Flight Fare Prediction for Aviation Sector Using Machine Learning

Sudhansu Mandal Sep 2,2021

➢ PROBLEM STATEMENT

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. Here we have prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

➢ OBJECTIVE

Here the objective is to predict prices given the various parameter. The task here would be regression problem since the target or dependent variable is price (continuous number).

➢ DATA ANALYSIS

- **Data Input for our analysis.**

For our analysis we have data set from between of March and June of 2019 and between various cities.
There are two separate dataset contains Training and Test file.
Training dataset having 10683 rows or records and 11 columns.
Test dataset having 2671 rows or records and 10 columns as shown in Fig 1.

Fig :1

```
Shape of Train Dataset: (10683, 11)
Total Rows in Train Dataset: 10683
Total Columns in Train Dataset: 11
************************************
Shape of Test Dataset: (2671, 10)
Total Rows in Test Dataset: 2671
Total Columns in Test Dataset: 10
```

- **Data Set's Features data type.**

  The Train and Test Dataset' columns/features having Object type data. Training dataset having one column float data type, which are the predictor variables or independent variables for our model and also having target variable or dependent variable for our analysis. (as shown in Fig: 2)

Fig: 2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
None

*******************************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2671 entries, 0 to 2670
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          2671 non-null   object
 1   Date_of_Journey  2671 non-null   object
 2   Source           2671 non-null   object
 3   Destination      2671 non-null   object
 4   Route            2671 non-null   object
 5   Dep_Time         2671 non-null   object
 6   Arrival_Time     2671 non-null   object
 7   Duration         2671 non-null   object
 8   Total_Stops      2671 non-null   object
 9   Additional_Info  2671 non-null   object
dtypes: object(10)
memory usage: 208.8+ KB
None
```

- **Statistic Summary of Data set.**

  Our training dataset and test dataset having object type data, while applying describe () method in categorial columns it provides some basis statistical details like count, unique, top and frequency of the dataset columns.   (As shown in Fig 3.)

Fig: 3

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
count	10683	10683	10683	10683	10682	10683	10683	10683	10682	10683
unique	12	44	5	6	128	222	1343	368	5	10
top	Jet Airways	18/05/2019	Delhi	Cochin	DEL → BOM → COK	18:55	19:00	2h 50m	1 stop	No info
freq	3849	504	4537	4537	2376	233	423	550	5625	8345

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
count	2671	2671	2671	2671	2671	2671	2671	2671	2671	2671
unique	11	44	5	6	100	199	704	320	5	6
top	Jet Airways	9/05/2019	Delhi	Cochin	DEL → BOM → COK	10:00	19:00	2h 50m	1 stop	No info
freq	897	144	1145	1145	624	62	113	122	1431	2148

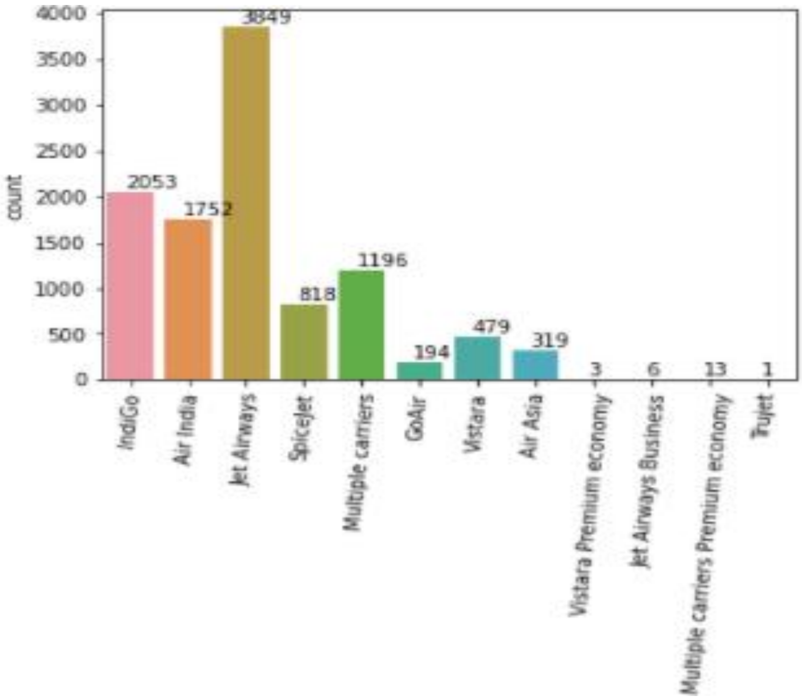
➢ EXPLOTERY DATA ANALYSIS(EDA)

Here we will try to visualize those patterns in data which are responsible for predicting fare and also will try to deep dive into all features to understand their relationship with themselves and also with dependant variable which is price columns.

- **Airline**

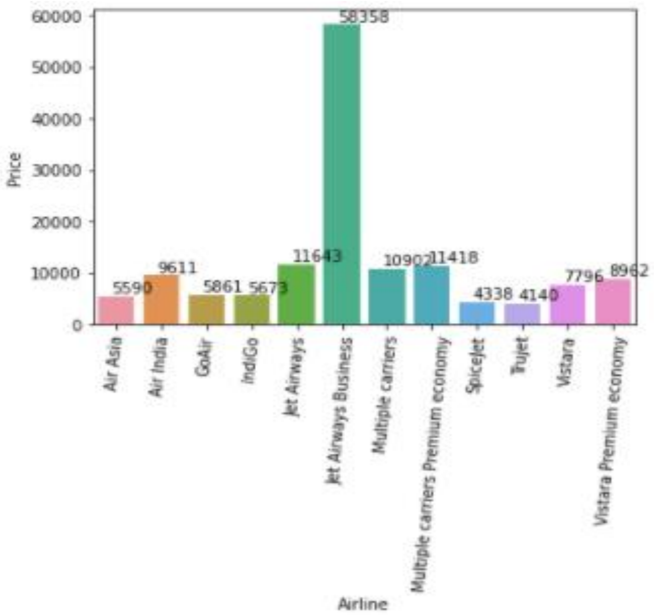
As shown in Fig 4, Jet Airways having maximum flight running multiple city then followed with Indigo and Air India.

Fig: 4



As shown in Fig 5, In economy class Jet Airways Avg price is high compare to other airlines.

Fig: 5

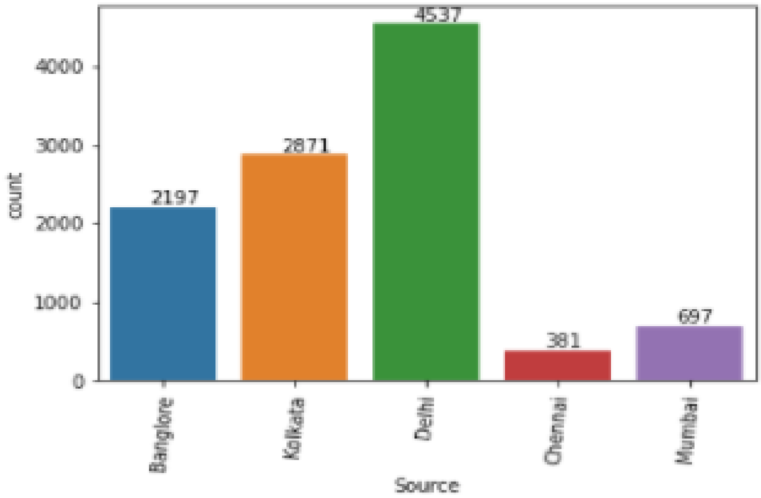


Airline	Price
Trujet	4140.000000
SpiceJet	4338.284841
Air Asia	5590.260188
IndiGo	5673.682903
GoAir	5861.056701
Vistara	7796.348643
Vistara Premium economy	8962.333333
Air India	9611.210616
Multiple carriers	10902.678094
Multiple carriers Premium economy	11418.846154
Jet Airways	11643.923357
Jet Airways Business	58358.666667

- **Source**

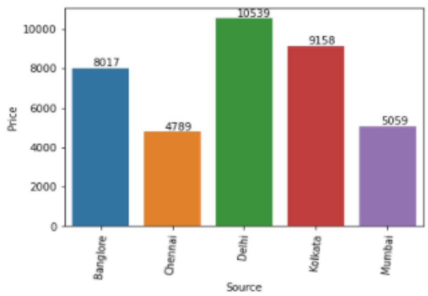
As shown in Fig 6, after visualizing the Source features we can see that Bangalore, Kolkata, Delhi, Chennai, Mumbai City as the source of airport city, where from Delhi airport maximum flight depart compare to another city.

Fig:6



As shown in Fig: 7, the avg price of source city for flight, as we can see that the avg price of flight ticket is high for those flight which source city is Delhi followed by Kolkata and Bangalore.

Fig:7



Source	Price
Chennai	4789.892388
Mumbai	5059.708752
Banglore	8017.464269
Kolkata	9158.389411
Delhi	10539.439057

- **Total Stops**

Here we can see that 1 Stop of flights are running more and very few flights having 3 or 4 stops, as shown in Fig 8.

Fig: 8

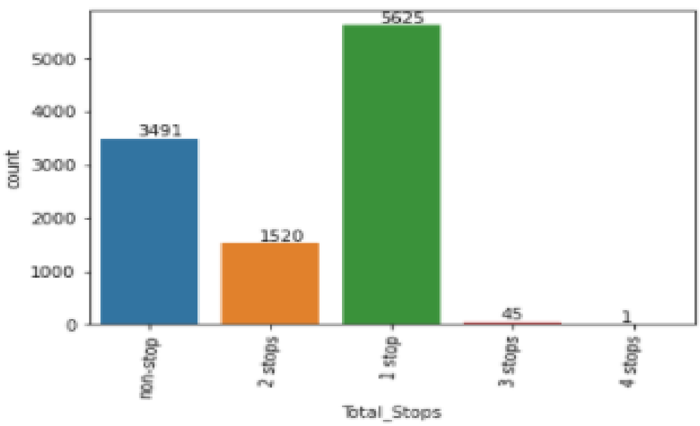
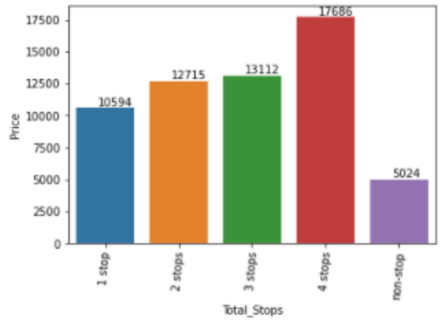


Fig: 9



Total_Stops	Price
non-stop	5024.900315
1 stop	10594.123556
2 stops	12715.807895
3 stops	13112.000000
4 stops	17686.000000

As shown in Fig 9, The Avg price of 4 stops flight is high and for non-stops flight is less. Here we can see some pattern in total stops where as the flight total stops increase the avg fare also increase.

- **Destination**

As shown in below Fig 10, The highest destination city for flight is Cochin followed by Bangalore & Delhi.

Fig: 10

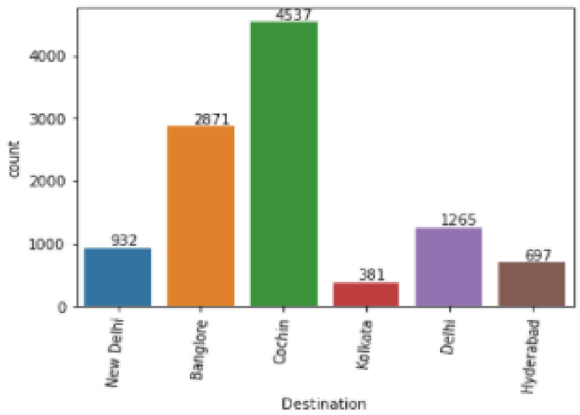
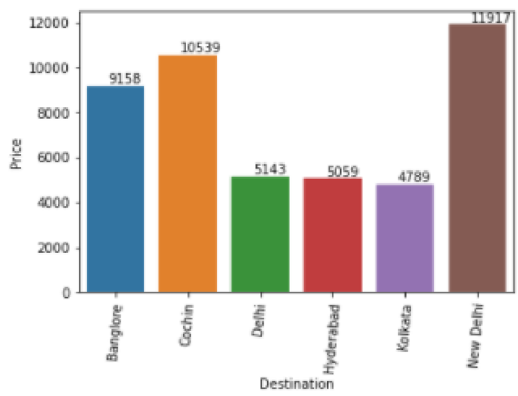


Fig:11



Destination	Price
Kolkata	4789.892388
Hyderabad	5059.708752
Delhi	5143.918577
Banglore	9158.389411
Cochin	10539.439057
New Delhi	11917.716738

As shown above Fig 11, The Destination flight of New Delhi City Avg fare is high followed by Cochin and Bangalore. Also, we can see that there is no source flight from Delhi and Cochin but having destination city for flight.

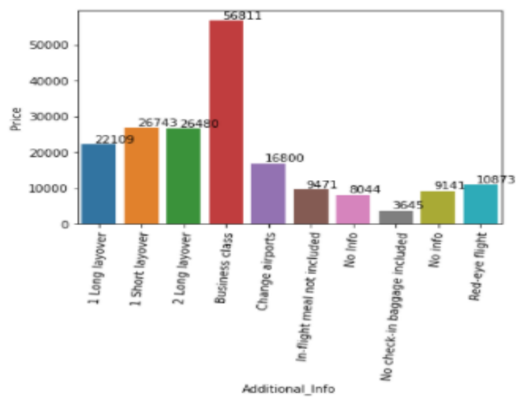
- **Additional Info**

As we can see that in maximum flight traveller has opt for in flight meal service, then no check-in baggage included and 1 long layover.

Fig: 12

```
Out[172]: No info                          8345
          In-flight meal not included      1982
          No check-in baggage included      320
          1 Long layover                     19
          Change airports                     7
          Business class                      4
          No Info                             3
          2 Long layover                      1
          Red-eye flight                      1
          1 Short layover                     1
          Name: Additional_Info, dtype: int64
```

Fig: 13



Additional_Info	Price
No check-in baggage included	3645.156250
No Info	8044.666667
No info	9141.320192
In-flight meal not included	9471.834511
Red-eye flight	10873.000000
Change airports	16800.714286
1 Long layover	22109.631579
2 Long layover	26480.000000
1 Short layover	26743.000000
Business class	56811.250000

As have seen in above Fig 13, The passenger who travelled in business class have paid more fare and then layover type also flight fare is high compare to Other services.

- **Date of Journey**

For our analysis from Date of Journey we have extracted only the month. As shown in Fig 14,

Fig: 14

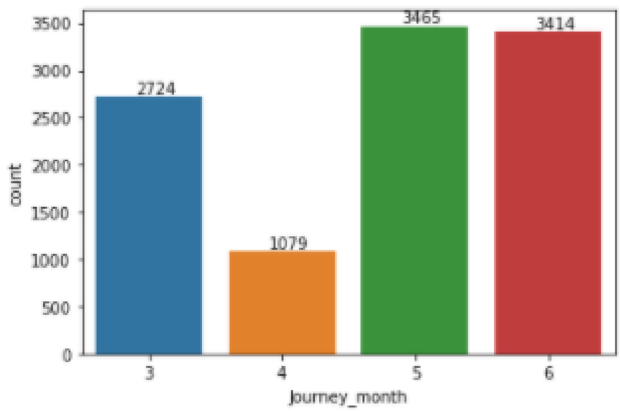
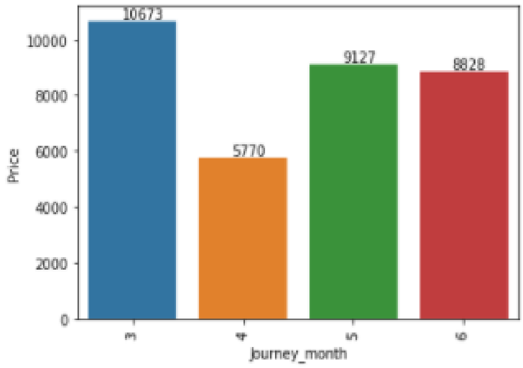


Fig: 15




```
Out[371]: Journey_month
          4     5770.847081
          6     8828.796134
          5     9127.722944
          3    10673.205580
          Name: Price, dtype: float64
```

As we can see in above Fig 15 that in the month of March the Avg fare of the flight is high followed by May and June.

➢ DATA PRE-PROCESSING

The Machine Learning Model Performance is not only depending on models and hyperparameters but also how we process and provide input of different types of variables to the model. Pre-processing of categorical variable is necessary steps as machine learning only accept numerical variables. As In this problem we have Training and Test Dataset separately so we need to do pre-processing step for both the dataset.

- ## Identifying and handling the missing values

After checking null value for Training and Test Dataset, we have seen that Test Dataset doesn't have null value but Training Dataset having some null value as shown in Fig 16. So, we need to deal with null values as an initial step.

Fig: 16

```
Out[205]: Airline            0
          Date_of_Journey    0
          Source             0
          Destination        0
          Route              1
          Dep_Time           0
          Arrival_Time       0
          Duration           0
          Total_Stops        1
          Additional_Info    0
          Price              0
          dtype: int64
```

Dealing with null values

As seen in Fig 16, that Route and Total Stops features having null value. Since row number 9039 having null values for both the features so will removed this particular row as shown in Fig 17.

Fig: 17

```
In [242]: df_train[df_train["Route"].isnull()]
Out[242]:
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
9039	Air India	6/05/2019	Delhi	Cochin	NaN	09:45	09:25 07 May	23h 40m	NaN	No info	7480

```
In [243]: df_train[df_train["Route"].isnull()]["Total_Stops"]
Out[243]: 9039    NaN
          Name: Total_Stops, dtype: object

In [244]: df_train=df_train.dropna(axis=0,subset=["Route"])
```

- # Feature Generation

In both the dataset training and test, Date of Journey, Arrival Time and Departure Time have Time and Date format data value, so we need to extract the information from these features to process our data into the model.

In Date of Journey columns, the date format is DD/MM/YYYY, since our training and test dataset belong to only year 2019, so we would not require year information from these columns we would need only date and month data from the feature as shown in below Fig 18.

Fig: 18

For Training Dataset

```
In [250]: df_train["Journey_Day"]=pd.to_datetime(df_train.Date_of_Journey,format="%d/%m/%Y").dt.day
          df_train["Journey_month"]=pd.to_datetime(df_train.Date_of_Journey,format="%d/%m/%Y").dt.month

In [251]: df_train.drop("Date_of_Journey",axis=1,inplace=True)
```

For Test Dataset

```
In [253]: df_test["Journey_Day"]=pd.to_datetime(df_test.Date_of_Journey,format="%d/%m/%Y").dt.day
          df_test["Journey_month"]=pd.to_datetime(df_test.Date_of_Journey,format="%d/%m/%Y").dt.month

In [254]: df_test.drop("Date_of_Journey",axis=1,inplace=True)
```

In Departure Time and Arrival Time columns we will extract information only Minute and Hour from both the features and dataset. i.e. training and test dataset as shown in Fig 19.

Fig: 19

For Training Dataset

```
In [255]: df_train["Dep_Hour"]=df_train["Dep_Time"].str.split(":").str[0]
          df_train["Dep_Minute"]=df_train["Dep_Time"].str.split(":").str[1]

In [256]: df_train["Dep_Hour"]=df_train["Dep_Hour"].astype(int)
          df_train["Dep_Minute"]=df_train["Dep_Minute"].astype(int)

In [257]: df_train.drop("Dep_Time",axis=1,inplace=True)
```

```
In [262]: df_train["Arrival_Time"]=df_train["Arrival_Time"].str.split(" ").str[0]
          df_train["Arr_Hour"]=df_train["Arrival_Time"].str.split(":").str[0]
          df_train["Arr_Minute"]=df_train["Arrival_Time"].str.split(":").str[1]
```

```
In [263]: df_train["Arr_Hour"]=df_train["Arr_Hour"].astype(int)
          df_train["Arr_Minute"]=df_train["Arr_Minute"].astype(int)
```

```
In [264]: df_train.drop("Arrival_Time",axis=1,inplace=True)
```

For Test Dataset

```
In [258]: df_test["Dep_Hour"]=df_test["Dep_Time"].str.split(":").str[0]
          df_test["Dep_Minute"]=df_test["Dep_Time"].str.split(":").str[1]
```

```
In [259]: df_test["Dep_Hour"]=df_test["Dep_Hour"].astype(int)
          df_test["Dep_Minute"]=df_test["Dep_Minute"].astype(int)
```

```
In [260]: df_test.drop("Dep_Time",axis=1,inplace=True)
```

```
In [265]: df_test["Arrival_Time"]=df_test["Arrival_Time"].str.split(" ").str[0]
          df_test["Arr_Hour"]=df_test["Arrival_Time"].str.split(":").str[0]
          df_test["Arr_Minute"]=df_test["Arrival_Time"].str.split(":").str[1]
```

```
In [266]: df_test["Arr_Hour"]=df_test["Arr_Hour"].astype(int)
          df_test["Arr_Minute"]=df_test["Arr_Minute"].astype(int)
```

```
In [267]: df_test.drop("Arrival_Time",axis=1,inplace=True)
```

In Both the Dataset having Duration features which represent the total duration of flight and data value have hour and minute format. So, for further process we would need to covert the data value in minute as shown in Fig 20.

Fig: 20

For Training Dataset

```
In [269]: df_train["Duration"]=df_train["Duration"].str.replace("h","*60").str.replace(" ","+").str.replace("m","*1").apply(eval)
```

```
In [270]: df_train["Duration"]
```

For Test Data

```
In [271]: df_test["Duration"]=df_test["Duration"].str.replace("h","*60").str.replace(" ","+").str.replace("m","*1").apply(eval)
```

```
In [272]: df_test["Duration"]
```

- # Feature Transformation

As we have seen before that our training and test dataset having object type data but we know that for processing our dataset into model we need to transform our data into numeric through various Encoding technique.

There are two types of Categorical data:

Ordinal Data:

Ordinal Data are those category where categorical variables has an inherent order.

Nominal Data:

Nominal Data are referring to category where categorical variable don’t have an inherent order.

Mentioned Below are the types of Encoding technique which we have used in this data:

- Ordinal Encoder:

We have encoded Total Stops categorical features to numerical features by using Ordinal Encoder. Since, these categories variable are in order. As example shown in Fig 21.

Fig: 21

For Training Dataset

```
In [298]: df_train["Total_Stops"].value_counts()
Out[298]: 1 stop      5625
          non-stop    3491
          2 stops     1520
          3 stops       45
          4 stops        1
          Name: Total_Stops, dtype: int64

In [299]: from sklearn.preprocessing import OrdinalEncoder

In [300]: enc=OrdinalEncoder(categories=[["non-stop","1 stop","2 stops","3 stops","4 stops"]])

In [301]: z=enc.fit_transform(df_train.Total_Stops.values.reshape(-1,1))

In [302]: df_train["Total_Stops"]=z

In [303]: df_train["Total_Stops"].value_counts()
Out[303]: 1.0    5625
          0.0    3491
          2.0    1520
          3.0      45
          4.0       1
          Name: Total_Stops, dtype: int64
```

For Test Dataset

```
In [304]: enc=OrdinalEncoder(categories=[["non-stop","1 stop","2 stops","3 stops","4 stops"]])
          z=enc.fit_transform(df_test.Total_Stops.values.reshape(-1,1))
          df_test["Total_Stops"]=z

In [305]: df_test["Total_Stops"].value_counts()
Out[305]: 1.0    1431
          0.0     849
          2.0     379
          3.0      11
          4.0       1
          Name: Total_Stops, dtype: int64

In [306]: df_train.shape,df_test.shape
Out[306]: ((10682, 26), (2671, 25))
```

- One Hot Encoding:

We have encoded Airline, Source, Additional Info and Destination categorical feature to numerical features with One Hot Encoding, since there is multiple variable with no order. As example shown in Fig 22.

Fig: 22

For Training Dataset

```
In [279]: from sklearn.preprocessing import OneHotEncoder

In [280]: onehotencoder=OneHotEncoder()

In [281]: x=onehotencoder.fit_transform(df_train.Airline.values.reshape(-1,1)).toarray()

In [282]: j=df_train["Airline"].value_counts()

In [283]: dfOnehot=pd.DataFrame(x,columns=["Airline_"+str(i)for i in j.index])

In [284]: df_train=pd.concat([df_train,dfOnehot],axis=1)

In [285]: df_train.drop("Airline",axis=1,inplace=True)
```

For Test Dataset

```
In [288]: onehotencoder=OneHotEncoder()
          x=onehotencoder.fit_transform(df_test.Airline.values.reshape(-1,1)).toarray()
          j=df_test["Airline"].value_counts()
          dfOnehot=pd.DataFrame(x,columns=["Airline_"+str(i)for i in j.index])
          df_test=pd.concat([df_test,dfOnehot],axis=1)

In [289]: df_test["Airline"].value_counts()

Out[289]: Jet Airways          897
          IndiGo               511
          Air India            440
          Multiple carriers    347
          SpiceJet             208
          Vistara              129
          Air Asia              86
          GoAir                 46
          Other                  7
          Name: Airline, dtype: int64

In [290]: df_test.drop("Airline",axis=1,inplace=True)
```

- ## Splitting Dataset for Model Building

As shown in Fig 23, we have separated train dataset into features and target variable (x and y) for processing the train dataset into model.

Fig: 23

```
In [350]: x=df_train.drop("Price",axis=1)

In [351]: y=df_train.Price

In [352]: x.shape

Out[352]: (10682, 34)

In [353]: y.shape

Out[353]: (10682,)
```

- ## Scale Transformation with Standard Scaler

We know that in this dataset Variable having different scale for measured and model do not contribute equally to the model fitting when Variable having different scales and model learned function and lead to creating a bias, so to avoid in this step we will transform our all features with Standard Scaler as shown in Fig 24.

Fig: 24

```
In [354]: from sklearn.preprocessing import StandardScaler

In [355]: scaler=StandardScaler()

In [356]: x_scaled=scaler.fit_transform(x)

In [357]: x_scaled.shape

Out[357]: (10682, 34)
```

- ## Splitting Train and Test Dataset

The dataset has been split into 70% for training and 30% for testing (shown in fig 25). The training dataset would be using to generate the model the chosen algorithms will use when exposed to new unseen data. The test data set is the final dataset which would be using to measure model performance based on some metrics.

Fig: 25

```
In [360]: x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.30,random_state=101)
```

➢ <u>BUILDING MACHINE LEARNING MODEL</u>

- ## Model:

For this analysis, we know that our target variable is continuous and the task is to predict the flight fare so we need to applied Regression model to analyse and compare their R2 score. Those models are listed below:

- ✓ Linear Regression
- ✓ KNN Regressor
- ✓ Random Forest Regressor
- ✓ Gradient Boosting Regressor

Regression is Supervised Machine Learning algorithm which works on finding the correlation among variables. A Regression task is applying when the output variable is a real or continuous value, here in our dataset our output variable is continuous.

Linear Regression:

Linear Regression shows the linear relationship between the independent variable and the target (dependent) variable. Here in our dataset, we have multiple input variable and the multiple linear regression task is to find the best fitted line which predict the flight fare for the model.

K-Nearest Neighbors (KNN) Regressor:

KNN algorithm works based on "feature similarity" to predict the value of any given new data points means new points is assigned a value based on how closely it resembles the points in the training set. This algorithm uses Euclidean and Manhattan Distance method for calculating the distance between the new points and each training point.

Random Forest Regressor:

Random Forest also is a Supervised Machine Learning Algorithm which builds decision tree based on different samples and takes their Majority Averages. This algorithm works on ensemble technique means combining multiple models or collection of models is used to predict.

Gradient Boosting Regressor:

Gradient Boosting is a ensemble machine learning algorithms that can used for classification or regression predictive modelling task.
Here ensembles are built from decision tree models. Trees are added one at a time to the ensemble and fit to correct the prediction error which made by prior models. Such type of ensemble machine learning model referred to as boosting.

- ## Evaluation Matrix:

Here we will evaluate our model performance with R2 Score and error will check with MAE (Mean Absolute Error).

After applying mentioned above model to our training and test dataset, we are getting R2 score for Linear Regression is 67.29%, KNN Regressor 78.84%, Random Forest Regressor is 87.75, Gradient Boosting Regressor is 82.06%.

Linear Regression:

R2 Score = 67.29%
MAE = 1821.64

While Checking mean of our target variable (Price) is 9087.as shown in Fig 26.

Fig: 26

```
In [589]: df_train["Price"].mean()

Out[589]: 9087.21456656057
```

As we know that we should always look for minimize MAE, after checking the target variable mean value 9087.21 and MAE value is 1821.64, it seems that error is more while comparing with mean value of target variable, which is almost 20% of mean value of target variable.

KNN Regressor:

R2 Score = 78.84%
MAE = 1070.31

In case of KNN Regressor, model is performing better than Linear Regression based on the model performance (78.84%) and MAE value (1070.31).
It looks like KNN's model MAE is almost 11% of mean value of target variable which is lesser than Liner Regression model.

Random Forest Regressor:

R2 Score = 87.87%
MAE = 713.8

In case of Random Forest Regressor, model is performing better than Linear Regression and KNN Regressor based on the model performance (87.75%) and MAE Value (713.8)

It looks like Random Forest Regressor model is almost 7.8% of mean value of target variable which is lesser than Linear Regression and KNN Regressor.

Gradient Boosting Regressor:

R2 Score= 82.05%
MAE = 1278.62

In case of Gradient Boosting Regressor, model is performing well compare to Linear Regressor and KNN Regressor, but now performing well than Random Forest Regressor. The model performance 82.05% and MAE Value is 1278.62.

- ## Choosing Best Model by Comparing Cross Validation Score:

Here, we will compare the applied model in training dataset and calculate the difference between on R2 Score and CV Score to ensure that model is not overfitting and has been checked at difference cross validation. Also we have analysed our model based on Error Value (MAE) to identify the best model for the training dataset as shown in Fig 27.

Fig: 27

Out[664]:

	Score	Cross Validation Score	Difference	MAE
LinearRegression	0.672903	0.647538	0.025365	1821.645167
KNeighborsRegressor	0.788440	0.766243	0.022197	1070.285117
RandomForestRegressor	0.878738	0.863598	0.015141	713.800786
GradientBoostRegressor	0.820587	0.859294	-0.038707	1278.624543

As shown in Fig 27, we can see that Random Forest Regressor model is working better than Linear Regression, KNN Regressor and Gradient Boost Regressor. Also, we have seen that Random Forest Regressor MAE value is better than any other model. So, we will choose Random Forest Regressor for this dataset. Further will see whether we can improve the performance of Random Forest Regressor with Hyper tuning.

- ## Hyper tuning the best model:

After Checking Cross Validation Score and MAE Random Forest Regressor model is performing well, so we will try to improve model accuracy with hyper tuning as shown in Fig 27.

Fig: 27

```
In [481]: from sklearn.model_selection import GridSearchCV

In [482]: param={"min_samples_split":[2,6,8],
                 "min_samples_leaf":[1,3,5,7,9],
                 "n_estimators":[150,200]
                 }

In [483]: RandomForestRegressor()

Out[483]: RandomForestRegressor()

In [484]: grd=GridSearchCV(rfr,param_grid=param)

In [485]: grd.fit(x_train,y_train)

Out[485]: GridSearchCV(estimator=RandomForestRegressor(),
                       param_grid={'min_samples_leaf': [1, 3, 5, 7, 9],
                                   'min_samples_split': [2, 6, 8],
                                   'n_estimators': [150, 200]})

In [486]: grd.best_params_
```

After doing hyper tuning of Random Forest Regressor model, the accuracy score has not improved (87.80) so we can say model is performing well with default parameter.

- ## Actual Vs Predicted

Here, we will visualize the Actual Vs Predicted value. we will use our test data and see how accurately our algorithm predicts the percentage score.

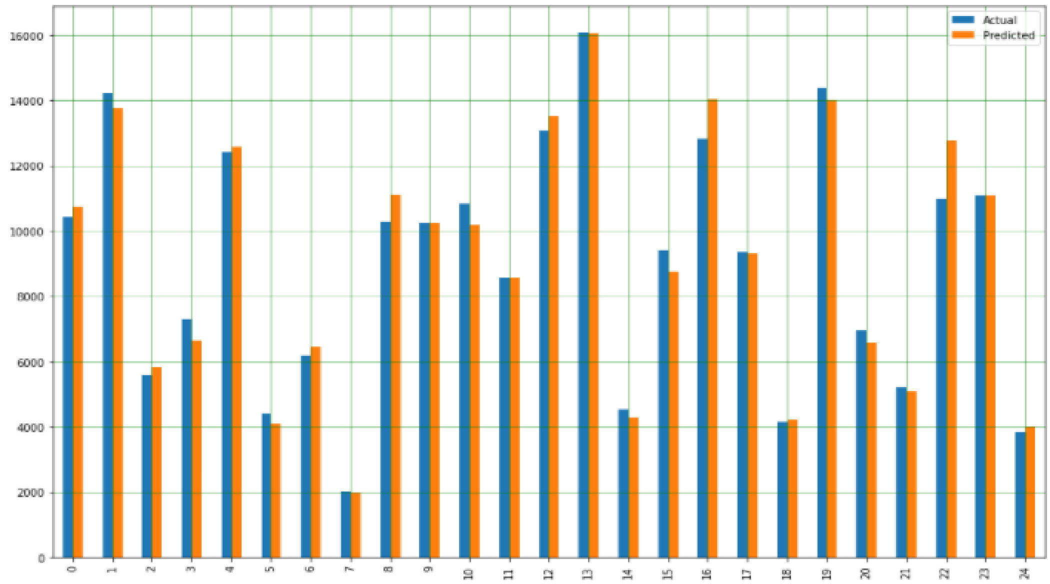
Fig: 28

8]:

	Actual	Predicted
0	10441	10746.868452
1	14231	13746.824593
2	5601	5832.709500
3	7295	6643.412611
4	12436	12582.852294
...	...	...
3200	11528	14666.456500
3201	4823	4715.243111
3202	5092	6196.156111
3203	6961	6813.313333
3204	5177	6813.076222

As row number is huge we have visualized first 25 Rows as shown in Fig 29.

Fig: 29



➢ PREDICTION ON TEST DATASET

As we know that there are two datasets provided to us. So, after building our model successfully and chosen the best model for our dataset now we need to use Random Forest Regressor model to predict on given test dataset as shown in Fig 30

Fig: 30

```
In [1082]: df_test.shape
Out[1082]: (2671, 34)

In [1083]: x=df_test

In [1084]: scaler=StandardScaler()

In [1085]: x_scaled=scaler.fit_transform(x)

In [1086]: x_scaled.shape
Out[1086]: (2671, 34)

In [1087]: prediction=rfr.predict(x_scaled)

In [1088]: prediction
Out[1088]: array([14630.99177778,  4250.12216667, 12898.        , ...,
                  15950.63333333, 14746.77036111,  7434.58677778])
```

➢ CONCLUSION

As we know that we have two separate dataset, training dataset and test dataset. In this project we have done pre-processing step on both the dataset. But while training model we have used only training dataset. We have built model by using different Regressor algorithm like Linear Regressor, KNN Regressor, Random Forest Regressor and Gradient Boosting Regressor.

After training these models we have evaluated our model based on MAE (mean absolute error) and Cross Validation Score and finally we have chosen our best model as Random Forest Regressor.

As final stage we have predicted test data which has given separately applied to the Random Forest Regressor.