|  |  |
| --- | --- |
| DATE | 27/10/2023 |
| Team ID | 3864 |
| Project Title | Credit Card Fraud Detection |

# PROGRAM:

# Importing Libraries

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import PowerTransformer  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import classification\_report  
from sklearn.linear\_model import LogisticRegression  
from sklearn.ensemble import RandomForestClassifier  
import xgboost as xgb

from google.colab import drive  
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

# READING DATA

df = pd.read\_csv("/content/drive/MyDrive/creditcard.csv")  
df.head()

Time V1 V2 V3 V4 V5 V6 V7 \  
0 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599   
1 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803   
2 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461   
3 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609   
4 2.0 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941   
  
 V8 V9 ... V21 V22 V23 V24 V25 \  
0 0.098698 0.363787 ... -0.018307 0.277838 -0.110474 0.066928 0.128539   
1 0.085102 -0.255425 ... -0.225775 -0.638672 0.101288 -0.339846 0.167170   
2 0.247676 -1.514654 ... 0.247998 0.771679 0.909412 -0.689281 -0.327642   
3 0.377436 -1.387024 ... -0.108300 0.005274 -0.190321 -1.175575 0.647376   
4 -0.270533 0.817739 ... -0.009431 0.798278 -0.137458 0.141267 -0.206010   
  
 V26 V27 V28 Amount Class   
0 -0.189115 0.133558 -0.021053 149.62 0   
1 0.125895 -0.008983 0.014724 2.69 0   
2 -0.139097 -0.055353 -0.059752 378.66 0   
3 -0.221929 0.062723 0.061458 123.50 0   
4 0.502292 0.219422 0.215153 69.99 0   
  
[5 rows x 31 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 284807 entries, 0 to 284806  
Data columns (total 31 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Time 284807 non-null float64  
 1 V1 284807 non-null float64  
 2 V2 284807 non-null float64  
 3 V3 284807 non-null float64  
 4 V4 284807 non-null float64  
 5 V5 284807 non-null float64  
 6 V6 284807 non-null float64  
 7 V7 284807 non-null float64  
 8 V8 284807 non-null float64  
 9 V9 284807 non-null float64  
 10 V10 284807 non-null float64  
 11 V11 284807 non-null float64  
 12 V12 284807 non-null float64  
 13 V13 284807 non-null float64  
 14 V14 284807 non-null float64  
 15 V15 284807 non-null float64  
 16 V16 284807 non-null float64  
 17 V17 284807 non-null float64  
 18 V18 284807 non-null float64  
 19 V19 284807 non-null float64  
 20 V20 284807 non-null float64  
 21 V21 284807 non-null float64  
 22 V22 284807 non-null float64  
 23 V23 284807 non-null float64  
 24 V24 284807 non-null float64  
 25 V25 284807 non-null float64  
 26 V26 284807 non-null float64  
 27 V27 284807 non-null float64  
 28 V28 284807 non-null float64  
 29 Amount 284807 non-null float64  
 30 Class 284807 non-null int64   
dtypes: float64(30), int64(1)  
memory usage: 67.4 MB

df.isna().sum()

Time 0  
V1 0  
V2 0  
V3 0  
V4 0  
V5 0  
V6 0  
V7 0  
V8 0  
V9 0  
V10 0  
V11 0  
V12 0  
V13 0  
V14 0  
V15 0  
V16 0  
V17 0  
V18 0  
V19 0  
V20 0  
V21 0  
V22 0  
V23 0  
V24 0  
V25 0  
V26 0  
V27 0  
V28 0  
Amount 0  
Class 0  
dtype: int64

df['Class'].unique()

array([0, 1])

df.describe().T

count mean std min 25% \  
Time 284807.0 9.481386e+04 47488.145955 0.000000 54201.500000   
V1 284807.0 1.168375e-15 1.958696 -56.407510 -0.920373   
V2 284807.0 3.416908e-16 1.651309 -72.715728 -0.598550   
V3 284807.0 -1.379537e-15 1.516255 -48.325589 -0.890365   
V4 284807.0 2.074095e-15 1.415869 -5.683171 -0.848640   
V5 284807.0 9.604066e-16 1.380247 -113.743307 -0.691597   
V6 284807.0 1.487313e-15 1.332271 -26.160506 -0.768296   
V7 284807.0 -5.556467e-16 1.237094 -43.557242 -0.554076   
V8 284807.0 1.213481e-16 1.194353 -73.216718 -0.208630   
V9 284807.0 -2.406331e-15 1.098632 -13.434066 -0.643098   
V10 284807.0 2.239053e-15 1.088850 -24.588262 -0.535426   
V11 284807.0 1.673327e-15 1.020713 -4.797473 -0.762494   
V12 284807.0 -1.247012e-15 0.999201 -18.683715 -0.405571   
V13 284807.0 8.190001e-16 0.995274 -5.791881 -0.648539   
V14 284807.0 1.207294e-15 0.958596 -19.214325 -0.425574   
V15 284807.0 4.887456e-15 0.915316 -4.498945 -0.582884   
V16 284807.0 1.437716e-15 0.876253 -14.129855 -0.468037   
V17 284807.0 -3.772171e-16 0.849337 -25.162799 -0.483748   
V18 284807.0 9.564149e-16 0.838176 -9.498746 -0.498850   
V19 284807.0 1.039917e-15 0.814041 -7.213527 -0.456299   
V20 284807.0 6.406204e-16 0.770925 -54.497720 -0.211721   
V21 284807.0 1.654067e-16 0.734524 -34.830382 -0.228395   
V22 284807.0 -3.568593e-16 0.725702 -10.933144 -0.542350   
V23 284807.0 2.578648e-16 0.624460 -44.807735 -0.161846   
V24 284807.0 4.473266e-15 0.605647 -2.836627 -0.354586   
V25 284807.0 5.340915e-16 0.521278 -10.295397 -0.317145   
V26 284807.0 1.683437e-15 0.482227 -2.604551 -0.326984   
V27 284807.0 -3.660091e-16 0.403632 -22.565679 -0.070840   
V28 284807.0 -1.227390e-16 0.330083 -15.430084 -0.052960   
Amount 284807.0 8.834962e+01 250.120109 0.000000 5.600000   
Class 284807.0 1.727486e-03 0.041527 0.000000 0.000000   
  
 50% 75% max   
Time 84692.000000 139320.500000 172792.000000   
V1 0.018109 1.315642 2.454930   
V2 0.065486 0.803724 22.057729   
V3 0.179846 1.027196 9.382558   
V4 -0.019847 0.743341 16.875344   
V5 -0.054336 0.611926 34.801666   
V6 -0.274187 0.398565 73.301626   
V7 0.040103 0.570436 120.589494   
V8 0.022358 0.327346 20.007208   
V9 -0.051429 0.597139 15.594995   
V10 -0.092917 0.453923 23.745136   
V11 -0.032757 0.739593 12.018913   
V12 0.140033 0.618238 7.848392   
V13 -0.013568 0.662505 7.126883   
V14 0.050601 0.493150 10.526766   
V15 0.048072 0.648821 8.877742   
V16 0.066413 0.523296 17.315112   
V17 -0.065676 0.399675 9.253526   
V18 -0.003636 0.500807 5.041069   
V19 0.003735 0.458949 5.591971   
V20 -0.062481 0.133041 39.420904   
V21 -0.029450 0.186377 27.202839   
V22 0.006782 0.528554 10.503090   
V23 -0.011193 0.147642 22.528412   
V24 0.040976 0.439527 4.584549   
V25 0.016594 0.350716 7.519589   
V26 -0.052139 0.240952 3.517346   
V27 0.001342 0.091045 31.612198   
V28 0.011244 0.078280 33.847808   
Amount 22.000000 77.165000 25691.160000   
Class 0.000000 0.000000 1.000000

**FEATURE ENGINEERING**

* Creating hour column in our dataset by dividing time column which shows time in seconds to 3600.

df['hour']= round(df['Time']/3600)  
df.head()

Time V1 V2 V3 V4 V5 V6 V7 \  
0 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599   
1 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803   
2 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461   
3 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609   
4 2.0 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941   
  
 V8 V9 ... V22 V23 V24 V25 V26 \  
0 0.098698 0.363787 ... 0.277838 -0.110474 0.066928 0.128539 -0.189115   
1 0.085102 -0.255425 ... -0.638672 0.101288 -0.339846 0.167170 0.125895   
2 0.247676 -1.514654 ... 0.771679 0.909412 -0.689281 -0.327642 -0.139097   
3 0.377436 -1.387024 ... 0.005274 -0.190321 -1.175575 0.647376 -0.221929   
4 -0.270533 0.817739 ... 0.798278 -0.137458 0.141267 -0.206010 0.502292   
  
 V27 V28 Amount Class hour   
0 0.133558 -0.021053 149.62 0 0.0   
1 -0.008983 0.014724 2.69 0 0.0   
2 -0.055353 -0.059752 378.66 0 0.0   
3 0.062723 0.061458 123.50 0 0.0   
4 0.219422 0.215153 69.99 0 0.0   
  
[5 rows x 32 columns]

class\_count = df['Class'].value\_counts()  
class\_count

0 284315  
1 492  
Name: Class, dtype: int64

duplicated\_count = df.duplicated().sum()  
duplicated\_count

1081

df = df.drop\_duplicates(keep='first')

fig = plt.figure(figsize=(10, 7))  
labels = class\_count.index  
data = class\_count.values  
  
plt.pie(data, labels=labels)  
plt.title('Class Distribution')  
plt.axis('equal')  
plt.show()

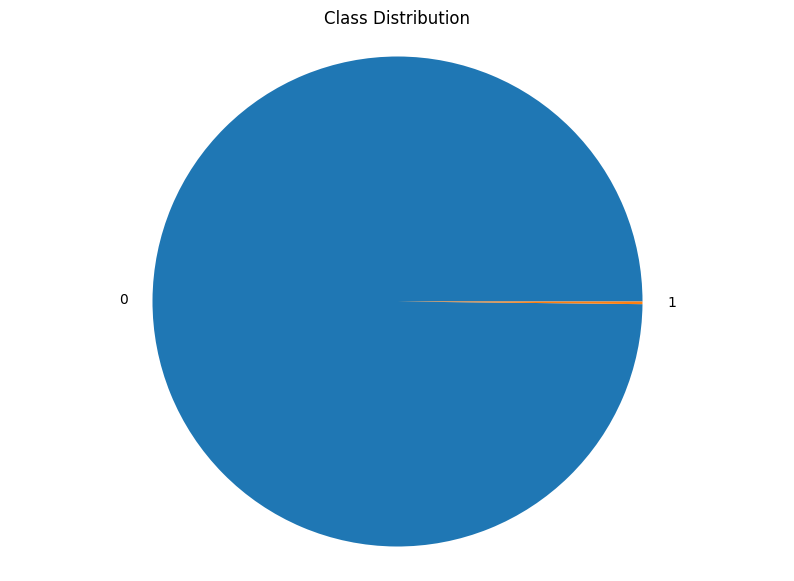


FIG 1: Class Distribution

Credit card fraud detection is a challenging problem due to the imbalanced class distribution, with a very small minority of transactions being fraudulent. Class balancing techniques, such as oversampling or undersampling, can be used to address this issue.

plt.figure(figsize=(10, 5))  
sns.countplot(data = df, x='Class')  
plt.title('Class Distribution (Count)')  
plt.xlabel('Class')  
plt.ylabel('Count')  
plt.show()

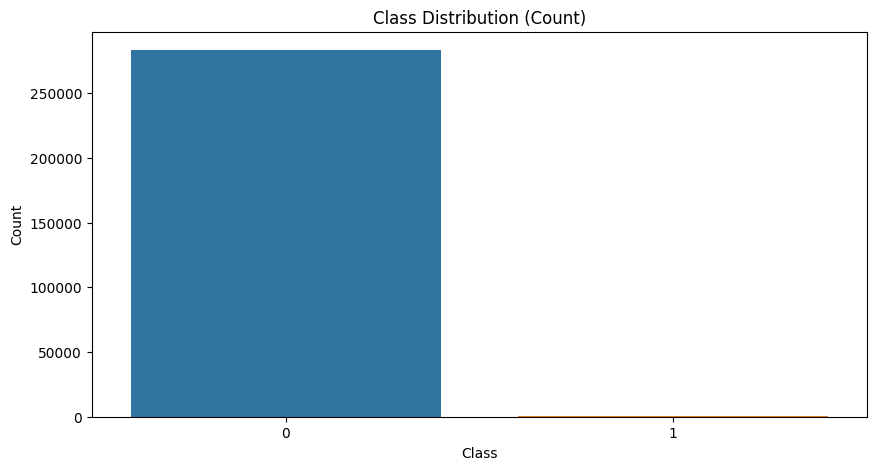


FIG 2: Class distribution(count)

Credit card fraud detection class distribution is typically imbalanced, with ~99.8% legitimate transactions and ~0.2% fraudulent transactions. This can make it difficult for machine learning models to detect fraud. Class balancing techniques, such as oversampling or undersampling, can be used to address this issue.

sns.boxplot(df['Amount'])

<Axes: >

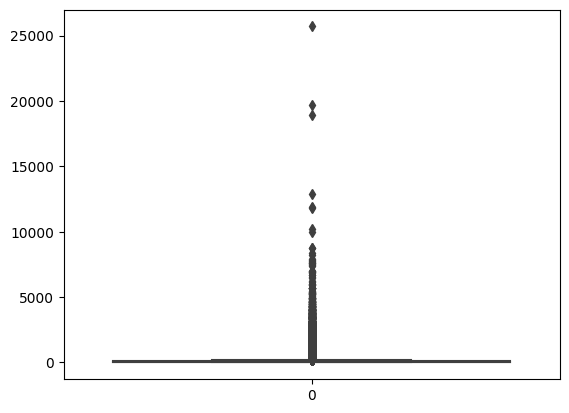


FIG 3:Count Graph

The credit card fraud detection class distribution count graph is a bar chart that shows the number of transactions in each class. The two classes are legitimate and fraudulent. The graph shows that the majority of transactions are legitimate, while only a small minority are fraudulent. This imbalance can make it difficult for machine learning models to detect fraudulent transactions.

sns.displot(df['Amount'])  
plt.show()

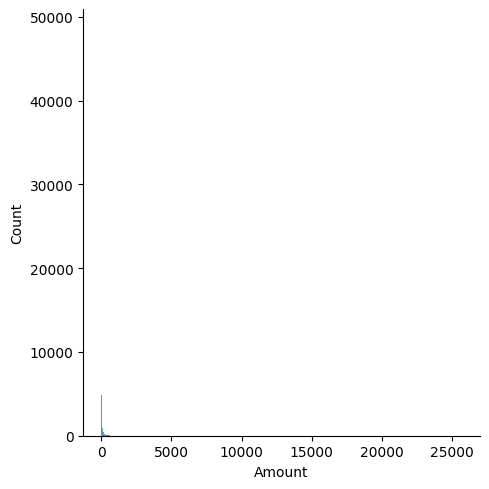


FIG 4: Distribution Graph

A correlation graph is a visualization that shows the relationship between two quantitative variables. It is typically a scatter plot, with the values of one variable on the horizontal axis and the values of the other variable on the vertical axis. Each point on the graph represents a single data point.

df['Amount'].skew()

16.978803370060476

**OBSERVATIONS**

* Data does not contains any NULL values but it does contains Duplicate Values.
* Data possesed very high positive skewness.
* Data is imbalanced.
* Data possesed Outliers.

pt = PowerTransformer(method='yeo-johnson')  
df['Amount'] = pt.fit\_transform(df[['Amount']])  
df['Amount'].skew()

0.018234140699062654

sns.distplot(df['Amount'])  
plt.show()

<ipython-input-23-ae7ad042f4b6>:1: UserWarning:   
  
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.  
  
Please adapt your code to use either `displot` (a figure-level function with  
similar flexibility) or `histplot` (an axes-level function for histograms).  
  
For a guide to updating your code to use the new functions, please see  
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751  
  
 sns.distplot(df['Amount'])

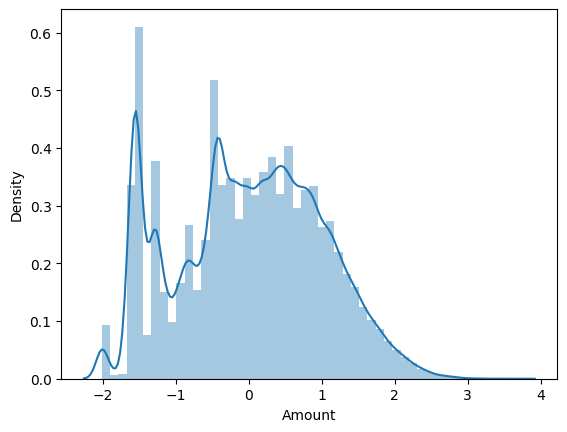


FIG 5:Distribution of time features

The strength of the correlation between the two variables can be inferred by the pattern of the points on the graph. If the points form a tight cluster, then there is a strong correlation between the two variables. If the points are spread out randomly, then there is a weak correlation between the two variables.

scaler = StandardScaler()  
  
df['Amount'] = scaler.fit\_transform(df[['Amount']])  
  
sns.boxplot(df['Amount'])

<Axes: >

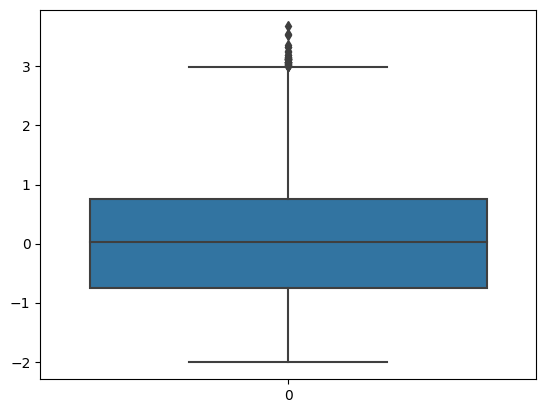


FIG 6: Cummulative Graph

The direction of the correlation can also be inferred by the pattern of the points on the graph. If the points slope upwards from left to right, then there is a positive correlation between the two variables. This means that as the value of one variable increases, the value of the other variable also tends to increase.

outliers = df['Amount'] > 3  
outliers.count()

283726

df['Amount']=df['Amount'] < 3  
class\_count

0 284315  
1 492  
Name: Class, dtype: int64

sns.heatmap(df.corr())  
plt.show()

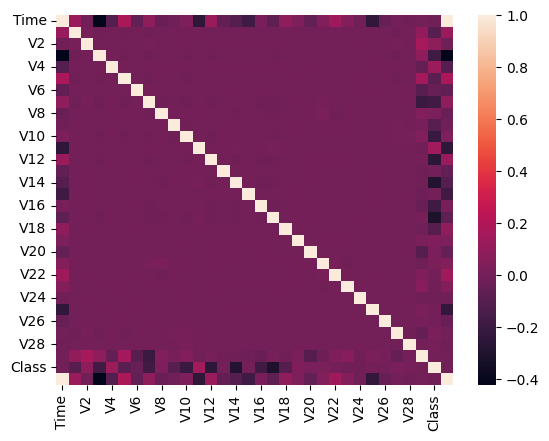


FIG 7:Correletion Graph

A scatter plot with two axes: x-axis for transaction amount and y-axis for fraud probability.Each point on the graph represents a single credit card transaction.A strong, positive correlation is shown by a tight cluster of points sloping upwards from left to right.This means that as the transaction amount increases, the probability of fraud also increases.This graph can be used to identify fraudulent transactions by identifying transactions with high transaction amounts and high fraud probabilities.

df.columns

Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',  
 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',  
 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',  
 'Class', 'hour'],  
 dtype='object')

x = df.drop(columns = ['Class'])  
y = df['Class']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, train\_size=0.8,random\_state =101, stratify =y)

# 

# LOGISTIC REGRESSION

model = LogisticRegression(class\_weight = 'balanced')  
  
model.fit(x\_train,y\_train)

LogisticRegression(class\_weight='balanced')

train\_pred = model.predict(x\_train)

print(classification\_report(y\_train,train\_pred))

precision recall f1-score support  
  
 0 1.00 0.96 0.98 226602  
 1 0.03 0.88 0.07 378  
  
 accuracy 0.96 226980  
 macro avg 0.52 0.92 0.52 226980  
weighted avg 1.00 0.96 0.98 226980

test\_pred = model.predict(x\_test)

print(classification\_report(y\_test,test\_pred))

precision recall f1-score support  
  
 0 1.00 0.96 0.98 56651  
 1 0.03 0.84 0.06 95  
  
 accuracy 0.96 56746  
 macro avg 0.52 0.90 0.52 56746  
weighted avg 1.00 0.96 0.98 56746

# Random Forest Classifier

model\_r = RandomForestClassifier(class\_weight = 'balanced')

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

RandomForestClassifier(class\_weight='balanced')

trainpred = model\_r.predict(x\_train)  
print(classification\_report(y\_train,trainpred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 226602  
 1 1.00 1.00 1.00 378  
  
 accuracy 1.00 226980  
 macro avg 1.00 1.00 1.00 226980  
weighted avg 1.00 1.00 1.00 226980

testpred = model\_r.predict(x\_test)  
  
print(classification\_report(y\_test,testpred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 56651  
 1 0.96 0.71 0.81 95  
  
 accuracy 1.00 56746  
 macro avg 0.98 0.85 0.91 56746  
weighted avg 1.00 1.00 1.00 56746

# XGBoost Classifier

model\_X = xgb.XGBClassifier()  
model\_X.fit(x\_train,y\_train)

XGBClassifier(base\_score=None, booster=None, callbacks=None,  
 colsample\_bylevel=None, colsample\_bynode=None,  
 colsample\_bytree=None, device=None, early\_stopping\_rounds=None,  
 enable\_categorical=False, eval\_metric=None, feature\_types=None,  
 gamma=None, grow\_policy=None, importance\_type=None,  
 interaction\_constraints=None, learning\_rate=None, max\_bin=None,  
 max\_cat\_threshold=None, max\_cat\_to\_onehot=None,  
 max\_delta\_step=None, max\_depth=None, max\_leaves=None,  
 min\_child\_weight=None, missing=nan, monotone\_constraints=None,  
 multi\_strategy=None, n\_estimators=None, n\_jobs=None,  
 num\_parallel\_tree=None, random\_state=None, ...)

train\_x\_pred = model\_X.predict(x\_train)  
print(classification\_report(y\_train,train\_x\_pred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 226602  
 1 1.00 1.00 1.00 378  
  
 accuracy 1.00 226980  
 macro avg 1.00 1.00 1.00 226980  
weighted avg 1.00 1.00 1.00 226980

test\_x\_pred = model\_r.predict(x\_test)  
  
print(classification\_report(y\_test,test\_x\_pred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 56651  
 1 0.96 0.71 0.81 95  
  
 accuracy 1.00 56746  
 macro avg 0.98 0.85 0.91 56746  
weighted avg 1.00 1.00 1.00 56746

# CONCLUSION

Data is Highly imbanced hence, first we have balance the data and then create and train model by using Logistic Regression,Random forest classifier and XGBoost Classifier.By using Logistic regression we got Accuracy of model on training data around 0.96 and on test data 0.96. We can said that our model is performance is Equivalent on both training and testing data.By using Random forest classifier we got accuracy of model on training data around 1.00 and on test data 1.00. We can said that our model is working really good on training and testing data.By using XGBoost classifier we got accuracy of model on training data around 1.00 and on test data 1.00. We can said that our model is working really good on training and testing data.