
**International Centre for Education and Research (ICER)**
**VIT-Bangalore**

`

# Walmart Transaction Fraud Detection System

**CS6510 – PROJECT 1**

**REPORT**

***Submitted by***

**S.P.Dhinesh– 23MSP3032**

***In partial fulfilment for the award of the degree of***

**POST GRADUATE PROGRAMME**

**INTERNATIONAL CENTRE FOR HIGHER EDUCATION AND RESEARCH**

**VIT BANGALORE**

**DECEMBER, 2023**


**International Centre for Education and Research (ICER)**
**VIT-Bangalore**

# BONAFIDE CERTIFICATE

Certified that this project report **"Walmart Transaction Fraud Detection System "** is the bonafide record of work done by **"S.P.Dhinesh – 23MSP3032"** who carried out the project work under my supervision.

| **Signature of the Supervisor** | **Signature of Director** |
|---|---|
| **Dr. Pitchumani Angayarkanni S** | **Prof. Prema M** |
| **Professor,** | **Director,** |
| ICER | ICER |
| VIT Bangalore | VIT Bangalore. |

**Evaluation Date:**


**International Centre for Education and Research (ICER)**
**VIT-Bangalore**

# ACKNOWLEDGEMENT

I express my sincere gratitude to our director of ICER **Prof. Prema M.** for their support and for providing the required facilities for carrying out this study.

I wish to thank my faculty supervisor(s), **Dr. Pitchumani Angayarkanni S, Professor**, ICER for extending help and encouragement throughout the project. Without his/her continuous guidance and persistent help, this project would not have been a success for me.

I am grateful to all the members of ICER, my beloved parents, and friends for extending the support, who helped us to overcome obstacles in the study.

# TABLE OF CONTENTS

# ABSTRACT

Walmart's patent, "Self-Calibrating Fraud Detection," describes a system and method for enhanced fraud detection in automated electronic transactions, utilizing machine learning to analyze previous payment transfer data and prevent fraudulent transactions by comparing generated fraud detection values to a threshold.Machine learning models are trained and validated using this structured dataset as the basis.A variety of machine learning techniques are used to build a strong fraud detection model, such as decision trees, neural networks, and anomaly detection. Through constant adaptation to changing fraud patterns and gradual accuracy improvement, the models are trained on previous data. Integrating these models into Walmart's transaction processing systems makes real-time monitoring easier. To sum up, this study provides a thorough foundation for transaction fraud.In summary, this article offers a thorough framework for Walmart transaction fraud detection that makes use of cutting-edge analytical tools to improve real-time monitoring capabilities, accuracy, and adaptability. The suggested solution puts Walmart at the forefront of applying cutting-edge technology to counteract changing risks in the retail industry while also bolstering the security of financial transactions.

**Keywords:**Transaction Fraud Detection,Data Preprocessing, Feature Engineering, Machine Learning Models, Decision Trees,Security of Financial Transactions,Data Extraction,Financial Security

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1
# INTRODUCTION

## 1.1. FRAUD DETECTION IN WALMART

Online transactions are become an essential component of the shopping experience in today's ever changing digital environment, where efficiency and convenience are the driving forces behind the retail sector. Businesses face a major difficulty in ensuring the security of their financial ecosystem as a result of the growing risk of fraudulent activities in transactions resulting from consumers turning more and more to e-commerce platforms. Being a major worldwide retailer, Walmart understands how vital it is to protect its customers' transactions from fraud.

The purpose of this introduction is to clarify the critical function that transaction fraud detection systems play in Walmart's day-to-day operations. Walmart, a leader in the retail sector, is dedicated to utilizing cutting-edge technologies to strengthen its safeguards against fraudulent transactions. This dedication is evidence of more than just protecting its own financial.

## 1.2. STATISTICAL INFORMATION

Checking Walmart's official financial reports, news announcements, or other updates from the firm itself will yield the most up-to-date and accurate information.

Walmart is anticipated to operate in a context where worldwide e-commerce sales approach trillions of dollars yearly, offering a vast landscape for possible fraud, even though particular Early statistics on the retail giant's transaction fraud detection efforts may not be easily accessible. Walmart's commitment to utilizing cutting-edge technologies, like machine learning and real-time detection systems, is consistent with market trends that demonstrate how effective these strategies are at reducing false positives and improving the precision of fraud detection.

Walmart's methods must constantly change in order to safeguard the integrity of its financial transactions, as seen by the dynamic nature of fraud rates that are driven by both economic conditions and technology improvements. Seeking official papers, press announcements, and industry assessments that are exclusive to Walmart's for the most recent and accurate statistical insights

## 1.3. CURRENT FRAUD DETECTION TECHNIQUES

These are the types of techniques that are commonly used in the retail industry, and these may align with industry best practices.

### 1.3.1 Machine Learning Algorithms:

Employing advanced machine learning algorithms to analyze vast amounts of transaction data, identify patterns, and detect anomalies indicative of fraudulent activities. These algorithms continuously learn and adapt to evolving fraud patterns.

### 1.3.2 Real-time Transaction Monitoring:

Implementing systems that monitor transactions in real-time, enabling the immediate detection and response to suspicious activities as they occur.

### 1.3.3 Behavioral Analytics:

Analyzing user behavior to establish a baseline and identify deviations or anomalies that may indicate fraudulent activity. This could include changes in purchasing patterns, unusual transaction times, or unexpected locations.

### 1.3.4 Biometric Authentication:

Utilizing biometric verification methods, such as fingerprints or facial recognition, to enhance user identity authentication and reduce the risk of unauthorized transactions.

### 1.3.5 Device Fingerprinting:

Examining unique characteristics of devices used for transactions, such as device type, IP address, and location, to identify and flag irregularities.

### 1.3.6 Geolocation Tracking:

Verifying the geographic location of a transaction to ensure it aligns with the user's typical locations, helping to identify and prevent fraudulent activities.

### 1.3.7 Two-Factor Authentication (2FA):

Requiring users to provide two forms of identification before completing a transaction for an additional layer of security.

### 1.3.8 Collaborative Filtering:

Comparing a user's behavior with the behavior of a larger user community to identify patterns and potential outliers that may indicate fraudulent activity.

Among these models Machine Learning Algorithms was chose to analyze vast amounts of transaction data, identify patterns, and detect anomalies indicative of fraudulent activities. These algorithms continuously learn and adapt to evolving fraud patterns.

# CHAPTER 2

## LITERATURE REVIEW

This section talks about the other previous works on prediction of Walmart Transaction Fraud Detection System.

The paper [1] The 48th Euromicro Conference on SEAA in 2022 featured a paper by R. Bajaj et al. that introduced an AutoML framework that was implemented within Walmart's ecosystem. In order to make the creation and upkeep of machine learning (ML) models easier, this framework includes a number of tools, such as automatic feature engineering. The subject in question presents various obstacles, including those pertaining to model generalizability, data privacy, and the capacity to adjust quickly to market fluctuations, as noted by the authors. It comprises two TL-based models: namely, DenseNet-121 and Densenet-201 for features extraction, whereas in the second phase of implementation, it carries out three different ML classifiers like SVM and XGBoost for classification purposes. The final classifier outcomes are evaluated by means of permutations of the voting mechanism. The proposed model achieved accuracy of 91.75%, specificity of 96.5%, and an F1-score of 90.25.

In paper [2], The study by M. V. Ramasami et al., which was presented at the 2023 International Conference on ICDATE, focuses on the use of data analytics tools for exploratory data analysis of Walmart shops' sales. To find trends and factors affecting sales, the study makes use of statistical methods and visualizations. But it also draws attention to some of its possible drawbacks, like the dependence on data integrity and the risk of missing future trends when depending just on previous data. In order to solve the shortcomings of historical data and promote a more forward-looking approach to sales forecasting, the paper proposes expanding the analysis's scope to include predictive analytics and real-time data analysis. This would improve the study's capacity to provide dynamic market insights. The top-performing classification models were the support vector machine and random forest classifiers trained on BERT embedding, which both achieved an accuracy of

85.4% on the test set. The performance on both tasks illustrates the feasibility of using speech to classify AD and predict neuro psychological scores.

In paper [3] An adaptable machine learning model designed specifically for Walmart sales prediction is introduced in the article written by S. B. Latha et al. and presented at the 2023 ICCPCT, IEEE 2023. In order to achieve improved forecasting accuracy and robustness, the model integrates complex algorithms intended for dynamic flexibility. On the other hand, the risk of overfitting to certain datasets and the constraints of incorporating real-time market trend data are acknowledged as potential obstacles in the paper. The study proposes that in order to overcome these obstacles and guarantee the model's relevance in the face of changing market circumstances, its flexibility should be further improved. Random Forests(RF) , Support Vector Machines, and KNN setting was adopted. The classifiers were trained and tested to predict whether a speech segment was uttered by a non-AD or AD patient where it was found that DT gave the highest and most stable accuracy and features gave a higher accuracy of 0.625 for acoustic features.

## 2.1. OBJECTIVE

Walmart's efforts to prevent transaction fraud are motivated by a variety of goals, including protecting consumer information, strengthening financial transactions, and maintaining confidence in its vast retail network. Prioritizing these goals is the dedication to reducing monetary losses through the use of sophisticated fraud detection tools. The implementation of real-time detection and prevention systems is crucial as it allows Walmart to rapidly discover and thwart fraudulent transactions, thereby protecting its financial interests and upholding customer confidence. One of Walmart's main goals is to constantly adjust to new fraud strategies. It does this by upgrading and improving its fraud detection algorithms in an effort to stay ahead of new threats. Simultaneously, there is a determined attempt to achieve equilibrium by diminishing false positives, guaranteeing that lawful transactions transpire without any problems.

- Walmart's fraud detection system uses real-time monitoring to quickly react and minimize financial losses, with the goal of achieving a high degree of accuracy in recognizing and stopping fraudulent activity.

- The system places a high priority on its capacity to adjust to new threats and keep ahead of evolving fraud strategies. One of the main goals is scalability, which allows the system to easily manage high transaction volumes.

- Furthermore, the solution aims to reduce interference for reputable clients by offering thorough protection against different types of fraud and facilitating a smooth integration with payment gateways.

# CHAPTER 3
## PROPOSED METHODOLOGY

The proposed methodology for Walmart's transaction fraud detection developed a comprehensive framework to strengthen its financial operations and protect client data, which includes a process for detecting transaction fraud. Ensuring that transaction data from various channels—such as in-store and online platforms—is carefully gathered and integrated is the first step. In order to differentiate between genuine transactions and possible fraud, feature engineering then extracts relevant information. The goal is to enable the system with effective pattern identification and anomaly detection so that it can adjust to changing fraud strategies by utilizing a suite of machine learning algorithms, including supervised and unsupervised techniques.

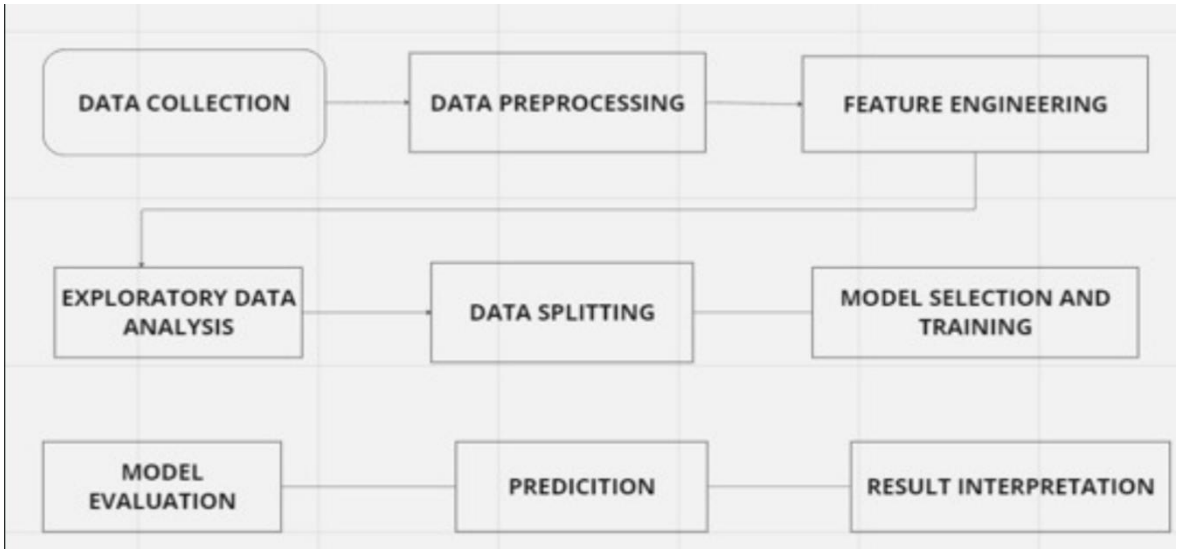



Fig 3.1 Methodology

The key element is real-time monitoring, which makes sure that any fraudulent activity is quickly identified and dealt with to reduce the possibility of financial loss. By creating models that identify abnormalities in user behavior, behavioral analytics can be used to provide an extra level of fraud detection. A strong layer of protection is added to user identity verification during transactions with biometric authentication, such as fingerprint or face recognition.

## 3.1. TOOLS AND TECHNIQUES

For the purpose of detecting transaction fraud, Walmart, like many other companies, probably uses a mix of methods and instruments. I can give a general summary of typical technologies and methods utilized in the sector, even if Walmart's fraud detection system's specifics are confidential:

**Machine Learning Algorithms:** For classification applications, such as fraud detection, Decision Trees, Random Forests, and Gradient Boosting Machines are typically employed as algorithms. Useful for binary classification tasks and appropriate in fraud detection scenarios are logistic regression and support vector machines (SVM). Complex patterns in transaction data can be captured by deep learning models, and neural networks in particular.

**Anomaly Detection:** Methods such as clustering, Gaussian mixture models, and Z-score analysis can be used to find odd patterns in transaction data. With high-dimensional data, isolation forests are useful for separating out anomalies.

**Feature Engineering:** Feature Engineering is the process of examining transaction quantities, trends, and frequency to spot abnormal activity. evaluating the transactions' consistency with respect to geography.

**Graph Analysis:** Using relationship modeling to uncover potentially fraudulent networks between entities (users, accounts, etc.). discovering hidden patterns by looking at relationships between entities and transactions.

**Ensemble Models:** Mixing the output of a decision tree model with a neural network model, for example, to increase overall accuracy by integrating numerous models.

**Fraud Intelligence Platforms:** To keep up to date on recognized fraud trends and changing risks, subscribe to external threat intelligence feeds.

**Continuous Learning and Adaptation:** Putting in place processes that take feedback into account and update models often in response to fresh information and new fraud trends.

# CHAPTER 4

# IMPLEMENTATION

## 4.1 WALMART TRANSACTION FRAUD DETECTION BY USING ML MODELS

### 4.1.1. About The Dataset:

**Source of the Dataset:** Walmart fraud detection (kaggle.com)

**No. of Observations:**6362604

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | PAYMENT | 9839.64 | C1231006815 | 170136.00 | 160296.36 | M1979787155 | 0.00 | 0.00 | 0 | 0 |
| 1 | 1 | PAYMENT | 1864.28 | C1666544295 | 21249.00 | 19384.72 | M2044282225 | 0.00 | 0.00 | 0 | 0 |
| 2 | 1 | TRANSFER | 181.00 | C1305486145 | 181.00 | 0.00 | C553264065 | 0.00 | 0.00 | 1 | 0 |
| 3 | 1 | CASH_OUT | 181.00 | C840083671 | 181.00 | 0.00 | C38997010 | 21182.00 | 0.00 | 1 | 0 |
| 4 | 1 | PAYMENT | 11668.14 | C2048537720 | 41554.00 | 29885.86 | M1230701703 | 0.00 | 0.00 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6362615 | 743 | CASH_OUT | 339682.13 | C786484425 | 339682.13 | 0.00 | C776919290 | 0.00 | 339682.13 | 1 | 0 |
| 6362616 | 743 | TRANSFER | 6311409.28 | C1529008245 | 6311409.28 | 0.00 | C1881841831 | 0.00 | 0.00 | 1 | 0 |
| 6362617 | 743 | CASH_OUT | 6311409.28 | C1162922333 | 6311409.28 | 0.00 | C1365125890 | 68488.84 | 6379898.11 | 1 | 0 |
| 6362618 | 743 | TRANSFER | 850002.52 | C1685995037 | 850002.52 | 0.00 | C2080388513 | 0.00 | 0.00 | 1 | 0 |
| 6362619 | 743 | CASH_OUT | 850002.52 | C1280323807 | 850002.52 | 0.00 | C873221189 | 6510099.11 | 7360101.63 | 1 | 0 |

6362620 rows × 11 columns

Fig 4.1 Dataset

**Column/Feature Details:**

**step:** This could represent the unit of time, possibly indicating the chronological order or time elapsed since a reference point.

**type:** Denotes the type of transaction (e.g., payment, transfer, etc.), providing insights into the nature of the financial activity.

**amounts:** Represents the monetary value of the transaction, indicating the sum involved in the financial operation.

**nameOrig:** Likely refers to the name or identifier of the originator (sender) of the transaction.

**oldbalanceOrg:** Indicates the initial account balance of the originator before the transaction.

**newbalanceOrg:** Represents the updated account balance of the originator after the transaction.

**nameDest:** Refers to the name or identifier of the destination (recipient) of the transaction.

**oldbalanceDest:** Denotes the initial account balance of the destination before the transaction.

**newbalanceDest:** Represents the updated account balance of the destination after the transaction.

**isFraud:** A binary indicator (likely 0 or 1) that flags whether the transaction is fraudulent (1) or not (0).

**isFlaggedFraud:** Another binary indicator that may signal whether the transaction is flagged as fraudulent based on certain criteria.

### 4.1.2. Preprocessing

Pre-processing is just as important in deep learning as it is in other areas of machine learning and data analysis. In fact, it can be argued that it is even more critical in deep learning due to the complexity of the models and the large amount of data typically involved.

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points. Techniques include resizing, zoom, rotating, cropping, padding, etc. It helps to address issues like overfitting and data scarcity, and it makes the model robust with better performance.

### 4.1.3. Evaluation

To determine how well a Walmart transaction fraud detection system performs, a variety of criteria and metrics that are usually used to evaluate the model's ability to recognize fraudulent transactions. The following are some typical assessment criteria and measures.

**1. Accuracy:** Accuracy is the most commonly used evaluation metric for classification tasks. It measures the proportion of correct predictions made by the model.

**2. Precision and Recall:** Precision and Recall are evaluation metrics used in binary classification tasks. Precision measures the proportion of true positives among all positive predictions, while recall measures the proportion of true positives among all actual positive cases.

**3. F1 Score:** The F1 score is a weighted average of precision and recall that combines the two metrics into a single score. It is often used in binary classification tasks where precision and recall are both important.

**4. Cross Validation:** To make sure the model is stable and generalizable over several data subsets, use cross-validation approaches.

**5. Confusion Matrix:** The effectiveness of a classification system is frequently assessed using a table called a confusion matrix. On a classification task, it provides an overview of a model's predictions.

**6. Feature Importance:** To find out which features are most important for fraud detection, evaluate each feature's significance within the model.

**7. Specificity:** The proportion of correctly identified non-fraudulent transactions. vital to comprehending how well the approach performs with reputable deals

**8. Temporal Evaluation:** Evaluate the model's long-term results. Assess its ability to adjust to evolving fraudulent conduct trends.

**9. Benchmarketing Against Baselines:** To make sure the machine learning model adds a meaningful amount of value, compare the fraud detection system's performance to baseline models or rules-based systems.

**10. Receiver Operating Characteristic (ROC) Curve:** ROC curves show how true positive rate (sensitivity) and false positive rate are traded off.

## 4.2. SPEECH DATA CLASSIFICATION

### 4.2.1. About The Dataset

**Source:** Kaggle

**License:** Dr. C. Shyamala, P. Sabarish, T. Vignesh, S. Yogeendran. 2021 WALMART Fraud Detection using MACHINE LEARNING ALGORITHMS. , K.Ramakrishnan College of Technology Retrieved from Semantic Scholar

### 4.2.2. Exploratory Data Analysis

The process of developing a strong fraud detection system for Walmart transactions begins with exploratory data analysis, or EDA. In order to find patterns, linkages, and potential insights that guide future modeling efforts, this procedure entails a thorough review of the information. Analysts examine many facets of the transaction data during EDA, such as transaction volumes, frequency, time distributions, and user behavior, among others. To find patterns and abnormalities in the data, visualization tools like scatter plots, heatmaps, and histograms can be used.EDA includes statistical analyses as well in order to comprehend the distribution and central tendencies of important variables. These analyses aid in the identification of possible anomalies or outliers that could be signs of fraud. Walmart's fraud detection team can make wise choices about feature engineering, preprocessing procedures, and the choice of suitable machine learning models by using EDA to extract more insight from the data. EDA serves as an essential base in the end, influencing the model development phases that come after and enhancing the system's ability to detect and reduce transaction fraud in Walmart's extensive and intricate transaction environment.

### 4.2.3. Feature Extraction

In order to improve the discriminating power of fraud detection models, feature extraction is a crucial stage in the Walmart transaction fraud detection process. It involves extracting pertinent information from raw transaction data. Features in this context refer to quantifiable traits or qualities that are taken from every transaction and can help differentiate between honest and dishonest behavior.

The transaction amount, frequency, location, time of day, user behavior patterns, and correlations with previous transactions are examples of relevant features. In order to minimize dimensionality while keeping important information, sophisticated methods like Principal Component Analysis (PCA) or autoencoders might be used. The objective is to produce a small but useful collection of attributes that captures the distinct patterns connected to fraud. Walmart's fraud detection technology may identify unusual activities by extracting significant information, which helps the customers to do safe transaction and it also helps in walmart’s buisness growth.

### 4.2.4. Pre-Processing

Preprocessing is a critical step in any machine learning project, including classification tasks such as Walmart Transaction Fraud classification.

Train-test split involves dividing the available data into two subsets: a training set and a testing set. The training set is used to train the machine learning model, while the testing set is used to evaluate the performance of the trained model. This technique is essential for preventing overfitting and ensuring that the model generalizes well to new data.

Standard scaling, also known as Z-score normalization, is a preprocessing technique used to scale the data to have zero mean and unit variance. Standard scaling is applied separately to each feature in the data, and it involves subtracting the mean of the feature and dividing by its standard deviation.

A statistical approach called "winsorization" is used to reduce the impact of extreme values, or outliers, in a dataset by "winsorizing" or restricting their values to fall within a given range. The impact of outliers on statistical models and analysis is mitigated in part by this approach. Winsorization is a technique that minimizes the impact of extreme values while maintaining the overall distribution of the data by setting them to a threshold value . The process is commonly represented as a percentage, signifying the percentage of extreme values that will be substituted with the selected threshold. When working with datasets that contain outliers,

winsorization is frequently used in data preprocessing to increase the stability of machine learning models and the robustness of statistical studies.

A technique called SMOTE, or Synthetic Minority Over-sampling Technique, is applied to machine learning datasets to rectify class imbalance, especially when it comes to classification issues. Underrepresentation of one class in the dataset relative to the other classes (often the minority class) is known as class imbalance.

### 4.2.5. Models

After having performed data preprocessing, we apply classification models, namely,

**Logistic Regression**

- Logistic regression is used to forecast the likelihood that an outcome will fall into one of two groups in binary categorization. To model the link between input factors and the probability of the binary outcome, it makes use of the logistic function.
- The logistic function, also known as the sigmoid function, is utilized by the logistic regression model to convert a linear combination of input characteristics into a probability score between 0 and 1, which is subsequently utilized for prediction purposes.
- The change in the dependent variable's log-odds is represented by the logistic regression coefficient. The influence of input variables on the probability that an event will occur can be understood using the odds ratio.

**Random Forest**

- Random Forest is an ensemble learning algorithm that is commonly used for classification and regression tasks in machine learning.
- Random Forest algorithm is based on probability.
- To make a prediction on new data, the random forest algorithm first runs each data point through each of the individual decision trees in the forest, and then

takes the majority vote of the predicted class (in classification tasks) or the average prediction (in regression tasks).

**Support Vector Machine**

- Support Vector Machines (SVMs) are a type of supervised learning algorithm used for classification and regression tasks. The basic idea behind SVMs is to find a hyperplane or a set of hyperplanes in a high-dimensional space that can best separate the data into different classes.
- In the case of classification, SVMs aim to find a decision boundary that maximizes the margin between the classes.

**K-Nearest Neighbor**

- KNN is a simple yet effective algorithm that does not make any assumptions about the underlying distribution of the data.
- The algorithm works by calculating the distance between the test example and each training example in the feature space. K-Nearest Neighbors (KNN) is a type of supervised machine learning algorithm that is used for both classification and regression tasks.
- The value of K is a hyper parameter that needs to be tuned for each specific problem, and a larger value of K will result in a smoother decision boundary but may also introduce more bias.

**Decision Tree:**

- Decision tree is a supervised machine-learning algorithm that is commonly used for classification and regression tasks.
- It is a type of predictive modeling algorithm that makes predictions by learning simple decision rules from the input features of the training data(by recursively splitting training data into subsets based on input features).

**Autoencoder:**

- Neural network architectures called autoencoders are utilized in unsupervised learning. By encoding and decoding input data, they learn effective representations of it with the goal of reproducing the original input.
- In order to reduce dimensionality and capture the key characteristics of the input data in a compressed representation—a feature that can be helpful for tasks like data denoising and feature extraction—autoencoders are frequently used.
- An encoder network compresses the input data into a latent space representation, and a decoder network reconstructs the input from this representation. This is how autoencoders function. The model is encouraged to produce outputs that closely resemble the original inputs throughout the training phase.

**MLP Classifier:**

- A particular kind of artificial neural network called MLP is intended for supervised learning applications. It is made up of several levels of networked nodes, or neurons, that process incoming data and generate an output by way of hidden layers.
- Information flows in a single direction in a feedforward architecture, which is used by MLPs. It goes from the input layer via the hidden levels and ends at the output layer. A weight is assigned to each link between nodes, and during training the network discovers the ideal weights.

**Isolation Forest:**

- A machine learning approach for anomaly identification that operates without supervision is called Isolation Forest. By separating them in a binary tree structure, it effectively finds anomalies or outliers in a dataset.

- Recursively partitioning the data until anomalies are isolated with shorter paths in the trees is how Isolation Forest builds isolation trees. A feature and a split value are chosen at random. The average path lengths of anomalies in the trees are shorter than those of normal data points, which helps identify them.

**XGBooster:**

- XGBooster (Extreme Gradient Boosting) is a potent machine learning technique. It sequentially constructs an ensemble of weak learners, usually decision trees, where each new tree fixes the aggregate model's faults.
- Parallel Processing and Regularization: XGBoost prevents overfitting and improves the model's generalization skills by including regularization terms in its objective function. It can handle big datasets and is computationally efficient due to its implementation of parallel processing.

**One Class SVM:**

- An approach for machine learning called One-Class SVM (Support Vector Machine) is used to detect anomalies, especially in situations where most of the data falls into one class (normal) and anomalies are infrequent.
- Unsupervised Learning: One-Class Support Vector Machines (SVMs) acquire a representation of normal data by training solely on the normal class. Slight deviations from this acquired normal representation are identified as possible anomalies during inference.

**Artificial Neural Network:**

- The architecture and operation of biological neural networks serve as the basis for artificial neural networks, which are computational models. Layers including an input layer, one or more hidden layers, and an output layer are among the interconnected nodes (neurons) that make up this structure.

- Information moves unidirectionally from the input layer through the hidden layers and out to the output layer in feedforward architectures, which are commonly used in ANN operations. The network acquires these weights during training in order to translate inputs to outputs. Each link between nodes has a weight assigned to it.

| **Hyper Parameter** | **Values** |
|---|---|
| Epoch | 20 |
| Hidden layers | 3 |
| Activation Function | Relu |
| Optimizer | Adam Optimizer |
| Metrics | Accuracy |

Table 4.1. Hyperparameter

### 4.2.6. Evaluation

**1.Accuracy:** Accuracy is the most commonly used evaluation metric for classification tasks. It measures the proportion of correct predictions made by the model.

**2. Precision and recall:** These metrics are important for evaluating the performance of models in cases where the data is imbalanced. Precision measures the proportion of true positives among all positive predictions, while recall measures the proportion of true positives among all actual positive cases.

**3. F1 score:** It provides a balanced measure of the model's performance and is particularly useful for evaluating models in cases where both precision and recall are important.

| **Models** | **Accuracy** |
|---|---|
| Logistic Regression | 0.964 |
| Decision Tree Classifier | 0.999 |
| Random Forest Classifier | 0.998 |
| SVC | 0.998 |
| MPL Classifer | 0.998 |
| Isolation Forest | 0.886 |
| KNN | 0.999 |
| AdaBoost Classifer | 0.998 |
| One Class SVM | 0.520 |
| XG Booster | 0.999 |
| Autoencoder | 0.5 |
| ANN | 0.999 |

Table 4.2 Model Accuracy

The above table represents the accuracy for various models on training data

# CHAPTER 5

## RESULTS AND DISCUSSION

After comparing the accuracy for several models, we can clearly see that Logistic Regression performs the best in training and validation data i.e., It trades-off bias and variance . Flask API is used to get user inputs and make predictions using the final models. And this product is deployed in Heruko. And the accuracy for the training data is 0.996 and for testing data is 0.949. The Logistic threshold for spliting is 0.73 , where a transactiom is considered as a fraud if its predicted probability is more than 0.73 and a transaction is considered as non-fraud if the predicted probabllity value is less than or equal to 0.73.

# CHAPTER 6

# CONCLUSION

In conclusion, the classification of dementia using speech and MRI scans shows promising results. Both modalities provide complementary information for the accurate detection and diagnosis of different types of dementia.

Speech analysis allows for the identification of subtle changes in language patterns, while MRI scans provide information on structural and functional brain changes. The integration of these two modalities can improve diagnostic accuracy and enable early detection of dementia, allowing for timely intervention and management. However, further research is needed to improve the specificity and sensitivity of these classification methods, as well as to evaluate their practicality and applicability in clinical settings.

# CHAPTER 7

## FUTURE ENHANCEMENT

There are some broad patterns and prospective future developments in machine learning approaches that were anticipated to influence the subject. Remember that the terrain might have changed since then. Here are a few potential upgrades in the future enhancing the walmart fault detection system, those include:

**Interpretability and Explainability:**

A stronger focus on improving the interpretability and explainability of machine learning models, particularly in key applications where comprehending model conclusions is essential.

**Sturdiness and Safety:**

A greater emphasis on creating machine learning models that are resistant to adversarial assaults and making sure that machine learning systems are secure, particularly in areas like banking, healthcare, and driverless cars.

**Automated Feature Engineering and AutoML:**

Continuous improvements in automated machine learning, or AutoML, to facilitate the application of machine learning methods by non-experts. It's possible that powerful automated feature engineering tools will advance, simplifying the feature engineering procedure.

**Pre-trained Models and Transfer Learning:**

Increased usage of pre-trained models and transfer learning, which allow models to employ information from one job to perform better on another even in the case of sparse labeled data.

**Quantum computing integration:**

Investigation of quantum computing for machine learning applications, which could significantly speed up the resolution of challenging optimization issues that are common in the field.

**Developments in Reinforcement Learning:**

Additional developments in reinforcement learning methodologies, specifically in domains such as deep reinforcement learning, to tackle the obstacles encountered in training increasingly intricate and expandable models.

**Exponential Growth in Scalability and Data:**

Modifying machine learning methods to deal with the ever-growing amount and complexity of data, possibly by utilizing distributed computing and better scalability algorithms.

**AI Ethics and Justice:**

Stepping up efforts to address moral issues in machine learning, emphasizing responsibility, transparency, and justice in algorithmic decision-making.

**Combining Other Technologies:**

Combining machine learning with other cutting-edge technologies to create more intelligent and responsive systems, such as edge computing, 5G, and the Internet of Things (IoT).

**Sustained Development of Neural Network Structures:**

Creation of innovative neural network topologies and architectures, possibly going beyond the parameters of conventional deep learning architectures. Examples include investigating neuromorphic computing and spiking neural networks.

**Advances in Natural Language Processing:**

Additional advancements in natural language processing models, such as enhanced comprehension of sentiment, context, and subtleties in spoken language.

**Enhanced Data Analysis:**

The addition of machine learning to analytics tools to improve their capacity for data analysis and give decision-makers more useful information.

The above enhancement are in the process of development and some are only coined, Adding these not only in walmart fraud detection but also to other machine learning algorithms will increase security, creditbility and eases the existing process.

# CHAPTER 8

# APPENDICIES

## 8.1 CODE AND OUTPUT SCREENSHOTS

```
from google.colab import drive

drive.mount("/content/drive")

path=("/content/drive/MyDrive/Walmart /PS_20174392719_1491204439457_log.csv")


import pandas as pd

import numpy as np

import os

import csv

import seaborn as sns

import matplotlib.pyplot as plt

import sklearn

import random

import missingno as msno

import xgboost as xgb

from sklearn import *

from sklearn.preprocessing import LabelEncoder

from imblearn.over_sampling import SMOTE

from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.model_selection import train_test_split

from scipy.stats.mstats import winsorize

from sklearn.linear_model import LogisticRegression

from sklearn.datasets import make_classification

from sklearn.tree import DecisionTreeClassifier,plot_tree

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import mean_squared_error

from sklearn.svm import SVC

from sklearn.inspection import permutation_importance

from xgboost import plot_importance, plot_tree
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler
from keras.models import Model
from keras.layers import Input, Dense
from tensorflow.keras.models import Sequential
from sklearn.metrics import roc_curve, auc
from tensorflow.keras.layers import Dense
from sklearn.decomposition import PCA
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import roc_curve, auc
from sklearn.datasets import make_classification
from keras.models import Sequential
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_recall_curve, average_precision_score
#Reading the CSV
data=pd.read_csv(path)
data
```

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | PAYMENT | 9839.64 | C1231006815 | 170136.00 | 160296.36 | M1979787155 | 0.00 | 0.00 | 0 | 0 |
| **1** | 1 | PAYMENT | 1864.28 | C1666544295 | 21249.00 | 19384.72 | M2044282225 | 0.00 | 0.00 | 0 | 0 |
| **2** | 1 | TRANSFER | 181.00 | C1305486145 | 181.00 | 0.00 | C553264065 | 0.00 | 0.00 | 1 | 0 |
| **3** | 1 | CASH_OUT | 181.00 | C840083671 | 181.00 | 0.00 | C38997010 | 21182.00 | 0.00 | 1 | 0 |
| **4** | 1 | PAYMENT | 11668.14 | C2048537720 | 41554.00 | 29885.86 | M1230701703 | 0.00 | 0.00 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **6362615** | 743 | CASH_OUT | 339682.13 | C786484425 | 339682.13 | 0.00 | C776919290 | 0.00 | 339682.13 | 1 | 0 |
| **6362616** | 743 | TRANSFER | 6311409.28 | C1529008245 | 6311409.28 | 0.00 | C1881841831 | 0.00 | 0.00 | 1 | 0 |
| **6362617** | 743 | CASH_OUT | 6311409.28 | C1162922333 | 6311409.28 | 0.00 | C1365125890 | 68488.84 | 6379898.11 | 1 | 0 |
| **6362618** | 743 | TRANSFER | 850002.52 | C1685995037 | 850002.52 | 0.00 | C2080388513 | 0.00 | 0.00 | 1 | 0 |
| **6362619** | 743 | CASH_OUT | 850002.52 | C1280323807 | 850002.52 | 0.00 | C873221189 | 6510099.11 | 7360101.63 | 1 | 0 |

6362620 rows × 11 columns

Displaying the first 10 values in the dataset to study the data

```
data.head(10)
```

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | PAYMENT | 9839.64 | C1231006815 | 170136.00 | 160296.36 | M1979787155 | 0.0 | 0.00 | 0 | 0 |
| 1 | 1 | PAYMENT | 1864.28 | C1666544295 | 21249.00 | 19384.72 | M2044282225 | 0.0 | 0.00 | 0 | 0 |
| 2 | 1 | TRANSFER | 181.00 | C1305486145 | 181.00 | 0.00 | C553264065 | 0.0 | 0.00 | 1 | 0 |
| 3 | 1 | CASH_OUT | 181.00 | C840083671 | 181.00 | 0.00 | C38997010 | 21182.0 | 0.00 | 1 | 0 |
| 4 | 1 | PAYMENT | 11668.14 | C2048537720 | 41554.00 | 29885.86 | M1230701703 | 0.0 | 0.00 | 0 | 0 |
| 5 | 1 | PAYMENT | 7817.71 | C90045638 | 53860.00 | 46042.29 | M573487274 | 0.0 | 0.00 | 0 | 0 |
| 6 | 1 | PAYMENT | 7107.77 | C154988899 | 183195.00 | 176087.23 | M408069119 | 0.0 | 0.00 | 0 | 0 |
| 7 | 1 | PAYMENT | 7861.64 | C1912850431 | 176087.23 | 168225.59 | M633326333 | 0.0 | 0.00 | 0 | 0 |
| 8 | 1 | PAYMENT | 4024.36 | C1265012928 | 2671.00 | 0.00 | M1176932104 | 0.0 | 0.00 | 0 | 0 |
| 9 | 1 | DEBIT | 5337.77 | C712410124 | 41720.00 | 36382.23 | C195600860 | 41898.0 | 40348.79 | 0 | 0 |

Displaying the last 10 values in the dataset to study the data

data.tail(10)

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6362610 | 742 | TRANSFER | 63416.99 | C778071008 | 63416.99 | 0.0 | C1812552860 | 0.00 | 0.00 | 1 | 0 |
| 6362611 | 742 | CASH_OUT | 63416.99 | C994950684 | 63416.99 | 0.0 | C1662241365 | 276433.18 | 339850.17 | 1 | 0 |
| 6362612 | 743 | TRANSFER | 1258818.82 | C1531301470 | 1258818.82 | 0.0 | C1470998563 | 0.00 | 0.00 | 1 | 0 |
| 6362613 | 743 | CASH_OUT | 1258818.82 | C1436118706 | 1258818.82 | 0.0 | C1240760502 | 503464.50 | 1762283.33 | 1 | 0 |
| 6362614 | 743 | TRANSFER | 339682.13 | C2013999242 | 339682.13 | 0.0 | C1850423904 | 0.00 | 0.00 | 1 | 0 |
| 6362615 | 743 | CASH_OUT | 339682.13 | C786484425 | 339682.13 | 0.0 | C776919290 | 0.00 | 339682.13 | 1 | 0 |
| 6362616 | 743 | TRANSFER | 6311409.28 | C1529008245 | 6311409.28 | 0.0 | C1881841831 | 0.00 | 0.00 | 1 | 0 |
| 6362617 | 743 | CASH_OUT | 6311409.28 | C1162922333 | 6311409.28 | 0.0 | C1365125890 | 68488.84 | 6379898.11 | 1 | 0 |
| 6362618 | 743 | TRANSFER | 850002.52 | C1685995037 | 850002.52 | 0.0 | C2080388513 | 0.00 | 0.00 | 1 | 0 |
| 6362619 | 743 | CASH_OUT | 850002.52 | C1280323807 | 850002.52 | 0.0 | C873221189 | 6510099.11 | 7360101.63 | 1 | 0 |

Displaying the basic information about the datas in the dataset

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

Displaying the number of rows and columns in the dataset

data.shape

```
(6362620, 11)
```

Displaying statistical information about the dataset

data.describe()

| | step | amount | oldbalanceOrg | newbalanceOrig | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|
| count | 6.362620e+06 | 6.362620e+06 | 6.362620e+06 | 6.362620e+06 | 6.362620e+06 | 6.362620e+06 | 6.362620e+06 | 6.362620e+06 |
| mean | 2.433972e+02 | 1.798619e+05 | 8.338831e+05 | 8.551137e+05 | 1.100702e+06 | 1.224996e+06 | 1.290820e-03 | 2.514687e-06 |
| std | 1.423320e+02 | 6.038582e+05 | 2.888243e+06 | 2.924049e+06 | 3.399180e+06 | 3.674129e+06 | 3.590480e-02 | 1.585775e-03 |
| min | 1.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |
| 25% | 1.560000e+02 | 1.338957e+04 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 |
| 50% | 2.390000e+02 | 7.487194e+04 | 1.420800e+04 | 0.000000e+00 | 1.327057e+05 | 2.146614e+05 | 0.000000e+00 | 0.000000e+00 |
| 75% | 3.350000e+02 | 2.087215e+05 | 1.073152e+05 | 1.442584e+05 | 9.430367e+05 | 1.111909e+06 | 0.000000e+00 | 0.000000e+00 |
| max | 7.430000e+02 | 9.244552e+07 | 5.958504e+07 | 4.958504e+07 | 3.560159e+08 | 3.561793e+08 | 1.000000e+00 | 1.000000e+00 |

Finding the missing values and summing if any

#checking for missing values

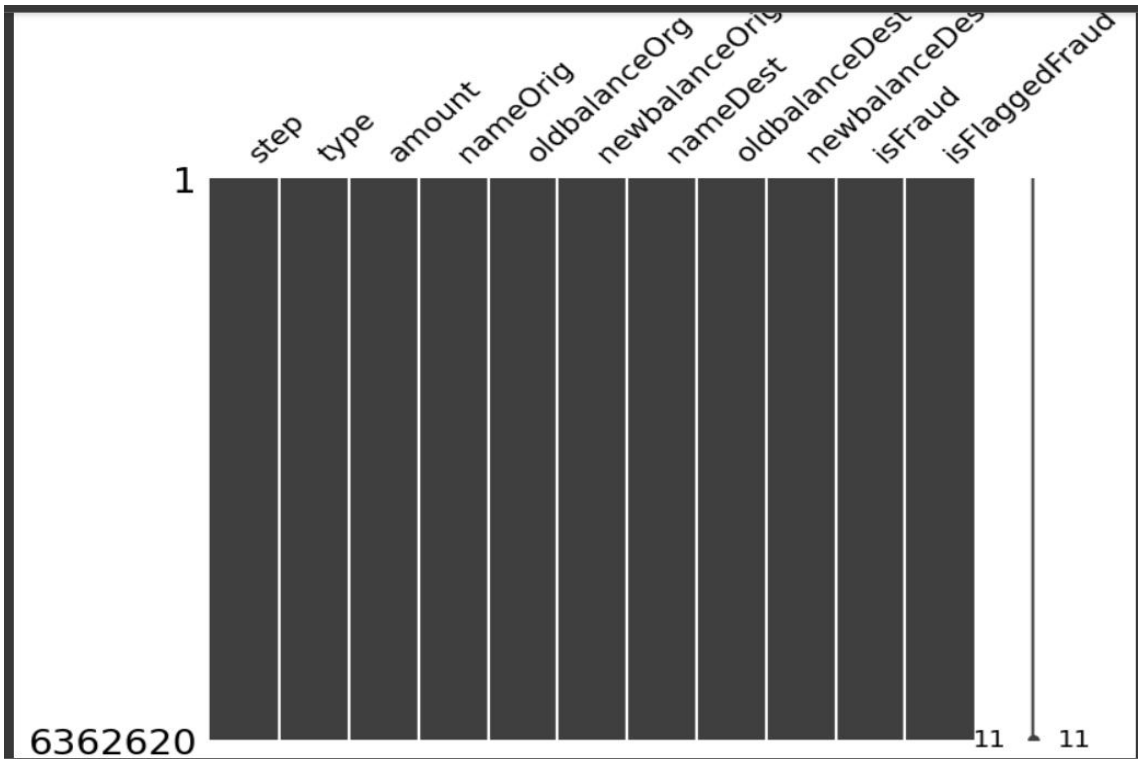
data.isnull().sum()

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Plotting graph to display the missing values

#checking for missing values

msno.matrix(data,figsize=(8, 6))




The corr() method is used to calculate the corelation matrix

correlation = data.corr()

correlation

```
<ipython-input-15-521f87fcc686>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future
  correlation = data.corr()
```

| | step | amount | oldbalanceOrg | newbalanceOrig | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|
| step | 1.000000 | 0.022373 | -0.010058 | -0.010299 | 0.027665 | 0.025888 | 0.031578 | 0.003277 |
| amount | 0.022373 | 1.000000 | -0.002762 | -0.007861 | 0.294137 | 0.459304 | 0.076688 | 0.012295 |
| oldbalanceOrg | -0.010058 | -0.002762 | 1.000000 | 0.998803 | 0.066243 | 0.042029 | 0.010154 | 0.003835 |
| newbalanceOrig | -0.010299 | -0.007861 | 0.998803 | 1.000000 | 0.067812 | 0.041837 | -0.008148 | 0.003776 |
| oldbalanceDest | 0.027665 | 0.294137 | 0.066243 | 0.067812 | 1.000000 | 0.976569 | -0.005885 | -0.000513 |
| newbalanceDest | 0.025888 | 0.459304 | 0.042029 | 0.041837 | 0.976569 | 1.000000 | 0.000535 | -0.000529 |
| isFraud | 0.031578 | 0.076688 | 0.010154 | -0.008148 | -0.005885 | 0.000535 | 1.000000 | 0.044109 |
| isFlaggedFraud | 0.003277 | 0.012295 | 0.003835 | 0.003776 | -0.000513 | -0.000529 | 0.044109 | 1.000000 |

# checking type column categories

data["type"].unique()

```
array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
      dtype=object)
```

# getting the categories in type column

unique_categories = data['type'].unique()

print(unique_categories)

```
['PAYMENT' 'TRANSFER' 'CASH_OUT' 'DEBIT' 'CASH_IN']
```

```
data = pd.DataFrame(data)


# Find duplicate rows based on all columns
duplicate_rows = data[data.duplicated()]
# Display the duplicate rows
print("Duplicate Rows except first occurrence:")
print(duplicate_rows)
```

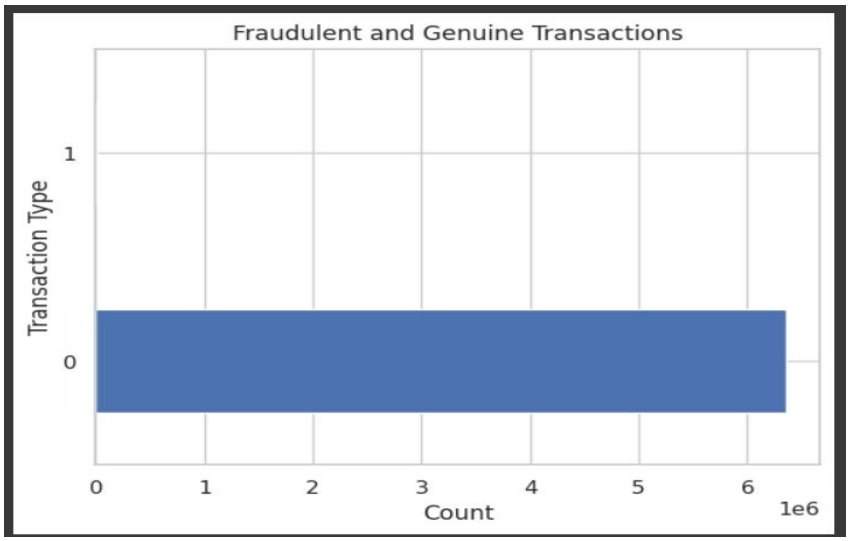
```
Duplicate Rows except first occurrence:
Empty DataFrame
Columns: [0]
Index: []
```

```
quantity = data['type'].values
print(quantity)
```

```
['PAYMENT' 'PAYMENT' 'TRANSFER' ... 'CASH_OUT' 'TRANSFER' 'CASH_OUT']
```

As we see above there is no NULL Values in data, we dont have to think about filling NAs we can proceed to EDA

```
data['isFraud'].value_counts().plot(kind="barh")
plt.xlabel('Count')
plt.ylabel('Transaction Type')
plt.title('Fraudulent and Genuine Transactions')
plt.show()
```



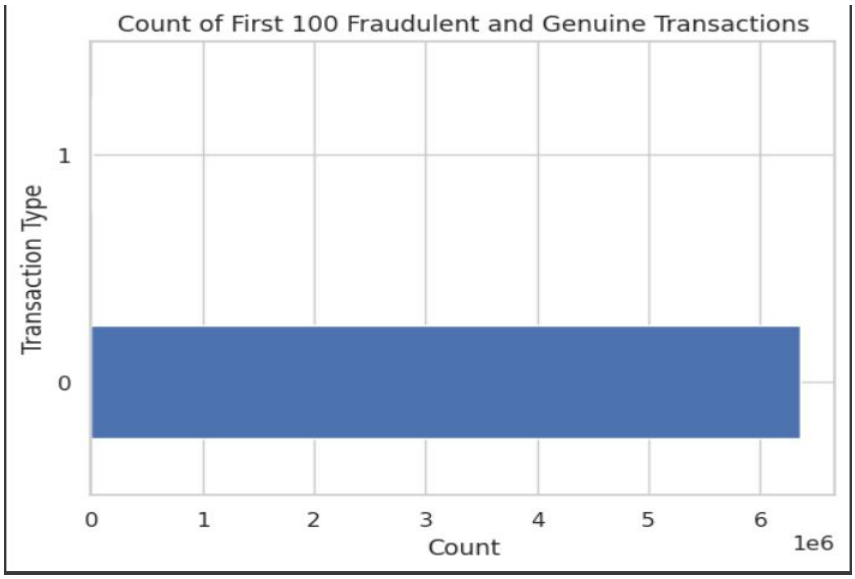

data['isFraud'].value_counts().head(100).plot(kind="barh")

plt.xlabel('Count')

plt.ylabel('Transaction Type')

plt.title('Count of First 100 Fraudulent and Genuine Transactions')

plt.show()




The Feature isFlaggedFound is a categorical feature and that is the target feature too, this is a classification based problem

data["isFlaggedFraud"].value_counts()

```
0    6362604
1         16
Name: isFlaggedFraud, dtype: int64
```

While the Target Column isFlaggedFraud is represented in bar graph, we can find that the data for fraud transaction is less (less than 10%) . The data for non fraud transaction will tend to dominate the fraud transactions.

**Exploratory Data Analysis**

data["step"].value_counts()

```
19     51352
18     49579
187    49083
235    47491
307    46968
       ...
432        4
706        4
693        4
112        2
662        2
Name: step, Length: 743, dtype: int64
```

plt.rcParams["figure.figsize"] = [8,5]

plt.rcParams["figure.autolayout"] = True

fig, ax = plt.subplots()

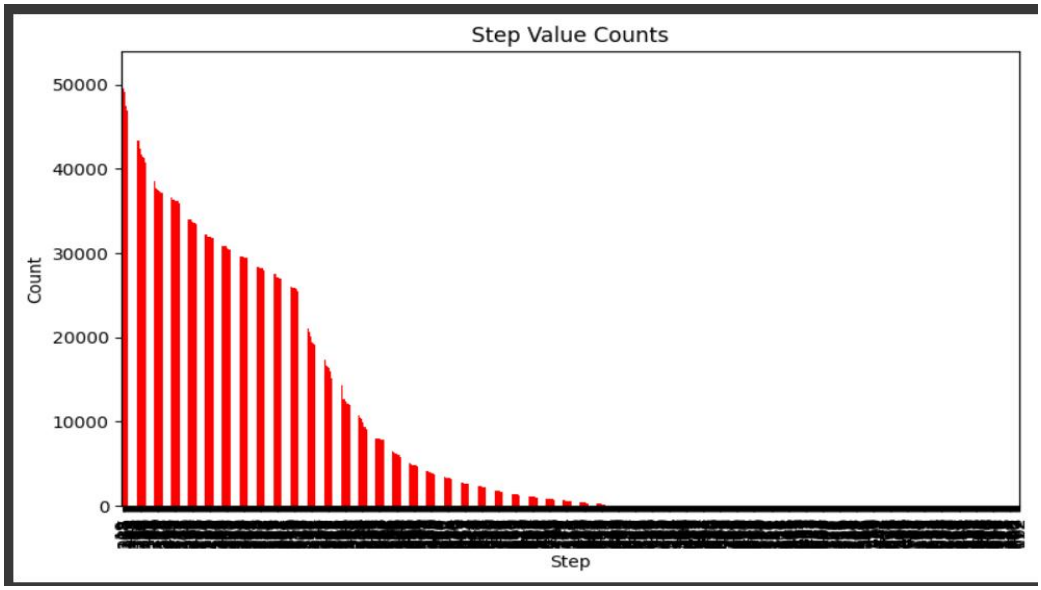
data["step"].value_counts().plot(ax=ax, kind="bar", color='red')

plt.title('Step Value Counts')

plt.xlabel('Step')

plt.ylabel('Count')

plt.show()



data["nameDest"].value_counts()

```
C1286084959    113
C985934102     109
C665576141     105
C2083562754    102
C248609774     101
               ...
M1470027725      1
M1330329251      1
M1784358659      1
M2081431099      1
C2080388513      1
Name: nameDest, Length: 2722362, dtype: int64
```

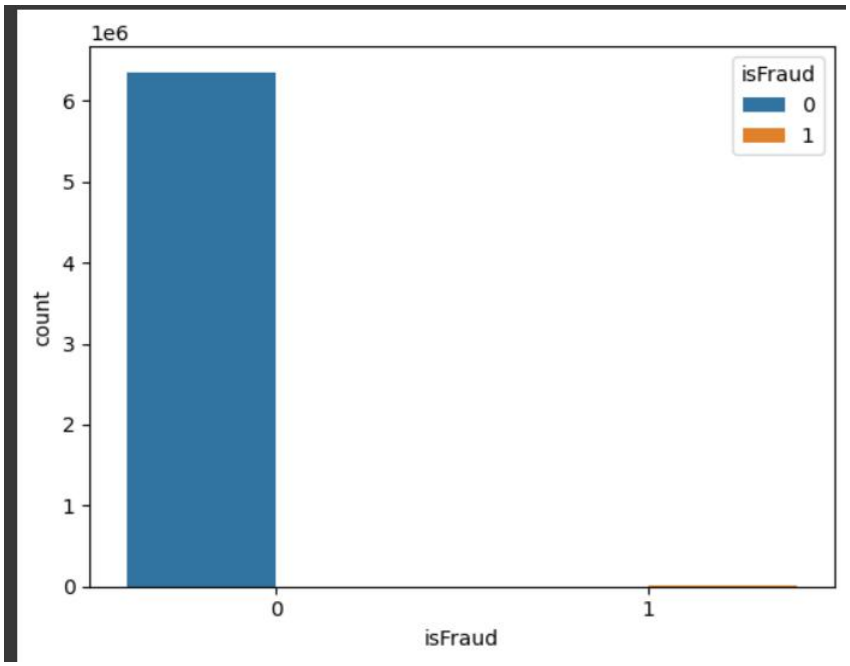
data["nameOrig"].value_counts()

```
C1902386530    3
C363736674     3
C545315117     3
C724452879     3
C1784010646    3
               ..
C98968405      1
C720209255     1
C1567523029    1
C644777639     1
C1280323807    1
Name: nameOrig, Length: 6353307, dtype: int64
```

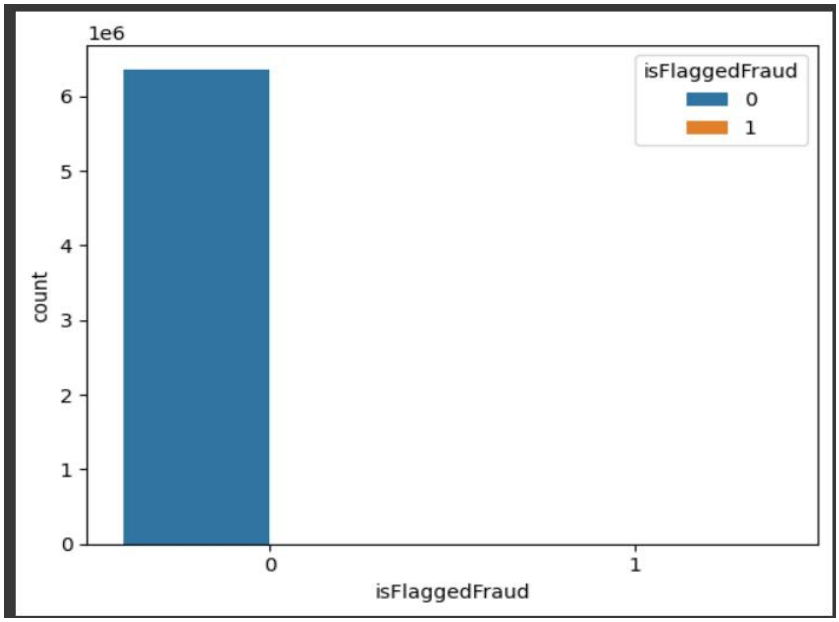
data['isFraud'].value_counts()

```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

sns.countplot(x = 'isFraud', hue = 'isFraud', data = data);




sns.countplot(x = 'isFlaggedFraud', hue = 'isFlaggedFraud', data = data);




fig, axes = plt.subplots(2, 2, figsize = (10,5))

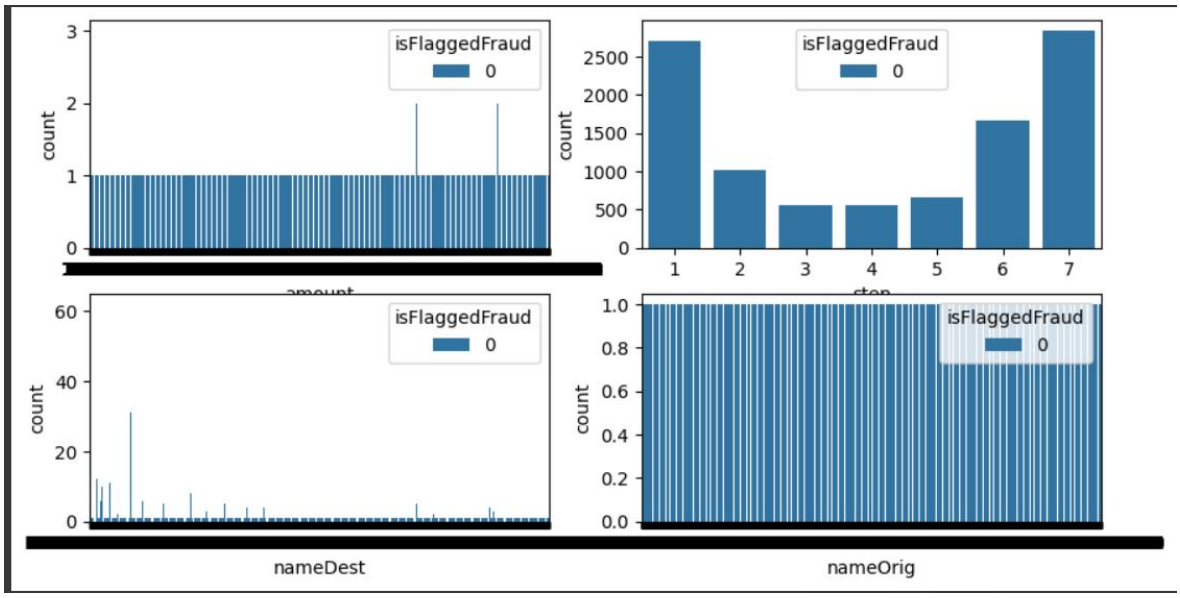
sns.countplot(ax = axes[0, 0], x = 'amount', hue = 'isFlaggedFraud', data = data.head(10000))

sns.countplot(ax = axes[0, 1], x = 'step', hue = 'isFlaggedFraud', data = data.head(10000))

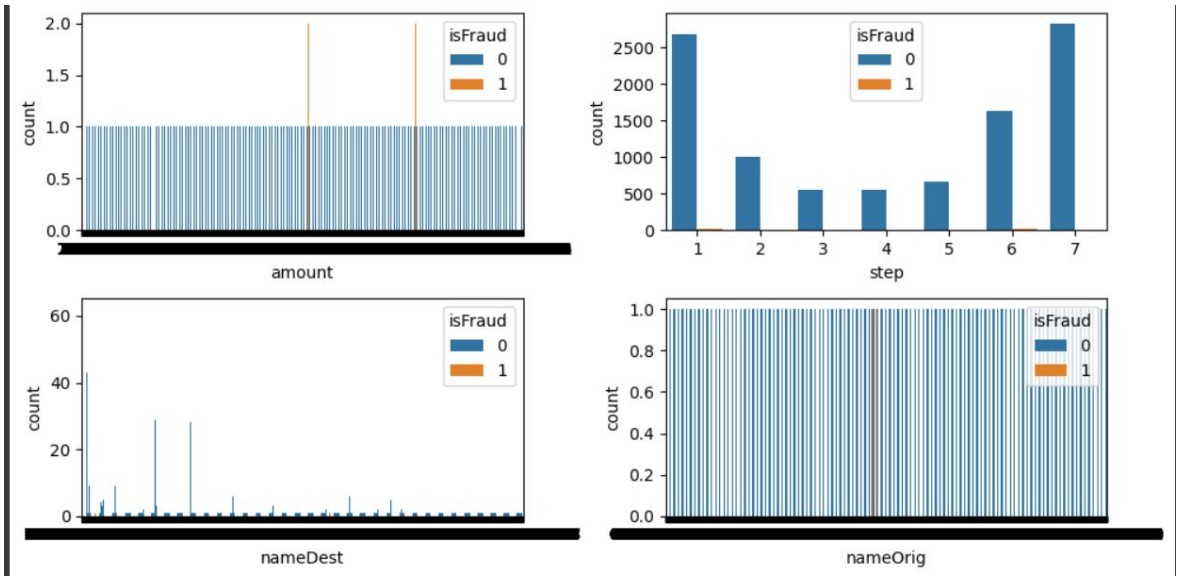
sns.countplot(ax = axes[1, 0], x = 'nameDest', hue = 'isFlaggedFraud', data = data.head(10000))

sns.countplot(ax = axes[1, 1], x = 'nameOrig', hue = 'isFlaggedFraud', data = data.head(10000))

plt.show()



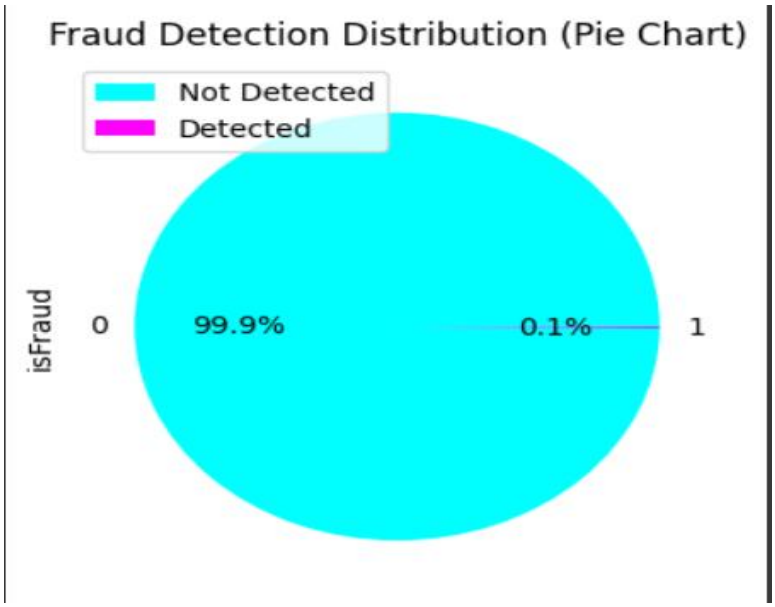

```
fig, axes = plt.subplots(2, 2, figsize = (10, 5))
sns.countplot(ax = axes[0, 0], x = 'amount', hue = 'isFraud', data = data.head(10000))
sns.countplot(ax = axes[0, 1], x = 'step', hue = 'isFraud', data = data.head(10000))
sns.countplot(ax = axes[1, 0], x = 'nameDest', hue = 'isFraud', data = data.head(10000))
sns.countplot(ax = axes[1, 1], x = 'nameOrig', hue = 'isFraud', data = data.head(10000))
plt.show()
```

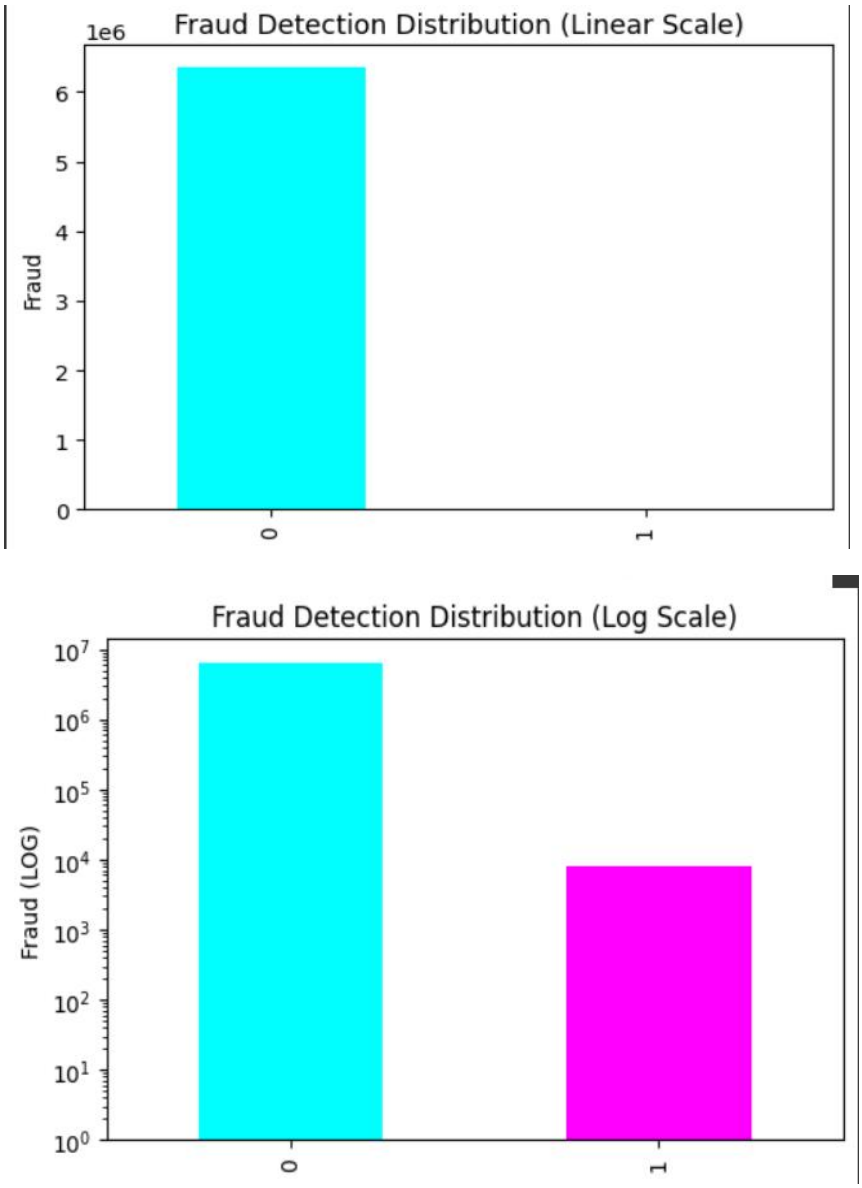



lternatively we can look at the pie chart. 99.8% of the transactions are normal, while only 0.2% are fraudulent.

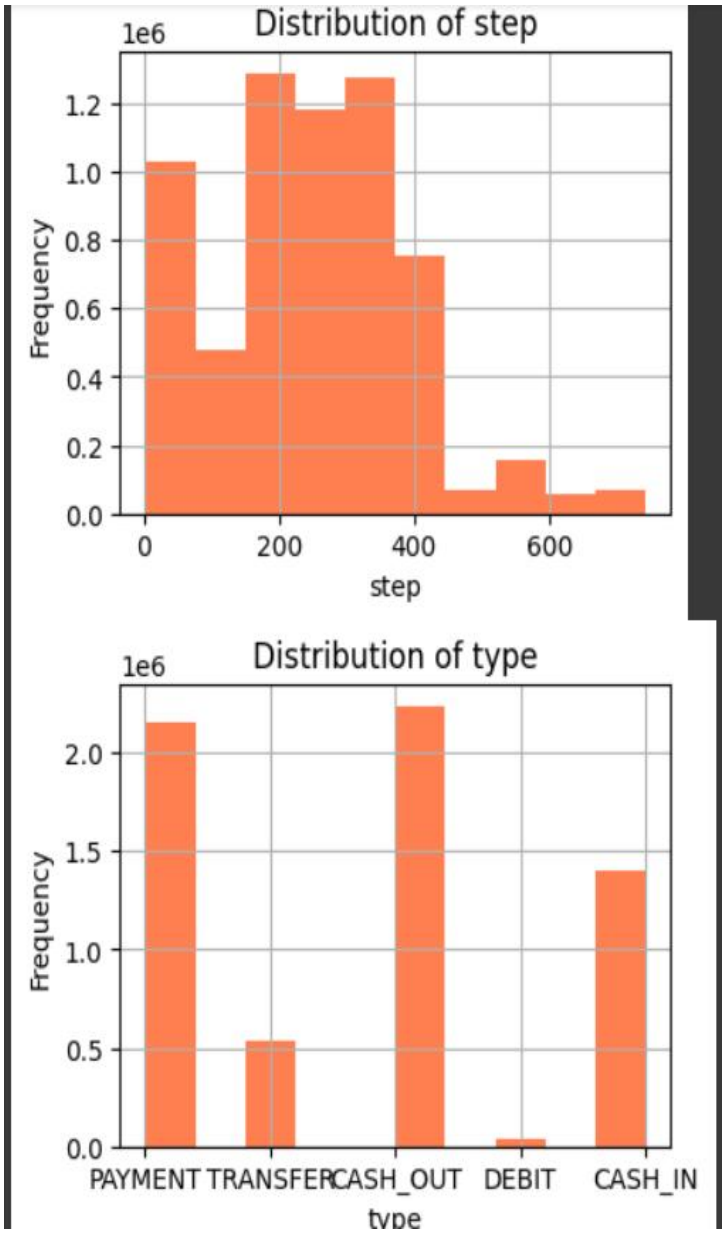
```
#Checking for class imbalence in target feature
fraud_count = data["isFraud"].value_counts()
colors = ['cyan','magenta']
```

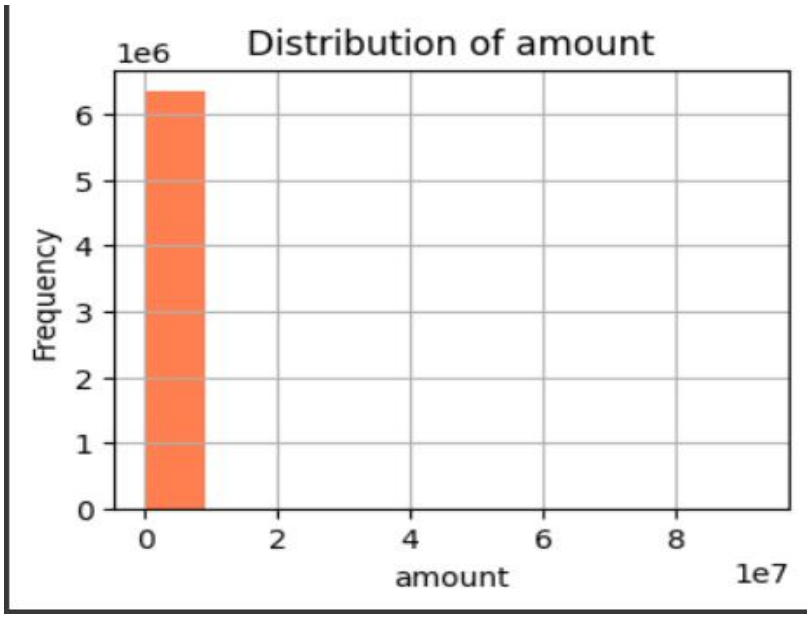
```
#piechart
plt.figure(figsize=(6,4))
fraud_count.plot(kind='pie', autopct='%1.1f%%', colors=colors)
plt.legend(loc='upper left', labels=['Not Detected', 'Detected'])
plt.title('Fraud Detection Distribution (Pie Chart)')
plt.show()

# Bar chart (linear scale)
plt.figure(figsize=(6,4))
ax = fraud_count.plot(kind='bar', color=colors, ylabel='Fraud', log=False)
plt.title('Fraud Detection Distribution (Linear Scale)')
plt.show()

# Bar chart (log scale)
plt.figure(figsize=(6,4))
ax = fraud_count.plot(kind='bar', color=colors, ylabel='Fraud (LOG)', log=True)
plt.title('Fraud Detection Distribution (Log Scale)')
plt.show()
```






```
#Histogram
def plotPerColumnDistribution(dataframe, color='yellow', figsize=(6, 4)):
    for column in dataframe.columns:
        plt.figure(figsize=figsize)
        dataframe[column].hist(color=color)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()
plotPerColumnDistribution(data, color='coral', figsize=(4, 3))
```

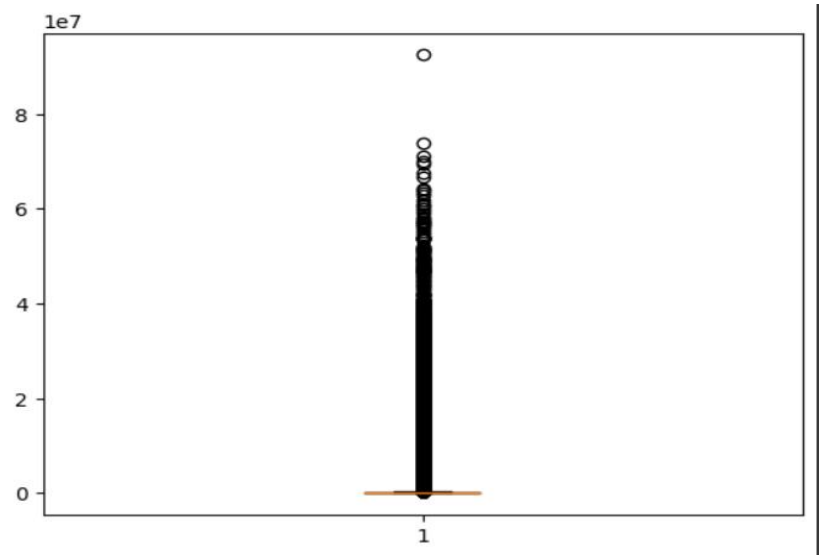

Distribution of step
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
0
200
400
600
step
Distribution of type
1e6
2.0
1.5
1.0
0.5
0.0
Frequency
PAYMENT
TRANSFER
CASH_OUT
DEBIT
CASH_IN
type




#Checking for Outliers

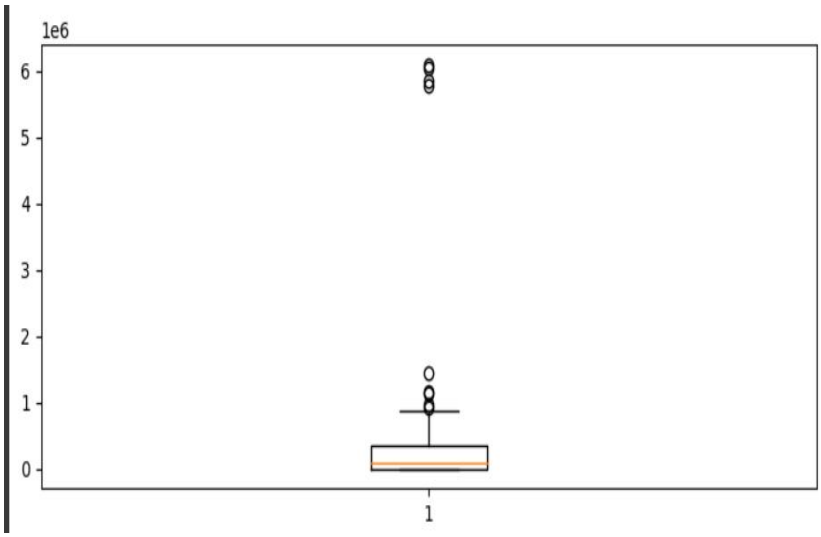
plt.boxplot(data["amount"])

plt.show()

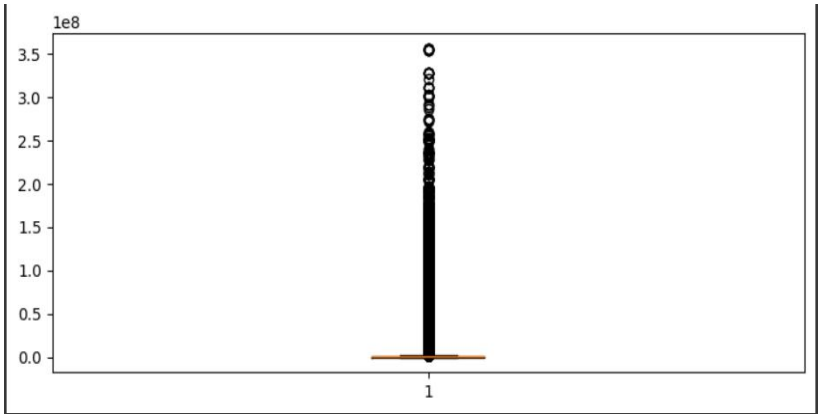



plt.boxplot(data["oldbalanceDest"][100:200])

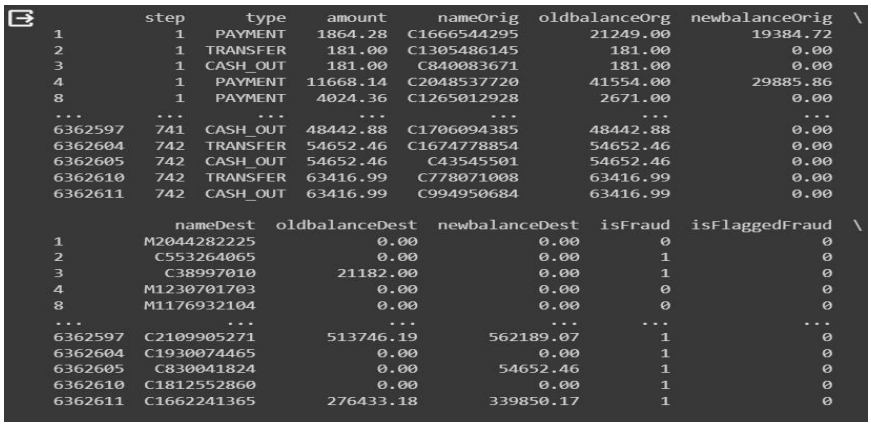
plt.show()



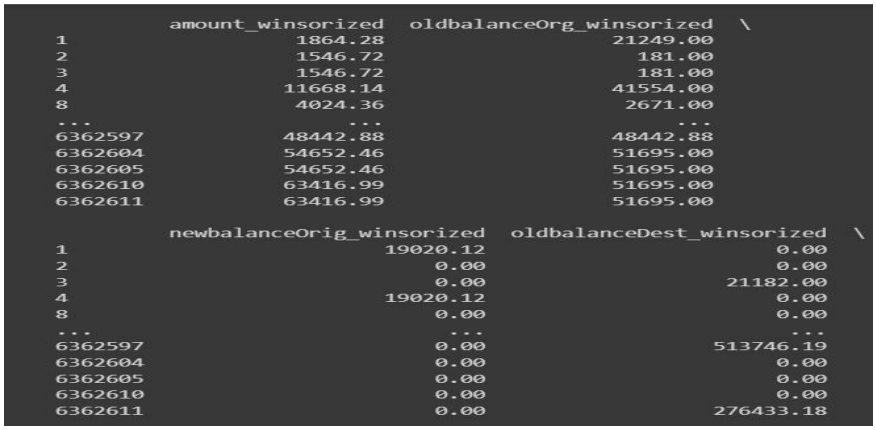

```
plt.boxplot(data["newbalanceDest"])
plt.show()
```



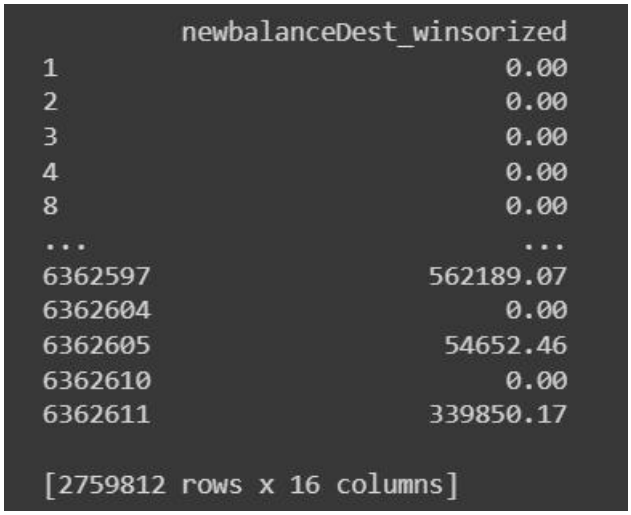

```
#handling outliers using WINSORIZE
numeric_columns = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']
for column in numeric_columns:
    data[column + '_winsorized'] = winsorize(data[column], limits=[0.05, 0.05])
print(data)
```



| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | PAYMENT | 1864.28 | C1666544295 | 21249.00 | 19384.72 | M2044282225 | 0.00 | 0.00 | 0 | 0 |
| 2 | 1 | TRANSFER | 181.00 | C1305486145 | 181.00 | 0.00 | C553264065 | 0.00 | 0.00 | 1 | 0 |
| 3 | 1 | CASH_OUT | 181.00 | C840083671 | 181.00 | 0.00 | C38997010 | 21182.00 | 0.00 | 1 | 0 |
| 4 | 1 | PAYMENT | 11668.14 | C2048537720 | 41554.00 | 29885.86 | M1230701703 | 0.00 | 0.00 | 0 | 0 |
| 8 | 1 | PAYMENT | 4024.36 | C1265012928 | 2671.00 | 0.00 | M1176932104 | 0.00 | 0.00 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6362597 | 741 | CASH_OUT | 48442.88 | C1706094385 | 48442.88 | 0.00 | C2109905271 | 513746.19 | 562189.07 | 1 | 0 |
| 6362604 | 742 | TRANSFER | 54652.46 | C1674778854 | 54652.46 | 0.00 | C1930074465 | 0.00 | 0.00 | 1 | 0 |
| 6362605 | 742 | CASH_OUT | 54652.46 | C43545501 | 54652.46 | 0.00 | C830041824 | 0.00 | 54652.46 | 1 | 0 |
| 6362610 | 742 | TRANSFER | 63416.99 | C778071008 | 63416.99 | 0.00 | C1812552860 | 0.00 | 0.00 | 1 | 0 |
| 6362611 | 742 | CASH_OUT | 63416.99 | C994950684 | 63416.99 | 0.00 | C1662241365 | 276433.18 | 339850.17 | 1 | 0 |



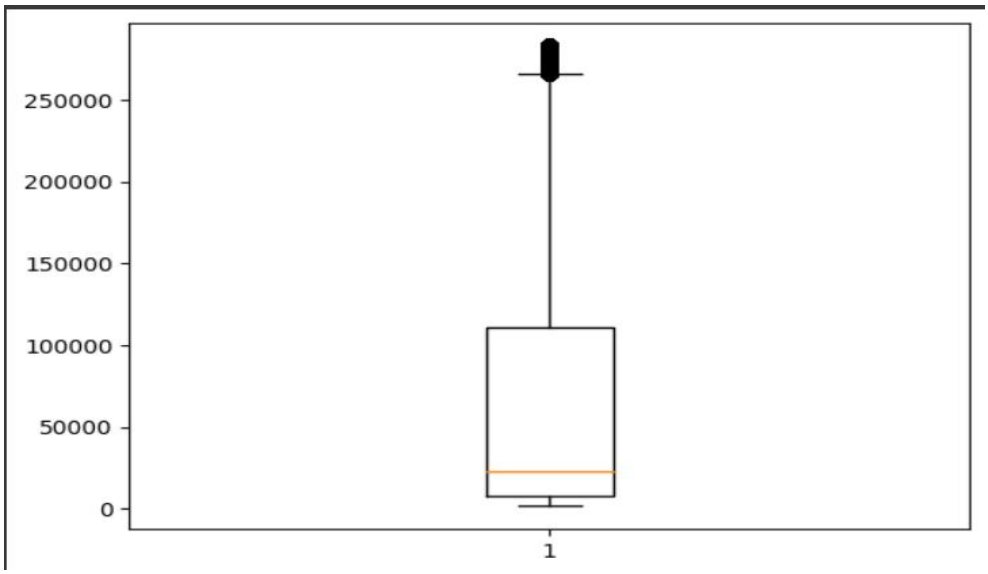
| | amount_winsorized | oldbalanceOrg_winsorized | newbalanceOrig_winsorized | oldbalanceDest_winsorized |
|---|---|---|---|---|
| 1 | 1864.28 | 21249.00 | 19020.12 | 0.00 |
| 2 | 1546.72 | 181.00 | 0.00 | 0.00 |
| 3 | 1546.72 | 181.00 | 0.00 | 21182.00 |
| 4 | 11668.14 | 41554.00 | 19020.12 | 0.00 |
| 8 | 4024.36 | 2671.00 | 0.00 | 0.00 |
| ... | ... | ... | ... | ... |
| 6362597 | 48442.88 | 48442.88 | 0.00 | 513746.19 |
| 6362604 | 54652.46 | 51695.00 | 0.00 | 0.00 |
| 6362605 | 54652.46 | 51695.00 | 0.00 | 0.00 |
| 6362610 | 63416.99 | 51695.00 | 0.00 | 0.00 |
| 6362611 | 63416.99 | 51695.00 | 0.00 | 276433.18 |



| | newbalanceDest_winsorized |
|---|---|
| 1 | 0.00 |
| 2 | 0.00 |
| 3 | 0.00 |
| 4 | 0.00 |
| 8 | 0.00 |
| ... | ... |
| 6362597 | 562189.07 |
| 6362604 | 0.00 |
| 6362605 | 54652.46 |
| 6362610 | 0.00 |
| 6362611 | 339850.17 |

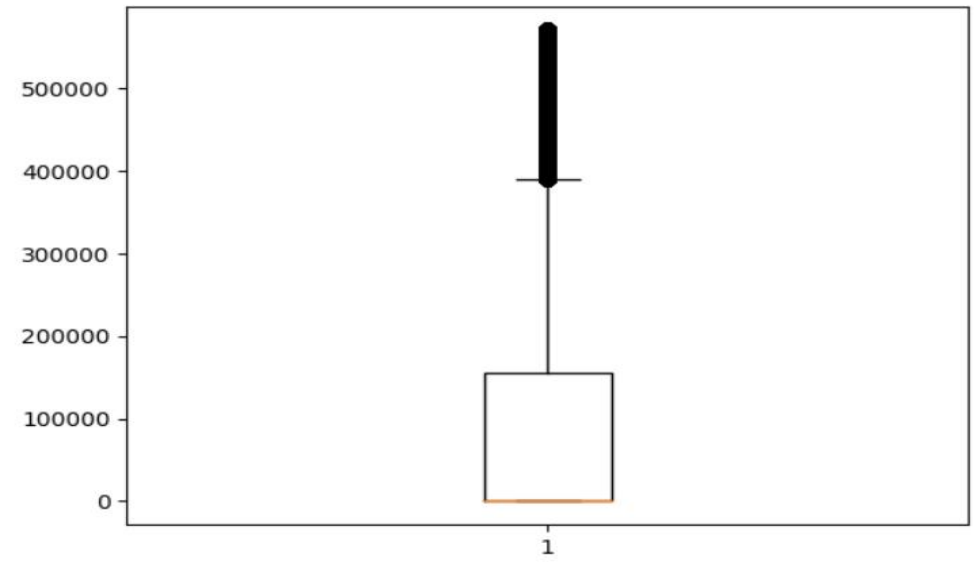
[2759812 rows x 16 columns]

```
plt.boxplot(data["amount_winsorized"])

plt.show()
```



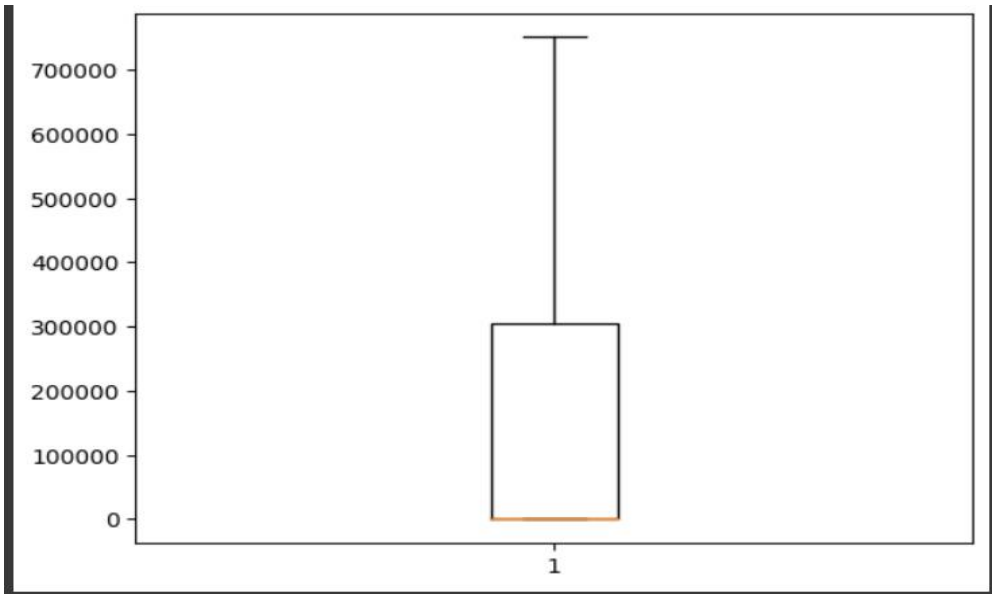

plt.boxplot(data["oldbalanceDest_winsorized"])

plt.show()



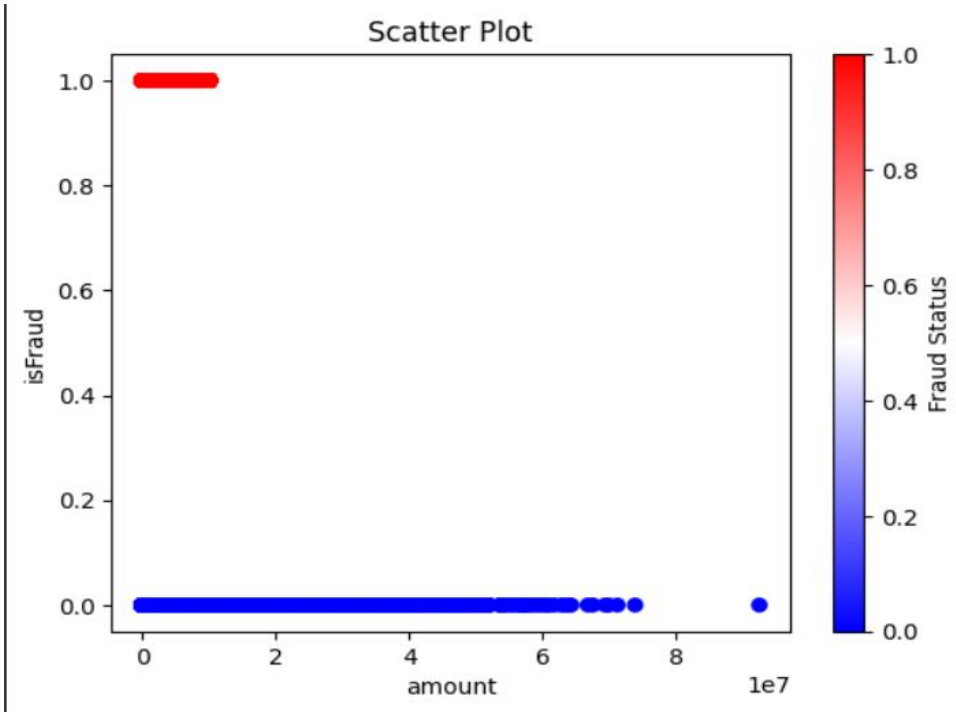

plt.boxplot(data["newbalanceDest_winsorized"])

plt.show()



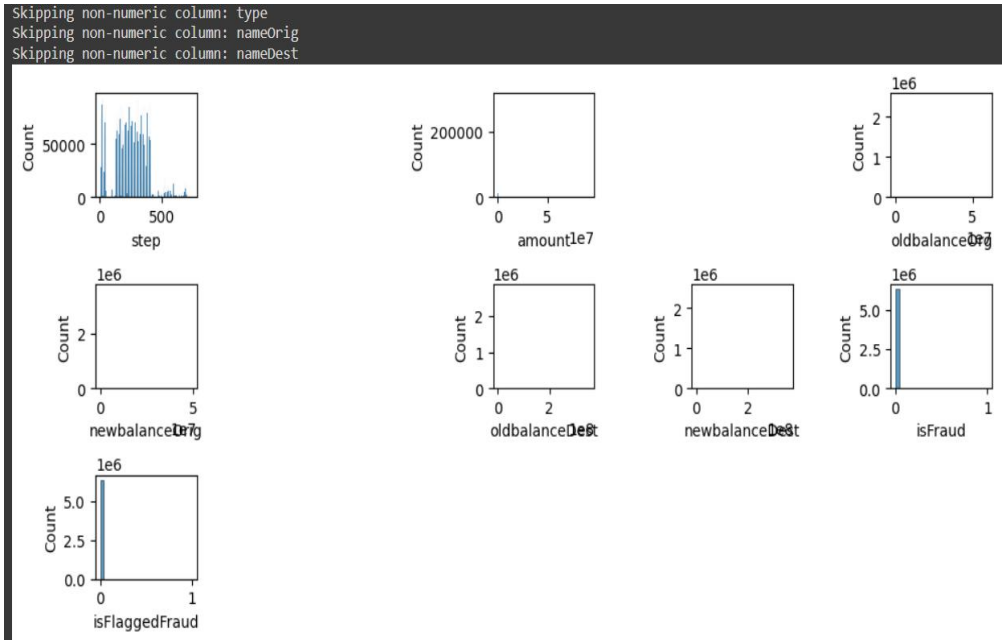

Creating a scatter plot using the 'amount' as the x-axis and 'isFraud' as the y-axis. The color of each point is determined by the 'isFraud' column, using a colormap

```
plt.scatter(data['amount'], data['isFraud'], c=data['isFraud'], cmap='bwr')

plt.title('Scatter Plot')

plt.xlabel('amount')

plt.ylabel('isFraud')

plt.colorbar(label='Fraud Status')

plt.show()
```



This code generates a grid of histograms for numeric columns in the dataset, providing a quick overview of the distribution of each numerical variable. Non-numeric columns are skipped, and the layout is organized in a 3x5 grid for better visualization.

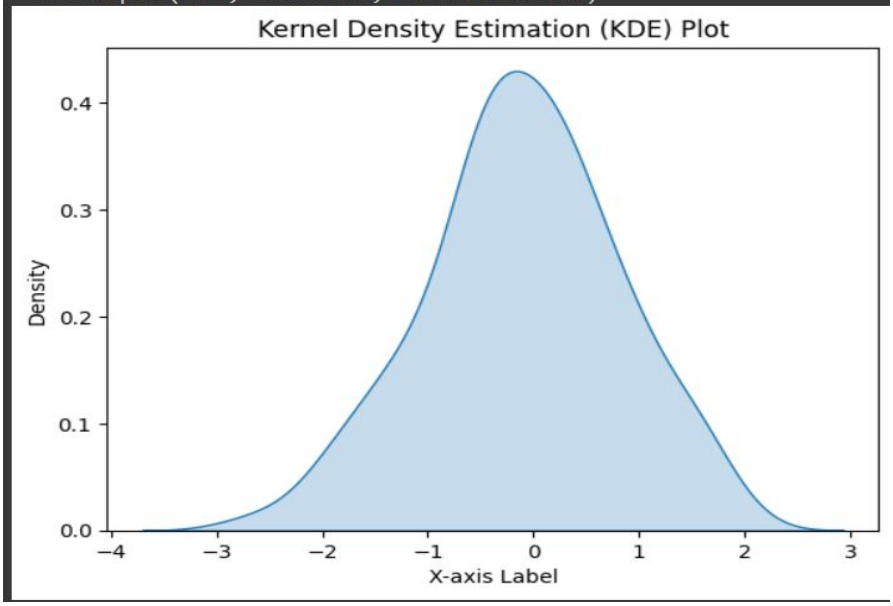
```
# checking numerical features distribution

plt.figure(figsize=(10, 5))

plotnumber = 1

for column in data.columns:

    if plotnumber <= 14:

        if data[column].dtype in [np.float64, np.int64]:

            ax = plt.subplot(3, 5, plotnumber)

            sns.histplot(data[column])

            plt.xlabel(column)

        else:

            print(f"Skipping non-numeric column: {column}")


    plotnumber += 1

plt.tight_layout()

plt.show()
```



This code generates a Kernel Density Estimation (KDE) plot for a dataset of 100 randomly generated numbers. The shaded area represents the estimated probability density function, providing insights into the underlying distribution.

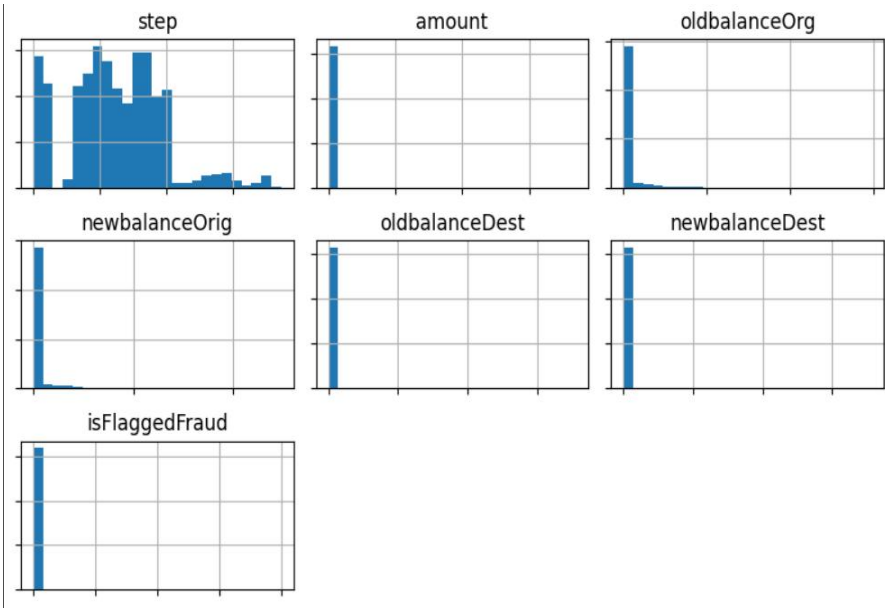
```
np.random.seed(42)

data = np.random.randn(100)

sns.kdeplot(data, shade=True, label='KDE Plot')

plt.xlabel('X-axis Label')

plt.ylabel('Density')

plt.title('Kernel Density Estimation (KDE) Plot')

plt.show()
```

```
`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data, shade=True, label='KDE Plot')
```



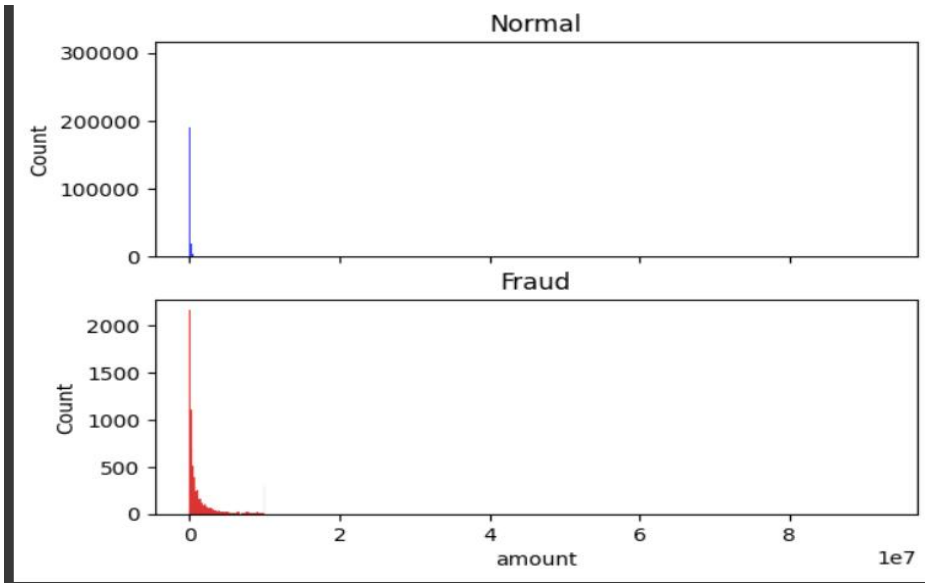

Another way to get a feel of the data is to plot a histogram for each numerical attribute. For step to isFlaggedFraud, the distributions of most of the PCA components are Gaussian, and may be centered around zero, suggesting that the attributes have been normalized by the PCA transformation.

```
ax = data.drop('isFraud', axis=1).hist(bins=25, figsize=(8, 5))

for axis in ax.flatten():

    axis.set_xticklabels([])

    axis.set_yticklabels([])

plt.show()
```




Both normal and fraud transactions happen across all the time span. The normal transations show some pattern. Compared with normal transactions above, the fraud's pattern is not very clear

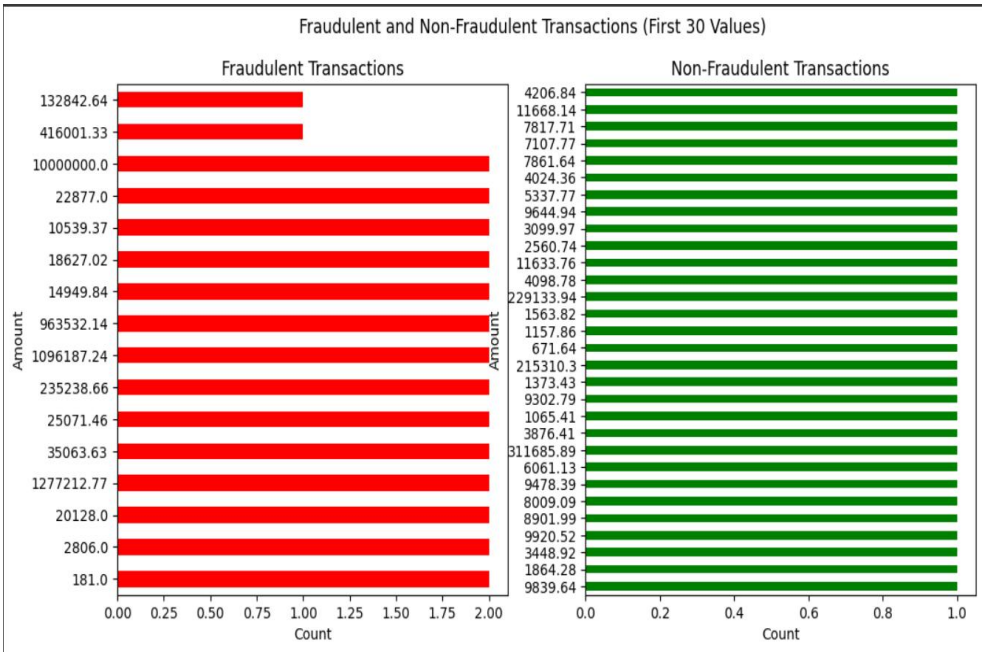
```
normal = data[data['isFraud'] == 0]

fraud = data[data['isFraud'] == 1]


fig, axes = plt.subplots(2, 1, sharex=True)


# Plot histogram for 'Time' in the 'Normal' case

axes[0].set_title('Normal')

sns.histplot(ax=axes[0], data=normal, x="amount", color='b')


# Plot histogram for 'Time' in the 'Fraud' case

axes[1].set_title('Fraud')

sns.histplot(ax=axes[1], data=fraud, x="amount", color='r')

plt.show()
```




The side-by-side bar chart compares the distribution of transaction amounts for the first 30 values in both fraudulent and non-fraudulent transactions. The left plot ('Fraudulent Transactions') indicates the count of unique amounts in red, while the right plot ('Non-Fraudulent Transactions') shows the count in green. This visualization provides insights into the frequency and variation of transaction amounts for both fraudulent and non-fraudulent cases in the sampled data

```
fig, axes = plt.subplots(1, 2, figsize=(12, 6))


# Plotting for fraudulent transactions
data[data['isFraud'] == 1].head(30)['amount'].value_counts().plot(kind="barh", ax=axes[0], color='r')
axes[0].set_title('Fraudulent Transactions')
axes[0].set_xlabel('Count')
axes[0].set_ylabel('Amount')


# Plotting for non-fraudulent transactions
data[data['isFraud'] == 0].head(30)['amount'].value_counts().plot(kind="barh", ax=axes[1], color='g')
axes[1].set_title('Non-Fraudulent Transactions')
axes[1].set_xlabel('Count')
axes[1].set_ylabel('Amount')


plt.suptitle('Fraudulent and Non-Fraudulent Transactions (First 30 Values)')
```

plt.show()

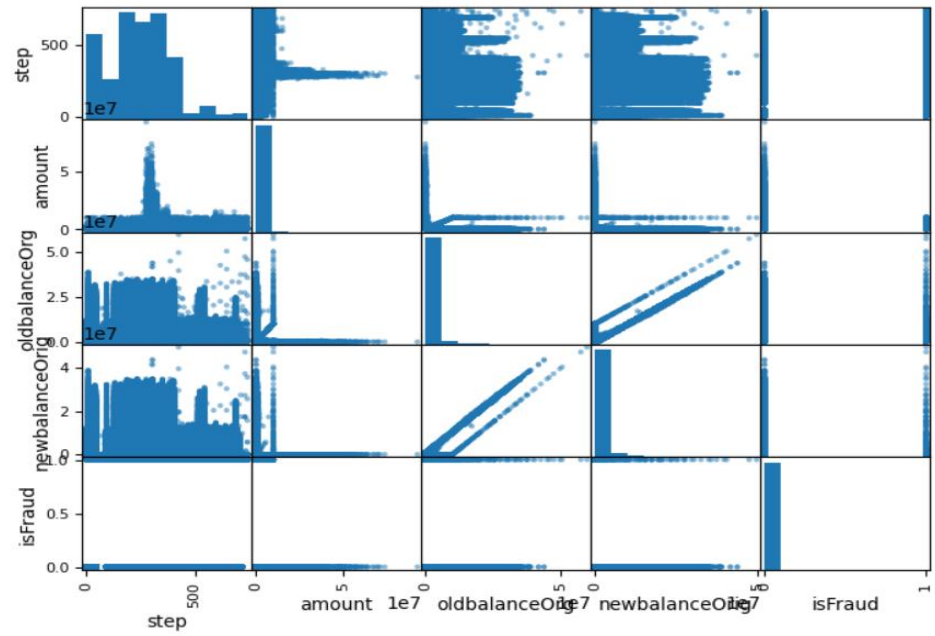



This scatter matrix displays pairwise relationships among the selected columns step, amount, oldbalanceOrg, newbalanceOrig, isFraud

columns_of_interest = ['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'isFraud']

scatter_matrix(data[columns_of_interest], alpha=0.5, figsize=(10, 8), diagonal='hist')

plt.show()

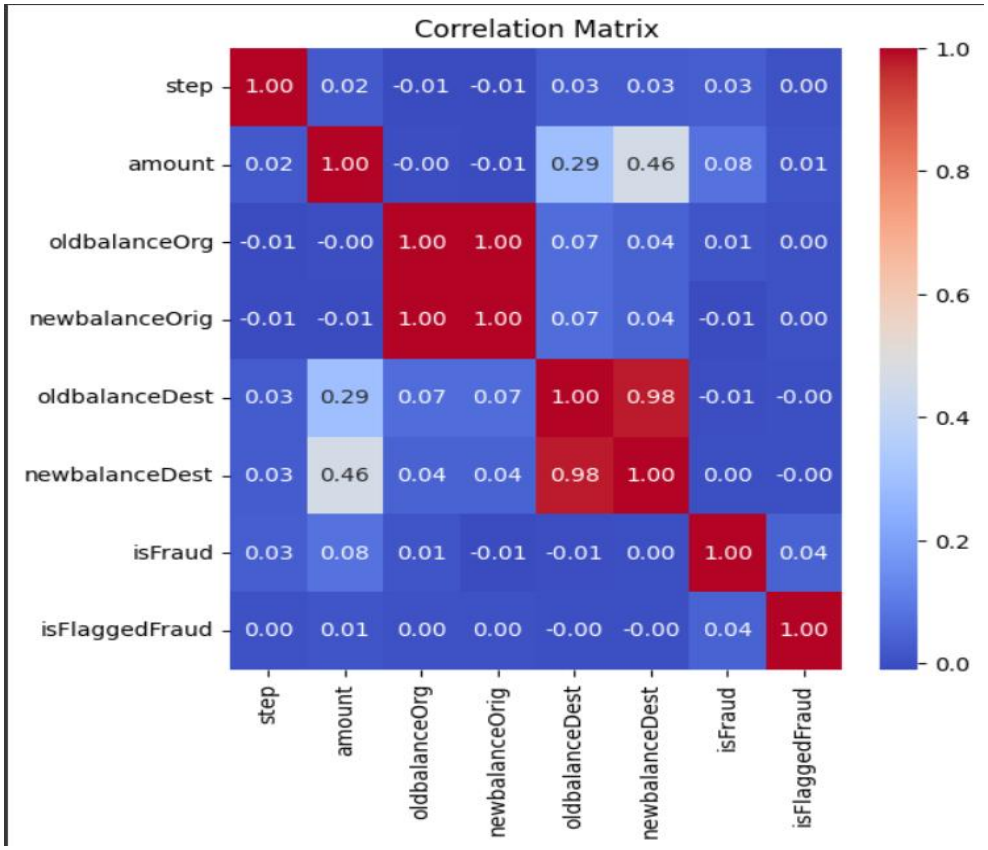


Here, we used Pair Plot because each variable in the dataset is compared with every other variable.

```
new_data = data.head(10000)

sns.pairplot(new_data, plot_kws={'alpha': 0.5},height=2, aspect=1.5)

plt.show()
```

To visualize the correlation matrix of a dataset we used Heatmap.Heatmap is a graphical representation of data where values in a matrix are represented as colors.

```
# Heatmap: Correlation matrix

correlation_matrix = data.corr()

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

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Matrix')

plt.show()
```




By using confusion matrix and classification report, we are performing on the test set, providing insights into the model's ability to accurately identify fraud (1) and non-fraud (0) instances.

```
X = data[['amount']]
y = data['isFraud']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = IsolationForest(contamination=0.1, random_state=42)
model.fit(X_train_scaled)
y_pred = model.predict(X_test_scaled)
y_pred_binary = [1 if pred == -1 else 0 for pred in y_pred]
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_binary))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_binary))
```

```
Confusion Matrix:
[[1144441  126463]
 [    746     874]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.90      0.95   1270904
           1       0.01      0.54      0.01      1620

    accuracy                           0.90   1272524
   macro avg       0.50      0.72      0.48   1272524
weighted avg       1.00      0.90      0.95   1272524
```

```
def classify_fraud(row):
    if row['isFraud'] == 1:
        return 'Fraud'
    else:
        return 'Non-Fraud'
data['isFraud'] = data.apply(classify_fraud, axis=1)
print(data['isFraud'])
```

```
0          Non-Fraud
1          Non-Fraud
2          Non-Fraud
3          Non-Fraud
4          Non-Fraud
             ...
6362615    Non-Fraud
6362616    Non-Fraud
6362617    Non-Fraud
6362618    Non-Fraud
6362619    Non-Fraud
Name: isFraud, Length: 6362620, dtype: object
```

```
non_fraud_rows = data[data['isFraud'] == 0]

fraud_rows = data[data['isFraud'] == 1]

print("Non-Fraud values:")

print(non_fraud_rows)

print("\nFraud values:")

print(fraud_rows)
```

```
Non-Fraud values:
Empty DataFrame
Columns: [step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest, isFraud, isFlaggedFraud, fraud_class]
Index: []

Fraud values:
Empty DataFrame
Columns: [step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig, nameDest, oldbalanceDest, newbalanceDest, isFraud, isFlaggedFraud, fraud_class]
Index: []
```

```
x_df=[]

y_df=[]


x_df=data.drop(['isFraud'],axis=1)

y_df=data["isFraud"]

len(data)

ind_col=x_df.columns

length=data.shape[0]


for i in range(4):

  var=y_df[i]

  if(var==1):

    x_df.append(x_df)

    print(var,x_df.iloc[i,])

  else:
```

```
print(var,x_df.iloc[i,])
```

```
0 step                        1
type                    PAYMENT
amount                  9839.64
nameOrig             C1231006815
oldbalanceOrg          170136.0
newbalanceOrig         160296.36
nameDest             M1979787155
oldbalanceDest             0.0
newbalanceDest             0.0
isFlaggedFraud               0
Name: 0, dtype: object
0 step                        1
type                    PAYMENT
amount                  1864.28
nameOrig             C1666544295
oldbalanceOrg          21249.0
newbalanceOrig         19384.72
nameDest             M2044282225
oldbalanceDest             0.0
newbalanceDest             0.0
isFlaggedFraud               0
```

```
1 step
type                    TRANSFER
amount                     181.0
nameOrig             C1305486145
oldbalanceOrg              181.0
newbalanceOrig               0.0
nameDest             C553264065
oldbalanceDest               0.0
newbalanceDest               0.0
isFlaggedFraud               0
Name: 2, dtype: object
1 step                        1
type                     CASH_OUT
amount                     181.0
nameOrig             C840083671
oldbalanceOrg              181.0
newbalanceOrig               0.0
nameDest             C38997010
oldbalanceDest           21182.0
newbalanceDest               0.0
isFlaggedFraud               0
Name: 3, dtype: object
```

len(data)

```
6362620
```

data.head(4)

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | PAYMENT | 9839.64 | C1231006815 | 170136.0 | 160296.36 | M1979787155 | 0.0 | 0.0 | 0 | 0 |
| 1 | 1 | PAYMENT | 1864.28 | C1666544295 | 21249.0 | 19384.72 | M2044282225 | 0.0 | 0.0 | 0 | 0 |
| 2 | 1 | TRANSFER | 181.00 | C1305486145 | 181.0 | 0.00 | C553264065 | 0.0 | 0.0 | 1 | 0 |
| 3 | 1 | CASH_OUT | 181.00 | C840083671 | 181.0 | 0.00 | C38997010 | 21182.0 | 0.0 | 1 | 0 |

```
random_data = []
# Selecting 15,000 random samples
random_samples = np.random.choice(data.index, size=15000, replace=False)

for i in random_samples:
    label = data.loc[i, 'isFraud']
    random_data.append(data.loc[i, :])
random_data = pd.DataFrame(random_data)
print(random_data)
```

```
         step      type      amount     nameOrig  oldbalanceOrg  \
1640081   157   PAYMENT    17131.25  C2057763801      706288.82
1520712   153  TRANSFER  1084092.43   C663148259           0.00
1300767   136  TRANSFER     7295.26  C1415385963           0.00
1943910   177  CASH_OUT   219137.03  C1585181145           0.00
909492     43   CASH_IN    81628.50  C1881519428      160819.93
...       ...       ...         ...          ...           ...
333718     16   PAYMENT    11502.00   C455856147           0.00
1158166   131   PAYMENT     7559.39  C1193613165           0.00
2329339   188   CASH_IN   152018.89   C553607766      254418.76
425760     18   PAYMENT    25781.16  C1278549836           0.00
3555051   260   PAYMENT     6100.25   C412379240       15356.26

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
1640081       689157.57  M1972679676            0.00            0.00        0
1520712            0.00  C1313381818      1179602.94      2070891.54        0
1300767            0.00  C1060928555      2104850.76      2557383.04        0
1943910            0.00  C1828825845      2864275.22      3293748.45        0
909492        242448.42  C1802243408      1500630.36      1419001.86        0
...                 ...          ...             ...             ...      ...
333718             0.00  M1061942061            0.00            0.00        0
1158166            0.00   M284190298            0.00            0.00        0
2329339       406437.65   C980696620        41067.53            0.00        0
425760             0.00   M344673576            0.00            0.00        0
3555051         9256.01  M1950952818            0.00            0.00        0
```

```
          isFlaggedFraud
1640081                0
1520712                0
1300767                0
1943910                0
909492                 0
...                  ...
333718                 0
1158166                0
2329339                0
425760                 0
3555051                0

[15000 rows x 11 columns]
```

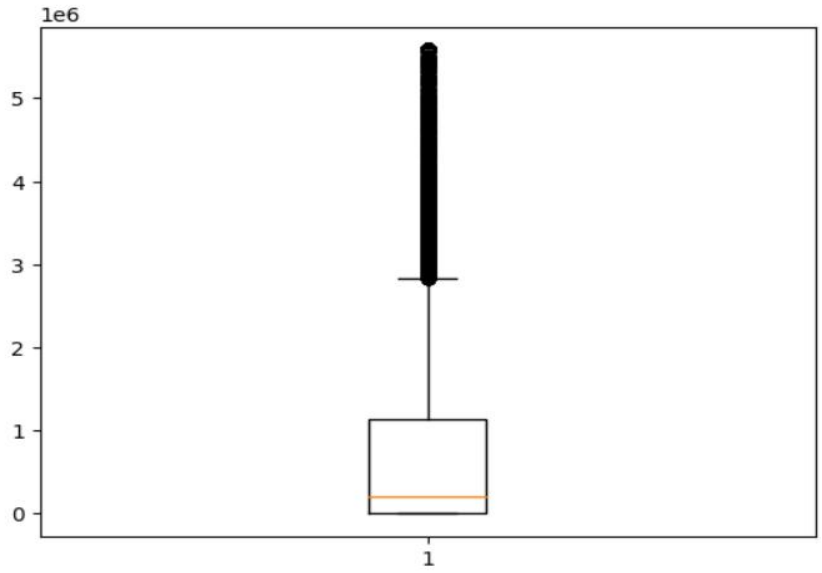
len(random_data)

```
15000
```

random_data['normalisednewbalanceDest']=winsorize(random_data["newbalanceDest"], limits=[0.05, 0.05])

plt.boxplot(random_data["normalisednewbalanceDest"])

plt.show()




columns_to_label_encode = ['type','nameOrig', 'nameDest']

label_encoder = LabelEncoder()

for column in columns_to_label_encode:

    random_data[column + '_encoded'] = label_encoder.fit_transform(random_data[column])

encoded = random_data.drop(columns=columns_to_label_encode)

print(encoded)

```
         step      amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
1366043   138    29683.45           0.00            0.00            0.00
5524443   381     2601.89      935543.75       932941.86       603570.04
3547935   260   456830.95      105867.00            0.00            0.00
645036     35   175821.76       23867.00            0.00       591846.73
3576469   261    15953.15       19318.00         3364.85            0.00
...       ...         ...            ...             ...             ...
3401413   255   176377.29       11621.00       187998.29      1391652.23
4565085   327    53312.95         287.00            0.00            0.00
2344837   189    12180.70     1266013.65      1278194.36       151782.59
1881224   164   125525.36       23635.00            0.00       379393.93
4772286   335    63077.59       90389.00        27311.41            0.00

         newbalanceDest  isFraud  isFlaggedFraud  type_encoded  \
1366043            0.00        0               0             3
5524443       606171.93        0               0             1
3547935       456830.95        0               0             4
645036        767668.49        0               0             1
3576469           0.00        0               0             3
...                  ...      ...             ...           ...
3401413      1215274.93        0               0             0
4565085        53312.95        0               0             1
2344837       139601.89        0               0             0
1881224       504919.29        0               0             1
4772286        63077.59        0               0             4

         nameOrig_encoded  nameDest_encoded
1366043             12022             11271
5524443              9671              7181
3547935               787              6302
645036               5682              8590
3576469              9712             12022
...                   ...               ...
3401413              2156              4209
4565085              8292              2100
2344837             12680              4757
1881224             14213               886
4772286             13144              5709
```

random_data.head()

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2071126 | 181 | CASH_OUT | 219626.67 | C322687638 | 100057.0 | 0.0 | C1668430365 | 0.0 | 219626.67 | 0 | 0 |
| 2817355 | 225 | PAYMENT | 47414.04 | C1289239103 | 0.0 | 0.0 | M1766511204 | 0.0 | 0.00 | 0 | 0 |
| 3590604 | 262 | CASH_OUT | 96921.22 | C554595301 | 21506.0 | 0.0 | C1362505632 | 25490.6 | 122411.82 | 0 | 0 |
| 3915939 | 284 | PAYMENT | 12945.10 | C286946403 | 13005.0 | 59.9 | M936907676 | 0.0 | 0.00 | 0 | 0 |
| 280416 | 15 | PAYMENT | 18596.89 | C1693319373 | 4179.0 | 0.0 | M848645043 | 0.0 | 0.00 | 0 | 0 |

print(encoded)

```
         step     amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
1366043   138   29683.45           0.00            0.00            0.00
5524443   381    2601.89      935543.75       932941.86       603570.04
3547935   260  456830.95      105867.00            0.00            0.00
645036     35  175821.76       23867.00            0.00       591846.73
3576469   261   15953.15       19318.00         3364.85            0.00
...       ...        ...            ...             ...             ...
3401413   255  176377.29       11621.00       187998.29      1391652.23
4565085   327   53312.95         287.00            0.00            0.00
2344837   189   12180.70     1266013.65      1278194.36       151782.59
1881224   164  125525.36       23635.00            0.00       379393.93
4772286   335   63077.59       90389.00        27311.41            0.00

         newbalanceDest  isFraud  isFlaggedFraud  type_encoded  \
1366043            0.00        0               0             3
5524443       606171.93        0               0             1
3547935       456830.95        0               0             4
645036        767668.49        0               0             1
3576469           0.00        0               0             3
...                ...      ...             ...           ...
3401413      1215274.93        0               0             0
4565085        53312.95        0               0             1
2344837       139601.89        0               0             0
1881224       504919.29        0               0             1
4772286        63077.59        0               0             4

         nameOrig_encoded  nameDest_encoded
1366043             12022             11271
5524443              9671              7181
3547935               787              6302
645036               5682              8590
3576469              9712             12022
...                   ...               ...
3401413              2156              4209
4565085              8292              2100
2344837             12680              4757
1881224             14213               886
4772286             13144              5709
```

smote=SMOTE(random_state=42)

model_data=random_data.drop(['isFraud','type','nameOrig','nameDest'],axis=1)

X_train_resampled , Y_train_resampled=smote.fit_resample(model_data,random_data["isFraud"])

smote = SMOTE(random_state=42)

model_data = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)

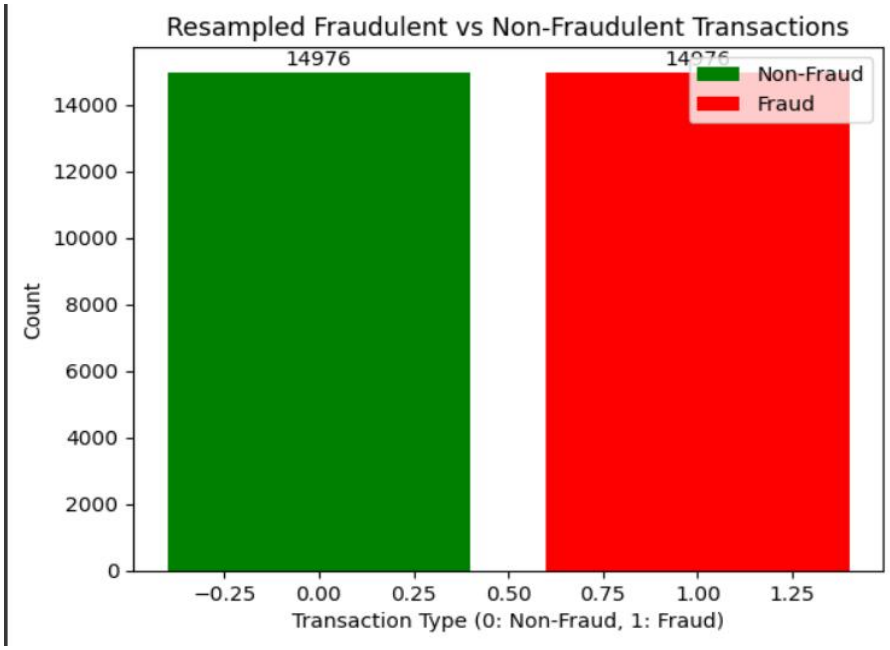
X_train_resampled, Y_train_resampled = smote.fit_resample(model_data, random_data["isFraud"])

resampled_data = pd.DataFrame(X_train_resampled, columns=model_data.columns)

resampled_data['isFraud'] = Y_train_resampled

fraud_counts_resampled = resampled_data['isFraud'].value_counts()

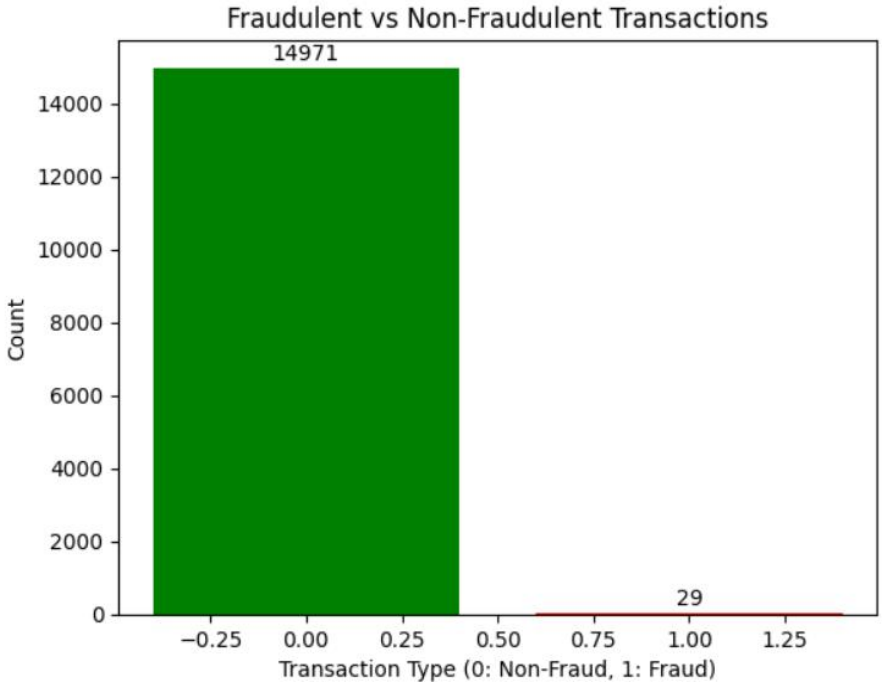
```
plt.bar(fraud_counts_resampled.index, fraud_counts_resampled, color=['green', 'red'], label=['Non-Fraud', 'Fraud'])

plt.title('Resampled Fraudulent vs Non-Fraudulent Transactions')

plt.xlabel('Transaction Type (0: Non-Fraud, 1: Fraud)')

plt.ylabel('Count')


for i, count in enumerate(fraud_counts_resampled):
    plt.text(i, count + 100, str(count), ha='center', va='bottom')

plt.legend()

plt.show()
```




```
count_0=0

count_1=0

for i in Y_train_resampled:

  if i==1:

    count_0=count_0+1

  else:

    count_1=count_1+1

print(count_0)

print(count_1)
```

```
14976
14976
```

```
fraud_counts = random_data['isFraud'].value_counts()
plt.bar(fraud_counts.index, fraud_counts, color=['green', 'red'])
plt.title('Fraudulent vs Non-Fraudulent Transactions')
plt.xlabel('Transaction Type (0: Non-Fraud, 1: Fraud)')
plt.ylabel('Count')
for i, count in enumerate(fraud_counts):
    plt.text(i, count + 100, str(count), ha='center', va='bottom')
plt.show()
```




MODEL

```
X_train = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
Y_train = random_data['isFraud']
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=42)
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_train shape: (12000, 7)
Y_train shape: (12000,)
X_test shape: (3000, 7)
Y_test shape: (3000,)
```

X_train

| | step | amount | oldbalanceOrg | newbalanceOrig | oldbalanceDest | newbalanceDest | isFlaggedFraud |
|---|---|---|---|---|---|---|---|
| **23108** | 8 | 5626.01 | 5000164.80 | 5005790.82 | 21619.00 | 0.00 | 0 |
| **1616269** | 156 | 10944.80 | 39853.00 | 28908.20 | 0.00 | 0.00 | 0 |
| **2908913** | 229 | 81581.62 | 22449.00 | 0.00 | 262489.46 | 344071.08 | 0 |
| **997990** | 45 | 6167.58 | 8356.00 | 2188.42 | 0.00 | 0.00 | 0 |
| **1947080** | 177 | 223192.76 | 21516.00 | 0.00 | 1646967.91 | 1870160.67 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **5286112** | 373 | 12867.85 | 33046.00 | 20178.15 | 0.00 | 0.00 | 0 |
| **5105659** | 355 | 130127.67 | 0.00 | 0.00 | 460877.40 | 591005.07 | 0 |
| **621816** | 34 | 152734.29 | 8708198.13 | 8860932.42 | 411379.28 | 258644.99 | 0 |
| **4550347** | 327 | 254124.28 | 12591.00 | 0.00 | 105474.11 | 359598.39 | 0 |
| **2209446** | 186 | 286564.63 | 2200593.59 | 2487158.22 | 796360.61 | 509795.98 | 0 |

12000 rows × 7 columns

Y_train

```
23108      0
1616269    0
2908913    0
997990     0
1947080    0
          ..
5286112    0
5105659    0
621816     0
4550347    0
2209446    0
Name: isFraud, Length: 12000, dtype: int64
```

#LogisticRegression

model = LogisticRegression(random_state=42)

model.fit(X_train, Y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)

conf_matrix = confusion_matrix(Y_test, y_pred)

classification_rep = classification_report(Y_test, y_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)

```
Accuracy: 1.0
Confusion Matrix:
 [[2995    0]
 [   0    5]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       1.00      1.00      1.00         5

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=42)

model = LogisticRegression()

model.fit(X, y)

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

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3)

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Paired)

plt.title('Logistic Regression Decision Boundary')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()


#DecisionTreeClassifier

model = DecisionTreeClassifier(random_state=42)

model.fit(X_train, Y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)

conf_matrix = confusion_matrix(Y_test, y_pred)

classification_rep = classification_report(Y_test, y_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)

```
Accuracy: 0.9993333333333333
Confusion Matrix:
 [[2994    1]
 [   1    4]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.80      0.80      0.80         5

    accuracy                           1.00      3000
   macro avg       0.90      0.90      0.90      3000
weighted avg       1.00      1.00      1.00      3000
```

X = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)

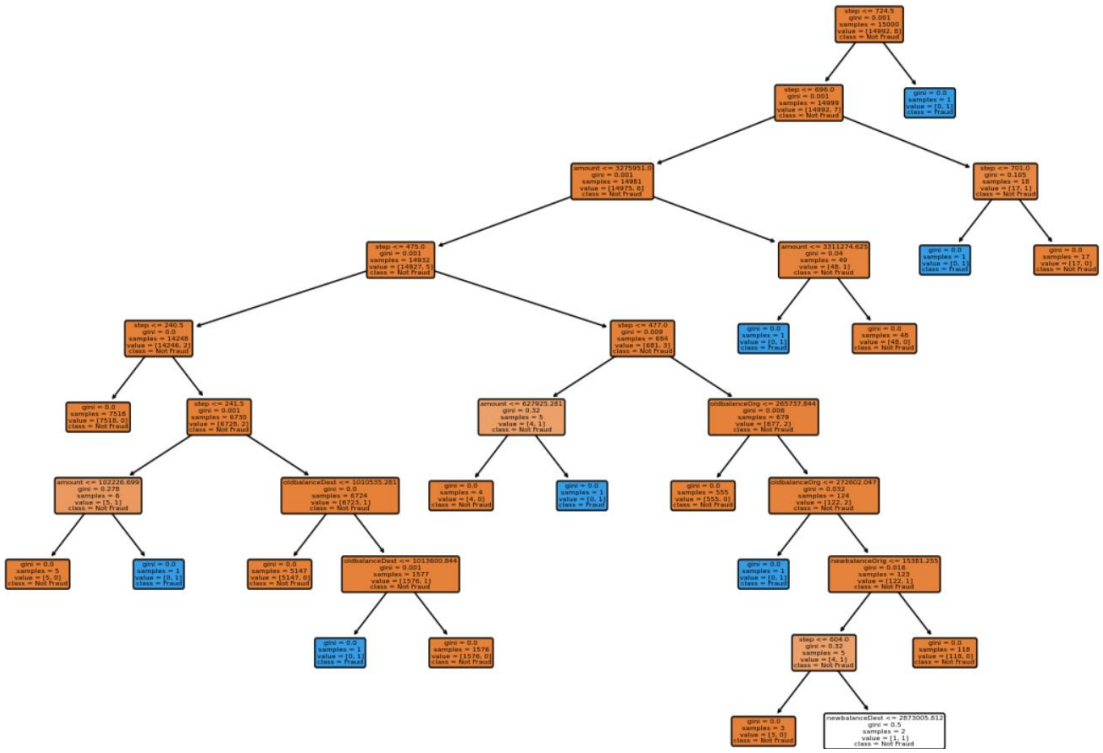
y = random_data['isFraud']

model = DecisionTreeClassifier(random_state=42)

model.fit(X, y)

plt.figure(figsize=(15, 10))

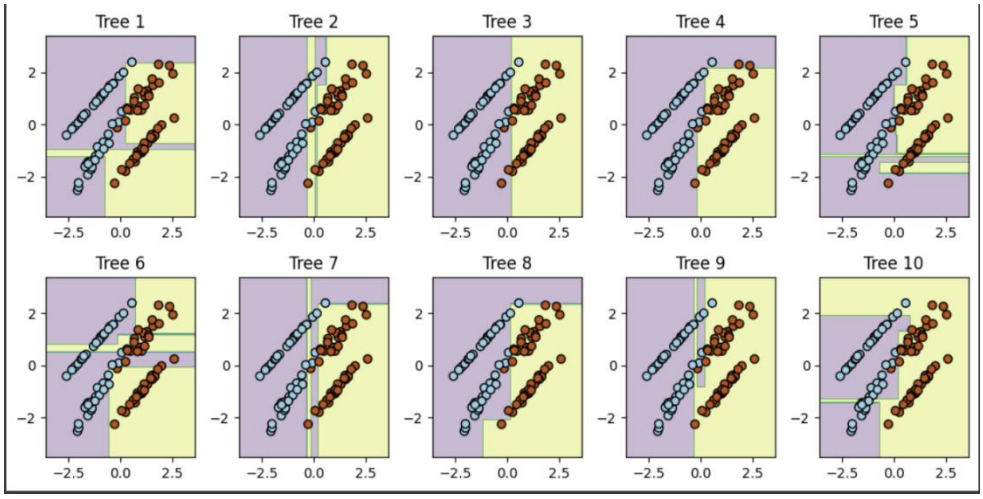
```
plot_tree(model, feature_names=X.columns, class_names=['Not Fraud', 'Fraud'], filled=True, rounded=True)

plt.show()
```



```
#RandomForestClassifier

model = RandomForestClassifier(random_state=42)

model.fit(X_train, Y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)

conf_matrix = confusion_matrix(Y_test, y_pred)

classification_rep = classification_report(Y_test, y_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9983333333333333
Confusion Matrix:
 [[2995    0]
 [   5    0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.00      0.00      0.00         5

    accuracy                           1.00      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       1.00      1.00      1.00      3000
```

```
X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=42)

model = RandomForestClassifier(n_estimators=10, random_state=42)

model.fit(X, y)

plt.figure(figsize=(10, 5))

for tree_idx, tree in enumerate(model.estimators_):
    plt.subplot(2, 5, tree_idx + 1)
     x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
    Z = tree.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Paired)
    plt.title(f'Tree {tree_idx + 1}')
plt.tight_layout()
plt.show()
```



#SVC

model = SVC(random_state=42)

model.fit(X_train, Y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)

conf_matrix = confusion_matrix(Y_test, y_pred)

classification_rep = classification_report(Y_test, y_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)

```
Accuracy: 0.9983333333333333
Confusion Matrix:
 [[2995    0]
 [   5    0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.00      0.00      0.00         5

    accuracy                           1.00      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       1.00      1.00      1.00      3000
```

X, y = make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0, random_state=42)

model = SVC(kernel='linear', C=1)

model.fit(X, y)

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

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3)

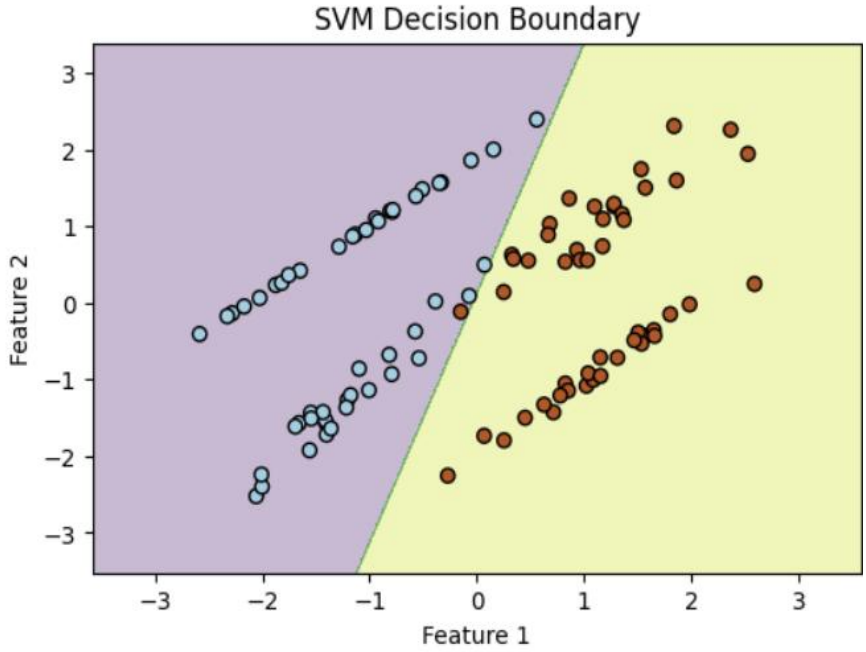
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', cmap=plt.cm.Paired)

plt.title('SVM Decision Boundary')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()




#MPL Classifer

model = MLPClassifier(random_state=42)

model.fit(X_train, Y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)

conf_matrix = confusion_matrix(Y_test, y_pred)

classification_rep = classification_report(Y_test, y_pred)

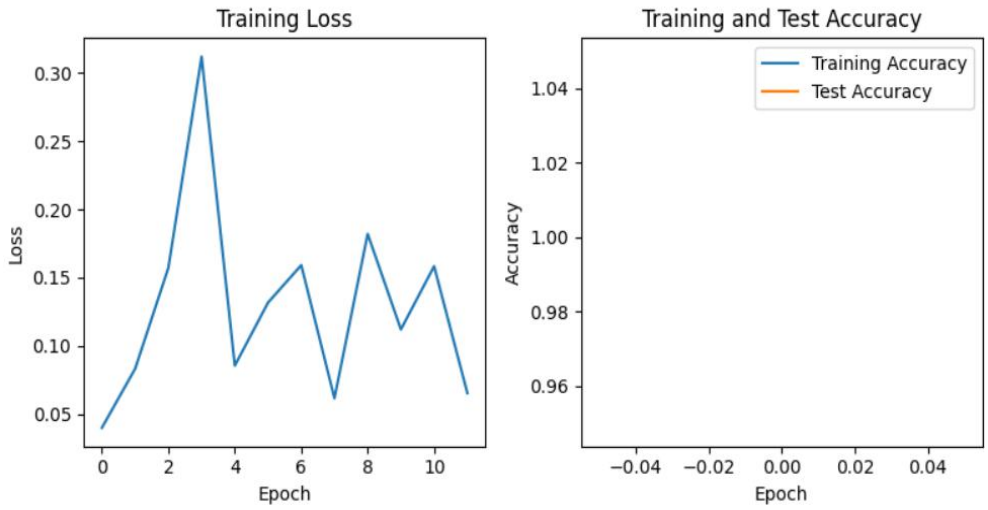
print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)

```
Accuracy: 0.9986666666666667
Confusion Matrix:
 [[2993    2]
 [   2    3]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2995
           1       0.60      0.60      0.60         5

    accuracy                           1.00      3000
   macro avg       0.80      0.80      0.80      3000
weighted avg       1.00      1.00      1.00      3000
```

```
model = MLPClassifier(random_state=42)
model.fit(X_train, Y_train)
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(model.loss_curve_)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.subplot(1, 2, 2)
plt.plot(model.score(X_train, Y_train), label='Training Accuracy')
plt.plot(accuracy, label='Test Accuracy')
plt.title('Training and Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
```



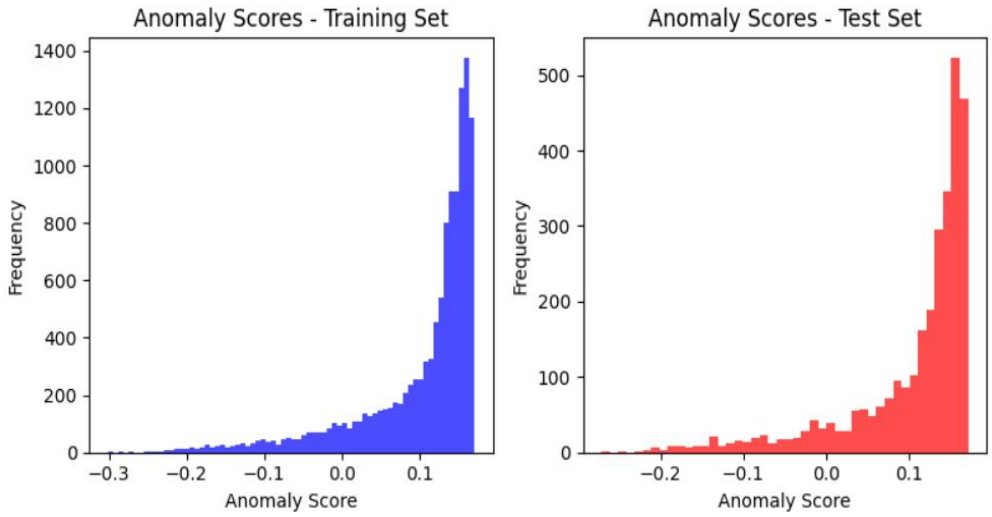

```
#IsolationForest
model = IsolationForest(random_state=42)
```

```
model.fit(X_train)
y_pred = model.predict(X_test)
y_pred_binary = [1 if pred == -1 else 0 for pred in y_pred]
accuracy = accuracy_score(Y_test, y_pred_binary)
conf_matrix = confusion_matrix(Y_test, y_pred_binary)
classification_rep = classification_report(Y_test, y_pred_binary)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.8863333333333333
Confusion Matrix:
 [[2657  338]
 [   3    2]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.89      0.94      2995
           1       0.01      0.40      0.01         5

    accuracy                           0.89      3000
   macro avg       0.50      0.64      0.48      3000
weighted avg       1.00      0.89      0.94      3000
```

```
model = IsolationForest(random_state=42)
model.fit(X_train)
anomaly_scores_train = model.decision_function(X_train)
anomaly_scores_test = model.decision_function(X_test)
threshold = 0.0
y_pred_binary_train = [1 if score < threshold else 0 for score in anomaly_scores_train]
y_pred_binary_test = [1 if score < threshold else 0 for score in anomaly_scores_test]
accuracy_train = accuracy_score(Y_train, y_pred_binary_train)
accuracy_test = accuracy_score(Y_test, y_pred_binary_test)
conf_matrix_test = confusion_matrix(Y_test, y_pred_binary_test)
classification_rep_test = classification_report(Y_test, y_pred_binary_test)
print("Training Accuracy:", accuracy_train)
print("Test Accuracy:", accuracy_test)
print("Confusion Matrix (Test Set):\n", conf_matrix_test)
print("Classification Report (Test Set):\n", classification_rep_test)
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
```

```
plt.hist(anomaly_scores_train, bins='auto', color='blue', alpha=0.7)
plt.title('Anomaly Scores - Training Set')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.subplot(1, 2, 2)
plt.hist(anomaly_scores_test, bins='auto', color='red', alpha=0.7)
plt.title('Anomaly Scores - Test Set')
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```




```
#K Nearest Neighbours
X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train, test_size=0.2, random_state=42)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, Y_train)
Y_pred = knn_classifier.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
conf_matrix = confusion_matrix(Y_test, Y_pred)
classification_rep = classification_report(Y_test, Y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9991666666666666
Confusion Matrix:
 [[2398    0]
 [   2    0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2398
           1       0.00      0.00      0.00         2

    accuracy                           1.00      2400
   macro avg       0.50      0.50      0.50      2400
weighted avg       1.00      1.00      1.00      2400
```

knn_classifier = KNeighborsClassifier(n_neighbors=3)

knn_classifier.fit(X_train, Y_train)

Y_pred = knn_classifier.predict(X_test)

accuracy = accuracy_score(Y_test, Y_pred)

conf_matrix = confusion_matrix(Y_test, Y_pred)

classification_rep = classification_report(Y_test, Y_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)

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

sns.scatterplot(x=X_test.iloc[:, 0], y=X_test.iloc[:, 1], hue=Y_test, palette='viridis', label='True Labels', alpha=0.7)

sns.scatterplot(x=X_test.iloc[:, 0], y=X_test.iloc[:, 1], hue=Y_pred, palette='inferno', marker='X', label='Predicted Labels', alpha=0.5)

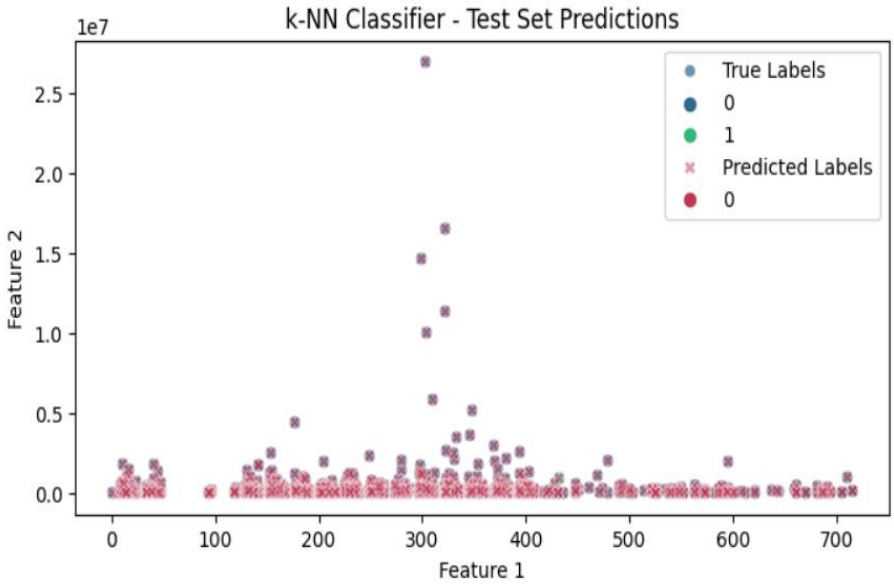
plt.title('k-NN Classifier - Test Set Predictions')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.legend()

plt.show()




#AdaBoost Classifer(Ensemble methods)

model = AdaBoostClassifier(random_state=42)

model.fit(X_train, Y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)

conf_matrix = confusion_matrix(Y_test, y_pred)

classification_rep = classification_report(Y_test, y_pred)

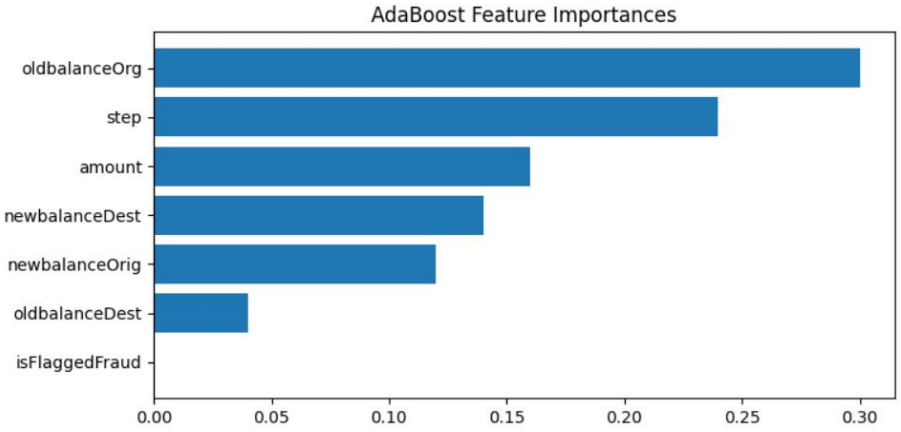
print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)

```
Accuracy: 0.99875
Confusion Matrix:
 [[2397    1]
 [   2    0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2398
           1       0.00      0.00      0.00         2

    accuracy                           1.00      2400
   macro avg       0.50      0.50      0.50      2400
weighted avg       1.00      1.00      1.00      2400
```

```
model = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1), random_state=42)
model.fit(X_train, Y_train)
feature_importances = model.feature_importances_
sorted_indices = np.argsort(feature_importances)
plt.figure(figsize=(8, 4))
plt.barh(range(len(sorted_indices)), feature_importances[sorted_indices], align="center")
plt.yticks(range(len(sorted_indices)), np.array(X_train.columns)[sorted_indices])
plt.title('AdaBoost Feature Importances')
plt.show()
if X_train.shape[1] >= 2:
    x_min, x_max = X_train.iloc[:, 0].min() - 1, X_train.iloc[:, 0].max() + 1
    y_min, y_max = X_train.iloc[:, 1].min() - 1, X_train.iloc[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
     plt.figure(figsize=(10, 6))
    for tree in model.estimators_:
        Z = tree.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, alpha=0.1)
    plt.scatter(X_train.iloc[:, 0], X_train.iloc[:, 1], c=Y_train, cmap='viridis', marker='o')
    plt.title('AdaBoost Decision Boundaries')
    plt.xlabel(X_train.columns[0])
    plt.ylabel(X_train.columns[1])
    plt.show()
else:
    print("Not enough features to visualize decision boundaries.")
```




#One Class Svm

model = OneClassSVM()

model.fit(X_train)

y_pred = model.predict(X_test)

y_pred_binary = [1 if pred == -1 else 0 for pred in y_pred]

accuracy = accuracy_score(Y_test, y_pred_binary)

conf_matrix = confusion_matrix(Y_test, y_pred_binary)

classification_rep = classification_report(Y_test, y_pred_binary)

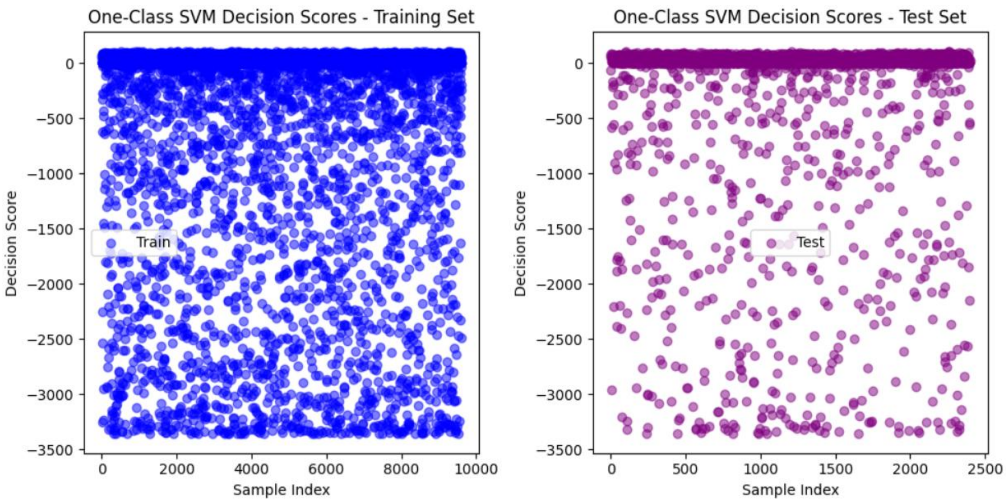
print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", classification_rep)

```
Accuracy: 0.5204166666666666
Confusion Matrix:
 [[1248 1150]
 [   1    1]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.52      0.68      2398
           1       0.00      0.50      0.00         2

    accuracy                           0.52      2400
   macro avg       0.50      0.51      0.34      2400
weighted avg       1.00      0.52      0.68      2400
```

```
model = OneClassSVM()
model.fit(X_train)
decision_scores_train = model.decision_function(X_train)
decision_scores_test = model.decision_function(X_test)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(range(len(X_train)), decision_scores_train, c='blue', label='Train', alpha=0.5)
plt.title('One-Class SVM Decision Scores - Training Set')
plt.xlabel('Sample Index')
plt.ylabel('Decision Score')
plt.legend()
plt.subplot(1, 2, 2)
plt.scatter(range(len(X_test)), decision_scores_test, c='purple', label='Test', alpha=0.5)
plt.title('One-Class SVM Decision Scores - Test Set')
plt.xlabel('Sample Index')
plt.ylabel('Decision Score')
plt.legend()
plt.tight_layout()
plt.show()
```

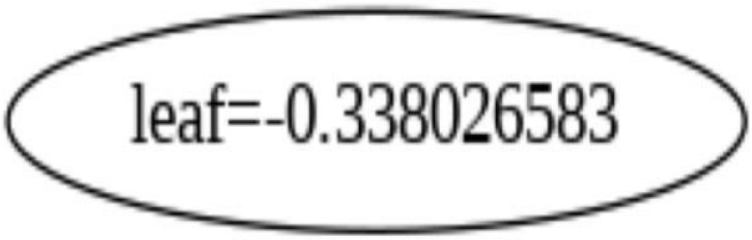


#XG Booster

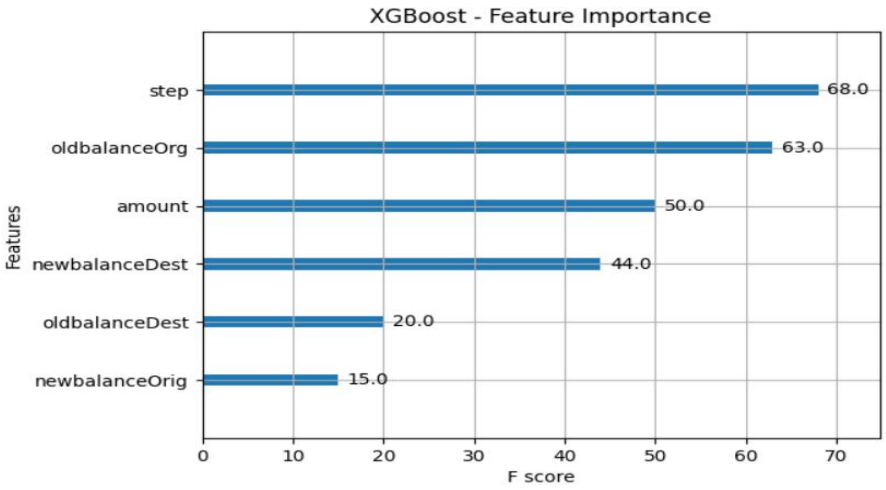
```
model = xgb.XGBClassifier(random_state=42)
model.fit(X_train, Y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
conf_matrix = confusion_matrix(Y_test, y_pred)
classification_rep = classification_report(Y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.9991666666666666
Confusion Matrix:
 [[2398    0]
 [   2    0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2398
           1       0.00      0.00      0.00         2

    accuracy                           1.00      2400
   macro avg       0.50      0.50      0.50      2400
weighted avg       1.00      1.00      1.00      2400
```

```
model = xgb.XGBClassifier(random_state=42)
model.fit(X_train, Y_train)
booster = model.get_booster()
plt.figure(figsize=(20, 10))
xgb.plot_tree(booster, num_trees=0, rankdir='LR')
plt.show()
```



```
plt.figure(figsize=(10, 6))
plot_importance(model, max_num_features=10)
plt.title('XGBoost - Feature Importance')
plt.show()
```




```
#Autoencoder
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
input_dim = X_train.shape[1]
encoding_dim = 10
input_layer = Input(shape=(input_dim,))
encoder_layer = Dense(encoding_dim, activation="relu")(input_layer)
decoder_layer = Dense(input_dim, activation="sigmoid")(encoder_layer)

autoencoder = Model(inputs=input_layer, outputs=decoder_layer)
autoencoder.compile(optimizer="adam", loss="mean_squared_error")
autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True, validation_data=(X_test,
X_test))

decoded_data = autoencoder.predict(X_test)

mse = np.mean(np.power(X_test - decoded_data, 2), axis=1)

threshold = np.percentile(mse, 95)
```

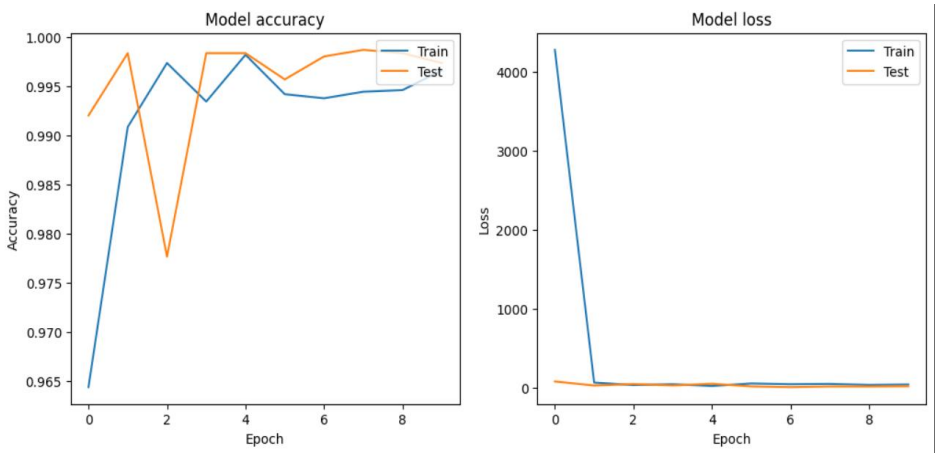
```
y_pred = np.where(mse > threshold, 1, 0)


accuracy = np.mean(y_test == y_pred)
conf_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
classification_rep = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.5
Confusion Matrix:
 Predicted   0  1
Actual
0          10  1
1           9  0
Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.91      0.67        11
           1       0.00      0.00      0.00         9

    accuracy                           0.50        20
   macro avg       0.26      0.45      0.33        20
weighted avg       0.29      0.50      0.37        20
```

```
#Artificial Neural Network (ANN)
model = Sequential()
model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
y_pred_prob = model.predict(X_test)
y_pred = np.round(y_pred_prob)
accuracy = np.mean(y_test == y_pred.flatten())
conf_matrix = pd.crosstab(y_test, y_pred.flatten(), rownames=['Actual'], colnames=['Predicted'])
classification_rep = classification_report(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
Accuracy: 0.999
Confusion Matrix:
 Predicted   0.0
Actual
0          2997
1             3
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2997
           1       0.00      0.00      0.00         3

    accuracy                           1.00      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       1.00      1.00      1.00      3000
```

```
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, Y_train, epochs=10, batch_size=32, validation_data=(X_test, Y_test))
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper right')
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```




#Model Comparison

Comapraring all the models and finding which model gives the best accuracy

```
X = random_data.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = random_data['isFraud']
def create_neural_network():
    model = Sequential()
    model.add(Dense(units=64, activation='relu', input_dim=X.shape[1]))
    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
models = {
    'LogisticRegression': LogisticRegression(random_state=42),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'MLP': MLPClassifier(random_state=42),
    'XGBClassifier': xgb.XGBClassifier(random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=3),
    'SVM': SVC(random_state=42),
    'OneClassSVM': OneClassSVM(),
    'Autoencoder': StandardScaler(),
    'Isolation Forest': IsolationForest(random_state=42),
```

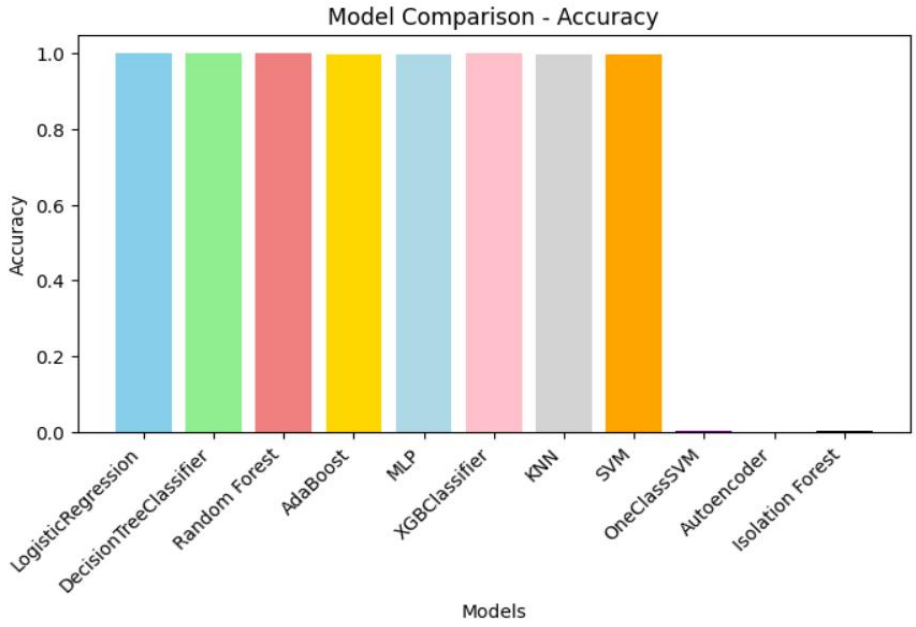
```
    'ANN': sequential()
}
scorer = make_scorer(accuracy_score)
results = {}
for model_name, model in models.items():
    try:
        scores = cross_val_score(model, X, y, cv=5, scoring=scorer, n_jobs=-1)
        mean_score = scores.mean()
        results[model_name] = mean_score
    except Exception as e:
        continue
best_model = max(results, key=results.get)
best_accuracy = results[best_model]
print("Model Comparison:")
for model_name, result in results.items():
    print(f"{model_name}: {result}")
print(f"\nBest Model: {best_model} with Accuracy: {best_accuracy}")
```

```
Model Comparison:
LogisticRegression: 0.9998666666666667
DecisionTreeClassifier: 0.9988666666666667
Random Forest: 0.9994666666666667
AdaBoost: 0.9996
MLP: 0.9932000000000001
XGBClassifier: 0.9994666666666667
KNN: 0.9994666666666667
SVM: 0.9994666666666667
OneClassSVM: 0.0002
Autoencoder: nan
Isolation Forest: 0.0002

Best Model: LogisticRegression with Accuracy: 0.9998666666666667
```

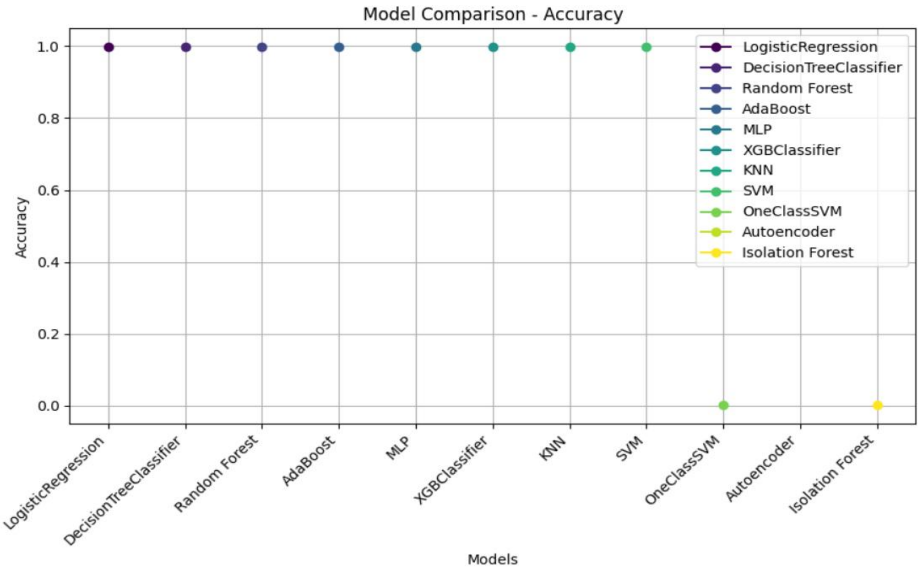
Comparing the accuracy of different models using a bar chart. Each model is represented by a colored bar, and the chart provides a quick overview of their relative performance. The varying colors help distinguish between models, offering a concise summary of their accuracy in a visually appealing manner.

```
plt.figure(figsize=(8, 4))
colors = ['skyblue', 'lightgreen', 'lightcoral', 'gold', 'lightblue', 'pink', 'lightgray', 'orange', 'purple', 'brown', 'black', 'cyan']
plt.bar(results.keys(), results.values(), color=colors)
plt.xlabel('Models')
plt.ylabel('Accuracy')
```

```
plt.title('Model Comparison - Accuracy')
plt.xticks(rotation=45, ha='right')
plt.show()
```

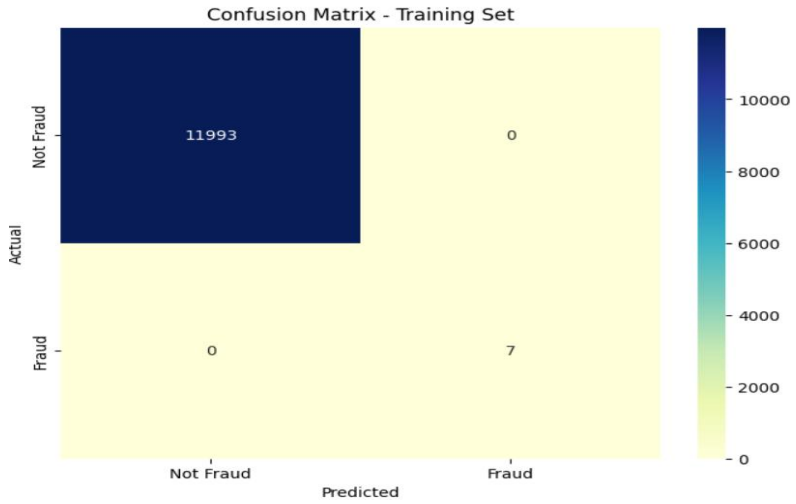



```
plt.figure(figsize=(9, 6))
colors = plt.cm.viridis(np.linspace(0, 1, len(results)))
for i, (model_name, result) in enumerate(results.items()):
    plt.plot(i, result, marker='o', color=colors[i], label=model_name)
plt.title('Model Comparison - Accuracy')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.xticks(np.arange(len(results)), list(results.keys()), rotation=45, ha='right')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




confusion matrix visualizes the model's performance on the training set, showing counts of true positive, true negative, false positive, and false negative predictions

```
classifier = RandomForestClassifier(random_state=42)

classifier.fit(X_train, Y_train)

Y_pred_train = classifier.predict(X_train)

cm_train = confusion_matrix(Y_train, Y_pred_train)

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

sns.heatmap(cm_train, annot=True, fmt='d', cmap='YlGnBu',
        xticklabels=['Not Fraud', 'Fraud'], yticklabels=['Not Fraud', 'Fraud'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Training Set')

plt.show()
```

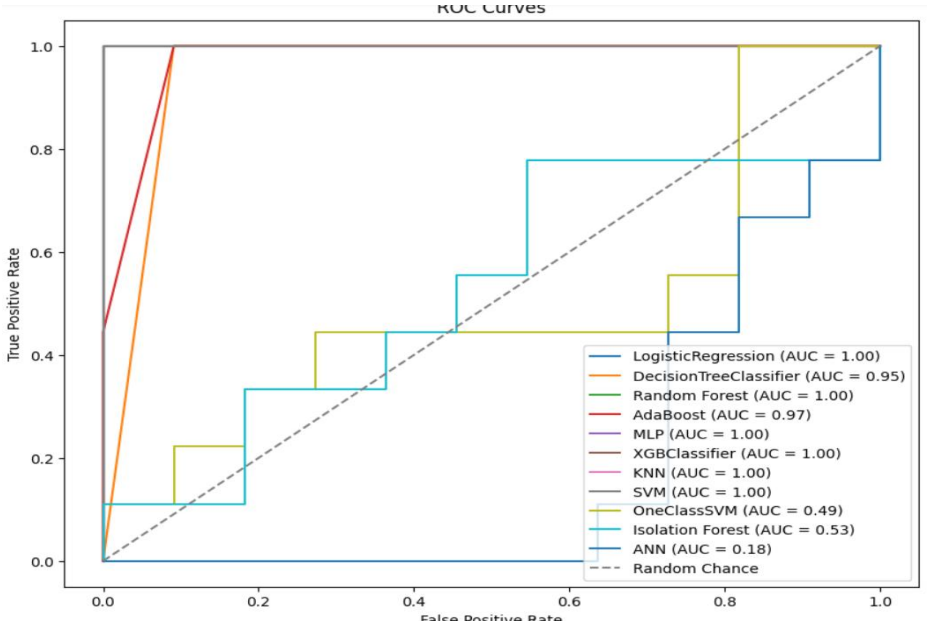



ROC Curve

This code compares Receiver Operating Characteristic (ROC) curves for various classifiers, assessing their binary classification performance.The plot helps identify classifiers with higher AUC values, indicating better trade-offs between true positive and false positive rates.

```
classifiers = {
    'LogisticRegression': LogisticRegression(random_state=42),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'MLP': MLPClassifier(random_state=42),
    'XGBClassifier': xgb.XGBClassifier(random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=3),
    'SVM': SVC(random_state=42, probability=True),
    'OneClassSVM': OneClassSVM(),
    'Isolation Forest': IsolationForest(random_state=42),
    'ANN': Sequential()
}

plt.figure(figsize=(10, 8))
for name, classifier in classifiers.items():
    if name == 'Autoencoder':
```

```
            continue

        if name == 'ANN':

            classifier.add(Dense(1, activation='sigmoid'))
            classifier.compile(optimizer='adam', loss='binary_crossentropy')
            y_pred_proba = classifier.predict(X_test)
        elif isinstance(classifier, (OneClassSVM, IsolationForest)):

            classifier.fit(X_train)
            y_pred_proba = -classifier.decision_function(X_test)
        else:

            classifier.fit(X_train, y_train)
            if hasattr(classifier, 'predict_proba'):
                y_pred_proba = classifier.predict_proba(X_test)[:, 1]
            else:
                y_pred_proba = classifier.decision_function(X_test)



        fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
        roc_auc = auc(fpr, tpr)

        plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random Chance')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curves    ')
    plt.legend()
    plt.show()
```

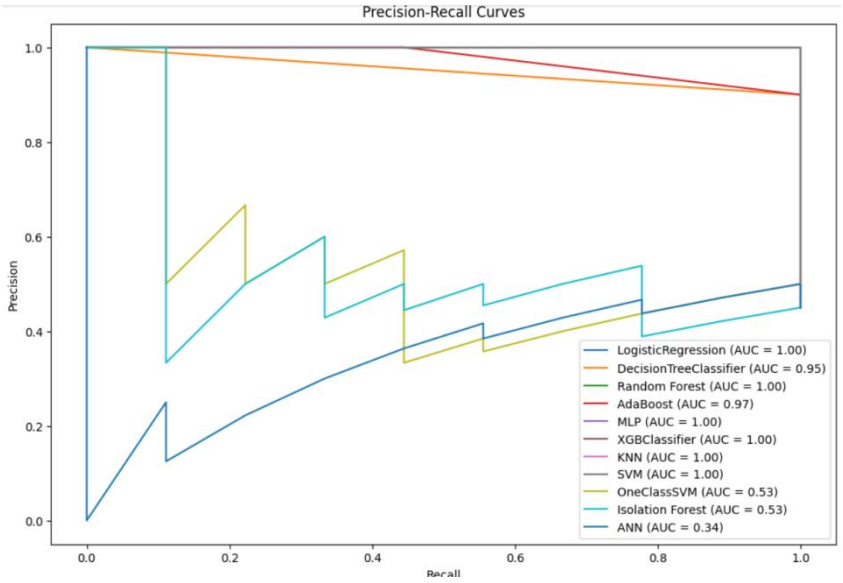



Precision Recall Curve

This code compares the Precision-Recall curves for various classifiers, evaluating their performance on a binary classification task.The plot helps identify classifiers with better trade-offs between precision and recall, crucial for tasks with imbalanced classes.

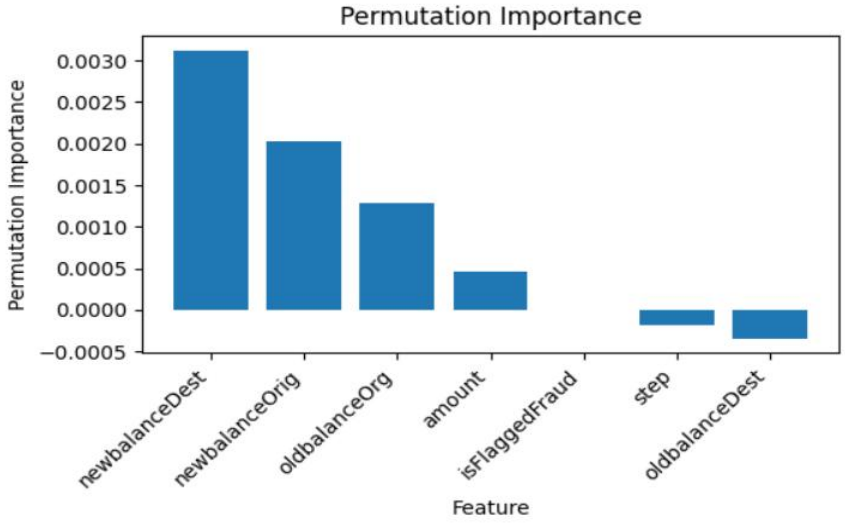
```
classifiers = {
    'LogisticRegression': LogisticRegression(random_state=42),
    'DecisionTreeClassifier': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'AdaBoost': AdaBoostClassifier(random_state=42),
    'MLP': MLPClassifier(random_state=42),
    'XGBClassifier': xgb.XGBClassifier(random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=3),
    'SVM': SVC(random_state=42, probability=True),
    'OneClassSVM': OneClassSVM(),
    'Isolation Forest': IsolationForest(random_state=42),
    'ANN': Sequential()
}
plt.figure(figsize=(12, 8))
for name, classifier in classifiers.items():
    if name == 'Autoencoder':
```

```
        continue
    if name == 'ANN':
        classifier.add(Dense(1, activation='sigmoid'))
        classifier.compile(optimizer='adam', loss='binary_crossentropy')
        y_pred_proba = classifier.predict(X_test)
    elif isinstance(classifier, (OneClassSVM, IsolationForest)):
        classifier.fit(X_train)
        y_pred_proba = -classifier.decision_function(X_test)
    else:
        classifier.fit(X_train, y_train)
        if hasattr(classifier, 'predict_proba'):
            y_pred_proba = classifier.predict_proba(X_test)[:, 1]
        else:
            y_pred_proba = classifier.decision_function(X_test)
    precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
    pr_auc = auc(recall, precision)
    plt.plot(recall, precision, label=f'{name} (AUC = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curves')
plt.legend()
plt.show()
```



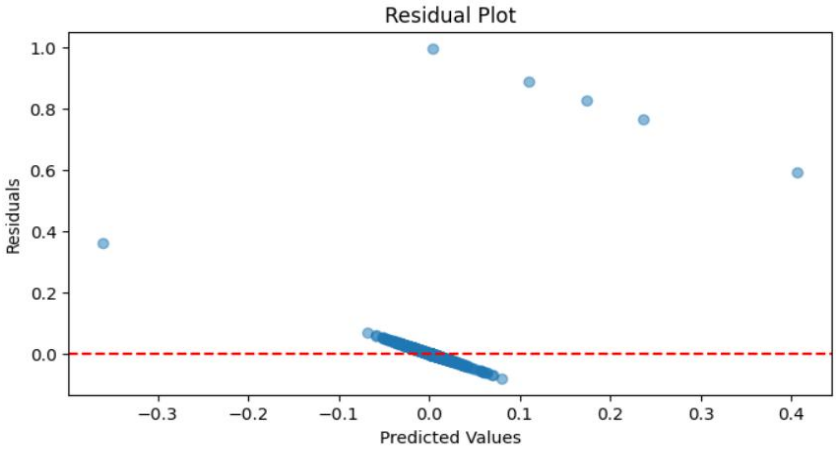
This code calculates and visualizes the permutation importance of features in a machine learning model. Permutation importance measures the impact of shuffling each feature's values on the model's performance

```
perm_importance = permutation_importance(model, X_test, y_test, n_repeats=30, random_state=42)

indices = np.argsort(perm_importance.importances_mean)[::-1]

plt.figure(figsize=(6, 3))

plt.bar(range(X_test.shape[1]), perm_importance.importances_mean[indices])

plt.xticks(range(X_test.shape[1]), X_test.columns[indices], rotation=45, ha='right')

plt.xlabel('Feature')

plt.ylabel('Permutation Importance')

plt.title('Permutation Importance')

plt.show()
```




```
#Residual Plot

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

residuals = y_test - y_pred

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

```
plt.scatter(y_pred, residuals, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residual Plot')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```



## 8.2 PLAGIARISM REPORT

# CHAPTER 9

## REFERENCES

*[1] R. Bajaj et al., "WALTS: Walmart AutoML Libraries, Tools and Services," 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Gran Canaria, Spain, 2022, pp. 21-28, doi: 10.1109/SEAA56994.2022.00013.*

*[2] M. V. Ramasami, R. Thangaraj, S. Manoj Kumar and S. Eswaran, "Exploratory Data Analysis of Walmart Outlets Sales using Data Analytics Techniques," 2023 International Conference on Digital Applications, Transformation & Economy (ICDATE), Miri, Sarawak, Malaysia, 2023, pp. 1-4, doi: 10.1109/ICDATE58146.2023.10248586.*

*[3] S. B. Latha, C. Dastagiraiah, A. Kiran, S. Asif, D. Elangovan and P. C. S. Reddy, "An Adaptive Machine Learning model for Walmart sales prediction," 2023 International Conference on Circuit Power and Computing Technologies (ICCPCT), Kollam, India, 2023, pp. 988-992, doi: 10.1109/ICCPCT58313.2023.10245029.*

*[4] Z. Qiao, "Walmart Sale Forecasting Model Based On LightGBM," 2020 2nd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI), Taiyuan, China, 2020, pp. 76-79, doi: 10.1109/MLBDBI51377.2020.00020*

*[5] V. Jain, H. Kavitha and S. Mohana Kumar, "Walmart Credit Card Fraud Analytics," 2022 IEEE International Conference on Data Science and Information System (ICDSIS), Hassan, India, 2022, pp. 1-5, doi: 10.1109/ICDSIS55133.2022.9915901.*

*[6] C. G. Tekkali and K. Natarajan, "synthetic Financial Datasets For Fraud Detection," 2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS), Coimbatore, India, 2023, pp. 523-526, doi: 10.1109/ICAIS56108.2023.10073712*

*[7] Luz, S., Haider, F., de la Fuente, S., Fromm, D., & MacWhinney, B. (2020). Alzheimer's dementia recognition through spontaneous speech: The ADReSS challenge. arXiv preprint arXiv:2004.06833.*

*[8] A. Mahajan, V. S. Baghel and R. Jayaraman, "Walmart Credit Card Fraud Detection," 2023 10th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2023, pp. 339-342.*

*[9] T. Zhang and S. Gao, "Fraud Detection," 2022 4th International Conference on Intelligent Information Processing (IIP), Guangzhou, China, 2022, pp. 272-275, doi: 10.1109/IIP57348.2022.00063.*

*[10] Z. Chen, M. Cai and Z. Wang, "Online Payments Fraud Detection," 2022 IEEE 4th International Conference on Power, Intelligent Computing and Systems (ICPICS), Shenyang, China, 2022, pp. 36-40, doi: 10.1109/ICPICS55264.2022.9873622.*

*[11] A. Namiranian and M. R. Hashemi, "A. Namiranian and M. R. Hashemi, "A new DCT based scalable distributed fraud detection architecture," 6th International Symposium on Telecommunications (IST), Tehran, Iran, 2012, pp. 1076-1081, doi: 10.1109/ISTEL.2012.6483146.," 6th International Symposium on Telecommunications (IST), Tehran, Iran, 2012, pp. 1076-1081, doi: 10.1109/ISTEL.2012.6483146*

**Web References:**

**1.Synthetic Financial Datasets For Fraud Detection:**

https://www.kaggle.com/datasets/ealaxi/paysim1

**2.Credit Card Fraud Detection:**

https://www.kaggle.com/code/moazeldsokyx/credit-card-fraud-detection

**3.Performance evaluation in Machine Learning :**
https://ieeexplore.ieee.org/document/8251305

**4.Preprocessing techniques for fraud data:** https://towardsdatascience.com/ieee-cis-fraud-detection-by-lightgbm-b8956a8e4b53

**5.Machine Learning for Fraud Detection:**
https://www.itransition.com/machine-learning/fraud-detection

# CHAPTER 10

# WORKLOG

**International Centre for Education and Research (ICER), VIT-Bangalore**

**PROJECT**

**STUDENT'S WORKLOG SHEET**

**Student Name:** S.P.DHINESH.
**Register Number:** 23MSP3032.
**Project Title:** WALMART TRANSACTION FRAUD DETECTION SYSTEM.
**Domain:** MARKETING & BUISNESS.
**Tools Learned / used:**

| Day | Date | Task | Student's Remarks | Signature of Student with Date | Signature of the Project Mentor |
|---|---|---|---|---|---|
| Day 1 | 15/11/23 | Domain Selection and Base Paper Selection | Selected domain and Base Paper. | 15/11/2023 | 24/11/23 |
| Day 2 | 16/11/23 | Zeroth review | zeroth review completed. | 16/11/2023 | 24/11/23 |
| Day 3 | 17/11/23 | Topic Finalization | topic finalized. | 17/11/2023 | 24/11/23 |
| Day 4 | 18/11/23 | Rough Draft Abstract (Objective, Dataset, methodology, Tools, Expected Outcome) | A rough draft of the abstract was composed including the objectives, methodology and ML tools to be used with the outcome. | 19/11/2023 | 24/11/2023 |
| Day 5 | 20/11/23 | Final Abstract (Objective, Dataset, methodology, Tools, Expected Outcome) | The finalized draft of abstract has been prepared with the purpose, tools methodology, and expected results and impact of the study. | 20/11/2023 | 24/11/2023 |

| | | | | | |
|---|---|---|---|---|---|
| Day 6 | 21/11/23 | Draft Literature Review(10 Papers) and Tabulate the understanding | 10 research papers have been studied and the literature review of those papers are completed | 21/11/2023 | 24/11/2023 |
| Day 7 | 22/11/23 | Final Literature Review(10 Papers) Tabulate the understanding | The 10 research papers that were studied have been converted into a literature review in tabular format instructed. | 22/11/2023 | 24/11/2023 |
| Day 8 | 23/11/23 | Approach to the Problem write up 2 page (Aim, Research Objective, Research questions, Algorithm, Tools, Dataset, Proposed architecture, Expected outcome, References) | Approach to the problem has been written up including aim, objective, questions, tools, algorithm, architecture outcome, expected & reference | 23/11/2023 | 7/12 |
| Day 9 | 24/11/23 | Exploratory Data Analysis(EDA) | ~~Exploratory~~ Data Analysis completed for the given dataset for the study. | 24/11/2023 | 24/11 |
| Day 10 | 25/11/23 | EDA and Documentation | EDA and Documentation for the same is completed. | 24/11/2023 | 24/11 |

| | | | | | |
|---|---|---|---|---|---|
| Day 11 | 27/11/23 | First Review | completed first review | 27/11/2023 | 1/12 |
| Day 12 | 28/11/23 | Implementation | Started exploring SPAM MAX model & logistic regression method. | 28/11/2023 | 1/12 |
| Day 13 | 29/11/23 | Implementation | completed EDA and logistic regression model and looking for more accurate model. | 29/11/2023 | 1/12 |
| Day 14 | 30/11/23 | Implementation | Proposed model is built | 30/11/2023 | 1/12 |
| Day 15 | 01/12/23 | Implementation | Proposed model is built | 01/12/2023 | 1/12 |

| | | | | | |
|---|---|---|---|---|---|
| Day 16 | 02/12/23 | Implementation | Proposed model was built | 02/12/2023 | 2/12 |
| Day 17 | 04/12/23 | Second Review | Second Review ppt done. | 04/12/2023 | 5/12 |
| Day 18 | 05/12/23 | Second Review | Second Review Done. | 05/12/2023 | 6/12 |
| Day 19 | 06/12/23 | Results and Discussion | Results and Discussion are Done. | 06/12/2023 | 7/12 |
| Day 20 | 07/12/23 | Conclusion and future scope | conclusion and future scope are Done. | 07/12/2023 | 7/12 |

| | | | | | |
|---|---|---|---|---|---|
| Day 21 | 08/12/23 | Rough Draft of Report submission | Rough Draft of Report submission done. | 08/12/2023 | 8/12 |
| Day 22 | 09/12/23 | PPT for Final Review Submission | Submitted PPT for final review. | 09/12/2023 | 9/12 |
| Day 23 | 11/12/23 | PPT for Final Review Submission | PPT for final review submission done. | 11/12/2023 | 11/12 |
| Day 24 | 12/12/23 | Final Review | final review done | 12/12/2023 | |
| Day 25 | 13/12/23 | Final Review | final review done. | 13/12/2023 | 13/12 |