Statistics_Day1

Your Name

9/7/2021

Here we will cover some formatting instructions. The results of the formatting will only be visible once we knit the file (convert it to pdf/html/word) but we must use the proper coding in R to produce that result.
Please refer to the pdf version to see the end result of each command

The command below allows us to create a header. The more hashtags we add before the words “Header x”, the smaller the header will be:

Header 1

Header 2

Header 3

To produce plain text, all you need to do is type it in the white blank space
To ensure the line breaks after you end your sentence, make sure to end your lines with two spaces

Below is an example of what happens if you do not add in two spaces (see pdf):

if you do not the line will not break

Add in one asterisk * on each side of the text you would like italicized and two ** on each side of the text you would like bolded:

italics and **bold**

If you would like to create a horizontal line through your document put *** alone in a line and it will appear when you knit your file! Below is an example (see pdf):

By placing an asterisk on only the left-hand side of text, you will create a bulleted/unordered list. Make sure to put a space after the asterisk to create a bullet. If you would like to create an indented item, you must hit tab once and put a + symbol to the left of the text:

- unordered list
- asterisks
    - indented item

If you wish to create a numbered/ordered list you must type in the number of the item with a . symbol after it. If you wish to create an indented item, you will hit tab once and put a + symbol to the left of the text as with the previous example:

1. ordered list
2. item 2

- indent

When we want to make an equation in R we can put a dollar sign before and after the equation (see pdf for result). Using 1 dollar sign allows the equation to appear inline with the code/text when we transform the R code to pdf/word/html version.

inline latex $E = mc^2$

If we want to make a stand alone equation which is not in line with the text we must use 2 dollar signs before and after the equation. The use of a backslash allows R to use symbols that are saved in R via the latex system (built in symbols)

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Some things to keep in mind:

- **The environment tab** -> shows us the variables we have created and the values we have set them to
- **The console tab** -> allows us to run console style commands, we will not be using this for long style commands like we are coding below
  -> Also allows us to better understand commands by typing a question mark and then afterward the name of the function (ie; ?mean - if we were looking to understand how to use the mean function)

Time to begin by coding in the terminal, just a cute “hello world”. Let’s make a chunk:

Chunks represent the sections in R where we type our commands. We can create a chunk by typing three backticks followed by {r} at the beginning of our chunk and three backticks at the end of our chunk (see below) OR by clicking insert, then clicking R

Below we are creating the “print” command which tells R to output the text we put inside it. We provide the text by putting quotation marks around what we seek to be spit back out to us

Run your entire code chunk by pressing the small green triangle next to it. If you wish to run only one line put your cursor next to it and press command/control + enter. If you wish to run the entire chunk instead press command/control + shift + enter

```
print("hello world!")
```

```
## [1] "hello world!"
```

```
print("this is my second piece of R code")
```

```
## [1] "this is my second piece of R code"
```

Let us examine how we would assign a variable in R. You can type in the code name then use either <- or = in order to assign the value to the variable

Keep in mind that whatever variable you assign in that code chunk will persist in that specific chunk and only after that chunk. This also means that if you assign a different value to a variable, the new value will persist over the old one

See some examples of assigning variables below:

```
var1 <- "hello world"
var2 = "hello world"
```

Something to keep in mind:

- We can assign a variable value to any letter of the variable except for c . This letter has a command associated with it in R so we cannot use it for variable naming

The command below allows us to delete everything in our environment:

If we wish to only delete one variable from our environment we would type the variable name in the parentheses instead

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

Data type

- vectors
- lists
- matrices
- arrays
- factors *dataframes

Vectors

Let us look at an example of assigning a vector below:

```
apple<-c("red","green","yellow")
print(apple)
```

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

Some things to keep in mind:

- The difference between us having a vector and a list is that the values in a vector must all be of the same type
- When indexing the values in a vector/list R will always start numbering them from 1 instead of 0
- When we name a vector we should not use the letter c as the name as it is a function in R that is used to define a vector

** Accessing elements in a vector

We will make a list of the days of the week to use as an example here. We do so by naming our vector then typing <-c() with the vecotrs we want to use inside the parentheses. Make sure to separate your values by a comma.

```
t<-c("Sun","Mon","Tues","Wed","Thurs","Fri","Sat")
```

Now that we have created a vector to use, let us select individual values in the vector. We do so by typing in the vector name, then a set of brackets [ ] and inside of the brackets indicate the number of the value we choose to pull out (Remember that R starts numbering from 1)

Below we have two examples, the first line showing us how to select one value from the vector and the second line showing us how to select several continuous values in the vector. To select multiple values we must put the starting numbering, then a colon and the ending value inside of our brackets. R will include the starting and ending values that you mention

```
t<-c("Sun","Mon","Tues","Wed","Thurs","Fri","Sat")
t[3]
```

```
## [1] "Tues"
```

```
t[3:5]
```

```
## [1] "Tues"  "Wed"   "Thurs"
```

If we wish to select discontinuous values we cannot follow the same protocol as above.

Instead of just putting the number of the values we want inside of the brackets and a colon we must type c() inside of the brackets. The parentheses should include the number of the values we seek to pull out of the vector separated by a comma

```
t<-c("Sun","Mon","Tues","Wed","Thurs","Fri","Sat")
t[c(1,6,7)]
```

```
## [1] "Sun" "Fri" "Sat"
```

We can also do something called logical indexing/masking an array. This means that we indicate which values from our vector we would like to pull out by using true and false statements. Where we type in true, the value will show up, where we type in false, the value will not.

```
t<-c("Sun","Mon","Tues","Wed","Thurs","Fri","Sat")
tf<-c(TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE)
t[tf]
```

```
## [1] "Sun"   "Tues"  "Thurs" "Sat"
```

Let us do a basic math operation using vectors:

```
v1<-c(3,8,4,5,0,11)
v2<-c(4,11,0,8,1,2)
add<-v1+v2
print(add)
```

```
## [1]  7 19  4 13  1 13
```

```
print(v1-v2)
```

```
## [1] -1 -3  4 -3 -1  9
```

```
print(v1*v2) #elementwise - R will do math with each individual value/element of the vec
tors
```

```
## [1] 12 88  0 40  0 22
```

```
print(v1/v2)
```

```
## [1] 0.7500000 0.7272727       Inf 0.6250000 0.0000000 5.5000000
```

The method we used above to do basic math operations should also work for strings!

Let us try doing arithmetic with vectors of a different length and see what happens:

```
v3 <- c(4,11)
print(v1+v3)
```

```
## [1]  7 19  8 16  4 22
```

When we run the code above we wil see that we can even add vectors of different lengths!
This is possible because R does something called *vector recycling*, meaning that it will continue adding the shorter vector from the start. In this example this would mean that from v 3 it adds 4, then 11, then 4 again, then 11 again and so on….

Note: The values of the vectors need not be multiples of each other for us to run commands with them.

Lists

Lists are very similar to vectors but recall that they do not require us to use elements of the same type (ie;all words or all numbers)

Let us try to create a list with different element types and see what happens:

```
list4<-list(21,3,c(2,4,3),sin)
print(list4)
```

```
## [[1]]
## [1] 21
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] 2 4 3
##
## [[4]]
## function (x)  .Primitive("sin")
```

Ta da! It worked. The list we created above has 3 different types of elements. Element 1 is a number (21,3), Element 2 is a vector (c(2,4,3)) and Element 3 is a function object (sin)

Let us try to do another example of a list with different element types below:

```
list1<-list(c("Jan","Feb","Mar"),matrix(c(3,9,5,1,-2,8),nrow=2),list("green",12.3),TRUE)
names(list1)<-c("1st Quarter","A_Matrix","Inner List","Boolean")
print(list1)
```

```
## $`1st Quarter`
## [1] "Jan" "Feb" "Mar"
##
## $A_Matrix
##      [,1] [,2] [,3]
## [1,]    3    5   -2
## [2,]    9    1    8
##
## $`Inner List`
## $`Inner List`[[1]]
## [1] "green"
##
## $`Inner List`[[2]]
## [1] 12.3
##
##
## $Boolean
## [1] TRUE
```

Element 1[named 1st Quarter] in that list is a vector, Element 2[named A_Matrix] in that list is a matrix (nrow=2 just means how many rows we would like), Element 3[named Inner List] is two lists (inception in a way) and Element 4[named Boolean] is a boolean character!

Note that the command names(listname) <- c(name1, name2, name3,…) allows us to give names to each element in our lists

This may be a lot of elements to wrap your head around now, do not worry you do not need to memorize them yet, we just want to grasp that lists have a lot of flexibilities with element types

Combining vectors and lists

Let us try combining multiple vectors into one large vector:

Below we will be making vector 1 and vector 2 and putting them together into one big vector called vector 3

```
v1<-1:3
v2<-4:6
v3<-c(v1,v2)
print(v3)
```

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

We can also combine lists just like we did with vectors:

```
list1<-list(1,2,3)
list2<-list("Mon","Tues","Wed")
c(list1,list2)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] "Mon"
##
## [[5]]
## [1] "Tues"
##
## [[6]]
## [1] "Wed"
```

Matrices,abridged

Let us try to create a matrix of values:

```
M<-matrix(c(3:14),nrow=4,byrow=FALSE)
print(M)
```

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

```
M2<-matrix(c(3:14),nrow=4,byrow=TRUE)
print(M2)
```

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

The command byrow=FALSE allows our matrix to appear in columns rather than going down by row. Take a look at the difference in the order of values between the two matrixes above to get a better understanding

If we wish to name our rows and columns we can use the functions rownames and colnames respectively. We will not be exploring that funciton here but keep it in mind for future usage

If we wanted to select specific parts of a matrix we can do so

```
M[1,1] #first element
```

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

```
M[,1] #first column
```

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

```
M[1,] #first row
```

```
## [1]  3  7 11
```

Each of the commands above allows us to access their respectively labeled part of a matrix

Note: The idea of vector recycling also applies to matrixes

Fun Fact: The shortcut to matrix multiplication is: %*% .We will also not be covering this topic at the moment but it useful to keep in mind for future use

Something to note is that if we wanted to make a matrix with anything other than 2 dimensions we cannot use the matrix function, we must use the array function instead - we will come back to this in the future

dataframes

Data frames are a cool function in R as it essentially allows us to create an excel spreadsheet in R. What we do in data frames is give R our desired row and column names as well as what values we would like to be put in them and it creates it for us

Note: * Rows do not have to be the same data type but columns do * Instead of using the arrow in these functions when assigning values we use an = sign

```
data<-data.frame(
  id=1:5,
  name=c("Rick","Dan","Michelle","Ryan","Gary"),
  salary=c(623.3,515.2,611.0,729.0,843.25)
)
print(data)
```

```
##   id     name salary
## 1  1     Rick 623.30
## 2  2      Dan 515.20
## 3  3 Michelle 611.00
## 4  4     Ryan 729.00
## 5  5     Gary 843.25
```

If we only wanted to see a few lines of our data frame, we can do so by using the function below:

```
head(data,n=2)
```

	id <int>	name <chr>	salary <dbl>
1	1	Rick	623.3
2	2	Dan	515.2

2 rows

By adding n=2 we are specifying that we want to see the top(head) 2 lines

We can also ask to see a summary of our data frame:

```
summary(data)
```

```
##        id        name               salary     
##  Min.   :1   Length:5           Min.   :515.2  
##  1st Qu.:2   Class :character   1st Qu.:611.0  
##  Median :3   Mode  :character   Median :623.3  
##  Mean   :3                      Mean   :664.4  
##  3rd Qu.:4                      3rd Qu.:729.0  
##  Max.   :5                      Max.   :843.2  
```

This reports back to us the quartiles of our data frame (if applicable for a numerical dataset)

We are also able to look at only the values of one vector of our data frame. Let us try looking at only the salary vector from our data set above:

```
data$salary
```

```
## [1] 623.30 515.20 611.00 729.00 843.25
```

We did this by writing the name of the data set, followed by a $ sign and followed by the vector type we wish to see

This function is also useful if we wish to add in a column to our dataframe. Let us try adding in a vector called dept:

```
data$dept<-c("IT","Operations","IT","HR","Finance")
print(data)
```

```
##   id     name salary       dept
## 1  1     Rick 623.30         IT
## 2  2      Dan 515.20 Operations
## 3  3 Michelle 611.00         IT
## 4  4     Ryan 729.00         HR
## 5  5     Gary 843.25    Finance
```

Control flow

We will briefly be going over a few different types of functions below:

##If else statements

```
set.seed(420)
x<-sample(-10:10,1)
print(x)
```

```
## [1] -6
```

```
if(x<0){
  print("negative")
} else if (x>0){
  print("Positive")
}else {
  print("Zero")
}
```

```
## [1] "negative"
```

Let us go over what each function used above means: * The set.seed(420) function just made sure that our random number list started from the same spot, this is not something you need to know just yet. * The x<-sample(-10:10,1) allowed us to generate a random number from the list of values in sample (-10:10) and we indicated that we only wish to obtain 1 random value

The if else function above told R that if our random number is a negative number, we want it to tell us “negative”, if our number is positive we want it to tell us “positive” and if neither we want it to tell us “zero”

Note: For any python users this is very similar to python except we do not necessarily need to indent and we use brackets instead of colons

##For loop

```
for(x in 1:10){
  print(x)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Above we asked R to print all the numbers between 1 and 10 for us

##While statements

```
i<-1
while(i<6){
  print(i)
  i<-i+1
}
```

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

Above we asked R to start with a value of i which was 1 and then put it in the function i<-i+1 , meaning that it adds 1 to our value of i each time the loop runs. We then asked for it to only print out our i values if they are less than 6 (i<6)

functions

Let us examine a built in function called mean:

```
x<-c(12,7,3,4,18,2,54,-21,8,-5)
var1<-mean(x)
print(var1)
```

```
## [1] 8.2
```

It is just as its name sounds, it allows us to obtain the mean of a list of values

Let us now try creating our own function:

```
# def quad_formula(a,b,c) is the equivalent in python

quadratic_formula<-function(a,b,c){
  result1<-(-b+sqrt(b^2-4*a*c))/(2*a)
  result2<-(-b-sqrt(b^2-4*a*c))/(2*a)
  return(c(result1,result2))
}

#now that we have created a function let us try running it with some values (each corres
ponding to a,b and c respectively)
quadratic_formula(1,-3,2)
```

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

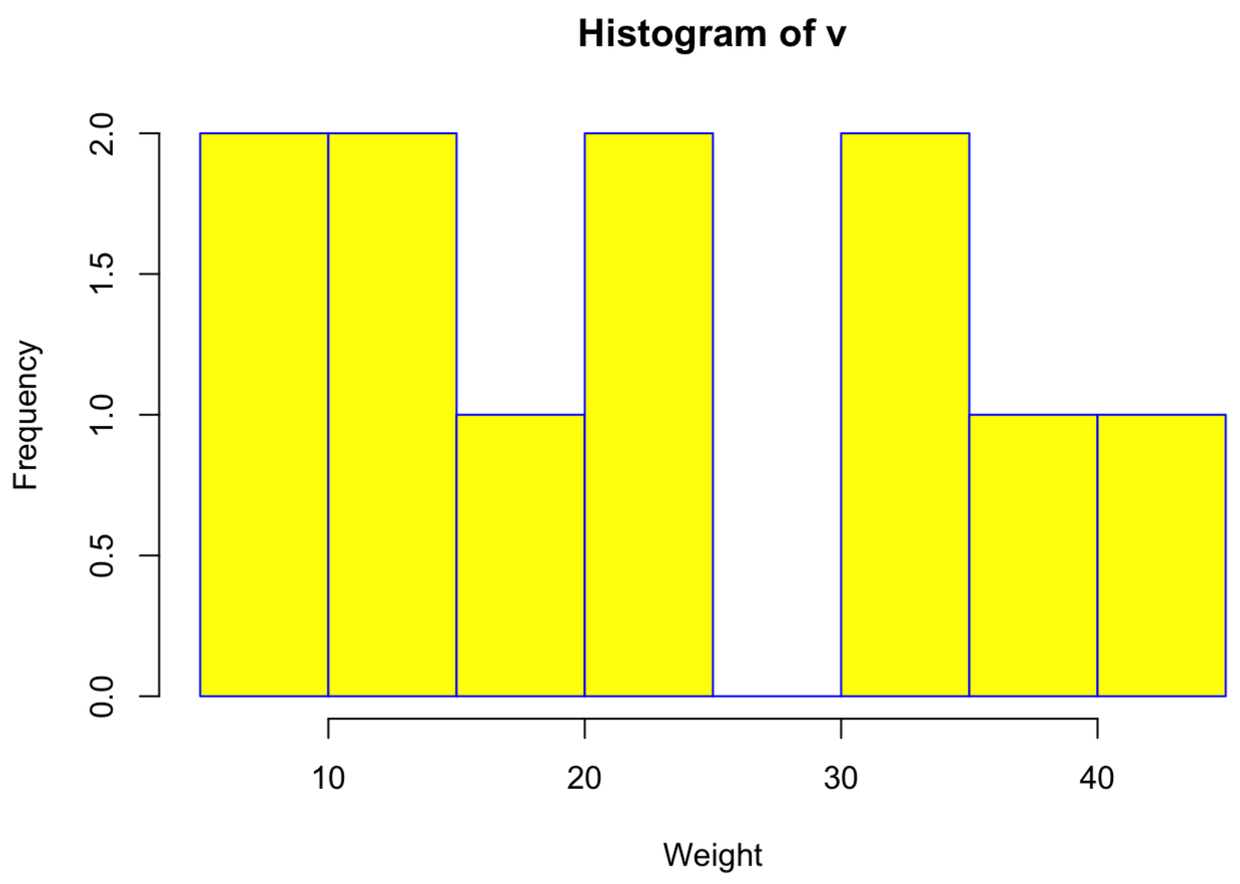
```
#We can also just tell R which variable corresponds to which function instead of having
 it assume it is in the order we provided in the formula
quadratic_formula(b=5,a=2,c=-3)
```

```
## [1]  0.5 -3.0
```

Histogram

Let us try creating a histogram with a set of values:

```
v<-c(9,13,21,8,36,22,12,41,31,33,19)
hist(v,xlab="Weight",col="yellow",border="blue",breaks=10)
```

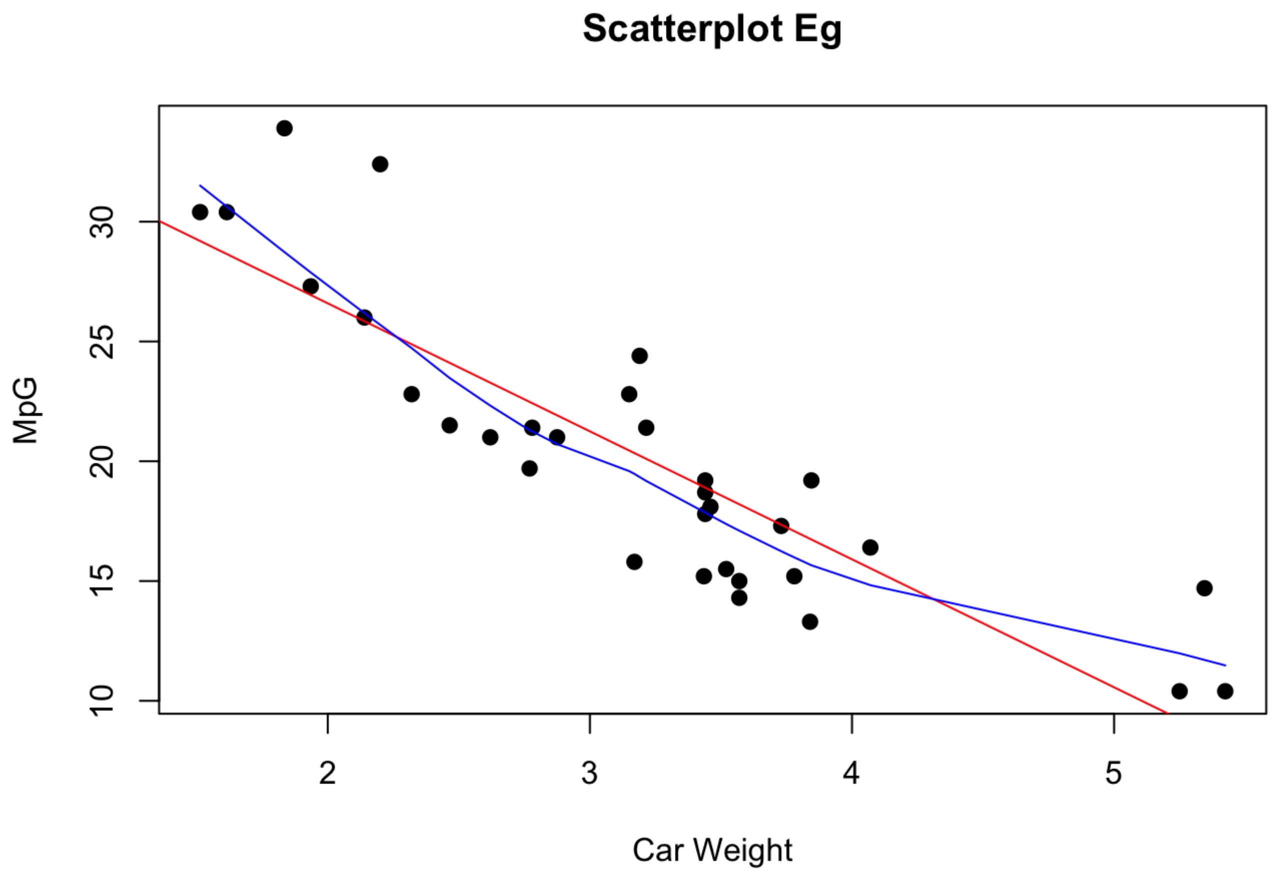


As you will likely notice there are a lot of things added in the function hist() above. Let us go through each one: * xlab= “name” allows us to assign a name to our x axis * col= “Yellow” allows us to assign a color to our bars * border=“blue” allows us to assign a color to the outline of our bars * breaks= # allows us to determine how thick we want each bar to be. The greater the number, the thinner the bars will be

Let us now create a histogram with a set of values from a premade dataset:

```
library(MASS)
attach(mtcars)
plot(wt,mpg,main="Scatterplot Eg",xlab="Car Weight",ylab="MpG",pch=19)

abline(lm(mpg~wt),col="red")
lines(lowess(wt,mpg),col="blue")
```



The function library(name) allows us to download a library of values from R. The function attach(name) allows us to call a specific variable name, we will look at this again tomorrow.

Xlab and ylab allow us to name our x and y axis, respectively.

The values we wrote in plot are the variable names in the library MASS. our pch value of 19 is meant to represent that we want our graph to have big dark circles, each number represents a different shape. Look up the pch value table to determine which shape fits your plot.

Abline() allows us to create a line of best fit. We will discuss the lm function tomorrow! Abline creates a perfectly straight line while lines gives us a line graph on top. Lowess is a smoother of the line that is graphed.

***Fun Formatting Fact: When we knit a file, R will split up code chunks at every “print” command statement. If we do not want our code to be split up like this when it is knit we can put all our print commands at the end of our code chunk and it will keep it together. Alternatively we can type in ```{r,echo=FALSE} when starting off our chunk and it will not be separated