Example Markdown for Njal's Questions

David Hall

11/09/2020

Notes on reading this document

This document is writting in R markdown (more on that later). For now, there's three things to note. First text like this is to be treated like any other text you read, say from a course handout. Text in the grey box (see below) is the code you would write in R as code. You can literally copy, paste it into R, and run it as is and you should get the same outputs. Lastly, the outputs you'd observe in R are printed after the two hastags (##), see below.

```
# This is a comment, anythign after the '#' won't be interpreted by R.
# Use comments to comunicate ideas and concepts to make your code legibly

# This is the input box, i.e. the R code we ran
paste("This is the ouput box")
```

```
## [1] "This is the ouput box"
```

Part 1: R basics

Task 1: Subtract 3 from 5

```
3 - 5
```

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

R has numerous mathematical operators. These work the same as they would on your calculator. Table 1 below shows example of the other mathemical operators in R.

Table 1: Arithmetic Operators in R

Operator	Description	Example	... evaluates to
+	addition	1 + 2	3
-	subtraction	1 - 2	-1
	multiplication	3 * 2	6
^	exponent	3 ^ 3	27
/	division	7 / 3	2.333333
%%	modulus (remainder from division)	7 %% 2	1
%/%	integer division	7 %/% 3	2

Task 2: Make an object called "a" and assign this object a value of 5

```
a <- 5
a = 5
5 -> a

# All of the above work, but the '<-' notation is prefered due to legibility.

a # simply calling a variable will return it's value
```

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

There's an historic reason behing the `<-` notation, but in the current version of R, `=` and `->` work the same. **That being said, the preferred notation in R is `<-`** for legibility reasons. When you see something like `a <- 5` you read it as "I'm putting the value of 5 into the variable a". `<-` allows you to explicitly express to readers of your code which value goes where, wheras something like `a = b` is ambiguous to the unfamiliar reader. It could mean "a has the value of b" or "a is equal to b" (leading to confusion with `=` and `==`, which have very different meanings in R). *Even though you can don't use the `5 -> a`.*

##Task 3: Make an object called "b" and assign it a value of 3

```
# Using the prefered '<-' notation

b <- 3
```

In R studio take a look at the "Environment" window on the upper right panel. Here you'll see a list of all variables and functions in your current R session. You should see that "a" has a value of "5" and that "b" has a value of "3". You can now reference these values using using the variable names (see next task).

Task 4: Subtract "a" from "b"

```
b - a
```

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

Because both variables "a" and "b" are numeric, we can manipulate them using mathematical operators. 'b

Task 5: Overwrite "a" into a value of your choosing

```
a <- 10
a
```

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

The way to think of 'overwritting' is that a variable is a box. Only one type of thing can be in it (a number, a data.frame, etc.). When you use the `<-` notation you're "putting a value into the box". When you 'overwrite' a variable like we did above with `a <- 10` you're "kicking out the previously value from the box and placing the new value into it". This type of stuff is really useful later on when you're making functions and loops, but we'll cross that bridge later on.

Task 7: Overwrite "a" into a name of your choosing

a <- "test"

```
a <- "test" # Note the quotation marks
a
```

```
## [1] "test"
```

R can work with text, but it's called "strings". When you code in R, anything between quotation marks (i.e. "some text" or 'some other text') is interpreted as a string. In the above, we kicked out the value of 10 from "a" and inserted the value of "test". Now when you call "a" it'll return "test". Working with strings is important for a lot of data analysis later on, and there's plenty of strange tricks to it, but for now, remember that anything between quotation marks is a string, even if it has number.

```
a <- 10
is.numeric(a)
```

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

```
a <- "10"
is.numeric(a)
```

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

Task 8: Subtract b from a

```
a <- "test"
b <- 3

a - b
```

```
## Error in a - b: non-numeric argument to binary operator
```

Note that it doesn't work. The mathematical oeprators only work on numeric values (i.e. numbers like 1 or 3.14159...). You can't substract the value of "2" from the word "test".

Part 2: Functions & plotting

R can be used to manipulate numbers, but it's much more powerful than that. R comes pre-programmed with many functions that allow you do virtually anything with numbers and data. Functions are always defined as a description with no spaces, e.g., "examplefunction", followed by a set of parentheses. e.g., "examplefunction()". Within the parentheses the user specifies how the function is applied, based on the rules of that function.

```
# R can be used to manipulate numbers, but it's much more powerful than that. R comes pre-programmed wi

#Task 10: Let's find the mean of my.data. Try googling it.

#Task 11: Let's find the variance of my.data ... again, try googling it to find out how

# Task 12: Lets get R to randomly generate set of 50 values from a normal distribution with a mean of 3

#Now use that function to generate the dataset


# Task 13: Note that R has produced the dataset for you, but R has not saved it for you. This is why ob

# Task 14: Let's use a histogram to plot it. Not sure how to make a histogram in R? Try googling it in

# Task 15: Lets get R to randomly generate set of 500 values from a normal distribution with a mean of


#Part 3: installing and using packages
# So far we have been using functions that are part of the R "base package", i.e., these functions like

#Let's use a package to do a data transformation on our Normal.Dist.2 data. Recall that our Normal.Dist

# Task 16: Your goal is to find out what R package will allow you to do a logit transformation on your

# Task 17: Now perform a logit transformation on your Normal.Dist.2 data object.

#Now save the transformed data as a new object, called "logit.Normal.Dist.2"

#Task 18: Make a histogram of these data

#Part 3: Introduction to Data Analysis
# Lastly, let's use R to analyze a hypothetical dataset. Let's say want to collect trout data from a co
#Use rnorm to make up some fake fish weights.
FishWeight.Cont.data<-rnorm(20,mean=11,sd=1)
FishWeight.Clean.data<-rnorm(20,mean=11,sd=1)

####Now we'll use R to make a dataframe, so that we can analyze the data

#tell R to create a vector where the word "Cont" is 'repeated' (rep) the same number of times (length)
Lake.Cont<-rep("Cont", times=length(FishWeight.Cont.data))

#tell R to create a vector where the word "Clean" is 'repeated' (rep) the same number of times (length)
Lake.Clean<-rep("Clean", times=length(FishWeight.Clean.data))
#cbind is "column bind" so now we're telling R to created an object where FishWeight and LakeCont are s
dat.Cont<-cbind(FishWeight.Cont.data, Lake.Cont)

#have a look at what you did
dat.Cont
```

```
##       FishWeight.Cont.data Lake.Cont
##  [1,] "12.3245993028556"   "Cont"
##  [2,] "12.1249994455228"   "Cont"
```

```
##  [3,] "11.2183940690797"   "Cont"
##  [4,] "10.4260045442873"   "Cont"
##  [5,] "13.1694000952803"   "Cont"
##  [6,] "10.5904068525877"   "Cont"
##  [7,] "11.9864498561997"   "Cont"
##  [8,] "11.0968768876232"   "Cont"
##  [9,] "9.9179364218123"    "Cont"
## [10,] "11.8454959466158"   "Cont"
## [11,] "10.6112226183487"   "Cont"
## [12,] "10.8315989899853"   "Cont"
## [13,] "12.1469380935755"   "Cont"
## [14,] "9.05889592585452"   "Cont"
## [15,] "11.94361362431"     "Cont"
## [16,] "9.69380623094638"   "Cont"
## [17,] "10.3480547360616"   "Cont"
## [18,] "10.8236383003829"   "Cont"
## [19,] "11.7960636626196"   "Cont"
## [20,] "11.4976703750809"   "Cont"
```

```
#now do the same for the data for the clean lake
dat.Clean<-cbind(FishWeight.Clean.data, Lake.Clean)


#now use the function rbind (row bind) to place dat.Cont and dat.Clean in the same rows, one atop the o
dat<-rbind(dat.Cont,dat.Clean)

#To make sure R analyzes it properly, tell R that what you created is a dataframe
dat<-as.data.frame(dat)

#Tell R that the fist column [1] in the data frame is called "FishWeight", and the second column [2] is
colnames(dat)[1] <- "FishWeight"
colnames(dat)[2] <- "Lake"

#Tell R that "lake" is a factor (i.e., a grouping variable, not a continuous variable).
dat$Lake<-as.factor(dat$Lake)

# Tell R that "FishWeight" is a continuous variable (i.e., not a grouping variable). To do this, you mu
dat$FishWeight<-as.character(dat$FishWeight)
dat$FishWeight<-as.numeric(dat$FishWeight)


#now voila, have a look at your dataframe!
dat
```

```
##    FishWeight  Lake
## 1   12.324599  Cont
## 2   12.124999  Cont
## 3   11.218394  Cont
## 4   10.426005  Cont
## 5   13.169400  Cont
## 6   10.590407  Cont
## 7   11.986450  Cont
## 8   11.096877  Cont
## 9    9.917936  Cont
```

```
## 10  11.845496  Cont
## 11  10.611223  Cont
## 12  10.831599  Cont
## 13  12.146938  Cont
## 14   9.058896  Cont
## 15  11.943614  Cont
## 16   9.693806  Cont
## 17  10.348055  Cont
## 18  10.823638  Cont
## 19  11.796064  Cont
## 20  11.497670  Cont
## 21  11.382464 Clean
## 22  11.436586 Clean
## 23  11.179496 Clean
## 24  11.714266 Clean
## 25   9.996585 Clean
## 26   9.170040 Clean
## 27  11.758003 Clean
## 28   9.830427 Clean
## 29   9.876813 Clean
## 30  10.380949 Clean
## 31  10.654511 Clean
## 32   9.857094 Clean
## 33  12.141620 Clean
## 34  12.046688 Clean
## 35  10.665015 Clean
## 36   9.924422 Clean
## 37   9.613073 Clean
## 38  10.255371 Clean
## 39  11.084448 Clean
## 40  12.105628 Clean
```

```
#Now lets run the anova
my.anova<-aov(FishWeight~Lake, data=dat)
summary(my.anova)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## Lake         1   1.76  1.7550   1.845  0.182
## Residuals   38  36.15  0.9513
```

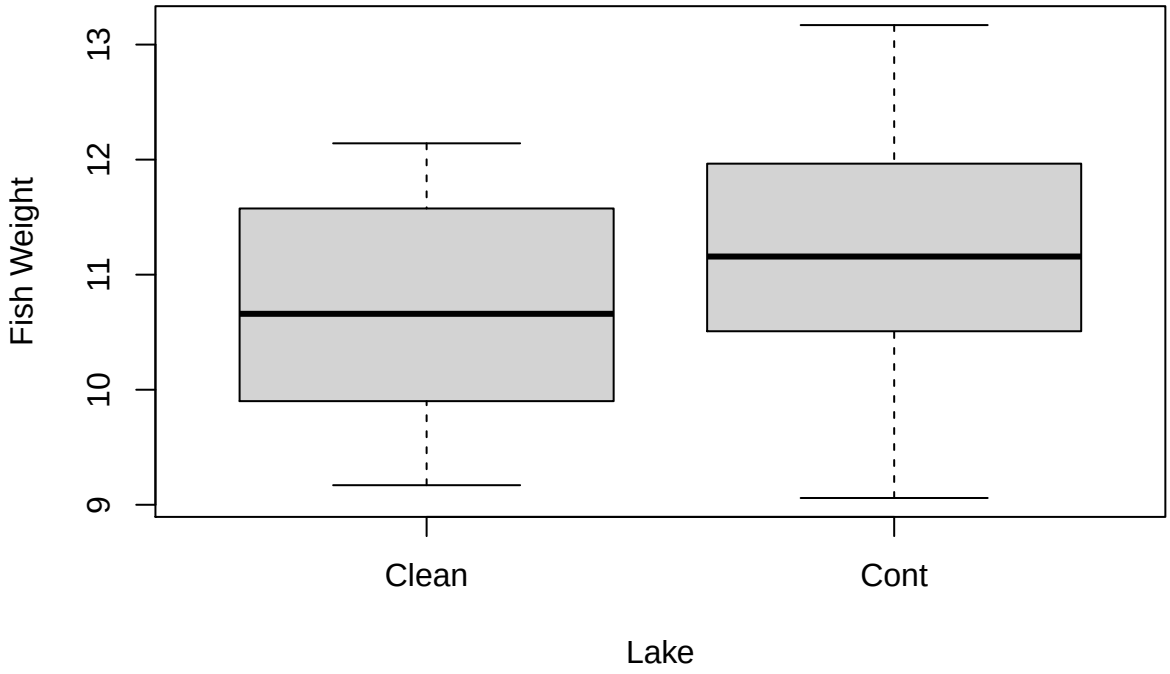
```
#Let's look at the data means for each type of lake

data.Cont<-(dat[which(dat$Lake=="Cont"), ])
mean(data.Cont$FishWeight)
```

```
## [1] 11.1726
```

```
data.Clean<-(dat[ which(dat$Lake=="Clean"), ])
mean(data.Clean$FishWeight)
```

```
## [1] 10.75368
```

```r
#We can even make a quick boxplot it if we like
plot(dat$FishWeight~dat$Lake, xlab="Lake", ylab="Fish Weight")
```




```r
#Let's back up a minute, and see how we can do things more efficiently by using a function

FishWeights<-c(FishWeight.Cont.data, FishWeight.Clean.data)
LakeID<-c(Lake.Cont, Lake.Clean)

#Let's use the dataframe function, allowing us to skip a few steps
fish.data <- data.frame(FishWeights, LakeID, stringsAsFactors=FALSE)

#now we can run the anova again, with the same result.
my.anova<-aov(FishWeights~LakeID, data=fish.data)
summary(my.anova)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## LakeID       1   1.76  1.7550   1.845  0.182
## Residuals   38  36.15  0.9513
```

```r
#Let's look at the data means for each type of lake

data.Cont<-(fish.data [which(fish.data $LakeID=="Cont"), ])
mean(data.Cont$FishWeights)
```

```
## [1] 11.1726
```

```r
data.Clean<-(fish.data [ which(fish.data $LakeID=="Clean"), ])
mean(data.Clean$FishWeights)
```

```
## [1] 10.75368
```

```r
#We can even make a quick boxplot it if we like
boxplot(FishWeights~LakeID, xlab="Lake", ylab="Fish Weight", data=fish.data)
```