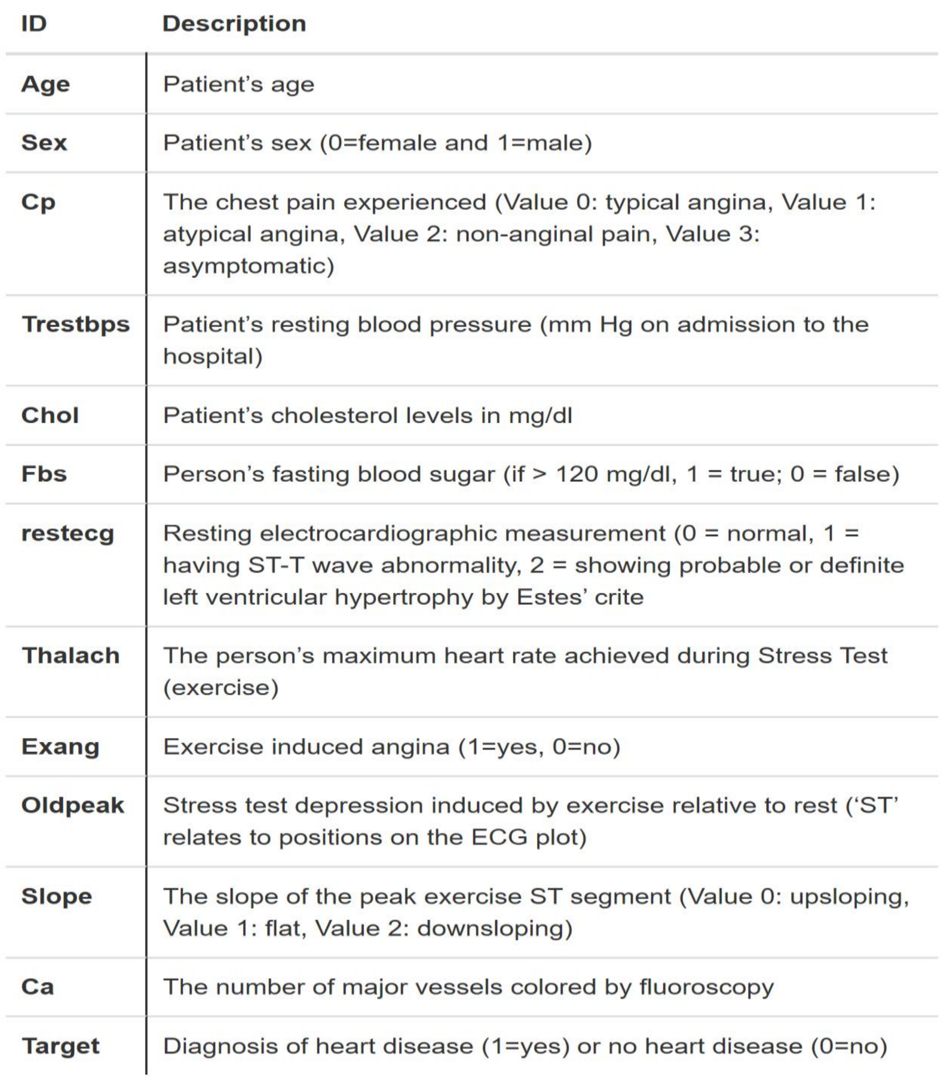
R Programming Assignment - Final

Elianne Mora

The Data Set

In this project we analyze the ‘Heart Disease’ dataset from the clinical and noninvasive test results of 303 patients at the Cleveland Clinic in Cleveland, Ohio. While the original data collected from the 303 patients contained 75 variables, the final subset that was made available contained only 14. We have focused on the subset data for the application of KNN classification tool; however, only 13 variables were kept. The variable *thal* was eliminated due to inconsistent reporting of the values and their respective meanings.

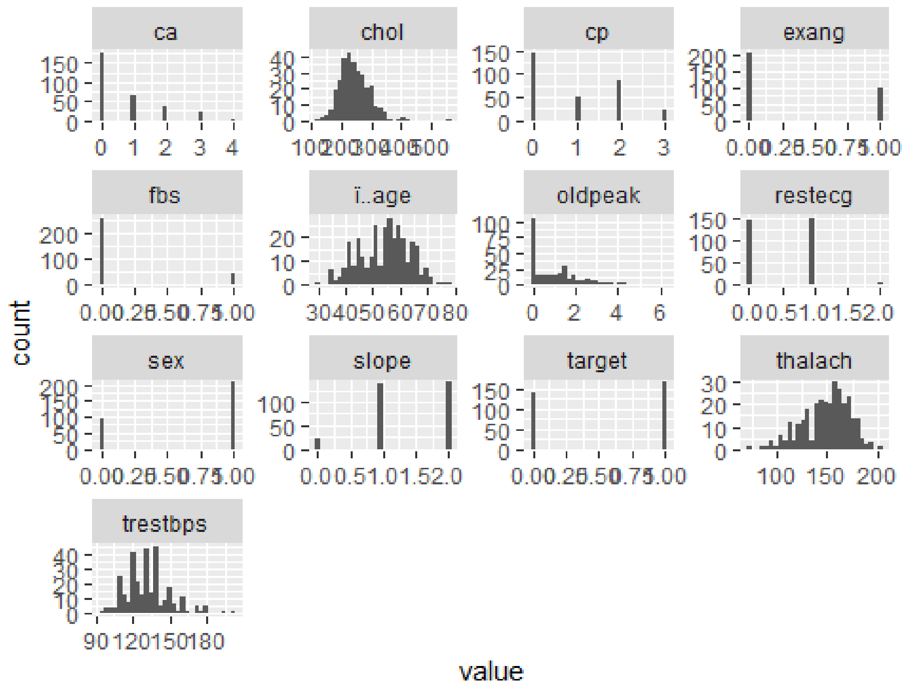


ID	Description
Age	Patient’s age
Sex	Patient’s sex (0=female and 1=male)
Cp	The chest pain experienced (Value 0: typical angina, Value 1: atypical angina, Value 2: non-anginal pain, Value 3: asymptomatic)
Trestbps	Patient’s resting blood pressure (mm Hg on admission to the hospital)
Chol	Patient’s cholesterol levels in mg/dl
Fbs	Person’s fasting blood sugar (if > 120 mg/dl, 1 = true; 0 = false)
restecg	Resting electrocardiographic measurement (0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes’ crite
Thalach	The person’s maximum heart rate achieved during Stress Test (exercise)
Exang	Exercise induced angina (1=yes, 0=no)
Oldpeak	Stress test depression induced by exercise relative to rest (‘ST’ relates to positions on the ECG plot)
Slope	The slope of the peak exercise ST segment (Value 0: upsloping, Value 1: flat, Value 2: downsloping)
Ca	The number of major vessels colored by fluoroscopy
Target	Diagnosis of heart disease (1=yes) or no heart disease (0=no)

In this project we are interested in the predictive capabilities of the K-nn algorithm we developed, for the classification of the variable *Target*, given all other records. We wish to find the optimum k value that will yield a high level of accuracy in the prediction of a diagnosis. In this scenario, wrongly diagnosing a patient with heart disease in the absence of it, is as bad as misdiagnosing a person who is in fact ill. In the first case, the patient will likely be admitted to a hospital, further incur medical expenses, intake unnecessary drugs, be submitted to further tests and procedures, etc. The latter will be sent home at risk of having a stroke, heart attack, further complications affecting daily life, and even death.

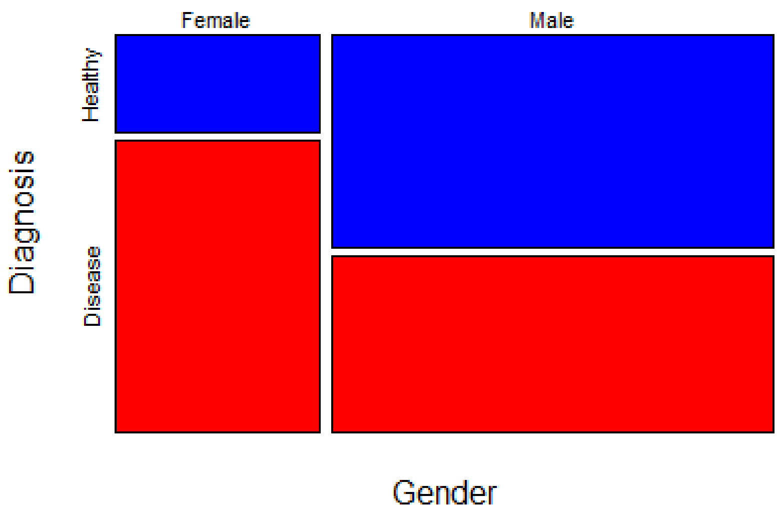
Data Visualization

The data collected was complete, no missing values nor zeros were found in the continuous variables.

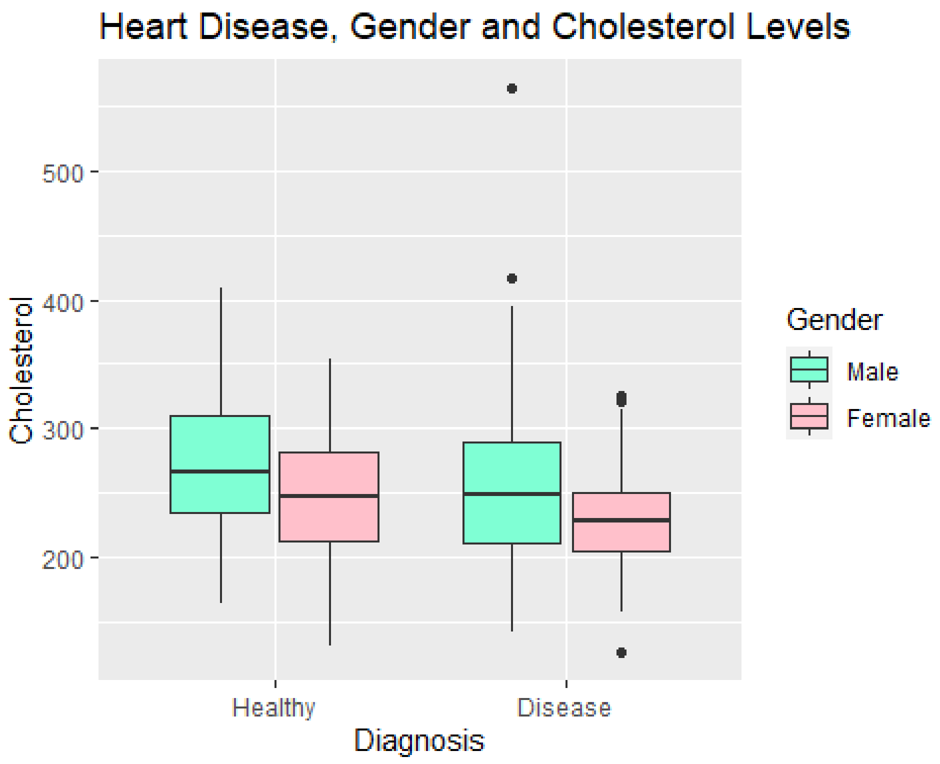


From the table below we can observe that the dataset contained 96 females of which 72 were diagnosed with heart disease (75%), while out of 207 males, 93 were diagnosed with heart disease (45%).

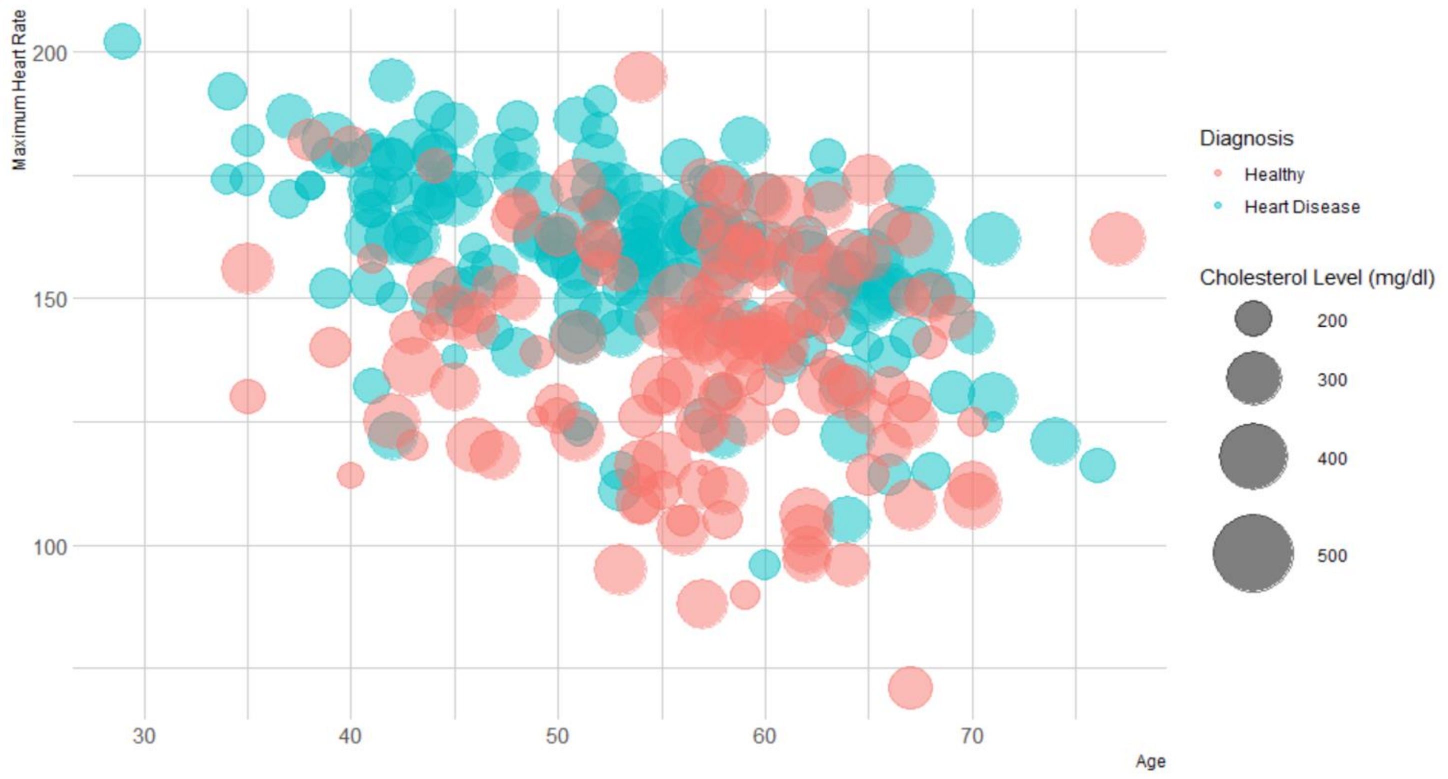
	Diagnosis	
	Healthy	**Disease**
Gender		
Female	24	72
Male	114	93
#Total cases	138	165



The literature shows that men and women are equally at risk of heart disease. While genetics and family history are significant factors, other characteristics that increase the risk of heart disease are age, unhealthy diets and sedentarism leading to high blood pressure and high cholesterol levels, among many others. Our data, however, shows that most of the patients diagnosed with heart disease have cholesterol levels equal to or below those of the subjects with negative diagnosis.



From the plot below we may conclude that there seems to be no definitive difference in diagnosis given age, cholesterol levels and the maximum heart rate achieved during the stress test. Paying close attention to the sick patients (blue points), on the left upper corner we observe the youngest patient (29 year old male) with a maximum rate of 202 bpm and a cholesterol level of 204 mg/dl. In contrast, we observe the oldest healthy patient (77 year old male), with a maximum heart rate of 162 bpm and a cholesterol level of 304 mg/dl.



K-nn Algorithm

Our data set contains 12 attributes that will help us predict whether a new patient has heart disease or not.

The we developed Knn function takes 5 inputs:

- train: data set used for training, in our case 50% of the original dataset, without the target variable (151x12 matrix of attributes).
- target: the target is a column vector extracted from the training dataset, it includes the classification (Disease or Healthy) only.
- observation: this input takes a matrix of attributes only (testing or validation set), so that our function can predict the target values.
- k_val: this input takes an integer such that we check the kth closest neighbors in order to determine the category
- dmethod: takes a character input where we specify how to calculate the distances, in our case there are only two options “euclidean” or “manhattan”.

If the *Euclidian* distance is selected, then we calculate the distance of the new observation from the already existing set as

$$\sqrt{(age_1 - age_2) + (sex_1 - sex_2) + (cp_1 - cp_2) + (trestbps_1 + tresbps_2) + (chol_1 + chol2) + \ldots}$$

where the one represents the observed data and two refers to the new observations for which we want to determine the diagnosis. If the *Manhattan* distance is selected, the calculation is simply the absolute difference between the points. The function is found below:

```
my_KNN<-function(train,target,observation, k_val, dmethod="euclidean"){
  #first we establish the calculation of the distances given the chosen input method
  if(dmethod == "euclidean"){
    distancias = sqrt(rowSums(sweep(train,2,as.numeric(observation))^2))
  }
  if(dmethod == 'manhattan'){
    distancias = rowSums(abs(sweep(train,2,as.numeric(observation))))
  }
  #the distances are ordered in ascending order so that we can establish the category to
be predicted, given its closeness to the majority of the neighbors.
  indices = order(distancias)
  target_ordered = target[indices]
  k_obs = target_ordered[1:k_val]
  prediction = names(which.max(table(k_obs)))
  return(prediction)
}
```

Accuracy

In order to determine the optimum value of k, we will measure the accuracy of the prediction given by our algorithm by comparing it to the actual labels of the testing/validation set.

```
#Funtion accuracy - we include the computation of the accuracy of the predictions. Here
#the prediction is held against the true reported values, if it's correctly labeled then
#we add it to the count of "correct" labels.
accuracy<-function(test){
  correct=0
  for(i in c(1:nrow(test))){
    if(test[i,1]==test[i,2]){
      correct=correct+1
    }
  }
  real_accuracy=correct/nrow(test)*100
  return(real_accuracy)
}
```

Data Partition

Before we implement our function, the data is scaled with the exception of our *target* variable, which contains the labels to be predicted. Also, the data is randomly split into three different sets, where 50% is used as training set, 25% is used as a testing set, and lastly 25% as a validation set.

```
heart<-data.frame(scale(Heart[,c(1,2,3,4,5,6,7,8,9,10,11,12)]),Heart[,13])

intrain <- sample(3, nrow(heart), replace = TRUE, prob = c(0.5, 0.25,0.25))

train <-heart[intrain==1,]
test <- heart[intrain==2,]
validate<-heart[intrain==3,]
```

K-nn Implementation: Euclidean vs. Manhattan Distance (K=1,...5)

Now, we implement our algorithm such that it takes different values of k (1-5). The predicted diagnosis is "Healthy" and reported accuracies using the Euclidean distance is displayed below:

```
################# K = 1 & euclidian methods
tries = c()
for(i in 1:nrow(test)){
  tries = c(tries,my_KNN(train[,-13],train[,13],test[i,-13],1,dmethod="euclidean"))}
```

```
check = data.frame(as.factor(tries),test[,13])
ac1=accuracy(check)

################# K = 2 & euclidean method
tries = c()
for(i in 1:nrow(test)){
  tries = c(tries,my_KNN(train[,-13],train[,13],test[i,-13],2,dmethod="euclidean"))}

check = data.frame(as.factor(tries),test[,13])
ac2=accuracy(check)

################# K = 3 & euclidean method
tries = c()
for(i in 1:nrow(test)){
  tries = c(tries,my_KNN(train[,-13],train[,13],test[i,-13],3,dmethod="euclidean"))}

check = data.frame(as.factor(tries),test[,13])
ac3=accuracy(check)

################# K = 4 & euclidean method
tries = c()
for(i in 1:nrow(test)){
  tries = c(tries,my_KNN(train[,-13],train[,13],test[i,-13],4,dmethod="euclidean"))}

check = data.frame(as.factor(tries),test[,13])
ac4=accuracy(check)

################# K = 5 & euclidean method
tries = c()
for(i in 1:nrow(test)){
  tries = c(tries,my_KNN(train[,-13],train[,13],test[i,-13],5,dmethod="euclidean"))}

check = data.frame(as.factor(tries),test[,13])
ac5=accuracy(check)

data_acc = data.frame(1:5)
data_acc=cbind(data_acc,c(ac1,ac2,ac3,ac4,ac5))
colnames(data_acc)=c('K',"Accuracy")
data_acc

##   K Accuracy
## 1 1 71.64179
## 2 2 65.67164
## 3 3 70.14925
## 4 4 71.64179
## 5 5 71.64179
```

Likewise, we implement our algorithm using the training and testing sets and the Manhattan distance. The reported accuracies are as follows:

```
##   K Accuracy
## 1 1 73.13433
## 2 2 68.65672
## 3 3 70.14925
## 4 4 67.16418
## 5 5 71.64179
```

Resampling and Knn

In this section, we resample our data at random 100 times. Then our algorithm is implemented for the i^th data sample, using the validation set, which takes values of k between 1-10. This process yields a 10x101 matrix (first column includes the values of k), which is then converted into a 10x2 matrix since we compute the mean accuracies for each k-value.

The computing time of our serial function, given the 100 random partitions, the 10 values of k used and the euclidean distance, is between 315-350 seconds.

```
#Now we add the random sampling (repeated 100 times) to the above loop

#initialize the dataframe where all accuracies will be recorded along with their K-val

data_100 = data.frame(1:10)
colnames(data_100) = 'k'
#initialize loop with 1-100 resamplings
tic()
for(j in 1:100){
  intrain <- sample(3, nrow(heart), replace = TRUE, prob = c(0.5, 0.25,0.25))

  train <-heart[intrain==1,]
  test <- heart[intrain==2,]
  validate<-heart[intrain==3,]
  #initialize empty col vector where accuracies will be stored
  accuracies = c()
  #include loop for k values to be implemented with the validation set
  for(k in 1:10){
    tries = c()
    for(i in 1:nrow(validate)){
      tries = c(tries,my_KNN(train[,-13],train[,13],validate[i,-13],k,dmethod="euclidean"
))}
    #include accuracy computation by sample per K-value
    check = data.frame(as.factor(tries),validate[,13])
    accuracies = c(accuracies,accuracy(check))
  }
  #this dataframe will have dimension 10x101 - 10 k-vals and 100 accuracies by k-val.
  data_100 = cbind(data_100,accuracies)

}
toc()

## 334.95 sec elapsed

#we create a new dataframe were accuracies are grouped by k value and their means
#are computed
final_accuracies = data.frame(data_100$k,apply(data_100[,2:100],1,mean))
colnames(final_accuracies) = c('k','mean accuracy')
#sort(final_accuracies[,2])
final_accuracies[order(final_accuracies$`mean accuracy`, decreasing = TRUE),]

##     k mean accuracy
## 10 10      79.54496
## 9   9      79.48258
## 7   7      78.92053
## 8   8      78.73182
## 5   5      78.61211
## 3   3      78.28748
## 6   6      77.80057
## 4   4      77.13463
## 1   1      74.61463
## 2   2      73.84829
```

We also complete the same process using the Manhattan distance. The results below show that, given the mean accuracy reported, the optimum value of K would be 9. In contrast, the optimum value of K when using the euclidean distance computation is 10. This implementation takes between 200-225 seconds, a bit faster than using the euclidean distance.

```
## 219.59 sec elapsed

##      k mean accuracy
## 9    9      80.88611
## 7    7      80.84328
## 10 10      80.79643
## 8    8      80.26684
## 5    5      79.79672
## 3    3      79.79196
## 6    6      79.61857
## 4    4      78.73570
## 1    1      76.14922
## 2    2      74.93947
```

The Best Value of K

We will now implenment our Knn function for the testing set using Euclidean distance. From the previous output, the optimum value of K to be use to classify the test set was either 9 or 10, depending on the distance calculation selected. From the results below, we can see that the accuracy reported for k=9 is higher than the accuracy reported in the validation set.

```
##      k Accuracy
## 9    9 84.41558
## 7    7 83.11688
## 8    8 83.11688
## 10 10 83.11688
## 6    6 81.81818
## 5    5 80.51948
## 1    1 79.22078
## 3    3 77.92208
## 4    4 76.62338
## 2    2 70.12987
```

Parallel Programming

In this section we implement the random resample of our data (100 times) and the implementation of our Knn function for K=1,...,10.

The parallelization yields results within 90-100 seconds, which is a lot more faster than the serial version.

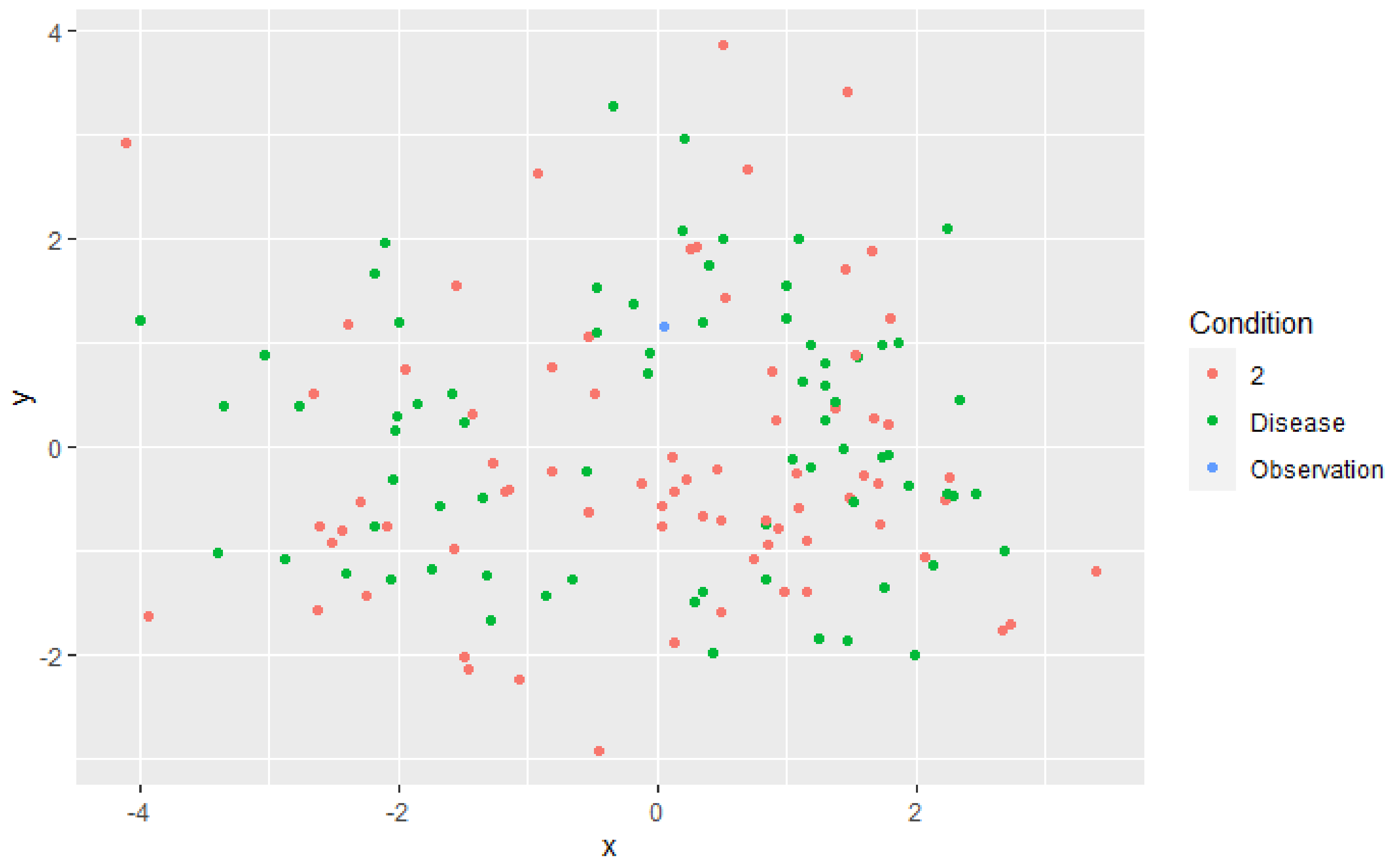
```
## 103.12 sec elapsed

##      k mean accuracy
## 10 10      79.26286
## 9   9      79.21518
## 3   3      78.51888
## 7   7      78.46828
## 8   8      78.01733
## 5   5      77.98088
## 6   6      77.46436
## 4   4      76.59314
## 1   1      74.60389
## 2   2      73.19830
```

Adding a Plot as Output

Finally, we attempt to add another output to our function: a plot. We use the testing set, however, in this instance we only take one row of attributes (1x12) instead of all the attributes of the testing set. The resulting plot then shows all observed values from the training set and their respective classification and the label of the observation as predicted by our Knn algorithm. In this instance, our function predicted a diagnosis of heart disease. The plot below shows that our observation (in blue) is in fact surrounded by other observed patients with heart disease – its nearest neighbors are all labeled as “Disease”.

These results go hand-in-hand with the previous values of k (9,10). Given the nature of the data, it is better to evaluate a high number of neighbors in order to better classify it.


x
y
Condition
Disease
Observation