

# Credit Card Fraud Detection

## (Project Report)



CMPE-256 Section 49

Advanced Data Mining

Submitted by

**Team Data Marvels**

Priyanka Math	015240134
---------------	-----------

Jithesh Kurungote Balakrishnan	014654679
--------------------------------	-----------

Tharun Mukka	014640496
--------------	-----------

**Instructor:** Chandrasekar Vuppalapati

## **ABSTRACT**

In recent years, credit cards have become the most popular method of payment. As technology advances, the number of fraud instances grows, necessitating the development of a fraud detection algorithm to accurately detect and eliminate fraudulent actions.

The identification of credit card fraud is currently the most common problem in the modern world. This is because internet transactions and e-commerce platforms are on the rise. Credit card fraud occurs when a card is stolen and used for unlawful reasons, or when a fraudster utilizes the credit card information for his or her own interests. To determine if a transaction is fraudulent or not, we must first determine the customer's spending behavior using existing data and then use machine learning to determine whether the transaction is legitimate or not.

The primary goal of this project is to tackle credit card fraud by building Classification models such as XG Boost, Decision tree and KNN to classify fraud transactions and evaluate the created classification models using the evaluation metrics.

# **TABLE OF CONTENTS**

## **Chapter 1**

Project Requirements

## **Chapter 2**

Knowledge Discovery In Databases

## **Chapter 3**

Feature Engineering

## **Chapter 4**

High Level Architecture Design

## **Chapter 5**

Component Level Designs

## **Chapter 6**

Workflow With Data Science Algorithms

## **Chapter 7**

Interfaces – Restful & Server-Side Design

## **Chapter 8**

Model Deployment

## **Chapter 9**

Client-Side Design And Testing

## **Chapter 10**

Deployment And Source Code Materials

# **Chapter 1 Project Requirements**

**Programming Language** – Python

**Development tool** - Google Colab and Flask

**Packages** -

- **Pandas** to work with data
- **NumPy** to work with arrays
- **Scikit-learn** for data split, building and evaluating the classification models
- **Xgboost** package for the xgboost classifier model algorithm

**Deployment environment** – Google Cloud Platform

## **Chapter 2 Knowledge Discovery In Databases**

### **Understanding Application Domain**

E-commerce has come a long way since its inception. It has become an essential means for most organizations, companies, and government agencies to increase their productivity in global trade. One of the main reasons for the success of e-commerce is the easy online credit card transaction [2]. Whenever we talk about monetary transactions, we also have to take financial fraud into consideration. Financial fraud is an intentional crime in which a fraudster benefits himself/herself by denying a right to a victim or by obtaining financial gain [3]. As credit card transactions are the most common method of payment in recent years, the fraud activities have increased rapidly.

The solutions to the fraud can be categorized into prevention, which involves preventing the fraud in the source itself and detection, which is the action taken after the occurrence of the event. The technologies like the Address Verification System (AVS) and Card Verification System (CVM) are usually operated to prevent fraud. Basically, rule-based filters and data mining methods are used for the prevention [4].

However, when fraud cannot be prevented from occurring, then it has to be detected as soon as possible, and necessary actions should be taken against it. Fraud detection is the process of detecting whether a transaction is legitimate or not [5]. Automated fraud detection systems are required especially considering the huge traffic of transaction data, and it is not possible for humans to check manually every transaction one by one if it is fraudulent or not.

### **Objective**

The main objective is to perform predictive analysis on credit card transaction dataset using machine learning techniques and detect the fraudulent transactions from the given dataset. The focus is to identify if a transaction comes under normal class or fraudulent class using predictive models. Different sampling techniques will be implemented and a series of machine learning algorithms will be implemented on the dataset, and the results will be reported.

## **Data Description**

Datasets are an integral part in the field of machine learning. Gathering data is one of the hardest tasks, especially when it is related to the financial domain like credit card fraud. The dataset considered from Kaggle contains the record of credit card transactions made by European cardholders and occurred in two days in September 2013. The dataset contains 284,807 transactions out of which only 492 are fraudulent. The dataset is highly unbalanced as the positive class accounts for only 0.172% of the total transactions.

The dataset contains 30 independent variables. Features V1, V2, ..., V28 are the principal components obtained with PCA, and the only features that have not been transformed into principal components are 'Amount' and 'Time'. The 'Class' attribute is a discrete-valued binary attribute that takes value '0' for non-fraudulent transactions and '1' for fraud transactions.

## **Data Selection**

The 'Time' feature which is of no use to build the models, can be neglected. The remaining features are the 'Amount' feature that contains the total amount of money being transacted, features V1 to V28 which are the principal components obtained by PCA and the 'Class' feature that contains whether the transaction is a fraud case or not.

## **Data Preprocessing**

There are only 492 fraud cases out of 284,807 samples which is only 0.172% of the total samples. Hence, the dataset is highly unbalanced. To resolve this issue, resampling techniques like SMOTE (Oversampling) and Tomek Link (Under sampling) will be performed.

## **Data Transformation**

There are a total 30 features out of which 28 features have been generated by principal component analysis. PCA is a dimensionality-reduction technique in which a large number of original variables are condensed into a smaller subset of feature variables. The only features that have not been transformed into principal components are 'Amount' and 'Time'.

The values in the 'Amount' variable are varying enormously when compared to the rest of the variables. To reduce its wide range of values, it needs to be normalized using the standard scaler technique.

## **Data Mining**

Predictive Analysis is performed by building classification models like XG Boost, Decision tree and K-Nearest Neighbors (KNN) to classify fraud transactions.

## **Evaluation**

Evaluation of the created classification models is performed using F1-Score matrix and Confusion matrix.

## **Chapter 3 Feature Engineering**

There are a total 30 independent features out of which 28 features have been generated by principal component analysis. The only features that have not been transformed into principal components are 'Amount' and 'Time'.

Feature 'Time' contains the second elapsed between each transaction and the first transaction in the dataset, and it is not included in our analysis because it is just a timestamp, which is unrelated to our analysis.

Feature 'Class' is the response variable and it takes value '1' for fraud transactions and '0' for non-fraud transactions. Finally, 29 independent variables and one dependent variable are considered for analysis.

The values in the 'Amount' variable are varying enormously when compared to the rest of the variables. To reduce its wide range of values, it needs to be normalized between (-1, 1) using the 'StandardScaler' method in Python.

There are only 492 fraud cases out of 284,807 samples which is only 0.172% of the total samples. Hence, the dataset is highly unbalanced. To resolve this issue, resampling techniques like SMOTE (Oversampling) and Tomek Link (Under sampling) will be performed.

The approach is to heavily over-sample the fraudulent transaction class using SMOTE and use Tomek Links to lightly under-sample the legitimate transaction class. The most important aspect of this is the over-sampling. However, the way 'imblearn' package implements this combination of under- and over-sampling is to first under-sample and then over-sample because it is more efficient.



The goal of the Tomek Link (under-sampling) is to eliminate some of the legitimate transactions that are near the edges of, or are surrounded by, the set of fraudulent transactions. This will make the boundary between the two classes more clear. Under-sampling first will make the generation of new fraudulent transaction members more efficient and will produce more homogeneity with respect to the geometric/spatial distributions of the two classes.

The classical version of SMOTE (Synthetic Minority Over-sampling Technique) does the following to create synthetic minority class samples.

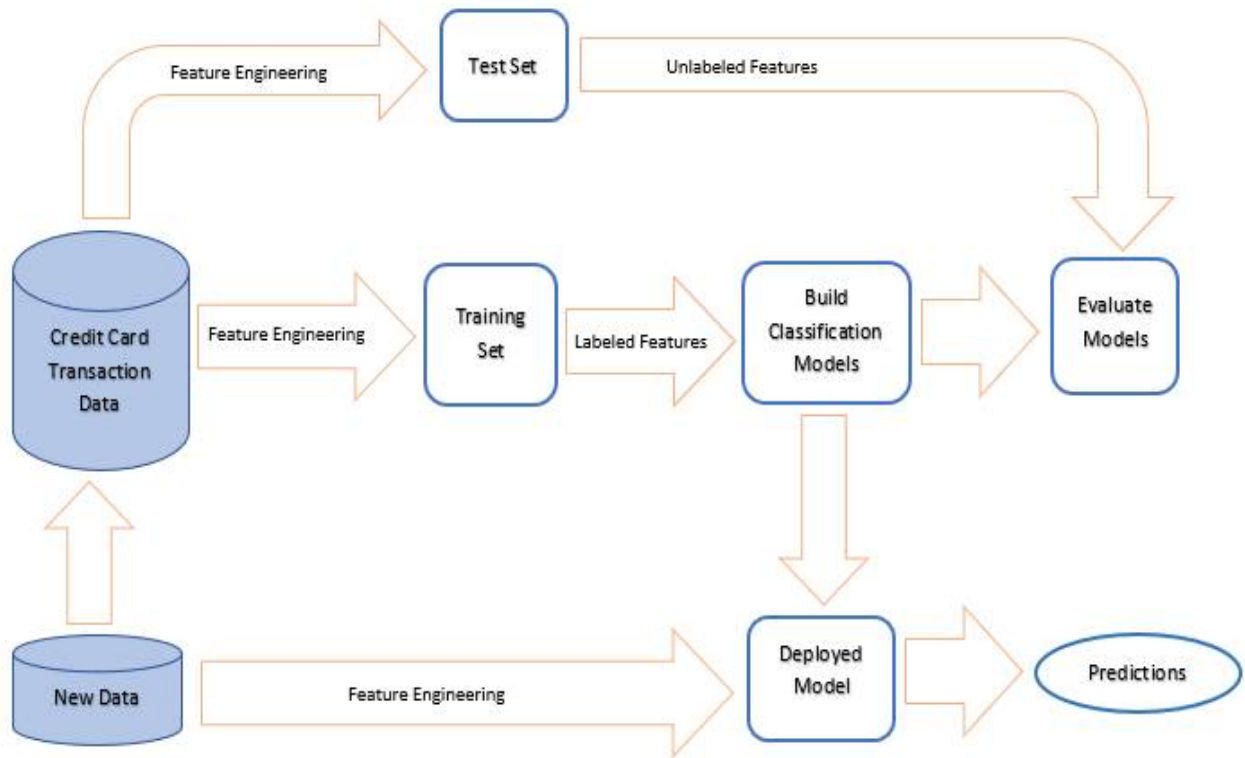
First, SMOTE chooses a minority class data point.

Then, SMOTE uses part of the k-Nearest Neighbors (kNN) method to find other minority class data points nearby.

Then, SMOTE creates synthetic minority samples that lie on the lines connecting the chosen minority class point to some or all of its nearest neighbors of the same class.

The split between testing and training needs to be done before any over- or under-sampling is performed. The data science terminology for this type of error is data leakage. Care must be taken to avoid doing this. The test set data points must be representative in both the features of the data and the class imbalance or we will not be able to have the confidence to generalize the model to detect credit card fraud on other credit fraud datasets.

## Chapter 4 High Level Architecture Design

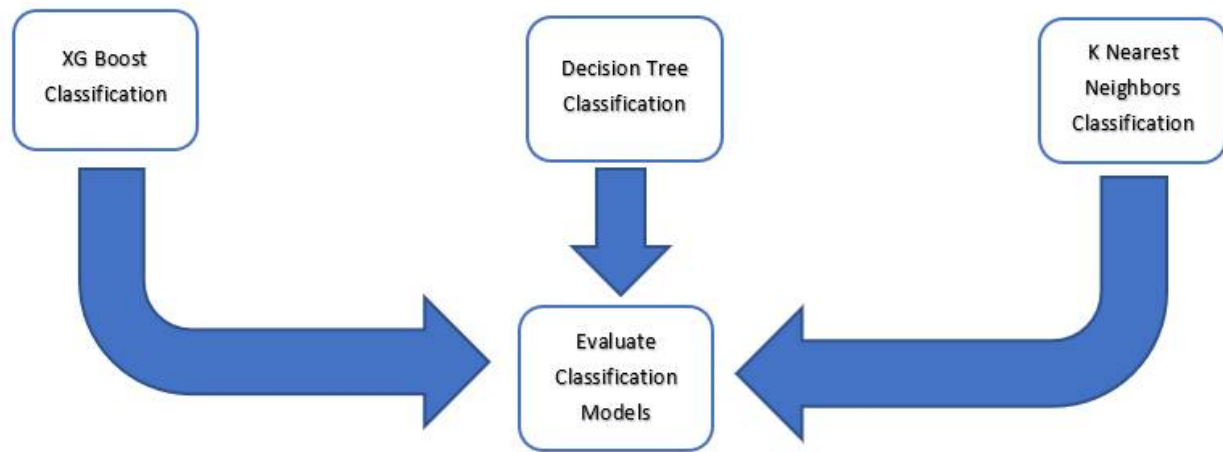


## Chapter 5 Component Level Designs

### Feature Engineering Process:



## Classification Model Training:



## **Chapter 6 Workflow With Data Science Algorithms**

### **1. Importing the required dependencies for the project**

For this project, our primary packages are going to be Pandas to work with data, NumPy to work with arrays, scikit-learn for pre-processing the data, training and evaluating the classification models, and finally the xgboost package for the xgboost classifier model algorithm.

#### **Importing the Dependencies**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import itertools
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.preprocessing import StandardScaler
import imblearn
from collections import Counter
```

### **2. Data Importing and EDA**

**Dataset:** <https://www.kaggle.com/mlg-ulb/creditcardfraud>

The dataset contains transactions made by credit cards in September 2013 by European cardholders.

This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

There are a total 30 independent features out of which 28 features have been generated by principal component analysis. The only features that have not been transformed into principal components are 'Amount' and 'Time'.

Loading Dataset to Pandas DataFrame

```
[ ] credit_data=pd.read_csv('/content/creditcard.csv')
```

```
[ ] credit_data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.631418
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.175121

The values in the 'Amount' variable are varying enormously when compared to the rest of the variables. To reduce its wide range of values, it needs to be normalized using the standard scaler technique.


### Scaling the Amount Feature to be in range between (-1,1)


```
credit_data.Amount.describe()
```

```
count    284807.000000
mean      88.349619
std       250.120109
min         0.000000
25%        5.600000
50%       22.000000
75%       77.165000
max     25691.160000
Name: Amount, dtype: float64
```

```
[ ] Non_Fraud_Transaction.Amount.describe()
```

```
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_Transaction.Amount.describe()`

 `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

```
[ ] std_scaler=StandardScaler()  
    credit_data['Amount'] = std_scaler.fit_transform(X_tra)
```

```
[ ] credit_data['Amount']
```

```
0      0.244964  
1     -0.342475  
2      1.160686  
3      0.140534  
4     -0.073403  
...  
284802 -0.350151  
284803 -0.254117  
284804 -0.081839  
284805 -0.313249  
284806  0.514355  
Name: Amount, Length: 284807, dtype: float64
```

```
[ ] credit_data.Amount.values
```

```
array([ 0.24496426, -0.34247454,  1.16068593, ..., -0.0818393 ,  
       -0.31324853,  0.51435531])
```

### 3. Data Selection

The 'Time' feature which is of no use to build the models, can be neglected. The remaining features are the 'Amount' feature that contains the total amount of money being transacted, features V1 to V28 which are the principal components obtained by PCA and the 'Class' feature that contains whether the transaction is a fraud case or not.

```
[ ] X = credit_data.drop(columns = ['Class','Time'])  
    y = credit_data['Class']
```

```
[ ] X.shape
```

```
(284807, 29)
```

```
[ ] y.shape
```

```
(284807,)
```

### 4. Splitting the Data into test and training dataset

In this process, we are going to define the independent (X) and the dependent variables (y) to split the data between training (70%) and testing (30%) dataset.

Splitting the data to train and test the model

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,stratify=y,random_state=2)
```

```
[ ] print(X.shape,X_train.shape,X_test.shape)
```

```
(284807, 29) (199364, 29) (85443, 29)
```

```
[ ] y.head()
```

```
0    0  
1    0  
2    0  
3    0  
4    0  
Name: Class, dtype: int64
```

## 5. Smote and Tomek Technique for Resampling the data

There are only 492 fraud cases out of 284,807 samples which is only 0.172% of the total samples. Hence, the dataset is highly unbalanced. To resolve this issue, resampling techniques like SMOTE (Oversampling) and Tomek Link (Under sampling) will be performed.

We have tried multiple combinations for over sampling and under sampling.

```
[ ] ## Import the SMOTE package
    # from imblearn.over_sampling import SMOTE
    # # Synthesize minority class datapoints using SMOTE
    # sm = SMOTE(random_state=42, sampling_strategy='minority')
    # smote_x_train, smote_y_train = sm.fit_resample(X_train, y_train)
    # print(smote_x_train.shape, smote_y_train.shape)
```

(454902, 29) (454902,)

```
▶ # smote = RandomForestClassifier(n_estimators=1000).fit(smote_x_train, smote_y_train)

    ## Predict on training set
    # smote_preds = smote.predict(X_test)
    # print('F1 Score: ', f1_score(y_test, smote_preds), '\n\n')
```

👤 F1 Score: , 0.276540987654

```
[ ] #over_and_under_sample = imblearn.combine.SMOTETomek(sampling_strategy = 1.0, n_jobs = -1, random_state = 2)
    smotek= imblearn.combine.SMOTETomek(random_state=2)
    #smotek.fit(X_train,y_train)
    X_train, y_train = smotek.fit_resample(X_train, y_train)
```

(454902, 29) (454902,)

After comparing both combinations, we found SMOTETomek is better and using it for further modelling.



## 6. Classification Modelling

Predictive Analysis is performed by building classification models like XG Boost, Decision tree and K-Nearest Neighbors (KNN) to classify fraud and non-fraud transactions.

All these models can be built feasibly using the algorithms provided by the scikit-learn package. Only for the XGBoost model, we are using the xgboost package.

```
[ ] # 1. Decision Tree
```

```
tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)
```



```
# 2. K-Nearest Neighbors
```

```
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)
```

```
[ ] # 3. XGBoost
```

```
xgb = XGBClassifier(max_depth = 4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)
```

## 7. Evaluation using F1 Score and Confusion Matrix

The F1 score or F-score is one of the most popular evaluation metrics used for evaluating classification models. It can be simply defined as the harmonic mean of the model's precision and recall. It is calculated by dividing the product of the model's precision and recall by the value obtained on adding the model's precision and recall and finally multiplying the result with 2.

$$\text{F1 score} = 2((\text{precision} * \text{recall}) / (\text{precision} + \text{recall}))$$

From below screenshot, we can observe that F1 score for KNN model is better when compared to XGBoost and Decision Tree models.

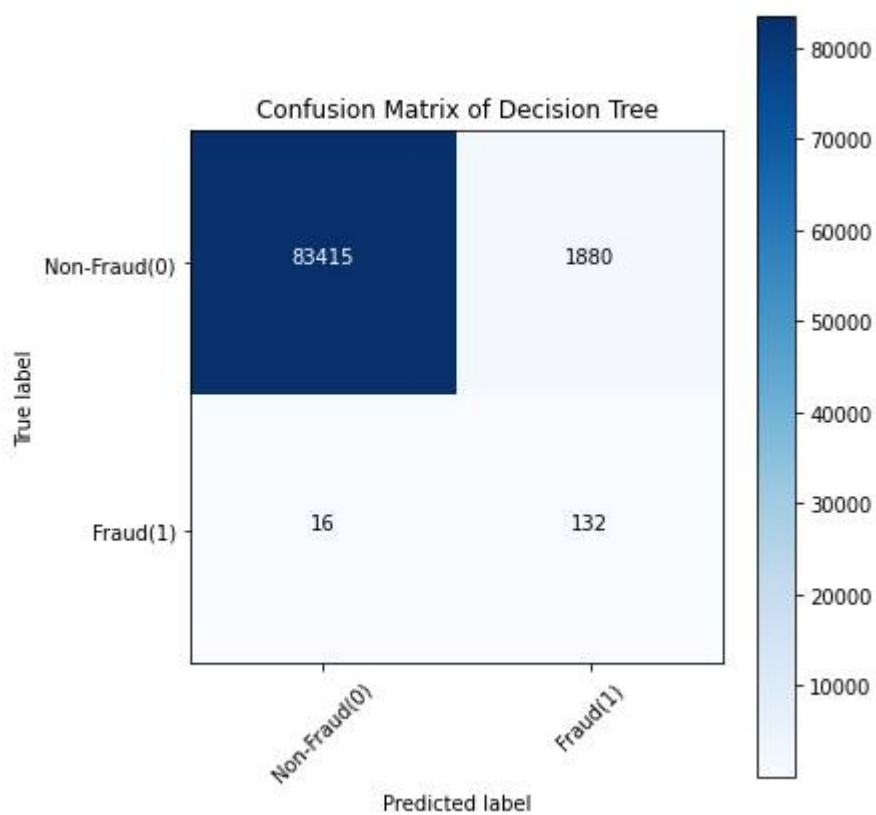
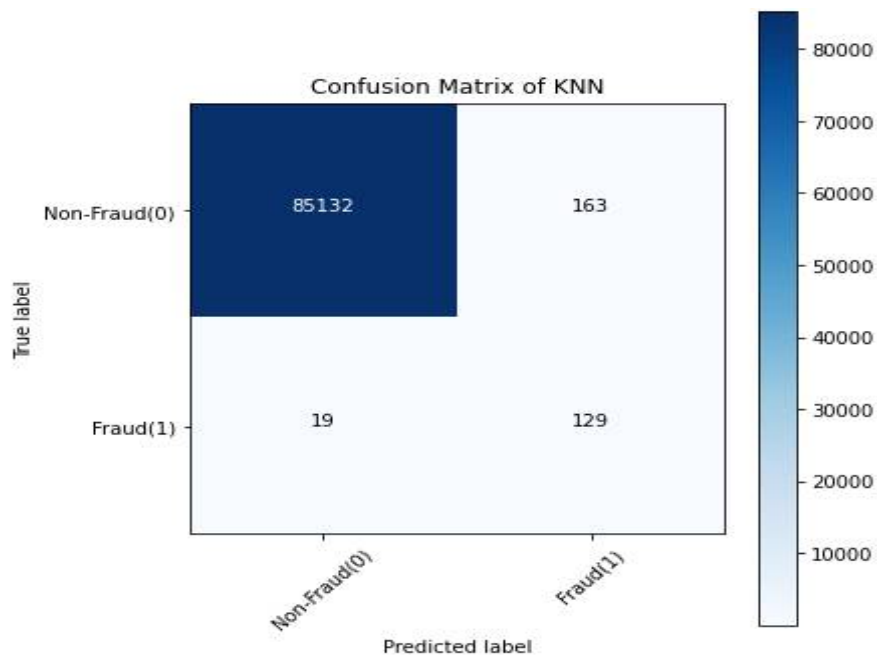
### Evaluation Scores (F1)

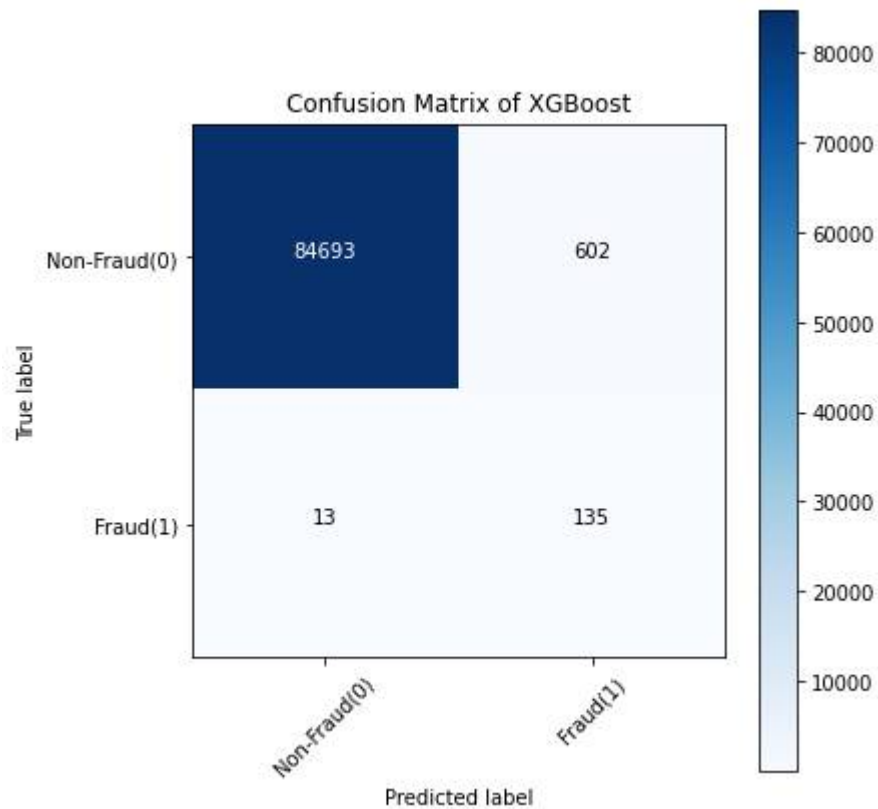
```
[ ] print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, tree_yhat)))  
    print('F1 score of the KNN model is {}'.format(f1_score(y_test, knn_yhat)))  
  
    print('F1 score of the XGBoost model is {}'.format(f1_score(y_test, xgb_yhat)))  
  
F1 score of the Decision Tree model is 0.12222222222222223  
F1 score of the KNN model is 0.5863636363636363  
F1 score of the XGBoost model is 0.3050847457627119
```

### Confusion Matrix

Typically, a confusion matrix is a visualization of a classification model that shows how well the model has predicted the outcomes when compared to the original ones. Usually, the predicted outcomes are stored in a variable that is then converted into a correlation table. Using the correlation table, the confusion matrix is plotted in the form of a heatmap.

Below are the screenshots of the confusion matrix created for KNN, Decision Tree and XGBoost models.



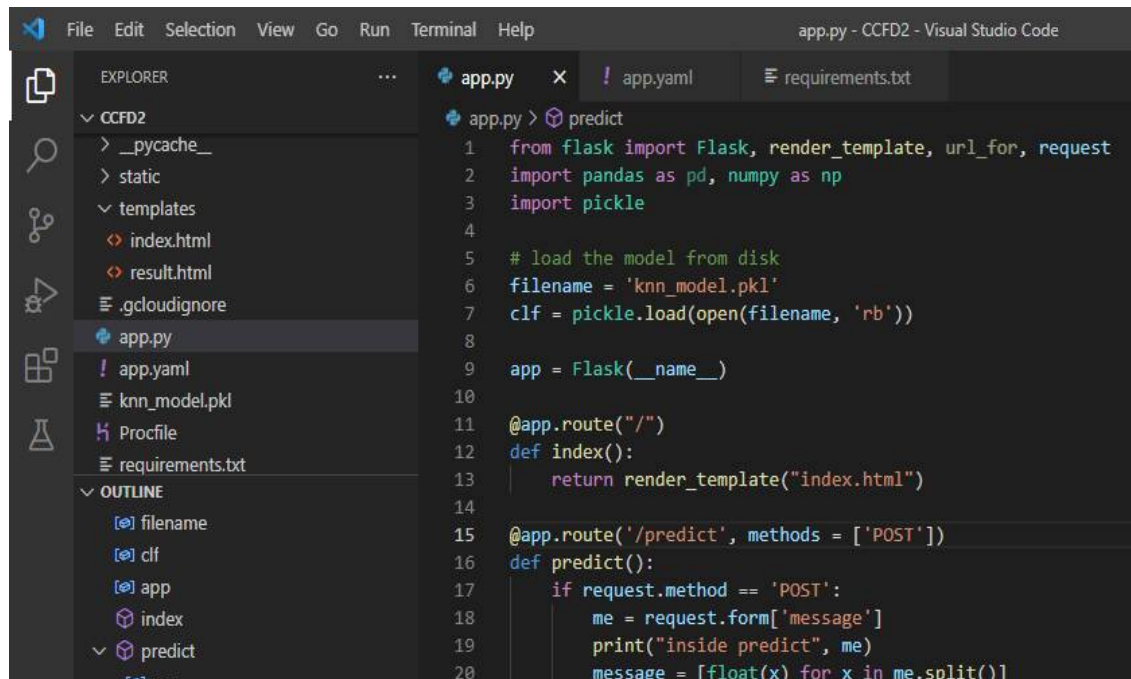


From all of the confusion matrix plotted above we can observe that KNN model has performed well when compared to XGBoost and Decision Tree models as it could detect 85132 Non-Fraud Transactions out of 85295. Also, for Non-Fraud transactions 129 of them were detected out of 148 transactions.

Hence, we can conclude that the most appropriate model which can be used for our case is the K-Nearest Neighbors model based on the results of the confusion matrix and accuracy score.

## Chapter 7 Interfaces – Restful & Server-Side Design

We have created a flask application to create RESTful APIs. Flask application uses the pickle files created for the classification models.



The screenshot displays the Visual Studio Code interface for a project named 'CCFD2'. The Explorer panel on the left shows the project structure, including files like `__pycache__`, `static`, `templates` (with `index.html` and `result.html`), `.gitignore`, `app.py`, `app.yaml`, `knn_model.pkl`, `Procfile`, and `requirements.txt`. The Outline panel shows variables like `filename`, `clf`, `app`, `index`, and `predict`. The main editor shows the `app.py` file with the following Python code:

```
1 from flask import Flask, render_template, url_for, request
2 import pandas as pd, numpy as np
3 import pickle
4
5 # load the model from disk
6 filename = 'knn_model.pkl'
7 clf = pickle.load(open(filename, 'rb'))
8
9 app = Flask(__name__)
10
11 @app.route("/")
12 def index():
13     return render_template("index.html")
14
15 @app.route('/predict', methods = ['POST'])
16 def predict():
17     if request.method == 'POST':
18         me = request.form['message']
19         print("inside predict", me)
20         message = [float(x) for x in me.split()]
```

## **Chapter 8 Model Deployment**

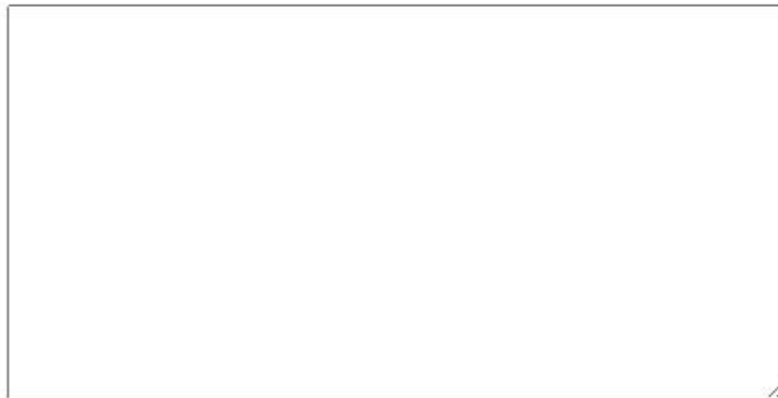
The flask application created is deployed on Google Cloud Platform.

**Deployed Application Link:** <https://creditcardfraud-333007.uw.r.appspot.com/>

## **Chapter 9 Client-Side Design And Testing**

# Credit Card Fraud Detection

Enter the 29 feature values below (in order):



Predict

**Testing for Non-Fraud Transaction:**

# Credit Card Fraud Detection

Enter the 29 feature values below (in order):

-1.53390402048031	-1.07378024389573
0.8627690222278699	-1.16375028009722
0.183810583824839	-0.301033763307833
-0.43125408762814904	-0.8476814210491809
0.100169974023655	0.0394749704942784
0.37236123309163793	-0.504906487940887
0.0805800627541363	0.0260289178604425
-0.2972561862516922	

Predict

**Prediction Result:**

## Credit Card Fraud Detection Results

According to the model trained, the provided transaction is **NOT** a Fraud transaction.

Retest

Testing for Fraud Transaction:

# Credit Card Fraud Detection

Enter the 29 feature values below (in order):

```
0.830910291033798    -9.533257050393189
-18.750641147467398  -8.09264877340557
3.32675827497024    0.42720343146936
-2.1826919456095504  0.5205430723666421
-0.7605564151887328  0.6627666383972359
-0.948454306235033    0.12179592582979301
-3.3818429293561    -1.2565236213625801
0.20610286556509647
```

Predict

Prediction Result:

## Credit Card Fraud Detection Results

**According to the model trained, this transaction is a Fraud transaction.**

Retest



## **Chapter 10 Deployment And Source Code Materials**

### **Colab Link:**

[https://colab.research.google.com/drive/1yQ0XrOouZJs1yM7JsA\\_y3v01NHtWZcBk?authuser=2](https://colab.research.google.com/drive/1yQ0XrOouZJs1yM7JsA_y3v01NHtWZcBk?authuser=2)

### **Source Code:**

<https://github.com/PriyankaMath/CreditCardFraudDetection>

### **Presentation slides:**

<https://docs.google.com/presentation/d/1Ln00PJDhR68gJyTvdQ8JinqEa3hc2ibbIGi2eLLie6E/edit#slide=id.p>

## REFERENCES

- [1] D. Tanouz, R Raja Subramanian, D. Eswar, G V Parameswara Reddy, A. Ranjith Kumar and CH V N M Praneeth. Credit Card Fraud Detection Using Machine Learning. 5th International Conference on Intelligent Computing and Control Systems (ICICCS), 2021.
- [2] Bertrand Lebuchot, Fabian Braun, Olivier Caelen, and Marco Saerens. A graph-based, semi-supervised, credit card fraud detection system. In COMPLEX NETWORKS, 2016.
- [3] John O. Awoyemi, Adebayo Olusola Adetunmbi, and Samuel Adebayo Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. 2017 International Conference on Computing Networking and Informatics (ICCNI), pages 1–9, 2017
- [4] Piotr Juszczak, Niall M. Adams, David J. Hand, Christopher Whitrow, and David John Weston. Off-the-peg and bespoke classifiers for fraud detection. Computational Statistics Data Analysis, 52:4521–4532, 2008.
- [5] Sam Maes, Karl Tuyls, Bram Vanschoenwinkel, and Bernard Manderick. Credit card fraud detection using bayesian and neural networks. In: Maciunas RJ, editor. Interactive image-guided neurosurgery. American Association Neurological Surgeons, pages 261–270, 1993
- [6] Ronish Shakya. Application of Machine Learning Techniques in Credit Card Card Fraud Detection
- [7] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. SMOTE: synthetic minority over-sampling technique.