FLIGHT PRICE PREDICTION PROJECT

Submitted by:

HARSH NEMA

ACKNOWLEDGMENT

I would like to express my profound gratitude to the **Flip Robo team**, who has given me this opportunity to deal with a beautiful dataset and it has helped me to improve my analytical skills. I would like to express my special thanks to our mentor **Mr. Shwetank Mishra Sir** (SME Flip Robo) for their contributions to the completion of my project titled **'Flight Price Prediction Project'**.

I am eternally grateful to **"Datatrained"** for giving me the opportunity for an internship at Flip Robo. Last but not least I have to thank my parents and wife for their love and support during my project.

References:

1. SCIKIT Learn Library Documentation
2. Datatrained recorded videos for Data Science Course
3. Hands-on Machine learning with Scikit learn and tensor flow by Aurelien Geron
4. Flight Fare Prediction Using Machine Learning, https://www.analyticsvidhya.com/blog/2022/01/flight-fare-prediction-using-machine-learning/
5. Airline Price Prediction using Machine Learning, Jaya Shukla et al., International Journal of Research in Engineering, IT and Social Sciences, ISSN 2250-0588, Impact Factor: 6.565, Volume 10 Issue 05, May 2020, Page 15-18
6. Airline ticket price and demand prediction: A survey; Juhar AhmedAbdell, NM Zaki, Khaled Shuaib and Fahad Khan

1. INTRODUCTION

• Business Problem Framing

Airline Companies are considered one of the most enlightened industries to allocate airline prices dynamically. These industries are trying to keep their revenue as high as possible and boost their profit. **Data science** comes as a very **important tool to solve problems** in the domain to help companies increase their overall revenue, and profits, improving their marketing strategies and focusing on changing trends in the price of the flight.

Regression is a *supervised learning algorithm in machine learning* which is used for prediction by learning and forming a relationship between present statistical data and target value i.e. **Price** in this case.

On account of the high intricacy of the pricing models applied by the airlines, it's very complicated for a customer to buy an air ticket at the least price, the formulation of tickets bought in advance is low priced, and customers who secure a ticket earlier may pay more than those who bought the same ticket later. We will try to understand how exactly the prices vary with the different variables to predict the actual price of the flights.

• Conceptual Background of the Domain Problem

Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on:

1. Time of purchase patterns (making sure last-minute purchases are expensive)

2. Keeping the flight as full as they want it (raising prices on a flight that is filling up to reduce sales and hold back inventory for those expensive last-minute expensive purchases)

Dynamic pricing allows more absolute forecasting of ticket prices on sparkling factors such as a change in demand and price discrimination. However dynamic pricing is challenging as it is highly dominated by various factors including internal and external factors, competition among airlines and strategic customers.

Early prediction of the demand along a given route could help an airline company pre-plan the flights and determine appropriate pricing for the route. Existing demand prediction models generally try to predict passenger demand for a single flight/route and the market share of an individual airline. Price discrimination allows an airline company to categorize customers based on their willingness to pay and thus charge them different prices.

• Review of Literature

From the customer side, two kinds of models are proposed by different researchers to save money for customers: models that predict the optimal time to buy a ticket and models that predict the minimum ticket price.

From the customer's point of view, determining the minimum price or the best time to buy a ticket is the key issue. The concept of “tickets bought in advance are cheaper” is no longer working (William Groves and Maria Gini, 2013). Moreover, early purchasing implies a risk of commitment to a specific schedule that may need to be changed usually for a fee. The ticket price may be affected by several factors and thus may change continuously. To address this, various studies were conducted to support the customer in determining an optimal ticket purchase time and ticket price prediction (Anastasia Lantseva et al., 2015, Chawla and Kaur, 2017, Domínguez-Menchero, 2014; K. Tziridis et al., 2017, Li et al., 2014, Santana and Saulo, 2017, L. Li and K. Chu, 2017; T. Wohlfarth et al., 2011, Vu et al., 2018, Groves and Gini, 2013, William Groves and Maria Gini, 2015; Y. Chen et al., 2015, Xu and Cao, 2017).

Most of the studies performed on the customer side focus on the problem of predicting optimal ticket purchase time using statistical methods. As noted by Y. Chen et al. (2015), predicting the actual ticket price is a more difficult task than predicting an optimal ticket purchase time due to various

reasons: absence of enough datasets, external factors influencing ticket prices, dynamic behaviour of ticket pricing, competition among airlines, proprietary nature of airlines ticket pricing policies, etc. Nevertheless, few studies have attempted to predict actual ticket prices with the work done by the authors (Anastasia Lantseva et al., 2015, Domínguez-Menchero, 2014; K. Tziridis et al., 2017, Santana and Saulo, 2017, L. Li and K. Chu, 2017, Vu et al., 2018) as examples.

Nowadays, social media sentiment analysis has become a good source of information for various data mining models. For example, social media data has been used for event prediction (A. Dingli et al., 2015, Arif Nurwidyantoro and Edi Winarko, 2013; Hila Becker et al., 2012; Mario Cordeiro, 2012; Nikolaos Panagiotou et al., 2016; Takeshi Sakaki et al., 2010; Xiaowen Dong et al, 2015), competitor intelligence (Lipika Dey et al., 2011; Malu Castellanos et al., 2011; Martin Längkvist et al., 2014; Wu He et al., 2015), price prediction (A. Porshnev et al., 2013; J. Santos Domínguez-Menchero et al., 2014; L. Bing et al., 2014, L. Li and K. Chu, 2017) and tourist traffic flow prediction (R. Linares et al., 2015) and many more.

Motivation for the Problem Undertaken

The project is provided to me by Flip Robo Technologies as a part of the internship program. The exposure to real-world data and the opportunity to deploy my skillset in solving a real-time problem has been my primary motivation.

1) Nowadays there are lots of apps for flight ticket booking, if a passenger wants to travel from one space to another space so they don't know actually what the prices of that same space flight are.

2) To save their money and time we will decide to develop such a system in which users can book the flight ticket according to their needs.

2. Analytical Problem Framing

- ## Mathematical/ Analytical Modeling of the Problem

The first phase of problem modeling involves data scraping of flights from the internet. For that purpose, flight data is scrapped from www.yatra.com for a timeframe of 06 Jan 2023 to 16 Jan 2023. Data is scraped for flights on the route from New Delhi to Mumbai. Data is scrapped for Economy class, Premium Economy class & Business class flights. The next phase is data cleaning & pre-processing for building ML Model. Our objective is to predict flight prices which can be resolved by the use of a regression-based algorithm. Further hyperparameter tuning was performed to build a more accurate model out of the best models.

- ## Data Sources and their formats

Data is collected from www.yatra.com for a timeframe of 06 Jan 2023 to 16 Jan 2023 using selenium and saved in a CSV file. Data was scraped for flights on the route from New Delhi to Mumbai. Data is scrapped for Economy class, Premium Economy class & Business class flights. Around 2400 flight details are collected for this project.

The dataset looks as shown below:

```
1 df.reset_index()
```

	index	Airline	Flight_No	Date	Departure_Time	Arrival_Time	Source	Destination	Stops	Duration	Price	Class
0	0	IndiGo	6E-2519	Fri, 6 Jan 2023	23:00	01:20\n+ 1 day	New Delhi	Mumbai	Non Stop	2h 20m	5899	Economy Class
1	1	Go First	G8-336	Fri, 6 Jan 2023	17:35	19:45	New Delhi	Mumbai	Non Stop	2h 10m	6502	Economy Class
2	2	Go First	G8-323	Fri, 6 Jan 2023	18:40	20:55	New Delhi	Mumbai	Non Stop	2h 15m	6502	Economy Class
3	3	Vistara	UK-933	Fri, 6 Jan 2023	15:30	17:35	New Delhi	Mumbai	Non Stop	2h 05m	6795	Economy Class
4	4	Vistara	UK-955	Fri, 6 Jan 2023	17:45	19:55	New Delhi	Mumbai	Non Stop	2h 10m	6795	Economy Class
...	...	...	...	...	...	...	...	...	...	...	...	...
2363	2427	Vistara Business	UK-843/848	Mon, 16 Jan 2023	20:55	17:05\n+ 1 day	Mumbai	New Delhi	1 Stop	20h 10m	55572	Business Class
2364	2428	Vistara Business	UK-877/880	Mon, 16 Jan 2023	10:45	15:35	Mumbai	New Delhi	1 Stop	4h 50m	55684	Business Class
2365	2429	Vistara Business	UK-877/890	Mon, 16 Jan 2023	10:45	19:55	Mumbai	New Delhi	1 Stop	9h 10m	55684	Business Class
2366	2430	Vistara Business	UK-873/880	Mon, 16 Jan 2023	06:20	15:35	Mumbai	New Delhi	1 Stop	9h 15m	55684	Business Class
2367	2431	Vistara Business	UK-841/848	Mon, 16 Jan 2023	11:25	17:05	Mumbai	New Delhi	1 Stop	5h 40m	56238	Business Class

2368 rows × 12 columns

There are 11 features in dataset including target feature ‘Price’ and one unnecessary column name 'Unnamed: 0' dropped. The data types of different features are as shown below:

```
No. of Rows in the dataset : 2432
No. of Columns in the dataset : 12
```

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

```
[ ]  1 # lets sort columns by their datatype
     2 df.columns.to_series().groupby(df.dtypes).groups
```

```
{int64: ['Price'], object: ['Airline', 'Flight_No', 'Date', 'Departure_Time', 'Arrival_Time', 'Source', 'Destination', 'Stops', 'Duration', 'Class']}
```

- **<u>Data Preprocessing</u>**

The dataset is large and it may contain some data errors. To make clean and error-free data, some data cleaning & data pre-processing was performed on the scraped dataset.

- Checking Null values in the dataset

```
[ ]  1 df.isnull().sum()
```

```
Airline          0
Flight_No        0
Date             0
Departure_Time   0
Arrival_Time     0
Source           0
Destination      0
Stops            0
Duration         0
Price            0
Class            0
dtype: int64
```

No missing values in the dataset.Good to go ahead

- No missing values were found in the dataset
- Duplicated entries found in the dataset and dropped

```
1 # Checking for duplicate data
2 df.duplicated().sum()
```

```
64
```

```
[ ]  1 df = df.drop_duplicates()
```

- Conversion of Duration column from hour & Minutes format into minutes

Conversion of Duration column into minutes

```
[ ]  1 df['Duration_in_min'] = df['Duration'].map(lambda x : x.replace('05m','5m'))
```

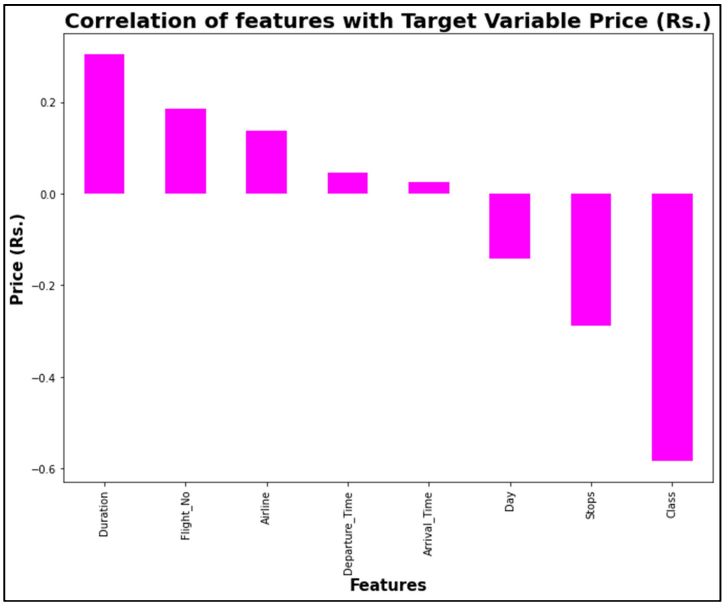
```
1 # Conversion of Duration column from hr & Minutes format to Minutes
2 df['Duration_in_min'] = df['Duration_in_min'].str.replace('h','*60').str.replace(' ','+').str.replace('m','*1').apply(eval)
3
4 # convert this column into a numeric datatypes
5 df['Duration_in_min']= pd.to_numeric(df['Duration_in_min'])
```

- Changing date format

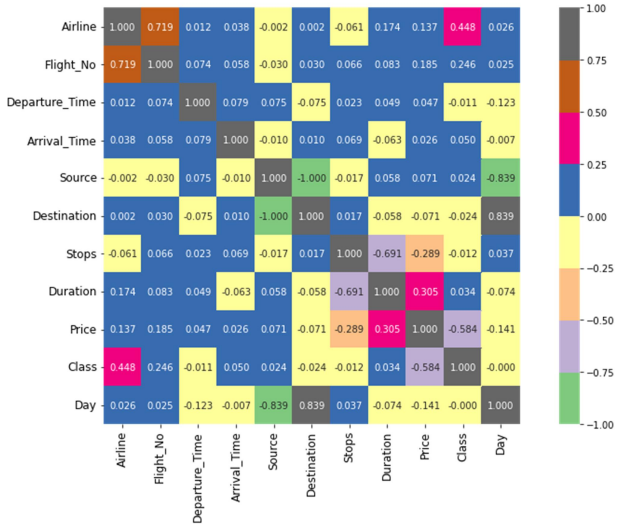
```
[ ]  1 df['Date_chng_format']=pd.to_datetime(df['Date'],format = '%a, %d %b %Y')

[ ]  1 df['Day'] = df['Date_chng_format'].dt.day
     2 df['Month'] = df['Date_chng_format'].dt.month
     3 df['Year'] = df['Date_chng_format'].dt.year
```

The new column for 'Day', 'Month', and 'Year' is extracted from the Date column

- **Data Inputs- Logic- Output Relationships**



A correlation heat map is plotted to gain an understanding of the relationship between target features & independent features.



We can see that the class feature is correlated for more than -0.6 with the target variable Price. The remaining feature is moderately or poorly correlated with the target variable price.

- **Hardware and Software Requirements and Tools Used**

  Hardware Used -

  1. Processor — Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz 3.60 GHz

  2. RAM — 16.0 GB

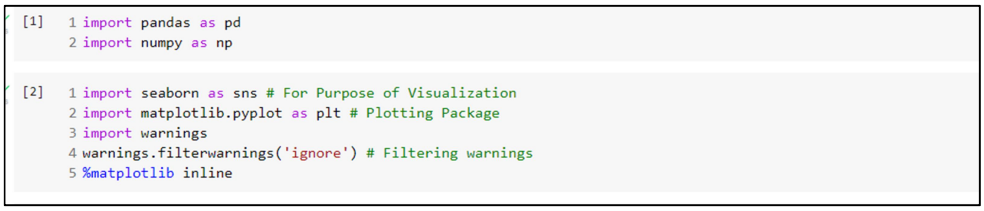
  3. GPU — 2GB AMD Radeon Graphics card

  Software utilized -

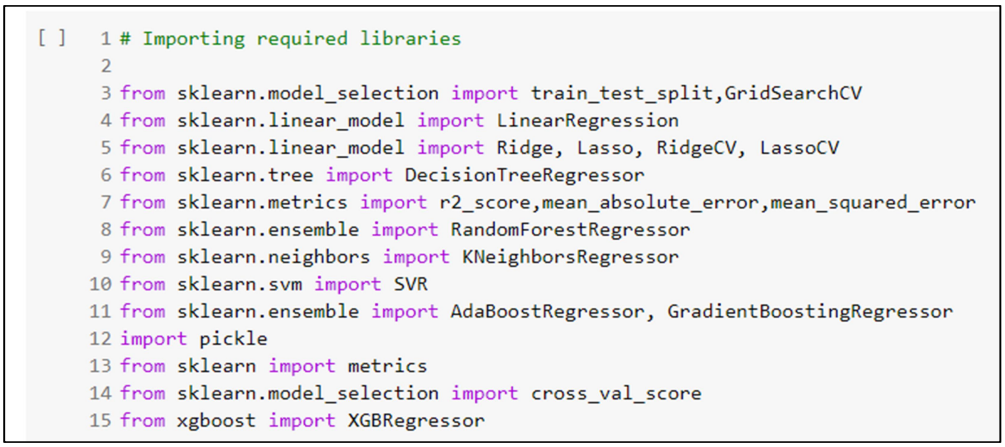
  1. Google Colab Notebook - to write and execute arbitrary python code through the browser

  2. Microsoft office – for making project reports and ppt

  Libraries Used – General libraries used for data wrangling



```
[1]  1 import pandas as pd
     2 import numpy as np

[2]  1 import seaborn as sns # For Purpose of Visualization
     2 import matplotlib.pyplot as plt # Plotting Package
     3 import warnings
     4 warnings.filterwarnings('ignore') # Filtering warnings
     5 %matplotlib inline
```



```
[ ]  1 # Importing required libraries
     2
     3 from sklearn.model_selection import train_test_split,GridSearchCV
     4 from sklearn.linear_model import LinearRegression
     5 from sklearn.linear_model import Ridge, Lasso, RidgeCV, LassoCV
     6 from sklearn.tree import DecisionTreeRegressor
     7 from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
     8 from sklearn.ensemble import RandomForestRegressor
     9 from sklearn.neighbors import KNeighborsRegressor
    10 from sklearn.svm import SVR
    11 from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor
    12 import pickle
    13 from sklearn import metrics
    14 from sklearn.model_selection import cross_val_score
    15 from xgboost import XGBRegressor
```

3. Models Development and Evaluation

- ## Identification of possible problem-solving approaches (methods)

  Our objective is to **predict flight prices** and analyze features impacting the price. This problem can be solved using a **regression-based machine learning algorithm** like linear regression. For that purpose, the first task is to convert a categorical variable into numerical features. Once data encoding is done the data is scaled using a **standard scalar**. The final model is built over this scaled data. For building an ML model before implementing the regression algorithm, data is **split into training & test data** using train & test split from the model selection module of the sklearn library.

  Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. After that model is trained with various regression algorithms and 5-fold cross-validation is performed. Further **hyperparameter tuning** was performed to build a more accurate model out of the best model.

- ## Testing of Identified Approaches (Algorithms)

The different regression algorithms used in this project to build the ML model are as below:

- ✓ Linear Regression
- ✓ KNeighbors Regressor
- ✓ Random Forest Regressor
- ✓ Ada Boost Regressor
- ✓ Support Vector Regressor
- ✓ Extreme Gradient Boosting Regressor (XGB)

- **Run and Evaluate selected models**

1. Linear Regression Model

```
[ ]
lr = LinearRegression()

lr.fit(X_train,y_train)

pred_train = lr.predict(X_train)

pred_test = lr.predict(X_test)

score_train = r2_score(y_train,pred_train)

score_test = r2_score(y_test,pred_test)

print("Training accuracy:",score_train*100)

print("Testing accuracy:",score_test*100)

cv_score = cross_val_score(lr,X_scaler,y,cv=5)

cv_mean = cv_score.mean()

print(f"At cross fold 5, the cv score is {cv_mean} and accuracy score for training is {score_train} and accuracy for testing is {score_test}")

# MSE - ignoring some outliers means larger errors are punished. Hard to interpretate but most popular

mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error:",rmse)
```

```
Training accuracy: 66.2638642333782
Testing accuracy: 74.50033731402048
At cross fold 5, the cv score is -1.0331149390361531 and accuracy score for training is 0.662638642333782 and accuracy for testing is 0.7450033731402048
Root mean squared Error: 9135.849849368999
```

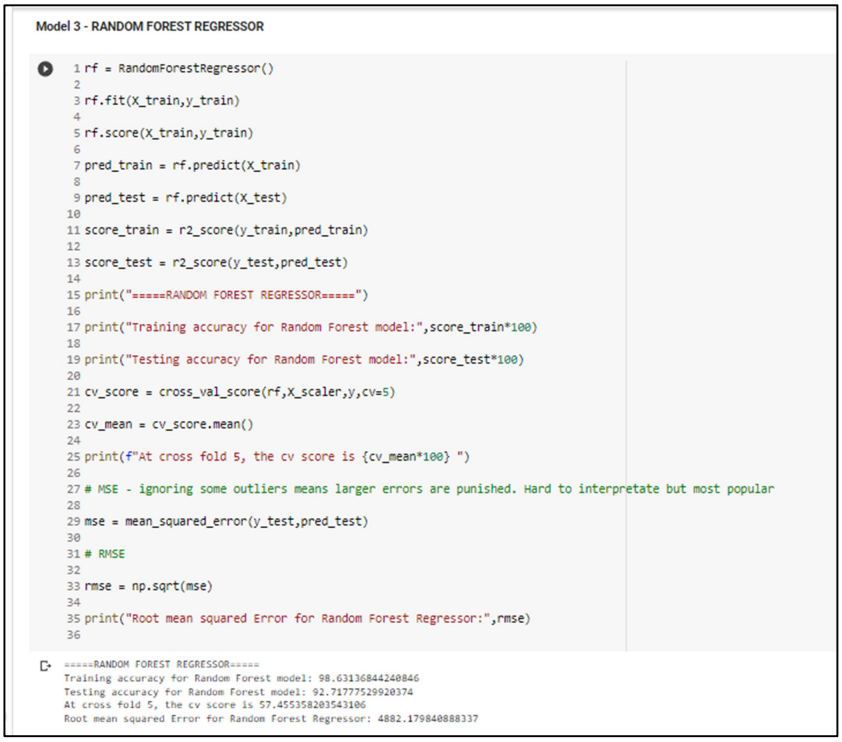
2. KNN Regressor

Model 2 - KNN REGRESSOR

```
[ ]
knn = KNeighborsRegressor()

knn.fit(X_train,y_train)

pred_train = knn.predict(X_train)       # Predicting training data with the model

acc_train = metrics.r2_score(y_train,pred_train)    # Training accuracy

print("=====K Neighbors Regressor=====")

print("R square score for training dataset for K Neighbors regressor: ", acc_train)

pred_test = knn.predict(X_test)       # Predicting test data with the model

acc_test = metrics.r2_score(y_test,pred_test)     # Testing accuracy

print("R square score for test dataset for K Neighbors Regressor: ", acc_test)

knn_score = cross_val_score(knn,X_scaler,y,cv=5)

knn_m = knn_score.mean()

print("Cross val score for K Neighbors Regresor:",knn_m*100)


mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error:",rmse)
```

```
=====K Neighbors Regressor=====
R square score for training dataset for K Neighbors regressor:  0.9098012833592373
R square score for test dataset for K Neighbors Regressor:  0.898871456658288
Cross val score for K Neighbors Regresor: -87.89534886692118
Root mean squared Error: 5753.321074144402
```

3. Random Forest Regressor

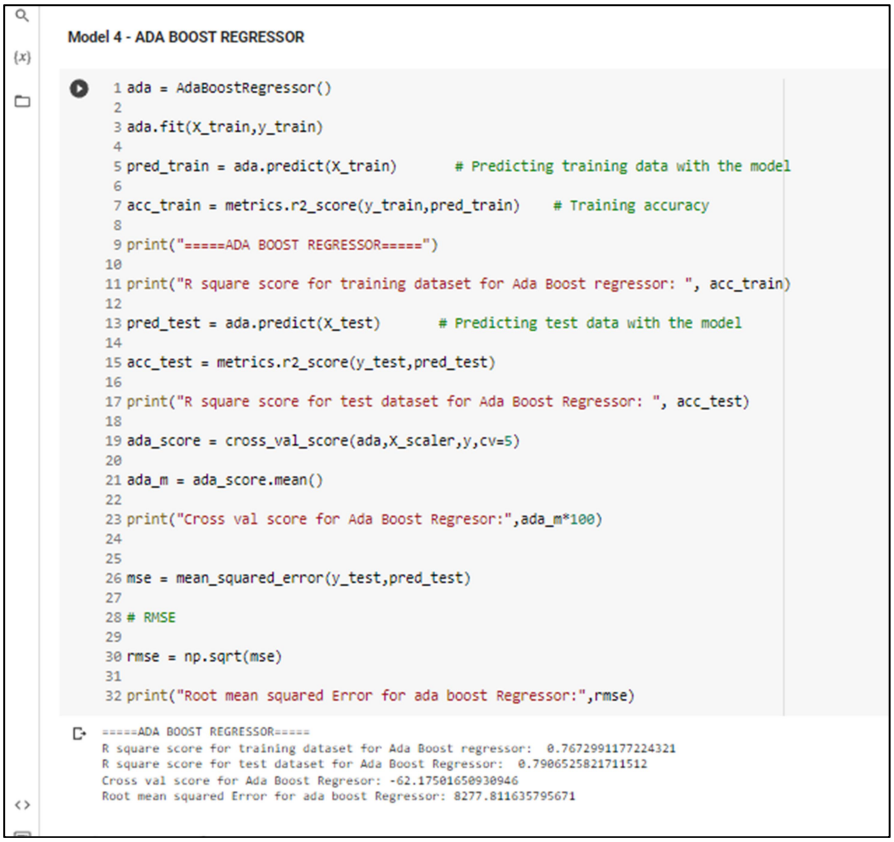


Model 3 - RANDOM FOREST REGRESSOR

```
rf = RandomForestRegressor()

rf.fit(X_train,y_train)

rf.score(X_train,y_train)

pred_train = rf.predict(X_train)

pred_test = rf.predict(X_test)

score_train = r2_score(y_train,pred_train)

score_test = r2_score(y_test,pred_test)

print("=====RANDOM FOREST REGRESSOR=====")

print("Training accuracy for Random Forest model:",score_train*100)

print("Testing accuracy for Random Forest model:",score_test*100)

cv_score = cross_val_score(rf,X_scaler,y,cv=5)

cv_mean = cv_score.mean()

print(f"At cross fold 5, the cv score is {cv_mean*100} ")

# MSE - ignoring some outliers means larger errors are punished. Hard to interpretate but most popular

mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for Random Forest Regressor:",rmse)

```

```
=====RANDOM FOREST REGRESSOR=====
Training accuracy for Random Forest model: 98.63136844240846
Testing accuracy for Random Forest model: 92.71777529920374
At cross fold 5, the cv score is 57.455358203543106
Root mean squared Error for Random Forest Regressor: 4882.179840888337
```

4. Ada Boost Regressor



Model 4 - ADA BOOST REGRESSOR

```
ada = AdaBoostRegressor()

ada.fit(X_train,y_train)

pred_train = ada.predict(X_train)       # Predicting training data with the model

acc_train = metrics.r2_score(y_train,pred_train)    # Training accuracy

print("=====ADA BOOST REGRESSOR=====")

print("R square score for training dataset for Ada Boost regressor: ", acc_train)

pred_test = ada.predict(X_test)       # Predicting test data with the model

acc_test = metrics.r2_score(y_test,pred_test)

print("R square score for test dataset for Ada Boost Regressor: ", acc_test)

ada_score = cross_val_score(ada,X_scaler,y,cv=5)

ada_m = ada_score.mean()

print("Cross val score for Ada Boost Regresor:",ada_m*100)


mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for ada boost Regressor:",rmse)
```

```
=====ADA BOOST REGRESSOR=====
R square score for training dataset for Ada Boost regressor:  0.7672991177224321
R square score for test dataset for Ada Boost Regressor:  0.7906525821711512
Cross val score for Ada Boost Regresor: -62.17501650930946
Root mean squared Error for ada boost Regressor: 8277.811635795671
```

5. Support Vector Regressor

Model 5 - SUPPORT VECTOR REGRESSOR

```
svr = SVR()

svr.fit(X_train,y_train)

pred_train = svr.predict(X_train)       # Predicting training data with the model

acc_train = metrics.r2_score(y_train,pred_train)    # Training accuracy

print("R square score for training dataset for SVR: ", acc_train)

pred_test = svr.predict(X_test)       # Predicting test data with the model

acc_test = metrics.r2_score(y_test,pred_test)

print("R square score for test dataset for SVR: ", acc_test)

svr_score = cross_val_score(svr,X_scaler,y,cv=5)

svr_m = svr_score.mean()

print("Cross val score for SVR:",svr_m*100)


mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for SVR:",rmse)
```

```
R square score for training dataset for SVR:  -0.1145313586749237
R square score for test dataset for SVR:  -0.150396272743321
Cross val score for SVR: -177.8788502326215
Root mean squared Error for SVR: 19404.634565938213
```

6. XGradient Boosting Regressor

Model 6 - Extreme Gradient Boosting Regressor

```
xgb = XGBRegressor()

xgb.fit(X_train,y_train)

xgb.score(X_train,y_train)

pred_train = xgb.predict(X_train)

pred_test = xgb.predict(X_test)

score_train = r2_score(y_train,pred_train)

score_test = r2_score(y_test,pred_test)

print("=====XGBoosting REGRESSOR=====")

print("Training accuracy for XGradient Boost model:",score_train*100)

print("Testing accuracy for XGradient Boost model:",score_test*100)

cv_score = cross_val_score(xgb,X_scaler,y,cv=5)

cv_mean = cv_score.mean()

print(f"At cross fold 5, the cv score is {cv_mean*100} ")

# MSE - ignoring some outliers means larger errors are punished. Hard to interpretate but most popular

mse = mean_squared_error(y_test,pred_test)

# RMSE

rmse = np.sqrt(mse)

print("Root mean squared Error for XGradient Boosting model:",rmse)
```

```
[10:05:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
=====XGBoosting REGRESSOR=====
Training accuracy for XGradient Boost model: 92.20843757213169
Testing accuracy for XGradient Boost model: 91.18479392429315
[10:05:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:05:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Hyper Parameter Tuning for XGradient Boost Regressor

Tuning parameters for XGB

```
params_xgb = {'max_depth':[3,4,5,6],
              'learning_rate':(0.01,0.001,0.1),
              'min_child_weight':(0,1,2),
              'max_delta_step':(0,1,2)
              }

grd_xgb = GridSearchCV(xgb,params_xgb)

grd_xgb.fit(X_train,y_train)

print(grd_xgb.best_params_)
```

```
[10:08:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:08:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
{'learning_rate': 0.1, 'max_delta_step': 0, 'max_depth': 5, 'min_child_weight': 2}
```

```
xgb = grd_xgb.best_estimator_
xgb.fit(X_train,y_train)
xgb.score(X_train,y_train)
pred_train = xgb.predict(X_train)
pred_test = xgb.predict(X_test)
score_train = r2_score(y_train,pred_train)
score_test = r2_score(y_test,pred_test)
print("=====XGradient Boosting REGRESSOR=====")
print("Training accuracy for XGradient Boost model:",score_train*100)
print("Testing accuracy for XGradient Boost model:",score_test*100)
cv_score = cross_val_score(xgb,X_scaler,y,cv=5)
cv_mean = cv_score.mean()
print(f"At cross fold 5, the cv score is {cv_mean*100} ")
# MSE - ignoring some outliers means larger errors are punished. Hard to interpretate but most popular
mse = mean_squared_error(y_test,pred_test)
# RMSE
rmse = np.sqrt(mse)
print("Root mean squared Error for XGradient Boosting model:",rmse)
```

```
[10:09:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
=====XGradient Boosting REGRESSOR=====
Training accuracy for XGradient Boost model: 96.46038032319278
Testing accuracy for XGradient Boost model: 92.50573204920447
[10:09:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:09:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:09:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:09:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[10:09:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
At cross fold 5, the cv score is 56.40740832801063
Root mean squared Error for XGradient Boosting model: 4952.749283690575
```

There is minimum difference between training and testing R2 score for XGB model

Hence XGB is the model which best fits our data in predicting the price of flights

- Key Metrics for success in solving a problem under consideration

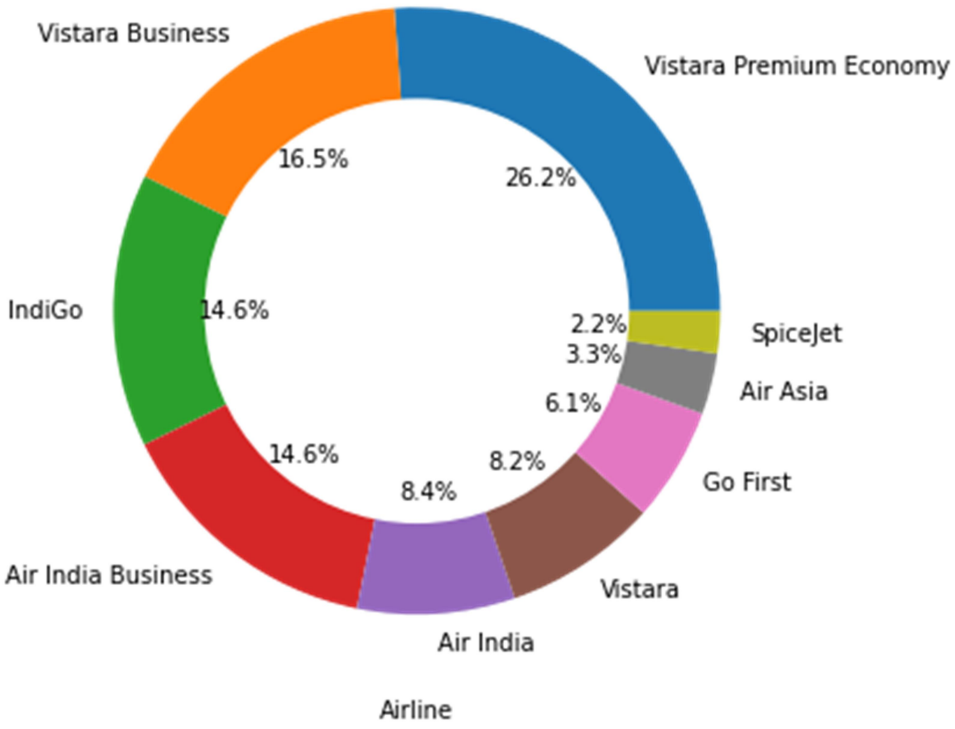
Following metrics used for evaluation:

1. Root mean square error is one of the most commonly used measures for evaluating the quality of predictions.

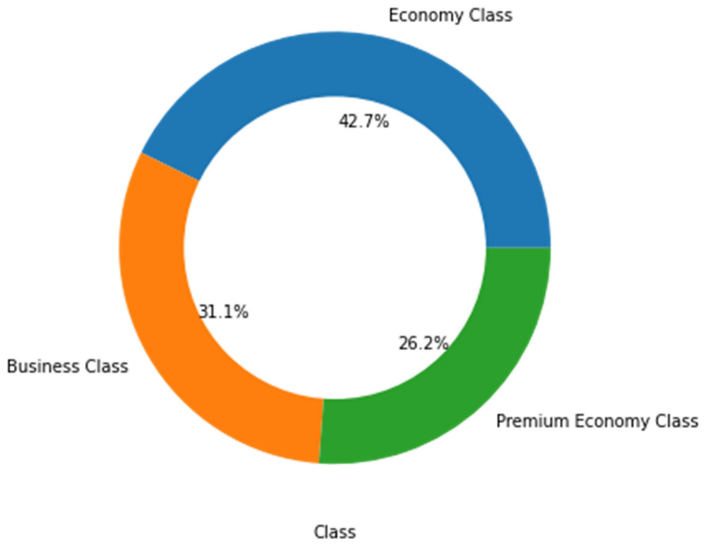
2. R2 score which tells us how accurately our model predicts a result, is going to be an important evaluation criterion

3. Cross-Validation Score

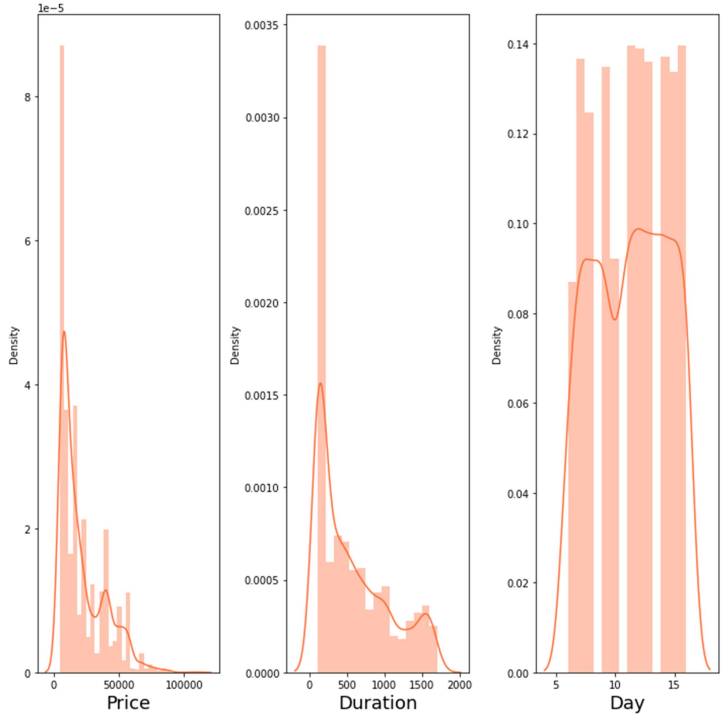
- Visualizations



✓ We can see the maximum number of flights run by **Vistara Premium Economy** while the minimum Flights run by Spicejet

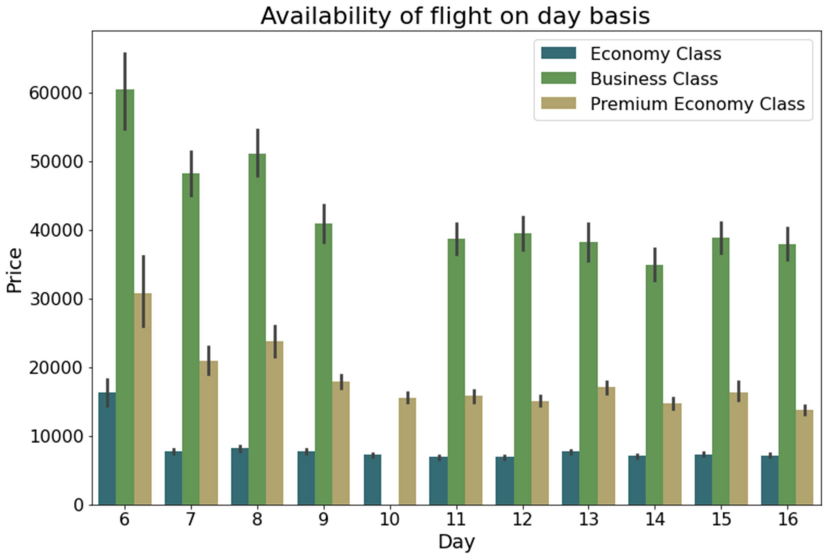


- ✓ Around 26% of flights of Premium Economy Class and 31% of flights are Business class
- ✓ 42.7% of flights are of Economy class, as they are low-cost flights & most people prefer it



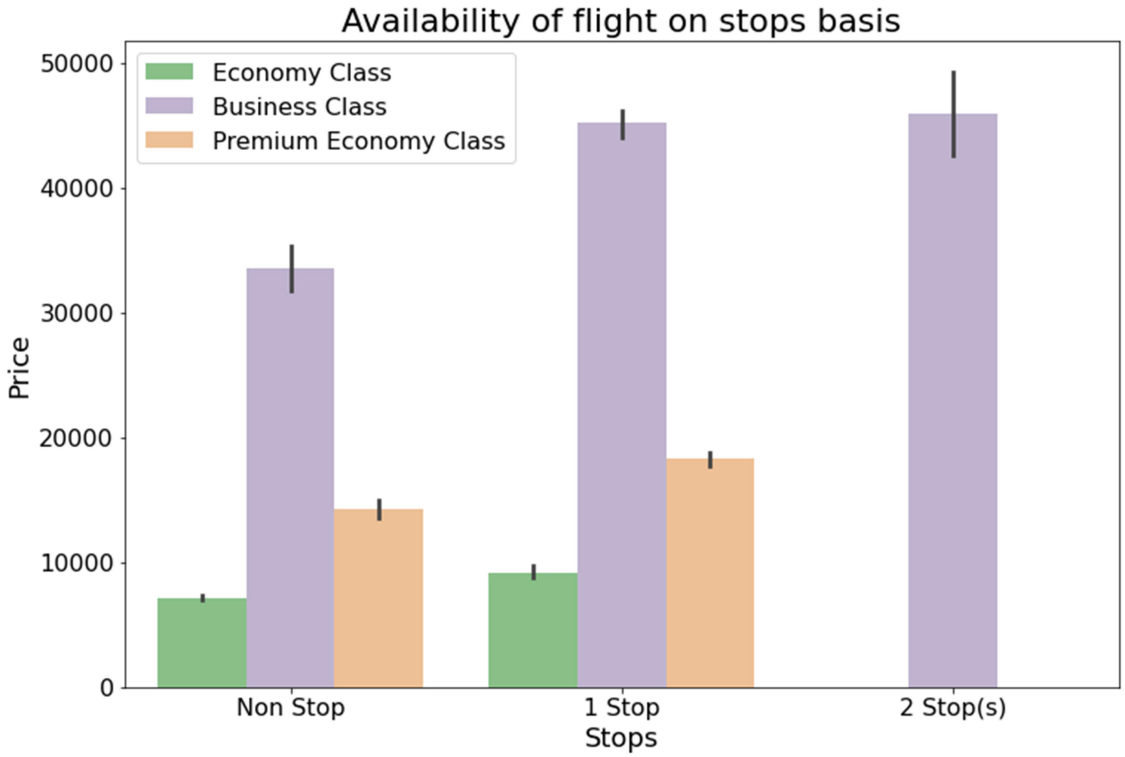
✓ Data distribution of Price and Duration are positively skewed

❖ **Is any day of week affect flight pricing?**



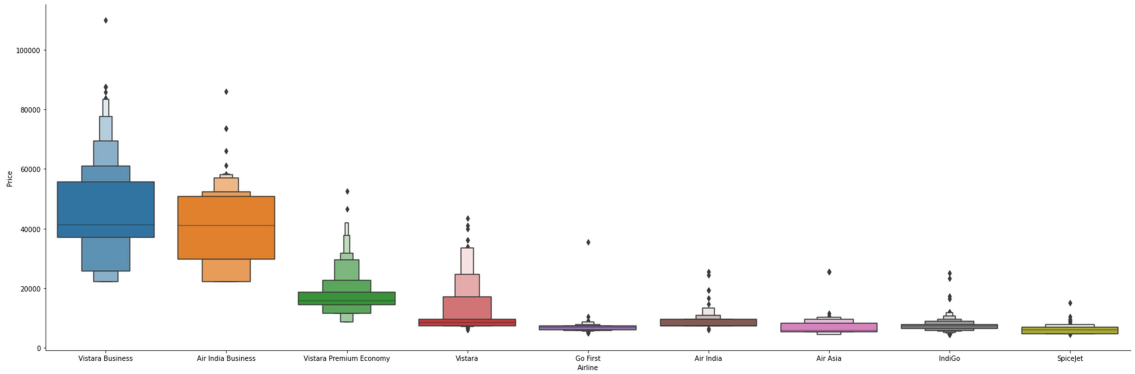
- ✓ The average price of all the classes is on the higher side on 6th Jan2023 i.e. Friday
- ✓ So if we book flights a day before, it will be costlier

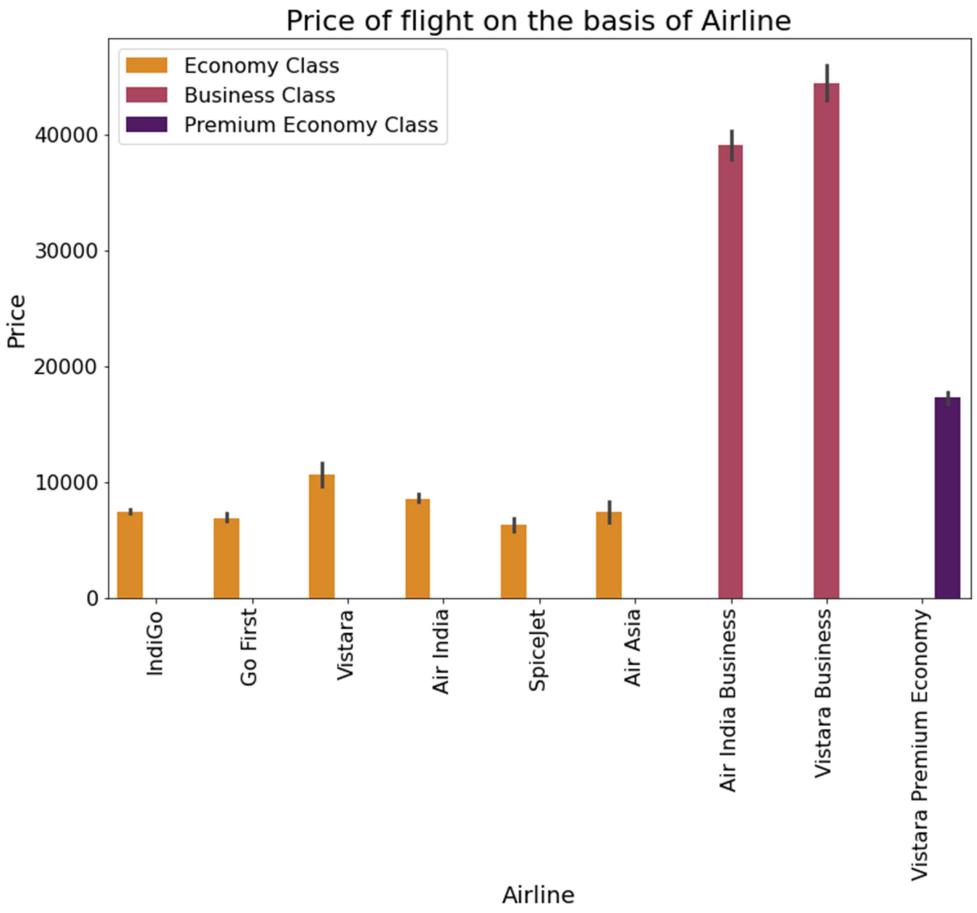
❖ Effect of number of stops on flight pricing



- ✓ Nonstop flights are the cheapest
- ✓ As the number of stops increases, the price of the flight increases

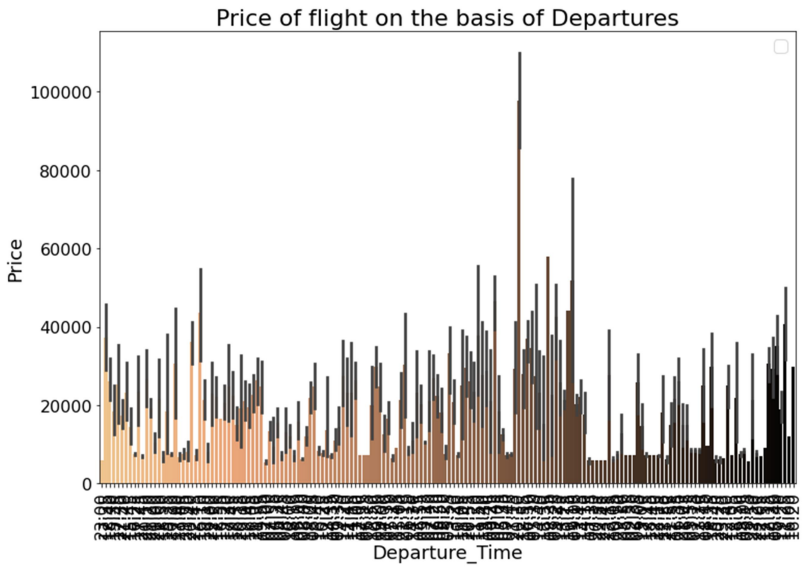
❖ Effect of the airline on flight Pricing





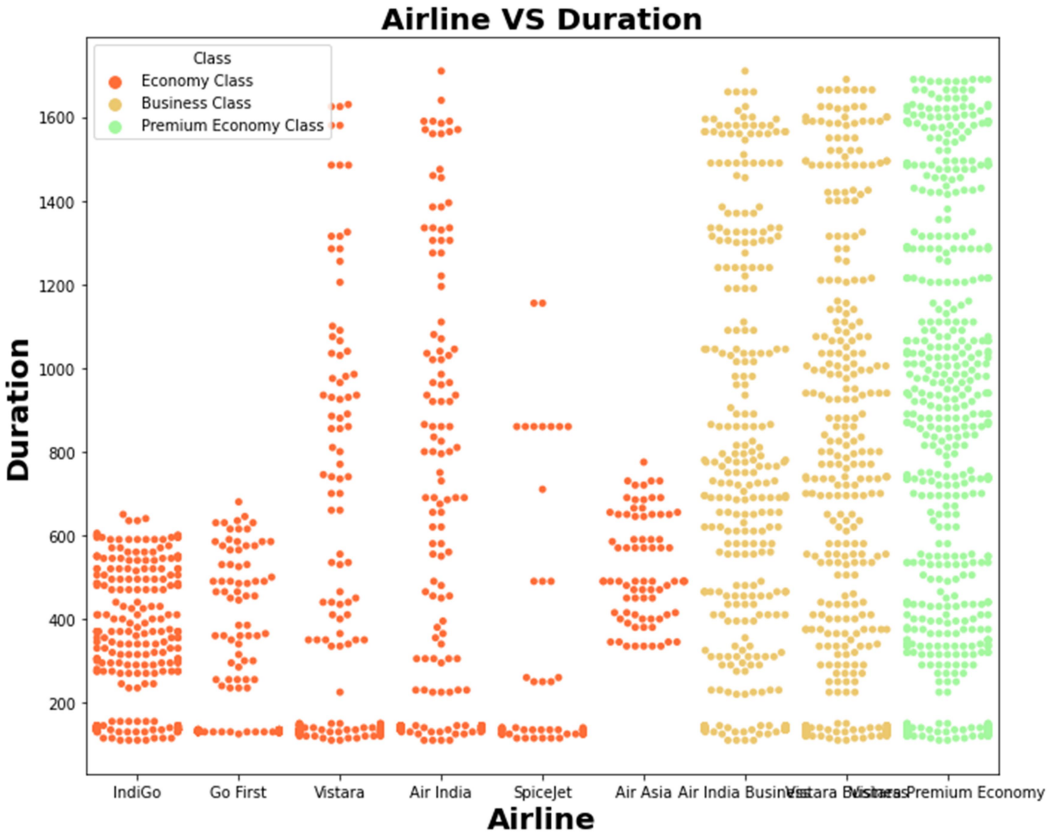
- ✓ Spicejet flights have the lowest fare among all the flights
- ✓ Followed by Indigo having a lower price of flights

❖ **Effect of Departures on flight pricing**



✓ It seems **day flights are costlier** than morning and evening flights

❖ **Airline Vs. Duration**



4. CONCLUSION

- **Key Findings and Conclusions of the Study**

ML Algorithm	R2 Score		CV Score
	Train	Test	
Linear Regression	66.26%	74.50%	-1.03
K Neighbors Regressor	90.98%	89.88%	87.89%
Random Forest Regressor	98.63%	92.72%	57.45%
Ada Boost Regressor	76.72%	79.06%	62.17%
Support Vector Regressor	11.45%	15.03%	-177.87
Extreme Gradient Boost Regressor(XGB)	92.2%	91.18%	59.47%
Hyper Parameter Tuning for XGB	96.46%	92.50%	56.40%

- ✓ The extreme Gradient Boosting(XGB) model gives the best result for the given dataset
- ✓ As we get the highest test accuracy at 91.18% and training accuracy at 92.2%

- **Learning Outcomes of the Study in respect of Data Science**

- ✓ XGB works well compared to other machine-learning algorithms
- ✓ XGBoost consists of the objective function and base learners
- ✓ The loss function is present in the objective function
- ✓ The loss function shows the difference between actual values and predicted values
- ✓ Regularisation term is used for showing how far is actual value away from the predicted value
- ✓ XGBoost considers many models which are known as base learners for predicting a single value

- ✓ Not all base learners are expected to have bad predictions so after summing up all of them bad predictions canceled out by good prediction

- **<u>Limitations of this work and Scope for Future Work</u>**

  Deep learning algorithms may enhance the prediction of the price of the flight and decrease the test error rate percentage.