Blog Submission

Insurance Claims-Fraud Detection

Submitted By:Vishal kumar

INTRODUCTION

The auto insurance industry is complicated and involves millions of dollars changing hands every day. And whenever there is a large amount of money running through complex systems, there is opportunity for fraud. This fraud can be committed by professionals and company working in the industry. But it can also be committed against them. By reading this, the first thing that comes into your mind is, what insurance fraud is. Let's understand this.

So, what is insurance fraud?

"Improper community committed by an individual in order to obtain favourable outcomes from the insurance company".

The insurance fraud can be broadly classified into 2 types.

1. **Soft Insurance Fraud**
2. **Hard Insurance Fraud**

Soft insurance fraud: A common example for this is, if the accident has taken place, but the amount of damage what has happened to the vehicle is very less. In such cases, the individual claims to the insurance company that huge amount of damage have occurred to the vehicle with the goal of charging the insurance company a higher bill.

Hard insurance fraud: In this type of fraud, an individual intentionally plans and invest the loss so that he can claims for the insurance from the company.

Common example for this type of fraud is staging a car wreck with the goal of benefitting from the resulting claim.

This project focuses on claim data of an Automobile insurance company. Because of fraudulent claims the insurance companies are losing huge amounts of money, which indirectly affects the public. Therefore, it is important to know which claims are genuine and which are fraud.

In this article we'll walk through how to spot insurance fraud and the consequences of engaging in it by building machine learning models and getting prediction of which claims are likely to be fraudulent. This enables an insurer to detect more fraudulent claims

1. Problem Definition

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, we are provided with 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.

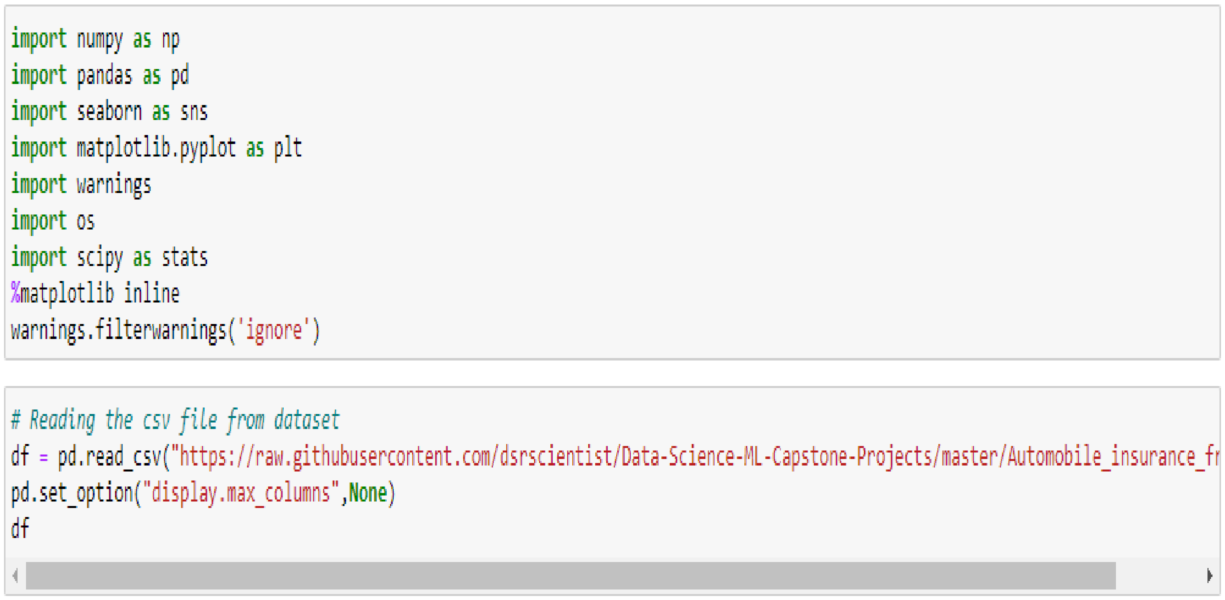
In this example, we will be working with some auto insurance data to demonstrate how we can create a predictive model that predicts if an insurance claim is fraudulent or not

The problem statement explains that the target variable contains the categories, so it is a "Classification Problem" where we need to predict whether an insurance claim is fraudulent or not.

2. Data Analysis

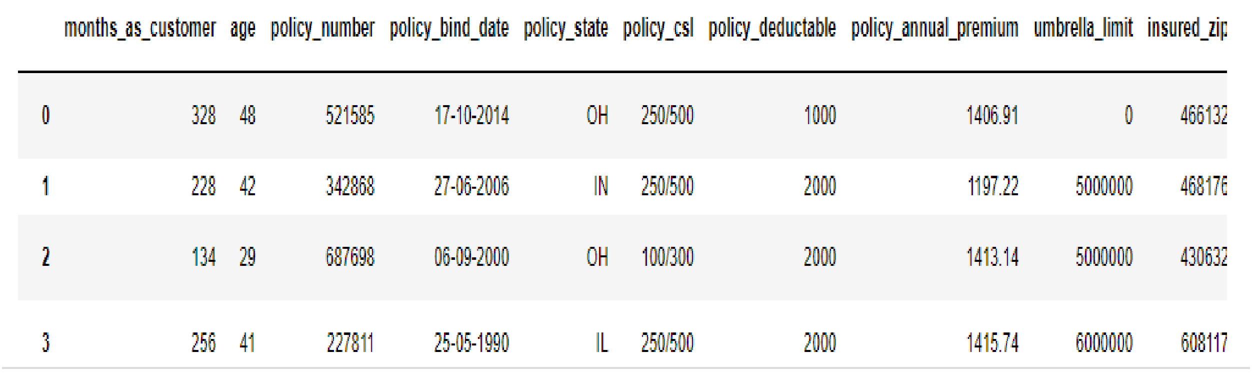
The process of cleaning, transforming and extracting data to discover the useful information for business decision making is called data analysis.

Importing necessary libraries and dataset



```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import os
import scipy as stats
%matplotlib inline
warnings.filterwarnings('ignore')
```

```
# Reading the csv file from dataset
df = pd.read_csv("https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fr
pd.set_option("display.max_columns",None)
df
```



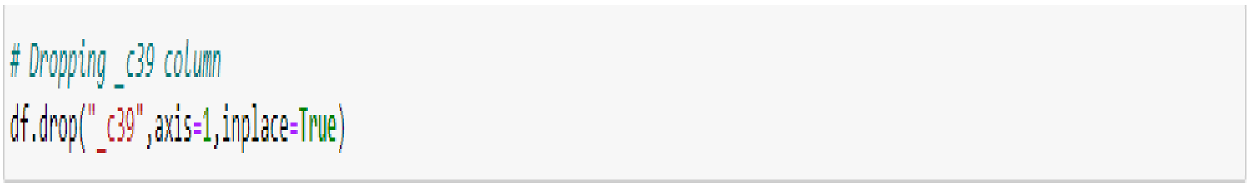
	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	17-10-2014	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	27-06-2006	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	06-09-2000	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	25-05-1990	IL	250/500	2000	1415.74	6000000	608117

I have imported the dataset which was in the csv file using pandas. The dataset contains 1000 rows and 40 columns having both numerical and categorical data. Here I can make use of PCA to reduce the columns, but this will give huge data loss. To avoid this, I am keeping the dataset as it is. We can also observe the null values, some "?" signs and some of the columns are in date format (dd/mm/yy), so we need to perform lots of preprocessing, cleaning to make our data usable to build the ML models.

Data Preparation and cleaning:

- **First, we need to check some statistical information about the dataset like checking shape, datatypes, nunique, value counts, info() etc.**
- **After checking the value counts, if we find any unwanted columns in the dataset then we need to do feature engineering on those columns based on the problem.**

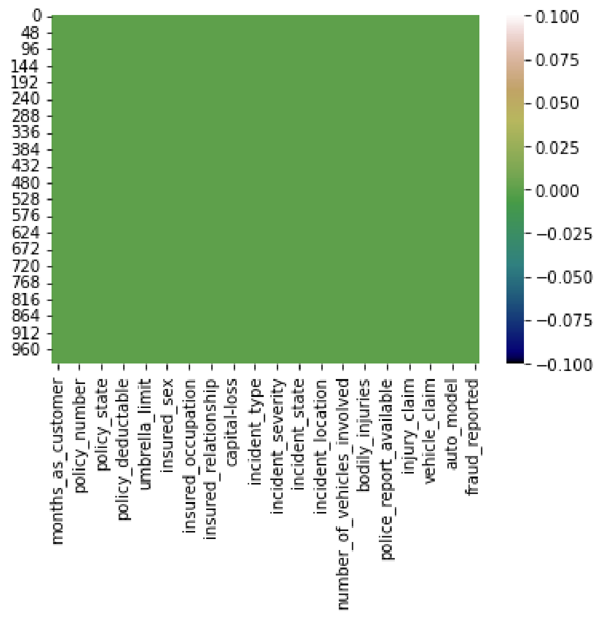
While I ran df.info(), I found c_39 column having one unique count as NAN throughout the dataset and it is of no use, so I dropped that column



```
# Dropping _c39 column
df.drop("_c39",axis=1,inplace=True)
```

After dropping the above column, if we observe the info, the counts of all the columns are same hence there are no missing values present in the dataset.

```
# Let's visualize the null values clearly
sns.heatmap(df.isnull(), cmap="gist_earth")
plt.show()
```

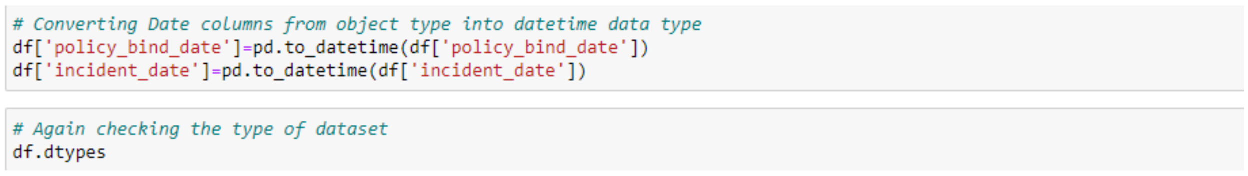


We can observe there are missing values present in the data. So we can proceed further.

Feature Extraction

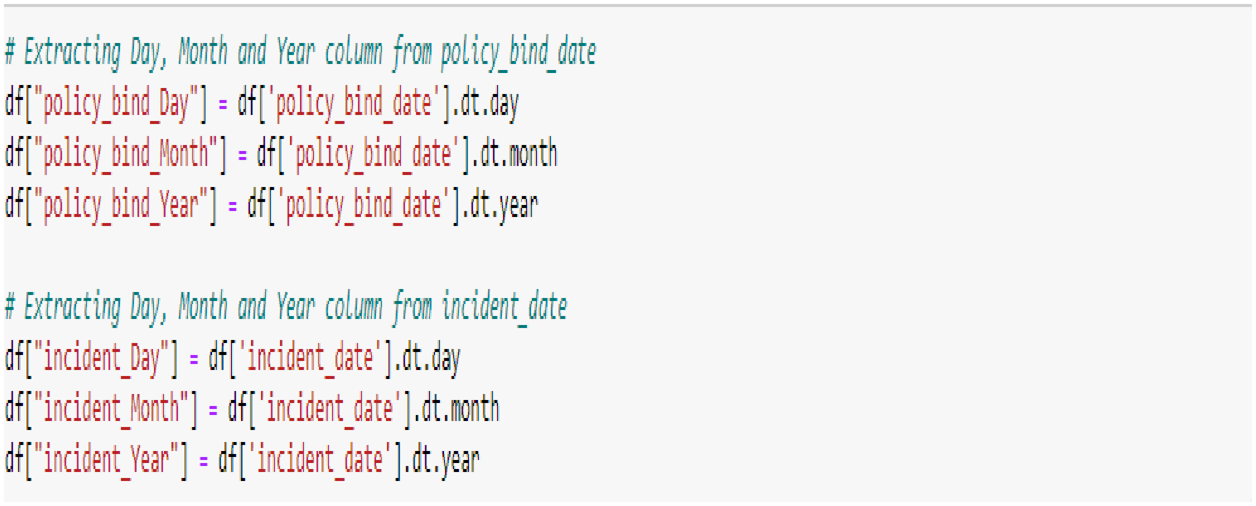
- **The columns policy_bind_date and incident_date have data in the form of dd/mm/yy and showing object data type which should be in datetime data type that means the python is not able to understand the type of these columns and giving default data type, this could be because of some special characters present in the data.**
- **So, we will convert this object data type into datetime data type to use them properly for the prediction and we will extract the values from these columns.**

policy_bind_date: Policy bind date means, on which date the policy was made. On this basis I have extracted day, month and year of policy made and dropped policy_bind_date as it is of no use.



```
# Converting Date columns from object type into datetime data type
df['policy_bind_date']=pd.to_datetime(df['policy_bind_date'])
df['incident_date']=pd.to_datetime(df['incident_date'])
```

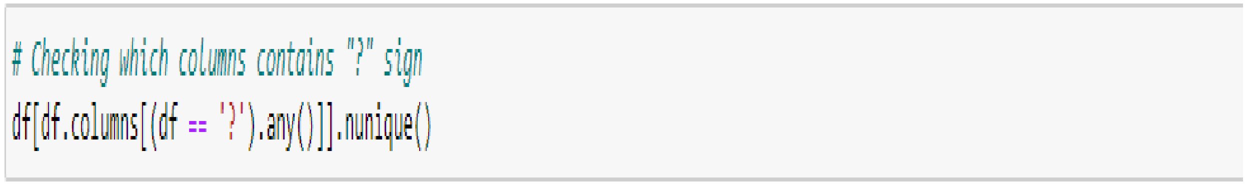
```
# Again checking the type of dataset
df.dtypes
```



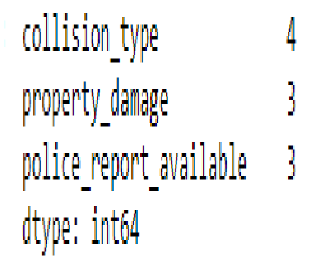
```
# Extracting Day, Month and Year column from policy_bind_date
df["policy_bind_Day"] = df['policy_bind_date'].dt.day
df["policy_bind_Month"] = df['policy_bind_date'].dt.month
df["policy_bind_Year"] = df['policy_bind_date'].dt.year

# Extracting Day, Month and Year column from incident_date
df["incident_Day"] = df['incident_date'].dt.day
df["incident_Month"] = df['incident_date'].dt.month
df["incident_Year"] = df['incident_date'].dt.year
```

Incident_date: It gives us that, on which date the incident occurs. This column contains only 2015 year data so I have extracted only day and month from this column and after that dropped the column accordingly.<u>Replacing "?" sign</u>

By looking at the dataset and value count function, we have found some columns having "?" sign. These are not to be considered as NAN values but we need to fill them. First, we will check which columns contains "?" sign

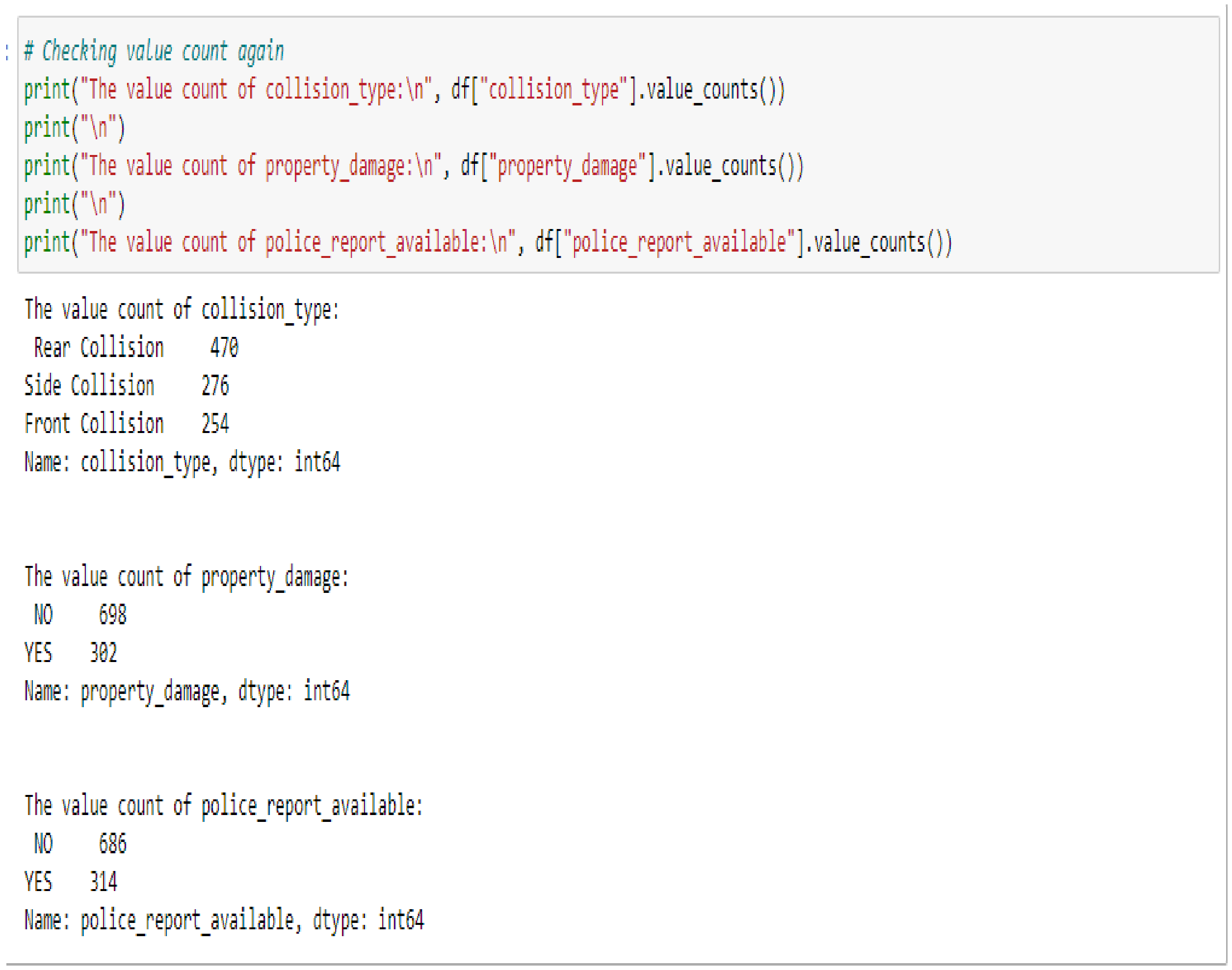


```
# Checking which columns contains "?" sign
df[df.columns[(df == '?').any()]].nunique()
```



```
collision_type           4
property_damage          3
police_report_available  3
dtype: int64
```

- **These are the columns which contains "?" sign.**
- **There are two ways to fill these values. Either you can fill them by using appropriate values or you can fill them by using their mode.**
- **Since these columns seems to be categorical so we will replace "?" values with most frequently occuring values of the respective columns that is their mode values. To do this let's check the value count of these columns.**

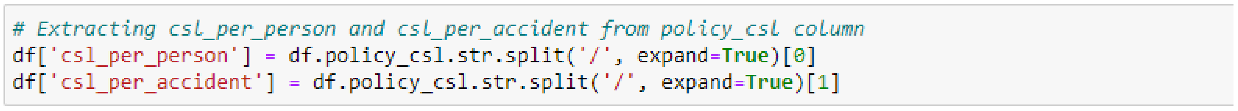


```
# Checking value count again
print("The value count of collision_type:\n", df["collision_type"].value_counts())
print("\n")
print("The value count of property_damage:\n", df["property_damage"].value_counts())
print("\n")
print("The value count of police_report_available:\n", df["police_report_available"].value_counts())
```

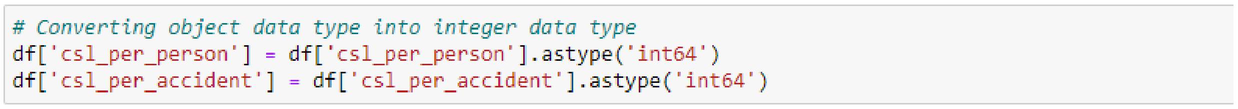
```
The value count of collision_type:
 Rear Collision     470
Side Collision     276
Front Collision    254
Name: collision_type, dtype: int64


The value count of property_damage:
 NO     698
YES    302
Name: property_damage, dtype: int64


The value count of police_report_available:
 NO     686
YES    314
Name: police_report_available, dtype: int64
```

- **The mode of property_damage and police_report_available is again “?”, so we will use the second highest count in these columns that is “NO”.**
- **So, we will replace the “?” sign in collision_type, property_damage and police_report_available are Rear Collision and NO respectively.**

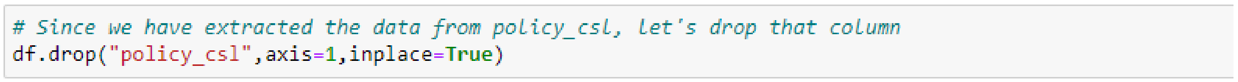
- **The policy_csl column is also showing object data type but it is numerical in nature. May be, it is because of the presence of "/" sign in that column.**
- **This policy csl auto insurance is a combination of both bodily injury and property damage. On this basis we will extract two columns namely csl_per_person and csl_per_accident from policy_csl colums and then will convert their object data type into integer data type.**



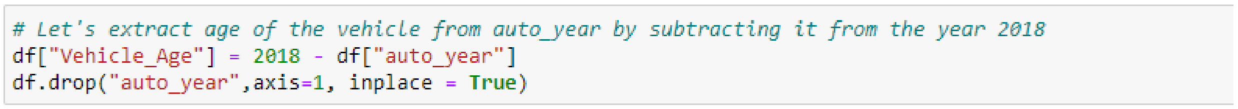
```
# Extracting csl_per_person and csl_per_accident from policy_csl column
df['csl_per_person'] = df.policy_csl.str.split('/', expand=True)[0]
df['csl_per_accident'] = df.policy_csl.str.split('/', expand=True)[1]
```



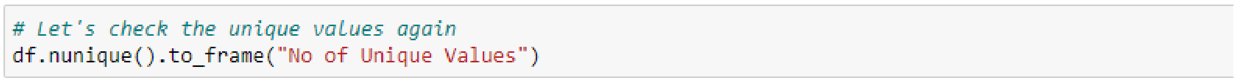
```
# Converting object data type into integer data type
df['csl_per_person'] = df['csl_per_person'].astype('int64')
df['csl_per_accident'] = df['csl_per_accident'].astype('int64')
```



```
# Since we have extracted the data from policy_csl, let's drop that column
df.drop("policy_csl",axis=1,inplace=True)
```



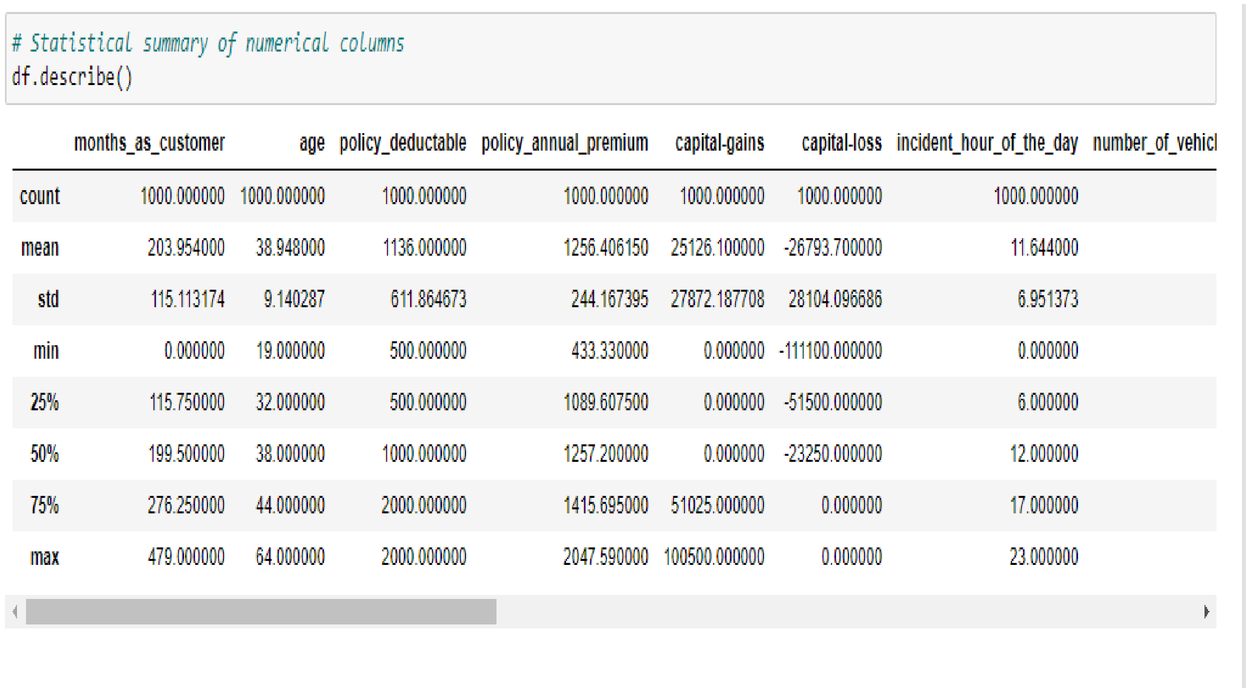
```
# Let's extract age of the vehicle from auto_year by subtracting it from the year 2018
df["Vehicle_Age"] = 2018 - df["auto_year"]
df.drop("auto_year",axis=1, inplace = True)
```



```
# Let's check the unique values again
df.nunique().to_frame("No of Unique Values")
```

Here I have extracted 2 columns from policy_csl and converted them into integer data and dropped policy_csl column after getting the required data from it.

- **In the dataset there is one column named auto_year which gives the age of the vehicle. So, we will extract age of the vehicle from auto_year by subtracting it by 2018. Assuming the data is collected in the 2018.**
- **We have successfully dealt with the feature engineering and now we will move further to know about the statistical summary of the dataset.**



```
# Statistical summary of numerical columns
df.describe()
```

	months_as_customer	age	policy_deductable	policy_annual_premium	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicl
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	
mean	203.954000	38.948000	1136.000000	1256.406150	25126.100000	-26793.700000	11.644000	
std	115.113174	9.140287	611.864673	244.167395	27872.187708	28104.096686	6.951373	
min	0.000000	19.000000	500.000000	433.330000	0.000000	-111100.000000	0.000000	
25%	115.750000	32.000000	500.000000	1089.607500	0.000000	-51500.000000	6.000000	
50%	199.500000	38.000000	1000.000000	1257.200000	0.000000	-23250.000000	12.000000	
75%	276.250000	44.000000	2000.000000	1415.695000	51025.000000	0.000000	17.000000	
max	479.000000	64.000000	2000.000000	2047.590000	100500.000000	0.000000	23.000000	

The describe method gives the statistical information of the dataset. The summary of this dataset looks perfect since there are no negative/ invalid values present. It gives the summary of numerical data. From the above description we can observe the following things

- **Here the counts of all the columns are equal which means there are no missing values in the dataset.**
- **In some of the columns like policy_deductable, capital-gains, injury_claim etc we can observe the mean value is greater than the median (50%) which means the data in those columns are skewed to right.**
- **And in some of the columns like total_claim_amount, vehicle_claim...etc we can observe the median is greater than the mean which means the data in the columns are skewed to left.**
- **And some of the columns have equal mean and median that means the data symmetric and is normally distributed and no skewness present.**
- **There is a huge difference in 75% and max it shows that huge outliers present in the columns.**

Data Visualization

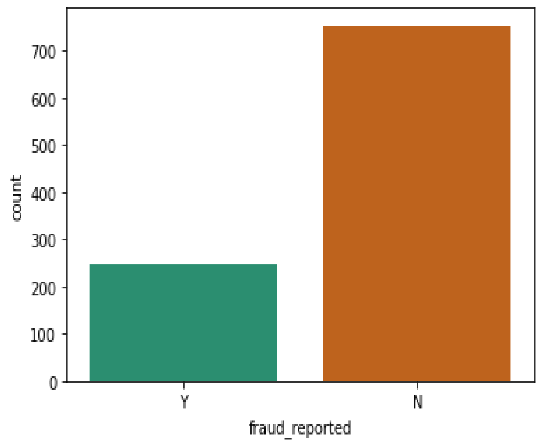
Before going to visualize the data, first we need to separate numerical and categorical columns.

```
# Separating numerical and categorcal columns

# Checking for categorical columns
categorical_col=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_col.append(i)
print("Categorical columns are:\n",categorical_col)
print("\n")

# Now checking for numerical columns
numerical_col=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_col.append(i)
print("Numerical columns are:\n",numerical_col)
```

```
Categorical columns are:
 ['policy_state', 'insured_sex', 'insured_education_level', 'insured_occupation', 'insured_hobbies', 'insured_relationship', 'i
ncident_type', 'collision_type', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_city', 'property_dam
age', 'police_report_available', 'auto_make', 'auto_model', 'fraud_reported']


Numerical columns are:
 ['months_as_customer', 'age', 'policy_deductable', 'policy_annual_premium', 'capital-gains', 'capital-loss', 'incident_hour_of
_the_day', 'number_of_vehicles_involved', 'bodily_injuries', 'witnesses', 'total_claim_amount', 'injury_claim', 'property_clai
m', 'vehicle_claim', 'policy_bind_Day', 'policy_bind_Month', 'policy_bind_Year', 'incident_Day', 'incident_Month', 'csl_per_per
son', 'csl_per_accident', 'Vehicle_Age']
```

```
#Visualizing how many insurance claims is fraudulent
print(df["fraud_reported"].value_counts())
sns.countplot(df["fraud_reported"],palette="Dark2")
plt.show()
```

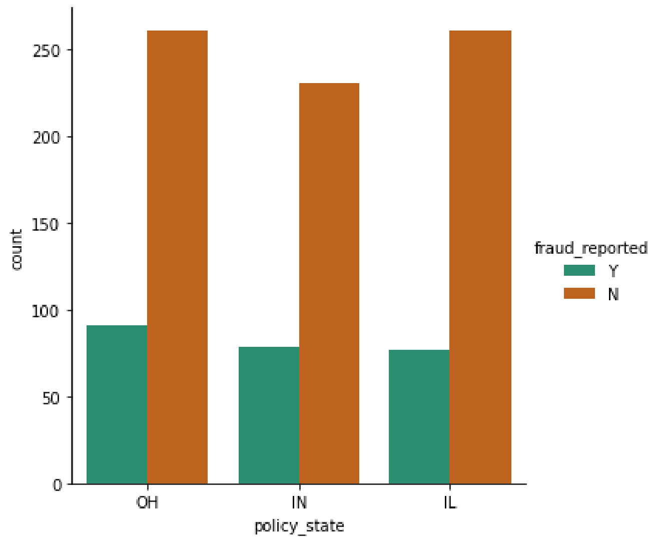
```
N    753
Y    247
Name: fraud_reported, dtype: int64
```



- **From the plot we can observe that the count of "N" is high compared to "Y". Here we can assume that "Y" stands for "Yes" that is the insurance claim is fraudulent and "N" stands for "No" means the insurance claim is not fraudulent. There are 247 frauds and 753 non-frauds.**
- **Since we are dealing the classification problem, the target fraud_reported it indicates the class imbalance issue. We need to balance the data before proceed with our models.**

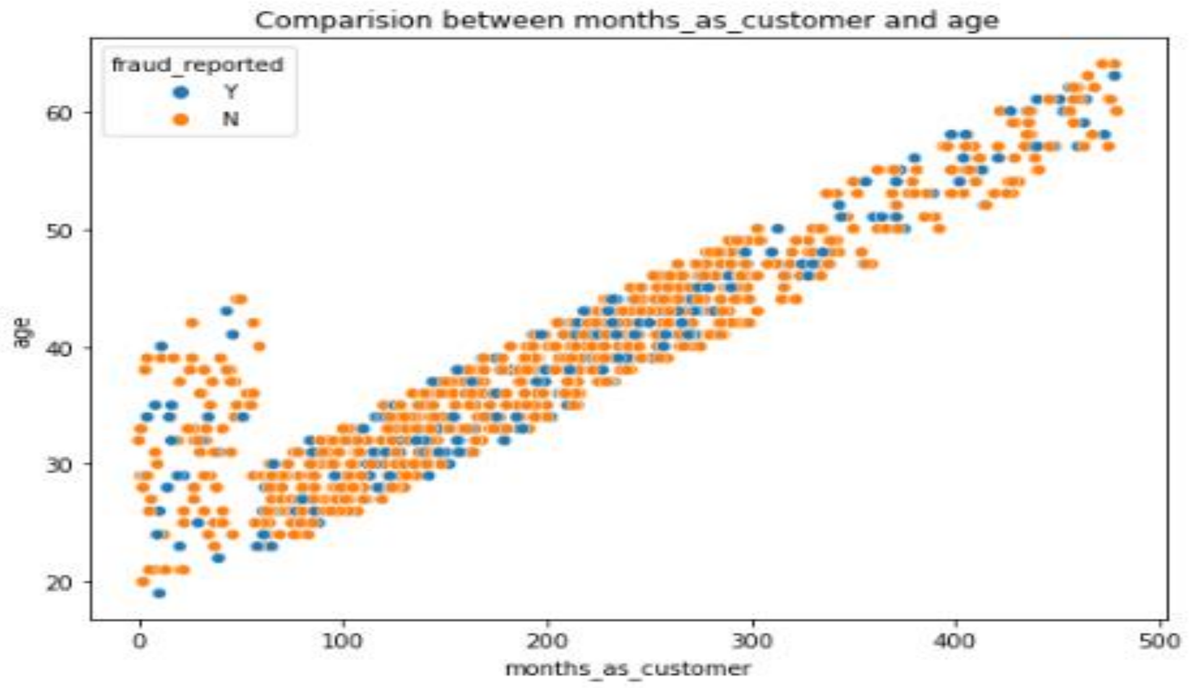
Now we will compare the features and label by visualizing the data.

```
# Comparing policy_state and fraud_reported
sns.factorplot('policy_state',kind='count',data=df,hue='fraud_reported',palette="Dark2")
plt.show()
```



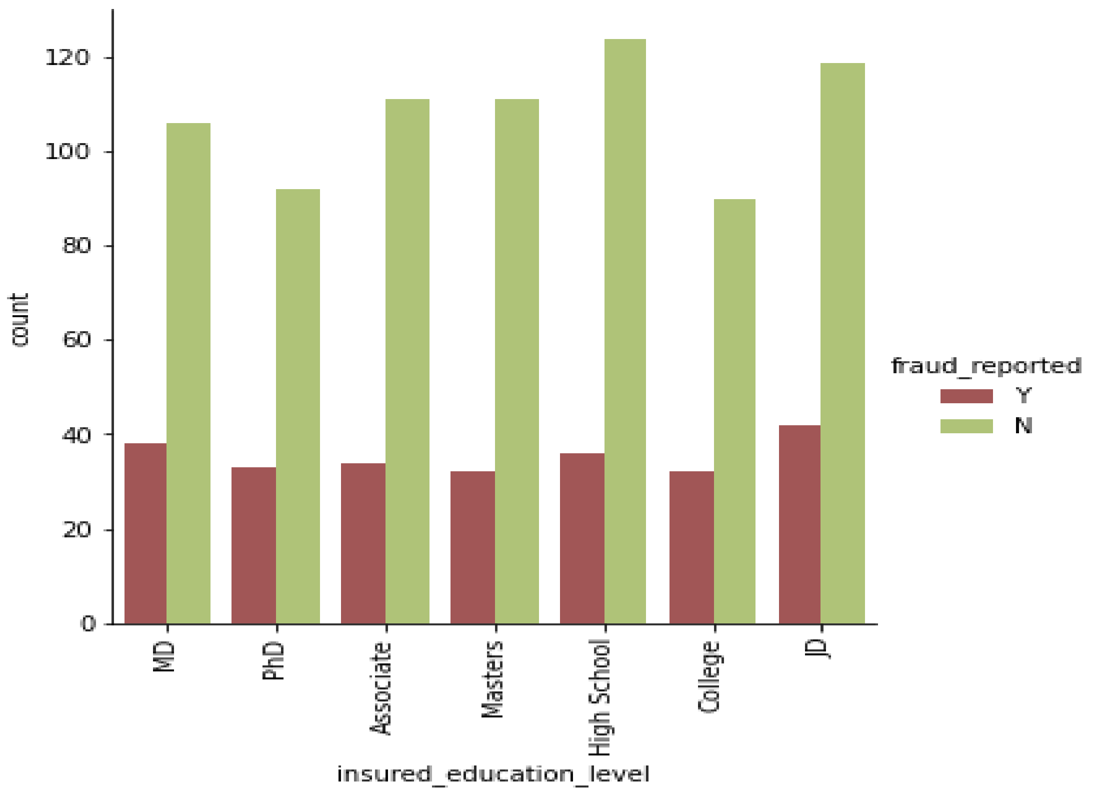
I have used count plot to compare policy state and fraud report. I can notice that the types of the policy claimed by the customers are almost same but still the fraud report is bit high in OH policy state compared to others and the types IL and IN have similar fraud reports.

```
plt.subplot(2,2,1)
plt.title('Comparision between months_as_customer and age')
sns.scatterplot(df['months_as_customer'],df['age'],hue=df['fraud_reported']);
```

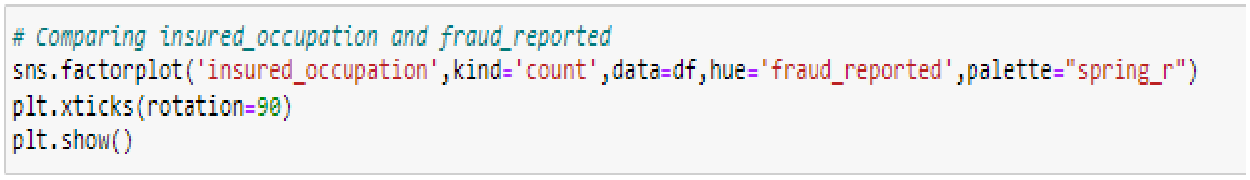


From the above scatter plot, we can observe the strong linear relationship between the age and month_as_customer. Which means as the month_as_customer increases, the age of the person also increases. Also, as the person getting older, the frequency of the both fraud report classes are vanishing slowly. That means, the people having young age are more likely to have high fraud reports.

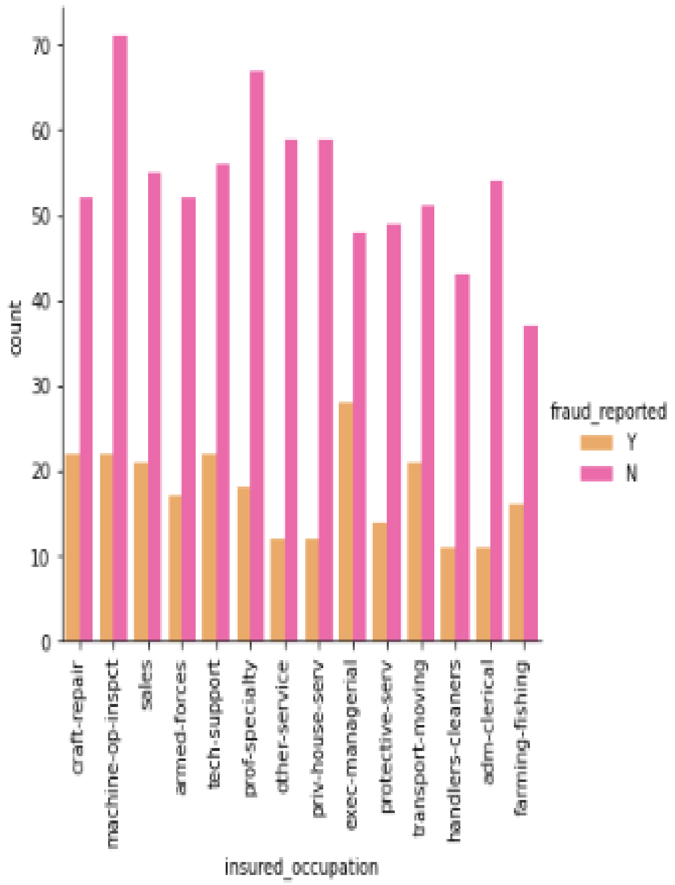
```
# Comparing insured_education_level and fraud_reported

sns.factorplot('insured_education_level',kind='count',data=df,hue='fraud_reported',palette="tab20b_r")
plt.xticks(rotation=90)
plt.show()
```



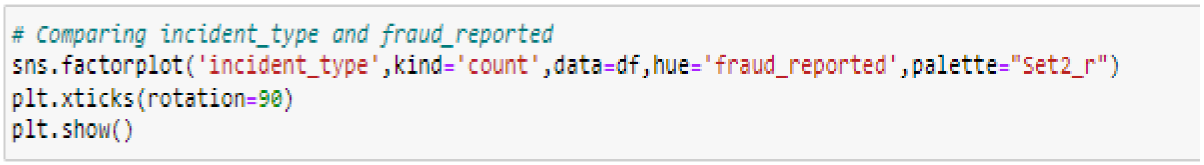
- **From the above factor plot we can observe the fraudulent level is very less for the people who have high school education and the people who have completed their "JD" education have high fraud report.**
- **The people who have high insured education are facing insurance fraudulent compared to the people with less insured education level. That means the people with less education level are more trustworthy.**



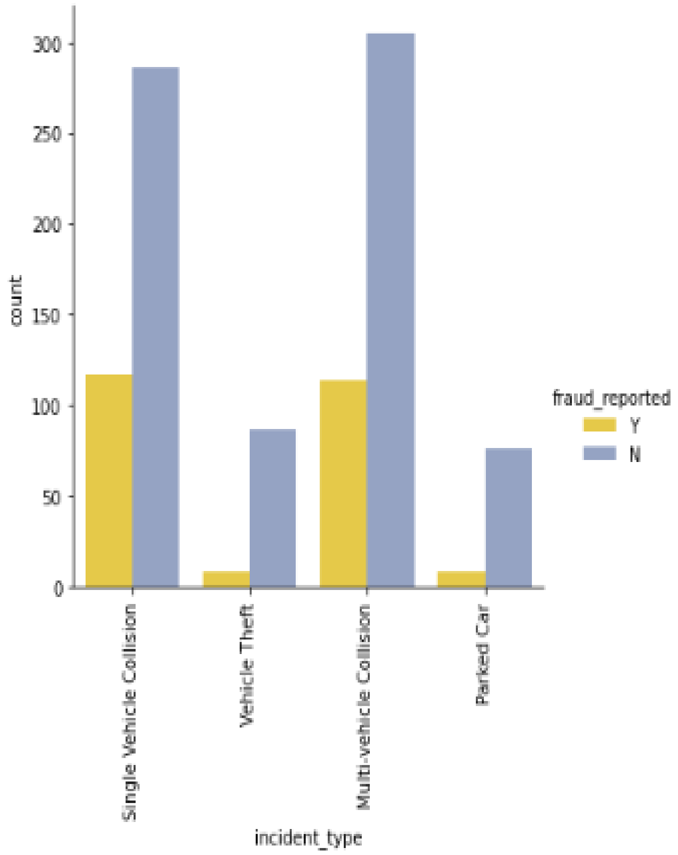
```
# Comparing insured_occupation and fraud_reported
sns.factorplot('insured_occupation',kind='count',data=df,hue='fraud_reported',palette="spring_r")
plt.xticks(rotation=90)
plt.show()
```



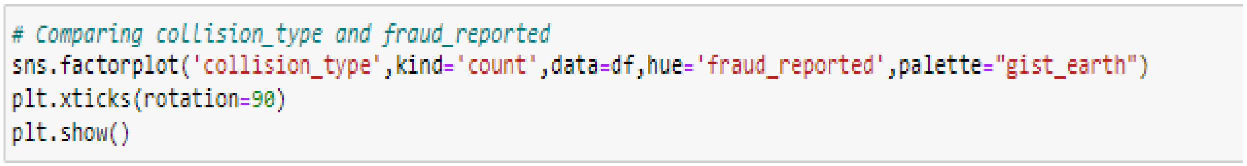
- **In the insured occupation we can observe most of the data is covered by machine operation inspector followed by professional speciality. Apart from this all the other insured occupations have almost same counts.**
- **The people who are in the position exec-managerial have high fraud reports compared to others and the non-fraud report is high for machine operation inspector that means the people having good occupations have more fraud reports.**



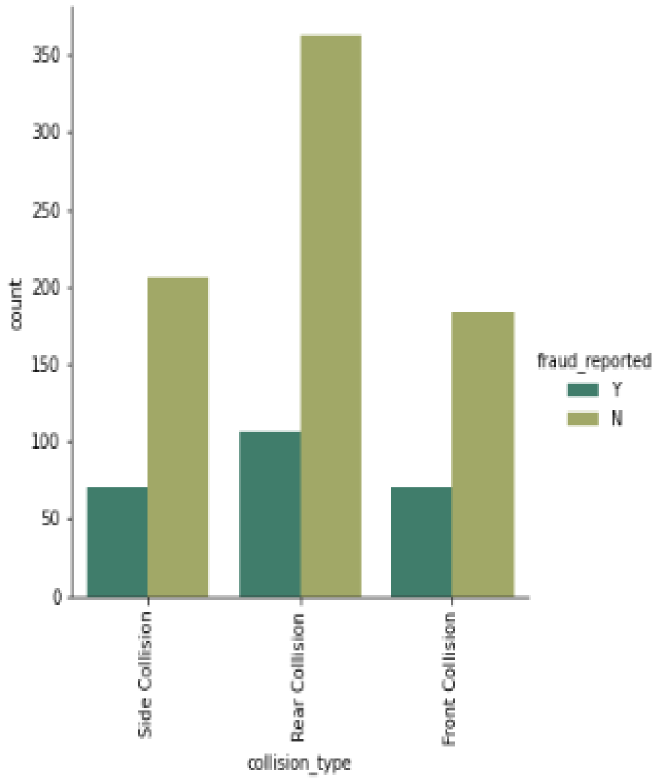
```
# Comparing incident_type and fraud_reported
sns.factorplot('incident_type',kind='count',data=df,hue='fraud_reported',palette="Set2_r")
plt.xticks(rotation=90)
plt.show()
```



- **From the above factor plot we can notice that the Multi-vehicle collision and Single Vehicle Collision have pretty much similar counts. But the count is very less in Parked car and Vehicle Theft.**
- **In Multivehicle collision and single vehicle collision, the fraud report is very high compared to others and the nonfraud reports are also almost similar for these types.**

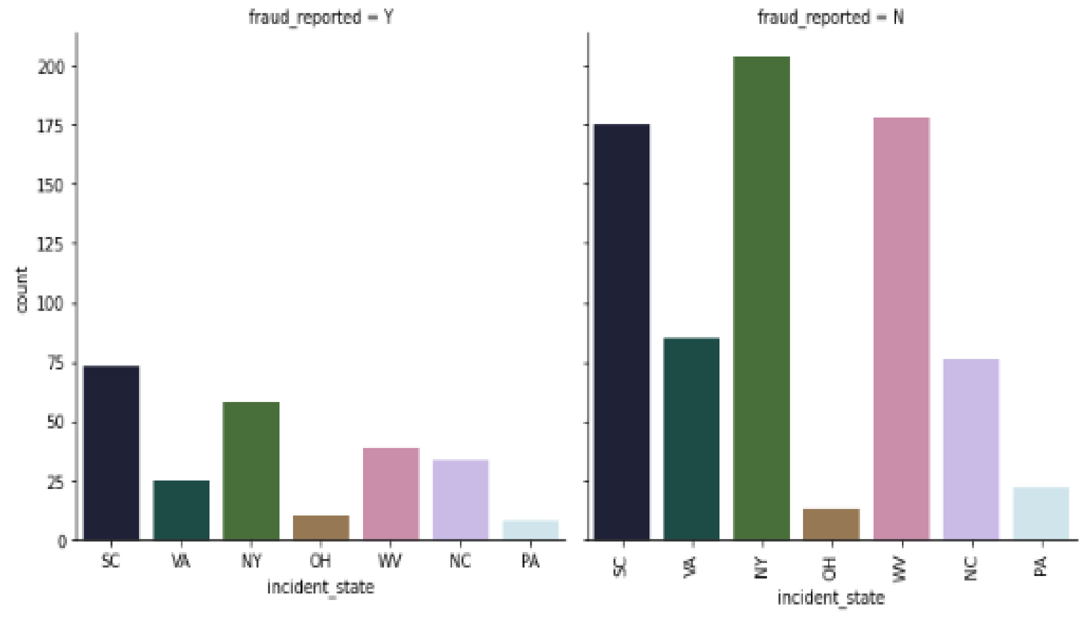


```
# Comparing collision_type and fraud_reported
sns.factorplot('collision_type',kind='count',data=df,hue='fraud_reported',palette="gist_earth")
plt.xticks(rotation=90)
plt.show()
```



- The collision type has 3 different types. The count is high in Rear collision and the other two types have almost equal counts.
- The fraud level is high in the collision type Rear Collision and other two collision type have average reports. Also the nonfraud reports also high in Rear collision type.

```
# Comparing incident_state and fraud_reported
sns.factorplot('incident_state',kind='count',data=df,col='fraud_reported',palette="cubehelix")
plt.xticks(rotation=90)
plt.show()
```

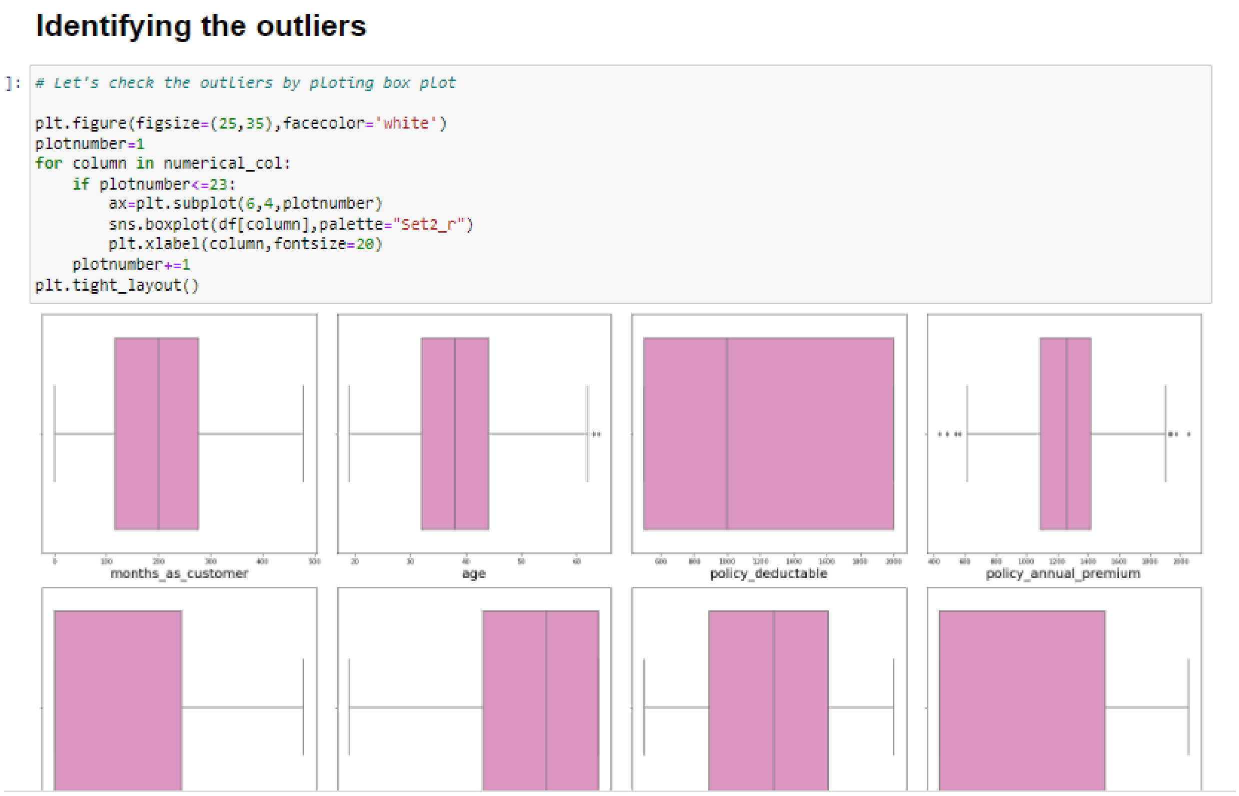


- **With respect to the incident state, New York, South Carolina and West Virginia states have highest counts.**
- **The state SC has high fraud reports followed by New York compared to other states.**

3.EDA Concluding Remark

- **I have checked the null values in the dataset and there were no missing values found.**
- **I have dropped some of the irrelevant columns to overcome with the multicollinearity problem.**
- **I have extracted some new features from the existing features to get better results without any hindrance. And dropped the old columns, if I keep them as it is they will act as duplicates and that leads to multicollinearity problem.**
- **Replaced the corrupted entries “?” in the columns with their respective mode values.**
- **Coming to the visualization part, we have found when and where the fraud reports are high in number.**
- **To get the better insights about the features, I have used violin plots, box plots, scatter plots and distribution plots.**
-

Checking outliers and skewness in the data

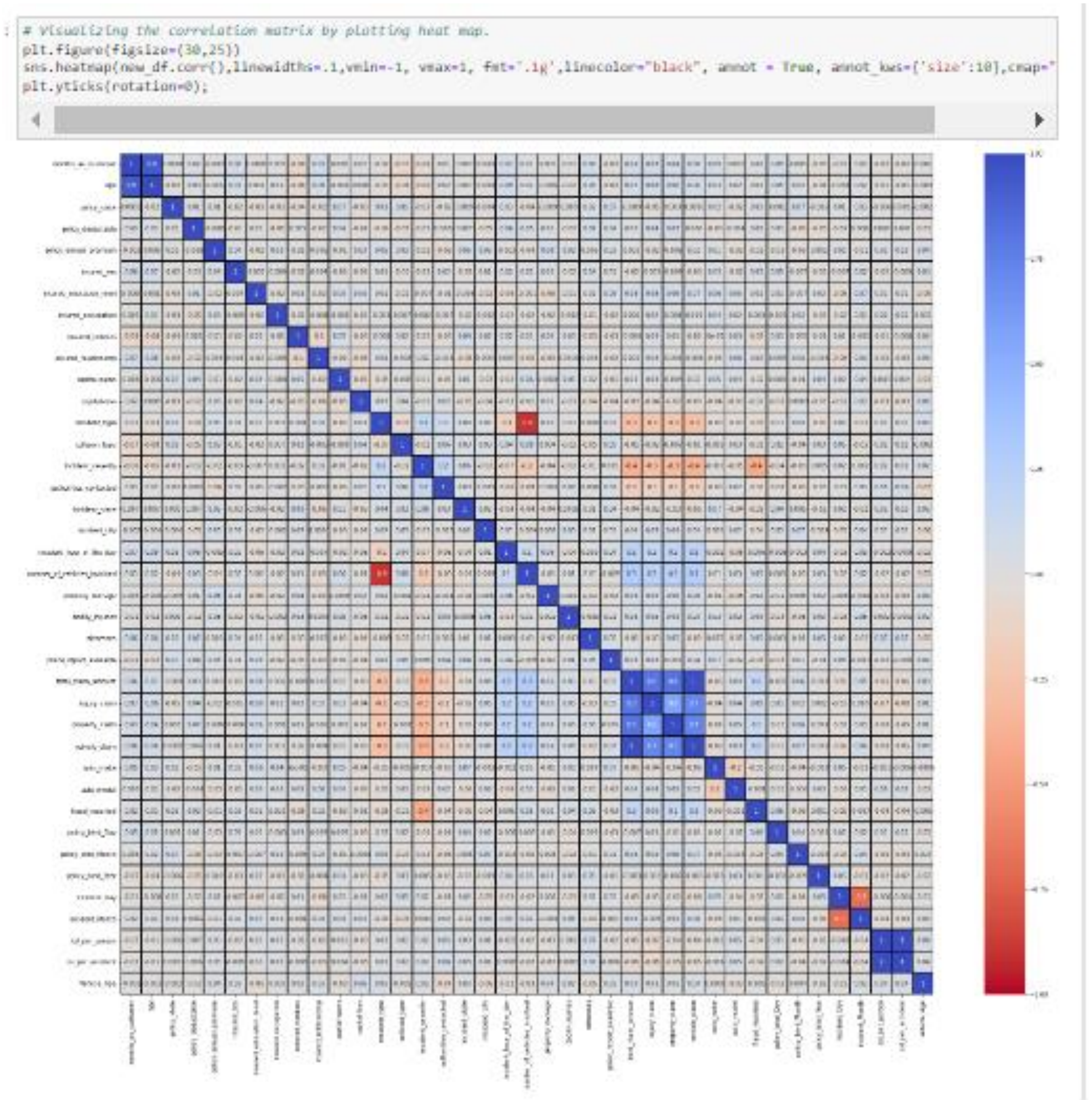


Identifying the outliers

```
]: # Let's check the outliers by ploting box plot

plt.figure(figsize=(25,35),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=23:
        ax=plt.subplot(6,4,plotnumber)
        sns.boxplot(df[column],palette="Set2_r")
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

- I have used boxplots to check the outliers in the dataset and found outliers in age, policy_annual_premium, total_claim_amount, property_claim amd incident_Month.
- To remove outliers, I used both Zscore and IQR methods

<u>Correlation between the label and features using Heat map</u>



```
# Visualizing the correlation matrix by plotting heat map.
plt.figure(figsize=(30,25))
sns.heatmap(new_df.corr(),linewidths=.1,vmin=-1, vmax=1, fmt='.1g',linecolor="black", annot = True, annot_kws={'size':10},cmap="
plt.yticks(rotation=0);
```

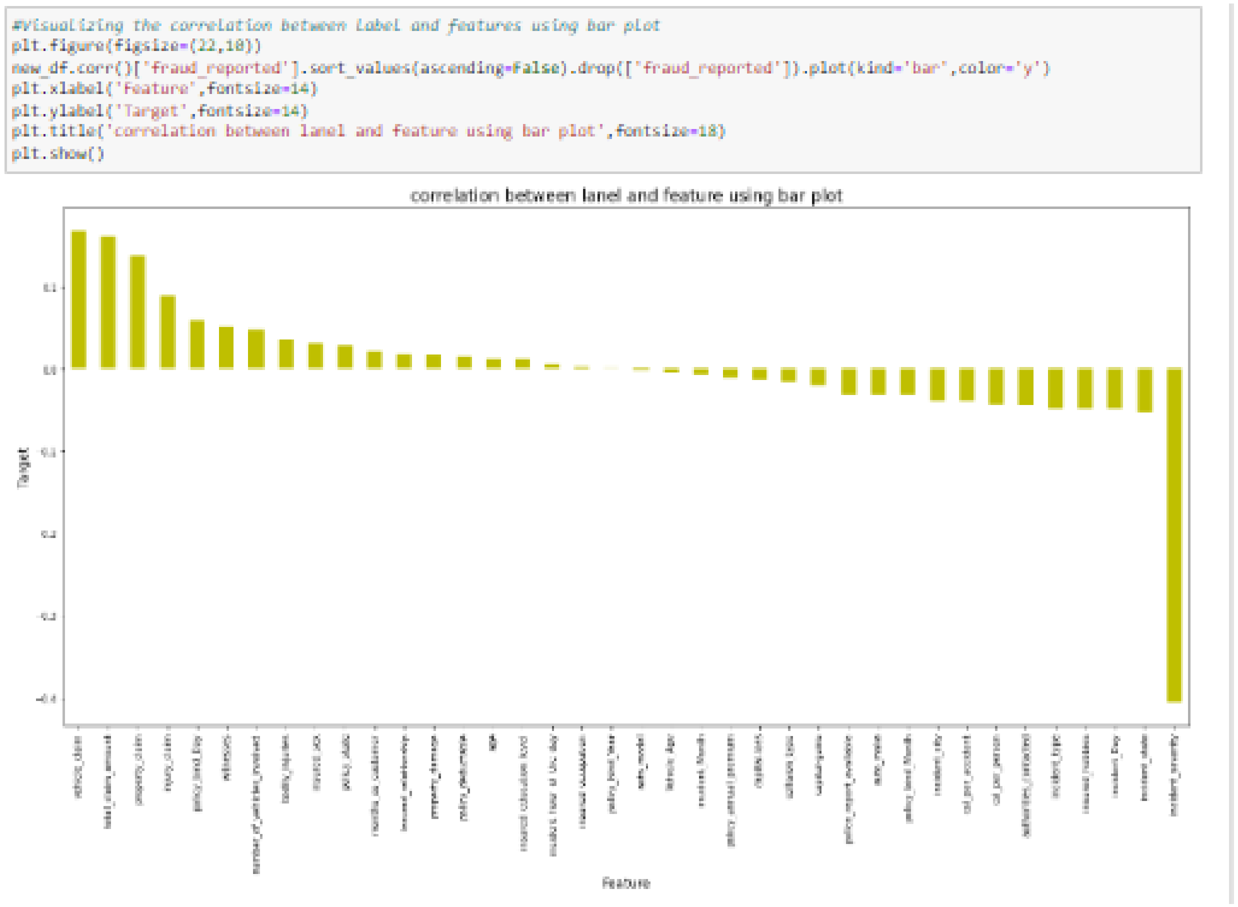
This heatmap shows the correlation matrix by visualizing the data. we can observe the relation between one feature to other.

This heat map contains both positive and negative correlation.

- **There is very less correlation between the target and the label.**
- **We can observe the most of the columns are highly correlated with each other which leads to the multicollinearity problem.**

- **We will check the VIF value to overcome with this multicollinearity problem.**

To get the better insights from the heat map I have used bar plot to show the positive and negative correlation.



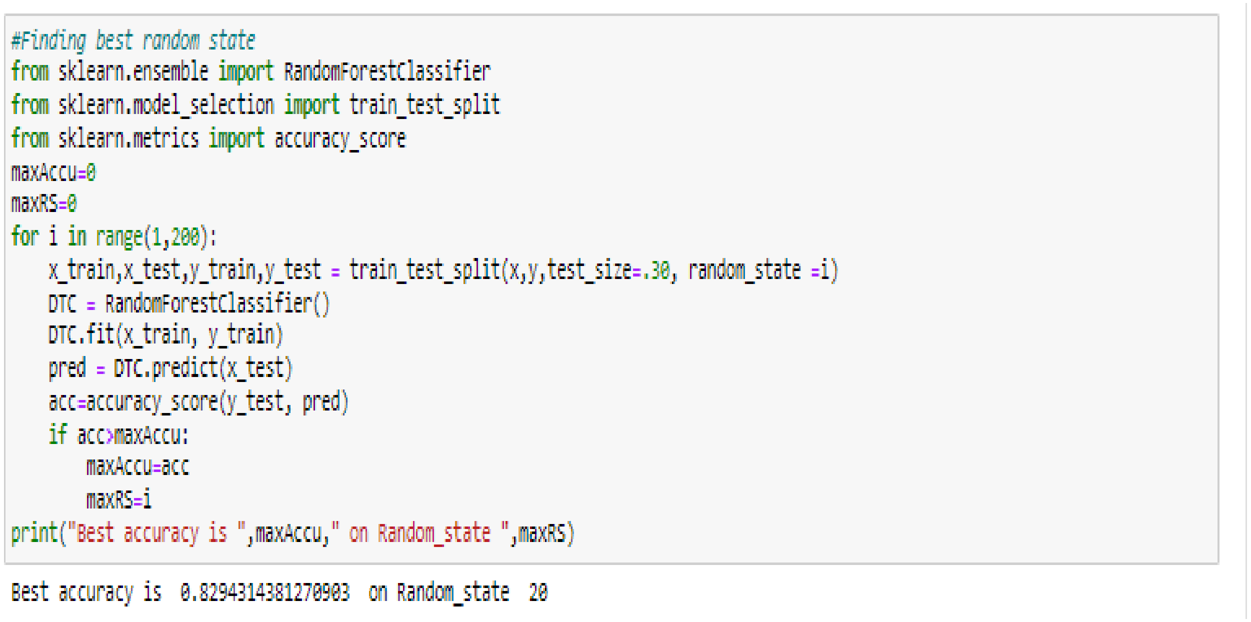
```
#Visualizing the correlation between label and features using bar plot
plt.figure(figsize=(22,10))
new_df.corr()['fraud_reported'].sort_values(ascending=False).drop(['fraud_reported']).plot(kind='bar',color='y')
plt.xlabel('Feature',fontsize=14)
plt.ylabel('Target',fontsize=14)
plt.title('correlation between lanel and feature using bar plot',fontsize=18)
plt.show()
```

- **From the bar plot we can observe that the columns policy_bind_Year, insured_occupation and auto_model age are very less correlated with the target. We can drop these columns if necessary.**

- **But for now, I am keeping these columns as it is, if I do not get the better accuracy then I can drop these columns in order to get better accuracy.**

Building Machine Learning Models

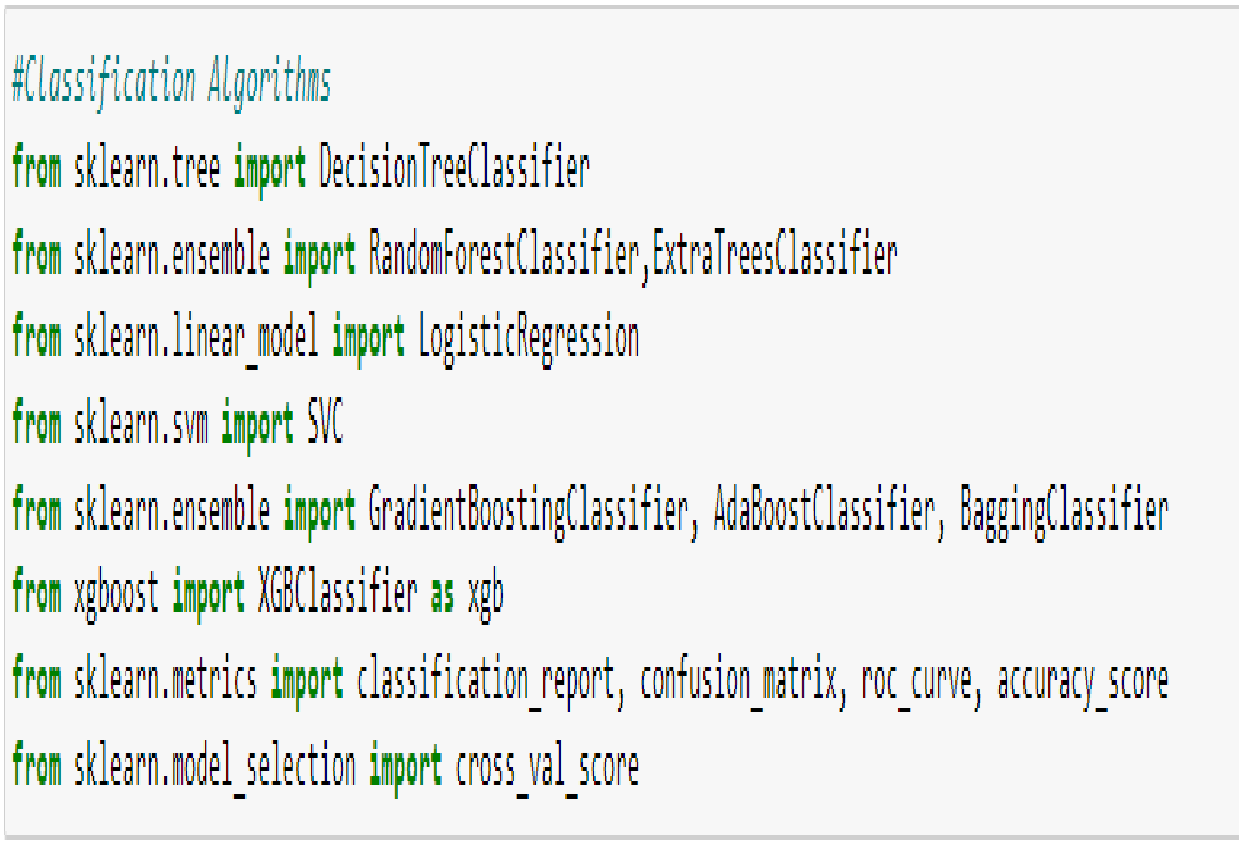
Before building the models, first we need to find the best random state and accuracy using any one of the classification models.



```
#Finding best random state
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
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)
    DTC = RandomForestClassifier()
    DTC.fit(x_train, y_train)
    pred = DTC.predict(x_test)
    acc=accuracy_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.8294314381270903  on Random_state  20
```

- **I have got best random state as 20 and best accuracy as 82.94% using Random Forest Classifier.**
- **Now let's create new train and test and fit them into the models to find our ideal model.**

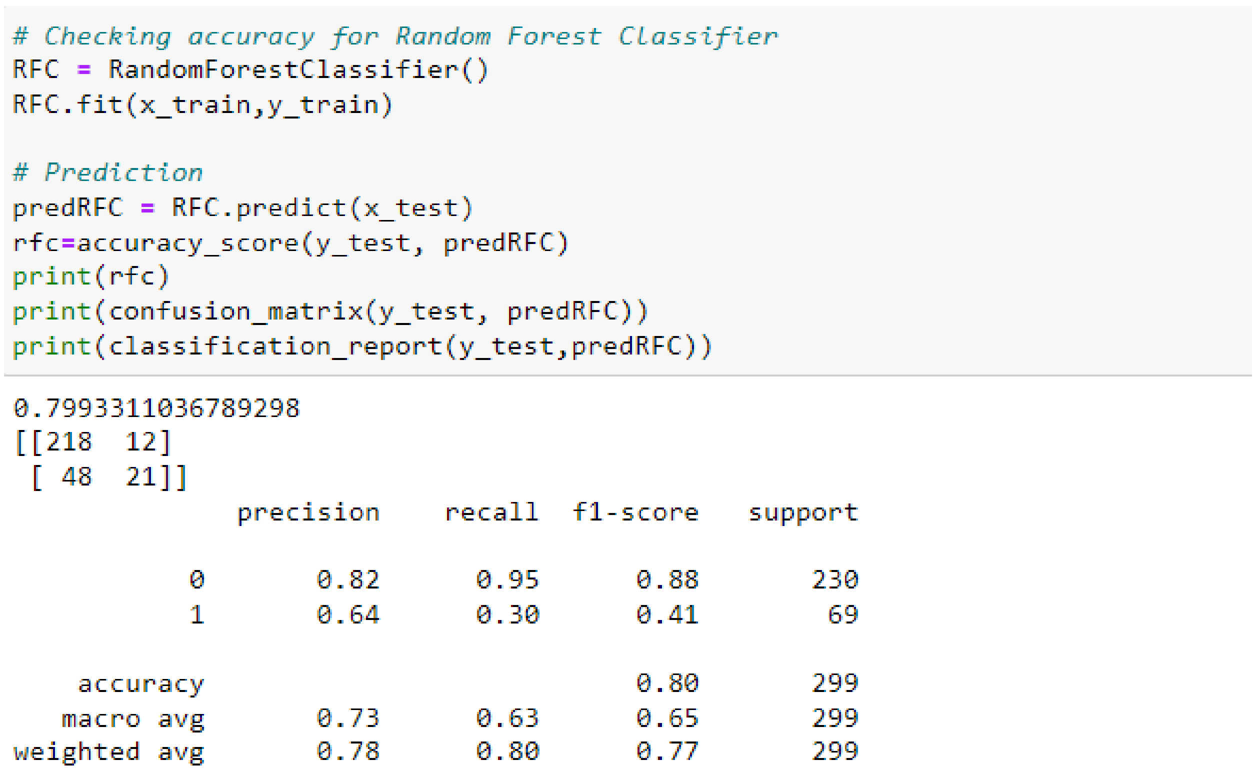


```
#Classification Algorithms
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier
from xgboost import XGBClassifier as xgb
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, accuracy_score
from sklearn.model_selection import cross_val_score
```

- **I will be using the above classification algorithms to get the ideal one for prediction.**

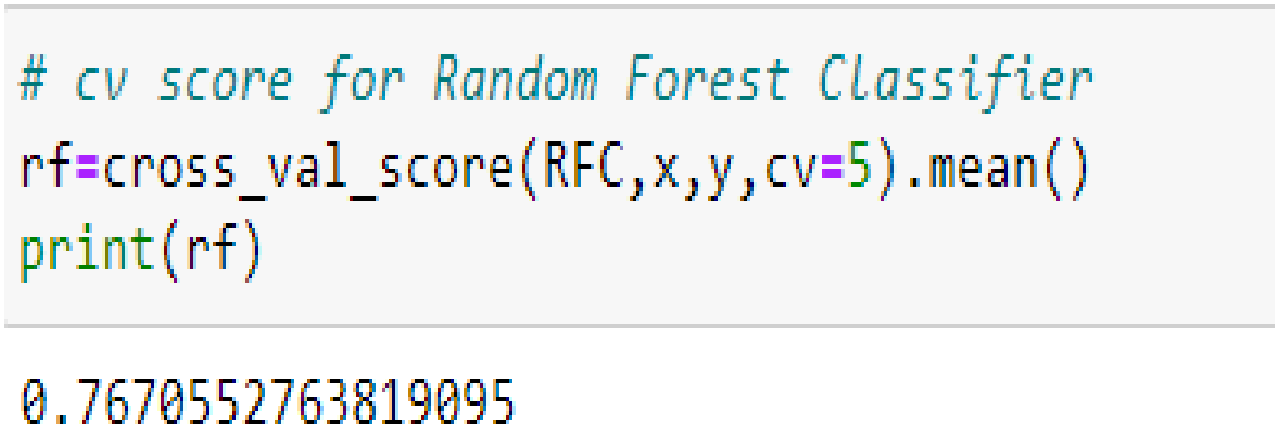
- **I have used evaluation metrics like classification report, confusion matrix, roc score and accuracy score. Also used cross validation score to get the difference from the model accuracy.**

Random Forest Classifier



```
# Checking accuracy for Random Forest Classifier
RFC = RandomForestClassifier()
RFC.fit(x_train,y_train)

# Prediction
predRFC = RFC.predict(x_test)
rfc=accuracy_score(y_test, predRFC)
print(rfc)
print(confusion_matrix(y_test, predRFC))
print(classification_report(y_test,predRFC))
```

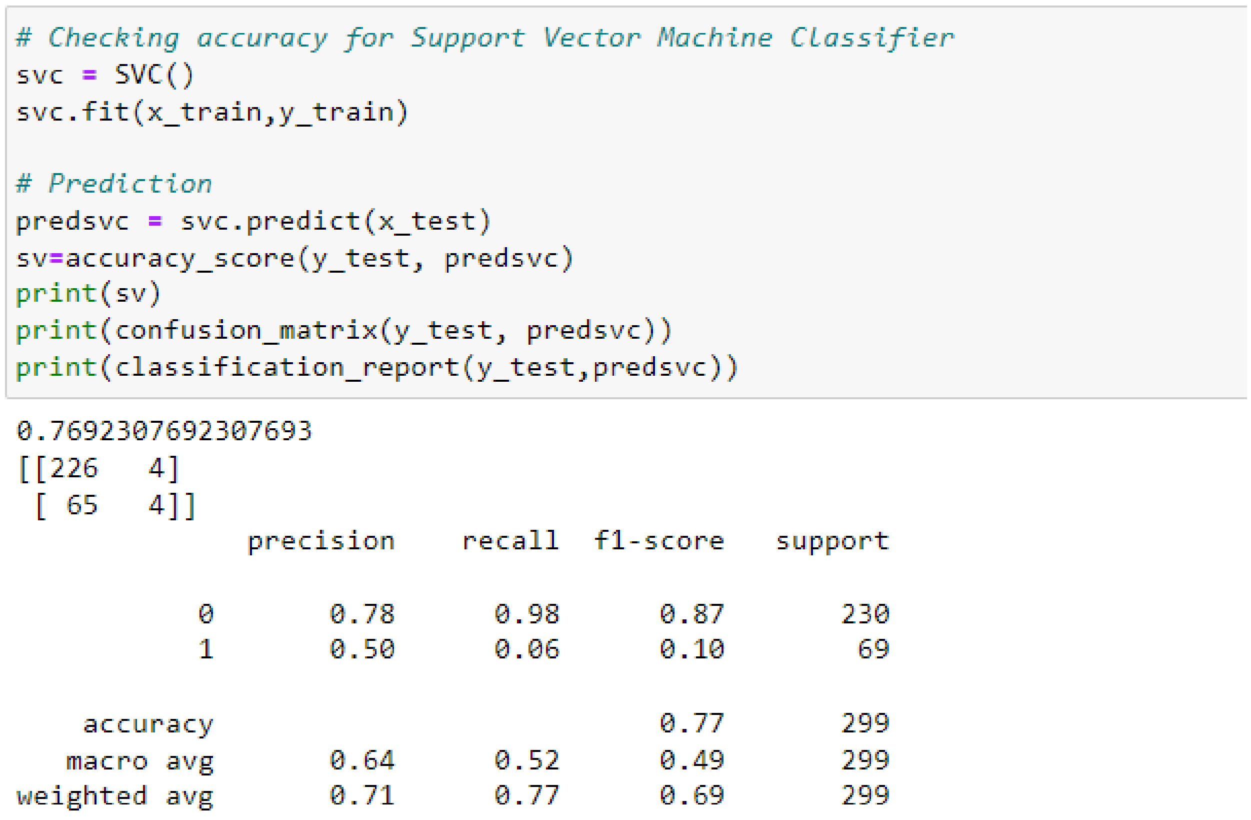
```
0.7993311036789298
[[218  12]
 [ 48  21]]
              precision    recall  f1-score   support

           0       0.82      0.95      0.88       230
           1       0.64      0.30      0.41        69

    accuracy                           0.80       299
   macro avg       0.73      0.63      0.65       299
weighted avg       0.78      0.80      0.77       299
```



```
# cv score for Random Forest Classifier
rf=cross_val_score(RFC,x,y,cv=5).mean()
print(rf)
```

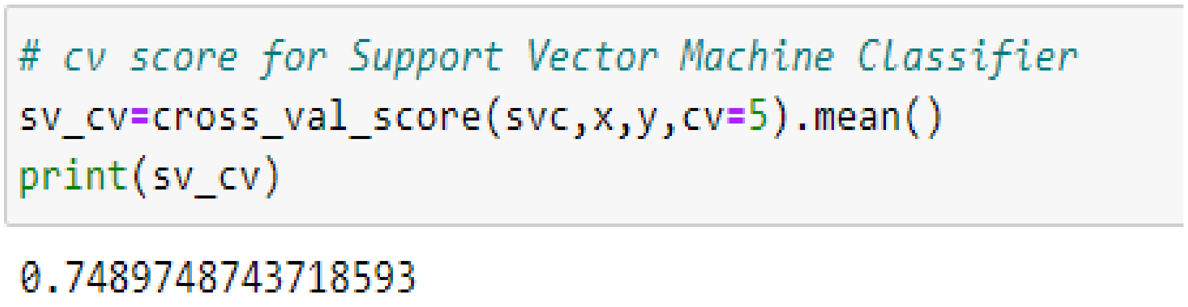
```
0.7670552763819095
```

Support Vector Machine Classifier



```
# Checking accuracy for Support Vector Machine Classifier
svc = SVC()
svc.fit(x_train,y_train)

# Prediction
predsvc = svc.predict(x_test)
sv=accuracy_score(y_test, predsvc)
print(sv)
print(confusion_matrix(y_test, predsvc))
print(classification_report(y_test,predsvc))
```

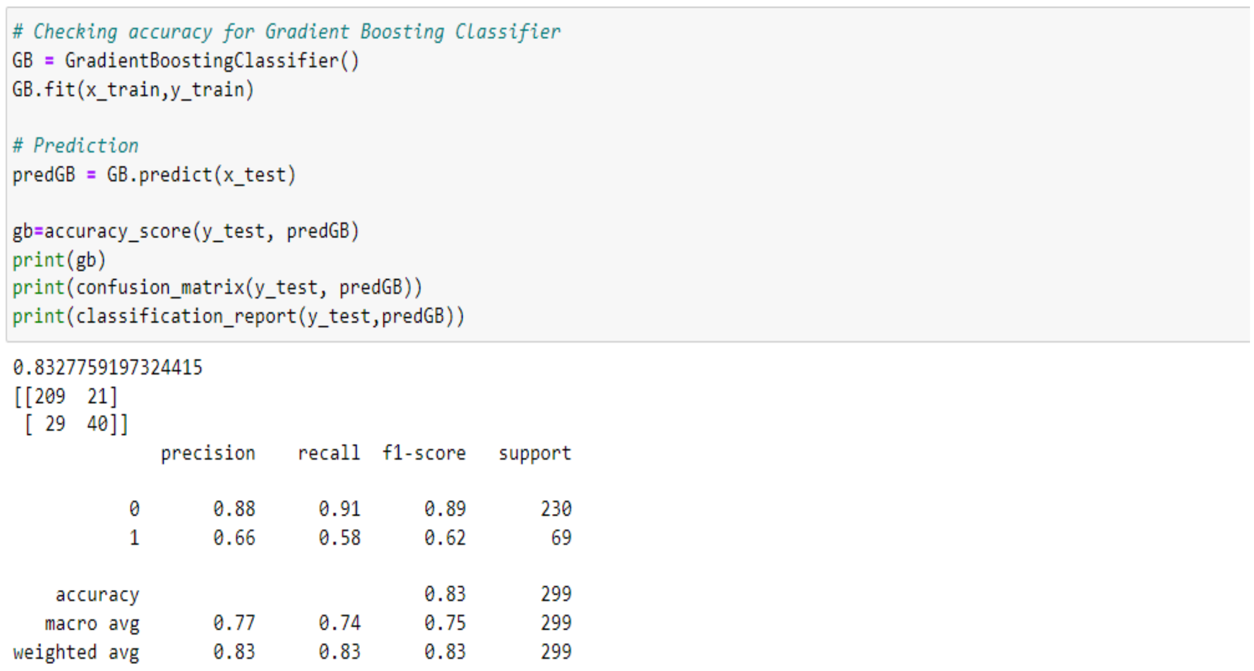
```
0.7692307692307693
[[226   4]
 [ 65   4]]
              precision    recall  f1-score   support

           0       0.78      0.98      0.87       230
           1       0.50      0.06      0.10        69

    accuracy                           0.77       299
   macro avg       0.64      0.52      0.49       299
weighted avg       0.71      0.77      0.69       299
```



```
# cv score for Support Vector Machine Classifier
sv_cv=cross_val_score(svc,x,y,cv=5).mean()
print(sv_cv)
```

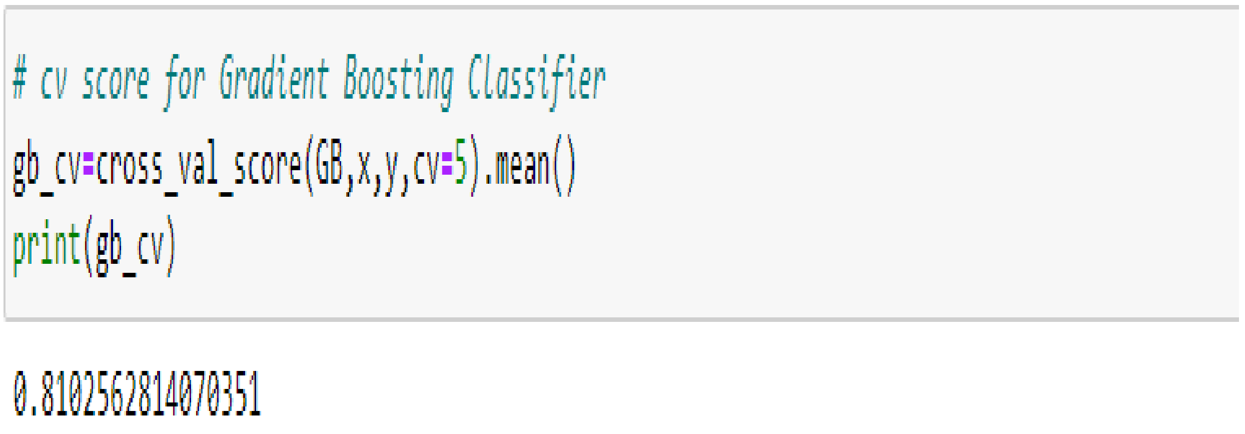
```
0.7489748743718593
```

Gradient Boosting Classifier



```
# Checking accuracy for Gradient Boosting Classifier
GB = GradientBoostingClassifier()
GB.fit(x_train,y_train)

# Prediction
predGB = GB.predict(x_test)

gb=accuracy_score(y_test, predGB)
print(gb)
print(confusion_matrix(y_test, predGB))
print(classification_report(y_test,predGB))
```

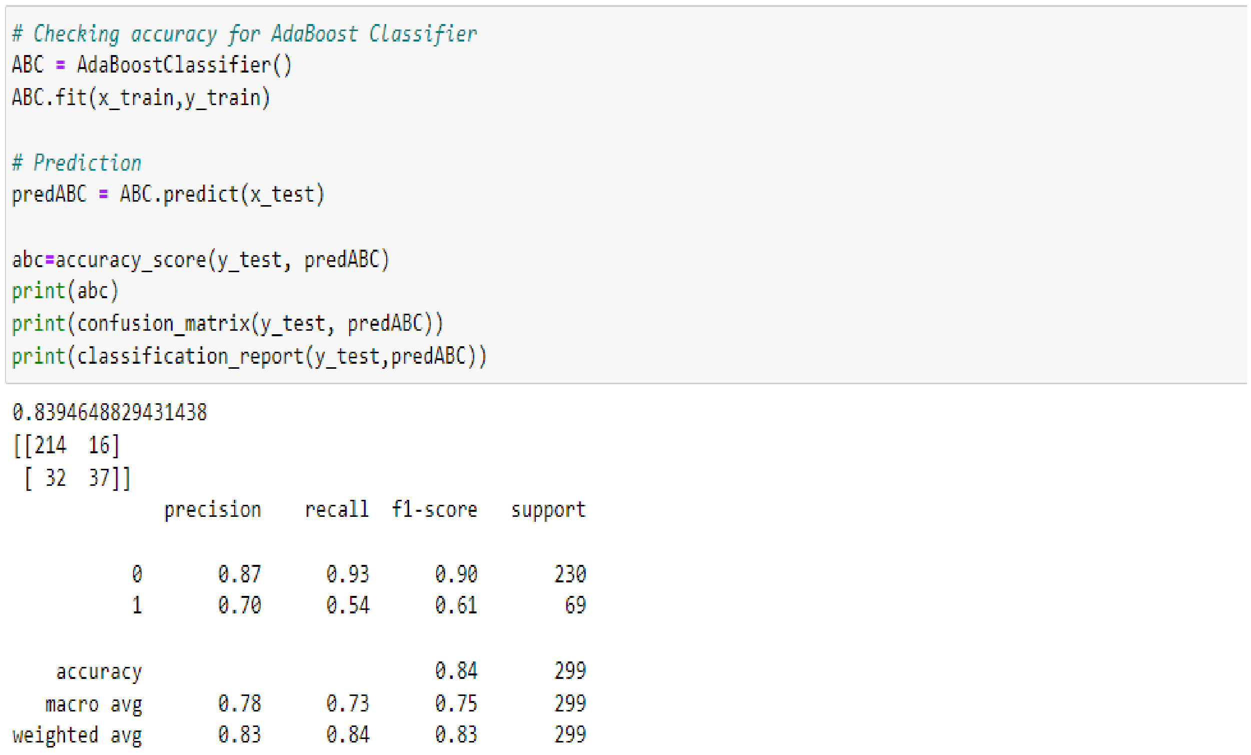
```
0.8327759197324415
[[209  21]
 [ 29  40]]
              precision    recall  f1-score   support

           0       0.88      0.91      0.89       230
           1       0.66      0.58      0.62        69

    accuracy                           0.83       299
   macro avg       0.77      0.74      0.75       299
weighted avg       0.83      0.83      0.83       299
```



```
# cv score for Gradient Boosting Classifier
gb_cv=cross_val_score(GB,x,y,cv=5).mean()
print(gb_cv)
```

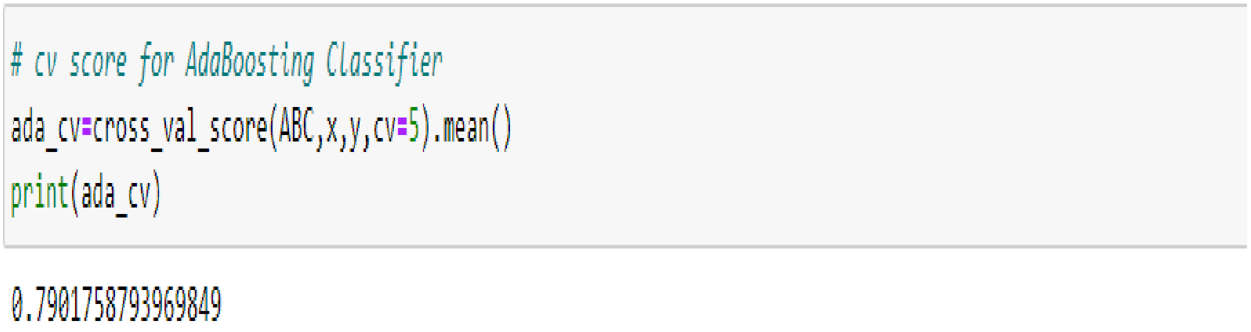
```
0.8102562814070351
```

AdaBoost Classifier



```
# Checking accuracy for AdaBoost Classifier
ABC = AdaBoostClassifier()
ABC.fit(x_train,y_train)

# Prediction
predABC = ABC.predict(x_test)

abc=accuracy_score(y_test, predABC)
print(abc)
print(confusion_matrix(y_test, predABC))
print(classification_report(y_test,predABC))
```

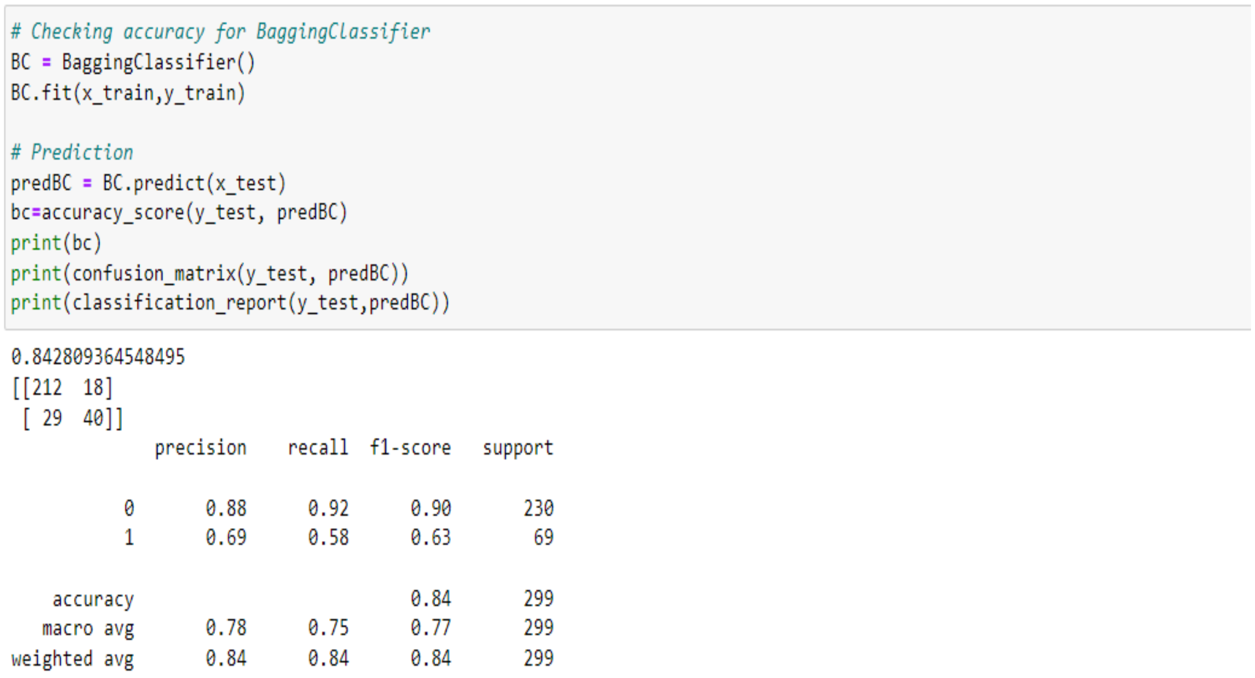
```
0.8394648829431438
[[214  16]
 [ 32  37]]
              precision    recall  f1-score   support

           0       0.87      0.93      0.90       230
           1       0.70      0.54      0.61        69

    accuracy                           0.84       299
   macro avg       0.78      0.73      0.75       299
weighted avg       0.83      0.84      0.83       299
```



```
# cv score for AdaBoosting Classifier
ada_cv=cross_val_score(ABC,x,y,cv=5).mean()
print(ada_cv)
```

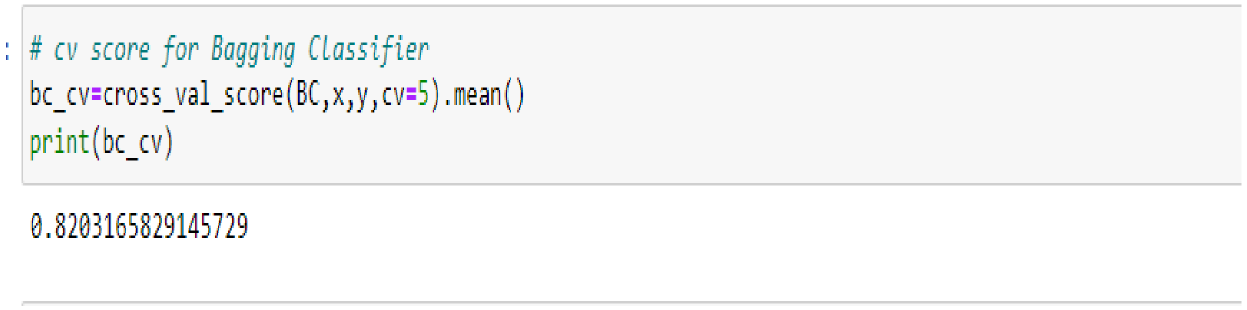
```
0.7901758793969849
```

Bagging Classifier



```
# Checking accuracy for BaggingClassifier
BC = BaggingClassifier()
BC.fit(x_train,y_train)

# Prediction
predBC = BC.predict(x_test)
bc=accuracy_score(y_test, predBC)
print(bc)
print(confusion_matrix(y_test, predBC))
print(classification_report(y_test,predBC))
```

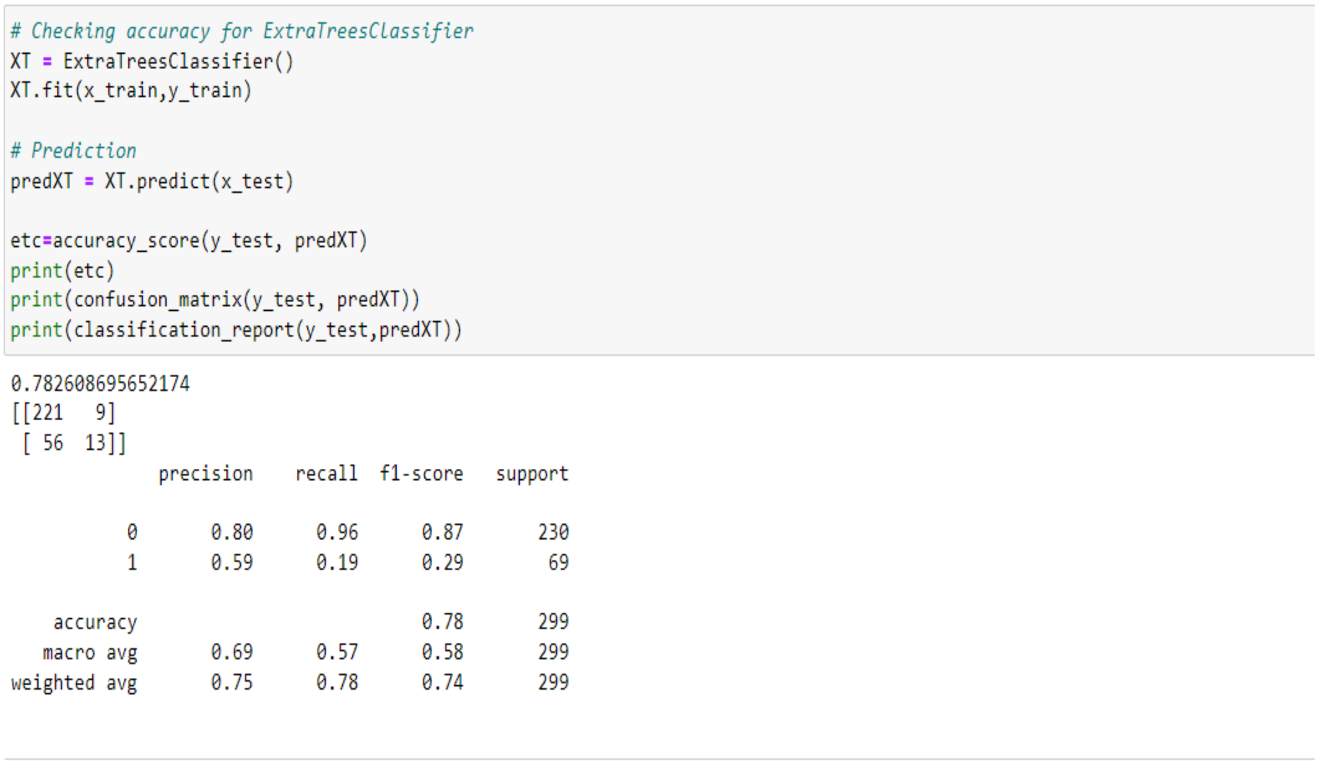
```
0.842809364548495
[[212  18]
 [ 29  40]]
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       230
           1       0.69      0.58      0.63        69

    accuracy                           0.84       299
   macro avg       0.78      0.75      0.77       299
weighted avg       0.84      0.84      0.84       299
```



```
# cv score for Bagging Classifier
bc_cv=cross_val_score(BC,x,y,cv=5).mean()
print(bc_cv)
```

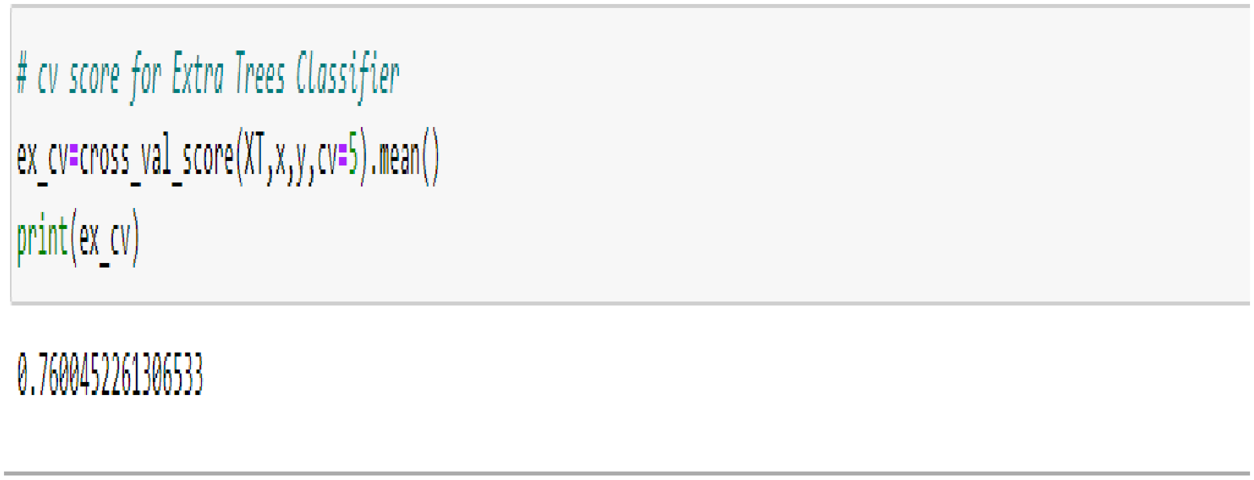
```
0.8203165829145729
```

Extra Trees Classifier



```
# Checking accuracy for ExtraTreesClassifier
XT = ExtraTreesClassifier()
XT.fit(x_train,y_train)

# Prediction
predXT = XT.predict(x_test)

etc=accuracy_score(y_test, predXT)
print(etc)
print(confusion_matrix(y_test, predXT))
print(classification_report(y_test,predXT))
```

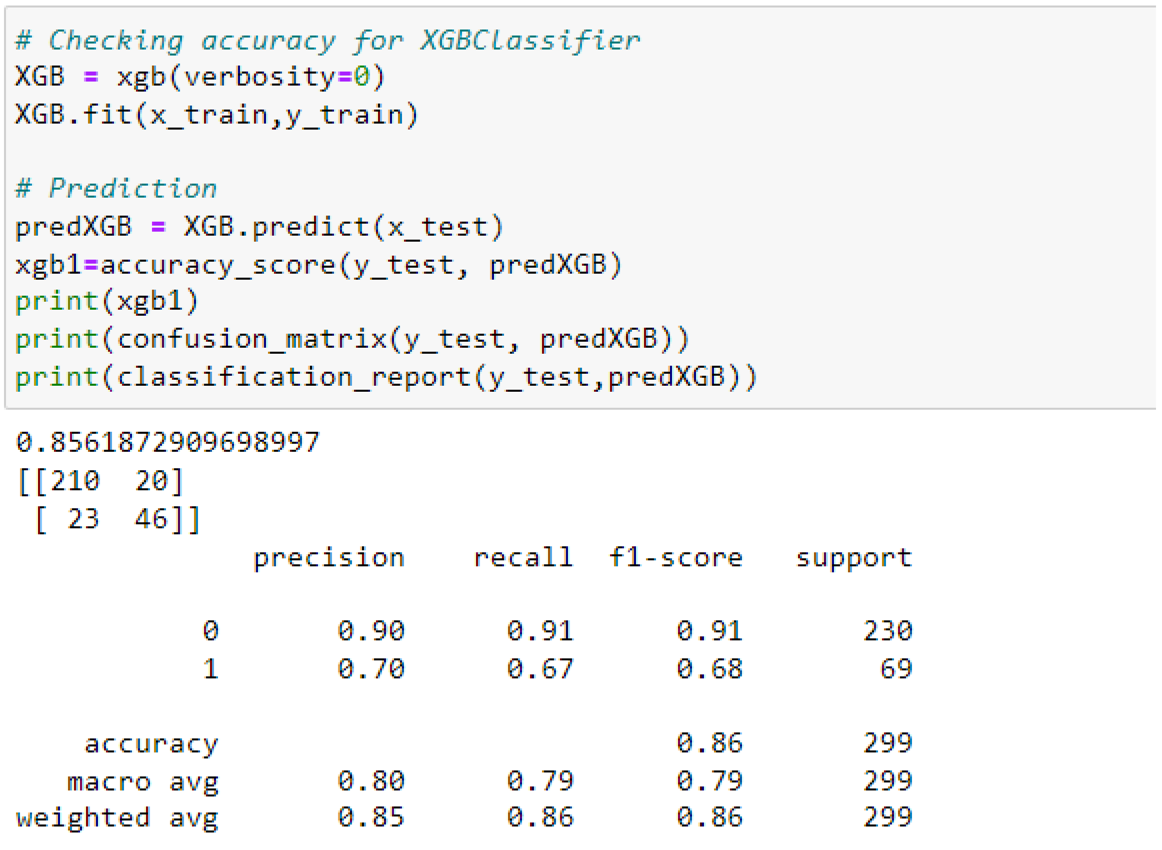
```
0.782608695652174
[[221   9]
 [ 56  13]]
              precision    recall  f1-score   support

           0       0.80      0.96      0.87       230
           1       0.59      0.19      0.29        69

    accuracy                           0.78       299
   macro avg       0.69      0.57      0.58       299
weighted avg       0.75      0.78      0.74       299
```



```
# cv score for Extra Trees Classifier
ex_cv=cross_val_score(XT,x,y,cv=5).mean()
print(ex_cv)
```

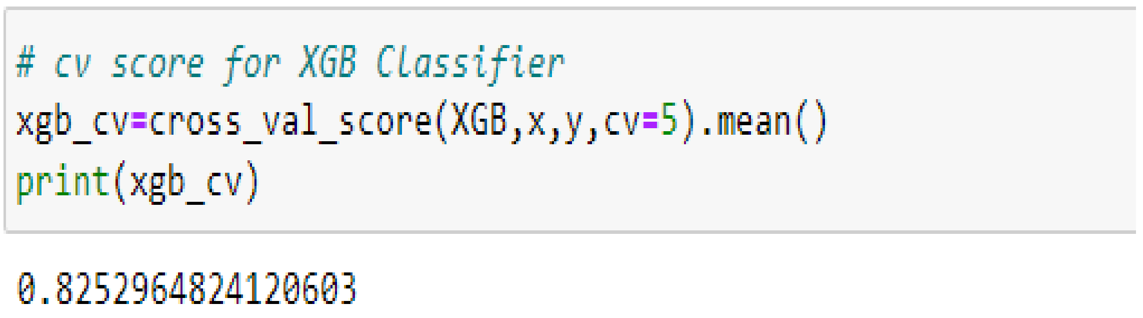
```
0.7600452261306533
```

Extreme Gradient Boosting Classifier (XGB)



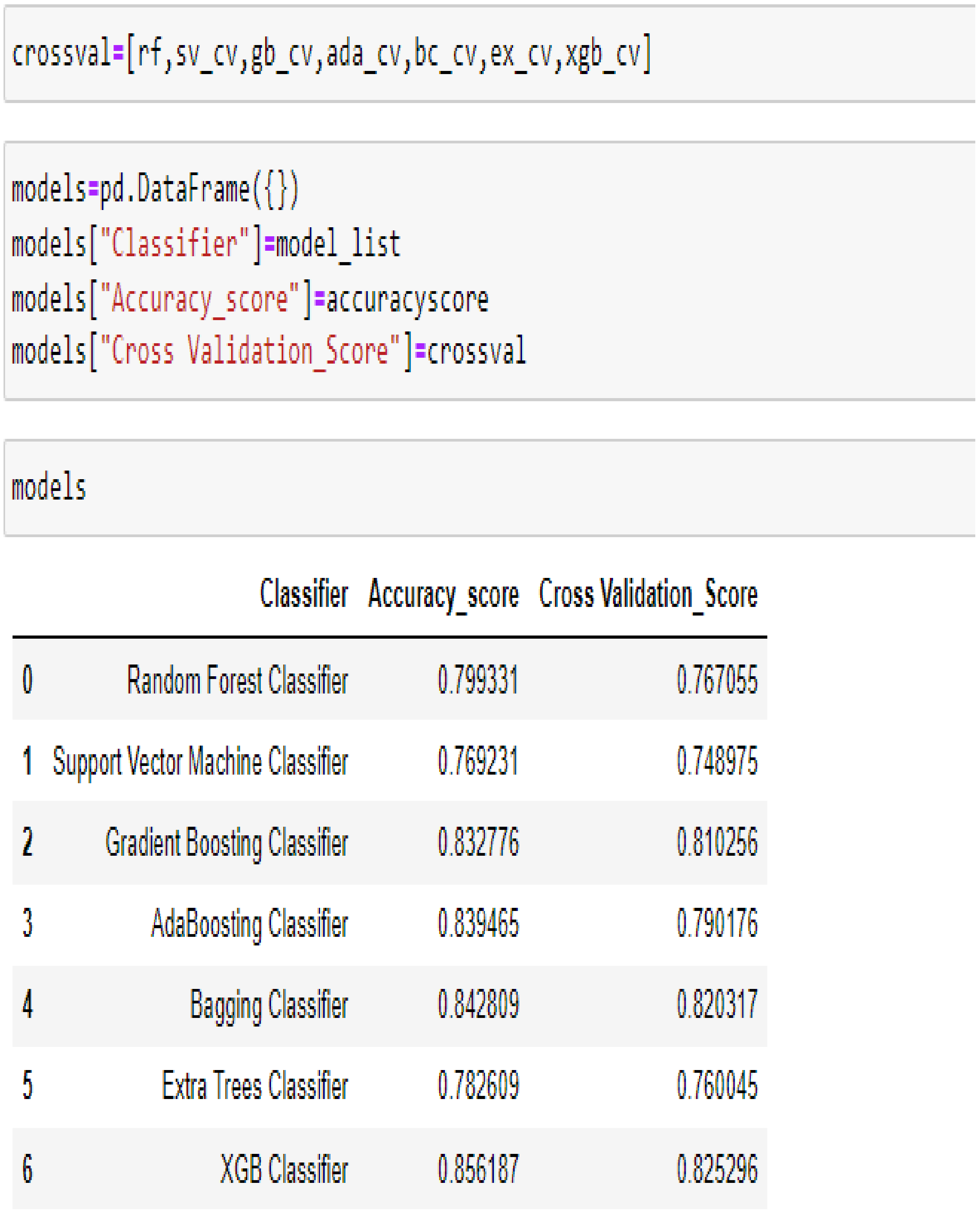
```
# Checking accuracy for XGBClassifier
XGB = xgb(verbosity=0)
XGB.fit(x_train,y_train)

# Prediction
predXGB = XGB.predict(x_test)
xgb1=accuracy_score(y_test, predXGB)
print(xgb1)
print(confusion_matrix(y_test, predXGB))
print(classification_report(y_test,predXGB))
```

```
0.8561872909698997
[[210  20]
 [ 23  46]]
              precision    recall  f1-score   support

           0       0.90      0.91      0.91       230
           1       0.70      0.67      0.68        69

    accuracy                           0.86       299
   macro avg       0.80      0.79      0.79       299
weighted avg       0.85      0.86      0.86       299
```



```
# cv score for XGB Classifier
xgb_cv=cross_val_score(XGB,x,y,cv=5).mean()
print(xgb_cv)
```

```
0.8252964824120603
```



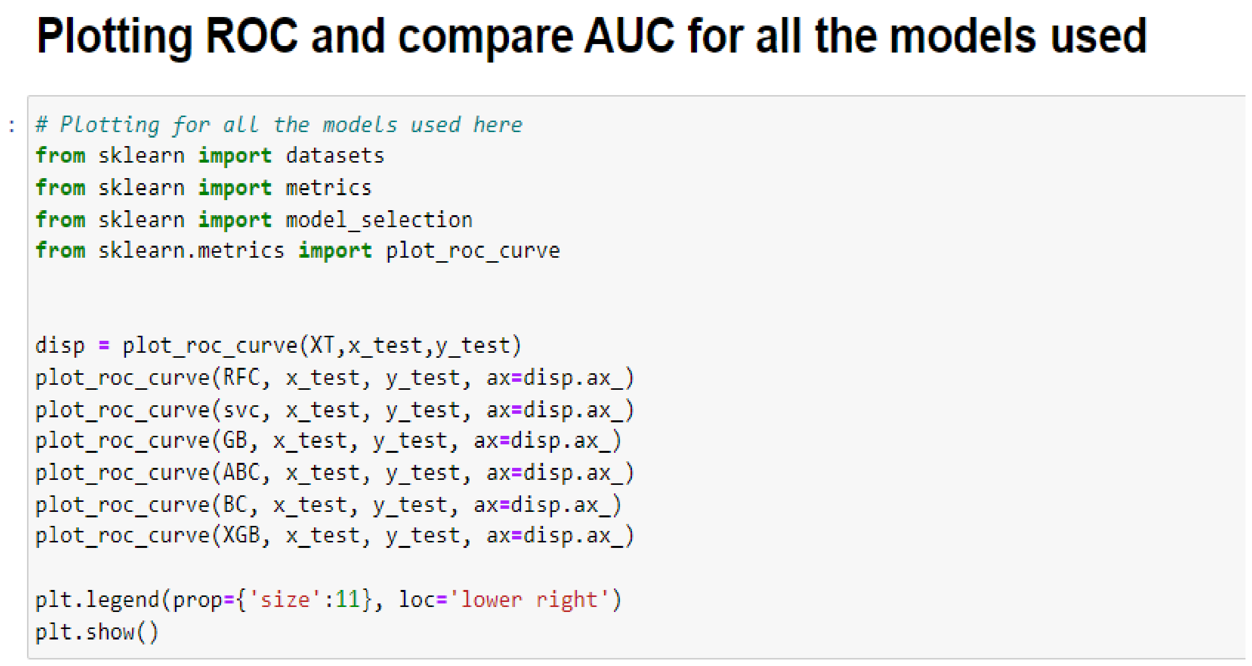
```
crossval=[rf,sv_cv,gb_cv,ada_cv,bc_cv,ex_cv,xgb_cv]
```

```
models=pd.DataFrame({})
models["Classifier"]=model_list
models["Accuracy_score"]=accuracyscore
models["Cross Validation_Score"]=crossval
```

```
models
```

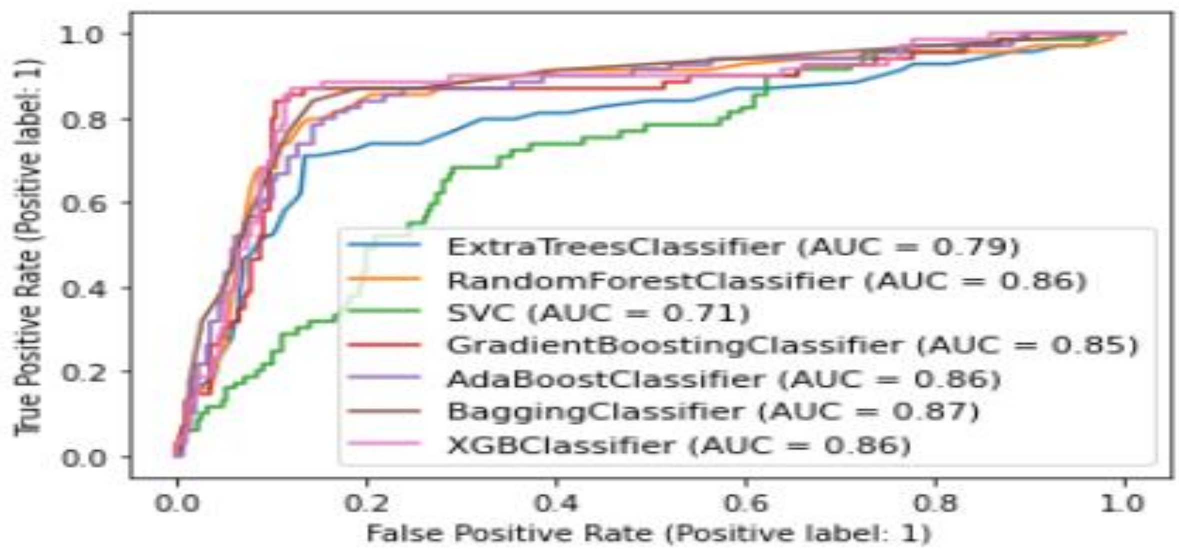
	Classifier	Accuracy_score	Cross Validation_Score
0	Random Forest Classifier	0.799331	0.767055
1	Support Vector Machine Classifier	0.769231	0.748975
2	Gradient Boosting Classifier	0.832776	0.810256
3	AdaBoosting Classifier	0.839465	0.790176
4	Bagging Classifier	0.842809	0.820317
5	Extra Trees Classifier	0.782609	0.760045
6	XGB Classifier	0.856187	0.825296

ROC-AUC curve for all the models

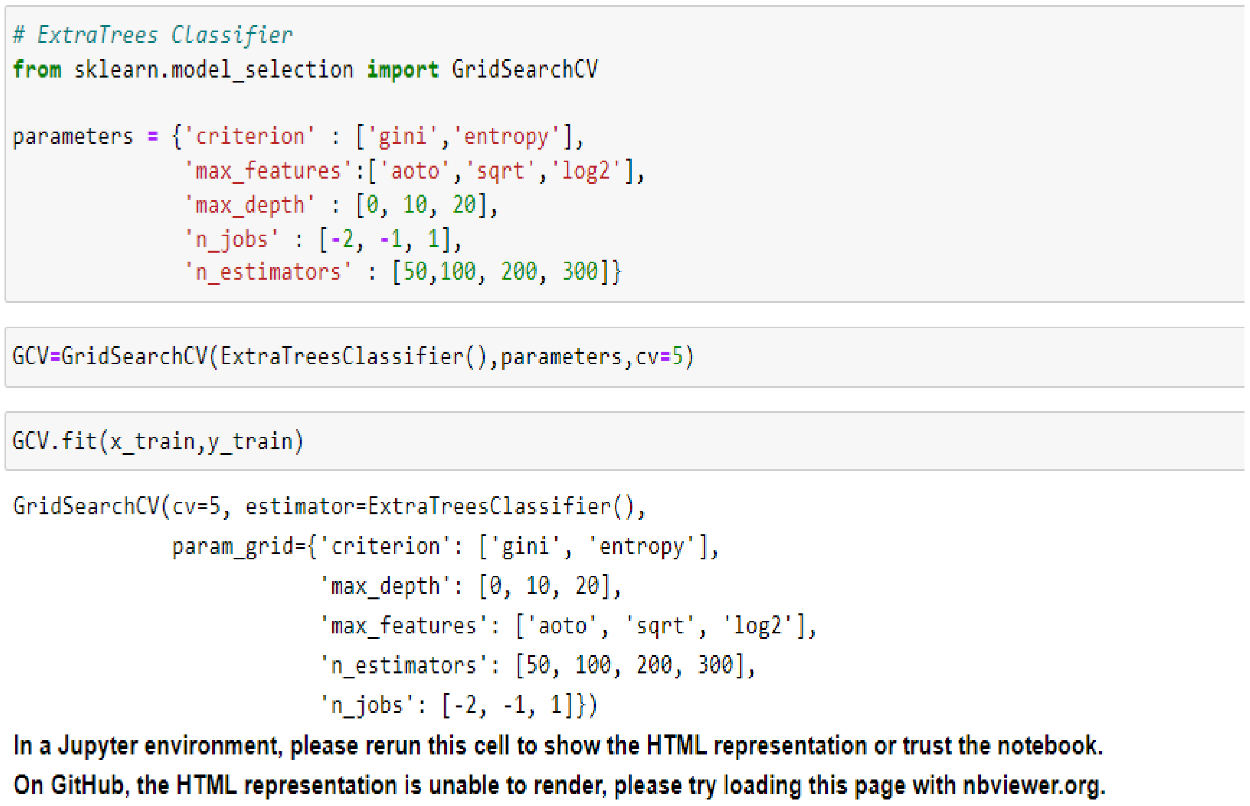


Plotting ROC and compare AUC for all the models used

```
: # Plotting for all the models used here
  from sklearn import datasets
  from sklearn import metrics
  from sklearn import model_selection
  from sklearn.metrics import plot_roc_curve


  disp = plot_roc_curve(XT,x_test,y_test)
  plot_roc_curve(RFC, x_test, y_test, ax=disp.ax_)
  plot_roc_curve(svc, x_test, y_test, ax=disp.ax_)
  plot_roc_curve(GB, x_test, y_test, ax=disp.ax_)
  plot_roc_curve(ABC, x_test, y_test, ax=disp.ax_)
  plot_roc_curve(BC, x_test, y_test, ax=disp.ax_)
  plot_roc_curve(XGB, x_test, y_test, ax=disp.ax_)

  plt.legend(prop={'size':11}, loc='lower right')
  plt.show()
```



Hyper Parameter Tuning



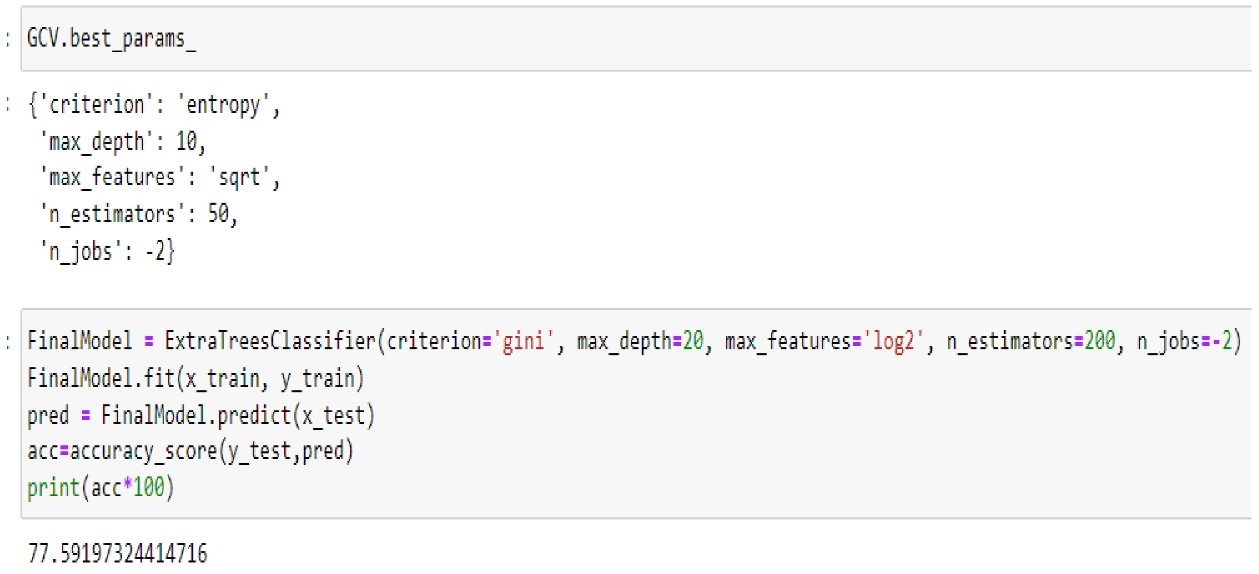
```
# ExtraTrees Classifier
from sklearn.model_selection import GridSearchCV

parameters = {'criterion' : ['gini','entropy'],
              'max_features':['aoto','sqrt','log2'],
              'max_depth' : [0, 10, 20],
              'n_jobs' : [-2, -1, 1],
              'n_estimators' : [50,100, 200, 300]}
```

```
GCV=GridSearchCV(ExtraTreesClassifier(),parameters,cv=5)
```

```
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [0, 10, 20],
                         'max_features': ['aoto', 'sqrt', 'log2'],
                         'n_estimators': [50, 100, 200, 300],
                         'n_jobs': [-2, -1, 1]})
```

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



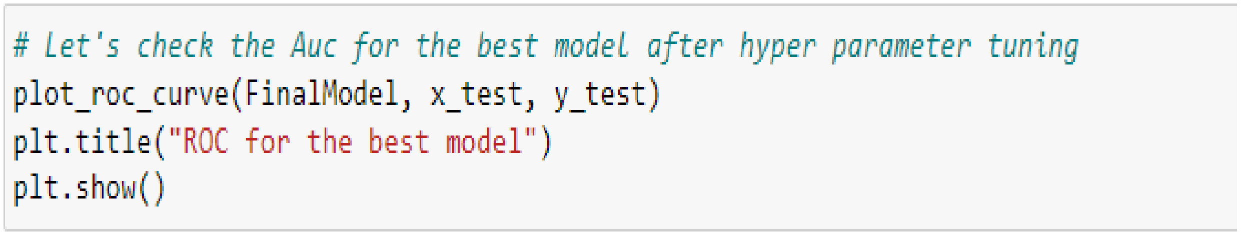
```
GCV.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 'sqrt',
 'n_estimators': 50,
 'n_jobs': -2}
```

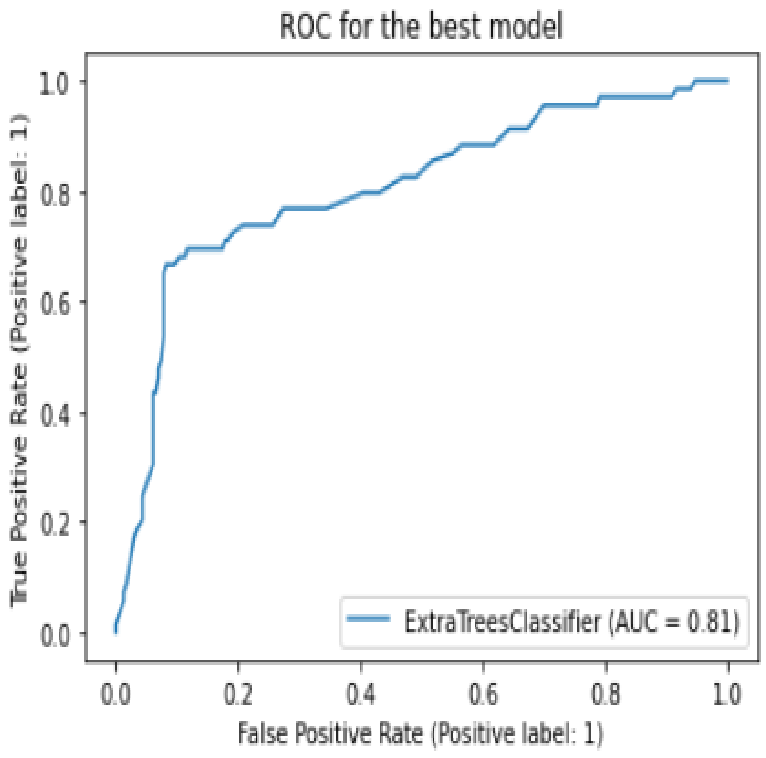
```
FinalModel = ExtraTreesClassifier(criterion='gini', max_depth=20, max_features='log2', n_estimators=200, n_jobs=-2)
FinalModel.fit(x_train, y_train)
pred = FinalModel.predict(x_test)
acc=accuracy_score(y_test,pred)
print(acc*100)
```

```
77.59197324414716
```

Now we will plot ROC curve and compare the AUC for the best model



```
# Let's check the Auc for the best model after hyper parameter tuning
plot_roc_curve(FinalModel, x_test, y_test)
plt.title("ROC for the best model")
plt.show()
```



The AUC value is also remained same as before. Since I have done with the model building, it's time to save the model. Here I have saved my model as below

```python
# Saving the model using .pkl
import joblib
joblib.dump(FinalModel,"InsuranceclaimFraudDetection.pkl")
```

```
['InsuranceclaimFraudDetection.pkl']
```

Predicting the saved model

```python
# Let's load the saved model and get the prediction

# Loading the saved model
model=joblib.load("InsuranceclaimFraudDetection.pkl")

#Prediction
prediction = model.predict(x_test)
prediction
```

```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
```

```python
pd.DataFrame([model.predict(x_test)[:],y_test[:]],index=["Predicted","Original"]).T
```

	Predicted	Original
0	0	1
1	1	1
2	0	0
3	0	1
4	0	0
...	...	...
294	0	0
295	0	0
296	0	0
297	0	1
298	0	0

299 rows × 2 columns

Concluding Remark

In this project we have gone through the feature engineering which is the most crucial thing and removed the outliers and skewness. Also handled the categorical columns by encoding the data, scaled the data and at last, we built different classification models to predict whether the insurance claim is fraudulent or not and performed the hyper tuning to improve the model by using different parameters.