Project Name: End-To-End ML with Deployment Flight Price Prediction (FileUse) – Random Forest – Regression

Table of Contents

Demo

```
#train_data = pd.read_excel(r"E:\MachineLearning\EDA\Flight_Price\Data_Train.xlsx")
train_data = pd.read_excel("Data_Train.xlsx")
```

```
pd.set_option('display.max_columns', None)
```

```
train_data.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

```
train_data.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
```

```
train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%Y").dt.day
```

```
train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
```

```
train_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302	1	3

```
# Time taken by plane to reach destination is called Duration
# It is the differnce betwwen Departure Time and Arrival time


# Assigning and converting Duration column into list
duration = list(train_data["Duration"])

for i in range(len(duration)):
    if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i]           # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration
```

```
# Adding duration_hours and duration_mins list to train_data dataframe

train_data["Duration_hours"] = duration_hours
train_data["Duration_mins"] = duration_mins
```

```
train_data.drop(["Duration"], axis = 1, inplace = True)
```

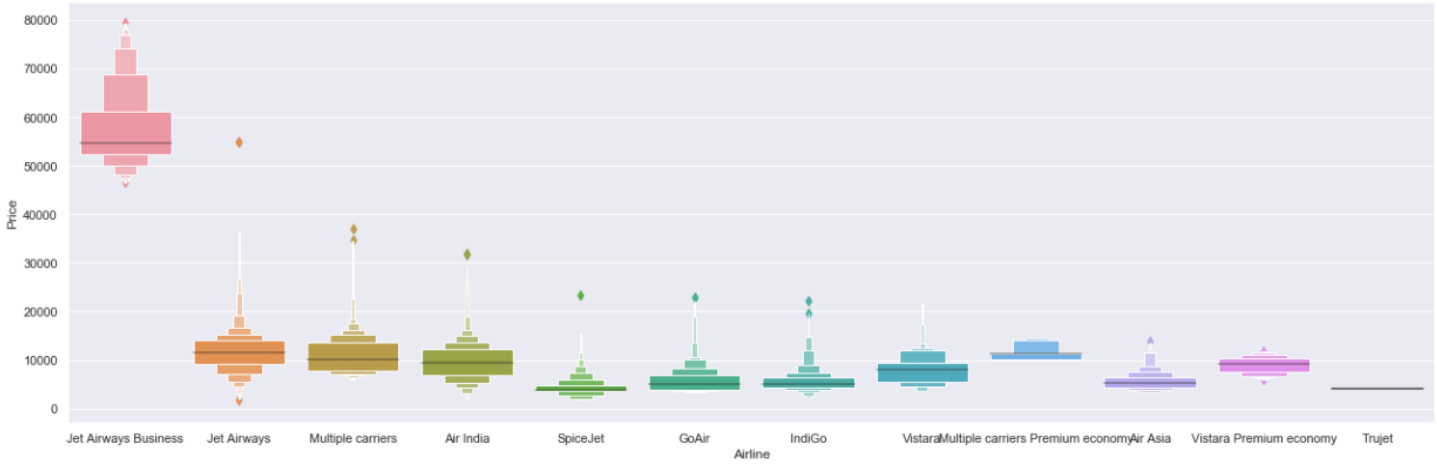
```
train_data.head()
```

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Du
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	22	20	1	10	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5	50	13	15	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	9	25	4	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	12	5	18	5	23	30	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	1	3	16	50	21	35	

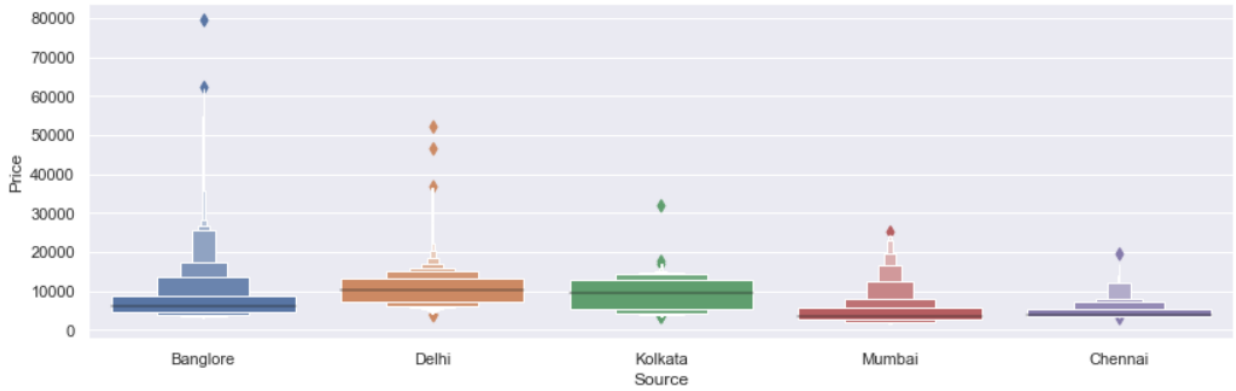
```
train_data["Airline"].value_counts()
```

```
Jet Airways                          3849
IndiGo                               2053
Air India                            1751
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64
```

```
# From graph we can see that Jet Airways Business have the highest Price.
# Apart from the first Airline almost all are having similar median

# Airline vs Price
sns.catplot(y = "Price", x = "Airline", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 6, aspe
plt.show()
```



```
# Source vs Price

sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 4, aspec
plt.show()
```



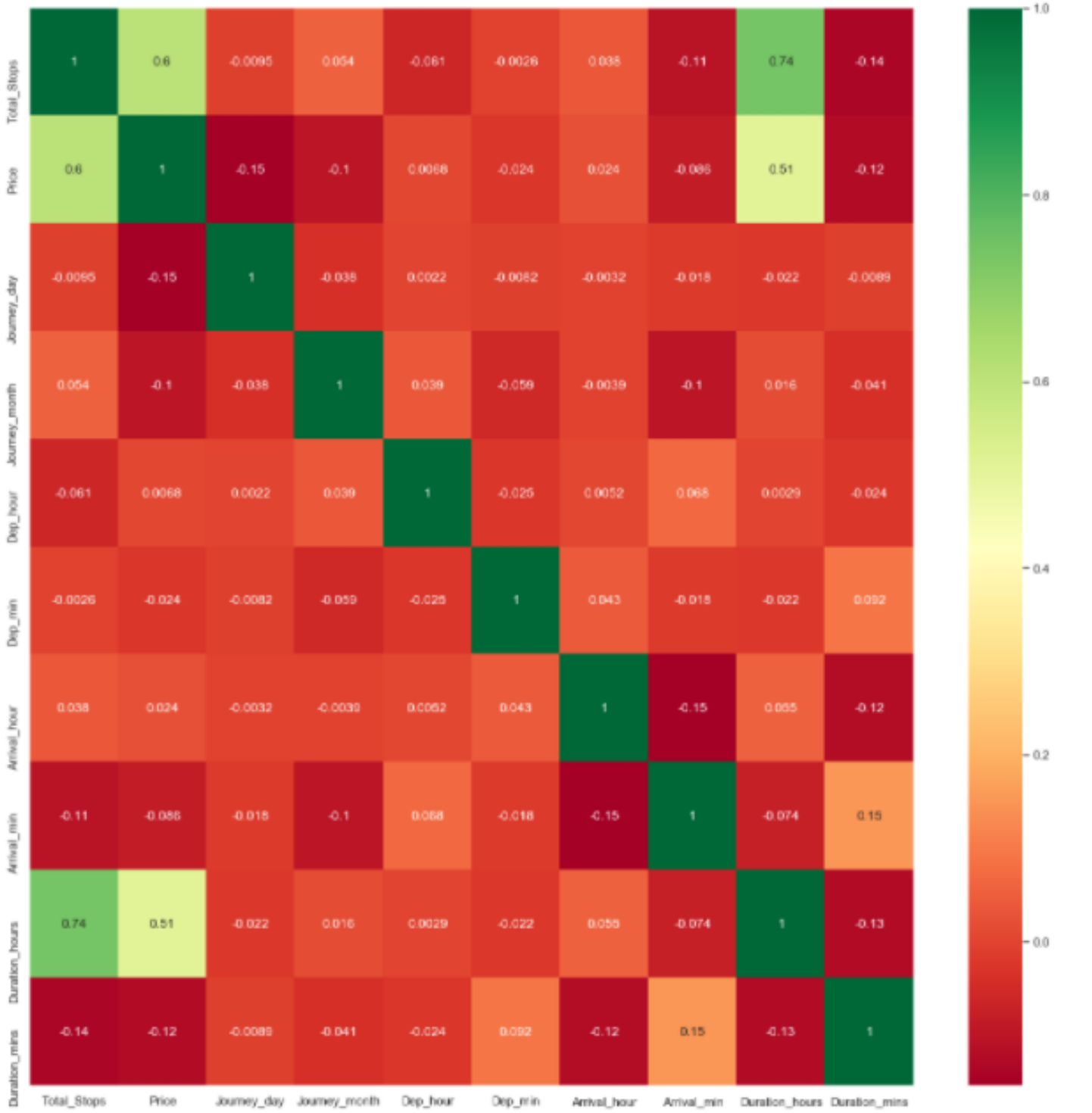

```
# As Destination is Nominal Categorical data we will perform OneHotEncoding

Destination = train_data[["Destination"]]

Destination = pd.get_dummies(Destination, drop_first = True)

Destination.head()
```

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	0	0	0	0	1
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

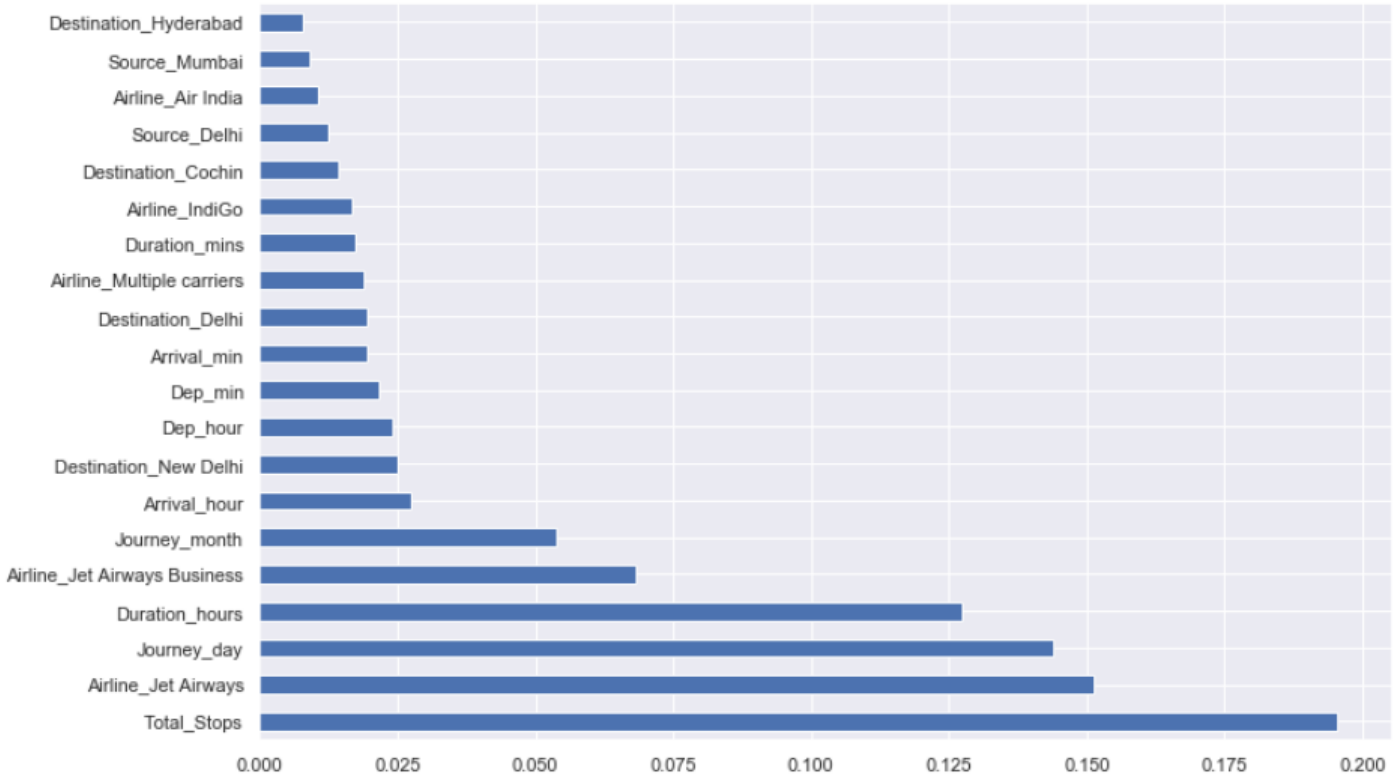
```
train_data["Route"]
```

```
0                  BLR → DEL
1          CCU → IXR → BBI → BLR
2          DEL → LKO → BOM → COK
3                CCU → NAG → BLR
4                BLR → NAG → DEL
                   ...
10678              CCU → BLR
10679              CCU → BLR
10680              BLR → DEL
10681              BLR → DEL
10682      DEL → GOI → BOM → COK
Name: Route, Length: 10682, dtype: object
```

```
# Finds correlation between Independent and dependent attributes

plt.figure(figsize = (18,18))
sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")

plt.show()
```



```
#plot graph of feature importances for better visualization

plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```




```
: from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

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

```
: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        max_samples=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        n_estimators=100, n_jobs=None, oob_score=False,
                        random_state=None, verbose=0, warm_start=False)
```

```
: y_pred = reg_rf.predict(X_test)
```

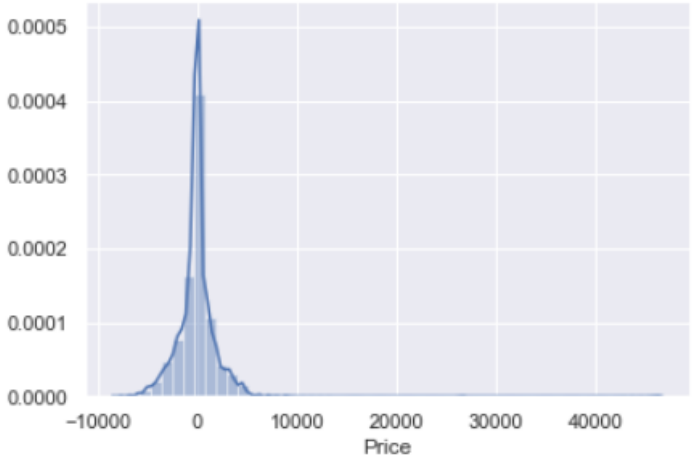
```
: reg_rf.score(X_train, y_train)
```

```
: 0.9539164511170628
```

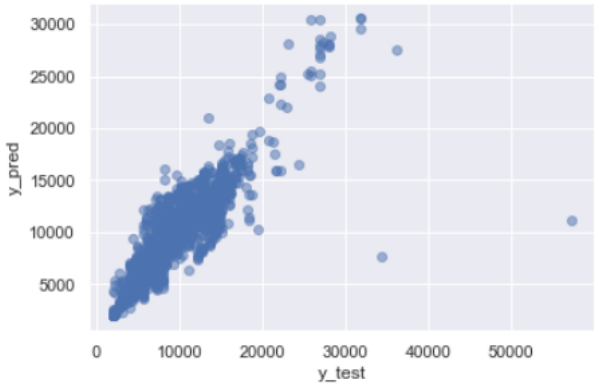
```
: reg_rf.score(X_test, y_test)
```

```
: 0.798383043987616
```

```
sns.distplot(y_test-y_pred)
plt.show()
```




```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```




```
from sklearn import metrics
```

```
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: 1172.5455945373583
MSE: 4347276.1614450775
RMSE: 2085.0122688955757
```

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

2090.5509/(max(y)-min(y))
```

```
0.026887077025966846
```

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

```
0.7983830439876158
```

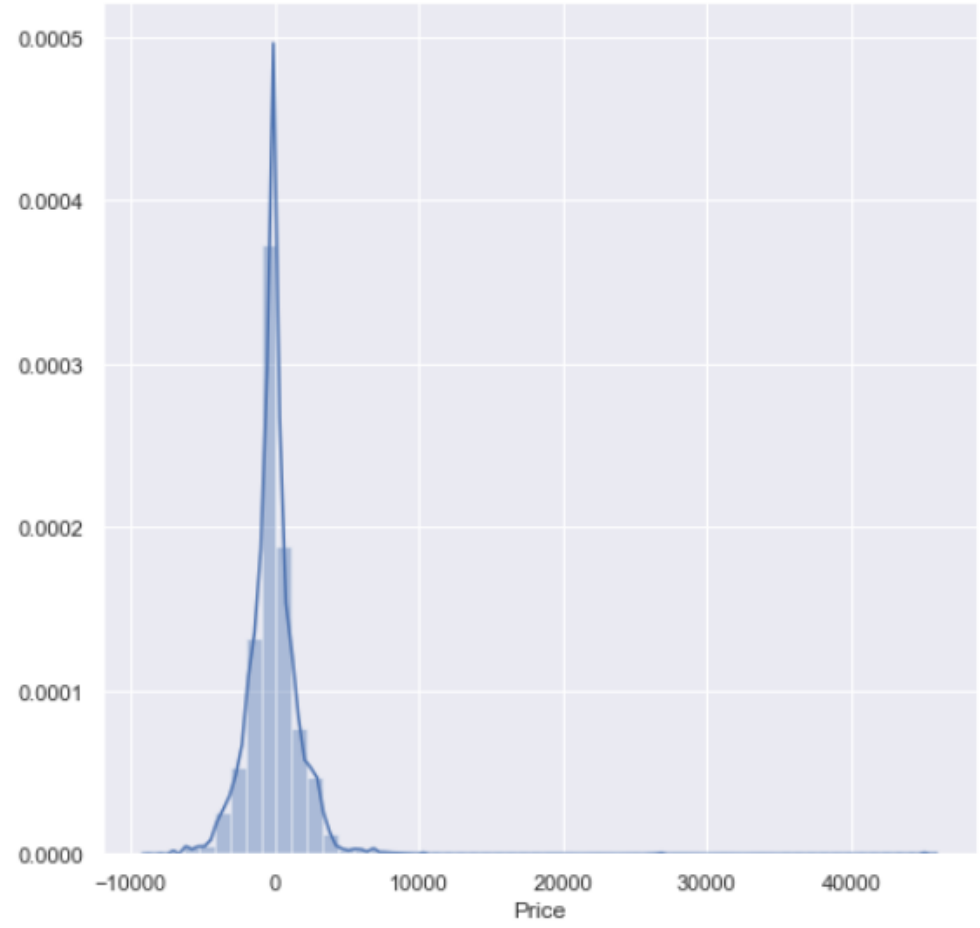
```
from sklearn.model_selection import RandomizedSearchCV
```

```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

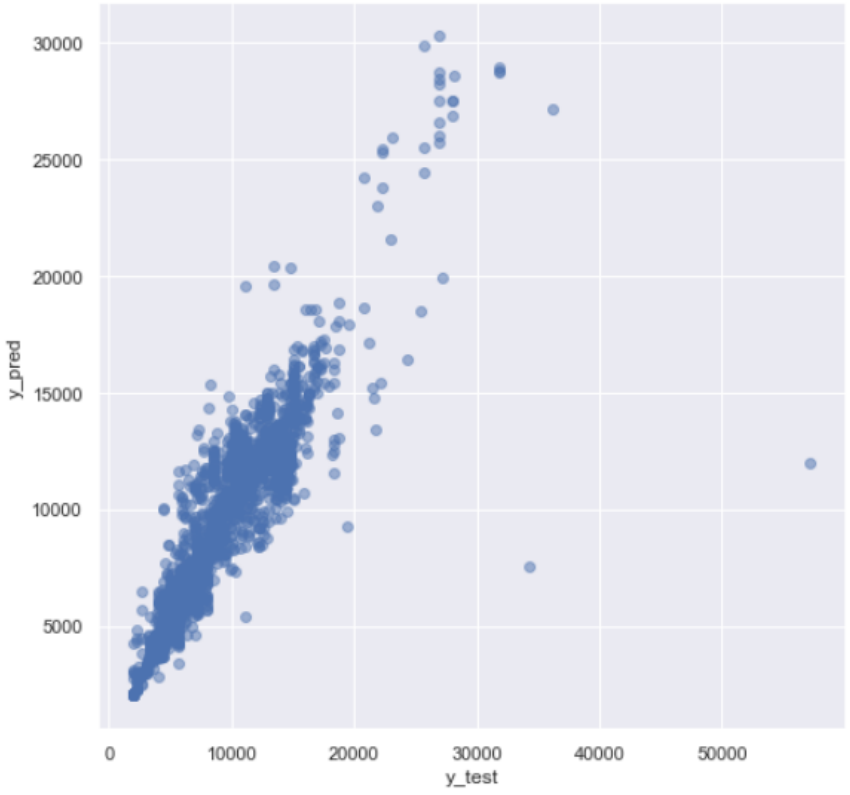
```
# Create the random grid

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter = 1
```

```
plt.figure(figsize = (8,8))
sns.distplot(y_test-prediction)
plt.show()
```




```
plt.figure(figsize = (8,8))
plt.scatter(y_test, prediction, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



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

```
MAE: 1165.606162629916
MSE: 4062650.6911608884
RMSE: 2015.6018186042818
```

Save the model to reuse it again

```
import pickle
# open a file, where you ant to store the data
file = open('flight_rf.pkl', 'wb')

# dump information to that file
pickle.dump(reg_rf, file)
```

```
model = open('flight_price_rf.pkl','rb')
forest = pickle.load(model)
```

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

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

```
0.8117443234361064
```

Live WebApp Demo Link

Deployment on Heroku: https://mlflightpricepredictionapp.herokuapp.com/

Abstract

The purpose of this report will be to use the Train_Data to predict fare of flight depending on details provided by user. This can be used to gain insight into how and why fare price is such at a given time. This can also be used as a model to gain a marketing advantage, by advertisement targeting those who are more likely to purchase flight ticket immediately or in span of 3 months to get statistic figure number of passengers travelling in particular season.
Flight Price Prediction is a regression problem, where using the various parameters or inputs from user model will supply result.

This is diving into Flight Price Prediction through Machine Learning Concept.

End to End Project means that it is step by step process, starts with data collection, EDA, Data Preparation which includes cleaning and transforming then selecting, training and saving ML Models, Cross-validation and Hyper-Parameter Tuning and developing web service then Deployment for end users to use it anytime and anywhere.

This repository contains the code for Flight Price Prediction using python's various libraries.

It used pandas, matplotlib, seaborn, sklearn, numpy, flask and pickle libraries.

These libraries help to perform individually one particular functionality.

Numpy is used for working with arrays. It stands for Numerical Python.
Flask is classified as a microframework because it does not require particular tools or libraries.
Pandas objects rely heavily on Numpy objects.
Matplotlib is a plotting library.
Seaborn is data visualization library based on matplotlib.
Sklearn has 100 to 200 models.
“Pickling” is the process whereby a Python object hierarchy is converted into a byte stream.
The purpose of creating this repository is to gain insights into Complete ML Project.
These python libraries raised knowledge in discovering these libraries with practical use of it.
It leads to growth in my ML repository.
These above screenshots and video in Video_File Folder will help you to understand flow of output.

Motivation

The reason behind building this project is, because I personally like to travel and explore new places and while doing so, I faced one of the challenges is to book flight tickets in cheap price as flight price fluctuates a lot therefore, I found out few tricks that to book tickets on Tuesday or mid-week rather than on weekends, that makes little difference and many other reasons. So, I created Flight Price Prediction Project to gain insights from IT perspective to know working of the model. Hence, I continue to spread tech wings in IT Heaven.

Acknowledgement

Dataset Available: https://www.kaggle.com/nikhilmittal/flight-fare-prediction-mh

The Data

It has 11 columns with only one column being numeric and rest all categorical.
This data has 10683 total observations(rows).

```
train_data.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
```

It displays number of missing/ null values in each column.

```
train_data.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
```

Here are all the features included in the data set, and a short description of them all.

ColumnName	Description
Airline	Displays brand/company of airline.
Date_of_Journey	Displays date and date format.
Source	Displays source/starting destination of flight for respective passenger.
Destination	Displays destination/end stop of flight for respective passenger.
Route	Indicates route of flight.
Dep_Time	Shows Departure Time of flight.
Arrival_Time	Shows Arrival Time of flight.
Duration	Indicates total duration of journey.
Total_Stops	Displays total number of stops that a passenger will have as per selection.
Additional_Info	Displays any important or emergency information regarding flight.

```
train_data.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

Analysis of the Data

Let's start by doing a general analysis of the data as a whole, including all the features the Random Forest algorithm will be using.

- Basic Statistics

```
train_data.describe()
```

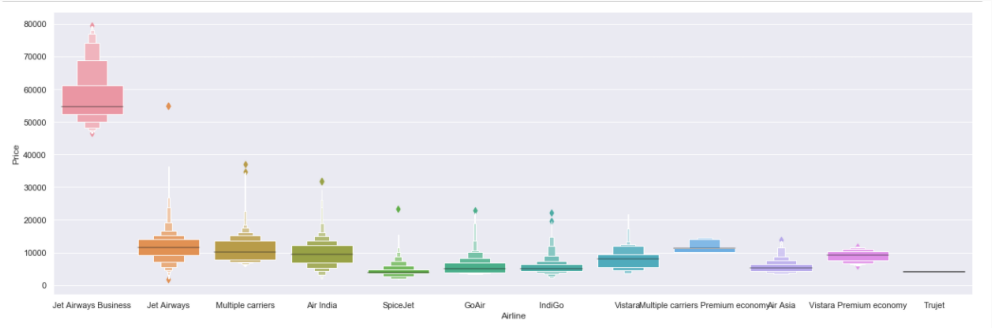
	Price
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000
50%	8372.000000
75%	12373.000000
max	79512.000000

```
test_data.describe()
```

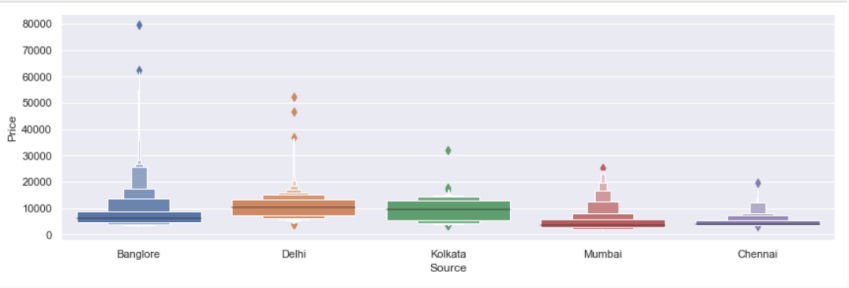
	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
count	2671	2671	2671	2671	2671	2671	2671	2671	2671	2671
unique	11	44	5	6	100	199	704	320	5	6
top	Jet Airways	9/05/2019	Delhi	Cochin	DEL → BOM → COK	10:00	19:00	2h 50m	1 stop	No info
freq	897	144	1145	1145	624	62	113	122	1431	2148

- Graphing of Features

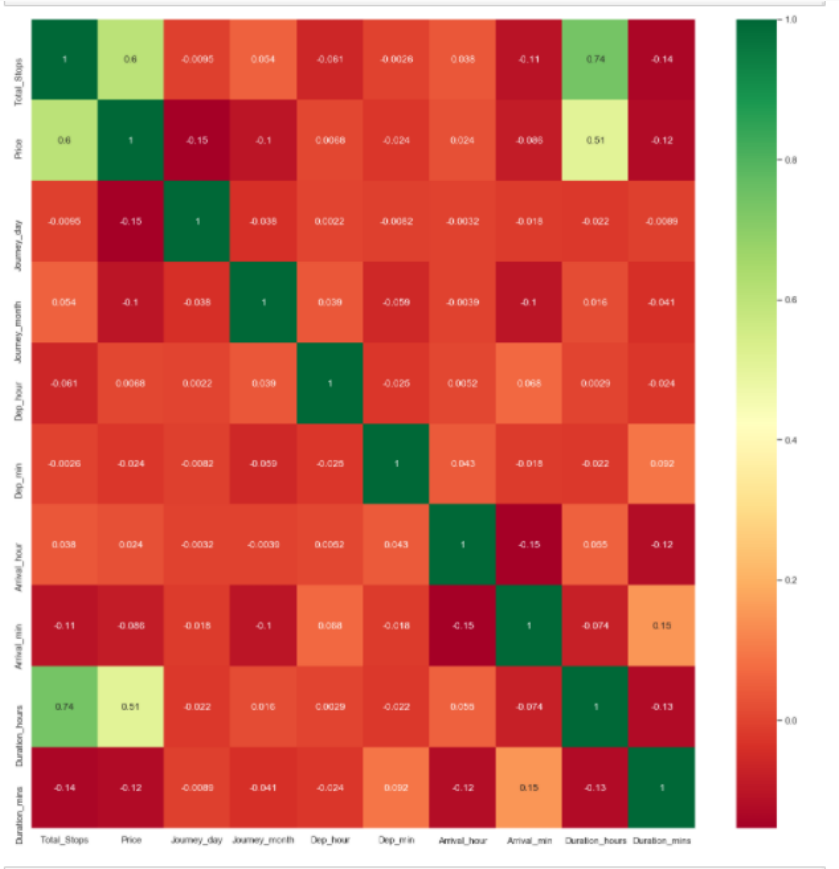
Graph Set 1



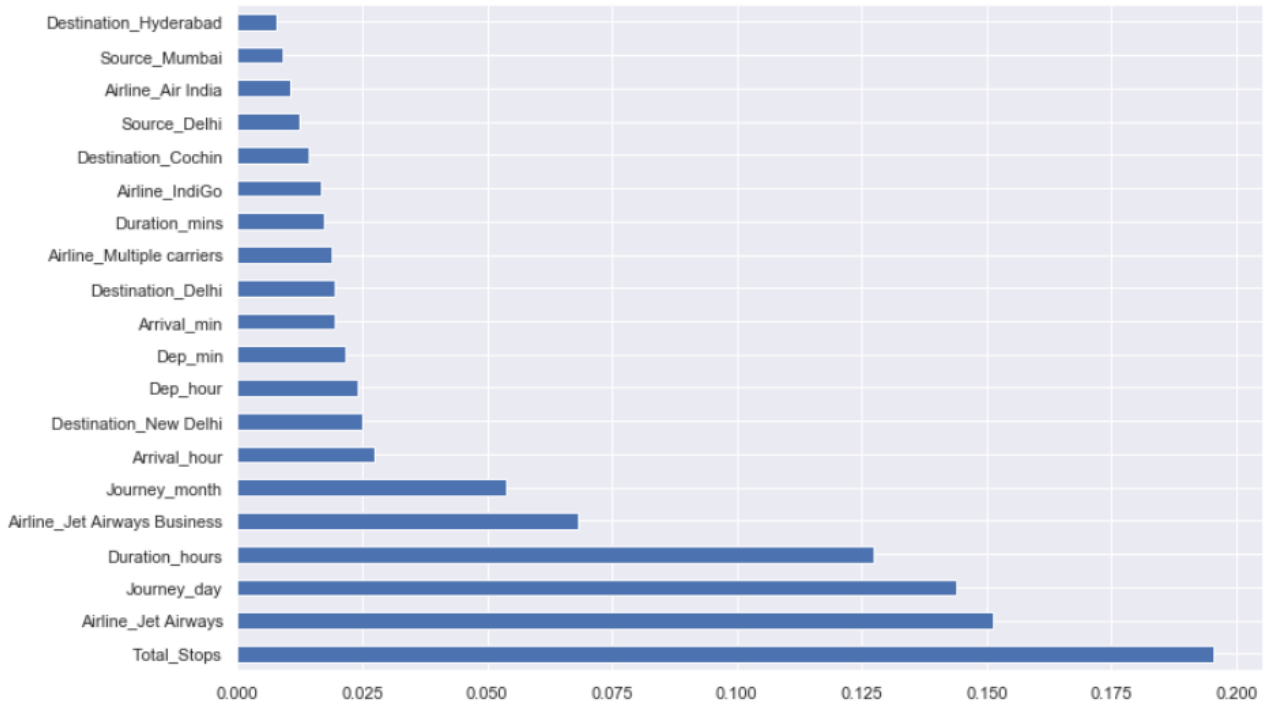
Graph Set 2



Graph Set 3



Graph Set 4



Modelling

The purpose of these models will be to get effective insight into the following:

1. If price of flight depending on various factors:
    - This insight can be used for Market Targeting.
2. Get insight into how changing RMSE of the predictions affect:
    - Spending more money to target the passengers that are most likely to retain their customer loyalty or spending less money by providing less flexibility services.

Where to Use RF Regression: RF Regression Example

Let's say you want to estimate the average household income in your town. You could easily find an estimate using the Random Forest Algorithm. You would start off by distributing surveys asking people to answer a number of different questions. Depending on how they answered these questions, an estimated household income would be generated for each person.

After you've found the decision trees of multiple people you can apply the Random Forest Algorithm to this data. You would look at the results of each decision tree and use random forest to find an average income between all of the decision trees. Applying this algorithm would provide you with an accurate estimate of the average household income of the people you surveyed.

- Math behind the metrics

Scale-dependent measures (for example: MSE, RMSE, MAE) because it is scale-dependent, i.e. dependent on your dependent variable.

The mean absolute error (MAE) of a model with respect to a test set is the mean of the absolute values of the individual prediction errors on over all instances in the test set.

In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. MSE is a risk function, corresponding to the expected value of the squared error loss. The fact that MSE is almost always strictly positive (and not zero) is because of randomness or because the estimator does not account for information that could produce a more accurate estimate.

The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

When using the Random Forest Algorithm to solve regression problems, you are using the mean squared error (MSE) to how your data branches from each node.

This formula calculates the distance of each node from the predicted actual value, helping to decide which branch is the better decision for your forest. Here, *yi* is the value of the data point you are testing at a certain node and *fi* is the value returned by the decision tree.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (fi - yi)^2$$

Where N is the number of data points,
fi is the value returned by the model and
yi is the actual value for data point i.

RMSE is computed as RMSE = mean((observeds - predicteds)^2) %>% sqrt() . The lower the RMSE, the better the model.

Try to play with other input variables, and compare your RMSE values. The smaller the RMSE value, the better the model. Also, try to compare your RMSE values of both training and testing data. If they are almost similar, your model is good.

The Mean Squared Error (MSE) is a measure of how close a fitted line is to data points.

The MSE has the units squared of whatever is plotted on the vertical axis. Another quantity that we calculate is the Root Mean Squared Error (RMSE). It is just the square root of the mean square error.

The MAE is a linear score which means that all the individual differences are weighted equally in the average. The RMSE is a quadratic scoring rule which measures the average magnitude of the error. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. Therefore, MAE is better than RMSE.

Algorithm for Random Forest Regression:

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

RandomizedSearchCV

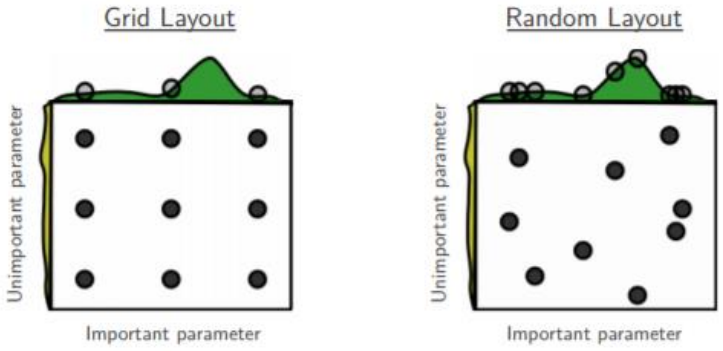


Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

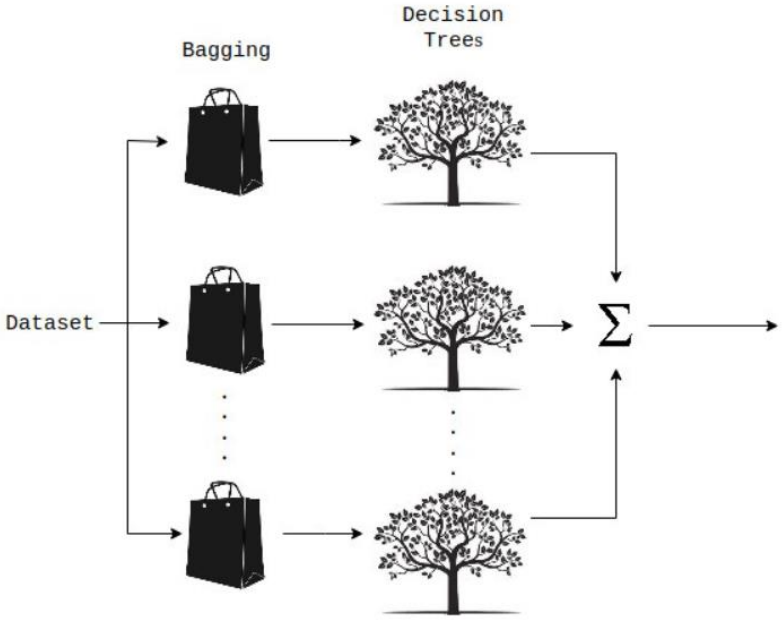
Reference Link: https://deepai.org/machine-learning-glossary-and-terms/random-forest

https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

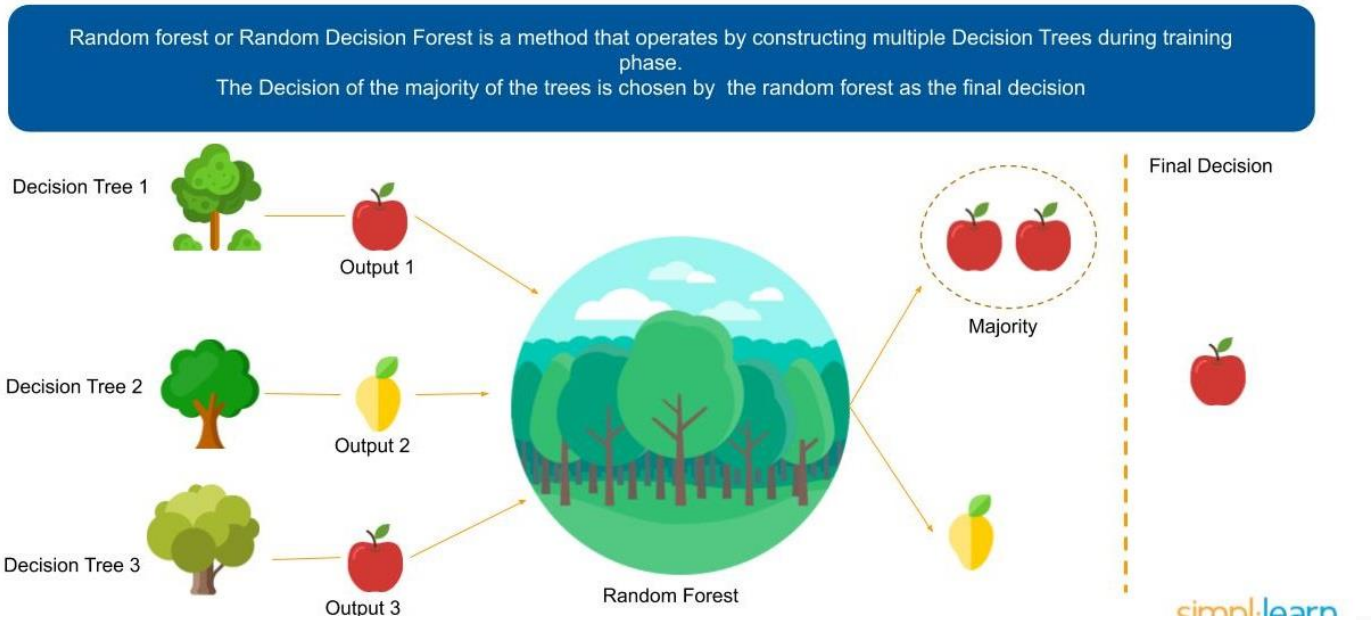
https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

- Model Architecture Process Through Visualization

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement.



Random Forest Regression: Process



The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

When Does RF Regression Fails and Why - https://neptune.ai/blog/random-forest-regression-when-does-it-fail-and-why

I will use RandomizedSearchCV from sklearn to optimize our hyperparamaters.
Now I have plugged these back into the model to see if it improved our performance.

- Quick Notes

Step 1: Imported required libraries.

Step 2: Imported Data_Train.xlsx Dataset and read it.

Step 3: Analysed the data using '.info(), .value_counts(), .isnull().sum()' commands and imputing it.

Step 4: Performed Exploratory Data Analysis (EDA) on the data.

- Converting Date_of_Journey Column from object type to datetime type that is, from categorical to integer datatype.
- Dropping the Date_of_Journey Column.
- Same procedure I followed for Dep_Time Column and Arrival_Time Column.
- Converting Duration Column into list type and appending hours and mins to it.

Step 5: Now, I have handled Categorical Data.

- Performed OneHotEncoding on Nominal Data as from visualization is clearly seen that Jet Airways has highest price on Airline Column and creating dummies for the same through get_dummies function.
- Creating dummies for also Source Column and Destination Column.
- Perfomed LabelEncoder on Total_stops Column.
- Concatenated above four columns and dropped three out of them.
- So, now I have 10683 rows and 30 columns.

Step 6: Now, read and pre-processed Test_set.xlsx Dataset.

Step 7: Analysed the data using '.info(), .value_counts(), .isnull().sum()' commands.

Step 8: Performed Exploratory Data Analysis (EDA) on the data.

- Converted Date_of_Journey Column from object type to datetime type that is, from categorical to integer datatype.
- Dropped the Date_of_Journey Column.
- Same procedure I followed for Dep_Time Column and Arrival_Time Column.
- Converted Duration Column into list type and appending hours and mins to it.

Step 9: Now, I have handled Categorical Data.

- Performed OneHotEncoding on Nominal Data as from visualization is clearly seen that Jet Airways has highest price on Airline Column and creating dummies for the same through get_dummies function.
- Created dummies for also Source Column and Destination Column.
- Perfomed LabelEncoder on Total_stops Column.
- Concatenated above four columns and dropped three out of them.
- So, now I have 2671 rows and 28 columns.

Step 10: Performed Feature Selection to find best feature which will contribute to target vars.

- Performed .loc and .iloc commands.
- Found correlation between Independent and dependent attributes through heatmap plotting.
- Imported feature using ExtraTreesRegressor.

Step 11: Plotted graph of feature importances for better visualization.

Step 12: Split dataset into train and test set in order to prediction w.r.t X_test.

Step 13: Imported RF Regressor Model and Fitted the Data on it.

Step 14: Predicted w.r.t. X_test.

Step 15: Checked RF Regressor Score through Visualization using distplot and scatterplot.

Step 16: Now, checked Model Evaluation by using MAE, MSE and RMSE Error Measures.

Step 17: Now, Tuned Hyper-parameter with the help of RandomizedSearchCV Method.

- Assigned hyper-parameters in the form of dictionary.
- Created random grid.

- Used 5-fold cross-validation.
- Fitted the model.
- Found best parameters and best scores.

Step 18: Checked RandomSearchCV Score through Visualization using distplot and scatterplot.
Step 19: Now, checked Model Evaluation by using MAE, MSE and RMSE Error Measures.
Step 20: Saved the model for re-use.
Step 21: Created Web App.

- The Model Analysis

Imported necessary libraries - When we import modules, we're able to call functions that are not built into Python. Some modules are installed as part of Python, and some we will install through pip. Making use of modules allows us to make our programs more robust and powerful as we're leveraging existing code.

Read Data - You can import tabular data from Excel files into pandas data frame by specifying a parameter value for the file. Imported Data_Train.xlsx file here.

Analysed the data - Using '.info(), .value_counts(), .isnull().sum()' commands and imputing it. Checking unique number of categories in each column. Checking missing values to deal with them.

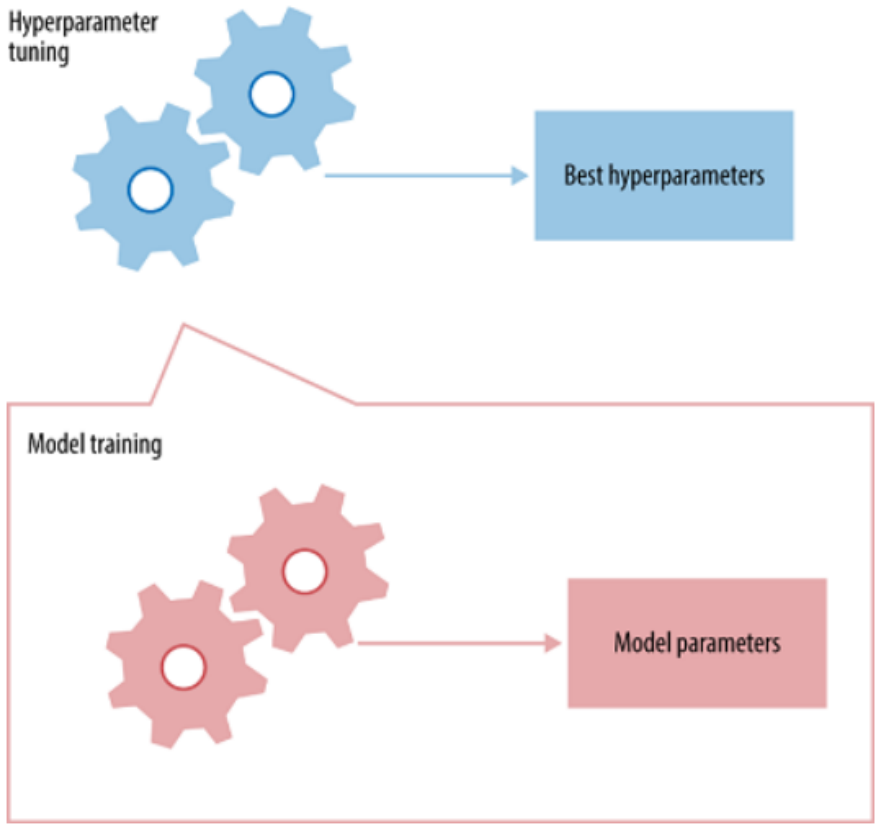
Performed Exploratory Data Analysis (EDA) on the data - The primary goal of EDA is to maximize the analyst's insight into a data set and into the underlying structure of a data set, while providing all of the specific items that an analyst would want to extract from a data set, such as: a good-fitting, parsimonious model and a list of outliers. There are many libraries available in python like pandas, NumPy, matplotlib, seaborn to perform EDA. The four types of EDA are univariate non-graphical, multivariate non- graphical, univariate graphical, and multivariate graphical. First, converted Date_of_Journey Column from object type to datetime type that is, from categorical to integer datatype. Then, I Dropped the Date_of_Journey Column because it is of no use. Third, Same procedure I followed for Dep_Time Column and Arrival_Time Column. Fourth, Converted Duration Column into list type and appending hours and mins to it.

Handled Categorical Data - There are various ways to handle categorical columns. Basic, idea is, handling means changing its data type be it from object data type to numerical/integer datatype. First, performed OneHotEncoding on Nominal Data as from visualization is clearly seen that Jet Airways has highest price on Airline Column and creating dummies for the same through get_dummies function. A one hot encoding allows the representation of categorical data to be more expressive. One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. There are two different ways to encoding categorical variables. One-hot encoding converts it into n variables, while dummy encoding converts it into n-1 variables. If we have k categorical variables, each of which has n values. Second, created dummies for also Source Column and Destination Column. Pandas has a function named get_dummies. It will convert your categorical string values into dummy variables. pd.get_dummies create a new dataframe containing unique values as columns which consists of zeros and ones. Third, perfomed LabelEncoder on Total_stops Column. LabelEncoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels. If a label repeats it assigns the same value to as assigned earlier. The categorical values have been converted into numeric values. Fourth, concatenated above four columns and dropped three out of them. Finally, now I have 10683 rows and 30 columns.

Read Data - You can import tabular data from Excel files into pandas data frame by specifying a parameter value for the file. Now, read and pre-processed Test_set.xlsx Dataset.

Analysed the data - Using '.info(), .value_counts(), .isnull().sum()' commands. Checking unique number of categories in each column. Checking missing values to deal with them.
Performed Exploratory Data Analysis (EDA) on the data - The primary goal of EDA is to maximize the analyst's insight into a data set and into the underlying structure of a data set, while providing all of the specific items that an analyst would want to extract from a data set, such as: a good-fitting, parsimonious model and a list of outliers. There are many libraries available in python like pandas, NumPy, matplotlib, seaborn to perform EDA. The four types of EDA are univariate non-graphical, multivariate non- graphical, univariate graphical, and multivariate graphical. First, converting Date_of_Journey Column from object type to datetime type that is, from categorical to integer datatype. Then, I Dropped the Date_of_Journey Column. Third, same procedure I followed for Dep_Time Column and Arrival_Time Column. Fourth, converting Duration Column into list type and appending hours and mins to it.
Handled Categorical Data - There are various ways to handle categorical columns. Basic, idea is, handling means changing its data type be it from object data type to numerical/integer datatype. First, performed OneHotEncoding on Nominal Data as from visualization is clearly seen that Jet Airways has highest price on Airline Column and creating dummies for the same through get_dummies function. A one hot encoding allows the representation of categorical data to be more expressive. One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. There are two different ways to encoding categorical variables. One-hot encoding converts it into n variables, while dummy encoding converts it into n-1 variables. If we have k categorical variables, each of which has n values. Second, created dummies for also Source Column and Destination Column. Pandas has a function named get_dummies. It will convert your categorical string values into dummy variables. pd.get_dummies create a new dataframe containing unique values as columns which consists of zeros and ones. Third, perfomed LabelEncoder on Total_stops Column. LabelEncoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels. If a label repeats it assigns the same value to as assigned earlier. The categorical values have been converted into numeric values. Fourth, concatenated above four columns and dropped three out of them. Finally, now I have 2671 rows and 28 columns.
Performed Feature Selection - To find best feature which will contribute to target vars. First, Performed .loc and .iloc commands. These are used in slicing of data from the Pandas DataFrame. They help in the convenient selection of data from the DataFrame. They are used in filtering the data according to some conditions. Advantage over loc is that this is faster. Disadvantage is that you can't use arrays for indexers. Second, found correlation between Independent and dependent attributes through heatmap plotting. The result will display the strength and direction of the relationship. Then, imported feature using ExtraTreesRegressor. The Extra-Trees algorithm builds an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with other tree-based ensemble methods are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample (rather than a bootstrap replica) to grow the trees.
Plotted graph of feature importance for better visualization - Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature.
Split dataset - Splitting data into train and test set in order to prediction w.r.t X_test. This helps in prediction after training data.

Imported RF Regressor Model and Fitted the Data on it – The fit() method takes the training data as arguments, which can be one array in the case of unsupervised learning, or two arrays in the case of supervised learning.
Predicted Test Data - w.r.t. X_test. It helps in better analysing performance of model.
Checked RF Regressor Score - Through Visualization using distplot and scatterplot. It gives understanding of model performance.
Checked Model Evaluation - By using MAE, MSE and RMSE Error Measures. It indicates how better model is performing.
Tuned Hyper-parameter - Used RandomizedSearchCV Method. First, assigned hyper-parameters in the form of dictionary. Then, created random grid. Third, used 5-fold cross-validation. Then, fitted the model. And, found best parameters and best scores.
The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance, just as we might turn the knobs of an AM radio to get a clear signal (or your parents might have!). While model parameters are learned during training — such as the slope and intercept in a linear regression — hyperparameters must be set by the data scientist before training. In the case of a random forest, hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node. (The parameters of a random forest are the variables and thresholds used to split each node learned during training). Scikit-Learn implements a set of sensible default hyperparameters for all models, but these are not guaranteed to be optimal for a problem. The best hyperparameters are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from a science into trial-and-error based engineering.

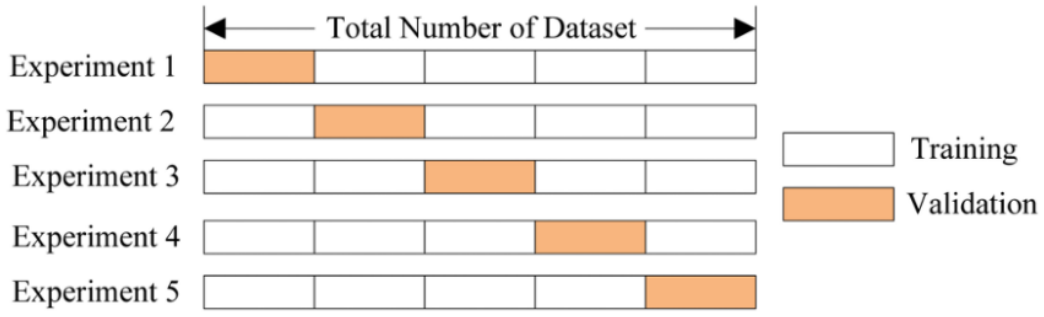


Hyperparameters and Parameters

Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations evaluate the performance of each model. However, evaluating each model only on the training set can lead to one of the most

fundamental problems in machine learning: overfitting. If we optimize the model for the training data, then our model will score very well on the training set, but will not be able to generalize to new data, such as in a test set. When a model performs highly on the training set but poorly on the test set, this is known as overfitting, or essentially creating a model that knows the training set very well but cannot be applied to new problems. It's like a student who has memorized the simple problems in the textbook but has no idea how to apply concepts in the messy real world. An overfit model may look impressive on the training set, but will be useless in a real application. Therefore, the standard procedure for hyperparameter optimization accounts for overfitting through cross validation.

The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data). As an example, consider fitting a model with K = 5. The first iteration we train on the first four folds and evaluate on the fifth. The second time we train on the first, second, third, and fifth fold and evaluate on the fourth. We repeat this procedure 3 more times, each time evaluating on a different fold. At the very end of training, we average the performance on each of the folds to come up with final validation metrics for the model.



5 Fold Cross Validation (Source)

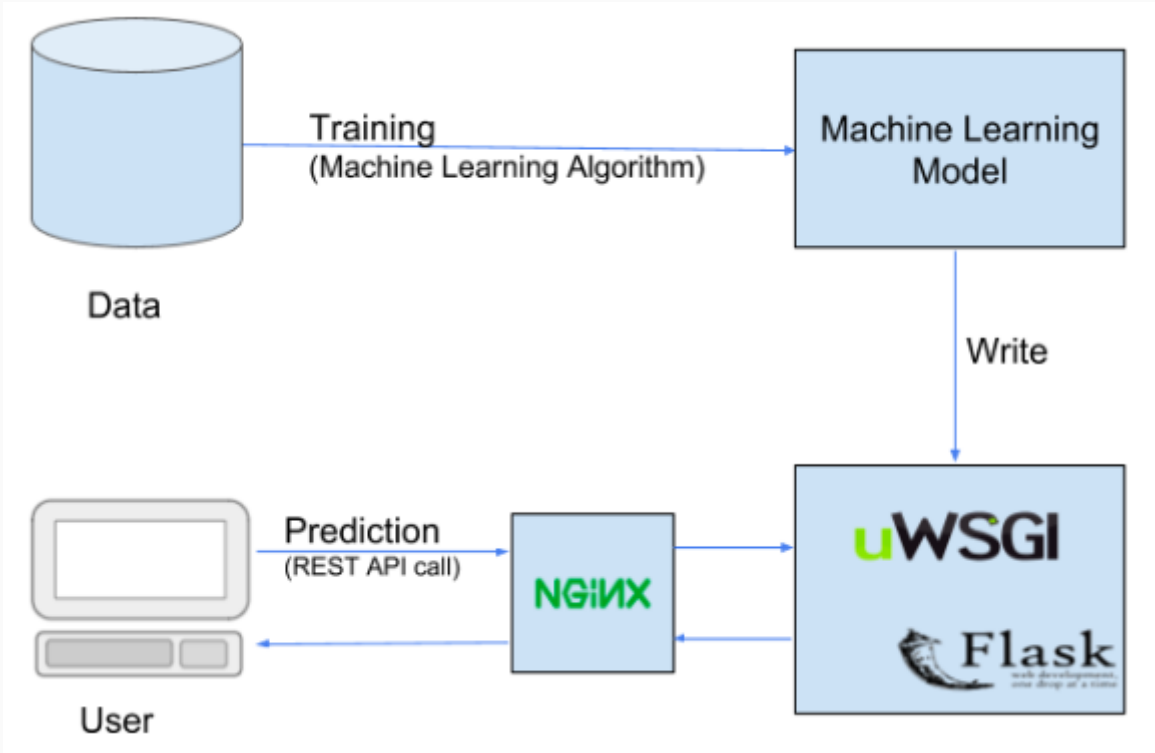
For hyperparameter tuning, we perform many iterations of the entire K-Fold CV process, each time using different model settings. We then compare all of the models, select the best one, train it on the full training set, and then evaluate on the testing set. This sounds like an awfully tedious process! Each time we want to assess a different set of hyperparameters, we have to split our training data into K fold and train and evaluate K times. If we have 10 sets of hyperparameters and are using 5-Fold CV, that represents 50 training loops. Fortunately, as with most problems in machine learning, someone has solved our problem and model tuning with K-Fold CV can be automatically implemented in Scikit-Learn.

Using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-Fold CV with each combination of values.

Checked RandomSearchCV Score through Visualization using distplot and scatterplot – It gives thorough knowledge on model insights.

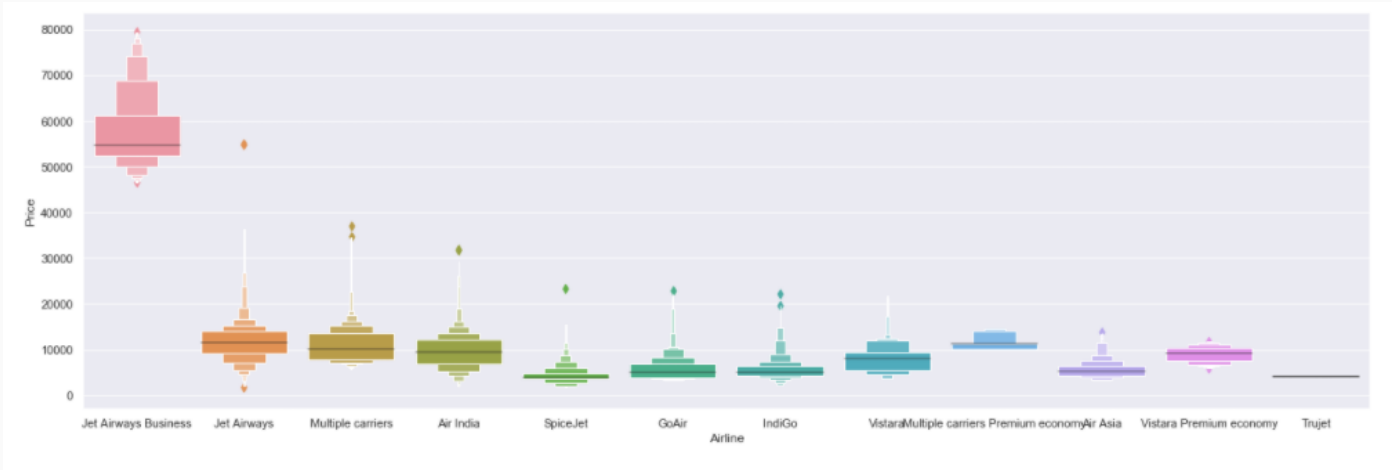
Checked Model Evaluation by using MAE, MSE and RMSE Error Measures - To evaluate the prediction error rates and model performance in regression analysis. Once you have obtained your error metric/s, take note of which X's have minimal impacts on y. Removing some of these features may result in an increased accuracy of your model. MAE: The easiest to understand. Represents average error.

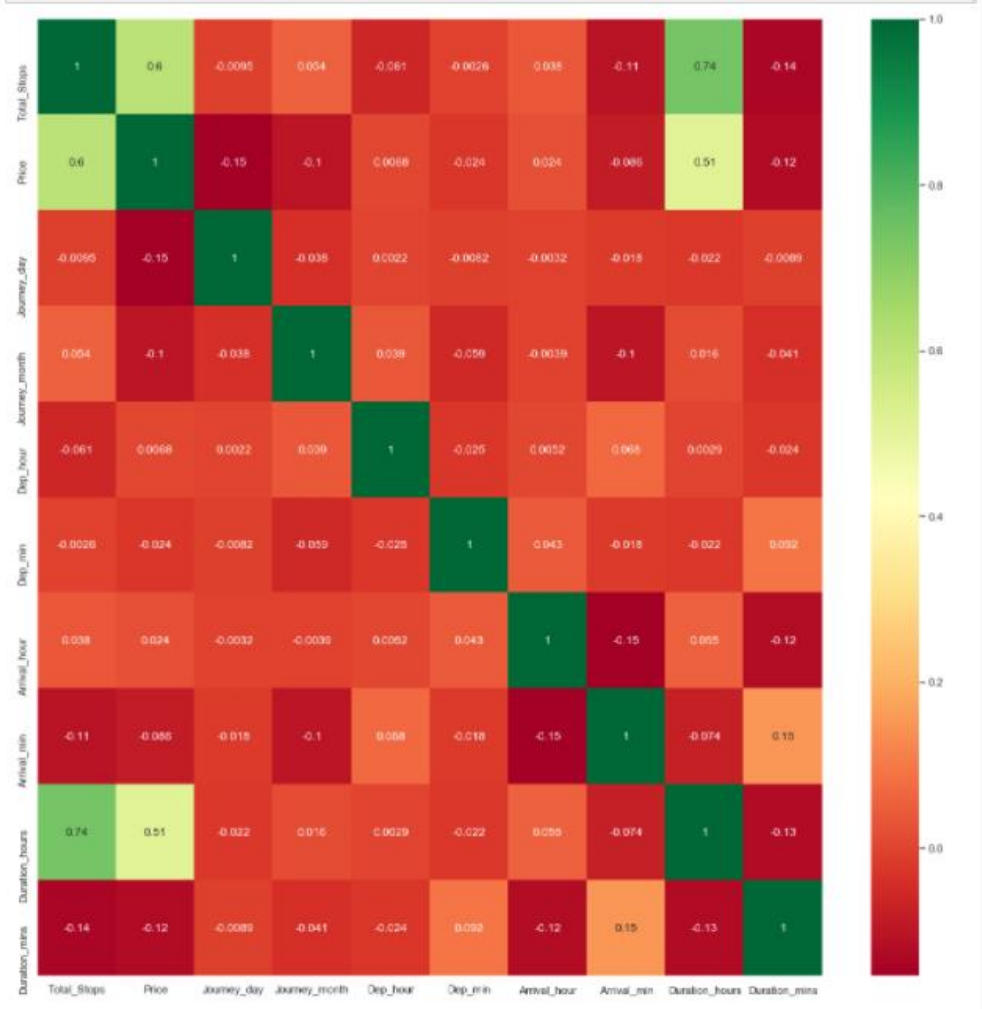
MSE: Similar to MAE but noise is exaggerated and larger errors are “punished”. It is harder to interpret than MAE as it’s not in base units, however, it is generally more popular.
RMSE: Most popular metric, similar to MSE, however, the result is square rooted to make it more interpretable as it’s in base units. It is recommended that RMSE be used as the primary metric to interpret your model. All of them require two lists as parameters, with one being your predicted values and the other being the true values.
Saved the model for re-use – Saving the created model for future reference and use so that we can directly access it rather than to go through full cycle again.
Created Web App - It is made in flask for end-users to use it.
Challenge that I faced, accuracy issue even after using random forest model. So, I used hyper-parameter tuning and cross-validation concepts.

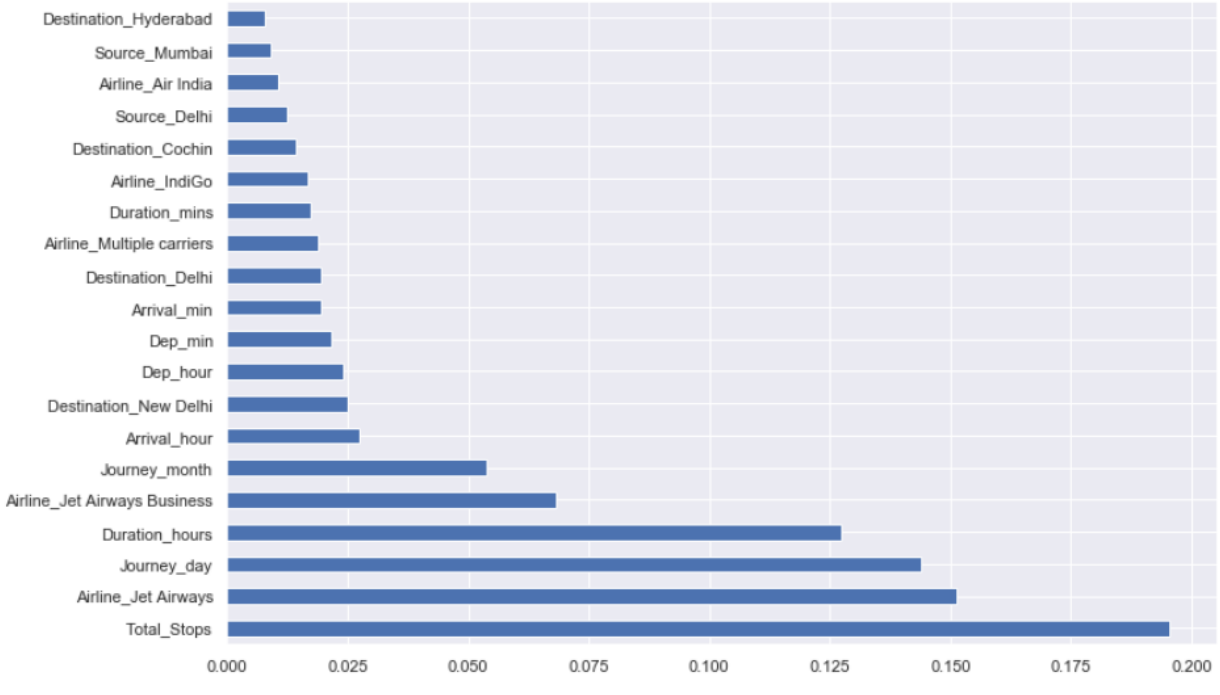


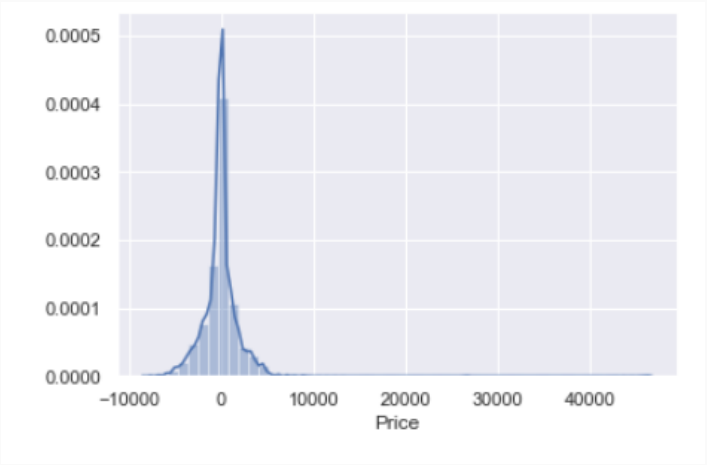
Checking the Model Visualization

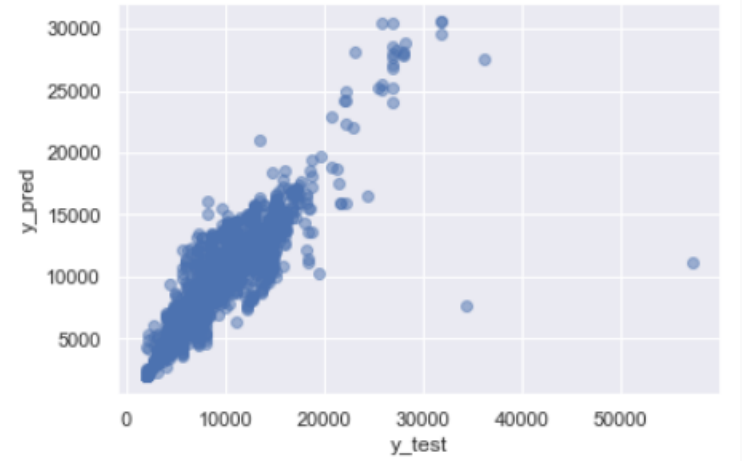
- Basic Model Evaluation Graphs

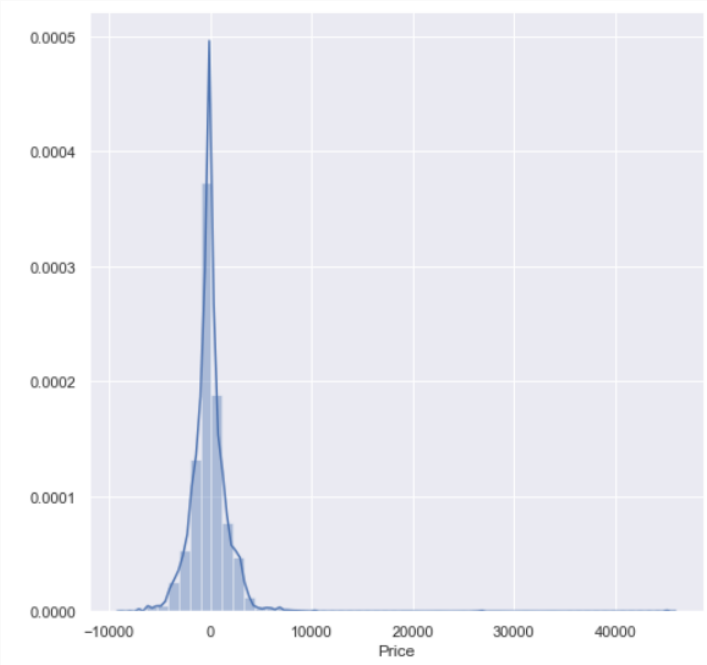


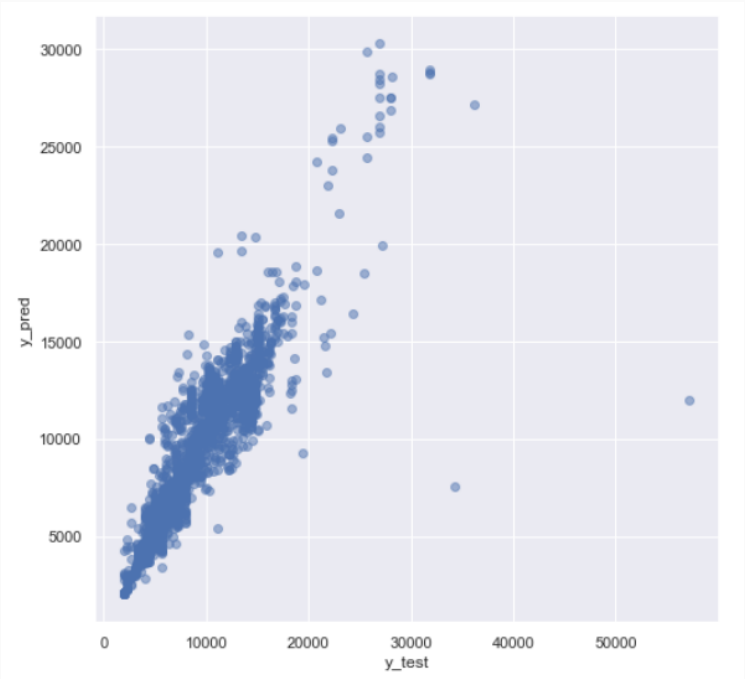

Total_Stops
Price
Journey_day
Journey_month
Dep_hour
Dep_min
Arrival_hour
Arrival_min
Duration_hours
Duration_mins
1.0
0.8
0.6
0.4
0.2
0.0


Destination_Hyderabad
Source_Mumbai
Airline_Air India
Source_Delhi
Destination_Cochin
Airline_IndiGo
Duration_mins
Airline_Multiple carriers
Destination_Delhi
Arrival_min
Dep_min
Dep_hour
Destination_New Delhi
Arrival_hour
Journey_month
Airline_Jet Airways Business
Duration_hours
Journey_day
Airline_Jet Airways
Total_Stops
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200









Creation of App

Here, I am creating Flask App. Loading the model that we saved then calling and calculating all the factors that are necessary in deciding price of flight with relevant datatype to take from user. Displaying message if condition matched.

Get the user entered value from the predict method and render respective .html page for solution. And displays 'Flight Price is Rs '.

You can create it with Streamlit also.

Technical Aspect

Numpy used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. It contains a multi-dimensional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays.

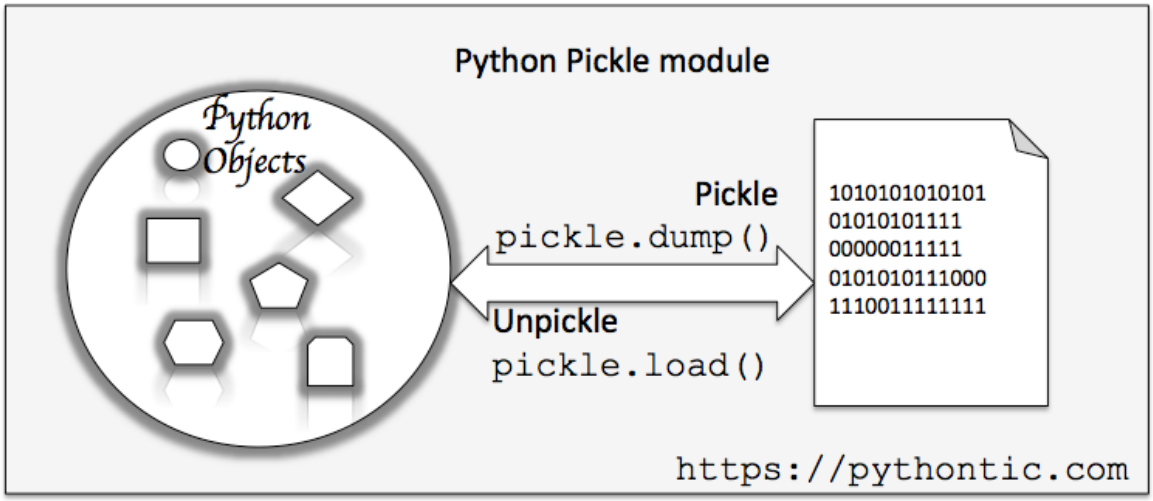
Pandas module mainly works with the tabular data. It contains Data Frame and Series. Pandas is 18 to 20 times slower than Numpy. Pandas is seriously a game changer when it comes to cleaning, transforming, manipulating and analyzing data.

Matplotlib is used for EDA. Visualization of graphs helps to understand data in better way than numbers in table format. Matplotlib is mainly deployed for basic plotting. It consists of bars, pies, lines, scatter plots and so on. Inline command display visualization inline within frontends like in Jupyter Notebook, directly below the code cell that produced it.

Seaborn provides a high-level interface for drawing attractive and informative statistical graphics. It provides a variety of visualization patterns and visualize random distributions.

Sklearn is known as scikit learn. It provides many ML libraries and algorithms for it. It provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

Pickle in Python is primarily used in serializing and deserializing a Python object structure. In other words, it's the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network.



Need to train_test_split - Using the same dataset for both training and testing leaves room for miscalculations, thus increases the chances of inaccurate predictions.

The function train_test_split allows you to break a dataset with ease while pursuing an ideal model. Also, keep in mind that your model should not be overfitting or underfitting.

Reason for doing Hyper-Parameter Tuning is, Setting the correct combination of hyperparameters is the only way to extract the maximum performance out of models.

The most important arguments in RandomizedSearchCV are n_iter, which controls the number of different combinations to try, and cv which is the number of folds to use for cross validation. RandomizedSearchCV implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used. If at least one parameter is given as a distribution, sampling with replacement is used.

In the realm of machine learning, the random forest regression algorithm can be more suitable for regression problems than other common and popular algorithms. Below are a few cases where you’d likely prefer a random forest algorithm over other regression algorithms:

1. There are non-linear or complex relationships between features and labels.
2. You need a model that’s robust, meaning its dependence on the noise in the training set is limited. The random forest algorithm is more robust than a single decision tree, as it uses a set of uncorrelated decision trees.
3. If your other linear model implementations are suffering from overfitting, you may want to use a random forest.

Extra Trees is an ensemble machine learning algorithm that combines the predictions from many decision trees.

It is related to the widely used random forest algorithm. It can often achieve as-good or better performance than the random forest algorithm, although it uses a simpler algorithm to construct the decision trees used as members of the ensemble.

It is also easy to use given that it has few key hyperparameters and sensible heuristics for configuring these hyperparameters.
The Extra Trees algorithm works by creating a large number of unpruned decision trees from the training dataset. Predictions are made by averaging the prediction of the decision trees in the case of regression or using majority voting in the case of classification.

- Regression: Predictions made by averaging predictions from decision trees.
- Classification: Predictions made by majority voting from decision trees.

Unlike bagging and random forest that develop each decision tree from a bootstrap sample of the training dataset, the Extra Trees algorithm fits each decision tree on the whole training dataset.
Like random forest, the Extra Trees algorithm will randomly sample the features at each split point of a decision tree. Unlike random forest, which uses a greedy algorithm to select an optimal split point, the Extra Trees algorithm selects a split point at random.
There are three main hyperparameters to tune in the algorithm; they are the number of decision trees in the ensemble, the number of input features to randomly select and consider for each split point, and the minimum number of samples required in a node to create a new split point.
Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Installation

Using intel core i5 9th generation with NVIDIA GFORECE GTX1650.
Windows 10 Environment Used.
Already Installed Anaconda Navigator for Python 3.x
The Code is written in Python 3.8.
If you don't have Python installed then please install Anaconda Navigator from its official site.
If you are using a lower version of Python you can upgrade using the pip package, ensuring you have the latest version of pip, *python* -m pip install --*upgrade pip and press Enter*.

Run/How to Use/Steps

Keep your internet connection on while running or accessing files and throughout too.
Follow this when you want to perform from scratch.
Open Anaconda Prompt, Perform the following steps:
cd <PATH>
pip install flask
pip install numpy
pip install pandas
pip install joblib
Note: If it shows error as 'No Module Found' , then install relevant module.
You can also create requirement.txt file as, pip freeze > requirements.txt
Create Virtual Environment:
conda create -n flight python=3.6
y
conda activate flight
cd <PATH-TO-FOLDER>

run .py or .ipynb files.
Paste URL to browser to check whether working locally or not.

Follow this when you want to just perform on local machine.
Download ZIP File.
Right-Click on ZIP file in download section and select Extract file option, which will unzip file.
Move unzip folder to desired folder/location be it D drive or desktop etc.
Open Anaconda Prompt, write cd <PATH> and press Enter.
eg: cd C:\Users\Monica\Desktop\Projects\Python Projects 1\
23)End_To_End_Projects\Project_5_ML_FileUse_EndToEnd_FlightPricePrediction\Project_ML_FlightPricePrediction

conda create -n flight python=3.6
y
conda activate flight
In Anconda Prompt, pip install -r requirements.txt to install all packages.
In Anconda Prompt, write python app.py and press Enter.
Paste URL 'http://127.0.0.1:5000/' to browser to check whether working locally or not.
Please be careful with spellings or numbers while typing filename and easier is just copy filename and then run it to avoid any silly errors.
Note: cd <PATH>
[Go to Folder where file is. Select the path from top and right-click and select copy option and paste it next to cd one space <path> and press enter, then you can access all files of that folder]
[cd means change directory]

Directory Tree/Structure of Project

Folder: 23)End_To_End_Projects > Project_5_ML_FileUse_EndToEnd_FlightPricePrediction> Project_ML_FlightPricePrediction

Model_Building.ipynb
app.py
Data_Train.xlsx
Test_set.xlsx
requirements.txt
Procfile

Folder: 23)End_To_End_Projects > Project_5_ML_FileUse_EndToEnd_FlightPricePrediction> Project_ML_FlightPricePrediction> static > css > styles.css

Folder: 23)End_To_End_Projects > Project_5_ML_FileUse_EndToEnd_FlightPricePrediction> Project_ML_FlightPricePrediction> templates > home.html

```
Project_ML_FlightPricePrediction/
├── Images_screenshot/
├── static/
│   ├── css
│   │   └── styles.css
├── templates/
│   └── home.html
├── app.py
├── Procfile
├── requirements.txt
├── Model_Building.ipynb
├── Data_Train.xlsx
├── Test_set.xlsx
└── flight_rf.pkl
```

To Do/Future Scope

Can deploy on AWS and Google Cloud.

Technologies Used/System Requirements/Tech Stack

Download the Material

You can Download Dataset from here: https://github.com/monicadesAI-tech/Project_72/blob/main/Data_Train.xlsx
https://github.com/monicadesAI-tech/Project_72/blob/main/Test_set.xlsx

You can Download Entire Project from here: https://github.com/monicadesAI-tech/Project_72

You can Download Model from here: https://github.com/monicadesAI-tech/Project_72/blob/main/Model_Building.ipynb

You can Download App_File from here: https://github.com/monicadesAI-tech/Project_72/blob/main/app.py

You can see Detailed Project at Website here: https://github.com/monicadesAI-tech.github.io/project/flightprice.html

Download the saved model from here: https://github.com/monicadesAI-tech/Project_72/blob/main/flight_rf.pkl

Download dependencies from here: https://github.com/monicadesAI-tech/Project_72/blob/main/requirements.txt

Conclusion

- Modelling

Can use XGBoost and see if any major performance difference.

- Analysis

```
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: 1172.5455945373583
MSE: 4347276.1614450775
RMSE: 2085.0122688955757
```

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

```
MAE: 1165.606162629916
MSE: 4062650.6911608884
RMSE: 2015.6018186042818
```

Credits

Krish Naik Channel

Paper Citation

https://www.sciencedirect.com/science/article/pii/S131915781830884X