FLIGHT PRICE PREDICTION

1. Introduction

In the era of globalization passenger airlines have seen tremendous growth in traffic all over the world. In India passenger traffic amounted to over 115 million at airports across India in financial year 2021, out of which over 10 million were international passengers. So much traffic would also generate tremendous amount of data which can be used to get insights about a lot of things. This is where data science comes into picture.

2. Problem Definition

In this project we will develop a machine learning model to predict the price of Indian domestic flight tickets. The dataset contains details of flight tickets for various airlines between the months of March and June of 2019. The dataset contains 10683 entries.

The features can be summarized as follows:

- Airline: The name of the airline.
- Date_of_Journey: The date of the journey.
- Source: The flight departure city.
- Destination: The flight arrival city.
- Route: The route taken by the flight to reach the destination.
- 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.
- Additional_Info: Additional information about the flight
- Price: The price of the ticket.(Label)

3. Data Analysis

First let's get the basic information about the columns and their data-type.

```
df.info()
```

```
<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
```

The dataset has all object data-type features and the label i.e. 'Price' is in integer format.

3.1 Check for missing values

```
df.isnull().sum()
```

```
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
```

Column 'Route' and 'Total_Stops' has one missing values each.

Let’s impute these with their respective mode.

```
# impute the missing values with mode.
df['Route']=df['Route'].fillna(df['Route'].mode()[0])
df['Total_Stops']=df['Total_Stops'].fillna(df['Total_Stops'].mode()[0])
```

3.2 Check for duplicate entries

```
# check for duplicates.
df.duplicated().value_counts()
```

```
False    10463
True       220
dtype: int64
```

We found 220 duplicate entries. Let’s drop them.

```
# dropping the duplicate entries.
df.drop_duplicates(inplace=True,ignore_index=True)
```

3.3 Drop Arrival_Time column

If we see the columns carefully we can see that there are already columns for departure time and duration. Arrival time is not required as it can be deduced from those two. Hence, dropping it.

3.4 Feature engineering with the Date column

The ‘Date_of_Journey’ feature is in object data format but machine learning models don’t understand text data. We have to convert it to numeric format. Fortunately pandas library have an amazing function called *to_datetime* which converts the data to pandas date-time format from which we can easily extract the corresponding day, month and year. Since

our dataset has only 2019 year data hence we will only extract the day and month into separate columns. Below is the code:

```
# Convert the 'Date_of_Journey' column to datetime format and extract the Day and Month into separate columns.
df['Date_of_Journey']=pd.to_datetime(df['Date_of_Journey'],format='%d/%m/%Y',dayfirst=True)
df['Day_of_month']= df['Date_of_Journey'].dt.day
df['Month']= df['Date_of_Journey'].dt.month
df.drop(columns=['Date_of_Journey'],inplace=True)
```

The original date column is deleted after extraction of relevant data.

3.5 Feature engineering with the Departure time column

Dep_Time column has data in text format. Here also we use the pandas *to_datetime* method to convert the data to hour and minutes. We will extract the hour and minute, then combine them to make float number which represents the original data. Then we replace the original column with the new column. Below is the code:

```
# Convert the 'Dep_Time' column to datetime format and convert it to 24 hours format of float-datatype
df['Dep_Time']=pd.to_datetime(df['Dep_Time'],format='%H:%M')
x=df['Dep_Time'].dt.hour+df['Dep_Time'].dt.minute/60
df['Dep_Time']=x
df.head()
```

	Airline	Source	Destination	Route	Dep_Time	Duration	Total_Stops	Additional_Info	Price	Day_of_month	Month
0	IndiGo	Banglore	New Delhi	BLR → DEL	22.333333	2h 50m	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	5.833333	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	9.416667	19h	2 stops	No info	13882	9	6
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18.083333	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16.833333	4h 45m	1 stop	No info	13302	1	3

3.6 Feature engineering with the Duration column

The data in ‘Duration’ column is in text format. If the duration is like 5 hour 30 minutes it is denoted by 5h 30m. What we want to do is that we remove the ‘h’ and ‘m’ and then divide the minutes by 60 and put it after the hour in decimals. For example 5h 30m will convert to 5.5. Below is the code:

```
# The h and m in duration data is removed and only numerics are extracted to a list in string format.
duratn=[]
for i in range(len(df['Duration'])):
    components= df['Duration'][i].strip().split(" ")
    if len(components)==2:
        hour= components[0].split("h")[0]
        minute=components[1].split("m")[0]
        duratn.append(hour+'.'+minute)
    elif (len(components)==1) & (components[0].find("h")!=(-1)):
        hour= components[0].split("h")[0]
        minute=str(0)
        duratn.append(hour+'.'+minute)
    elif (len(components)==1) & (components[0].find("m")!=(-1)):
        hour=str(0)
        minute=components[0].split("m")[0]
        duratn.append(hour+'.'+minute)
    else:
        duratn.append(df['Duration'][i])

# duration data is converted to numeric format.
df['Duration']= pd.to_numeric(duratn)
```

3.7 Encoding the Total Stops column

'Total_Stops' denotes the number of stops the flight made during the journey. The data is in text format. It holds five unique values: 'non-stop', '2 stops', '1 stop', '3 stops', '4 stops'. Let's encode them in order with the help of replace function.

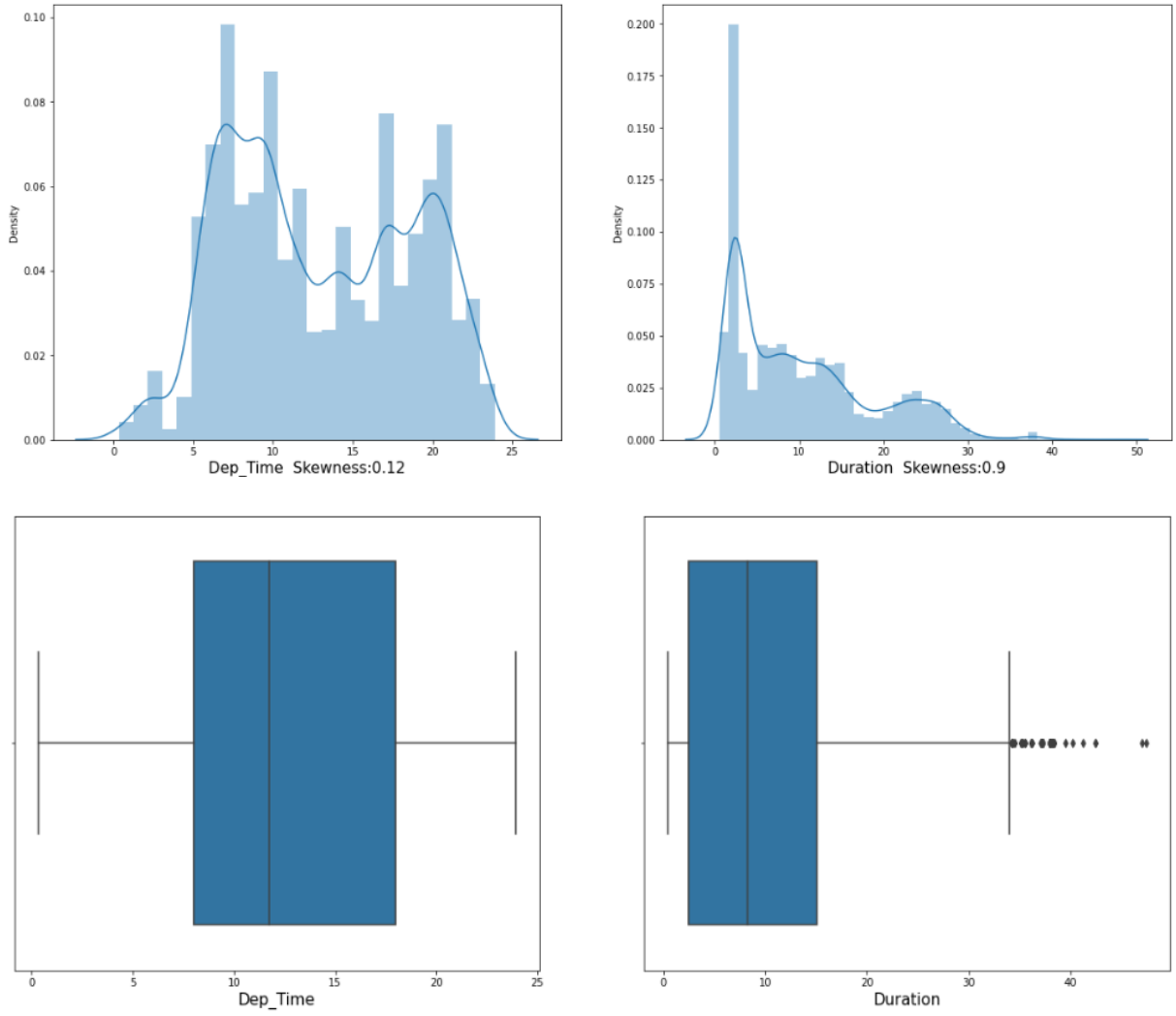
```
# encoding the data using replace function.
df['Total_Stops'].replace({'non-stop':0,
                           '1 stop':1,
                           '2 stops':2,
                           '3 stops':3,
                           '4 stops':4},inplace=True)
df['Total_Stops'].unique()
```

```
array([0, 2, 1, 3, 4], dtype=int64)
```

Before encoding the rest of categorical columns let's see some visualization plots as it is easier to identify the text-data categories unencoded in the plots.

3.8 Visualizations

Distribution-plot and Box-plot of Dep_Time and Duration column:



'Dep_Time' column is fairly symmetric with minimal skewness and no outliers can be seen in the box-plot. However the 'Duration' column has some skewness and outliers in the data. Let's remove the outliers using z-score method.

Z-score:

The formula:

Z score = (x - mean) / std. deviation

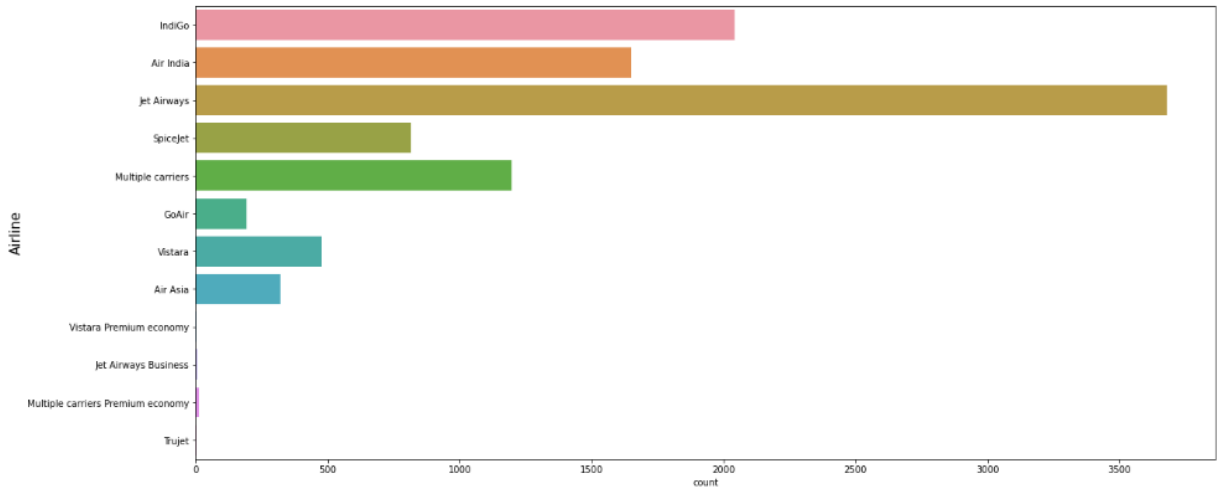
Z score tells how many standard deviations away a data point is from the mean. Data point outside of 3 standard deviations indicates that the data point is quite different from the other data points. Such a data point can be considered as an outlier.

Let's apply it on 'Duration' column:

```
# Removing outliers in 'Duration' column using z-score.
from scipy.stats import zscore
z_score= zscore(df[['Duration']])
abs_zscore= np.abs(z_score)
filtering_entry= (abs_zscore<3).all(axis=1)
df=df[filtering_entry]
df.shape
```

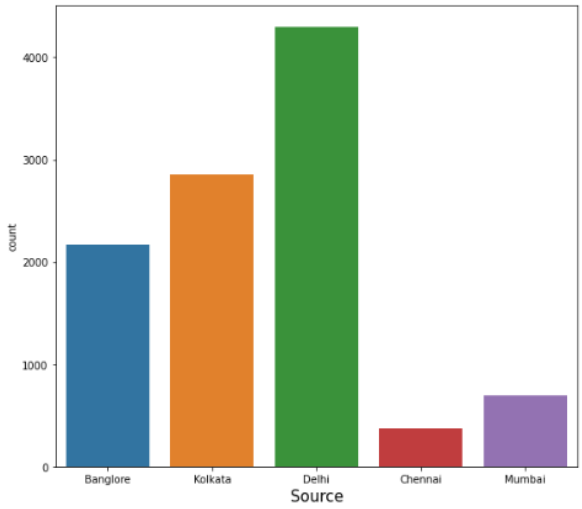
```
(10400, 11)
```

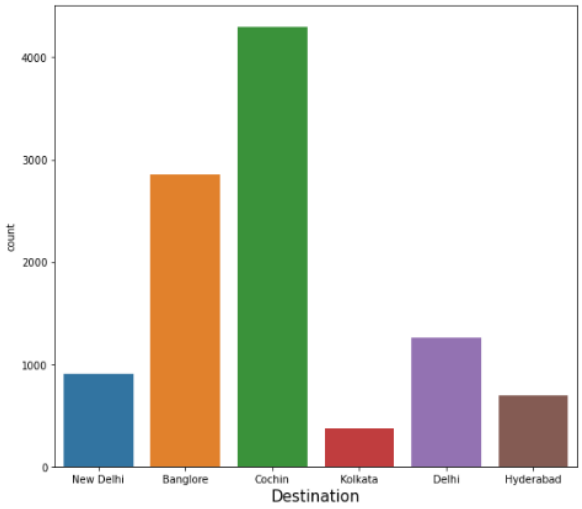
Count-plot of Airline column:



Majority of flights in the dataset are of Jet Airways.

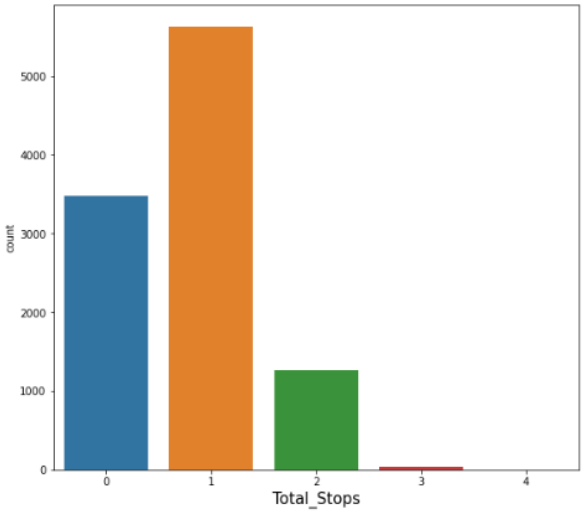
Count-plots of Source and Destination columns:





Delhi is the source city for majority of the flights. Cochin is the destination for majority of the flights.

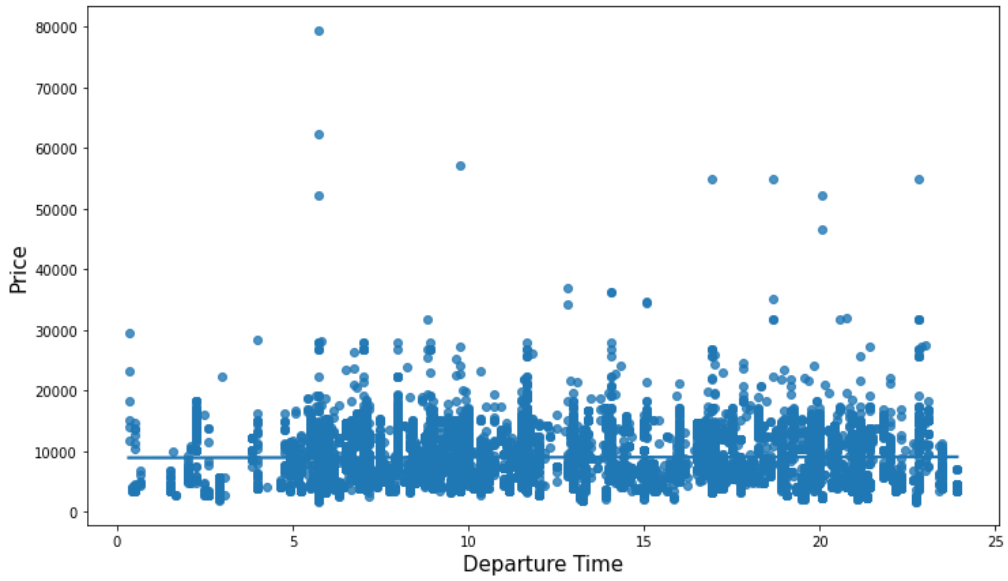
Count-plot of Total_Stops column:



Majority of the flights have 1 stop during the journey.

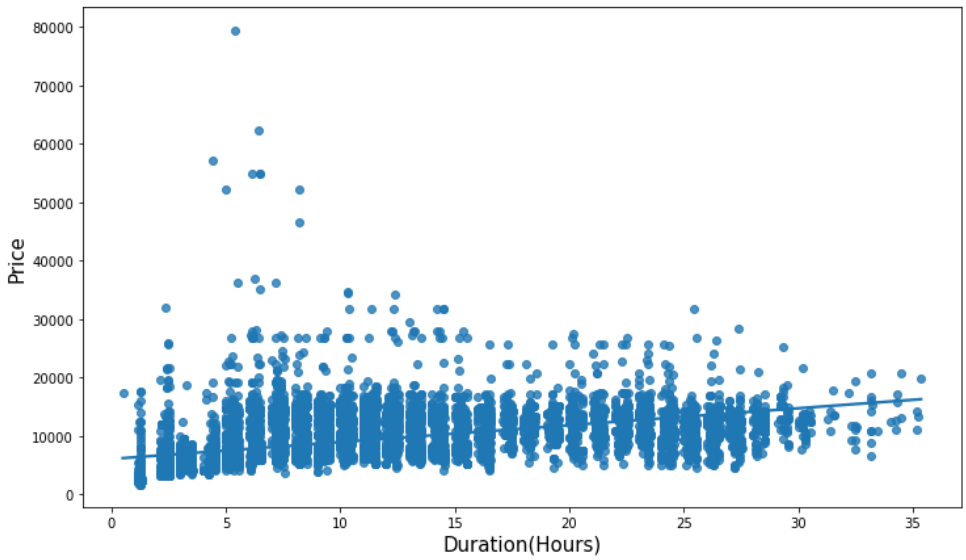
Now let’s analyze the relation between some of the features and the ticket price.

Regression-plot of Departure Time vs Price:



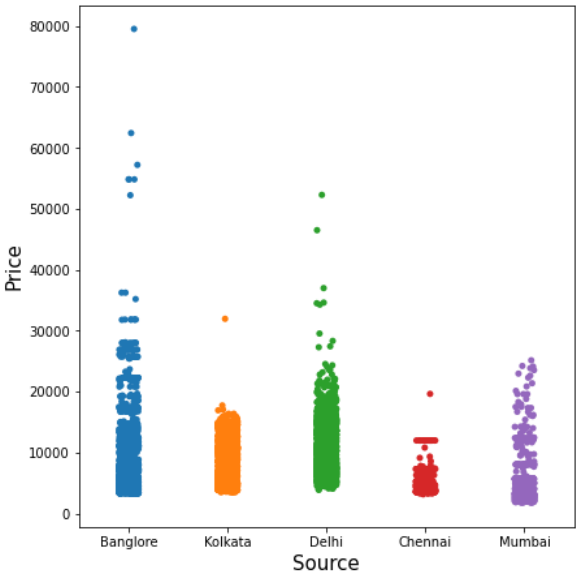
It can be seen that the regression fit line is almost horizontal. Hence we can say that the price of a flight ticket doesn’t depend on what time the flight departs.

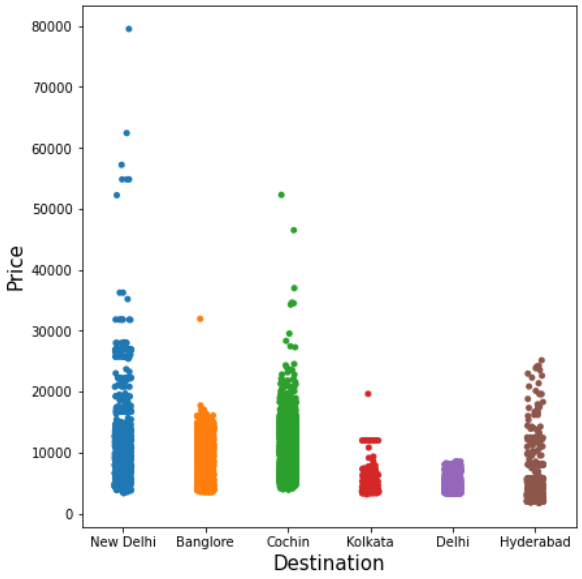
Regression-plot of Duration vs Price:

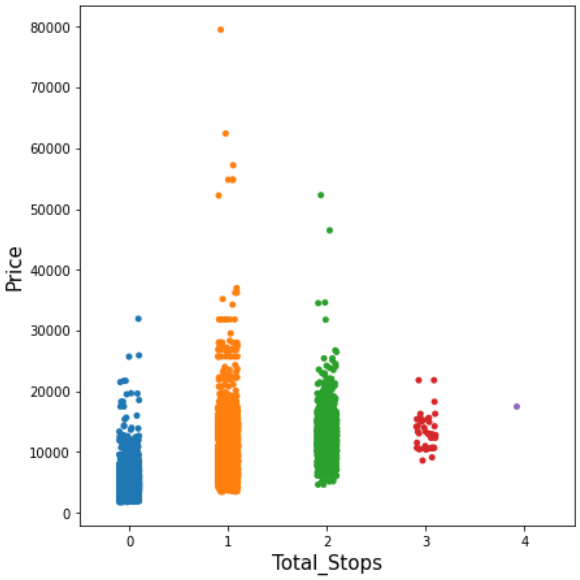


It can be seen that the regression fit line has a gentle positive slope. Hence we can say that there is a weak positive linear relation between flight duration and ticket price.

Strip-plots of Source, Destination and Total_Stops with respect to price:







3.9 Encoding using Label-Encoder

Label-Encoder is part of SciKit-learn library and is used to convert text or categorical data into numerical data. We still have 5 object data-type columns to encode. Let's do it.

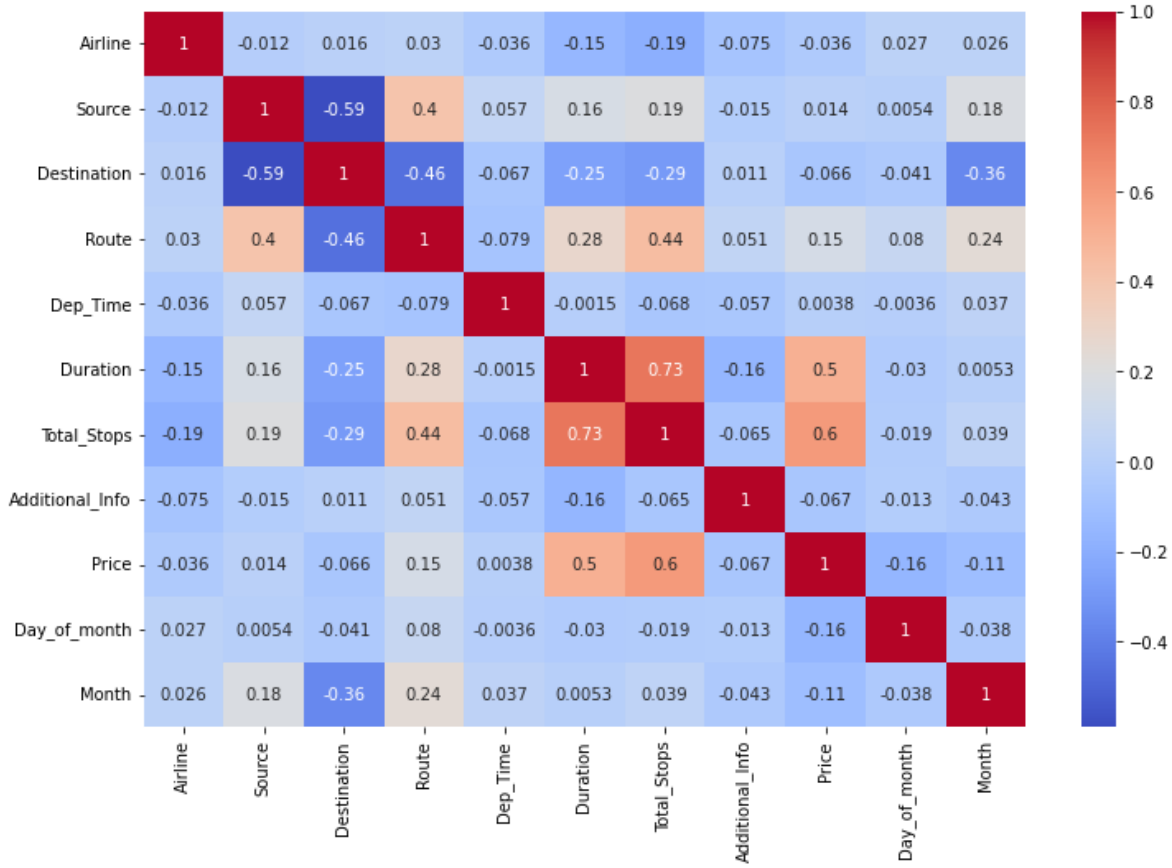
```
# Encoding the categorical columns with label encoder.
from sklearn.preprocessing import LabelEncoder
enc= LabelEncoder()

def label_encode(column):
    df[column]= enc.fit_transform(df[[column]])

df_to_encode=['Airline','Source','Destination','Route','Additional_Info']
for i in df_to_encode:
    label_encode(i)
df
```

	Airline	Source	Destination	Route	Dep_Time	Duration	Total_Stops	Additional_Info	Price	Day_of_month	Month
0	3	0	5	18	22.333333	2.50	0	7	3897	24	3
1	1	3	0	84	5.833333	7.25	2	7	7662	1	5
2	4	2	1	118	9.416667	19.00	2	7	13882	9	6
3	3	3	0	91	18.083333	5.25	1	7	6218	12	5
4	3	0	5	29	16.833333	4.45	1	7	13302	1	3
...	...	...	...	...	...	...	...	...	...	...	...
10458	0	3	0	64	19.916667	2.30	0	7	4107	9	4
10459	1	3	0	64	20.750000	2.35	0	7	4145	27	4
10460	4	0	2	18	8.333333	3.00	0	7	7229	27	4
10461	10	0	5	18	11.500000	2.40	0	7	12648	1	3
10462	1	2	1	108	10.916667	8.20	2	7	11753	9	5

All the columns are now in numeric format.

3.10 Correlation

Let's see the correlation heat-map:



'Total_Stops' and 'Duration' have the strongest positive correlation with 'Price'. The correlation coefficient of 'Price' vs 'Dep_Time' is 0.0036. It can be said that departure time has almost no relation with the ticket price. Hence we drop that column.

```
# drop the 'Dep_Time' column.
df.drop(columns=['Dep_Time'],inplace=True)
```

3.11 Feature Scaling

We have dropped the inconsequential features and finalized our list of features to be used in the model. Now let's separate the features and label in the dataset then scale the features.

Feature scaling is a method used to normalize or standardize the range of independent variables or features of data. It is done because scaled data is easier for machine learning models to interpret.

We will standardize the features using 'StandardScaler' object of scikit-learn. It uses the below formula:

$$y = (x - mean) / standard\_deviation$$

```
# Separating the features and label.
X= df.drop(columns=['Price'])
y= df['Price']
```

```
# applying standard scaler on the features and scaling them.
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
X_scaled= scaler.fit_transform(X)
```

4. EDA Concluding Remarks

- The dataset has all the features in object data-type format except the target variable i.e. 'Price'.
- The dataset have 2 missing values which are imputed with mode of the respective column.
- There are 220 duplicate entries which are dropped.
- 'Arrival_Time' column is dropped as we already have departure time and flight duration data.
- From the 'Date_of_Journey' column the day and month data is extracted to separate columns. 'Date_of_Journey' column is then dropped.
- The data in 'Dep_Time' and 'Duration' column are converted to get the data in proper format.

- ‘Total_Stops’ column is encoded using *replace* function. The remaining categorical columns are encoded using Label-Encoder.
- Majority of flights in the dataset are of Jet Airways.
- Delhi is the source city for majority of the flights.
- Cochin is the destination for majority of the flights.
- Most of the flights have 1 stop during the journey.
- ‘Total_Stops’ and ‘Duration’ have the strongest positive correlation with ‘Price’.
- In the correlation heat-map it is observed that ‘Dep_Time’ have almost no relation with ‘Price’. Hence dropped.

5. Pre-processing Pipeline

In this section, we list the data pre-processing steps to prepare datasets for machine learning algorithm implementation. Although this project is done without using pipeline and estimators but in the professional data science work it is recommended to use pipelines as it streamlines the entire preprocessing steps in a neat way. When we have to make predictions for new data then we don’t have to repeat all the preprocessing code steps manually and instead just feed the data into the pipeline and we get our data processed and ready for feeding to the model.

Steps in pre-processing pipeline:

- **Feature Engineering**: From the ‘Date_of_Journey’ column extract the day and month to separate columns.
- **Data conversion**: The data in ‘Duration’ column is to be converted to numeric format.
- **Column drop**: Drop the ‘Arrival_Time’, ‘Dep_Time’ and ‘Date_of_Journey’ columns.

- **Encoding**: Encode the ‘Total_Stops’ column using replace function. Encode 'Airline', 'Source', 'Destination', 'Route' and 'Additional_Info' columns using Label-Encoder.
- **Feature Scaling**: Scale the features using StandardScaler of sklearn.

6. Building Machine Learning Models

Let’s first use a range of baseline algorithms and evaluate the performance on different train-test splits. Then we do cross-validation of these models for 5-fold to 10-fold. After that we do hyperparameter tuning of the best performer.

We will consider the following algorithms:

- Linear Regression
- Adaboost Regression
- Random-Forests Regression
- XGBoost Regression
- K-Neighbors Regression.

We will use the coefficient of determination (R^2) to evaluate the models’ performance. The R^2 score for a model is a useful statistic in regression analysis, as it often describes how ‘good’ that model is at making predictions.

The values for R^2 range from 0 to 1. The closer the score is to 1 better is the model. A model can give a negative R^2 score as well, which indicates that the model is arbitrarily worse than the one that always predicts the mean of the target variable.

Code:

```
lr= LinearRegression()
ada = AdaBoostRegressor()
rf= RandomForestRegressor()
xgb_reg= xgb.XGBRegressor()
knr= KNeighborsRegressor()

models = [lr,ada,rf,xgb_reg,knr]

def get_score(model,train_x,test_x,train_y,test_y):
    model.fit(train_x,train_y)
    pred_y= model.predict(test_x)
    accuracy_test = round(r2_score(test_y,pred_y),2)
    return accuracy_test

scores=[[],[],[],[],[]]
for i in range(0,5):
    x_train,x_test,y_train,y_test = train_test_split(X_scaled,y,test_size=0.2,random_state=i)
    for model in models:
        score= get_score(model,x_train,x_test,y_train,y_test)
        scores[i].append(score)

results= pd.DataFrame(scores,columns=['Linear Regression','Adaboost Regression','Random-Forests Regression','XGBoost Regression'
                                      ,'K-Neighbors Regression'],
                              index=['Train-Test Split 1','Train-Test Split 2','Train-Test Split 3','Train-Test Split 4',
                                     'Train-Test Split 5'])
results
```

Results:

	Linear Regression	Adaboost Regression	Random-Forests Regression	XGBoost Regression	K-Neighbors Regression
Train-Test Split 1	0.45	0.56	0.85	0.84	0.80
Train-Test Split 2	0.47	0.47	0.90	0.91	0.82
Train-Test Split 3	0.47	0.47	0.84	0.84	0.79
Train-Test Split 4	0.44	0.50	0.81	0.81	0.77
Train-Test Split 5	0.49	0.50	0.84	0.85	0.81

After running the models on five different train-test splits we can see that Random-Forests and XGBoost are giving the best R^2 scores.

Let’s do cross-validation of different folds and check whether their performance is reliable or not.

Code:

```
cv_scores=[[],[],[],[],[],[]]
count=0
for i in range(5,11):
    for model in models:
        score= round(cross_val_score(model,X_scaled,y,cv=i).mean(),2)
        cv_scores[count].append(score)
    count+=1

cv_results= pd.DataFrame(cv_scores,columns=['Linear Regression','Adaboost Regression','Random-Forests Regression',
                                            'XGBoost Regression','K-Neighbors Regression'],
                                  index=['5-Fold Cross-Validation','6-Fold Cross-Validation','7-Fold Cross-Validation',
                                         '8-Fold Cross-Validation','9-Fold Cross-Validation','10-Fold Cross-Validation'])
cv_results
```

Cross-Validation Results:

	Linear Regression	Adaboost Regression	Random-Forests Regression	XGBoost Regression	K-Neighbors Regression
5-Fold Cross-Validation	0.46	0.50	0.85	0.88	0.80
6-Fold Cross-Validation	0.46	0.48	0.85	0.87	0.81
7-Fold Cross-Validation	0.46	0.48	0.84	0.88	0.80
8-Fold Cross-Validation	0.46	0.47	0.85	0.88	0.80
9-Fold Cross-Validation	0.46	0.50	0.85	0.88	0.81
10-Fold Cross-Validation	0.46	0.48	0.86	0.88	0.81

From the cross-validation results we can see that XGBoost is performing slightly better than Random-Forests. So, let's move ahead with the XGBoost model and try to tune is further.

Hyperparameter Tuning using GridSearchCV

We will try to tune 3 hyper-parameters of XGBoost algorithm using GridSearchCV. They are 'learning_rate', 'max_depth' and 'n_estimators'.

Code:

```
from sklearn.model_selection import GridSearchCV

params= {"n_estimators": [100,250,500,1000],
         "max_depth": [4,5,6,7,8],
         "learning_rate": [0.01,0.1,1]
         }

grd= GridSearchCV(xgb_reg, param_grid=params,cv=5,n_jobs=-1)
grd.fit(x_train,y_train)
print("Best Parameters:",grd.best_params_)
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500}
```

Now let's use these hyper-parameters:

```
# Performance of the tuned model.
xgb_tuned= xgb.XGBRegressor(learning_rate=0.1,max_depth=5, n_estimators=500)
x_train,x_test,y_train,y_test = train_test_split(X_scaled,y,test_size=0.2,random_state=1)

xgb_tuned.fit(x_train,y_train)
y_pred_train = xgb_tuned.predict(x_train)
accuracy_train = r2_score(y_train,y_pred_train)
y_pred= xgb_tuned.predict(x_test)
accuracy_test = r2_score(y_test,y_pred)

print("Training Score:",round(accuracy_train,2))
print("Testing Score: ",round(accuracy_test,2))

# cross-validating the tuned XGBoost regression model.
print("\n++++++++++++++++ CROSS VALIDATION +++++++++++++++++","\n")
for i in range(5,11):
    cv_score= cross_val_score(xgb_tuned,X_scaled,y,cv=i).mean()
    print("the cv score for",i,"fold:", round(cv_score,2))
```

```
Training Score: 0.94
Testing Score:  0.91

++++++++++++++ CROSS VALIDATION +++++++++++++++++

the cv score for 5 fold: 0.89
the cv score for 6 fold: 0.89
the cv score for 7 fold: 0.88
the cv score for 8 fold: 0.89
the cv score for 9 fold: 0.89
the cv score for 10 fold: 0.89
```

We can see that the cross-validation results have improved slightly.

7. Concluding Remarks

In this article we made a machine learning regression model to predict the price of flight tickets. During this process we found many insights related to flight-ticket price dynamics. But is it enough? No. Many more insights can be deduced if we dive deeper into analysis.

We used XGBoost Regression Algorithm in our finalized model in which we tuned 3 hyper-parameters. More hyper-parameters could be tuned to improve the model further.

Thanks for reading this article. Have a good day.