Table of Contents

1. Introduction

1.1 Overview

The project aims to develop a machine learning model to detect fraudulent credit card transactions. Credit card fraud is a significant concern for financial institutions and customers alike, leading to financial losses and compromised security. The goal is to create a system that can accurately identify fraudulent activities in real-time, mitigating risks and ensuring the security of transactions.

1. Data Collection: Gather a comprehensive dataset containing credit card transactions, including both legitimate and fraudulent ones.

2. Exploratory Data Analysis (EDA): Exploring the dataset to understand patterns and handling the highly imbalanced dataset.

3. Feature Engineering: Extract relevant features or create new ones that might enhance the model's predictive power.

4. Model Selection: Experiment with various machine learning algorithms such as logistic regression, random forests, Ensemble Learning to identify the most suitable model for fraud detection.

5. Model Training: Train the selected model using the prepared dataset, optimizing hyperparameters to achieve the best performance.

6. Model Evaluation: Assess the model's performance using metrics like accuracy, precision, recall, and F1-score.

1.2 Problem Definition and Scope

Credit card fraud involves unauthorized or deceptive transactions using someone else's credit card information. The problem lies in distinguishing between legitimate and fraudulent transactions, as fraudulent activities often mimic legitimate ones to evade detection. The challenge is to build a

robust predictive model that can accurately flag suspicious transactions while minimizing false positives that might inconvenience genuine cardholders.

In a real-time Credit Card Fraud Detection Project, the scope encompasses a variety of technical and operational aspects to ensure the continual and effective identification of fraudulent activities. Here are key elements within this scope:

1. Real-Time Data Processing: Implementing systems that can handle incoming credit card transactions in real-time, analyzing each transaction as it occurs. This involves setting up pipelines or streaming processes to continuously ingest, process, and evaluate transactions promptly.

2. Model Deployment and Integration: Deploying the trained fraud detection model into a real-time environment, ensuring seamless integration with transaction processing systems. This might involve utilizing APIs or microservices for model inference.

3. Scalability and Performance: Building systems that can handle large volumes of transactions efficiently without sacrificing performance. Ensuring the model's scalability to accommodate increased transaction loads while maintaining accuracy and speed.

4. Monitoring and Analytics: Setting up monitoring tools and dashboards to track the performance of the fraud detection system in real-time. Analyzing model performance metrics, false positives, false negatives, and overall accuracy to make necessary adjustments.

5. Regulatory Compliance and Security: Ensuring compliance with regulatory requirements related to data privacy and security. Implementing robust security measures to safeguard sensitive financial information.

The real-time scope of a Credit Card Fraud Detection Project requires a continuous effort to maintain the efficiency, accuracy, and adaptability of the detection system to stay ahead of evolving fraudulent tactics and protect against potential risks effectively.

2. Data Preparation

2.1 Dataset Used

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Owner of Dataset: The dataset has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group (*http://mlg.ulb.ac.be*) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

3. Exploratory Data Analysis

Getting Insights About the Dataset

1.Using .head() [To get the first five rows from the dataset]

```
credit_card_data.head()
```
✓ 0.0s Python

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
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
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
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
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
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

5 rows × 31 columns

2.Using .tail() [To get the last five rows from the dataset]

```
[11]  credit_card_data.isnull().sum()
```
✓ 0.0s

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
...
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

4. Using .columns [To get all the column names of the dataset]

```
credit_card_data.columns
```

✓ 0.0s Python

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

5. Using .describe() [To get the statistical summary of the dataset]

```
credit_card_data.describe()
```

✓ 0.3s Python

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.

8 rows × 31 columns

6. Using .info [To get the information related to the dataset]

```
credit_card_data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
...
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

3.Using .shape [To get the dimension of the dataset]

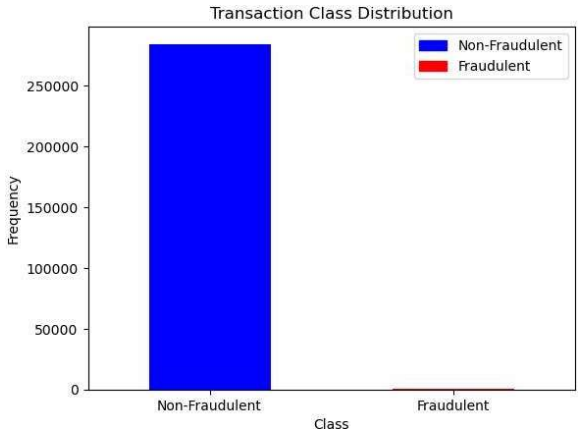
```
print("df.shape -->", credit_card_data.shape)
print("Rows     -->", credit_card_data.shape[0])
print("Columns  -->", credit_card_data.shape[1])
```

✓ 0.0s Python

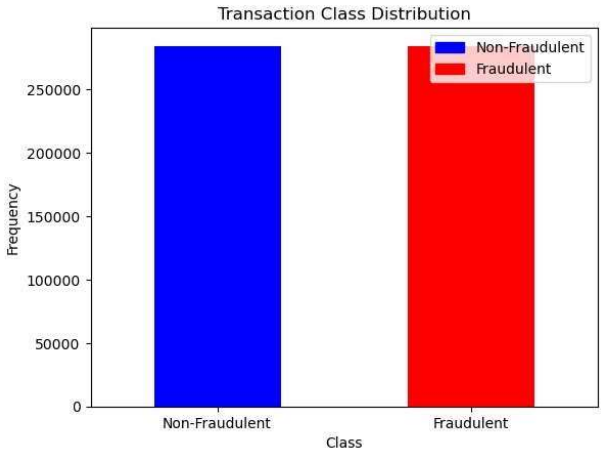
```
df.shape --> (284807, 31)
Rows     --> 284807
Columns  --> 31
```

Data visualization

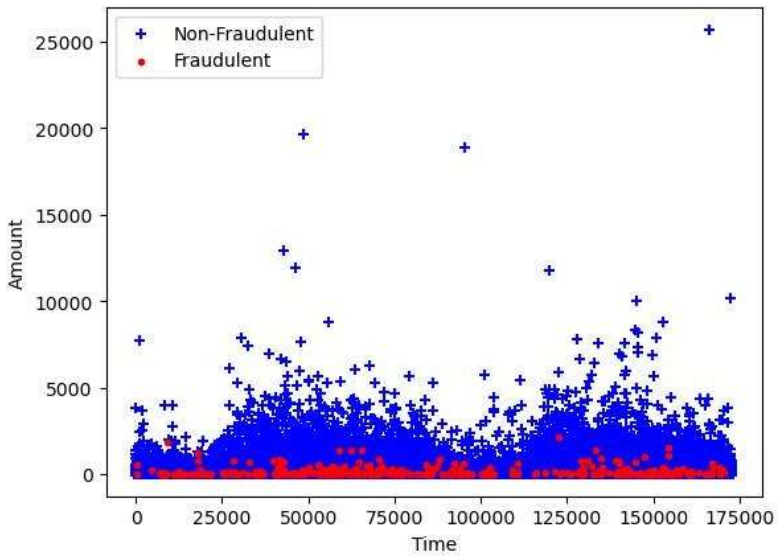
1. **Bar graph classification between non-fraudulent and fraudulent cases before normalization.**



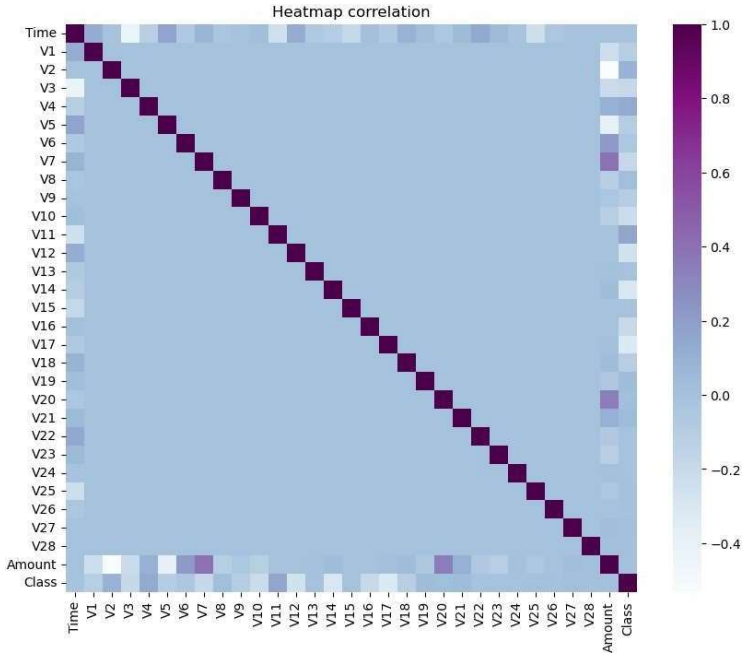
2. **Bar graph classification between non-fraudulent and fraudulent cases after normalization.**



3. Scatter plot classification between non-fraudulent and fraudulent cases wrt ‘Amount’ and ‘Time’.

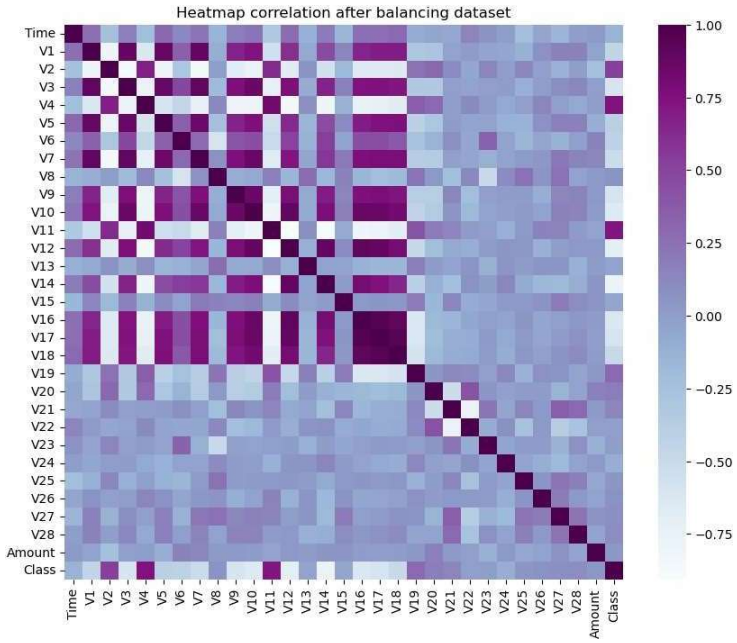


5. Correlation between features and the likelihood of the transaction to be fraud on imbalanced dataset.



In the HeatMap we can clearly see that most of the features do not correlate to other features but there are some features that either has a positive or a negative correlation with each other. For example, V2 and V5 are highly negatively correlated with the feature called Amount. We also see some correlation with V20 and Amount.

6. Correlation between features and the likelihood of the transaction to be fraud on balanced dataset.



Negative and Positive Correlations can be seen in the matrix above:

- Features, V14, V12, V10 and V3 show negative correlation towards the 'Class', As, lower are these values, more likely the transaction is Fraud.
- Features V4, V11, V2 and V19 show positive correlation. As, higher are these values, more likely the transaction is Fraud.

4. Methods Used

Algorithm

1. Logistic Regression:

Definition:
Logistic Regression is a statistical method used for binary classification problems, where the outcome or target variable is categorical and has two classes (e.g., yes/no, 0/1, true/false). It predicts the probability that a given input belongs to a particular class.

Theory:
Logistic Regression uses the logistic function (also known as the sigmoid function) to model the relationship between the independent variables and the probability of a particular outcome. The logistic function converts any input into a value between 0 and 1.

Mathematical Formula:
$P(Y=1|X)=1/(1+e^{-(b0+b1X)})$

Where:
- (P(Y=1|X)) is the probability of the dependent variable being 1 given the input (X).
- (b0) is the intercept.
- (b1 X) represents the linear combination of features and their coefficients.

Usage:
It's applied here as the initial classifier to establish a baseline performance. The model learns the relationship between input features and the likelihood of a transaction being fraudulent.

2. Random Forest Classifier:

Definition:
A Random Forest is an ensemble learning method used in machine

learning for both classification and regression tasks. It operates by constructing a multitude of decision trees during the training phase and outputs the mode (for classification) or the mean prediction (for regression) of the individual trees.

Theory:

The "random" in Random Forest refers to two key aspects:

1. Random Sampling of Data: Each tree in the forest is trained on a random subset of the training data (with replacement), known as bootstrapping. This allows each tree to be slightly different from the others.

2. Random Subset of Features: At each node of the decision tree, only a random subset of features is considered for splitting. This randomness prevents individual trees from being highly correlated and can lead to better overall performance.

The final prediction in a Random Forest is determined by aggregating the predictions of all the trees, either through majority voting (for classification) or averaging (for regression). This ensemble approach often results in improved accuracy, robustness to overfitting, and better generalization to unseen data compared to individual decision trees.

Usage:

It's utilized as a more sophisticated classifier compared to Logistic Regression. Random Forest can capture complex interactions among features and tends to handle noise well.

3. AdaBoost Classifier:

Definition:

AdaBoost, short for Adaptive Boosting, is a popular ensemble learning method in machine learning. It combines multiple "weak" or base learners (classifiers) to create a strong classifier. The basic idea behind AdaBoost is to sequentially train these weak learners, giving more weight to misclassified samples in each iteration. By doing this, subsequent learners focus more on the difficult-to-classify examples, gradually improving the overall performance of the ensemble.

Theory:

Here's a breakdown of how AdaBoost works:

1. Initialization: Each sample in the dataset is initially assigned an equal weight.

2. Training Weak Learners: AdaBoost trains a series of weak learners (often decision trees or simple classifiers) on the dataset. Each weak learner is trained to minimize the error, with more weight given to misclassified samples in the previous iteration.

3. Weight Update: After each iteration, the weights of misclassified samples are increased, making them more influential in the next iteration. This allows subsequent weak learners to focus on the mistakes of the previous ones.

4. Combining Weak Learners: Each weak learner is assigned a weight based on its accuracy, and their predictions are combined to create a strong learner. The final prediction is typically a weighted sum of the weak learners' predictions.

5. Final Model: AdaBoost continues this iterative process until a predefined number of weak learners are trained or until a certain level of accuracy is achieved.

Usage:

AdaBoost is effective in improving classification accuracy by focusing more on misclassified instances. It sequentially corrects the errors of the weak classifiers and improves the overall performance.

4. XGBoost (Extreme Gradient Boosting):

Definition:

XGBoost, which stands for eXtreme Gradient Boosting, is an advanced implementation of gradient boosting, a machine learning technique used for classification and regression problems.

Theory:

Ensemble learning involves combining multiple base models to build a

more robust and accurate model. XGBoost is an ensemble learning method based on decision trees, specifically designed to improve upon the shortcomings of traditional gradient boosting algorithms.

The core principles of XGBoost involve:

1. Gradient Boosting: It works by building trees sequentially, where each subsequent tree corrects the errors made by the previous one. It minimizes the overall prediction error by adding weak learners, which are decision trees in this context.

2. Regularization: XGBoost includes L1 and L2 regularization terms in its objective function to control model complexity and prevent overfitting. This helps in penalizing overly complex models and discourages them from fitting to noise in the training data.

3. Customizable Objective Function: It allows users to define their own objective functions, offering flexibility in solving different types of problems.

4. Tree Pruning: XGBoost employs a technique called pruning, which removes splits that do not provide significant information gain during tree construction. This helps in controlling tree depth and reducing overfitting.

5. Parallel Processing: It is optimized for parallel computation, making it much faster compared to other gradient boosting implementations.

Usage:

It's employed here as an advanced boosting algorithm. XGBoost often outperforms other algorithms due to its ability to handle complex relationships in data and its regularization techniques to prevent overfitting.

SMOTE (Synthetic Minority Over-sampling Technique):

Definition:

SMOTE stands for Synthetic Minority Over-sampling Technique. It's a method used in the field of machine learning and data mining to address class imbalance in datasets. Class imbalance refers to situations where the number of instances of one class (the minority class) is significantly

lower than the number of instances of the other class (the majority class).

Theory:

The goal of SMOTE is to generate synthetic examples of the minority class to balance the class distribution, thereby improving the performance of machine learning algorithms, especially those that are sensitive to class imbalance.

Here's how SMOTE works:

1. It identifies the minority class examples that are close to one another, often referred to as the "nearest neighbors."
2. Synthetic examples are created by selecting a sample from the minority class and introducing synthetic examples along the line segments joining any or all of its k nearest neighbors.

SMOTE helps in mitigating issues like biased models that tend to favor the majority class due to the class imbalance problem. By synthetically expanding the minority class, it aims to provide a more balanced and representative dataset for training machine learning models.

Usage:

In the code, SMOTE is applied to balance the classes before model training to avoid biased predictions due to imbalanced data.

Evaluation Metrics:

- Accuracy: Measures the overall correctness of the model.
- Precision: Measures the proportion of correctly predicted positive cases out of all predicted positive cases.
- Recall (Sensitivity): Measures the proportion of correctly predicted positive cases out of all actual positive cases.
- F1 Score: The harmonic mean of precision and recall, giving a balance between the two.

Evaluation metrics such as accuracy, precision, recall, and F1-score are computed for both training and test datasets to assess the performance of each model. These metrics provide insights into how well the models are performing in terms of correctly identifying fraudulent transactions while minimizing misclassifications.

5. Results and Evaluation

5.1 Results:

Performance metrics for each algorithm[20%test]:

```
# Assuming you have performance metrics for each algorithm[20%test]
accuracy_scores = [0.957503121537731, 0.9998593109755025, 0.9872940224750717, 0.9994460369660412]
precision_scores = [0.9669215306195706, 0.9997187011023401, 0.9917302573203194, 0.9989283581040722]
recall_scores = [0.9474174770940682,1.0 ,0.9827831806271213 ,0.9999648277438756]
f1_scores = [0.9570701462972668, 0.9998593307661198,0.9872364481110826, 0.999446324207936]
```

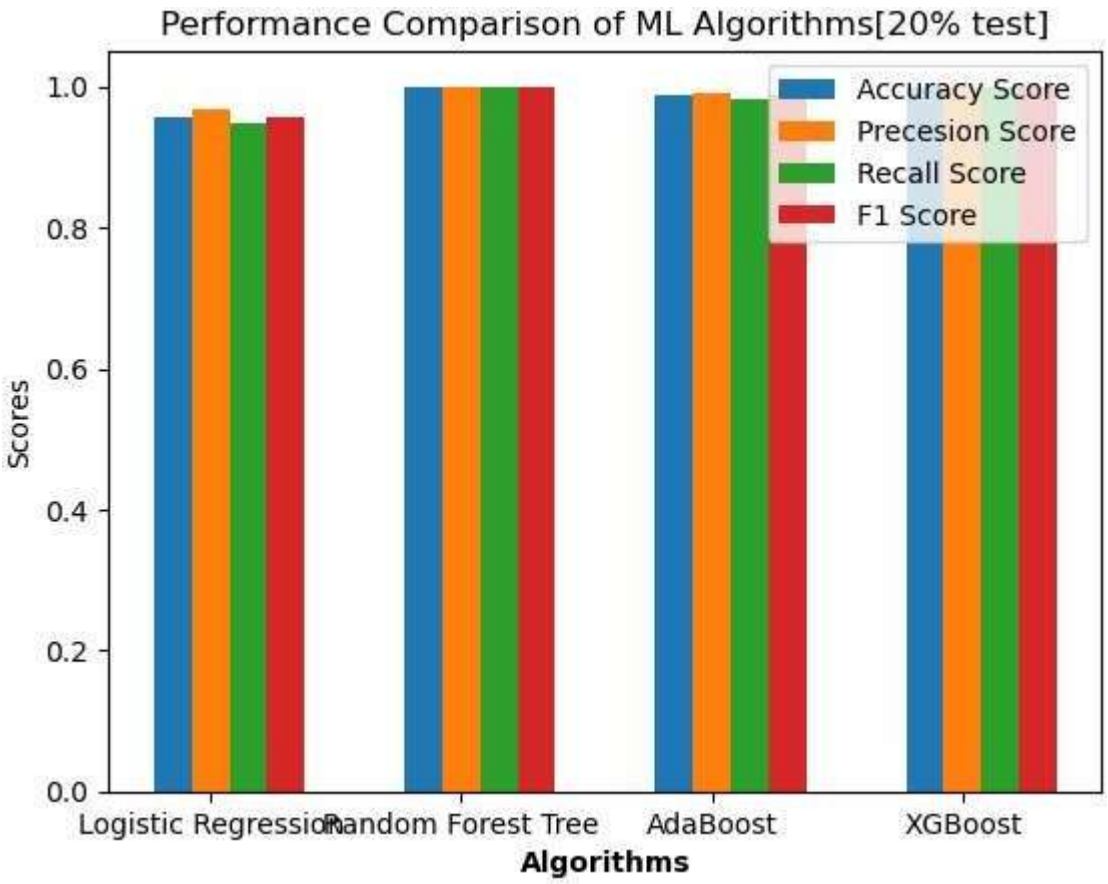


Fig 1. Comparison of accuracy, precesion, recall, f1 scores for Different Classifier
Split the data into training and testing sets (e.g., 80% train, 20% test)

Performance metrics for each algorithm[30%test]

```
# Assuming you have performance metrics for each algorithm[30%test]
accuracy_scores = [0.9719794359542526,0.9998300007620655, 0.9869335068497969,0.9994431059446975]
precision_scores = [0.979409803267756, 0.9997069682939694, 0.9913867559541416,0.9989458766207147]
recall_scores = [0.9642296058339391,0.9999531033835909 ,0.9824020446924754,0.9999413792294886]
f1_scores = [0.9717604244207342,0.9998300206905849,0.9868739510643937, 0.9994433800308192]
```

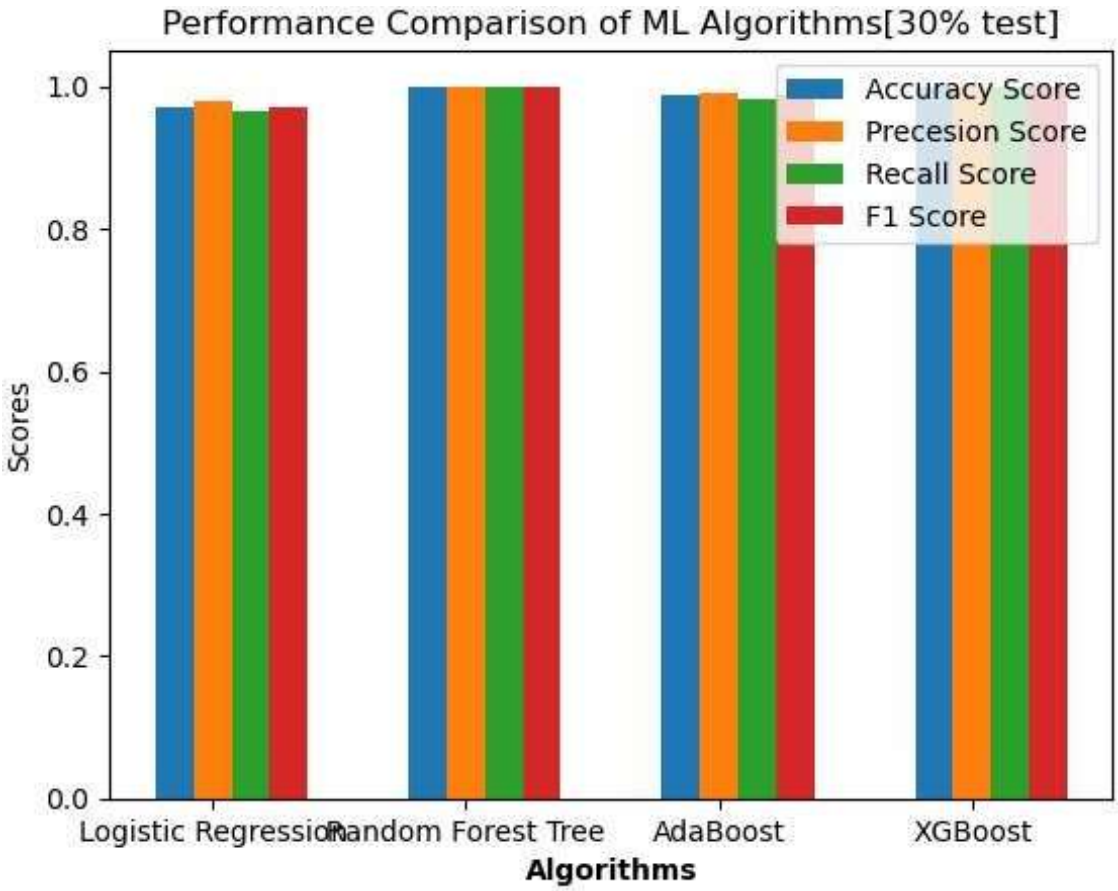


Fig 2. Comparison of accuracy, precesion, recall, f1 scores for Different Classifier
Split the data into training and testing sets (e.g., 70% train, 30% test)

Performance metrics for each algorithm[40%test]:

```
# Assuming you have performance metrics for each algorithm[40%test]
accuracy_scores = [0.971695126883914,0.9998461213794558,0.9868719554015792,0.9994064681779012]
precision_scores = [0.9790327189598514,0.9997186936837941,0.990955843234616,0.9988932125822405]
recall_scores = [0.9640363681128326,0.9999736208079067,0.9827128361148726,0.9999208624237201]
f1_scores = [0.9714766736077267,0.9998461409963909,0.9868171262571411,0.9994067733302866]
```

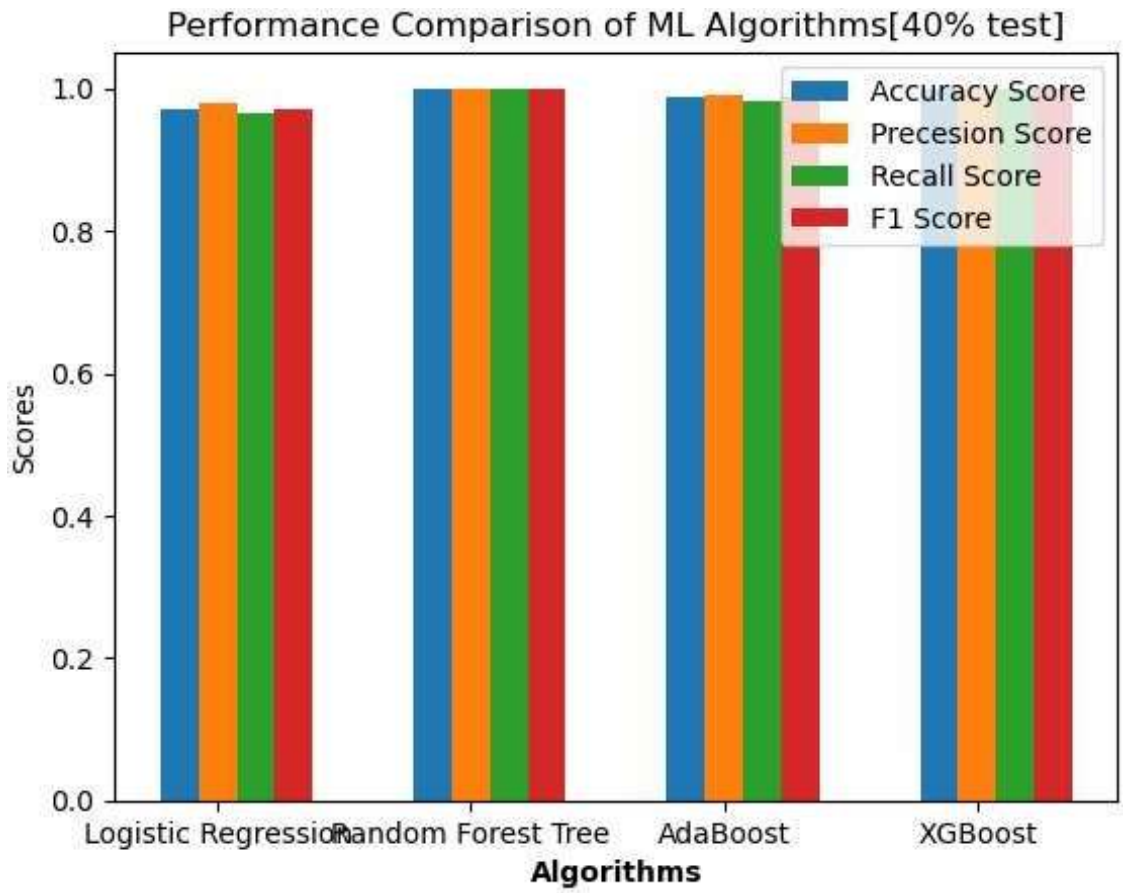


Fig 3. Comparison of accuracy, precesion, recall, f1 scores for Different Classifier
Split the data into training and testing sets (e.g., 60% train, 40% test)

6. Conclusion

In the realm of credit card fraud detection, the project leveraged various robust algorithms—Logistic Regression, Random Forest, AdaBoost, and XGBoost—to assess their performance in identifying fraudulent transactions. Across different test set sizes—20%, 30%, and 40%—these algorithms consistently showcased commendable efficacy.

Notably, Random Forest and XGBoost exhibited exceptional performance across all test set proportions, consistently maintaining accuracy above 99.9%. Moreover, their precision, recall, and F1 scores consistently reflected their robustness in identifying fraudulent activities.

While Logistic Regression and AdaBoost also demonstrated strong performance, their accuracy and other metrics were slightly lower compared to Random Forest and XGBoost. Nonetheless, their results remained commendable, especially considering their computational efficiency.

In conclusion, this project underscores the effectiveness of ensemble methods like Random Forest and XGBoost in credit card fraud detection, offering high accuracy and reliability across varying test set sizes. However, Logistic Regression and AdaBoost also prove to be valuable contenders, showcasing strong potential for fraud detection in real-world scenarios, albeit with a slightly lower but still impressive performance. The choice of algorithm should consider a balance between accuracy, computational resources, and the specific needs of the application.

7. References

- https://www.kaggle.com
- https://www.google.com
- https://chat.openai.com
- https://scikit-learn.org
- https://github.com