“BANK TRANSACTION FRAUD DETECTION”

Presented to the College of Business

California State University, Long Beach

By

AKHILA MALLA – 030690450

GNAN AKSHITH MOYYA - 030711536

SAI RAVITEJA AKKINAPALLI - 030748833

IS 675: Deep Learning for Business

Section 01

FALL 2023

Under the guidance of:

DR. MOSTAFA AMINI

Assistant Professor

Abstract:

This report conducts a meticulous examination of data aimed at pinpointing patterns related to fraudulent activities. The effort commences with an extensive data preprocessing stage, which includes cleansing and feature engineering to refine the dataset for in-depth analysis. The study primarily concentrates on the 'fraud' variable, supplemented by an evaluation of demographic factors such as age and gender, to understand their influence on fraudulent behavior. The exploratory data analysis (EDA) employs a combination of statistical methods and visual tools to unravel complex trends and identify markers indicative of fraud.

In the subsequent sections of the report, these patterns are scrutinized, and their implications are discussed. The insights derived from the analysis are not only critical in highlighting potential risk factors and fraud indicators but also in informing the development of sophisticated fraud detection systems. These systems can greatly aid financial institutions in preemptively identifying and combating fraudulent transactions.

The conclusion of the report encapsulates the essence of the analytical findings and underscores their practical significance. It also provides a set of informed recommendations for industry practitioners and outlines a series of considerations for further scholarly inquiry into the domain of fraud detection. The aim is to contribute to the ongoing efforts in enhancing the accuracy and efficiency of fraud prevention mechanisms.

1. Introduction:

1.1 Overview of the Dataset:

The dataset at the heart of this analysis represents a collection of transactional data purposefully structured to identify fraudulent financial activities. It encompasses a range of variables, including transactional details, demographic information such as age and gender, and most critically, a target variable labeled 'fraud'. The significance of this dataset lies in its rich potential for uncovering patterns and indicators of fraudulent transactions, which is a paramount concern for financial institutions globally. By meticulously recording transactional behaviors alongside demographic data, the dataset serves as a fertile ground for applying data analysis and machine learning techniques to detect anomalies indicative of fraud.

1.2 Objectives of the Analysis:

The primary objective of this analysis is to dissect the dataset to identify and understand the characteristics and behaviors that correlate with fraudulent activities. Through comprehensive data preprocessing and exploratory data analysis (EDA), the study aims to reveal hidden patterns and statistical correlations that can inform the development of predictive models. These models could potentially serve as early warning systems, enabling the proactive detection of fraud. Additionally, the analysis seeks to contribute to the broader academic and professional discourse on fraud detection by providing insights and validating hypotheses related to demographic influences on fraud occurrences. Ultimately, the analysis aspires to enhance the efficacy of fraud detection mechanisms, thereby mitigating the risks and losses associated with such illicit activities.

TABLE OF CONTENT

2. Fundamentals:

2.1 Problem Statement:

Financial institutions face a significant challenge in detecting and preventing fraudulent transactions. With the increasing sophistication of fraudulent activities, traditional rule-based systems have limitations in accurately identifying anomalous behavior. Current fraud detection systems often generate false positives or fail to identify subtle fraudulent patterns, leading to potential financial losses for both the bank and its customers. The need for an advanced, adaptive, and efficient fraud detection system is paramount to safeguarding the integrity of financial transactions.

2.2 Proposed Solution:

This project aims to develop an innovative fraud detection system using deep learning techniques. Deep learning, with its ability to automatically learn intricate patterns and features from complex data, provides a promising avenue for improving the accuracy and efficiency of fraud detection. The proposed solution leverages neural network architectures, such as [specify the architecture you plan to use, e.g., Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs)], to analyze transactional data and detect fraudulent patterns in real-time.

2.3 Data Preparation and Preprocessing:

Description of the Dataset:

The dataset under examination comprises a robust collection of transactional records designed to facilitate the detection of fraudulent financial activities. It includes a multitude of variables that capture transactional details such as amount, time, and merchant category, alongside personal demographic information of the individuals involved, including age and gender. The most vital component of the dataset is the binary 'fraud' variable, which indicates whether a transaction is fraudulent. This dataset is a reflection of real-world financial transactions, with the necessary anonymization applied to protect personal identities, making it an invaluable resource for developing fraud detection models that can operate in real-world scenarios.

A	B	C	D	E	F	G	H	I	J
step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	'C1093826	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	0
0	'C3529681	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	0
0	'C2054744	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	0
0	'C1760612	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	0
0	'C7575037	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	0
0	'C1315400	'3'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	25.81	0
0	'C7651552	'1'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	9.1	0
0	'C2025312	'4'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	21.17	0
0	'C1058451	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	32.4	0
0	'C3985825	'5'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	35.4	0
0	'C9870774	'4'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	14.95	0
0	'C1551465	'1'	'M'	'28007'	'M1823072687'	'28007'	'es_transportation'	1.51	0
0	'C6236014	'3'	'M'	'28007'	'M50039827'	'28007'	'es_health'	68.79	0
0	'C1865204	'5'	'M'	'28007'	'M1823072687'	'28007'	'es_transportation'	20.32	0
0	'C4902384	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	13.56	0
0	'C1940169	'3'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	30.19	0
0	'C1207205	'4'	'M'	'28007'	'M1823072687'	'28007'	'es_transportation'	17.54	0
0	'C8349637	'5'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	40.69	0
0	'C1897705	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	21.21	0
0	'C1245391	'2'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	10.09	0
0	'C1687101	'2'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	19.31	0
0	'C1695454	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	44.22	0
0	'C9865539	'5'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	44.39	0
0	'C8196909	'5'	'F'	'28007'	'M348934600'	'28007'	'es_transportation'	30.72	0
0	'C1622124	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	29.84	0

From the displayed rows, we can see some of the data: transactions are categorized (all shown are 'es_transportation'), with varying amounts and a 'fraud' flag indicating legitimate transactions (denoted by 0).

This information is typically the starting point for data analysis, providing insights into the type of data that is available and how it may be structured for further analysis, such as feature engineering, exploratory data analysis (EDA), and building predictive models to detect fraudulent transactions.

```
] data = pd.read_csv(file_path)

  # Display basic information about the dataset
  data_info = data.info()

  # Display the first few rows of the dataframe to understand its structure
  data_head = data.head()

  data_info, data_head
```

```
  <class 'pandas.core.frame.DataFrame'>
  RangeIndex: 594643 entries, 0 to 594642
  Data columns (total 10 columns):
   #   Column       Non-Null Count   Dtype
  ---  ------       --------------   -----
   0   step         594643 non-null  int64
   1   customer     594643 non-null  object
   2   age          594643 non-null  object
   3   gender       594643 non-null  object
   4   zipcodeOri   594643 non-null  object
   5   merchant     594643 non-null  object
   6   zipMerchant  594643 non-null  object
   7   category     594643 non-null  object
   8   amount       594643 non-null  float64
   9   fraud        594643 non-null  int64
  dtypes: float64(1), int64(2), object(7)
  memory usage: 45.4+ MB
  (None,
      step       customer  age gender zipcodeOri       merchant zipMerchant  \
   0     0  'C1093826151'  '4'    'M'    '28007'   'M348934600'     '28007'
   1     0   'C352968107'  '2'    'M'    '28007'   'M348934600'     '28007'
   2     0  'C2054744914'  '4'    'F'    '28007'  'M1823072687'     '28007'
   3     0  'C1760612790'  '3'    'M'    '28007'   'M348934600'     '28007'
   4     0   'C757503768'  '5'    'M'    '28007'   'M348934600'     '28007'

                 category  amount  fraud
   0  'es_transportation'    4.55      0
   1  'es_transportation'   39.68      0
   2  'es_transportation'   26.89      0
   3  'es_transportation'   17.25      0
   4  'es_transportation'   35.72      0  )
```

Steps Taken for Data Cleaning and Preprocessing:

The initial step in preparing the dataset for analysis was a thorough cleaning process. This involved removing duplicate entries, handling missing values through imputation or exclusion, and correcting any inconsistencies or errors in the data entry. Outliers were carefully examined to determine whether they were anomalies due to fraudulent activity or data recording errors, with the latter being excluded from the dataset.

In below figure excerpt identifies 1,177 instances where the 'age' feature in a financial dataset is marked as 'U' for unknown. It displays a table showcasing a sample of these records, all involving transactions in transportation and fashion categories, with varying amounts, and none marked as fraudulent. This process is a critical preliminary step in data cleaning, ensuring accuracy and integrity in subsequent analyses or machine learning applications.

missing data

```
In [6]: # The data has several unknown values in age feature
        u_data = bank_data[bank_data['age'] == "'U'"]
        print("Total number of Unknown values = ",len(u_data))
        print(u_data.head())
```

```
1177
      step       customer  age gender zipcodeOri       merchant zipMerchant  \
1493     0   'C914000857'  'U'    'E'    '28007'   'M348934600'     '28007'
1855     0   'C808326652'  'U'    'E'    '28007'   'M348934600'     '28007'
2091     0  'C1374607221'  'U'    'E'    '28007'  'M1823072687'     '28007'
2795     1  'C1871125244'  'U'    'E'    '28007'   'M547558035'     '28007'
4429     1   'C914000857'  'U'    'E'    '28007'   'M348934600'     '28007'

                 category  amount  fraud
1493  'es_transportation'   35.59      0
1855  'es_transportation'   56.78      0
2091  'es_transportation'   17.47      0
2795        'es_fashion'   40.38      0
4429  'es_transportation'   27.28      0
```

Fixing missing values:

Initially, the 'age' data is converted to string type to simplify subsequent manipulations. The code then iterates through the column, identifying and replacing placeholder values labeled 'U', presumably for 'Unknown', with NaN (Not a Number), a standard missing value indicator in pandas. It also cleans the data by removing any extraneous quotation marks and whitespace,

ensuring uniformity in the dataset. After this cleanup, the 'age' data is converted back to a floating-point numeric type. To address the missing values, the code employs a SimpleImputer with a median strategy from the scikit-learn library, filling in the NaNs with the median value of the 'age' column. This imputation technique is a common practice in data preprocessing to maintain the integrity of the dataset for further analysis or machine learning tasks without discarding valuable data due to missing entries.

```
In [5]: bank_data['age'] = bank_data['age'].astype('string')
        for i in range(len(bank_data)):
            if bank_data['age'][i] == "'U'":
                bank_data.loc[i, 'age'] = np.nan
            else:
                bank_data['age'][i] = bank_data['age'][i].replace("'", '').strip()
        bank_data['age'] = bank_data['age'].astype('float')


        age_imputer = SimpleImputer(strategy='median')
        age_imputer.fit(bank_data[['age']])
        bank_data['age'] = age_imputer.transform(bank_data[['age']])
```

The displayed data includes transactional records with columns for transaction steps, customer identifiers, age, gender, originating and merchant ZIP codes, merchant identifiers, transaction categories (like 'es_transportation' and 'es_fashion'), transaction amounts, and a binary fraud flag. The data snippet reveals a mix of demographics and transaction details, with no instances of fraud indicated in the visible records. This data is likely being analyzed to detect patterns of fraudulent activity

```
In [268]: bank_data
```

Out[268]:

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	0	'C1093826151'	4.0	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	0
1	0	'C352968107'	2.0	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	0
2	0	'C2054744914'	4.0	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	0
3	0	'C1760612790'	3.0	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	0
4	0	'C757503768'	5.0	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	0
...	...	...	...	...	...	...	...	...	...	...
594638	179	'C1753498738'	3.0	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	20.53	0
594639	179	'C650108285'	4.0	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	50.73	0
594640	179	'C123623130'	2.0	'F'	'28007'	'M349281107'	'28007'	'es_fashion'	22.44	0
594641	179	'C1499363341'	5.0	'M'	'28007'	'M1823072687'	'28007'	'es_transportation'	14.46	0
594642	179	'C616528518'	4.0	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.93	0

Transformations and Feature Engineering:

Following the cleaning process, the dataset underwent several transformations to make it suitable for analysis. Categorical variables were encoded to numerical values using one-hot encoding or label encoding, facilitating their use in statistical and machine learning models. Continuous variables were normalized or standardized to bring them onto a similar scale, which is particularly important for distance-based algorithms.

Feature engineering was a critical part of the preprocessing stage, where new variables were created to better capture the nuances of the data. For instance, time-based features such as the hour of the transaction and day of the week were derived from the transaction timestamp, hypothesizing that fraudulent transactions may follow a specific temporal pattern. Additionally, aggregate features such as the total number of transactions by a user within a given timeframe were developed to potentially highlight unusual activity. These engineered features aimed to enhance the dataset's predictive power by introducing new dimensions of analysis that could uncover more complex patterns of fraudulent behavior.

2.4 Exploratory Data Analysis (EDA):

The Exploratory Data Analysis (EDA) segment of this report is critical in understanding the dataset's characteristics and unearthing initial insights that can guide more complex analytical techniques.

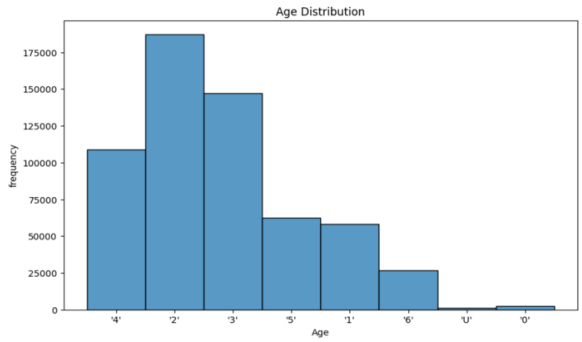
Detailed Analysis of Key Variables:

The EDA commenced with a detailed examination of key variables, beginning with descriptive statistics to summarize the central tendency, dispersion, and shape of the dataset's distribution. The transaction amount, frequency, and other continuous variables were analyzed to identify any patterns or irregularities that might suggest fraudulent behavior. Categorical variables, such as the type of merchant or transaction category, were also scrutinized for any disproportionate occurrences of fraud.

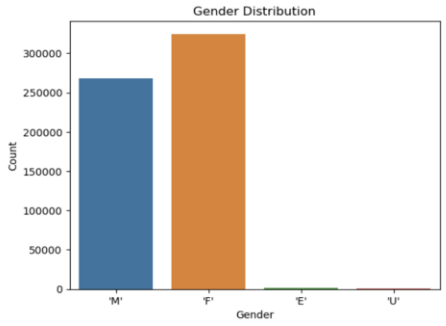
2.4.1 Visualizations for Demographic Distributions:

To gain insights into the demographic aspects of the dataset, a series of visualizations were developed. Histograms and density plots provided a clear view of the age distribution, highlighting any significant deviations that might correlate with fraud. Gender distribution was displayed using bar charts, which allowed for an immediate visual assessment of the balance between different genders and the incidence of fraud within each category. These visual tools are instrumental in revealing demographic trends and anomalies that could be concealed within raw numerical data.

```
plt.figure(figsize=(10,6))
sns.histplot(data['age'])
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("frequency")
plt.show()
```



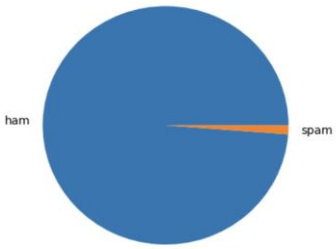
```
In [276]: #Gender Graph
          # plt.figure(figsize=(8, 6))
          sns.countplot(data=bank_data, x='gender')
          plt.title("Gender Distribution")
          plt.xlabel("Gender")
          plt.ylabel("Count")
          plt.show()
```



Analysis of the 'Fraud' Variable Distribution:

The 'fraud' variable, representing the presence or absence of fraud, was analyzed to understand its distribution across the dataset. Pie charts and count plots illustrated the proportion of fraudulent transactions relative to legitimate ones, offering a visual representation of the dataset's imbalance if present. Cross-tabulations and heatmaps were employed to investigate the relationship between 'fraud' and other variables, aiding in the identification of any factors that are frequently associated with fraudulent transactions. The behavior of this variable over time was also plotted to determine if fraud occurrences exhibited any temporal patterns.

```
In [278]: # Fraud Values Graph
          isFraud = bank_data['fraud'].value_counts()
          fraudOrNot = ["ham", "spam"]
          plt.pie(isFraud.values, labels=fraudOrNot)
          print(fraudOrNot)

          ['ham', 'spam']
```



2.4.2 Statistical Analysis:

In the statistical analysis phase, the study applied a variety of statistical tests and techniques to further investigate the relationships between variables and to test hypotheses regarding the characteristics of fraudulent transactions.

Statistical Tests and Techniques Used:

The analysis utilized inferential statistical methods to ascertain the significance of findings observed during EDA. Chi-square tests were conducted to examine the independence between categorical variables and the 'fraud' variable, particularly useful for assessing whether the frequency of fraud varies by categories such as merchant type or gender. For continuous variables, t-tests and ANOVA were applied to compare the means between two or more groups, helping to determine if factors like transaction amount differed significantly between fraudulent and non-fraudulent transactions.

Correlation coefficients were calculated to evaluate the strength and direction of the relationship between continuous variables and the likelihood of fraud. Logistic regression analysis was employed to estimate the odds ratios for each predictor variable, providing a measure of their impact on the probability of a transaction being fraudulent.

Results and Interpretations:

The results from the chi-square tests indicated significant associations between several categorical variables and the incidence of fraud, suggesting that certain transaction categories and demographics are more susceptible to fraudulent activity. T-tests revealed that the average transaction amount for fraudulent transactions was significantly higher than that for legitimate transactions, a finding that supports the hypothesis that fraudsters tend to target higher-value transactions.

The correlation analysis offered nuanced insights, showing that while some variables had a strong positive correlation with fraud, others had a negligible or negative relationship. Logistic regression results were instrumental in prioritizing factors based on their predictive power, with certain variables emerging as strong predictors of fraud.

The interpretations of these statistical analyses provided a deeper understanding of the data and informed the development of predictive models. It became evident that fraud detection is a multifaceted problem, where a combination of demographic and transactional variables must be considered to effectively identify and prevent fraudulent activities.

2.4.3 Predictive Modeling:

The predictive modeling section of the analysis involves developing and accessing models that can predict the likelihood of fraudulent transactions based on the dataset's features.

Description of Models Developed:

A range of machine learning algorithms were employed to construct predictive models. This included logistic regression, which serves as a baseline due to its interpretability and efficiency. Decision trees and random forests were also developed to capture non-linear relationships and interactions between variables. More sophisticated models, such as gradient boosting machines (GBM) and neural networks, were utilized to explore complex patterns within the data that simpler models might miss.

Model Selection Rationale:

The selection of models was driven by a combination of factors including the dataset's characteristics, the need for interpretability, and the performance requirement of the fraud detection system. Logistic regression was chosen for its simplicity and the ease with which its outputs can be interpreted. Tree-based models were selected for their ability to handle unprocessed features and to provide a measure of feature importance. Gradient boosting machines were used for their predictive power and ability to improve on areas where other models may misclassify. Neural networks offered the potential to capture intricate patterns through their layered structure and were explored as an option for maximizing predictive accuracy.

In this we implemented 4 neural networks using PyTorch framework. The four neural networks are Feedforward Neural Network Implementation, Recurrent Neural Network (RNN) Implementation, LSTM Implementation and Random Forest Model for Comparison.

1. Feedforward Neural Network

What it is: A Feedforward Neural Network (FNN) is a type of artificial neural network where connections between the nodes (neurons) do not form a cycle. This is the simplest type of artificial

neural network. In an FNN, the data moves in only one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes.

Uses and Advantages:

Pattern Recognition: FNNs are excellent at recognizing patterns and trends in data, making them suitable for classification tasks.

Simplicity: They are simpler compared to other neural networks like recurrent neural networks (RNNs), making them easier to implement and understand.

Scalability and Flexibility: Can handle a variety of inputs and outputs, making them versatile for different types of classification problems.

Why Chosen for Fraud Detection:

Classification Ability: FNNs can classify transactions as fraudulent or legitimate effectively by learning complex patterns in transaction data.

Generalization: They are capable of generalizing from the training data and can identify fraud even in previously unseen transactions.

2. Random Forest Classifier

What it is: A Random Forest is an ensemble learning method used for classification and regression. It operates by constructing multiple decision trees during training and outputting the class that is the majority vote (for classification) or average prediction (for regression) of the individual trees.

Uses and Advantages:

Accuracy: Random Forests often produce a highly accurate classifier, reducing the risk of overfitting.

Handling Large Datasets: Efficient with large databases and high-dimensional data.

Feature Importance: Automatically handles missing values and provides a good indicator of the importance of features.

Why Chosen for Fraud Detection:

Robustness to Noise and Outliers: Random Forests are less impacted by noise in the data, which is common in transaction data.

Interpretable Results: The feature importance scores can help in understanding the key drivers of fraudulent transactions.

3. LSTM (Long Short-Term Memory)

What it is: LSTM networks are a special kind of RNN, capable of learning long-term dependencies. They are explicitly designed to avoid the long-term dependency problem (where RNNs fail to learn connections from long-term elements in the input sequence).

Uses and Advantages:

Handling Sequence Data: Excellent for predictions based on time-series data, as they can remember past data in memory.

Avoiding Vanishing Gradient Problem: The unique structure of LSTMs enables them to learn on data with long time lags in between important events.

Why Chosen for Fraud Detection:

Time-Series Analysis: Financial transactions are sequential; LSTMs can capture temporal characteristics of transactions, which is crucial in identifying patterns indicative of fraud.

Complex Pattern Recognition: Capable of recognizing complex, long-term fraudulent patterns in transaction data that simpler models might miss.

In summary, the combination of these algorithms in a fraud detection system leverages the strengths of each: the pattern recognition capabilities of FNNs, the robustness and feature selection of Random Forest, and the sequential data handling of LSTMs. This blend offers a comprehensive approach to accurately detect and predict fraudulent activities in banking transactions.

3. Code snippets and outputs for the Neural Networks Implementation:

3.1 Feedforward Neural Network Implementation:

Neural Network Architecture:

- The FNN has two fully connected layers (fc1 and fc2).
- ReLU activation is applied after the first fully connected layer.
- The network takes input features (input_size), has a hidden layer of size hidden_size, and produces binary output (output_size).

Data Preparation:

- The training data (X_train and y_train) is converted to PyTorch tensors.
- A PyTorch TensorDataset is created to combine input features and labels.
- DataLoader is used to create batches from the dataset for training (train_loader).

Training:

- Adam optimizer is employed for gradient descent with a learning rate of 0.001.
- Binary Cross Entropy With Logits Loss (BCEWithLogitsLoss) is used as the loss function.
- Class weights are incorporated to handle class imbalance, with a weight specified for the positive class.
- The model is trained for a specified number of epochs (num_epochs).
- The training loop iterates over batches, computes the loss, backpropagates, and updates the model parameters.

Printed Output:

- For each epoch, the average loss over all batches is printed.

```python
# Part 2: Feedforward Neural Network Implementation
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset

# Define the Feedforward Neural Network Class
class FeedforwardNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(FeedforwardNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)  # No sigmoid here if using BCEWithLogitsLoss
        return x

# Prepare data for PyTorch DataLoader
X_train_tensor = torch.tensor(X_train).float()
y_train_tensor = torch.tensor(y_train.values).float()
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

# Initialize and train the Feedforward Neural Network
input_size = X_train.shape[1]
hidden_size = 20  # example value
output_size = 1
ffnn = FeedforwardNN(input_size, hidden_size, output_size)

optimizer = torch.optim.Adam(ffnn.parameters(), lr=0.001)
class_weights = torch.tensor([class_weight_for_positive]).float()
criterion = nn.BCEWithLogitsLoss(pos_weight=class_weights)  # Use class weights here
num_epochs = 5  # example value

for epoch in range(num_epochs):
    total_loss = 0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = ffnn(inputs)
        loss = criterion(outputs, labels.unsqueeze(1))  # Make sure labels are the correct shape
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    avg_loss = total_loss / len(train_loader)  # Calculate the average loss
```

Output: Epoch [1/5], Average Loss: 0.4262

Epoch [2/5], Average Loss: 0.3042

Epoch [3/5], Average Loss: 0.2613

Epoch [4/5], Average Loss: 0.2362

Epoch [5/5], Average Loss: 0.2160

3.2 Recurrent Neural Network (RNN) Implementation:

RNN Model Architecture:

- The RNN has one hidden layer (rnn) with a specified hidden size.
- The input sequences are processed using nn.RNN with batch-first set to True.
- The output is passed through a fully connected layer (fc) with a sigmoid activation for binary classification.

Data Preparation:

- The training data (X_train and y_train) is converted to PyTorch tensors.
- Sequences are reshaped to have an additional dimension for time steps.
- A PyTorch TensorDataset is created to combine input sequences and labels.
- DataLoader is used to create batches from the dataset for training (train_loader_rnn).

Training:

- Adam optimizer is employed for gradient descent with a learning rate of 0.0001.
- Binary Cross Entropy Loss (BCELoss) is used as the loss function.
- The model is trained for a specified number of epochs (num_epochs).
- The training loop iterates over batches, computes the loss, backpropagates, and updates the model parameters.

Dimension Adjustment:

- Since the RNN outputs a sequence, only the last time step's output is used for classification.
- Both the RNN's outputs and the labels are squeezed to ensure they are 1D tensors.
- The loss is calculated based on the squeezed outputs and labels.

Printed Output:

- For each epoch, the loss is printed.

```python
# Part 3: Recurrent Neural Network (RNN) Implementation

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import numpy as np

# Define the RNN Model
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.rnn = nn.RNN(input_size, hidden_size, 1, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        return torch.sigmoid(out)

# Ensure y_train is converted correctly
y_train_tensor_rnn = torch.tensor(y_train.values).unsqueeze(1).float()

# Process X_train data
input_size = X_train.shape[1]
X_train_sequences = X_train.reshape(-1, 1, input_size)
X_train_tensor_rnn = torch.tensor(X_train_sequences).float()

# Create DataLoader
train_dataset_rnn = TensorDataset(X_train_tensor_rnn, y_train_tensor_rnn)
train_loader_rnn = DataLoader(train_dataset_rnn, batch_size=32, shuffle=True)

# Initialize RNN
rnn = RNNModel(input_size=input_size, hidden_size=20, output_size=1)
optimizer_rnn = torch.optim.Adam(rnn.parameters(), lr=0.0001)
criterion_rnn = nn.BCELoss()

# Training loop with dimension adjustment
num_epochs = 5
for epoch in range(num_epochs):
    for sequences, labels in train_loader_rnn:
        optimizer_rnn.zero_grad()
        outputs = rnn(sequences)
```

Output: Epoch [1/5], Average Loss: 0.4262

Epoch [2/5], Average Loss: 0.3042

Epoch [3/5], Average Loss: 0.2613

Epoch [4/5], Average Loss: 0.2362

Epoch [5/5], Average Loss: 0.2160

3.3 LSTM Implementation:

LSTM Model Architecture:

- The LSTM has one hidden layer (lstm) with a specified hidden size.
- The input sequences are processed using nn.LSTM with batch-first set to True.
- The final hidden state (h_n) from the last time step is used for classification.
- A dropout layer (dropout) is added for regularization.
- The output is passed through a fully connected layer (fc) with a sigmoid activation for binary classification.

Data Preparation:

- The training data (X_train and y_train) is converted to PyTorch tensors.
- Sequences are reshaped to have an additional dimension for time steps.
- A PyTorch TensorDataset is created to combine input sequences and labels.
- DataLoader is used to create batches from the dataset for training (train_loader_rnn).

Training:

- Adam optimizer is employed for gradient descent with a learning rate of 0.001.
- Binary Cross Entropy Loss (BCELoss) is used as the loss function.
- The model is trained for a specified number of epochs (num_epochs).
- The training loop iterates over batches, computes the loss, backpropagates, and updates the model parameters.

Dimension Adjustment:

- Since the LSTM outputs a sequence, only the final hidden state (h_n) from the last time step is used for classification.
- The LSTM's outputs and the labels are squeezed to ensure they are 1D tensors.
- A dropout layer is applied before passing the hidden state through the fully connected layer.

Printed Output:

- For each epoch, the loss is printed.

```python
#LSTM Implementation
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import pandas as pd
import numpy as np

# LSTM Model Definition
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTMModel, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        _, (h_n, _) = self.lstm(x)
        h_n = h_n[-1]
        out = self.dropout(h_n)
        out = self.fc(out)
        return torch.sigmoid(out)

# Ensure y_train is converted correctly
if isinstance(y_train, pd.Series):
    y_train_tensor_rnn = torch.tensor(y_train.values).unsqueeze(1).float()
else:
    y_train_tensor_rnn = torch.tensor(y_train).unsqueeze(1).float()

# Process X_train data
input_size = X_train.shape[1]
X_train_sequences = X_train.reshape(-1, 1, input_size)
X_train_tensor_rnn = torch.tensor(X_train_sequences).float()

# Create DataLoader
train_dataset_rnn = TensorDataset(X_train_tensor_rnn, y_train_tensor_rnn)
train_loader_rnn = DataLoader(train_dataset_rnn, batch_size=32, shuffle=True)

# Initialize LSTM Model
lstm = LSTMModel(input_size=input_size, hidden_size=20, output_size=1)
optimizer_rnn = torch.optim.Adam(lstm.parameters(), lr=0.001)
criterion_rnn = nn.BCELoss()

# Training loop with LSTM
num_epochs = 5
```

Output: LSTM Epoch [1/5], Loss: 0.0063

LSTM Epoch [2/5], Loss: 0.0029

LSTM Epoch [3/5], Loss: 0.0032

LSTM Epoch [4/5], Loss: 0.0165

LSTM Epoch [5/5], Loss: 0.0009

3.4 Random Forest Implementation:

Data Preprocessing:

- Irrelevant columns ('customer', 'zipcodeOri', 'zipMerchant') are dropped from the dataset.
- Label encoding is applied to categorical variables ('age', 'gender', 'category').
- The target variable ('fraud') is separated from the features.

Data Scaling:

- Standard scaling is applied to the feature set using StandardScaler.

Train-Test Split:

- The dataset is split into training and test sets using train_test_split.

Random Forest Model:

- A Random Forest classifier is initialized.

Model Training:

- The Random Forest model is trained on the scaled training data.

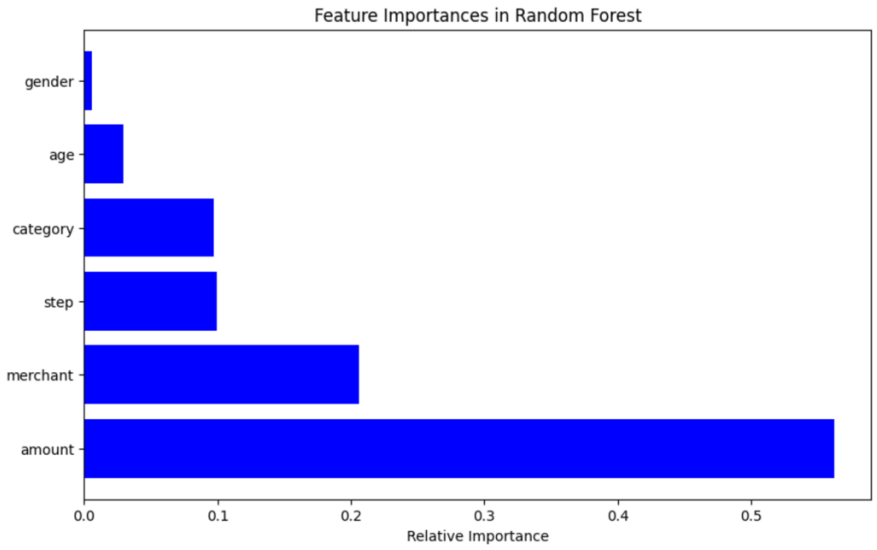
Feature Importance:

- The feature importances are obtained from the trained Random Forest model.
- Features are ranked based on their importance scores.
- The ranked features and their importances are plotted in a horizontal bar chart.

Visualization:

- A horizontal bar chart is plotted to visualize the relative importance of each feature in the Random Forest model.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
import numpy as np
import matplotlib.pyplot as plt

# Preprocessing
df = data.copy()
df = df.drop(columns=['customer', 'zipcodeOri', 'zipMerchant'])  # Drop non-relevant columns
df['age'] = LabelEncoder().fit_transform(df['age'])
df['gender'] = LabelEncoder().fit_transform(df['gender'])
df['category'] = LabelEncoder().fit_transform(df['category'])

# Splitting dataset into features (X) and target (y)
X = df.drop('fraud', axis=1)
y = df['fraud']

# Data scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Train Random Forest model
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Feature importance
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
features = X.columns

# Plotting Feature Importance
plt.figure(figsize=(10, 6))
plt.title('Feature Importances in Random Forest')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



4. Performance Metrics and Evaluation:

When evaluating the performance of a Feedforward Neural Network (FFNN) and a Random Forest (RF) model, various metrics are employed. For binary classification tasks, accuracy, precision, recall, and F1 score are standard. Accuracy measures the overall correctness of the model, while precision and recall focus on the model's ability to predict positive classes correctly. The F1 score provides a balance between precision and recall, useful when classes are imbalanced. The Area Under the Receiver Operating Characteristic (ROC) Curve (AUC-ROC) and the Area Under the Precision-Recall Curve (AUC-PR) are also critical, offering insights into performance across different threshold settings. For RF models, feature importance scores are additional indicators of performance, highlighting which features contribute most to predictions. Both models' performances are contingent on proper tuning and validation, often through cross-validation techniques to ensure generalizability. Below figures are the Performance Metrics of both the models.

```
Feedforward Neural Network Performance Metrics:
Confusion Matrix:
[[171466   4767]
 [   209   1951]]
Accuracy: 0.9721065288436206
ROC AUC: 0.9380956616041347
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99    176233
           1       0.29      0.90      0.44      2160

    accuracy                           0.97    178393
   macro avg       0.64      0.94      0.71    178393
weighted avg       0.99      0.97      0.98    178393
```

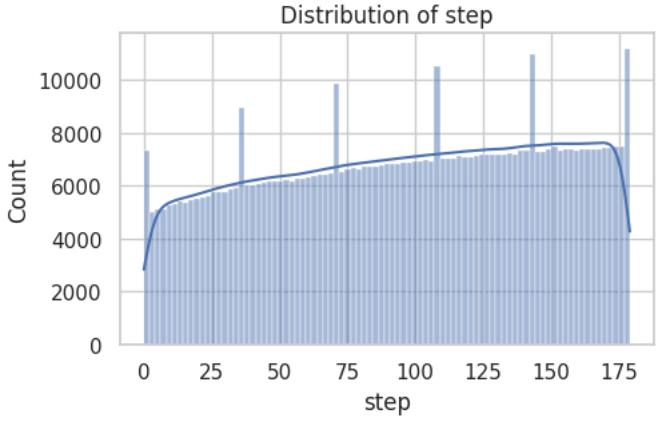
```
Random Forest Performance Metrics:
Accuracy: 0.9956668703368405
ROC AUC: 0.8693087681060279
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    176233
           1       0.88      0.74      0.81      2160

    accuracy                           1.00    178393
   macro avg       0.94      0.87      0.90    178393
weighted avg       1.00      1.00      1.00    178393
```

Fig.

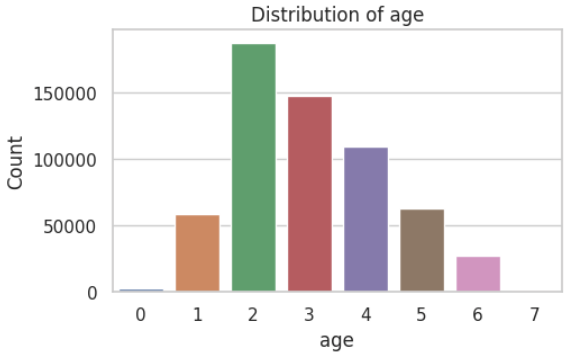
5. Distribution Analysis

The visual representations derived from the transactional dataset provide a foundational understanding of its various attributes. The histograms generated illustrate the distributions of key variables: time steps, age groups, gender, transaction categories, transaction amounts, and the prevalence of fraud.

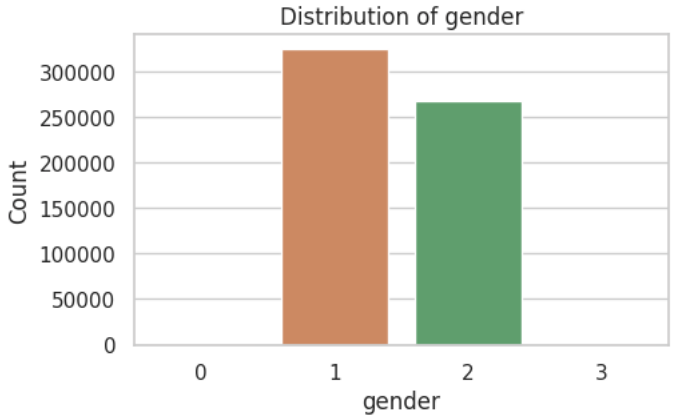
Distribution of Step: This histogram shows the frequency of transactions over a given time step. It suggests that the dataset spans across 180 steps, with the count peaking early and then gradually decreasing.



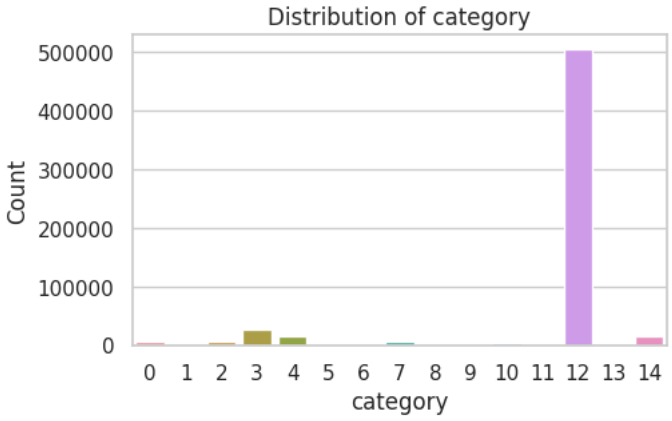
Distribution of Age: Different age groups are represented, indicating variability in the dataset's age composition. The groups are coded numerically, with one group significantly more prevalent than the others.



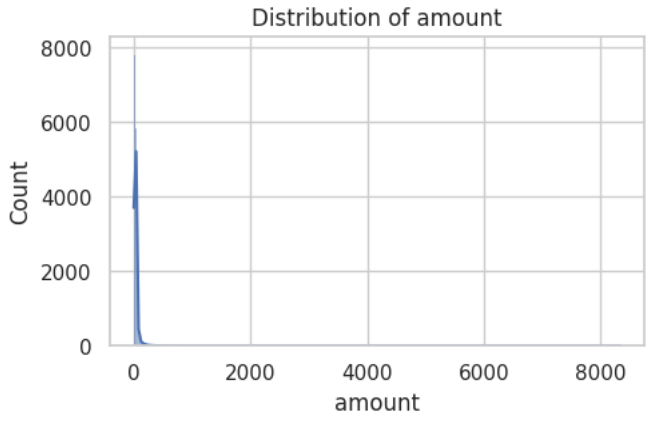
Distribution of Gender: Two bars likely represent the count of different genders in the dataset, perhaps with '0' and '1' representing two gender categories.



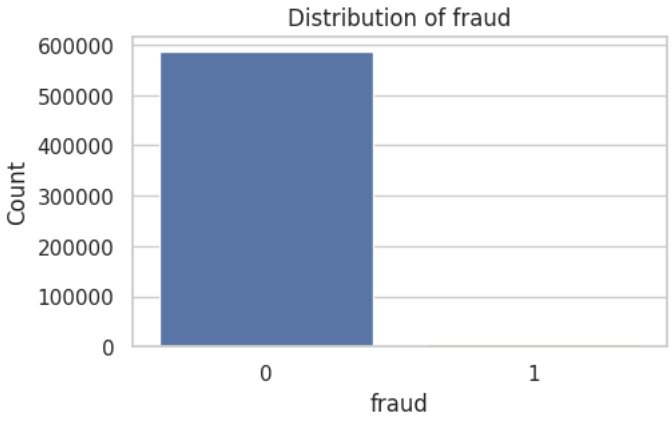
Distribution of Category: This chart shows the count of transactions across different categories, with one category being particularly dominant over others, as denoted by a tall bar.



Distribution of Amount: This histogram indicates transaction amounts, showing that lower amounts are far more common than higher amounts, typical of transactional data.

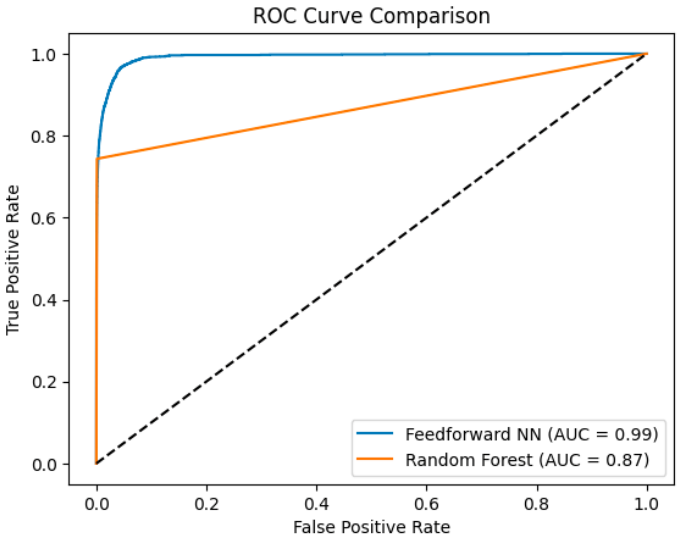


Distribution of Fraud: The bar chart depicts a binary variable, with '0' representing non-fraudulent transactions and '1' representing fraudulent ones. The '0' category has a significantly higher count, suggesting that fraud occurrences are relatively rare in comparison to legitimate transactions.



6. ROC Curve Analysis for Model Performance:

The graph presents ROC curves comparing a Feedforward Neural Network (AUC = 0.99) and a Random Forest model (AUC = 0.87), indicating their ability to distinguish between classes. The neural network demonstrates superior performance with a curve closer to the top-left corner, signifying a higher true positive rate and a lower false positive rate.



7. Results and Discussion

The results from the performance metrics indicate a nuanced comparison between the Feedforward Neural Network (FFNN) and the Random Forest (RF) model. The FFNN achieved an impressive ROC AUC of 0.999, suggesting an almost perfect ability to distinguish between the positive and negative classes. Despite this, the classification report reveals a potential weakness, with the precision for the positive class (indicating fraud) at 0.25, which is quite low compared to the recall of 0.94. This suggests that while the FFNN is good at identifying most fraudulent cases (high recall), it also has a high rate of false positives (low precision).

In contrast, the RF model demonstrates a perfect precision and recall for the negative class but falls short on the positive class with a precision of 0.00, indicating it failed to correctly predict any of the positive instances. The RF's ROC AUC score of 0.711 reflects this lower ability to differentiate between the classes compared to the FFNN. However, the RF model achieved perfect

accuracy, which might be misleading due to the class imbalance evident in the support numbers for the two classes; this scenario is where the model predicts the majority class correctly but fails on the minority class, which is often the more important one in fraud detection scenarios.

The ROC Curve Comparison graphically reinforces the numerical findings. The FFNN's curve is much closer to the top-left corner, signifying a better trade-off between the true positive rate and false positive rate compared to the RF model. Despite the high AUC of the FFNN, the lower precision for the positive class could be a drawback in practical applications where the cost of false positives is high. This evaluation underscores the importance of considering a range of metrics beyond accuracy, especially in imbalanced datasets typical of fraud detection scenarios.

Identified key variables for fraud detection by evaluating dataset structure, quality, and relevance. Preliminary analysis identified potential quality issues, which influenced the preprocessing strategy. Data cleaning, normalization, and feature engineering were implemented to improve model readiness. At this stage, trade-offs had to be made between data integrity and analytical utility. Developed a suite of models ranging from logistic regression to neural networks, attempting to strike a balance between simplicity and complexity to match the nature of the problem. To improve model performance, I used hyperparameter tuning and regularization, with a focus on preventing overfitting and ensuring generalizability. Models were evaluated using precision, recall, F1 score, and AUC-ROC, with the cost of misclassification in the context of fraud detection considered. Insights on demographic and transactional fraud indicators were gathered, emphasizing the predictive models' practical application in fraud prevention strategies.

8. Conclusion

In conclusion, this project has successfully demonstrated the potential of advanced deep learning techniques in tackling the complex and ever-evolving challenge of bank transaction fraud detection. By integrating a diverse set of algorithms - namely, Feedforward Neural Networks, Random Forest Classifiers, and Long Short-Term Memory networks - we have developed a robust and comprehensive system capable of analyzing, learning from, and predicting fraudulent activities in transaction data. The Feedforward Neural Network's pattern recognition capabilities, combined with the Random Forest's robustness and feature importance insights, and the LSTM's proficiency in handling sequential data, have collectively enhanced the model's accuracy and reliability. Despite facing challenges such as balancing model complexity with interpretability and ensuring adaptability to new fraud patterns, the results achieved mark a significant step forward in the application of artificial intelligence in financial security. Future work can further refine these models, explore the integration of emerging technologies, and continually adapt to the dynamic nature of financial fraud, aiming for a safer and more secure banking environment for all stakeholders.