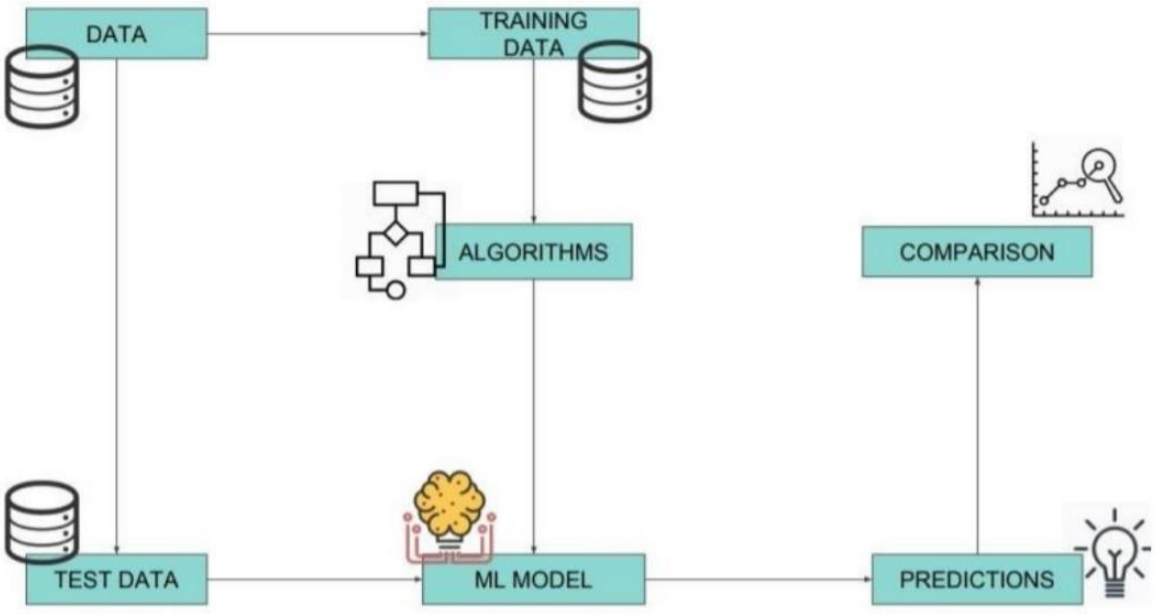
Flight Price Prediction Project Blog

In this blog we are going to use machine learning to predict the flight price of different airlines during the period of March to June 2019 using different parameters as below

1. **Airline**: The name of the airline.
2. **Date of Journey**: The date of the journey
3. **Source**: The source from which the service begins.
4. **Destination**: The destination where the service ends.
5. **Route**: The route taken by the flight to reach the destination.
6. **Dep Time**: The time when the journey starts from the source.
7. **Arrival Time**: Time of arrival at the destination.
8. **Duration**: Total duration of the flight.
9. **Total Stops**: Total stops between the source and destination.
10. **Additional Info**: Additional information about the flight
11. **Price**: The price of the ticket

Airlines use complex algorithms to calculate the price of ticket on any given day using the above parameters. Apart from the above parameters the price also gets affected by financial, marketing and social factors into account.

The flight price data flow can be seen as below



The above diagram gives us the different data sets used to predict the price. Here we are using the training and test data set from the GitHub using the following link

" https://github.com/dsrscientist/Data-Science-ML-Capstone-Projects/blob/master/Flight_Ticket_Participant_Datasets-20190305T100527Z-001.zip "

and we get training set which comprises 10683 rows and 11 attributes and test set containing 2671 rows and 10 attributes.

Before we head into tuning and processing of the data lets first import the needed libraries into the Jupyter notebook.

```
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import missingno
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import metrics
from scipy.stats import zscore
from sklearn.preprocessing import OrdinalEncoder, StandardScaler, MinMaxScaler, power_transform
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

Then using the read function, we read the training dataset and see the first five and last five rows of our entire dataset. It looks like the only column that has proper numeric data is "Price" which is also our target column making all the remaining columns as our features using which we need to predict our label. Since the values present in the "Price" column has continuous data it makes this to be a Regression problem!

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302
...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

Contents of the article

This article explains the complete process to build a machine learning model. Below mentioned are the various phases that we will go through, throughout the project -

1. Exploratory data analysis and Data modelling
2. Visualization
3. Encoding the data — Original Encoder
4. Outliers Presence and Finding
5. Skewness Presence and Finding
6. Correlation Using Heatmap
7. Splitting the dataset into 2 variables namely 'X' and 'Y' for features and label
8. Scaling the data — Standard scaler
9. Finding the best random state for building Regression Models
10. Feature Importance Bar Graph
11. Machine Learning Model for Regression
12. Model hyperparameter tuning on ML Model
13. Saving the final model and prediction using saved model

EDA (Exploratory Data Analysis)

Checking if null values are present in our dataset using info() function and checking the datatypes of the different variables.

```
df_train.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
```

We can see that there are 10 object data type columns and 1 integer data type column.

```
df_train.isna().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
```

We can see that there are 2 columns "Route" and "Total Stops" that have missing data.

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

```
Rows and Columns before dropping duplicates:  (10683, 11)
Rows and Columns after dropping duplicates:  (10463, 11)
```

In the above cell we have removed all the duplicate rows and checked for the row and column information.

```
print(f"Rows and Columns before dropping null values: ", df_train.shape)
df_train.dropna(inplace=True)
print(f"Rows and Columns after dropping null values: ", df_train.shape)
```

```
Rows and Columns before dropping null values:  (10463, 11)
Rows and Columns after dropping null values:  (10462, 11)
```

In the above cell we have removed all the null valued rows and checked for the row and column information.

So, we have checked for duplicate values and null values and dropped them using the shape command.

```
df_train.isnull().sum()
```

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

Now we can see that there are no null values present in our dataset.

```
df_train.nunique().to_frame("Unique Values")
```

	Unique Values
Airline	12
Date_of_Journey	44
Source	5
Destination	6
Route	128
Dep_Time	222
Arrival_Time	1343
Duration	368
Total_Stops	5
Additional_Info	10
Price	1870

The above cell gives us the number of unique values present in each column.

So, we have created a function that basically separates the date information into proper numerical format instead of making it an object datatype. Also have dropped the Date of Journey column since it's data was already bifurcated, then removed the Route column as we have source and destination data so Route was not adding much of an insight and finally removed the Duration column as later on I have time separations to deal with as well.

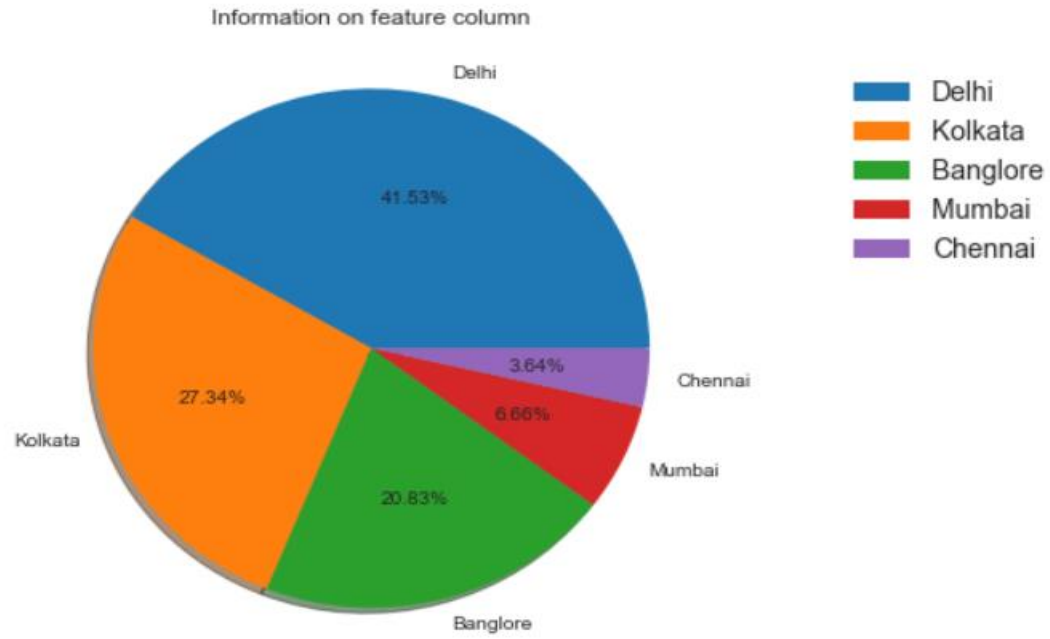
```
def date_bifurcation(df):
    df=df.copy()
    df['Date_of_Journey']=pd.to_datetime(df['Date_of_Journey'])
    df['Year_of_Journey']=df['Date_of_Journey'].dt.year
    df['Month_of_Journey']=df['Date_of_Journey'].dt.month
    df['Day_of_Journey']=df['Date_of_Journey'].dt.day
    df=df.drop(['Route', 'Date_of_Journey', 'Duration'],axis=1)
    return df

df_train = date_bifurcation(df_train)
print(f"Rows and Columns:", df_train.shape)
df_train.head()
```

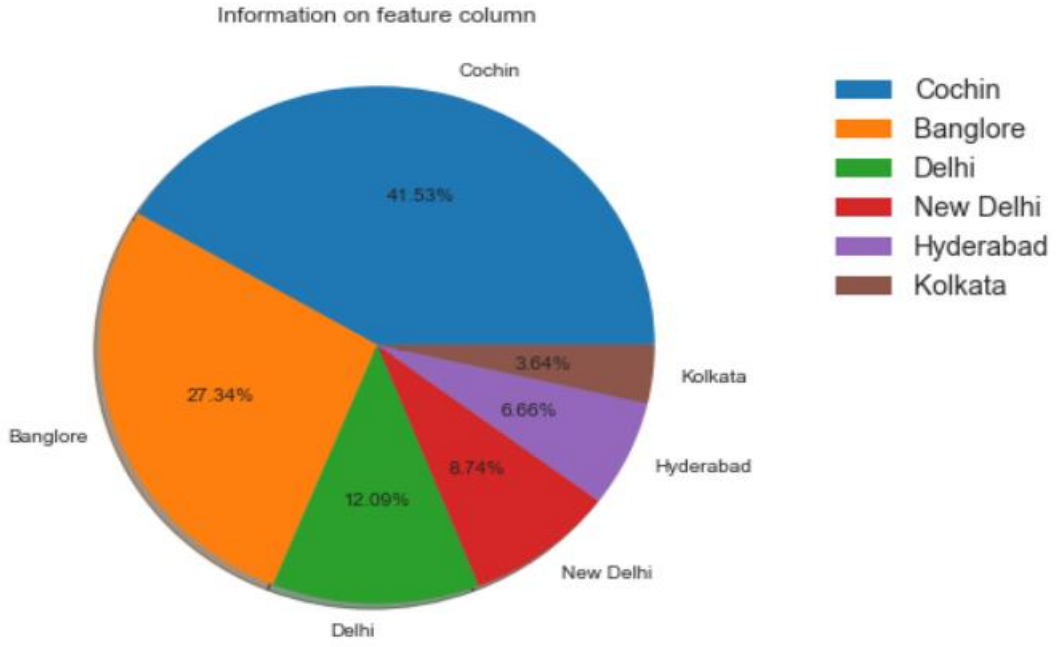
```
Rows and Columns: (10462, 11)
```

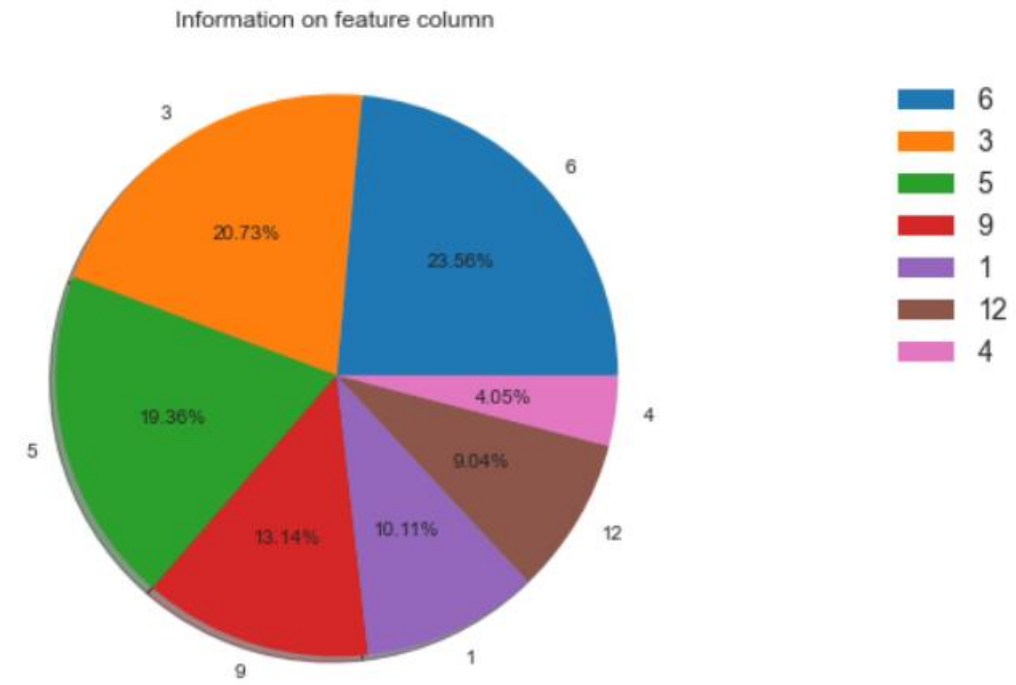
	Airline	Source	Destination	Dep_Time	Arrival_Time	Total_Stops	Additional_Info	Price	Year_of_Journey	Month_of_Journey	Day_of_Journey
0	IndiGo	Banglore	New Delhi	22:20	01:10 22 Mar	non-stop	No info	3897	2019	3	24
1	Air India	Kolkata	Banglore	05:50	13:15	2 stops	No info	7662	2019	1	5
2	Jet Airways	Delhi	Cochin	09:25	04:25 10 Jun	2 stops	No info	13882	2019	9	6
3	IndiGo	Kolkata	Banglore	18:05	23:30	1 stop	No info	6218	2019	12	5
4	IndiGo	Banglore	New Delhi	16:50	21:35	1 stop	No info	13302	2019	1	3

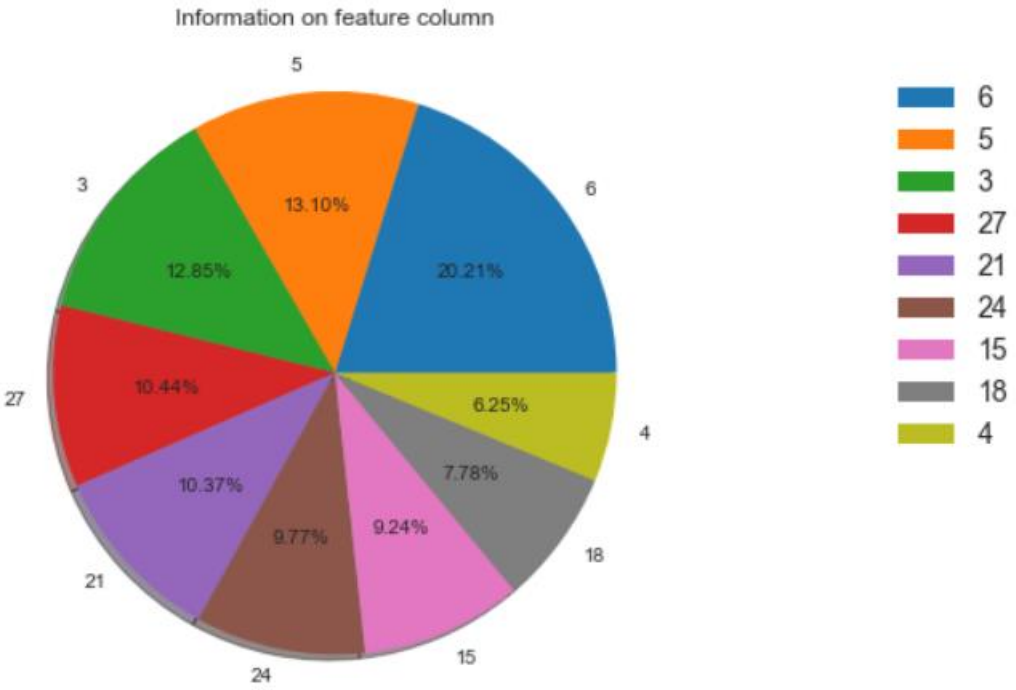
Created another function that deals with the separation on timings for arrival and departure. This allows us to get a proper insight on the flight duration details and got rid of the duration, Route and Date of Journey column.

Visualization



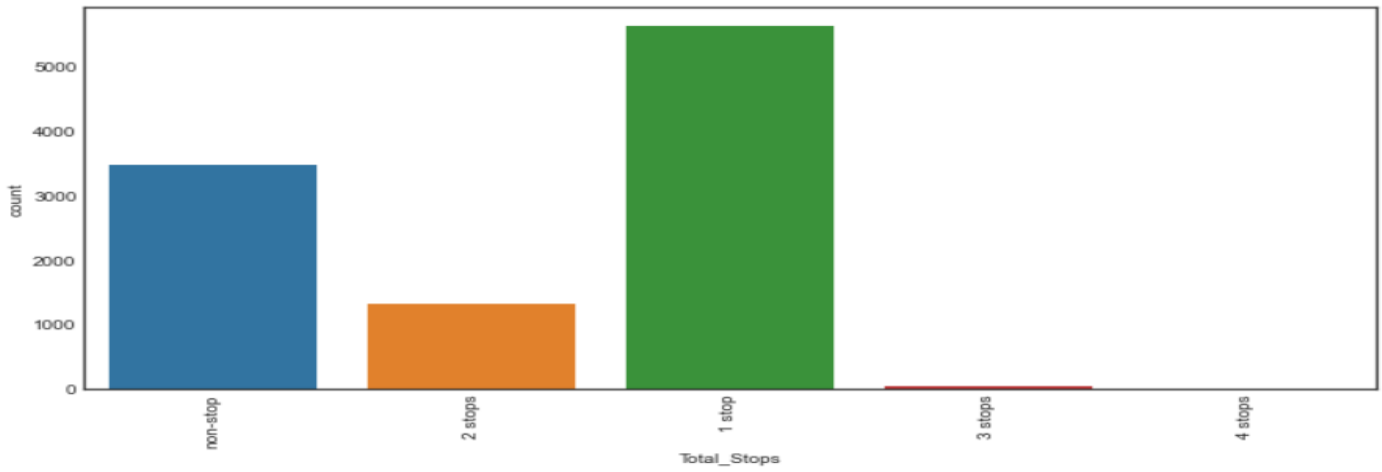


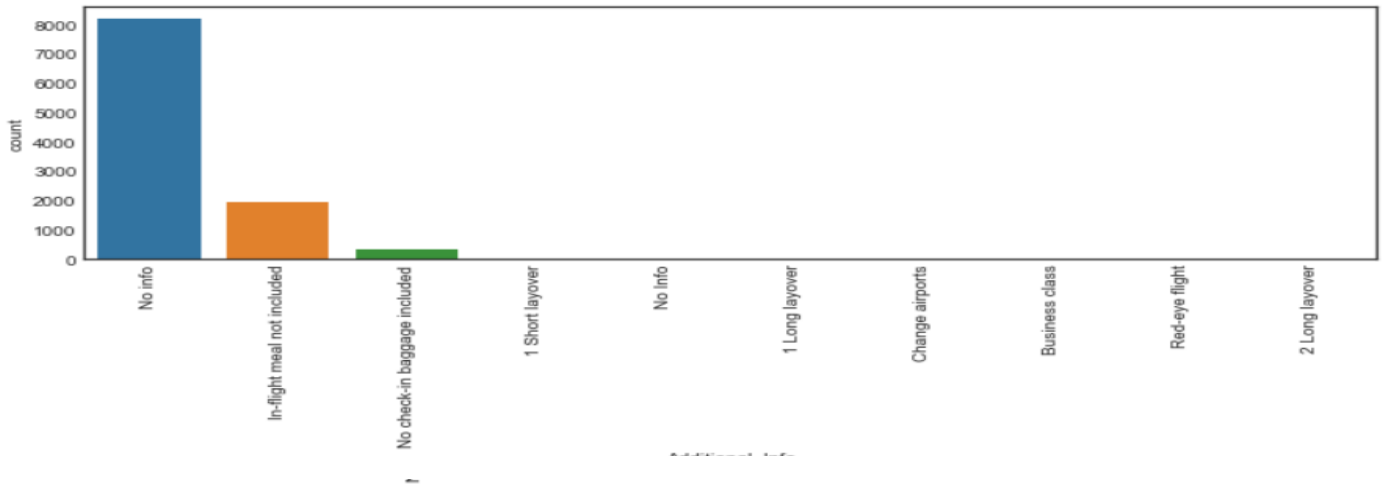


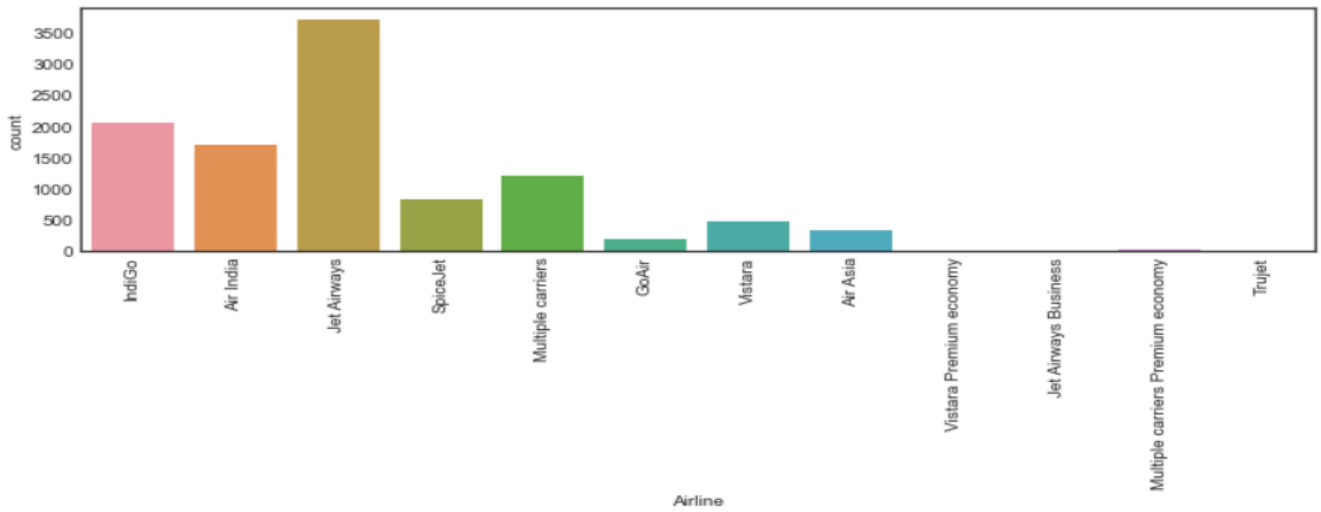


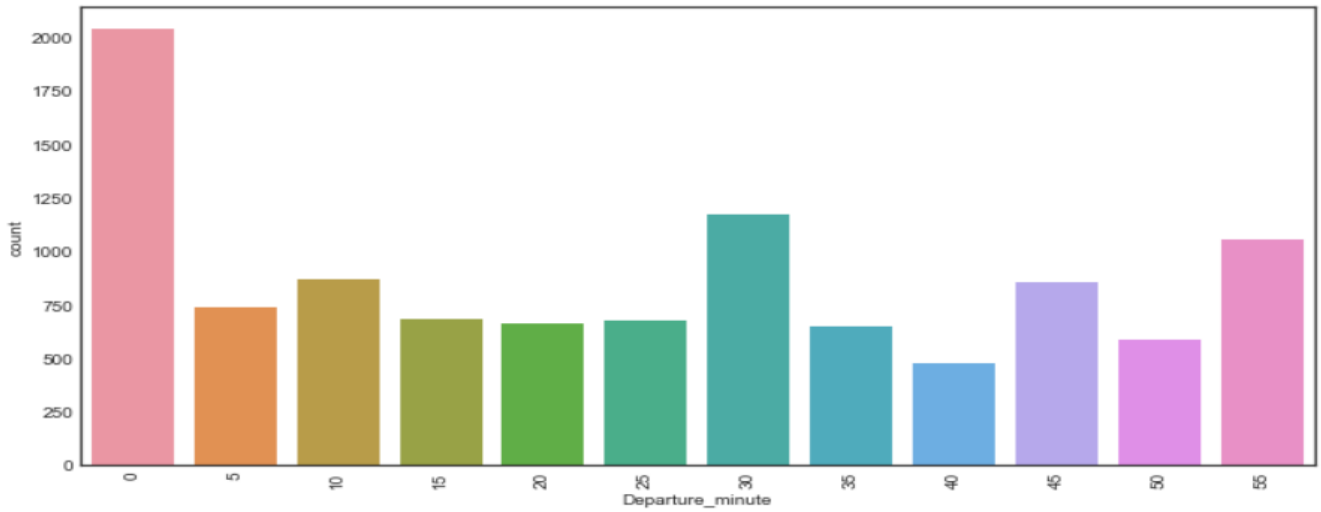
Have created a function to generate pie plots for our feature columns and we can observe:

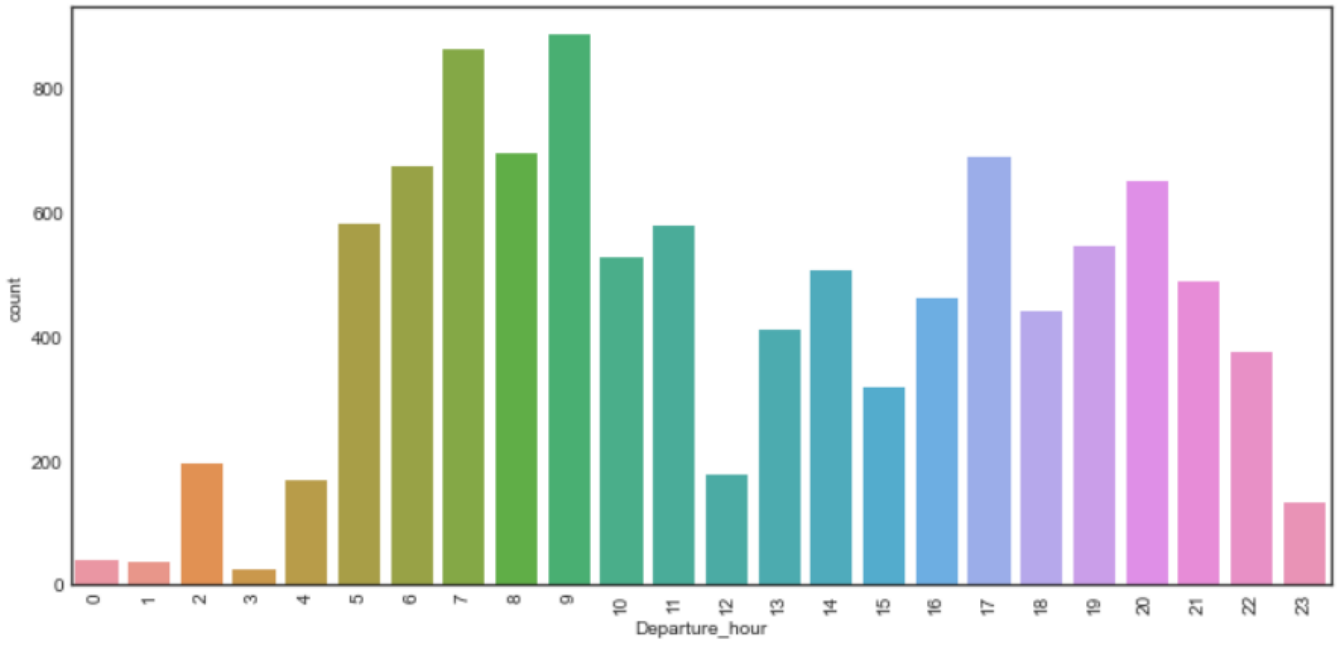
1. In the source column we have the highest number of rows covered by Delhi and the lowest number of rows covered by Chennai
2. In the destination column we have the highest number of rows covered by Cochin and the lowest number of rows covered by Kolkata
3. We see that in our data set that most of the journey made by people were in the month of June and least in the month of April
4. We also see that in our data set that most of the journey made by people were on dates 6,5 and the least travelling dates were for 4,18

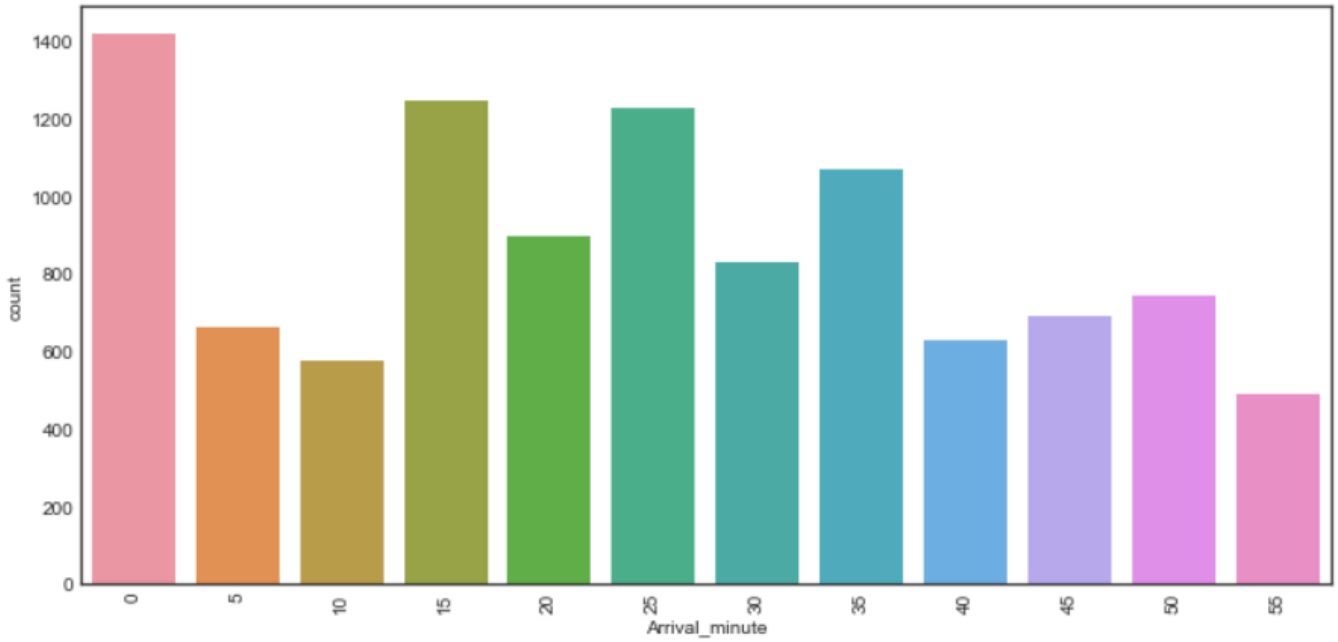

count
5000
4000
3000
2000
1000
0
non-stop
2 stops
1 stop
3 stops
4 stops
Total_Stops

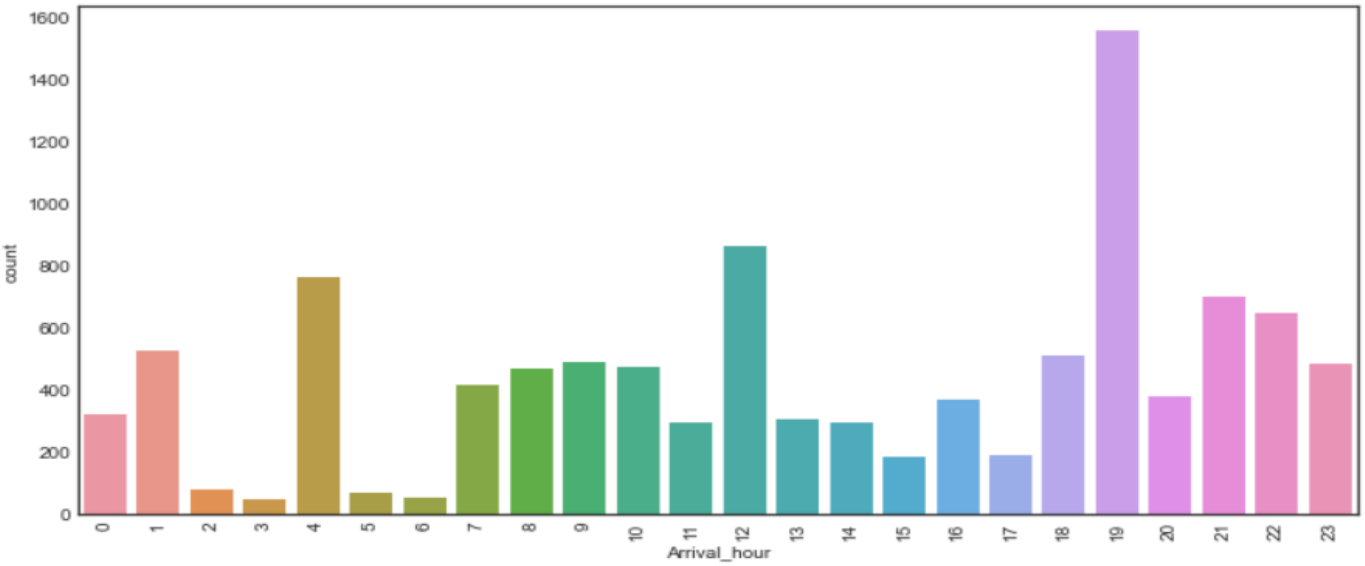

count
8000
7000
6000
5000
4000
3000
2000
1000
0
No info
In-flight meal not included
No check-in baggage included
1 Short layover
No Info
1 Long layover
Change airports
Business class
Red-eye flight
2 Long layover
Additional_Info


count
3500
3000
2500
2000
1500
1000
500
0
IndiGo
Air India
Jet Airways
SpiceJet
Multiple carriers
GoAir
Vistara
Air Asia
Vistara Premium economy
Jet Airways Business
Multiple carriers Premium economy
Trujet
Airline


count
2000
1750
1500
1250
1000
750
500
250
0
0
5
10
15
20
25
30
35
40
45
50
55
Departure_minute

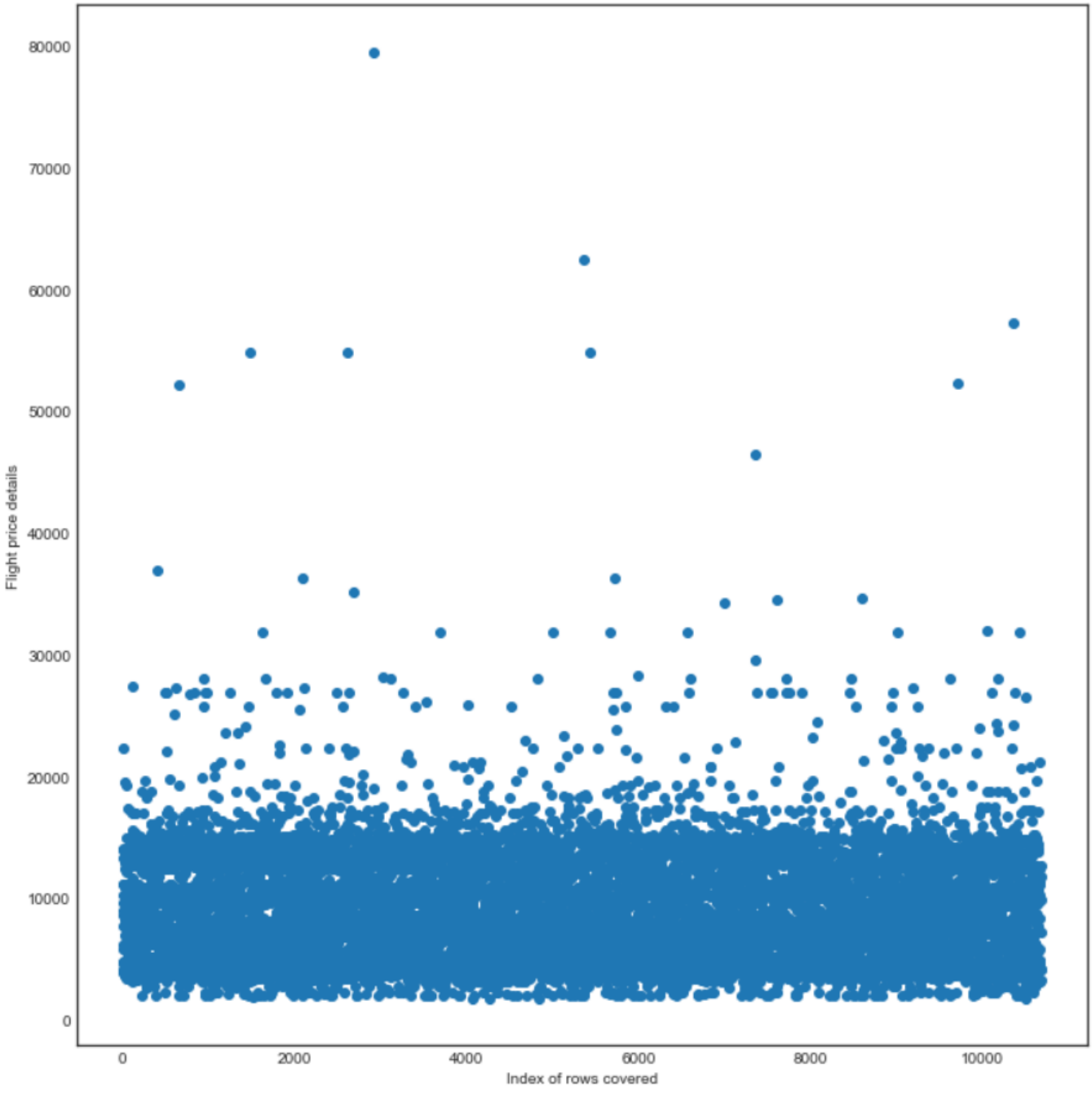

count
0
200
400
600
800
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Departure_hour


count
0
200
400
600
800
1000
1200
1400
0 5 10 15 20 25 30 35 40 45 50 55
Arrival_minute

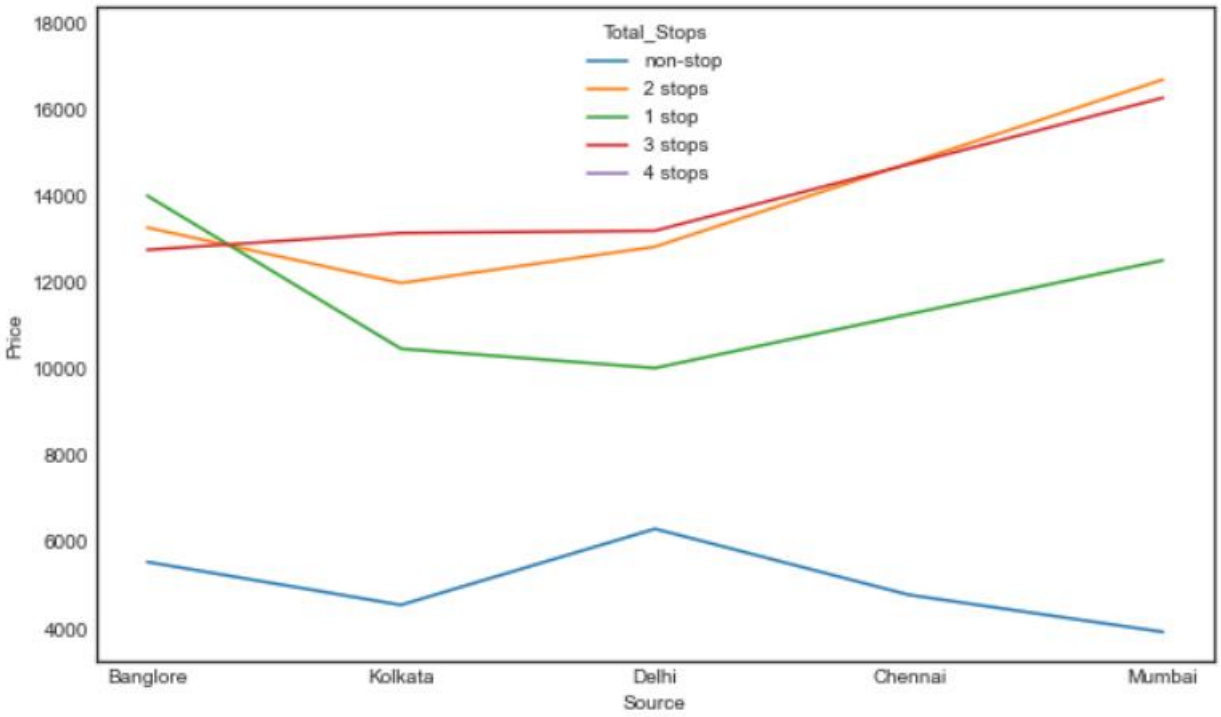

count
0
200
400
600
800
1000
1200
1400
1600
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Arrival_hour

Have created a function to generate count plots for our feature columns and we can observe:

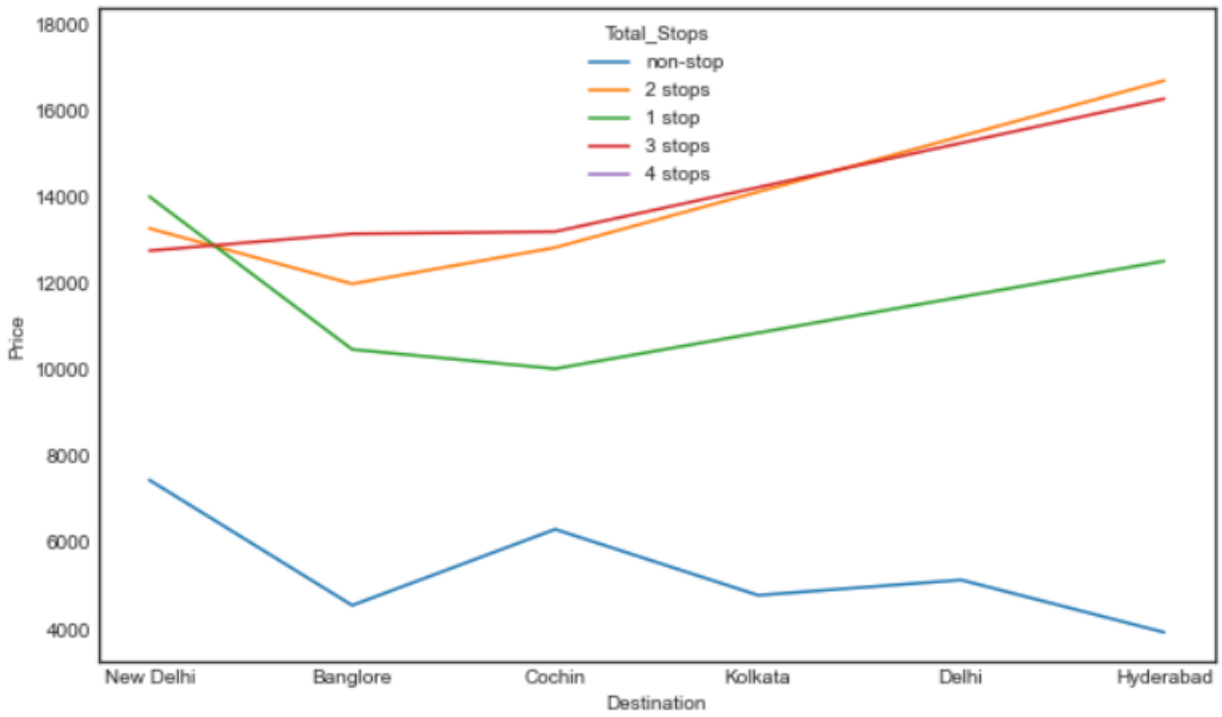
1. In the total stop's column, we see highest count of rows covered by 1 stop flight hauls and the least numbers are for 3 and 4 stops
2. In additional information column most number of rows are covered by no info values and rest of the values cover very less to negligible data points
3. The airline column shows that highest number of flight details are present in our dataset for Jet Airways followed by Indigo and Air India
4. The departure minute column gives us the indication that most number of flights get scheduled at 0 minutes for departure
5. The departure hour column gives us the indication that most number of flights get scheduled at 7 and 9 hour morning time and then there is a spike at 17 and 20 hour evening time.
6. The arrival minute column gives us the indication that most number of flights get scheduled at 0 minutes for arrival
7. The arrival hour column gives us the indication that most number of flights get scheduled at 19 hour in the evening and then the chosen option for arrivals are 12 in the noon or 4 in the night



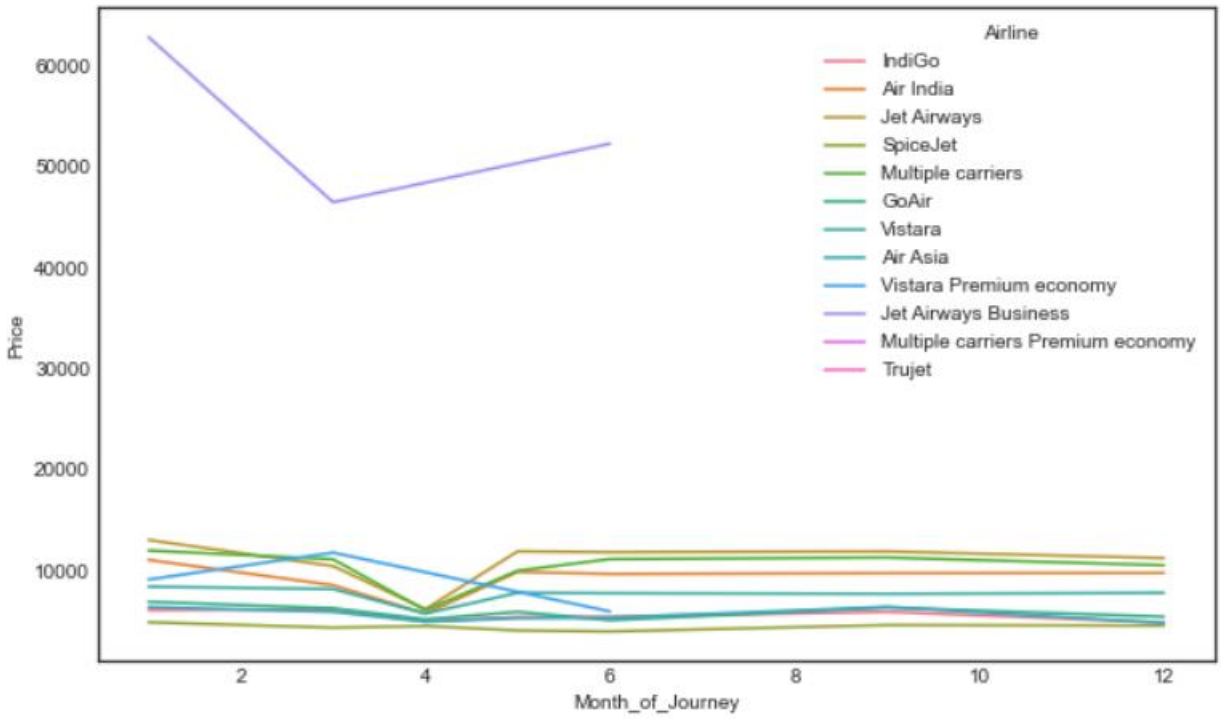
In the above scatter plot, we are able to see that most of the flight price values are accumulated between 0-20000 and very rare data points are distributed above that number.



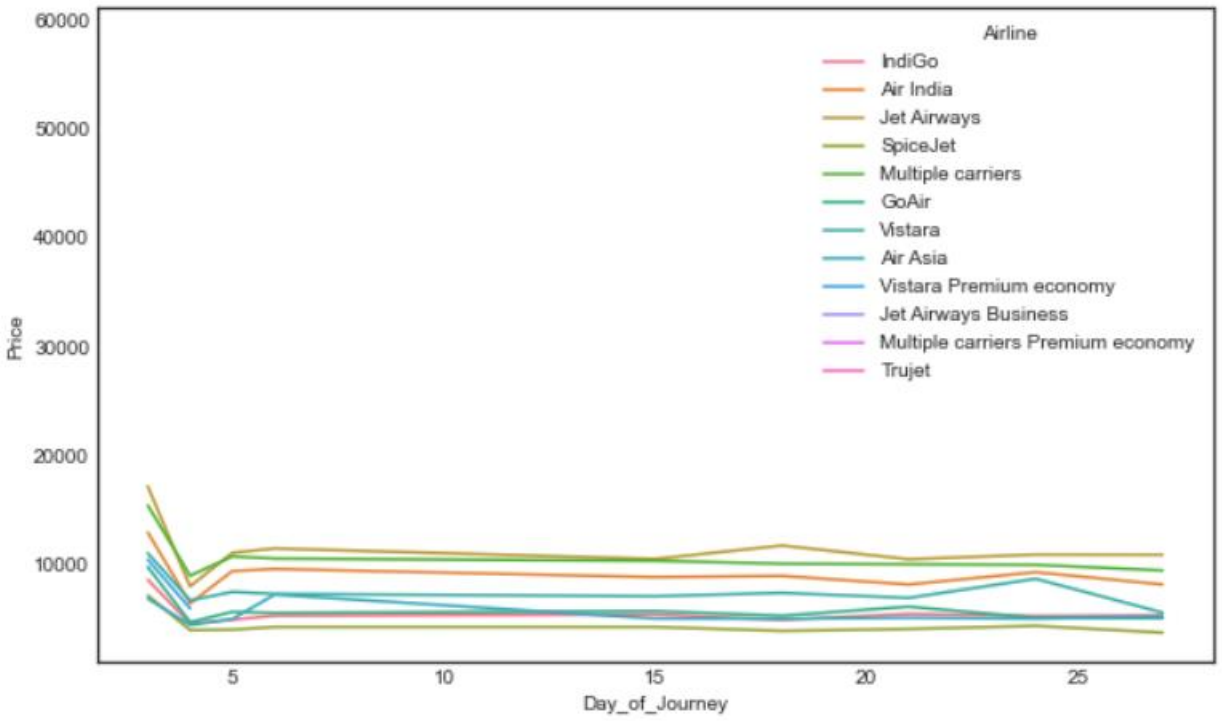
In the above line plot, we see that non-stop flights have lower price irrespective of the source as compared to flights that have 1 or more than 1 stops in the flight haul.



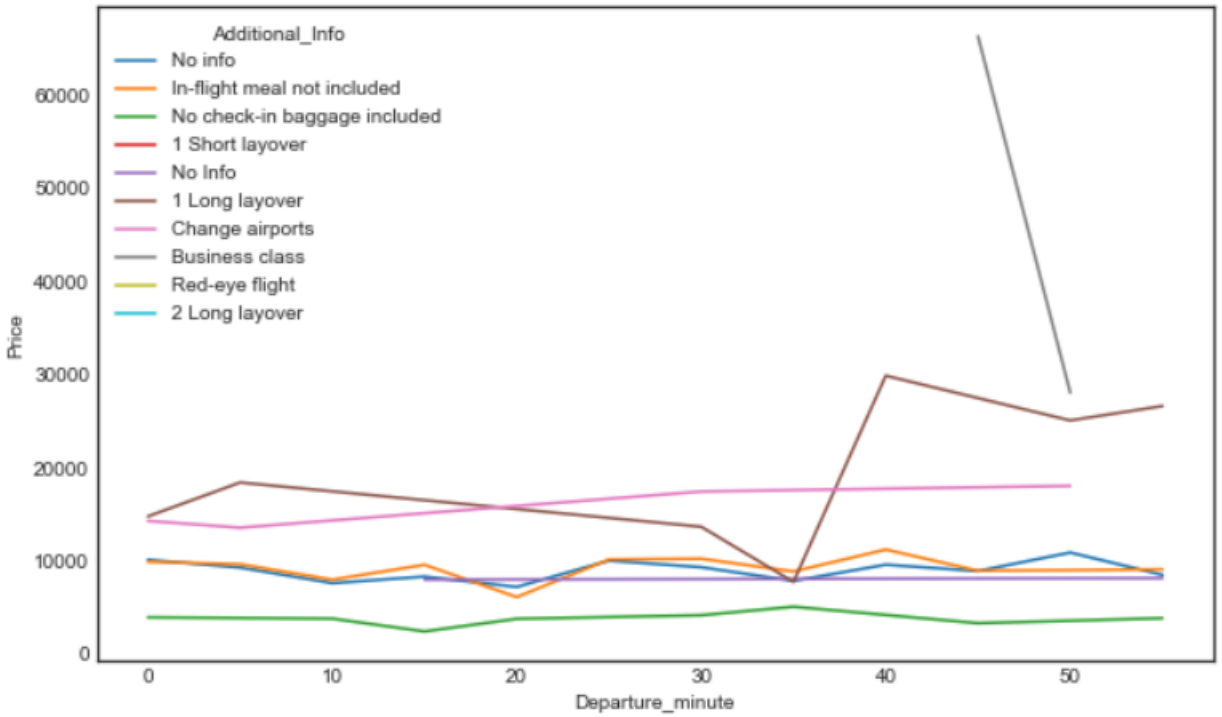
In the above line plot, we see that non-stop flights have lower price irrespective of the destination as compared to flights that have 1 or more than 1 stops in the flight haul.



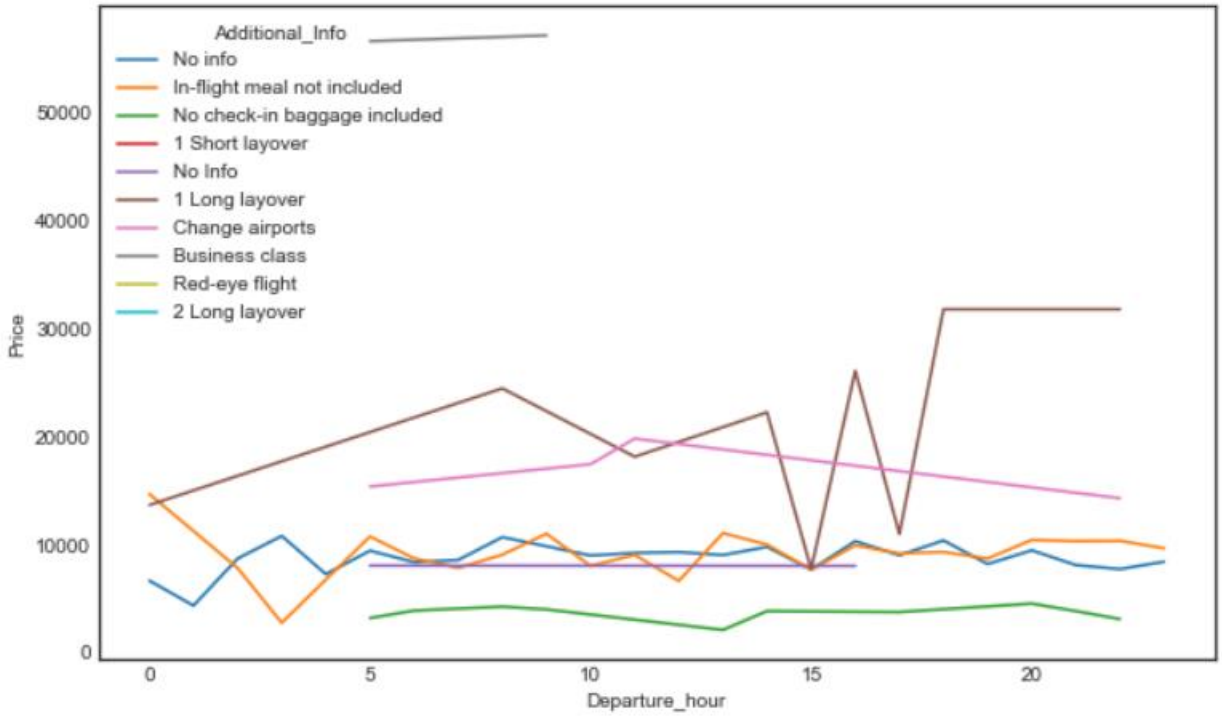
In the above line plot, we see that Jet Airways Business class has the highest price than the rest possibly because the remaining offer the economy class data.



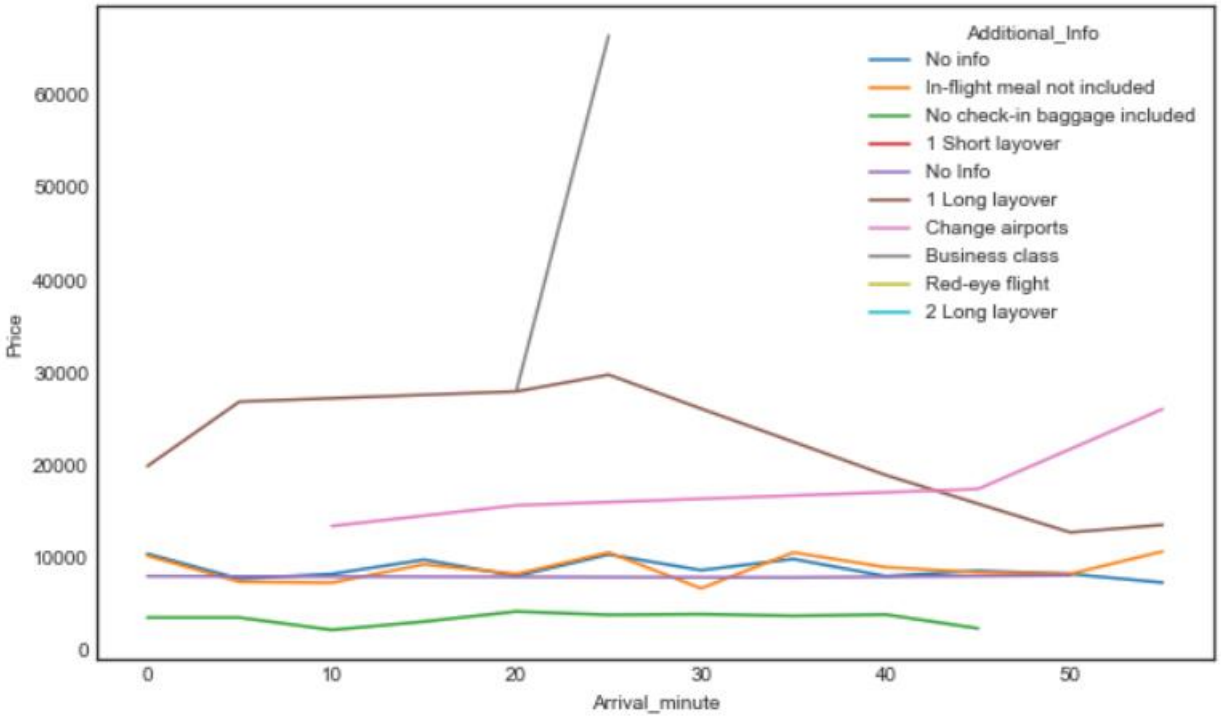
In the above line plot, we see that all the airlines have high price between 1-5 days of a month and that reduces a bit on rest of the days apart from them.



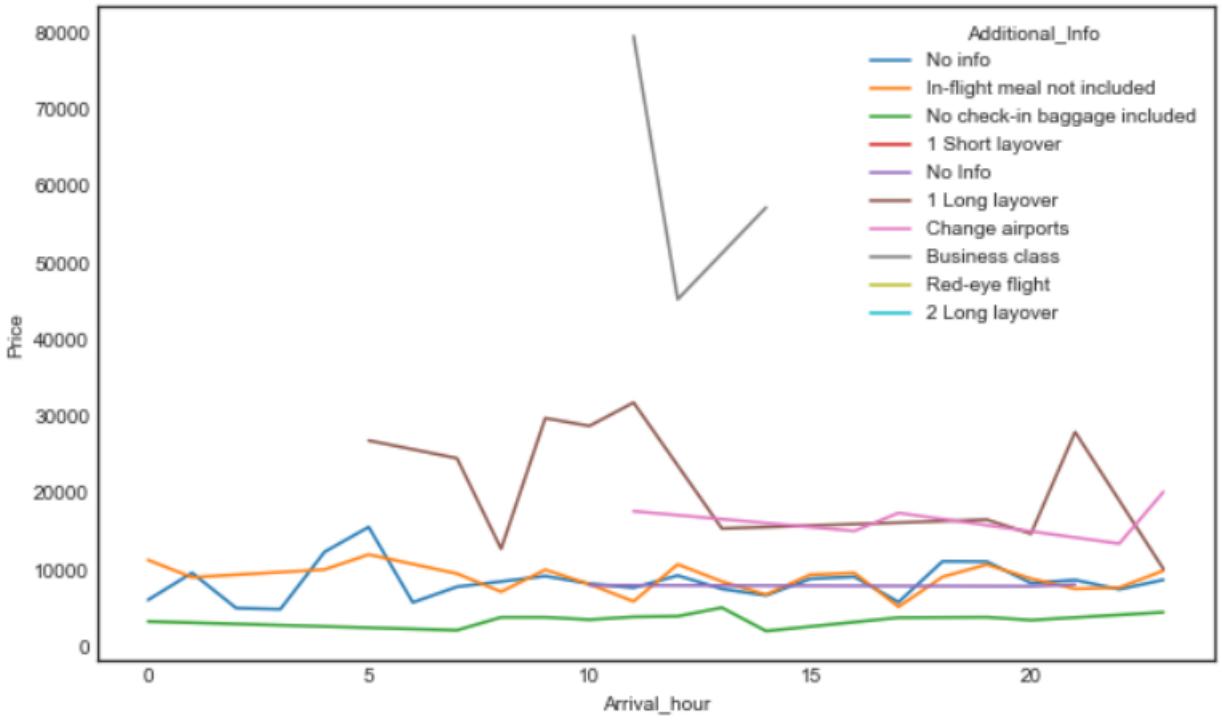
In the above line plot, we see that Business class has high price and has data coverage for departure minutes between 45-50 minutes roughly.



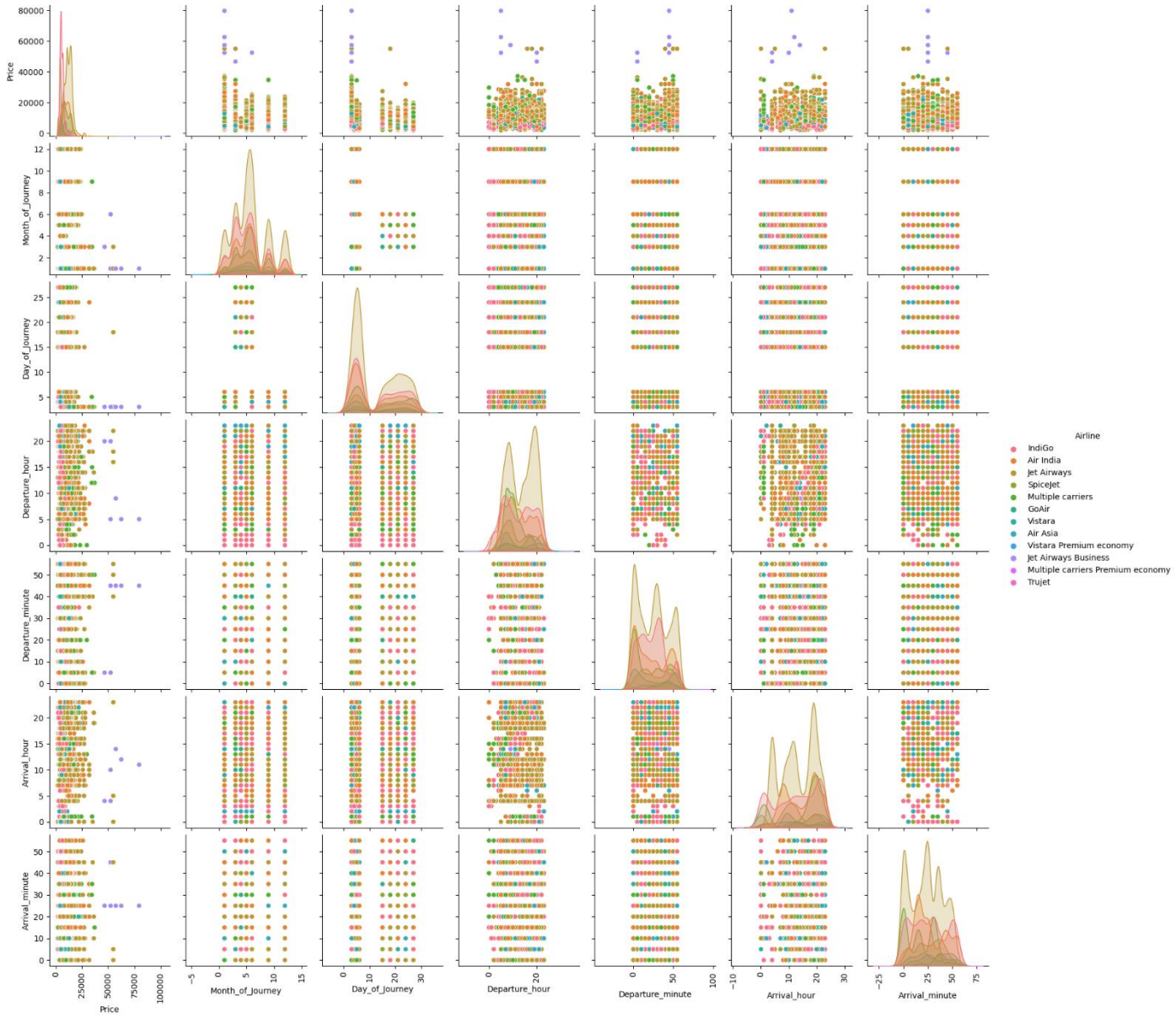
In the above line plot, we see that business class has high price and it's departure hour is between 5-10 but the second highest pricing is for 1 long layover type with spike in between 17-22 departure hour.



In the above line plot, we see that business class again has an exponential price rise and the arrival minutes mostly range between 20-30 minutes



In the above line plot, we see that price for no check-in bag included is least as compared to the business class being highest and the arrival hour for business class is spread only between 10-15 minutes.

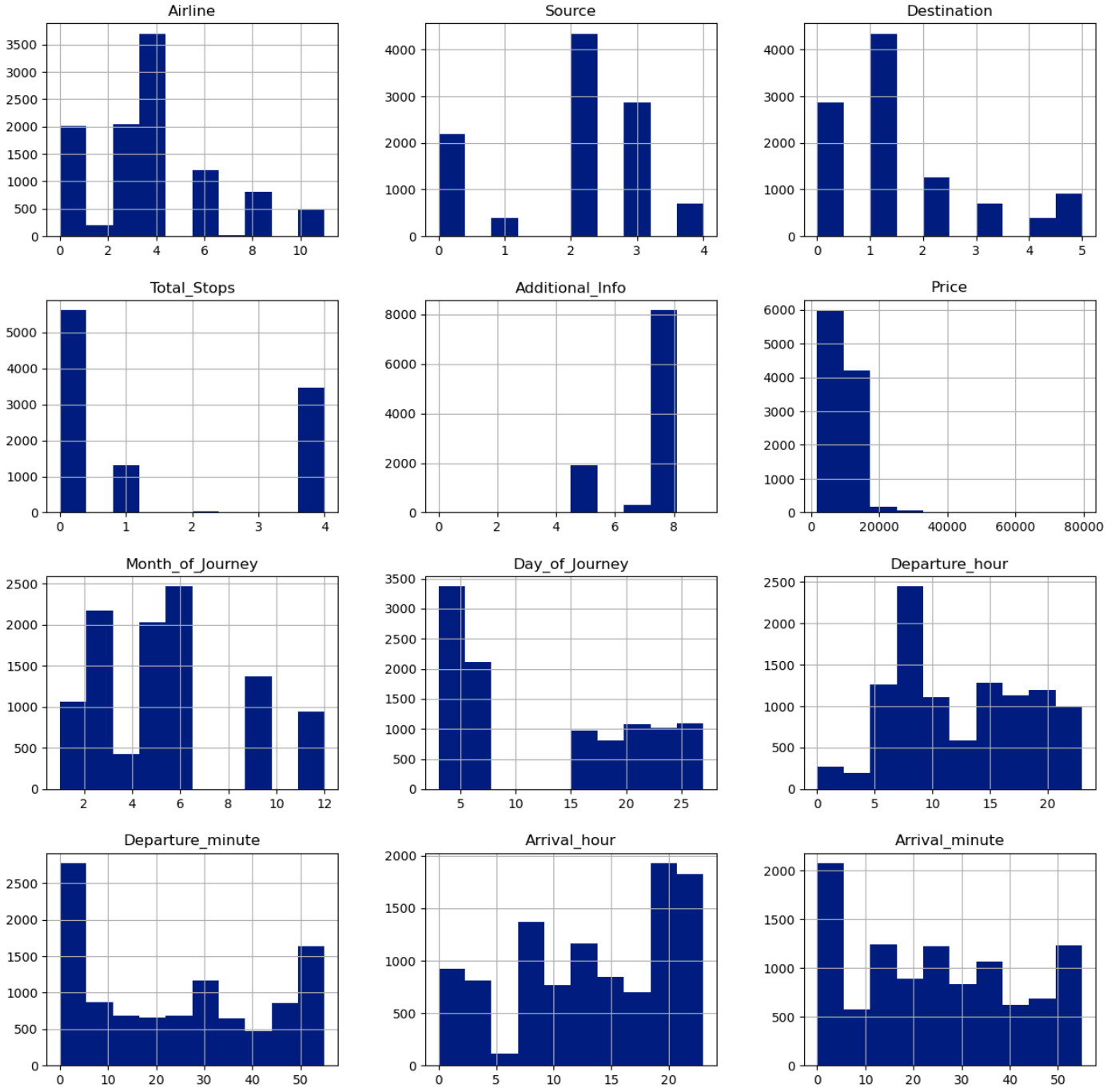


The above pair plot gives us an indication on the numerical data considering the different airlines present in our dataset.

Encoding The Data

```
oe=OrdinalEncoder()
def ordinal_encoder(df, col):
    df[col]=oe.fit_transform(df[col])
    return df

df_train=ordinal_encoder(df_train,['Airline', 'Source', 'Destination', 'Total_Stops', 'Additional_Info'])
```

Using the “Ordinal Encoder” method for encoding my categorical features since they all are present in an orderly format and will not increase the number of columns like in the usage of One Hot Encoding.

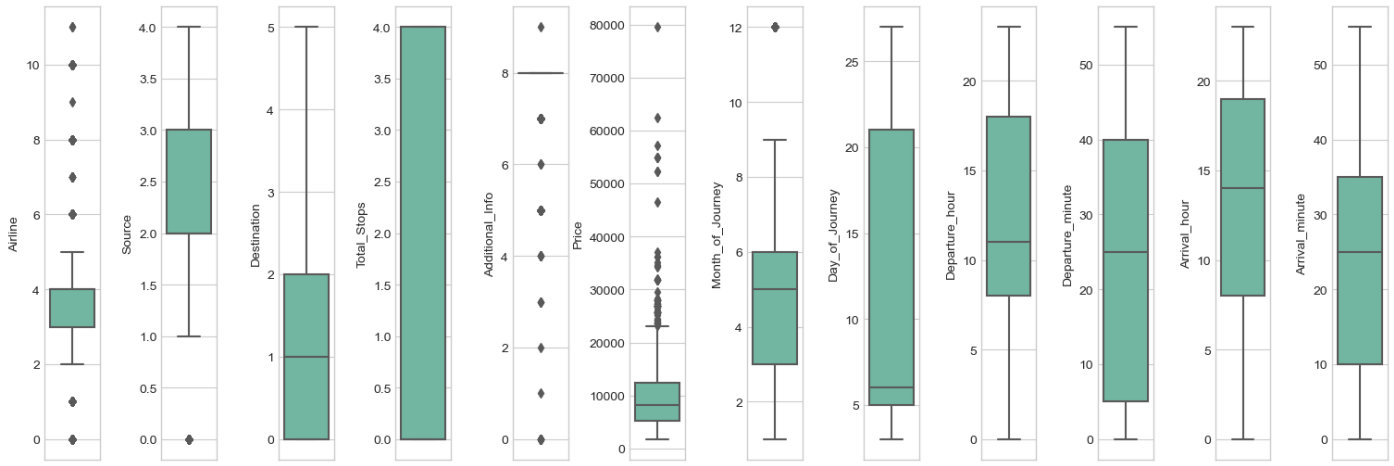


With the help of above histogram, we are able to see the data distribution in our columns after the encoding technique was applied on all the categorical columns.

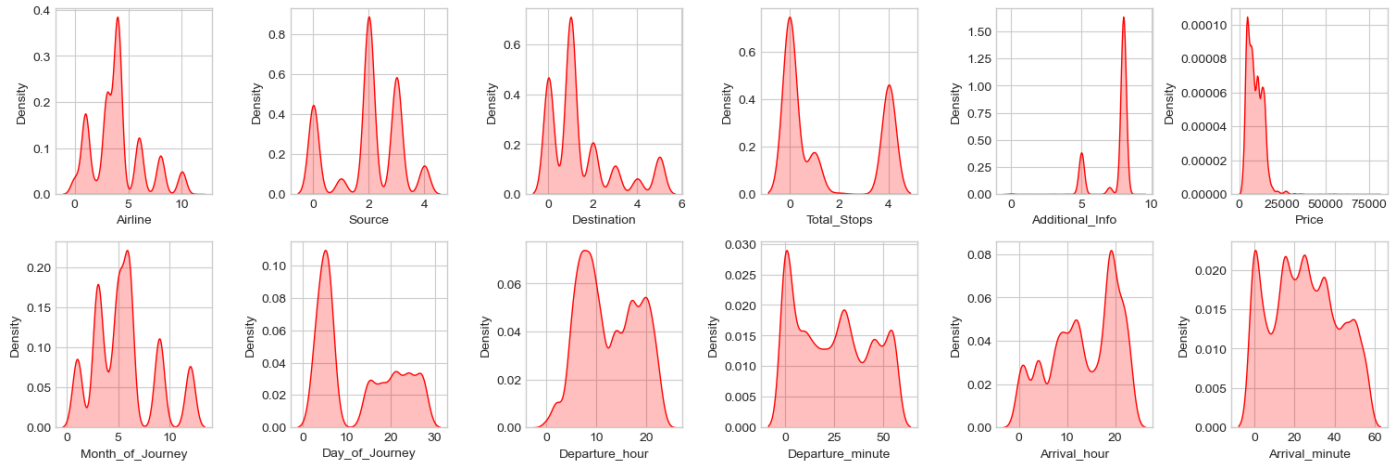
Outliers Presence and Finding

```
plt.style.use('seaborn-whitegrid')
fig, ax = plt.subplots(ncols=12, nrows=1, figsize=(15,5))
index = 0
ax = ax.flatten()
for col, value in df_train.items():
    sns.boxplot(y=col, data=df_train, ax=ax[index], palette="Set2")
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.4, h_pad=1.0)
plt.show()
```

Using the box plot we are able to notice the outliers present in our dataset but since all of the feature columns are categorical data, we will not have to worry about the presence of outliers here.



Skewness Presence and Finding



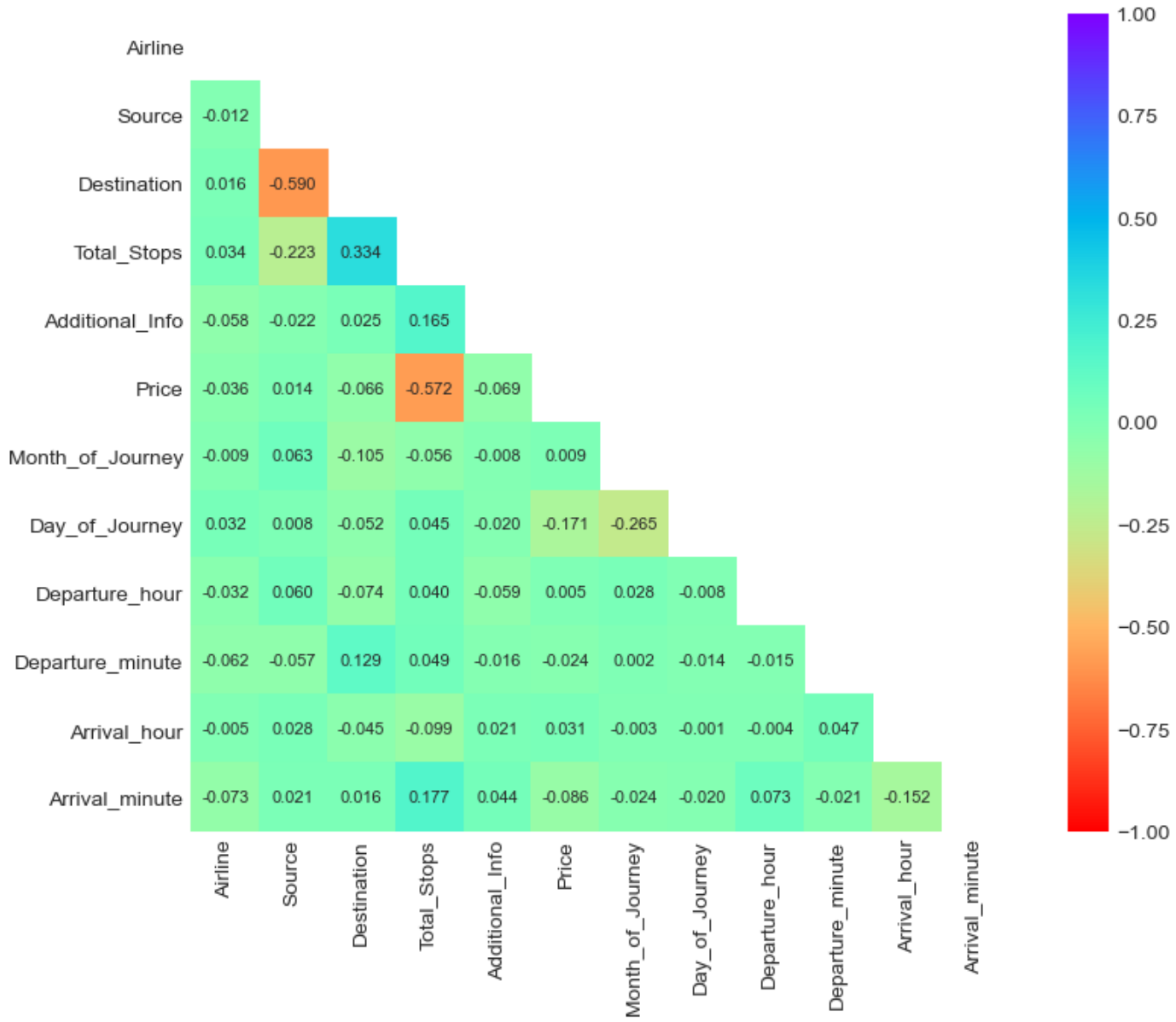
Just like in the case of outliers with the help of above distribution plots on our dataset gives us an indication on presence of skewness in our columns however for categorical data columns we do not have to worry about the outliers or the skewness in them.

Correlation Using Heatmap

-> Positive correlation - A correlation of +1 indicates a perfect positive correlation, meaning that both variables move in the same direction together.

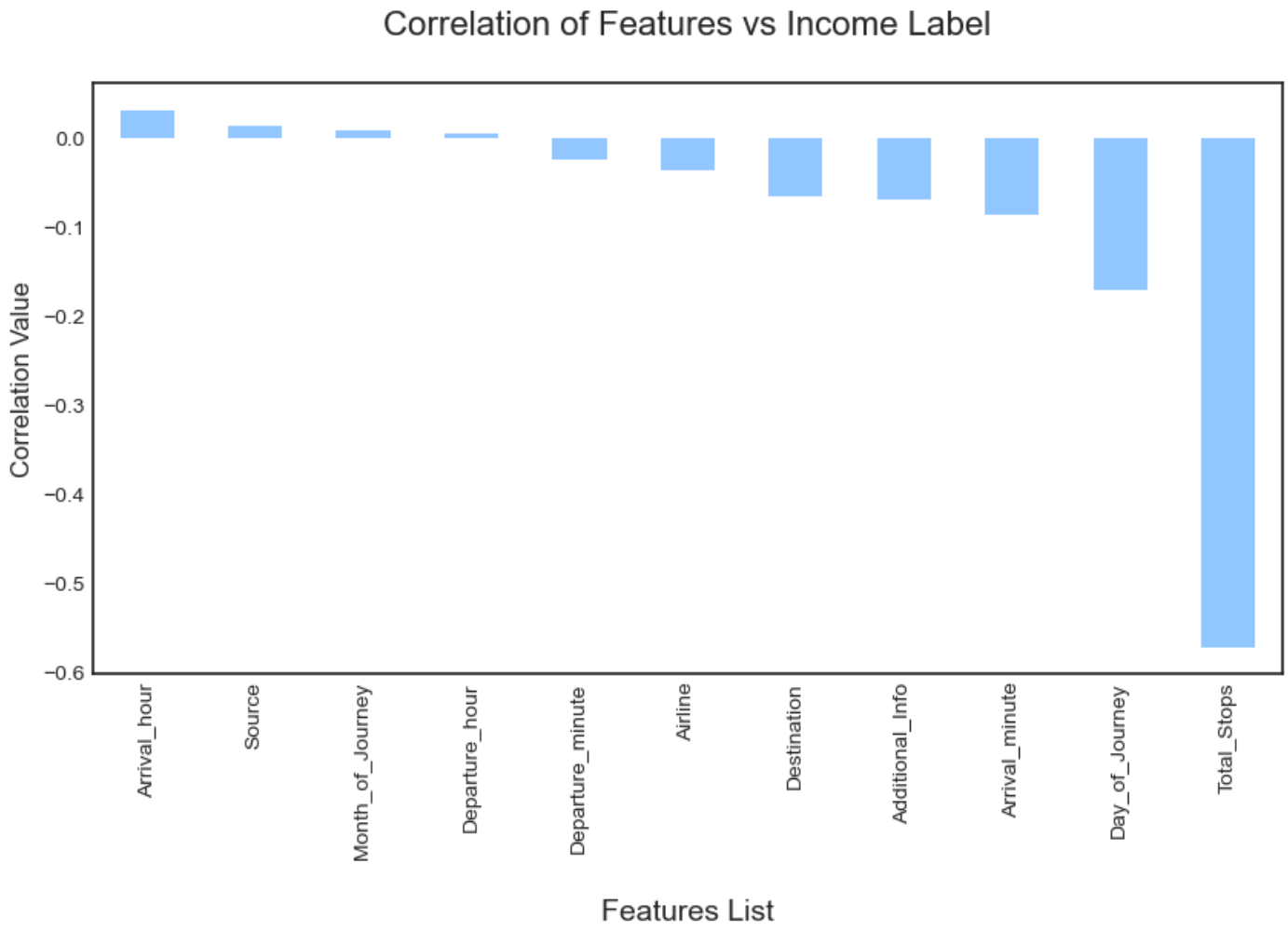
-> Negative correlation - A correlation of –1 indicates a perfect negative correlation, meaning that as one variable goes up, the other goes down.

With the help of about heatmap we are able to notice the correlation details between our label and feature and also amongst our labels. After eye bawling the above heatmap we can see that there are no multi collinearity concerns in our dataset so we won't have to worry about dealing with them.



Correlation Using Bar Plot

With the usage of above bar plot for viewing the correlation information between our Features and Label we can see that only column arrival hour, source, month of journey and departure hours are positively correlated with our target and remaining all our feature columns are negatively correlated where total stops is highly negatively correlated indicating that as the number of total stops in an itinerary increases the price of that particular flight increases and vice a versa.



Splitting the dataset into 2 variables namely 'X' and 'Y' for features and label

```
X = df_train.drop('Price', axis=1)
Y = df_train['Price']
```

We have separated the dataset into features and label where X represents all the feature columns and Y represents the regression target label column.

Feature Scaling

```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head() # Displaying all the features after applying scaling technique to avoid bias output
```

Using the “Standard Scaler” method to normalize my feature values and ensure that my regression model does not have any kind of biasness towards a particular column value.

Airline	Source	Destination	Total_Stops	Additional_Info	Month_of_Journey	Day_of_Journey	Departure_hour	Departure_minute	Arrival_hour	Arrival_
-0.414916	-1.646877	2.404213	1.392916	0.497963	-0.846221	1.293326	1.662578	-0.234123	-1.807045	-0.8883
-1.260756	0.882057	-0.972012	-0.254274	0.497963	-1.513956	-0.867922	-1.305841	1.360364	-0.056554	-0.5866
0.008004	0.039079	-0.296767	-0.254274	0.497963	1.156984	-0.754172	-0.607390	0.031625	-1.369422	0.0169(
-0.414916	0.882057	-0.972012	-0.803337	0.497963	2.158586	-0.867922	0.964126	-1.031367	1.402189	0.3186[
-0.414916	-1.646877	2.404213	-0.803337	0.497963	-1.513956	-1.095422	0.614900	1.360364	1.110440	0.6204[

Finding the best random state for building Regression Models

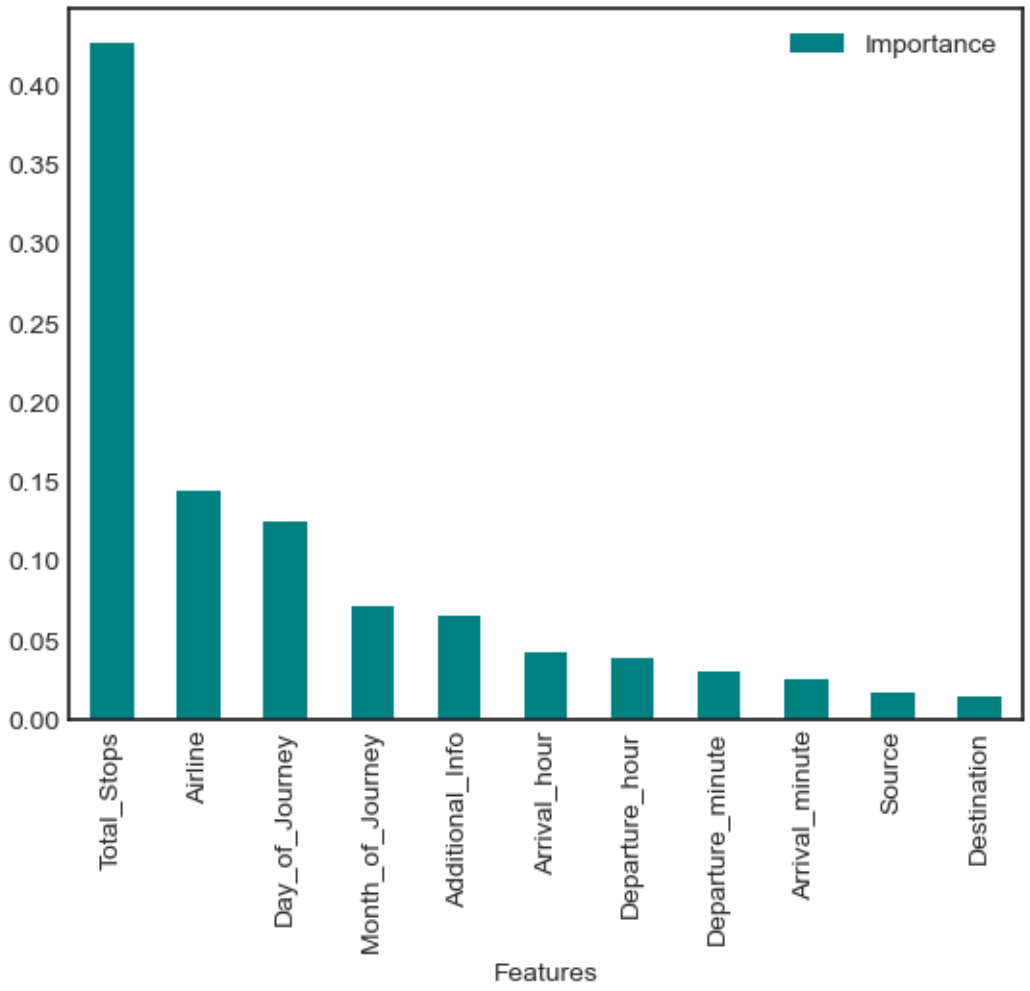
```
maxAccu=0
maxRS=0

for i in range(1, 1000):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=i)
    lr=LinearRegression()
    lr.fit(X_train, Y_train)
    pred = lr.predict(X_test)
    r2 = r2_score(Y_test, pred)

    if r2>maxAccu:
        maxAccu=r2
        maxRS=i

print("Best R2 score is", maxAccu,"on Random State", maxRS)
```

```
Best R2 score is 0.4194390126695826 on Random State 754
```

Feature importance bar graph

```
rf=RandomForestRegressor()
rf.fit(X_train, Y_train)
importances = pd.DataFrame({'Features':X.columns, 'Importance':np.round(rf.feature_importances_,3)})
importances = importances.sort_values('Importance', ascending=False).set_index('Features')
importances.plot.bar(color='teal')
importances
```

	Importance
Features	
Total_Stops	0.426
Airline	0.144
Day_of_Journey	0.125
Month_of_Journey	0.071
Additional_Info	0.066
Arrival_hour	0.042
Departure_hour	0.039
Departure_minute	0.030
Arrival_minute	0.025
Source	0.017
Destination	0.015



Here with the help of "`RandomForestRegressor`" we are able to list down the importance or priority given to a column as per its involvement or weightage in predicting our label.

Machine Learning Model for Regression with Evaluation Metrics

Built a regression function that splits the training and testing features and labels, then trains the model, predicts the label, calculates the RMSE score, generates the R2 score, calculates the Cross Validation score and finally finds the difference between the R2 score and Cross Validation score.

```
# Regression Model Function

def reg(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=754)

    # Training the model
    model.fit(X_train, Y_train)

    # Predicting Y_test
    pred = model.predict(X_test)

    # RMSE - a lower RMSE score is better than a higher one
    rmse = mean_squared_error(Y_test, pred, squared=False)
    print("RMSE Score is:", rmse)

    # R2 score
    r2 = r2_score(Y_test, pred, multioutput='variance_weighted')*100
    print("R2 Score is:", r2)

    # Cross Validation Score
    cv_score = (cross_val_score(model, X, Y, cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    # Result of r2 score minus cv score
    result = r2 - cv_score
    print("R2 Score - Cross Validation Score is", result)
```

```
# XGB Regressor

model=XGBRegressor()
reg(model, X, Y)
```

```
RMSE Score is: 1408.8013601931516
R2 Score is: 89.11286083744568
Cross Validation Score: 89.31337098569448
R2 Score - Cross Validation Score is -0.20051014824879587
```

Created the XGB regression model and checked for all its evaluation metrics as well.

```
Final_Model = XGBRegressor(booster='dart', eta=0.1, importance_type='gain', n_estimators=500)
Classifier = Final_Model.fit(X_train, Y_train)
fmod_pred = Final_Model.predict(X_test)
fmod_r2 = r2_score(Y_test, fmod_pred)*100
print("R2 score for the Best Model is:", fmod_r2)
```

```
R2 score for the Best Model is: 87.55683252006776
```

We have built our final model using the hyper tuned parameters which look like it is not doing better than the default values but it can always be tuned better to get better results.

Saving the Best Regression ML Model

```
filename = "FinalModel_Flight Price Prediction.pkl"
joblib.dump(Final_Model, filename)
```

```
['FinalModel_Flight Price Prediction.pkl']
```

Loading Test Data

```
df_test = pd.read_excel("Downloads\Flight_Ticket_Participant_Datasets\Test_set.xlsx")
df_test
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1 stop	No info
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	1 stop	In-flight meal not included
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1 stop	No info
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	23:55	02:45 25 Jun	2h 50m	non-stop	No info
...	...	...	...	...	...	...	...	...	...	...
2666	Air India	6/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:30	20:25 07 Jun	23h 55m	1 stop	No info
2667	IndiGo	27/03/2019	Kolkata	Banglore	CCU → BLR	14:20	16:55	2h 35m	non-stop	No info
2668	Jet Airways	6/03/2019	Delhi	Cochin	DEL → BOM → COK	21:50	04:25 07 Mar	6h 35m	1 stop	No info
2669	Air India	6/03/2019	Delhi	Cochin	DEL → BOM → COK	04:00	19:15	15h 15m	1 stop	No info
2670	Multiple carriers	15/06/2019	Delhi	Cochin	DEL → BOM → COK	04:55	19:15	14h 20m	1 stop	No info

Have imported the testing dataset now that only consists of feature columns and we need to predict the target label. However, before we apply our final regression model, we will need to perform all the pre-processing steps that were applied on our training dataset.

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

```
Rows and Columns before dropping duplicates:  (2671, 10)
Rows and Columns after dropping duplicates:  (2645, 10)
```

We are removing all the duplicate rows from our testing dataset using the drop_duplicates option.

```
df_test = date_bifurcation(df_test)
print(f"Rows and Columns:", df_test.shape)
df_test.head()
```

```
Rows and Columns: (2645, 10)
```

	Airline	Source	Destination	Dep_Time	Arrival_Time	Total_Stops	Additional_Info	Year_of_Journey	Month_of_Journey	Day_of_Journey
0	Jet Airways	Delhi	Cochin	17:30	04:25 07 Jun	1 stop	No info	2019	6	6
1	IndiGo	Kolkata	Banglore	06:20	10:20	1 stop	No info	2019	12	5
2	Jet Airways	Delhi	Cochin	19:15	19:00 22 May	1 stop	In-flight meal not included	2019	5	21
3	Multiple carriers	Delhi	Cochin	08:00	21:00	1 stop	No info	2019	5	21
4	Air Asia	Banglore	Delhi	23:55	02:45 25 Jun	non-stop	No info	2019	6	24

Using the date_bifurcation function we are able separate the date information and dropping all the unnecessary columns as well.

```
df_test = time_bifurcation(df_test)
print(f"Rows and Columns:", df_test.shape)
df_test.head()
```

```
Rows and Columns: (2645, 12)
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Year_of_Journey	Month_of_Journey	Day_of_Journey	Departure_hour	Departure_minute	Arr
0	Jet Airways	Delhi	Cochin	1 stop	No info	2019	6	6	17	30	4
1	IndiGo	Kolkata	Banglore	1 stop	No info	2019	12	5	6	20	10
2	Jet Airways	Delhi	Cochin	1 stop	In-flight meal not included	2019	5	21	19	15	19
3	Multiple carriers	Delhi	Cochin	1 stop	No info	2019	5	21	8	0	21
4	Air Asia	Banglore	Delhi	non-stop	No info	2019	6	24	23	55	2

Using the time_bifurcation function we are separating time related values and creating additional columns from them.

```
predited_values = Final_Model.predict(df_test)
data = pd.DataFrame(predited_values, columns=['Predicted Flight Prices'])
data
```

	Predicted Flight Prices
0	14110.669922
1	4543.466309
2	12488.573242
3	10740.701172
4	3604.916504
...	...
2640	10002.468750
2641	4299.227539
2642	16538.347656
2643	12429.273438
2644	7668.941895

2645 rows × 1 columns

Here we are using our best model algorithm to predict the target label and show it in a data frame.

Hence, at the end, we were successfully able to train our regression model ‘XGB Regressor’ to predict the flights of prices with an r2_score of 88%, and have achieved the required task successfully.