Liver Disease Diagnosis CYO Project

Daniel Haye

6/3/2020

Contents

1 Introduction

The issue of Liver disease has been plaguing India for a number of years. It was even suggested that the increase in liver disease patients may have stemmed from the fact that persons are now increasing their consumption of alcohol, inhale of harmful gases, intake of contaminated food, pickles, and drugs.

In general, the liver is evaluated with a group of test that all include: alanine transaminase, aspartate aminotransferase, alkaline phosphatase, albumin, total protein and bilirubin.

While taking a blood test, if any of thes chemicals (bilirubin, Alkaline Phosphotase, Alamine Aminotransferase) are above normal range, it may result in liver disease. Additionally if the albumin or the total protien level in your bload is below the normal range, it may also mean that patient may have a liver disease. The normal range for each chemical are as follows:

- Total Bilirubin: 0.3–1.0 mg/dL
- Direct Bilirubin: 0–0.4 mg/dL
- Alkaline Phosphotase: 20 to 140 IU/L
- Alamine Aminotransferase: male 29-33 IU/L, female 19-25 IU/L
- Aspartate Aminotransferase: male less than 50, female less than 45
- Total Protiens: 6-8.3 g/dL
- Albumin: 3.4-5.4 g/dL

1.1 Project Scope

This project is the final assignment in the HarvardX: PH125.9x Data Science: Capstone course. For this project, I had the option to choose any dataset to model a machine algorithm on. As such, I choose to use the Indian Liver Patient Records where the patient records were collected from North East of Andhra Pradesh, India.

This dataset will then be wrangled and an exploratory data analysis will be carried out in order to develop a machine learning algorithm. This algorithm will predict whether an individual is diagnosed with liver disease or not. After this, the model with the least error will be selected. Results will be explained followed by a conclussion with the closing remarks about the model.

1.2 Aim of the project

The aim of this project is to train machine learning models to predict whether a person has liver disease or not to try an reduce the workload on the Indian doctors. This will be done by assessing the chemical compounds (bilrubin, albumin, protiens, alkaline phosphatase) that are present in human body to diagnose the person. Data will be transformed and its dimension reduced to reveal patterns in the dataset and create a more robust analysis.

The model that optimizes the algorithm will be chosen following the resulting accuracy, sensitivity and f1 score. These metrics will be defined later down in this project.

The machine learning models that we would like for this project should provide a good mix between a high accuracy level combined with a high sensitivity level. Here, accuracy indicates the number of outcomes that was currectly predicted by the model while sensitivity (True Positive Rate) indicates what proportion of people 'with liver disease' were correctly classified. Additionally, F1 score can be seen as the harmonic mean of precision and sensitivity.

1.3 Dataset Summary

This data set contains 416 liver patient records and 167 non liver patient records collected from North East of Andhra Pradesh, India. The "Dataset" column is a class label used to divide groups into liver patient (liver disease) or not (no disease).

Source:www.kaggle.com

2 Knowing the Data

2.1 Data Exploration

To perform an analysis on a data, you must first know the dataset. As such, this section will give you a brief synapsis of the dataset.

This dataset consists of 583 observations with 11 variables. The collection of data spanned between 70 different ages where 441 were males and 142 were females. This shows that there is an imbalance between the number of males and females in the dataset. The top 5 ages represented in the data were: 60, 45, 50, 38 then 42.

Additionally, the variable dataset represents if the person is a liver disease patient or not. Here, 1 represent a patient with liver disease while 2 represent not having the disease. The dataset has only 4 NAs which are all in the "Albumin_and_Globulin_Ratio" variable.

pdata dataset

```
## Loading the dataset from my github profile

pdata <- read.csv("https://raw.githubusercontent.com/danielhaye17/Harvard-X-Capsole-Project/master/datas



#Finding the amount of observations and variables in the datset
dim(pdata)
```

```
## [1] 583  11
```

```
#Viewing the datatypes of the variables in the dataset
str(pdata)
```

```
## 'data.frame':    583 obs. of  11 variables:
##  $ Age                       : int  65 62 62 58 72 46 26 29 17 55 ...
##  $ Gender                    : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 2 1 1 2 2 ...
##  $ Total_Bilirubin           : num  0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.9 0.7 ...
##  $ Direct_Bilirubin          : num  0.1 5.5 4.1 0.4 2 0.7 0.2 0.3 0.3 0.2 ...
##  $ Alkaline_Phosphotase      : int  187 699 490 182 195 208 154 202 202 290 ...
##  $ Alamine_Aminotransferase  : int  16 64 60 14 27 19 16 14 22 53 ...
##  $ Aspartate_Aminotransferase: int  18 100 68 20 59 14 12 11 19 58 ...
##  $ Total_Protiens            : num  6.8 7.5 7 6.8 7.3 7.6 7 6.7 7.4 6.8 ...
##  $ Albumin                   : num  3.3 3.2 3.3 3.4 2.4 4.4 3.5 3.6 4.1 3.4 ...
##  $ Albumin_and_Globulin_Ratio: num  0.9 0.74 0.89 1 0.4 1.3 1 1.1 1.2 1 ...
##  $ Dataset                   : int  1 1 1 1 1 1 1 1 2 1 ...
```

```
#Summarizing the data
summary(pdata)
```

```
##       Age           Gender    Total_Bilirubin  Direct_Bilirubin
##  Min.   : 4.00   Female:142   Min.   : 0.400   Min.   : 0.100
##  1st Qu.:33.00   Male  :441   1st Qu.: 0.800   1st Qu.: 0.200
##  Median :45.00                Median : 1.000   Median : 0.300
##  Mean   :44.75                Mean   : 3.299   Mean   : 1.486
```

```
##  3rd Qu.:58.00                3rd Qu.: 2.600   3rd Qu.: 1.300
##  Max.   :90.00                Max.   :75.000   Max.   :19.700
##
##  Alkaline_Phosphotase Alamine_Aminotransferase Aspartate_Aminotransferase
##  Min.   :  63.0       Min.   :  10.00          Min.   :  10.0
##  1st Qu.: 175.5       1st Qu.:  23.00          1st Qu.:  25.0
##  Median : 208.0       Median :  35.00          Median :  42.0
##  Mean   : 290.6       Mean   :  80.71          Mean   : 109.9
##  3rd Qu.: 298.0       3rd Qu.:  60.50          3rd Qu.:  87.0
##  Max.   :2110.0       Max.   :2000.00          Max.   :4929.0
##
##  Total_Protiens     Albumin      Albumin_and_Globulin_Ratio    Dataset
##  Min.   :2.700   Min.   :0.900   Min.   :0.3000             Min.   :1.000
##  1st Qu.:5.800   1st Qu.:2.600   1st Qu.:0.7000             1st Qu.:1.000
##  Median :6.600   Median :3.100   Median :0.9300             Median :1.000
##  Mean   :6.483   Mean   :3.142   Mean   :0.9471             Mean   :1.286
##  3rd Qu.:7.200   3rd Qu.:3.800   3rd Qu.:1.1000             3rd Qu.:2.000
##  Max.   :9.600   Max.   :5.500   Max.   :2.8000             Max.   :2.000
##                                  NA's   :4
```

```
#Finding how many different ages are in the dataset
n_distinct(pdata$Age)
```

```
## [1] 72
```

```
#Finding the number of males and females in the dataset
pdata%>%group_by(Gender)%>%summarise(count=n())%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 10,
                 full_width = FALSE)
```

Gender	count
Female	142
Male	441

```
#Finding the top 5 ages represented in the dataset
pdata%>%group_by(Age)%>%summarise(count=n())%>%
  arrange(desc(count))%>%head(n=5)%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 10,
                 full_width = FALSE)
```

Age	count
60	34
45	25
50	23
38	21
42	21

```
#Checking to see the number ofliver disease patients
pdata%>%group_by(Dataset)%>%summarise(count=n())%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 10,
                 full_width = FALSE)
```

Dataset	count
1	416
2	167

```
#Finding out if there any data with N/As
sapply(pdata, function(g){
  sum(is.na(g))
})%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 10,
                 full_width = FALSE)
```

	x
Age	0
Gender	0
Total_Bilirubin	0
Direct_Bilirubin	0
Alkaline_Phosphotase	0
Alamine_Aminotransferase	0
Aspartate_Aminotransferase	0
Total_Protiens	0
Albumin	0
Albumin_and_Globulin_Ratio	4
Dataset	0

```
#Finding the mean of the Albumin_and_Globulin_Ratio variable
mean(pdata$Albumin_and_Globulin_Ratio,na.rm = TRUE)
```

```
## [1] 0.9470639
```

```
#Viewing the NAs in the Albumin_and_Globulin_Ratio variable
pdata%>%filter(is.na(Albumin_and_Globulin_Ratio))%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 8,
                 full_width = TRUE)
```

Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Dataset
45	Female	0.9	0.3	189	23	33	6.6	3.9	NA	1
51	Male	0.8	0.2	230	24	46	6.5	3.1	NA	1
35	Female	0.6	0.2	180	12	15	5.2	2.7	NA	2
27	Male	1.3	0.6	106	25	54	8.5	4.8	NA	2

In general, the pdata dataset is a data frame which consist of eleven variables:

1. **Age**: An integer datatype that contains the age of each person.
2. **Gender**: A factor with two levels, Male and Female
3. **Total_Bilirubin**: A number datatype that contains the total number of billirubins for each person
4. **Direct_Bilirubin**: A number datatype that contains the number of direct billirubins for each person
5. **Alkaline_Phosphotase**: An integer that contains the number of Alkaline Phosphotase for each person
6. **Alamine_Aminotransferase**: An integer that contains the number of Alamine Aminotransferase for each person
7. **Aspartate_Aminotransferase**: An integer that contains the number of Aspartate Aminotransferase for each person
8. **Total_Protiens**: A number datatype that contains the total number of protiens for each person
9. **Albumin**: A number datatype that contains the number of albumin for each person
10. **Albumin_and_Globulin_Ratio**: A number datatype that contains the Albumin and Globulin Ratio for each person
11. **Dataset**: An integer datatype which consist of 1 for liver patients and 2 for non-liver patients.

First 10 Rows of pdata dataset

Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Dataset
65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1
62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1
46	Male	1.8	0.7	208	19	14	7.6	4.4	1.30	1
26	Female	0.9	0.2	154	16	12	7.0	3.5	1.00	1
29	Female	0.9	0.3	202	14	11	6.7	3.6	1.10	1
17	Male	0.9	0.3	202	22	19	7.4	4.1	1.20	2
55	Male	0.7	0.2	290	53	58	6.8	3.4	1.00	1

2.2 Data Cleaning

Looking at the first six rows of the pdata data frame, the data needs additional work for the data to be classified as clean data. There are four NAs in the Albumin and Globulin ratio. Since it is a ration, we can change the NAs to the average of the ratio. To really work on the dataset, we would also need to make changes to the Dataset variable. This is because it would be easier to calculate if we use 0 to represent not being a liver patient instead of using 2. Fixing these three problems will result in the data becoming clean.

Therefore, for the pre-processing phase, the following steps will be taken:

1. Replace the NAs in the Albumin and Globulin ratio with 0s

```
#1.    Replace the NAs in the Albumin and Globulin ratio with 0s

       pdata$Albumin_and_Globulin_Ratio[is.na(pdata$Albumin_and_Globulin_Ratio)]<-
                   mean(pdata$Albumin_and_Globulin_Ratio,na.rm = TRUE)

       sapply(pdata, function(g){
         sum(is.na(g))
       })%>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                position = "center",
                font_size = 10,
                full_width = FALSE)
```

	x
Age	0
Gender	0
Total_Bilirubin	0
Direct_Bilirubin	0
Alkaline_Phosphotase	0
Alamine_Aminotransferase	0
Aspartate_Aminotransferase	0
Total_Protiens	0
Albumin	0
Albumin_and_Globulin_Ratio	0
Dataset	0

2. Replacing the 2s in the Dataset column with 0s

```
#2. Replacing the 2s in the Dataset column with 0s
       pdata<- pdata%>%mutate(Dataset= ifelse(Dataset ==2,0,1))
       pdata%>%group_by(Dataset)%>%summarise(count=n())%>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                position = "center",
                font_size = 8,
                full_width = TRUE)
```

Dataset	count
0	167
1	416

3. Create a column for Indirect Bilirubin

```
#3. Create a column for Indirect Bilirubin
  pdata<- pdata%>%mutate(Indirect_Bilirubin=
                                   Total_Bilirubin - Direct_Bilirubin )

    #Changing the order of the dataset
    pdata<- pdata%>%select(Age,Gender,Total_Bilirubin,
                                 Direct_Bilirubin,Indirect_Bilirubin,
                                 Alkaline_Phosphotase, Alamine_Aminotransferase,
                                 Aspartate_Aminotransferase,Total_Protiens,
                                 Albumin,Albumin_and_Globulin_Ratio, Dataset)
 pdata%>%head(n=3)%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 8,
                 full_width = TRUE)
```

Age	Gender	Total_Bilirubin	Direct_Bilirubin	Indirect_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Dataset
65	Female	0.7	0.1	0.6	187	16	18	6.8	3.3	0.90	1
62	Male	10.9	5.5	5.4	699	64	100	7.5	3.2	0.74	1
62	Male	7.3	4.1	3.2	490	60	68	7.0	3.3	0.89	1

4. Change Dataset variable name to Disease_Status

```
pdata<- pdata%>%mutate(Disease_Status=Dataset)%>%select(-Dataset)
pdata%>%head(n=3)%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 8,
                 full_width = TRUE)
```

Age	Gender	Total_Bilirubin	Direct_Bilirubin	Indirect_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Disease_Status
65	Female	0.7	0.1	0.6	187	16	18	6.8	3.3	0.90	1
62	Male	10.9	5.5	5.4	699	64	100	7.5	3.2	0.74	1
62	Male	7.3	4.1	3.2	490	60	68	7.0	3.3	0.89	1

First 10 rows in Cleaned pdata dataset

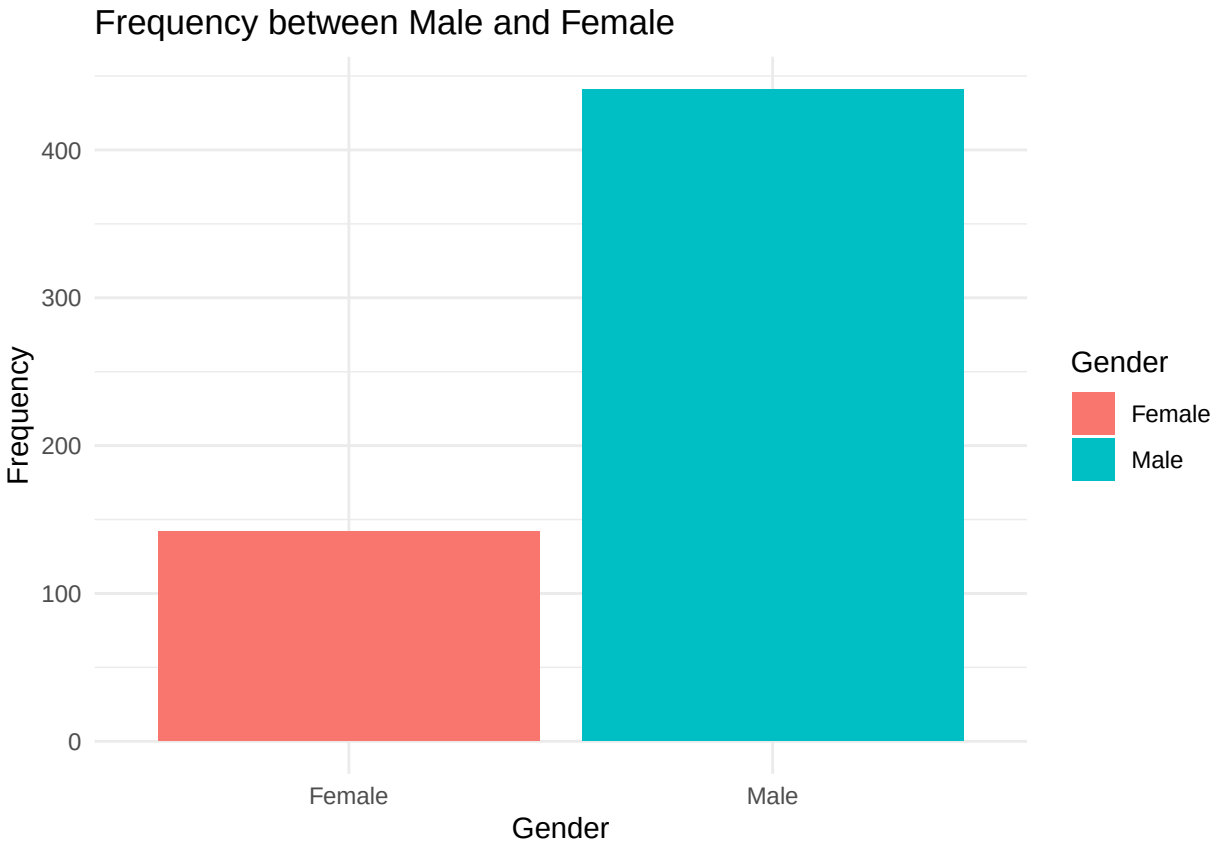
Age	Gender	Total_Bilirubin	Direct_Bilirubin	Indirect_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Disease_Status
65	Female	0.7	0.1	0.6	187	16	18	6.8	3.3	0.90	1
62	Male	10.9	5.5	5.4	699	64	100	7.5	3.2	0.74	1
62	Male	7.3	4.1	3.2	490	60	68	7.0	3.3	0.89	1
58	Male	1.0	0.4	0.6	182	14	20	6.8	3.4	1.00	1
72	Male	3.9	2.0	1.9	195	27	59	7.3	2.4	0.40	1
46	Male	1.8	0.7	1.1	208	19	14	7.6	4.4	1.30	1
26	Female	0.9	0.2	0.7	154	16	12	7.0	3.5	1.00	1
29	Female	0.9	0.3	0.6	202	14	11	6.7	3.6	1.10	1
17	Male	0.9	0.3	0.6	202	22	19	7.4	4.1	1.20	0
55	Male	0.7	0.2	0.5	290	53	58	6.8	3.4	1.00	1

2.3 Visualization

We will start off by taking a look at the level of disparity between males and females in the data.

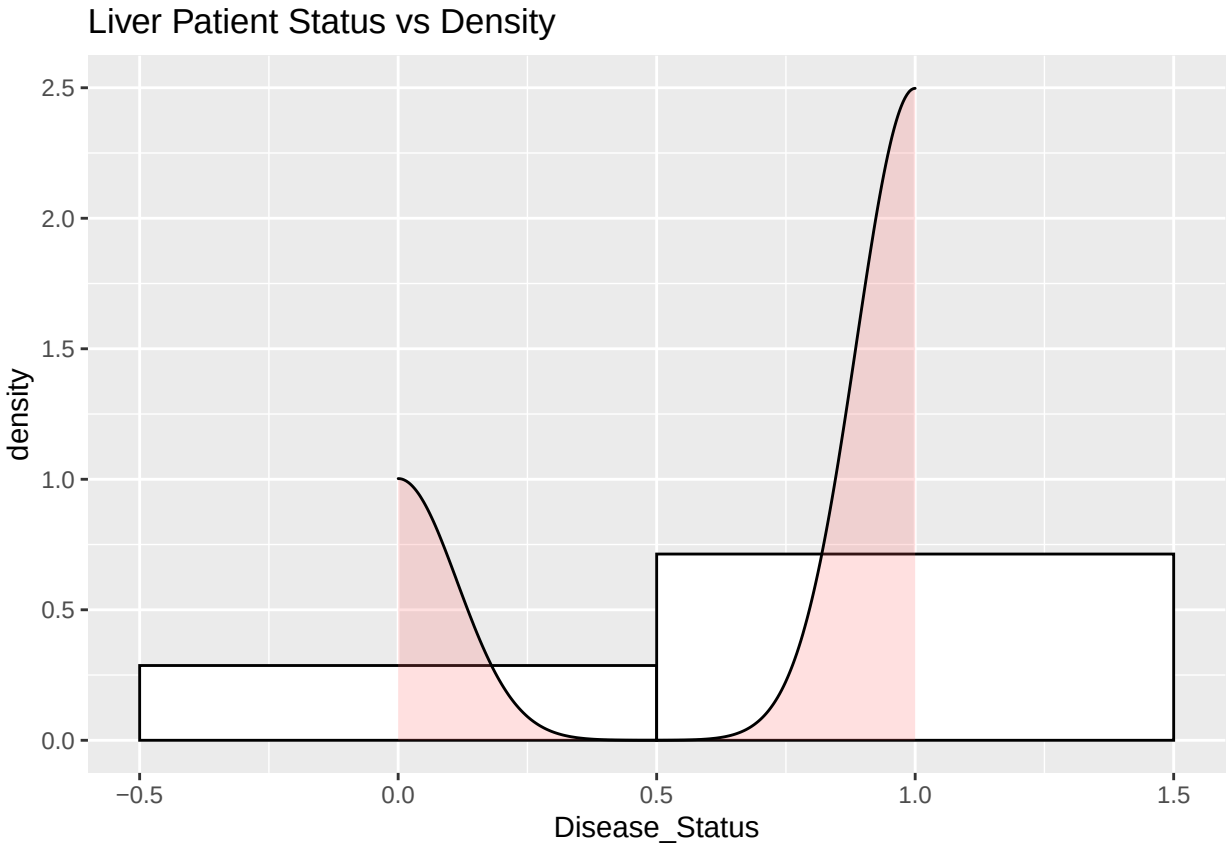
2.3.1 Frequency between Male and Female

This shows that is really a great level of disparity between the amount of males and females inside the dataset.



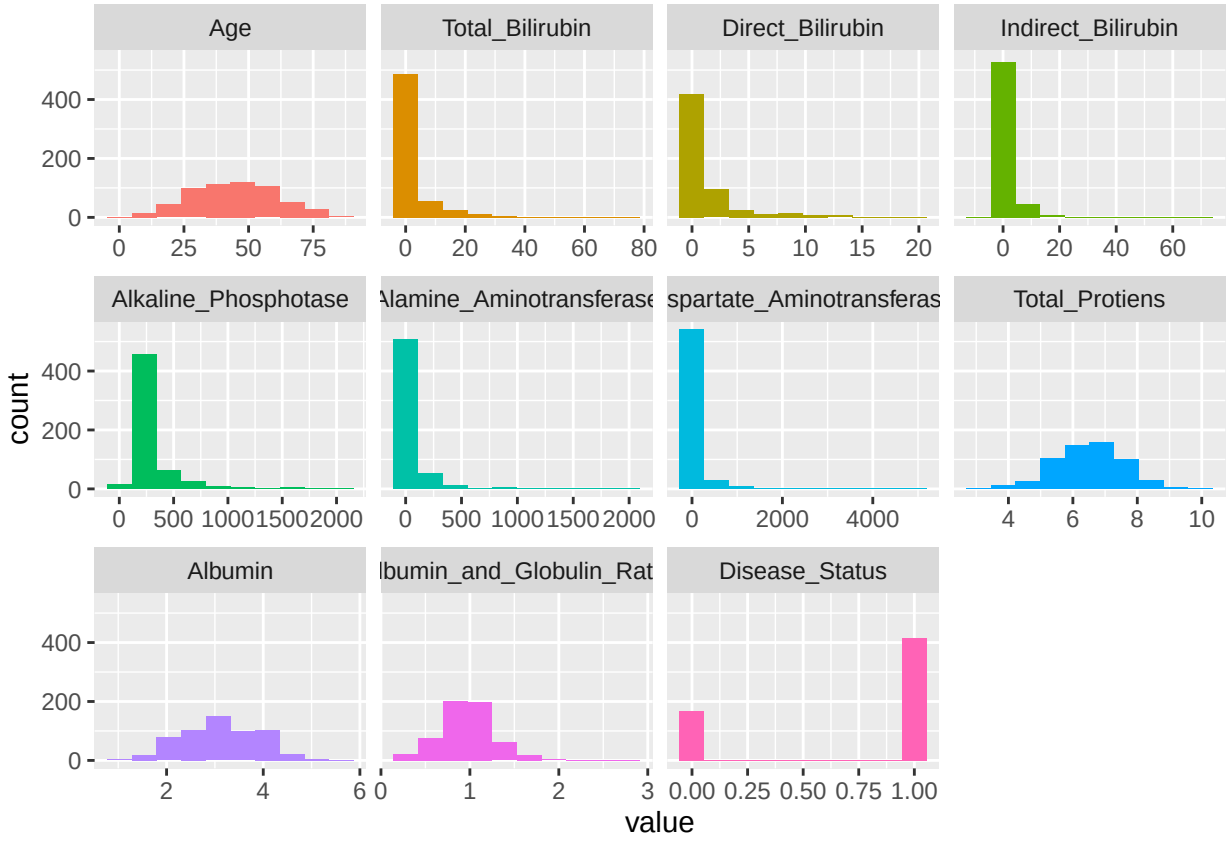
2.3.2 Liver Patient Status vs Density

This graph shows us that more than half of the persons in the dataset have liver disease.



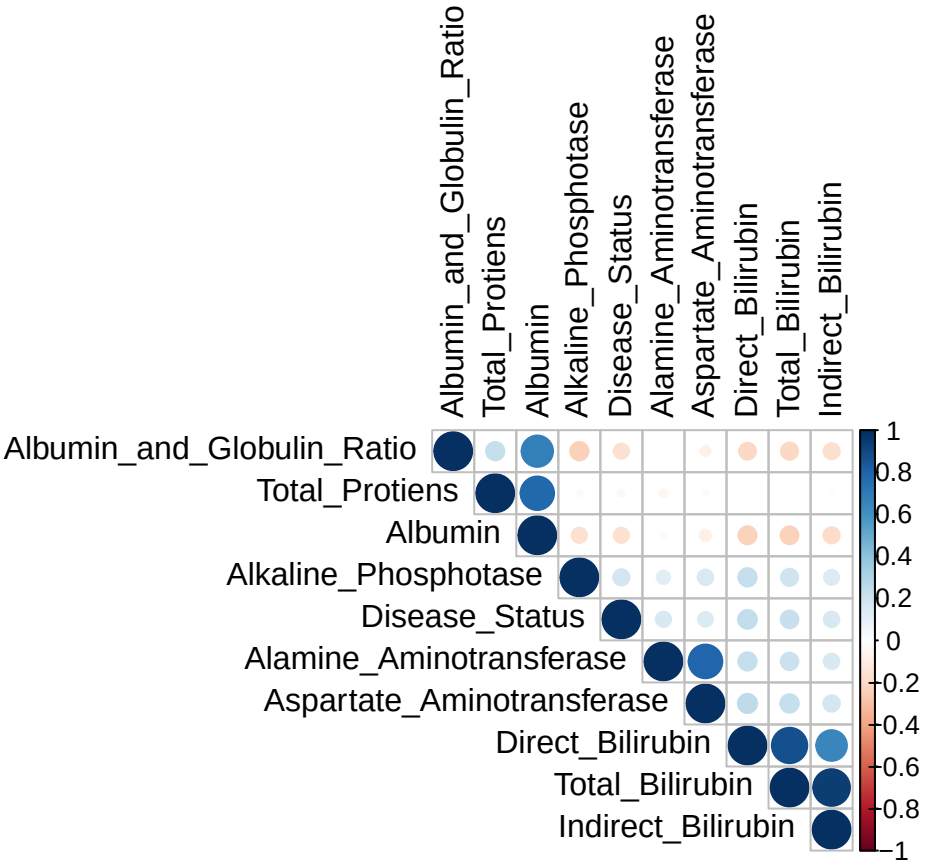
Gender	Num_Liver_Patients
Female	92
Male	324

2.3.3 Visualizing all the numeric data in the dataset



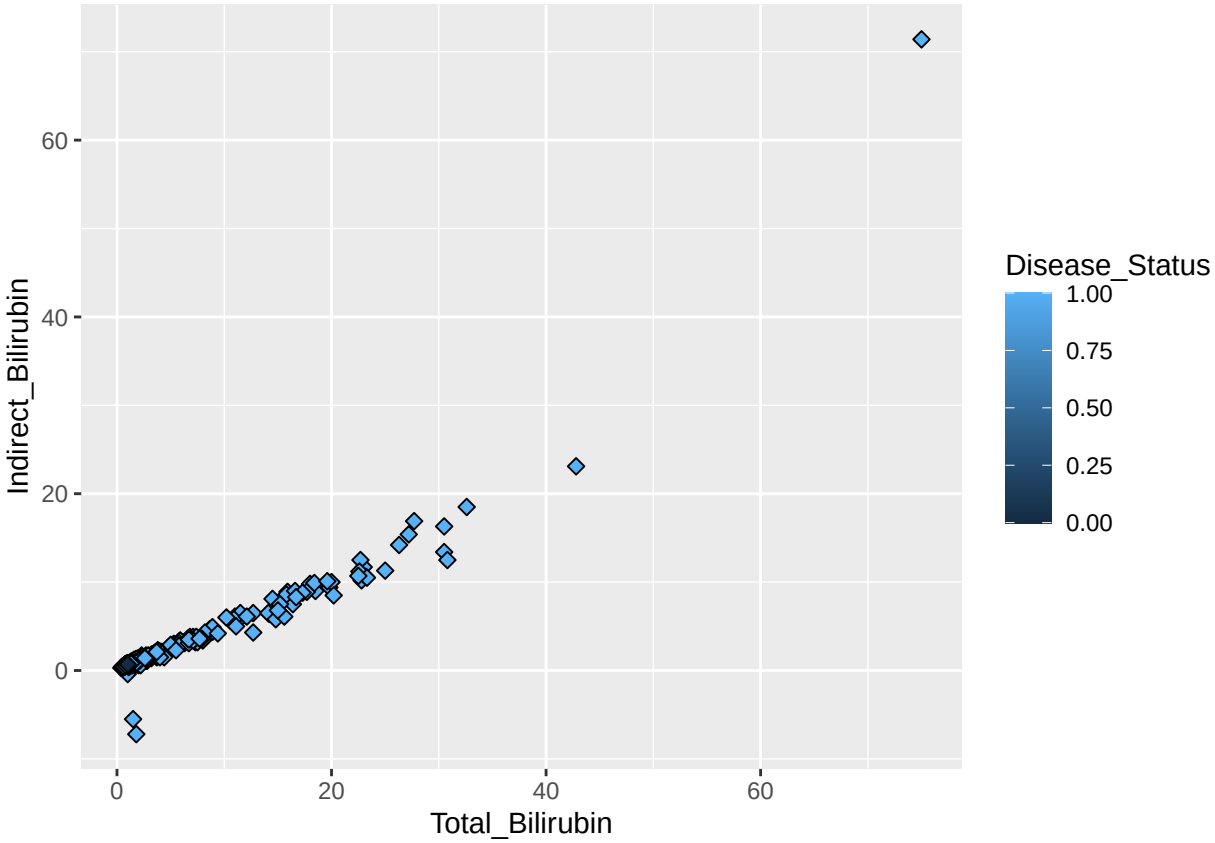
2.3.4 Checking the correlation between the chemicals

Machine learning algorithms suggest that all predictor variable shourl be independent of each other. From this correlation visualization one may suggest that there is not much correlation between the chemicals. However, there is some level of correlation between Total Bilirubin and Indirect Bilirubin.



2.3.5 Correlation between Total Bilirubin and Indirect Bilirubin

There is a high level of correlation between the two parameter. Therefore to get the best out of our model, we will have to remove one. In the Caret package, there is a function called *"findCorrelation"* that will determine which parameter to take out of our model. With a cutoff at 90%, the fuction determined that Total Bilirubin should be the parameter taken out.


Indirect_Bilirubin
60
40
20
0
0
20
40
60
Total_Bilirubin
Disease_Status
1.00
0.75
0.50
0.25
0.00

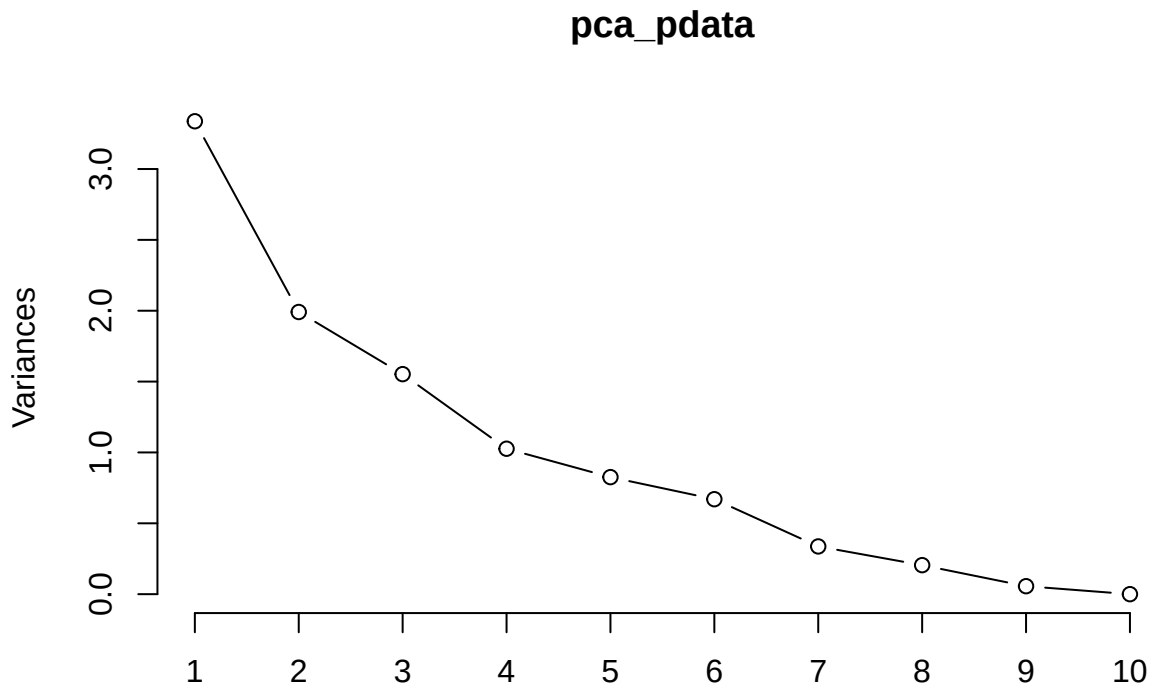
3 Modeling Approach

3.1 Principal Component Analysis (PCA)

This is a dimensionality-reduction technique that is used to explain the variance-covariance structure of a set of variables through linear combinations. The main reason for using the PCA is to reduce the amount of memory and computation power used since we would have less variables.

We will use the function *'prncomp'* to calculate the PCA so that we can prevent redundancy and relavancy.

```
#Calculating and plotting PCA
pca_pdata <- prcomp(pdata[,3:ncol(pdata)], center = TRUE, scale = TRUE)
        plot(pca_pdata, type="l")
```



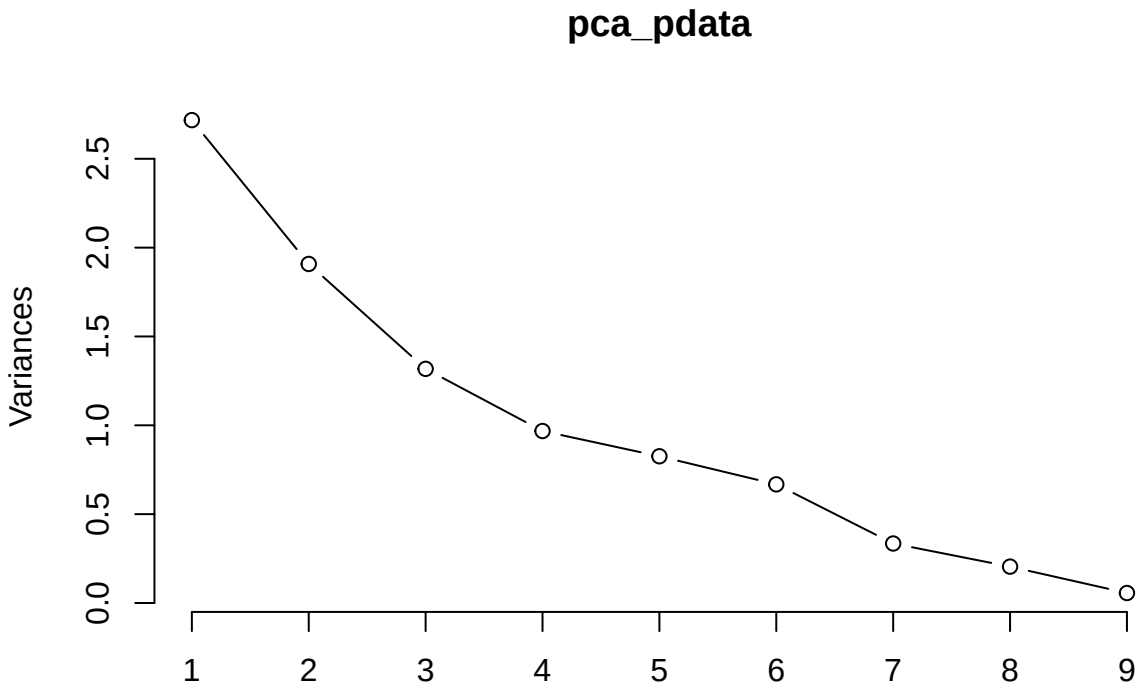

```
#Finding the level of Importance for each paramter
summary(pca_pdata)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5    PC6     PC7
## Standard deviation     1.8267 1.4110 1.2460 1.0130 0.90872 0.8185 0.58036
## Proportion of Variance 0.3337 0.1991 0.1552 0.1026 0.08258 0.0670 0.03368
## Cumulative Proportion  0.3337 0.5328 0.6881 0.7907 0.87324 0.9402 0.97392
##                           PC8     PC9      PC10
## Standard deviation     0.45237 0.23705 1.855e-15
## Proportion of Variance 0.02046 0.00562 0.000e+00
## Cumulative Proportion  0.99438 1.00000 1.000e+00
```

Now, let use the decision that we got from *findCorrelation* to optimize the variables needed for our model.

```
#Removing Total Biliribin from the model
        pdata<-pdata%>%select(-Total_Bilirubin)

    #Calculating and plotting PCA
        pca_pdata <- prcomp(pdata[,3:ncol(pdata)], center = TRUE, scale = TRUE)
        plot(pca_pdata, type="l")
```




```
    #Finding the level of Importance for each paramter
        summary(pca_pdata)
```

```
## Importance of components:
##                           PC1   PC2    PC3    PC4     PC5     PC6    PC7
## Standard deviation     1.6485 1.381 1.1479 0.9838 0.90866 0.81710 0.5786
## Proportion of Variance 0.3019 0.212 0.1464 0.1075 0.09174 0.07418 0.0372
## Cumulative Proportion  0.3019 0.514 0.6604 0.7679 0.85965 0.93383 0.9710
##                           PC8     PC9
## Standard deviation     0.45237 0.23692
## Proportion of Variance 0.02274 0.00624
## Cumulative Proportion  0.99376 1.00000
```

As such, to get the best model, we should in deep remove the Total Biliribin parameter.

3.2 Creation of Training and Test Set

Before we can start modelling our data, we first have to decide on which cross validation technique to choose. *Cross-validation* refers to a set of methods for measuring the performance of a given predictive model on new test data sets.

For this project, we have chosen the 10-fold cross-validation as our cross-validation method for assessing model performance. Ihave chosen this model because the dataset is not that huge, therefore the 10-fold cross-validation should be less bias than the other cross-validation methods. To use this method, however, we will set the seed to 1 for reproductivity.

We first have to split our data into a training and a test set. The the Caret package contains the fuction *"createDataPartition"* that will do the splitting for us. Here, we have chosen to split our data, with 20% going to the test set and 80% going to the training set. This ratio was chosen becase would like to have as much of our data to train while having our test set being large enough to obtain stable estimates of the loss.

Lets start by converting our outcome variable Disease_Status to a 2 factor variable

```
# Setting outcome as a 2 level factor variable which changing format to use ROC
pdata<- pdata%>%mutate(Disease_Status=
                                  ifelse(Disease_Status ==1,"yes","no"))
        pdata$Disease_Status <- as.factor(pdata$Disease_Status)
```

```
 # Split the dataset into train and test set
   # Set seed as a starting point
        set.seed(1, sample.kind='Rounding')

        train_index <- createDataPartition(
          y = pdata$Disease_Status,
          p = 0.2,
          list = F
        )
        training_set <- pdata[-train_index,]
        test_set <- pdata[train_index,]
```

```
#The dimension of the training set
        dim(training_set)
```

```
## [1] 465  11
```

```
#The dimension of the test set
        dim(test_set)
```

```
## [1] 118  11
```

After the splitting of the data we will create a function *"train.control"* to allow us set train controll as 10-fold cross-validation. Finally, we will then train our algorithm on the training set and use the test set to predict and compute our apparent errors.

```
# Define training control
      #method = cv -> Cross Validation method
      #number = 10 -> The number of folds
      # Set seed as a starting point
```

```
        set.seed(1, sample.kind='Rounding')
        train.control <- trainControl(method = "cv",
                                      number = 10,
                                      classProbs = TRUE,
                                      summaryFunction = twoClassSummary)
```

```
head(test_set)%>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                position = "center",
                font_size = 8,
                full_width = TRUE)
```

	Age	Gender	Direct_Bilirubin	Indirect_Bilirubin	Alkaline_Phosphotase	Aspartate_Transaminase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Disease_Status
5	72	Male	2.0	1.9	195	27	59	7.3	2.4	0.40	yes
11	57	Male	0.1	0.5	210	51	59	5.9	2.7	0.80	yes
13	64	Male	0.3	0.6	310	61	58	7.0	3.4	0.90	no
27	34	Male	2.0	2.1	289	875	731	5.0	2.7	1.10	yes
35	38	Female	1.2	1.4	410	59	57	5.6	3.0	0.80	no
39	48	Male	0.6	0.8	263	38	66	5.8	2.2	0.61	yes

```
head(training_set)%>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                position = "center",
                font_size = 8,
                full_width = TRUE)
```

	Age	Gender	Direct_Bilirubin	Indirect_Bilirubin	Alkaline_Phosphotase	Aspartate_Transaminase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Disease_Status
1	65	Female	0.1	0.6	187	16	18	6.8	3.3	0.90	yes
2	62	Male	5.5	5.4	699	64	100	7.5	3.2	0.74	yes
3	62	Male	4.1	3.2	490	60	68	7.0	3.3	0.89	yes
4	58	Male	0.4	0.6	182	14	20	6.8	3.4	1.00	yes
6	46	Male	0.7	1.1	208	19	14	7.6	4.4	1.30	yes
7	26	Female	0.2	0.7	154	16	12	7.0	3.5	1.00	yes

3.3 Logistic Regression Model (GLM)

We will start off our modeling with GLM. Firstly, Logical Regression is seen as one of the specified cases of the general linear model. This model takes categoric variables as its outcome, which is similar to what we have in our dataset, 1 for liver patient and 0 for non liver patient. As such we will use the function *"glm"* to fit our model. However in order to use this function, we would have to use the family function to specify the version of this model that we would like to use.

```
set.seed(1, sample.kind='Rounding')

        glm_fit<- train(Disease_Status ~., data = training_set , method = "glm",
                        metric = "ROC",
                        preProcess = c("scale", "center"),  # in order to normalize the data
                        trControl= train.control,
                        family = "binomial")
```

```
        # Predciting on the test data
        glm_pred<- predict( glm_fit, test_set)

        # Calculating the confusion matrix
        glm_conf_matrix <- confusionMatrix(glm_pred, test_set$Disease_Status, positive = "yes")

        #Storing the Accuracy and sensitivity level for glm model
         glm_Accuracy <- glm_conf_matrix$overall[['Accuracy']]
         glm_Sensitivity <- sensitivity(glm_pred, test_set$Disease_Status, positive="yes")


        #Calculating F1 Score

        precision <- posPredValue(glm_pred, test_set$Disease_Status, positive="yes")
        recall <- sensitivity(glm_pred, test_set$Disease_Status, positive="yes")

        glm_F1_Score <- (2 * precision * recall) / (precision + recall)


        # Creating a data frame to store the results from our models
        model_results <- data.frame(model="Logistic Regression Model",
                                    Accuracy =glm_Accuracy,
                                    Sensitivity = glm_Sensitivity ,
                                    F1_Score = glm_F1_Score)

        table("Accuracy" = glm_Accuracy, "Sensitivity" = glm_Sensitivity,
              "F1_Score" = glm_F1_Score  )%>%
              kable() %>%
              kable_styling(bootstrap_options = c("striped", "hover",
                                                  "condensed", "responsive"),
                                                position = "center",
                                                font_size = 10,
                                                full_width = FALSE)
```

Accuracy	Sensitivity	F1_Score	Freq
0.720338983050847	0.892857142857143	0.819672131147541	1

3.4 K-nearest neighbors (kNN) Model

This algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. As such, it relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data. However, it is easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.

```
set.seed(1, sample.kind='Rounding')

        knn_fit<- train(Disease_Status ~., data = training_set , method = "knn",
                        metric = "ROC",
                        preProcess = c("scale", "center"),  # in order to normalize the data
                        trControl= train.control,
                        tuneLength=10)
```

```
        # Predciting on the test data
        knn_pred<- predict( knn_fit, test_set)

        # Calculating the confusion matrix
        knn_conf_matrix <- confusionMatrix(knn_pred, test_set$Disease_Status, positive = "yes")

        #Storing the Accuracy and sensitivity level for knn model
        knn_Accuracy <- knn_conf_matrix$overall[['Accuracy']]
        knn_Sensitivity <- sensitivity(knn_pred, test_set$Disease_Status, positive="yes")


        #Calculating F1 Score
        precision <- posPredValue(knn_pred, test_set$Disease_Status, positive="yes")
        recall <- sensitivity(knn_pred, test_set$Disease_Status, positive="yes")

        knn_F1_Score <- (2 * precision * recall) / (precision + recall)

        #Adding the result to the data frame created earlier
        model_results <- model_results %>% add_row(model="K-Nearest Neighbors Model",
                                                   Accuracy =knn_Accuracy,
                                                   Sensitivity = knn_Sensitivity ,
                                                   F1_Score = knn_F1_Score)

        table("Accuracy" = knn_Accuracy, "Sensitivity" = knn_Sensitivity,
              "F1_Score" = knn_F1_Score  )%>%
              kable() %>%
              kable_styling(bootstrap_options = c("striped", "hover",
                                                  "condensed", "responsive"),
                                                position = "center",
                                                font_size = 10,
                                                full_width = FALSE)
```

Accuracy	Sensitivity	F1_Score	Freq
0.677966101694915	0.892857142857143	0.797872340425532	1

3.5 Rain Forrest Model

This is a renowed machine learning approach that solves the errors from the decision tree. Rain Forrest is an algorithm of constructing a decision tree (how to do splitting) when the dataset is so large that it does not fit the memory. In rain forest, the whole dataset is not required for making a splitting decision. Only some aggregated information (AVC-set for an attribute or AVC-group if you have more memory) is required.

```
set.seed(1, sample.kind='Rounding')

        rf_fit<- train(Disease_Status ~., data = training_set , method = "rf",
                       metric = "ROC",
                       preProcess = c("scale", "center"),  # in order to normalize the data
                       trControl= train.control,
                       tuneLength=10)
```

```
## note: only 9 unique complexity parameters in default grid. Truncating the grid to 9 .
```

```
        # Predciting on the test data
        rf_pred<- predict( rf_fit, test_set)

        # Calculating the confusion matrix
        rf_conf_matrix <- confusionMatrix(rf_pred, test_set$Disease_Status, positive = "yes")

        #Storing the Accuracy and sensitivity level for knn model
        rf_Accuracy <- rf_conf_matrix$overall[['Accuracy']]
        rf_Sensitivity <- sensitivity(rf_pred, test_set$Disease_Status, positive="yes")


        #Calculating F1 Score

        precision <- posPredValue(rf_pred, test_set$Disease_Status, positive="yes")
        recall <- sensitivity(rf_pred, test_set$Disease_Status, positive="yes")

        rf_F1_Score <- (2 * precision * recall) / (precision + recall)

        #Adding the result to the data frame created earlier
        model_results <- model_results %>% add_row(model="Random Forrest Model",
                                                   Accuracy =rf_Accuracy,
                                                   Sensitivity = rf_Sensitivity ,
                                                   F1_Score = rf_F1_Score)

        table("Accuracy" = rf_Accuracy, "Sensitivity" = rf_Sensitivity,
              "F1_Score" = rf_F1_Score  )%>%
              kable() %>%
              kable_styling(bootstrap_options = c("striped", "hover",
                                                  "condensed", "responsive"),
                                                position = "center",
                                                font_size = 10,
                                                full_width = FALSE)
```

Accuracy	Sensitivity	F1_Score	Freq
0.754237288135593	0.916666666666667	0.841530054644809	1

3.6 Support Vector Machines Algorithm

This is a supervised learning method that looks at data and sorts it into one of two categories. An SVM outputs a map of the sorted data with the margins between the two as far apart as possible. In fact, this machine algorithm can be used to analyzes data for classification and regression analysis.

```
set.seed(1, sample.kind='Rounding')

        svm_fit<- svm(Disease_Status~., data = training_set, kernel = "linear")
        svm_fit
```

```
##
## Call:
## svm(formula = Disease_Status ~ ., data = training_set, kernel = "linear")
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  291
```

```
        summary(svm_fit)
```

```
##
## Call:
## svm(formula = Disease_Status ~ ., data = training_set, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  291
##
##  ( 158 133 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
        svm_pred = predict(svm_fit,test_set)
        svm_conf_matrix <- confusionMatrix(reference =
                                    as.factor(test_set$Disease_Status), data = svm_pred)

        svm_Accuracy <- svm_conf_matrix$overall[['Accuracy']]
        svm_Sensitivity <- sensitivity(svm_pred, test_set$Disease_Status, positive="yes")


        #Calculating F1 Score

        precision <- posPredValue(svm_pred, test_set$Disease_Status, positive="yes")
        recall <- sensitivity(svm_pred, test_set$Disease_Status, positive="yes")

        svm_F1_Score <- (2 * precision * recall) / (precision + recall)

        #Adding the result to the data frame created earlier
        model_results <- model_results %>% add_row(model="Support Vector Model",
                                                   Accuracy =svm_Accuracy,
                                                   Sensitivity = svm_Sensitivity ,
                                                   F1_Score = svm_F1_Score)
        #Table of results
        table("Accuracy" = svm_Accuracy, "Sensitivity" = svm_Sensitivity,
              "F1_Score" = svm_F1_Score)%>%
```

```
            kable() %>%
            kable_styling(bootstrap_options = c("striped", "hover",
                                                "condensed", "responsive"),
                                              position = "center",
                                              font_size = 10,
                                              full_width = FALSE)
```

Accuracy	Sensitivity	F1_Score	Freq
0.711864406779661	1	0.831683168316832	1

4 Results

This is the summary results for all the model built, that were trained on trainig set dataset and validated on the test set dataset.

```
model_results%>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 10,
                 full_width = FALSE)
```

model	Accuracy	Sensitivity	F1_Score
Logistic Regression Model	0.7203390	0.8928571	0.8196721
K-Nearest Neighbors Model	0.6779661	0.8928571	0.7978723
Random Forrest Model	0.7542373	0.9166667	0.8415301
Support Vector Model	0.7118644	1.0000000	0.8316832

In order to evaluate how good the predictions made by that model are, we first have to look at three matrics of our models.

1. Accuracy - We will look at this aspect because it it is the measure of all the correctly identified cases. In order for the doctors to rely on this algorithm, it has to be trust worthy.

$$\text{Accuracy} = \frac{TruePositive + TrueNegative}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

2. Sensitivity - We will also look at sensitivity because it shows how many times the algorithm diagnosed a person with the liver disease that actually had the disease. Therefore the most important aspect is to make shore that the algorithm gives the correct verdict.

$$\text{Sensitivity} = \frac{TruePositive}{TruePositive + FalseNegative}$$

3. F1 Score - The last matric we will look at is the F1 score. This is the harmonic mean of Precision and Recall and gives a better measure of the incorrectly classified cases than the Accuracy Metric.Here precision is the measure of the correctly identified positive cases from all the predicted positive cases while Recall is the measure of the correctly identified positive cases from all the actual positive cases.

$$\text{Recall} = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$\text{Precision} = \frac{TruePositive}{TruePositive + FalsePositive}$$

$$\text{F1 Score} = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

Accuracy is used when the True Positives and True negatives are more important while F1-score is used when the False Negatives and False Positives are crucial.

Accuracy can be used when the class distribution is similar while F1-score is a better metric when there are imbalanced classes as in the above case.

Accuracy can be used when the class distribution is similar while F1-score is a better metric when there are imbalanced classes as in the above case.

However, the it is quite easy to select the model that best optimizes the model, since the rain forrest model had the highest result for both Accuracy and F1 Score. In addition, approximately 92% percent of the persons with liver disease were correctly classified.

Even though the support vector had the 100% percent of the persons with liver disease were correctly classified, it still wasnot as precise and accurate as the rain forrest model.

Both the Logistic Regression model and the K-Nearest Neighbors Model had the same level of sensitivity. However, the Logical Regression Model had the higher accuracy and F1 score.

Therefore, it is fair to say that the worst model was the K-Nearest Neighbors Model while the best model was the Logistic Regression Model.

5 Conclusion

For this project, we investigated several machine learning models and we selected the optimal model by observing the accuracy, sensitivity along with the F1 Score of each model. The verdict of our model was that the worst model was the K-Nearest Neighbors Model while the best model was the Logistic Regression Model.