A black background with white text

Description automatically generated with low confidence

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

**Data Science FINAL PROJECT REPORT**

**Project Title:**

Enhancing Fraud Detection: A Comparative Study of Ensemble Learning, Machine Learning, and Retrieval-Augmented Generation

**Student Name and SRN:**

Dhanuja Tharinda Memmenda Arachchi (21062872)

Supervisor: Vito Graffagnino

Date Submitted: 29/08/2024

Word Count: 7030

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at [Assessment Offences and Academic Misconduct](https://www.herts.ac.uk/__data/assets/pdf_file/0007/237625/AS14-Apx3-Academic-Misconduct-v17.0.pdf) and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the reportor code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Tharinda Arachchi

Student Name signature: A close-up of a signature

Description automatically generated

Student SRN number: 21062872

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

## ABSTRACT

***Financial fraud is a high and increasing threat to the world economy. Fraudulent transactions are very sophisticated nowadays and present a big problem for traditional detection methods. The key purpose of this study is to investigate the possibility of Retrieval-Augmented Generation in the enrichment of complex and emerging fraud pattern detection within financial transactions and customizing existing models for better capturing patterns and increasing accuracy of predictions. Up to now, traditional models such as random forests and neural networks have been applied to fraud detection. Very often, this kind of model fails to identify new or evolving fraud strategies since it is being trained on historical data. In contrast, RAG combines retrieval-based models and generative models in a way that caters to the dynamic integration of external knowledge for context-aware analysis hence may raise the bar for the detection of novel fraud patterns. This research deals with a comparison between the performances of RAG with the traditional models in terms of accuracy, adaptability, and effectiveness. The paper also designs a prototype RAG application, which uses a vector store and generative AI for the retrieval and analysis of related transactions based on a user query. Furthermore, investigating optimal embeddings, distance metrics and configurations for model is also intended through this study. Though the RAG has flexible and dynamic retrieval capacities, precision and accuracy are matters where the traditional models perform better. However, the study demonstrates that quality embedding and the treatment of false positives make the RAG models much more effective. But such limitations are the synthesis made up of datasets and poor access to state-of-the-art embedding techniques. Future work must now concentrate on optimized retrieval mechanisms of RAG, alternative embedding methods, and the unification of RAG with advanced machine learning models to develop a hybrid approach for providing further effective fraud detection.***

## GLOSSARY OF ABBREVATIONS

**BERT:** Bidirectional Encoder Representations from Transformers

**EDA:** Exploratory Data Analysis

**FAISS:** Facebook AI Similarity Search

**GPT:** Generative Pre-trained Transformer

**LLM:** Large Language Model

**NLP:** Natural Language Processing

**NN:** Neural Networks

**SMOTE:** Synthetic Minority Over-sampling Technique

**RAG:** Retrieval-Augmented Generation

**TF-IDF:** Term Frequency-Inverse Document Frequency

**UK:** United Kingdom

**XGBoost:** Extreme Gradient Boost

**Table of Contents**

[ABSTRACT 3](#_Toc175651986)

[GLOSSARY OF ABBREVATIONS 4](#_Toc175651987)

[INTRODUCTION 8](#_Toc175651988)

[Background 8](#_Toc175651989)

[Problem Statement 9](#_Toc175651990)

[Justification of the Study 9](#_Toc175651991)

[REVIEW OF LITERATURE 12](#_Toc175651992)

[Related work in the area 15](#_Toc175651993)

[METHODOLOGY 16](#_Toc175651994)

[Overview of Data 16](#_Toc175651995)

[Review of Columns 17](#_Toc175651996)

[Data Pre-processing, EDA and Feature Engineering 17](#_Toc175651997)

[Trained Algorithms to Analyse Performance 24](#_Toc175651998)

[Random Forest Classifier 24](#_Toc175651999)

[Model Configuration 24](#_Toc175652000)

[Model Customizations and Optimizations 24](#_Toc175652001)

[Neural Network 25](#_Toc175652002)

[Model Architecture 25](#_Toc175652003)

[Model Customizations and Optimizations 25](#_Toc175652004)

[Extreme Gradient Boosting (XGBoost) 25](#_Toc175652005)

[Model Architecture 25](#_Toc175652006)

[Retrieval Augmented Generation Model (RAG) 26](#_Toc175652007)

[Model Architecture 26](#_Toc175652008)

[Model customizations and optimizations 26](#_Toc175652009)

[Evaluation 27](#_Toc175652010)

[RESULTS AND ANALYSIS 29](#_Toc175652011)

[Experiment 1 29](#_Toc175652012)

[Analysis: 29](#_Toc175652013)

[Output: 29](#_Toc175652014)

[Experiment 2: 30](#_Toc175652015)

[Analysis 30](#_Toc175652016)

[Experiment 3: 31](#_Toc175652017)

[Analysis 31](#_Toc175652018)

[Experiment 4: 32](#_Toc175652019)

[Analysis: 32](#_Toc175652020)

[Experiment 5 32](#_Toc175652021)

[Experiment 6: 33](#_Toc175652022)

[Analysis: 33](#_Toc175652023)

[Experiment 7: 34](#_Toc175652024)

[Analysis 34](#_Toc175652025)

[Experiment 8: 35](#_Toc175652026)

[Analysis: 35](#_Toc175652027)

[Experiment 9: 35](#_Toc175652028)

[CONCLUSION 38](#_Toc175652029)

[LIMITATIONS 38](#_Toc175652030)

[FUTURE WORK 38](#_Toc175652031)

[REFERENCES 39](#_Toc175652032)

[APPENDIX 42](#_Toc175652033)

**List of Figures**

[Figure 1 Overview of Retrieval-Augmented Generation (AWS, 2024) 14](#_Toc175652154)

[Figure 2 Overview structure of raw dataset 16](#_Toc175652155)

[Figure 3 Datatypes of raw dataset 16](#_Toc175652156)

[Figure 4 Distribution of some features of dataset 18](#_Toc175652157)

[Figure 5 Distribution of fraud and non-fraud data across transaction amount 19](#_Toc175652158)

[Figure 6 Percentage of fraud to non-fraud transactions by customer age 19](#_Toc175652159)

[Figure 7 Distribution of features of fraud transactions 21](#_Toc175652160)

[Figure 8 Relationship of customer age, amount and fraud or not 21](#_Toc175652161)

[Figure 9 Co-relationship heatmap 22](#_Toc175652162)

[Figure 10 Imbalanced dataset 23](#_Toc175652163)

[Figure 11 SMOTE Oversampled dataset 23](#_Toc175652164)

[Figure 12 Text Embeddings 27](#_Toc175652165)

[Figure 13Basic model with Imbalanced dataset 29](#_Toc175652166)

[Figure 14 Customized model, SMOTE dataset 29](#_Toc175652167)

[Figure 15 Basic NN on imbalanced dataset 30](#_Toc175652168)

[Figure 16 Customized NN on imbalanced dataset 30](#_Toc175652169)

[Figure 17 Loss and Accuracy plot 30](#_Toc175652170)

[Figure 18 Customized NN model on imbalanced dataset 31](#_Toc175652171)

[Figure 19 Customized NN model on SMOTE dataset 31](#_Toc175652172)

[Figure 20 XGBoost on imbalanced data 32](#_Toc175652173)

[Figure 21 XGBoost on SMOTE data 32](#_Toc175652174)

[Figure 22 Shape of TF-IDF 32](#_Toc175652175)

[Figure 23 Shape of BERT 32](#_Toc175652176)

[Figure 24 Embedding cosine vs Euclidean distance 33](#_Toc175652177)

[Figure 25 Cosine approach 33](#_Toc175652178)

[Figure 26 Euclidean approach 33](#_Toc175652179)

[Figure 27 BERT Embeddings 34](#_Toc175652180)

[Figure 28 OPEN AI Embeddings 34](#_Toc175652181)

[Figure 29 Evaluation of similarity search 35](#_Toc175652182)

[Figure 30 Evaluation of RAG model (full text) 36](#_Toc175652183)

[Figure 31 RAG Prediction (full text) 36](#_Toc175652184)

[Figure 32 2D representation of embeddings 36](#_Toc175652185)

[Figure 33 Generative response from RAG model 37](#_Toc175652186)

## INTRODUCTION

# Background

**Understanding Financial Fraud and its Impact**

In today’s fast-paced world, financial transactions are happening continuously, every millisecond. Most of the financial institutions are moving into digitalization to supply the emerging demand arising from consumers globally. With the significant growth of the industry and improved digitalization, there has been a corresponding increase in scammers and fraudsters. Consequently, fraudulent activities have become a major concern for businesses, banks and consumers from around the world as such activities are becoming both more sophisticated and more frequent.

Financial fraud can happen in many forms. Credit card frauds, insurance frauds, identity theft and money laundering are some of the most prominent ways of fraud in the present society. The results from such transactions influence the individual, business success, and sometimes the stability of the entire financial system. The global cost of financial fraud amounts to trillions of dollars every year, which in turn affects consumer trust, increases the operational cost for businesses, and undermines economic growth (ACFE, 2022)

**Traditional Approaches to Financial Fraud Detection**

Recognizing prevention and detection of financial frauds can have enormous impacts. Several years ML methods and Ensemble learning have been a great deal in detecting fraudulent activities by analysing patterns of activity arising from large and highly complex datasets. Widely used and most prominent detection models are Random Forest, Neural Networks and XGBoost.

**Limitations of Traditional Models**

Despite the popularity of these traditional fraud detection models, there are concerns about their effectiveness in identifying new and evolving patterns of fraudulent transactions. These models highly rely on the historical data, rather than identifying evolving patterns of fraudulent transactions. This highlights a raising question about the abilities of these traditional models in detecting emerging fraud trends.

**Introduction to Retrieval-Augmented Generation (RAG)**

In recent years, advancements in the artificial intelligence, particularly in the field of natural language processing (NLP) have introduced few possibilities. It has shown the potential to overcome thus limitations. One such advancement is Retrieval Augmented Generation (RAG), a novel approach which combines retrieval-based models with generative models. RAG has shown promising results in different domains such like question answering and text generation. RAG leverages the large corpus of relevant information during inference. In RAG, a retrieval mechanism is employed to gather relevant documents or data segments from an external corpus. Then this information is fed into a generative model (like GPT) to produce a final output (Lewis et al., 2020). This hybrid approach allows the model to generate informed responses based on both the input data and the retrieved information, leading to more accurate and contextually rich predictions.

# Problem Statement

Financial fraud continues to be a significant threat to the global economy. While fraudulent transactions are becoming increasingly sophisticated and more frequent, the ability to detect and prevent of thus transactions is becoming further challenging. Traditional models like Random Forest Classification, Neural Network models and even XGBoost are widely equipped around the world in detection of fraudulent activities. However, these models fall short in identifying new or evolving fraud patterns due to their reliance of historical data and incapability of retrieving external, contextually relevant information (Breiman, 2001; LeCun, Bengio & Hinton, 2015; Chen & Guestrin, 2016).

As fraudsters continuously refine their tactics, there is a critical demand for a more advanced model that can dynamically integrate external knowledge and provide context aware analysis. Retrieval Augmented Generation (RAG) provides a promising solution by combining both retrieval-based models with generative models. This enhances the detection of complex and emerging fraud patterns. However, the application of RAG in the context of financial fraud detection remains underexplored.

This study aims to address the gap by investigating the potential of RAG in financial fraud detection systems. Also, investigate the effectiveness of the traditional models by customizing and fine tuning these models to retrieve higher performance.

# Justification of the Study

Due to prevailing economic issues around the world, financial fraud has increased significantly throughout the world. According to the UK Finance annual fraud report 2024, losses due to fraud in the UK reached £1.17 billion in 2023 (UK Finance, 2024). This implies the current systems in place are not successful enough, demanding a more advanced model which can detect novel and complex patterns of fraud.

Retrieval Augmented Generation (RAG) has shown promising results in the realm of Natural Language Processing and widely used in creating question answering systems. In this research I tried the potential of leveraging the retrieval and generation capabilities of a RAG model and incorporate It into identifying fraudulent transactions.

In the realm of data science and machine learning, usage of embeddings, vector stores and querying are underexplored in the context of non-textual datasets. Hence, there was an urge to utilize RAG model along with embeddings, vector stores, indexing and querying in a structured transactional dataset, in this case the financial fraud transaction dataset. I took this opportunity to investigate this un-explored area.

In this study, I have utilized traditional models like Random Forest, Neural Network and XGBoost to set up a benchmark to compare the performance of RAG integrated model. Also, I have tried multiple measuring metrics when it comes to finding the nearest matching transaction based on the similarity score. Additionally, creating a protype model of generating responses based on the user query prompt is also considered, as the need for such model is helpful in detecting fraudulent transactions more effectively.

Finally, I have extensively customized and fine-tuned traditional models to investigate how these techniques can enhance model performance.

Research Questions

This research focus on answering below questions.

1. What are the capabilities of Retrieval-Augmented Generation (RAG) in identifying fraudulent transactions?
2. How does the effectiveness of Retrieval-Augmented Generation in detecting fraudulent transactions compared to traditional machine learning methods?
3. How can a prototype model be developed using generative AI to retrieve related transactions based on user queries?

Aims and Objectives

The primary aim of this research is to explore and evaluate the effectiveness of Retrieval Augmented Generation (RAG) in detection of fraudulent transactions within financial dataset and creating a prototype RAG application.

**Objectives**

**Assess the Capabilities of RAG:** Investigate how Retrieval-Augmented Generation (RAG) can be utilized to identify and analyse fraudulent transactions  
**Compare RAG with Traditional Methods:** Evaluate the performance of RAG in detecting fraudulent transactions compared to established machine learning techniques, such as Random Forest, Neural Networks, and XGBoost, in terms of accuracy, adaptability, and effectiveness.  
**Develop a Prototype Model:** Create and implement a prototype RAG application that leverages vector stores and generative AI to retrieve and analyse related transactions based on user queries, demonstrating the practical application of RAG in fraud detection.  
**Customize Traditional Models:** Customizing traditional models to capture more insight and increase the accuracy of predictions  
**Investigate how class imbalance effect model performance:** Conducting experiments with imbalanced dataset and oversampled dataset and see how it effects model performance

Ethical Considerations

This adheres to some important principles that guide ethics: the dataset is anonymous and genuinely ethically gathered, and both code and report are an original work. Also, it is in accordance with privacy and data security norms with respect to human life, the law, environment, and the general welfare. It underlines precision and rigour by citing all relevant articles and journals in full, which therefore demonstrates commitment to exact and reliable research. This further serves as a good example of leadership and communication through the use of ethical research techniques and presentation of results in a responsible and transparent way.

## REVIEW OF LITERATURE

**Financial Fraud Detection**

Financial fraud is a persistent issue vastly impacting not only the individuals and financial institutions but also the global economy itself. Early detection of fraudulent activities is crucial due to the growing fraud schemes. Various machine learning models have been developed and deployed to address this challenge.

**Traditional models for Fraud Detection**

Traditional models for fraud detection primarily rely on rule-based system and statistical methods. These systems utilize pre-defined set of rules identified by the subject experts to flag suspicious activities. These rules are more often derived from historical data and domain knowledge (Phua, Lee, Smith & Gayler, 2010). Statistical methods, including logistic regression, is also widely used to model the probability of fraud occurrences (Bolton & Hand, 2002). While this model provides a foundation for fraud detection, these models mostly struggle with scalability and adaptability to new and evolving patterns.

**Random Forest**

Random Forest is a robust ensemble learning method widely known for high accuracy in fraud detection models. It works during training by building multiple decision trees and outputting the mode of the classes in case of classification tasks. In fact, in one study, Abdallah, Maarof, and Zainal (2016) prove that Random Forest is superior compared to other algorithms such as Decision Trees and Logistic Regression in cases of detecting financial fraud because of its strong features in dealing with big data and minimizing overfitting.

**Neural Network**

Neural network, especially deep learning models have shown promising results in the context of fraud detection. Neural network model consists of multiple layers which process input data through weighted connections (Goodfellow, Bengio & Courville, 2016). Also, these models can capture complex patterns and relationship of data. Hence, neural network models can identify subtle and sophisticated fraud schemes.

**XGBoost**

XGBoost is an extension of gradient-boosted decision trees. This has emerged as a strong tool in fraud detection due to its efficiency and performance. These models are precise for handling large-scale datasets and providing better generalization. A study by Pourhabibi et al. (2020) has emphasized the superior accuracy of XGBoost among other tested models in fraud detection tasks. This ensures that the model is capable of placing more focus on hard-to-classify cases by iteratively adjusting weights.

**Class Imbalance and SMOTE**

One of the significant challenges when dealing with financial fraud analysis dataset is the class imbalance problem. The ratio of majority vs minority class is highly different. These imbalances can lead to biased models that Favors the major class resulting in poor detection of fraudulent transactions (Jing et al., 2018).

The Synthetic Minority Over-Sampling Technique (SMOTE) is a widely used method to tackle this issue. SMOTE generates synthetic data for the minority class by interpolating between existing minority instances, thus balancing the dataset (Chawla et al., 2002). In the context of fraud detection SMOTE has shown a significant improvement in the performance of models by providing a more balanced representation of classes. Balancing the weight among the classes is critical for effective model training (Dal Pozzolo et al., 2015).

**Embedding, Vector Stores and Similarity Search**

Embeddings convert categorical variables into continuous vector representation, capturing the semantic relationship among them. This approach is undoubtedly useful in financial data where categorical features like ‘transaction type’ , ‘transaction behaviour patterns’ are prevalent (Mikolov et al., 2013).

Vector stores allow for efficient storage and retrieval of these embeddings. These vector stores also allow similarity search on these embeddings. Similarity search is a technique used to find instances in the data according to the query instance provided. In the context of financial fraud this basically mean detecting transactions which are similar to known fraudulent transactions (Johnson, Douze & Jégou, 2019).

**Retrieval Augmented Generation (RAG)**

RAG is a relatively recent paradigm in natural language processing (NLP). The primary goal of RAG is to enhance the quality and relevance of generative text by integrating a retrieval mechanism and generative model. Unlike traditional text generation models purely rely on pre-trained data, RAG retrieve relevant information to inform their output. In below sections I will talk about a comprehensive overview of RAG, discussing key research developments and their findings, strengths and limitations.

1. **The concept of Retrieval Augmented Generation**

RAG combines retrieval-based methods with generative models to produce more contextually accurate and informative text. Lewis et al. (2020) introduced the RAG model, which integrates a dense retrieval mechanism with a generative model based on BART (Bidirectional and Auto-Regressive Transformers). The retriever searches an external data source to find relevant passages. Then generator uses the findings to generate coherent and context rich response. This dual approach allows RAG to stand out from traditional generative models, which suffers from limited factual accuracy (Lewis et al., 2020).

1. **Mechanism of RAG**

The RAG model consists of two main components: the retriever and the generator. The retriever uses dense passage retriever (DPR) or other embedding based method efficiently search through large data source. Karpukhin et al. (2020) explored the use of dual encoders for dense retrieval. It demonstrated high accuracy in selecting relevant documents by encoding both queries and documents into a shared vector space. The generator leverages these retrieved documents to produce the final output. Generator models such as BERT or GPT to generate coherent and contextually enriched responses (Karpukhin et al., 2020)

A diagram of a computer process

Description automatically generated

Figure 1 Overview of Retrieval-Augmented Generation (AWS, 2024)

1. **Advancements of Retrieval Mechanisms**

Recent research has focused on improving retrieval mechanisms to enhance RAG performance. Qu et al. (2021) proposed an improved retrieval strategy that combines both sparse and dense retrieval techniques to better handle diverse types of queries. This hybrid approach allows for more nuanced retrieval. Also, it balances precision and recall.

1. **Enhancements in Generative process**

Shuster et al. (2021) explored various ways to incorporate retrieved knowledge into the generation processes. They have tried direct concatenation of retrieved documents and query-focused re-ranking to prioritize more relevant passages.

1. **Applications of RAG in NLP Tasks**

RAG has been successfully applied in the context of NLP and showcased promising results. In dialogue systems, RAG has improved response relevance and coherence by retrieving pertinent background information to inform the conversation (Shuster et al., 2021)

By thoroughly observing these facts and to fill in the gap of RAG usage in more structured datasets, I have taken the step forward to investigate how RAG can be utilized in detecting frauds in a transactional dataset.

# Related work in the area

The use of Retrieval-Augmented Generation (RAG) in fraud detection is a new and emerging area of research. RAG is widely used in question answering models. Rag retrieves contextual information from a large dataset and uses it to guide predictions, improving detection accuracy.

Lewis et al. (2020) introduced RAG and showcased its effectiveness in tasks which requires extensive knowledge. They found that RAG outperforms traditional transformer models by incorporating a retrieval step which brings external knowledge. While they primary has focused on Natural Language Processing tasks, the principals of RAG too can be applied to structure dataset such like fraud detection. It can retrieve past transaction fraud patterns or similar cases which can then inform the detection model.

RAG can identify complex fraud schemes that traditional models might miss. For example, Khattak et al. (2022) used RAG in a legal question-answering system. They have shown how RAG could generate precise answers by retrieving and integrating relevant cases. Similarly, this can be applied in the domain of fraud detection as well. Integrating RAG into this domain could lead to more nuanced and informed decision making, especially in complex and more sophisticated cases where traditional models might fail.

While the direct application of RAG in the domain of fraud detection is still in early stages, there are lot of potential benefits.

## METHODOLOGY

# Overview of Data

The dataset used in this is obtained from Kaggle.com. According to Kaggle (n.d.), the dataset is collected from a Spanish bank from a simulation run for 6 months. Dataset has around 600,000 transactions across more than 4000 users. Furthermore, the dataset is highly imbalanced as out of all these transactions only 7200 is categorized as fraud which is 1 % of the data.

A screenshot of a computer

Description automatically generated

Figure 2 Overview structure of raw dataset

A screenshot of a phone

Description automatically generated

Figure 3 Datatypes of raw dataset

# Review of Columns

Original dataset consists of below columns.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Keep?** | **Explanation** |
| Step | Time-step identifier | Yes | Values ranging from 0-180. Assuming a time-step represents a one transactional day. |
| customer | Customer identifier | Yes | Categorical feature. Random unique number to identify customer, ensuring the anonymity |
| age | Age group of customers | Yes | Values ranging from 0-6. |
| gender | Gender of the customer | Yes | Unique values are ‘M’, ‘F’ and ‘U’ |
| zipCodeOri | Zip code of the originated location | No | Removing as the value is static throughout the dataset |
| Merchant | Identification of the merchant | Yes | Categorical feature. Random unique number to identify customer, ensuring the anonymity |
| zipMerchant | Zip code of the merchant location | No | Removing as the value is static throughout the dataset |
| Category | Transaction type | Yes | Categorical feature. Specify for which category the transaction is originated |
| Amount | Transaction amount | Yes | Numerical Feature. |
| Fraud | is Fraud yes or no | Yes | Target variable. If fraud=0 means legitimate transaction. Fraud =1 means transaction is identified as Fraud. |

Table 1 Overview of columns

Before diving into data processing and methodology, I will briefly explain the code structure and integration used in this project. The code is written in Python. I used Google Colab framework, with GitHub used for version control. To streamline the workflow, I introduced a novel approach by directly integrating GitHub with Google Colab. This allowed for seamless communication with GitHub without any hassle. Additionally, I adopted a modular code structure using classes, instances, and functions to enhance readability and minimize redundancy.

# Data Pre-processing, EDA and Feature Engineering

The data set was pre-processed to be informative following many other steps, such as using label encoding to convert categorical data into numerical representations, removing redundant or duplicate entries, handling missing or unknown values, and removing unwanted characters from the data set. Such steps are prerequisites to ensure that the dataset becomes more clean and ready for modelling.

For categorical encoding in this project, I used label encoding. Label encoding is a technique which used to convert categorical data into numerical format. Each unique category value is assigned a unique integer by allowing machine learning algorithms to process the data more effectively.  
Then, I removed ‘zipCodeOri’ and ‘zipMerchant’ columns from the dataset as it contains one unique value across the dataset. Also, I have removed the un-necessary characters in the values before parsing to modelling.

Finally, I have removed unknown values especially in customer age column which happened to be ‘U’ as value.

Next, the dataset was analysed by plotting it on a variety of attributes for an Exploratory Data Analysis (EDA).

|  |  |
| --- | --- |
| A diagram of a customer age  Description automatically generated | A graph with numbers and text  Description automatically generated with medium confidence |
|  |
| A graph with blue lines  Description automatically generated |

Figure 4 Distribution of some features of dataset

A diagram of red and blue dots

Description automatically generated

Figure 5 Distribution of fraud and non-fraud data across transaction amount

By observing these plots, I have got a brief understanding of the data set. It is evident that majority of the recorded customers are females and are in age category of 2 and 3. Also the merchant category used in this data set is widely used for transportation. Transaction amount is varying from cents to more than 8000.

A graph of blue and red bars

Description automatically generated

Figure 6 Percentage of fraud to non-fraud transactions by customer age

I also plotted various graphs to explore the relationships among different variables and their connection to the target variable (isFraud).

Above plot depicts that fraud transactions is common in each age group while majority recorded is in ‘0’ age group.

By closely observing the plots created against fraud dataset, high number of frauds are observed in customers of 2,3 and 4 age group. Least number of frauds is observed in the 0,6-age group. Comparing the gender of customers marked as fraud, female customers are almost double compared to male. Also, highly flagged merchant category is sports and toys, and it is followed by health. Transactions done in fashion and in tech seems to be more legitimate.

|  |  |
| --- | --- |
| A graph with blue lines  Description automatically generated | A graph of a number of numbers  Description automatically generated with medium confidence |
|  |
| A graph with blue lines  Description automatically generated |

Figure 7 Distribution of features of fraud transactions

Below graphs and plots shows the co-relationships among each feature in the dataset corresponding to target variable.

However, upon reviewing the relationship plots, it is evident that most variables do not significantly influence the target variable. Only amount shows a high correlation.

A group of graphs with numbers

Description automatically generated with medium confidence

Figure 8 Relationship of customer age, amount and fraud or not

A screenshot of a graph

Description automatically generated

Figure 9 Co-relationship heatmap

Feature engineering is the engineering new features from dataset to improve the performance of the machine learning model. According to Guyon and Elisseeff (2003), feature engineering includes the identification and selection of variables that are most relevant for the model, which can significantly impact its accuracy and efficiency.

I have engineered below additional features to increase the model accuracy.

|  |  |
| --- | --- |
| **Feature** | **Description** |
| encoded\_gender | Encoded from customer gender column. Used label encoding |
| encoded\_category | Encoded from merchant category column. Used label encoding |
| IslargeTransaction | This feature is created to categorize and flag large transactions. I have used a value of 5000 as threshold (changeable). |
| countForCustomerSameTime | This is counter which counts number of transactions done by same customer in a same time step |
| avgTransactionAmount | Calculates and store average transaction amount for each customer |
| transaction\_behavior | Stores a text prompt of behavior of transaction. This is Used for text embeddings. |

Table 2 Feature engineering

One of the major challenges that this dataset has is that it is extremely imbalanced. In the context of fraud detection, this would mean a very low number of fraudulent transactions, or positive cases, in comparison with the number of legitimate ones, which are the negative cases. This can result in a set of issues related to machine learning models. Model performing more biased towards the majority class is a prominent issue.

A blue rectangular bar with red and white text

Description automatically generated

Figure 10 Imbalanced dataset

To mitigate the issue of data imbalance in the dataset, it was essential to implement such strategies that could enhance the model’s ability to accurately predict both the majority and minority class. Out of the various methods available to address this challenge, I chose the Synthetic Minority Over-Sampling Technique (SMOTE).

A graph with red and blue squares

Description automatically generated

Figure 11 SMOTE Oversampled dataset

# Trained Algorithms to Analyse Performance

Before performing the data on RAG model, I have tried the dataset with traditional models to set up a benchmark performance. I have further incorporated customized model optimizations, hype parameter tuning to these traditional models expecting to obtain a better performance accuracy.

### Random Forest Classifier

The Random Forest Classifier is one of the most robust ensemble learning methods. It is common in tasks such as classifications, fraud detections, and so on. During training numerous numbers of decision trees are created and then aggregated the output to finalize a decision. Each decision tree in the forest is built by recursively splitting the data based on feature values. Final prediction is made based on the majority vote from all trees (Liaw and Wiener, 2002).

### Model Configuration

|  |  |  |
| --- | --- | --- |
| Configuration | Value | Description |
| n\_estimators | 100 | Number of trees in the forest. More the better, however could impact computational cost. |
| max\_depth | None | Maximum death of the tree. I have configured to expand until all leaves are pure |
| min\_samples\_split | 2 | Minimum number of samples required to split an internal node |
| min\_samples\_leaf | 1 | Minimum number of samples at leaf level |

Table 3 Model configuration for Random Forest

### Model Customizations and Optimizations

1. Handling data imbalance problem by SMOTE oversampling and adjusting the bias towards model.
2. Custom hyper parameters usage to provide a more balance between computational efficiency and performance (i.e: n\_estimators, max\_depth, min\_samples\_split and min\_samples\_leaf)
3. Label encoding done to utilize categorical features in model predictions.
4. Standard scaling to standardize features to have zero mean and unit variance.
5. Feature importance calculation.

# Neural Network

Basically, NN models are recognizing patterns and making a decision based on input data as similar to how humans do. These models have multiple layers connected with neurons, and every connection is associated with a weight which is adjusted while training. Passing the data through the created layers makes the network learn. That is how it models complicated relationships and performs classification and regression tasks efficiently.

### Model Architecture

|  |  |
| --- | --- |
| **Configuration** | **Description** |
| Input layer | Based on the input dataset shape. |
| Dense layer | 1st layer – 64 neurons, Activation - ReLU |
| Dropout layer | Dropout rate - 0.5 |
| 2nd dense layer | 32 neurons, Activation - ReLU |
| Output layer | Single neuron, Activation - Sigmoid |

Model was trained for 10 epochs in 32 batches. Trains and test set ratio was 75% to 25%.

Model uses ‘Adam’ optimizer to adjust the weights of the neurons based on the difference between predicted and actual outcomes. Binary cross-entropy is used as the loss function to measure the error. Additionally, model is customized using early stopping mechanism to halt training if the model’s performance on validation data stops improving, resulting in preventing overfitting.

### Model Customizations and Optimizations

1. Early stopping to prevent overfitting.
2. Adding dropout layers to regularize 50% of neurons to 0 during each training iteration. Expecting model to not rely too much on specific neurons, ultimately reduce overfitting
3. Feature standardization.
4. Label encoding.

# Extreme Gradient Boosting (XGBoost)

XGBoost is an optimized distributed gradient boosting library structured to be highly efficient in terms of running speed and model execution performance (Chen & Guestrin, 2016). Key novelties of XGBoost over other gradient boosting techniques include advanced computational techniques, such as regularization, parallel computing, and optimization.

### Model Architecture

The XGBoost classifier is used for its efficiency and effectiveness in handling classification tasks. It builds multiple decision trees to make predictions and aggregates their results to provide a final output. XGBoost is known for its high performance in large datasets and its ability to handle imbalanced classes, which is useful for detecting rare fraudulent transactions.

# Retrieval Augmented Generation Model (RAG)

Retrieval Augmented Generation model with retriever and generator model is widely used in NLP tasks. Specifically in the domain of question answering and chat bot creations. Utilizing and leveraging the capability of RAG in terms of finding and flagging fraudulent transactions is examined.

### Model Architecture

In this study, BERT (Bidirectional Encoder Representations from Transformers) is utilized for tokenization and generating text embeddings. BERT enables effective capture of contextual information from text (Devlin et al., 2019). For managing and retrieving vector representations, FAISS (Facebook AI Similarity Search) is employed. FAISS offers efficient similarity search and clustering of dense vectors (Johnson et al., 2017). For the generative component, GPT-3.5 is used, a state-of-the-art model that leverages a transformer-based architecture to produce high-quality and coherent text based on the retrieved information (Brown et al., 2020).   
Also, a different embeddings method like OPEN AI, which is external API equipped to compare the effectiveness of text embeddings.

Model customizations and optimizations

1. Usage of various embedding methods like BERT, OPEN AI, TF-IDF and identified optimal for the model, which provides meaningful, dense representation of text data.
2. FAISS vector store for efficient search, and capability to handle large datasets.
3. Custom scoring with high priority terms to give more importance in the similarity search for specific keywords.
4. GPT 3.5 for response generation for informative response generation.

Multiple experiments were conducted to identify which embeddings to use. I will discuss this further in next chapter along with the findings. As the first step, features were converted to embeddings and stored in the FAISS index.

Then FAISS index is read/loaded and used the query embeddings to identify and flag anomalous transactions using Euclidean distance and cosine dissimilarity.

Also, I implemented a similarity search method for retrieval and developed a prototype generative model to showcase how RAG can be applied to fraud detection. This involves necessary pre-processing, text embeddings, vectorization, similarity search with input query prompt and context rich response generation to identify whether the transaction is fraudulent or not.

A screenshot of a computer

Description automatically generated

Figure 12 Text Embeddings

As per the above image, additional text column is describing the behaviour of each transaction. Then these texts were converted to embeddings using BERT tokenizers. Next, these were added to FAISS vector store/index.

When a user inputs a query, the system searches the vector store using similarity measures finding and ranking based on the distance metric and use the results to determine if a transaction is likely fraudulent or not. Then provides contextual information about the transaction with the usage of generative models.  
Similarity search is an approach to find the most nearby neighbour in simple terms. This typically means that finding the vector in a vector space that are close to the query vector based on any distance metric. Widely used distance metrics are Euclidean and Cosine similarity. In this approach, I have utilized both these metrics.

# Evaluation

All the models were tested based on a couple of key metrics that gives an overview of how well each model is performing. Then, calculate the Accuracy Score, which measures what proportion of predictions a model gets correct. Also, generate a Classification Report, which gives detailed reports on how every model performs in terms of precision, recall, and F1-score.

**Precision** measures how well the model is at avoiding false positives.

Equation 1 Precision

**Recall** shows the actual positives that the model has correctly classified.

Equation 2 Recall

**F1-score** provides a well-balanced view of the model performance.

Equation 3 F1 Score

Also, a confusion matrix was created to visualize the performance of model. This matrix shows the counts of true positive, true negative, false positive, and false negative predictions. This helps to understand where the model is making errors as well as where it is performing well.

To evaluate similarity search of the retriever model, below metrics were used.

**Average Precision (AP)** The average precision is defined as the average of precisions at every relevant position in the ranked list. AP gives a summary of how well the model performs across all relevant items and not just the top-K results. It combines precision and recall in one number, thus indicating the ranking performance of the model.

**Normalized Discounted Cumulative Gain (NDCG):** NDCG measures the usefulness of the results based on the position in the ranked list by giving more weight to relevant items that appear earlier in the list. This metric helps in understanding how well the model orders/sorts the relevant items, which is critical for applications where the order of results impacts decision-making.

## RESULTS AND ANALYSIS

Experiment 1**:** Performing Random Forest Classifier on imbalance dataset vs SMOTE oversampled dataset and optimized model

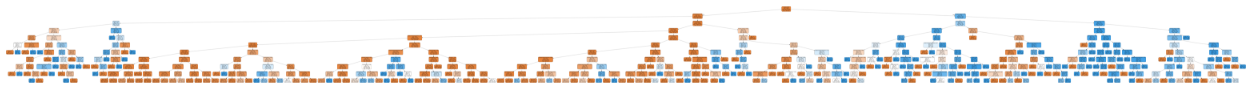
|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated  Figure 13Basic model with Imbalanced dataset | A screenshot of a computer  Description automatically generated  Figure 14 Customized model, SMOTE dataset |

### Analysis:

Model 2 performs better than model 1 in precision and recall toward fraud detection, with slightly greater accuracy and generally better metrics. This is because Model 2 was fitted with SMOTE for dataset balancing and customized configurations. Model 1, although effective, is constrained with this imbalanced dataset and hence gives lower precision and recall values for the minority class.

### Output:

Below is the generated classification tree. Original Tree can be found in GitHub repository under reports folder.



Experiment 2: Basic Neural Network model vs Customized Neural Network Model on imbalanced dataset

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated  Figure 15 Basic NN on imbalanced dataset | A screenshot of a computer  Description automatically generated  Figure 16 Customized NN on imbalanced dataset |

### Analysis

Model 2 significantly outperforms Model 1. Model 1 achieves a very high accuracy of 98.84%, but it misses fraud transactions, and hence the recall for fraud detection is 0%. In contrast, Model 2 uses customized features like dropout layers and early stopping to get an accuracy of 99.50% and significantly improved fraudulent transaction detection with a precision of 0.94 and recall of 0.60. This makes Model 2 more effective at classifying real and fraudulent transactions and better handling class imbalance.

A graph of different colored lines

Description automatically generated with medium confidence

Figure 17 Loss and Accuracy plot

As per the training and validation loss, model is learning effectively and generalizing well to the validation data. The model’s performance seems stable and improving without overfitting.

Experiment 3: Customized NN model on imbalanced dataset vs SMOTE oversampled dataset

|  |  |
| --- | --- |
| Figure 18 Customized NN model on imbalanced dataset | Figure 19 Customized NN model on SMOTE dataset |

### Analysis

The first model attains a high level of accuracy, 99.50%. It can identify genuine transactions with both precision and recall of 1.0. It really suffers on fraudulent transactions but leads to a recall of 0.60 for such samples. In this way, it performs only well on the minority class, though having a high accuracy.

Just a little lower on the accuracy, Model 2 gives an impression of 98.59% with wholly significant improvement on the fraudulent transaction that belonged to a high precision of 0.98 and 0.99 in recall; it represents a balanced performance across the classes. Model 2 using oversampling thus handles the class imbalance better to detect both genuine and fraudulent transactions better.

Experiment 4: XGBoost on imbalanced vs SMOTE oversample dataset

|  |  |
| --- | --- |
| A screenshot of a computer screen  Description automatically generated | A screenshot of a computer  Description automatically generated |
| A graph of confusion matrix  Description automatically generated  Figure 20 XGBoost on imbalanced data | A blue squares with white text  Description automatically generated  Figure 21 XGBoost on SMOTE data |

Analysis:Model 1: an XGBoost classifier over an imbalanced dataset with a very high test accuracy of 99.68%, although this has lower recall for fraud detection. Model 2, with the same XGBoost model but on a SMOTE-oversampled dataset, gives somewhat lower accuracies but very good performance for fraudulent transactions, where precision and recall reach 0.99 and 1.00, respectively.

Experiment 5**:** TF-IDF embedding vs BERT Embedding

|  |  |
| --- | --- |
| Figure 22 Shape of TF-IDF | Figure 23 Shape of BERT |

**Analysis:** BERT embeddings offer a more comprehensive and contextually aware representation of text as represented as large vector size (768 dimensions).

Experiment 6:Identifying anomalous transactions by embedding cosine approach vs embedding Euclidean approach using distance metrics.

|  |  |
| --- | --- |
|  |  |

Figure 24 Embedding cosine vs Euclidean distance

### Analysis:

Majority of the performance metrics in terms of accuracy, precision, recall, and F1-score are higher for the Embedding-Cosine Approach. Moreover, it also detects more anomalies. That means it is, in general, more effective and robust for the task of flagging anomalous transactions based on distance measures.

Embedding-Euclidean Approach: This showed slightly worse performance metrics compared to the Cosine Approach. Still, it could prove useful depending on the application requirements—where, in particular, a lower rate for anomaly detection is desired.

In most cases, the Cosine Approach should hence be the better choice due to its increased accuracy and, more importantly, well-balanced performance. However, the final choice between these approaches could also depend on specific needs and sensitivity trade-offs of anomaly detection.

|  |  |
| --- | --- |
| A chart with yellow and purple dots  Description automatically generated  Figure 25 Cosine approach | A graph with yellow dots  Description automatically generated  Figure 26 Euclidean approach |

Experiment 7: Identifying anomalous transactions by embedding cosine approach vs embedding Euclidean approach using BERT vs OPEN AI embeddings

|  |  |
| --- | --- |
| Figure 27 BERT Embeddings | Figure 28 OPEN AI Embeddings |

In this experiment I have utilized BERT and OPEN AI embedding methods to generate embeddings and investigated efficiency of identifying anomalies using distance matrix.

### Analysis

It is evident that Model 1 which uses BERT embeddings, and by all parameters, such as accuracy, precision, recall, and F1-score, the Embedding-Cosine Approach goes higher than the Embedding-Euclidean Approach. Moreover, it is able to detect more unique anomalies: 228 against 12 by the second approach. Contrary to this, Model 2 with OpenAI embeddings shows the Embedding-Euclidean Approach performing better than the Embedding-Cosine Approach in terms of accuracy with a value of 0.576 against 0.45 and in terms of precision with a value of 0.578 against 0.468, with an F1-score a little higher. The Embedding-Cosine approach with OpenAI embeddings, however, detects more of the unique anomalies: 204 against 25. Conclusively, BERT embeddings generally give better performance metrics, while OpenAI embeddings in the Euclidean approach give better accuracy and precision. The choice about which model to use may be, therefore, dependent on whether the priority is higher detection sensitivity or better overall performance on the classification.

Experiment 8: Evaluating similarity search on FAISS index

A screenshot of a computer

Description automatically generated

Figure 29 Evaluation of similarity search

### Analysis:

**Average Precision (AP)** shows that the model is effective in identifying relevant items with high precision early in the result set.  
**NDCG** suggests that while the model performs reasonably well, but still performs low in ranking strategy, which can be improved.  
Overall, the model exhibits strong performance in terms of precision but could enhance its ranking strategy to improve NDCG and better position relevant results.

Experiment 9: Creating a prototype RAG model to detect transaction type (fraud or genuine)

Evaluation of generative model response is complex and not straightforward as traditional models. However, for evaluation I have parsed the exact query prompt, and the model has identified it correctly.

A screenshot of a computer program

Description automatically generated

Figure 30 Evaluation of RAG model (full text)

A screenshot of a computer

Description automatically generated

Figure 31 RAG Prediction (full text)

However, if the query prompt is partial these results may vary. A diagram of blue dots

Description automatically generated

Figure 32 2D representation of embeddings

Query Prompt: ‘*what is the type of A recurring transaction originated from a Teen male customer 180936571 from merchant 980657600 to category es\_sportsandtoys with considerable amount is categorized as?* ‘

Generated Response:

A black and white text

Description automatically generated with medium confidence

Figure 33 Generative response from RAG model

‘*Prediction:  
This transaction is a fraud transaction*

*Generated Response:  
Based on the provided documents, a recurring transaction originated from a Teen male customer with a considerable amount to the category es\_sportsandtoys is categorized as a fraud transaction. ‘*

## CONCLUSION

This research explored the capabilities of Retrieval Augmented Generation (RAG) in identifying fraudulent transactions and compared its effectiveness with traditional models. Furthermore, these traditional models were customized and fine-tuned to increase accuracy and predictions. Also, experiments were carried out to see how class imbalance effect model performance.

While RAG shows adaptability and dynamic retrieval of relevant information, traditional models like Random Forest and XGBoost shows higher accuracy, precision and recall in fraud detection. The developed prototype model using vector store for efficient retrieval and generative AI for generative context rich response demonstrates the practical application of RAG. However, the effectiveness of RAG model depends on the quality of embeddings and handling of false positives.

## LIMITATIONS

This study has several limitations. The dataset used in this study was synthetically generated. Hence, it lacks the complexities and nuances of real-world transactions. Additionally, there was a limited access to more advanced embedding methods like ‘text-embedding-ada-003’. These methods can generate higher dimensional and higher quality embeddings which potentially increase the performance of RAG application. Furthermore, training a RAG application model is computationally expensive. This was a major challenge in optimizing and scaling the model effectively.

## FUTURE WORK

Future research should aim to optimize RAG’s retrieval mechanism to enhance precision and reduce false positives. Exploring alternative embedding methods and ranking strategies is also required. Integrating RAG with advanced machine learning models and developing a hybrid approach would also be worth of investigating.

## REFERENCES

Abdallah, A., Maarof, M.A. and Zainal, A., 2016. Fraud detection system: A survey. Journal of Network and Computer Applications, 68, pp.90-113.

Araci, D. and Genç, Z., 2020. *Financial Sentiment Analysis with Pre-trained Language Models*.

A Comprehensive Survey of Data Mining-based Fraud Detection Research. (n.d.). Available at: https://arxiv.org/pdf/1009.6119.

‌

A Report to the nations ® OCCUPATIONAL FRAUD 2022: A REPORT TO THE NATIONS. (n.d.). Available at: https://www.acfe.com/-/media/files/acfe/pdfs/rttn/2022/2022-report-to-the-nations.pdf.‌

Amazon Web Services, Inc. (n.d.). *What is RAG? - Retrieval-Augmented Generation Explained - AWS*. [online] Available at: https://aws.amazon.com/what-is/retrieval-augmented-generation/.

‌

Bolton, R.J. and Hand, D.J., 2002. Statistical fraud detection: A review. *Statistical Science*, 17(3), pp.235-255

Breiman, L., 2001. Random Forests. *Machine Learning*, 45(1), pp.5-32

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D., 2020. Language models are few-shot learners. *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*.

Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16(16), pp.321–357. doi:https://doi.org/10.1613/jair.953.

‌

Chen, T. and Guestrin, C. (2016). XGBoost: a Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’16*, pp.785–794.

‌

Dal Pozzolo, A., Caelen, O., Johnson, R.A. and Bontempi, G., 2015. Calibrating probability with undersampling for unbalanced classification. *IEEE Symposium Series on Computational Intelligence*.

Devlin, J., Chang, M.-W., Lee, K., Google, K. and Language, A. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. [online] pp.4171–4186. Available at: https://aclanthology.org/N19-1423.pdf.

‌

Friedman, J.H., 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), pp.1189-1232.

Gerz, D., Su, P.-H., Kusztos, R., Mondal, A., Lis, M., Singhal, E., Mrkšić, N., Wen, T.-H. and Vulić, I. (2021). *Multilingual and Cross-Lingual Intent Detection from Spoken Data*. [online] arXiv.org. Available at: https://arxiv.org/abs/2104.08524 [Accessed 18 Aug. 2024].

‌

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep Learning*. MIT Press

Izacard, G. and Grave, E. (2021). *Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2007.01282.

‌

Jarvelin, K. and Kekäläinen, J., 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4), pp.422-446.

Jason Brownlee (2020). *How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/.

‌

Johnson, J., Douze, M. and Jegou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3), pp.1–1. doi:https://doi.org/10.1109/tbdata.2019.2921572.

‌

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., Yih, W.-T. and Ai, F. (n.d.). *Dense Passage Retrieval for Open-Domain Question Answering*. [online] Available at: https://arxiv.org/pdf/2004.04906.

‌

LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep Learning. *Nature*, 521(7553), pp.436-444.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., Kiela, D., Facebook and Research, A. (2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. [online] Available at: https://arxiv.org/pdf/2005.11401.

‌

Liaw, A. and Wiener, M. (2002). Classification and Regression by randomForest. *Classification and Regression by randomForest*, [online] 2(3). Available at: https://journal.r-project.org/articles/RN-2002-022/RN-2002-022.pdf.

‌

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. [online] Available at: https://arxiv.org/pdf/1301.3781.

‌

Milvus.io. (2019). *What is Milvus | Milvus Documentation*. [online] Available at: https://milvus.io/docs/overview.md [Accessed 18 Aug. 2024].

Pattisapu, P., 2024. Credit card fraud detection using Gen AI & ML. *LinkedIn*. Available at: <https://www.linkedin.com/pulse/credit-card-fraud-detection-using-gen-ai-ml-prakash-pattisapu-wib9c/>

‌‌

Pourhabibi, T., Ong, K.L., Kam, B. and Boo, Y.L., 2020. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133, p.113303. Available at: <https://doi.org/10.1016/j.dss.2020.113303>

Qu, Y., Ding, Y., Liu, J., Liu, K., Ren, R., Zhao, W.X., Dong, D., Wu, H. and Wang, H. (2021). *RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2010.08191.

‌

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. and Liu, P.J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, [online] 21(140), pp.1–67. Available at: https://jmlr.org/papers/v21/20-074.html.

‌

Reimers, N. and Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. [online] pp.3982–3992. Available at: https://aclanthology.org/D19-1410.pdf.

‌

UK Finance. (n.d.). *Annual Fraud Report 2024*. [online] Available at: <https://www.ukfinance.org.uk/policy-and-guidance/reports-and-publications/annual-fraud-report-2024>.

www.kaggle.com. (n.d.). *Synthetic data from a financial payment system*. [online] Available at: https://www.kaggle.com/datasets/ealaxi/banksim1/data.

‌

www.proquest.com. (n.d.). *Fraud Detection using Machine Learning in e-Commerce - ProQuest*. [online] Available at: https://www.proquest.com/openview/6f17cc61f7d88de70e3ade6c284b3fa3/1?pq-origsite=gscholar&cbl=5444811.‌

‌

## APPENDIX

Link to Github Repository: https://github.com/21062872/fraud-detection-rag.git

Link to Dataset : <https://www.kaggle.com/datasets/ealaxi/banksim1/data>

Code:

# -\*- coding: utf-8 -\*-

"""main.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/github/21062872/fraud-detection-rag/blob/main/code\_files/main.ipynb

## Environment Setup

"""

# To verify if the notebook is connected to gpu

gpu\_info = !nvidia-smi

gpu\_info = '\n'.join(gpu\_info)

if gpu\_info.find('failed') >= 0:

print('Not connected to a GPU')

else:

print(gpu\_info)

# To get requirements.txt from github repository

!wget https://raw.githubusercontent.com/21062872/fraud-detection-rag/main/code\_files/requirements.txt

# Install all libraries from requirements.txt

!pip install -r requirements.txt

"""## Data Ingestion"""

import pandas as pd

# Class to load external files

class DataLoader:

def \_\_init\_\_(self, url):

self.url = url

self.dataframe = None

def load\_data(self):

"""Load data from the URL into a Pandas DataFrame."""

self.dataframe = pd.read\_csv(self.url)

return self.dataframe

# load data from external file

dataset\_url = 'https://raw.githubusercontent.com/21062872/fraud-detection-rag/main/data\_files/dataset.csv'

data\_loader = DataLoader(dataset\_url)

df\_raw = data\_loader.load\_data()

df\_raw.head(5)

df\_raw.dtypes

"""## Data Pre-processing"""

import numpy as np

class DataPreprocessor:

@staticmethod

def preprocess(dataframe):

"""Preprocess the DataFrame by stripping single quotes from string values."""

return dataframe.map(lambda x: x.strip("'") if isinstance(x, str) else x)

@staticmethod

def rename\_columns(dataframe):

"""Rename columns of the DataFrame according to specified mappings. This is done as raw dataset has additional characters beside values"""

renamed\_columns = {

'step': 'timeStep',

'customer': 'customerId',

'age': 'customerAge',

'gender': 'customerGender',

'zipcodeOri': 'originZipCode',

'merchant': 'merchantName',

'zipMerchant': 'merchantZipCode',

'category': 'merchantCategory',

'amount': 'amount',

'fraud': 'isFraud'

}

return dataframe.rename(columns=renamed\_columns)

@staticmethod

def preprocess\_gender(dataframe):

"""

Preprocess the 'customerGender' column by standardizing values and handling unknowns

Returns:

The DataFrame with the preprocessed 'customerGender' column

"""

dataframe['customerGender'] = dataframe['customerGender'].replace({'E': np.nan, 'U': np.nan})

return dataframe

@staticmethod

def convert\_to\_category\_codes(df, column\_name):

"""

Convert a categorical columns category codes

"""

df[column\_name] = df[column\_name].astype('category').cat.codes

return df

@staticmethod

def filter\_records\_with\_unidentified\_data(df, column\_name):

"""

Filters out rows where 'customerAge' is 'U'

Returns:

A DataFrame with rows where 'customerAge' is not 'U'

"""

filtered\_df = df[df[column\_name] != 'U']

return filtered\_df

@staticmethod

def process\_columns(df):

"""

Remove the first letter from 'customerId' and 'merchantName' columns,

convert these columns to integers, and drop 'originZipCode' and

'merchantZipCode' columns.

"""

# Remove the first letter and convert to integer for 'customerId'

df['customerId'] = df['customerId'].str[1:].astype(int)

# Remove the first letter and convert to integer for 'merchantName'

df['merchantName'] = df['merchantName'].str[1:].astype(int)

# Drop 'originZipCode' and 'merchantZipCode' columns

columns\_to\_remove = ['originZipCode', 'merchantZipCode']

df = df.drop(columns=[col for col in columns\_to\_remove if col in df.columns])

return df

import pandas as pd

from sklearn.preprocessing import LabelEncoder

class CatergoricalEncoder:

@staticmethod

def encode\_gender(dataframe):

"""

Encode the 'customerGender' column using label encoding

"""

# Initialize the label encoder

label\_encoder = LabelEncoder()

# Fit and transform the 'customerGender' column

dataframe['encoded\_gender'] = label\_encoder.fit\_transform(dataframe['customerGender'].astype(str))

return dataframe

@staticmethod

def encode\_merchant\_category(dataframe):

"""

Encode the 'merchantCategory' column using label encoding

"""

# Initialize the label encoder

label\_encoder = LabelEncoder()

# Fit and transform the 'merchantCategory' column

dataframe['encoded\_category'] = label\_encoder.fit\_transform(dataframe['merchantCategory'].astype(str))

return dataframe

# Preprocessing the dataset

data\_preprocessor = DataPreprocessor()

df\_processed = data\_preprocessor.preprocess(df\_raw)

df\_processed = data\_preprocessor.rename\_columns(df\_processed)

df\_processed = data\_preprocessor.preprocess\_gender(df\_processed)

df\_processed = data\_preprocessor.filter\_records\_with\_unidentified\_data(df\_processed, 'customerAge')

df\_processed = data\_preprocessor.process\_columns(df\_processed)

# Encode the 'customerGender' column

df\_processed = CatergoricalEncoder.encode\_gender(df\_processed)

# Encode the 'merchantCategory' column

df\_processed = CatergoricalEncoder.encode\_merchant\_category(df\_processed)

df\_processed.tail(5)

df\_processed.dtypes

"""### Exploratory Data Analysis (EDA)"""

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import missingno as msno

class EDA:

def \_\_init\_\_(self, df):

self.df = df

def plot\_imbalance(self, target\_variable):

"""Plot the class imbalance for fraud variable."""

# Setting the aesthetic style of the plots

sns.set\_style("whitegrid")

counts = self.df[target\_variable].value\_counts()

plt.figure(figsize=(6, 4))

counts.plot(kind='bar', color=['blue', 'red'])

plt.title(f'Imbalance of {target\_variable}')

plt.xlabel(f'{target\_variable} (1) vs Not {target\_variable} (0)')

plt.ylabel('Count')

plt.xticks(rotation=0)

plt.show()

def plot\_categorical\_distribution(self, df, column):

"""

Plot categorical distribution for a given column.

Parameters:

df (pd.DataFrame)

"""

# Setting the aesthetic style of the plots

sns.set\_style("whitegrid")

plt.figure(figsize=(8, 6))

sns.countplot(x=column, data=df)

plt.title(f'Distribution of {column}')

plt.xlabel(column)

plt.ylabel('Count')

plt.xticks(rotation=90)

plt.show()

def plot\_missing\_values\_bar(self, dataframe):

"""

Plot missing values in the given DataFrame using a bar chart.

Parameters:

dataframe (pd.DataFrame)

Returns:

None

"""

# Setting the aesthetic style of the plots

sns.set\_style("whitegrid")

msno.bar(dataframe)

plt.title('Non-Missing Values Bar Chart')

plt.xlabel('Columns')

plt.ylabel('Count of Non-Missing Values')

plt.show()

def plot\_customer\_age\_and\_gender\_distribution(self, dataframe, column):

"""

Plot the distribution of customer age and gender using a histogram with KDE.

Parameters:

dataframe (pd.DataFrame)

Returns:

None

"""

# Setting the aesthetic style of the plots

sns.set\_style("whitegrid")

plt.figure(figsize=(8, 4))

sns.histplot(dataframe[column], bins=10, kde=True)

plt.title(f'Distribution of {column}')

plt.xlabel(f'{column}')

plt.ylabel('Frequency')

plt.show()

def plot\_amount\_distribution(sef, dataframe):

"""

Plot a scatter plot to see the distribution of the 'amount' column,

marking fraud transactions in red (isFraud = 1).

Parameters:

dataframe (pd.DataFrame)

Returns:

None

"""

# Setting the aesthetic style of the plots

sns.set\_style("whitegrid")

plt.figure(figsize=(10, 6))

# Scatter plot for non-fraudulent transactions (isFraud = 0)

plt.scatter(dataframe[dataframe['isFraud'] == 0]['timeStep'],

dataframe[dataframe['isFraud'] == 0]['amount'],

alpha=0.5, c='blue', label='Non-Fraudulent')

# Scatter plot for fraudulent transactions (isFraud = 1)

plt.scatter(dataframe[dataframe['isFraud'] == 1]['timeStep'],

dataframe[dataframe['isFraud'] == 1]['amount'],

alpha=0.5, c='red', label='Fraudulent')

# Plot styling

plt.title('Distribution of Transaction Amounts Over Time')

plt.xlabel('Time Step')

plt.ylabel('Transaction Amount')

plt.legend()

plt.show()

def plot\_amount\_boxplot(self,dataframe):

"""

Plot a detailed and customized box plot to see the distribution of the 'amount' column.

Parameters:

dataframe (pd.DataFrame)

Returns:

None

"""

# Setting the aesthetic style of the plots

sns.set\_style("whitegrid")

# Creating the box plot

plt.figure(figsize=(12, 8))

box\_plot = sns.boxplot(

y=dataframe['amount'],

color='skyblue',

flierprops={'marker': 'o', 'markersize': 5, 'markerfacecolor': 'red'}

)

# Customizing the plot

plt.title('Box Plot of Transaction Amounts', fontsize=16)

plt.ylabel('Transaction Amount', fontsize=14)

plt.xlabel('Transactions', fontsize=14)

# Adding grid lines for better readability

plt.grid(True, linestyle='--', linewidth=0.5)

# Annotating the plot with more details

plt.annotate(

'Outliers',

xy=(0, dataframe['amount'].max()),

xytext=(0, dataframe['amount'].max() + 10),

arrowprops=dict(facecolor='black', arrowstyle='->'),

fontsize=12

)

# Adding mean and median lines

mean\_value = dataframe['amount'].mean()

median\_value = dataframe['amount'].median()

plt.axhline(mean\_value, color='green', linestyle='--', linewidth=1.5, label=f'Mean: {mean\_value:.2f}')

plt.axhline(median\_value, color='blue', linestyle='-', linewidth=1.5, label=f'Median: {median\_value:.2f}')

plt.legend()

# Showing the plot

plt.show()

def plot\_fraud\_percentage\_by\_age(self, df, age\_column='customerAge'):

"""

Creates a stacked bar plot to show the percentage of fraud and non-fraud transactions by customer age.

Parameters:

df (pd.DataFrame)

age\_column (str)

"""

# Calculate the percentage of fraud and not fraud transactions by the specified age column

age\_fraud\_summary = df.groupby([age\_column, 'isFraud']).size().unstack(fill\_value=0)

age\_fraud\_summary\_percentage = age\_fraud\_summary.div(age\_fraud\_summary.sum(axis=1), axis=0) \* 100

# Create the stacked bar plot

fig, ax = plt.subplots(figsize=(12, 8))

age\_fraud\_summary\_percentage.plot(kind='bar', stacked=True, ax=ax, color=['blue', 'red'])

# Set labels and title

ax.set\_xlabel('Customer Age')

ax.set\_ylabel('Percentage of Transactions')

ax.set\_title('Percentage of Fraud and Not Fraud Transactions by Customer Age')

ax.legend(title='Transaction Type', labels=['Not Fraud', 'Fraud'])

# Show the plot

plt.show()

# plot imabalnce of target variable

eda = EDA(df\_processed)

eda.plot\_imbalance('isFraud')

#Plot categorical distribution for merchantCategory column

eda.plot\_categorical\_distribution(df\_processed, 'merchantCategory')

# Plot non-missing values in the given DataFrame using a bar chart

eda.plot\_missing\_values\_bar(df\_processed)

#Plot the distribution of customer age

eda.plot\_customer\_age\_and\_gender\_distribution(df\_processed, 'customerAge')

#Plot the distribution of customer gender

eda.plot\_customer\_age\_and\_gender\_distribution(df\_processed, 'customerGender')

eda.plot\_fraud\_percentage\_by\_age(df\_processed, age\_column='customerAge')

eda.plot\_amount\_distribution(df\_processed)

eda.plot\_amount\_boxplot(df\_processed)

"""EDA for fraudlent transactions"""

# For fraud Transactions

df\_fraud\_trx = df\_processed[df\_processed['isFraud'] == 1]

edaFraud = EDA(df\_fraud\_trx)

#Plot categorical distribution for merchantCategory column

edaFraud.plot\_categorical\_distribution(df\_fraud\_trx, 'merchantCategory')

#Plot the distribution of customer age

edaFraud.plot\_customer\_age\_and\_gender\_distribution(df\_fraud\_trx, 'customerAge')

#Plot the distribution of customer gender

edaFraud.plot\_customer\_age\_and\_gender\_distribution(df\_fraud\_trx, 'customerGender')

"""### Relationships"""

def plot\_filtered\_pairplot(df, num\_vars, hue\_column, palette='viridis', title='Pair Plot'):

"""

Generate a pair plot for specified numerical variables in the DataFrame.

Returns:

None: Displays the pair plot.

"""

# Convert customerAge to numeric

df['customerAge'] = pd.to\_numeric(df['customerAge'], errors='coerce')

# Ensure hue\_column is categorical

if not pd.api.types.is\_categorical\_dtype(df[hue\_column]):

df[hue\_column] = df[hue\_column].astype('category')

# Filter DataFrame to include only numerical variables and hue column

df\_filtered = df[num\_vars + [hue\_column]].dropna()

# Create the pair plot

pair\_plot = sns.pairplot(df\_filtered, vars=num\_vars, hue=hue\_column, palette=palette)

# Set the title of the plot

pair\_plot.fig.suptitle(title, y=1.02)

# Show the plot

plt.show()

plot\_filtered\_pairplot(df\_processed, num\_vars=['customerAge', 'amount'], hue\_column='isFraud')

def plot\_correlation\_heatmap(df, exclude\_columns, target\_column):

"""

Create a correlation heatmap for numerical columns in the DataFrame, excluding specified columns.

Returns: Displays the correlation heatmap.

"""

# Drop excluded columns

df\_filtered = df.drop(columns=exclude\_columns)

# Ensure target\_column is categorical

if not pd.api.types.is\_categorical\_dtype(df[target\_column]):

df[target\_column] = df[target\_column].astype('category')

# Convert categorical target\_column to numerical for correlation calculation

df\_filtered[target\_column] = df\_filtered[target\_column].cat.codes

# Compute the correlation matrix

correlation\_matrix = df\_filtered.corr()

# Create the heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()

exclude\_columns = ['customerGender', 'merchantCategory']

target\_column = 'isFraud'

plot\_correlation\_heatmap(df\_processed, exclude\_columns, target\_column)

"""### More pre-processing to handle categorical values"""

df\_processed\_rag = df\_processed.copy()

"""## Feature Engineering"""

def process\_transactions(df):

"""

Adds 'IslargeTransaction' and 'countForCustomerSameTime' columns to the DataFrame

Returns: DataFrame with the new columns added

"""

# Add 'IslargeTransaction' column based on the condition

df['IslargeTransaction'] = df['amount'].apply(lambda x: 1 if x >= 5000 else 0)

# Calculate the count of transactions for each customer at each time step

df['countForCustomerSameTime'] = df.groupby(['timeStep', 'customerId'])['customerId'].transform('count')

return df

def add\_avg\_transaction\_amount(df, customer\_id\_col='customerId', amount\_col='amount'):

"""

Adds a new column to the DataFrame that contains the average transaction amount

for each customerId.

Returns:The DataFrame with an added column 'avgTransactionAmount'

"""

# Calculate the average transaction amount for each customerId

avg\_transaction\_amount = df.groupby(customer\_id\_col)[amount\_col].mean()

# Map the average transaction amount back to the original DataFrame

df['avgTransactionAmount'] = df[customer\_id\_col].map(avg\_transaction\_amount)

return df

# Apply the function to process the DataFrame

df\_processed = process\_transactions(df\_processed)

df\_processed\_rag = process\_transactions(df\_processed\_rag)

df\_processed = add\_avg\_transaction\_amount(df\_processed)

df\_processed\_rag = add\_avg\_transaction\_amount(df\_processed\_rag)

df\_processed.head()

"""# Baseline models for Fraud Detection"""

df\_processed.dtypes

"""## Random Forest Classifier"""

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.preprocessing import LabelEncoder

def train\_random\_forest(df, categorical\_columns, target\_column='isFraud', test\_size=0.25, n\_estimators=10, random\_state=42):

"""

Trains a Random Forest classifier and evaluates its performance.

Returns:

None

"""

# Encode categorical features using LabelEncoder

label\_encoders = {}

for col in categorical\_columns:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

label\_encoders[col] = le

# Define features and target variable

X = df.drop(columns=[target\_column])

y = df[target\_column]

# Split the data into training and testing sets

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

# Initialize the Random Forest Classifier

rf\_clf = RandomForestClassifier(n\_estimators=n\_estimators, random\_state=random\_state)

# Train the model

rf\_clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = rf\_clf.predict(X\_test)

# Evaluate the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

"""\*\*Experiment 1: RandomForest without oversampling\*\*"""

# Columns to encode

categorical\_columns = ['customerId', 'merchantName', 'customerAge', 'merchantCategory', 'customerGender']

# Call the function to train the Random Forest classifier

train\_random\_forest(df\_processed, categorical\_columns)

"""### Addressing Dataset Imbalance

### SMOTE - (Synthetic Minority Over-sampling Technique)

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from imblearn.over\_sampling import SMOTE

from sklearn.preprocessing import StandardScaler, LabelEncoder

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

class SmoteOverSampling:

def \_\_init\_\_(self, dataframe, target\_column, test\_size=0.2, random\_state=42):

"""

Initialize the SmoteOverSampling with the given parameters.

"""

self.dataframe = dataframe

self.target\_column = target\_column

self.test\_size = test\_size

self.random\_state = random\_state

def encode\_categorical(self):

"""

Encode categorical features using integer encoding.

"""

label\_encoders = {}

for column in self.dataframe.select\_dtypes(include=['object']).columns:

label\_encoders[column] = LabelEncoder()

self.dataframe[column] = label\_encoders[column].fit\_transform(self.dataframe[column])

self.label\_encoders = label\_encoders

def apply\_smote(self):

"""

Apply SMOTE to balance the dataset.

"""

X = self.dataframe.drop(self.target\_column, axis=1)

y = self.dataframe[self.target\_column]

smote = SMOTE(random\_state=self.random\_state)

X\_resampled, y\_resampled = smote.fit\_resample(X, y)

self.X\_resampled = pd.DataFrame(X\_resampled, columns=X.columns)

self.y\_resampled = pd.Series(y\_resampled, name=self.target\_column)

# Create DataFrame for the original and synthetic samples

self.df\_processed\_os = self.X\_resampled.copy()

self.df\_processed\_os[self.target\_column] = self.y\_resampled

def scale\_features(self):

"""

Scale the features using StandardScaler.

"""

scaler = StandardScaler()

self.X\_resampled\_scaled = scaler.fit\_transform(self.X\_resampled)

def split\_data(self):

"""

Split the data into training and testing sets.

"""

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

self.X\_resampled\_scaled, self.y\_resampled,

test\_size=self.test\_size, random\_state=self.random\_state

)

return X\_train, X\_test, y\_train, y\_test

# Initialize the preprocessor

preprocessor = SmoteOverSampling(df\_processed, target\_column='isFraud')

# Encode categorical features

preprocessor.encode\_categorical()

# Apply SMOTE to balance the dataset

preprocessor.apply\_smote()

df\_processed\_os = preprocessor.df\_processed\_os

preprocessor.scale\_features()

X\_train, X\_test, y\_train, y\_test = preprocessor.split\_data()

# plot imabalnce of target variable after SMOTE

eda = EDA(df\_processed\_os)

eda.plot\_imbalance('isFraud')

"""\*\*Experiment 2: RandomForest with oversampled data (SMOTE) and optimization\*\*"""

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.preprocessing import LabelEncoder, StandardScaler

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

def train\_random\_forest(df, categorical\_columns, target\_column='isFraud', test\_size=0.25, n\_estimators=100, max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, random\_state=42):

"""

Trains a Random Forest classifier and evaluates its performance with optimized hyperparameters.

"""

# Encode categorical features using LabelEncoder

label\_encoders = {}

for col in categorical\_columns:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

label\_encoders[col] = le

# Define features and target variable

X = df.drop(columns=[target\_column])

y = df[target\_column]

#Standardize features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split the data into training and testing sets

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

# Initialize the Random Forest Classifier with additional hyperparameters

rf\_clf = RandomForestClassifier(n\_estimators=n\_estimators,

max\_depth=max\_depth,

min\_samples\_split=min\_samples\_split,

min\_samples\_leaf=min\_samples\_leaf,

random\_state=random\_state)

# Train the model

rf\_clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = rf\_clf.predict(X\_test)

# Evaluate the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Feature importances

feature\_importances = pd.DataFrame(rf\_clf.feature\_importances\_,

index=[f'Feature\_{i}' for i in range(X.shape[1])],

columns=['Importance']).sort\_values('Importance', ascending=False)

print("\nFeature Importances:\n", feature\_importances)

# Plot feature importances

plt.figure(figsize=(10, 6))

sns.barplot(x='Importance', y=feature\_importances.index, data=feature\_importances, palette='viridis')

plt.title('Feature Importances')

plt.xlabel('Importance')

plt.ylabel('Feature')

plt.show()

# List of categorical columns

categorical\_columns = ['customerGender', 'merchantCategory']

# Call the train\_random\_forest function

train\_random\_forest(df\_processed,

categorical\_columns=categorical\_columns,

target\_column='isFraud',

test\_size=0.3,

n\_estimators=100,

max\_depth=10,

min\_samples\_split=2,

min\_samples\_leaf=1,

random\_state=42)

def visualize\_random\_forest\_tree(df, categorical\_columns, target\_column='isFraud', test\_size=0.25, n\_estimators=100, max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, random\_state=42, tree\_index=0):

"""

Trains a Random Forest classifier and visualizes one of the trees in the forest.

Returns:

None

"""

# Encode categorical features using LabelEncoder

label\_encoders = {}

for col in categorical\_columns:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

label\_encoders[col] = le

# Define features and target variable

X = df.drop(columns=[target\_column])

y = df[target\_column]

# Standardize features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split the data into training and testing sets

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

# Initialize the Random Forest Classifier with additional hyperparameters

rf\_clf = RandomForestClassifier(n\_estimators=n\_estimators,

max\_depth=max\_depth,

min\_samples\_split=min\_samples\_split,

min\_samples\_leaf=min\_samples\_leaf,

random\_state=random\_state)

# Train the model

rf\_clf.fit(X\_train, y\_train)

# Extract one tree from the forest

tree\_clf = rf\_clf.estimators\_[tree\_index]

# Visualize the tree

dot\_data = export\_graphviz(tree\_clf, out\_file=None,

feature\_names=df.drop(columns=[target\_column]).columns,

class\_names=['Not Fraud', 'Fraud'],

filled=True, rounded=True, special\_characters=True)

graph = graphviz.Source(dot\_data)

graph.render(filename='random\_forest\_tree', format='png', cleanup=False) # Save as PNG

print("Tree visualized and saved as 'random\_forest\_tree.png'.")

# Display the tree

plt.figure(figsize=(20, 15))

plt.imshow(plt.imread('random\_forest\_tree.png'))

plt.axis('off')

plt.show()

import graphviz

from sklearn.tree import export\_graphviz

# List of categorical columns

categorical\_columns = ['customerGender', 'merchantCategory']

# Call the function

visualize\_random\_forest\_tree(df\_processed,

categorical\_columns=categorical\_columns,

target\_column='isFraud',

test\_size=0.3,

n\_estimators=100,

max\_depth=10,

min\_samples\_split=2,

min\_samples\_leaf=1,

random\_state=42,

tree\_index=0)

"""# Neural Network

### Experiment 3: on basic neural network model

"""

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

def create\_nn\_basic\_model(input\_dim):

"""

Creates a neural network model.

Parameters:

input\_dim

Returns:

model (Sequential): Compiled neural network model.

"""

model = Sequential()

model.add(Dense(64, input\_dim=input\_dim, activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(1, activation='sigmoid')) # Output layer for binary classification

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

def train\_and\_evaluate\_nn\_basic(df, categorical\_columns, target\_column='isFraud', test\_size=0.25, random\_state=42):

"""

Trains a neural network on the data and evaluates its performance.

"""

# Define features and target variable

X = df.drop(columns=[target\_column])

y = df[target\_column]

# Split the data into training and testing sets

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

# Create the neural network model

model = create\_nn\_model(input\_dim=X\_train.shape[1])

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2, verbose=1)

# Make predictions

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

y\_pred = (y\_pred\_prob > 0.5).astype(int).flatten()

# Evaluate the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Specify categorical columns and target column

categorical\_columns = ['customerGender', 'merchantCategory']

target\_column = 'isFraud'

# Call the function to train and evaluate the neural network

train\_and\_evaluate\_nn\_basic(df\_processed, categorical\_columns, target\_column, test\_size=0.25, random\_state=42)

"""### Experiment 4: on optimized neural network model"""

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Input

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.callbacks import EarlyStopping

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

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

def create\_nn\_model(input\_dim):

"""

Creates a neural network model with dropout layers.

Returns: Compiled neural network model.

"""

model = Sequential()

model.add(Input(shape=(input\_dim,))) # Define the input shape using Input layer

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5)) # Dropout layer with 50% dropout rate

model.add(Dense(32, activation='relu'))

model.add(Dropout(0.5)) # Dropout layer with 50% dropout rate

model.add(Dense(1, activation='sigmoid')) # Output layer for binary classification

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

def train\_and\_evaluate\_nn(df, categorical\_columns, target\_column='isFraud', test\_size=0.25, random\_state=42):

"""

Trains a neural network on the data and evaluates its performance.

Returns:

None

"""

# Convert categorical target variable to numeric

df[target\_column] = LabelEncoder().fit\_transform(df[target\_column])

# Encode categorical features using LabelEncoder

label\_encoders = {}

for col in categorical\_columns:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col])

label\_encoders[col] = le

# Define features and target variable

X = df.drop(columns=[target\_column])

y = df[target\_column]

# Standardize features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split the data into training and testing sets

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

# Create the neural network model

model = create\_nn\_model(input\_dim=X\_train.shape[1])

# Early stopping to prevent overfitting

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2, callbacks=[early\_stopping], verbose=1)

# Make predictions

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

y\_pred = (y\_pred\_prob > 0.5).astype(int).flatten()

# Evaluate the model

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Plot training & validation loss and accuracy

plt.figure(figsize=(12, 5))

# Plot loss

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

# Plot accuracy

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

"""\*\*Experiment 4.1 : For imbalanced dataset\*\*"""

# Specify categorical columns and target column

categorical\_columns = ['customerGender', 'merchantCategory']

target\_column = 'isFraud'

# Call the function to train and evaluate the neural network

train\_and\_evaluate\_nn(df\_processed, categorical\_columns, target\_column, test\_size=0.25, random\_state=42)

"""\*\*Experiment 4.2 : For oversampled dataset\*\*"""

# Specify categorical columns and target column

categorical\_columns = ['customerGender', 'merchantCategory']

target\_column = 'isFraud'

# Call the function to train and evaluate the neural network

train\_and\_evaluate\_nn(df\_processed\_os, categorical\_columns, target\_column, test\_size=0.25, random\_state=42)

"""# XGBOOST"""

df\_processed.head()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

import xgboost as xgb

def preprocess\_data(df):

"""

Preprocess the data by splitting into features and target, and performing train-test split.

:return: X\_train, X\_test, y\_train, y\_test

"""

X = df.drop(columns=['isFraud'])

y = df['isFraud']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.25, stratify=y, random\_state=42

)

return X\_train, X\_test, y\_train, y\_test

def build\_pipeline():

"""

Build and return a machine learning pipeline with preprocessing and XGBoost classifier.

:return: Pipeline

"""

numeric\_features = ['amount', 'customerAge', 'countForCustomerSameTime', 'timeStep', 'avgTransactionAmount']

categorical\_features = ['customerGender', 'merchantName', 'merchantCategory']

numeric\_transformer = Pipeline(steps=[

('scaler', StandardScaler())

])

categorical\_transformer = Pipeline(steps=[

('onehot', OneHotEncoder(handle\_unknown='ignore'))

])

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_features),

('cat', categorical\_transformer, categorical\_features)

]

)

xgb\_model = xgb.XGBClassifier(

objective='binary:logistic',

eval\_metric='logloss',

use\_label\_encoder=False,

random\_state=42

)

pipeline = Pipeline(steps=[

('preprocessor', preprocessor),

('classifier', xgb\_model)

])

return pipeline

def plot\_confusion\_matrix(cm, labels):

"""

Plot the confusion matrix.

:param cm: Confusion matrix

:param labels: List of class labels

"""

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()

def evaluate\_model(pipeline, X\_train, X\_test, y\_train, y\_test):

"""

Train and evaluate the model.

:param pipeline: Pipeline

:param X\_train: Training features

:param X\_test: Testing features

:param y\_train: Training target

:param y\_test: Testing target

"""

# Train the model

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

# Make predictions

y\_pred\_train = pipeline.predict(X\_train)

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

# Evaluate the model

train\_accuracy = accuracy\_score(y\_train, y\_pred\_train)

test\_accuracy = accuracy\_score(y\_test, y\_pred\_test)

print(f"Training Accuracy: {train\_accuracy:.4f}")

print(f"Test Accuracy: {test\_accuracy:.4f}")

print("Classification Report:")

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

print("Confusion Matrix:")

cm = confusion\_matrix(y\_test, y\_pred\_test)

print(cm)

# Plot the confusion matrix

plot\_confusion\_matrix(cm, labels=['Non-Fraud', 'Fraud'])

def main(df):

"""

Main function to run the data processing, model training, and evaluation.

"""

X\_train, X\_test, y\_train, y\_test = preprocess\_data(df)

pipeline = build\_pipeline()

evaluate\_model(pipeline, X\_train, X\_test, y\_train, y\_test)

"""\*\*Experiment 5: XGBoost (Extreme Gradient Boosting)\*\*"""

# raw dataset

if \_\_name\_\_ == "\_\_main\_\_":

main(df\_processed)

# SMOTE dataset

if \_\_name\_\_ == "\_\_main\_\_":

main(df\_processed\_os)

"""## RAG"""

import faiss

print(faiss.\_\_version\_\_)

df\_processed.dtypes

"""# Retrieval Model"""

import os

# File path to the FAISS index

index\_file\_path = 'faiss\_index.index'

def delete\_faiss\_index(index\_file\_path):

"""

Delete the FAISS index file if it exists.

"""

if os.path.exists(index\_file\_path):

os.remove(index\_file\_path)

print(f"FAISS index file {index\_file\_path} has been deleted.")

else:

print(f"FAISS index file {index\_file\_path} does not exist.")

# Call the function to delete the FAISS index file

delete\_faiss\_index(index\_file\_path)

import numpy as np

import faiss

import pandas as pd

import os

# Assume df\_processed is already defined and processed

features = df\_processed.drop(columns=['isFraud', 'customerGender', 'merchantCategory'])

embedding\_matrix = features.values.astype('float32')

# File path to save/load the FAISS index

index\_file\_path = 'faiss\_index.index'

def create\_faiss\_index(embedding\_matrix, index\_file\_path):

"""

Create a FAISS index from the embedding matrix and save it to a file.

"""

# Ensure the array is C-contiguous

embedding\_matrix = np.ascontiguousarray(embedding\_matrix)

dimension = embedding\_matrix.shape[1]

index = faiss.IndexFlatL2(dimension) # L2 distance

index.add(embedding\_matrix) # Add embeddings to the index

faiss.write\_index(index, index\_file\_path)

print(f"FAISS index created and saved to {index\_file\_path}")

def load\_faiss\_index(index\_file\_path):

"""

Load a FAISS index from a file.

"""

return faiss.read\_index(index\_file\_path)

def get\_faiss\_index(embedding\_matrix, index\_file\_path):

"""

Get the FAISS index

"""

if os.path.exists(index\_file\_path):

print(f"Loading FAISS index from {index\_file\_path}")

index = load\_faiss\_index(index\_file\_path)

else:

print(f"FAISS index not found. Creating new index.")

create\_faiss\_index(embedding\_matrix, index\_file\_path)

index = load\_faiss\_index(index\_file\_path)

return index

# Create or load the FAISS index

index = get\_faiss\_index(embedding\_matrix, index\_file\_path)

def get\_similar\_transactions(transaction\_vector, k=3):

"""

Retrieve k most similar transactions to the given transaction vector along with their similarity scores

"""

transaction\_vector = np.ascontiguousarray(transaction\_vector).astype('float32')

D, I = index.search(np.array([transaction\_vector]), k)

return I[0], D[0]

transaction\_vector = embedding\_matrix[88] # Example transaction vector

similar\_indices, similarity\_scores = get\_similar\_transactions(transaction\_vector)

print(f"Indices of similar transactions: {similar\_indices}")

print(f"Similarity scores (distances): {similarity\_scores}")

print(df\_processed.iloc[88])

print(df\_processed.iloc[3901])

print(df\_processed.iloc[16988])

"""## Experiment with Text embeddings using Sentence Transformer"""

df\_processed\_before = df\_processed.copy()

df\_processed = df\_processed\_before.copy()

import pandas as pd

def categorize\_amount(amount):

"""

Categorize the amount into different spending categories.

Returns: The category of the amount.

"""

if amount < 1:

return 'low amount'

elif 1 <= amount < 100:

return 'intermediate amount'

elif 100 <= amount < 1000:

return 'considerable amount'

elif 1000 <= amount < 5000:

return 'large amount'

else:

return 'hefty amount'

# Apply the categorize\_amount

df\_processed['bhv\_amount'] = df\_processed['amount'].apply(categorize\_amount)

# Categorize 'countForCustomerSameTime' into 'uncommon transaction' or 'recurring transaction'

df\_processed['bhv\_frequent'] = df\_processed['countForCustomerSameTime'].apply(

lambda x: 'uncommon transaction' if x == 1 else 'recurring transaction'

)

# Categorize 'isFraud' into 'fraud transaction' or 'genuine transaction'

df\_processed['bhv\_isFraud'] = df\_processed['isFraud'].apply(

lambda x: 'fraud transaction' if x == 1 else 'genuine transaction'

)

# Categorize 'customerGender' into 'male' or 'female'

df\_processed['bhv\_gender'] = df\_processed['customerGender'].apply(

lambda x: 'male' if x == 'M' else 'female'

)

# Convert 'customerAge' to numeric, handling errors by coercing to NaN

df\_processed['customerAge'] = pd.to\_numeric(df\_processed['customerAge'], errors='coerce')

def categorize\_customer\_age(customer\_age):

"""

Categorize the customer age into different age groups.

Parameters:

customer\_age (float): The customer age to categorize.

Returns:

str: The category of the customer age.

"""

if pd.isna(customer\_age): # Handle NaN values

return 'Unknown'

elif customer\_age < 2:

return 'Child'

elif 2 <= customer\_age < 4:

return 'Teen'

else:

return 'Adult'

# Apply the categorize\_customer\_age function to the 'customerAge' column

df\_processed['bhv\_customerAge'] = df\_processed['customerAge'].apply(categorize\_customer\_age)

# Create a new column 'transaction\_behavior' with optimized string formatting

df\_processed['transaction\_behavior'] = df\_processed.apply(

lambda row: (

f"A {row['bhv\_frequent']} originated from "

f"a {row['bhv\_customerAge']} {row['bhv\_gender']} customer {row['customerId']} "

f"from merchant {row['merchantName']} "

f"to category {row['merchantCategory']} "

f"with {row['bhv\_amount']} is categorized as {row['bhv\_isFraud']}"

),

axis=1

)

df\_processed.tail(5)

"""## Experiment 6: Embedding method (TF-IDF vs BERT)"""

# Using TF-IDF Vectorization

from sklearn.feature\_extraction.text import TfidfVectorizer

# Sample data

text = ["Transaction Behavior: A uncommon transaction originated from Adult male customer 1038277619 from merchant 980657600 to category sportsandtoys with considerable amount is categorized as fraud transaction."]

# Initialize the TF-IDF vectorizer

vectorizer = TfidfVectorizer()

# Fit and transform the data

tfidf\_matrix = vectorizer.fit\_transform(text)

# Convert the TF-IDF matrix to an array

tfidf\_vector = tfidf\_matrix.toarray()

print('Input Text :', text)

print('Empbedding Vector Shape :', tfidf\_vector.shape)

print('Empbedding :', tfidf\_vector)

# Using Transformer Models (e.g., BERT)

from transformers import BertTokenizer, BertModel

import torch

# Load pre-trained model and tokenizer

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

model = BertModel.from\_pretrained('bert-base-uncased')

# Sample data

text = ["Transaction Behavior: A uncommon transaction originated from Adult male customer 1038277619 from merchant 980657600 to category sportsandtoys with considerable amount is categorized as fraud transaction."]

# Tokenize and encode the text

inputs = tokenizer(text, return\_tensors='pt', padding=True, truncation=True)

# Forward pass to get embeddings

with torch.no\_grad():

outputs = model(\*\*inputs)

# Get the embeddings (use the last hidden state or pooled output)

embeddings = outputs.last\_hidden\_state.mean(dim=1).numpy()

print('Input Text :', text)

print('Empbedding Vector Shape :', embeddings.shape)

print('Empbedding :', embeddings)

'''

import pandas as pd

# Ensure that df\_processed is not empty and has the expected columns

if df\_processed is None or 'isFraud' not in df\_processed.columns:

raise ValueError("DataFrame is not loaded properly or missing 'isFraud' column")

# Filter the DataFrame based on isFraud values

fraud\_records = df\_processed[df\_processed['isFraud'] == 1]

non\_fraud\_records = df\_processed[df\_processed['isFraud'] == 0]

# Calculate the number of records needed

total\_records = 100000

fraud\_percentage = 0.5

fraud\_needed = int(total\_records \* fraud\_percentage)

non\_fraud\_needed = total\_records - fraud\_needed

# Sample the required number of records

fraud\_sample = fraud\_records.sample(n=fraud\_needed, random\_state=1)

non\_fraud\_sample = non\_fraud\_records.sample(n=non\_fraud\_needed, random\_state=1)

# Combine the sampled data

df\_subset = pd.concat([fraud\_sample, non\_fraud\_sample], ignore\_index=True)

# Shuffle the DataFrame to randomize the order

df\_subset = df\_subset.sample(frac=1, random\_state=1).reset\_index(drop=True)

'''

df\_before\_subset = df\_processed.copy()

df\_processed = df\_before\_subset.copy()

df\_subset.head(3)

# Shaping this dataframe for below experiment

# df\_test = df\_subset.drop('embeddings', axis = 1) --2nd time

df\_test = df\_subset # 1st time run

"""## Experiment: Flag anomolous transaction using BERT tokenizer

### 0.5 weight

"""

import pandas as pd

import numpy as np

import datetime

import os

import concurrent.futures

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from scipy.spatial.distance import pdist, squareform

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

import torch

from transformers import BertTokenizer, BertModel

# Load BERT model and tokenizer

bert\_model\_name = "bert-base-uncased"

tokenizer = BertTokenizer.from\_pretrained(bert\_model\_name)

bert\_model = BertModel.from\_pretrained(bert\_model\_name)

def preprocess\_data(df):

numerical\_columns = ['amount', 'avgTransactionAmount', 'timeStep', 'merchantName', 'customerId', 'IslargeTransaction', 'countForCustomerSameTime']

features = df[numerical\_columns].copy()

# One-hot encoding

features = pd.concat([features, pd.get\_dummies(df['merchantCategory'])], axis=1)

features = pd.concat([features, pd.get\_dummies(df['customerGender'])], axis=1)

return features

def train\_test\_split\_data(features, labels):

return train\_test\_split(features, labels, test\_size=0.5, random\_state=42)

def scale\_features(X\_train, X\_test):

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

return X\_train\_scaled, X\_test\_scaled

def handle\_imbalanced\_data(X\_train\_scaled, y\_train):

smote = SMOTE(random\_state=42)

return smote.fit\_resample(X\_train\_scaled, y\_train)

def train\_random\_forest(X\_train\_smote, y\_train\_smote):

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_train\_smote, y\_train\_smote)

return rf\_model

def evaluate\_model(model, X\_test\_scaled, y\_test):

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

return {

"accuracy": accuracy\_score(y\_test, y\_pred),

"precision": precision\_score(y\_test, y\_pred),

"recall": recall\_score(y\_test, y\_pred),

"f1": f1\_score(y\_test, y\_pred),

"conf\_matrix": confusion\_matrix(y\_test, y\_pred)

}

def get\_bert\_embeddings(texts):

embeddings = []

for text in texts:

inputs = tokenizer(text, return\_tensors='pt', padding=True, truncation=True, max\_length=512)

with torch.no\_grad():

outputs = bert\_model(\*\*inputs)

# Use the [CLS] token's representation as the sentence embedding

cls\_embedding = outputs.last\_hidden\_state[:, 0, :].squeeze().numpy()

embeddings.append(cls\_embedding)

return embeddings

def process\_embeddings(df\_test):

if os.path.exists('bert\_embeddings\_idx1.npy'):

embeddings\_array = np.load('bert\_embeddings\_idx1.npy')

else:

df\_test = df\_test.fillna('').astype(str)

df\_test['combined'] = df\_test.apply(lambda row: ' '.join(row), axis=1)

combined\_texts = df\_test['combined'].tolist()

embeddings\_list = get\_bert\_embeddings(combined\_texts)

embeddings\_array = np.array([np.array(embedding).flatten() for embedding in embeddings\_list])

np.save('bert\_embeddings\_idx1.npy', embeddings\_array)

return embeddings\_array

def normalize\_embeddings(embeddings\_array):

scaler = StandardScaler()

return scaler.fit\_transform(embeddings\_array)

def calculate\_dissimilarity(embeddings\_array):

cosine\_dissimilarity\_matrix = squareform(pdist(embeddings\_array, 'cosine'))

euclidean\_distance\_matrix = squareform(pdist(embeddings\_array, 'euclidean'))

return cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix

def find\_anomalies(distance\_matrix, threshold):

anomaly\_indices = np.where(distance\_matrix > threshold)

anomaly\_transactions = set()

for i, j in zip(\*anomaly\_indices):

if i != j:

anomaly\_transactions.add(i)

anomaly\_transactions.add(j)

return anomaly\_transactions

def add\_anomaly\_column(df\_test, anomaly\_indices, column\_name):

index\_mapping = df\_test.index.tolist()

mapped\_anomaly\_indices = [index\_mapping[i] for i in anomaly\_indices]

df\_test[column\_name] = 0

df\_test.loc[mapped\_anomaly\_indices, column\_name] = 1

def calculate\_metrics(y\_true, y\_pred):

return {

"accuracy": accuracy\_score(y\_true, y\_pred),

"precision": precision\_score(y\_true, y\_pred, zero\_division=0),

"recall": recall\_score(y\_true, y\_pred, zero\_division=0),

"f1": f1\_score(y\_true, y\_pred, zero\_division=0)

}

def compare\_anomalies(df\_test, X\_test, rf\_predictions):

embedding\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'] == 1].index)

euclidean\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'] == 1].index)

rf\_anomalies = set(X\_test.index[rf\_predictions == 1])

matched\_anomalies = embedding\_anomalies.intersection(rf\_anomalies)

unique\_to\_embeddings = embedding\_anomalies - rf\_anomalies

unique\_to\_rf = rf\_anomalies - embedding\_anomalies

unique\_to\_euclidean = euclidean\_anomalies - rf\_anomalies - embedding\_anomalies

return {

"matched\_anomalies": len(matched\_anomalies),

"unique\_to\_embeddings": len(unique\_to\_embeddings),

"unique\_to\_rf": len(unique\_to\_rf),

"unique\_to\_euclidean": len(unique\_to\_euclidean)

}

def generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison):

summary = {

"Random Forest Approach": {\*\*rf\_metrics, "Unique Anomalies": comparison.get("unique\_to\_rf", 0)},

"Embedding-Cosine Approach": {\*\*embedding\_metrics, "Unique Anomalies": comparison.get("unique\_to\_embeddings", 0)},

"Embedding-Euclidean Approach": {\*\*euclidean\_metrics, "Unique Anomalies": comparison.get("unique\_to\_euclidean", 0)}

}

# Debugging output to check the structure of summary

print("Summary dictionary structure:")

for approach, metrics in summary.items():

if isinstance(metrics, dict):

print(f"{approach}:")

for metric, value in metrics.items():

print(f" {metric}: {value}")

else:

print(f"{approach}: {metrics}")

print()

# Functions for dimensionality reduction and visualization

def reduce\_dimensions(embeddings\_array, method='PCA', n\_components=2):

if method == 'PCA':

reducer = PCA(n\_components=n\_components)

elif method == 't-SNE':

reducer = TSNE(n\_components=n\_components, random\_state=42)

else:

raise ValueError("Method should be either 'PCA' or 't-SNE'")

return reducer.fit\_transform(embeddings\_array)

def plot\_embeddings(embeddings\_reduced, labels, title):

plt.figure(figsize=(10, 8))

scatter = plt.scatter(embeddings\_reduced[:, 0], embeddings\_reduced[:, 1], c=labels, cmap='viridis', alpha=0.6)

plt.colorbar(scatter, label='Anomaly Status')

plt.title(title)

plt.xlabel('Component 1')

plt.ylabel('Component 2')

plt.show()

def visualize\_embeddings(embeddings\_array, df\_test):

# Reduce dimensionality using PCA

embeddings\_pca = reduce\_dimensions(embeddings\_array, method='PCA')

plot\_embeddings(embeddings\_pca, df\_test['embedding\_euclidean\_isAnomaly'], 'PCA of Embeddings')

# Reduce dimensionality using t-SNE

embeddings\_tsne = reduce\_dimensions(embeddings\_array, method='t-SNE')

plot\_embeddings(embeddings\_tsne, df\_test['embedding\_euclidean\_isAnomaly'], 't-SNE of Embeddings')

def main(df):

# Preprocess data

features = preprocess\_data(df)

labels = df['isFraud']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split\_data(features, labels)

# Scale features

X\_train\_scaled, X\_test\_scaled = scale\_features(X\_train, X\_test)

# Handle imbalanced data

X\_train\_smote, y\_train\_smote = handle\_imbalanced\_data(X\_train\_scaled, y\_train)

# Train Random Forest model

rf\_model = train\_random\_forest(X\_train\_smote, y\_train\_smote)

# Evaluate Random Forest model

rf\_predictions = rf\_model.predict(X\_test\_scaled)

df\_test = df.loc[X\_test.index]

df\_test['rf\_isAnomaly'] = rf\_predictions

rf\_metrics = calculate\_metrics(y\_test, rf\_predictions)

# Process embeddings

embeddings\_array = process\_embeddings(df\_test)

normalized\_embeddings = normalize\_embeddings(embeddings\_array)

# Calculate dissimilarity

cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix = calculate\_dissimilarity(normalized\_embeddings)

# Detect anomalies using cosine dissimilarity

mean\_cosine = np.mean(cosine\_dissimilarity\_matrix)

std\_cosine = np.std(cosine\_dissimilarity\_matrix)

threshold\_cosine = mean\_cosine + 1.5 \* std\_cosine

cosine\_anomalies = find\_anomalies(cosine\_dissimilarity\_matrix, threshold\_cosine)

add\_anomaly\_column(df\_test, cosine\_anomalies, 'embedding\_cosine\_isAnomaly')

# Detect anomalies using Euclidean distance

mean\_euclidean = np.mean(euclidean\_distance\_matrix)

std\_euclidean = np.std(euclidean\_distance\_matrix)

threshold\_euclidean = mean\_euclidean + 1.5 \* std\_euclidean

euclidean\_anomalies = find\_anomalies(euclidean\_distance\_matrix, threshold\_euclidean)

add\_anomaly\_column(df\_test, euclidean\_anomalies, 'embedding\_euclidean\_isAnomaly')

# Visualize embeddings

visualize\_embeddings(normalized\_embeddings, df\_test)

# Evaluate embedding-based anomaly detection

embedding\_predictions = df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly']

embedding\_metrics = calculate\_metrics(y\_test, embedding\_predictions)

euclidean\_predictions = df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly']

euclidean\_metrics = calculate\_metrics(y\_test, euclidean\_predictions)

# Compare anomalies

comparison = compare\_anomalies(df\_test, X\_test, rf\_predictions)

# Generate summary report

generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison)

# Save results

df\_test.to\_csv('transactions\_with\_anomalies.csv', index=False)

if \_\_name\_\_ == "\_\_main\_\_":

main(df\_test)

"""### 0.2 weight"""

import pandas as pd

import numpy as np

import datetime

import os

import concurrent.futures

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from scipy.spatial.distance import pdist, squareform

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

import torch

from transformers import BertTokenizer, BertModel

# Load BERT model and tokenizer

bert\_model\_name = "bert-base-uncased"

tokenizer = BertTokenizer.from\_pretrained(bert\_model\_name)

bert\_model = BertModel.from\_pretrained(bert\_model\_name)

def preprocess\_data(df):

numerical\_columns = ['amount', 'avgTransactionAmount', 'timeStep', 'merchantName', 'customerId', 'IslargeTransaction', 'countForCustomerSameTime']

features = df[numerical\_columns].copy()

# One-hot encoding

features = pd.concat([features, pd.get\_dummies(df['merchantCategory'])], axis=1)

features = pd.concat([features, pd.get\_dummies(df['customerGender'])], axis=1)

return features

def train\_test\_split\_data(features, labels):

return train\_test\_split(features, labels, test\_size=0.5, random\_state=42)

def scale\_features(X\_train, X\_test):

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

return X\_train\_scaled, X\_test\_scaled

def handle\_imbalanced\_data(X\_train\_scaled, y\_train):

smote = SMOTE(random\_state=42)

return smote.fit\_resample(X\_train\_scaled, y\_train)

def train\_random\_forest(X\_train\_smote, y\_train\_smote):

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_train\_smote, y\_train\_smote)

return rf\_model

def evaluate\_model(model, X\_test\_scaled, y\_test):

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

return {

"accuracy": accuracy\_score(y\_test, y\_pred),

"precision": precision\_score(y\_test, y\_pred),

"recall": recall\_score(y\_test, y\_pred),

"f1": f1\_score(y\_test, y\_pred),

"conf\_matrix": confusion\_matrix(y\_test, y\_pred)

}

def get\_bert\_embeddings(texts):

embeddings = []

for text in texts:

inputs = tokenizer(text, return\_tensors='pt', padding=True, truncation=True, max\_length=512)

with torch.no\_grad():

outputs = bert\_model(\*\*inputs)

# Use the [CLS] token's representation as the sentence embedding

cls\_embedding = outputs.last\_hidden\_state[:, 0, :].squeeze().numpy()

embeddings.append(cls\_embedding)

return embeddings

def process\_embeddings(df\_test):

if os.path.exists('bert\_embeddings.npy'):

embeddings\_array = np.load('bert\_embeddings.npy')

else:

df\_test = df\_test.fillna('').astype(str)

df\_test['combined'] = df\_test.apply(lambda row: ' '.join(row), axis=1)

combined\_texts = df\_test['combined'].tolist()

embeddings\_list = get\_bert\_embeddings(combined\_texts)

embeddings\_array = np.array([np.array(embedding).flatten() for embedding in embeddings\_list])

np.save('bert\_embeddings.npy', embeddings\_array)

return embeddings\_array

def normalize\_embeddings(embeddings\_array):

scaler = StandardScaler()

return scaler.fit\_transform(embeddings\_array)

def calculate\_dissimilarity(embeddings\_array):

cosine\_dissimilarity\_matrix = squareform(pdist(embeddings\_array, 'cosine'))

euclidean\_distance\_matrix = squareform(pdist(embeddings\_array, 'euclidean'))

return cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix

def find\_anomalies(distance\_matrix, threshold):

anomaly\_indices = np.where(distance\_matrix > threshold)

anomaly\_transactions = set()

for i, j in zip(\*anomaly\_indices):

if i != j:

anomaly\_transactions.add(i)

anomaly\_transactions.add(j)

return anomaly\_transactions

def add\_anomaly\_column(df\_test, anomaly\_indices, column\_name):

index\_mapping = df\_test.index.tolist()

mapped\_anomaly\_indices = [index\_mapping[i] for i in anomaly\_indices]

df\_test[column\_name] = 0

df\_test.loc[mapped\_anomaly\_indices, column\_name] = 1

def calculate\_metrics(y\_true, y\_pred):

return {

"accuracy": accuracy\_score(y\_true, y\_pred),

"precision": precision\_score(y\_true, y\_pred, zero\_division=0),

"recall": recall\_score(y\_true, y\_pred, zero\_division=0),

"f1": f1\_score(y\_true, y\_pred, zero\_division=0)

}

def compare\_anomalies(df\_test, X\_test, rf\_predictions):

embedding\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'] == 1].index)

euclidean\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'] == 1].index)

rf\_anomalies = set(X\_test.index[rf\_predictions == 1])

matched\_anomalies = embedding\_anomalies.intersection(rf\_anomalies)

unique\_to\_embeddings = embedding\_anomalies - rf\_anomalies

unique\_to\_rf = rf\_anomalies - embedding\_anomalies

unique\_to\_euclidean = euclidean\_anomalies - rf\_anomalies - embedding\_anomalies

return {

"matched\_anomalies": len(matched\_anomalies),

"unique\_to\_embeddings": len(unique\_to\_embeddings),

"unique\_to\_rf": len(unique\_to\_rf),

"unique\_to\_euclidean": len(unique\_to\_euclidean)

}

def generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison):

summary = {

"Random Forest Approach": {\*\*rf\_metrics, "Unique Anomalies": comparison.get("unique\_to\_rf", 0)},

"Embedding-Cosine Approach": {\*\*embedding\_metrics, "Unique Anomalies": comparison.get("unique\_to\_embeddings", 0)},

"Embedding-Euclidean Approach": {\*\*euclidean\_metrics, "Unique Anomalies": comparison.get("unique\_to\_euclidean", 0)},

"Matched Anomalies": comparison.get("matched\_anomalies", 0)

}

# Debugging output to check the structure of summary

print("Summary dictionary structure:")

for approach, metrics in summary.items():

if isinstance(metrics, dict):

print(f"{approach}:")

for metric, value in metrics.items():

print(f" {metric}: {value}")

else:

print(f"{approach}: {metrics}")

print()

# Functions for dimensionality reduction and visualization

def reduce\_dimensions(embeddings\_array, method='PCA', n\_components=2):

if method == 'PCA':

reducer = PCA(n\_components=n\_components)

elif method == 't-SNE':

reducer = TSNE(n\_components=n\_components, random\_state=42)

else:

raise ValueError("Method should be either 'PCA' or 't-SNE'")

return reducer.fit\_transform(embeddings\_array)

def plot\_embeddings(embeddings\_reduced, labels, title):

plt.figure(figsize=(10, 8))

scatter = plt.scatter(embeddings\_reduced[:, 0], embeddings\_reduced[:, 1], c=labels, cmap='viridis', alpha=0.6)

plt.colorbar(scatter, label='Anomaly Status')

plt.title(title)

plt.xlabel('Component 1')

plt.ylabel('Component 2')

plt.show()

def visualize\_embeddings(embeddings\_array, df\_test):

# Reduce dimensionality using PCA

embeddings\_pca = reduce\_dimensions(embeddings\_array, method='PCA')

plot\_embeddings(embeddings\_pca, df\_test['embedding\_euclidean\_isAnomaly'], 'PCA of Embeddings')

# Reduce dimensionality using t-SNE

embeddings\_tsne = reduce\_dimensions(embeddings\_array, method='t-SNE')

plot\_embeddings(embeddings\_tsne, df\_test['embedding\_euclidean\_isAnomaly'], 't-SNE of Embeddings')

def main(df):

# Preprocess data

features = preprocess\_data(df)

labels = df['isFraud']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split\_data(features, labels)

# Scale features

X\_train\_scaled, X\_test\_scaled = scale\_features(X\_train, X\_test)

# Handle imbalanced data

X\_train\_smote, y\_train\_smote = handle\_imbalanced\_data(X\_train\_scaled, y\_train)

# Train Random Forest model

rf\_model = train\_random\_forest(X\_train\_smote, y\_train\_smote)

# Evaluate Random Forest model

rf\_predictions = rf\_model.predict(X\_test\_scaled)

df\_test = df.loc[X\_test.index]

df\_test['rf\_isAnomaly'] = rf\_predictions

rf\_metrics = calculate\_metrics(y\_test, rf\_predictions)

# Process embeddings

embeddings\_array = process\_embeddings(df\_test)

normalized\_embeddings = normalize\_embeddings(embeddings\_array)

# Calculate dissimilarity

cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix = calculate\_dissimilarity(normalized\_embeddings)

# Detect anomalies using cosine dissimilarity

mean\_cosine = np.mean(cosine\_dissimilarity\_matrix)

std\_cosine = np.std(cosine\_dissimilarity\_matrix)

threshold\_cosine = mean\_cosine + 1.5 \* std\_cosine

cosine\_anomalies = find\_anomalies(cosine\_dissimilarity\_matrix, threshold\_cosine)

add\_anomaly\_column(df\_test, cosine\_anomalies, 'embedding\_cosine\_isAnomaly')

# Detect anomalies using Euclidean distance

mean\_euclidean = np.mean(euclidean\_distance\_matrix)

std\_euclidean = np.std(euclidean\_distance\_matrix)

threshold\_euclidean = mean\_euclidean + 1.5 \* std\_euclidean

euclidean\_anomalies = find\_anomalies(euclidean\_distance\_matrix, threshold\_euclidean)

add\_anomaly\_column(df\_test, euclidean\_anomalies, 'embedding\_euclidean\_isAnomaly')

# Visualize embeddings

visualize\_embeddings(normalized\_embeddings, df\_test)

# Evaluate embedding-based anomaly detection

embedding\_predictions = df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly']

embedding\_metrics = calculate\_metrics(y\_test, embedding\_predictions)

euclidean\_predictions = df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly']

euclidean\_metrics = calculate\_metrics(y\_test, euclidean\_predictions)

# Compare anomalies

comparison = compare\_anomalies(df\_test, X\_test, rf\_predictions)

# Generate summary report

generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison)

# Save results

df\_test.to\_csv('transactions\_with\_anomalies.csv', index=False)

if \_\_name\_\_ == "\_\_main\_\_":

# Assuming df\_test is already loaded

main(df\_test)

"""## Experinment: Flag anomalous transactions using OPEN AI embedings

### 0.5 weight

"""

import pandas as pd

import numpy as np

import datetime

import os

import concurrent.futures

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from scipy.spatial.distance import pdist, squareform

import openai

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

from sklearn.preprocessing import StandardScaler

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

from scipy.spatial.distance import pdist, squareform

import datetime

# Set OpenAI API Key

os.environ["OPENAI\_API\_KEY"] = # KEY HERE

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

def preprocess\_data(df):

numerical\_columns = ['amount', 'avgTransactionAmount', 'timeStep', 'merchantName', 'customerId', 'IslargeTransaction', 'countForCustomerSameTime']

features = df[numerical\_columns].copy()

# One-hot encoding

features = pd.concat([features, pd.get\_dummies(df['merchantCategory'])], axis=1)

features = pd.concat([features, pd.get\_dummies(df['customerGender'])], axis=1)

return features

def train\_test\_split\_data(features, labels):

return train\_test\_split(features, labels, test\_size=0.5, random\_state=42)

def scale\_features(X\_train, X\_test):

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

return X\_train\_scaled, X\_test\_scaled

def handle\_imbalanced\_data(X\_train\_scaled, y\_train):

smote = SMOTE(random\_state=42)

return smote.fit\_resample(X\_train\_scaled, y\_train)

def train\_random\_forest(X\_train\_smote, y\_train\_smote):

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_train\_smote, y\_train\_smote)

return rf\_model

def evaluate\_model(model, X\_test\_scaled, y\_test):

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

return {

"accuracy": accuracy\_score(y\_test, y\_pred),

"precision": precision\_score(y\_test, y\_pred),

"recall": recall\_score(y\_test, y\_pred),

"f1": f1\_score(y\_test, y\_pred),

"conf\_matrix": confusion\_matrix(y\_test, y\_pred)

}

def get\_embeddings\_parallel(texts):

with concurrent.futures.ThreadPoolExecutor() as executor:

futures = [executor.submit(get\_single\_embedding, text) for text in texts]

return [future.result() for future in concurrent.futures.as\_completed(futures)]

def get\_single\_embedding(text):

response = openai.Embedding.create(model="text-embedding-ada-002", input=text)

return response['data'][0]['embedding']

def process\_embeddings(df\_test):

if os.path.exists('embeddings\_test\_ex1.npy'):

embeddings\_array = np.load('embeddings\_test\_ex1.npy')

else:

df\_test = df\_test.fillna('').astype(str)

df\_test['combined'] = df\_test.apply(lambda row: ' '.join(row), axis=1)

combined\_texts = df\_test['combined'].tolist()

embeddings\_list = get\_embeddings\_parallel(combined\_texts)

embeddings\_array = np.array([np.array(embedding).flatten() for embedding in embeddings\_list])

np.save('embeddings\_test\_ex1.npy', embeddings\_array)

return embeddings\_array

def normalize\_embeddings(embeddings\_array):

scaler = StandardScaler()

return scaler.fit\_transform(embeddings\_array)

def calculate\_dissimilarity(embeddings\_array):

cosine\_dissimilarity\_matrix = squareform(pdist(embeddings\_array, 'cosine'))

euclidean\_distance\_matrix = squareform(pdist(embeddings\_array, 'euclidean'))

return cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix

def find\_anomalies(distance\_matrix, threshold):

anomaly\_indices = np.where(distance\_matrix > threshold)

anomaly\_transactions = set()

for i, j in zip(\*anomaly\_indices):

if i != j:

anomaly\_transactions.add(i)

anomaly\_transactions.add(j)

return anomaly\_transactions

def add\_anomaly\_column(df\_test, anomaly\_indices, column\_name):

index\_mapping = df\_test.index.tolist()

mapped\_anomaly\_indices = [index\_mapping[i] for i in anomaly\_indices]

df\_test[column\_name] = 0

df\_test.loc[mapped\_anomaly\_indices, column\_name] = 1

def calculate\_metrics(y\_true, y\_pred):

return {

"accuracy": accuracy\_score(y\_true, y\_pred),

"precision": precision\_score(y\_true, y\_pred, zero\_division=0),

"recall": recall\_score(y\_true, y\_pred, zero\_division=0),

"f1": f1\_score(y\_true, y\_pred, zero\_division=0)

}

def compare\_anomalies(df\_test, X\_test, rf\_predictions):

embedding\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'] == 1].index)

euclidean\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'] == 1].index)

rf\_anomalies = set(X\_test.index[rf\_predictions == 1])

matched\_anomalies = embedding\_anomalies.intersection(rf\_anomalies)

unique\_to\_embeddings = embedding\_anomalies - rf\_anomalies

unique\_to\_rf = rf\_anomalies - embedding\_anomalies

unique\_to\_euclidean = euclidean\_anomalies - rf\_anomalies - embedding\_anomalies

return {

"matched\_anomalies": len(matched\_anomalies),

"unique\_to\_embeddings": len(unique\_to\_embeddings),

"unique\_to\_rf": len(unique\_to\_rf),

"unique\_to\_euclidean": len(unique\_to\_euclidean)

}

def generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison):

summary = {

"Random Forest Approach": {\*\*rf\_metrics, "Unique Anomalies": comparison.get("unique\_to\_rf", 0)},

"Embedding-Cosine Approach": {\*\*embedding\_metrics, "Unique Anomalies": comparison.get("unique\_to\_embeddings", 0)},

"Embedding-Euclidean Approach": {\*\*euclidean\_metrics, "Unique Anomalies": comparison.get("unique\_to\_euclidean", 0)},

"Matched Anomalies": comparison.get("matched\_anomalies", 0)

}

# Debugging output to check the structure of summary

print("Summary dictionary structure:")

for approach, metrics in summary.items():

if isinstance(metrics, dict):

print(f"{approach}:")

for metric, value in metrics.items():

print(f" {metric}: {value}")

else:

print(f"{approach}: {metrics}")

print()

# Functions for dimensionality reduction and visualization

def reduce\_dimensions(embeddings\_array, method='PCA', n\_components=2):

if method == 'PCA':

reducer = PCA(n\_components=n\_components)

elif method == 't-SNE':

reducer = TSNE(n\_components=n\_components, random\_state=42)

else:

raise ValueError("Method should be either 'PCA' or 't-SNE'")

return reducer.fit\_transform(embeddings\_array)

def plot\_embeddings(embeddings\_reduced, labels, title):

plt.figure(figsize=(10, 8))

scatter = plt.scatter(embeddings\_reduced[:, 0], embeddings\_reduced[:, 1], c=labels, cmap='viridis', alpha=0.6)

plt.colorbar(scatter, label='Anomaly Status')

plt.title(title)

plt.xlabel('Component 1')

plt.ylabel('Component 2')

plt.show()

def visualize\_embeddings(embeddings\_array, df\_test):

# Reduce dimensionality using PCA

embeddings\_pca = reduce\_dimensions(embeddings\_array, method='PCA')

plot\_embeddings(embeddings\_pca, df\_test['embedding\_euclidean\_isAnomaly'], 'PCA of Embeddings')

# Reduce dimensionality using t-SNE

embeddings\_tsne = reduce\_dimensions(embeddings\_array, method='t-SNE')

plot\_embeddings(embeddings\_tsne, df\_test['embedding\_euclidean\_isAnomaly'], 't-SNE of Embeddings')

def main(df):

# Preprocess data

features = preprocess\_data(df)

labels = df['isFraud']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split\_data(features, labels)

# Scale features

X\_train\_scaled, X\_test\_scaled = scale\_features(X\_train, X\_test)

# Handle imbalanced data

X\_train\_smote, y\_train\_smote = handle\_imbalanced\_data(X\_train\_scaled, y\_train)

# Train Random Forest model

rf\_model = train\_random\_forest(X\_train\_smote, y\_train\_smote)

# Evaluate Random Forest model

rf\_predictions = rf\_model.predict(X\_test\_scaled)

df\_test = df.loc[X\_test.index]

df\_test['rf\_isAnomaly'] = rf\_predictions

rf\_metrics = calculate\_metrics(y\_test, rf\_predictions)

# Process embeddings

embeddings\_array = process\_embeddings(df\_test)

normalized\_embeddings = normalize\_embeddings(embeddings\_array)

# Calculate dissimilarity

cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix = calculate\_dissimilarity(normalized\_embeddings)

# Detect anomalies using cosine dissimilarity

mean\_cosine = np.mean(cosine\_dissimilarity\_matrix)

std\_cosine = np.std(cosine\_dissimilarity\_matrix)

threshold\_cosine = mean\_cosine + 1.5 \* std\_cosine

cosine\_anomalies = find\_anomalies(cosine\_dissimilarity\_matrix, threshold\_cosine)

add\_anomaly\_column(df\_test, cosine\_anomalies, 'embedding\_cosine\_isAnomaly')

# Detect anomalies using Euclidean distance

mean\_euclidean = np.mean(euclidean\_distance\_matrix)

std\_euclidean = np.std(euclidean\_distance\_matrix)

threshold\_euclidean = mean\_euclidean + 1.5 \* std\_euclidean

euclidean\_anomalies = find\_anomalies(euclidean\_distance\_matrix, threshold\_euclidean)

add\_anomaly\_column(df\_test, euclidean\_anomalies, 'embedding\_euclidean\_isAnomaly')

# Visualize embeddings

visualize\_embeddings(normalized\_embeddings, df\_test)

# Evaluate embedding-based anomaly detection

embedding\_predictions = df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly']

embedding\_metrics = calculate\_metrics(y\_test, embedding\_predictions)

euclidean\_predictions = df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly']

euclidean\_metrics = calculate\_metrics(y\_test, euclidean\_predictions)

# Compare anomalies

comparison = compare\_anomalies(df\_test, X\_test, rf\_predictions)

# Generate summary report

generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison)

# Save results

df\_test.to\_csv('transactions\_with\_anomalies.csv', index=False)

if \_\_name\_\_ == "\_\_main\_\_":

main(df\_test)

"""### 0.2 weight"""

import pandas as pd

import numpy as np

import datetime

import os

import concurrent.futures

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

from imblearn.over\_sampling import SMOTE

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from scipy.spatial.distance import pdist, squareform

import openai

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE

from sklearn.preprocessing import StandardScaler

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

from scipy.spatial.distance import pdist, squareform

import datetime

# Set OpenAI API Key

os.environ["OPENAI\_API\_KEY"] = #api-key-here

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

def preprocess\_data(df):

numerical\_columns = ['amount', 'avgTransactionAmount', 'timeStep', 'merchantName', 'customerId', 'IslargeTransaction', 'countForCustomerSameTime']

features = df[numerical\_columns].copy()

# One-hot encoding

features = pd.concat([features, pd.get\_dummies(df['merchantCategory'])], axis=1)

features = pd.concat([features, pd.get\_dummies(df['customerGender'])], axis=1)

return features

def train\_test\_split\_data(features, labels):

return train\_test\_split(features, labels, test\_size=0.5, random\_state=42)

def scale\_features(X\_train, X\_test):

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

return X\_train\_scaled, X\_test\_scaled

def handle\_imbalanced\_data(X\_train\_scaled, y\_train):

smote = SMOTE(random\_state=42)

return smote.fit\_resample(X\_train\_scaled, y\_train)

def train\_random\_forest(X\_train\_smote, y\_train\_smote):

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_train\_smote, y\_train\_smote)

return rf\_model

def evaluate\_model(model, X\_test\_scaled, y\_test):

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

return {

"accuracy": accuracy\_score(y\_test, y\_pred),

"precision": precision\_score(y\_test, y\_pred),

"recall": recall\_score(y\_test, y\_pred),

"f1": f1\_score(y\_test, y\_pred),

"conf\_matrix": confusion\_matrix(y\_test, y\_pred)

}

def get\_embeddings\_parallel(texts):

with concurrent.futures.ThreadPoolExecutor() as executor:

futures = [executor.submit(get\_single\_embedding, text) for text in texts]

return [future.result() for future in concurrent.futures.as\_completed(futures)]

def get\_single\_embedding(text):

response = openai.Embedding.create(model="text-embedding-ada-002", input=text)

return response['data'][0]['embedding']

def process\_embeddings(df\_test):

if os.path.exists('embeddings\_test.npy'):

embeddings\_array = np.load('embeddings\_test.npy')

else:

df\_test = df\_test.fillna('').astype(str)

df\_test['combined'] = df\_test.apply(lambda row: ' '.join(row), axis=1)

combined\_texts = df\_test['combined'].tolist()

embeddings\_list = get\_embeddings\_parallel(combined\_texts)

embeddings\_array = np.array([np.array(embedding).flatten() for embedding in embeddings\_list])

np.save('embeddings\_test.npy', embeddings\_array)

return embeddings\_array

def normalize\_embeddings(embeddings\_array):

scaler = StandardScaler()

return scaler.fit\_transform(embeddings\_array)

def calculate\_dissimilarity(embeddings\_array):

cosine\_dissimilarity\_matrix = squareform(pdist(embeddings\_array, 'cosine'))

euclidean\_distance\_matrix = squareform(pdist(embeddings\_array, 'euclidean'))

return cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix

def find\_anomalies(distance\_matrix, threshold):

anomaly\_indices = np.where(distance\_matrix > threshold)

anomaly\_transactions = set()

for i, j in zip(\*anomaly\_indices):

if i != j:

anomaly\_transactions.add(i)

anomaly\_transactions.add(j)

return anomaly\_transactions

def add\_anomaly\_column(df\_test, anomaly\_indices, column\_name):

index\_mapping = df\_test.index.tolist()

mapped\_anomaly\_indices = [index\_mapping[i] for i in anomaly\_indices]

df\_test[column\_name] = 0

df\_test.loc[mapped\_anomaly\_indices, column\_name] = 1

def calculate\_metrics(y\_true, y\_pred):

return {

"accuracy": accuracy\_score(y\_true, y\_pred),

"precision": precision\_score(y\_true, y\_pred, zero\_division=0),

"recall": recall\_score(y\_true, y\_pred, zero\_division=0),

"f1": f1\_score(y\_true, y\_pred, zero\_division=0)

}

def compare\_anomalies(df\_test, X\_test, rf\_predictions):

embedding\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly'] == 1].index)

euclidean\_anomalies = set(df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'][df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly'] == 1].index)

rf\_anomalies = set(X\_test.index[rf\_predictions == 1])

matched\_anomalies = embedding\_anomalies.intersection(rf\_anomalies)

unique\_to\_embeddings = embedding\_anomalies - rf\_anomalies

unique\_to\_rf = rf\_anomalies - embedding\_anomalies

unique\_to\_euclidean = euclidean\_anomalies - rf\_anomalies - embedding\_anomalies

return {

"matched\_anomalies": len(matched\_anomalies),

"unique\_to\_embeddings": len(unique\_to\_embeddings),

"unique\_to\_rf": len(unique\_to\_rf),

"unique\_to\_euclidean": len(unique\_to\_euclidean)

}

def generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison):

summary = {

"Random Forest Approach": {\*\*rf\_metrics, "Unique Anomalies": comparison.get("unique\_to\_rf", 0)},

"Embedding-Cosine Approach": {\*\*embedding\_metrics, "Unique Anomalies": comparison.get("unique\_to\_embeddings", 0)},

"Embedding-Euclidean Approach": {\*\*euclidean\_metrics, "Unique Anomalies": comparison.get("unique\_to\_euclidean", 0)},

"Matched Anomalies": comparison.get("matched\_anomalies", 0)

}

# Debugging output to check the structure of summary

print("Summary dictionary structure:")

for approach, metrics in summary.items():

if isinstance(metrics, dict):

print(f"{approach}:")

for metric, value in metrics.items():

print(f" {metric}: {value}")

else:

print(f"{approach}: {metrics}")

print()

# Functions for dimensionality reduction and visualization

def reduce\_dimensions(embeddings\_array, method='PCA', n\_components=2):

if method == 'PCA':

reducer = PCA(n\_components=n\_components)

elif method == 't-SNE':

reducer = TSNE(n\_components=n\_components, random\_state=42)

else:

raise ValueError("Method should be either 'PCA' or 't-SNE'")

return reducer.fit\_transform(embeddings\_array)

def plot\_embeddings(embeddings\_reduced, labels, title):

plt.figure(figsize=(10, 8))

scatter = plt.scatter(embeddings\_reduced[:, 0], embeddings\_reduced[:, 1], c=labels, cmap='viridis', alpha=0.6)

plt.colorbar(scatter, label='Anomaly Status')

plt.title(title)

plt.xlabel('Component 1')

plt.ylabel('Component 2')

plt.show()

def visualize\_embeddings(embeddings\_array, df\_test):

# Reduce dimensionality using PCA

embeddings\_pca = reduce\_dimensions(embeddings\_array, method='PCA')

plot\_embeddings(embeddings\_pca, df\_test['embedding\_euclidean\_isAnomaly'], 'PCA of Embeddings')

# Reduce dimensionality using t-SNE

embeddings\_tsne = reduce\_dimensions(embeddings\_array, method='t-SNE')

plot\_embeddings(embeddings\_tsne, df\_test['embedding\_euclidean\_isAnomaly'], 't-SNE of Embeddings')

def main(df):

# Preprocess data

features = preprocess\_data(df)

labels = df['isFraud']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split\_data(features, labels)

# Scale features

X\_train\_scaled, X\_test\_scaled = scale\_features(X\_train, X\_test)

# Handle imbalanced data

X\_train\_smote, y\_train\_smote = handle\_imbalanced\_data(X\_train\_scaled, y\_train)

# Train Random Forest model

rf\_model = train\_random\_forest(X\_train\_smote, y\_train\_smote)

# Evaluate Random Forest model

rf\_predictions = rf\_model.predict(X\_test\_scaled)

df\_test = df.loc[X\_test.index]

df\_test['rf\_isAnomaly'] = rf\_predictions

rf\_metrics = calculate\_metrics(y\_test, rf\_predictions)

# Process embeddings

embeddings\_array = process\_embeddings(df\_test)

normalized\_embeddings = normalize\_embeddings(embeddings\_array)

# Calculate dissimilarity

cosine\_dissimilarity\_matrix, euclidean\_distance\_matrix = calculate\_dissimilarity(normalized\_embeddings)

# Detect anomalies using cosine dissimilarity

mean\_cosine = np.mean(cosine\_dissimilarity\_matrix)

std\_cosine = np.std(cosine\_dissimilarity\_matrix)

threshold\_cosine = mean\_cosine + 1.5 \* std\_cosine

cosine\_anomalies = find\_anomalies(cosine\_dissimilarity\_matrix, threshold\_cosine)

add\_anomaly\_column(df\_test, cosine\_anomalies, 'embedding\_cosine\_isAnomaly')

# Detect anomalies using Euclidean distance

mean\_euclidean = np.mean(euclidean\_distance\_matrix)

std\_euclidean = np.std(euclidean\_distance\_matrix)

threshold\_euclidean = mean\_euclidean + 1.5 \* std\_euclidean

euclidean\_anomalies = find\_anomalies(euclidean\_distance\_matrix, threshold\_euclidean)

add\_anomaly\_column(df\_test, euclidean\_anomalies, 'embedding\_euclidean\_isAnomaly')

# Visualize embeddings

visualize\_embeddings(normalized\_embeddings, df\_test)

# Evaluate embedding-based anomaly detection

embedding\_predictions = df\_test.loc[X\_test.index, 'embedding\_cosine\_isAnomaly']

embedding\_metrics = calculate\_metrics(y\_test, embedding\_predictions)

euclidean\_predictions = df\_test.loc[X\_test.index, 'embedding\_euclidean\_isAnomaly']

euclidean\_metrics = calculate\_metrics(y\_test, euclidean\_predictions)

# Compare anomalies

comparison = compare\_anomalies(df\_test, X\_test, rf\_predictions)

# Generate summary report

generate\_summary\_report(rf\_metrics, embedding\_metrics, euclidean\_metrics, comparison)

# Save results

df\_test.to\_csv('transactions\_with\_anomalies.csv', index=False)

if \_\_name\_\_ == "\_\_main\_\_":

main(df\_test)

import numpy as np

import pandas as pd

from sentence\_transformers import SentenceTransformer

import faiss

def load\_model(model\_name: str = 'paraphrase-MiniLM-L6-v2') -> SentenceTransformer:

"""

Load and return the SentenceTransformer model

"""

return SentenceTransformer(model\_name)

def generate\_embeddings(df: pd.DataFrame, column: str, model: SentenceTransformer) -> pd.DataFrame:

"""

Generate embeddings for a specific column in the DataFrame

"""

df['embeddings'] = df[column].apply(lambda x: model.encode(x))

return df

def initialize\_faiss\_index(embeddings: np.ndarray) -> faiss.IndexFlatL2:

"""

Initialize and return a FAISS index

"""

dimension = embeddings.shape[1]

index = faiss.IndexFlatL2(dimension)

index.add(embeddings)

return index

def save\_faiss\_index(index: faiss.IndexFlatL2, filepath: str):

"""

Save the FAISS index to disk

"""

faiss.write\_index(index, filepath)

def search\_similar\_transactions(query: str, model: SentenceTransformer, index: faiss.IndexFlatL2,

df: pd.DataFrame, k: int = 3) -> dict:

"""

Search for similar transactions based on the query and return results

"""

query\_embedding = model.encode(query)

D, I = index.search(np.array([query\_embedding]), k)

# Collect results

top\_k\_results = []

for idx, dist in zip(I[0], D[0]):

result = {

'transaction\_behavior': df.iloc[idx]['transaction\_behavior'],

'similarity\_score': 1 / (1 + dist) # Convert distance to similarity score

}

top\_k\_results.append(result)

# Extract the last 2 words of the highest similarity transaction

highest\_similarity\_transaction = top\_k\_results[0]['transaction\_behavior']

prediction = ' '.join(highest\_similarity\_transaction.split()[-2:])

return {

'top\_k\_results': top\_k\_results,

'prediction': prediction

}

def display\_results(results: dict):

"""

Display the search results and prediction

"""

print("Top K Results:")

for i, res in enumerate(results['top\_k\_results']):

print(f"Result {i+1}:")

print(f"Transaction Behavior: {res['transaction\_behavior']}")

print(f"Similarity Score: {res['similarity\_score']:.4f}")

print("-" \* 80)

print("Prediction :")

print("This transaction is a ", results['prediction'])

if \_\_name\_\_ == "\_\_main\_\_":

# Load model

model = load\_model()

# Generate embeddings and add to DataFrame

df\_processed = generate\_embeddings(df\_processed, 'transaction\_behavior', model)

# Prepare embeddings and initialize FAISS index

embeddings = np.vstack(df\_processed['embeddings'].values)

index = initialize\_faiss\_index(embeddings)

# SAVE the index

save\_faiss\_index(index, "faiss\_index\_2.index")

df\_processed[['transaction\_behavior', 'embeddings']]

# Perform search with a query

query2 = "A uncommon transaction .. from a Teen male customer 976805632 .. category es\_travel with large amount"

results = search\_similar\_transactions(query2, model, index, df\_subset)

# Display the search results

display\_results(results)

# Perform search with a query

query2 = "A uncommon transaction .. Child female customer 1035222482 from merchant 348934600 to category es\_transportation with intermediate"

results = search\_similar\_transactions(query2, model, index, df\_subset)

# Display the search results

display\_results(results)

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

import seaborn as sns

# Perform PCA for dimensionality reduction to 2D

pca = PCA(n\_components=50)

pca\_result = pca.fit\_transform(embeddings)

# Perform t-SNE for further dimensionality reduction to 2D

tsne = TSNE(n\_components=2, perplexity=30, n\_iter=300)

tsne\_result = tsne.fit\_transform(pca\_result)

# Plotting the t-SNE result

plt.figure(figsize=(10, 8))

sns.scatterplot(x=tsne\_result[:, 0], y=tsne\_result[:, 1], alpha=0.7)

plt.title('t-SNE visualization of text embeddings')

plt.xlabel('t-SNE Component 1')

plt.ylabel('t-SNE Component 2')

plt.show()

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

def evaluate\_model(df: pd.DataFrame, model: SentenceTransformer, index: faiss.IndexFlatL2, queries: list, true\_labels: list) -> dict:

"""

Evaluate the model's performance on a set of queries

"""

predictions = []

actual\_labels = []

for query, true\_label in zip(queries, true\_labels):

results = search\_similar\_transactions(query, model, index, df)

predicted\_label = results['prediction']

# Collect actual and predicted labels

actual\_labels.append(true\_label)

predictions.append(predicted\_label)

# Calculate metrics

accuracy = accuracy\_score(actual\_labels, predictions)

precision = precision\_score(actual\_labels, predictions, average='binary', pos\_label='fraud transaction') # Adjust pos\_label as necessary

recall = recall\_score(actual\_labels, predictions, average='binary', pos\_label='fraud transaction')

f1 = f1\_score(actual\_labels, predictions, average='binary', pos\_label='fraud transaction')

# Create DataFrame with predictions and actual labels

predictions\_df = pd.DataFrame({

'transaction\_behavior': queries,

'actual': actual\_labels,

'prediction': predictions

})

return {

'accuracy': accuracy,

'precision': precision,

'recall': recall,

'f1\_score': f1,

'classification\_report': classification\_report(actual\_labels, predictions),

'predictions\_df': predictions\_df

}

# Pathto index file

index\_file\_path = 'faiss\_index\_2.index'

# Load the FAISS index

index = faiss.read\_index(index\_file\_path)

# Verify that the index was loaded correctly

print(f"Loaded FAISS index with type: {type(index)}")

if \_\_name\_\_ == "\_\_main\_\_":

model = load\_model()

print(model)

print(index)

queries = df\_processed['transaction\_behavior'].tolist()

true\_labels = df\_processed['bhv\_isFraud'].tolist()

# Evaluate the model

metrics = evaluate\_model(df\_processed, model, index, queries, true\_labels)

# Print evaluation metrics

print("Evaluation Metrics:")

print(f"Accuracy: {metrics['accuracy']:.4f}")

print(f"Precision: {metrics['precision']:.4f}")

print(f"Recall: {metrics['recall']:.4f}")

print(f"F1 Score: {metrics['f1\_score']:.4f}")

print("Classification Report:")

print(metrics['classification\_report'])

print(metrics['predictions\_df']['prediction'])

# Append predictions to the sample DataFrame

metrics['predictions\_df']['prediction']

metrics['predictions\_df']['actual']

metrics['predictions\_df']

import numpy as np

import pandas as pd

from sklearn.metrics import precision\_score, recall\_score, average\_precision\_score, ndcg\_score

import matplotlib.pyplot as plt

import seaborn as sns

def evaluate\_similarity\_search\_advanced(df: pd.DataFrame, model: SentenceTransformer, index: faiss.IndexFlatL2, queries: list, true\_labels: list, k: int = 3) -> dict:

"""

Evaluate the similarity search model's performance on a set of queries.

"""

all\_relevant = []

all\_scores = []

predictions = []

for query, true\_label in zip(queries, true\_labels):

results = search\_similar\_transactions(query, model, index, df, k)

# Extract results

top\_k\_results = results['top\_k\_results']

predicted\_label = results['prediction']

# Collect the predicted label

predictions.append(predicted\_label)

# Collect relevant items and their scores

relevant\_items = [1 if true\_label in result['transaction\_behavior'] else 0 for result in top\_k\_results]

all\_scores.extend([result['similarity\_score'] for result in top\_k\_results])

all\_relevant.extend(relevant\_items)

# Precision at K

precision\_at\_k = np.mean([sum([1 if item == 1 else 0 for item in relevant\_items]) / k for relevant\_items in [all\_relevant]])

# Recall at K

recall\_at\_k = np.mean([sum(all\_relevant) / len(true\_labels)])

# Mean Average Precision (MAP)

average\_precision = average\_precision\_score(all\_relevant, all\_scores)

# NDCG

if len(all\_relevant) > 0:

ndcg = ndcg\_score([all\_relevant], [all\_scores], k=k)

else:

ndcg = 0.0

return {

'precision\_at\_k': precision\_at\_k,

'recall\_at\_k': recall\_at\_k,

'average\_precision': average\_precision,

'ndcg': ndcg

}

if \_\_name\_\_ == "\_\_main\_\_":

model = load\_model()

# Prepare the sample

sample\_df = df\_processed.sample(n=10, random\_state=42)

queries = sample\_df['transaction\_behavior'].str.slice(0, 120).tolist()

true\_labels = sample\_df['bhv\_isFraud'].tolist()

# Initialize FAISS index and generate embeddings

embeddings = np.vstack(df\_processed['embeddings'].values)

index = initialize\_faiss\_index(embeddings)

# Evaluate the model

metrics = evaluate\_similarity\_search\_advanced(df\_processed, model, index, queries, true\_labels, k=3)

# Print evaluation metrics

print("Evaluation Metrics:")

print(f"Average Precision: {metrics['average\_precision']:.4f}")

print(f"NDCG: {metrics['ndcg']:.4f}")

# Metrics values

metrics\_dict = {

'Average Precision (AP)': metrics['average\_precision'],

'Normalized Discounted Cumulative Gain (NDCG)': metrics['ndcg']

}

# Create a DataFrame for plotting

metrics\_df = pd.DataFrame(list(metrics\_dict.items()), columns=['Metric', 'Score'])

# Plotting the metrics

plt.figure(figsize=(10, 6))

# Bar plot

sns.barplot(x='Metric', y='Score', data=metrics\_df, palette='viridis')

# Add titles and labels

plt.title('Evaluation Metrics')

plt.xlabel('Metric')

plt.ylabel('Score')

plt.ylim(0, 1)

plt.xticks(rotation=45, ha='right')

# Add value annotations

for index, row in metrics\_df.iterrows():

plt.text(index, row['Score'] + 0.02, f'{row["Score"]:.4f}', ha='center')

plt.tight\_layout()

plt.show()

"""## More Generative"""

import numpy as np

from sentence\_transformers import SentenceTransformer

import faiss

def load\_model(model\_name: str = 'paraphrase-MiniLM-L6-v2') -> SentenceTransformer:

"""

Load and return the SentenceTransformer model.

"""

return SentenceTransformer(model\_name)

def generate\_embeddings(df: pd.DataFrame, column: str, model: SentenceTransformer) -> pd.DataFrame:

"""

Generate embeddings for a specific column in the DataFrame.

"""

df['embeddings'] = df[column].apply(lambda x: model.encode(x))

return df

def initialize\_faiss\_index(embeddings: np.ndarray) -> faiss.IndexFlatL2:

"""

Initialize and return a FAISS index.

"""

dimension = embeddings.shape[1]

index = faiss.IndexFlatL2(dimension)

index.add(embeddings)

return index

def search\_and\_generate\_response(query: str, model: SentenceTransformer, index: faiss.IndexFlatL2,

df: pd.DataFrame, api\_key: str, k: int = 3, high\_priority\_terms: list = None) -> dict:

"""

Search for similar transactions and generate a response based on the query and high-priority terms.

"""

if high\_priority\_terms is None:

high\_priority\_terms = []

query\_embedding = model.encode(query)

D, I = index.search(np.array([query\_embedding]), k)

# Adjust the similarity scores based on high-priority terms

# Customization

results = []

for idx, dist in zip(I[0], D[0]):

transaction\_behavior = df.iloc[idx]['transaction\_behavior']

similarity\_score = 1 / (1 + dist) # Convert distance to similarity score

# Check for high-priority terms in the result

priority\_score = sum(term in transaction\_behavior for term in high\_priority\_terms)

adjusted\_score = similarity\_score + priority\_score

result = {

'transaction\_behavior': transaction\_behavior,

'similarity\_score': adjusted\_score

}

results.append(result)

# Sort results based on the adjusted score

results = sorted(results, key=lambda x: x['similarity\_score'], reverse=True)

# Generate a response based on the top results

response = generate\_response(results, query, api\_key)

# Extract the highest similarity transaction

highest\_similarity\_transaction = results[0]['transaction\_behavior']

prediction = ' '.join(highest\_similarity\_transaction.split()[-2:])

return {

'top\_k\_results': results,

'prediction': prediction,

'response': response

}

import openai

def generate\_response(retrieved\_docs: list, query: str, api\_key: str) -> str:

"""

Generate a response using a generative model based on the retrieved documents.

"""

openai.api\_key = api\_key

# Format the prompt for a chat-based model

messages = [

{"role": "system", "content": "You are a helpful assistant."},

{"role": "user", "content": f"Query: {query}"},

{"role": "user", "content": "Documents:"},

{"role": "user", "content": "\n".join(doc['transaction\_behavior'] for doc in retrieved\_docs)}

]

response = openai.ChatCompletion.create(

model="gpt-3.5-turbo",

messages=messages,

max\_tokens=150,

temperature=0.7

)

return response.choices[0].message['content'].strip()

if \_\_name\_\_ == "\_\_main\_\_":

model = load\_model()

embeddings = np.vstack(df\_processed['embeddings'].values)

index = initialize\_faiss\_index(embeddings)

# User Quyery

query = "what is the type of A recurring transaction originated from a Teen male customer 180936571 from merchant 980657600 to category es\_sportsandtoys with considerable amount is categorized as?"

# Define words which needs high attention

high\_priority\_terms = ["Teen", "considerable"]

# For security purposes removed api key from the code. To generate the response assign an open\_api\_key to api\_key variable below

api\_key = #api-key-here

# Perform search with a query and priority terms

results = search\_and\_generate\_response(query, model, index, df\_processed, api\_key, k=3, high\_priority\_terms=high\_priority\_terms)

# Display the search results

print("Top K Results:")

for i, res in enumerate(results['top\_k\_results']):

print(f"Result {i+1}:")

print(f"Transaction Behavior: {res['transaction\_behavior']}")

print(f"Adjusted Similarity Score: {res['similarity\_score']:.4f}")

print("-" \* 80)

print("Prediction:")

print(f"This transaction is a {results['prediction']}")

print("Generated Response:")

print(results['response'])