Before the workshop

- Download Data Files: https://github.com/julipetal/Intro_to_R

Introduction to R

Part I

Part I

Please interrupt with questions

Basic Introduction

Good Coding Practice

Data Handling

Basic Data Visualization

Basic Introduction

What is R?

Free software environment for statistical computing and graphics

Runs on most UNIX platforms, Windows and MacOS

Website: http://www.r-project.org/

CRAN - Comprehensive R Archive Network

Popular in academia, and growing software in industry

Frequently releases updated versions with funny names (like ‘Bug in Your Hair’), check every 6 month or so to ensure you are using up-to-date version

Tons of different libraries (of which some do the same thing)

R code

Any R-script (fileName.r) can be opened in an editor of your choice

- For windows: Notepad++ is good because it highlights different attributes of the code, unlike Notepad

Copy and past R-code from scripts into the R window, also called R console

Directly write your code into the R window

Using UP and DOWN arrows will reveal lines of code that have been typed before within the current R history

To execute a line of code press enter

Basic Math Functions

- Standard math operations and functions
  - +, -, *, /
  - `exp( ), log( ), sin( ), …`
- Assigning numbers to variables
  - `> X = 65`
  - `> x = 43`
  - `> X + x`
  - You will also see <- instead of an equal sign, it does the same thing
  - `> y <- 2`

NOTE: R is case sensitive

- **R** Reference Card: https://cran.r-project.org/doc/contrib/Short-refcard.pdf

Help!!!

```
> help.start()
```

```
> help(<functionName>)
```

Replace `<functionName>` with `which`

```
> ?<functionName>
```

Again try `which` and then `for`

```
> help.search("<functionName>")
```

```
> example(<functionName>)
```

```
> demo()
```

```
> demo(graphics)
```

Easy examples and explanations for basic R functions

http://www.statmethods.net/

RStudio as an editor

https://www.rstudio.com/products/rstudio/download/#download

Closing an R Session/Console

To close a session

```
> q()
```

R will promote you if you want to save your session

If you kept a script and saved important data to file, there is no need to save the session.

Otherwise you might want to say yes. If you do, an .Rhistory file will be created. This will overwrite any previously created file if not renamed. The .Rhistory file includes all lines of code without output and often includes nonfunctional code.

Better practice is to keep a clean script file with comments.

Good Coding Practice

Good Practice

Save work and code by using scripts

Keep these scripts clean

- Avoid saving code that did not work or produced unwanted results
- Avoid saving R output within these script files (.r)
- Comment your code using #
  - Write a bullet point of what the line of code does for future reference

Find a way to structure and organize your scripts and data

Save often ... save again

Open your R window in a work directory away from backups and original files - you do not want to accidently overwrite them. R does not ask if “you are sure”.

Script files - your notebook of code

Here is an example of a script file viewed in Notepad++

Comments start with #

R recognizes # and will not try to execute these lines. Also also notepad++ highlights lines starting with # green making it easy to spot.

```
# Checking for zero data entries
length(as.vector(which(as.vector(expM)==0)))
# [1] 419362

### MM file created, checked if all measured were included
### split MM into 6 files for computational reasons

library(petal)
### first sub-matrix done step-wise to double-check output
m1 = readinTable("smallMM_1.txt")
vec = as.numeric(m1[,3])
indexRM = as.vector(which(abs(as.numeric(m1[,3]))<0.5))
newCurrentMM = m1[-indexRM,]
rm(indexRM, m1)

### integrating remaining 5 files automatically
for(i in 2:6){
    currentMM = readinTable(paste("smallMM_",i,".txt",sep=""))
    tmpVec = as.numeric(currentMM[,3])
    vec = c(vec, tmpVec)
    indexRM = as.vector(which(abs(as.numeric(currentMM[,3]))<0.5))
    tmpnewCurrentMM = currentMM[-indexRM,]
    newCurrentMM = rbind(newCurrentMM, newCurrentMM)
    rm(currentMM, tmpVec, indexRM, tmpnewCurrentMM)
}

### checking distribution of association measure (Spearman)
measures = vec
min(measures, na.rm = TRUE) - 0.5, digits = 0)
maxExp = round(max(measures, na.rm = TRUE) + 0.5, digits = 0)
png(filename = "Histogram_MM.png"), width = 7, height = 4.25, units = "in", pointsize = 12, bg = "white", res = 600)
h = hist(measures, breaks = seq(-1, 1, by = 0.05))
dev.off()

min(measures)
# [1] -0.872
max(measures)
# [1] 1
dim(newCurrentMM)
# [1] 33132192        3

### saving output
write.table(newCurrentMM,file = "MM_reduced.txt", sep = "\t", quote=FALSE, row.names = FALSE, col.names = TRUE)
save(newCurrentMM, file="newCurrentMM.RData")
```

Directory

Directory is just another word for folder

Make sure you know in which directory your R window is

```
> getwd()

> setwd("C:/ <TAB> )

> getwd()
```

You are learning a new language

Syntax

- What is the difference between f(x) versus f[x]?

Vocabulary

- What functionality is part of R and what do I have to code myself?
- What are these functions called?

Thinking in that language rather than translating

- “Did you go to school?” Versus “Did you to school go?” (German)

Do not try to memorize everything - write scripts and know where to find them.

Variables

- Any letter, sequence of letters, or letters followed by a number can be used as variable names
  - `X, xx, myData, myData1, myData2`
- A variable name cannot start with a number
- Short variable names are great for typing, long variable names often make it easier to understand the code at a later time
- Do not overwrite R's predefined functions, in most cases if you try an error message will appear
- <Tab> autocompletes variable name, function names, or filenames - very handy
- R is case sensitive

Data Handling

Indexing, Simple Manipulation, Uploading, Investigation, Exporting data

Vectors & Operations & Functions

A vector is an ordered sequence of numbers

Multiple ways to make a vector

```
> x = c(5, 64, 7, 3)
> y = 4:7
> z = seq(4,7)
> t = seq(1, 11, by=2)
> u = seq(1, 11, by=2)
> tmp = rep(10, 6)
```

```
> x = c(5, 64, 7, 3)
> y = c(34, 65, 0, 7)
> new1 = cbind(x,y)
> new2 = rbind(x,y)
> x+y
> x^2
> x[3]
> x[10]= 4
```

```
> length(x)
> sum(x)
> sum(x)/length(x)
> mean(x)
```

Indexing and Simple Manipulations

R has some integrated dataset available

```
> data()
> women
> dim(women)
> rownames(women)
> colnames(women)
> women[1,1]
> women[1,]
> women[,2]
> women[1:4,1:2]
```

```
> women$height
> meanh = mean(women[,1])
> meanw = mean(women[,2])
> hmin = min(women[,1])
> hmax = max(women[,1])
> table(women)
> hist(women[,1])
> hist(women[,1], breaks = seq(57, 73, by=1))
```

With this many trees I cannot see the forest

```
> data()

> dim(trees)

> colnames(trees)

> rownames(trees)

> trees[1:4,]

> trees[12,]

> trees_mod = trees

> trees_mod[12,2] = 79
```

```
> numCherry = rep(150, 31)

> new_trees = cbind(trees_mod, numCherry)

> colnames(new_trees)

> colnames(new_trees[4]) = “Number of
Cherries”

>
```

Investigating Data

```
> range(new_trees)

> range(new_trees[,2])

> colnames(new_trees)

> index = as.vector(which(new_trees[,3]>=70))

> new_trees[index,]

> table(new_trees[,3])

> hist(new_trees[,3])

> abline(v = 50, col="red", lwd = 2)

> sort(new_trees[,3])
```

Want to know how many tree have a height of 70 and above

Correlation Analysis and Standard Stats

```
> cor(new_trees[,2:3], method="spearman")
> cor(new_trees[,2:3], method="pearson")
> corM = cor(new_trees[2:3], method="spearman")
> corM[1,2]
> plot(new_trees[,2], new_trees[,4])
> barplot(new_trees[,3])
```

```
> mn = mean(new_trees[,3])
> md = median(new_trees[,3])
> vr = var(new_trees[,3])
> st = sd(new_trees[,3])
```

LET'S TAKE A BREAK

Data Upload

Function	
`read.csv()`	Convenient function for .csv files Returns a list-object
`read.table()`	Any delimited text files can be read-in, can be clunky at times, Returns a list-object
`scan()`	Reads file line by line Returns a vector-object

```
read.csv(data.csv, header = TRUE)
```

```
read.table("data.txt", header=TRUE,
                                sep="\t")
read.table("data.txt", header=FALSE,
                                sep="\t")
```

```
scan("data.txt", what="", sep="\t",
                    nline=1, skip=1)
scan("data.txt", what="", sep="\t",
                      n=15, skip=1)
```

Data Upload Script

```
fileName = ""

# make sure the file is in table format
# line 1 should have the same length as line 2

l1 = scan(fileName, sep = "\t", what = "", nlines = 1)

tmpM= matrix(scan(fileName, what = "", sep = "\t", skip = 1), ncol = length(l1), byrow = TRUE)
myData= matrix(as.numeric(tmpM[, -1]), ncol = length(l1) -1, byrow = FALSE)
colnames(myData) = l1[-1]
rownames(myData) = tmpM[, 1]

# remove variables (objects) you no longer need
rm(fileName, tmpM, l1)

# check if the file was read in correctly
dim(myData)
myData[1:4,1:3]

# myData is a numeric matrix object
```

Exporting Data

```
> write.table(myData, "myDataM.csv", quote=FALSE, row.names=TRUE,
                                       col.names=FALSE, sep=",")
> write.table(myData, "myDataM.txt", quote=FALSE, row.names=FALSE,
                                       col.names=TRUE, sep="\t", append=FALSE)
```

Objects can also be saved as `.RData` files

```
> save(new_trees, file="trees.RData")
> q()
> load("trees.RData")
> objects()
```

Can save more than one object
Good coding tip: give the objects meaningful names, not `x` and `y`

```
> save(x, y, file="dataObjects.RData")
```

Data Visualization

Histograms and Scatter Plots

Histogram - hist()

```
> hist(myData[,1])

> min(myData)

> max(myData)

> hist(myData[,1], breaks=seq(2,14.5,by=0.5))

> hist(myData[,1], breaks=seq(2,14.5,by=0.5), col="pink")

> hist(myData[,1], breaks=seq(2,14.5,by=0.5), col="pink", xlab="Control",
main="Values for Control Group")
```

Histogram - hist()

```
# number of histogram's subintervals
numBins = 100

# MIN and MAX will always be integers
MIN = round(min(myData[,1])-0.5, digits=0)
MAX = round(max(myData[,1])+0.5, digits=0)

h = hist(myData[,1], breaks=seq(MIN, MAX, by = (MAX-MIN)/numBins),
col="green", xlab="Control", main="Values for Control Group")

# see the statistics of h
h

# barplot() versus hist()
barplot(myData[,1])
```

Histogram - hist()script

```
TITLE =
xAXIS =
COLOR =
vec =
numBins =                          # number of histogram's subintervals
fileName =

# MIN and MAX will always be integers
MIN = round(min(vec)-0.5, digits=0)
MAX = round(max(vec)+0.5, digits=0)

png(filename = paste(fileName, ".png", sep=""), width=7, height=8.5,
units="in", pointsize=12, bg="white",res=600)
h = hist(vec, breaks=seq(MIN, MAX, by=(MAX-MIN)/numBins), col=COLOR,
xlab=xAXIS, main=TITLE)
dev.off()
```

Plotting two graphics intro one window

```
vec = as.vector(myData)

MIN = round(min(vec)-0.5, digits=0)
MAX = round(max(vec)+0.5, digits=0)


par(mfrow = c(2, 1))

hist(myData[,1], breaks=seq(MIN, MAX, by = (MAX-MIN)/numBins), col="green",
xlab="Control", main="Values for Control Group")

hist(myData[,2], breaks=seq(MIN, MAX, by = (MAX-MIN)/numBins), col="red",
xlab="Treatment", main="Values for Treatment Group")
```

Scatterplot - plot()

```
> plot(myData[,1], myData[,2])
> plot(myData[,1], myData[,2], ylab="Treatment", xlab="Control",
main="Drug1")
```

Changing circles to dots, `pch=` has a number of different plotting symbol options

```
> plot(myData[,1], myData[,2], ylab="Treatment", xlab="Control",
main="Drug1",
                                                        pch=19)
```

Making dots smaller by setting `cex=`

```
> plot(myData[,1], myData[,2], ylab="Treatment", xlab="Control",
main="Drug",
                                                        pch=19, cex=0.5)
> plot(myData[,1], myData[,2], ylab="Treatment", xlab="Control",
main="Drug",
                                                        pch=19, cex=0.3)
```

Scatterplot - plot()

Adding a regression line `lm(y~x)`

```
> abline(lm(myData[,2]~myData[,1]), col="blue")
```

Changing thickness of regression line

```
> abline(lm(myData[,2]~myData[,1]), col="blue", lwd=2)
> abline(lm(myData[,2]~myData[,1]), col="blue", lwd=4)
```

Changing color of regression line

```
> abline(lm(myData[,2]~myData[,1]), col="maroon2", lwd=4)
```

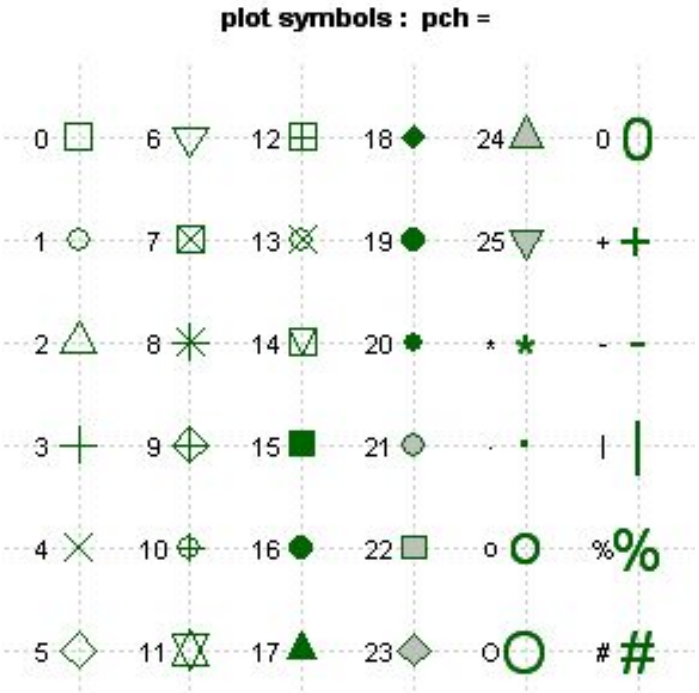
Adding some more lines just for fun

`v=` for vertical line

`h=` for horizontal line

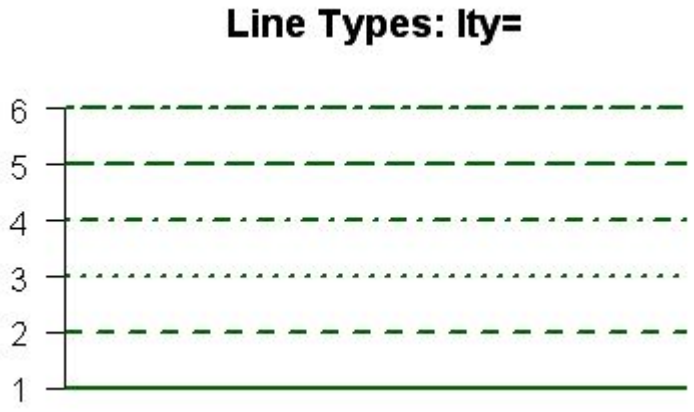
```
> abline(v=8, col="SpringGreen2", lwd=2)
> abline(h=12, col="DeepSkyBlue", lwd=6)
> abline(h=6, col="firebrick1", lwd=6, lty=4)
```

Plotting Symbols



Line Types

`lty`	line type
`lwd`	line thickness, default =1



http://www.statmethods.net/advgraphs/parameters.html