

A (Very) Short Introduction to R for Wet Lab Scientists

Kieran O'Neill

Introduction

RStudio

R Commands and Objects

Installing Add-on Packages

Loading in Files

Basic Statistical Tests

Basic Plots

Credits

Introduction

Course Logistics

- ▶ This is a very brief introduction to a very large topic.
- ▶ There will be at least one follow-up workshop going into more depth.
- ▶ Depending on demand (and our time), we may create more.
- ▶ You can find all of the slides at https://github.com/oneillkza/R_Workshops
- ▶ In the last slide are links to materials from some excellent and much longer courses.
- ▶ Please ask questions as we go!

What is R?

R is a versatile, open source programming language that was specifically designed for data analysis. As such R is extremely useful both for statistics and data science. Inspired by the programming language S.

- ▶ Open source software under GPL.
- ▶ Superior (if not just comparable) to commercial alternatives. R has over 5,000 user contributed packages at this time. It's widely used both in academia and industry.
- ▶ Available on all platforms.
- ▶ Large and growing community of peers.
- ▶ Bioconductor: largest (and arguably the best) free collection of software for biological data analysis anywhere.

Why Not Just Use Excel, FlowJo, GraphPad, etc?

1. Reproducibility

- ▶ Its really important that you know what you did.
- ▶ More journals/grants/etc. are also requiring this.
- ▶ The best way to know what you did is to provide all the code.
- ▶ GUI software makes this difficult
- ▶ If you keep a lab notebook, why not do the same thing with your analysis?

2. Flexibility, capabilities and pretty pictures

- ▶ R can handle much larger data sets, much faster, and much more easily than Excel.
- ▶ Huge range of statistical tests, biological data types, etc.
- ▶ Plotting in R is far more sophisticated than any available GUI.

Proof of What I Mean By Pretty Pictures:



Figure 1: Gerstung et al (2015) Nature Communications (CC-BY)

RStudio

Set up a new project

- ▶ Click 'File', then 'New project'
- ▶ Click 'New directory' then 'Empty Project', then pick a directory
- ▶ With the project set up, click 'File', then 'New' (or ctrl+shift+n)
- ▶ Click 'File', 'Save' (ctrl+s)
- ▶ Save the file as something meaningful, like `lecture1_examples.R`

Note: for Mac users, where I say 'ctrl', use your weird Mac command key instead.

Quick overview of RStudio

The image shows the RStudio interface with several annotations in red text:

- File editing**: Points to the source editor window showing R code for vector manipulation and data frames.
- Environment (and other things)**: Points to the Environment pane on the right, which lists objects like 'iris', 'iris_dat', and 'dismo'.
- Plots, help, file view**: Points to the Plots pane on the right, which displays three box plots for the 'setosa', 'versicolour', and 'virginica' species.
- Console**: Points to the Console pane at the bottom, which shows the output of R commands and warnings.

The source editor code includes comments and functions for accessing vectors, using ranges, and creating data frames. The Environment pane shows the 'Data' tab with a list of objects and their attributes. The Plots pane shows three box plots for the 'setosa', 'versicolour', and 'virginica' species. The Console pane shows the output of R commands, including warnings about package installation.

Figure 2: RStudio Interface

Working between the script and console

*Type the following into the console, and press enter:

```
print("Hello")
```

```
## [1] "Hello"
```

- ▶ Now type it into the file window, and with the cursor on that line, press ctrl+enter
- ▶ Messing around in the console is fun.
- ▶ But it's better to keep your work in a file which you save often.

R Commands and Objects

Objects

You can assign values to objects:

```
some_number <- 5  
some_number + 3
```

```
## [1] 8
```

```
some_other_number <- some_number ^ 3  
some_other_number
```

```
## [1] 125
```

Take a look in your environment window in RStudio. You can also see what objects are defined using the `ls()` command:

```
ls()
```

```
## [1] "some_number"      "some_other_number"
```

Basic Data Types

You can find out the type of an object using `typeof()`:

```
typeof(some_number)
```

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

```
some_text <- "5"  
typeof(some_text)
```

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

Numeric vs Character

```
some_number + 5
```

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

```
some_text + 5    # This would give an error -- try it.
```

```
as.numeric(some_text) + 5
```

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

Vectors

Vectors are one-dimensional objects. You can create them with the `c()` function:

```
my_vector <- c(1,3,5,6,7,8)
```

You can apply operations to a whole vector.

```
my_vector^3
```

```
## [1] 1 27 125 216 343 512
```

You can join vectors with `c()`:

```
c(my_vector, my_vector^3, my_vector^4)
```

```
## [1] 1 3 5 6 7 8 1 27 125 216 343 512 1 81
## [15] 625 1296 2401 4096
```


Accessing Vectors

To get to parts of a vector, you use square bracket notation: `[]`

```
my_vector[3]
```

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

You can use vectors to access a vector:

```
my_vector[c(1,3,5)]
```

```
## [1] 1 5 7
```

You can also use ranges:

```
my_vector[2:4]
```

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

Data Frames

Data frames are special objects in R for storing mixed data. You can think of them as something like a sheet in Excel. We're going to load in one of R's default data sets, a series of esophageal cancer vs smoking data. We'll use the `head()` function to just view the first few rows.

```
data(esoph)
head(esoph)
```

##	agegp	alcgp	tobgp	ncases	ncontrols
## 1	25-34	0-39g/day	0-9g/day	0	40
## 2	25-34	0-39g/day	10-19	0	10
## 3	25-34	0-39g/day	20-29	0	6
## 4	25-34	0-39g/day	30+	0	5
## 5	25-34	40-79	0-9g/day	0	27
## 6	25-34	40-79	10-19	0	7

Also try clicking on `esoph` in the Environment window in RStudio.

Working With Data Frames:

You can access columns in a data frame using \$, or rows, columns, or individual values using []

```
head(esoph$agegp)      # column
```

```
## [1] 25-34 25-34 25-34 25-34 25-34 25-34  
## Levels: 25-34 < 35-44 < 45-54 < 55-64 < 65-74 < 75+
```

```
head(esoph[, 'agegp']) # column using []
```

```
## [1] 25-34 25-34 25-34 25-34 25-34 25-34  
## Levels: 25-34 < 35-44 < 45-54 < 55-64 < 65-74 < 75+
```

Working With Data Frames (ctd):

```
esoph[2,]           # row
```

```
##   agegp      alcgp tobgp ncases ncontrols  
## 2 25-34 0-39g/day 10-19      0         10
```

```
esoph[2,'agegp']     # Single value
```

```
## [1] 25-34  
## Levels: 25-34 < 35-44 < 45-54 < 55-64 < 65-74 < 75+
```

```
esoph[2,1]           # Single value using numbers
```

```
## [1] 25-34  
## Levels: 25-34 < 35-44 < 45-54 < 55-64 < 65-74 < 75+
```

Another Useful Function: Summary()

```
summary(esoph[,3:5])
```

##	tobgp	ncases	ncontrols
##	0-9g/day:24	Min. : 0.000	Min. : 1.00
##	10-19 :24	1st Qu.: 0.000	1st Qu.: 3.00
##	20-29 :20	Median : 1.000	Median : 6.00
##	30+ :20	Mean : 2.273	Mean :11.08
##		3rd Qu.: 4.000	3rd Qu.:14.00
##		Max. :17.000	Max. :60.00

Installing Add-on Packages

CRAN

- ▶ Most of R's power comes from free third-party add-ons
- ▶ CRAN is the Comprehensive R Archive Network
- ▶ It is the main repository for R packages
- ▶ You can install packages like so:

```
install.packages('beeswarm')
```

When you start a new session, you can then load a package using `library`:

```
library('beeswarm')
```

Bioconductor

- ▶ Bioconductor is a big part of what makes R awesome for biologists.
- ▶ Bioconductor is a repository specifically for (molecular) biology R packages.
- ▶ It has very stringent rules for those packages regarding documentation, examples and code quality.
- ▶ There are packages to handle a vast range of data, from BAM files to microarrays to flow cytometry and many more.

Check it out at www.bioconductor.org

To install Bioconductor packages (note: don't run this now, it can take ten minutes or more the first time):

```
source("http://bioconductor.org/biocLite.R")  
biocLite('flowCore') #Or whatever the package is called.
```


Loading in Files

Edgar Anderson's Iris Data Set

Provides the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris.



Edgar Anderson's Iris Data Set (ctd)

Let's load the Iris data set.

```
data(iris)
iris_dat <- iris
```

`str()` and `colnames()` the object as a sanity check.

```
str(iris_dat, max.level=0)
```

```
## 'data.frame':    150 obs. of  5 variables:
```

```
colnames(iris_dat)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

What are the species of iris in this data set?

```
levels(iris_dat$Species)
```

```
## [1] "setosa"      "versicolor" "virginica"
```

Creating directories and downloading data

In many situations, we have to download our data to a place we can use. We can use `dir.create()` to create a data directory in our project directory.

```
dir.create("data")
```

Next, we can download the data for this exercise.

```
download.file("http://ateucher.github.io/rcourse_site/data/iris.csv",  
             destfile = "data/iris.csv")
```

Loading in CSV

The simplest way to input and output data is in the form of comma separated files. Comma separated files, which have the suffix `.csv`, are recognised by almost all statistical and spreadsheet programs including R and Excel.

To load comma separated files in R:

```
iris_dat <- read.csv("data/iris.csv")
```

Loading in from Excel

Unfortunately, there is no base package support for importing data directly from MS Excel. You could save it in another format, THEN import this new file.

Alternatively, you could use the gdata package.

```
# install gdata and load as dependency  
install.packages("gdata")  
library(gdata)  
  
# load data  
iris_dat <- read.xls("data/iris.xls")
```

Basic Statistical Tests

Student's t-test

Using the Iris data set, let's find out if the difference in sepal length between two species is significant.

`subset()` data frame into *Iris versicolor* and *virginica*.

```
versicolor <- subset(iris_dat, iris_dat$Species == "versicolor")  
virginica <- subset(iris_dat, iris_dat$Species == "virginica")
```

We can use `t.test()` to answer our question.

```
t.test(versicolor$Sepal.Length, virginica$Sepal.Length)
```


Examining the Results

```
t.test(versicolor$Sepal.Length, virginica$Sepal.Length)
```

```
##  
## Welch Two Sample t-test  
##  
## data: versicolor$Sepal.Length and virginica$Sepal.Length  
## t = -5.6292, df = 94.025, p-value = 1.866e-07  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.8819731 -0.4220269  
## sample estimates:  
## mean of x mean of y  
## 5.936 6.588
```

You can also assign the above to an object and extract only the p-value.

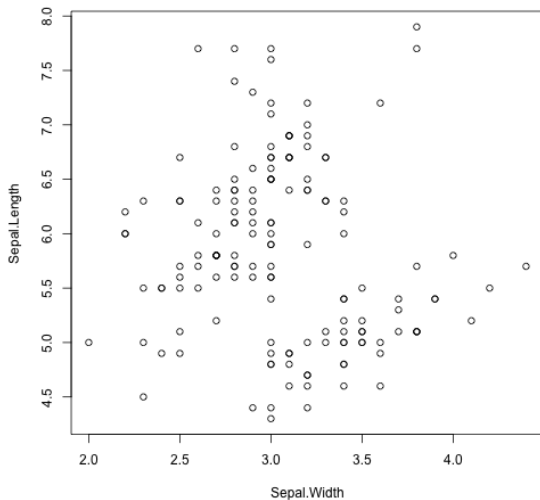
```
iris_test <- t.test(versicolor$Sepal.Length, virginica$Sepal.Length)  
paste("p-value:", iris_test$p.value)
```

```
## [1] "p-value: 1.866144387377e-07"
```

Basic Plots

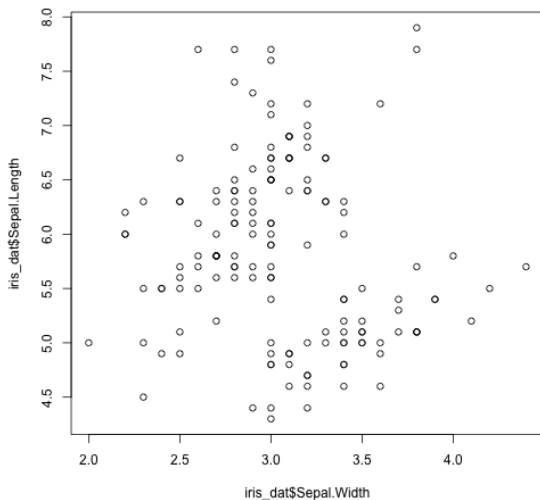
Scatter Plot

```
data(iris)
iris_dat <- iris
plot(Sepal.Length~Sepal.Width, data=iris_dat)
```



Scatter Plot, Alternate Way of Calling

```
plot(iris_dat$Sepal.Width, iris_dat$Sepal.Length)
```



Scatter Plot, With Some Options

```
plot(iris_dat$Sepal.Width, iris_dat$Sepal.Length,  
     pch=16,  
     col=iris$Species,  
     main='Sepal Length vs Sepal Width',  
     xlab='Length',  
     ylab='Width')
```

Scatter Plot, With Some Options

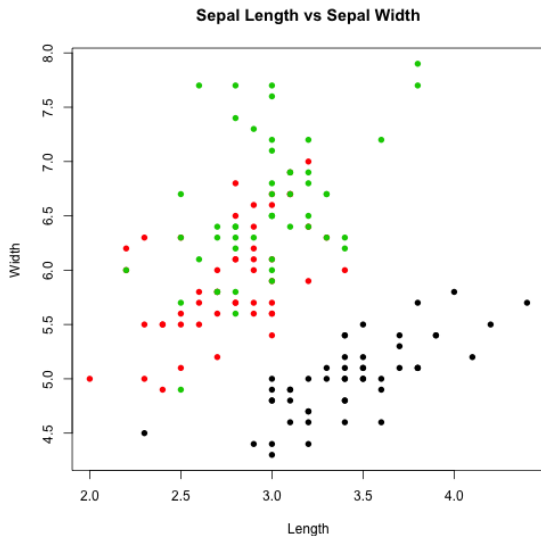


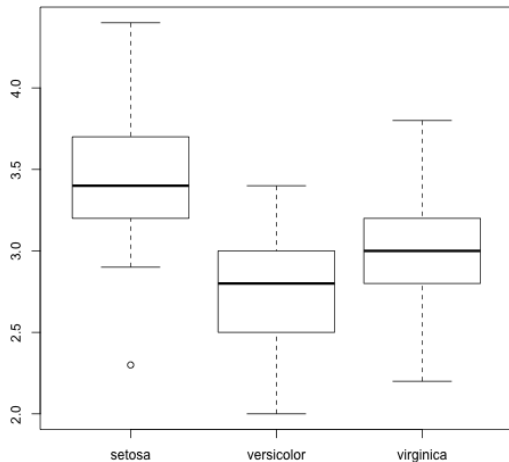
Figure 6:plot of chunk unnamed-chunk-36

Dynamite Plots

- ▶ A lot of papers use bar plots with error bars to show data with multiple measurements per treatment.
- ▶ These have a lot of shortcomings: data being hidden, assumptions about the confidence intervals used, and wasted ink.
- ▶ Unsurprisingly, R does not have an way to to these.
- ▶ Instead, R does allow box plots, which are much better.
- ▶ There is also a package for beeswarm plots.

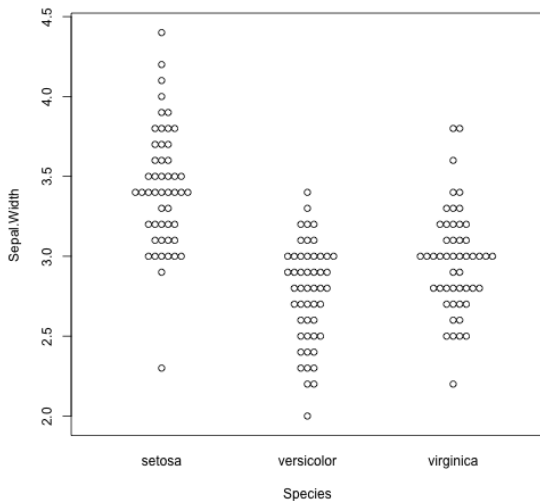
Box Plots

```
boxplot(Sepal.Width~Species, data=iris_dat)
```



Beeswarm Plots

```
library(beeswarm)
beeswarm(Sepal.Width~Species, data=iris_dat)
```



Beeswarm Plots With More Options

```
beeswarm(jitter(Sepal.Width)~Species, data=iris_dat,  
         method='swarm',  
         col=rainbow(3),  
         pch=16)
```

Beeswarm Plots With More Options

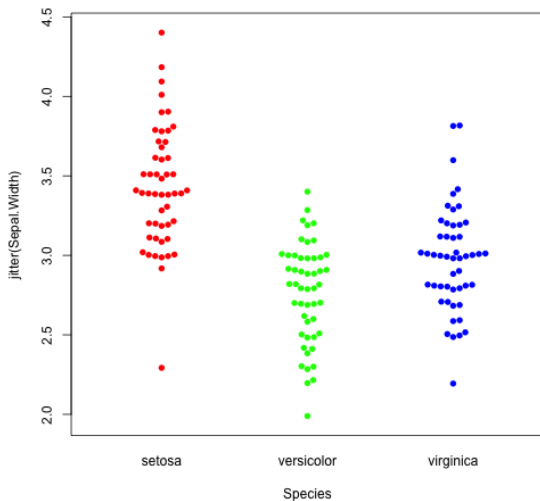


Figure 9: plot of chunk unnamed-chunk-40

Beeswarm and Boxplots Combined

```
boxplot(Sepal.Width~Species, data=iris_dat)
beeswarm(jitter(Sepal.Width)~Species, data=iris_dat,
         method='swarm',
         col=rainbow(3),
         pch=16,
         add=TRUE)
```

Beeswarm and Boxplots Combined

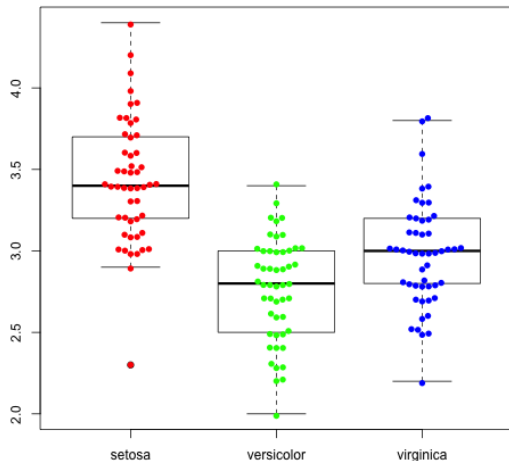


Figure 10: plot of chunk unnamed-chunk-42

Other Plotting Packages - ggplot2

```
# install ggplot2 and load dependency
install.packages("ggplot2")
library(ggplot2)

# plot
ggplot(iris_dat, aes(Sepal.Length, Sepal.Width)) +
  geom_point() +
  theme_bw() +
  xlab("Sepal length (cm)") +
  ylab("Sepal width (cm)") +
  ggtitle("Sepal width vs. sepal length in Iris data set") +
  facet_grid(Species~.)
```

Other Plotting Packages - ggplot2

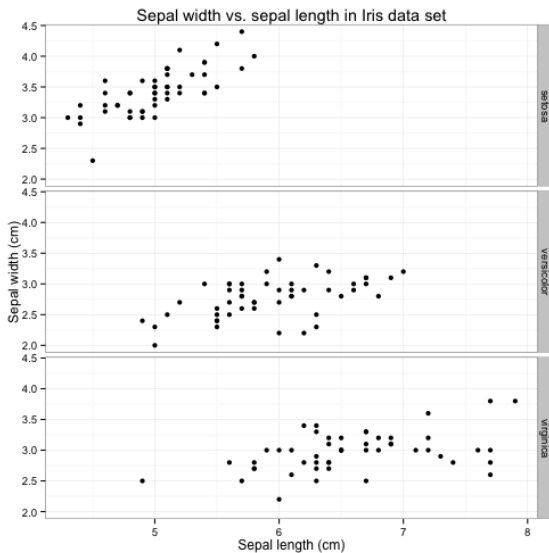


Figure 11: plot of chunk unnamed-chunk-45

Credits

This Workshop Brought to You By...

Course Developers:

- ▶ Kieran O'Neill
- ▶ Eva Yap
- ▶ Alice Zhu (for next session)

Starting Material:

Much material was reused from Software Carpentry's Bootcamp workshops and from Andy Teucher's short R course, both under the terms of the Creative Commons Attribution License.

Pizza and Logistics:

- ▶ GraSPoDS (especially Eva Yap and Jessica Pilsworth)

License:

You are free to download, copy and modify this work in accordance with the Creative Commons Attribution License.