TABLE OF CONTENTS

PROBLEM STATEMENT

Background:

Ethereum, one of the famous cryptocurrencies, has been growing since 2009. Along with the popularity of Ethereum, the frauds and scams using this cryptocurrency are skyrocketing.

A few famous attacks are DAO (Decentralized Autonomous Organization), Ponzi and Pyramid schemes, and Phishing attacks.

Hustlers have adapted offline schemes to this new ecology by relying on the anonymity offered by the blockchain. Consequently, Ponzi scams posing as safe investment plans are widely available on Ethereum. Before failing and leaving the final investors with nothing, they repay early investors with money from the later investors. They are generating thousands of victims on Ethereum while stealing millions of dollars' worth of Ether, which is illegal in the real world.

In addition to causing financial losses, these fraudulent acts also erode public confidence in blockchain technology.

Objective and Significance:

The aim of this project is to make Ethereum users and investors safer by lowering the incidence of fraudulent activities. The project can remain ahead of new risks by utilizing data-driven strategies and equipping the neighborhood with the knowledge and resources necessary to safeguard their assets. Maintaining the cryptocurrency market's integrity is crucial to its long-term viability and to building confidence among existing and future investors.

Potential Contribution and Critical Importance:

This project has the potential to significantly contribute in the following ways to the issue domain of Ethereum fraud:

Data-Driven Insights: This project can offer insights into new fraud trends and strategies by applying data analysis as well as machine learning approaches. Users and developers of Ethereum can utilize this data to keep ahead of scams.

Preventive Measures: The project may suggest and put into action preventive measures including enhanced security procedures, decentralized verification of identities, and detection of fraud algorithms based on the insights acquired.

Instructional Materials: To assist users in identifying and avoiding fraudulent activity on the Ethereum network, the project can provide instructional materials and awareness campaigns.

Collaboration: The project can encourage a collaborative strategy for battling fraud within the ecosystem by cooperating with Ethereum developers, exchanges, and regulatory agencies.

DATA CLEANING

DATA COLLECTION:

Data Sources:

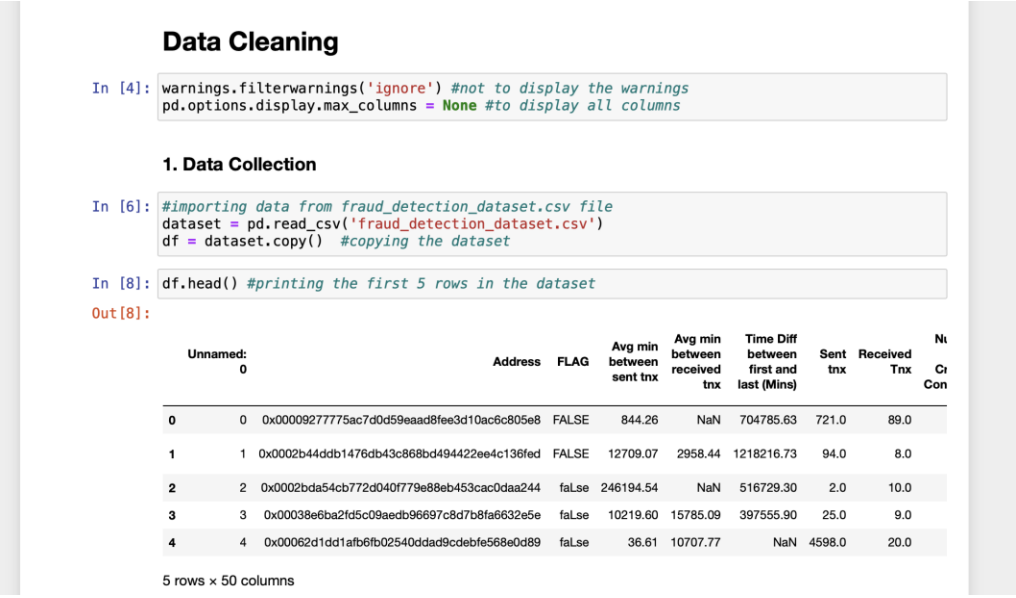
We have used the dataset "Ethereum_Dataset" from Kaggle.

https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset/data

Data Collection Methodology:

REST API's and Web Scraping is used to collect the data. And the sources are Etherscan API, ehterscamdb API.

The data is from November 2022 – September 2023



Data Cleaning

In [4]:
```
warnings.filterwarnings('ignore') #not to display the warnings
pd.options.display.max_columns = None #to display all columns
```

1. Data Collection

In [6]:
```
#importing data from fraud_detection_dataset.csv file
dataset = pd.read_csv('fraud_detection_dataset.csv')
df = dataset.copy()  #copying the dataset
```

In [8]:
```
df.head() #printing the first 5 rows in the dataset
```

Out[8]:

	Unnamed: 0		Address	FLAG	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Nu Cr Con
0	0	0x00009277775ac7d0d59eaad8fee3d10ac6c805e8		FALSE	844.26	NaN	704785.63	721.0	89.0	
1	1	0x0002b44ddb1476db43c868bd494422ee4c136fed		FALSE	12709.07	2958.44	1218216.73	94.0	8.0	
2	2	0x0002bda54cb772d040f779e88eb453cac0daa244		faLse	246194.54	NaN	516729.30	2.0	10.0	
3	3	0x00038e6ba2fd5c09aedb96697c8d7b8fa6632e5e		faLse	10219.60	15785.09	397555.90	25.0	9.0	
4	4	0x00062d1dd1afb6fb02540ddad9cdebfe568e0d89		faLse	36.61	10707.77	NaN	4598.0	20.0	

5 rows × 50 columns

Fig 1 Data Cleaning

DATA INSPECTION:

Data Summary:

The dataset consists of 10241 rows along with 50 features.

The following are the features that are in the dataset.

2. Data Inspection

```
In [9]: df.shape #dimensions of the dataset
Out[9]: (10241, 50)

In [10]: df.columns #names of the columns
Out[10]: Index(['Unnamed: 0', 'Address', 'FLAG', 'Avg min between sent tnx',
       'Avg min between received tnx',
       'Time Diff between first and last (Mins)', 'Sent tnx', 'Received Tnx',
       'Number of Created Contracts', 'Unique Received From Addresses',
       'Unique Sent To Addresses', 'min value received', 'max value received ',
       'avg val received', 'min val sent', 'max val sent', 'avg val sent',
       'min value sent to contract', 'max val sent to contract',
       'avg value sent to contract',
       'total transactions (including tnx to create contract',
       'total Ether sent', 'total ether received',
       'total ether sent contracts', 'total ether balance',
       ' Total ERC20 tnxs', ' ERC20 total Ether received',
       ' ERC20 total ether sent', ' ERC20 total Ether sent contract',
       ' ERC20 uniq sent addr', ' ERC20 uniq rec addr',
       ' ERC20 uniq sent addr.1', ' ERC20 uniq rec contract addr',
       ' ERC20 avg time between sent tnx', ' ERC20 avg time between rec tnx',
       ' ERC20 avg time between rec 2 tnx',
       ' ERC20 avg time between contract tnx', ' ERC20 min val rec',
       ' ERC20 max val rec', ' ERC20 avg val rec', ' ERC20 min val sent',
       ' ERC20 max val sent', ' ERC20 avg val sent',
       ' ERC20 min val sent contract', ' ERC20 max val sent contract',
       ' ERC20 avg val sent contract', ' ERC20 uniq sent token name',
       ' ERC20 uniq rec token name', ' ERC20 most sent token type',
       ' ERC20_most_rec_token_type'],
      dtype='object')
```

Fig 2 Data Inspection

The detailed summary of the columns to check for nulls and Data types.

```
In [13]: df.info() # Precise summary of the features
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10241 entries, 0 to 10240
Data columns (total 50 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   Unnamed: 0                                            10241 non-null  int64
 1   Address                                               10241 non-null  object
 2   FLAG                                                  10241 non-null  object
 3   Avg min between sent tnx                              9964 non-null   float64
 4   Avg min between received tnx                          10110 non-null  float64
 5   Time Diff between first and last (Mins)               10225 non-null  float64
 6   Sent tnx                                              9758 non-null   float64
 7   Received Tnx                                          9817 non-null   float64
 8   Number of Created Contracts                           10011 non-null  float64
 9   Unique Received From Addresses                        10015 non-null  float64
 10  Unique Sent To Addresses                              10137 non-null  float64
 11  min value received                                    9783 non-null   float64
 12  max value received                                    9897 non-null   float64
 13  avg val received                                      10151 non-null  float64
 14  min val sent                                          10011 non-null  float64
 15  max val sent                                          10091 non-null  float64
 16  avg val sent                                          9974 non-null   float64
 17  min value sent to contract                            9682 non-null   float64
 18  max val sent to contract                              9950 non-null   float64
 19  avg value sent to contract                            9922 non-null   float64
 20  total transactions (including tnx to create contract  9829 non-null   float64
 21  total Ether sent                                      9687 non-null   float64
 22  total ether received                                  9747 non-null   float64
 23  total ether sent contracts                            10102 non-null  float64
 24  total ether balance                                   10087 non-null  float64
 25   Total ERC20 tnxs                                     9109 non-null   float64
 26   ERC20 total Ether received                           8952 non-null   float64
```

Fig 3 Column details

The Detail description of the columns:

- Index: the index number of a row
- Address: the address of the ethereum account
- FLAG: whether the transaction is fraud or not
- Avg min between sent tnx: Average time between sent transactions for account in minutes
- Avg_min_between_received_tnx: Average time between received transactions for account in minutes
- Time_Diff_between_first_and_last(Mins): Time difference between the first and last transaction
- Sent_tnx: Total number of sent normal transactions
- Received_tnx: Total number of received normal transactions
- Number_of_Created_Contracts: Total Number of created contract transactions
- Unique_Received_From_Addresses: Total Unique addresses from which account received transactions
- Unique_Sent_To_Addresses20: Total Unique addresses from which account sent transactions
- Min_Value_Received: Minimum value in Ether ever received
- Max_Value_Received: Maximum value in Ether ever received
- Avg_Value_Received5Average value in Ether ever received
- Min_Val_Sent: Minimum value of Ether ever sent
- Max_Val_Sent: Maximum value of Ether ever sent
- Avg_Val_Sent: Average value of Ether ever sent
- Min_Value_Sent_To_Contract: Minimum value of Ether sent to a contract

- Max_Value_Sent_To_Contract: Maximum value of Ether sent to a contract
- Avg_Value_Sent_To_Contract: Average value of Ether sent to contracts
- Total_Transactions(Including_Tnx_to_Create_Contract): Total number of transactions
- Total_Ether_Sent:Total Ether sent for account address
- Total_Ether_Received: Total Ether received for account address
- Total_Ether_Sent_Contracts: Total Ether sent to Contract addresses
- Total_Ether_Balance: Total Ether Balance following enacted transactions
- Total_ERC20_Tnxs: Total number of ERC20 token transfer transactions
- ERC20_Total_Ether_Received: Total ERC20 token received transactions in Ether
- ERC20_Total_Ether_Sent: Total ERC20token sent transactions in Ether
- ERC20_Total_Ether_Sent_Contract: Total ERC20 token transfer to other contracts in Ether
- ERC20_Uniq_Sent_Addr: Number of ERC20 token transactions sent to Unique account addresses
- ERC20_Uniq_Rec_Addr: Number of ERC20 token transactions received from Unique addresses
- ERC20_Uniq_Rec_Contract_Addr: Number of ERC20token transactions received from Unique contract addresses
- ERC20_Avg_Time_Between_Sent_Tnx: Average time between ERC20 token sent transactions in minutes
- ERC20_Avg_Time_Between_Rec_Tnx: Average time between ERC20 token received transactions in minutes
- ERC20_Avg_Time_Between_Contract_Tnx: Average time ERC20 token between sent token transactions
- ERC20_Min_Val_Rec: Minimum value in Ether received from ERC20 token transactions for account
- ERC20_Max_Val_Sent: Maximum value in Ether sent from ERC20 token transactions for account
- ERC20_Avg_Val_Sent: Average value in Ether sent from ERC20 token transactions for account
- ERC20_Uniq_Sent_Token_Name: Number of Unique ERC20 tokens transferred
- ERC20_Uniq_Rec_Token_Name: Number of Unique ERC20 tokens received
- ERC20_Most_Sent_Token_Type: Most sent token for account via ERC20 transaction
- ERC20_Most_Rec_Token_Type: Most received token for account via ERC20 transactions

The datatypes of all the features are as follows:

```
In [12]: df.dtypes #type of each column

Out[12]: Unnamed: 0                                          int64
         Address                                            object
         FLAG                                               object
         Avg min between sent tnx                          float64
         Avg min between received tnx                      float64
         Time Diff between first and last (Mins)           float64
         Sent tnx                                          float64
         Received Tnx                                      float64
         Number of Created Contracts                       float64
         Unique Received From Addresses                    float64
         Unique Sent To Addresses                          float64
         min value received                                float64
         max value received                                float64
         avg val received                                  float64
         min val sent                                      float64
         max val sent                                      float64
         avg val sent                                      float64
         min value sent to contract                        float64
         max val sent to contract                          float64
         avg value sent to contract                        float64
         total transactions (including tnx to create contract float64
         total Ether sent                                  float64
         total ether received                              float64
         total ether sent contracts                        float64
         total ether balance                               float64
          Total ERC20 tnxs                                 float64
          ERC20 total Ether received                       float64
          ERC20 total ether sent                           float64
          ERC20 total Ether sent contract                  float64
          ERC20 uniq sent addr                             float64
          ERC20 uniq rec addr                              float64
          ERC20 uniq sent addr.1                           float64
```

click to scroll output; double click to hide

Fig 4 Datatypes of columns

DROPPING UNNECESSARY COLUMN:

We examined and discovered that in the first column, "Unnamed: 0", we could not identify a purpose for the column so dropped the column.

3. Dropping first unnecessary 'Unnamed : 0' feature

```
In [15]: df.drop(columns = 'Unnamed: 0', inplace = True)

In [17]: df.shape
Out[17]: (10241, 49)
```

Fig 5 Dropped unnecessary column

HANDLING THE COLUMN NAME:

- We examined and discovered that the column name ‘FLAG’ is used to report the status of the fraud. So, we renamed it to the “ Fraud_Status”
- We examined and discovered that the feature names have white spaces in the column names which can be observed in the Fig2

4.1 Changing the Target feature name from 'FLAG' to 'fraud_status'

4.2 Removing space from the feature names

```
In [24]: df = df.rename(columns = {'FLAG': 'fraud_status'}) #renaming target variable name

In [25]: df.columns = df.columns.str.strip() #Eliminating whitespace from the feature names.

In [26]: df.columns #printing column names

Out[26]: Index(['Address', 'fraud_status', 'Avg min between sent tnx',
       'Avg min between received tnx',
       'Time Diff between first and last (Mins)', 'Sent tnx', 'Received Tnx',
       'Number of Created Contracts', 'Unique Received From Addresses',
       'Unique Sent To Addresses', 'min value received', 'max value received',
       'avg val received', 'min val sent', 'max val sent', 'avg val sent',
       'min value sent to contract', 'max val sent to contract',
       'avg value sent to contract',
       'total transactions (including tnx to create contract',
       'total Ether sent', 'total ether received',
       'total ether sent contracts', 'total ether balance', 'Total ERC20 tnxs',
       'ERC20 total Ether received', 'ERC20 total ether sent',
       'ERC20 total Ether sent contract', 'ERC20 uniq sent addr',
       'ERC20 uniq rec addr', 'ERC20 uniq sent addr.1',
       'ERC20 uniq rec contract addr', 'ERC20 avg time between sent tnx',
       'ERC20 avg time between rec tnx', 'ERC20 avg time between rec 2 tnx',
       'ERC20 avg time between contract tnx', 'ERC20 min val rec',
       'ERC20 max val rec', 'ERC20 avg val rec', 'ERC20 min val sent',
       'ERC20 max val sent', 'ERC20 avg val sent',
       'ERC20 min val sent contract', 'ERC20 max val sent contract',
       'ERC20 avg val sent contract', 'ERC20 uniq sent token name',
       'ERC20 uniq rec token name', 'ERC20 most sent token type',
       'ERC20_most_rec_token_type'],
      dtype='object')
```

Fig 6 Handled the column name

HANDLING DUPLICATES:

We examined and discovered that the dataset contains few duplicates. So, we dropped them.

5. Handling Duplicate Samples (rows)

```
In [65]: #Checking for duplicates
         if df.duplicated().any():
             print('Dataset has duplicates.')
         else:
             print('No duplicates')

         Dataset has duplicates.

In [66]: df = df.drop_duplicates() #dropping duplicates

In [67]: #Re-checking duplicates
         if df.duplicated().any():
             print('Dataset has duplicates.')
         else:
             print('Data frame has no duplicates')

         Data frame has no duplicates
```

Fig 7 Handled Duplicates

CONVERTING 'CATEGORY' FEATURE VALUES TO LOWERCASE:

We examined and discovered that the dataset contains data that is not in same case across dataset. So, we changed all the 'category' to lowercase.

7. Converting 'category' feature values to lowercase

```
In [82]: for i in categorical_columns[: :-1]:
             df[i] = df[i].str.lower()
             print(f"The values of {i} are")
             print(df.loc[:10, i])     #printing first 10 rows vlaues
```

```
The values of Address are
0     0x00009277775ac7d0d59eaad8fee3d10ac6c805e8
1     0x0002b44ddb1476db43c868bd494422ee4c136fed
2     0x0002bda54cb772d040f779e88eb453cac0daa244
3     0x00038e6ba2fd5c09aedb96697c8d7b8fa6632e5e
4     0x00062d1dd1afb6fb02540ddad9cdebfe568e0d89
5     0x000895ad78f4403ecd9468900e68d6ee506136fd
6     0x000d63fc5df52b0204374c2f5a3249779805d5d1
7     0x000e001ab444fa8d6dc4a402f8d7cfc88fe8c64d
8     0x0012cb699c836049a4bbeaac2d8c4d47c688e0e4
9     0x0012f247c9f980eea0a9ad06893bfd95c3145794
10    0x0013e58a315d2e728f11630aa40abfbdddcab304
Name: Address, dtype: object
The values of fraud_status are
0     false
1     false
```

Fig 8 Converted case type of the data

BINARY ENCODING OF TARGET VARIABLES

We examined and discovered that the dataset contains data variables that are not uniform throughout the dataset. We have used binary encoding to make it to either ‘0’or ‘1’. We made changes in a way that 0 denotes Not fraud, 1 denotes fraud.

7. Binary Encoding of Target variable

```
In [90]: df['fraud_status'].unique() #finding the unique values in the column

Out[90]: array(['false', 'n', 'f', 'true'], dtype=object)

In [91]: replacing_value = {'false': '0', 'n' : '0', 'f' : '0', 'true' : '1'}
         replacing_value

Out[91]: {'false': '0', 'n': '0', 'f': '0', 'true': '1'}

In [92]: df['fraud_status'] = df['fraud_status'].replace(replacing_value) #replacing the value

In [94]: #re-checking the column values
         df['fraud_status'].unique()

Out[94]: array(['0', '1'], dtype=object)

In [95]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9841 entries, 0 to 9840
Data columns (total 49 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   Address                                         9841 non-null   object
 1   fraud_status                                    9841 non-null   object
 2   Avg min between sent tnx                        9572 non-null   float64
 3   Avg min between received tnx                    9718 non-null   float64
 4   Time Diff between first and last (Mins)         9825 non-null   float64
 5   Sent tnx                                        9383 non-null   float64
 6   Received Tnx                                    9436 non-null   float64
 7   Number of Created Contracts                     9617 non-null   float64
 8   Unique Received From Addresses                  9621 non-null   float64
 9   Unique Sent To Addresses                        9743 non-null   float64
 10  min value received                              9400 non-null   float64
```

Fig 9 Binary encoding of target variables

CONVERTING ‘STRING’ features to ‘Category’

We examined and discovered that the dataset contains data that is in ‘string’ datatype. We changed it to the ‘Category’. We did this because the category data is more readable than

string. Memory is effectively used in category. And for visualization category datatype is more convenient than string

8. Converting 'str' features to 'catergory'

```
In [101]: rical_columns = df.select_dtypes(include = ['object']) #storing 'object' dtype feature names

In [102]: #converting all object types to 'category' type
          for i in categorical_columns:
              df[i] = df[i].astype('category')

In [103]: df.info()

          <class 'pandas.core.frame.DataFrame'>
          Int64Index: 9841 entries, 0 to 9840
          Data columns (total 49 columns):
           #   Column                                                Non-Null Count  Dtype
          ---  ------                                                --------------  -----
           0   Address                                               9841 non-null   category
           1   fraud_status                                          9841 non-null   category
           2   Avg min between sent tnx                              9572 non-null   float64
           3   Avg min between received tnx                          9718 non-null   float64
           4   Time Diff between first and last (Mins)               9825 non-null   float64
           5   Sent tnx                                              9383 non-null   float64
           6   Received Tnx                                          9436 non-null   float64
           7   Number of Created Contracts                           9617 non-null   float64
           8   Unique Received From Addresses                        9621 non-null   float64
           9   Unique Sent To Addresses                              9743 non-null   float64
           10  min value received                                    9400 non-null   float64
           11  max value received                                    9517 non-null   float64
           12  avg val received                                      9754 non-null   float64
           13  min val sent                                          9618 non-null   float64
           14  max val sent                                          9698 non-null   float64
           15  avg val sent                                          9587 non-null   float64
           16  min value sent to contract                            9304 non-null   float64
           17  max val sent to contract                              9564 non-null   float64
           18  avg value sent to contract                            9536 non-null   float64
           19  total transactions (including tnx to create contract  9444 non-null   float64
           20  total Ether sent                                      9313 non-null   float64
           21  total ether received                                  9367 non-null   float64
           22  total ether sent contracts                            9709 non-null   float64
```

Fig 10 Datatypes of the columns post changes

HANDLING THE MISSING VALUES:

We examined and discovered that the dataset contains few missing values. So, we checked and found that all the missing values were of ‘int’, ‘category’ and ‘float’ datatypes.

9.1 - We replaced the null values with the mean of the column for the ‘int’ and ‘float’ datatypes

9.2 - We replaced the null values with the mode / frequently occurred value of the column for the ‘Category’ datatype

9. Handling Missing Values

```
In [104]: df.isnull().sum()
```

```
Out[104]: Address                                    0
          fraud_status                               0
          Avg min between sent tnx                 269
          Avg min between received tnx             123
          Time Diff between first and last (Mins)   16
          Sent tnx                                 458
          Received Tnx                             405
          Number of Created Contracts              224
          Unique Received From Addresses           220
          Unique Sent To Addresses                  98
          min value received                       441
          max value received                       324
          avg val received                          87
          min val sent                             223
          max val sent                             143
          avg val sent                             254
          min value sent to contract               537
          max val sent to contract                 277
          avg value sent to contract               305
```

Fig 11 Count of missing values of all the columns

```
In [107]: # Storing numerical feature columns names
          numerical_columns = df.select_dtypes(include = ['int', 'float']).columns
          numerical_columns
```

```
Out[107]: Index(['Avg min between sent tnx', 'Avg min between received tnx',
                 'Time Diff between first and last (Mins)', 'Sent tnx', 'Received Tnx',
                 'Number of Created Contracts', 'Unique Received From Addresses',
                 'Unique Sent To Addresses', 'min value received', 'max value received',
                 'avg val received', 'min val sent', 'max val sent', 'avg val sent',
                 'min value sent to contract', 'max val sent to contract',
                 'avg value sent to contract',
                 'total transactions (including tnx to create contract',
                 'total Ether sent', 'total ether received',
                 'total ether sent contracts', 'total ether balance', 'Total ERC20 tnxs',
                 'ERC20 total Ether received', 'ERC20 total ether sent',
                 'ERC20 total Ether sent contract', 'ERC20 uniq sent addr',
                 'ERC20 uniq rec addr', 'ERC20 uniq sent addr.1',
                 'ERC20 uniq rec contract addr', 'ERC20 avg time between sent tnx',
                 'ERC20 avg time between rec tnx', 'ERC20 avg time between rec 2 tnx',
                 'ERC20 avg time between contract tnx', 'ERC20 min val rec',
                 'ERC20 max val rec', 'ERC20 avg val rec', 'ERC20 min val sent',
                 'ERC20 max val sent', 'ERC20 avg val sent',
                 'ERC20 min val sent contract', 'ERC20 max val sent contract',
                 'ERC20 avg val sent contract', 'ERC20 uniq sent token name',
                 'ERC20 uniq rec token name'],
                dtype='object')
```

9.1 Filling numerical null values with mean

```
In [110]: for i in numerical_columns:
              df[i].fillna(df[i].mean(), inplace = True)
```

```
In [111]: #re-checking the null values of the numerical columns
          df.isnull().sum()
```

```
Out[111]: Address                                    0
          fraud_status                               0
          Avg min between sent tnx                   0
          Avg min between received tnx               0
          Time Diff between first and last (Mins)    0
          Sent tnx                                   0
```

Fig 12 Count of missing values of the columns post replacing them with mean

9.2 Filling Categorical values with Most frequent

```
In [120]: for i in categorical_columns:
              df[i].fillna(df[i].mode().iloc[0], inplace = True)

In [126]: #re-checking the null values of the categorical columns
          for i in categorical_columns:
              print(f"Number of null values of {i} : {df[i].isnull().sum()}")

Number of null values of Address : 0
Number of null values of fraud_status : 0
Number of null values of ERC20 most sent token type : 0
Number of null values of ERC20_most_rec_token_type : 0
```

Fig 13 Count of missing values of the columns post replacing them with most frequent data (MODE)

SETTING THE VALUES WITH PROPER PRECISION:

We examined and discovered that the dataset contains few decimal values that have more digits post decimal point. It seems to be very cluttered, so we have rounded that to 2 points to make to more readable.

10. Set Proper Precision

```
In [128]: df[numerical_columns] #Values doesn't have proper precision
```

Out[128]:

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Unique Sent To Addresses	min value received	max value received
0	844.26	8011.073909	7.047856e+05	721.0	89.0	0.0	40.0	118.0	0.000000	45.806785
1	12709.07	2958.440000	1.218217e+06	94.0	8.0	0.0	5.0	14.0	0.000000	2.613269
2	246194.54	8011.073909	5.167293e+05	2.0	10.0	0.0	10.0	2.0	0.113119	1.165453
3	10219.60	15785.090000	3.975559e+05	25.0	9.0	0.0	7.0	13.0	0.000000	500.000000
4	36.61	10707.770000	2.181936e+05	4598.0	20.0	1.0	7.0	19.0	43.643659	12.802411
...	...	...	...	...	...	...	...	...	...	...
9836	12635.10	631.390000	5.874848e+04	4.0	13.0	0.0	11.0	4.0	0.004082	12.000000
9837	0.00	8011.073909	0.000000e+00	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000
9838	2499.44	2189.290000	2.616019e+05	67.0	43.0	0.0	31.0	44.0	0.001078	21.310000
9839	0.00	0.000000	0.000000e+00	0.0	1.0	0.0	1.0	0.0	0.500000	0.500000
9840	37242.70	149.560000	6.708173e+05	18.0	3.0	0.0	1.0	5.0	0.795233	18998.000000

9841 rows × 45 columns

```
In [131]: for i in numerical_columns: #rounding the values to 0.02
              df[i] = df[i].round(2)
```

Fig 14 The cluttered data with many decimal values

In [131]:
```
for i in numerical_columns: #rounding the values to 0.02
    df[i] = df[i].round(2)
```

In [132]:
```
df[numerical_columns]
```

Out[132]:

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Unique Sent To Addresses	min value received	max value received	avg val received	mi v se
0	844.26	8011.07	704785.63	721.0	89.0	0.0	40.0	118.0	0.00	45.81	6.59	0.0
1	12709.07	2958.44	1218216.73	94.0	8.0	0.0	5.0	14.0	0.00	2.61	0.39	0.0
2	246194.54	8011.07	516729.30	2.0	10.0	0.0	10.0	2.0	0.11	1.17	0.36	0.0
3	10219.60	15785.09	397555.90	25.0	9.0	0.0	7.0	13.0	0.00	500.00	99.49	0.0
4	36.61	10707.77	218193.63	4598.0	20.0	1.0	7.0	19.0	43.64	12.80	2.67	0.0
...	...	...	...	...	...	...	...	...	...	...	...	
9836	12635.10	631.39	58748.48	4.0	13.0	0.0	11.0	4.0	0.00	12.00	2.82	0.4
9837	0.00	8011.07	0.00	0.0	0.0	0.0	0.0	0.0	0.00	0.00	0.00	0.0
9838	2499.44	2189.29	261601.88	67.0	43.0	0.0	31.0	44.0	0.00	21.31	1.23	0.0
9839	0.00	0.00	0.00	0.0	1.0	0.0	1.0	0.0	0.50	0.50	0.50	0.0
9840	37242.70	149.56	670817.33	18.0	3.0	0.0	1.0	5.0	0.80	18998.00	6333.27	0.9

9841 rows × 45 columns

In [ ]:

Fig 13 The de-cluttered data that is rounded to 2 decimal values

EXPLORATORY DATA ANALYSIS(EDA)

SUMMARY STATISTICS:

DESCRIBE ():

We used describe() to get statistical data. That function provides an informative summary of the important statistics and qualities of a dataset's numerical columns such as Mean, Standard deviation, count, min, Interquartile.

Exploratory Data Analysis

```
In [133]: df.head()
```

Out[133]:

:20 val :ec	ERC20 min val sent	ERC20 max val sent	ERC20 avg val sent	ERC20 min val sent contract	ERC20 max val sent contract	ERC20 avg val sent contract	ERC20 uniq sent token name	ERC20 uniq rec token name	ERC20 most sent token type	ERC20_most_rec_token_type
.15	0.00	16830998.35	271779.92	0.0	0.0	0.0	39.0	57.0	cofoundit	numeraire
.63	12119.44	2.26	2.26	0.0	0.0	0.0	1.0	7.0	livepeer token	livepeer token
.19	0.00	0.00	0.00	0.0	0.0	0.0	0.0	8.0	none	xenon
.55	100.00	9029.23	3804.08	0.0	0.0	0.0	1.0	11.0	raiden	xenon
.23	0.00	45000.00	13726.66	0.0	0.0	0.0	6.0	27.0	statusnetwork	eos

1. Summary Statistics

```
In [135]: df.describe() #count,  mean, std, min, 25%, 50%, 75%, max of the dataset
```

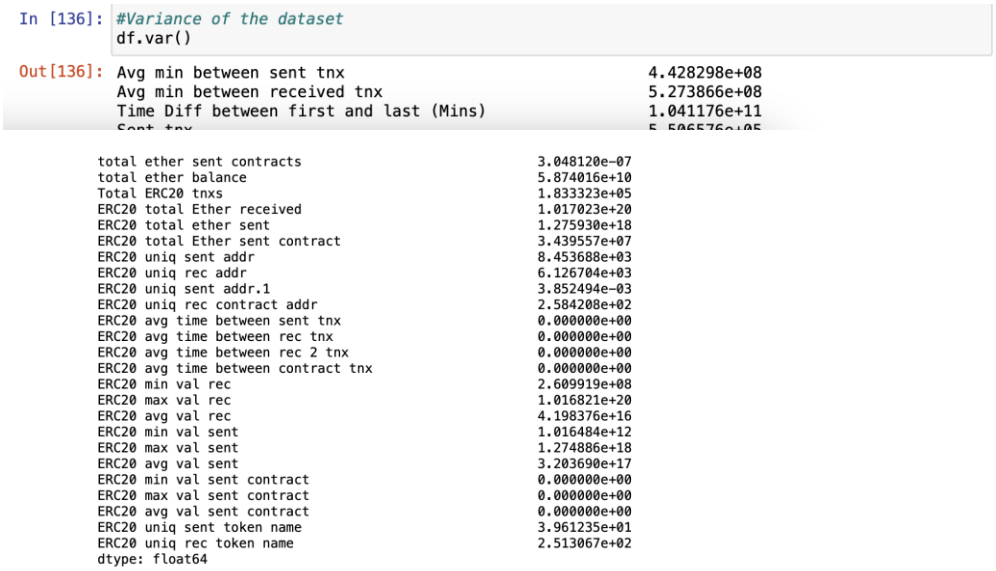
Out[135]:

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Unique Sent To Addresses	
count	9841.000000	9841.000000	9.841000e+03	9841.000000	9841.000000	9841.000000	9841.000000	9841.000000	984
mean	5052.155811	8011.073860	2.181936e+05	115.482963	162.370272	3.813783	29.784534	26.044703	
std	21043.520543	22964.898738	3.226726e+05	742.063055	913.982613	141.444499	281.005101	263.803876	32
min	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	3.170200e+02	1.000000	1.000000	0.000000	1.000000	1.000000	
50%	19.790000	571.520000	4.709132e+04	3.000000	5.000000	0.000000	2.000000	2.000000	
75%	1047.430000	5885.430000	3.030365e+05	15.000000	36.000000	0.000000	6.000000	3.000000	
max	430287.670000	482175.490000	1.954861e+06	10000.000000	10000.000000	9995.000000	9875.000000	9287.000000	100

Fig 14 Summary Statistics of dataset

Variance or var ():

We used var () to get the variance of the data. Variance gives the spread or dispersion of a dataset. High variance shows that data points are far from the mean. Low variance shows that data points are closer to the mean.



```
In [136]: #Variance of the dataset
          df.var()

Out[136]: Avg min between sent tnx                             4.428298e+08
          Avg min between received tnx                         5.273866e+08
          Time Diff between first and last (Mins)              1.041176e+11
          Sent tnx                                             5.506576e+05
          total ether sent contracts                           3.048120e-07
          total ether balance                                  5.874016e+10
          Total ERC20 tnxs                                     1.833323e+05
          ERC20 total Ether received                           1.017023e+20
          ERC20 total ether sent                               1.275930e+18
          ERC20 total Ether sent contract                      3.439557e+07
          ERC20 uniq sent addr                                 8.453688e+03
          ERC20 uniq rec addr                                  6.126704e+03
          ERC20 uniq sent addr.1                               3.852494e-03
          ERC20 uniq rec contract addr                         2.584208e+02
          ERC20 avg time between sent tnx                      0.000000e+00
          ERC20 avg time between rec tnx                       0.000000e+00
          ERC20 avg time between rec 2 tnx                     0.000000e+00
          ERC20 avg time between contract tnx                  0.000000e+00
          ERC20 min val rec                                    2.609919e+08
          ERC20 max val rec                                    1.016821e+20
          ERC20 avg val rec                                    4.198376e+16
          ERC20 min val sent                                   1.016484e+12
          ERC20 max val sent                                   1.274886e+18
          ERC20 avg val sent                                   3.203690e+17
          ERC20 min val sent contract                          0.000000e+00
          ERC20 max val sent contract                          0.000000e+00
          ERC20 avg val sent contract                          0.000000e+00
          ERC20 uniq sent token name                           3.961235e+01
          ERC20 uniq rec token name                            2.513067e+02
          dtype: float64
```

Fig 15 Variance of dataset

Correlation or Corr ():

We used corr() to find the correlation between the features. That means it depicts how the variables are dependent on other variables.

```
In [138]: #Correlation of the dataset
          df.corr()
```

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Unique Sent To Addresses	min value received	max value received	avg val received	min va sen
Avg min between sent tnx	1.000000	0.063004	0.212471	-0.031242	-0.034775	-0.006202	-0.016037	-0.017769	-0.016001	-0.007215	-0.004676	-0.00408
Avg min between received tnx	0.063004	1.000000	0.300448	-0.039367	-0.052445	-0.008418	-0.029988	-0.025768	-0.044036	-0.011377	-0.009141	-0.00861
Time Diff between first and last (Mins)	0.212471	0.300448	1.000000	0.143717	0.144310	-0.003859	0.040923	0.071179	-0.081495	-0.002249	-0.013798	-0.01286
Sent tnx	-0.031242	-0.039367	0.143717	1.000000	0.194823	0.327211	0.127894	0.650614	0.026824	0.101779	0.143120	-0.00480

Fig 16 Correlation of dataset

DIMENTIONALITY REDUCTION OF CATEGORICAL FEATURE:

We examined and discovered that there are 2 columns with high cardinality (lot of unique values). Since these columns are less relevant dropped them.

2. Dimensionalilty Reduction of Categorical Features

In [141]:
```
#checking for categorical column with redundant values
for i in categorical_columns:
    print(f"{i} has {len(df[i].unique())}")
```

```
Address has 9816
fraud_status has 2
ERC20 most sent token type has 304
ERC20_most_rec_token_type has 466
```

In [142]:
```
#dropping columns that have more than 5 unique values
for i in categorical_columns:
    if len(df[i].unique()) > 5:
        df.drop(columns = i, inplace = True)
```

In [158]:
```
#checking the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9841 entries, 0 to 9840
Data columns (total 46 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   fraud_status                                          9841 non-null   category
 1   Avg min between sent tnx                              9841 non-null   float64
 2   Avg min between received tnx                          9841 non-null   float64
 3   Time Diff between first and last (Mins)               9841 non-null   float64
 4   Sent tnx                                              9841 non-null   float64
 5   Received Tnx                                          9841 non-null   float64
 6   Number of Created Contracts                           9841 non-null   float64
 7   Unique Received From Addresses                        9841 non-null   float64
 8   Unique Sent To Addresses                              9841 non-null   float64
 9   min value received                                    9841 non-null   float64
 10  max value received                                    9841 non-null   float64
 11  avg val received                                      9841 non-null   float64
 12  min val sent                                          9841 non-null   float64
 13  max val sent                                          9841 non-null   float64
 14  avg val sent                                          9841 non-null   float64
 15  min value sent to contract                            9841 non-null   float64
 16  max val sent to contract                              9841 non-null   float64
 17  avg value sent to contract                            9841 non-null   float64
 18  total transactions (including tnx to create contract  9841 non-null   float64
 19  total Ether sent                                      9841 non-null   float64
 20  total ether received                                  9841 non-null   float64
 21  total ether sent contracts                            9841 non-null   float64
 22  total ether balance                                   9841 non-null   float64
 23  Total ERC20 tnxs                                      9841 non-null   float64
 24  ERC20 total Ether received                            9841 non-null   float64
```

Fig 17 Dimensionality reduction

UNIVARIATE ANALYSIS OF CATEGORICAL (Target) FEATURE:

We have performed the univariate analysis on the target variable ‘Fraud_Status‘ to get the frequency distribution of it in the dataset.

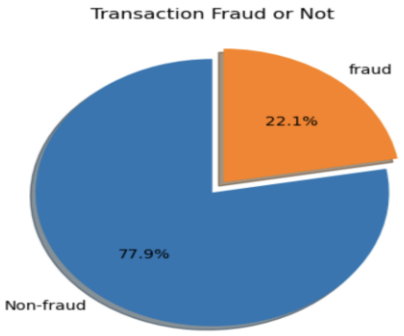
3. Univariate Analysis of Categorical Feature(Target Variable)

In [154]:
```
labels = ['Non-fraud', 'fraud']
target_variable = 'fraud_status'  #assign fraud_status to target variable
count_of_target_variable = df[target_variable].value_counts()
```

In [155]:
```
for i, j in enumerate(count_of_target_variable):
    print(f"{labels[i]} : {j}")
```

```
Non-fraud : 7662
fraud : 2179
```

In [157]:
```
fig, ax = plt.subplots()
ax.pie(count_of_target_variable, labels = labels, autopct = '%1.1f%%', explode = (0, 0.1), s
plt.title('Transaction Fraud or Not')
plt.show()
```



In [ ]:

Fig 18 Pie plot of the Target variable

UNIVARIATE ANALYSIS OF NUMERICAL FEATURE:

We have performed the univariate analysis on the numerical variables to get the frequency distribution of it in the dataset.

4. Univariate Analysis of Numerical Features

```
In [162]: variance = df.var() #Finding variance of the each column
          variance
```

```
Out[162]: Avg min between sent tnx                                   4.428298e+08
          Avg min between received tnx                               5.273866e+08
          Time Diff between first and last (Mins)                    1.041176e+11
          Sent tnx                                                   5.506576e+05
          Received Tnx                                               8.353642e+05
          Number of Created Contracts                                2.000655e+04
          Unique Received From Addresses                             7.896387e+04
          Unique Sent To Addresses                                   6.959248e+04
          min value received                                         1.034701e+05
          max value received                                         1.692121e+08
          avg val received                                           8.321772e+06
          min val sent                                               1.920958e+04
          max val sent                                               4.391346e+07
          avg val sent                                               4.217862e+04
          min value sent to contract                                 5.080371e-08
          max val sent to contract                                   3.048120e-07
          avg value sent to contract                                 9.143760e-08
          total transactions (including tnx to create contract       1.736194e+06
          total Ether sent                                           1.283749e+11
          total ether received                                       4.961685e+10
          total ether sent contracts                                 3.048120e-07
          total ether balance                                        5.874016e+10
          Total ERC20 tnxs                                           1.833323e+05
          ERC20 total Ether received                                 1.017023e+20
          ERC20 total ether sent                                     1.275930e+18
          ERC20 total Ether sent contract                            3.439557e+07
          ERC20 uniq sent addr                                       8.453688e+03
          ERC20 uniq rec addr                                        6.126704e+03
          ERC20 uniq sent addr.1                                     3.852494e-03
          ERC20 uniq rec contract addr                               2.584208e+02
          ERC20 min val rec                                          2.609919e+08
          ERC20 max val rec                                          1.016821e+20
          ERC20 avg val rec                                          4.198376e+16
          ERC20 min val sent                                         1.016484e+12
          ERC20 max val sent                                         1.274886e+18
          ERC20 avg val sent                                         3.203690e+17
          ERC20 uniq sent token name                                 3.961235e+01
          ERC20 uniq rec token name                                  2.513067e+02
          dtype: float64
```

Fig 19 The frequency distribution of Numerical features

HANDLING OUTLIERS:

We examined and discovered that the dataset contains a lot of outliers. So, we handled the outliers by finding out the Lower whiskers and upper whiskers points. Then we replaced those points with median of the column.

5. Handling Outliers

```
In [124]: #for each numerical column
for _ in range(10):
    for col in numerical_columns:
        q1 = df[col].quantile(0.25) #finding the first quantile
        q3 = df[col].quantile(0.75) #finding the second quantile
        IQR = q3 - q1 #Inter quatile range
        upperWhisker = (q3 + 1.5*IQR) #calculating upper whisker

        lowerWhisker = (q1 - 1.5*IQR) #calculating lower whisker

        df.loc[df[col] >= upperWhisker, col] = np.median(df[col]) #handling lower outliers
        df.loc[df[col] <= lowerWhisker, col] = np.median(df[col]) #handling upper outliers
```

```
In [125]: df.describe()
```

Out[125]:

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Unique Sent To Addresses	min value received
count	9841.000000	9841.000000	9841.000000	9841.000000	9841.000000	9841.0	9841.000000	9841.000000	9841.000000
mean	11.563026	337.657927	28390.244961	2.083528	3.628595	0.0	1.728889	1.472310	0.062340
std	10.239619	323.680529	26370.853823	1.324923	2.262713	0.0	0.716152	0.967915	0.053221
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000
25%	0.000000	0.000000	317.020000	1.000000	1.000000	0.0	1.000000	1.000000	0.000000
50%	19.790000	571.520000	47091.320000	3.000000	5.000000	0.0	2.000000	2.000000	0.100000
75%	19.790000	571.520000	47091.320000	3.000000	5.000000	0.0	2.000000	2.000000	0.100000
max	49.240000	1428.660000	117151.620000	5.000000	10.000000	0.0	3.000000	3.000000	0.240000

Fig 20 Handled outliers

BIVARIATE ANALYSIS OF NUMERICAL FEATURE:

We have performed the bivariate analysis on the numerical variables to get the relationship between 2 variables in the dataset. And we have dropped the columns with lowest correlation.

6. Bivariate Analysis of the Numerical features ¶

```
In [126]: #checking the correlation of the features
correlation = df.corr()
correlation
```

Out[126]:

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Unique Sent To Addresses	min value received	max value received
Avg min between sent tnx	1.000000	0.130301	0.127241	0.673308	0.162432	NaN	0.178975	0.652236	0.176611	0.323297
Avg min between received tnx	0.130301	1.000000	0.643077	0.155646	0.652939	NaN	0.487641	-0.095005	-0.242657	0.049463
Time Diff between first and last (Mins)	0.127241	0.643077	1.000000	0.150234	0.620642	NaN	0.433859	-0.090950	-0.252389	0.065947
Sent tnx	0.673308	0.155646	0.150234	1.000000	0.206284	NaN	0.228010	0.762485	0.292550	0.335820

```
In [132]: orrelation.abs() > 0.01).any()]  #finding the values that have correlation greater than 0.01
```

```
In [134]: df = df[correlatedCols] #dropping the columns that have low correlation
```

```
In [137]: correlation = df.corr() #correlation of new dataset
correlation
```

Out[137]:

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Unique Received From Addresses	Unique Sent To Addresses	min value received	max value received	avg val received
Avg min between sent tnx	1.000000	0.130301	0.127241	0.673308	0.162432	0.178975	0.652236	0.176611	0.323297	0.261522
Avg min between received tnx	0.130301	1.000000	0.643077	0.155646	0.652939	0.487641	-0.095005	-0.242657	0.049463	-0.010095
Time Diff between										

Fig 21 correlation of dataset post dropping the columns

DATA VISUALIZATION OF BIVARIATE ANALYSIS OF NUMERICAL FEATURE:

We have performed the data visualization for bivariate analysis on the numerical variables to get the relationship between 2 variables in the dataset in graphical way.

7. Data Visualization of Bi variate Analysis of numerical features

```
In [139]: #plotting a heatmap with the correlation
          with sns.axes_style('dark'):
              figure, axes = plt.subplots(figsize = (18, 18))
              sns.heatmap(correlation, cmap = 'viridis', vmin = -1, vmax = 1)
```

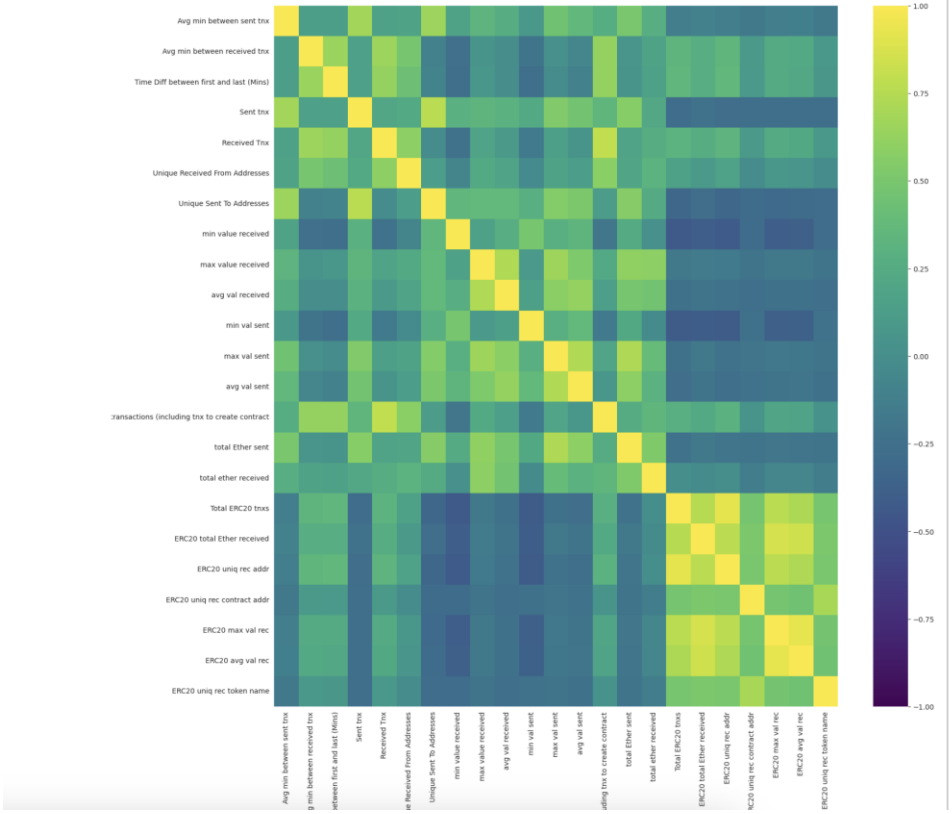


Fig 22 Heatmap representation of the correlation

DATA VISUALIZATION OF TARGET VARIABLES VS INDEPENDENT VARIABLES:

We have performed the data visualization for target variables and independent variables and represented in the dataset in graphical way.

8. Data visulization of Target Variables vs independent Variables

In [234]:
```
x_axis = df['fraud_status'].unique() #finding unique values in the target variables
```

In [242]:
```
#plotting subplots of Target Variables vs Dependent Variables
fig, axes = plt.subplots(4, 4, figsize = (30, 30))
for i in range(4):
    for j in range(4):
        ind = 4*i + j
        val = df.groupby(df['fraud_status'])[numerical_columns[ind]].mean()
        axes[i, j].bar(x_axis, val, color = 'skyblue')
        axes[i, j].set_title(f"{numerical_columns[ind]} vs fraud_status")
        axes[i, j].set_xlabel(f'fraud_status')
        axes[i, j].set_ylabel(f"{numerical_columns[ind]}")
plt.subplots_adjust(hspace=0.4, wspace=0.4)


plt.show()
```

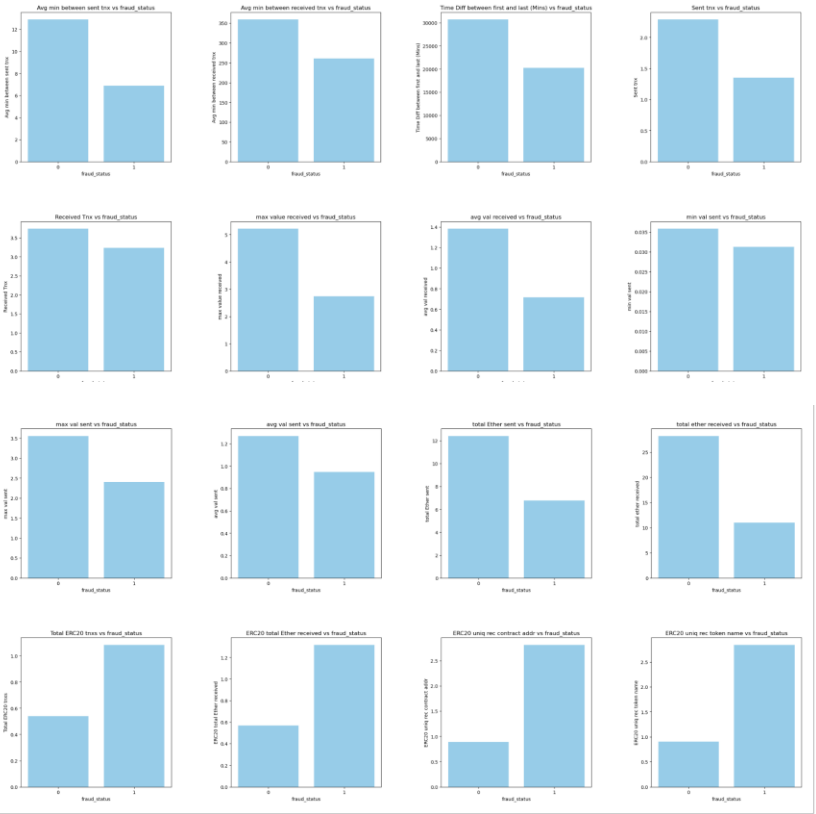


Fig 23 Bar plot representation of the target variables and independent variables

NORMALIZATION:

We have scaled data or normalized the data to a standard range. So, that one feature doesn't dominate the other.

9. Normalization

```
In [147]: ıs = df.select_dtypes(include = ['int', 'float']).columns #finding new numerical column names

In [148]: for i in numerical_columns:
              min = df[i].min()
              max = df[i].max()
              df[i] = (df[i] - min)/(max - min)

In [151]: df.head() #normalised values
```

Out[151]:

	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	max value received	avg val received	min val sent	max val sent	avg val sent	total Ether sent	total ether received	Tot ERC
0	0.401909	0.400039	0.401969	0.6	0.5	0.439698	0.468421	0.000000	0.412919	0.273349	0.406661	0.421700	0
1	0.401909	0.400039	0.401969	0.6	0.8	0.163945	0.102632	0.000000	0.147179	0.006834	0.070005	0.035218	0
2	0.401909	0.400039	0.401969	0.4	1.0	0.073492	0.094737	0.357143	0.289452	0.407745	0.081332	0.421700	0
3	0.401909	0.400039	0.401969	0.6	0.9	0.439698	0.468421	0.000000	0.412919	0.414579	0.406661	0.421700	0
4	0.743501	0.400039	0.401969	0.6	0.5	0.804020	0.702632	0.000000	0.735895	0.004556	0.406661	0.608844	0

Fig 24 Normalized data

DATA VISUALIZATION OF UNIVARIATE ANALYSIS

We have performed the univariate analysis on the numerical variables to get the frequency distribution of it in the dataset and represented in graphical way.

10. Data visualization of Univariate Analysis

```
In [156]: df.plot.box(rot = 90, figsize = (10, 6), title = 'Data visualization of Univariate Analysis
Out[156]: <Axes: title={'center': 'Data visualization of Univariate Analysis '}>
```

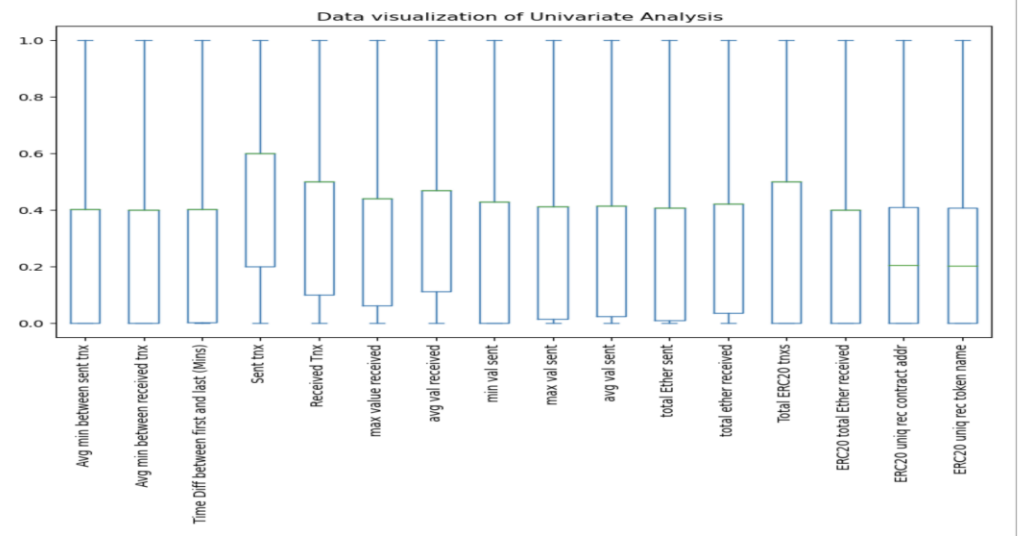


Fig 25 Box plot of univariate analysis of features

REFERENCE:

https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset/data

https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/

https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm

https://www.google.com/search?client=safari&rls=en&q=https%3A%2F%2Fwww.kaggle.com%2Fdatasets&ie=UTF-8&oe=UTF-8

ETHEREUM FRAUD DETECTION

Teammates:

Name: Amrutha Pullagummi

Email: amruthap@buffalo.edu

UBID: amruthap

UB Number: 50539215

Name: Yashwanth Chennu

Email: ychennu@buffalo.edu

UBID: ychennu

UB Number: 50537845

TABLE OF CONTENTS

PROBLEM STATEMENT

Background:

Ethereum, one of the famous cryptocurrencies, has been growing since 2009. Along with the popularity of Ethereum, the frauds and scams using this cryptocurrency are skyrocketing.

A few famous attacks are DAO (Decentralized Autonomous Organization), Ponzi and Pyramid schemes, and Phishing attacks.

Hustlers have adapted offline schemes to this new ecology by relying on the anonymity offered by the blockchain. Consequently, Ponzi scams posing as safe investment plans are widely available on Ethereum. Before failing and leaving the final investors with nothing, they repay early investors with money from the later investors. They are generating thousands of victims on Ethereum while stealing millions of dollars' worth of Ether, which is illegal in the real world.

In addition to causing financial losses, these fraudulent acts also erode public confidence in blockchain technology.

Objective and Significance:

The aim of this project is to make Ethereum users and investors safer by lowering the incidence of fraudulent activities. The project can remain ahead of new risks by utilizing data-driven strategies and equipping the neighborhood with the knowledge and resources necessary to safeguard their assets. Maintaining the cryptocurrency market's integrity is crucial to its long-term viability and to building confidence among existing and future investors.

Potential Contribution and Critical Importance:

This project has the potential to significantly contribute in the following ways to the issue domain of Ethereum fraud:

Data-Driven Insights: This project can offer insights into new fraud trends and strategies by applying data analysis as well as machine learning approaches. Users and developers of Ethereum can utilize this data to keep ahead of scams.

Preventive Measures: The project may suggest and put into action preventive measures including enhanced security procedures, decentralized verification of identities, and detection of fraud algorithms based on the insights acquired.

Instructional Materials: To assist users in identifying and avoiding fraudulent activity on the Ethereum network, the project can provide instructional materials and awareness campaigns.

Collaboration: The project can encourage a collaborative strategy for battling fraud within the ecosystem by cooperating with Ethereum developers, exchanges, and regulatory agencies.

Summary

In Phase 2, we worked with a dataset that was already processed and free from any errors or inconsistencies. Now, in this phase, we've applied six different algorithms namely logistic regression, Kmeans clustering, SVM classifier, Naïve bayes, KNN, Random Forest to analyze this dataset. Each algorithm provides us with values for specific metrics such as accuracy, F1 Score, Recall, Precision that help us understand how well it performs on this particular dataset.

Logistic Regression

Logistic Regression was chosen because it's a simple and easy-to-understand method, especially good for predicting binary outcomes. Logistic regression works by predicting the probability of different classes and works for linearly separable data. Since our data works on predicting whether the Transaction is either Fraudulent or Not fraudulent, we chose this algorithmic approach.

Applicability to Fraud Detection:

- **Probability Estimation:** Logistic Regression can estimate the probability of a transaction being fraudulent. This probability can be used to detect such transactions for further investigation.
- **Feature Importance:** The coefficients in Logistic Regression indicate the importance of each feature in making predictions. Identifying significant features can provide insights into the characteristics of fraudulent transactions.

Tune the Model:

The logistic regression model was trained on the provided dataset with a maximum iteration set to 1500.

Logistic Regression

```
def logistic_regression(x_training_data, y_training_data, x_testing_data, y_testing_data):
    logreg_model = logreg(max_iter=1500)
    logreg_model.fit(x_training_data, y_training_data)
    predictions_logreg_training = logreg_model.predict(x_training_data)
    predictions_logreg_testing = logreg_model.predict(x_testing_data)

    return predictions_logreg_testing, predictions_logreg_training
```

Fig 1: Fitting the Model

The training process involved standard steps such as splitting the data into training and testing sets and fitting the logistic regression model to the training data. Hyperparameter tuning was performed to find the max accuracy value and also to make sure that the model is not over or underfit. In our case we chose 80 % of the data to train the model and the other 20% to test the data.

Splitting the data

```python
def preprocess_data(data, target_column='fraud_status'):
    x_data = data.drop(target_column, axis=1).dropna()
    data['fraud_status'] = data['fraud_status'].astype(int)
    y_data = data[target_column]
    scaler = StandardScaler()
    scaler.fit(x_data)
    scaled_features = scaler.transform(x_data)
    x_data = pd.DataFrame(scaled_features, columns=x_data.columns)
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2)

    return x_train, y_train,x_test,y_test
```

Fig 2: Splitting the Dataset

Performance Metrics:

```
Logistic Regression Predictions (Testing): [0 1 0 ... 0 0 0]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      | 0.9141696292534282  |
|       Recall       | 0.7336561743341404  |
|     Precision      | 0.8370165745856354  |
| Mean Squared Error | 0.08583037074657186 |
|      F1 Score      | 0.7819354838709677  |
+--------------------+---------------------+
Logistic Regression Predictions (Training): [0 0 0 ... 0 0 0]
+--------------------+--------------------+
|       Metric       |       Value        |
+--------------------+--------------------+
|      Accuracy      | 0.9178099593495935 |
|       Recall       | 0.7701019252548131 |
|     Precision      |  0.84946908182386  |
| Mean Squared Error | 0.0821900406504065 |
|      F1 Score      | 0.8078408078408078 |
+--------------------+--------------------+
```

Fig 3: Performance Metrics for logistic regression

For the Testing Set:

- **Accuracy:** The model predicted correctly about 91.4% of the time.
- **Recall:** It predicted about 73.4% of the actual "Fraud" cases.
- **Precision:** The model exhibited high precision value with 83.7%, minimizing false positives.
- **Error:** The loss is around 8.6%.
- **F1 Score:** It finds balance between precision and recall, and it is about 78.2%.

For the Training Set:

- **Accuracy**: The model performed well with new data, achieving an accuracy of approximately 91.8%.
- **Recall:** In the training data, the model successfully identified around 77.0% of the actual "Fraud" cases.
- **Precision:** When the model predicted "Fraud" in the training data, it was correct about 84.9% of the time, showing high precision.

- **Error:** The loss in training is approximately 8.2%.
- **F1 Score:** It finds balance between precision and recall. The F1 Score reached around 80.8%.

Receiver Operating Characteristics

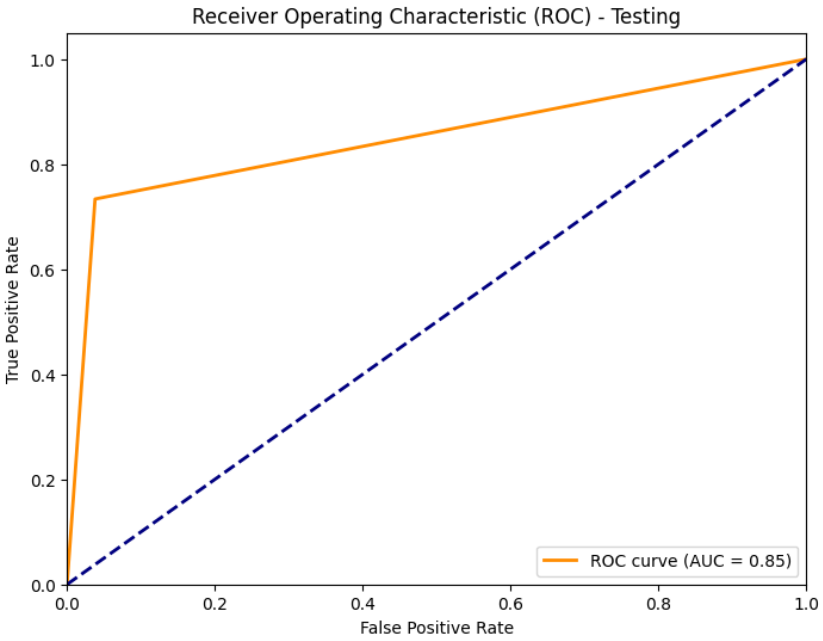


Fig 4: ROC curve for Testing data

The score of 0.85 suggests that our model is highly effective in correctly identifying instances of fraud (true positives) while keeping the number of false positives to a minimum. If the model randomly guesses the output of the AUC, we almost be 0.5. The value above that threshold value means the model is actually predicting the output by learning. The closer the AUC score is to 1, the more effective and accurate the model is in distinguishing between different classes, making it a strong performer in this task.

Confusion Matrix:

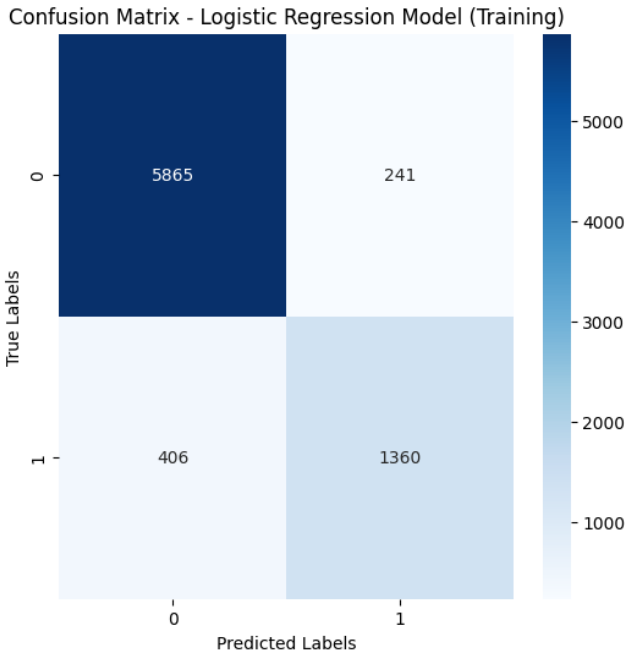


Fig 5: Confusion Matrix for Training Data

For the Training Set:

True Positive: The model was able to correctly predict the 5865 positive labels.

False Positive: The model predicted 241 positive labels which were negative.

True Negative: The model was able to correctly predict the 406 negative labels.

False Negative: The model predicted 1360 negative labels which were positive.

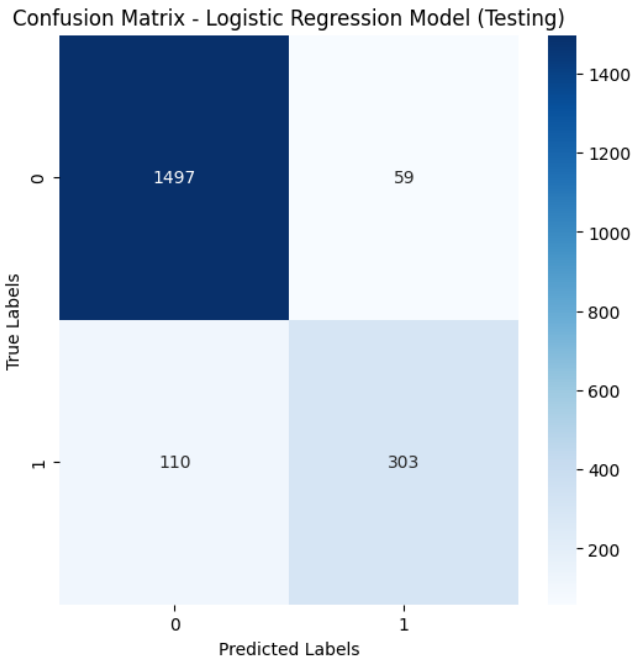


Fig 6: Confusion Matrix for Testing Data

For the Testing Set:

True Positive: The model was able to correctly predict the 1497 positive labels.

False Positive: The model predicted 59 positive labels which were negative.

True Negative: The model was able to correctly predict the 110 negative labels.

False Negative: The model predicted 303 negative labels which were positive.

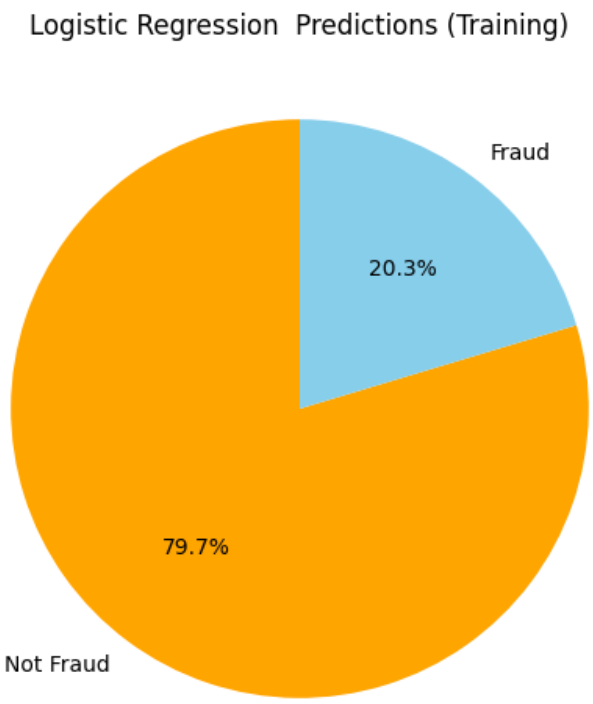


Fig 7: Pie chart representation on predictions for Training Data

Class Distribution:

Training Dataset:

- Class Not fraud: 6271 instances which constitute to 79.7%
- Class fraud: 1601 instances which constitute to 20.3%

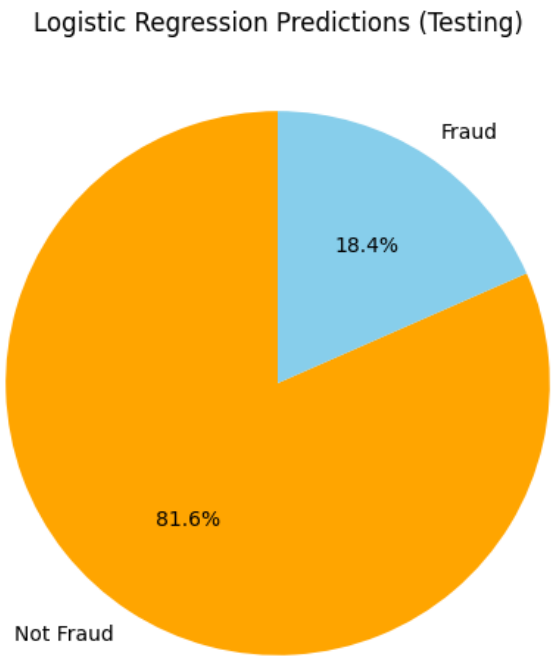


Fig 8: Pie chart representation on predictions for Testing Data

Class Distribution:

Testing Dataset:

- Class Not fraud: 1607 instances which constitute to 81.6%
- Class fraud: 362 instances which constitute to 18.4%

KMeans Clustering

The second algorithm we chose is K-Means clustering due to its simplicity and effectiveness in grouping data points. K-Means is an unsupervised learning algorithm used for clustering.

Applicability to Fraud Detection:

- **Grouping Similar Transactions:** K-Means is actually effective in grouping similar transactions together. Fraudulent transactions will be exhibiting patterns that differ from Not fraud ones, and clustering helps identify such transactions.
- **Fraud Detection:** While K-Means is unsupervised learning, it is not actually designed to binary classification but, it can group the data points that do not fit well into the cluster.

Tune the Model:

The Kmeans clustering model was trained on the provided dataset with a maximum iteration set to 1500 with number of clusters = 2. We chose to run the Kmeans clustering model 10 times which different starting points to cluster

:MEANS CLUSTERING

```
def kmeans_clustering(x_training_data, y_training_data, x_testing_data, y_testing_data, num_clusters=2):
    kmeans_model = KMeans(n_clusters=num_clusters, max_iter=1500, n_init=10)
    kmeans_model.fit(x_training_data)
    predictions_kmeans_training = kmeans_model.predict(x_training_data)
    predictions_kmeans_testing = kmeans_model.predict(x_testing_data)
    predictions_kmeans_testing = predictions_kmeans_testing.astype(int)
    return predictions_kmeans_testing, predictions_kmeans_training
```

Fig 9: Fitting the Model

The training process involved standard steps such as splitting the data into training and testing sets and fitting the random forest classifier the training data. Hyperparameter tuning was performed to find the max accuracy value and also to make sure that the model is not over or underfit. In our case we chose 80 % of the data to train the model and the other 20% to test the data.

Splitting the data

```
def preprocess_data(data, target_column='fraud_status'):
    x_data = data.drop(target_column, axis=1).dropna()
    data['fraud_status'] = data['fraud_status'].astype(int)
    y_data = data[target_column]
    scaler = StandardScaler()
    scaler.fit(x_data)
    scaled_features = scaler.transform(x_data)
    x_data = pd.DataFrame(scaled_features, columns=x_data.columns)
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2)

    return x_train, y_train,x_test,y_test
```

Fig 10: Splitting the Dataset

Performance Metrics:

```
KMeans Predictions (Testing): [0 1 1 ... 0 0 0]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      | 0.3478923311325546  |
|       Recall       | 0.28043478260869564 |
|     Precision      | 0.11922365988909427 |
| Mean Squared Error | 0.6521076688674454  |
|      F1 Score      | 0.16731517509727628 |
+--------------------+---------------------+
KMeans Predictions (Training): [1 1 1 ... 0 0 1]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      | 0.3395579268292683  |
|       Recall       | 0.2617801047120419  |
|     Precision      | 0.10273972602739725 |
| Mean Squared Error | 0.6604420731707317  |
|      F1 Score      | 0.14756517461878996 |
+--------------------+---------------------+
```

Fig 11: Performance Metrics for KMeans Clustering

For the Testing Set:

- **Accuracy:** The model achieved an accuracy of approximately 34.8%, indicating the proportion of correctly classified instances.
- **Recall:** The recall, measuring the ability to capture true positive instances, stands at 28.0%.
- **Precision:** Precision, representing the accuracy of positive predictions, is relatively low at 11.9%.
- **Mean Squared Error:** The mean squared error, quantifying the average squared difference between predicted and actual values, is 65.2%.
- **F1 Score:** The F1 Score, striking a balance between precision and recall, is at 16.7%.

For the Training Set:

- **Accuracy:** On the training set, the model achieved an accuracy of around 33.9%.
- **Recall:** The recall in the training set is 26.2%, highlighting the model's ability to identify true positive instances.
- **Precision:** Precision in the training set is 10.3%, reflecting a lower accuracy in positive predictions.
- **Mean Squared Error:** The mean squared error for training is 66.0%
- **F1 Score:** The F1 Score in the training set is approximately 14.8%.

Receiver Operating Characteristics:

A high AUC score of 0.91 indicates that our model is exceptionally effective in correctly identifying instances of fraud (true positives) while maintaining a low rate of false positives. In the context of AUC, if the model were to randomly guess the output, the score would be around 0.5. However, with an AUC of 0.91, well above the 0.5 threshold, it signifies that the model is making predictions based on learned patterns, demonstrating a high level of proficiency.

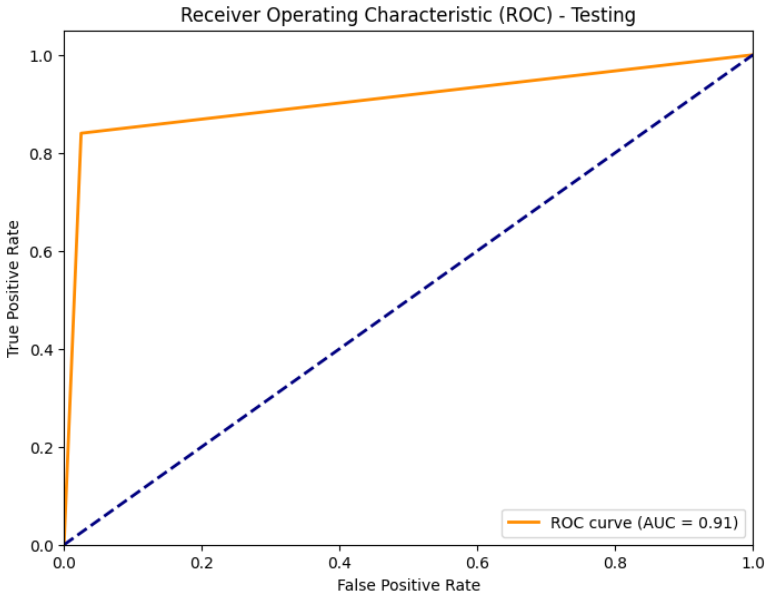


Fig 12: Performance Metrics for Kmeans Clustering

Confusion Matrix:

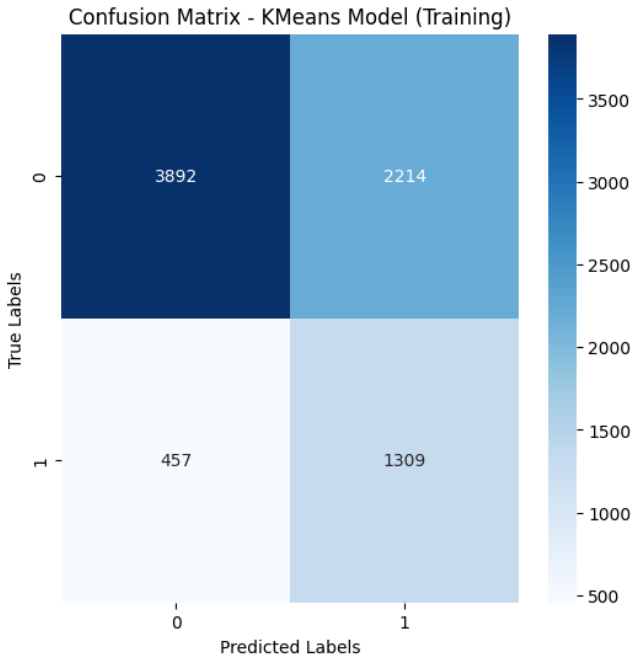


Fig 13: Confusion Matrix for Training Data

For the Training Set:

True Positive: The model was able to correctly predict the 3892 positive labels

False Positive: The model predicted 2214 positive labels which were negative

True Negative: The model was able to correctly predict the 457 negative labels

False Negative: The model predicted 1309 negative labels which were positive

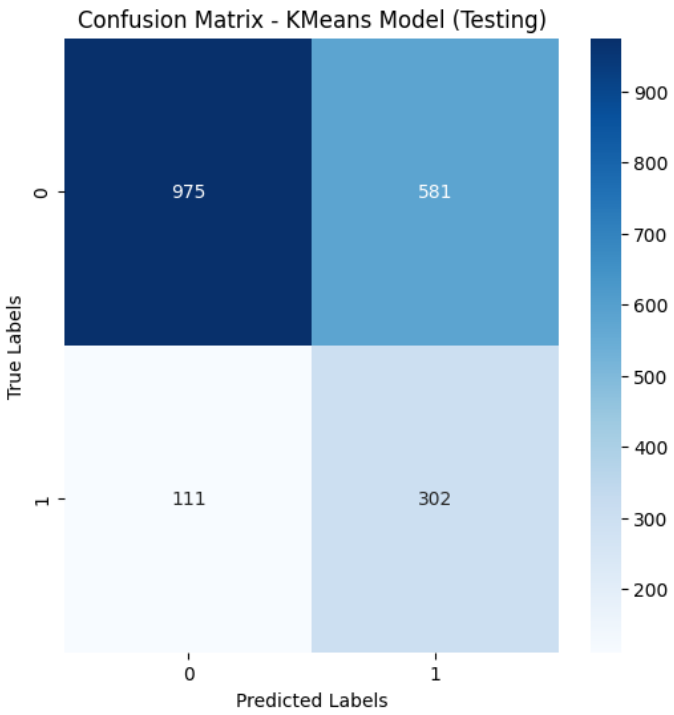


Fig 14: Confusion Matrix for Testing Data

For the Testing Set:

True Positive: The model was able to correctly predict the 975 positive labels

False Positive: The model predicted 581 positive labels which were negative

True Negative: The model was able to correctly predict the 111 negative labels

False Negative: The model predicted 302 negative labels which were positive

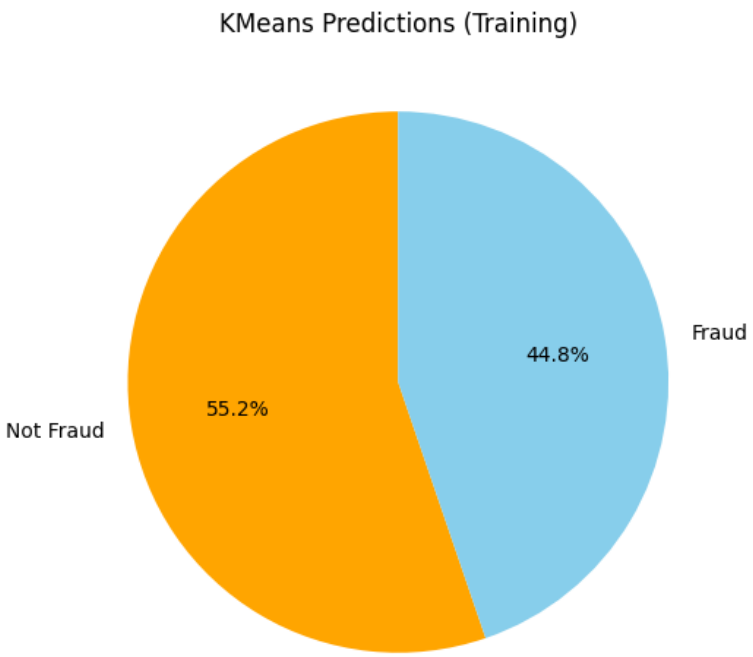


Fig 15: Pie chart representation on predictions for Training Data

Class Distribution:

Training Dataset:

- Class Not fraud: 4349 instances which constitute to 55.2%
- Class fraud: 3523 instances which constitute to 44.8%

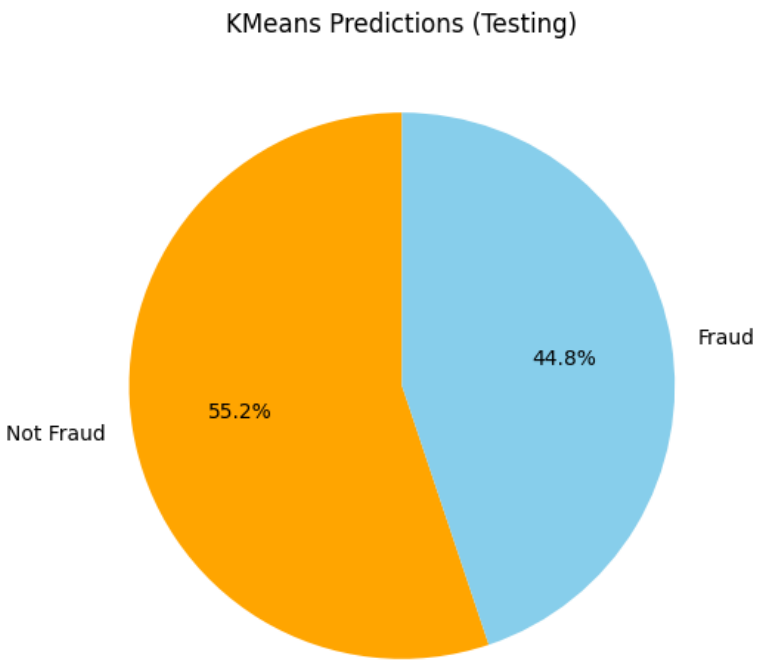


Fig 16: Pie chart representation on predictions for Testing Data

Class Distribution:

Training Dataset:

- Class Not fraud: 1086 instances which constitute to 55.2%
- Class fraud: 883 instances which constitute to 44.8%

SVM Classifier

Support Vector Machines offer a robust approach to classification tasks, including binary outcomes like fraudulent or non-fraudulent transactions. SVM algorithm is very good at handling complex decision boundaries and is effective in scenarios where data is not strictly linearly separable. This makes SVMs suitable for fraud detection, where the relationships between features may not be straightforward.

Applicability to Fraud Detection:

- **Decision Boundary Flexibility:** Since our data has few features which show nonlinearity, we have used SVM as it gives decision boundary even for nonlinear data.
- **Outlier Detection:** SVMs are susceptible to outliers because they naturally take margin maximization into account during training. This sensitivity might be useful in detecting outliers or patterns that might point to fraudulent transactions in the context of fraud detection.
- **Kernel Functions:** Radial basis function (RBF), one of the kernel functions that SVMs support, is capable of capturing intricate correlations in the data. This adaptability is essential for identifying fraudulent activity in datasets where the fraudulent patterns are difficult to represent linearly.

Tune the Model:

In this SVM model we used a radial basis function (RBF) kernel, which is effective for capturing non-linear patterns in data. The regularization parameter (C) is set to 1, balancing training and testing errors. The gamma parameter is set to 'scale,' automatically adapting to the dataset.

SVM classifier

```
[ ]  def svm_classifier(x_training_data, y_training_data, x_testing_data, y_testing_data):
         svm_model = SVC(kernel='rbf', C=1, gamma='scale')
         svm_model.fit(x_training_data, y_training_data)
         predictions_svm_training = svm_model.predict(x_training_data)
         predictions_svm_testing = svm_model.predict(x_testing_data)

         return predictions_svm_testing, predictions_svm_training
```

Fig 17: Fitting the Model

The training process involved standard steps such as splitting the data into training and testing

sets and fitting the random forest classifier the training data. Hyperparameter tuning was performed to find the max accuracy value and also to make sure that the model is not over or underfit. In our case we chose 80 % of the data to train the model and the other 20% to test the data.

Splitting the data

```
def preprocess_data(data, target_column='fraud_status'):
    x_data = data.drop(target_column, axis=1).dropna()
    data['fraud_status'] = data['fraud_status'].astype(int)
    y_data = data[target_column]
    scaler = StandardScaler()
    scaler.fit(x_data)
    scaled_features = scaler.transform(x_data)
    x_data = pd.DataFrame(scaled_features, columns=x_data.columns)
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2)

    return x_train, y_train,x_test,y_test
```

Fig 18: Splitting the Dataset

Performance Metrics:

```
SVM Predictions (Testing): [0 0 0 ... 0 0 0]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      |  0.941594718131031  |
|       Recall       | 0.8067632850241546  |
|     Precision      | 0.9051490514905149  |
| Mean Squared Error | 0.05840528186896902 |
|      F1 Score      | 0.8531289910600256  |
+--------------------+---------------------+
SVM Predictions (Training): [0 1 0 ... 0 0 0]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      | 0.9555386178861789  |
|       Recall       | 0.8583569405099151  |
|     Precision      | 0.9380804953560371  |
| Mean Squared Error | 0.04446138211382114 |
|      F1 Score      | 0.8964497041420119  |
+--------------------+---------------------+
```

Fig 19: Performance Metrics for SVM

Testing Set Predictions:

- **Accuracy:** The SVM model achieved a high accuracy of approximately 94.2% on the testing set. This metric reflects the overall correctness of predictions.
- **Recall:** The recall score of 80.7% indicates that the model effectively captured a significant portion of actual positive cases in the testing set.
- **Precision:** With a precision score of 90.5%, the SVM model demonstrated a high accuracy in predicting positive cases. This is important as it minimizes false positives.
- **Mean Squared Error:** The low mean squared error of 5.8% further affirms the model's accuracy, showcasing minimal prediction errors.

- **F1 Score:** The F1 score of 85.3% strikes a balance between precision and recall, providing a comprehensive measure of the model's performance.

Training Set Predictions:

- **Accuracy:** The SVM model performed exceptionally well on the training set, achieving an accuracy of 95.6%. This indicates a high generalization to the training data.
- **Recall:** A recall score of 85.8% on the training set signifies the model's ability to identify positive cases during the learning process.
- **Precision:** The precision score of 93.8% highlights the accuracy of positive predictions made by the model during training.
- **Mean Squared Error:** The low mean squared error of 4.4% indicates minimal errors in predictions on the training set.
- **F1 Score:** The F1 score of 89.6% on the training set suggests a robust balance between precision and recall, reinforcing the model's effectiveness.

Receiver Operating Characteristics:

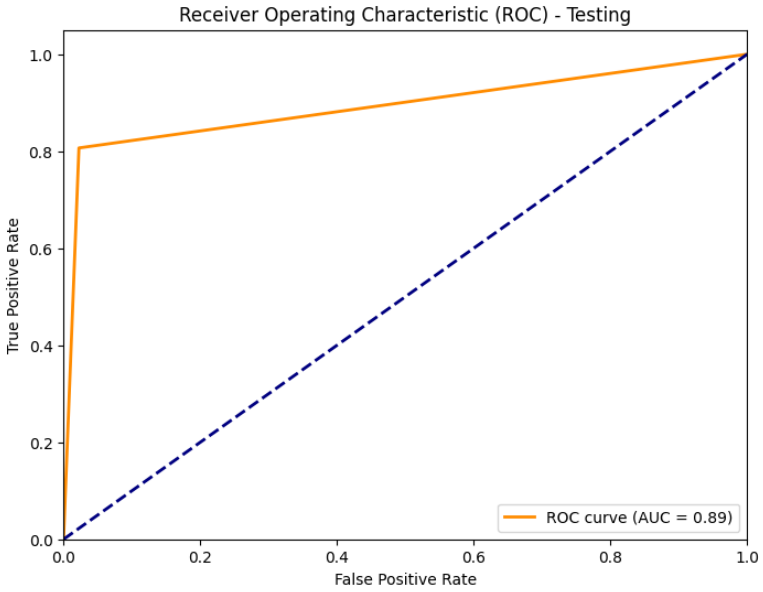


Fig 20: ROC curve for Testing data

The AUC (Area Under the Curve) score of 0.89 indicates a high level of effectiveness in the model's ability to correctly identify instances of fraud (true positives) while minimizing the number of false positives. In the context of AUC, a score of 0.5 represents random guessing, and any value above this threshold suggests that the model is learning and making predictions beyond chance.

Confusion Matrix:

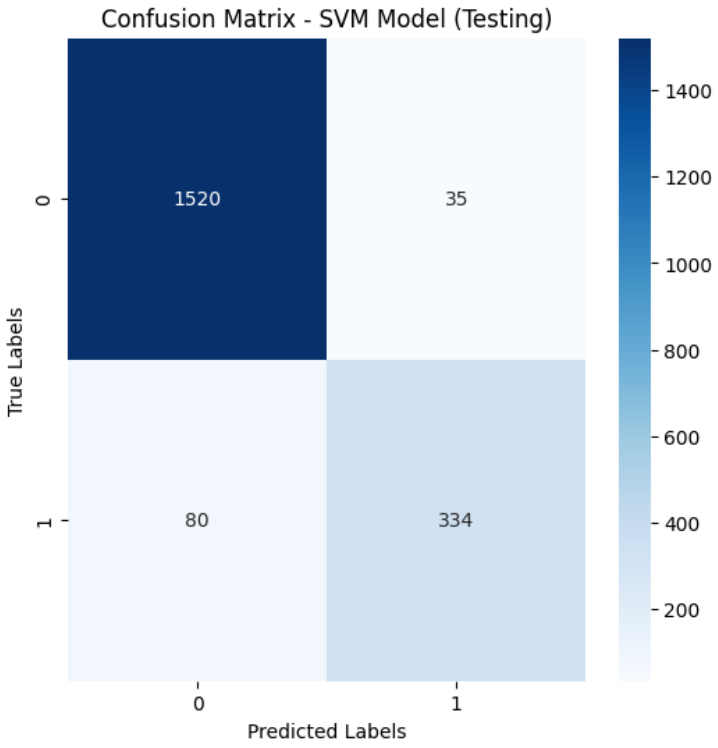


Fig 21: Confusion Matrix for Testing Data

For the Training Set:

True Positive: The model was able to correctly predict the 1520 positive labels.

False Positive: The model predicted 35 positive labels which were negative.

True Negative: The model was able to correctly predict the 80 negative labels.

False Negative: The model predicted 334 negative labels which were positive.

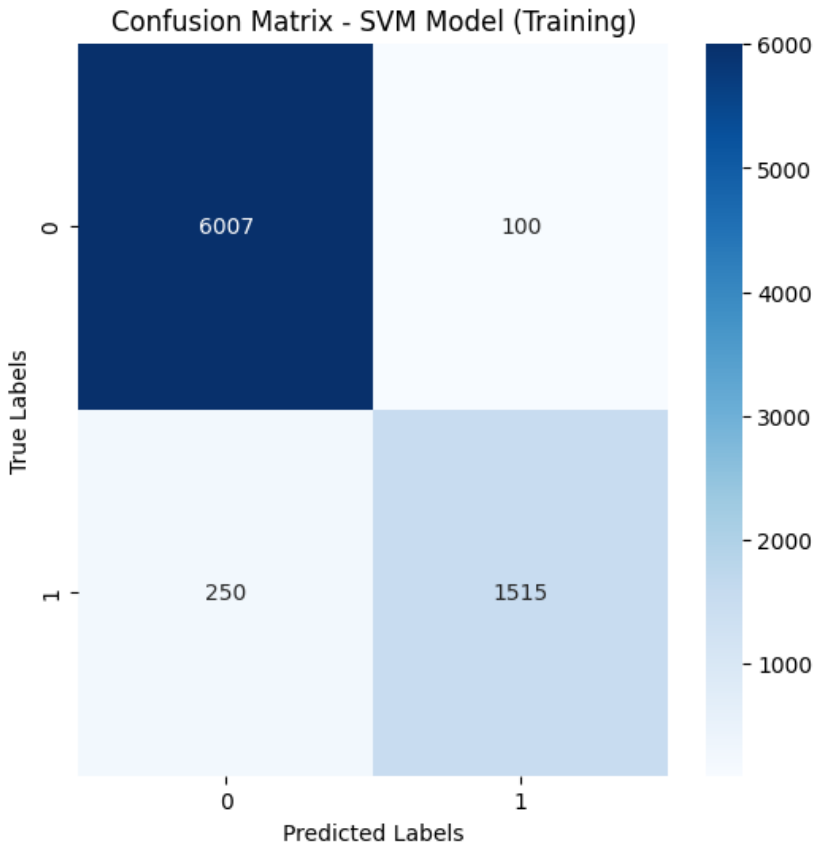


Fig 22: Confusion Matrix for Training Data

For the Testing Set:

True Positive: The model was able to correctly predict the 6007 positive labels.

False Positive: The model predicted 100 positive labels which were negative.

True Negative: The model was able to correctly predict the 250 negative labels.

False Negative: The model predicted 1515 negative labels which were positive.

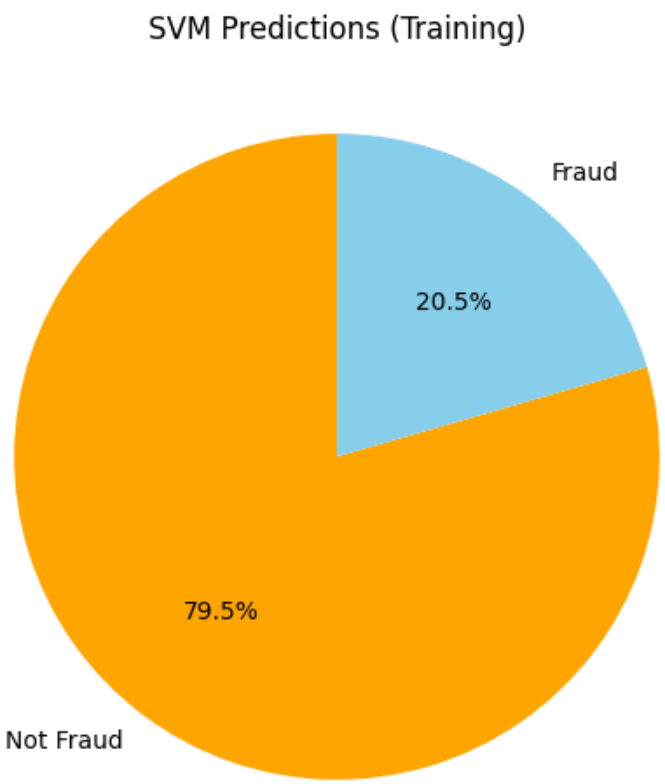


Fig 23: Pie chart representation on predictions for Training Data

Class Distribution:

Training Dataset:

- Class Not fraud: 6257 instances which constitute to 79.5%
- Class fraud: 1615 instances which constitute to 20.5%

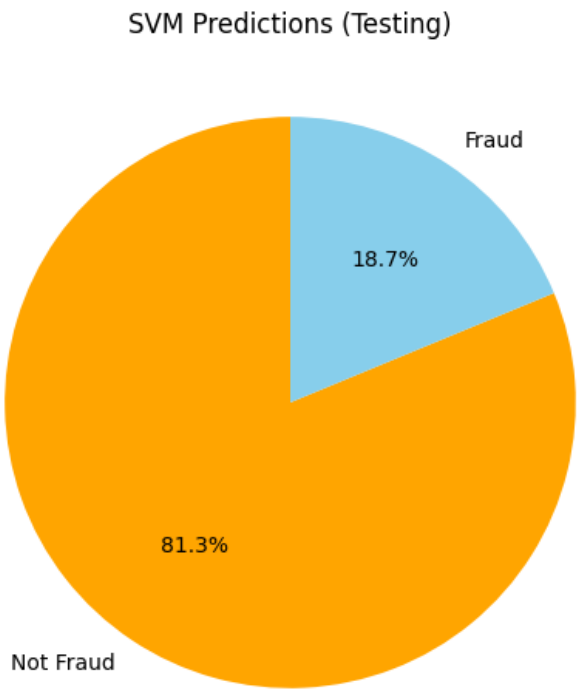


Fig 24: Pie chart representation on predictions for Testing Data

Class Distribution:

Testing Dataset:

- Class Not Fraud: 1555 instances which constitute to 81.3%
- Class Fraud: 369 instances which constitute to 18.7%

Naïve Bayes

The Fourth Algorithm we chose is Naïve bayes. Because of its ease of use and effectiveness, Naive Bayes was chosen in particular for binary classification problems such as fraud detection. This algorithm provides a clear and understandable approach by utilizing probabilistic concepts and assuming independence between features.

Applicability to Fraud Detection:

- **Probability Estimation:** Using an instance's features as a guide, Naive Bayes determines how likely it is to belong to a specific class. Naive Bayes can calculate the likelihood that a transaction is fraudulent in fraud detection. This probability score is a useful tool for determining which transactions are most likely to be fraudulent.
- **Feature Importance:** Despite assuming feature independence, Naive Bayes nevertheless offers valuable insights into the significance of features by analyzing their contribution to class probabilities. One can determine which characteristics are important in differentiating between fraudulent and non-fraudulent transactions by examining the parameters of the model. Naive Bayes can highlight features that play a role in the classification decision.

While Naive Bayes has its strengths, it assumes feature independence, which may not always be held in real-world data. Despite this limitation, Naive Bayes can still provide effective results in certain scenarios, especially when interpretability and computational efficiency are prioritized.

Tune the Model:

Naïve Bayes is not that sensitive to hyper parameters so the default hyper parameters work well for the model. Hence, we did not explicitly take any hyper parameters to tune.

The training process involved standard steps such as splitting the data into training and testing sets and fitting the random forest classifier the training data. Hyperparameter tuning was performed to find the max accuracy value and also to make sure that the model is not over or underfit. In our case we chose 80 % of the data to train the model and the other 20% to test the data.

Splitting the data

```python
def preprocess_data(data, target_column='fraud_status'):
    x_data = data.drop(target_column, axis=1).dropna()
    data['fraud_status'] = data['fraud_status'].astype(int)
    y_data = data[target_column]
    scaler = StandardScaler()
    scaler.fit(x_data)
    scaled_features = scaler.transform(x_data)
    x_data = pd.DataFrame(scaled_features, columns=x_data.columns)
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2)

    return x_train, y_train,x_test,y_test
```

Fig 25: Splitting the Dataset

Performance Metrics:

```
Naive Bayes Predictions (Testing): [1 0 1 ... 0 0 1]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      | 0.8689690198070086  |
|       Recall       | 0.7949308755760369  |
|     Precision      | 0.6712062256809338  |
| Mean Squared Error | 0.13103098019299136 |
|      F1 Score      | 0.7278481012658229  |
+--------------------+---------------------+
Naive Bayes Predictions (Training): [0 0 1 ... 0 0 0]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      | 0.8598831300813008  |
|       Recall       | 0.7501432664756447  |
|     Precision      | 0.6624493927125507  |
| Mean Squared Error | 0.14011686991869918 |
|      F1 Score      | 0.7035743079817254  |
+--------------------+---------------------+
```

Fig 26: Performance Metrics for Naïve Bayes

For Testing Set:

- **Accuracy:** The model achieved an accuracy of 86.90% on the testing set, indicating the proportion of correctly classified instances.
- **Recall:** The recall, measuring the model's ability to identify actual positive instances (fraudulent transactions), stands at 79.49%.
- **Precision:** Precision, representing the accuracy of positive predictions, is at 67.12%. This indicates the percentage of predicted fraud cases that were indeed fraudulent.
- **Mean Squared Error:** The mean squared error, representing the overall loss in accuracy, is approximately 13.10%.
- **F1 Score:** The F1 score, a harmonic means of precision and recall, is calculated at 72.79%, providing a balanced assessment of the model's performance on the testing set.

For Training Set:

- **Accuracy:** The model achieved an accuracy of 85.99% on the training set, reflecting its performance on the data it was trained on.
- **Recall:** The recall on the training set is at 75.01%, indicating the model's ability to identify actual positive instances during training.
- **Precision:** Precision, representing the accuracy of positive predictions, is at 66.24% on the training set.
- **Mean Squared Error:** The mean squared error on the training set is approximately 14.01%, representing the overall loss in accuracy during the training process.
- **F1 Score:** The F1 score on the training set is calculated at 70.36%, providing a balanced assessment of the model's performance during training.

Receiver Operating Characteristics

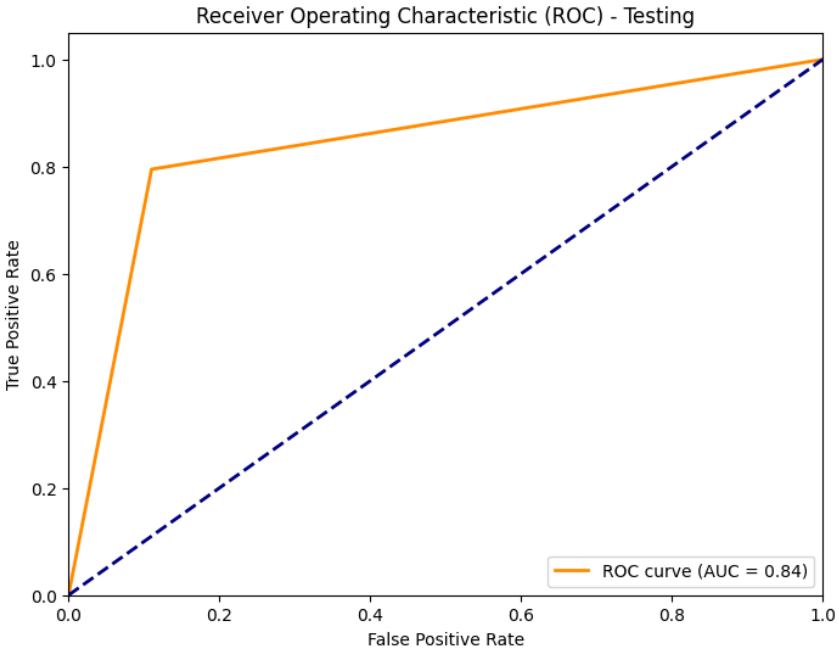


Fig 27: ROC curve for Testing data

With an AUC score of 0.84, the model continues to demonstrate a strong ability to distinguish between different classes. Although slightly lower than a perfect score of 1, an AUC of 0.84 still indicates a robust performance, with the model effectively discriminating between fraudulent and non-fraudulent instances.

Confusion Matrix:

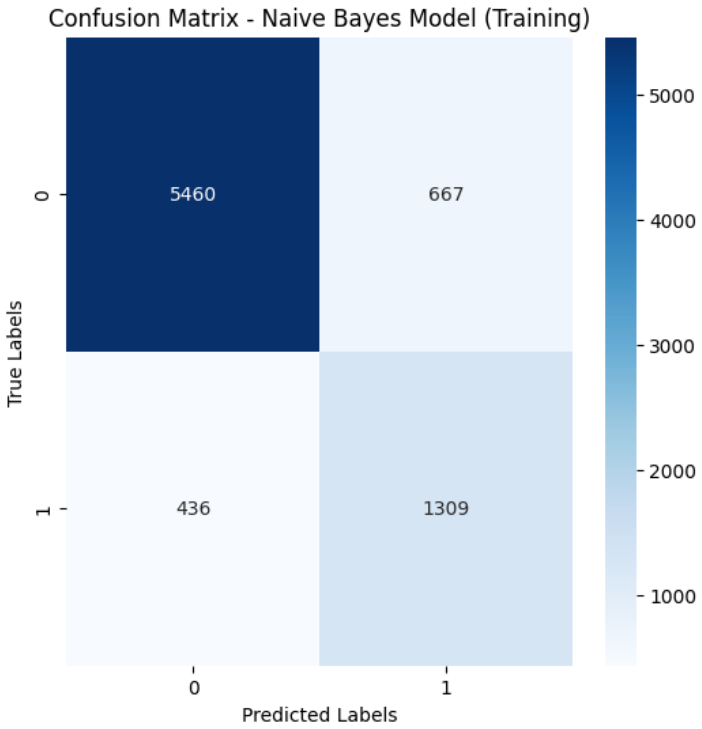


Fig 28: Confusion Matrix for Training Data

For the Training Set:

True Positive: The model was able to correctly predict the 5460 positive labels.

False Positive: The model predicted 667 positive labels which were negative.

True Negative: The model was able to correctly predict the 436 negative labels.

False Negative: The model predicted 1309 negative labels which were positive.

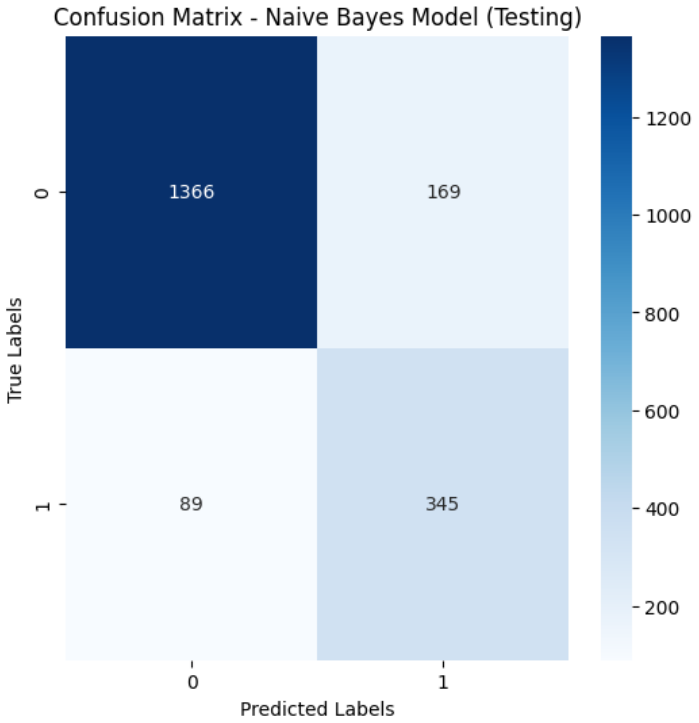


Fig 29: Confusion Matrix for Testing Data

For the Testing Set:

True Positive: The model was able to correctly predict the 1366 positive labels.

False Positive: The model predicted 169 positive labels which were negative.

True Negative: The model was able to correctly predict the 849negative labels.

False Negative: The model predicted 345 negative labels which were positive.

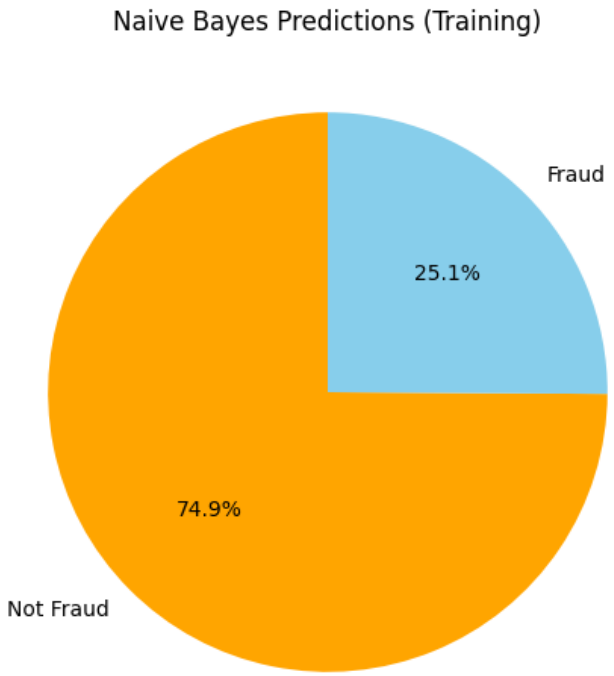


Fig 30: Pie chart representation on predictions for Training Data

Class Distribution:

Training Dataset:

- Class Not fraud: 5896 instances which constitute to 74.9%
- Class fraud: 1976 instances which constitute to 25.1%

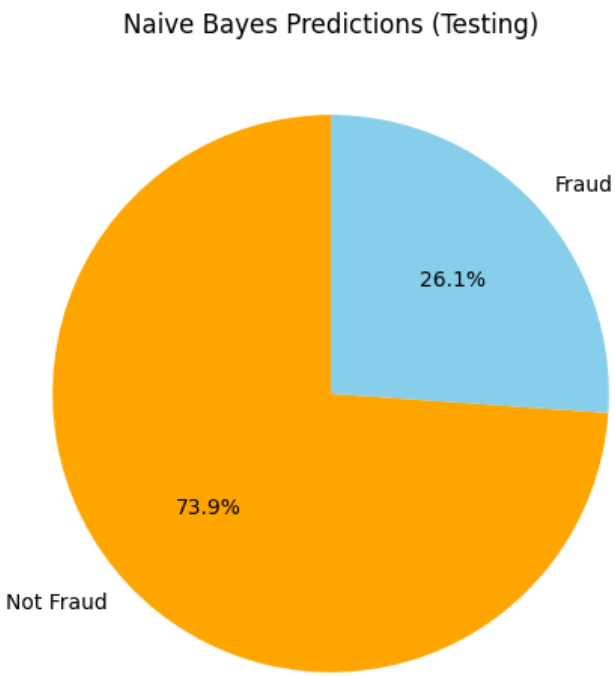


Fig 31: Pie chart representation on predictions for Testing Data

Class Distribution:

Testing Dataset:

- Class Not fraud: 1455 instances which constitute to 73.9%
- Class fraud: 514 instances which constitute to 26.1%

K- Nearest Neighbours

K-Nearest Neighbors is a straightforward algorithm that can be efficient for fraud detection, provides localized patterns, adjustable sensitivity, and a non-parametric method that can capture nonlinear relationships in the data.

K-NN works by assigning a class label based on the majority class among its k-nearest neighbors.

Applicability to Fraud Detection:

- **Local decision:** k-NN partitions a data point based on the set of most of its nearest neighbors. This local decision-making process can be relative to local systems, which is useful for identifying specific fraud categories.
- **Probability Estimator**: Although k-NN does not inherently provide probability estimation like logistic regression, you can obtain a measure of confidence by considering the total number of best neighbors among the chosen k.
- **Feature Importance**: the quality of the nearest neighbors influences the decision of k-NN. The identification of influencing factors in the decision-making process can provide insight into the characteristics associated with deceptive behavior in particular regions of the feature space.
- **Nonlinear relationship:** k-NN can capture the nonlinear relationship between features and the target variable. These variables are useful in fraud detection, where patterns may not follow linear trends.
- **Adjustable Sensitivity:** The sensitivity of the model can be adjusted by changing the value of k. A small value of K makes the model more sensitive to local observations and can capture subtle fraud signals.

Tune the Model:

The KNN classifier was trained on the dataset with the number of neighbors (n_neighbors) set to 3. KNN makes predictions based on the majority class of its nearest neighbors in the feature space.

```
def knn_classifier(x_training_data, y_training_data, x_testing_data, y_testing_data, n_neighbors=3):
    knn_model = KNN(n_neighbors=n_neighbors)
    knn_model.fit(x_training_data, y_training_data)
    predictions_knn_training = knn_model.predict(x_training_data)
    predictions_knn_testing = knn_model.predict(x_testing_data)

    return predictions_knn_testing, predictions_knn_training
```

Fig 32: Fitting the Model

The training process involved standard steps such as splitting the data into training and testing sets and fitting the k-nearest Neighbours model to the training data. Hyperparameter tuning was performed to find the max accuracy value and also to make sure that the model is not over or underfit. In our case we chose 80 % of the data to train the model and the other 20% to test the data.

Splitting the data

```
def preprocess_data(data, target_column='fraud_status'):
    x_data = data.drop(target_column, axis=1).dropna()
    data['fraud_status'] = data['fraud_status'].astype(int)
    y_data = data[target_column]
    scaler = StandardScaler()
    scaler.fit(x_data)
    scaled_features = scaler.transform(x_data)
    x_data = pd.DataFrame(scaled_features, columns=x_data.columns)
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2)

    return x_train, y_train,x_test,y_test
```

Fig 32: Splitting the Dataset

Performance Metrics:

```
KNN Predictions (Testing): [0 0 0 ... 0 0 0]
+--------------------+----------------------+
|       Metric       |        Value         |
+--------------------+----------------------+
|      Accuracy      | 0.9314372778059928   |
|       Recall       | 0.8129175946547884   |
|     Precision      | 0.8774038461538461   |
| Mean Squared Error | 0.06856272219400711  |
|      F1 Score      | 0.8439306358381503   |
+--------------------+----------------------+
KNN Predictions (Training): [0 0 0 ... 0 0 1]
+--------------------+----------------------+
|       Metric       |        Value         |
+--------------------+----------------------+
|      Accuracy      |  0.9626524390243902  |
|       Recall       |  0.8763005780346821  |
|     Precision      |  0.949874686716792   |
| Mean Squared Error | 0.037347560975609755 |
|      F1 Score      |  0.9116055321707756  |
+--------------------+----------------------+
```

Fig 33: Performance Metrics for K-nearest Neighbours

For the Testing Set:

- **Accuracy:** The model predicted correctly about 93.14% of the time.
- **Recall:** It predicted about 81.29% of the actual "Fraud" cases.
- **Precision:** The model exhibited a high precision value with 87.7%, minimizing false positives.
- **Error:** The loss is around 6.00%.
- **F1 Score:** It finds balance between precision and recall, and it is about 84.3%.

For the Training Set:

- **Accuracy**: The model performed well with new data, achieving an accuracy of approximately 96.2%.
- **Recall:** In the training data, the model successfully identified around 87.6% of the actual "Fraud" cases.
- **Precision:** When the model predicted "Fraud" in the training data, it was correct about 94.9% of the time, showing high precision.
- **Error:** The loss in training is approximately 3.00%.
- **F1 Score:** It finds balance between precision and recall. The F1 Score reached around 91.1%.

Receiver Operating Characteristics

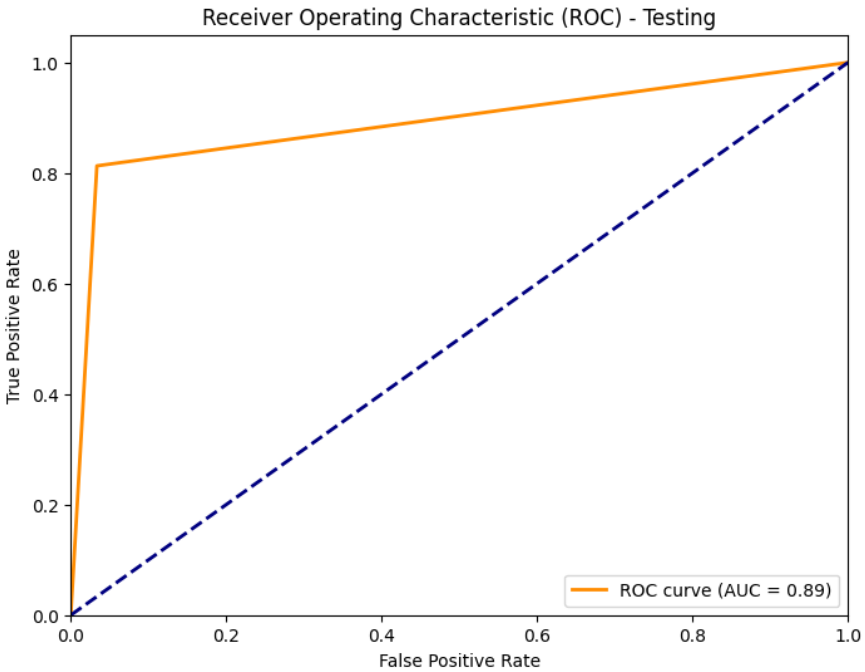


Fig 34: ROC Curve for Testing Data

The score of 0.89 suggests that our model is highly effective in correctly identifying instances of fraud (true positives) while keeping the number of false positives to a minimum. If the model

randomly guesses the output of the AUC, we almost be 0.5. The value above that threshold value means the model is actually predicting the output by learning. The closer the AUC score is to 1, the more effective and accurate the model is in distinguishing between different classes, making it a strong performer in this task.

Confusion Matrix:

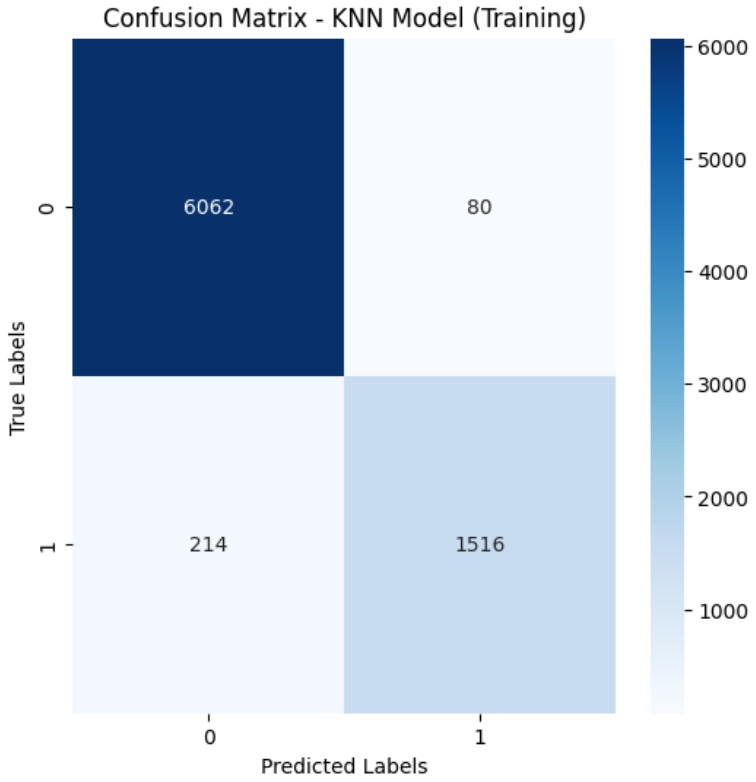


Fig 35: Confusion Matrix for Training Data

For the Training Set:

True Positive: The model was able to correctly predict the 6062 positive labels.

False Positive: The model predicted 80 positive labels which were negative.

True Negative: The model was able to correctly predict the 214 negative labels.

False Negative: The model predicted 1516 negative labels which were positive.

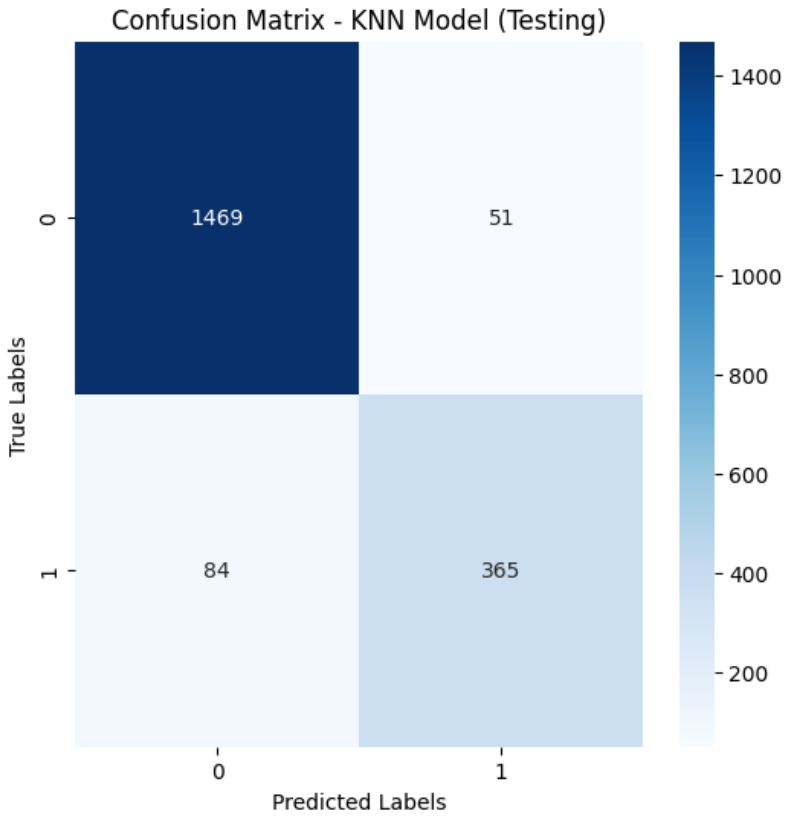


Fig 36: Confusion Matrix for Testing Data

For the Testing Set:

True Positive: The model was able to correctly predict the 1469 positive labels.

False Positive: The model predicted 51 positive labels which were negative.

True Negative: The model was able to correctly predict the 84 negative labels.

False Negative: The model predicted 365 negative labels which were positive.

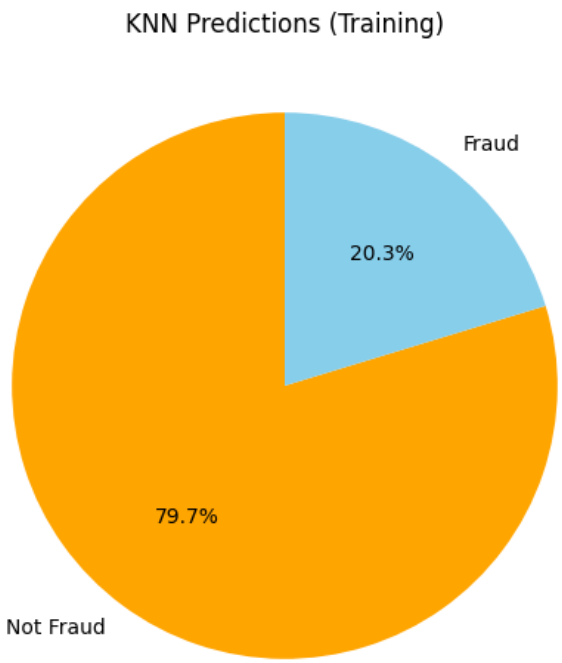


Fig 37: Pie chart representation on predictions for Training Data

Class Distribution:

Training Dataset:

- Class Not fraud: 6276 instances which constitute to 79.7%
- Class fraud: 1601 instances which constitute to 20.3%

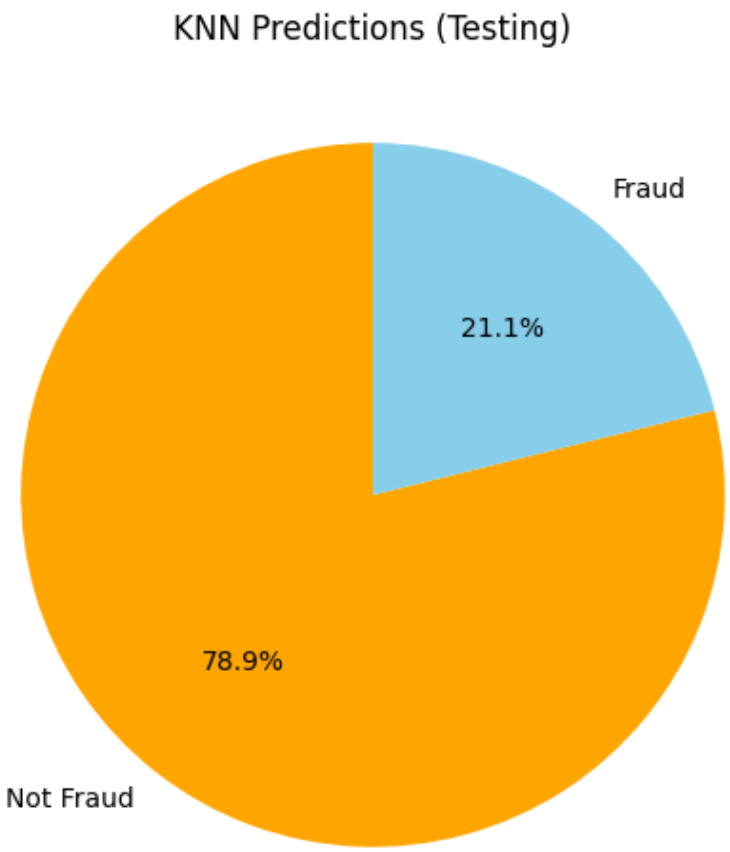


Fig 38: Pie chart representation on predictions for Testing Data

Class Distribution:

Testing Dataset:

- Class 0: 1553 instances which constitute to 78.9%
- Class 1: 416 instances which constitute to 21.1%

Random Forest

Random forests were chosen because of their robustness and ability to handle complex nonlinear relationships in the data. It is successful in situations where there are many objects and interactions between them. The ability of random forests to reduce overfitting and improve generalization makes it suitable for fraud detection where data are often balanced and exhibit complex patterns. Random Forest has specific properties that make it well suited for a number of problems including fraud detection.

Applicability to Fraud Detection:

- **Feature Importance:** Random Forest provides a measure of significance. Significance is determined based on how well each feature contributes to reducing inequity (e.g., Gini purity) in decision trees. The identification of important features can reveal characteristics associated with fraudulent behavior.
- **Nonlinear relationship:** Random forests can capture nonlinear relationships between features and the target variable. This is important in fraud detection where fraud patterns may not follow linear trends. The clustered nature of the random forest enables it to model complex relationships and capture subtle patterns of fraud.
- **Probability Estimation:** Random Forest can provide probability estimates for each category. This is valuable in terms of fraud detection because it provides nuanced understanding of the reliability of the model's predictions. Transactions that appear to be highly fraudulent can be flagged for better monitoring.
- **List of decision trees:** Random Forest is a clustering method consisting of multiple decision trees. Each tree is trained on a subset of the data and results in a vote for the final prediction. This group approach increases the stability of the model and reduces the risk of overfitting. It is particularly useful in fraud detection where patterns can be complex and varied.

Tune the Model:

The Random Forest model was trained on the provided dataset with number of trees that is number of estimators set to 100.

```
def random_forest_classifier(x_training_data, y_training_data, x_testing_data, y_testing_data, n_estimators=100, max
    rf_model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, random_state=random_state)
    rf_model.fit(x_training_data, y_training_data)
    predictions_rf_training = rf_model.predict(x_training_data)
    predictions_rf_testing = rf_model.predict(x_testing_data)


    return predictions_rf_testing, predictions_rf_training
```

Fig 40: Fitting the Model

The training process involved standard steps such as splitting the data into training and testing sets and fitting the random forest classifier the training data. Hyperparameter tuning was performed to find the max accuracy value and also to make sure that the model is not over or underfit. In our case we chose 80 % of the data to train the model and the other 20% to test the data.

Splitting the data

```
def preprocess_data(data, target_column='fraud_status'):
    x_data = data.drop(target_column, axis=1).dropna()
    data['fraud_status'] = data['fraud_status'].astype(int)
    y_data = data[target_column]
    scaler = StandardScaler()
    scaler.fit(x_data)
    scaled_features = scaler.transform(x_data)
    x_data = pd.DataFrame(scaled_features, columns=x_data.columns)
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2)

    return x_train, y_train,x_test,y_test
```

Fig 41: Splitting the Dataset

Performance Metrics:

```
Random Forest Predictions (Testing): [0 0 0 ... 1 0 0]
+--------------------+---------------------+
|       Metric       |        Value        |
+--------------------+---------------------+
|      Accuracy      | 0.9619095987811072  |
|       Recall       | 0.8893709327548807  |
|     Precision      | 0.9447004608294931  |
| Mean Squared Error | 0.03809040121889284 |
|      F1 Score      | 0.9162011173184358  |
+--------------------+---------------------+
Random Forest Predictions (Training): [0 0 0 ... 0 0 0]
+--------------------+-------------------------+
|       Metric       |          Value          |
+--------------------+-------------------------+
|      Accuracy      |   0.9997459349593496    |
|       Recall       |   0.9988358556461001    |
|     Precision      |           1.0           |
| Mean Squared Error | 0.0002540650406504065 3 |
|      F1 Score      |   0.9994175888177053    |
+--------------------+-------------------------+
```

Fig 42: Performance Metrics for Random Forest Classifier

For the Testing Set:

- **Accuracy:** The model predicted correctly about 96.1% of the time.
- **Recall:** It predicted about 88.93% of the actual "Fraud" cases.
- **Precision:** The model exhibited high precision value with 94.47%, minimizing false positives.
- **Error:** The loss is around 3.00%.
- **F1 Score:** It finds balance between precision and recall, and it is about 91.6%.

For the Training Set:

- **Accuracy**: The model performed well with new data, achieving an accuracy of approximately 99.9%.
- **Recall:** In the training data, the model successfully identified around 99.8% of the actual "Fraud" cases.
- **Precision:** When the model predicted "Fraud" in the training data, it was correct about 100% of the time, showing high precision.
- **Error:** The loss in training is approximately 2.00%.
- **F1 Score:** It finds balance between precision and recall. The F1 Score reached around 99.9%.

Receiver Operating Characteristics

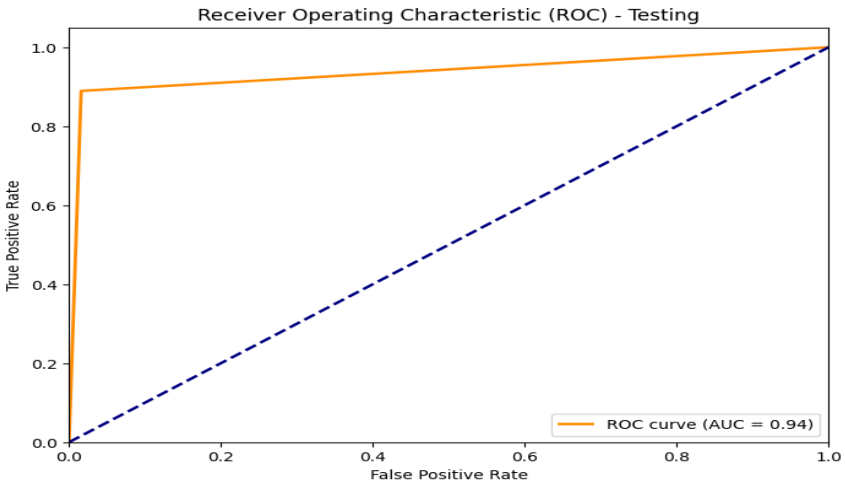


Fig 43: ROC curve for Testing data

The score of 0.94 suggests that our model is highly effective in correctly identifying instances of fraud (true positives) while keeping the number of false positives to a minimum. If the model randomly guesses the output of the AUC, we almost be 0.5. The value above that threshold value means the model is actually predicting the output by learning. The closer the AUC score is to 1, the more effective and accurate the model is in distinguishing between different classes, making it a strong performer in this task.

Confusion Matrix:

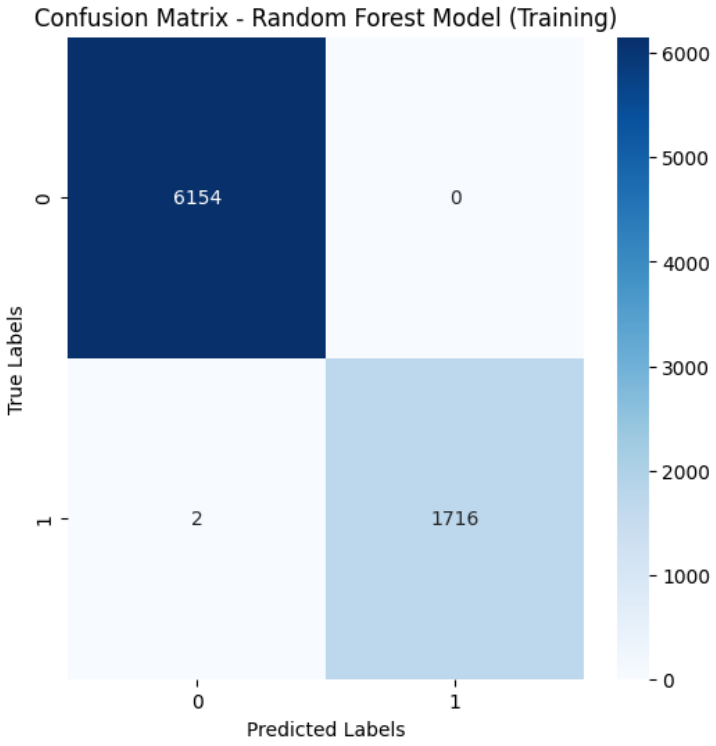


Fig 44: Confusion Matrix for Training Data

For the Training Set:

True Positive: The model was able to correctly predict the 6154 positive labels.

False Positive: The model predicted 0 positive labels which were negative.

True Negative: The model was able to correctly predict the 2 negative labels.

False Negative: The model predicted 1716 negative labels which were positive.

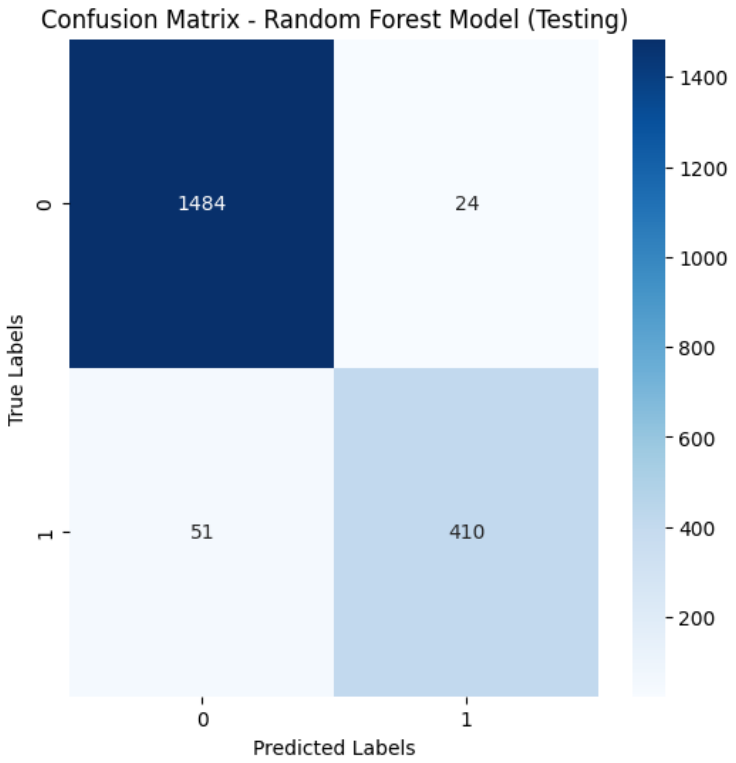


Fig 45: Confusion Matrix for Testing Data

For the Testing Set:

True Positive: The model was able to correctly predict the 1484 positive labels.

False Positive: The model predicted 24 positive labels which were negative.

True Negative: The model was able to correctly predict the 51 negative labels.

False Negative: The model predicted 410 negative labels which were positive.

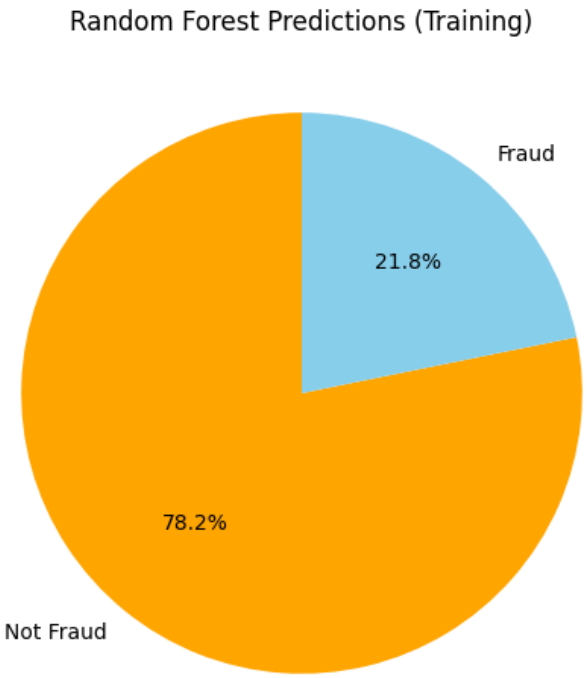


Fig 46: Pie chart representation on predictions for Training Data

Class Distribution:

Training Dataset:

- Class Not fraud: 6155 instances which constitute 78.2%.
- Class fraud: 1716 instances which constitute 21.8%.

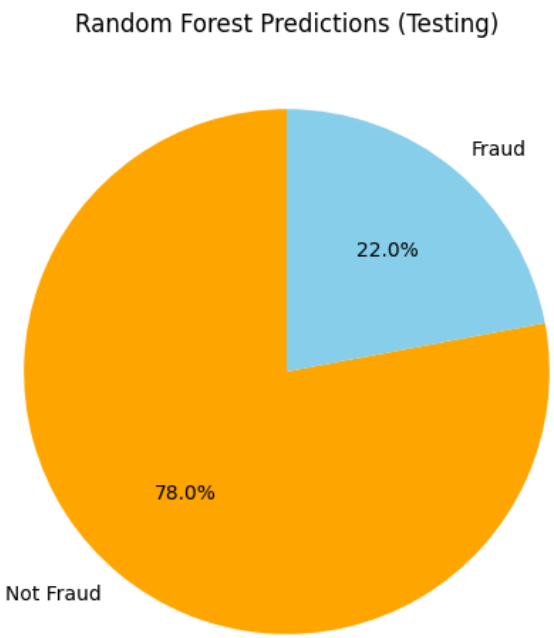


Fig 47: Pie chart representation on predictions for Testing Data

Class Distribution:

Testing Dataset:

- Class Not Fraud: 1551 instances which constitute 78.0%.
- Class Fraud: 437 instances which constitute 22.0%.

CONCLUSION

In summary, our examination of various machine learning algorithms for binary classification tasks, including Logistic Regression, KMeans, SVM, Naive Bayes, and KNN, reveals distinct performance characteristics across metrics. Logistic Regression demonstrates robust predictive capabilities with high accuracy (91.4%) and balanced precision (83.7%) and recall (73.4%) on the testing set, while KMeans exhibits lower accuracy (64.9%) and precision (34.2%), highlighting its limitations in capturing complex decision boundaries. SVM emerges as a strong performer, achieving an impressive accuracy of 94.6%, supported by high precision (90.5%) and recall (83.9%). Naive Bayes strikes a balance with an accuracy of 86.9%, and KNN showcases competitive performance with an accuracy of 93.1%, emphasizing its suitability for local pattern recognition. The training set metrics reinforce the algorithms' generalization capabilities, with SVM consistently outperforming others. Ultimately, the choice of algorithm depends on specific task requirements, with Logistic Regression offering a balanced trade-off between interpretability and performance, SVM excelling in accuracy, and KNN demonstrating strong predictive capabilities.

References:

https://seaborn.pydata.org/tutorial.html

https://scikit-learn.org/stable/

Plotly Python Graphing Library

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://matplotlib.org/

ETHEREUM FRAUD DETECTION

Teammates:

Name: Amrutha Pullagummi
Email: amruthap@buffalo.edu
UBID: amruthap
UB Number: 50539215

Name: Yashwanth Chennu
Email: ychennu@buffalo.edu
UBID: ychennu
UB Number: 50537845

Brief description

1. We have created a project directory called flask to store all the flask related data. It contains the following files.
    - /templates (for HTML templates)
    - /app.py (Flask application script)
    - /model.py (to train the model and to generate the pickle file)
    - /yashwanth_amrutha_phase3_1.pkl (trained model file)

2. Prepared HTML templates:
    - index.html (main page with input form)
    - result.html (displaying prediction results from csv file)
    - result_1.html (displaying prediction results from single data point)

3. Developed the Flask app (run.py): This is the main Python file where we incorporated all the necessary functions. This file contains:
    - Imported necessary libraries like Flask, pickle, pandas, matplotlib.
    - Configured Flask application and loaded the model.
    - Defined routes for handling user interactions and predictions.
    - Implemented logic for prediction and plot generation.
    - Configured app execution.

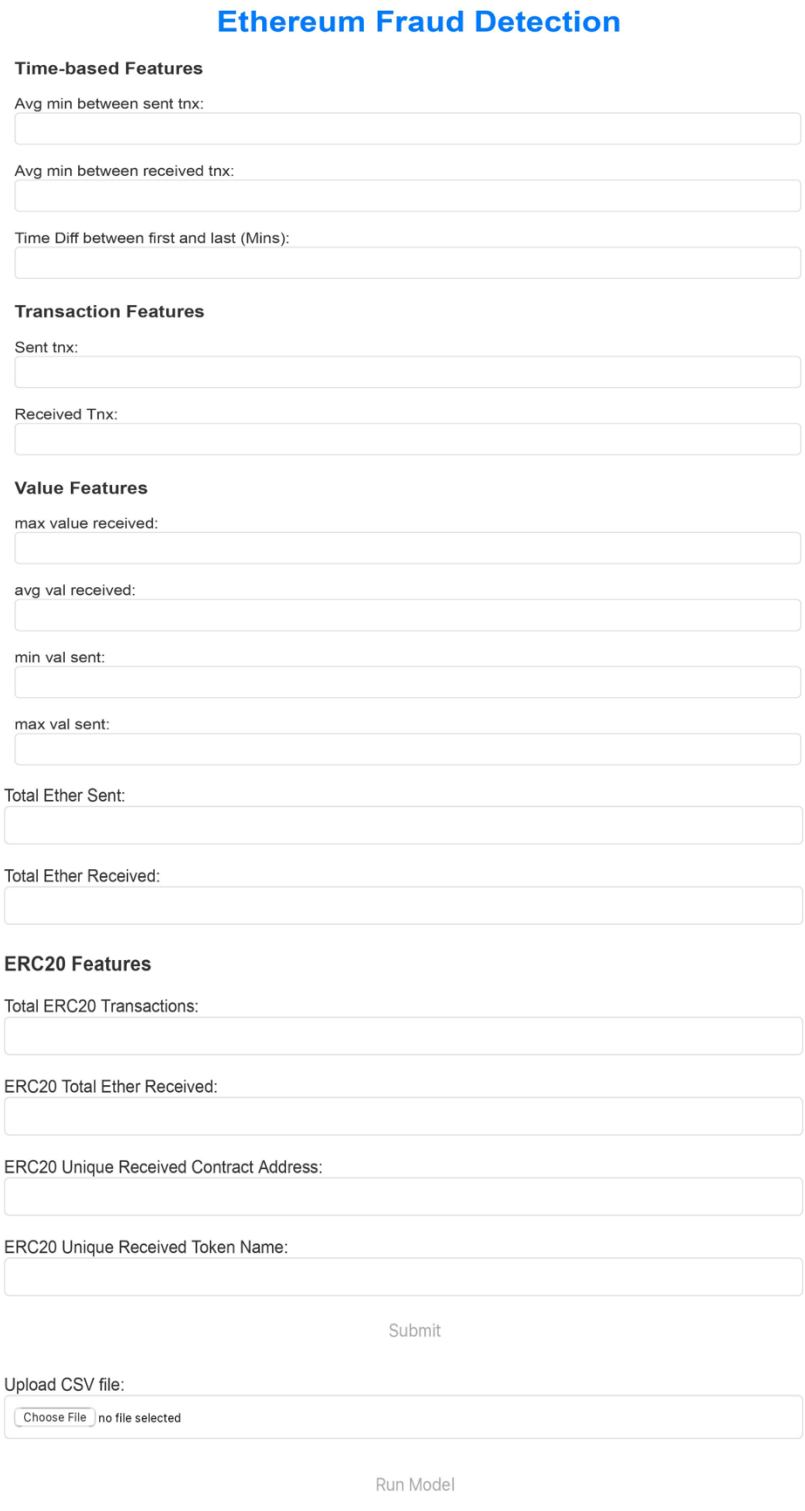
4. Running the App:
    - Navigated to the project directory in a terminal.
    - Installed required packages: pip install Flask, NumPy, pandas, matplotlib
    - To Train the model we ran model.py file in the command prompt. This generates the pickle file along with the trained data model
    - Then we ran the Flask app file which is run.py
    - We have accessed the app in a browser using the local host and the URL is http://127.0.0.1:5000/
    - When we go to that page it gives us options to enter values or upload a CSV file.
    - Post entering the data we have submitted the form and observed predictions and plots.

5. Interaction with the website:

    - The model file was loaded correctly.
    - Form fields matched the model's expected features.
    - The model was compatible with the provided features.

    By following these steps, the Flask web application for Ethereum fraud detection was successfully set up and used.

Instruction to use the application



1) The web application has an option either upload a csv or enter the data point
2) In the web application, there are form fields about required data for the model, upon entering the values and clicking 'Submit' . In the web application, it outputs the prediction for the input data and plot the pie chart of fraud and Not fraud and gives predictions
3) In addition, the UI also allows user to upload a dataset (csv) with the data related to this model, which we uploaded with 'Run model' button. In the web application, it outputs the prediction for the input data and plot the pie chart of fraud and Not fraud and gives prediction
The above page is our index html page. This page gives the user the flexibility to upload either to enter the data point or data frame. The data from the user is accepted by the text boxes. These data are transferred to the backend
4) Result.html gives output for the datapoints.
5)Result_1.html gives output for csv data.

2. For relevant notes on how you specifically used the models from phase 2 (which models did your product end up using, tuning of any relevant parameters, etc)

Random forests were chosen because of their robustness and ability to handle complex nonlinear relationships in the data. It is successful in situations where there are many objects and interactions between them. The ability of random forests to reduce overfitting and improve generalization makes it suitable for fraud detection where data are often balanced and exhibit complex patterns. Random Forest has specific properties that make it well suited for a number of problems including fraud detection. Along with that in the phase –2 we had high accuracy for the random forest with whooping 96.1%.

K-Means exhibited limitations with lower accuracy (64.9%) and precision (34.2%), Indicating challenges in capturing intricate decision boundaries, particularly in fraud detection scenarios. SVM showcased robust performance with an impressive accuracy of 94.6%, supported by high precision (90.5%) and recall (83.9%). Its accuracy makes it a compelling choice, especially in applications where precision and recall are crucial metrics.

Naive Bayes maintained a balanced performance with an accuracy of 86.9%, displaying reliability in classification tasks. KNN demonstrated competitive performance with an accuracy of 93.1%, emphasizing its suitability for local pattern recognition. However, its effectiveness may vary based on dataset characteristics.

While SVM consistently outperformed others in training set metrics, Random Forest presented a compelling alternative due to its ensemble learning approach, offering a balanced combination of accuracy, robustness, and interpretability.

The choice of algorithm depends on specific task requirements. For fraud detection, Random Forest's ensemble approach and ability to capture intricate patterns make it a strong contender, addressing the complexities inherent in such scenarios.

Applicability to Fraud Detection:

• **Feature Importance:** Random Forest provides a measure of significance. Significance is determined based on how well each feature contributes to reducing inequity (e.g., Ginipurity) in

decision trees. The identification of important features can reveal characteristics associated with fraudulent behavior.

• **Nonlinear relationship:** Random forests can capture nonlinear relationships between features and the target variable. This is important in fraud detection where fraud patterns may not follow linear trends. The clustered nature of the random forest enables it to model complex relationships and capture subtle patterns of fraud.

• **Probability Estimation:** Random Forest can provide probability estimates for each category. This is valuable in terms of fraud detection because it provides nuanced understanding of the reliability of the model's predictions. Transactions that appear to be highly fraudulent can be flagged for better monitoring.

• **List of decision trees**: Random Forest is a clustering method consisting of multiple decision trees. Each tree is trained on a subset of the data and results in a vote for the final prediction. This group approach increases the stability of the model and reduces the risk of overfitting. It is particularly useful in fraud detection where patterns can be complex and varied.

Tune the Model:

In training our Random Forest model, making the right adjustments to certain settings, known as hyperparameter tuning, was important for getting the best results. We focused on finding the sweet spot for these settings to strike a balance between accuracy and avoiding common issues like overfitting or underfitting. Here are some details about tweaking these important settings:

1. Number of estimator (Trees): The model was trained with 100 trees (estimators), and this value was determined as optimal through the tuning process. An ensemble of 100 decision trees contributes to the robustness of the model, ensuring accurate predictions.

2. Max Depth of Trees: During hyperparameter tuning, the maximum depth of the decision trees was adjusted to find the optimal value. Controlling the depth helps prevent overfitting, ensuring that the trees generalize well to new, unseen data.

3. Minimum Samples per Leaf: Tuning the minimum number of samples required to be in a leaf node was performed to strike a balance between model complexity and generalization. This parameter influences the granularity of the decision trees.

4. Training and Testing Set Split: The dataset was split into training (80%) and testing (20%) sets during model evaluation. While not a hyperparameter, the choice of the split ratio can impact model performance, and it was determined through iterative testing.

5. Performance Metrics Evaluation: Various performance metrics, including accuracy, recall, precision, error rate, F1 Score, and the Area Under the Curve (AUC) score, were carefully monitored during hyperparameter tuning. Adjustments were made to ensure optimal values for these metrics.

Overall, this tuning process involved making thoughtful adjustments to these settings, ensuring that our Random Forest model is well-tuned and can make accurate predictions even on new data. The choices we made, like having 100 trees and tweaking tree characteristics, contribute to the model's effectiveness in spotting fraud.

3. For recommendations related to your problem statement based on your analysis (what can users learn from your product, how does it help them solve problems related to your problem statement, other ideas for how to extend your project, or other avenues that could be explored related to the problem)

Answer:

Recommendations:

What users can learn from our website:

1. **Fraud Detection and Prevention**: Users can input transaction details through the website or CSV files to determine whether a transaction is fraudulent or not. The system employs sophisticated algorithms to analyze time-based features, transaction features, value features, and ERC20 features, enhancing its ability to detect anomalies and potential fraud.
2. **Improved Security Measures:** By identifying patterns in fraudulent transactions, security teams can implement targeted measures to strengthen security protocols and reduce the likelihood of future fraud incidents. The system can provide recommendations for enhanced security measures based on the analysis of past fraudulent transactions.
3. **Enhanced Customer Experience:** Companies can use the insights gained to improve their overall security infrastructure, ensuring a safer environment for their customers. Identifying and preventing fraud contributes to a more secure and trustworthy financial ecosystem, enhancing customer confidence in the company's services.
4. **Cost Savings:** Proactive fraud prevention can lead to significant cost savings by reducing financial losses associated with fraudulent transactions. Companies can allocate resources more efficiently by focusing on areas with a higher likelihood of fraudulent activities.

How does it help them:

This project can be used in various industries like banking, e-commerce, fintech, insurance, and healthcare by providing effective fraud detection, ensuring security and trust.

Banking: Enables banks to analyze transaction data and proactively identify and prevent fraudulent activities, safeguarding financial transactions and maintaining the institution's reputation.

E**-commerce**: Empowers e-commerce platforms to detect and prevent fraudulent transactions, reducing the risk of chargebacks and ensuring a secure online shopping experience for customers.

Fintech: Enhances cybersecurity measures for fintech companies, helping them identify patterns of fraudulent behavior and protect their platforms from financial crimes, fostering user trust.

Insurance: Assists insurance providers in analyzing claims data to identify irregularities and patterns indicative of fraudulent activity, reducing financial losses and maintaining fair pricing for customers.

Healthcare: Aids healthcare organizations in detecting and preventing fraudulent activities in billing and insurance claims, ensuring resources are allocated to genuine healthcare needs and safeguarding patient data.

Other ideas for how to extend your project:

1. **Real-time Monitoring and Alerts:** Implement real-time transaction monitoring with instant alerts to enable users to respond swiftly to potential fraud, minimizing the impact of suspicious activities.
2. **Behavioral Analysis for User Profiling:** Enhance fraud detection by incorporating behavioral analysis for user profiling, identifying anomalies in user behavior and patterns indicative of fraudulent activities.
3. **Cross-Platform Compatibility:** Ensure compatibility across various platforms (web, mobile, API) for seamless integration, making your fraud detection solution accessible and adaptable to different user environments.
4. **Continuous Machine Learning Refinement:** Establish a continuous learning system that refines machine learning models using new data, ensuring adaptability to evolving fraud tactics and improving overall accuracy.
5. **Collaboration with Law Enforcement:** Facilitate collaboration with law enforcement agencies to streamline the reporting and investigation process, enhancing the overall effectiveness of fraud detection and prevention efforts.

Other avenues that could be explored related to the problem:

1. Biometric Integration: Explore the integration of biometric authentication, such as fingerprints or facial recognition, to enhance the security of transactions and reduce the risk of unauthorized access.

2. Real-time Anomaly Detection: Implement real-time monitoring and anomaly detection features to promptly identify unusual transaction patterns, allowing for immediate response to potential fraud.

3. Blockchain for Transparent Transactions: Investigate the use of blockchain technology to create a transparent and tamper-proof transaction ledger, providing an added layer of security and accountability.

4. Collaborative Fraud Intelligence Sharing: Foster collaborations with other industry players to share insights and intelligence on emerging fraud trends, enabling a collective and proactive approach to fraud prevention.

5. Customer Security Awareness Program: Develop educational resources and platforms to raise customer awareness about secure transaction practices, reducing the risk of falling victim to social engineering or phishing attacks.