STAT 652 PROJECT: FLIGHT DELAY PREDICTION

ANUJ SABOO
301398628

1. Introduction

This is an analysis report on data from nycflights13 package which has information on flights from three New York airports in 2013. The dataset consists of 200,000 observations across 43 variables and a separate test set consisting of 136,776 observations which was released later.
The data consists of information about flights departure, observed weather, airport locations and flight details.

The **goal** is to predict if a flight would be delayed or not. In the pathway to accomplish this goal I would demonstrate various tasks such as handling missing values, exploratory data analysis and fitting various models to predict flight delay.

2. Data

Let us first look further into the dataset and gain knowledge on the various variables present. The data is combined from four datasets in the nycflights13 packages. We get information on the following aspects:

- \`flights\`: Data for all flights that departed NYC (i.e. JFK, LGA or EWR) in 2013
- \`weather\`: Hourly meteorological data for LGA, JFK and EWR
- \`airports\`: Information about the location of airports
- \`planes\` : Flight technical details

2.1 Data Cleaning

On conducting an initial analysis of the data, it was found that the data is missing for many variables. There are numerous possibilities to handle the scenario such as imputation or omitting rows. However, a threshold of > 5% missing data was chosen to eliminate the variable from the model. Therefore, I removed the variables year.y, type, manufacturer, model, engines, seats, speed, engine, wind_gust and pressure.

For the missing values that remain, I chose to omit these rows from my analysis. The training dataset now consist of 184316 observations for 33 variables.

2.2 Variable Selection

The variables listed below are removed from the analysis due to the following observations:

- year.x remains the same throughout our dataset
- dep_time, arr_time, arr_delay and sched_arr_time are already captured by other variables

- tailnum has 4000 planes and flight(flight number) is not relevant for us
- name is already captured by dest
- air_time has high correlation with distance, hour and minute in sched_dep_time
- time_hour can be obtained from sched_dep_time
- tz, dst, tzone are the time zone related information of the destination

2.3 Data Manipulation

2.3.1 Target Variable

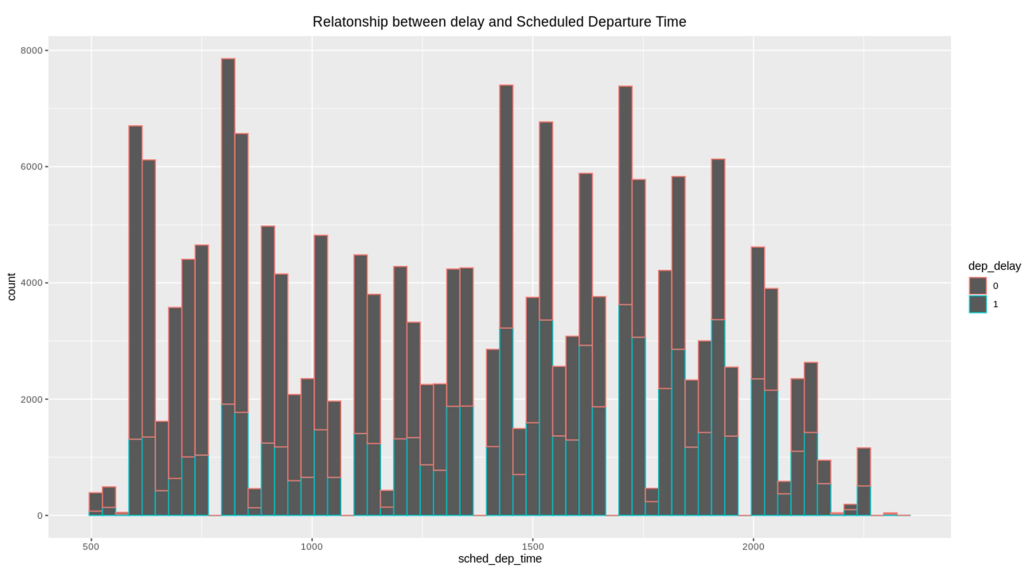
Since the aim is to predict if a flight will be delayed or not, we introduce a binary response variable encoded as 0 for no delay and 1 for delay. We generate this new variable from dep_delay column which denotes the minutes by which the flight gets delayed or departs early.

2.3.2 Data Wrangling

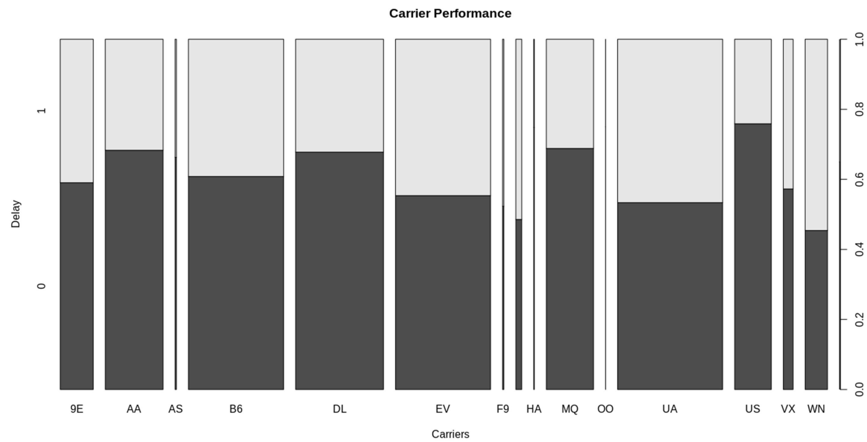
- The variable precip is replaced as a binary indicator of precipitation
- Variables year/month/day are converted to a date object

2.4 Exploratory Data Analysis

We observe that there are only a few early morning flights and they usually are on time. However, as the day progresses, due to increasing number of flights, the flights that get delayed increase with it being the highest in the afternoon.



The flight data is for 16 carriers, the below plot shows that United Air Lines Inc. has the highest number of flights. The highest proportion of on time flights are observed for US Airways Inc.



2.5 Validation Data

In order to test the model accuracy and tune parameters we split the training data into 2/3 training and 1/3 validation. This is an important step required for tuning various parameters by measuring model accuracy on unseen observations to get the appropriate fit.

3. Methods

I used the training data to fit various statistical approaches on the dataset to predict the binary response variable for flight delay. Then, the models are used to predict on the validation set to measure Model Accuracy on unseen observations.

The final model chosen for the prediction was **XGBoost**. The below models were considered

Logistic Regression: I predicted a binary dependent variable dep_delay using the predictors and generated probability prediction for the validation data to classify it as delay or not. The accuracy rate was lower than the other models tested later.

Gradient Boosting: It produces a prediction model in the form of an ensemble of weak prediction models. It gave slightly better results compared to the above. I faced issues in trying to tune the parameters such as n.trees and shrinkage due to hardware limitations.

Random Forest and SVM: Attempts to run these models failed due to limitation of compute power. Hence, we will not be discussing them in our analysis.

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. The algorithm works on building many weak classifiers which together act as a strong classifier along with having superior computational efficiency compared to the algorithms discussed above.

The algorithm provides a large number of parameters for tuning. I took the task to tune the following parameters(code available on Page 16, Section 3.2 of appendix):

- max.depth: Maximum depth of a tree
- eta: Control the learning rate between 0 to 1. Lower value means model is more robust to overfit
- nrounds: Max number of boosting iterations
- We do not obtain an overfit model and hence don't consider regularization parameters for tuning

After trying a few models with varying values I ran tests with max.depth <- c(6,7,8,9), eta <- c(0.1,0.3,0.5) and nrounds <-c(100,150,200).
The optimal value for **max.depth of the tree is found to be 8**(setting it to 9 only gave a very marginal increase). The **eta value** which controls how much information from a new tree will be used for boosting was found to be **0.1**. The **number of rounds being 150** gave me the best results.

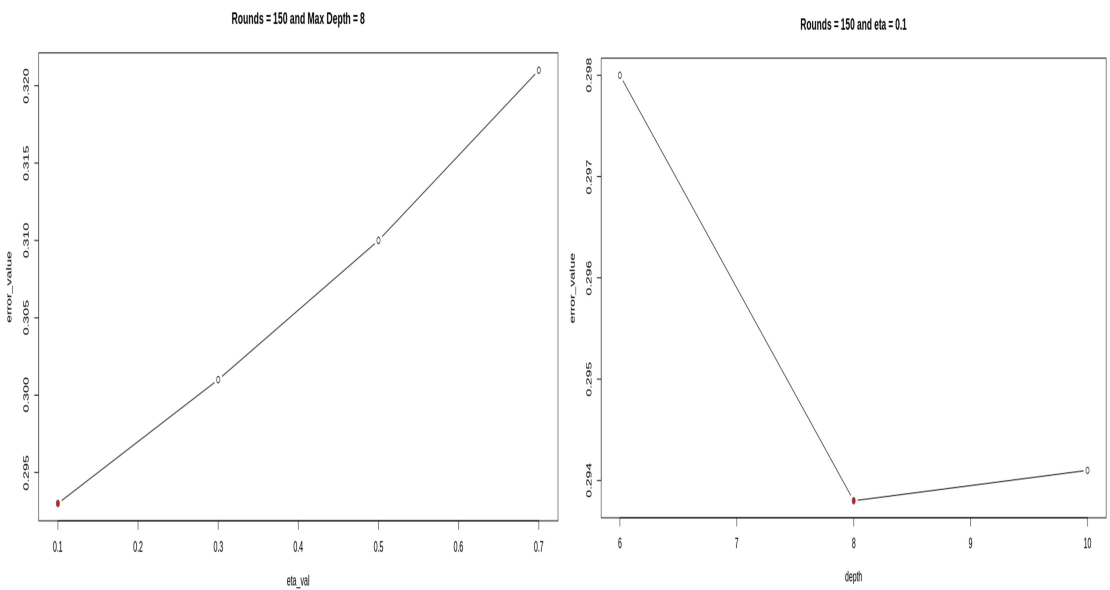


Figure: Error with respect to varying values of eta and max.depth

4. Results

4.1 Model Performance

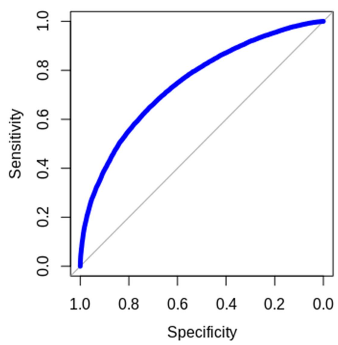
We predicted the departure delay on validation set for the below models. Choosing XGBoost with its tuned parameters, we find the accuracy on test data as **71.09 %**.

Algorithm	Accuracy on Validation Data	Accuracy on Test Data
XGBoost	70.61%	71.09%
Gradient Boosting	66.23%	-
Logistic Regression	64.35%	-

The code to evaluate the model on test data can be found on Page 24, Section 3.4 of the Appendix.The confusion matrix obtained on test data through XGBoost is given below:

Predicted vs Actual	0	1
0	65605	24935
1	11387	24107

The ROC Curve generated using predicted response as [0, 1] shows a sharp elbow bend. Instead when probability is used to plot the ROC curve, a smoother fit is obtained due to the increase in number of points being plotted on the curve.



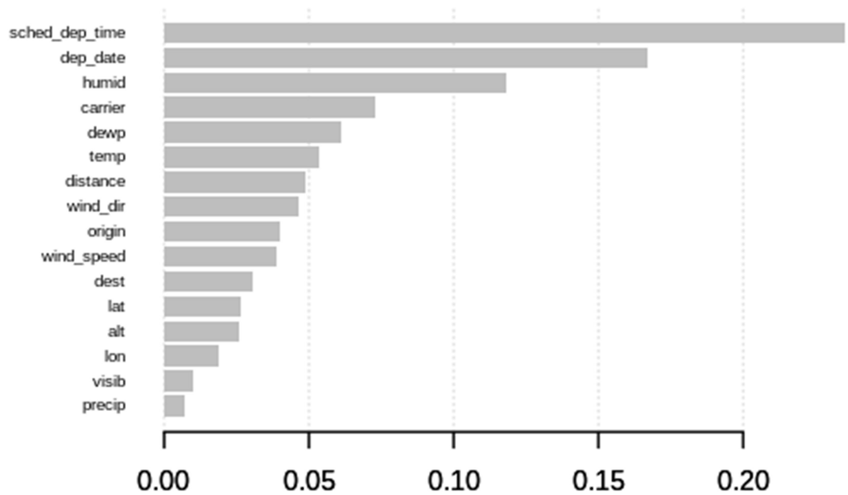

```
Call:
roc.default(response = fl_te_xg$dep_delay, predictor = rpred.xgb,
direction = ">")

Data: rpred.xgb in 37324 controls (fl_te_xg$dep_delay 0) > 24114
cases (fl_te_xg$dep_delay 1).
Area under the curve: 0.2519
```

Figure: ROC Curve

4.2 Result Inference

The variable importance plot below shows the importance of the predictors used to fit the model. As it can be observed, the most important variable that impacts flight delay is **sched_dep_time**. Other variables such as dep_date, career and weather related attributes also play a role in determining the flight delay. Surprisingly weather conditions such as Visibility and Precipitation don’t seem to contribute much in predicting flight delays.



5. Conclusions and Discussions

I attained a model accuracy of 71.09 % and found that scheduled departure time, departure date, humidity and carrier are the main factors contributing to departure delay. The XGBoost algorithm outperformed the others in terms of both accuracy and computational efficiency.

5.1 Short-Comings and Future Work

Computation power posed a difficult challenge for me to get the best outcome. I would like to work in the below direction for improving the model accuracy:

1. A lot of predictors from our analysis were removed due to missing data. There are a variety of ways to handle missing values and maybe imputing the missing values as mean or mode would lead to better results.
2. Due to limited computation power, I could not further tune or test other models such as Random Forest and Support Vector Machines.

Appendix

1. Software & Package Details

R version: 3.6.1

The following packages have been used

- `flights`
- `tidyverse`
- `lubridate`
- `gbm`
- `xgboost`
- `pROC`
- `caret`

2. Data Processing

I start with looking into the dataset and gain knowledge on the various variables present. The data is combined from four datasets in the nycflights13 packages. The information is from the following aspects:

- `flights` : Data for all flights that departed NYC (i.e. JFK, LGA or EWR) in 2013
- `weather` : Hourly meterological data for LGA, JFK and EWR
- `airports` : Information about location of airports
- `planes` : Technical Details of Flight

```
suppressMessages(library(tidyverse))
suppressMessages(library(lubridate))
suppressMessages(library(xgboost))
suppressMessages(library(pROC))
suppressMessages(library(caret))
file <- 'fltrain.csv.gz'
fltrain <- read_csv(file)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   carrier = col_character(),
##   tailnum = col_character(),
##   origin = col_character(),
##   dest = col_character(),
##   time_hour = col_datetime(format = ""),
##   name = col_character(),
##   dst = col_character(),
##   tzone = col_character(),
##   type = col_character(),
##   manufacturer = col_character(),
##   model = col_character(),
##   engine = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
summary(fltrain)
```

```
##      year.x         month            day          dep_time    sched_dep_tim
e
##  Min.   :2013   Min.   : 1.000   Min.   : 1.0   Min.   :   1   Min.   : 106
##  1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.0   1st Qu.: 907   1st Qu.: 905
##  Median :2013   Median : 7.000   Median :16.0   Median :1401   Median :1359
##  Mean   :2013   Mean   : 6.553   Mean   :15.7   Mean   :1349   Mean   :1344
##  3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.0   3rd Qu.:1745   3rd Qu.:1729
##  Max.   :2013   Max.   :12.000   Max.   :31.0   Max.   :2400   Max.   :2359
##                                                 NA's   :4898
##    dep_delay         arr_time    sched_arr_time   arr_delay
##  Min.   : -43.0   Min.   :   1   Min.   :   1   Min.   : -79.000
##  1st Qu.:  -5.0   1st Qu.:1104   1st Qu.:1124   1st Qu.: -17.000
##  Median :  -2.0   Median :1535   Median :1557   Median :  -5.000
##  Mean   :  12.7   Mean   :1502   Mean   :1537   Mean   :   6.969
##  3rd Qu.:  11.0   3rd Qu.:1941   3rd Qu.:1945   3rd Qu.:  14.000
##  Max.   :1301.0   Max.   :2400   Max.   :2359   Max.   :1272.000
##  NA's   :4898     NA's   :5169                  NA's   :5584
##    carrier              flight       tailnum            origin
##  Length:200000      Min.   :   1   Length:200000      Length:200000
##  Class :character   1st Qu.: 561   Class :character   Class :character
##  Mode  :character   Median :1499   Mode  :character   Mode  :character
##                     Mean   :1975
##                     3rd Qu.:3470
##                     Max.   :8500
##
##      dest              air_time        distance         hour
##  Length:200000      Min.   : 20.0   Min.   :  17   Min.   : 1.00
##  Class :character   1st Qu.: 82.0   1st Qu.: 502   1st Qu.: 9.00
##  Mode  :character   Median :129.0   Median : 872   Median :13.00
##                     Mean   :150.5   Mean   :1038   Mean   :13.18
##                     3rd Qu.:191.0   3rd Qu.:1389   3rd Qu.:17.00
##                     Max.   :695.0   Max.   :4983   Max.   :23.00
##                     NA's   :5584
##      minute        time_hour                        temp             dewp
##  Min.   : 0.00   Min.   :2013-01-01 10:00:00   Min.   : 10.94   Min.   :-9.9
4
##  1st Qu.: 8.00   1st Qu.:2013-04-04 20:00:00   1st Qu.: 42.08   1st Qu.:26.0
6
##  Median :29.00   Median :2013-07-03 15:00:00   Median : 57.20   Median :42.8
0
##  Mean   :26.22   Mean   :2013-07-03 12:05:05   Mean   : 56.98   Mean   :41.6
2
##  3rd Qu.:44.00   3rd Qu.:2013-10-01 12:00:00   3rd Qu.: 71.96   3rd Qu.:57.9
2
##  Max.   :59.00   Max.   :2014-01-01 04:00:00   Max.   :100.04   Max.   :78.0
8
```

```
##                                                  NA's   :948      NA's   :948
##      humid          wind_dir       wind_speed       wind_gust
##  Min.   : 12.74   Min.   :  0.0   Min.   : 0.000   Min.   :16.11
##  1st Qu.: 43.99   1st Qu.:130.0   1st Qu.: 6.905   1st Qu.:20.71
##  Median : 57.69   Median :220.0   Median :10.357   Median :24.17
##  Mean   : 59.57   Mean   :201.5   Mean   :11.107   Mean   :25.21
##  3rd Qu.: 75.33   3rd Qu.:290.0   3rd Qu.:14.960   3rd Qu.:28.77
##  Max.   :100.00   Max.   :360.0   Max.   :42.579   Max.   :66.75
##  NA's   :948      NA's   :5862    NA's   :982      NA's   :152260
##      precip          pressure         visib             name
##  Min.   :0.0000   Min.   : 985    Min.   : 0.000   Length:200000
##  1st Qu.:0.0000   1st Qu.:1013    1st Qu.:10.000   Class :character
##  Median :0.0000   Median :1018    Median :10.000   Mode  :character
##  Mean   :0.0045   Mean   :1018    Mean   : 9.252
##  3rd Qu.:0.0000   3rd Qu.:1023    3rd Qu.:10.000
##  Max.   :1.2100   Max.   :1042    Max.   :10.000
##  NA's   :937      NA's   :23092   NA's   :937
##       lat             lon               alt               tz
##  Min.   :21.32   Min.   :-157.92   Min.   :   3.0   Min.   :-10.000
##  1st Qu.:32.90   1st Qu.: -95.28   1st Qu.:  26.0   1st Qu.: -6.000
##  Median :36.10   Median : -83.35   Median : 433.0   Median : -5.000
##  Mean   :36.02   Mean   : -89.44   Mean   : 582.5   Mean   : -5.748
##  3rd Qu.:41.41   3rd Qu.: -80.15   3rd Qu.: 748.0   3rd Qu.: -5.000
##  Max.   :61.17   Max.   : -68.83   Max.   :6602.0   Max.   : -5.000
##  NA's   :4484    NA's   :4484      NA's   :4484     NA's   :4484
##      dst               tzone               year.y         type
##  Length:200000      Length:200000      Min.   :1956    Length:200000
##  Class :character   Class :character   1st Qu.:1999    Class :character
##  Mode  :character   Mode  :character   Median :2002    Mode  :character
##                                        Mean   :2001
##                                        3rd Qu.:2006
##                                        Max.   :2013
##                                        NA's   :34298
##  manufacturer          model              engines         seats
##  Length:200000      Length:200000      Min.   :1.000   Min.   :  2.0
##  Class :character   Class :character   1st Qu.:2.000   1st Qu.: 55.0
##  Mode  :character   Mode  :character   Median :2.000   Median :149.0
##                                        Mean   :1.994   Mean   :136.6
##                                        3rd Qu.:2.000   3rd Qu.:189.0
##                                        Max.   :4.000   Max.   :450.0
##                                        NA's   :31163   NA's   :31163
##      speed           engine
##  Min.   : 90.0    Length:200000
##  1st Qu.:105.0    Class :character
##  Median :112.0    Mode  :character
##  Mean   :150.8
##  3rd Qu.:127.0
```

```
## Max.   :432.0
## NA's   :199415
```

We can see above the statistics summary for the data consisting of numerous character and numerical variables. At this point we can go ahead and convert the character variables to factors.

```
fltrain <- fltrain
for(i in 1:ncol(fltrain)) {
  if(typeof(fltrain[[i]]) == "character") {
    fltrain[[i]] <- factor(fltrain[[i]])
  }
}
```

2.1 Missing Data

To begin with my analysis, I found that the data is missing for many variabes. We can choose to handle missing values in various ways such as imputation or omitting rows. We first find the amount of data missing in each variable. To do so, I apply the below functon to get the % of missing data in each column.

```
perc_missing <- function(x)
{
  sum(is.na(x))/length(x)*100
}
apply(fltrain,2,perc_missing)
```

```
##         year.x          month            day       dep_time sched_dep_time
##         0.0000         0.0000         0.0000         2.4490         0.0000
##      dep_delay       arr_time sched_arr_time      arr_delay        carrier
##         2.4490         2.5845         0.0000         2.7920         0.0000
##         flight        tailnum         origin           dest       air_time
##         0.0000         0.7460         0.0000         0.0000         2.7920
##       distance           hour         minute      time_hour           temp
##         0.0000         0.0000         0.0000         0.0000         0.4740
##           dewp          humid       wind_dir     wind_speed      wind_gust
##         0.4740         0.4740         2.9310         0.4910        76.1300
##         precip       pressure          visib           name            lat
##         0.4685        11.5460         0.4685         2.2420         2.2420
##            lon            alt             tz            dst          tzone
##         2.2420         2.2420         2.2420         2.2420         2.2420
##         year.y           type   manufacturer          model        engines
##        17.1490        15.5815        15.5815        15.5815        15.5815
##          seats          speed         engine
##        15.5815        99.7075        15.5815
```

I chose a threshold of > 5% missing data to remove the variable. As we can find missing data for year.y, type, manufacturer, model, engines, seats, speed, engine, wind_gust and pressure exceeding the 5% threshold, these variables will be discarded from the analysis.

```
fltrain <- fltrain %>% select(-year.y,-type,-manufacturer,-model,-engines,-seat
s, -speed, -engine,-wind_gust,-pressure)
```

For the missing values that remain, we choose to omit these rows from our analysis. Hence, we will be left with 184316 observations across 33 variables.

```
fltrain <- na.omit(fltrain)
```

We further make some keen observations about other variables and choose to omit them.

- year.x remains the same throughout our dataset
- dep_time, arr_time, arr_delay and sched_arr_time are already captured by other variables
- tailnum has 4000 planes and flight(flight number) is not relevant for us
- name is already captured by dest
- air_time has high correlation with distance, hour and minute in sched_dep_time
- time_hour can be obtained from sched_dep_time
- tz, dst, tzone are the time zone related information of the destination

- variables year/month/day are converted to a date object.
- replaced numeric precip with indicator of precipitation/none

```
fltrain <- fltrain %>% mutate(dep_date = make_date(year.x,month,day)) %>% selec
t(-year.x,-month,-day,-dep_time,-arr_time,-arr_delay,
         -sched_arr_time,-tailnum,-flight,-name,-air_time,
         -hour,-minute,-time_hour,-tz,-dst,-tzone) %>%
mutate(precip = as.numeric(precip>0))
```

2.2 Target Variable

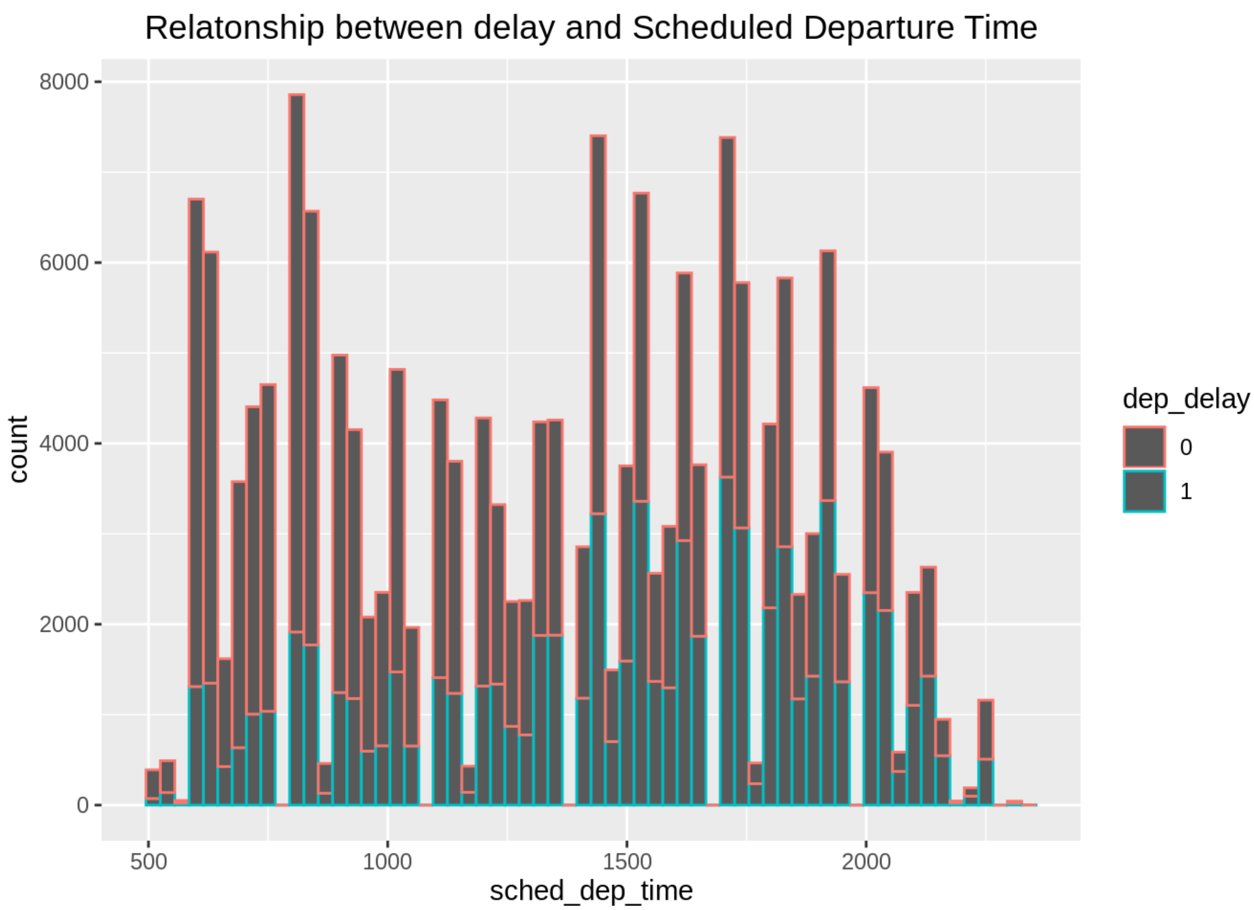
Since our aim is to predict if a flight will delay or not, a binary response variable is introduced and encoded as 0 for no delay and 1 for delay. This new variable is generated from dep_delay column which denotes the minutes by which the flight gets delayed or departs early.

```
fltrain$dep_delay = ifelse(fltrain$dep_delay<=0,0,1)
fltrain$dep_delay = as.factor(fltrain$dep_delay)
```

2.3 Exploratory Data Analysis

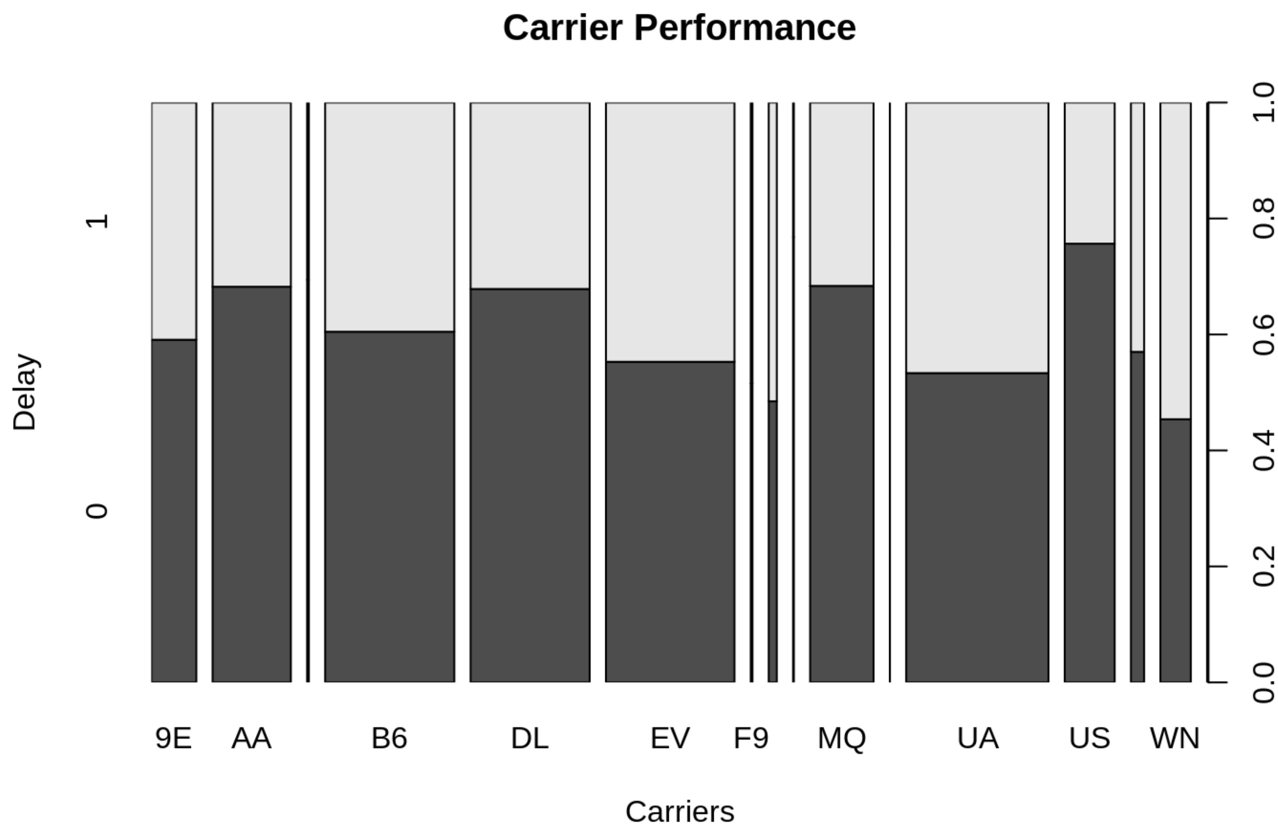
In order to understand the relationship of the target variable with the predictors, I begin exploratory data analysis. Key findings are:

```
ggplot(fltrain) + geom_histogram(mapping = aes(x=sched_dep_time,colour=dep_dela
y),binwidth = 30) + ggtitle("Relatonship between delay and Scheduled Departure
Time") + theme(plot.title = element_text(hjust = 0.5))
```



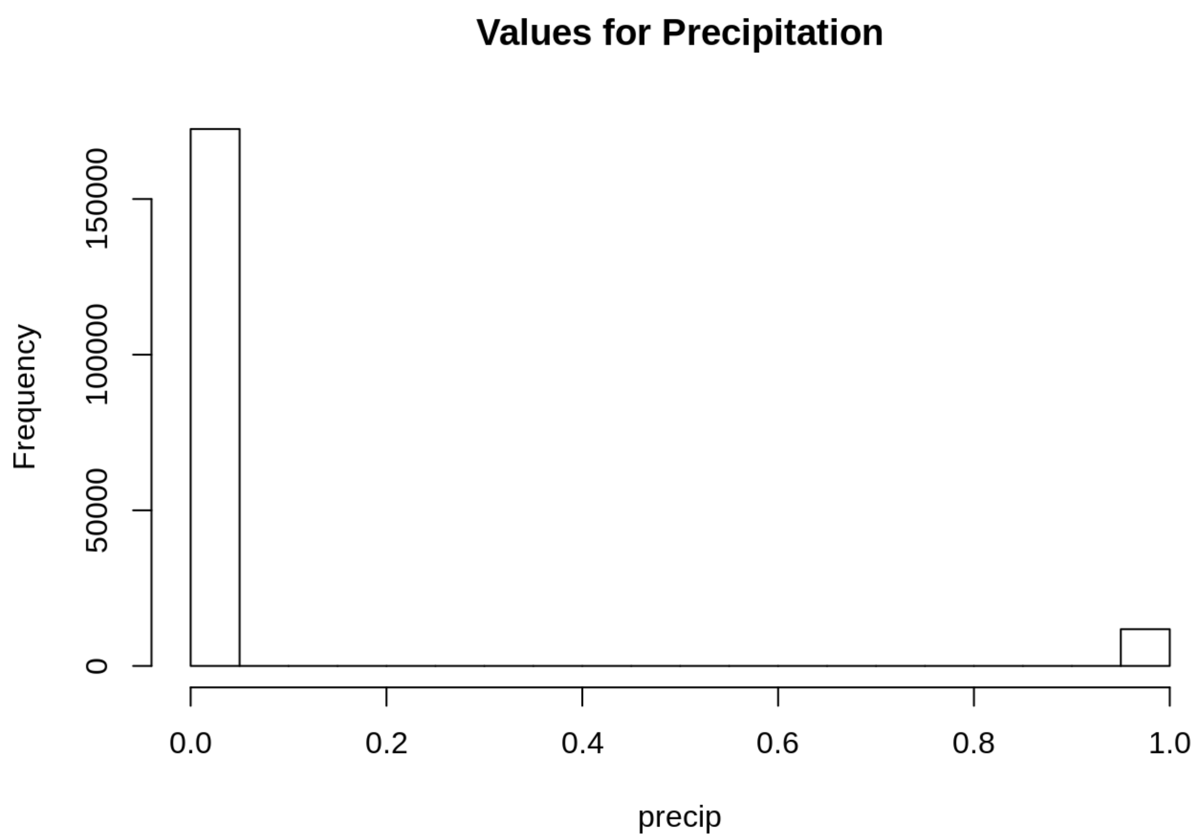
We observe that there are only a few early morning flights and they usually are on time. However, as the day progresses, due to rush hour the flights that get delayed increase with it being highest in the afternoon.

```
plot(fltrain$carrier,fltrain$dep_delay,xlab="Carriers",ylab="Delay",main="Carri
er Performance")
```



The flight data is for 16 carriers, the below plot shows that United Air Lines Inc. has the highest number of flights. The highest proportion of on time flights are observed for US Airways Inc.

```
hist(fltrain$precip,main="Values for Precipitation",xlab="precip")
```



The variable precip has 93% of values as 0. Though this would generally imply low variability, but after discussion with professor Brad, it was understood that due to the dataset size the 7% variable data is approx. 14,000 rows and hence should be taken into account for analysis.

2.4 Validation Data

Now we split our dataset into training and testing.

```
set.seed(123)
tr_size <- ceiling(2*nrow(fltrain)/3)
train <- sample(1:nrow(fltrain),size=tr_size)
fl_tr <- fltrain[train,]
fl_te <- fltrain[-train,]
```

3. Methods

I found XGBoost algorithm to give me the best performance. The code for other algorithms like Logistic Regression and Gradient Boosting is presented in Section 4.

3.1 Data Processing

The first task is to convert the dep_delay variable to 0 and 1 since XGBoost expects the response label to be in [0,1] for classification. This is being done below for both training and validation dataset.

```
fl_tr_xg <- fl_tr
fl_te_xg <- fl_te
fl_tr_xg$dep_delay <- (as.numeric(fl_tr_xg$dep_delay))-1
fl_te_xg$dep_delay <- (as.numeric(fl_te_xg$dep_delay))-1
```

Next, XGBoost expects data to be in the form of dgCMatrix. Hence, we need to convert the training and validated data in the expected data format.

```
traindata_xgb <- xgb.DMatrix(label=fl_tr_xg$dep_delay,data=data.matrix(fl_tr_xg
[-2]))
validationdata_xgb <- xgb.DMatrix(label=fl_te_xg$dep_delay,data=data.matrix(fl_
te_xg[-2]))
```

3.2 Parameter Tuning

To evaluate the best parameters for the model I use the below function to generate a matrix with validation errors using different values of max_depth, eta and nrounds.

```
paratab <- matrix(nrow = 36, ncol = 4)
i  <- 1
ndepth <-c(6,7,8,9)
netas <-c(0.1,0.3,0.5)
nrounds <-c(100,150,200)

for(d in ndepth){
  for(e in netas){
    for(r in nrounds){
      model.xgboost <- xgboost(data = traindata_xgb, max.depth = d, eta = e, nt
hread = 7,  nrounds = r, objective = "binary:logistic")
      pred.xgb <- predict(model.xgboost, validationdata_xgb)
      pred.xgb <- as.numeric(pred.xgb > 0.5)
      err <- mean(as.numeric(pred.xgb > 0.5) != fl_te_xg$dep_delay)
      paratab[i,1] <- d
      paratab[i,2] <- e
      paratab[i,3] <- r
      paratab[i,4] <- err
      i <- i+1
    }
  }
}
```

I found through the information generated above that the best parameters are max.depth = 8, eta = 0.1

and nrounds=150. Using them, let us go ahead and fit the model.

```
model.xgboost <- xgboost(data = traindata_xgb, max.depth = 8, eta = 0.1, nthrea
d = 7, nrounds = 150, objective = "binary:logistic",seed = 1)
```

```
## [1]  train-error:0.314320
## [2]  train-error:0.310617
## [3]  train-error:0.308306
## [4]  train-error:0.305620
## [5]  train-error:0.303984
## [6]  train-error:0.302520
## [7]  train-error:0.302031
## [8]  train-error:0.300558
## [9]  train-error:0.298743
## [10] train-error:0.298247
## [11] train-error:0.297694
## [12] train-error:0.296294
## [13] train-error:0.295806
## [14] train-error:0.294560
## [15] train-error:0.293771
## [16] train-error:0.292778
## [17] train-error:0.291639
## [18] train-error:0.290272
## [19] train-error:0.289222
## [20] train-error:0.288180
## [21] train-error:0.287147
## [22] train-error:0.286455
## [23] train-error:0.285478
## [24] train-error:0.284135
## [25] train-error:0.283134
## [26] train-error:0.281873
## [27] train-error:0.281010
## [28] train-error:0.280441
## [29] train-error:0.279025
## [30] train-error:0.278398
## [31] train-error:0.277723
## [32] train-error:0.277291
## [33] train-error:0.276730
## [34] train-error:0.276095
## [35] train-error:0.275794
## [36] train-error:0.274874
## [37] train-error:0.274573
## [38] train-error:0.273906
## [39] train-error:0.272807
## [40] train-error:0.272050
## [41] train-error:0.271546
## [42] train-error:0.271147
## [43] train-error:0.270284
## [44] train-error:0.269519
## [45] train-error:0.269080
## [46] train-error:0.268551
```

```
## [47] train-error:0.268063
## [48] train-error:0.267403
## [49] train-error:0.266435
## [50] train-error:0.265817
## [51] train-error:0.265320
## [52] train-error:0.264254
## [53] train-error:0.263522
## [54] train-error:0.263098
## [55] train-error:0.262732
## [56] train-error:0.262228
## [57] train-error:0.262130
## [58] train-error:0.261837
## [59] train-error:0.261267
## [60] train-error:0.260453
## [61] train-error:0.260160
## [62] train-error:0.259754
## [63] train-error:0.259412
## [64] train-error:0.258932
## [65] train-error:0.258736
## [66] train-error:0.258150
## [67] train-error:0.257939
## [68] train-error:0.257695
## [69] train-error:0.257434
## [70] train-error:0.257345
## [71] train-error:0.257206
## [72] train-error:0.256694
## [73] train-error:0.256067
## [74] train-error:0.255880
## [75] train-error:0.255579
## [76] train-error:0.254985
## [77] train-error:0.254537
## [78] train-error:0.254293
## [79] train-error:0.253772
## [80] train-error:0.253438
## [81] train-error:0.253007
## [82] train-error:0.252755
## [83] train-error:0.252218
## [84] train-error:0.251648
## [85] train-error:0.251388
## [86] train-error:0.251363
## [87] train-error:0.250989
## [88] train-error:0.250818
## [89] train-error:0.250566
## [90] train-error:0.250346
## [91] train-error:0.249760
## [92] train-error:0.249386
## [93] train-error:0.249117
```

```
## [94] train-error:0.248775
## [95] train-error:0.248002
## [96] train-error:0.247473
## [97] train-error:0.247302
## [98] train-error:0.246879
## [99] train-error:0.246708
## [100]    train-error:0.245788
## [101]    train-error:0.245390
## [102]    train-error:0.244763
## [103]    train-error:0.244511
## [104]    train-error:0.243787
## [105]    train-error:0.243477
## [106]    train-error:0.242981
## [107]    train-error:0.242794
## [108]    train-error:0.242655
## [109]    train-error:0.242460
## [110]    train-error:0.242240
## [111]    train-error:0.241907
## [112]    train-error:0.241264
## [113]    train-error:0.240881
## [114]    train-error:0.240661
## [115]    train-error:0.240499
## [116]    train-error:0.240027
## [117]    train-error:0.239880
## [118]    train-error:0.239693
## [119]    train-error:0.239270
## [120]    train-error:0.238692
## [121]    train-error:0.238269
## [122]    train-error:0.237764
## [123]    train-error:0.237626
## [124]    train-error:0.237089
## [125]    train-error:0.236731
## [126]    train-error:0.236356
## [127]    train-error:0.236169
## [128]    train-error:0.236153
## [129]    train-error:0.235600
## [130]    train-error:0.235429
## [131]    train-error:0.235250
## [132]    train-error:0.235079
## [133]    train-error:0.234712
## [134]    train-error:0.234444
## [135]    train-error:0.234338
## [136]    train-error:0.234045
## [137]    train-error:0.233891
## [138]    train-error:0.233817
## [139]    train-error:0.233508
## [140]    train-error:0.233191
```

```
## [141]    train-error:0.233199
## [142]    train-error:0.232930
## [143]    train-error:0.232857
## [144]    train-error:0.232540
## [145]    train-error:0.232076
## [146]    train-error:0.231653
## [147]    train-error:0.231246
## [148]    train-error:0.230904
## [149]    train-error:0.230896
## [150]    train-error:0.230350
```

3.3 Evaluate Accuracy on Validation Data

Next, I use the fitted model to predict on the validation set and measure the accuracy of the model.

```
rpred.xgb <- predict(model.xgboost, validationdata_xgb)
pred.xgb <- as.numeric(rpred.xgb > 0.5)
err <- mean(pred.xgb == fl_te_xg$dep_delay)
print(paste("Accuracy on Validation Data=", err))
```

```
## [1] "Accuracy on Validation Data= 0.70617533122823"
```

```
ct <- table(Predicted = pred.xgb,Actual = fl_te_xg$dep_delay)
se <- sensitivity(ct)
sp <- specificity(ct)
print(paste("The Sensitivty obtained is ",se))
```

```
## [1] "The Sensitivty obtained is  0.85122173400493"
```

```
print(paste("The Specificity obtained is ",sp))
```

```
## [1] "The Specificity obtained is  0.481670398938376"
```

The area under the ROC Curve is presented below along with a plot of the curve.

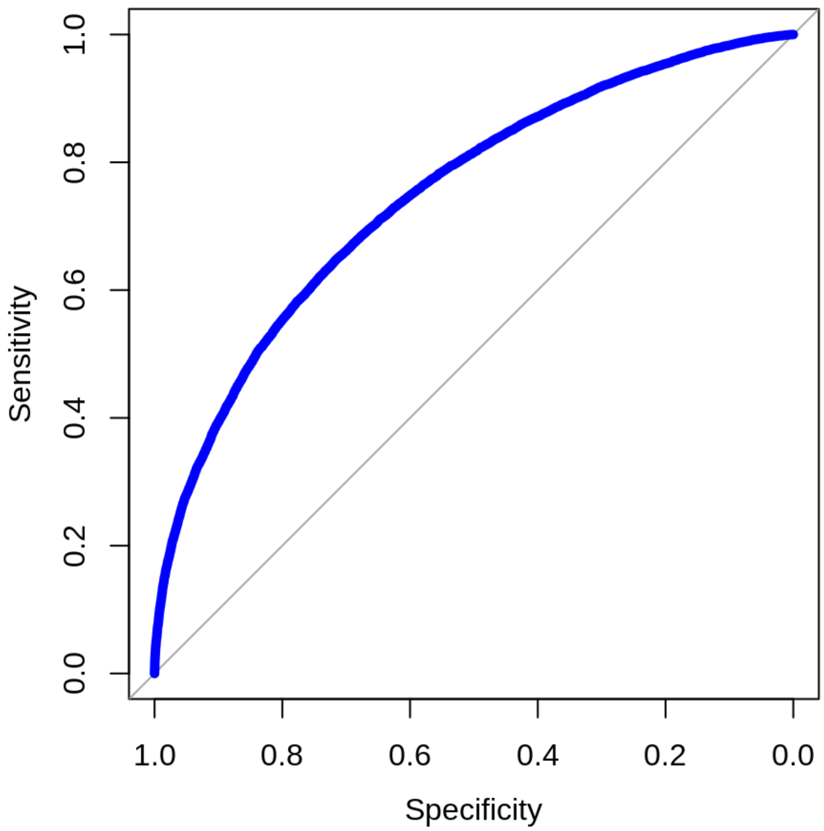
```
roc.curve <- roc(fl_te_xg$dep_delay,rpred.xgb,direction="<")
```

```
## Setting levels: control = 0, case = 1
```

```
print(roc.curve)
```

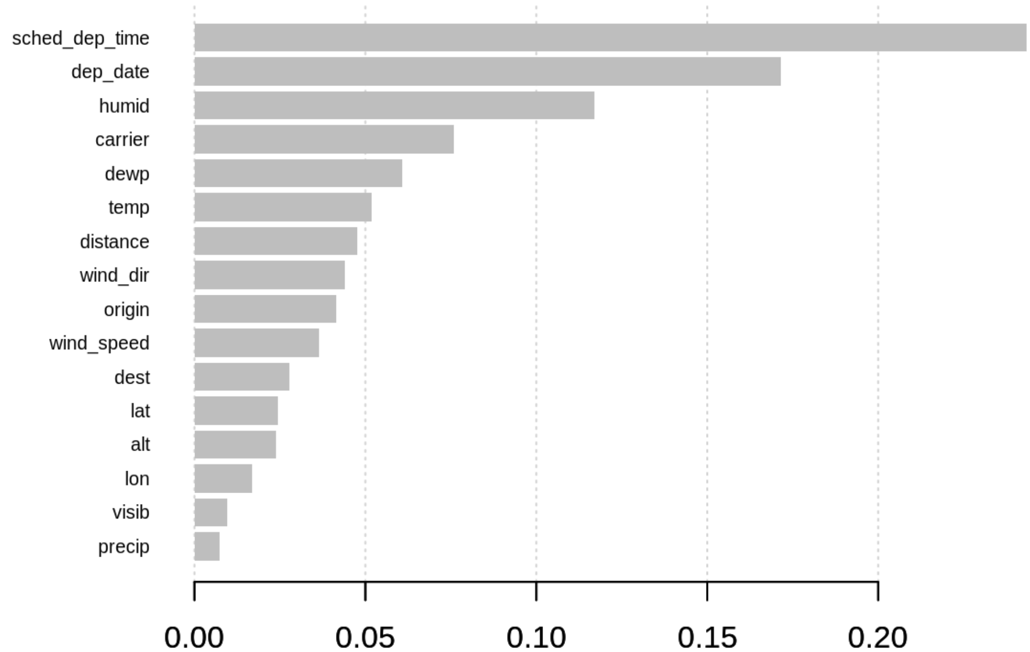
```
##
## Call:
## roc.default(response = fl_te_xg$dep_delay, predictor = rpred.xgb,     direct
ion = "<")
##
## Data: rpred.xgb in 37324 controls (fl_te_xg$dep_delay 0) < 24114 cases (fl_t
e_xg$dep_delay 1).
## Area under the curve: 0.7481
```

```
par(pty="s")
plot(roc.curve,col="blue",lwd=5)
```



The variable importance is generated using the below script.

```
importance_matrix <- xgb.importance(model = model.xgboost)
xgb.plot.importance(importance_matrix = importance_matrix)
```



3.4 Evaluate Accuracy on Test Data

At last, I use the test data to make the final prediction. The test data is imported and transformed into the desired format by applying the same operations done on the training data.

```
testfile <- 'fltest.csv.gz'
fltest <- read_csv(testfile)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   carrier = col_character(),
##   tailnum = col_character(),
##   origin = col_character(),
##   dest = col_character(),
##   time_hour = col_datetime(format = ""),
##   name = col_character(),
##   dst = col_character(),
##   tzone = col_character(),
##   type = col_character(),
##   manufacturer = col_character(),
##   model = col_character(),
##   engine = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
fltest <- fltest
for(i in 1:ncol(fltest)) {
  if(typeof(fltest[[i]]) == "character") {
    fltest[[i]] <- factor(fltest[[i]])
  }
}

fltest <- fltest %>% select(-year.y,-type,-manufacturer,-model,-engines,-seats,
-speed, -engine,-wind_gust,-pressure)
fltest <- na.omit(fltest)

fltest <- fltest %>% mutate(dep_date = make_date(year.x,month,day)) %>% select
(-year.x,-month,-day,-dep_time,-arr_time,-arr_delay,
         -sched_arr_time,-tailnum,-flight,-name,-air_time,
         -hour,-minute,-time_hour,-tz,-dst,-tzone) %>%
mutate(precip = as.numeric(precip>0))

fltest$dep_delay = ifelse(fltest$dep_delay<=0,0,1)
fltest$dep_delay = as.factor(fltest$dep_delay)
predvalue = fltest$dep_delay
fltest$dep_delay <- (as.numeric(fltest$dep_delay))-1
testdata <- xgb.DMatrix(label=fltest$dep_delay,data=data.matrix(fltest[-2]))
```

Now that the data is in the desired format, the trained model is used to make the prediction on the test set.

```
rpred.xgb <- predict(model.xgboost, testdata)
pred.xgb <- as.numeric(rpred.xgb > 0.5)
table(Predicted = pred.xgb,Actual = predvalue)
```

```
##          Actual
## Predicted     0     1
##         0 65714 25151
##         1 11278 23891
```

The confusion matrix is displayed above.

```
err <- mean(pred.xgb == predvalue)
print(paste("Accuracy on Test Data=", err))
```

```
## [1] "Accuracy on Test Data= 0.710958947585572"
```

4. Other Methods

Below are other methods that I tried but got a lower accuracy and could not tune the parameters due to computation issues.

4.1 Logistic Regression

```
logreg.model <- glm(dep_delay ~ .,data = fl_tr,family=binomial)
summary(logreg.model)
logred.predict = predict(logreg.model,newdata = fl_te,type="response")
var_d = rep(0,61438)
var_d[logred.predict > 0.5]=1
table(Predicted = var_d,Actual = fl_te$dep_delay)
mean (var_d == fl_te$dep_delay)
```

4.2 Gradient Boosting

```
library(gbm)
dep_date_numeric <- as.numeric(fl_tr$dep_date)
dep_date_numeric <- dep_date_numeric - mean(dep_date_numeric)
fl_tr_tem <- mutate(fl_tr,dep_date = dep_date_numeric)
fl_tr_tem$dep_delay = (as.numeric(fl_tr_tem$dep_delay) - 1)
gbm_fit <-gbm(dep_delay ~ .,data=fl_tr_tem,distribution="bernoulli",n.trees = 2
000, shrinkage = 0.2)

dep_date_numeric <- as.numeric(fl_te$dep_date)
dep_date_numeric <- dep_date_numeric - mean(dep_date_numeric)
fl_te_tem <- mutate(fl_te,dep_date = dep_date_numeric)
fl_te_tem$dep_delay = (as.numeric(fl_te_tem$dep_delay) - 1)
pred.boost = predict(gbm_fit,newdata=fl_te_tem,n.trees=2000,type="response")
var_d = rep(0,61438)
var_d[pred.boost > 0.5]=1
table(Predicted = var_d,Actual = fl_te_tem$dep_delay)
mean (var_d == fl_te_tem$dep_delay)
```