Bayesian by R

#Objectives for this part of the class

1. Learn about the different types of data handled by R
2. Learn about the types of objects R uses
3. Specify conditional statements
4. Looping using *for* and *while*
5. Writing and using your own R functions

https://bookdown.org/csgillespie/efficientR/

https://r4ds.had.co.nz/

Basic R programming notions

Types of Data

```
#-----------------------------------
#numeric
#create two R objects called x and y and save them, x should contain the value 5 and y the value 3
(x <- 5) #the paretheses surrounding x<-5, simply prints the constast of x
```

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

```
(y <- 3)
```

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

```
#-----------------------------------
#integer (create an integer object from the object x)
(z <- as.integer(x))
```

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

```
#-----------------------------------
#logical: create a new variable called test which stores the result of testing x>y
(test <- (x>y))
```

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

```
#-----------------------------------
#character: define an R object called "a", which stores the word (character string) "now"
a <- "now"
```

On numerical objects you can perform binary operations, such as sum, multiplication, division, etc.

Object Types in R

We already saw scalar, single string and logical objects of length 1. Now, let's learn about other types of objects where R can store data (vectors, character vectors, logical vectors, matrices, arrays, lists, data.frames)

Vectors

```
#-----------------------------------
#vectors
#create vector that contains the values 1.2, 4.3, 0.4, call it u, using the expression u<- c(,,)
u <- c(1.2, 4.3, 0.4)

#positions in vectors can be accessed to read or write by using square brackets and specifying the posi
u[2]
```

```
## [1] 4.3
```

```
u[c(3,1)]
```

```
## [1] 0.4 1.2
```

```
#-----------------------------------
#Access the second position of the vector u using the [2] and replace it's value with 10
u[2] <- 10

#-----------------------------------
#character vectors
#Create a string vector (call it s) that contains the words "hello" and "world"
s <- c("hello","world")

#-----------------------------------
#Create a logical vector (call it l) by testing the logical condition that values in the vector u (abov
l <- (u>1)
```

Single type objects with two or more dimensions

As with vectors, you can create matrices and arrays that store data as a single type.

```
#-----------------------------------
#character matrices
#Create a 2x3 matrix with the chacter vector c("a","b","c","d","e","f"),
#using the command matrix(data=##,nrow=##,ncol=##) (replace #### by the corresponding value)
#call this matrix "Sc"
Sc <- matrix(c("a","b","c","d","e","f"),nrow=2,ncol=3)

#Now create the matrix "Sr" using matrix(data=##,nrow=##,ncol=##,byrow=T)
Sr <- matrix(c("a","b","c","d","e","f"),nrow=2,ncol=3,byrow=T)

#-----------------------------------
#numerical matrices
#Create a 2x5 matrix with the numbers 1,2,3,...,10, call it A
(A <- matrix(1:10,nrow=2,ncol=5,byrow=T))
```

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

```
#As with vectors, to access positions in a matrix we use square brackets,
#but now we need to specify both row and column number, use [rownum,colnum],
#where "rownum" symbolizes the row number and "colnum" the column number
(A[2,4])
```

```
## [1] 9
```

```
(A[1:2,c(3,5)])
```

```
##      [,1] [,2]
## [1,]    3    5
## [2,]    8   10
```

```
#Create a new matrix B that is equal to A. Let's replace the value in the position
#(2,4) of B by 15
(B <- A)
```

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

```
(B[2,4] <- 15)
```

```
## [1] 15
```

```
#What happens to B if we replace that same position with a character string?
(B[2,4] <- "a")
```

```
## [1] "a"
```

```
#-----------------------------------
#logical matrices
#Let's create a logical matrix L that stores the outcome of the logical test
#that verifies the condition that a value in A is greater than 5
(L <- (A>5))
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] FALSE FALSE FALSE FALSE FALSE
## [2,]  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
#what happens to the matrix L if we replace position (2,3) with the value 5?
(L[2,3] <- 5)
```

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

```r
#-----------------------------------
#arrays (of any type)
#Arrays behave as matrices (in fact, matrices are two dimensional arrays) but
#can have more than 2 dimensions.
#Create a 3D array that contains 4 matrices of dimensions 2x5, each with
#the values 1,...,10, with the values entered by columns
#Notice here that the values in the vector 1:10 are recicled to fill in the array
(Arr <- array(data=1:10,dim=c(2,5,4)))
```

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

```r
#Notice how the array is filled in when I specify a vector with exactly
#the number of elements incldued in the array
(Arr <- array(data=1:40,dim=c(2,5,4)))
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   11   13   15   17   19
## [2,]   12   14   16   18   20
##
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   21   23   25   27   29
```

```
## [2,]   22   24   26   28   30
##
## , , 4
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   31   33   35   37   39
## [2,]   32   34   36   38   40
```

Multiple type objects

As we saw before, matrices and vectors can only store single type data. Many times datasets contain different types of data (i.e., numeric, character, logical). There are two very importan types of objects in R that can store this type of data without modifying their types.

```
#-----------------------------------
#lists
#Create a list named "thelist" with three positions (use the command "list").  In the first
#position add  the character vector s, in the second position include the logical
#value test, and in the third position include the numerical matrix A
(thelist <- list(s,test,A))
```

```
## [[1]]
## [1] "hello" "world"
##
## [[2]]
## [1] TRUE
##
## [[3]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```
#Access the second position in thelist and replace the value with FALSE
thelist[[2]] <- FALSE
thelist
```

```
## [[1]]
## [1] "hello" "world"
##
## [[2]]
## [1] FALSE
##
## [[3]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```
#Access the third position (matrix A) and replace the value in position (2,3) with 1000
thelist[[3]][2,3] <- 1000
thelist
```

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

```
## [1] "hello" "world"
##
## [[2]]
## [1] FALSE
##
## [[3]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7 1000    9   10
```

```
#Name the 3 positions in to thelist as "vecstr", "somelogic", and "thematrixA" respectively
names(thelist) <- c("vecstr", "somelogic", "thematrixA")
thelist
```

```
## $vecstr
## [1] "hello" "world"
##
## $somelogic
## [1] FALSE
##
## $thematrixA
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7 1000    9   10
```

```
#Access the position in the list called "vecstr", and replace its first position with
#the work "bye"
thelist$vecstr[1] <- "bye"
thelist
```

```
## $vecstr
## [1] "bye"   "world"
##
## $somelogic
## [1] FALSE
##
## $thematrixA
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7 1000    9   10
```

```
#-----------------------------------
#data frames
#Create a data frame using the command "data.frame", and include the vectors
#thevec=seq(1,10,by=2), thestring=c("a","b","c","d","e"), call it "thedata"
thedata <- data.frame(thevec=seq(1,10,by=2),thestring=c("a","b","c","d","e"))

#Print out only the first column in "thedata"
thedata[,1]
```

```
## [1] 1 3 5 7 9
```

```
thedata$thevec
```

```
## [1] 1 3 5 7 9
```

```
thedata[,"thevec"]
```

```
## [1] 1 3 5 7 9
```

```
thedata[[1]]
```

```
## [1] 1 3 5 7 9
```

```
thedata[["thevec"]]
```

```
## [1] 1 3 5 7 9
```

Looping

```
#-----------------------------------
#for
#create an empty vector called "result"
result <- numeric(0)

#Use a "for" loop to store in reverse order the values in the variable "thevec"
#inside of the data frame "thedata"
n <- length(thedata$thevec)
for(i in n:1){
  result[n+1-i] <- thedata$thevec[i]
}
result
```

```
## [1] 9 7 5 3 1
```

```
#-----------------------------------
#while
#Do the same excercise as above, but now use the "while" statement
i <- n
while(i > 0){
  result[n+1-i] <- thedata$thevec[i]
  i <- i-1
}
result
```

```
## [1] 9 7 5 3 1
```

Conditional statements

```
#-----------------------------------
#if statement
#Create an empty vector called "result2". Use a "for" loop, together with an
#if statement to pick out the values in the "thevec" variable inside of the
#"thedata" data frame, which correspond to the rows where the variable "thestring"
#is either "a", "d" or "e".  Store the corresponding values from "thevec" in "result2"
k <- 1
result2 <- numeric(0)
for(i in 1:n){
  #note here that I use the "|" command, which represents the or condition
  if(thedata$thestring[i]=="a" | thedata$thestring[i]=="d" | thedata$thestring[i]=="e"){
    result2[k] <- thedata$thevec[i]
    k <- k+1
  }
}
result2
```

```
## [1] 1 7 9
```

```
#-----------------------------------
#if else statement
#Create an empty vector called "result3".  Now, use a for loop in the same way as above,
#but in adition, for the values in "thevec" variable that are in rows where neither "a", "d"
#or "e" are, set those positions in result3 to 0
k <- 1
result3 <- numeric(0)
for(i in 1:n){
  #note here that I use the "|" command, which represents the or condition
  if(thedata$thestring[i]=="a" | thedata$thestring[i]=="d" | thedata$thestring[i]=="e"){
    result3[i] <- thedata$thevec[i]
  }else{
    result3[i] <- 0
  }
}
result3
```

```
## [1] 1 0 0 7 9
```

Writing and using your own functions

```
#-----------------------------------
#create a function that takes a value x and calculates the function
#f(x) = exp(x)-sin(x)*cos(x)
myfn <- function(x){

  f <- exp(x)-sin(x)*cos(x)

  return(f)
}
```

```
#create the function meanbyhand1 that takes as input a numeric variable,
#call it "thevar", and uses a "for" loop to calculate the mean of this variable
#the output of the meanbyhand function is the mean

#WARNING!!!! this is VERY INEFFICIENT, it's just to illustrate how to create a function


#create the function meanbyhand2 that uses the vectorized sum operation to
#calculate the mean of a variable


#Create the vector "mytestvector" that contains 10000 random values generated from a
#uniform distribution, and calculate the mean using the functions created above
```

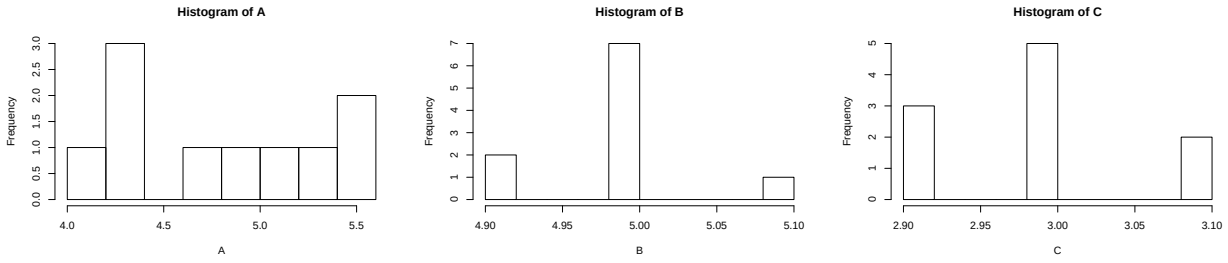
Some practical exercises with data

The weight of 10 objects whose true weight is 5 kg was measured on three instruments, with ten measurements on each one. The weighing measurements of are given below. Instrument A: 5.1, 5.4, 4.9, 5.6, 5.5, 4.8, 4.0, 4.3, 4.4, 4.3 Instrument B: 5.0, 5.0, 5.0, 5.0, 5.0, 5.0, 5.1, 5.0, 4.9, 4.9 Instrument C: 3.0, 3.1, 3.0, 3.1, 3.0, 3.0, 2.9, 2.9, 2.9, 3.0

a. Create objects named A, B, and C with the given data above. (use the c() function)

```
##  [1] 5.1 5.4 4.9 5.6 5.5 4.8 4.0 4.3 4.4 4.3
##  [1] 5.0 5.0 5.0 5.0 5.0 5.0 5.1 5.0 4.9 4.9
##  [1] 3.0 3.1 3.0 3.1 3.0 3.0 2.9 2.9 2.9 3.0
```

b. Create a histogram for each data set using hist() function.



c. Comment on the shape of each histogram.

d. Based on each histogram, identify an approximate value (a model) that represents the measurements for each machine. (That is, find the center of gravity for each histogram).

e. Create a function called "mysummary" that computes the median (using median() function), mean (using mean() function), and standard deviation (using sd() function) for the columns of any data frame.

```
mysummary <- function(x){

  median <- median(x)
```

```
  mean <- mean(x)
  standard_deviation <- sd(x)

  return(c(median, mean, standard_deviation))
}
```

f. Make a data frame named "Weights" that contains A, B, and C, name the columns Inst1, Inst2 and Inst3, respectively.

```
## 'data.frame':    10 obs. of  3 variables:
##  $ Inst1: num  5.1 5.4 4.9 5.6 5.5 4.8 4 4.3 4.4 4.3
##  $ Inst2: num  5 5 5 5 5 5 5.1 5 4.9 4.9
##  $ Inst3: num  3 3.1 3 3.1 3 3 2.9 2.9 2.9 3
```

g. Use the function "mysummary" created above to calculate the mean,median and standard deviations for all the columns in. Save these values to the data frame

	Inst1	Inst2	Inst3
1	5.1000000	5.0000000	3.0000000
2	5.4000000	5.0000000	3.1000000
3	4.9000000	5.0000000	3.0000000
4	5.6000000	5.0000000	3.1000000
5	5.5000000	5.0000000	3.0000000
6	4.8000000	5.0000000	3.0000000
7	4.0000000	5.1000000	2.9000000
8	4.3000000	5.0000000	2.9000000
9	4.4000000	4.9000000	2.9000000
10	4.3000000	4.9000000	3.0000000
median	4.8500000	5.0000000	3.0000000
mean	4.8300000	4.9900000	2.9900000
standard_deviation	0.5657836	0.0567646	0.0737865