CREDIT CARD FRAUD DETECTION

Abstract :

The purpose of this project is to detect the fraudulent transactions made by credit cards by the use of machine learning techniques, to stop fraudsters from the unauthorized usage of customers’ accounts. The increase of credit card fraud is growing rapidly worldwide, which is the reason actions should be taken to stop fraudsters. Putting a limit for those actions would have a positive impact on the customers as their money would be recovered and retrieved back into their accounts and they won’t be charged for items or services that were not purchased by them which is the main goal of the project. Detection of the fraudulent transactions will be made by using three machine learning techniques KNN, SVM and Logistic Regression, those models will be used on a credit card transaction dataset.

Introduction

Credit card fraud is a pervasive and costly issue in today's digital economy. It occurs when unauthorized individuals use stolen or counterfeit credit card information to make fraudulent transactions, resulting in financial losses for both cardholders and financial institutions. To combat this ever-evolving threat, credit card fraud detection systems have become indispensable. These systems employ a combination of advanced technologies, data analysis, and machine learning to identify and prevent fraudulent activities. In this introduction, we will explore the key aspects of credit card fraud detection, including its importance, the methods used to detect fraud, and the role of technology in this ongoing battle against financial crime.

Credit card fraud is a pervasive and costly issue in today's digital economy. It occurs when unauthorized individuals use stolen or counterfeit credit card information to make fraudulent transactions, resulting in financial losses for both cardholders and financial institutions. To combat this ever-evolving threat, credit card fraud detection systems have become indispensable. These systems employ a combination of advanced technologies, data analysis, and machine learning to identify and prevent fraudulent activities. In this introduction, we will explore the key aspects of credit card fraud detection, including its importance, the methods used to detect fraud, and the role of technology in this ongoing battle against financial crime.

Creditcard Fraud Detection

Credit card fraud detection is a critical application of machine learning that focuses on identifying unauthorised and fraudulent transactions in credit card transactions. With the widespread use of credit cards for both online and offline purchases, the risk of fraudulent activities has increased significantly. Fraudulent transactions can result in financial loses for both cardholders and financial institutions. Machine learning, particularly supervised learning techniques, has emerged as a powerful tool to combat this issue. By training model on historical transaction data, machine learning algorithms can learn to distinguish between legitimate and fraudulent transaction based on various features such as transacton amount, location, time and more. This allows for real-time monitoring and immediate detection of suspicious activities, helping financial institutions and cardholders take timely action to mitigate potential losses. In this project, we will explore the development of a credit card fraud detection system using Support Vector Machine algorithm to enhance the security of financial transactions and protect cardholders from unauthorised charges.

**Importing relevant Python libraries**

In [16]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.utils import resample

from sklearn.preprocessing import StandardScaler

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

from sklearn.svm import SVC

from sklearn.metrics import f1\_score, precision\_score, accuracy\_score

1. Data Understanding: Loading and exploring dataset

In [17]:

df=pd.read\_csv('/kaggle/input/creditcardfraud/creditcard.csv',nrows=10000)

pd.set\_option("display.max\_columns",31)

display(df)

|  | 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | -0.470401 | 0.207971 | 0.025791 | 0.403993 | 0.251412 | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 149.62 | 0 |
| 1 | 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 | 0.463917 | -0.114805 | -0.183361 | -0.145783 | -0.069083 | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | 2.69 | 0 |
| 2 | 1 | -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 | -2.890083 | 1.109969 | -0.121359 | -2.261857 | 0.524980 | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 378.66 | 0 |
| 3 | 1 | -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 | -1.059647 | -0.684093 | 1.965775 | -1.232622 | -0.208038 | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 123.50 | 0 |
| 4 | 2 | -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 | -0.451449 | -0.237033 | -0.038195 | 0.803487 | 0.408542 | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 69.99 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 9995 | 15001 | 1.178755 | 0.596025 | 0.074131 | 2.542393 | 0.450685 | -0.179355 | 0.326365 | -0.234949 | 0.473040 | 0.331248 | -0.406853 | -3.076411 | 1.421080 | 1.885435 | -0.335264 | 0.700895 | -0.076694 | 0.158451 | -0.815829 | -0.098736 | -0.228112 | -0.561559 | -0.182781 | -0.523241 | 0.667142 | 0.015699 | -0.067238 | 0.008709 | 53.19 | 0 |
| 9996 | 15004 | 1.228455 | 0.049488 | 1.022099 | 0.386471 | -0.973228 | -1.067822 | -0.383162 | -0.205407 | 1.699304 | -0.531014 | 0.950229 | -2.357223 | 1.395353 | 1.606754 | 0.553204 | 0.429342 | 0.452575 | -0.216733 | -0.365549 | -0.121607 | -0.322912 | -0.730294 | 0.205601 | 0.686302 | -0.071008 | 0.729846 | -0.092276 | 0.012277 | 12.18 | 0 |
| 9997 | 15008 | -0.971734 | 0.744625 | 2.334822 | -0.408046 | -0.999231 | -0.629294 | -0.377212 | 0.481230 | 1.599496 | -1.586419 | 0.107872 | -2.325503 | 1.170943 | 1.271288 | -1.118945 | 0.345479 | 0.722130 | -0.026780 | -0.641398 | -0.263964 | -0.116821 | -0.141219 | -0.026115 | 0.712719 | -0.372964 | 0.750323 | -0.107875 | 0.031272 | 4.05 | 0 |
| 9998 | 15010 | -1.529666 | 1.475870 | 1.507624 | -0.662935 | -1.037152 | -1.159860 | -0.303219 | 0.745766 | 0.946896 | -1.373455 | 0.433736 | -2.364680 | 1.342560 | 2.038849 | -0.099721 | 0.782406 | 0.567083 | -0.049721 | -0.862833 | -0.239655 | -0.197993 | -0.634088 | 0.100631 | 0.669449 | -0.269750 | 0.611964 | -0.169789 | 0.007846 | 4.05 | 0 |
| 9999 | 15012 | -1.181721 | 1.485264 | 1.958715 | 2.587943 | -0.504092 | -0.126697 | 0.939038 | 0.175638 | -0.756318 | -0.482658 | 0.765963 | -1.898121 | 3.334734 | 1.668615 | -0.179474 | 0.316425 | 0.566027 | 0.020470 | -0.492420 | 0.317257 | -0.010641 | -0.191361 | 0.204004 | 0.663928 | 0.288780 | -0.033612 | -0.142682 | 0.028149 | 159.28 | 0 |

10000 rows × 31 columns

In [18]:

shape= df.shape

print("The dimension of our dataset is as follows", shape, "**\n**")

The dimension of our dataset is as follows (10000, 31)

In [19]:

print("The information about the data type of each colums is as follows: ")

df.info()

The information about the data type of each colums is as follows:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 31 columns):

# Column Non-Null Count Dtype

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

0 Time 10000 non-null int64

1 V1 10000 non-null float64

2 V2 10000 non-null float64

3 V3 10000 non-null float64

4 V4 10000 non-null float64

5 V5 10000 non-null float64

6 V6 10000 non-null float64

7 V7 10000 non-null float64

8 V8 10000 non-null float64

9 V9 10000 non-null float64

10 V10 10000 non-null float64

11 V11 10000 non-null float64

12 V12 10000 non-null float64

13 V13 10000 non-null float64

14 V14 10000 non-null float64

15 V15 10000 non-null float64

16 V16 10000 non-null float64

17 V17 10000 non-null float64

18 V18 10000 non-null float64

19 V19 10000 non-null float64

20 V20 10000 non-null float64

21 V21 10000 non-null float64

22 V22 10000 non-null float64

23 V23 10000 non-null float64

24 V24 10000 non-null float64

25 V25 10000 non-null float64

26 V26 10000 non-null float64

27 V27 10000 non-null float64

28 V28 10000 non-null float64

29 Amount 10000 non-null float64

30 Class 10000 non-null int64

dtypes: float64(29), int64(2)

memory usage: 2.4 MB

In [20]:

print("Checking the missing/NaN values from each column of the dataframe:", df.isnull().sum())

Checking the missing/NaN values from each column of the dataframe: 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

In [21]:

df.describe() *#getting the discription of our dataset*

Out[21]:

|  | 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.00000 |
| mean | 5966.033400 | -0.241862 | 0.281949 | 0.906270 | 0.264148 | -0.046398 | 0.133108 | -0.071689 | -0.064778 | 0.802224 | -0.222998 | 0.766066 | -1.272513 | 0.851410 | 0.700597 | -0.129634 | -0.007757 | 0.318991 | -0.016074 | -0.070415 | 0.027511 | -0.051990 | -0.152671 | -0.033268 | 0.021335 | 0.087146 | 0.108140 | 0.005518 | 0.002915 | 63.030188 | 0.00380 |
| std | 4473.403739 | 1.521679 | 1.308139 | 1.159154 | 1.441235 | 1.182935 | 1.307311 | 1.077430 | 1.259064 | 1.155198 | 1.093548 | 1.168600 | 1.527660 | 1.213055 | 1.239290 | 0.975573 | 0.882057 | 0.966392 | 0.794259 | 0.808373 | 0.589994 | 0.913811 | 0.631083 | 0.487814 | 0.594430 | 0.428171 | 0.562793 | 0.410868 | 0.266247 | 184.486158 | 0.06153 |
| min | 0.000000 | -27.670569 | -34.607649 | -15.496222 | -4.657545 | -32.092129 | -23.496714 | -26.548144 | -23.632502 | -6.329801 | -13.193415 | -2.595325 | -17.769143 | -3.389510 | -19.214325 | -4.152532 | -12.227189 | -18.587366 | -6.920762 | -4.932733 | -13.276034 | -11.468435 | -8.527145 | -15.144340 | -2.512377 | -2.577363 | -1.338556 | -7.976100 | -3.509250 | 0.000000 | 0.00000 |
| 25% | 2072.750000 | -1.013283 | -0.208342 | 0.412799 | -0.614424 | -0.643390 | -0.629934 | -0.542336 | -0.190747 | 0.070868 | -0.688422 | -0.063689 | -2.368115 | -0.017984 | 0.080400 | -0.709531 | -0.495536 | -0.180029 | -0.450302 | -0.552134 | -0.149981 | -0.268120 | -0.549638 | -0.174120 | -0.327817 | -0.158137 | -0.327974 | -0.084489 | -0.015753 | 5.000000 | 0.00000 |
| 50% | 4563.500000 | -0.372799 | 0.288524 | 0.944361 | 0.219861 | -0.152769 | -0.152566 | -0.055585 | 0.012865 | 0.805275 | -0.340720 | 0.746752 | -1.621015 | 0.919134 | 0.899792 | -0.010078 | 0.066086 | 0.297423 | 0.025225 | -0.077208 | -0.021415 | -0.123273 | -0.136746 | -0.045794 | 0.079976 | 0.121001 | 0.042865 | -0.004568 | 0.015897 | 15.950000 | 0.00000 |
| 75% | 10233.250000 | 1.150864 | 0.901879 | 1.602903 | 1.125666 | 0.371081 | 0.505357 | 0.476280 | 0.274533 | 1.506299 | 0.174295 | 1.576540 | 0.082667 | 1.768889 | 1.499211 | 0.533501 | 0.547399 | 0.782865 | 0.459390 | 0.442908 | 0.156534 | 0.032707 | 0.247490 | 0.081665 | 0.410877 | 0.359058 | 0.476394 | 0.120811 | 0.077182 | 50.960000 | 0.00000 |
| max | 15012.000000 | 1.960497 | 8.636214 | 4.101716 | 10.463020 | 34.099309 | 21.393069 | 34.303177 | 5.060381 | 10.392889 | 12.259949 | 12.018913 | 3.774837 | 4.465413 | 5.748734 | 3.635042 | 4.087802 | 7.893393 | 4.115560 | 4.555359 | 8.012574 | 22.588989 | 4.534454 | 13.876221 | 3.200201 | 5.525093 | 3.517346 | 8.254376 | 4.860769 | 7712.430000 | 1.00000 |

2. Exploratory Data Analysis (EDA)

**2.1 Checking class imbalance of Credit\_card fraud detection. Visualised with pie chart and histogram.**

In [22]:

target\_dictionary= df['Class'].value\_counts().to\_dict()

target\_counts=[target\_dictionary[0],target\_dictionary[1]]

colour=['green','red']

counts\_label=['Non\_Fraudulent','Fraudulent']

fig,axes= plt.subplots(1,2,figsize=(10,10)) *#creating subplots*

*#pie*

axes[0].pie(target\_counts, labels=counts\_label, colors= colour, autopct='**%1.2f%%**')

axes[0].set\_title('Credit card fraudulent/non\_fraudulent pie chart')

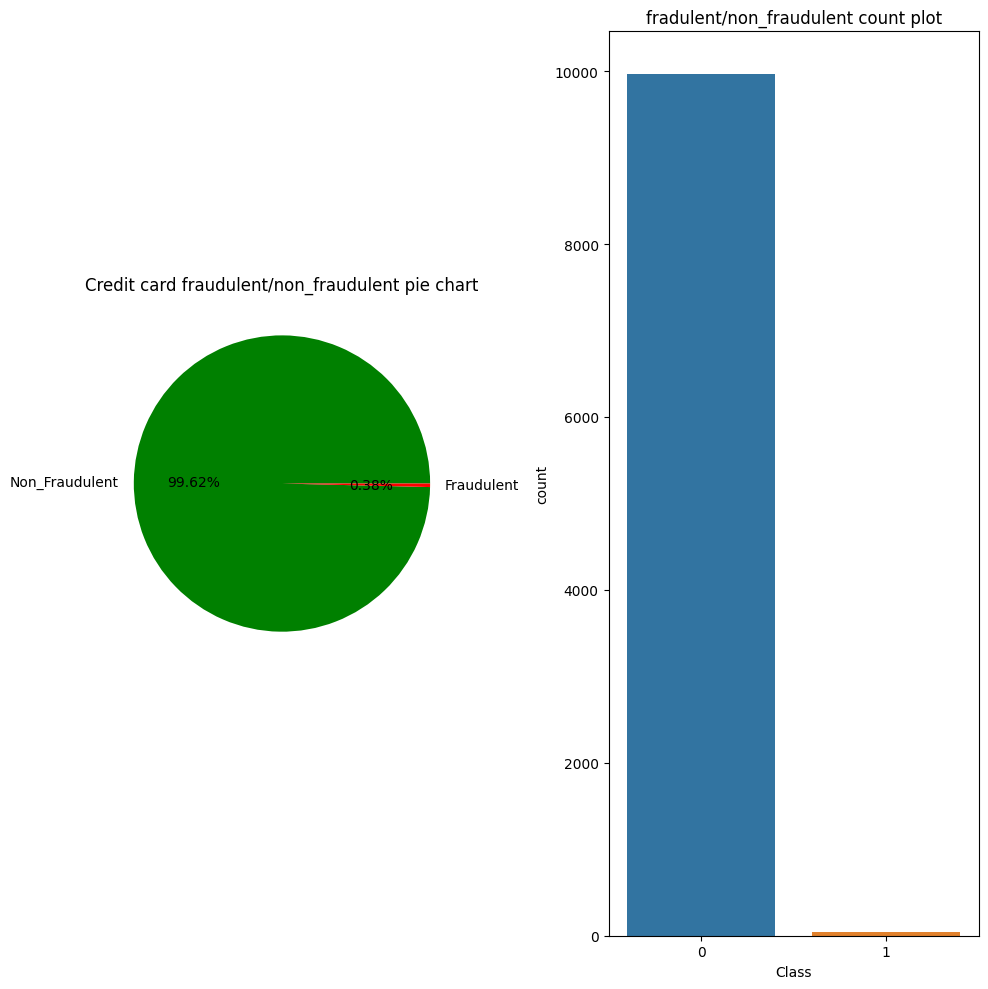
*#histogram for count plot*

sns.countplot(x=df['Class'], data=df, ax=axes[1])

axes[1].set\_title('fradulent/non\_fraudulent count plot')

plt.tight\_layout()

plt.show()



**2.2 visualising the relationship between "time" and "amount" to the target "variable"**

In [23]:

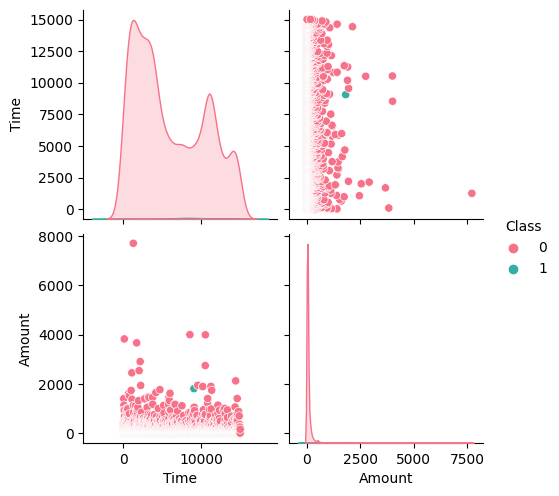
sns.pairplot(df, hue='Class',vars=['Time','Amount'], palette='husl')

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

Out[23]:

<seaborn.axisgrid.PairGrid at 0x7fb0ee6b10f0>



3. Data Preprocessing: Prepare the data modelling by handling missing values, encoding catagorical variables, scaling numerical features and handling class imbalance

**3.1 Scaling the feature (Amount) since it has not undergone PCA**

StandardScaler scales and centers features (variables) so that the have a mean of 0 and standard deviation of 1. This ensures that all features have the same scale. Algorithms like Support Vector Machine and k-means clustering are particularly sensitive to feature scaling. Standard scaler also improves the performance, stability and interpretability of machine learning models by ensuring that feature are on consistent scales.

In [24]:

scaler= StandardScaler()

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

display(df)

|  | 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | -0.470401 | 0.207971 | 0.025791 | 0.403993 | 0.251412 | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 0.469380 | 0 |
| 1 | 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 | 0.463917 | -0.114805 | -0.183361 | -0.145783 | -0.069083 | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | -0.327088 | 0 |
| 2 | 1 | -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 | -2.890083 | 1.109969 | -0.121359 | -2.261857 | 0.524980 | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 1.710945 | 0 |
| 3 | 1 | -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 | -1.059647 | -0.684093 | 1.965775 | -1.232622 | -0.208038 | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 0.327791 | 0 |
| 4 | 2 | -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 | -0.451449 | -0.237033 | -0.038195 | 0.803487 | 0.408542 | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 0.037727 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 9995 | 15001 | 1.178755 | 0.596025 | 0.074131 | 2.542393 | 0.450685 | -0.179355 | 0.326365 | -0.234949 | 0.473040 | 0.331248 | -0.406853 | -3.076411 | 1.421080 | 1.885435 | -0.335264 | 0.700895 | -0.076694 | 0.158451 | -0.815829 | -0.098736 | -0.228112 | -0.561559 | -0.182781 | -0.523241 | 0.667142 | 0.015699 | -0.067238 | 0.008709 | -0.053341 | 0 |
| 9996 | 15004 | 1.228455 | 0.049488 | 1.022099 | 0.386471 | -0.973228 | -1.067822 | -0.383162 | -0.205407 | 1.699304 | -0.531014 | 0.950229 | -2.357223 | 1.395353 | 1.606754 | 0.553204 | 0.429342 | 0.452575 | -0.216733 | -0.365549 | -0.121607 | -0.322912 | -0.730294 | 0.205601 | 0.686302 | -0.071008 | 0.729846 | -0.092276 | 0.012277 | -0.275645 | 0 |
| 9997 | 15008 | -0.971734 | 0.744625 | 2.334822 | -0.408046 | -0.999231 | -0.629294 | -0.377212 | 0.481230 | 1.599496 | -1.586419 | 0.107872 | -2.325503 | 1.170943 | 1.271288 | -1.118945 | 0.345479 | 0.722130 | -0.026780 | -0.641398 | -0.263964 | -0.116821 | -0.141219 | -0.026115 | 0.712719 | -0.372964 | 0.750323 | -0.107875 | 0.031272 | -0.319716 | 0 |
| 9998 | 15010 | -1.529666 | 1.475870 | 1.507624 | -0.662935 | -1.037152 | -1.159860 | -0.303219 | 0.745766 | 0.946896 | -1.373455 | 0.433736 | -2.364680 | 1.342560 | 2.038849 | -0.099721 | 0.782406 | 0.567083 | -0.049721 | -0.862833 | -0.239655 | -0.197993 | -0.634088 | 0.100631 | 0.669449 | -0.269750 | 0.611964 | -0.169789 | 0.007846 | -0.319716 | 0 |
| 9999 | 15012 | -1.181721 | 1.485264 | 1.958715 | 2.587943 | -0.504092 | -0.126697 | 0.939038 | 0.175638 | -0.756318 | -0.482658 | 0.765963 | -1.898121 | 3.334734 | 1.668615 | -0.179474 | 0.316425 | 0.566027 | 0.020470 | -0.492420 | 0.317257 | -0.010641 | -0.191361 | 0.204004 | 0.663928 | 0.288780 | -0.033612 | -0.142682 | 0.028149 | 0.521744 | 0 |

10000 rows × 31 columns

**3.2 Balancing our class through resampling since non-fraud outcomes far surpasses fraud outcome**

In [25]:

minority\_class= df[df['Class']==1]

majority\_class= df[df['Class']==0]

minority\_upsampling =resample(minority\_class,

replace=True,

n\_samples= len(majority\_class),

random\_state=42)

upsampling\_concat= pd.concat([minority\_upsampling, majority\_class])

df= upsampling\_concat.sort\_index() *#sort\_index sorts the data according to its index*

display(df)

|  | 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | -0.470401 | 0.207971 | 0.025791 | 0.403993 | 0.251412 | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 0.469380 | 0 |
| 1 | 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 | 0.463917 | -0.114805 | -0.183361 | -0.145783 | -0.069083 | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | -0.327088 | 0 |
| 2 | 1 | -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 | -2.890083 | 1.109969 | -0.121359 | -2.261857 | 0.524980 | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 1.710945 | 0 |
| 3 | 1 | -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 | -1.059647 | -0.684093 | 1.965775 | -1.232622 | -0.208038 | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 0.327791 | 0 |
| 4 | 2 | -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 | -0.451449 | -0.237033 | -0.038195 | 0.803487 | 0.408542 | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 0.037727 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 9995 | 15001 | 1.178755 | 0.596025 | 0.074131 | 2.542393 | 0.450685 | -0.179355 | 0.326365 | -0.234949 | 0.473040 | 0.331248 | -0.406853 | -3.076411 | 1.421080 | 1.885435 | -0.335264 | 0.700895 | -0.076694 | 0.158451 | -0.815829 | -0.098736 | -0.228112 | -0.561559 | -0.182781 | -0.523241 | 0.667142 | 0.015699 | -0.067238 | 0.008709 | -0.053341 | 0 |
| 9996 | 15004 | 1.228455 | 0.049488 | 1.022099 | 0.386471 | -0.973228 | -1.067822 | -0.383162 | -0.205407 | 1.699304 | -0.531014 | 0.950229 | -2.357223 | 1.395353 | 1.606754 | 0.553204 | 0.429342 | 0.452575 | -0.216733 | -0.365549 | -0.121607 | -0.322912 | -0.730294 | 0.205601 | 0.686302 | -0.071008 | 0.729846 | -0.092276 | 0.012277 | -0.275645 | 0 |
| 9997 | 15008 | -0.971734 | 0.744625 | 2.334822 | -0.408046 | -0.999231 | -0.629294 | -0.377212 | 0.481230 | 1.599496 | -1.586419 | 0.107872 | -2.325503 | 1.170943 | 1.271288 | -1.118945 | 0.345479 | 0.722130 | -0.026780 | -0.641398 | -0.263964 | -0.116821 | -0.141219 | -0.026115 | 0.712719 | -0.372964 | 0.750323 | -0.107875 | 0.031272 | -0.319716 | 0 |
| 9998 | 15010 | -1.529666 | 1.475870 | 1.507624 | -0.662935 | -1.037152 | -1.159860 | -0.303219 | 0.745766 | 0.946896 | -1.373455 | 0.433736 | -2.364680 | 1.342560 | 2.038849 | -0.099721 | 0.782406 | 0.567083 | -0.049721 | -0.862833 | -0.239655 | -0.197993 | -0.634088 | 0.100631 | 0.669449 | -0.269750 | 0.611964 | -0.169789 | 0.007846 | -0.319716 | 0 |
| 9999 | 15012 | -1.181721 | 1.485264 | 1.958715 | 2.587943 | -0.504092 | -0.126697 | 0.939038 | 0.175638 | -0.756318 | -0.482658 | 0.765963 | -1.898121 | 3.334734 | 1.668615 | -0.179474 | 0.316425 | 0.566027 | 0.020470 | -0.492420 | 0.317257 | -0.010641 | -0.191361 | 0.204004 | 0.663928 | 0.288780 | -0.033612 | -0.142682 | 0.028149 | 0.521744 | 0 |

19924 rows × 31 columns

4. Feature selection: selecting the most relevant features from the dataset that will be used as input variables

**By looking at the relationship between time and target variable for fraud and non-fraud, there is no clear relationship since fraud can happen anytime, so is non-fraud. Thus, we drop 'Time' column as a feature.**

In [26]:

selected\_features= df.drop(['Time'], axis=1)

display(selected\_features)

|  | 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | -0.470401 | 0.207971 | 0.025791 | 0.403993 | 0.251412 | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 0.469380 | 0 |
| 1 | 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 | 0.463917 | -0.114805 | -0.183361 | -0.145783 | -0.069083 | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | -0.327088 | 0 |
| 2 | -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 | -2.890083 | 1.109969 | -0.121359 | -2.261857 | 0.524980 | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 1.710945 | 0 |
| 3 | -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 | -1.059647 | -0.684093 | 1.965775 | -1.232622 | -0.208038 | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 0.327791 | 0 |
| 4 | -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 | -0.451449 | -0.237033 | -0.038195 | 0.803487 | 0.408542 | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 0.037727 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 9995 | 1.178755 | 0.596025 | 0.074131 | 2.542393 | 0.450685 | -0.179355 | 0.326365 | -0.234949 | 0.473040 | 0.331248 | -0.406853 | -3.076411 | 1.421080 | 1.885435 | -0.335264 | 0.700895 | -0.076694 | 0.158451 | -0.815829 | -0.098736 | -0.228112 | -0.561559 | -0.182781 | -0.523241 | 0.667142 | 0.015699 | -0.067238 | 0.008709 | -0.053341 | 0 |
| 9996 | 1.228455 | 0.049488 | 1.022099 | 0.386471 | -0.973228 | -1.067822 | -0.383162 | -0.205407 | 1.699304 | -0.531014 | 0.950229 | -2.357223 | 1.395353 | 1.606754 | 0.553204 | 0.429342 | 0.452575 | -0.216733 | -0.365549 | -0.121607 | -0.322912 | -0.730294 | 0.205601 | 0.686302 | -0.071008 | 0.729846 | -0.092276 | 0.012277 | -0.275645 | 0 |
| 9997 | -0.971734 | 0.744625 | 2.334822 | -0.408046 | -0.999231 | -0.629294 | -0.377212 | 0.481230 | 1.599496 | -1.586419 | 0.107872 | -2.325503 | 1.170943 | 1.271288 | -1.118945 | 0.345479 | 0.722130 | -0.026780 | -0.641398 | -0.263964 | -0.116821 | -0.141219 | -0.026115 | 0.712719 | -0.372964 | 0.750323 | -0.107875 | 0.031272 | -0.319716 | 0 |
| 9998 | -1.529666 | 1.475870 | 1.507624 | -0.662935 | -1.037152 | -1.159860 | -0.303219 | 0.745766 | 0.946896 | -1.373455 | 0.433736 | -2.364680 | 1.342560 | 2.038849 | -0.099721 | 0.782406 | 0.567083 | -0.049721 | -0.862833 | -0.239655 | -0.197993 | -0.634088 | 0.100631 | 0.669449 | -0.269750 | 0.611964 | -0.169789 | 0.007846 | -0.319716 | 0 |
| 9999 | -1.181721 | 1.485264 | 1.958715 | 2.587943 | -0.504092 | -0.126697 | 0.939038 | 0.175638 | -0.756318 | -0.482658 | 0.765963 | -1.898121 | 3.334734 | 1.668615 | -0.179474 | 0.316425 | 0.566027 | 0.020470 | -0.492420 | 0.317257 | -0.010641 | -0.191361 | 0.204004 | 0.663928 | 0.288780 | -0.033612 | -0.142682 | 0.028149 | 0.521744 | 0 |

19924 rows × 30 columns

5. Model Training: Split the dataset into training and testing sets

**5.1 Training the dataset with the Support Vector Machine model**

In [27]:

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

y= df['Class']

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

svm\_model = SVC(kernel='linear')

svm\_model.fit(X\_train, y\_train)

Out[27]:

SVC

SVC(kernel='linear')

6. Model Evaluation: Evaluating the performance of the trained model using appropriate evaluation metrics, i.e accuracy, precision and f1 score

In [28]:

y\_predict = svm\_model.predict(X\_test) *#predicting the testing set*

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

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

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

print('The accuracy score is: ', accuracy)

print('The precison score is: ', precision)

print('The f1 score is: ', f1)

The accuracy score is: 0.9997490589711417

The precison score is: 0.9995039682539683

The f1 score is: 0.9997519225998511

**DATA SET :**

**1.Import Libraries**: Start by importing the necessary Python libraries.

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler

1. **Load the Dataset**: Download the credit card transaction dataset and load it into a Pandas DataFrame.
2. You can use the **read\_csv** function to load a CSV file.

# Replace 'your\_dataset.csv' with the actual path to your dataset data = pd.re

ad\_csv('your\_dataset.csv')

1. **Explore the Data**: Before preprocessing, take a quick look at your data to understand its structure and check for any missing values.

# Display the first few rows of the dataset print(data.head())

# Check for missing values print(data.isnull().sum())

# Get summary statistics print(data.describe())

**4.Preprocess the Data**: Preprocessing is a crucial step in fraud detection to ensure that the data is ready for analysis. Common preprocessing steps include:

5. **Handling Imbalanced Data**: Check the balance between fraud and non-fraud transactions. If imbalanced, consider oversampling, undersampling, or using advanced techniques like Synthetic Minority Over-sampling Technique (SMOTE).

6.. **Feature Scaling**: Scale the features to have a mean of 0 and a standard deviation of 1. This is important for algorithms like logistic regression or k-nearest neighbors.

# Separate features and target variable

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

y = data['Class']

# Split the dataset into training and testing sets

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

# Standardize the data scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)

**7.Explore the Preprocessed Data**: After preprocessing, explore the data again to ensure everything looks good.

# Check the class balance print(y\_train.value\_counts())

# Visualize the data (e.g., his

tograms, scatter plots)

# You can use libraries like Matplotlib and Seaborn for this.

**8.Save Preprocessed Data**: If you want to save the preprocessed data for future use, you can use Pandas' **to\_csv** function.

X\_train\_df = pd.DataFrame(X\_train, columns=X.columns) X\_test\_df = pd.DataFrame(X\_test, columns=X.columns) X\_train\_df.to\_csv('preprocessed\_train\_data.csv', index=False) X\_test\_df.to\_csv('preprocessed\_test\_data.csv', index=False)

CODE :

# Import necessary libraries

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, roc\_auc\_score

# Step 1: Load the dataset

# Replace 'your\_dataset.csv' with the actual path to your dataset

data = pd.read\_csv('your\_dataset.csv')

# Step 2: Data Preprocessing

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)

# Step 3: Model Selection and Training

model = RandomForestClassifier()

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

# Step 4: Model Evaluation

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

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

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

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

CODE EXPLANATION :

# Import necessary libraries import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification\_report, roc\_auc\_score

In this section, we import the required libraries:

* **pandas** is used for data manipulation and handling DataFrames.
* **train\_test\_split** from **sklearn.model\_selection** is used to split the dataset into a training set and a test set.
* **RandomForestClassifier** from **sklearn.ensemble** is a machine learning model that will be used for fraud detection.
* **classification\_report** and **roc\_auc\_score** from **sklearn.metrics** are used to evaluate the model's performance.

# Step 1: Load the dataset data = pd.read\_csv('your\_dataset.csv')

Here, we load the credit card transaction dataset from a CSV file into a Pandas DataFrame. Replace **'your\_dataset.csv'** with the actual path to your dataset.

# Step 2: Data Preprocessing 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)

In this section, we perform data preprocessing:

* We separate the dataset into the feature matrix **X** (containing all columns except 'Class') and the target variable **y** (which is the 'Class' column, indicating whether a transaction is fraudulent or not).
* We split the data into a training set (**X\_train** and **y\_train**) and a test set (**X\_test** and **y\_test**) using **train\_test\_split**. The test set is 20% of the data, and we set a random seed (**random\_state**) for reproducibility.

# Step 3: Model Selection and Training model = RandomForestClassifier() model.fit(X\_train, y\_train)

In this section, we select a machine learning model for fraud detection. We chose a **RandomForestClassifier**, which is an ensemble learning method based on decision trees. We then fit (train) the model using the training data.

# Step 4: Model Evaluation y\_pred = model.predict(X\_test) print(classification\_report(y\_test, y\_pred)) roc\_auc = roc\_auc\_score(y\_test, y\_pred) print("ROC AUC:", roc\_auc)

In this final part, we evaluate the performance of the trained model:

* We make predictions on the test set using **model.predict(X\_test)**.
* We print a classification report, which provides metrics such as precision, recall, F1-score, and support for both classes (fraudulent and non-fraudulent).
* We calculate the ROC AUC score, which is a measure of how well the model distinguishes between the two classes. A higher ROC AUC indicates better performance

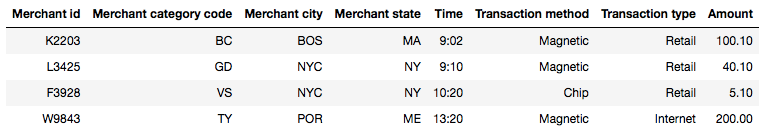
FEATURE ENGINEERING:



Feature engineering is the key to success in modeling.

Credit card fraud happens basically in two types: application fraud and transaction fraud. Application fraud is similar to identity fraud in that one person uses another person’s data to obtain a new card. Transaction fraud happens when a card is stolen or a lost card is obtained to conduct fraudulent transactions. Also, there has been a significant rise in counterfeit cards.

Feature Technicalities:

* **PCA Transformation:**The description of the data says that all the features went through a PCA transformation (Dimensionality Reduction technique) (Except for time and amount).
* **Scaling:** Keep in mind that in order to implement a PCA transformation features need to be previously scaled. (In this case, all the V features have been scaled or at least that is what we are assuming the people that develop the dataset did.)
* A fraudster will try to abuse the card as much as possible in a **short period** before the card is detected and suspended. So we should see **abnormal**transactions in a short period. With this goal, if we aggregate transactions over some time, we shall be able to discover abrupt changes.
* Let me present an example to demonstrate how features can be created. The table shows some transactions of a cardholder. The transactional data include the merchant id, the category of the merchant, the location of the merchant, a timestamp, the transaction method and type, and the transaction amount.
* 

MODEL TRAINING :

These applications are used as machine learning for credit card fraud detection:

Decision Tree and Random Forest

Artificial Neural Network

Naive Bayes

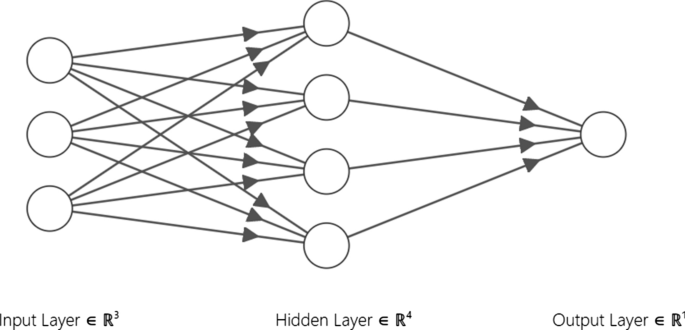
### Decision trees and random forest

Decision Tree (DT) is a supervised ML based approach that is utilized to solve regression and classification tasks. A DT contains the following types of nodes: root node, decision node and leaf node. The root node is the starting point of the algorithm. The decision node is a point whereby a choice is made in order to split the tree. A leaf node represents a final decision [[7](https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8#ref-CR7)]. The RF method conducts its predictions by using an ensemble of DTs [[8](https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8#ref-CR8)]. In the RF, a decision is reached by majority vote.

Given a number of trees *k*, a RF is defined as, RF = {�(�,��)}, where {��} represents independent identically distributed trees that cast a vote on input vector *X*. The label with the most votes is the prediction.

### Artificial Neural Network

Artificial Neural Network (ANN) is a supervised ML method that is inspired from the inner workings of the human brain. The simplest ANN have the following basic structure: an input layer, one hidden layer and an output layer. The input layer size is based on the number of features in a given dataset. The hidden layer size can be varied based on the complexity of a task and the output layer size depends on the type of problems to be solved. The most basic component of an ANN is a node or neuron. In this research, we consider feed forward ANNs. Therefore, the information flows in one direction (from its input to its output) through a neuron [[12](https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8#ref-CR12)]. Figure [1](https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8#Fig1) depicts a graphical representation of a simple ANN with 3 nodes in the input layer, a hidden layer with 4 nodes and an output layer with 1 node.

[](https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8/figures/1)

### Naive Bayes

The Naive Bayes (NB) is a supervised ML technique that is based on Bayes’ theorem. The NB method assumes the independence of each pair of attributes when provided with the dependant variable (the class). In this research, the Gaussian NB (GNB) classifier was used.

EVALUATION :

To evaluate this machine learning models we considered

two different method namely;

(1) Classification accuracy, which is the ratio of number

of correct prediction to the number of input sample,

as seen in equation 6. But this is very effective only if

there are equal number of samples in each class.

Accuracy = no. of correct prediction/total no. of prediction

(2) Confusion Matrix:

This gives a matrix as output and describe the complete performance of the model. Four

essential measurements are utilized in evaluating the

analyses, to be specific True Positive Ratio (TPR),

True Negative Ratio (TNR), False Positive Ratio

(FPR) and False Negative Ratio (FNR) rates metric

individually.

In which true positive, true negative, false positive and

false negative are the quantity characterized by true positive,

false positive, true negative, and false negative experiments.