FLIP ROBO

NAME OF THE PROJECT

Flight Price Prediction

Submitted by:

Ram Kumar

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to SME. Khushboo Garg .

We are highly indebted to Flip Robo technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I thanks and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

Thanks all.

Ram kumar

.

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)

- So, you have to work on a project where you collect data of flight fares with other features and work to make a model to predict fares of flights.

Analytical Problem Framing

Import library and load the dataset:

```
# import Library:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

```
# Loading the train  dataset:
train_df=pd.read_excel('Data_Train.xlsx')
```

```
# showing 5 raws of train data:
train_df.head()
```

	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


```
# Here infromation about full Train data:
train_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
```

```python
# Here Checking the different Airline Routes:
train_df["Route"].unique()
```

```
array(['BLR → DEL', 'CCU → IXR → BBI → BLR', 'DEL → LKO → BOM → COK',
       'CCU → NAG → BLR', 'BLR → NAG → DEL', 'CCU → BLR',
       'BLR → BOM → DEL', 'DEL → BOM → COK', 'DEL → BLR → COK',
       'MAA → CCU', 'CCU → BOM → BLR', 'DEL → AMD → BOM → COK',
       'DEL → PNQ → COK', 'DEL → CCU → BOM → COK', 'BLR → COK → DEL',
       'DEL → IDR → BOM → COK', 'DEL → LKO → COK',
       'CCU → GAU → DEL → BLR', 'DEL → NAG → BOM → COK',
       'CCU → MAA → BLR', 'DEL → HYD → COK', 'CCU → HYD → BLR',
       'DEL → COK', 'CCU → DEL → BLR', 'BLR → BOM → AMD → DEL',
       'BOM → DEL → HYD', 'DEL → MAA → COK', 'BOM → HYD',
       'DEL → BHO → BOM → COK', 'DEL → JAI → BOM → COK',
       'DEL → ATQ → BOM → COK', 'DEL → JDH → BOM → COK',
       'CCU → BBI → BOM → BLR', 'BLR → MAA → DEL',
       'DEL → GOI → BOM → COK', 'DEL → BDQ → BOM → COK',
       'CCU → JAI → BOM → BLR', 'CCU → BBI → BLR', 'BLR → HYD → DEL',
       'DEL → TRV → COK', 'CCU → IXR → DEL → BLR',
       'DEL → IXU → BOM → COK', 'CCU → IXB → BLR',
       'BLR → BOM → JDH → DEL', 'DEL → UDR → BOM → COK',
       'DEL → HYD → MAA → COK', 'CCU → BOM → COK → BLR',
       'BLR → CCU → DEL', 'CCU → BOM → GOI → BLR',
       'DEL → RPR → NAG → BOM → COK', 'DEL → HYD → BOM → COK',
       'CCU → DEL → AMD → BLR', 'CCU → PNQ → BLR',
       'BLR → CCU → GAU → DEL', 'CCU → DEL → COK → BLR',
       'BLR → PNQ → DEL', 'BOM → JDH → DEL → HYD',
       'BLR → BOM → BHO → DEL', 'DEL → AMD → COK', 'BLR → LKO → DEL',
       'CCU → GAU → BLR', 'BOM → GOI → HYD', 'CCU → BOM → AMD → BLR',
       'CCU → BBI → IXR → DEL → BLR', 'DEL → DED → BOM → COK',
       'DEL → MAA → BOM → COK', 'BLR → AMD → DEL', 'BLR → VGA → DEL',
       'CCU → JAI → DEL → BLR', 'CCU → AMD → BLR',
       'CCU → VNS → DEL → BLR', 'BLR → BOM → IDR → DEL',
       'BLR → BBI → DEL', 'BLR → GOI → DEL', 'BOM → AMD → ISK → HYD',
       'BOM → DED → DEL → HYD', 'DEL → IXC → BOM → COK',
       'CCU → PAT → BLR', 'BLR → CCU → BBI → DEL',
       'CCU → BBI → HYD → BLR', 'BLR → BOM → NAG → DEL',
       'BLR → CCU → BBI → HYD → DEL', 'BLR → GAU → DEL',
       'BOM → BHO → DEL → HYD', 'BOM → JLR → HYD',
       'BLR → HYD → VGA → DEL', 'CCU → KNU → BLR',
       'CCU → BOM → PNQ → BLR', 'DEL → BBI → COK',
       'BLR → VGA → HYD → DEL', 'BOM → JDH → JAI → DEL → HYD',
       'DEL → GWL → IDR → BOM → COK', 'CCU → RPR → HYD → BLR',
```

- Display all column name of dataset.

```python
# checking missing values of train data:
train_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
```

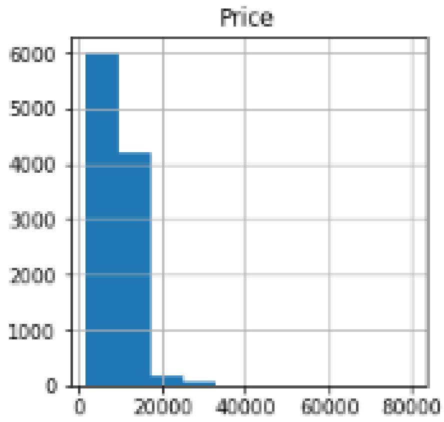
- Display statistical summary.

```
# Hrer describe train data:
train_df.describe()
```

	Price
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000
50%	8372.000000
75%	12373.000000
max	79512.000000

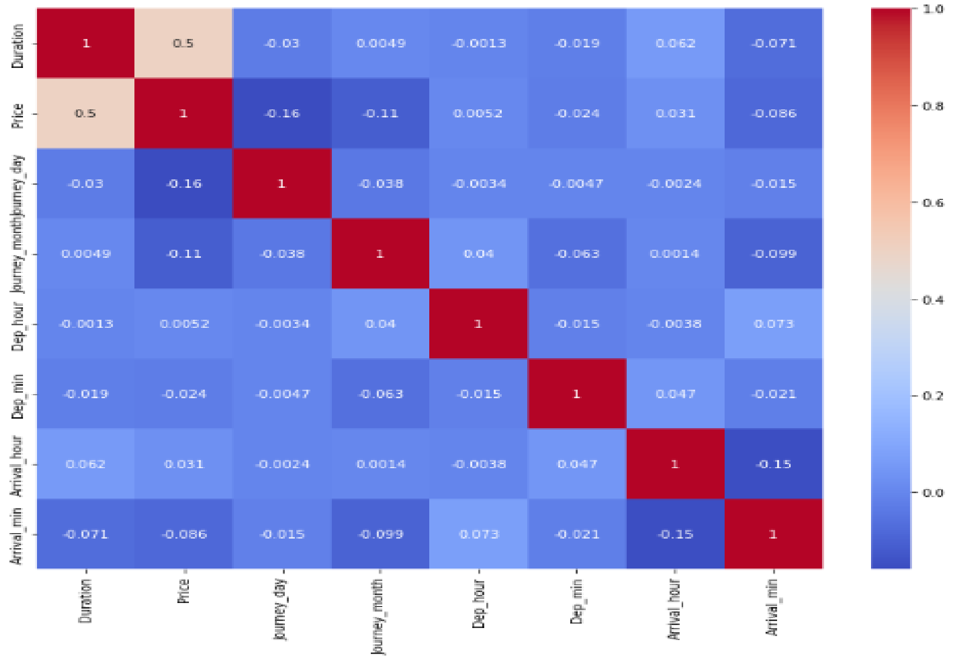
- Display histplot of all columns.

```
# display histogram:
train_df.hist(figsize=(12,12), layout=(3,3), sharex=False);
```



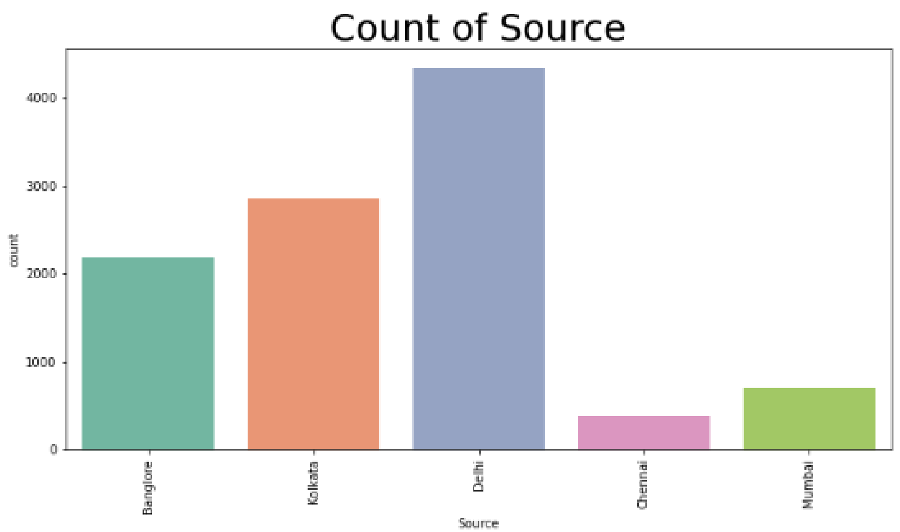
- Display correlation of columns using heatmap.

```
plt.figure(figsize = (12,10))
sns.heatmap(train_df.corr(), annot = True, cmap = "coolwarm")
plt.show()
```



- Display countplot of all columns.

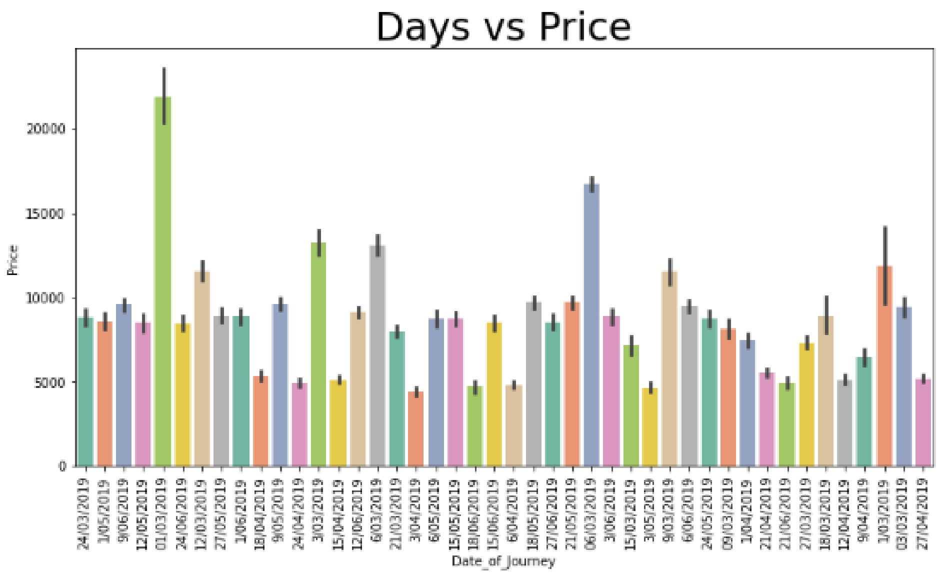
```
# Here ploting Count of Source :
plt.figure(figsize=(12,6))
sns.countplot(train_df['Source'], palette='Set2')
plt.title('Count of Source', size=30)
plt.xticks(rotation=90)
plt.show()
```



-

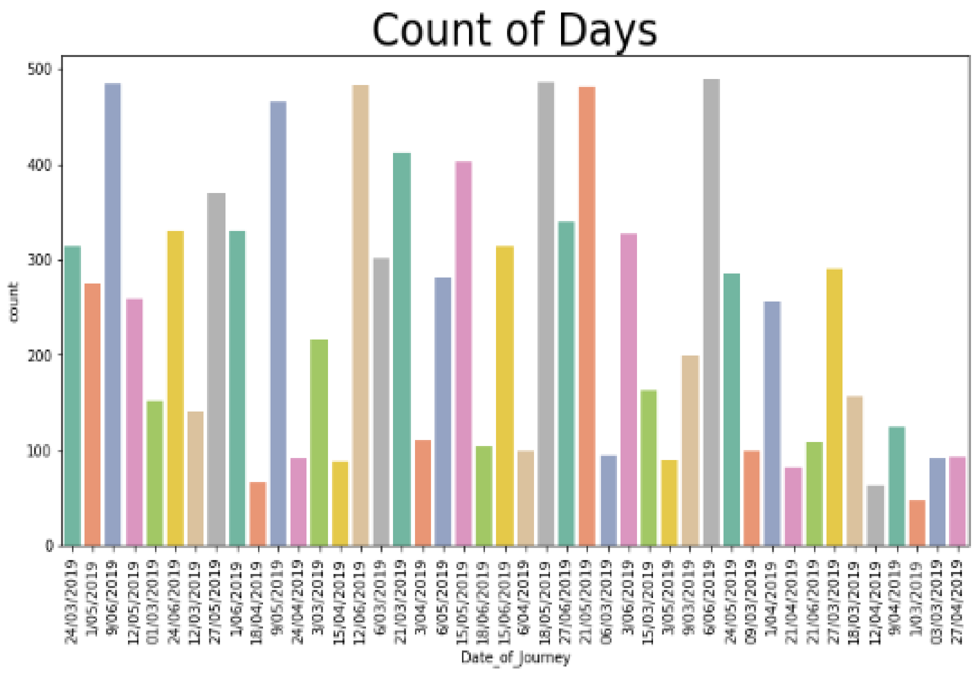
- Display barplot of all columns

```
# Here Plotting days vs price plot:
plt.figure(figsize=(12,6))
sns.barplot(train_df['Date_of_Journey'], train_df['Price'], palette='Set2')
plt.title('Days vs Price', size=30)
plt.xticks(rotation=90)
plt.show()
```



- Display countlot of all columns :

```
# Here ploting Count of Days :
plt.figure(figsize=(12,6))
sns.countplot(train_df['Date_of_Journey'], palette='Set2')
plt.title('Count of Days', size=30)
plt.xticks(rotation=90)
plt.show()
```



Model/s Development and Evaluation

Here Creating Model:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn import metrics
dtr=DecisionTreeRegressor()
dtr.fit(X_train,y_train)
pred=dtr.predict(X_test)
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 761.5920528746615
MSE: 3748900.1907018106
RMSE: 1936.2076827401058
```

```
# RMSE/(max(DV)-min(DV))

1871.8097/(max(y)-min(y))
```

```
0.024073793937211426
```

```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
lr=LinearRegression()
lr.fit(X_train,y_train)
pred=lr.predict(X_test)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, pred))
print('MSE:', metrics.mean_squared_error(y_test, pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
MAE: 2489.462235358398
MSE: 11884879.958170138
RMSE: 3447.4454249734163
```

```
# RMSE/(max(DV)-min(DV))

3447.4454/(max(y)-min(y))
```

```
0.044338422954741295
```

```
from xgboost import XGBRegressor
model =  XGBRegressor()
model.fit(X_train,y_train)
y_pred =  model.predict(X_test)
print('Training Score :',model.score(X_train, y_train))
print('Test Score     :',model.score(X_test, y_test))
```

```
Training Score : 0.9743603714513676
Test Score     : 0.8828956588310257
```

```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
```

```
RandomForestRegressor()
```

```
y_pred = reg_rf.predict(X_test)
reg_rf.score(X_train, y_train)
```

```
0.9791727734708787
```

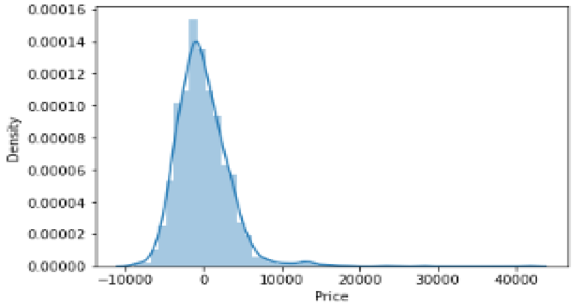
```
reg_rf.score(X_test, y_test)
```

```
0.8882312100025593
```

Testing of Identified Approaches (Algorithms):

```
sns.distplot((y_test-pred),bins=50)
```

```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```




```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 668.1295890667303
MSE: 2330410.161870988
RMSE: 1526.5680993231151
```

```
# RMSE/(max(DV)-min(DV))

1507.4573/(max(y)-min(y))
```

```
0.0193877702468072
```

Run and Evaluate selected models

```
#Cross Validation
from sklearn.model_selection import cross_val_score
for i in range(2,9):
    cv=cross_val_score(reg_rf,X,y,cv=i)
    print(reg_rf,cv.mean())
```

```
RandomForestRegressor() 0.8559751759343641
RandomForestRegressor() 0.8620621467936366
RandomForestRegressor() 0.87510945633491
RandomForestRegressor() 0.8793694518330085
RandomForestRegressor() 0.8813117606791585
RandomForestRegressor() 0.8823107369747097
RandomForestRegressor() 0.8815572505678455
```

```
#SCV
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

parameters = {'max_features': ['auto','sqrt','log2'],
              'criterion':['mse','mae']}
rf = RandomForestRegressor()
clf = GridSearchCV(rf,parameters)
clf.fit(X_train,y_train)

print(clf.best_params_)
```

```
{'criterion': 'mse', 'max_features': 'sqrt'}
```

```
#createing confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
pca= PCA()
pca.fit_transform(X_scaled)
```

```
array([[ 3.75748255,  0.31976053, -0.48242105, ..., -0.18064124,
         0.38869618, -0.17691866],
       [-0.86855725,  0.05798247,  0.08355669, ..., -0.21029531,
```

- **Hypertuning the model：**

```
#Hypertuning the model
from sklearn.model_selection import GridSearchCV
param_grid={'n_estimators':[10,30,50,100],'max_depth':[None,1,2,3],'max_samples':[50,100,250,500,1000],
            'min_samples_split':[2,4,10]}
gcv_reg_rf=GridSearchCV(reg_rf,param_grid,cv=3)
res=gcv_reg_rf.fit(X_train,y_train)
res.best_params_
```

```
{'max_depth': None,
 'max_samples': 1000,
 'min_samples_split': 2,
 'n_estimators': 50}
```

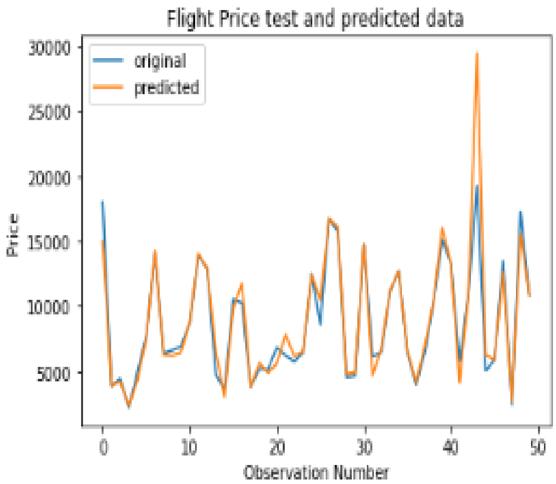
```
y_prediction = reg_rf.predict(X_test)
```

```
metrics.r2_score(y_test, y_prediction)
```

```
0.8882312100025593
```

- Flight Price test and predicted data

```python
number_of_observations=50

x_ax = range(len(y_test[:number_of_observations]))

plt.plot(x_ax, y_test[:number_of_observations], label="original")

plt.plot(x_ax, y_pred[:number_of_observations], label="predicted")

plt.title("Flight Price test and predicted data")

plt.xlabel('Observation Number')

plt.ylabel('Price')

plt.legend()

plt.show()
```



- Hardware and Software Requirements and Tools Used

➢ **Language :-** Python

➢ **Tool:-** Jupyter

➢ **OS:-** Windows 10

➢ **RAM:-** 8gb

CONCLUSION

the machine learning models in the computational intelligence feild that are evaluated before on different datasets are studied. their accuracy and performances are evaluated and compared in order to get better result. For the prediction of the ticket prices perfectly differnt 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 availble seats can improve the performance of the model.