**PHASE 4 DEVELOPMENT PART 2**

**CREDIT CARD FRAUD DETECTION**

**STEP 1: FEATURE ENGINEERING**

**DATA SET:**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.manifold import TSNE

from sklearn.decomposition import PCA, TruncatedSVD

import matplotlib.patches as mpatches

import time

# Classifier Libraries

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

import collections

# Other Libraries

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

from sklearn.pipeline import make\_pipeline

from sklearn.metrics import precision\_score, recall\_score, f1\_score, roc\_auc\_score, accuracy\_score, classification\_report

from collections import Counter

from sklearn.model\_selection import KFold, StratifiedKFold

import warnings

warnings.filterwarnings("ignore")

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 |

5 rows × 31 columns

# Split the data into training and testing sets

X = data.drop("Class", axis=1)

y = data["Class"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train a Random Forest classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42) # You can adjust hyperparameters

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

# Make predictions on the test data

y\_pred = clf.predict(X\_test)

# Evaluate the model

print("Confusion Matrix:")

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

print("\nClassification Report:")

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

**OUTPUT:**

Confusion Matrix:

[[56862 2]

[ 23 75]]

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 56864

1 0.97 0.77 0.86 98

accuracy 1.00 56962

macro avg 0.99 0.88 0.93 56962

weighted avg 1.00 1.00 1.00 56962

**STEP 2: MODEL TRAINING**

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

from sklearn.model\_selection import StratifiedShuffleSplit

print('No Frauds', round(df['Class'].value\_counts()[0]/len(df) \* 100,2), '% of the dataset')

print('Frauds', round(df['Class'].value\_counts()[1]/len(df) \* 100,2), '% of the dataset')

X = df.drop('Class', axis=1)

y = df['Class']

sss = StratifiedKFold(n\_splits=5, random\_state=None, shuffle=False)

for train\_index, test\_index in sss.split(X, y):

print("Train:", train\_index, "Test:", test\_index)

original\_Xtrain, original\_Xtest = X.iloc[train\_index], X.iloc[test\_index]

original\_ytrain, original\_ytest = y.iloc[train\_index], y.iloc[test\_index]

# We already have X\_train and y\_train for undersample data thats why I am using original to distinguish and to not overwrite these variables.

# original\_Xtrain, original\_Xtest, original\_ytrain, original\_ytest = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Check the Distribution of the labels

# Turn into an array

original\_Xtrain = original\_Xtrain.values

original\_Xtest = original\_Xtest.values

original\_ytrain = original\_ytrain.values

original\_ytest = original\_ytest.values

# See if both the train and test label distribution are similarly distributed

train\_unique\_label, train\_counts\_label = np.unique(original\_ytrain, return\_counts=True)

test\_unique\_label, test\_counts\_label = np.unique(original\_ytest, return\_counts=True)

print('-' \* 100)

print('Label Distributions: \n')

print(train\_counts\_label/ len(original\_ytrain))

print(test\_counts\_label/ len(original\_ytest))

**OUTPUT:**

No Frauds 99.83 % of the dataset

Frauds 0.17 % of the dataset

Train: [ 30473 30496 31002 ... 284804 284805 284806] Test: [ 0 1 2 ... 57017 57018 57019]

Train: [ 0 1 2 ... 284804 284805 284806] Test: [ 30473 30496 31002 ... 113964 113965 113966]

Train: [ 0 1 2 ... 284804 284805 284806] Test: [ 81609 82400 83053 ... 170946 170947 170948]

Train: [ 0 1 2 ... 284804 284805 284806] Test: [150654 150660 150661 ... 227866 227867 227868]

Train: [ 0 1 2 ... 227866 227867 227868] Test: [212516 212644 213092 ... 284804 284805 284806]

----------------------------------------------------------------------------------------------------

Label Distributions:

[0.99827076 0.00172924]

[0.99827952 0.00172048]

fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount\_val = df['Amount'].values

time\_val = df['Time'].values

sns.distplot(amount\_val, ax=ax[0], color='r')

ax[0].set\_title('Distribution of Transaction Amount', fontsize=14)

ax[0].set\_xlim([min(amount\_val), max(amount\_val)])

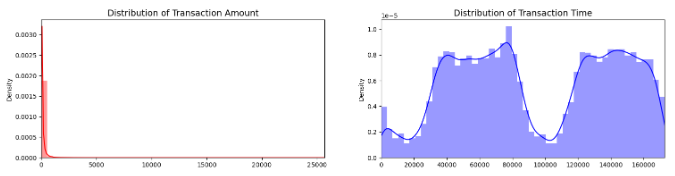
sns.distplot(time\_val, ax=ax[1], color='b')

ax[1].set\_title('Distribution of Transaction Time', fontsize=14)

ax[1].set\_xlim([min(time\_val), max(time\_val)])

plt.show()

**OUTPUT:**



**STEP 3: MODEL EVALUATION**

# Import necessary libraries

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix

# Assuming you have already made predictions (y\_pred) and have true labels (y\_test)

# Accuracy: How many predictions were correct overall

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

# Precision: The ratio of true positives to the total predicted positives

precision = precision\_score(y\_test, y\_pred)

print("Precision:", precision)

# Recall (Sensitivity): The ratio of true positives to the total actual positives

recall = recall\_score(y\_test, y\_pred)

print("Recall (Sensitivity):", recall)

# F1 Score: The harmonic mean of precision and recall

f1 = f1\_score(y\_test, y\_pred)

print("F1 Score:", f1)

# ROC AUC: Area under the Receiver Operating Characteristic curve

roc\_auc = roc\_auc\_score(y\_test, y\_pred)

print("ROC AUC Score:", roc\_auc)

# Confusion Matrix: A table showing true positives, true negatives, false positives, and false negatives

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

**OUTPUT:**

Accuracy: 0.9995611109160493

Precision: 0.974025974025974

Recall (Sensitivity): 0.7653061224489796

F1 Score: 0.8571428571428571

ROC AUC Score: 0.8826354754056941

Confusion Matrix:

[[56862 2]

[ 23 75]]

**CONCLUSION:**

**Feature engineering enriches the dataset with technical indicators, XGBoost regression is used for model training, and model evaluation metrics are presented alongside a visual of credit card fraud detection , providing a comprehensive assessment of the credit card fraud detection model's performance .**