Flight Price Prediction Project

Submitted by:

Nimit Shah

ACKNOWLEDGMENT

I have taken efforts in this project however it would not have been completed without guidance from other people. I warmly acknowledge the invaluable supervision and an inspired guidance by our SME Mr. Keshav Bansal, FlipRobo Technology.

I would also like to express my sincere thanks to Data trained Education and FlipRobo Technology for giving me an opportunity to work on this project.

I also want to express my gratitude towards my friends and family who have patiently extended all sorts of help for accomplishing this.

I am grateful to one and all who are directly or indirectly involved in successful completion of this project.

INTRODUCTION

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 -
1. Time of purchase patterns (making sure last-minute purchases are expensive)
2. 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)

The project consists of two phase :

1. **Data Collecting Phase**
2. **Model Building Phase**

In first phase we have to collect data of flights ticket from online websites. Here data is collected from “www.yatra.com” website using Selenium technique for web scraping. We have fetched data of popular flights available on website and a few flights from metropolitan cities.

Next, we have built a regression model to predict flight ticket prices. In the long term, this would allow people to better explain and review their purchase with each other in this increasingly digital world.

ANALYTICAL FRAMING

In our scrapped dataset, our target variable "**Price**" is a continuous variable. Therefore, we will be handling this modeling problem as classification.

This project is done in three parts:

- Data Collection phase
- Data Analysis Phase
- Model Building Phase

1. Data Collection Phase

You have to scrape at least 1500 rows of data. You can scrape more data as well, it's up to you, More the data better the model
In this section you have to scrape the data of flights from different websites (yatra.com, skyscanner.com, official websites of airlines, etc). The number of columns for data doesn't have limit, it's up to you and your creativity. Generally, these columns are airline name, date of journey, source, destination, route, departure time, arrival time, duration, total stops and the target variable price. You can make changes to it, you can add or you can remove some columns, it completely depends on the website from which you are fetching the data.

2. Data Analysis Phase

After cleaning the data, you have to do some analysis on the data.
Do airfares change frequently? Do they move in small increments or in large jumps? Do they tend to go up or down over time?
What is the best time to buy so that the consumer can save the most by taking the least risk?
Does price increase as we get near to departure date? Is Indigo cheaper than Jet Airways? Are morning flights expensive?

3. Model Building Phase

After collecting the data, you need to build a machine learning model. Before model building do all data pre-processing steps. Try different models with different hyper parameters and select the best model.
Follow the complete life cycle of data science. Include all the steps like

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

DATA SOURCES AND THEIR FORMATS

1. We collected the data from “yatra.com”. The data is scrapped using Web scraping technique and the framework used is Selenium.

2. We scrapped 1875 of the data and saved data in a data frame

3. In the end, we saved all the data in csv file as “data.csv”

4. The data fetched looks like what is shown below:

	Unnamed: 0	Name	Date	Departure	Arrival	Source	Destination	Stops	Duration	Price
0	0	IndiGo	24-Jan	22:40	01:30 + 1 day	New Delhi	Chennai	Non Stop	2h 50m	2,341
1	1	IndiGo	24-Jan	3:00	5:55	New Delhi	Chennai	Non Stop	2h 55m	2,341
2	2	IndiGo	24-Jan	8:35	11:30	New Delhi	Chennai	Non Stop	2h 55m	2,341
3	3	IndiGo	24-Jan	20:15	23:10	New Delhi	Chennai	Non Stop	2h 55m	2,341
4	4	IndiGo	24-Jan	9:50	12:50	New Delhi	Chennai	Non Stop	3h 00m	2,341

Features:

Name: name of Airline

Date: date of journey

Departure: time of departure

Arrival: time of arrival

Source: the source from which service begins

Destination: the destination where service ends

Stops: total number of stops between source and destination

Duration: total duration of flight

Price: Price of flight ticket

The dataset has no null values.

DATA PRE-PROCESSING

- First we will clean price column by removing ',' and changing it's data type to 'int'

```
# Changing data type of price
p=[]
price=df["Price"]
for i in range(len(price)):
    st=price[i]
    p.append(st.replace(",",""))
df["Price"]=p

df_type_dict={'Price':int}
df=df.astype(df_type_dict)
df.dtypes
```

```
Unnamed: 0      int64
Name           object
Date           object
Departure      object
Arrival        object
Source         object
Destination    object
Stops          object
Duration       object
Price           int32
dtype: object
```

- Next we have removed unnecessary columns and cleaned data in "Arrival", "Departure", "Duration" and "Date" and derived new features from each given feature. I have first formed a new data frame and then done all the processing. The following code is used to clean data and deriving new features from it:

```
list=[]
hrs=[]
min=[]
list=time["Departure"]
for i in range(len(list)):
    str=[]
    str=list[i].split(':')
    hrs.append(str[0])  ## Seperating hours and minutes
    min.append(str[1])
time["Dep_time_hours"]=hrs
time["Dep_time_min"]=min
```

```
list=[]
hrs=[]
min=[]
list=time["Duration"]
for i in range(len(list)):
    str=[]
    str=list[i].split(' ')
    if(len(str)>1):
        hrs.append(str[0][:-1])   ## Seperating hours and minutes
        min.append(str[1][:-1])
    else:
        hrs.append(list[i][:-1])
        min.append('0')
time["Duration_hours"]=hrs
time["Duration_min"]=min
```

```
list=[]
hrs=[]
min=[]
list=time["Arrival"]
for i in range(len(list)):
    str=[]
    str=list[i].split(':')
    hrs.append(str[0])  ## Seperating hours and minutes
    min.append(str[1][:3])
time["Arrival_time_hours"]=hrs
time["Arrival_time_min"]=min
```

```
list=[]
d=[]
m=[]
list=time["Date"]
for i in range(len(list)):
    d.append(list[i][:2])  ## Seperating date and month
    m.append(list[i][3:])
time["day"]=d
time["month"]=m
```

```
month=[]
for i in time["month"]:
    if i=="Jan":
        month.append(1)
    elif i=="Feb":
        month.append(2)
    elif i=="Mar":
        month.append(3)
    elif i=="Apr":
        month.append(4)
    elif i=="May":
        month.append(5)
    elif i=="Jun":
        month.append(6)
    elif i=="Jul":
        month.append(7)
    elif i=="Aug":
        month.append(8)
    elif i=="Sep":
        month.append(9)
    elif i=="Oct":
        month.append(10)
    elif i=="Nov":
        month.append(11)
    elif i=="Dec":
        month.append(12)
    else:
        month.append(np.nan)
time["month"]=month
```

- After executing the above lines of code we will get 8 new columns Dep_time_hours, Dep_time_min, Duration_hours, Duration_min, Arrival_time_hours, Arrival_time_min, day and month. Each feature now has integer data type. Since all the usefull information is now extracted we can drop previous columns.

	Departure	Arrival	Duration	Date	Dep_time_hours	Dep_time_min	Duration_hours	Duration_min	Arrival_time_hours	Arrival_time_min	day	month
0	22:40	01:30 + 1 day	2h 50m	24-Jan	22	40	2	50	01	30	24	1
1	3:00	5:55	2h 55m	24-Jan	3	00	2	55	5	55	24	1
2	8:35	11:30	2h 55m	24-Jan	8	35	2	55	11	30	24	1
3	20:15	23:10	2h 55m	24-Jan	20	15	2	55	23	10	24	1
4	9:50	12:50	3h 00m	24-Jan	9	50	3	00	12	50	24	1

- Next we have introduced two more columns as “Number_of_days” giving the ticket price number of days before the flight service and “Total_duration” giving total time of service. Following code is used for this step.

```
data["month"].value_counts()
```

```
11    744
1     703
12    428
Name: month, dtype: int64
```

```
for i in data["month"]:
    if i==1:
        data["Number_of_days"]=(data["day"]+31+4+30)
    elif i==12:
        data["Number_of_days"]=(data["day"]+30+4)
    elif i==11:
        data["Number_of_days"]=(data["day"]+4)
    else:
        print("Add another condition")
```

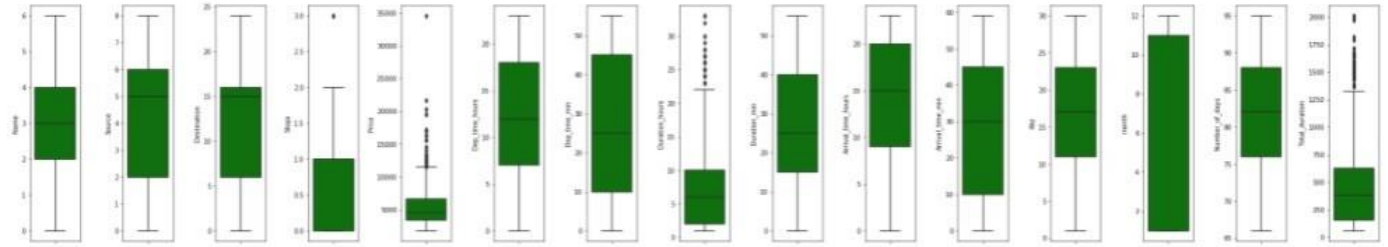
```
#introducing a new column total duration
data["Total_duration"]=data["Duration_hours"]*60+data["Duration_min"]
```

- Encoding variables with object data type: We have encoded “Stops” manually and used LabelEncoder for other variables.

```
# Encoding Total_Stops Column
data["Stops"]=data["Stops"].replace({'Non Stop':0,'1 Stop':1,'2 Stop(s)':2,'3 Stop(s)':3,'4 Stop(s)':4})
data.head()
```

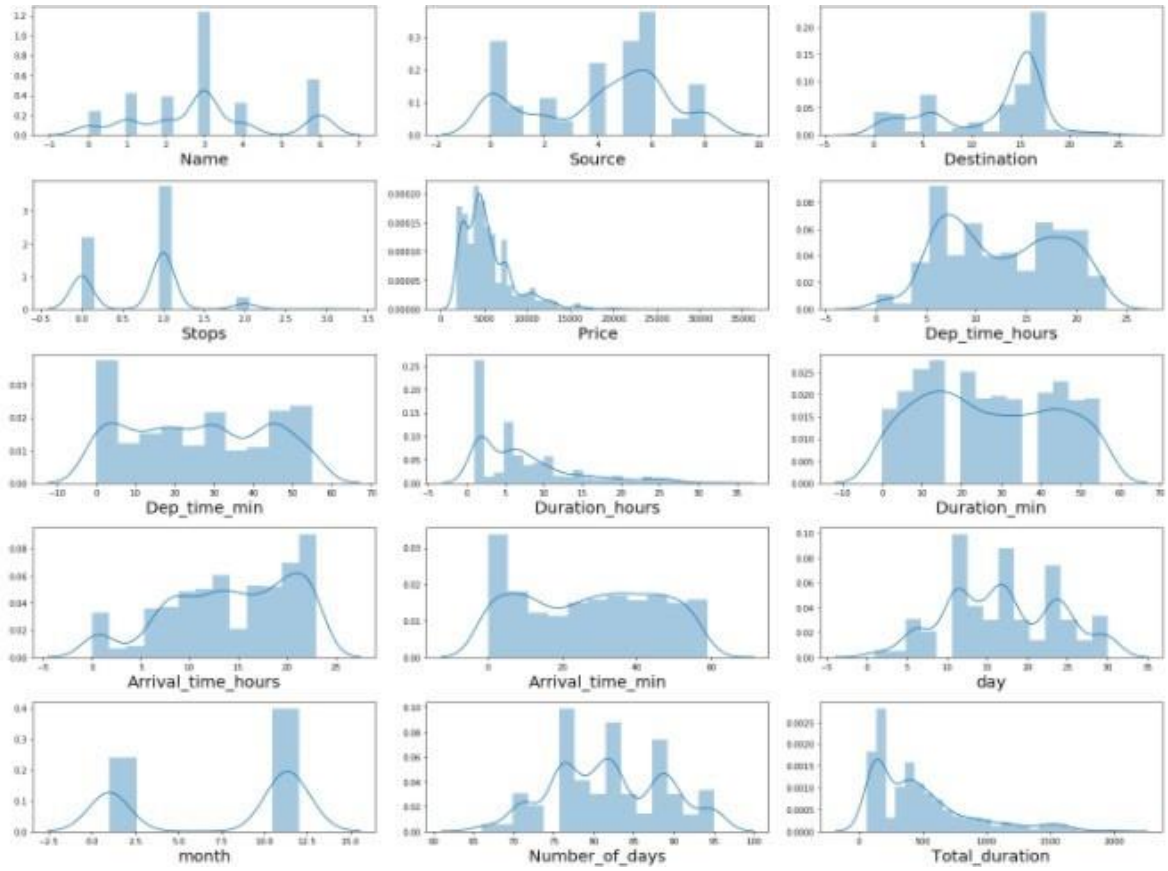
```
lab_enc=LabelEncoder()
cols=["Name", "Source", "Destination"]
for i in cols:
    df1= lab_enc.fit_transform(data[i])
    data[i]=df1
data.head()
```

CHECKING OUTLIERS



We observe outliers in 'Price', 'Duration_hours' and 'Total_duration'.

CHECKING SKEWNESS AND DATA DISTRIBUTION



To handle outliers and skewness we have used z-score method and log transformation by which we faced a data loss of 2.5%.

HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED

For doing this project, the hardware used is a laptop with a stable internet connection. While coming to software part, I have used **Jupyter notebook** to do my python programming and analysis. We also need Google chrome webdriver to scrap data.
Libraries Used:

- scikit-learn
- matplotlib
- pandas
- numpy
- Selenium

PREPARING DATA FOR MODEL

Making our Data ready for model Building phase we will first separate target variable from other features. Then use StandardScaler to scale data and use train_test_split to split data into train and test to make it ready for model.

We found out the best random state for our linear regression model and then run each model on this random state.

```
#Finding the best random state and r2_score
for i in range(100):
    x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=.20,random_state=i)
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    if round(r2_score(y_train,pred_train)*100,1)==round(r2_score(y_test,pred_test)*100,1):
        print('At random state',i,',the model performs well')
        print('Training r2_score is: ',r2_score(y_train,pred_train)*100)
        print('Testing r2_score is: ',r2_score(y_test,pred_test)*100)
```

```
At random state 52 ,the model performs well
Training r2_score is:  58.532365266999285
Testing r2_score is:  58.495003867753695
At random state 54 ,the model performs well
Training r2_score is:  58.5370270597652
Testing r2_score is:  58.476926499476775
At random state 81 ,the model performs well
Training r2_score is:  58.501148001440754
Testing r2_score is:  58.52425650963049
```

MODEL BUILDING AND EVALUATION

Algorithms used are:

- Linear Regression
- Decision Tree Regressor
- KNN Regressor
- Random Forest Regressor
- Gradient Boosting Regressor

Since range of target variable is too high we are getting a high value of mse therefore we will look for R2 score and CV Score to determine our best model.

Linear Regression

```
**** LinearRegression ****

accuracy_score:  0.5852425650963049

cross_val_score:  0.5727340366915723

mean_squared_error  2423438.683034735
```

KNeighboursRegressor

```
**** KNeighborsRegressor ****

accuracy_score:  0.7460475895421008

cross_val_score:  0.7698852848280546

mean_squared_error  1483850.6639344261
```

DecisionTreeRegressor

```
**** DecisionTreeRegressor ****

accuracy_score:  0.7599649123898086

cross_val_score:  0.8201500680455596

mean_squared_error  1402531.3777322404
```

RandomForestRegressor

```
**** RandomForestRegressor ****

accuracy_score:  0.8606367626369499

cross_val_score:  0.8886455523278205

mean_squared_error  814303.0889777505
```

GradientBoostingRegressor

```
**** GradientBoostingRegressor ****

accuracy_score:  0.8261236774436168

cross_val_score:  0.8435510487665692

mean_squared_error  1015963.9603442102
```

Choosing Best Model

After running the loop we get a dataframe showing each model and scores obtained from it.

	Model	Accuracy_score	Cross_val_score	Mean_Squared_Error
0	LinearRegression	58.524257	57.273404	2.423439e+06
1	KNeighborsRegressor	74.604759	76.988528	1.483851e+06
2	DecisionTreeRegressor	75.996491	82.015007	1.402531e+06
3	RandomForestRegressor	86.063676	88.864555	8.143031e+05
4	GradientBoostingRegressor	82.612368	84.355105	1.015964e+06

Looking the various metrics we conclude "**Random Forest Model**" as our best model and hence we will now tune our model.

Hyper-parametric tuning

```
rmf= RandomForestRegressor()
params={'max_features':['auto','sqrt'],'n_estimators':[50,80], 'criterion':['mse','mae'],
       'max_depth':[5,10], 'min_samples_split':[4,6],
       'min_samples_leaf':[2,3]}
grd=GridSearchCV(rmf,param_grid=params)
grd.fit(x_train,y_train)
print('best params=>',grd.best_params_)
```

```
best params=> {'criterion': 'mse', 'max_depth': 10, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 4, 'n_e
stimators': 50}
```

```
rmf= RandomForestRegressor(criterion= 'mse', max_depth= 10, max_features= 'auto', min_samples_leaf= 2,
                           min_samples_split= 4, n_estimators= 50)
rmf.fit(x_train,y_train)
y_pred=rmf.predict(x_test)
print("Random Forest Regression: Accuracy = ",rmf.score(x_test,y_test))
print("\n Mean Squared Error= ",mean_squared_error(y_test,y_pred))
print("\n Root Mean Squared Error= ",np.sqrt(mean_squared_error(y_test,y_pred)))
print("\n Mean Absolute Error= ",mean_absolute_error(y_test,y_pred))
```

```
Random Forest Regression: Accuracy =  0.8453601476367489

 Mean Squared Error=  903564.7552476081

 Root Mean Squared Error=  950.5602323091409

 Mean Absolute Error=  547.8271263504754
```


Our final model gives an accuracy of 84.53%

Comparing original and predicted price of the model

```
#Comparing actual and predicted values with the help of a dataframe
predictions=pd.DataFrame({'Original_price':y_test, 'Predicted_price':y_pred})
predictions
```

	Original_price	Predicted_price
196	2126	2204.848000
1250	7487	7527.094500
1117	3513	2955.645642
1201	5941	5456.679404
1452	7489	7476.393333
...	...	...
205	4002	5024.182102
1163	4754	5470.688316
455	6367	6126.394908
1611	4429	7027.430524
1763	5555	4919.413194

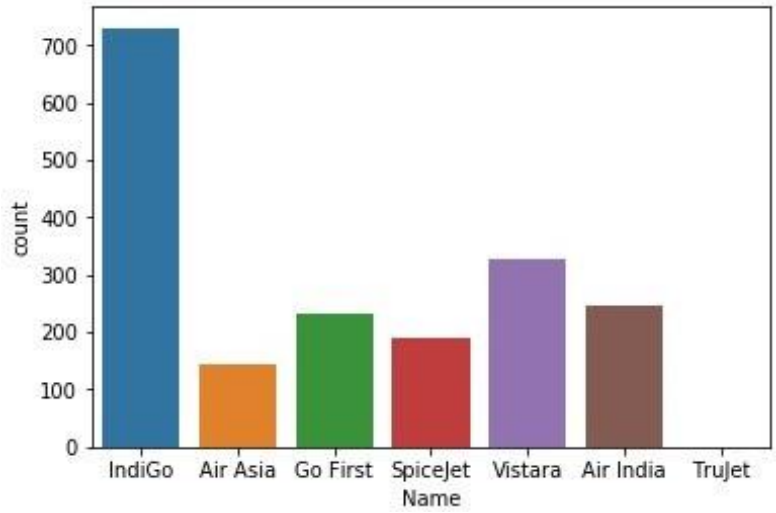
366 rows × 2 columns

Saving the model

```
## savig model
filename= "flightPrice_prediction.pickle"
pickle.dump(rmf, open(filename, 'wb'))
```

EXPLORATORY DATA ANALYSIS

Airline Names

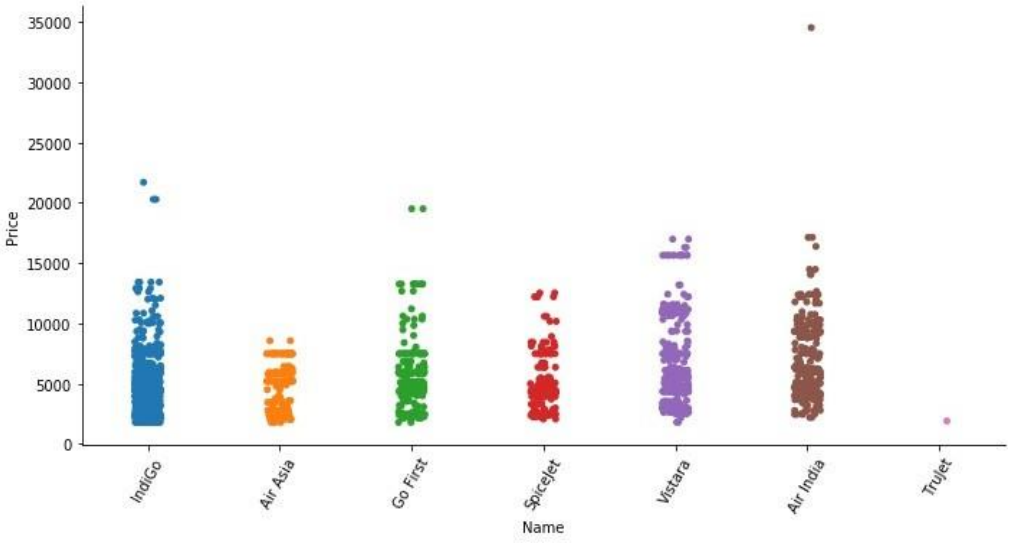


Matrix showing airlines and flights with number of stops

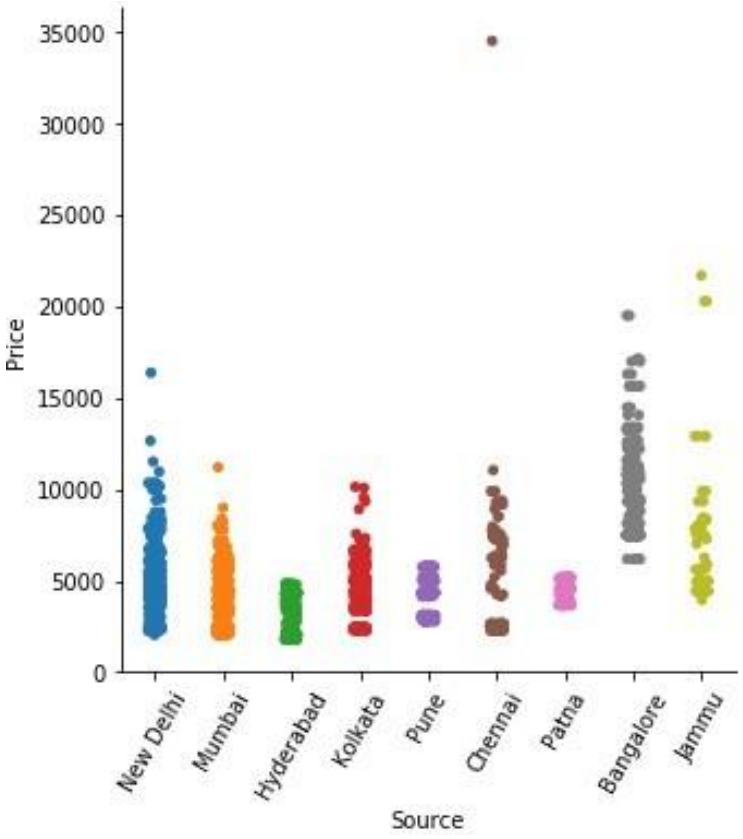
```
#stats of airlines with total stops
pd.crosstab(df['Name'],df['Stops'])
```

Stops	1 Stop	2 Stop(s)	3 Stop(s)	Non Stop
Name				
Air Asia	86	0	0	58
Air India	147	58	1	42
Go First	129	1	0	101
IndiGo	452	15	0	264
SpiceJet	104	2	0	85
TruJet	0	0	0	1
Vistara	192	32	4	101

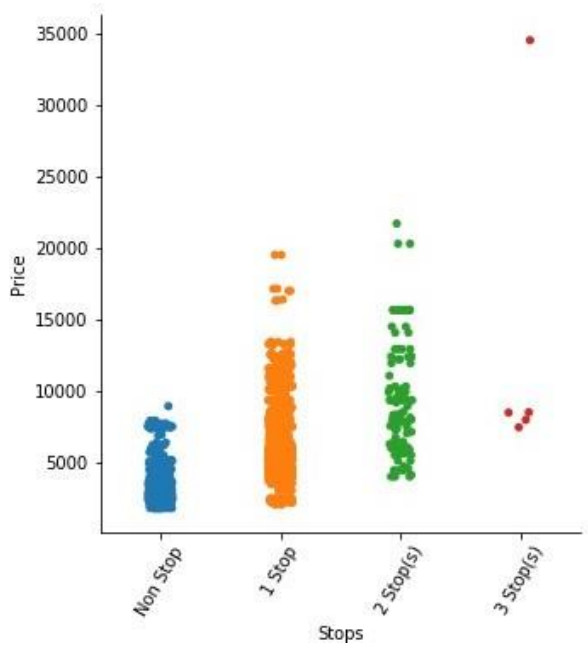
Name v/s Price



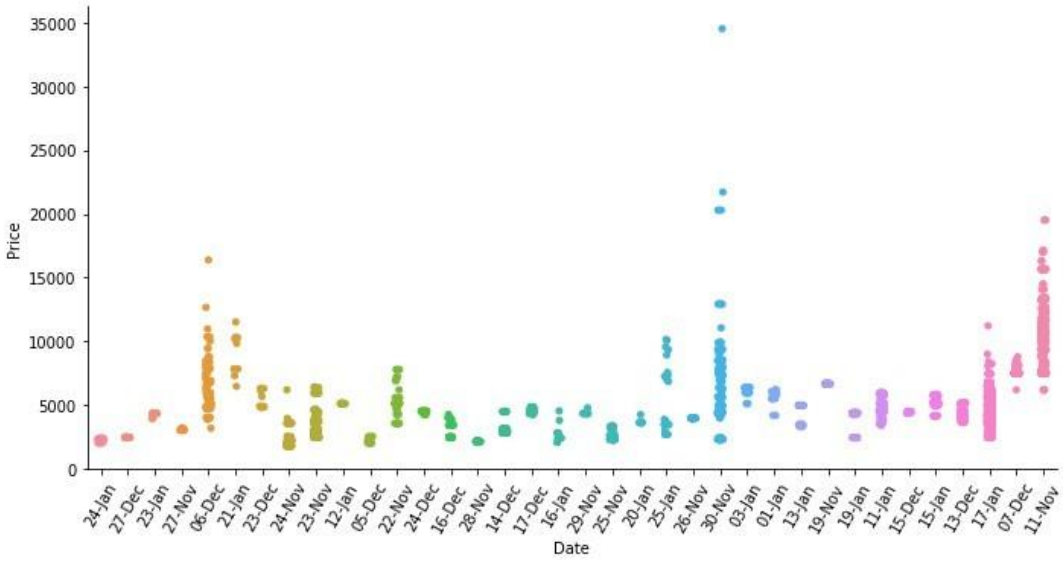
Source v/s Price



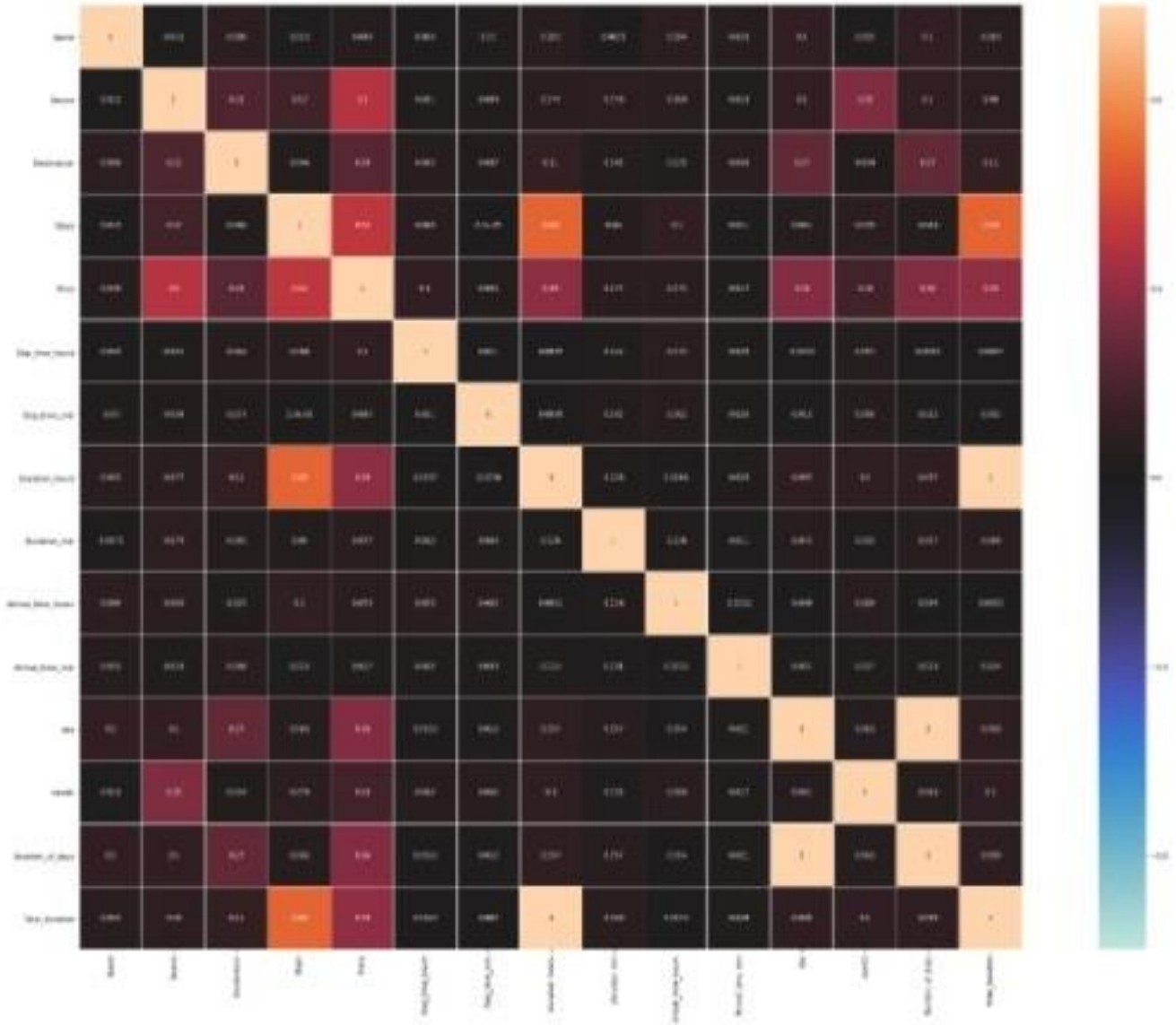
Stops v/s Price



Date v/s Price



Correlation matrix and heatmap



CONCLUSIONS

KEY FINDINGS AND CONCLUSIONS OF THE STUDY

First, we collected flight data from “yatra.com” it was done by using Web scraping. The framework used for web scraping was Selenium, which has an advantage of automating our process of collecting data. . Then the scrapped data was saved in a csv file to use it for modeling purpose.

From the extensive EDA performed in this project we observe Flights from Bangalore and Jammu have higher prices. Flights with longer route i.e. high number of stops have high prices. Also, prices of flight in next month are high as compared to those in coming months. From the given data we can also conclude that AirIndia and vistara flights are expensive as compared to other flights.

The model build after hyper-parametric tuning gives an accuracy for 84.53%.

LEARNING OUTCOMES OF THE STUDY IN RESPECT OF DATA SCIENCE

After the completion of this project, we got an insight of how to collect data, pre-processing the data, analyzing the data and building a model. It helped me to gain conclusions from graphs. Also it helped me in exploring multiple algorithms and metrics to get the best output.

LIMITATIONS OF THIS WORK AND SCOPE FOR FUTURE WORK

Since the data keeps changing we cannot fully rely on this project in the distant future we need to update it with updation in data. Also the scrapping of data took a lot of time as there was no such detail mentioned on fetching data. Random sources and destinations are used to pick up data.
This project is done with limited resources and can be made more efficient in future.