Topic: Flight Price Prediction using AI/ML Algorithms

By: Musaddik Maulavi

Problem Statement

Given past data on flight attributes such as airline, date of journey, source, destination, route, departure and arrival times, duration, and total stops, along with corresponding ticket prices, the objective is to develop predictive models capable of accurately forecasting flight ticket prices. By applying machine learning algorithms and regression techniques, the project aims to assist airlines and travelers in making informed decisions regarding ticket purchases and revenue management strategies in the dynamic aviation industry.

INTRODUCTION

Accurate pricing techniques are essential for airlines to sustain profit and satisfy customer needs in the highly competitive and dynamic aviation business. Artificial intelligence (AI) and machine learning are changing the way organizations examine large information to get insights and make choices in recent years. Predicting flight pricing is one such use, a difficult undertaking impacted by a wide range of variables such as demand variations, seasonal patterns, fuel prices, and route attractiveness. The goal of this research is to use artificial intelligence (AI) to create a prediction model that can correctly estimate flight costs. Our goal is to develop a strong algorithm that can give airlines and travelers useful insights into future price patterns by utilizing past pricing data in conjunction with pertinent contextual information. With this project, I hope to solve the issues related to price uncertainty in the aviation sector and provide stakeholders with the knowledge they need to make better decisions about revenue management tactics and ticket sales. In this report, I will delve into the methodologies employed, the dataset utilized, the preliminary analysis conducted, and the results obtained through our flight price prediction model. Furthermore, I will discuss the implications of our findings and potential avenues for future research in this dynamic field.

DATASET

The dataset being used for this project includes a variety of attributes related to flight travel, including the airline, the date of the trip, the source and destination airports, the flight route, the times of departure and arrival, the duration of the flight, the total number of stops, any additional information about the flight, and the associated ticket prices. This dataset attempts to solve the problem of airline ticket pricing prediction. This is a crucial responsibility for airlines and travelers alike since precise pricing forecast helps consumers make well-informed ticket purchases and allows airlines to maximize revenue management techniques.

The original source of this dataset is Kaggle, a platform for sharing and discovering datasets, as it is commonly used by data science practitioners and researchers. However, it's crucial to ensure the dataset complies with any licensing or usage agreements specified by its original source.

In order to extract data, it may be necessary to gather and combine information from a variety of sources, such as flight booking websites, online travel agents, and airline databases. To guarantee consistency and quality, this procedure could need preprocessing, standardization, and data cleansing. To improve the model's prediction power, additional attempts may be taken to enrich the dataset by adding pertinent external data sources, such as meteorological conditions, economic indicators, or seasonal trends.

This project aims to create a strong predictive model that can accurately forecast flight prices by utilizing a large dataset and cutting-edge machine learning techniques. This will help travelers and airlines make better decisions in the ever-changing aviation industry.

PRELIMINARY ANALYSIS

In the preliminary analysis phase, several key preprocessing steps were undertaken to prepare the dataset for further exploration and modeling in our flight price prediction project. Initially, I addressed inconsistencies in the 'Duration' column by defining a function to split the duration into hours and minutes, ensuring uniformity across the dataset. Furthermore, missing minutes were appended where necessary to maintain data integrity. Following this, the 'Date_of_Journey' column was converted to a datetime format, facilitating the extraction of day, month, and year components. These components were subsequently segregated into individual columns, providing finer granularity for temporal analysis.

Moreover, to enhance the quality of our dataset, observations with missing values in crucial columns such as 'Route', 'Total_Stops', and 'Price' were removed, resulting in a refined dataset ready for analysis. Furthermore, an examination of unique values within categorical columns such as 'Airline', 'Source', 'Destination', 'Additional_Info', and 'Total_Stops' was conducted. This exploration revealed the diverse range of categories within each feature, highlighting the complexity of the dataset and emphasizing the need for robust modeling techniques capable of capturing such variability.

Overall, these preprocessing steps laid a solid foundation for subsequent exploratory analysis and predictive modeling endeavors. By ensuring data consistency, completeness, and relevance, I aimed to minimize biases and maximize the effectiveness of our flight price prediction model.

METHODS

This project developed predictive models that could reliably estimate ticket prices by using a variety of regression methods to solve the flight price prediction problem. The algorithms used were deemed appropriate for the given job and shown efficacy in capturing the intricate correlations between the target variable, price, and the input data.

Firstly, I utilized the Linear Regression algorithm as a baseline model to establish a benchmark for performance evaluation. The Linear Regression model assumes a linear relationship between the input features and the target variable and aims to minimize the mean squared error between the observed and predicted prices. Our implementation of Linear Regression yielded an accuracy score of 41% on the test dataset. Additionally, the mean squared error (MSE), root mean squared error (RMSE), and R-squared (R2) metrics were computed to evaluate the model's performance.

Subsequently, I explored the Random Forest Regressor algorithm, a powerful ensemble learning technique that combines multiple decision trees to improve predictive accuracy. Random Forests are particularly well-suited for handling nonlinear relationships and complex datasets, making them an ideal choice for our flight price prediction task. Our Random Forest Regressor model achieved an accuracy score of 83.08% on the test dataset, outperforming the Linear Regression baseline. Furthermore, the MSE, RMSE, and R2 metrics provided valuable insights into the model's predictive performance.

Lastly, I implemented the Decision Tree Regressor algorithm, which constructs a binary tree structure to partition the feature space and predict the target variable. Decision trees offer interpretability and flexibility, making them suitable for exploratory analysis and model understanding. Our Decision Tree Regressor model yielded an accuracy score of 77.97% on the test dataset. Additionally, the computed MSE, RMSE, and R2 metrics provided comprehensive assessments of the model's accuracy and generalization capabilities.

With everything taken into account, our goal in using and contrasting several regression techniques was to find the best model for predicting travel prices while also learning more about the underlying relationships and trends in the dataset. These findings, which offer insightful information about each algorithm's fit for the given task and its projected performance, are an essential part of our project report.

RESULTS

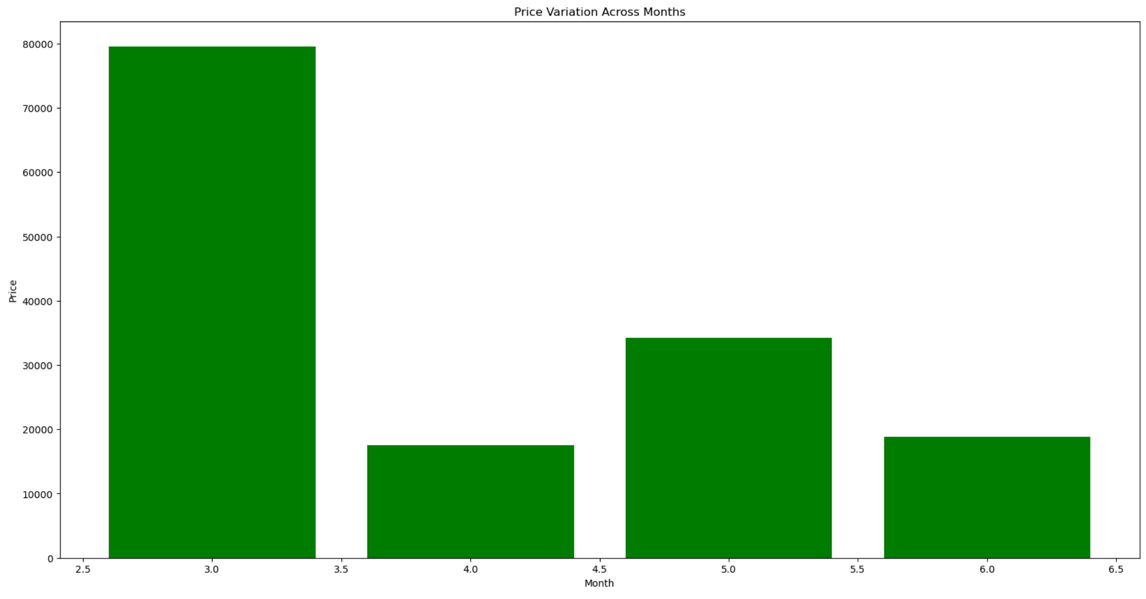
In assessing the results of our flight price prediction models, I employed a comprehensive set of accuracy metrics, carefully selected target variables, and optimized hyperparameters to ensure the adequacy and reliability of our findings.

The choice of accuracy metrics, including mean squared error (MSE), root mean squared error (RMSE), and R-squared (R2), provided a holistic evaluation of each model's performance, capturing both predictive accuracy and goodness of fit. These measurements played a crucial role in measuring the degree of difference between actual and anticipated prices, hence enabling thorough comparisons between various regression methods.

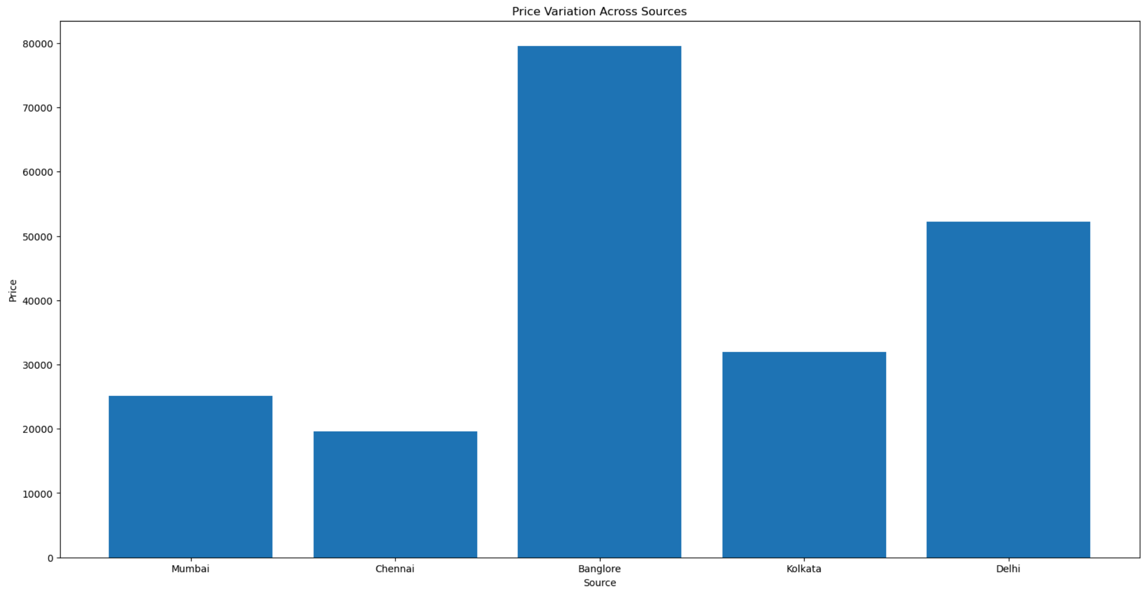
In addition, price was chosen as the target variable due to its importance to travelers and airlines alike, as well as its centrality to the issue area. Our models were designed to help stakeholders make well-informed decisions about revenue management tactics and ticket purchases by providing realistic pricing predictions.

FIGURES

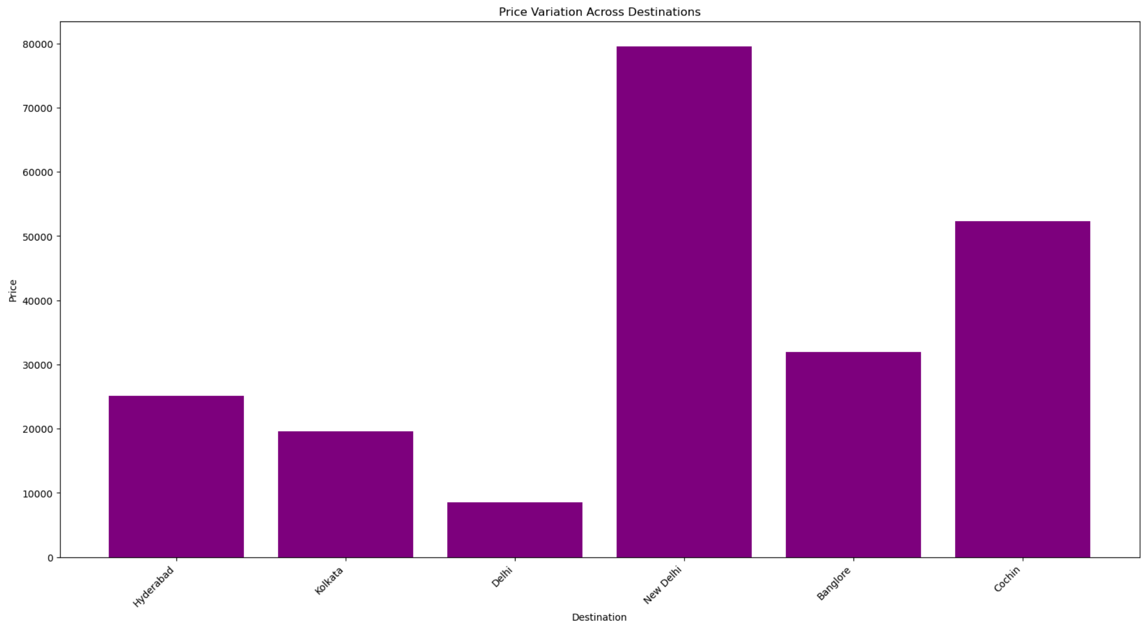
A. The below plot illustrates Price Variation Accross Months (March to June)



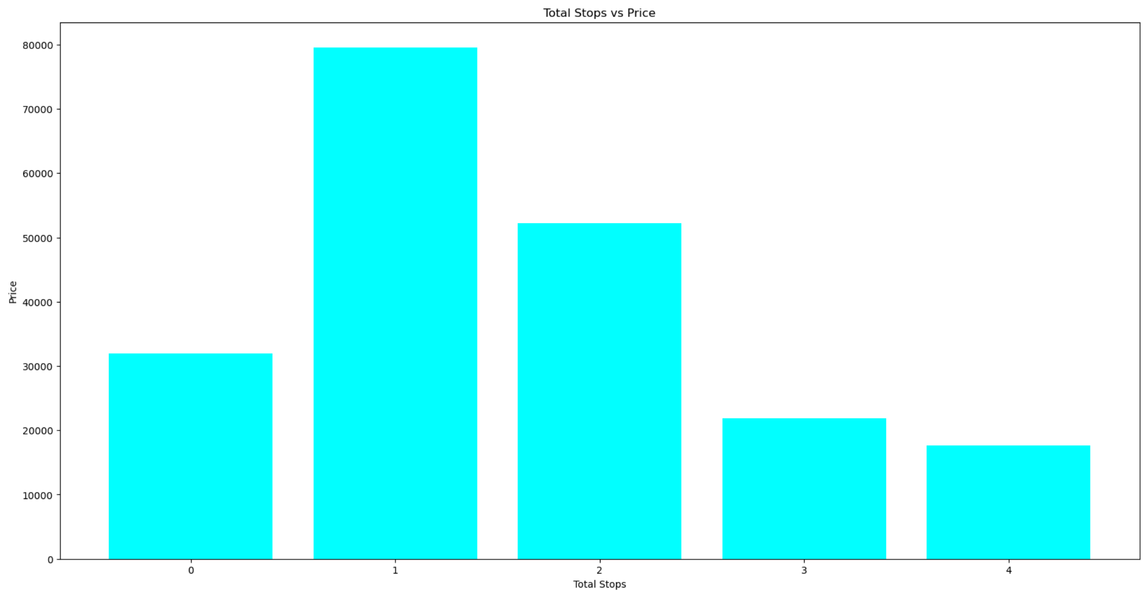
B. The below plot illustrates Price Variation Accross Sources



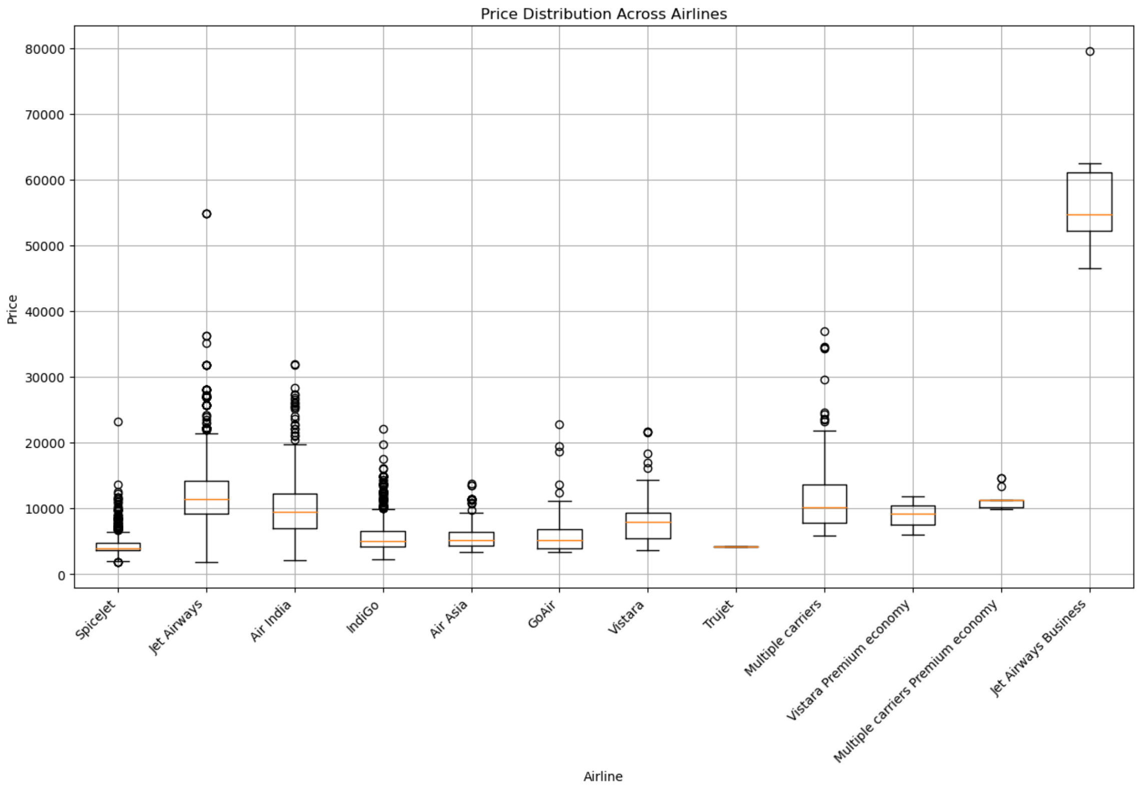
C. The below plot illustrates Price Variation Accross Destinaton



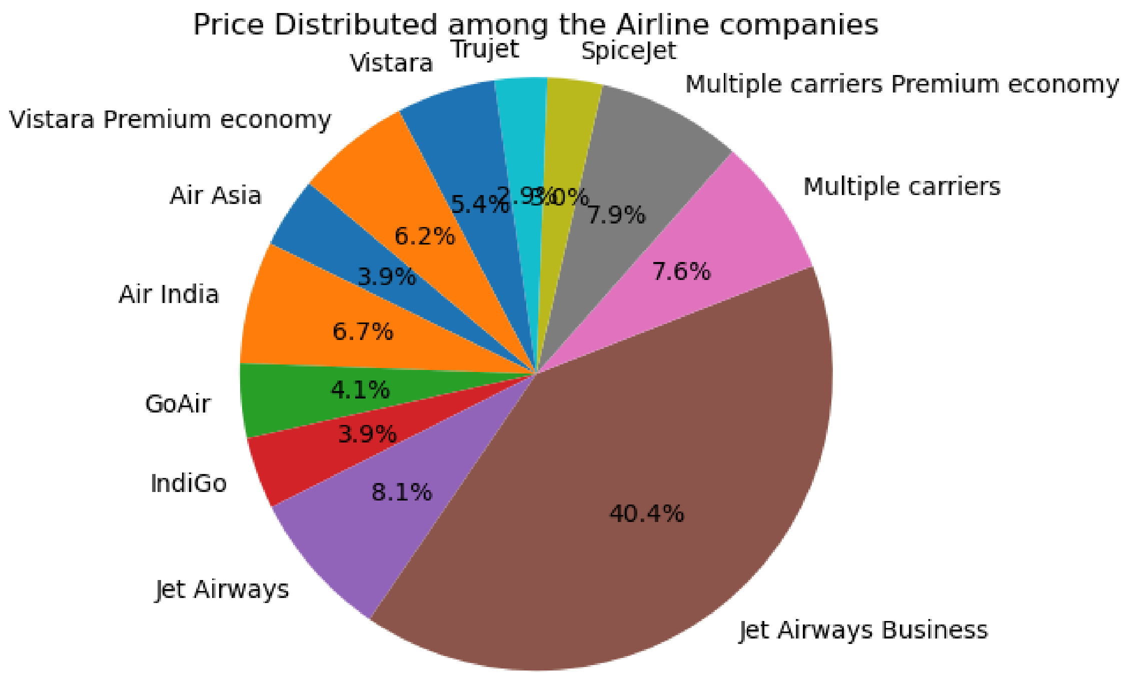
D.The below bar-plot illustrates Price Variation Accross Total Number of Stops



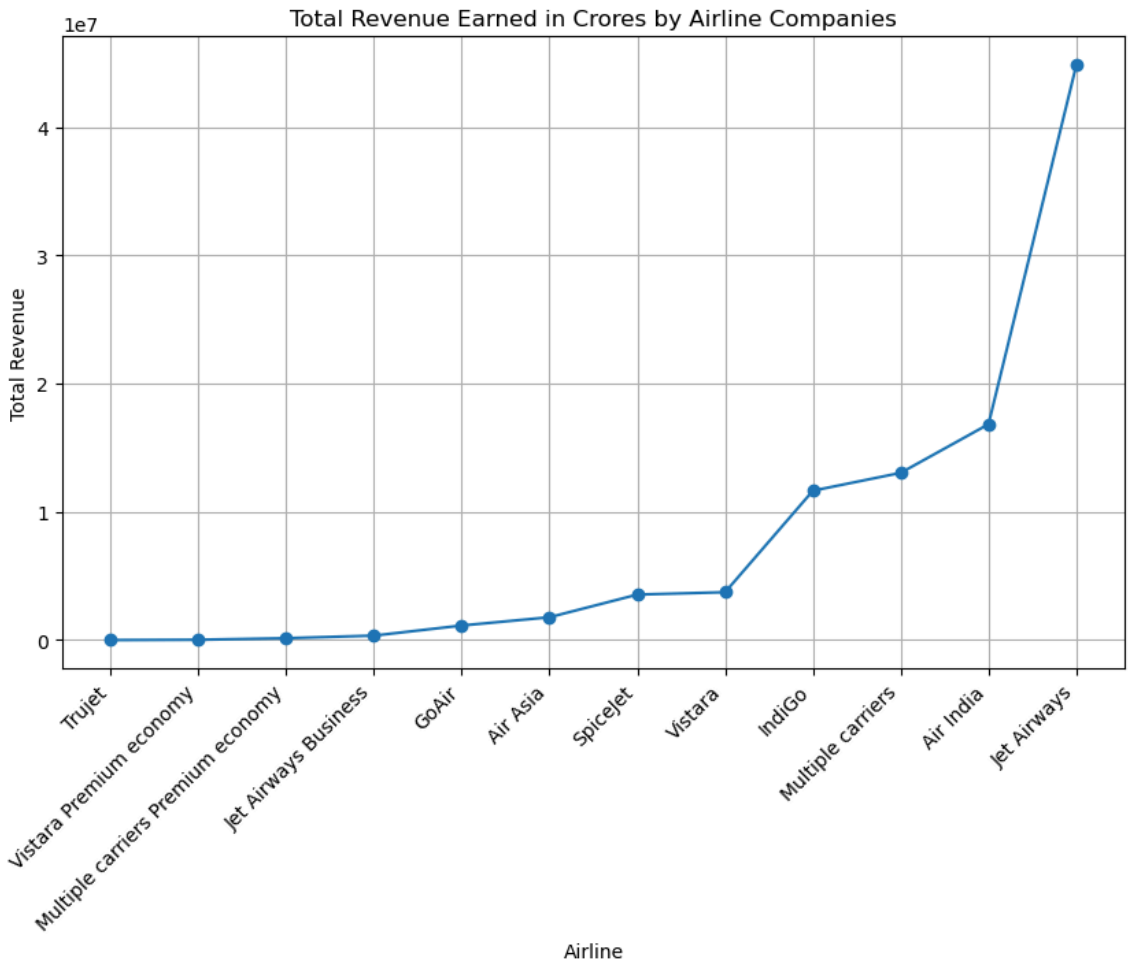
E. The below box-plot illustrates the Price Range of Different Airline companies.



F. The below Pie Chart illustrates total percentage of revenue earned by Airline companies



G. The below line graph illustrates Revenue earned by Airlines in Crores



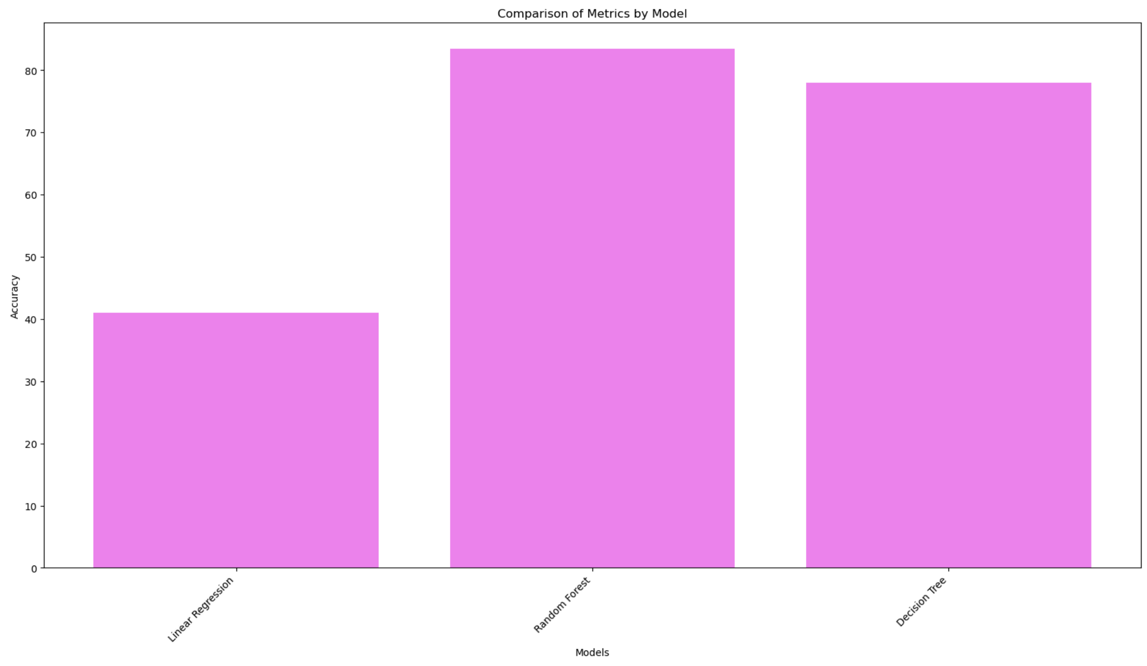
CONCLUSION

This study of flight price prediction resulted in significant findings on the complex processes involved in predicting ticket costs in the aviation sector. By creating and assessing prediction models using different regression techniques, I have tackled the main goals mentioned in our problem statement.

This study confirms that it is possible to use machine learning algorithms to reliably forecast the cost of airline tickets using historical data that includes important parameters like the airline, the dates of the travel, the route, and the length of the flight. I have shown that a variety of regression methods, such as Random Forest Regressor, Decision Tree Regressor, and Linear Regression, are effective in capturing the complex relationships that exist between input variables and ticket pricing.

Nonetheless, this analysis serves as a valuable foundation for further research and application in the evolving landscape of the aviation industry.

As you can see in the graph below, the accuracy of Decision Tree and Random Forest Algorithm is way more than Linear Regression. Therefore the Random Forest Algorithm is best algoritm for prediction FLight prices among the three algorithms used.



Code

In [38]:
```
# Importing all the necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder
```

In [39]:
```python
# Importing the dataset (2 CSV Files)
train_data = pd.read_csv("Data_Train.csv")
test_data = pd.read_csv("Test_set.csv")
```

In [40]:
```python
# Concatinating the two CSV files
proj_data = pd.concat([train_data, test_data])
```

In [41]:
```python
# Displaying few rows of dataset
proj_data.head()
```

Out[41]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Tota
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	n
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	

Data Exploration

In [42]:
```python
# Displaying the dimensions of our dataset
proj_data.shape
```

Out[42]:
```
(13354, 11)
```

In [43]:
```python
proj_data.head()
```

Out[43]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Tota
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	n
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	

In [44]:

```python
# Printing Duration column
print(proj_data['Duration'])
```

```
0        2h 50m
1        7h 25m
2           19h
3        5h 25m
4        4h 45m
          ...
2666    23h 55m
2667     2h 35m
2668     6h 35m
2669    15h 15m
2670    14h 20m
Name: Duration, Length: 13354, dtype: object
```

Data Manipulation

In [45]:

```python
# Function to separate hour and minute from the Duration column.
def minutes(duration):

    duration_split = duration.split()

    if len(duration_split) == 1:
        duration += ' 00m'

    return duration

proj_data['Duration'] = proj_data['Duration'].apply(minutes)
proj_data.head()
```

Out[45]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Tota
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	n
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h 00m	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	

In [46]:

```
# Function to split duration into hours and minutes
def split_duration(duration):
    parts = duration.split()
    if len(parts) >= 2:
        hours = int(parts[0][:-1])
        minutes = int(parts[1][:-1])
        return hours, minutes
    else:
        # Return default values or handle the error as needed
        return None, None

# Apply the function to the DataFrame
proj_data[['Hours', 'Minutes']] = proj_data['Duration'].apply(lambda x: pd.Series(s

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

In [47]:

```
proj_data.head()
```

Out[47]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Total_Stops	A
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	non-stop	
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	2 stops	
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	2 stops	
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	1 stop	
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	1 stop	

In [48]:
```
# Converting the Date into to correct '%d/%m/%Y' format
proj_data['Date_of_Journey'] = pd.to_datetime(proj_data['Date_of_Journey'], format=

# Extract day, month, and year into separate columns
proj_data['Day'] = proj_data['Date_of_Journey'].dt.day
proj_data['Month'] = proj_data['Date_of_Journey'].dt.month
proj_data['Year'] = proj_data['Date_of_Journey'].dt.year
```

Data Cleaning

In [49]:
```
# Removing column 'Date_of_Journey', as we have already separated day, month and ye
# And removing Arrival Time and Departure Time as it will not help our model for pr
proj_data.drop(columns = ['Date_of_Journey','Arrival_Time','Dep_Time'], axis = 1, i
```

In [50]:
```
# Checking what all columns are there in 'Hours' column.
proj_data['Hours'].unique()
```

Out[50]:
```
array([ 2,  7, 19,  5,  4, 15, 21, 25, 13, 12, 26, 22, 23, 20, 10,  6, 11,
        8, 16,  3, 27,  1, 14,  9, 18, 17, 24, 30, 28, 29, 37, 34, 38, 35,
       36, 47, 33, 32, 31, 42, 39, 41, 40], dtype=int64)
```

In [51]:
```
# Checking total number of NA values in each column.
proj_data.isna().sum()
```

Out[51]:
```
Airline            0
Source             0
Destination        0
Route              1
Total_Stops        1
Additional_Info    0
Price           2671
Hours              0
Minutes            0
Day                0
Month              0
Year               0
dtype: int64
```

In [52]:
```
proj_data.shape
```

Out[52]:
```
(13354, 12)
```

In [53]:
```
# Dropping NA values of rows Route, Total Stops and Price
proj_data = proj_data.dropna(subset=['Route', 'Total_Stops', 'Price'])
```

In [54]:
```
proj_data.head()
```

Out[54]:

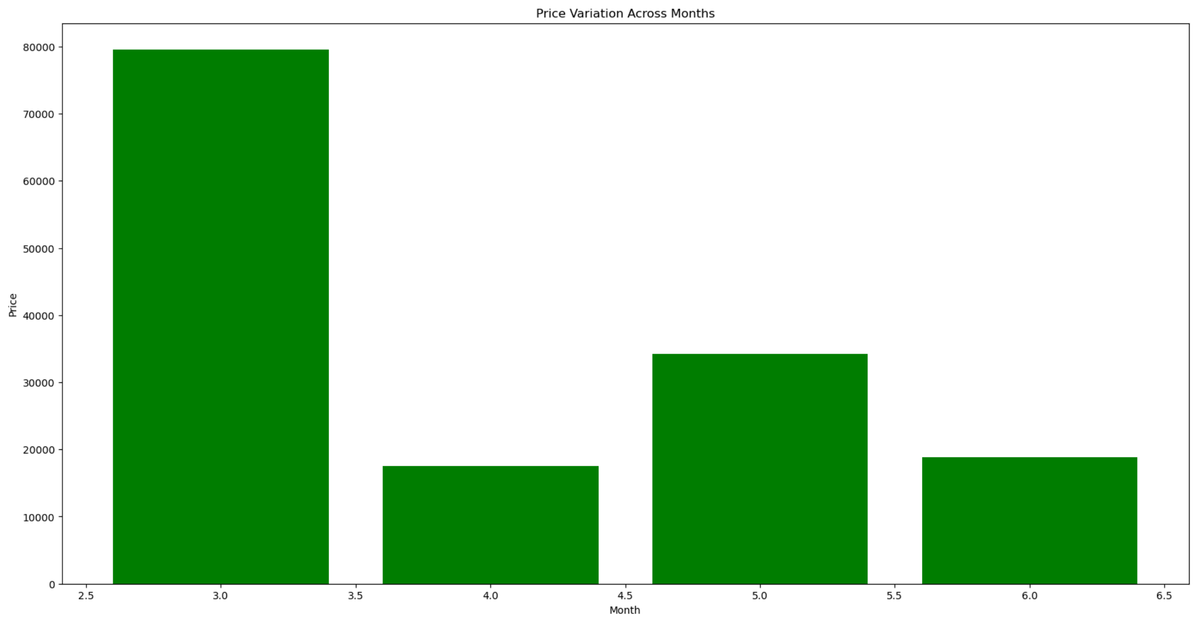
	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Hours	Minutes	D
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897.0	2	50	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662.0	7	25	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882.0	19	0	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218.0	5	25	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302.0	4	45	

In [55]:
```
proj_data.shape
```

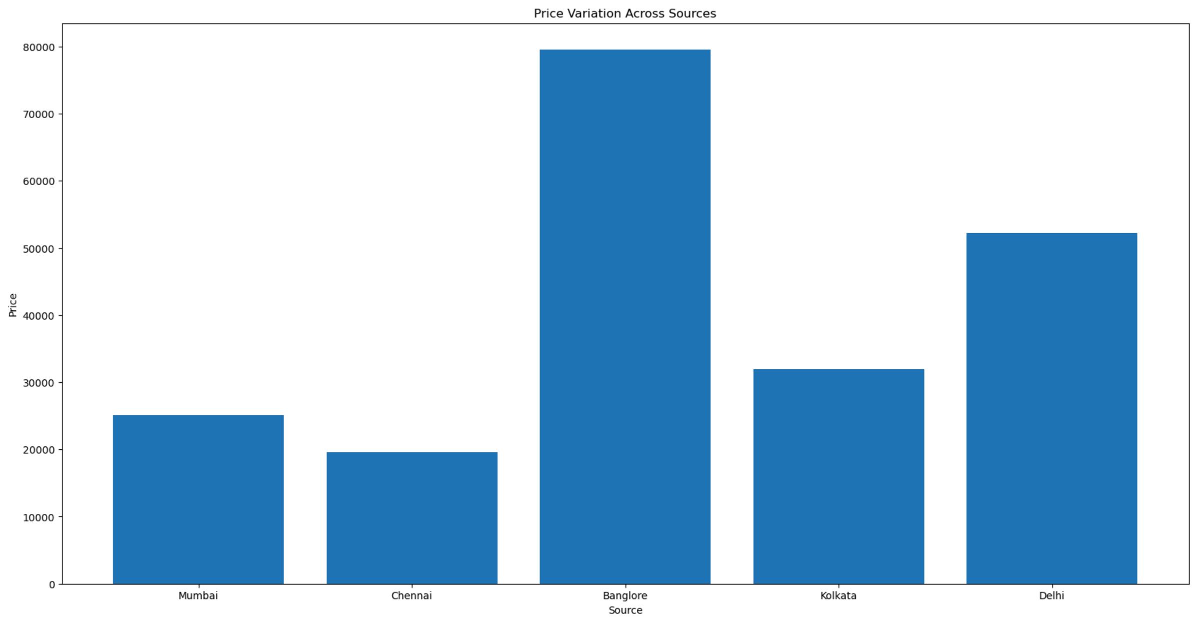
Out[55]:
```
(10682, 12)
```

Data Visualization

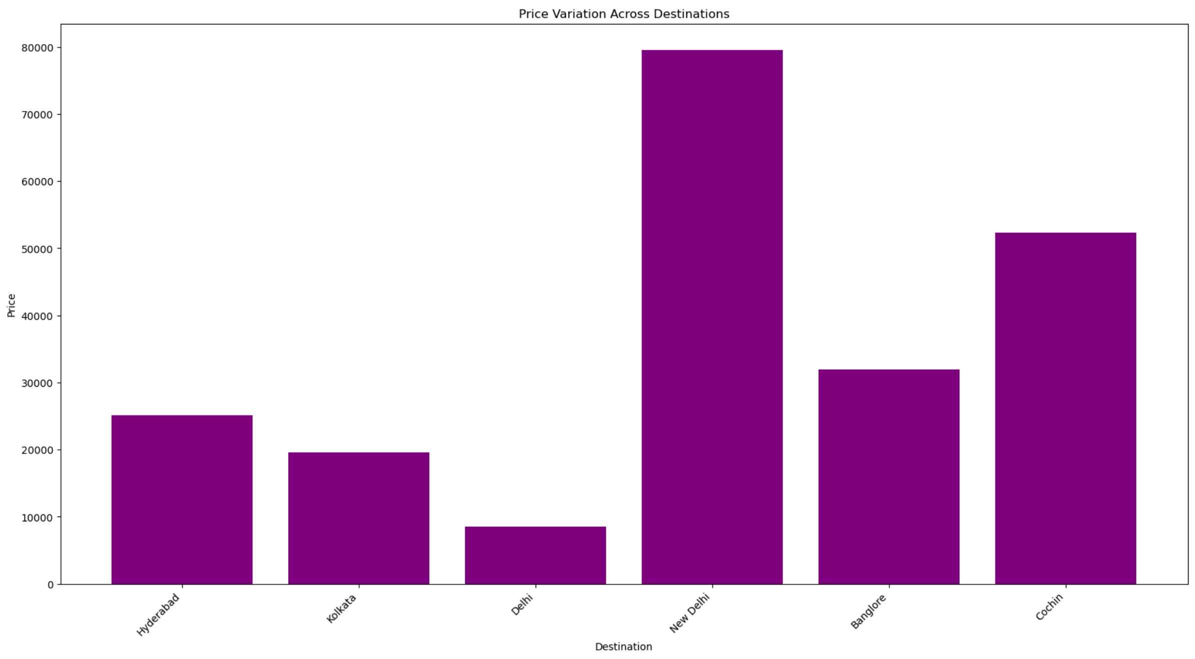
In [56]:
```
# Plotting graph for Price Variation Across Months
plt.figure(figsize=(20, 10))
plt.bar(proj_data['Month'], proj_data['Price'],color='green')
plt.title('Price Variation Across Months')
plt.xlabel('Month')
plt.ylabel('Price')
plt.show()
```



In [57]:
```
# Sorting the data by 'Price'
sorted_data = proj_data.sort_values('Price')
# Plotting graph for Price Variation Across Sources
plt.figure(figsize=(20, 10))
plt.bar(sorted_data['Source'], sorted_data['Price'])
plt.title('Price Variation Across Sources')
plt.xlabel('Source')
plt.ylabel('Price')
plt.show()
```



In [58]:
```
# Plotting graph for Price Variation Across Destinations
plt.figure(figsize=(20, 10))
plt.bar(sorted_data['Destination'], sorted_data['Price'],color='purple')
plt.title('Price Variation Across Destinations')
plt.xlabel('Destination')
plt.ylabel('Price')
plt.xticks(rotation=45, ha='right')  # Rotating x-axis labels for better readabilit
plt.show()
```



In [59]:
```
# Define a dictionary to map unique values to specific numbers in Total Stops colum
# By doing this only integer value will be stored in Total Stops column.
value_mapping = {'non-stop':0, '1 stop':1,'2 stops':2, '3 stops':3, '4 stops':4}

# Replace unique values in the column with specific numbers

proj_data['Total_Stops'] = proj_data['Total_Stops'].replace(value_mapping)
```

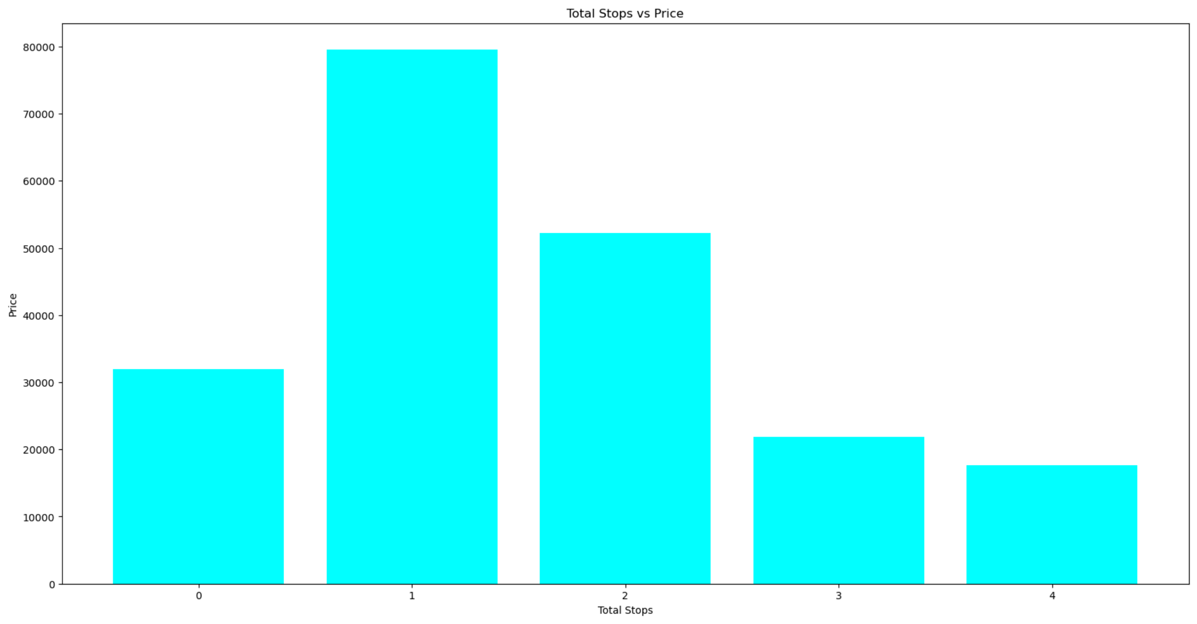
In [60]:
```
# Printing few rows of column to check if we have
proj_data.head()
```

Out[60]:

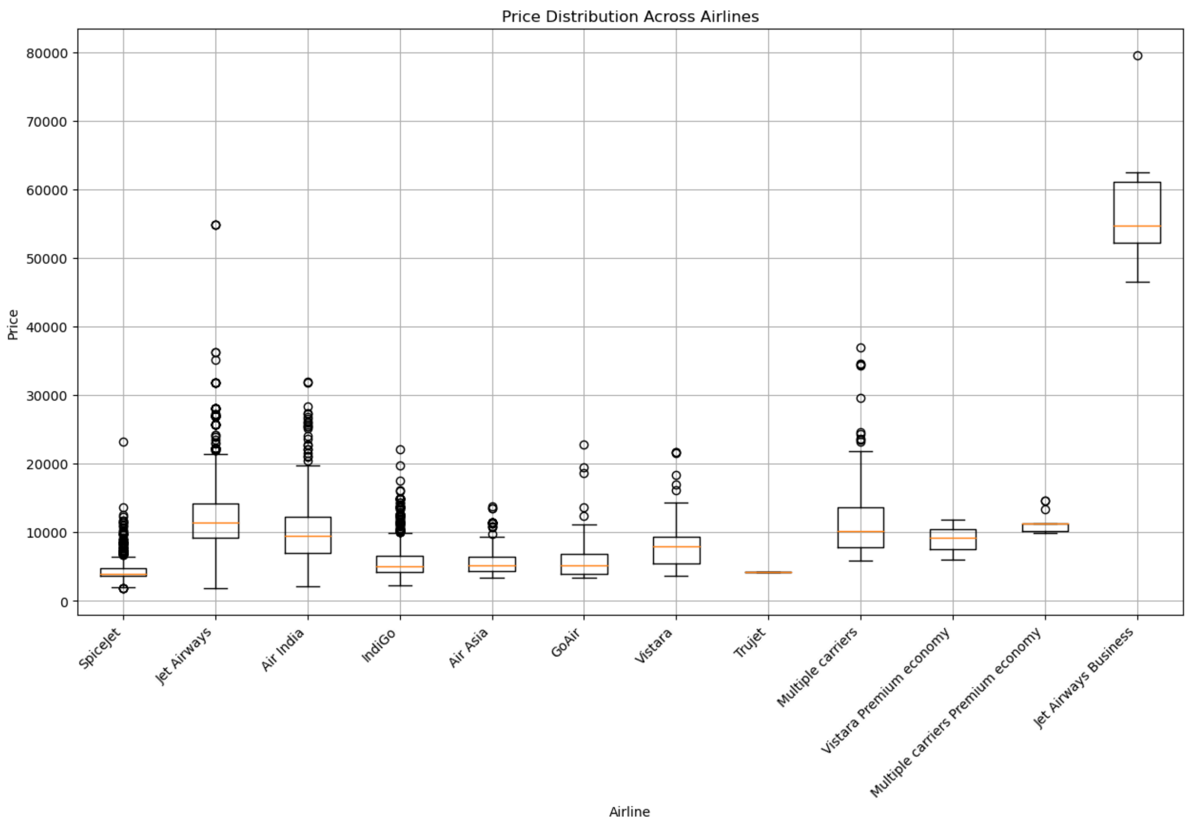
	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Hours	Minutes	D
0	IndiGo	Banglore	New Delhi	BLR → DEL	0	No info	3897.0	2	50	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2	No info	7662.0	7	25	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2	No info	13882.0	19	0	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	1	No info	6218.0	5	25	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	1	No info	13302.0	4	45	

In [61]:

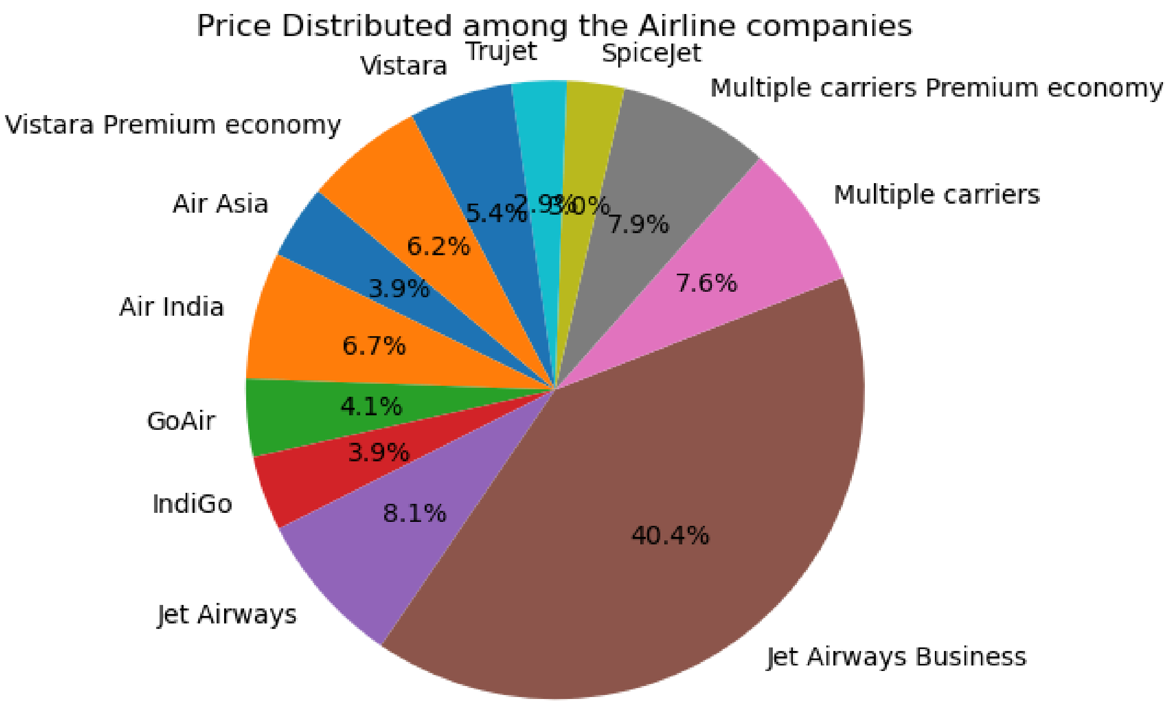
```python
# Plotting the graph for Total Stops and their corresponding price.
# Assuming proj_data is your DataFrame
proj_data_sorted = proj_data.sort_values('Price')

plt.figure(figsize=(20, 10))
plt.bar(proj_data_sorted['Total_Stops'], proj_data_sorted['Price'], color='cyan')
plt.xlabel('Total Stops')
plt.ylabel('Price')
plt.title('Total Stops vs Price')
plt.show()
```



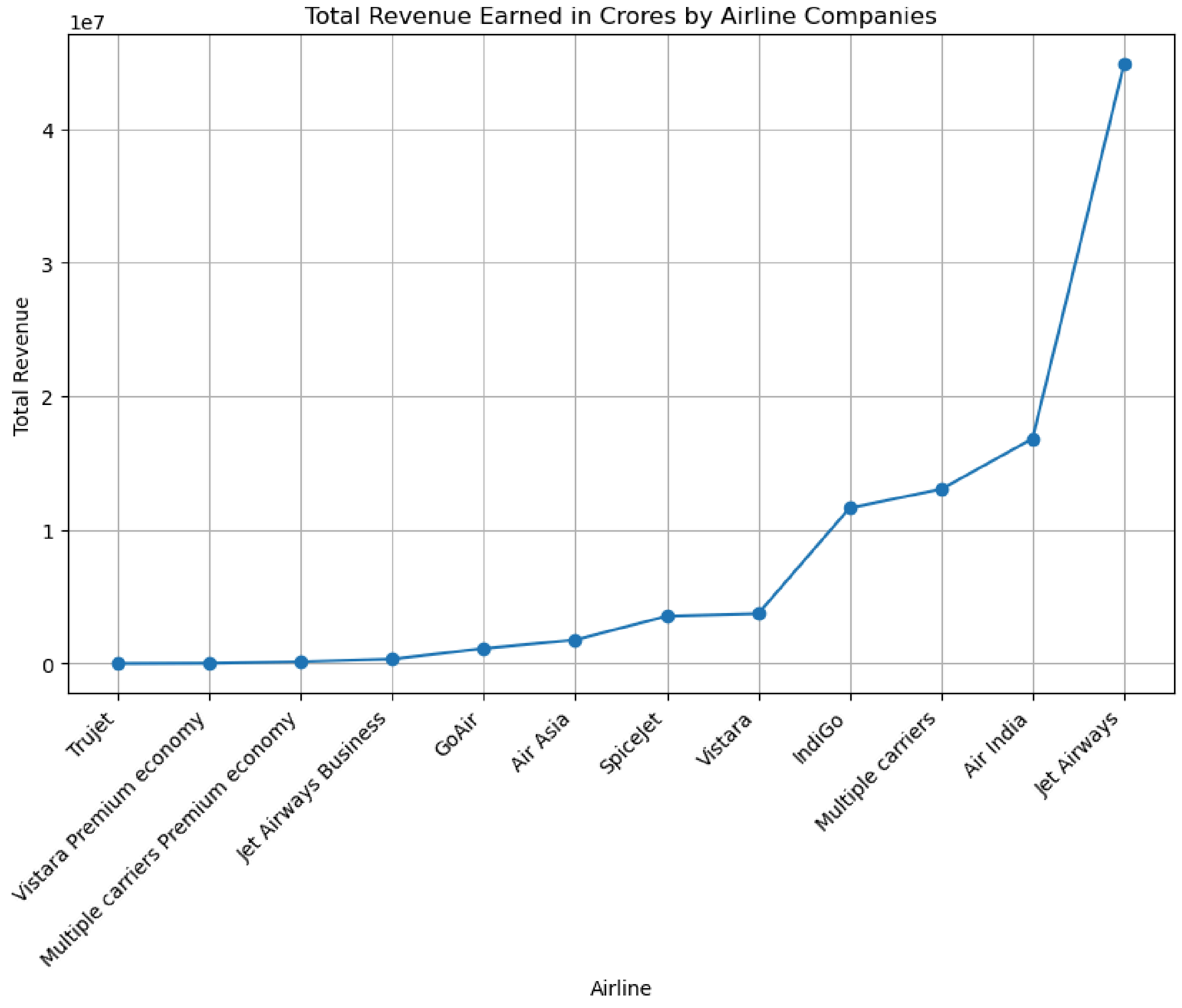
In [62]:
```python
# PLotting a box plot for Prices Distribution accross the Airline companies.
plt.figure(figsize=(15, 8))
plt.boxplot([sorted_data[sorted_data['Airline'] == airline]['Price'] for airline ir
plt.xticks(rotation=45, ha='right')  # Rotating x-axis labels for better readabilit
plt.title('Price Distribution Across Airlines')
plt.xlabel('Airline')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



In [63]:
```python
# Sorting the data by 'Price'
sorted_data = proj_data.sort_values('Price')

# Grouping data by 'Airline' and calculating mean price
mean_prices = sorted_data.groupby('Airline')['Price'].mean()


plt.pie(mean_prices, labels=mean_prices.index, autopct='%1.1f%%', startangle=140)
plt.title('Price Distributed among the Airline companies')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



```
In [64]: # Calculate total revenue earned by each airline
         revenue_by_airline = proj_data.groupby('Airline')['Price'].sum().sort_values()

         plt.figure(figsize=(10, 6))
         plt.plot(revenue_by_airline.index, revenue_by_airline.values, marker='o', linestyle
         plt.title('Total Revenue Earned in Crores by Airline Companies')
         plt.xlabel('Airline')
         plt.ylabel('Total Revenue')
         plt.xticks(rotation=45, ha='right')
         plt.grid(True)
         plt.show()
```



Model Prediction

In [65]:
```
# Performing One hot encoding using Label Encoder for the categorical column
# This step is crucial because models cannot transform string values to numbers or
# run analysis and apply them on the data; all values must be numeric.
cols = ['Airline', 'Source', 'Destination', 'Route', 'Total_Stops', 'Additional_Inf
for col in cols:
    proj_data[col] = LabelEncoder().fit_transform(proj_data[col])
```

In [66]:
```
proj_data.head()
```

Out[66]:

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Hours	Minutes	Day
0	3	0	5	18	0	8	3897.0	2	50	24
1	1	3	0	84	2	8	7662.0	7	25	1
2	4	2	1	118	2	8	13882.0	19	0	9
3	3	3	0	91	1	8	6218.0	5	25	12
4	3	0	5	29	1	8	13302.0	4	45	1

In [67]:
```
# Putting response and predictors in Y and X variable respectively.
X = proj_data.drop(['Price'],axis=1)
Y = proj_data['Price']
```

In [68]:
```
# Splitting the data into train and test using 'train_test_split' function
X_train , X_test , y_train , y_test = train_test_split(X , Y , test_size = 0.2, rar
```

In [69]:
```
# Printing all the train and test data
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

```
      Airline  Source  Destination  Route  Total_Stops  Additional_Info  \
8450        3       3            0     64            0                8
637         1       0            2     18            0                8
9938        1       2            1    112            2                8
8830        8       3            0     64            0                7
4509        3       1            4    127            0                8
...       ...     ...          ...    ...          ...              ...
9373        8       1            4    127            0                7
7291        6       2            1    104            1                8
1344        3       0            2     18            0                8
7293        4       0            5      5            1                5
1289        6       2            1    104            1                8

      Hours  Minutes  Day  Month  Year
8450      2       40    9      3  2019
637       2       50   27      4  2019
9938     26       20    6      3  2019
8830      2       35   27      4  2019
4509      2       25    1      6  2019
...     ...      ...  ...    ...   ...
9373      2       15   15      6  2019
7291     12       10    6      6  2019
1344      2       50   21      6  2019
7293     10       25   15      3  2019
1289      8       45   24      6  2019

[8545 rows x 11 columns]
      Airline  Source  Destination  Route  Total_Stops  Additional_Info  \
2389        4       3            0     66            1                5
5411        4       3            0     73            1                8
2674        4       3            0     73            1                5
970         1       2            1    104            1                8
5845        1       2            1    104            1                8
...       ...     ...          ...    ...          ...              ...
3727        3       2            1    111            1                8
7639        3       2            1    104            1                8
860         3       0            5     18            0                8
9115        8       3            0     82            1                8
8102        8       1            4    127            0                8

      Hours  Minutes  Day  Month  Year
2389     12       10   12      6  2019
5411     22       45   15      5  2019
2674     18        0    6      6  2019
970       9       15   21      3  2019
5845     11       15    9      3  2019
...     ...      ...  ...    ...   ...
3727      6       20    6      6  2019
7639      6       55    3      6  2019
860       2       45    3      3  2019
9115      5       30   15      5  2019
8102      2       15    9      5  2019

[2137 rows x 11 columns]
8450     4148.0
637      5228.0
9938     9190.0
8830     3841.0
4509     3540.0
          ...
9373     3543.0
7291    14848.0
1344     4823.0
7293     9134.0
```

```
1289    12192.0
Name: Price, Length: 8545, dtype: float64
2389     6224.0
5411    14151.0
2674    10539.0
970      7934.0
5845    16754.0
         ...  
3727     6938.0
7639     6628.0
860      6144.0
9115     7804.0
8102     3597.0
Name: Price, Length: 2137, dtype: float64
```

Multiple Linear Regression

In [70]:
```
# Training the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluating the model

mse_lr = mean_squared_error(y_test, y_pred)
rmse_lr = np.sqrt(mse_lr)
r2_lr = r2_score(y_test, y_pred)

accuracy_test_lr = model.score(X_test, y_test)
print("Model Accuracy:",accuracy_test_lr*100)
print("Mean Squared Error:", mse_lr)
print("Root Mean Squared Error:", rmse_lr)
print("R-squared:", r2_lr)
```

```
Model Accuracy: 41.0495651380572
Mean Squared Error: 12739045.0012673
Root Mean Squared Error: 3569.1798779645865
R-squared: 0.410495651380572
```

Hyperparameter Tuning for Linear Model

In [71]:
```
# Hyperparameter Tuning for Linear Model
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Define the range of alpha values to try
alpha_values = [0.001, 0.01, 0.1, 1, 10, 100]

# Creating a dictionary of hyperparameters to search
param_grid = {'alpha': alpha_values}

# Creating the Ridge regression model
ridge_model = Ridge()

# Performing a grid search with cross-validation
grid_search = GridSearchCV(ridge_model, param_grid, cv=5, scoring='neg_mean_squarec
grid_search.fit(X_train, y_train)

# Getting the best hyperparameters
```

```
best_alpha = grid_search.best_params_['alpha']

# Training the model with the best hyperparameters
best_model = Ridge(alpha=best_alpha)
best_model.fit(X_train, y_train)

# Making predictions on the testing set
y_pred = best_model.predict(X_test)

# Evaluating the model
mse_lr_tuned = mean_squared_error(y_test, y_pred)
rmse_lr_tuned = np.sqrt(mse_lr_tuned)
r2_lr_tuned = r2_score(y_test, y_pred)

accuracy_test_lr_tuned = best_model.score(X_test, y_test)
print("Best Model Accuracy:", accuracy_test_lr_tuned * 100)
print("Best Model Mean Squared Error:", mse_lr_tuned)
print("Best Model Root Mean Squared Error:", rmse_lr_tuned)
print("Best Model R-squared:", r2_lr_tuned)
print("Best Alpha Value:", best_alpha)
```

```
Best Model Accuracy: 41.049535062038146
Best Model Mean Squared Error: 12739051.500621565
Best Model Root Mean Squared Error: 3569.1807884473383
Best Model R-squared: 0.41049535062038145
Best Alpha Value: 1
```

It appears that hyperparameter tuning did not significantly improve the performance of the model. Both before and after hyperparameter tuning, the model's accuracy, mean squared error, root mean squared error, and R-squared value remained nearly unchanged.

The negligible difference in performance metrics suggests that the default hyperparameters or the selected hyperparameters during tuning did not significantly impact the model's performance. This outcome could be due to various factors, such as the nature of the dataset, the chosen algorithm, or the range of hyperparameters explored during tuning. Further experimentation with different hyperparameter ranges or alternative algorithms may be necessary to achieve significant improvements in model performance.

Random Forest Model

In [72]:

```
# Train the Random Forest model
model = RandomForestRegressor()
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

accuracy_test_rf = model.score(X_test, y_test)
print("Model Accuracy: ",accuracy_test_rf*100)

mse_rf = mean_squared_error(y_test, y_pred)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse_rf)
print("Root Mean Squared Error:", rmse_rf)
print("R-squared:", r2_rf)
```

```
Model Accuracy:  83.16373762143249
Mean Squared Error: 3638275.181447012
Root Mean Squared Error: 1907.4263239892155
R-squared: 0.8316373762143249
```

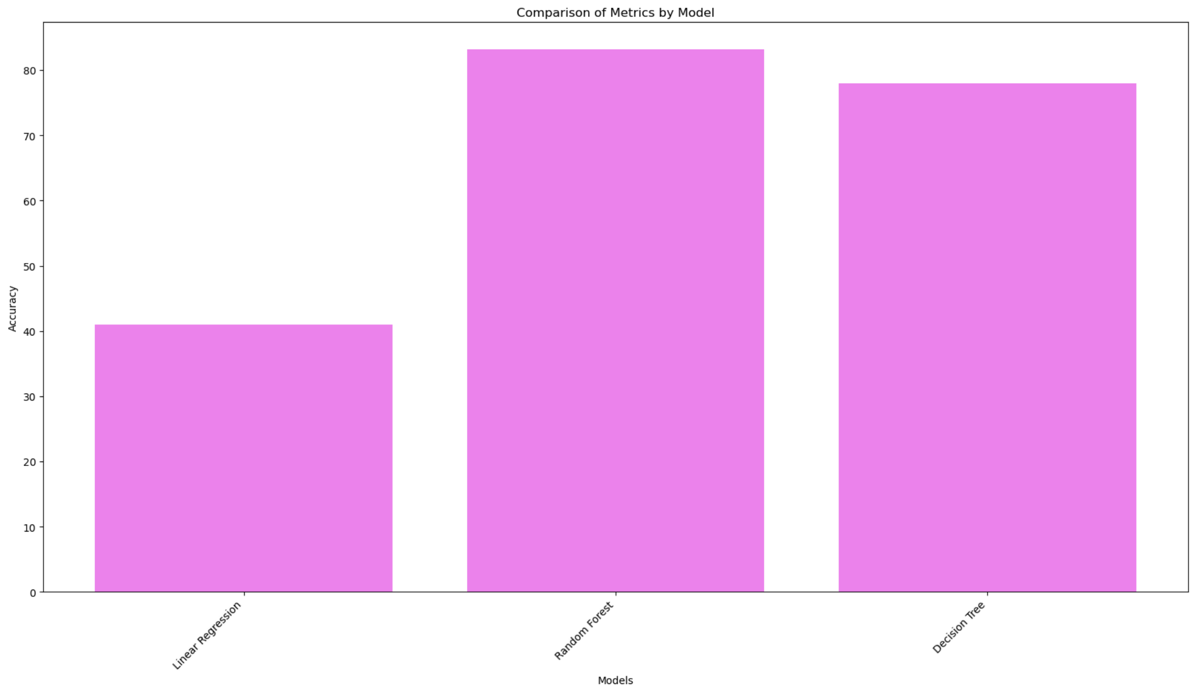
Decision Tree Regression

In [73]:
```
model = DecisionTreeRegressor(random_state=42)

model.fit(X_train, y_train)

# Predict the values for the training and testing sets
y_pred_test = model.predict(X_test)

# Computing the accuracy, MSE, and R2 for the testing set
accuracy_test_dt = model.score(X_test, y_test)

mse_dt = mean_squared_error(y_test, y_pred_test)
rmse_dt = np.sqrt(mse_dt)
r2_dt = r2_score(y_test, y_pred_test)

print("Model Accuracy: ",accuracy_test_dt*100)
print("MSE: ", mse_dt)
print("Root Mean Squared Error:", rmse_dt)
print("R2:",r2_dt)
```

```
Model Accuracy:  77.97837558866946
MSE:  4758819.252715191
Root Mean Squared Error: 2181.4718088288905
R2: 0.7797837558866947
```

Comparing the Accuracies of models

In [74]:
```
accuracies = [accuracy_test_lr*100,accuracy_test_rf*100,accuracy_test_dt*100]

models = ['Linear Regression', 'Random Forest', 'Decision Tree']

plt.figure(figsize=(20, 10))
plt.bar(models, accuracies,color='violet')
plt.title('Comparison of Metrics by Model')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(rotation=45, ha='right')  # Rotating x-axis labels for better readabilit
plt.show()
```



References

https://www.kaggle.com/code/jillanisofttech/flight-price-prediction/notebook