Flight Fare Prediction Project

Submitted by:

Aishwary Shukla

ACKNOWLEDGMENT

The aim of this project was to predict Ticket Fares of of flights using Machine Learning. A data set was provided. From this dataset models were build in Jupyter notebook using Python and ML libraries. After a preliminary study of the available algorithms and data review, it became apparent that the problem fell under regression category. Thus, we have used regression algorithms; linear regression, decision tree, Random Forest, KNeighbours Regressor, Support Vector Machine, AdaBoostRegressor respectively. The major discovery is that the machine learning approach should be suitable for these types of problems due to many aspects. Python programming language and its libraries namely Pandas, Numpy, Matplotlib, Seaborn etc are also a good choice for a first step, not the least because of the easily grasped user interface, as well as the wide availability of algorithms within machine learning. Advanced methods such as hyper parameter tuning of best models is also available.

References:

- https://skyscanner.com/

Special thanks to:

Mr. Shankar Gaud Tegimanni, DataTrained

Mr. Shwetank Mishra, FlipRobo

Mr. Prateeek Rajvanshi, Clevered Institute

And all colleagues, mentors, trainers from DataTrained and FlipRobo for training me and giving me the opportunity to be an intern and expand my knowledge in the field of Data Science and Artificial Intelligence.

INTRODUCTION

Problem Statement: Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on -

1. Time of purchase patterns (making sure last-minute purchases are expensive)

2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases)

With Data science and machine learning we will to **make a model that predicts fare of flights from Delhi to Dubai that will help the passengers understand the factors affecting flight fares and select the best deal. Flight companies can also use the conclusions from this analysis to understand customer mindset and improve customer experience and provide best deals.** This will also increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in flights. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for many travel search websites.

The data is provided in the CSV file below.

We are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not.

We will investigate:

- Which variables are important to predict the fares?
- How do these variables describe the fare of the flights?

Business Goal: We will be analyzing the flight fare using data that we have scrapped and analyse it using essential exploratory data analysis techniques then will draw some predictions about the price of the flight based on some features such as what type of airline it is, what is the arrival time, what is the departure time, what is the duration of the flight, source, destination and more.

Conceptual Background of the Domain Problem

It's no secret that flight prices fluctuate constantly. Even great deals get dropped just in a few hours and the prices even get doubled.

So what affects flight prices to cause such dramatic variation? Most airlines use a pricing system called dynamic pricing. This is why the price moves up and down at any given time rather.

Several factors go into the equation, some of which might seem obvious but others that might surprise you. Here, we'll take a look at these price-influencing factors. We'll also look at some of the secret weapons of frequent travelers to find the best prices, such as Skyscanner, and more.

Review of Literature

Steps followed to conduct the research of finding important features that positively affect the fares of flights.

- Exploratory data analysis
- Feature Engineering
- Relation between different features/amenities of house and target column i.e. sales price.
- Grouped Data Analysis to get more insights.
- Correlation and identification of multicollinearity problem.
- Identifying and Selecting best features that affect fare price.
- Data pre-processing.
- Model Initialization and evaluation.
- Model Validation
- Model Testing and Pipelining.

Motivation for the Problem Undertaken

My objective behind making this project is to implement techniques and methods I have learned and practiced during my PG programme in Data Science. Further I continuously strive to improve my skills, adapt new techniques and methods in data analysis and modelling. Travel and tourism interests me thus working on this project will give me domain knowledge and technical expertise in deploying ml models for solving real world problems.

Data collection

- Scrapping Data from SkyScanner website.
    1. Scrapping different features of flight such as airline name, date of journey, source, destination, route, departure time, arrival time, duration, total stops and the target variable price.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem
- Data Sources and their formats

  Data sources: Skyscanner site.

  Origins: Flight data belongs to flights departing from Delhi and arriving at Dubai.

  Formats: Shared ahead.

- Data Preprocessing:
    1. **Data Cleaning**
    2. **Feature Engineering**
- Data Inputs- Logic- Output Relationships
  (Features and label after Feature Engineering)

Departure Time	float64
Arrival Time	float64
Extra	object
From	object
To	object
Prices	float64
total_time_taken_in_hours	float64
Day of flight	object

date of flight	int64
month of flight	object
stop_details	object
first_stop	object
second_stop	object
third_stop	object
fourth_stop	object
airline1	object
airline2	object
airline3	object
Price Range	object

Feature Description:

- From column contains data about from where the flights are leaving to Dubai so hair the data is perfectly from Delhi to Dubai so all the flights are living from Delhi only.
- To column contains data for Dubai international airport and Abu dhabi international airport.
- Since it is one week data the day of flights range from Monday to Sunday respectively and the month is January.
- The stop details column contain whether the flight from Delhi to final destination is direct or there are one stop to or three or four stops in between them.
- First stop second stop third stop and 4th stop column contain detail of stops such as different Indian states France etc.
- Airlines one airline to airline 3 columns data contain airlines details that are flying between Delhi and Dubai. This data is regarding connecting flights between different stops.
- We have distributed price column into 3 rangers namely average priced lower than average and expensive based on the fare price.

Relationship between features and label:

Amongst all features, total_time_in_hours column is highly correlated with fare price. Flights taking less time are more expensive as compared to flights taking more time.

Other factors such as Arrival time, quality of airlines, date of flight, also affect fares.

Hardware and software requirements:

- Python and its libraries relevant to machine learning
- Computer with adequate specification.

Tools, libraries and packages used:

Libraries imported For Data Scraping:

from selenium import webdriver

from selenium.webdriver.common.by import By

import time

from time import sleep

Libraries imported For Data analysis and Machine Learning:

NumPy

Pandas

Matplotlib.pyplot and Seaborn

%matplotlib inline

statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.feature_selection import SelectKBest, f_classif

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

Importing algorithms

from sklearn.linear_model import LinearRegression

from sklearn.linear_model import Ridge,Lasso,RidgeCV,LassoCV

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import AdaBoostRegressor

Importing metrics

from sklearn import metrics

from sklearn.metrics import r2_score,mean_squared_error

from sklearn.metrics import mean_absolute_error

Removing warnings

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

Spliting data and hyperparameter tuning

from sklearn.model_selection import train_test_split,GridSearchCV

Pipelining

from sklearn.pipeline import Pipeline

Model/s Development and Evaluation

- Problem-solving approach taken into consideration:
  1. Exploratory data analysis.
  2. Treatment of null values.
  3. Feature Engineering
  4. Analysis using Plots:
     - Univariate analysis
     - Bivariate analysis
     - Multivariate analysis
  5. Grouped Data Analysis
  6. Corelation and multicollinearity analysis
  7. Best Feature selection and PCA
  8. Model preprocessing, instantiating, training, testing, Hyperparameter tuning
  9. Best model selection and Pipelining

Exploratory Data Analysis:

Functions and methods used:

1. Reading csv files using pd.read_csv function.

```
df1 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub1.cs
df2 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub2.cs
df3 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub3.cs
df4 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub4.cs
df5 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub5.cs
df6 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub6.cs
df7 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub7.cs
df8 = pd.read_csv(r'C:\Users\Lenovo\OneDrive\Desktop\Filp Robo worksheets and projects\flight data\dubai flight details\dtdub8.cs

list_a = [df1,df2,df3,df4,df5,df6,df7,df8]
df = pd.concat(list_a,ignore_index=True)
```

2. Inspecting dataframe and renaming columns.

	Unnamed: 0	dept	arr	ex	From	To	Total time taken	stops	prices	airlines	Date
0	0	16:15	18:35	Nill	DEL	DXB	3h 50	Direct\n	₹ 31,780	Emirates	sunday, 8th Jan
1	1	13:20	16:00	Nill	DEL	DXB	4h 10	Direct\n	₹ 12,788	Air India Express	sunday, 8th Jan
2	2	07:30	10:10	Nill	DEL	DXB	4h 10	Direct\n	₹ 13,380	SpiceJet	sunday, 8th Jan
3	3	08:40	11:15	Nill	DEL	DXB	4h 05	Direct\n	₹ 14,118	IndiGo	sunday, 8th Jan
4	4	06:05	08:35	Nill	DEL	DXB	4h	Direct\n	₹ 14,451	Air India	sunday, 8th Jan
5	5	16:15	18:35	Nill	DEL	DXB	3h 50	Direct\n	₹ 31,780	Emirates	sunday, 8th Jan
6	6	19:25	22:05	Nill	DEL	DXB	4h 10	Direct\n	₹ 13,380	SpiceJet	sunday, 8th Jan
7	7	17:15	19:50	Nill	DEL	DXB	4h 05	Direct\n	₹ 14,717	IndiGo	sunday, 8th Jan
8	8	20:15	22:45	Nill	DEL	DXB	4h	Direct\n	₹ 14,451	Air India	sunday, 8th Jan
9	9	16:30	19:00	Nill	DEL	DXB	4h	Direct\n	₹ 15,401	Air India	sunday, 8th Jan
10	10	16:15	18:35	Nill	DEL	DXB	3h 50	Direct\n	₹ 31,780	Emirates	sunday, 8th Jan

```
df = df.rename(columns = {'dept':'Departure Time','arr':'Arrival Time','ex':'Extra','stops':'Stops','prices':'Prices','airlines':

df.drop(columns=['Unnamed: 0'],inplace=True)
```

3. Inspecting rows and columns in dataset:

```
]: df.head(5)
```

]:

	Departure Time	Arrival Time	Extra	From	To	Total time taken	Stops	Prices	Airlines	Date
0	16:15	18:35	Nill	DEL	DXB	3h 50	Direct\n	₹ 31,780	Emirates	sunday, 8th Jan
1	13:20	16:00	Nill	DEL	DXB	4h 10	Direct\n	₹ 12,788	Air India Express	sunday, 8th Jan
2	07:30	10:10	Nill	DEL	DXB	4h 10	Direct\n	₹ 13,380	SpiceJet	sunday, 8th Jan
3	08:40	11:15	Nill	DEL	DXB	4h 05	Direct\n	₹ 14,118	IndiGo	sunday, 8th Jan
4	06:05	08:35	Nill	DEL	DXB	4h	Direct\n	₹ 14,451	Air India	sunday, 8th Jan

4. Feature Engineering:

```
# Feature Engineering
```

```
# Prices column:

df['Prices'] = df['Prices'].str.lstrip('₹ ')

pr = []
for i in df['Prices']:
    pr.append((i.translate({ord(i): None for i in ','})))

df['Prices'] = pr

df['Prices'] = df['Prices'].astype('float64')
```

```
# Total time taken by flight

t = []
for i in df['Total time taken']:
    t.append(i.split(' '))
t

time_taken_in_hours = []
for i in t:
    time_taken_in_hours.append(i[0].rstrip('h'))


df['time_taken_in_hours'] = time_taken_in_hours
df['time_taken_in_hours'] = df['time_taken_in_hours'].astype('int64')


extra_time_taken_in_minutes = []
for i in t:
    if len(i) == 1:
        extra_time_taken_in_minutes.append('0')
    else:
        extra_time_taken_in_minutes.append(i[1])

df['extra_time_taken_in_minutes'] = extra_time_taken_in_minutes
df['extra_time_taken_in_minutes'] = df['extra_time_taken_in_minutes'].astype('int64')

total_time_taken_in_minutes = df['time_taken_in_hours']*60 + df['extra_time_taken_in_minutes']

df['total_time_taken_in_hours'] = total_time_taken_in_minutes/60
```

```
# Date column:

day = []
for i in df['Date']:
    day.append(i.split(',')[0])
df['Day of flight'] = day

date = []
month = []
for i in df['Date']:
    date.append(i.split(' ')[1].rstrip('th'))
    month.append(i.split(' ')[2])

df['date of flight'] = date
df['date of flight'] = df['date of flight'].astype('int64')

df['date of flight'] = df['date of flight'].astype('int64')
df['month of flight'] = month
```

```
# Departure column:

dept= []
for i in df['Departure Time']:
    dept.append(i.replace(':','.'))
df['Departure Time'] = dept
df['Departure Time'] = df['Departure Time'].astype('float64')
art= []
for i in df['Arrival Time']:
    art.append(i.replace(':','.'))
df['Arrival Time'] = art
df['Arrival Time'] = df['Arrival Time'].astype('float64')
```

```
# Stop column feature engineering

stopz = []
stop_details = []
first_stop = []
second_stop = []
third_stop = []
fourth_stop = []
fifth_stop = []


for i in df['Stops']:
    stopz.append(i.split('\n'))

for i in stopz:
    if ',' in i:
        i.remove(',')

for i in stopz:
    if ' ' in i:
        i.remove(' ')

for i in stopz:
    stop_details.append(i[0])

for i in stopz:
    if len(i)>=2:
        first_stop.append(i[1])
    else:
        first_stop.append('NA')
```

```
        first_stop.append(i[1])
    else:
        first_stop.append('NA')

for i in stopz:
    if len(i)>=3:
        second_stop.append(i[2])
    else:
        second_stop.append('NA')


for i in stopz:
    if len(i)>=5:
        third_stop.append(i[4])
    else:
        third_stop.append('NA')

for i in stopz:
    if len(i)>=7:
        fourth_stop.append(i[6])
    else:
        fourth_stop.append('NA')



df['stop_details'] = stop_details
df['first_stop'] = first_stop
df['second_stop'] = second_stop
df['third_stop'] = third_stop
df['fourth_stop'] = fourth_stop
```

```
# Airlines

airlinez = []
for i in df['Airlines']:
    if '+' not in i:
        airlinez.append(i)
    elif '+' in i:
        airlinez.append(i.split('+'))

df['Airlines'] = airlinez

airline1 = []
airline2 = []
airline3 = []


for i in airlinez:
    if len(i)==2 or len(i)==3:
        airline1.append(i[0])
    else:
        airline1.append(i)

for i in airlinez:
    if len(i)==2 or len(i)==3:
        airline2.append(i[1])
    else:
        airline2.append('Na')

for i in airlinez:
    if len(i)==3:
        airline3.append(i[2])
    else:
        airline3.append('Na')

df['airline1'] = airline1
df['airline2'] = airline2
df['airline3'] = airline3

df.drop(columns = ['Airlines'],axis=1,inplace=True)
```

```
# Price range

prRange = []
for i in df['Prices']:
    if i > 12187 and i < 23554:
        prRange.append('Lower than average')
    elif i > 23554 and i < 55665:
        prRange.append('Average priced')
    else:
        prRange.append('Expensive')

df['Price Range'] = prRange
```

```
print('First 10 rows')
df.head(10)
```

First 10 rows

Departure Time	Arrival Time	Extra	From	To	Prices	total_time_taken_in_hours	Day of flight	date of flight	month of flight	stop_details	first_stop	second_stop	third_stop	fourth_stop
16.15	18.35	Nill	DEL	DXB	31780.0	3.833333	sunday	8	Jan	Direct	NA	NA	NA	NA
13.20	16.00	Nill	DEL	DXB	12788.0	4.166667	sunday	8	Jan	Direct	NA	NA	NA	NA
7.30	10.10	Nill	DEL	DXB	13380.0	4.166667	sunday	8	Jan	Direct	NA	NA	NA	NA
8.40	11.15	Nill	DEL	DXB	14118.0	4.083333	sunday	8	Jan	Direct	NA	NA	NA	NA
6.05	8.35	Nill	DEL	DXB	14451.0	4.000000	sunday	8	Jan	Direct	NA	NA	NA	NA
16.15	18.35	Nill	DEL	DXB	31780.0	3.833333	sunday	8	Jan	Direct	NA	NA	NA	NA
19.25	22.05	Nill	DEL	DXB	13380.0	4.166667	sunday	8	Jan	Direct	NA	NA	NA	NA
17.15	19.50	Nill	DEL	DXB	14717.0	4.083333	sunday	8	Jan	Direct	NA	NA	NA	NA
20.15	22.45	Nill	DEL	DXB	14451.0	4.000000	sunday	8	Jan	Direct	NA	NA	NA	NA
16.30	19.00	Nill	DEL	DXB	15401.0	4.000000	sunday	8	Jan	Direct	NA	NA	NA	NA

5. Inspecting datatypes of all columns & Shape of dataframe:

```
df.dtypes
```

```
Departure Time                float64
Arrival Time                  float64
Extra                          object
From                           object
To                             object
Prices                        float64
total_time_taken_in_hours     float64
Day of flight                  object
date of flight                  int64
month of flight                object
stop_details                   object
first_stop                     object
second_stop                    object
third_stop                     object
fourth_stop                    object
airline1                       object
airline2                       object
airline3                       object
Price Range                    object
dtype: object
```

```
df.shape
```

```
(1703, 19)
```

6. Null Values in columns:

```
null_data = df.isnull().sum()
null_data.sort_values(ascending=0)
```

```
Departure Time                0
stop_details                  0
airline3                      0
airline2                      0
airline1                      0
fourth_stop                   0
third_stop                    0
second_stop                   0
first_stop                    0
month of flight               0
Arrival Time                  0
date of flight                0
Day of flight                 0
total_time_taken_in_hours     0
Prices                        0
To                            0
From                          0
Extra                         0
Price Range                   0
dtype: int64
```

7. Data description: of numerical type columns and categorical columns:

```
desc = df.describe().T
desc['range']=desc['max']-desc['min']
desc
```

	count	mean	std	min	25%	50%	75%	max	range
Departure Time	1703.0	13.330417	6.195079	0.100000	8.100000	15.100000	19.100000	23.500000	23.400000
Arrival Time	1703.0	14.383147	6.566746	0.150000	10.000000	17.200000	18.400000	23.590000	23.440000
Prices	1703.0	53552.631826	63203.389509	12187.000000	23554.000000	31780.000000	55665.500000	819876.000000	807689.000000
total_time_taken_in_hours	1703.0	13.302701	8.857707	3.833333	7.166667	10.916667	18.166667	56.416667	52.583333
date of flight	1703.0	12.146213	2.748634	8.000000	9.000000	12.000000	15.000000	16.000000	8.000000

```
df.describe(include='object').T
```

	count	unique	top	freq
Extra	1703	3	Nill	963
From	1703	1	DEL	1703
To	1703	2	DXB	1666
Day of flight	1703	8	Mon	236
month of flight	1703	1	Jan	1703
stop_details	1703	5	1 stop	965
first_stop	1703	42	NA	389
second_stop	1703	22	NA	1354
third_stop	1703	9	NA	1620
fourth_stop	1703	4	NA	1690
airline1	1703	40	Emirates	320
airline2	1703	17	Na	1254
airline3	1703	2	Na	1694
Price Range	1703	3	Average priced	851

8. Selecting categorical and numerical columns:

```
cont_data = df.select_dtypes(include=['int64','float64'])

cat_data= df.select_dtypes(include=['object'])

cont_columns = cont_data.columns

cat_columns = cat_data.columns
```

```
cont_columns
```

```
Index(['Departure Time', 'Arrival Time', 'Prices', 'total_time_taken_in_hours',
       'date of flight'],
      dtype='object')
```

```
cat_columns
```

```
Index(['Extra', 'From', 'To', 'Day of flight', 'month of flight',
       'stop_details', 'first_stop', 'second_stop', 'third_stop',
       'fourth_stop', 'airline1', 'airline2', 'airline3', 'Price Range'],
      dtype='object')
```

9. Most frequent values in dataset columns:

```
from scipy import stats

for i in cat_columns:
    print('For',i,', most frequent value is: ',stats.mode(df[i]),'\n')
```

```
For Extra , most frequent value is:  ModeResult(mode=array(['Nill'], dtype=object), count=array([963]))

For From , most frequent value is:  ModeResult(mode=array(['DEL'], dtype=object), count=array([1703]))

For To , most frequent value is:  ModeResult(mode=array(['DXB'], dtype=object), count=array([1666]))

For Day of flight , most frequent value is:  ModeResult(mode=array(['Mon'], dtype=object), count=array([236]))

For month of flight , most frequent value is:  ModeResult(mode=array(['Jan'], dtype=object), count=array([1703]))

For stop_details , most frequent value is:  ModeResult(mode=array(['1 stop'], dtype=object), count=array([965]))

For first_stop , most frequent value is:  ModeResult(mode=array(['NA'], dtype=object), count=array([389]))

For second_stop , most frequent value is:  ModeResult(mode=array(['NA'], dtype=object), count=array([1354]))

For third_stop , most frequent value is:  ModeResult(mode=array(['NA'], dtype=object), count=array([1620]))

For fourth_stop , most frequent value is:  ModeResult(mode=array(['NA'], dtype=object), count=array([1690]))

For airline1 , most frequent value is:  ModeResult(mode=array([' Emirates'], dtype=object), count=array([320]))

For airline2 , most frequent value is:  ModeResult(mode=array(['Na'], dtype=object), count=array([1254]))

For airline3 , most frequent value is:  ModeResult(mode=array(['Na'], dtype=object), count=array([1694]))

For Price Range , most frequent value is:  ModeResult(mode=array(['Average priced'], dtype=object), count=array([851]))
```

10.Unique values and value counts:

```
for i in cat_columns:
    print('For column',i,'unique values are: ',df[i].unique())
    print('For column',i,'count of unique values are: ',df[i].nunique(),'\n\n')
```

```
For column Extra unique values are:  ['Nill' '+1' '+2']
For column Extra count of unique values are:  3


For column From unique values are:  ['DEL']
For column From count of unique values are:  1


For column To unique values are:  ['DXB' 'XNB']
For column To count of unique values are:  2


For column Day of flight unique values are:  ['sunday' 'Monday' 'Tuesday' 'Thursday' 'Fri' 'Sat' 'Sun' 'Mon']
For column Day of flight count of unique values are:  8


For column month of flight unique values are:  ['Jan']
For column month of flight count of unique values are:  1


For column stop_details unique values are:  ['Direct' '1 stop' '2 stops' '3 stops' '4 stops']
For column stop_details count of unique values are:  5


For column first_stop unique values are:  ['NA' ' BOM' ' IDR' ' AMD' ' BAH' ' LKO' ' CCJ' ' MAA' ' MCT' ' HYD'
 ' KWI' ' AUH' ' BLR' ' DED' ' ALA' ' JED' ' CCU' ' DOH' ' COK' ' GOI'
 ' PNQ' ' CMB' ' RUH' ' JAI' ' IXC' ' CNN' ' DMM' ' ADD' ' DAC' ' LHR'
 ' MUC' ' ZRH' ' IXL' ' TAS' ' IST' ' SIN' ' ATQ' ' UDR' ' FRA' ' HKG'
 ' WAW' ' BHO']
For column first_stop count of unique values are:  42


For column second_stop unique values are:  ['NA' 'BOM' 'AMD' 'LKO' 'CCJ' 'NQZ' 'MCT' 'PNQ' 'CMB' 'JAI' 'HYD' 'AUH'
 'IST' 'JED' 'FRA' 'IXC' 'DOH' 'CAI' 'RUH' 'CGP' 'MUC' 'MAA']
For column second_stop count of unique values are:  22
```

```
For column third_stop unique values are:  ['NA' 'BOM' 'JAI' 'PNQ' 'AMD' 'IXC' 'HYD' 'MAA' 'LKO']
For column third_stop count of unique values are:  9


For column fourth_stop unique values are:  ['NA' 'BOM' 'JAI' 'MAA']
For column fourth_stop count of unique values are:  4


For column airline1 unique values are:  [' Emirates' ' Air India Express' ' SpiceJet' ' IndiGo' ' Air India'
 ' flydubai' ' Gulf Air' ' Go First ' ' Vistara ' ' IndiGo ' ' Vistara'
 ' SpiceJet ' ' Oman Air' ' Kuwait Airways' ' Etihad Airways'
 ' Air Astana' ' Saudia' ' Qatar Airways' ' SriLankan Airlines' ' Flynas'
 ' Qatar Airways ' ' Air India ' ' Ethiopian Airlines' ' Oman Air '
 ' Biman Bangladesh Airlines' ' British Airways' ' Lufthansa' ' SWISS'
 ' Uzbekistan Airways' ' Turkish Airlines' ' EgyptAir'
 ' Ethiopian Airlines ' ' Singapore Airlines' ' Jazeera Airways'
 ' Turkish Airlines ' ' Cathay Pacific' ' AirAsia India ' ' Lufthansa '
 ' LOT' ' Biman Bangladesh Airlines ']
For column airline1 count of unique values are:  40


For column airline2 unique values are:  ['Na' ' SpiceJet' ' Air India Express' ' IndiGo' ' Emirates' ' flydubai'
 ' Oman Air' ' SriLankan Airlines' ' Gulf Air' ' Etihad Airways'
 ' Turkish Airlines' ' Air India ' ' Vistara ' ' Qatar Airways' ' Flynas'
 ' Cathay Pacific' ' Air India']
For column airline2 count of unique values are:  17


For column airline3 unique values are:  ['Na' ' Emirates']
For column airline3 count of unique values are:  2
```

```
: for i in cat_columns:
      print('For column --',i,'-- value counts are: \n',df[i].value_counts(),'\n\n')
```

```
For column -- Extra -- value counts are:
 Nill    963
+1       718
+2        22
Name: Extra, dtype: int64


For column -- From -- value counts are:
 DEL    1703
Name: From, dtype: int64


For column -- To -- value counts are:
 DXB    1666
XNB       37
Name: To, dtype: int64


For column -- Day of flight -- value counts are:
 Mon         236
Sun          230
Fri          222
Thursday     216
Monday       215
sunday       213
Tuesday      213
Sat          158
Name: Day of flight, dtype: int64


For column -- month of flight -- value counts are:
 Jan    1703
Name: month of flight, dtype: int64
```

```
For column -- stop_details -- value counts are:
 1 stop     965
Direct      389
2 stops     266
3 stops      70
4 stops      13
Name: stop_details, dtype: int64


For column -- first_stop -- value counts are:
 NA       389
 BOM      377
 AMD      179
 BLR       60
 HYD       58
 JAI       51
 DOH       50
 MAA       47
 IXC       45
 BAH       40
 LKO       37
 ADD       32
 KWI       30
 MCT       27
 ALA       22
 COK       19
 DED       17
 CCU       17
 LHR       17
 AUH       16
 RUH       16
 CMB       15
 IDR       14
 IST       14
 IXL       12
 SIN       12
 GOI       12
 PNQ       11
 JED        9
 ZRH        8
 FRA        8

For column -- second_stop -- value counts are:
 NA      1354
AMD      104
BOM       87
JAI       23
AUH       21
PNQ       17
JED       14
IXC       14
MCT       12
NQZ       10
LKO        9
CMB        8
HYD        7
IST        5
CAI        5
CCJ        3
RUH        3
FRA        2
MAA        2
DOH        1
CGP        1
MUC        1
Name: second_stop, dtype: int64


For column -- third_stop -- value counts are:
 NA      1620
BOM       32
AMD       20
JAI       11
PNQ        9
IXC        5
MAA        4
HYD        1
LKO        1
Name: third_stop, dtype: int64
```

```
For column -- fourth_stop -- value counts are:
 NA     1690
BOM        9
JAI        3
MAA        1
Name: fourth_stop, dtype: int64


For column -- airline1 -- value counts are:
  Emirates                        320
 IndiGo                           295
 Vistara                          213
 Air India                        134
 Vistara                           91
 SpiceJet                          90
 IndiGo                            87
 Air India                         70
 SpiceJet                          49
 Qatar Airways                     41
 Gulf Air                          33
 Ethiopian Airlines                28
 Kuwait Airways                    27
 Oman Air                          22
 Air Astana                        22
 British Airways                   17
 Etihad Airways                    16
 Flynas                            13
 Lufthansa                         13
 Turkish Airlines                  12
 Saudia                            12
 SriLankan Airlines                12
 Singapore Airlines                12
 Qatar Airways                      9
 Air India Express                  8
 SWISS                              8
 flydubai                           6
 Biman Bangladesh Airlines          5
 Oman Air                           5
 EgyptAir                           5
 Uzbekistan Airways                 4
 Go First                           4
```

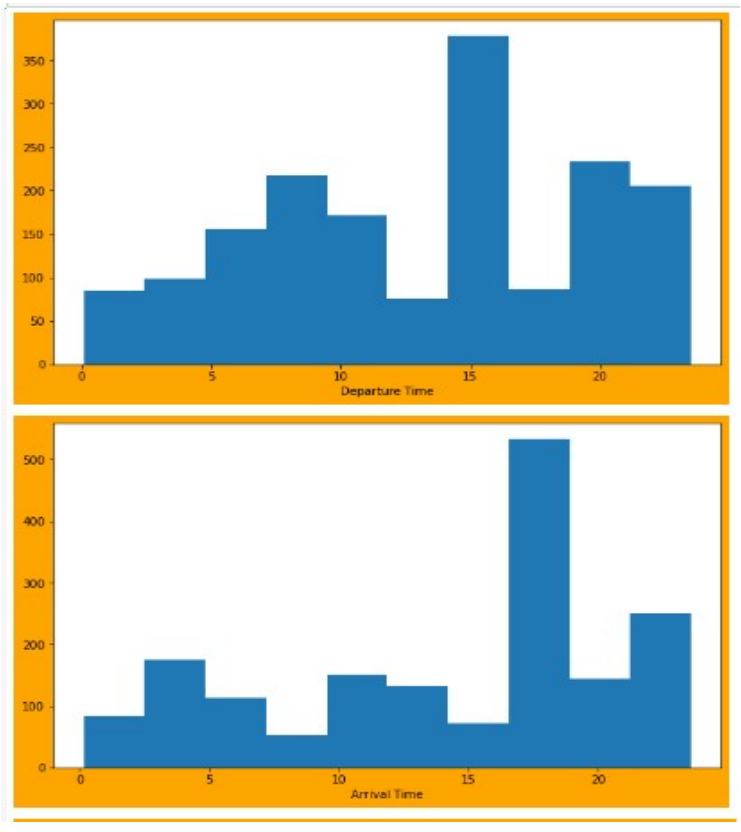
```
For column -- airline1 -- value counts are:
  Emirates                        320
 IndiGo                           295
 Vistara                          213
 Air India                        134
 Vistara                           91
 SpiceJet                          90
 IndiGo                            87
 Air India                         70
 SpiceJet                          49
 Qatar Airways                     41
 Gulf Air                          33
 Ethiopian Airlines                28
 Kuwait Airways                    27
 Oman Air                          22
 Air Astana                        22
 British Airways                   17
 Etihad Airways                    16
 Flynas                            13
 Lufthansa                         13
 Turkish Airlines                  12
 Saudia                            12
 SriLankan Airlines                12
 Singapore Airlines                12
 Qatar Airways                      9
 Air India Express                  8
 SWISS                              8
 flydubai                           6
 Biman Bangladesh Airlines          5
 Oman Air                           5
 EgyptAir                           5
 Uzbekistan Airways                 4
 Go First                           4
 Ethiopian Airlines                 4
 Jazeera Airways                    3
 Cathay Pacific                     3
 AirAsia India                      3
 Turkish Airlines                   2
 Lufthansa                          2
 LOT                                2
 Biman Bangladesh Airlines          1
```

```
For column -- airline2 -- value counts are:
 Na                    1254
 Emirates               213
 SpiceJet                80
 IndiGo                  41
 flydubai                35
 Etihad Airways          21
 Oman Air                12
 SriLankan Airlines      11
 Air India Express        8
 Gulf Air                 7
 Vistara                  6
 Turkish Airlines         5
 Flynas                   4
 Air India                3
 Qatar Airways            1
 Cathay Pacific           1
 Air India                1
Name: airline2, dtype: int64


For column -- airline3 -- value counts are:
 Na           1694
 Emirates        9
Name: airline3, dtype: int64
```

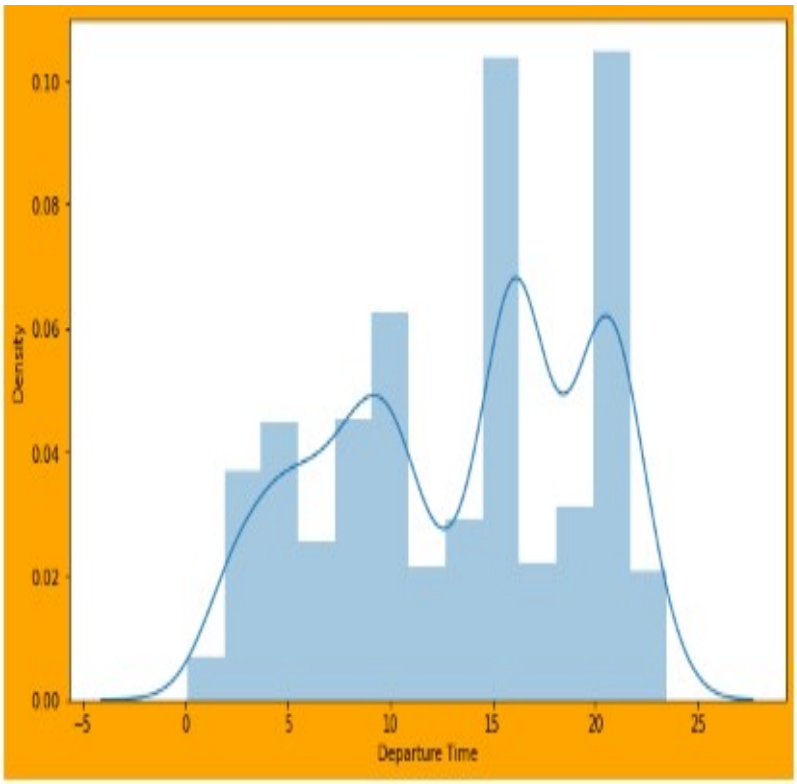
Visualizations

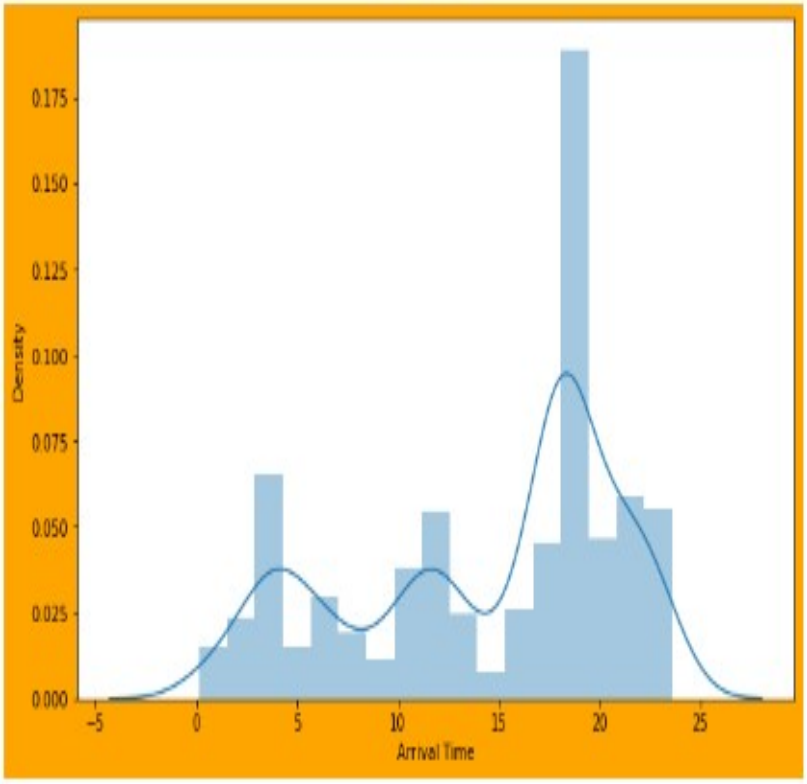
Plots used are:

- Frequency analysis plots:
  1. Histogram plot.

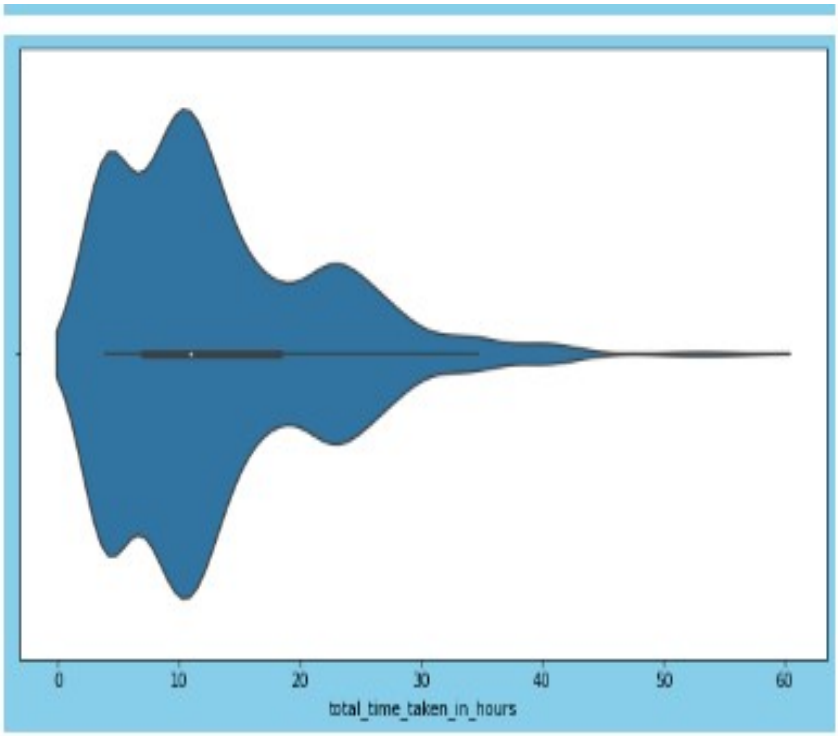


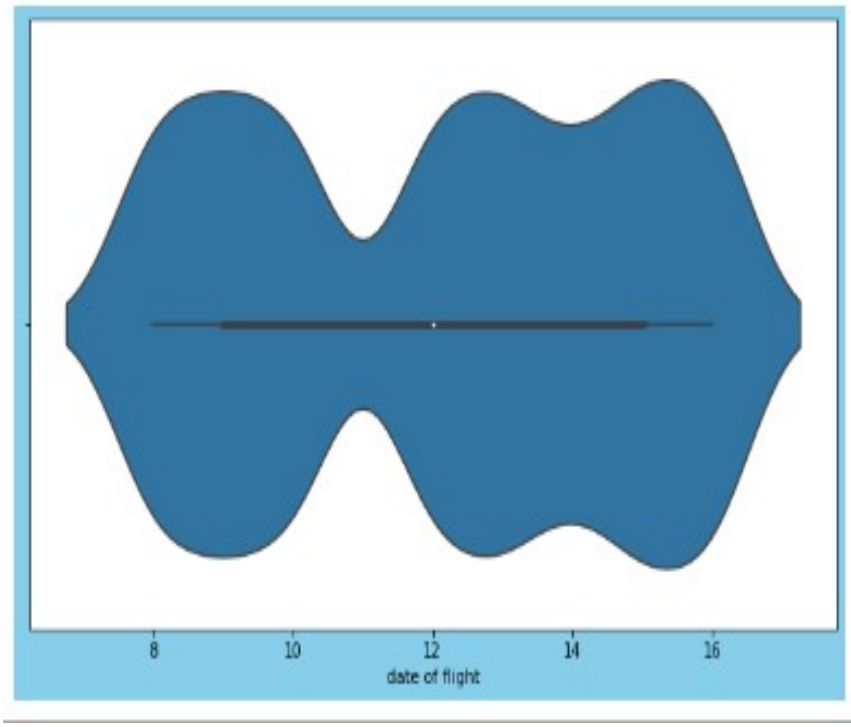
2. Distribution plot



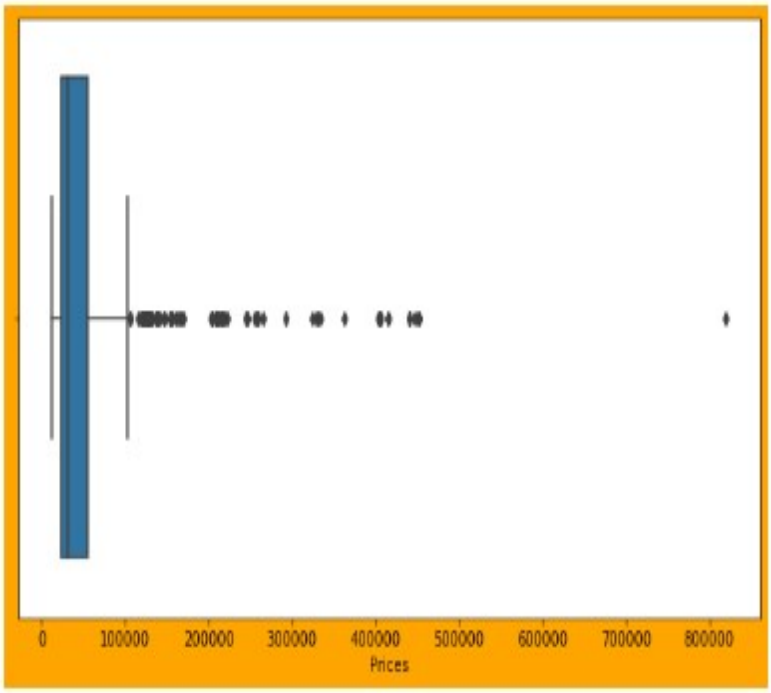


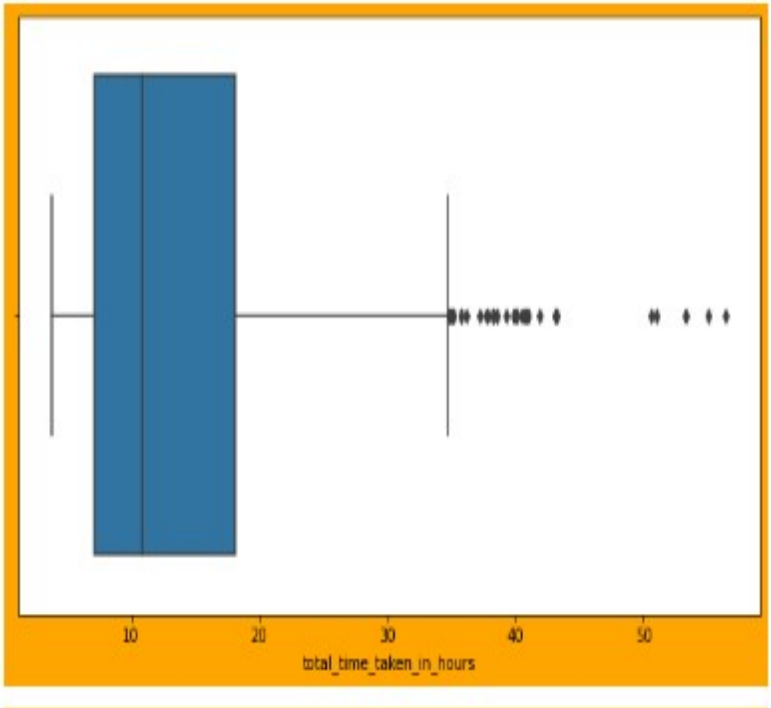
3. Violin plot



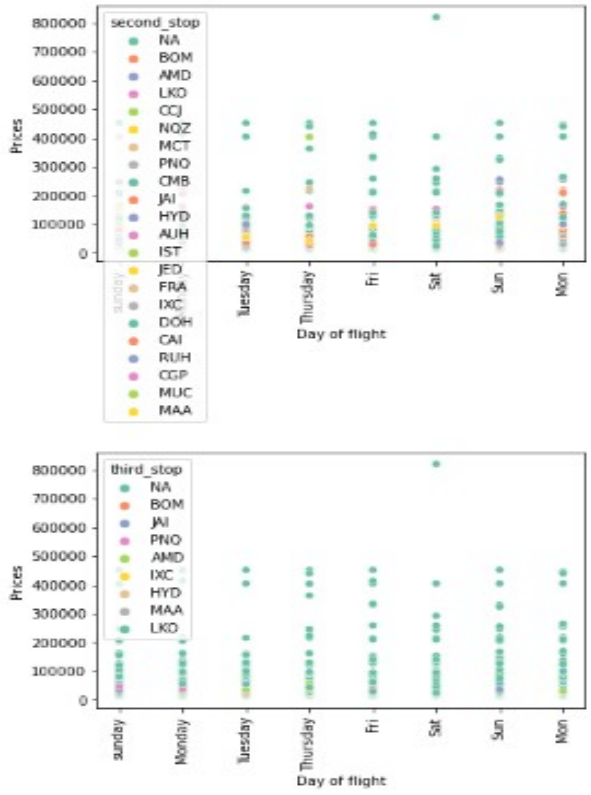


- Outlier presence analysis:
  1. Boxplots



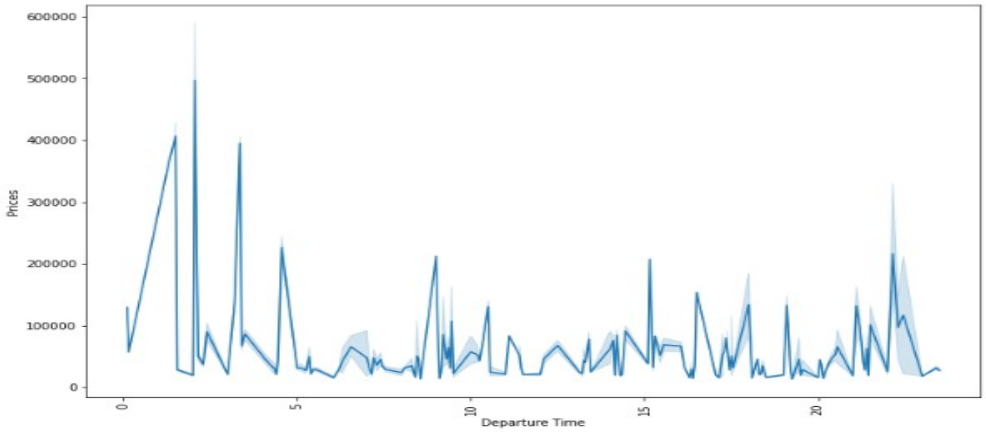


- Univariate Analysis:
  1. Scatterplot

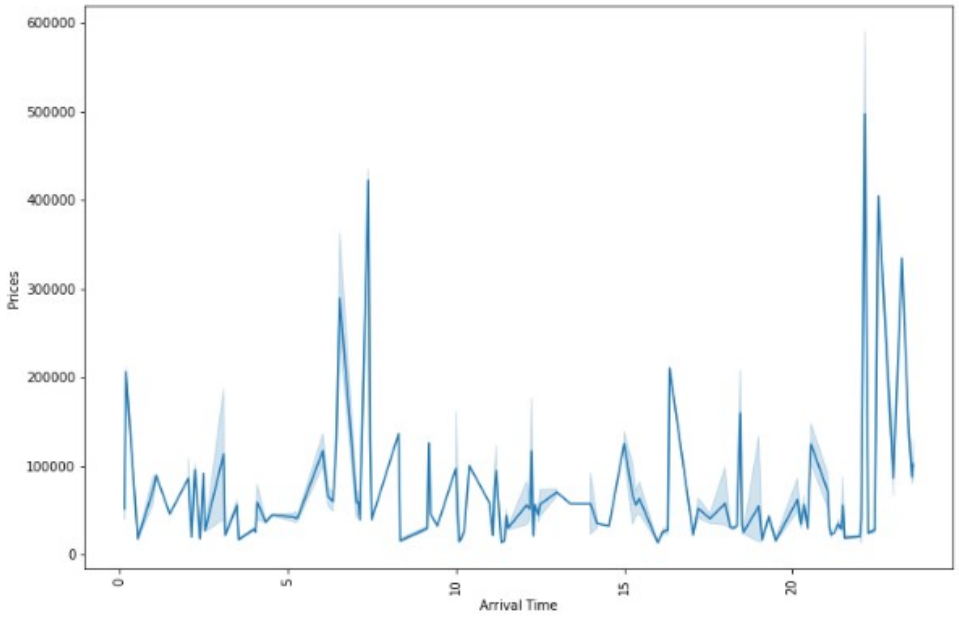


  2. Lineplot

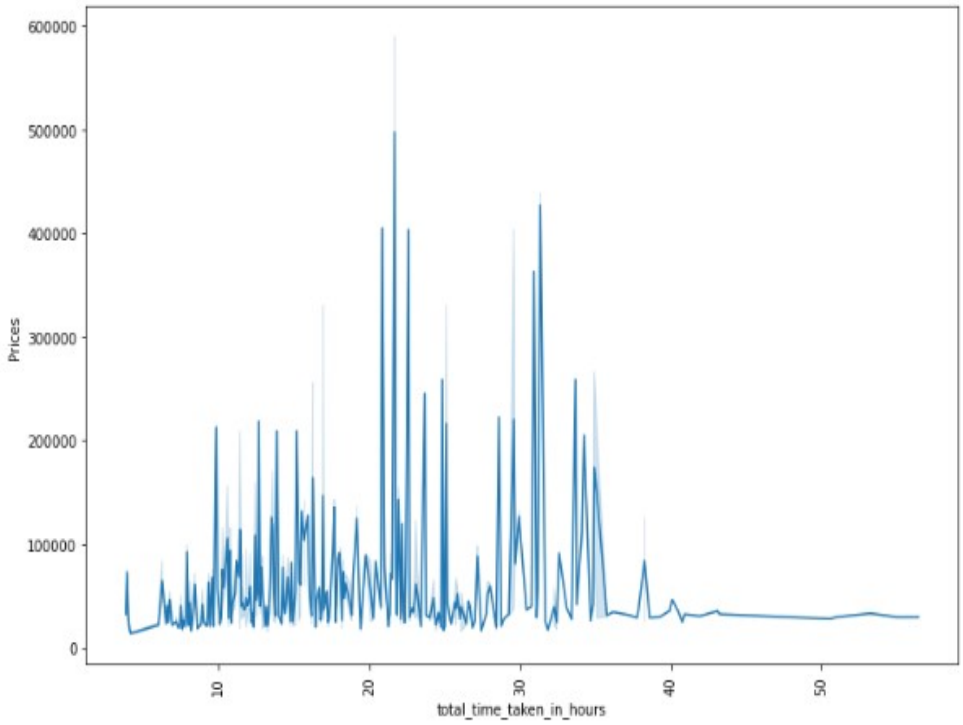
```
plt.figure(figsize=(12,8))
sns.lineplot(x = df['Departure Time'], y = 'Prices', data = df)
plt.xticks(rotation=90)
plt.show()
```



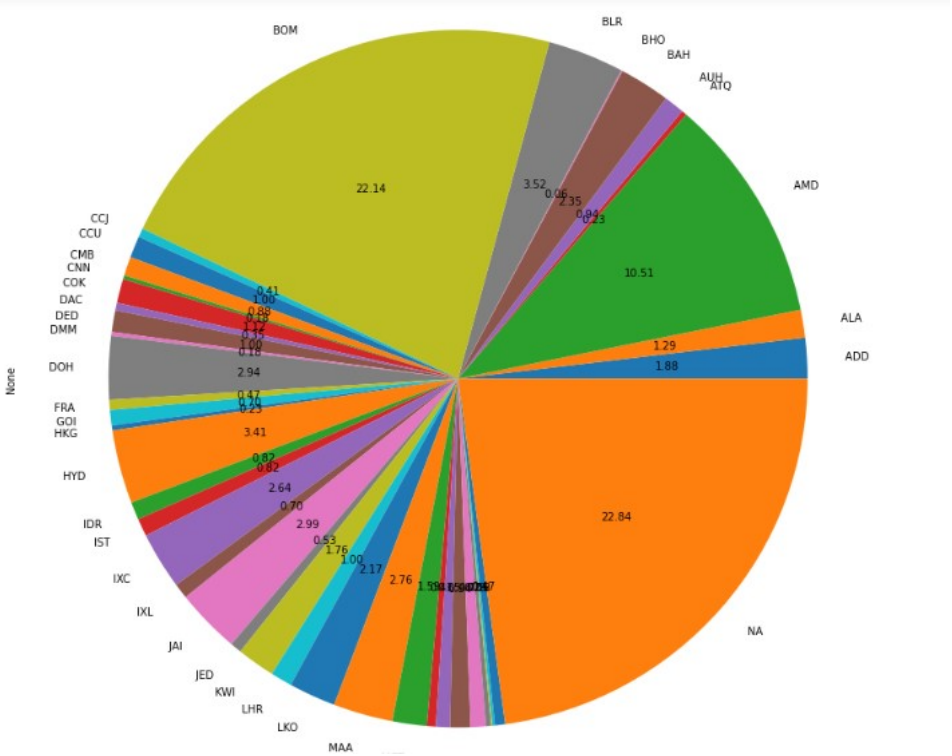
```
]: plt.figure(figsize=(12,8))
   sns.lineplot(x = df['Arrival Time'], y = 'Prices', data = df)
   plt.xticks(rotation=90)
   plt.show()
```




```
plt.figure(figsize=(12,8))
sns.lineplot(x = df['total_time_taken_in_hours'], y = 'Prices', data = df)
plt.xticks(rotation=90)
plt.show()
```

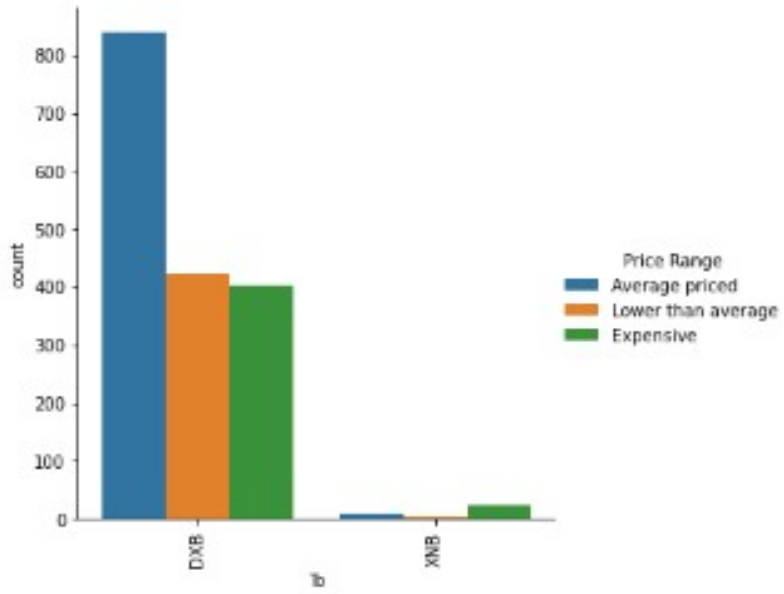


3. PiePlot:

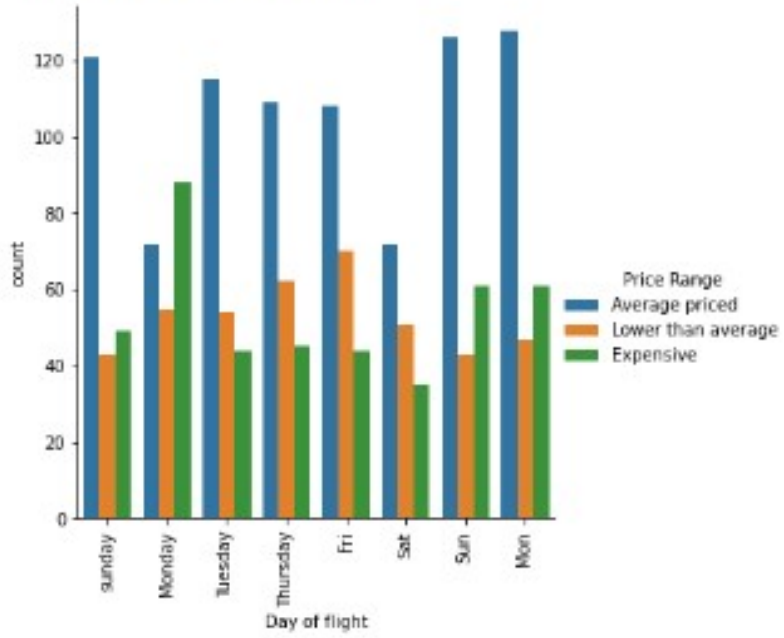


- Bivariate Analysis
  1. Relplot

2. Categorical plots and box plots

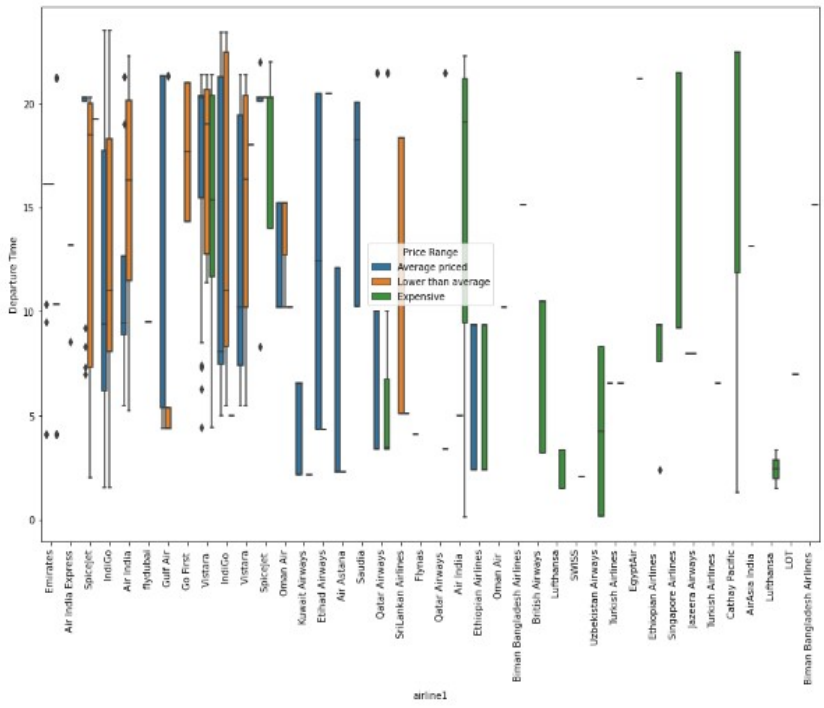



```
<Figure size 864x576 with 0 Axes>
```

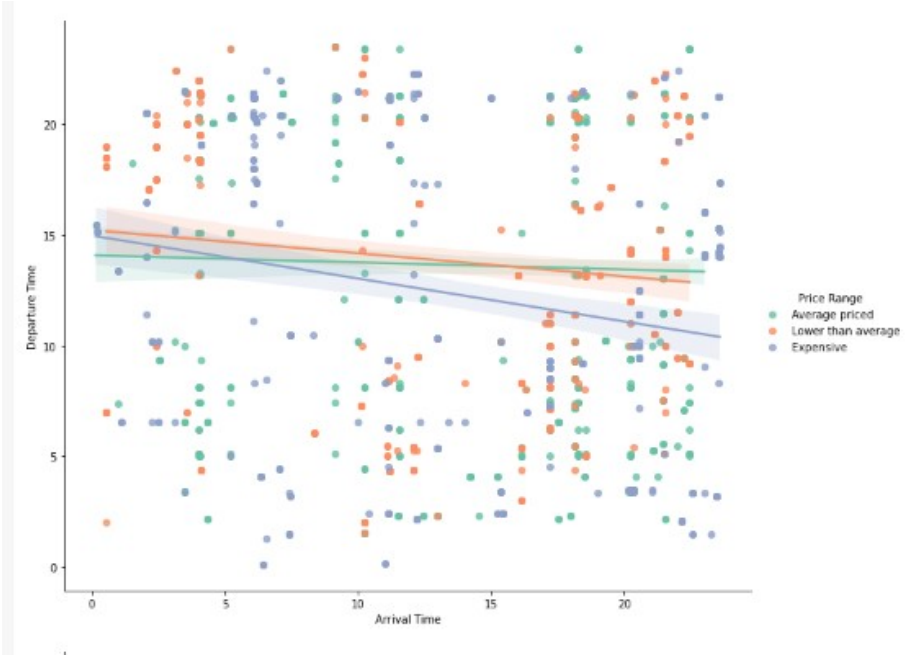



```
<Figure size 864x576 with 0 Axes>
```

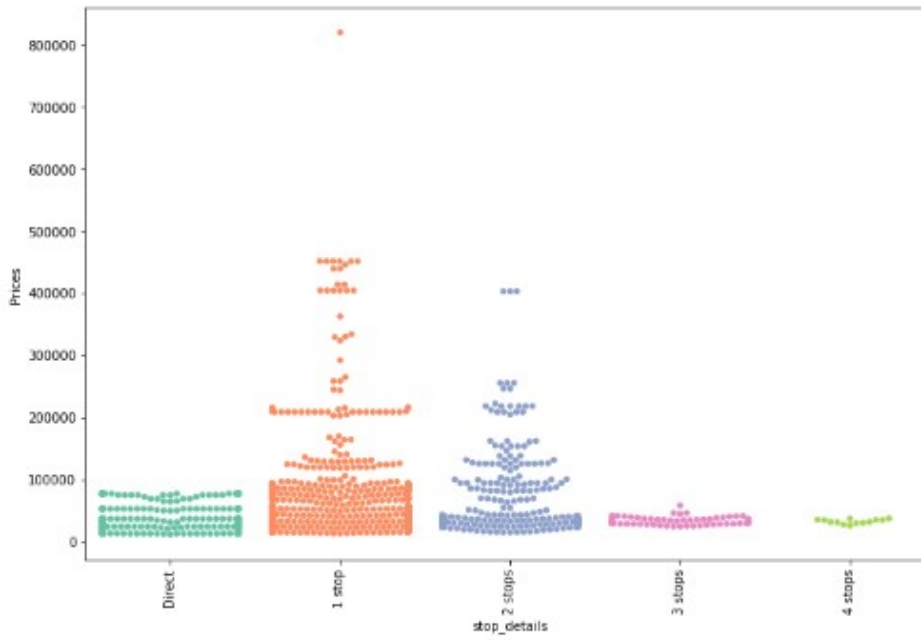
```
for i in cont_columns:
    plt.figure(figsize=(15,10))
    sns.boxplot(x ='airline1', y = i, data = df, hue ='Price Range')
    plt.xticks(rotation=90)
```

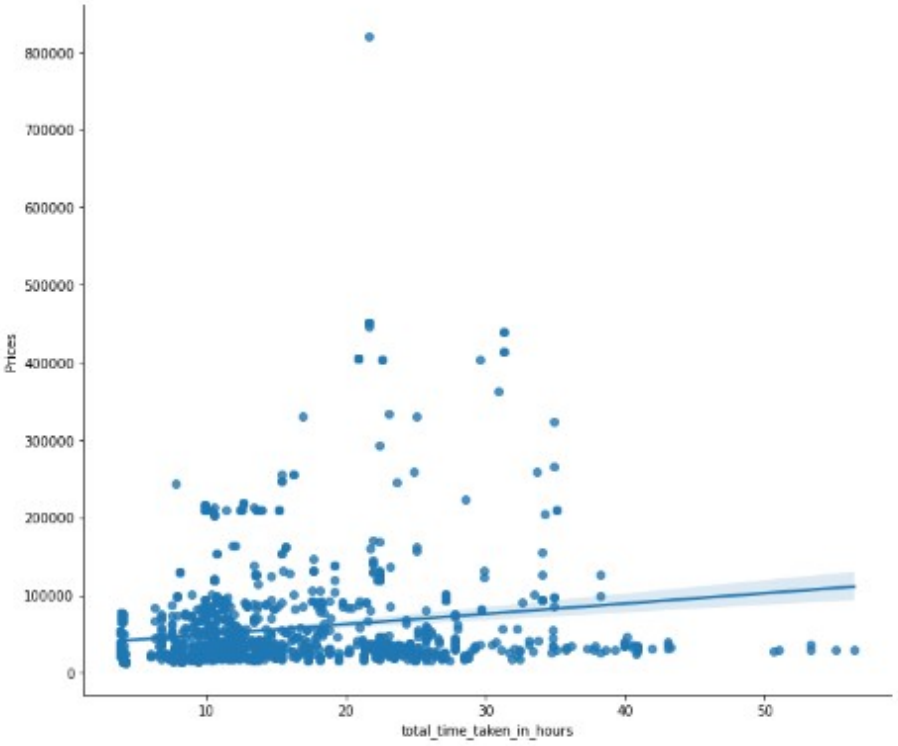


3. LMplot:

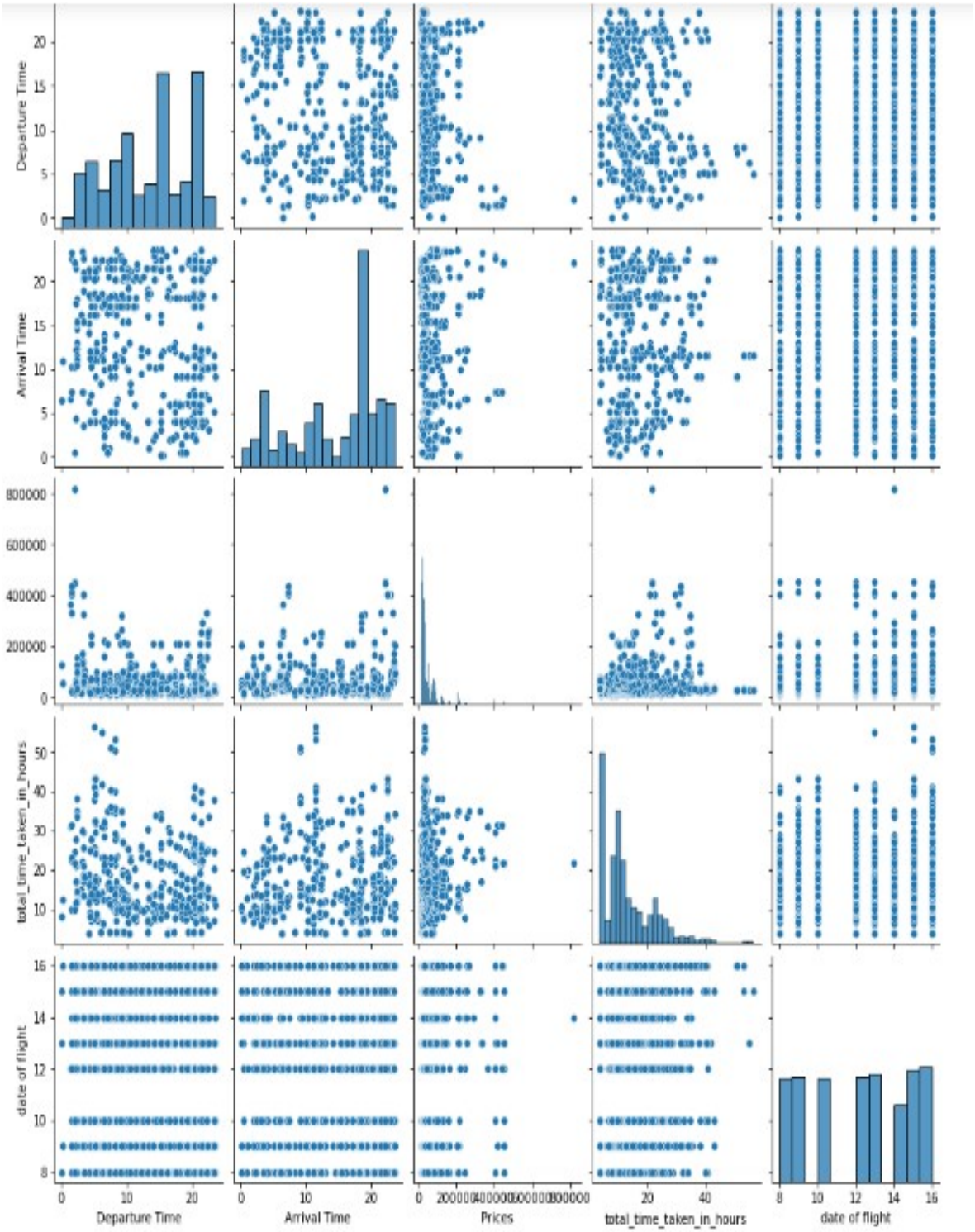


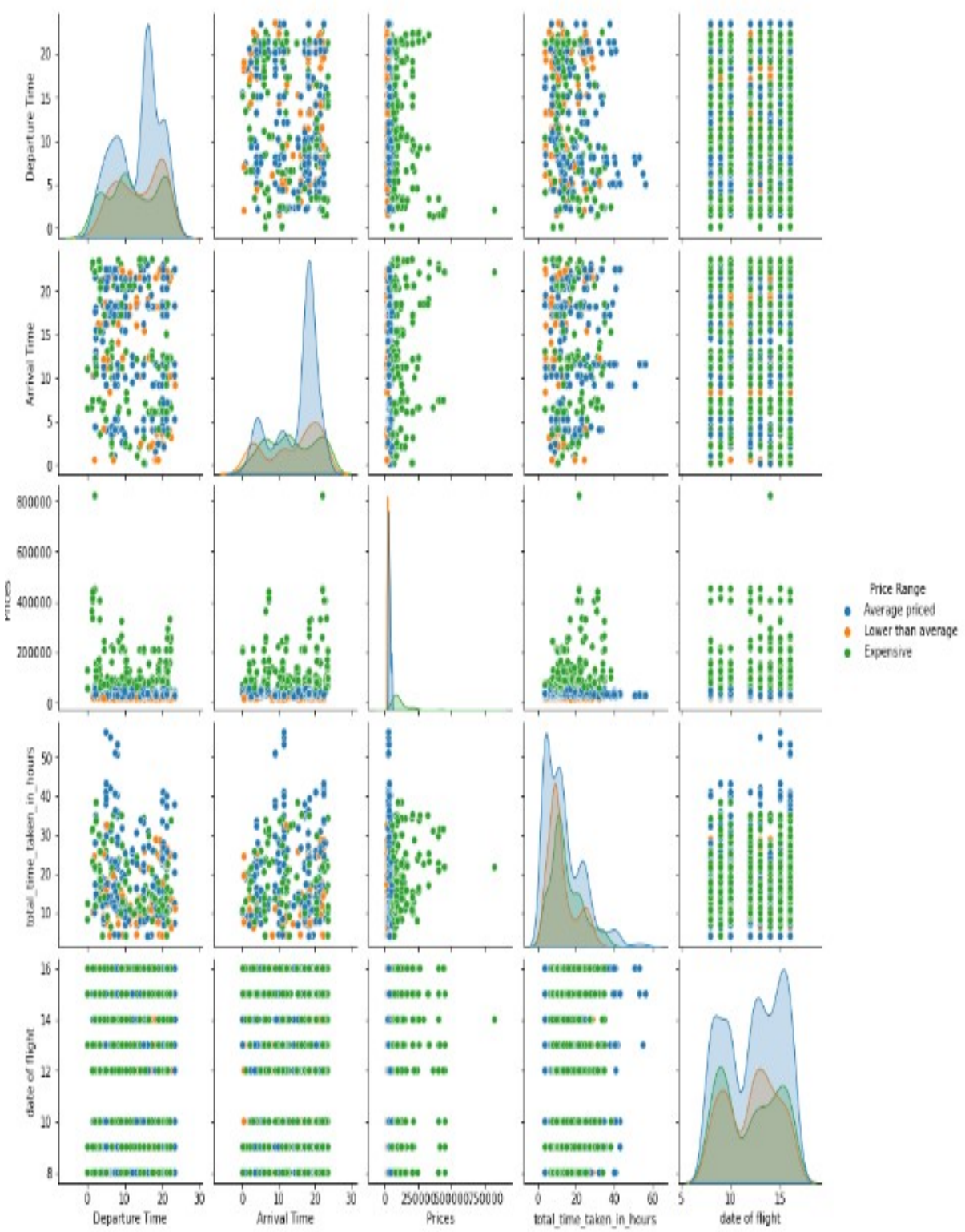
4. SwarmPlot


800000
700000
600000
500000
400000
300000
200000
100000
0
Prices
Direct
1 stop
2 stops
3 stops
4 stops
stop_details


800000
700000
600000
500000
400000
300000
200000
100000
0
Prices
10
20
30
40
50
total_time_taken_in_hours

- Multivariate Analysis:
  1. Pairplots





- HeatMap: (shown ahead in correlation analysis section)

Inferences:

- Most flights are landing at Dubai international airport.
- Most flights are deboarding on Monday.
- Most flights have at least one stop between point of departure and final destination.
- For most flights first stop is Bombay.
- Emirates Indigo and vistara are most common flights leaving for Dubai international airport.
- Air India, Emirates, Ethiopian airlines, British Airways, Lufthansa, Swiss, Uzbekistan Airways, Jazeera Airways cafe Pacific or most expensive airlines and one must avoid booking flights in them.
- Flights having departure time between 10 to 15 are lower than average priced.
- Sri Lankan airlines Indigo SpiceJet are one of the cheapest airlines to Dubai.

Grouped Data Analysis:

```
pd.set_option('display.max_rows', None)
df.groupby('airline1').mean().sort_values(by='Prices',ascending=False)
```

airline1	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight	stop_details
SWISS	2.050000	22.150000	497096.875000	21.666667	12.125000	0.000000
Lufthansa	2.496154	16.757692	411241.692308	25.012821	12.461538	0.230769
Lufthansa	2.425000	21.125000	331993.500000	20.000000	13.000000	0.000000
Cathay Pacific	15.400000	11.716667	293518.666667	29.805556	13.000000	0.000000
Biman Bangladesh Airlines	15.150000	3.100000	212369.000000	13.416667	16.000000	1.000000
LOT	7.000000	16.350000	210239.000000	35.083333	13.500000	0.000000
Biman Bangladesh Airlines	15.150000	4.878000	205991.600000	15.313333	11.400000	0.200000
Turkish Airlines	6.550000	6.550000	191814.500000	25.166667	11.000000	1.000000
Air India	15.042857	11.994286	173346.900000	13.238095	12.114286	0.442857
Singapore Airlines	16.375000	18.450000	159517.916667	27.625000	12.916667	0.000000
British Airways	7.064706	13.147059	134463.882353	23.166667	12.705882	0.000000
EgyptAir	21.200000	15.600000	124030.400000	19.766667	11.800000	1.000000
Ethiopian Airlines	7.612500	10.962500	100396.000000	28.750000	13.750000	1.000000
Oman Air	10.200000	2.350000	94689.600000	17.750000	12.200000	1.000000
Qatar Airways	7.422222	14.161111	83805.555556	13.472222	11.777778	1.000000
Ethiopian Airlines	5.378571	11.037500	74760.392857	24.264881	12.392857	0.464286
Turkish Airlines	6.550000	5.066667	73914.666667	23.861111	12.250000	0.000000
Vistara	15.912911	13.316808	70716.727700	12.497105	12.248826	0.136150
Qatar Airways	8.563415	14.610976	64046.536585	15.741870	12.341463	0.000000
Uzbekistan Airways	4.225000	11.000000	58301.750000	20.125000	12.000000	0.000000
Saudia	15.516667	11.225000	44413.583333	11.347222	12.250000	0.000000
Kuwait Airways	4.105556	12.838889	44100.925926	16.500000	12.703704	0.000000

```
pd.set_option('display.max_rows', None)
df.groupby('airline2').mean().sort_values(by='Prices',ascending=False)
```

airline2	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight	stop_details
Cathay Pacific	8.450000	6.550000	245408.000000	23.666667	12.000000	0.000000
Vistara	9.783333	18.083333	193202.166667	13.777778	12.833333	1.000000
Air India	18.000000	6.050000	166918.000000	13.583333	11.000000	1.000000
Etihad Airways	19.195238	8.450000	138715.047619	14.734127	12.190476	1.000000
Gulf Air	19.100000	4.100000	120300.857143	10.500000	12.000000	0.000000
Oman Air	7.920833	11.970833	112520.666667	15.416667	12.000000	1.000000
Flynas	7.612500	10.962500	100396.000000	28.750000	13.750000	1.000000
Emirates	14.757042	14.111549	98864.061033	12.198826	12.215962	0.098592
Turkish Airlines	10.200000	2.350000	94689.600000	17.750000	12.200000	1.000000
SriLankan Airlines	20.963636	21.500000	88690.545455	26.151515	11.636364	0.727273
Qatar Airways	21.400000	20.250000	68045.000000	24.250000	9.000000	1.000000
Na	12.664633	14.838150	44631.452153	12.492544	12.151515	1.423445
flydubai	17.431429	6.064286	40675.885714	13.430952	11.914286	0.314286
SpiceJet	13.385625	15.245000	31683.175000	24.182292	12.087500	1.100000
IndiGo	18.546341	13.485366	26401.560976	16.902439	11.902439	0.878049
Air India	20.400000	2.400000	23536.000000	7.500000	14.000000	0.000000
Air India Express	16.100000	12.400000	22564.750000	6.833333	12.500000	0.000000

```
pd.set_option('display.max_rows', None)
df.groupby('airline3').mean().sort_values(by='Prices',ascending=False)
```

airline3	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight	stop_details
Emirates	12.522222	14.072222	184440.777778	13.712963	12.222222	1.000000
Na	13.334711	14.384799	52857.240260	13.300521	12.145809	1.175915

```
pd.set_option('display.max_rows', None)
df.groupby('fourth_stop').mean().sort_values(by='Prices',ascending=False)
```

fourth_stop	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight	stop_details
NA	13.381657	14.407899	53711.211243	13.144921	12.144379	1.160947
BOM	6.033333	11.261111	34105.000000	30.805556	11.555556	3.000000
JAI	8.100000	11.550000	31091.000000	37.250000	13.666667	3.000000
MAA	8.100000	9.150000	27967.000000	50.583333	16.000000	3.000000

```
pd.set_option('display.max_rows', None)
df.groupby('second_stop').mean().sort_values(by='Prices',ascending=False)
```

second_stop	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight	stop_details
MUC	3.350000	7.400000	403410.000000	29.583333	12.000000	1.000000
FRA	1.500000	22.550000	403258.000000	22.583333	10.000000	1.000000
CGP	15.150000	23.590000	205117.000000	34.233333	9.000000	1.000000
HYD	21.628571	11.964286	199484.000000	15.773810	13.285714	1.000000
AUH	19.195238	8.450000	138715.047619	14.734127	12.190476	1.000000
CAI	21.200000	15.600000	124030.400000	19.766667	11.800000	1.000000
MCT	7.920833	11.970833	112520.666667	15.416667	12.000000	1.000000
RUH	2.400000	13.633333	108913.333333	36.666667	11.666667	1.000000
JED	6.371429	11.150000	102614.928571	30.130952	12.928571	1.000000
IST	10.200000	2.350000	94689.600000	17.750000	12.200000	1.000000
DOH	21.400000	20.250000	68045.000000	24.250000	9.000000	1.000000
BOM	14.575862	15.170115	56849.137931	15.854406	12.160920	1.000000
NA	13.308272	14.520133	51417.064254	10.999335	12.146233	1.149188
NQZ	5.240000	13.450000	41813.800000	26.583333	12.000000	1.000000
CMB	20.518750	21.500000	40861.875000	26.552083	11.875000	1.000000
AMD	13.886538	13.549038	35933.490385	24.283654	11.951923	1.605769
JAI	11.039130	11.589130	33242.869565	32.474638	12.086957	1.608696
PNQ	19.847059	22.450000	32729.294118	28.191176	12.235294	1.000000
MAA	18.925000	9.150000	31760.000000	27.541667	15.500000	1.000000
IXC	8.835714	11.664286	31305.500000	30.166667	13.428571	2.357143
LKO	4.333333	16.150000	30396.444444	13.416667	12.000000	1.000000
CCJ	5.250000	11.916667	19132.666667	32.166667	10.333333	1.000000

Out[44]:

To	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight	stop_details
XNB	16.267568	7.660811	96700.810811	11.605856	12.162162	0.567568
DXB	13.265186	14.532443	52594.358944	13.340386	12.145858	1.188475

In [45]:
```
pd.set_option('display.max_rows', None)
df.groupby('total_time_taken_in_hours').mean().sort_values(by='Prices',ascending=False)
```

Out[45]:

total_time_taken_in_hours	Departure Time	Arrival Time	Prices	date of flight	stop_details
21.666667	2.050000	22.150000	497096.875000	12.125000	0.000000
31.333333	1.500000	7.400000	427005.500000	12.500000	0.000000
20.833333	3.350000	22.550000	404699.000000	14.000000	0.000000
22.583333	1.500000	22.550000	403258.000000	10.000000	1.000000
30.916667	1.300000	6.550000	362900.000000	12.000000	0.000000
24.833333	22.450000	22.050000	258828.000000	13.000000	0.000000
33.666667	22.450000	6.550000	258828.000000	14.000000	0.000000
20.916667	10.300000	17.750000	245804.500000	13.000000	0.000000
23.666667	8.450000	6.550000	245408.000000	12.000000	0.000000
28.583333	6.550000	10.000000	222581.000000	12.000000	1.000000

```
for i in cont_columns:
    for j in cat_columns:
        print('Grouped data by',j, 'on',i,':\n\n',df.groupby(j)[i].mean().sort_values(ascending = False),'\n\n')
```

```
Grouped data by Extra on Departure Time :

 Extra
+2      17.254545
+1      16.564554
Nill    10.829439
Name: Departure Time, dtype: float64


Grouped data by From on Departure Time :

 From
DEL    13.330417
Name: Departure Time, dtype: float64


Grouped data by To on Departure Time :

 To
```

```
for i in cont_columns:
    for j in cat_columns:
        print('Grouped data by',j, 'on',i,':\n\n',df.groupby(j)[i].median().sort_values(ascending = False),'\n\n')
```

```
XNB    20.5
DXB    15.1
Name: Departure Time, dtype: float64


Grouped data by Day of flight on Departure Time :

 Day of flight
Tuesday     16.150
sunday      16.150
```

Answers to major questions using Data Analysis:

1. **Do airfares change frequently?**

Ans. Yes air fares do change frequently. Approximately, the minimum value of a flight fare is 12000 where as the maximum value of a flight fare is 819000. The mean value of fare prices 53000.

2. **Do they move in small increments or in large jumps?**

Ans. Yes they definitely 10 to go up and down overtime, in general we can give a remark that early morning flight fares are over priced as compared to flights departing in afternoon and mid evening.

3. **Do they tend to go up or down over time?**

Ans. Flight fare where is a lot depending upon the departure time arrival time and total time taken by the flight.

By noting down the minimum maximum and mean fair price for flight tickets, the standard deviation in our data is 63000 approximately which is quite large.

For departure time between 0 to 5 fluctuation in fair price is huge as compared to fluctuation in fair prices between 5 to 20.

Large fluctuation can be seen in fair prices when total time taken by the flights is between 20 hours to 35 hours.

4. **What is the best time to buy so that the consumer can save the most by taking the least risk?**

Ans. A consumer can book flight ticket when the departure time is between timing 5 to 7 and between 10 to 15 or at 20.

Flight fares for arrival time between 1 to 5 and 10 to 20 is also less as compared to other arrival Times.

For flights taking more total time in hours, fare prices for them is also less as compared to other flights who are taking less time.

5. **Does price increase as we get near to departure date?**

Ans. Yes price do increase as we get near to departure time.

6. **Is Indigo cheaper than SpiceJet?**

Ans. Grouping airlines1 by mean prices it is clear that mean fare price of SpiceJet that is approximately 425 00 is more than mean fair price of Indigo that is approximately 33500.

Even after checking by grouping airline2 by mean prices we can note that approximate mean fare price for SpiceJet is 31600 rupees which is more as compared to mean fair price of Indigo that is 26000 rupees approximately.

7. **Are morning flights expensive?**

Ans. Yes morning flights are most expensive and are over priced as compared to other flights at other Times.

11.Checking skewness and Outlier removal:

```
# Skewness
```

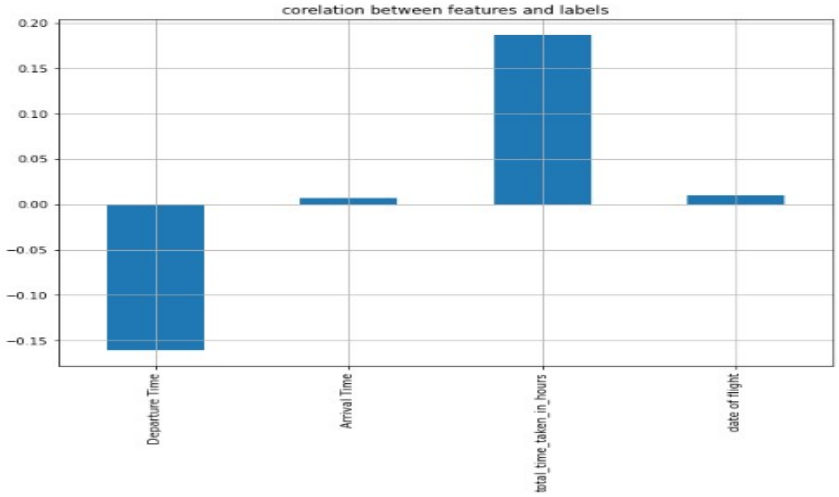
```
df.skew()
```

```
Departure Time              -0.247124
Arrival Time                -0.585123
Prices                       4.364011
total_time_taken_in_hours    1.206085
date of flight              -0.112928
dtype: float64
```

```
# Using Z Statistics to check and remove any more outliers:
from scipy.stats import zscore
z_score = zscore(df1[cont_columns])
abs_z_score = np.abs(z_score)
filtering_entry = (abs_z_score < 3).all(axis=1)
df1 = df1[filtering_entry]
df1.describe()
```

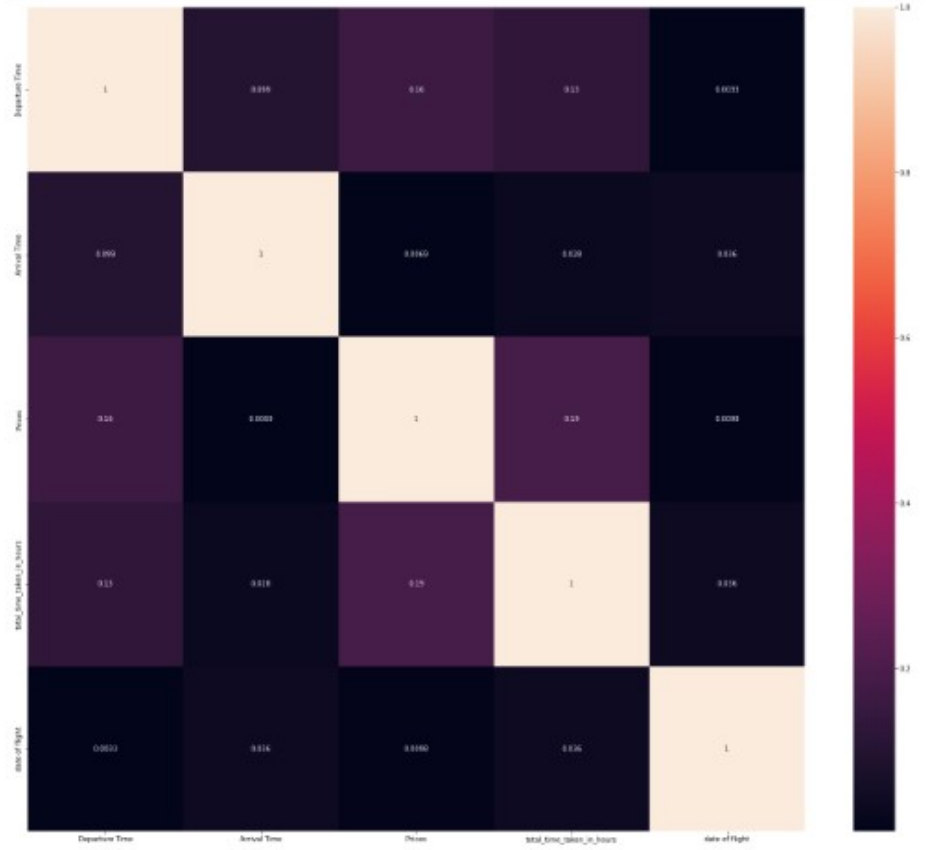
	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight
count	1640.000000	1640.000000	1640.000000	1640.000000	1640.000000
mean	13.488902	14.303323	46647.155488	12.596392	12.115244
std	6.057877	6.575078	39012.066557	7.917335	2.750031
min	0.100000	0.150000	12187.000000	3.833333	8.000000
25%	8.400000	9.237500	23165.000000	6.750000	9.000000
50%	15.250000	17.200000	31780.000000	10.750000	12.000000
75%	19.025000	18.350000	53621.250000	17.250000	15.000000
max	23.500000	23.590000	222581.000000	39.250000	16.000000

12. Corelation between features and Sales Price:



13.Heatmap:

```
df_corr = df.corr().abs()
plt.figure(figsize=(25,22))
sns.heatmap(df_corr,annot=True,annot_kws={'size':18})
plt.show()
```



14. Variance inflation factor:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor


vif = pd.DataFrame()
vif['vif'] = [variance_inflation_factor(df[cont_columns],i) for i in range(df[cont_columns].shape[1])]
vif['features'] = df[cont_columns].columns
vif.sort_values(by='vif',ascending=False)
```

	vif	features
4	9.652482	date of flight
1	4.939356	Arrival Time
0	4.675905	Departure Time
3	3.178724	total_time_taken_in_hours
2	1.772152	Prices

15. Encoding categorical column data:

```
df_encoded = pd.get_dummies(df)

df_encoded
```

	Departure Time	Arrival Time	Prices	total_time_taken_in_hours	date of flight	Extra_+1	Extra_+2	Extra_Nill	From_DEL	To_DXB	...	airline2_ Oman Air	airline2_ Qatar Airways	airline2_ SpiceJet	airli SriLa Air
0	16.15	18.35	31780.0	3.833333	8	0	0	1	1	1	...	0	0	0	
1	13.20	16.00	12788.0	4.166667	8	0	0	1	1	1	...	0	0	0	
2	7.30	10.10	13380.0	4.166667	8	0	0	1	1	1	...	0	0	0	
3	8.40	11.15	14118.0	4.083333	8	0	0	1	1	1	...	0	0	0	
4	6.05	8.35	14451.0	4.000000	8	0	0	1	1	1	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1698	21.40	12.25	255817.0	16.250000	16	1	0	0	1	1	...	0	0	0	
1699	3.35	22.55	404699.0	20.833333	16	0	0	1	1	1	...	0	0	0	
1700	9.20	18.45	265167.0	34.916667	16	1	0	0	1	1	...	0	0	0	
1701	2.05	22.15	446204.0	21.666667	16	0	0	1	1	1	...	0	0	0	
1702	1.50	7.40	439798.0	31.333333	16	1	0	0	1	1	...	0	0	0	

1703 rows × 161 columns

16.Feature and Label Selection:

```
# Splitting data into features and label:

y = df_encoded['Prices']
X = df_encoded.drop('Prices',axis=1)
```

17. Standard Scaler and PCA:

SYNTAX

scaler = StandardScaler()

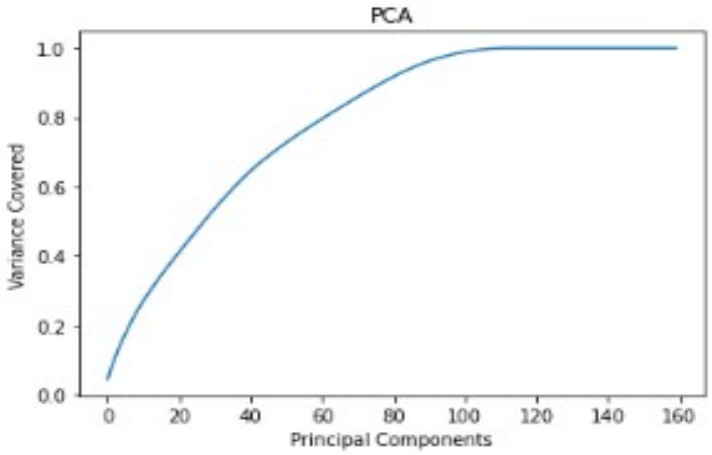
X_scaled = scaler.fit_transform(X)

PCA:

```
from sklearn.decomposition import PCA
# Using PCA i.e. Principal Component Analysis that is a diamensionallity reduction technique:

pcaf = PCA()
pcaf.fit_transform(X_scaled)
```

```
array([[-2.69862781e+00,  2.22168888e+00, -9.14650441e-01, ...,
        -2.57533235e-16, -1.62779732e-16,  4.54703351e-17],
       [-2.55864918e+00,  2.09032712e+00, -7.90143255e-01, ...,
        -9.44731097e-17,  8.92144361e-17, -4.91940682e-17],
       [-2.08033717e+00,  1.56834152e+00, -5.59779022e-01, ...,
        -2.78082813e-17,  4.02991849e-17, -4.12270694e-17],
       ...,
       [ 6.52412059e-01, -8.22643381e-01,  7.79585110e-01, ...,
        -2.25939480e-17, -5.57409304e-17, -1.76261002e-17],
       [-1.20133812e+00,  8.46773937e-01,  5.21422415e-01, ...,
        -6.68393228e-17,  2.87248947e-16,  1.15419411e-16],
       [ 6.11788722e-01, -6.19070779e-01,  8.31031125e-01, ...,
        -2.61362711e-17, -4.64705548e-17,  1.44515643e-17]])
```

```
# Using Scree Plot to identify best components:

plt.figure()
plt.plot(np.cumsum(pcaf.explained_variance_ratio_))
plt.xlabel('Principal Components')
plt.ylabel('Variance Covered')
plt.title('PCA')
plt.show()
```



```
from sklearn.decomposition import PCA

pcaf = PCA(n_components=110)
new_pcompf = pcaf.fit_transform(X_scaled)
princi_compf = pd.DataFrame(new_pcompf)
princi_compf
```

	0	1	2	3	4	5	6	7	8	9	...	100	101	102	103
0	-2.698628	2.221689	-0.914650	2.014228	0.129282	0.371153	0.288505	-0.148873	-0.812969	-0.495424	...	0.062133	-0.063327	0.124399	0.015021
1	-2.558649	2.090327	-0.790143	1.807476	0.153190	0.199249	0.253511	-0.099706	-0.548043	-0.395042	...	0.080656	-0.082754	0.140567	0.011090
2	-2.080337	1.568342	-0.559779	1.173639	0.134316	-0.093388	0.236101	0.086930	-0.245041	-0.759584	...	-0.197593	0.322642	-0.797197	0.249284
3	-1.981053	1.803014	-0.527727	1.171771	-0.012738	-0.057126	0.174443	0.001395	0.103851	-0.395300	...	-0.199219	0.228498	-0.094828	0.182212
4	-2.218550	1.594575	-0.415543	1.115033	0.166611	-0.089638	0.194993	0.069870	-0.037182	-0.553605	...	-0.211424	0.595494	-0.218823	0.229839
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1698	1.998000	-2.856838	0.228053	-0.449947	1.094905	2.404442	0.112578	-0.814316	-3.264437	-0.154079	...	-0.120840	-0.099103	-0.031725	-0.035474
1699	-0.971791	0.582800	0.454678	-1.414767	0.430503	-1.468381	-0.675097	1.194740	3.483979	1.806082	...	0.028733	-0.020900	-0.036094	0.029463
1700	0.662412	-0.822643	0.779586	-2.365800	-0.513754	-1.050079	0.032019	1.270315	0.517598	-1.278730	...	-0.208898	0.217855	0.311522	0.025346
1701	-1.201338	0.846774	0.521422	-1.491646	0.050253	-1.286513	-0.760751	0.900954	3.638799	2.700023	...	0.056759	-0.006404	-0.067966	0.025335
1702	0.611789	-0.619071	0.831031	-2.102747	0.420220	-1.981760	-0.350930	1.579814	3.094027	0.675687	...	0.196583	-0.487518	-0.509749	0.105801

1703 rows × 110 columns

18. Splitting Data

```
# Splitting our data to training data and testing data
# X_train,X_test,y_train,y_test

X_train,X_test,y_train,y_test = train_test_split(princi_compf,y,test_size=0.30,random_state=1)

# Here we are keeping training data as our scalled data and testing data as our label or target.
```

ALGORITHMS USED TO PREDICT PRICE OF HOUSES:

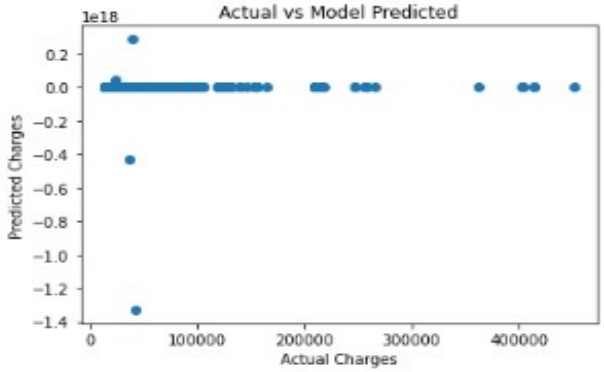
- Linear Regression
- Ridge and Lasso regression
- Decission Tree Regressor
- Random Forest Regressor (with Hyperparameter tuning)
- K-Neighbours Regressor (with Hyperparameter tuning)
- Support Vector Regressor
- AdaBoost Regressor (with Hyperparameter tuning)

Run and Evaluating selected models

1. Linear Regression:

```
# MOdel instantiating and training

rm = LinearRegression()
rm.fit(x_train,y_train)
# here we will pass training data

# Testing our model with Adjusted R2 Square:

# on training data

print('Training dataset score: ',rm.score(x_train,y_train))

# PLotting and visualizing

y_pred = rm.predict(x_test)

plt.scatter(y_test,y_pred)
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Model Predicted')
plt.show()
```

```
Training dataset score:  0.9217501040791363
```




```
import sklearn.metrics as metrics
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error


#model performance

print("MAE: ", mean_absolute_error(y_test, y_pred))
print("MSE: ", mean_squared_error(y_test, y_pred))
print("RMSE: ", mean_squared_error(y_test, y_pred, squared=False))
print("R2: ", metrics.r2_score(y_test, y_pred), "\n")
print("Score: ", rm.score(x_test, y_test))
```

```
MAE:  4094964412034645.5
MSE:  3.9826567268008814e+33
RMSE:  6.3108293645137336e+16
R2:  -1.1614624277607183e+24

Score:  -1.1614624277607183e+24
```

2. Ridge and lasso regression:

```
from sklearn.linear_model import Ridge,Lasso,RidgeCV,LassoCV

lassocv = LassoCV(alphas = None , max_iter = 100, normalize = True)
lassocv.fit(x_train,y_train)
```

```
LassoCV(max_iter=100, normalize=True)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
# Best alpha parameter
alpha = lassocv.alpha_ # Best alpha rate
alpha
```

```
92.67425291094905
```

```
# Now since we have the best parameter, lasso regression will be used:
lasso_reg = Lasso(alpha)
lasso_reg.fit(x_train,y_train)
# i.e. when model is training it will learn at this speed 6....
```

```
Lasso(alpha=92.67425291094905)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
lasso_reg.score(x_test,y_test)
```

```
0.8653652616722204
```

```
#model performance

print("MAE: ", metrics.mean_absolute_error(y_test, y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, y_pred))
print("RMSE: ", metrics.mean_squared_error(y_test, y_pred, squared=False))
print("R2: ", metrics.r2_score(y_test, y_pred), "\n")
print("Score: ", lasso_reg.score(x_test, y_test))
```

```
MAE:  4094964412034645.5
MSE:  3.9826567268008814e+33
RMSE:  6.3108293645137336e+16
R2:  -1.1614624277607183e+24

Score:  0.8653652616722204
```

```
# Ridge Method:

ridgecv = RidgeCV(alphas = np.arange(0.001,0.1,0.01),normalize = True)
ridgecv.fit(x_train,y_train)
```

```
RidgeCV(alphas=array([0.001, 0.011, 0.021, 0.031, 0.041, 0.051, 0.061, 0.071, 0.081,
       0.091]),
        normalize=True)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
ridgecv.alpha_ # Best alpha rate
```

```
0.001
```

```
ridge_model = Ridge(alpha = ridgecv.alpha_)
ridge_model.fit(x_train,y_train)
```

```
Ridge(alpha=0.001)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
ridge_model.score(x_test,y_test)
```

```
-3.3107683348598336
```

```
#model performance

print("MAE: ", metrics.mean_absolute_error(y_test, y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, y_pred))
print("RMSE: ", metrics.mean_squared_error(y_test, y_pred, squared=False))
print("R2: ", metrics.r2_score(y_test, y_pred), "\n")
print("Score: ", ridge_model.score(x_test, y_test))
```

```
MAE:  4094964412034645.5
MSE:  3.9826567268008814e+33
RMSE:  6.3108293645137336e+16
R2:  -1.1614624277607183e+24

Score:  -3.3107683348598336
```

3. Decision Tree and Random Forest Models:

```
# Using decision tree:
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
model = DecisionTreeRegressor()
model.fit(x_train, y_train)
y_preddt = model.predict(x_test)
r2_score(y_test,y_preddt)
```

```
0.857870271159503
```

```
# Random forest:
from sklearn.ensemble import RandomForestRegressor
regressor_rf = RandomForestRegressor(n_estimators = 200)
regressor_rf.fit(x_train, y_train)
lr_normal_rf = regressor_rf.score(x_train, y_train)
lr_normal_rf
```

```
0.9826907580040413
```

```
y_predrf = regressor_rf.predict(x_test)
lr_normal_rf_test = regressor_rf.score(x_test, y_test)
lr_normal_rf_test
mse_lr_normal_rf  = mean_absolute_error(y_test, y_predrf)
mse_lr_normal_rf
```

```
7232.312701228118
```

```
# Model Evaluation: MAE , MSE , RMSE

from sklearn.metrics import mean_absolute_error,mean_squared_error

print("MAE: ", metrics.mean_absolute_error(y_test, y_predrf))
print("MSE: ", metrics.mean_squared_error(y_test, y_predrf))
print("RMSE: ", metrics.mean_squared_error(y_test, y_predrf, squared=False))
print("R2: ", metrics.r2_score(y_test, y_predrf), "\n")
print("Score: ", regressor_rf.score(x_test, y_predrf))
```

```
MAE:  7232.312701228118
MSE:  260155778.2323428
RMSE:  16129.34525119798
R2:  0.9241307542906338

Score:  1.0
```

4. XG Boost Regressor:

```
# Using XGBoost:

xgb_reg = xgb.XGBRegressor()
xgb_reg.fit(x_train,y_train)
xgb_reg_score = xgb_reg.score(x_train, y_train)

print('Training score: ',xgb_reg.score(x_train, y_train))

print('Testing score: ',xgb_reg.score(x_test,y_test))

y_predxb = xgb_reg.predict(x_test)

print("MAE: ", metrics.mean_absolute_error(y_test, y_predxb))
print("MSE: ", metrics.mean_squared_error(y_test, y_predxb))
print("RMSE: ", metrics.mean_squared_error(y_test, y_predxb, squared=False))
print("R2: ", metrics.r2_score(y_test, y_predxb), "\n")
print("Score: ", xgb_reg.score(x_test, y_predxb))

r2_score(y_test,y_predxb)
```

```
Training score:  0.9994837892421666
Testing score:  0.924760844168243
MAE:  7192.381972579807
MSE:  257995198.92338738
RMSE:  16062.228952526713
R2:  0.924760844168243

Score:  1.0

0.924760844168243
```

5. K-Nearest Neighbour:

```
# Innitiate k neighbour Regressor:

from sklearn.neighbors import KNeighborsRegressor

params = {'n_neighbors':[0,1,2,3,4,5,6,7,8,9]}

knn = KNeighborsRegressor()

knnmodel = GridSearchCV(knn, params, cv=5)
knnmodel.fit(x_train,y_train)
knnmodel.best_params_
```

```
{'n_neighbors': 2}
```

```
knn_best = KNeighborsRegressor(n_neighbors = 1)
knn_best.fit(x_train,y_train)
```

```
KNeighborsRegressor(n_neighbors=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
y_predknn = knn_best.predict(x_test)

r2_score(y_test,y_predknn)
```

```
0.8848945266256593
```

6. AdaBoost Regressor:

```
# Adaboost

from sklearn.ensemble import AdaBoostRegressor

regr = AdaBoostRegressor(random_state=0, n_estimators=100)

regr.fit(x_train,y_train)

ypredada = regr.predict(x_test)

r2_score(y_test,ypredada)
```

```
0.5413019252383878
```

```
# Model Evaluation: MAE , MSE , RMSE

from sklearn.metrics import mean_absolute_error,mean_squared_error

print("MAE: ", metrics.mean_absolute_error(y_test, ypredada))
print("MSE: ", metrics.mean_squared_error(y_test, ypredada))
print("RMSE: ", metrics.mean_squared_error(y_test, ypredada, squared=False))
print("R2: ", metrics.r2_score(y_test, ypredada), "\n")
print("Score: ", regr.score(x_test, ypredada))
```

```
MAE:  35922.77128892319
MSE:  1572876512.7099824
RMSE:  39659.50721718542
R2:  0.5413019252383878

Score:  1.0
```

7. Gradient Boosting Regressor:

```
# Gradient Boosting regressor

from sklearn.ensemble import GradientBoostingRegressor

reg = GradientBoostingRegressor(random_state=0)
reg.fit(x_train,y_train)

ypredgbr = regr.predict(x_test)

r2_score(y_test,ypredgbr)
```

```
0.5413019252383878
```

```
# Model Evaluation: MAE , MSE , RMSE

from sklearn.metrics import mean_absolute_error,mean_squared_error

print("MAE: ", metrics.mean_absolute_error(y_test, ypredgbr))
print("MSE: ", metrics.mean_squared_error(y_test, ypredgbr))
print("RMSE: ", metrics.mean_squared_error(y_test, ypredgbr, squared=False))
print("R2: ", metrics.r2_score(y_test, ypredgbr), "\n")
print("Score: ", reg.score(x_test, ypredgbr))
```

```
MAE:  35922.77128892319
MSE:  1572876512.7099824
RMSE:  39659.50721718542
R2:  0.5413019252383878

Score:  0.207349294266309
```

```
from sklearn.model_selection import KFold , cross_val_score
k_f = KFold(n_splits =3)
k_f

# Cross Validation score to check if model is overfitting:

cross_val_score(reg,x_train,y_train,cv=5) #
```

```
array([0.89705515, 0.85343463, 0.90857903, 0.89285896, 0.80988779])
```

Evaluation Metrics Used

(MSE MAE RMSE R2_SCORE and SCORE for each regression algorithm is used as metrics and indicated in above screenshots)

CROSS VALIDATION USING K-FOLD CROSS VALIDATION

```
from sklearn.model_selection import KFold , cross_val_score
k_f = KFold(n_splits =3)
k_f

# Cross Validation score to check if model is overfitting:

cross_val_score(rf_boosted,x_train,y_train,cv=5) #
```

```
array([0.87082185, 0.84371176, 0.86559031, 0.83156398, 0.69951973])
```

Results/Observations over different Evaluation Metrics

As the data set contains data of flights with wearing departure Times and arrival Times, the most affected and fluctuating is our label that is the price.

Data set also contains a lot of outliers due to which linear regression and regularisation techniques are not performing well.

Errors are very large.

Amongst all, we are getting the best result by random forest.

We have also done the hyper parameter tuning with adjusting an estimators as 100 max depth of the tree as 20 min samples split as 11 and main samples leaf size as 5.

By checking the cross validation score of our boosted random forest model using k fold cross validation we can conclude that our model is prone to over fitting and further we can reduce by removing outliers from the data set.

Creating Model Pipeline

```
# Creating pipeline:
from sklearn import metrics

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error,mean_squared_error



from sklearn.pipeline import Pipeline

pipe1 = Pipeline([('scaler',StandardScaler()),
                  ('pca',PCA(n_components=110)),('base_model1',RandomForestRegressor(n_estimators=100,max_depth=20,
                                 min_samples_split=11,min_samples_leaf=5))])

pipe1.fit(x_train,y_train)

y_pred = pipe1.predict(x_test)

print("MAE: ", metrics.mean_absolute_error(y_test, y_pred))
print("MSE: ", metrics.mean_squared_error(y_test, y_pred))
print("RMSE: ", metrics.mean_squared_error(y_test, y_pred, squared=False))
print("R2: ", metrics.r2_score(y_test, y_pred), "\n")
print("Score: ", pipe1.score(x_test, y_pred))


# Saving regression model to pickle string


import pickle
saved_model1 = pickle.dumps(pipe1)
pipe_pickle1 = pickle.loads(saved_model1)
pipe_pickle1.predict(x_test) # predicting testing data
```

```
MAE:  7774.370543681344
MSE:  262617885.4663187
RMSE:  16205.489362136483
R2:  0.9234127298055868

Score:  1.0

array([ 45428.89192858,  76695.33191447,  25477.44784919,  34703.41431553,
```

Limitations of this work and Scope for Future Work

1. Sample data is small.
2. Intense hyperparameter tuning could have been performed.
3. Overfitting in the model could have been reduced more.

With bigger dataset covering larger time spam and improvements in this model, it can be deployed to find best deals in flight and predict flight fares for different country locations and further can be used to help customers/travellers and companies for curating better deals and offers/strategies giving better options and flexibility to travellers.