Flight Price Prediction

Build predictive models to automate the process of predicting the price of the flight ticket for the desired airline.

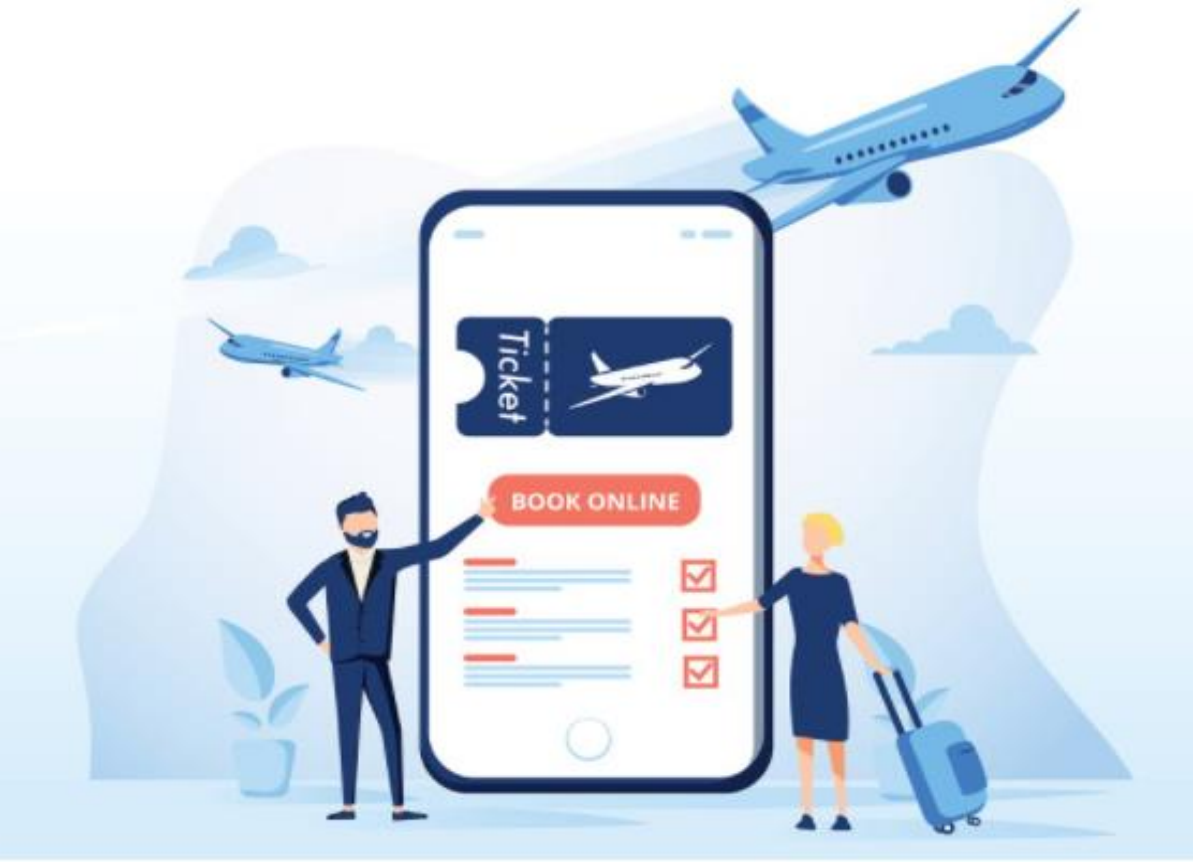


Table of Contents:

1. Problem Definition.
2. Data Analysis.
3. EDA Concluding Remark.
4. Pre-Processing Pipeline.
5. Building Machine Learning Models.
6. Concluding Remarks.

Problem Definition

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travellers saying that flight ticket prices are so unpredictable.

Objective:

Here, with the help of the prices of flight tickets for various airlines, which has been scrapped from yatra.com & Ixigo.com for the months of November 2021 for few cities, we would be able to predict flight prices with the help of real time data.

Type of problem:

The above problem statement clearly explains that the target variable is continuous & it's a regression problem as we need to predict the price of the flight tickets. So this can be solved by any of the below Regression Machine learning algorithms:

1. Linear Regression.

2. Decision Tree Regressor.

3. Random Forest Regressor.

I have mentioned only few. We would be performing each of the techniques later in this blog.

Importing libraries:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')
```

Data Analysis

Data Set Description:

Data has been scrapped from online flight ticket platform.

1) Train file will be used for training the model, i.e. the model will learn from this file. It contains all the independent variables and the target variable. Size of training set: 1520 records.

Read the excel file and convert into data frame:

	Unnamed: 0	Airline Name	Date of journey	Source	Destination	Departure Time	Arrival Time	Duration	Total Stops	Price
0	0	SpiceJet	Mon, 22 Nov	Mumbai	Kolkata	6:15	8:55	2h 40m	Non Stop	4326
1	1	SpiceJet	Mon, 22 Nov	Mumbai	Kolkata	18:40	21:25	2h 45m	Non Stop	4326
2	2	Air Asia	Mon, 22 Nov	Mumbai	Kolkata	2:35	8:05	5h 30m	1 Stop	7409
3	3	Air Asia	Mon, 22 Nov	Mumbai	Kolkata	13:05	19:20	6h 15m	2 Stop(s)	7409
4	4	Air Asia	Mon, 22 Nov	Mumbai	Kolkata	13:05	21:05	8h 00m	1 Stop	7409

Airline Name: Name of the airline used for traveling

Date of Journey: Date at which a person travelled

Source: Starting location of flight

Destination: Ending location of flight

Departure Time: Departure time of flight from starting location

Arrival Time: Arrival time of flight at destination

Duration: Duration of flight in hours/minutes

Total Stops: Number of total stops flight took before landing at the destination.

Price: Price of the flight.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1520 entries, 0 to 1519
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline Name     1520 non-null   object
 1   Date of journey  1520 non-null   object
 2   Source           1520 non-null   object
 3   Destination      1520 non-null   object
 4   Departure Time   1520 non-null   object
 5   Arrival Time     1520 non-null   object
 6   Duration         1520 non-null   object
 7   Total Stops      1520 non-null   object
 8   Price            1520 non-null   int64 
dtypes: int64(1), object(8)
memory usage: 107.0+ KB
```

- We have 1520 rows & 9 columns in the dataset.
- We have maximum columns as categorical, which is an independent feature.

Let's check for each independent features & convert the data type into required Data type:

Training data set is having all independent feature as categorical and only target variable is numerical, also we can see some special character also being used because of that we need to perform data transformation on it before building the model.

1) Date of Journey: it is given as day-month-year, so we need to split & use date time to convert it & separate it with day & month.

```
# Lets split the date format
Day=df['Date of journey'].str.split(' ',expand=True)
df["Day"] = Day[0].apply(lambda x: x[:-1])
df["Date"] = Day[1].apply(lambda x: x[0:])
df["Month"] = Day[2].apply(lambda x: x[0:])
```

```
## Since we have converted the date of journey with new separate column of date & month, we can drop these column.
df.drop(["Date of journey"], axis = 1, inplace = True)
```

2) Similarly, we can separate the hour & minutes for Departure Time & Arrival Time.

```
# Separating hours
df["Dep_hour"] = pd.to_datetime(df["Departure Time"]).dt.hour

# Separating mins
df["Dep_min"] = pd.to_datetime(df["Departure Time"]).dt.minute

# We can drop the Dep_Time column as it is of no use now
df.drop(["Departure Time"], axis = 1, inplace = True)
```

```
# Separating hours
df["Air_hour"] = pd.to_datetime(Time[0]).dt.hour

# Separating mins
df["Air_min"] = pd.to_datetime(Time[0]).dt.minute
```

After dropping unwanted columns as it is of no use now: Arrival Time, Departure Time & Date of Journey, we have:

	Airline Name	Source	Destination	Duration	Total Stops	Price	Day	Date	Month	Dep_hour	Dep_min	Air_hour	Air_min
0	SpiceJet	Mumbai	Kolkata	2h 40m	Non Stop	4326	Mon	22	Nov	6	15	8	55
1	SpiceJet	Mumbai	Kolkata	2h 45m	Non Stop	4326	Mon	22	Nov	18	40	21	25
2	Air Asia	Mumbai	Kolkata	5h 30m	1 Stop	7409	Mon	22	Nov	2	35	8	5
3	Air Asia	Mumbai	Kolkata	6h 15m	2 Stop(s)	7409	Mon	22	Nov	13	5	19	20
4	Air Asia	Mumbai	Kolkata	8h 00m	1 Stop	7409	Mon	22	Nov	13	5	21	5

3) For duration column, if we see above in the dataset, it is basically the difference between arrival & departure time & it is given as string data type. If we could split than it will be again in string format. So we need to split it & add 0 hours & mins.

```
# same above concept can be applied in small lines of below code
duration=df['Duration'].str.split(' ',expand=True)
duration[1].fillna('0m',inplace=True)
df["Duration_hours"] = duration[0].apply(lambda x: x[:-1])
df["Duration_mins"] = duration[1].apply(lambda x: x[:-1])
```

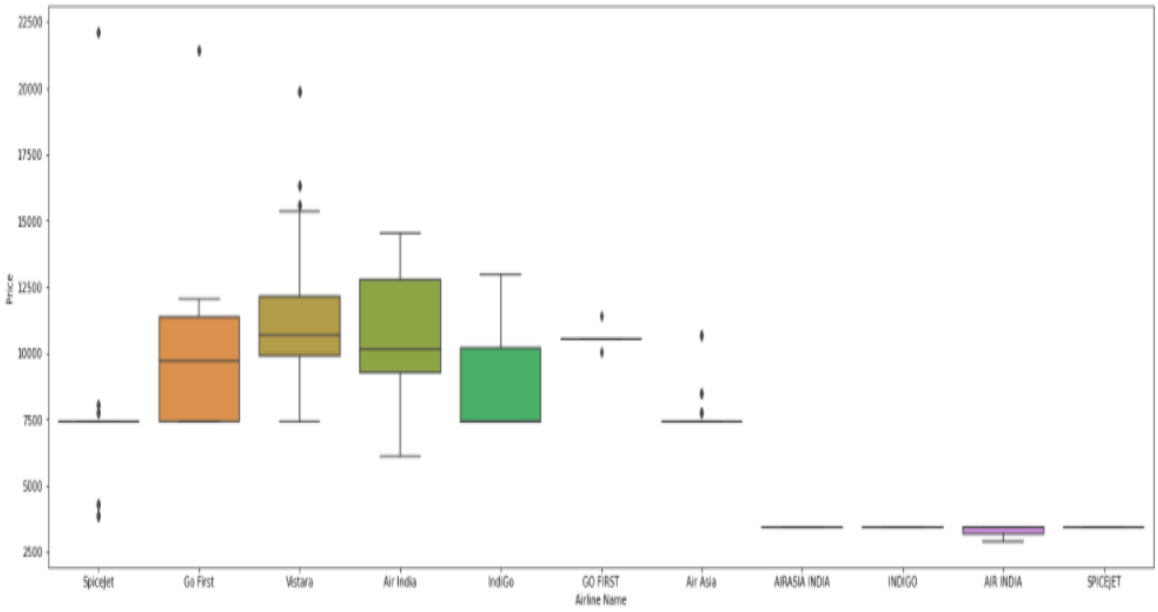
EDA Concluding Remark

Handling categorical columns:

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Airline Name    1520 non-null   object
 1   Source          1520 non-null   object
 2   Destination     1520 non-null   object
 3   Total Stops     1520 non-null   object
 4   Price           1520 non-null   int64
 5   Day             1520 non-null   object
 6   Date            1520 non-null   int32
 7   Month           1520 non-null   object
 8   Dep_hour        1520 non-null   int64
 9   Dep_min         1520 non-null   int64
 10  Air_hour        1520 non-null   int64
 11  Air_min         1520 non-null   int64
 12  Duration_hours  1520 non-null   int32
 13  Duration_mins   1520 non-null   int32
```

So, we have Airline Name, Source, Destination, Total Stops, Day & Month as categorical columns. Let's check the relation for each feature with Price.

1) Airline Vs Price

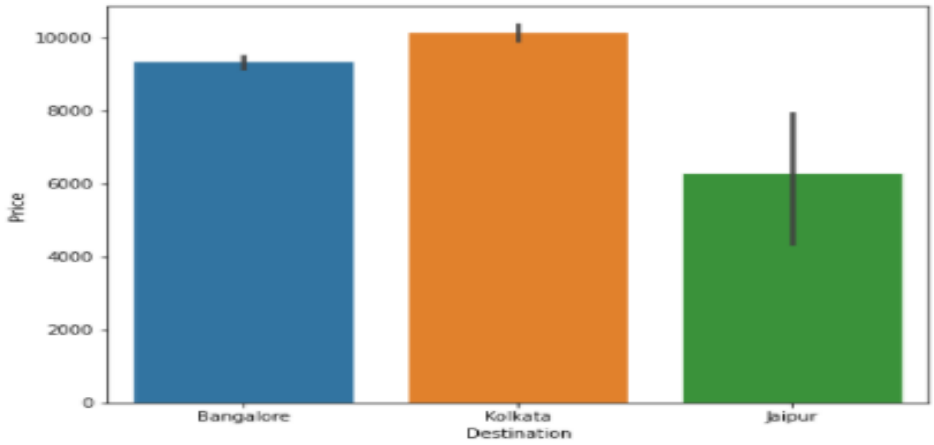


Spice jet is selling tickets with maximum fare followed by Go first & Vistara.

2) Source vs Price

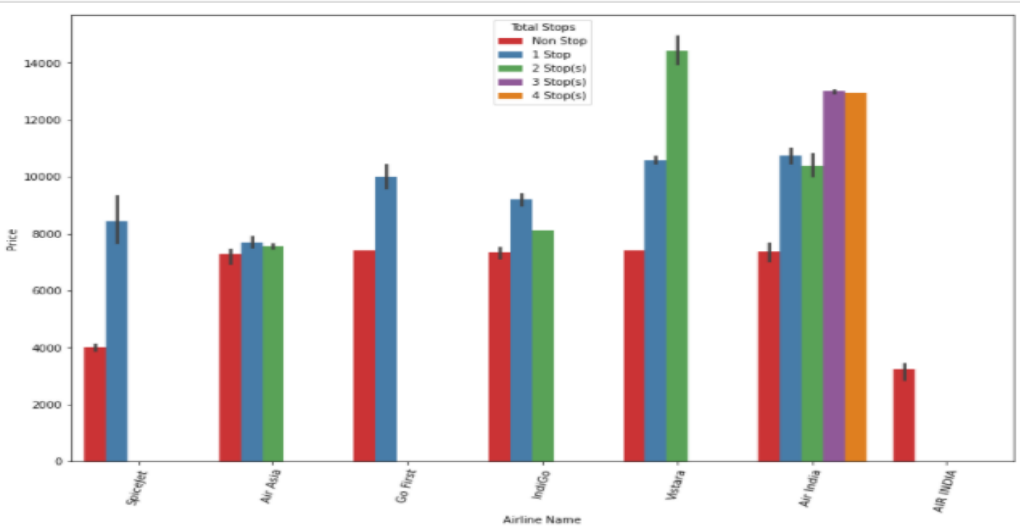
Maximum fare of the flights is departing from New Delhi followed by Mumbai.

3) Destination Vs Price



Maximum destination is Kolkata having highest avg fare.

4) Total Stops(Airlines) Vs Price



Vistara is showing the highest price having a 2 stops followed by 1 stop.

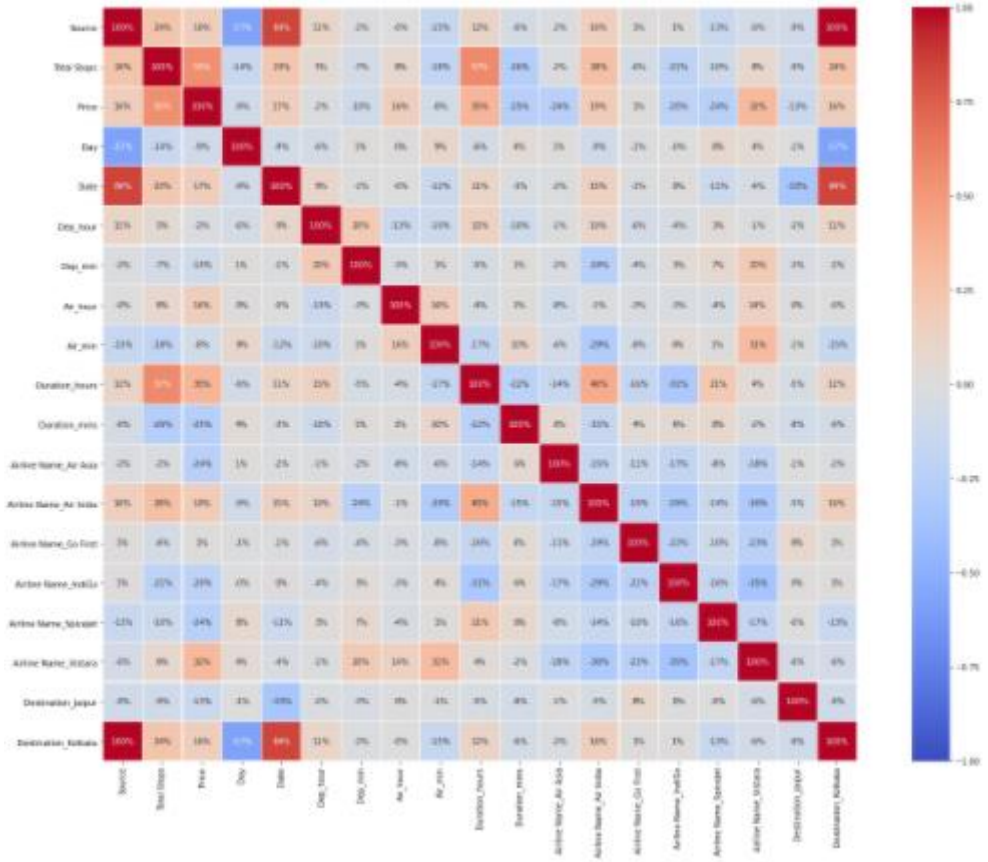
Action Taken based on observation:

We have seen how categorical data is distributed & correlated with the target variable. Also, I have mentioned the findings below each plot.

Performed One Hot Encoding for Nominal categorical columns: Airline & Destination & performed label encoding for an ordinal categorical column: Total Stops, Source & Day.

No need to check for outliers/skewness as all the independent features are categorical.

Visualizing the correlation in heat map to check if there is any coefficient of multicolinearity



1)Total stops is more correlated with target variable.
2)Multi collinearity: Date & Source, Date & Destination kolkata.

Pre-Processing Pipeline

Divide the data set into test and train

1) Now that we have converted all the categorical columns into numerical using the encoding technique. The next step is to split the data into test and train and drop the price column from the data set as we need to predict the price.

```
## Divide data set into features and label
y = df['Price']
x = df.drop(columns=['Price'], axis=1)
```

2) Finding best Random_ state: We have used Random Forest Regressor to obtain best random state.

3) Final dimension of data: 1520 rows & 19 columns.

4) Created train test split: We have split the train & test data in 0.2 test size with best random state.

```
# Split the data into train and test. Model will be built on training data and tested on test data.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=maxRS)
y_train.head()
```

```
969     10680
417      7413
778     10523
531      7409
1144    10680
Name: Price, dtype: int64
```

```
x_train.shape
```

```
(1216, 18)
```

```
y_train.shape
```

```
(1216,)
```

```
x_train.shape
```

```
(1216, 18)
```

```
y_test.shape
```

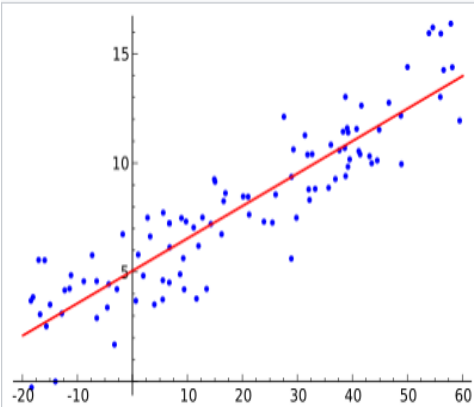
```
(304,)
```

Building Machine Learning Models

Model Creation:

Now, after performing the train test split, we have x_train, x_test, y_train & y_test, which are required to build Machine learning models. We would build multiple regressor models to get the best R2(coefficient of determination) regression score function out of all the models.

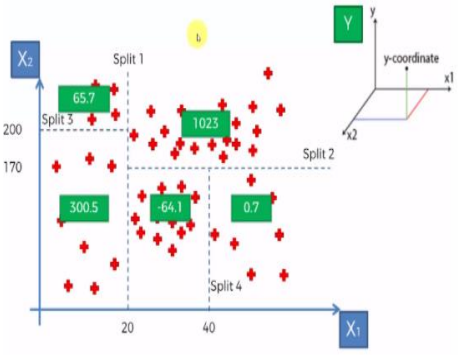
1) Linear Regression: It is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).




```
LR=LinearRegression()
LR.fit(x_train,y_train)
pred_test=LR.predict(x_test)

print(r2_score(y_test,pred_test))
```

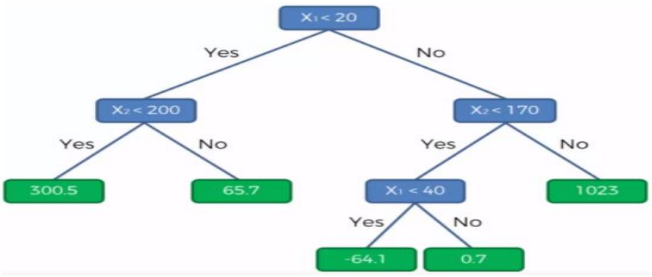
```
0.5074628444294847
```

2) Decision Tree Regression: Decision Regression trees are needed when the response variable is numeric or continuous. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

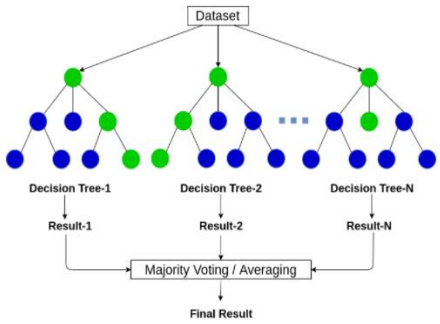



```
from sklearn.tree import DecisionTreeRegressor

DTR=DecisionTreeRegressor()
DTR.fit(x_train,y_train)
pred_test=DTR.predict(x_test)

print(r2_score(y_test,pred_test))
```

```
0.985095799986906
```



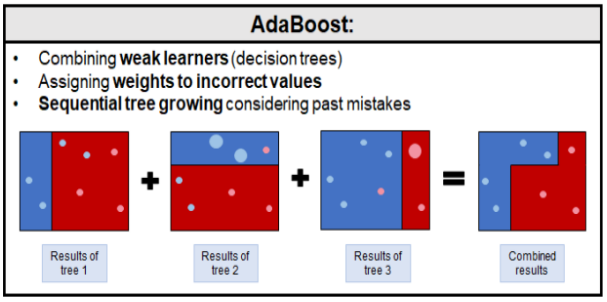
3) <u>Random Forest Regressor</u>: It uses ensemble learning method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction based on majority voting/averaging.




```
from sklearn.ensemble import RandomForestRegressor
RFR= RandomForestRegressor()
RFR.fit(x_train,y_train)
pred_test=RFR.predict(x_test)

print(r2_score(y_test,pred_test))
```

```
0.9958466967265808
```

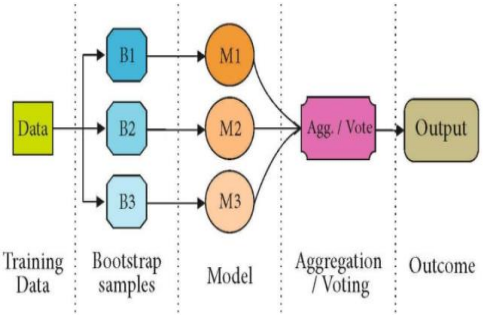
4) <u>AdaBoost Regressor</u>: It is used to boost the performance of any machine learning algorithm. It helps you combine multiple “weak classifiers” into a single “strong classifier”.




```
from sklearn.ensemble import AdaBoostRegressor
ADR= AdaBoostRegressor()
ADR.fit(x_train,y_train)
pred_test=ADR.predict(x_test)

print(r2_score(y_test,pred_test))
```

```
0.6028294470254514
```

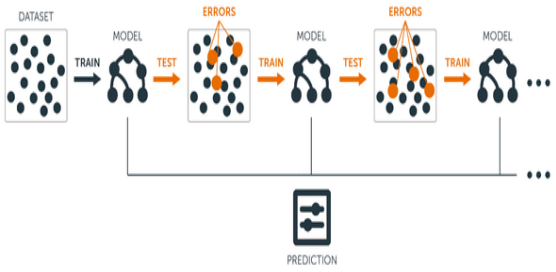
5) Bagging Regressor: It is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form a final prediction.




```
from sklearn.ensemble import BaggingRegressor
BR= BaggingRegressor()
BR.fit(x_train,y_train)
pred_test=BR.predict(x_test)

print(r2_score(y_test,pred_test))
```

```
0.9942754336540243
```

6) Gradient Boosting Regressor: It produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.




```
from sklearn.ensemble import GradientBoostingRegressor
GBR= GradientBoostingRegressor()
GBR.fit(x_train,y_train)
pred_test=GBR.predict(x_test)

print(r2_score(y_test,pred_test))
```

```
0.8382801122257566
```

1) Observation:

For every model, we have train the data (x_train, y_train) & predict with the help of x_test. Now, with the help of the y_test & prediction test we got the R2 score. So, based on every model we have best the R2 score for Random Forest Regressor with an 99% score, however this could be because

of overfitting. Hence we need perform cross validation to check if the model is overfitted or not.

Cross Validation:

We would perform cross validation for every model & compare it with the R2 score, which ever model gives less difference between Cross validation score & R2 score is the best fit model.

Models	R2 Score	CV Score	Difference
Logistic Regression	50	43	7
Decision Tree Regressor	98	95	3
Random Forest Regressor	**99**	**97**	**2**
Ada Boost Regressor	60	62	-2
Bagging Regressor	99	95	4
Gradient boosting Regressor	83	73	10

As we have seen, the Ada boost regressor is giving -2 difference but it is having a very low R2 Score. Random Forest Regressor is giving best R2 score with less difference of R2 Score & CV Score, however, let's proceed with model evaluation to get the MSE(mean squared error), RMSE(root mean squared error), & MAE(mean absolute error). These evalutaion metrics will help to decide the best model.

Model Evaluation:

```
#RandomForestRegressor
RFR= RandomForestRegressor()
RFR.fit(x_train,y_train)
pred_test=RFR.predict(x_test)
print("MSE:",mean_squared_error(y_test,pred_test))
print("RMSE:",math.sqrt(mean_squared_error(y_test,pred_test)))
print("MAE:",mean_absolute_error(y_test,pred_test))
```

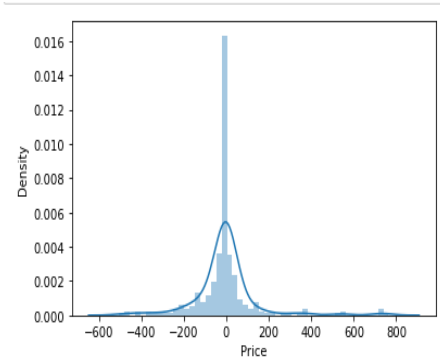
```
MSE: 28099.723130592127
RMSE: 167.62972030816053
MAE: 74.71009868421054
```

Random Forest Regressor is giving best result as compare to other models. Lets check its test & train score.

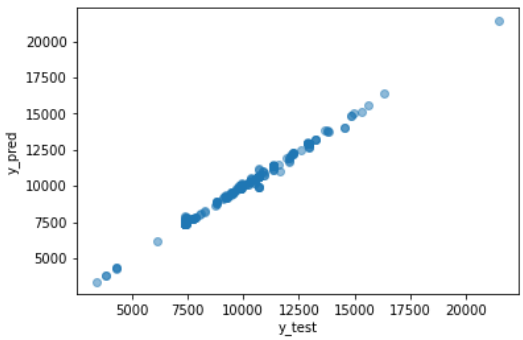
Train Score:

```
## RandomForestRegressor
RF = RandomForestRegressor()
RF.fit(x_train, y_train)
y_pred = RF.predict(x_test)
RF.score(x_train, y_train)
```

```
0.9988561231579637
```



Test Score:




```
from sklearn import metrics
metrics.r2_score(y_test, y_pred)
```

```
0.995672710481765
```

As we have seen, we have less difference of train & test score, and the predicted value & test value is normally distributed, Also, in the scatter plot the test value & the prediction value is linearly distributed. The test & prediction values are almost close to each other.

Since Random Forest Regressor is the best model in terms of model score, cross validation difference, test & train r2 score difference, also as per the evaluation metrics, we choose Random Forest Regressor to be the final model. Let's see if we can increase the score by using hyper parameter tuning.

Hyper Parameter Tuning:

We would try to increase the model by giving the best parameters (params as seen below) of Random Forest Regressor & again train to get an increased R2 score.

```
params = {'n_estimators': range(50,200,10),
                'max_features': ['auto', 'sqrt'],
                'max_depth': range(5,30,6),
                'min_samples_split': [2, 5, 10, 15, 100],
                'min_samples_leaf': [1, 2, 5, 10]}
```

```
from sklearn.model_selection import RandomizedSearchCV
rf = RandomizedSearchCV(RandomForestRegressor(),params,scoring='neg_mean_squared_error',n_iter = 10,cv=5, n_jobs= -1,random_sta
rf.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
                   param_distributions={'max_depth': range(5, 30, 6),
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': range(50, 200, 10)},
                   random_state=42, scoring='neg_mean_squared_error')
```

```
rf.best_params_
```

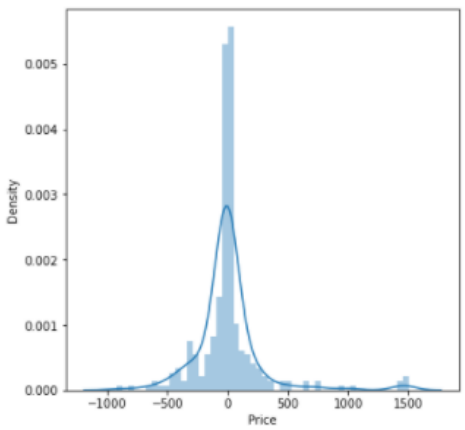
```
{'n_estimators': 90,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_features': 'auto',
```

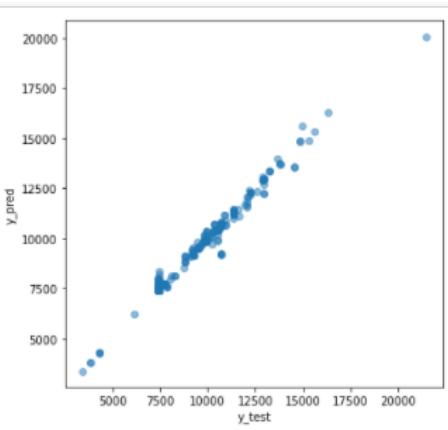
Prediction:

```
pred = rf.predict(x_test)
print(r2_score(y_test,pred))
```

```
0.9843256918579462
```

As we can see, Score has decreased to 1%.





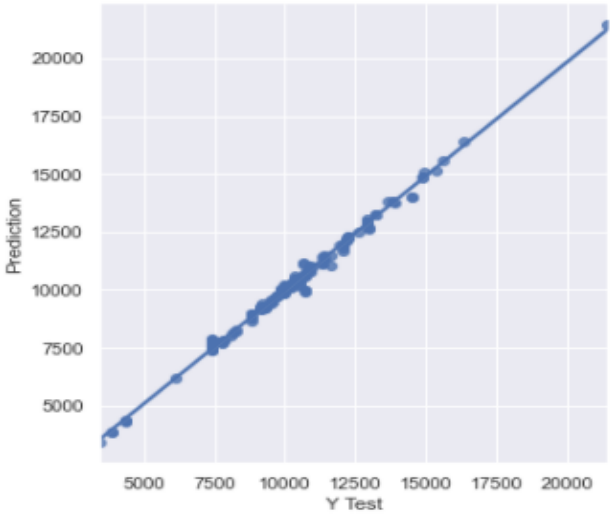

```
print("MSE:",mean_squared_error(y_test,pred))
print("RMSE:",math.sqrt(mean_squared_error(y_test,pred)))
print("MAE:",mean_absolute_error(y_test,pred))
```

```
MSE: 81332.10738559006
RMSE: 285.1878457886838
MAE: 144.5799726360584
```

After performing Hyper Parameter tuning, we have not increased the Score, evaluation metrics are also not showing better results, also test & prediction value is normally distributed. The Test & Prediction value is also linearly related to each other.

Let's predict & compare the results:

	Y Test	Prediction
467	9946	10147.04
929	9202	9166.44
971	10680	10680.00
1039	7425	7425.00
1372	7425	7424.99



Concluding Remarks

1) Saving the model: The model is ready & we have saved the model in ‘pkl’ format by using “joblib”.

Final conclusion:

As we have seen, the prediction is showing a similar relationship with the actual price from the scrapped data set, which means the model predicted correctly & this could help airlines by predicting what prices they can maintain. It could also help customers to predict future flight prices and plan the journey accordingly because it is difficult for airlines to maintain prices since it changes dynamically due to different conditions. Hence by using Machine learning techniques we can solve this problem.

WRITTEN BY:
DEEPAM PURKAYASTHA