FLIP ROBO

FLIGHT PRICE PREDICTION

Submitted By

Shivanchal Asthana

ACKNOWLEDGMENT

It is my sensual gratification to present this report. Working on this project was an incredible experience that will have a tremendous impact on my career. I would like to express my sincere thanks to the company Flip Robo Technologies for a regular follow up and valuable suggestions provided throughout. They always been an origin of spark and direction. I also thank all the respondents who have given their valuable time, views and valid information for this project.

Shivanchal Asthana

INTRODUCTION

• Business Problem Framing

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travellers saying that flight ticket prices are so unpredictable.

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)

• Conceptual Background of the Domain Problem:

The price of an airline ticket is affected by several factors, such as flight distance, purchasing time, fuel price, etc. Each carrier has its own proprietary rules and algorithms to set the price accordingly.

Flight ticket data is not well organized and ready for direct analysis, collecting and processing those data always requires a great deal of effort. Recent advance in Artificial Intelligence (AI) and Machine Learning (ML) makes it possible to infer such rules and model the price variation. It can also help customers to predict future flight prices and plan their journey accordingly.

• Review of Literature:

The flight ticket buying system is to purchase a ticket many days prior to flight take-off to stay away from the effect of the most extreme charge. Mostly, aviation routes don't agree this procedure. Plane organizations may diminish the cost at the time, they need to build the market and at the time when the tickets are less accessible.

They may maximize the costs. So, the cost may rely upon different factors. To foresee the costs this venture uses AI to exhibit the ways of flight tickets after some time. All organizations have the privilege and opportunity to change its ticket costs at any time. Explorer can set aside cash by booking a ticket at the least costs. People who had travelled by flight frequently are aware of price fluctuations.

The airlines use complex policies of Revenue Management for execution of distinctive evaluating systems. The evaluating system as a result changes the charge depending on time, season, and festive days to change the header or footer on successive pages. The aim of the airways is to earn profit whereas the customer searches for the minimum rate. Customers usually try to buy the ticket well in advance of departure date to avoid hike in airfare as date comes closer. But this is not the fact. The customer may wind up by giving more than they ought to for the same seat.

• Motivation for the Problem Undertaken:

The main objective of doing this project is to build a machine learning model to predict the prices of the house based on the supporting features. Will be predicting the same with the help of

machine learning algorithms. The data is first scrapped by us and then machine learning algorithms to predict the actual price of the flights. To explore various impacts of features on prediction methods, this project will apply machine learning approaches to investigate difference between several models. This project will also validate multiple techniques in model implementation on regression and provide an optimistic result for predicting the flight prices.

Analytical Problem Framing

● Mathematical/ Analytical Modeling of the Problem

We are building a machine learning model to predict the actual value of the flight ticket prices and to decide when to book tickets or not. This model will help us to determine which variables are important to predict the price of variables and also how do these variables describe the price of the flight. With the help of the independent variables, will be able to determine the price of the flight. Regression analysis is a form of predictive modelling technique which investigates the relationship between dependent and independent variables. This technique is used for forecasting, time series modelling, and also find the relationship between the variables. The most common form of regression analysis is linear regression, in which one finds the line that most closely fits the data according to a mathematical criterion.

● Data Sources and their formats:

Data is first scrapped by “Make my trip” Website in the csv format.

1. Dataset contains 3262 rows and 10 columns.

● Data Preprocessing Done:

Data Pre-processing refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for building and training machine learning models. Whenever the data is gathered from various sources, it is collected in raw format. Data pre-processing is considered as an integral part in Machine Learning. It’s extremely important to pre-process the data before feeding it to our model. Hence, it is considered to be

the first and crucial step while creating a machine learning model. Data pre-processing steps used in this project are as below:

1. Loading the dataset
2. Checked the number of rows and columns in the dataset.
3. Checked the null values in the dataset
4. Checked the zero values and duplicate values
5. Check the outliers and remove them by using IQR Method.
6. Dropped the unrequired columns from the dataset.
7. Checked the unique values.
8. Converted all the data to numeric data type using Label Encoding technique.

- Data Inputs- Logic- Output Relationships:

Data inputs are the specifications of the flights like Airline's name, arrival time, departure time. Source, destination etc. and Output is the price of the flight.

- State the set of assumptions (if any) related to the problem under consideration:

  The assumption part for me was relying strictly on the scrapped data and I assume that dataset gathered from website is complete random sample which is representative of the population.

- Hardware and Software Requirements and Tools Used:

Hardware is used by me that is i5 intel core, 8GB RAM, 64Bit processor, and software is Jupyter Notebook for coding along with MS Word to make useful report, MS – PowerPoint to make presentation. For coding, we need to install NumPy, Pandas, Sklearn libraries.

Model/s Development and Evaluation

- **Identification of possible problem-solving approaches (methods):**

The data is first scrapped by using web scarping selenium and scrapped by using “make my trip” website and this dataset contains 3262 rows and 10 columns. There are no null values, no zero values and no duplicate values are found in these datasets. In label, or we can say Price column, we found outliers, we treat them by using IQR method and after cleaning the dataset. We see there are many categorical columns, we use label encoder then to convert all categorical values into numerical form.

After that, we try to remove multicollinearity problem by using Heatmap and Variance inflation method. By using standard scaling, we scale the data. Then we use almost 5-6 models to check the accuracy and select the model which is giving us best accuracy and then we perform Hyperparameter tuning to improve the accuracy and we save our best model.

- **Testing of Identified Approaches (Algorithms):**

• Linear regression’
•Decision Tree regressor
•kNeighbors regressor
•Support Vector Regressor
•Random forest Regressor
•Lasso regressor
•Ridge Regressor
•AdaBoost Regressor
•XGBoost Regressor
•ExtremeGradientBoost Regressor

- **Run and Evaluate selected models**

We check the accuracy by using 5 models

```
#Model instantiating and training
lr = LinearRegression()
dtr=DecisionTreeRegressor()
rfo = RandomForestRegressor()
svr = SVR()
knn = KNeighborsRegressor()
```

```
#train all the models
print('Linear Regression:----->',lr.fit(x_train, y_train))
time.sleep(2)
print('Decsion Tree:------->',dtr.fit(x_train, y_train))
time.sleep(2)
print('Random Forest:-------->',rfo.fit(x_train, y_train))
time.sleep(2)
print('Support vector machine:-------->',svr.fit(x_train, y_train))
time.sleep(2)
print('KNeighborsRegressor:-------->',knn.fit(x_train, y_train))
```

```
Linear Regression:-----> LinearRegression()
Decsion Tree:-------> DecisionTreeRegressor()
Random Forest:--------> RandomForestRegressor()
Support vector machine:--------> SVR()
KNeighborsRegressor:--------> KNeighborsRegressor()
```

```
y_pred_lr = lr.predict(x_test)#sa
y_pred_dtr = dtr.predict(x_test)
y_pred_rfo = rfo.predict(x_test)
y_pred_svr = svr.predict(x_test)
y_pred_knn = knn.predict(x_test)
```

```
#Let's check how well model fits the test data
print('Linear regression Score:-------->', lr.score(x_test, y_test))
time.sleep(2)
print('Decision Tree score:------->', dtr.score(x_test,y_test))
time.sleep(2)
print('Random Forest score:------>',rfo.score(x_test, y_test))
time.sleep(2)
print('Support vector machine score:------>',svr.score(x_test, y_test))
time.sleep(2)
print('KNeighborsRegressor score:------>',knn.score(x_test, y_test))
```

```
Linear regression Score:--------> 0.3315107514799117
Decision Tree score:-------> 0.6086166802727256
Random Forest score:------> 0.7259860339540465
Support vector machine score:------> 0.022144746853470565
KNeighborsRegressor score:------> 0.46290184196277717
```

CV Score:

```
#Check the mean of all models cv score one by one
print("Linear regression CV Score:")
print(cross_val_score(lr,x_scaled,y,cv=5).mean())

time.sleep(2)
print("\n",'*'*50)

print("Decision Tree CV Score:")
print(cross_val_score(dtr,x_scaled,y,cv=5).mean())

time.sleep(2)
print("\n",'*'*50)

print("Random Forest CV Score:")
print(cross_val_score(rfo,x_scaled,y,cv=5).mean())

time.sleep(2)
print("\n",'*'*50)

print("KNeighbour Classifier CV Score:")
print(cross_val_score(knn,x_scaled,y,cv=5).mean())

time.sleep(2)
print("\n",'*'*50)

print("Support Vector Machine CV Score:")
print(cross_val_score(svr,x_scaled,y,cv=5).mean())
```

```
Linear regression CV Score:
0.3708003145068156

 **********
Decision Tree CV Score:
0.561512032221587

 **********
Random Forest CV Score:
0.7542653467553254

 **********
KNeighbour Classifier CV Score:
0.5117131567070897

 **********
Support Vector Machine CV Score:
0.026144396420736606
```

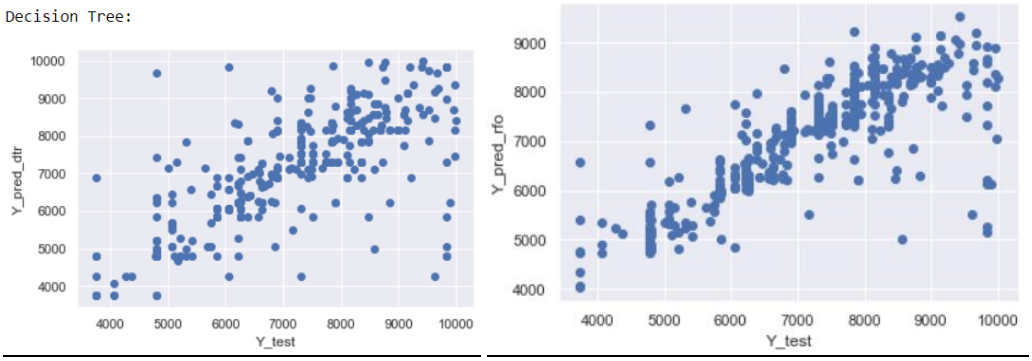
MAE/MSE/RSME:

```
#calculate Mean absolute error

print("Decision Tree:")
print('MAE:--------->',mean_absolute_error(y_test, y_pred_dtr))#calculate Mean absolute error
time.sleep(1)
print('MSE:---------->',mean_squared_error(y_test,y_pred_dtr))#calculate mean squared error
time.sleep(1)
print('RSME:--------->',np.sqrt(mean_squared_error(y_test,y_pred_dtr)))#calculate root mean square error

time.sleep(2)

print("Random Forest:")
print('MAE:--------->',mean_absolute_error(y_test, y_pred_rfo))#calculate Mean absolute error
time.sleep(1)
print('MSE:---------->',mean_squared_error(y_test,y_pred_rfo))#calculate mean squared error
time.sleep(1)
print('RSME:--------->',np.sqrt(mean_squared_error(y_test,y_pred_rfo)))#calculate root mean square error
```

```
Decision Tree:
MAE:---------> 320.148244473342
MSE:----------> 567699.9947984396
RSME:---------> 753.4586881829949
Random Forest:
MAE:---------> 300.99395557000435
MSE:----------> 397456.20024732815
RSME:---------> 630.4412742257031
```

Plot of Actual and Predicted Value:

Decision Tree:



LASSO Regression:

```
#import Ridge, Lasso, RidgeCV, LassoCV
from sklearn.linear_model import Ridge, Lasso, RidgeCV, LassoCV
```

```
#to find best alpha
lassocv= LassoCV(alphas=None, max_iter = 1000, normalize = True)

#train the model
print(lassocv.fit(x_train,y_train))
print()
time.sleep(1)
print('*'*50)
time.sleep(1)
#best alpha parameter
alpha = lassocv.alpha_
print("Alpha value is:----->",alpha)
time.sleep(1)
print()
lasso_reg = Lasso(alpha) #fit the data
time.sleep(1)
print('*'*50)
time.sleep(1)
print(lasso_reg.fit(x_train,y_train)) #train the data
time.sleep(1)
print()
print('*'*50)
time.sleep(1)
print("Accuracy of the model is:---->", lasso_reg.score(x_test,y_test))
```

```
LassoCV(normalize=True)

**************************************************
Alpha value is:-----> 0.013191142641497912

**************************************************
Lasso(alpha=0.013191142641497912)

**************************************************
Accuracy of the model is:----> 0.3315114297721996
```

Ridge Regression:

```
ridgecv= RidgeCV(alphas=np.arange(0.001,0.1,0.01), normalize = True)
#train the data
print(ridgecv.fit(x_train,y_train))
print()
time.sleep(1)
print('*'*50)
time.sleep(1)
#best alpha parameter
print("Alpha value is:----->",ridgecv.alpha_)
print()
time.sleep(1)
print('*'*50)
time.sleep(1)
#fit the data
ridge_model = Ridge(alpha = ridgecv.alpha_)
print(ridge_model.fit(x_train,y_train))
print()
time.sleep(1)
print('*'*50)
time.sleep(1)
print("Accuracy of the model is:----->",ridge_model.score(x_test,y_test))
```

```
RidgeCV(alphas=array([0.001, 0.011, 0.021, 0.031, 0.041, 0.051, 0.061, 0.071, 0.081,
       0.091]),
        normalize=True)

**************************************************
Alpha value is:-----> 0.020999999999999998

**************************************************
Ridge(alpha=0.020999999999999998)

**************************************************
Accuracy of the model is:-----> 0.33151086317674117
```

GridSearchCV – RandomForestRegressor as an Estimator

Gridsearchcv by using Random Forest Regressor as a estimator

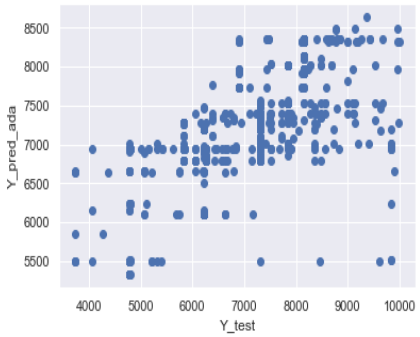
```
params = {'n_estimators':[10,20,30],
          'max_features':['auto',"sqrt","log2"],
          'min_samples_split':[10,11]}
gridsearch = GridSearchCV(estimator=rfo, param_grid= params)#apply GridSearchCV
time.sleep(2)
print(gridsearch.fit(x_train,y_train)) #train the model
time.sleep(1)
print("\n",'*'*50)
time.sleep(1)
print('GridSearchCV best parameters:----->',gridsearch.best_params_) #get best parameters
time.sleep(1)
print("\n",'*'*50)
time.sleep(1)
#put best params
rfc = gridsearch.best_estimator_
print('Trained Model using best parameters:----->',rfc.fit(x_train,y_train)) #train the model
time.sleep(1)
print("\n",'*'*50)
time.sleep(1)
print('Accuracy score:------>',rfc.score(x_train,y_train))#check the accuracy score
```

```
GridSearchCV(estimator=RandomForestRegressor(),
             param_grid={'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [10, 11],
                         'n_estimators': [10, 20, 30]})

 ************************************************
GridSearchCV best parameters:-----> {'max_features': 'sqrt', 'min_samples_split': 11, 'n_estimators': 20}

 ************************************************
Trained Model using best parameters:-----> RandomForestRegressor(max_features='sqrt', min_samples_split=11,
                      n_estimators=20)

 ************************************************
Accuracy score:------> 0.7568975786697465
```

AdaBoostRegressor:

```
ada = AdaBoostRegressor()
time.sleep(2)
print('Adaboost Regressor:-------->',ada.fit(x_train,y_train))
print()
time.sleep(1)
print('*'*50)
time.sleep(1)
print('Adaboost Regressor Training Score:-------->',ada.score(x_train,y_train))
print()
time.sleep(1)
print('*'*50)
time.sleep(1)
print('Adaboost Regressor Testing Score:-------->',ada.score(x_test,y_test))
time.sleep(2)
y_pred_ada = ada.predict(x_test)

time.sleep(2)
print("\n",'*'*50)

print("Ada Boost CV Score:")
print(cross_val_score(ada,x_scaled,y,cv=5))

time.sleep(2)
print("\n",'*'*50)

print("Ada Boost CV Score:")
print(cross_val_score(ada,x_scaled,y,cv=5).mean())

time.sleep(2)
print("\n",'*'*50)
print("Ada Booster:")
plt.scatter(y_test,y_pred_ada)
plt.xlabel('Y_test')
plt.ylabel('Y_pred_ada')
plt.show()
```

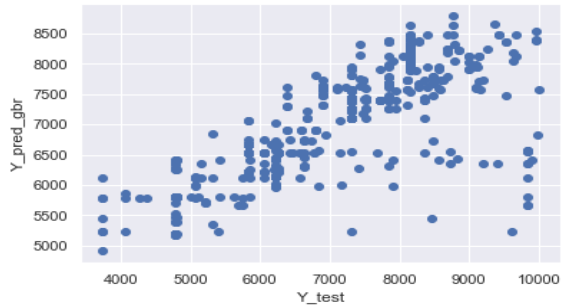
```
Adaboost Regressor:--------> AdaBoostRegressor()

 ************************************************
Adaboost Regressor Training Score:--------> 0.4891225820013917

 ************************************************
Adaboost Regressor Testing Score:--------> 0.40236348892649476

 ************************************************
Ada Boost CV Score:
[0.39933854 0.41381877 0.4055669  0.44522793 0.42911119]

 ************************************************
Ada Boost CV Score:
0.4060560889832595

 ************************************************
Ada Booster:
```



GradientBoostingRegressor:

```
gbr = GradientBoostingRegressor(max_depth=3, n_estimators=6, learning_rate=.4)
time.sleep(1)
print("GradientBoostingRegressor Training Score:----->", gbr.fit(x_train,y_train))
time.sleep(1)
print("*"*50)
time.sleep(1)
y_pred_gbr = gbr.predict(x_test)
print("GradientBoostingRegressor Testing score:------>", r2_score(y_test,y_pred_gbr))
time.sleep(2)
print("\n",'*'*50)

print("GBR CV Score:")
print(cross_val_score(gbr,x_scaled,y,cv=5))

time.sleep(2)
print("\n",'*'*50)

print("GBR CV Score:")
print(cross_val_score(gbr,x_scaled,y,cv=5).mean())

time.sleep(2)
print("\n",'*'*50)
print("GBR:")
plt.scatter(y_test,y_pred_gbr)
plt.xlabel('Y_test')
plt.ylabel('Y_pred_gbr')
plt.show()
```

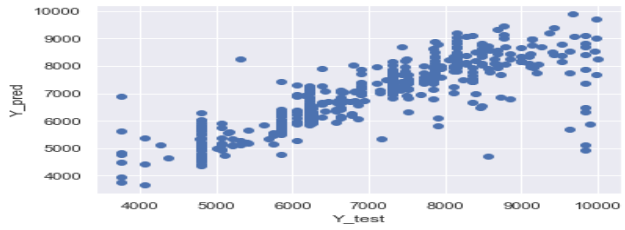
```
GradientBoostingRegressor Training Score:-----> GradientBoostingRegressor(learning_rate=0.4, n_estimators=6)
**************************************************
GradientBoostingRegressor Testing score:------> 0.5706009089178948

 **************************************************
GBR CV Score:
[0.55756646 0.62602796 0.6179395  0.60632713 0.53155697]

 **************************************************
GBR CV Score:
0.5878836028534188

 **************************************************
GBR:
```



XGBoostRegressor:

```
xgbr = xgb.XGBRegressor()
time.sleep(1)
print("XGBoost Training score:------->",xgbr.fit(x_train,y_train))
time.sleep(1)
print('*'*50)
time.sleep(1)
y_pred = xgbr.predict(x_test)
print("XGBoost Testing Score:-------->", r2_score(y_test, y_pred))

time.sleep(2)
print("\n",'*'*50)

print("XGBoost CV Score:")
print(cross_val_score(xgbr,x_scaled,y,cv=5))

time.sleep(2)
print("\n",'*'*50)

print("XGBoost CV Score:")
print(cross_val_score(xgbr,x_scaled,y,cv=5).mean())

time.sleep(2)
print("\n",'*'*50)
print("XGBooster:")
plt.scatter(y_test,y_pred)
plt.xlabel('Y_test')
plt.ylabel('Y_pred')
plt.show()
```

```
XGBoost Training score:------> XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
**************************************************
XGBoost Testing Score:-------> 0.7197832747537083

 **************************************************
XGBoost CV Score:
[0.70585563 0.79978771 0.76872448 0.77711978 0.72257445]

 **************************************************
XGBoost CV Score:
0.754812410448275

 **************************************************
XGBooster:
```



Grid SearchCV on XGBoostRegressor:

```
start_time = time.time()
params = {
    'max_depth':range(4,12,2),
    'learning_rate':np.arange(0.1,1,0.1),
    'min_samples_split': range(4,8,2),
    'nthread':[4],
    'objective':['reg:linear'],
    'learning_rate': [.03, 0.05, .07], #so called `eta` value
    'max_depth': [5, 6, 7],
    'min_child_weight': [4],
    'silent': [1],
    'subsample': [0.7],
    'colsample_bytree': [0.7],
    'n_estimators': [500]}

gridsearch = GridSearchCV(estimator=xgbr, param_grid= params, cv=2,n_jobs = 5,
                          verbose=True )#apply GridSearchCV
time.sleep(2)
print(gridsearch.fit(x_train,y_train)) #train the model
time.sleep(1)
print("\n",'*'*50)
time.sleep(1)
print('GridSearchCV best parameters:----->',gridsearch.best_params_) #get best parameters
time.sleep(1)
print("\n",'*'*50)
time.sleep(1)
#put best params
xgbr_1 = gridsearch.best_estimator_
print('Trained Model using best parameters:----->',xgbr_1.fit(x_train,y_train)) #train the model
time.sleep(1)
print("\n",'*'*50)
time.sleep(1)
print('Accuracy score:------>',xgbr_1.score(x_train,y_train))#check the accuracy score
print("\n",'*'*50)
end_time = time.time()
T = (end_time - start_time)/60
print("Total time taken (in mins):------>", T )
```

```
 ***************************************************
GridSearchCV best parameters:-----> {'colsample_bytree': 0.7, 'learning_rate': 0.03, 'max_depth': 5, 'min_child_weight': 4, 'mi
n_samples_split': 4, 'n_estimators': 500, 'nthread': 4, 'objective': 'reg:linear', 'silent': 1, 'subsample': 0.7}

 ***************************************************
[16:12:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linea
r is now deprecated in favor of reg:squarederror.
[16:12:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:576:
Parameters: { "min_samples_split", "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually being used
  but getting flagged wrongly here. Please open an issue if you find any such cases.


Trained Model using best parameters:-----> XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.03, max_delta_step=0,
             max_depth=5, min_child_weight=4, min_samples_split=4, missing=nan,
             monotone_constraints='()', n_estimators=500, n_jobs=8, nthread=4,
             num_parallel_tree=1, objective='reg:linear', predictor='auto',
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             silent=1, subsample=0.7, tree_method='exact', ...)

 ***************************************************
Accuracy score:------> 0.9048992246858683

 ***************************************************
Total time taken (in mins):------> 0.3121574282646179
```

• Key Metrics for success in solving problem under consideration:

Will go with XGBoost Regressor Reasons

1. Random Forest is also giving us good accuracy by any other model but xGBoost is giving us better accuracy than other.

2. The key metrics used here were R2_Score, MSE, MAE, RMSE and Cross Validation Score. We also tried to find out the best parameters by using Hyper Parameter Tuning and Random Grid Search to increase the accuracy score.

- ## Visualizations:
  - ### All Columns:

```
#check the number of columns
df.columns
```

```
Index(['Unnamed: 0', 'Airline_Name', 'Date-of_Journey', 'Source',
       'Destination', 'Duration', 'Total_Stop', 'Departure_Time',
       'Arrival_Time', 'Price'],
      dtype='object')
```

```
#check the duplicate values
df.duplicated().sum()
```

```
0
```

```
#check the null values
df.isnull().sum()
```

```
Airline_Name        0
Date-of_Journey     0
Source              0
Destination         0
Duration            0
Total_Stop          0
Departure_Time      0
Arrival_Time        0
Price               0
dtype: int64
```

```
#check the zero values
df.all()
```

```
Airline_Name        True
Date-of_Journey     True
Source              True
Destination         True
Duration            True
Total_Stop          True
Departure_Time      True
Arrival_Time        True
Price               True
dtype: bool
```

```
#check the unique value of the column 'Airline_name'
df["Airline_Name"].unique()
```

```
array(['IndiGo', 'Go First', 'Air India', 'SpiceJet', 'AirAsia',
       'Vistara', 'IndiGo, Star Air'], dtype=object)
```

```
#check the unique value of the column 'Source'
df["Source"].unique()
```

```
array(['Mumbai', 'Goa', 'Ahmedabad', 'Kolkata', 'New Delhi'], dtype=object)
```

```
#check the unique value of the column 'Destination'
df["Destination"].unique()
```

```
array(['New Delhi', 'Mumbai', 'Hyderabad', 'Bengaluru', 'Chennai'],
      dtype=object)
```

```
df["Day_of_Journey"] = df["Date_of_Journey"].str.split(" ").str.slice(0,1).str.join('').str.replace(',','')
```

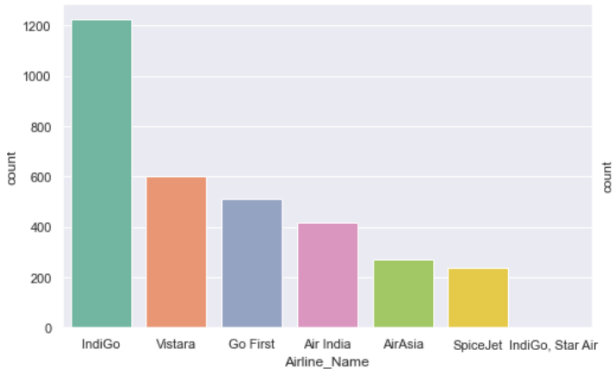
```
df["Month_of_Journey"] = df["Date_of_Journey"].str.split(" ").str.slice(1,2).str.join('').str.replace(',','')
```

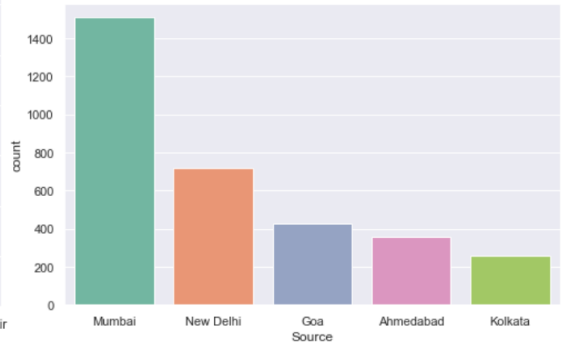
```
df["Date_of_Journey"] = df["Date_of_Journey"].str.split(" ").str.slice(2,).str.join('').str.replace(',','')
```

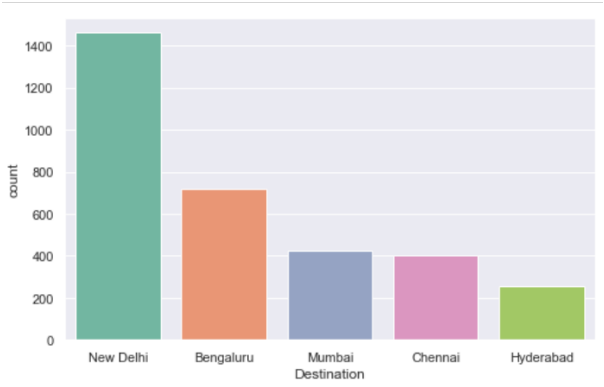
```
df['Arrival_Time'].str.split(':')
time.sleep(2)
df['Arrival_Time_hours'] = df['Arrival_Time'].str[0:2]
time.sleep(2)
df['Arrival_Time_mins'] = df['Arrival_Time'].str[3:]
```

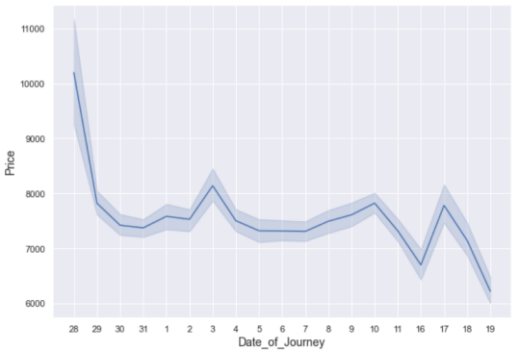
```
df['Duration'].str.split(' ')
time.sleep(2)
df['Duration_hours'] = df['Duration'].str[0:2]
time.sleep(2)
df['Duration_mins'] = df['Duration'].str[5:7]
```

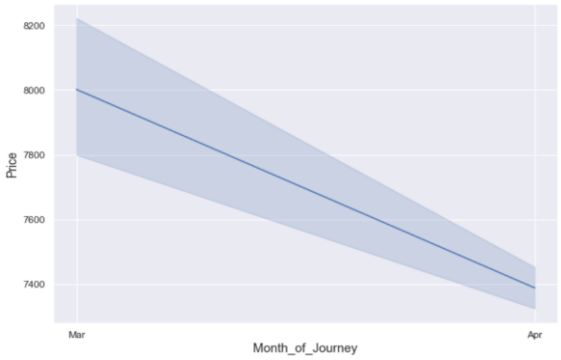
```
df['Departure_Time'].str.split(':')
time.sleep(2)
df['Departure_Time_hours'] = df['Departure_Time'].str[0:2]
time.sleep(2)
df['Departure_Time_mins'] = df['Departure_Time'].str[3:]
```

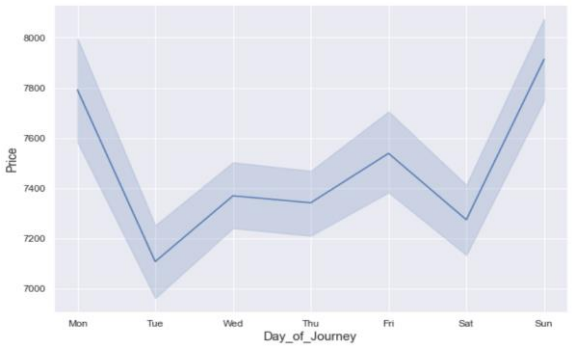


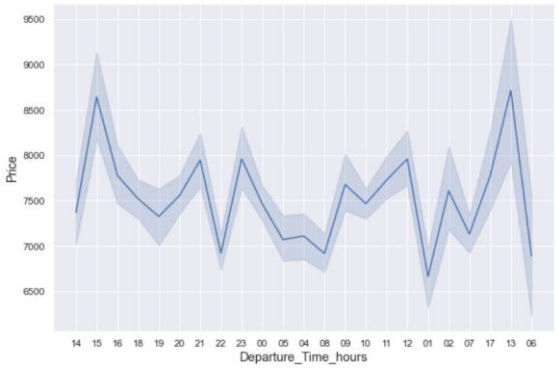


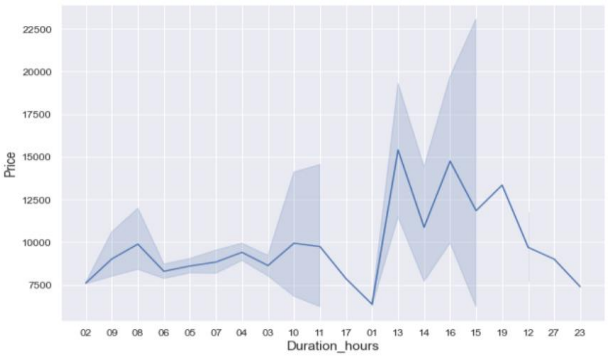


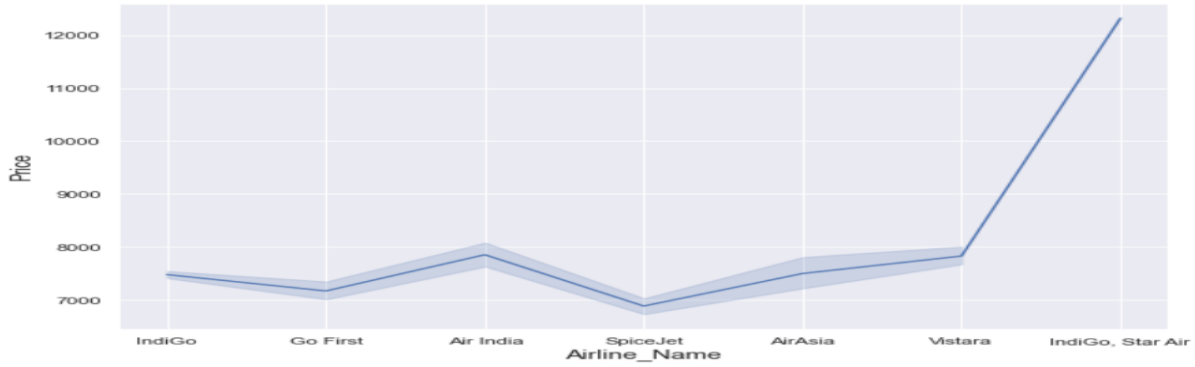












INTERPRETATION

We have trained several models above for the dataset we had prepared, and we got different results for different algorithm. For the selected test data set , output of the model is plotted across the test dataset. Graph shows the comparative study of original values and predicted results. By the analysis of the results obtained from the algorithm such as Linear Regression, Random Forest, Decision Tree, Gradient Boosting, XG Boosting gives the

predicted values of the fare to purchase the flight ticket at the right time.

XGBoost Regressor is giving 90.4% Accuracy.

CONCLUSION

· Key Findings and Conclusions of the Study

To evaluate the conventional algorithm, a dataset is built for various routes in India and studied a trend of price variation for the period of limited days. Machine Learning algorithms are applied on the dataset to predict the dynamic fare of flights. This gives the predicted values of flight fare to get a flight ticket at minimum cost. Data is collected from the websites which sell the flight tickets so only limited information can be accessed. The values of R-squared obtained from the algorithm give the accuracy of the model.

· Learning Outcomes of the Study in respect of Data Science

1. This project has demonstrated the importance of having large dataset for

training and testing the machine learning model.

2. Through data cleaning we were able to remove unnecessary columns and outliers from our dataset due to which our model would have suffered from overfitting or underfitting.

3. Through different powerful tools of visualization, we were able to analyse and interpret different hidden insights about the data.

4. Build Models, since it was a supervised regressiion problem, I built 5 models to evaluate performance of each of them: a. Linear Regression b. Random Forest c. Decision Tree d. XGboost regressor e. Gradient Boosting.

· Limitations of this work and Scope for Future Work

In the future, if more data could be accessed such as the current an availability of seats, the predicted results will be more accurate. One can focus on collection of real time customer-oriented data which can be useful for EDA. And more inference can be provided based on the analysis.