WHAT INFLUENCES THE PRICE OF FLIGHTS

Nishant Redu

We will be solving the problem of what affects the prices of flights with the help of Machine Learning. The problem at hand is that we are having a dataset that contains the following attributes-:

Airline – The Airline Brand

Date\_of\_Joruney – Date of Journey

Source – City From where the flight takes off

Destination – City where the flight lands after covering all the stops

Route – The Route of the city

Dep\_Time – Departure Time of Flight

Arrival\_Time – Arrival Time in the Destination City

Duration – Total Duration From Source to Destination

Additional\_info – Additional Information about flight

Price – The Price from Source to Destination

Using this dataset, we have to build a machine learning model that predicts the prices of flights. We are dealing with a regression problem here as we need to predict a continuos variable.

STEP 1 - Loading the dataset and Doing initial Work

We will first load the dataset using the reads\_csv() function of pandas library, the file we are loading Is ‘Data\_Train.csv’, and it is present in the same direcotry as our jupyter notebook.

*data = pd.read\_csv('Data\_Train.csv')*

Lets Look at the shape of our dataset.

*print(data.shape)*

Our dataset has 10683 rows and 11 columns.

To check if there are any null values in our dataset, we will isnull().sum() at our dataFrame object.

*print(data.isnull().sum())*

Airline 0

Date\_of\_Journey 0

Source 0

Destination 0

Route 1

Dep\_Time 0

Arrival\_Time 0

Duration 0

Total\_Stops 1

Additional\_Info 0

Price 0

As we can see there are only two null values in the dataset, one in Route and one in Total\_Stops, we can go ahead and drop these values as don’t want to make our dataset bad because of only two values.

*data = data.dropna()*

After dropping the null values, we need to look at the information of our dataset

*data.info()*

Int64Index: 10682 entries, 0 to 10682

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Airline 10682 non-null object

1 Date\_of\_Journey 10682 non-null object

2 Source 10682 non-null object

3 Destination 10682 non-null object

4 Route 10682 non-null object

5 Dep\_Time 10682 non-null object

6 Arrival\_Time 10682 non-null object

7 Duration 10682 non-null object

8 Total\_Stops 10682 non-null object

9 Additional\_Info 10682 non-null object

10 Price 10682 non-null int64

dtypes: int64(1), object(10)

We can see here that weneed to trasnform a few variables like Date\_of\_Journey, Route, Dep\_Time, Arrivall\_Time, Duration as there variable are of type object but they represent either Time of Date. So we need to transform there variables.

STEP 2 – FEATRUE ENGINEERING

FEATURE TRANSFROMATION OF DATE\_OF\_JOURNEY COLUMN

Lets look at how the entries in the Date\_Of\_Journey column look like

*Data.Date\_of\_Journey*

0 24/03/2019

1 1/05/2019

2 9/06/2019

3 12/05/2019

4 01/03/2019

...

10678 9/04/2019

10679 27/04/2019

10680 27/04/2019

10681 01/03/2019

10682 9/05/2019

We can see that it has dd/mm/yyyy format, we can either use datetime function of pandas, or we can use the string split functions for each entry. We will follow the later procedure here.

If we split each entry by ‘/’, and then take the 0th index of the spliiter entry, we will get the day of the month as a string, we havce to convert it to int before adding inot our dataFrame. Similarlry if we take 1st index of the splitted entry , we will get the month as a string, and we if take the 2nd index, we will get the year of the entry.

*data.Date\_of\_Journey[0].split('/')[0]*

'24'

We will create three empty lists, one for each of the day, month and year.

*journey\_day = []*

*journey\_month = []*

*journey\_year = []*

We will loop through the Date\_of\_Joruney column, and then split each entry using the procecure mentioned above and add them to each of their respective lists.

*for date in data.Date\_of\_Journey:*

*day = int(date.split('/')[0])*

*month = int(date.split('/')[1])*

*year = int(date.split('/')[2])*

*journey\_day.append(day)*

*journey\_month.append(month)*

*journey\_year.append(year)*

Then we will create a new column for each of the list in the dataframe and assign them the values in this list.

*data['Journey\_day'] = journey\_day*

*data['Journey\_month'] = journey\_month*

*data['Journey\_year'] = journey\_year*

We can now drop Date\_Of\_Journey columns from our dataset.

*data = data.drop(columns=['Date\_of\_Journey'])*

FEATURE TRANSFORMATION OF ROUTE COLUMN

Lets have a look at how the route column looks like.

*data.Route*

0 BLR → DEL

1 CCU → IXR → BBI → BLR

2 DEL → LKO → BOM → COK

3 CCU → NAG → BLR

4 BLR → NAG → DEL

...

10678 CCU → BLR

10679 CCU → BLR

10680 BLR → DEL

10681 BLR → DEL

10682 DEL → GOI → BOM → COK

We can see here that there is a stop and then the next stop is mentioned after -> marker. **We are gonna split each entry in route column by '→', and then if the length of splitted list is 3, then there is one stop and we add that to the first stop column, if the length is 4 , then we add that to second stop column and so on.**

**There are only 4 maximum stops in all the entries in our data.**

**We wil creata a list for each of the stops.**

*stop\_1 = []*

*stop\_2 = []*

*stop\_3 = []*

*stop\_4 = []*

We will loop through each entry in Route column and then split and append in our lists.

*for route in data.Route:*

*rl = route.split('→')*

*if len(rl)==2:*

*stop\_1.append(np.nan)*

*stop\_2.append(np.nan)*

*stop\_3.append(np.nan)*

*stop\_4.append(np.nan)*

*if len(rl)==3:*

*stop\_1.append(rl[1])*

*stop\_2.append(np.nan)*

*stop\_3.append(np.nan)*

*stop\_4.append(np.nan)*

*if len(rl)==4:*

*stop\_1.append(rl[1])*

*stop\_2.append(rl[2])*

*stop\_3.append(np.nan)*

*stop\_4.append(np.nan)*

*if len(rl)==5:*

*stop\_1.append(rl[1])*

*stop\_2.append(rl[2])*

*stop\_3.append(rl[3])*

*stop\_4.append(np.nan)*

*if len(rl)==6:*

*stop\_1.append(rl[1])*

*stop\_2.append(rl[2])*

*stop\_3.append(rl[3])*

*stop\_4.append(rl[4])*

We will than create new columns for each stop in our dataset, and add these lists to their respecitve columns.

*data['stop\_1'] = stop\_1*

*data['stop\_2'] = stop\_2*

*data['stop\_3'] = stop\_3*

*data['stop\_4'] = stop\_4*

We can now drop the Route Column.

*data = data.drop(columns=['Route'])*

FEATURE TRANSFORMATION OF DEP\_TIME COLUMN

We split the entries in the Dep\_Time column by ‘:’, the 0th index will be added in hour and the first index will be added in minute.

*hours = []*

*mins = []*

*for time in data.Dep\_Time:*

*time = time.split(':')*

*hours.append(int(time[0]))*

*mins.append(int(time[1]))*

*data['Dep\_hour'] = hours*

*data['Dep\_min'] = mins*

FEATURE TRANSFORMATION OF ARRIVAL\_TIME COLUMN

We will split each entry by ‘ ‘ and if the length of the splitted variable is more than 1 , then the flight will reach the next day, and we add an extra column for that next day. Then we do the same as we did for Dep\_Time Column.

*Same\_day = []*

*Arrival\_Hour = []*

*Arrival\_Minute = []*

*for arr in data.Arrival\_Time:*

*arr = arr.split(' ')*

*if len(arr)>1:*

*Same\_day.append(0)*

*else:*

*Same\_day.append(1)*

*Arrival\_Hour.append(int(arr[0].split(':')[0]))*

*Arrival\_Minute.append(int(arr[0].split(':')[1]))*

*data['Same\_day'] = Same\_day*

*data['Arrival\_Hour'] = Arrival\_Hour*

*data['Arrival\_Minute'] = Arrival\_Minute*

FEATURE TRANSFORMATION OF DURATION COLUMN

The duration column is in the following format -: 12h 30m, and if the duration is less than an hour, then it is of the format-: 30m, we will first split each entry by ‘ ‘, and then if the length is 0, then we either have the flight of exactly duration in hours , or only in mins. In that case we will try to add up the hours, and if there is an exception we will add only for minutes. If the length is not zero, then we follow our normal procedure of converting duration into mins.

*Duration = []*

*for dur in data.Duration:*

*dur = dur.split(' ')*

*if len(dur)==1:*

*try:*

*hour = int(dur[0].split('h')[0])*

*dur = (hour\*60)*

*except:*

*mins = int(dur[0].split('m')[0])*

*dur = mins*

*else:*

*hour = int(dur[0].split('h')[0])*

*mins = int(dur[1].split('m')[0])*

*dur = (hour\*60) +mins*

*Duration.append(dur)*

*data['Duration'] = Duration*

FEATURE TRANSFORMATION OF TOTAL STOPS COLUMN

The total stops column is currently of type obejct and we need to convert that to int, we will simply use replace function of pandas to do this. We will pass the a dictionary as an argument to the replace funtion. The keys in dictionary will be the old values, and the value will be the new values which we want to add in place of the old values.

*data['Total\_Stops'] = data['Total\_Stops'].replace({*

*'non-stop':0,*

*'2 stops':2,*

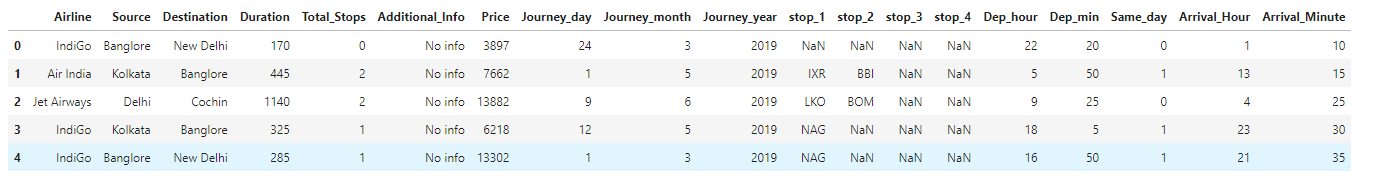
*'1 stop':1,*

*'3 stops':3,*

*'4 stops':4*

*})*

After all the feature Engineering steps here is how our data looks like-:



STEP 3 – EXPLORATORY DATA ANALYSIS

For exploratory data analysis, first we need to decide which columns are catefogircal and which are numerical. For this the columns whose datatype is object are categorical and those whose data type is not object are cateogrical.

*cat\_cols = [col for col in data.columns if data[col].dtype=='object']*

*num\_cols = [col for col in data.columns if col not in cat\_cols]*

And if have a look at our data, then we find that the Same\_day Column is categorical, as it has only two values, but as it is not an object, we have to add it manually to our lists.

*cat\_cols.append('Same\_day')*

*num\_cols.remove('Same\_day')*

Lets have a look at our numerical columns and categorical columns

*num\_cols*

['Duration',

'Total\_Stops',

'Price',

'Journey\_day',

'Journey\_month',

'Journey\_year',

'Dep\_hour',

'Dep\_min',

'Arrival\_Hour',

'Arrival\_Minute']

*cat\_cols*

['Airline',

'Source',

'Destination',

'Additional\_Info',

'stop\_1',

'stop\_2',

'stop\_3',

'stop\_4',

'Same\_day']

SCATTERPLOTS

These are the plots we build to find the relation between two numerical variables, and we will use a regplot to make sure we are able to see a clear line that shows which way the trend is going.

*plt.figure(figsize=(24,24))*

*plotnumber=1*

*for col in num\_cols:*

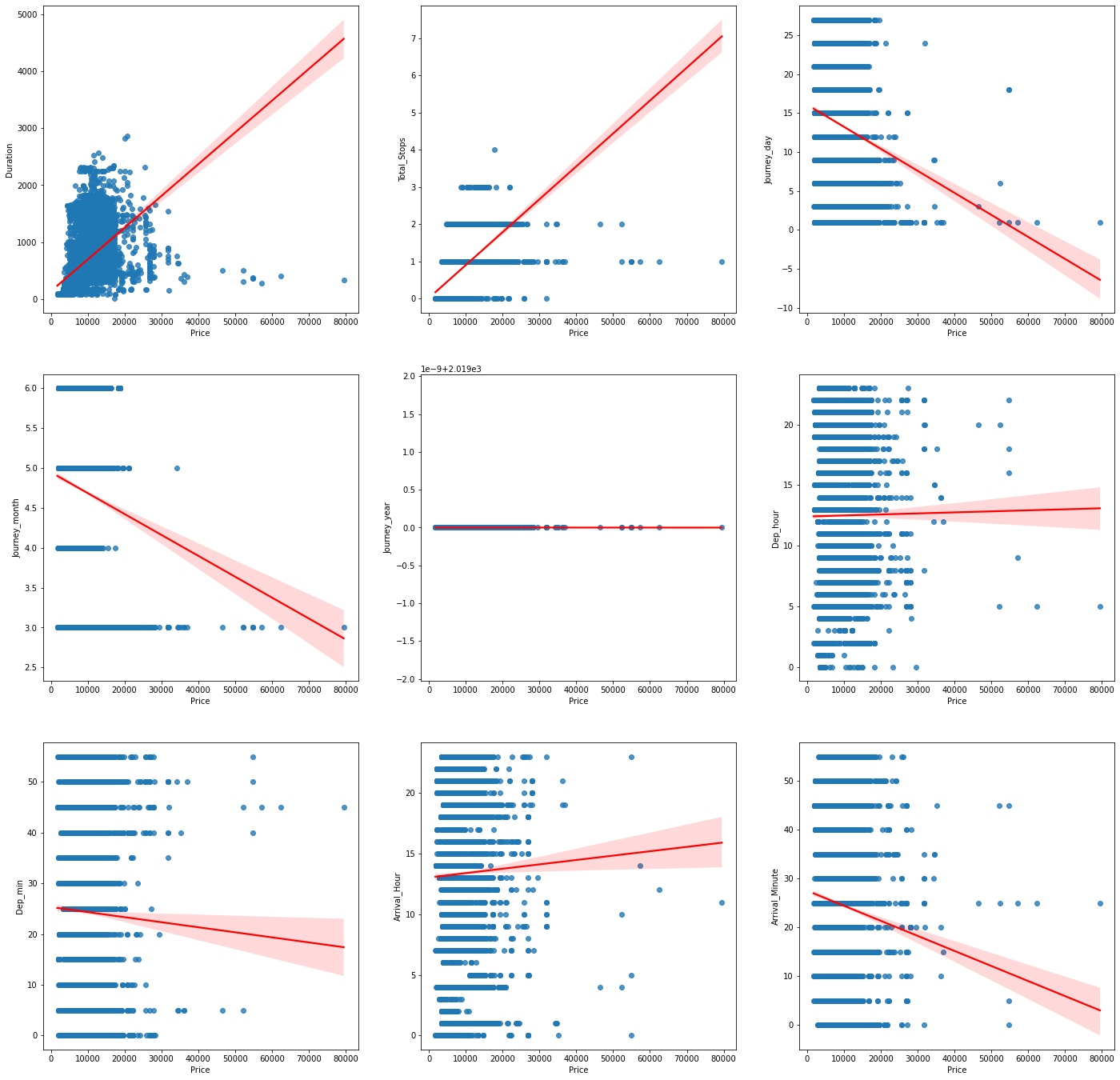
*if col=='Price':*

*continue*

*plt.subplot(3,3, plotnumber)*

*sns.regplot('Price', col, data=data, line\_kws={'color':'red'})*

*plotnumber+=1*



The assumptions we are able to make using these plots are-:

1. Duration follow an upward trend with price, which means that the flight gets longer the price goes up.
2. Total Stops follow an upward trend with price, which means that as the flight stops more, the prices increase
3. Journey\_Day follows a downward trend with price, which means that the price will be high at the start of the month and low towards the end of the month
4. Journey\_Month follows a downward trend with price, which means that towards the end of the year the prices go low
5. Journey\_year has only one value, we can drop this column

*data = data.drop(columns=['Journey\_year'])*

*num\_cols.remove('Journey\_year')*

1. Dep\_Hour has no clear trend with Price
2. Dep\_Min also has no clear trend with price
3. Arrival\_Hour has slight upward trend with price, which means that if the flight arrivew towards the end of the day the prices will be high
4. Arrival\_Minute has downward trend with price, which means that if the flight arrives around the start of the hour, the price will be high

STRIP PLOTS

Note-: StripPlots, KDEplots and Mean Plots all work together, so first all the plots will be shown and then the assumptions will be made.

*plt.figure(figsize=(24,24))*

*plotnumber=1*

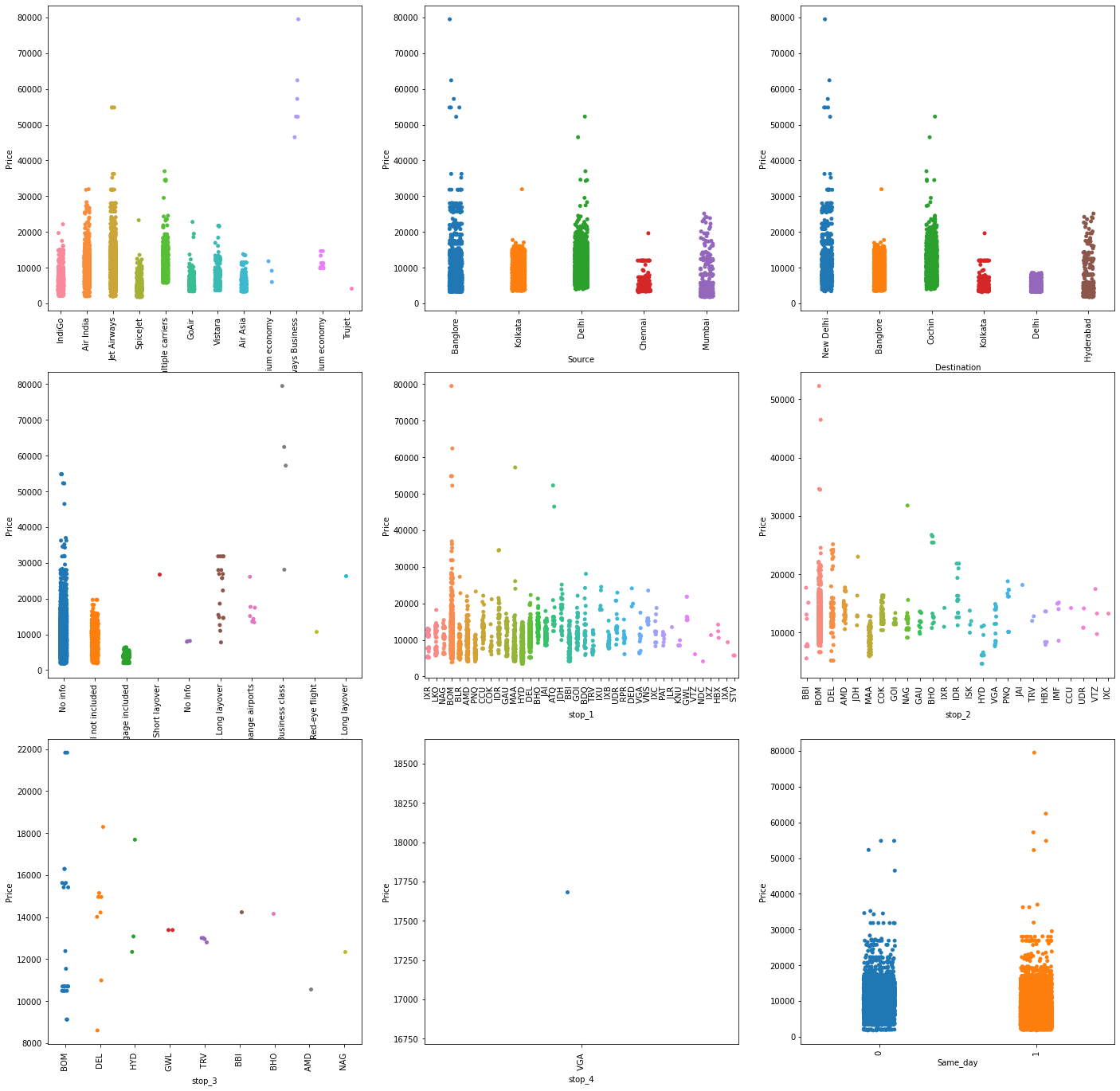
*for col in cat\_cols:*

*plt.subplot(3,3,plotnumber)*

*sns.stripplot(col, 'Price', data=data)*

*plt.xticks(rotation='vertical')*

*plotnumber+=1*



KDE Plots

Kde plots show the density of a numerical variable according to different cateforical variables.

*plt.figure(figsize=(24,24))*

*plotnumber=1*

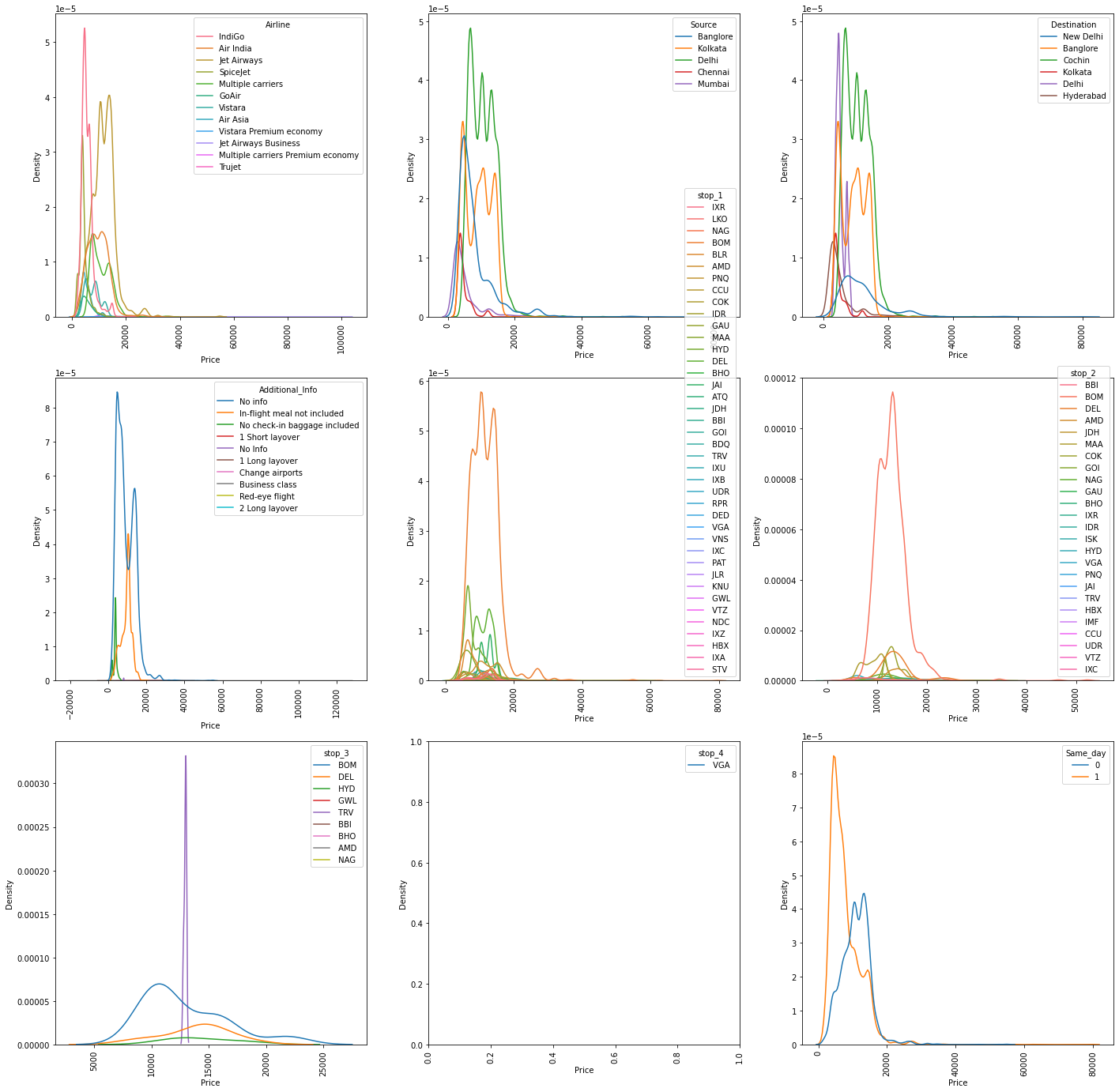
*for col in cat\_cols:*

*plt.subplot(3,3,plotnumber)*

*sns.kdeplot('Price',data=data, hue=col)*

*plt.xticks(rotation='vertical')*

*plotnumber+=1*



MEAN Plots

Mean plots show the mean of the target variable according to different categorical variables.

*plt.figure(figsize=(24,24))*

*plotnumber = 1*

*for col in cat\_cols:*

*mean\_vals = pd.DataFrame()*

*unique\_val = data[col].unique().tolist()*

*means = []*

*for val in unique\_val:*

*mean\_val = data[data[col] == val]*

*mean\_val = mean\_val['Price'].mean()*

*means.append(mean\_val)*

*mean\_vals['Values'] = unique\_val*

*mean\_vals['Mean'] = means*

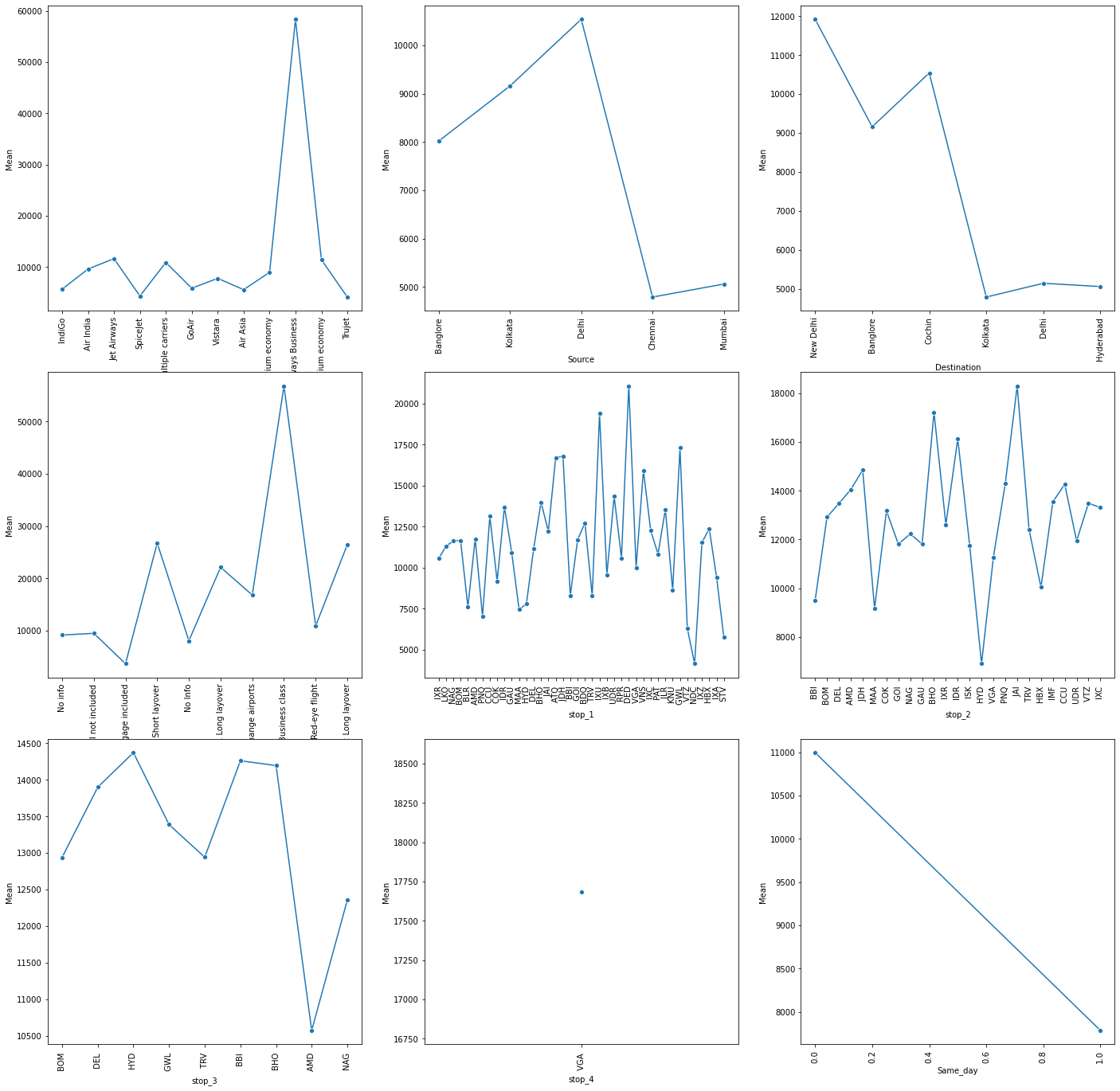
*plt.subplot(3,3, plotnumber)*

*sns.lineplot(x='Values', y='Mean', data=mean\_vals, marker='o')*

*plt.xlabel(col)*

*plt.xticks(rotation='vertical')*

*plotnumber+=1*



Outcome from above three plots

1. If the flight arrives the same day, the prices is low
2. Jet Airways Business has higher prices compared to other
3. Chennai and Mumbai have lowest prices as source
4. Delhi has highest prices source
5. Kolkata, Delhi and Hyderabad have high prices as Destination
6. New Delhi has highest prices as destination

CORRELATION MATRIX

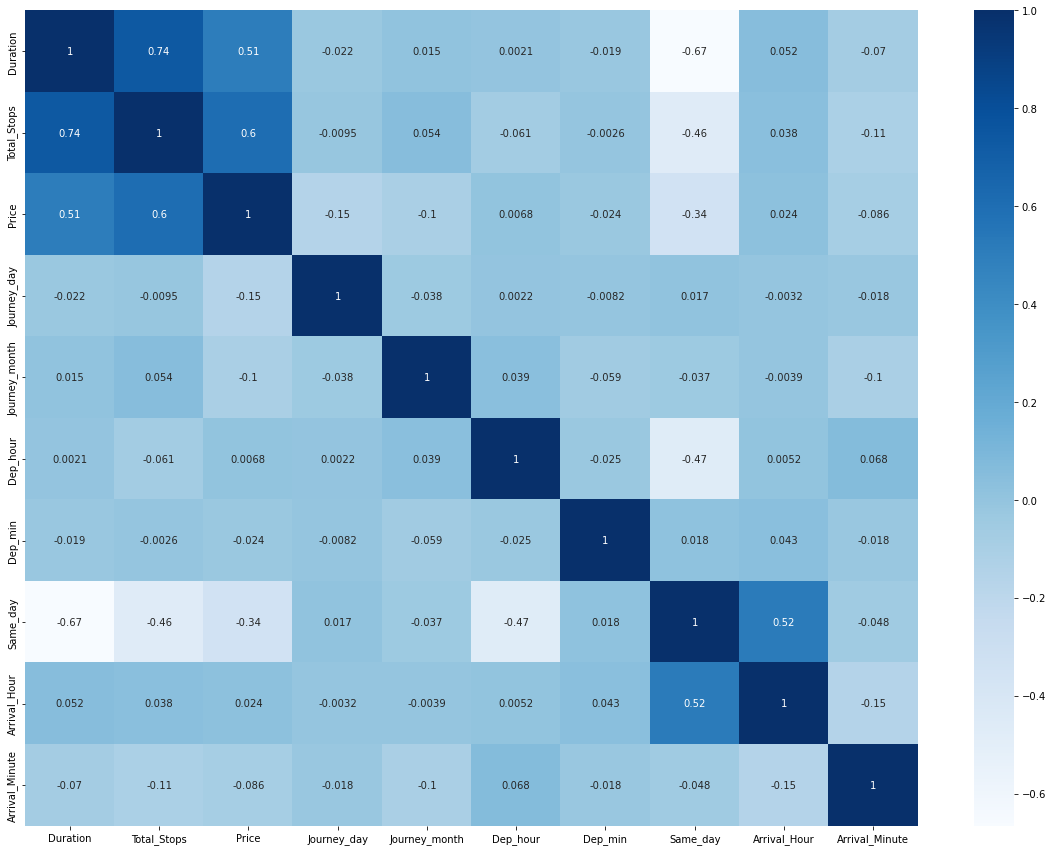
We will now use the correlation matrix. Correlation is the scale at which two values are dependent on each other, it can range from -1 to 1. If corrleation is -1, then both values follow reverse trend to each other i.e. if one increases the other decreases. If correlation is 1, then both values follow a positive trend with each other, if one increase, the other also increases. A correlation values of 0 means, both values are indexpendent of each other.

We can view the correlation matrix as an image using the following code.

*plt.figure(figsize=(20,15))*

*sns.heatmap(data.corr(), cmap='Blues', annot=True)*

The arguments ‘cmap’, sets the color mapping of the image, and argument ‘annot’ tells whether or not to show the correlation value in the cell. Also, correlation is only shown for numerical variables i.e. those whose data type is not object. The output of above code snippnet for this dataset is-:



While reading a correlation matrix, we only look at the values which are either reallly dark or really light, meaning the values whose correlation is either greater than 0.5 or less than -0.5, because these are the only values which may cause a problem in our dataset. Also, the values above and below the diagonal are all same.

Outcomes Of Correlation Matrix-:

1. Total Stops has 74% correlation with duration.
2. Price has 51% correlaion with duration.
3. Price has 60% correlation with Total Stops.
4. Duration has -67% correlation with Same\_Day.
5. Arrival\_Hour has 52% correlation with Same\_Day

DESCRIPTIVE STATISTICS

We will now have a look at the descriptive Statistics of our dataset. These show the count, mean, Standard Deviation, Minimum, 25 Percentile, 50 Percentile, 75 Percentile and maximum of numeircal values in our dataset.

We can view these statistics using below code snippnet-:

*data.describe()*

The output of above code snippet is-:



While reading the above dataFrame we look at if there are any major difference between min or 25%, 25% or 50%, 50% or 75%, 75% or max, If there are that means our dataset is skewed. Looking at the above dataFrame, we can conclude the following-:

1. Duration has a major differnce between 75 percentile and maximum.
2. Price is also having a major difference between 75 Percentile and maximum.
3. All other columns are having fairly equal distribution.

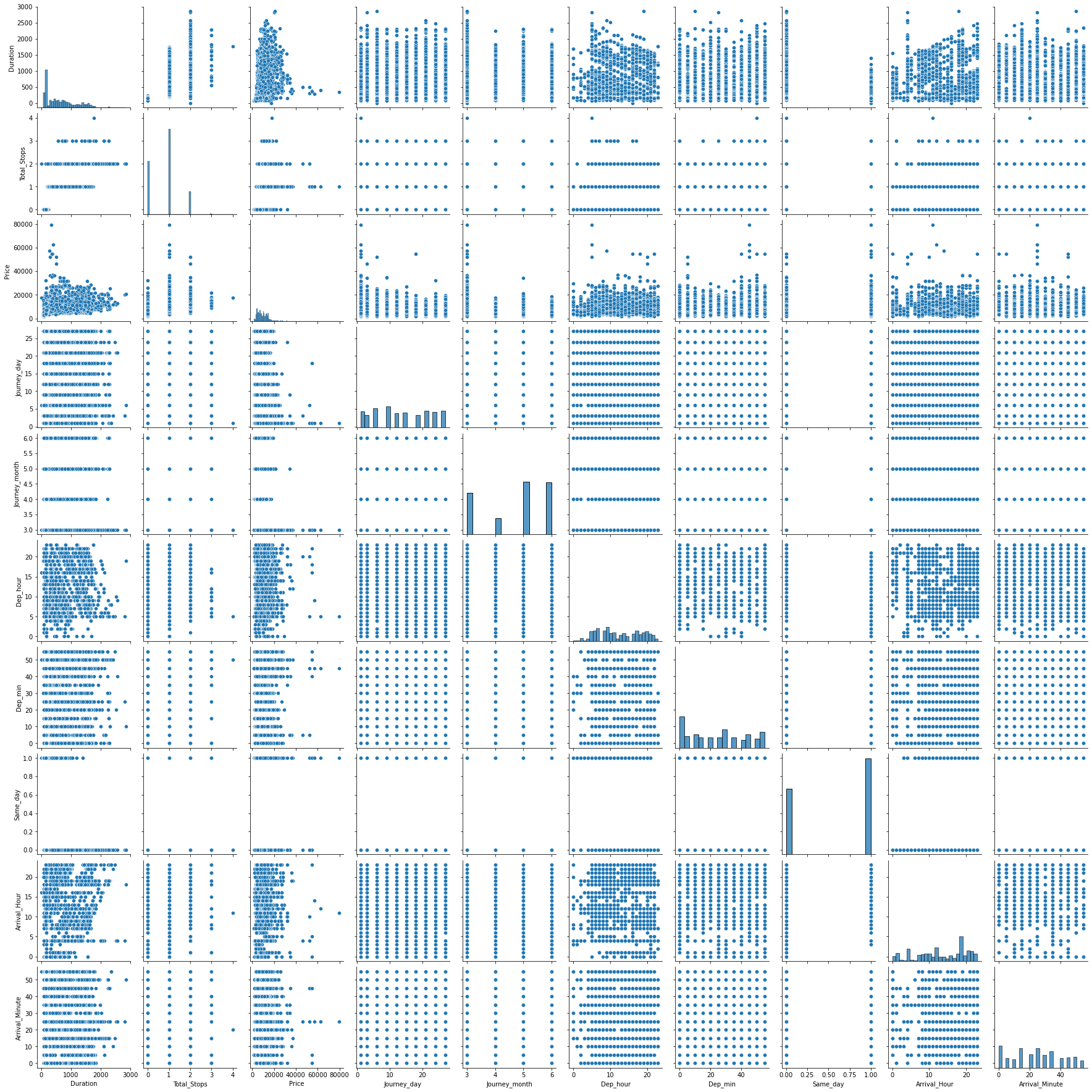
PAIRPLOT

Pairplot shows the scatterplot between all numerical variables and shows the distribuion of all numerical variables.

We can view pairplot using below code snippet.

*sns.pairplot(data)*

The output of above code snippet is-:



As we observed in our correlation matrix and Desriptive Statistics-:

1. Duration and price are skewed.
2. Total Stops has correlation with duration.
3. Price has correlaion with duration.
4. Price has correlation with total Stops.

STEP 3 – PREPROCESSING THE DATASET

This step includes cleanig the dataset, which means there should be no outliers left, and no skew should be present, and the dataset should be scaled. We do not apply any of these steps to the target variable.

HANDLING OUTLIERS

Let’s have a look at the distribution plot of all the numerical variables which we need to preprocess. For that we need to remove the ‘Price’ columns from numerical columns.

*num\_cols.remove('Price')*

Now lets select the numerical columns from our dataset.

*X = data[num\_cols]*

Now we will plot the distribution plot of this dataset.

*plt.figure(figsize=(24,24))*

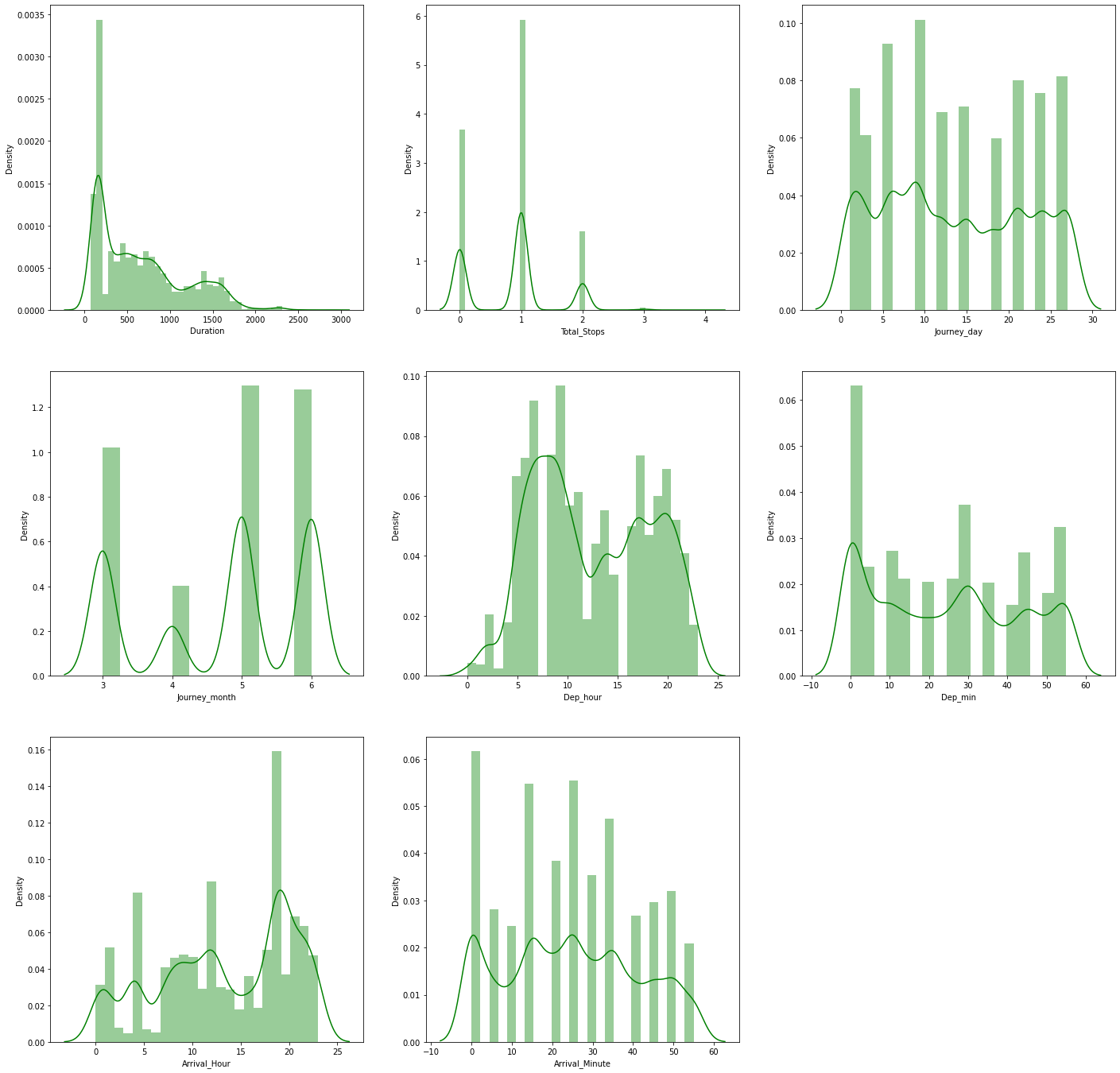
*plotnumber=1*

*for col in X.columns:*

*plt.subplot(3,3,plotnumber)*

*sns.distplot(X[col], color='green')*

*plotnumber+=1*



As we can see from the plot Duraion and Total\_stops are having skew, and rest all are close to normal.

Lets check the skew of all variables now-:

*X.skew()*

Duration 0.861411

Total\_Stops 0.317109

Journey\_day 0.118174

Journey\_month -0.387409

Dep\_hour 0.112924

Dep\_min 0.167234

Arrival\_Hour -0.370146

Arrival\_Minute 0.110945

We can see that duraion and Total\_Stops have the highest skew.

Lets now see the boxplots of these numerical variables.

*plt.figure(figsize=(24,12))*

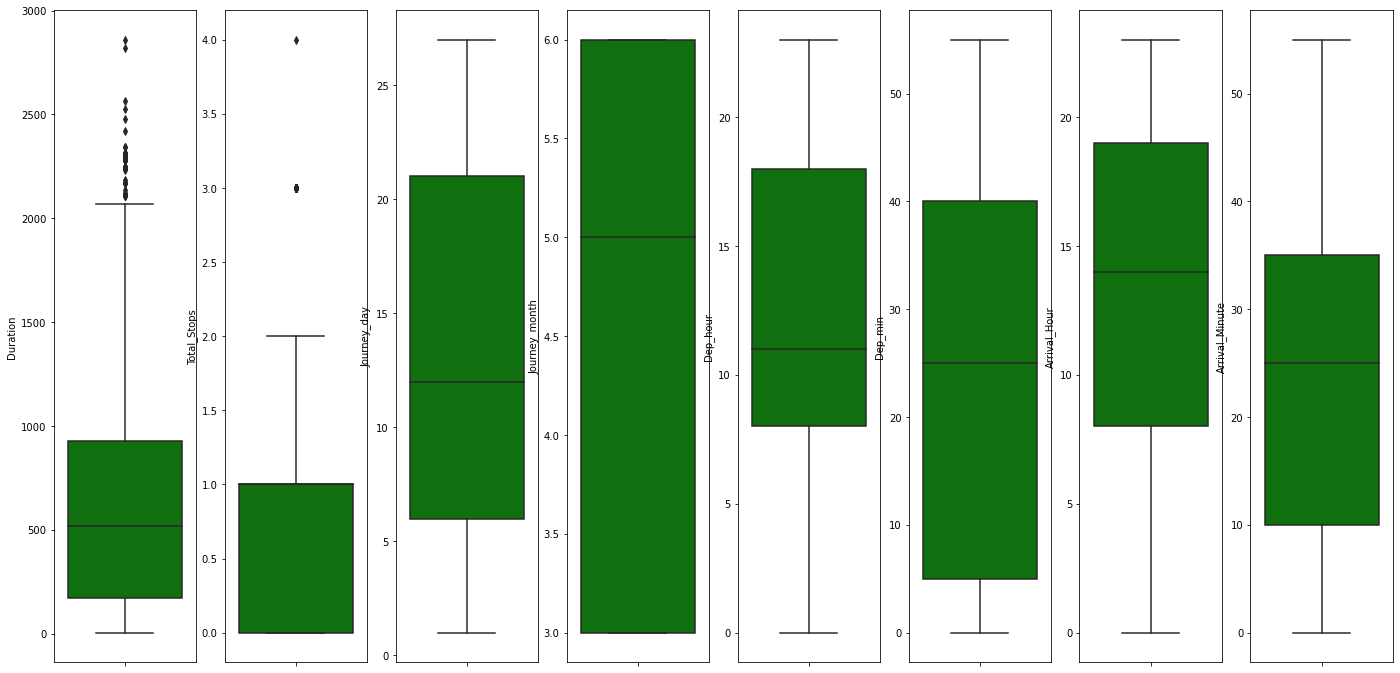
*plotnumber=1*

*for col in X.columns:*

*plt.subplot(1,8,plotnumber)*

*sns.boxplot(y=X[col], color='green')*

*plotnumber+=1*



We can also see in the boxplots here that there are more outliers in the Duration column and a few in the Total\_Stops column. Rest all variable have no outliers present.

Now lets try to remove these outliers using z-score tezhnique.

Z-score technique will retain all the values lying in 3 standard Deviation around the mean.

We will apply z-score on X dataset which we chose earlier and drop all the rows which are out of 3 STDs of the mean.

*from scipy.stats import zscore*

*z = np.abs(zscore(X))*

*data\_new = data[(z<3).all(axis=1)]*

*print("Old Shape", data.shape)*

*print("New Shape", data\_new.shape)*

*print("Total Dropped Rows", data.shape[0] - data\_new.shape[0])*

ld Shape (10682, 18)

New Shape (10583, 18)

Total Dropped Rows 99

Now we have dropped 99 rows in total. Lets have a look at the Distribution Plots now.

*plt.figure(figsize=(24,24))*

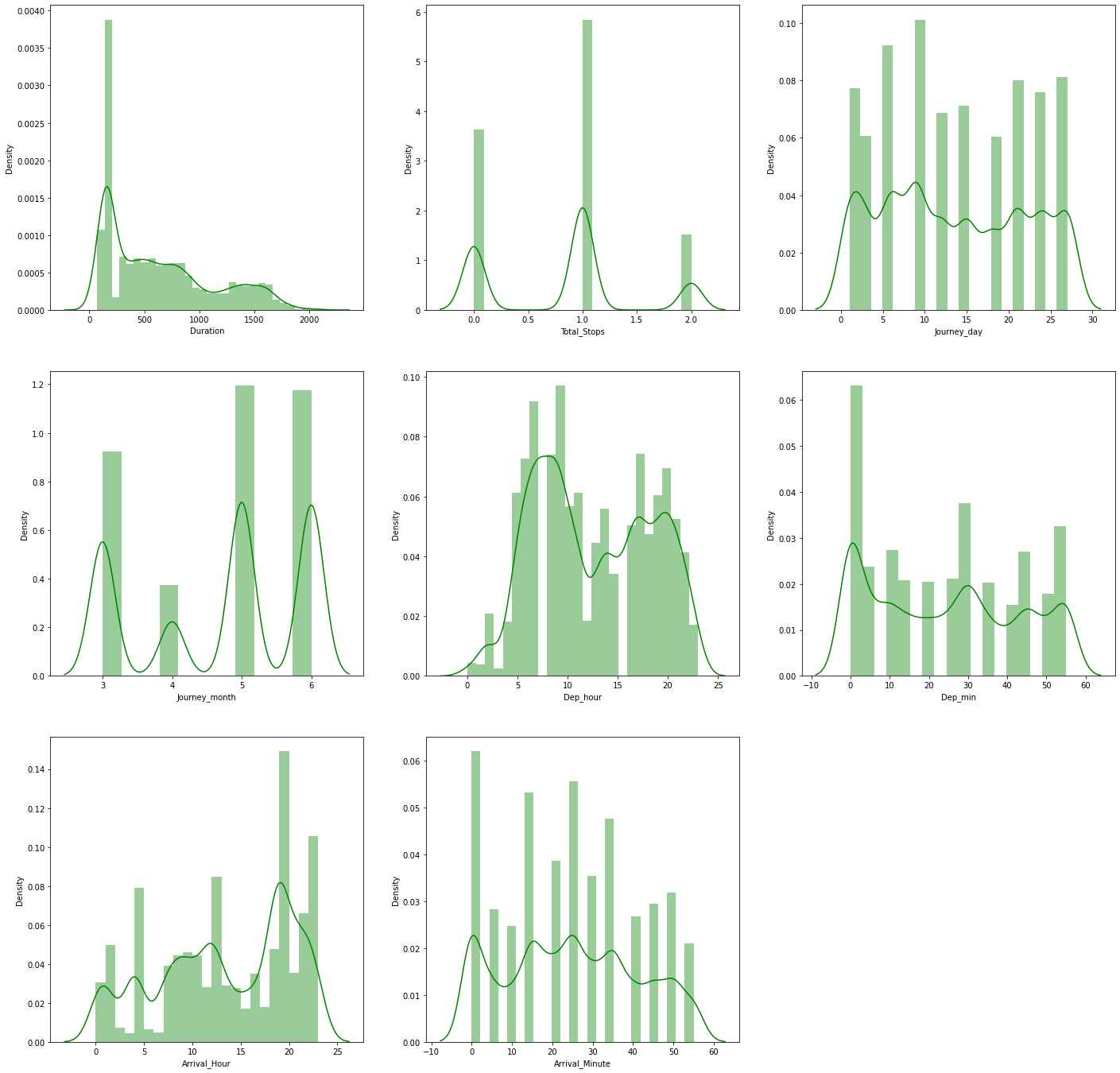
*plotnumber=1*

*for col in X.columns:*

*plt.subplot(3,3,plotnumber)*

*sns.distplot(data\_new[col], color='green')*

*plotnumber+=1*



Now we can see that Duration and Total\_Stops are close to normal than before.

Lets look at the skew of the new variables.

*data\_new[num\_cols].skew()*

Duration 0.775665

Total\_Stops 0.223931

Journey\_day 0.115429

Journey\_month -0.396832

Dep\_hour 0.100933

Dep\_min 0.166199

Arrival\_Hour -0.362019

Arrival\_Minute 0.107319

We can see that the skew is slightly less in these variables now.

ENCODING USING GET\_DUMMIES

We will now use the pd.get\_dummies to encode the categorical values.

Let’s first separate the dependent and independent variable.

*X = data\_new.drop(columns=['Price'])*

*y = data\_new.Price*

Now we will encode the X variable using pd.get\_dummies, it will automatically handle the missing values in the categorical columns.

*X = pd.get\_dummies(X)*

Lets have a look at the shape of X.

*X.shape*

(10583, 101)

Now we have 101 columns in our dataset.

TRANSFORMATION TO REMOVE SKEW

We will use yeo-johnson transformation transformation to remove the remaining skew and bringing all values close to normal.

*from sklearn.preprocessing import power\_transform*

*cols = X.columns*

*X = power\_transform(X, method='yeo-johnson')*

*X = pd.DataFrame(X, columns=cols)*

The power\_trasnform function returns a numpy array, so we need to transform it back to a dataFrame. The last two lines in above code snippet do that.

Lets have a look at the skew now.

*X[num\_cols].skew()*

Duration -0.048477

Total\_Stops -0.071888

Journey\_day -0.204695

Journey\_month -0.218849

Dep\_hour -0.106907

Dep\_min -0.359777

Arrival\_Hour -0.353761

Arrival\_Minute -0.350083

We can see that now there is no heavy skew present in our dataset.

Lets have a look at the distribution plots now.

*plt.figure(figsize=(24,24))*

*plotnumber=1*

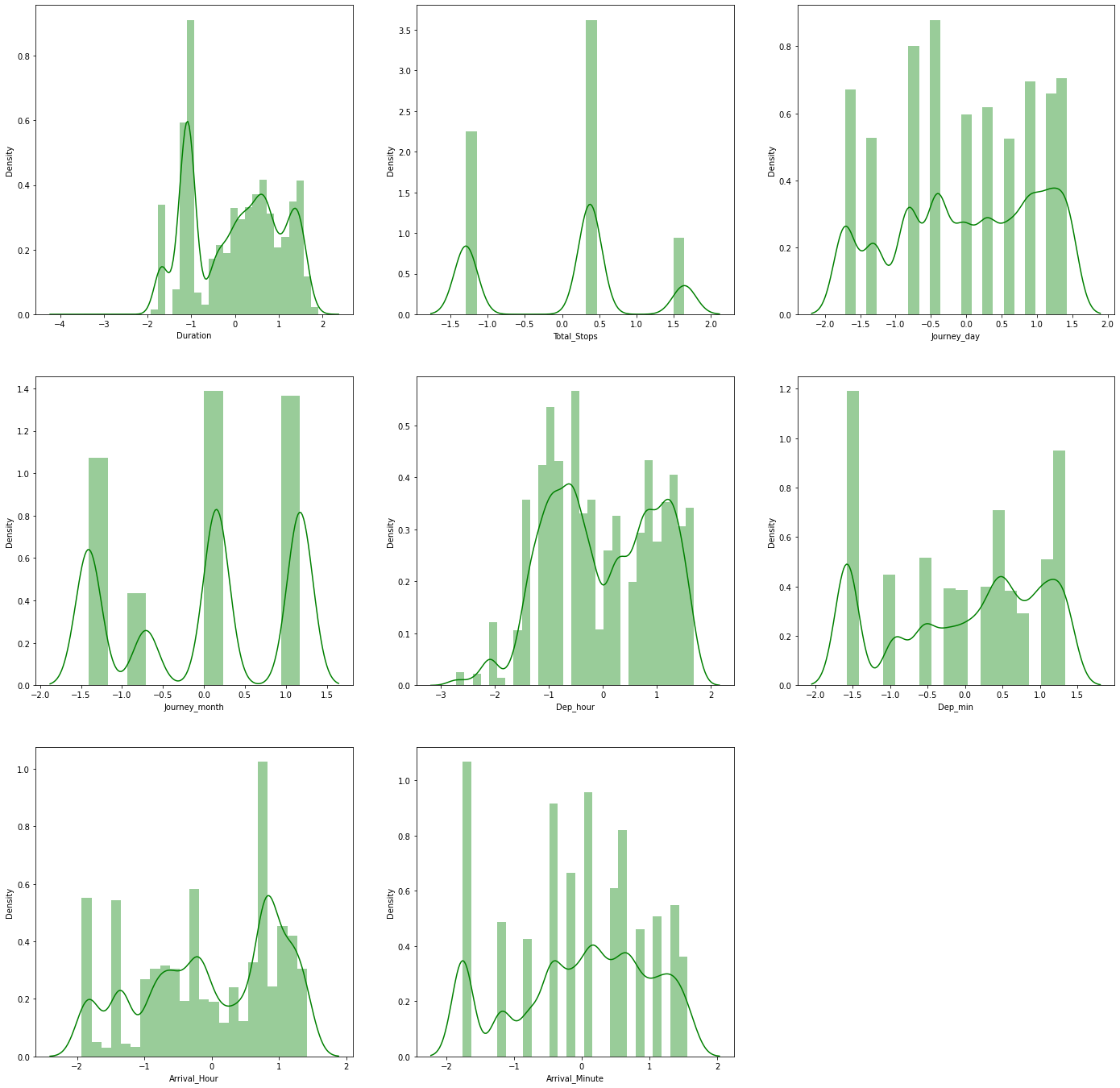
*for col in num\_cols:*

*plt.subplot(3,3,plotnumber)*

*sns.distplot(X[col], color='green')*

*plotnumber+=1*

The output of above code snippet is-:



We can now see that all variables are close to normal distribution.

SCALING THE DATASET

We will now standard Scaler to scale the dataset and save its instance as we later need to use it on test data to scale those values.

*from sklearn.preprocessing import StandardScaler*

*sc = StandardScaler()*

*X = sc.fit\_transform(X)*

*X = pd.DataFrame(X, columns=cols)*

Here also the standard scaler returns a numpy array and we are transforming it back to a dataFrame in the last line.

CHECKING FOR MULTICOLLINEARITY

We will check for multicollineairty in our datast using VIF. If the VIF is above 5 that means there is a multicollinearity problem.

*from statsmodels.stats.outliers\_influence import variance\_inflation\_factor*

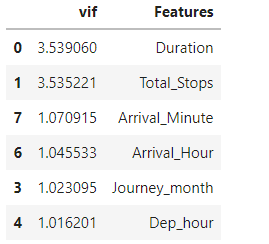
*vif = pd.DataFrame()*

*vif["vif"] = [variance\_inflation\_factor(X[num\_cols].values, i) for i in range(X[num\_cols].shape[1])]*

*vif["Features"] = X[num\_cols].columns*

*vif.nlargest(6, 'vif')*

The output of above code is-:



We can see that VIF is below 5 for all variables. So, there is no multicollinearitry problem.

STEP 4 – BUILDING AND TRAINING MODELS

Now we will split our dataset into training and test set, use cross validation and HyperParameter Tuning to build our models. Use metrics like r2\_score, mean\_squared\_error, mean\_absolute\_error, root\_mean\_sauread\_error to evaluate our model.

The code snipppet given below imports all libraries and splits the training at test set.

*from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score*

*from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.3, random\_state=4)*

*print(X\_train.shape)*

*print(X\_test.shape)*

(7408, 101)

(3175, 101)

For each model we will first import the library and create an instance of that model and train it on the training set. Then we will predict the values on the testing set. We will use cross validation to verify our observsation. The model which will have the least differnce between crosss validation score and testing score will be the best model. In the end we will show a scatter plot of the predicted values and actual values. As close the plots are towards the diagonal, as good our model would be.

MODEL 1 – LINEAR REGRESSION

Linear regression fits a line a close as possible with all the trainig observations and predicts the values in test dataset based on that line. All values are predicted according to the equation of that line.

*from sklearn.linear\_model import LinearRegression*

*lr = LinearRegression()*

*print('Cross Val Scores', cross\_val\_score(lr, X\_train, y\_train, cv=5, scoring='r2'))*

*print("Corss Validation Score Mean===>", cross\_val\_score(lr, X\_train, y\_train, cv=5).mean())*

*lr.fit(X\_train, y\_train)*

*y\_pred\_train = lr.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = lr.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

The output of above code snippet is-:

Cross Val Scores [-3.40807122e+19 -3.10999205e+24 -1.11382948e+25 -1.92691217e+24

-1.41496450e+26]

Corss Validation Score Mean===> -3.1534336542357928e+25

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.7392823700455808

Training MSE : 5406793.131881714

Training MAE : 1553.276535924001

Training RMSE : 2325.2511975874163

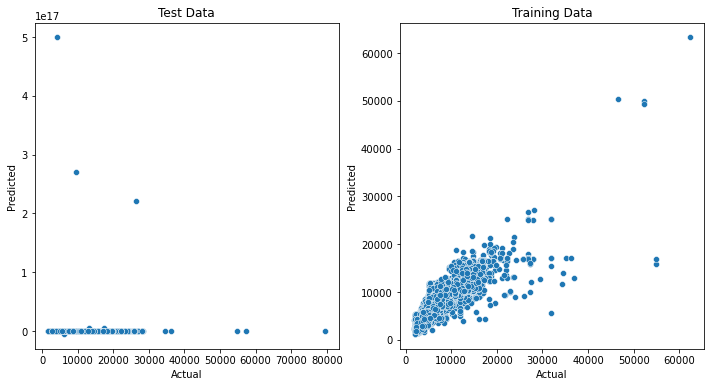
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : -5.246875601130436e+24

Testing MSE : 1.1701257808195525e+32

Testing MAE : 316702379521527.9

Testing RMSE : 1.0817235232810428e+16



We can see that this model is clearly overfitting on training data and the cross val score and testing score are also very low. This model is clearly unable to identify the pattern. Lets check whether after hyperparameter tuning, this model will be able to perfrom better or not.

HYPERPARAMETER TUNING ON LINEAR REGRESSION

*lrcv = LinearRegression()*

*grid\_parmas = {*

*'fit\_intercept':[True, False],*

*'positive':[True, False]*

*}*

*grid\_lr = GridSearchCV(lrcv, param\_grid = grid\_parmas, cv=3, verbose=3)*

*grid\_lr.fit(X\_train, y\_train)*

*print('Best Score -:', grid\_lr.best\_score\_)*

*print('Best Params -:', grid\_lr.best\_params\_)*

Best Score -: -2.831915800928987e+18

Best Params -: {'fit\_intercept': False, 'positive': True}

BUILDING MODEL WITH BEST PARAMETERS AND PREDICTING

*lrcv = LinearRegression(positive=False, fit\_intercept=True)*

*print("Corss Validation Score Mean===>", cross\_val\_score(lrcv, X\_train, y\_train, cv=5).mean())*

*lrcv.fit(X\_train, y\_train)*

*y\_pred\_train = lrcv.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = lrcv.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> -3.1534336542357928e+25

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.7392823700455808

Training MSE : 5406793.131881714

Training MAE : 1553.276535924001

Training RMSE : 2325.2511975874163

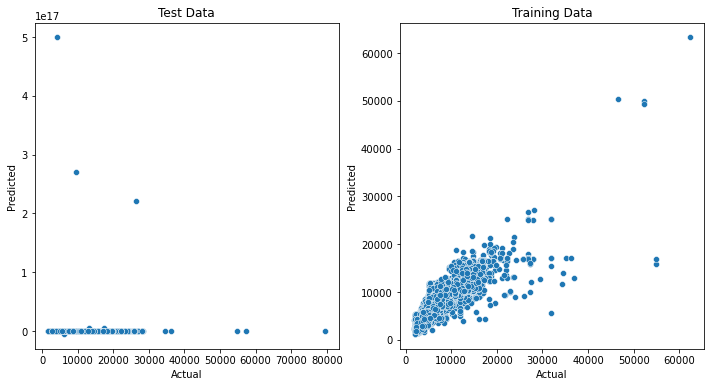
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : -5.246875601130436e+24

Testing MSE : 1.1701257808195525e+32

Testing MAE : 316702379521527.9

Testing RMSE : 1.0817235232810428e+16



Even after hyperParameter Tuning there is no improvement in the performance of the model. Lets try to build a few more models.

MODEL 2 – KNEIGHBORS REGRESSOR

Kneighbors Regressors looks at the k values which are closest to the testing obsrevation and returns the mean of those values.

*from sklearn.neighbors import KNeighborsRegressor*

*knn = KNeighborsRegressor()*

*print("Corss Validation Score Mean===>", cross\_val\_score(knn, X\_train, y\_train, cv=5).mean())*

*knn.fit(X\_train, y\_train)*

*y\_pred\_train = knn.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = knn.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.8015531252554562

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.8880445376959263

Training MSE : 2321745.655512959

Training MAE : 742.0831803455723

Training RMSE : 1523.727552915205

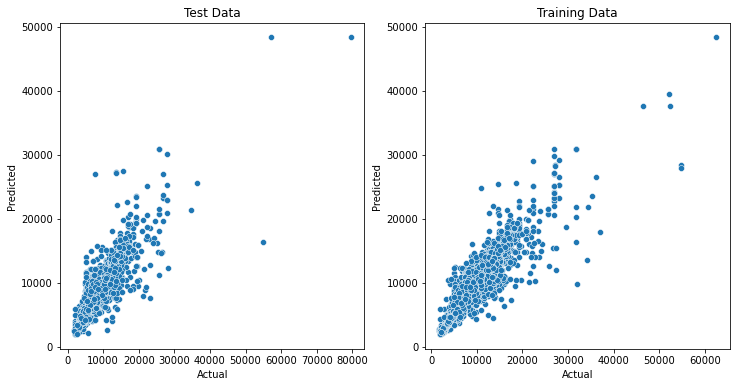
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.8092483662227825

Testing MSE : 4254025.088151181

Testing MAE : 1013.3666771653544

Testing RMSE : 2062.5288090475683



This model is cleary performing much better than LinearRegression model. The test score is close to Cross val score and it also close to 1. All the plots in testing data are also close to the diagonal.Lets check if hyperparameter tuning improve the performance of this model.

HYPERPARAMETER TUNING

*knncv = KNeighborsRegressor()*

*grid\_param = {*

*'n\_neighbors':[3,5,7,9,11,13,15,17,19,21],*

*'weights' : ['uniform', 'distance'],*

*'p':[1,2]*

*}*

*grid\_knn = GridSearchCV(knncv, param\_grid = grid\_param, cv=3, n\_jobs=-1, verbose=3)*

*grid\_knn.fit(X\_train, y\_train)*

*print('Best Score -:', grid\_knn.best\_score\_)*

*print('Best Params -:', grid\_knn.best\_params\_)*

Best Score -: 0.8179939562247472

Best Params -: {'n\_neighbors': 7, 'p': 1, 'weights': 'distance'}

BUILDING MODEL WITH BEST PARAMETERS

*knncv = KNeighborsRegressor(n\_neighbors=7, p=1,weights='distance')*

*print("Corss Validation Score Mean===>", cross\_val\_score(knncv, X\_train, y\_train, cv=5).mean())*

*knncv.fit(X\_train, y\_train)*

*y\_pred\_train = knncv.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = knncv.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.8208602598469031

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.995891877461443

Training MSE : 85194.73244015478

Training MAE : 38.31873200143989

Training RMSE : 291.8813670657221

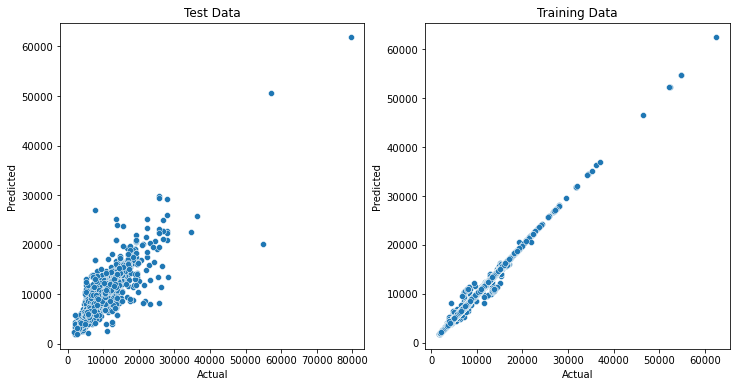
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.8346780251133481

Testing MSE : 3686908.5462822076

Testing MAE : 940.7213627825956

Testing RMSE : 1920.1324293605917



HyperParameter Tuning has improved the performance of our model. But it has a problem of overfitting to training data.

MODEL 3 – DECISION TREE REGRESSOR

This model build up tree accroding to a few conditions.

*from sklearn.tree import DecisionTreeRegressor*

*dt = DecisionTreeRegressor()*

*print("Corss Validation Score Mean===>", cross\_val\_score(dt, X\_train, y\_train, cv=5).mean())*

*dt.fit(X\_train, y\_train)*

*y\_pred\_train = dt.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = dt.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.8091441232203997

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.995891877461443

Training MSE : 85194.73244015478

Training MAE : 38.31873200143989

Training RMSE : 291.8813670657221

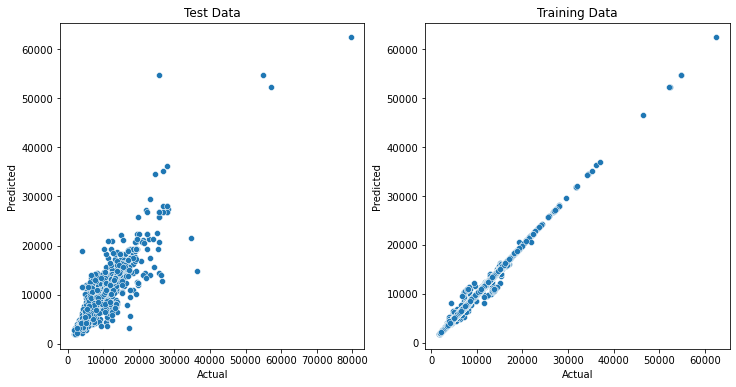
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.8602890530563151

Testing MSE : 3115747.2238582675

Testing MAE : 740.3327559055118

Testing RMSE : 1765.1479325706011



This model performs better on testing data, but there is a difference between testing score and r2 score , so it is not a better model than Kneighbors Regressor. Lets check whether hyperParameter Tuning can make its performance better.

HYPERPARAMETER TUNING

*dtcv = DecisionTreeRegressor()*

*param\_grid = {*

*'max\_depth':[None,10,11,12,13,14,15],*

*'min\_samples\_split':[2,3,4,5,6,7,8],*

*'min\_samples\_leaf':[2,3,4,5,6,7,8],*

*'max\_leaf\_nodes':[None,85,90,95,100,105]*

*}*

*grid\_dt = GridSearchCV(dtcv, param\_grid = param\_grid, cv=3)*

*grid\_dt.fit(X\_train, y\_train)*

*print('Best Score -->', grid\_dt.best\_score\_)*

*print('Best Params -->', grid\_dt.best\_params\_)*

Best Score --> 0.8378059914038821

Best Params --> {'max\_depth': 13, 'max\_leaf\_nodes': None, 'min\_samples\_leaf': 2, 'min\_samples\_split': 3}

BUILDING MODEL WITH BEST PARAMETERS

*dtcv = DecisionTreeRegressor(max\_leaf\_nodes=None, min\_samples\_leaf=2, min\_samples\_split=3, max\_depth=13)*

*print("Corss Validation Score Mean===>", cross\_val\_score(dtcv, X\_train, y\_train, cv=5).mean())*

*dtcv.fit(X\_train, y\_train)*

*y\_pred\_train = dtcv.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = dtcv.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.831207822411909

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.9449878302593283

Training MSE : 1140848.9006891306

Training MAE : 474.87295923404156

Training RMSE : 1068.1052853951855

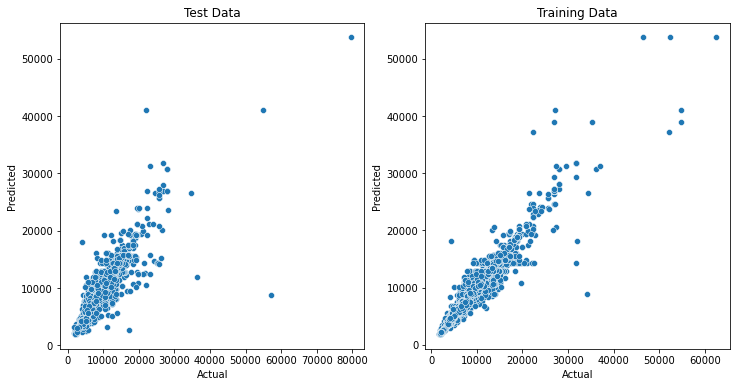
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.8391337084093311

Testing MSE : 3587540.650182795

Testing MAE : 798.3376890736223

Testing RMSE : 1894.0804233671797



This model is performing better than all above as it has higher score than them, and also there is less difference between cross val score and test score. DecisionTree after Hyperparameter Tuning is best performing model until now. Lets try and build a few more models.

MODEL 4 – ADABOOST REGRESSOR

AdaBoost performs boosting rounds, it is an ensemble technique it tries to reduce the error term after each round.

*from sklearn.ensemble import AdaBoostRegressor*

*ad = AdaBoostRegressor()*

*print("Corss Validation Score Mean===>", cross\_val\_score(ad, X\_train, y\_train, cv=5).mean())*

*ad.fit(X\_train, y\_train)*

*y\_pred\_train = ad.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = ad.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.23623416619567875

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.3424512447362912

Training MSE : 13636324.073898444

Training MAE : 3118.8985518623845

Training RMSE : 3692.7393725929865

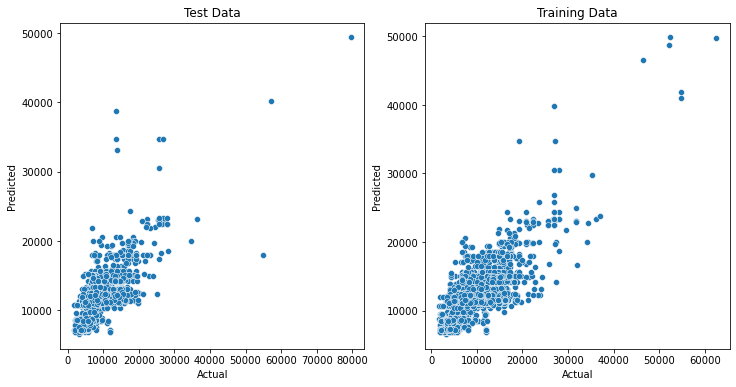
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.3202793874983595

Testing MSE : 15158709.161529712

Testing MAE : 3187.3491834723704

Testing RMSE : 3893.4186984615094



This model is not doing better than previous model, as it has low testing score and there is also very much difference between testing scor and cross val score. Lets check if hyperparameter Tuning can help us improve the performance.

HYPERPARAMETER TUNING

*adcv = AdaBoostRegressor()*

*grid\_params = {*

*'n\_estimators':[10,20,30,50,100, 150],*

*'learning\_rate':[0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.5, 0.7, 0.8],*

*'loss' : ['linear', 'square', 'exponential']*

*}*

*grid\_ad = GridSearchCV(adcv, param\_grid = grid\_params, cv=3, n\_jobs=-1, verbose=3)*

*grid\_ad.fit(X\_train,y\_train)*

*print('Best Score -->', grid\_ad.best\_score\_)*

*print('Best Params -->', grid\_ad.best\_params\_)*

Best Score --> 0.6041336592276539

Best Params --> {'learning\_rate': 0.005, 'loss': 'exponential', 'n\_estimators': 20}

BUILDING MODEL WITH BEST PARAMETERS

*adcv = AdaBoostRegressor(learning\_rate=0.005, loss='exponential', n\_estimators=20)*

*print("Corss Validation Score Mean===>", cross\_val\_score(adcv, X\_train, y\_train, cv=5).mean())*

*adcv.fit(X\_train, y\_train)*

*y\_pred\_train = adcv.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = adcv.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.5824615135627937

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.6119459692019039

Training MSE : 8047510.51505406

Training MAE : 1899.341444176942

Training RMSE : 2836.8134438228503

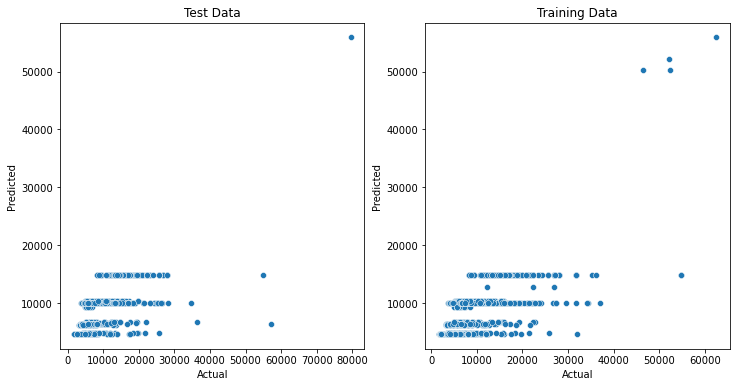
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.5551656494725669

Testing MSE : 9920420.862162754

Testing MAE : 2003.7028004827428

Testing RMSE : 3149.6699608312542



Hyperparameter tuning is also not helping much in improving the performance of this model, as it has still low testing score than DecisionTree Model. Lets try one last model.

MODEL 5 – XTREME GRADIENT BOOST REGRESSOR

This model also peforms boosting rounds and tries to reduce the error term after each boosting round.

*from xgboost import XGBRegressor*

*xg = XGBRegressor()*

*print("Corss Validation Score Mean===>", cross\_val\_score(xg, X\_train, y\_train, cv=5).mean())*

*xg.fit(X\_train, y\_train)*

*y\_pred\_train = xg.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = xg.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.8944218427271637

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.9665821864785172

Training MSE : 693022.5802606648

Training MAE : 500.10539412035024

Training RMSE : 832.4797776887225

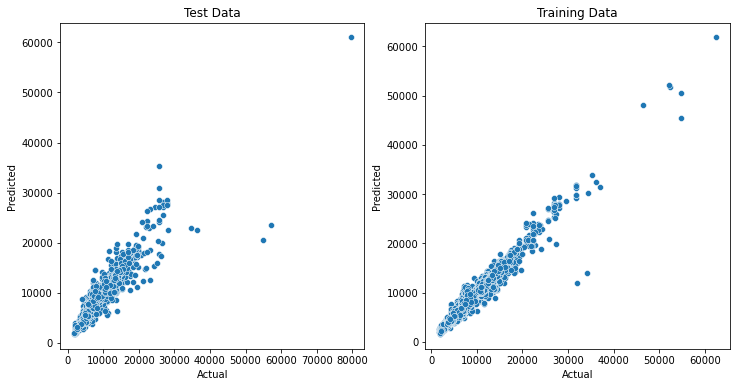
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.8947171565676046

Testing MSE : 2347952.929390822

Testing MAE : 742.1629959707185

Testing RMSE : 1532.3031453961132



This model is performing much better than all other model as it has high testing score and there is also not much differnce between cross val score and testing score.Lets check if we can improve its performance using Hyperparameter tuning.

HYPERPARAMETER TUNING

*xgcv = XGBRegressor()*

*grid\_params = {*

*'n\_estimators':[10,20,50,100,150, 180,200],*

*'max\_depth':[6,7,8,9,10,11,12,13],*

*'learning\_rate':[0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.5],*

*'n\_jobs':[-1]*

*}*

*grid\_xg = GridSearchCV(xgcv, param\_grid=grid\_params, cv=3, n\_jobs=-1, verbose=3)*

*grid\_xg.fit(X\_train, y\_train)*

*print('Best Score -->', grid\_xg.best\_score\_)*

*print('Best Params -->', grid\_xg.best\_params\_)*

Best Score --> 0.8926238959340481

Best Params --> {'learning\_rate': 0.1, 'max\_depth': 9, 'n\_estimators': 180, 'n\_jobs': -1}

BUILDING MODEL WITH BEST PARAMETERS

*xgcv = XGBRegressor(learning\_rate=0.1, max\_depth=9, n\_estimators=180)*

*print("Corss Validation Score Mean===>", cross\_val\_score(xgcv, X\_train, y\_train, cv=5).mean())*

*xgcv.fit(X\_train, y\_train)*

*y\_pred\_train = xgcv.predict(X\_train)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Training R2 Score :", r2\_score(y\_train, y\_pred\_train))*

*print("Training MSE :", mean\_squared\_error(y\_train, y\_pred\_train))*

*print("Training MAE :", mean\_absolute\_error(y\_train, y\_pred\_train))*

*print("Training RMSE :", np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)))*

*y\_pred\_test = xgcv.predict(X\_test)*

*print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")*

*print("Testing R2 Score :", r2\_score(y\_test, y\_pred\_test))*

*print("Testing MSE :", mean\_squared\_error(y\_test, y\_pred\_test))*

*print("Testing MAE :", mean\_absolute\_error(y\_test, y\_pred\_test))*

*print("Testing RMSE :", np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)))*

*plt.figure(figsize=(12,6))*

*plt.subplot(1,2,1)*

*sns.scatterplot(y\_test,y\_pred\_test)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Test Data')*

*plt.subplot(1,2,2)*

*sns.scatterplot(y\_train,y\_pred\_train)*

*plt.xlabel('Actual')*

*plt.ylabel('Predicted')*

*plt.title('Training Data')*

Corss Validation Score Mean===> 0.9037385785575207

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training R2 Score : 0.9822407371129697

Training MSE : 368293.70005267515

Training MAE : 356.86955187078684

Training RMSE : 606.8720623431888

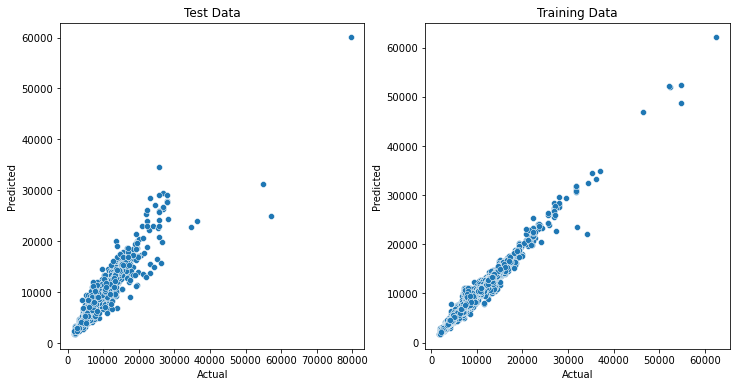
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Testing R2 Score : 0.9120138536479858

Testing MSE : 1962212.6771839682

Testing MAE : 651.3224695112574

Testing RMSE : 1400.790018947868



Hyperparameter Tuning has helped in increasing the model performance. Lets choose this as the final model and save it for further use.

STEP 5 – MODEL SAVING

*import pickle*

*pickle.dump(xgcv, open('model.pickle','wb'))*

STEP 6 – CONCLUSION

We are now able to see what are the factors which affect the pricing of a flight, it can be the airline, the time of departure, which city we are travelling to and many others. But the most important factor was the duration of the flight. While Feature Engineering we saw how the date column can be handled as it is going to have many different values and we cannot treat it as a categorical variable. This was a very good problem in terms of feature engineering. There were a lot of steps taken to build a dataset which was clean and well represnataitve. We also saw that there were very few flights that were of very long duration, we removed them as we had to clean the outliers. Coming to the later part, we saw LinearRegression is not always a good fit for regression problems. Sometimes algorithms which tend to perform better in case of classification( Kneighbors and DecisionTree) problems can perform better than linear Regression. XGBoost turned out the best model for classificaction. One important note, if we want to predict values for new coming data, we have to make sure they have the same columns as we had in our training Set. We can do this by feature engineering.