## **Building a Linear Regression Model**



In statistics, linear regression is**a linear approach to modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables).** The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression.

Target Variable is our Dependent Variable and Rest are Known for Independent Variable.

Problem Statement

Flight ticket price are unpredictable. It is difficult to guess the price of Airlines and we see daily changes in price.

Data Analysis

Moto is to predict the price of flight for various Airlines of Different Cities.

**Attributes**

**Airline**: The name of the airline.

**Date\_of\_Journey**: The date of the journey

**Source**: The source from which the service begins.

**Destination**: The destination where the service ends.

**Route**: The route taken by the flight to reach the destination.

**Dep\_Time**: The time when the journey starts from the source.

**Arrival\_Time**: Time of arrival at the destination.

**Duration**: Total duration of the flight. Pandas – Allow Importing Data from various files like csv, json etc.

Numpy – Numerical Python Consisting of Array.

Seaborn - It is a Data Visualization Library.

Matplotlib – It is used for Plotting the Graph.

Warnings – To filter the warnings.

**Total\_Stops**: Total stops between the source and destination.

**Additional\_Info**: Additional information about the flight

**Price**: The price of the ticket

* Import the Basic Libraries

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sns**

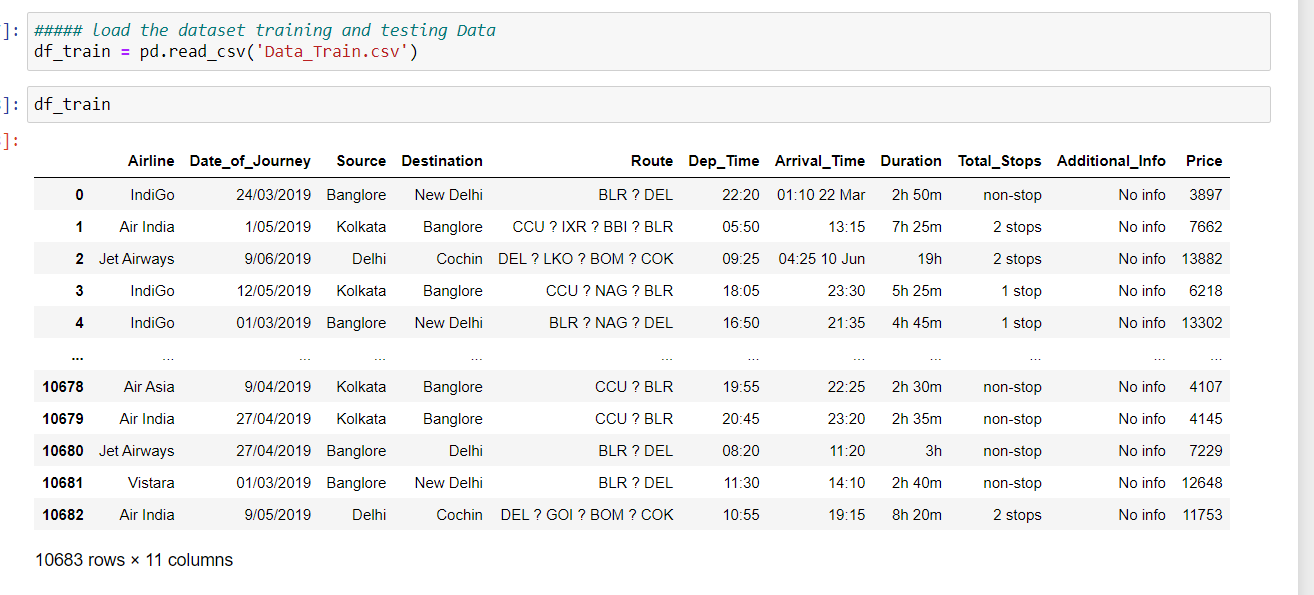
**import** **matplotlib.pyplot** **as** **plt**

**import** **warnings**

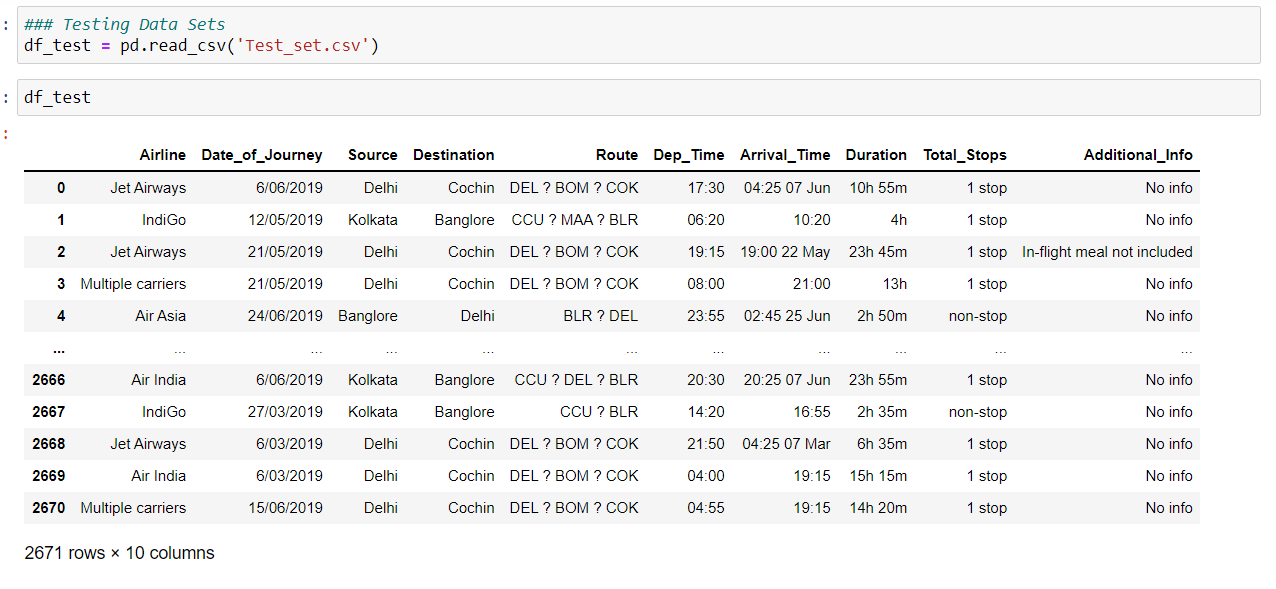
warnings.filterwarnings('ignore')

* Load the Training Datasets

Having 10683 Rows and 11 Columns with Target Variable name Price (Continuous in nature)



* Load the Test Datasets

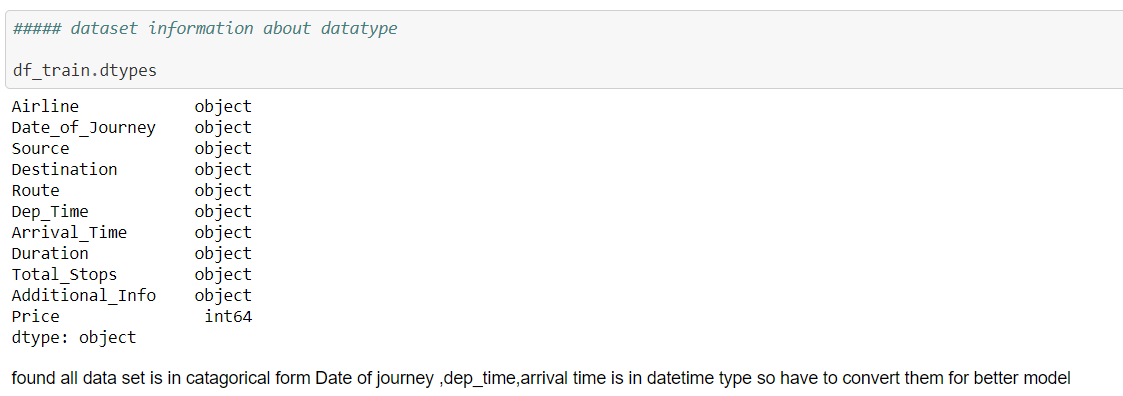
Having 2671 Rows and 10 Columns (No Target Variable is present in Test Dataset)

* We Predict the Model on behalf of Training Dataset.

**EDA (Exploratory Data Analysis)**

* Statistical Information

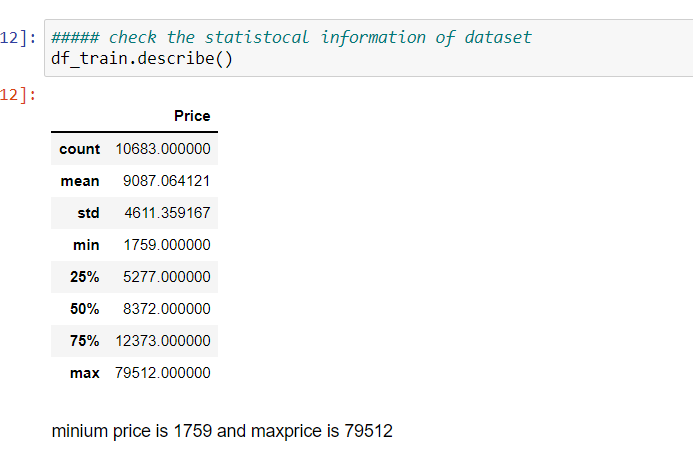
df\_train.dtypes



* Statistical Description

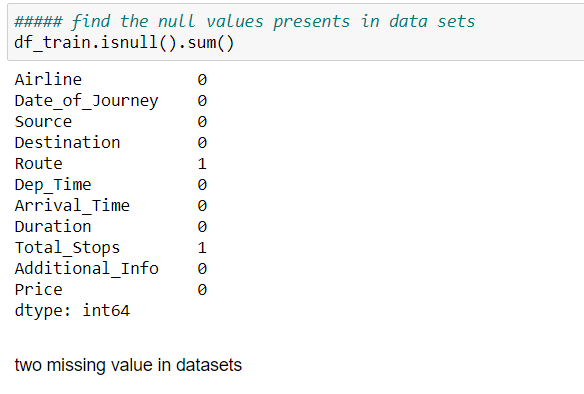
df\_train.describe()

Its shows Number of Counts, Min Value, Max Value ,25th Percentile ,50th Percentile, 75th Percentile, Standard Deviation, Mean.

Only Numeric Value can be seen using describe method.

* Find the Null Value

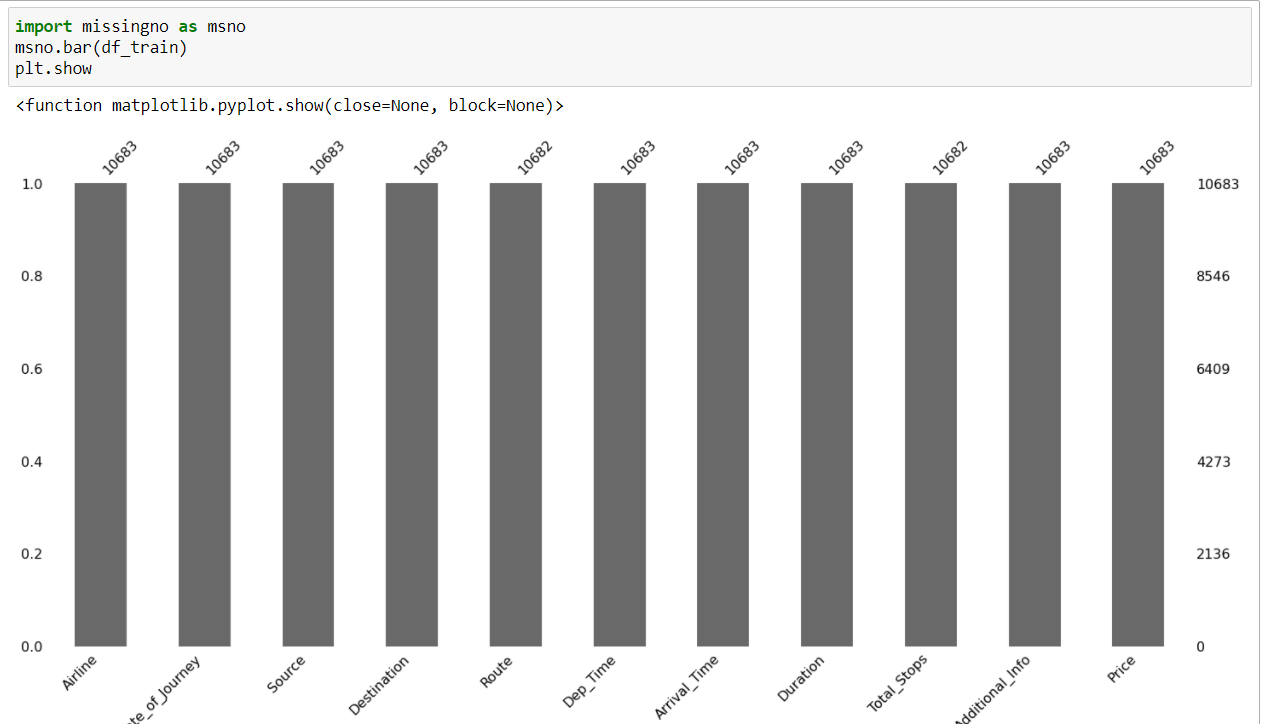
df\_train.isnull().sum()



* Using Graph we can also see the missing value showing in Price and and Additional Info

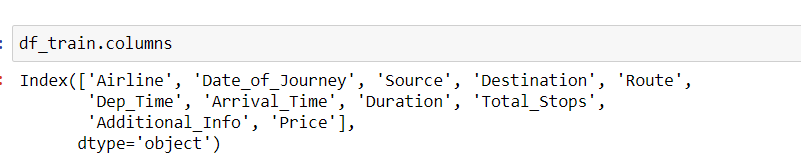
Use pip Command to laod the Libraries / Pacakges





* Number of Columns

train\_df.columns



**Pre-Processing Pipeline and Data Cleaning**

* Pre-Processing is a Techniques to evaluate date in understandable format. Real world is often Incomplete, Inconsistence, lots of Missing Value may or may not depends on Data.
* Pre-Processing deals how data is handling so it can be able to predict good score.
* For Better Model Pre-Processing is one of the major role to deal with value contain unwanted stuff like missing value, unwanted keys like ?, ‘-‘ etc,
* In my case Datasets having only two missing value is Present so in such case drop the Missing value.

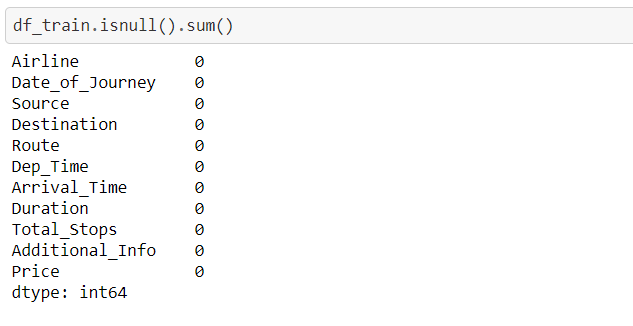
In case more missing value is present in Data fill missing value with Mode, Median and Mean.

Mode – Fill Categorical Value with Mode.

Median/Mean - For Numerical use Median and Mean.

In my case missing value is very less so I have dropped the missing value its not affect the data.

df\_train.dropna(inplace=**True**)



* Convert the columns as per its Data type. Suppose Date of Journey is in Date Format Convert into datetime. It helps to find the better prediction of Model.

Extract the Day, Month, Year from Date column and drop the Original one. Because model understand Numerical value well. Let’s Have a Code to convert into Date time and Extract into individual Columns.

Create function to Change\_into\_Datetime and Extract the columns.

def change\_into\_datetime(col):

df\_train[col]=pd.to\_datetime(df\_train[col])

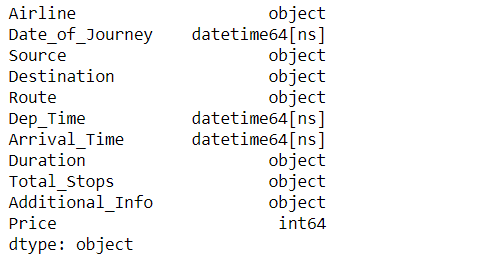
Call change into datetime function to convert the columns in Datetime Format

**for** i **in** ['Date\_of\_Journey','Dep\_Time','Arrival\_Time']:

change\_into\_datetime(i)

check the date types weather it works or not.

df\_train.dtypes



Extract the Datetime columns into separate columns (Day, Month , Year )

*# Extract Date and Month from Data of journey*

df\_train['journey\_day'] = df\_train['Date\_of\_Journey'].dt.day

df\_train['journey\_month'] = df\_train['Date\_of\_Journey'].dt.month

Here I have store Day and Month Columns Only because Year is unique one for all so its not important for me, if it helpful than keep year columns also

Let’s Create function to extract the hours and Minute’s columns and extract them to separate in new columns.

*##### craete Function to Extract Hours and Mins from columns like dep\_time,Arrival\_time*

**def** extract\_hrs(data,col):

data[col+'\_hrs'] = data[col].dt.hour

**def** extract\_min(data,col):

data[col+'mins'] = data[col].dt.minute

*#### lastly drop the columns*

**def** dropColumns(data,col):

data.drop(col,axis=1,inplace=**True**)

*##### call the function to extract hrs from dep\_time*

extract\_hrs(df\_train,'Dep\_Time')

*##### call the function to extract min from dep\_time* extract\_min(df\_train,'Dep\_Time')

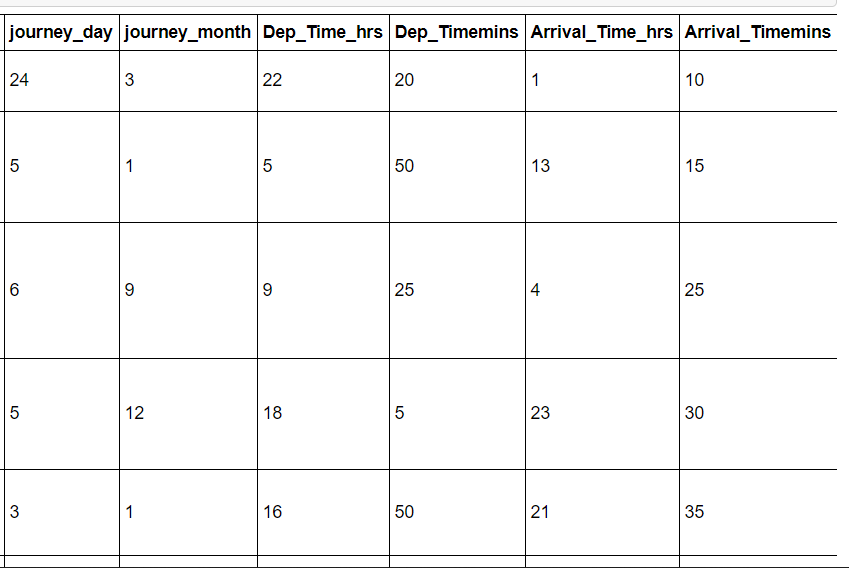
*#### call function to drop the columns after extracting data*

dropColumns(df\_train,'Dep\_Time')

*##### call function to drop the hrs and min from arrival\_time* extract\_hrs(df\_train,'Arrival\_Time') extract\_min(df\_train,'Arrival\_Time')

*#### call function to drop the columns after extracting data* dropColumns(df\_train,'Arrival\_Time')

See the Data after Extracting the columns



There is also inbuilt pandas function to extract Hours and Minutes from Duration Columns

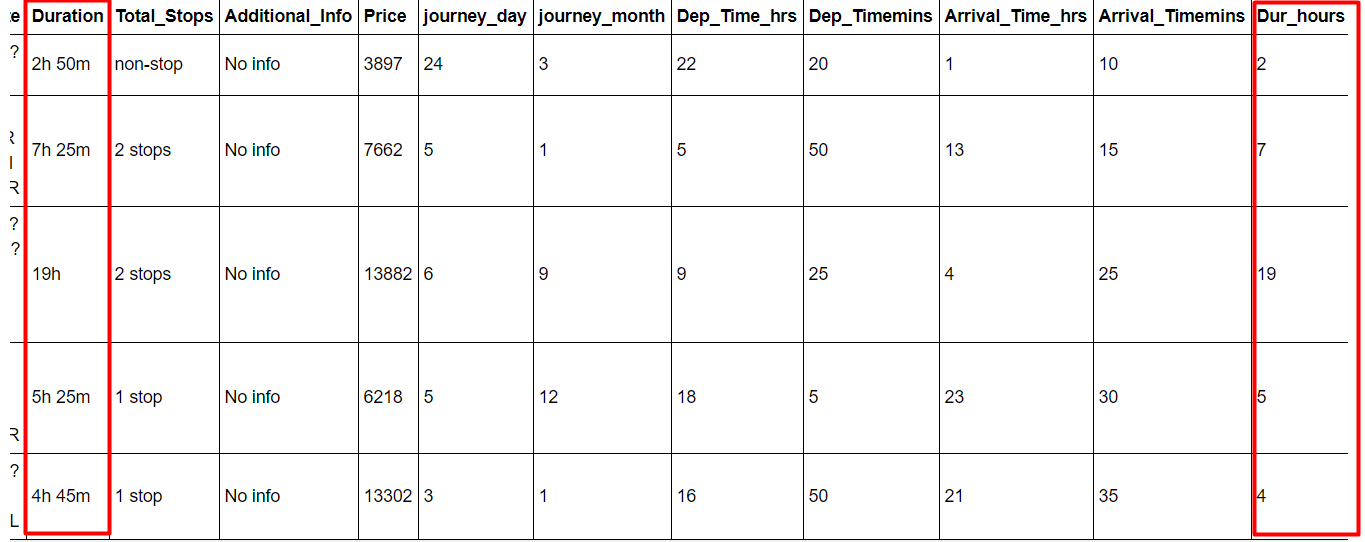
*#https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to\_timedelta.html*

s=pd.to\_timedelta(df\_train['Duration'])

s

df\_train['Dur\_hours']=s.dt.components['hours']

df\_train['Dur\_minutes']=s.dt.components['minutes']



Same columns for Minutes .

Now extract the duration columns hence we have extract the columns and separated them into two individual columns.

*#### after extracting drop the duration columns from datasets* df\_train.drop(['Duration'],axis=1,inplace=**True**)

Use of inplace=True means its removed from data finally.

* Check for the Categorical Columns in Data Using the following Ways:

columns = [columns **for** columns **in** df\_train.columns **if** df\_train[columns].dtypes=='object']

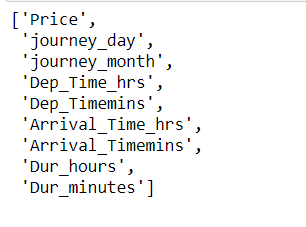
columns



* Check for the continuous Columns

count\_col = [count\_col **for** count\_col **in** df\_train.columns **if** df\_train[count\_col].dtypes!='object']

count\_col



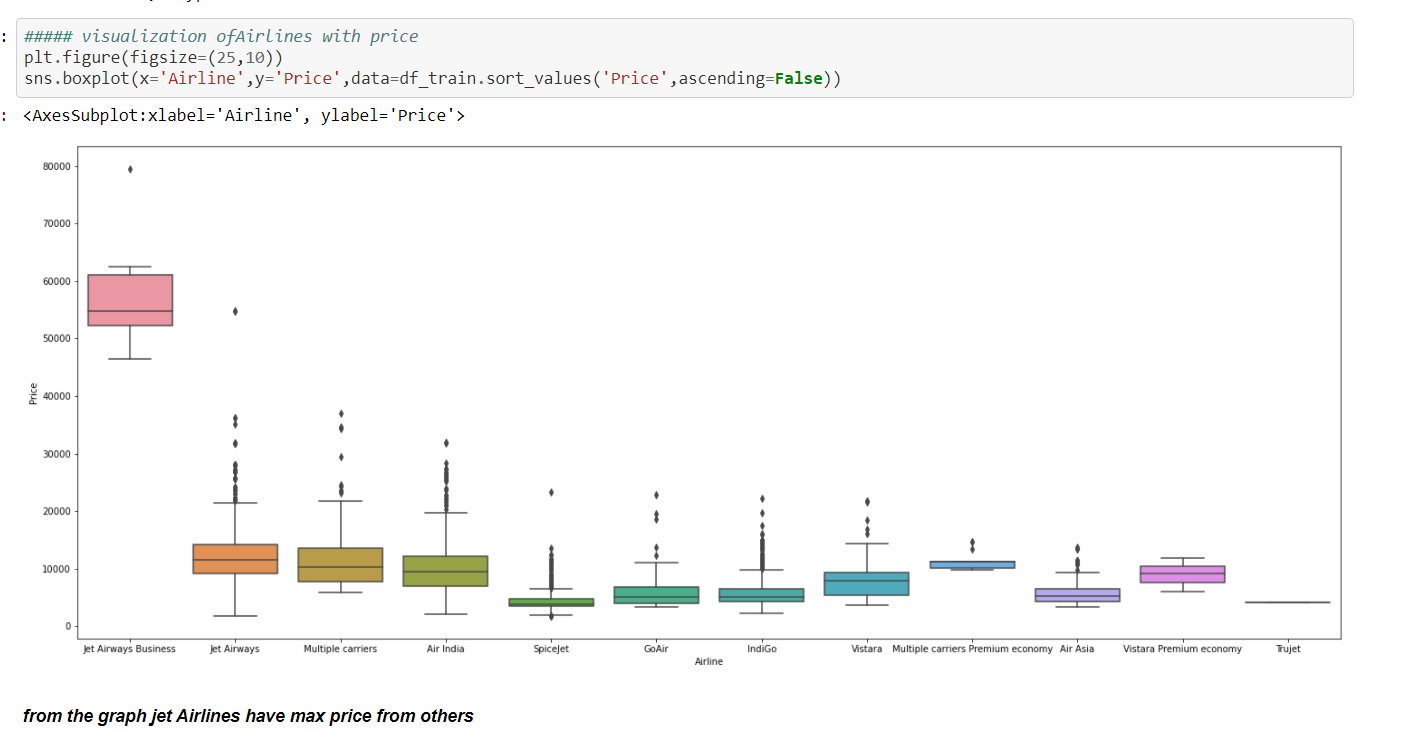
* To handle the categorical columns with Encoding Techniques.

Nominal Data – Data that are Not in Order use One Hot Encoding.

Ordinal Data – Data are in Order use Label Encoder.

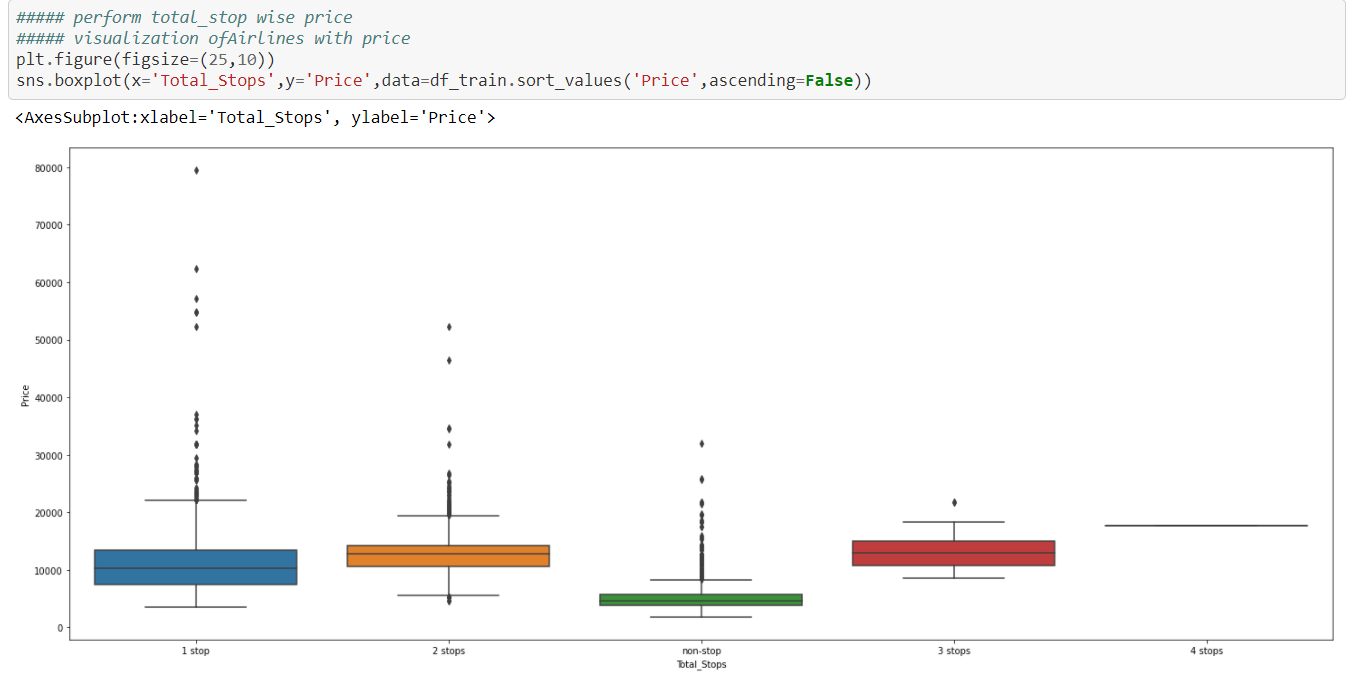
* Visualization of Data helps in cleaning of Data and It became easy to handle.

*##### visualization ofAirlines with price* plt.figure(figsize=(25,10)) sns.boxplot(x='Airline',y='Price',data=df\_train.sort\_values('Price',ascending=**False**))



plt.figure(figsize=(25,10))

sns.boxplot(x='Total\_Stops',y='Price',data=df\_train.sort\_values('Price',ascending=**False**))

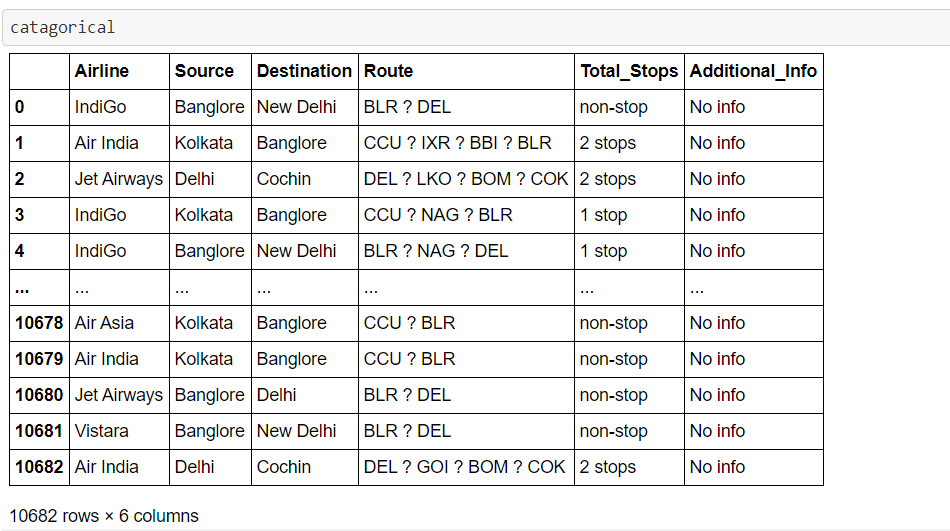


From graph Jet Airways Business have Max Price from Others.

It easy to see the Total Stops wise Price.

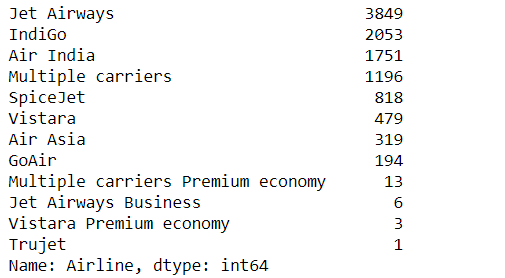
Its better to show graph of Categorical columns Before Encoding it’s given you the better visualization because encoding of columns shows in 0 and 1for categorical value.

* For Nominal Data Use One Hot Encoding such as Airlines, Source, Destination etc.
* Use **pd.get\_dummies** to convert into one hot Encoding. On applying one hot encoding how values look like let’s have a look.



See the number of Counts

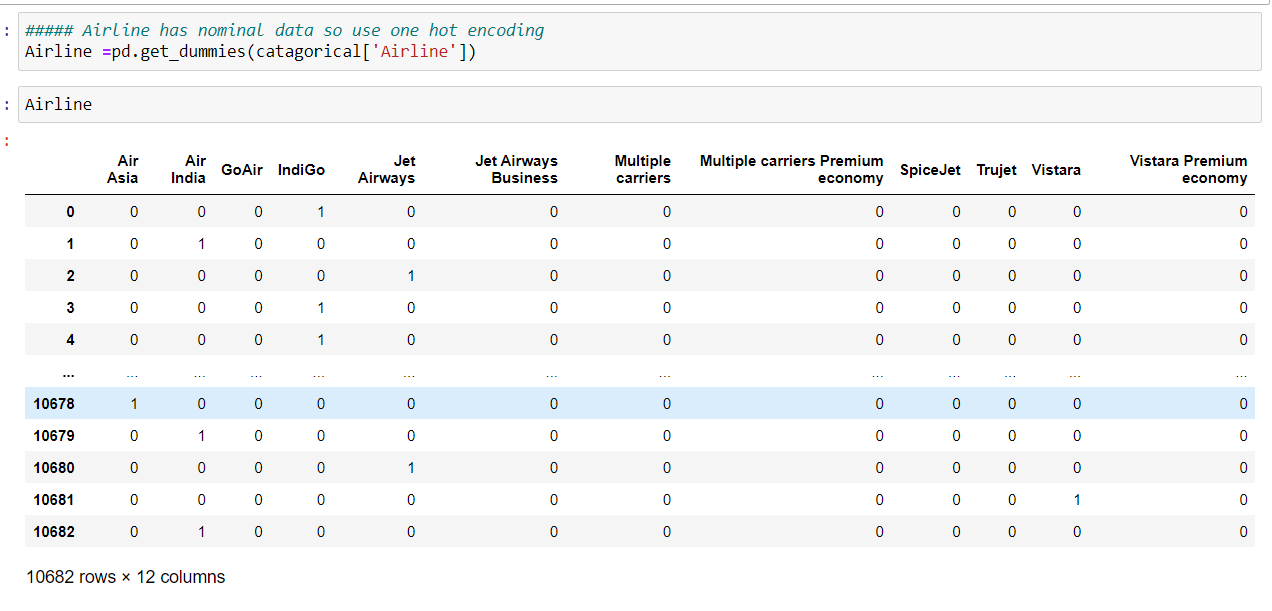
catagorical['Airline'].value\_counts()



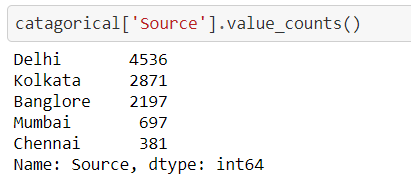
*##### Airline has nominal data so use one hot encoding*

Airline =pd.get\_dummies(catagorical['Airline'])

After Applying Encoding Techniques

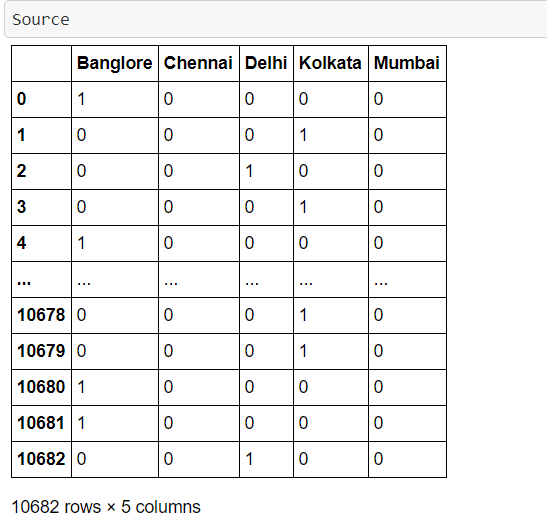


catagorical['Source'].value\_counts()



*##### Source is also nominal data apply one hot enconding*

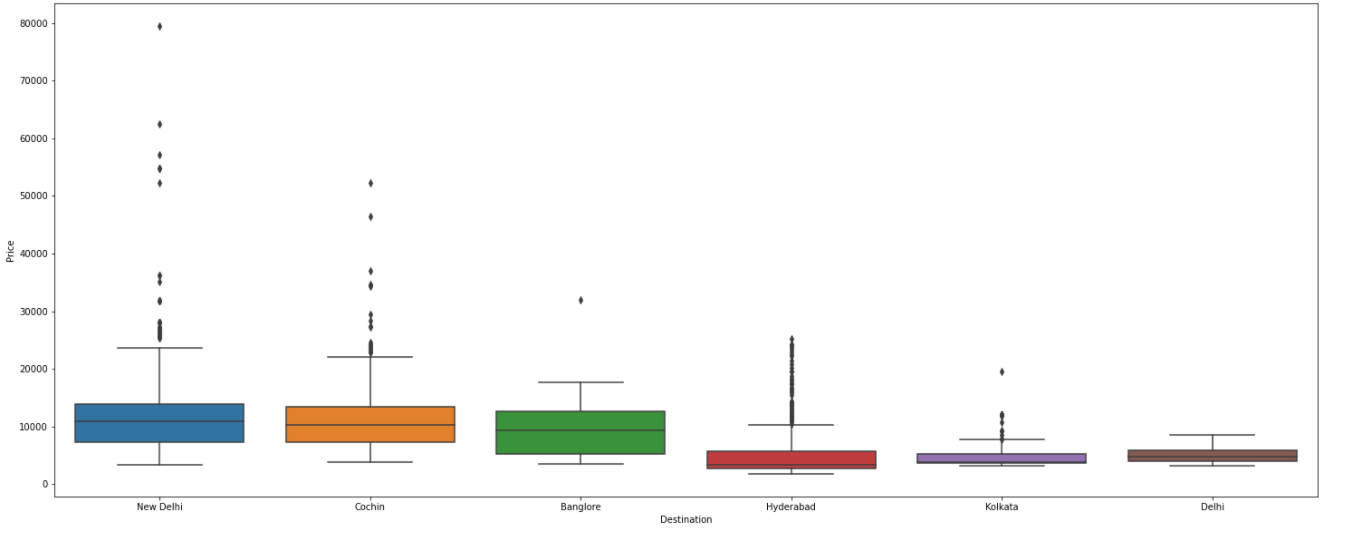
Source = pd.get\_dummies(catagorical['Source'])



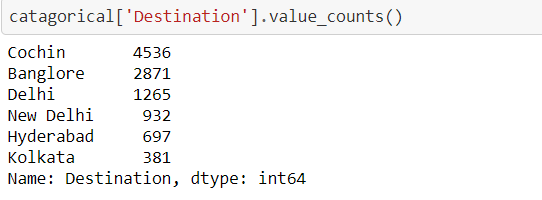
Same for Destination Columns

*##### perform desitination wise price*

plt.figure(figsize=(25,10)) sns.boxplot(x='Destination',y='Price',data=df\_train.sort\_values('Price',ascending=**False**))

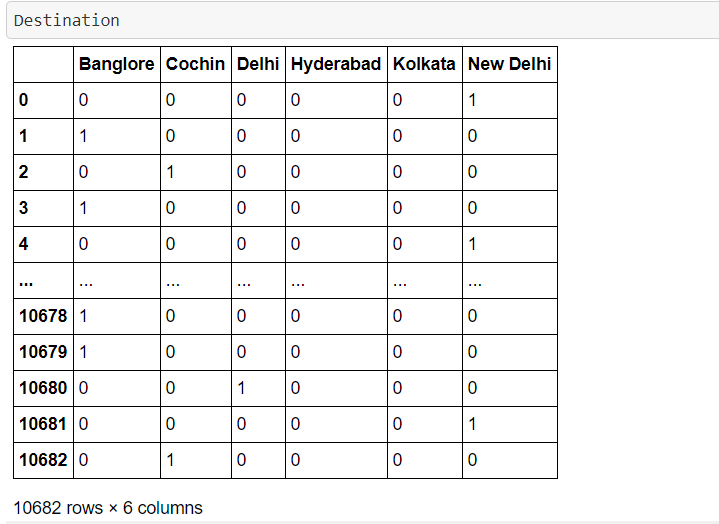


catagorical['Destination'].value\_counts()



*##### Destination is also nominal Data apply one hot encoding tech*

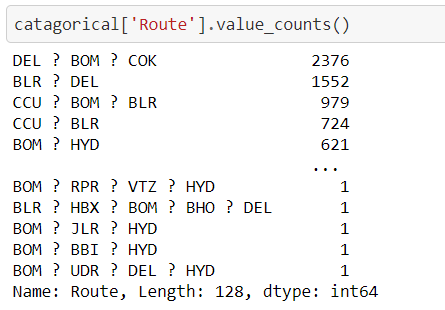
Destination = pd.get\_dummies(catagorical['Destination'])



Here I have Encoded the many columns and stored in separate variables, it helps me to visualize all the columns separately and also my original data set will remain same so I can identify changes

Continue with Pre-Processing Techniques will show more understandable of data.

Let’s Handle the Route columns having some stuff like question mark.



Have to split the ? from route columns and separate each route with new columns

*##### lets split ? from Route and separate in different columns*

catagorical['Route1']=catagorical['Route'].str.split('?').str[0]

catagorical['Route2']=catagorical['Route'].str.split('?').str[1]

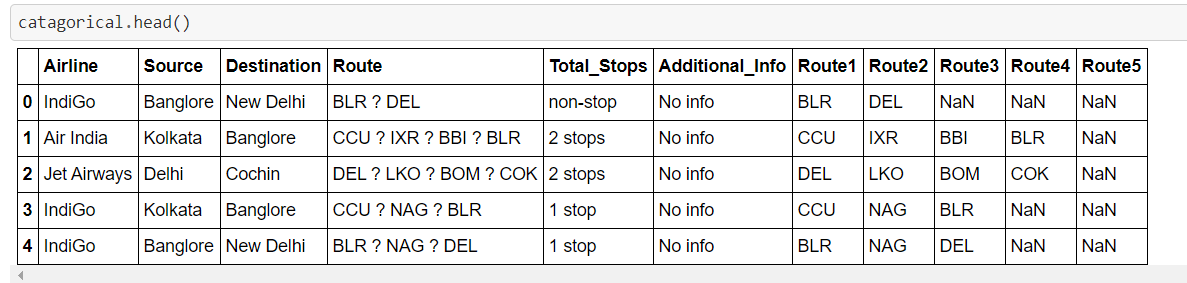
catagorical['Route3']=catagorical['Route'].str.split('?').str[2]

catagorical['Route4']=catagorical['Route'].str.split('?').str[3]

catagorical['Route5']=catagorical['Route'].str.split('?').str[4]

#### Use of head menthod to find first five columns

catagorical.head()



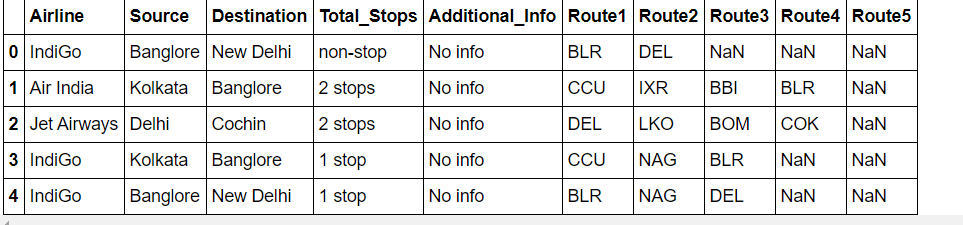
It has been separated Route 1, Route 2, Route 3, Route 4, Route 5

All separated columns contain having null lets handle them. Here we can’t drop value having NaN . Use Mode Method to fill all missing values.

Let’s Drop the Original Columns and fill the missing values

*##### from extract data Route lets drop the columns*

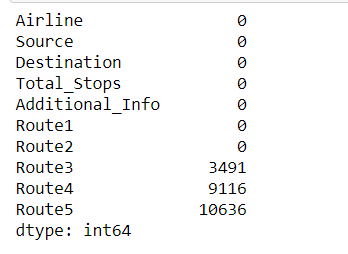
catagorical.drop(['Route'],axis=1,inplace=**True**)



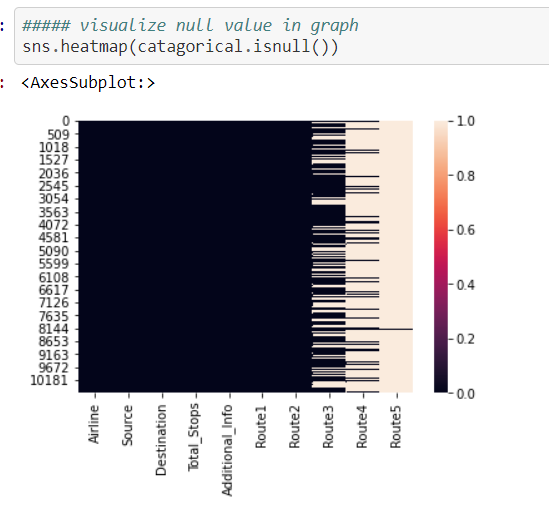
Check for null values.

*###### check null value in catagorical columns*

catagorical.isnull().sum()



Visualize null value using graph we use seaborn library for visualization *##### visualize null value in graph* sns.heatmap(catagorical.isnull())

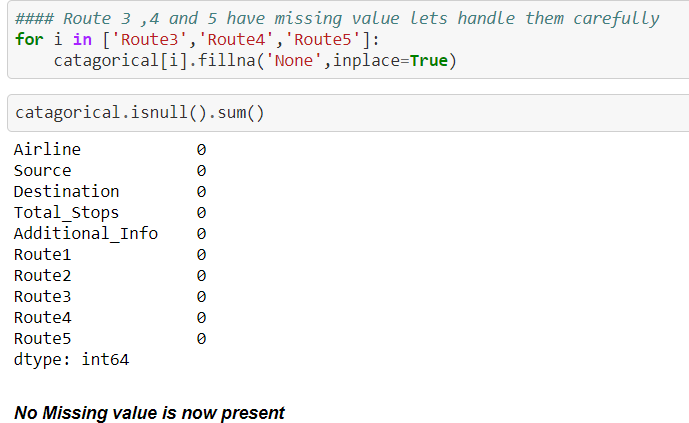


Null value seems to be in different color in graph see its easy to identify.

**for** i **in** ['Route3','Route4','Route5']:

catagorical[i].fillna('None',inplace=**True**)

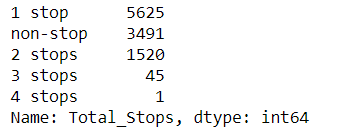
I have fill Null Value with None because some routes have no stoppage so its better to fill with None.



Encode Total stops.

*### lets encode total\_stops*

catagorical['Total\_Stops'].value\_counts()

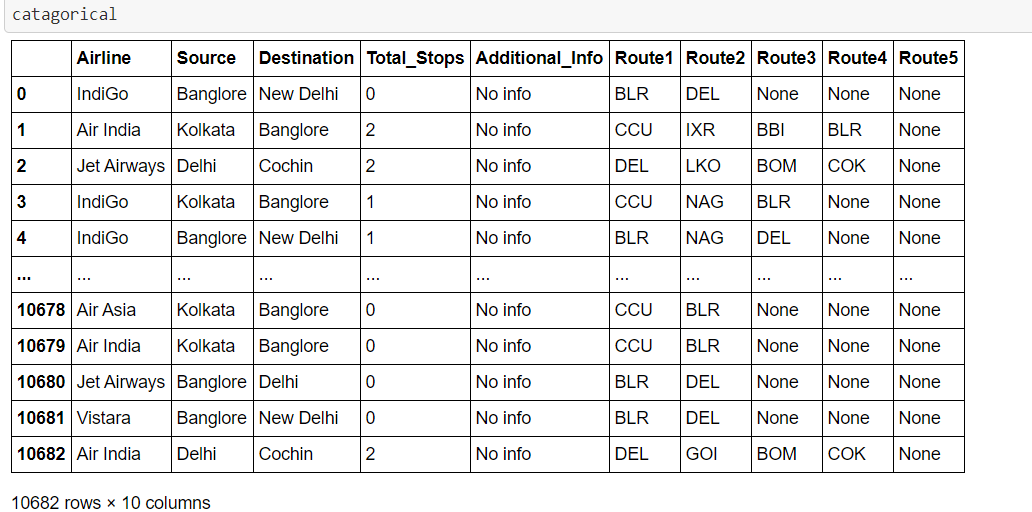


dict = {'non-stop':0,'1 stop':1,'2 stops':2,

'3 stops':3,'4 stops':4}

catagorical['Total\_Stops'] = catagorical['Total\_Stops'].map(dict)

catagorical



Let’s Apply Label Encoder Techniques.

Import Sklearn Libraries

**import** **sklearn**

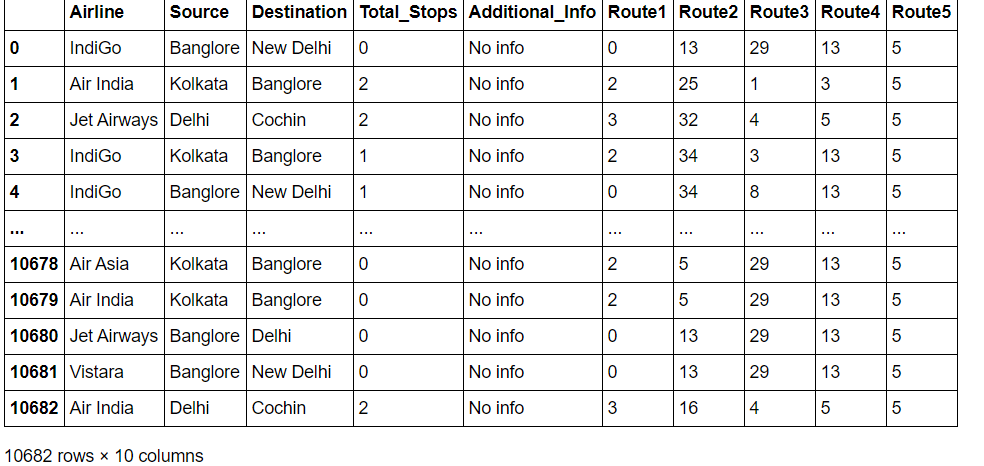
**from** **sklearn.preprocessing** **import** LabelEncoder

le=LabelEncoder()

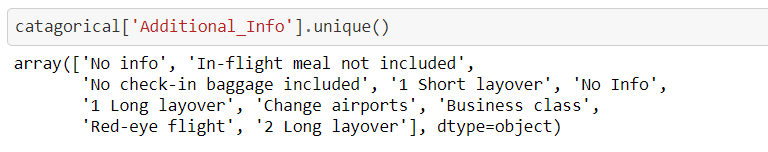
**for** i **in** ['Route1','Route2','Route3','Route4','Route5']:

catagorical[i]=le.fit\_transform(catagorical[i])

catagorical



catagorical['Additional\_Info'].unique()



I have drop the Additional Columns because additional columns may not effect the price of Airlines.

*##### drop the additional\_information because additional information may not effect the price*

catagorical.drop(['Additional\_Info'],axis=1,inplace=**True**)

Drop the Columns which has been Encoded.

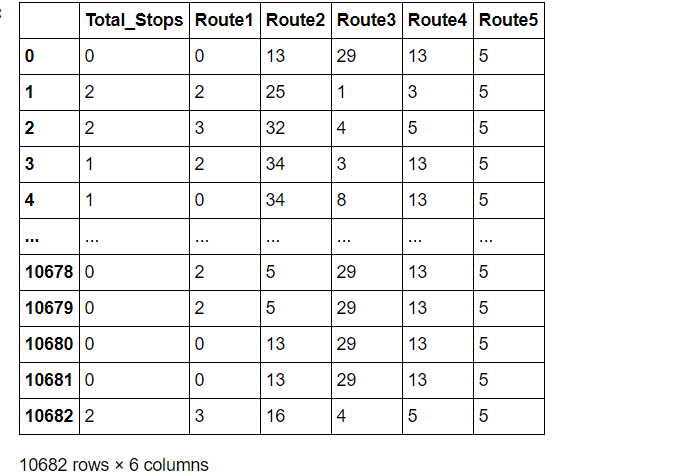
*#### drop columns that has been encoded*

catagorical.drop(['Airline'],axis=1,inplace=**True**)

catagorical.drop(['Source'],axis=1,inplace=**True**)

catagorical.drop(['Destination'],axis=1,inplace=**True**)

catagorical

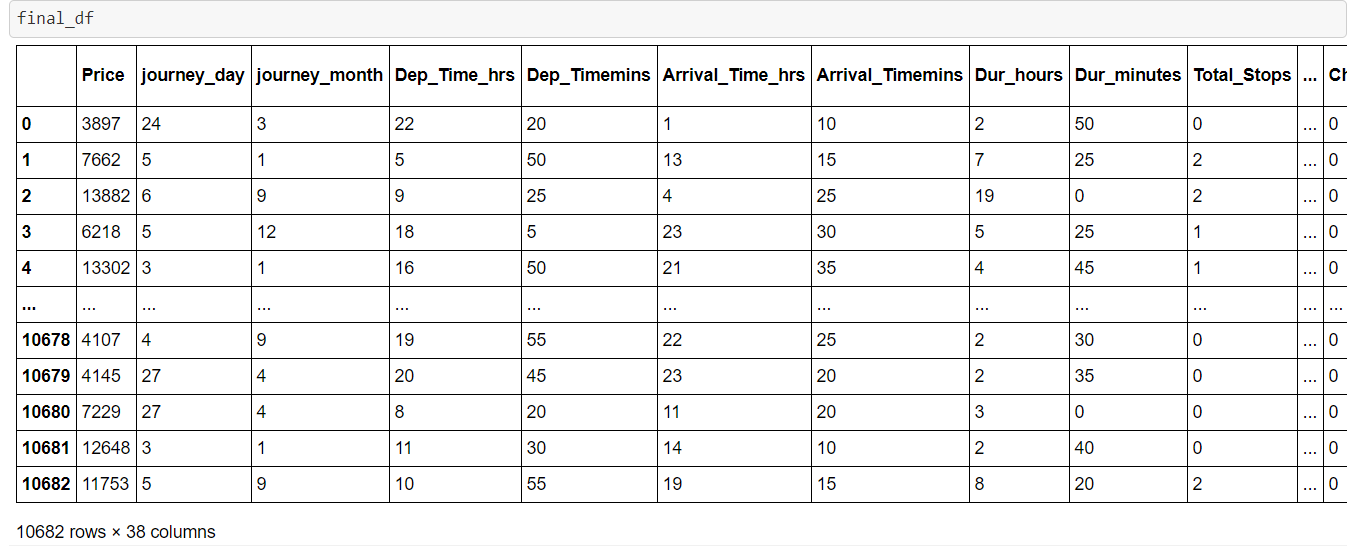


Concat all columns to make new dataframe which is ready for modeling

*##### lets concat the data for modeling*

final\_df =pd.concat([df\_train[count\_col],catagorical,Airline,Source,Destination],axis=1)

final\_df



Check for the Outliers in column price which is our target variable

*##### check for the outliers in continous column*

**def** check\_outlier(data,col):

fig,(ax1,ax2)=plt.subplots(2,1)

sns.distplot(data[col],ax=ax1)

sns.boxplot(data[col],ax=ax2)

check\_outlier(final\_df,'Price')

*##### found outliers in columns price*

*#### lets find median*

median = np.median(final\_df['Price'])

final\_df['Price']=np.where(final\_df['Price']>=40000,median,final\_df['Price'])

check\_outlier(final\_df,'Price')

Check for shape loss of data may leads to bad prediction of model

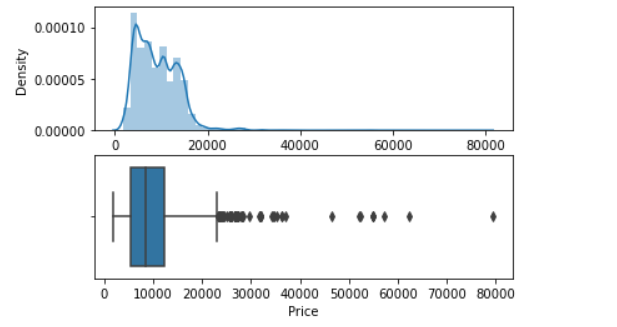
final\_df.shape *### new shape*



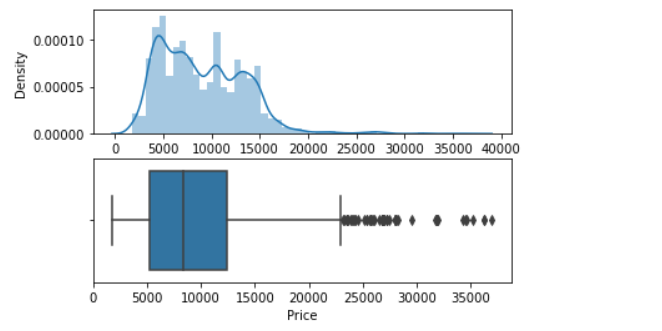
df\_train.shape



Before Removal of Outliers



After Removal of Outliers



Some Outliers seems to handle

**Modeling**

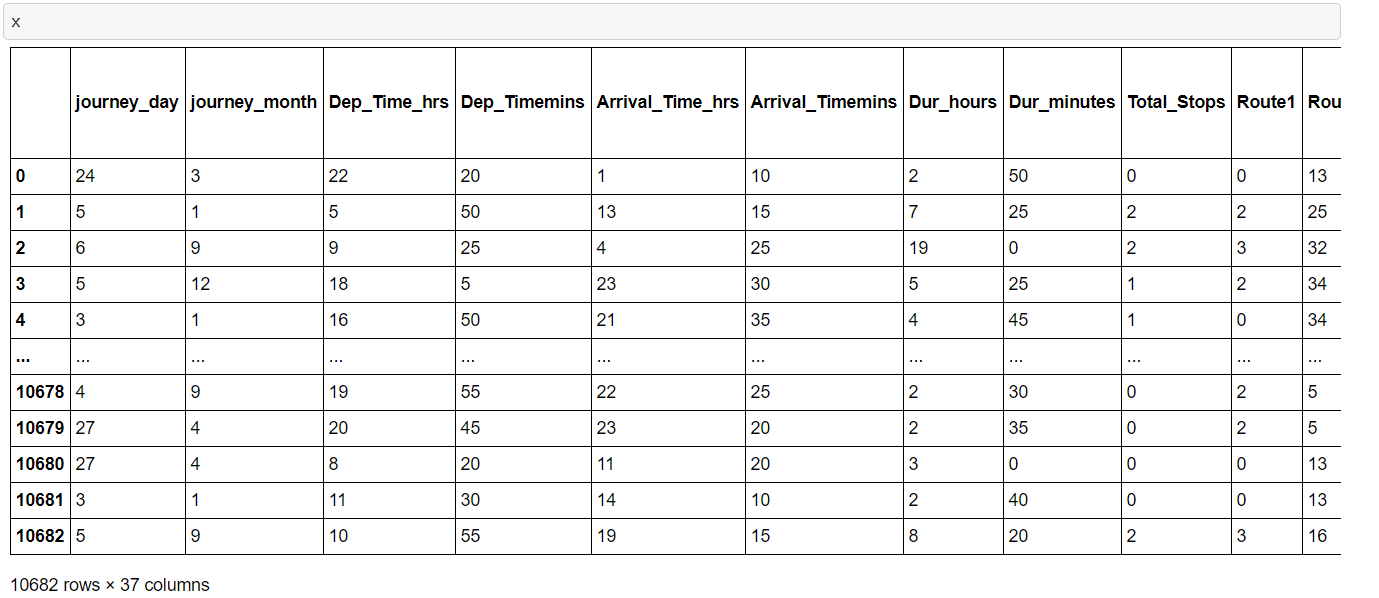
* Now data is Ready for Modelling.

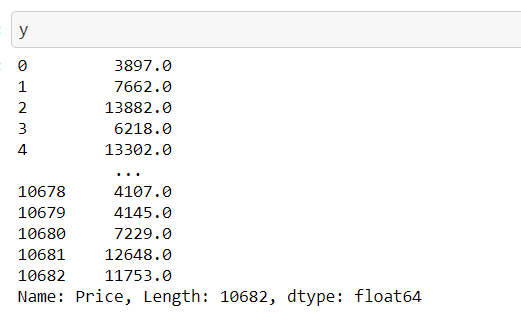
Separate the Independent Columns and Dependent Columns

*#####separate the datasets in X and Y columns*

y=final\_df['Price']

x=final\_df.drop('Price',axis=1)





* Import the Libraries for modelling

**from** **sklearn.linear\_model** **import** LinearRegression

**from** **sklearn.model\_selection** **import** cross\_val\_score

**from** **sklearn.ensemble** **import** GradientBoostingRegressor,RandomForestRegressor

**from** **sklearn.tree** **import** DecisionTreeRegressor rf=RandomForestRegressor()

dtc = DecisionTreeRegressor()

lr=LinearRegression()

**from** **sklearn.metrics** **import** r2\_score

**from** **sklearn.neighbors** **import** KNeighborsRegressor

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.20,random\_state=39)

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

* Define Function Predict

**def** predict(ml\_model):

print('Model is : **{}**'.format(ml\_model))

model = ml\_model.fit(x\_train,y\_train)

print("Training Score : **{}**".format(model.score(x\_train,y\_train))) predictions = model.predict(x\_test)

print("Predictions are : **{}**" ,format(predictions))

print('**\n**')

print('Testing Prediction')

r2score = r2\_score(y\_test,predictions)

print("r2 Score is : **{}**",format(r2score))

print('Cross Validation Score:**{}**'.format(cross\_val\_score(ml\_model,x\_train,y\_train,cv=5,scoring='r2')))

print('MAE: **{}**'.format(mean\_absolute\_error(y\_test,predictions))) print('MSE: **{}**'.format(mean\_squared\_error(y\_test,predictions))) print('RMSE: **{}**'.format(np.sqrt(mean\_squared\_error(y\_test,predictions)))) print('**\n**')

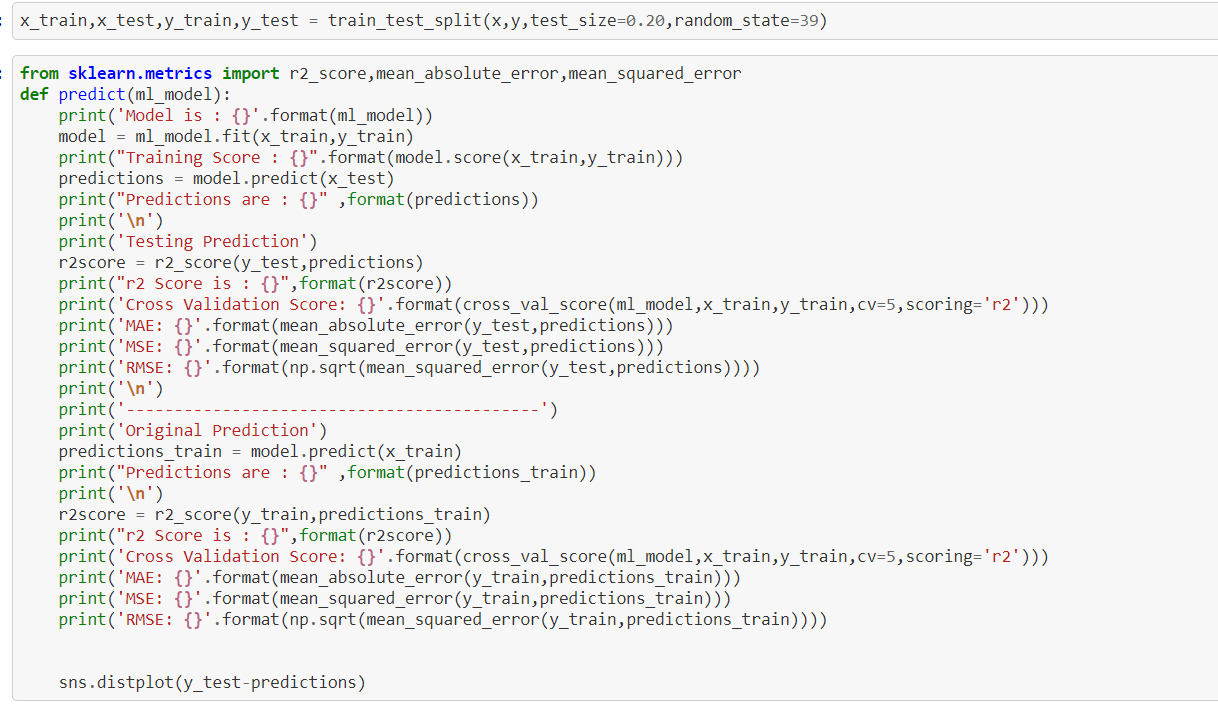
print('-------------------------------------------') print('Original Prediction')

predictions\_train = model.predict(x\_train)

print("Predictions are : **{}**" ,format(predictions\_train)) print('**\n**') r2score = r2\_score(y\_train,predictions\_train) print("r2 Score is : **{}**",format(r2score))

print('Cross Validation Score: **{}**'.format(cross\_val\_score(ml\_model,x\_train,y\_train,cv=5,scoring='r2')))

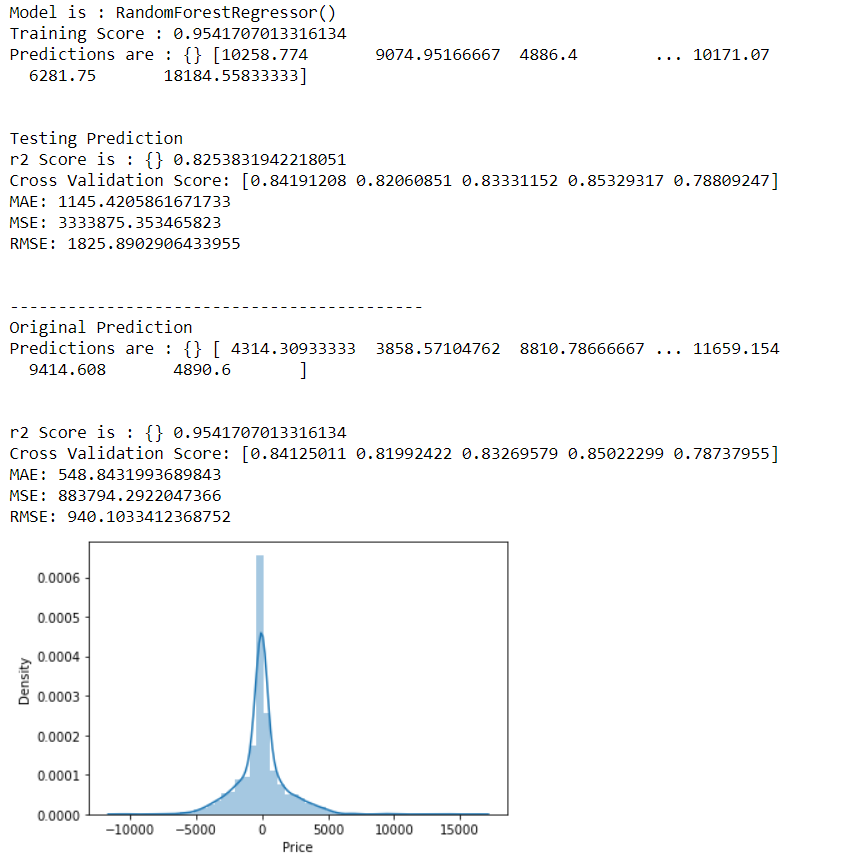
print('MAE: **{}**'.format(mean\_absolute\_error(y\_train,predictions\_train))) print('MSE: **{}**'.format(mean\_squared\_error(y\_train,predictions\_train))) print('RMSE: **{}**'.format(np.sqrt(mean\_squared\_error(y\_train,predictions\_train)))) sns.distplot(y\_test-predictions)



* RandomForestRegressor

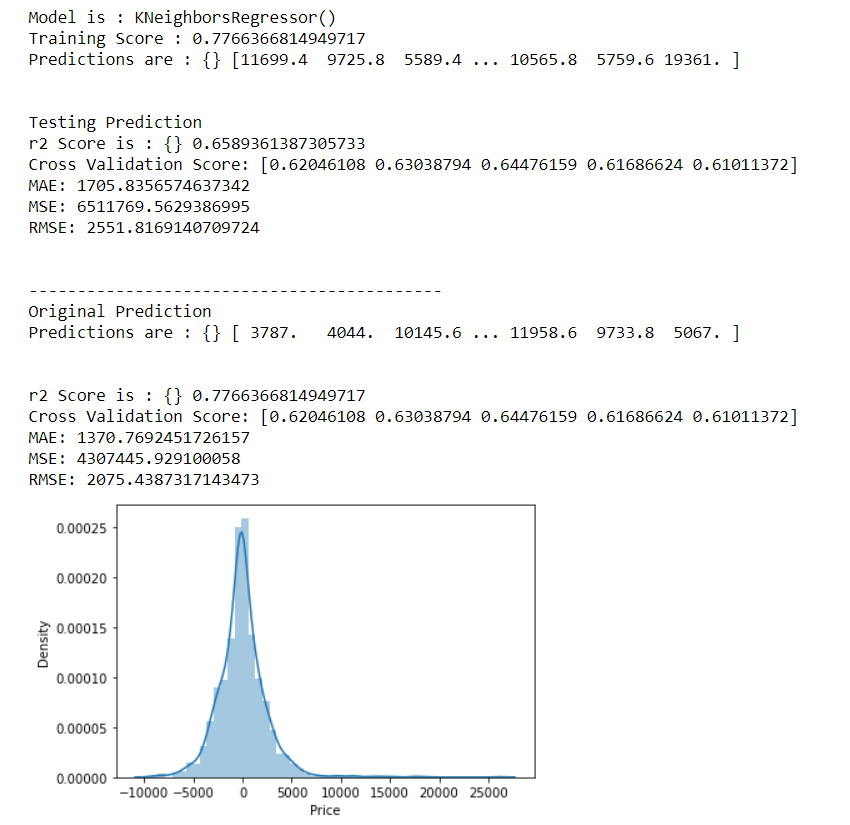
Esemble Techniques Not sensitive to Outliers

predict(RandomForestRegressor())



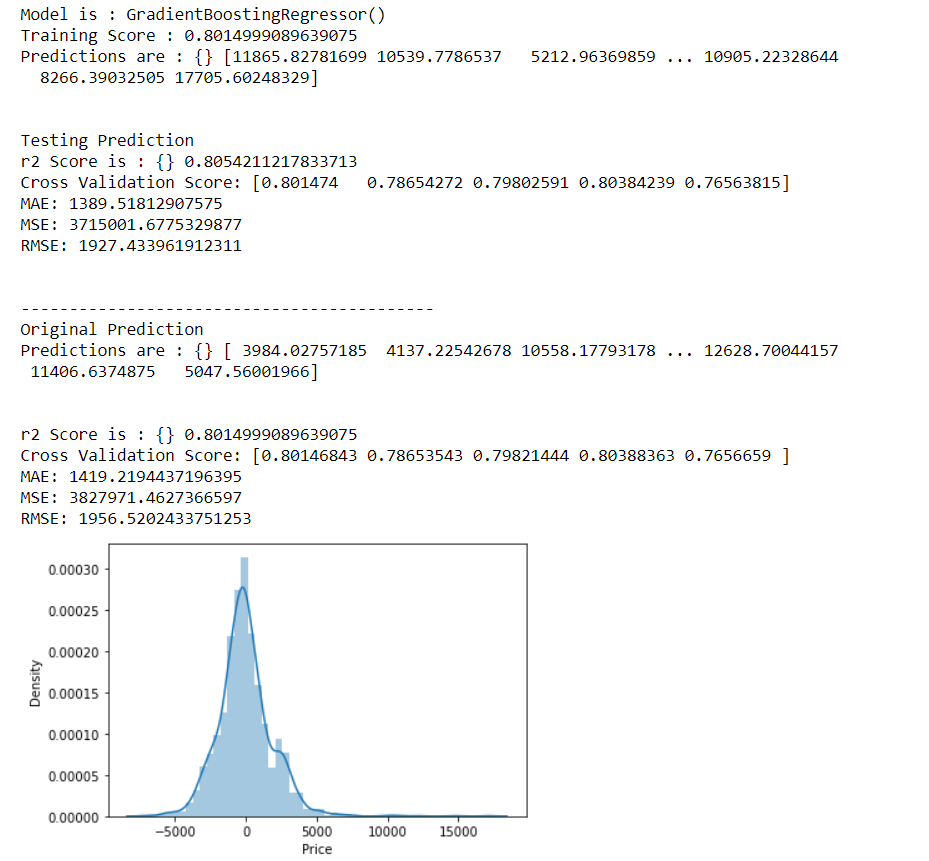
* K-Nearest Neighbors

predict(KNeighborsRegressor())



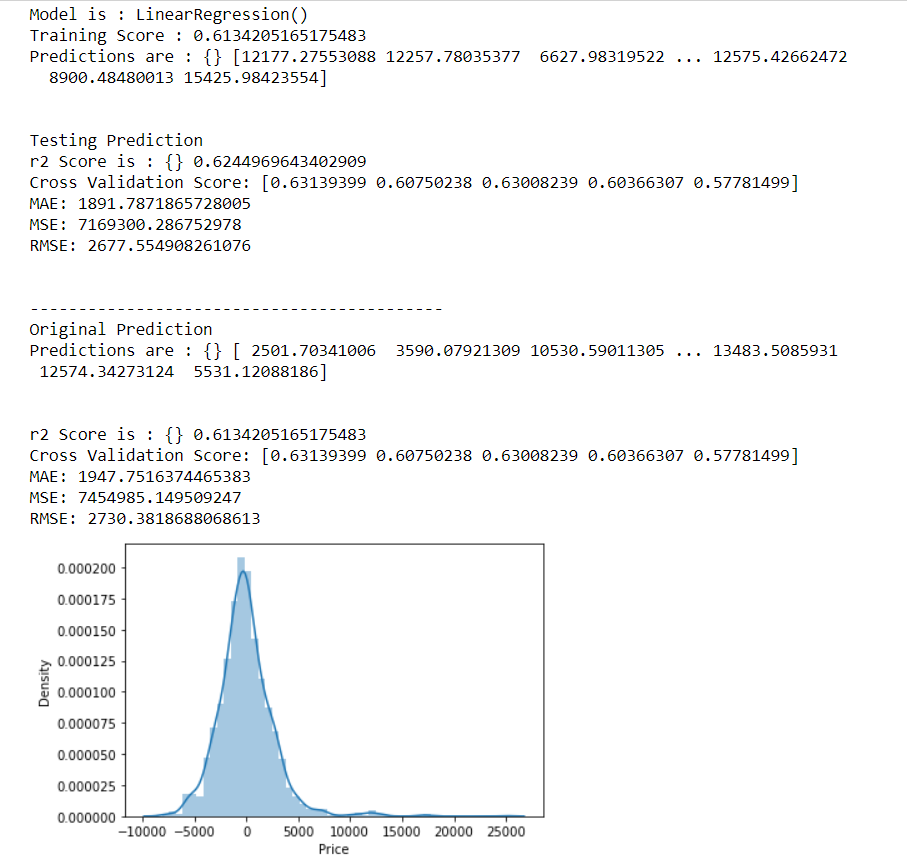
* GradientBoostingRegressor

predict(GradientBoostingRegressor())



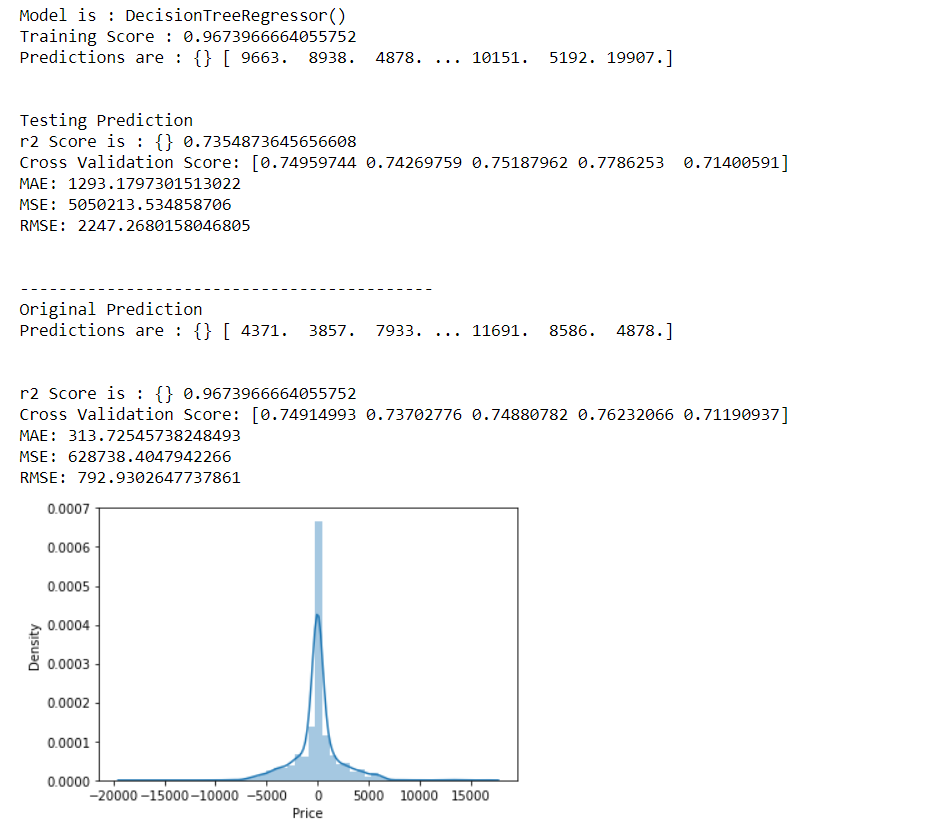
* LinearRegression

predict(LinearRegression())



* DecisionTreeRegressor

predict(DecisionTreeRegressor())



I have Perform with 5 Techniques to predict the Model and GradientBoostingRegressor performs best.

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

For More check the scikit Learn Documentaion .

[sklearn.ensemble.GradientBoostingRegressor — scikit-learn 0.24.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html)

Check how the model is fitted in best fit model

gdb = GradientBoostingRegressor()

gdb.fit(x\_train,y\_train)

pred\_test=gdb.predict(x\_test)

pred\_train=gdb.predict(x\_train)

plt.figure(figsize=(8,7))

plt.scatter(x=y\_test,y=pred\_test,color='r')

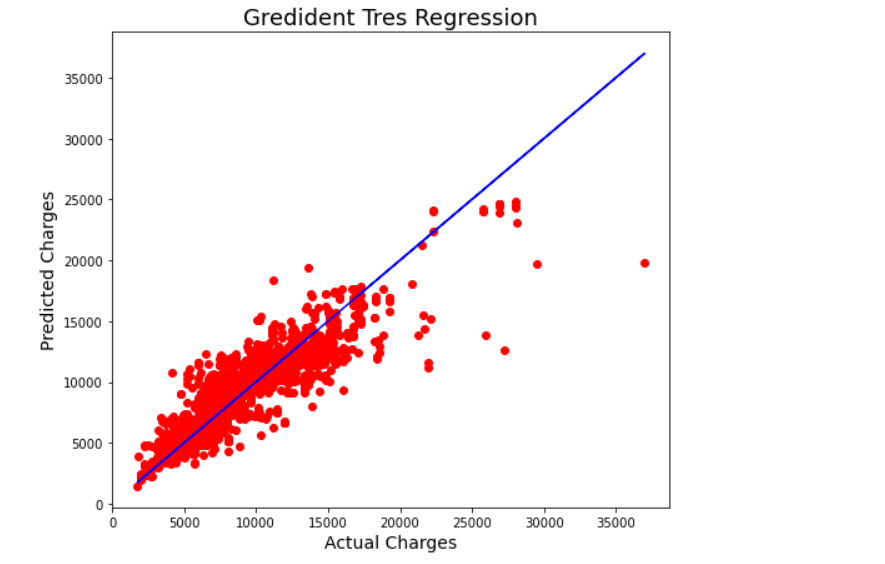
plt.plot(y\_test,y\_test,color='b')

plt.xlabel('Actual Charges',fontsize=14)

plt.ylabel('Predicted Charges',fontsize=14)

plt.title('Gredident Tres Regression',fontsize=18)

plt.show()



Finally Perform the Hyper tuning

Hyper Tunning Increase the Model Score. Perform GridSearchCv.

**from** **sklearn.model\_selection** **import** GridSearchCV

parameters = {

"n\_estimators":[5,50,250,500],

"max\_depth":[1,3,5,7,9],

"learning\_rate":[0.01,0.1,1,10,100]

}

GCV=GridSearchCV(GradientBoostingRegressor(),parameters,cv=5)

GCV.fit(x\_train,y\_train)

GCV.best\_params\_

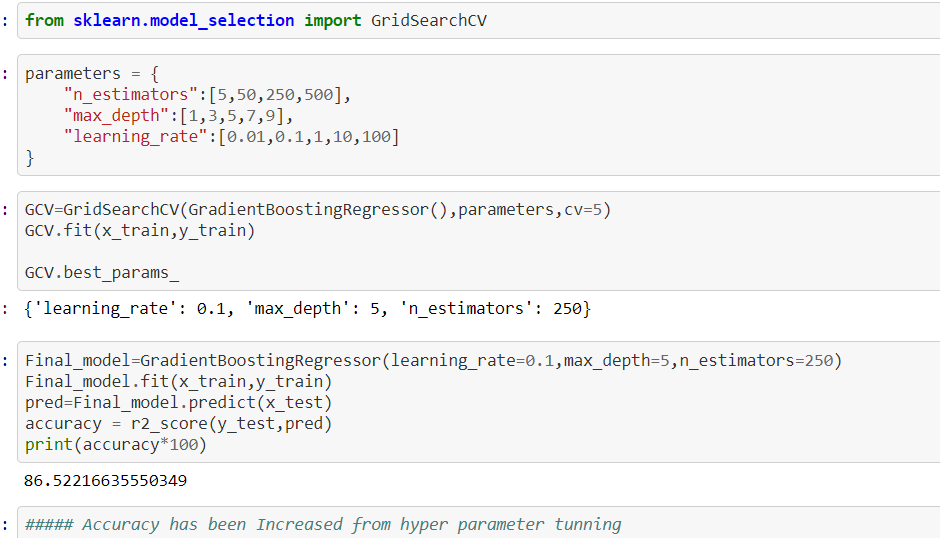
Final\_model=GradientBoostingRegressor(learning\_rate=0.1,max\_depth=5,n\_estimators=250)

Final\_model.fit(x\_train,y\_train)

pred=Final\_model.predict(x\_test)

accuracy = r2\_score(y\_test,pred)

print(accuracy\*100)



Here it shows that the Accuracy has been Increased and Now it is final score for the model.

**Conclusion**

We have Started with Data Exploratory Analysis where we see how data is Showing. We checked for Missing Value, Columns Present in Datasets, Statistical Information of Data sets.

Use Seaborn and Matplotlib for visualization of data. Visualization helps to know the dependent feature linked with independent Features.

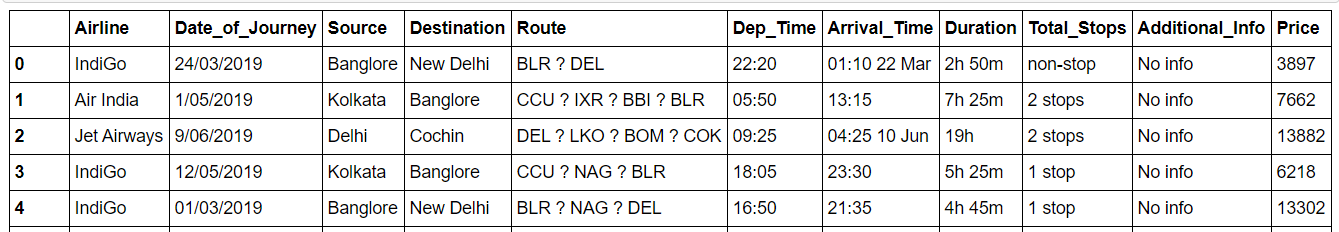
During the Data Pre-Processing Part we computed missing values, Some unwanted stuffs present in data like question mark, check for the importance of columns and remove the columns having no importance. We use encoding techniques.

Most of time takes Pre-processing techniques model is good trained when pre-processing is done well.

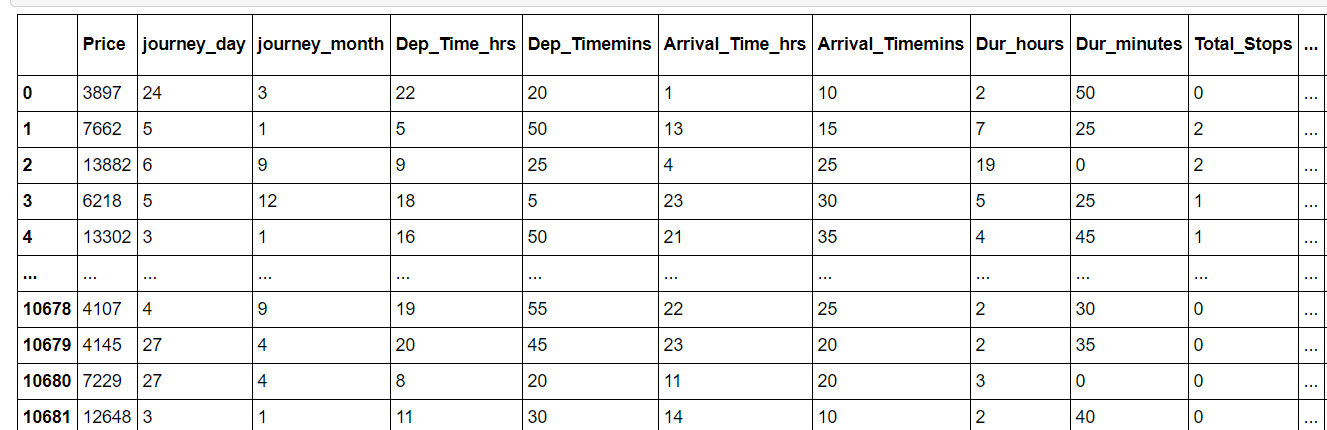
Finally, we go for modelling of Data we score the model and check for MAE, RMSE, MSE.

Check for best Model and Lastly we Tune the model for better accuracy.

Let’s have Look for Data which we started and Data on which model perform for training.



After



Still there is scope of Improvement by doing Expensive features. Looking for feature Engineering and Find the best Random state. Perform various model like ensemble and many more modelling. Practice and Learning improves the things and give some perfection.

Thanks, and Regards

Kumar Aman