DSE Capstone Project

Capstone Final Report Submission

Group 6

Batch details	PGPDSE – BLR – FT – JUNE2022 BATCH
Team members	1. Ujjwal Gupta 2. Barsha Priyadarshini Sahoo 3. Vedpratap Singh Rajpurohit 4. Syamprasad KJ 5. Keerthi vardhan
Domain of Project	Flight Industry
Proposed Project Title	Flight price prediction
Group Number	6
Team Leader	Ujjwal Gupta
Mentor Name	Jatinder Bedi

Date: 29/11/2022

Jatinder Bedi

Signature of the Mentor

Ujjwal Gupta

Signature of the Team Leader

1. Project Overview:

I Business problem statement (GOALS)

1. Business Problem Understanding/ Problem understanding

The number of persons using aircraft has dramatically increased in recent years. Travelers sometimes complain that plane tickets are unreliable and difficult to forecast. If a price is seen today and checked for the same flight tomorrow, it would be a whole different scenario. Prices for airline tickets might be challenging to estimate. In order to construct a model that predicts travel prices using multiple input features, we have been given the pricing of flight tickets for a number of Data that was collected for 50 days, from February 11th to March 31st, 2022.

2. Business Objective

Anyone who has purchased a plane ticket is aware of how costs may change suddenly. Airlines employ highly technical, quasi-academic "revenue management" or "yield management" strategies. Over time, the price of the least costly ticket for a specific date increase or decreases. Typically, this is done in an effort to increase sales based on the following factors:

* Time of purchase patterns (making sure last-minute purchases are expensive)
* Maintaining their desired level of passenger load (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) We could therefore assist travellers to save money on their journeys if we could provide them with information on the best time to purchase their flight tickets based on historical data and also show them relevant trends in the airline business.

The objectives of the project can broadly be laid down by the following questions –

1. Flight Trends Do airfares change frequently?
2. Do they move in small increments or in large jumps?

Thus, Finding the most convenient price for the desired travel itineraries.

3.Approach

o **Raw Data Collection:** For any prediction/classification problem, we need historical data to work with. In this project, past flight prices for each route needs to be collected on a daily basis from “Ease My Trip” website.

o **Cleaning & Preparing Data**: After we have the data, we need to clean & prepare the data according to the model's requirements. In any machine learning problem, this is the step that is the most important and the most time consuming. We used various statistical techniques & logics and implemented them using built-in Python packages.

o **Analyzing & Building Models:** Data preparation is followed by analyzing the data, uncovering hidden trends and then applying various predictive & classification models on the training set to increase the accuracy. Further statistical models’ model can build to increase the accuracy of the ML algorithms for this task.

o **Merging Models & Accuracy Calculation:** Having built various models, we have to test the models on our testing set and calculate the model’s best accuracy in predicting most convenient ticket fare.

4. Conclusions

'Easemytrip' is an internet platform for booking flight tickets, and hence a platform that potential passengers use to buy tickets. A thorough study of the data and thereby performing ‘fare prediction’ model building will aid in the discovery of valuable conclusions that will be of enormous value to passengers.

2. Table of contents:

3. DATASET PREPARATION:

3.1 Reading a Raw CSV file:

The Project contains two separate datasets

1. Economy dataset
2. Business dataset

Economy Dataset

```
1 df_Business.head()
```

	date	airline	ch_code	num_code	dep_time	from	time_taken	Class	stop	arr_time	to	price
0	11-02-2022	Air India	AI	868	18:00	Delhi	02h 00m	Business	non-stop	20:00	Mumbai	25,612
1	11-02-2022	Air India	AI	624	19:00	Delhi	02h 15m	Business	non-stop	21:15	Mumbai	25,612
2	11-02-2022	Air India	AI	531	20:00	Delhi	24h 45m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20:45	Mumbai	42,220
3	11-02-2022	Air India	AI	839	21:25	Delhi	26h 30m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:55	Mumbai	44,450
4	11-02-2022	Air India	AI	544	17:15	Delhi	06h 40m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:55	Mumbai	46,690

Business Dataset

```
1 df_Business.head()
```

	date	airline	ch_code	num_code	dep_time	from	time_taken	Class	stop	arr_time	to	price
0	11-02-2022	Air India	AI	868	18:00	Delhi	02h 00m	Business	non-stop	20:00	Mumbai	25,612
1	11-02-2022	Air India	AI	624	19:00	Delhi	02h 15m	Business	non-stop	21:15	Mumbai	25,612
2	11-02-2022	Air India	AI	531	20:00	Delhi	24h 45m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20:45	Mumbai	42,220
3	11-02-2022	Air India	AI	839	21:25	Delhi	26h 30m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:55	Mumbai	44,450
4	11-02-2022	Air India	AI	544	17:15	Delhi	06h 40m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:55	Mumbai	46,690

3.2 Concatenating Dataset:

To achieve the goal of the project we are concatenating the two datasets (Economy, Business)

```
1 df_full=pd.concat([df_Economy,df_Business])
```

```
1 df_full.tail()
```

	date	airline	ch_code	num_code	dep_time	from	time_taken	Class	stop	arr_time	to	price
93482	31-03-2022	Vistara	UK	822	09:45	Chennai	10h 05m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t...	19:50	Hyderabad	69,265
93483	31-03-2022	Vistara	UK	826	12:30	Chennai	10h 25m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t...	22:55	Hyderabad	77,105
93484	31-03-2022	Vistara	UK	832	07:05	Chennai	13h 50m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t...	20:55	Hyderabad	79,099
93485	31-03-2022	Vistara	UK	828	07:00	Chennai	10h 00m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t...	17:00	Hyderabad	81,585
93486	31-03-2022	Vistara	UK	822	09:45	Chennai	10h 05m	Business	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t\t...	19:50	Hyderabad	81,585

4. Data Preprocessing:

4.1 Data manipulation:

Column Route

Column Route is created by joining ‘from’ and ‘to’ columns.

Route column – This column gives information about the routes in which flights travels.

Dataset after doing above steps

```
df_full.head()
```

	date	airline	ch_code	num_code	dep_time	from	time_taken	Class	stop	arr_time	to	price	route
0	11-02-2022	SpiceJet	SG	8709	18:55	Delhi	02h 10m	Economy	non-stop	21:05	Mumbai	5,953	Delhi-Mumbai
1	11-02-2022	SpiceJet	SG	8157	06:20	Delhi	02h 20m	Economy	non-stop	08:40	Mumbai	5,953	Delhi-Mumbai
2	11-02-2022	AirAsia	I5	764	04:25	Delhi	02h 10m	Economy	non-stop	06:35	Mumbai	5,956	Delhi-Mumbai
3	11-02-2022	Vistara	UK	995	10:20	Delhi	02h 15m	Economy	non-stop	12:35	Mumbai	5,955	Delhi-Mumbai
4	11-02-2022	Vistara	UK	963	08:50	Delhi	02h 20m	Economy	non-stop	11:10	Mumbai	5,955	Delhi-Mumbai

Column Distance:

```
Route = {
'Delhi-Mumbai':'1148', 'Delhi-Bangalore':'1740', 'Delhi-Kolkata':'1305',
       'Delhi-Hyderabad':'1253', 'Delhi-Chennai':'1757', 'Mumbai-Delhi':'1137',
       'Mumbai-Bangalore':'845', 'Mumbai-Kolkata':'1652', 'Mumbai-Hyderabad':'617',
       'Mumbai-Chennai':'1031', 'Bangalore-Delhi':'1740', 'Bangalore-Mumbai':'833',
       'Bangalore-Kolkata':'970', 'Bangalore-Hyderabad':'453', 'Bangalore-Chennai':'289',
       'Kolkata-Delhi':'1305', 'Kolkata-Mumbai':'1652', 'Kolkata-Bangalore':'1560',
       'Kolkata-Hyderabad':'1180', 'Kolkata-Chennai':'1383', 'Hyderabad-Delhi':'1264',
       'Hyderabad-Mumbai':'621', 'Hyderabad-Bangalore':'453', 'Hyderabad-Kolkata':'1209',
       'Hyderabad-Chennai':'507', 'Chennai-Delhi':'1760', 'Chennai-Mumbai':'1031',
       'Chennai-Bangalore':'268', 'Chennai-Kolkata':'1383', 'Chennai-Hyderabad':'514'
}
```

```
df_full['Distance'] = df_full['route'].map(Route)
```

```
df_full.head()
```

	date	airline	ch_code	num_code	dep_time	from	time_taken	Class	stop	arr_time	to	price	route	Distance
0	11-02-2022	SpiceJet	SG	8709	18:55	Delhi	02h 10m	Economy	non-stop	21:05	Mumbai	5,953	Delhi-Mumbai	1148
1	11-02-2022	SpiceJet	SG	8157	06:20	Delhi	02h 20m	Economy	non-stop	08:40	Mumbai	5,953	Delhi-Mumbai	1148
2	11-02-2022	AirAsia	I5	764	04:25	Delhi	02h 10m	Economy	non-stop	06:35	Mumbai	5,956	Delhi-Mumbai	1148
3	11-02-2022	Vistara	UK	995	10:20	Delhi	02h 15m	Economy	non-stop	12:35	Mumbai	5,955	Delhi-Mumbai	1148
4	11-02-2022	Vistara	UK	963	08:50	Delhi	02h 20m	Economy	non-stop	11:10	Mumbai	5,955	Delhi-Mumbai	1148

We have manipulated the dataset by adding a distance column through a dictionary where we imputed the air distances between the source and destination airports.

This distance column may be an important feature in predicting the prices but there are various flights having multiple stops but the dataset provided by easemytrip do not provide any relevant information about the corresponding stops.

As a result of which we are not going to consider the distance column for the future analysis.

4.2 Data cleaning:

Column price

Column price have the thousand separators, so we need to remove it.

BEFORE

price
5,953
5,953
5,956
5,955
5,955

AFTER

price
5953
5953
5956
5955
5955

The datatype of the price variable is converted to int

Column stops

Column stop have values like non-stop, 1 stop, 2 stop, it has been converted to 0,1,2

BEFORE	AFTER
non-stop	0
1-stop	1
2-stop	2

Column time taken

- ‘h’ is replaced as ‘’
- ‘m’ is replaced as ‘’
- ‘ ’ is replaced as ‘.’
- ‘1.01.’ is replaced as ‘1.01’
- ‘1.02.’ is replaced as ‘1.02’
- ‘1.03.’ is replaced as ‘1.03’

Before

time_taken
02h 10m
02h 20m
02h 10m
02h 15m
02h 20m

After

time_taken
02.10
02.20
02.10
02.15
02.20

The datatype of the time taken column changed to float after doing the above process.

The above columns give the information that, the total duration that flight takes to travel from one place to another.

Column Flight

Column Flight code is created by joining ch_code column with num_code column.

Flight code – Code name of the flight

After creating the column flight code, the ch_code and num_code columns are no longer useful. So, we are dropping it from the dataset.

Dataset after Data Cleaning

```
1 df_full.head()
```

	date	airline	Flight_code	dep_time	from	time_taken	Class	stop	arr_time	to	price	route
0	11-02-2022	SpiceJet	SG-8709	18:55	Delhi	2.10	Economy	0	21:05	Mumbai	5953	Delhi-Mumbai
1	11-02-2022	SpiceJet	SG-8157	06:20	Delhi	2.20	Economy	0	08:40	Mumbai	5953	Delhi-Mumbai
2	11-02-2022	AirAsia	I5-764	04:25	Delhi	2.10	Economy	0	06:35	Mumbai	5956	Delhi-Mumbai
3	11-02-2022	Vistara	UK-995	10:20	Delhi	2.15	Economy	0	12:35	Mumbai	5955	Delhi-Mumbai
4	11-02-2022	Vistara	UK-963	08:50	Delhi	2.20	Economy	0	11:10	Mumbai	5955	Delhi-Mumbai

4.3Missing value:

There is no missing value in the dataset.

```
1 df_full.isnull().sum()
```

```
date           0
airline        0
Flight_code    0
dep_time       0
from           0
Class          0
stop           0
arr_time       0
to             0
price          0
route          0
Distance       0
dtype: int64
```

There are no missing values. So, no need to perform any kind of missing value treatment like mean, median mode imputation or iqr method.

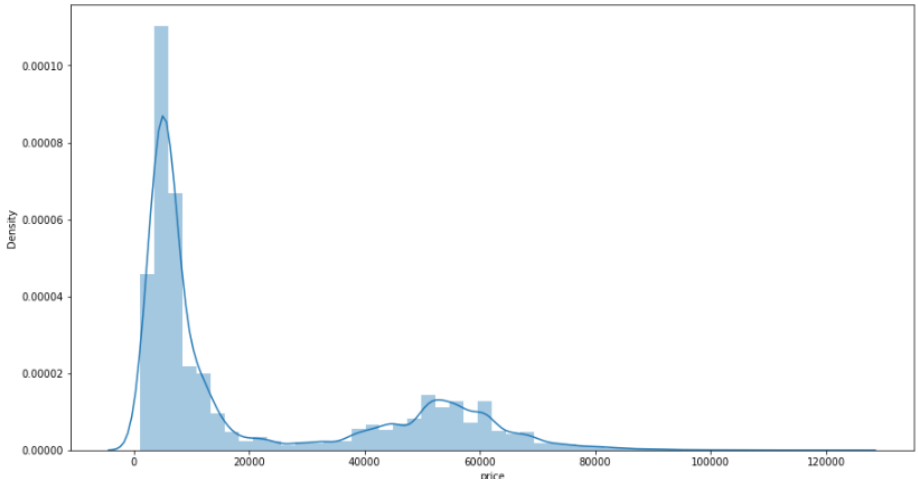
5. EDA:

5.1 Univariant analysis:

In univariant analysis we are getting information about each variable in the dataset.

- Numerical Columns:

  Column price:

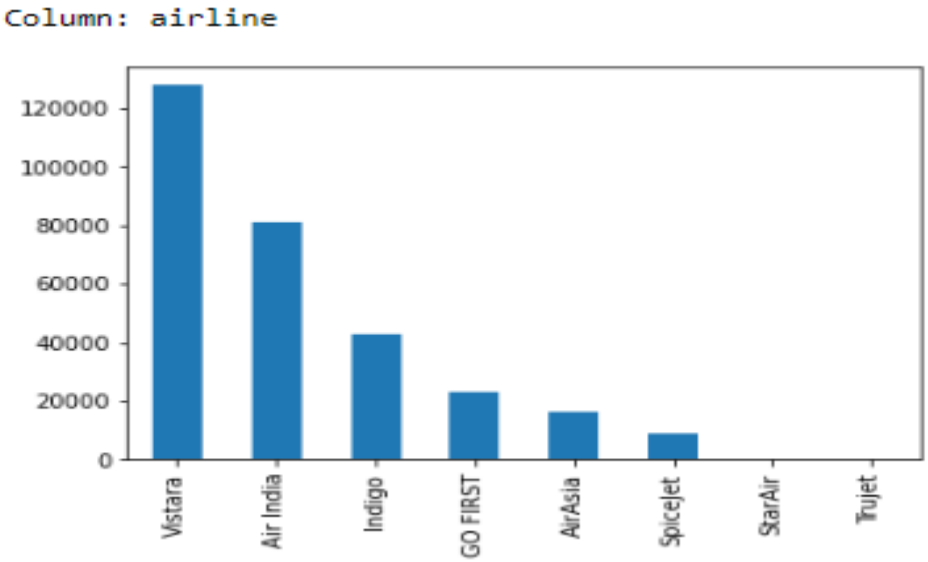


  Skewness: 1.0618915252479717

  As shown in the above dist-plot, the target variable seems to be highly right skewed in nature.

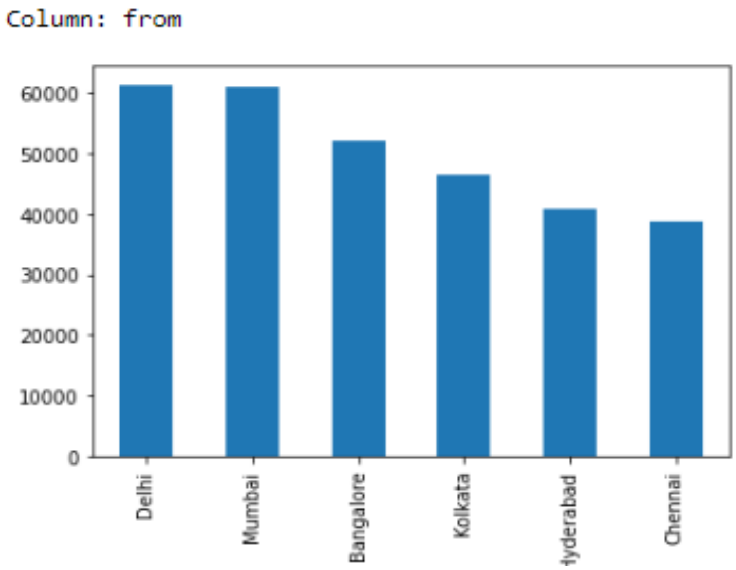
- Categorical columns:

  Column Airline:



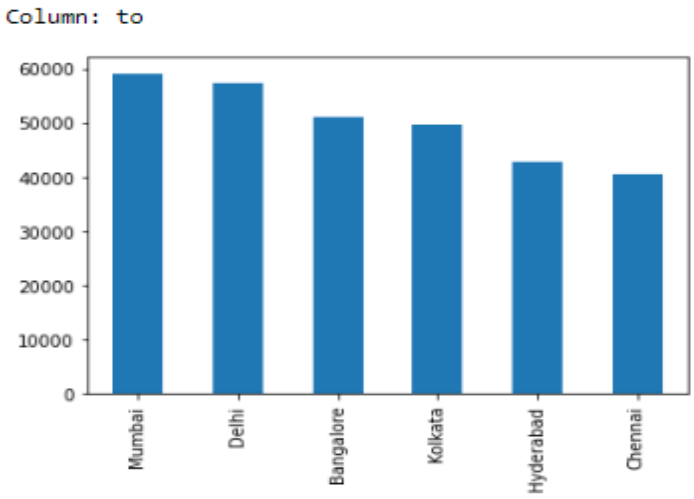
Inference: The Max number of flights are from Vistara and the lowest number of flights are from StarAir and TrueAir.

Column From:



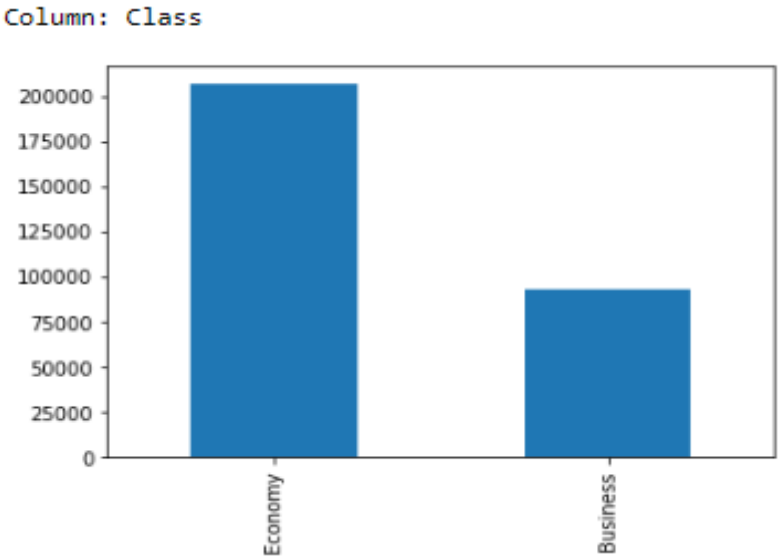
Inference: Flights running from Delhi as well as Mumbai airport are largest in number while Chennai being the lowest.

Column To



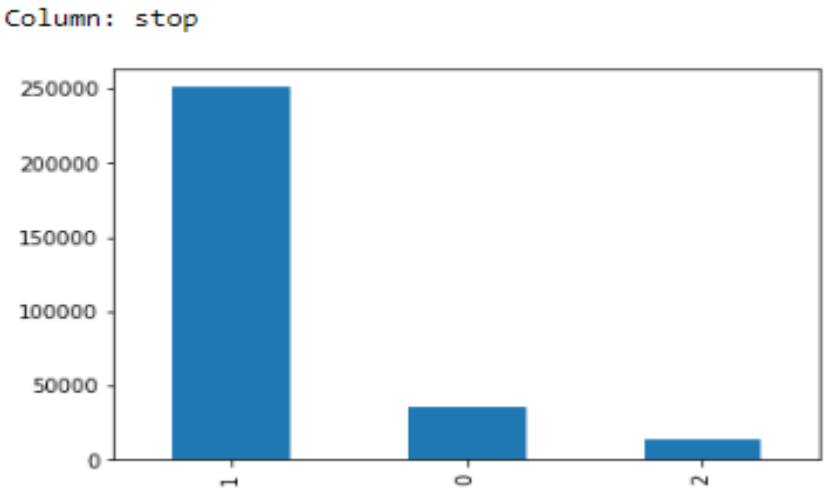
Inference: Flights reaching Mumbai airport are largest in number while in Chennai flights arriving are the lowest.

Column Class:



Inference: Flights with economy classes are larger in number.

Column Stop:



Inference: Majority of flights are having single stop.

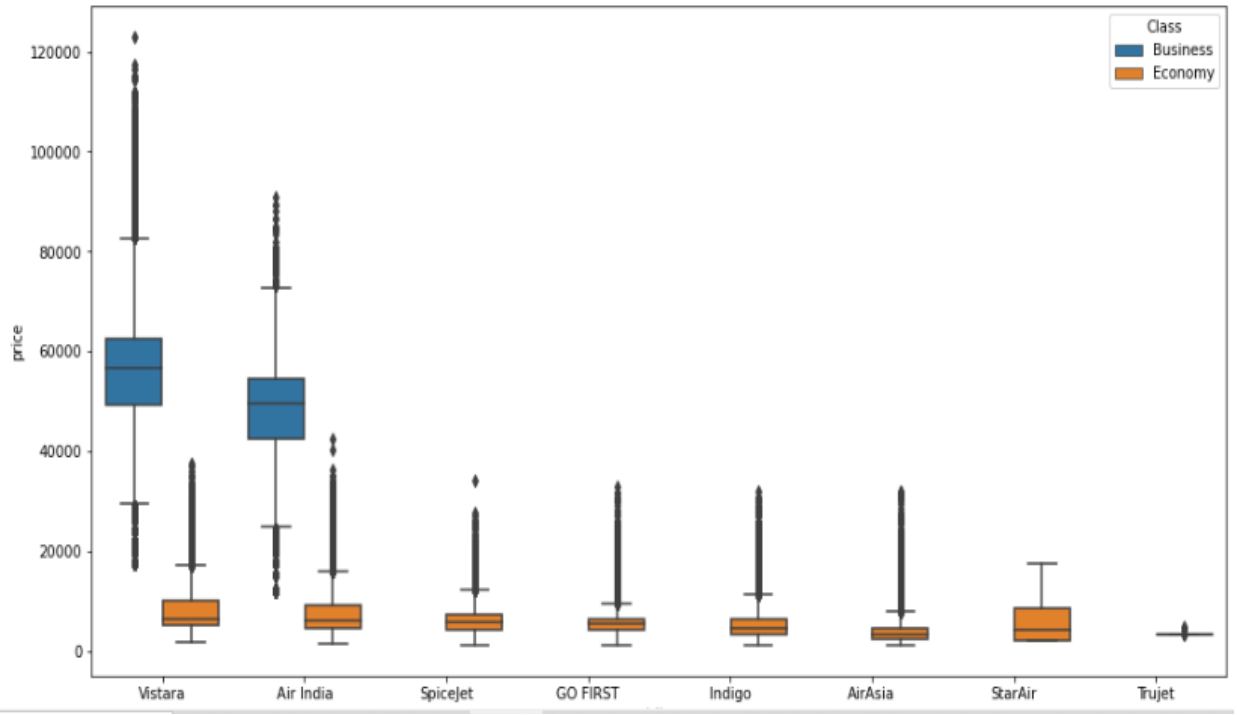
5.2 Bivariant and multivariant analysis:

In bivariant analysis we get the information by comparing each independent variable with target variable.

Columns Airline and Price:

```
plt.figure(figsize=(15,8))
sns.boxplot(data=df_full.sort_values('price',ascending=False),y='price',x='airline',hue='Class')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

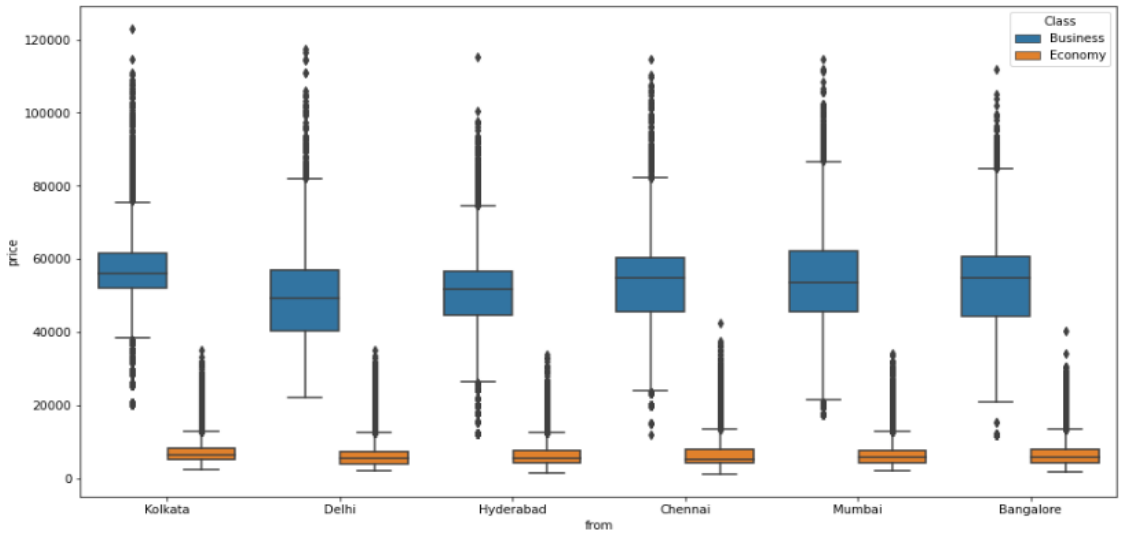


Inference: From above boxplot, we can say that ticket price for Business class in Vistara airways is most expensive.

Columns From and Price:

```
plt.figure(figsize=(15,8))
sns.boxplot(data=df_full.sort_values('price',ascending=False),y='price',x='from',hue='Class')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

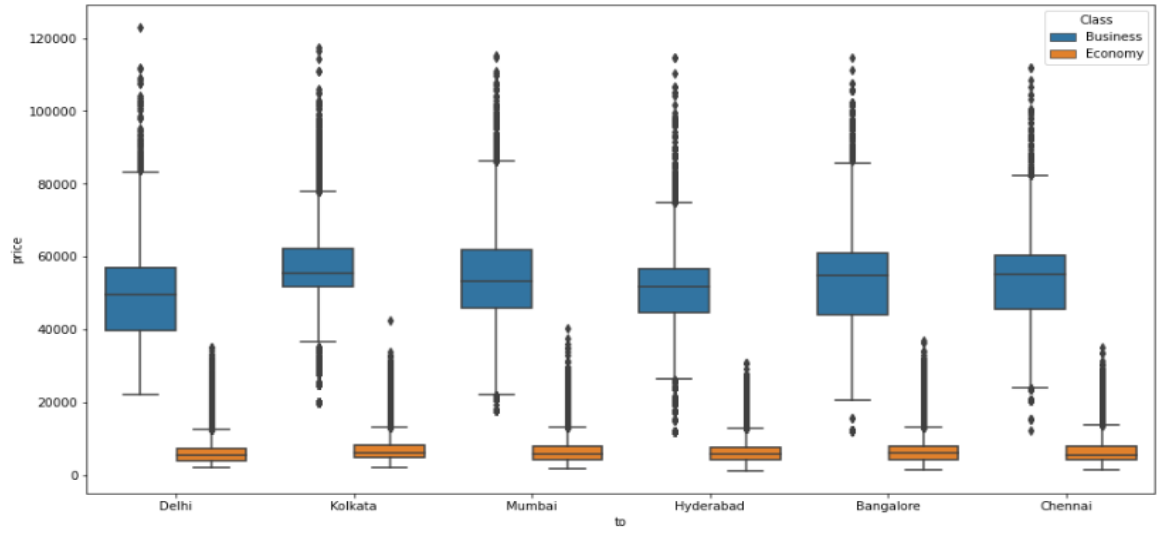


Inference: From above boxplot, we can say that ticket price is most expensive from Kolkata.

Columns To and Price:

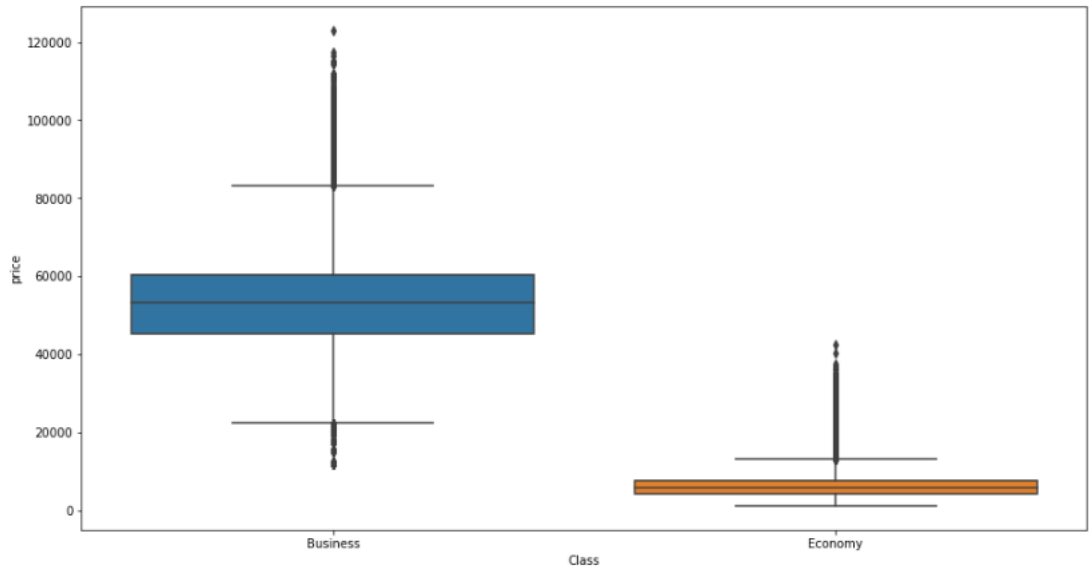
```
plt.figure(figsize=(15,8))
sns.boxplot(data=df_full.sort_values('price',ascending=False),y='price',x='to',hue='Class')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Inference: From above boxplot, we can say that ticket price is most expensive for the flights heading to Kolkata.

Columns Class and Price:

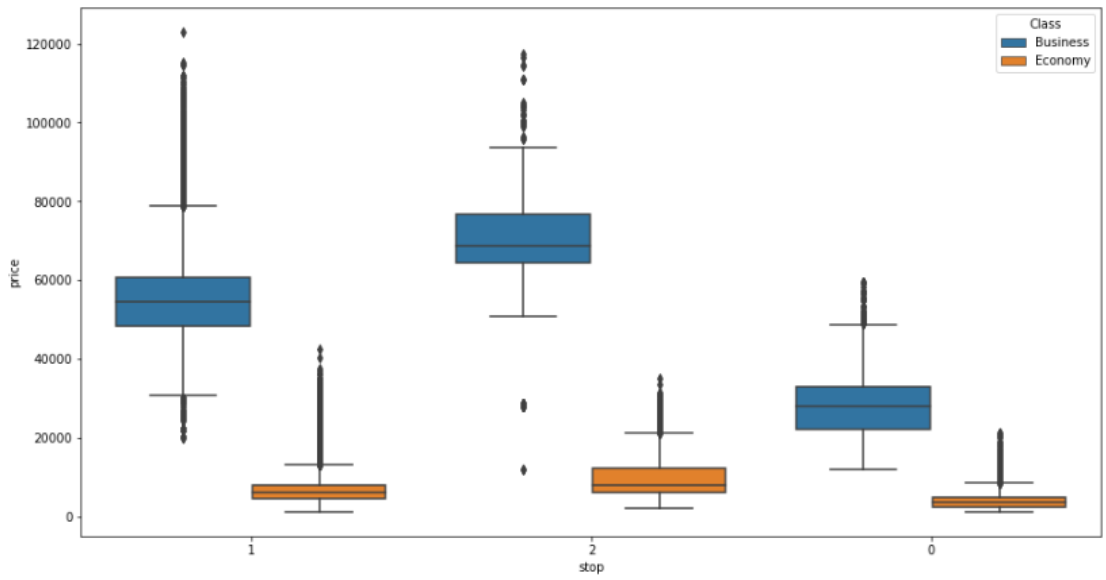


Inference: Flight tickets for business class is most expensive when compare to economy class.

Columns Stop and Price:

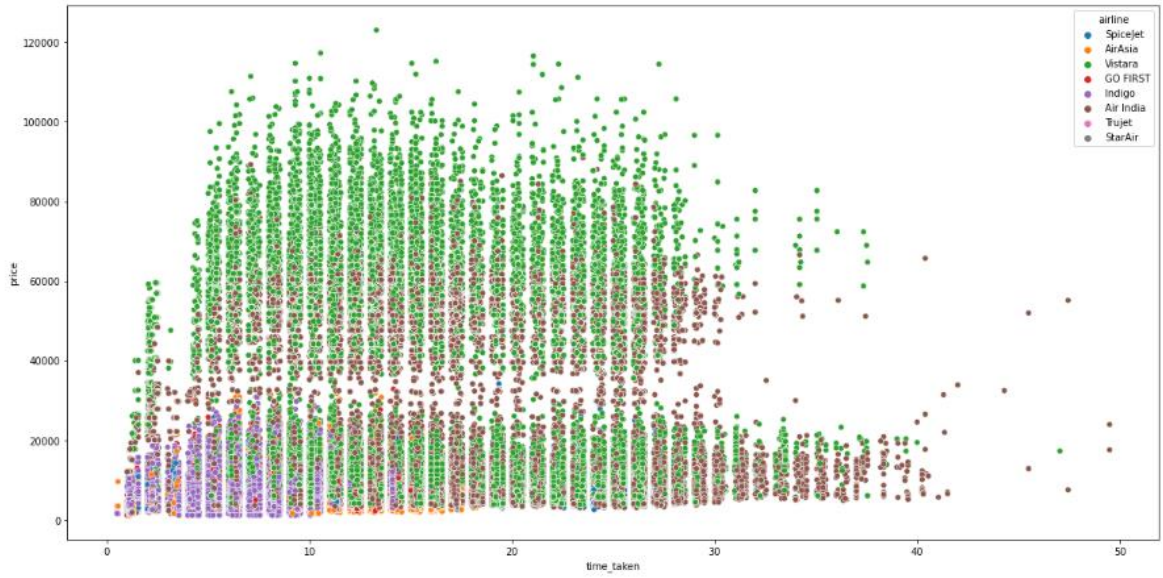
```
1 plt.figure(figsize=(15,8))
2 sns.boxplot(data=df_full.sort_values('price',ascending=False),y='price',x='stop',hue='Class')
3 plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Inference: Ticket prices are most expensive when there are only single stop.

Columns Time taken, price, airline:



Inference: We can see from the above scatter plot that there are many green dots which are related to Vistara,clearly signifies that the airline is most expensive.
Time taken by Air India and Vistara seems to be the largest among all.

Heatmap:

Inference: There is no strong correlation between price, time taken columns.

6. Feature engineering:

Creating a new column called day, week, month
The information for the new columns is extracted from the date column
Column Day – Which day of the month
Column Week – Which week of the year
Column Month – Which month of the year

Dataset after adding new columns:

```
1 df_full.head()
```

	date	airline	Flight_code	dep_time	from	time_taken	Class	stop	arr_time	to	price	route	Day	Week	Month
0	11-02-2022	SpiceJet	SG-8709	18:55	Delhi	2.10	Economy	0	21:05	Mumbai	5953	Delhi-Mumbai	11	6	2
1	11-02-2022	SpiceJet	SG-8157	06:20	Delhi	2.20	Economy	0	08:40	Mumbai	5953	Delhi-Mumbai	11	6	2
2	11-02-2022	AirAsia	I5-764	04:25	Delhi	2.10	Economy	0	06:35	Mumbai	5956	Delhi-Mumbai	11	6	2
3	11-02-2022	Vistara	UK-995	10:20	Delhi	2.15	Economy	0	12:35	Mumbai	5955	Delhi-Mumbai	11	6	2
4	11-02-2022	Vistara	UK-963	08:50	Delhi	2.20	Economy	0	11:10	Mumbai	5955	Delhi-Mumbai	11	6	2

The week column is categorical, so the Datatype of the week column is converted into object from int.

Column Departure time:

Creating a new column called dep_time_in_min by using dep_time column

dep_time_in_min – Hour's format to minutes format

Creating a new column called departure_time by using dep_time_in_min

Departure_time – There is a five category in this column

Early Morning – 0 to 288 (min)

Morning - 288 to 576 (min)

Afternoon - 576 to 864 (min)

Evening - 864 to 1152 (min)

Midnight - 1152 to 0 (min)

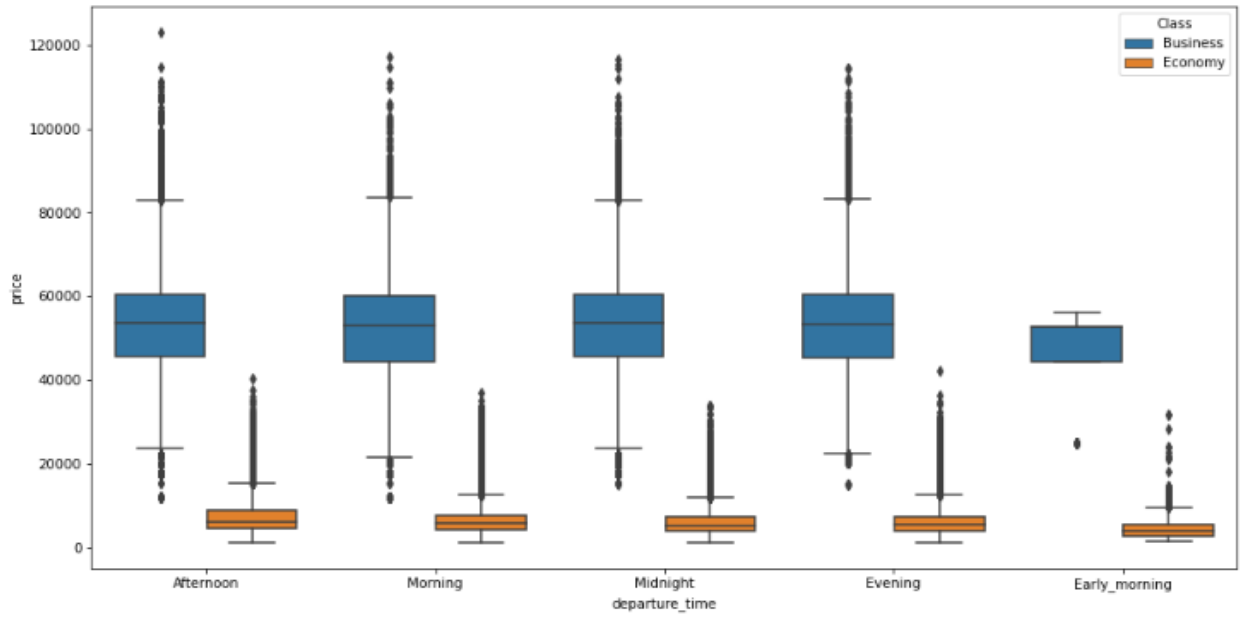
dep_time
18:55
06:20

dep_time_in_min
1135
380

departure_time
Evening
Morning

```
1 plt.figure(figsize=(15,8))
2 sns.boxplot(data=df_full.sort_values('price',ascending=False),x='departure_time',y='price',hue='Class')
```

```
<AxesSubplot:xlabel='departure_time', ylabel='price'>
```



From the above plot it is very evident that Afternoon flights are more costly compared to the Early morning flights

Column arrival_time:

Creating a new column called arr_time_in_min by using arr_time column

arr_time_in_min – Hour's format to minutes format

Creating a new column called arrival_time by using arr_time_in_min

arrival_time – There is a five category in this column

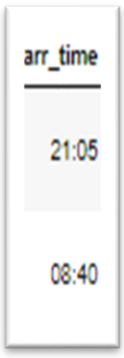
Early Morning – 0 to 288 (min)

Morning - 288 to 576 (min)

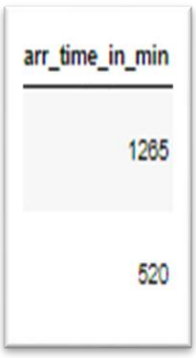
Afternoon - 576 to 864 (min)

Evening - 864 to 1152 (min)

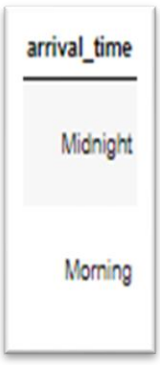
Midnight - 1152 to 0 (min)



arr_time
21:05
08:40



arr_time_in_min
1265
520



arrival_time
Midnight
Morning

Drop the columns:

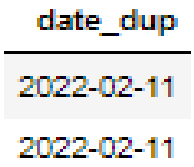
dep_time_in_min, arr_time_in_min, dep_time, arr_time

The above columns are dropped because the new column is created using these columns information, so these columns are no longer useful.

Column day_of_week:

Creating a new column called date_dup from date.

Date_dup – The column is in date format



date_dup
2022-02-11
2022-02-11

The day_of_week column is created using date_dup

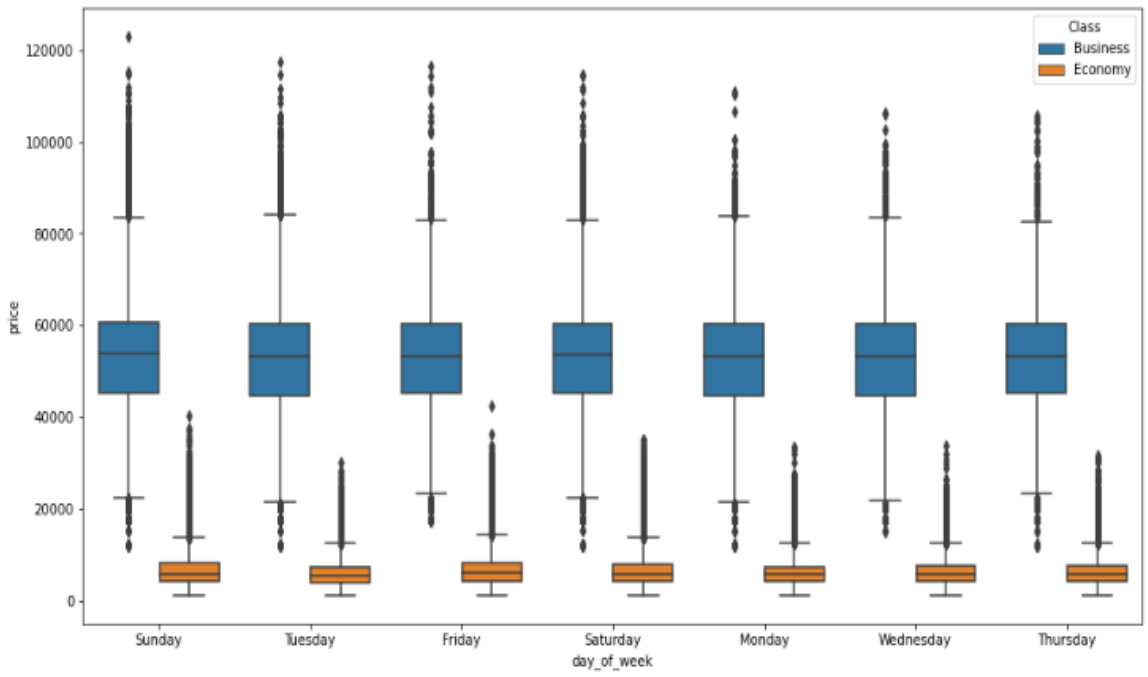
Day_of_week – Gives information about which day of the week

day_of_week
Friday

Friday

```
plt.figure(figsize=(15,8))
sns.boxplot(data=df_full.sort_values('price',ascending=False),x='day_of_week',y='price',hue='Class')
```

```
<AxesSubplot:xlabel='day_of_week', ylabel='price'>
```



From the above plot we can see that on Sunday's tickets are more expensive

We are dropping the columns (date_dup, date) because we already got the information what we want so it no longer helpful.

Drop the unwanted columns like Flight_code, Day.

By checking the standard deviation for the dataset, we came to know that the month column is near to 0.
So, we are dropping the month column because of Standard deviation

The final Dataset before encoding and scaling

```
1 df_full.head()
```

	airline	from	time_taken	Class	stop	to	price	Week	departure_time	arrival_time	day_of_week
0	SpiceJet	Delhi	2.10	Economy	0	Mumbai	5953	6	Evening	Midnight	Friday
1	SpiceJet	Delhi	2.20	Economy	0	Mumbai	5953	6	Morning	Morning	Friday
2	AirAsia	Delhi	2.10	Economy	0	Mumbai	5956	6	Early_morning	Morning	Friday
3	Vistara	Delhi	2.15	Economy	0	Mumbai	5955	6	Afternoon	Afternoon	Friday
4	Vistara	Delhi	2.20	Economy	0	Mumbai	5955	6	Morning	Afternoon	Friday

7. Encoding and Scaling:

7.1 Scaling:

There is only one numerical column(time_taken), So no need to do scaling

7.2 Encoding:

(N-1) dummy encoding (using pandas)

It is used to create dummy variables from a categorical variable

For a categorical variable that can take k values, k-1 dummy variables are created

These are the columns which we got after doing dummy encoding

```
1 dummy_var.columns
```

```
Index(['airline_AirAsia', 'airline_GO FIRST', 'airline_Indigo',
       'airline_SpiceJet', 'airline_StarAir', 'airline_Trujet',
       'airline_Vistara', 'from_Chennai', 'from_Delhi', 'from_Hyderabad',
       'from_Kolkata', 'from_Mumbai', 'Class_Economy', 'stop_1', 'stop_2',
       'to_Chennai', 'to_Delhi', 'to_Hyderabad', 'to_Kolkata', 'to_Mumbai',
       'Week_7', 'Week_8', 'Week_9', 'Week_10', 'Week_11', 'Week_12',
       'Week_13', 'departure_time_Early_morning', 'departure_time_Evening',
       'departure_time_Midnight', 'departure_time_Morning',
       'arrival_time_Early_morning', 'arrival_time_Evening',
       'arrival_time_Midnight', 'arrival_time_Morning', 'day_of_week_Monday',
       'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thursday',
       'day_of_week_Tuesday', 'day_of_week_Wednesday'],
      dtype='object')
```

The final dataset are formed by concatinating the numerical and categorical columns.

The final Dataset:

```
1 X.head()
```

	time_taken	airline_AirAsia	airline_GO FIRST	airline_Indigo	airline_SpiceJet	airline_StarAir	airline_Trujet	airline_Vistara	from_Chennai	from_Delhi	...	arrival_time
0	2.10	0	0	0	1	0	0	0	0	1	...	
1	2.20	0	0	0	1	0	0	0	0	1	...	
2	2.10	1	0	0	0	0	0	0	0	1	...	
3	2.15	0	0	0	0	0	0	1	0	1	...	
4	2.20	0	0	0	0	0	0	1	0	1	...	

5 rows × 42 columns

There are totally 300261 rows and 43 columns.

8. Train-test split:

The train_test_split() method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into X_train, X test, y_train, and y_test. X_train and y_train sets are used for training and fitting the model.

After splitting the shape of the X_train, X test, y_train, and y_test is:

```
X_train (210182, 43)
y_train (210182,)
X_test (90079, 43)
y_test (90079,)
```

The columns of X_train, X_test:

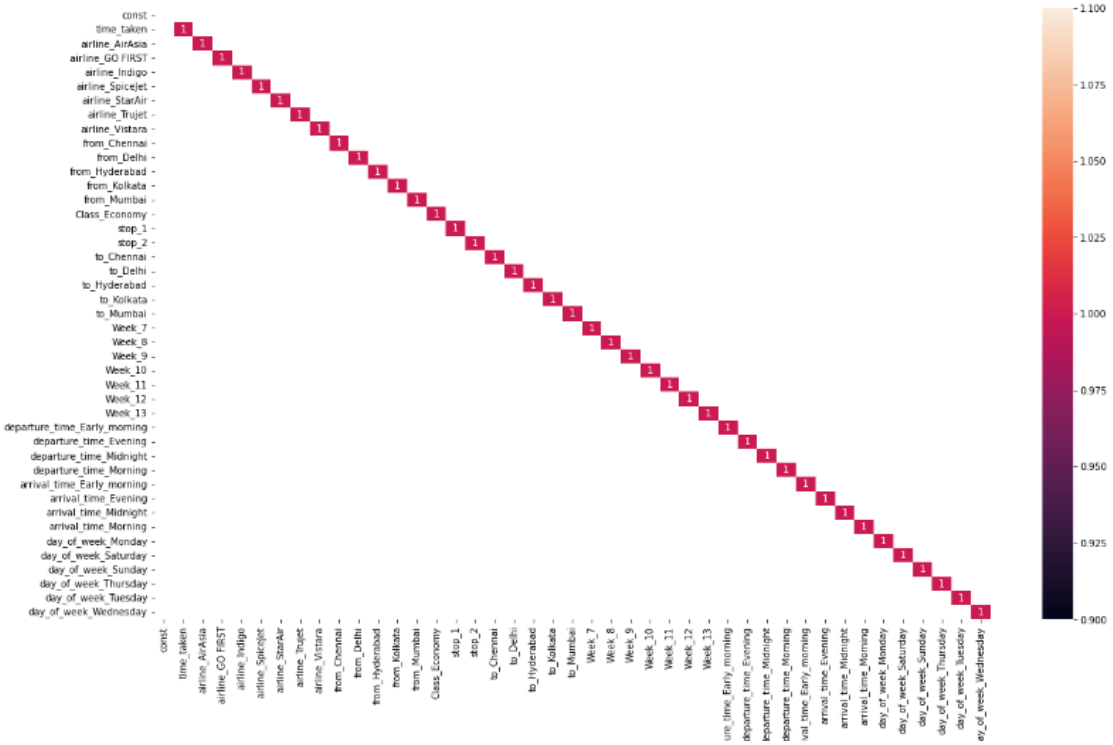
```
Index(['const', 'time_taken', 'airline_AirAsia', 'airline_GO FIRST',
       'airline_Indigo', 'airline_SpiceJet', 'airline_StarAir',
       'airline_Trujet', 'airline_Vistara', 'from_Chennai', 'from_Delhi',
       'from_Hyderabad', 'from_Kolkata', 'from_Mumbai', 'Class_Economy',
       'stop_1', 'stop_2', 'to_Chennai', 'to_Delhi', 'to_Hyderabad',
       'to_Kolkata', 'to_Mumbai', 'Week_7', 'Week_8', 'Week_9', 'Week_10',
       'Week_11', 'Week_12', 'Week_13', 'departure_time_Early_morning',
       'departure_time_Evening', 'departure_time_Midnight',
       'departure_time_Morning', 'arrival_time_Early_morning',
       'arrival_time_Evening', 'arrival_time_Midnight', 'arrival_time_Morning',
       'day_of_week_Monday', 'day_of_week_Saturday', 'day_of_week_Sunday',
       'day_of_week_Thursday', 'day_of_week_Tuesday', 'day_of_week_Wednesday'],
      dtype='object')
```

9. Statistical test:

The two-sample t-test is used to determine if two population means are equal. A common application is to test if a new process or treatment is superior to a current process or treatment. There are several variations on this test. The data may either be paired or not paired.
Null hypothesis: u1=u2 (mean of both the sample is same, i.e., samples belong to same population)
Alternate hypothesis: u1! =u2 (mean of both the sample is different, i.e., samples do not belong to same population)
Ttest_indResult(statistic=0.481467618703109, pvalue=0.6301845393990948)
Here as pvalue > 0.05, so we fail to reject null hypothesis. Thus we can say that both the samples belong to same population.

10. Multicollinearity check:

Multicollinearity occurs when independent variables in a regression model are correlated. This correlation is a problem because independent variables should be independent. If the degree of correlation between variables is high enough, it can cause problems when you fit the model and interpret the results.



Interpretation: The above heatmap, show the no variables are having a strong positive/negative correlation. The variables with dark shade represent the strong positive correlation.

11.Model building:

11.1 Building MLR model using OLS Techniques:

```
MLR_Model=sm.OLS(y_train.values.reshape(-1,1),X_train).fit()
print(MLR_Model.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.914
Model:	OLS	Adj. R-squared:	0.914
Method:	Least Squares	F-statistic:	5.323e+04
Date:	Sat, 26 Nov 2022	Prob (F-statistic):	0.00
Time:	16:26:24	Log-Likelihood:	-2.1485e+06
No. Observations:	210182	AIC:	4.297e+06
Df Residuals:	210139	BIC:	4.298e+06
Df Model:	42		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.077e+04	123.820	409.997	0.000	5.05e+04	5.1e+04
time_taken	33.0493	2.751	12.013	0.000	27.657	38.441
airline_AirAsia	-232.3165	74.721	-3.109	0.002	-378.767	-85.866
airline_GO FIRST	1624.4165	64.114	25.336	0.000	1498.754	1750.079
airline_Indigo	1739.2651	55.654	31.251	0.000	1630.185	1848.345
airline_SpiceJet	2260.9821	90.565	24.965	0.000	2083.478	2438.487
airline_StarAir	5921.0663	1005.463	5.889	0.000	3950.384	7891.749
airline_Trujet	-1369.6430	1306.782	-1.048	0.295	-3930.904	1191.618
airline_Vistara	3983.6345	36.716	108.498	0.000	3911.672	4055.598
from_Chennai	-25.7259	54.334	-0.473	0.636	-132.220	80.768
from_Delhi	-1497.1752	49.322	-30.355	0.000	-1593.844	-1400.506
from_Hyderabad	-1698.8218	53.825	-31.562	0.000	-1804.317	-1593.327
from_Kolkata	1592.1222	52.361	30.406	0.000	1489.495	1694.749
from_Mumbai	-221.0593	49.105	-4.502	0.000	-317.303	-124.815
Class_Economy	-4.493e+04	35.459	-1267.141	0.000	-4.5e+04	-4.49e+04
stop_1	7704.6932	53.992	142.701	0.000	7598.870	7810.516
stop_2	9848.5983	90.541	108.775	0.000	9671.140	1e+04
to_Chennai	-187.2388	54.005	-3.467	0.001	-293.088	-81.390
to_Delhi	-1575.4634	50.567	-31.156	0.000	-1674.573	-1476.353
to_Hyderabad	-1764.8780	53.470	-33.007	0.000	-1869.677	-1660.079
to_Kolkata	1439.3409	51.645	27.870	0.000	1338.117	1540.565
to_Mumbai	-60.1501	49.707	-1.210	0.226	-157.575	37.274
Week_7	-4476.1963	90.432	-49.498	0.000	-4653.441	-4298.952
Week_8	-6663.2148	89.568	-74.393	0.000	-6838.766	-6487.663
Week_9	-8695.6355	89.464	-97.197	0.000	-8870.982	-8520.289
Week_10	-8908.1215	89.471	-99.564	0.000	-9083.483	-8732.760
Week_11	-8994.7192	89.431	-100.577	0.000	-9170.002	-8819.437
Week_12	-9251.4021	89.471	-103.401	0.000	-9426.763	-9076.041
Week_13	-9436.4570	99.290	-95.040	0.000	-9631.063	-9241.851
departure_time_Early_morning	1164.1474	165.300	7.043	0.000	840.163	1488.131
departure_time_Evening	-154.1619	42.850	-3.598	0.000	-238.146	-70.178
departure_time_Midnight	-31.7269	45.824	-0.692	0.489	-121.541	58.087
departure_time_Morning	-315.8024	39.915	-7.912	0.000	-394.035	-237.569
arrival_time_Early_morning	562.0150	78.585	7.152	0.000	407.991	716.039
arrival_time_Evening	428.0308	46.737	9.158	0.000	336.428	519.634
arrival_time_Midnight	562.5260	41.578	13.529	0.000	481.034	644.018
arrival_time_Morning	-364.2109	51.050	-7.134	0.000	-464.268	-264.154
day_of_week_Monday	74.3926	55.650	1.337	0.181	-34.680	183.466
day_of_week_Saturday	-215.5648	55.822	-3.862	0.000	-324.974	-106.155
day_of_week_Sunday	-178.5917	55.652	-3.209	0.001	-287.668	-69.516
day_of_week_Thursday	-23.6153	55.711	-0.424	0.672	-132.808	85.578
day_of_week_Tuesday	12.8531	55.743	0.231	0.818	-96.403	122.109
day_of_week_Wednesday	-14.9602	55.590	-0.269	0.788	-123.914	93.994

```
Omnibus:                    26581.460   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           242952.793
Skew:                           0.286   Prob(JB):                         0.00
Kurtosis:                       8.236   Cond. No.                     1.27e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.27e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Interpretation:

1)From the above build model we can see that Condition Number is more than 1000, which implies that there is high multicollinearity present.

2)The summary output shows that the value of the Durbin-Watson is approximately equal to 2, which means there is no-autocorrelation.

3)The p-value came from JB Test is less than 0.05, which shows that errors are not normal in nature.

11.2 Checking which features are multicollinear with the help of VIF technique:

	VIF_Factor	Features
0	72.740552	const
1	4.884097	Week_12
2	4.878726	Week_11
3	4.872988	Week_9
4	4.865496	Week_10
5	4.815639	Week_8
6	4.417305	Week_7
7	3.488750	Week_13
8	1.961269	arrival_time_Midnight
9	1.902651	stop_1
10	1.879899	from_Delhi
11	1.872402	to_Delhi
12	1.858796	to_Mumbai
13	1.856849	time_taken
14	1.848336	from_Mumbai
15	1.842334	day_of_week_Wednesday
16	1.841173	day_of_week_Monday
17	1.838537	day_of_week_Thursday
18	1.836381	day_of_week_Tuesday
19	1.805645	airline_Indigo
20	1.783593	day_of_week_Sunday
21	1.774385	day_of_week_Saturday
22	1.768863	arrival_time_Evening
23	1.742310	to_Kolkata
24	1.698132	from_Kolkata
25	1.656279	to_Hyderabad
26	1.650393	stop_2
27	1.613917	from_Hyderabad
28	1.602586	to_Chennai
29	1.601637	departure_time_Midnight
30	1.585268	arrival_time_Morning
31	1.578765	departure_time_Morning
32	1.573854	departure_time_Evening
33	1.569881	from_Chennai
34	1.564204	airline_Vistara
35	1.390024	airline_GO FIRST
36	1.337803	airline_AirAsia
37	1.326951	arrival_time_Early_morning
38	1.279981	Class_Economy
39	1.141729	airline_SpiceJet
40	1.066775	departure_time_Early_morning
41	1.003905	airline_StarAir
42	1.002132	airline_Trujet

Interpretation:

From above table we can infer that the VIF for every feature seems lesser than 5. As a result of which there is the presence of only moderate multicollinearity. So we can ignore and move forward with model building.

11.3 Heteroskedasticity:

It tests whether the variance of the errors from a regression is dependent on the values of the independent variables. It is a $\chi 2$ test. You can perform the test using the fitted values of the model, the predictors in the model and a subset of the independent variables.

```
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
```

```
name = ['f-value','p-value']
test = sms.het_breuschpagan(MLR_Model.resid, MLR_Model.model.exog)
lzip(name, test[2:])
```

```
[('f-value', 1361.8654671312615), ('p-value', 0.0)]
```

Interpretation: We observe that the p-value is less than 0.05; thus, we conclude that there is heteroskedasticity present in the data.

11.4 Normality in residual:

```
from scipy import stats
from scipy.stats import shapiro
```

```
#With the help of Shapiro-wilk test we are looking for normality in Residuals:
stat, p_value = shapiro(MLR_Model.resid)


print('Test statistic:', stat)
print('P-Value:', p_value)
```

```
Test statistic: 0.9118881821632385
P-Value: 0.0
```

Interpretation: From the above test we can see that the p-value is 0.0 (less than 0.05), thus we can say that the residuals are not normally distributed.

11.5 MLR model:

The multiple linear regression (MLR) model assumes that in addition to the p independent x-variables, a response variable y is measured, which can be explained as a linear combination of the x-variables.

Building a MLR model on a training dataset:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

```
# initiate linear regression model
linreg = LinearRegression()


MLR_model = linreg.fit(X_train, y_train)


MLR_model.score(X_train, y_train)
```

```
0.9140865665498183
```

```
print('RMSE on train set: ', get_train_rmse(MLR_model))
print('RMSE on test set: ', get_test_rmse(MLR_model))
difference = abs(get_test_rmse(MLR_model) - get_train_rmse(MLR_model))
print('Difference between RMSE on train and test set: ', difference)
```

```
RMSE on train set:  6655.1247
RMSE on test set:  6647.4941
Difference between RMSE on train and test set:  7.630600000000413
```

Interpretation:

RMSE on the training set is **6655.1247**, while on the test set it is **6647.4941**. We can see that there is very minimal difference in the RMSE of the train and the test set. This implies that our model is **normal-fitted**.

11.6 Linear Regression (using SGD):

Linear model fitted by minimizing a regularized empirical loss with SGD.

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

Build MLR model using SGD method.

```
from sklearn.linear_model import SGDRegressor

# instantiate the SGDRegressor
# set 'random_state' to generate the same dataset each time you run the code
sgd = SGDRegressor(random_state = 10)

# build the model on train data
# use fit() to fit the model
linreg_with_SGD = sgd.fit(X_train, y_train)

# print RMSE for train set
# call the function 'get_train_rmse'
print('RMSE on train set:', get_train_rmse(linreg_with_SGD))

# print RMSE for test set
# call the function 'get_test_rmse'
print('RMSE on test set:', get_test_rmse(linreg_with_SGD))
```

```
RMSE on train set: 6716.6479
RMSE on test set: 6707.8989
```

Interpretation:

RMSE on the **training** set is **6716.6479**, while on the **test** set it is **6707.8989.**

11.7 Ridge Regression:

Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated. It has been used in many fields including econometrics, chemistry, and engineering.

Build regression model using Ridge Regression for alpha = 1.

Build regression model using Ridge Regression for alpha = 1.

```
from sklearn.linear_model import Ridge

# use Ridge() to perform ridge regression
# 'alpha' assigns the regularization strength to the model
# 'max_iter' assigns maximum number of iterations for the model to run
ridge = Ridge(alpha = 1, max_iter = 500)

# fit the model on train set
ridge.fit(X_train, y_train)

# print RMSE for test set
# call the function 'get_test_rmse'
print('RMSE on test set:', get_test_rmse(ridge))
```

```
RMSE on test set: 6647.4899
```

Interpretation:

The **test RMSE** is **6647.4899**, There is much difference while comparing with other models.

Build regression model using Ridge Regression for alpha = 2.

Build regression model using Ridge Regression for alpha = 2.

```
# use Ridge() to perform ridge regression
# 'alpha' assigns the regularization strength to the model
# 'max_iter' assigns maximum number of iterations for the model to run
ridge = Ridge(alpha = 2, max_iter = 500)

# fit the model on train set
ridge.fit(X_train, y_train)


# print RMSE for test set
# call the function 'get_test_rmse'
print('RMSE on train set: ', get_train_rmse(ridge))
print('RMSE on test set: ', get_test_rmse(ridge))
difference = abs(get_test_rmse(ridge) - get_train_rmse(ridge))
print('Difference between RMSE on train and test set: ', difference)
```

```
RMSE on train set:  6655.1268
RMSE on test set:  6647.4867
Difference between RMSE on train and test set:  7.64009999999962
```

Interpretation:

The **test RMSE** is **6647.4867**. By changing the alpha value we didn't get any diffrence in the Test RMSE.

11.8 Lasso regression:

Lasso regression is also called Penalized regression method. This method is usually used in machine learning for the selection of the subset of variables. It provides greater prediction accuracy as compared to other regression models. Lasso Regularization helps to increase model interpretation.

```
from sklearn.linear_model import Lasso
# use Lasso() to perform lasso regression
# 'alpha' assigns the regularization strength to the model
# 'max_iter' assigns maximum number of iterations for the model to run
lasso = Lasso(alpha = 0.01, max_iter = 500)

# fit the model on train set
lasso.fit(X_train, y_train)

# print RMSE for test set
# call the function 'get_test_rmse'
print('RMSE on train set: ', get_train_rmse(lasso))
print('RMSE on test set: ', get_test_rmse(lasso))
difference = abs(get_test_rmse(lasso) - get_train_rmse(lasso))
print('Difference between RMSE on train and test set: ', difference)
```

```
RMSE on train set:  6655.1248
RMSE on test set:  6647.491
Difference between RMSE on train and test set:  7.63379999999961
```

Interpretation:

The **Test RMSE: 6647.491**

11.9 Elastic net regression:

Elastic net linear regression uses the penalties from both the lasso and ridge techniques to regularize regression models. The technique combines both the lasso and ridge regression

methods by learning from their shortcomings to improve the regularization of statistical models.

Elastic Net Regression

```
from sklearn.linear_model import ElasticNet
# use ElasticNet() to perform Elastic Net regression
# 'alpha' assigns the regularization strength to the model
# 'l1_ratio' is the ElasticNet mixing parameter
# 'l1_ratio = 0' performs Ridge regression
# 'l1_ratio = 1' performs Lasso regression
# pass number of iterations to 'max_iter'
enet = ElasticNet(alpha = 0.1, l1_ratio = 0.01, max_iter = 500)

# fit the model on train data
enet.fit(X_train, y_train)


# print RMSE for test set
# call the function 'get_test_rmse'
print('RMSE on train set: ', get_train_rmse(enet))
print('RMSE on test set: ', get_test_rmse(enet))
difference = abs(get_test_rmse(enet) - get_train_rmse(enet))
print('Difference between RMSE on train and test set: ', difference)
```

```
RMSE on train set:  9580.4613
RMSE on test set:  9562.7331
Difference between RMSE on train and test set:  17.72820000000138
```

Interpretation:
Test RMSE: 9562.7331

11.10 Grid search CV:

GridSearchCV is a technique for finding the optimal parameter values from a given set of parameters in a grid. It's essentially a cross-validation technique. The model as well as the parameters must be entered. After extracting the best parameter values, predictions are made. Grid search CV for Ridge regression:

Find optimal value of alpha for Ridge Regression

```
from sklearn.model_selection import GridSearchCV
# create a dictionary with hyperparameters and its values
# 'alpha' assigns the regularization strength to the model
# 'max_iter' assigns maximum number of iterations for the model to run
tuned_paramaters = [{'alpha':[1e-15, 1e-10, 1e-8, 1e-4,1e-3, 1e-2, 0.1, 1, 5, 10, 20, 40, 60, 80, 100]}]

# initiate the ridge regression model
ridge = Ridge()

# use GridSearchCV() to find the optimal value of alpha
# estimator: pass the ridge regression model
# param_grid: pass the list 'tuned_parameters'
# cv: number of folds in k-fold i.e. here cv = 10
ridge_grid = GridSearchCV(estimator = ridge,
                          param_grid = tuned_paramaters,
                          cv = 10)

# fit the model on X_train and y_train using fit()
ridge_grid.fit(X_train, y_train)

# get the best parameters
print('Best parameters for Ridge Regression: ', ridge_grid.best_params_, '\n')

# print the RMSE for test set using the model having optimal value of alpha
print('RMSE on train set: ', get_train_rmse(ridge_grid))
print('RMSE on test set: ', get_test_rmse(ridge_grid))
difference = abs(get_test_rmse(ridge_grid) - get_train_rmse(ridge_grid))
print('Difference between RMSE on train and test set: ', difference)
```

```
Best parameters for Ridge Regression:  {'alpha': 0.1}

RMSE on train set:  6655.1247
RMSE on test set:  6647.4937
Difference between RMSE on train and test set:  7.631000000000313
```

Interpretation:

Test RMSE: 6647.49

Grid search for lasso regression:

Find optimal value of alpha for Lasso Regression

```
# create a dictionary with hyperparameters and its values
# 'alpha' assigns the regularization strength to the model
# 'max_iter' assigns maximum number of iterations for the model to run
tuned_paramaters = [{'alpha':[1e-15, 1e-10, 1e-8, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20]}]

# 'max_iter':100,500,1000,1500,2000

# initiate the lasso regression model
lasso = Lasso()

# use GridSearchCV() to find the optimal value of alpha
# estimator: pass the lasso regression model
# param_grid: pass the list 'tuned_parameters'
# cv: number of folds in k-fold i.e. here cv = 10
lasso_grid = GridSearchCV(estimator = lasso,
                          param_grid = tuned_paramaters,
                          cv = 10)

# fit the model on X_train and y_train using fit()
lasso_grid.fit(X_train, y_train)

# get the best parameters
print('Best parameters for Lasso Regression: ', lasso_grid.best_params_, '\n')

# print the RMSE for the test set using the model having optimal value of alpha
print('RMSE on train set: ', get_train_rmse(lasso_grid))
print('RMSE on test set: ', get_test_rmse(lasso_grid))
difference = abs(get_test_rmse(lasso_grid) - get_train_rmse(lasso_grid))
print('Difference between RMSE on train and test set: ', difference)
```

```
Best parameters for Lasso Regression:  {'alpha': 0.01}

RMSE on train set:  6655.1248
RMSE on test set:  6647.491
Difference between RMSE on train and test set:  7.63379999999961
```

Interpretation:

Test RMSE:6647.491

Grid search CV for Elastic net regression:

```
# create a dictionary with hyperparameters and its values
# 'alpha' assigns the regularization strength to the model
# 'l1_ratio' is the ElasticNet mixing parameter
# 'max_iter' assigns maximum number of iterations for the model to run
tuned_paramaters = [{'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 20, 40, 60],
                     'l1_ratio':[0.0001, 0.0002, 0.001, 0.01, 0.1, 0.2]}]

# initiate the elastic net regression model
enet = ElasticNet()

# use GridSearchCV() to find the optimal value of alpha and l1_ratio
# estimator: pass the elastic net regression model
# param_grid: pass the list 'tuned_parameters'
# cv: number of folds in k-fold i.e. here cv = 10
enet_grid = GridSearchCV(estimator = enet,
                         param_grid = tuned_paramaters,
                         cv = 10)



# fit the model on X_train and y_train using fit()
enet_grid.fit(X_train, y_train)

# get the best parameters
print('Best parameters for Elastic Net Regression: ', enet_grid.best_params_, '\n')

# print the RMSE for the test set using the model having optimal value of alpha and l1-ratio
print('RMSE on train set: ', get_train_rmse(enet_grid))
print('RMSE on test set: ', get_test_rmse(enet_grid))
difference = abs(get_test_rmse(enet_grid) - get_train_rmse(enet_grid))
print('Difference between RMSE on train and test set: ', difference)
```

```
Best parameters for Elastic Net Regression:  {'alpha': 0.0001, 'l1_ratio': 0.2}

RMSE on train set:  6655.23
RMSE on test set:  6647.5242
Difference between RMSE on train and test set:  7.705799999999726
```

Interpretation:

Test RMSE:6647.52

11.11 Score card:

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Ridge Regression (with alpha = 1)	1	-	0.914087	0.914069	6647.489900	44.980602
1	Ridge Regression (with alpha = 1)	1	-	0.914087	0.914069	6647.489900	44.980602
2	Lasso Regression	0.01	-	0.914087	0.914069	6647.491000	44.982941
3	Lasso Regression	0.01	-	0.914087	0.914069	6647.491000	44.982941
4	Lasso Regression (using GridSearchCV)	0.010000	-	0.914087	0.914069	6647.491000	44.982941
5	Linear Regression	-	-	0.914087	0.914069	6647.494100	44.984405
6	Elastic Net Regression (using GridSearchCV)	0.000100	0.200000	0.914084	0.914066	6647.524200	44.925137
7	Linear Regression (using SGD)	-	-	0.912491	0.912473	6707.898900	43.109089
8	Elastic Net Regression	0.1	0.01	0.821958	0.821922	9562.733100	73.375146
9	Elastic Net Regression	0.1	0.01	0.821958	0.821922	9562.733100	73.375146

11.12 Decision Tree:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

Decision Tree

```
1 from sklearn.tree import DecisionTreeRegressor
```

```
1 from sklearn.metrics import r2_score
2 DT_Model=DecisionTreeRegressor()
3 DT_Model.fit(X_train,y_train)
4 y_predict=DT_Model.predict(X_test)
5 print('R2 score:',r2_score(y_test,y_predict))
6 print('Test RMSE:',mean_squared_error(y_test,y_predict,squared=False))
```

```
R2 score: 0.9696198784005559
Test RMSE: 3952.0590049350017
```

Performing Grid search CV for Decision Tree:

```
depth = list(range(3,30))
param_grid = dict(max_depth = depth)
tree = GridSearchCV(DecisionTreeRegressor(), param_grid, cv = 10)
tree.fit(X_train,y_train)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
                                       15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
                                       25, 26, 27, 28, 29]})
```

```
 # Predicting train and test results
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)
print("Train Results for Decision Tree Regressor Model:")
print("Root Mean squared Error: ",np.sqrt(mse(y_train.values, y_train_pred)))
print("R-Squared: ", r2_score(y_train.values, y_train_pred))
print("Test Results for Decision Tree Regressor Model:")
print("Root Mean Squared Error: ",np.sqrt(mse(y_test, y_test_pred)))
print("R-Squared: ", r2_score(y_test, y_test_pred))
```

```
Train Results for Decision Tree Regressor Model:
Root Mean squared Error:  2490.038299549153
R-Squared:  0.9879729053012675
Test Results for Decision Tree Regressor Model:
Root Mean Squared Error:  3582.4218092773713
R-Squared:  0.9750370384349579
```

We can see that Decision Tree model after Hyperparameter Tuning is giving lower Test RMSE. Thus, is better comparatively.

11.13 Random forest:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
RF_Model=RandomForestRegressor(n_estimators=650,random_state=102,min_samples_leaf=.0001)
RF_Model.fit(X_train,y_train)
y_predict=RF_Model.predict(X_test)
print('R2 score:',r2_score(y_test,y_predict))
print('Test RMSE:',mean_squared_error(y_test,y_predict,squared=False))
```

```
R2 score: 0.9775744978465906
Test RMSE: 3395.4688798634065
```

We have done the following above model building before feature selection intentionally in order to find out which model is performing the best by including all the features. We found that Random_Forest is giving the best R2 score with lowest RMSE. Thus is the best model.

Performing Random search CV for Random Forest:

```
tuned_params = {'n_estimators': [100, 200], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]}
random_regressor = RandomizedSearchCV(RandomForestRegressor(), tuned_params, n_iter = 3, scoring = 'neg_mean_absolute_error'
random_regressor.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=3, n_jobs=-1,
                   param_distributions={'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [100, 200]},
                   scoring='neg_mean_absolute_error')
```

```
# Predicting train and test results
y_train_pred = random_regressor.predict(X_train)
y_test_pred = random_regressor.predict(X_test)
print("Train Results for Random Forest Regressor Model:")
print("Root Mean Squared Error: ", np.sqrt(mse(y_train.values, y_train_pred)))
print("R-Squared: ",r2_score(y_train.values, y_train_pred))
print("Test Results for Random Forest Regressor Model:")
print("Root Mean Squared Error: ", np.sqrt(mse(y_test, y_test_pred)))
print("R-Squared: ",r2_score(y_test, y_test_pred))
```

```
Train Results for Random Forest Regressor Model:
Root Mean Squared Error:  1915.3354290875689
R-Squared:  0.992883960327441
Test Results for Random Forest Regressor Model:
Root Mean Squared Error:  2884.9076963564257
R-Squared:  0.9838115036253952
```

We can see that Random Forest model after Hyperparameter Tuning is giving lower Test RMSE. Thus, is better comparatively.

11.14 Feature selection:

Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data. It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve.

Recursive feature elimination (RFE):

Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached.

RFE is basically a backward selection of the predictors. This technique begins by building a model on the entire set of predictors and computing an importance score for each predictor.

Columns before doing feature selection:

No of columns: 43

```
1 X_train.columns
```

```
Index(['const', 'time_taken', 'airline_AirAsia', 'airline_GO FIRST',
       'airline_Indigo', 'airline_SpiceJet', 'airline_StarAir',
       'airline_Trujet', 'airline_Vistara', 'from_Chennai', 'from_Delhi',
       'from_Hyderabad', 'from_Kolkata', 'from_Mumbai', 'Class_Economy',
       'stop_1', 'stop_2', 'to_Chennai', 'to_Delhi', 'to_Hyderabad',
       'to_Kolkata', 'to_Mumbai', 'Week_7', 'Week_8', 'Week_9', 'Week_10',
       'Week_11', 'Week_12', 'Week_13', 'departure_time_Early_morning',
       'departure_time_Evening', 'departure_time_Midnight',
       'departure_time_Morning', 'arrival_time_Early_morning',
       'arrival_time_Evening', 'arrival_time_Midnight', 'arrival_time_Morning',
       'day_of_week_Monday', 'day_of_week_Saturday', 'day_of_week_Sunday',
       'day_of_week_Thursday', 'day_of_week_Tuesday', 'day_of_week_Wednesday'],
      dtype='object')
```

Columns After doing feature selection:

No of columns:21

```
Index(['airline_GO FIRST', 'airline_Indigo', 'airline_SpiceJet',
       'airline_StarAir', 'airline_Trujet', 'airline_Vistara',
       'from_Hyderabad', 'from_Kolkata', 'Class_Economy', 'stop_1', 'stop_2',
       'to_Delhi', 'to_Hyderabad', 'to_Kolkata', 'Week_7', 'Week_8', 'Week_9',
       'Week_10', 'Week_11', 'Week_12', 'Week_13'],
      dtype='object')
```

11.15 Score card (after feature selection):

	Model_Name	Alpha (Wherever Required)	l1-ratio	R-Squared	Adj. R-Squared	Test_RMSE	Test_MAPE
0	Ridge Regression (with alpha = 2)	2	-	0.913197	0.913179	6679.510800	45.397384
1	Ridge Regression (with alpha = 1)	1	-	0.913197	0.913180	6679.515900	45.400081
2	Lasso Regression	0.01	-	0.913197	0.913180	6679.519400	45.401515
3	Ridge Regression (using GridSearchCV)	0.100000	-	0.913197	0.913180	6679.521400	45.402542
4	Linear Regression	-	-	0.913197	0.913180	6679.522000	45.402819
5	Lasso Regression (using GridSearchCV)	0.000000	-	0.913197	0.913180	6679.522000	45.402819
6	Elastic Net Regression (using GridSearchCV)	0.000100	0.200000	0.913195	0.913177	6679.522500	45.360994
7	Linear Regression (using SGD)	-	-	0.913172	0.913154	6679.919900	45.675011
8	Elastic Net Regression	0.1	0.01	0.820598	0.820561	9594.301600	77.000525

Performing Grid search CV for Decision Tree with Feature selection:

```
depth = list(range(3,30))
param_grid = dict(max_depth = depth)
tree = GridSearchCV(DecisionTreeRegressor(), param_grid, cv = 10)
tree.fit(New_X_train,y_train)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
                                       15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
                                       25, 26, 27, 28, 29]})
```

```
 # Predicting train and test results
y_train_pred = tree.predict(New_X_train)
y_test_pred = tree.predict(New_X_test)
print("Train Results for Decision Tree Regressor Model:")
print("Root Mean squared Error: ",np.sqrt(mse(y_train.values, y_train_pred)))
print("R-Squared: ", r2_score(y_train.values, y_train_pred))
print("Test Results for Decision Tree Regressor Model:")
print("Root Mean Squared Error: ",np.sqrt(mse(y_test, y_test_pred)))
print("R-Squared: ", r2_score(y_test, y_test_pred))
```

```
Train Results for Decision Tree Regressor Model:
Root Mean squared Error:  5207.109531931735
R-Squared:  0.9474052854062345
Test Results for Decision Tree Regressor Model:
Root Mean Squared Error:  5248.12091783339
R-Squared:  0.9464264613821125
```

For Decision tree the R2 score and Test RMSE are 0.9464 and 5248.140 respectively. (WITH OUT GRID SEARCH CV)

We can see that Decision model after Hyperparameter Tuning is giving similar Test RMSE results. (WITH GRID SEARCH CV)

Performing Random search CV for Random Forest with Feature selection:

```
tuned_params = {'n_estimators': [100, 200], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]}
random_regressor = RandomizedSearchCV(RandomForestRegressor(), tuned_params, n_iter = 3, scoring = 'neg_mean_absolute_error'
random_regressor.fit(New_X_train, y_train)
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=3, n_jobs=-1,
                   param_distributions={'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [100, 200]},
                   scoring='neg_mean_absolute_error')
```

```
#Predicting train and test results
y_train_pred = random_regressor.predict(New_X_train)
y_test_pred = random_regressor.predict(New_X_test)
print("Train Results for Random Forest Regressor Model:")
print("Root Mean Squared Error: ",np.sqrt(mse(y_train.values, y_train_pred)))
print("R-Squared: ",r2_score(y_train.values, y_train_pred))
print("Test Results for Random Forest Regressor Model:")
print("Root Mean Squared Error: ",np.sqrt(mse(y_test, y_test_pred)))
print("R-Squared: ",r2_score(y_test, y_test_pred))
```

```
Train Results for Random Forest Regressor Model:
Root Mean Squared Error:  5207.973759479509
R-Squared:  0.9473878255955891
Test Results for Random Forest Regressor Model:
Root Mean Squared Error:  5246.113115883599
R-Squared:  0.946467445376615
```

After doing feature selection the R2 score and Test RMSE for Random Forest model are 0.9460 and 5267.335 (WITH OUT RANDOM SEARCH CV)

We can see that Random Forest model after Hyperparameter Tuning is giving lower Test RMSE. Thus, is better comparatively. (WITH RANDOM SEARCH CV)

12.Conclusion:

Sl.No	Before Feature Selection			After Feature Selection		
0	Model Name	R2	Test RMSE	Model Name	R2	Test RMSE
1	Linear Regression	0.914087	6647.4941	Linear Regression	0.913197	6679.522
2	Linear Regression (using SGD)	0.912491	6707.8989	Linear Regression (using SGD)	0.913172	6679.9199
3	Lasso Regression	0.914087	6647.491	Lasso Regression	0.913197	6679.5194
4	Ridge Regression (with alpha = 1)	0.914087	6647.4899	Ridge Regression (with alpha = 1)	0.9131	6679.5159
5	Elastic Net Regression	0.821958	9562.733	Elastic Net Regression	0.820598	9594.3016
6	Lasso Regression(GridSearchCV)	0.914087	6647.491	Lasso Regression(GridSearchCV)	0.913197	6679.522
7	Ridge Regression(GridSearchCV)	0.914087	6647.4937	Ridge Regression(GridSearchCV)	0.913197	6679.5214
8	Elastic Net Regression(GridSearchCV)	0.914084	6647.5242	Elastic Net Regression(GridSearchCV)	0.913195	6679.5225
9	Random Forest	0.97757	3395.469	Random Forest	0.94603	5267.3356
10	Random Forest (Tuned)	0.9838	2884.9077	Random Forest (Tuned)	0.94646	5246.1131
11	Decision Tree	0.9698	3939.488	Decision Tree	0.94642	5248.1405
12	Decision Tree (Tuned)	0.975	3582.421	Decision Tree (Tuned)	0.9464	5248.1209

Random Forest Model is performing the **best** while including all the features as well as post feature selection also. But still, we will be considering the Random Forest model after hyperparameter tuning to be the best one as Normal RF model seems to be overfitted.

13. Project Summary:

- The project, ‘Flight price prediction’ aims to find out the best prices of the airplane tickets for the provided itenaries ,in order to make the air journey convenient for passengers so that more tickets can be booked which directly benefit the airlines such that flights may run on their full capacities.
- This aim is being planned to fulfill by making a machine learning based regression model which when deployed can be able to find out the best prices for the future travels of any passenger.
- This machine learning model is being planned to be worked on a dataset that is being provided by EaseMyTrip for a span of 2 months, 2022.The dataset was divided into Economy and Business classes initially which was later merged into a single dataset and the further processes are being carried over it.
- We have gone through various challenges while performing Data Cleaning from cleaning various columns contains unknown characters to changing their datatypes to the nature they need to be dealt with. While performing the same we have observed the stops column comprising of 0,1,2 and 2+ stops but when the column was being further analyzed, it was found that the stops are missing for more than 1 stops, making that column not relevant.
- We have tried to do Data manipulation by adding a distance column from source to destination. But since as the stops were not giving the sufficient information, simply adding the distance between source and destination airports may be of no use. As a result of which we have decided to not consider it for our future analysis.
- Then we headed with the Exploratory Data Analysis where we did univariate, bivariate as well as multivariate analysis out of which we concluded many things about various airlines, ticket prices corresponding to both classes, no. of stops, source airport where we found that Vistara airways, Business classes, flights from Kolkata, single stop flights, are having the most expensive flight tickets.
- We then headed with the Feature Engineering part where we made a new column called departure time and arrival time being extracted from flight arrival and departure times making our analysis even easier to move with.
- We then performed dummy encoding over our categorical features. No scaling was performed as we are left with only one numerical feature apart of target variable,sollowed by train test split.
- Checking for assumptions being our next step where we found that there is high multicollinearity present as well as the residuals are not normal in nature. We further tried removing multicollinearity with the help of VIF but we found that there is moderate multicollinearity present.
- Also, the residuals for time_taken column was seen to follow the linearity and no other residual follow.
- We then headed with base model model building followed with 7 other models from where we found that the Hyper-tuned Random Forest model to be performing the best but because of

significant difference between train-test RMSE, it got overfitted. Also, the algorithms were taking so much time to run. As a result of which we headed with finding out the most significant features use RFE technique.

- Again, with the new selected features, we again headed with the building of all the 8 models. Now the algorithms were taking lesser time to run with significantly improved accuracy.
- We found that with the best features selected, Hyper-tuned Random Forest model to be performing as the best one with the overfitting being reduced.

14.Limitations:

- One of the Limitation is that the data was being available for the span of only two months.
- The time taken column shows that there are many flights that are taking 35 hours which seems quite unrealistic when dealing with real life scenario. They could be treated as outliers.
- The dataset didn't provide 'total distance covered' by flight, which if provided can be very much helpful in future analysis.
- The stops column was full of ambiguities where majority of the stop details was not present.

15.Closing Reflections:

- We have learned many things from Data Extraction, Data Cleaning, Data manipulation, EDA, Feature Engineering, Statistical aspects to model building, Model building including error functions and further Hyperparameter tuning in order to boost the model performances.
- For the next time first, we will opt to choose a dataset of a larger span having proper mentioning of Routes especially. In addition to this, we may also introduce the algorithms associated with Time Series.
- As a result of this model may be also able to predict the uneven price surges on the basis of various festivals and holidays, i.e., Dynamic price surging prediction.