## 1. Source of Dataset

The dataset is based on an airline passenger satisfaction survey. It is obtained from Kaggle and the URL of the page is https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction. There are two separate files, namely "test.csv" and "train.csv" provided by the author so that users do not need to perform the train-test split procedure. Nonetheless, only the "train.csv" dataset is used in this assignment and the train-test split procedure will be performed manually. 80% of the dataset will be the training data whereas 20% will be the test set.

There are a total of 25 attributes and 103,904 records in the dataset. However, there are two attributes namely, "X" and "id" which will not be included in our model building as these attributes do not contribute any meaningful interpretation in our model building. As shown in table 1 below is the list of attributes that are going to be used during our model building.

| No. | Variables | Description | Data Type |
|---|---|---|---|
| 1 | Gender | Gender of passenger (Male, Female) | Categorical |
| 2 | Age | Age of passengers (7-85) | Numerical |
| 3 | Type of Travel | Purpose of the flight (Personal , Business) | Categorical |
| 4 | Class | Travel class of the passenger (Business, Eco, Eco Plus) | Categorical |
| 5 | Customer Type | Loyalty of customer (Loyal customer, Disloyal customer) | Categorical |
| 6 | Flight distance | Flight distance of the journey | Numerical |
| 7 | Inflight wifi service | Satisfaction level of wifi service (On the scale of 0 to 5) | Ordinal |
| 8 | Ease of Online booking | Satisfaction level of online booking service (On the scale of 0 to 5) | Ordinal |
| 9 | Inflight service | Satisfaction level inflight service (On the scale of 0 to 5) | Ordinal |
| 10 | Online boarding | Satisfaction level of online boarding service (On the scale of 0 to 5) | Ordinal |

| | | | |
|---|---|---|---|
| 11 | Inflight entertainment | Satisfaction level of inflight entertainment (On the scale of 0 to 5) | Ordinal |
| 12 | Food and drink | Satisfaction level of food and drinks (On the scale of 0 to 5) | Ordinal |
| 13 | Seat comfort | Satisfaction level of seat comfort (On the scale of 0 to 5) | Ordinal |
| 14 | On-board service | Satisfaction level of on-board service (On the scale of 0 to 5) | Ordinal |
| 15 | Leg room service | Satisfaction level of leg room (On the scale of 0 to 5) | Ordinal |
| 16 | Departure/Arrival time convenient | Satisfaction level of Departure/Arrival time (On the scale of 0 to 5) | Ordinal |
| 17 | Baggage handling | Satisfaction level of baggage handling (On the scale of 0 to 5) | Ordinal |
| 18 | Gate location | Satisfaction level of gate location (On the scale of 0 to 5) | Ordinal |
| 19 | Cleanliness | Satisfaction level of cleanliness (On the scale of 0 to 5) | Ordinal |
| 20 | Check-in service | Satisfaction level of check-in service | Ordinal |
| 21 | Departure Delay in Minutes | Time delayed during departure in minutes | Numerical |
| 22 | Arrival Delay in Minutes (To be removed during model building) | Time delayed during arrival in minutes | Numerical |
| 23 | Satisfaction | Customer's satisfaction level (Satisfaction, neutral or dissatisfied) | Caetegorical |

*Table 1: Description of variables in the dataset*

## 1.1 R Packages

R language is a widely used programming language in statistical computing field, particularly in data science. There are many R packages freely available to users on the central software repository called Comprehensive R Archive Network (CRAN). As shown in table 2 is the list of packages that are going to be used for pre-processing, visualization, model building and analysis of the data

| No. | Name of R Package | Description |
|---|---|---|
| 1 | library(DataExplorer) | Express data exploration with minimum coding |
| 2 | library(dplyr) | Provides tools for efficient data manipulation |
| 3 | library(caTools) | Provides several utility functions |
| 4 | library(caret) | Contains functions to ease model training process for complex regression and classification problems |
| 5 | library(h2o) | Scalable Machine Learning Platform |
| 6 | library(pROC) | Display and analyze ROC curves |
| 7 | library(kernlab) | Kernel-based machine learning methods for classification, regression, clustering |
| 8 | library(e1071) | Functions for support vector machines, Naive Bayes etc. |
| 9 | library(reshape2) | Provides functions to restructure data |

*Table 2: List of R Packages Used*

## 2. Dataset Preparation

Data exploration allows for a thorough understanding of the data before performing any further analysis. It is a crucial step because it allows the analyst to understand the structure of the dataset so that appropriate measures can be taken before using the data.

## 2.1 Data Cleaning and Pre-processing

### a) Structure Of The Dataset

```
str(data)
```

```
'data.frame':   103904 obs. of  25 variables:
 $ X                                : int  0 1 2 3 4 5 6 7 8 9 ...
 $ id                               : int  70172 5047 110028 24026 119299 111157 82113 96462 79485 65725 ...
 $ Gender                           : chr  "Male" "Male" "Female" "Female" ...
 $ Customer.Type                    : chr  "Loyal Customer" "disloyal Customer" "Loyal Customer" "Loyal Customer" ...
 $ Age                              : int  13 25 26 25 61 26 47 52 41 20 ...
 $ Type.of.Travel                   : chr  "Personal Travel" "Business travel" "Business travel" "Business travel" ...
 $ Class                            : chr  "Eco Plus" "Business" "Business" "Business" ...
 $ Flight.Distance                  : int  460 235 1142 562 214 1180 1276 2035 853 1061 ...
 $ Inflight.wifi.service            : int  3 3 2 2 3 3 2 4 1 3 ...
 $ Departure.Arrival.time.convenient: int  4 2 2 5 3 4 4 3 2 3 ...
 $ Ease.of.Online.booking           : int  3 3 2 5 3 2 2 4 2 3 ...
 $ Gate.location                    : int  1 3 2 5 3 1 3 4 2 4 ...
 $ Food.and.drink                   : int  5 1 5 2 4 1 2 5 4 2 ...
 $ Online.boarding                  : int  3 3 5 2 5 2 2 5 3 3 ...
 $ Seat.comfort                     : int  5 1 5 2 5 1 2 5 3 3 ...
 $ Inflight.entertainment           : int  5 1 5 2 3 1 2 5 1 2 ...
 $ On.board.service                 : int  4 1 4 2 3 3 3 5 1 2 ...
 $ Leg.room.service                 : int  3 5 3 5 4 4 3 5 2 3 ...
 $ Baggage.handling                 : int  4 3 4 3 4 4 4 5 1 4 ...
 $ Checkin.service                  : int  4 1 4 1 3 4 3 4 4 4 ...
 $ Inflight.service                 : int  5 4 4 4 3 4 5 5 1 3 ...
 $ Cleanliness                      : int  5 1 5 2 3 1 2 4 2 2 ...
 $ Departure.Delay.in.Minutes       : int  25 1 0 11 0 0 9 4 0 0 ...
 $ Arrival.Delay.in.Minutes         : num  18 6 0 9 0 0 23 0 0 0 ...
 $ satisfaction                     : chr  "neutral or dissatisfied" "neutral or dissatisfied" "satisfied" "neutral or dissatisfied" ...
```

*Figure 1: Structure of the dataset*

As shown in figure 1 is the structure of the dataset. Some of the ordinal numeric variables and categorical variables are found to be incorrectly labelled as “integer”/”chr” instead of “factor”. Also, the “X” and “id” variables are redundant variables as they do not contain any meaningful information. These issues will be dealt with in the following steps.

### b) Removing useless variables

```
#Removing useless variables
library(dplyr)
data= data%>%select(-'X', -'id')
str(data)
```

```
'data.frame':   103904 obs. of  23 variables:
 $ Gender                           : chr  "Male" "Male" "Female" "Female" ...
 $ Customer.Type                    : chr  "Loyal Customer" "disloyal Customer" "Loyal Customer" "Loyal Customer" ...
 $ Age                              : int  13 25 26 25 61 26 47 52 41 20 ...
 $ Type.of.Travel                   : chr  "Personal Travel" "Business travel" "Business travel" "Business travel" ...
 $ Class                            : chr  "Eco Plus" "Business" "Business" "Business" ...
 $ Flight.Distance                  : int  460 235 1142 562 214 1180 1276 2035 853 1061 ...
 $ Inflight.wifi.service            : int  3 3 2 2 3 3 2 4 1 3 ...
 $ Departure.Arrival.time.convenient: int  4 2 2 5 3 4 4 3 2 3 ...
 $ Ease.of.Online.booking           : int  3 3 2 5 3 2 2 4 2 3 ...
 $ Gate.location                    : int  1 3 2 5 3 1 3 4 2 4 ...
 $ Food.and.drink                   : int  5 1 5 2 4 1 2 5 4 2 ...
 $ Online.boarding                  : int  3 3 5 2 5 2 2 5 3 3 ...
 $ Seat.comfort                     : int  5 1 5 2 5 1 2 5 3 3 ...
 $ Inflight.entertainment           : int  5 1 5 2 3 1 2 5 1 2 ...
 $ On.board.service                 : int  4 1 4 2 3 3 3 5 1 2 ...
 $ Leg.room.service                 : int  3 5 3 5 4 4 3 5 2 3 ...
 $ Baggage.handling                 : int  4 3 4 3 4 4 4 5 1 4 ...
 $ Checkin.service                  : int  4 1 4 1 3 4 3 4 4 4 ...
 $ Inflight.service                 : int  5 4 4 4 3 4 5 5 1 3 ...
 $ Cleanliness                      : int  5 1 5 2 3 1 2 4 2 2 ...
 $ Departure.Delay.in.Minutes       : int  25 1 0 11 0 0 9 4 0 0 ...
 $ Arrival.Delay.in.Minutes         : num  18 6 0 9 0 0 23 0 0 0 ...
 $ satisfaction                     : chr  "neutral or dissatisfied" "neutral or dissatisfied" "satisfied" "neutral or dissatisfied" ...
```

*Figure 2: Structure of dataset after removing useless variables*

In figure 2, the “dplyr” library was used in the process of removing “X” and “id”. They are removed from the dataset as they do not contribute any meaningful information. After removing these two variables, there are only 23 variables left.

### c) Converting ordinal and categorical variables to factor

```
#Converting ordinal variables to factor
data = data%>%
  mutate(Gender = factor(Gender),
         Customer.Type = factor(Customer.Type),
         Type.of.Travel = factor(Type.of.Travel),
         Class = factor(Class),
         Inflight.wifi.service = factor(Inflight.wifi.service),
         Departure.Arrival.time.convenient = factor(Departure.Arrival.time.convenient),
         Ease.of.Online.booking = factor(Ease.of.Online.booking),
         Gate.location = factor(Gate.location),
         Food.and.drink = factor(Food.and.drink),
         Online.boarding = factor(Online.boarding),
         Seat.comfort = factor(Seat.comfort),
         Inflight.entertainment = factor(Inflight.entertainment),
         On.board.service = factor(On.board.service),
         Leg.room.service = factor(Leg.room.service),
         Baggage.handling = factor(Baggage.handling),
         Checkin.service = factor(Checkin.service),
         Inflight.service = factor(Inflight.service),
         Cleanliness = factor(Cleanliness))

str(data)
```

```
'data.frame':   103904 obs. of  23 variables:
 $ Gender                          : Factor w/ 2 levels "Female","Male": 2 2 1 1 2 1 2 1 1 2 ...
 $ Customer.Type                   : Factor w/ 2 levels "disloyal Customer",..: 2 1 2 2 2 2 2 2 2 1 ...
 $ Age                             : int  13 25 26 25 61 26 47 52 41 20 ...
 $ Type.of.Travel                  : Factor w/ 2 levels "Business travel",..: 2 1 1 1 1 2 2 1 1 1 ...
 $ Class                           : Factor w/ 3 levels "Business","Eco",..: 3 1 1 1 1 2 2 1 1 2 ...
 $ Flight.Distance                 : int  460 235 1142 562 214 1180 1276 2035 853 1061 ...
 $ Inflight.wifi.service           : Factor w/ 6 levels "0","1","2","3",..: 4 4 3 3 4 4 3 5 2 4 ...
 $ Departure.Arrival.time.convenient: Factor w/ 6 levels "0","1","2","3",..: 5 3 3 6 4 5 5 4 3 4 ...
 $ Ease.of.Online.booking          : Factor w/ 6 levels "0","1","2","3",..: 4 4 3 6 4 3 3 5 3 4 ...
 $ Gate.location                   : Factor w/ 6 levels "0","1","2","3",..: 2 4 3 6 4 2 4 5 3 5 ...
 $ Food.and.drink                  : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 5 2 3 6 5 3 ...
 $ Online.boarding                 : Factor w/ 6 levels "0","1","2","3",..: 4 4 6 3 6 3 3 6 4 4 ...
 $ Seat.comfort                    : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 6 2 3 6 4 4 ...
 $ Inflight.entertainment          : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 4 2 3 6 2 3 ...
 $ On.board.service                : Factor w/ 6 levels "0","1","2","3",..: 5 2 5 3 4 4 4 6 2 3 ...
 $ Leg.room.service                : Factor w/ 6 levels "0","1","2","3",..: 4 6 4 6 5 5 4 6 3 4 ...
 $ Baggage.handling                : Factor w/ 5 levels "1","2","3","4",..: 4 3 4 3 4 4 4 5 1 4 ...
 $ Checkin.service                 : Factor w/ 6 levels "0","1","2","3",..: 5 2 5 2 4 5 4 5 5 5 ...
 $ Inflight.service                : Factor w/ 6 levels "0","1","2","3",..: 6 5 5 5 4 5 6 6 2 4 ...
 $ Cleanliness                     : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 4 2 3 5 3 3 ...
 $ Departure.Delay.in.Minutes      : int  25 1 0 11 0 0 9 4 0 0 ...
 $ Arrival.Delay.in.Minutes        : num  18 0 0 9 0 0 23 0 0 0 ...
 $ satisfaction                    : chr  "neutral or dissatisfied" "neutral or dissatisfied" "satisfied" "neutral or dissatisfied" ...
```

*Figure 3: Structure of dataset after converting ordinal and categorical variables to factor*

Figure 3 shows the results of the converted variables. Those categorical variables are now converted into factor class except for “satisfaction”

### d) Recoding “Satisfaction” into “0” and “1”

```
#reocoding satisfaction variable into 0 and 1 (0 = satisfied, 1 = neutral or dissastisfied)
data = data%>%mutate(satisfaction = factor(satisfaction, levels=c('satisfied','neutral or dissatisfied'),
                                           labels=c("0", "1")))

str(data)
```

```
> str(data)
'data.frame':   103904 obs. of  23 variables:
 $ Gender                          : Factor w/ 2 levels "Female","Male": 2 2 1 1 2 1 2 1 1 2 ...
 $ Customer.Type                   : Factor w/ 2 levels "disloyal Customer",..: 2 1 2 2 2 2 2 2 2 1 ...
 $ Age                             : int  13 25 26 25 61 26 47 52 41 20 ...
 $ Type.of.Travel                  : Factor w/ 2 levels "Business travel",..: 2 1 1 1 1 2 2 1 1 1 ...
 $ Class                           : Factor w/ 3 levels "Business","Eco",..: 3 1 1 1 1 2 2 1 1 2 ...
 $ Flight.Distance                 : int  460 235 1142 562 214 1180 1276 2035 853 1061 ...
 $ Inflight.wifi.service           : Factor w/ 6 levels "0","1","2","3",..: 4 4 3 3 4 4 3 5 2 4 ...
 $ Departure.Arrival.time.convenient: Factor w/ 6 levels "0","1","2","3",..: 5 3 3 6 4 5 5 4 3 4 ...
 $ Ease.of.Online.booking          : Factor w/ 6 levels "0","1","2","3",..: 4 4 3 6 4 3 3 5 3 4 ...
 $ Gate.location                   : Factor w/ 6 levels "0","1","2","3",..: 2 4 3 6 4 2 4 5 3 5 ...
 $ Food.and.drink                  : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 5 2 3 6 5 3 ...
 $ Online.boarding                 : Factor w/ 6 levels "0","1","2","3",..: 4 4 6 3 6 3 3 6 4 4 ...
 $ Seat.comfort                    : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 6 2 3 6 4 4 ...
 $ Inflight.entertainment          : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 4 2 3 6 2 3 ...
 $ On.board.service                : Factor w/ 6 levels "0","1","2","3",..: 5 2 5 3 4 4 4 6 2 3 ...
 $ Leg.room.service                : Factor w/ 6 levels "0","1","2","3",..: 4 6 4 6 5 5 4 6 3 4 ...
 $ Baggage.handling                : Factor w/ 5 levels "1","2","3","4",..: 4 3 4 3 4 4 4 5 1 4 ...
 $ Checkin.service                 : Factor w/ 6 levels "0","1","2","3",..: 5 2 5 2 4 5 4 5 5 5 ...
 $ Inflight.service                : Factor w/ 6 levels "0","1","2","3",..: 6 5 5 5 4 5 6 6 2 4 ...
 $ Cleanliness                     : Factor w/ 6 levels "0","1","2","3",..: 6 2 6 3 4 2 3 5 3 3 ...
 $ Departure.Delay.in.Minutes      : int  25 1 0 11 0 0 9 4 0 0 ...
 $ Arrival.Delay.in.Minutes        : num  18 6 0 9 0 0 23 0 0 0 ...
 $ satisfaction                    : Factor w/ 2 levels "0","1": 2 2 1 2 1 2 2 1 2 2 ...
>
```

*Figure 4: Structure of dataset after recoding the target variable*

The “satisfaction” variable is now being recoded into “0” and “1” respectively. “0” will represent satisfied while “1” represents neutral/dissatisfied.

### e) Missing Values

```
#check for missing data
plot_missing(data)
colSums(is.na(data))
```

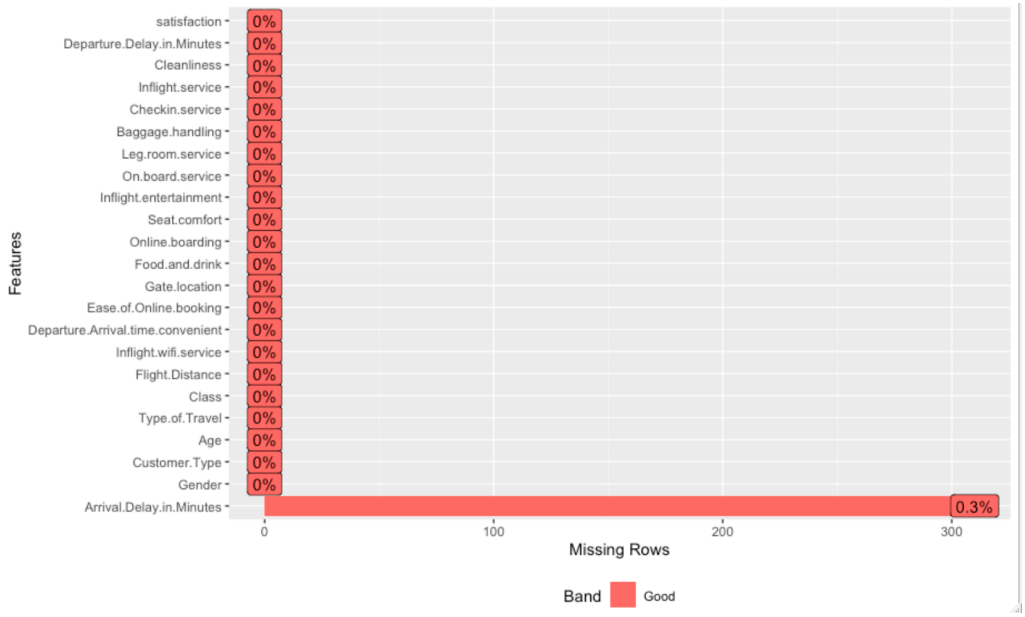



*Figure 3: Plot of missing data*

Based on Figure 3, only the “Arrival.Delay.In.Minutes” variable has missing values. There is around 0.3% of missing data in that particular variable.

```
                      Gender                    Customer.Type                              Age
                           0                                0                                0
             Type.of.Travel                            Class                  Flight.Distance
                           0                                0                                0
      Inflight.wifi.service Departure.Arrival.time.convenient          Ease.of.Online.booking
                           0                                0                                0
            Gate.location                    Food.and.drink                  Online.boarding
                           0                                0                                0
             Seat.comfort                Inflight.entertainment                On.board.service
                           0                                0                                0
        Leg.room.service                     Baggage.handling                  Checkin.service
                           0                                0                                0
        Inflight.service                          Cleanliness          Departure.Delay.in.Minutes
                           0                                0                                0
Arrival.Delay.in.Minutes                         satisfaction
                         310                                0
```

*Figure 4: Summary of missing data in each of the variables*

Based on figure 4, there are 310 missing values in “arrival.delay.in.minutes” .

**f) Correlation Analysis of Variables**

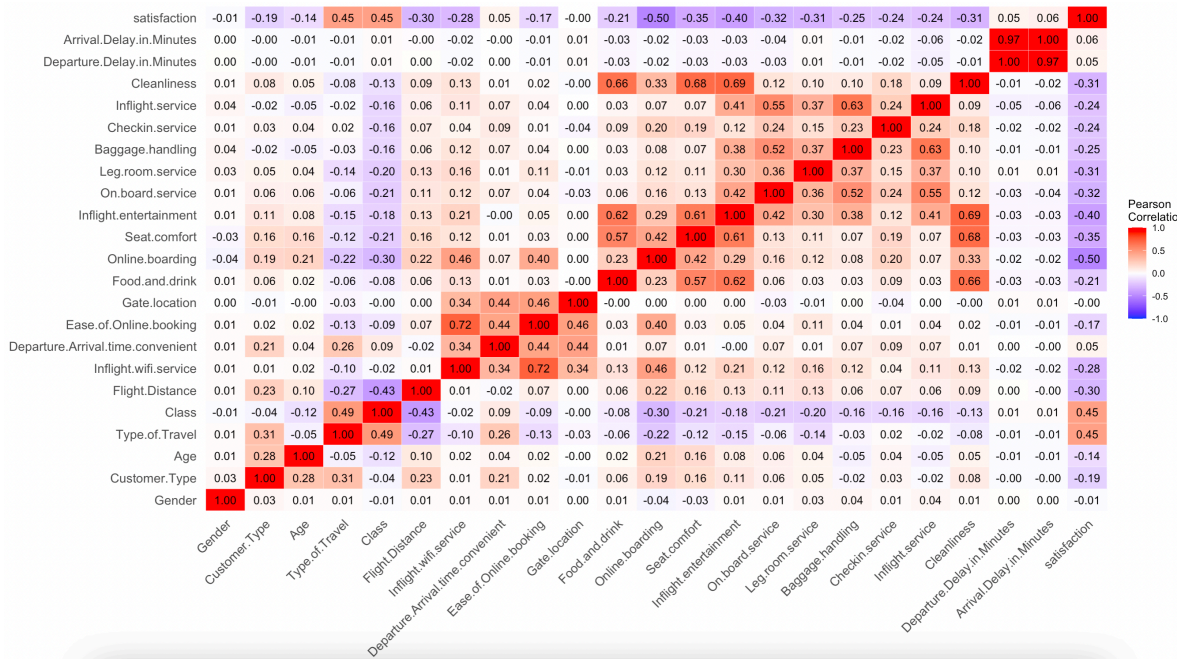



*Figure 5: Correlation heatmap of all variables*

Due to the fact that there is missing data in “Arrival.Delay.In.Minutes”, a custom made correlation heatmap is created to show its correlation with other variables. Otherwise, variables with missing data will be excluded from the correlation heatmap if functions like “plot_correlation()” were to be used.

Based on the correlation heatmap, there is a strong correlation between arrival delay and departure delay (Pearson correlation = 0.97). Since there are some missing values in "Arrival.Delay.In.Minutes", they will be imputed with the data of "Departure.Delay.In.Minutes". This makes absolute sense because if the plane departure time is delayed by 5 minutes, the arrival time should be also delayed by around 5 minutes.

Interestingly, the delay in flight departure does not affect the satisfaction level of a customer as the correlation coefficient is close to 0. Instead, variables such as seat comfort, online boarding service and inflight wifi service have much more weight in affecting a customer's satisfaction level.

Besides that, cleanliness seems to be correlated with food and drinks, seat comfort and inflight entertainment. Also, the "ease of booking" is strongly correlated with inflight wifi service with a pearson correlated value of 0.72. This may indicate that if the inflight wifi service is excellent, customers will have a stable and fast connection to make online bookings and thus give a high rating for "Ease.of.Online.Booking".

### g) Imputation of Missing Values

```
#impute missing values
data = data %>%
  mutate(Arrival.Delay.in.Minutes= coalesce(Arrival.Delay.in.Minutes ,Departure.Delay.in.Minutes))

colSums(is.na(data))
```

```
                    Gender                  Customer.Type                            Age
                         0                              0                              0
             Type.of.Travel                          Class                 Flight.Distance
                         0                              0                              0
      Inflight.wifi.service Departure.Arrival.time.convenient          Ease.of.Online.booking
                         0                              0                              0
             Gate.location                Food.and.drink                 Online.boarding
                         0                              0                              0
              Seat.comfort          Inflight.entertainment                On.board.service
                         0                              0                              0
          Leg.room.service              Baggage.handling                  Checkin.service
                         0                              0                              0
          Inflight.service                    Cleanliness  Departure.Delay.in.Minutes
                         0                              0                              0
  Arrival.Delay.in.Minutes                   satisfaction
                         0                              0
```

*Figure 6: Summary of missing data after imputation*

After imputing data of "Arrival.Delay.in.Minutes" with values from "Departure.Delay.in.Minutes", the dataset is now perfect and free from any missing values.

### h) Removal of extremely correlated variables

```
#Removal of highly correlated features
data= data%>%select(-'Arrival.Delay.in.Minutes')
```

It is noted that arrival delay and departure delay have a very high correlation of 0.97. For this reason, the arrival delay variable is removed from the dataset because highly correlated features will not bring any additional information when building the model, but will increase the complexity of the algorithm and the risk of errors. There are now only 22 variables left in our dataset.

## 3. Exploratory Data Analysis

### a) Proportion of Target Variable (Satisfaction)

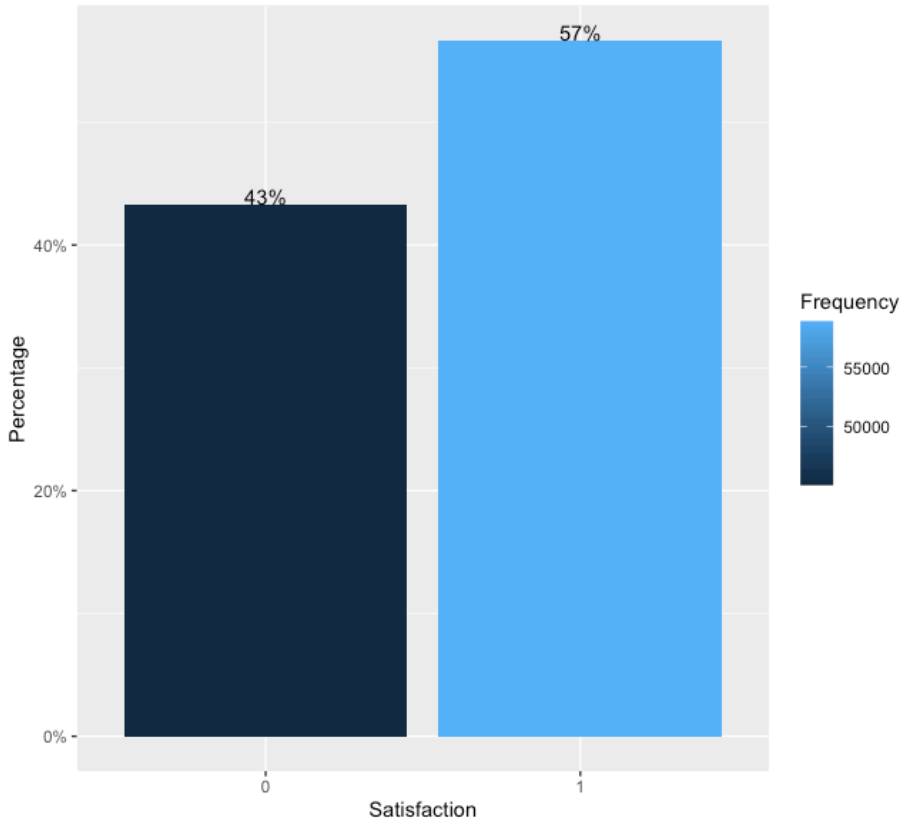



*Figure 7: Barplot of satisfaction*

Class imbalance should be checked before building a predictive model. The imbalance issue may arise due to biased sampling methods or measurement errors/unavailability of the classes. If the class distribution is skewed to one side, the prediction model may be biased towards predicting the skewed class. Hence, the prediction model would not be a good model.

Based on figure 7, 57% of the data is in one class and the remaining 43% is in another class. Thus, it can be said that the proportion of our target variable is fairly balanced.

### b) Satisfaction By Customer Type

```
#Barplot of customer type vs satisfaction level
ggplot(data, aes(x = Customer.Type, fill = satisfaction)) +
  geom_bar(stat='Count', position='dodge') +
  labs(x = 'Type of Customer')+
  guides(fill = guide_legend(title.position="top", title ="Satisfaction") ) +
  scale_fill_brewer(palette = "Set1", labels = c("0(Satisfied)", "1(Neutral/Dissastisfied)"))
```

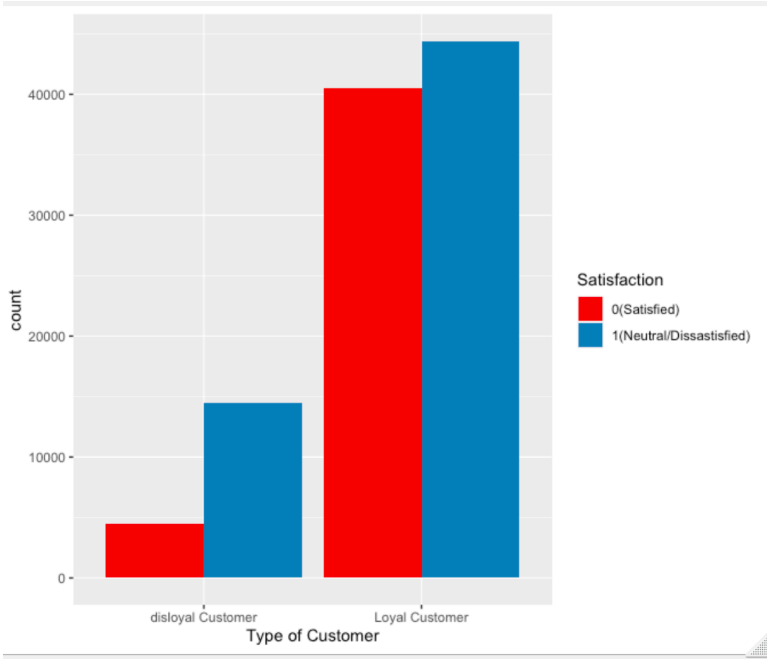



*Figure 8: : Barplot of satisfaction by customer type*

Based on figure 8, disloyal customers are more likely to be dissatisfied/neutral with the airline compared to loyal customers.

**c) Satisfaction by Travel Class and Type of Travel**

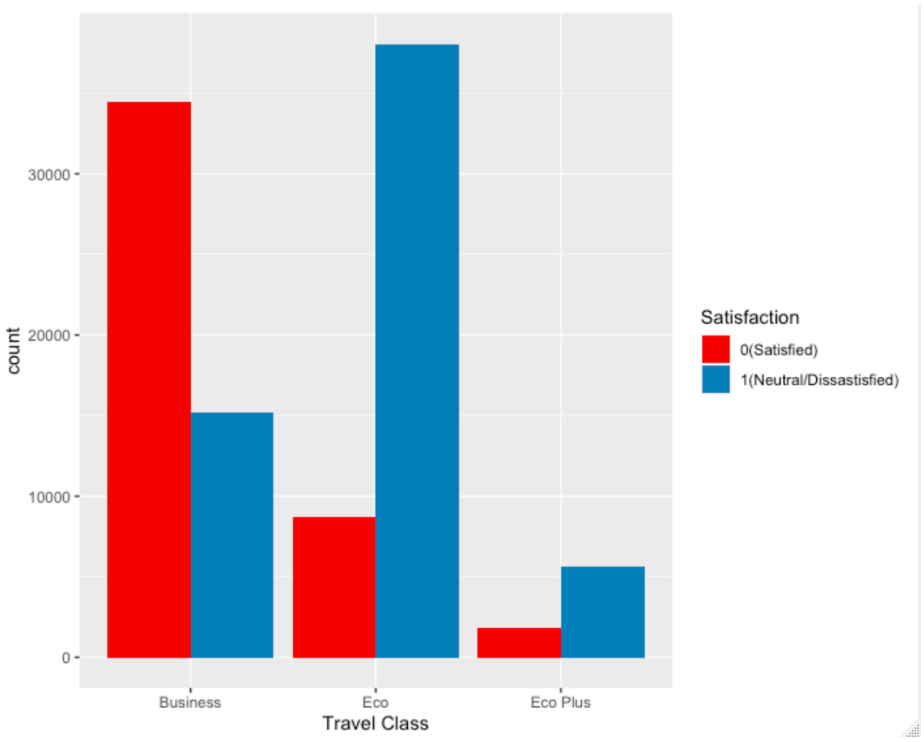



*Figure 9: Barplot of satisfaction by travel class*

Figure 9 shows a barplot of satisfaction by travel class. It is shown that customers who travel in the business class are much more likely to be satisfied of their flight experience compared to those who travel in economy or economy plus.

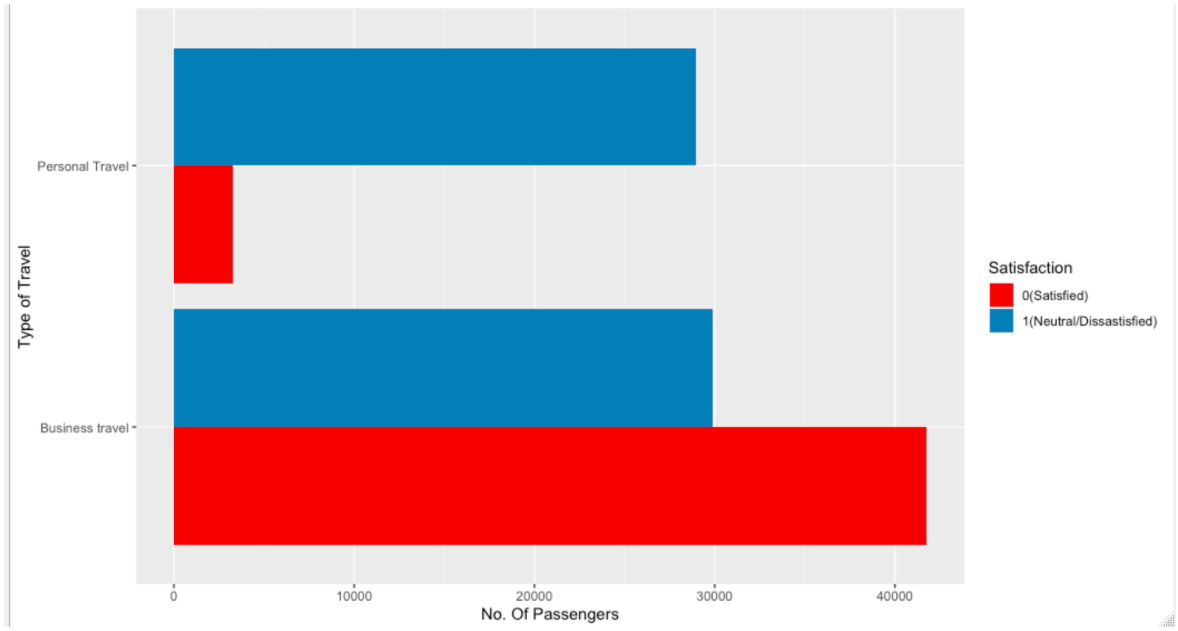



*Figure 10: Barplot of satisfaction by type of travel*

Based on the barplot in figure 10 passengers who travel for personal purposes (most likely vacation) had a much lower satisfaction ratio when compared with those who travel for business purposes.

# 4. Model Implementation

## 4.1 Data Partition

```
#dividing dataset into train and test
library(caTools)
set.seed(1234)
split = sample.split(data$satisfaction, SplitRatio = 0.8)
training_set = subset(data, split == TRUE)
test_set = subset(data, split == FALSE)
```

| | |
|---|---|
| test_set | 20781 obs. of 22 variables |
| training_set | 83123 obs. of 22 variables |

*Figure 11: The training and test set after data partition*

Before training out models , the dataset is split into 2 sections. Due to the huge size of data, the dataset is split in 80:20 ratio. 80% of the data goes into the training set and 20% of the data goes into the testing set. The set.seed() from "caTools" package is used to ensure reproduceable results every time we conduct the splitting process. Figure 11 shows that the splitting process has been successfully completed.

```
#check proportion of target variable
prop.table(table(data$satisfaction))
prop.table(table(training_set$satisfaction))
prop.table(table(test_set$satisfaction))
```

```
> prop.table(table(data$satisfaction))

        0         1
0.4333327 0.5666673
> prop.table(table(training_set$satisfaction))

        0         1
0.4333337 0.5666663
> prop.table(table(test_set$satisfaction))

        0         1
0.4333285 0.5666715
```

*Figure 12: Proportion of the target variable in original, training, and test dataset*

Figure 12 shows that the proportion of target variable, “satisfaction” is the same for train, test and original dataset.

## 4.2 Support Vector Model

### 4.2.1 Model Building Different Kernel Functions

### a) SVM Model (Gaussian RBF)

```
> #building svm model with rbf kernel
> classifier_rbf = ksvm(satisfaction~., data = training_set,
+                       kernel = 'rbfdot')
> classifier_rbf
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0365198259655443

Number of Support Vectors : 12289

Objective Function Value : -9453.755
Training error : 0.037811
```

*Figure 13: SVM Model (Gaussian RBF)*

Figure 13 shows the SVM model being built with Gaussian RBF kernel function with training error of 0.037811.

**b) SVM Model (Linear)**

```
> #building svm model with linear kernel
> library(kernlab)
> library(e1071)
> library(caret)
> classifier = ksvm(satisfaction~., data = training_set, kernel = "vanilladot")
 Setting default kernel parameters
> classifier
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 14201

Objective Function Value : -14115.23
Training error : 0.063893
```

*Figure 14: SVM Model (Linear)*

Figure 14 shows the SVM model being built with linear kernel function with training error of 0.063893.

**c) SVM Model (Hyperbolic Tangent Sigmoid)**

```
> #building svm model with tanhdot
> classifier_tanhdot = ksvm(satisfaction~., data = training_set,
+                       kernel = 'tanhdot')

> classifier_tanhdot
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Hyperbolic Tangent kernel function.
 Hyperparameters : scale =  1  offset =  1

Number of Support Vectors : 35884

Objective Function Value : -25103405
Training error : 0.431529
```

*Figure 15: SVM Model (Hyperbolic Tangent Sigmoid)*

Figure 15 shows the SVM model being built with hyperbolic tangent sigmoid kernel function with training error of 0.431529.

### d) SVM Model (Polynomial)

```
> #building svm model with polydot
> classifier_polydot = ksvm(satisfaction~., data = training_set,
+                           kernel = 'polydot')
 Setting default kernel parameters
> classifier_polydot
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Polynomial kernel function.
 Hyperparameters : degree =  1  scale =  1  offset =  1

Number of Support Vectors : 14204

Objective Function Value : -14115.23
Training error : 0.063917
```

*Figure 16: SVM Model (Polynomial)*

Figure 16 shows the SVM model being built with polynomial kernel function with training error of 0.063917.

### 4.2.2 Model Testing With Training & Test Set

### a) Model Testing with SVM Model (Gaussian RBF)

**Training Set**

```
Confusion Matrix and Statistics


predict_train_rbf     0     1
                0 33911  1034
                1  2109 46069

               Accuracy : 0.9622
                 95% CI : (0.9609, 0.9635)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9227

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9414
            Specificity : 0.9780
         Pos Pred Value : 0.9704
         Neg Pred Value : 0.9562
             Prevalence : 0.4333
         Detection Rate : 0.4080
   Detection Prevalence : 0.4204
      Balanced Accuracy : 0.9597

       'Positive' Class : 0
```

**Test Set**

```
Confusion Matrix and Statistics


predict_test_rbf     0     1
               0  8421   274
               1   584 11502

               Accuracy : 0.9587
                 95% CI : (0.9559, 0.9614)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9156

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9351
            Specificity : 0.9767
         Pos Pred Value : 0.9685
         Neg Pred Value : 0.9517
             Prevalence : 0.4333
         Detection Rate : 0.4052
   Detection Prevalence : 0.4184
      Balanced Accuracy : 0.9559

       'Positive' Class : 0
```

*Figure 17: Confusion Matrix Results of Training & Test Set*

Figure 17 shows the comparison of the confusion matrix results between training and test set using Gaussian RBF kernel function. The training set accuracy is 96.2% while the test set accuracy is slightly lower, 95.9%.

**b) Model Testing with SVM Model (Linear)**

**Training Set**

```
Confusion Matrix and Statistics


predict_train     0     1
            0 32923  2214
            1  3097 44889

               Accuracy : 0.9361
                 95% CI : (0.9344, 0.9378)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8695

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9140
            Specificity : 0.9530
         Pos Pred Value : 0.9370
         Neg Pred Value : 0.9355
             Prevalence : 0.4333
         Detection Rate : 0.3961
   Detection Prevalence : 0.4227
      Balanced Accuracy : 0.9335

       'Positive' Class : 0
```

**Test Set**

```
Confusion Matrix and Statistics


predict_test     0     1
           0  8243   553
           1   762 11223

               Accuracy : 0.9367
                 95% CI : (0.9333, 0.94)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8708

 Mcnemar's Test P-Value : 9.7e-09

            Sensitivity : 0.9154
            Specificity : 0.9530
         Pos Pred Value : 0.9371
         Neg Pred Value : 0.9364
             Prevalence : 0.4333
         Detection Rate : 0.3967
   Detection Prevalence : 0.4233
      Balanced Accuracy : 0.9342

       'Positive' Class : 0
```

*Figure 18: Confusion Matrix Results of Training & Test Set*

Figure 18 shows the comparison of the confusion matrix results between training and test set using Linear kernel function. The training set accuracy is 93.6% while the test set accuracy is slightly higher, 93.7%.

**c) Model Testing with SVM Model (Hyperbolic Tangent Sigmoid)**

**Training Set**

```
Confusion Matrix and Statistics


predict_train_tanhdot     0     1
                    0 18086 17936
                    1 17934 29167

               Accuracy : 0.5685
                 95% CI : (0.5651, 0.5718)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.1477

                  Kappa : 0.1213

 Mcnemar's Test P-Value : 0.9958

            Sensitivity : 0.5021
            Specificity : 0.6192
         Pos Pred Value : 0.5021
         Neg Pred Value : 0.6192
             Prevalence : 0.4333
         Detection Rate : 0.2176
   Detection Prevalence : 0.4334
      Balanced Accuracy : 0.5607

       'Positive' Class : 0
```

**Test Set**

```
Confusion Matrix and Statistics


predict_test_tanhdot    0    1
                   0 4592 4348
                   1 4413 7428

               Accuracy : 0.5784
                 95% CI : (0.5717, 0.5851)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.0003222

                  Kappa : 0.1408

 Mcnemar's Test P-Value : 0.4941273

            Sensitivity : 0.5099
            Specificity : 0.6308
         Pos Pred Value : 0.5136
         Neg Pred Value : 0.6273
             Prevalence : 0.4333
         Detection Rate : 0.2210
   Detection Prevalence : 0.4302
      Balanced Accuracy : 0.5704

       'Positive' Class : 0
```

*Figure 19: Confusion Matrix Results of Training & Test Set*

Figure 19 shows the comparison of the confusion matrix results between training and test set using hyperbolic tangent sigmoid kernel function. The training set accuracy is 56.9% while the test set accuracy is slightly higher, 57.8%.

### c) Model Testing with SVM Model (Polynomial)

**Training Set**

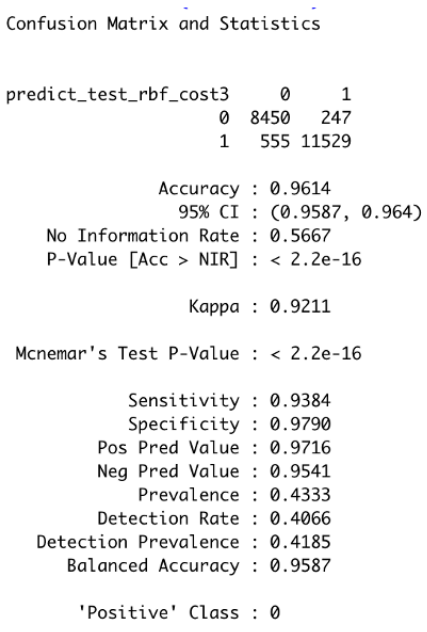
```
Confusion Matrix and Statistics


predict_train_polydot     0     1
                    0 32921  2214
                    1  3099 44889

               Accuracy : 0.9361
                 95% CI : (0.9344, 0.9377)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8695

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9140
            Specificity : 0.9530
         Pos Pred Value : 0.9370
         Neg Pred Value : 0.9354
             Prevalence : 0.4333
         Detection Rate : 0.3961
   Detection Prevalence : 0.4227
      Balanced Accuracy : 0.9335

       'Positive' Class : 0
```

**Test Set**

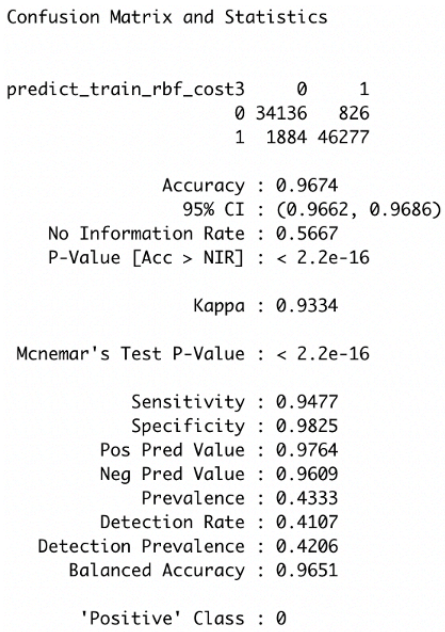
```
Confusion Matrix and Statistics


predict_test_polydot     0     1
                   0  8242   552
                   1   763 11224

               Accuracy : 0.9367
                 95% CI : (0.9333, 0.94)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8708

 Mcnemar's Test P-Value : 6.995e-09

            Sensitivity : 0.9153
            Specificity : 0.9531
         Pos Pred Value : 0.9372
         Neg Pred Value : 0.9363
             Prevalence : 0.4333
         Detection Rate : 0.3966
   Detection Prevalence : 0.4232
      Balanced Accuracy : 0.9342

       'Positive' Class : 0
```

Figure xx shows the comparison of the confusion matrix results between training and test set using polynomial kernel function. The training set accuracy is 93.6% while the test set accuracy is slightly higher, 93.7%.

### 4.2.3 Model Tuning




Test Set




Training Set

Since Gaussian RBF kernel yields the best results, it is selected as the choice of function in the model tuning phase. During the model tuning phase, the cost parameter, C is increased from 1 to 3. This has resulted in a minimal increase in accuracy of both test and training set.

### 4.2.4 Results

| Type of kernel | Accuracy | | Model Fitness | Training Error of Model |
|---|---|---|---|---|
| | Training Set | Test Set | | |
| Gaussian RBF (rbfdot) | 0.9622 | 0.9587 | Good Fit | 0.037811 |
| Linear (vanilladot) | 0.9361 | 0.9367 | Good Fit | 0.063893 |
| Hyperbolic Tangent Sigmoid (tanhdot) | 0.5685 | 0.5784 | Good Fit | 0.431529 |
| Polynomial (polydot) | 0.9361 | 0.9367 | Good Fit | 0.063917 |
| Gaussian RBF with cost parameter = 3 (Tuned Model) | 0.9674 | 0.9614 | Good Fit | 0.0326 |

The table above shows the overall results of the different SVM models built using different kernel functions.

Kernel function specifies a nonlinear mapping such as radial basis, polynomial, hyperbolic tangent sigmoid or linear. The ksvm() function will use the Gaussian RBF kernel by default. In this dataset, the SVM model built using Gaussian RBF kernel function outperforms the other models built using different functions. This result is expected as Gaussian RBF kernel function is a popular function due to its good performance shown in the past for many types of data (Lantz, 2019).

## 4.3 Random Forest

### 4.3.1 Model Building

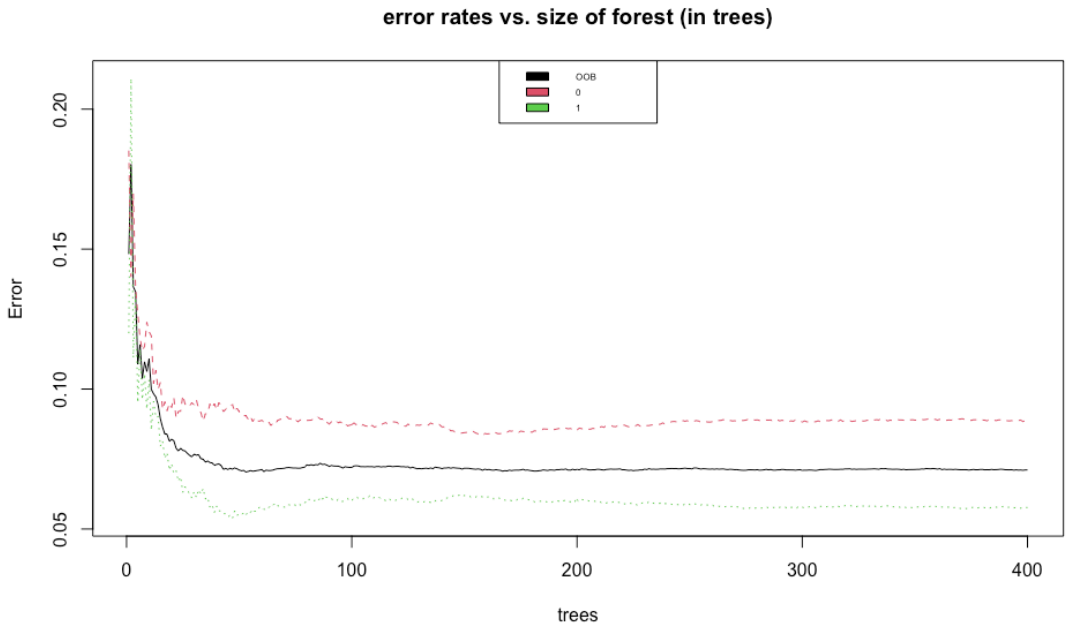
```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Model Building~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
library(randomForest)
library(caret)
set.seed(1234)
randomforest_classifier = randomForest(satisfaction ~ ., data= training_set,
                                       ntree = 400,
                                       replace = TRUE,
                                       sampsize = 200,
                                       nodesize = 5,
                                       importance = TRUE,
                                       proximity = FALSE,
                                       norm.votes = TRUE,
                                       keep.forest = TRUE,
                                       keep.inbag = TRUE)
```

```
Call:
 randomForest(formula = satisfaction ~ ., data = training_set,      ntree = 400, replace = TRUE, sampsize = 200, nodesize = 5,
 importance = TRUE, proximity = FALSE, norm.votes = TRUE,      keep.forest = TRUE, keep.inbag = TRUE)
               Type of random forest: classification
                     Number of trees: 400
No. of variables tried at each split: 4

        OOB estimate of  error rate: 7.11%
Confusion matrix:
      0     1 class.error
0 32825  3195  0.08870072
1  2716 44387  0.05766087
```

The random forest is built starting with 400 trees and the OOB estimate of error rate is 7.11%

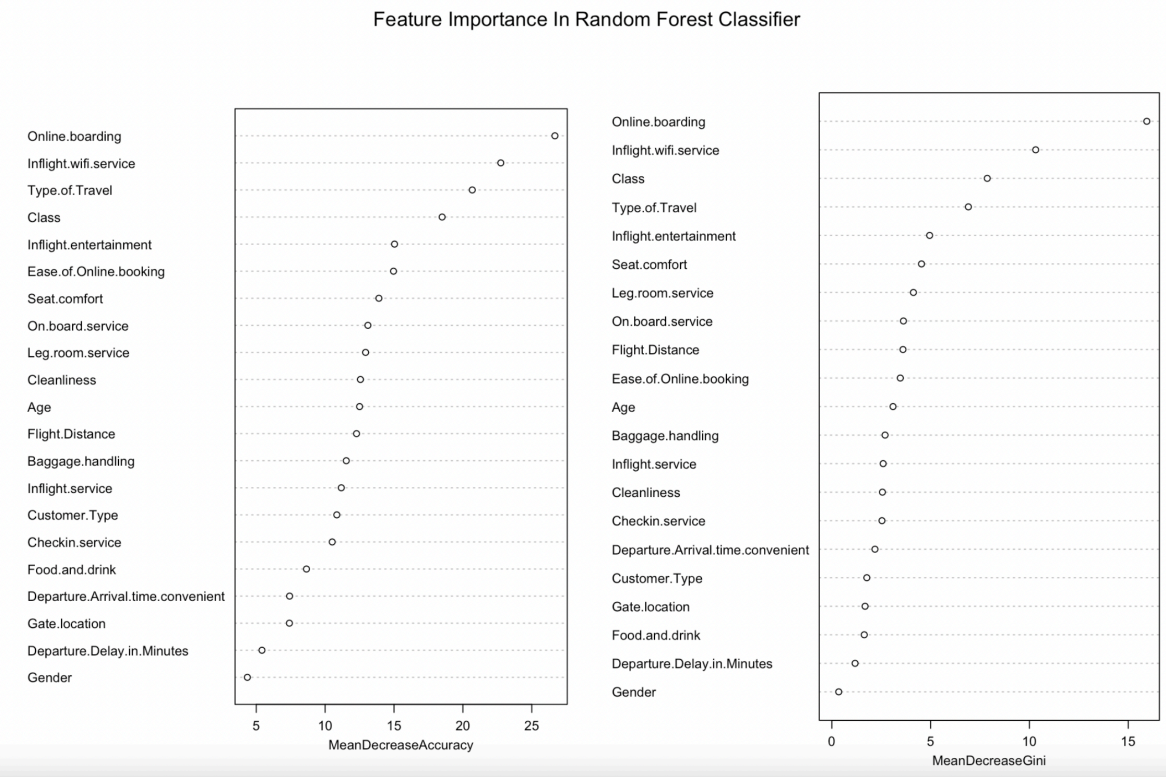
### 4.3.2 Error analysis and variable significance

**a) OOB Error Plot**




The figure above shows the out-of-error (OOB) error plot in our random forest model. It is observed that the error is having diminishing returns starting from around 100 trees. Since the OOB error is calculated on the samples that were not selected (out of the bag) on each iteration of the algorithm, it is just an estimation of our model performance. It can be assumed that the smaller the errors, the better the performance of the model.

## b) Feature Importance




The plot above shows the importance of each of the variables in the random forest model which is generated using varImpPlot() function in R. Online boarding, inflight wifi service, and airline classes are labelled as top three most important factors which can greatly affect the satisfaction of a customer.

### 4.3.3 Model Testing

**Training Set**

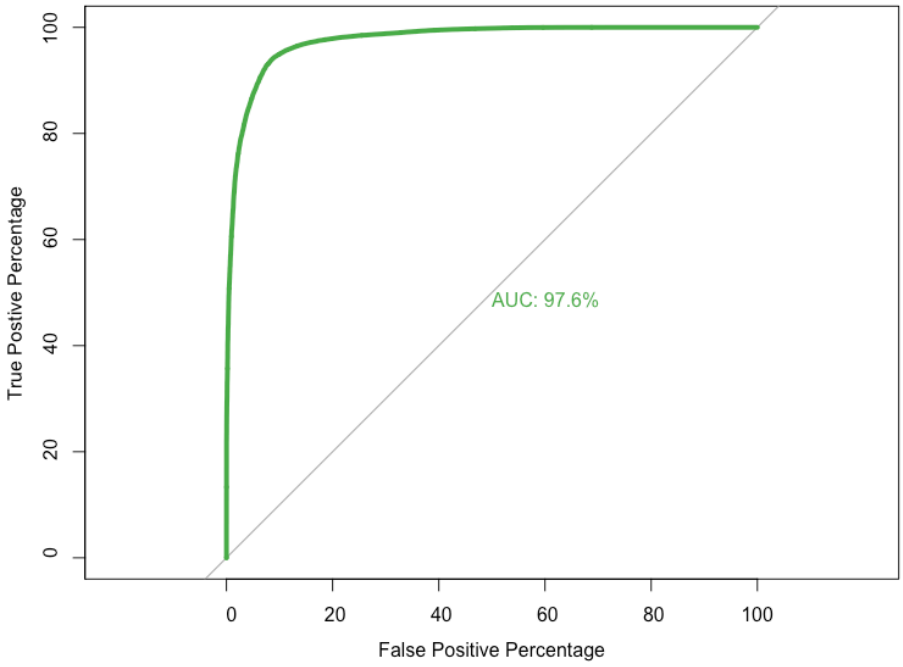
```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 32845  2699
         1  3175 44404

               Accuracy : 0.9293
                 95% CI : (0.9276, 0.9311)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8559

 Mcnemar's Test P-Value : 5.731e-10

            Sensitivity : 0.9119
            Specificity : 0.9427
         Pos Pred Value : 0.9241
         Neg Pred Value : 0.9333
             Prevalence : 0.4333
         Detection Rate : 0.3951
   Detection Prevalence : 0.4276
      Balanced Accuracy : 0.9273

       'Positive' Class : 0
```

**Test Set**

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0  8169   659
         1   836 11117

               Accuracy : 0.9281
                 95% CI : (0.9245, 0.9315)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8532

 Mcnemar's Test P-Value : 5.317e-06

            Sensitivity : 0.9072
            Specificity : 0.9440
         Pos Pred Value : 0.9254
         Neg Pred Value : 0.9301
             Prevalence : 0.4333
         Detection Rate : 0.3931
   Detection Prevalence : 0.4248
      Balanced Accuracy : 0.9256

       'Positive' Class : 0
```

The figure above shows the comparison of the confusion matrix results of the random forest model between training and test set. The training set accuracy is 92.9% while the test set accuracy is slightly lower, 92.8%. Hence, the model is a good fit. Also, noted that the sensitivity (true positive rate) is lower than the specificity (true negative rate) which indicated that the random forest model is slightly better at classifying a customer who is neutral/dissatisfied compared to a satisfied customer.

### 4.3.4 ROC & AUC




From the ROC plot above, the model is a very good fit as the curve is well above the average line. The high AUC of 97.6% also suggested that the model is a very good model.

### 4.3.5 Results

| Model | Accuracy | | Model Fitness | AUC |
|---|---|---|---|---|
| | Training Set | Test Set | | |
| Random Forest with 400 trees | 0.9361 | 0.9367 | Good Fit | 0.976 |

The random forest model has achieved good results with 93.7% accuracy on the test set with a high AUC of 0.976.

## 4.4 Artificial Neural Network

In artificial neural network, the neural networks work best when the input data are scaled to a very narrow range around 0. In our dataset, some the variables like flight distance have a wide range of values up to 5000. Hence, normalization/standardization will help to rescale these values.

### 4.4.1 Normalization of training and test set

```
#normalizing the data
normalize = function(x) {
  return((x- min(x))/ (max(x) - min(x)))
}
```

The min-max normalization formula is created using the code shown in the above figure. Prior to the normalization process, all the variables in the training and test dataset must be converted into numeric class (refer to the Rscript).

```
#normalize training and test
training_set_normalized =as.data.frame(lapply(training_set_converted[,c(1:22)], normalize))
str(training_set_normalized)

test_set_normalized =as.data.frame(lapply(test_set_converted[,c(1:22)], normalize))
str(test_set_normalized)

#convert "satisfaction" back to factor class
training_set_normalized$satisfaction = as.factor(training_set_normalized$satisfaction)
test_set_normalized$satisfaction = as.factor(test_set_normalized$satisfaction)
```

In the final step, only the target variable is converted back into factor class as artificial neural network models only accept the independent variables in numeric form.

```
'data.frame':   20781 obs. of  22 variables:
 $ Gender                          : num  1 0 1 0 1 0 1 0 0 1 ...
 $ Customer.Type                   : num  1 1 1 1 1 0 1 1 1 1 ...
 $ Age                             : num  0.5128 0.2436 0.0256 0.4615 0.4231 ...
 $ Type.of.Travel                  : num  1 0 0 1 1 0 0 0 1 0 ...
 $ Class                           : num  0.5 0 0.5 0.5 1 0 0 0 0 0 ...
 $ Flight.Distance                 : num  0.251 0.422 0.231 0.146 0.105 ...
 $ Inflight.wifi.service           : num  0.4 0.6 0.4 0.6 0.8 0.6 0 0.2 0.4 0.6 ...
 $ Departure.Arrival.time.convenient: num  0.8 0.6 0.8 1 0.6 1 0 0.2 0.8 0.6 ...
 $ Ease.of.Online.booking          : num  0.4 0.6 0.4 0.6 0.8 0.6 0 0.2 0.4 0.6 ...
 $ Gate.location                   : num  0.5 0.5 0.75 1 0.25 0.75 0.5 0 0 0.5 ...
 $ Food.and.drink                  : num  0.4 0.8 0.4 1 0.4 0.6 0.4 0.6 0.8 1 ...
 $ Online.boarding                 : num  0.4 0.8 0.4 0.8 0.8 0.6 1 1 0.4 0.4 ...
 $ Seat.comfort                    : num  0.4 0.8 0.2 1 0.4 0.6 0.6 1 0.8 0.6 ...
 $ Inflight.entertainment          : num  0.4 0.8 0.4 0.6 0.4 0.6 0.8 0.6 0.8 1 ...
 $ On.board.service                : num  0.6 1 0.2 0.6 1 0.8 0.8 0.6 0.8 1 ...
 $ Leg.room.service                : num  0.6 0.6 1 0.6 0.2 0.8 0.8 0.6 1 1 ...
 $ Baggage.handling                : num  0.75 0.75 0.5 1 0.75 1 0.75 0.5 0.75 1 ...
 $ Checkin.service                 : num  0.6 1 0.8 0.6 0.6 1 0.8 0.8 0.8 0.8 ...
 $ Inflight.service                : num  1 0.8 0.6 0.6 0.8 1 0.8 0.6 1 1 ...
 $ Cleanliness                     : num  0.4 0.8 0.4 0.8 0.4 0.6 0.6 1 0.8 0.6 ...
 $ Departure.Delay.in.Minutes      : num  0.00965 0.05252 0 0.05573 0.02465 ...
 $ satisfaction                    : Factor w/ 2 levels "0","1": 2 1 2 2 2 2 1 1 2 1 ...
>
```

The figure above shows the example of the final form of our test set.

### 4.4.2 Model Building with h2o package

**a) Starting h2o package**

```
#building model with h2o package
install.packages('h2o')
library(h2o)
library(caret)

#Start a local cluster with 6GB RAM and all the cores"
h2o.init(nthreads = -1, max_mem_size = '6G')


#converting training and test set into H2o dataframe
train = as.h2o(training_set_normalized)
test = as.h2o(test_set_normalized)
str(train)
train
#take a look at the contents of the h2oframe(train and test set)
h2o.describe(train)
h2o.describe(test)
View(train)
```

Before building the model, a local cluster needs to be started. Then, the training and test set needs to be converted into a "h2o frame" as data frame is not accepted in the h2o package.

**b) Building ANN model**

```
#building neural network using H2o package, ReClu function
# 1 layers with 5 neurons  and 100 iterations (epoch),
n = h2o.deeplearning(x = 1:21,
                     y = 'satisfaction',
                     training_frame = train,
                     standardize = FALSE,
                     activation = 'RectifierWithDropout',
                     hidden = 5,
                     seed = 123,
                     epochs = 100)

n
```

The neural network model is built with rectifier function, 1 layer containing only 5 neurons and 100 iterations.  Standardize is set to "FALSE" as the dataset has already been normalized.

## c) Performance of the model

```
> h2o.performance(n)
H2OBinomialMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 9928 samples **

MSE:  0.08189903
RMSE:  0.2861801
LogLoss:  0.2827313
Mean Per-Class Error:  0.1010104
AUC:  0.9635161
AUCPR:  0.9701968
Gini:  0.9270323

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
         0    1    Error       Rate
0     3708  633 0.145819  =633/4341
1      314 5273 0.056202  =314/5587
Totals 4022 5906 0.095387  =947/9928

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.390662    0.917602 278
2                       max f2  0.257926    0.945321 333
3                 max f0point5  0.724971    0.924681 160
4                 max accuracy  0.437136    0.904613 262
5                max precision  0.999434    1.000000   0
6                   max recall  0.001765    1.000000 398
7              max specificity  0.999434    1.000000   0
8             max absolute_mcc  0.390662    0.806321 278
9   max min_per_class_accuracy  0.531000    0.900483 229
10 max mean_per_class_accuracy  0.469810    0.901472 250
11                      max tns  0.999434 4341.000000   0
12                      max fns  0.999434 5553.000000   0
13                      max fps  0.000082 4341.000000 399
14                      max tps  0.001765 5587.000000 398
15                      max tnr  0.999434    1.000000   0
16                      max fnr  0.999434    0.993914   0
17                      max fpr  0.000082    1.000000 399
18                      max tpr  0.001765    1.000000 398
```

The performance of the model is shown by using the h2o.performance() function. It is observed that the RMSE and MSE of the model is 0.2862 and 0.0818 respectively, which are relatively low. Also, the model has a high AUC score of 0.9635 which indicates that the model is a good model.

### 4.4.3 Model Testing

**Training Set**

```
Confusion Matrix and Statistics

          0     1
  0 30705  2463
  1  5315 44640

               Accuracy : 0.9064
                 95% CI : (0.9044, 0.9084)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8077

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8524
            Specificity : 0.9477
         Pos Pred Value : 0.9257
         Neg Pred Value : 0.8936
             Prevalence : 0.4333
         Detection Rate : 0.3694
   Detection Prevalence : 0.3990
      Balanced Accuracy : 0.9001

       'Positive' Class : 0
```

**Test Set**

```
Confusion Matrix and Statistics

          0     1
  0  7774   728
  1  1231 11048

               Accuracy : 0.9057
                 95% CI : (0.9017, 0.9097)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8068

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8633
            Specificity : 0.9382
         Pos Pred Value : 0.9144
         Neg Pred Value : 0.8997
             Prevalence : 0.4333
         Detection Rate : 0.3741
   Detection Prevalence : 0.4091
      Balanced Accuracy : 0.9007

       'Positive' Class : 0
```

The figure above shows the comparison of the confusion matrix results of the ANN model between training and test set. The training set accuracy is 90.64% while the test set accuracy is almost the same as the training set (90.57%). Hence, the model is a perfect fit.

### 4.4.4 Model Tuning

#### a) Changing Parameters

```
#IMPROVING THE MODEL , ReClu function
#2 layers with 20 neurons each , 3 fold cross validation and 1000 iterations (epoch)
nimproved = h2o.deeplearning(x = 1:22,
                             y = 'satisfaction',
                             training_frame = train,
                             standardize = FALSE,
                             nfolds = 3,
                             activation = 'RectifierWithDropout',
                             hidden = c(20, 20),
                             seed = 123,
                             epochs = 1000)
```

The model is tuned by changing some of the parameters. The improved model now has an extra layer, where each of the layer has 20 neurons. Also, a 3-fold cross-validation is used to validate a model internally so that the model performance can be estimated without having to sacrifice a validation split.

**b) Performance of the tuned model**

```
> h2o.performance(nimproved)
H2OBinomialMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 9928 samples **

MSE:  0.06816196
RMSE:  0.2610785
LogLoss:  0.221841
Mean Per-Class Error:  0.06797337
AUC:  0.9841816
AUCPR:  0.987162
Gini:  0.9683632

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0    1    Error       Rate
0      3935  406 0.093527  =406/4341
1       237 5350 0.042420  =237/5587
Totals 4172 5756 0.064766  =643/9928

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.705635    0.943313 185
2                       max f2  0.585264    0.965994 237
3                 max f0point5  0.708748    0.957038 184
4                 max accuracy  0.705635    0.935234 185
5                max precision  0.999953    1.000000   0
6                   max recall  0.000003    1.000000 399
7              max specificity  0.999953    1.000000   0
8             max absolute_mcc  0.705635    0.868345 185
9   max min_per_class_accuracy  0.705635    0.906473 185
10 max mean_per_class_accuracy  0.705635    0.932027 185
11                      max tns  0.999953 4341.000000   0
12                      max fns  0.999953 5372.000000   0
13                      max fps  0.000003 4341.000000 399
14                      max tps  0.000003 5587.000000 399
15                      max tnr  0.999953    1.000000   0
16                      max fnr  0.999953    0.961518   0
17                      max fpr  0.000003    1.000000 399
18                      max tpr  0.000003    1.000000 399
```

It is observed that the performance of the tuned model has increased substantially. The RMSE and MSE of the model has now dropped to 0.2610 and 0.06816 respectively, which is even lower than before. Also, AUC of the model has increased to 0.9842 which is almost close to perfection.

**c) Accuracy of tuned model on training and test set**

**Training Set**

```
Confusion Matrix and Statistics

      0     1
0 32679  1921
1  3341 45182

               Accuracy : 0.9367
                 95% CI : (0.935, 0.9383)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8705

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9072
            Specificity : 0.9592
         Pos Pred Value : 0.9445
         Neg Pred Value : 0.9311
             Prevalence : 0.4333
         Detection Rate : 0.3931
   Detection Prevalence : 0.4163
      Balanced Accuracy : 0.9332

       'Positive' Class : 0
```

**Test Set**

```
Confusion Matrix and Statistics

      0     1
0  8093   514
1   912 11262

               Accuracy : 0.9314
                 95% CI : (0.9279, 0.9348)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8595

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8987
            Specificity : 0.9564
         Pos Pred Value : 0.9403
         Neg Pred Value : 0.9251
             Prevalence : 0.4333
         Detection Rate : 0.3894
   Detection Prevalence : 0.4142
      Balanced Accuracy : 0.9275

       'Positive' Class : 0
```

The confusion matrix demonstrated that the tuned model has some improvement in terms of accuracy. The training set accuracy has increased by around 3.1% and same goes to the test set.

### 4.4.5 Results

| Model | Accuracy | | Model Fitness | AUC |
|---|---|---|---|---|
| | Training Set | Test Set | | |
| ANN Model with Rectifier function (1 layer with 5 neurons without cross validation) | 0.9064 | 0.9057 | Good Fit | 0.9635 |
| Tuned ANN Model with Rectifier function (2 layers, 20 neurons each, 3 fold cross validation) | 0.9367 | 0.9314 | Good Fit | 0.9842 |

The tuned ANN model has achieved good results with 93.1% accuracy on the test set with a high AUC of 0.9842.

## 5. Evaluation & Conclusion

By first glance at these three models, random forest is the model that outperforms all the other models with the highest accuracy. However, accuracy is not the only metric that is looked upon in determining the most appropriate model. Other metrics like AUC should be considered when evaluating the model. The AUC value reflects the capability of the model to distinguish between the classes. A value of 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding. In short, the higher the AUC value, the better the model is at predicting the satisfaction of airline passengers in this situation. In this case, our ANN model has a highest AUC value 0.9842 after tuning. However, random forest is chosen as the most appropriate mode because it is less computationally expensive and is able to train a model within a short period of time compared to ANN and SVM.

Finally, online boarding, inflight wifi service, and airline classes are labelled as top three most important factors which can greatly affect the satisfaction of a customer.