Detecting Fraudulent Transactions

Mid-Term Project Report

CSCI: 5502 Data Mining

By

Tapas Das, Siddhant Sharma, Yan Xia

Under the supervision of:

Alfonso G. Bastias, Ph.D
University of Colorado Boulder

University of Colorado **Boulder**

University of Colorado, Boulder,
Boulder, U.S.A

March 2024

University of Colorado, Boulder

Abstract

Master's of Science

Detecting Fraudulent Transactions

by Tapas Das, Siddhant Sharma, Yan Xia

Fraudulent transactions pose significant threats to financial institutions and businesses, leading to substantial financial losses and reputational damage. Consequently, there is a pressing need for robust fraud detection systems capable of identifying fraudulent activities in real-time. Machine learning (ML) techniques have emerged as a promising solution due to their ability to analyze large volumes of transactional data and detect patterns indicative of fraudulent behavior. This abstract outlines the key components and methodologies employed in building an effective fraud detection system using ML techniques.

The proposed system leverages a variety of ML algorithms, including but not limited to supervised learning methods such as logistic regression, decision trees, random forests, and support vector machines, as well as unsupervised learning techniques like clustering and anomaly detection. Feature engineering plays a crucial role in extracting relevant information from transactional data, including transaction amount, frequency, location, time, and various other contextual attributes

Furthermore, we will try to incorporate ensemble methods to enhance predictive performance. In conclusion, the proposed fraud detection system demonstrates the efficacy of leveraging machine learning techniques to combat fraudulent activities in financial transactions. By harnessing the power of advanced algorithms and comprehensive feature engineering, coupled with real-time monitoring and adaptive learning capabilities, the system provides a robust defense against fraud while minimizing false positives and ensuring operational efficiency for financial institutions and businesses.

Acknowledgments

We would like to express our sincere gratitude to Professor Alfonso G. Bastias, Ph.D., for his invaluable guidance, mentorship, and support throughout this project. His expertise in the field of data science and insightful feedback significantly contributed to the development and success of the Credit Card Fraud Detection project.

We are also thankful to the research community for their open sharing of knowledge and datasets, which facilitated our exploration and experimentation. Additionally, we extend our appreciation to our peers and colleagues for their encouragement and collaboration.

Thank you to everyone who contributed to this endeavor in any capacity. Your support has been instrumental in the accomplishment of our goals.

Contents

Chapter 1

Introduction

With more and more people making purchases online, credit card theft is a major issue for companies and consumers in the modern day. Fraudulent activities, such as unlawful transactions and identity theft, present substantial financial risks and can damage the reputation of financial institutions. It is crucial to detect and prevent credit card fraud to uphold trust in the financial system and protect consumers.

Credit card fraud affects both financial institutions and customers significantly. Fraud victims may suffer cash losses, credit score harm, and mental misery. Furthermore, fraud may have widespread consequences, impacting not just individual victims but businesses, who may suffer financial losses from chargebacks and fraudulent transactions. Thus, it is crucial to have efficient fraud detection systems in place to reduce these risks and safeguard all participants in the financial environment.
Credit card fraud criminals persist in developing intricate strategies to attack system flaws, despite technological breakthroughs and enhanced security measures. Conventional rule-based systems and manual assessments are unable to address the changing nature of fraud schemes.

Consequently, there is an increasing focus on utilizing data-driven methods like machine learning and artificial intelligence to improve fraud detection capacities.

Financial institutions may now evaluate large volumes of transactional data in real time due to the widespread use of big data and advanced analytics technologies. Machine learning algorithms can differentiate between genuine and fraudulent transactions with high accuracy by recognizing patterns, anomalies, and behavioral trends.

Furthermore, these algorithms have the capability to adjust and acquire knowledge from fresh data, which enhances their ability to identify developing fraud trends.

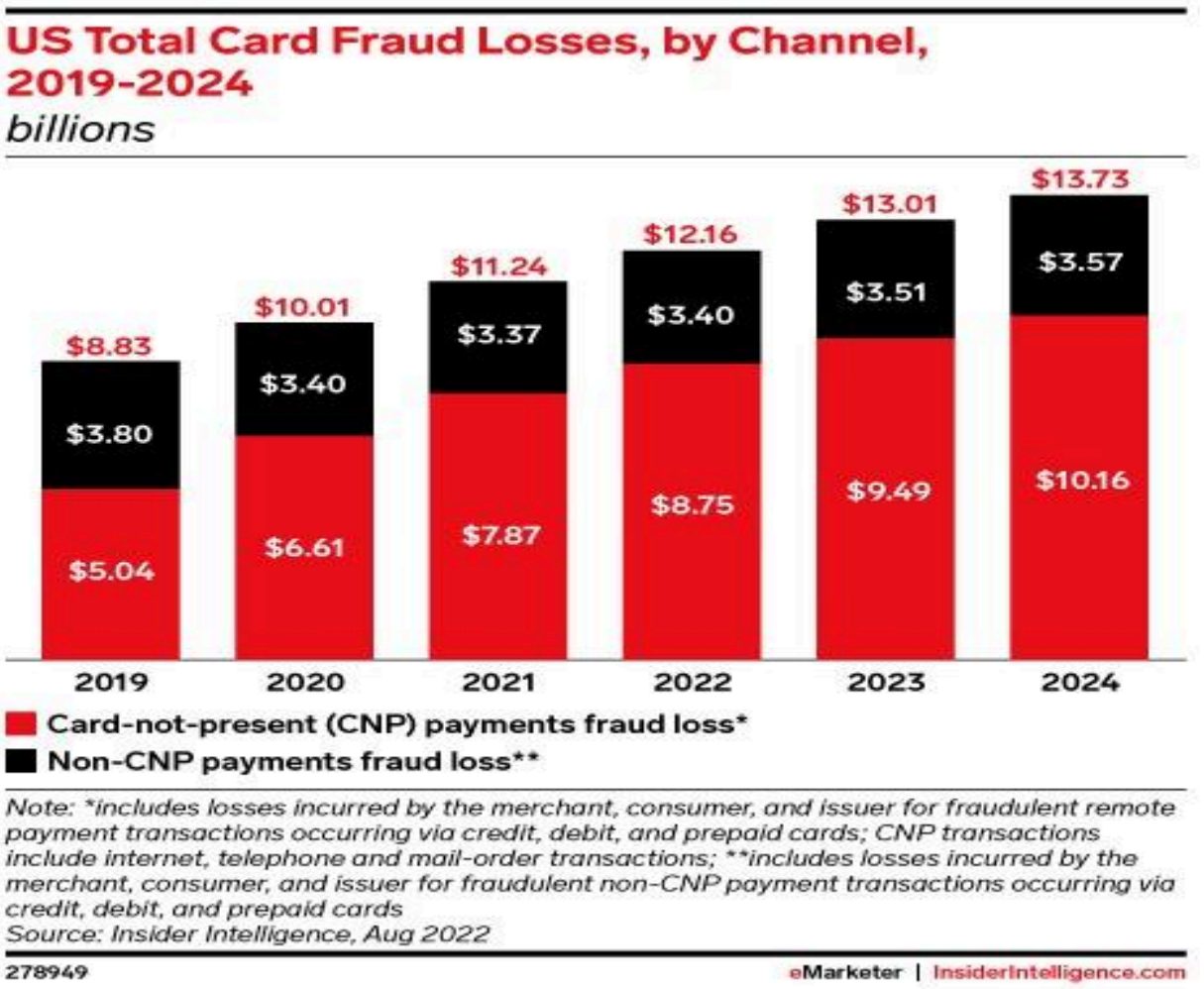


Fig 1.1 Total Credit Card Fraud Losses 2019-2024

Chapter 2

Data Exploration

2.1 Dataset Introduction

The dataset comprises credit card transactions conducted by European cardholders in September 2013. This dataset comprises transactions that took place during two days, with a total of 492 instances of fraud out of a total of 284,807 transactions.

The dataset comprises numerical input variables that have undergone a Principal Component Analysis (PCA) transformation. Some of the original characteristics are distance_from_home, distance_from_last_transaction, ratio_to_median_purchase_price, repeat_retailer, used_chip, used_pin_number, and online_order, which are transformed into variables using PCA. The principle components derived with PCA are denoted as V1, V2,... V28. The only characteristics that have not undergone PCA transformation are 'Time' and 'Amount'. Characteristic In the dataset, the variable 'Time' represents the duration in seconds between each transaction and the initial transaction. The 'Amount' feature represents the transaction amount and can be utilized for example-dependent cost-sensitive learning. Characteristic The response variable, denoted as 'Class', assumes a value of 1 when fraud is present and 0 when it is not.
The dataset exhibits a significant imbalance, with the positive class (defined as frauds) representing a mere 0.172% of the total transactions. There are a total 284,807 records and 31 fields.

To solve this imbalanced issue we will implement the SMOTE algorithm to make the transaction biased. Dataset contains numerical input variables which are the result of a PCA transformation.

Source - https://data.world/raghu543/credit-card-fraud-data

2.2 Data Preparation and Cleaning

The dataset used for detecting fraudulent credit card transactions comprises records of transactions conducted by European cardholders in September 2013. It consists of two days' worth of transactions, totaling 284,807 instances, with 492 instances identified as fraudulent.

Data Cleaning:

The initial step involves ensuring data cleanliness by handling missing values appropriately. Null values in the dataset are addressed by replacing them with the mean value of their respective columns. This ensures that the dataset remains intact for subsequent analysis and modeling. The following Python code demonstrates this data cleaning process:

```python
import numpy as np
import pandas as pd

# Read the dataset
data_path =  r'./drive/My Drive/creditcard.csv'
df = pd.read_csv(data_path)

# Replace null values with mean of respective columns
for column in df.columns:
    if df[column].isna().sum() > 0:
        mean_value = df[column].mean()
        df[column].fillna(mean_value, inplace=True)

# Verify absence of null values
print(df.isna().sum())
```

The output confirms the successful removal of null values from the dataset.

Handling Imbalanced Data:

Addressing the significant class imbalance between fraudulent and non-fraudulent transactions is crucial for developing an effective fraud detection model. Synthetic Minority Over-sampling Technique (SMOTE) is employed to mitigate this issue by oversampling the minority class (fraudulent transactions) to achieve a more balanced dataset.

Further Data Exploration:

Additionally, exploratory data analysis (EDA) techniques such as visualization and statistical analysis can be applied to gain insights into the distribution of features, identify potential outliers, and understand the relationship between variables. In this particular dataset, since all features are numerical after PCA transformation, there is no need for encoding categorical variables. However, outlier detection techniques could be applied if necessary to ensure the robustness of the model.

By meticulously preparing the dataset, including handling missing values, addressing class imbalance, and exploring the data, we create a solid foundation for building a machine-learning model capable of accurately detecting fraudulent credit card transactions.

2.3 Treating Data Imbalance

Addressing Class Imbalance with SMOTE

Detecting fraudulent transactions in credit card data is a critical task for financial institutions. However, the challenge lies in the severe class imbalance where the instances of fraud are significantly outnumbered by legitimate transactions. Traditional machine learning algorithms trained on imbalanced data tend to be biased towards the majority class, leading to poor performance in identifying fraudulent activities. To mitigate this issue, techniques such as Synthetic Minority Over-sampling Technique (SMOTE) are employed to rebalance the dataset and improve the performance of fraud detection models.

Understanding Class Imbalance:

In the context of credit card fraud detection, class imbalance refers to the situation where the number of legitimate transactions (negative class) far outweighs the number of fraudulent transactions (positive class). This imbalance can be substantial, with fraudulent transactions accounting for only a tiny fraction of the total dataset, typically less than 1%. Imbalanced data poses a significant challenge for machine learning algorithms as they tend to focus on optimizing accuracy, which may lead to a biased model that performs poorly in detecting minority class instances.

Introduction to SMOTE:

Synthetic Minority Over-sampling Technique (SMOTE) is a popular method used to address class imbalance by generating synthetic samples of the minority class. The goal of SMOTE is to create a more balanced dataset by oversampling the minority class, thereby improving the classifier's ability to learn from rare instances and making it more robust in identifying minority class patterns.

How SMOTE Works:

SMOTE works by synthesizing new minority class instances based on the existing minority samples in the dataset. Here's a step-by-step explanation of the SMOTE algorithm:

1. Identifying Minority Class Instances: SMOTE begins by identifying the minority class instances in the dataset.
2. Selecting Nearest Neighbors: For each minority class instance, SMOTE identifies its k nearest neighbors in the feature space. The value of k is a parameter specified by the user.
3. Generating Synthetic Samples: SMOTE then generates synthetic samples by interpolating between the minority class instance and its selected nearest neighbors. This interpolation is performed by randomly selecting a point along the line segment connecting the minority instance and one of its nearest neighbors.
4. Adding Synthetic Samples: The synthetic samples are added to the original dataset, effectively increasing the representation of the minority class.

By generating synthetic samples in this manner, SMOTE helps to alleviate class imbalance and create a more balanced dataset for training machine learning models.

Advantages of SMOTE:

1. Improves Minority Class Representation: By synthesizing new instances of the minority class, SMOTE helps to address the class imbalance problem and ensures that the classifier receives sufficient training data for the minority class.
2. Preserves Information: SMOTE generates synthetic samples based on the existing minority class instances, ensuring that the synthetic samples are representative of the underlying distribution of the minority class.
3. Reduces Bias: By creating a more balanced dataset, SMOTE reduces the bias towards the majority class, allowing the classifier to learn from both classes more effectively.
4. Compatibility with Various Algorithms: SMOTE is compatible with a wide range of machine learning algorithms, making it a versatile technique for addressing class imbalance in various applications.

Challenges and Considerations:

While SMOTE is a powerful technique for addressing class imbalance, it's essential to consider potential challenges and limitations:

Impact on Model Performance: Oversampling with SMOTE can lead to overfitting, especially if the synthetic samples are not representative of the underlying distribution. Careful evaluation and validation

are necessary to ensure that the oversampling does not degrade the model's performance on unseen data.

Computational Complexity: Generating synthetic samples can be computationally intensive, particularly for large datasets with high-dimensional feature spaces. Efficient implementations and parameter tuning may be necessary to manage computational resources effectively.

Handling Noise and Outliers: SMOTE may generate synthetic samples in regions of feature space that contain noise or outliers. Preprocessing steps such as outlier removal and feature scaling can help mitigate these issues.

In conclusion, the Synthetic Minority Over-sampling Technique (SMOTE) is a valuable tool for addressing class imbalance in credit card fraud detection and other machine learning applications. By generating synthetic samples of the minority class, SMOTE helps to rebalance the dataset, improve model performance, and enhance the robustness of fraud detection systems.

2.4 Exploratory Data Analysis

1. Pie Chart of Class Distribution: Used a pie chart to visually represent the distribution of classes (fraudulent vs. non-fraudulent) before and after preprocessing.

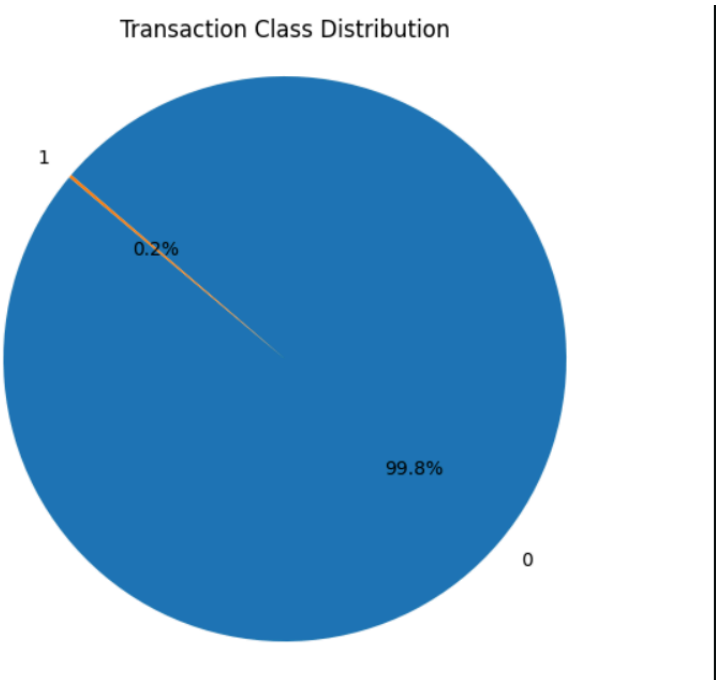


Fig 1.2 - Transaction class distribution

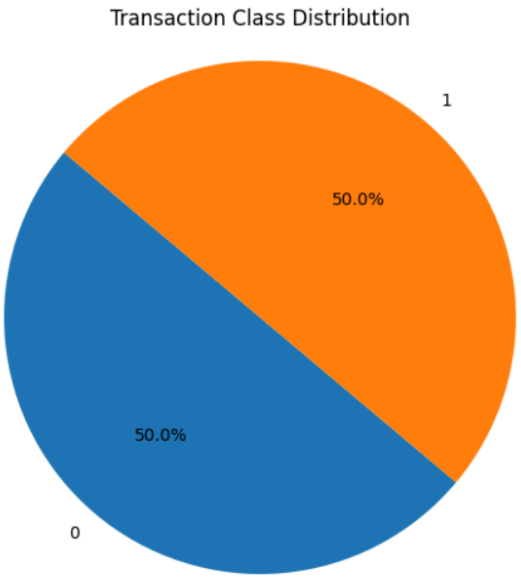


Fig 1.3 Transaction class Distribution after preprocessing

2. Transaction Amount Over Time: Utilized line plots or time series plots to visualize the trend of transaction amounts over time, distinguishing between fraudulent and non-fraudulent transactions. This will help us understand any temporal patterns or anomalies associated with transaction amounts.

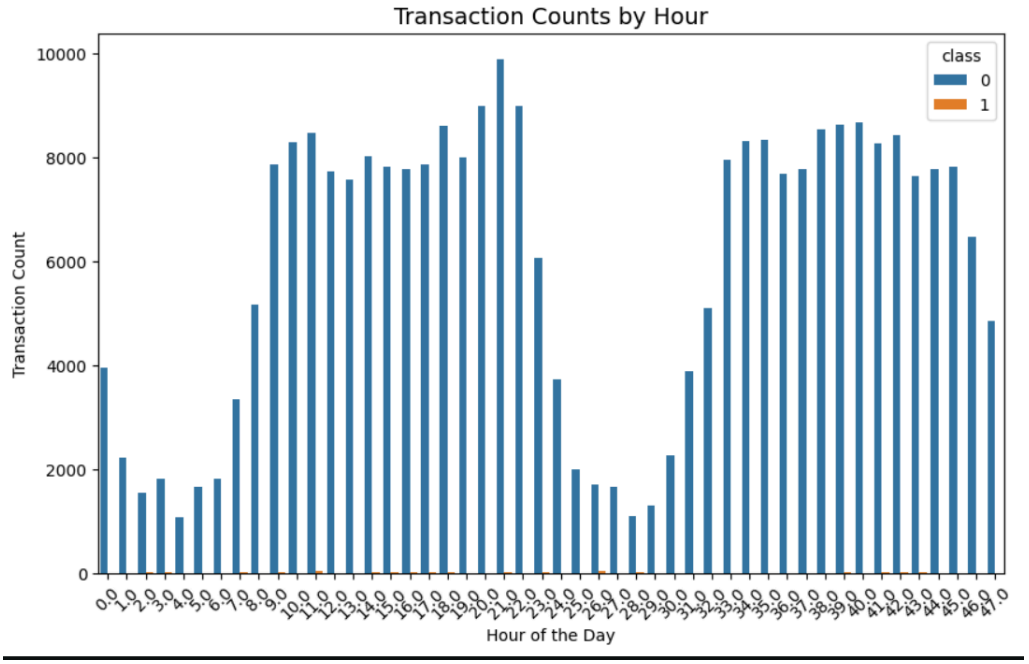


Fig 1.3 Transaction Count by an hour before preprocessing

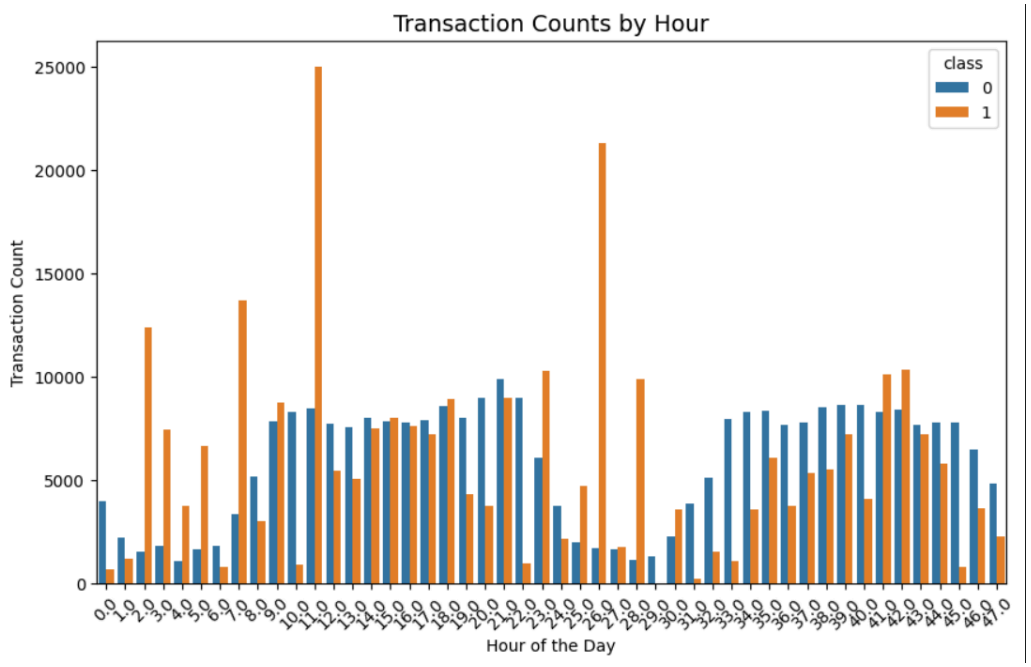


Fig 1.4 Transaction count by an hour after pre-processing.

3. Transaction Counts per Day: Used bar charts or histograms to display the distribution of transaction counts per day, categorized by class. This will allow for easy comparison and identification of any changes in the distribution after preprocessing.

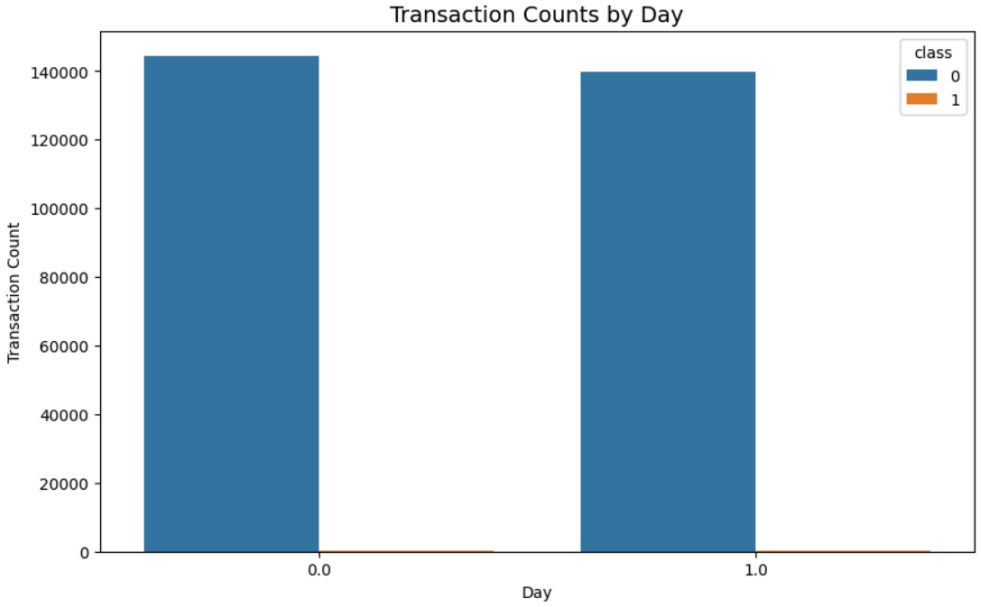


Fig 1.5 Transaction count by the day before preprocessing

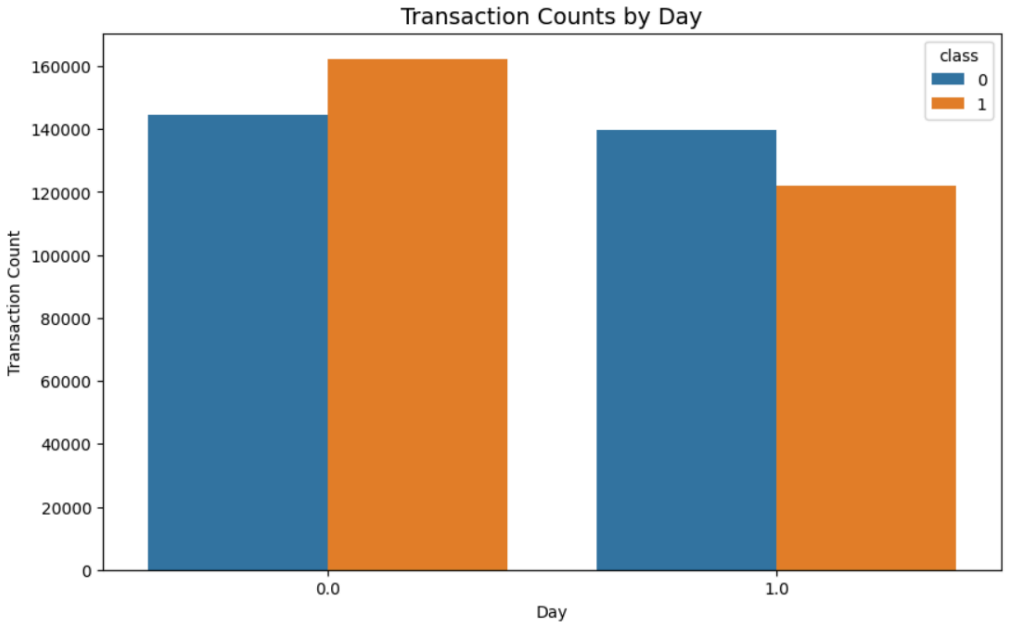


Fig 1.6 Transaction count by day after preprocessing

4. Distribution of Transactions: Used histogram with a density plot to check the distribution of fraudulent transactions. We can observe that fraudulent transactions have generally a lower value than genuine transactions.

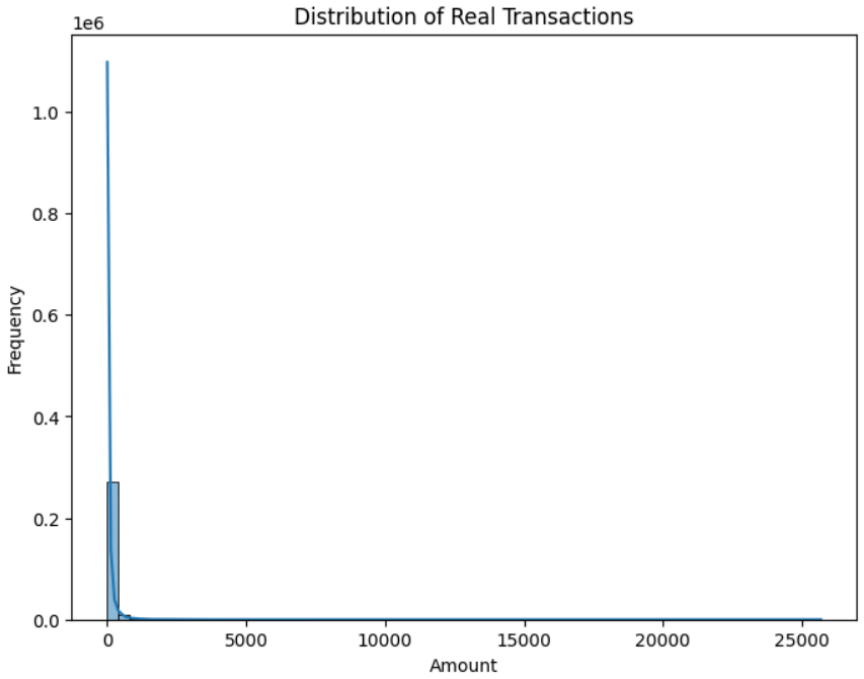


Fig 1.7 Distribution of real transactions

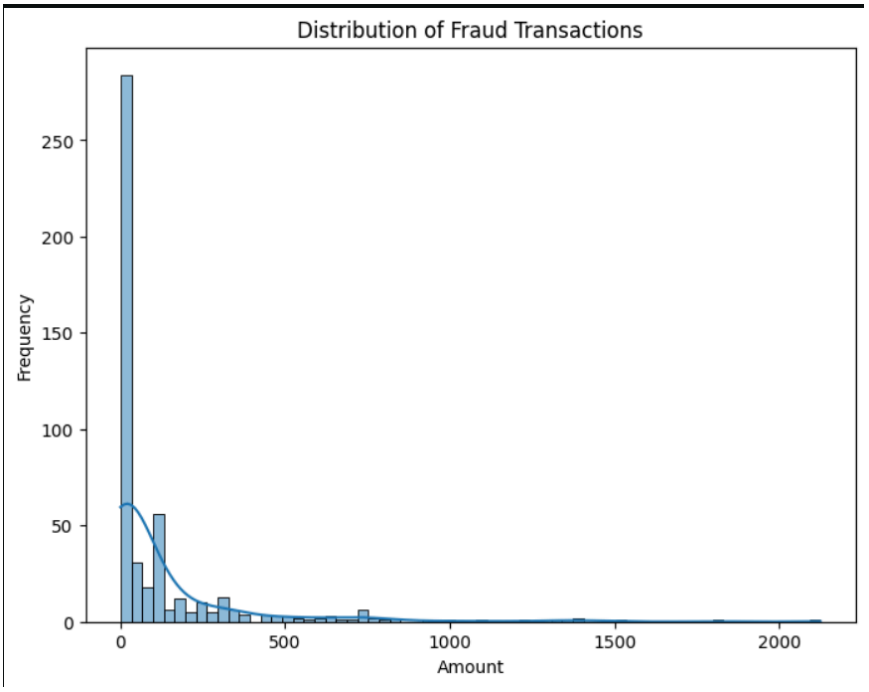


Fig 1.8 Distribution of fraudulent transactions

5. Detecting Outliers using Box Plot: We used a box plot to determine the outliers in real and fraudulent transactions present in the dataset. We observed that the values are more spread out towards the head than the tail.

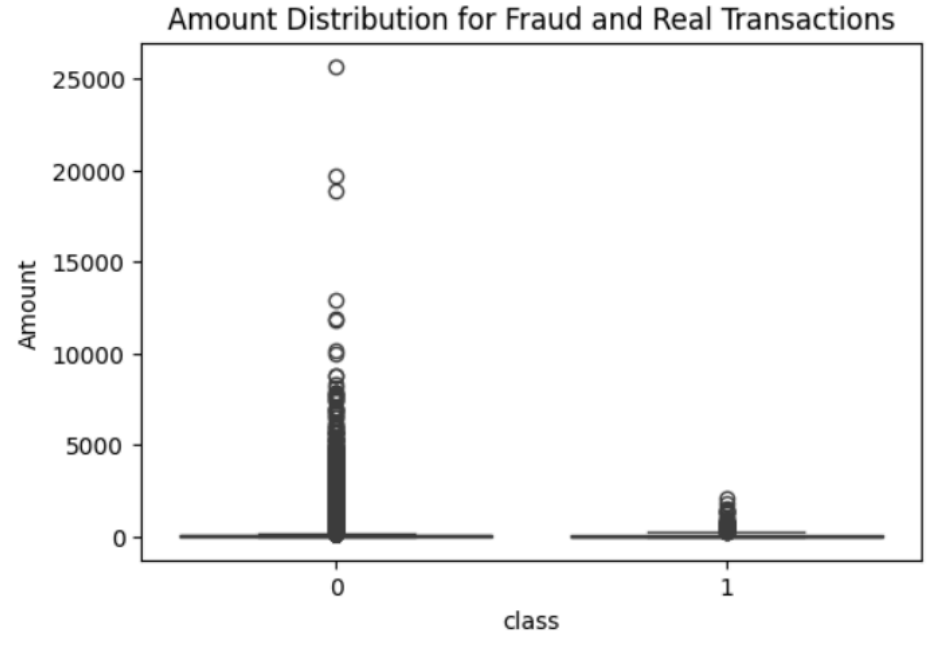


Fig 1.9: The above box plot helps in detecting the outliers in the data.

6. Feature Distribution: We can visualize the distribution of features for both fraudulent and non-fraudulent transactions before and after preprocessing. We have used a scatter plot with the features for this.

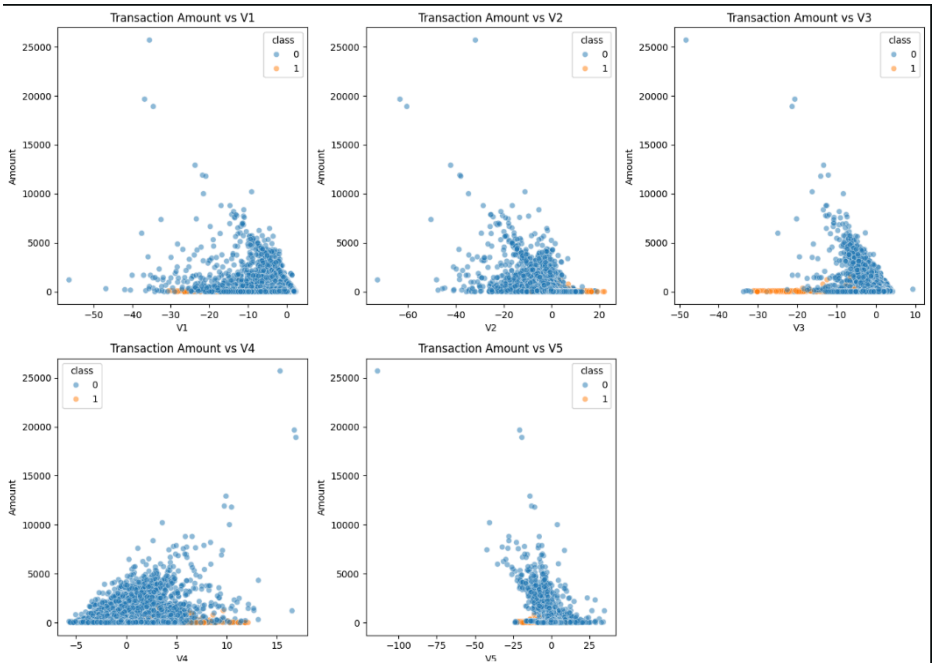


Fig 1.10: Scatter Plot for amount vs features before data pre-processing

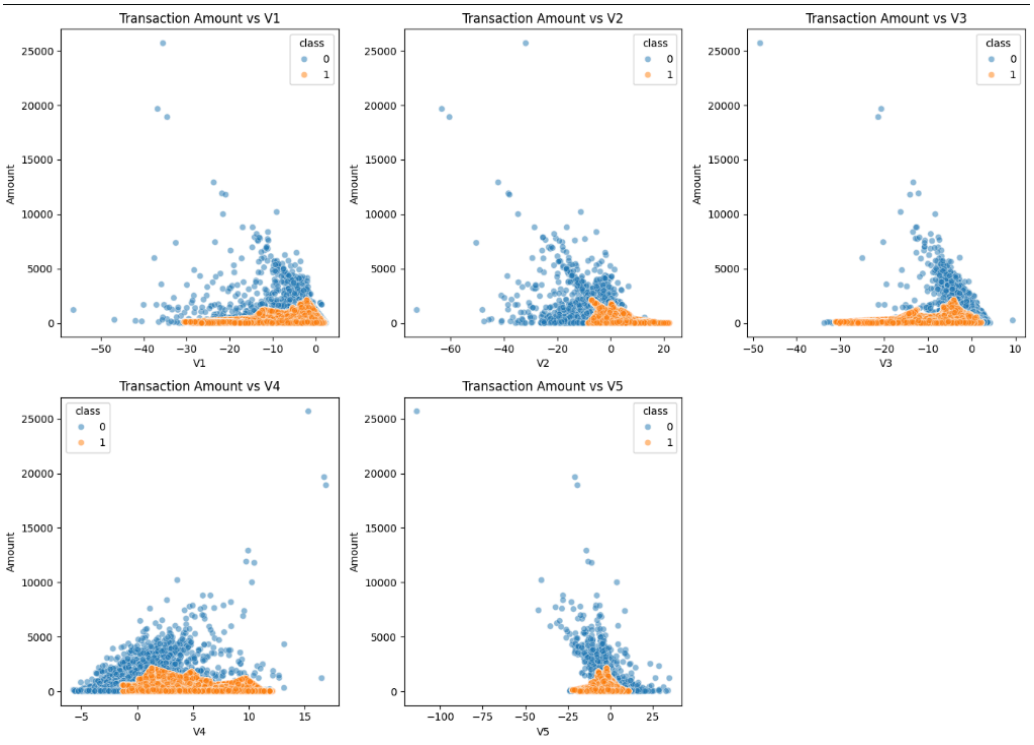


Fig 1.11: Scatter Plot for amount vs feature before data after pre-processing

2.5 Dataset Before and After Processing Comparison

```
[ ] creditcard_df
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.02105
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.01472
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.05975
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.0614
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.2151
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.8237
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.0535
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.0265
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.1045
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.0136

284807 rows × 31 columns

Fig 1.12: Dataset Before Processing

```
[ ] balanced_df
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	
0	0.000000	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.0
1	0.000000	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.0
2	1.000000	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.0
3	1.000000	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.0
4	2.000000	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
568625	144838.659385	-6.379157	1.672637	-5.885670	2.068340	-0.668576	-3.336450	-4.995823	2.632847	-2.275158	...	0.641337	-0.249308	-2.311290	-0.159402	1.190079	-0.258067	0.777265	-0.7
568626	65965.011763	-2.479028	0.958932	-1.782249	1.541783	-1.191990	-0.466794	-1.957161	0.312580	-0.433956	...	0.351983	0.208869	-0.235986	-0.404446	0.220454	0.685263	-0.890346	0.5
568627	34592.129093	-1.799894	2.368957	-2.673997	1.705968	-1.355923	-1.121788	-2.057832	-1.677459	-0.659287	...	1.473371	-0.581778	-0.013899	-0.144597	0.120315	0.242272	-0.121166	-0.5
568628	129683.002907	0.255234	2.432041	-5.388252	3.793925	-0.230814	-1.382725	-1.572929	0.748305	-1.600633	...	0.316760	-0.036858	0.182968	0.190701	-0.339250	-0.272824	0.315507	-0.0
568629	91471.277869	-4.453646	3.210469	-5.294410	1.449911	-1.264653	-0.493626	-3.130644	-4.165957	0.998760	...	4.414468	-1.065864	0.798149	0.299668	0.064660	-0.446730	-0.363233	1.0

568630 rows × 31 columns

Fig 1.13: Dataset After Processing

Chapter 3

Model Development

3.1 Model Selection

The selection of diverse models for credit card fraud detection is strategic, catering to the complexity of the problem and aiming to enhance detection accuracy.

Naive Bayes, a probabilistic classifier, is efficient for its simplicity and speed. It's particularly useful for its ability to handle high-dimensional data and relatively small training sets, making it suitable for initial exploration and benchmarking. Logistic Regression is a classic choice for binary classification problems like fraud detection. Its interpretability allows for easy understanding of the importance of each feature in determining fraudulence, aiding in risk assessment and model transparency. Decision Trees provide clear decision-making pathways, mimicking human decision-making processes. They are adept at capturing nonlinear relationships in the data, making them effective in identifying intricate patterns indicative of fraudulent activities. Random Forests, an ensemble of decision trees, improve upon the robustness of decision trees by reducing overfitting and increasing generalization performance. They excel in handling large datasets with high dimensionality, contributing to enhanced fraud detection accuracy.

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), are ideal for sequential data analysis, capturing temporal dependencies in credit card transactions. Their ability to retain information over long periods makes them adept at detecting subtle fraudulent patterns over time.
k-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm, effective for identifying local patterns in the data. Its simplicity and flexibility make it valuable for detecting anomalies or outliers, potentially indicative of fraudulent transactions. Feedforward Neural Networks (FNN), with their ability to learn complex patterns, are well-suited for credit card fraud detection tasks. They can automatically learn features from raw data, enabling them to capture intricate patterns that may not be apparent through traditional methods.

By employing this diverse set of models, the system can leverage the unique strengths of each algorithm, ultimately improving the overall accuracy and robustness of the fraud detection system.

3.2 Model Details

In this project we have implemented 7 models on imbalanced and balanced dataset achieved with the SMOTE algorithm -

1. Naive Bayes
2. Logistic Regression
3. Decision Tree
4. Random Forest
5. Long short term memory (LSTM)
6. k-nearest neighbors (KNN)
7. Feedforward neural Network(FNN)

3.2.1 Naive Bayes

Definition:
Naive Bayes is a family of probabilistic algorithms that apply Bayes' theorem with the naive assumption of conditional independence between every pair of features given the value of the class variable. This model is easy to build and particularly useful for very large datasets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Pros:

1. Efficiency: Naive Bayes classifiers require a small amount of training data to estimate the necessary parameters to get the results.
2. Speed: They are extremely fast compared to more sophisticated methods.
3. Simplicity: Naive Bayes works well with an assumption of independence among predictors.
4. Performance: Often outperforms more complex models when the independence assumption holds.
5. Good results obtained in multi-class problems.

Cons:

1. Strong feature independence assumptions: Real-world scenarios rarely have completely independent features, which can affect the performance.
2. Bad estimator: Probabilities obtained are not to be taken too seriously.
3. Limited to prediction: It predicts the classes but doesn't give underlying probabilities unless explicitly programmed to output them.

3.2.2 Logistic Regression

Definition:
Logistic Regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model; it is a form of binomial regression.

Pros:

1. Efficiency: Logistic regression is less intensive computationally, hence faster for small to medium-sized datasets.
2. Good probability estimates: Outputs have a nice probabilistic interpretation, and the algorithm can be regularized to avoid overfitting.
3. Versatility: Can be extended to multiclass classification (e.g., multinomial logistic regression).
4. Interpretability: Coefficients of the model can be used to interpret the importance of different features.

Cons:

1. Assumption of linearity between the dependent variable and the independent variables: It can only be used when there is a linear decision boundary.
2. Not flexible enough to capture complex relationships: It can underperform if the relationships are inherently non-linear.
3. Sensitive to overfitting: Especially in cases where the number of observations is lesser than the number of features.

3.2.3 Decision Tree

Definition:
A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Pros:

1. Easy to understand and interpret: Trees can be visualized, which makes them easy to interpret.
2. Requires little data preparation: Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
3. Non-parametric method: Does not assume any statistical distribution of the data.
4. Handles both numerical and categorical data: Can also handle multi-output problems.

Cons:

1. Overfitting: Decision trees can create over-complex trees that do not generalize well from the training data.
2. Instability: Small variations in the data might result in a completely different tree being generated.
3. Biased trees: If some classes dominate, decision trees can create biased trees. It is therefore recommended to balance the dataset prior to fitting.

3.2.4 Random Forest

Definition:

Random Forest is an ensemble learning method for classification, regression, and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forests correct for decision trees' habit of overfitting to their training set.

Pros:

1. Accuracy: Generally high accuracy and robustness due to the ensemble of decision trees, particularly effective in reducing the variance component of the error.
2. Handles overfitting: By averaging multiple trees, it reduces the risk of overfitting which is a common problem with regular decision trees.
3. Automatic feature selection: Random Forests can automatically handle and select the most significant features for classification.
4. Versatility: Can be used for both regression and classification tasks.

Cons:

1. Complexity and Size: Can be quite large and are significantly more complex and resource-intensive than individual decision trees.
2. Interpretability: Less interpretable compared to individual decision trees due to its complexity.
3. Performance Issues: Although they are fast to train, they can be slow to create predictions once trained, especially with large numbers of trees or deep trees.
4. Memory Consumption: The large number of trees can consume a lot of system memory and can slow down the training and prediction processes.

3.2.5 Long short term memory (LSTM)

Definition:

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) capable of learning order dependence in sequence prediction problems. This is particularly important in applications such as unsegmented, connected handwriting recognition or speech recognition. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.

Pros:

1. Handling Long-Term Dependencies: LSTMs are specifically designed to avoid the long-term dependency problem, remembering information for long periods as their default behavior.
2. Flexibility: Can be used for a variety of sequence prediction problems such as time-series

prediction, speech synthesis, and even music composition.
3. Robustness: They can deal with a range of data types, sizes, and variability, making them very robust.
4. Effectiveness in NLP: Exceptional performance on a variety of natural language processing tasks.

Cons:

1. Training Time: LSTMs are slow to train, largely due to their complexity and the large number of parameters that need to be learned.
2. Resource Intensive: They require a lot of memory and processing power, especially for large sequence data.
3. Complexity: Their internal architecture is complex, making them harder to understand and modify.
4. Vanishing Gradient Problem: Despite improvements, they can still suffer from vanishing gradient issues in practice, particularly with very long sequences.

3.2.6 K-nearest neighbors (KNN)

Definition:

K-Nearest Neighbors (KNN) is a simple, easy-to-implement supervised machine learning algorithm that can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. This algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

Pros:

1. Simplicity: KNN is very simple and easy to implement.
2. No Training Period: KNN doesn't require a training period as it stores all of the training data. Model building corresponds simply to storing and perhaps indexing the training data.
3. Naturally handles multi-class cases: Can be used for both binary and multi-class classification problems.
4. Very flexible: Features or distances can be weighted to fine-tune the outcome on specific data sets.

Cons:

1. Scalability: KNN can be very slow if the dataset is large, as it searches for the nearest neighbors through all the dataset.
2. High Memory Requirement: Needs to store all of the training data during the testing phase which can lead to high memory storage.
3. Sensitive to Irrelevant Features: Performs poorly if there are many irrelevant features in the data.
4. Sensitive to the scale of the data: All features need to be on the same scale; otherwise, the larger-scale features will dominate the outcome.

3.2.7 Feedforward neural Network(FNN)

Definition:

A Feedforward Neural Network (FNN) is an artificial neural network wherein connections between the nodes do not form a cycle. This type of neural network consists of multiple layers including an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next layer in the network. Nodes in the same layer do not connect with each other, and data moves in only one direction, forward, from the input nodes, through the hidden nodes (if any), and to the output nodes.

Pros:

1. Model Flexibility: Can model complex non-linear relationships that other algorithms can't easily

model.
2. Scalability: Performs well with large datasets and can be extended with more layers for improved complexity.
3. Adaptability: Can be used for both regression and classification problems effectively.

Cons:

1. Overfitting Risk: Without proper regularization, they tend to overfit, especially with not enough training data.
2. Computationally Intensive: Training FNNs require a lot of computation, particularly as the network size grows.
3. Data Sensitivity: Requires careful preprocessing of data — normalization, encoding categorical variables, etc.
4. Black Box Nature: Neural networks in general are often considered black boxes; i.e., their predictions are not easy to interpret.

3.3 Model Optimization

In model optimization, we employed hyperparameter tuning, scaling of features, and ensemble methods.

3.3.1 Hyperparameter Tuning

Hyperparameter tuning is utilized using **GridSearchCV**, a technique that systematically searches through a predefined grid of hyperparameters to identify the optimal combination that yields the best model performance. By tuning hyperparameters such as regularization strength and penalty in logistic regression or maximum depth and minimum samples split in decision trees, the models can better capture the underlying patterns in the data, leading to improved accuracy and generalization.

3.3.2 Scaling of features

Scaling of features is another important optimization technique applied in the provided codes. Standard scaling using **StandardScaler** is performed to standardize the distribution of features, ensuring that they have a mean of zero and a standard deviation of one. This preprocessing step helps in improving the convergence rate of optimization algorithms and prevents features with larger magnitudes from dominating the model training process, resulting in more stable and efficient model training.

3.3.3 Ensemble Methods

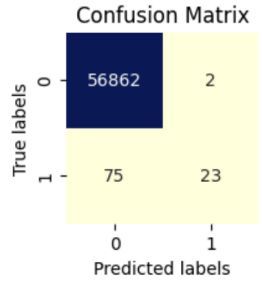
Ensemble methods involve combining multiple base models to form a stronger predictive model, leveraging the diversity of individual models to improve overall performance. Techniques like bagging (e.g., Random Forest) and boosting (e.g., AdaBoost) are effective in reducing variance and bias, respectively, leading to more robust and accurate predictions.

3.4 Model Performance Analysis

3.4.1 Results from Each Model Application on Imbalanced Data

3.4.1.1 Naive Bayes Results on Imbalanced Data

Accuracy: 0.9986482216214319

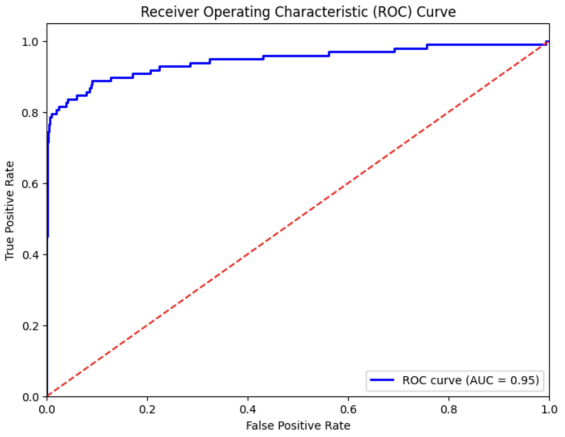


F1 Score: 0.37398373983739835
Precision: 0.92
Recall: 0.23469387755102042

Classification Report:

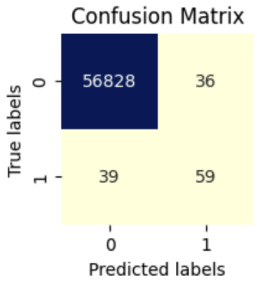
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.92	0.23	0.37	98
accuracy			1.00	56962
macro avg	0.96	0.62	0.69	56962
weighted avg	1.00	1.00	1.00	56962

ROC_AUC: 0.9453806001860509



3.4.1.2 Logistic Regression Results on Imbalanced Data

Accuracy: 0.9986833327481479

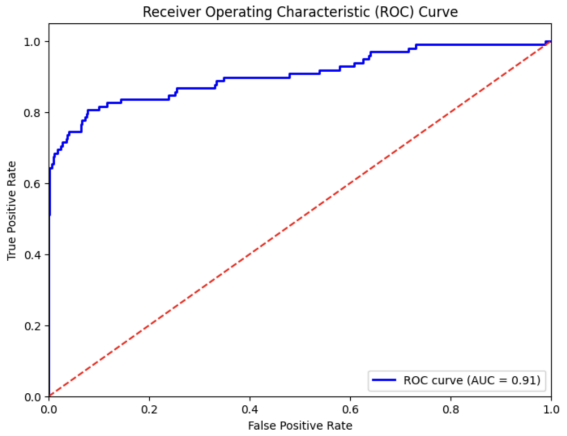


F1 Score: 0.6113989637305699
Precision: 0.6210526315789474
Recall: 0.6020408163265306

Classification Report:

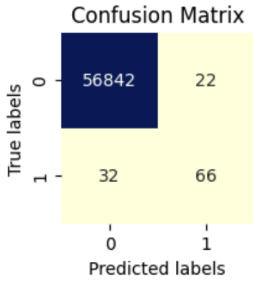
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.62	0.60	0.61	98
accuracy			1.00	56962
macro avg	0.81	0.80	0.81	56962
weighted avg	1.00	1.00	1.00	56962

ROC_AUC: 0.9050803636029539



3.4.1.3 Decision Tree Results on Imbalanced Data

Accuracy: 0.9990519995786665

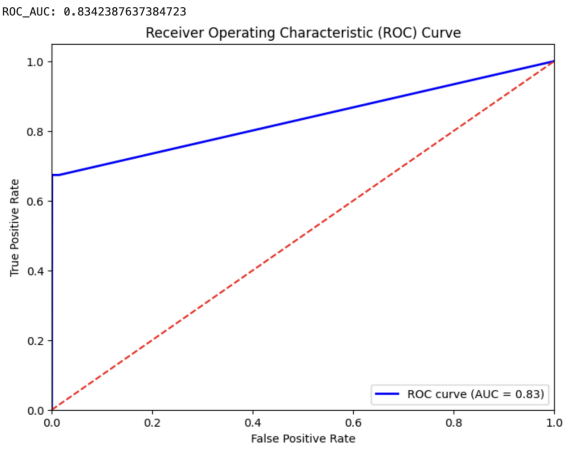


F1 Score: 0.7096774193548386
Precision: 0.75
Recall: 0.673469387755102

Classification Report:

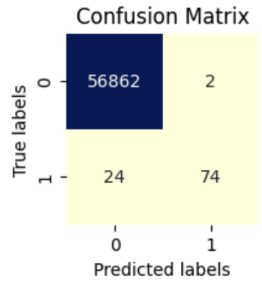
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.75	0.67	0.71	98
accuracy			1.00	56962
macro avg	0.87	0.84	0.85	56962
weighted avg	1.00	1.00	1.00	56962

ROC_AUC: 0.8342387637384723



3.4.1.4 Random Forest Results on Imbalanced Data

Accuracy: 0.9995435553526912

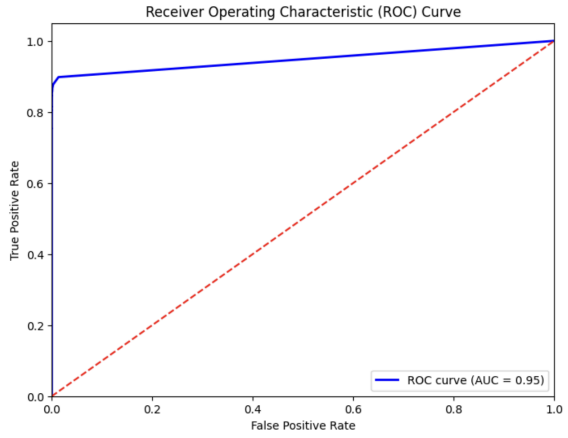


F1 Score: 0.8505747126436782
Precision: 0.9736842105263158
Recall: 0.7551020408163265

Classification Report:

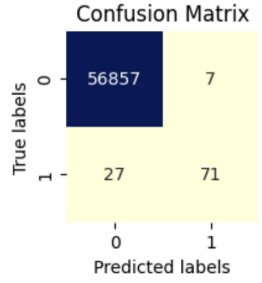
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.97	0.76	0.85	98
accuracy			1.00	56962
macro avg	0.99	0.88	0.93	56962
weighted avg	1.00	1.00	1.00	56962

ROC_AUC: 0.9480902518576366



3.4.1.5 LSTM Results on Imbalanced Data

Accuracy: 0.999403110845827

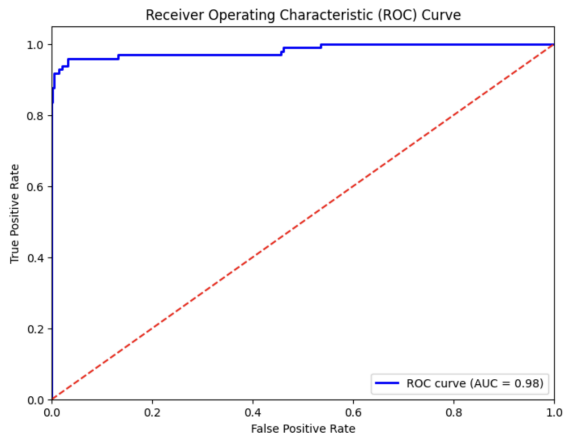


F1 Score: 0.8068181818181819
Precision: 0.9102564102564102
Recall: 0.7244897959183674

Classification Report:

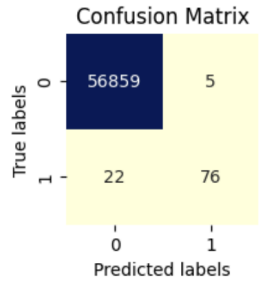
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.91	0.72	0.81	98
accuracy			1.00	56962
macro avg	0.95	0.86	0.90	56962
weighted avg	1.00	1.00	1.00	56962

AUC-ROC: 0.9826097785765967



3.4.1.6 KNN Results on Imbalanced Data

Accuracy: 0.9995259997893332

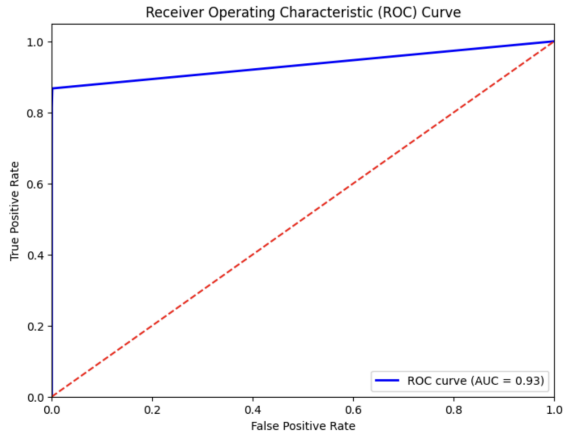


F1 Score: 0.8491620111731844
Precision: 0.9382716049382716
Recall: 0.7755102040816326

Classification Report:

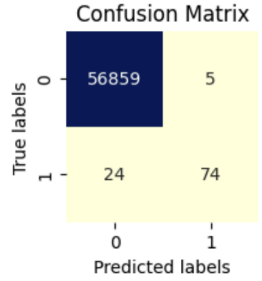
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.94	0.78	0.85	98
accuracy			1.00	56962
macro avg	0.97	0.89	0.92	56962
weighted avg	1.00	1.00	1.00	56962

AUC-ROC: 0.9335599690776705



3.4.1.7 FNN Results on Imbalanced Data

Accuracy: 0.9994908886626171

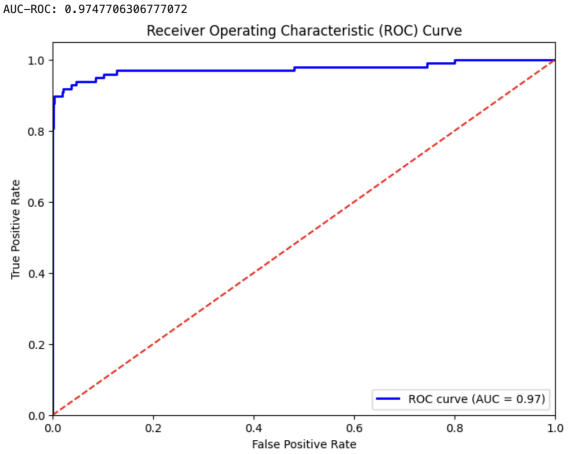


F1 Score: 0.8361581920903955
Precision: 0.9367088607594937
Recall: 0.7551020408163265

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.94	0.76	0.84	98
accuracy			1.00	56962
macro avg	0.97	0.88	0.92	56962
weighted avg	1.00	1.00	1.00	56962

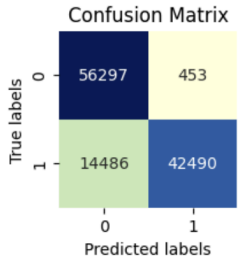
AUC-ROC: 0.9747706306777072



3.4.2 Results from Each Model Application on Balanced Data

3.4.2.1 Naive Bayes Results on Balanced Data

Accuracy: 0.8686404164395125

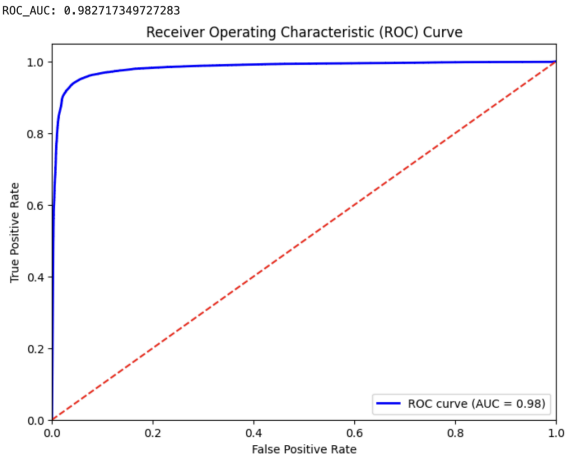


F1 Score: 0.8504888960057647
Precision: 0.9894511328970962
Recall: 0.745752597584948

Classification Report:

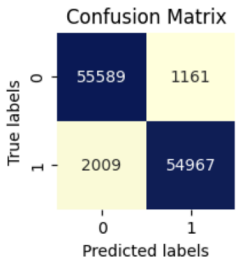
	precision	recall	f1-score	support
0	0.80	0.99	0.88	56750
1	0.99	0.75	0.85	56976
accuracy			0.87	113726
macro avg	0.89	0.87	0.87	113726
weighted avg	0.89	0.87	0.87	113726

ROC_AUC: 0.982717349727283



3.4.2.2 Logistic Regression Results on Balanced Data

Accuracy: 0.9721259870214375

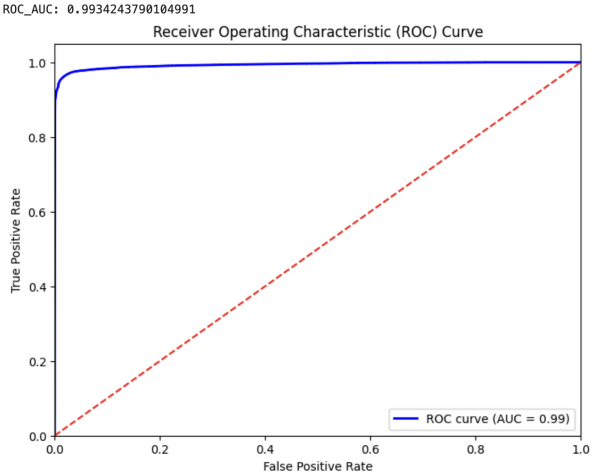


F1 Score: 0.9719726976941576
Precision: 0.9793151368301026
Recall: 0.9647395394552092

Classification Report:

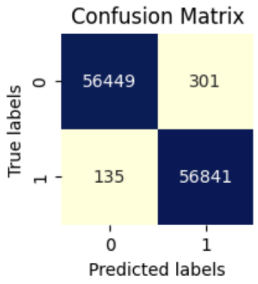
	precision	recall	f1-score	support
0	0.97	0.98	0.97	56750
1	0.98	0.96	0.97	56976
accuracy			0.97	113726
macro avg	0.97	0.97	0.97	113726
weighted avg	0.97	0.97	0.97	113726

ROC_AUC: 0.9934243790104991



3.4.2.3 Decision Tree Results on Balanced Data

Accuracy: 0.9961662240824438

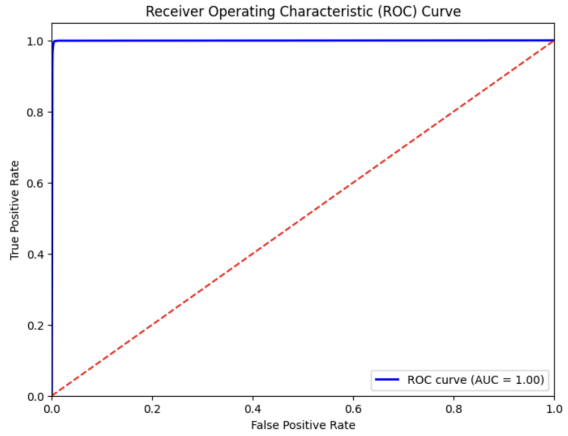


F1 Score: 0.9961793932596085
Precision: 0.9947324209863148
Recall: 0.9976305812973884

Classification Report:

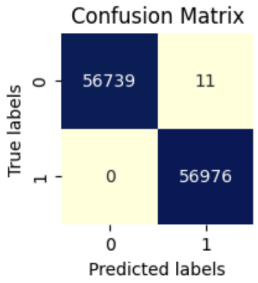
	precision	recall	f1-score	support
0	1.00	0.99	1.00	56750
1	0.99	1.00	1.00	56976
accuracy			1.00	113726
macro avg	1.00	1.00	1.00	113726
weighted avg	1.00	1.00	1.00	113726

ROC_AUC: 0.9988852839498382



3.4.2.4 Random Forest Results on Balanced Data

Accuracy: 0.999903276295658

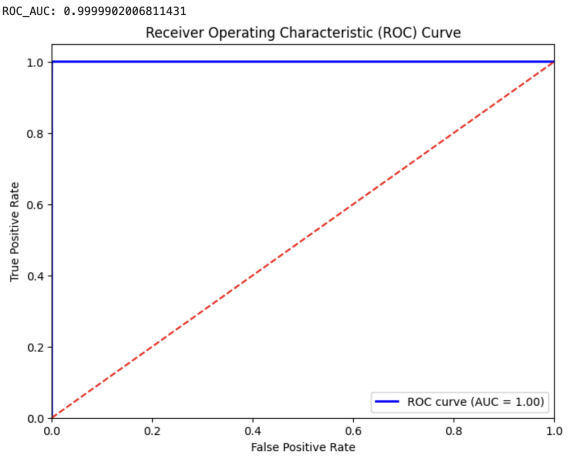


F1 Score: 0.9999034774444338
Precision: 0.9998069735202766
Recall: 1.0

Classification Report:

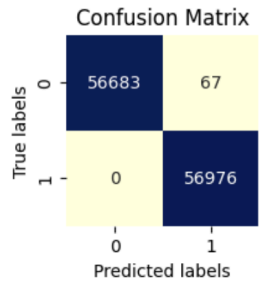
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56750
1	1.00	1.00	1.00	56976
accuracy			1.00	113726
macro avg	1.00	1.00	1.00	113726
weighted avg	1.00	1.00	1.00	113726

ROC_AUC: 0.9999902006811431



3.4.2.5 LSTM Results on Balanced Data

Accuracy: 0.9994108647099168

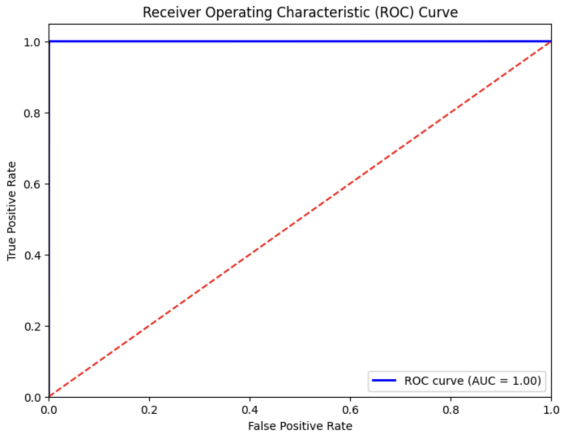


F1 Score: 0.999412378638648
Precision: 0.9988254474694529
Recall: 1.0

Classification Report:

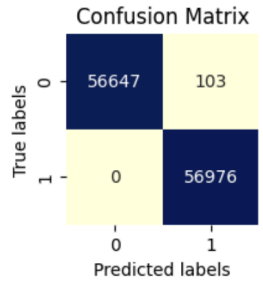
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56750
1	1.00	1.00	1.00	56976
accuracy			1.00	113726
macro avg	1.00	1.00	1.00	113726
weighted avg	1.00	1.00	1.00	113726

AUC-ROC: 0.999976808536433



3.4.2.6 KNN Results on Balanced Data

Accuracy: 0.9990943144047975

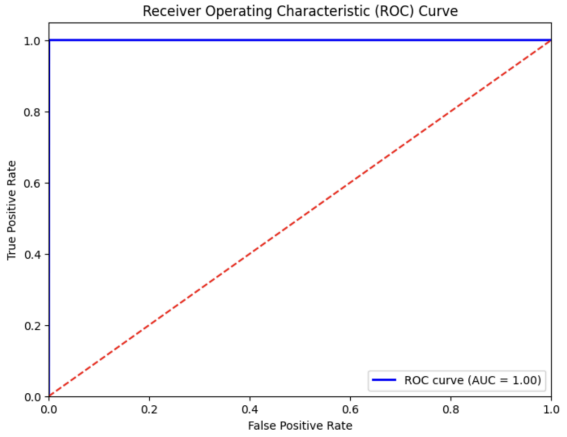


F1 Score: 0.9990969269212222
Precision: 0.9981954834527584
Recall: 1.0

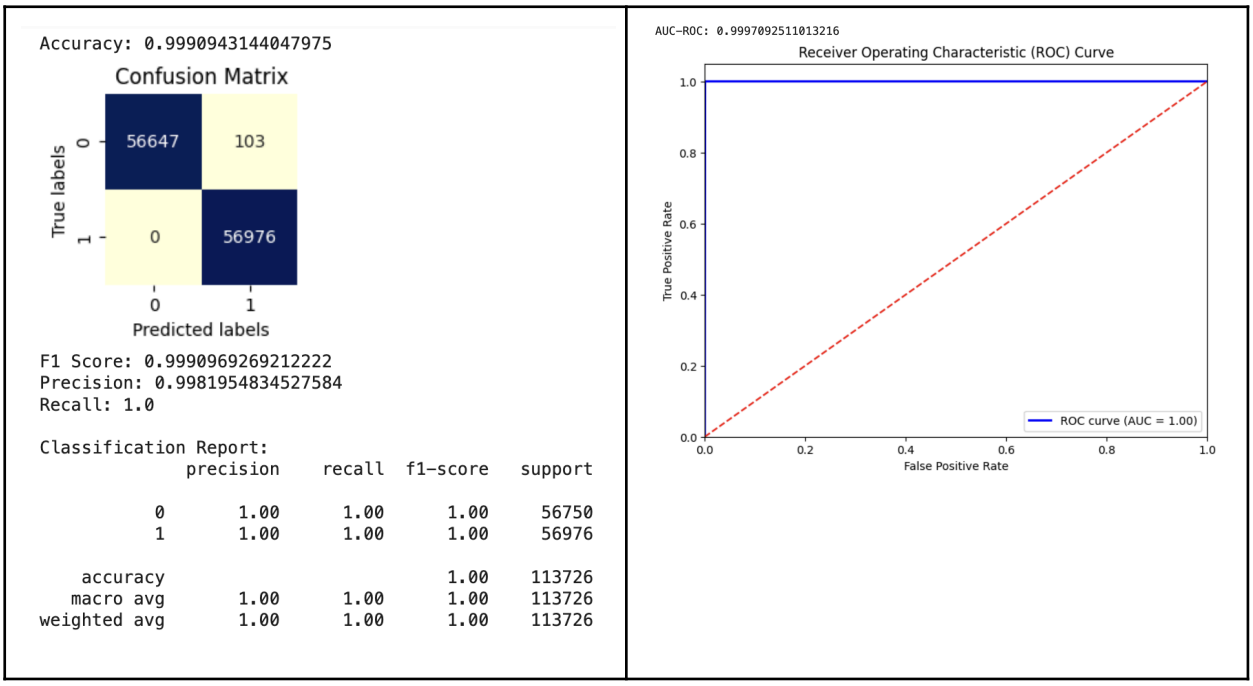
Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56750
1	1.00	1.00	1.00	56976
accuracy			1.00	113726
macro avg	1.00	1.00	1.00	113726
weighted avg	1.00	1.00	1.00	113726

AUC-ROC: 0.9997092511013216



3.4.2.7 FNN Results on Balanced Data



3.4.3 Comparison of Each Model Application

3.4.3.1 Comparison of Each Model Application on Imbalanced Data

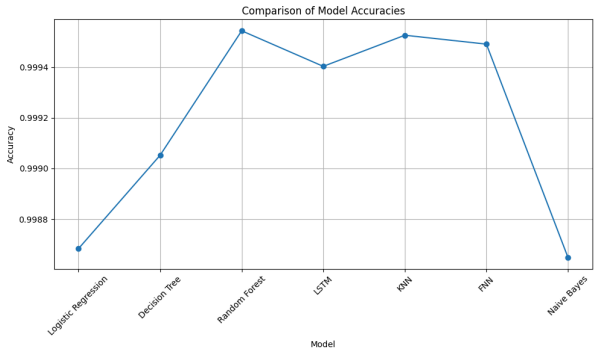
Models	Accuracy	F1 Score	Precision	Recall	ROC_AUC
Naive Bayes	0.9986482216	0.3739837398	0.92	0.2346938776	0.9453806002
Logistic Regression	0.9986833327	0.6113989637	0.6210526316	0.6020408163	0.9050803636
Decision Tree	0.9990519996	0.7096774194	0.75	0.6734693878	0.8342387637
Random Forest	0.9995435554	0.8505747126	0.9736842105	0.7551020408	0.9480902519
LSTM	0.9994031108	0.8068181818	0.9102564103	0.7244897959	0.9826097786
KNN	0.9995259998	0.8491620112	0.9382716049	0.7755102041	0.9335599691
FNN	0.9994908887	0.8361581921	0.9367088608	0.7551020408	0.9747706307

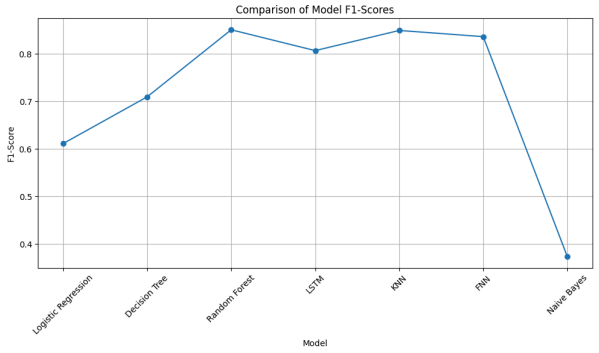
3.4.3.2 Comparison of Each Model Application on Balanced Data

Models	Accuracy	F1 Score	Precision	Recall	ROC_AUC
Naive Bayes	0.8686404164	0.850488896	0.9894511329	0.7457525976	0.9827173497
Logistic Regression	0.972125987	0.9719726977	0.9793151368	0.9647395395	0.993424379
Decision Tree	0.9961662241	0.9961793933	0.994732421	0.9976305813	0.9988852839
Random Forest	0.9999032763	0.9999034774	0.9998069735	1	0.9999902007
LSTM	0.9994108647	0.9994123786	0.9988254475	1	0.9999768085
KNN	0.9990943144	0.9990969269	0.9981954835	1	0.9997092511
FNN	0.9990943144	0.9990969269	0.9981954835	1	0.9997092511

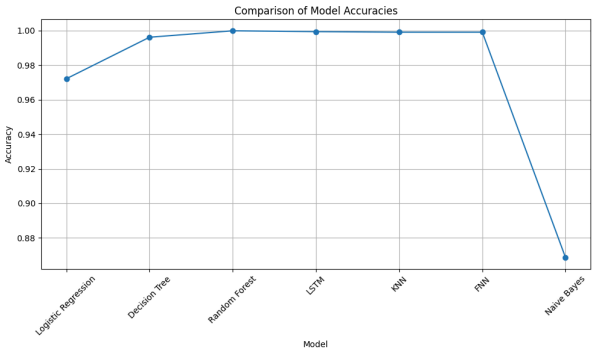
3.4.4 Comparison of Each Model Application on Imbalanced Data & Balanced Data

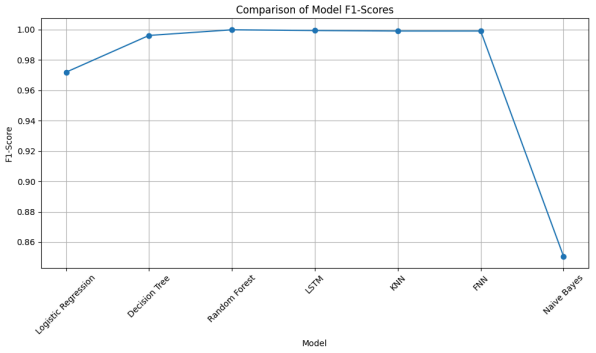
Comparison of Results on Imbalanced Data

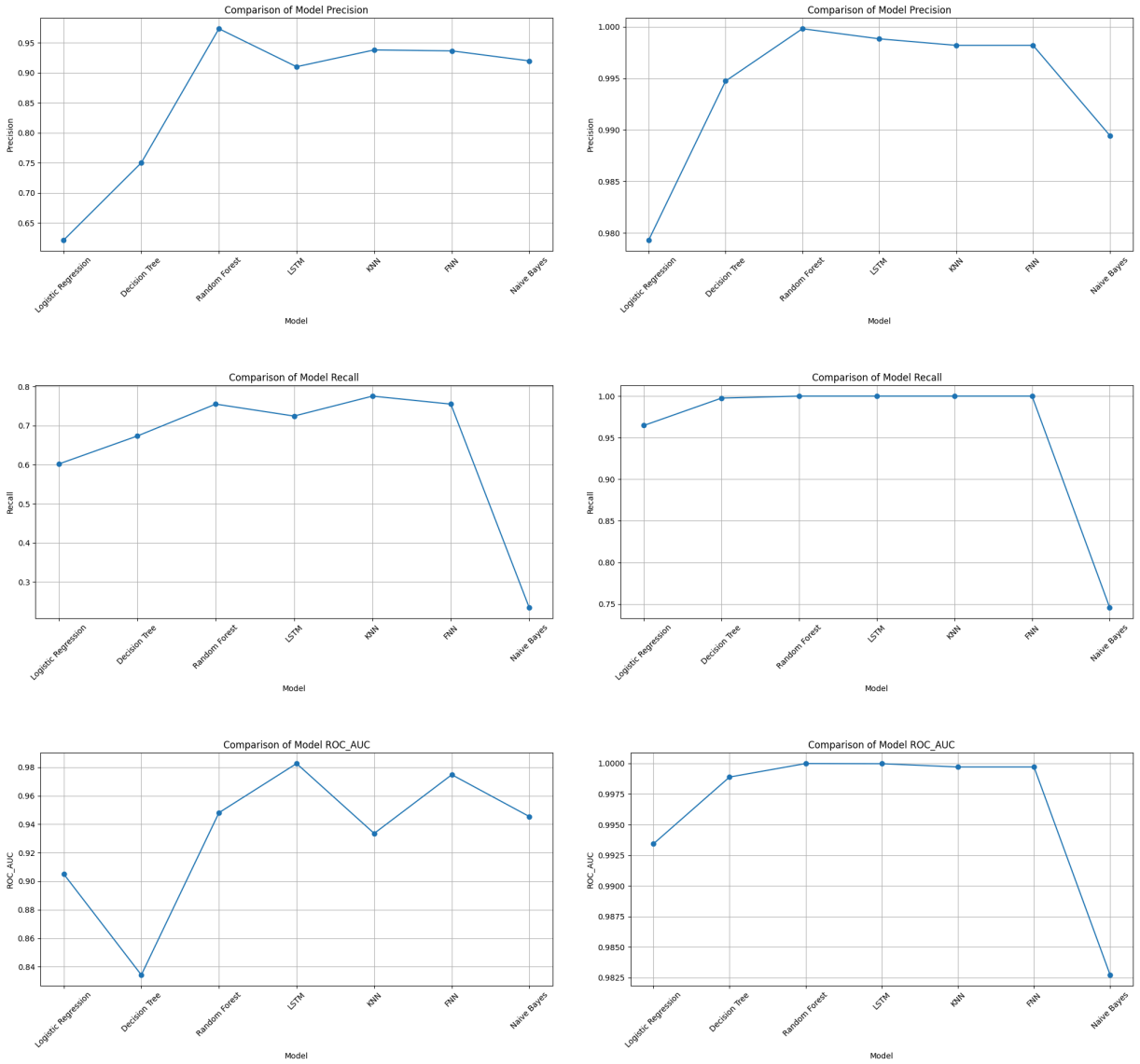




Comparison of Results on Balanced Data







3.5 Project Inquiry: Framing the Research Queries

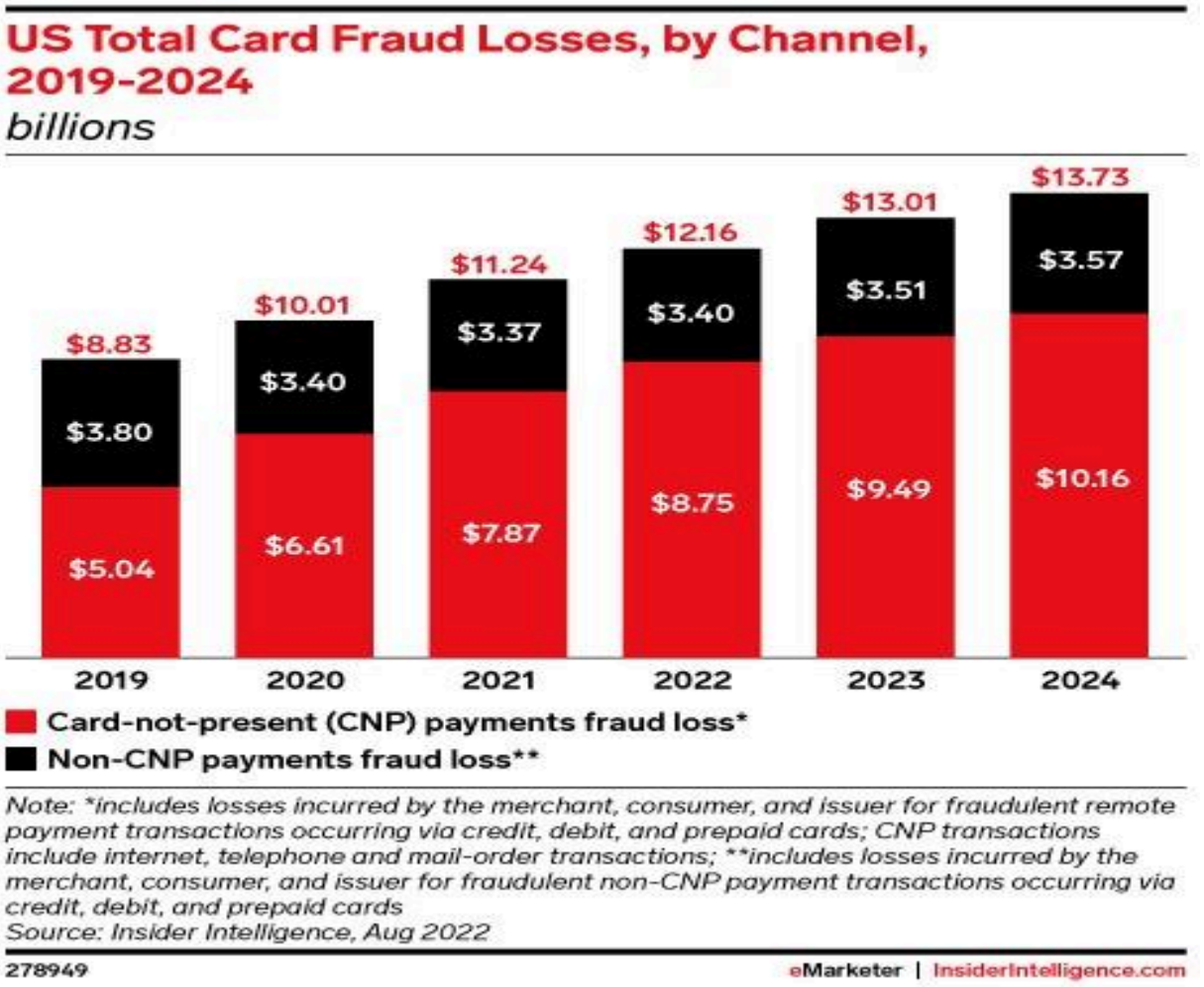
1. **Can we accurately classify credit card transactions as fraudulent or not based on the provided features?**

In the context of a real-world dataset characterized by significant class imbalance, classification posed challenges. However, through preprocessing techniques and the implementation of the Synthetic Minority Over-sampling Technique (SMOTE), the dataset was rebalanced, significantly enhancing classification performance. Consequently, models trained on the optimized balanced dataset demonstrated improved accuracy in classifying credit card transactions with greater precision.

2. **How does the distribution of transaction amounts differ between fraudulent and non-fraudulent transactions?**

The distribution of transaction amounts typically differs between fraudulent and non-fraudulent transactions. Fraudulent transactions often involve smaller amounts to avoid detection, while legitimate transactions span a wider range of values. Additionally, fraudulent transactions may exhibit unusual patterns, such as repeated small transactions or occasional large transactions, which deviate from the typical spending behavior of legitimate users. Analyzing the distribution of transaction amounts can help

identify anomalous patterns indicative of fraudulent activity and inform the development of effective fraud detection models.



3. **Are there any patterns or trends in the time elapsed between transactions for fraudulent transactions compared to non-fraudulent ones?**

Yes, there can be patterns or trends in the time elapsed between transactions for fraudulent transactions compared to non-fraudulent ones. Fraudulent transactions may exhibit certain distinct patterns in the time intervals between transactions, such as:

1. *Bursts of Activity*: Fraudulent transactions may occur in rapid succession within a short time frame, indicating coordinated fraudulent activity.
2. *Unusual Time Intervals*: Fraudulent transactions may occur at irregular intervals, deviating from the typical spending patterns of legitimate users.
3. *Off-Hours Activity*: Fraudulent transactions may be more prevalent during off-hours or non-peak times when monitoring systems are less likely to be active.
4. *Consistent Time Intervals*: In some cases, fraudsters may follow consistent time intervals between transactions to mimic regular spending behavior, although these intervals may still differ from those of legitimate users.

4. **How effective is the SMOTE algorithm in balancing the dataset, and does it improve the performance of classification models?**

The Synthetic Minority Over-sampling Technique (SMOTE) effectively balances imbalanced datasets by generating synthetic samples for the minority class. By creating artificial instances, SMOTE helps mitigate class imbalance, improving classification model performance. Its effectiveness varies based on

the degree of imbalance, dataset complexity, and chosen algorithm. SMOTE prevents model bias towards the majority class, enhancing generalization and performance metrics such as accuracy, precision, recall, and F1-score, especially for minority class instances. However, it may introduce noise or overfitting, necessitating careful evaluation through experimentation and cross-validation to determine its impact on model performance.

5. **Are there any specific challenges or limitations encountered during the implementation of the SMOTE algorithm on this dataset?**

During the implementation of the SMOTE algorithm on this dataset, challenges may include data complexity, especially when the minority class is intricate or hard to discern from the majority. The curse of dimensionality can hinder SMOTE's effectiveness in high-dimensional spaces, impacting its ability to accurately capture the data distribution. Model sensitivity to class imbalance and computational overhead in generating synthetic samples are additional concerns. Furthermore, evaluating model performance may be biased due to SMOTE's impact on dataset balance. Addressing these challenges entails careful consideration of dataset characteristics, model selection, and alternative oversampling techniques to mitigate limitations.

6. **How do different classification algorithms (e.g., logistic regression, decision trees, random forests) perform in detecting fraudulent transactions?**

After comparing all the models, including Naive Bayes, Logistic Regression, Decision Tree, Random Forest, LSTM, KNN, and FNN, Random Forest emerges as the best model for detecting fraudulent transactions. It demonstrates robustness against overfitting, handles high-dimensional data effectively, and offers superior generalization performance compared to other algorithms. Moreover, its ensemble learning approach enhances accuracy by aggregating multiple decision trees' predictions. Thus, Random Forest proves to be the most reliable and effective choice for fraud detection in this context.

7. **What is the impact of feature scaling (e.g., normalization, standardization) on the performance of classification models?**

Feature scaling, including normalization and standardization, significantly impacts classification model performance. These techniques ensure all features contribute equally, preventing dominance by those with larger magnitudes. They improve optimization algorithm convergence rates, leading to faster training and stable models. For algorithms relying on distance metrics like K-nearest neighbors (KNN) and Support Vector Machines (SVM), scaling ensures meaningful distances between data points. Overall, feature scaling enhances model stability, performance, and interpretability, making it a crucial preprocessing step in machine learning workflows.

8. **Can we optimize model performance further by tuning hyperparameters or exploring ensemble methods?**

Yes, optimizing model performance can be achieved further by tuning hyperparameters or exploring ensemble methods. Hyperparameter tuning involves adjusting the settings that control the learning process of the algorithm, such as regularization parameters or tree depths, to find the best combination for improved performance. Ensemble methods, such as bagging, boosting, or stacking, combine multiple base models to create a stronger predictive model, leveraging the diversity of individual models to improve overall accuracy and robustness. By systematically exploring hyperparameters and ensemble techniques, we can enhance model performance and achieve better results in detecting fraudulent transactions.

9. **How will the performance vary for these models over the balanced set vs imbalanced set?**

Overall the performance of the models is better for the balanced dataset with respect to the metrics that we have considered viz. accuracy, precision, recall and AUC. In case of im-balanced dataset though we have a high accuracy still we have a very poor recall and precision.

10. **What additional steps or techniques can be employed to optimize model performance further, beyond feature scaling and algorithm selection, when detecting fraudulent transactions?**

Beyond feature scaling and algorithm selection, optimizing model performance in fraud detection involves several additional steps. Feature engineering can enhance model effectiveness by creating new features or transforming existing ones to capture more relevant information about transactions. Anomaly detection techniques, such as unsupervised learning, can identify unusual patterns or outliers in transaction data, complementing supervised approaches. Ensemble learning methods, like stacking or blending, combine multiple models to leverage their strengths and improve predictive accuracy. Advanced preprocessing, including outlier removal and feature selection, can enhance data quality. Robust cross-validation helps assess model generalization, while model calibration improves probability estimates' reliability. Employing these additional steps and techniques can further optimize model performance and enhance fraud detection system accuracy and effectiveness.

3.6 Conclusion

In conclusion, when evaluating the performance of various models for credit card fraud detection, several key insights emerge. Firstly, on imbalanced data, models such as Naive Bayes and Logistic Regression demonstrate high accuracy but struggle with achieving balanced precision and recall scores. In contrast, Decision Trees, Random Forests, LSTM, KNN, and FNN exhibit superior performance across multiple metrics, particularly Random Forests and LSTM which outperform others in terms of F1 score and ROC_AUC.

Upon balancing the dataset using SMOTE, a notable enhancement in model performance is observed across the board. Specifically, Naive Bayes shows a substantial improvement in recall, albeit with a trade-off in precision. Logistic Regression maintains high precision and recall scores, showcasing its robustness. Decision Trees, Random Forests, LSTM, KNN, and FNN continue to excel, with all models achieving near-perfect scores in accuracy, F1 score, precision, recall, and ROC_AUC.

Ultimately, the results underscore the importance of model selection and dataset balancing in credit card fraud detection. While simpler models like Naive Bayes and Logistic Regression may offer initial insights, more complex algorithms such as Random Forests, LSTM, KNN, and FNN prove to be more effective in capturing nuanced patterns of fraudulent behavior.