FLIGHT PRICE PREDICTION PROJECT

Submitted by:

SURAJ CHAKRABORTY

ACKNOWLEDGMENT

First and foremost, praises and thanks to the God, the Almighty, for his Shower of blessings throughout my project work to complete the project successfully.

I would like to express my deep and sincere gratitude to my SME Miss. Sapna Verma of Flip Robo technology, for giving me the opportunity to make this project and providing guidance on how to make the project. It was a great privilege to work under his mentorship. I would also like to thank my datatrained institute mentors for giving me all the knowledge in data science, which I am able to implement now for making the project. The projects which I had made while learning with datatrained helped me a lot in providing reference to make this project. The files attached with the data file of this project also helped me to get a better understanding of the project and steps that I needed to follow. Also the internet played a key role in helping me make the project as whenever I got stuck, I looked up on the internet for possible solutions that I could use and also to get rid of errors.

I am extremely grateful to my mother for her love, prayers and care and sacrifices for educating and preparing me for my future.

INTRODUCTION

- ## Business Problem Framing

  Booking a flight nowadays has become very unpredictable with respect to the fare of the flights. Also with the covid 19 impact on the transport industry, it has become more challenging to keep up with the fluctuating prices of fares of the flights. Anyone who has booked a flight ticket knows how unexpectedly the prices of tickets 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)

  With an aim to increase their profits the airlines play these kinds of tricks very often, making the customers clueless on what the fare might be. So here, we have to work on this project where we need to collect data of flight fares along with other features and work to make a model to predict fares of flights.

- ## Conceptual Background of the Domain Problem

  With the prices of flight tickets varying very unexpectedly and because of which the problems that the passengers are facing, we are assigned with the project where we have to collect data of flight fares along with other features and make a model using those to predict fares of flights.

  The project consists of three phases, the data collection phase, the data analysis phase and the model building phase. In the data collection phase, we have to scrape at least 1500 records of data. In

this section we need to scrape the data of flights from different websites.

After collecting the data, we have to do some analysis on them. Do airfares change frequently? Do they move in small increments or in large jumps? Do they tend to go up or down over time? What is the best time to buy so that the consumer can save the most by taking the least risk? Does price increase as we get near to departure date? Are morning flights expensive? We need to analyse the data and find answers to all these queries.

After analysing the data and finding the answers, we need to build a machine learning model, and before model building we need to do all data pre-processing steps then try different models with different hyper parameters and select the best model.

- Review of Literature

While scraping for data I got many insights into the aviation sector regarding its ticket fares which helped me with my project a lot. I have also received a lot of important information regarding the topic form the internet. And I found some articles regarding the topic in different websites like towardsdatascience.com, kaggle.com, medium.com etc. Those articles helped me get a better understanding of the use case and how to approach and deal with the problems. And also the project description provided to us helped me understand the steps I needed to follow in order to complete the project and get a better model as a result.

- Motivation for the Problem Undertaken

The objective of the project is come up with the best model, which can be used to get an understanding on the aviation industry regarding their fares and how exactly the fare prices vary with respect to the features. With this knowledge the passengers can decide when and how to buy tickets in order to get the best deals on the fares and that they can save some money.

What motivated me to take the project is that the project is based on the aviation sector for which I have a keen interest. I also got motivated by the fact that the project is a fully fledged project that involves scraping the data, analysing the data and creating the ML model using the data. And also the fact that we get to decide the features we want and don't want for the project and scrape according to that, got me more involved into the project and gave me more knowledge on selecting feature while scraping data.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

  In this project, we build a regression model which can be used to predict the fares of flight transportation based on different features. We used regression as the target variable contained continuous numeric data. We performed different steps including exploratory data analysis, data cleaning, data analysis, data pre-processing and model building. We used different models for training, checked their CV scores and also determined the optimal values for different Hyper Parameters and selected the final model based on their scores.

- Data Sources and their formats

  The project consisted of three phases and in the first phase we had to scrape the data for different flights from different websites and hence those websites acted as the data sources for our project. We

preferred using multiple websites for collecting the data such as yatra.com, makemytrip.com, etc. as it would help our model in avoiding over fitting problem. We scraped different types of features for the flights such as date, airline, duration, stops, etc. Some of these were string data and some numeric data and the target i.e. price was of integer data type which contained the fare of the flights.

- Data Pre-processing Done

  - At first we scraped the data using a jupyter notebook and saved the dataset in an excel sheet. And then we loaded the dataset into a new jupyter notebook.

  - We dropped the unwanted column from the dataset which was the index column that had no role to play.

```
df.drop(['Unnamed: 0'], axis=1, inplace=True)
```

  - Checked for missing values in the dataset and we found no missing data.

```
# Checking for null values.

df.isnull().sum()
```

```
Airline          0
Date             0
Source           0
Destination      0
Departure_time   0
Arrival_time     0
Duration         0
Stops            0
Price            0
dtype: int64
```

Screenshot of the code for first column

  - We checked the unique values for all the columns in the dataset to find the data that needed cleaning.

```
# checking the unique values of each columns.

for i in range(len(df.columns)):
    print(df.columns[i])
    print(df[df.columns[i]].unique())
```

```
Airline
['Air India' 'Air Asia' 'Vistara' 'SpiceJet' 'IndiGo' 'Go First'
 'AirAsia India' 'AirIndia' 'GO FIRST' 'TruJet']
Date
['9 Oct' '10 Oct' '12 Oct' '14 Oct' '16 Oct' '18 Oct' '20 Oct' '24 Oct'
 'Oct 10 2021' 'Oct 11 2021' 'Oct 12 2021' 'Oct 13 2021' 'Oct 14 2021'
 'Oct 15 2021' 'Oct 16 2021' 'Oct 17 2021' 'Oct 18 2021' 'Oct 19 2021'
 'Oct 20 2021' 'Oct 21 2021' 'Oct 22 2021' 'Oct 23 2021' 'Oct 24 2021'
 'Oct 25 2021']
Source
['New Delhi' 'Mumbai' 'Bangalore' 'Chennai' 'Hyderabad' 'Goa' 'Kolkata'
 'Pune' 'Jaipur' 'Delhi']
Destination
['Mumbai' 'Bangalore' 'Chennai' 'Hyderabad' 'Goa' 'Kolkata' 'Pune'
 'Jaipur' 'Lucknow' 'Delhi' 'Patna' 'Ahmedabad']
Departure_time
['04:55' '05:20' '05:35' '05:45' '06:00' '06:05' '06:10' '06:15' '06:50'
 '07:00' '07:10' '07:20' '07:30' '07:40' '08:00' '08:10' '06:35' '07:15'
 '05:00' '06:20' '05:55' '06:45' '06:55' '07:05' '07:25' '05:30' '07:50'
```

Screenshot of the code

- In the Airline column, we found some duplicate values which were written in different manner, we merged them by replacing one of every duplicate value with the other.

```
df["Airline"].replace("Air India", "AirIndia", inplace=True)
df["Airline"].replace("AirAsia India", "Air Asia", inplace=True)
df["Airline"].replace("GO FIRST", "Go First", inplace=True)
```

Screenshot of the code

We also found a value that that contained only 3 records of data and as this may affect the model, we dropped those 3 records.

```
df= df[df["Airline"].str.contains("TruJet")==False]
```

Screenshot of the code

- In the Date column, we found that all the dates belonged to the same month and same year, and hence we removed the month and year data from the date values and converted the date column to integer data type.

```
df['Date'] = df['Date'].str.replace(' 2021','')
df['Date'] = df['Date'].str.replace(' Oct','')
df['Date'] = df['Date'].str.replace('Oct ','')
```

```
df['Date'].unique()
```

```
array(['9', '10', '12', '14', '16', '18', '20', '24', '11', '13', '15',
       '17', '19', '21', '22', '23', '25'], dtype=object)
```

Now we can see that the feature contains only the date.

```
df['Date']= df['Date'].astype(int)
```

Screenshot of the code

- In the Source columns, we found two duplicate values which which were written in different manner, we merged them by replacing one of the values.

```
df["Source"].replace("New Delhi", "Delhi", inplace=True)
```

Screenshot of the code

- In the Departure time column, the data was in time format. Hence we first converted the column to dateandtime data type and then extracted the hour and minute data into two different columns and deleted the base column.

```
# converting object datatype to datetime.

df['Departure_time']= pd.to_datetime(df['Departure_time'])
```

```
# Extracting the hour and minute into two seperate columns.

df['Dep_hr']=df['Departure_time'].dt.hour
df['Dep_min']=df['Departure_time'].dt.minute
```

```
# dropping the base column.

df.drop(['Departure_time'], axis=1, inplace=True)
df
```

Screenshot of the code

- In the Arrival time column, we performed the same operation as Departure time column as the data was in the time format. We first converted the column to dateandtime data type and then extracted the hour and minute data into two different columns and deleted the base column.

```
# converting object datatype to datetime.

df['Arrival_time']= pd.to_datetime(df['Arrival_time'])
```

```
# Extracting the hour and minute into two seperate columns.

df['Arv_hr']=df['Arrival_time'].dt.hour
df['Arv_min']=df['Arrival_time'].dt.minute
```

```
# dropping the base column.

df.drop(['Arrival_time'], axis=1, inplace=True)
df
```

Screenshot of the code

- In the Duration column, the data was in hour and minute format, we first split the hours and minutes into two different columns and removed the unwanted strings from those columns and then converted the columns to integer data type, and finally we dropped the base column.

```
# extracting the hour and minute into another column by splitting.

df[['Dur_hr','Dur_min']] = df.Duration.str.split('h ',1, expand=True)
```

```
# cleaning the duration_minute column.

df['Dur_min'] = df['Dur_min'].str.replace('m','')
df['Dur_min'] = df['Dur_min'].str.replace(' ','')
```

```
# Converting the datatype to integer.

df['Dur_min']= df['Dur_min'].astype(int)
df['Dur_hr']= df['Dur_hr'].astype(int)
```

```
# dropping the base column.

df.drop(['Duration'], axis=1, inplace=True)
df
```

Screenshot of the code

- In the Stops column, we found some duplicate values which were due to the presence of some unwanted strings attached to the values. We first removed all the unwanted strings and characters from the data and then converted the column to integer data type.

```
df['Stops']= df['Stops'].str.replace(r" Stop\(.*\)","")
df['Stops']= df['Stops'].str.replace('Stop', '')
```

```
# Replacing 'Non-stop' flights with 0.

df['Stops']= df['Stops'].str.replace('Non-', '0')
df['Stops']= df['Stops'].str.replace('Non ', '0')
df['Stops']= df['Stops'].str.replace(' ', '')
```

```
df['Stops'].unique()
```

```
array(['1', '2', '0', '3', '4'], dtype=object)
```

We can see that the data has been cleaned but it still belongs to object datatype. lets correct this.

```
df['Stops']= df['Stops'].astype(int)
```

Screenshot of the code

- In our target column Price, we had punctuation in the data due to which the column was of object type, we removed the punctuation from the data and then converted the column to integer data type.

```
df['Price']= df['Price'].str.replace(',', '')
```

```
# Converting datatype to integet.

df['Price']= df['Price'].astype(int)
```

Screenshot of the code

- Visualized the distribution and count of the data values in different features of the dataset.

```
sns.set(style='darkgrid')
df.plot(kind='density', subplots= True, layout=(3,3),sharex=False, legend=True, figsize=[20,10])
plt.show
```

Screenshot of the code

- Encoded the categorical object type features in the dataset using ordinal encoder.

```
# Encoding

from sklearn.preprocessing import OrdinalEncoder

enc= OrdinalEncoder()
for i in df.columns:
    if df[i].dtypes== 'O':
        df[i]= enc.fit_transform(df[i].values.reshape(-1,1))
```

Screenshot of the code

- Checked the correlation between all the columns in the dataset along with the target and we dropped a column that showed no correlation with the target column. We also tried to find the answers to all queries using different visualization techniques.

```
cor= df.corr()
cor
```

Screenshot of one of the code

```
plt.figure(figsize=(15,5))
cor['Price'].sort_values(ascending=False).drop(['Price']).plot(kind='bar')
plt.xlabel('Feature')
plt.ylabel('Correlation with target')
plt.title('Correlation')
plt.show()
```

Screenshot of one of the code

- Checked for outliers in the numeric columns in the dataset. We found few outliers in one of the columns but they were very close to the threshold, hence to did not remove them.

```
df.iloc[:,6:].boxplot(figsize=[20,7])
plt.show()
```

Screenshot of the code

- Checked for skewness in the continuous data and we found some present in some of the columns, we them minimised the skewness for those columns using power transformation.

```
# treating skewness by taking the threshold of -0.5 to +0.5.

from sklearn.preprocessing import power_transform

for i in df[col]:
    if df[i].skew()>0.5:
        df[i]= power_transform(df[i].values.reshape(-1,1))

    if df[i].skew()<-0.5:
        df[i]= power_transform(df[i].values.reshape(-1,1))

df.skew()
```

Screenshot of the code

- Finally we split the dataset into features and target and then scaled the features using standard scaler.

```
# First lets split the data into target and features.

x= df.drop(['Price'], axis=1)
y= df['Price']
```

```
# Scaling

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
x=pd.DataFrame(sc.fit_transform(x), columns=x.columns)
x
```

Screenshot of the code

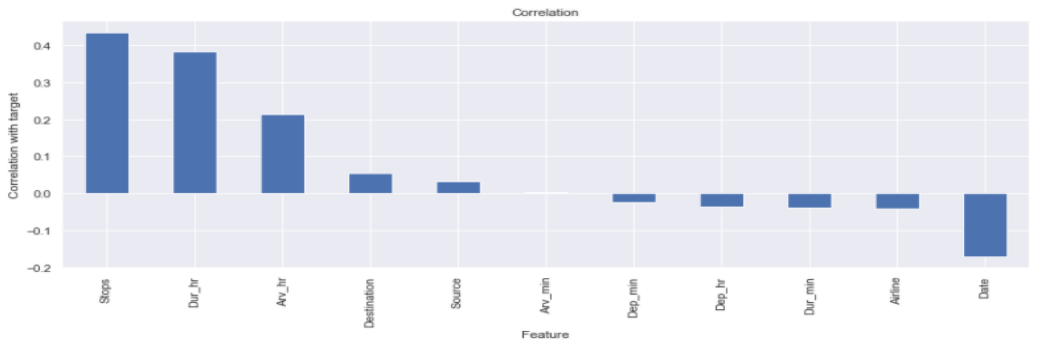
- Data Inputs- Logic- Output Relationships

We checked the distribution of data in the numeric independent columns and the counts in the categorical columns in the dataset and found some numeric columns having variance in the distribution and some having skewness in the data. We then checked the correlation among all the columns of the dataset and then visualized it.

We then visualized the correlation of each of the independent columns of the dataset with the target separately, where we found the feature ‘Stops’ and ‘Duration hr’ showing a very positive correlation with the target, which meant that more the number of stops and duration, the higher the prices are. Also we found the features ‘Date’ showing a high negative correlation with the target which meant that as we got closer to the departure date, the more costly the fare got.

The feature ‘Arrival min’ showed us no correlation with the target column, a near zero correlation was found between them and hence we dropped the feature.

```
plt.figure(figsize=(15,5))
cor['Price'].sort_values(ascending=False).drop(['Price']).plot(kind='bar')
plt.xlabel('Feature')
plt.ylabel('Correlation with target')
plt.title('Correlation')
plt.show()
```



Screenshot of correlation of target with all features

- ## Hardware and Software Requirements and Tools Used

Hardware required: Minimum Intel i3 processor with 8 GB of RAM.

Operating system: Windows 7, 8, or 10 or LINUX or Mac OS

Tools used:

- Jupyter notebook: For the coding and creating the project.
- Microsoft edge: For hosting.
- Chrome driver: For scrapping data from websites.
- Microsoft word: For making the project report.
- Microsoft PowerPoint: For making the PPT.

Libraries & Packages used:

- Numpy: Used to perform log transformation on the dataset for treating the skewness. Also used to find the exponent of the final predicted result to inverse the logarithm and round up the values.
- Pandas: Used to load the dataset into the jupyter notebook and controlling the viewing pattern of the data.
- Selenium: Used for controlling web browser through programs and performing browser automation.
    - Webdriver: Used for initializing the web driver to a variable.

    - Exceptions: Used to deal with any unwanted exception during scraping.
- Time: Used to halt the scrapping process for a certain time to let a page load successfully
- Matplotlib: Uses to visualize and plot the distribution of data in columns as well as the relation between different columns.
- Seaborn: Used to visualize the distribution and correlation between data using different types of figures and graphs.
- Sklearn: Used to perform different operations using different modules in it.
    - r2_score: To measure how close the predicted data are to the actual data.
    - mean_squared_error: Gives the average squared difference between the estimated values and true value.
    - mean_absolute_error: This measure the absolute average distance between the real data and the predicted data.
    - StandardScaler: Used to scale the dataset.
    - OrdinalEncoder: Used for encoding the object type data.
    - power_transform: Used to reduce the skewness in the columns.
    - Train_test_split: To split the data in train and test data.
    - Using sklearn we also imported different models for the training the data e.g. Linear Regression, Lasso etc.
    - Cross_val_score: Used to check the cross validation scores for different models.
    - RandomizedSearchCV: Used to perform hyper-parameter tuning to find the best parameters for the models.
- xgboost: Used to call the XGBRegressor model for training.
- Joblib: Used to save the model that we created in .pkl format.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  The approach that we followed is:

  - Loaded the dataset into a jupyter notebook
  - Performed EDA on the dataset to get better insights into the data.
  - Checked for missing values in the dataset.
  - Performed data cleaning by cleaning all junk data from the columns and converting all numeric columns to integer data type.
  - Performed visualization to check the distribution and the value counts of data in different columns.
  - Encoded the columns containing categorical data.
  - Checked the correlation between all the columns in the dataset.
  - Performed bivariate analysis to check the relationship between the features and the target and tried finding answers to different queries.
  - Checked for outliers in the dataset.
  - Checked for skewness and then handled the skewness in the dataset.
  - Split the dataset into target and features.
  - Performed scaling on the features.
  - Found the best random state and preformed train test split using that.
  - Trained different models using the train datasets and checked their scores on the test dataset.
  - Checked CV score for all the models for any overfitting or underfitting.
  - Hyper-parameter tuned the better performing models to find the best parameters for those models.

- Trained those models using the parameters found to be performing best during the hyper parameter tuning.
- Saved the best performing model as the final model.

- Testing of Identified Approaches (Algorithms)

  The algorithms that we used are:

  - Linear Regression
  - Decision Tree Regressor
  - Random Forest Regressor
  - Support Vector Regressor
  - Lasso regularization
  - XGBRegressor

- Run and Evaluate selected models

  - Linear Regression: It measures the relationship between continuous numeric dependent variable and the independent variables by estimating probabilities.

```
lr.fit(x_train,y_train)
predlr= lr.predict(x_test)

print('Score: ',lr.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predlr))
print('Mean absolute error:', mean_absolute_error(y_test,predlr))
print('Mean squared error:', mean_squared_error(y_test,predlr))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predlr)))
```

```
Score:  0.25895014660531457
r2 score:  0.3196552990001651
Mean absolute error: 2365.0665882032818
Mean squared error: 11440676.34430043
Root mean squared error: 3382.4068862720274
```

Screenshot of the code

Using linear regression, we found the training score to be 26%, the R2 score to be 32% and the errors being high.

- Decision tree regressor: Builds regression models in the form of a tree structure. It breaks down the dataset into smaller subsets.

```
from sklearn.tree import DecisionTreeRegressor

dt= DecisionTreeRegressor()
dt.fit(x_train,y_train)
preddt= dt.predict(x_test)

print('Score: ',dt.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,preddt))
print('Mean absolute error:', mean_absolute_error(y_test,preddt))
print('Mean squared error:', mean_squared_error(y_test,preddt))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,preddt)))
```

```
Score:  0.9988874940041893
r2 score:  0.7536678152554132
Mean absolute error: 701.5667808219179
Mean squared error: 4142321.965182649
Root mean squared error: 2035.2695067687348
```

Screenshot of the code

Using Decision tree regressor, we found the training score to be 99%, the R2 score to be 75%, and errors are less.

- Random forest regressor: It builds multiple decision trees and merges them together to get a more accurate prediction.

```
from sklearn.ensemble import RandomForestRegressor

fr=RandomForestRegressor()
fr.fit(x_train,y_train)
predfr=fr.predict(x_test)

print('Score: ',fr.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predfr))
print('Mean absolute error:', mean_absolute_error(y_test,predfr))
print('Mean squared error:', mean_squared_error(y_test,predfr))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predfr)))
```

```
Score:  0.9712505417803431
r2 score:  0.7974581764516215
Mean absolute error: 849.7857910433839
Mean squared error: 3405943.260814737
Root mean squared error: 1845.5197806620056
```

Screenshot of the code

Using Random forest regressor, we found the training score to be 97%, the R2 score to be 80%, and with less errors.

- Support vector regressor: It looks at data and sorts it into one of two categories. It helps in determining the closest match between the data points and the function which is used to represent them.

```
from sklearn.svm import SVR

svr= SVR(kernel='linear')
svr.fit(x_train,y_train)
preds= svr.predict(x_test)

print('Score: ',svr.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,preds))
print('Mean absolute error:', mean_absolute_error(y_test,preds))
print('Mean squared error:', mean_squared_error(y_test,preds))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,preds)))
```

```
Score:  0.1893517293107334
r2 score:  0.2528631394752313
Mean absolute error: 2407.0639032352387
Mean squared error: 12563853.284370162
Root mean squared error: 3544.552621187921
```

Screenshot of the code

Using support vector regressor, we found the training score to be 11%, the R2 score to be 25%, and with high errors.

- Lasso regularization: It uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models i.e. models with fewer parameters.

```
from sklearn.linear_model import Lasso

ls= Lasso()
ls.fit(x_train,y_train)
predrd= ls.predict(x_test)

print('Score: ',ls.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predrd))
print('Mean absolute error:', mean_absolute_error(y_test,predrd))
print('Mean squared error:', mean_squared_error(y_test,predrd))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predrd)))
```

```
Score:  0.2589496294467364
r2 score:  0.31968173343515505
Mean absolute error: 2365.027754207589
Mean squared error: 11440231.82284738
Root mean squared error: 3382.3411748147732
```

Screenshot of the code

Using Lasso regressor, we found the training score to be 25%, the R2 score to be 32%, and with high errors.

- XGB regressor: It is an efficient implementation of gradient boosting that can be used for regression predictive modelling.

```
from xgboost import XGBRegressor

xgb = XGBRegressor()
xgb.fit(x_train,y_train)
predx= xgb.predict(x_test)

print('Score: ',xgb.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predx))
print('Mean absolute error:', mean_absolute_error(y_test,predx))
print('Mean squared error:', mean_squared_error(y_test,predx))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predx)))
```

```
Score:  0.9550452101497624
r2 score:  0.7516614514340874
Mean absolute error: 1065.9408841067798
Mean squared error: 4176060.9787664535
Root mean squared error: 2043.5412838419618
```

Screenshot of the code

Using XGB regressor, we found the training score to be 95%, the R2 score to be 75%, with the errors being comparatively less.

- Key Metrics for success in solving problem under consideration

Key metrics used are:

- R2 score: R-squared is used to gives the measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination. In general, it is used to know the model performance. As, the higher the R-squared, the better the model fits the data.
- Mean absolute error: This is used because in statistics, the mean absolute error (MAE) is a way to measure the accuracy of a given model. This tells us that the average difference between the actual data value and the value predicted by the model. The lower the MAE for a given model, the more closely the model is able to predict the actual values.
- Mean squared error: The mean squared error (MSE) is used to determine the performance of the algorithm. The mean square error gives measure of the average of error squares

i.e. the average of the square of the difference between the observed and predicted values of a variable.

- Root mean squared error: The RMSE is used, as it is a good measure of accuracy. It is the square root of the mean of the square of all of the errors**.** Basically it gives the square root of the mean squared error.

- Visualizations

The plots that are used are:

- Heatmap:

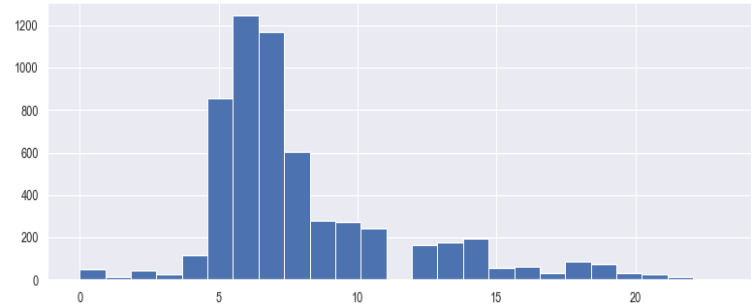
```
plt.figure(figsize=[15,7])
sns.heatmap(cor, annot=True, linewidth=0.1);
plt.title('Correlation matrix')
plt.show()
```

Screenshot of the plot

Heat map was used on two occasions in the project, once for checking any missing values in the dataset and the next time to plot the correlation matrix of the dataset. In the first case, we found no missing values in the dataset. And in the second case we found many types of correlation between the columns. There were some positive as well as negative relations in the matrix, and also some columns showed no correlation.

- Histogram:

```
plt.figure(figsize=[13,4])
plt.hist(df['Dep_hr'], bins=25)
plt.show()
```

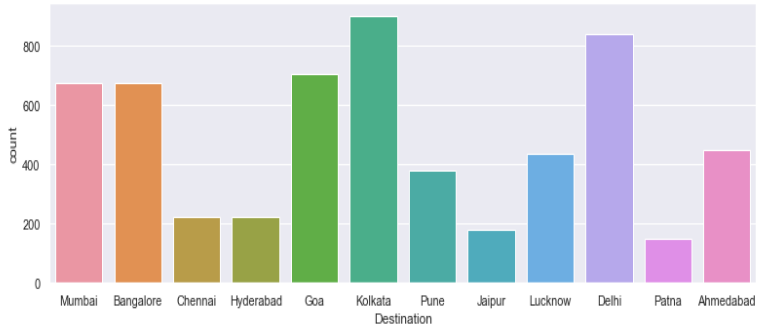


Screenshot of the plot

We used the histogram to see the distribution of data in the continuous numeric columns. We found some columns having skewness in the data and some had variance in their data.

- Count Plot:

```
plt.figure(figsize=[13,4])
sns.countplot(df['Destination'])
plt.show()
```

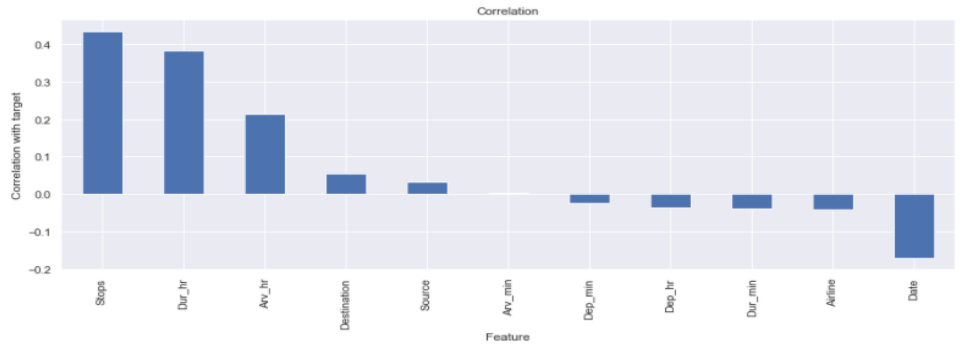


Screenshot of the plot

We used the count plot to check the count of values in different columns of the dataset. And we found some values being more, some being less and some having equally distributed values in different columns of the dataset.

- Bar graph:

```
plt.figure(figsize=(15,5))
cor['Price'].sort_values(ascending=False).drop(['Price']).plot(kind='bar')
plt.xlabel('Feature')
plt.ylabel('Correlation with target')
plt.title('Correlation')
plt.show()
```

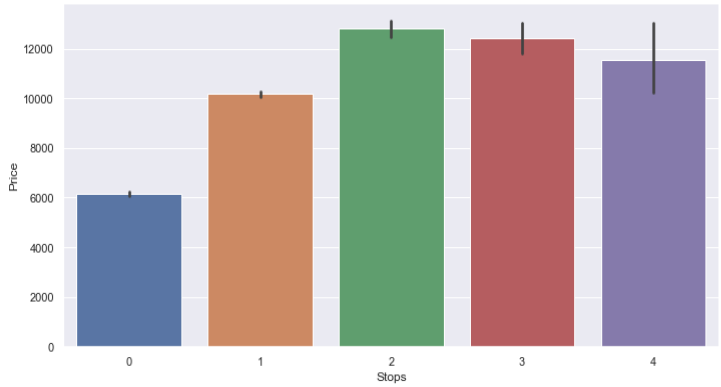


Screenshot of the plot

We plotted this bar graph to visualize the correlation of all the independent columns with the target column. We found some positive as well as negative corelation in between the columns, and also a column showed no correlation.

- Catplot:

```
sns.catplot(x='Stops',y='Price', data=df, kind='bar', aspect=2);
```

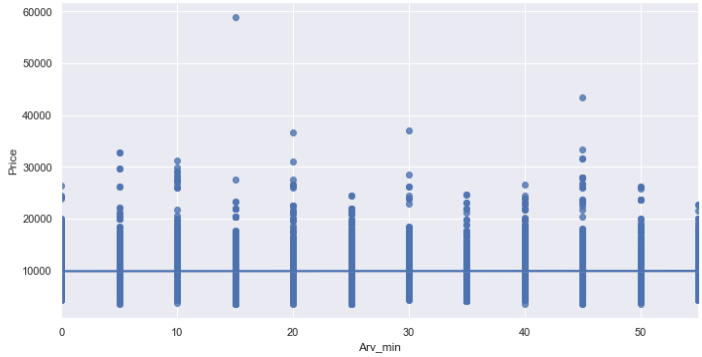


Screenshot of the plot

We used the catplot multiple times to visualize the relationship between an independent columns and the target. Here in this plot we found a good positive correlation between the Stops feature and the price. This meant that more the number of stops, the higher the price gets.

- Lineplot:

```
sns.lmplot(x ='Arv_min',y ='Price', data = df, aspect=2);
```

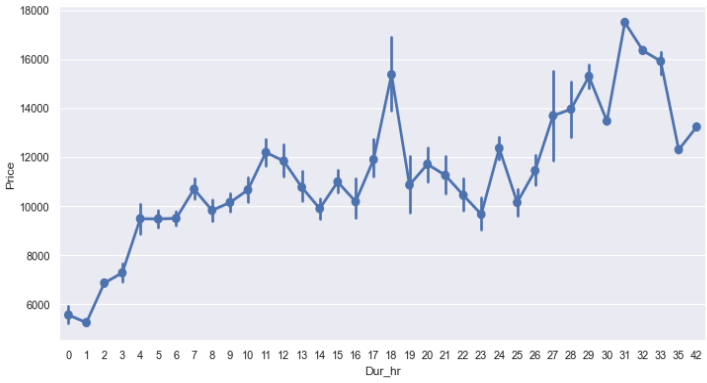


Screenshot of the plot

We used the lineplot multiple times to visualize the relationship between an independent columns and the target. Here in this plot we found no correlation between the target and the arrival min column, which showed that the price do not depend on the min of arrival.

- Factorplot:

```
sns.factorplot(x ='Dur_hr', y ='Price', data = df, aspect=2);
```

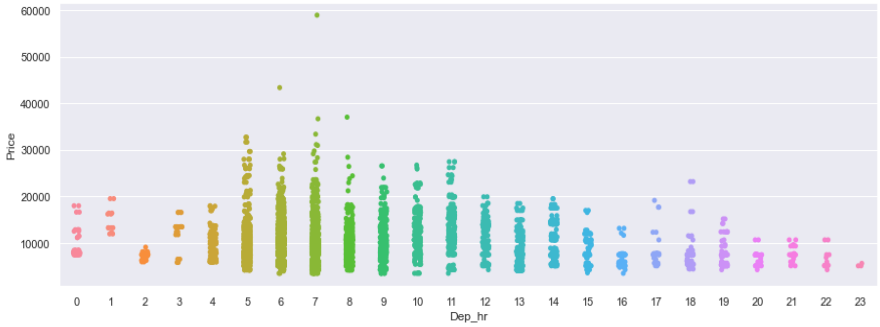


Screenshot of the plot

We used the factor plot multiple times to visualize the relationship between an independent columns and the target. Here in this plot we can see the trend of the graph, which is going up, this showed that the duration hr is positive correlated with the target.

- Stripplot:

```
plt.figure(figsize=[15,5])
sns.stripplot(data= df1, x='Dep_hr', y='Price')
plt.show()
```

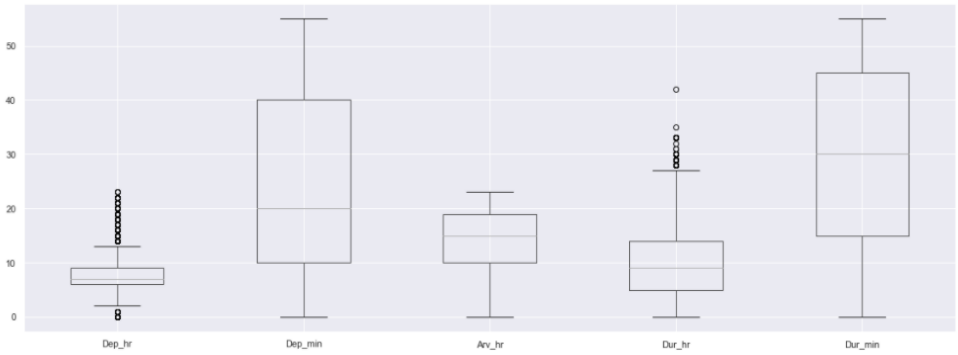


Screenshot of the plot

We used the Stripplot to visualize the relationship between an independent columns and the target. The graph here showed some negative correlation between the columns. This showed that the more expensive flights run between 5 to 12 AM of the day.

- Boxplot:

```
df.iloc[:,6:].boxplot(figsize=[20,7])
plt.show()
```



Screenshot of the plot

We used the boxplot to check for any outliers in the dataset. In this plot we can see some outliers present in some columns which are very close to the threshold.

- Interpretation of the Results

From the pre-processing we interpreted:

- Some unwanted columns were present in the dataset, which had no role to play in model building.
- There were no missing values present in the dataset.
- The dataset contained many unwanted strings and characters with the actual values along with some duplicate values written in different manners.
- In the airline feature, we found an airline having only three records in the entire dataset, which we removed as it may have affected the training.
- Majority of features were related to date and time.
- There were few categorical columns present which needed proper encoding .
- One of the features had a near zero correlation with the target which could affect the model performance.
- It is better not to removing outliers which are close to threshold as this can lead to data loss.
- Some of the numeric column showed skewness in the distribution of data, which was then treated.
- Scaling the features is very important before training the model in order to get a good model performance.

From the visualization we interpreted:

- The distribution of data in the continuous numeric columns had variance and skewness in the data.
- Among the independent columns there were all kinds of relation present, some showed positive relation and some negative.
- The count of values in the categorical columns also showed some variance as well as skewness.
- The features like ‘Stops’, ‘Dur hr’ gave a good positive correlation with the target.

- The feature like ‘Date’ showed a negative relationship with the target data.
- The feature ‘Dep min’ had no correlation with the target data.
- There were few outliers present in some of the features in the dataset, but were close to the threshold.

From the modelling we interpreted:

- Finding the best performing random state can be very useful in getting a good score.
- Scaled data give a higher model performance as compared to non-scaled data.
- Training multiple models helps us to choose the best fit model for our dataset.
- Performing cross validation test is very important to detect if there are overfitted or underfitted models.
- The decision tree model gave us a very good training and r2 score, but when checked cv score the result was very poor, which meant that the model was being overfitted.
- Hyper parameter tuning play a very crucial role in finding the best parameters for the models which helps us in getting the best results for the models.
- In this project the XGB regressor model performed the best among all the models, which was then saved as the final model.

CONCLUSION

- Key Findings and Conclusions of the Study

From the project, we found that while scraping the data it is important to use different websites as the data source. We fetched nearly 5900 records, which formed the dataset for the project and since the target was price, we had to follow the regression approach. We found that the data contained an unwanted column which was the index column which we dropped, also we found many junk data attached to the actual data for which the numeric columns turned to object data type, along with this some duplicate values were also found which were written in different manners. During data cleaning we found that for the feature airline, a value contained only three records in the entire dataset, which we then removed as it may have affected our model. We found that majority of the features were of date and time, for time data we had to split the features into hours and minutes also the date contained different dates but of the same month and year, hence we kept only the date data. Also we found a feature 'arrival minute' that had no correlation with the target data, for which we dropped the feature.

As we were given some queries to find answers for, we analysed the data visually by checking correlations between different features and the target and we came up with the following analysis:

1. Yes, airfares do change frequently, it changes with different times of the day and also for different dates.
2. The fare for the present day flights are very high and for the next day there is a drop in the price with a quite large jump, and following this the fares of flights vary in small jumps in the prices decreasing as we go further in the date.
3. The prices tend to go up and down over time.
4. As we get closer to the departure date, the price of the fares increases.

5. The expensive flights run between 5 to 12 AM in the morning when compared to other times of the day, but there are cheaper flights as well during that time of the day. So we can say morning that flights have both expensive as well as cheaper flights.
6. We can say that the best time to buy tickets is a week or two before the travel date, and if this is not possible customers can buy tickets at least 2 days before the travel and they can opt for flights between 5AM to 2PM of the day as the majority of the flights fly between these hours and the customers will find more options on flights during these hours, also the customers should keep into consideration the duration of the flight they opt for as the duration is positive correlated with the price of fare.

   We found that the dataset contained few outliers, but they were very close to the threshold, also some skewness were found in certain features which we then treated. During the model building, while checking for CV score we found that the decision tree model was being overfitted and finally by hyper parameter tuning we found that the XGBoost model was performing the best in our project and hence we saved the model as our final model.

- Learning Outcomes of the Study in respect of Data Science

  From the project we got much knowledge on the approach on building a good performing model. We learnt that performing EDA thoroughly can give us many insights into the data. We also learned that Data cleaning play a very crucial role in the project, without cleaning the data we could not have sent the dataset for training. The visualization of the dataset is also very important in understanding the distribution of data in different features, visualization is also very useful is finding the relation between columns through which we were able to identify the important and non important features of the use case. We learnt that the features which do not have any correlation with the target data should be avoided in building the model as it may affect the model

performance. Also we got the idea that scaling the dataset can be very useful in improving model score. We also learnt how checking cross validation score can help us in identifying the overfitted models.

We learnt that if there are few outliers in the data and they are very close to the threshold then we can skip removing of the outliers to avoid data loss. And finally performing the hyper parameter tuning gave us the best parameter values for our models which helped us in determining the final model for our project.

- Limitations of this work and Scope for Future Work

  The model we created was based on the dataset that we scraped which belonged to different dates of same month of this year, but since flight prices vary a lot with respect to time and are very unpredictable, the model may perform slightly weak in case of flights running way after the time period for which the data is scraped. Also the solution provided can perform weak if large numbers of outliers are present in the data.

  The model we created can be used by the customers to predict the prices of fares for different flights and they can compare the prices to find the better deal. It can also be used to determine the best time to buy tickets for the travel so that the customers can save the most.

  To improve the results further, more data of different months can be fetched to train the model. Also we can fetch data for more routes which will help our model perform even better. We can also use more different websites for scraping data and add more features to the scraping list, this will help us to avoid overfitting and we can identify more important features. Also different other models can be used to check which model performs best. And for hyper parameter tuning the models we can use more number of parameters in order to improve the model performance.