IMZ

CAPSTONE PROJECT – FLIGHT PRICE PREDICTION

FINAL REPORT

Table of Contents

1. INTRODUCTION:

1.1 Problem Statement Definition:

To predict the price of a flight using machine Learning techniques with the help of the different variables in the given dataset.

1.2 Project Objectives:

- ✓ To understand the **flight price pattern and trends** at different times
- ✓ To identify the **variables that are most significant** in influencing the flight price and plot the relationship between those variables
- ✓ To check **fluctuations in flight prices during weekdays and peak hours** of a day.
- ✓ To **build a robust predictive model** to predict the prices of Flights and deploy them in future datasets that lacks information on price.

1.3 Scope of Project:

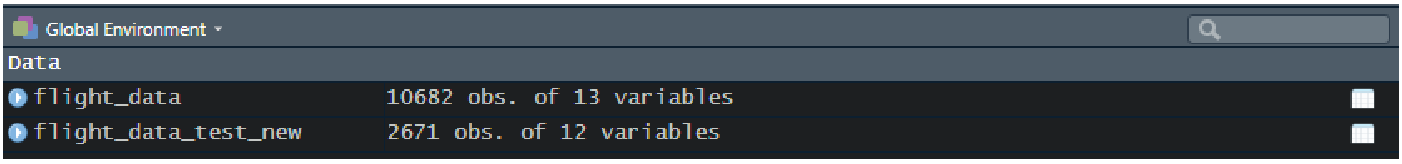
Understanding the patterns and trends in Flight price across different variables helps :

- ✓ To enable the end users / customers (eg. Passengers, Business Deligates, etc.) to plan their journeys accordingly.
- ✓ To enable the Tourism agencies / Travel agencies to provide exclusive promotions and offers for bulk bookings during the periods of lower flights prices.

2. EXPLORATORY DATA ANALYSIS:

2.1 Importing dataset:

The given dataset is in .xlsx format. Hence, the command ‘read_excel’ is used to import file



Note:

flight_data -- > *original data set on which models are trained (included Prioce variable)*

flight_data_test_new -- > *Dataset for which Price is to be predicted using best model algorithm*

2.2 Visual Inspection of Dataset:

The given dataset is explored to identify the different variables present, number of rows and columns and also the datatypes used for different variables.

❖ Flight_data:

```
> names(flight_data)
 [1] "Airline"         "Date_of_Journey" "Source"          "Destination"     "Route"           "Dep_Time"
 [7] "Arrival_Time"    "Duration"        "Total_Stops"     "Additional_Info" "Price"
> nrow(flight_data)
[1] 10683
> ncol(flight_data)
[1] 11
> dim(flight_data)
[1] 10683    11
> str(flight_data)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':	10683 obs. of  11 variables:
 $ Airline        : chr  "IndiGo" "Air India" "Jet Airways" "IndiGo" ...
 $ Date_of_Journey: chr  "24/03/2019" "1/05/2019" "9/06/2019" "12/05/2019" ...
 $ Source         : chr  "Banglore" "Kolkata" "Delhi" "Kolkata" ...
 $ Destination    : chr  "New Delhi" "Banglore" "Cochin" "Banglore" ...
 $ Route          : chr  "BLR <U+2192> DEL" "CCU <U+2192> IXR <U+2192> BBI <U+2192> BLR" "DEL <U+2192> LKO
+2192> COK" "CCU <U+2192> NAG <U+2192> BLR" ...
 $ Dep_Time       : chr  "22:20" "05:50" "09:25" "18:05" ...
 $ Arrival_Time   : chr  "01:10 22 Mar" "13:15" "04:25 10 Jun" "23:30" ...
 $ Duration       : chr  "2h 50m" "7h 25m" "19h" "5h 25m" ...
 $ Total_Stops    : chr  "non-stop" "2 stops" "2 stops" "1 stop" ...
 $ Additional_Info: chr  "No info" "No info" "No info" "No info" ...
 $ Price          : num  3897 7662 13882 6218 13302 ...
> summary(flight_data$Price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1759    5277    8372    9087   12373   79512
```

- ✓ The train set consists of 10683 entries with 11 variables.
- ✓ It contains 10 variables of “Character” class and 1 variable with “numeric” class.
- ✓ The Flight price ranges from Rs.1759 to Rs.79512

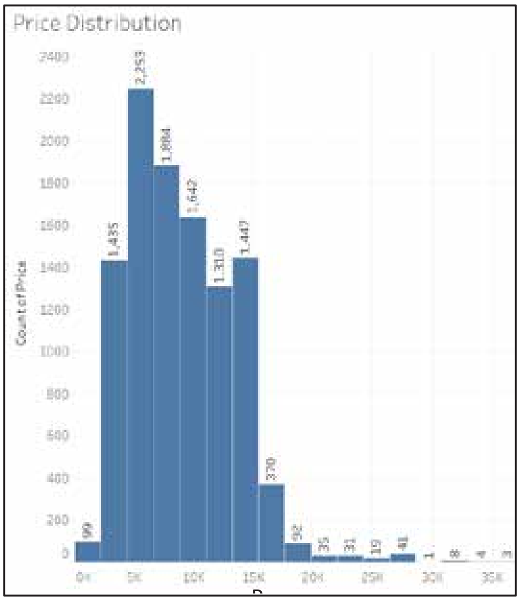
❖ Flight_data_test_new:

```
> names(flight_data_test)
 [1] "Airline"         "Date_of_Journey" "Source"          "Destination"     "Route"           "Dep_Time"
 [7] "Arrival_Time"    "Duration"        "Total_Stops"     "Additional_Info"
> nrow(flight_data_test)
[1] 2671
> ncol(flight_data_test)
[1] 10
> dim(flight_data_test)
[1] 2671   10
> str(flight_data_test)
Classes 'tbl_df', 'tbl' and 'data.frame':	2671 obs. of  10 variables:
 $ Airline        : chr  "Jet Airways" "IndiGo" "Jet Airways" "Multiple carriers" ...
 $ Date_of_Journey: chr  "6/06/2019" "12/05/2019" "21/05/2019" "21/05/2019" ...
 $ Source         : chr  "Delhi" "Kolkata" "Delhi" "Delhi" ...
 $ Destination    : chr  "Cochin" "Banglore" "Cochin" "Cochin" ...
 $ Route          : chr  "DEL <U+2192> BOM <U+2192> COK" "CCU <U+2192> MAA <U+2192> BLR" "DEL <U+2192> BOM <U+2192> COK"
 "DEL <U+2192> BOM <U+2192> COK" ...
 $ Dep_Time       : chr  "17:30" "06:20" "19:15" "08:00" ...
 $ Arrival_Time   : chr  "04:25 07 Jun" "10:20" "19:00 22 May" "21:00" ...
 $ Duration       : chr  "10h 55m" "4h" "23h 45m" "13h" ...
 $ Total_Stops    : chr  "1 stop" "1 stop" "1 stop" "1 stop" ...
 $ Additional_Info: chr  "No info" "No info" "In-flight meal not included" "No info" ...
```

- ✓ The train set consists of 2671 entries with 10 variables.
- ✓ It contains 10 variables of “Character” class
- ✓ The Flight price is not given in test dataset and it is to be predicted using predictive algorithms

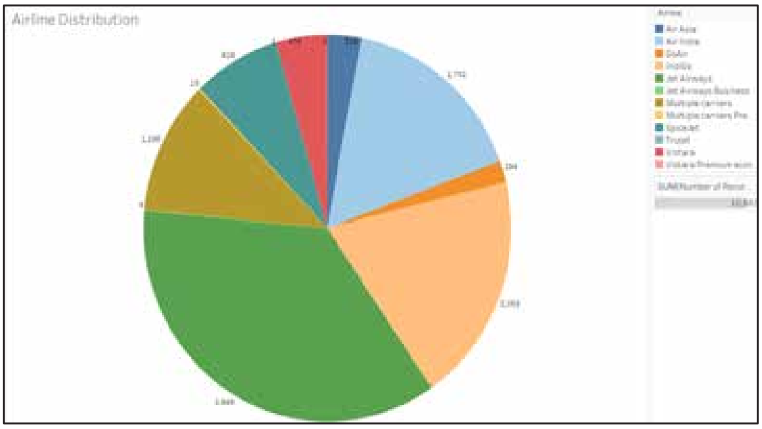
2.3 Univariate Analysis:

(i) Price - Distribution



Highest frequency of entries = Rs.4000-5000

(ii) Airline - Distribution

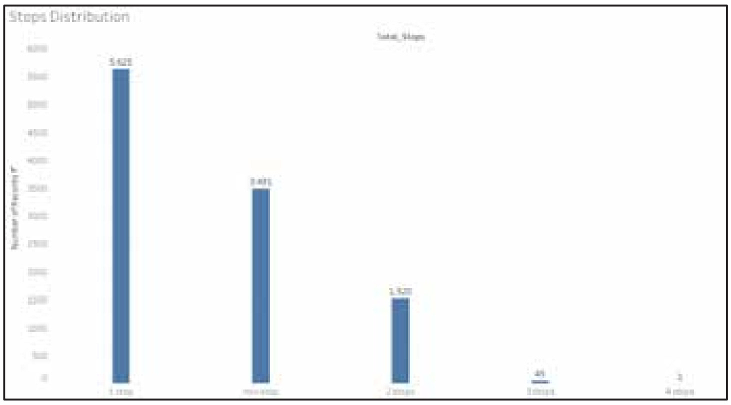


Highest frequency of entries = Jet Airways
Lowest frequency of entries = Trujet

(iii) Most Popular Routes

The most travelled route in the given data is
Delhi to Cochin via Bombay

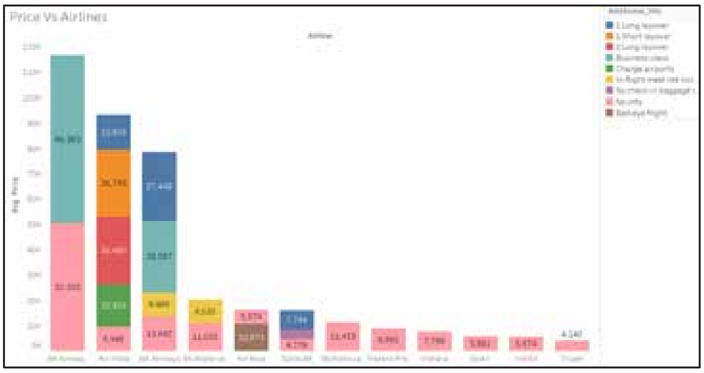
(iv) No. of Stops - Distribution



Highest frequency of entries for Stops is
1 Stop (5625 records)

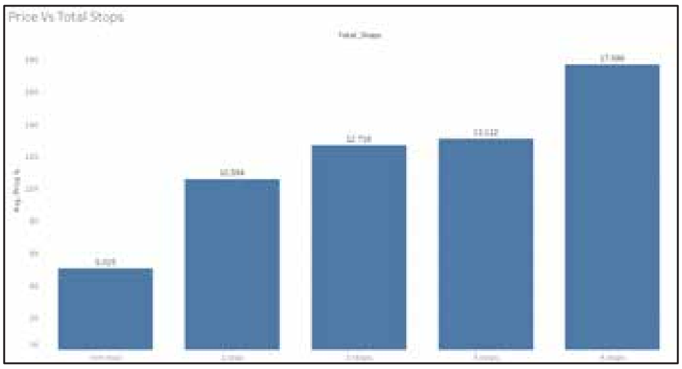
2.4 Bi-variate / Multi-variate Analysis:

(i) Price Vs Airlines



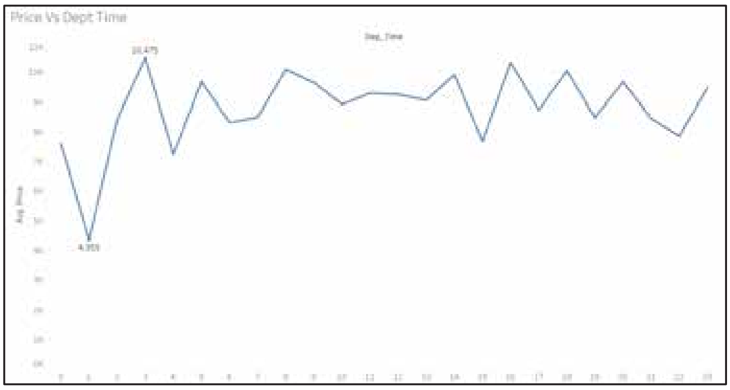
Based on average price,
Costliest Air carrier = **Jet Airways' Business Class**
Cheapest Airline = **piceJet (with no-check in baggage)**

(ii) Price Vs No. of Stops



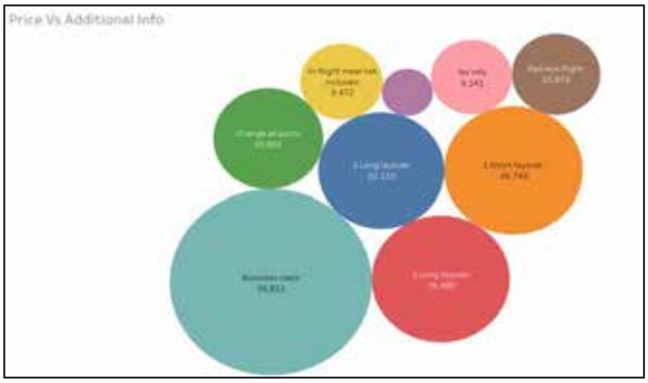
Based on average price,
Higher the number of stops, higher is the flight price.

(iii) Price Vs Dep_Time



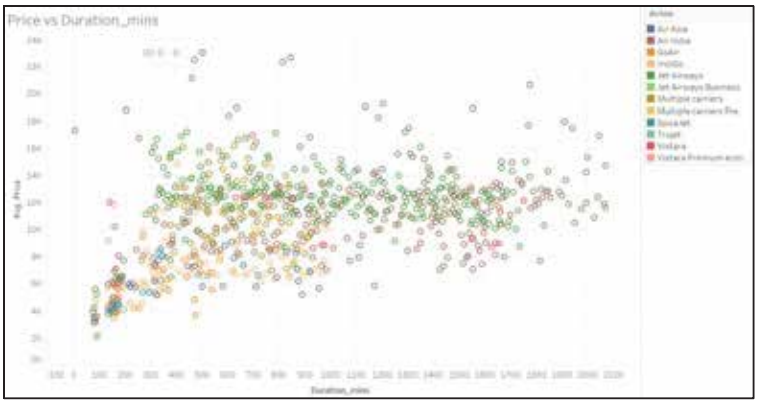
Flight price keeps fluctuating during different times of the day.
Highest price recorded at 3 am on average
Lowest price recorded at 1 am on average`

(iv) Price Vs Additional_Info



Passengers on Business class pay highest price for Flight tickets
Passengers pay lowest fares when they travel with no check-in baggages

(v) Price Vs Duration_mins



On Average, the flight prices are cheapest for journeys that last less than 100 minutes and they gradually start to increase with increase in duration.
However, it also depends on the Ticket Class of travel – A business Class travel costs more than an Economy class ticket for the same duration.

2.5 EDA – Summary:

→ Based on average flight price**, Trujet and Spicejet airlines' flights tickets are the cheapest** compared to other carriers.

→ The Average Price of a **non-stop flight is the cheapest** compared to journeys that include connecting flights from different stops.

→ On average, **the flight price shows increasing trend during peak times** of the day. The average price remains above Rs.8000 between 9 am to 9 pm and gradually drops to below Rs.5000 during early hours of day

→ The Flight prices are the cheapest for journeys with lesser durations and gradually increases with increase in duration.

→ It is also worth noting that on average, flight prices are the cheapest for a ticket with no check-in baggage.

3. DATA CLEANING AND PRE-PROCESSING:

3.1 Missing Value Treatment:

The missing values in given dataset are treated as follows:

❖ Flight_data:

✓ Identification:

```
Global Environment
Data
flight_data              10682 obs. of 13 variables
flight_data_test_new     2671 obs. of 12 variables
```

```
> anyNA(flight_data)
[1] TRUE
> sum(is.na(flight_data))
[1] 2
> colSums(is.na(flight_data))
        Airline Date_of_Journey          Source     Destination           Route        Dep_Time    Arrival_Time
              0               0               0               0               1               0               0
       Duration     Total_Stops Additional_Info           Price
              0               1               0               0
```

- ✓ There are 2 missing values in the given dataset
- ✓ They are located in the same row in “Route” and “Total_Stops” variable

▪ Treatment:

```
> #Missing value Treatment - Train
> flight_data = na.omit(flight_data)
> #Missing value Identification - Train
> anyNA(flight_data)
[1] FALSE
```

- ✓ The missing values are eliminated using na.omit() function
- ✓ The total entries have reduced from 10683 to 10682

❖ Flight_data_test_new:

✓ Identification:

```
Global Environment
Data
flight_data              10682 obs. of 13 variables
flight_data_test_new     2671 obs. of 12 variables
```

```
> #Missing value Identification - Test
> anyNA(flight_data_test)
[1] FALSE
```

No missing values identified in Test set

3.2 Addition of new variables

(i) dep.hours; dep.mins; arr.hours; arr.mins; Duration_mins:

```
#Addition of variables - train set
flight_data = transform(flight_data, dep = colsplit(flight_data$Dep_Time, split = "\\:", names = c('hours','mins')))
flight_data$Arrival_Time = as.POSIXct(flight_data$Arrival_Time, format = "%H:%M") %>% format("%H:%M")
flight_data = transform(flight_data, arr = colsplit(flight_data$Arrival_Time, split = "\\:", names = c('hours','mins')))
flight_data$Duration_mins = as.numeric(period(toupper(flight_data$Duration)), "minutes")
```

→ Dept_Time and Arrival_Time in both Train & Test set are split into additional rows of Hours and Minutes

→ Duration variable in both Train & test sets are converted into minutes in a new variable “Duration_mins

(i) Day; Wknd.Wkday; Peak.normalhrs; stops.count:

```
flight_data = transform(flight_data, stops = colsplit(flight_data$Total_Stops, split = "\\ ", names = c('count','dummy')))
flight_data$stops.count = ifelse(flight_data$stops.count == "non-stop","0",flight_data$stops.count)
flight_data$Day = weekdays(as.Date(flight_data$Date_of_Journey,"%d/%m/%Y"))
flight_data$wknd.wkday = ifelse(flight_data$Day == "Sunday"|flight_data$Day == "Saturday","Weekend","Weekday")
flight_data$peak.normalhrs = ifelse(flight_data$dep.hours>=9 &flight_data$dep.hours<=21,"Peak_Hour","Normal_hour")
```

→ These second set of variables are created to be used in development of hypothgesis testing.

3.3 Variable transformation:

Since the datatypes of most of the variables in the given dataset are identified to be **"Character"**, it is necessary to transform these variables into their appropriate datatypes as follows:

```
#Variable Transformation
flight_data$Airline = as.factor(flight_data$Airline)
flight_data$Source = as.factor(flight_data$Source)
flight_data$Destination = as.factor(flight_data$Destination)
flight_data$Route = as.factor(flight_data$Route)
flight_data$Total_Stops = as.factor(flight_data$Total_Stops)
flight_data$Additional_Info = as.factor(flight_data$Additional_Info)
flight_data$Date_of_Journey = as.Date(flight_data$Date_of_Journey)
flight_data$stops.count = as.numeric(flight_data$stops.count)
flight_data$Day = as.factor(flight_data$Day)
flight_data$wknd.wkday = as.factor(flight_data$wknd.wkday)
flight_data$peak.normalhrs = as.factor(flight_data$peak.normalhrs)
```

3.4 Removal of Unwanted variables:

Since the variables like "Dep_Time", "Arrival_Time", "Duration" and "Total_stops" are being converted to separate additional columns, they are not to be used in modelling algorithms and are therefore unwanted variables. Such variables are eliminated from dataset as follows:

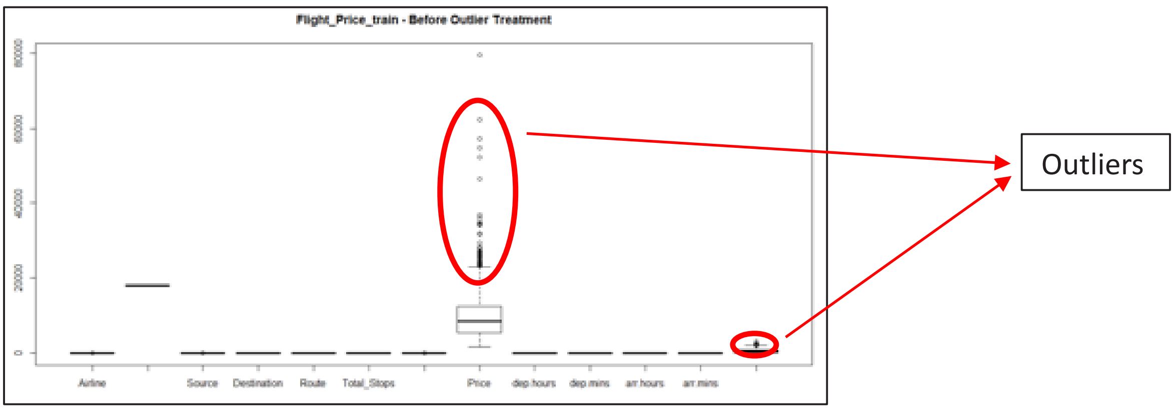
```
> #Removal of variables - train set
> flight_data_cleaned = flight_data[,-c(6:8)]
```

Note:

Steps – 3.2, 3.3 and 3.4 are repeated for **Flight_data_test_new** dataset also on which the best model algorithm is to be applied.

3.5 Outlier Treatment:

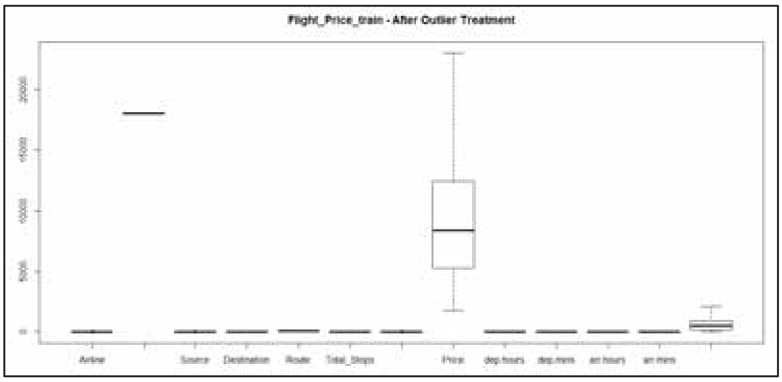
The outliers in the given dataset are identified using "Box plot" as follows:



✓ There are outliers identified in "Price" and "Duration_mins" variables.

- Treatment:

The outliers that are identified are treated using **“Capping method”,** by capping these outliers to the highest value of the corresponding variable as follows:



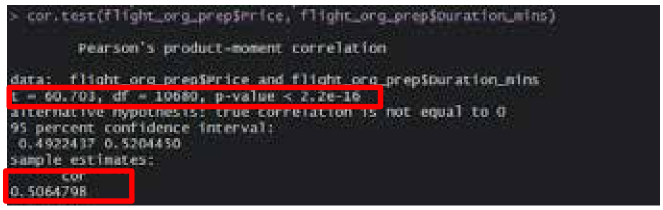
No Outliers detected after treatment using “Capping method”

Note: The outliers in the **Flight_data_test_new** are treated using the same procedure

4. INTERPRETATIONS FROM DATA:

4.1 Relationship between Flight time & price:

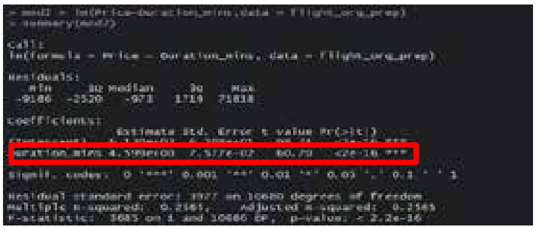
4.1.1 Correlation test:



Interpretation: Correlation between Flight Price and Duration of journey = 0.5064 ; p-value = 2.2 e^-16.

Hence, a **very strong positive correlation exists between Flight price and Duration**.

4.1.2 Using Linear Regression:



```
> mod2 = lm(Price~Duration_mins,data = flight_org_prep)
> summary(mod2)

Call:
lm(formula = Price ~ Duration_mins, data = flight_org_prep)

Residuals:
   Min     1Q Median     3Q    Max
 -9186  -2520   -973   1719  71838

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
Duration_mins 4.599e+00  7.577e-02   60.70   <2e-16 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3977 on 10680 degrees of freedom
Multiple R-squared:  0.2565,	Adjusted R-squared:  0.2565
F-statistic:  3685 on 1 and 10680 DF,  p-value: < 2.2e-16
```

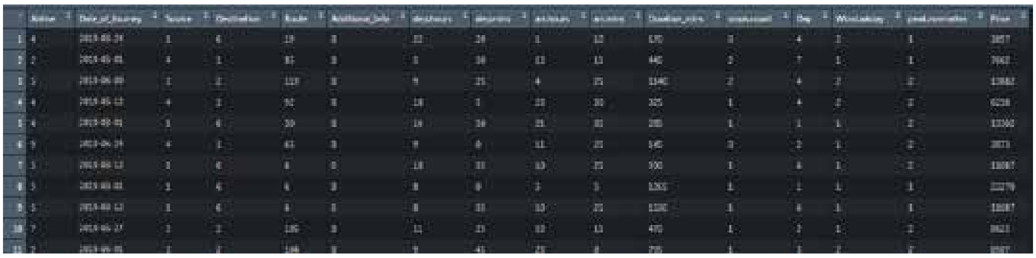
Interpretation: Co-efficient estimate = **4.599** & Adj. R-squared = **0.2565**

That is, **increase in duration by 1 minute, increases the flight price by Rs. 4.599.**

4.2 Identification of significant Independent variables:

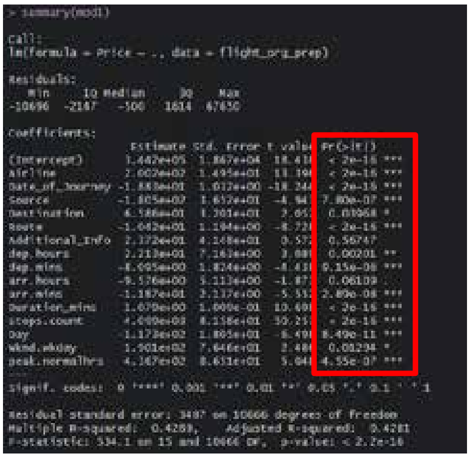
4.2.1 One-hot coding:

The categorical variables present in given dataset are assigned integer values in order to check for significance of variables using "One-hot encoding" as follows:



4.2.2 Linear Regression to Linear Regression to identify significant variables:

To identify the most significant independent variables that influence the Flight price, linear regression model is built and the output is obtained as follows:



```
> summary(mod1)

Call:
lm(formula = Price ~ ., data = flight_org_prep)

Residuals:
   Min     1Q Median     3Q    Max
-10696  -2147   -500   1614  47650

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      1.442e+05  1.867e+04  18.41   < 2e-16 ***
Airline          2.002e+02  1.495e+01  13.39   < 2e-16 ***
Date_of_Journey -1.883e+01  1.017e+00 -18.24   < 2e-16 ***
Source          -1.805e+02  1.652e+01  -4.94  7.80e-07 ***
Destination      6.586e+01  1.301e+01   2.05   0.03968 *
Route           -1.042e+01  1.194e+00  -8.72   < 2e-16 ***
Additional_Info  2.372e+01  4.148e+01   0.57   0.56747
dep.hours        2.213e+01  7.163e+00   3.08   0.00201 **
dep.mins        -8.095e+00  1.824e+00  -4.43  9.15e-06 ***
arr.hours       -9.576e+00  5.113e+00  -1.87   0.06109 .
arr.mins        -1.187e+01  2.137e+00  -5.55  2.89e-08 ***
Duration_mins    1.070e+00  1.000e-01  10.69   < 2e-16 ***
stops.count      4.000e+03  8.138e+01  30.21   < 2e-16 ***
Day             -1.173e+02  1.805e+01  -6.49  8.49e-11 ***
wknd.wkday       1.901e+02  7.646e+01   2.48   0.01294 *
peak.normalhrs   4.167e+02  8.631e+01   5.04  4.55e-07 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3487 on 10666 degrees of freedom
Multiple R-squared:  0.4289,	Adjusted R-squared:  0.4281
F-statistic: 534.1 on 15 and 10666 DF,  p-value: < 2.2e-16
```

Interpretation:

Except "Additional Info" and "Arr.hours", all other variables are significant at 5% significance level.

4.3 Development of Hypothesis Testing:

4.3.1 Flight Prices on Weekdays are cheaper than flight prices on weekends:

❖ **Bartlett Test:**

To identify if the variances between the Prices on Weekends and Weekdays are Equal or not.

Null Hypothesis (Ho) = variances are equal.

Alternate Hypothesis (H_1) = Variances are not equal

Output:

```
        Bartlett test of homogeneity of variances

data:  flight_data_org$Price and flight_data_org$Wknd.wkday
Bartlett's K-squared = 66.974, df = 1, p-value = 2.751e-16
```

p-value = **2.75 e $^{-16}$** < 0. 05 -- > **Null Hypothesis (Ho) is rejected.**

Thus, **variances are not equal.**

❖ **Two-sample Hypothesis test:**

Null Hypothesis (Ho) = Flight prices on weekdays are not cheaper than on weekends

Alternate Hypothesis (H_1) = Flight Prices on Weekdays are cheaper than flight prices on weekends.

```
> t.test(Price~Wknd.wkday)

        Welch Two Sample t-test

data:  Price by Wknd.wkday
t = -2.2059, df = 6620, p-value = 0.02742
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -387.45611  -22.84045
sample estimates:
mean in group Weekday mean in group Weekend
             9026.834              9231.982
```

Mean Weekday price = Rs. 9026.834 ; Mean Weekend price = Rs. 9231.982

p-value = **0.0274** < 0. 05 -- > **Null Hypothesis (Ho) is rejected.**

Thus, its statistically evident that **Flight Prices on Weekdays are cheaper than flight prices on weekends** on average.

Business Recommendation:

Since it has been statistically proven that **Flight Prices on Weekdays are cheaper than flight prices on weekends** and **there is a significant difference of Rs. 205.148**, it is advisable for the passengers/tourist agencies to plan their journeys on a weekday, in order to save cost on flight tickets.

4.3.2 Flight Prices during peak hours are costlier than flights at other times:

❖ **Bartlett Test:**

To identify if the variances between the Prices of Peak hours and Normal hours are Equal or not.

Null Hypothesis (Ho) = variances are equal.

Alternate Hypothesis (H_1) = Variances are not equal

Output:

```
> bartlett.test(flight_data_org$Price,flight_data_org$peak.normalhrs)

        Bartlett test of homogeneity of variances

data:  flight_data_org$Price and flight_data_org$peak.normalhrs
Bartlett's K-squared = 39.94, df = 1, p-value = 2.618e-10
```

p-value = **2.62 e $^{-10}$** < 0. 05 -- > **Null Hypothesis (Ho) is rejected.**

Thus, **variances are not equal.**

❖ **Two-sample Hypothesis test:**

Null Hypothesis (Ho) = Flight Prices during peak hours are not costlier than flights at other times

Alternate Hypothesis (H_1) = Flight Prices during peak hours are costlier than flights at other times

```
> t.test(Price~peak.normalhrs)

        Welch Two Sample t-test

data:  Price by peak.normalhrs
t = -4.3889, df = 7528.6, p-value = 1.155e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -602.4678 -230.4516
sample estimates:
mean in group Normal_hour   mean in group Peak_Hour
                 8822.531                  9238.991
```

Mean price in Normal Hour = Rs. 8822.531 ; Mean price in Peak Hour = Rs. 9238.991

p-value = **1.1.55e-5** < 0. 05 -- > **Null Hypothesis (Ho) is rejected**

Thus, its statistically evident that **Flight Prices during peak hours are costlier than flights at other times** on average.

Business Recommendation:

Since it has been statistically proven that **Flight Prices during peak hours (9am to 9pm) are costlier than flights at other times** and **there is a significant difference of Rs. 416.46**, it is recommended to avoid travels during peak hours.

5. BUILDING PREDICTIVE MODELS:

5.1 Assumptions for Model building:

The following assumptions are considered before building predictive modelling algorithms:

a) Normal distribution of errors:

The errors are all assumed to be **normally distributed** above and below the best fit line of the model equation, that is, **mean of error terms = 0**.

b) Homoscedasticity of errors:

A **constant variance or Homoscedasticity** is assumed to exist among the error terms which ensures equal importance for all the data points.

c) No auto-correlation of errors:

When error terms are correlated to each other, it is called **auto-correlation**. It is assumed that there is **no auto-correlation among error terms**.

d) X variables are statistically independent of Error terms:

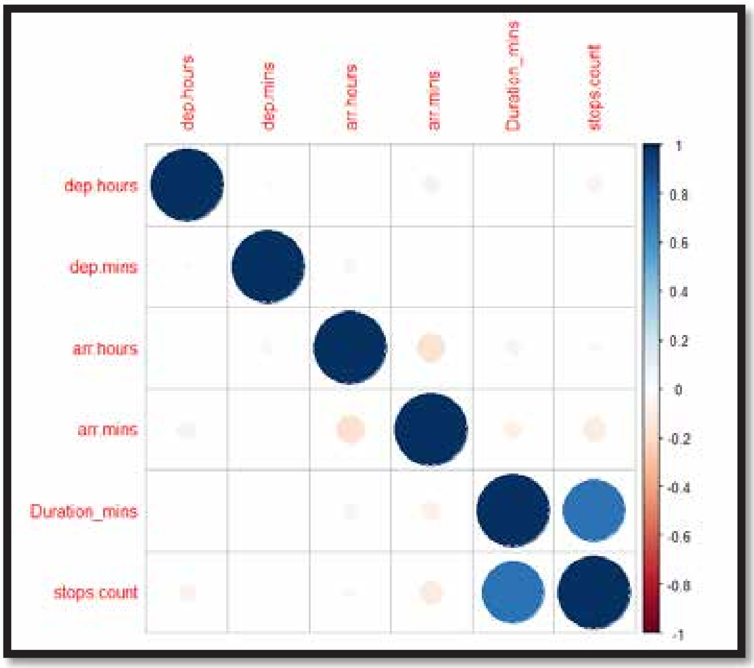
The X- variables (independent variables) are assumed to be **statistically independent of error terms**, that is, any changes to X- variables doesn't affect the error terms and vice versa.

e) Model specification:

Model is assumed to be correctly specified without any over-fitting or under-fitting to be existing in the model.

5.2 Test for Multi-collinearity:

Correlation-plot



Interpretation:

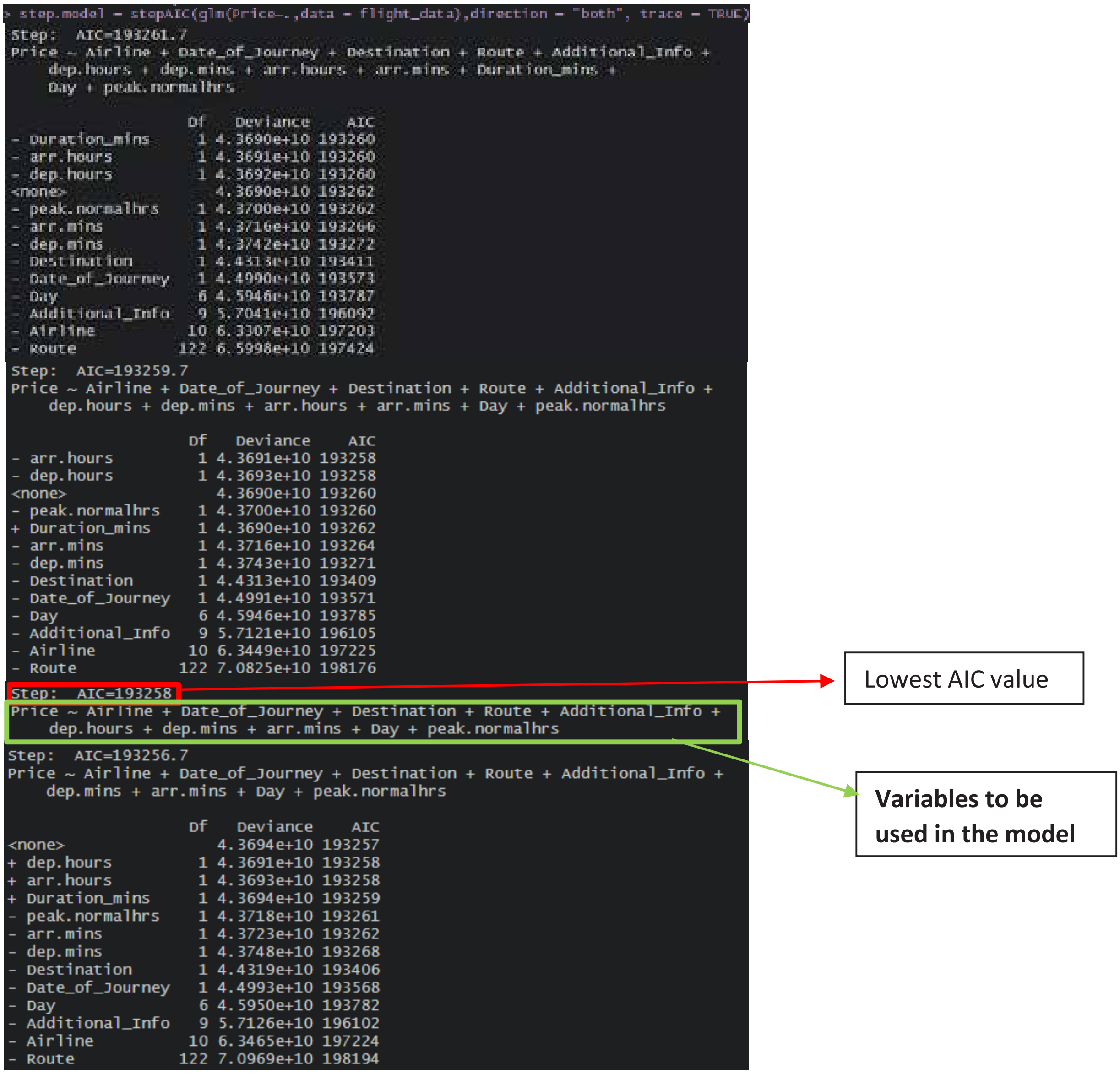
High positive correlation -- > "Duration_mins" and "stops.count" & "dep.hours and "arr.mins"

High negative correlation -- > "arr.hours and "arr.mins" ; "arr.mins and "Duration_mins" & "arr.mins" and "stops.count".

Since there are correlations present between the independent variables, **multi-collinearity exists in the given dataset.**

5.3 Multi-collinearity treatment / Feature Identification:

The variables that are devoid of multi-collinearity and to be used in predictive models are identified after numerous iterations based on their **lower AIC values** using **Step-wise Regression** as follows:



```
> step.model = stepAIC(glm(Price~.,data = flight_data),direction = "both", trace = TRUE)
Step:  AIC=193261.7
Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.hours + dep.mins + arr.hours + arr.mins + Duration_mins +
    Day + peak.normalhrs

                   Df   Deviance    AIC
- Duration_mins     1 4.3690e+10 193260
- arr.hours         1 4.3691e+10 193260
- dep.hours         1 4.3692e+10 193260
<none>                4.3690e+10 193262
- peak.normalhrs    1 4.3700e+10 193262
- arr.mins          1 4.3716e+10 193266
- dep.mins          1 4.3742e+10 193272
- Destination       1 4.4313e+10 193411
- Date_of_Journey   1 4.4990e+10 193573
- Day               6 4.5946e+10 193787
- Additional_Info   9 5.7041e+10 196092
- Airline          10 6.3307e+10 197203
- Route           122 6.5998e+10 197424

Step:  AIC=193259.7
Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.hours + dep.mins + arr.hours + arr.mins + Day + peak.normalhrs

                   Df   Deviance    AIC
- arr.hours         1 4.3691e+10 193258
- dep.hours         1 4.3693e+10 193258
<none>                4.3690e+10 193260
- peak.normalhrs    1 4.3700e+10 193260
+ Duration_mins     1 4.3690e+10 193262
- arr.mins          1 4.3716e+10 193264
- dep.mins          1 4.3743e+10 193271
- Destination       1 4.4313e+10 193409
- Date_of_Journey   1 4.4991e+10 193571
- Day               6 4.5946e+10 193785
- Additional_Info   9 5.7121e+10 196105
- Airline          10 6.3449e+10 197225
- Route           122 7.0825e+10 198176

Step:  AIC=193258
Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.hours + dep.mins + arr.mins + Day + peak.normalhrs

Step:  AIC=193256.7
Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.mins + arr.mins + Day + peak.normalhrs

                   Df   Deviance    AIC
<none>                4.3694e+10 193257
+ dep.hours         1 4.3691e+10 193258
+ arr.hours         1 4.3693e+10 193258
+ Duration_mins     1 4.3694e+10 193259
- peak.normalhrs    1 4.3718e+10 193261
- arr.mins          1 4.3723e+10 193262
- dep.mins          1 4.3748e+10 193268
- Destination       1 4.4319e+10 193406
- Date_of_Journey   1 4.4993e+10 193568
- Day               6 4.5950e+10 193782
- Additional_Info   9 5.7126e+10 196102
- Airline          10 6.3465e+10 197224
- Route           122 7.0969e+10 198194
```


Hence, the set of variables that gives the lowest AIC values are chosen to be included in the predictive modelling algorithms.

❖ Splitting dataset:

The given dataset is split into two subsets -- > **Train and test sets**, so that the predictive algorithms are trained using the Training dataset (*flight_data.train*) and are validated on the Test dataset (*flight_data.test*) for its performance:

OUTPUT:

```
flight_data.test    3205 obs. of 18 variables
flight_data.train   7477 obs. of 18 variables
```

5.4 Model Algorithms:

5.4.1 Linear Regression:

❖ Reasons for Choosing model:

- ✓ Target variable (Price) is a continuous variable, thus Linear regression model algorithm is best suited for prediction of continuous variables.
- ✓ The Multiple linear regression generally uses **Ordinary Least Square** approach to arrive at the "**best fit**" line by minimizing the **Residual Sum of Squares (RSS)** thus improving the predictive power of algorithm.
- ✓ The multiple linear regression model not only provides the strength of linear relationship between Predictors and Response variables but also quantifies the relationship using the Slope (β-values).

❖ Output:

Although the variables to be used in the model are identified using Step-wise regression method, different combinations of variables are tried with four different linear regression models to identify the best amongst them and the output of the best amongst them is as follows:

```
> cv_model1
Linear Regression

7477 samples
  10 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 6728, 6728, 6729, 6731, 6731, 6728, ...
Resampling results:

  RMSE      Rsquared   MAE
  2098.083  0.7598658  1499.802

Tuning parameter 'intercept' was held constant at a value of TRUE
```

Interpretation: This model was performed on train set with 10 independent variables.

RMSE = 2098.083 -- > Average RMSE across 10 CV-folds Indicates that on average, the predictions made by the model are Rs.2098.083 off from the actual Flight price.

R-sq = 0.7598 -- > Indicates that this model is able to explain about 75.98% of the variations in Flight price through all the independent variables used.

[Note: For evaluation of all four models of linear regression and choosing best one, refer Fig.1 in appendix]

5.4.2 KNN-Algorithm:

❖ **Reasons for Choosing model:**

- ✓ Since the target variable (Price) and few of the Predictor variables are continuous variables, it is possible to use KNN-algorithm, which is a **distance based algorithm**.
- ✓ A very simple, but effective algorithm in which each observation is predicted based on its **“similarity”** to other observations.

❖ **Output:**

A cross-validated KNN algorithm, with k-value ranging from 2 to 10 are tested and the model with best k-value is chosen based on the least RMSE value, the output of which is shown below:

```
> knn_fit
k-Nearest Neighbors

7477 samples
  10 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 6730, 6729, 6729, 6730, 6729, 6729, ...
Resampling results across tuning parameters:

  k   RMSE      Rsquared   MAE
   2  3413.246  0.4109088  2463.006
   3  3322.018  0.4213250  2473.639
   4  3268.861  0.4287652  2485.135
   5  3243.907  0.4325355  2493.880
   6  3221.577  0.4371615  2492.518
   7  3218.280  0.4367800  2505.547
   8  3217.141  0.4362246  2519.288
   9  3228.941  0.4317594  2542.946
  10  3241.471  0.4271933  2563.528
  11  3254.925  0.4225525  2582.838
  12  3263.077  0.4199008  2596.980
  13  3273.596  0.4164142  2611.640
  14  3286.598  0.4118930  2628.554
  15  3295.788  0.4089015  2640.748

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was k = 8.
```

Interpretation:

From the above output, **model with k = 8 is found to be the optimal performing model based on RMSE value.**

RMSE = 3217.141 --> Indicates that on average, the predictions made by the model are Rs. **3217.141** off from the actual Flight price.

R-sq = 0.4362 --> Indicates that this model is able to explain about **43.62%** of the variations in Flight price through all the independent variables used.

5.4.3 Decision trees:

❖ **Reasons for Choosing model:**

- ✓ A CART-based algorithm works well in improving the predictive power of continuous variables. Since the target variable in this case is also a continuous variable, it is possible to use CART based Decision tree algorithm for prediction.
- ✓ There are also numerous Predictor variables that are categorical that can be easily handled by CART algorithms without any pre-processing requirements.
- ✓ A well pruned Decision tree can also further improve the predictive performance of CART algorithm.

❖ **Output:**

The initial decision tree was built using default parameters to identify the optimum CP value to prune the tree. The output of Pruned Decision tree is shown below:

```
> Flight_price__dt2
CART

7477 samples
  10 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 6728, 6730, 6729, 6729, 6730, 6729, ...
Resampling results across tuning parameters:

  cp           RMSE      Rsquared   MAE
  0.002754593  1926.261  0.7974005  1338.727
  0.002865704  1942.660  0.7939820  1357.072
  0.002889322  1943.421  0.7938149  1358.116
  0.003604796  1999.616  0.7819276  1390.454
  0.003882663  2003.188  0.7811348  1395.388
  0.004461982  2035.276  0.7740826  1428.259
  0.004797043  2059.518  0.7686439  1465.041
  0.006114937  2101.407  0.7591383  1513.489
  0.006308244  2132.686  0.7519875  1541.754
  0.006454503  2151.796  0.7475158  1551.977
  0.006637239  2172.776  0.7424760  1567.011
  0.008497348  2204.518  0.7350757  1602.393
  0.016228300  2254.319  0.7229134  1632.742
  0.022217508  2318.207  0.7068075  1667.760
  0.024326289  2404.667  0.6845306  1711.012
  0.035787058  2551.646  0.6448910  1830.258
  0.055081952  2672.966  0.6096759  1967.149
  0.078354059  2944.977  0.5260429  2249.631
  0.088421038  3135.948  0.4631518  2410.381
  0.415525117  4000.771  0.3792532  3291.290

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was cp = 0.002754593.
```

Interpretation:

The pruned tree model has identified the optimum model with best **CP value = 0.0027** based on least RMSE value through 10-fold CV.

For this optimum model,

RMSE = 1926.261 -- > Indicates that on average, the predictions made by the model are **Rs. 1926.261** off from the actual Flight price.

R-sq = 0.7974 -- > Indicates that this model is able to explain about **79.74%** of the variations in Flight price through all the independent variables used.

[Note: For identification of CP value, refer Fig.2 in appendix]

5.4.4 Random Forests:

❖ Reasons for Choosing model:

- ✓ Random Forests model provides high predictive accuracy as the algorithm works by introducing more randomness into the basic decision tree model.
- ✓ It also ensures highly reduced instability of model as well as correlation between different Decision trees with considerably higher computational speed.
- ✓ It also gives out-of-the-box performance with least variability in prediction.

❖ Output:

The initial Random Forest model is built using default parameters, after which the optimum “mtry” and “ntree” values are identified and applied in Final model, whose output is shown below:

```
> flight_rf_fit_final

Call:
 randomForest(formula = Price ~ Airline + Date_of_Journey + Destination +      Additional_Info + dep.mins +
 arr.mins + stops.count + Day +      peak.normalhrs, data = flight_data.train, ntree = 101, mtry = 9,
 nodesize = 10, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 101
No. of variables tried at each split: 9

          Mean of squared residuals: 1886230
                    % Var explained: 89.69
```

Interpretation:

MSE = 1886230 → RMSE = 1373.400 -- > Indicates that on average, the predictions made by the model are **Rs. 1373.4** off from the actual Flight price.

R-sq = 0.8969 -- > Indicates that this model is able to explain about **89.69%** of the variations in Flight price in the dataset.

Thus, the **final RF model performs better than the initial RF model**, as evident from its better RMSE and R-sq values.

[Note: For identification of ntree and mtry values, refer Fig.3 & Fig.4 respectively in appendix]

5.4.5 Ensemble Models:

5.4.5.1. Bagging:

❖ **Reasons for Choosing model:**

- ✓ The bagging algorithm is designed to improve the stability and accuracy of prediction in regression problems using a “Bootstrap aggregating” approach that creates bootstrap samples (subsets) from original data with replacement, equivalent to a cross-validated model.
- ✓ The aggregation process also helps reducing the variance in prediction which improves the model performance.

❖ **Output:**

```
> flight_price_bagging1

Bagging regression trees with 50 bootstrap replications

Call: bagging.data.frame(formula = Price ~ Airline + Date_of_Journey +
    Destination + Route + Additional_Info + dep.mins + arr.mins +
    stops.count + Day + peak.normalhrs, data = flight_data.train,
    nbagg = 50, coob = TRUE)

Out-of-bag estimate of root mean squared error:  1987.15
```

Interpretation:

RMSE = 1987.15

Thus from this model, it indicates that on average, the predictions made by the model are **Rs. 1987.15** off from the actual Flight price.

5.4.5.2. Gradient Boosting:

❖ **Reasons for Choosing model:**

- ✓ GBM algortithm is very flexible as it can be applied to both categorical and continuous variables.
- ✓ It requires no pre-processing and provide greater accuracy in prediction.
- ✓ The performance can be further improved by tuning the hyperparameters.

❖ Output:

```
> flight_price_gbm
gbm(formula = Price ~ Airline + Destination + Route + Additional_Info +
    dep.mins + arr.mins + stops.count + Day + peak.normalhrs,
    distribution = "gaussian", data = flight_data.train,
    n.trees = 10000, interaction.depth = 3, shrinkage = 0.001,
    verbose = FALSE, n.cores = NULL)
A gradient boosted model with gaussian loss function.
10000 iterations were performed.
There were 9 predictors of which 9 had non-zero influence.
> flight_gbm_RMSE
[1] 1907.075
```

Interpretation:

RMSE = 1907.075

Thus from this model, it indicates that on average, the predictions made by the model are **Rs. 1907.075** off from the actual Flight price.

5.4.5.3. XG Boosting:

❖ Reasons for Choosing model:

- ✓ XG Boost model prevents over-fitting by using a regularized approach in its algorithms.
- ✓ It is also computationally very fast since it employs parallel processing.
- ✓ It also performs cross-validation at each iteration by default, thereby improving the model performance.

❖ Output:

To build up XG boost ensemble model, it requires a matrix input for the features and the response to be a vector. This is achieved using "One-hot coding" method and output of final XG boost algorithm is shown below:

```
> flight_price_xgb
##### xgb.cv 10-folds
    iter train_rmse_mean train_rmse_std test_rmse_mean test_rmse_std
       1        9072.737       20.48139       9072.083      194.7606
       2        8255.683       19.21186       8255.332      189.2693
       3        7528.104       16.95833       7528.728      187.0628
       4        6881.005       15.44746       6883.910      181.5136
       5        6306.012       14.99226       6308.910      178.2016
---
    1419        1572.899       13.13335       1806.636      116.7131
    1420        1572.817       13.16026       1806.687      116.8051
    1421        1572.731       13.16522       1806.640      117.1837
    1422        1572.632       13.19504       1806.588      117.1921
    1423        1572.569       13.19049       1806.638      117.0775
Best iteration:
 iter train_rmse_mean train_rmse_std test_rmse_mean test_rmse_std
 1373        1577.092       13.17232       1806.111      116.2879
> min(flight_price_xgb$evaluation_log$test_rmse_mean)
[1] 1806.111
```

Interpretation :

After performing 10-fold CV, the algorithm has identified the best model with:

RMSE = 1806.111

Thus from this model, it indicates that on average, the predictions made by the model are **Rs. 1806.111** off from the actual Flight price.

5.5 Model Tuning:

The above models are tuned using their hyperparameters to improve their performance and accuracy as follows:

(i) Bagging – Tuned model:

❖ **Paramaters tuned:**

nbagg = 1000

minsplit = 2

cp = 0.0027

minbucket = 10

❖ **Output:**

```
> flight_price_bagging2

Bagging regression trees with 1000 bootstrap replications

Call: bagging.data.frame(formula = Price ~ Airline + Date_of_Journey +
    Destination + Route + Additional_Info + dep.mins + arr.mins +
    stops.count + Day + peak.normalhrs, data = flight_data.train,
    nbagg = 1000, coob = TRUE, control = rpart.control(minsplit = 2,
        cp = 0.0027, minbucket = 10))

Out-of-bag estimate of root mean squared error:  1721.15
```

Interpretation:

RMSE = 1721.15

➔ Thus tuned GBM model, indicates that on average, the predictions made by the model are**.** **Rs.1721.15** off from the actual Flight price.

➔ Difference in RMSE value between normal GBM and tuned GBM models = **266**

➔ Hence, there is a **13% improvement in prediction** with a tuned GBM model compared to normal one based on RMSE values.

(ii) Gradient Boosting – Tuned model:

❖ **Paramaters tuned:**

shrinkage <- c(0.001, 0.01, 0.1)

interaction.depth = c(1, 3, 5)

n.minobsinnode = c(5, 10, 15)

❖ **Output:**

```
> flight_price_gbm2
gbm(formula = Price ~ Airline + Destination + Route + Additional_Info +
    dep.mins + arr.mins + stops.count + Day + peak.normalhrs,
    distribution = "gaussian", data = flight_data.train,
    n.trees = 10000, interaction.depth = tuning_grid1$interaction.depth[i],
    n.minobsinnode = tuning_grid1$n.minobsinnode[i], shrinkage = tuning_grid1$Var1[i],
    verbose = FALSE, n.cores = NULL)
A gradient boosted model with gaussian loss function.
10000 iterations were performed.
There were 9 predictors of which 9 had non-zero influence.
> flight_gbm_RMSE2 = sqrt(min(flight_price_gbm2$train.error))
> flight_gbm_RMSE2
[1] 1421.868
```

Interpretation:

RMSE = 1421.868

➔ Thus tuned GBM model, indicates that on average, the predictions made by the model are **Rs.1421.868** off from the actual Flight price.

➔ Difference in RMSE value between normal GBM and tuned GBM models = **485.21**

➔ Hence, there is a **25% improvement in prediction** with a tuned GBM model compared to normal one based on RMSE values.

(iii) XG Boosting – Tuned model:

❖ **Paramaters tuned:**

learning_rate<- c(0.001, 0.01, 0.1)

max_depth <-c(1,3,5)

❖ **Output:**

```
> flight_price.xgb2
##### xgb.Booster
raw: 22 Mb
call:
  xgb.train(params = params, data = dtrain, nrounds = nrounds,
    watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
    early_stopping_rounds = early_stopping_rounds, maximize = maximize,
    save_period = save_period, save_name = save_name, xgb_model = xgb_model,
    callbacks = callbacks, objective = "reg:linear", eta = ..2,
    md = ..3, gamma = 5)
params (as set within xgb.train):
  objective = "reg:linear", eta = "0.1", md = "5", gamma = "5", silent = "1"
xgb.attributes:
  niter
callbacks:
  cb.evaluation.log()
# of features: 9
niter: 6000
nfeatures : 9
evaluation_log:
    iter train_rmse
       1   9042.845
       2   8195.658
---
    5999   1114.999
    6000   1114.999
> flight_xgb2_RMSE
[1] 1114.999
```

Interpretation:

RMSE = 1114.99

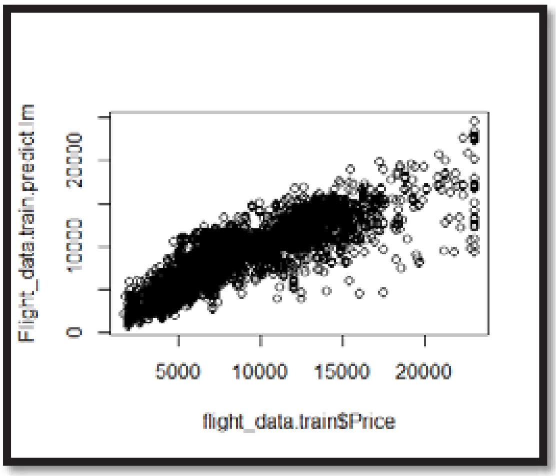
➔ Thus tuned xgb model, indicates that on average, the predictions made by the model are **Rs. 1114.99** off from the actual Flight price.

➔ Difference in RMSE value between normal xgb and tuned xgb models = **691.121**

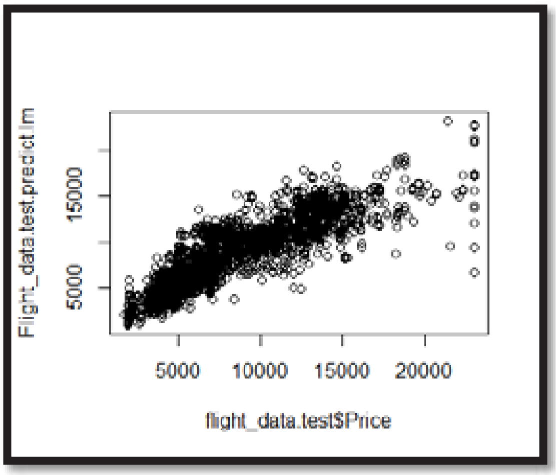
➔ Hence, there is a **38% improvement in prediction** with a tuned GBM model compared to normal one based on RMSE values.

6. PREDICTION BASED ON MODEL ALGORITHMS:

(i) Linear Regression:

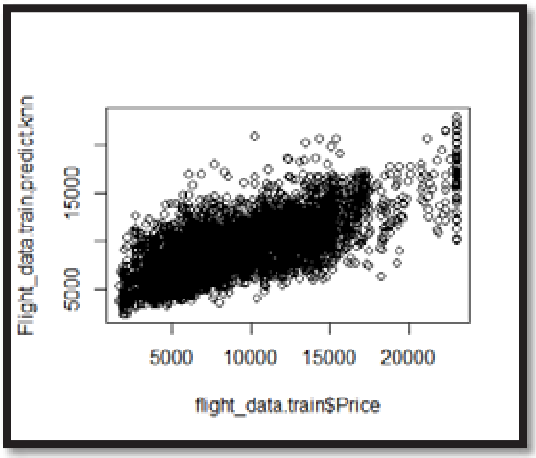


Train data

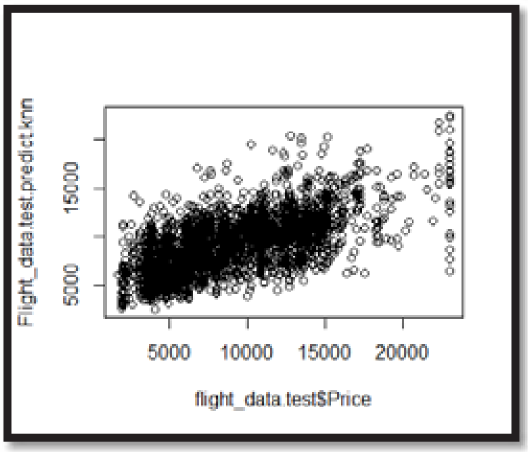


Test data

ii) KNN Algorithm

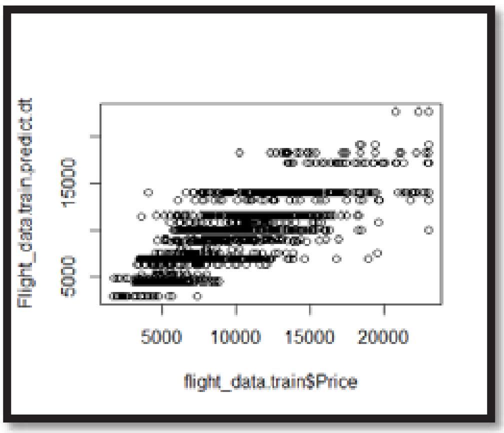


Train data

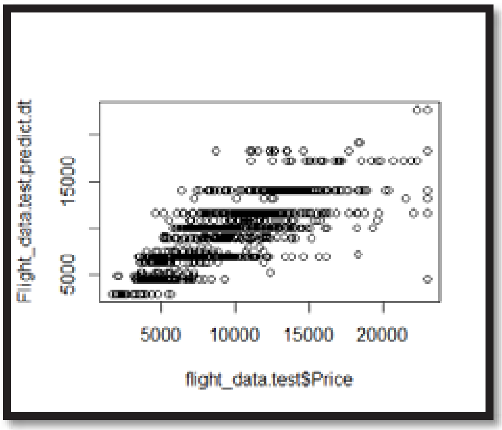


Test data

(iii) Decision tree:

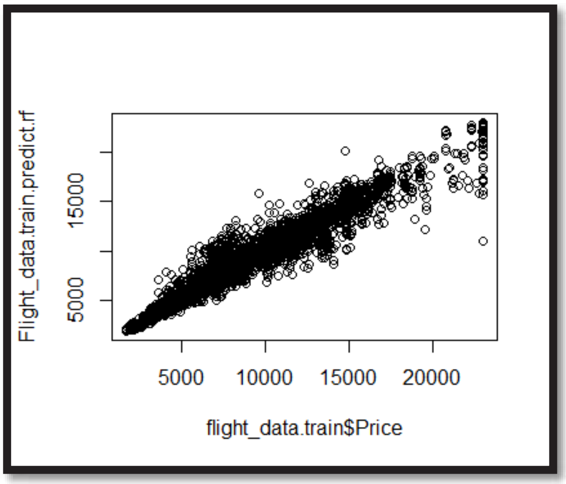


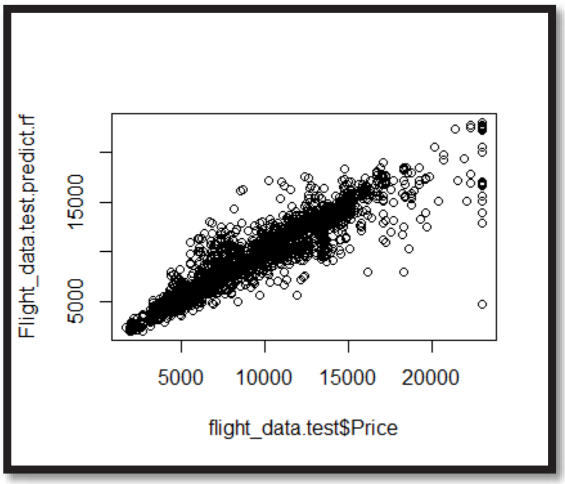
Train data



Test data

(iv) Random Forest:

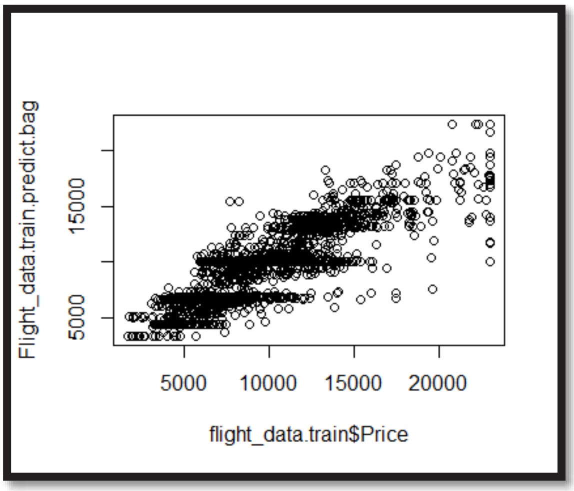


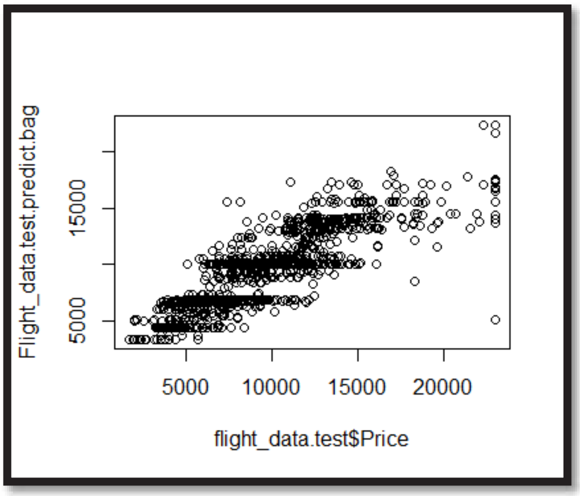


Train data

Test data

(v) Bagging:

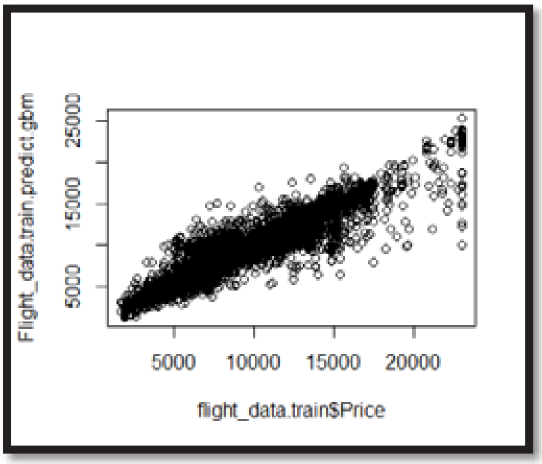


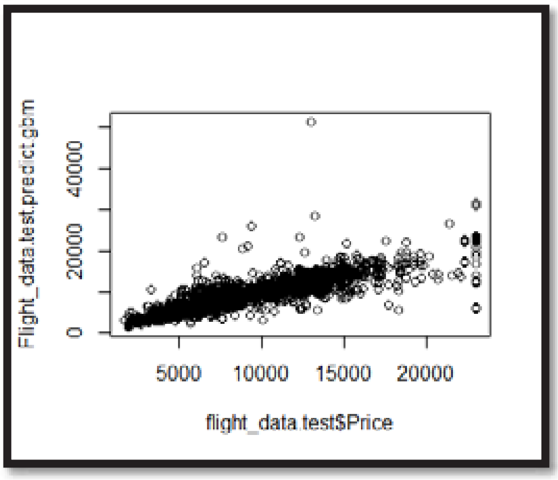


Train data

Test data

(vi) Gradient Boosting:

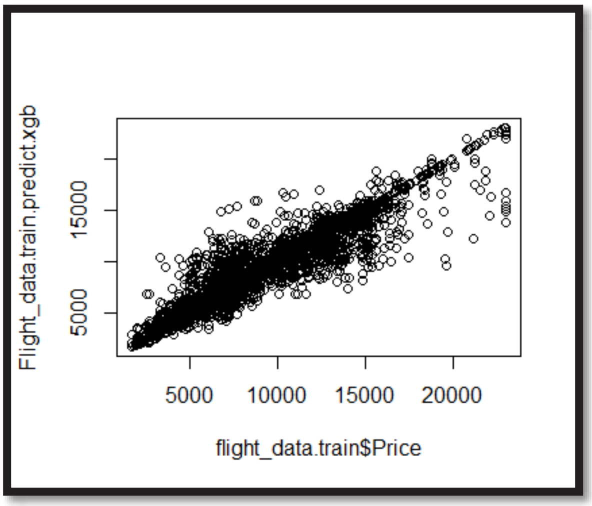




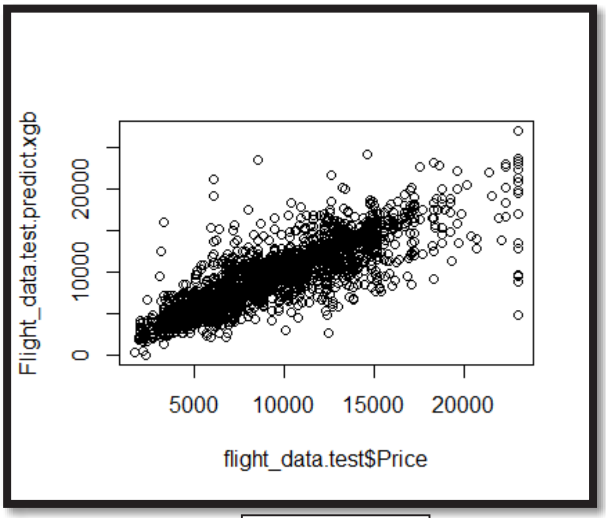
Train data

Test data

(vii) XG Boost:



Train data



Test data

Interpretation:

- The above graphs denote the datapoints of Flight prices plotted with **Actual prices on the X-axis** and **Predicted prices on the Y-axis**
- From these graphs, the variances are found to be very high in KNN, Decision tree and bagging algorithms. However, the variances look comparatively lesser on other predictive algorithms.

❖ Predicted values on test data (flight_data.test):

	flight_data.test.Price	flight_data.test.predict.lm	flight_data.test.predict.knn	flight_data.test.predict.dt	flight_data.test.predict.rf	flight_data.test.predict.bag	flight_data.test.predict.gbm	flight_data.test.predict.xgb
6	3873	3199	9658	4544	3874	4345	4150	3993
10	8625	10821	11439	10031	8409	10082	9747	8392
14	9663	9626	10995	10863	9465	10685	10218	10606
20	12898	10149	11591	10863	12893	10687	10319	10340
22	6955	7702	8453	6988	6638	8911	7201	6015
23	3943	4234	5760	4544	4163	4349	4153	3829
27	8238	7234	9813	6988	6931	6697	6234	6248
34	10919	10525	11295	10863	10871	10705	10772	10906
35	12373	10572	12168	10863	12348	10688	11109	9524
40	14924	13402	12348	14061	14814	14162	16576	17843
43	12373	10447	9193	10863	12358	10688	10972	10173
45	13062	11517	8680	10031	10146	10091	11070	10763
47	3943	3491	6087	4544	4163	4349	4155	3931
51	7202	7524	14114	4899	7420	6815	6699	6472
54	3943	3307	8341	4544	3878	4349	4096	3945

[Note: Only first 15 rows are shown above]

Interpretation:

The above image shows a table of the **actual price** (1st column) and the **predicted prices based on all 7 predictive algortihms** carried out on test data.

7. MODEL PERFORMANCE MEASURES:

The various model performance metrics for regression problems like R-squared, MAPE, RMSE, SSE, MAE, MSE are calculated for each model to identify the best performing model:

- **Output:**

```
> ##Linear Regression Model
> ###R-sq
> lm.train.rsq = cor(flight_data.train$Price, Flight_data.train.predict.lm)^2
> lm.train.rsq
[1] 0.7735187
> lm.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.lm)^2
> lm.test.rsq
[1] 0.7692499
> ###MAPE
> lm.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.lm)
> lm.train.mape
[1] 0.1811674
> lm.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.lm)
> lm.test.mape
[1] 0.1829262
> ###RMSE
> lm.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.lm)
> lm.train.rmse
[1] 2035.914
> lm.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.lm)
> lm.test.rmse
[1] 2025.86
> ###SSE
> lm.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.lm)
> lm.train.sse
[1] 30991761382
> lm.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.lm)
> lm.test.sse
[1] 13153662050
> ###MAE
> lm.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.lm)
> lm.train.mae
[1] 1457.593
> lm.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.lm)
> lm.test.mae
[1] 1454.664
> ###MSE
> lm.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.lm)
> lm.train.mse
[1] 4144946
> lm.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.lm)
> lm.test.mse
[1] 4104107
```

Interpretation:

The above output shows the various model performance metrics calculated for Linear Regression model and its corresponding values.

Note: The model performance metrics for other predictive algorithms are calculated in a similar way and the final output is presented in a tabulated form.

❖ Final output – Tabulation:

MEASURES		R-sq	MAPE	RMSE	SSE	MAE	MSE
Linear Regression	Test data	0.7692	0.1829	2025.86	13153662050	1454.664	4104107
KNN Algorithm	Test data	0.4448	0.3500	3147.44	31750020622	2434.767	9906403
Decision Trees	Test data	0.7806	0.1720	1976.19	12516602735	1387.406	3905336
Random Forest	Test data	0.8930	0.0887	1379.27	6097140493	770.91	1902384
Bagging	Test data	0.8328	0.1619	1728.34	9573840649	1238.69	2987158
Gradient Boosting	Test data	0.7804	0.1433	2001.40	12837894782	1181.84	4005583
XG Boosting	Test data	0.7935	0.1435	1940.01	12062471952	1184.61	3763642

Interpretation:

From the above table, based on RMSE value of test data:

"RANDOM FOREST" algorithm is found to be the best performing model (RMSE = 1379.27)

❖ Predicted values on new dataset (Flight_data_test_new):

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Airline	Date_of_Jo	Source	Destinatic	Route	Total_Stoj	Additional	dep.hours	dep.mins	arr.hours	arr.mins	Duration_	stops.cour	stops.dum	Day	Wknd.wkd	peak.nor	Predicted.Price	
2	Jet Airway	06-06-2019	Delhi	Cochin	CCU → DE	1 stop	In-flight m	17	30	4	25	655	1	stop	Thursday	Weekday	Peak_Ho	10316	
3	IndiGo	12-05-2019	Kolkata	Banglore	CCU → BC	1 stop	In-flight m	6	20	10	20	240	1	stop	Sunday	Weekend	Normal_	4572	
4	Jet Airway	21-05-2019	Delhi	Cochin	CCU → DE	1 stop	Business	19	15	19	0	1425	1	stop	Tuesday	Weekday	Peak_Ho	12880	
5	Multiple c	21-05-2019	Delhi	Cochin	CCU → DE	1 stop	In-flight m	8	0	21	0	780	1	stop	Tuesday	Weekday	Normal_	9753	
6	Air Asia	24-06-2019	Banglore	Delhi	BLR → CC	non-stop	In-flight m	23	55	2	45	170	0	non-stop	Monday	Weekday	Normal_	4516	
7	Jet Airway	12-06-2019	Delhi	Cochin	CCU → DE	1 stop	Business	18	15	12	35	1100	1	stop	Wednesda	Weekday	Peak_Ho	10272	
8	Air India	12-03-2019	Banglore	New Delhi	BLR → HY	1 stop	In-flight m	7	30	22	35	905	1	stop	Tuesday	Weekday	Normal_	11478	
9	IndiGo	01-05-2019	Kolkata	Banglore	BOM → U	1 stop	In-flight m	15	15	20	30	315	1	stop	Wednesda	Weekday	Peak_Ho	5622	
10	IndiGo	15-03-2019	Kolkata	Banglore	BOM → C	non-stop	In-flight m	10	10	12	55	165	0	non-stop	Friday	Weekday	Peak_Ho	4568	
11	Jet Airway	18-05-2019	Kolkata	Banglore	BOM → D	1 stop	In-flight m	16	30	22	35	365	1	stop	Saturday	Weekend	Peak_Ho	10487	
12	Jet Airway	21-03-2019	Delhi	Cochin	CCU → PA	2 stops	Business	13	55	18	50	1735	2	stops	Thursday	Weekday	Peak_Ho	8741	
13	IndiGo	15-06-2019	Delhi	Cochin	CCU → IXI	1 stop	In-flight m	6	50	16	10	560	1	stop	Saturday	Weekend	Normal_	5979	
14	Multiple c	15-05-2019	Delhi	Cochin	CCU → DE	1 stop	In-flight m	9	0	19	15	615	1	stop	Wednesda	Weekday	Peak_Ho	10130	
15	Jet Airway	12-03-2019	Banglore	New Delhi	BLR → BC	1 stop	In-flight m	5	45	10	25	280	1	stop	Tuesday	Weekday	Normal_	10993	

[Note: Only first 15 rows are shown above]

Interpretation:

Therefore, the best model algorithm (Random Forest) is applied on the new dataset *(Flight_data_test_new)* (for which the Flight Price is to be predicted) and is updated in the new "Predicted.price" column in the dataset.

(Note: Full excel output file attached along with the final report)

8. BUSINESS INTERPRETATIONS & RECOMMENDATIONS:

❖ Business Interpretation:

- **Based on EDA:**

✓ The end-users are highly recommended to check for flights from **, Trujet and Spicejet** Airlines first, before looking for other Flight carriers.

✓ It is recommended to the customers to book a **direct flight** to their destination to save cost of Flight tickets.

✓ The customers are advised to plan their departures during **early hours of the day** (12 am to 1 am) to avoid higher flight fares.
Note: The average flight price suddenly starts rising and reaches as high as above Rs.10000 at 3am.

✓ It is advisable for the end-users to also consider the duration of the journey along with Airlines and Stops at the time of booking and confirm their bookings for the flight with least duration of travel.

✓ Business delegates and other frequent travelers who don't have high volumes of luggages are recommended to avail this and look for Airlines that offer cheaper tickets based on **no check-in baggage** (eg: Spice jet)

- **<u>Based on Best Model (Random Forest):</u>**

.

✓ On comparing the RMSE values of test data of all predictive algorithms, **Random Forest** was found to perform the best among all other models

✓ The Random Forest model has an **RMSE value of 1379.27**, which implies that, the predictions made by this model on a new dataset is approximate to about Rs. 1379.27 from the actual Flight price.

✓ Based on 'Importance' metric of final Random Forest model:

```
> importance(flight_rf_fit_final)
                  %IncMSE IncNodePurity
Airline         223.72816   25387288946
Date_of_Journey 120.18547   23017226189
Destination      61.00713    4338545805
Additional_Info 125.48607    8393512872
dep.mins         53.65680    3629634388
arr.mins         49.89823    3626417710
stops.count     122.20801   59610646418
Day              21.87222    2377795846
peak.normalhrs   34.41649    1137602120
```

The top 3 most important variables that have highest influence on Flight price are:

Airlines

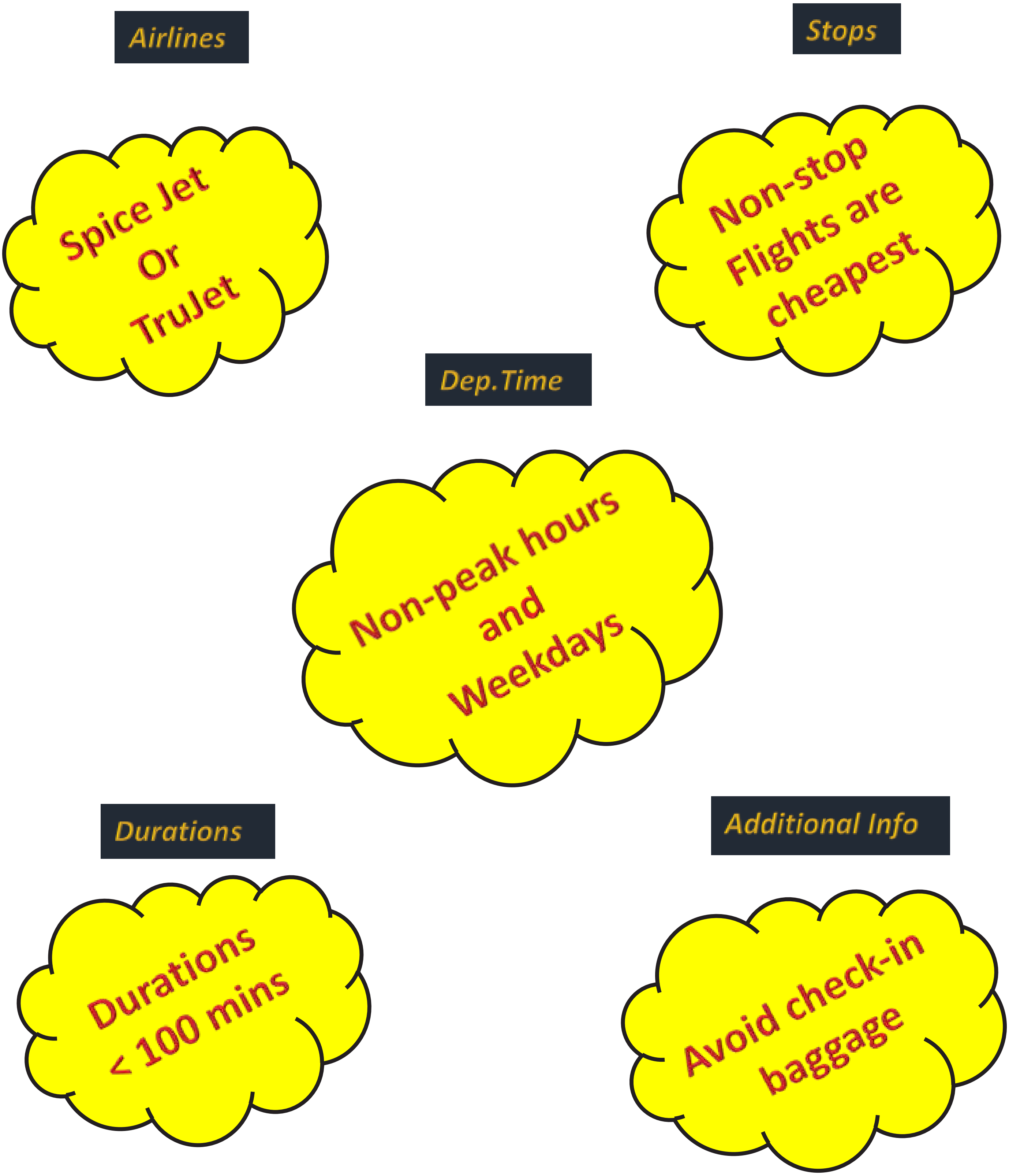
Additional_Info

Stops.count

❖ Business Recommendations:

- **For Individual Passengers / Travellers/ Business Delegates:**

  Assuming that the prime motive of a normal passengers or travelers would be to **book a cheapest flight ticket**, the following recommendations are made:



- **<u>For Travel & Tourism agencies / Holiday Planners:</u>**

The following recommendations to introduce new promotional offers are advisable to Clients from Tourism & Travel industry to attract more customers and leverage stable growth in their business:

FLY INDIGENOUS
&
FLY CHEAP

An exclusive scheme that provides 15% OFF for Tourists who choose **Spicejet / Trujet** Airlines.

MID-WEEK
MELA

A promotional offer of upto 10% cashback for Tourists who plan to depart on **weekdays and in non-peak hours**

FLY LITE
&
SAVE HEFTY

An exclusive offer of upto 8% that encourages tourists to travel without **check-in baggage** on holidays that are than 2 nights stay.

Appendix

❖ Packages and libraries:

readxl	-	To import excel dataset into R script
chron	-	To deal with “Date” variables
lubridate	-	To deal with “Date” variables
reshape	-	To transform variables into required datatype (character to integers etc.)
corrplot	-	To build correlation plot between numerical variables
MASS	-	To perform variable selection method to given dataset to identify those variables that are not multi-collinear.
recipes	-	To perform one-hot coding operation to transform categorical variables into integer values.
caret	-	To build KNN algorithm for predicting flight prices.
Ipred & rpart	-	To build Decision tree algorithm
rpart.plot	-	To visualize the output of decision trees model.
randomForest	-	To build Random Forest Algorithm
gbm	-	To build Gradient boosting algorithm
Metrics	-	To determine Model performance measures for each algorithm.

❖ Evaluation of linear Regression Models:

```
> #Model Accuracy
> summary(resamples(list(
+   model1 = cv_model1,
+   model2 = cv_model2,
+   model3 = cv_model3,
+   model4 = cv_model4
+ )))

Call:
summary.resamples(object = resamples(list(model1 = cv_model1, model2 = cv_model2, model3 =
 cv_model3, model4 = cv_model4)))

Models: model1, model2, model3, model4
Number of resamples: 10

MAE
           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
model1 1438.917 1472.816 1497.491 1499.802 1515.580 1574.464    0
model2 1438.788 1476.345 1498.913 1501.361 1512.788 1573.639    0
model3 1437.121 1477.480 1498.950 1501.108 1513.264 1573.599    0
model4 1441.080 1471.932 1496.620 1500.202 1515.629 1574.433    0

RMSE
           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
model1 1967.359 2028.080 2106.778 2098.083 2156.860 2247.545    0
model2 1970.022 2029.384 2108.669 2099.847 2161.856 2248.091    0
model3 1970.837 2030.234 2109.242 2099.426 2161.285 2247.106    0
model4 1966.457 2027.540 2106.859 2098.590 2158.110 2247.672    0

Rsquared
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
model1 0.7301639 0.7460101 0.7600223 0.7598658 0.7701086 0.7937541    0
model2 0.7301158 0.7464976 0.7595715 0.7594848 0.7697830 0.7932112    0
model3 0.7302994 0.7462715 0.7594460 0.7595805 0.7696107 0.7930220    0
model4 0.7301766 0.7459883 0.7600039 0.7597460 0.7701912 0.7939594    0
```

Fig 1

Interpretation:

As evident from above output, based on the four linear regression models,

Model 1 is found to have the **least MAE, RMSE and highest R-squared values** (mean values considered)

Thus, **Model 1 performs better** compared to other 3 models and is considered for comparison with other models

❖ Initial Decision tree model (with default parameters):

A decision tree algorithm is used to train the model with train set as follows:

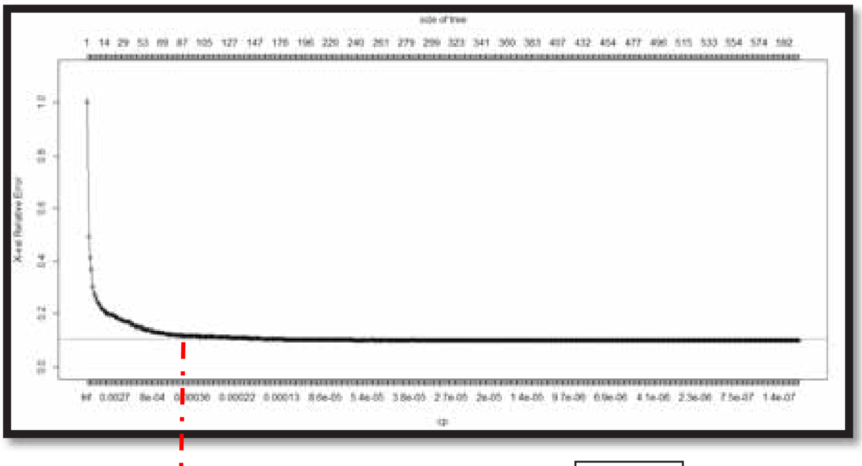
```
> printcp(Flight_price__dt)

Regression tree:
rpart(formula = Price ~ Airline + Date_of_Journey + Destination +
    Route + Additional_Info + dep.mins + arr.mins + stops.count +
    Day + peak.normalhrs, data = flight_data.train, method = "anova",
    control = list(cp = 0, xval = 10))

Variables actually used in tree construction:
 [1] Additional_Info Airline         arr.mins        Date_of_Journey Day             dep.mins
 [7] Destination     peak.normalhrs  Route           stops.count

Root node error: 1.3684e+11/7477 = 18301497

n= 7477

           CP nsplit rel error  xerror      xstd
1   5.1183e-01      0  1.000000 1.00025 0.0162541
2   8.3727e-02      1  0.488167 0.49289 0.0121922
3   4.1535e-02      2  0.404440 0.41348 0.0096386
4   3.2214e-02      3  0.362905 0.37021 0.0093773
5   2.7551e-02      5  0.298478 0.30458 0.0090058
6   1.9603e-02      6  0.270926 0.27635 0.0081172
7   1.5918e-02      7  0.251323 0.26326 0.0079491
8   1.0408e-02      8  0.235405 0.24814 0.0078535
9   8.9345e-03      9  0.224997 0.23923 0.0069912
10  8.5546e-03     10  0.216063 0.23042 0.0068105
```

[Note: Only first few rows are shown above]

Note:

The decision tree was plotteed using **rpart.plot()** function. However, it could not be included in this business report since,the number of trees was huge and it was not possible to display it here with clarity.

❖ CP v X-error:



CP = 0.00038

Fig 2

Thus from the above graph, the X-relative error seems to remain constant at CP = 0.00038. Thus, this CP value is used to prune the tree.

❖ Initial Random Forest model (with default parameters):

A Random forest algorithm is used to train the model with train set as follows:

```
> print(flight_rf_fit)

Call:
 randomForest(formula = Price ~ Airline + Date_of_Journey + Destination +      Additional_Info + dep.mins + ar
r.mins + stops.count + Day +      peak.normalhrs, data = flight_data.train, ntree = 501, mtry = 3,      nodesiz
e = 10, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 501
No. of variables tried at each split: 3

          Mean of squared residuals: 1997635
                    % Var explained: 89.08
```

Interpretation:

MSE = 1997635 → RMSE = 1413.377 -- > Indicates that on average, the predictions made by the model are **Rs. 1413.377** off from the actual Flight price.

R-sq = 0.8908 -- > Indicates that this model is able to explain about **89.08%** of the variations in Flight price in the dataset.

To find no. of trees:

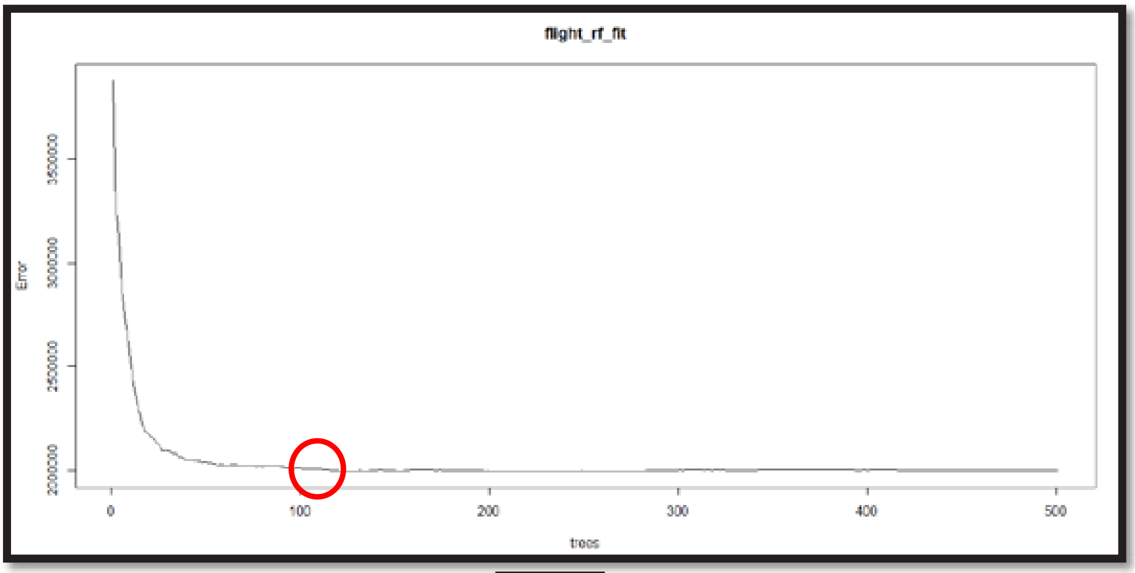


Fig 3

Thus from the above graph, the error is set to remain stable with **trees = 101**

Importance of variables:

```
> importance(flight_rf_fit)
                  %IncMSE IncNodePurity
Airline         184.00447   37193905276
Date_of_Journey 111.13875   19803334500
Destination      37.77139   13814548850
Additional_Info 115.99505    6936286732
dep.mins         58.99534    3533272966
arr.mins         62.23479    3763421187
stops.count      70.39696   36530820766
Day              47.90470    3962059981
peak.normalhrs   45.87064     926209405
```

Only three variables are found to be most important as evident from their higher values of %INC MSE.

To find mtry:

```
> flight_rf_fit.tuned = tuneRF(x = flight_data.train[,c(1,2,4,7,10,12,14,16,18)], y = flight_data.train$Price,
 mtrystart = 3, stepfactor = 1.5, nTreetry = 101,
+                              improve = 0.0001, nodesize = 10, plot = TRUE, doBest = TRUE, importance = TRUE)
mtry = 3  OOB error = 2038173
Searching left ...
mtry = 2        OOB error = 2444861
-0.1995358 1e-04
Searching right ...
mtry = 6        OOB error = 1941299
0.04752995 1e-04
mtry = 9        OOB error = 1929366
0.006147068 1e-04
```

The least OOB error is obtained for **mtry = 9**

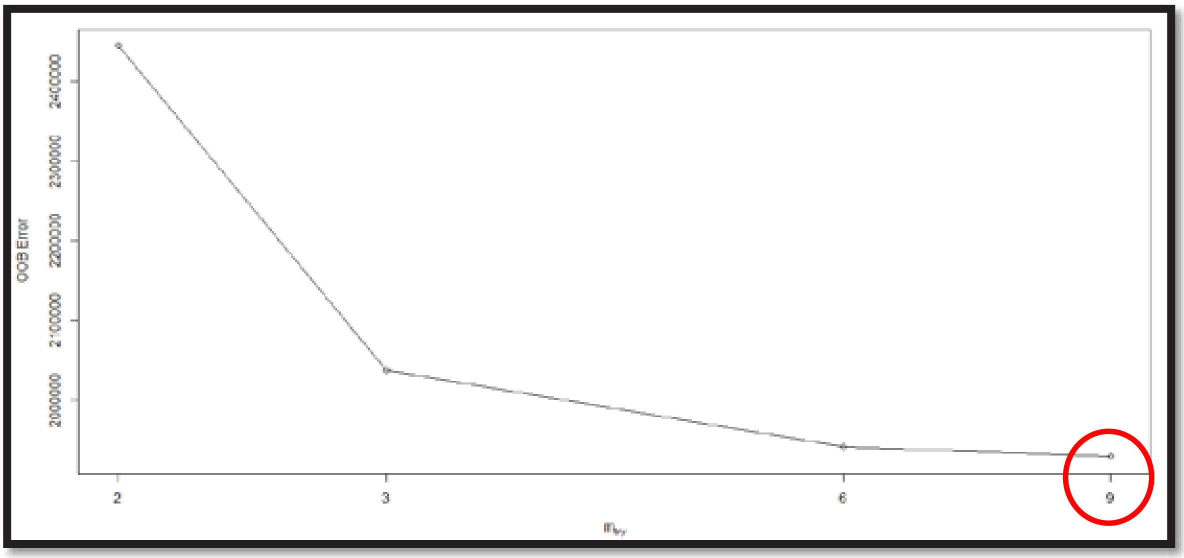


Fig 4

As evident from above graph, **mtry = 9** gives the least OOB error rate.

❖ Tuning parameters – terminologies:

a) nbagg:

- controls number of iterations / bootstraps to be included in the model
- higher the value, better will be the outcome as it averages all the samples batter.

b) coob:

- indicates if OOB error rate to be computed or not.

  (TRUE – computes OOB Error rate; FALSE – does not compute OOB Error rate)

c) minsplit:

- part of control parameters of Bagging function.
- indicates the minimum number of observations that must exist in a node in order for a split to be attempted.

d) cp:

- part of control parameters of Bagging function.
- It is the complexity parameter. Any split that does not decrease the overall lack of fit by a factor of cp is not attempted.

e) minbucket:

- part of control parameters of Bagging function.
- the minimum number of observations in any terminal (leaf) node.

f) Shrinkage :

- Indicates the contribution of each tree to final outcome and controls how quickly the algorithm proceeds down the gradient descent.
- Typical values range between 0.001–0.3. the smaller this value, the more accurate the model can be but also will require more trees in the sequence.

g) Interaction depth:

- Controls the depth of the individual trees and typical values range from a depth of 3–8.
- Smaller depth trees such as decision stumps are computationally efficient however, higher depth trees allow the algorithm to capture unique interactions but higher risk of over-fitting

h) n.minobsinnode:

- Indicates minimum number of observations in terminal nodes.
- Typical values range from 5–15 where higher values help prevent a model from over-fitting.

i) n.trees:

- The total number of trees in the sequence or ensemble.
- Optimal number of trees to be selected that minimize the loss function of interest with cross validation.

j) Verbose:

- Controls the display of errors after each stump / trees
  (TRUE – displays errors; FALSE – does not display errors)

k) min_child_weight:

- It refers to the minimum number of instances required in a child node.
- Higher values will be prone to over-fit the model, thus the value
  to be decided using CV.

l) early_stopping_rounds:

- controls stopping the process if no improvement observed for given consecutive number of trees.

m) nrounds:

- It controls the maximum number of iterations and should be tuned using CV.

n) subsample:

- It controls the number of samples (observations) supplied to a tree.
- Typically, its values lie between (0.5-0.8)

o) colsample_bytree:

- It control the number of features (variables) supplied to a tree
- Typically, its values lie between (0.5 - 0.9)

❖ R-Codes:

```
#Setting up directories
setwd("C:/Users/Rahul moorthy.rahul-PC/Desktop/PGP-BABI/Capstone/8. Flight Price Prediction")
getwd()

#Installing libraries
library(readxl)
install.packages("corrplot")
library(corrplot)
install.packages("MASS")
library(MASS)
install.packages("caret")
library(caret)
install.packages("ranger")
library(ranger)
install.packages("ipred")
library(ipred)
install.packages("rpart")
library(rpart)
install.packages("gbm")
library(gbm)
library(tidyr)
library(dplyr)
library(lubridate)
library(reshape)
install.packages("xgboost")
library(xgboost)
install.packages("Metrics")
library(Metrics)
install.packages("randomForest")
library(randomForest)
install.packages("rpart.plot")
library(rpart.plot)
library(xlsx)

#Importing dataset
flight_data = read_excel("FlightPrice_train_cleaned.xlsx")
str(flight_data)
attach(flight_data)
flight_data_test_new = read_excel("FlightPrice_test_cleaned.xlsx")

#Variable Transformation
flight_data$Airline = as.factor(flight_data$Airline)
flight_data$Source = as.factor(flight_data$Source)
flight_data$Destination = as.factor(flight_data$Destination)
flight_data$Route = as.factor(flight_data$Route)
flight_data$Total_Stops = as.factor(flight_data$Total_Stops)
flight_data$Additional_Info = as.factor(flight_data$Additional_Info)
flight_data$Date_of_Journey = as.Date(flight_data$Date_of_Journey)
flight_data = transform(flight_data, stops = colsplit(flight_data$Total_Stops, split = "\\ ", names = c('count','dummy')))
flight_data$stops.count = ifelse(flight_data$stops.count == "non-stop","0",flight_data$stops.count)
flight_data$stops.count = as.numeric(flight_data$stops.count)
flight_data$Day = weekdays(as.Date(flight_data$Date_of_Journey,"%d/%m/%Y"))
flight_data$Wknd.wkday = ifelse(flight_data$Day == "Sunday"|flight_data$Day == "Saturday","Weekend","Weekday")
```

```
flight_data$peak.normalhrs = ifelse(flight_data$dep.hours>=9
&flight_data$dep.hours<=21,"Peak_Hour","Normal_hour")
flight_data$Day = as.factor(flight_data$Day)
flight_data$Wknd.wkday = as.factor(flight_data$Wknd.wkday)
flight_data$peak.normalhrs = as.factor(flight_data$peak.normalhrs)
```

#Check for multicollinerarity

```
flight_data.corr = cor(flight_data[,c(9:14)])
flight_data.corrplot = corrplot(flight_data.corr, method = "circle")
```

#Multi collinearity treatment - feature selection

```
step.model = stepAIC(glm(Price~.,data = flight_data),direction = "both", trace = TRUE)
```

#Splitting data

```
set.seed(1000)
indices = sample(1:nrow(flight_data),0.7*nrow(flight_data))
flight_data.train = flight_data[indices,]
flight_data.test = flight_data[-indices,]
attach(flight_data.train)
```

#Building Linear Regression model
##Train model using CV

```
set.seed(1000)
(cv_model1 =  train(
  form = Price~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.mins + arr.mins + stops.count + Day + peak.normalhrs,
  data = flight_data.train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))
summary(cv_model1)

set.seed(1000)
(cv_model2 =  train(
  form = Price~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.hours + arr.hours + stops.count + Day + peak.normalhrs,
  data = flight_data.train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))

set.seed(1000)
(cv_model3 =  train(
  form = Price~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    Duration_mins + stops.count + Day + peak.normalhrs,
  data = flight_data.train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))

set.seed(1000)
(cv_model4 =  train(
  form = Price~ .,
  data = flight_data.train,
  method = "lm",
```

```
  trControl = trainControl(method = "cv", number = 10)
))
```

#Model Accuracy

```
summary(resamples(list(
  model1 = cv_model1,
  model2 = cv_model2,
  model3 = cv_model3,
  model4 = cv_model4
)))
```

#KNN

```
cv <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5
)

k_val <- expand.grid(k = seq(2, 15, by = 1))

knn_fit <- train(
  Price~Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.mins + arr.mins + stops.count + Day + peak.normalhrs,
  data = flight_data.train,
  method = "knn",
  trControl = cv,
  tuneGrid = k_val,
  metric = "RMSE"
)
```

#Decision Trees

```
Flight_price__dt = rpart(
  formula = Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.mins + arr.mins + stops.count + Day + peak.normalhrs,
  data    = flight_data.train,
  method  = "anova",
  control = list(cp = 0, xval = 10)
)

printcp(Flight_price__dt)
plotcp(Flight_price__dt)
rpart.plot(Flight_price__dt)

Flight_price__dt2 <- train(
  Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.mins + arr.mins + stops.count + Day + peak.normalhrs,
  data = flight_data.train,
  method = "rpart",
  trControl = trainControl(method = "cv", number = 10),
  control = list(cp = 0.00038, xval = 10),
  tuneLength = 20
)

rpart.plot(Flight_price__dt2)
```

#RANDOM FOREST

```
set.seed(1000)
flight_rf_fit = randomForest(Price ~ Airline + Date_of_Journey + Destination + Additional_Info +
                              dep.mins + arr.mins + stops.count + Day + peak.normalhrs, data = flight_data.train,
                             ntree = 501, mtry = 3, nodesize = 10, importance = TRUE)

print(flight_rf_fit)
plot(flight_rf_fit)
importance(flight_rf_fit)
```

##Tuned model

```
set.seed(1000)
flight_rf_fit.tuned = tuneRF(x = flight_data.train[,c(1,2,4,7,10,12,14,16,18)], y = flight_data.train$Price, mtrystart = 3,
stepfactor = 1.5, nTreetry = 101,
                             improve = 0.0001, nodesize = 10, plot = TRUE, doBest = TRUE, importance = TRUE)

set.seed(1000)
flight_rf_fit_final = randomForest(Price ~ Airline + Date_of_Journey + Destination + Additional_Info +
                              dep.mins + arr.mins + stops.count + Day + peak.normalhrs, data = flight_data.train,
                             ntree = 101, mtry = 9, nodesize = 10, importance = TRUE)

importance(flight_rf_fit_final)
```

#Bagging

```
flight_price_bagging1 <- bagging(
  formula = Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.mins + arr.mins + stops.count + Day + peak.normalhrs,
  data = flight_data.train,
  nbagg = 50,
  coob = TRUE,
)


flight_price_bagging2 <- bagging(
  formula = Price ~ Airline + Date_of_Journey + Destination + Route + Additional_Info +
    dep.mins + arr
  .mins + stops.count + Day + peak.normalhrs,
  data = flight_data.train,
  nbagg = 1000,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0.0027,minbucket = 10)
)
```

#Gradient Boosting

```
flight_price_gbm = gbm(formula = Price ~ Airline + Destination + Route + Additional_Info +
                        dep.mins + arr.mins + stops.count + Day + peak.normalhrs ,
                       distribution = "gaussian",
                       data = flight_data.train,
                       n.trees = 10000,
                       interaction.depth = 3,
                       shrinkage = 0.001,
                       n.cores = NULL,
                       verbose = FALSE
                       )
```

```
flight_gbm_RMSE = sqrt(min(flight_price_gbm$train.error))
```

###Tuned model

```
tuning_grid1 = expand.grid(
  shrinkage <- c(0.001, 0.01, 0.1),
  interaction.depth = c(1, 3, 5),
  n.minobsinnode = c(5, 10, 15)

)

for(i in seq_len(nrow(tuning_grid1))) {
  set.seed(1000)
  flight_price_gbm2 <- gbm(
    formula = Price ~ Airline + Destination + Route + Additional_Info +
      dep.mins + arr.mins + stops.count + Day + peak.normalhrs ,
    data = flight_data.train,
    distribution = "gaussian",
    n.trees = 10000,
    n.cores = NULL,
    verbose = FALSE,
    shrinkage = tuning_grid1$Var1[i],
    interaction.depth = tuning_grid1$interaction.depth[i],
    n.minobsinnode = tuning_grid1$n.minobsinnode[i]
  )
}

flight_gbm_RMSE2 = sqrt(min(flight_price_gbm2$train.error))
```

#XG Boosting
##One-hot encoding - Train set

```
library(recipes)
xgb_prep <- recipe(Price ~ Airline + Destination + Route + Additional_Info +
            dep.mins + arr.mins + stops.count + Day + peak.normalhrs, data = flight_data.train) %>%
  step_integer(all_nominal()) %>%
  prep(training = flight_data.train, retain = TRUE) %>%
  juice()

X <- as.matrix(xgb_prep[setdiff(names(xgb_prep), "Price")])
Y <- xgb_prep$Price
```

##One-hot coding - Test set

```
xgb_prep_test <- recipe(Price ~ Airline + Destination + Route + Additional_Info +
            dep.mins + arr.mins + stops.count + Day + peak.normalhrs, data = flight_data.test) %>%
  step_integer(all_nominal()) %>%
  prep(training = flight_data.test, retain = TRUE) %>%
  juice()

X_test <- as.matrix(xgb_prep_test[setdiff(names(xgb_prep_test), "Price")])
Y_test <- xgb_prep_test$Price
```

##Model Building

```
flight_price_xgb <- xgb.cv(
  data = X,
  label = Y,
  nrounds = 6000,
  objective = "reg:linear",
```

```
  early_stopping_rounds = 50,
  nfold = 10,
  params = list(
    eta = 0.1,
    max_depth = 3,
    min_child_weight = 3,
    subsample = 0.8,
    colsample_bytree = 1.0),
  verbose = 0
)

flight_xgb_RMSE = min(flight_price_xgb$evaluation_log$test_rmse_mean)
```

###Tuned model

```
tuning.grid2 = expand.grid(
  lr <- c(0.001, 0.01, 0.1),
  md<-c(1,3,5)
)
for(i in seq_len(nrow(tuning.grid2))) {
  set.seed(1000)
  flight_price.xgb2 <- xgboost(
    data = X,
    label = Y,
    nrounds = 6000,
    objective = "reg:linear",
    verbose = 0,
     eta = tuning.grid2$Var1[i],
     md = tuning.grid2$Var2[i],
     gamma = 5
  )
}

flight_xgb2_RMSE = min(flight_price.xgb2$evaluation_log$train_rmse)
```

#Model Prediction
##Linear Regression - Model 1
###Train data

```
Flight_data.train.predict.lm = predict(cv_model1, data = flight_data.train)
plot(flight_data.train$Price, Flight_data.train.predict.lm)
```

####Test data

```
Flight_data.test.predict.lm = predict(cv_model1, newdata = flight_data.test)
plot(flight_data.test$Price, Flight_data.test.predict.lm)
```

##KNN Algorithm
###Train data

```
Flight_data.train.predict.knn = predict(knn_fit, data = flight_data.train)
plot(flight_data.train$Price, Flight_data.train.predict.knn)
```

####Test data

```
Flight_data.test.predict.knn = predict(knn_fit, newdata = flight_data.test)
plot(flight_data.test$Price, Flight_data.test.predict.knn)
```

##Decision Trees
###Train data
```
Flight_data.train.predict.dt = predict(Flight_price__dt2, newdata = flight_data.train)
plot(flight_data.train$Price, Flight_data.train.predict.dt)
```

####Test data
```
Flight_data.test.predict.dt = predict(Flight_price__dt2, newdata = flight_data.test)
plot(flight_data.test$Price, Flight_data.test.predict.dt)
```

##Random Forest
###Train data
```
Flight_data.train.predict.rf = predict(flight_rf_fit_final, newdata = flight_data.train)
plot(flight_data.train$Price, Flight_data.train.predict.rf)
```

####Test data
```
Flight_data.test.predict.rf = predict(flight_rf_fit_final, newdata = flight_data.test)
plot(flight_data.test$Price, Flight_data.test.predict.rf)
```

##Bagging
###Train data
```
Flight_data.train.predict.bag = predict(flight_price_bagging2, newdata = flight_data.train)
plot(flight_data.train$Price, Flight_data.train.predict.bag)
```

####Test data
```
Flight_data.test.predict.bag = predict(flight_price_bagging2, newdata = flight_data.test)
plot(flight_data.test$Price, Flight_data.test.predict.bag)
```

##Gradient Boosting
###Train data
```
Flight_data.train.predict.gbm = predict(flight_price_gbm2, newdata = flight_data.train, n.trees = 10000)
plot(flight_data.train$Price, Flight_data.train.predict.gbm)
```

####Test data
```
Flight_data.test.predict.gbm = predict(flight_price_gbm2, newdata = flight_data.test, n.trees = 10000)
plot(flight_data.test$Price, Flight_data.test.predict.gbm)
```

##XG Boost
###Train data
```
Flight_data.train.predict.xgb = predict(flight_price.xgb2, newdata = X, label = Y)
plot(flight_data.train$Price, Flight_data.train.predict.xgb)
```

####Test data
```
Flight_data.test.predict.xgb = predict(flight_price.xgb2, newdata = X_test, label = Y_test)
plot(flight_data.test$Price, Flight_data.test.predict.xgb)
```

#Model Performance Measures
##Linear Regression Model
###R-sq
```
lm.train.rsq = cor(flight_data.train$Price, Flight_data.train.predict.lm)^2
lm.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.lm)^2
```

###MAPE
```
lm.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.lm)
lm.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.lm)
```

###RMSE
lm.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.lm)
lm.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.lm)

###SSE
lm.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.lm)
lm.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.lm)

###MAE
lm.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.lm)
lm.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.lm)

###MSE
lm.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.lm)
lm.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.lm)

##KNN Algorithm
###R-sq
knn.train.rsq = cor(flight_data.train$Price, Flight_data.train.predict.knn)^2
knn.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.knn)^2

###MAPE
knn.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.knn)
knn.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.knn)

###RMSE
knn.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.knn)
knn.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.knn)

###SSE
knn.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.knn)
knn.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.knn)

###MAE
knn.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.knn)
knn.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.knn)

###MSE
knn.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.knn)
knn.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.knn)

##Decision tree model
###R-sq
dt.train.rsq = cor(flight_data.train$Price, Flight_data.train.predict.dt)^2
dt.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.dt)^2

###MAPE
dt.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.dt)
dt.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.dt)

###RMSE
dt.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.dt)
dt.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.dt)

###SSE
dt.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.dt)
dt.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.dt)

###MAE
dt.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.dt)
dt.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.dt)

###MSE
dt.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.dt)
dt.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.dt)

##Random Forest model
###R-sq
rf.train.rsq = cor(flight_data.train$Price, Flight_data.train.predict.rf)^2
rf.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.rf)^2

###MAPE
rf.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.rf)
rf.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.rf)

###RMSE
rf.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.rf)
rf.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.rf)

###SSE
rf.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.rf)
rf.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.rf)

###MAE
rf.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.rf)
rf.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.rf)

###MSE
rf.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.rf)
rf.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.rf)

##Bagging
###R-sq
bag.train.rsq = cor(flight_data.train$Price, Flight_data.train.predict.bag)^2
bag.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.bag)^2

###MAPE
bag.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.bag)
bag.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.bag)

###RMSE
bag.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.bag)
bag.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.bag)

###SSE
bag.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.bag)
bag.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.bag)

###MAE
bag.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.bag)

bag.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.bag)

###MSE
bag.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.bag)
bag.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.bag)

##Gradient Boosting
###R-sq
gbm.tain.rsq = cor(flight_data.train$Price, Flight_data.train.predict.gbm)^2
gbm.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.gbm)^2

###MAPE
gbm.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.gbm)
gbm.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.gbm)

###RMSE
gbm.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.gbm)
gbm.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.gbm)

###SSE
gbm.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.gbm)
gbm.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.gbm)

###MAE
gbm.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.gbm)
gbm.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.gbm)

###MSE
gbm.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.gbm)
gbm.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.gbm)

##XG Boost Model
###R-sq
xgb.train.rsq = cor(flight_data.train$Price, Flight_data.train.predict.xgb)^2
xgb.test.rsq = cor(flight_data.test$Price, Flight_data.test.predict.xgb)^2

###MAPE
xgb.train.mape = mape(flight_data.train$Price, Flight_data.train.predict.xgb)
xgb.test.mape = mape(flight_data.test$Price, Flight_data.test.predict.xgb)

###RMSE
xgb.train.rmse = rmse(flight_data.train$Price, Flight_data.train.predict.xgb)
xgb.test.rmse = rmse(flight_data.test$Price, Flight_data.test.predict.xgb)

###SSE
xgb.train.sse = sse(flight_data.train$Price, Flight_data.train.predict.xgb)
xgb.test.sse = sse(flight_data.test$Price, Flight_data.test.predict.xgb)

###MAE
xgb.train.mae = mae(flight_data.train$Price, Flight_data.train.predict.xgb)
xgb.test.mae = mae(flight_data.test$Price, Flight_data.test.predict.xgb)

###MSE
xgb.train.mse = mse(flight_data.train$Price, Flight_data.train.predict.xgb)
xgb.test.mse = mse(flight_data.test$Price, Flight_data.test.predict.xgb)

#Prediction dataframe

```
Prediction.train = data.frame(flight_data.train$Price, Flight_data.train.predict.lm,
Flight_data.train.predict.knn,Flight_data.train.predict.dt,Flight_data.train.predict.rf,
                         Flight_data.train.predict.bag,Flight_data.train.predict.gbm,Flight_data.train.predict.xgb )
Prediction.train = round(Prediction.train)
Prediction.test =
data.frame(flight_data.test$Price,Flight_data.test.predict.lm,Flight_data.test.predict.knn,Flight_data.test.predict.dt,
Flight_data.test.predict.rf,
                        Flight_data.test.predict.bag,Flight_data.test.predict.gbm, Flight_data.test.predict.xgb)
Prediction.test = round(Prediction.test)
```

#Prediction on New dataset based on best model

```
flight_data_test_new$Airline = as.factor(flight_data_test_new$Airline)
flight_data_test_new$Source = as.factor(flight_data_test_new$Source)
flight_data_test_new$Destination = as.factor(flight_data_test_new$Destination)
flight_data_test_new$Route = as.factor(flight_data_test_new$Route)
flight_data_test_new$Total_Stops = as.factor(flight_data_test_new$Total_Stops)
flight_data_test_new$Additional_Info = as.factor(flight_data_test_new$Additional_Info)
flight_data_test_new$Date_of_Journey = as.Date(flight_data_test_new$Date_of_Journey)
flight_data_test_new = transform(flight_data_test_new, stops = colsplit(flight_data_test_new$Total_Stops, split = "\\ ",
names = c('count','dummy')))
flight_data_test_new$stops.count = ifelse(flight_data_test_new$stops.count == "non-
stop","0",flight_data_test_new$stops.count)
flight_data_test_new$stops.count = as.numeric(flight_data_test_new$stops.count)
flight_data_test_new$Day = weekdays(as.Date(flight_data_test_new$Date_of_Journey,"%d/%m/%Y"))
flight_data_test_new$Wknd.wkday = ifelse(flight_data_test_new$Day == "Sunday"|flight_data_test_new$Day ==
"Saturday","Weekend","Weekday")
flight_data_test_new$peak.normalhrs = ifelse(flight_data_test_new$dep.hours>=9
&flight_data_test_new$dep.hours<=21,"Peak_Hour","Normal_hour")
flight_data_test_new$Day = as.factor(flight_data_test_new$Day)
flight_data_test_new$Wknd.wkday = as.factor(flight_data_test_new$Wknd.wkday)
flight_data_test_new$peak.normalhrs = as.factor(flight_data_test_new$peak.normalhrs)
str(flight_data_test_new)

common_levels <- intersect(names(flight_data.train), names(flight_data_test_new))
for (p in common_levels) {
  if (class(flight_data.train[[p]]) == "factor") {
    levels(flight_data_test_new[[p]]) <- levels(flight_data.train[[p]])
  }
}

flight_data_test_new$Predicted.Price = round(predict(flight_rf_fit_final, newdata = flight_data_test_new))
```

#Exporting the output file

```
Flight.Prediction_output = write.xlsx(flight_data_test_new, file = "Flight.Prediction_Output.xlsx", row.names = TRUE,
append = FALSE)
```