AUTOMOBILE INSURANCE FRAUD DETECTION

# Introduction

Insurance fraud is any act committed to fraud the process of insurance. It occurs when a claimant attempts to obtain advantage they are not entitled to, or when an insurer knowingly rejects some benefit that is due.

In case of Automobile insurance fraud, fraud rings or groups can fake traffic deaths or stage collisions to make false insurance or exaggerated claims to collect insurance money. The ring may involve insurance claims adjusters and other people who create phony police reports in order to process claims.

## Problem Statement:

Business case: Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

In this project, you are provided a dataset which has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

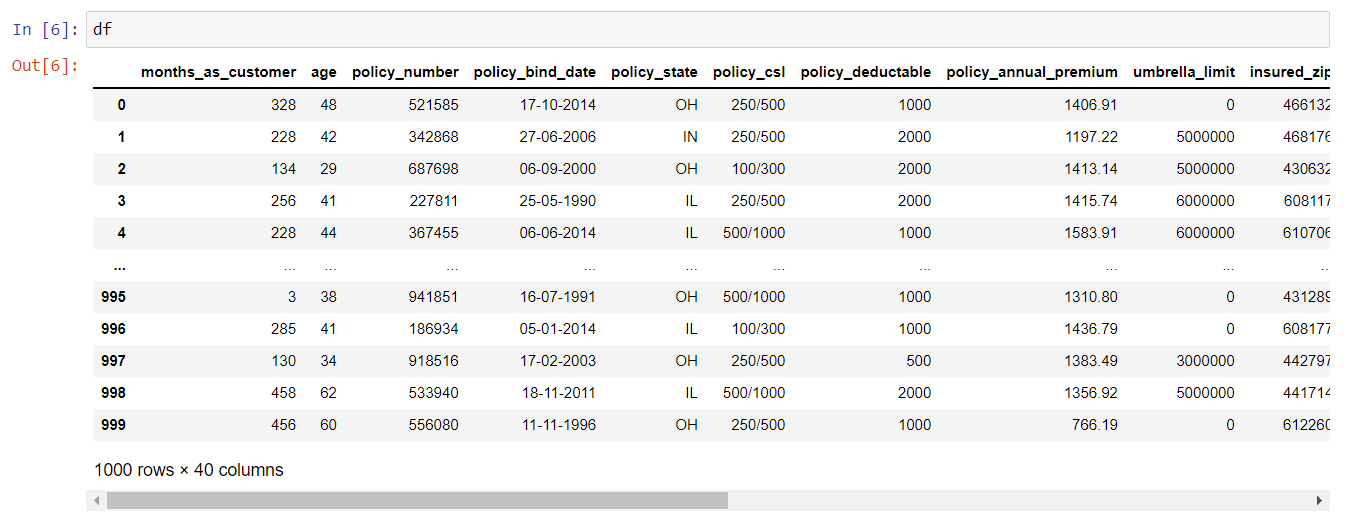
In this example, we will be working with some auto insurance data to demonstrate how you can create a predictive model that predicts if an insurance claim is fraudulent or not.

## Dataset Analysis:

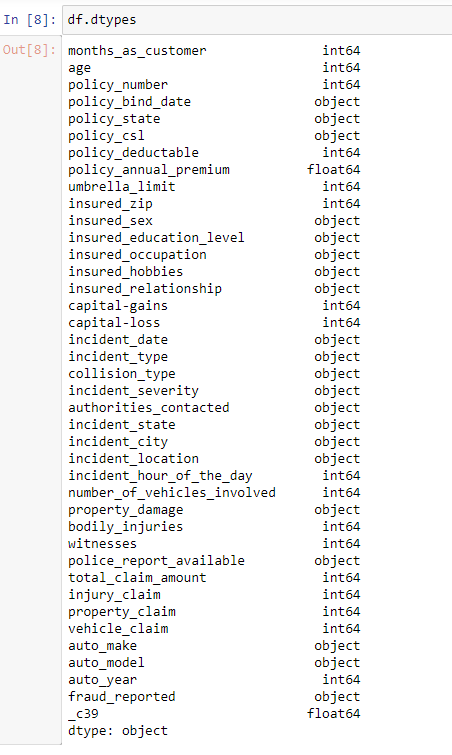
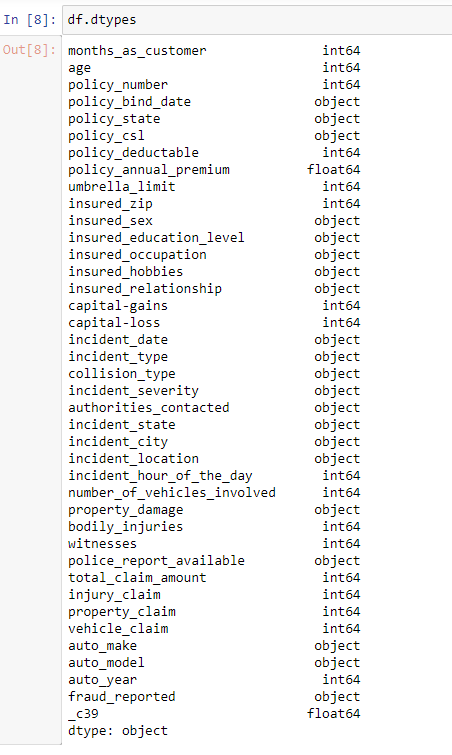
Analysis of the columns in the dataset and their significance.

|  |  |
| --- | --- |
| months\_as\_customer | Depicts the total number of months the customer has been associated with the company |
| age | Age of the policy holder |
| policy\_number | Policy number |
| policy\_bind\_date | The date when the policy was signed/bound |
| policy\_state | State (geographical) in the United States |
| policy\_csl | Policy Combined Single Limits, used to limit the components of a clain to a single dollar. |
| policy\_deductable | Deductible is the amount that a policy holder has to pay before the insurance company starts paying up |
| policy\_annual\_premium | The total amount of premium paid annually is called the annualized premium |
| umbrella\_limit | This insurance gives extra liability coverage over the stipulated limit of your car insurance. Once you exhaust the limit of your other insurance policies, umbrella insurance comes into play to provide you additional coverage. |
| insured\_zip | Zip code |
| insured\_sex | Gender of the policy holder |
| insured\_education\_level | Educational level of the policy holder |
| insured\_occupation | Occupation of the policy holder |
| insured\_hobbies | Hobbies of the policy holder |
| insured\_relationship | Relationship of the insured with the policy holder |
| capital-gains | Capital gain is the profit you make on sale of capital assets. |
| capital-loss | Capital loss is the loss you make on sale of capital assets. |
| incident\_date | Date on which the incident occurred |
| incident\_type | Type of the incident |
| collision\_type | In cases of accident, this explains the type of collision |
| incident\_severity | Explains the severity of the accident/ incident |
| authorities\_contacted | Concerns the authorities contacted upon the occurrence of the incident |
| incident\_state | State in which the incident occurred |
| incident\_city | City of the state in which the incident occurred |
| incident\_location | Exact location in the city |
| incident\_hour\_of\_the\_day | Hour in terms of 24 hr clock |
| number\_of\_vehicles\_involved | Explains the number of vehicles involved |
| property\_damage | Denotes whether any property damages occurred or not |
| bodily\_injuries | Denotes the number of bodily injuries |
| witnesses | Witnesses to the incident |
| police\_report\_available | Whether police report is available for the concerned incident |
| total\_claim\_amount | The total claim amount by the claimant |
| injury\_claim | Amount claimed for bodily injuries |
| property\_claim | Amount claimed for property damages |
| vehicle\_claim | Amount claimed for vehicular damages |
| auto\_make | Brand of the auto mobile involved |
| auto\_model | Model of the auto mobile |
| auto\_year | Year of manufacture of the auto mobile |
| fraud\_reported | Explains whether fraud was reported or not |
| \_c39 | Blank column |

The data set given to us consists of 40 columns and 1000 rows and hence is a huge data set.



The data types of the columns are studied.



Libraries used:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sklearn

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix, classification\_report

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

from scipy.stats import zscore #to remove outliers

from scipy.stats import skew

import requests

import pandas\_profiling

import io

import warnings

warnings.filterwarnings('ignore')

Importing the data set:

df = pd.read\_csv("Automobile\_insurance\_fraud.csv")

Functions used:

df.shape – used to check the size of the data set

df.dtypes – used to check the data types of the columns in the dataset

df.info() - The info() function is used to print a concise summary of a DataFrame. This method prints information about a DataFrame including the index dtype, column dtypes, non-null values and memory usage.

df.nunique() – used to find the unique values in each column

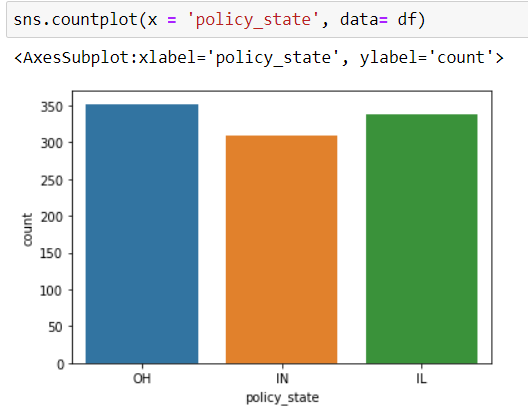
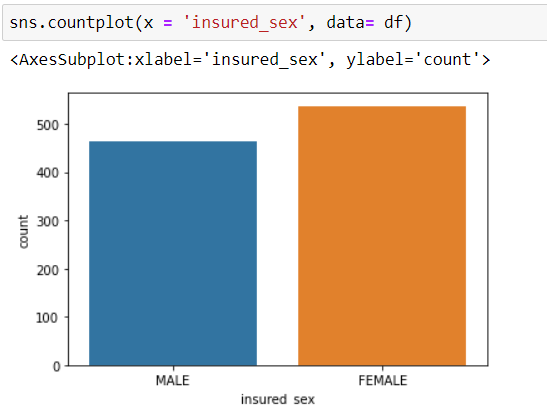
Conclusions from the above functions:

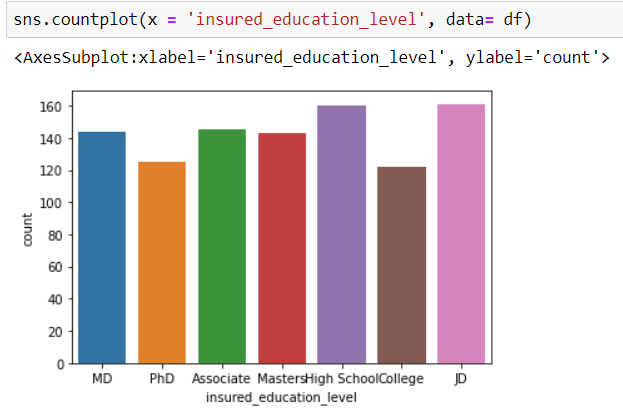
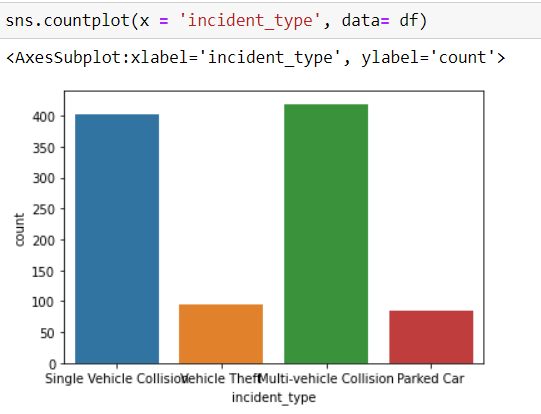
* After using the df.nunique() function, we find that the column policy\_number has 1000 unique values which mean that the entire column has unique values and hence cannot be used for model building.
* The column \_c39 is an empty column and has no relevance to the data.
* Hence both the coulmns, policy\_number and \_c39 are deleted from the data set using

df.drop(columns=["\_c39", "policy\_number"], axis=1,inplace=True)

Exploratory Data Analysis:

1. To find the object type columns in the data set, df.columns.astype(object), is used, so as to help us create graphs for analysis.
2. Countplots are created to identify:





Conclusions from graphs:

* There are 3 unique states and state of Indiana has least number of claimants
* There are more female claimants than male.
* There are 4 unique claim types and majority of the claims are due to Collison accident type, while vehicle theft and parked car are less.
* There are 7 unique education level among the people and all are almost equally distributed.

1. **Now we will separate the categorical columns from the entire data set, so as to convert those values into integer using Label Encoding**.

from sklearn.preprocessing import LabelEncoder

# creating instance of labelencoder

labelencoder = LabelEncoder()

# Assigning numerical values and storing in another column

df1 = df1.apply(LabelEncoder().fit\_transform)

1. **Handling null values:**

Once Label encoding is done, we need to check for null values.

df.isna().sum(), is used to check the presence of null values in each column.

Null value presence is checked using a heatmap by,

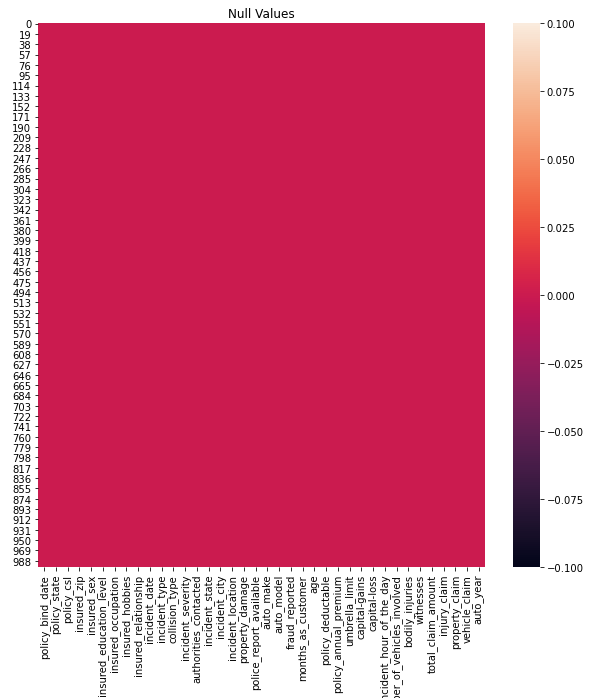
plt.figure(figsize = (10,10))

sns.heatmap(df.isnull())

plt.title("Null Values")

plt.show()

There are no null values in the dataset, hence null value handling is not required for this dataset.



1. **Skewness of the data is checked using KDE plot**

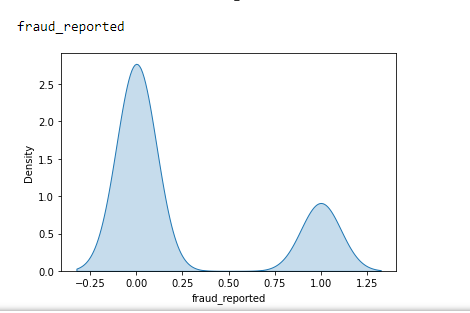
for col in df:

print(col)

plt.figure()

sns.kdeplot(df[col], shade = True)

plt.show()



The plot shows that the data is imbalanced and hence skewed.

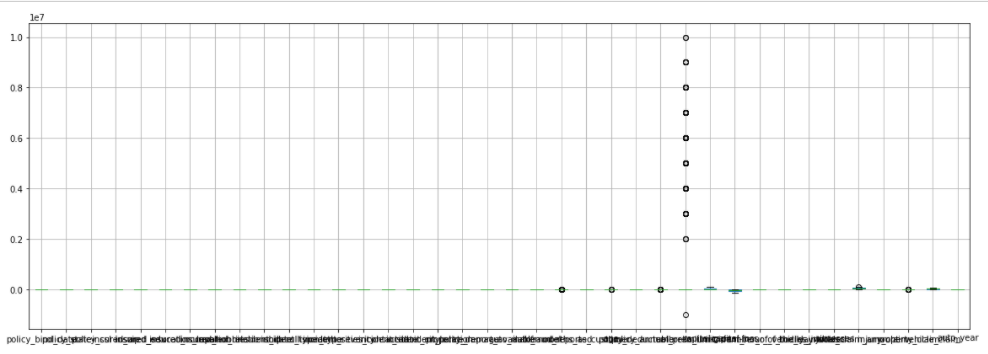
Additionally, Pandas profiling can be used to do EDA, Pandas profiling is an open source Python module with which we can quickly do an exploratory data analysis with just a few lines of code. Besides, it also generates interactive reports in web format that can be presented to any person, even if they don’t know programming.

pre\_profile = df.profile\_report(title="Automobile\_insurance\_fraud")

1. **Outlier check**

df.boxplot(figsize=[20,8])

plt.subplots\_adjust(bottom=0.25)

plt.show()

Since outliers are found, they need to be treated.

Outlier are analyzed using Z score method.

A z-score describes the position of a raw score in terms of its distance from the mean, when measured in standard deviation units. The z-score is positive if the value lies above the mean, and negative if it lies below the mean.

from scipy.stats import zscore

z = np.abs(zscore(df))

new\_df = df[(z<3).all(axis=1)]

The data loss is found out to be 2.0%.

Since it is less than 7%, the outliers are deleted.

1. **Correlation check**

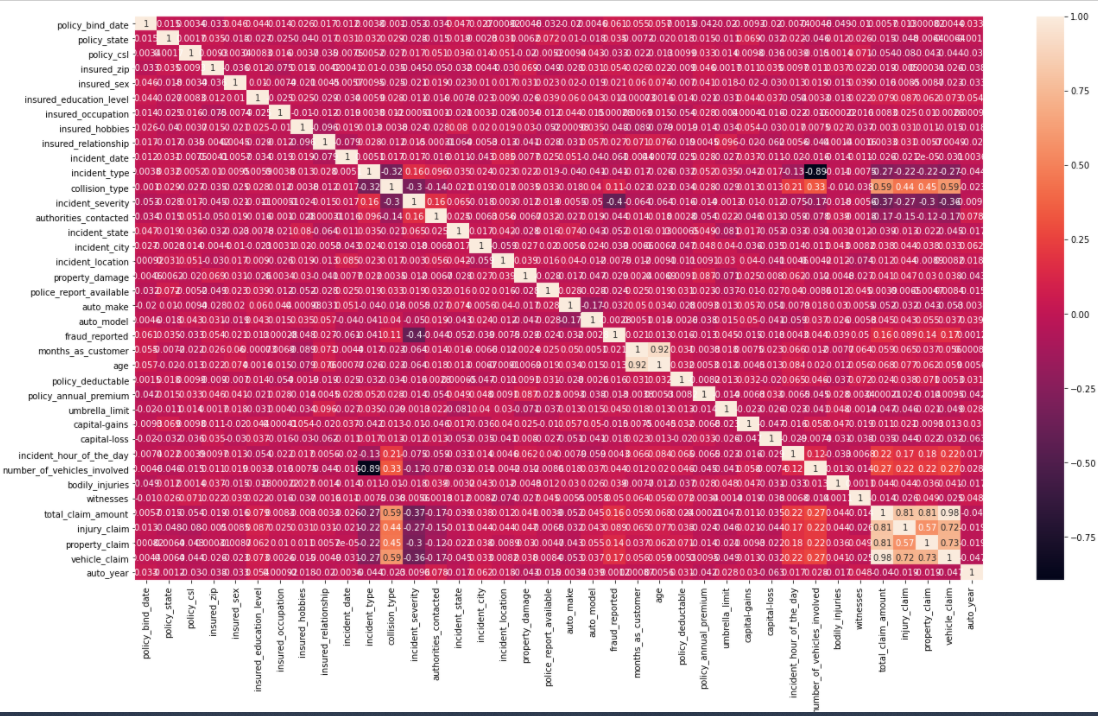
Correlation testing is done to check the dependencies between columns in a data set.

plt.figure(figsize=[22,12])

cor = df.corr()

sns.heatmap(cor, annot = True)

plt.show()

Correlation is plotted with a heatmap and subsequently sorted with respect to the target variable to understand the impact of various columns on the target variable

Target variable is separated as Y and the remaining columns are kept as a dataframe x .

Skewness is checked and it is seen that a few columns such as total claim amount, umbrella limit and vehicle claim (0.5%)

Power transform is applied to remove skewness from the dataset.

x = df.drop('fraud\_reported',axis=1)

y = df['fraud\_reported']

from sklearn.preprocessing import power\_transform

df\_new = power\_transform(x)

df\_new = pd.DataFrame(df\_new, columns = x.columns)

x.agg(['skew', 'kurtosis']).transpose()

1. **Finding best random state**

We use the following to find the best random state.

from sklearn.linear\_model import LogisticRegression

maxAccu=0

maxRS=0

for i in range(1,200):

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=.30, random\_state = i)

LR = LogisticRegression()

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

predLR = LR.predict(x\_test)

acc = accuracy\_score(y\_test, predLR)

if acc>maxAccu:

maxAccu = acc

maxRS=i

print("Best accuracy is", maxAccu," on Random State ",maxRS)

86% efficiency with GridSearchCv applied on decision tree classifier.

The best random state is found out to be 63.

1. **Train, test, split**

We have found the best random state. We will create our train\_test\_split using this random state.

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=63)

1. **Handling data imbalance**

Data imbalance is checked with countplot on the target variable and it is seen that the ratio is less than 50% for one of the categories in the target variable.

sns.countplot(df['fraud\_reported'])

y.value\_counts()

pip install imbalanced\_learn

pip install delayed

Using the library imbalanced learn we use the SMOTE technique to balance the data set.

from imblearn.over\_sampling import SMOTE

oversample=SMOTE(k\_neighbors=4)

#transform the dataset

x,y=oversample.fit\_resample(x,y)

1. **Classification**

Now the classification is carried out using:

* **Logistic regression**

from sklearn.linear\_model import LogisticRegression

LR = LogisticRegression()

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

predlr = LR.predict(x\_test)

print(accuracy\_score(y\_test,predlr))

print(confusion\_matrix(y\_test,predlr))

print(classification\_report(y\_test,predlr))

lr\_acc = accuracy\_score(y\_test,predlr) \*100

lr\_acc

Accuracy= 83.33333333333334

* **Support vector machines**

from sklearn.svm import SVC

svc = SVC()

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

predsvc = svc.predict(x\_test)

print(accuracy\_score(y\_test,predsvc))

print(confusion\_matrix(y\_test,predsvc))

print(classification\_report(y\_test,predsvc))

svc\_acc = accuracy\_score(y\_test,predsvc) \*100

svc\_acc

Accuracy= 79.25170068027211

* **Decision Tree**

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

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

preddt = dt.predict(x\_test)

print(accuracy\_score(y\_test,preddt))

print(classification\_report(y\_test,preddt))

print(confusion\_matrix(y\_test,preddt))

dt\_acc = accuracy\_score(y\_test,preddt)\*100

dt\_acc

Accuracy= 81.29251700680273

* **Random forest**

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n\_estimators=1000)

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

predrf = rf.predict(x\_test)

print(accuracy\_score(y\_test,predrf))

print(classification\_report(y\_test,predrf))

print(confusion\_matrix(y\_test,predrf))

rf\_acc = accuracy\_score(y\_test,predrf)\*100

rf\_acc

Accuracy= 82.99319727891157

1. **Checking cross validation scores**

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

scr = cross\_val\_score(LR, x, y, cv=5)

print("CrossValidation Score of LogisticRegression Model: ", scr.mean())

lr\_cv = scr.mean() \*100

lr\_cv

Cross Validation Score of Logistic Regression Model: 74.12162162162163

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

scr = cross\_val\_score(svc, x, y, cv=5)

print("CrossValidation Score of SVC Model: ", scr.mean())

svc\_cv = scr.mean() \*100

svc\_cv

Cross Validation Score of SVC Model: 87.29729729729729

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

scr = cross\_val\_score(dt, x, y, cv=5)

print("CrossValidation Score of DecisionTree Model: ", scr.mean())

dt\_cv = scr.mean() \*100

dt\_cv

Cross Validation Score of Decision Tree Model: 84.93243243243244

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

scr = cross\_val\_score(rf, x, y, cv=5)

print("CrossValidation Score of RandomForest Model: ", scr.mean())

rf\_cv = scr.mean() \*100

rf\_cv

Cross Validation Score of Random Forest Model: 88.2432432432432

Model with least difference between Model accuracy and cross validation is selected as the best model.

#Linear Regression -> 9.21171171171171

lr\_acc - lr\_cv

# SVM -> -8.045596617025183

svc\_acc - svc\_cv

# Decision Tree -> -3.6399154256297095

dt\_acc - dt\_cv

# Random Forest -> -5.250045964331676

rf\_acc - rf\_cv

Hence, we find that Decision Tree model has least difference and is the best model.

1. **Hyper parameter tuning**

### Manual Hyperparameter Tuning

model=RandomForestClassifier(n\_estimators=300,criterion='entropy',max\_features='sqrt',min\_samples\_leaf=10,random\_state=100).fit(x\_train,y\_train)

predictions=model.predict(x\_test)

print(confusion\_matrix(y\_test,predictions))

print(accuracy\_score(y\_test,predictions))

print(classification\_report(y\_test,predictions))

import numpy as np

from sklearn.model\_selection import RandomizedSearchCV

# Number of trees in random forest

n\_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]

# Number of features to consider at every split

max\_features = ['auto', 'sqrt','log2']

# Maximum number of levels in tree

max\_depth = [int(x) for x in np.linspace(10, 1000,10)]

# Minimum number of samples required to split a node

min\_samples\_split = [2, 5, 10,14]

# Minimum number of samples required at each leaf node

min\_samples\_leaf = [1, 2, 4,6,8]

# Create the random grid

random\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'criterion':['entropy','gini']}

print(random\_grid)

rf=RandomForestClassifier()

rf\_randomcv=RandomizedSearchCV(estimator=rf,param\_distributions=random\_grid,n\_iter=100,cv=3,verbose=2,

random\_state=100,n\_jobs=-1)

### fit the randomized model

rf\_randomcv.fit(x\_train,y\_train)

rf\_randomcv.best\_params\_

rf\_randomcv

best\_random\_grid=rf\_randomcv.best\_estimator\_

from sklearn.metrics import accuracy\_score

y\_pred=best\_random\_grid.predict(x\_test)

print(confusion\_matrix(y\_test,y\_pred))

print("Accuracy Score {}".format(accuracy\_score(y\_test,y\_pred)))

print("Classification report: {}".format(classification\_report(y\_test,y\_pred)))

1. **Grid Search CV on Random Forest**

rf\_randomcv.best\_params\_

from sklearn.model\_selection import GridSearchCV

param\_grid = {

'criterion': [rf\_randomcv.best\_params\_['criterion']],

'max\_depth': [rf\_randomcv.best\_params\_['max\_depth']],

'max\_features': [rf\_randomcv.best\_params\_['max\_features']],

'min\_samples\_leaf': [rf\_randomcv.best\_params\_['min\_samples\_leaf'],

rf\_randomcv.best\_params\_['min\_samples\_leaf']+2,

rf\_randomcv.best\_params\_['min\_samples\_leaf'] + 4],

'min\_samples\_split': [rf\_randomcv.best\_params\_['min\_samples\_split'] - 2,

rf\_randomcv.best\_params\_['min\_samples\_split'] - 1,

rf\_randomcv.best\_params\_['min\_samples\_split'],

rf\_randomcv.best\_params\_['min\_samples\_split'] +1,

rf\_randomcv.best\_params\_['min\_samples\_split'] + 2],

'n\_estimators': [rf\_randomcv.best\_params\_['n\_estimators'] - 200, rf\_randomcv.best\_params\_['n\_estimators'] - 100,

rf\_randomcv.best\_params\_['n\_estimators'],

rf\_randomcv.best\_params\_['n\_estimators'] + 100, rf\_randomcv.best\_params\_['n\_estimators'] + 200]

}

print(param\_grid)

#### Fit the grid\_search to the data

rf=RandomForestClassifier()

grid\_search=GridSearchCV(estimator=rf,param\_grid=param\_grid,cv=10,n\_jobs=-1,verbose=2)

grid\_search.fit(x\_train,y\_train)

grid\_search.best\_estimator\_

best\_grid=grid\_search.best\_estimator\_

y\_pred=best\_grid.predict(x\_test)

print(confusion\_matrix(y\_test,y\_pred))

print("Accuracy Score {}".format(accuracy\_score(y\_test,y\_pred)))

print("Classification report: {}".format(classification\_report(y\_test,y\_pred)))

1. **Grid Search CV on Decision Tree Algorithm**

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV

params = {'max\_leaf\_nodes': list(range(2, 100)), 'min\_samples\_split': [2, 3, 4]}

grid\_search\_cv = GridSearchCV(DecisionTreeClassifier(random\_state=63), params, verbose=1, cv=3)

grid\_search\_cv.fit(x\_train, y\_train)

1. **Saving the model**

import joblib

joblib.dump(best\_grid, "model.pkl")

prediction = model.predict(x\_test)

**Conclusion:**

86% efficiency is achieved with Grid Search CV applied on decision tree classifier.

FLIGHT PRICE PREDICTION

# PROBLEM STATEMENT

Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable. Here you will be provided with prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities.

# DATASET ANALYSIS

Size of training set: 10683 records

Size of test set: 2671 records

|  |  |
| --- | --- |
| **Airline** | Contains the airline name |
| **Date\_of\_Journey** | Contains the date of journey |
| **Source** | Contains the source/ city from which a person is taking the flight |
| **Destination** | Contains the destination/ city on to which the passenger is flying to |
| **Route** | Contains the route of the flight through the flight's flight |
| **Dep\_Time** | Contains the departure time from source |
| **Arrival\_Time** | Contains the arrival time at destination |
| **Duration** | Contains the duration of the flight |
| **Total\_Stops** | Contains the total stops midway through the flight's journey |
| **Additional\_Info** | Contains additional info like whether the baggage has been checked in or food is available in the flight or not |
| **Price** | Contains the price of the flight journey |

# PROCESS:

1. **Importing libraries**

We will be importing libraries first.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import sklearn

from sklearn import metrics

from sklearn.metrics import accuracy\_score

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error,explained\_variance\_score

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

from scipy.stats import zscore #to remove outliers

from scipy.stats import skew

import requests

import pandas\_profiling

import io

import warnings

warnings.filterwarnings('ignore')

1. **Importing the data set for training**

df = pd.read\_excel("Data\_Train.xlsx")

1. **Finding the dataset properties**

df.shape gives us the shape of the data frame which is (10683, 11)

df.columns gives us the names of the columns which are

Index(['Airline', 'Date\_of\_Journey', 'Source', 'Destination', 'Route', 'Dep\_Time', 'Arrival\_Time', 'Duration', 'Total\_Stops', 'Additional\_Info', 'Price'],dtype='object')

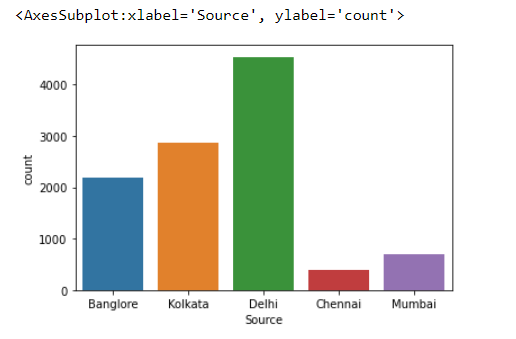
df.dtypes gives us the data types of the columns

df.info() gives us more information like the non null count, data types and memory usage

1. **Exploratory Data Analysis**

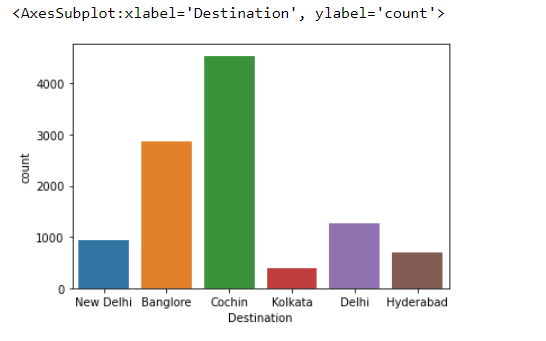
Now we use countplot to check the graphical representation of important columns

* sns.countplot(df['Source'])



Shows us that maximum number of flights start from Delhi and minimum start from Chennai

* sns.countplot(df['Destination'])



Shows us that maximum number of flight’s destination is Cochin and minimum is Kolkata.

1. **Handling categorical data in the train data**

Next we need to convert our categorical columns into numerical type.

Categorical columns in the dataset are as follows:

#'Airline', 'Date\_of\_Journey', 'Source', 'Destination', 'Route',

#'Dep\_Time', 'Arrival\_Time', 'Duration', 'Total\_Stops',

#'Additional\_Info'

from sklearn.preprocessing import LabelEncoder

LE=LabelEncoder()

df["Airline"] = LE.fit\_transform(df["Airline"])

df[["day","month","year"]] = df["Date\_of\_Journey"].str.split("/", expand = True)

df.drop(columns=["Date\_of\_Journey", "Route", "Additional\_Info"], axis = 1, inplace= True)

df["year"].nunique()

#since all values are for 2019, year can be dropped

df.drop(columns=["year"], axis = 1, inplace= True)

df["Source"] = LE.fit\_transform(df["Source"])

df["Destination"] = LE.fit\_transform(df["Destination"])

df["Duration"] = df["Duration"].astype(str)

df["Total\_Stops"] = df["Total\_Stops"].astype(str)

for i in range(0,len(df)):

df["Duration"][i] = df["Duration"][i][0]

df["Total\_Stops"][i] = df["Total\_Stops"][i][0]

for i in range(0,len(df)):

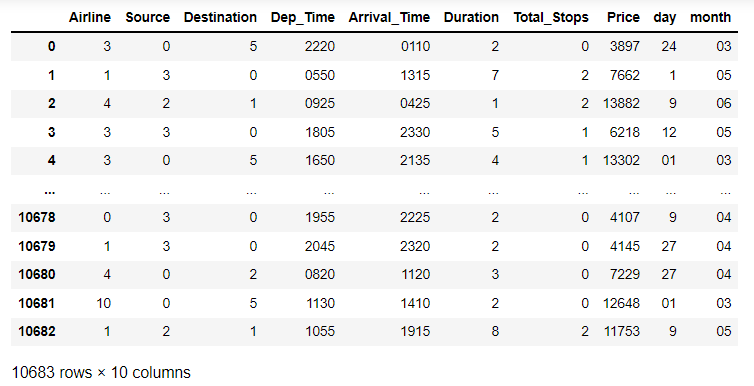
df["Total\_Stops"][i] = df["Total\_Stops"][i].replace("n", "0")

for i in range(0,len(df)):

df["Arrival\_Time"][i] = df["Arrival\_Time"][i][0:5]

df["Arrival\_Time"][i] = df["Arrival\_Time"][i].replace(":","")

df["Dep\_Time"][i] = df["Dep\_Time"][i].replace(":","")

We have now successfully converted all the categorical columns into numerical using Label encoder.

Converting the datatypes into integer is done by

df = df.astype(int)

df.dtypes

Airline int32

Source int32

Destination int32

Dep\_Time int32

Arrival\_Time int32

Duration int32

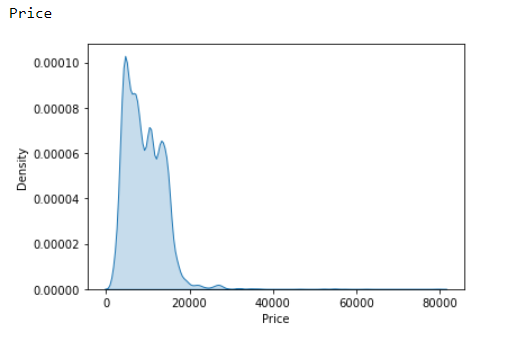
Total\_Stops int32

Price int32

day int32

month int32

1. **Checking the skewness in the train data**



Shows us that the prices is skewed.

pre\_profile = df.profile\_report(title="flight price")

HBox(children=(HTML(value='Summarize dataset'), FloatProgress(value=0.0, max=23.0), HTML(value='')))

HBox(children=(HTML(value='Generate report structure'), FloatProgress(value=0.0, max=1.0), HTML(value='')))

HBox(children=(HTML(value='Render HTML'), FloatProgress(value=0.0, max=1.0), HTML(value='')))

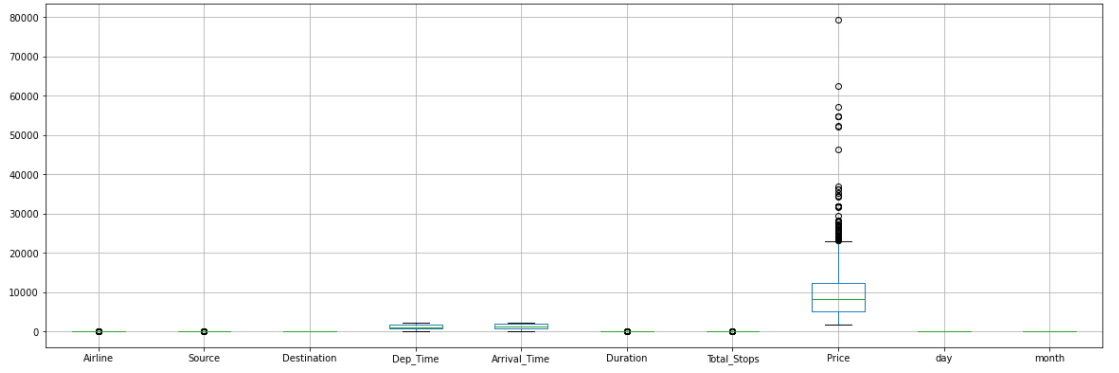
1. **Checking for outliers in the train data**

We need to remove the outliers before removing the skewness and the train, test, split.

df.boxplot(figsize=[20,8])

plt.subplots\_adjust(bottom=0.25)

plt.show()



Checking the Z score value

from scipy.stats import zscore

z = np.abs(zscore(df))

new\_df = df[(z<3).all(axis=1)]

We find the data loss to be 1.392 and since it is less than 7%, we can drop the outliers.

1. **Checking the correlation**

plt.figure(figsize=[22,12])

cor = df.corr()

sns.heatmap(cor, annot = True)

plt.show()

cor["Price"].sort\_values(ascending=False)

Price 1.000000

Total\_Stops 0.670373

Duration 0.126816

Source 0.075584

Arrival\_Time 0.033870

Dep\_Time 0.008006

Airline -0.043091

month -0.048553

day -0.117294

Destination -0.174427

Name: Price, dtype: float64

df.columns

Index(['Airline', 'Source', 'Destination', 'Dep\_Time', 'Arrival\_Time',

'Duration', 'Total\_Stops', 'Price', 'day', 'month'],

dtype='object')

columns = ['Airline', 'Source', 'Destination', 'Dep\_Time', 'Arrival\_Time',

'Duration', 'Total\_Stops', 'Price', 'day', 'month']

sns.pairplot(df[columns])

1. **Checking for Skewness in the train set**

x = df.drop('Price',axis=1)

y = df['Price']

for col in df:

print(col)

print(skew(df[col]))

plt.figure()

sns.distplot(df[col])

plt.show()

x.skew() # check skewness

We see that,

Airline 0.732092

Source -0.437111

Destination 1.266304

Dep\_Time 0.115228

Arrival\_Time -0.370819

Duration 1.490828

Total\_Stops 0.227272

day 0.109448

month -0.409026

dtype: float64

For removing skewness by using power transform,

from sklearn.preprocessing import power\_transform

df\_new = power\_transform(x)

df\_new = pd.DataFrame(df\_new, columns = x.columns)

df\_new.skew()

Airline -0.014454

Source -0.235084

Destination 0.040804

Dep\_Time -0.113476

Arrival\_Time -0.440023

Duration 0.216489

Total\_Stops -0.071360

day -0.203104

month -0.221736

dtype: float64

1. **Finding the best random state**

from sklearn.metrics import r2\_score

from sklearn.linear\_model import LinearRegression

maxAccu=0

maxRS=0

for i in range(1,200):

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=.30, random\_state = i)

LR = LinearRegression()

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

predLR = LR.predict(x\_test)

acc = r2\_score(y\_test, predLR)

if acc > maxAccu:

maxAccu = acc

maxRS=i

print("Best accuracy is", maxAccu," on Random State ",maxRS)

We see that,

Best accuracy is 0.53582779050782 is achieved on Random State 59

1. **Train, test and split**

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=59)

1. **Linear Regression Model**

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

predLR = LR.predict(x\_test)

acc = r2\_score(y\_test, predLR)

acc

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

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

plt.xlabel('Y Test')

plt.ylabel('Predicted Y')

print('MAE:', metrics.mean\_absolute\_error(y\_test, predLR))

print('MSE:', metrics.mean\_squared\_error(y\_test, predLR))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, predLR)))

print('Variance:', metrics.explained\_variance\_score(y\_test, predLR))

print('R2 Score:', r2\_score(y\_test, predLR))

Output:

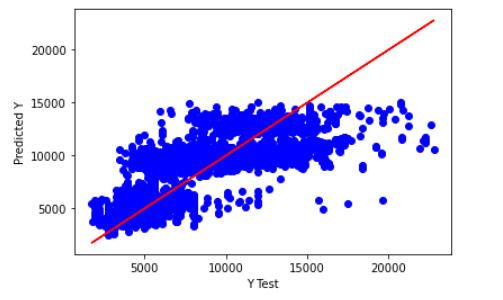
MAE: 2194.552032585386

MSE: 7725269.786149219

RMSE: 2779.43695487939

Variance: 0.5358367803808664

R2 Score: 0.53582779050782



1. **Cross validation of the model**

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

for j in range(2,10):

cv\_score = cross\_val\_score(LR, x, y, cv = j)

cv\_mean = cv\_score.mean()

print(f"At cross fold {j} the cv score is {cv\_mean} and accuracy score for training is {acc}")

print("\n")

Output:

At cross fold 2 the cv score is 0.5107967858926861 and accuracy score for training is 0.53582779050782

At cross fold 3 the cv score is 0.5111576025397495 and accuracy score for training is 0.53582779050782

At cross fold 4 the cv score is 0.5114362959023608 and accuracy score for training is 0.53582779050782

At cross fold 5 the cv score is 0.5112826466848748 and accuracy score for training is 0.53582779050782

At cross fold 6 the cv score is 0.511009954024671 and accuracy score for training is 0.53582779050782

At cross fold 7 the cv score is 0.510997253701374 and accuracy score for training is 0.53582779050782

At cross fold 8 the cv score is 0.5113626209578535 and accuracy score for training is 0.53582779050782

At cross fold 9 the cv score is 0.5114904048778174 and accuracy score for training is 0.5358277905078

Since number of folds don't have much impact on the accuracy and cv\_score, cv = 5 is selected.

1. **Regularization**

from sklearn.model\_selection import GridSearchCV

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

from sklearn.linear\_model import Lasso

parameters = {'alpha': [.0001, .001, .01, 1, 10], 'random\_state': list(range(0,10))}

ls = Lasso()

clf = GridSearchCV(ls, parameters)

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

print(clf.best\_params\_)

{'alpha': 1, 'random\_state': 0}

ls = Lasso(alpha=1, random\_state= 0)

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

ls.score(x\_train, y\_train)

pred\_ls = ls.predict(x\_test)

lss = r2\_score(y\_test, pred\_ls)

lss

0.5358281427003027

* **Cat Boost Regressor**

from catboost import CatBoostRegressor

# Initialize CatBoostRegressor

model = CatBoostRegressor(iterations=10,learning\_rate=0.5,depth=2)

# Fit model

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

# Get predictions

preds = model.predict(x\_test)

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

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

plt.xlabel('Y Test')

plt.ylabel('Predicted Y')

print('MAE:', metrics.mean\_absolute\_error(y\_test, preds))

print('MSE:', metrics.mean\_squared\_error(y\_test, preds))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, preds)))

print('R2 Score', r2\_score(y\_test, preds))

print('Variance:',metrics.explained\_variance\_score(y\_test, preds))

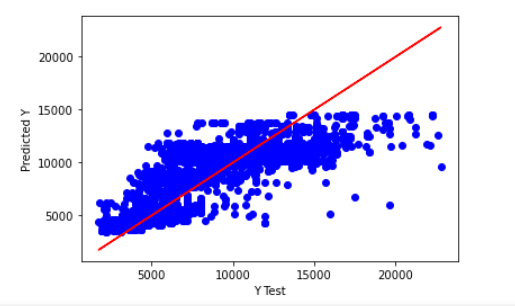
MAE: 1808.7806403428372

MSE: 5573756.674025824

RMSE: 2360.8804870272074

R2 Score 0.665101281615697

Variance: 0.6651020384937714



* **Decision Tree Regressor**

from sklearn.tree import DecisionTreeRegressor

import matplotlib.pyplot as plt

# Fit regression model

regr\_1 = DecisionTreeRegressor(max\_depth=5)

regr\_1.fit(x\_train,y\_train)

# Predict

preds = regr\_1.predict(x\_test)

# Plot the results

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

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

plt.xlabel('Y Test')

plt.ylabel('Predicted Y')

print('MAE:', metrics.mean\_absolute\_error(y\_test, preds))

print('MSE:', metrics.mean\_squared\_error(y\_test, preds))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, preds)))

print('Variance:',metrics.explained\_variance\_score(y\_test, preds))

print('R2 Score:',metrics.r2\_score(y\_test, preds))

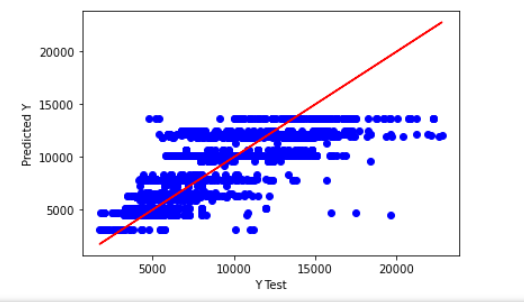
MAE: 1708.4866492921017

MSE: 5187955.870919908

RMSE: 2277.708469255868

Variance: 0.6882822884114064

R2 Score: 0.6882820916273555



1. **Hyper parameter tuning**

from sklearn.model\_selection import GridSearchCV

model = DecisionTreeRegressor()

gs = GridSearchCV(model,

param\_grid = {'max\_depth': range(1, 11),

'min\_samples\_split': range(10, 60, 10)},

cv=5,

n\_jobs=1,

scoring='neg\_mean\_squared\_error')

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

print(gs.best\_params\_)

print(-gs.best\_score\_)

{'max\_depth': 10, 'min\_samples\_split': 10}

3584896.852771991

new\_model = DecisionTreeRegressor(max\_depth=10, \_samples\_split=20)

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

preds = new\_model.predict(x\_test)

# Plot the results

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

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

plt.xlabel('Y Test')

plt.ylabel('Predicted Y')

print('MAE:', metrics.mean\_absolute\_error(y\_test, preds))

print('MSE:', metrics.mean\_squared\_error(y\_test, preds))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, preds)))

print('Variance:',metrics.explained\_variance\_score(y\_test, preds))

print('R2 Score:',metrics.r2\_score(y\_test, preds))

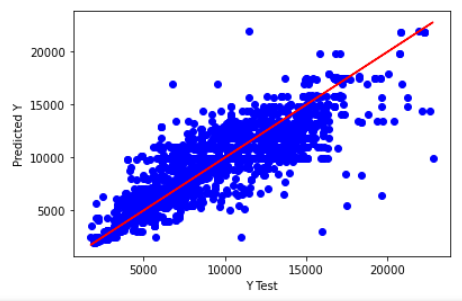
MAE: 1338.4353162219593

MSE: 3591487.8387284637

RMSE: 1895.1221171018146

Variance: 0.7842061436158307

R2 Score: 0.7842057440562394



1. **Handling categorical data in the test data**

from sklearn.preprocessing import LabelEncoder

LE=LabelEncoder()

x\_test["Airline"] = LE.fit\_transform(x\_test["Airline"])

x\_test[["day","month","year"]] = x\_test["Date\_of\_Journey"].str.split("/", expand = True)

x\_test.drop(columns=["Date\_of\_Journey", "Route", "Additional\_Info"], axis = 1, inplace= True)

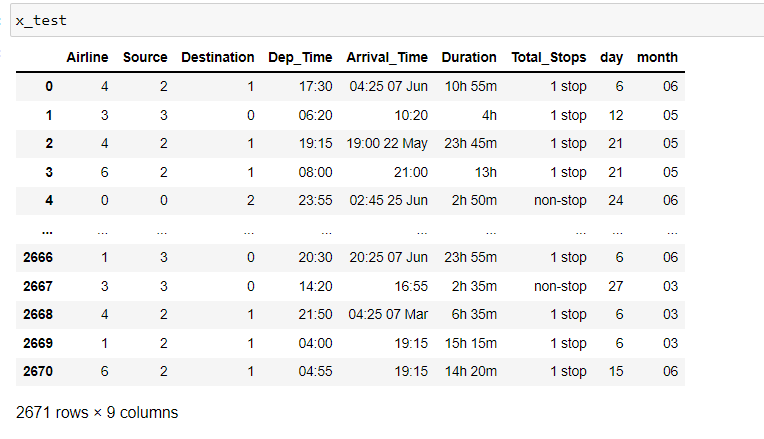
x\_test["year"].nunique()

#since all values are for 2019, year can be dropped

x\_test.drop(columns=["year"], axis = 1, inplace= True)

x\_test["Source"] = LE.fit\_transform(x\_test["Source"])

x\_test["Destination"] = LE.fit\_transform(x\_test["Destination"])



x\_test["Duration"] = x\_test["Duration"].astype(str)

x\_test["Total\_Stops"] = x\_test["Total\_Stops"].astype(str)

for i in range(0,len(x\_test)):

x\_test["Duration"][i] = x\_test["Duration"][i][0]

x\_test["Total\_Stops"][i] = x\_test["Total\_Stops"][i][0]

for i in range(0,len(x\_test)):

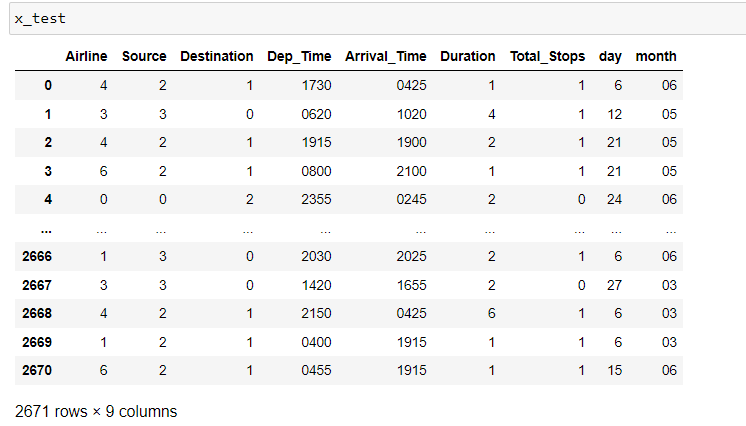
x\_test["Total\_Stops"][i] = x\_test["Total\_Stops"][i].replace("n", "0")

for i in range(0,len(x\_test)):

x\_test["Arrival\_Time"][i] = x\_test["Arrival\_Time"][i][0:5]

x\_test["Arrival\_Time"][i] = x\_test["Arrival\_Time"][i].replace(":","")

x\_test["Dep\_Time"][i] = x\_test["Dep\_Time"][i].replace(":","")



x\_test = x\_test.astype(int)

x\_test.dtypes

Airline int32

Source int32

Destination int32

Dep\_Time int32

Arrival\_Time int32

Duration int32

Total\_Stops int32

day int32

month int32

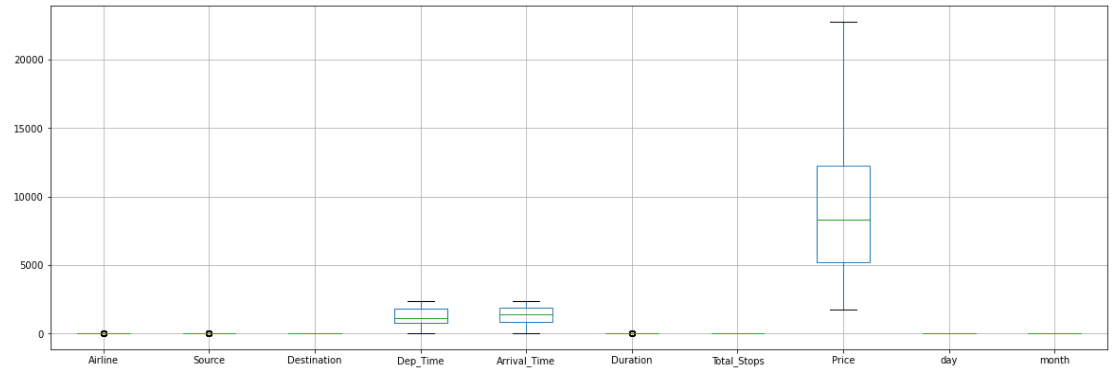
dtype: object

1. **Checking for outliers in the test data**

df.boxplot(figsize=[20,8])

plt.subplots\_adjust(bottom=0.25)

plt.show()



#Removing outliers by z score

from scipy.stats import zscore

z = np.abs(zscore(x\_test))

new\_df = x\_test[(z<3).all(axis=1)]

new\_df.shape

Data loss is 0.44926993635342566

#Data loss is negligible, hence dropping outliers

x\_test = new\_df

1. **Checking for skewness in the test data**

x\_test.skew() # check skewness

Airline 0.482865

Source -0.423251

Destination 1.262024

Dep\_Time 0.097679

Arrival\_Time -0.443477

Duration 1.449125

Total\_Stops 0.202178

day 0.199729

month -0.407966

dtype: float64

from sklearn.preprocessing import power\_transform

df\_new = power\_transform(x\_test)

df\_new = pd.DataFrame(df\_new, columns = x\_test.columns)

df\_new.skew()

Airline -0.042926

Source -0.238297

Destination 0.039494

Dep\_Time -0.114226

Arrival\_Time -0.499037

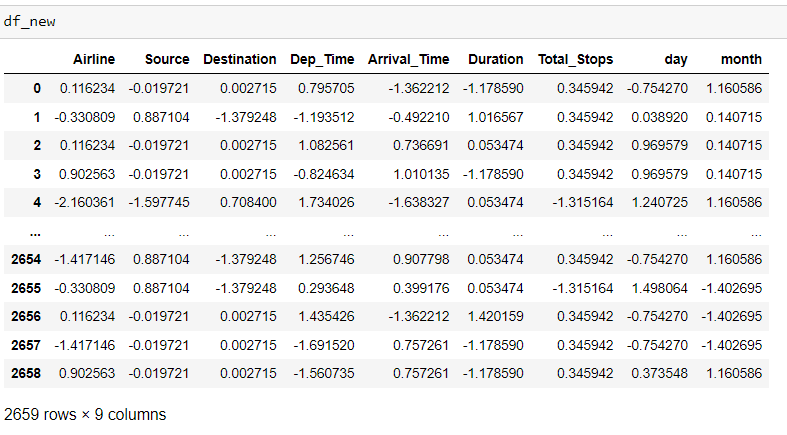
Duration 0.228801

Total\_Stops -0.075457

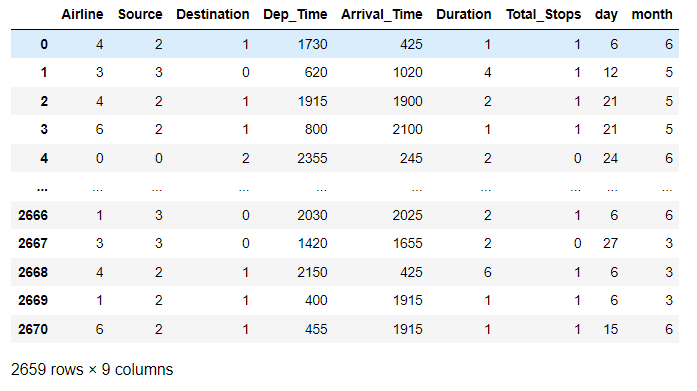
day -0.179598

month -0.228122

dtype: float64



x\_test



x\_test = df\_new

new\_model = DecisionTreeRegressor(max\_depth=4, min\_samples\_split=20)

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

predictions = new\_model.predict(x\_test)

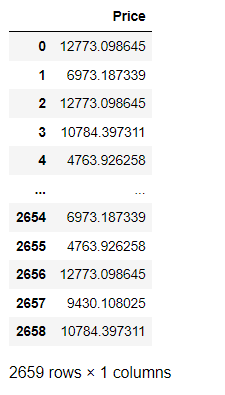
submission = pd.DataFrame()

submission['Price']= predictions

submission.head()

submission

submission.to\_csv('submission.csv',index=False)



We have achieved results using our final model which is Decision Tree with Hyper-parameter Tuning. The model efficiency was initially 68% and was boosted to 78% with hyper-parameter tuning.