AIRLINE TICKET FARE PREDICTION

A PROJECT REPORT

Submitted by TEAM 8

SUSHMA SINDHE – TEAM LEAD

HARSHITHA VEJELLA – STORYTELLING AND DATA VISUALIZATION

PHANIDEEPAK GADDAM – DATA COLLECTION AND PREPROCESSING

LAKSHMI SAI KISHORE SAVARAPU – DATA SCIENTIST

Under the guidance of

Professor Ms. Ardiana Sula

For the Course DSCI-6007-02

DISTRIBUTED AND SCALABLE DATA ENGINEERING

UNIVERSITY OF NEW HAVEN

WEST HAVEN, CONNECTICUT

FALL 2023

TABLE OF CONTENTS

ABSTRACT

The Airline industry is a significant component of the global travel and transportation sector. It encompasses the buying and selling of airline tickets for passenger travel, both domestic and international. Travelers look for best deals and the Airlines try to make profit by selling their tickets. As the industry evolves, building a flight ticket fare prediction model will be a valuable tool for travelers and the airline industry. These models can benefit both consumers looking for the best deals and airlines striving to optimize their pricing strategies.

Airline ticket prices are influenced by a multitude of factors, making them a complex and dynamic pricing challenge. However, with the power of data and machine learning, we can create models that provide accurate fare predictions. As the industry evolves, these models can adapt to changing market dynamics, ultimately benefiting both consumers and airlines.

Data science and machine learning have a wide range of applications across various domains and industries. These technologies have the potential to extract valuable insights, make predictions, and automate tasks, leading to improved decision-making and efficiency. Predicting airline ticket fares is a complex task that involves multiple variables and factors. Data Science and Machine learning techniques can be used to build an effective solution for predicting airline ticket fares.

Dataset has been collected for the above requirement and preprocessing has been performed on it using various python libraries. Feature extraction has been performed to get valuable insights from the dataset. Machine Learning models like Linear Regressor and Random Forest regressor have been build and the models performance has been evaluated and it is observed that the model is able to perform well with a good prediction rate and can be deployed for real time usage.

CHALLENGE

How to predict the price of an Airline ticket based on certain features?

Buying an airline ticket depends on various factors like source, destination, number of stops between source and destination, price range, flight duration, airline company etc. Travelers look for the best deal at a low price with certain specifications while buying a flight ticket. Building a model considering all these factors is a challenging task. The model has to include various requirements like:

What is the source and destination?

What is the price range?

Number of tickets needed?

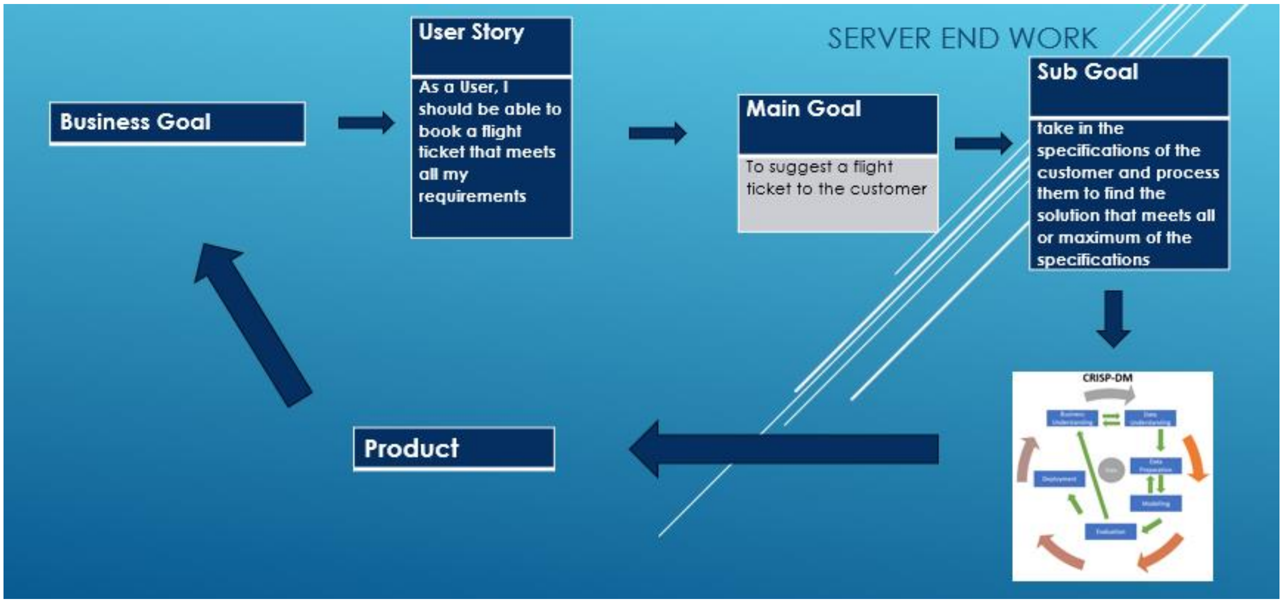
Number of stops in between source and destination?

What is the flight duration?

SOLUTION

Considering all sorts of requirements like the above, a Machine Learning model can be built that can predict the price of an airline ticket. Data Science and Machine Learning techniques have been used to solve these kinds of problems in various domains and industries. Many industries have implemented Data Science and Machine Learning techniques in real time cases and have achieved great yield. Fraud detection, Recommendation systems, E-commerce pricing, Healthcare diagnostics are some examples of real time applications of Data Science and Machine Learning techniques. With boundless number of applications of Data Science and Machine Learning Techniques, it is possible to build an efficient model that can be used in real time for predicting Airline ticket price.

USER STORY



Title: Flight ticket Fare Prediction

User story: As a user, I should be able to get a predicted price of a flight ticket depending on my requirements so that I can make a decision on my travel plans.

Acceptance Criteria:

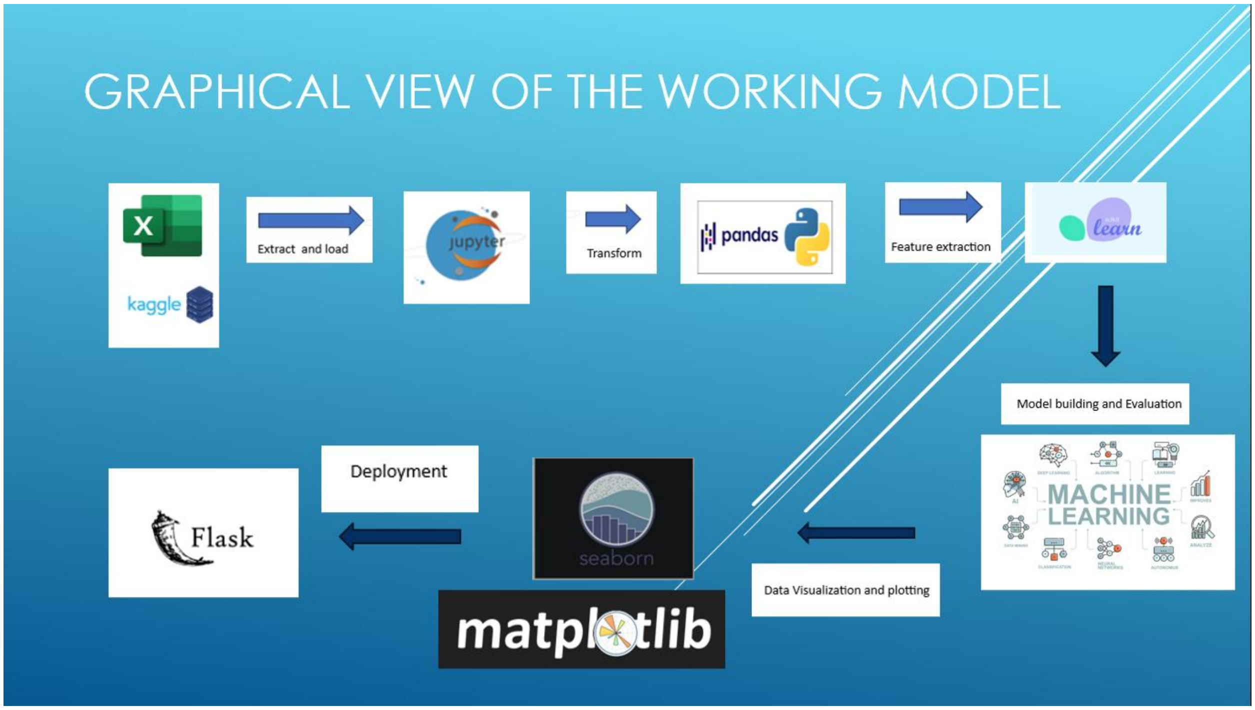
Input Parameters:

Source and destination, Number of stops in between, Flight duration, Airline Company, etc.

Historical data is required for the model building so that the input parameters can be considered while predicting the ticket price and suggesting it to the user.

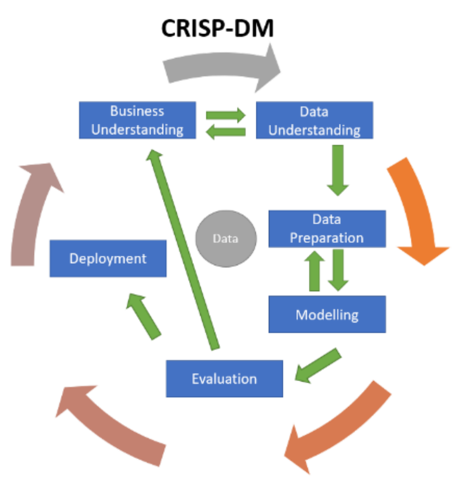
Output: Machine Learning model is built that takes input of all the parameters and predicts the ticket price to the customer.

GRAPHICAL VIEW OF THE WORKING MODEL



CRISP-DM IMPLEMENTATION

CRISP-DM stands for Cross Industry Standard Process.



The steps involved in CRISP-DM Methodology are:

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modelling a solution
5. Evaluation of the model
6. Deployment of the model

We are going to implement CRISP-DM Methodology to the project in order to build a successful working Model.

Business Understanding:

Primary goal:

To predict the price of an Airline ticket based on given inputs such as source, destination, arrival time, departure time, number of stops in between, duration of journey, date of journey etc.

Stakeholder Identification:

This model helps Airline companies as well as Passengers looking for flight tickets. Travel Agencies can also be taken into consideration here.

Scope and Constraints:

The data is generated before the pandemic i.e., 2019. And the sources and destinations are confined to cities in the country India.

Business Questions:

1. Which Airline company flight tickets are expensive?
2. Which Airline company flight tickets are economical?
3. Which location is the most frequent origin and which is the most frequent destination?
4. Which is the busiest route?

DATASET

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

5 V's of Data is checked to ensure right set of data is collected for the problem statement.

Volume: 10683 samples of data.

Variety: Structured Data.

Velocity: Data generated during the pre-covid period.

Veracity: Kaggle is said to be the trusted go to point of information.

Value: Data can be used to estimate domestic flight fares in India.

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302
...	...	...	...	...	...	...	...	...	...	...	...
10678	Air Asia	9/04/2019	Kolkata	Banglore	CCU → BLR	19:55	22:25	2h 30m	non-stop	No info	4107
10679	Air India	27/04/2019	Kolkata	Banglore	CCU → BLR	20:45	23:20	2h 35m	non-stop	No info	4145
10680	Jet Airways	27/04/2019	Banglore	Delhi	BLR → DEL	08:20	11:20	3h	non-stop	No info	7229
10681	Vistara	01/03/2019	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	non-stop	No info	12648
10682	Air India	9/05/2019	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2 stops	No info	11753

10683 rows × 11 columns

Data Understanding:

This is the phase in a data science project where the focus lies on comprehensively exploring and familiarizing oneself with the dataset that will be used for analysis and modelling. It is a crucial step in data mining or machine learning process.

The data.info() function in python is typically used with the pandas library to retrieve information about the data.

```
In [6]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

The data.shape gives the number of columns and rows in the dataset. Columns are the features of the dataset and rows are the samples of it.

```
In [7]: data.shape

Out[7]: (10683, 11)
```

The data.describe function in python is used to generate descriptive statistics of a dataframe. When it is applied to a dataframe, it provides the summary statistics for the numerical columns in the dataset.

```
In [9]: data.describe

Out[9]: <bound method NDFrame.describe of            Airline Date_of_Journey    Source Destination  \
0           IndiGo      24/03/2019  Banglore   New Delhi
1        Air India       1/05/2019   Kolkata    Banglore
2      Jet Airways       9/06/2019     Delhi      Cochin
3           IndiGo      12/05/2019   Kolkata    Banglore
4           IndiGo      01/03/2019  Banglore   New Delhi
...            ...             ...       ...         ...
10678     Air Asia       9/04/2019   Kolkata    Banglore
10679    Air India      27/04/2019   Kolkata    Banglore
10680  Jet Airways      27/04/2019  Banglore       Delhi
10681      Vistara      01/03/2019  Banglore   New Delhi
10682    Air India       9/05/2019     Delhi      Cochin

                       Route Dep_Time  Arrival_Time Duration Total_Stops  \
0                  BLR → DEL    22:20  01:10 22 Mar   2h 50m    non-stop
1      CCU → IXR → BBI → BLR    05:50         13:15   7h 25m     2 stops
2      DEL → LKO → BOM → COK    09:25  04:25 10 Jun      19h     2 stops
3            CCU → NAG → BLR    18:05         23:30   5h 25m      1 stop
4            BLR → NAG → DEL    16:50         21:35   4h 45m      1 stop
...                      ...      ...           ...      ...         ...
10678              CCU → BLR    19:55         22:25   2h 30m    non-stop
10679              CCU → BLR    20:45         23:20   2h 35m    non-stop
10680              BLR → DEL    08:20         11:20       3h    non-stop
10681              BLR → DEL    11:30         14:10   2h 40m    non-stop
10682  DEL → GOI → BOM → COK    10:55         19:15   8h 20m     2 stops

      Additional_Info  Price
0             No info   3897
1             No info   7662
2             No info  13882
3             No info   6218
4             No info  13302
...               ...    ...
10678         No info   4107
10679         No info   4145
10680         No info   7229
10681         No info  12648
10682         No info  11753

[10683 rows x 11 columns]>
```

Data Preparation:

Data Preparation, also known as Data Preprocessing, is a crucial phase in any data science or machine learning project. It involves transforming raw data into a clean, organized format that is suitable for analysis or modeling. Here are key steps involved in data preparation:

Handling missing values, dealing with outliers, encoding categorical variables, feature engineering, train test splitting of data, data transformation, handling skewed data etc.

Identify Missing Data: Determine where data is missing or null in the dataset.

Imputation or Removal: Fill missing values using techniques like mean, median, mode imputation or remove rows/columns with excessive missing data.

Detect Outliers: Identify and handle outliers that might skew analysis or modeling.

Transformation, or Removal: Apply techniques to address outliers based on the context of the data.

Convert Categorical Data: Encode categorical variables into numerical format using techniques like one-hot encoding or label encoding.

Scale Numerical Features: Normalize or scale numerical features to a similar range to prevent dominance of certain features in the model.

Create New Features: Generate new features from existing ones that might improve model performance.

Dimensionality Reduction: Use techniques like PCA or feature selection to reduce the number of features while retaining relevant information.

Split Data: Divide the dataset into training and testing sets to train the model on one portion and validate its performance on another.

Prepare Data for Modeling: Reshape, reformat, or prepare data specifically tailored for the chosen machine learning algorithms.

Normalize Skewed Data: Address skewed distributions in certain columns through techniques like log transformation.

Handling of null values:

In [13]:
```
data.isna().sum()
```

Out[13]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

In [14]:
```
data[data['Route'].isna() | data['Total_Stops'].isna()]
```

Out[14]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
9039	Air India	6/05/2019	Delhi	Cochin	NaN	09:45	09:25 07 May	23h 40m	NaN	No info	7480

In [15]:
```
data.dropna(inplace = True)
```

In [16]:
```
data.isna().sum()
```

Out[16]:
```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              0
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        0
Additional_Info    0
Price              0
dtype: int64
```

Type conversion of features:

In [18]:
```
#Duration
def convert_duration(duration):
    if len(duration.split()) == 2:
        hours = int(duration.split()[0][: -1])
        minutes = int(duration.split()[1][: -1])
        return hours * 60 + minutes
    else:
        return int(duration[: -1]) * 60
```

In [19]:
```
data['Duration'] = data['Duration'].apply(convert_duration)
data.head()
```

Out[19]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	170	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	445	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	1140	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	325	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	285	1 stop	No info	13302

In [20]:
```
# departure and arrival time
data['Dep_Time'] = pd.to_datetime(data['Dep_Time'])
data['Arrival_Time'] = pd.to_datetime(data['Arrival_Time'])
data.dtypes
```

Out[20]:
```
Airline                    object
Date_of_Journey            object
Source                     object
Destination                object
Route                      object
Dep_Time           datetime64[ns]
Arrival_Time       datetime64[ns]
Duration                    int64
Total_Stops                object
Additional_Info            object
Price                       int64
dtype: object
```

In [21]:
```
data['Dep_Time_in_hours'] = data['Dep_Time'].dt.hour
data['Dep_Time_in_minutes'] = data['Dep_Time'].dt.minute
data['Arrival_Time_in_hours'] = data['Arrival_Time'].dt.hour
data['Arrival_Time_in_minutes'] = data['Arrival_Time'].dt.minute
data.head()
```

Deleting unnecessary features:

In [24]:
```
data['Date_of_Journey'].dt.year.unique()
```
Out[24]:
```
array([2019], dtype=int64)
```

In [25]:
```
data['Day'] = data['Date_of_Journey'].dt.day
data['Month'] = data['Date_of_Journey'].dt.month
data.head()
```
Out[25]:

	Airline	Date_of_Journey	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Dep_Time_in_hours	Dep_Time_in_minutes	Arrival_Tim
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	170	non-stop	No info	3897	22	20	
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	445	2 stops	No info	7662	5	50	
2	Jet Airways	2019-09-06	Delhi	Cochin	DEL → LKO → BOM → COK	1140	2 stops	No info	13882	9	25	
3	IndiGo	2019-12-05	Kolkata	Banglore	CCU → NAG → BLR	325	1 stop	No info	6218	18	5	
4	IndiGo	2019-01-03	Banglore	New Delhi	BLR → NAG → DEL	285	1 stop	No info	13302	16	50	

In [26]:
```
data.drop('Date_of_Journey',axis = 1, inplace = True)
data.head()
```

In [30]:
```
# additional info

data['Additional_Info'].value_counts()
```
Out[30]:
```
No info                         8344
In-flight meal not included     1982
No check-in baggage included     320
1 Long layover                    19
Change airports                    7
Business class                     4
No Info                            3
1 Short layover                    1
Red-eye flight                     1
2 Long layover                     1
Name: Additional_Info, dtype: int64
```

In [31]:
```
data.drop('Additional_Info', axis = 1, inplace = True)
```

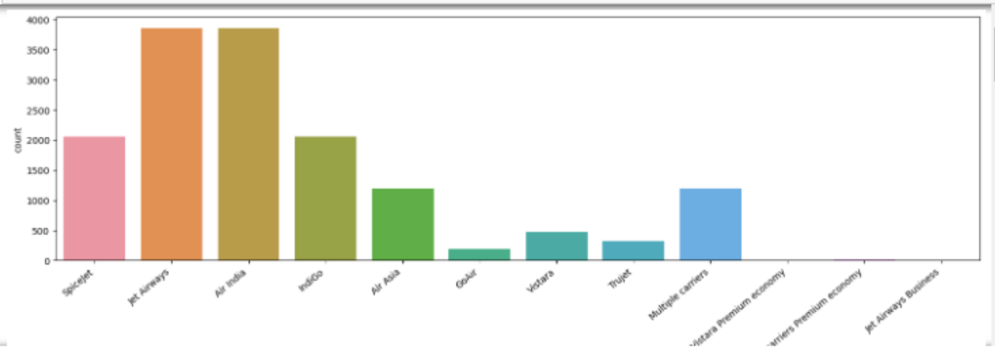
In [27]:
```
#Total stops
data['Total_Stops'].value_counts()
```
Out[27]:
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [28]:
```
data['Total_Stops'] = data['Total_Stops'].map({
    'non-stop' : 0,
    '1 stop' : 1,
    '2 stops' : 2,
    '3 stops' : 3,
    '4 stops' : 4
})
```

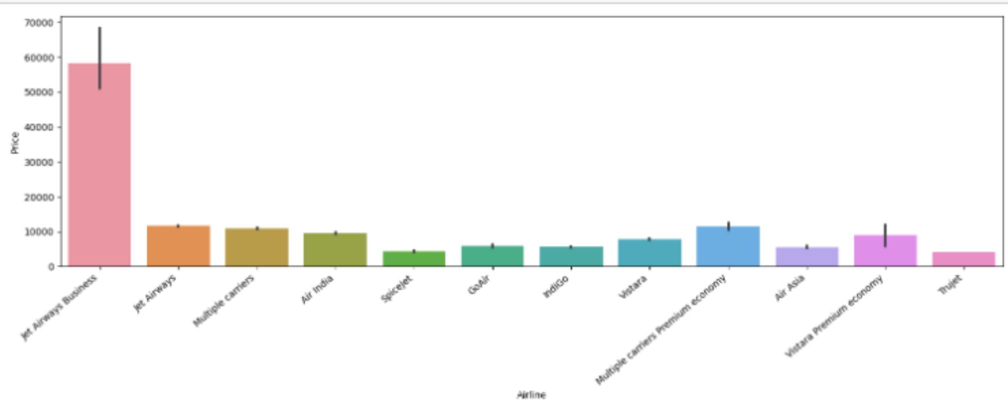
Data Visualization:

It is the process of making graphical plots on the data. It is performed to draw hidden insights about the data.

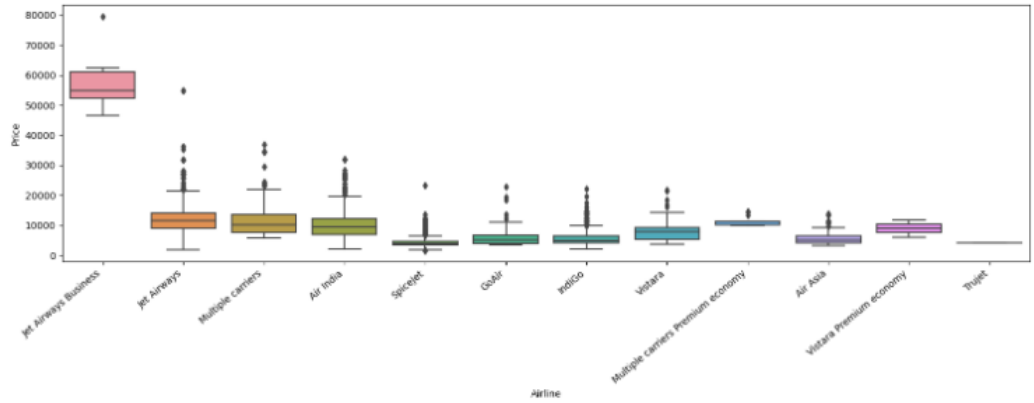
In [34]:
```
for i in ['Airline','Source','Destination','Total_Stops']:
    plt.figure(figsize = (15,6))
    sns.countplot(data = data, x=i)
    ax = sns.countplot(x=i, data = data.sort_values('Price', ascending = True))
    ax.set_xticklabels(ax.get_xticklabels(),rotation = 40, ha = 'right')
    plt.tight_layout()
    plt.show()
    print('\n\n')
```



In [36]:
```
plt.figure(figsize = (15,6))
ax = sns.barplot(x = 'Airline', y = 'Price', data = data.sort_values('Price', ascending = False))
ax.set_xticklabels(ax.get_xticklabels(),rotation = 40, ha = 'right')
plt.tight_layout()
plt.show()
```



```
In [37]: plt.figure(figsize = (15,6))
         ax = sns.boxplot(x = 'Airline', y = 'Price', data = data.sort_values('Price', ascending = False))
         ax.set_xticklabels(ax.get_xticklabels(),rotation = 40, ha = 'right')
         plt.tight_layout()
         plt.show()
```



Splitting of data:

4. Splitting of Dataset

```
In [53]: x = data.drop(columns='Price')
         y = data['Price']
         X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.2, random_state=2)
```

```
In [54]: # Columns of x
         print("Columns of x:")
         print(x.columns)

         # Columns of y (since y is a Series, not a DataFrame)
         print("\nColumn name of y:")
         print(y.name)

         Columns of x:
         Index(['Duration', 'Total_Stops', 'Dep_Time_in_hours', 'Dep_Time_in_minutes',
                'Arrival_Time_in_hours', 'Arrival_Time_in_minutes', 'Day', 'Month',
                'Air India', 'GoAir', 'IndiGo', 'Jet Airways', 'Jet Airways Business',
                'Multiple carriers', 'Multiple carriers Premium economy', 'SpiceJet',
                'Trujet', 'Vistara', 'Vistara Premium economy', 'Source_Chennai',
                'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai', 'Destination_Cochin',
                'Destination_Delhi', 'Destination_Hyderabad', 'Destination_Kolkata',
                'Destination_New Delhi', 'Route_1', 'Route_2', 'Route_3', 'Route_4',
                'Route_5'],
               dtype='object')

         Column name of y:
         Price
```

Filtering the dataset with Standard scale:

Why Preprocessing ??? To increase the efficacy of our predictions.

1. Standard Scaler
   - Mean is 0, variance is 1

```
In [63]: # Scaling of Data with Standard Scale
         from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         x_scaled = scaler.fit_transform(x)
         X_train, X_test, Y_train, Y_test = train_test_split(x_scaled, y)
```

Filtering the dataset with Robust scale:

2. Robust Scaler
    - Similar to Standard Scaler, but uses median and quartiles as opposed to mean and variance
    - Result: Resistant to outliers, just like robust regressors etc

In [70]:
```
# Scaling the data with Robust Scale
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
x_scaled = scaler.fit_transform(x)
X_train, X_test, Y_train, Y_test = train_test_split(x_scaled, y)
```

Modelling a solution:

As the target variable is a continuous variable, Regression models listed below are built using the dataset for the problem statement.

- Linear Regression model
- Decision Tree model
- Random Forest model
- Ordinary Least Squared Model
- Support Vector Regressor Model
- AdaBoost Regressor Model
- Bagging Regressor Model
- Gradient Boost Regressor Model
- XG Boost Regressor Model

```
In [62]: # Support Vector Regressor Model, AdaBoost Regressor Model, Bagging Regressor Model, Gradient Boost Regressor Model, XG Boost Re
from sklearn.svm import SVR
from sklearn.ensemble import AdaBoostRegressor, BaggingRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error, r2_score

SVR = SVR()
ABC = AdaBoostRegressor(n_estimators=100)
BC = BaggingRegressor(n_estimators=100)
GBC = GradientBoostingRegressor(n_estimators=100)
XGB = XGBRegressor(n_estimators=100, seed=555, eval_metric='rmse', use_label_encoder=False)

regressors = [SVR, ABC, BC, GBC, XGB]
regressor_names = ['Support Vector Regression', 'AdaBoost', 'Bagging', 'Gradient Boosting', 'XGBoost']

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

print('5-fold cross-validation:\n')
for reg, name in zip(regressors, regressor_names):
    scores = cross_val_score(reg, X_train, y_train, cv=5, scoring='r2')
    print(f"Train CV R-squared: {scores.mean():.3f} (+/- {scores.std():.3f}) [{name}]")
    reg.fit(X_train, y_train)
    y_pred = reg.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"Test MSE: {mse:.4f}")
    print(f"Test R-squared: {r2:.4f}\n")
```

```
In [57]: # Linear Regression Model
regressor = LinearRegression()
regressor.fit(X_train, Y_train)
predictions = regressor.predict(X_test)
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
r2 = r2_score(Y_test, predictions)
print('R-squared score: {:.2f}'.format(r2))
```

```
R-squared score: 0.61
```

```
In [58]: # Decision Tree Model
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=42)
dtree.fit(X_train, Y_train)
dtree.predict(X_test)
from sklearn.metrics import r2_score
r2 = r2_score(Y_test, predictions)
print('R-squared score: {:.2f}'.format(r2))
```

```
R-squared score: 0.61
```

```
In [59]: # Random Forest Model
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, Y_train)
predictions = rf_regressor.predict(X_test)
r2 = r2_score(Y_test, predictions)
print('R-squared score: {:.2f}'.format(r2))
```

```
R-squared score: 0.81
```

```
In [60]: # Ordinary Least Square Model
import statsmodels.api as sm
import statsmodels
model = sm.OLS(y, x)
results = model.fit()
results.params
```

Evaluation of the models:

The built models are evaluated by calculating the R squared value and Mean Absolute Error.

The coefficient of determination, often denoted as R-squared (R^2), is a statistical measure that represents the proportion of variance in the dependent variable (target) that is explained by the independent variables (features) in a regression model.

In simple terms, R-squared measures how well the independent variables explain the variability of the dependent variable. It ranges between 0 and 1, with:

0 indicating that the model does not explain any variance in the target variable.

1 indicating that the model perfectly explains the variance in the target variable.

In the context of regression analysis, when you fit a regression model to a dataset, the R-squared value tells you how well the model fits the observed data.

The R-squared values for the models are shown below:

Model	R^2 Without Preprocessing	R^2 after preprocessing with Standard Scale	R^2 after preprocessing with Robust Scale
Linear Regressor	0.61	0.64	0.60
Decision Tree	0.61	0.64	0.60
Random Forest	0.81	0.85	0.84
Ordinary Least Square	0.62	0.12	0.62
Support Vector Regressor	0.15	0.70	0.07
Ada Boost Regressor	0.40	0.32	0.25
Bagging Regressor	0.81	0.80	0.82
Gradient Boost Regressor	0.80	0.80	0.77
XG Boost Regressor	0.86	0.84	0.84

Mean Absolute Error was also calculated for the models.

Mean Absolute Error (MAE) is a metric used to evaluate the accuracy of a predictive model, often in the context of regression analysis. It measures the average absolute difference between the actual and predicted values. The formula for calculating MAE is:

$$MAE = (1/n)\ \Sigma(i=1 \text{ to } n)\ |y\_i - \hat{y}\_i|$$

y_i = prediction, ŷ_i = True value, n = total number of data points

In simpler terms, MAE is the average of the absolute differences between the actual and predicted values. It provides a straightforward and easy-to-interpret measure of the average magnitude of errors in the predictions.

A lower MAE indicates better accuracy, as it means that, on average, the model's predictions are closer to the actual values. MAE is less sensitive to outliers compared to some other metrics like Mean Squared Error (MSE), making it a good choice when dealing with datasets that may contain extreme values.

The Mean Absolute Error values for the models are shown below:

Model	Mean Absolute Error without Preprocessing	Mean Absolute Error after preprocessing with standard scale	Mean Absolute Error after preprocessing with Robust Scale
Linear Regressor	0.45	0.38	0.42
Decision Tree	0.43	0.40	0.41
Random Forest	0.17	0.11	0.13
Ordinary Least Square	0.41	0.85	0.35
Support Vector Regressor	0.88	0.27	0.91
Ada Boost Regressor	0.62	0.68	0.75
Bagging Regressor	0.15	0.19	0.15
Gradient Boost Regressor	0.13	0.18	0.20
XG Boost Regressor	0.10	0.14	0.14

Deployment of the model:

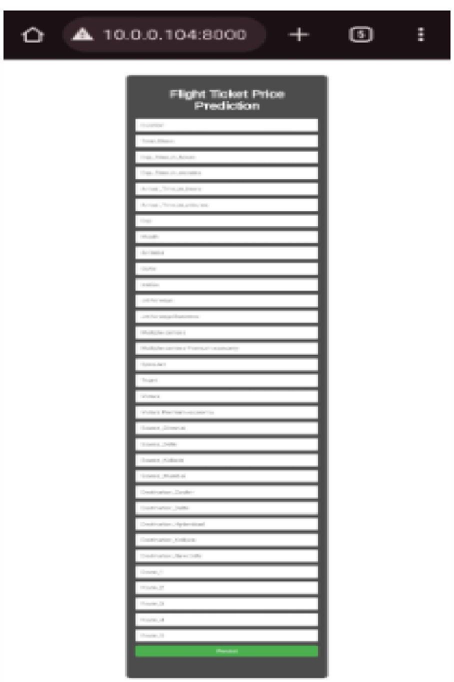
Deploying a model involves making it available for use in a production environment where it can generate predictions or perform the intended task.

Of all the models built, from evaluation it is concluded that XG Boost Regressor model performed better than all the other models. So, deploying that model will give better predictions and more business value to the end users.

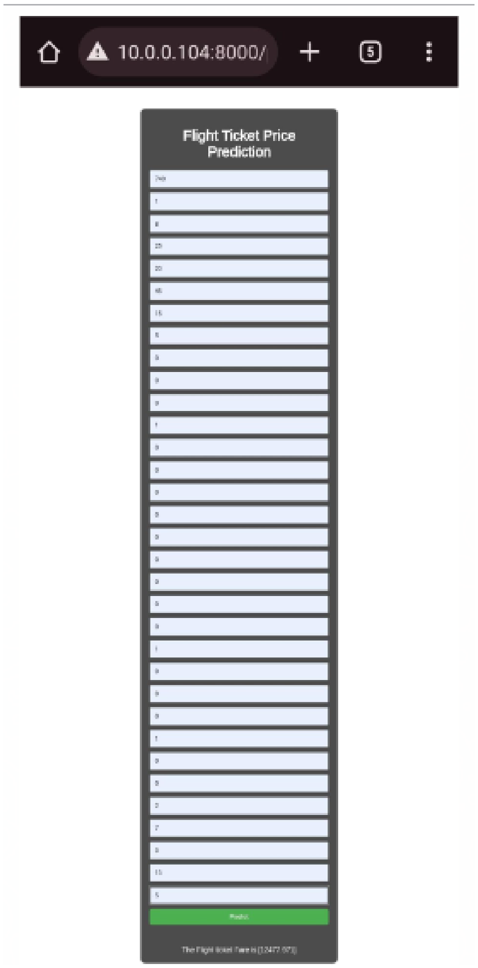
Flask is a popular web framework in Python used for building web applications, including deploying machine learning models as web services or APIs.

```
In [ ]: import numpy as np
        from flask import Flask, request, jsonify, render_template
        import pickle

        # Create flask app
        flask_app = Flask(__name__)
        model = pickle.load(open("Model.pkl", "rb"))

        @flask_app.route("/")
        def Home():
            return render_template("index.html")

        @flask_app.route("/predict", methods = ["POST"])
        def predict():
            float_features = [float(x) for x in request.form.values()]
            features = [np.array(float_features)]
            prediction = model.predict(features)
            return render_template("index.html", prediction_text = "The Flight ticket Fare is {}".format(prediction))

        if __name__ == "__main__":
            flask_app.run(host='0.0.0.0', port=8000)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://10.0.0.104:8000/ (Press CTRL+C to quit)
10.0.0.104 - - [03/Dec/2023 13:34:26] "GET / HTTP/1.1" 200 -
10.0.0.104 - - [03/Dec/2023 13:34:27] "GET /favicon.ico HTTP/1.1" 404 -
10.0.0.104 - - [03/Dec/2023 13:36:44] "POST /predict HTTP/1.1" 200 -
```

The http address: http://10.0.0.104:8000/ given by the flask environment is where the en d users can give the input to get the output.



Feeding inputs on the page will give the predicted output. In this case, the output is the flight ticket price in Indian rupees.



INFERENCE

The XG Boost Machine Learning model gave better performance than the other regressor models. After deploying the model using Flask web server, the model gave required outputs and met all the business requirements. This model can be used to get the flight ticket price by providing the inputs to it through the flask web application.

FUTURE SCOPE

The regressor model performance can be evaluated with other performance metrices.

A Custom ensemble model (Super Learner) model can be built using multiple models to improve the performance of the model and get better results from it.

The model is confined to only domestic flights in India. Models can be built to predict ticket prices for international flights so that when deployed, the model can be used by international airlines and passengers.

REFERENCES

- https://youtu.be/WjLou_0YVMo?si=O_MsRoI5q4ooTXeC
- https://youtu.be/46bRwRvW0PI?si=KIchjcJvDOZkfBMq
- https://youtu.be/KFPX0FLhscs?si=VSFVpvoKYssh3epv
- https://www.kaggle.com/datasets/nikhilmittal/flight-fare-prediction-mh/code