CENTER FOR STATISTICAL AND MATHEMATICAL COMPUTING, INDIANA UNIVERSITY

Getting Started with R for Windows

Dani M. Marinova, Summer 2010

Edited by Eric A. Applegate, Winter 2011

TABLE OF CONTENTS

INTRODUCTION 2
How to Use this Document 2
What is R? 2
How do I download R? 2
ORIENTATION 2
Windows in R 2
Menus in R 3
Packages 3
PREPARING DATA FOR ANALYSIS 4
Data types 4
Functions 4
Reading data files into R 5
Manipulating data in R 6
Recoding variables 7
Saving your data 8
DESCRIPTIVE DATA ANALYSIS 8
Frequency Tables 8
Descriptive statistics 9
Visualizing data 10
DATA ANALYSIS 12
Two-Sample t-test 12
Simple linear regression 12
One-way ANOVA 14
R OUTPUT 16
Working with output 16
Working with graphs 16
FURTHER HELP 17
REFERENCES 17

Introduction

How to Use this Document

This tutorial introduces researchers to R for Windows. Although this tutorial is written specifically for users running Windows, much of the material also applies to R running on the Mac and Linux operating systems. We assume knowledge of basic statistical concepts but no prior experience with R. We recommend several resources for more advanced users of R at the end of this document.

What is R?

R is an open source statistical program, which can be used as a matrix-based programming language or as a standard statistical package. The main features of R include powerful data analysis and graphical tools, manipulation of matrices, and data handling and storage. R is an implementation of the S programming language, which is used in the commercial package S-Plus.

How do I download R?

R is available free of charge through the Comprehensive R Archive Network (CRAN). Please consult the GNU General Public License of the Free Software Foundation for the terms of usage. Download the executable file from one of the CRAN mirrors, and R will install itself. For detailed instructions on installing and troubleshooting R for Windows, consult the CRAN webpage

Orientation

Windows in R

Launch R by selecting the program file from the Windows Start Menu. You will see a typical Windows graphical user interface (GUI) with menus and toolbars. Additionally, you will see a window called the "R Console", or command window. The Console already has some text displayed, stating general information about the version number, licensing, citations, and help.

The prompt ">" at the end of introductory text indicates that you can begin typing commands directly in the Console window. For instance, you can type 3 + 5, press enter, and the answer will appear on the next line, followed by another prompt.

```
> 3 + 5
[1] 8
>
```

You can continue to type commands, one at each prompt, for the duration of your R session. Alternatively, you can use the R Editor to type several commands and submit them to the Command window (similar to what one might do in the SAS or SPSS syntax editors). The R Editor is accessible from the File menu (File -> New Script). Run commands from the Editor by highlighting the text, right-clicking on it and selecting Run line or selection (alternatively, highlight the text and press Ctrl+R). The

R Editor is a basic text editor, and several more sophisticated editors are available. Among the alternatives for the Windows platform are Tinn-R, WinEdt and Vim. Consult the CRAN site for a complete list of script editors and their features. Emacs Speaks Statistics (ESS) is an add-on package for the Emacs editor; it allows you to edit text and send commands directly to R. You can also copy/paste text into the Console from the Notepad text editor in Windows.

Menus in R

The drop-down menus allow you to customize your working environment. The "Change dir…" item in the **File** menu enables you to set the working directory for the current session of R. You may want to change this if you plan on opening or saving files from a particular storage location during your session. The GUI preferences under **Edit** will allow you to configure the console, graphics windows and editor. Under **Help**, you can access the R manuals in pdf format and seek help in the R online archives. Under **Packages**, you can set the CRAN mirror and select repositories for installing new packages, as discussed in the next section.

Packages

The R program you downloaded from CRAN contains the base R packages, which include some basic statistical commands and graphics utilities. For many types of statistical analyses, you will need to install additional packages. For instance, in order to work with regression splines on your data points, you may need to install and load the "splines" package. You can do so from the drop-down menu (Packages -> Install packages). When you choose this menu option, you will be asked to select a CRAN mirror; you may choose any mirror that you like, but it makes sense to choose a mirror that is not too far from you geographically. After you have selected a mirror and pressed "OK", a list of packages will appear. Find the package you are interested in using, and press "OK". You will see that there is new text in the Command window displaying the details of the package download and installation. Before using a package, you must load it for the current working session. You may do this by choosing (Packages-> Load package) from the Packages menu, or by typing: `library(pkgname)` at the prompt, where `pkgname` should be replaced by the name of the package you want to load. Once you have installed a package on your system, you will NOT need to re-install it each time you want to use it. You WILL, however, need to re-load the package every time you start a new R session in which you want to use it. Consult the CRAN site for a list of R packages available for download.

Preparing Data for Analysis

Data types

There are several types of data structures in R: vectors, matrices, arrays, factors, time series, data frames and lists. This tutorial will focus on data frames because they are the most commonly used structures for statistical analysis. Data frames are two-dimensional objects that include variable names and information about the variables (for example, whether they are numerical or categorical). Data frames can contain missing values coded as NA; however, most statistical analyses will require you to delete missing values. Data frames can be created with the data.frame() function applied to matrices or lists, or by importing external data files.

Functions

Many of the statistical procedures and commands you use in R will be in the form of a function. In a general sense, a function is some type of procedure that takes input arguments and produces output. For instance, the c() function combines together the arguments that are passed to it. So, in order to make a vector of 5 data points, I can type the following command:

```
> x=c(1,4,7,5,3)
```

and press enter. I have created a new vector named x and it has the five elements of 1,4,7,5, and 3. To view x and its values, type x at the prompt and press enter. Now, I can use other functions to calculate mathematical quantities from that vector. For instance to compute the sum and the mean of the five values, I can pass the vector x as the argument to the sum() and mean() functions:

```
> sum(x)
[1] 20
> mean(x)
[1] 4
```

These examples show that functions may need one argument (such as supplying the vector name x to the sum or mean functions) or multiple arguments (such as the five values to put into the vector x). Throughout the rest of this tutorial you will see many functions that “take” various arguments including data frames and option parameters. You have also seen that the output from a function can either be displayed in the command window (such as we did with the sum and mean functions) or it can be assigned to another data element (such as when we assigned the 5 values to the vector named x). The vector x will continue to have those 5 values associated with it throughout the rest of the R session, unless we re-assign new data to it or clear it.

To learn more about any function in R, use the help function with the argument being the function you want to see the help files for. For instance, to learn more about the c() function, typing:

```
> help(c)
```

will bring up a webpage with information about that function and how to use it.

Reading data files into R

For all examples in this tutorial, we will use a dataset from Jeffrey D. Sachs and Andrew M. Warner, "Sources of Slow Growth in African Economies," which is available through the Center for International Development at Harvard University. The function read.table() imports an external data file into R and creates a data frame for statistical analysis. Thus, in order to read the dataset into R, we use the following command:

```
    > mydata <- read.table("C:/user/temp/AfricaData.txt", header=TRUE,
sep=",")
```

The above command reads a text file from the user's temporary folder (you may need to specify a different directory) into an R data frame called mydata. Note the use of forward slashes in file names in R instead of backslashes. The argument header=TRUE specifies that the first line of the text file contains the names of the variables, and sep="," indicates that the values are separated by a comma. Typing:

```
    > mydata
```

and pressing enter will allow you to look at the data you've just read in:

```
            country growth govspend invest colony openmarket accesssea
1           ALGERIA  1.478   0.0632  27.14      1  0.0000000         0
2         ARGENTINA -0.688   0.0515  16.87      0  0.0000000         0
3         AUSTRALIA  1.152   0.0263  27.44      0  1.0000000         0
4           AUSTRIA  2.161   0.0753  25.89      0  1.0000000         1
5        BANGLADESH  0.141   0.2938   3.13      1  0.0000000         0

    institutions
1        4.36458
2        4.28125
3        9.42968
4        9.44791
5        2.73958
(The rest of the output is omitted.)
```

The dataset contains 117 countries and 8 variables. The variable country is a character while the rest of the variables are numeric. The variable country lists the country for each observation; growth indicates economic growth from 1970 to 1990; govspend is a measure of government spending; invest indicates investment share from 1970 to 1989; colony is a dummy variable indicating whether the country is a former colony; openmarket is the share of years between 1965 and 1990 for which the country had an open market economy; and institutions is a measure of the quality of institutions. Notice (in the full output on your screen) that there are multiple missing values, labeled NA.

The function class() will return the data type, and summary() will produce descriptive statistics for each variable:

```
> class(mydata)
[1] "data.frame"

> summary(mydata)
(Output is omitted.)
```

Manipulating data in R

Most statistical analyses in R cannot be implemented with missing values. To create a dataset without the missing values, ask R to omit all values labeled NA using the na.omit() function:

```
> mydata <- na.omit(mydata)
```

Note that this command overwrote the original data frame named mydata. You could specify a new data frame name for the non-missing data instead. To call a specific variable (e.g., country) from the dataset, type mydata$country or mydata[,"country"]. This is helpful if you have multiple data frames involved in your program. For instance if you have a country variable in both mydata1 and mydata2, you would want to make sure you are referencing the correct data by stating either mydata1$country or mydata2$country. If you only have one data frame that you are working with, or each data frame has different variable names, you can avoid having to type the mydata$ by using the attach() function. Attaching a data frame in R is like telling R, "From now on, look at this data frame as I reference variable names." The function detach() undoes attach(). The following commands attach the mydata data frame, and then ask for a listing of the country variable values:

```
> attach(mydata)

> country
[1] ALGERIA ARGENTINA AUSTRALIA AUSTRIA BANGLADESH BELGIUM
[7] BENIN BOLIVIA BRAZIL BURKINA FASO BURUNDI CAMEROON
[13] CANADA CAPE VERDE IS. CENTRAL AFR.R. CHAD CHILE CHINA
(The rest of the output is omitted.)
```

Again, the reason we could type country and not mydata$country is because we attached the data frame. The names() function will produce a list of variable names in the data frame specified:

```
> names(mydata)
[1] "country" "growth" "govspend" "invest" "colony" "openmarket"
[8] "institutions"
```

To rename variables or create variables names (if your data does not contain variable names), use the following syntax:

```
> names(mydata) <- c("Country", "Growth", "Gov.Spend", "Invest",
"Colony", "Open.Market", "Institutions")
```

Here we see another use of the c() function, this time it puts together a list of string values (in quotation marks) and assigns it to the names of the mydata data frame. Now examine the variable names of mydata and see how they have changed:

```
> names(mydata)
[1] "Country" "Growth" "Gov.Spend" "Invest" "Colony" "Open.Market"
[8] "Institutions"
```

To make sure that R associates the variable names with the variables, use the function attach() after making changes to the dataset.

Handy Tip: in the last three commands, we typed names(mydata) three times. In the R Console, it remembers the last commands you have used. So, instead of typing that command three times, we could have pressed the up arrow key on our keyboard to recall the last command used. The more times you press the up arrow key, the further back you can go to retrieve previously used commands. (Similarly, if you are reviewing previously used commands, you can also press the down arrow button on your keyboard to scroll through commands in the opposite direction).

Recoding variables

Suppose you want to create a dummy variable that is coded 1 if the observation meets certain criterion and 0 otherwise. For example, say you want to create a dummy variable based on the variable Open.Market, which is the fraction of years (from 1965 to 1990) during which a country is rated as an open market. Open.Market.Dummy should be equal to 1 when Open.Market is greater than 0.5 and equal to 0 when Open.Market is less than 0.5. By using the notation mydata$, you can create a new variable within the existing dataset mydata.

```
> mydata$Open.Market.Dummy <- as.numeric(Open.Market>0.5)
```

The as.numeric() function creates or coerces objects to be numeric. Since we specified a condition as the argument, the resulting value of that condition will create a numeric value of 1 (if the condition is true) or 0 (if the condition is false). Suppose now that you want to create a new variable, Open.Market.Cat, which has three categories: 1 when Open.Market is less than 0.33, 2 when Open.Market is between 0.33 and 0.66 and 3 when Open.Market is greater than 0.66. The following lines assign the specified values (1,2, or 3) to the new variable mydata$Open.Market.Cat depending on whether the condition(s) in the brackets is true or not:

```
> mydata$Open.Market.Cat[Open.Market < 0.33] <- 1
> mydata$Open.Market.Cat[Open.Market >= 0.33 & Open.Market <=
0.66] <- 2
> mydata$Open.Market.Cat[Open.Market > 0.66] <-3
```

In addition to creating new variables, you may also want to delete existing variables from your data set. To do so, assign that variable the value of NULL:

```
> mydata$Open.Market.Dummy <- NULL
```

Although R is not an ideal environment for entering and coding data manually, you may change the values of specific data cells by specifying the column and row numbers. For example, notice that Australia is not coded as a former colony (Colony has a value of 0). To change this, specify the column and row numbers of the observation you wish to change. Australia is the third country in the dataset, and Colony is the fifth variable. Then assign that cell a value of 1:

```
> mydata[3,5] <- 1
```

Saving your data

To save your data as a text file, use the command write.table(). Specify the name of the data object you wish to export (in this case, mydata), the directory where you wish to save the file, and the separation method.

```
> write.table(mydata, file="C:/user/temp/AfricaData1.txt",
sep = ",")
```

Descriptive Data Analysis

Frequency Tables

To generate frequency tables, use the table() function. If you are interested in the frequencies of values for one variable, specify the variable name as the only argument to the table() function. You can also specify two variables separated by a comma in the function to create a contingency (cross-tabulation) table – the first variable listed will be put in the rows and the second variable will be put in the columns. The table() function will create tables of frequencies only. The commands margin.table() and prop.table() create tables of marginal frequencies and proportions for an existing contingency table that you have created. In these two functions, the first argument should be the name of a table object you have saved, and the second argument can be 1, 2, or blank. A second argument of 1 would analyze by row, 2 would analyze by column, and blank would analyze based on the total of the table. Try the following examples. The function ftable() displays the results more legibly.

Handy Tip: You can insert comments into your R syntax by typing a # (pound) sign. Any text after the # sign is not processed. #

```
> attach(mydata)
> table1 <- table(Colony, Open.Market.Cat)
> ftable(table1)
> table2 <- margin.table(table1, 1) #Frequencies summed over
Open.Market.Cat
> table3 <- margin.table(table1, 2) #Frequencies summed over Colony
> table4 <- prop.table(table1)
```

```
> ftable(table4)

           Open.Market.Cat          1          2          3
        Colony
        0                  0.22666667 0.01333333 0.33333333
        1                  0.34666667 0.01333333 0.06666667
```

For two-way tables, you can use the chisq.test() function to test the independence of the row and column variables.

```
> chisq.test(table1)

    Pearson's Chi-squared test
        data: table1 X-squared = 11.7733, df = 2, p-value = 0.002776
```

Descriptive statistics

In addition to the summary() function discussed above, descriptive statistics may be obtained for individual variables, using the following functions: mean(), median(), max(), min(), range(), var(), sd(), quantile(), fivenum(), length(), which.max(), which.min(). For example, to obtain the range and standard deviation of the variable Gov.Spend, type:

```
> range(Gov.Spend)
[1] 0.0057 0.3280

> sd(Gov.Spend)
[1] 0.06049456
```

For bivariate descriptive statistics, use the cor() and cov() functions for correlation and covariance, respectively. To find the correlation between the variables Growth and Invest, type:

```
> cor(Growth, Invest)
[1] 0.4751891
```

To find the correlation and covariance matrices of mydata, select all variables but Country (because it is a character string) by asking R to include only observations from the second to the eighth column:

```
> cor(mydata[,2:8])
> cov(mydata[,2:8])
(Output is omitted.)
```

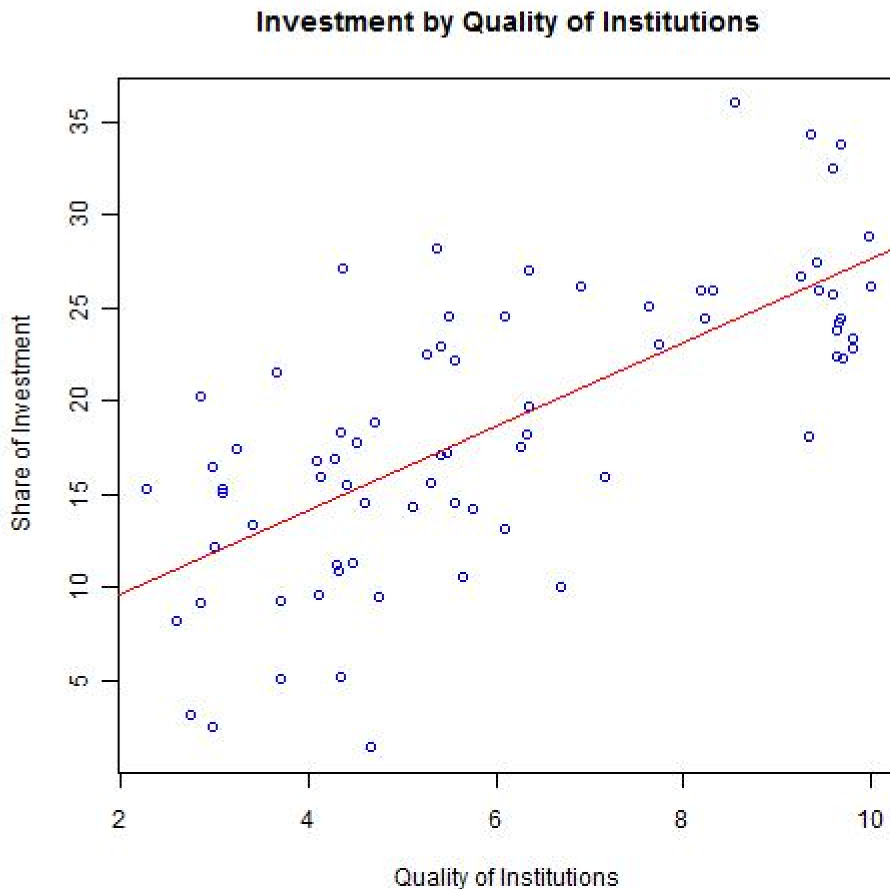
We have used similar bracket notation before when selecting a specific observation in the data matrix, i.e .[2,3] represents row 2 and column 3. In this instance, by not specifying a row index, we are telling R to use all the rows. By specifying 2:8 in the column index, we are telling R to use columns 2,3,4,…7,8 in the dataset for this particular analysis.

Visualizing data

To create a scatterplot of two variables, say, Invest and Institutions, use the following syntax:

```
> plot(Institutions, Invest, xlab="Quality of Institutions",
ylab="Share of Investment", main="Investment by Quality of
Institutions", col="blue")
> abline(lm(Invest~Institutions), col="red")
```

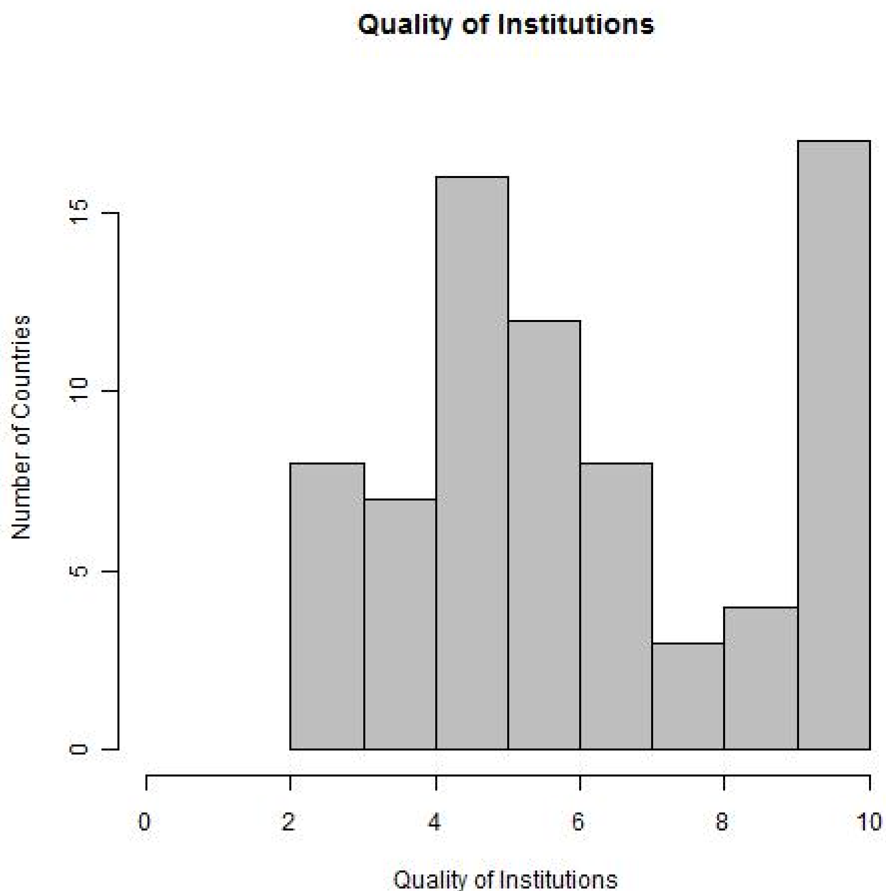
The code produces the scatter plot below. The first argument lists the variable on the horizontal axis and the second arguments lists the variable to use on the vertical axis. The arguments main, xlab and ylab specify the title of the plot and labels of the x and y axes, respectively. The argument col changes the color of the points from the default, black. The abline() function allows you to add a line to the scatterplot by specifying the slope and intercept of the line with the lm() function, which is explained below.



You may also wish to visualize your data using a histogram. The following function produces a histogram of the Institutions variable:

```
    > hist(Institutions, col="gray", xlim=c(0,10), ylim=c(0,18),
xlab="Quality of Institutions", ylab="Number of Countries",
main="Quality of Institutions")
```

The argument col fills the bars with gray; xlim and ylim set the ranges on the x- and y-axes by specifying vectors (made with the c() function) representing the minimum and maximum values for each range. The xlab and ylab arguments label the x- and y-axes, respectively; main sets the title of the plot.



Data Analysis

Two-Sample t-test

To illustrate the procedure of a two-sample t-test in R, suppose you are interested in whether or not former colonies and countries without a colonial past differ in the quality of their institutions. The data allows for such a comparison with the Colony and Institutions variables. Implementing a t-test in R is possible with the t.test() function. The t.test() function requires specification of the variable you are comparing (in this case, Institutions) and the group variable (Colony). The alternative statement specifies the type of test: two.sided, less, or greater are the available options. The var.equal argument can be set to treat the variances as equal; if TRUE, the pooled variance estimates are used, and if FALSE, the Welch approximation to the degrees of freedom is used.

```
        > t.test(Institutions~Colony, alternative="two.sided",
var.equal=TRUE, conf.level=0.95)
        Two Sample t-test
        data: Institutions by Colony
        t = 3.7596, df = 73, p-value = 0.0003405
        alternative hypothesis: difference in means not equal to 0
        95 percent confidence interval:
        0.914618 2.978317
    mean in group 0 mean in group 1
       6.889169        4.942702
```

The p-value is less than $\alpha = .05$, thus we reject the null hypothesis that there is no difference in quality of institutions between colonies and non-colonies. Colonies and countries without a colonial past have different mean quality of institutions.

One sample t-tests are easily implemented in R by omitting the group variable in the t.test() function. Paired t-tests are also possible with the t.test() function by setting the argument paired=TRUE.

Simple linear regression

Say you want to implement the following model in R:

$$Invest = ß_0 + ß_1(Institutions) + ß_2(Open.Market)$$

The syntax for the model above is the following:

```
        > results1 <- lm(Invest~Institutions+Open.Market, data=mydata)
```

The function lm() is used to fit linear models in R. The function requests specification of the model's formula as the first argument, which is Invest ~ Institutions+Open.Market (the intercept is included by default). In addition, you need to specify in which data frame the function should look for the variables/data by using the data= argument. Storing results in an object (results1 in this example) will allow you to invoke several functions for useful information about the model results. The function

summary() provides the model's residuals, estimates for the model's coefficients, standard errors, t statistics and p-values, as well as model statistics (F-statistic, R-squared, etc.).

```
    > summary(results1)
Call: lm(formula = Invest ~ Institutions + Open.Market, data = mydata)

    Residuals: Min 1Q Median 3Q Max -12.67273 -3.26294 0.03802 3.21995
13.48049

    Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
    (Intercept)     7.833      1.790   4.377    4e-05 ***
    Institutions    1.335      0.370   3.607 0.000567 ***
    Open.Market     6.487      1.986   3.266 0.001672 **
    ---
    Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

    Residual standard error: 5.138 on 72 degrees of freedom
    Multiple R-squared: 0.5616,     Adjusted R-squared: 0.5495
    F-statistic: 46.13 on 2 and 72 DF,  p-value: 1.276e-13
```

Other useful functions are coef(), resid() and fitted(), which return the model's coefficients, residual errors on the dependent variable, and the predicted values of the dependent variable, respectively.

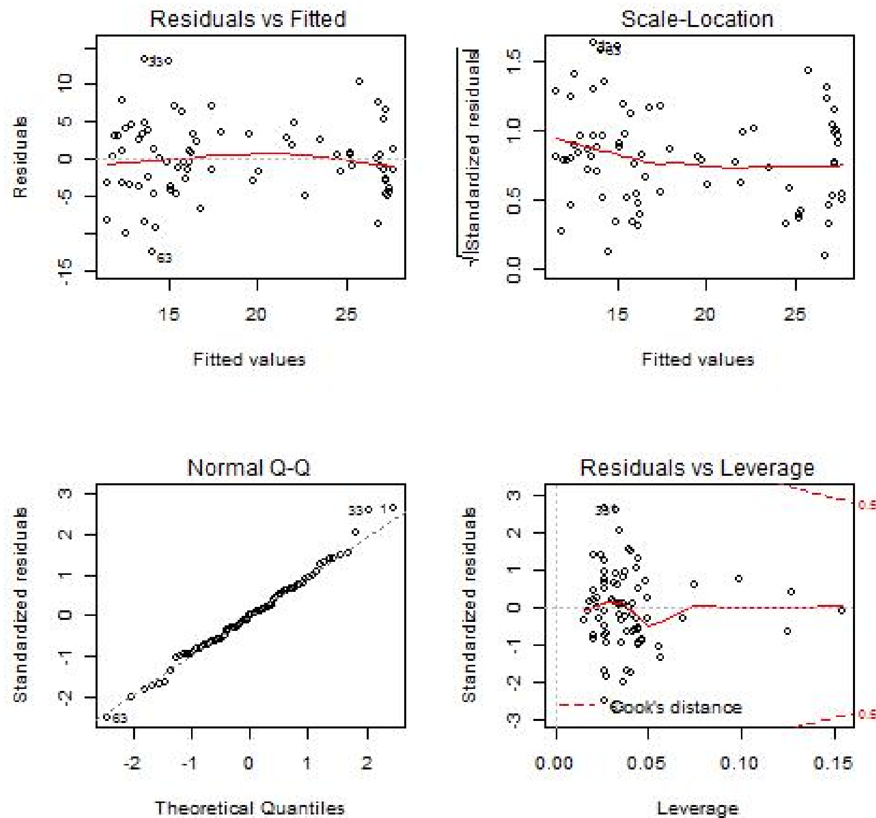
In addition to fitting a first-order model in Institutions and Open.Market, you may also want to include an interaction term between Institutions and Open.Market, include a polynomial term (e.g., Institutions^2) or exclude the intercept from the model. Each of these models is described in Table 1.

Model	R Code
Interaction Term: Invest = $ß_0$+ $ß_1$(Institutions) + $ß_2$(Open.Market) + $ß_3$(Institutions*Open.Market)	`results2 <- lm(Invest ~ Institutions + Open.Market + Institutions:Open.Market, data=mydata)`
Polynomial Term: Invest= $ß_0$+ $ß_1$(Institutions) + $ß_2$(Open.Market) + $ß_3$(Institutions^2)	`results3 <- lm(Invest ~ Institutions + Open.Market + I(Institutions^2), data=mydata)`
Exclude Intercept: Invest = $ß_1$(Institutions) + $ß_2$(Open.Market)	`results4 <- lm(Invest ~ -1 + Institutions + Open.Market, data=mydata)`

Finally, you should evaluate the results of your model by examining the residual errors and scanning the data for significant outliers. One way to do so is by using the plot() function:

```
> layout(matrix(1:4, 2, 2))
> plot(results1)
> layout(1)
```

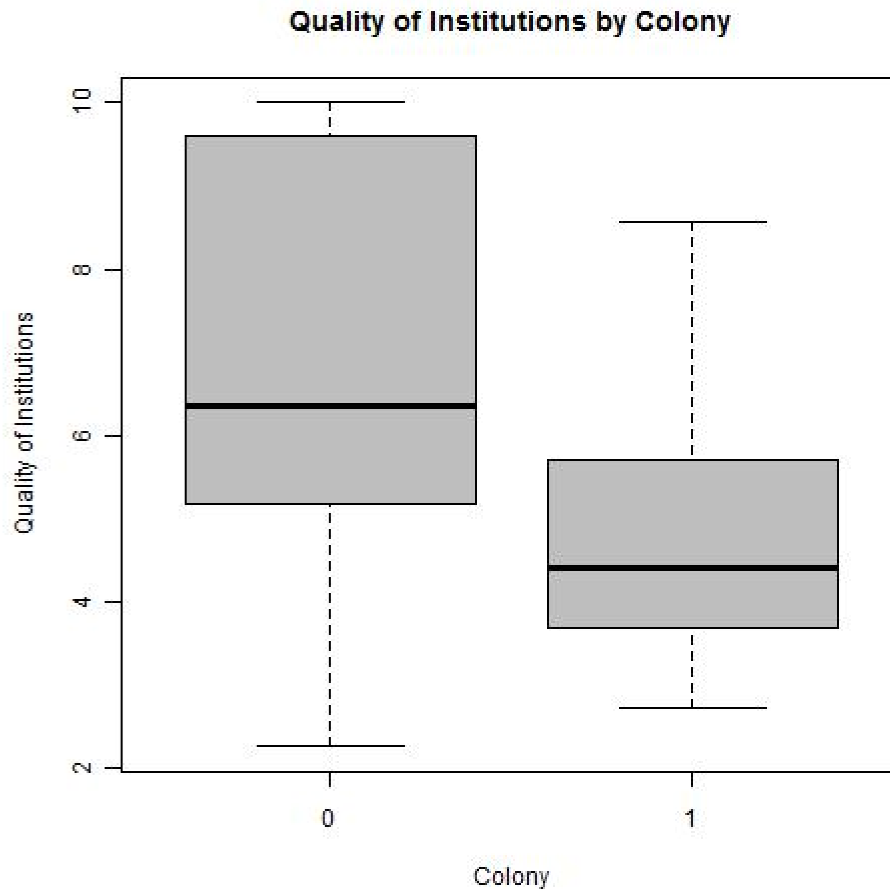
The layout() function formats the window for the subsequent plot function. In this case, we create a 2x2 window with four slots. Next, the plot() function generates four plots for the first model we fit, whose results were stored in the object results1. The top-left slot graphs the residual errors against the fitted values; the bottom-left slot is a Q-Q plot; the top-right slot graphs the square root of the standardized residuals against the fitted values; and the bottom-right slot graphs the leverage of each observation, with Cook's distance superimposed on the plot. R produces the axes and plot labels automatically.



One-way ANOVA

Analysis of variance (ANOVA) is easily implemented in R. Returning to the example from earlier on implementing a t-test, suppose you want to compare the mean values of quality of institutions for former colonies and countries without a colonial history through one-way analysis of variance. You might begin your analysis with a boxplot in order to compare the distributions of quality of institutions for colonies and not-colonies.

```
      > boxplot(Institutions~Colony, xlab="Colony", ylab="Quality of Institutions",
main="Quality of Institutions by Colony", col="gray")
```



For a formal ANOVA test, use the aov() function, specifying the model's formula and the dataset to be used, as in the lm() function.   Use the summary() function to view the results.

```
> results5 <- aov(Institutions~Colony, data=mydata)

> summary(results5)

            Df  Sum Sq Mean Sq F value     Pr(>F)
Colony       1   70.02  70.016  14.134 0.0003405 ***
Residuals   73 361.61   4.954
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Finally, you may wish to examine the results visually. Use the layout() and plot() functions, as described earlier.

R Output

Working with output

Instead of copying R output and pasting it into a text file, the function sink() allows you to send R output directly into an external text file. The first argument of sink() specifies the directory and file where your output will be saved. A second argument, split, is used to control whether the output appears in the console window or not. If the split argument is TRUE, your output will be sent to an external file as well as displayed in the R console (set split to FALSE if you prefer your output not to be displayed in the console). You would use the sink() function before any analysis that you want directed to the output file. After running your analyses (one or multiple), the sink() function without any arguments interrupts the process, and subsequent analyses are not sent to the output.txt file.

```
> sink("C:/user/temp/output.txt", split=T)
> lm(Invest~Institutions+Gov.Spend+Growth+Open.Market)
> sink()
```

Working with graphs

To export R graphs to an external file, use the jpeg() function, in which you specify the directory and file (extension .jpg) where the R graph should be saved. As is the case with sink(), you would use the jpeg() function before creating the graph, and then use the dev.off() function to shut down the jpeg() function so subsequent graphs are not saved as external .jpeg files.

```
> jpeg("C:/user/temp/modeloutput.jpg")
> layout(matrix(1:4, 2, 2))
> plot(results1)
> dev.off()
```

Further Help

To get help from the R online documentation, use the help() and help.search() functions. The help() function is useful if you already know the command you will be using and need information on its syntax. The help.search() function searches the R documentation for keywords.

```
> help(plot)
> help.search("regression")
```

To quit R, use the quit() function:

```
> quit()
```

For a more comprehensive introduction to R, consult the pdf "An Introduction to R."

The Indiana University Knowledge Base provides answers to questions about using R on the Indiana University campuses as well as general questions about R functions.

If you have prior experience with another statistical package, you may find Quick R for SAS/SPSS/Stata Users helpful.

For SAS users, the SAS-and-R Blog may be a particularly useful tool.

The UCLA Academic Technology Services offer R resources for advanced users.

References

Jeffrey D. Sachs and Andrew M. Warner (1997) "Sources of Slow Growth in African Economies," Journal of African Economies, 6(3): pp. 335-76.

Kristel Van Steen, "Using R for Linear Regression."