**ML Mini Project (Task 1)**

Name: Hitanshu Shah

Branch: CSE(DS)

BATCH: D1-2

SAP ID: 60009220164

**Flight Price Prediction Project**

**Problem Description**

The problem we aim to address is flight price prediction. Flight prices are known to fluctuate due to various factors such as seasonality, demand, availability, and market conditions. It is crucial for customers to have an accurate estimation of flight prices to make informed decisions when planning their travels. Our goal is to build a model that can predict flight prices based on historical data and other relevant attributes.

**Justification as a Data Science Problem**

Flight price prediction is inherently a data science problem due to the following reasons:

1. **Data-Driven Approach:** Solving this problem requires leveraging historical data and applying data science techniques to build a predictive model. By analyzing patterns and relationships within the data, we can identify factors that significantly impact flight prices.
2. **Complexity and Variables:** Predicting flight prices involves dealing with a large number of variables such as the date of the journey, source and destination airports, route, departure time, arrival time, flight duration, total stops, and additional information. The relationships between these variables and flight prices are often non-linear and complex, making it suitable for data science methodologies.
3. **Machine Learning Techniques:** Data science provides a range of machine learning algorithms and predictive modeling techniques that can be employed to accurately forecast flight prices. These algorithms can learn from historical data, capture underlying patterns, and make predictions on unseen data.

**Data Justification**

The chosen dataset is appropriate to build a model to solve the problem of flight price prediction. It contains the following essential attributes:

* **Airline:** The airline operating the flight can greatly impact the ticket prices.
* **Date\_of\_Journey:** The date of the flight is a fundamental factor affecting pricing.
* **Source and Destination:** The departure and arrival locations influence flight prices.
* **Route:** The specific route taken by the flight can impact pricing.
* **Dep\_Time and Arrival\_Time:** The departure and arrival times play a role in determining prices.
* **Duration:** The duration of the flight is a significant factor in pricing.
* **Total\_Stops:** The number of stops during the journey affects ticket prices.
* **Additional\_Info:** Additional information such as aircraft type or service class may have an impact on prices.
* **Price:** The target variable we aim to predict.

By utilizing this dataset, we can build a model that incorporates historical flight prices and attributes that contribute to pricing. The inclusion of variables such as airline, date of journey, source and destination, and other relevant information enables the model to capture the complexity of flight pricing dynamics.

In conclusion, the flight price prediction project is a relevant data science problem that requires analyzing historical data and utilizing machine learning techniques to build an accurate predictive model. The chosen dataset encompasses the necessary attributes to construct such a model, enabling us to make reliable predictions and support informed decision-making for flight bookings.

**Data Preprocessing**

importing all important libraries such as numpy pandas and

# 

## matplotlib

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

creating a datframe named train\_df and reading the excel sheet

# 

## using the pandas read\_excel function

train\_df = pd.read\_excel('/content/Flight\_train.xlsx') train\_df.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | **Date\_of\_Journey** | **Source** | **Destination** | **Route** | **Dep\_Time** | **Arrival\_Time** | **Dur** |
|  |  |  |  | BLR |  |  |  |
| **0** IndiGo | 24/03/2019 | Banglore | New Delhi | → DEL | 22:20 | 01:10 22 Mar | 2 |
|  |  |  |  | CCU |  |  |  |
|  |  |  |  | → IXR |  |  |  |
| **1** Air India | 1/05/2019 | Kolkata | Banglore | → | 05:50 | 13:15 | 7 |
|  |  |  |  |  |  |  |  |

extracting info about the dataset such as column names and

# 

## their data types

train\_df.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 10683 entries, 0 to 10682 Data columns (total 11 columns):

# Column Non-Null Count Dtype

1. Airline 10683 non-null object
2. Date\_of\_Journey 10683 non-null object
3. Source 10683 non-null object
4. Destination 10683 non-null object
5. Route 10682 non-null object
6. Dep\_Time 10683 non-null object
7. Arrival\_Time 10683 non-null object
8. Duration 10683 non-null object
9. Total\_Stops 10682 non-null object
10. Additional\_Info 10683 non-null object
11. Price 10683 non-null int64 dtypes: int64(1), object(10)

memory usage: 918.2+ KB

##  Checking for Null values in the dataset

train\_df.isnull().sum()

|  |  |
| --- | --- |
| 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 |  |

finding the location where null values are present in the route

# 

## column

train\_df[train\_df['Route'].isnull()]

**Airline Date\_of\_Journey Source Destination Route Dep\_Time Arrival\_Time D**

**9039** Air India

6/05/2019

Delhi

Cochin NaN

09:45 09:25 07 May

##  deleting the tuple which has null value of route

train\_df.drop(9039,axis=0,inplace=True)

##  finding location in total stops column which had a null value

train\_df[train\_df['Total\_Stops'].isnull()]

**Airline Date of Journey Source Destination Route Dep Time Arrival Time Durat**

##  null values of Route and Total Stops were in the same tuple

train\_df.isnull().sum()

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

train\_df.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | **Date\_of\_Journey** | **Source** | **Destination** | **Route** | **Dep\_Time** | **Arrival\_Time** | **Dur** |
|  |  |  |  | BLR |  |  |  |
| **0** IndiGo | 24/03/2019 | Banglore | New Delhi | → DEL | 22:20 | 01:10 22 Mar | 2 |
|  |  |  |  | CCU |  |  |  |
|  |  |  |  | → IXR |  |  |  |
| **1** Air India | 1/05/2019 | Kolkata | Banglore | → | 05:50 | 13:15 | 7 |
|  |  |  |  |  |  |  |  |

splittig the date of journey column into 3 parts date month and

# 

## year

train\_df['Date']=train\_df['Date\_of\_Journey'].str.split('/').str[0] train\_df['Month']=train\_df['Date\_of\_Journey'].str.split('/').str[1] train\_df['Year']=train\_df['Date\_of\_Journey'].str.split('/').str[2] train\_df.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | **Date\_of\_Journey** | **Source** | **Destination** | **Route** | **Dep\_Time** | **Arrival\_Time** | **Dur** |
|  |  |  |  | BLR |  |  |  |
| **0** IndiGo | 24/03/2019 | Banglore | New Delhi | → DEL | 22:20 | 01:10 22 Mar | 2 |
|  |  |  |  | CCU |  |  |  |
|  |  |  |  | → IXR |  |  |  |
| **1** Air India | 1/05/2019 | Kolkata | Banglore | → | 05:50 | 13:15 | 7 |
|  |  |  |  |  |  |  |  |

##  dropping the date of journey column

train\_df.drop(columns=['Date\_of\_Journey'],inplace=True)

##  converting the data type of date month and year into integer

train\_df['Date'] = train\_df['Date'].astype(int) train\_df['Month'] = train\_df['Month'].astype(int) train\_df['Year'] = train\_df['Year'].astype(int) train\_df.info()

<class 'pandas.core.frame.DataFrame'> Index: 10682 entries, 0 to 10682

Data columns (total 13 columns):

# Column Non-Null Count Dtype

1. Airline 10682 non-null object
2. Source 10682 non-null object
3. Destination 10682 non-null object
4. Route 10682 non-null object
5. Dep\_Time 10682 non-null object
6. Arrival\_Time 10682 non-null object
7. Duration 10682 non-null object
8. Total\_Stops 10682 non-null object
9. Additional\_Info 10682 non-null object
10. Price 10682 non-null int64
11. Date 10682 non-null int64
12. Month 10682 non-null int64
13. Year 10682 non-null int64 dtypes: int64(4), object(9)

memory usage: 1.4+ MB

splitting departure time into departure hour and departure min

# 

## and converting their datatype into integer

train\_df['Dept\_Hour']=train\_df['Dep\_Time'].str.split(':').str[0] train\_df['Dept\_Min']=train\_df['Dep\_Time'].str.split(':').str[1]

train\_df['Dept\_Hour']=train\_df['Dept\_Hour'].astype(int) train\_df['Dept\_Min']=train\_df['Dept\_Min'].astype(int)

train\_df.drop(columns=['Dep\_Time'],inplace=True) train\_df.info()

<class 'pandas.core.frame.DataFrame'> Index: 10682 entries, 0 to 10682

Data columns (total 14 columns):

# Column Non-Null Count Dtype

1. Airline 10682 non-null object
2. Source 10682 non-null object
3. Destination 10682 non-null object
4. Route 10682 non-null object
5. Arrival\_Time 10682 non-null object
6. Duration 10682 non-null object
7. Total\_Stops 10682 non-null object
8. Additional\_Info 10682 non-null object
9. Price 10682 non-null int64
10. Date 10682 non-null int64
11. Month 10682 non-null int64
12. Year 10682 non-null int64
13. Dept\_Hour 10682 non-null int64
14. Dept\_Min 10682 non-null int64 dtypes: int64(6), object(8)

memory usage: 1.5+ MB

train\_df.head()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Airline** | **Source** | **Destination** | **Route** | **Arrival\_Time** | **Duration Total\_Stops Additio** |
|  |  |  | BLR |  |  |
| **0** IndiGo | Banglore | New Delhi | → DEL | 01:10 22 Mar | 2h 50m non-stop |
|  |  |  | CCU |  |  |
|  |  |  | → IXR |  |  |
| **1** Air India | Kolkata | Banglore | → | 13:15 | 7h 25m 2 stops |
|  |  |  |  |  |  |

splitting arrival time into arrival hour and arrival minute and

# 

## converting its data type into integer

train\_df['Arrival\_Time']=train\_df['Arrival\_Time'].str.split(' ').str[ train\_df['Arrival\_Hour']=train\_df['Arrival\_Time'].str.split(':').str[ train\_df['Arrival\_Min']=train\_df['Arrival\_Time'].str.split(':').str[1 train\_df['Arrival\_Hour']=train\_df['Arrival\_Hour'].astype(int)

train\_df['Arrival\_Min']=train\_df['Arrival\_Min'].astype(int) train\_df.drop(columns=['Arrival\_Time'],inplace=True)

train\_df.info()

<class 'pandas.core.frame.DataFrame'> Index: 10682 entries, 0 to 10682

Data columns (total 15 columns):

# Column Non-Null Count Dtype

1. Airline 10682 non-null object
2. Source 10682 non-null object
3. Destination 10682 non-null object
4. Route 10682 non-null object
5. Duration 10682 non-null object
6. Total\_Stops 10682 non-null object
7. Additional\_Info 10682 non-null object
8. Price 10682 non-null int64
9. Date 10682 non-null int64
10. Month 10682 non-null int64
11. Year 10682 non-null int64
12. Dept\_Hour 10682 non-null int64
13. Dept\_Min 10682 non-null int64
14. Arrival\_Hour 10682 non-null int64
15. Arrival\_Min 10682 non-null int64 dtypes: int64(8), object(7)

memory usage: 1.6+ MB

train\_df.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | **Source** | **Destination** | **Route** | **Duration** | **Total\_Stops** | **Additional\_Info** | **Pric** |
|  |  |  | BLR |  |  |  |  |
| **0** IndiGo | Banglore | New Delhi | → DEL | 2h 50m | non-stop | No info | 389 |
|  |  |  | CCU |  |  |  |  |
|  |  |  | → IXR |  |  |  |  |
| **1** Air India | Kolkata | Banglore | → BBI  → | 7h 25m | 2 stops | No info | 766 |
|  |  |  |  |  |  |  |  |

##  splitting the duration into hour and minutes

train\_df['Duration\_Hour'] = train\_df['Duration'].str.split(' ').str[ train\_df['Duration\_Min'] = train\_df['Duration'].str.split(' ').str[1

deleting the column which had duration of 5mins, as it is an

# 

## outlier

train\_df[train\_df['Duration']=='5m']

**Airline Source Destination Route Duration Total\_Stops Additional\_Info Pr**

**6474** Air India Mumbai Hyderabad

BOM

→ GOI

→ PNQ

→

HYD

5m 2 stops No info 17

train\_df.drop(6474,axis=0,inplace=True)

train\_df['Duration\_Hour'] = train\_df['Duration\_Hour'].astype(int)

there are cases where the minutes column has null values so we

# 

## fill them with integer 0

train\_df['Duration\_Min'].unique()

array(['50', '25', nan, '45', '30', '5', '15', '35', '10', '20', '55',

'40'], dtype=object)

train\_df['Duration\_Min'].fillna(0,inplace=True)

train\_df['Duration\_Min'] = train\_df['Duration\_Min'].astype(int)

we drop the column duration as we already have the values as hour and min

train\_df.drop(columns=['Duration'],inplace=True) train\_df.info()

<class 'pandas.core.frame.DataFrame'> Index: 10681 entries, 0 to 10682

Data columns (total 16 columns):

# Column Non-Null Count Dtype

1. Airline 10681 non-null object
2. Source 10681 non-null object
3. Destination 10681 non-null object
4. Route 10681 non-null object
5. Total\_Stops 10681 non-null object
6. Additional\_Info 10681 non-null object
7. Price 10681 non-null int64
8. Date 10681 non-null int64

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 8 | Month | 10681 | non-null | int64 |
| 9 | Year | 10681 | non-null | int64 |
| 10 | Dept\_Hour | 10681 | non-null | int64 |
| 11 | Dept\_Min | 10681 | non-null | int64 |
| 12 | Arrival\_Hour | 10681 | non-null | int64 |
| 13 | Arrival\_Min | 10681 | non-null | int64 |
| 14 | Duration\_Hour | 10681 | non-null | int64 |
| 15 | Duration\_Min | 10681 | non-null | int64 |

dtypes: int64(10), object(6) memory usage: 1.4+ MB

train\_df.head()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | | **Source** | **Destination** | **Route** | **Total\_Stops** | **Additional\_Info** | **Price** | **Date M** |
|  | |  |  | BLR |  |  |  |  |
| **0** IndiGo | | Banglore | New Delhi | → | non-stop | No info | 3897 | 24 |
|  | |  |  | DEL |  |  |  |  |
|  | |  |  | CCU |  |  |  |  |
|  | |  |  | → |  |  |  |  |
|  | |  |  | IXR |  |  |  |  |
| **1** Air India | | Kolkata | Banglore | → | 2 stops | No info | 7662 | 1 |
|  | |  |  | BBI |  |  |  |  |
|  | |  |  | → |  |  |  |  |
|  | |  |  | BLR |  |  |  |  |
|  | |  |  | DEL |  |  |  |  |
|  | |  |  | → |  |  |  |  |
| **2** | Jet Airways | Delhi | Cochin | LKO  → BOM | 2 stops | No info | 13882 | 9 |
|  |  |  |  | → |  |  |  |  |
|  |  |  |  | COK |  |  |  |  |
|  |  |  |  | CCU |  |  |  |  |
|  |  |  |  | → |  |  |  |  |
| **3** | IndiGo | Kolkata | Banglore | NAG | 1 stop | No info | 6218 | 12 |
|  |  |  |  | → |  |  |  |  |
|  |  |  |  | BLR |  |  |  |  |
|  |  |  |  | BLR |  |  |  |  |
|  |  |  |  | → |  |  |  |  |
| **4** | IndiGo | Banglore | New Delhi | NAG | 1 stop | No info | 13302 | 1 |
|  |  |  |  | → |  |  |  |  |
|  |  |  |  | DEL |  |  |  |  |

train\_df['Airline'].value\_counts()

|  |  |
| --- | --- |
| Airline  Jet Airways | 3849 |
| IndiGo | 2053 |
| Air India | 1750 |
| Multiple carriers | 1196 |
| SpiceJet | 818 |
| Vistara | 479 |

Air Asia 319

GoAir 194

Multiple carriers Premium economy 13

|  |  |
| --- | --- |
| Jet Airways Business | 6 |
| Vistara Premium economy | 3 |
| Trujet  Name: count, dtype: int64 | 1 |

import seaborn as sns

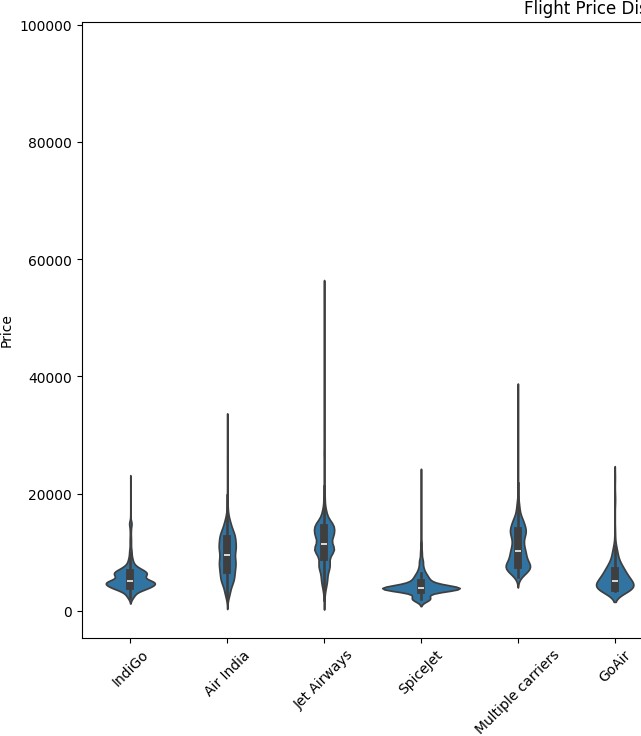
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 8))

sns.violinplot(x='Airline', y='Price', data=train\_df) plt.xticks(rotation=45)

plt.title('Flight Price Distribution by Airline') plt.xlabel('Airline')

plt.ylabel('Price') plt.show()



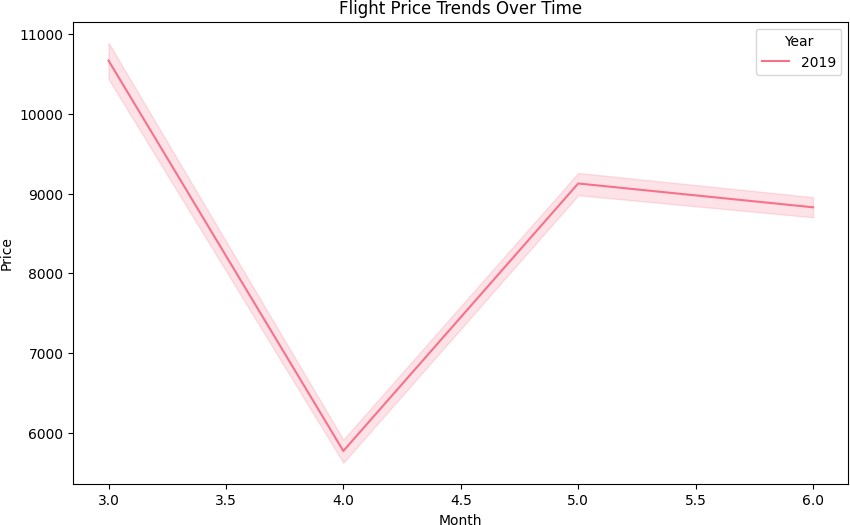
##  Airlines and their costs are plot using violin plot

plt.figure(figsize=(10, 6))

sns.lineplot(x='Month', y='Price', hue='Year', data=train\_df, palette plt.title('Flight Price Trends Over Time')

plt.xlabel('Month') plt.ylabel('Price')

plt.legend(title='Year') plt.show()



average cost of the flights according to the months are plot

# 

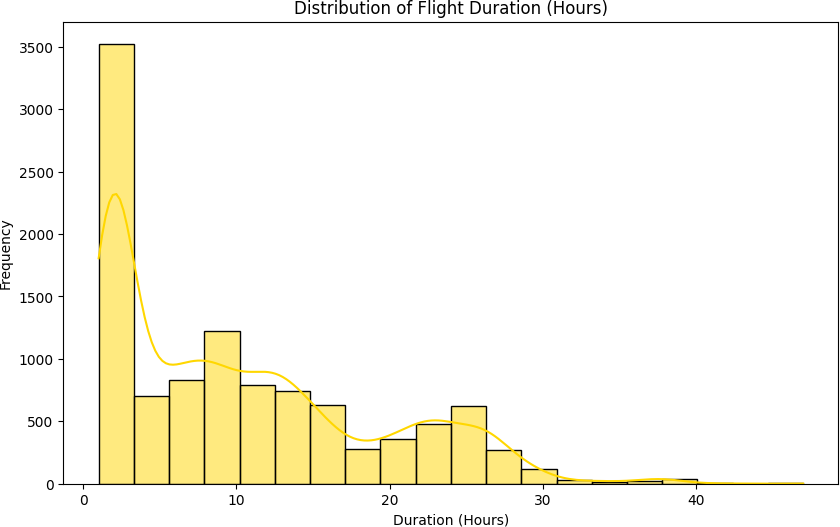
## using lineplot

plt.figure(figsize=(10, 6))

sns.histplot(train\_df['Duration\_Hour'], bins=20, kde=True, color='gol plt.title('Distribution of Flight Duration (Hours)')

plt.xlabel('Duration (Hours)') plt.ylabel('Frequency')

plt.show()



it shows that the flights of longer duration are very few whereas

# 

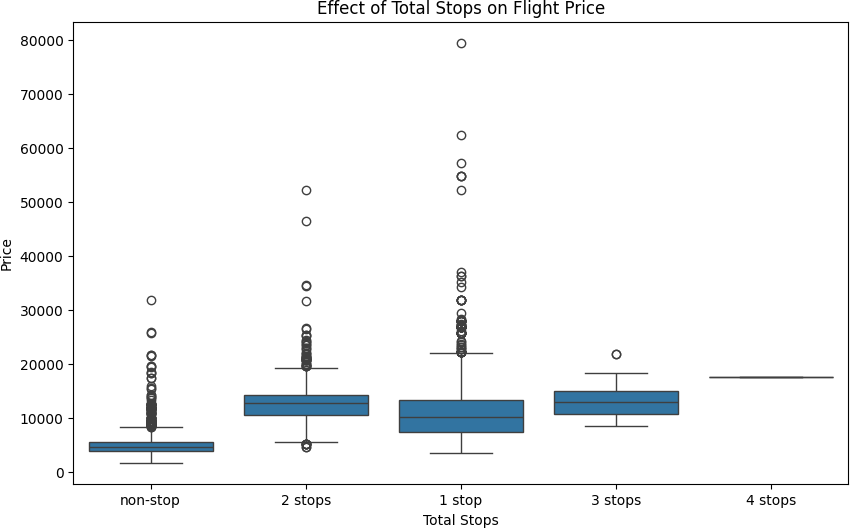
## flights of shorter duration have higher frequency

plt.figure(figsize=(10, 6))

sns.boxplot(x='Total\_Stops', y='Price', data=train\_df) plt.title('Effect of Total Stops on Flight Price')

plt.xlabel('Total Stops') plt.ylabel('Price')

plt.show()



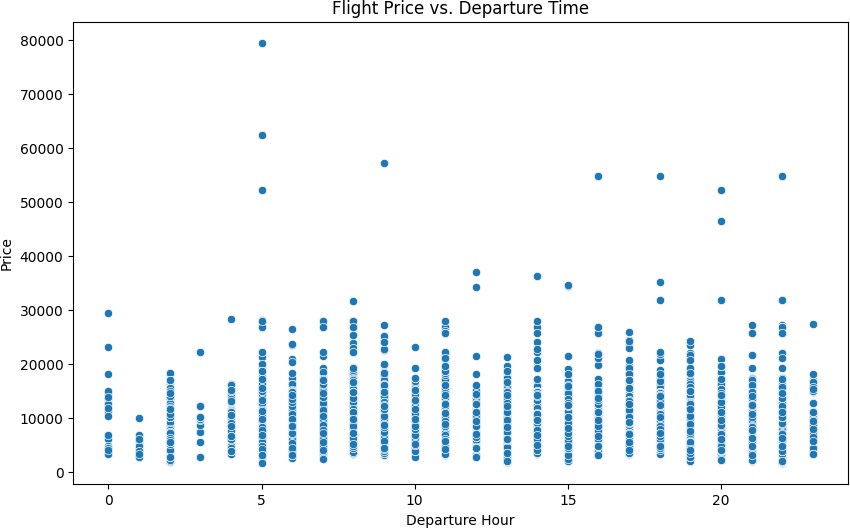
##  It clearly shows the effect of total stops on cost of the flight

plt.figure(figsize=(10, 6))

sns.scatterplot(x='Dept\_Hour', y='Price', data=train\_df) plt.title('Flight Price vs. Departure Time')

plt.xlabel('Departure Hour') plt.ylabel('Price')

plt.show()



Multiple carriers Premium economy 13 Jet Airways Business 6

Vistara Premium economy 3 Trujet 1

clubbing these airlines together into category of 'other' since

# 

## they have very low frequency

Airline = train\_df[['Airline']]

Current\_Airline\_List = Airline['Airline'] New\_Airline\_List = []

for carrier in Current\_Airline\_List:

if carrier in ['Jet Airways','IndiGo','Air India','Multiple carrier New\_Airline\_List.append(carrier)

else:

New\_Airline\_List.append('other')

Airline['Airline'] = pd.DataFrame(New\_Airline\_List) Airline['Airline'].value\_counts()

<ipython-input-35-fd259cce665f>:11: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/u](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)s Airline['Airline'] = pd.DataFrame(New\_Airline\_List)

Airline

Jet Airways 3849

IndiGo 2053

Air India 1749

Multiple carriers 1195

SpiceJet 818

Vistara 479

Air Asia 319

GoAir 194

other 23

Name: count, dtype: int64

Generating Dummy values for Airlines as we need numeric

# 

## values to train our model

Airline = pd.get\_dummies(Airline, drop\_first=True) Airline = Airline.astype(int)

Airline.head()

**Airline\_Air**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **India** |  |  | **Airways** | **carriers** |
| **0** | 0 | 0 | 1 | 0 | 0 |
| **1** | 1 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 1 | 0 | 0 |

**Airline\_GoAir Airline\_IndiGo Airline\_Jet**

**Airline\_Multiple**

**Airline**

train\_df['Source'].value\_counts()

|  |  |
| --- | --- |
| Source  Delhi | 4536 |
| Kolkata | 2871 |
| Banglore | 2197 |
| Mumbai | 696 |
| Chennai | 381 |
| Name: count, | dtype: int64 |

##  Generating dummy values for Source

Source = train\_df[['Source']]

Source = pd.get\_dummies(Source, drop\_first=True) Source = Source.astype(int)

Source.head()

|  |  |  |  |
| --- | --- | --- | --- |
| **Source\_Chennai** | **Source\_Delhi** | **Source\_Kolkata** | **Source\_Mumbai** |
| **0** 0 | 0 | 0 | 0 |
| **1** 0 | 0 | 1 | 0 |
| **2** 0 | 1 | 0 | 0 |
| **3** 0 | 0 | 1 | 0 |
| **4** 0 | 0 | 0 | 0 |

train\_df['Destination'].value\_counts()

|  |  |
| --- | --- |
| Destination  Cochin | 4536 |
| Banglore | 2871 |
| Delhi | 1265 |
| New Delhi | 932 |
| Hyderabad | 696 |
| Kolkata | 381 |
| Name: count, | dtype: int64 |

##  Clubbing the columns having destination as New Delhi and Delhi

Destination = train\_df[['Destination']]

Current\_Destination\_List = Destination['Destination'] New\_Destination\_List = []

for value in Current\_Destination\_List: if value in ['New Delhi']:

New\_Destination\_List.append('Delhi') else:

New\_Destination\_List.append(value)

Destination['Destination'] = pd.DataFrame(New\_Destination\_List)

<ipython-input-40-45107eb6b7bc>:11: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame. Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/u](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)s Destination['Destination'] = pd.DataFrame(New\_Destination\_List)

##  Generating dummy values for destination

Destination = pd.get\_dummies(Destination,drop\_first=True) Destination = Destination.astype(int)

Destination.head()

**Destination\_Cochin Destination\_Delhi Destination\_Hyderabad Destination\_Kolkata**

**0** 0 1 0 0

**1** 0 0 0 0

**2** 1 0 0 0

**3** 0 0 0 0

**4**

0

1

0

0

Dropping Route and Additional Info column as they are not

# 

## required anymore

train\_df.drop(columns=['Route','Additional\_Info'],inplace=True)

##  mapping total stops column into numeric values by removing the strings from them

train\_df['Total\_Stops']=train\_df['Total\_Stops'].map({'non-stop':0,'1 train\_df.head()

Airways

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline Source** | **Destination** | **Total\_Stops** | **Price** | **Date** | **Month** | **Year** | **Dept\_Hour** | **D** |
| **0** IndiGo Banglore | New Delhi | 0 | 3897 | 24 | 3 | 2019 | 22 |  |
| **1** Air India Kolkata | Banglore | 2 | 7662 | 1 | 5 | 2019 | 5 |  |
| **2** Jet Delhi | Cochin | 2 | 13882 | 9 | 6 | 2019 | 9 |  |
| **3** IndiGo Kolkata | Banglore | 1 | 6218 | 12 | 5 | 2019 | 18 |  |

We concat all the columns with the dataframe and the dummy

# 

## values and store it in new data frame named final\_df

final\_df = pd.concat([train\_df,Airline,Source,Destination],axis=1)

final\_df.drop(columns=['Airline','Source','Destination'],inplace=True

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| final\_df.head() |  | | | | | | |
| **Total\_Stops** | **Price** | **Date** | **Month** | **Year** | **Dept\_Hour** | **Dept\_Min** | **Arrival\_Hour Arrival\_** |
| **0** 0 | 3897 | 24 | 3 | 2019 | 22 | 20 | 1 |
| **1** 2 | 7662 | 1 | 5 | 2019 | 5 | 50 | 13 |
| **2** 2 | 13882 | 9 | 6 | 2019 | 9 | 25 | 4 |
| **3** 1 | 6218 | 12 | 5 | 2019 | 18 | 5 | 23 |
| **4** 1 | 13302 | 1 | 3 | 2019 | 16 | 50 | 21 |

5 rows × 27 columns

print(final\_df.shape)

(10681, 27)

final\_df.info()

**MODEL TRAINING**

 <class 'pandas.core.frame.DataFrame'> Index: 10681 entries, 0 to 10682

Data columns (total 27 columns):

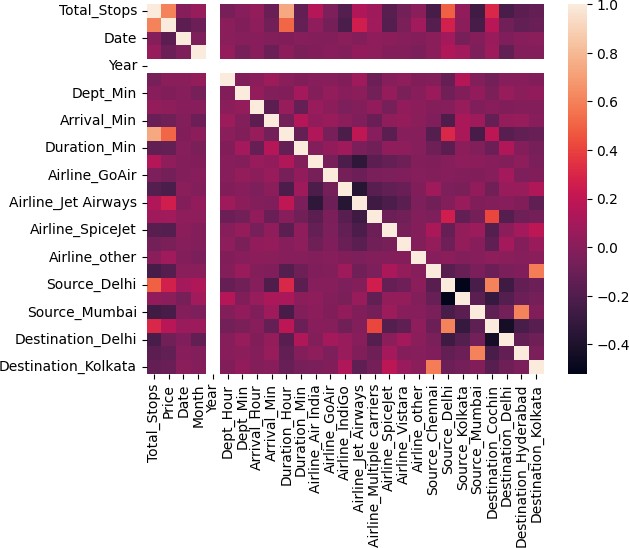
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # |  | Column | Non-Null Count |  | Dtype |
| 0 |  | Total\_Stops | 10681 non-null |  | int64 |
| 1 |  | Price | 10681 non-null |  | int64 |
| 2 |  | Date | 10681 non-null |  | int64 |
| 3 |  | Month | 10681 non-null |  | int64 |
| 4 |  | Year | 10681 non-null |  | int64 |
| 5 |  | Dept\_Hour | 10681 non-null |  | int64 |
| 6 |  | Dept\_Min | 10681 non-null |  | int64 |
| 7 |  | Arrival\_Hour | 10681 non-null |  | int64 |
| 8 |  | Arrival\_Min | 10681 non-null |  | int64 |
| 9 |  | Duration\_Hour | 10681 non-null |  | int64 |
| 10 |  | Duration\_Min | 10681 non-null |  | int64 |
| 11 |  | Airline\_Air India | 10681 non-null |  | int64 |
| 12 |  | Airline\_GoAir | 10681 non-null |  | int64 |
| 13 |  | Airline\_IndiGo | 10681 non-null |  | int64 |
| 14 |  | Airline\_Jet Airways | 10681 non-null |  | int64 |
| 15 |  | Airline\_Multiple carriers | 10681 non-null |  | int64 |
| 16 |  | Airline\_SpiceJet | 10681 non-null |  | int64 |
| 17 |  | Airline\_Vistara | 10681 non-null |  | int64 |
| 18 |  | Airline\_other | 10681 non-null |  | int64 |
| 19 |  | Source\_Chennai | 10681 non-null |  | int64 |
| 20 |  | Source\_Delhi | 10681 non-null |  | int64 |
| 21 |  | Source\_Kolkata | 10681 non-null |  | int64 |
| 22 |  | Source\_Mumbai | 10681 non-null |  | int64 |
| 23 |  | Destination\_Cochin | 10681 non-null |  | int64 |
| 24 |  | Destination\_Delhi | 10681 non-null |  | int64 |
| 25 |  | Destination\_Hyderabad | 10681 non-null |  | int64 |
| 26 |  | Destination\_Kolkata | 10681 non-null |  | int64 |

dtypes: int64(27)

memory usage: 2.3 MB

##  plotting heatmap to show co-relation between the columns in the dataframe

sns.heatmap(final\_df.corr()) plt.show()



##  Creating a 2D array named X for all independent variables

X=final\_df[['Total\_Stops','Date','Month','Year','Dept\_Hour','Dept\_Min','Arrival\_Hour','Arrival\_Min','Dura

##  Dependent variable y

y=final\_df['Price']

print(X.shape,y.shape)

(10681, 26) (10681,)

##  We use extra trees regressor ensemble method to find importance of each feature

from sklearn.ensemble import ExtraTreesRegressor selection = ExtraTreesRegressor()

selection.fit(X,y)

print(selection.feature\_importances\_)

[0.25550262 0.13772616 0.06692761 0. 0.03719684 0.03639477

0.04732834 0.03184791 0.18817184 0.02631088 0.00753913 0.00187986

0.00677012 0.03598518 0.00667466 0.00376218 0.00301156 0.01812422

0.00165935 0.02876969 0.01793846 0.00574803 0.01168851 0.01541995

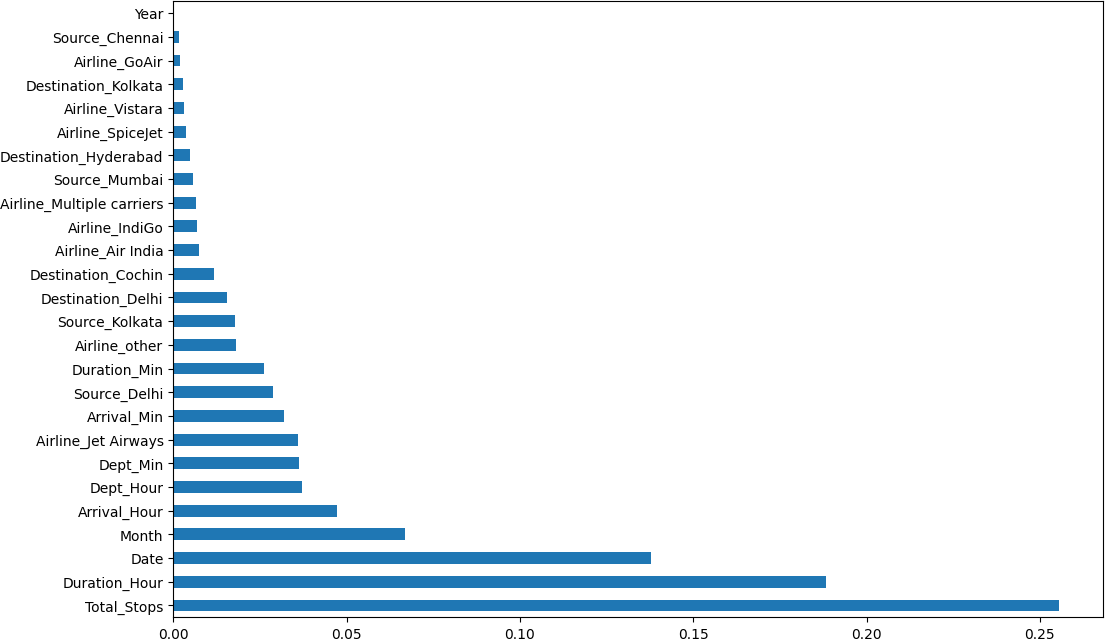
0.00490171 0.00272044]

##  Plotting all features according to their importance values

plt.figure(figsize=(12,8))

feat\_importances = pd.Series(selection.feature\_importances\_, index=X.columns) feat\_importances.nlargest(26).plot(kind='barh')

plt.show()



##  Calculation Variance Inflation Factor(VIF) for each feature

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor def calc\_vif(z):

vif = pd.DataFrame()

vif['variables'] = z.columns

vif['VIF'] = [variance\_inflation\_factor(z.values, i) for i in range(z.shape[1])] return(vif)

|  |  |  |
| --- | --- | --- |
| calc\_vif(X) |  | |
|  | **variables** | **VIF** |
| **0** | Total\_Stops | 2.944884 |
| **1** | Date | 1.021059 |
| **2** | Month | 1.101143 |
| **3** | Year | 86.749649 |
| **4** | Dept\_Hour | 1.049742 |
| **5** | Dept\_Min | 1.057674 |
| **6** | Arrival\_Hour | 1.057896 |
| **7** | Arrival\_Min | 1.115797 |
| **8** | Duration\_Hour | 2.375687 |
| **9** | Duration\_Min | 1.127660 |
| **10** | Airline\_Air India | 5.607061 |
| **11** | Airline\_GoAir | 1.593437 |
| **12** | Airline\_IndiGo | 6.147530 |
| **13** Airline\_Jet Airways | | 8.577139 |
| **14** Airline\_Multiple carriers | | 4.609269 |
| **15** Airline\_SpiceJet | | 3.388133 |
| **16** Airline\_Vistara | | 2.440108 |
| **17** Airline\_other | | 1.078473 |
| **18** Source\_Chennai | | 1.800734 |
| **19** Source\_Delhi | | 3.824347 |
| **20** Source\_Kolkata | | 3.014481 |
| **21** Source\_Mumbai | | 1.980430 |
| **22** Destination\_Cochin | | 2.562402 |
| **23** Destination\_Delhi | | 2.199341 |
| **24** Destination\_Hyderabad | | 1.848748 |
| **25** Destination\_Kolkata | | 1.759373 |

Year column has very high VIF and its feature importance is also 0 hence we remove it from the

# 

## list of independent variables

X=final\_df[['Total\_Stops','Date','Month','Dept\_Hour','Dept\_Min','Arrival\_Hour','Arrival\_Min','Duration\_Ho

X.head()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Total\_Stops** | **Date** | **Month** | **Dept\_Hour** | **Dept\_Min** | **Arrival\_Hour** | **Arrival\_Min Duration** |
| **0** 0 | 24 | 3 | 22 | 20 | 1 | 10 |
| **1** 2 | 1 | 5 | 5 | 50 | 13 | 15 |
| **2** 2 | 9 | 6 | 9 | 25 | 4 | 25 |
| **3** 1 | 12 | 5 | 18 | 5 | 23 | 30 |
| **4** 1 | 1 | 3 | 16 | 50 | 21 | 35 |

5 rows × 25 columns

VIF value > 15 is considered too high, month variable has high VIF but has great value in feature

# 

## importance so we let it be as it is

|  |  |  |
| --- | --- | --- |
| calc\_vif(X) |  | |
|  | **variables** | **VIF** |
| **0** | Total\_Stops | 7.322454 |
| **1** | Date | 3.471022 |
| **2** | Month | 15.384614 |
| **3** | Dept\_Hour | 5.697768 |
| **4** | Dept\_Min | 2.735111 |
| **5** | Arrival\_Hour | 4.720609 |
| **6** | Arrival\_Min | 3.377540 |
| **7** | Duration\_Hour | 5.813870 |
| **8** | Duration\_Min | 4.003182 |
| **9** | Airline\_Air India | 4.435427 |
| **10** | Airline\_GoAir | 1.400338 |
| **11** | Airline\_IndiGo | 4.845272 |
| **12** Airline\_Jet Airways | | 8.337666 |
| **13** Airline\_Multiple carriers | | 3.500706 |
| **14** Airline\_SpiceJet | | 2.536598 |
| **15** Airline\_Vistara | | 1.936893 |
| **16** Airline\_other | | 1.040538 |
| **17** Source\_Chennai | | 1.839899 |
| **18** Source\_Delhi | | 6.464506 |
| **19** Source\_Kolkata | | 3.951421 |
| **20** Source\_Mumbai | | 2.062669 |
| **21** Destination\_Cochin | | 4.367171 |
| **22** Destination\_Delhi | | 2.542039 |
| **23** Destination\_Hyderabad | | 1.971336 |
| **24** Destination\_Kolkata | | 1.817265 |

 Train Test Split into 80% training data and 20% testing data

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=42)

##  Training LinearRegression Model

from sklearn.linear\_model import LinearRegression linear=LinearRegression()

linear.fit(X\_train,y\_train)



▾ LinearRegression

LinearRegression()

linear\_pred = linear.predict(X\_test) from sklearn.metrics import r2\_score

linear\_score=r2\_score(linear\_pred,y\_test) linear\_score

0.003600909976055666

##  Training Decision Tree Regressor

from sklearn.tree import DecisionTreeRegressor tree = DecisionTreeRegressor()

tree.fit(X\_train,y\_train)



▾ DecisionTreeRegressor

DecisionTreeRegressor()

tree\_pred = tree.predict(X\_test)

tree\_score = r2\_score(tree\_pred,y\_test) tree\_score

0.6475461116441672

##  Training Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor random\_forest = RandomForestRegressor()

random\_forest.fit(X\_train,y\_train)



▾ RandomForestRegressor

RandomForestRegressor()

random\_forest\_pred = random\_forest.predict(X\_test)

random\_forest\_score = r2\_score(random\_forest\_pred,y\_test) random\_forest\_score

0.7368181898325543

Random Forest Regressor has the highest value of r2 score thus we select this model to train our

# 

## model

print(round(random\_forest.score(X\_train,y\_train)\*100,2))

96.25

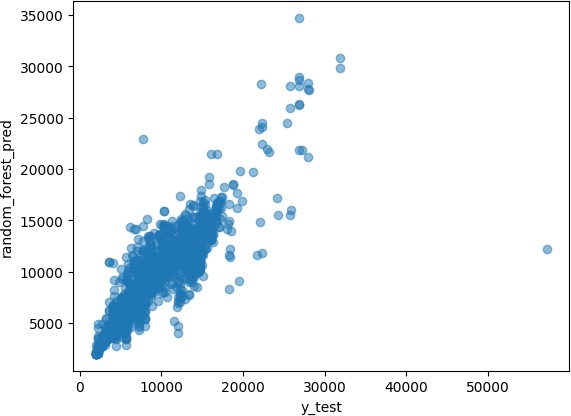
print(round(random\_forest.score(X\_test,y\_test)\*100,2))

78.34

##  It has 96.25 of training score and 78.34 testing score

plt.scatter(y\_test,random\_forest\_pred, alpha=0.5) plt.xlabel('y\_test')

plt.ylabel('random\_forest\_pred') plt.show()



##  We plot the values predicted by random forest regressor and y\_test

r2\_scores = {

"Linear Regression": 0.003600909976055666,

"Decison Tree Regressor": 0.6320709717106078, "Random Forest Regressor": 0.7356878314698677

}

plt.pie(r2\_scores.values(),labels=r2\_scores.keys(),autopct='%0.2f%%')

([<matplotlib.patches.Wedge at 0x7be94409bc40>,

<matplotlib.patches.Wedge at 0x7be94409b0d0>,

<matplotlib.patches.Wedge at 0x7be9409a7760>],

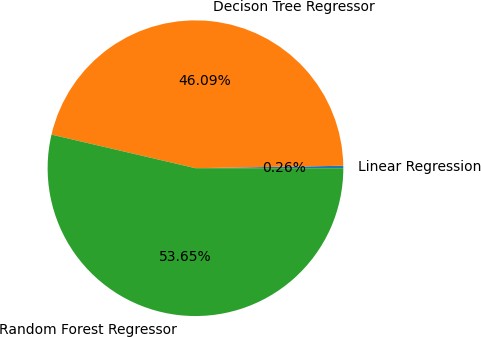
[Text(1.0999625732783682, 0.009073995086544892, 'Linear Regression'),

Text(0.11672318882996248, 1.0937896037124164, 'Decison Tree Regressor'),

Text(-0.12574197440347715, -1.092789529540394, 'Random Forest Regressor')], [Text(0.5999795854245644, 0.004949451865388122, '0.26%'),

Text(0.06366719390725226, 0.5966125111158634, '46.09%'),

Text(-0.0685865314928057, -0.5960670161129421, '53.65%')])



## Pie plot indicates that Random Forest Regressor has highest r2 score

 We evaluate the performace by calculating MSE MAE RMSE

from sklearn.metrics import mean\_squared\_error from sklearn.metrics import mean\_absolute\_error

mse=mean\_squared\_error(y\_test,random\_forest\_pred) print('Mean Squared Error:',mse)

mae=mean\_absolute\_error(y\_test,random\_forest\_pred) print('Mean Absolute Error:',mae)

rmse=np.sqrt(mse)

print('Root Mean Squared Error:',rmse)

nrmse = round(np.sqrt(mse)/(max(y\_test)-min(y\_test)),2) print('Normalized Root Mean Squared Error:',nrmse)

print('Max Value:',max(y)) print('Min Value',min(y))

Mean Squared Error: 4572937.277253112 Mean Absolute Error: 1250.011887263124

Root Mean Squared Error: 2138.442722462566 Normalized Root Mean Squared Error: 0.04

Max Value: 79512

Min Value 1759

##  We save the trained model into binary format as 'mlproject' using pickle

import pickle

with open('mlproject','wb') as f: pickle.dump(random\_forest,f)

with open('mlproject','rb') as f: a = pickle.load(f)

##  We perform Cross Validation to find best parameters to train our Model

from sklearn.model\_selection import RandomizedSearchCV

##  Dictionary of Parameters with all possible values

random\_grid = {

'n\_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200],

'max\_depth': [5, 10, 15, 20, 25, 30],

'min\_samples\_split': [2, 5, 10, 15, 100],

'min\_samples\_leaf': [1, 2, 5, 10]

}

rf\_random = RandomizedSearchCV(estimator = random\_forest, param\_distributions = random\_grid,

scoring='neg\_mean\_squared\_error', n\_iter = 10, cv = 5, verbose=2, random\_state=42, n\_jobs = 1)

rf\_random.fit(X\_train,y\_train)

Fitting 5 folds for each of 10 candidates, totalling 50 fits

[CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=15, n\_estimators=1100; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=15, n\_estimators=1100; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=15, n\_estimators=1100; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=15, n\_estimators=1100; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=15, n\_estimators=1100; t [CV] END max\_depth=20, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=900; tot [CV] END max\_depth=20, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=900; tot [CV] END max\_depth=20, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=900; tot [CV] END max\_depth=20, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=900; tot [CV] END max\_depth=20, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=900; tot [CV] END max\_depth=30, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=1100; t [CV] END max\_depth=30, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=1100; t [CV] END max\_depth=30, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=1100; t [CV] END max\_depth=30, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=1100; t [CV] END max\_depth=30, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=1100; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=100, n\_estimators=300; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=100, n\_estimators=300; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=100, n\_estimators=300; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=100, n\_estimators=300; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=100, n\_estimators=300; t [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=400; tot [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=400; tot [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=400; tot [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=400; tot [CV] END max\_depth=25, min\_samples\_leaf=5, min\_samples\_split=5, n\_estimators=400; tot [CV] END max\_depth=25, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=100; to [CV] END max\_depth=25, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=100; to [CV] END max\_depth=25, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=100; to [CV] END max\_depth=25, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=100; to [CV] END max\_depth=25, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=100; to [CV] END max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=2, n\_estimators=200; tota [CV] END max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=2, n\_estimators=200; tota [CV] END max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=2, n\_estimators=200; tota [CV] END max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=2, n\_estimators=200; tota [CV] END max\_depth=5, min\_samples\_leaf=5, min\_samples\_split=2, n\_estimators=200; tota [CV] END max\_depth=10, min\_samples\_leaf=10, min\_samples\_split=15, n\_estimators=1100; [CV] END max\_depth=10, min\_samples\_leaf=10, min\_samples\_split=15, n\_estimators=1100; [CV] END max\_depth=10, min\_samples\_leaf=10, min\_samples\_split=15, n\_estimators=1100; [CV] END max\_depth=10, min\_samples\_leaf=10, min\_samples\_split=15, n\_estimators=1100; [CV] END max\_depth=10, min\_samples\_leaf=10, min\_samples\_split=15, n\_estimators=1100; [CV] END max\_depth=30, min\_samples\_leaf=1, min\_samples\_split=15, n\_estimators=300; to [CV] END max\_depth=30, min\_samples\_leaf=1, min\_samples\_split=15, n\_estimators=300; to [CV] END max\_depth=30, min\_samples\_leaf=1, min\_samples\_split=15, n\_estimators=300; to [CV] END max\_depth=30, min\_samples\_leaf=1, min\_samples\_split=15, n\_estimators=300; to [CV] END max\_depth=30, min\_samples\_leaf=1, min\_samples\_split=15, n\_estimators=300; to [CV] END max\_depth=10, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=700; to [CV] END max\_depth=10, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=700; to [CV] END max\_depth=10, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=700; to [CV] END max\_depth=10, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=700; to [CV] END max\_depth=10, min\_samples\_leaf=2, min\_samples\_split=10, n\_estimators=700; to

**▸**

**RandomizedSearchCV**

**▸ estimator: RandomForestRegressor**

▸ RandomForestRegressor

##  Finding best parameters to train our model

rf\_random.best\_params\_

{'n\_estimators': 300,

'min\_samples\_split': 15,

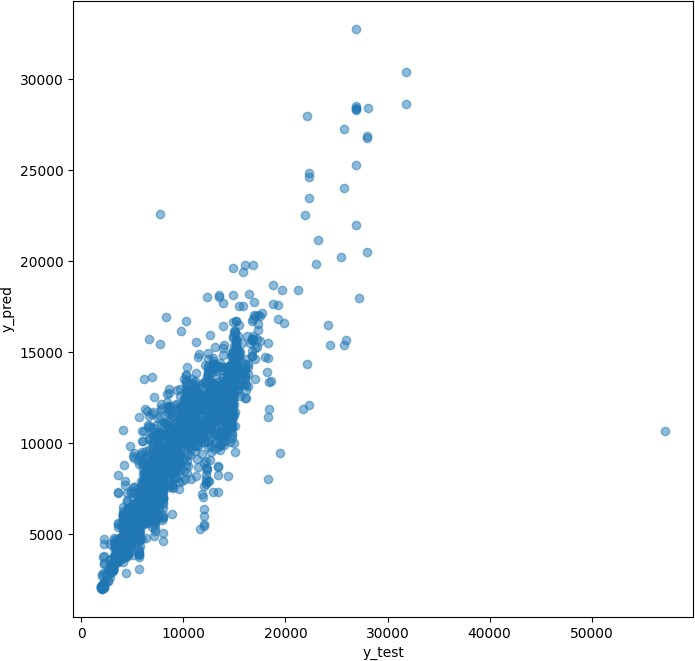
'min\_samples\_leaf': 1,

'max\_depth': 30}

prediction = rf\_random.predict(X\_test) plt.figure(figsize = (8,8))

plt.scatter(y\_test, prediction, alpha = 0.5) plt.xlabel("y\_test")

plt.ylabel("y\_pred") plt.show()



 Evaluating Performance after Cross Validation

from sklearn import metrics

print('R2 value: ', round(metrics.r2\_score(y\_test, prediction),2))

print('RMSE: ', round(np.sqrt(metrics.mean\_squared\_error(y\_test, prediction)),2))

print('Normalized RMSE: ', round(np.sqrt(metrics.mean\_squared\_error(y\_test, prediction))/(max(y\_test)-min print('Max Value: ', max(y\_test), '\nMin Value: ', min(y\_test))

R2 value: 0.79

RMSE: 2122.46

Normalized RMSE: 0.04

Max Value: 57209

Min Value: 1965

import matplotlib.pyplot as plt

# R^2 scores before and after hyperparameter tuning

r2\_scores = {'Before Tuning': 0.73, 'After Tuning': 0.79}

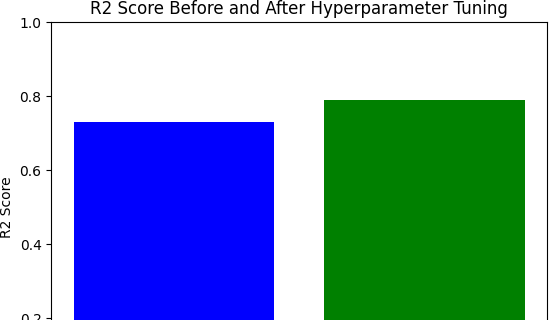
# Plotting the R^2 scores

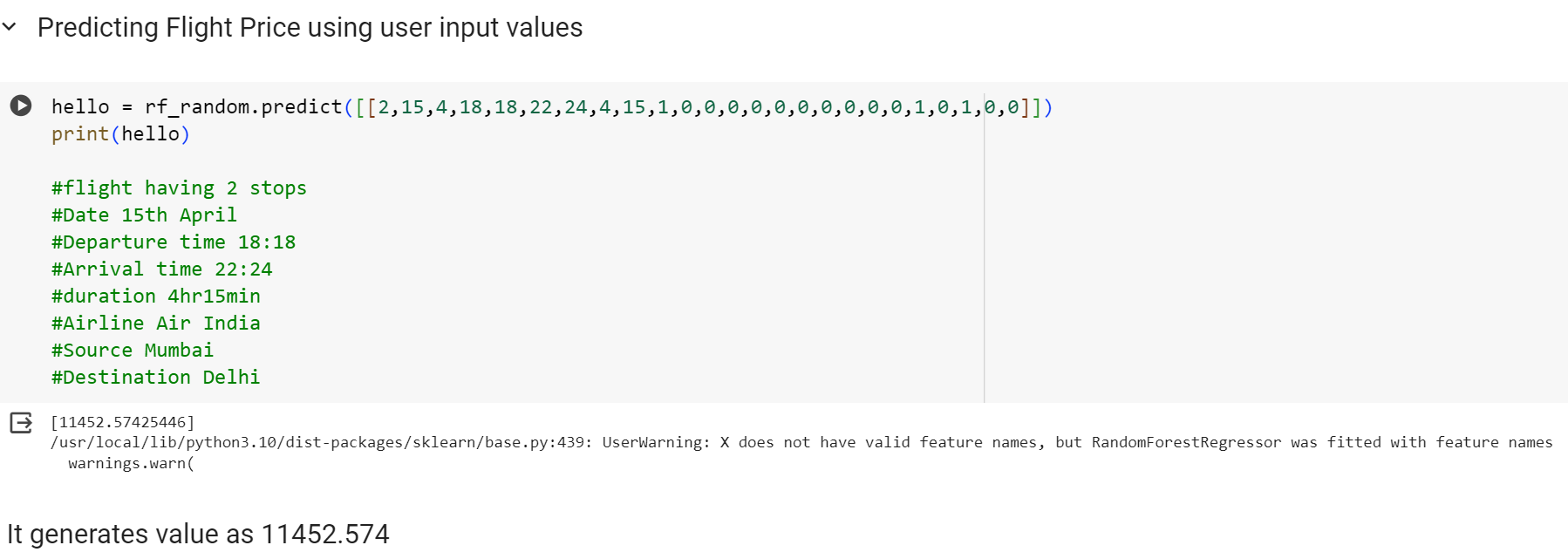
plt.bar(r2\_scores.keys(), r2\_scores.values(), color=['blue', 'green']) plt.xlabel('Scenario')

plt.ylabel('R2 Score')

plt.title('R2 Score Before and After Hyperparameter Tuning') plt.ylim(0, 1) # Set y-axis limits to ensure consistency

plt.show()





Deploying Model Using Flask

