Getting Started with R and RStudio for Statistics

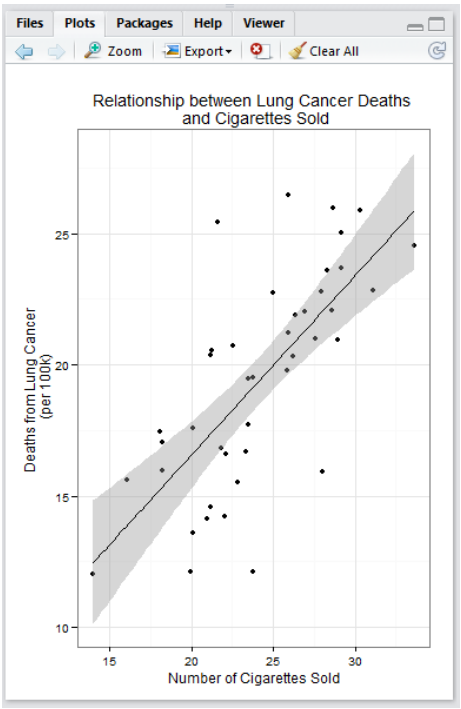
Dale Berger with Summer Clay, Nicole Gray, Minami Hattori,
Sarah Mason, Jieqi Jiang, and Cody Packard 1/19/2015

Goals:

After working through this guide, you will be able to

- Explain why R is important to learn
- Download, install, and explore R and R Studio
- Download analysis packages and understand what they are
- Create and import data sets from text files, Excel, and SPSS
- Run some basic analyses in R using RStudio
- Find help online and through RStudio

Contents:



Source: http://wise.cgu.edu Guides and Downloads

Why should I learn about R?

Your old friend Bumble asks: "I can calculate basic statistics using Microsoft Excel, and I know how to conduct more sophisticated statistical analyses using SPSS. Why should I learn R?"

1. It's FREE!!!
2. R is growing rapidly in all areas that use data analysis, including in social science.
3. R code is included in many journal articles and other publications.
4. R has incredible graphics capability, including 3D and interactive graphs.
5. Over 6000 analyses and graphics packages are available for free download.
6. You can do customized analyses with R, and even create and share your own packages.
7. Datasets from other statistical software can be imported into R.
8. There is a large international user community of helpful people.
9. After you gain experience with R, it becomes relatively easy to use.
10. R is listed as a desirable skill for an increasing number of jobs.

Install R

To download R, go to http://cran.r-project.org, the home base for R.

For Mac users, click "Download R for (Mac) OS X" and just accept all defaults.

For Windows users: Click on "Download R for Windows" > "base" > "Download R 3.x.x for Windows" Double-click the downloaded installer file. You can just accept all default settings though you may have some different preferences.

After the program is installed, an icon for R will appear on your desktop.

You can read more about the R project at http://www.r-project.org/ where you will find links to manuals, FAQs, etc.

When you open R, you are presented with the bare prompt **" > "** indicating it is your turn to do something. RStudio provides a friendlier interface, with pull-down menus, and separate windows for entering commands and showing output. For that reason, I recommend that you install RStudio and that you use it to interface with R.

Install RStudio

Download R Studio at http://RStudio.org.
Click "Download RStudio" > "RStudio Desktop" > "Download RStudio Desktop" > "RStudio xx.xx.xxx" (select Windows or Mac version as appropriate for your system)
Save the file, execute the file, and accept all default options.
An RStudio icon will appear on your desktop.

Introduction to RStudio

RStudio integrates the features of R. Open RStudio. It displays four panes, some of which have tabs and some may be minimized or hidden. Options to change the display can be found by clicking "Tools" > "Global Options" > "Pane Layout > then select viewer under Source.

The lower-left pane is the 'Console' where commands are entered to the R prompt > and some results are shown immediately. Other panes may be hidden, but the R console is always available.

The upper-left pane is a 'Source' window where syntax files and other information can be seen. R script (i.e., syntax) can be entered here.

The upper-right pane is the 'Workspace' that shows files that you have imported and a tab shows the history of commands you have used.

The lower-right pane shows files, with tabs to show plots that you generate, packages in your library, and help that offers ability to search.

Working Directory

Your working directory is the specific location on your computer where files available to R by default are stored. To find your working directory, enter the command **getwd()** to the > prompt**.** After you enter this command and press Enter, your working directory is shown immediately in the console window, following [1], and a new > prompt is shown.

Many commands in R can take 'arguments' that indicate specific information. Even when no arguments are specified, both left and right parentheses must be included with those commands.

```
> getwd()
[1] "C:/Users/Dale/Documents"
>
```

Here the command to get the working directory **getwd()** includes the left and right parentheses.

Important note: R is very fussy about capitalization. Thus, HELLO, Hello, and hello are three different objects in R. When you get an error message saying that an object is not found or that R can't do something, check to make sure you have used exactly the correct capitalization.

It is good to use a special directory for R, separate from our other documents. First, create the directory that you want to use, and then give the **setwd** command to change to that directory. I created a directory called "C:Users/Dale/Documents/R" with Windows Explorer. Then I told R

```
> setwd("c:/Users/Dale/Documents/R")
Error in setwd("c:/Users/Dale/Documents/R") :
  cannot change working directory
```

Oh, oh! What happened? Really R, you can't do it??? Oops, I forgot to capitalize the C!

```
> setwd("C:/Users/Dale/Documents/R")
```

R doesn't tell me that it worked, but at least it didn't scold me. Let's check to see if it worked.

```
> getwd()
[1] "C:/Users/Dale/Documents/R"
```

Yes, now the working directory is the special R directory that I created.

You can set the working directory easily with RStudio: Session; Set Working Directory

As with SPSS, R uses data files and syntax files. Data files can be created in R, downloaded as R data files, or imported from SPSS, Excel, or many other formats. Syntax files can be saved for future use and reference. Let's begin with some simple R syntax.

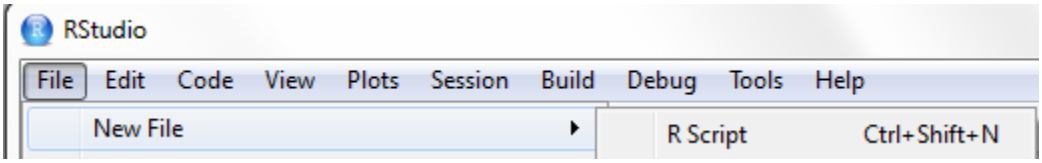
To assign a value to a variable, it is standard practice in R to use the symbols <− . For example, to assign the value of 4 to x, we give the command x <− 4 to the R console > prompt.

```
> x <- 4
```
(Nothing seemed to happen. However, now enter x and R will return the new value of x.)

```
> x
[1] 4
>
```

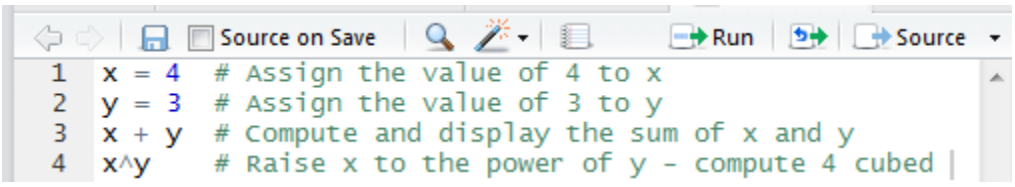
Syntax Files

When you enter commands into the R console one line at a time, some results may be displayed immediately. Previous commands can be found by using the up-arrow key, so they can be edited and resubmitted. However, it is often more efficient to create the commands in a separate syntax file that can be saved. To open a new R Script file in RStudio, click File, New File, and R Script, Enter. Many menu commands have keyboard equivalent commands – here Ctrl+Shift+N.



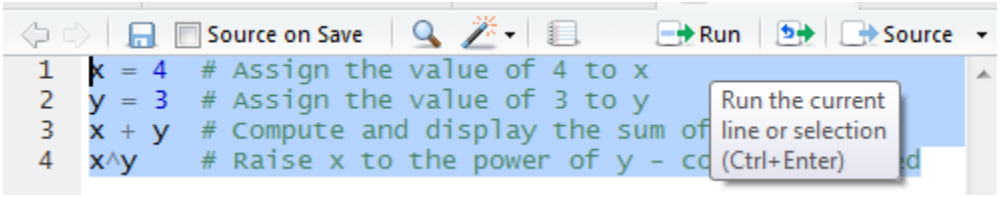
This opens a new syntax window in the top-left pane. Now you can enter several lines of code and run them all at once. The number sign (#) indicates a comment in R, and anything on a line following # will not be evaluated by R. It is important to use comments to document your code.

For illustration, I entered four lines of code with a lot of documentation in the syntax window:



Notice that I used an equal sign rather than the <− symbols. The equal sign could be confusing in a command like x = y. Quiz: What would happen if we gave the command x = y on step 5? Answer: R would assign the value of y to x, so both x and y would be equal to 3. To avoid ambiguity, most R practitioners use the <− symbols to assign values. Reverse assignment −> also works.

You can highlight (left-mouse drag) commands and press the icon to run the current selection, the first box with a green arrow. You can hover over the icons to see what they do.



Now in the R console we find the following:

```
> x = 4   # Assign the value of 4 to x
> y = 3   # Assign the value of 3 to y
> x + y   # Compute and display the sum of x and y
[1] 7
> x^y     # Raise x to the power of y - compute 4 cubed
[1] 64
>
```

Yay! It worked. We could save this syntax file as an .R file with Ctrl-S or with the Save icon.

Packages

Packages are collections of R resources that people have made for special purposes. Some packages come automatically with R, and you can install others for your own purposes. You can find a complete list of contributed packages at http://cran.r-project.org/web/packages/ . As of December 14, 2014, the repository contained 6107 available packages. Packages need to be installed only once, but when you begin a new R session, you must load the installed packages that you will use in that session. Some packages, including package:stats, are loaded by default.

```
library()    # see all installed packages listed in the Source pane
search()     # see packages currently loaded, including the defaults
```

Find, Install, and Load the psych Package

In the bottom-right panel of RStudio, select Packages; Click Install to open Install Packages; Repository (CRAN, CRANextra) is fine – you have the option of selecting a ZIP file but that is not needed for normal size packages. In Packages, enter **psych**. The default library location is fine. Check the Install dependencies box. Click Install.

The R console should show messages indicating that package '**psych'** successfully unpacked. Now click the Packages tab in the bottom right pane, and you should find psych is included in the User Library of installed packages.

If you enter > search**()** in the R console, you will see a list of loaded packages, but **psych** is not listed. You can load **psych** by checking the box next to **psych** in the list of packages shown under the Packages tab. Now when you enter > search(), the list of loaded packages will include "package:psych". An installed package (stored on your computer) must be loaded for R to use it.

You can click on **psych** in the Packages tab to find documentation for the psych package. The **psych** package includes code for things like scale construction and factor analysis.

Create Data by Typing Data into R

Let's begin by entering data directly into R, and then see how to import files from other programs like Excel, SPSS, etc.

Variables can be defined as individual values as we did in the Syntax Files section, like x = 4.

Data can also be entered as a string of scores, which we call a vector of scores. Suppose you have the body weight for five people. You can collect these five scores into a vector with the c() command and call it **Weight**. 'c' stands for 'collect' or maybe 'column' vector.

```
> Weight = c(185, 162, 118, 149, 149)  # This is similar to a column of data in SPSS.
```
Now if you simply enter `> Weight`, the data will be listed (it is a good idea to check).

You also can specify the height for these five people.

```
> Height = c(72, 70, 65, 65, 63)  # The height information is collected into a column vector.
```

Enter Height to the > prompt to see how Height is presented by R.

```
> Height
[1] 72 70 65 65 63
>
```

Now you can use these data to compute the Body Mass Index (BMI) for each case.

```
> BMI = Weight/(Height^2)*703  #You provide the name BMI and the formula.
```

Enter BMI to the > prompt to see the results.

```
> BMI
[1] 25.08777 20.37265 19.63408 23.12828 24.62006
>
```

You can set up a standard matrix of data with variables in columns and cases in rows by defining a 'data frame.' One way to do this is to combine vector data for individual variables from a set of cases. Here you have data for five cases on each of three variables, which you can combine into a standard 5x3 data file. You can name this data frame "BMIdata" or any other name.

```
> BMIdata = data.frame(Height,Weight,BMI) #Creates a standard 5x3 data file.
> BMIdata
  Height Weight      BMI
1     72    185 25.08777
2     70    142 20.37265
3     65    118 19.63408
4     65    139 23.12828
5     63    139 24.62006
>
```

Copy data from Excel

Here is an easy way to bring a small data set from Excel into R. The Excel file 'Smoking and Cancer.xlsx' was created from data downloaded from the DASL collection, accessed through http://wise.cgu.edu . http://lib.stat.cmu.edu/DASL/Stories/cigcancer.html. The data set includes cigarettes sold and rates of various types of cancer per 100,000 for DC and 43 states in 1960.

	A	B	C	D	E	F
1	State	Cigarettes	Bladder	Lung	Kidney	Leukemia
2	AK	30.34	3.46	25.88	4.32	4.90
3	AL	18.20	2.90	17.05	1.59	6.15
4	AR	18.24	2.99	15.98	2.02	6.94
5	AZ	25.82	3.52	19.80	2.75	6.61
6	CA	28.60	4.46	22.07	2.66	7.06
7	CT	31.10	5.11	22.83	3.35	7.20

Hold down the left mouse button and highlight the portion of the data set you wish to copy from Excel, including labels; right-click and select 'copy' to put the information into your clipboard. You can also use Ctrl-c when material is highlighted to place it into your clipboard.

With the Excel material in the clipboard, you can define a new data frame to be 'smoking' and tell R that the headers (variable names) are included in the data file with header = TRUE.

```
> smoking = read.table("clipboard",header=TRUE)
> smoking
   State Cigarettes Bladder  Lung Kidney Leukemia
1     AK       30.34    3.46 25.88   4.32     4.90
2     AL       18.20    2.90 17.05   1.59     6.15
3     AR       18.24    2.99 15.98   2.02     6.94
4     AZ       25.82    3.52 19.80   2.75     6.61
5     CA       28.60    4.46 22.07   2.66     7.06
6     CT       31.10    5.11 22.83   3.35     7.20
7     DC       40.46    5.60 27.27   3.13     7.08
```

Mac users may need to install and load the package Kmisc and then use the following syntax:
> smoking=read.cb (header=T)

For more information, see: http://stackoverflow.com/questions/14547069/how-to-write-from-r-to-the-clipboard-on-a-mac/14547293#14547293

Variable names, attaching and detaching files

Great – the data set was read into R successfully. Now you might think that you could refer to a variable by name to use that variable. Let's ask to see a list of the states.

```
> State
Error: object 'State' not found
```

Nuts! R can't find State. Why not? R can have several data files open at the same time, so there is potential ambiguity regarding which data set we are referring to.

You can specify a variable that you want by including the name of the data set with the variable name: DataSet$VariableName.

```
> smoking$State
 [1] AK AL AR AZ CA CT DC DE FL ID IL IN IO KS KY LA MA MD
[19] ME MI MN MO MS MT NB ND NE NJ NM NY OH OK PE RI SC SD
[37] TE TX UT VT WA WI WV WY
44 Levels: AK AL AR AZ CA CT DC DE FL ID IL IN IO ... WY
```

Now you can see which states are actually in the data set. Hmm – we are missing PA but we have PE??? There is much more exploring to do at another time.

If you are working with one specific data set, you can avoid the need to name the data set each time you refer to a variable, by 'attaching' the data set to the search path.

```
> attach(smoking)
> State
 [1] AK AL AR AZ CA CT DC DE FL ID IL IN IO KS KY LA MA MD
[19] ME MI MN MO MS MT NB ND NE NJ NM NY OH OK PE RI SC SD
[37] TE TX UT VT WA WI WV WY
44 Levels: AK AL AR AZ CA CT DC DE FL ID IL IN IO ... WY
>
```

Now the simple command > State works as expected

```
> detach(smoking)
>
```

When you are finished using an attached data set, be sure to detach it. If you don't use the detach command, you may have more than one copy of a file installed. Many experienced R users avoid the attach command.

You can see all of the installed packages and data sets with the command > search() .

A data file can be edited by entering >edit(filename)
Leave the editor by clicking the X at the top right, and edits are saved.

Import delimited files, e.g., Comma Separated Variables (CSV), or tab delimited

A delimited file uses a character such as a comma to separate successive data values. A generic command to read such files is as follows:

```
Myfile = read.delim("DataFileName.txt", header = TRUE, sep = "")
```

This command will create a file called Myfile by importing data from a file called DataFileName.txt. It will read the first line of data as a header that contains variable names. The sep command identifies the separator between variables. The command sep = "" indicates white space separates variables, either blanks or tabs.

sep = "\t" indicates tab separators, sep = "," for comma separated variables, etc.

Here is an important trick for importing files. Rather than providing the full name for the file enclosed in quotes, you can ask RStudio to open a window that will allow you to simply select the file that you want. The command is file.choose()

To demonstrate, I created a small text file called TabDelDemo.txt with three variables (Y, X1, and X2) where the numbers are separated by tabs, and it ends with an 'Enter' key press.

```
> TabsDemo = read.delim(file.choose(), header=T, sep = "\t")
> TabsDemo
   Y X1 X2
1 12  3  9
2 19  7 13
3 11  4 10
4 27  9 18
>
```

When the first command was executed, RStudio opened Windows Explorer, so I could search through my directories to find the file TabDelDemo.txt and simply select it.

If you enter the full name manually, there is a pitfall to avoid – R misinterprets backward slashes (\) in file names. You can avoid the problem by replacing backward slashes with forward slashes (/) or double backslashes (\\). This is a bit annoying. Let me demonstrate what happens if we don't deal with the slashes.

```
> Tabs = read.delim("C:\Users\Dale\Documents\Classes\MR15\R\TabDelDemo.txt",header = TRUE, sep = "\t")
Error: '\U' used without hex digits in character string starting ""C:\U"
> Tabs = read.delim("C:\\Users\Dale\Documents\Classes\MR15\R\TabDelDemo.txt",header = TRUE, sep = "\t")
Error: '\D' is an unrecognized escape in character string starting ""C:\\Users\D"
> Tabs = read.delim("C:\\Users\\Dale\\Documents\\Classes\\MR15\\R\\TabDelDemo.txt",header=TRUE, sep = "\t")
> Tabs
   Y X1 X2
1 12  3  9
2 19  7 13
3 11  4 10
4 27  9 18
>
```

The first error was because R was confused by "**\U**". When I replaced "**\U**" with "**\\U**" then R stopped at the next occurrence of a single back slash, "**\D**". When I replaced all \ with \\, it ran with no error message.

To check that the table was read correctly, I entered the name of the table(> **Tabs**), and then R displayed the table in all of its glory.

Note that I could have replaced all of the back slashes (\ ) with forward slashes (/) instead.

Import a .dat file

Howell provided a data file on average course evaluations for 50 courses, along with information on various characteristics of each course. This file is saved in .dat format, Tab15-1.dat. We can read this table into R using the read.table command. To learn more about this command, you can enter >help(read.table) or >?read.table. Help files vary in usefulness.

If you copy the Tab15-1.dat file into your active R directory, you can refer to the file with its name alone. Else, you need to include the full address for the file as I have done here. I located the file and right-clicked it, selected Properties, and copied the file location and added the file name. I named the file CourseEval and changed the backslashes (\) to forward slashes (/).
I could have used the file.choose() command instead of the file name (that is a lot easier).

```
CourseEval = read.table("C:/Users/Dale/Documents/Classes/MR15/Computer/Tab15-1.dat",header=FALSE)
```

When I entered the command > CourseEval, R listed 50 cases followed by rows of NA for cases 51 through 397. NA indicates 'not available' which means missing.

There are at least two ways to fix this. First, when I opened the original .dat file and highlighted down the file, I saw that the file includes a lot of blank lines at the end. You could simply delete those blank fields, save the edited file, and re-import into R. That is what I would recommend – there is no reason to have all those blank fields in the file.

```
48  3.4  3.4  3.6  3.3  3.3  40
49  4.0  4.2  4.0  4.4  4.1  18
50  3.5  3.4  3.9  4.4  3.3  90
51   NA   NA   NA   NA   NA  NA
52   NA   NA   NA   NA   NA  NA
53   NA   NA   NA   NA   NA  NA
54   NA   NA   NA   NA   NA  NA
```

A second approach is to deal with the problem by using R. You can ask R to retain only a subset of the cases with the command > subset(file.name, condition). Here you could ask R to save only the subset of cases where the value on V1 is greater than zero.

```
> CourseEval = subset(CourseEval,V1>0)
> CourseEval
    V1  V2  V3  V4  V5  V6
1  3.4 3.8 3.8 4.5 3.5  21
2  2.9 2.8 3.2 3.8 3.2  50
3  2.6 2.2 1.9 3.9 2.8 800
```

Now when we enter the command > CourseEval, the data file is shown for only 50 cases.

The .dat file did not provide names for our variables, so R just called them V1, V2, etc. You can ask R to add names for the variables. The variables, in order, are Overall = overall quality of lectures, Teach = teaching skills of instructor, Exam = quality of exams, Knowledge = perceived knowledge of instructor, Grade = expected grade, and Enroll = number of students in the class.

```
> names(CourseEval) <- c("Overall","Teach","Exam","Knowledge","Grade","Enroll")
> CourseEval
   Overall Teach Exam Knowledge Grade Enroll
1      3.4   3.8  3.8       4.5   3.5     21
2      2.9   2.8  3.2       3.8   3.2     50
3      2.6   2.2  1.9       3.9   2.8    800
4      3.8   3.5  3.5       4.1   3.3    221
```

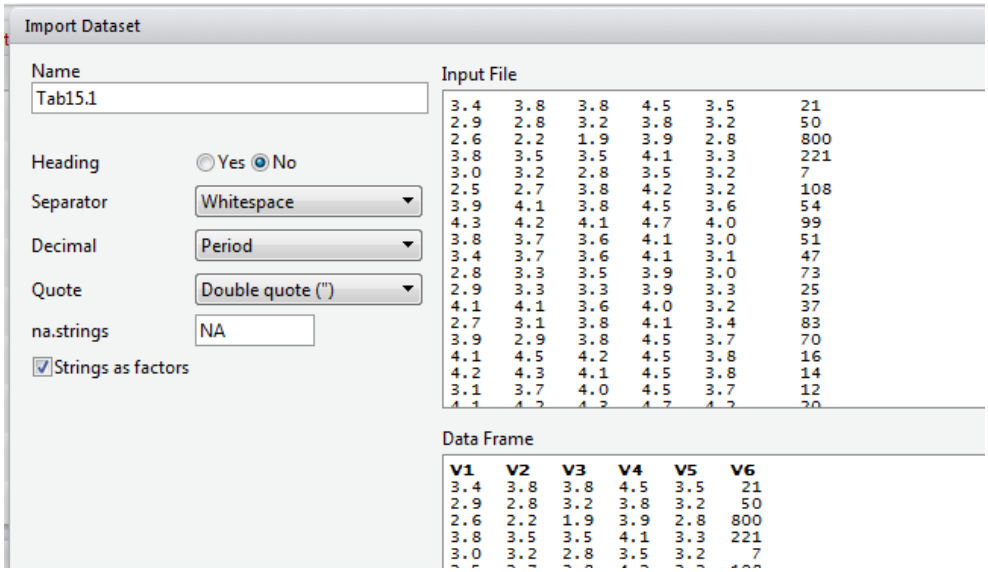
You can see a list of all variable names with the command names(myfilename).

```
> names(CourseEval)
[1] "Overall"   "Teach"     "Exam"      "Knowledge" "Grade"     "Enroll"
```

Mac users can convert .dat files created by Windows by going into Terminal and typing:
cat name of file you can't read | col -b >new name of file
For more information:
https://schmeits.wordpress.com/2010/08/26/dos2unix-alternative-those-darn-m-characters/

Import a Text File Using RStudio

RStudio has a handy wizard for importing text files. In the upper right pane, select the Environment tab, click Import Dataset. Select "From Text File…" and browse to find the text file you wish to import. For illustration, I selected Tab15-1.dat from Howell, which R calls Tab15.1. You can change the name if you wish by highlighting and renaming.



Import an Excel file using RStudio

For illustration, I will use a tiny file that Tabachnick and Fidell (5th ed., p. 617) used to demonstrate factor analysis.

Create a simple Excel data file with 5 columns and 6 rows including a header row with labels, as shown here.

	A	B	C	D	E
1	Skiers	Cost	Lift	Depth	Powder
2	1	32	64	65	67
3	2	61	37	62	65
4	3	59	40	45	43
5	4	36	62	34	35
6	5	62	46	43	40

**Save this Excel file as SkiersCSV.csv using the Save as type: CSV (Comma Separated Variables) or 'comma delimited' (comma separated variables).

In RStudio, top-right panel, select Import Dataset, From Text File…, select SkiersCSV , open. You will see the input file and also the Data Frame file as it is used in R. Check Yes for Heading, Separator is Comma, Decimal is Period, Quote is Double quote ("), click Import.
The data file will appear in the top-left panel of RStudio.
Verify that R is ready to use the file by entering > **SkiersCSV**

You can edit the data file by entering > **edit(SkiersCSV)** [Click X to exit; edits are retained]

Import an SPSS .SAV file

In most cases importing from .txt will be easier. However, reading an SPSS table into R allows the data frame to gain the attribute variable.labels, which provides a description of variable names in a manner similar to the actual SPSS file.

To read in an SPSS file, you need to load the **foreign** package. The **foreign** package is a default package in R, but for whatever reason, the package is not automatically included in a user's library. Thus, there is no need to install the package, but you do need to load the package. The file.choose() command instead of the file name works with read.spss, too.

```
> # load foreign package.
> library(foreign)
Warning message:
package 'foreign' was built under R version 3.1.2
>
> # import dataset using read.spss() function. Set the argument to.data.frame as TRUE
> # You will get an error message that doesn't seem to affect the data... yet!
> ClassEval <- read.spss("C:\\Users\\Dale\\Documents\\Classes\\MR15\\R\\Tab15-1.sav",
to.data.frame = TRUE)
Warning message:
In read.spss("C:\\Users\\Dale\\Documents\\Classes\\MR15\\R\\Tab15-1.sav",  :
  C:\Users\Dale\Documents\Classes\MR15\R\Tab15-1.sav: Unrecognized record type 7, sub
type 18 encountered in system file
>
> # view data frame
> View(ClassEval)
```

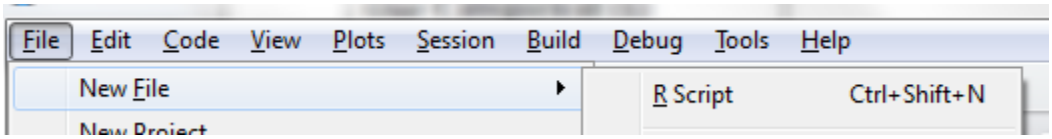
The warning messages can be ignored. Note that when I used the full location name for the SPSS data file, I needed to change the single back slashes to double back slashes.

The "View" command asks RStudio to show the data file in the top-left pane, so we can verify that the data were loaded appropriately. (Note that "View" is capitalized.) For more information on the data file, use the command > attributes(filename).

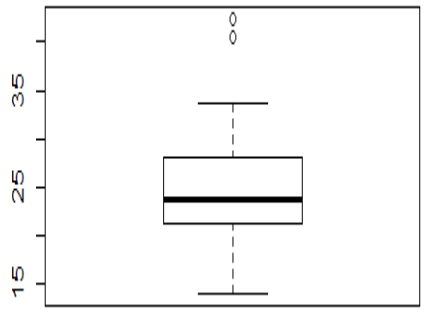
Basic Data Analysis Commands

Here we will demonstrate some basic commands using the smoking data set. Is there a relationship between number of cigarettes sold and lung cancer rates in these states?

I'll create a new syntax file by clicking File, New File, R Script. This opens a blank screen in the upper left pane.




```
> attach(smoking)   #Allow naming variables without file name
> mean(Cigarettes)  #Gives the mean
[1] 24.91409
> median(Cigarettes)
[1] 23.765
> sd(Cigarettes)    #Gives the standard deviation
[1] 5.573286
> summary(Cigarettes) # 5 number summary plus mean
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  14.00   21.23   23.76   24.91   28.10   42.40
> boxplot(Cigarettes) # Look at the data!
```



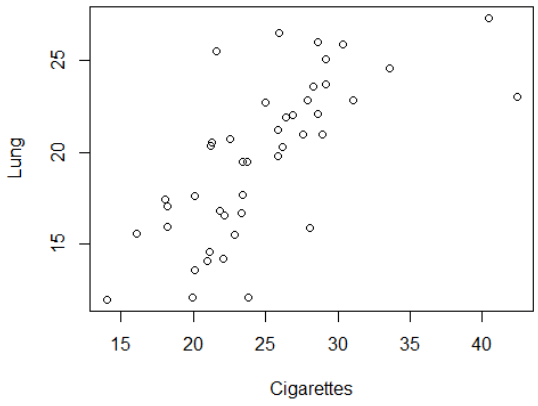
When the syntax file is run, most results appear in the console, but the boxplot appears in the bottom-right pane of RStudio. Notice that the box plot shows two outliers.

An important next step would be to track down those two outlier cases to see what is going on. (They are NV and DC, places where many nonresidents buy cigarettes, so the per capita sales is inflated. This suggests that sales may be underreported in other states, especially neighboring states.)

```
> cor(Cigarettes,Lung)
[1] 0.6974025
>
```

There is a strong correlation between number corof cigarettes sold and lung cancer rates across these states. Let's take a look at the plot.

```
> plot(Cigarettes, Lung)
>
```



Those two outliers appear to affect the correlation. Let's see what you get when those two cases are removed.

Selecting a subset of data for analysis

You can remove the two outliers by creating a new data set that is a subset of the original data set where Cigarettes is less than 35. I called the reduced data set smoking42 (42 cases remain).

```
> smoking42 = subset(smoking, Cigarettes < 35)  #Creates a new data set with outliers removed
>detach(smoking)    #Good idea to keep only one file attached at a time
>attach(smoking42)  # We need to tell R to use the new data set – R will give some warnings.
```

```
> cor(Cigarettes, Lung)
[1] 0.71448
>
```

The correlation is now a bit larger. The plot appears in the bottom right pane of RStudio when you open the Plot tab.

The tests of statistical significance are not reported automatically.
R is an introvert, and will answer when asked, but won't volunteer much.

```
# Another way to select a subset of data for further analyses
>newdata = olddata[, 2:12]  #selects all rows and columns 2 through 12
                    [1:40,] would select all columns for the first 40 rows.
```

Summary function

The summary() function generates information about variables, data sets, models, etc. For a single variable, summary(variablename) generates a 5-number summary plus mean.

```
> summary(Cigarettes)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  14.00   21.19   23.60   24.13   27.82   33.60
```

The summary() function can be applied to a data file, as > summary(smoking42) to request the five number summary for all variables in the active file (not shown here).

Regression

lm(Y~X) creates a linear model predicting lung cancer rates (Y) from Cigarette sales (X); I named the model Regression1.

There is no output until you ask for it.

When summary() is applied to a regression model, we are given information about the residuals along with a summary of the model and statistical tests.

```
> Regression1 = lm(Lung~Cigarettes)
> summary(Regression1)

Call:
lm(formula = Lung ~ Cigarettes)

Residuals:
    Min      1Q  Median      3Q     Max
-7.0440 -1.4035  0.4473  1.7062  7.7984

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.9138     2.5907   1.125    0.267
Cigarettes    0.6829     0.1057   6.459 1.07e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.919 on 40 degrees of freedom
Multiple R-squared:  0.5105, Adjusted R-squared:  0.4982
F-statistic: 41.71 on 1 and 40 DF,  p-value: 1.072e-07
```

Describe Function

Another useful function to describe numeric data is the describe() function.

```
> describe(smoking42)
           vars  n  mean    sd median trimmed   mad   min   max range  skew kurtosis   se
State*        1 42 22.76 12.91  22.50   22.82 16.31  1.00 44.00 43.00 -0.02    -1.29 1.99
Cigarettes    2 42 24.13  4.31  23.59   24.20  4.53 14.00 33.60 19.60 -0.09    -0.57 0.67
Bladder       3 42  4.03  0.88   4.04    4.00  1.14  2.86  5.98  3.12  0.16    -1.26 0.14
Lung          4 42 19.39  4.12  20.05   19.44  4.86 12.01 26.48 14.47 -0.09    -1.09 0.64
Kidney        5 42  2.79  0.53   2.83    2.78  0.48  1.59  4.32  2.73  0.16     0.42 0.08
Leukemia      6 42  6.83  0.65   6.86    6.84  0.58  4.90  8.28  3.38 -0.29     0.61 0.10
```

Note that "se" at the end is the standard error of the mean, not the standard error for kurtosis. Also, note that statistics are provided for the non-numeric string variable State. R apparently coded State with sequential numbers 1 to 44 (with two missing) and treated those numbers as a variable.

Graphing using *ggplot2*

R comes with a basic graphs function called plot() which is helpful, but somewhat functionally and aesthetically limited compared to the graphing capabilities in the the *ggplot2* package. The *ggplot2* package was developed by Hadley Wickham and is based on Tufte's graphing recommendations and Wilkinson's (2005) grammar of graphics. The functions in the *ggplot2* package can take some time to learn, but the main concept behind *ggplot2* is that a graph is a series of layers with each layer composed of some geometric element that can be aesthetically modified.

The *ggplot2* package needs to be both installed into your library and loaded for use.

> In RStudio, bottom-right pane, click Install, and in the Packages slot enter ggplot2, click Install. It will take a few seconds to install. Now click the Packages tab and check the box next to ggplot2 to load it.

Building a graph layer by layer requires creating an object that stores the plot using the ggplot() function. The most basic arguments for the ggplot() function are specifying the data set and the variables used for the x- and y-axes. In general, the function aes() is a way to specify the aesthetic modifications of your graph for each element layer, of which the plot object is the first layer. The aes() function as an argument allows you to specify the x- and y-axes.

You can change the relative size of the X and Y axes by changing the window size for Plots in RStudio (stretch or shrink the entire RStudio display in one direction or the other).

ggplot2 offers many sophisticated options, only a tiny bit is shown here.

[The following example uses an earlier version of the smoking file where the variable Cigarettes was called Cig.]

First create a plot object.

```
# create plot object.
my.plot.object = ggplot(smoking, aes(Cigarettes, Lung))
```

Next add a layer specifying which type of graph you would like to return. In this case, let's make a scatter plot; the function geom_point() can be used to add points to your plot object.

Suppose you would like to identify only the outliers with their state names.

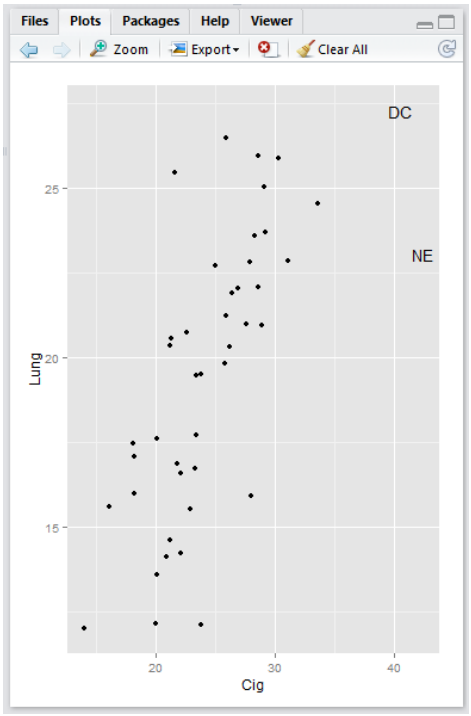
```
# create scatter plot in which only outliers are labeled.
my.outliers.plot = my.plot.object + geom_point(data =
subset(smoking, Cigarettes < 35)) + geom_text(data =
subset(smoking, Cigarettes > 35), aes(label = State))
my.outliers.plot
```

To save the graph, use the "export" button in the plot window of R-Studio or write the instructions at the command line:

```
ggsave("filename")
```

You can explore applications of ggplot by using >help(ggplot2), or use Google, YouTube, …

R code for many graph applications
https://www.stat.auckland.ac.nz/~paul/RG2e/



Correlations

Is the relationship between cigarettes sold and lung cancer rates statistically significant?
R offers several built-in options for computing correlations: cor(), cor.test(), and rcorr().
Here is an example using cor.test() using the full names for variables. If smoking42 is attached, I could use the short names of the variables: cor.test(Cigarettes, Lung) .

```
> cor.test(smoking42$Cigarettes, smoking42$Lung)

 Pearson's product-moment correlation

data:  smoking42$Cigarettes and smoking42$Lung
t = 6.4586, df = 40, p-value = 1.072e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.5244301 0.8367160
sample estimates:
    cor
0.71448
```

Let's compute the matrix of correlations between all variables (except State)

```
> # run correlations on all numeric variable from data frame and view as table.
> View(cor(subset(smoking42, select = Cigarettes:Leukemia)))
```

The table appears in the top-left pane:

	row.names	Cigarettes	Bladder	Lung	Kidney	Leukemia
1	Cigarettes	1.0000000	0.6076264	0.7144800	0.5790799	-0.1010087
2	Bladder	0.6076264	1.0000000	0.6404900	0.3707462	0.1832207
3	Lung	0.7144800	0.6404900	1.0000000	0.2667645	-0.1722790
4	Kidney	0.5790799	0.3707462	0.2667645	1.0000000	0.1848013
5	Leukemia	-0.1010087	0.1832207	-0.1722790	0.1848013	1.0000000

The correlations indicate that the number of cigarettes sold per capita show a positive linear relationship with per capita rates of bladder cancer, lung cancer, and kidney cancer, but a near zero correlation with leukemia. We can test the correlation between Cigarettes and Leukemia with cor.test(); the 95% confidence interval for that correlation ranges from -.40 to +.21.

```
> cor.test(Cigarettes, Leukemia)

 Pearson's product-moment correlation

data:  Cigarettes and Leukemia
t = -0.6421, df = 40, p-value = 0.5245
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.3928788  0.2093496
sample estimates:
       cor
-0.1010087
```

Missing data

Tips for dealing with missing data from http://www.statmethods.net/input/missingdata.html

```
# recode 99 to missing for variable v1
# select rows where v1 is equal to 99, and recode 99 to NA
mydata$v1[mydata$v1==99] <- NA

# create a new dataset without missing data
newdata <- na.omit(mydata)

# compute and show the total number of missing cases for a single variable
sum(is.na(mydata$v1))

# view percentage of missing cases for a single variable
# if a case is missing, is.na() is equal to 1 (i.e., TRUE)
# if not missing, is.na() is equal to 0
mean(is.na(mydata$v1))*100
```

Managing rows or columns from a data frame

It is very easy to select or remove whole rows and columns from the data set. Let's assume we are working with a data frame named mydata:

```
# select a single row, e.g. row 2
newdata <- mydata[2,]

# select multiple rows, e.g. rows 2, 4, and 6
newdata <- mydata[c(2,4,6),]

# select a sequence of rows, e.g. rows 2 through 4
Newdata <- mydata[c(2:4),]

# remove a single row, e.g. row 2
newdata <- mydata[-2,]

# remove several rows, e.g. rows 2, 4, and 6
newdata <- mydata[-c(2, 4, 6),]

# remove a sequence of rows, e.g. rows 2 through 4
Newdata <- mydata[-c(2:4),]

# select a single column, e.g. column 2
newdata <- mydata[,2]

# select multiple columns, e.g. columns 2, 4, and 6
newdata <- mydata[,c(2, 4, 6)]

# select a sequence of rows, e.g. rows 2 through 4
Newdata <- mydata[,c(2:4)]

# remove a single column, e.g. column 2
newdata <- mydata[,-2]

# remove multiple columns, e.g. columns 2, 4, and 6
newdata <- mydata[,-c(2, 4, 6)]

# remove a sequence of rows, e.g. rows 2 through 4
Newdata <- mydata[,-c(2:4)]
```

The subset() function is also useful. UCLA has a very nice tutorial on the subset() function here: http://www.ats.ucla.edu/stat/r/faq/subset_R.htm.

Multiple Regression

How well can we predict cigarettes sold using all four of the cancer measures? The linear model lm() function is used to fit linear models. We can call the model anything we want, such as Regression2.

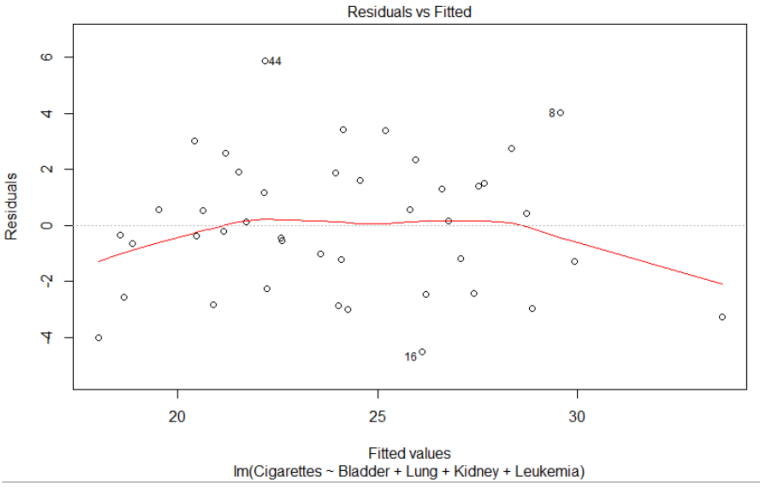
After running Regression2 = lm(data=filename, dv ~iv1 + iv2 + iv3), R doesn't show anything until we ask to see what it has done. The command summary() works well (note that summary is not Summary).

```
> Regression2 = lm(data=smoking42, Cigarettes ~ Bladder + Lung + Kidney + Leukemia)
> summary(Regression2)

Call:
lm(formula = Cigarettes ~ Bladder + Lung + Kidney + Leukemia,
    data = smoking42)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5293 -2.0267 -0.0444  1.5926  5.8630

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   7.5896     5.0320   1.508 0.139980
Bladder       0.8795     0.6427   1.368 0.179433
Lung          0.4892     0.1347   3.632 0.000848 ***
Kidney        3.3577     0.8082   4.154 0.000185 ***
Leukemia     -0.8557     0.6701  -1.277 0.209567
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.509 on 37 degrees of freedom
Multiple R-squared:  0.6945, Adjusted R-squared:  0.6615
F-statistic: 21.03 on 4 and 37 DF,  p-value: 4.114e-09
```

Here we see the regression coefficients shown as 'Estimate.' Both Lung and Kidney have unique contributions beyond all of the other variables. The multiple correlation squared is .6945, with p = .000000004114, which we could report as p < .001.

Diagnostic plots for regression model

```
> plot(Regression2)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
```



The plot() command applied to a regression model generates a series of diagnostic plots, including residuals as a function of predicted scores.There is a hint of curvilinearity in the plot, but notice that the dip at the right end is driven by only one data point.

HELP!

You can get information on any built-in R command by typing ? followed by the function, or with > help(function.name). >example(function) gives an example of using the function.

For functions in non-default packages, you can use >??function to search the web.
You can also use ??PackageName to find help on a package.

Within RStudio, there is a Help tab in the bottom-right pane.

Many basic statistical analyses are available in R by default, including univariate and bivariate descriptives and simple plots. Additional packages are available for very sophisticated and specialized analyses. If you want a specialized analysis, someone may already have written the code – your challenge is to find it. The home site for R http://cran.r-project.org is a good place to begin the search.

Tips to Avoid Sources of Frustration

Remember to back up your data and important syntax files.
Be liberal with documentation.
R pays attention to capitalization: `Hi, HI, Hl, H1,` and `hi` are all different objects in R.
Also the number 1 is different from letter l; number 0 is different from letter O.
Files must be active to be accessed. You can use > attach(filename) and >detach(filename).
If you use attach(filename), be sure to use detach(filename) when you are finished.
R gets confused by / in file addresses; you can use // or \ instead.
file.choose() instead of the file name opens a menu so you can search for the file. This avoids the problem with \ vs. /.
Single quotes can be used instead of double quotes.
"=" can be used instead of <- (I find = much easier to enter.)
R is an extreme introvert and won't tell you what it does unless you ask.
You can ask for an example of how a function is used with >example(function)

Please tell me about other frustrations you encounter dale.berger@cgu.edu

Common R Commands

```
getwd()                  #return the name of the current working directory
setwd("new directory")   #set the working directory to new directory
library()                #list all installed packages in the Source pane
search()                 #list packages currently loaded
library(packagename)     #load packagename
?function                #get help on a function
example(function)        #show an example of using the function

ls()                     #list the variables in the workspace
rm(x)                    #remove x from the workspace
attach(filename)         #makes the file active in the workspace
detach(filename)         #releases the file
view(filename)           #gives information on a file

myfile = read.delim("DataFileName.txt", header = TRUE, sep = "")
                         #imports a data file with labels separated by spaces
myfile = read.spss(file.choose(), to.data.frame = T)

newfile=oldfile[,-n]     #drop the nth column from oldfile
newfile=oldfile[-n,]     #drop the nth row from oldfile
newfile=subset(oldfile,logical) #select cases that meet the logical condition
  Example: Data2=subset(Data1,Age < 21)
newfile=oldfile[n1:n12,v5:v7] #select rows n1 thru n12 with vars v5 thru v7
newfile=oldfile[,v4:v25]  #select all rows with columns v4 through v25
names(filename) = c("Var1", "Var2", "Var3")  #gives names to variables
edit(filename)           #opens an editor where the file can be edited
                         #exit the editor by clicking the X on top right
varname                  #lists values on varname for all cases
filename                 #lists data in filename

summary(var1)            #min, Q1, median, mean, Q3, max
summary(filename)        #gives summary information on all variables
summary(modelname)       #summary of a model, such as a regression model
boxplot(var1)            #generates a box plot of var1
plot(var1,var2)          #generates a bivariate scatterplot
cor(var1,var2)           #generates correlation of var1 and var2
r.test(n,r)              #gives a p-value for correlation = r with n cases
Reg1 = lm(Y~X1+X2+X3)    #generates a linear regression model called Reg1
                          predicting Y from X1, X2, and X3
summary(Reg1)            #gives Reg1 model coefficients, tests of statistical
                          significance for coefficients and R squared
t.test()                 #see help(t.test) for more information
```

Let me know if you have good candidates for this list of common commands.

See Summaries of Common R Commands under Resources

Resources for Learning More about R

Written Materials for Self-Teaching the Basics of R

An Introduction to R (http://www.cran.r-project.org/doc/manuals/R-intro.pdf)
This .pdf document is supported by the R project. Appendix A has basic coding applications.

TryR by codeschool: http://tryr.codeschool.com/ Very gentle interactive introduction

http://swirlstats.com/students.html Learn R by using R and RStudio

R Tutorials by William King http://ww2.coastal.edu/kingw/statistics/R-tutorials/index.html

Introduction to R by David Armstrong (http://www.quantoid.net/ICPSRR.html)
These packets were used for a two-week workshop on R at ICPSR Summer Institute.

Introducing R by German Rodriquez http://data.princeton.edu/R/

R for Beginners by E. Paradis (http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf)
This textbook-like packet is a comprehensive introduction to basics of R, written for a beginner.

Kickstarting R (http://cran.r-project.org/doc/contrib/Lemon-kickstart/index.html)

Quick-R by Rob Kabacoff (http://www.statmethods.net/index.html)
Guides to many applications for R, including descriptive, hypothesis testing, and plotting

Video clips for Self-Teaching the Basics of R

Gordon David: A good series of introductory tutorials beginning with downloading R
https://www.youtube.com/watch?v=WJDrYUqNrHg&list=PL8BE0E317807A9A21

Mike Marin videos https://www.youtube.com/watch?v=qPk0YEKhqB8

Jeromy Anglim compiled links to introductory and advanced video tutorials on R
(http://jeromyanglim.blogspot.com/2010/05/videos-on-data-analysis-with-r.html).

Summaries of Common R Commands

See http://cran.r-project.org/doc/contrib/Short-refcard.pdf for R-Project summary

http://www.personality-project.org/r/r.commands.html Summary of common R commands

http://www.calvin.edu/~scofield/courses/m143/materials/RcmdsFromClass.pdf Stat examples

https://stat.duke.edu/courses/Spring14/sta101.001/UsersGuide.pdf

Let me know of especially useful resources for R that you find. Dale.Berger@cgu.edu
Google and YouTube are your friends!