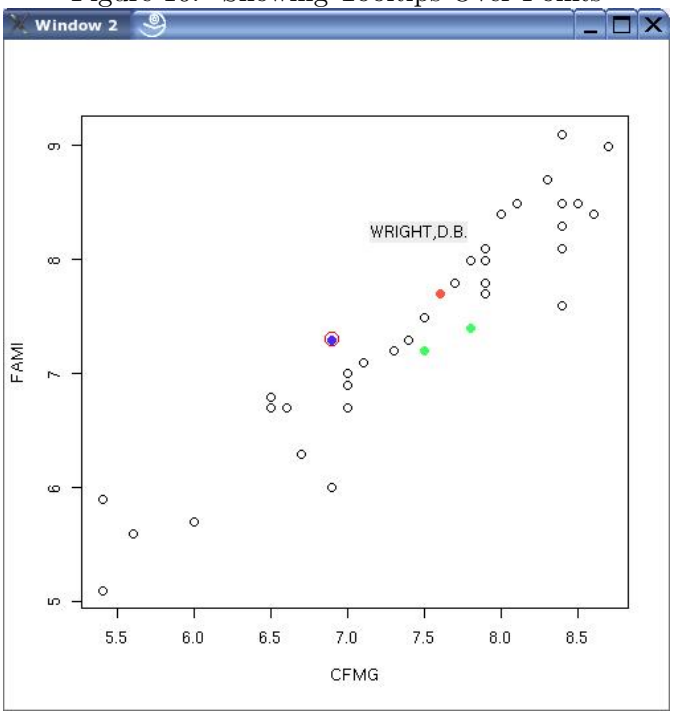
```
+     setTooltips("rowNames")
+ }
```

Showing Tooltips

Now go to one of the scatterplots and move the cursor over the points (make sure the device is active by clicking on the plot first). A tooltip will appear with the row name of that point in the tooltip window. Note that this is the one callback function where the views are not linked. Tooltips appear over the points in the active plot, but nothing happens in the other plots. To see linked plots with the mouse over event, please refer to Section 7.4.2.

Figure 16 shows a tooltip (for row "Wright,D.B.") over a point on the scatterplot.

Figure 16: Showing Tooltips Over Points



7.4.2 Highlighting Points as the Cursor Moves Over Them

Next, let's set the mouse over event to cause a point to be highlighted.

Using the GUI

To do this choose the "highlight points as cursor moves over them" radio button and click the Set button on the control window (after choosing the Set Mouse Over menu item from the Event menu).

Using the Command Line

To set the mouse over event to highlight a point using the command line, call the function, `setCallFunc`, with "mouseOver" as the first parameter and "highlight points as cursor moves over them" as the second parameter.

Highlighting Points

Now move the cursor over the points in a scatterplot. As the cursor lingers over a point, that point will be highlighted and then when the cursor moves off the point, that point will be un-highlighted. Unlike the tooltips option for the mouse over event, all plots will be linked so any corresponding points (i.e. points that show data from the same row in the data frame) in other plots will also be highlighted. Also if the spreadsheet view is open, when a point is highlighted, then the corresponding row in the spreadsheet will be selected and when the point is un-highlighted, the corresponding row in the spreadsheet will be unselected.

7.4.3 No Action

The last option and the default is to have no response to a mouse over event. When the first view of the active data set is created, no response will occur when the cursor moves over the plot. If the user decides to show tooltips or highlight points in response to a mouse over event, the user can always reset the response to nothing for the mouse over event.

Using the GUI

To do this choose the "no action" radio button and click the Set button on the control window (after choosing the Set Mouse Over menu item under the Event menu).

Using the Command Line

To set the mouse over event to have no response using the command line, call the function, `setCallFunc`, with "mouseOver" as the first parameter and "" as the second parameter.

8 Editing Data

8.1 Deleting a MVC Object

If the user decides that he/she is no longer interested in studying one of the MVC objects that has been loaded, then the user has the option of deleting this MVC object. Only MVC objects that are not active can be deleted. Thus, if the user wants to delete a MVC object that is currently active, the user needs to set another MVC object to be active first.

Note that if only one MVC object is loaded, then no MVC objects can be deleted. When there is only one MVC object that means the one MVC object must be the active MVC object and the active MVC object cannot be deleted.

Deleting a MVC object removes all aspects of the MVC object. In other words, the data are removed, all views of this data set are closed, and any information pertaining to this MVC object is deleted. This is not reversible. Note that if the user wants to delete all MVC objects, then the user should quit the program. See Section 9 for more information on quitting.

Using the GUI

Select the Delete MVC menu item under the Edit menu or press Ctrl-E. If there is more than one MVC object loaded, then a new window will appear with the names of the non-active MVC objects. The user can then choose one of the non-active MVC objects to delete by selecting the radio button next to the MVC object that they would like to delete and clicking the Delete button.

Using the Command Line

To delete a non-active MVC object from the command line, call the function, `deleteMVC`. The only parameter that is required by this function is the name of the MVC object to delete. If the user is not aware of the names of the loaded MVC objects, the user can call `getModelNames` to see the names of the loaded MVC objects. If the user wants to see the active MVC object's name, the user can call `getActiveMVC`. It is this active MVC object that cannot be deleted.

```
> if (interactive()) {
+     getModelNames()
+     getActiveMVC()
+     deleteMVC("USArrests")
+     getModelNames()
+ }
```

8.2 Find

Now suppose you are interested in finding the points that correspond to a particularly row in a data set. To perform Find, the left button click event must be set to some callback function so that when the row is found, something happens to that row and the points that represent that row. For this example, the left button click event is set to "highlight a point". Other responses to the left button click event are available. Please see Section 7.1 for more information.

Find can be performed either by looking for a row name or by looking for a data element that is in the data set.

Using the GUI

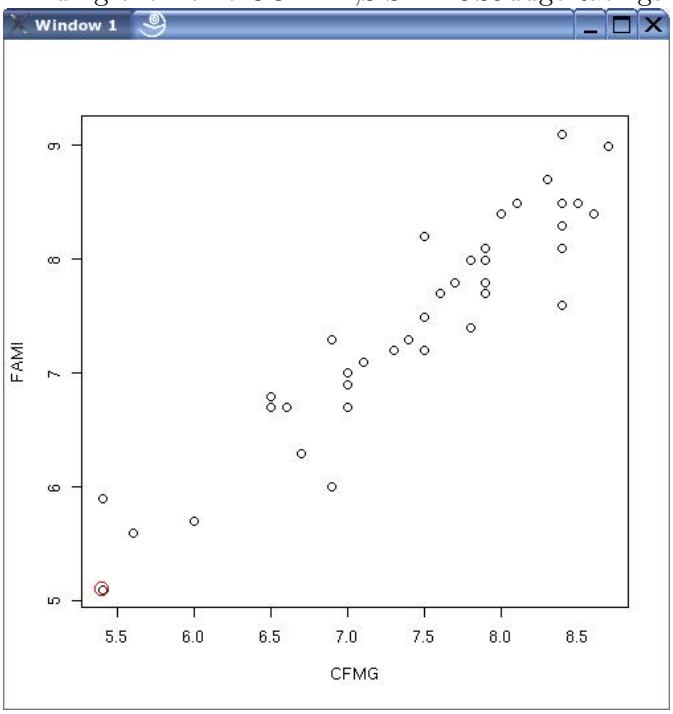
Select the Find menu item under the Edit menu or press Ctrl-F. If there is at least one open view of the active data set and if the left button click event is set to some callback function (i.e. not the "no action" option), then a new window that looks like Figure 17 will appear (except the text box will be empty).

In the window shown in Figure 17 set the 'Find based on' drop down box to Row, enter the

Figure 17: Finding Points in the Scatterplot

row name you want to find, and press the Ok button. For example, in the text box enter "COHEN,S.S.", except without the quotations. Make sure you do not enter any spaces and that you capitalize all of the letters, and then press the Ok button. The active scatterplot will have the point corresponding to COHEN,S.S. highlighted, as shown in Figure 18. Any other plots that depend on the data from "USJudgeRatings" will also show points from that row as being highlighted.

Figure 18: Finding the Point COHEN,S.S. in USJudgeRatings Scatterplot



Now let's find a row based on a data element. This time set the 'Find based on' drop down box to Data Element, enter a value from the data set (for example, enter the number 8.8) and press the Ok button. Now the first row that contains this data element will be highlighted because the callback function for the left button click event is set to highlight a point. For the "USJudgeRatings" data set, the first row containing the value, 8.8, is the second row, ALEXANDER,J.M. This point is now highlighted on the plots.

Using the Command Line

Recall that for this example the left button click event has been set to highlight a point. To perform find through the command line, call the function, `findElement`. The `findElement` function has two parameters: value and matchOn, where value is the value to find and matchOn is either "Row" if the value is a row name or "Data" if the value is a data element.

First, to find a row name, call the function, `findElement`, with "COHEN,S.S." as the first parameter and "Row" as the second parameter. Now the active scatterplot has the point corresponding to COHEN,S.S. highlighted. Any other plots that depend on the data from "USJudgeRatings" also show points from that row as being highlighted.

```
> if (interactive()) {
+     setCallFunc("leftButtonClick", "highlight a point")
+     findElement("COHEN,S.S", matchOn = "Row")
+ }
```

Next let's find a data element. Again call the function, `findElement`, with "8.8" as the first parameter and "Data" as the second parameter. Now the first row that contains this data element is highlighted because the callback function for the left button click event is set to highlight a point. For the "USJudgeRatings" data set, the first row containing the value, 8.8, is the second row, ALEXANDER,J.M. This point will now be highlighted on the plots.

```
> if (interactive()) {
+     findElement("8.8", matchOn = "Data")
+ }
```

8.3 Replot

Replotting the data is useful if you have made many changes to the data set and you want to clean up the views by replotting them. Replotting does not make any changes to the data set like Reset does, which is described in Section 8.4. Replotting only refreshes the plots and remove any selected rows on the spreadsheet view.

If you have multiple views drawn that depend on different data sets, only views that are part of the active MVC object are replotted. All views that reflect the active data set are redrawn so that includes both scatterplot and spreadsheet views.

Using the GUI

To replot views of your active data, choose the Replot menu item under the Edit menu or press Ctrl-R. A message box will appear asking if the user wants to replot the data, as shown in Figure 19. Click the Yes button and all views based on the active data set are replotted.

Using the Command Line

To replot views of the active data, call the function, `replotViews`. For more information on this function, please see its man page.

Figure 19: Replotting the Data


```
> if (interactive()) {
+     replotViews()
+ }
```

8.4 Reset

If you want to reset the data so it appears as it did when it was first loaded, then you want to use reset. This removes any changes you have made to the data, such as setting the color or highlighting a point. Resetting cannot be undone.

If you have multiple data sets loaded, reset only changes the active data set.

Using the GUI

Select the Reset menu item under the Edit menu or press Ctrl-T, which results in a message box appearing to ask if the user really wants to reset the data, as shown in Figure 20. Click the Yes button and all plots based on "USJudgeRatings", which is the active MVC, are replotted after the data set has been returned to its original state.

Figure 20: Resetting the Data

Using the Command Line

To reset the data using the command line, call the function, `resetData`. Recall that resetting the data is permanent so be sure you really want to call this function. For more information on this function, please see its man page.

```
> if (interactive()) {
+     resetData()
+ }
```

9 Quitting

Using the GUI

When the user is ready to quit, select the Quit menu item under the File menu or press Ctrl-Q. A message box will appear that asks if the user really wants to quit. By clicking the Yes button, all windows such as scatterplots and spreadsheets as well as the control window are closed. Also all data sets that have been loaded are removed.

Figure 21: Quitting Interactive Scatterplots

Using the Command Line

To quit using the command line, call the function, `quitiSPlot`. The `quitiSPlot` function closes all views and removes all loaded data. This action is not reversible. Please see the man page for `quitiSPlot` for more information.

10 Command Line Functions

Most of the command line functions have been described in the previous sections according to their functionality, but a quick review is given here. All of the operations that can be performed through the GUI can also be performed using the R command line. To find more information about any of the command line functions, please see their man pages.

The load data functions are `loadData` and `loadFile`. Please see Section 3.2 for more information.

To see what the active MVC object is, the user can call `getActiveMVC` and to set the active MVC object, the user can call `setActiveMVC`. Please see Section 4.2 for more information.

To create views of the active data set, there are two functions: `createSpreadsheet`, which creates a spreadsheet, and `createSPlot`, which creates a scatterplot. Please see Sections 5.2 and 6.2 for more information.

There are several functions to help a user link a callback function to an event. The `getEvents` function returns the names of the events that can have callback functions linked to them (currently the events are leftButtonClick, middleButtonClick, rightButtonClick, and mouseOver). The `getDescForEvent` function returns descriptions of all possible callback functions for an event (note that this function does not return the "no action" option, though that is an option). Finally, the `setCallFunc` actually links a callback function to an event. The `setCallFunc` func-

tion provides the same functionality as all of the menu items under the Event menu. Please see Section 7 for more information.

Note that two callback functions require some extra information. These callback functions are color a point and show tooltips. For coloring a point, the users must say which color should be used and for showing tooltips, users must say what the tooltips should show (either row names or values from a column in the data). To allow users to input this information, there are two command line functions: `setColor` and `setTooltips`. Also, if users want to know what these values are set to, they can call `getColor` or `getTooltips`. For more information about coloring a point, please see Section 7.1.1 and for more information about tooltips, please see Section 7.4.1.

To delete a MVC object, call the function, `deleteMVC`. Please see Section 8.1 for more information.

To find an element, there is the function, `findElement`. Please see Section 8.2 for more information.

To replot views of the active data set, call the function, `replotViews`. Please see Section 8.3 for more information.

To reset the data to its original state, call the function, `resetData`. Please see Section 8.4 for more information.

To quit the program, call the function, `quitiSPlot`. Please see Section 9 for more information.

To see the names of the loaded data sets (and thus, the names of the MVC objects), the user can call the function, `getModelNames`. This returns all the names of the currently loaded MVC objects.

To see all of the information about the loaded MVC objects (i.e. the model information, the view information, and the controller information), the user can call the function, `getMVCList`. This function returns a list of all the MVC objects that are currently loaded.

To return the MVC object that corresponds with a view, the user can call the function, `getMVCFromWinNum`. This function is available for developers that want to create new callback functions for an event. In Section 11.2.2, `getMVCFromWinNum` is called within the new callback function, `changePch`. The `getMVCFromWinNum` function is needed by new callback functions because a user may interact with a view that is not from the active MVC object. Please see 7 for more information on how a user can interact with views that are not from the active MVC.

In Section 11 a few more command line functions, which let the user extend the *iSPlot* package's functionality, are discussed. These functions include `addCBFunction` to add a callback function, `addMenuItem` to add a new menu item to the main menu, and `addSubMenuItem` to add a new sub menu item.

The `addToKeyVals` function adds an accelerator key to all Gtk+ windows that have been created through the *iSPlot* package. Thus, if you want a key press to be available on all windows, you would call the `addToKeyVals` function with the new key value and the function

that will be called in response to the key press event.

The `getData` function returns the model data from a loaded MVC object.

The `resetWinSize` function resets the control window to its original size.

A few functions are not meant to be called by the user. These include `loadModel`, `setToggleX`, and `setToggleY`.

11 Extensibility in the *iSPlot* Package

Having an extensible design was one of the goals for the *iSPlot* package discussed in Section 1.1. Thus, there are several places where users can make additions based on the needs of their data. Some of the additions can be done directly at the R command line, such as adding new menus and menu items to the GUI, and adding potential callback functions for an event, while other additions are more complicated and require making additions to the code in the *iSPlot* and/or the *MVCClass* package. Additions that require adding code in a package include creating new model or view classes and adding new events that views will respond to. All of these possible ways of extending the functionality of the *iSPlot* package are discussed in the following sections. Note that all extensions of the *iSPlot* package only require the user to be able to program in R.

11.1 Adding New Model or View Classes

Currently in the *iSPlot* package, the user has access to a data frame (or matrix) model and can create two types of views of this model. These two views are the spreadsheet view and the scatterplot view. These model and view classes are defined in the *MVCClass* package. If the user wants additional model and view classes to be available in the *iSPlot* package, then the user should start by reading the *MVCClass* Vignette to see which model and view classes are currently defined in that package. Another R package called *iSNetwork* has been created that has a greater variety of model and view classes so users are encouraged to look at that package to see if it meets their needs. The *iSNetwork* package also uses the class and generic function definitions made in the *MVCClass* package as well as classes defined in the *BioMVCClass* package.

11.1.1 Adding a New Model Class

To add a new model class that the *iSPlot* package could use, the user needs to add a new class definition and potentially generic function definitions that extend the definitions made in the *MVCClass* package. After the new model class is defined, the user needs to make several additions to the code in the *iSPlot* package. These additions to the *iSPlot* package include creating definitions for the `initialize` and the `updateModel` methods, and changing several functions in *iSPlot* that are only expecting data of type data frame or matrix. It is strongly

suggested that if the user wants more model classes, the user should use the *iSNetwork* package, which gives the user access to a wider range of model types.

11.1.2 Adding a New View Class

Similarly to adding a new model class, adding a new view class requires the user to add a new class definition and potentially generic function definitions that extend the definitions made in the *MVCClass* package. For example, if the user wants to create a view class for a histogram, then the user needs to create a class, which could be called `hPlotView`, and this new class will inherit from the `plotView` class. See the *MVCClass* Vignette for more information on these view classes.

After the new view class is defined, the user needs to make several additions to the code in the *iSPlot* package. The user needs to create definitions for the `initialize`, the `updateView`, and the `redrawView` methods. Depending on the events this new view class should respond to, the user may also need to create definitions for methods that respond to events, such as a `clickEvent` or a `motionEvent` method. The user also needs to add a menu item to the Display menu. The new Display menu item will create this new view when the menu item is activated. The necessary steps for adding a histogram view to the *iSPlot* package are detailed in the use case below.

Use Case for Adding a Histogram

Primary Actor: Programmer
Scope: *iSPlot* and *MVCClass* Package
Level: Summary
Stakeholders and Interests: Users of *iSPlot*
Precondition: Programmer has access to the *iSPlot* and *MVCClass* code
Success Guarantee: A new histogram view, which is an interactive and linked view, is available to users.
Main Success Scenario:

1. Programmer adds a new menu item to the Display menu on the control window in the *iSPlot* package. This menu item allows a user to create a histogram view.
2. Programmer creates a function that is called when the menu item (created in step one) is activated. This function allows the user to choose which variable to plot and then creates the histogram with the appropriate data from the active model.
3. Programmer creates a new view class that represents a histogram view and that inherits from one or more classes in the *MVCClass* package. This new view class needs to store information, such as the column and rows that were used to create the histogram.
4. Programmer must define methods for the new histogram view class in the *iSPlot* package and these methods must include an `initialize`, an `updateView` and a `redrawView` method. The `initialize` method properly sets up the view, creates the view instance, and makes sure that the view can respond to certain events, such as key press, delete,

focus in, and button clicks (see the next step). The `updateView` method updates the view when the model has changed. This method takes two parameters: object, which is the view object, and vData, which is a list of the following four elements: row name, column name, old value and new value. This information passed in the parameters allows the programmer to update the view so it is synchronized with the model. The `redrawView` method completely redraws the view. Thus, the `redrawView` method only has one parameter: object, which is the view object to redraw.

5. For the histogram view to be interactive, the view needs to respond to the following events: key press event, delete event, focus in event, the mouse button press event, and possibly the motion notify event. The key press event adds accelerators to the view, the delete event ensures that the view list is current, and the focus in event ensures that the device is active when the histogram has the focus. The mouse button press event allows the user to change plot information for the data (such as the color of a histogram bin) by interacting with the view. Similarly, the motion notify event, if the programmer wants to respond to it, would let the user interact with the view by moving the cursor over the view. Responding to these events is set in the `initialize` method for this new view class, which is defined by the programmer.

Extensions:

1. Other events besides those listed in step four may need to be noticed depending on what interactivity the programmer wants to support.
2. A controller, such as a slider, can be added to the histogram view to control the histogram bin width.

11.2 Adding Events and Callback Functions

11.2.1 Adding a New Event

Currently, callback functions can be linked to four events in the *iSPlot* package. These events are the left button click event, the middle button click event, the right button click event, and the mouse over event. Other events are being responded to in the *iSPlot* package, but the user is currently not allowed to change the callback function linked to these events. These events include the delete event, the focus in event, and the key press event. The key press event is slightly different in that by adding a menu item to the GUI, a new key press event is added, and users can add key values to the key press event through the `addToKeyVals` function. More information on adding a new menu item is given in Section 11.3.

If users would like to add a new event (like brushing) to this list, they need to perform the following steps.

1. Add this event to the possEvents variable in the environment, mvcEnv, in the *iSPlot* package. This variable, possEvents, is used so that functions in the *iSPlot* package know which events they must potentially respond to.

2. Add a new variable to the controller of each MVC object. This step is done by adding a new variable to the `setControllerDefaults` function in the *iSPlot* package.
3. Add a new menu item to the Event menu. When this menu item is activated, it will call the `setCallFunEvent` function with the name of the event list.
4. Depending on the event, the user may need to add a new signal handler to catch this event in the `initialize` method of the views that respond to this event. Currently, the scatterplot view has a signal handler to catch the button click events and the mouse over event, while the spreadsheet view has a signal handler to catch the row selection events. The user also has to add a new method for this event. For example, the signal handler for the mouse over event on the scatterplot view calls the `motionEvent` method for the scatterplot view. A similar method would need to be defined for all views that respond to the new event. Note that by adding a new method, a new generic function must also be defined.

11.2.2 Adding a new callback function

To add a callback function for an event that is currently available, the user can call the function, `addCBFunction`, from the command line. The `addCBFunction` function can add potential callback functions for the following events: mouseOver, leftButtonClick, middleButtonClick, and rightButtonClick.

The `addCBFunction` function has four parameters, which are callFunction, shortName, preprocessingFun, and eventLists. The callFunction parameter is the callback function, the shortName parameter is a short description of what the callback function does, the preprocessingFun parameter is a list of functions that must be called before the callback function (or it can be NULL if there are no functions to call), and the eventLists parameter is a vector of event names that this callback function can be linked to.

The callback function that is used in the callFunction parameter must take only one argument and that argument must of be of type character to represent a row name from the data frame. This callback function must create and handle a `gUpdateDataMessage` so that the data is updated.

Not all callback functions need preprocessing functions so the preprocessingFun parameter can be set to NULL. If the callback function needs some information besides the row name (which is the only parameter a callback function has), then the callback function must have one or more preprocessing functions. For example, when the callback function is to color a point, there must be a preprocessing function that has users set the color they want to use.

Currently, the eventLists parameter can be a vector containing one or more of the following events: “mouseOver”, “leftButtonClick”, “middleButtonClick”, and “rightButtonClick”. These are the events in the *iSPlot* package that the user can change the response to.

The following code adds a callback function that lets the user change the plotting character of a point. The new callback function is first defined and here the new callback function is called,

changePch. After the callback function has been defined, then a call to the addCBFunction function is made. In the call to addCBFunction, the callback function is changePch, the description is "change the plotting character", the preprocessing functions are set to NULL, and the events this callback function can be linked to are any of the button click events ("leftButtonClick", "middleButtonClick", and "rightButtonClick"). In this callback function example, the new plotting character has been set to a triangle by hard coding the new pch value in the code: newPchValue<-2. Most likely a preprocessing function should instead be used to let the user decide what the new plotting character will be.

```
> if (interactive()) {
+     changePch <- function(rowName) {
+         curMVC <- getMVCFromWinNum()
+         virtualData <- virtualData(model(curMVC))
+         colName <- "pch"
+         colIndex <- match(colName, colnames(virtualData))
+         rowIndex <- match(rowName, rownames(virtualData))
+         newPchValue <- 2
+         mData <- as.list(newPchValue)
+         names(mData) <- rowName
+         dfMessage <- new("gUpdateDataMessage", type = colName,
+             mData = mData, dataName = modelName(model(curMVC)),
+             from = "")
+         handleMessage(dfMessage)
+     }
+     addCBFunction(changePch, "change the plotting character",
+         c(), c("leftButtonClick", "middleButtonClick", "rightButtonClick"))
+ }
```

Now this callback function can be linked to the left button click event, for example, by using the Set Left Button Click menu item under the Event menu on the GUI or by calling the function, setCallFunc. Once the changePch function has been linked to the left button click event, then the user can left click on a point in a scatterplot and the point will be redrawn as a triangle.

11.3 Adding Menu Items

If the user wants to add menu items to the main menu on the control window from the R command line, then the following two R functions let the user do that: addMenuItem and addSubMenuItem.

11.3.1 Adding Menu Items

The function `addMenuItem` adds a menu item to the menu bar on the control window. So the new menu item appears after Window on the menu bar (other menu items on the menu bar are File, Edit, Display, Event, and Window).

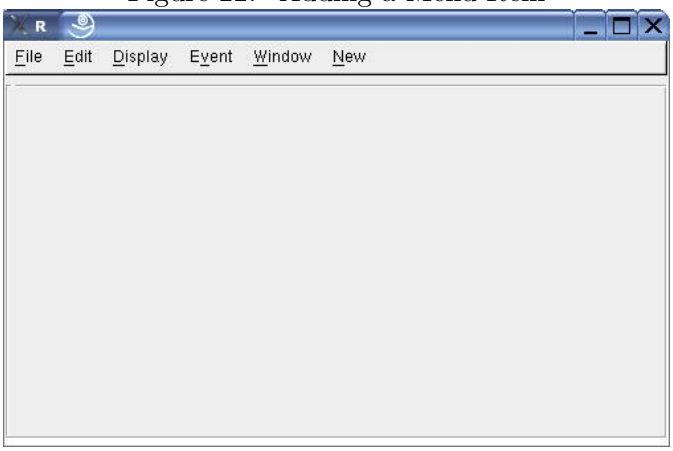
To add an accelerator to this menu item, place an underscore before the letter that you want to be the accelerator. In the following code example, the accelerator is 'N' because there is an underscore placed before the 'N' in New.

The default modifier type for adding a menu item (parameter modType) is the Alt button. So for the new menu item that the following code creates, the accelerator is Alt-N. See the man page for `addMenuItem` for descriptions of the function's parameters.

```
> if (interactive()) {
+     addMenuItem(menuName = "new", labelText = "_New")
+ }
```

After performing the above code, the control window looks as shown in Figure 22.

Figure 22: Adding a Menu Item



11.3.2 Adding Sub Menu Items

If instead the user wants to add a sub menu item to an already existing menu, then the user should call the function `addSubMenuItem`. Examples of sub menu items that are already in the main menu are Open Data, Open File, and Quit under the File menu.

Again, to add an accelerator to the sub menu item, place an underscore before the letter that you want to be the accelerator. In the following code example, the accelerator is 'N' because the underscore appears before the letter 'N' in Different.

The default modifier type for adding a sub menu item (parameter modType) is the Ctrl button. So for the new sub menu item that the following code creates, the accelerator is Ctrl-N. See the man page for `addSubMenuItem` for descriptions of the function's parameters.

Also, the user must give a character string for action (the third parameter) that corresponds to the name of a function that will be called when this sub menu item is activated. This function may have parameters. In the code below, action is set to "testfun", the name of the function that is defined in the following code chunk.

In the following example, the sub menu item, Different, is added to the menu item, New.

```
> if (interactive()) {
+     testfun <- function() {
+         w <- gtkWindow(show = FALSE)
+         lab <- gtkLabel("It works!")
+         w$Add(lab)
+         w$Show()
+     }
+     addSubMenuItem(menuName = "new", labelText = "Differe_nt  Ctrl+N",
+         action = "testfun")
+ }
```

If the user now chooses the Different menu item from the New menu, then a new window will open that says "It works!".