Course Content for Biom612, Week 1

Deanne Taylor and Joe Dybas

Jan 8th, 2018

Contents

1 Introduction to R

R is a powerful and useful computing language, and is written to be easy to use. If you can learn some basic R programming skills, they will be very useful to you in the future. Learning R is more useful overall than learning 3rd party analysis software packages.

Back in the 1970s, 9 IT scientists at Bell Labs came up with a Statistical programming language they named 'S'. Since S was so popular, but cost money to license, a group of people set out to re-create and improve on 'S' with a version that was a free and open-source, meaning that communities of people could contribute to the language development. They named this language "R". R has become quite popular, so one of the major advantages of using R is that it has many published and pre-configured statistical software packages submitted by a vast community of developers in every quantitative scientific field. With all these packages and plenty of references to back them up, there's no need to re-write the same code to do specialized computational tasks. In fact, there are hundreds of different, useful packages ("libraries") provided in the R repositories, provided by academic and industrial groups whose submitted work is scrutinized, curated, and then accepted by the R-using community. Many methods and R packages have been published in peer-reviewed journals and some cited thousands of times.

2 Getting R

The "native" operating environment for R is the command line.

Figure 1 shows an R session in a Mac Terminal, accessed by simply typing "R" on a command line in an environment where R is installed. We'll be installing R next.

The top lines that appear at opening the R environment gives information on the R version, the platform for which it was compiled, and some information on the program

2.1 Download R for your platform

Download R from CRAN (the comprehensive R archive network) at this link:

http://cran.mirrors.hoobly.com/

Click on one of the download platforms and follow the directions to install R.

3 Installing the GUI: RStudio

R in the command line is powerful but not super user-friendly. For that reason, several "wrappers" have been designed that sit "on top of" R in order to provide a better user experience. RStudio is one such GUI (Graphical User Interface) to R that provides a great desktop environement.

After you install R, install RStudio from this link: https://www.rstudio.com/products/rstudio/download2/

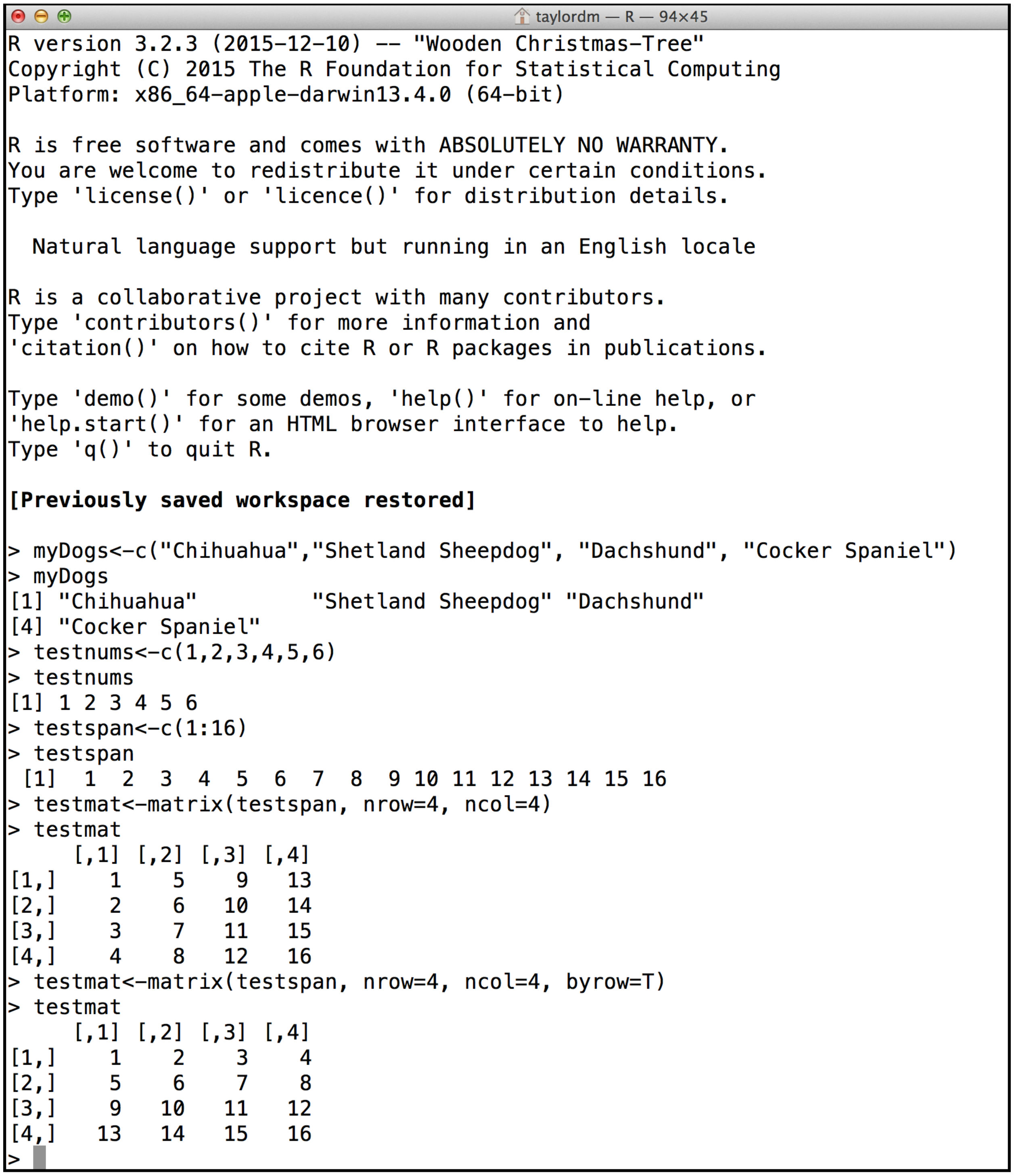
Install with defaults for your platform.

1. Open RStudio. Double Click on the R Studio icon. In RStudio, "projects" are just organized directories and collections of files. It's a good way to keep track of all your data associated with a particular task.
2. Create a Project. On your computer's desktop, or anywhere else you'd like to create your directory that's easy to access, create a new directory "RCourse". Once in RStudio, we will create a new "project". To navigate, you will select from the menu, `File -> New Project -> Existing Directory`, and navigate to the folder you just created.

You can repeatedly use this organized directory ("project") for this assignment by using `File -> 'Recent Projects'`

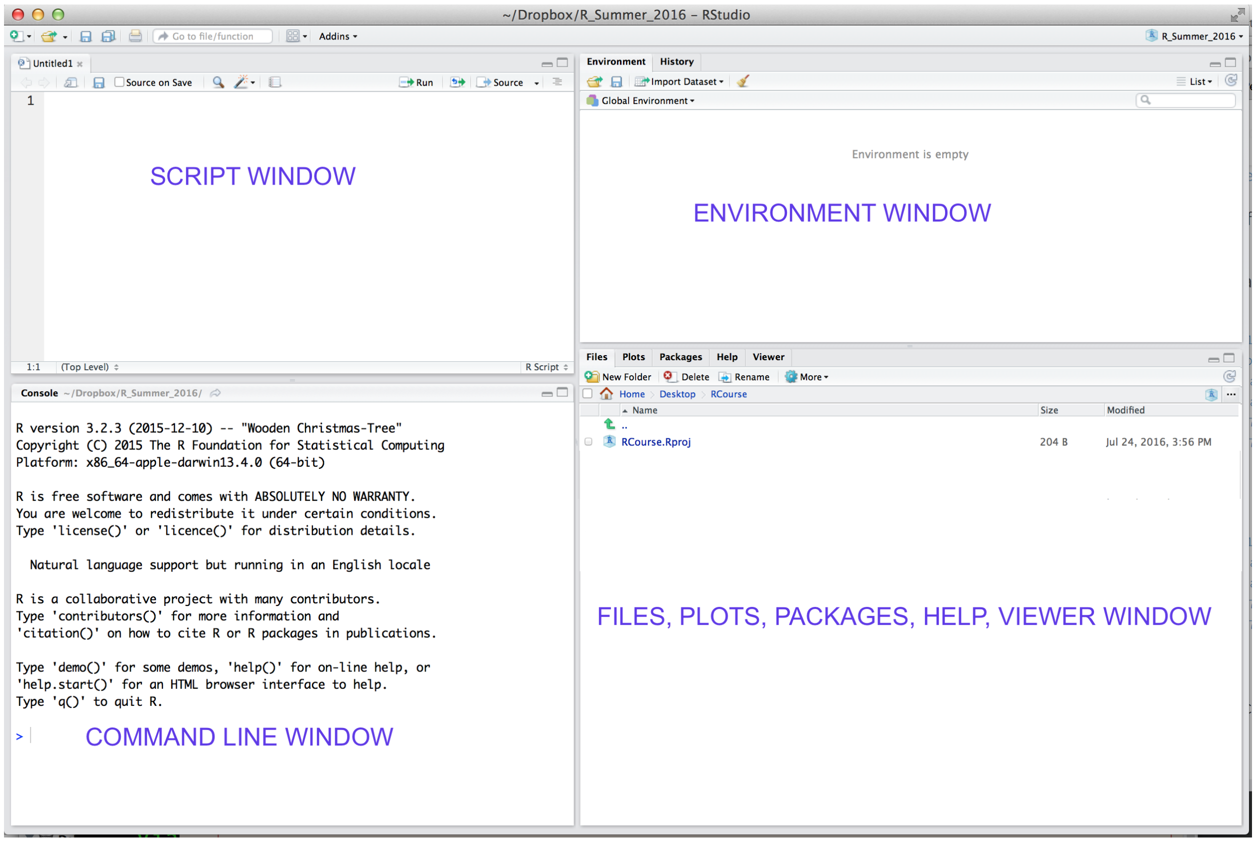
3. Create a File. There are many different types of files we can create that can produce R output. We'll start with a simple script. From the menu, select `File -> New File -> R Script`.

Your environment should look like that in Figure 2, minus the big blue letters of course. :)


taylordm — R — 94×45

```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

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]

> myDogs<-c("Chihuahua","Shetland Sheepdog", "Dachshund", "Cocker Spaniel")
> myDogs
[1] "Chihuahua"         "Shetland Sheepdog" "Dachshund"
[4] "Cocker Spaniel"
> testnums<-c(1,2,3,4,5,6)
> testnums
[1] 1 2 3 4 5 6
> testspan<-c(1:16)
> testspan
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
> testmat<-matrix(testspan, nrow=4, ncol=4)
> testmat
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> testmat<-matrix(testspan, nrow=4, ncol=4, byrow=T)
> testmat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
>
```

Figure 1: Command Line with Some Examples



```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

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.

>
```

Figure 2: The R Studio Environment after creating an R Script file with added annotations for each window

3.1 Getting Help

When asking for help on an R function or operator, the best bet is to enclose them in quotes after a question mark.

`?"+"` gives you help on the plus symbol. `?"sub"` gives you help on the substitution operator. If you want to do a more global search for a term in the help files, use double question marks, such as `??"Operators"`.

4 Installing R packages

In your lower right window, click on the Packages tab. Click on the Install tab that shows up and type in a package name (assuming it's not already installed). To activate a package, you can either check the box in the RStudio display, or you can type

`library(packagename)` or `require(packagename)`

Alternatively you can use a function from a package without loading the whole package (assuming you already have the library installed), using the function call `package::function` such as `knitr::kable` where `knitr` is the library and `kable` is the function.

5 The .Rmd file in R Studio

Because we named this file that you're reading right now, to end with the suffix ".Rmd", RStudio understands this file to be an R Markdown type file. This means we'll have more functionality in how RStudio offers to run and execute code, than if it was a simple R script.

Help on Markdown has been selected and assembled from help files and other sources as noted. More markdown help can be found at http://rmarkdown.rstudio.com.

The PDF output from this document represents the output of an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents.

R Markdown has simple formatting rules that can be interepreted to generate professional-looking documents. We discuss more of those below.

6 Basic Data Types and Data Entry

As mentioned above, the command line is R's native habitat. The RStudio GUI is simply a wrapper on top of R's executable that also helps with the management of libraries (packages), plots, and the computational environment.

Let's start with some command-line entry. Click in the lower left window (Command Line Window in Figure 2). You can paste any R text into your window or use the provided script if you have this document open in R. Select the text you'd like to run in this or any script, and click "Run", then "Run Selected Lines" in the upper part of the script window.

The code inside the code chunks shown in this document should be able to be pasted into R directly or run line-by-line, but beware double quotes that are sometimes changed in translation in any document. Error messages may be due to cut-and-paste transforms of double quotes into something that R can't understand. If you get error messages on something with double or single quotes, try typing them into the command line manually to make sure you're using the correct text-based quotes.

6.1 Variable Assignment

Like a calculator, R can calculate on any number without assigning it a variable, for instance, "`5+1`" or "`10/2`" can be typed right in the R command window on the lower left window of your R Studio application. However, you will often want to store values in a variable, called 'assignment' of a number to a variable.

In R, a value is assigned to a variable (such as 'x') using one of several assignment operators:

- `x <- value` (this is the most common form). Assign value to x.
- `x <<- value` (used most often in functions)
- `value -> x` (less used form)
- `value ->> x` (used in functions)
- `x=value` (can be used in special conditions, see below)

The operator `<-` assigns into the environment in which it is evaluated. The operator `<-` can be used anywhere, whereas the operator `=` is only allowed at the top level (e.g., in the complete expression typed at the command prompt) or as one of the subexpressions in a braced list of expressions.

When in doubt, use the typical assignment operator '`<-`'.

6.2 Numbers and representations

Characters are represented with quotes around them. `"Hi"`, numbers are just entered in as numbers or decimal points. There are other special types: `NA, NaN, Inf, -Inf`. `NA` is "Not Available" and used for missing values. There are no "blank" assignments in R. `NaN` is used for number results and stands for 'Not A Number'. `Inf` and `-Inf` are positive and negative infinity. Instead of producing an error in many cases, R will produce these special values that can show up in a result.

7 R Code Chunks

R code chunks are specific for markdown type documents. Chunks are used as a means to delineate and render R output. You have seen code chunks if you open an R Markdown (.Rmd) file in R. They are defined by three backquotes and a bracketed name. They begin with the small letter 'r' and following that is the name of the chunk. You can create code chunks with the "Insert" menu at the top of your script window (look for the little green "c") or you can "manually" type three backticks, a bracketed command, and three backticks to close out the chunk.

In your script window you will see the code chunks colored a light grey, as long as the file is named with an .Rmd suffix:

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Other options in the bracketed header at the top of each chunk can relay how the data is output and viewed. Chunks cannot share the same name. R code in chunks are executed as the document is assembled. R code can be displayed except if certain options are selected, such as `echo=FALSE`: `{r cars, echo=FALSE}` which prevents the code from being displayed before the results are shown.

There are several other options in chunks. . . to display R code without evaluating it, you specify the `eval=FALSE` chunk option: `{r, eval=FALSE}`

By default data frames and matrixes are output as they would be in the R terminal (in a monospaced font). However, if you prefer that data be displayed with additional formatting you can use the `knitr::kable` function. The kable function includes several options to control the maximum number of digits displayed in numeric columns, the column contents alignment and etc. See documentation (`help("kable")`)

The Tidy function helps make sure text is well behaved, wraps properly and/or observes page boundaries.. Options can be set globally at the beginning of the document or within the chunk definition itself.

When you click the **Knit** button in R Studio when an R Markdown file is open, a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

```
knitr::kable(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

8 Built-in data sets and basic plotting

R has many built-in data sets for testing. They are 'installed' when you install R. They are available at any time for you to test code or graphics against.

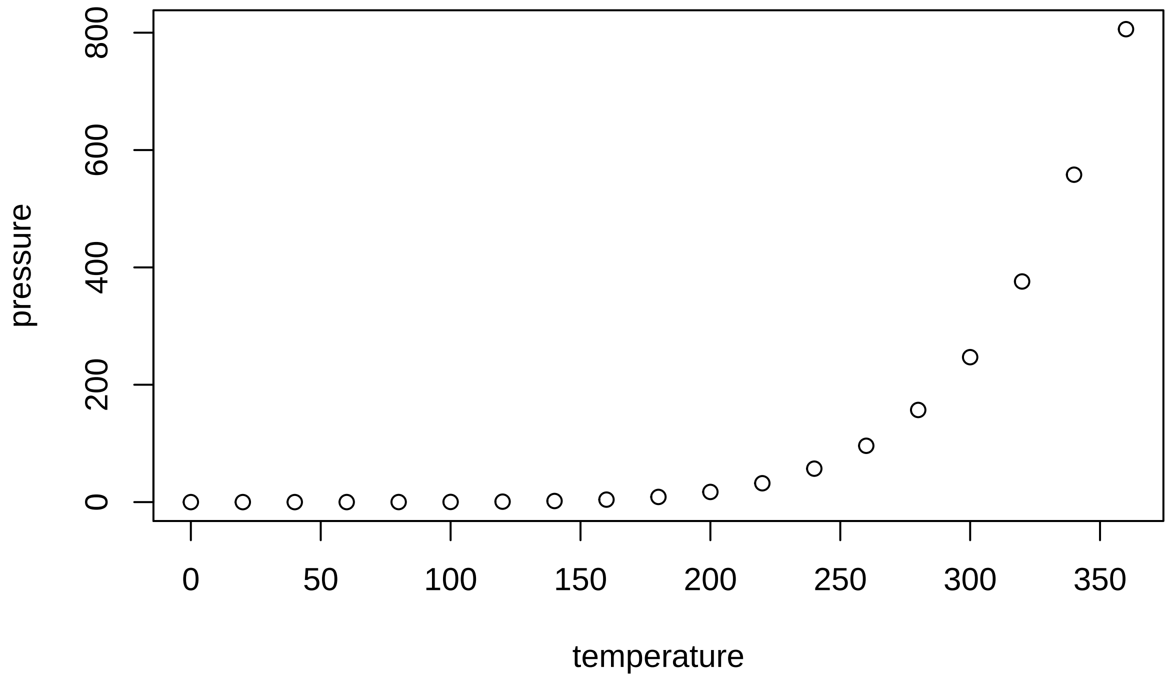
One such data set is the pressure data set.

Using R chunks, you can embed plots in the R markdown document, for example:

```r
# Let's look at the pressure table, using the nice formatting table function
# "kable" from knitr:

knitr::kable(pressure)
```

temperature	pressure
0	0.0002
20	0.0012
40	0.0060
60	0.0300
80	0.0900
100	0.2700
120	0.7500
140	1.8500
160	4.2000
180	8.8000
200	17.3000
220	32.1000
240	57.0000
260	96.0000
280	157.0000
300	247.0000
320	376.0000
340	558.0000
360	806.0000

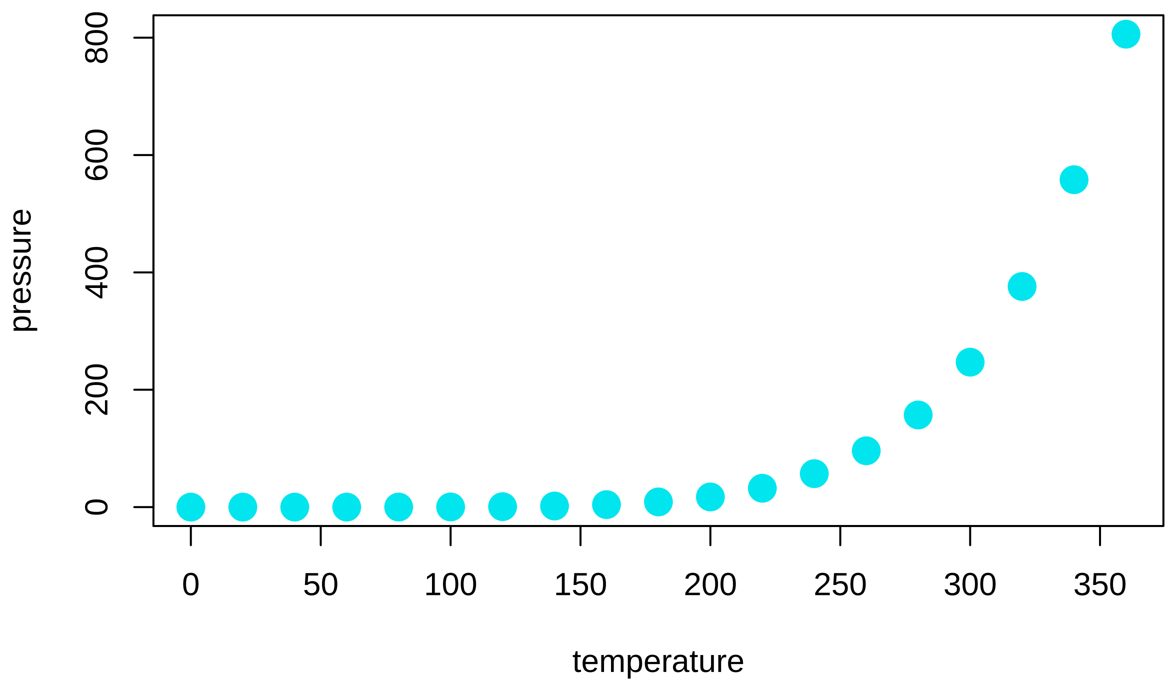
```r
# Plotting in R is very easy, but has lots of options available. Let's start
# with the most basic:

plot(pressure)
```



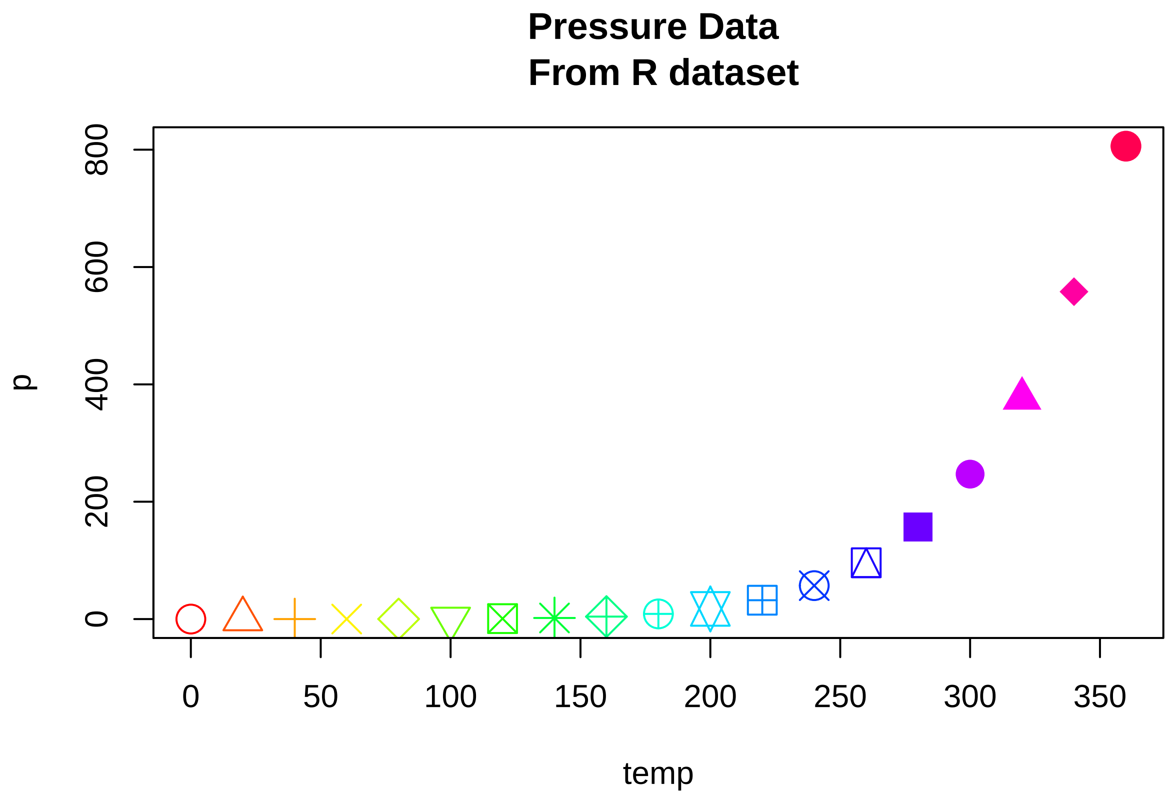
```r
# Let's look at the first fifty colors R offers (for the full range of hundreds
# of colors, just type colors() )

colors()[1:50]
```

```
##  [1] "white"          "aliceblue"      "antiquewhite"   "antiquewhite1"
##  [5] "antiquewhite2"  "antiquewhite3"  "antiquewhite4"  "aquamarine"
##  [9] "aquamarine1"    "aquamarine2"    "aquamarine3"    "aquamarine4"
## [13] "azure"          "azure1"         "azure2"         "azure3"
## [17] "azure4"         "beige"          "bisque"         "bisque1"
## [21] "bisque2"        "bisque3"        "bisque4"        "black"
## [25] "blanchedalmond" "blue"           "blue1"          "blue2"
## [29] "blue3"          "blue4"          "blueviolet"     "brown"
## [33] "brown1"         "brown2"         "brown3"         "brown4"
## [37] "burlywood"      "burlywood1"     "burlywood2"     "burlywood3"
## [41] "burlywood4"     "cadetblue"      "cadetblue1"     "cadetblue2"
## [45] "cadetblue3"     "cadetblue4"     "chartreuse"     "chartreuse1"
## [49] "chartreuse2"    "chartreuse3"
```

```r
# Let's re-do that plot:

plot(pressure, pch=16, cex=2, col="turquoise2")
```



```
# R also has some built-in color gradients, such as rainbow:

plot(pressure, pch=c(1:19), cex=2, col=rainbow(nrow(pressure)),
     xlab="temp", ylab="p",
     main="Pressure Data \n From R dataset")
```



Typical plot symbols for R scatterplots:

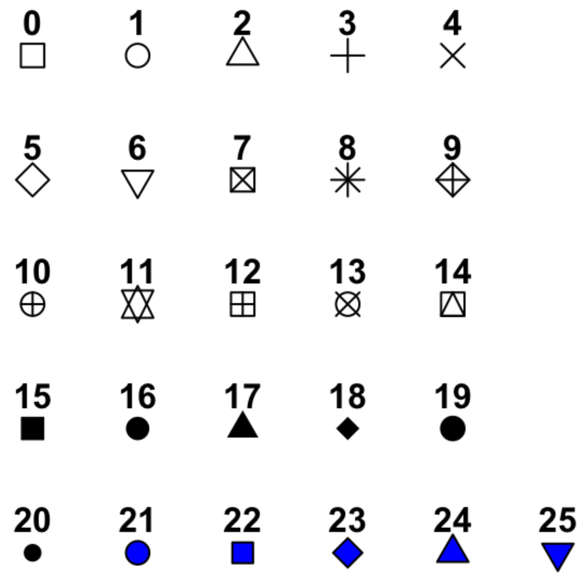


Figure 3: PCH Symbols

9 Deleting your environment elements

Want to delete your old data from the environment and start fresh? Use the `rm()` command like so:

```
rm(list=ls())
```

10 Basics of R Markdown Format

Everything outside of a code chunk in an .Rmd file is parsed as R Markdown. There are some very simple rules to produce a great quality document. A good source for details is this page or examples throughout this document.

You can also peruse the R studio markdown pages here: http://rmarkdown.rstudio.com/authoring_basics.html

Also check out the R Markdown reference "cheatsheet": http://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf

Subscripts are produced using the tilde sign on either side:

`My~subscript~`

$\text{My}_{\text{subscript}}$

11 Data Types

11.1 Vectors

Vectors are a basic data type in R. Most everything you will be operating on in R is a vector. To create a vector in R, one way is ot use the '`c`' function, which stands for 'combine'.

To create a vector, you can type this:

```r
myvec <- c(1,2,3,4)

# type the variable name to return what is stored in the variable:
myvec
```

```
## [1] 1 2 3 4
```

but you will not be able to do this without the combine function. (note the missing 'c' in front of the parenthesis!)

```r
myvec <- (1,3,4,5)
```

There is no such thing as a 'naked' number or character in R. In R, `x<-5` assigns a vector of length 1. It is equivalent to `x<-c(5)` The maximum length of (number of elements in) a vector is `2^31-1`, about 2 billion.

11.1.1 Vector types

Vector types are: "logical", "integer", "numeric" (also synonym "double"), "complex", "character" and "raw".

The three most common vector types you will encouter are "integer", "numeric" and "character".

Vectors are "strictly typed". That is, a vector is only allowed to be one type. This is a very important point. If you try to insert a character such as "Hi" int o a numeric vector, it will change the entire vector into a character vector. Your numbers will become characters. Some frustrating errors can happen that way, where your calculations are not working because your vector is a character not a number – which is why you need the functions typeof and structure ('`str`') to help you figure out what you're working with. R cannot do numerical calculations on a character vector.

```r
# character vector
myDogs<-c("Chihuahua","Shetland Sheepdog", "Dachshund", "Cocker Spaniel")
myDogs # type the name of the item to see it returned.
```

```
## [1] "Chihuahua"         "Shetland Sheepdog" "Dachshund"
## [4] "Cocker Spaniel"
```

```r
?str # help on structure.
str(myDogs) # structure of an R object
```

```
##  chr [1:4] "Chihuahua" "Shetland Sheepdog" "Dachshund" ...
```

```r
?typeof # type or storage mode of an object
typeof(myDogs)
```

```
## [1] "character"
```

```r
# number vector
testnums<-c(1,2,3,4,5,6)
testnums
```

```
## [1] 1 2 3 4 5 6
```

```r
str(testnums)
```

```
##  num [1:6] 1 2 3 4 5 6
```

```r
typeof(testnums)
```

```
## [1] "double"
```

```
# fast way to generate a number vector
testspan<-c(1:16)
testspan
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

```
# add together two vectors

newvec<-append(testnums, c(7,8,9))
newvec
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
# try appending a character vector
newvec2<-append(testnums, c("Hi", "there"))
newvec2
```

```
## [1] "1"     "2"     "3"     "4"     "5"     "6"     "Hi"    "there"
```

```
str(newvec2)
```

```
##  chr [1:8] "1" "2" "3" "4" "5" "6" "Hi" "there"
```

```
typeof(newvec2)
```

```
## [1] "character"
```

11.2 Factors

Factors are a special type of data in R. It is a way of representing (encoding) strings (characters) as numbers. Factors can be an annoyance if you just want the character vectors as they essentially force categories (levels) on each string. The default for reading in files is to turn all character columns into factors, so in this course we will show you how to avoid that complication.

Imagine you had a column of length 10 that only held three different types of items “red”, “blue”, “green”. Factors represent those three different types as ‘levels’, the levels being (“blue”, “green”, “red”) here in alphabetical order. Let’s try it out.

```
myfactordata<-c("red", "blue", "red", "green", "red", "blue", "blue", "red", "green", "green")
str(myfactordata)
```

```
##  chr [1:10] "red" "blue" "red" "green" "red" "blue" "blue" "red" ...
```

```
typeof(myfactordata)
```

```
## [1] "character"
```

```
# now let's turn myfactordata into a factor.
myfactor<-factor(myfactordata)
str(myfactor)
```

```
##  Factor w/ 3 levels "blue","green",..: 3 1 3 2 3 1 1 3 2 2
```

```
typeof(myfactor)  # what happened? My character vector is now an integer?
```

```
## [1] "integer"
```

```
myfactor
```

```
##  [1] red   blue  red   green red   blue  blue  red   green green
## Levels: blue green red
```

```
myfactor[2]
```

```
## [1] blue
## Levels: blue green red
```

```
levels(myfactor)
```

```
## [1] "blue"  "green" "red"
```

11.3 Matrices

Matrices are strongly typed arrays of numbers or characters. Vectors can be coerced into matrices. Many R functions require matrices and will not accept data frames. We will be discussing matrices throughout the course, so we will simply introduce them here.

```
# matrices are arrays of a single type (either number, or character...)
testmat_col<-matrix(testspan, nrow=4, ncol=4)
testmat_col
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

```
# another way to arrange a matrix.
testmat_row<-matrix(testspan, nrow=4, ncol=4, byrow=T)
testmat_row
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

11.4 Data Frames

Data frames are one of the major data types in R. Data Frames are different than matrices as you can have mixed types in data frames, but matrices are strongly typed. Data frames can be considered containers of vectors. Each column in a dataframe is a vector, and must have the same length as every other vector in the data frame. You can think of a data frame as a container for vectors.

Below, I use the '`kable`' command from the knitr package to neaten up the view. If you don't have the kable package installed you won't be able to use this command. You can just type the data frame name to see the data.

```
# create a bunch of mixed vectors.

dogweights <- c(10.11, 20.42, 15.15, 35.73) #Numerical
dogages<-c("4", "5","6", "7") #Integer
dogbreeds<-c("Chihuahua","Shetland Sheepdog", "Dachshund", "Cocker Spaniel") #character
dognames<-c("Spike", "Trixie", "Buster", "Basil")

# combine these into a data frame with names representing the columns.
```

```
mydat1<-data.frame("Name"=dognames, "Breed"=dogbreeds,"Weight"=dogweights,
                   "Age"=dogages, stringsAsFactors = F)

kable(mydat1)
```

Name	Breed	Weight	Age
Spike	Chihuahua	10.11	4
Trixie	Shetland Sheepdog	20.42	5
Buster	Dachshund	15.15	6
Basil	Cocker Spaniel	35.73	7

```
# creating a data frame with no labels
mydat2<-data.frame(dognames, dogbreeds,dogweights, dogages,stringsAsFactors = F)
kable(mydat2)
```

dognames	dogbreeds	dogweights	dogages
Spike	Chihuahua	10.11	4
Trixie	Shetland Sheepdog	20.42	5
Buster	Dachshund	15.15	6
Basil	Cocker Spaniel	35.73	7

```
# create the data in situ in the data frame.

mydat3<-data.frame(c(10.11, 20.42, 15.15, 35.73),
                   c("4", "5","6", "7"),
                   c("Chihuahua","Shetland Sheepdog", "Dachshund", "Cocker Spaniel"),
                   c("Spike", "Trixie", "Buster", "Basil"),
                   stringsAsFactors = F)

kable(mydat3)
```

c.10.11..20.42..15.15..35.73.	c..4....5....6....7..	c..Chihuahua....Shetland.Sheepdog....Dachshund....Cocker.Spaniel..
10.11	4	Chihuahua
20.42	5	Shetland Sheepdog
15.15	6	Dachshund
35.73	7	Cocker Spaniel

```
# well that looks terrible!
# let's add some actual names to those columns.

colnames(mydat3)<-c("Name", "Breed", "Weight", "Age")
rownames(mydat3)<-c("Dog1", "Dog2", "Dog3", "Dog4")
kable(mydat3)
```

	Name	Breed	Weight	Age
Dog1	10.11	4	Chihuahua	Spike
Dog2	20.42	5	Shetland Sheepdog	Trixie
Dog3	15.15	6	Dachshund	Buster

	Name	Breed	Weight	Age
Dog4	35.73	7	Cocker Spaniel	Basil

```
# What is the breed of the third dog in the table?
# addressing a data frame goes by [row,column]
mydat3[3,3]
```

```
## [1] "Dachshund"
```

```
# which dog is fourth (row) in the data frame?
mydat3[4,]
```

```
##       Name Breed         Weight   Age
## Dog4 35.73     7 Cocker Spaniel Basil
```

```
# Pull out the second column.
mydat3[,2]
```

```
## [1] "4" "5" "6" "7"
```

12 Operators and Functions

12.1 Arithmetic Operators

There are familiar arithmetic operators in R, but a few that are specific for the R language.

```
x=10
y=5

+ x      # (change the sign to positive)
```

```
## [1] 10
```

```
- x      # (change the sign to negative)
```

```
## [1] -10
```

```
x + y    # (add y to x)
```

```
## [1] 15
```

```
x - y    # (subtract y from x)
```

```
## [1] 5
```

```
x * y    # (multiple x by y)
```

```
## [1] 50
```

```
x / y    # (divide x by y)
```

```
## [1] 2
```

```
x ^ y    # (raise x to the power of y)
```

```
## [1] 1e+05
```

```
x %% y  # (x mod y)
```

```
## [1] 0
```

```
x %/% y # (integer division (does not return a floating point number))
```

```
## [1] 2
```

12.2 Basic Functions

R has some basic functions that appear in (no surprise) the R library called '`base`'. These include summation, rounding, etc. You can get a list of all the functions in the base package with the `library(help='package')` command:

```
library(help = "base")
```

Here are a few examples:

```
testnums<-c(1:10)  # create a vector.
typeof(testnums) # what type is it?
```

```
## [1] "integer"
```

```
mean(testnums) # mean
```

```
## [1] 5.5
```

```
sum(testnums)  # sum
```

```
## [1] 55
```

```
sd(testnums)  # standard deviation
```

```
## [1] 3.02765
```

```
round(sd(testnums),digits=2)  # let's round up that example.
```

```
## [1] 3.03
```

```
summary(testnums)  # produce useful statistics on the vector.
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

12.3 Logical Operators

We'll cover the subset function, a convenience function to manipulate data frames built into basic R.

In order to use subset, you'll need to know about logical operators. Here are the most common ones used:

- `! x` : NOT x
- `x & y` : x AND y
- `x | y` : x OR y
- `== x` : is equal to x
- `> x` : greater than x
- `< x` : less than x
- `<= x` : less than or equal to x
- `>= x` : greater than or equal to x

Also, there are others that are useful that also deliver TRUE/FALSE results:

- `x %in% y` : Is x in y?
- `is.na(x)` : Is x an NA (not available) result?
- `is.nan(x)` : Is x a Not A Number result?
- `is.finite(x)` : Is x a finite number?
- `is.infinite(x)` : Is x an infinite result?

Let's try the logical operation again:

```
x <- 5
y <- c(1:10)

y != x
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
y==x
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
y>x
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
y<x
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
y>= x
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
y<=x
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
y>8 & y !=9
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

```
y<3 | y>7
```

```
##  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

```
x %in% y #x in y
```

```
## [1] TRUE
```

```
!(x %in% y) #NOT x in y?
```

```
## [1] FALSE
```

So what do the logical vectors allow us to do? In several ways of manipulating R vectors and data frames, it allows us to select out elements, based on the logical operators that are TRUE versus FALSE.

In order to do the subset, you have to give the column name that you are trying to do a logical operaton on. To figure out which column names we have we'll use the mtcars data set as an example.

12.4 Manipulating and reporting data frames with Logical Operators

As we mentioned, data frames are collections of column vectors.

We'll use the mtcars data set included in R for this exercise.

```r
# pick out some data
mtcars[3, 4]  #access data by row-col (position)
```

```
## [1] 93
```

```r
mtcars["Datsun 710", "hp"]  #Same data this time by row-col names
```

```
## [1] 93
```

```r
# subset data across a larger collection. subset() is the easiest way in
# basic R to get this accomplished.
subset(mtcars, hp >= 250)
```

```
##                 mpg cyl disp  hp drat   wt qsec vs am gear carb
## Ford Pantera L 15.8   8  351 264 4.22 3.17 14.5  0  1    5    4
## Maserati Bora  15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
```

```r
# you can also use subset to select out certain columns for your subset.

subset(mtcars, hp >= 250, select = c(mpg, gear))
```

```
##                 mpg gear
## Ford Pantera L 15.8    5
## Maserati Bora  15.0    5
```

```r
# make a wider selection
subset(mtcars, hp >= 250, select = c(mpg:gear))
```

```
##                 mpg cyl disp  hp drat   wt qsec vs am gear
## Ford Pantera L 15.8   8  351 264 4.22 3.17 14.5  0  1    5
## Maserati Bora  15.0   8  301 335 3.54 3.57 14.6  0  1    5
```

```r
subset(mtcars, (hp >= 250 | hp < 70), select = c(mpg:wt))
```

```
##                 mpg cyl  disp  hp drat    wt
## Merc 240D      24.4   4 146.7  62 3.69 3.190
## Fiat 128       32.4   4  78.7  66 4.08 2.200
## Honda Civic    30.4   4  75.7  52 4.93 1.615
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170
## Maserati Bora  15.0   8 301.0 335 3.54 3.570
```

As a convenience function, there are other ways to do subsetting, including with the which() function.

First we'll show how which() can use logical operators.

How do we tell which() which column to select? Clearly subset can do this pretty easily…but for which we'll use the $ operator. This is called Extract (note capital E) and you will have to type help("Extract") to get more information.

```r
mtcars$mpg  # extract out just the column with mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

```r
mtcars[,1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

```
mtcars$mpg > 30
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

```
which(mtcars$mpg > 30)
```

```
## [1] 18 19 20 28
```

```
# let's just select the rows where mpg > 30 in mtcars, which is the form that
# mimics subset.

mtcars[which(mtcars$mpg >30),]
```

```
##                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
## Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

```
mtcars[which((mtcars$mpg >30 | mtcars$hp >200) & mtcars$cyl==8 ),]
```

```
##                      mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
## Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
## Ford Pantera L      15.8   8  351 264 4.22 3.170 14.50  0  1    5    4
## Maserati Bora       15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
```

12.5 Convenience Functions

These functions allow you to start exploring a data set using shortcuts.

```
# select a random sample of 10 cars from the mtcars data w/out replacement.
# This sample will change every time you invoke the function.
mtcar_sample <- mtcars[sample(1:nrow(mtcars), 10,
   replace=FALSE),]

knitr::kable(mtcar_sample)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

```r
# Details on the mtcars dataframe
# number of rows, columns in dataframe
nrow(mtcars)
```

```
## [1] 32
```

```r
ncol(mtcars)
```

```
## [1] 11
```

```r
# report out just the top of the data frame
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
# get the row,column names of the data frame.
rownames(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

```r
colnames(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```r
# manipulate your column names

colnames(mtcars)[1]<-"MPG"

# create a new vector of the same length as the columns of the data frame:

newvec1<-rownames(mtcars)

# check the length
length(newvec1)
```

```
## [1] 32
```

```r
# add the new vector to your data frame with column bind:
new_col_mtcars<-cbind(mtcars, newvec1)

knitr::kable(new_col_mtcars)
```

	MPG	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	newvec1
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4	Mazda RX4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4	Mazda RX4 Wag
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1	Datsun 710
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1	Hornet 4 Drive
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2	Hornet Sportabout
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1	Valiant
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4	Duster 360
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2	Merc 240D
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2	Merc 230
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4	Merc 280
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4	Merc 280C
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3	Merc 450SE
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3	Merc 450SL
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3	Merc 450SLC
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4	Cadillac Fleetwood
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4	Lincoln Continental
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4	Chrysler Imperial
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1	Fiat 128
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2	Honda Civic
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1	Toyota Corolla
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1	Toyota Corona
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2	Dodge Challenger
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2	AMC Javelin
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4	Camaro Z28
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2	Pontiac Firebird
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1	Fiat X1-9
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2	Porsche 914-2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2	Lotus Europa
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4	Ford Pantera L
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6	Ferrari Dino
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8	Maserati Bora
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2	Volvo 142E

```r
# create a new row with the same length as the rows in mtcars:
newvec2<-c(1:ncol(mtcars))

# add the new row to mtcars
new_row_mtcars<-rbind(mtcars, newvec2)

knitr::kable(new_row_mtcars)
```

	MPG	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2

	MPG	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
33	1.0	2	3.0	4	5.00	6.000	7.00	8	9	10	11

13 Reading and writing files into R

Save the following file into your RCourse folder and unzip the file: ftp://ftp.ncbi.nlm.nih.gov/geo/datasets/GDS4nnn/GDS4778/soft/GDS4778_full.soft.gz

After unzipping, your file should be named `GDS4778_full.soft`

If we try to read that file into R without any manipulation, we'll get an error message because the first 29 rows are information about the experiment.

This is where Bioconductor will come in handy in two weeks, as we can read these files "natively" and all the information will be stored. For now, let's read the data frame directly into R by skipping the first 70 lines.

There are several ways of reading and writing data from R, but the functions I'll cover are "`read.csv`", "`write.csv`", "`read.table`", and "`write.table`"

`read.csv` and `write.csv` are convenience functions for the `read.table` and `write.table`.

Note we use the `stringsAsFactors=F` again. We don't want the text-based part of this large file to be transformed into factors.

The header parameter defaults to "`T`" but I put it in for emphasis – not all your data will have a header (column names).

`row.names` parameter tells R where to get the rownames. Here I'm saying column 1 (`row.names=1`) is where the row names are contained.

`Sep` is a parameter indicating the delimiter for your file. In `read.csv` and `write.csv` it defaults to comma. In `read.table` it needs to be delimited. You should know that the encoding for a tab character in R is "`\t`", which we will use to parse the SOFT file. You can set this in `read.csv`. I suggest looking at the help for `read.table` to get more information.