Lab 2

Your Name Here

11:59PM February 25, 2021

More Basic R Skills

- Create a function `my_reverse` which takes as required input a vector and returns the vector in reverse where the first entry is the last entry, etc. No function calls are allowed inside your function otherwise that would defeat the purpose of the exercise! (Yes, there is a base R function that does this called `rev`). Use `head` on `v` and `tail` on `my_reverse(v)` to verify it works.

```
#TO-DO
my_reverse = function(v){
  v_rev = rep(NA, times=length(v))
  for(i in length(v):1){
    v_rev[length(v) - i + 1] = v[i] #reverse index
  }
  v_rev
}
#sanity check
v = 1:10
my_reverse(v)
```

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

- Create a function `flip_matrix` which takes as required input a matrix, an argument `dim_to_rev` that returns the matrix with the rows in reverse order or the columns in reverse order depending on the `dim_to_rev` argument. Let the default be the dimension of the matrix that is greater.

```
#TO-DO
flip_matrix = function(X, dim_to_rev=NULL){
  if(is.null(dim_to_rev)){
    dim_to_rev = ifelse(nrow(x) >= ncol(x), "rows", "cols")
  }
  if (dim_to_rev == "rows"){
    X[my_reverse(1:nrow(x)), ]
  } else if(dim_to_rev == "cols"){
    x[ ,my_reverse(1:ncol(x))]
  }else{
    stop("Illegal argument")
  }
}

x = matrix(rnorm(100),nrow=25)
x
```

```
##              [,1]        [,2]        [,3]         [,4]
##  [1,] -0.36201881  0.58805239 -0.14226364 -1.045503501
##  [2,]  0.46109992  0.54790932 -0.22185270  1.268906762
##  [3,] -0.42972194  2.52407099 -0.92395984  1.728578290
##  [4,] -0.95655590  0.42940512  1.14624134 -1.960074628
##  [5,]  1.12161485 -1.91308468  1.46307425 -0.001398607
##  [6,] -0.35873480  0.24300326 -0.40234817 -0.550885422
##  [7,]  0.77007066 -1.80327997  0.57304628  0.337155969
##  [8,]  0.03820303  0.10738859  1.12318491 -0.879785921
##  [9,]  0.63674874 -1.26941178  0.03489378 -0.499763704
## [10,] -0.14896686 -0.40844277 -0.61670700 -0.533024750
## [11,] -0.39423760 -0.51905546 -0.23551907 -0.146273763
## [12,] -0.26110319 -2.27527958  1.02363168  1.525089714
## [13,] -1.33697464 -0.09812065 -0.63831091  0.436984058
## [14,]  0.98063361 -0.70943692 -0.01135532  0.906014051
## [15,] -0.04446002  0.44145839  0.47381305  1.529595818
## [16,] -0.79311416  0.69093098 -0.12752487 -0.458314826
## [17,]  0.86874732  0.48361062 -1.14823672  1.470804448
## [18,] -0.12748059  0.20472599  0.46587766 -0.752002436
## [19,] -1.36968825 -0.60979235  1.01292792 -1.184606216
## [20,]  1.17632176 -0.36002499  1.97370236  2.256240082
## [21,]  1.48016025 -0.28797686 -0.27715812  0.768127639
## [22,]  0.31150954  0.89335712  0.42046590  0.178200982
## [23,] -0.49858942  0.33697082  0.96520018  1.099367992
## [24,] -1.58369042  0.81031798  0.75117369  0.302293281
## [25,]  0.51074520 -1.09918717 -2.21394782  0.086921080
```

```
flip_matrix(x)
```

```
##              [,1]        [,2]        [,3]         [,4]
##  [1,]  0.51074520 -1.09918717 -2.21394782  0.086921080
##  [2,] -1.58369042  0.81031798  0.75117369  0.302293281
##  [3,] -0.49858942  0.33697082  0.96520018  1.099367992
##  [4,]  0.31150954  0.89335712  0.42046590  0.178200982
##  [5,]  1.48016025 -0.28797686 -0.27715812  0.768127639
##  [6,]  1.17632176 -0.36002499  1.97370236  2.256240082
##  [7,] -1.36968825 -0.60979235  1.01292792 -1.184606216
##  [8,] -0.12748059  0.20472599  0.46587766 -0.752002436
##  [9,]  0.86874732  0.48361062 -1.14823672  1.470804448
## [10,] -0.79311416  0.69093098 -0.12752487 -0.458314826
## [11,] -0.04446002  0.44145839  0.47381305  1.529595818
## [12,]  0.98063361 -0.70943692 -0.01135532  0.906014051
## [13,] -1.33697464 -0.09812065 -0.63831091  0.436984058
## [14,] -0.26110319 -2.27527958  1.02363168  1.525089714
## [15,] -0.39423760 -0.51905546 -0.23551907 -0.146273763
## [16,] -0.14896686 -0.40844277 -0.61670700 -0.533024750
## [17,]  0.63674874 -1.26941178  0.03489378 -0.499763704
## [18,]  0.03820303  0.10738859  1.12318491 -0.879785921
## [19,]  0.77007066 -1.80327997  0.57304628  0.337155969
## [20,] -0.35873480  0.24300326 -0.40234817 -0.550885422
## [21,]  1.12161485 -1.91308468  1.46307425 -0.001398607
## [22,] -0.95655590  0.42940512  1.14624134 -1.960074628
## [23,] -0.42972194  2.52407099 -0.92395984  1.728578290
## [24,]  0.46109992  0.54790932 -0.22185270  1.268906762
```

```
## [25,] -0.36201881  0.58805239 -0.14226364 -1.045503501
```

- Create a list named `my_list` with keys "A", "B", … where the entries are arrays of size 1, 2 x 2, 3 x 3 x 3, etc. Fill the array with the numbers 1, 2, 3, etc. Make 8 entries according to this sequence.

```
#TO-DO
my_list = list()
counter = 1;
for( e  in LETTERS){
    if(counter >= 9){
      break;
    }
    my_list[[e]]= array(data=1:counter^counter,dim=c(rep(counter, times=counter)))
    counter = counter+1
}

# arrays[["C"]] = array(data=1:27,dim=c(3,3,3))
#commented print statement out to avoid 1k pages
#my_list
```

Run the following code:

```
#run this after the above exercise
lapply(my_list, object.size)
```

```
## $A
## 224 bytes
##
## $B
## 232 bytes
##
## $C
## 352 bytes
##
## $D
## 1248 bytes
##
## $E
## 12744 bytes
##
## $F
## 186864 bytes
##
## $G
## 3294416 bytes
##
## $H
## 67109104 bytes
```

Use `?object.size` to read about what these functions do. Then explain the output you see above. For the later arrays, does it make sense given the dimensions of the arrays?

#TO-DO Yes, it makes sense given the dimensions of the array because it increases exponentially.

Now cleanup the namespace by deleting all stored objects and functions:

```
#TO-DO
?object.size
```

```
## starting httpd help server ... done
```

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

A little about strings

- Use the `strsplit` function and `sample` to put the sentences in the string `lorem` below in random order. You will also need to manipulate the output of `strsplit` which is a list. You may need to learn basic concepts of regular expressions.

```
lorem = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi posuere varius volutpat. Morbi

#TO DO
split_lorem = strsplit(lorem, "[.] ")
lorem = sample(split_lorem[[1]], length(split_lorem[[1]]), replace = FALSE, prob = c(rep(1/length(split_
lorem = paste(c(lorem, ""), collapse =". ")
lorem
```

```
## [1] "Morbi posuere varius volutpat. Aenean nulla ante, iaculis sed vehicula ac, finibus vel arcu. Do
```

You have a set of names divided by gender (M / F) and generation (Boomer / GenX / Millenial):

- M / Boomer "Theodore, Bernard, Gene, Herbert, Ray, Tom, Lee, Alfred, Leroy, Eddie"
- M / GenX "Marc, Jamie, Greg, Darryl, Tim, Dean, Jon, Chris, Troy, Jeff"
- M / Millennial "Zachary, Dylan, Christian, Wesley, Seth, Austin, Gabriel, Evan, Casey, Luis"
- F / Boomer "Gloria, Joan, Dorothy, Shirley, Betty, Dianne, Kay, Marjorie, Lorraine, Mildred"
- F / GenX "Tracy, Dawn, Tina, Tammy, Melinda, Tamara, Tracey, Colleen, Sherri, Heidi"
- F / Millennial "Samantha, Alexis, Brittany, Lauren, Taylor, Bethany, Latoya, Candice, Brittney, Cheyenne"

Create a list-within-a-list that will intelligently store this data.

```
#HINT:
#strsplit("Theodore, Bernard, Gene, Herbert, Ray, Tom, Lee, Alfred, Leroy, Eddie", split = ", ")[[1]]
#TO-DO
men = list()
women = list()

men$Boomer = strsplit("Theodore, Bernard, Gene, Herbert, Ray, Tom, Lee, Alfred, Leroy, Eddie", split =
men$GenX = strsplit("Marc, Jamie, Greg, Darryl, Tim, Dean, Jon, Chris, Troy, Jeff", split = ", ")[[1]]
men$Millennial = strsplit("Zachary, Dylan, Christian, Wesley, Seth, Austin, Gabriel, Evan, Casey, Luis"
women$Boomer = strsplit("Gloria, Joan, Dorothy, Shirley, Betty, Dianne, Kay, Marjorie, Lorraine, Mildre
women$GenX = strsplit("Tracy, Dawn, Tina, Tammy, Melinda, Tamara, Tracey, Colleen, Sherri, Heidi", spli
women$Millennial = strsplit("Samantha, Alexis, Brittany, Lauren, Taylor, Bethany, Latoya, Candice, Brit

names_generation = list(men = men, women = women)

names_generation
```

```
## $men
## $men$Boomer
##  [1] "Theodore" "Bernard"  "Gene"     "Herbert"  "Ray"      "Tom"
##  [7] "Lee"      "Alfred"   "Leroy"    "Eddie"
##
## $men$GenX
##  [1] "Marc"   "Jamie"  "Greg"   "Darryl" "Tim"    "Dean"   "Jon"    "Chris"
##  [9] "Troy"   "Jeff"
##
## $men$Millennial
##  [1] "Zachary"   "Dylan"     "Christian" "Wesley"    "Seth"      "Austin"
##  [7] "Gabriel"   "Evan"      "Casey"     "Luis"
##
##
## $women
## $women$Boomer
##  [1] "Gloria"   "Joan"     "Dorothy"  "Shirley"  "Betty"    "Dianne"
##  [7] "Kay"      "Marjorie" "Lorraine" "Mildred"
##
## $women$GenX
##  [1] "Tracy"   "Dawn"    "Tina"    "Tammy"   "Melinda" "Tamara"  "Tracey"
##  [8] "Colleen" "Sherri"  "Heidi"
##
## $women$Millennial
##  [1] "Samantha" "Alexis"   "Brittany" "Lauren"   "Taylor"   "Bethany"
##  [7] "Latoya"   "Candice"  "Brittney" "Cheyenne"
```

Dataframe creation

Imagine you are running an experiment with many manipulations. You have 14 levels in the variable "treatment" with levels a, b, c, etc. For each of those manipulations you have 3 submanipulations in a variable named "variation" with levels A, B, C. Then you have "gender" with levels M / F. Then you have "generation" with levels Boomer, GenX, Millenial. Then you will have 6 runs per each of these groups. In each set of 6 you will need to select a name without duplication from the appropriate set of names (from the last question). Create a data frame with columns treatment, variation, gender, generation, name and y that will store all the unique unit information in this experiment. Leave y empty because it will be measured as the experiment is executed.

```
#TO-DO
n = 14 * 3 * 2 * 3 * 10
#X = data.frame(treatment = rep(NA,n), variation = rep(NA,n), gender = rep(NA,n), generation = rep(NA, 
X = data.frame(treatment = rep(letters[1:14], each = n/14), variation = rep(LETTERS[1:3], each = n/(14*3
```

```
## Warning in data.frame(treatment = rep(letters[1:14], each = n/14), variation
## = rep(LETTERS[1:3], : row names were found from a short variable and have been
## discarded
```

```
View(X)
#X
#summary(X$treatment)
#table(X$treatment)
#table(X$variation)
```

```
#table(X$gender)
#table(X$generation)
#table(X$name)
```

Packages

Install the package `pacman` using regular base R.

```
#install.packages("pacman")
```

First, install the package `testthat` (a widely accepted testing suite for R) from https://github.com/r-lib/testthat using `pacman`. If you are using Windows, this will be a long install, but you have to go through it for some of the stuff we are doing in class. LINUX (or MAC) is preferred for coding. If you can't get it to work, install this package from CRAN (still using `pacman`), but this is not recommended long term.

```
pacman::p_load(testthat)
```

- Create vector `v` consisting of all numbers from -100 to 100 and test using the second line of code su

```
v= seq(-100, 100)
expect_equal(v, -100 : 101)
```

If there are any errors, the `expect_equal` function will tell you about them. If there are no errors, then it will be silent.

Test the `my_reverse` function from lab2 using the following code:

```
v = 1:100
expect_equal(my_reverse(v), rev(v))

expect_equal(my_reverse(c("A", "B", "C")), c("C", "B", "A"))
```

Multinomial Classification using KNN

Write a $k = 1$ nearest neighbor algorithm using the Euclidean distance function. This is standard "Roxygen" format for documentation. Hopefully, we will get to packages at some point and we will go over this again. It is your job also to fill in this documentation.

```
#' Nearest neighbor classifier
#'
#' Classify an observation based on the label of the closest observation in the set of training observa
#'
#'
#'
#' @param Xinput      A matrix of features for training data observations
#' @param y_binary    The vector of training data labels
#' @param Xtest       A test observation as a row vector.
#' @return            The predicted label of the test observation
nn_algorithm_predict = function(Xinput, y_binary, xtest){
  #TO-DO
```

```
  n = nrow(Xinput)
  distances = array(NA, n)
  for(i in 1:n){
    distances[i] = sum((Xinput[i, ]-xtest)^2)
    #Xinput[1, ]-xtest gives u the differences between each dimension (diff between each feature of xte
  }

  y_binary[which.min(distances)]

}
```

Write a few tests to ensure it actually works:

```
#TO-DO
data(iris)
iris = iris[iris$Species != "virginica",]
x = iris[, 1:4]
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 25           4.8         3.4          1.9         0.2     setosa
## 26           5.0         3.0          1.6         0.2     setosa
## 27           5.0         3.4          1.6         0.4     setosa
## 28           5.2         3.5          1.5         0.2     setosa
## 29           5.2         3.4          1.4         0.2     setosa
## 30           4.7         3.2          1.6         0.2     setosa
## 31           4.8         3.1          1.6         0.2     setosa
## 32           5.4         3.4          1.5         0.4     setosa
## 33           5.2         4.1          1.5         0.1     setosa
```

```
## 34          5.5         4.2          1.4         0.2     setosa
## 35          4.9         3.1          1.5         0.2     setosa
## 36          5.0         3.2          1.2         0.2     setosa
## 37          5.5         3.5          1.3         0.2     setosa
## 38          4.9         3.6          1.4         0.1     setosa
## 39          4.4         3.0          1.3         0.2     setosa
## 40          5.1         3.4          1.5         0.2     setosa
## 41          5.0         3.5          1.3         0.3     setosa
## 42          4.5         2.3          1.3         0.3     setosa
## 43          4.4         3.2          1.3         0.2     setosa
## 44          5.0         3.5          1.6         0.6     setosa
## 45          5.1         3.8          1.9         0.4     setosa
## 46          4.8         3.0          1.4         0.3     setosa
## 47          5.1         3.8          1.6         0.2     setosa
## 48          4.6         3.2          1.4         0.2     setosa
## 49          5.3         3.7          1.5         0.2     setosa
## 50          5.0         3.3          1.4         0.2     setosa
## 51          7.0         3.2          4.7         1.4 versicolor
## 52          6.4         3.2          4.5         1.5 versicolor
## 53          6.9         3.1          4.9         1.5 versicolor
## 54          5.5         2.3          4.0         1.3 versicolor
## 55          6.5         2.8          4.6         1.5 versicolor
## 56          5.7         2.8          4.5         1.3 versicolor
## 57          6.3         3.3          4.7         1.6 versicolor
## 58          4.9         2.4          3.3         1.0 versicolor
## 59          6.6         2.9          4.6         1.3 versicolor
## 60          5.2         2.7          3.9         1.4 versicolor
## 61          5.0         2.0          3.5         1.0 versicolor
## 62          5.9         3.0          4.2         1.5 versicolor
## 63          6.0         2.2          4.0         1.0 versicolor
## 64          6.1         2.9          4.7         1.4 versicolor
## 65          5.6         2.9          3.6         1.3 versicolor
## 66          6.7         3.1          4.4         1.4 versicolor
## 67          5.6         3.0          4.5         1.5 versicolor
## 68          5.8         2.7          4.1         1.0 versicolor
## 69          6.2         2.2          4.5         1.5 versicolor
## 70          5.6         2.5          3.9         1.1 versicolor
## 71          5.9         3.2          4.8         1.8 versicolor
## 72          6.1         2.8          4.0         1.3 versicolor
## 73          6.3         2.5          4.9         1.5 versicolor
## 74          6.1         2.8          4.7         1.2 versicolor
## 75          6.4         2.9          4.3         1.3 versicolor
## 76          6.6         3.0          4.4         1.4 versicolor
## 77          6.8         2.8          4.8         1.4 versicolor
## 78          6.7         3.0          5.0         1.7 versicolor
## 79          6.0         2.9          4.5         1.5 versicolor
## 80          5.7         2.6          3.5         1.0 versicolor
## 81          5.5         2.4          3.8         1.1 versicolor
## 82          5.5         2.4          3.7         1.0 versicolor
## 83          5.8         2.7          3.9         1.2 versicolor
## 84          6.0         2.7          5.1         1.6 versicolor
## 85          5.4         3.0          4.5         1.5 versicolor
## 86          6.0         3.4          4.5         1.6 versicolor
## 87          6.7         3.1          4.7         1.5 versicolor
```

```
## 88          6.3         2.3          4.4         1.3 versicolor
## 89          5.6         3.0          4.1         1.3 versicolor
## 90          5.5         2.5          4.0         1.3 versicolor
## 91          5.5         2.6          4.4         1.2 versicolor
## 92          6.1         3.0          4.6         1.4 versicolor
## 93          5.8         2.6          4.0         1.2 versicolor
## 94          5.0         2.3          3.3         1.0 versicolor
## 95          5.6         2.7          4.2         1.3 versicolor
## 96          5.7         3.0          4.2         1.2 versicolor
## 97          5.7         2.9          4.2         1.3 versicolor
## 98          6.2         2.9          4.3         1.3 versicolor
## 99          5.1         2.5          3.0         1.1 versicolor
## 100         5.7         2.8          4.1         1.3 versicolor
```

```
y_bin = as.numeric(iris$Species != "setosa")

xtest_1 = c(6.4,    3.2,    4.5,    1.5) #1
xtest_0 = c(5.1,    3.5,    1.4,    0.2) #0
nn_algorithm_predict(x,y_bin, xtest_1) #get a 1
```

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

```
nn_algorithm_predict(x,y_bin, xtest_0) #get a 0
```

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

We now add an argument `d` representing any legal distance function to the `nn_algorithm_predict` function. Update the implementation so it performs NN using that distance function. Set the default function to be the Euclidean distance in the original function. Also, alter the documentation in the appropriate places.

```
#TO-DO
#' Nearest neighbor classifier
#'
#' Classify an observation based on the label of the closest observation in the se of training observat
#'
#'
#'
#' @param Xinput      A matrix of features for training data observations
#' @param y_binary    The vector of training data labels
#' @param Xtest       A test observation as a row vector.
#' @param d           A distance function which take inputs of two different row vectors
#' @return            The predicted label of the test observation
nn_algorithm_predict = function(Xinput, y_binary, xtest, d=function(v1,v2){
  sum((v1-v2)^2)
}){
  #TO-DO
  n = nrow(Xinput)
  distances = array(NA, n)
  for(i in 1:n){
    distances[i] = d(Xinput[i, ],xtest)
    #Xinput[1, ]-xtest gives u the differences between each dimension
  }
```

```
  y_binary[which.min(distances)]

}
```

For extra credit (unless you're a masters student), add an argument `k` to the `nn_algorithm_predict` function and update the implementation so it performs KNN. In the case of a tie, choose $\hat{y}$ randomly. Set the default `k` to be the square root of the size of $\mathcal{D}$ which is an empirical rule-of-thumb popularized by the "Pattern Classification" book by Duda, Hart and Stork (2007). Also, alter the documentation in the appropriate places.

```
#TO-DO for the 650 students but extra credit for undergrads
```

Basic Binary Classification Modeling

- Load the famous `iris` data frame into the namespace. Provide a summary of the columns using the `skim` function in package `skimr` and write a few descriptive sentences about the distributions using the code below and in English.

```
#TO-DO
data(iris)
pacman::p_load(skimr)
skim(iris)
```

Table 1: Data summary

Name	iris
Number of rows	150
Number of columns	5
Column type frequency:	
factor	1
numeric	4
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Species	0	1	FALSE	3	set: 50, ver: 50, vir: 50

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Sepal.Length	0	1	5.84	0.83	4.3	5.1	5.80	6.4	7.9	
Sepal.Width	0	1	3.06	0.44	2.0	2.8	3.00	3.3	4.4	
Petal.Length	0	1	3.76	1.77	1.0	1.6	4.35	5.1	6.9	
Petal.Width	0	1	1.20	0.76	0.1	0.3	1.30	1.8	2.5	

#TO-DO: describe this data There are 3 different species – virginica, setosa, and versicolor– and they are all evenly distrbuted (50 each). For Sepal.length, there is almost a uniform-like distributed percentile. While for Sepal.width, there is a normal distributed percentile. Both Petal.Length and Petal.width have a bimodal distribution percentile.

The outcome / label / response is `Species`. This is what we will be trying to predict. However, we only care about binary classification between "setosa" and "versicolor" for the purposes of this exercise. Thus the first order of business is to drop one class. Let's drop the data for the level "virginica" from the data frame.

```
#TO-DO
iris = iris[iris$Species != "virginica",]
```

Now create a vector `y` that is length the number of remaining rows in the data frame whose entries are 0 if "setosa" and 1 if "versicolor".

```
#TO-DO
#checking
iris$Species == 'setosa'
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [49]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE
```

```
as.numeric(iris$Species != 'setosa')
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
y = as.numeric(iris$Species != 'setosa')
y
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

- Write a function `mode` returning the sample mode.

```
#TO-DO
mode = function(v){
  sorted_counts = sort(table(v),decreasing=TRUE)
  names(sorted_counts[1])
}
mode(y)
```

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

```
#sanity check
```

- Fit a threshold model to `y` using the feature `Sepal.Length`. Write your own code to do this. What is the estimated value of the threshold parameter? Save the threshold value as `threshold`.

```
#TO-DO
rows = length(y)
num_errors_by_parameter = matrix(NA, nrow = rows, ncol= 2)
colnames(num_errors_by_parameter) = c("threshold_param", "num_errors")
for(i in 1:rows){
  cur_threshold = iris$Sepal.Length[i]
  num_errors = sum((iris$Sepal.Length > cur_threshold) != y)
  num_errors_by_parameter[i, ] = c(cur_threshold,num_errors)
}
num_errors_by_parameter
```

```
##        threshold_param num_errors
##   [1,]             5.1         18
##   [2,]             4.9         31
##   [3,]             4.7         39
##   [4,]             4.6         41
##   [5,]             5.0         25
##   [6,]             5.4         11
##   [7,]             4.6         41
##   [8,]             5.0         25
##   [9,]             4.4         46
##  [10,]             4.9         31
##  [11,]             5.4         11
##  [12,]             4.8         34
##  [13,]             4.8         34
##  [14,]             4.3         49
##  [15,]             5.8         24
##  [16,]             5.7         22
##  [17,]             5.4         11
##  [18,]             5.1         18
##  [19,]             5.7         22
##  [20,]             5.1         18
##  [21,]             5.4         11
##  [22,]             5.1         18
##  [23,]             4.6         41
##  [24,]             5.1         18
##  [25,]             4.8         34
##  [26,]             5.0         25
##  [27,]             5.0         25
##  [28,]             5.2         16
##  [29,]             5.2         16
##  [30,]             4.7         39
##  [31,]             4.8         34
##  [32,]             5.4         11
##  [33,]             5.2         16
##  [34,]             5.5         14
##  [35,]             4.9         31
##  [36,]             5.0         25
```

```
##  [37,]            5.5         14
##  [38,]            4.9         31
##  [39,]            4.4         46
##  [40,]            5.1         18
##  [41,]            5.0         25
##  [42,]            4.5         45
##  [43,]            4.4         46
##  [44,]            5.0         25
##  [45,]            5.1         18
##  [46,]            4.8         34
##  [47,]            5.1         18
##  [48,]            4.6         41
##  [49,]            5.3         15
##  [50,]            5.0         25
##  [51,]            7.0         50
##  [52,]            6.4         41
##  [53,]            6.9         49
##  [54,]            5.5         14
##  [55,]            6.5         42
##  [56,]            5.7         22
##  [57,]            6.3         39
##  [58,]            4.9         31
##  [59,]            6.6         44
##  [60,]            5.2         16
##  [61,]            5.0         25
##  [62,]            5.9         26
##  [63,]            6.0         30
##  [64,]            6.1         34
##  [65,]            5.6         19
##  [66,]            6.7         47
##  [67,]            5.6         19
##  [68,]            5.8         24
##  [69,]            6.2         36
##  [70,]            5.6         19
##  [71,]            5.9         26
##  [72,]            6.1         34
##  [73,]            6.3         39
##  [74,]            6.1         34
##  [75,]            6.4         41
##  [76,]            6.6         44
##  [77,]            6.8         48
##  [78,]            6.7         47
##  [79,]            6.0         30
##  [80,]            5.7         22
##  [81,]            5.5         14
##  [82,]            5.5         14
##  [83,]            5.8         24
##  [84,]            6.0         30
##  [85,]            5.4         11
##  [86,]            6.0         30
##  [87,]            6.7         47
##  [88,]            6.3         39
##  [89,]            5.6         19
##  [90,]            5.5         14
```

```
##  [91,]             5.5         14
##  [92,]             6.1         34
##  [93,]             5.8         24
##  [94,]             5.0         25
##  [95,]             5.6         19
##  [96,]             5.7         22
##  [97,]             5.7         22
##  [98,]             6.2         36
##  [99,]             5.1         18
## [100,]             5.7         22
```

```
num_errors_by_parameter[order(num_errors_by_parameter[, "num_errors"]), ]
```

```
##        threshold_param num_errors
##   [1,]             5.4         11
##   [2,]             5.4         11
##   [3,]             5.4         11
##   [4,]             5.4         11
##   [5,]             5.4         11
##   [6,]             5.4         11
##   [7,]             5.5         14
##   [8,]             5.5         14
##   [9,]             5.5         14
##  [10,]             5.5         14
##  [11,]             5.5         14
##  [12,]             5.5         14
##  [13,]             5.5         14
##  [14,]             5.3         15
##  [15,]             5.2         16
##  [16,]             5.2         16
##  [17,]             5.2         16
##  [18,]             5.2         16
##  [19,]             5.1         18
##  [20,]             5.1         18
##  [21,]             5.1         18
##  [22,]             5.1         18
##  [23,]             5.1         18
##  [24,]             5.1         18
##  [25,]             5.1         18
##  [26,]             5.1         18
##  [27,]             5.1         18
##  [28,]             5.6         19
##  [29,]             5.6         19
##  [30,]             5.6         19
##  [31,]             5.6         19
##  [32,]             5.6         19
##  [33,]             5.7         22
##  [34,]             5.7         22
##  [35,]             5.7         22
##  [36,]             5.7         22
##  [37,]             5.7         22
##  [38,]             5.7         22
##  [39,]             5.7         22
##  [40,]             5.8         24
```

```
##  [41,]            5.8         24
##  [42,]            5.8         24
##  [43,]            5.8         24
##  [44,]            5.0         25
##  [45,]            5.0         25
##  [46,]            5.0         25
##  [47,]            5.0         25
##  [48,]            5.0         25
##  [49,]            5.0         25
##  [50,]            5.0         25
##  [51,]            5.0         25
##  [52,]            5.0         25
##  [53,]            5.0         25
##  [54,]            5.9         26
##  [55,]            5.9         26
##  [56,]            6.0         30
##  [57,]            6.0         30
##  [58,]            6.0         30
##  [59,]            6.0         30
##  [60,]            4.9         31
##  [61,]            4.9         31
##  [62,]            4.9         31
##  [63,]            4.9         31
##  [64,]            4.9         31
##  [65,]            4.8         34
##  [66,]            4.8         34
##  [67,]            4.8         34
##  [68,]            4.8         34
##  [69,]            4.8         34
##  [70,]            6.1         34
##  [71,]            6.1         34
##  [72,]            6.1         34
##  [73,]            6.1         34
##  [74,]            6.2         36
##  [75,]            6.2         36
##  [76,]            4.7         39
##  [77,]            4.7         39
##  [78,]            6.3         39
##  [79,]            6.3         39
##  [80,]            6.3         39
##  [81,]            4.6         41
##  [82,]            4.6         41
##  [83,]            4.6         41
##  [84,]            4.6         41
##  [85,]            6.4         41
##  [86,]            6.4         41
##  [87,]            6.5         42
##  [88,]            6.6         44
##  [89,]            6.6         44
##  [90,]            4.5         45
##  [91,]            4.4         46
##  [92,]            4.4         46
##  [93,]            4.4         46
##  [94,]            6.7         47
```

```
##  [95,]             6.7         47
##  [96,]             6.7         47
##  [97,]             6.8         48
##  [98,]             4.3         49
##  [99,]             6.9         49
## [100,]             7.0         50
```

```
best_row = order(num_errors_by_parameter[, "num_errors"])[1]
threshold = c(num_errors_by_parameter[best_row, "threshold_param"], use.names = FALSE)
threshold
```

```
## [1] 5.4
```

What is the total number of errors this model makes?

```
#TO-DO
total_error_model = sum((iris$Sepal.Length > threshold) != y)
total_error_model
```

```
## [1] 11
```

Does the threshold model's performance make sense given the following summaries:

```
threshold
```

```
## [1] 5.4
```

```
summary(iris[iris$Species == "setosa", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   4.800   5.000   5.006   5.200   5.800
```

```
summary(iris[iris$Species == "versicolor", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.900   5.600   5.900   5.936   6.300   7.000
```

#TO-DO: Write your answer here in English. Yes. This answer makes sense by the looks of the data distribution from the median and mean. Most setosa sepal length are around 5.0 according to the median (and is similar to the average sepal length of setosas). Most versicolor length are around 5.9 according to the median and the average length of sepal is 5.936. Therefore, having 5.4 as the threshold makes sense as anything above 5.4 will be considered versicolor from the data distribution.

Create the function `g` explicitly that can predict `y` from `x` being a new `Sepal.Length`.

```
g = function(x){
  #TO-DO
  ifelse(x > threshold, 1, 0)
}

#sanity check
g(6) #versicolor
```

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

```
g(3) #setosa
```

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

Perceptron

You will code the "perceptron learning algorithm" for arbitrary number of features p. Take a look at the comments above the function. Respect the spec below:

```
#' Perceptron Learning Algorithm
#'
#' Finds the best line of separation for the data.
#'
#' @param Xinput      A matrix of features for training data observations
#' @param y_binary    The binary vector of training data labels
#' @param MAX_ITER    Max number of iterations
#' @param w           vector of Weights for the features
#'
#' @return            The computed final parameter (weight) as a vector of length p + 1
perceptron_learning_algorithm = function(Xinput, y_binary, MAX_ITER = 1000, w = NULL){
  #TO-DO
  X1 = as.matrix(cbind(1, Xinput[, ,drop = FALSE]))
  print(X1)
  print(Xinput)
  if( length(w) == 0 ){
    w = rep(0, ncol(X1))
  }
  for(t in 1 : MAX_ITER){
    for(i in 1 : nrow(X1)){
      X1_i = X1[i, ]
      yhat_i = ifelse(sum(X1_i * w) > 0, 1, 0)
      y_i = y_binary[i]
      for( j in 1 : length(w)){
        w[j] = w[j] + (y_i - yhat_i)* X1_i[j]
      }
    }
  }
  w
}
```

To understand what the algorithm is doing - linear "discrimination" between two response categories, we can draw a picture. First let's make up some very simple training data $\mathbb{D}$.

```
Xy_simple = data.frame(
 response = factor(c(0, 0, 0, 1, 1, 1)), #nominal
 first_feature = c(1, 1, 2, 3, 3, 4),    #continuous
 second_feature = c(1, 2, 1, 3, 4, 3)    #continuous
)
```

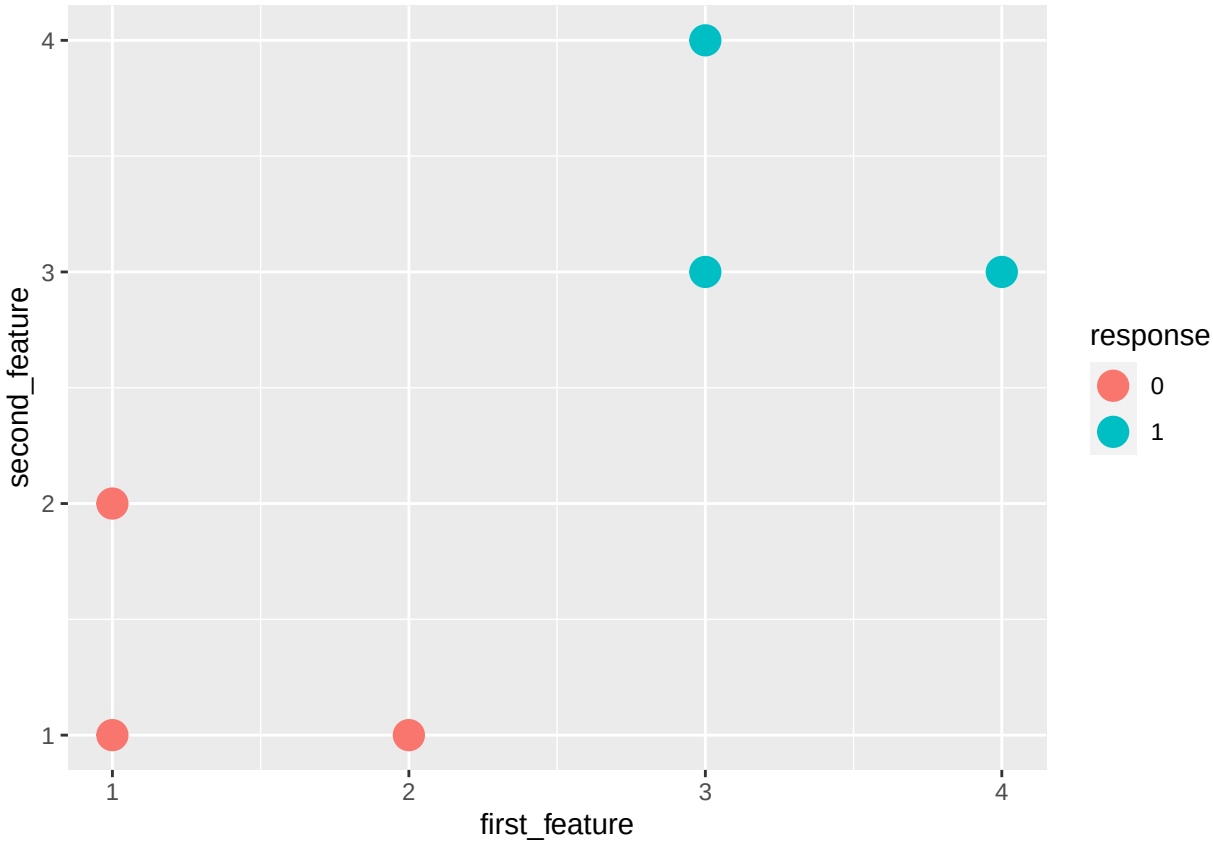
We haven't spoken about visualization yet, but it is important we do some of it now. Thus, I will write this code for you and you will just run it. First we load the visualization library we're going to use:

```
pacman::p_load(ggplot2)
```

We are going to just get some plots and not talk about the code to generate them as we will have a whole unit on visualization using `ggplot2` in the future.

Let's first plot y by the two features so the coordinate plane will be the two features and we use different colors to represent the third dimension, y.

```
simple_viz_obj = ggplot(Xy_simple, aes(x = first_feature, y = second_feature, color = response)) +
  geom_point(size = 5)
simple_viz_obj
```



#TO-DO: Explain this picture. This picture shows a perfectly linear separable data set.

Now, let us run the algorithm and see what happens:

```
w_vec_simple_per = perceptron_learning_algorithm(
  cbind(Xy_simple$first_feature, Xy_simple$second_feature),
  as.numeric(Xy_simple$response == 1))
```

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

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

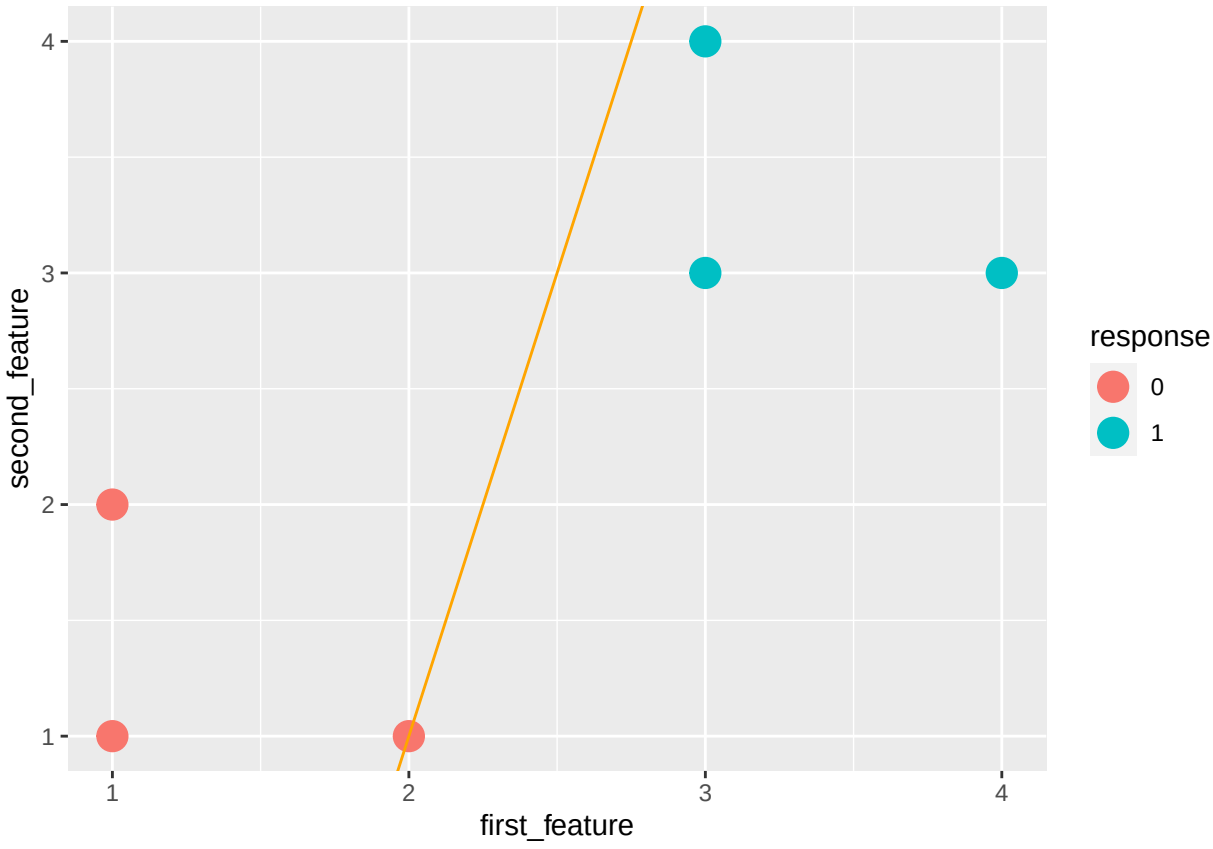
```
w_vec_simple_per
```

```
## [1] -7  4 -1
```

Explain this output. What do the numbers mean? What is the intercept of this line and the slope? You will have to do some algebra.

#TO-DO -7, 4, -1 represents the bias, weight_0, weight_1, respectively. The intercept of the line is -7 and the slope is 4.

```
simple_perceptron_line = geom_abline(
  intercept = -w_vec_simple_per[1] / w_vec_simple_per[3],
  slope = -w_vec_simple_per[2] / w_vec_simple_per[3],
  color = "orange")
simple_viz_obj + simple_perceptron_line
```



Explain this picture. Why is this line of separation not "satisfying" to you?

#TO-DO

This line of separation is not satisfying because it is not separating the points in the data set by being between them. In fact, it's actually going through one of the data set points which is not really "separating" the points.

For extra credit, program the maximum-margin hyperplane perceptron that provides the best linear discrimination model for linearly separable data. Make sure you provide ROxygen documentation for this function.x

```
#TO-DO
```