FLIP ROBO

Flight Price Prediction

Submitted by:

Dheerajkumar Pittala

Acknowledgment

First of all I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity.

The Data was collected from the below websites

- https://www.yatra.com/

Most of the concepts used to predict the Micro-Credit loan defaulters are learned from Data Trained Institute and below documentations.

- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://www.scipy.org/
- Stack-overflow
- https://imbalanced-learn.org/stable/

Introduction

Airline companies use complex algorithms to calculate flight prices given various conditions present at that particular time. These methods take financial, marketing, and various social factors into account to predict flight prices.

Nowadays, the number of people using flights has increased significantly. It is difficult for airlines to maintain prices since prices change dynamically due to different conditions. That's why we will try to use machine learning to solve this problem. This can help airlines by predicting what prices they can maintain. It can also help customers to predict future flight prices and plan their journey accordingly.

Data was collected from yatra.com for the period of two weeks for economy, premium and business classes.

Analytical Problem Framing

The dataset has around 28830 rows and 10 columns. Using this dataset we will be training the Machine Learning models on 70% of the data and the models will be tested on 30% data.

There are no missing values in the dataset. However, we can expect outliers and unrealistic values for certain variables.

Below are the definition for each variable available on the dataset

Price	Price of the flight
airlines	Company name of the flight
stops	Number of stops to reach a destination
from_city	City of departure
to_city	Destination city
fclass	Flight class(Economy, Premium and Business)
departure	Departure time
Journey date	Date of journey
duration	Flight duration

Importing the necessary libraries and looking at the glimpse of the data

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder,power_transform
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV
import matplotlib.pyplot as plt
import seaborn as sns
```

```
dataset = pd.concat([d1,d2,d3,d4,d5,d6,d7], axis = 0, ignore_index = True)
dataset.head()
```

	Unnamed: 0	airlines	departure	duration	stops	from_city	to_city	price	journey_date	fclass
0	0	Go First	19:45	2h 35m	Non Stop	New Delhi	Bangalore	7424	2021-10-28	Economy
1	1	Go First	22:45	9h 00m	1 Stop	New Delhi	Bangalore	7424	2021-10-28	Economy
2	2	Go First	21:30	10h 15m	1 Stop	New Delhi	Bangalore	7424	2021-10-28	Economy
3	3	Vistara	20:40	2h 40m	Non Stop	New Delhi	Bangalore	7425	2021-10-28	Economy
4	4	SpiceJet	21:50	2h 45m	Non Stop	New Delhi	Bangalore	7425	2021-10-28	Economy

Pre-Processing:

Before we can proceed with the analysis, we have to extract features from the scraped data. Features like day of week, sessions in a day (i.e., morning, afternoon, night and so on), month, day, departure hour, departure minute and total duration in minutes.

```
dataset['hour'] = dataset['duration'].str.split(' ', expand = True)[0]
dataset['minute'] = dataset['duration'].str.split(' ', expand = True)[1]
```

```
dataset['hour'] = pd.to_numeric(dataset['hour'].str.replace('h',''))
dataset['minute'] = pd.to_numeric(dataset['minute'].str.replace('m',''))
```

```
dataset['duration_in_min'] = (dataset['hour']*60)+dataset['minute']
```

I'm converting the duration to minutes by multiplying 60 to the hour and adding the minutes to it, finally the column duration will be in minutes.

Splitting the departure column into departure hour and minute using ':' as reference

```
dataset['dep_hour'] = dataset['departure'].str.split(':', expand = True)[0]
dataset['dep_minute'] = dataset['departure'].str.split(':', expand = True)[1]
```

After extraction, I have dropped the original feature from which I have the necessary information extracted.

```
dataset = dataset.drop(columns = ['departure','duration'])
```

```
dataset.drop(columns = ['hour','minute'])
```

Extracting features from the journey date function. Features like, day of week, year, month and day.

```
dataset['journey_date'] = pd.to_datetime(dataset['journey_date'])
dataset['day_of_week'] = dataset['journey_date'].apply(lambda time: time.dayofweek)
```

```
dataset['day'] = dataset['journey_date'].apply(lambda time: time.day)
dataset['month'] = dataset['journey_date'].apply(lambda time: time.month)
dataset['year'] = dataset['journey_date'].apply(lambda time: time.year)
```

Dropping the journey date feature post extraction

```
dataset = dataset.drop(columns = 'journey_date')
```

Further, extracting sessions in a day (i.e., morning, afternoon, night and so on) from departure hour (dep_hour) and looking at the glimpse of data after the feature engineering process.

```
dataset['dep_hour'] = dataset['dep_hour'].astype(int)
dataset['dep_minute'] = dataset['dep_minute'].astype(int)
```

```
def f(x):
    if (x > 4) and (x <= 8):
        return 'Early Morning'
    elif (x > 8) and (x <= 12 ):
        return 'Morning'
    elif (x > 12) and (x <= 16):
        return'Noon'
    elif (x > 16) and (x <= 20) :
        return 'Evening'
    elif (x > 20) and (x <= 24):
        return'Night'
    elif (x <= 4):
        return'Late Night'
```

```
dataset['session'] = dataset['dep_hour'].apply(f)
dataset.head()
```

	airlines	stops	from_city	to_city	price	fclass	hour	minute	dep_hour	dep_minute	day_of_week	duration_in_min	day	month	year	session
0	Go First	Non Stop	New Delhi	Bangalore	7424	Economy	2	35	19	45	3	155	28	10	2021	Evening
1	Go First	1 Stop	New Delhi	Bangalore	7424	Economy	9	0	22	45	3	540	28	10	2021	Night
2	Go First	1 Stop	New Delhi	Bangalore	7424	Economy	10	15	21	30	3	615	28	10	2021	Night
3	Vistara	Non Stop	New Delhi	Bangalore	7425	Economy	2	40	20	40	3	160	28	10	2021	Evening
4	SpiceJet	Non Stop	New Delhi	Bangalore	7425	Economy	2	45	21	50	3	165	28	10	2021	Night

I can see that the features like airlines, stops, from_city, to_city, fclass and session are of categorical type data.

Therefore I'm using ordinal encoder to convert the categorical data to numeric.

```
enc = OrdinalEncoder()
for col in dataset:
    if dataset[col].dtypes == 'object':
        dataset[col] =enc.fit_transform(dataset[col].values.reshape(-1,1))
```

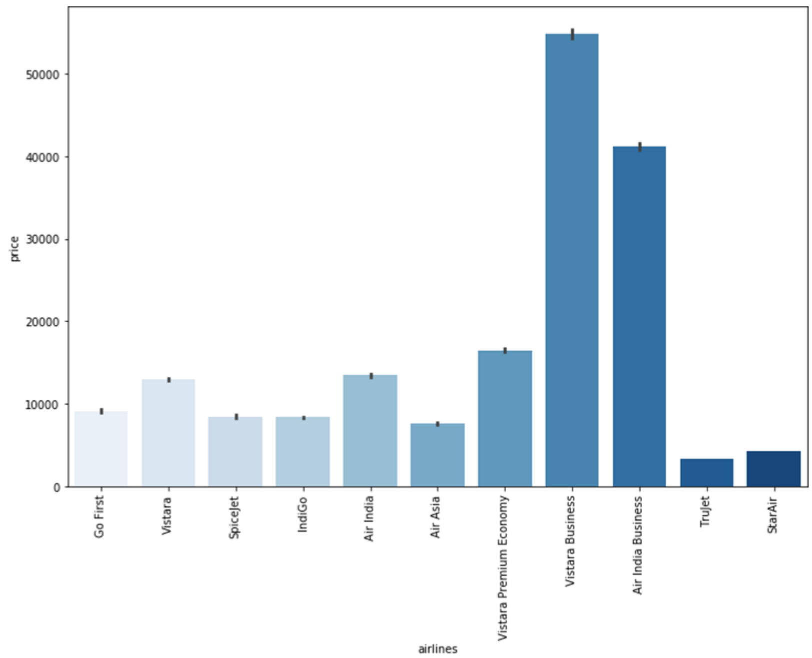
Now we have extracted all the necessary features and we have converted the categorical data to numeric. Let's check for the data types of the available feature

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28830 entries, 0 to 28829
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   airlines          28830 non-null  float64
 1   stops             28830 non-null  float64
 2   from_city         28830 non-null  float64
 3   to_city           28830 non-null  float64
 4   price             28830 non-null  int64  
 5   fclass            28830 non-null  float64
 6   dep_hour          28830 non-null  int32  
 7   dep_minute        28830 non-null  int32  
 8   day_of_week       28830 non-null  int64  
 9   duration_in_min   28830 non-null  int64  
 10  day               28830 non-null  int64  
 11  month             28830 non-null  int64  
 12  year              28830 non-null  int64  
 13  session           28830 non-null  float64
dtypes: float64(6), int32(2), int64(6)
memory usage: 2.9 MB
```

Now that we have done the pre-processing part. We can explore the data and its relationship with the target variable.

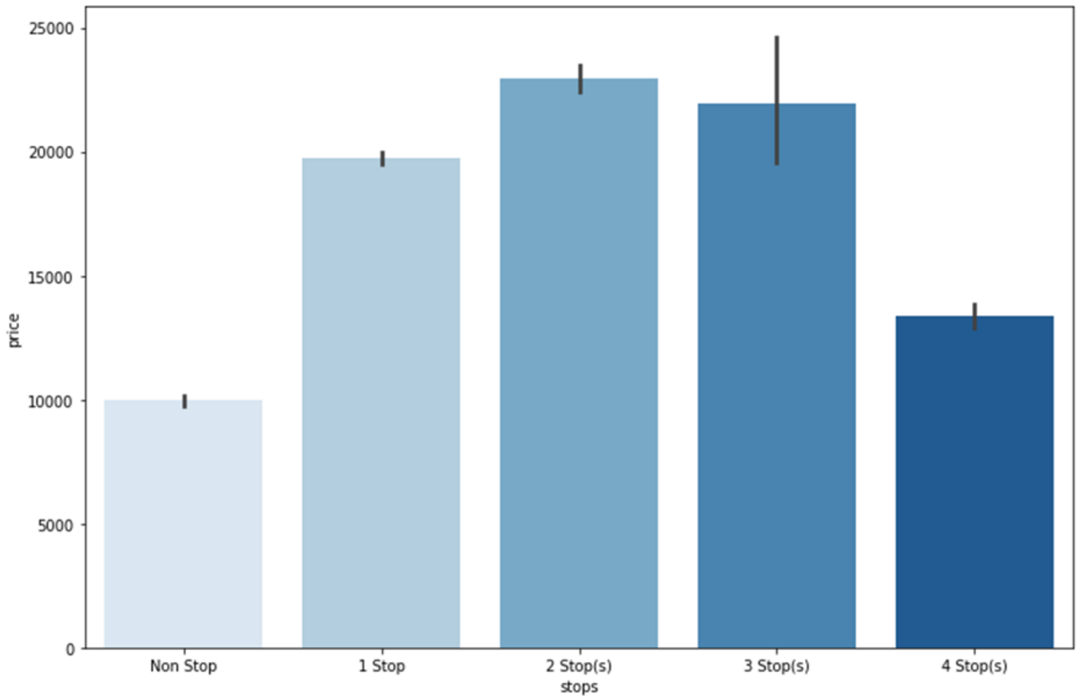
Firstly we'll visualize the relationship between the dependent variable and independent variables. I'm using seaborn library to visualize the same

- airlines v/s price



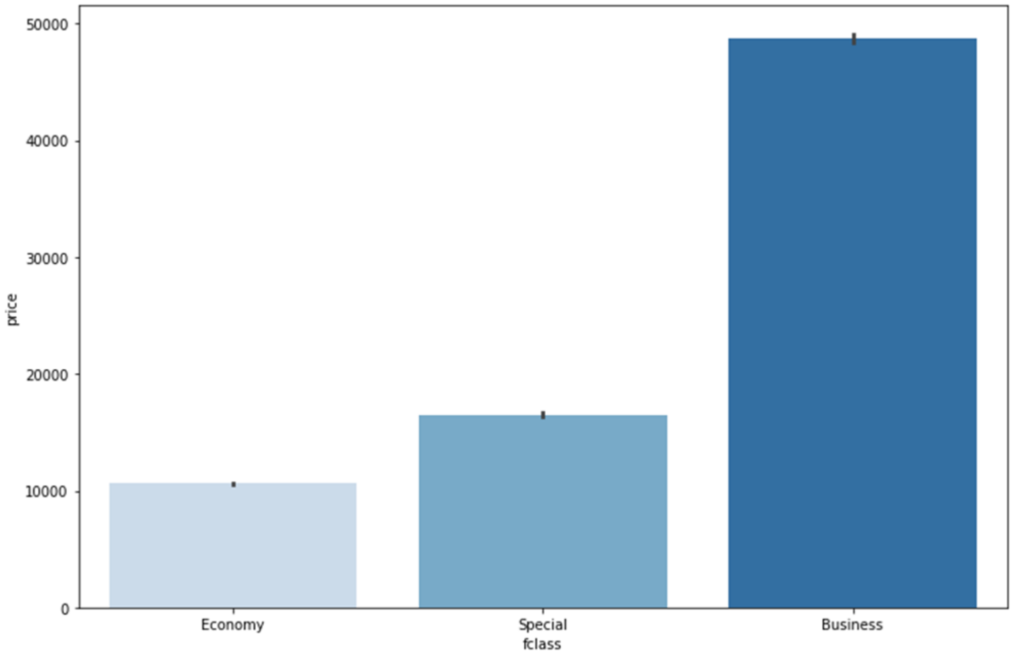
Vistara Business is costlier than all the other airlines and Trujet is the cheapest

- stops v/s price



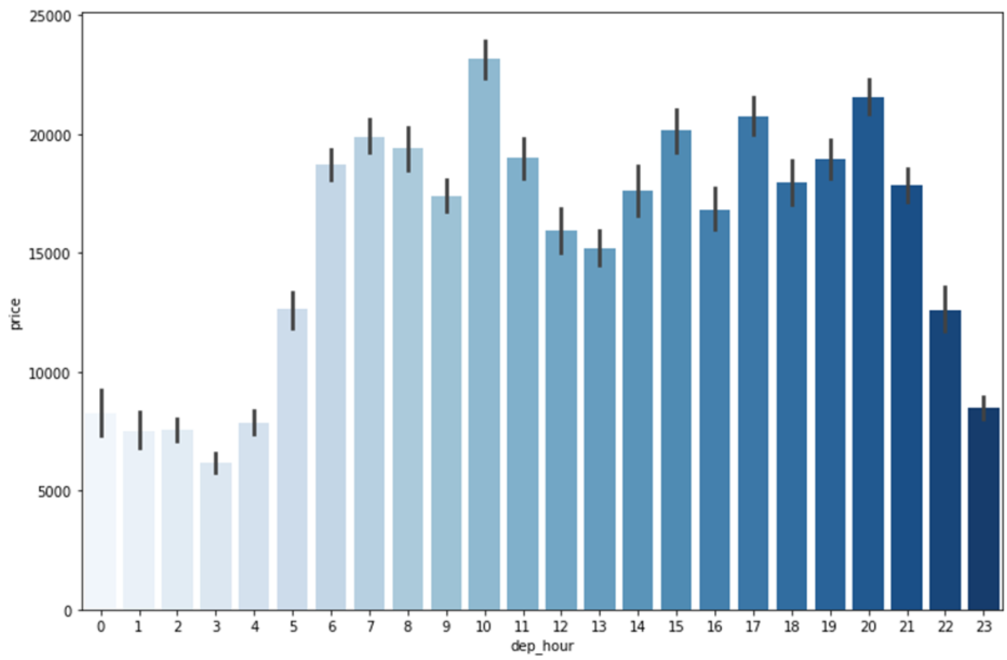
We can see that the nonstop flights and flights with more than 3 stops were cheaper when compared to flights with 1, 2 or 3 stops before reaching the destination

- class v/s price



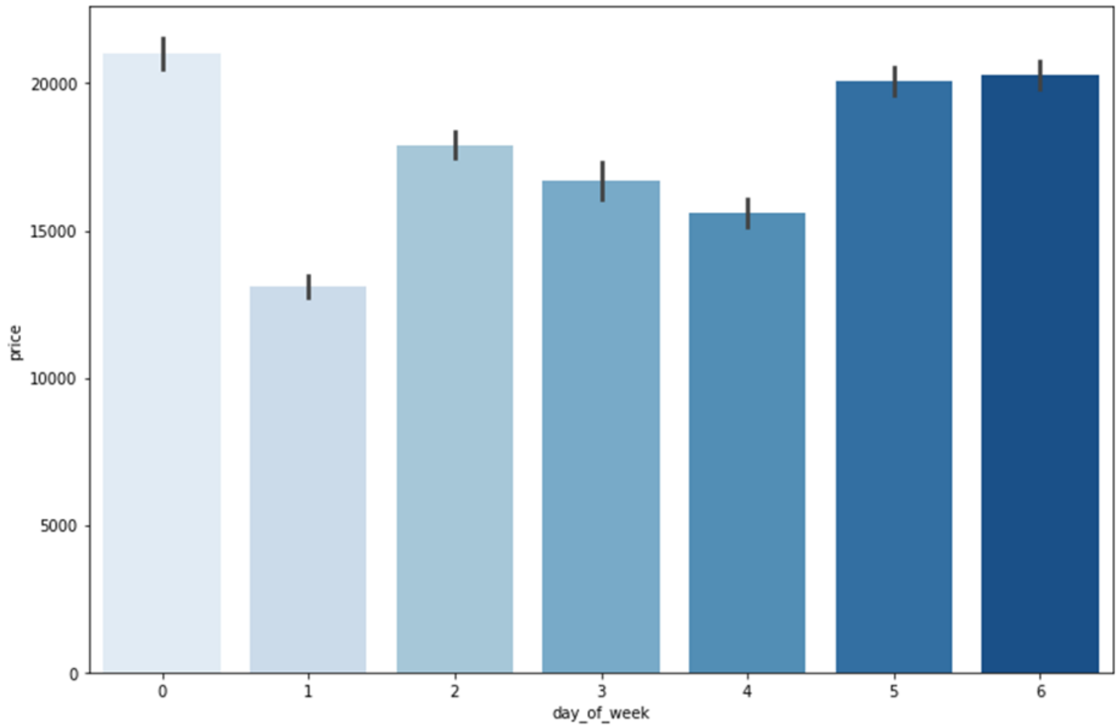
Business class flights were costlier when compared to other classes

- departure hour v/s price



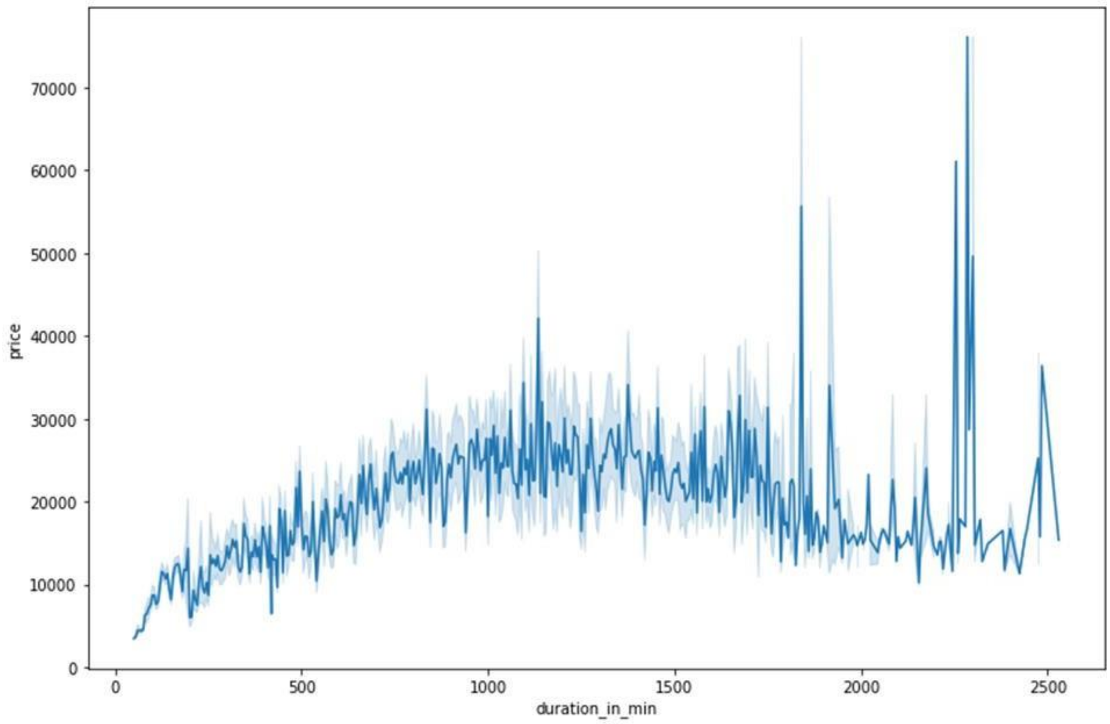
The flight price were cheaper between 11 PM to 4 AM and costlier during other time periods in a day

- day of week v/s price



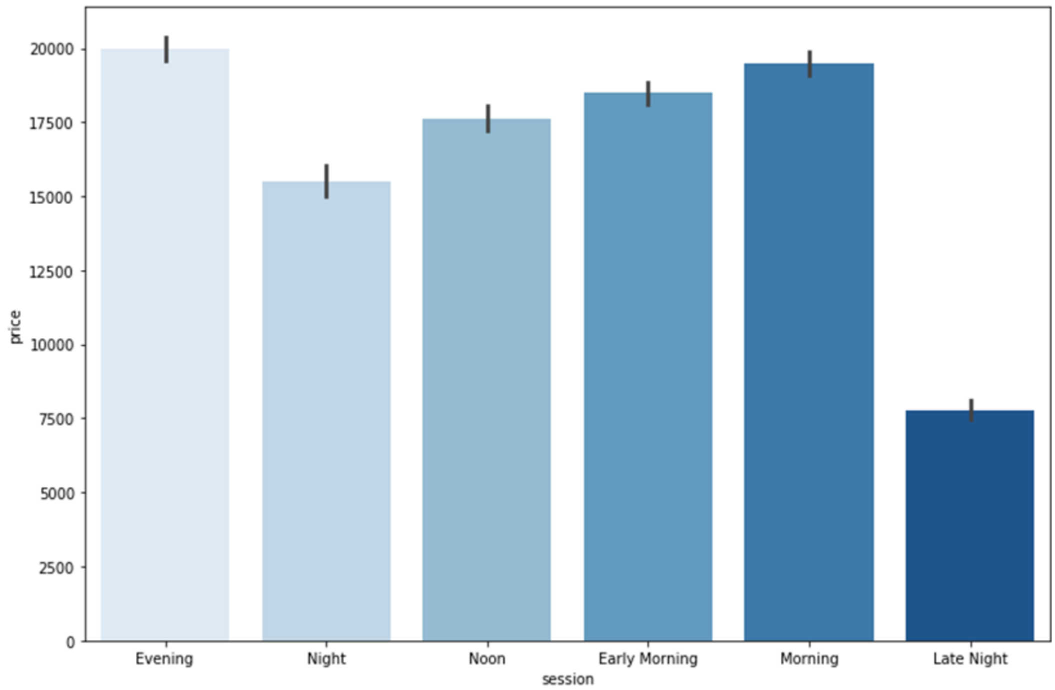
The flight charges were costlier during weekends (i.e., Friday, Saturday and Sunday) and cheaper on Mondays

- duration in minutes v/s price



The duration of flight is also related to the price, here when the flight duration increases, the price also increases.

- session in a day v/s price



The evening flights were costlier and late night price were cheaper

We can see from this correlation coefficient table that the highly correlated variables are fclass, duration_in_min and airlines

Further there are certain variables that do not show any correlation with the target. Hence removing these variables as they will not be useful in flight price prediction.

```
corr_data = dataset.corr()
corr_data['price'].sort_values(ascending = False)
```

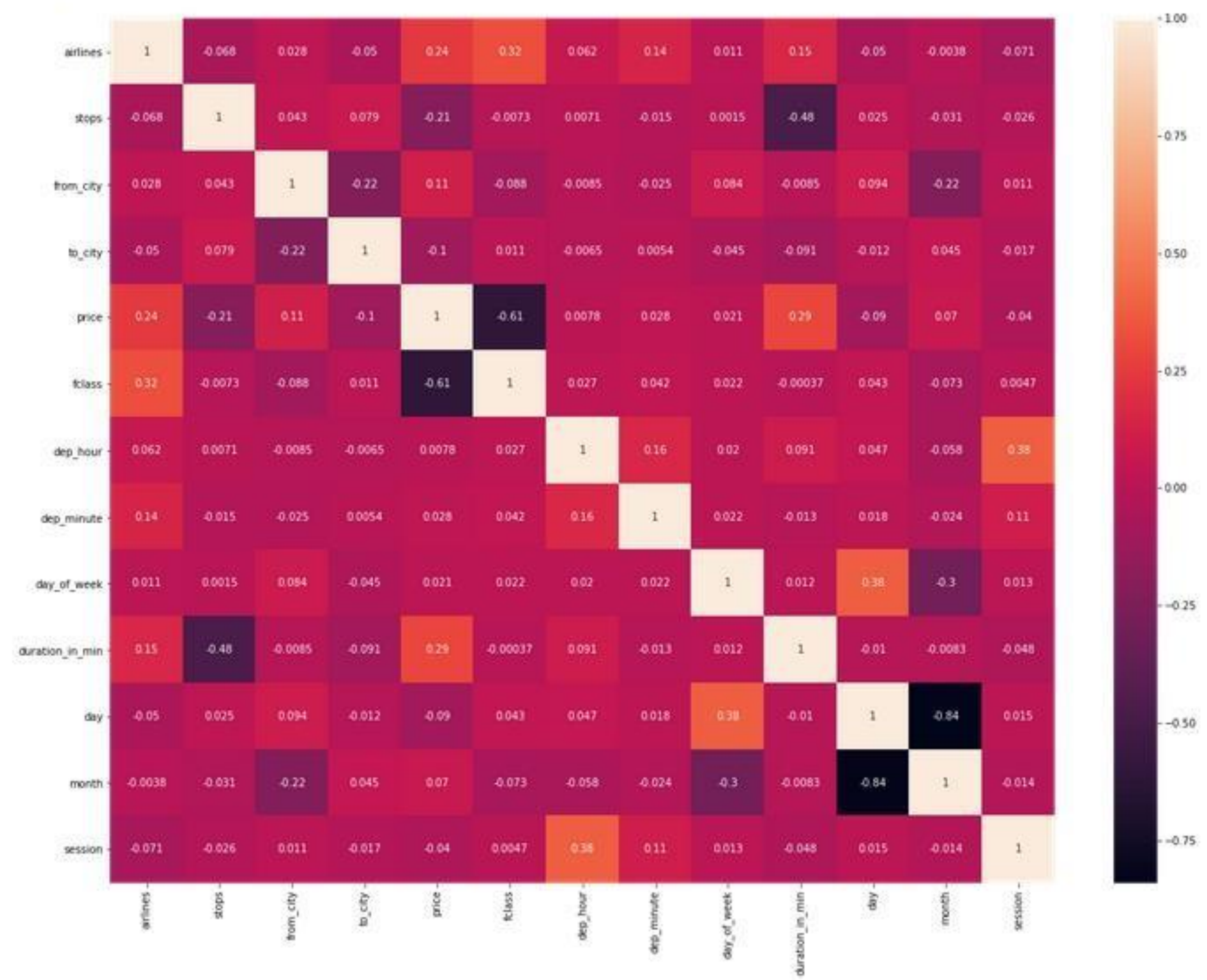
```
price              1.000000
duration_in_min    0.293467
airlines           0.238363
from_city          0.112362
month              0.069922
dep_minute         0.028491
day_of_week        0.021383
dep_hour           0.007818
session           -0.039702
day               -0.089516
to_city           -0.101130
stops             -0.214317
fclass            -0.609827
year                    NaN
Name: price, dtype: float64
```

```
dataset = dataset.drop(columns = 'year')
```

Now we can proceed further in checking for multi-collinearity in the dataset. In order to achieve that we are plotting a heatmap using the correlation table

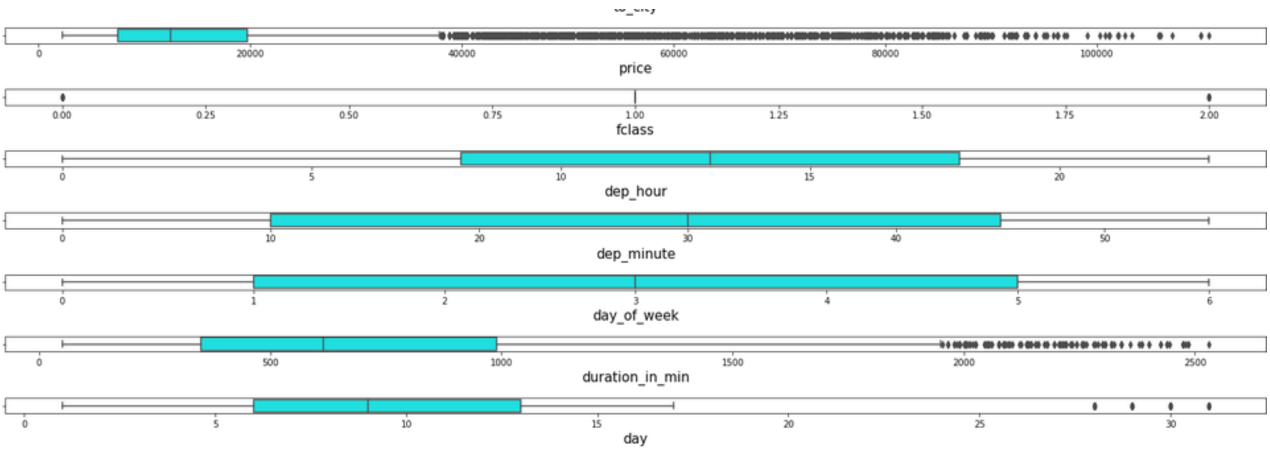
```
cor = dataset.corr()
plt.figure(figsize = (20,15))
sns.heatmap(cor, annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x275cb131da0>
```



Upon reviewing, I can see that there are few independent variables which are highly correlated with each other. However, I’m not removing them at this stage because multi-collinearity in a dataset will not affect the prediction in any manner.

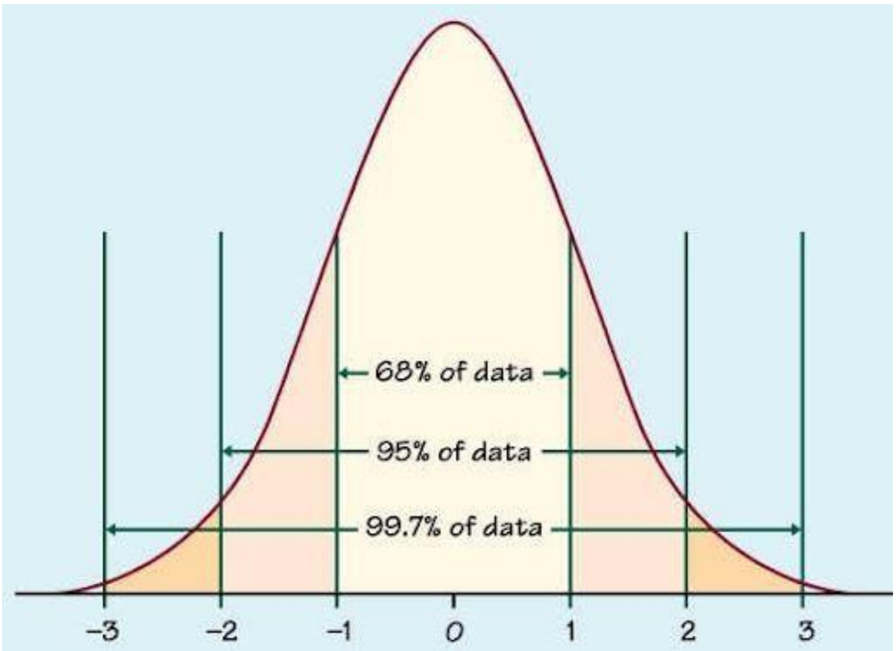
Now I’m proceeding with further analysis and checking for outliers in the continuous data variables within the dataset using boxplot



I can see that there are lot of outliers in the dataset. Hence I'm proceeding with handling the outliers with the z-score method.

It is assumed that close to 99.7% data lies between -3 to +3 standard deviation. We can consider the remaining data (0.03) to be outlier.

Therefore using the z-score method, I'm taking the data within the range of -2.5 to +2.5 standard deviation to control the outlier.



```
from scipy.stats import zscore
z = np.abs(zscore(dataset[['duration_in_min','price']]))
z.head()
```

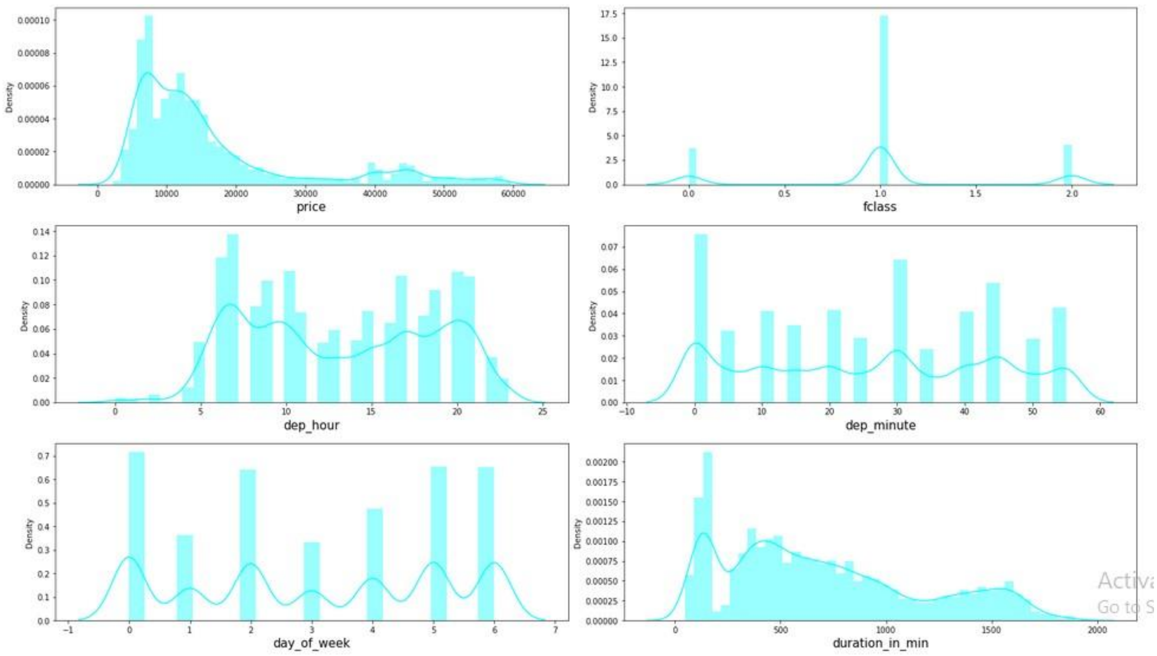
	duration_in_min	price
0	1.175061	0.670644
1	0.362754	0.670644
2	0.204512	0.670644
3	1.164512	0.670583
4	1.153962	0.670583

```
data_new = dataset[(z<2.5).all(axis =1)]
print(data_new.shape)
print(dataset.shape)
```

```
(27499, 13)
(28830, 13)
```

Post outlier removal I can see that the "new_data" dataset contains 27499 rows out of the actual dataset consisting of 28830 rows. If we are to proceed with outlier then there should be only 7 to 8 % data loss and we are losing close to 8% data loss therefore we are proceeding with the outlier removal.

Now let's check the data distribution to understand the data.



From the above distribution, we can clearly see that continuous data variables does contain outliers.

In order to be a good dataset, we assume that all the continuous variables follow normal distribution. However I can see that the continuous columns are skewed and we will have to control skewness to make data follow normal distribution.

Skewness coefficient:

```
x.skew()

airlines            0.068872
stops               1.483352
from_city          -0.587191
to_city            -0.069740
fclass              0.004635
dep_hour            0.044502
dep_minute          0.021164
day_of_week        -0.071076
duration_in_min     0.613722
day                 1.334588
month              -2.607424
session             0.249862
dtype: float64
```

It is assumed that the skewness co-efficient within the range of -0.5 to +0.5 is acceptable. Proceeding with the same assumption to get the skewness under control.

In order to perform the same we are using power transformation using cube-root transformation on the entire dataset excluding the target variable.

Once performed, most of the skewness are under control except few and we are proceeding with the model building assuming that outliers is not the cause of the skewness in the dataset.

Skewness co-efficient post transformation:

```
x.skew()

airlines           -0.179469
stops               0.909934
from_city          -0.396173
to_city            -0.210764
fclass              0.005818
dep_hour           -0.111341
dep_minute         -0.392582
day_of_week        -0.236084
duration_in_min    -0.099181
day                -0.009763
month              -2.607424
session            -0.099549
dtype: float64
```

Assumptions:

1. We assume that all the variables follow normal distribution
2. The multicollinearity in the dataset will not affect the prediction in any manner

Hardware and Software Requirements and Tools Used:

- Python version 3
- Jupyter interactive notebook
- Windows 10 professional
- Sci-kit learn Library
- Sci-py Library
- Seaborn Library
- Matplotlib Library
- Intel-core i3 processor
- 4GB RAM and 500 MB ROM
- Snipping tool

Model/s Development and Evaluation

Further, before build the model we will have to split the data to test and train. The best possible way to split the data is by finding the best random state to split and the benefit is that we can control over fitting up to certain extent before even building the model.

We are trying to match the R2 score of the training data set and the test dataset, which ever split (**random state**) satisfies the condition (**r2 score of training dataset = r2 score of testing dataset**). We'll take the same random state to split the dataset and build the model.

We are using a simple for loop to achieve the same.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
rs = 0
for i in range(0,2000):
    x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = i, test_size = 0.3)
    lr = LinearRegression()
    lr.fit(x_train,y_train)
    tr_score = lr.score(x_train,y_train)
    ts_score = lr.score(x_test,y_test)
    if round(tr_score*100,1) == round(ts_score*100,1):
        if i> rs:
            rs = i
print('the best random state is', rs)
```

```
the best random state is 1990
```

Now, I can say that the best random state for the split is 1990 and we will be splitting the dataset 70% train and 30% test with the random state 1990.

I'm testing the results with the below algorithms.

1. Linear Regression
2. Random Forest Regressor
3. Extra Trees Regressor
4. XG Boost Regressor

In order to test the model, I'm using r2 score and RMSE (Root Mean Squared Error), further in order to verify the model's fit, I'm using cross val score to identify the best model.

Model 1: Linear Regression

The first Machine Learning model I'm using to predict the Sale price is Linear Regression, this gives us with better understanding of the dataset and it's a simple model to build.

```
lin = LinearRegression()
lin.fit(x_train,y_train)
lin_pred = lin.predict(x_test)
lin_score = lin.score(x_test,y_test)
lin_score
```

```
0.566219757874286
```

```
lin_rmse = np.sqrt(mean_squared_error(y_test,lin_pred))
print('RMSE for Linear Regression: ', lin_rmse)
```

```
RMSE for Linear Regression:  8379.761670537326
```

Using the Linear Regression, we were able to get the r2 score of 0.57 and the RMSE of 8379.76

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and check for overfitting.

```
cv = cross_val_score(lin,x,y,scoring ='r2', cv = 5)
cv = cv.mean()
cv
```

```
0.4763581909091602
```

Model 2: Random Forest Regressor

Here, I'm using ensemble techniques to predict the outcome and I'm using Random Forest Regressor.

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(x_train,y_train)
rfr_pred = rfr.predict(x_test)
rfr_score = rfr.score(x_test,y_test)
rfr_score
```

```
0.9511317551434585
```

```
rfr_rmse = np.sqrt(mean_squared_error(y_test,rfr_pred))
print('RMSE for Random Forest Regression: ', rfr_rmse)
```

```
RMSE for Random Forest Regression:  2812.6141298626967
```

Here I can see that the Random Forest Regressor, is predicting the Sale Price with r2-score of 0.95 and the RMSE of 2812.61

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and check for overfitting.

```
cv1 = cross_val_score(rfr,x,y,scoring ='r2', cv = 5)
cv1 = cv1.mean()
cv1
```

```
0.8542381132105195
```

Model 3: Extra Trees Regressor

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

```
from sklearn.ensemble import ExtraTreesRegressor
et = ExtraTreesRegressor()
et.fit(x_train,y_train)
et_pred = et.predict(x_test)
et_score = et.score(x_test,y_test)
et_score
```

```
0.9563939816541226
```

```
et_rmse = np.sqrt(mean_squared_error(y_test,et_pred))
print('RMSE for Extra Trees Regression: ', et_rmse)
```

```
RMSE for Extra Trees Regression:  2656.868135110867
```

Here I can see that the Extra Trees Regressor is predicting the sale price with r2-score of 0.96 and the RMSE of 2656.86

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and check for overfitting.

```
cv2 = cross_val_score(et,x,y,scoring ='r2', cv = 5)
cv2= cv2.mean()
cv2
```

```
0.8680882586992162
```

Model 4: XG Boost Regressor

Extreme Gradient Boosting Algorithm. Gradient boosting refers to a class of ensemble machine learning algorithms that can be used for classification or regression predictive modelling problems. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models.

```
from xgboost import XGBRegressor
xgb = XGBRegressor()
xgb.fit(x_train,y_train)
xgb_pred = xgb.predict(x_test)
xgb_score = xgb.score(x_test,y_test)
xgb_score
```

```
0.9446356570952139
```

```
xgb_rmse = np.sqrt(mean_squared_error(y_test,xgb_pred))
print('RMSE for XGB Regression: ', xgb_rmse)
```

```
RMSE for XGB Regression:  2993.724685902187
```

Here I can see that the XG Boost Regressor, is predicting the sale price with F1-score of 0.95 and the RMSE of 2993.72.

Further, I'm verifying the fit using cross_val_score with cross validation of 5 and see that the model is not overfitting.

```
cv3 = cross_val_score(xgb,x,y,scoring ='r2', cv = 5)
cv3= cv3.mean()
cv3
```

```
0.8719888083254311
```

Finding the best model by subtracting the model's r2 score with the cross validation r2 score.

```
mod = [lin_score,rfr_score,et_score,xgb_score]
cv = [cv,cv1,cv2,cv3]
rmse = [lin_rmse,rfr_rmse,et_rmse,xgb_rmse]

model_sel = pd.DataFrame({})
model_sel['mod'] = mod
model_sel['cv'] = cv
model_sel['rmse'] = rmse
model_sel['diff'] = model_sel['mod'] - model_sel['cv']
model_sel
```

	mod	cv	rmse	diff
0	0.566220	0.476358	8379.761671	0.089862
1	0.951132	0.854238	2812.614130	0.096894
2	0.956394	0.868088	2656.868135	0.088306
3	0.944636	0.871989	2993.724686	0.072647

Here I can see that the Extra Trees model has highest accuracy and the highest r2-score. Further the model is also not overfitting.

Therefore, Extra Trees is the best Machine Learning model for the dataset. Therefore proceeding with the Hyper Parameter Tuning.

```
params = {'n_estimators':[150,200,250,350],
          'max_depth':[9,11,13,15],
          'min_samples_split':[3,4,6,8],
          'bootstrap':[True, False]
          }
```

```
gcv = GridSearchCV(ExtraTreesRegressor(),params,cv =5, n_jobs = -1)
gcv.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=ExtraTreesRegressor(), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_depth': [9, 11, 13, 15],
                         'min_samples_split': [3, 4, 6, 8],
                         'n_estimators': [150, 200, 250, 350]})
```

```
gcv.best_params_
```

```
{'bootstrap': False,
 'max_depth': 15,
 'min_samples_split': 3,
 'n_estimators': 350}
```

```
fnl_mod = ExtraTreesRegressor(bootstrap = False, max_depth = 15, min_samples_split = 3, n_estimators = 350,n_jobs =-1)
fnl_mod.fit(x_train,y_train)
fnl_pred = fnl_mod.predict(x_test)
fnl_score = fnl_mod.score(x_test,y_test)
print(' The R2 score for the hyper tuned model is', fnl_score)
```

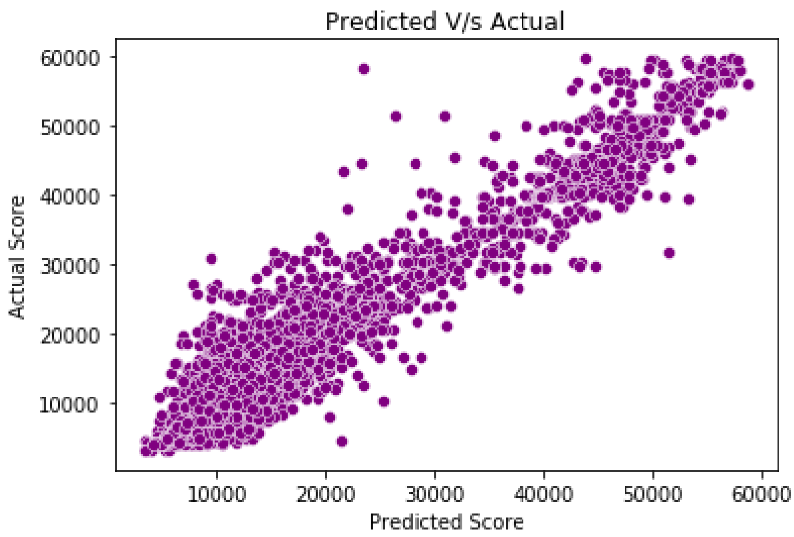
```
 The R2 score for the hyper tuned model is 0.9516051352789318
```

```
fnl_rmse = np.sqrt(mean_squared_error(y_test,fnl_pred))
print('RMSE for KNeighbors Regression: ', fnl_rmse)
```

```
RMSE for KNeighbors Regression:  2798.958270791175
```

Performing the hyper parameter tuning doesn't improve the scores, the Key Metric used to finalize the model was RMSE and R2-Score. And the Extra Trees is the best model at predicting the sale price of the used cars.

```
sns.scatterplot(x = fnl_pred, y = y_test, color = 'purple')
plt.xlabel('Predicted Score')
plt.ylabel('Actual Score')
plt.title('Predicted V/s Actual')
plt.show()
```



Conclusion

We have successfully built a model using multiple models, we found that the Extra Trees Regressor model and performed hyper parameter tuning on the same. Below are the best parameters

```
gcv.best_params_
```

```
{'bootstrap': False,
 'max_depth': 15,
 'min_samples_split': 3,
 'n_estimators': 350}
```

Below are the details of the model's metrics predicting the dataset

- R2- score of 0.95
- RMSE of 2799

Limitations of this work and Scope for Future Work.

- Due to unrealistic flight prices in the website, the error might be higher for certain regions and duration of flight. For example, We can see that Bangalore to Goa flights are in the range of 5000 to 6000 and for the same date and flight there are prices greater than 10000
- Due to this there might be good amount of difference than expected in the future prediction in a new dataset.

Other than these above limitations, I couldn't find more scope for improvement.