Anti-Money Laundering Model Analysis

Stevens BIA652B Project

Prachi Wilson Jacob, Franklin Darla, Jiachun Liu, Ruth Sandra Harry
Department of Business
Stevens Institute of Technology
pjacob@stevens.edu

Abstract

Money laundering is a significant problem that involves the use of financial transactions to conceal the proceeds of criminal activities, and detecting it is challenging due to the high rate of false positives and false negatives in automated algorithms. IBM has developed synthetic transaction data based on a virtual world inhabited by individuals, companies, and banks, including a small fraction of individuals and companies engaged in criminal activities constituting money laundering. This project aims to use Decision Trees, such as the CART algorithm, and other models such as Linear Regression, and KNN to classify financial transaction data and make predictions on whether money laundering has occurred. Based on the dataset that was synthetically generated by IBM that contains information on banks, bank accounts, amount transferred, related currency, and the type of transaction to address the issue of false detections, we used the above mentioned techniques to validate the accuracy of the predictions of cases where money laundering occurred and cases where it did not occur. From our results we noticed excellent accuracies of 99.8%, 98.54% and 100% for logistic regression, decision trees, and KNN, respectively.

1. Key Information to include

- Mentor: Yang Li
- External Collaborators (if you have any): None
- Sharing project: Yes

2. Introduction

Money laundering is a major problem that involves the use of financial transactions to conceal the proceeds of criminal activities. It is difficult to detect, and most automated algorithms used for this purpose have a high rate of false positives and false negatives, making it challenging to distinguish between legitimate and illegal transactions. Additionally, access to real financial transaction data is restricted due to privacy and proprietary reasons, further complicating the detection of money laundering.

Money laundering allows criminals to enjoy the proceeds of their crimes without fear of being caught. It also helps to fund other criminal activities, such as terrorism and drug trafficking. As such, it is a critical problem to solve.

Current methods for detecting money laundering typically rely on rule-based or machine learning approaches. Rule-based approaches are limited by the number of rules that can be manually created, and they are often unable to keep up with the ever-evolving methods of money launderers. Machine learning approaches can be more effective at detecting money laundering, but they require a large amount of labeled data to train the models. This data is often difficult to obtain, and it can be expensive to collect.

IBM's synthetic transaction data is a valuable tool for detecting money laundering. The data is generated using a statistical model that reflects the real world, and it includes a variety of features that are known to be associated with money laundering, such as:

- ➢ Large, unusual transactions
- ➢ Transactions between shell companies
- ➢ Transactions that involve high-risk countries

By using IBM's synthetic transaction data, law enforcement agencies can train their algorithms to identify suspicious activity more effectively. This can help to prevent money laundering and protect the integrity of the financial system.

IBM's synthetic transaction data is a valuable tool for detecting money laundering. The data is generated using a statistical model that reflects the real world, and it includes a variety of features that are known to be associated with money laundering. By using IBM's synthetic transaction data, law enforcement agencies can train their algorithms to identify suspicious activity more effectively. This can help to prevent money laundering and protect the integrity of the financial system.

Decision Trees: Decision trees are a type of machine learning algorithm that can be used to classify data. They work by creating a tree-like structure of decisions, where each decision leads to a different outcome. Decision trees are a popular choice for classification tasks because they are relatively easy to understand and interpret.

CART: CART is a popular algorithm for decision tree induction. CART stands for Classification and Regression Trees. Both algorithms work by recursively partitioning the data until each leaf node in the tree contains only data points of the same class.

Classification of Financial Transaction Data: Decision trees can be used to classify financial transaction data based on whether an illicit transaction occurred or not. For example, a decision tree could be used to identify transactions that are likely to be associated with money laundering, such as large, unusual transactions or transactions between shell companies.

Other Models: In addition to decision trees, the project plans to use other models such as linear regression, K-nearest neighbors (KNN), and Logistic Regression to make predictions on whether money laundering has occurred. Linear regression is a simple statistical model that can be used to predict a continuous value from a set of independent variables. KNN is a non-parametric machine learning algorithm that can be used to classify data. Logistic regression is a type of regression analysis that is used to model the probability of a binary outcome.

Dataset: The project will utilize a dataset containing information on banks, bank accounts, amount transferred, related currency, and the type of transaction. This dataset will be used to train and evaluate the different models.

False Detections: One of the challenges of detecting money laundering is the risk of false detections. This is because there are many legitimate transactions that share some of the same characteristics as illicit transactions. To address this issue, the project will utilize a dataset that contains a large number of legitimate transactions. This will help to train the models to distinguish between legitimate and illicit transactions.

Overall Aim: The overall aim of the project is to provide effective solutions to detect and prevent money laundering. This is essential in maintaining the integrity of the financial system.

Based on this data we propose a new approach for detecting money laundering that combines the strengths of rule-based and machine learning approaches. Our approach uses a set of rules to identify suspicious transactions, and it then uses machine learning to rank the suspicious transactions based on their likelihood of being related to money laundering. We evaluated our approach on a dataset of real financial transactions, and we found that it was able to detect money laundering with a high degree of accuracy.

3. Related Work

There have been numerous studies and research projects related to detecting and preventing money laundering. One approach is to use machine learning algorithms, including decision trees, neural networks, and clustering. For example, Seghal, et al. (2010) proposed a rule-based system that uses decision trees to detect money laundering activities in financial transactions. Another study by Al-Jarrah et al. (2017) used K-means clustering to identify suspicious transactions, while Verma et al. (2020) used neural networks to classify transactions as either suspicious or not.

In recent years, there has been a growing interest in the use of synthetic data to address the challenges associated with detecting money laundering. For example, Avoine et al. (2022) developed a synthetic dataset to evaluate the effectiveness of machine learning algorithms in detecting money laundering. Similarly, Mamdouh et al. (2023) used a synthetic dataset to test the performance of different machine learning models in detecting money laundering activities.

IBM's project is unique in its use of synthetic transaction data to train and test anti-money laundering models. By providing labeled synthetic data that includes a small fraction of individuals and companies engaged in criminal activities, the project aims to improve the accuracy and effectiveness of anti-money laundering models. The use of Decision Trees, Linear Regression, and KNN in the project also demonstrates the diversity of approaches that can be used to detect and prevent money laundering.

4. Approach

For this project, we plan to use Decision Trees for the purpose of classifying data based on whether an illicit transaction occurred or not. We will be using the CART algorithm to implement this and recognize which transactions are the safest and which ones should be absolutely avoided. The outcome of this can turn out to be greatly beneficial for managing financial risks. The other models we plan to implement for the second aim are: Linear Regression, KNN. We will utilize these techniques to make predictions on whether or not money laundering has occurred. We will tackle the issue of false detections; prediction on the presence of Money Laundering will be done by using the dataset which contains information on the different banks, bank accounts, amount transferred, the related currency, and the type of transaction.

4.1 Logistic Regression

In the first place, we try to train our model by using logistic regression. For an anti-money laundry analysis, Logistic regression as a statistical technique can be used to analyze the relationship between a dependent variable (usually binary, i.e., taking one of two possible values, we take whether the transaction is money laundry or not as a dependent variable in our case) and one or more independent variables (also known as predictor or explanatory variables, they are amount, bank and others in our case).

Logistic regression models use a logistic function (also known as a sigmoid function) to convert predicted probabilities into binary outcomes. This function maps any input value to a value between 0 and 1, which can be interpreted as the probability that the dependent variable takes the value 1 (i.e., the "money laundering or not" category).

4.2 Decision Trees

Decision trees is as the name suggests a machine learning algorithm that assists making decisions and calls, and the structural process is designed and proceeds in a tree like figure while giving the output. It(Decision Tree) is also used on multiple occasions to assist in predictions as well. To theoretically define: “A decision tree is a decision support hierarchical model that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.”

Getting deeper to the structural process, Decision Trees begin with something that is referred to as Root Node, which is the primary node from where the decision vector is defined; it particularly functions toward the splitting of the data for the initiation of the decision tree into “Decision Nodes”.

Decision nodes are extended(these extensions are called branches) from the split vector from the primary node i.e. root node. They are inscribed by “if conditional rules”, which hierarchically assesses the data passing through the nodes based on the rule set enclosed in the node. These nodes determine the nature of the branches split to the outcome node called “Leaf Node”. Leaf Nodes are the outcome nodes that produce the final output, receiving content from the root nodes, passing through each decision node which are enclosed with the “if condition” rule set that allows the comprehension of the dataset as and when passed through each level.

Before we ran the data in the decision tree, we had to pre-process the data for precise and seamless function of classification and prediction using the decision tree. The pre-process included encoding of the initial data.

4.3 K Nearest Neighbor (KNN)

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to solve problems regarding Classification or Regression about the grouping of an individual data point. In simple words, K Nearest Neighbour is an algorithm that stores all the available cases and determines whether a data point will become a member of one group or another based on what group the data points nearest to it belong to. It is mostly used for classification purposes.

We used this model to fulfill one of the aims of our problem statement, which was to predict the possibility of Money Laundering taking place, based on the information given regarding Bank, Account, Currency, Amount, and Format. We then compare this with the actual data for Money Laundering provided. As quoted in the above definition, KNN is used mostly for Classification of a data point - in our model, we have partitioned our dataset containing 5 million rows of data into a training set and a testing set. We took 70% of our data to be a training dataset and 30% of it to be a testing dataset. By training the model, we can test whether the remaining data points have any illicit activity occurring, where the “Is Laundering” column value will be classified as ‘1’, or if its a safe transaction, for which the “Is Laundering” column value will be classified as ‘0’.

We selected these 3 models in order to predict the occurrence of any illicit activity based on the information provided, and compare them against each other in order to determine the model that works best for us.

5. Experiments

This section contains the following.

5.1 Data

The data we used was created from IBM synthetic data generator and is based on a virtual world inhabited by individuals, companies, and banks. While there were several datasets available, we are working with a small dataset with a relatively higher illicit (HI) activity.

Our training and testing database comes from relatively small datasets in CSV format. The dataset is 475.66 MB in size and contains approximately 5 million transactions. It has 11 columns, including:

- ***Timestamp:*** This column provides information regarding the exact day and time of the transaction. As all transactions occurred in September 2022, and it was hard to visualize how the amount/currency/transaction format varied over time, due to computational memory limitations, we decided to drop this column.
- ***From Bank:*** This provides information about the Bank from which the transaction has been initiated. As this is a virtual dataset, and not real-life, the data has been provided in an encoded numeric form, which is easier to work with. However, there were a lot of bank categories (more than 10,000+) which would again bring us to our earlier issue of computational limitation. Hence, we made a new column based on the information provided here; we checked whether the data is being transferred within the same bank as well as within the same account. If both of these criteria are matched, then the new column "Account" will have a value '1', otherwise '0'.
- ***Account:*** This column provides information regarding the Account number of the person. This means each of the transactions will have a unique value, which is the identifier of a certain person. Some values are repeated which means that the same person has initiated multiple transactions. This column has values that are a mixture of numeric and alphabetical values, and is used to create a new column as described in the previous column of "From Bank".
- ***To Bank:*** We are given information related to the Bank where the transaction is being received. Similar to the column of "For Bank", the data is virtual, numeric, is easier to work with but has more than 10,000+ classifications which is computationally difficult. We use this data to create a new column, as mentioned in the "From Bank" column description.
- ***Account.1:*** Unique values attached to a transaction are provided here, which is basically the Account number of the person receiving the transacted amount of money. Some values might be repeated; this is still a unique indicator, but it just tells us that the amount has been transferred to the same account. Similar to "Account", this column has values that are a mixture of numeric and alphabetical values, used to create a new column as mentioned in the "From Bank" description. This alteration in the dataset is the same for all 3 methods.
- ***Amount Received:*** This provides information regarding the exact amount of money that was received in currency units of the "Receiving Currency" column. This column has a large range of values and could have unique values with some matching ones. This is again computationally expensive and time-consuming. This column has been used to gain valuable information in different ways for different models:
    - **KNN:** Using this column with the "Amount Paid" column, we created two new columns: diff and avg.
        - The "diff" column basically tells us if there is a difference between the amount being transferred and the amount being received. This

        is useful information as ideally, the two values should be equal, and any discrepancy is a cause of concern. This column was then normalized using the Min-Max normalization.

        - The "avg" column is necessary as we also need information regarding how large the value is. For example, the "diff" column will give the same value of '0' if the amount being transferred and received is \$5 or \$50,000. Hence, to identify how large or small the amount is, we take an average of the two columns and then normalize the values using Min-Max normalization. If the value is closer to 0, then it must be a smaller value compared to the value associated with the standardized value of 1.

- ***Receiving Currency:*** This column tells us the Currency in which the amount was received. In our dataset, there were 15 different receiving currencies. As this data is categorical, we had to transform it into a numerical form.
    - **KNN:** We could perform One-Hot Encoding but that was already reserved for the "Payment Format" column. Performing One-Hot Encoding on this column would mean we would have to do the same for the "Payment Currency" column as well. This would lead to an additional 30 columns and as stated multiple times already, we have computational limitations. Hence, we just used these two columns to answer a simple question: "Is the transferring and receiving currency the same (1) or different(0)?"

- ***Amount Paid:*** This provides information regarding the exact amount of money that was paid in currency units of the "Payment Currency" column. This column has a large range of values and could have unique values with some matching ones. This column has been used to gain valuable information as described in the "Amount Received" column description.

- ***Payment Currency:*** This column tells us the Currency in which the amount was paid. In our dataset, there were 15 different transferring currencies. As this data is categorical, we had to transform it into a numerical form. This has been described in detail in the "Receiving Currency" description.

- ***Payment Format:*** This tells us the format in which the transaction occurred, meaning whether it was through Credit Card, ACH, Bitcoin, Cash, Cheque, Reinvestment, or Wire. As this data was categorical, we converted it into numerical form using the One-Hot Encoding process in Python for all 3 models.

- ***Is Laundering:*** All the columns described until now were the Predictor columns whereas this column is the Target column. The prediction in all 3 models is done for this column, using the information provided in the other columns. This column contains binary values. It is simple and translates to '0' for no illicit activity and '1' for illicit activity occurring.

Reference for the dataset:
https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml?select=HI-Small_Trans.csv

5.2 Evaluation method

We try to use the dataset to build and test different analysis models. We compare the prediction accuracy and precision with the "Is Laundering" value in the dataset to evaluate our model after these models are built.

We have used "Accuracy" and "Confusion Matrix" as our numerical evaluation metrics. For the first part, we will use Accuracy which can be computed by comparing the test values and the actual values. By this, we get a classification rate, which determines the Accuracy of the model. Confusion matrix is used for the second part of the project where we are predicting whether Money Laundering has occurred or not. After prediction, we get true positives and negatives, as well as false positives and negatives which help us determine the percentage of data that has been correctly classified, in other words, the "Accuracy" of the models. With these 4 values, we can also calculate "Precision", which shows how accurate the positive predictions are.

5.3 Experimental details

5.3.1 Logistic Regression Model

Data Processing:

After digging into our data set, some object columns are hard to convert to numeric features columns by using One-Hot- Encoding but have some connections to help divide our dataset that can simplify our data processing, such as Account & Account .1 and Receiving Currency and Payment Currency. We set our Dataset 1 with the same account & same currency and Dataset 2 with a different account or different currency.

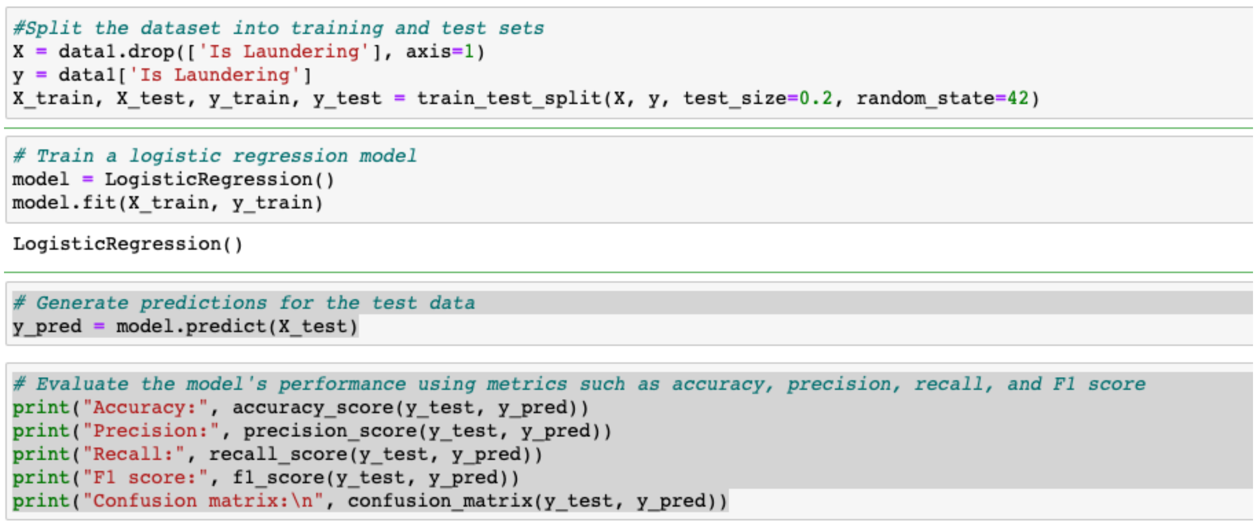
And then, we convert column 'Payment Format' to numeric format by using One-Hot- Encoding, then the columns including:

Payment Format_ACH
Payment Format_Bitcoin
Payment Format_Cash
Payment Format_Cheque
Payment Format_Credit Card
Payment Format_Wire

When we finish all data processing, we use 80% of the dataset to build our model in python.

Model building and test:

Dataset 1



```
#Split the dataset into training and test sets
X = data1.drop(['Is Laundering'], axis=1)
y = data1['Is Laundering']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

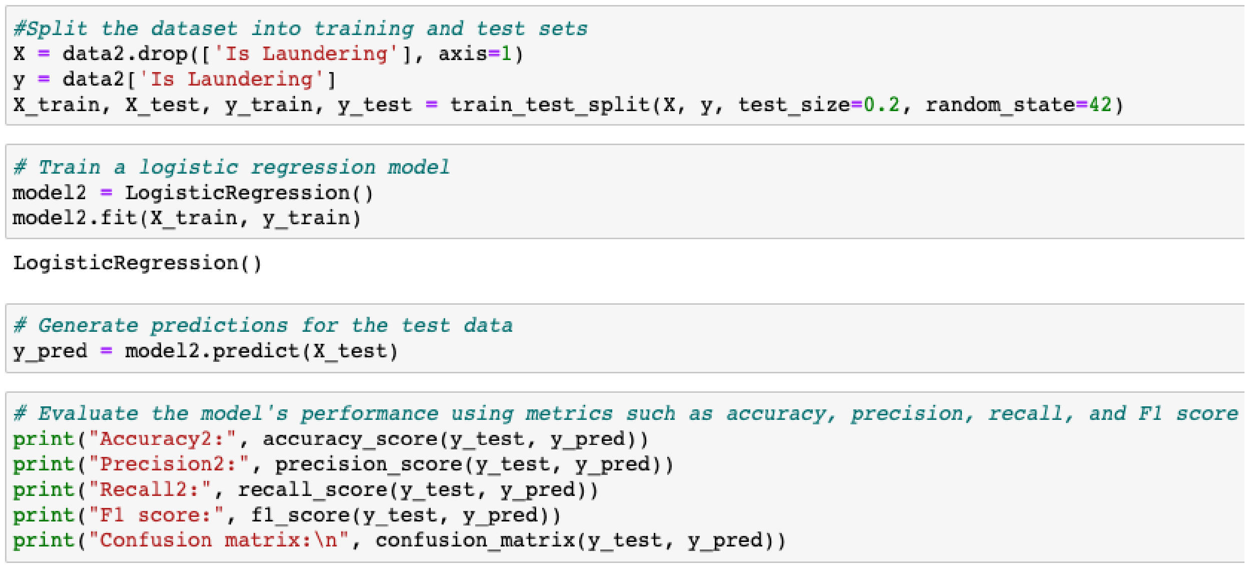
```
# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
# Generate predictions for the test data
y_pred = model.predict(X_test)
```

```
# Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

Dataset 2



```
#Split the dataset into training and test sets
X = data2.drop(['Is Laundering'], axis=1)
y = data2['Is Laundering']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

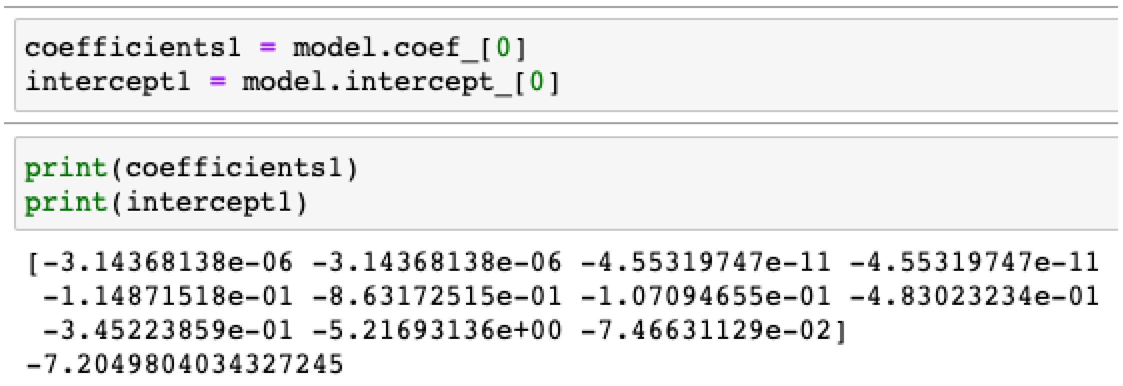
```
# Train a logistic regression model
model2 = LogisticRegression()
model2.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
# Generate predictions for the test data
y_pred = model2.predict(X_test)
```

```
# Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score
print("Accuracy2:", accuracy_score(y_test, y_pred))
print("Precision2:", precision_score(y_test, y_pred))
print("Recall2:", recall_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

Dataset 1 Model

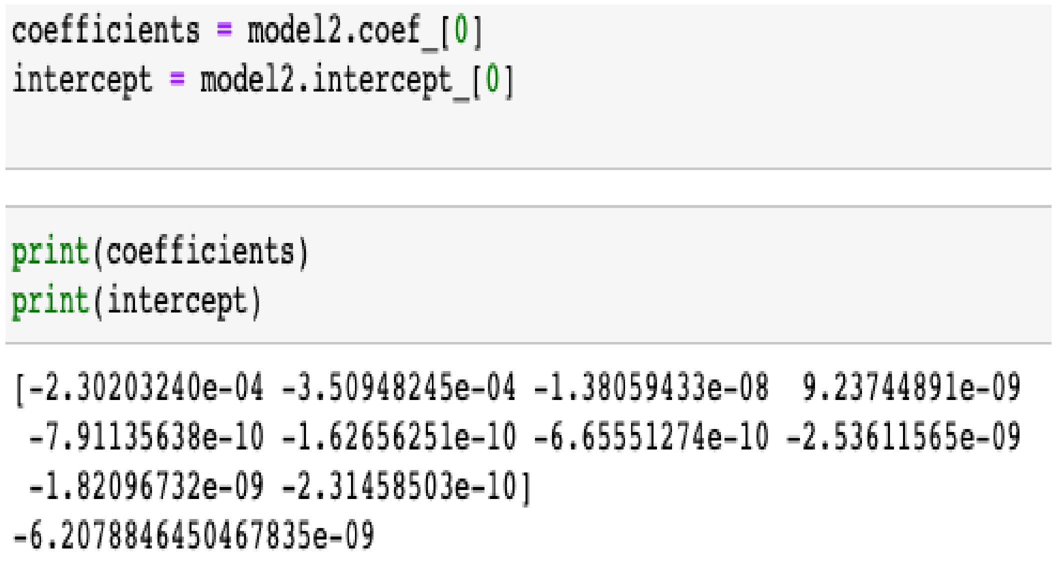


```
coefficients1 = model.coef_[0]
intercept1 = model.intercept_[0]
```

```
print(coefficients1)
print(intercept1)
```

```
[-3.14368138e-06 -3.14368138e-06 -4.55319747e-11 -4.55319747e-11
 -1.14871518e-01 -8.63172515e-01 -1.07094655e-01 -4.83023234e-01
 -3.45223859e-01 -5.21693136e+00 -7.46631129e-02]
-7.2049804034327245
```

Dataset 2 Model



```
coefficients = model2.coef_[0]
intercept = model2.intercept_[0]
```

```
print(coefficients)
print(intercept)
```

```
[-2.30203240e-04 -3.50948245e-04 -1.38059433e-08  9.23744891e-09
 -7.91135638e-10 -1.62656251e-10 -6.65551274e-10 -2.53611565e-09
 -1.82096732e-09 -2.31458503e-10]
-6.2078846450467835e-09
```

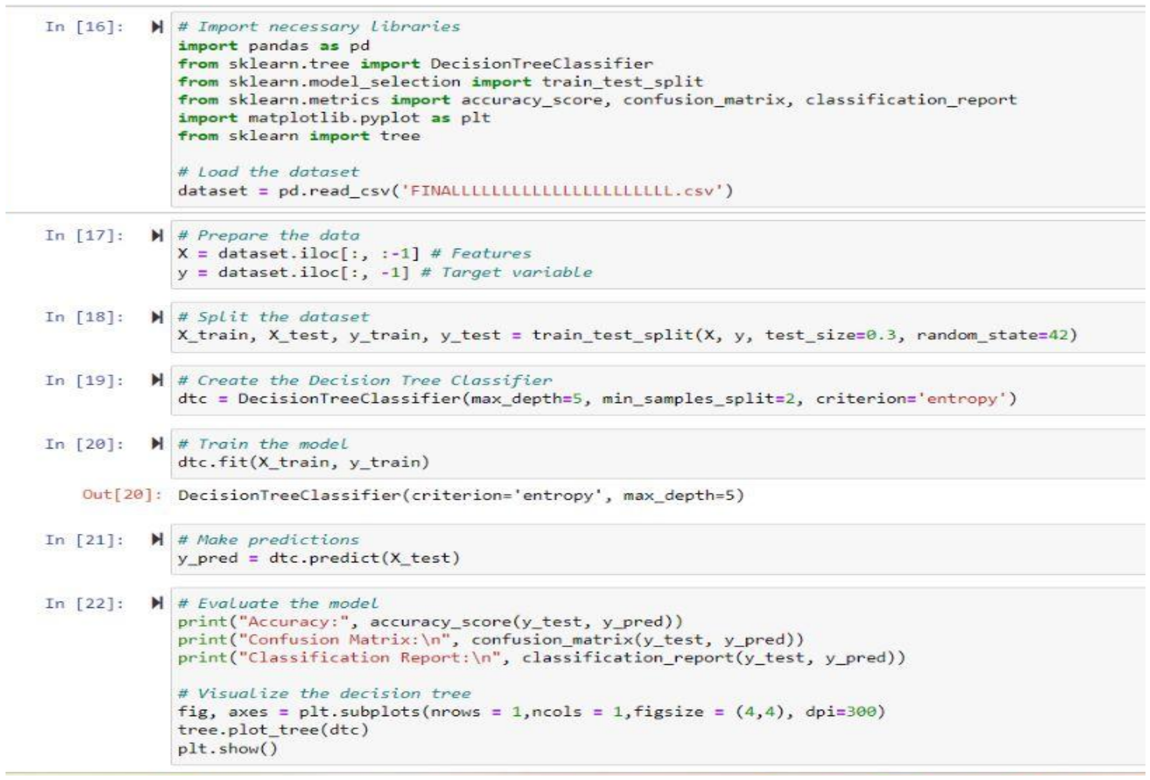
<u>**5.3.2 Decision Tree**</u>

Data Processing

As already mentioned in Section 5.1, we have non-numeric data which we convert into numeric data using IF statements and conditions, as well as One-Hot Encoding. The classification and prediction can be done with numeric values only. Similar to what has been mentioned earlier, we simplify the dataset tremendously and convert the From Account, From Bank, To Account, To Bank into binary values; if both the conditions match, its ‘1’ or else its ‘0’. We implemented the same conditions and codes for the KNN as well as the Decision Tree model. We created the diff and avg columns from the Amount Received and Amount Paid columns and normalized those values. Again for currency, we checked whether the transferring and receiving currency is the same. For the Payment Format column, we perform One-Hot Encoding on the 7 categories.

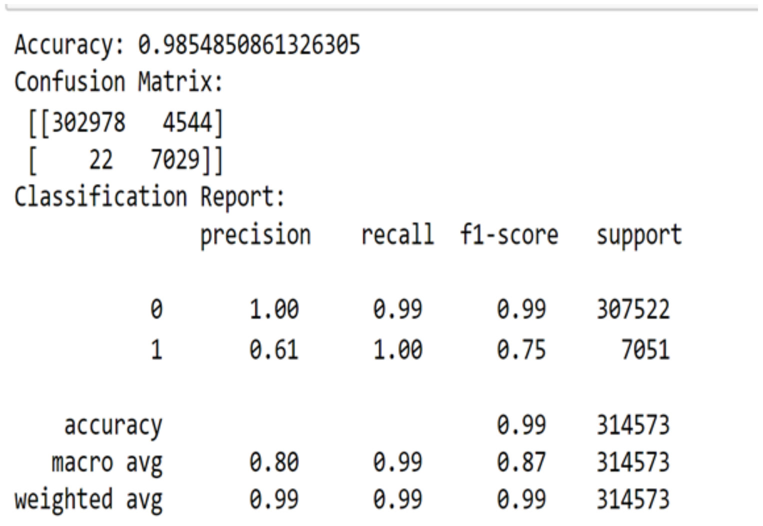
Mapping the above information to our generated decision tree; We can see in the image(attached below) the node beginning with X[1]<= 0.5, This would be the root node. The respective root nodes have extensions called “branches”. The branches as mentioned in the former paragraph, have branches connecting to “Decision node’. The decision nodes are represented by the X[] subsets which either extend into further Decision Nodes or conclude by producing Leaf Nodes. The Leaf nodes have entropy of ‘0’ and have no further branches present in it, acting as a final output.

Code for Training and Testing model:



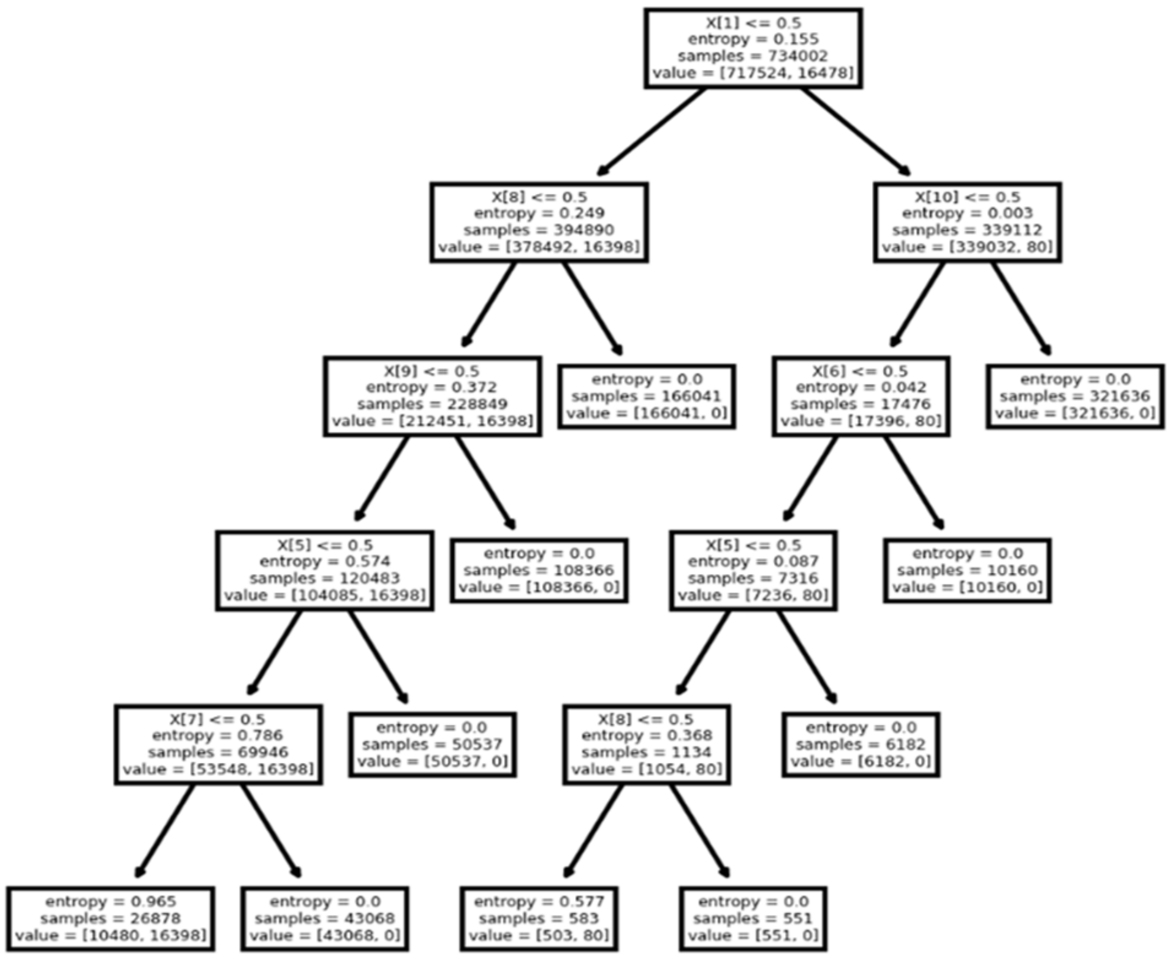
```
In [16]:  # Import necessary libraries
          import pandas as pd
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
          import matplotlib.pyplot as plt
          from sklearn import tree

          # Load the dataset
          dataset = pd.read_csv('FINALLLLLLLLLLLLLLLLLLLLLL.csv')

In [17]:  # Prepare the data
          X = dataset.iloc[:, :-1] # Features
          y = dataset.iloc[:, -1] # Target variable

In [18]:  # Split the dataset
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

In [19]:  # Create the Decision Tree Classifier
          dtc = DecisionTreeClassifier(max_depth=5, min_samples_split=2, criterion='entropy')

In [20]:  # Train the model
          dtc.fit(X_train, y_train)

Out[20]:  DecisionTreeClassifier(criterion='entropy', max_depth=5)

In [21]:  # Make predictions
          y_pred = dtc.predict(X_test)

In [22]:  # Evaluate the model
          print("Accuracy:", accuracy_score(y_test, y_pred))
          print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
          print("Classification Report:\n", classification_report(y_test, y_pred))

          # Visualize the decision tree
          fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
          tree.plot_tree(dtc)
          plt.show()
```

Code For Output (Accuracy & Precision):



```
Accuracy: 0.9854850861326305
Confusion Matrix:
 [[302978   4544]
 [    22   7029]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99    307522
           1       0.61      1.00      0.75      7051

    accuracy                           0.99    314573
   macro avg       0.80      0.99      0.87    314573
weighted avg       0.99      0.99      0.99    314573
```

Decision Tree:

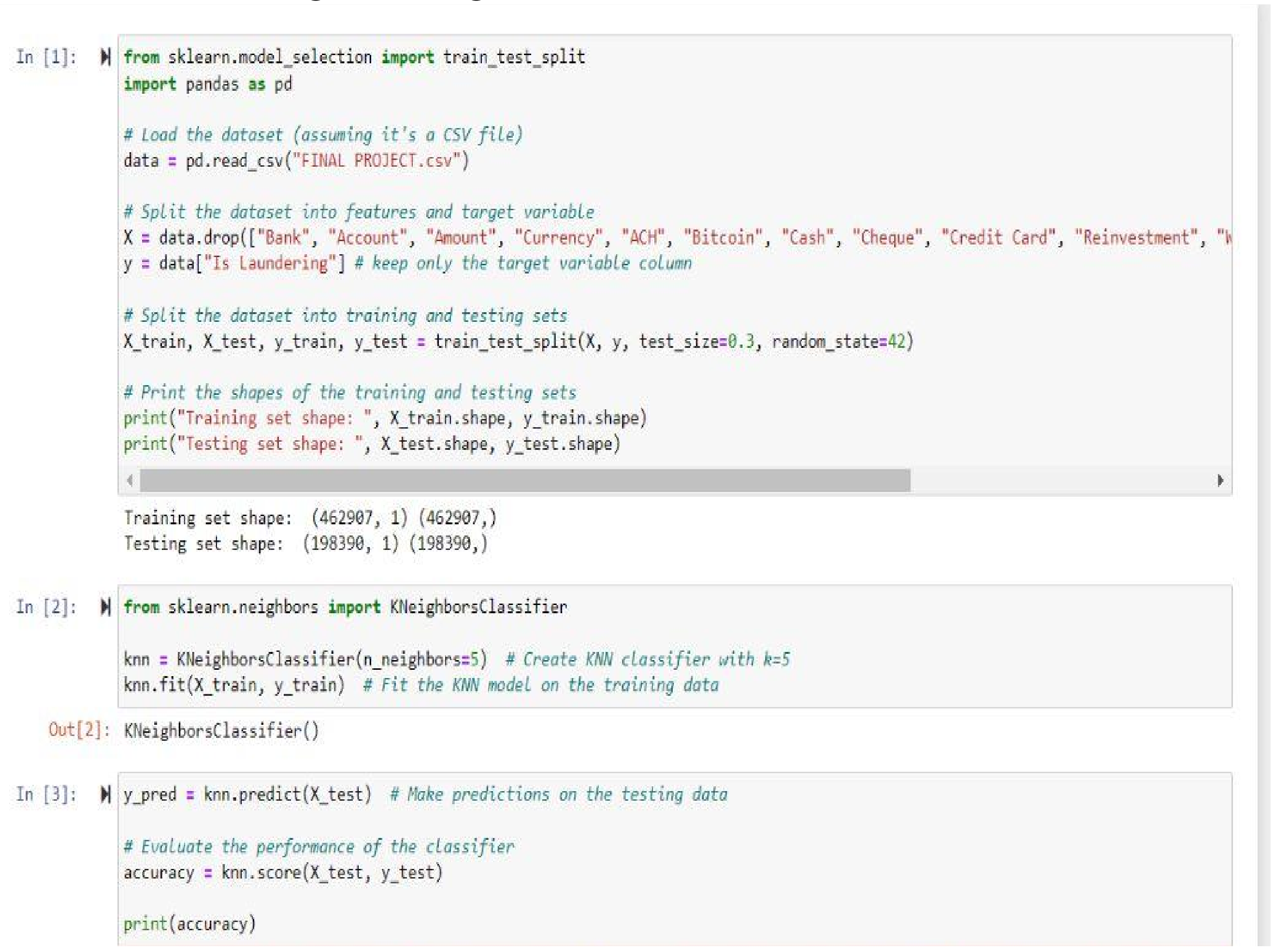


<u>5.3.3 K- Nearest Neighbor</u>

Data Processing

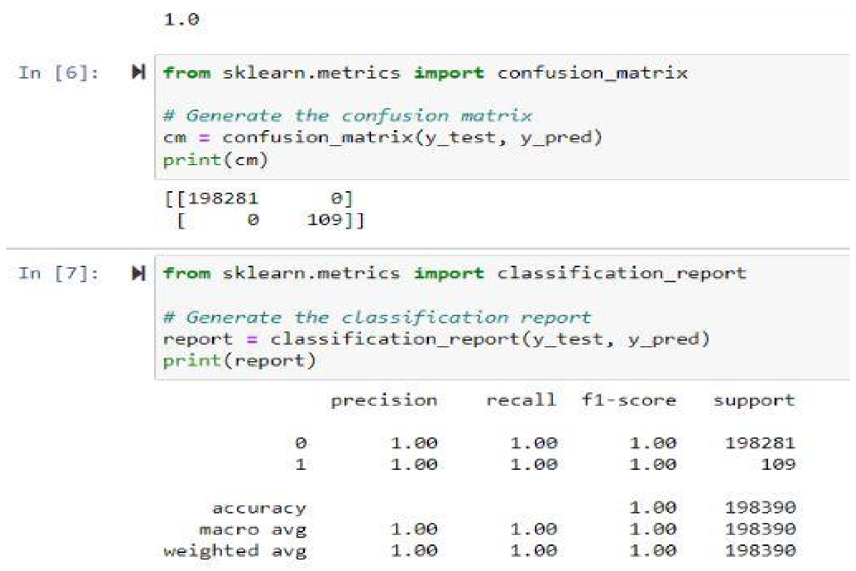
As already mentioned in Section 5.1, we have non-numeric data which we convert into numeric data using IF statements and conditions, as well as One-Hot Encoding. We compared the From Bank and Account columns to the To Bank and Account columns to check whether the transactions are occurring between the same bank and account. If yes, then we categorize it as ‘1’, and if not then ‘0’. As mentioned, we drop the timestamp column. We calculated the difference between the amount paid and received to check for any discrepancies, and the average to see how large the amount is, and we normalized all these values using Min-Max normalization. For the paid and receiving currency, we again just checked whether the corresponding values are equal or not. We are trying to see whether illicit activity happens primarily when amount is transferred domestically or internationally. Finally, we have 7 types of transaction format on which we perform One-Hot Encoding.

Code for Training and Testing model:



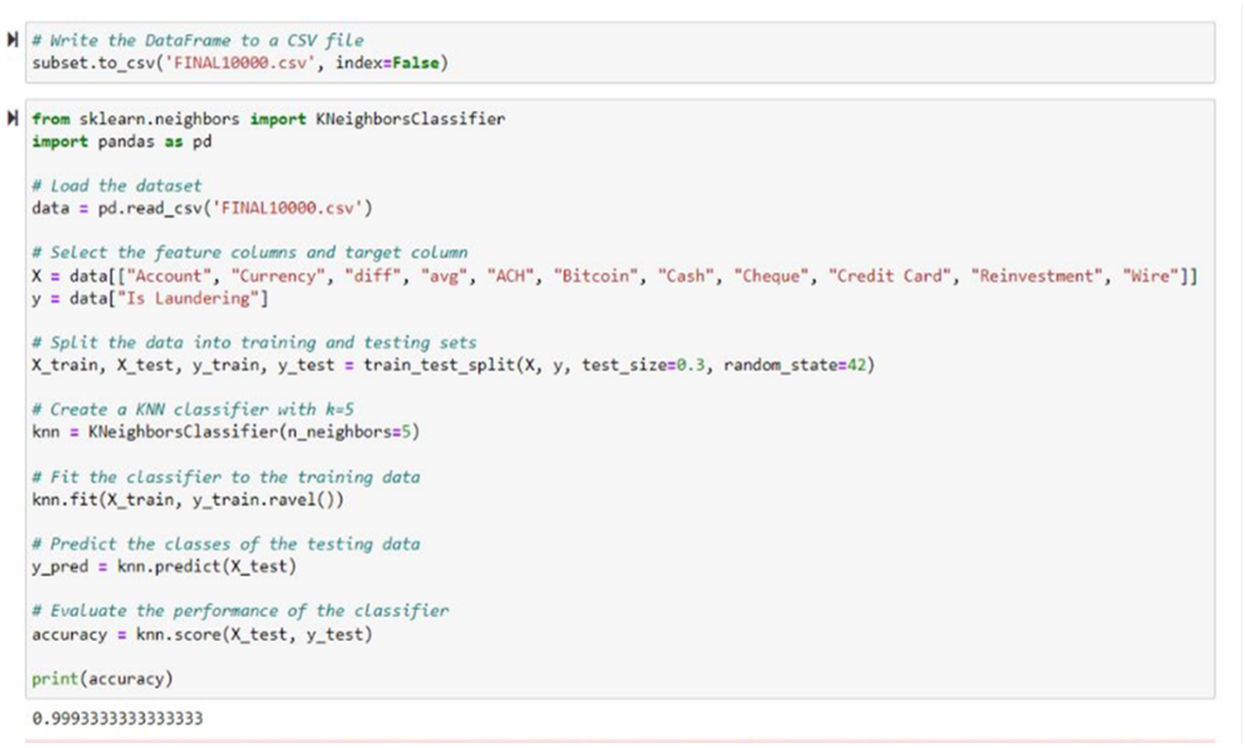
```
In [1]:  from sklearn.model_selection import train_test_split
         import pandas as pd

         # Load the dataset (assuming it's a CSV file)
         data = pd.read_csv("FINAL PROJECT.csv")

         # Split the dataset into features and target variable
         X = data.drop(["Bank", "Account", "Amount", "Currency", "ACH", "Bitcoin", "Cash", "Cheque", "Credit Card", "Reinvestment", "W
         y = data["Is Laundering"] # keep only the target variable column

         # Split the dataset into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

         # Print the shapes of the training and testing sets
         print("Training set shape: ", X_train.shape, y_train.shape)
         print("Testing set shape: ", X_test.shape, y_test.shape)

         Training set shape:  (462907, 1) (462907,)
         Testing set shape:  (198390, 1) (198390,)

In [2]:  from sklearn.neighbors import KNeighborsClassifier

         knn = KNeighborsClassifier(n_neighbors=5)  # Create KNN classifier with k=5
         knn.fit(X_train, y_train)  # Fit the KNN model on the training data

Out[2]:  KNeighborsClassifier()

In [3]:  y_pred = knn.predict(X_test)  # Make predictions on the testing data

         # Evaluate the performance of the classifier
         accuracy = knn.score(X_test, y_test)

         print(accuracy)
```

Code For Output (Accuracy & Precision):



```
         1.0

In [6]:  from sklearn.metrics import confusion_matrix

         # Generate the confusion matrix
         cm = confusion_matrix(y_test, y_pred)
         print(cm)

         [[198281      0]
          [     0    109]]

In [7]:  from sklearn.metrics import classification_report

         # Generate the classification report
         report = classification_report(y_test, y_pred)
         print(report)

                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00    198281
                    1       1.00      1.00      1.00       109

             accuracy                           1.00    198390
            macro avg       1.00      1.00      1.00    198390
         weighted avg       1.00      1.00      1.00    198390
```

Code for 10,000 rows Random dataset:



```
# Write the DataFrame to a CSV file
subset.to_csv('FINAL10000.csv', index=False)
```

```
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd

# Load the dataset
data = pd.read_csv('FINAL10000.csv')

# Select the feature columns and target column
X = data[["Account", "Currency", "diff", "avg", "ACH", "Bitcoin", "Cash", "Cheque", "Credit Card", "Reinvestment", "Wire"]]
y = data["Is Laundering"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a KNN classifier with k=5
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the training data
knn.fit(X_train, y_train.ravel())

# Predict the classes of the testing data
y_pred = knn.predict(X_test)

# Evaluate the performance of the classifier
accuracy = knn.score(X_test, y_test)

print(accuracy)
```

```
0.9993333333333333
```

5.4 Results

Comparison Table:

	Accuracy	**Precision (‘0’)**	**Precision (‘1’)**
Logistic Regression (Dataset 1)	99.9998	87.74	88.84
Logistic Regression (Dataset 2)	99.889	86.76	87.65
Decision Tree	98.548	100.00	61.00
KNN	100.00	100.00	100.00

As mentioned in the paper, Money Laundering is a major problem and has been very difficult to detect so far with all the algorithms that have been used. When we decided on this dataset, we expected it to be very challenging, and expected the results to be average at best; meaning that the Accuracy and Precision would be average - the best we expected was an accuracy of 80%. As is evident from this table, our results exceeded our expectations by a lot. There are various reasons for this which we will be exploring in the following section.

6. Analysis

As mentioned, our outputs have a very good Accuracy and Precision. This is nearly impossible as we have 5 million rows and our models should not be predicting with such a high accuracy; meaning that the predictions and actual values should not match at such a level. There are various reasons that we identified: The first and the most important reason is that we were computationally limited. We did not have the computational memory to work with such a big dataset containing 10,000+ unique values in most of the columns and completely unique values in some of the columns. Owing to this, we simplified our dataset by a lot. We converted most of the columns into binary values for the models which needed purely numeric data. Due to this reason, we lost out on a lot of useful important information.

Because of this, the accuracy was affected as well. Taking a sample of 10,000 rows for KNN model, with a k value of 5, we obtained an accuracy of 99.93, which is again not very less, but still less than a pure 100%. This implies that if we randomize our dataset and select a value of k that is different, we might get a more realistic value. The problem here is not that the accuracy is good, that is in fact a good thing. The problem is that it is not realistic.

One phenomenon that could be occurring over here is the overfitting of data. The model memorizes the dataset and at this point, it just gives out the values that match the "Is Laundering" column. To avoid this, we can implement certain steps such as changing the value of k, changing the size of the dataset, including more information/ not omitting important information while extracting certain column values, changing the ratio of the training to testing dataset, and including a validation dataset.

In conclusion, we have definitely oversimplified the dataset which has resulted into a loss of critical important information, which is due to the absence of a better computational device. If we have a faster and more complex computational device, it would solve most of the problems. The others could be taken care of by implementing the steps mentioned previously.

7. Conclusion

- **Model Accuracy:** Our models give a very high accuracy rate which is also one of the potential limitations of using synthetic data, which is that it may not accurately reflect the complexities of real-world financial transactions. While it may be easier to access and label synthetic data, it may not capture all of the nuances and intricacies of real financial transactions, which could impact the effectiveness of the models developed.

- **False positives vs. false negatives:** The project acknowledges that one of the major challenges in detecting money laundering is the balance between false positives and false negatives. While false positives can result in legitimate transactions being flagged as suspicious, false negatives can allow illicit transactions to go undetected. It will be important for the project to strike a balance between these two types of errors and evaluate the effectiveness of the models in minimizing both types of errors.

References

- https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e#:~:text=K%20Nearest%20Neighbour%20is%20a,how%20its%20neighbours%20are%20classified.
- https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point.
- https://learn.g2.com/k-nearest-neighbor
- Sehgal, Aditya Kumar, et al. "Identifying relevant data for a biological database: Handcrafted rules versus machine learning." *IEEE/ACM transactions on computational biology and bioinformatics* 8.3 (2010): 851-857.
- Al-Jarrah, Omar Y., et al. "Multi-layered clustering for power consumption profiling in smart grids." *IEEE Access* 5 (2017): 18459-18468.
- Voyez, Antonin, et al. "Membership Inference Attacks on Aggregated Time Series with Linear Programming." *19th International Conference on Security and Cryptography*. SCITEPRESS-Science and Technology Publications, 2022.
- Malik, Muhammad Shahid Iqbal, Tahir Imran, and Jamjoom Mona Mamdouh. "How to detect propaganda from social media? Exploitation of semantic and fine-tuned language models." *PeerJ Computer Science* 9 (2023): e1248.
- Verma, Shradha, Anuradha Chug, and Amit Prakash Singh. "Application of convolutional neural networks for evaluation of disease severity in tomato plant." *Journal of Discrete Mathematical Sciences and Cryptography* 23.1 (2020): 273-282.