FLIP ROBO

Flight Price Prediction

Submitted by:

Jyuthika Mankar

ACKNOWLEDGMENT

I am thankful to Flip Robo Technologies, situated in Bengaluru for giving me this opportunity to work as an Intern in their company. The experience which I am accumulating with the kind of projects Flip Robo Technologies is providing are nurturing. “Flight Price Prediction” is one of the projects given to me for Web Scrapping, Machine Learning and Data Analysis. The dataset has been scrapped from multiple websites and working on that data has been proved to be a learning experience in every way. I would like to thank my SME Ms Sapna Verma for being considerate always to the interns and resolving the issues whenever they arose regarding the current project.

INTRODUCTION

- Business Problem Framing

  The airline industry is considered as one of the most sophisticated industries in using complex pricing strategies. Nowadays, ticket prices can vary dynamically and significantly for the same flight, even for nearby seats. The ticket price of a specific flight can change up to 7 times a day. Customers are seeking to get the lowest price for their ticket, while airline companies are trying to keep their overall revenue as high as possible and maximize their profit. However, mismatches between available seats and passenger demand usually leads to either the customer paying more or the airlines company losing revenue. Airlines companies are generally equipped with advanced tools and capabilities that enable them to control the pricing process. However, customers are also becoming more strategic with the development of various online tools to compare prices across various airline companies. In addition, competition between airlines makes the task of determining optimal pricing is hard for everyone.

  Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on:

– Time of purchase patterns (making sure last-minute purchases are expensive)
– Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases)

- Conceptual Background of the Domain Problem

  The airline implements dynamic pricing for the flight ticket. According to the survey, flight ticket prices change during the morning and evening time of the day. Also, it changes with the holidays or festival season. There are several different factors on which the price of the flight ticket depends. The seller has information about all the factors, but buyers are able to access limited information only which is not enough to predict the airfare prices. Considering the features such as departure time, the number of days left for departure and time of the day it will give the best time to buy the ticket. The purpose of this is to study the factors which influence the fluctuations in the airfare prices and how they are related to the change in the prices. Then using this information, build a system that can help buyers whether to buy a ticket or not.

- ## Review of Literature

  Data was scrapped from bookings.com website between August-November 2022 and it was later analysed and studied to predict the flight fares. A paper titled 'A Survey on Flight Pricing Prediction using Machine Learning' by S. Rajankar, N. Sakharkar proposed the overall survey for the dynamic price changes in the flight tickets which gives the information about the highs and lows in the airfares according to the days, weekend and time of the day that is morning, evening and night. For the prediction of the ticket prices perfectly different prediction models are tested for the better prediction accuracy. As the pricing models of the company are developed in order to maximize the revenue management. So, to get result with maximum accuracy regression analysis is used. From the studies, the feature that influences the prices of the ticket are to be considered. In future the details about number of available seats can improve the performance of the model.

- ## Motivation for the Problem Undertaken

  With airfares fluctuating frequently, knowing when to buy and when to wait for a better deal to come along is tricky. The fluctuation in prices is frequent and one has limited time to book the cheapest ticket as the prices keep varying due to constant manipulation by Airline companies. Therefore, it is necessary to work on a predictive model based on deterministic and aggregate feature data that would predict with good accuracy the most optimal Air fare for a particular destination, route and schedule.

Analytical Problem Framing

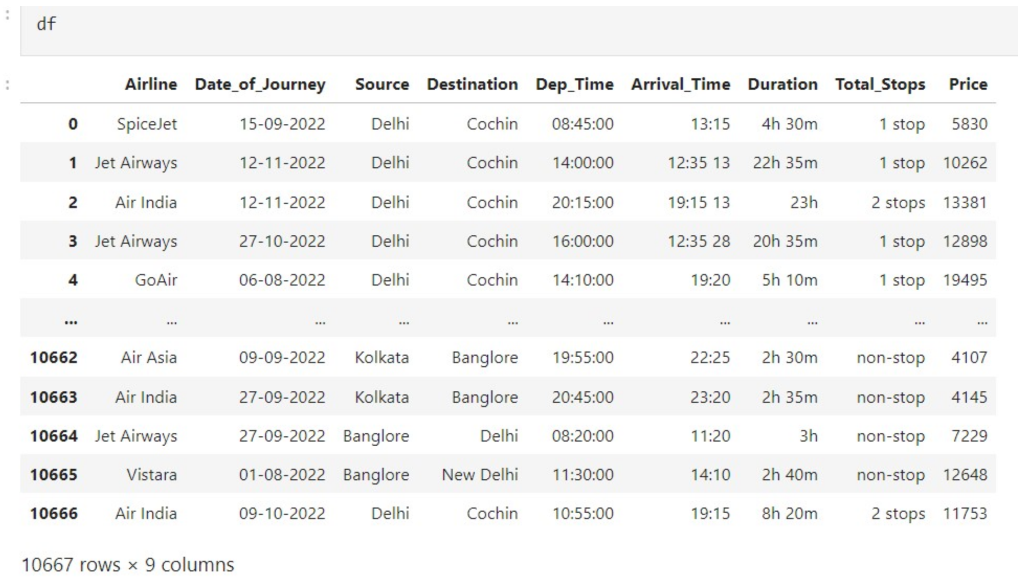
- ### Mathematical/ Analytical Modelling of the Problem

  Various Regression analysis techniques were used to build predictive models to understand the relationships that exist between Flight ticket price and Deterministic and Aggregate features of Air travel. The Regression analysis models were used to predict the Flight ticket price value for changes in Air travel deterministic and aggregate attributes. Regression modelling techniques were used in this Problem since Air Ticket Price data distribution is continuous in nature. In order to forecast Flight Ticket price, predictive models such as ridge regression Model, Random Forest Regression model, Decision tree Regression Model, Support Vector Machine Regression model and Extreme Gradient Boost Regression model were used to describe how the values of Flight Ticket Price depended on the independent variables of various Air Fare attributes.

- ### Data Sources and their formats

  At least 1500 rows of data were to be scrapped from different websites (yatra.com, skyscanner.com, official websites of airlines, etc). The data in this model building was scrapped from bookings.com and was stored in the excel as well as .csv format. The data was loaded from the uploaded csv file in jupyter notebook and studied for EDA and model building.
  The columns in the dataset were airline name, date of journey, source, destination, departure time, arrival time, duration, total stops and the target variable price.

```
#Loading the dataset
df=pd.read_csv("https://raw.githubusercontent.com/jman2031/FR-Flight--Pred/main/Flight%20Price-1.txt")
```



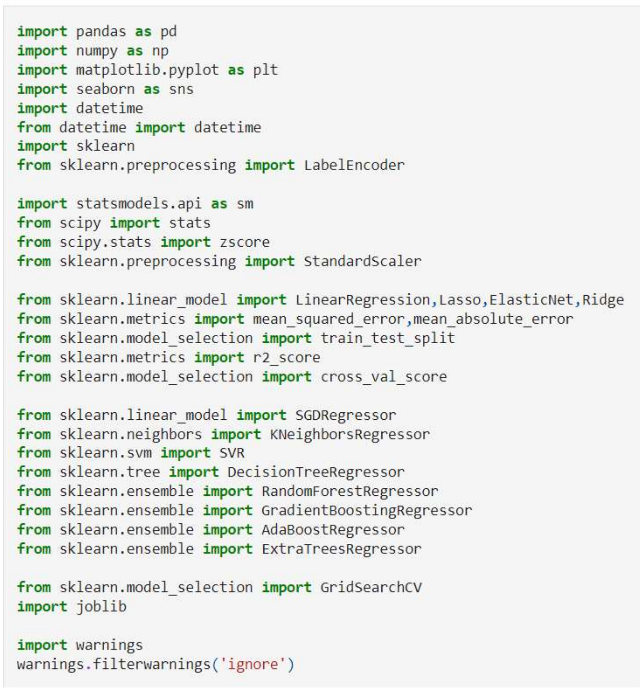
```
df
```

	Airline	Date_of_Journey	Source	Destination	Dep_Time	Arrival_Time	Duration	Total_Stops	Price
0	SpiceJet	15-09-2022	Delhi	Cochin	08:45:00	13:15	4h 30m	1 stop	5830
1	Jet Airways	12-11-2022	Delhi	Cochin	14:00:00	12:35 13	22h 35m	1 stop	10262
2	Air India	12-11-2022	Delhi	Cochin	20:15:00	19:15 13	23h	2 stops	13381
3	Jet Airways	27-10-2022	Delhi	Cochin	16:00:00	12:35 28	20h 35m	1 stop	12898
4	GoAir	06-08-2022	Delhi	Cochin	14:10:00	19:20	5h 10m	1 stop	19495
...	...	...	...	...	...	...	...	...	...
10662	Air Asia	09-09-2022	Kolkata	Banglore	19:55:00	22:25	2h 30m	non-stop	4107
10663	Air India	27-09-2022	Kolkata	Banglore	20:45:00	23:20	2h 35m	non-stop	4145
10664	Jet Airways	27-09-2022	Banglore	Delhi	08:20:00	11:20	3h	non-stop	7229
10665	Vistara	01-08-2022	Banglore	New Delhi	11:30:00	14:10	2h 40m	non-stop	12648
10666	Air India	09-10-2022	Delhi	Cochin	10:55:00	19:15	8h 20m	2 stops	11753

10667 rows × 9 columns

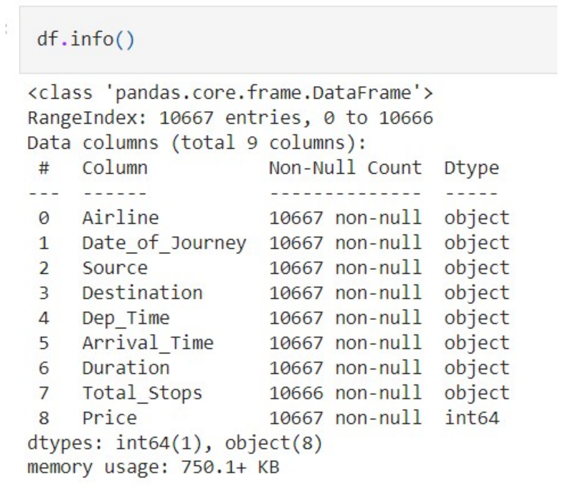
- Data Pre-processing

  For solving the problem, we first import different libraries as per the need. Here we are using the Regression method because our variable column (Price) is having continuous data, and not binary values.



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
from datetime import datetime
import sklearn
from sklearn.preprocessing import LabelEncoder

import statsmodels.api as sm
from scipy import stats
from scipy.stats import zscore
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression,Lasso,ElasticNet,Ridge
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor

from sklearn.model_selection import GridSearchCV
import joblib

import warnings
warnings.filterwarnings('ignore')
```

The data was loaded in the Jupyter notebook and following observations were made with respect to the dataset, number and names of the columns:



```
df.info()
```

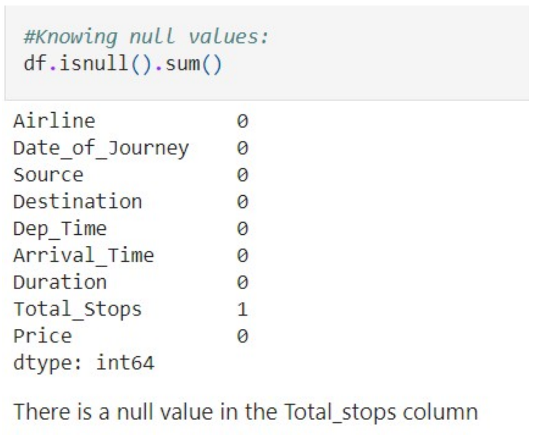
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10667 entries, 0 to 10666
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10667 non-null  object
 1   Date_of_Journey  10667 non-null  object
 2   Source           10667 non-null  object
 3   Destination      10667 non-null  object
 4   Dep_Time         10667 non-null  object
 5   Arrival_Time     10667 non-null  object
 6   Duration         10667 non-null  object
 7   Total_Stops      10666 non-null  object
 8   Price            10667 non-null  int64 
dtypes: int64(1), object(8)
memory usage: 750.1+ KB
```

FEATURES:

- Airline: The name of the airline.
- Date_of_Journey: The date of the journey
- Source: The source from which the service begins.
- Destination: The destination where the service ends.
- Dep_Time: The time when the journey starts from the source.
- Arrival_Time: Time of arrival at the destination.
- Duration: Total duration of the flight.
- Total_Stops: Total stops between the source and destination.
- Price: The price of the ticket

The dataset had all the columns in object data-type format except the Price column which was in integer format.

Checking null values showed that there was one 1 null value present in the Total_Stops column which was filled with the imputation method.

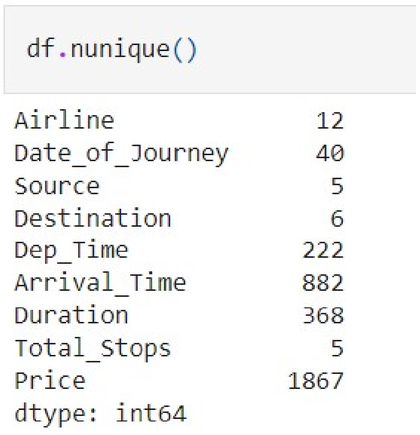


```
#Knowing null values:
df.isnull().sum()
```

```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Price              0
dtype: int64
```

There is a null value in the Total_stops column

The unique elements in each column were found out to understand the dataset in a better way.

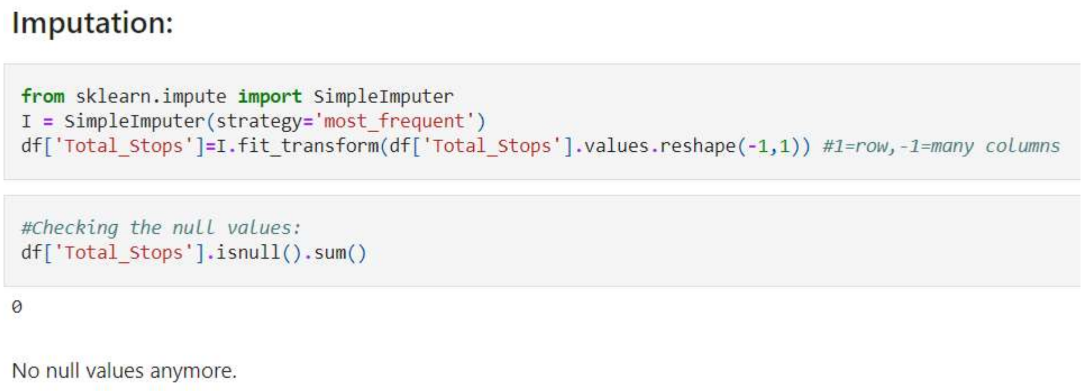


```
df.nunique()
```

```
Airline              12
Date_of_Journey      40
Source                5
Destination           6
Dep_Time            222
Arrival_Time        882
Duration            368
Total_Stops           5
Price              1867
dtype: int64
```

- Data Inputs- Logic- Output Relationships

We filled the null values using imputation as shown below by using Simple Imputer.



Imputation:

```
from sklearn.impute import SimpleImputer
I = SimpleImputer(strategy='most_frequent')
df['Total_Stops']=I.fit_transform(df['Total_Stops'].values.reshape(-1,1)) #1=row,-1=many columns
```

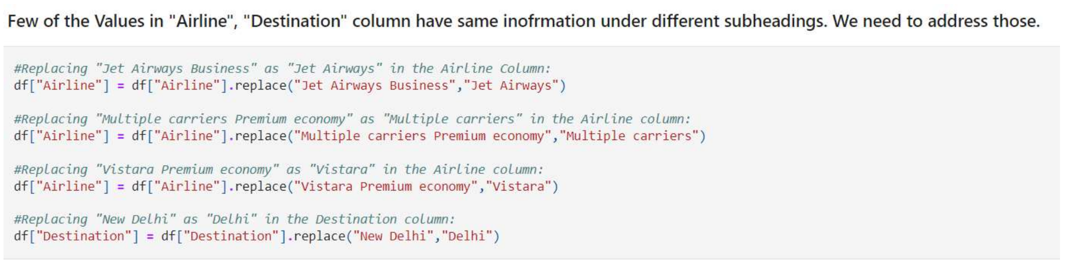
```
#Checking the null values:
df['Total_Stops'].isnull().sum()
```

```
0
```

No null values anymore.

After re-checking the null values, no null values were found.

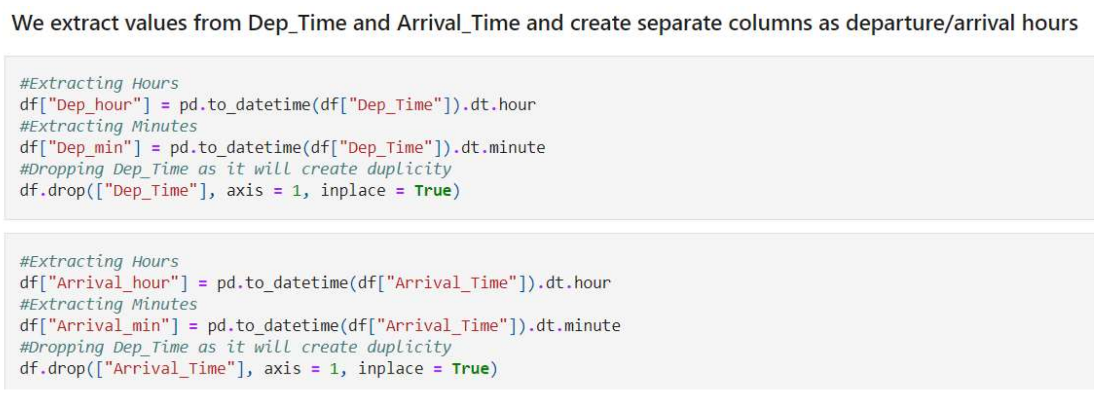
The Airlines columns had few duplicity/redundancies, so to remove it we clustered the same headings under one heading.



Few of the Values in "Airline", "Destination" column have same inofrmation under different subheadings. We need to address those.

```
#Replacing "Jet Airways Business" as "Jet Airways" in the Airline Column:
df["Airline"] = df["Airline"].replace("Jet Airways Business","Jet Airways")

#Replacing "Multiple carriers Premium economy" as "Multiple carriers" in the Airline column:
df["Airline"] = df["Airline"].replace("Multiple carriers Premium economy","Multiple carriers")

#Replacing "Vistara Premium economy" as "Vistara" in the Airline column:
df["Airline"] = df["Airline"].replace("Vistara Premium economy","Vistara")

#Replacing "New Delhi" as "Delhi" in the Destination column:
df["Destination"] = df["Destination"].replace("New Delhi","Delhi")
```

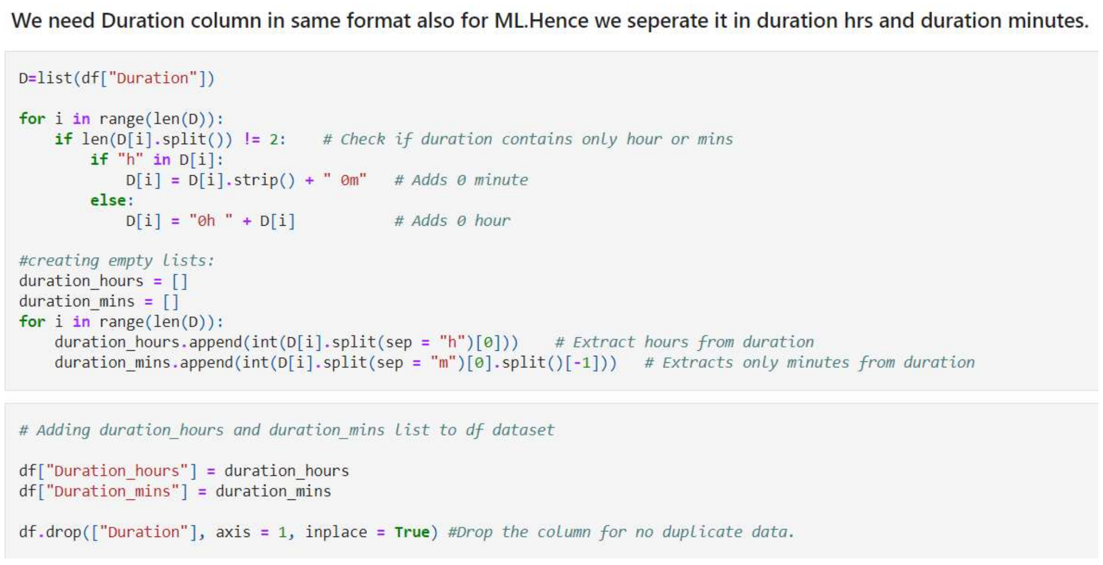
We also changed the arrival time and Departure time in hours and minutes format separately, for Machine Learning.


We extract values from Dep_Time and Arrival_Time and create separate columns as departure/arrival hours

```
#Extracting Hours
df["Dep_hour"] = pd.to_datetime(df["Dep_Time"]).dt.hour
#Extracting Minutes
df["Dep_min"] = pd.to_datetime(df["Dep_Time"]).dt.minute
#Dropping Dep_Time as it will create duplicity
df.drop(["Dep_Time"], axis = 1, inplace = True)
```

```
#Extracting Hours
df["Arrival_hour"] = pd.to_datetime(df["Arrival_Time"]).dt.hour
#Extracting Minutes
df["Arrival_min"] = pd.to_datetime(df["Arrival_Time"]).dt.minute
#Dropping Dep_Time as it will create duplicity
df.drop(["Arrival_Time"], axis = 1, inplace = True)
```

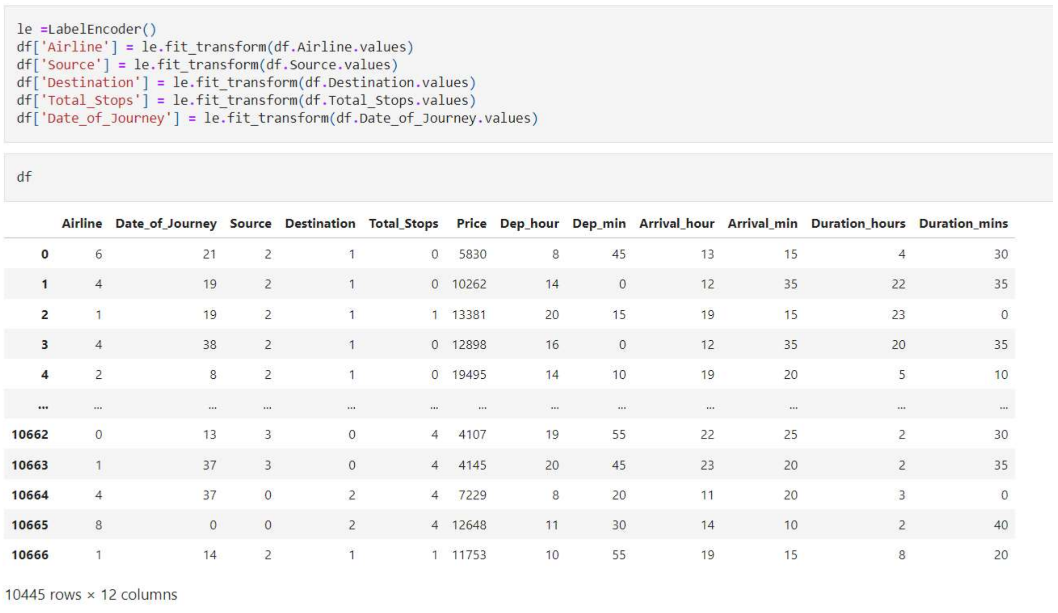
The duration column was split in Duration hours and duration minutes too similarly.


We need Duration column in same format also for ML.Hence we seperate it in duration hrs and duration minutes.

```
D=list(df["Duration"])

for i in range(len(D)):
    if len(D[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in D[i]:
            D[i] = D[i].strip() + " 0m"   # Adds 0 minute
        else:
            D[i] = "0h " + D[i]           # Adds 0 hour

#creating empty lists:
duration_hours = []
duration_mins = []
for i in range(len(D)):
    duration_hours.append(int(D[i].split(sep = "h")[0]))    # Extract hours from duration
    duration_mins.append(int(D[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration
```

```
# Adding duration_hours and duration_mins list to df dataset

df["Duration_hours"] = duration_hours
df["Duration_mins"] = duration_mins

df.drop(["Duration"], axis = 1, inplace = True) #Drop the column for no duplicate data.
```

For ML, the categorical columns in the dataset were encoded using Label encoder.


```
le =LabelEncoder()
df['Airline'] = le.fit_transform(df.Airline.values)
df['Source'] = le.fit_transform(df.Source.values)
df['Destination'] = le.fit_transform(df.Destination.values)
df['Total_Stops'] = le.fit_transform(df.Total_Stops.values)
df['Date_of_Journey'] = le.fit_transform(df.Date_of_Journey.values)
```

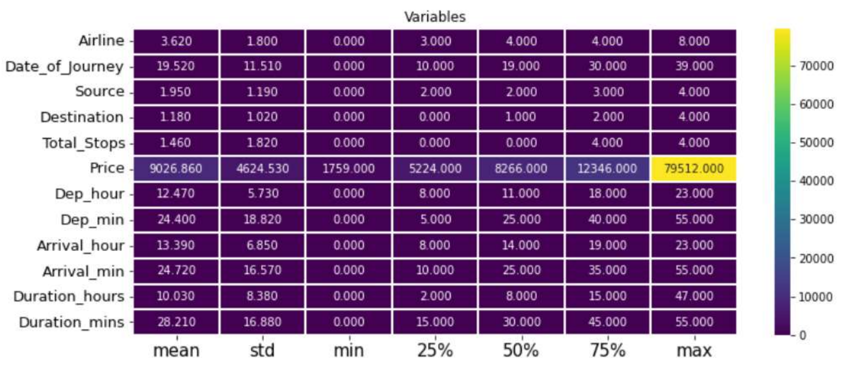
```
df
```

	Airline	Date_of_Journey	Source	Destination	Total_Stops	Price	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins
0	6	21	2	1	0	5830	8	45	13	15	4	30
1	4	19	2	1	0	10262	14	0	12	35	22	35
2	1	19	2	1	1	13381	20	15	19	15	23	0
3	4	38	2	1	0	12898	16	0	12	35	20	35
4	2	8	2	1	0	19495	14	10	19	20	5	10
...	...	...	...	...	...	...	...	...	...	...	...	...
10662	0	13	3	0	4	4107	19	55	22	25	2	30
10663	1	37	3	0	4	4145	20	45	23	20	2	35
10664	4	37	0	2	4	7229	8	20	11	20	3	0
10665	8	0	0	2	4	12648	11	30	14	10	2	40
10666	1	14	2	1	1	11753	10	55	19	15	8	20

10445 rows × 12 columns

EDA was performed, where followings were observed.

```
#plotting heat map for better understanding:
plt.figure(figsize=(12,5))
sns.heatmap(round(df.describe()[1:].transpose(),2),linewidth=2, annot=True,fmt='0.3f',cmap='viridis')
plt.xticks(fontsize=15)
plt.yticks(fontsize=13)
plt.title("Variables")
plt.show()
```



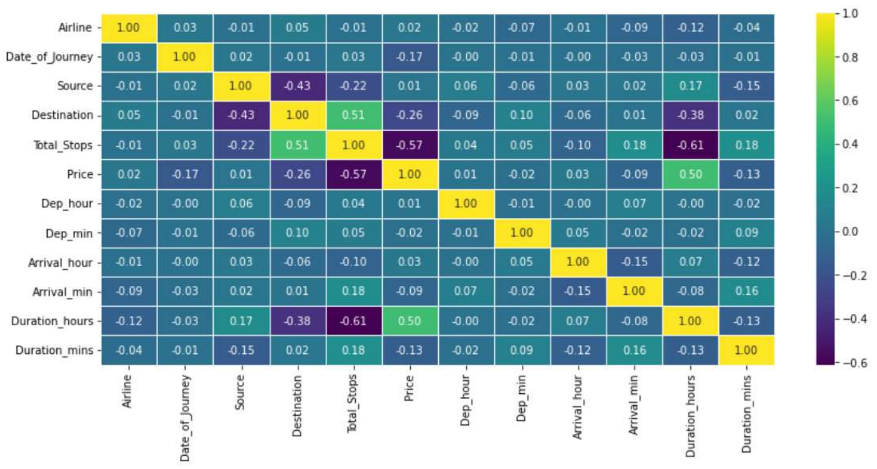
	mean	std	min	25%	50%	75%	max
Airline	3.620	1.800	0.000	3.000	4.000	4.000	8.000
Date_of_Journey	19.520	11.510	0.000	10.000	19.000	30.000	39.000
Source	1.950	1.190	0.000	2.000	2.000	3.000	4.000
Destination	1.180	1.020	0.000	0.000	1.000	2.000	4.000
Total_Stops	1.460	1.820	0.000	0.000	0.000	4.000	4.000
Price	9026.860	4624.530	1759.000	5224.000	8266.000	12346.000	79512.000
Dep_hour	12.470	5.730	0.000	8.000	11.000	18.000	23.000
Dep_min	24.400	18.820	0.000	5.000	25.000	40.000	55.000
Arrival_hour	13.390	6.850	0.000	8.000	14.000	19.000	23.000
Arrival_min	24.720	16.570	0.000	10.000	25.000	35.000	55.000
Duration_hours	10.030	8.380	0.000	2.000	8.000	15.000	47.000
Duration_mins	28.210	16.880	0.000	15.000	30.000	45.000	55.000

- The difference between 75% and max is a lot in duration hours, Arrival Minutes, Arrival hour, Dep Minute, departure hour, Price. This indicates that there might be outliers in these columns. But these columns are not numerical so, we won't consider the outliers in those.
- The mean > Median in Duration hours, departure hour, Price which means data is right skewed.
- Std deviation is high in Price column meaning the data is highly spread. But Price column is the target column so we can ignore the deviation.

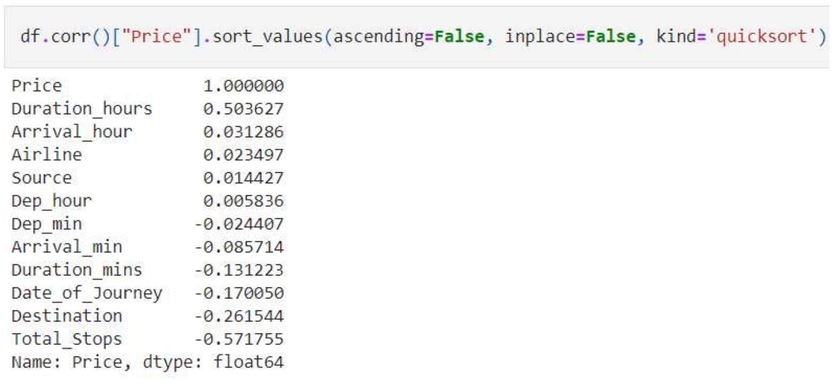
Correlation of each column with the target variable was found out as follows:

```
plt.figure(figsize=(14,6))
sns.heatmap(df.corr(),cmap='viridis',annot=True,linewidth=0.5,fmt='0.2f')
```

```
<AxesSubplot:>
```



	Airline	Date_of_Journey	Source	Destination	Total_Stops	Price	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins
Airline	1.00	0.03	-0.01	0.05	-0.01	0.02	-0.02	-0.07	-0.01	-0.09	-0.12	-0.04
Date_of_Journey	0.03	1.00	0.02	-0.01	0.03	-0.17	-0.00	-0.01	-0.00	-0.03	-0.03	-0.01
Source	-0.01	0.02	1.00	-0.43	-0.22	0.01	0.06	-0.06	0.03	0.02	0.17	-0.15
Destination	0.05	-0.01	-0.43	1.00	0.51	-0.26	-0.09	0.10	-0.06	0.01	-0.38	0.02
Total_Stops	-0.01	0.03	-0.22	0.51	1.00	-0.57	0.04	0.05	-0.10	0.18	-0.61	0.18
Price	0.02	-0.17	0.01	-0.26	-0.57	1.00	0.01	-0.02	0.03	-0.09	0.50	-0.13
Dep_hour	-0.02	-0.00	0.06	-0.09	0.04	0.01	1.00	-0.01	-0.00	0.07	-0.00	-0.02
Dep_min	-0.07	-0.01	-0.06	0.10	0.05	-0.02	-0.01	1.00	0.05	-0.02	-0.02	0.09
Arrival_hour	-0.01	-0.00	0.03	-0.06	-0.10	0.03	-0.00	0.05	1.00	-0.15	0.07	-0.12
Arrival_min	-0.09	-0.03	0.02	0.01	0.18	-0.09	0.07	-0.02	-0.15	1.00	-0.08	0.16
Duration_hours	-0.12	-0.03	0.17	-0.38	-0.61	0.50	-0.00	-0.02	0.07	-0.08	1.00	-0.13
Duration_mins	-0.04	-0.01	-0.15	0.02	0.18	-0.13	-0.02	0.09	-0.12	0.16	-0.13	1.00



```
df.corr()["Price"].sort_values(ascending=False, inplace=False, kind='quicksort')
```

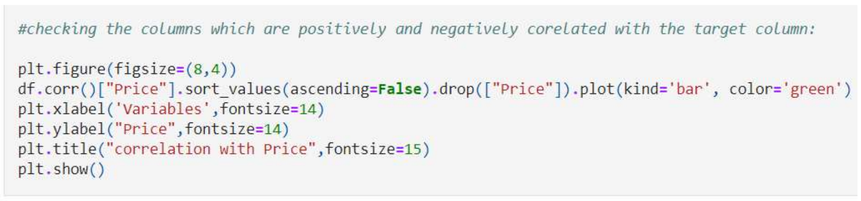
```
Price             1.000000
Duration_hours    0.503627
Arrival_hour      0.031286
Airline           0.023497
Source            0.014427
Dep_hour          0.005836
Dep_min          -0.024407
Arrival_min      -0.085714
Duration_mins    -0.131223
Date_of_Journey  -0.170050
Destination      -0.261544
Total_Stops      -0.571755
Name: Price, dtype: float64
```

Our target column is "Price": - Highest positive correlation is seem with the following columns: Duration_hours

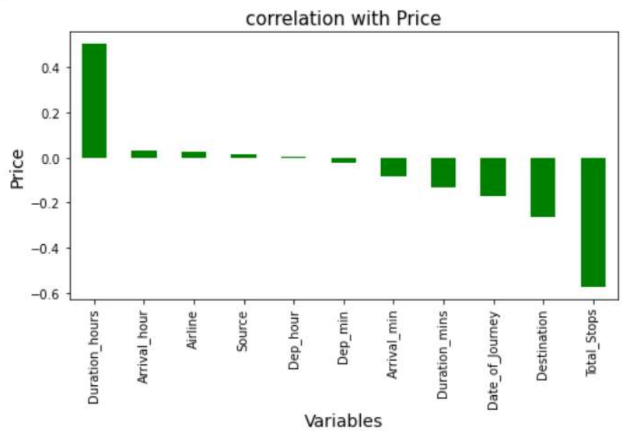
Almost neutral correlation: Arrival_hour, Airline, Source, Dep_hour

While negative correlation is seemed with the columns: Journey_day, Destination, Total_Stops.

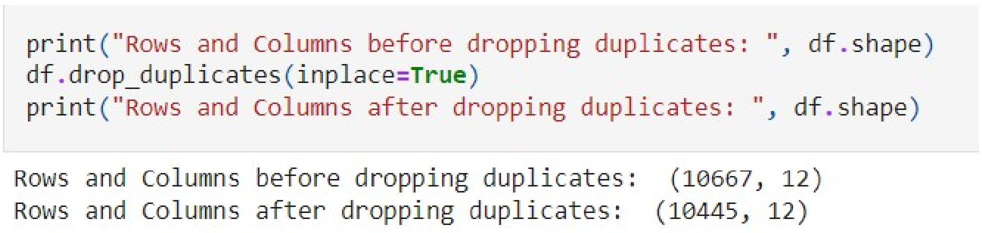
With the help of a bar graph, it can be well understood.



```
#checking the columns which are positively and negatively corelated with the target column:

plt.figure(figsize=(8,4))
df.corr()["Price"].sort_values(ascending=False).drop(["Price"]).plot(kind='bar', color='green')
plt.xlabel('Variables',fontsize=14)
plt.ylabel("Price",fontsize=14)
plt.title("correlation with Price",fontsize=15)
plt.show()
```



- State the set of assumptions (if any) related to the problem under consideration
    - Since the dataset is big, there might be some duplicate values. Hence, we had to remove those too.



```
print("Rows and Columns before dropping duplicates: ", df.shape)
df.drop_duplicates(inplace=True)
print("Rows and Columns after dropping duplicates: ", df.shape)
```

```
Rows and Columns before dropping duplicates:  (10667, 12)
Rows and Columns after dropping duplicates:  (10445, 12)
```

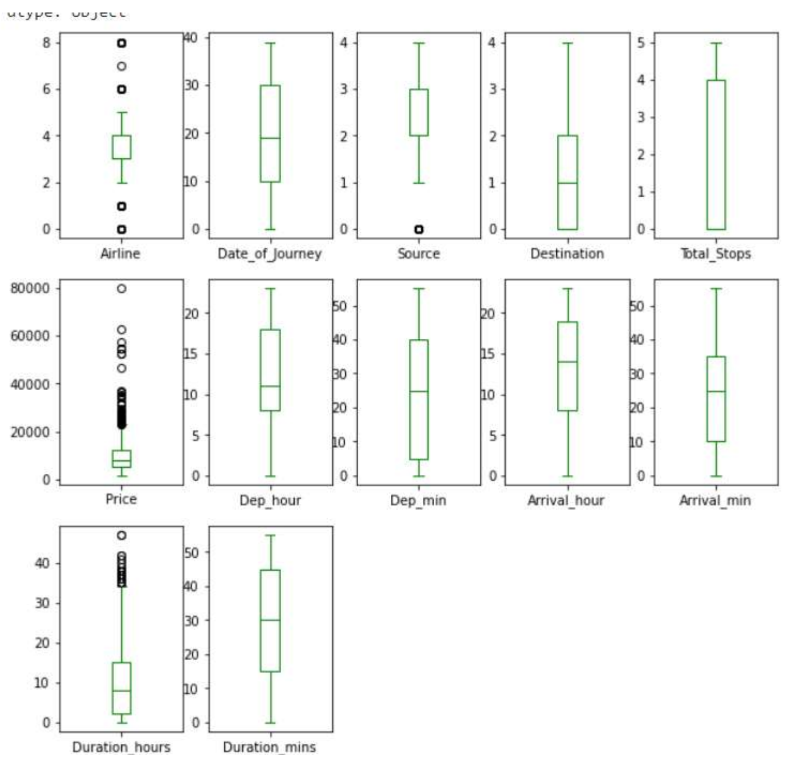
- The Price fares can vary based on various factors including the rush of the passengers to fly on a particular day, the frequency of airline operations, the route taken, number of stops, festivals/holidays occurring in the span when passengers decide to fly. Based on these criteria the fare price can vary on a single day.
- For removing the skewness, we generally use the log, cube root, squarer root or power transform method. But since the data was all object datatype, and no numerical columns were present except the target column we didn't remove the skewness.

- Hardware and Software Requirements and Tools Used
  - RAM : 8 GB
  - 512GB SSD
  - Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz
  - Python via Jupyter Notebook
  - Libraries/Packages specifically being used:
    - Pandas, NumPy, matplotlib, seaborn, scikit-learn

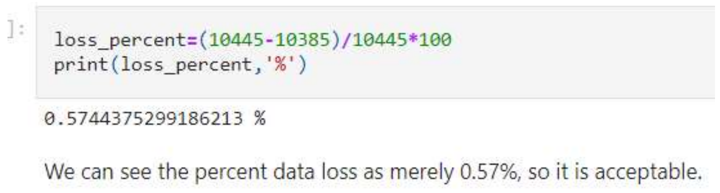
Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  Outliers were checked in the dataset, where if the Z score was above 3, then outliers were present. These were removed by the Z score method.



After removing the Outliers, a 0.6% data loss incurred, which was acceptable for the ML.

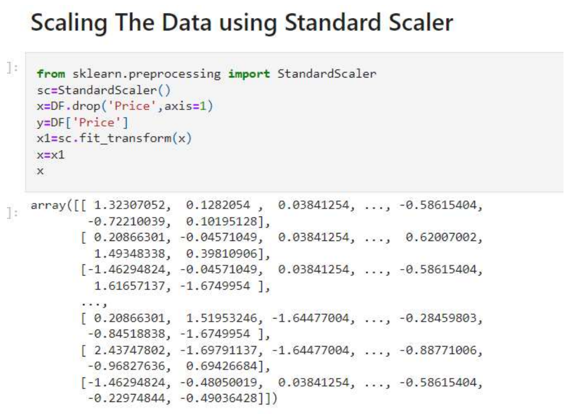


```
loss_percent=(10445-10385)/10445*100
print(loss_percent,'%')
```

```
0.5744375299186213 %
```

We can see the percent data loss as merely 0.57%, so it is acceptable.

Dataset was standardized by using the Standard scaler method and made ready for Machine Learning.



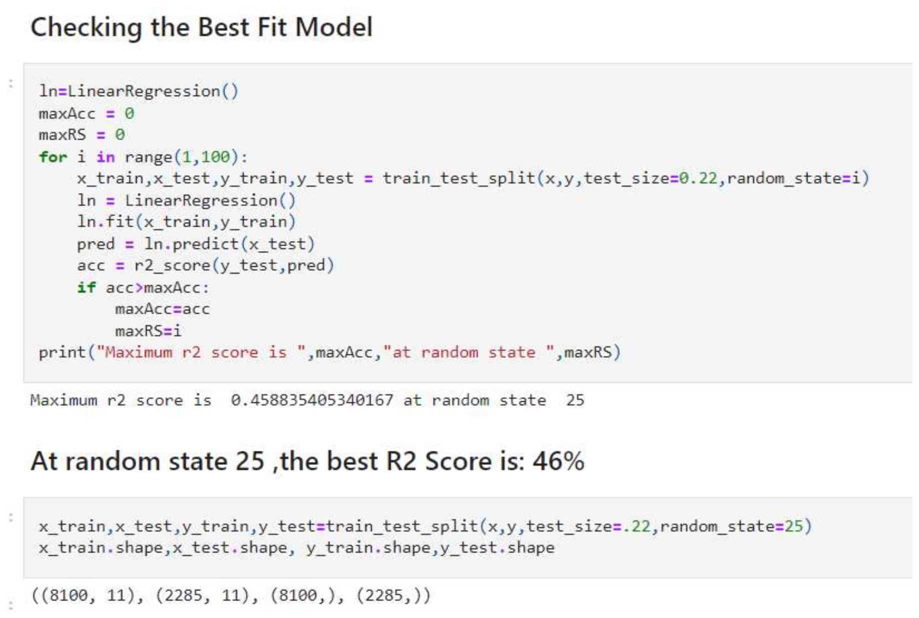
Scaling The Data using Standard Scaler

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=DF.drop('Price',axis=1)
y=DF['Price']
x1=sc.fit_transform(x)
x=x1
x
```

```
array([[ 1.32307052,  0.1282054 ,  0.03841254, ..., -0.58615404,
        -0.72210039,  0.10195128],
       [ 0.20866301, -0.04571049,  0.03841254, ...,  0.62007002,
         1.49348338,  0.39810906],
       [-1.46294824, -0.04571049,  0.03841254, ..., -0.58615404,
         1.61657137, -1.6749954 ],
       ...,
       [ 0.20866301,  1.51953246, -1.64477004, ..., -0.28459803,
        -0.84518838, -1.6749954 ],
       [ 2.43747802, -1.69791137, -1.64477004, ..., -0.88771006,
        -0.96827636,  0.69426684],
       [-1.46294824, -0.48050019,  0.03841254, ..., -0.58615404,
        -0.22974844, -0.49036428]])
```

Since the target column is not binary, we will use the regression algorithms for Machine Learning.

- Testing of Identified Approaches (Algorithms)
  - The following regression machine learning algorithms/ensemble methods used are:
    - Linear Regression Model
    - Ridge Regularization Model
    - Lasso Regularization Model
    - Support Vector Regression Model
    - Decision Tree Regression Model
    - Random Forest Regression Model
    - K Neighbours Regression Model
    - Gradient Boosting Regression Model
    - Ada Boost Regression Model
    - Extra Trees Regression Model
  - We split the data for Machine learning as x_train, x_test, y_train, y_test where we get the best R2 score for Linear Regression.



Checking the Best Fit Model

```
ln=LinearRegression()
maxAcc = 0
maxRS = 0
for i in range(1,100):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.22,random_state=i)
    ln = LinearRegression()
    ln.fit(x_train,y_train)
    pred = ln.predict(x_test)
    acc = r2_score(y_test,pred)
    if acc>maxAcc:
        maxAcc=acc
        maxRS=i
print("Maximum r2 score is ",maxAcc,"at random state ",maxRS)
```

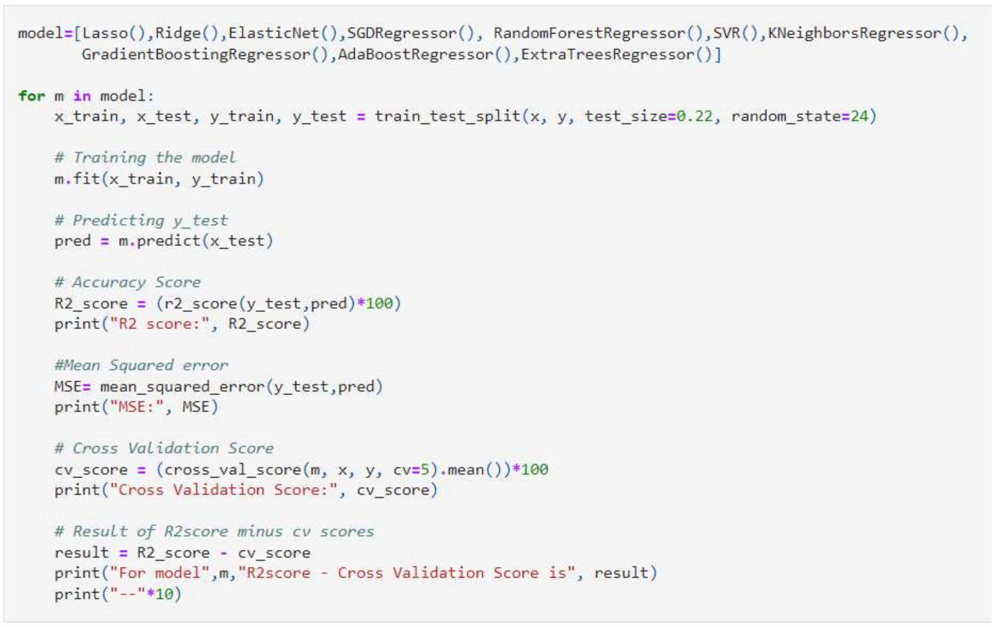
```
Maximum r2 score is  0.458835405340167 at random state  25
```

At random state 25 ,the best R2 Score is: 46%

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.22,random_state=25)
x_train.shape,x_test.shape, y_train.shape,y_test.shape
```

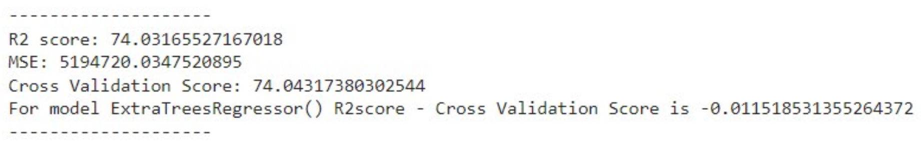
```
((8100, 11), (2285, 11), (8100,), (2285,))
```

  - We perform regularization and ensemble methods to get the best R2 score for the given split, and take forward that model for Hyper Tuning after checking the Cross Validation score.



```
model=[Lasso(),Ridge(),ElasticNet(),SGDRegressor(), RandomForestRegressor(),SVR(),KNeighborsRegressor(),
       GradientBoostingRegressor(),AdaBoostRegressor(),ExtraTreesRegressor()]

for m in model:
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.22, random_state=24)

    # Training the model
    m.fit(x_train, y_train)

    # Predicting y_test
    pred = m.predict(x_test)

    # Accuracy Score
    R2_score = (r2_score(y_test,pred)*100)
    print("R2 score:", R2_score)

    #Mean Squared error
    MSE= mean_squared_error(y_test,pred)
    print("MSE:", MSE)

    # Cross Validation Score
    cv_score = (cross_val_score(m, x, y, cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    # Result of R2score minus cv scores
    result = R2_score - cv_score
    print("For model",m,"R2score - Cross Validation Score is", result)
    print("--"*10)
```

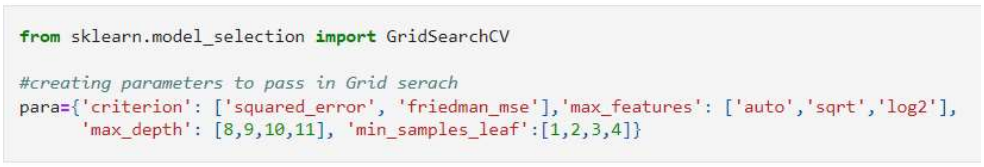
- The model whose |R2Score-CV score| is least we consider that model for the hyper tuning process. Here we are considering the Extra Trees Regressor method for Hyper tuning and getting the final result for prediction.



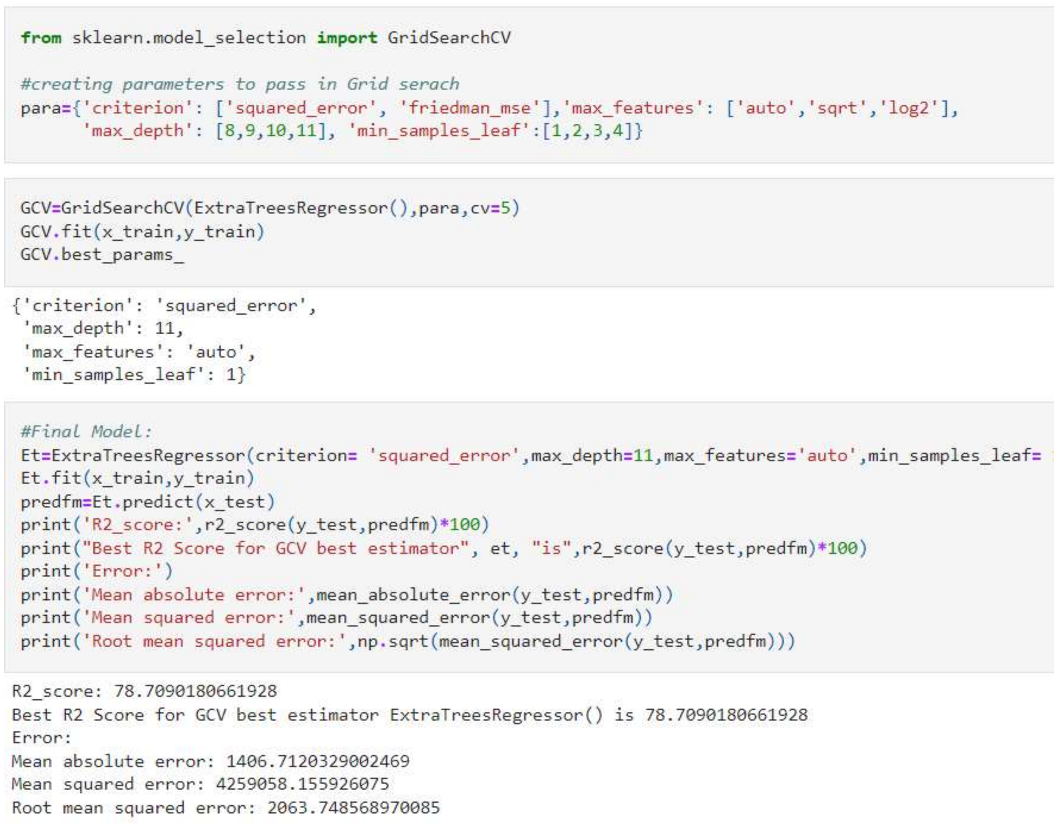
```
--------------------
R2 score: 74.03165527167018
MSE: 5194720.0347520895
Cross Validation Score: 74.04317380302544
For model ExtraTreesRegressor() R2score - Cross Validation Score is -0.011518531355264372
--------------------
```

- Run and evaluate selected models

  We selected the Extra Trees Regressor algorithm based on the R2-CV score compared to other algorithms, and hyper tuned it in the following manner. The best parameters gave us the actual score for the Model prediction.



```
from sklearn.model_selection import GridSearchCV

#creating parameters to pass in Grid serach
para={'criterion': ['squared_error', 'friedman_mse'],'max_features': ['auto','sqrt','log2'],
      'max_depth': [8,9,10,11], 'min_samples_leaf':[1,2,3,4]}
```



```
from sklearn.model_selection import GridSearchCV

#creating parameters to pass in Grid serach
para={'criterion': ['squared_error', 'friedman_mse'],'max_features': ['auto','sqrt','log2'],
      'max_depth': [8,9,10,11], 'min_samples_leaf':[1,2,3,4]}
```

```
GCV=GridSearchCV(ExtraTreesRegressor(),para,cv=5)
GCV.fit(x_train,y_train)
GCV.best_params_
```

```
{'criterion': 'squared_error',
 'max_depth': 11,
 'max_features': 'auto',
 'min_samples_leaf': 1}
```

```
#Final Model:
Et=ExtraTreesRegressor(criterion= 'squared_error',max_depth=11,max_features='auto',min_samples_leaf=
Et.fit(x_train,y_train)
predfm=Et.predict(x_test)
print('R2_score:',r2_score(y_test,predfm)*100)
print("Best R2 Score for GCV best estimator", et, "is",r2_score(y_test,predfm)*100)
print('Error:')
print('Mean absolute error:',mean_absolute_error(y_test,predfm))
print('Mean squared error:',mean_squared_error(y_test,predfm))
print('Root mean squared error:',np.sqrt(mean_squared_error(y_test,predfm)))
```

```
R2_score: 78.7090180661928
Best R2 Score for GCV best estimator ExtraTreesRegressor() is 78.7090180661928
Error:
Mean absolute error: 1406.7120329002469
Mean squared error: 4259058.155926075
Root mean squared error: 2063.748568970085
```

Here we found out that the R2 score increased from 74% to 79% after hyper tuning the parameters for Extra Trees Regression, which means that the model which we build is 79% efficient in predicting the fare price of the airlines.

- Key Metrics for success in solving problem under consideration
  - **RMSE Score:** Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

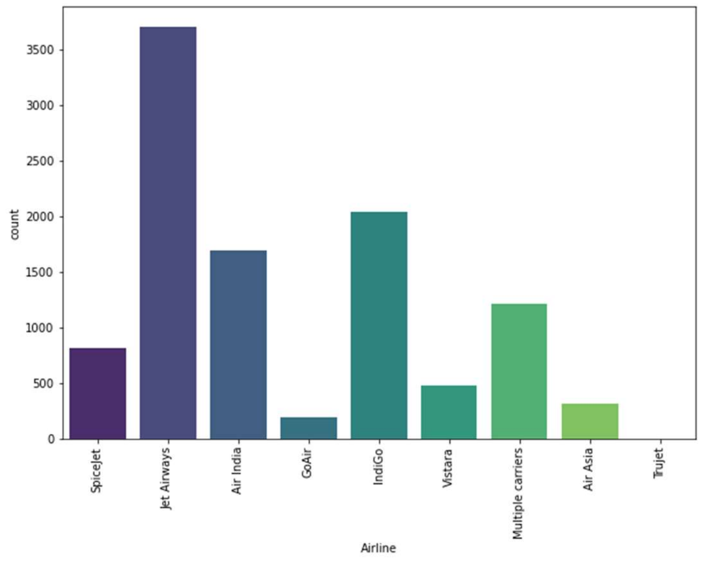
  - **R2 Score:** The R2 score is a very important metric that is used to evaluate the performance of a regression-based machine learning model. It is pronounced as R squared and is also known as the coefficient of determination. It works by measuring the amount of variance in the predictions explained by the dataset.

- **Cross Validation Score:** Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods. The k-fold cross validation is a procedure used to estimate the skill of the model on new data. There are common tactics that you can use to select the value of k for your dataset (I have used 5-fold validation in this project). There are commonly used variations on cross-validation such as stratified and repeated that are available in scikit-learn.

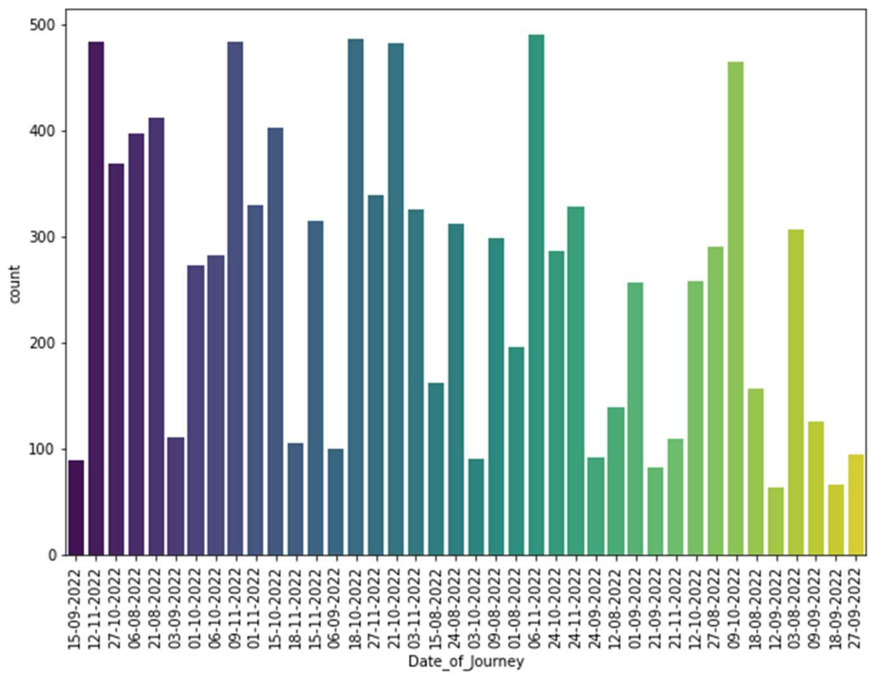
- **Hyper Parameter Tuning:** In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

- Visualizations

  Univariate, Bivariate and multivariate visualizations were done for the dataset:

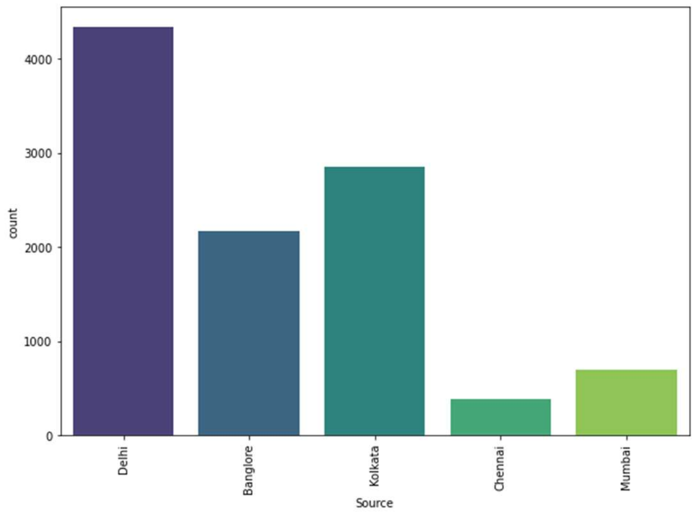


- 9 airlines/carriers have been mentioned in the dataset. - Indigo, Air India, JetAirways, SpiceJet, GoAir, Vistara, AirAsia, Trujet, Multiple carriers.
- Jet Airways had the highest number of flights (count=3750 approx) operating, followed by Indigo (2000 counts) in the span of 3 months.
- Least number of flights which flew in these 3 months were of Trujet.



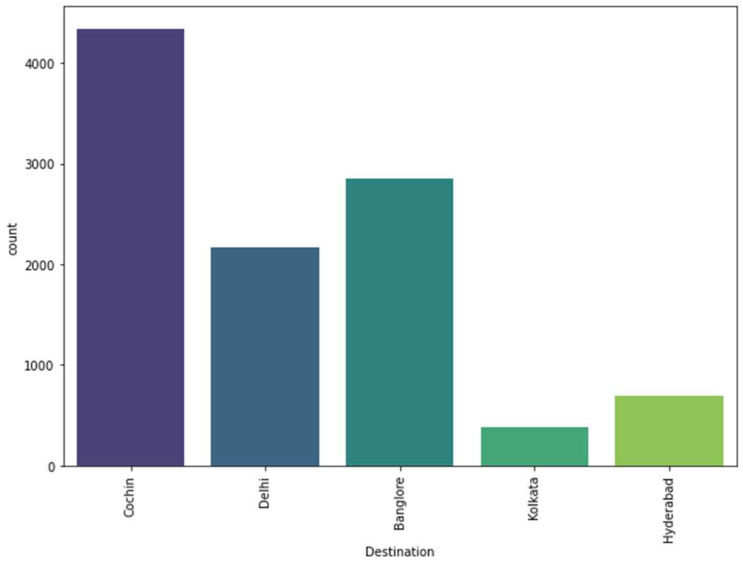
Date of Journey:

- 12th Nov, 9th Nov, 18th Oct, 21st Oct, 6th Nov and 9th Oct have highest count of passengers. While least are on 12th Sept. In general September month shows less travels undertaken.



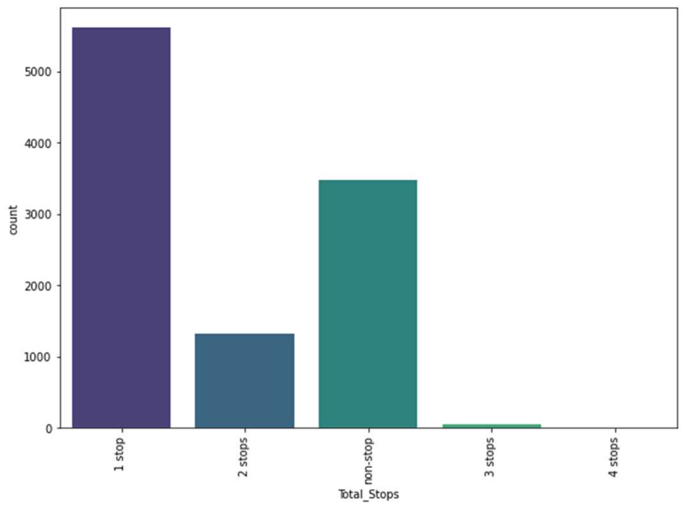
Source:

- The source cities are Bangalore, Kolkata, Delhi, Chennai, Mumbai.
- The origin of most flights was Delhi airport (count=4500), followed by Kolkata (around 3000). Least flights flew from Chennai (count=Approx. 500)



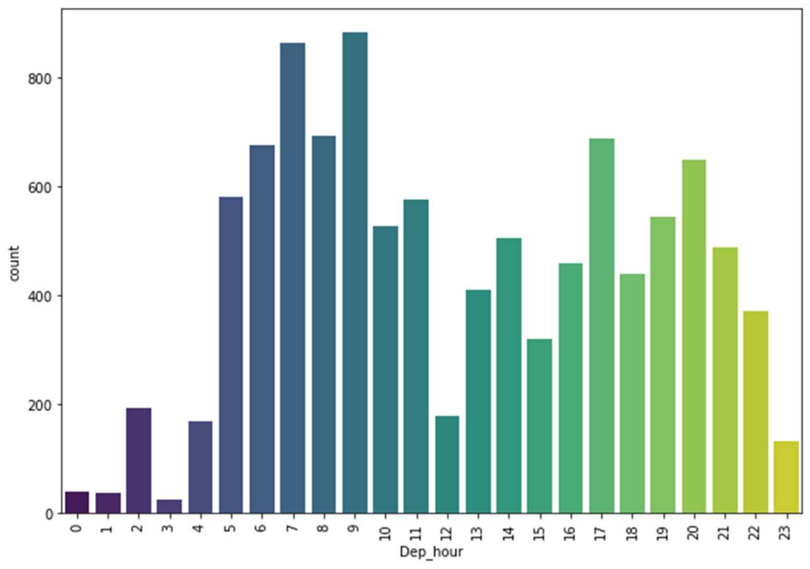
Destination:

- The destination cities are Delhi, Bangalore, Cochin, Kolkata, Hyderabad.
- The Destination was Kochi where most flights landed at the end of the day (count=4500), followed by Bangalore (almost 3000).
- The least flights which landed were reported in Kolkata.



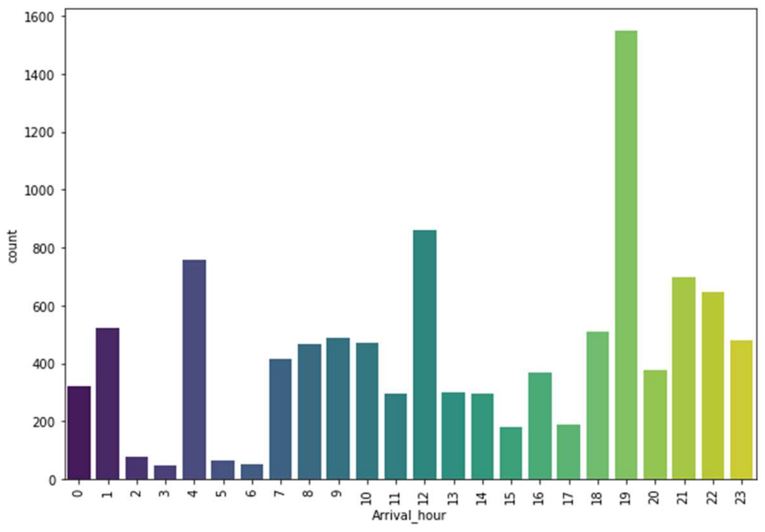
Total Stops:

- Most flights (almost 6000) made 1 stop on their route, while (almost 3500) flew nonstop. About 1500 made 2 stops while very few made 3 stops on their route. Handful or least flights took a 4 stop stop-over.



Departure hour:

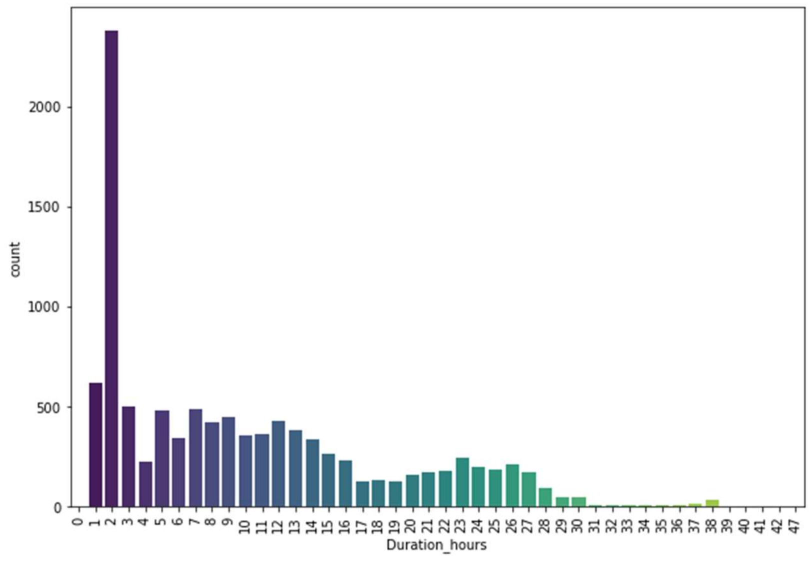
- Maximum flights flew at 9Am followed by 7Am. Least flew at 3 AM.



Arrival Hour:

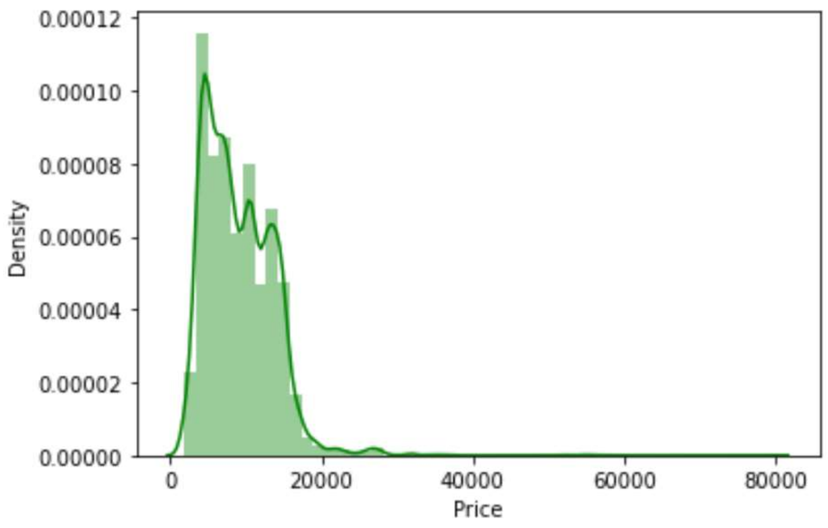
- Maximum flights arrived at 7pm, while least arrived at 3am.

Duration:



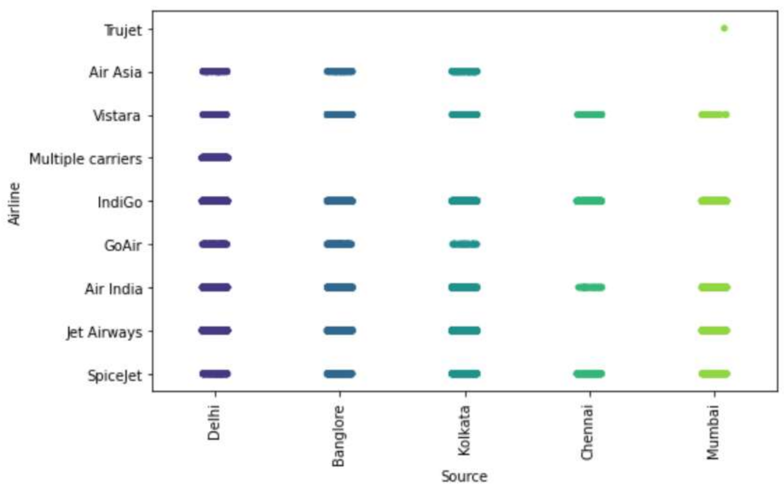
- Most flights had a duration of 2 hrs fly time.

```
sns.distplot(df['Price'],color='green')
plt.ticklabel_format(style='plain')
plt.show()
```



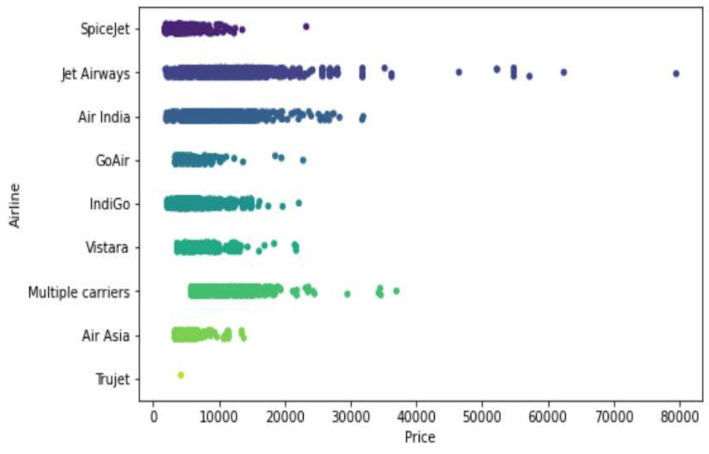
- The price of flights ranged between 2.5K to 80K.
- Maximum frequency of the price was between 5k-10K.
- The graph shows right skewness (positive skewness)

```
plt.figure(figsize=(8,5))
sns.stripplot(x="Source",y="Airline",data=df,palette='viridis')
plt.xticks(rotation=90)
plt.show()
```



- Delhi had maximum number of flights flying towards their destination, while Vistara, Air India, Indigo, and Spicejet operated from all the 5 cities.
- Jet airways didn't fly from Chennai. Go Air didn't fly from Chennai and Mumbai as a source. While TruJet only flew from Mumbai.

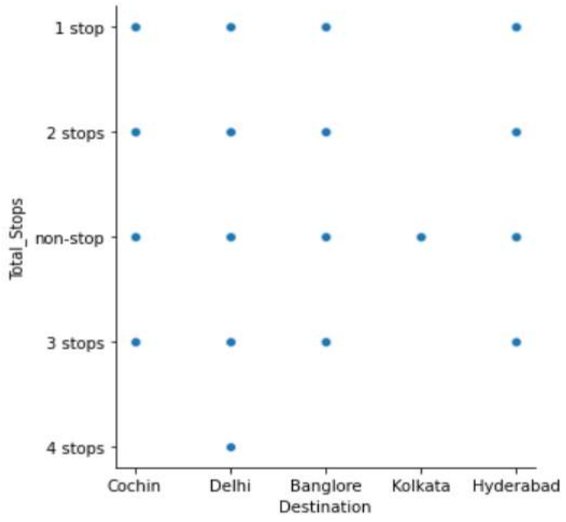
```
plt.figure(figsize=(8,5))
sns.stripplot(x="Price",y="Airline",data=df,palette='viridis')
plt.show()
```



- Price for Jetairways was highest of all the carriers ranging maximum mostly till 35000, while some fares rached around 80000 too.
- Least price were for TruJet which was 5000 maximum.

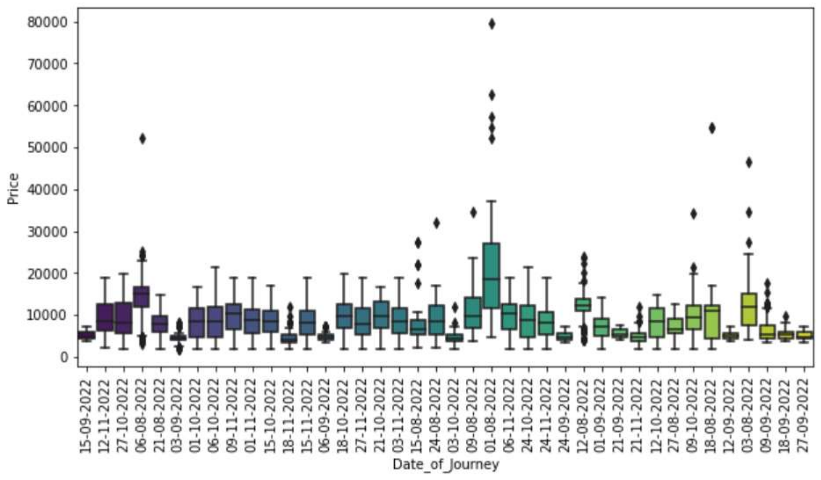
```
plt.figure(figsize=(8,5))
sns.relplot(x="Destination",y="Total_Stops",data=df,palette='viridis')
plt.show()
```

```
<Figure size 576x360 with 0 Axes>
```



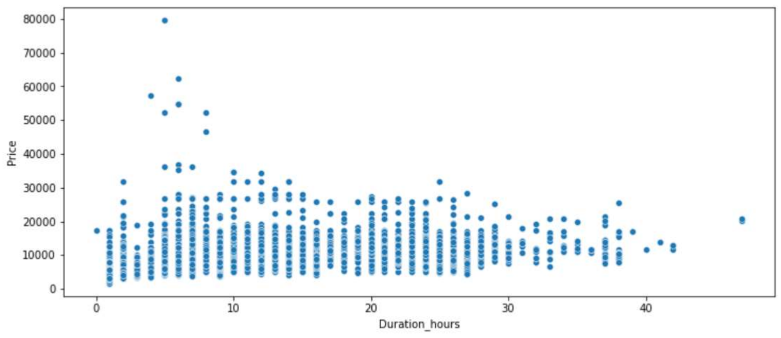
- Kolkata had flight which was nonstop only. Rest all places had flights with 1/2/3 stops.
- Delhi was the only city where to reach 4 stops were taken.

```
plt.figure(figsize=(10,5))
sns.boxplot(x='Date_of_Journey',y='Price',data=df,palette='viridis')
plt.xticks(rotation=90)
plt.show()
```



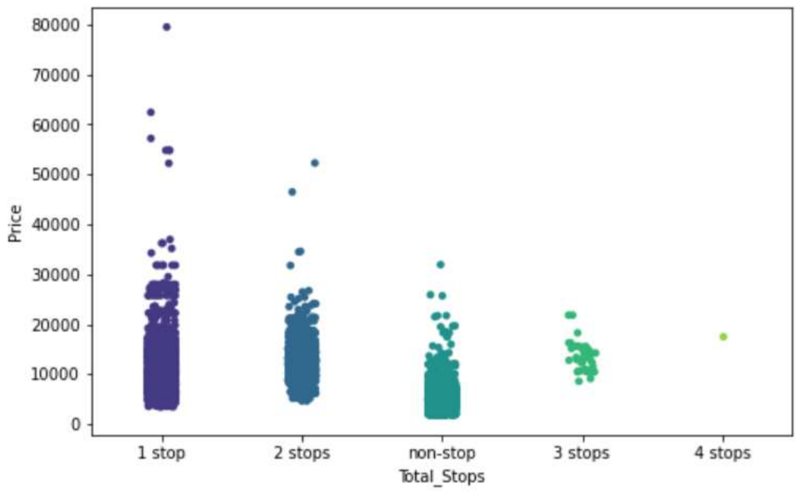
- Most tickets price are for 1st Oct which seems to be around 40000, followed by 3rd Oct.
- September shows the least fare price.

```
plt.figure(figsize=(12,5))
sns.scatterplot(x='Duration_hours',y='Price',data=df,palette='viridis')
plt.show()
```



- More the hours, lesser were the price of the ticket.
- While maximum price ranged between 1-8 hrs of flight duration, between 10-28hrs, the flight price remained in a single range between 20000-30000.

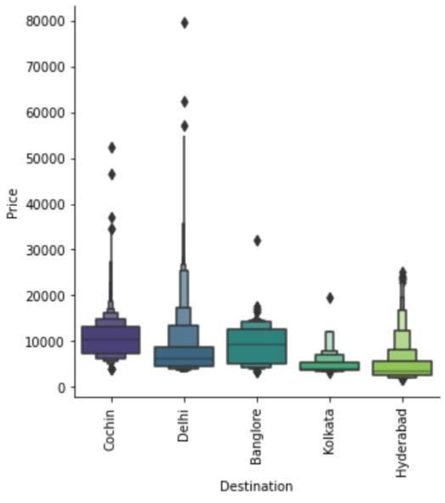
```
plt.figure(figsize=(8,5))
sns.stripplot(x='Total_Stops',y='Price',data=df,palette='viridis')
plt.show()
```



- More the stop, lesser was the price.
- With 1 stop, the maximum price was at 80000, but the maximum density in price was till 30000.

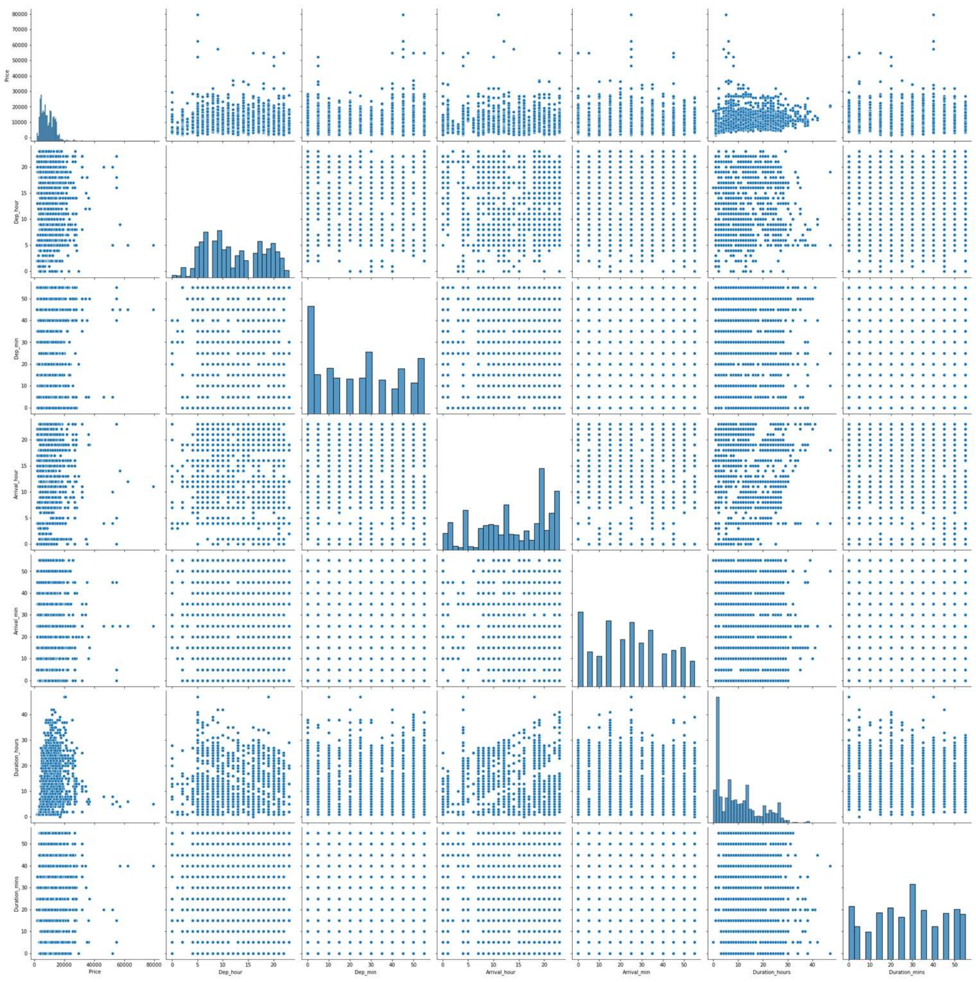
```
plt.figure(figsize=(8,5))
sns.catplot(x='Destination',y='Price',data=df,palette='viridis',kind='boxen')
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 576x360 with 0 Axes>
```

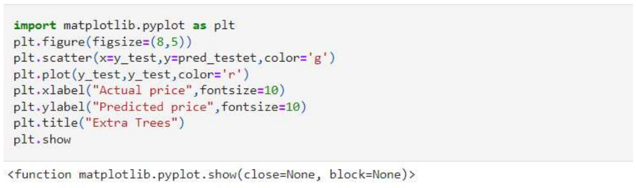


Flights to Delhi as final destination had highest price of 80000, while flights to Kolkata had lowest price of about 25000.

Multivariate Analysis:

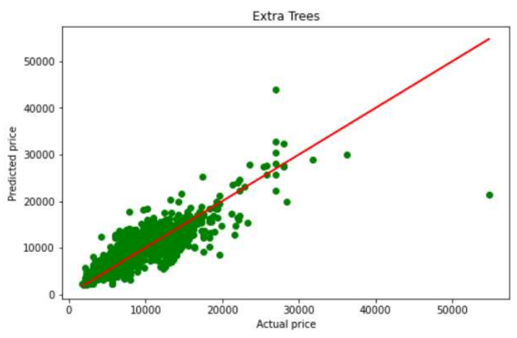


- Interpretation of the Results
  - From the above EDA we can easily understand the relationship between features and we can even see which things are affecting the price of flights.
  - Extra Trees Regressor, improved the predictive accuracy and controlled the over-fitting. It performed well despite having to work on small dataset and produced good predictions that can be understood easily, as the accuracy increased from 74% to 79%.
  - Following can be seen with the help of the best fit line:



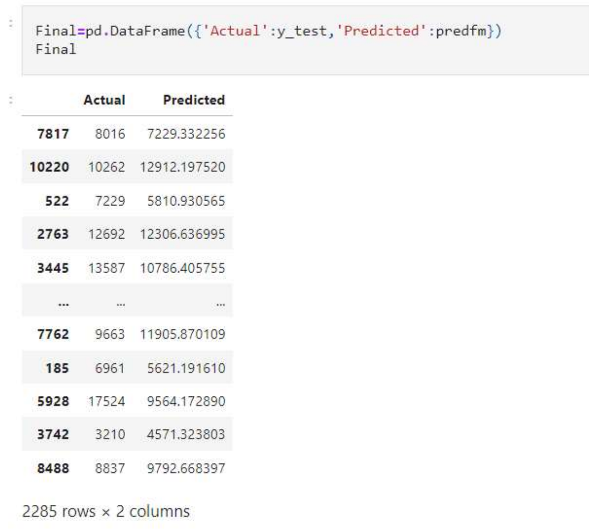
```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5))
plt.scatter(x=y_test,y=pred_testet,color='g')
plt.plot(y_test,y_test,color='r')
plt.xlabel("Actual price",fontsize=10)
plt.ylabel("Predicted price",fontsize=10)
plt.title("Extra Trees")
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



The best fit line is covering all of the point == model is trained and is a good fit.

The predicted flight Prices are as follows after saving the final model in obj format and loading the final model for prediction:



```
Final=pd.DataFrame({'Actual':y_test,'Predicted':predfm})
Final
```

	Actual	Predicted
7817	8016	7229.332256
10220	10262	12912.197520
522	7229	5810.930565
2763	12692	12306.636995
3445	13587	10786.405755
...	...	...
7762	9663	11905.870109
185	6961	5621.191610
5928	17524	9564.172890
3742	3210	4571.323803
8488	8837	9792.668397

2285 rows × 2 columns

- With these predictions we can know the fare of the flights efficiently.

CONCLUSION

- Key Findings and Conclusions of the Study
  - Based on the in-depth analysis of the Flight Price Prediction Project, the Exploratory analysis of the datasets, and the analysis of the outputs of the models the following observations are made:
  - Air Fare attributes like Date, Month, Duration, Total Stops etc play a big role in influencing the used Flight price.
  - Airline Brand also has a very important role in determining the Flight fare.
  - Various plots like Bar plots, Count plots and Line plots helped in visualising the feature-label relationships which corroborated the importance of Air Fare features and attributes for estimating Flight ticket prices.
  - Due to the dataset being very small, only very small amount of the outliers was removed to ensure proper training of the models.
  - Extra Trees Regressor, improved the predictive accuracy and controlled the over-fitting. It performed well despite having to work on small dataset and produced good predictions that can be understood easily, as the accuracy increased from 74% to 79%.

- Learning Outcomes of the Study in respect of Data Science
  - Data cleaning was a very important step in removing plenty of anomalous data from the huge dataset that was provided.
  - Visualising data helped identify outliers and the relationships between target and feature columns as well as analysing the strength of correlation that exists between them.

- Limitations of this work and Scope for Future Work
  - A small dataset to work with posed a challenge in building highly accurate models.

- This project also relied heavily data which was unable to account for various other factors that influence demand and ticket pricing like pandemic status affecting demand, Air travel guidelines change, changes in routes, weather conditions, etc.
- Most airline companies also do no publicly make available their ticket pricing strategies, which makes gathering price and air fare related datasets using web scraping the only means to build a dataset for building predicting models. Availability of more features and a larger dataset would help build better models.