**CREDIT CARD FRAUD DETECTION**

**PHASE 5**

**PHASES OF DEVELOPMENT** :

1. Data Collection

2. Data Preprocessing

3. Feature Engineering

4. Model Training

5. Model Evaluation

**DATA COLLECTION:**

The data was collected from Kaggle and converted into a CSV file.

**Dataset Link: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud**

**LOAD DATA :**

import numpy as np

import pandas as pd

import sklearn

import scipy

import matplotlib.pyplot asplt

import seaborn as sns

from sklearn.metrics import classification\_report,accuracy\_score

from sklearn.ensemble import IsolationForest

from sklearn.neighbors import LocalOutlierFactor

from sklearn.svm import OneClassSVM

from pylab import rcParams

rcParams['figure.figsize'] = 14, 8

RANDOM\_SEED = 42

LABELS = ["Normal", "Fraud"]

#import plotly.plotly as py

import plotly.graph\_objs as go

import plotly

import plotly.figure\_factory as

from plotly.offline import init\_notebook\_mode, iplot

df = pd.read\_csv(r'D:\Krishna priya\creditcard.csv')

df.head()

**OUTPUT :**

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

**DATA PREPROCESSING :**

Data information revealed that there were no missing values in the dataset, indicating that it was a clean dataset.

Descriptive statistics provided insights into the distribution and characteristics of the data, including mean, standard deviation, minimum, and maximum values for each feature.

data1= data.sample(frac = 0.1,random\_state=1)

data1.shape

**OUTPUT:**

(28481, 31)

## Preprocessing the dataset :

data.isnull().sum()

**OUTPUT:**

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

data.describe()

|  |
| --- |

**OUTPUT**:

|  | **Time** | **V1** | **V2** | **V3** | **V4** | **V5** | **V6** | **V7** | **V8** | **V9** | **...** | **V21** | **V22** | **V23** | **V24** | **V25** | **V26** | **V27** | **V28** | **Amount** | **Class** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 284807.000000 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | ... | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 284807.000000 | 284807.000000 |
| **mean** | 94813.859575 | 1.168375e-15 | 3.416908e-16 | -1.379537e-15 | 2.074095e-15 | 9.604066e-16 | 1.487313e-15 | -5.556467e-16 | 1.213481e-16 | -2.406331e-15 | ... | 1.654067e-16 | -3.568593e-16 | 2.578648e-16 | 4.473266e-15 | 5.340915e-16 | 1.683437e-15 | -3.660091e-16 | -1.227390e-16 | 88.349619 | 0.001727 |
| **std** | 47488.145955 | 1.958696e+00 | 1.651309e+00 | 1.516255e+00 | 1.415869e+00 | 1.380247e+00 | 1.332271e+00 | 1.237094e+00 | 1.194353e+00 | 1.098632e+00 | ... | 7.345240e-01 | 7.257016e-01 | 6.244603e-01 | 6.056471e-01 | 5.212781e-01 | 4.822270e-01 | 4.036325e-01 | 3.300833e-01 | 250.120109 | 0.041527 |
| **min** | 0.000000 | -5.640751e+01 | -7.271573e+01 | -4.832559e+01 | -5.683171e+00 | -1.137433e+02 | -2.616051e+01 | -4.355724e+01 | -7.321672e+01 | -1.343407e+01 | ... | -3.483038e+01 | -1.093314e+01 | -4.480774e+01 | -2.836627e+00 | -1.029540e+01 | -2.604551e+00 | -2.256568e+01 | -1.543008e+01 | 0.000000 | 0.000000 |
| **25%** | 54201.500000 | -9.203734e-01 | -5.985499e-01 | -8.903648e-01 | -8.486401e-01 | -6.915971e-01 | -7.682956e-01 | -5.540759e-01 | -2.086297e-01 | -6.430976e-01 | ... | -2.283949e-01 | -5.423504e-01 | -1.618463e-01 | -3.545861e-01 | -3.171451e-01 | -3.269839e-01 | -7.083953e-02 | -5.295979e-02 | 5.600000 | 0.000000 |
| **50%** | 84692.000000 | 1.810880e-02 | 6.548556e-02 | 1.798463e-01 | -1.984653e-02 | -5.433583e-02 | -2.741871e-01 | 4.010308e-02 | 2.235804e-02 | -5.142873e-02 | ... | -2.945017e-02 | 6.781943e-03 | -1.119293e-02 | 4.097606e-02 | 1.659350e-02 | -5.213911e-02 | 1.342146e-03 | 1.124383e-02 | 22.000000 | 0.000000 |
| **75%** | 139320.500000 | 1.315642e+00 | 8.037239e-01 | 1.027196e+00 | 7.433413e-01 | 6.119264e-01 | 3.985649e-01 | 5.704361e-01 | 3.273459e-01 | 5.971390e-01 | ... | 1.863772e-01 | 5.285536e-01 | 1.476421e-01 | 4.395266e-01 | 3.507156e-01 | 2.409522e-01 | 9.104512e-02 | 7.827995e-02 | 77.165000 | 0.000000 |
| **max** | 172792.000000 | 2.454930e+00 | 2.205773e+01 | 9.382558e+00 | 1.687534e+01 | 3.480167e+01 | 7.330163e+01 | 1.205895e+02 | 2.000721e+01 | 1.559499e+01 | ... | 2.720284e+01 | 1.050309e+01 | 2.252841e+01 | 4.584549e+00 | 7.519589e+00 | 3.517346e+00 | 3.161220e+01 | 3.384781e+01 | 25691.160000 | 1.000000 |

**FEATURE ENGINEERING :**

The first step is to prepare the data for anomaly detection. We will start by importing the necessary libraries and loading the dataset into a Pandas.Convert categorical variables into numerical representations for model compatibility.

count\_classes = pd.value\_counts(data['Class'], sort = True)

count\_classes.plot(kind = 'bar', rot=0)

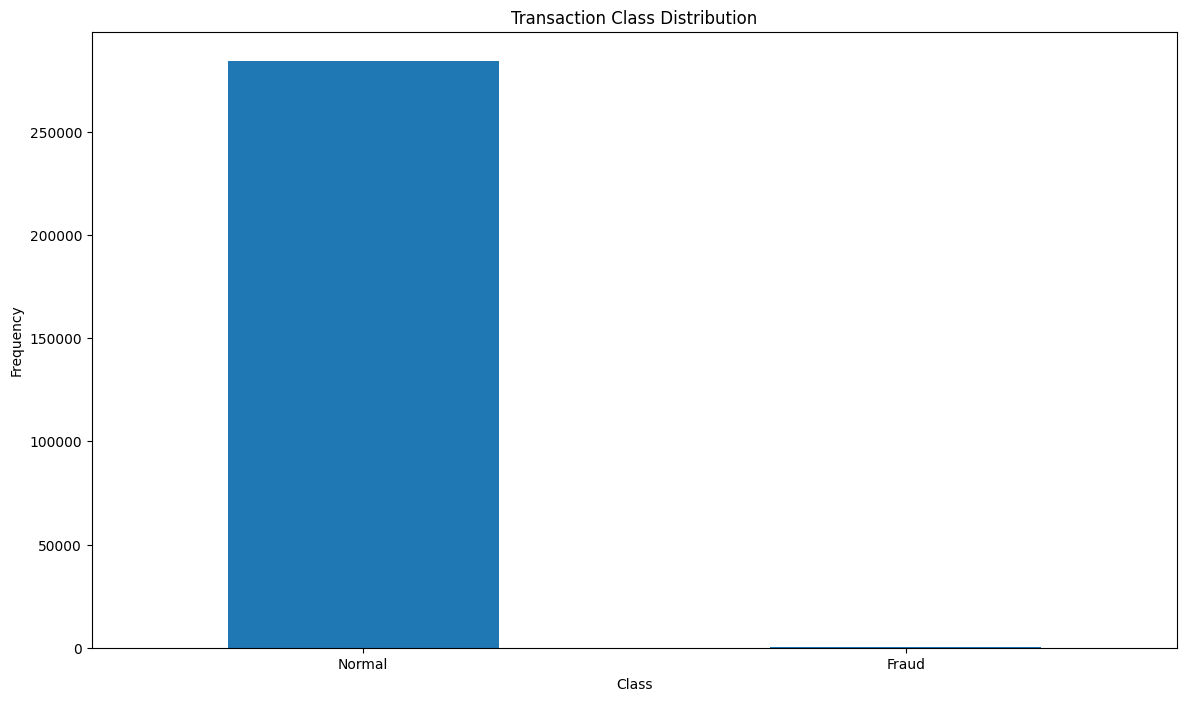
plt.title("Transaction Class Distribution")

plt.xticks(range(2), LABELS)

plt.xlabel("Class")

plt.ylabel("Frequency");

**OUTPUT:**



Normal = data[data['Class']==0]

Fraud = data[data['Class']==1]

Normal.shape

**OUTPUT**:

(284315, 31)

**MODEL TRAINING :**

Algorithm Selection:

Choose the advanced techniques such as anomaly detection algorithms (e.g., Isolation Forest, One-Class SVM) considering the dataset's characteristics and complexity. Experiment with ensemble methods for improved fraud detection accuracy.

(i) Anomaly Detection Algorithms

(ii) Ensemble techniques

Fraud.shape

OUTPUT:

(492, 31)

Normal.Amount.describe()

OUTPUT:

count 284315.000000

mean 88.291022

std 250.105092

min 0.000000

25% 5.650000

50% 22.000000

75% 77.050000

max 25691.160000

Name: Amount, dtype: float64

Fraud.Amount.describe()

OUTPUT:

count 492.000000

mean 122.211321

std 256.683288

min 0.000000

25% 1.000000

50% 9.250000

75% 105.890000

max 2125.870000

Name: Amount, dtype: float64

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Amount per transaction by class')

bins = 50

ax1.hist(Fraud.Amount, bins = bins)

ax1.set\_title('Fraud')

ax2.hist(Normal.Amount, bins = bins)

ax2.set\_title('Normal')

plt.xlabel('Amount ($)')

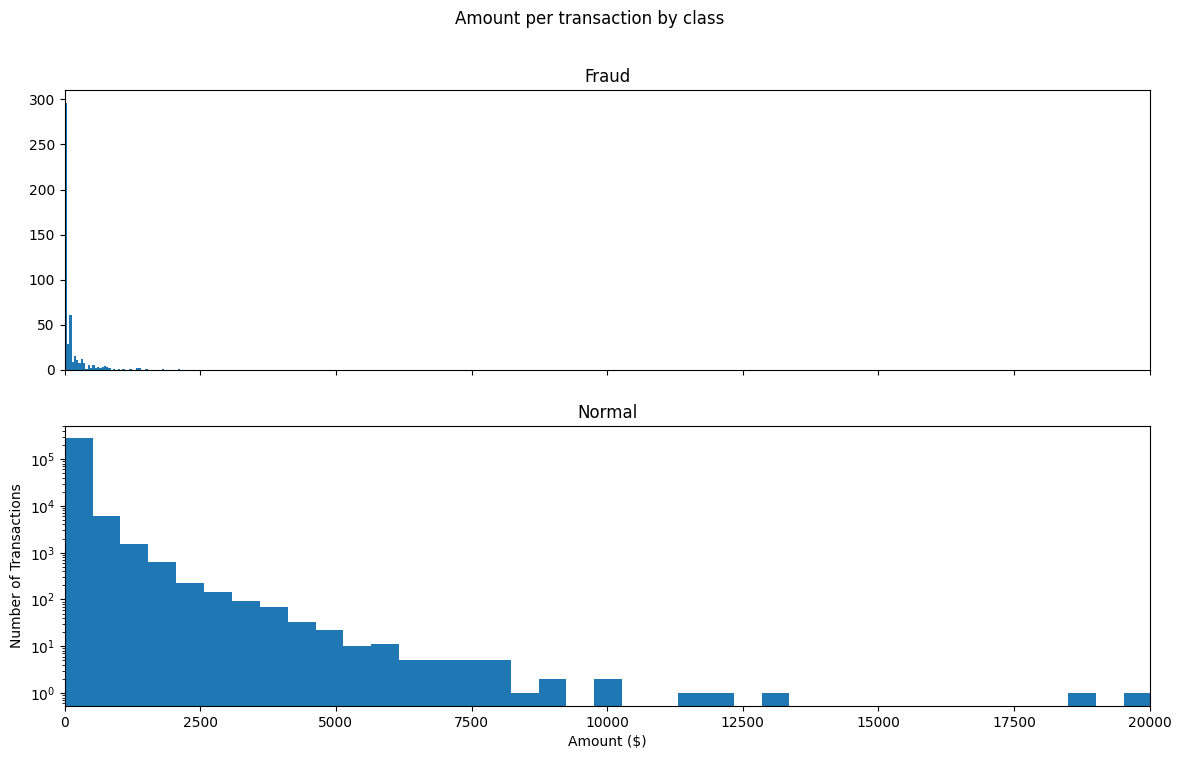
plt.ylabel('Number of Transactions')

plt.xlim((0, 20000))

plt.yscale('log')

plt.show();

**OUTPUT:**



**MODEL EVALUATION :**

The final step is to evaluate the selected model performance to make predictions on new data. In this step, we will use the selected model to make predictions on the test dataset and evaluate its performance using classification metrics.

We will use the predict method of the trained model to make predictions on the test data, and then evaluate the model’s performance using accuracy score, precision score, recall score, F1\_score and ROU-AUC metrics from the sklearn.metrics module.

## **Correlation Matrix Analysis**

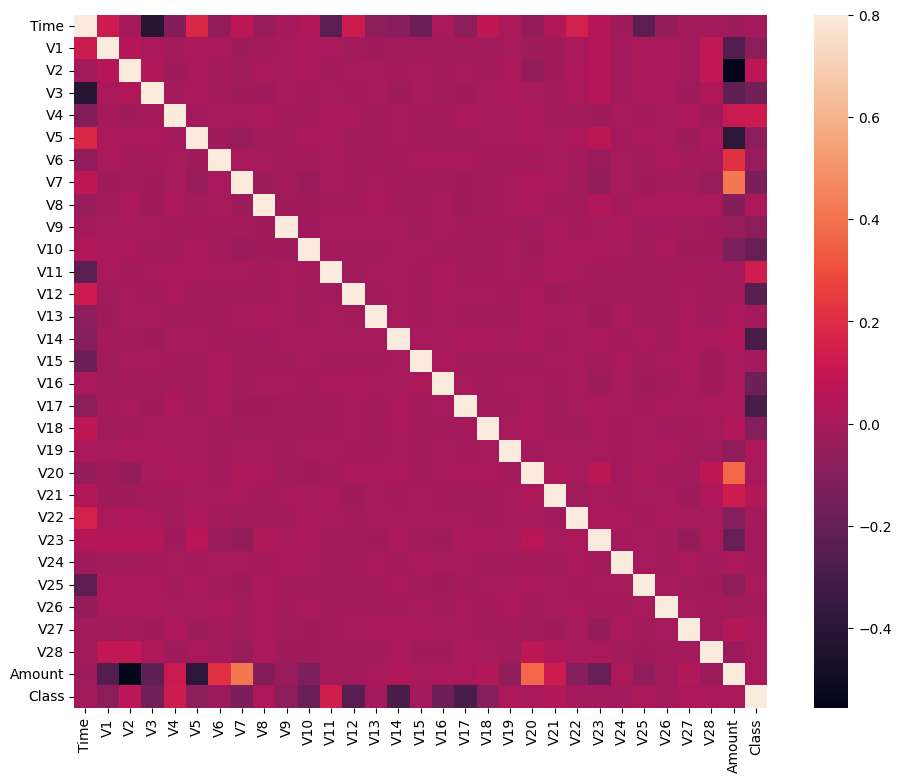
correlation\_matrix = data1.corr()

fig = plt.figure(figsize=(12,9))

sns.heatmap(correlation\_matrix,vmax=0.8,square = True)

plt.show()

OUTPUT:



LOGISTIC REGRESSION

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

# training

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

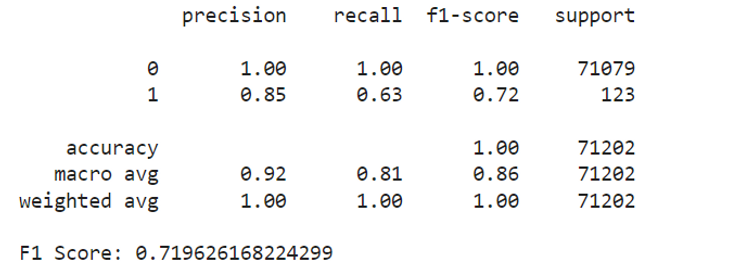
# testing

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

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

print("F1 Score:",f1\_score(y\_test, y\_pred))

OUTPUT:



RANDOM FOREST :

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n\_jobs=-1)

# training

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

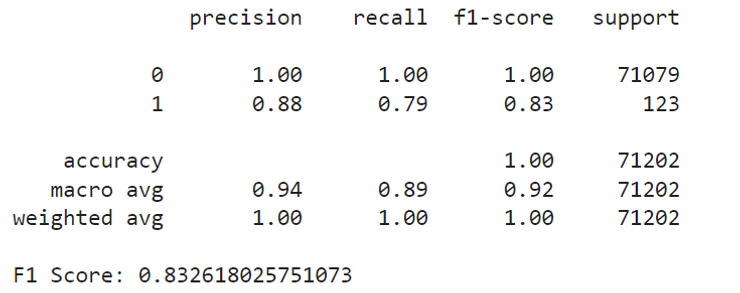
# testing

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

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

print("F1 Score:",f1\_score(y\_test, y\_pred))

OUTPUT:



print('Logistic Regression:')

print(classification\_report(original\_ytest, org\_log\_reg\_pred))

print('KNears Neighbors:')

print(classification\_report(original\_ytest, org\_knears\_pred))

print('Support Vector Classifier:')

print(classification\_report(original\_ytest, org\_svc\_pred))

'''print('Tree Classifier:')

print(classification\_report(original\_ytest, org\_tree\_pred))

print('Random Forest Classifier:')

print(classification\_report(original\_ytest, org\_rfc\_pred))

print('XG:')

print(classification\_report(original\_ytest, org\_xg\_pred))''

OUTPUT:

Logistic Regression:

precision recall f1-score support

0 1.00 0.97 0.99 56773

1 0.06 0.90 0.11 98

accuracy 0.97 56871

macro avg 0.53 0.94 0.55 56871

weighted avg 1.00 0.97 0.99 56871

KNears Neighbors:

precision recall f1-score support

0 1.00 0.99 0.99 56773

1 0.10 0.88 0.17 98

accuracy 0.99 56871

macro avg 0.55 0.93 0.58 56871

weighted avg 1.00 0.99 0.99 56871

Support Vector Classifier:

precision recall f1-score support

0 1.00 0.97 0.98 56773

1 0.04 0.90 0.09 98

accuracy 0.97 56871

macro avg 0.52 0.93 0.53 56871

weighted avg 1.00 0.97 0.98 56871

**CONCLUSION**:

In conclusion, the credit card fraud detection project has successfully demonstrated the importance of leveraging advanced machine learning algorithms and data analysis techniques to protect financial institutions and cardholders from fraudulent activities. By accurately identifying and preventing fraudulent transactions, this project not only enhances security but also fosters trust in the financial system, ultimately benefiting both businesses and consumers. Continued vigilance, innovation, and collaboration in the field of fraud detection are essential to stay one step ahead of evolving fraudsters and safeguard the integrity of financial transactions.