FLIP ROBO

NAME OF THE PROJECT

FLIGHT PRICE PREDICTION

SUBMITTED BY

TANUJA PATIL

ABSTRACT:

Someone who purchases flight tickets frequently would be able to predict the right time to procure a ticket to obtain the best deal. Many airlines change ticket prices for their revenue management. The airline may increase the prices when the demand is to be expected to increase the capacity. To estimate the minimum airfare, data for a specific air route has been collected including the features like departure time, arrival time and airways over a specific period. Features are extracted from the collected data to apply Machine Learning (ML) models.

INTRODUCTION

The flight ticket buying system is to purchase a ticket many days prior to flight takeoff so as to stay away from the effect of the most extreme charge. Mostly, aviation routes don’t agree this procedure. Plane organizations may diminish the cost at the time, they need to build the market and at the time when the tickets are less accessible. They may maximize the costs. So the cost may rely upon different factors. To foresee the costs this venture uses AI to exhibit the ways of flight tickets after some time. All organizations have the privilege and opportunity to change it’s ticket costs at anytime. Explorer can set aside cash by booking a ticket at the least costs. People who had travelled by flight frequently are aware of price fluctuations. The airlines use complex policies of Revenue Management for execution of distinctive evaluating systems. The evaluating system as a result changes the charge depending on time, season, and festive days to change the header or footer on successive pages. The ultimate aim of the airways is to earn profit whereas the customer searches for the minimum rate. Customers usually try to buy the ticket well in advance of departure date so as to avoid hike in airfare as date comes closer. But actually this is not the fact. The customer may wind up by giving more than they ought to for the same seat.

LITERATURE SURVEY

It is hard for the client to buy an air ticket at the most reduced cost. For this few procedures are explored to determine time and date to grab air tickets with minimum fare rate. The majority of these systems are utilizing the modern computerized system known as Machine Learning. Data is collected from yatra.com for different dates. Two features such as number of days for departure and whether departure is on weekend or weekday are considered to develop the model. The model guesses airfare well in advance from the departure date. But the model isn't convincing in a situation for an extensive time allotment, it closes the departure date. Wohlfarth proposed a ticket purchasing time improvement model subject to a significant pre-processing known as macked point processors, data mining frameworks (course of action and grouping) and quantifiable examination system. This framework is proposed to change various added value arrangements into included added value arrangement heading which can support to solo gathering estimation. This value heading is packed into get together reliant on near evaluating conduct. Headway model measure the value change plans. A tree-based analysis used to pick the best planning gathering and a short time later looking at the progression model. The model provides the most acceptable number of days before buying the flight ticket. The model considers two types of a variable such as the entry and is date of obtainment.

DATA COLLECTION

The accumulation of information is the most significant part of this venture. The different wellsprings of the information on various sites are utilized to prepare the models. Sites provide data about the numerous courses, times, aircrafts and charge. Different sources from API's to customer travel sites are accessible for information scratching. In this segment information of the different sources and parameters that are gathered are talked about. To verify this, information is collected from ―yatra.com site and the models are implemented using python .

Data Collection

The python-script take out the data from the site, and providesoutput as a CSV record. The document contains the data with features and its details A significant perspective is to choose the features required for calculation of expected flight price. Output gathered from the site contains number of parameters for each flight: yet not all are required, so just the accompanying components are,

• Date of journey

• Time of Departure

• Place of Departure

• Time of Arrival

• Place of Destination/Arrival

• Airway company

• Total Fare

• Stops

```
1 pd.read_csv("Flight_price_csv.csv")
```

	Unnamed: 0	Airline	Date	Source	Destination	Stops	Arrival_Time	Dep_Time	Duration	Add_info	Price
0	0	Air India	Tue, 23 Nov 2021	Bangalore	New Delhi	Non Stop	20:20	17:20	3h 00m	Free Meal	3,234
1	1	Air India	Tue, 23 Nov 2021	Bangalore	New Delhi	Non Stop	08:45	05:45	3h 00m	Free Meal	3,546
2	2	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	23:45	18:45	5h 00m	No Meal Fare	3,546
3	3	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	18:10	13:05	5h 05m	No Meal Fare	3,546
4	4	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	20:25	15:20	5h 05m	No Meal Fare	3,546
...	...	...	...	...	...	...	...	...	...	...	...
2192	2192	Vistara	Sat, 6 Nov 2021	Chennai	Pune	2 Stop(s)	18:35	07:00	11h 35m	No Meal Fare	20,163
2193	2193	Vistara	Sat, 6 Nov 2021	Chennai	Pune	2 Stop(s)	18:35	07:00	11h 35m	No Meal Fare	20,163
2194	2194	Air India	Sat, 6 Nov 2021	Chennai	Pune	2 Stop(s)	18:10	08:30	9h 40m	No Meal Fare	21,528
2195	2195	Air India	Sat, 6 Nov 2021	Chennai	Pune	2 Stop(s)	18:10	06:20	11h 50m	No Meal Fare	22,788
2196	2196	Air India	Sat, 6 Nov 2021	Chennai	Pune	2 Stop(s)	18:10	05:55	12h 15m	Free Meal	37,531

2197 rows × 11 columns

Above fig shows collected data.

DATA ANALYSIS:

Data analysis involves manipulating, transforming, and visualizing data in order to infer meaningful insights from the results. Individuals, businesses, and even governments often take direction based on these insights. In Data analysis, we have check data types, missing values, and many more things. so let’s do it one by one.

First import all necessary libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split,cross_val_score
```

Load the data set:

```
df=pd.read_csv("Flight_price_csv.csv")
df.head(20)
```

	Unnamed: 0	Airline	Date	Source	Destination	Stops	Arrival_Time	Dep_Time	Duration	Add_info	Price
0	0	Air India	Tue, 23 Nov 2021	Bangalore	New Delhi	Non Stop	20:20	17:20	3h 00m	Free Meal	3,234
1	1	Air India	Tue, 23 Nov 2021	Bangalore	New Delhi	Non Stop	08:45	05:45	3h 00m	Free Meal	3,546
2	2	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	23:45	18:45	5h 00m	No Meal Fare	3,546
3	3	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	18:10	13:05	5h 05m	No Meal Fare	3,546
4	4	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	20:25	15:20	5h 05m	No Meal Fare	3,546
5	5	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	00:45\n+ 1 day	18:45	6h 00m	No Meal Fare	3,546
6	6	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	22:25	16:20	6h 05m	No Meal Fare	3,546
7	7	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	19:55	13:05	6h 50m	No Meal Fare	3,546
8	8	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	01:40\n+ 1 day	18:40	7h 00m	No Meal Fare	3,546
9	9	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	18:30	11:20	7h 10m	No Meal Fare	3,546
10	10	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	22:35	15:20	7h 15m	No Meal Fare	3,546
11	11	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	23:45	16:20	7h 25m	No Meal Fare	3,546
12	12	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	07:15\n+ 1 day	23:45	7h 30m	No Meal Fare	3,546
13	13	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	17:35	09:50	7h 45m	No Meal Fare	3,546
14	14	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	18:10	10:15	7h 55m	No Meal Fare	3,546
15	15	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	00:45\n+ 1 day	16:20	8h 25m	No Meal Fare	3,546
16	16	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	18:10	09:20	8h 50m	No Meal Fare	3,546
17	17	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	16:10	06:55	9h 15m	No Meal Fare	3,546
18	18	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	22:25	13:05	9h 20m	No Meal Fare	3,546
19	19	IndiGo	Tue, 23 Nov 2021	Bangalore	New Delhi	1 Stop	07:15\n+ 1 day	21:55	9h 20m	No Meal Fare	3,546

pvnb#

Shape of the data set:

```
df.shape
```

```
(2197, 11)
```

Data set have 2197 rows and 11 columns.

Checking data types of the data :

Dataset have all values of object type. Later we have to change data type of some values such as price must be int type instead of object type.

```
df.dtypes
```

```
Unnamed: 0       int64
Airline         object
Date            object
Source          object
Destination     object
Stops           object
Arrival_Time    object
Dep_Time        object
Duration        object
Add_info        object
Price           object
dtype: object
```

Info about data set:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2197 entries, 0 to 2196
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Unnamed: 0    2197 non-null   int64
 1   Airline       2197 non-null   object
 2   Date          2197 non-null   object
 3   Source        2197 non-null   object
 4   Destination   2197 non-null   object
 5   Stops         2197 non-null   object
 6   Arrival_Time  2197 non-null   object
 7   Dep_Time      2197 non-null   object
 8   Duration      2197 non-null   object
 9   Add_info      2197 non-null   object
 10  Price         2197 non-null   object
dtypes: int64(1), object(10)
memory usage: 188.9+ KB
```

Dataset have 10 columns and 2197 rows which are of object type.

Checking missing values:

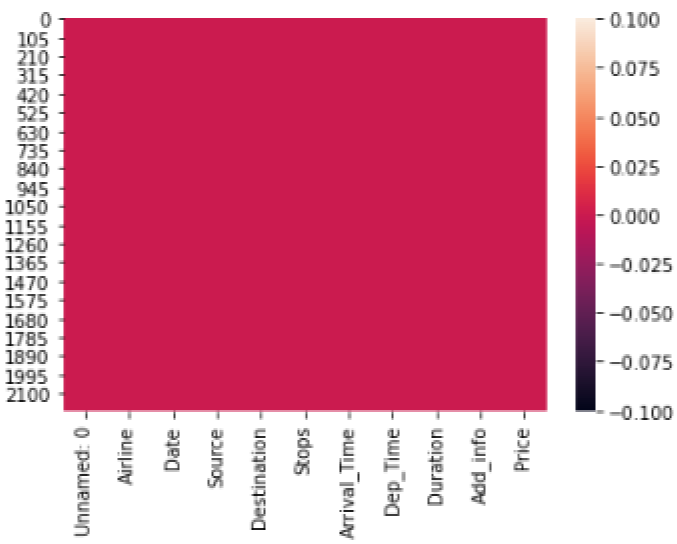
Following fig shows that data set have no missing values.

```
df.isnull().sum()
```

```
Unnamed: 0      0
Airline         0
Date            0
Source          0
Destination     0
Stops           0
Arrival_Time    0
Dep_Time        0
Duration        0
Add_info        0
Price           0
dtype: int64
```

```
sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```



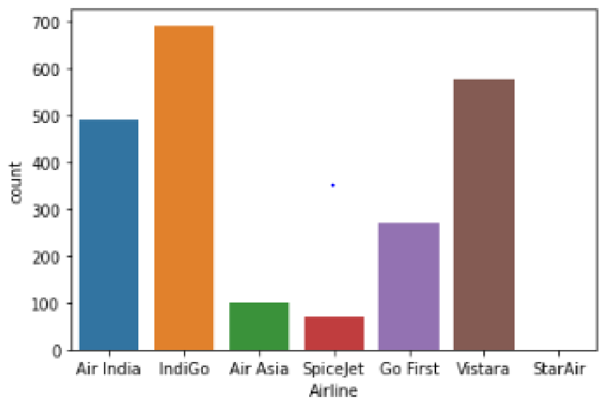
Above heatmap shows that there is no missing values in data set.

Now let's look into each column separately.

```
print(df['Airline'].value_counts())
sns.countplot(df['Airline'])
```

```
IndiGo       692
Vistara      576
Air India    492
Go First     269
Air Asia      98
SpiceJet      69
StarAir        1
Name: Airline, dtype: int64
```

```
<AxesSubplot:xlabel='Airline', ylabel='count'>
```



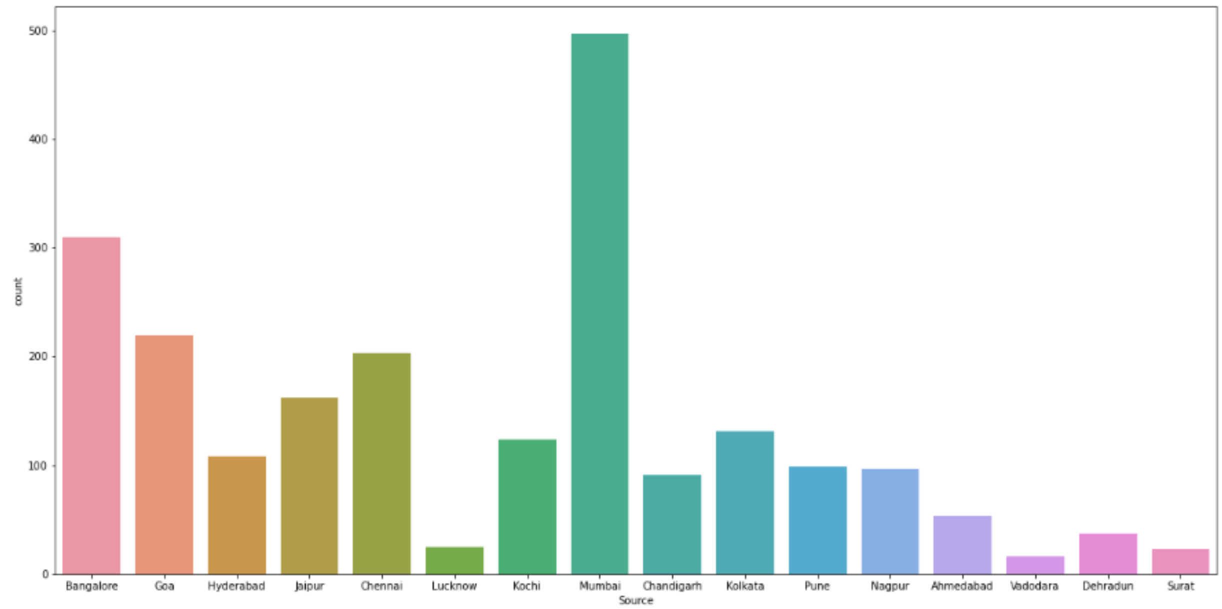
Above fig shows the 7 values for airline column.

```
print(df['Source'].value_counts())
```

```
Mumbai        497
Bangalore     310
Goa           220
Chennai       203
Jaipur        162
Kolkata       131
Kochi         124
Hyderabad     108
Pune           99
Nagpur         97
Chandigarh     91
Ahmedabad      54
Dehradun       37
Lucknow        25
Surat          23
Vadodara       16
Name: Source, dtype: int64
```

```
plt.figure(figsize=(20,10))
sns.countplot(df['Source'])
```

```
<AxesSubplot:xlabel='Source', ylabel='count'>
```

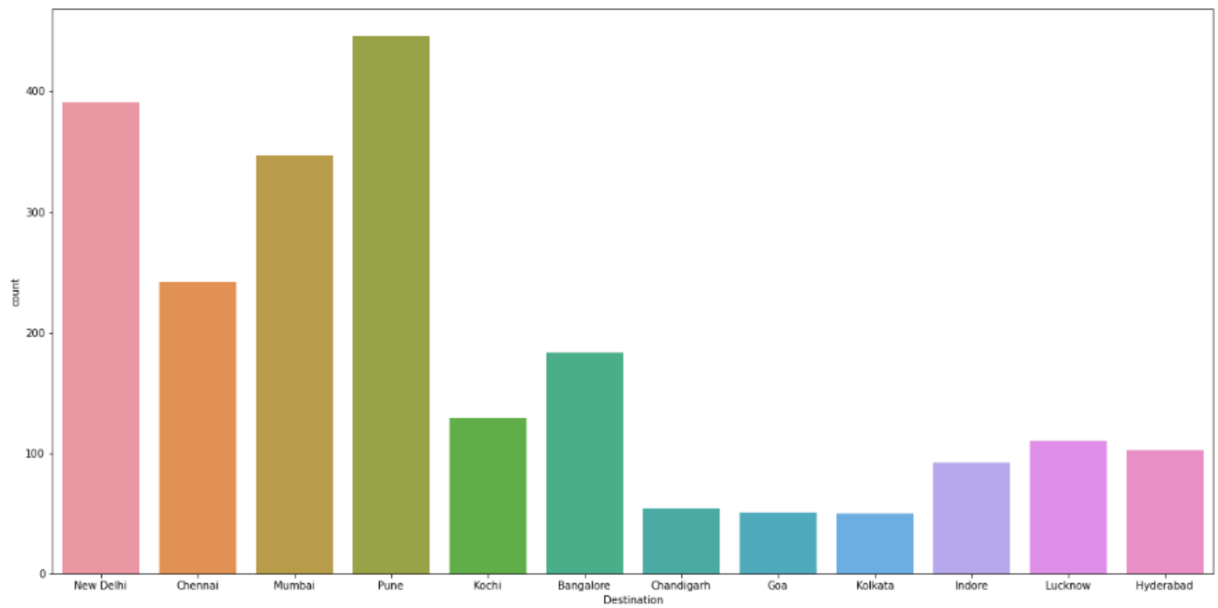


Above figure shows different source values.

```
print(df['Destination'].value_counts())
plt.figure(figsize=(20,10))
sns.countplot(df['Destination'])
```

```
Pune          446
New Delhi     391
Mumbai        347
Chennai       242
Bangalore     183
Kochi         129
Lucknow       110
Hyderabad     102
Indore         92
Chandigarh     54
Goa            51
Kolkata        50
Name: Destination, dtype: int64
```

<AxesSubplot:xlabel='Destination', ylabel='count'>

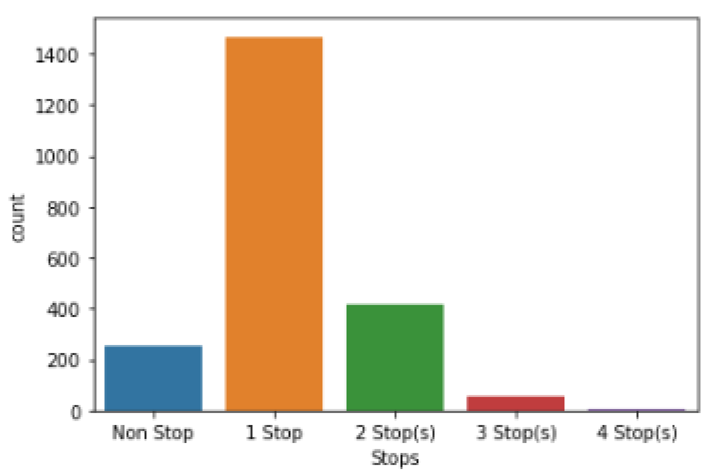


Above fig shows that different destination values.

```
print(df['Stops'].value_counts())
sns.countplot(df['Stops'])
```

```
1 Stop        1467
2 Stop(s)      418
Non Stop       251
3 Stop(s)       57
4 Stop(s)        4
Name: Stops, dtype: int64
```

<AxesSubplot:xlabel='Stops', ylabel='count'>

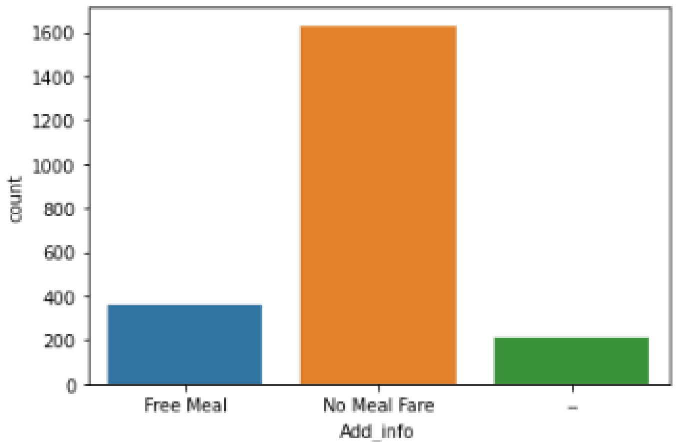


Above fig shows different stop values.

```
print(df['Add_info'].value_counts())
sns.countplot(df['Add_info'])
```

```
No Meal Fare    1630
Free Meal        358
--               209
Name: Add_info, dtype: int64
```

```
<AxesSubplot:xlabel='Add_info', ylabel='count'>
```



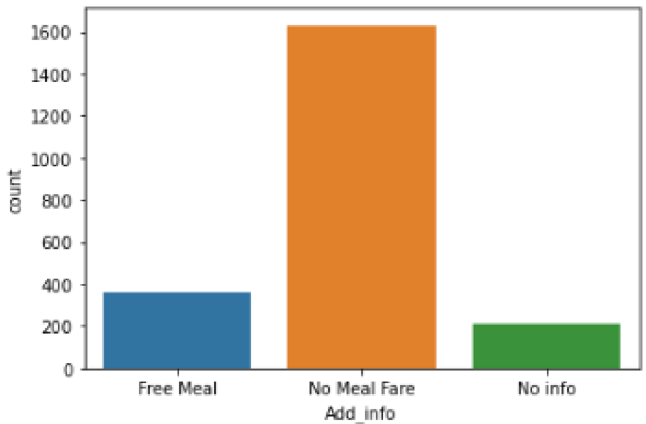
Aove fig shows Add_info column values. Addition info column have "—" values, which we need to handle, so let's change "—" values to "no info" values.

```python
df['Add_info']=df['Add_info'].replace('--','No info')
```

```python
print(df['Add_info'].value_counts())
sns.countplot(df['Add_info'])
```

```
No Meal Fare    1630
Free Meal        358
No info          209
Name: Add_info, dtype: int64
```

```
<AxesSubplot:xlabel='Add_info', ylabel='count'>
```



```python
print(df['Arrival_Time'].unique())
```

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

Above fig shows unique values of arrival time column.

```
print(df['Dep_Time'].unique())
```

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

Above fig shows values for departure time column.

```
print(df['Date'].value_counts())
```

```
Tue, 9 Nov 2021     314
Wed, 3 Nov 2021     234
Tue, 23 Nov 2021    161
Sat, 6 Nov 2021     156
Thu, 11 Nov 2021    146
Tue, 19 Oct 2021    132
Tue, 2 Nov 2021     111
Mon, 15 Nov 2021    106
Sat, 13 Nov 2021    102
Tue, 26 Oct 2021    101
Thu, 28 Oct 2021     92
Thu, 21 Oct 2021     88
Sun, 31 Oct 2021     85
Mon, 8 Nov 2021      79
Fri, 12 Nov 2021     75
Sun, 7 Nov 2021      61
Mon, 1 Nov 2021      52
Wed, 15 Dec 2021     51
Sat, 11 Dec 2021     48
Fri, 10 Dec 2021      2
Mon, 25 Oct 2021      1
Name: Date, dtype: int64
```

Above fig shows date column values.

DATA PREPROCESSING:

1)Stop column:

Stops column have values such as non stops, 1 stop ,two stop,3 stops,4 stops which we need to convert it into numerical data.

```
df['Stops']=df['Stops'].replace("Non Stop",0)
df['Stops']=df['Stops'].replace("1 Stop",1)
df['Stops']=df['Stops'].replace("2 Stop(s)",2)
df['Stops']=df['Stops'].replace("3 Stop(s)",3)
df['Stops']=df['Stops'].replace("4 Stop(s)",4)
```

```
df['Stops']=df['Stops'].astype(int)
df['Stops']
```

```
0       0
1       0
2       1
3       1
4       1
       ..
2192    2
2193    2
2194    2
2195    2
2196    2
Name: Stops, Length: 2197, dtype: int32
```

2)Price column:

In the given data set price column is object type, which we need to convert into numerical form. This is shows in below fig.

```
df['Price']
```

```
0        3,234
1        3,546
2        3,546
3        3,546
4        3,546
         ...
2192    20,163
2193    20,163
2194    21,528
2195    22,788
2196    37,531
Name: Price, Length: 2197, dtype: object
```

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

```
0        3234
1        3546
2        3546
3        3546
4        3546
        ...
2192    20163
2193    20163
2194    21528
2195    22788
2196    37531
Name: Price, Length: 2197, dtype: int32
```

3)Date column:

In the given data set date column is object type, but it should be datetime type, so we have to change date column to datetime type.

```
df['Date']=df['Date'].str.split(",").str.get(1)
```

```
df['Date']
```

```
0        23 Nov 2021
1        23 Nov 2021
2        23 Nov 2021
3        23 Nov 2021
4        23 Nov 2021
            ...
2192      6 Nov 2021
2193      6 Nov 2021
2194      6 Nov 2021
2195      6 Nov 2021
2196      6 Nov 2021
Name: Date, Length: 2197, dtype: object
```

```
df['Date']=pd.to_datetime(df['Date'])
```

```
df['Date']
```

```
0      2021-11-23
1      2021-11-23
2      2021-11-23
3      2021-11-23
4      2021-11-23
          ...
2192   2021-11-06
2193   2021-11-06
2194   2021-11-06
2195   2021-11-06
2196   2021-11-06
Name: Date, Length: 2197, dtype: datetime64[ns]
```

Deriving new features:

From date column we can derive new features such as day,month and year. Following fig shows this:

```python
df["flight_date"] = pd.to_datetime(df["Date"]).dt.day
df["flight_month"] = pd.to_datetime(df["Date"]).dt.month
df['flight_year']=pd.to_datetime(df["Date"]).dt.year
```

```python
df.drop('Date',axis=1,inplace=True)
```

```python
df
```

	Unnamed: 0	Airline	Source	Destination	Stops	Arrival_Time	Dep_Time	Duration	Add_info	Price	flight_date	flight_month	flight_year
0	0	Air India	Bangalore	New Delhi	0	20:20	17:20	3h 00m	Free Meal	3234	23	11	2021
1	1	Air India	Bangalore	New Delhi	0	08:45	05:45	3h 00m	Free Meal	3546	23	11	2021
2	2	IndiGo	Bangalore	New Delhi	1	23:45	18:45	5h 00m	No Meal Fare	3546	23	11	2021
3	3	IndiGo	Bangalore	New Delhi	1	18:10	13:05	5h 05m	No Meal Fare	3546	23	11	2021
4	4	IndiGo	Bangalore	New Delhi	1	20:25	15:20	5h 05m	No Meal Fare	3546	23	11	2021
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2192	2192	Vistara	Chennai	Pune	2	18:35	07:00	11h 35m	No Meal Fare	20163	6	11	2021
2193	2193	Vistara	Chennai	Pune	2	18:35	07:00	11h 35m	No Meal Fare	20163	6	11	2021
2194	2194	Air India	Chennai	Pune	2	18:10	08:30	9h 40m	No Meal Fare	21528	6	11	2021
2195	2195	Air India	Chennai	Pune	2	18:10	06:20	11h 50m	No Meal Fare	22788	6	11	2021
2196	2196	Air India	Chennai	Pune	2	18:10	05:55	12h 15m	Free Meal	37531	6	11	2021

2197 rows × 13 columns

From departure time ,Duration,arrival time we can derive new features such minute and hours.

```python
df["Dep_hour"] = pd.to_datetime(df["Dep_Time"]).dt.hour
df["Dep_min"] = pd.to_datetime(df["Dep_Time"]).dt.minute

```

```python
df['Arrival_Time']=df['Arrival_Time'].str.split("\n").str.get(0)
```

```python
df["arr_hr"] = pd.to_datetime(df["Arrival_Time"]).dt.hour
df["arr_min"] = pd.to_datetime(df["Arrival_Time"]).dt.minute
```

```python
df.head()
```

ine	Source	Destination	Stops	Arrival_Time	Dep_Time	Duration	Add_info	Price	flight_date	flight_month	flight_year	Dep_hour	Dep_min	arr_hr	arr_min
Air dia	Bangalore	New Delhi	0	20:20	17:20	3h 00m	Free Meal	3234	23	11	2021	17	20	20	20
Air dia	Bangalore	New Delhi	0	08:45	05:45	3h 00m	Free Meal	3546	23	11	2021	5	45	8	45
iGo	Bangalore	New Delhi	1	23:45	18:45	5h 00m	No Meal Fare	3546	23	11	2021	18	45	23	45
iGo	Bangalore	New Delhi	1	18:10	13:05	5h 05m	No Meal Fare	3546	23	11	2021	13	5	18	10
iGo	Bangalore	New Delhi	1	20:25	15:20	5h 05m	No Meal Fare	3546	23	11	2021	15	20	20	25

```
df['Dur_hrs']=df['Duration'].str.split('h').str.get(0)
df['Dur_min']=df['Duration'].str.split('h').str.get(1)
```

```
df['Dur_min']=df['Dur_min'].str.replace('m','')
```

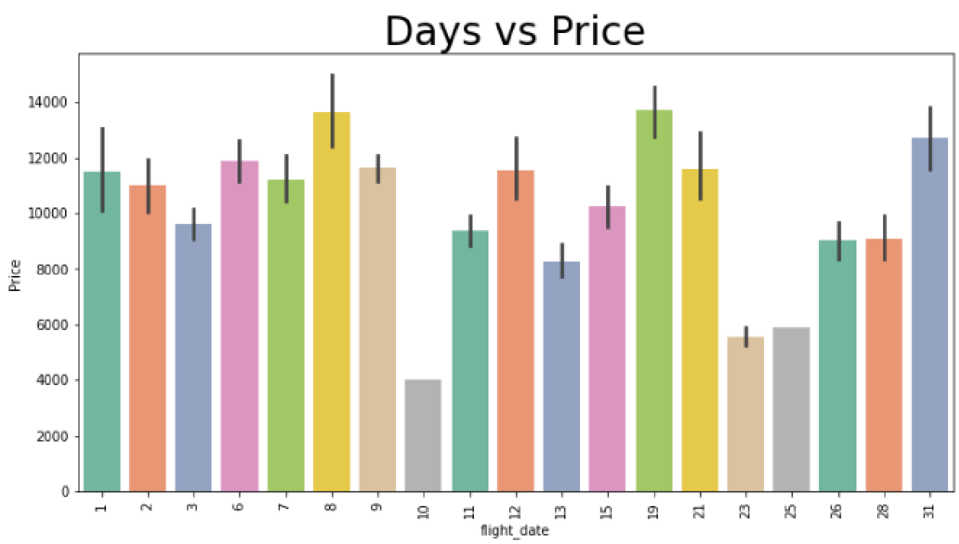
```
df.head(10)
```

Airline	Source	Destination	Stops	Duration	Add_info	Price	flight_date	flight_month	flight_year	Dep_hour	Dep_min	arr_hr	arr_min	Dur_hrs	Dur_min
Air India	Bangalore	New Delhi	0	3h 00m	Free Meal	3234	23	11	2021	17	20	20	20	3	00
Air India	Bangalore	New Delhi	0	3h 00m	Free Meal	3546	23	11	2021	5	45	8	45	3	00
IndiGo	Bangalore	New Delhi	1	5h 00m	No Meal Fare	3546	23	11	2021	18	45	23	45	5	00
IndiGo	Bangalore	New Delhi	1	5h 05m	No Meal Fare	3546	23	11	2021	13	5	18	10	5	05
IndiGo	Bangalore	New Delhi	1	5h 05m	No Meal Fare	3546	23	11	2021	15	20	20	25	5	05
IndiGo	Bangalore	New Delhi	1	6h 00m	No Meal Fare	3546	23	11	2021	18	45	0	45	6	00
IndiGo	Bangalore	New Delhi	1	6h 05m	No Meal Fare	3546	23	11	2021	16	20	22	25	6	05
IndiGo	Bangalore	New Delhi	1	6h 50m	No Meal Fare	3546	23	11	2021	13	5	19	55	6	50
IndiGo	Bangalore	New Delhi	1	7h 00m	No Meal Fare	3546	23	11	2021	18	40	1	40	7	00
IndiGo	Bangalore	New Delhi	1	7h 10m	No Meal Fare	3546	23	11	2021	11	20	18	30	7	10

VISUALIZATION:

Let’s check on which days prices are high.

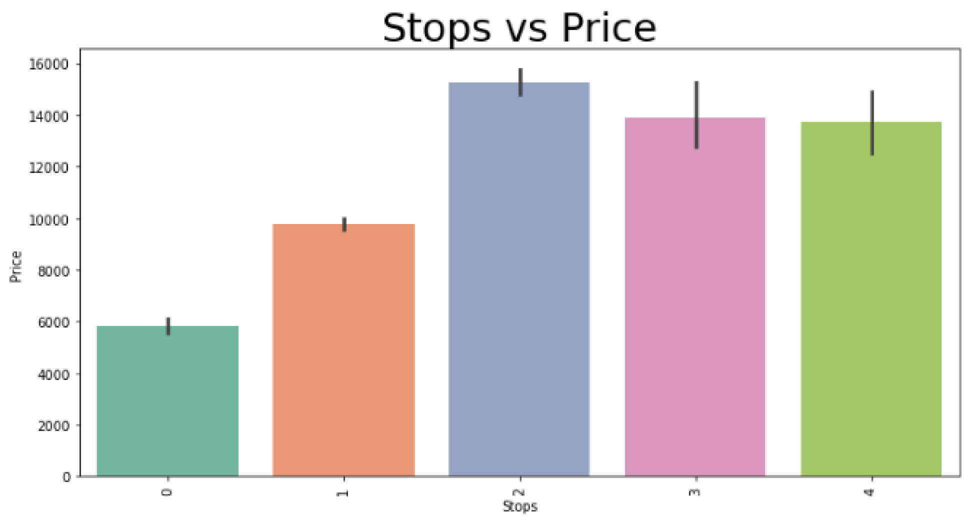
```
plt.figure(figsize=(12,6))
sns.barplot(df['flight_date'], df['Price'], palette='Set2')
plt.title('Days vs Price', size=30)
plt.xticks(rotation=90)
plt.show()

```



Above fig shows that on some days prices are very high compred to others.

2)Now let’s check stops vs price:

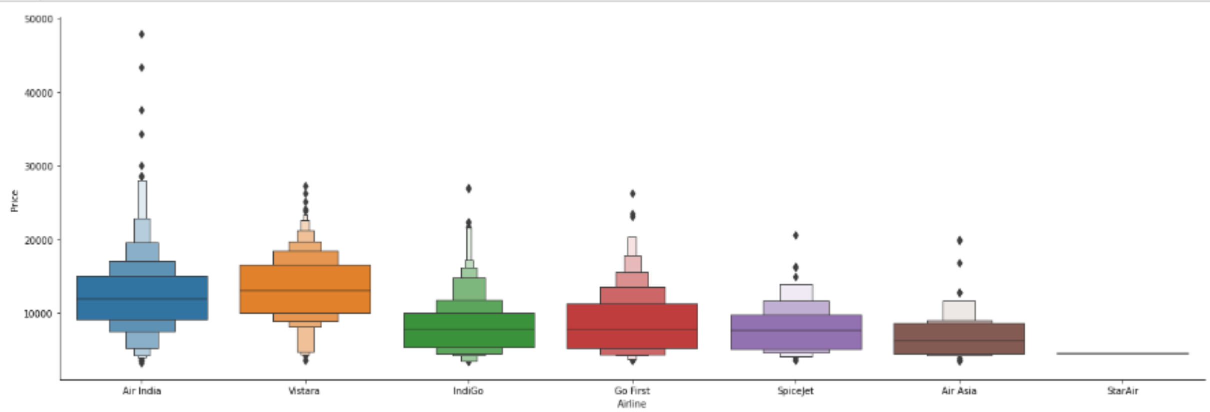
```python
plt.figure(figsize=(12,6))
sns.barplot(df['Stops'], df['Price'], palette='Set2')
plt.title('Stops vs Price', size=30)
plt.xticks(rotation=90)
plt.show()
```



Above fig shows that the prices are high for those flights which have 2, 3,4 stops .

3)let’s check airlines vs prics:

```python
sns.catplot(y = "Price", x = "Airline", data = df.sort_values("Price", ascending = False), kind="boxen", height = 6, aspect
plt.show()
```



From above fig we can say that air india have high price.

ENCODING:

First we have to encode the categorical data into numerical data. There are different techniques of encoding:

- One Hot Encoder: Encode categorical integer features using a one hot one-of-K scheme. The input to this transformer should be a matrix of integers, denoting the values taken on by categorical (discrete) features. The output will be a sparse matrix where each column corresponds to one possible value of one feature.
- Label Encoder: Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important preprocessing step for the structured dataset in supervised learning
- OrdinalEncoder: In ordinal encoding, each unique category value is assigned an integer value. For example, “red” is 1, “green” is 2, and “blue” is 3. This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.

In this project categorical values are encoded using get_dummies method of one hot encoder.

```
dummies=pd.get_dummies(df[['Airline','Source','Destination']])
```

```
dummies
```

	Airline_Air Asia	Airline_Air India	Airline_Go First	Airline_IndiGo	Airline_SpiceJet	Airline_StarAir	Airline_Vistara	Source_Ahmedabad	Source_Bangalore	Source_Chandi
0	0	1	0	0	0	0	0	0	1	
1	0	1	0	0	0	0	0	0	1	
2	0	0	0	1	0	0	0	0	1	
3	0	0	0	1	0	0	0	0	1	
4	0	0	0	1	0	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	
2192	0	0	0	0	0	0	1	0	0	
2193	0	0	0	0	0	0	1	0	0	
2194	0	1	0	0	0	0	0	0	0	
2195	0	1	0	0	0	0	0	0	0	
2196	0	1	0	0	0	0	0	0	0	

2197 rows × 35 columns

Now concat encoded data with dataframe:

```
df1=pd.concat([dummies, df._get_numeric_data()], axis=1)
```

```
x=df1.drop(['Price','flight_year'],axis=1)
y=df1['Price']
```

```
df1
```

	Airline_Air Asia	Airline_Air India	Airline_Go First	Airline_IndiGo	Airline_SpiceJet	Airline_StarAir	Airline_Vistara	Source_Ahmedabad	Source_Bangalore	Source_Chandi
0	0	1	0	0	0	0	0	0	1	
1	0	1	0	0	0	0	0	0	1	
2	0	0	0	1	0	0	0	0	1	
3	0	0	0	1	0	0	0	0	1	
4	0	0	0	1	0	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	
2192	0	0	0	0	0	0	1	0	0	
2193	0	0	0	0	0	0	1	0	0	
2194	0	1	0	0	0	0	0	0	0	
2195	0	1	0	0	0	0	0	0	0	
2196	0	1	0	0	0	0	0	0	0	

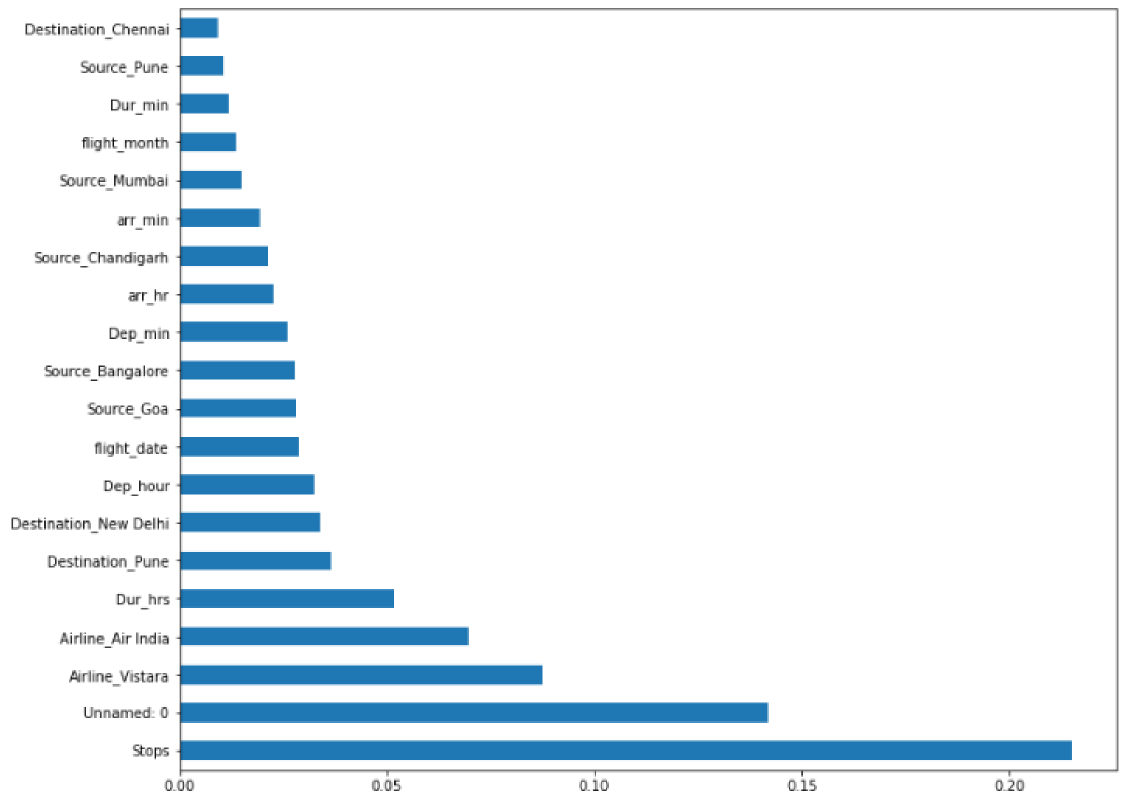
2197 rows × 47 columns

Feature importance:

```
from sklearn.ensemble import ExtraTreesRegressor
s1=ExtraTreesRegressor()
s1.fit(x,y)
```

```
ExtraTreesRegressor()
```

```
plt.figure(figsize=(12,10))
fea_imp=pd.Series(s1.feature_importances_,index=x.columns)
fea_imp.nlargest(20).plot(kind='barh')
plt.show()
```



Above fig shows bar graph which gives better understanding of feature importance. Stops is feature which is most important.

Split data for training and testing:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=462)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1647, 44)
(550, 44)
(1647,)
(550,)
```

Above fig shows that data is splitted using train_test_split for training and testing. Now create instance of modules. As this is regression type problem so we have to import regression algorithms.

```
#create instace of regression algorithm
lr=LinearRegression()
dtr=DecisionTreeRegressor()

Knn=KNeighborsRegressor()
svr=SVR()
l1=Lasso(alpha=0.001)
r1=Ridge(alpha=0.001)
```

Fit and predict:

Now fit data into model and predict the output.

```
#fit data  and predict
lst1=[lr,dtr,svr,l1,r1]
for i in lst1:
    i.fit(x_train,y_train)
    pred=i.predict(x_test)
    print("accuracy_scores",i)
    print("r2_score",r2_score(y_test,pred))
    print("mean_squared_error",mean_squared_error(y_test,pred))
    print("mean_absolute_error",mean_absolute_error(y_test,pred))

```

```
accuracy_scores LinearRegression()
r2_score 0.5530955403183299
mean_squared_error 10680431.023366034
mean_absolute_error 2329.1688352272727
accuracy_scores DecisionTreeRegressor()
r2_score 0.5564671709164789
mean_squared_error 10599853.46979798
mean_absolute_error 1616.4630303030303
accuracy_scores SVR()
r2_score -0.020007812321015672
mean_squared_error 24376850.234497488
mean_absolute_error 3807.7812882159615
accuracy_scores Lasso(alpha=0.001)
r2_score 0.5531070686278259
mean_squared_error 10680155.511874212
mean_absolute_error 2329.0822844724876
accuracy_scores Ridge(alpha=0.001)
r2_score 0.553106625398069
mean_squared_error 10680166.104484655
mean_absolute_error 2329.080582448983
```

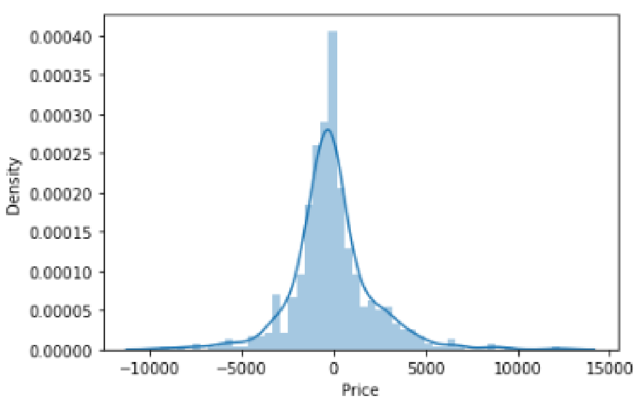
```
r1=RandomForestRegressor()
r1.fit(x_train,y_train)
pred1=r1.predict(x_test)
```

```
r1.score(x_test,y_test),r1.score(x_train,y_train),r2_score(y_test,pred1)
```

```
(0.7956736752466541, 0.9536544157622294, 0.7956736752466541)
```

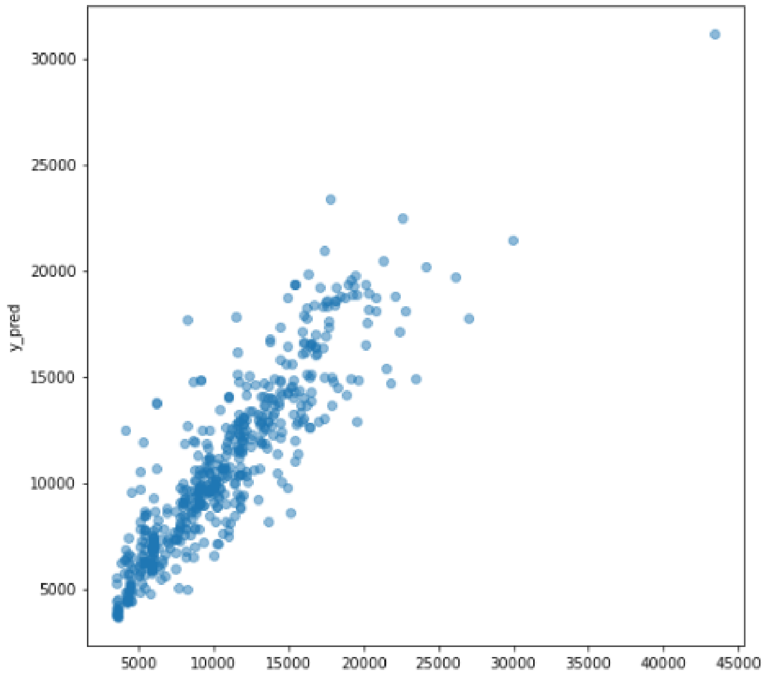
```
sns.distplot(y_test-pred1)
```

```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



By observing metrics we can say that r2_score of random Forest Regressor is high.

```
plt.figure(figsize = (8,8))
plt.scatter(y_test, pred1, alpha = 0.5)
plt.xlabel("y_t")
plt.ylabel("y_pred")
plt.show()
```



predicted price and actual price showed in scatterplot.

HyperParameterTunning:

```
from sklearn.model_selection import GridSearchCV
param_grid = {
            "n_estimators"      : [10,20,30,50,60],
            "max_features"      : ["auto", "sqrt", "log2"],
            "min_samples_split" : [2,4,8,10,12],
            "bootstrap": [True, False],
            }

grid = GridSearchCV(estimator=rf, param_grid=param_grid, n_jobs=-1, cv=7)

grid.fit(x_train,y_train)
```

```
GridSearchCV(cv=7, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'bootstrap': [True, False],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [2, 4, 8, 10, 12],
                         'n_estimators': [10, 20, 30, 50, 60]})
```

```
grid.best_params_
```

```
{'bootstrap': False,
 'max_features': 'sqrt',
 'min_samples_split': 2,
 'n_estimators': 50}
```

We can tune different parameters of model so we can improve model's score.

```
from sklearn.ensemble import RandomForestRegressor
rf1=RandomForestRegressor(bootstrap=False,max_features= 'sqrt',min_samples_split=2,n_estimators=50)
rf1.fit(x_train,y_train)
rpred=rf1.predict(x_test)
cv3=cross_val_score(rf1,x_train,y_train,cv=5)
print("score",cv3)

print('mean squared error',mean_squared_error(rpred,y_test))
print("r2_score",r2_score(y_test,rpred))
```

```
score [0.68798924 0.60568683 0.64765182 0.66727542 0.66488287]
mean squared error 5005112.139319637
r2_score 0.7905696004800506
```

After applying parameters which are derived by tunning, r2_score is not increased, so we have to select model which gave high score. So by observing scores of RandomForestRegressor before tuning, we get high score, so that model is best.

Now let's create object file.

```
import joblib
```

```
joblib.dump(r1,"Flight_price.obj")
```

```
['Flight_price.obj']
```

```
f1=joblib.load("Flight_price.obj")
```

```
f1.predict(x_test)
```

```
array([12633.34      ,  8848.32      ,  5183.98      , 12018.54      ,
       18616.85      , 12879.45      , 19698.63      , 19815.925     ,
       16912.92      , 12861.42      , 18320.66380952, 13094.53      ,
        8976.59      , 14928.91      , 11745.43      ,  8671.45      ,
       11949.355     , 17822.13      ,  8855.55      ,  4901.9       ,
        4024.28      ,  8147.62      , 18878.54333333, 11823.34      ,
        9935.31      , 12296.75      ,  4446.91      ,  6572.85      ,
       14581.14      ,  6267.09      , 11417.39      ,  5770.08      ,
        6378.91333333, 16169.135     , 12565.605     , 11266.57      ,
        9029.02      ,  6395.25      , 13683.85      , 12422.25      ,
       16449.13      , 10659.57      ,  8432.13      , 13872.41      ,
        8918.87      , 12678.68      ,  9715.28      , 11058.96      ,
        6067.85      ,  3721.14      ,  5699.88      , 10354.54      ,
        9212.59      , 15393.58      ,  5101.53      ,  8895.4       ,
        4471.16      , 10118.57      ,  6972.43      , 11992.84      ,
       12885.09      , 12788.00833333, 11565.75      , 14202.63      ,
        9411.69      ,  8979.89      ,  8807.28      ,  7199.71      ,
       13251.56      , 12542.44      ,  9691.        , 12836.98      ,
       13984.0175    ,  6882.82      ,  9210.49      ,  4480.64      ,
        8679.01      , 11624.84      ,  8983.75      , 13828.93      ,
```

CONCLUSION

To evaluate the conventional algorithm, a dataset is built for different routes and studied a trend of price variation for the period of limited days. Machine Learning algorithms are applied on the dataset to predict the dynamic fare of flights. This gives the predicted values of flight fare to get a flight ticket at minimum cost. Data is collected from the websites which sell the flight tickets so only limited information can be accessed. The values of R-squared obtained from the algorithm give the accuracy of the model. In the future, if more data could be accessed such as the current availability of seats, the predicted results will be more accurate.