ES207- Lab1

Christiana Ade

January 28, 2018

First R session

1. What happens if you try to print out the 4th element of x? Print your result nad provie the answer in your lab write up.

- An NA is returned because there is no 4th element of the vector x.

```
x <- c(1,2,4)
x[4]
```

```
## [1] NA
```

2. Try creating a variable “s” that is the standard devitation of q. Make sure you print it out to confirm it worked.

```
q <- c(x,x,8)
s <- sd(q)
s
```

```
## [1] 2.478479
```

```
#s is 2.478479
```

Some Basic Operations in R

3. Can you make R write your name

```
paste("Christiana","Ade")
```

```
## [1] "Christiana Ade"
```

```
# if I wanted to save my name as a variable
myName <- paste("Christiana","Ade")
myName
```

```
## [1] "Christiana Ade"
```

List All Objects in the Environment

4. What objects are left in you R session after removing m?

- This depends on if you are asking us what remains in our environment after we run the code you asked us to write for homework too, or it is just the variables you have assigned in the lab.
- If it is just the assigned variables in the homework, then we would be left with “a” “b” “c” “d” “q” “x” “y”
- If it is including the variables you asked us to assign then tge variables left are below:
- I most likely have one extra variable than the other students because I assigned myName to variable

```
m <- 100
ls()
```

```
## [1] "m"      "myName" "q"      "s"      "x"
```

```
rm(m)
ls()
```

```
## [1] "myName" "q"      "s"      "x"
```

Introduction to Functions

5. How many odd numbers were in this vector? (line 59)

- There were 4 even numbers in this character vector.

```
oddcount <- function(x)
{
    k <- 0 # assign 0 to k
    for (n in x) {
        if(n %% 2 == 1)
        {
            k <- k+1 # %% is the modulo operator
        }
    }
  return(k)
}
oddcount(x <- c(1,2,3,7,9))
```

```
## [1] 4
```

6. Try creating a new function “evencount” that counts the even numbers in a vector. Turn in your script as a .R script with appropriate comments.

```
evencount <- function(x){
  k <- 0  # assign 0 to k
  for (n in x){
    if (n %% 2 == 0){ ## divide the given number by 2 if the remainder is 0 then it i
s an even number
      k <- k+1 # add this number to count of even numbers
    }
  }
  return(paste("There are",k,"even numbers!")) # return the amount even numbers
}
evencount(x <- c(1,2,6,10,11))
```

```
## [1] "There are 3 even numbers!"
```

Making good directory layouts

7. What are the three main reasions you want to have a good project layout? Can you think of any others?
    i. Integrity of data
    ii. Portability of the project
    iii. Easier to pick the project back up after a break
    - answers from :https://nicercode.github.io/blog/2013-04-05-projects/ (https://nicercode.github.io/blog/2013-04-05-projects/)
    - All of these reasons help with transparency and reproducability issues.
8. What are the three primary principles to follow in a good project layout.
    i. Treat data as read only.
    - This ensures that you do not accidenally edit your data.
    ii. Treat generated output as a disposable
    - Any edits to your data, and output generated including figures etc, should be able to be deleted at any moment during the project.
    iii. Separate function definition and application
        - When completing your own project it is likely that you will write your own functions that you use repeatedly.Hhousing them in a separate place and then calling them at the beginning of your script (like packages) will make your scripts cleaner and shorter.

Full, base, and relative paths

9. Write out the full path for your R installation. Use the format of the operating system you are currently using.

```
R.home()
```

```
## [1] "C:/PROGRA~1/R/R-33~1.0"
```

```
# My path installation is "C:\Program Files\R\R-3.3.0"
```

10. Write the path above using a different operating system

```
#Not really sure where the program files of macs go, but it would be something like th
is
#/Program Files/R/R-3.3.0
```

11. Write out the full path for the directory structure you have set up for this class all the way to where you have saved this .html tutorial

```
#"C:\Users\cade\Documents\PhDMerced\Spr18Courses\EnvironmentalDataAnalysis\Homework\La
b1\Lab1_tutorial.html"
```

12. Write out the relative path for this .html file assuming your working directory to be set to your equivalent of Users/DavidBowie/classes/.

```
#"C:\Users\cade\Documents\PhDMerced\Spr18Courses"
```

13.Write out the paths in an operating system other than your own.

```
  '/In Q11:
  /Users/cade/Documents/PhDMerced/Spr18Courses/EnvironmentalDataAnalysis/Homework/Lab
1/Lab1_tutorial.html
  - In Q12:
  /Users/cade/Documents/PhDMerced/Spr18Courses
/'
```

```
## [1] "/In Q11: \n  /Users/cade/Documents/PhDMerced/Spr18Courses/EnvironmentalDataAna
lysis/Homework/Lab1/Lab1_tutorial.html\n  - In Q12: \n  /Users/cade/Documents/PhDMerce
d/Spr18Courses\n/"
```

Obtaining Help in R

14. Using the help documentation, provide an alternative usage for the function ncol()

- I am not sure exactly what you mean by an alternative usage. The help recommends using nrow () as another function, but I am assuming you are refering to using dim() as a way to figure out what the dimenisions of a matrix are, which would also allow you to see the number of columns in your dad.

Generating Vectors

15. Create and print a new vector, w that is a subset of the first 4 elements of vector v, where v <- c (8:19)

```
v <- c(8:19)
w <- c(v[1:4])
w
```

```
## [1]  8  9 10 11
```

16. Create and print a new vector, h that is a subset of the middle 5 elements of vector d, where d <- c(8:20)

```
d <- c(8:20)
#which(d %in% median(d)) #check where the middle is...only works vectors with an odd l
ength
h <- c(d[5:9])
h
```

```
## [1] 12 13 14 15 16
```

Obtaining the length of a vector

17. What is the length of vector d from above?

- The length is 13

```
length(d)
```

```
## [1] 13
```

Types, Testing & Coercion of Vectors

18. What are the 3 properties of a vector other than its contents
    i. Type. What kind of vector is it. Can be tested using `typeof()`
    ii. Length. How many elements it contains. Tested using `length()`
    iii. Attributes. Any addition metadata. `Attributes()`

Coercing Vectors

19. Can you predict the output of the following uses of c()? c(3,FALSE) c("a",1) c(4L,2) My predictions:
    i. "double" - one is double and other is logical. Doesn't make sense that it would go to logical.
    ii. "character" - similar to example
    iii. "double" - one is integer and other is double

```
#Let's see what really happens
# find typeof the three situtations and save to a variable
a <- typeof(c(3, FALSE))
b <- typeof(c("a",1))
c <- typeof(c(4L,2))

myPrediction = c("double","character","double")
realAnswers = c(a,b,c)
# test if the two are identical if true then print "your predictions were right"
if (identical(myPrediction,realAnswers) == TRUE){
  print("Your predictions were right")
} else{
  print("nooooooo")
}
```

```
## [1] "Your predictions were right"
```

20. What are the 4 types of common atomic vectors? What are the 2 rare types? *Main*
    i. Character
    ii. Numeric
    iii. Integer
    iv. Logical

    Rare
    i. complex
    ii. raw
21. Why is 1==“1” true? Why is -1 < FALSE true? Why is “one< 2 false?

a. 1==“1”.

- if we look at the documentation of == by typing in ?“==”
- we see that: If the two arguments are atomic vectors of different types, one is coerced to the type of the other, the (decreasing) order of precedence being character, complex, numeric, integer, logical and raw.
- they are both atomic. Therefore, R is coercing one side of the == to the other type of atomic vector. I would assume that it is coercing 1 to become “1” since if we do type(n) it returns character and that character is the first in the list of precedence above
- however if we test they are identical without using == then FALSE is returned.

```
n <- "1"
m <- 1
is.atomic(m)
```

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

```
is.atomic(n)
```

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

```
identical(1,"1")
```

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

b. Similarly to before one type must be coerced to the other and here integer is more flexible than logical (according to list above). Based on the notation I would assume that FALSE gets coerced into int 0. In which -1 is less than 0.

c. assuming you meant “one” < 2 I would not have ever expected this to be true because “one”" is a character and the other is a number.Therefore, R will coerce 2 into “2” and this statement will be FALSE.

22. Why is the default missing value NA, a logical vector? What’s so special about logical vectors? Hint: think about c(FALSE,NA_character)

- Besides raw, logical vectors are the highest type on the hierarchy of coercion. This means that they will be coerced before the other datatypes (such as doubleand character). If we look down below we see that vector b will result in type double because NA (logical) is coerced to an integer. If NA was of class character and then paired with numbers (such as in vector c1) the type will be coerced to character. Since NA’s occur often in environmental data collection and are likely to be included in a vector with numerical data, everything in that vector will get treated as type character. This is generally bad, since we most likely will want to do some type of computing on numbers, not characters.

```
y <- c(FALSE,NA)
typeof(y)
```

```
## [1] "logical"
```

```
x <- c(FALSE, NA_character_)
typeof(x)
```

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

```
b <- c(1,NA)
typeof(b)
```

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

```
c1 <- c(1, NA_character_)
typeof(c1)
```

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

Indexing Vectors

23. Provide the value of the last element in the result of the vector operation d * 2 from the vectors defined above

```
#assuming this is vector d
d <- c(8:20)
newD <- d * 2
newD[length(d)]
```

```
## [1] 40
```

```
# 40 is the last element
```

24. Create a vector a with a sequence of 11 elements in it ranging from 0.6 to 1.9. Create another, logical vector b that identifies which elements in a are greater than or equal to 1.1. Print your results.

```
a <- seq(0.6,1.9,by = 0.1)
b <- which(a >= 1.1)
b
```

```
## [1]  6  7  8  9 10 11 12 13 14
```

25. Set all elements in vector a from above to 2.0 if they are greater than or equal to 1.1.

```
a[a >= 1.1] <- 2.0
a
```

```
##  [1] 0.6 0.7 0.8 0.9 1.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0
```

Generating Lists

26. Use str() to compare the results of using list() and c() on l and j for: l <-c(list(1,2),c(3,4)) and j=list (list(1,2),c(3,4))

```
l <- c(list(1,2),c(3,4)) #combines several lists. atomic vectors + list --> coerce to
list before combining
j <- list(list(1,2),c(3,4))
str(l)
```

```
## List of 4
##  $ : num 1
##  $ : num 2
##  $ : num 3
##  $ : num 4
```

```
# returns a list of 4 numbers
str(j) # a list of two numbers and then vector numbers 3 and 4
```

```
## List of 2
##  $ :List of 2
##   ..$ : num 1
##   ..$ : num 2
##  $ : num [1:2] 3 4
```

Types, Testing & Coercion of Lists

27. Show what happens when you use unlist() on l and j

```
unlist(l)
```

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

```
unlist(j)
```

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

```
# the same thing happens to both
```

Factors

28. If factors are essentially integers under the hood, why do we use them, and not just use integers (e.g., 1s and 2s instead of male, female)?

- They are important for statistical modeling because you can compute statistics using categorical variables, but you need them to be entered in differently than continuous variables.
- Since you can specify levels you can also compare levels.

Generating Factors

29. Using table(), provide the frequency of no and yes in my.fac.

```
my.fac <- factor(c("yes", "yes","no","yes","yes","yes"))
table(my.fac)
```

```
## my.fac
##  no yes
##   1   5
```

Types, Testing and Coercion of Factors

30. What does this R code below do? How are f1 and f2 different from f3?

- f1: letters are first convered to factors. Then the levels of f1 are reversed. Which both reverses the way that the letters are displayed and the levels.

```
f1 <- factor(letters)
levels(f1) <-rev(levels(f1))
f1
```

```
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

- f2 you are only reversing the order of the factors, but not the order of the levels

```
f2 <- rev(factor(letters))
f2
```

```
##  [1] z y x w v u t s r q p o n m l k j i h g f e d c b a
## Levels: a b c d e f g h i j k l m n o p q r s t u v w x y z
```

- f3 you are greating factors where

```
f3 <- factor(letters, levels=rev(letters))
f3
```

```
##  [1] a b c d e f g h i j k l m n o p q r s t u v w x y z
## Levels: z y x w v u t s r q p o n m l k j i h g f e d c b a
```

In f1 and f2 you are changing the letters to factors first and then changing either the levels or the letter order to be reserved, where as in f3 you are telling the function factor to change letters to a factor and then just reverse the levels within the factor command. In f1 both are reversed. In f2 the letters are reversed by not the levels. In f3 the letters only the levels are reversed.

Indexing Matrices

31. If is.matrix(x) is TRUE, what will is.array(x) return?
    - It will also return true because its just a matrix is a two-dimensional array

```
y<-matrix(1:12*3,nrow=4,ncol=3)
is.matrix(y)
```

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

```
is.array(y)
```

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

Generating Data Frames

32. Why didn't that work? Hint: read the errors -This didn't work because we are trying to make a dataframe with two columns that have different low lengths. The first column kids only has 2 entries and the second column ages has three

```
d <- data.frame(kids=c("Jack","Jill"),ages=c(12,10))
d
```

```
##   kids ages
## 1 Jack   12
## 2 Jill   10
```

```
#f <- data.frame(kids=c("Jack","Jill"),ages=c(12,10,11))
```

33. What does as.matrix() do when applied to a data frame with columns of different types?
    - If we print the matrix it appears if it turns everything in the data frame to a character

```
m <- as.matrix(d)
m
```

```
##      kids   ages
## [1,] "Jack" "12"
## [2,] "Jill" "10"
```

```
# to verify that
class(m[1,2])
```

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

34. What if we don't have the same variable name? Change the data frame variables names using names(d2) <- c("ages","kids"). Then, use the help documentation to merge d1 and d2. Print your results.

```
d1 <- data.frame(names=c("Jack","Jill","John"),states=c("CA","IL","IL"))
d2 <- data.frame(ages=c(10,7,12),names=c("Jill","Jillian","Jack"))
names(d2) <- c("ages","kids")
merge(d1,d2, by.x = "names", by.y= "kids")
```

```
##   names states ages
## 1  Jack     CA   12
## 2  Jill     IL   10
```

Types, Testing and Coercion

** is this supposed to be the name of this section?

35. Can you have a data frame with 0 rows? What about 0 columns?

```
#Yes, you can have a data frame with 0 rows
noRows <- data.frame("class" = integer(0))
noRows
```

```
## [1] class
## <0 rows> (or 0-length row.names)
```

```
dim(noRows) # 0 1
```

```
## [1] 0 1
```

```
class(noRows) # data.frame
```

```
## [1] "data.frame"
```

```
#Yes, you can have a dataframe with 0 columns, but it will also have 0 rows
df <-data.frame(NULL)
df
```

```
## data frame with 0 columns and 0 rows
```

```
dim(df) #0 0
```

```
## [1] 0 0
```

```
class(df)
```

```
## [1] "data.frame"
```

36. Plot a histogram of all the Sepal.Width values in the dataset iris that are less than or equal to 3.0

```
# one way
# select all the columns with matrix notation
df <- iris[iris$Sepal.Width >= 3.0, ]
hist(df$Sepal.Width, main ="Frequency of Sepal Width Values in Iris greater or equal t
o 3.0")
# can select with subset
df <- subset(iris, Sepal.Width >= 3.0)
# With tidyverse just to practice
require(tidyverse)
```

```
## Loading required package: tidyverse
```

```
## Warning: package 'tidyverse' was built under R version 3.3.3
```

```
## -- Attaching packages --------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 2.2.1     v purrr   0.2.4
## v tibble  1.4.1     v dplyr   0.7.4
## v tidyr   0.7.2     v stringr 1.2.0
## v readr   1.1.1     v forcats 0.2.0
```

```
## Warning: package 'ggplot2' was built under R version 3.3.3
```

```
## Warning: package 'tibble' was built under R version 3.3.3
```

```
## Warning: package 'tidyr' was built under R version 3.3.3
```

```
## Warning: package 'readr' was built under R version 3.3.3
```

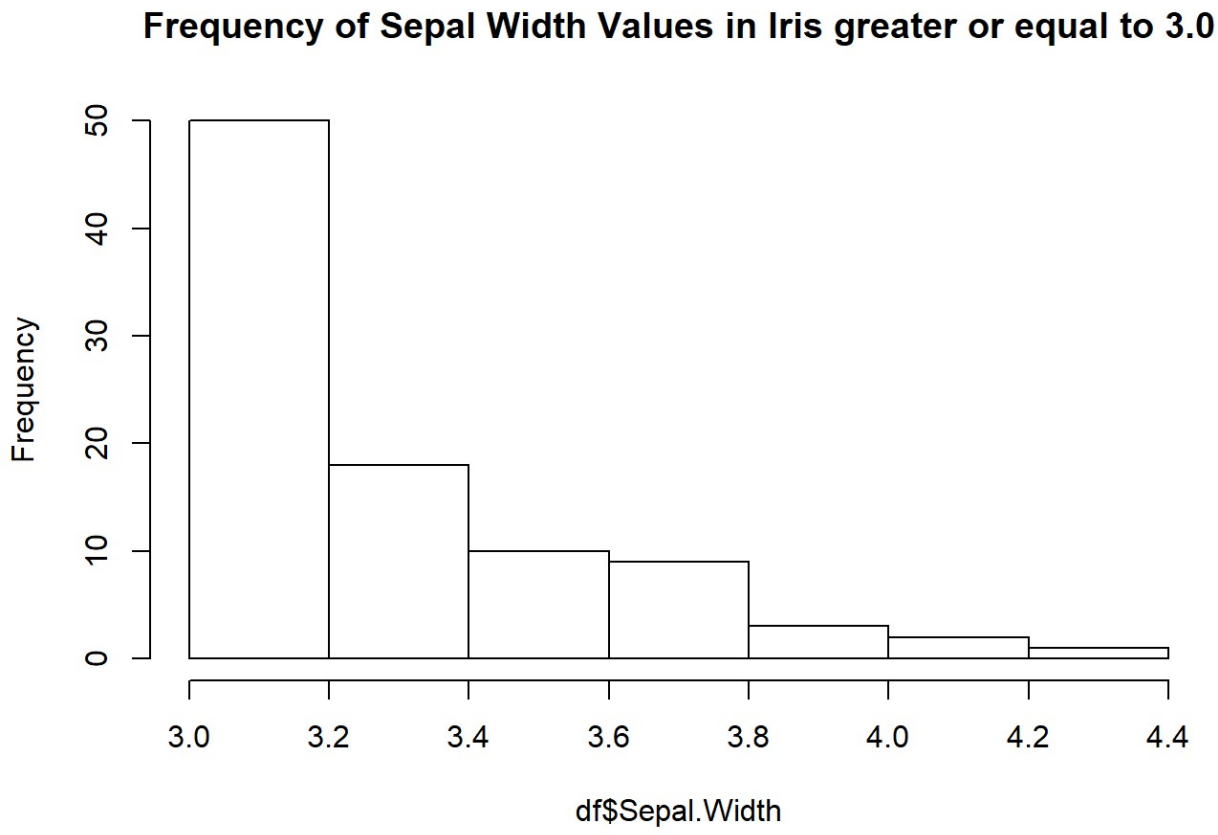
```
## Warning: package 'purrr' was built under R version 3.3.3
```

```
## Warning: package 'dplyr' was built under R version 3.3.3
```

```
## Warning: package 'stringr' was built under R version 3.3.3
```

```
## Warning: package 'forcats' was built under R version 3.3.3
```

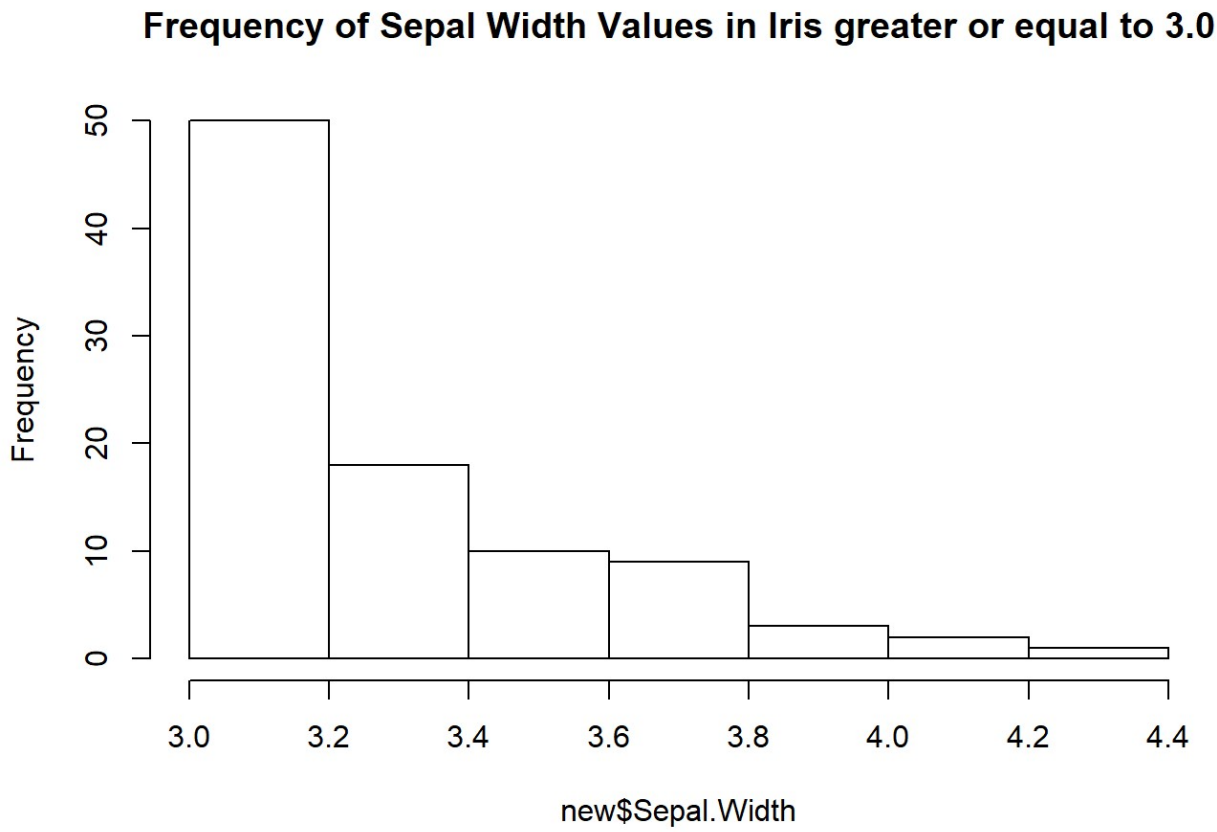
```
## -- Conflicts ------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```




```
new <- filter(iris, Sepal.Width >=3.0)
```

```
## Warning: package 'bindrcpp' was built under R version 3.3.3
```

```
hist(new$Sepal.Width,main ="Frequency of Sepal Width Values in Iris greater or equal t
o 3.0")
```




```
'/ Dont forget to ask Erin how to do this?
  n <- iris %>%
  filter(Sepal.Width >= 3.0) %>%
  select(Sepal.Width)%>%
  do(a=hist(.))
/'
```

```
## [1] "/ Dont forget to ask Erin how to do this?\n  n <- iris %>%\n  filter(Sepal.Wid
th >= 3.0) %>%\n  select(Sepal.Width)%>%\n  do(a=hist(.)) \n/"
```

Applying functions to dataframes

37. Try using apply on ‘d’. What happens? Why?

    - we get an error because the first column gender is of class factor while the others are of class numeric. In order for apply to work they all have to be of the same class type.

```
d <- data.frame(gender=c("M","M","F","M","F","F"),
        age=c(47,59,21,32,33,24),
        income=c(55000,88000,32450,76500,123000,45650))
d$over25 <- ifelse(d$age > 25, 1, 0) # 1 is row by row, 2 column by column
#apply(d,2, mean)

class(d$gender)
```

```
## [1] "factor"
```

```
#if we delete that column then we can use apply
d <- d[,-1]
apply(d,2,mean)
```

```
##          age       income       over25
## 3.600000e+01 7.010000e+04 6.666667e-01
```

Loops

38. What is a loop? Can you provide me with a real-life example in either remote sensing or your own research where such a thing would be handy?.

    - A loop is how you repeat the same set of instructions/opperations for a given set of variables/objects.
    - This is very useful in remote sensing when conducting time series analysis. It is particularly useful for image processing because you will want a large set of images to be processed in the same way.

39. What does it mean to “grow” variables or datasets using a loop? Show an example of code that does this.

    - Growing variables means that a variable assignment is made for every iteration.

```
#adapted from example on https://www.r-bloggers.com/a-tutorial-on-loops-in-r-usage-and
-alternatives/
mymat<-matrix(c(seq(1,25,1)), nrow = 5, ncol =5 ) # create matrix of normal random num
bers
mydframe=data.frame(mymat)    # transform into data frame
for (i in 1:2) {
   for (j in 1:2) {
     mydframe[i,j]<-mydframe[i,j] + 2
     print(mydframe)
   }
 }
```

```
##   X1 X2 X3 X4 X5
## 1  3  6 11 16 21
## 2  2  7 12 17 22
## 3  3  8 13 18 23
## 4  4  9 14 19 24
## 5  5 10 15 20 25
##   X1 X2 X3 X4 X5
## 1  3  8 11 16 21
## 2  2  7 12 17 22
## 3  3  8 13 18 23
## 4  4  9 14 19 24
## 5  5 10 15 20 25
##   X1 X2 X3 X4 X5
## 1  3  8 11 16 21
## 2  4  7 12 17 22
## 3  3  8 13 18 23
## 4  4  9 14 19 24
## 5  5 10 15 20 25
##   X1 X2 X3 X4 X5
## 1  3  8 11 16 21
## 2  4  9 12 17 22
## 3  3  8 13 18 23
## 4  4  9 14 19 24
## 5  5 10 15 20 25
```

40. In 500 words or less, discuss why you would want to write loops, why some people look down on them in R, what the suggested alternative is.

    - You would want to write loops when you

For Loops

41. Write a loop that calculates the square of the first 10 elements in the vector x, where x <- rnorm (40)

```
x <- rnorm(40)
for (i in 1:10){
  n <- x[i] ^ 2
  print(n)
}
```

```
## [1] 3.827145
## [1] 0.3749848
## [1] 0.869381
## [1] 1.339718
## [1] 0.243632
## [1] 0.8430399
## [1] 0.9953936
## [1] 0.9196293
## [1] 0.000994944
## [1] 0.5087072
```

For Loops

42. Write a while loop that prints the even numbers from 4 through 29.

```
for(i in 4:29)
{
    if(i %% 2 == 1) next
    print(i)
}
```

```
## [1] 4
## [1] 6
## [1] 8
## [1] 10
## [1] 12
## [1] 14
## [1] 16
## [1] 18
## [1] 20
## [1] 22
## [1] 24
## [1] 26
## [1] 28
```

43. Write a while loop that prints multiples of 10 from 100 to 190.

```
i <- 100
while (i <= 190){
  print(i)
  i <- i + 10
}
```

```
## [1] 100
## [1] 110
## [1] 120
## [1] 130
## [1] 140
## [1] 150
## [1] 160
## [1] 170
## [1] 180
## [1] 190
```

44. Create a 10 x 10 matrix of random numbers. Write a loop that replaces the 3rd row and 5th column of numbers with “ciao”.

```
m <-  matrix(runif(100), nrow= 10,ncol=10)

for (i in 1:10){
  if(i == 3  ){
    m[i,] <- "ciao"
  } else if( i == 10){
    m[,i] <- "ciao"
  }
}
m
```

```
##       [,1]                [,2]                [,3]
##  [1,] "0.190752455033362"  "0.506805869285017" "0.69676662934944"
##  [2,] "0.425314057385549"  "0.710725581506267" "0.067407117690891"
##  [3,] "ciao"               "ciao"              "ciao"
##  [4,] "0.983102553989738"  "0.858769533457235" "0.744285393739119"
##  [5,] "0.832967339083552"  "0.768407440977171" "0.715651279082522"
##  [6,] "0.919296364532784"  "0.352569436887279" "0.576098373625427"
##  [7,] "0.522789367474616"  "0.178407463012263" "0.758272334234789"
##  [8,] "0.692982271080837"  "0.994302849052474" "0.692374589154497"
##  [9,] "0.266104442533106"  "0.774222711101174" "0.273896565660834"
## [10,] "0.0257011821959168" "0.252824091585353" "0.362029532436281"
##       [,4]                [,5]                [,6]
##  [1,] "0.86275093536824"  "0.944747177651152"  "0.570118241943419"
##  [2,] "0.802917363587767" "0.901978265494108"  "0.213301240233704"
##  [3,] "ciao"              "ciao"               "ciao"
##  [4,] "0.49363612011075"  "0.136712177656591"  "0.601175104267895"
##  [5,] "0.583286171313375" "0.282478220993653"  "0.784025680273771"
##  [6,] "0.986846381099895" "0.40119218127802"   "0.51799276098609"
##  [7,] "0.464155935915187" "0.0421020686626434" "0.295340749667957"
##  [8,] "0.136314960196614" "0.707573836436495"  "0.388472435763106"
##  [9,] "0.793805054854602" "0.402393641648814"  "0.757860829355195"
## [10,] "0.143320393748581" "0.70962374471128"   "0.732923099305481"
##       [,7]                [,8]                 [,9]                [,10]
##  [1,] "0.882984599331394"  "0.0877220712136477" "0.804724732646719" "ciao"
##  [2,] "0.540680983802304"  "0.934747422812507"  "0.410372564801946" "ciao"
##  [3,] "ciao"               "ciao"               "ciao"              "ciao"
##  [4,] "0.264751341426745"  "0.316640836885199"  "0.974921484710649" "ciao"
##  [5,] "0.429258614545688"  "0.397228649351746"  "0.606906874570996" "ciao"
##  [6,] "0.464439331553876"  "0.41188743035309"   "0.448315153596923" "ciao"
##  [7,] "0.737862962065265"  "0.169730390422046"  "0.628092864761129" "ciao"
##  [8,] "0.876102001173422"  "0.900270892539993"  "0.784965183818713" "ciao"
##  [9,] "0.737852379214019"  "0.736159290652722"  "0.851221519056708" "ciao"
## [10,] "0.0168776114005595" "0.935266837943345"  "0.105244971113279" "ciao"
```

Boolean operators

45. Create a 100 x 100 matrix of random numbers. Use the system.time() function to measure the execution of a loop that calculates 10*sin(0.9*pi) on the matrix. Then use system.time() to measure the vectorized solution.

loop solution

```
set.seed(45);
m=100; n=100;
mymat<-matrix(runif(10000), nrow= 100,ncol=100) # create matrix
mydframe=data.frame(mymat)    # transform into data frame
system.time(
  for (i in 1:m) {
    for (j in 1:n) {
      mydframe[i,j]<-mydframe[i,j] + 10*sin(0.75*pi)
    }
  }
)
```

```
##    user  system elapsed
##    0.33    0.00    0.33
```

```
#   user  system elapsed
#   0.34    0.00    0.35
# loop solution
```

vector solution

```
system.time(
  mydframe<-mydframe + 10*sin(0.75*pi)
)
```

```
##    user  system elapsed
##       0       0       0
```

```
#user  system elapsed
  # 0.02    0.00    0.01
```

Functions

46. What are the general elements of a function? Which elements are not required?

    - name
    - argument
    - body of code which contains a return statement
    - An argument is not required. For example, there was that function that was function() in our pre-test. You do not need to have a return statement, but it is nesssecary if you want to pass a value from the local environment of the function to the global environment

47. How do you pass a value of an object from the local environment to the global environment (outside of a function)?

    - You pass the value of the object by using a return statement in your function.

48. What is the name of the R function that can help you identify errors in your function?
    - `debug()`
49. Aside from using the function identified in 3. above, what is another way to troubleshoot your functions? Hint: we used this in our first function oddcount in lab 1.
    - Put print statements throughout your function
50. What is the name of a function to write your own error messages when writing a function?.
    - `stop()` and `stopifnot`
51. Why do you want to keep your functions short?
    - Makes code cleaner and easier to test
    - Makes code easier to update
52. What are local versus global variables? Why do they matter when you write functions?
    - Local variables are the variables that are within the function themselves and global variables are the ones that make it to your R environment and that you can reference later and continue to do calculations on. They matter because we want to pass through the value of the local variable to a global variable by using `return()` so that we can continue to conduct analysis with this variable.

Creating Functions

53. What happens if you write the function as following in the code below and test it on wavelength = 0.02 m?

```
#wave_to_freq: converts wavelengths to frequency
wave_to_freq <- function(wave) {
  #note wavelength needs to be in meters
  freq <- (299792458/wave) #speed of light in m/s
  freq
}
paste("A wavelength of 10 cm has a frequency of", wave_to_freq(0.02),"Hz"  )
```

```
## [1] "A wavelength of 10 cm has a frequency of 14989622900 Hz"
```

54. Using the new function definition from 7., what happens if you assign the function wave_to_freq() to object x and call x?
    - I am a bit confused as to what 7 is referencing, but you cannot set a function to a variable if you include the () it will say that the arguement wave is missing with no default. If you just assign the function name to x, do not try to call the function, and print x, then R returns the body of the fuction/the setup of the fuction.

```
x <- wave_to_freq
x
```

```
## function(wave) {
##   #note wavelength needs to be in meters
##   freq <- (299792458/wave) #speed of light in m/s
##   freq
## }
```

55. Write a function that converts nanometers (nm) to meters (m).

```
# Converts nanometers to meters
nano_to_meters <- function(nano){
  meters <- nano * 1e-9
  return(meters)
}
nano_to_meters(3)
```

```
## [1] 3e-09
```

Compound Functions

56.Write a function that can take wavelength in micrometers as an input, and output the equivalent value in Giga Herz (GHz)

```
#converts micrometers to gigahertz
micro_to_ghz <- function(mm){
   giga <- (299792458/(mm*1e-6))*1e-9
   return(giga)
}
micro_to_ghz(1)
```

```
## [1] 299792.5
```

```
#double check that if you reference the function before you get the same answer
wave_to_freq(1e-6) * 1e-9
```

```
## [1] 299792.5
```

57. Using the function you wrote above, what radar band (provide the letter) corresponds to 30,000 micrometers?

    - You get the X band which is between 8 to 10 GHz

```
micro_to_ghz(30000)
```

```
## [1] 9.993082
```

lapply

58. Use lapply() and a function to find the coefficient of variation (the standard deviation divided by the mean) for all columns in the mtcars dataset.

```
m2 <- lapply(mtcars, function(x) sd(x) /mean(x))
m2
```

```
## $mpg
## [1] 0.2999881
##
## $cyl
## [1] 0.2886338
##
## $disp
## [1] 0.5371779
##
## $hp
## [1] 0.4674077
##
## $drat
## [1] 0.1486638
##
## $wt
## [1] 0.3041285
##
## $qsec
## [1] 0.1001159
##
## $vs
## [1] 1.152037
##
## $am
## [1] 1.228285
##
## $gear
## [1] 0.2000825
##
## $carb
## [1] 0.5742933
```

59. The function below scales a vector so it falls in the range [0, 1]. How would you apply it to every column of a data frame? How would you apply it to every numeric column in a data frame?

```
# apply to every column of the mtcars dataframe and print the head
scale01 <- function(x) {
  rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
head(sapply(mtcars, scale01))
```

```
##            mpg cyl      disp        hp      drat        wt      qsec vs am
## [1,] 0.4510638 0.5 0.2217511 0.2049470 0.5253456 0.2830478 0.2333333  0  1
## [2,] 0.4510638 0.5 0.2217511 0.2049470 0.5253456 0.3482485 0.3000000  0  1
## [3,] 0.5276596 0.0 0.0920429 0.1448763 0.5023041 0.2063411 0.4892857  1  1
## [4,] 0.4680851 0.5 0.4662010 0.2049470 0.1474654 0.4351828 0.5880952  1  0
## [5,] 0.3531915 1.0 0.7206286 0.4346290 0.1797235 0.4927129 0.3000000  0  0
## [6,] 0.3276596 0.5 0.3838863 0.1872792 0.0000000 0.4978266 0.6809524  1  0
##      gear      carb
## [1,]  0.5 0.4285714
## [2,]  0.5 0.4285714
## [3,]  0.5 0.0000000
## [4,]  0.0 0.0000000
## [5,]  0.0 0.1428571
## [6,]  0.0 0.0000000
```

```
# apply to just numeric columns
scale01_numeric <- function(x) {
  if(is.numeric(x)){
    rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
  }
}
iris_scaled <- iris
iris_scaled[] <- lapply(iris,scale01_numeric)
head(iris_scaled)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1   0.22222222   0.6250000   0.06779661  0.04166667
## 2   0.16666667   0.4166667   0.06779661  0.04166667
## 3   0.11111111   0.5000000   0.05084746  0.04166667
## 4   0.08333333   0.4583333   0.08474576  0.04166667
## 5   0.19444444   0.6666667   0.06779661  0.04166667
## 6   0.30555556   0.7916667   0.11864407  0.12500000
```

60. Use both for loops and lapply() to fit linear models to the mtcars data frame using the formulas stored in this list:

```
formulas <- list(
  mpg ~ disp,
  mpg ~ I(1 / disp),
  mpg ~ disp + wt,
  mpg ~ I(1 / disp) + wt
)

# for loop
for (i in 1:length(formulas)){
  mymodel <- lm(formulas[[i]], data=mtcars) #i
  print(mymodel)
}
```

```
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)         disp
##    29.59985     -0.04122
##
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)    I(1/disp)
##       10.75      1557.67
##
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)         disp           wt
##    34.96055     -0.01772     -3.35083
##
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)    I(1/disp)           wt
##      19.024     1142.560       -1.798
```

```
#list
lapply(formulas, lm, data=mtcars)
```

```
## [[1]]
##
## Call:
## FUN(formula = X[[i]], data = ..1)
##
## Coefficients:
## (Intercept)         disp
##    29.59985     -0.04122
##
##
## [[2]]
##
## Call:
## FUN(formula = X[[i]], data = ..1)
##
## Coefficients:
## (Intercept)    I(1/disp)
##       10.75      1557.67
##
##
## [[3]]
##
## Call:
## FUN(formula = X[[i]], data = ..1)
##
## Coefficients:
## (Intercept)         disp           wt
##    34.96055     -0.01772     -3.35083
##
##
## [[4]]
##
## Call:
## FUN(formula = X[[i]], data = ..1)
##
## Coefficients:
## (Intercept)    I(1/disp)           wt
##      19.024     1142.560       -1.798
```

61. What are the sys.time() calculations for the loop and lapply() solutions above?

```
# for loop
system.time(for (i in 1:length(formulas)){
  mymodel <- lm(formulas[[i]], data=mtcars) #i
  print(mymodel)
})
```

```
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)         disp
##    29.59985     -0.04122
##
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)    I(1/disp)
##       10.75      1557.67
##
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)         disp           wt
##    34.96055     -0.01772     -3.35083
##
##
## Call:
## lm(formula = formulas[[i]], data = mtcars)
##
## Coefficients:
## (Intercept)    I(1/disp)           wt
##      19.024     1142.560       -1.798
```

```
##    user  system elapsed
##       0       0       0
```

```
#   user  system elapsed
#   0.00    0.04    0.07

#list
system.time(lapply(formulas, lm, data=mtcars))
```

```
##    user  system elapsed
##       0       0       0
```

```
#   user  system elapsed
#   0.02    0.00    0.02
```