Back to basics - R

Caitlin Simopoulos

2017-05-18

Some R basics...

Helpful R websites

1. http://www.r-tutor.com
2. https://www.r-bloggers.com
3. http://stackoverflow.com/documentation/r/topics This one is cool and new!!

R Studio

I would recommend downloading https://www.rstudio.com. It's a better version of the R GUI. And when you become obsessed with VIM (you will, don't worry), you can set R Studio to VI mode and all will be good in the world.

R Markdown

This is an R Markdown document (markdown == "simplified" latex...). I encourage you to write your "lab notes" for anything you do in R/python in this kind of document (BUT do as I say, not as I do!). R Markdown is based on "Markdown" which is used on a lot of websites (ahem..Reddit), and is used for all those README.md files you see on github. In R Markdown, you can knit output to PDF, HTML and Word documents.

However...if you really like LaTeX, you can call R in all TeX stuff using notation similar to R Markdown (this can be really useful when making presentations in beamer). For more details on using R Markdown see http://rmarkdown.rstudio.com and https://gist.github.com/jeromyanglim/2716336.

Important and helpful R commands/functions

As you might know, R is a statistical programming language, meaning, it's gotta be good at math. R has a lot of great built in commands. For example, it can be used as a calculator:

```
## Comment your code with number signs :)

(sqrt(62527) + (67*3275)) / 8.2
```

```
## [1] 26789.64
```

```
# assign variables with <-
someNum <- (sqrt(62527) + (67*3275)) / 8.2
```

For more information on why I use `<-` to assign variables instead of `=` (even though both will work...), see this blog post.

Some examples of commands that I use every day:

```r
# keep a seperate working directory for every project!

# creating and assignming a vector to a variable
x <- c(4,75,89,12, 9)
y <- c(1:5)
# set seed for reproducibility
set.seed(123)
# random, normally distributed of length 10
z <- c(rnorm(5))

# bind by columns, create matrix
mat <- cbind(x,y,z)
# bind by rows, create matrix
mat2 <- rbind(x,y,z)
print(mat)
```

```
##       x y          z
## [1,]  4 1 -0.56047565
## [2,] 75 2 -0.23017749
## [3,] 89 3  1.55870831
## [4,] 12 4  0.07050839
## [5,]  9 5  0.12928774
```

```r
print(mat2)
```

```
##         [,1]       [,2]      [,3]        [,4]      [,5]
## x  4.0000000 75.0000000 89.000000 12.00000000 9.0000000
## y  1.0000000  2.0000000  3.000000  4.00000000 5.0000000
## z -0.5604756 -0.2301775  1.558708  0.07050839 0.1292877
```

```r
# dimensions of matrix
dim(mat)
```

```
## [1] 5 3
```

```r
# get column 2
mat[,2]
```

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

```r
# get row 3
mat[3,]
```

```
##         x         y         z
## 89.000000  3.000000  1.558708
```

```r
# Data frames are similar to matrices, except you are easily able to mix data types
newDF <- data.frame(hello=x, weird=y, fun=c("one", "two", "three", "four", "five"))
print(newDF)
```

```
##   hello weird   fun
## 1     4     1   one
## 2    75     2   two
## 3    89     3 three
## 4    12     4  four
## 5     9     5  five
```

```
which(newDF < 1)
```

```
## Warning in Ops.factor(left, right): '<' not meaningful for factors
```

```
## integer(0)
```

```
which(newDF$hello == 89)
```

```
## [1] 3
```

Some fucntions only work on data with a certain structure of class. You can check your data's structure/class by:

```
str(mat) # gives structure information
class(mat) # gives class information
dataNum <- as.numeric(mat)
dataChar <- as.character(mat)
df <- as.data.frame(mat)
newMat <- as.matrix(newDF)
```

Important note

Get into the habit of having a seperate working directory for every project. Nothing makes me cringe more than helping someone and seeing them use their Desktop as their working directory for EVERYTHING. Create a file folder/directory for each project to stay organised!

```
# setting your working directory

setwd("~/some/file/path")
```

Using data in R

There are 3 different ways you can use/find data in R

1. Use built-in datasets
2. Download them from the web
3. Import your own CSVs

R has a lot of built-in datasets. You can access them like:

```
data(cars)
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

```
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

PS. easy keyboard shortcut to inserting code chunks in R Markdown is Ctrl + Alt + I (OS X: Cmd + Option + I)

The interwebs also have a lot of R datasets you can play around with....

```
library(data.table)
mydat <- fread('http://www.stats.ox.ac.uk/pub/datasets/csb/ch11b.dat')
head(mydat)
```

```
##    V1  V2   V3    V4 V5
## 1:  1 307  930 36.58  0
## 2:  2 307  940 36.73  0
## 3:  3 307  950 36.93  0
## 4:  4 307 1000 37.15  0
## 5:  5 307 1010 37.23  0
## 6:  6 307 1020 37.24  0
```

Finally, you can also import your own data...

```
data <- read.csv("file/path/data.csv", row.names=1,
header=FALSE)
# writing a csv is similar
write.csv(data, file="~/file/path/2/write/data.csv", sep=",")
```

Installing/loading packages

There are 2 main locations for R packages that you'll be using: CRAN and Bioconductor. Bioconductor has more "bioinformatic"-y packages.

```
# Installing from CRAN
install.packages('ggplot2') #notice the quotes!
library(ggplot2) # notice NO quotes!


# Installing from Bioconductor
source("https://bioconductor.org/biocLite.R")
biocLite() # this installs/updates bioconductor
biocLite('WGCNA') # this will install packages, notice the quotes
library(WGCNA)
```

But what if you don't know how to use the packages...

R Packages come with PDFs and help files. The good packages come with full-blown tutorials. One of the best things about these help files is that, as long as you've installed the package, they are stored locally and do not require internet access. To access these files you can:

```
library(ggplot2)
?ggplot
help(ggplot)
vignette("parallel")
```

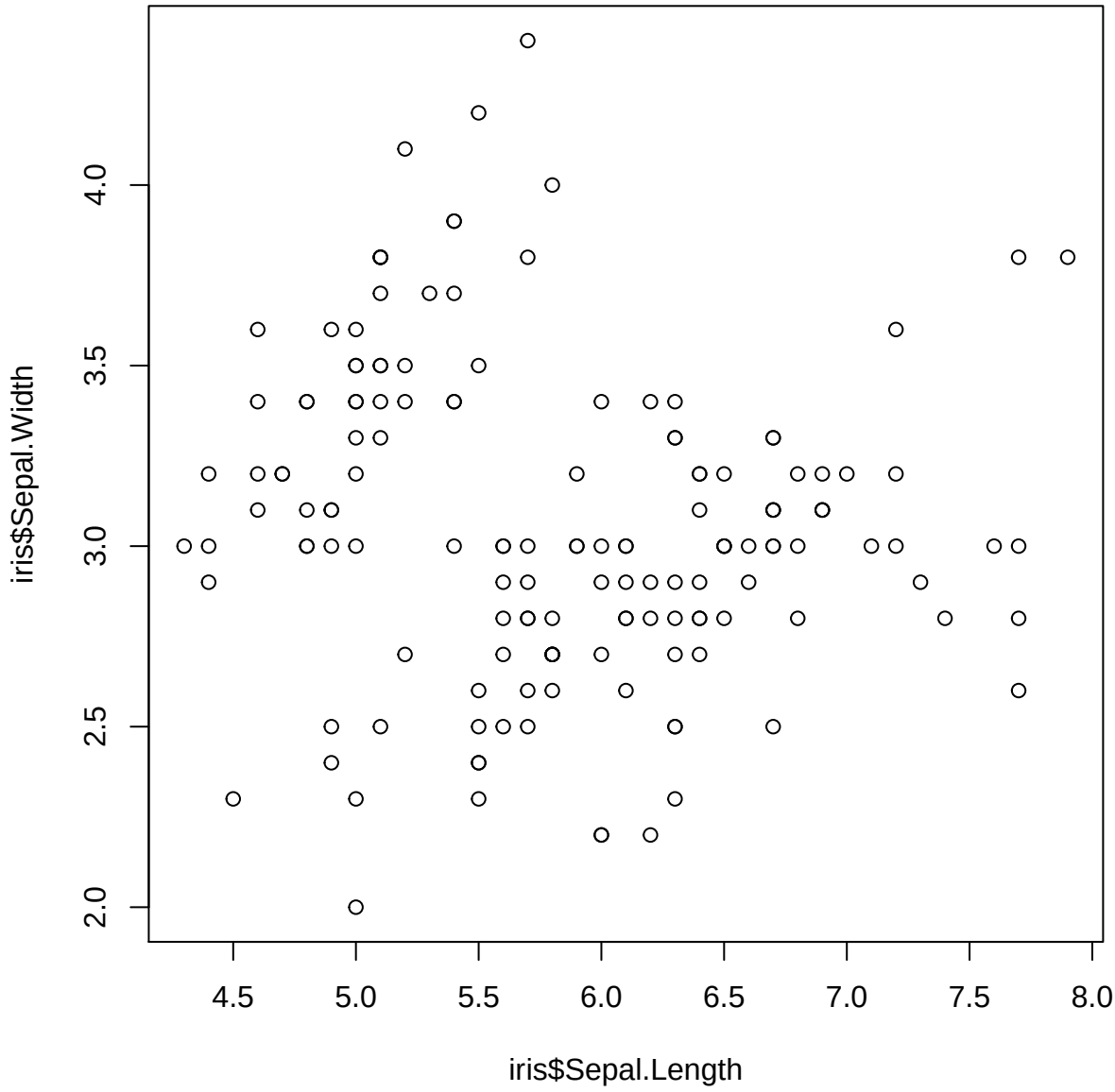
```
## What if you need a function, but forget what package it's in?

??predict #"??"" searches R help files!
```

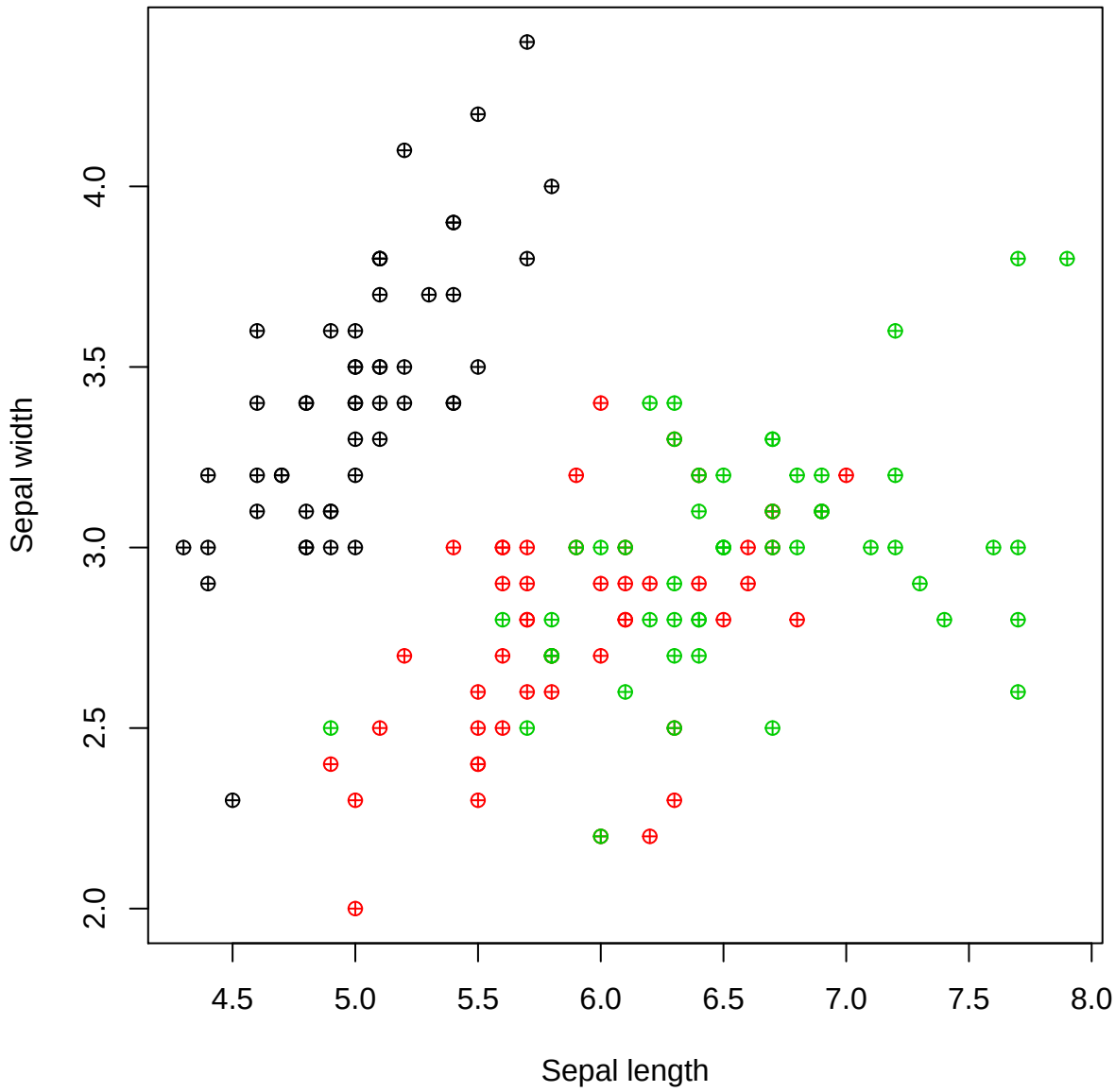
Plotting (you don't need ggplot!)

You can use R to create basic plots. You can still make them pretty, as we'll see during our linear regression session.

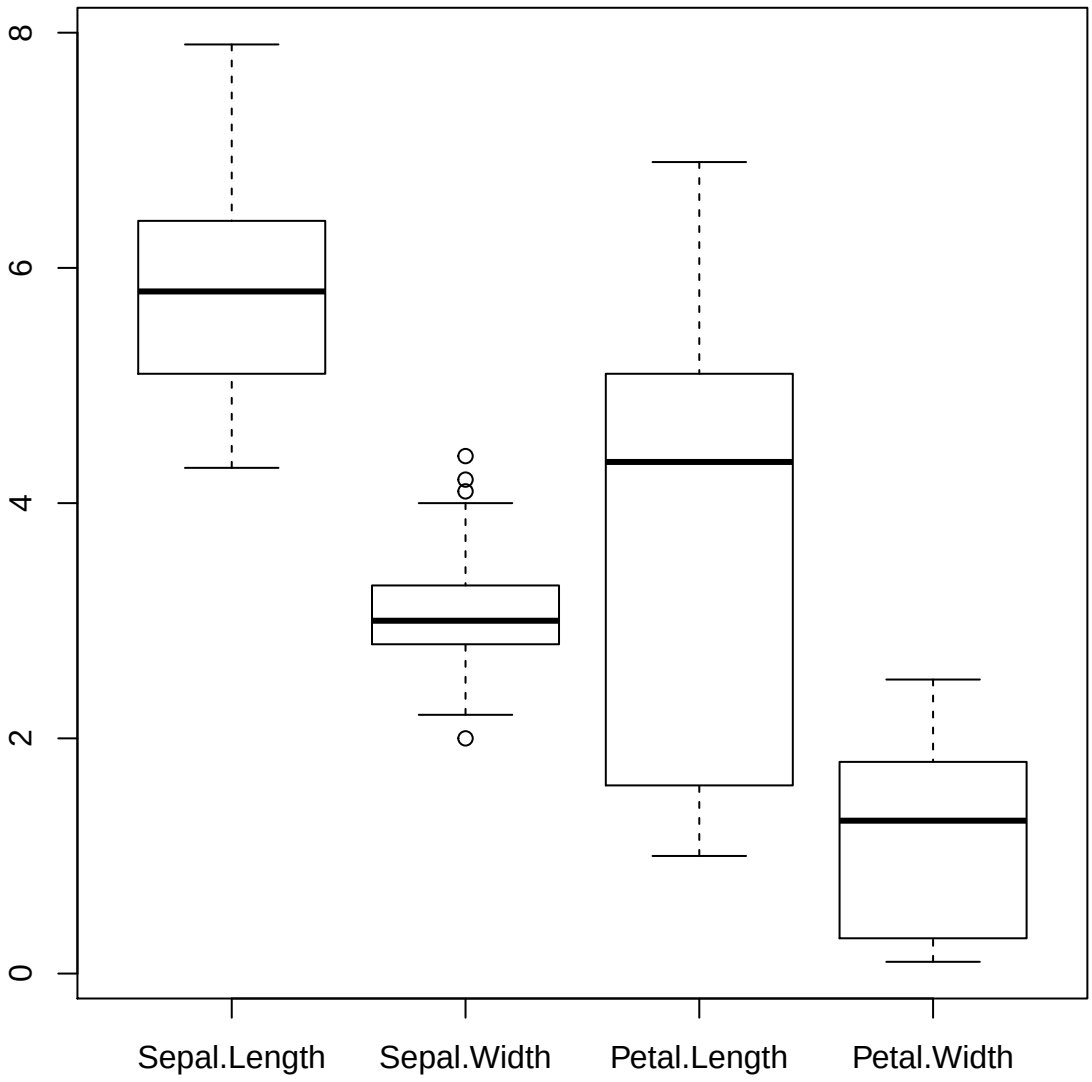
```
data(iris)
plot(iris$Sepal.Length, iris$Sepal.Width)
```




```
plot(iris$Sepal.Length, iris$Sepal.Width,
     pch=10, col=iris$Species,
     xlab = "Sepal length", ylab = "Sepal width")
```




```
boxplot(iris[,1:4])
```


0
2
4
6
8
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width