Flight Price Prediction

SUBMITTED BY:
SHRAVANI NATAKALA

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

- 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.

- Our goal is to build a regression model to predict price of flight ticket.

- We have also performed the EDA to gain insights of the data.

Data Set Description

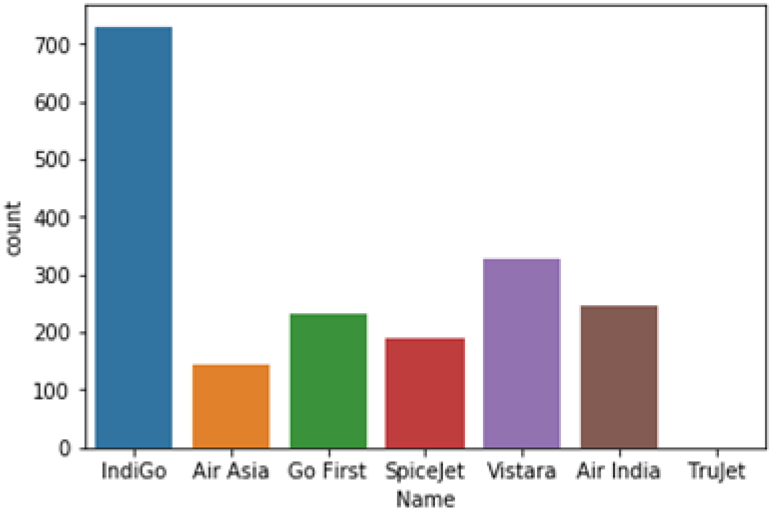
Features:

- **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 Visualizations

Airline Names

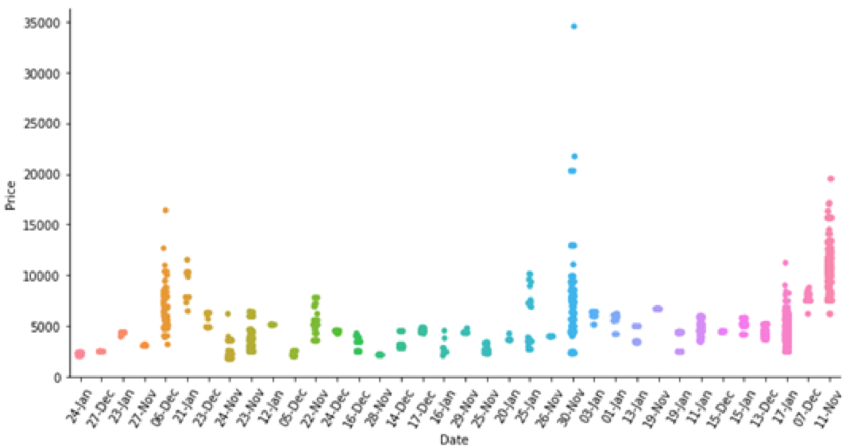


Matrix showing airlines and flights with number of stops

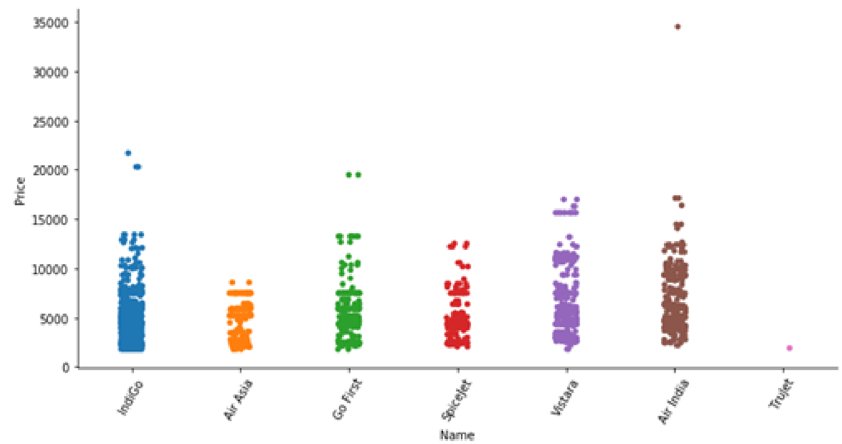
```
#stats of airlines with total stops
pd.crosstab(df['Name'],df['Stops'])
```

Stops Name	1 Stop	2 Stop(s)	3 Stop(s)	Non Stop
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

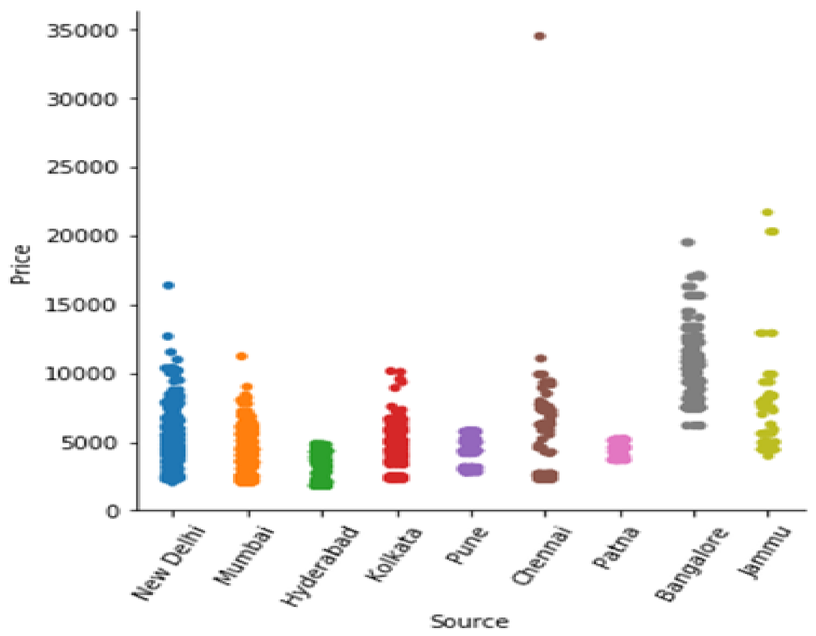
Date v/s Price



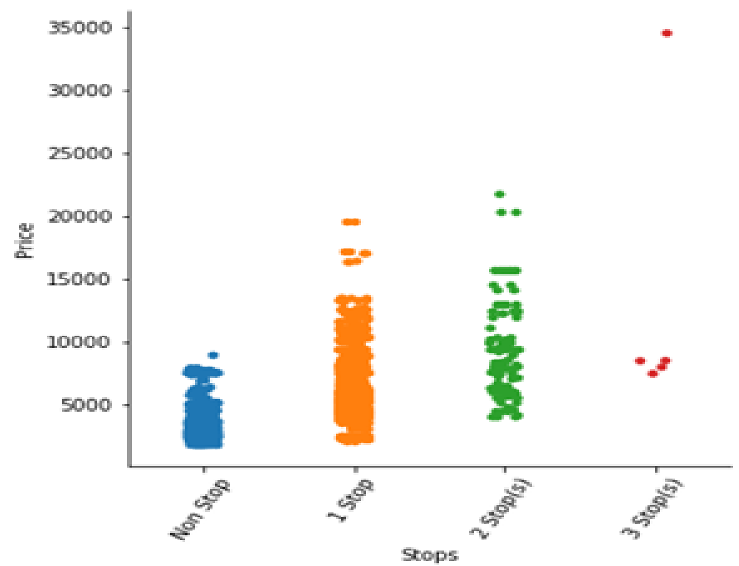
Name v/s Price



Source v/s Price



Stops v/s Price



Data Pre-Processing

```
# 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
```

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

- 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.

```
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
```

- 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.

```
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.
- We also observed outliers and skewness in data for which we used z-score method and log transformation to deal with it. In this process we faces a data loss of 2.5%.

```
# 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()
```

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.

- For train_test_split we found the best random state by running a loop on linear regression and checking for best accuracy.

MODEL BUILDING AND EVALUATION

Algorithms used are:

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

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.

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

	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

Hyper-Parametric Tuning

In [63]:
```
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}
```

In [64]:
```
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
```

Final prediction gives 84.53% accuracy

Conclusion

- First, we collected data on flight ticket prices from "yatra.com", it was done by using Web scraping. The framework used for web scraping was Selenium.

- 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 observed:

a) Flights from Bangalore and Jammu have higher prices.
b) Flights with longer route i.e. high number of stops have high prices.
c) Also, prices of flight in next month are high as compared to those in coming months.
d) 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%

- 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.

- 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..