Using R for Windows and Macintosh

R is the most commonly used statistical package among researchers in Statistics. It is freely distributed open source software. For detailed information about downloading and installing R, please see the R project website at http://www.r-project.org.

R is a programming language; therefore the challenges for beginners to learn R lie in writing the commands for analysis. Some investment of effort is required before mastering R. One of the great advantages of R is its flexibility. R allows you to download add-on packages for targeted analysis, write your own packages, and share your packages with others. This document covers the basic features and statistical commands of R.

Throughout this document, text written in `this font` will indicate text that you should type verbatim into the R console. Text *in italics* will indicate text that you should substitute with the correct filename, pathname, command name, or other information specific to your needs.

Table of Contents

Getting Started with R

Similar to SPSS, Stata and SAS, R is a command- and window-driven application. That is, it has a graphical interface consisting of pull-down menus, allows users to issue commands for procedures, and displays results. You can perform numerical and graphical tasks in R by writing commands in the R console window. Once installed, you can access R on a PC from the Start menu, on a Macintosh in the Applications list in the Finder, or by double-clicking on the R icon (two variants are shown below) on the desktop, if applicable.

When you start R, a screen like the one below will appear:

Windows:
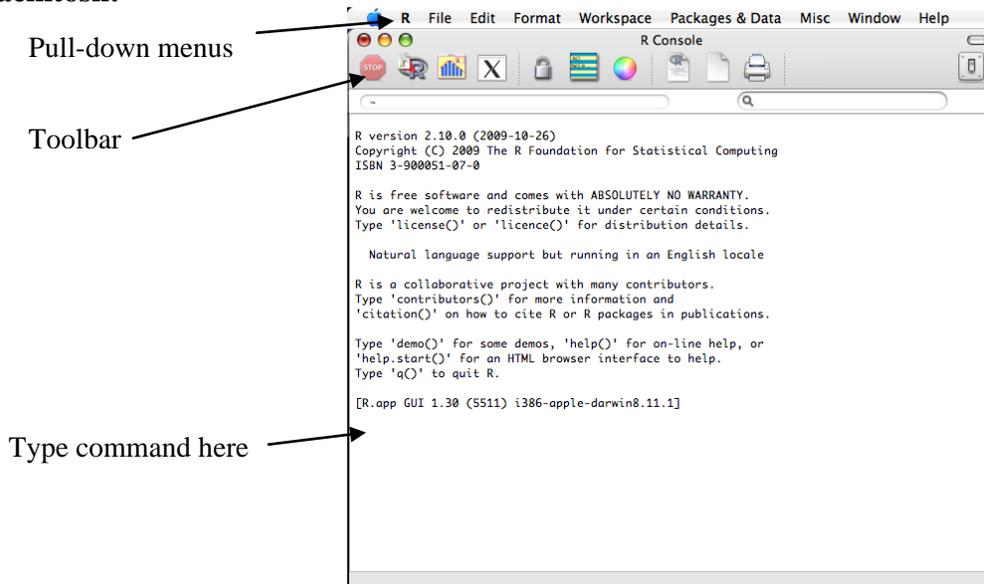
Pull-down menus

Toolbar


```
R version 2.9.2 (2009-08-24)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

Type command here

Macintosh:

Pull-down menus

Toolbar


```
R version 2.10.0 (2009-10-26)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.30 (5511) i386-apple-darwin8.11.1]
```

Type command here

While the Macintosh and Windows interfaces look different, they are functionally very similar. Like most window-based programs, R has a toolbar and a menu bar with pull-down menus that you can use to access many of the features of the program. The toolbar contains buttons for more commonly used procedures. To see what each button does, hold the mouse over the button for a moment and a description of what the button does will appear. The following is a summary of the main pull-down menus and their functions:

Menu	Functions
File	Open source R code, create, open and save script, load and save workspace, load and save history, display files, change working directory, print files, and exit R.
Edit	Copy, paste, select all, clear console, data editor, R configuration window editor.
Misc	Stop current computation, buffer output, list objects in the memory, remove all objects, and list search path.
Packages / Packages & Data	Load, install, update packages, set CRAN mirror, select repositories, install packages, install packages from local zip files.
Window(s)	Cascade and tile R console windows. Arrange icons and switch among windows.
Help	Get help on R procedures, commands, and connect to the R website for more help information.

R Windows

The following is a summary of the functions and contents of three different types of windows in R. All of the windows are accessible from the pull-down **Windows** menu.

You may write R commands in the **R Console** window (pictured above). This window also displays all the commands R has run, the results, and error report. This window appears when you launch R. To bring this window to the foreground, click on the window or go to the **Windows** pull-down menu and choose the **R Console**. You can type and run commands in this window line by line.

Alternatively, you can write all the commands in **R Editor** window and allow R to run part or all of the commands at once. You open this window by clicking the **New Script** option in the **File** menu. If you want to redo your analysis at a later time, or send your code in a file to someone else, you can save the scripts. You can also print the scripts by clicking the **Print** option from the **File** menu.

The **R Graphics** window opens automatically when you create a graph. To bring a graph to the

foreground, click on the graphics window or go to the **Windows** pull-down menu and choose the **R Graphics window**. Graphs can be saved in various formats, such as jpg, png, bmp, ps, pdf, emf, pictex, x_g and so on.

Accessing Help in R

R provides help files for all its functions. To access a help file, use the Help pull-down menu, or type ? followed by a function name in the R console. For example, to get help on the scan function, type:

```
> ?scan
```

Notation and Common R Operators

```
In this document, all R commands will be shown in this font.
```

>   Indicates the prompt at the start of each new line in the R console.

\#   The comment operator. Often called the "hash sign" or "pound sign".

          Any type following the # on a line indicates a comment. R will ignore (not execute or attempt to interpret) anything written as a comment. Comments are a good way to indicate what you intend a line to do, or to describe a variable or command. They are useful to you when you read your code later or, and are useful to others with whom you share your code.

<-   The assignment operator. This operator assigns the value on the right to the symbol on the left. Some pronounce this as "gets", so the statement x <- 5 is read "x gets 5". Once you have assigned 5 to x, R will fill in the value 5 in expressions using x.

          You can also use the = sign for assignment, but in R the <- sign is more conventional.

==   Boolean equality operator. A double = sign is used in logical statements, assessing whether two quantities are equal.

```
x <- 5  # assigns the value 5 to x

x == 5  # returns TRUE

x = 6  # assigns the value 6 to x, overwriting the previous
statement x <- 5. [Note that '=' accomplishes the same thing
as '<-']  Now,

x == 5  # returns FALSE
```

Working with Data Files

Entering Data into R

There are several ways to read external data files into R. This section covers three common ways to read data files into R.

Reading Data from a Line

Use **scan()** to read data from a line. The following is an example that reads height and gender data of 5 children and puts them into a dataset named "data1". This strategy makes the most sense when you have a small amount of data to read in.

```
> height <- scan() # Create variable: height

> 1: 2.2 2.5 3.4 2.9 # Type in data points with white space as
                      separator.
                      # Note: After you call scan(), R prompts you with
                      # the 1:, meaning the first thing you enter will be
                      # the first element in the vector of values.

> 5:               # Here, R is prompting you to enter another value,
                   # which would be the 5th value in the vector you are
                   # entering.  If you are finished entering data, hit
                   # return again.

> gender <- scan() # Create another variable: gender

> 1: 1 0 1 1       # Type in data

> 5:               # Type return again.

> data1 <- data.frame (height, gender)

                   # Create data file data1 which has two variables:
                   # height, and gender
```

Often, you will want to store these variables together, in a data frame. To do so, use the `data.frame` function to combine them:

```
> measurements <- data.frame(height, gender)      # Creating the
                  # measurements data frame with variables height and
                  # gender.

> measurements  # Typing the name of an object in R gets R to print
                # out the contents of that object—in this case, the
                # variables and values in the data frame.


  height gender
1    2.2      1
2    2.5      0
3    3.4      0
4    2.9      1
```
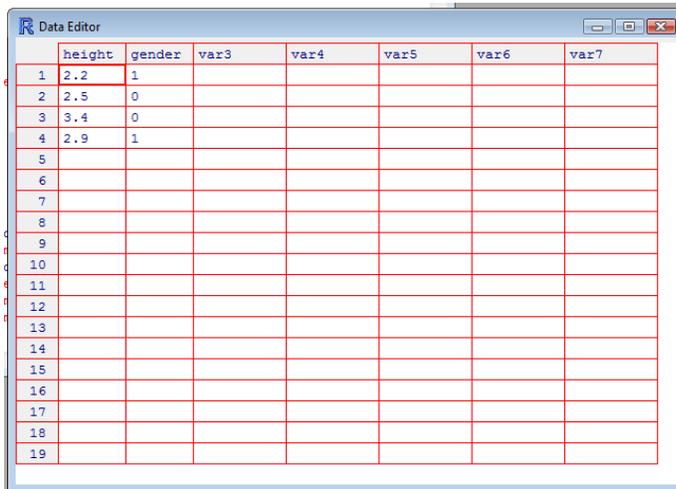
The `data.frame` function assumes the observations for each variable were entered in the same order, so the first height observation corresponds to the first gender observation, and so on.

Using the Edit GUI

R provides a spreadsheet-style data editor. To access it, first create your data frame, and then use the edit function.
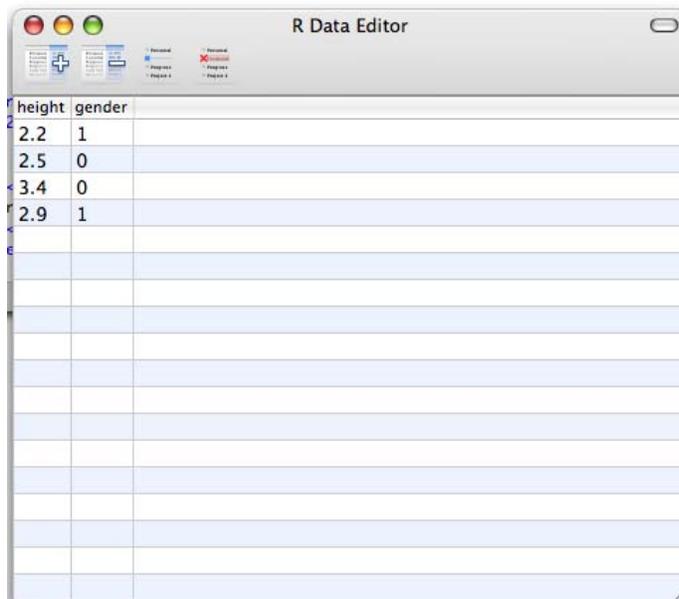
```
> measurements <- edit(measurements)
```

This will open the data editor window. Here is the Windows version:

Here is the Macintosh Version:

Notice that the data you have already entered is shown, with each variable occupying a column, and each case occupying a row. You can add new variables and new cases, and can change entered values.

Unlike other spreadsheet programs (e.g. Microsoft Excel), you cannot save your data as you are working on it. You save your data by assigning it to a variable name. For example, you can access the data editor with this command:

```
> edit(measurements)
```

This will open the data editor, and it will look exactly the same as above, but **any changes you make will not be saved.**

To make changes that will be saved, you must assign the result of your edits to a dataset, and *you must do so when you open the data editor*. You could assign the result to a new version of the dataset measurements:

```
> measurements2 <- edit(measurements)
```

Or, you could assign it to the same name, in effect updating measurements with the edits you are about to make:

```
> measurements <- edit(measurements)
```

You could also use the `fix` function, which calls `edit`, and then assigns the new version to the same name.

```
> fix(measurements) # equivalent to measurements <- edit(measurements)
```

In addition to the challenges of saving, the R data editor lacks many of the helpful features of a spreadsheet program. For example, it does not allow you to cut and paste parts of rows or columns. For this reason, we don't recommend using it to enter large amounts of data. However, it can be quite handy for making minor changes to data you have already read into R.

Importing Delimited Data Files

The `read.table()` function is to read delimited data files. The *delimiter* in a text file is the character or set of characters used to separate values for different variables on a row.

For a complete list of possible arguments to the `read.table()` function, type `?read.table` in the R console. Most arguments are optional, with the exception of the file name from which to read the data. Some commonly used arguments are:

`file`	The complete path and filename of the file to be read. Required.
`header`	Indicates whether the first line of the file is a header line, containing the names of the columns (likely variable names). By default, this is false. To indicate that a file contains a header row, use `header = T`.
`sep`	The character(s) acting as a separator between values within a row. Often this will be a comma, a space, or a tab. To indicate a tab, use \t; to indicate any whitespace, use \w.

`nrows`	An integer value specifying how many rows should be read. By default, `read.table` will read the entire file. The `nrows` argument allows you to specify a smaller number, which will start from the first line of the file.
`skip`	An integer value specifying how many rows should be skipped from the top of the file. By default, `read.table` will start from the first line. The `skip` argument can be useful if a file contains multiple lines of header information that should not be read in.
`dec`	The character used to indicate a decimal. By default, this is a dot: "."

R provides some variants of `read.table`: `read.csv` and `read.delim`, which have different default values and are tailored for csv files and tab-delimited files respectively.

In `read.csv`, default values are: `header = T, sep = ",", dec = "."`

In `read.csv2`, default values are: `header = T, sep = ";", dec = ","`

In `read.delim`, default values are: `header = T, sep = "\t", dec = "."`

In `read.delim2`, default values are: `header = T, sep = "\t", dec = ","`

Example:

```
baseball <- read.table("C:/Documents/baseball.txt", header=T, sep=" ")
```

Here, `baseball` is the name of the dataset, once it's read into R; `C:/Documents/baseball.txt` is the path and name of the file from which to read the data; `header=T` indicates that there is a header line of column names, and the values within each row are separated by a single space.

Importing Data over the Internet

The `read.table()` function can also be used to read a data file over the internet. The following is an example of reading data from a course website.

```
hayfever <-read.table('http://www-
stat/~jtaylo/courses/stats191/data/hayfever.table',header=T, sep=",")
```

Importing Fixed Width Format Files

Fixed width files are those data files that have no field delimiters. We may use the function `read.fwf()` to read this kind of data into R. By using the argument width, we let R know the width of each variable in the file.

```
datafile<-read.fwf(data, width=c(2,2,3))
```

Here, the first variable is one column wide (and is in columns 1-2), the second variable is two

columns wide (and is in columns 3-4), the third variable is three columns wide (and is in column 5-7).

Fixed width format files do not include the variable names in the first row. R will automatically name the variable `V1, V2, V3`… after you read in the data.

You could use `colnames` to specify the variable names that you want.

```
colnames(datafile) <- c("id", "height", "weight")
```

Importing Data Files from Other Statistical Software Packages

R can import data from other statistical software packages. To do so, you need to first load the package "`foreign`". It allows you to import data from many popular statistical software packages, such as Stata, SPSS, SAS, Minitab, S, Systat, dBase, and others. For more information on using packages in R, see the Guide to R Packages on the SSDS "Getting Started Guides" page. To install the `foreign` package, use these commands:

```
> install.packages("foreign")
```

```
> library(foreign)
```

After installing, to view the packages in the library, use:

```
> library(help=foreign)
```

To read in an SPSS file called "datafile.sav", use:

```
> datafile <- read.spss(file = "C:/Documents/datafile.sav")
```

Similarly, use read.dta to read in Stata binary files, and read.ssd to read in SAS files.

To read in data from Excel, it is best to use the **Save As** function in Excel to save the data as a .csv file or a tab-delimited text file (.txt), and then read the data using `read.table` (or `read.csv`, or `read.delim`) as described above.

Viewing Data in R

After you have imported or entered a dataset, you can view it by typing the name of the dataset. R will display the data within the console window:

```
> airquality
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
.....
```

To check the dimensions of the data:
```
> dim(airquality)
[1] 153    6
```

Here, 153 is the number of cases (or rows), and 6 is the number of variables (or columns).

To display the names of the variables in the dataset:
```
> names(airquality)

[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
```
To access a single variable, or column, of a dataset, you can us the $ operator in between the name of the data frame and the name of the variable:
```
> airquality$Ozone

[1] 41 36 12 18
```
Alternatively, if you are planning to work extensively with one dataset, you can `attach` the dataset. This allows you to access its variables directly:
```
> attach(airquality)
> Ozone
[1] 41 36 12 18
> Wind
[1]  7.4  8.0 12.6 11.5
```
When you have finished with the dataset, you can `detach` it:
```
> detach(airquality)
```
It is important to exercise caution when attaching datasets. For example, if you already had an object in your R environment called `Day`, the `Day` variable from the attached dataset would be masked by the pre-existing `Day` object, which may be completely different. Additionally, if you want to make changes to a variable in the `airquality` dataset, for example to the variable Ozone, you will need to assign the altered version to `airquality$Ozone`. To illustrate, let's say you want to double all values of the Ozone variable. If you have attached the `airquality` data frame, then this code will appear to work:
```
> Ozone <- Ozone*2
> Ozone
[1] 82 72 24 36
```
In fact, this code creates a separate object called `Ozone`, which is not part of the `airquality` data frame. The new values are assigned to this new object. If you would like the new values to be part of the `airquality` data frame, you need to do this:
```
> airquality$Ozone <- airquality$Ozone*2

> airquality$Ozone

[1] 82 72 24 36
```
This code will work regardless of whether the `airquality` dataset is attached at the time.

Selecting Subsets of a Dataset

Viewing Subsets of a Dataset

A data frame in R is constructed as a set of named vectors. In this sense, it is also a two-dimensional array, made up rows and columns. You can use square brackets [ ] to access values at a given index in the array using *data.frame.name*[*row, column*]. For example, to access the value at the second row, third column, of the `airquality` data frame use:

```
> airquality[2,3]

[1] 8
```

If you leave one of the indices blank, you can access an entire row or column.

To view the second column of the dataset, use:

```
> airquality[,2]
[1] 190 118 149 313
```

To view the fourth row, use:

```
> airquality[4,]
  Ozone Solar.R Wind Temp Month Day
4    36     313 11.5   62     5   4
```

The colon ： is used to indicate a range of rows or columns. To view the second through fourth columns, use:

```
> airquality[,2:4]
  Solar.R Wind Temp
1     190  7.4   67
2     118  8.0   72
3     149 12.6   74
4     313 11.5   62
```

To view the second through fourth columns of only the third row, use:

```
> airquality[3,2:4]
  Solar.R Wind Temp
3     149 12.6   74
```

The combine function, `c`, is used to combine individual values into a set. Here, you can use it to specify a set of columns to select from the dataset.

```
> airquality[,c(2,3,6)]
  Solar.R Wind Day
1     190  7.4   1
2     118  8.0   2
3     149 12.6   3
4     313 11.5   4
```

You can also use the square brackets to specify logical conditions. For example, to view observations where Wind is greater than 10 and Temp is less than 70:

```
> airquality[airquality$Wind>10 & airquality$Temp<70,]
```

```
  Ozone Solar.R Wind Temp Month Day
4    36     313 11.5   62     5   4
```

Subsetting a Dataset

You can extract and use any of the above subsets in analysis by assigning them to a new name:

```
> cool_&_windy <- airquality[airquality$Wind>10 & airquality$Temp<70,]
> cool_&_windy
  Ozone Solar.R Wind Temp Month Day
4    36     313 11.5   62     5   4
```

You can accomplish the same end with the `subset` function:

```
> cool_n_windy <- subset(airquality, Wind>10 & Temp<70)
> cool_n_windy
  Ozone Solar.R Wind Temp Month Day
4    36     313 11.5   62     5   4
```

The `subset` function also allows you to specify only certain columns to include. Here, we include only the `Month` to `Day` columns and only in the case that `Wind>10` and `Temp<70`:

```
# select = Month:Day tells R to give the columns from Month to Day and
# anything in between (which is nothing here)

> cool_n_windy <- subset(airquality, Wind>10 & Temp<70, select =
Month:Day)
> cool_n_windy
  Month Day
4    5    4
```

Exporting Data

We usually use `write.table` to export an R dataset into a different format. The `write.table` function has many of the same arguments as `read.table` (above). For example, to export `cool_n_windy` from the previous example as a tab-delimited textfile format to the "C:/Mydata" directory:

```
> write.table(cool_n_windy, "C:/Mydata/cool_n_windy.txt", sep = "\t")
```

To export it as .csv format, you can use the `write.csv` variant:

```
> write.csv(cool_n_windy, "C:/Mydata/cool_n_windy.csv")
```

Here, by default, `sep = ","` and `dec = "."`. In `write.csv2`, by default `sep = ";"` and `dec = ","`. There is no write.delim function.

Analyzing Data

R has many functions for statistical analyses and graphics. The results and graphs produced by R are displayed on the screen and can also be saved for later use. You can perform many statistical procedures in R by typing commands in the Console window or run the Scripts in the R Editor window. Some advanced functions need to be installed from the package before they can be executed. Please also keep in mind that the R language is case sensitive. It discriminates between uppercase and lowercase letters in the names of the objects.

Descriptive Statistics

To get quick descriptive statistics for all variables in the dataset, we can use the `summary` function:

```
> summary(airquality)
     Ozone          Solar.R            Wind            Temp           Month          Day
 Min.   :24.0   Min.   :118.0   Min.   : 7.400   Min.   :62.00   Min.   :5   Min.   :1.00
 1st Qu.:33.0   1st Qu.:141.2   1st Qu.: 7.850   1st Qu.:65.75   1st Qu.:5   1st Qu.:1.75
 Median :54.0   Median :169.5   Median : 9.750   Median :69.50   Median :5   Median :2.50
 Mean   :53.5   Mean   :192.5   Mean   : 9.875   Mean   :68.75   Mean   :5   Mean   :2.50
 3rd Qu.:74.5   3rd Qu.:220.8   3rd Qu.:11.775   3rd Qu.:72.50   3rd Qu.:5   3rd Qu.:3.25
 Max.   :82.0   Max.   :313.0   Max.   :12.600   Max.   :74.00   Max.   :5   Max.   :4.00
```

We can get individual descriptive statistics (mean, variance, standard deviation, minimum and maximum) of a single variable:

```
> mean(airquality$Solar.R)
[1] 192.5
> var(airquality$Solar.R)
[1] 7323
> sd(airquality$Solar.R)
[1] 85.57453
> min(airquality$Solar.R)
[1] 118
> max(airquality$Solar.R)
[1] 313
```

The `fivenum` function returns the minimum, 25th percentile, median, 75th percentile, and maximum of a variable:

```
> fivenum(airquality$Solar.R)
[1] 118.0 133.5 169.5 251.5 313.0
```

You can also use `summary` with a single variable and get the same information as from `fivenum` plus the mean:

```
> summary(airquality$Solar.R)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    7.0   115.8   205.0   185.9   258.8   334.0
```

The `cor` function will give the correlation matrix of all variables in the dataset (type

`cor(airquality)`to get the complete correlation matrix for the `airquality` dataset). We can also get the correlation matrix of the second, third, and fourth variables in the dataset:

```
> cor(airquality [c(2,3,4)])
           Solar.R         Wind        Temp
Solar.R  1.0000000 0.33913763 -0.92683069
Wind     0.3391376 1.00000000  0.03809167
Temp    -0.9268307 0.03809167  1.00000000
```

Regression

We use the command `lm(Y ~ X + Z)` to perform a linear regression, where Y is the dependent variable and X and Z are both independent variables. The summary statistics for the linear regression model displays the regression model with the residuals, estimated coefficients, and R-squared value.

```
> air.model <- lm(Wind~Temp+Ozone)     #Naming the regression model
                                       #air.model and specifying its
                                       #variables.
> summary(air.model)                   #This line tells R to give the
                                       #summary statistics for the
                                       #regression model air.model.

Call:
lm(formula = Wind ~ Temp + Ozone)

Residuals:
       1        2        3        4
 0.12792 -0.14183  0.06489 -0.05098

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 15.476496   1.572559   9.842   0.0645 .
Temp        -0.009930   0.022378  -0.444   0.7341
Ozone       -0.091940   0.004317 -21.297   0.0299 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2081 on 1 degrees of freedom
Multiple R-squared: 0.9978,    Adjusted R-squared: 0.9934
F-statistic: 227.1 on 2 and 1 DF,  p-value: 0.04687
```

If you would like to include interaction terms in the model (e.g., between independent variables X and Z, as shown), the command is `lm(Y ~ X * Z)`. By default, R includes the main effects (X and Z) as well as the interaction term (X*Z).

Creating Graphs

We use the commands `hist`, `boxplot`, and `plot(X,Y)` to create a histogram, box plot, and scatter plot, respectively. A useful function to display several graphs on one graphic sheet is `par`. For example, you can use `par(mfrow=c(2,2))` to put four graphs on one sheet.

```
> par(mfrow=c(2,2))
> hist(Wind, sub=1)
> hist(Temp, sub=2)
> boxplot(Wind,main='Wind',sub=3)
> plot(Wind,Temp,sub=4)
```
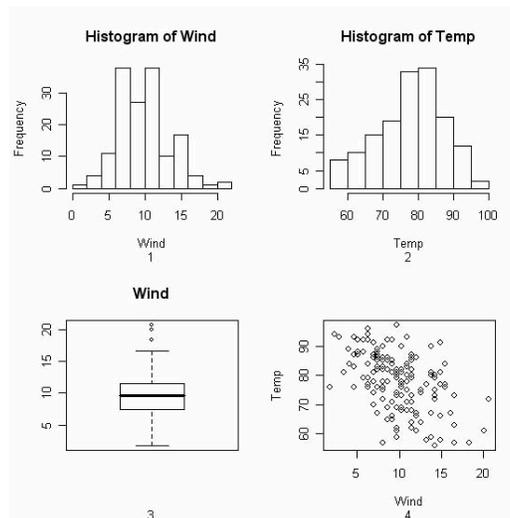

You may want to graph the scatter plot matrix for variables Solar.R, Wind and Temp.

```
> pairs(airquality [c(2,3,4)])
```
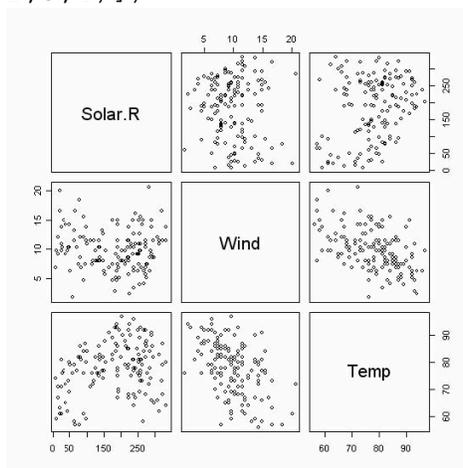

You may use the `plot` command with the name of the object in the brackets to plot 4 basic diagnostic graphs for your regression model: residual versus fitted plot, normal qqplots of the residuals, standardized residual versus fitted plot, and scatter plots with the regression line overlaid.

```
> par(mfrow=c(2,2))
> plot(model)
```
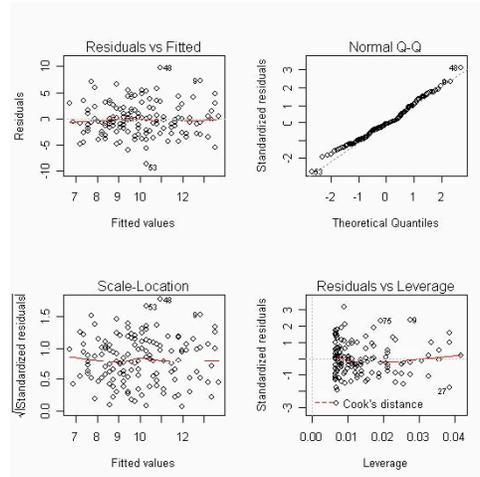

Saving Output Files and Graphs

To save your commands and output in the R console window, you may go to the "save to file" submenu of the "File" menu. Your file will be saved in .txt format and you can open it later. Or you could highlight the area you want in the console window, then copy and paste it into Word or Notepad for subsequent analysis. R displays the graphs in the R Graphics window, and you need to save the graphs separately from the text output. The graphs can be saved in various formats, such as jpg, png, bmp, ps, pdf, emf, pictex, x_g and so on.

For More Information and Assistance

Documentation and Books

You can read and photocopy R books and manuals in the Velma Denning Room (Green Library Bing Wing, room 120F) or purchase them at the Stanford Bookstore. Some R manuals are also available to be checked out from Green Library Reserves.

The SSDS Getting Started Guides page has a list of Resources for Learning R, including books and online resources. We include some highlights here.

You can download R manuals from the R Project's manuals page: http://cran.r-project.org/manuals.html.

R in a Nutshell by Joseph Adler is available in bookstores, and online to Stanford affiliates through http://searchworks.stanford.edu, and is an excellent source for beginners and intermediate users.

The following user-contributed documentation is helpful in learning R, all of which available as PDF files on http://cran.r-project.org/other-docs.html:

- *R for Beginners* by Emmanuel Paradis is good for new R users;
- *R Reference Card* by Jonathan Baron, for more advanced users;
- *R Reference Card* by Tom Short, for more advanced users.

The R Frequently Asked Question site also provides useful help.

> http://cran.r-project.org/doc/FAQ/R-FAQ.html

SSDS Software Services at Stanford

The software consultants at Social Science Data and Software (SSDS) provide technical support for SPSS users at Stanford. Users can view documents, access information about our drop-in hours, and submit questions from our web page at:

http://ssds.stanford.edu

Note: this document is based on R 2.9.2 for Windows, and R 2.10.0 for Macintosh.

Social Science Data and Software
Document revised: 9/21/2010