SOCIETY OF ACTUARIES

Article from:

CompAct

April 2009 – Issue 31

R Corner[i]

By Steve Craighead

Editor's note: R Corner[i] is a series by Steve Craighead introducing readers to the "R" language used for statistics and modeling of data. The first column was published in the October 2008 issue, and explains how to download and install the package, as well as providing a basic introduction to the language. Refer to each CompAct issue since then for additional articles in the series. The introductory article can be found on p. 24 of the October 2008 issue on the SOA Web site: *http://soa.org/library/newsletters/compact/2008/october/com-2008-iss29.pdf*

In the last article, I discussed how to get data into and out of the R platform, by either using comma delimited formatted files or using ODBC techniques to query data from Microsoft Access. In this article I will discuss how to use data once it is within R.

There are many different ways that one can store and access data within R. The primary ones that I want to discuss will be scalars, vectors, matrices and dataframes. More advanced structures are lists and objects that can be specifically designed for your applications, which I will not be able to address in this article.

Effectively one can assign a value to a variable by the use of the "<-" assignment symbol within R. For instance the command (following the ">" symbol)

a<-1

assigns the value of 1 to the variable "a". In this case the variable "a" is a scalar. To display the content of "a", all one needs to do is type the command

a

and R returns

[1] 1

Note how the [1] indicates the number of the element starting in the row displayed by "a". If you want to assign a series of values to a vector, you can do this several ways.

The most primitive is to assign a list by

a<- c(1,2,4,10,25,67,4)

Notice how the data is encapsulated within a "c(…)" structure. If you display "a", you get:

[1] 1 2 4 10 25 67 4

Warning: Don't use "c" as a variable, since the "c(…)" structure may produce very odd results, if used as a variable name.

Another way to get data within a variable uses the ":" symbol. Type

a <- 1:20

This stores the values 1 through 20 within "a". If you would like to count down instead of up, you would use

a<- 30:1

R would display "a" in this way:

a
[1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6
[26] 5 4 3 2 1

Notice how [26] is displayed in the second row. So the 26th element of vector "a" is "5".

To create lists of values that skip at different than unit intervals use the seq() command. For instance the commands

a<- seq(10,20,.5)

a

[1] 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0 16.5 17.0

[16] 17.5 18.0 18.5 19.0 19.5 20.0

demonstrates the method to count from 10 to 20 by 0.5.

To concatenate two vectors, you can use the "c()" structure again. For example, notice how

b<-c(1:5,a)

b

[1] 1.0 2.0 3.0 4.0 5.0 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5

[16] 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0

CONTINUED ON **PAGE 10**

Steve Craighead, ASA, MAAA, is an actuarial consultant at TowersPerrin in Atlanta, Ga.
He can be reached at steven.craighead@towersperrin.com.

combines the 1:5 vector with the "a" vector immediately above. Notice how the two types (1:5 vs. the vector "a") can be easily mixed.

To access various elements of a vector, you will use the "[]" structure. For instance

b[3:8]

[1] 3.0 4.0 5.0 10.0 10.5 11.0

displays the third through the eighth value in "b".
To drop a value, append a negative sign "-" to the numeric reference with the "[]" structure. For instance:

b[-26]

[1] 1.0 2.0 3.0 4.0 5.0 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5

[16] 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5

removed the last value "20.0" from "b". To determine the length of a vector use the "length()" function. So,

length(b)

[1] 26

Next let's look at some of the basic commands when using matrices.

MATRICES

To work with matrices you will use the matrix() command. For instance

d<-matrix(1:9,nrow=3,byrow=T)

> d

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 4 5 6

[3,] 7 8 9

Notice, how we used the vector "1:9" to load the matrix, we specified the number of rows with the "nrow" component and we had to use the "byrow=T" component to make sure that we loaded the matrix row by row. If this component is left out, the default is to load the matrix by columns. Examine further use of the matrix() function, by using the ?matrix command.

To access a specific element within the matrix "d", again you use the "[]" structure, but you need to insert a comma to distinguish the dimensions. For instance

d[1:2,3]

[1] 3 6

extracts the values from column 3 of rows 1 and 2 of the matrix "d".

Just like "length()" specifies the length of a vector "dim()" specifies the dimensions of a matrix.

dim(d)

[1] 3 3.

If you want to extract the second dimension of the results of "dim()", do this

dim(d)[2]

[1] 3.

To insert a new value within either a vector or a matrix, use the "[]" structure on the left-hand-side of an assignment statement. For instance, if you want to insert the value "200" into the middle of "d", you would do the following:

d[2,2]<-200

> d

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 4 200 6

[3,] 7 8 9.

If you would like to add a row (or column) to a matrix you can use rbind() (cbind()). For instance,
e<-rbind(d,c(3,4,5))

produces

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 4 200 6

[3,] 7 8 9

[4,] 3 4 5.

Other useful matrix functions and operators are "t()" for transpose, "%*%" for matrix products, and "diag()" to manipulate the matrix diagonals.

Next, let us look at dataframes.

DATAFRAMES

In the prior article we examined how to get data into and out of R and our emphasis was on storing the results in a dataframe. A dataframe has the basic attributes of a matrix except the dataframe columns don't have to be just numeric. The basic field types of dataframe can be numeric, alphanumeric or categorical. We will take as our example of a dataframe one of the existing datasets that is installed in the base R application. If you enter the "data()" command R will display a window of the various data sets in the package "datasets". If you would like to see all of the data sets available from all packages that you have installed in R, execute the following command:

data(package = .packages(all.available = TRUE))

To store a dataset in your current R application from the stored datasets just use the "data()" command as follows (we will use the "Seatbelts" dataset, which is the Road Casualties in Great Britain from 1969 through 1984)

data(Seatbelts)

If you type the "objects()" command, you should see the dataframe "Seatbelts" displayed as an object in your application.

If you want information on the contents of the dataframe, type the command "help(Seatbealts)" and R will display a help screen giving an extensive description of the data. Actually, Seatbelts is a much more complex object called a multivariate time series, but we will recast it to the simpler dataframe object by the following command:

Seatbelts<-data.frame(Seatbelts)

One basic command that is very useful when using datasets is the "names()" command. For instance, to display the names within the dataset do the following

names(Seatbelts)

[1] "DriversKilled" "drivers" "front" "rear" "kms" "PetrolPrice" "VanKilled" "law"

There are several means to access these fields. One way is to reference the data by name. So, to display all of the data associated with drivers killed, you would use either the command "Seatbelts$DriversKilled" or "Seatbelts[,1]". Notice how the second command displays all of the data in the first column, which corresponds to the "DriversKilled" field.

To obtain specific values within a dataframe you can use the same "[]" structure that was discussed regarding matrices earlier. However, you can also use conditional statements as well. For instance, if you wanted to know how many months the number of drivers killed exceeds 100, you could use the following command

sum(Seatbelts$DriversKilled > 100)

[1] 157

or you could use

sum(Seatbelts[,1]>100)

[1] 157

If you wanted to know the total number of deaths to rear seat passengers (the field "rear") in the months where the number of Drivers killed exceeds 100, you would use a conditional command like this

sum(Seatbelts[Seatbelts$DriversKilled>100,]$rear)

[1] 64586

or

sum(Seatbelts[Seatbelts[,1]>100,4])

[1] 64586

If you wanted the average you could replace the "sum()" function with the "mean()" function, or if you wanted the variance you would use the "var()" function.

If you wanted to replicate compound conditionals you can use the "|" construct for the "OR" logic operator or the "&" construct for "AND". For instance, the command

mean(Seatbelts[75 < Seatbelts$DriversKilled & Seatbelts$DriversKilled <= 110,]$rear)

[1] 367.7761

CONTINUED ON **PAGE 12**

displays the average number of rear seat passenger who died in the months where the number of drivers killed was between 75 and 110 (inclusive).

Another useful function that you can use on the data-frame is the "summary()" function. This function will produce a table of various statistics on each of the data fields. Observe its use in Figure 1.

CONCLUSION

This article has discussed some of the fundamentals required to manipulate your data in R. In the next article, we will discuss how R uses the Model Formulae framework to allow you to create multiple statistical models.

If you have found these articles to be beneficial, you might consider obtaining the "The R Book" by Michael J. Crawley, published by Wiley and Sons. This is an excellent book on R which can benefit both the amateur and professional in the pursuit of using the R language. ■

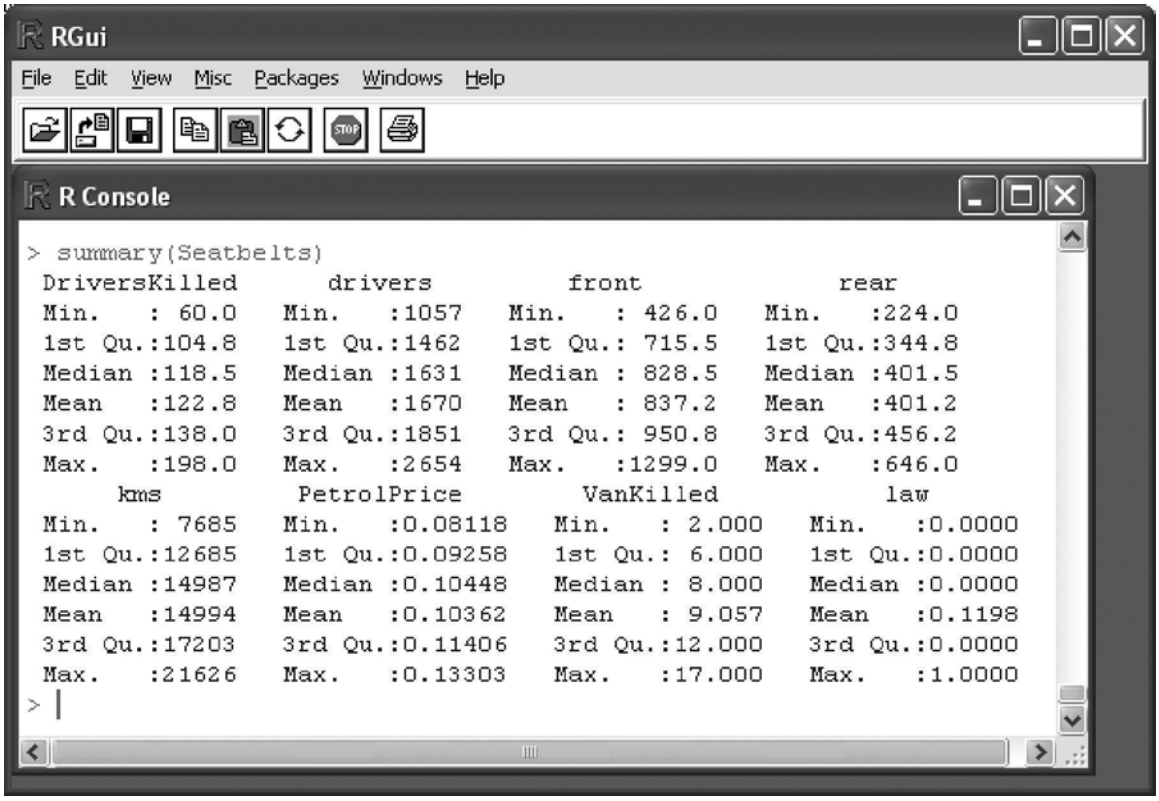

Figure 1: Results of the use of the "summary()" function.

FOOTNOTES

[i] R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.