"A PROJECT REPORT ON FLIGHT TICKET PRICE PREDICTION"

SUBMITTED BY

HIMAJA IJJADA

ACKNOWLEDGMENT

I express my sincere gratitude to FlipRobo Technologies for giving me the opportunity to work on “**A PROJECT REPORT ON FLIGHT TICKET PRICE PREDICTION”** using machine learning algorithms. I would also like to thank FlipRobo Technologies for providing me with the requisite knowledge to webscrape the datasets to work with. And I would like to express my gratitude to Mr. Mohd Kashif (SME FlipRobo) and Ms. Sapna Verma (SME FlipRobo) for being of a great help in completion of the project.

Most of the concepts used to predict the Prices of flight tickets project are learned from Data Trained Institute and below documentations.

- https://scikit-learn.org/stable/
- https://seaborn.pydata.org/
- https://www.scipy.org/

CONTENTS

Introduction

Business Problem Framing

Airline industry is one of the most sophisticated in its use of dynamic pricing strategies to maximize revenue, based on proprietary algorithms and hidden variables. That is why the airline companies use complex algorithms to calculate the flight ticket prices. There are several different factors on which the price of the flight ticket depends. The seller has information about all the factors, but buyers are able to access limited information only which is not enough to predict the airfare prices. Considering the features such as departure time, arrival time and time of the day it will give the best time to buy the ticket. Nowadays, the number of people using flights has increased significantly. It is difficult for airlines to maintain prices since prices change dynamically due to different conditions. That's why we will try to use machine learning models to solve this problem. This can help airlines by predicting what prices they can maintain. It can also help customers to predict future flight prices and plan their journey accordingly.

Business goal: The main aim of this project is to predict the price of flight tickets based on various features. The purpose of the paper is to study the factors which influence the fluctuations in the airfare prices and how they are related to the change in the prices. Then using this information, build a system that can help buyers whether to buy a ticket or not. So, we will deploy Machine Learning model for flight ticket price prediction and analysis. This model will provide the approximate selling price for the flight tickets based on different features.

Conceptual Background of the Domain Problem

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, and it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable. 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 2 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).

Here we are trying to help the buyers to understand the price of the flight tickets by deploying machine learning models. These models would help the sellers/buyers to understand the flight ticket prices in market and accordingly they would be able to book their tickets.

Review of Literature

Literature review covers relevant literature with the aim of gaining insight into the factors that are important to predict the flight ticket prices in the market. In this study, we discuss various applications and methods which inspired us to build our supervised ML techniques to predict the price of flight tickets in different locations. We did a background survey regarding the basic ideas of our project and used those ideas for the collection of data information by doing web scraping from www.yatra.com website which is a web platform where buyers can book their flight tickets. This project is more about data exploration, feature engineering and preprocessing that can be done on this data.

Since we scrape huge amount of data that includes more flight related features, we can do better data exploration and derive some interesting features using the available columns. Different techniques like ensemble techniques, and decision trees have been used to make the predictions. The goal of this project is to build an application which can predict the price of flight tickets with the help of other features. In the long term, this would allow people to better explain and reviewing their purchase in this increasing digital world.

Motivation for the Problem Undertaken

Air travel is the fastest mode of transport around, and can cut hours or days off of a trip. But we know how unexpectedly the prices vary due to the **Dynamic pricing**. So, I was interested in Flight Fares Prediction listings to help individuals and find the right fares based on their needs. And also, to get hands on experience and to know that how the data scientist approaches and work in an industry end to end.

Analytical problem framing

Mathematical/ Analytical Modeling of the Problem

We need to develop an efficient and effective Machine Learning model which predicts the price of flight tickets. So, “Price” is our target variable which is continuous in nature. Clearly it is a Regression problem where we need to use regression algorithms to predict the results. This project is done on three phases:

• Data Collection Phase: I have done web scraping to collect the data of flights from the well-known website www.yatra.com where I found more features of flights compared to other websites and I fetch data for different locations. As per the requirement we need to build the model to predict the prices of flight tickets.

• Data Analysis: After cleaning the data I have done some analysis on the data by using different types of visualizations.

• Model Building Phase: After collecting the data, I built a machine learning model. Before model building, have done all data pre-processing steps. The complete life cycle of data science that I have used in this project are as follows:

- Data Cleaning
- Exploratory Data Analysis
- Data Pre-processing
- Model Building
- Model Evaluation
- Selecting the best model

Data Sources and their formats

We have collected the dataset from the website www.yatra.com which is a web platform where one can book their flight tickets. The data is scrapped using Web scraping technique and the framework used is Selenium. We scrapped approximately 5300 of the data rows and fetched the data for flights between different locations and collected the additional information of different flights and saved the collected data in excel format. The dimension of the dataset is 5303 rows and 9 columns including target variable “Price”.

```
# Reading excel file
df = pd.read_excel("Flight_Prices.xlsx")
df
```

	Airline	Departure_time	Time_of_arrival	Duration	Source	Destination	Meal_availability	Number_of_stops	Price
0	Go First	06:00	10:40	4h 40m	New Delhi	Mumbai	No Meal Fare	1 Stop	4,941
1	Go First	16:35	21:25	4h 50m	New Delhi	Mumbai	No Meal Fare	1 Stop	4,941
2	Go First	09:00	15:45	6h 45m	New Delhi	Mumbai	No Meal Fare	1 Stop	4,941
3	Go First	09:10	16:15	7h 05m	New Delhi	Mumbai	No Meal Fare	1 Stop	4,941
4	Go First	05:25	14:05	8h 40m	New Delhi	Mumbai	No Meal Fare	1 Stop	4,941
...	...	...	...	...	...	...	...	...	...
5298	Air India	14:00	19:40	29h 40m	Jaipur	Lucknow	No Meal Fare	2 Stop(s)	15,614
5299	Air India	14:00	19:40	29h 40m	Jaipur	Lucknow	No Meal Fare	3 Stop(s)	15,614
5300	Air India	14:00	08:15	18h 15m	Jaipur	Lucknow	No Meal Fare	2 Stop(s)	16,192
5301	Air India	14:00	08:15	18h 15m	Jaipur	Lucknow	No Meal Fare	2 Stop(s)	16,192
5302	Air India	03:45	18:00	14h 15m	Jaipur	Lucknow	No Meal Fare	2 Stop(s)	16,559

5303 rows × 9 columns

Here I am importing the collected dataset which is in excel format and storing it into data frame (df) for further usage. Here we can observe first 5 and last 5 rows of the dataset. There are 5303 rows and 10 columns in the data frame. The dataset contains both numerical and categorical data. There are both dependent and independent variables present in the data frame. We have our target variable **"Price"** which stores the price of the flight tickets and it is continuous in nature which makes this problem to be a **"Regression Problem"**.

Features Information:

- Airline - The Name of airline
- Departure_time - The time when the journey starts from the source
- Time_of_arrival - Time of arrival at the destination
- Duration - Total duration taken by the flight to reach the destination from the source
- Source - The source from which the service begins
- Destination - The destination where the service ends
- Meal_availability - Availability of meals in the flight
- Number_of_stops - Total stops between the source and destination
- Price - The price of the flight ticket

Data Preprocessing Done

Data pre-processing is the process of converting raw data into a well-readable format to be used by Machine Learning model. Data pre-processing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we pre-process our data before feeding it into our model. I have used following pre-processing steps:

Checking the dimensions of the dataset

```
# Checking the dimensions of the dataset
print("There are {} rows and {} columns in our dataframe".format(df.shape[0], df.shape[1]))
```

```
There are 5303 rows and 9 columns in our dataframe
```

The dataset contains 5303 rows and 9 columns. Out of 9 columns 8 are independent variables and remaining one is our target variable "Price" which is dependent variable.

Checking the column names in the dataset

```
# Checking the column names in the dataset
print("Columns present in our dataset:\n",df.columns)
```

```
Columns present in our dataset:
 Index(['Airline', 'Departure_time', 'Time_of_arrival', 'Duration', 'Source',
       'Destination', 'Meal_availability', 'Number_of_stops', 'Price'],
      dtype='object')
```

These are the columns present in our dataset.

```
# Checking number of unique values in each column of dataset
df.nunique().to_frame("No of Unique Values")
```

	No of Unique Values
Airline	6
Departure_time	247
Time_of_arrival	253
Duration	398
Source	8
Destination	9
Meal_availability	3
Number_of_stops	5
Price	1711

Above are the number of unique values present in each of the columns present in the dataset.

```
# To get good overview of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5303 entries, 0 to 5302
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Airline            5303 non-null   object
 1   Departure_time     5303 non-null   object
 2   Time_of_arrival    5303 non-null   object
 3   Duration           5303 non-null   object
 4   Source             5303 non-null   object
 5   Destination        5303 non-null   object
 6   Meal_availability  5303 non-null   object
 7   Number_of_stops    5303 non-null   object
 8   Price              5303 non-null   object
dtypes: object(9)
memory usage: 373.0+ KB
```

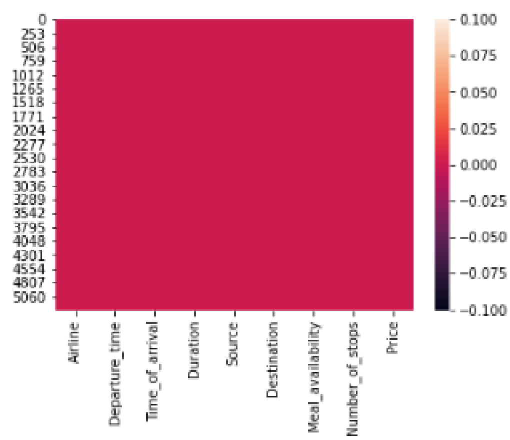
- This info() method gives the information about the dataset which includes indexing type, column type, no-null values and memory usage.
- The dataset contains object type data. We will encode the object datatypes using appropriate encoding techniques before building machine learning models.
- Since counts of all the columns are same, which means there are no null values present in the dataset.

```
# Checking null values in the dataset
df.isnull().sum()
```

```
Airline              0
Departure_time       0
Time_of_arrival      0
Duration             0
Source               0
Destination          0
Meal_availability    0
Number_of_stops      0
Price                0
dtype: int64
```

As we can see there are no missing values in any of the columns.

```
# Let's visualize the null values clearly
sns.heatmap(df.isnull())
plt.show()
```



Here we can clearly observe there are no missing values.

Value count function

Let's check the list of value counts in each columns to find if there are any unexpected or corrupted entries present in the dataset

```
# Checking the value counts of each columns
for i in df.columns:
        print(df[i].value_counts())
        print('*'*80)
```

```
Air India    1460
Vistara      1347
IndiGo       1344
Go First      556
Air Asia      416
SpiceJet      180
Name: Airline, dtype: int64
********************************************************************************
03:45    149
13:30    145
07:00    128
14:00    105
14:20     96
        ...
```

These are the value counts of the columns present in the dataset.

Feature Engineering

The columns Time_of_arrival and Departure_Time showing object data type which means python is not able to understand the type of data in this column due to some string values or categorical signs like ":" which we can observe in the value count function. Therefore, we have to convert this datatype into timestamp (datetime) to use them properly for prediction.

```
# Converting columns from object type to Datetime Type
df["Departure_time"] = pd.to_datetime(df["Departure_time"])
df["Time_of_arrival"] = pd.to_datetime(df["Time_of_arrival"])
```

```
#Checking the data types of all columns again
df.dtypes
```

```
Airline                      object
Departure_time       datetime64[ns]
Time_of_arrival      datetime64[ns]
Duration                     object
Source                       object
Destination                  object
Meal_availability            object
Number_of_stops              object
Price                        object
dtype: object
```

Duration

The column Duration has values in terms of minutes and hours. Duration means the time taken by the plane to reach the destination and it is the difference between the arrival time and Departure time. Let's extract proper duration time in terms of float data type from Time_of_arrival and Departure_time columns.

```
#Extracting Duration column using Time_of_arrival and Departure_Time
Difference = (df["Time_of_arrival"]-df["Departure_time"])
Diff_list = list()
for i in range(len(Difference)):
    duration = Difference.iloc[i].seconds/3600  # Converting difference into seconds and Dividing it by
    Diff_list.append(duration)
df["Duration"] = Diff_list
```

Departure_time

Let's extract values from Departure_time. Departure time means when a flight leaves the airport and this column contains hours and minutes so we will extract hours and minutes from Departure_time.

```
# Departure time means the time when the journey starts from the source.

# Extracting Hours from Departure_time column
df["Departure_Hour"] = pd.to_datetime(df["Departure_time"]).dt.hour

# Extracting Minutes from Dep_Time column
df["Departure_Min"] = pd.to_datetime(df["Departure_time"]).dt.minute
```

Now we have extracted hour and minute from Departure_time column. Let's drop Departure_time column as it is of no use now.

```
# Dropping Departure_time column
df.drop("Departure_time",axis=1,inplace=True)
```

Time_of_arrival

Similarly we can extract hours and minutes from Time_of_arrival column and dropping Time_of_arrival column.

```
# Arrival time is time of arrival at the destination.

# Extracting hour from Time_of_arrival column
df["Arrival_Hour"] = pd.to_datetime(df["Time_of_arrival"]).dt.hour

# Extracting Minutes from Arrival_Time column
df["Arrival_Min"] = pd.to_datetime(df["Time_of_arrival"]).dt.minute

# Dropping Arrival_Time column
df.drop("Time_of_arrival",axis=1,inplace=True)
```

Now we have extracted required data from the columns.

Price

The target column should be in continuous numeric data type but it is appearing as object data type due to some categorical sign ",". Let's replace this sign by empty space and convert the type into float.

```
# Let's replace "," sign by empty space
df['Price'] = df['Price'].str.replace(',','')
# Let's convert data type of Price column to float
df['Price'] = df['Price'].astype('float')
```

Meal_availability

From the value count function of Meal_availability we can observe "eCash 250" entry which does not belongs to meals so we can replace it as "None". Also, the other two entries "No meal fare" and "Free meal" belongs to same category that is they give same meaning so we can group them as well. We can also drop this column, but there are only few features in the dataset so, trying to retain the columns for prediction.

```
# Replacing "eCash250" by "None"
df['Meal_availability'] = df['Meal_availability'].replace('eCash 250','None')

# Grouping the entries with same meaning
df['Meal_availability'] = df['Meal_availability'].replace('No Meal Fare','Free Meal')
```

Number_of_stops

From the value count function of Number_of_stops we can observe the categorical values, let's replace them with numeric data.

```
# Replacing categorical values with numeric data
df.Number_of_stops.replace({"Non Stop": 0,"1 Stop": 1,"2 Stop(s)": 2,"3 Stop(s)": 3,"4 Stop(s)": 4},inp
```

Now we have successfully cleaned our data, let's have a look at dataframe.

```
# Checking dataset again
df
```

	Airline	Duration	Source	Destination	Meal_availability	Number_of_stops	Price	Departure_Hour	Departure_Min	Arri
0	Go First	4.666667	New Delhi	Mumbai	Free Meal	1	4941.0	6	0	
1	Go First	4.833333	New Delhi	Mumbai	Free Meal	1	4941.0	16	35	
2	Go First	6.750000	New Delhi	Mumbai	Free Meal	1	4941.0	9	0	
3	Go First	7.083333	New Delhi	Mumbai	Free Meal	1	4941.0	9	10	
4	Go First	8.666667	New Delhi	Mumbai	Free Meal	1	4941.0	5	25	
...	...	...	...	...	...	...	...	...	...	
5298	Air India	5.666667	Jaipur	Lucknow	Free Meal	2	15614.0	14	0	
5299	Air India	5.666667	Jaipur	Lucknow	Free Meal	3	15614.0	14	0	
5300	Air India	18.250000	Jaipur	Lucknow	Free Meal	2	16192.0	14	0	
5301	Air India	18.250000	Jaipur	Lucknow	Free Meal	2	16192.0	14	0	
5302	Air India	14.250000	Jaipur	Lucknow	Free Meal	2	16559.0	3	45	

5303 rows × 11 columns

```
# Checking shape of data after cleaning
df.shape
```

Now the dataset contains 5303 rows and 11 columns.

```
# Let's check the data types of the columns
df.dtypes
```

```
Airline              object
Duration            float64
Source               object
Destination          object
Meal_availability    object
Number_of_stops       int64
Price               float64
Departure_Hour        int64
Departure_Min         int64
Arrival_Hour          int64
Arrival_Min           int64
dtype: object
```

The dataframe has 3 types of data that is object, integer and float data types. We will encode the object data types before building the ML model.

```python
# Checking the uniqueness of target column
df["Price"].unique()
```

```
array([ 4941.,  5401.,  5953., ..., 10624., 16192., 16559.])
```

These are the unique values present in the target column.

```python
# Checking whether the target contains any space
df.loc[df['Price']==" "]
```

Airline	Duration	Source	Destination	Meal_availability	Number_of_stops	Price	Departure_Hour	Departure_Min	Arrival_Hou

There are no any empty spaces in any of the columns.

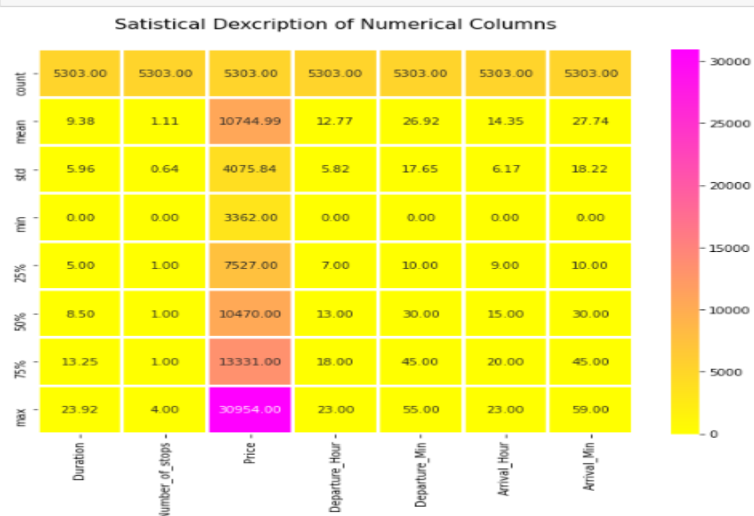
Data Inputs- Logic- Output Relationships

Description of Dataset

```python
# Statistical summary of dataset
df.describe()
```

	Duration	Number_of_stops	Price	Departure_Hour	Departure_Min	Arrival_Hour	Arrival_Min
count	5303.000000	5303.000000	5303.000000	5303.000000	5303.000000	5303.000000	5303.000000
mean	9.378550	1.108806	10744.987366	12.769187	26.921554	14.345276	27.741279
std	5.961941	0.638011	4075.843078	5.819190	17.645870	6.165711	18.222984
min	0.000000	0.000000	3362.000000	0.000000	0.000000	0.000000	0.000000
25%	5.000000	1.000000	7527.000000	7.000000	10.000000	9.000000	10.000000
50%	8.500000	1.000000	10470.000000	13.000000	30.000000	15.000000	30.000000
75%	13.250000	1.000000	13331.000000	18.000000	45.000000	20.000000	45.000000
max	23.916667	4.000000	30954.000000	23.000000	55.000000	23.000000	59.000000

```python
# Visualizing the statistical description of numeric datatype columns

plt.figure(figsize = (10,8))
sns.heatmap(round(df.describe()[0:],2), linewidth = 2, annot= True, fmt = ".2f", cmap="spring_r")
plt.title("Satistical Dexcription of Numerical Columns\n",fontsize=15)

plt.show()
```



From the heat map we can observe the statistical summary of the numerical features present in the dataset.

Assumptions

This gives the statistical information of the dataset. The summary of this dataset looks perfect since there is no negative/ invalid values present. It gives the summary of numerical data.

From the above description we can observe the following things

- The counts of every column is same which means there are no missing values present in the dataset.
- The mean value is greater than the median (50%) in the columns Duration, Number_of_stops, and Price so we can say these columns are skewed to right.
- The median (50%) is bit greater than mean in Departure_Hour, Departure_Min, Arrival_Hour, Arrival_Hour and Arrival_Min which means these columns are skewed to left.
- From the description we can say the minimum price of the flight ticket is Rs.3362.00 and maximum price is Rs.30954.00 also the mean is 10744.987366.
- In summarizing the data we can observe that there is huge difference in maximum and 75% percentile in the columns Price, Duration, etc that means huge outliers present in those columns. These differences can also be seen in many other columns. So we need to remove outliers and skewness to get better model and prediction.

Hardware and Software Requirements and Tools Used

To build the machine learning projects it is important to have the following hardware and software requirements and tools.

Hardware required:

- Processor: core i5 or above
- RAM: 8 GB or above
- ROM/SSD: 250 GB or above

Software required:

- Distribution: Anaconda Navigator
- Programming language: Python
- Browser based language shell: Jupyter Notebook
- Chrome: To scrape the data

Model/s Development and evaluation

Visualizations

```
# Separating numerical and categorcal columns in the dataset

# Checking for categorical columns
categorical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_columns.append(i)
print("Categorical columns present in the dataset are:\n",categorical_columns)

# Now checking for numerical columns
numerical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_columns.append(i)
print("\nNumerical columns present in the dataset are:\n",numerical_columns)
```

```
Categorical columns present in the dataset are:
 ['Airline', 'Source', 'Destination', 'Meal_availability']

Numerical columns present in the dataset are:
 ['Duration', 'Number_of_stops', 'Price', 'Departure_Hour', 'Departure_Min', 'Arrival_Hour', 'Arrival
_Min']
```

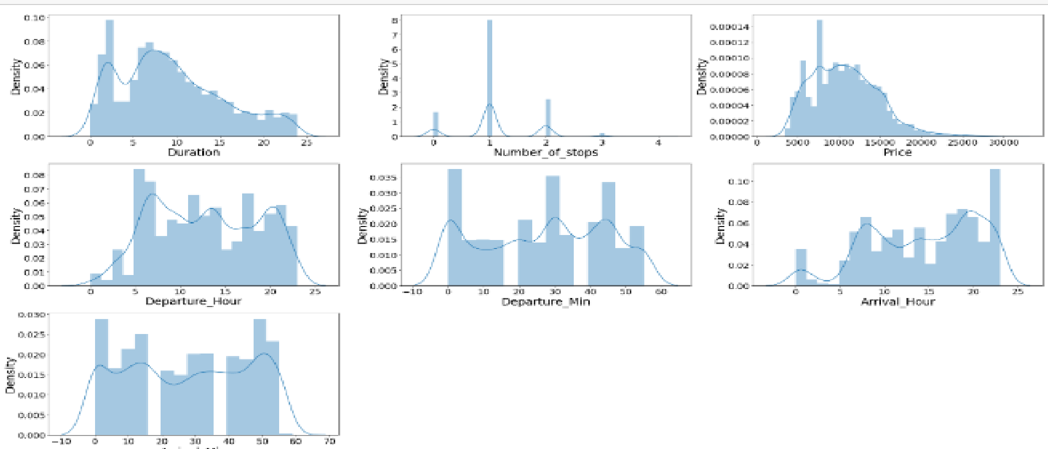
Exploratory Data Analysis (EDA)

Data Visualization

Univariate Analysis

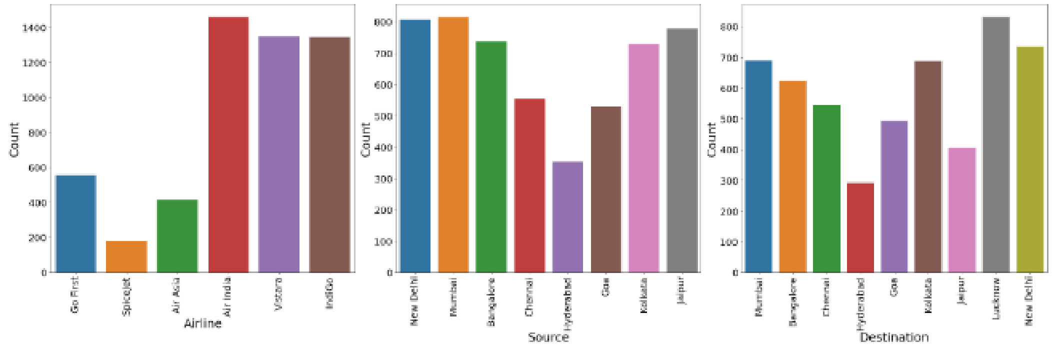
Plotting categorical Variables

```
#Distribution plot for all numerical columns
plt.figure(figsize = (30,16))
plotnumber = 1
for column in df[numerical_columns]:
    if plotnumber <=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize = 25)
        plt.ylabel('Density',fontsize = 25)
        plt.xticks(fontsize=20)
        plt.yticks(fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



There is no skewness in any of the numerical columns.

```python
#Bar plot for all Categorical columns
plt.figure(figsize = (30,10))
plotnumber = 1
for column in df[categorical_columns]:
    if plotnumber <=3:
        ax = plt.subplot(1,3,plotnumber)
        sns.countplot(df[column])
        plt.xlabel(column,fontsize = 25)
        plt.ylabel('Count',fontsize = 25)
        plt.xticks(rotation=90,fontsize=20)
        plt.yticks(fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

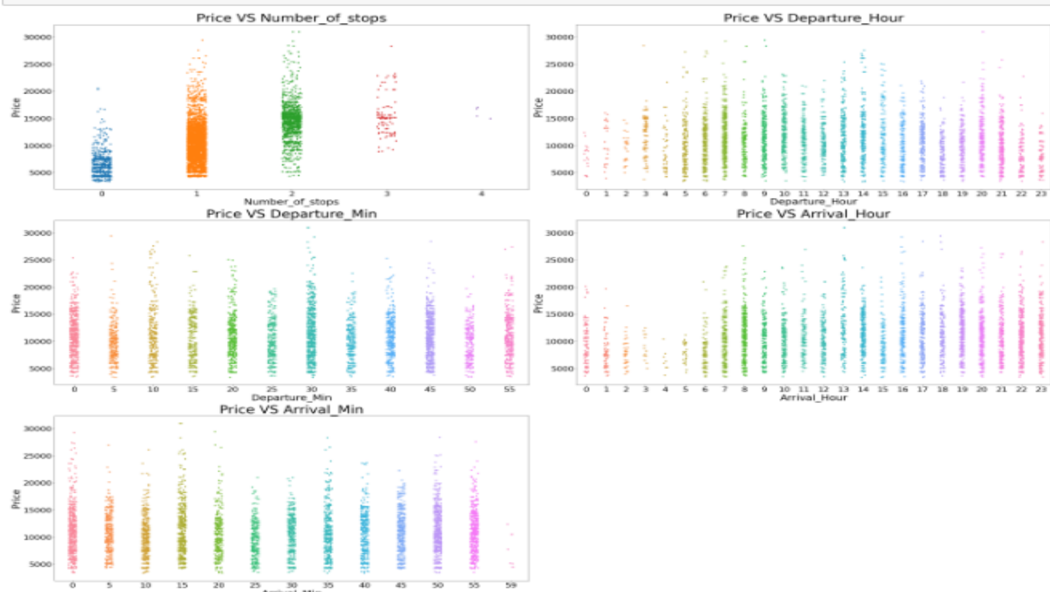


- Indigo has maximum count which means most of the passengers preferred Indigo for there travelling.
- New Delhi has maximum count for source which means maximum passengers are choosing New Delhi as there source.
- New Delhi has maximum count for Destination which means maximum passengers are choosing New Delhi as there Destination.

Bivariate Analysis:

```python
col=['Number_of_stops', 'Departure_Hour', 'Departure_Min', 'Arrival_Hour', 'Arrival_Min']
```

```python
#stripplot for numerical columns
plt.figure(figsize=(40,40))
for i in range(len(col)):
    plt.subplot(4,2,i+1)
    sns.stripplot(x=df[col[i]] , y=df['Price'])
    plt.title(f"Price VS {col[i]}",fontsize=40)
    plt.xticks(fontsize=25)
    plt.yticks(fontsize=25)
    plt.xlabel(col[i],fontsize = 30)
    plt.ylabel('Price',fontsize = 30)
    plt.tight_layout()
```

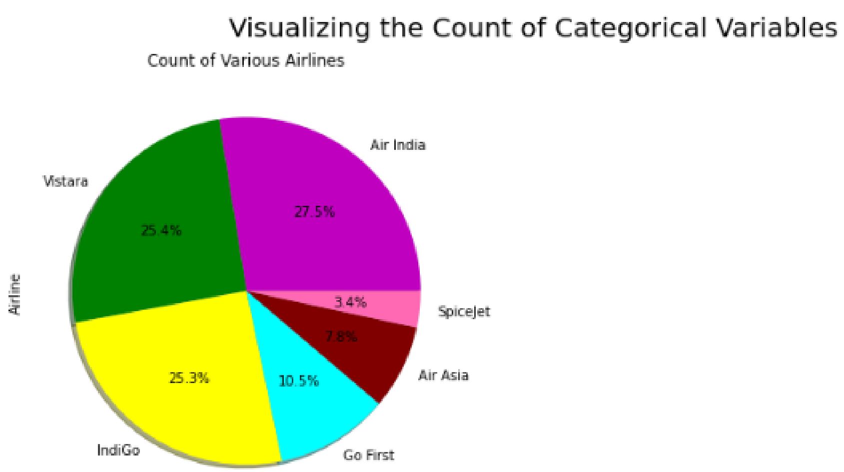


Observations:

- Flights with 1 stop costs more price compared to other flights.
- At 2PM departure time of every day the flight Prices are high so it looks good to book flights rather than this departure time.
- And Departure minute has less relation with target Price.
- At 7AM to 1PM Arrival time of every day the flight Prices are high so it looks good to book flights rather than this arrival time.
- And Arrival minute has less relation with target Price.

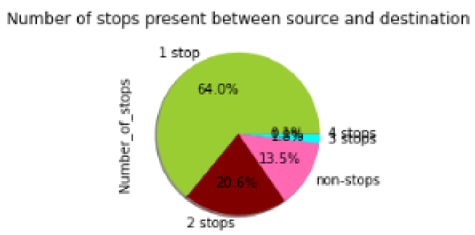
```python
# Visualizing the count of categorical variables
plt.figure(figsize=(15,6))
plt.suptitle('Visualizing the Count of Categorical Variables',fontsize=20)

# Visualizing the count of categorical variables
plt.figure(figsize=(15,6))
plt.suptitle('Visualizing the Count of Categorical Variables',fontsize=20)

# Visualizing the count of Airlines
plt.subplot(1,2,1)
print(df["Airline"].value_counts(),"\n")
plt.title('Count of Various Airlines')
labels = ['Air India', 'Vistara', 'IndiGo', 'Go First', 'Air Asia', 'SpiceJet']
colors = ["m", "green","yellow", "cyan", "maroon", "hotpink"]
df['Airline'].value_counts().plot.pie(autopct='%1.1f%%',shadow=True,labels=labels,fontsize=10,colors=c
```

```
Air India    1460
Vistara      1347
IndiGo       1344
Go First      556
Air Asia      416
SpiceJet      180
Name: Airline, dtype: int64


<AxesSubplot:title={'center':'Count of Various Airlines'}, ylabel='Airline'>

<Figure size 1080x432 with 0 Axes>
```



```
# Visualizing the count of Number_of_stops
plt.subplot(1,2,2)
print(df['Number_of_stops'].value_counts())
plt.title('Number of stops present between source and destination')
labels = ['1 stop', '2 stops', 'non-stops', '3 stops', '4 stops']
colors = ["yellowgreen","maroon","hotpink","cyan","red"]
df['Number_of_stops'].value_counts().plot.pie(autopct='%1.1f%%',shadow=True,labels=labels,fontsize=10,c
plt.show()
```

```
1    3396
2    1093
0     716
3      94
4       4
Name: Number_of_stops, dtype: int64
```

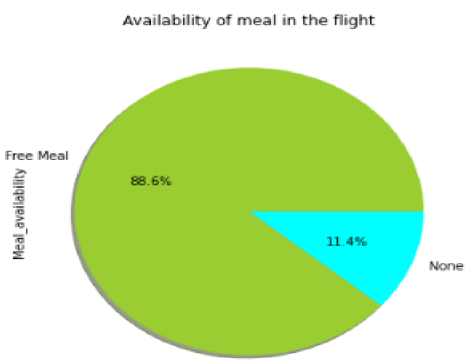


Observations:

- **Airline:** From the pie plot we can infer that there are more number of flights of "Air India", "Vistara" and "Indigo" compared to others. Also, the count of Spicejet flights are very less.
- **Number_of_stops:** From the above pie plot we can infer that 64% of the flights have only 1 stop during the journey and some of the flights (20.6%) have 2 stops where only few flights have 3 and 4 stops.

```
# Visualizing the availability of meal in the flight
plt.figure(figsize=(6,6))
print(df["Meal_availability"].value_counts())
plt.title('Availability of meal in the flight')
labels = ['Free Meal', 'None']
colors = ["yellowgreen","cyan"]
df['Meal_availability'].value_counts().plot.pie(autopct='%1.1f%%',shadow=True,labels=labels,fontsize=1(
plt.show()
```

```
Free Meal    4701
None          602
Name: Meal_availability, dtype: int64
```

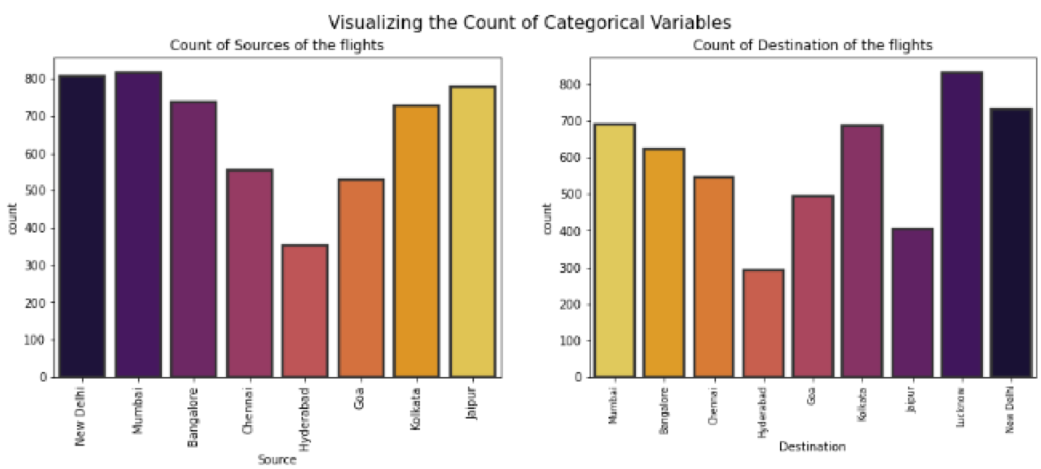


Observations:

- **Meal_availability:** Most of the flights providing free meals and only few flights are not providing any meals.

```python
# Visualizing the count of categorical variables
plt.figure(figsize=(15,5))
plt.suptitle('Visualizing the Count of Categorical Variables',fontsize=15)

# Visualizing the count of Sources of the flights
plt.subplot(1,2,1)
print(df['Source'].value_counts(),"\n")
plt.title('Count of Sources of the flights')
sns.countplot('Source', data=df, palette="inferno",linewidth=2.3, edgecolor=".2")
plt.xticks(rotation=90)

# Visualizing the count of Destination of the flights
plt.subplot(1,2,2)
print(df['Destination'].value_counts())
plt.title('Count of Destination of the flights')
sns.countplot('Destination', data=df, palette="inferno_r",linewidth=2.3, edgecolor=".2")
plt.xticks(fontsize='8')
plt.xticks(rotation=90)
plt.show()
```

```
Mumbai       816
New Delhi    807
Jaipur       778
Bangalore    738
Kolkata      728
Chennai      553
Goa          529
Hyderabad    354
Name: Source, dtype: int64

Lucknow      832
New Delhi    734
Mumbai       690
Kolkata      687
Bangalore    624
Chennai      546
Goa          494
Jaipur       405
Hyderabad    291
Name: Destination, dtype: int64
```



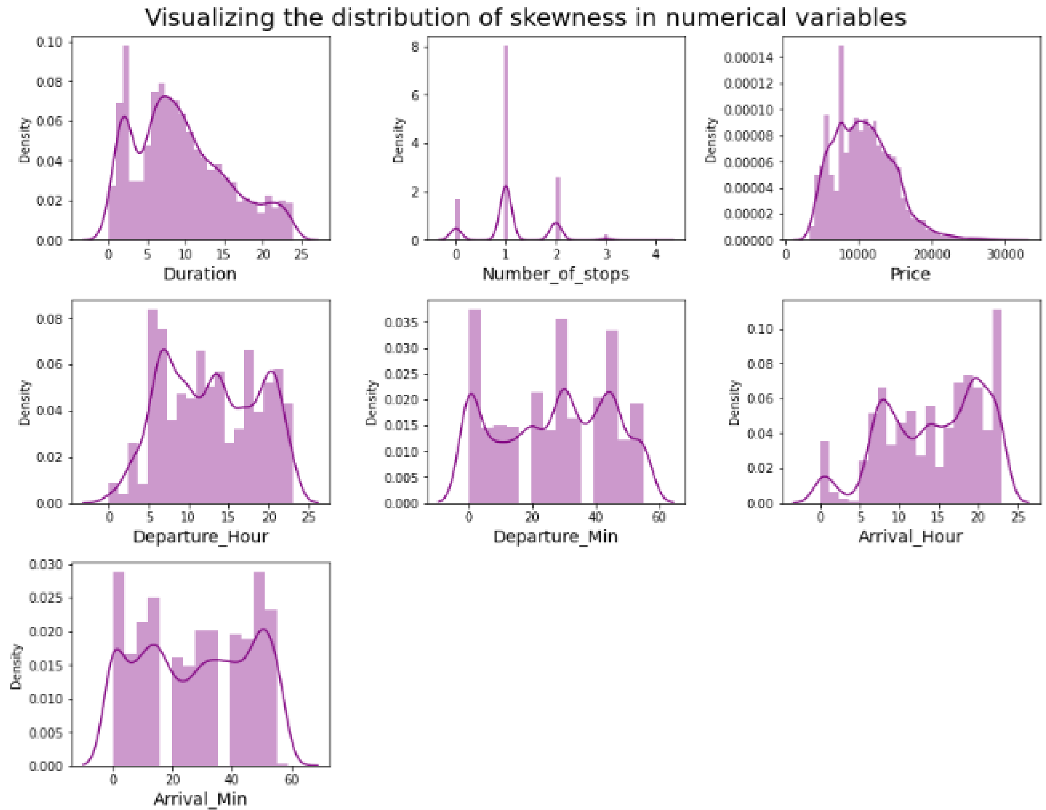
Observations:

- **Source:** From the count plot we can observe more number of flights are from Mumbai, New Delhi, Jaipur, Kolkata and Bangalore. Only few flights are from Hyderabad.
- **Destination:** More number of flights are heading towards Lucknow, New Delhi and Kolkata. Only few flights are travelling to Hyderabad.

Distribution of skewness

Plotting Numerical Variables

```
# Checking how the data has been distriubted in each column

plt.figure(figsize=(12,12),facecolor='white')
plt.suptitle("Visualizing the distribution of skewness in numerical variables",fontsize=20)
plotnumber=1
for column in numerical_columns:
    if plotnumber<=7:
        ax=plt.subplot(4,3,plotnumber)
        sns.distplot(df[column], color="purple")
        plt.xlabel(column,fontsize=14)
    plotnumber+=1
plt.tight_layout()
```



Observations:

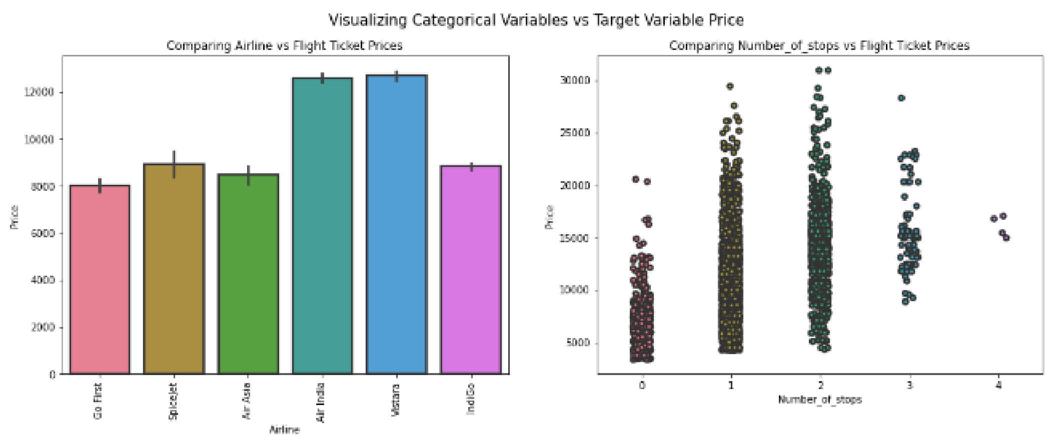
Above plot shows how the data has been distributed in each of the columns.

- From the distribution plot we can observe the columns are somewhat distributed normally as they have no proper bell shape curve.
- The columns like "Duration", "Number_of_stops" and "Price" are skewed to right as the mean value in these columns are much greater than the median(50%).
- Also the data in the column Arrival_Hour skewed to left since the mean values is less than the median.
- Since there is presence of skewness in the data, we need to remove skewness in the numerical columns to overcome with any kind of data biasness.

Bivariate Analysis

Visualizing Categorical Variables vs Target Variable Price

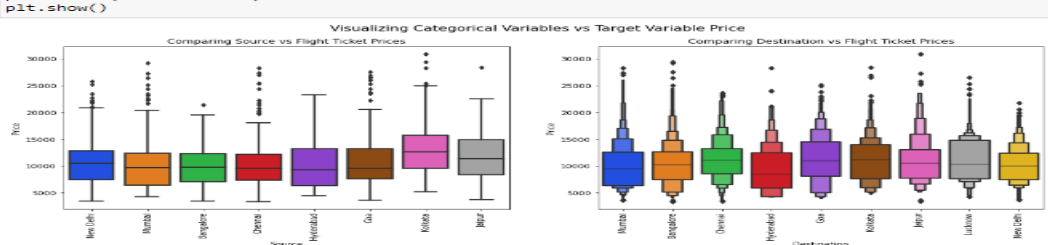
```
plt.figure(figsize = (18,6))
plt.suptitle("Visualizing Categorical Variables vs Target Variable Price",fontsize=15)

# Checking which Airline is expensive based on Price of tickets
plt.subplot(1,2,1)
plt.title("Comparing Airline vs Flight Ticket Prices")
sns.barplot(x= df['Airline'],y= df['Price'],palette = "husl",linewidth=2.3, edgecolor=".2")
plt.xticks(rotation = 90)

# Checking flights which have meals availability are expensive or not?
plt.subplot(1,2,2)
plt.title("Comparing Number_of_stops vs Flight Ticket Prices")
sns.stripplot(x = df['Number_of_stops'],y= df['Price'],palette = "husl",linewidth=2.3, edgecolor=".2")
plt.show()
```



Observations:

- **Airline vs Price:** From the bar plot we can notice "Vistara" and "Air India" airlines have highest ticket prices compared to other airlines.
- **Number_of_stops vs Price:** From the strip plot we can notice the flights which have 1 and 2 stops between source and destination have highest ticket prices compared to others. The airlines which have 4 stops during the journey have very less ticket price. So we can say as the stops increases, ticket price decreases.

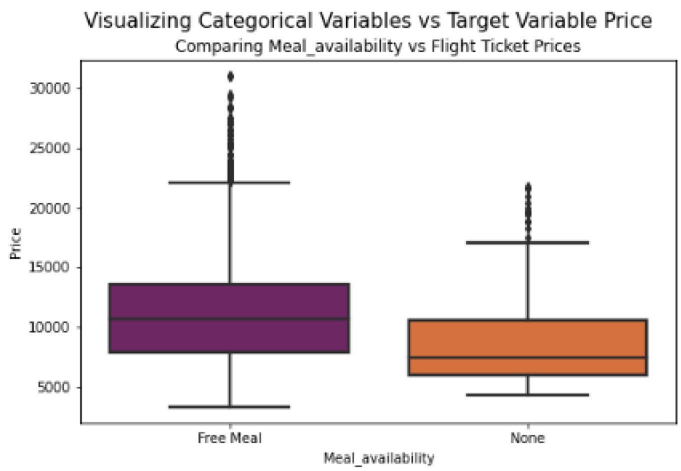
```
plt.figure(figsize = (18,6))
plt.suptitle("Visualizing Categorical Variables vs Target Variable Price\n",fontsize=15)

# Checking which source has highest ticket price
plt.subplot(1,2,1)
plt.title("Comparing Source vs Flight Ticket Prices")
sns.boxplot(x= df['Source'],y= df['Price'],palette = "bright",linewidth=2.3)
plt.xticks(rotation = 90)

# Checking how prices changes in each destination
plt.subplot(1,2,2)
plt.title("Comparing Destination vs Flight Ticket Prices")
sns.boxenplot(x = df['Destination'],y= df['Price'],palette = "bright",linewidth=2.3)
plt.xticks(rotation = 90)
plt.show()
```



Observations:

- **Source vs Price:** From the box plot we can observe the flights from Kolkata are having somewhat higher prices compared to other sources.
- **Destination vs Price:** From the boxen plot we can notice that the flights travelling to Goa have higher flight ticket prices.

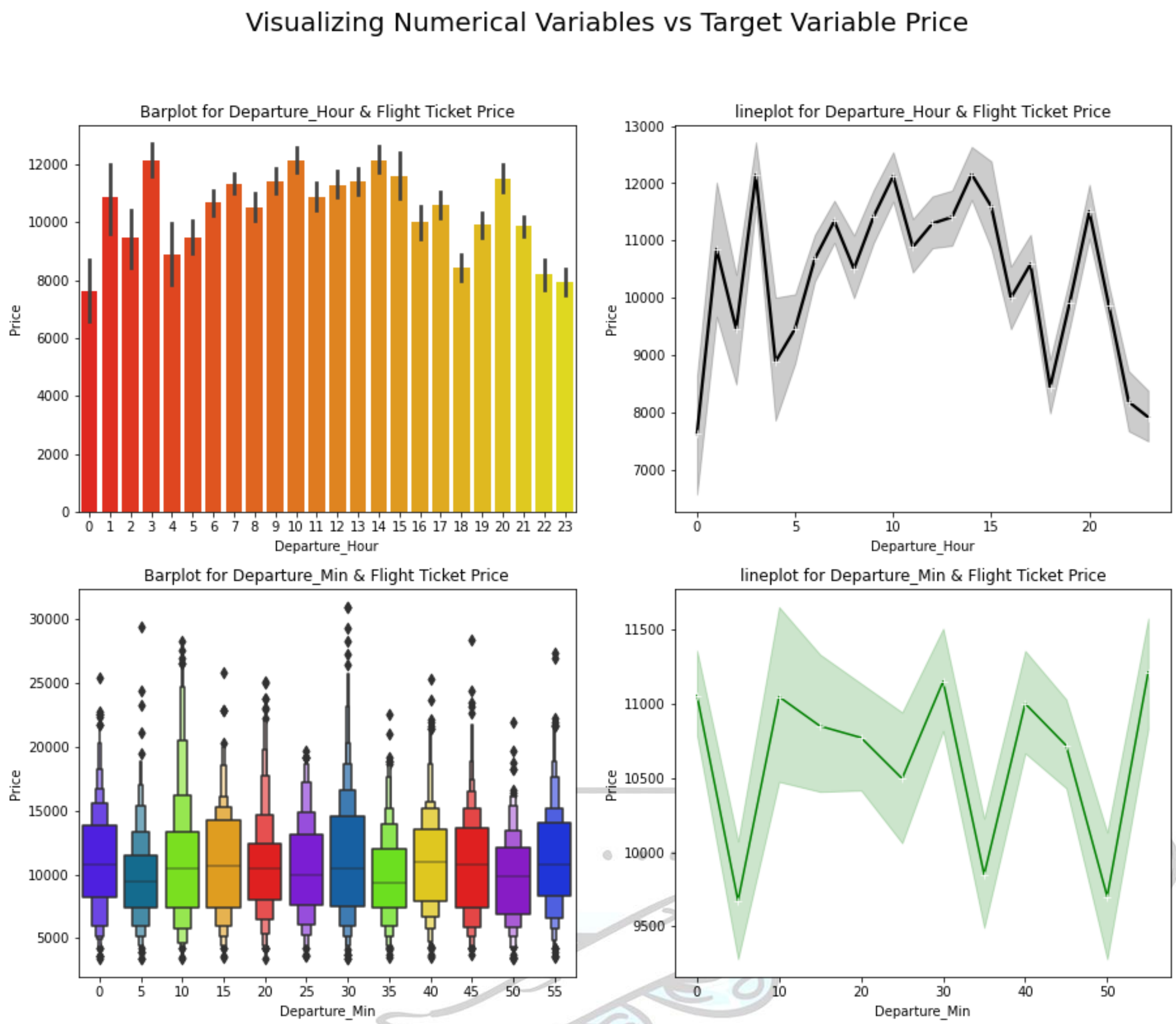
```
plt.figure(figsize = (8,5))
plt.suptitle("Visualizing Categorical Variables vs Target Variable Price\n",fontsize=15)
plt.title("Comparing Meal_availability vs Flight Ticket Prices")
sns.boxplot(x= df['Meal_availability'],y= df['Price'],palette = "inferno",linewidth=2.3)
plt.show()
```



Observations:

- **Meal_availability vs Price:** The boxplot shows the flights having Free meal facility have high ticket prices.

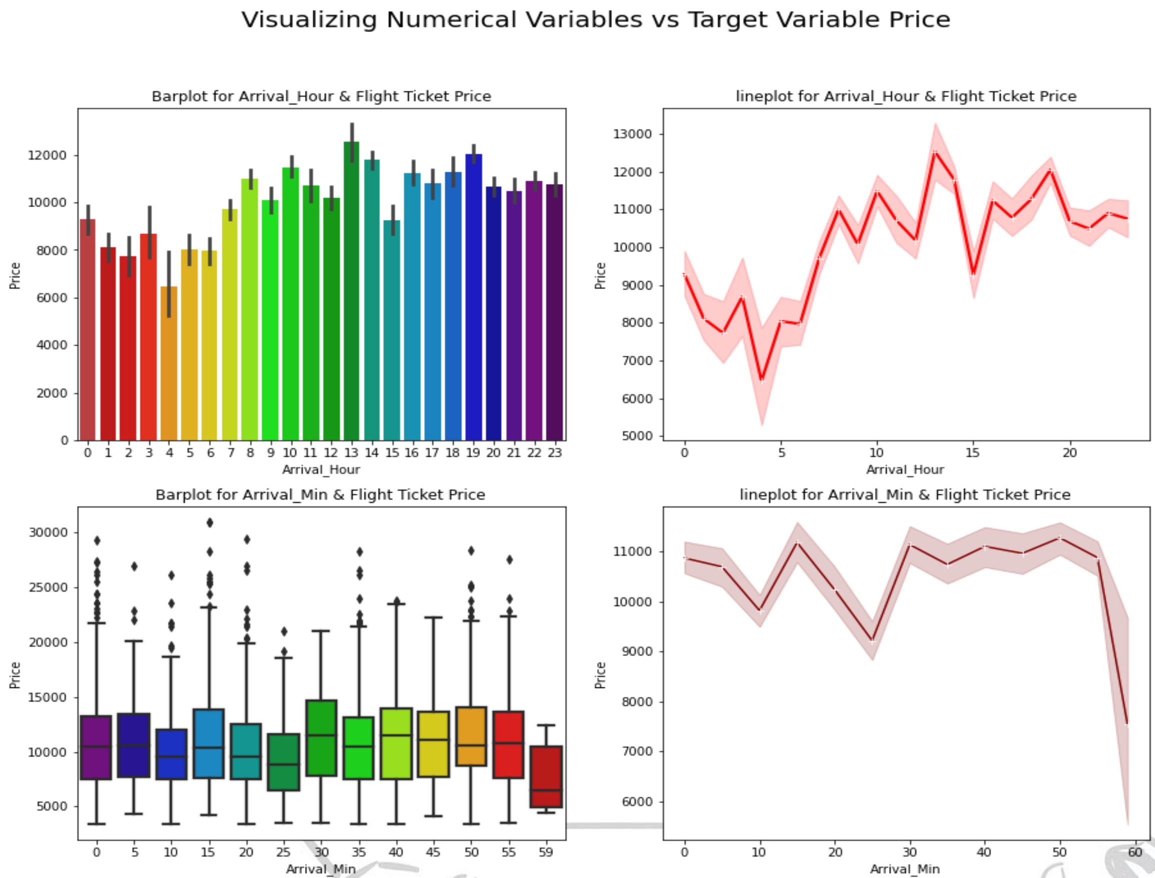
```
plt.figure(figsize = (15,12))
plt.suptitle("Visualizing Numerical Variables vs Target Variable Price",fontsize=20)

plt.subplot(2,2,1)
plt.title("Barplot for Departure_Hour & Flight Ticket Price")
sns.barplot(x= df['Departure_Hour'],y= df['Price'],palette = "autumn",linewidth=2.3)

plt.subplot(2,2,2)
plt.title("lineplot for Departure_Hour & Flight Ticket Price")
sns.lineplot(x = df['Departure_Hour'],y= df['Price'],marker="+",color='k',linewidth=2.3)

plt.subplot(2,2,3)
plt.title("Barplot for Departure_Min & Flight Ticket Price")
sns.boxenplot(x='Departure_Min',y='Price',data=df,palette = "prism",color='k')

plt.subplot(2,2,4)
plt.title("lineplot for Departure_Min & Flight Ticket Price")
sns.lineplot(x='Departure_Min',y='Price',data=df,marker="+",color='g')

plt.show()
```



Observations:

- **Departure_Hour vs Price:** From the bar plot and line plot we can see that there are some flights departing in the early morning 3 AM having most expensive ticket prices compared to late morning flights. We can also observe the flight ticket prices are higher during afternoon (may fluctuate) and it decreases in the evening.
- **Departure_Min vs Price:** The boxen plot and line plot gives there is no significant difference between price and departure min.

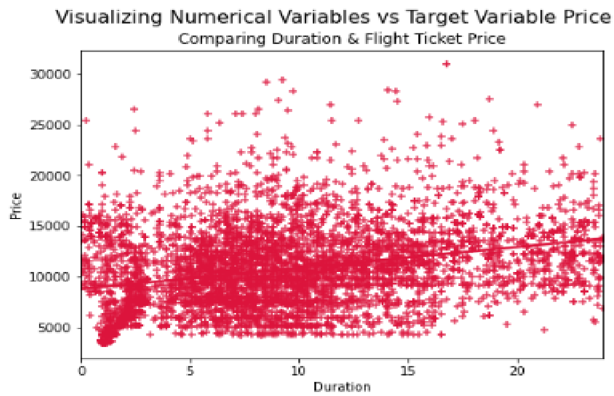
```
plt.figure(figsize = (15,12))
plt.suptitle("Visualizing Numerical Variables vs Target Variable Price",fontsize=20)

plt.subplot(2,2,1)
plt.title("Barplot for Arrival_Hour & Flight Ticket Price")
sns.barplot(x= df['Arrival_Hour'],y= df['Price'],palette = "nipy_spectral_r",linewidth=2.3)

plt.subplot(2,2,2)
plt.title("lineplot for Arrival_Hour & Flight Ticket Price")
sns.lineplot(x = df['Arrival_Hour'],y= df['Price'],marker="+",color='r',linewidth=2.3)

plt.subplot(2,2,3)
plt.title("Barplot for Arrival_Min & Flight Ticket Price")
sns.boxplot(x='Arrival_Min',y='Price',data=df,palette = "nipy_spectral",linewidth=2.3)

plt.subplot(2,2,4)
plt.title("lineplot for Arrival_Min & Flight Ticket Price")
sns.lineplot(x='Arrival_Min',y='Price',data=df,marker="+",color='maroon')

plt.show()
```



Observations:

- **Arrival_Hour vs Price:** From the bar plot and line plot we can observe that very few flights are arriving in the early morning that is 0 to 6 AM they have very less ticket price. Also, the flights which are arriving in the afternoon and evening have somewhat higher price. So, we can conlude this column has some positive correlation with price.
- **Arrival_Min vs Price:** There is no significant difference between this feature and price. We can say flight ticket prices are not much dependent on the Arrival_min.

```
# Visualizing duration and price
plt.figure(figsize = (7,5))
plt.suptitle("Visualizing Numerical Variables vs Target Variable Price",fontsize=15)

plt.title("Comparing Duration & Flight Ticket Price")
sns.regplot(x= df['Duration'],y= df['Price'],marker="+",color='crimson')
plt.show()
```

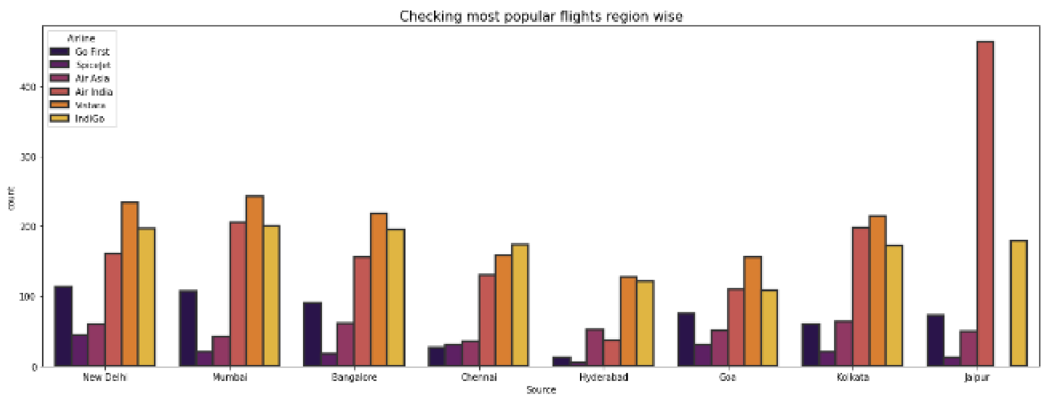


Observations:

- **Duration vs Price:** From the reg plot we can observe some positive linear relation between Duration and Price. Flights having 1-12 hours of duration, they have ticket price of around 15000.

Till now we have checked the relation between the independent variables and dependent variable that is our target column "Price". Now let's check the relation between two independent variables and compare each of them with others.

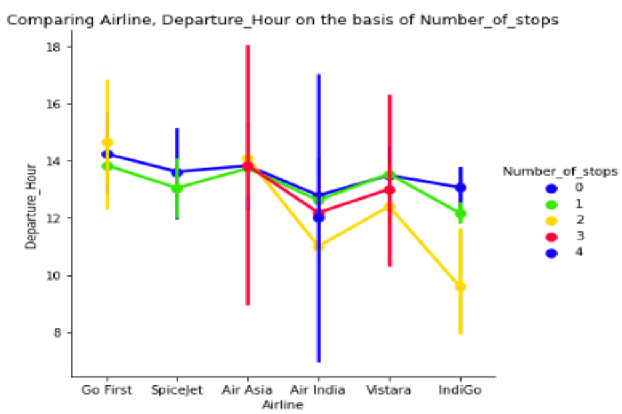
```
# Lets chcek the most popular flight region wise
plt.figure(figsize=(20,7))
sns.countplot(x = "Source", hue = "Airline", data = df, palette = "inferno",linewidth=2.3, edgecolor="
plt.title("Checking most popular flights region wise",fontsize=15)
plt.show()
```



Observations:

- **Source vs Airline:** The plot showing the region wise count of airlines which tells us that Jaipur source is not having Vistara flights and it has Air India flights in higher count compared to other sources. Other sources have Air India, Vistara and Indigo flights with higher count.

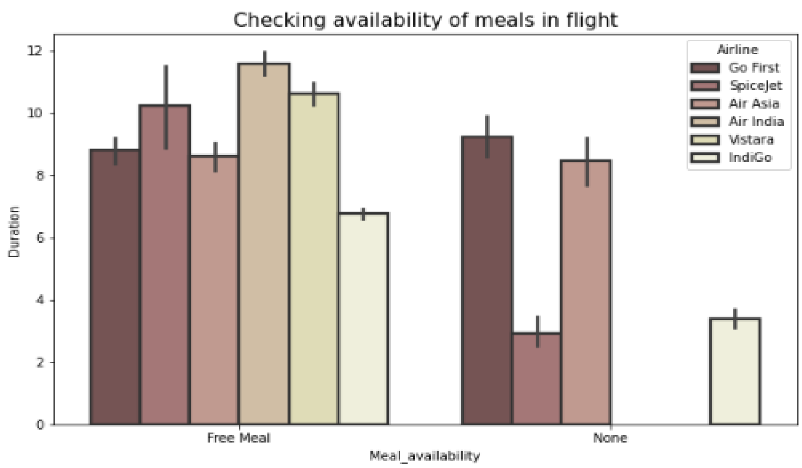
```
# Lets check the relation between independent variables
sns.factorplot(x= "Airline", y="Departure_Hour",hue="Number_of_stops",palette="prism", data=df)
plt.title("Comparing Airline, Departure_Hour on the basis of Number_of_stops")
plt.show()
```



Observations:

- Above plot gives the relation between Airline and Departure hour based on Number of stops. Air India and Air Asia flights are departing in the evening and they have less than 4 stops during the journey.

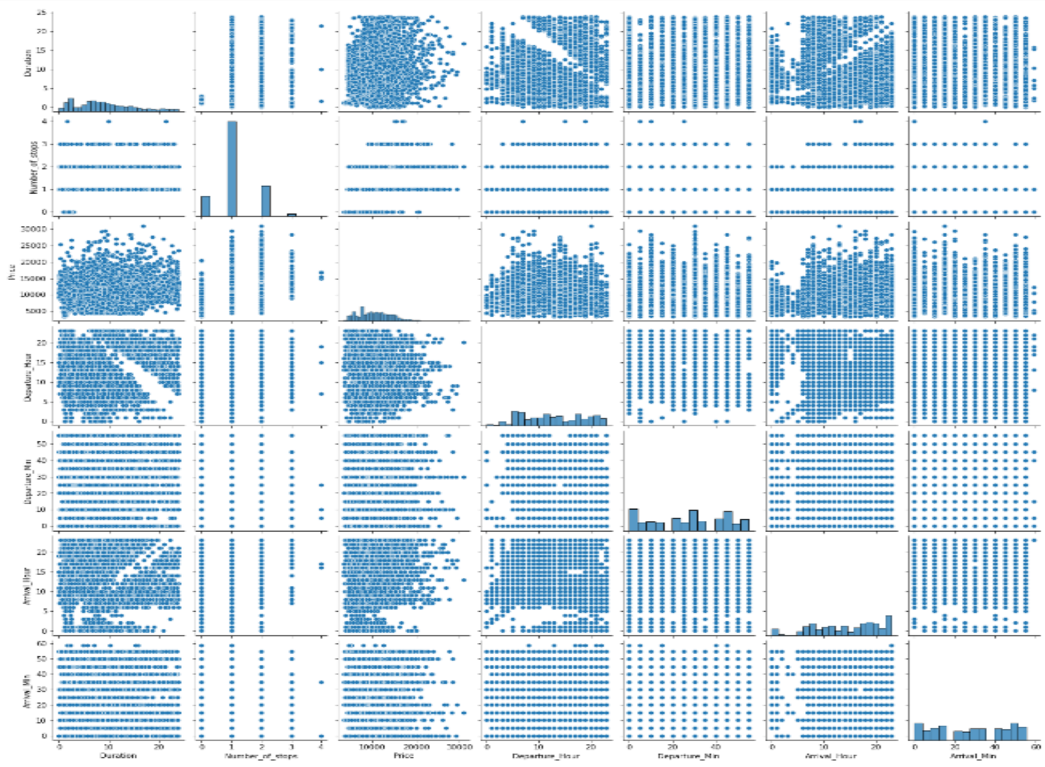
```
# Lets chcek the most popular flight region wise
plt.figure(figsize=(10,6))
sns.barplot(x = "Meal_availability", y = "Duration",hue="Airline",data = df, palette = "pink",linewidtl
plt.title("Checking availability of meals in flight",fontsize=15)
plt.show()
```



Observations:

- All the airlines provides free meals during the journey having the duration below 11 hours.

```
sns.pairplot(df)
plt.show()
```

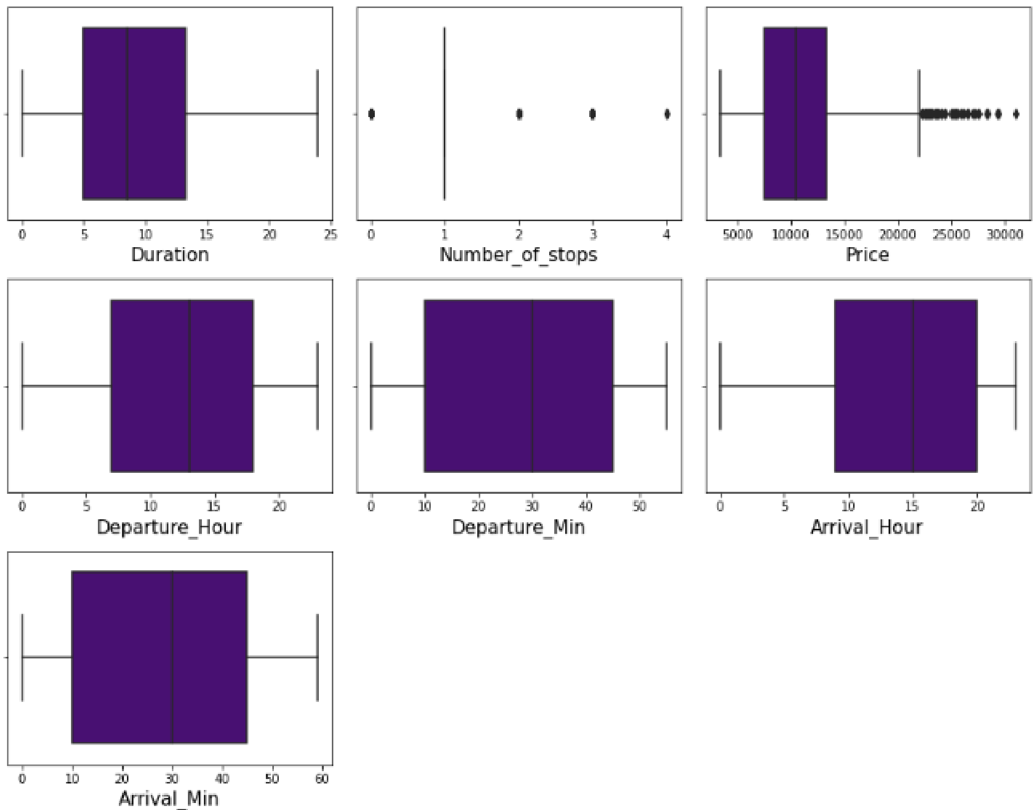


- This pair plot gives the pairwise relation between the columns, we can observe the relation between the features
- Here we can observe the correlation between the features and on the diagonal we can notice the distribution plot which shows whether the column has skewness or not.

Identification of possible problem-solving approaches (methods)

Identifying the outliers

```
# Identifying the outliers using boxplot

plt.figure(figsize=(12,12),facecolor='white')
plotnumber=1
for column in numerical_columns:
    if plotnumber<=7:
        ax=plt.subplot(4,3,plotnumber)
        sns.boxplot(df[column],color="indigo")
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.tight_layout()
```



- The outliers present in Number_of_stops and "Price" columns.
- Since Price is our target column and Number_of_stops is our categorical variable so no need to remove outliers in this columns. Finally there is no need to remove outliers in the dataset.

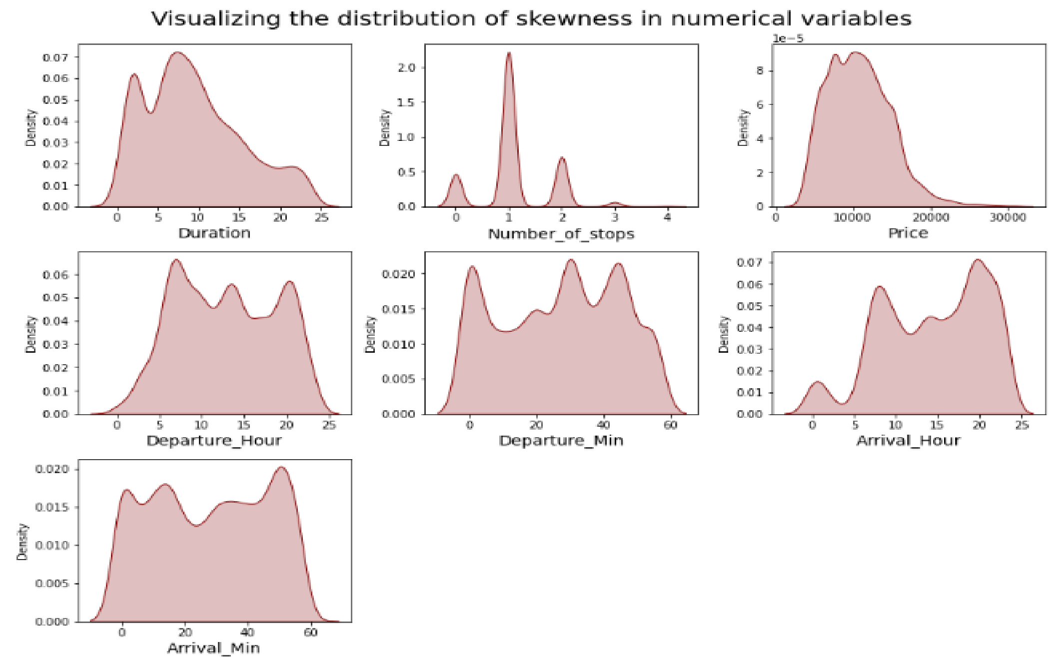
Checking for skewness in the data

```
# Checking the skewness
df.skew()
```

```
Duration           0.549978
Number_of_stops    0.381915
Price              0.700980
Departure_Hour     0.035683
Departure_Min     -0.116381
Arrival_Hour      -0.418887
Arrival_Min       -0.032736
dtype: float64
```

We can find the skewness in Duration column and Price column. Price is our target variable we should not loose any data so, no need to remove skewness in this column. The skewness in Duration column is also near normal so, let's not remove skewness in Duration column.

```
# Checking how the data has been distriubted after removing skewness in the column
plt.figure(figsize=(12,12),facecolor='white')
plt.suptitle("Visualizing the distribution of skewness in numerical variables",fontsize=20)
plotnumber=1
for column in numerical_columns:
    if plotnumber<=7:
        ax=plt.subplot(4,3,plotnumber)
        sns.distplot(df[column],hist=False, color="maroon", kde_kws={"shade": True})
        plt.xlabel(column,fontsize=14)
    plotnumber+=1
plt.tight_layout()
```



Encoding the categorical columns using Label Encoder Method

```
# Converting categorical data into numerical using Label Encoder method
from sklearn.preprocessing import LabelEncoder

LE=LabelEncoder()
df[categorical_columns]= df[categorical_columns].apply(LE.fit_transform)
```

```
# Displaying dataframe after encoding
df
```

	Airline	Duration	Source	Destination	Meal_availability	Number_of_stops	Price	Departure_Hour	Departure_Min	Arri
0	2	4.666667	7	7	0	1	4941.0	6	0	
1	2	4.833333	7	7	0	1	4941.0	16	35	
2	2	6.750000	7	7	0	1	4941.0	9	0	
3	2	7.083333	7	7	0	1	4941.0	9	10	
4	2	8.666667	7	7	0	1	4941.0	5	25	
...	...	...	...	...	...	...	...	...	...	
5298	1	5.666667	4	6	0	2	15614.0	14	0	
5299	1	5.666667	4	6	0	3	15614.0	14	0	
5300	1	18.250000	4	6	0	2	16192.0	14	0	
5301	1	18.250000	4	6	0	2	16192.0	14	0	
5302	1	14.250000	4	6	0	2	16559.0	3	45	

5303 rows × 11 columns

Now we have converted the categorical columns into numerical columns using label encoding method.

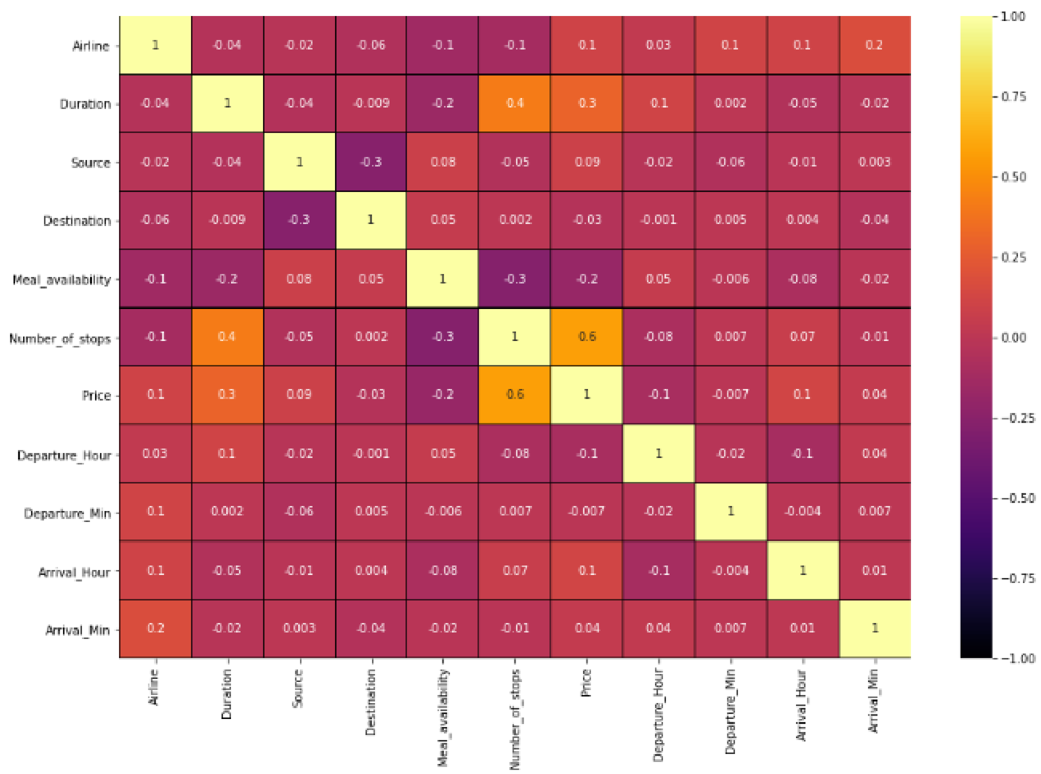
Correlation between the target variable and independent variables using HEAT map

```
# Checking the correlation between features and the target
cor = df.corr()
cor
```

	Airline	Duration	Source	Destination	Meal_availability	Number_of_stops	Price	Departure_Hour
Airline	1.000000	-0.037365	-0.016840	-0.055112	-0.134730	-0.115706	0.106153	0.032536
Duration	-0.037365	1.000000	-0.040688	-0.008555	-0.159021	0.415225	0.302224	0.112678
Source	-0.016840	-0.040688	1.000000	-0.265382	0.084165	-0.053723	0.093260	-0.020957
Destination	-0.055112	-0.008555	-0.265382	1.000000	0.052409	0.001920	-0.034189	-0.001497
Meal_availability	-0.134730	-0.159021	0.084165	0.052409	1.000000	-0.271619	-0.187175	0.049236
Number_of_stops	-0.115706	0.415225	-0.053723	0.001920	-0.271619	1.000000	0.575886	-0.078935
Price	0.106153	0.302224	0.093260	-0.034189	-0.187175	0.575886	1.000000	-0.100619
Departure_Hour	0.032536	0.112678	-0.020957	-0.001497	0.049236	-0.078935	-0.100619	1.000000
Departure_Min	0.111513	0.001946	-0.060523	0.005001	-0.005787	0.007392	-0.006722	-0.019034
Arrival_Hour	0.104351	-0.047371	-0.014840	0.003929	-0.082810	0.068120	0.124571	-0.103664
Arrival_Min	0.175095	-0.017577	0.003363	-0.039590	-0.015569	-0.010702	0.043381	0.035249

This gives the correlation between the dependent and independent variables. We can visualize this by plotting heat map.

```
# Visualizing the correlation matrix by plotting heat map.
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),linewidths=.1,vmin=-1, vmax=1,fmt='.1g',linecolor="black",annot=True,annot_kws={
plt.yticks(rotation=0);
```



This heatmap shows the correlation matrix by visualizing the data. we can observe the relation between one feature to other.

- This heat map contains both positive and negative correlation.
- The features Number_of_stops, Duration Arrival_Hour and Airline are highly positively correlated with the target column compared to other features.
- The other features have very less correlation with the target column.
- From the map we can also observe there is no multicollinearity issue exists.

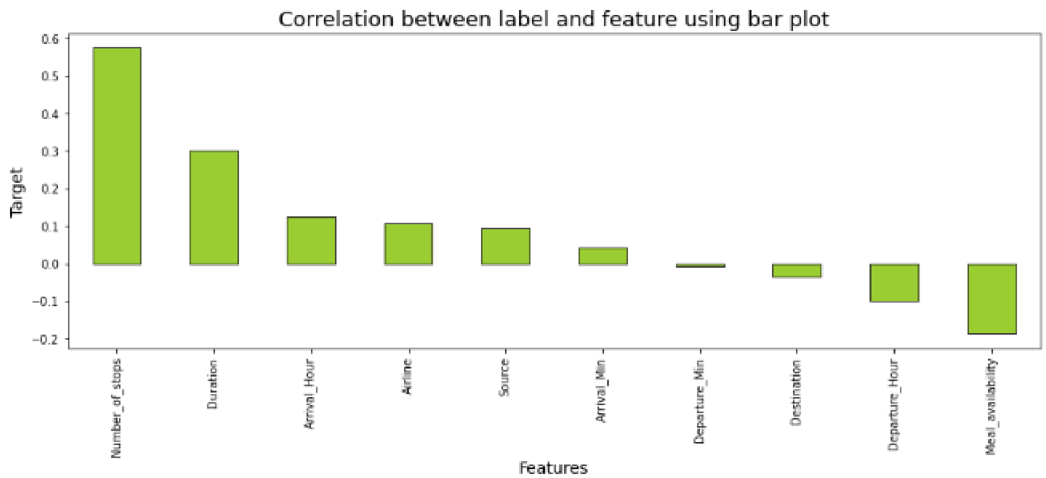
```
cor['Price'].sort_values(ascending=False)
```

```
Price              1.000000
Number_of_stops    0.575886
Duration           0.302224
Arrival_Hour       0.124571
Airline            0.106153
Source             0.093260
Arrival_Min        0.043381
Departure_Min     -0.006722
Destination       -0.034189
Departure_Hour    -0.100619
Meal_availability -0.187175
Name: Price, dtype: float64
```

Here we can notice the positive and negative correlation between features and label in the descending order.

Visualizing the correlation between label and features using bar plot

```
plt.figure(figsize=(15,5))
df.corr()['Price'].sort_values(ascending=False).drop(['Price']).plot(kind='bar',color='yellowgreen',edg
plt.xlabel('Features',fontsize=14)
plt.ylabel('Target',fontsize=14)
plt.title('Correlation between label and feature using bar plot',fontsize=18)
plt.show()
```



From the bar plot we can clearly observe the positive and negative correlation between the label and features. Here the column "Departure_Min" has less correlation with the label compared to other features, we can drop this column if necessary but for now let's keep it as it is.

Separating the feature and label into x and y

```
x = df.drop("Price", axis=1)
y = df["Price"]
```

We have separated both dependent and independent variables.

```
# Dimension of x and y
x.shape, y.shape
```

```
((5303, 10), (5303,))
```

After data cleaning and preprocessing we are left with 10 columns which we are using to train our machine learning model for predicting the ticket price of the flights.

Feature Scaling Using StandardScaler

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
X.head()
```

	Airline	Duration	Source	Destination	Meal_availability	Number_of_stops	Departure_Hour	Departure_Min	Arrival_Hour
0	-0.386354	-0.790402	1.354839	0.978349	-0.357852	-0.170556	-1.163362	-1.525801	-0.704815
1	-0.386354	-0.762444	1.354839	0.978349	-0.357852	-0.170556	0.555252	0.457853	1.079413
2	-0.386354	-0.440930	1.354839	0.978349	-0.357852	-0.170556	-0.647778	-1.525801	0.106198
3	-0.386354	-0.385014	1.354839	0.978349	-0.357852	-0.170556	-0.647778	-0.959043	0.268400
4	-0.386354	-0.119416	1.354839	0.978349	-0.357852	-0.170556	-1.335224	-0.108906	-0.056005

We have scaled the data using StandardScaler method to overcome with the issue of data biasness and displayed the data of independent variables after scaling.

Checking for multicolinearity issue using VIF:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif["Features"]=X.columns
vif
```

	vif_Features	Features
0	1.107968	Airline
1	1.252048	Duration
2	1.093615	Source
3	1.086621	Destination
4	1.132104	Meal_availability
5	1.334488	Number_of_stops
6	1.045793	Departure_Hour
7	1.017979	Departure_Min
8	1.036639	Arrival_Hour
9	1.034466	Arrival_Min

There is no multicolinearity issue in this dataset.

Building Machine Learning Models

```
# mporting necessary libraries

# Evaluation Metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.metrics import r2_score
from sklearn import metrics
```

Finding the Best Random State and Accuracy

```
from sklearn.ensemble import RandomForestRegressor
maxAccu=0
maxRS=0
for i in range(1,200):
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30, random_state =i)
    mod = RandomForestRegressor()
    mod.fit(X_train, y_train)
    pred = mod.predict(X_test)
    acc=r2_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.7393882195150385  on Random_state  126
```

```
Best accuracy is  0.7393882195150385  on Random_state  126
```

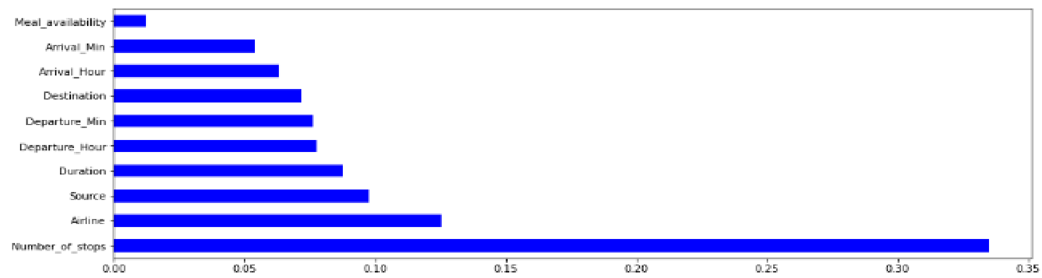
With the help of random state selection process we have found our random state to be 126 amongst 1-1000 with best accuracy as 73.93% using Random Forest Regressor.

Feature Importance

```
# Lets chkeck the feature importance using Random Forest Regressor

RFR = RandomForestRegressor()
RFR.fit(X_train, y_train)
importances = pd.DataFrame({'Features':X.columns, 'Importance':np.round(RFR.feature_importances_,3)})
importances = importances.sort_values('Importance', ascending=False).set_index('Features')
importances
```

Features	Importance
Number_of_stops	0.335
Airline	0.125
Source	0.098
Duration	0.088
Departure_Hour	0.077
Departure_Min	0.076
Destination	0.072
Arrival_Hour	0.063
Arrival_Min	0.054
Meal_availability	0.012

```
plt.figure(figsize=(15,5))
importances=pd.Series(RFR.feature_importances_,index=X.columns)
importances.nlargest(30).plot(kind='barh',color="b")
plt.show()
```



Here with the help of RandomForestRegressor we are able to list down the importance given to a column as per it's involvement in predicting our label. Here the column "Number_of_stops", "Airline" and "Source" contributing more for prediction which means these features are important for the predictions.

Creating new train test split

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.30,random_state=maxRS)
```

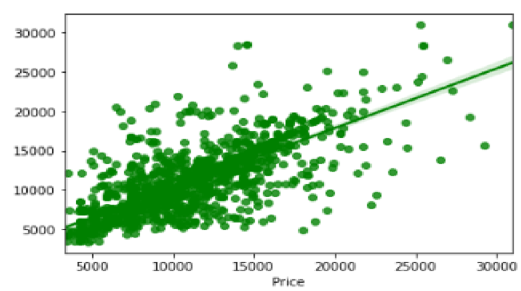
I am taking 30 percent of the complete dataset for training purpose and the remaining 70 percent will be used to train the machine learning models using the random state.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import BaggingRegressor
import xgboost as xgb
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from sklearn import metrics
```

i) Decision Tree Regressor

```
# Checking R2 score for Decision Tree Regressor
DTR=DecisionTreeRegressor()
DTR.fit(X_train,y_train)

# prediction
predDTR=DTR.predict(X_test)
R2_score = r2_score(y_test,predDTR)*100
print('R2_Score:',R2_score)
# Evaluation Metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, predDTR))
print('Mean Squared Error:',metrics.mean_squared_error(y_test, predDTR))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test, predDTR)))

# Visualizing the predicted values
sns.regplot(y_test,predDTR,color="g")
plt.show()
```

```
R2_Score: 50.82605956584663
Mean Absolute Error: 1672.539283469516
Mean Squared Error: 8522783.735700818
Root Mean Squared Error: 2919.380710990058
```

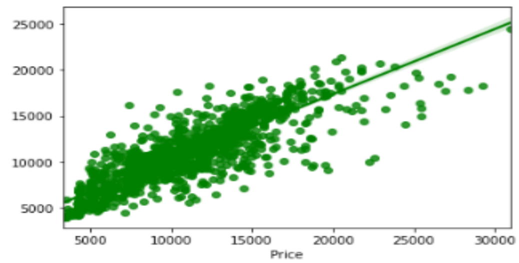


- Created Decision Tree Regressor model and checked for its evaluation metrics. The model is giving R2 score as 50.82%.
- From the graph we can observe how our model is mapping. In the graph we can observe the straight line which is our actual dataset and dots are the predictions that the model has given.

ii) Random Forest Regressor

```
# Checking R2 score for Random Forest Regressor
RFR=RandomForestRegressor()
RFR.fit(X_train,y_train)

# prediction
predRFR=RFR.predict(X_test)
R2_score = r2_score(y_test,predRFR)*100      # R squared score
print('R2_Score:',R2_score)
# Evaluation Metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, predRFR))
print('Mean Squared Error:',metrics.mean_squared_error(y_test, predRFR))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test, predRFR)))

# Visualizing the predicted values
sns.regplot(y_test,predRFR,color="g")
plt.show()
```

```
R2_Score: 74.11384314837284
Mean Absolute Error: 1346.886632196981
Mean Squared Error: 4486565.742891269
Root Mean Squared Error: 2118.151491959739
```

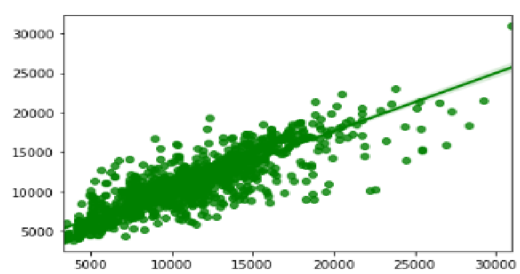


- Created Random Forest Regressor model and checked for it's evaluation metrics. The model is giving R2 score as 74.11%.
- From the graph we can observe how our model is mapping. In the graph we can observe the straight line which is our actual dataset and dots are the predictions that our model has given.

iii) Extra Trees Regressor

```
# Checking R2 score for Extra Trees Regressor
XT=ExtraTreesRegressor()
XT.fit(X_train,y_train)

# prediction
predXT=XT.predict(X_test)
R2_score = r2_score(y_test,predXT)*100      # R squared score
print('R2_Score:',R2_score)
# Evaluation Metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, predXT))
print('Mean Squared Error:',metrics.mean_squared_error(y_test, predXT))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test, predXT)))

# Visualizing the predicted values
sns.regplot(y_test,predXT,color="g")
plt.show()
```

```
R2_Score: 76.30615478986948
Mean Absolute Error: 1249.7518141630003
Mean Squared Error: 4106596.2339039836
Root Mean Squared Error: 2026.4738423932306
```

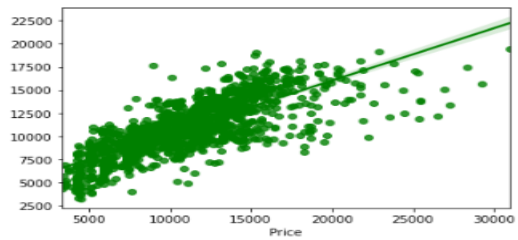


- Created Extra Trees Regressor model and checked for its evaluation metrics. The model is giving R2 score as 76.30%.
- From the graph we can observe how our model is mapping. In the graph we can observe the straight line which is our actual dataset and dots are the predictions that our model has given.

iv) GradientBoosting Regressor

```
# Checking R2 score for GradientBoosting Regressor
GB=GradientBoostingRegressor()
GB.fit(X_train,y_train)

# prediction
predGB=GB.predict(X_test)
R2_score = r2_score(y_test,predGB)*100      # R squared score
print('R2_Score:',R2_score)
# Evaluation Metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, predGB))
print('Mean Squared Error:',metrics.mean_squared_error(y_test, predGB))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test, predGB)))

# Visualizing the predicteed values
sns.regplot(y_test,predGB,color="g")
plt.show()
```

```
R2_Score: 61.510488965334396
Mean Absolute Error: 1823.8119960301044
Mean Squared Error: 6670967.909935666
Root Mean Squared Error: 2582.821695343228
```

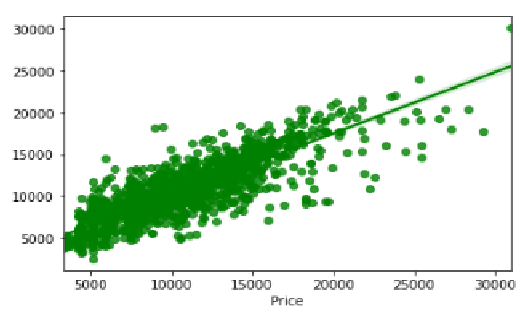


- Created GradientBoosting Regressor model and checked for its evaluation metrics. The model is giving R2 score as 61.51%.
- From the graph we can observe how our model is mapping. In the graph we can observe the straight line which is our actual dataset and the dots are the predictions that our model has given.

v) Extreme Gradient Boosting Regressor (XGB)

```
# Checking R2 score for XGB Regressor
from xgboost import XGBRegressor as xgb
XGB=xgb(verbosity=0)
XGB.fit(X_train,y_train)

# prediction
predXGB=XGB.predict(X_test)
R2_score = r2_score(y_test,predXGB)*100      # R squared score
print('R2_Score:',R2_score)
# Evaluation Metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, predXGB))
print('Mean Squared Error:',metrics.mean_squared_error(y_test, predXGB))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test, predXGB)))

# Visualizing the predicteed values
sns.regplot(y_test,predXGB,color="g")
plt.show()
```

```
R2_Score: 72.3563938283186
Mean Absolute Error: 1461.3120605161848
Mean Squared Error: 4791165.300076106
Root Mean Squared Error: 2188.873066232052
```

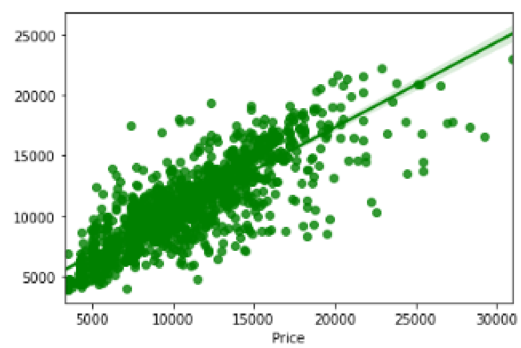


- Created XGB Regressor model and checked for its evaluation metrics. The model is giving R2 score as 72.35%.
- From the graph we can observe how our model is mapping. In the graph we can observe the straight line which is our actual dataset and the dots are the predictions that our model has given.

vi) Bagging Regressor

```
# Checking R2 score for BaggingRegressor
BR=BaggingRegressor()
BR.fit(X_train,y_train)

# prediction
predBR=BR.predict(X_test)
R2_score = r2_score(y_test,predBR)*100       # R squared score
print('R2_Score:',R2_score)
# Evaluation Metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, predBR))
print('Mean Squared Error:',metrics.mean_squared_error(y_test, predBR))
print("Root Mean Squared Error:",np.sqrt(metrics.mean_squared_error(y_test, predBR)))

# Visualizing the predicteed values
sns.regplot(y_test,predBR,color="g")
plt.show()
```

```
R2_Score: 70.48680259098631
Mean Absolute Error: 1445.7863073538654
Mean Squared Error: 5115201.194886711
Root Mean Squared Error: 2261.6810550753416
```



- Created Bagging Regressor model and checked for its evaluation metrics. The model is giving R2 score as 70.48%.
- From the graph we can observe how our model is mapping. In the graph we can observe the straight line which is our actual dataset and the dots are the predictions that our model has given.

Model Selection

From the above created models, Extra Trees Regressor algorithm has high R2 score and less RMSE value. So, we can conclude that "Extra Trees Regressor" as the best fitting model. Let's try to increase our model score by tuning the best model using different types of hyper parameters.

Testing of Identified Approaches (Algorithms)

Hyper Parameter Tuning

```
# Let's Use the GridSearchCV to find the best paarameters in XGBRegressor
from sklearn.model_selection import GridSearchCV

# Extra Trees Regressor
parameter = {'n_estimators':[10,100,1000],
             'criterion':['squared_error','mse','absolute_error','mae'],
             'min_samples_split': [1,2,3,4],
             'max_features':['auto','sqrt','log2'],
             'n_jobs':[-2,-1,1,2]}
```

I have used 5 Extra Trees Regressor parameters to be saved under the variable "parameter" that will be used in GridSearchCV for finding the best output.

```
GCV=GridSearchCV(ExtraTreesRegressor(),parameter,cv=5)
```

Assigning a variable to the GridSearchCV function after entering all the necessary inputs.

```
# Running GridSearchCV
GCV.fit(X_train,y_train)
```

```
GridSearchCV(cv=5, estimator=ExtraTreesRegressor(),
             param_grid={'criterion': ['squared_error', 'mse', 'absolute_error',
                                       'mae'],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [1, 2, 3, 4],
                         'n_estimators': [10, 100, 1000],
                         'n_jobs': [-2, -1, 1, 2]})
```

Now we use our training data set to make the GridSearchCV aware of all the hyper parameters that needs to be applied on our best model.

```
# Finding best parameters
GCV.best_params_
```

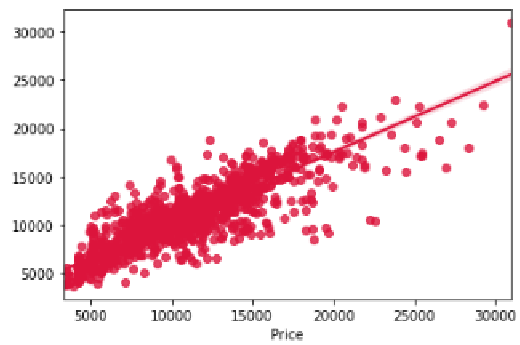
```
{'criterion': 'mae',
 'max_features': 'auto',
 'min_samples_split': 2,
 'n_estimators': 1000,
 'n_jobs': 2}
```

This gives us the list of best parameters which will be used further in our final model creation.

Run and Evaluate selected models

```
# Creating final model
Flight_price_model = ExtraTreesRegressor(criterion='mae',max_features='auto',min_samples_split=2,n_esti

# Prediction
Flight_price_model.fit(X_train, y_train)
pred = Flight_price_model.predict(X_test)
print('R2_Score:',r2_score(y_test,pred)*100)

# Metric Evaluation
print('Mean absolute error:',metrics.mean_absolute_error(y_test, pred))
print('Mean squared error:',metrics.mean_squared_error(y_test, pred))
print('Root Mean Squared error:',np.sqrt(metrics.mean_squared_error(y_test, pred)))

# Visualizing the predicted values
sns.regplot(y_test,pred,color="crimson")
plt.show()
```

```
R2_Score: 77.00369027273109
Mean absolute error: 1233.434366121936
Mean squared error: 3985700.002771813
Root Mean Squared error: 1996.4217998138101
```



- We have successfully incorporated the hyper parameter tuning using best parameters of Extra Trees Regressor and the R2 score of the model has been increased after hyperparameter tuning and received the R2 score as 77% which is very good.
- From the graph we can observe how our final model is mapping. In the graph we can observe the best fit line which is our actual dataset and the dots are the predictions that our best final model has given.

Saving the Final model

```
# Saving the model using joblib library
import joblib
joblib.dump(Flight_price_model,"Flight_Ticket_Price_Prediction.pkl")
```

```
['Flight_Ticket_Price_Prediction.pkl']
```

I am using the joblib option to save the final regression model in the form of .pkl.

Loading the saved model and predicting Flight Ticket Price

```
# Loading the saved model
Model=joblib.load("Flight_Ticket_Price_Prediction.pkl")

#Prediction
prediction = Model.predict(X_test)
prediction
```

```
array([9599.109, 7015.847, 9339.41 , ..., 9685.993, 9389.677, 7616.363])
```

These are the predicted price of the flight tickets.

Creating DataFrame for the predicted values

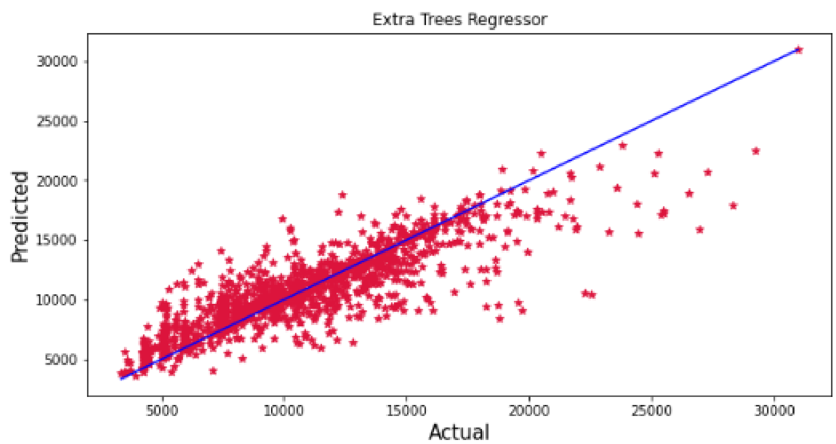
```
Predicted_Flight_Ticket_Price = pd.DataFrame([Model.predict(X_test)[:],y_test[:]],index=["Predicted","A
Predicted_Flight_Ticket_Price
```

	0	1	2	3	4	5	6	7	8	9	...	1581
Predicted	9599.109	7015.847	9339.41	12306.887	12978.084	11093.491	10624.0	12356.486	13426.982	4737.756	...	6330.616
Actual	8883.000	5943.000	9141.00	11675.000	13830.000	11945.000	10624.0	8516.000	13104.000	4275.000	...	5882.000

2 rows × 1591 columns

Using regression model, we have got the predicted price of the flight tickets. From the above output we can observe that predicted values are almost near to the actual values.

```
# Visualizing actual and predicted values
plt.figure(figsize=(10,5))
plt.scatter(y_test, prediction, c='crimson',marker="*")
p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('Actual', fontsize=15)
plt.ylabel('Predicted', fontsize=15)
plt.title("Extra Trees Regressor")
plt.show()
```



The graph shows how our final model is mapping. The plot gives the linear relation between predicted and actual price of the flight tickets. The blue line is the best fitting line which gives the actual values/data and red dots gives the predicted values/data.

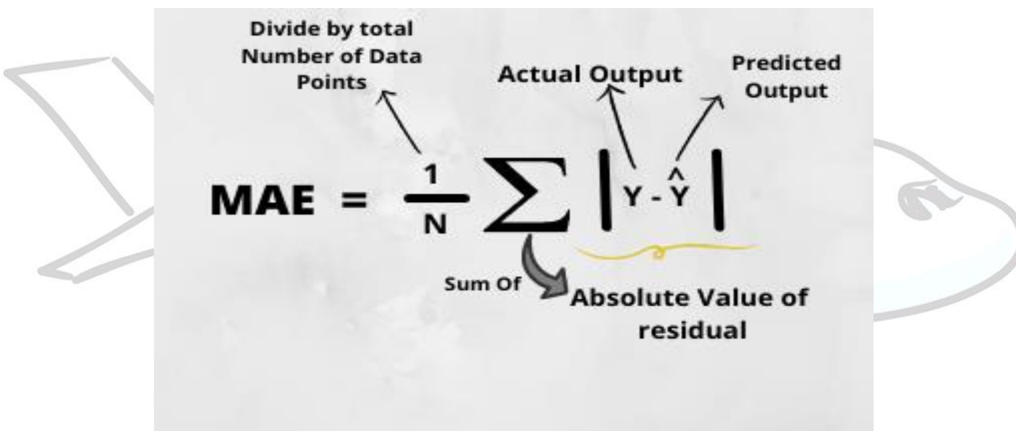
```
# Saving the predicted car price values in csv file
Predicted_Flight_Ticket_Price.to_csv("Predicted_Flight_Ticket_Price.csv",index=False)
```

We have saved the predicted flight ticket price values in csv file

Key Metrics for success in solving problem under consideration

The essential step in any machine learning model is to evaluate the accuracy and determine the metrics error of the model. I have used Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R2 Score metrics for my model evaluation:

- **Mean Absolute Error (MAE):** MAE is a popular error metric for regression problems which gives magnitude of absolute difference between actual and predicted values. The MAE can be calculated as follows:



- **Mean Squared Error (MSE):** MSE is a most used and very simple metric with a little bit of change in mean absolute error. Mean squared error states that finding the squared difference between actual and predicted value. We perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

❖ **Root Mean Squared Error (RMSE):** RMSE is an extension of the mean squared error. The square root of the error is calculated, which means that the units of the RMSE are the same as the original units of the target value that is being predicted.

$$\text{RMSE} = \sqrt{\text{MSE}}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)}$$

❖ **R2 Score:** I have used R2 score which gives the accurate value for the models used. On the basis of R2 score I have created final model.

Interpretation of the Results

Visualizations: In univariate analysis I have used count plots and pie plots to visualize the counts in categorical variables and distribution plot to visualize the numerical variables. In bivariate analysis I have used bar plots, strip plots, line plots, reg plots, box plots, and box plots to check the relation between label and the features. Used pair plot to check the pairwise relation between the features. The heat map and bar plot helped me to understand the correlation between dependent and independent features. Detected outliers and skewness with the help of box plots and distribution plots respectively. And I found some of the features skewed to right as well as to left. I got to know the count of each column using bar plots.

Pre-processing: The dataset should be cleaned and scaled to build the ML models to get good predictions. I have performed few processing steps which I have already mentioned in the pre-processing steps where all the important features are present in the dataset and ready for model building.

Model building: After cleaning and processing data, I performed train test split to build the model. I have built multiple regression models to get the accurate R2 score, and evaluation metrics like MAE, MSE and RMSE. I got Extra Trees Regressor as the best model which gives 76.30%R2score.After tuning the best model, the R2 score of Extra Trees Regressor has been increased to77% and also got low evaluation metrics. Finally, I saved my final model and got the good predictions results for price of flight tickets.

Conclusion

Key Findings and Conclusions of the Study

The case study aims to give an idea of applying Machine Learning algorithms to predict the price of the flight tickets. After the completion of this project, we got an insight of how to collect data, pre-processing the data, analyze the data, cleaning the data and building a model. In this study, we have used multiple machine learning models to predict the flight ticket price. We have gone through the data analysis by performing feature engineering, finding the relation between features and label through visualizations. And got the important feature and we used these features to predict the car price by building ML models. Performed hyper parameter tuning on the best model and the best model's R2 score increased and was giving R2 score as 77%. We have also got good prediction results of ticket price.

Findings:

- Flight ticket prices change during the morning and evening time of the day. From the distribution plots we came to know that the prices of the flight tickets are going up and down, they are not fixed at a time. Also, from this graph we found prices are increasing in large amounts.
- Some flights are departing in the early morning 3 AM having most expensive ticket prices compared to late morning flights. As the time goes the flight ticket fares increased and midnight flight fares are very less (say after 10 PM). Also, from categorical and numerical plots we found that the prices are tending to go up as the time is approaching from morning to evening.
- From the categorical plots (bar and box) we came to know that early morning and late-night flights are cheaper compared to working hours.
- From the categorical plots we found that the flight ticket prices increaseas the person get near to departure time. That is last minute flights are very expensive.
- From the bar plot we got to know that both Indigo and Spicejet airways almost having same ticket fares.
- Not all flights are expensive during morning, only few flights departing in the early morning 3 AM are expensive. Apart from this the flight ticket fares are less compared to other timing flight fares.

Learning Outcomes of the Study in respect of Data Science

While working on this project I learned many things about the features of flights and about the flight ticket selling web platforms and got the idea that how the machine learning models have helped to predict the price of flight tickets. I found that the project was quite interesting as the dataset contains several types of data. I used several types of plotting to visualize the relation between target and features. This graphical representation helped me to understand which features are important and how these features describe price of tickets. Data cleaning was one of the important and crucial things in this project where I dealt with features having string values, features extraction and selection. Finally got Extra Trees Regressor as best model.

The challenges I faced while working on this project was when I was scrapping the real time data from yatra website, it took so much time to gather data. Finally, our aim was achieved by predicting the flight ticket price and built flight price evaluation model that could help the buyers to understand the future flight ticket prices.

Limitations of this work and Scope for Future Work

Limitations: The main limitation of this study is the low number of records that have been used. In the dataset our data is not properly distributed in some of the columns many of the values in the columns are having string values which I had taken care. Due to some reasons our models may not make the right patterns and the performance of the model also reduces. So these issues need to be taken care.

Future work: The greatest shortcoming of this work is the shortage of data. Anyone wishing to expand upon it should seek alternative sources of historical data manually over a period of time. Additionally, a more varied set of flights should be explored, since it is entirely plausible that airlines vary their pricing strategy according to the characteristics of the flight (for example, fares for regional flights out of small airports may behave differently than the major, well flown routes we considered here). Finally, it would be interesting to compare our system's accuracy against that of the commercial systems available today (preferably over a period of time).