A Beginners Guide to R Studio

Laura Boehm Vock --- January 2014 (Stat 272 version)

Getting Started

RStudio is a user interface for the statistical programming software R. While some operations can be done by pointing and clicking with the mouse, you will need to learn to write program code. This is like learning a new language- there is specific syntax, grammar and vocabulary, and it will take time to get used to. Learning R will ultimately give you complete control, flexibility, and creativity when analyzing and visualizing data... but fluency in this new language will take time. Be precise, go slow, copy other's code and be patient!

Dowload R and R Studio

R is a FREE software package for statistics and statistical graphics. It can be downloaded on UNIX, Windows, and Mac operating systems.

Go to http://cran.us.r-project.org/ and follow the instructions appropriate to your OS. In theory, you should just be able to run the .exe (Windows) or .pkg (Mac) file once downloaded and it will magically install. *I recommend using all the default settings when downloading R*.

Once you have downloaded R, you can install RStudio from http://www.rstudio.com/

Click "Download now", and then click "Download RStudio Desktop". Select and download the version appropriate for your operating system.

R Basics

R is an object based language – objects include matrices, vectors, data frames (a special type of matrix), and functions. All these objects float around in the **workspace**.

Many functions exist in "base R"- the basic set of functions and other objects available when you open RStudio for the first time. These include functions for plotting data, basic mathematical operations, and some statistical procedures which we will learn. Additional functions can be added by loading **packages**- these are sets of objects compiled by other R users and made publicly available. You can also write your own functions.

When you open RStudio you will see four windows:

Script

The script is a document to store a list of R commands. This window may not appear when you first open RStudio. To create a new script, Click "File → New → R Script."

Console

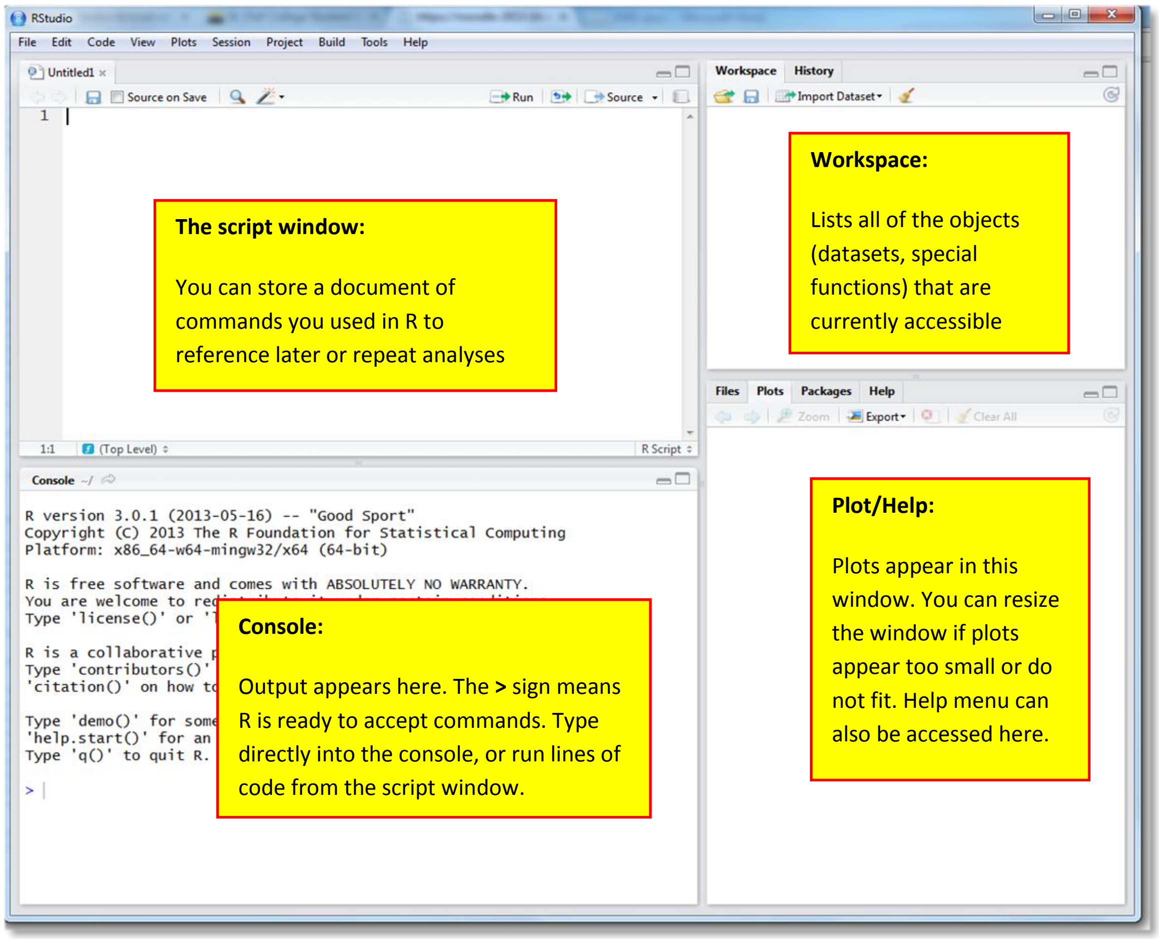
Output appears here. The **>** sign (also called the "prompt") means that R is ready to accept commands. You can type commands directly into the console. However, it is a good habit to instead type into the script window and run commands from there. **Nothing in the console can be saved.** You can however save your commands in a script file, and then repeat your analysis later. This is especially helpful if you are working on a big project or if you'd like to keep your code to refer back to later.

Workspace

This workspace window lists the objects currently available to you. Functions that are part of "base R" or packages will not appear here (there are just too many to make that practical!) Special functions that you write yourself or that are part of a previously saved workspace will appear here.

Plot/Help

The final window has several tabs, including a help tab with a search feature. When you create plots they will appear in this window, which you can resize to get a better view. The "Files" tab also shows you the files on your computer as one way to access R Scripts you have previously written. Be careful-deleting files in this window deletes them from your computer.



The script window:

You can store a document of commands you used in R to reference later or repeat analyses

Workspace:

Lists all of the objects (datasets, special functions) that are currently accessible

```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to red
Type 'license()' or '

R is a collaborative p
Type 'contributors()'
'citation()' on how to

Type 'demo()' for some
'help.start()' for an
Type 'q()' to quit R.

> |
```

Console:

Output appears here. The **>** sign means R is ready to accept commands. Type directly into the console, or run lines of code from the script window.

Plot/Help:

Plots appear in this window. You can resize the window if plots appear too small or do not fit. Help menu can also be accessed here.

Loading Workspaces and Datasets in RStudio

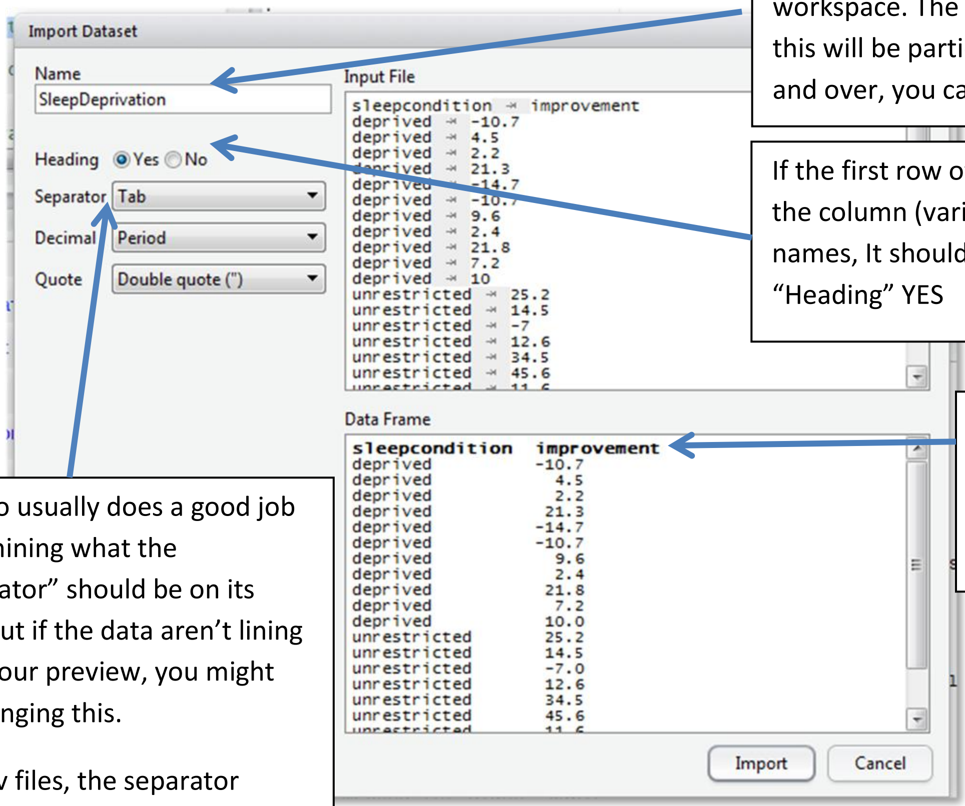
Workspaces are saved using the .RData file extension. A workspace is a convenient way to store multiple datasets or a set of functions you have written, particularly when running the code to produce the datasets may take a long time. To load a workspace, click the folder icon under the "Workspace" tab in the upper right RStudio window. Navigate to wherever you have saved the workspace and open it. You should now see a list of objects in the workspace window at right.

To load a **dataset** into your workspace, you need to click on the **Import data button** Import Dataset and select "From file" or "From URL" as appropriate. You can load csv or txt files that you have saved on your computer via "From file." When data appear as text files online, you may be able to load them directly from the URL.

Now you just need to be sure the preview of the Data Frame looks correct.

Name is what this dataset will be called in your workspace. The default will be the file name… if this will be particularly annoying to type over and over, you can change it here.

If the first row of the file is the column (variable) names, It should say "Heading" YES



Name: SleepDeprivation

Heading: [✓ Yes] No

Separator: Tab

Decimal: Period

Quote: Double quote (")

Input File

```
sleepcondition	improvement
deprived	-10.7
deprived	4.5
deprived	2.2
deprived	21.3
deprived	-14.7
deprived	-10.7
deprived	9.6
deprived	2.4
deprived	21.8
deprived	7.2
deprived	10
unrestricted	25.2
unrestricted	14.5
unrestricted	-7
unrestricted	12.6
unrestricted	34.5
unrestricted	45.6
```

Data Frame

```
sleepcondition improvement
deprived             -10.7
deprived               4.5
deprived               2.2
deprived              21.3
deprived             -14.7
deprived             -10.7
deprived               9.6
deprived               2.4
deprived              21.8
deprived               7.2
deprived              10.0
unrestricted          25.2
unrestricted          14.5
unrestricted          -7.0
unrestricted          12.6
unrestricted          34.5
unrestricted          45.6
```

Check that column (variable) names appear bolded. This ensures they will be treated as column names.

RStudio usually does a good job determining what the "Separator" should be on its own. But if the data aren't lining up in your preview, you might try changing this.

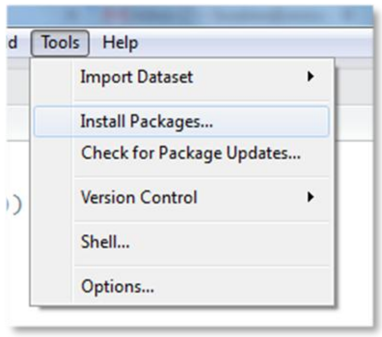
For .csv files, the separator should be comma. For .txt files, the correct separator is most likely tab or white space.

Once you import the dataset, a new data frame will appear in your workspace with whatever name was in the "Name" box.

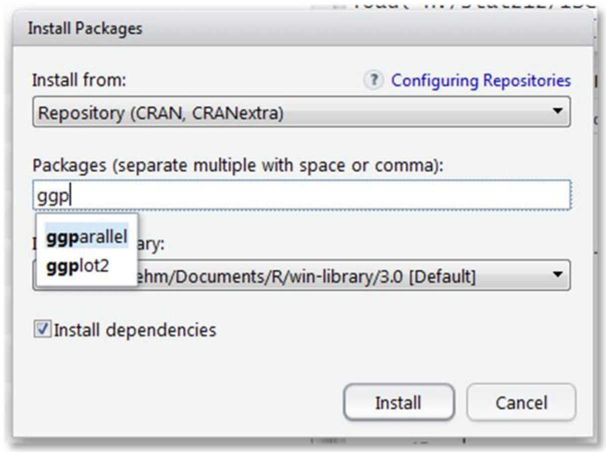
Packages

While many useful functions are included in "base R," users and developers can create and submit their own add-on packages with specialized functions and datasets. Accessing these packages requires two steps: installing the package onto your computer (only needs to be done once) and loading the library into your workspace (needs to be done *every time* you open RStudio). For example, many specialized plotting functions are included in `ggplot2`.

Packages can be installed by point and click in RStudio. First click on "Tools" and select "Install Packages"



Begin typing the package name. You can see auto complete options will appear. It is recommended to leave the "Install dependencies" box checked, as this will automatically install any packages required to run your desired package. Click "Install."



Install Packages

Install from: Configuring Repositories

Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma):

ggp

ggparallel
ggplot2

ary:

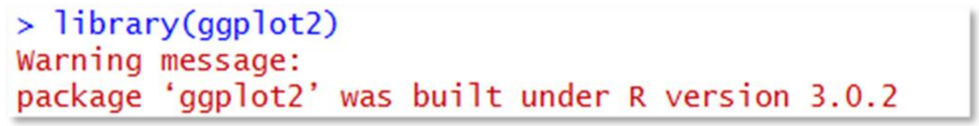
ehm/Documents/R/win-library/3.0 [Default]

☑ Install dependencies

Some warning messages may appear if the package was built under a different R version or if other packages are installed because of dependencies. Errors may occur at this step, so be sure to read the text which appears in red. When the package is successfully installed, you should see a message in the console which says:

```
package 'ggplot2' successfully unpacked and MD5 sums checked
```

To use the package, you must use function `library()` to load the desired package **every time you reopen RStudio.** Libraries will not automatically load whenever you use RStudio.



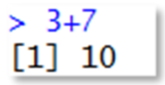
```
> library(ggplot2)
Warning message:
package ‘ggplot2’ was built under R version 3.0.2
```

Warning messages may appear. Usually these are safe to ignore, but with less commonly used or very outdated packages, check CRAN online to see if there are reported problems or if the package is still supported.

Basic Examples

R runs code line by line. That is, you tell it one thing, and it does it right away. (Sometimes if our one “line” of code is super long, it will actually be written as multiple lines on a page, but R treats it as one super-long sentence of code).

With numbers, we can use R like a calculator. The following is an example of what appears in the console window when we type 3 + 7 and hit enter.



```
> 3+7
[1] 10
```

Basic Math operators in R

+	3+7	$3+7=10$
-	3-7	$3-7=-4$
*	3*7	$3\times 7 = 21$
/	3/7	$3\div 7 = 0.42857$
^	7^3	$7^3=343$
sqrt	sqrt(3)	$\sqrt{3}=1.732051$
log	log(2)	NATURAL log: $\ln 2 = 0.693$
exp	exp(log(2))	$e^{\ln 2}=2$

We can also **store objects using names**. We see this most often in this class with named data frames. (aka data sets). We will also store tables, function output, or single values. A simple example is the following code:

```
se <- sqrt(.75*.25/200)
```

This is an example where I might want to store the standard error for a sample proportion of .75 with 200 observations as “se” in my workspace. This is convenient if I am going to be using it over and over in equations. You’ll notice that if you run this line of code, *no output appears in your console*. But a new “Value” appears in your workspace, called `se`. You can also use “=” to assign values rather than “<-“. The textbook tends to use “=” but many prefer to use the arrow as a convention; as you write more code, you will tend to develop your own style.

Note that **R is case sensitive.** The object `se` is not the same as SE.

Using Functions

Two functions we will commonly use in Stat 272 are `lm` and `plot`. The function lm is used for limple and multiple linear regression and takes many possible *arguments* or inputs, though we commonly only use two: the formula (model statement) and the dataset.

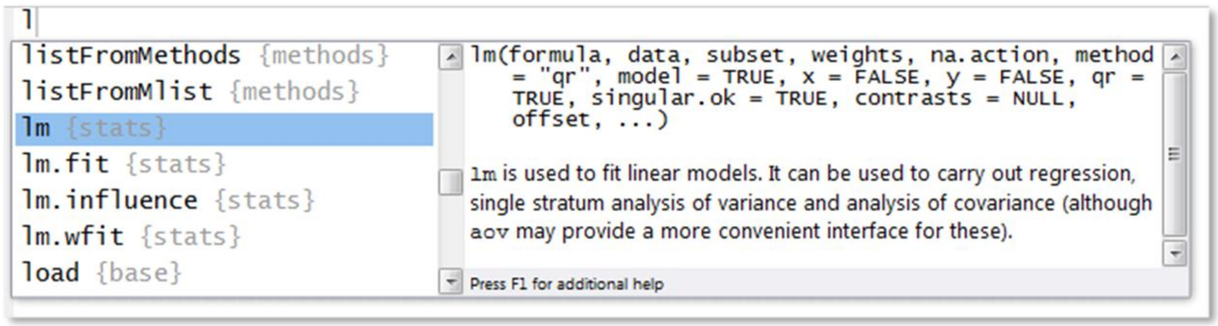
```
lm(formula, data)
```

When using R functions, you can specify which argument is which by name, or based on the order of entering information. For function inputs, you can also truncate the names. That is, all of the following will give equivalent results:

```
lm(formula= y~x, data=example1)
lm(form=y~x, dat=example1)
lm(y~x, data=example1)
```

Notice that the first argument y~x is assumed to be the formula. See the help menu or autocomplete hints for the expected order of arguments. If you use the argument names, the order does not matter. (This can be helpful for preventing errors).

Another useful feature of RStudio is **auto completion.** Try typing just "l" or "lm" in the script window and hit the **Tab** key.



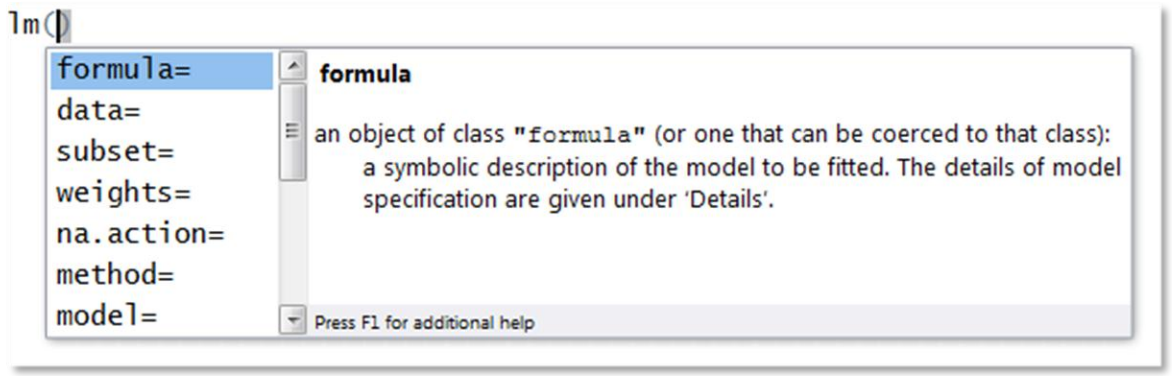
```
l
listFromMethods {methods}
listFromMlist {methods}
lm {stats}
lm.fit {stats}
lm.influence {stats}
lm.wfit {stats}
load {base}
```

```
lm(formula, data, subset, weights, na.action, method
    = "qr", model = TRUE, x = FALSE, y = FALSE, qr =
    TRUE, singular.ok = TRUE, contrasts = NULL,
    offset, ...)
```

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

A list of possible functions that begin with "l" will appear. Information about the selected function appears to the right.

You can also use auto completion when you do not remember the names of the function arguments. Hit tab while your cursor is within the parentheses.



```
lm()
formula=
data=
subset=
weights=
na.action=
method=
model=
```

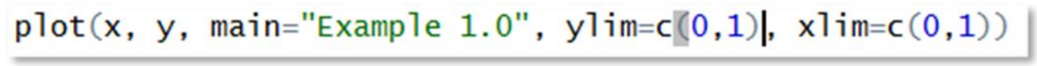
formula

an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.

Some function arguments are optional... as you can see, there are many options with the lm function, but we usually only specify the formula (model choice) and sometimes the dataset.

Once you have typed your complete line of code in the script window, run it by placing your cursor anywhere in the line, and hitting "**Ctrl+Enter**" or "**Ctrl+R**"

When you type into the script window, you will notice coloration of your inputs:



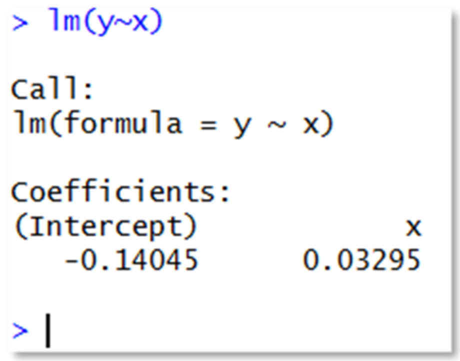
```
plot(x, y, main="Example 1.0", ylim=c(0,1), xlim=c(0,1))
```

This is *one big advantage of entering code in the script window* rather than directly into the console.

You'll notice several things:

Numbers are blue. Text (denoted by quotes) is green
Function, object and argument names are black.
Parentheses are gray. Closed parenthesis is automatically produced with the open parenthesis.
By placing my cursor by a parenthesis, the matching one is highlighted in grey.

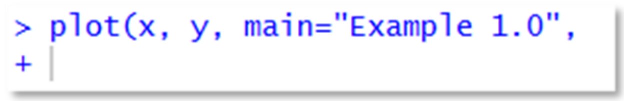
When you run a line of code, you will see the line and any output in the console window:



```
> lm(y~x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -0.14045      0.03295

> 
```

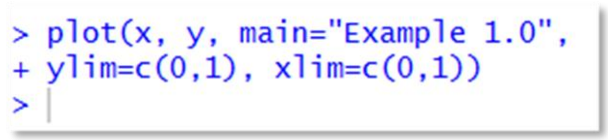
The blue line shows the command that was run, and the black is output from this function. Should you receive error messages, they will be red. Plots, if any, will appear in the plot window.

Notice that another blue > "prompt sign" has appeared beneath the output, indicating this command is completed, and R is ready to take another command. If a + appears, that indicates the line or "sentence" of code has not yet been completed (most likely a missing parenthesis) and R has not yet run the command.



```
> plot(x, y, main="Example 1.0",
+ 
```

Starting another line of code right away without finishing this command will probably result in an error. Code "sentences" can break over multiple lines; just don't forget to finish them. For example:



```
> plot(x, y, main="Example 1.0",
+ ylim=c(0,1), xlim=c(0,1))
> 
```

The plot function does not produce any output in the console but should produce plots in the plot window.

Writing Code in the Script Window

It can be really tempting to just type everything directly into the console- and if you are only running one or two lines of an analysis which you will never repeat, that can be ok. However, when doing homework and projects it will be essential to have a copy of the code you have run. I will frequently share R scripts with you which include examples. It is a good idea to keep these, and even add comments and notes of your own as we use them in class.

Advantages of writing code in the script window:

1. Auto-coloration and parentheses highlighting make errors easier to find.
2. You can save your code and write notes to yourself to reference later.
3. Makes it easy to share your code with classmates when working on projects, or to share with me if you have questions.
4. Makes your analysis **repeatable,** easy to edit and copy.

Scripts have the file extension ".R" If you don't have R installed on your computer, you can look at .R files using a text editor such as Notepad (although then you'll just see plain text, not colors). Plain text files (.txt) can also be opened in RStudio as script files.

One of the great things about script files is the ability to include **comments.** These are notes inserted along with the R commands that will not be run in R. Here is an example of a short script- the parts marked with # are comments.

This first part is called the "header"—it's a good idea to put something descriptive here. It can also be helpful if you're passing code back and forth when working on a project

```
# Example R Script
# Created by Laura Boehm Vock on Jan 24, 2014
# Last Edited by Laura Boehm Vock on Jan 28, 2014
################################################################################
```

I like to use full lines of ### to visually separate a long code document into chunks- especially on big projects or to separate homework problems.

```
# Create variables x and y
x <- rnorm(20)
y <- rnorm(20)

# Fit the linear model
model1 <- lm(y ~ x)

# Plot y versus x and include line of best fit
plot(x, y, main = "Example plot")
abline(model1)        #this is a comment too.
```

Comments can take up a whole line, or the end of a line. Anything after # but before a new line is commented and will not be run when you hit "Ctrl+Enter" or "Ctrl+R."

What if you type something into the console, and then decide you want to keep it as part of your script? Rather than retype the whole thing, either copy and paste directly from the console OR use the "History" tab which is in the same pane as the "Workspace" tab. Click where you want the line inserted in your script; then highlight the desired line in the "History" tab and click To Source. The line will be copied and pasted where your cursor last clicked in your script. This is especially useful if you load a workspace, package or data by point-and-click. RStudio generates a line of code that you can use to repeat this action in the future.

Writing Readable Code – The Importance of Style

As you become more comfortable writing code in R, you might want to consider your code writing **style**. As you gain more experience, you'll notice that every person's code looks just a little bit different- particularly choices for spacing, naming of objects and functions, and where comments are placed. If you are experienced in using other programming languages, you may have learned about programming style guides- a set of conventions programmers use in developing code that makes their code easier to read.

There is no single "correct" style for R programming, but it helps to develop some consistency about your coding choices. Here are some resources to get you thinking about this:

Google's R style guide: http://google-styleguide.googlecode.com/svn/trunk/Rguide.xml (the original R style guide- many, including Wickham's below, are based mainly on this guide).

Hadley Wickham's style guide: http://adv-r.had.co.nz/Style.html (Dr. Wickham created the package ggplot2 and now works for RStudio, so he knows something about elegant codewriting.)

Reverse engineering the R development team's style: http://cran.r-project.org/web/packages/rockchalk/vignettes/Rstyle.pdf

R Punctuation

#	Used to create comments in an R Script. Running these lines will create no output.
`""` or `''`	Double or single quotes indicate chunks of text that should be treated as text rather than names of functions or other objects.

`()`	Parentheses can be used to indicate order of operations in complicated computations:
	`(3+2)^2`
	
	They are used to indicate functions:
	`function(arg1, arg2, arg3)`

`[]`	Square brackets are used to subset matrices and vectors.
	
	This indicates the first column of the matrix “data”
	`data[, 1]`
	
	This indicates the second row of the matrix “data”
	`data[2, ]`
	
	This indicates the single item in the first column and second row
	`data[2, 1]`
	
	This indicates the third item in the vector called “variable”. Notice we don’t need a comma since vectors only have one set of indices.
	`variable[3]`

`:`	Used to create a list of integers. Can be used when creating “for loops” or for subsetting data This gives the first ten rows of data: `data[1:10, ]`

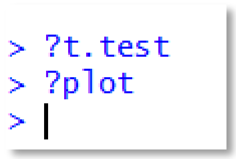
`$`	Used to subset a dataframe or list object.
	
	Get the variable called “var1” from the dataframe called “data”
	`data$var1`
	
	When output from linear model regression function is saved as “model” can access coefficients, etc.
	`model <- lm(y~x, data=data)`
	`model$coeff`

`{}`	Used for creating chunks of code, particularly for loops.
	`for(i in 1:100){`
	`    iscambinomtest(i, 100, .5, “greater”)`
	`    }`

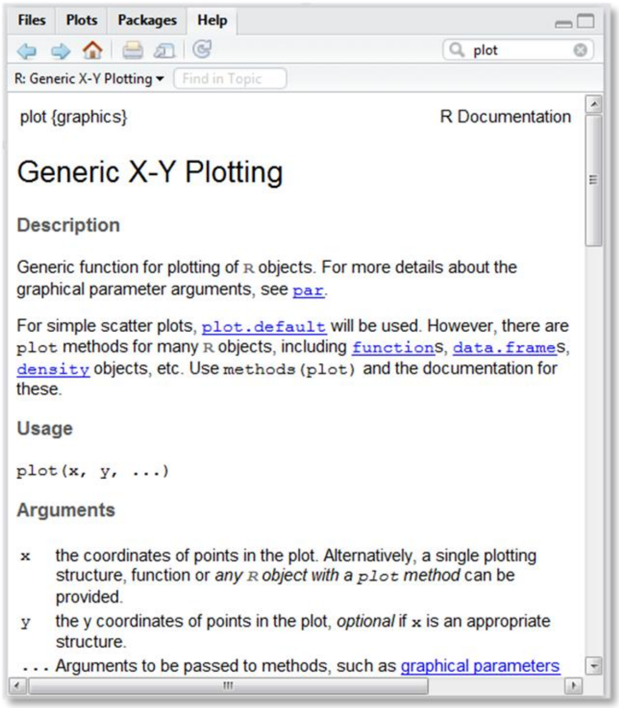
`~`	Used for creating formula/model statements, e.g. for `t.test` or `lm` function. `t.test(response~explanatory, data=data)`

Getting Help

For any R functions (that are part of base R or added through packages) you can access the help menu by typing ? before the function, or by searching directly in the “Help” tab in the lower right.



```
> ?t.test
> ?plot
> |
```



plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

Usage

```
plot(x, y, ...)
```

Arguments

x	the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any R object with a plot method* can be provided.
y	the y coordinates of points in the plot, *optional* if x is an appropriate structure.
...	Arguments to be passed to methods, such as graphical parameters

1. Check out the “R Code Library” on Moodle- this is a student generated forum for you to share code with each other.

2. Google it! Seriously. Just Google “R t-test” or “R plotting” and you will be amazed.

3. Ask your professor. This shouldn’t be your last resort, but I’m not always available immediately.

Common Error Messages

If you get red output, you have experienced an error. Here are a few of the most common error messages you will encounter.

`Error: Object '...' not found`

This means the object referred to is not in your workspace. This could be because:

1. You forgot to load data or install a package.
2. You made a type-o or capitalization error.
3. You may have forgotten quotation marks, e.g. "greater" as the function input for hypothesis tests. Also check that logicals such as TRUE/FALSE are in all caps.
4. You forgot to run a line of code earlier which created the object you are referring to. (Scroll up through your console to make sure).
5. You may be attempting to refer to a variable within a specific dataset. For example, you want to look at the variable "`time`" within the dataset "`OldFaithful`". Instead of referring just to time, try `OldFaithful$time` or specify `data=OldFaithful` within your function (The first way should always work... the second way only works for certain functions)

`Error in plot.new() : figure margins too large`

You have created a plot but it doesn't fit in your plot window. Try increasing the size of the plot window and rerunning your plot command.

`Error: unexpected numeric constant in: ...`

You are most likely missing a parenthesis, a comma, or ran a line of code with a + prompt when you thought the previous line had completed. Carefully read your code line and check for all the proper syntax.

`Error in ....: undefined columns selected`

This means the columns of the data set you've selected don't exist. If selecting columns numerically, be sure you have the indices correct. If selecting by name, check spelling and capitalization. Finally, check to be sure you loaded the data correctly and that variables names are appearing as column headings and not as the first row of data.

`You keep getting + when you expect >`

You've probably missed a parenthesis or accidently ran only half a line of code. Just hit **Esc** to get out of this mess and then carefully reread your code in the script window, checking for parentheses.

`You keep getting NA as your answer`

This means you have missing values in your dataset. You can either "clean" your data to get rid of missing values, or check the help menus for the function you are using to see if there are options for dealing with missing values.

`Warning messages` mean the code ran, but there may be a problem. Don't ignore these! Read them! They may indicate a problem with your code or that statistical assumptions are not met.

R functions used in Stat 272

This is not an exhaustive list, but hits the ones we will use repeatedly. You might consider keeping your own "Code document" with examples of how to use these functions as we come across them rather than having to search through the book or Moodle every time. The R reference card on Moodle is also a good resource.

```
abline
barplot
boxplot
cbind
chisq.test
cor
exp
for
glm
hist
histogram {lattice}
legend
lm
log
matrix
mean
par
plot
prop.table
qqnorm
rbind
rbinom
round
sample
sd
step
summary
t.test
table
```