FLIGHT PRICE PREDICTION USING PYTHON

> *TODO- Introducing ML project for flight price prediction. in this presentation, I will go-through the whole process of creating a ML model.*

Problem Statement:

Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on -
1. Time of purchase patterns (making sure last-minute purchases are expensive).
2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases).

So, here we have to work on a project where we have collected data of flight fares with other features and work to make a model to predict fares of flights.

Here's a step-by-step outline of the project:
1.Importing necessary libraries.
2.Exploratory Data Analysis (EDA).
3.Data Preprocessing.
4. Feature Engineering.
5.Model building and Saving.

Importing necessary libraries :

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Importing Dataset :

```
df_flight = pd.read_csv("Flight_Data.csv")

df_flight.head()
```

	Unnamed: 0	Airline	Departure	Arrival	Dep_time	Arr_time	Duration	Journey_Type	Price
0	0	IndiGo	New Delhi	Varanasi	15:30	16:50	1h 20m	1 Stop	2,060
1	1	IndiGo	New Delhi	Varanasi	05:05	06:30	1h 25m	1 Stop	2,060
2	2	Go First	New Delhi	Varanasi	09:10	10:40	1h 30m	2 Stop(s)	2,060
3	3	IndiGo	New Delhi	Varanasi	17:45	19:20	1h 35m	1 Stop	2,060
4	4	SpiceJet	New Delhi	Varanasi	17:30	19:00	1h 30m	1 Stop	2,068

Through the head().method we can observed 9 rows. Here we can see there are many features which are in categorical so we have to convert into numeric in order to feed our data to machine learning model. As per our problem statement we have to predict the fare. So the problem is regression and need build regression model.

Checking the shape and other info of date set :

```
df_flight.shape

(1562, 9)

df_flight.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1562 entries, 0 to 1561
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Unnamed: 0    1562 non-null   int64 
 1   Airline       1562 non-null   object
 2   Departure     1562 non-null   object
 3   Arrival       1562 non-null   object
 4   Dep_time      1562 non-null   object
 5   Arr_time      1562 non-null   object
 6   Duration      1562 non-null   object
 7   Journey_Type  1562 non-null   object
 8   Price         1562 non-null   object
dtypes: int64(1), object(8)
memory usage: 110.0+ KB
```

Data Preprocessing & Exploratory Data Analysis :

The data was further processed based on the parameters mentioned below and cleaned based on appropriate considerations -

1.Departure (origin city)
2. Arrival (Arrival city)
3. Departure time
4. Departure time
5. Duration
6. Journey_Type (Total stops)

```
df_flight.isnull().sum()


Unnamed: 0      0
Airline         0
Departure       0
Arrival         0
Dep_time        0
Arr_time        0
Duration        0
Journey_Type    0
Price           0
dtype: int64
```

As we can check above there is no null values. However, there are few columns like “arr_time” contains nanoseconds which might not be much effective values hence we can drop them.

> *Checking the unique values of the all columns:*

```
df_flight.columns
```

```
Index(['Unnamed: 0', 'Airline', 'Departure', 'Arrival',
'Dep_time', 'Arr_time',
       'Duration', 'Journey_Type', 'Price'],
      dtype='object
```

```
for column in df_flight:
    print(f'{column} : {df_flight[column].unique()}')
```

```
Unnamed: 0 : [   0    1    2 ... 1559 1560 1561]
Airline : ['IndiGo' 'Go First' 'SpiceJet' 'Vistara' 'Air
India' 'Air Asia' 'TruJet']
Departure : ['New Delhi' 'Mumbai' 'Bangalore' 'Chennai'
'Hyderabad' 'Pune' 'Kochi'
 'Kolkata']
Arrival : ['Varanasi' 'Ahmedabad' 'Jammu' 'Goa' 'Hyderabad'
'Nagpur' 'Kochi'
 'Bagdogra' 'Indore' 'New Delhi' 'Chennai' 'Bangalore'
'Kolkata'
 'Chandigarh' 'Mumbai' 'Visakhapatnam' 'Pune']
Dep_time : ['15:30' '05:05' '09:10' '17:45' '17:30' '06:25'
'13:20' '18:55' '10:25'
 '10:15' '05:45' '06:00' '17:50' '20:40' '21:00' '19:00'
'21:40' '02:00'
 '22:30' '10:20' '09:30' '05:50' '16:55' '12:50' '20:00'
'10:40' '09:00'
 '07:30' '19:50' '11:40' '08:50' '07:00' '17:15' '06:05'
'17:10' '15:05'
 '13:40' '14:20' '17:20' '05:15' '06:40' '13:30' '22:15'
'11:25' '06:10'
 '20:10' '18:30' '20:20' '19:55' '15:00' '13:00' '07:50'
'05:00' '21:20'
 '06:55' '19:10' '08:45' '10:35' '07:25' '12:55' '08:00'
'10:30' '09:20'
 '17:00' '13:10' '14:15' '07:35' '13:25' '09:35' '15:15'
'10:50' '11:55'
 '09:55' '09:15' '11:45' '23:35' '02:10' '21:55' '23:45'
'04:45' '00:05'
 '14:05' '15:20' '04:55' '22:05' '05:30' '11:15' '16:20'
'12:00' '09:25'
 '19:05' '11:30' '16:00' '13:05' '19:20' '10:45' '06:20'
'22:40' '10:00'
 '21:30' '12:15' '18:00' '09:05' '15:25' '07:05' '22:55'
'21:15' '07:20'
 '20:15' '22:45' '20:55' '14:40' '19:45' '15:45' '14:35'
'22:50' '06:15'
```

```
 '12:45' '06:50' '16:45' '06:30' '12:25' '09:50' '05:10'
'19:40' '18:05'
 '18:20' '08:35' '14:00' '18:40' '05:55' '23:10' '08:10'
'07:15' '06:45'
 '04:15' '02:30' '21:45' '12:05' '16:50' '05:35' '07:55'
'15:10' '21:25'
 '19:30' '16:05' '20:45' '04:25' '18:35' '14:10' '19:35'
'09:45' '11:10'
 '13:50' '15:50' '08:15' '23:20' '07:40' '07:10' '18:45'
'14:30' '16:25'
 '18:50' '16:15' '15:35' '18:15' '14:50' '10:55' '14:45'
'08:30' '18:25'
 '05:25' '11:50' '11:05' '21:10' '05:40' '11:00' '16:30'
'04:40' '19:15'
 '22:10' '23:00' '10:05' '02:40' '11:20' '00:10' '00:45'
'17:25' '17:05'
 '17:35' '20:50' '15:40' '09:40' '13:55' '21:50' '22:25'
'13:45' '17:40'
 '21:05' '22:35' '23:30' '16:35' '08:25' '06:35' '20:30'
'12:30' '14:55'
 '23:05' '08:20' '12:10' '19:25' '18:10' '23:15' '01:05'
'22:20' '20:35'
 '15:55' '08:05' '11:35' '13:35' '21:35' '16:10' '13:15'
'08:40' '12:35'
 '16:40' '01:15' '20:25' '23:40' '07:45']
Arr_time : ['16:50' '06:30' '10:40' '19:20' '19:00' '08:20'
'14:45' '20:10' '11:55'
 '11:35' '15:35' '12:30' '12:40\n+ 1 day' '15:35\n+ 1 day'
 '13:20\n+ 1 day' '13:20' '13:40' '12:40' '16:25' '06:50'
'08:15' '15:05'
 '23:50' '13:10' '07:30' '07:40' '17:05' '21:50' '19:55'
'22:00' '21:30'
 '07:55\n+ 1 day' '20:15' '15:30' '09:10' '20:40' '10:20'
'17:35' '14:25'
 '22:55' '17:00' '19:05' '20:25' '18:30' '21:55' '08:50'
'10:45'
 '00:05\n+ 1 day' '18:20' '14:35' '11:05' '12:20' '13:30'
'12:45' '15:50'
 '13:15\n+ 1 day' '13:15' '00:35\n+ 1 day' '03:35' '23:00'
 '01:00\n+ 1 day' '05:55' '01:15' '06:25' '15:15' '16:30'
'06:10' '23:25'
 '22:25' '12:50' '17:45' '18:00' '14:05\n+ 1 day' '20:30'
'12:35'
 '00:35\n+ 2 days' '23:45' '23:10' '14:05' '17:10' '14:55\n+
1 day'
 '15:25\n+ 1 day' '13:30\n+ 1 day' '22:35' '21:20\n+ 1 day'
 '17:00\n+ 1 day' '13:50\n+ 1 day' '14:55' '06:50\n+ 1 day'
 '08:45\n+ 1 day' '17:40' '23:20' '15:25' '08:30\n+ 1 day'
 '09:30\n+ 1 day' '07:35' '07:50' '21:25' '20:55' '08:30'
'12:05' '12:15'
 '21:35' '18:35' '20:55\n+ 1 day' '19:20\n+ 1 day' '19:15'
'23:35'
 '10:30\n+ 1 day' '11:55\n+ 1 day' '21:05\n+ 1 day'
'12:25\n+ 1 day'
 '15:20' '21:05' '12:10\n+ 1 day' '12:10' '10:00' '09:40\n+
1 day' '23:40'
 '19:10' '08:40' '17:35\n+ 1 day' '09:55\n+ 1 day' '08:00'
 '07:50\n+ 1 day' '08:55\n+ 1 day' '07:00' '21:45' '08:35'
'17:20' '15:40'
```

```
 '20:00' '15:10' '20:35' '20:35\n+ 1 day' '15:00' '05:35\n+
1 day'
 '02:05\n+ 1 day' '00:45\n+ 1 day' '05:05\n+ 1 day' '20:50'
'10:05'
 '19:50' '08:45' '16:10' '09:30' '04:25\n+ 1 day' '12:00'
'22:10' '18:40'
 '13:35' '10:55' '07:00\n+ 1 day' '08:05' '22:45' '07:10\n+
1 day' '19:25'
 '02:25\n+ 1 day' '14:20' '14:15' '22:50' '09:10\n+ 1 day'
 '19:15\n+ 1 day' '09:45' '09:20' '11:50' '15:55' '16:05'
'11:40' '14:10'
 '10:20\n+ 1 day' '10:15' '11:45' '06:40\n+ 1 day' '08:25\n+
1 day'
 '07:20' '11:30' '20:05' '17:50' '12:55' '07:45\n+ 1 day'
'22:20' '23:55'
 '18:10' '19:30' '12:15\n+ 1 day' '07:15\n+ 1 day' '10:00\n+
1 day'
 '21:15' '08:40\n+ 1 day' '00:25\n+ 1 day' '23:30' '14:30'
'15:45'
 '00:30\n+ 1 day' '10:40\n+ 1 day' '01:45\n+ 1 day' '12:25'
'16:15'
 '02:10\n+ 1 day' '10:30' '20:20' '16:45' '14:50' '20:45'
'01:05\n+ 1 day'
 '02:30\n+ 1 day' '10:50' '04:15\n+ 1 day' '01:40\n+ 1 day'
'13:00'
 '09:40' '02:55' '08:55' '16:35\n+ 1 day' '04:40\n+ 1 day'
 '10:15\n+ 1 day' '11:10\n+ 1 day' '18:15\n+ 1 day'
'09:00\n+ 1 day'
 '15:10\n+ 1 day' '19:55\n+ 1 day' '03:40\n+ 1 day'
'18:30\n+ 1 day'
 '00:55\n+ 1 day' '00:10\n+ 1 day' '13:50' '21:50\n+ 1 day'
'18:45'
 '21:30\n+ 1 day' '23:55\n+ 1 day' '01:15\n+ 1 day'
'00:40\n+ 1 day'
 '00:50\n+ 1 day' '02:00\n+ 1 day' '14:20\n+ 1 day'
'03:35\n+ 1 day'
 '02:55\n+ 1 day' '21:00' '09:00' '11:25' '09:35\n+ 1 day'
 '12:30\n+ 1 day' '08:05\n+ 1 day' '08:35\n+ 1 day'
'16:55\n+ 1 day'
 '11:00\n+ 1 day' '18:05' '16:55' '19:40' '14:30\n+ 1 day'
 '23:45\n+ 1 day' '17:55' '21:20' '08:00\n+ 1 day' '21:10'
 '01:55\n+ 1 day' '18:50' '10:25' '05:55\n+ 1 day' '01:10\n+
1 day'
 '17:25' '07:40\n+ 1 day' '10:10' '16:20' '12:50\n+ 1 day'
 '09:05\n+ 1 day' '11:50\n+ 1 day' '17:05\n+ 1 day' '23:05'
 '07:20\n+ 1 day' '10:55\n+ 1 day' '11:20\n+ 1 day' '19:35'
'16:40'
 '21:10\n+ 1 day' '14:40' '14:00' '00:20\n+ 1 day' '16:25\n+
1 day'
 '21:25\n+ 1 day' '01:25\n+ 1 day' '01:35\n+ 1 day' '09:05'
 '07:30\n+ 1 day' '09:35' '09:25' '18:15' '10:35\n+ 1 day'
 '07:25\n+ 1 day' '18:25' '23:15' '04:15' '07:55' '03:20'
'11:15' '16:35'
 '07:15' '09:15' '22:30' '18:05\n+ 1 day' '19:40\n+ 1 day'
'13:55' '17:30'
 '10:35' '13:35\n+ 1 day' '08:15\n+ 1 day' '11:00' '16:00'
'21:40' '17:15'
 '07:05' '18:55' '08:10\n+ 1 day' '10:10\n+ 1 day' '09:50\n+
1 day'
```

```
 '10:05\n+ 1 day' '13:00\n+ 1 day' '20:30\n+ 1 day'
'21:15\n+ 1 day'
 '19:45' '03:45' '01:20\n+ 1 day' '14:25\n+ 1 day' '22:05'
 '23:00\n+ 1 day' '18:40\n+ 1 day' '08:50\n+ 1 day'
'14:40\n+ 1 day'
 '19:50\n+ 1 day']
Duration : ['1h 20m' '1h 25m' '1h 30m' '1h 35m' '1h 55m' '1h
15m' '9h 50m' '6h 30m'
 '18h 50m' '18h 55m' '16h 20m' '17h 40m' '17h 55m' '11h 20m'
'14h 50m'
 '5h 15m' '6h 05m' '7h 50m' '22h 40m' '26h 45m' '16h 40m'
'26h 00m'
 '27h 40m' '8h 05m' '19h 45m' '24h 05m' '27h 55m' '6h 45m'
'5h 40m'
 '19h 25m' '20h 40m' '21h 50m' '10h 20m' '22h 25m' '21h 35m'
'25h 55m'
 '16h 00m' '25h 15m' '19h 20m' '1h 45m' '1h 40m' '12h 55m'
'7h 15m'
 '7h 40m' '9h 30m' '10h 30m' '10h 35m' '2h 15m' '7h 00m' '8h
35m'
 '10h 00m' '6h 20m' '8h 00m' '11h 05m' '9h 55m' '5h 35m' '6h
25m' '6h 50m'
 '8h 45m' '9h 45m' '8h 25m' '2h 55m' '1h 10m' '2h 50m' '4h
40m' '5h 20m'
 '8h 10m' '25h 35m' '16h 15m' '11h 15m' '14h 45m' '1h 00m'
'1h 05m'
 '17h 05m' '20h 35m' '10h 45m' '12h 40m' '5h 10m' '22h 05m'
'8h 55m'
 '11h 00m' '18h 45m' '27h 20m' '31h 45m' '17h 35m' '14h 00m'
'14h 35m'
 '16h 35m' '27h 05m' '9h 40m' '10h 55m' '13h 15m' '15h 10m'
'6h 35m'
 '5h 00m' '7h 05m' '9h 10m' '30h 50m' '17h 30m' '16h 30m'
'13h 45m'
 '15h 15m' '23h 25m' '20h 45m' '22h 35m' '15h 55m' '15h 05m'
'24h 25m'
 '23h 10m' '26h 50m' '7h 35m' '16h 55m' '18h 05m' '20h 20m'
'34h 35m'
 '10h 40m' '11h 25m' '12h 35m' '8h 15m' '8h 20m' '9h 35m'
'10h 10m'
 '13h 05m' '23h 55m' '25h 45m' '9h 20m' '12h 15m' '12h 20m'
'14h 30m'
 '13h 00m' '13h 35m' '14h 25m' '15h 30m' '15h 35m' '19h 40m'
'26h 55m'
 '4h 55m' '11h 50m' '13h 55m' '14h 55m' '13h 25m' '14h 15m'
'5h 25m'
 '6h 10m' '7h 10m' '7h 45m' '4h 50m' '5h 05m' '8h 30m' '8h
50m' '3h 25m'
 '31h 35m' '33h 35m' '34h 30m' '24h 55m' '7h 55m' '18h 30m'
'10h 50m'
 '12h 10m' '13h 40m' '3h 20m' '14h 40m' '38h 40m' '6h 15m'
'4h 45m'
 '24h 35m' '5h 30m' '9h 15m' '37h 40m' '11h 35m' '5h 45m'
'9h 25m'
 '7h 20m' '3h 00m' '3h 10m' '3h 15m' '5h 50m' '5h 55m' '6h
40m' '7h 25m'
 '8h 40m' '11h 30m' '12h 25m' '16h 10m' '7h 30m' '10h 15m'
'11h 55m'
 '15h 50m' '27h 00m' '23h 15m' '24h 15m' '25h 25m' '9h 05m'
```

```
'28h 10m'
 '2h 00m' '2h 05m' '13h 20m' '4h 25m' '4h 30m' '21h 45m' '2h
30m'
 '11h 10m' '12h 30m' '14h 10m' '3h 45m' '12h 05m' '4h 05m'
'9h 00m'
 '4h 00m' '4h 35m' '12h 50m' '13h 50m' '17h 15m' '17h 50m'
'23h 20m'
 '25h 00m' '25h 10m' '26h 05m' '24h 30m' '11h 45m' '25h 20m'
'15h 00m'
 '17h 20m' '18h 20m' '13h 10m' '17h 45m' '21h 25m' '2h 40m'
'2h 45m'
 '6h 00m' '2h 35m' '26h 30m' '21h 40m' '25h 30m' '19h 05m'
'23h 05m'
 '16h 45m' '19h 15m' '15h 25m' '19h 00m' '24h 00m' '23h 40m'
'25h 50m'
 '26h 40m' '3h 05m' '27h 10m' '20h 25m' '15h 45m' '11h 40m'
'18h 00m'
 '32h 50m' '25h 40m' '6h 55m' '12h 00m' '14h 20m' '24h 40m'
'26h 25m'
 '19h 30m' '20h 50m' '24h 10m' '22h 15m' '23h 50m' '13h 30m'
'22h 20m'
 '24h 20m' '4h 20m' '2h 10m' '2h 20m' '10h 25m' '0h 55m' '4h
15m'
 '10h 05m' '15h 40m' '16h 25m' '20h 30m' '31h 05m' '21h 20m'
'26h 15m'
 '4h 10m' '3h 40m' '18h 25m' '20h 15m' '31h 50m' '34h 50m'
'19h 35m'
 '23h 35m' '12h 45m' '14h 05m' '16h 05m' '25h 05m' '2h 25m'
'28h 40m'
 '21h 30m' '28h 00m' '18h 10m']
Journey_Type : ['1 Stop' '2 Stop(s)' '3 Stop(s)' '4 Stop(s)'
'Non Stop']
Price : ['2,060' '2,068' '2,154' '2,625' '2,793' '3,666'
'5,674' '5,819' '5,894'
 '6,023' '6,067' '6,209' '6,230' '6,401' '6,409' '6,521'
'6,548' '6,629'
 '6,724' '6,758' '6,934' '6,960' '7,249' '7,283' '7,741'
'8,644' '8,729'
 '8,854' '9,110' '9,422' '9,444' '9,599' '9,654' '2,161'
'2,312' '2,341'
 '2,418' '3,115' '3,120' '3,215' '3,446' '3,702' '3,820'
'3,994' '4,259'
 '4,260' '4,303' '4,353' '4,364' '4,393' '4,593' '4,651'
'4,723' '2,172'
 '2,256' '2,275' '2,592' '2,803' '2,989' '3,015' '3,219'
'3,593' '4,065'
 '5,845' '6,093' '6,139' '6,300' '6,634' '2,594' '2,688'
'2,804' '3,053'
 '3,118' '3,368' '3,780' '4,305' '4,484' '4,628' '5,100'
'5,297' '5,350'
 '5,523' '5,599' '5,628' '5,665' '6,035' '6,384' '6,483'
'6,910' '7,029'
 '7,225' '7,256' '7,401' '7,645' '7,865' '8,170' '8,195'
'8,425' '8,462'
 '8,590' '9,475' '9,518' '9,632' '10,167' '10,427' '10,563'
'10,626'
 '10,952' '11,069' '11,151' '11,162' '11,477' '11,595'
'11,687' '11,996'
 '12,002' '12,212' '12,483' '12,798' '4,261' '4,262' '4,263'
```

```
'4,421'
 '4,473' '4,578' '4,736' '4,840' '5,471' '5,825' '6,899'
'7,584' '7,585'
 '8,046' '8,148' '8,283' '8,926' '8,936' '8,988' '9,356'
'9,879' '10,058'
 '10,103' '10,406' '10,593' '10,644' '10,931' '11,036'
'11,141' '11,351'
 '11,403' '11,452' '11,639' '11,928' '12,410' '13,086'
'5,114' '5,115'
 '5,430' '5,572' '6,900' '6,942' '9,683' '9,945' '10,401'
'10,474'
 '10,523' '10,628' '10,680' '10,712' '10,785' '10,985'
'10,995' '11,166'
 '11,310' '11,352' '11,861' '12,255' '12,728' '9,105'
'9,106' '9,357'
 '9,472' '9,525' '10,155' '10,242' '10,312' '10,470'
'11,940' '12,238'
 '12,360' '12,964' '13,568' '13,620' '5,690' '5,640' '5,850'
'6,485'
 '6,533' '7,163' '8,003' '8,055' '8,370' '8,738' '9,053'
'3,289' '4,337'
 '4,338' '4,339' '4,548' '4,707' '5,509' '5,754' '5,808'
'5,966' '7,121'
 '7,278' '7,532' '7,813' '7,948' '8,224' '8,260' '8,359'
'9,379' '9,850'
 '9,880' '10,205' '11,205' '11,383' '11,420' '11,509'
'11,584' '11,919'
 '5,109' '5,494' '5,598' '5,985' '6,144' '7,486' '7,487'
'7,488' '7,489'
 '8,505' '8,780' '9,054' '9,222' '9,484' '9,720' '9,771'
'9,904' '10,009'
 '10,071' '10,124' '10,272' '10,324' '10,534' '10,717'
'10,744' '5,149'
 '5,725' '5,849' '5,867' '6,072' '6,347' '6,410' '7,352'
'7,353' '7,458'
 '7,562' '7,877' '7,998' '8,193' '8,685' '9,419' '10,049'
'10,136'
 '10,243' '10,398' '10,443' '10,608' '10,743' '10,976'
'11,448' '11,553'
 '11,944' '12,527' '12,708' '5,738' '5,942' '5,952' '6,495'
'7,311'
 '7,425' '7,426' '7,530' '7,635' '7,933' '7,976' '8,265'
'9,195' '9,630'
 '9,736' '9,840' '9,925' '10,428' '10,496' '11,100' '11,520'
'11,625'
 '11,629' '5,444' '5,864' '5,937' '6,417' '6,480' '7,241'
'7,423' '7,424'
 '5,429' '5,953' '5,954' '5,955' '3,434' '3,435' '3,488'
'3,645' '4,275'
 '4,941' '5,023' '5,949' '4,899' '6,132' '6,133' '6,134'
'6,343' '6,344'
 '6,553' '6,868' '7,183' '7,236' '7,392' '7,393' '8,233'
'8,496' '8,594'
 '9,073' '9,074' '9,126' '9,283' '9,441' '10,018' '10,123'
'10,438'
 '10,753' '10,858' '10,989' '11,016' '11,331' '11,488'
'4,453' '4,454'
 '4,829' '6,763' '7,603' '8,928' '13,759' '14,221' '14,683'
'16,003'
```

```
 '5,385' '5,956' '6,060' '6,376' '6,377' '6,795' '6,932'
'7,005' '7,007'
 '5,000' '5,688' '6,271' '6,272' '6,273' '6,274' '6,693'
'6,798' '6,851'
 '6,901' '7,008' '7,113' '8,373' '9,161' '9,528' '9,896'
'10,001' '4,102'
 '5,176' '5,178' '5,179' '7,646' '8,066' '8,328' '11,164'
'15,262'
 '16,197' '17,359' '7,549' '7,617' '9,030' '9,034' '9,035'
'9,401' '9,806'
 '10,293' '10,767' '13,549' '6,487' '6,488' '6,489' '6,490'
'6,594'
 '6,804' '6,907' '6,909' '7,118' '7,119' '7,329' '7,434'
'7,466' '7,954'
 '8,064' '8,442' '8,930' '9,324' '9,534' '9,907' '10,059'
'10,112'
 '10,395' '10,720' '11,634' '11,844' '11,860' '12,054'
'12,159' '12,551'
 '12,894' '13,314' '13,325' '13,386' '13,524' '13,734'
'13,813' '13,839'
 '13,944' '14,049' '14,364' '14,574' '14,710' '14,772'
'14,889']
```

As checked unique values above we have to do clean data values and make useful for ML model. In the price column our target variable as ‘,’ which we need to remove similarly columns like “Dep_time & Arr_time” also has categorical which we can convert into datetime format and create new features.

> ***Converting Dep_time column into 2 columns “Dep_hrs and “Dep_min”***

```
df_flight['Price'] =
df_flight['Price'].str.replace(',','').astype('int')


df_flight['Dep_hour'] =
pd.to_datetime(df_flight['Dep_time']).dt.hour


df_flight['Dep_min'] =
pd.to_datetime(df_flight['Dep_time']).dt.minute


df_flight.drop(['Dep_time'],axis=1,inplace= True)




df_flight['Arr_time'] =
df_flight['Arr_time'].str.replace('\n','')
```

```
df_flight['Arr_time'] =
df_flight['Arr_time'].str.replace('+','')
df_flight['Arr_time'] =
df_flight['Arr_time'].str.replace('1','')
df_flight['Arr_time'] =
df_flight['Arr_time'].str.replace('day','')


df_flight['Arr_time'].unique()


array(['6:50', '06:30', '0:40', '9:20', '9:00', '08:20',
'4:45', '20:0',
       ':55', ':35', '5:35', '2:30', '2:40  ', '5:35  ',
'3:20  ', '3:20',
       '3:40', '2:40', '6:25', '06:50', '08:5', '5:05',
'23:50', '3:0']
```

> ***Converting Arr_time column into 2 columns "Arr_hrs and "Arr_min"***

```
df_flight['Arr_time'] =
pd.to_datetime(df_flight['Arr_time'], errors = 'coerce')
```

In the above code we are removing the nanoseconds values.

```
df_flight['Arr_hour'] =
pd.to_datetime(df_flight['Arr_time']).dt.hour

df_flight['Arr_min'] =
pd.to_datetime(df_flight['Arr_time']).dt.minute

df_flight.drop(['Arr_time'],axis=1,inplace=True)
```

> Changing columns name to better understanding

```
df_flight=df_flight.rename(columns={'Journey_Type'
:'Total_Stops'})
df_flight=df_flight.rename(columns={'Departure' :'Origin'})
df_flight=df_flight.rename(columns={'Arrival'
:'Destination'})
```

> *Converting "Total_stops" columns into numeric form*

```
df_flight['Total_Stops'] =
df_flight['Total_Stops'].str.replace('1
Stop','1').replace('2 Stop(s)','2').replace('3
Stop(s)','3').replace('4 Stop(s)','4').replace('Non
Stop','0').astype('int64')


df_flight['Total_Stops'].unique()


array([1, 2, 3, 4, 0], dtype=int64)
```

> *Duration Columns has hrs and min extracting values and creating new columns.*

```
duration = list(df_flight['Duration'])

for i in range(len(duration)):
    if len(duration[i].split()) !=2: #checking if duraation
contains only hrs or min
        if "h" in duration[i]:
            duration['i']=duration[i].strip() + " 0m"  #
Adds 0 minute
        else:
            duration[i]="0h " + duration[i]   # Adds 0 hour


duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")
[0]))    # Extract hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")
[0].split()[-1]))   # Extracts only minutes from duration




# Adding duration_hours and duration_mins list to train_data
dataframe

df_flight["Duration_hours"] = duration_hours
df_flight["Duration_mins"] = duration_mins

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

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

df_flight.head()
```

	Airline	Origin	Destination	Total_Stops	Price	Dep_hour	Dep_min	Arr_hour	Arr_min	Duration_hours	Duration_mins
0	IndiGo	New Delhi	Varanasi	1	2060	15	30	6.0	50.0	1	20
1	IndiGo	New Delhi	Varanasi	1	2060	5	5	6.0	30.0	1	25
2	Go First	New Delhi	Varanasi	2	2060	9	10	0.0	40.0	1	30
3	IndiGo	New Delhi	Varanasi	1	2060	17	45	9.0	20.0	1	35
4	SpiceJet	New Delhi	Varanasi	1	2068	17	30	9.0	0.0	1	30

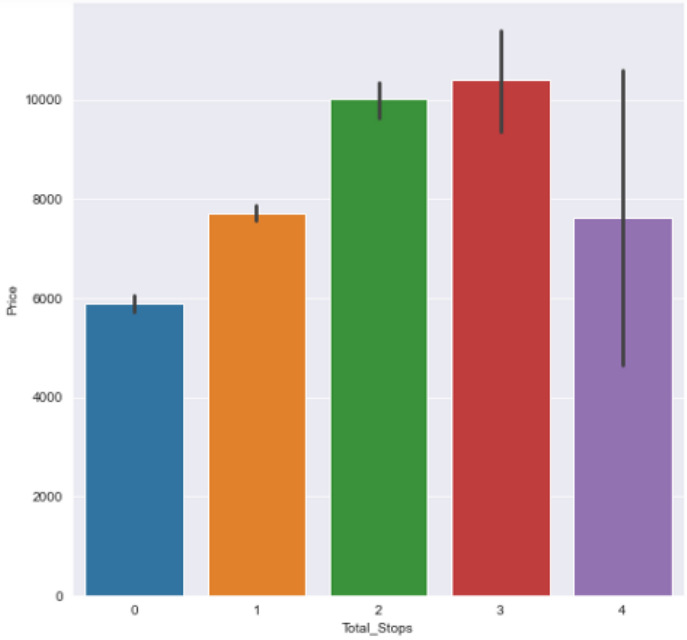
Now, we can see we have created the new columns and removed unnecessary columns. In the arr_min and arr_hrs columns we have removed nanosecond values now need to check the null values and need to fill.

```
df_flight.isnull().sum()

Airline             0
Origin              0
Destination         0
Total_Stops         0
Price               0
Dep_hour            0
Dep_min             0
Arr_hour          281
Arr_min           281
Duration_hours      0
Duration_mins       0
dtype: int64

#using mean function to fil the missing value
df_flight['Arr_hour']=df_flight['Arr_hour'].fillna(df_flight
.groupby("Origin")['Arr_hour'].transform("mean"))
df_flight['Arr_min']=df_flight['Arr_min'].fillna(df_flight.g
roupby("Origin")['Arr_min'].transform("mean"))


df_flight.isnull().sum()

Airline           0
Origin            0
Destination       0
Total_Stops       0
Price             0
Dep_hour          0
Dep_min           0
Arr_hour          0
```

```
Arr_min           0
Duration_hours    0
Duration_mins     0
dtype: int64
```

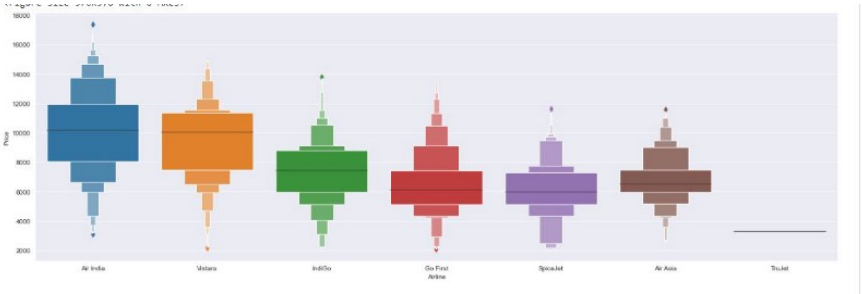
> *Visualization:*

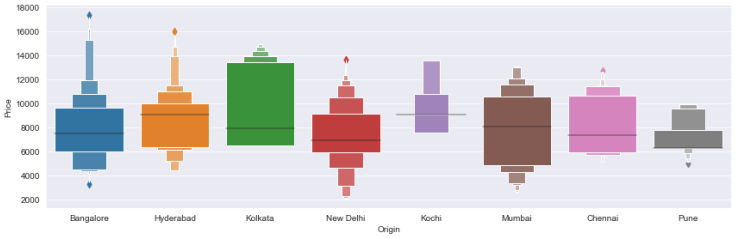
```
# checking the relation with price and total_stops
plt.figure(figsize=(8,8))
sns.barplot(x='Total_Stops',y='Price',data=df_flight)
```



as we can observe direct flight which has no stops coming cheaper and which has more then 1 price slight increased.

```
plt.figure(figsize=(8,8))
# Airline vs Price
sns.catplot(y = "Price", x = "Airline", data =
df_flight.sort_values("Price", ascending = False),
kind="boxen", height = 6, aspect = 3)
plt.show()



# Destination vs Price
sns.catplot(y = "Price", x = "Origin", data =
df_flight.sort_values("Price", ascending = False),
kind="boxen", height = 4, aspect = 3)
plt.show()
```

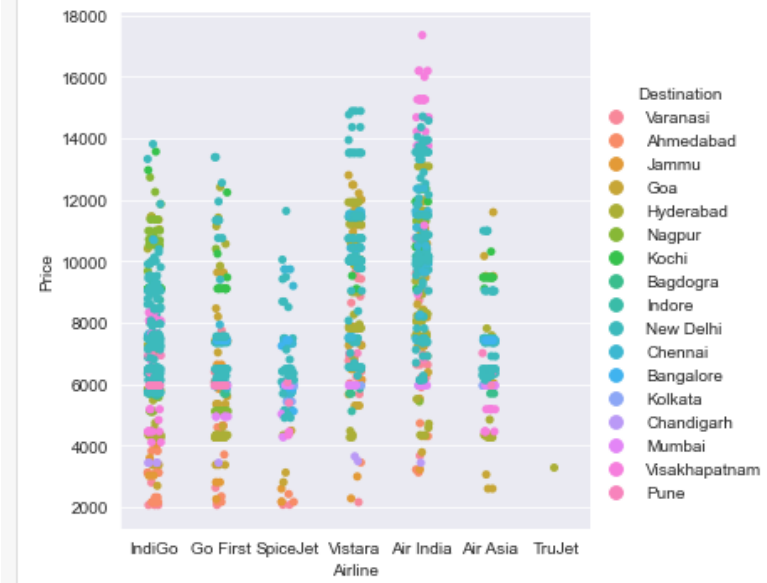




As per above trend air-india and vistara shows highest fare among them and indigo, go-air ,spicejet,airasia and truejet are lowcost airline as their fare coming cheaper .

plt.figure(figsize=(15,12)) # Airline vs Price by destination sns.catplot(x = 'Airline', y = 'Price', hue = 'Destination', data = df_flight)

```
<Figure size 1080x864 with 0 Axes>
```



Handling Categorical Data

```
df_flight=pd.get_dummies(df_flight)


pd.set_option('display.max_columns',100)
df_flight.head()
```

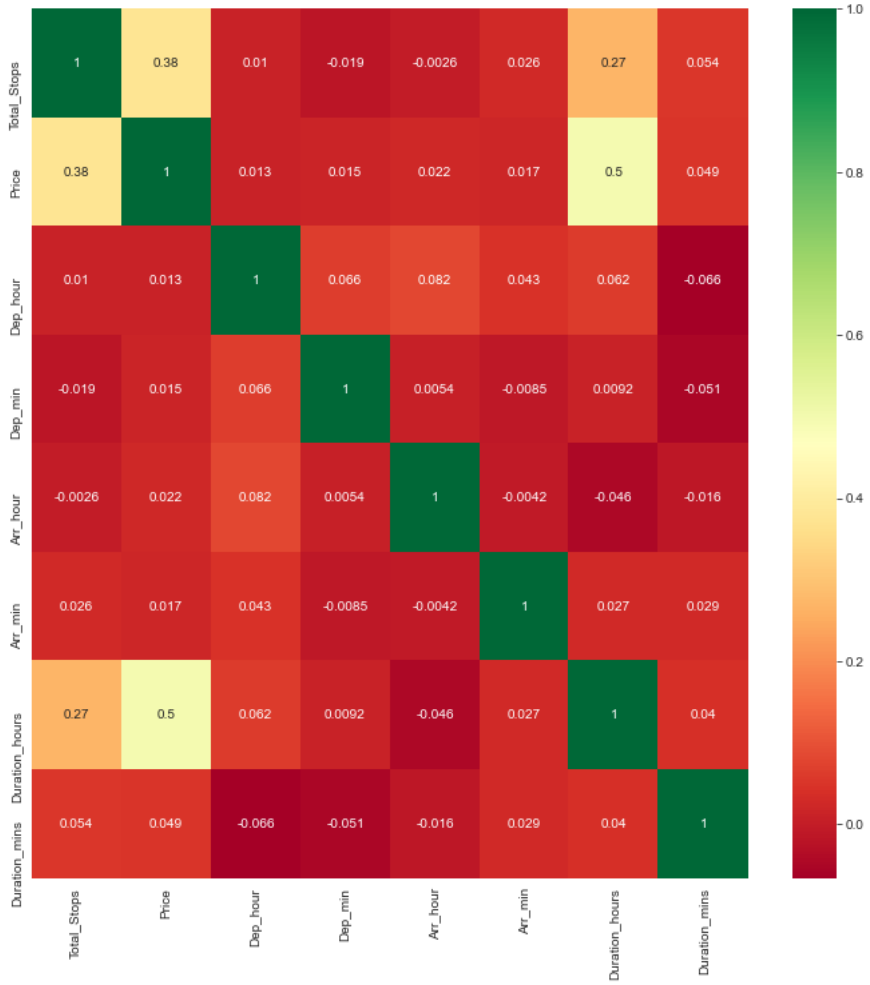
Feature Selection

Finding out the best feature which will contribute and have good relation with target variable. Following are some of the feature selection methods.

```
df_flight.columns


Index(['Total_Stops', 'Price', 'Dep_hour', 'Dep_min',
'Arr_hour', 'Arr_min',
       'Duration_hours', 'Duration_mins', 'Airline_Air
Asia',
       'Airline_Air India', 'Airline_Go First',
'Airline_IndiGo',
       'Airline_SpiceJet', 'Airline_TruJet',
'Airline_Vistara',
       'Origin_Bangalore', 'Origin_Chennai',
```

```
'Origin_Hyderabad',
       'Origin_Kochi', 'Origin_Kolkata', 'Origin_Mumbai',
'Origin_New Delhi',
       'Origin_Pune', 'Destination_Ahmedabad',
'Destination_Bagdogra',
       'Destination_Bangalore', 'Destination_Chandigarh',
       'Destination_Chennai', 'Destination_Goa',
'Destination_Hyderabad',
       'Destination_Indore', 'Destination_Jammu',
'Destination_Kochi',
       'Destination_Kolkata', 'Destination_Mumbai',
'Destination_Nagpur',
       'Destination_New Delhi', 'Destination_Pune',
'Destination_Varanasi',
       'Destination_Visakhapatnam'],
      dtype='object')
```

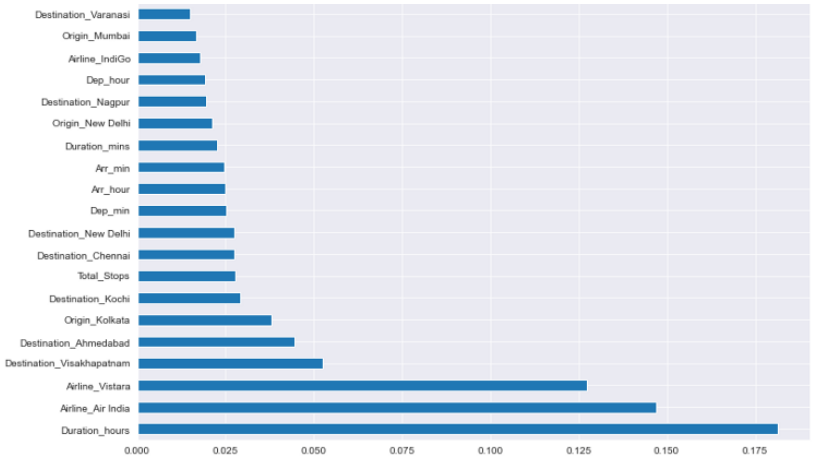
```
df_columns = df_flight[['Total_Stops', 'Price', 'Dep_hour',
'Dep_min', 'Arr_hour', 'Arr_min',
       'Duration_hours', 'Duration_mins']]

# Finds correlation between Independent and dependent
attributes
plt.figure(figsize = (12,12))
sns.heatmap(df_columns.corr(), annot = True, cmap =
"RdYlGn")

plt.show()
```



Using heatmap we can see total stops, arr_min showing good relationship but some of them are showing negative relation.

```
x = df_flight[['Total_Stops', 'Dep_hour', 'Dep_min',
'Arr_hour', 'Arr_min',
       'Duration_hours', 'Duration_mins', 'Airline_Air
Asia',
       'Airline_Air India', 'Airline_Go First',
'Airline_IndiGo',
       'Airline_SpiceJet', 'Airline_TruJet',
'Airline_Vistara',
       'Origin_Bangalore', 'Origin_Chennai',
'Origin_Hyderabad',
       'Origin_Kochi', 'Origin_Kolkata', 'Origin_Mumbai',
'Origin_New Delhi',
       'Origin_Pune', 'Destination_Ahmedabad',
'Destination_Bagdogra',
       'Destination_Bangalore', 'Destination_Chandigarh',
       'Destination_Chennai', 'Destination_Goa',
'Destination_Hyderabad',
       'Destination_Indore', 'Destination_Jammu',
```

```
'Destination_Kochi',
       'Destination_Kolkata', 'Destination_Mumbai',
'Destination_Nagpur',
       'Destination_New Delhi', 'Destination_Pune',
'Destination_Varanasi',
       'Destination_Visakhapatnam']]


y = df_flight['Price']


# Important feature using ExtraTreesRegressor
from sklearn.ensemble import ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(x, y)


#plot graph of feature importances for better visualization

plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_,
index=x.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```



Model Building

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

```
X = df_flight.drop(['Price'],axis=1)
y= df_flight['Price']


print(X.shape)
print(y.shape)


(1562, 39)
(1562,
```

> *Scalling the features value using standardscaler:*

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scale = scaler.fit_transform(X)


x_scale
```

> *Spliting in train & test :*

```
X_train, X_test, y_train, y_test  =
train_test_split(x_scale, y, test_size=0.30,random_state=42)


lr=LinearRegression()
knn=KNeighborsRegressor()
dt=DecisionTreeRegressor()
rf=RandomForestRegressor()
adb=AdaBoostRegressor()
svm=SVR()
gboost=GradientBoostingRegressor()
xgboost=XGBRegressor()
print("Model is created")


#Model is trained


lr.fit(X_train, y_train)
knn.fit(X_train, y_train)
dt.fit(X_train, y_train)
rf.fit(X_train, y_train)
adb.fit(X_train, y_train)
svm.fit(X_train, y_train)
gboost.fit(X_train, y_train)
xgboost.fit(X_train, y_train)
print("Model is trained")
```

```
print("lr_score",lr.score(X_train, y_train))
print("knn_score",knn.score(X_train, y_train))
print("dt_score",dt.score(X_train, y_train))
print("rf_score",rf.score(X_train, y_train))
print("adb_score",adb.score(X_train, y_train))
print("svm_score",svm.score(X_train, y_train))
print("gboost_score",gboost.score(X_train, y_train))
print("xgboost_score",xgboost.score(X_train, y_train))
```

```
lr_score 0.6545256430920336
knn_score 0.7983554177227351
dt_score 0.9999524243951425
rf_score 0.97693206560485
adb_score 0.5856223329669882
svm_score -0.022058439806933006
gboost_score 0.8215614421496338
xgboost_score 0.9974358831133999
```

> *Model Evaluation*

```
from sklearn.metrics import
mean_absolute_error,mean_squared_error


lr_pred_y = lr.predict(X_test)
knn_pred_y = knn.predict(X_test)
dt_pred_y = dt.predict(X_test)
rf_pred_y = rf.predict(X_test)
adb_pred_y = adb.predict(X_test)
svm_pred_y = svm.predict(X_test)
gboost_pred_y = svm.predict(X_test)
xgboost_pred_y = svm.predict(X_test)
```

```
print("lr_score",mean_squared_error(y_test,lr_pred_y))
print("knn_score",mean_squared_error(y_test,knn_pred_y))
print("dt_score",mean_squared_error(y_test,dt_pred_y))
print("rf_score",mean_squared_error(y_test,rf_pred_y))
print("adb_score",mean_squared_error(y_test,adb_pred_y))
print("svm_score",mean_squared_error(y_test,svm_pred_y))
print("gboost_score",mean_squared_error(y_test,gboost_pred_y
))
print("xgboost_score",mean_squared_error(y_test,xgboost_pred
_y))
```

```
lr_score 1.7037812648936504e+34
knn_score 2260411.2698507463
dt_score 2478340.4946695096
```

```
rf_score 1498075.2163815699
adb_score 3310945.0891640517
svm_score 7868798.934144937
gboost_score 7868798.934144937
xgboost_score 7868798.934144937
```

> *Cross validation:*

```
from sklearn.model_selection import KFold,cross_val_score
k_f = KFold(n_splits=5,shuffle=True)
k_f


print("Cross validation score for lr
model","=>",cross_val_score(lr,X,y,cv=5).mean())
print("Cross validation score for knn
model","=>",cross_val_score(knn,X,y,cv=5).mean())
print("Cross validation score for dt
model","=>",cross_val_score(dt,X,y,cv=5).mean())
print("Cross validation score for rf
model","=>",cross_val_score(rf,X,y,cv=5).mean())
print("Cross validation score for adb
model","=>",cross_val_score(adb,X,y,cv=5).mean())
print("Cross validation score for svm
model","=>",cross_val_score(svm,X,y,cv=5).mean())
print("Cross validation score for gboost
model","=>",cross_val_score(gboost,X,y,cv=5).mean())
print("Cross validation score for xgboost
model","=>",cross_val_score(xgboost,X,y,cv=5).mean())
```

```
Cross validation score for lr model => 0.16195505955216938
Cross validation score for knn model => -0.12016245152150842
Cross validation score for dt model => 0.09399392580193404
Cross validation score for rf model => 0.4023648404733654
Cross validation score for adb model => 0.3289283108437268
Cross validation score for svm model => -0.12340830350876501
Cross validation score for gboost model => 0.42984343393611
Cross validation score for xgboost model =>
0.3931434961697732
```

As per above cross validation and evaluate metrics decision tree is good predictor checking the hyperparameter tuning to check the chances the accuracy.

> *HyperParameter Tuning :*

```
from sklearn.model_selection import GridSearchCV


dt.get_params().keys()  # to check the parameters


param_grid = {'criterion':['squared_error', 'friedman_mse',
'absolute_error','poisson'],
              'splitter' : ['best','random'],'max_depth':
[10,20,30,40,50],
              'min_samples_split' :
[2,3,4,5],'min_samples_leaf': [2,4,5],
              'max_features':
['auto','sqrt','log2'],'max_leaf_nodes' : [10,20,30,40]}
param_grid


{'criterion': ['squared_error', 'friedman_mse',
'absolute_error', 'poisson'],
 'splitter': ['best', 'random'],
 'max_depth': [10, 20, 30, 40, 50],
 'min_samples_split': [2, 3, 4, 5],
 'min_samples_leaf': [2, 4, 5],
 'max_features': ['auto', 'sqrt', 'log2'],
 'max_leaf_nodes': [10, 20, 30, 40]}




gridsearch = GridSearchCV(dt,param_grid =
param_grid,cv=5,verbose = 2 ,n_jobs =5)


gridsearch.fit(X_train,y_train)


Fitting 5 folds for each of 5760 candidates, totalling 28800
fits
```

Out[124]:

```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
n_jobs=5,
             param_grid={'criterion': ['squared_error',
'friedman_mse',
                                       'absolute_error',
'poisson'],
                         'max_depth': [10, 20, 30, 40, 50],
                         'max_features': ['auto', 'sqrt',
'log2'],
                         'max_leaf_nodes': [10, 20, 30, 40],
                         'min_samples_leaf': [2, 4, 5],
```

```
                          'min_samples_split': [2, 3, 4, 5],
                          'splitter': ['best', 'random']},
             verbose=2)
```

```
gridsearch.best_params_
```

```
{'criterion': 'friedman_mse',
 'max_depth': 50,
 'max_features': 'auto',
 'max_leaf_nodes': 40,
 'min_samples_leaf': 4,
 'min_samples_split': 3,
 'splitter': 'random'}
```

```
dt_hpy_tuning=DecisionTreeRegressor(criterion=
'friedman_mse',
 max_depth= 50,
 max_features='auto',
 max_leaf_nodes= 40,
 min_samples_leaf= 4,
 min_samples_split= 3,
 splitter= 'random')
```

```
dt_hpy_tuning.fit(X_train, y_train)
```

```
DecisionTreeRegressor(criterion='friedman_mse',
max_depth=50,
                      max_features='auto',
max_leaf_nodes=40,
                      min_samples_leaf=4,
min_samples_split=3,
                      splitter='random')
```

```
dt_hpy_tuning.score(X_train, y_train)
```

```
y_pred_dt = dt_hpy_tuning.predict(X_test)
```

```
mean_squared_error(y_test,y_pred_dt)
```

```
2378954.8716015876
```

As observed i am getting almost same accuarcy with or without hyperparameter tuing.so saving model both models without hypermeter tuning.

> *Model saving*

```
import pickle

Flight_price = 'Flight_price.pickle'

pickle.dump(dt,open(Flight_price,'wb'))
```

> *Concluding Remarks*

Data analysis is a proven way for any organizations and enterprises to gain the information they need to make better decisions, serve their customers, and increase productivity and revenue. The benefits of data analysis are almost too numerous to count, and some of the most rewarding benefits include getting the right information for business, getting more value out of it. And it creates more effective marketing campaigns, gaining a better understanding of customers, and so on.To summarize, for this project first we have collected data. Since the APIs by Indian companies like yatra.com returned data in a complex format resulting in a lot of time to clean the data before analyzing, therefore we decided to build a web spider that extracts the required values from a website and stores it as a CSV file. In addition, The data was further processed based on the parameters like departure time , arrival time , duration and all. Furthermore, after visualization to get better insights from data points. There were few columns contains categorical values which converted into numeric using pandas get_dummies function. And used various advanced model to build ML project. And as per cross validation we have found that decision tree model giving the good accuracy in order to predict out flight price which has been saved as base model using pickle library.