



# UNIVERSITÀ DI PISA

Department of Information Engineering - MSc AIDE  
Data Mining and Machine Learning - University of Pisa  
July 2024

## **Credit Card Fraud Detection**

Student:

Ali Asadnia

Supervisors:

Professor Francesco Marcelloni

Professor Alessandro Renda

<b>I. Introduction</b>	<b>3</b>
<b>II. Dataset Description</b>	<b>3</b>
II.1. Overview	3
II.2. Features	4
II.3. Distribution	4
II.4. Purpose and Usage	5
II.5. Privacy Considerations	6
<b>III. Preprocessing</b>	<b>6</b>
<b>IV. Machine Learning Pipeline</b>	<b>9</b>
IV.1. SMOTE	10
IV.2. Feature Selection	12
1. Correlation Coefficient:	12
2. Recursive Feature Elimination:	12
3. Sequential Feature Selection:	12
IV.3. Random Forest	13
Classification Report	14
IV.4. Gradient Boosting	15
Classification Report	17
IV.5. AdaBoost	18
Classification Report	19
IV.6. Naive Bayes	20
Classification Report	21
IV.7. Anomaly Detection	21
IV.7.1. Isolation Forest	22
IV.7.2. Local Outlier Factor(LOF)	24
IV.7.3. DBSCAN	25
<b>V. Deployment</b>	<b>27</b>

# I. Introduction

In today's digital age, credit card fraud is a major concern for financial institutions and customers. The project is to experiment with multiple machine learning techniques to detect fraudulent transactions. Fraudulent transactions follow complex patterns that are challenging to model with traditional programming. Therefore, it is essential to use data mining and machine learning techniques to learn from existing patterns and effectively identify unseen fraudulent transactions.

## II. Dataset Description

### II.1. Overview

The dataset under analysis comprises 284,806 anonymised credit card transactions made by European cardholders in the year 2023. This dataset is specifically designed for the purpose of fraud detection in financial transactions. Each transaction is represented by 28 anonymized features (V1-V28), a transaction amount (Amount), a transaction time(Time), and a binary label indicating whether the transaction is fraudulent (Class).

### II.2. Features

1. **Time:** The elapsed time in seconds between this transaction and the first transaction in the dataset.
2. **V1 to V28:** These anonymized features are likely derived from Principal Component Analysis (PCA), ensuring privacy and security while retaining the essential patterns necessary for fraud detection. The exact nature of these

features is undisclosed due to confidentiality and privacy concerns, but they represent transformed versions of the original transaction attributes.

3. **Amount:** The monetary value of the transaction. This feature can be used to understand the distribution of transaction amounts and their correlation with fraudulent activities.
4. **Class:** The binary label indicating the nature of the transaction:
  - 0: Legitimate transaction
  - 1: Fraudulent transaction

## II.3. Distribution

The dataset is highly imbalanced, with a significant majority of legitimate transactions (Class = 0) and a relatively small number of fraudulent transactions (Class = 1). Specifically, there are 284,315 legitimate transactions and 492 fraudulent transactions. In the project, I outputted the description and the information related to the data points. The information table about the data points is below:

```
df.info()

<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
20  V20         284807 non-null  float64
21  V21         284807 non-null  float64
22  V22         284807 non-null  float64
23  V23         284807 non-null  float64
24  V24         284807 non-null  float64
25  V25         284807 non-null  float64
26  V26         284807 non-null  float64
27  V27         284807 non-null  float64
28  V28         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
```

## II.4. Purpose and Usage

This dataset serves as a benchmark for developing and evaluating machine learning models aimed at detecting fraudulent credit card transactions. Due to the imbalance in class distribution, specialized techniques such as resampling, and appropriate evaluation metrics (precision, recall, F1-score, ROC-AUC) are essential for effectively training and assessing fraud detection models.

## II.5. Privacy Considerations

The features V1 to V28 are anonymized to protect the privacy of the cardholders. These features are likely transformed through PCA, which reduces dimensionality and anonymizes the data while preserving the critical structures and patterns necessary for detecting fraudulent transactions. This approach ensures that sensitive information is not exposed, aligning with privacy and data protection regulations.

## III. Preprocessing

To maximize the pattern recognition of the model for this dataset, a few preprocessing steps were carried out. I used Pandas data structure and other models to tackle this task.

First I dealt with the Null Values to see if the dataset contains any null value. Then, I needed to understand if any duplicate record is present in the dataset here is why:

1. Duplicates can skew the results of the analysis, leading to inaccurate conclusions.
2. Removing duplicates ensures that each transaction is unique.
3. Duplicate data can introduce bias in the machine learning models, especially if the duplicates are not equally distributed across classes.
4. Duplicates can affect the evaluation metrics of the model, making them appear better or worse than they actually are.

The next phase of the preprocessing for this project is to Scale our data. How to scale and which method to choose among all the scaling method ? As we can see in the dataset Time and Amount aren't within the same range, e.g., for the record number 1100 we have: 861 and 10 respectively. So let's scale them and based on

the PCA which is applied before to this dataset, let's put the range of these two column values between -1 and 1(It could be possible by MinMaxScaler).

```
df.iloc[1100]
Time      861.000000
V1        -0.945987
V2         1.232753
V3         0.820313
V4         0.311064
V5         0.981736
V6         0.935188
V7         0.566926
V8        -0.008604
V9        -0.621912
V10        0.380949
V11        0.454998
V12        0.813052
V13        1.348536
V14        0.012636
V15        0.714735
V16       -0.007153
V17       -0.796042
V18        0.403174
V19        0.913516
V20       -0.044043
V21        0.124922
V22        0.284644
V23       -0.197951
V24       -1.326061
V25       -0.161715
V26       -0.382919
V27       -0.488500
V28        0.144470
Amount    10.000000
Class      0.000000
Name: 1107, dtype: float64
```

The first scaling method is Robust Scaler from SkLearn library which is good for data with outliers. It uses the median and interquartile range for scaling and also it is less sensitive to the outliers. I renamed the name of the Time to TimeScaledRobust and Amount to AmountScaledRobust.

```
robust_scaler = RobustScaler()
df['TimeScaled'] = robust_scaler.fit_transform(df['Time'].values.reshape(-1,1))
df['AmountScaled'] = robust_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df.drop(['Time', 'Amount'], axis=1, inplace=True)
```

df

V6	V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Class	TimeScaled	AmountScaled
388	0.239599	0.098698	0.363787	0.090794	...	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0	-0.995290	1.774718
361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0	-0.995290	-0.268530
199	0.791461	0.247676	-1.514654	0.207643	...	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0	-0.995279	4.959811
203	0.237609	0.377436	-1.387024	-0.054952	...	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0	-0.995279	1.411487
221	0.592941	-0.270533	0.817739	0.753074	...	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0	-0.995267	0.667362
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
337	-4.918215	7.305334	1.914428	4.356170	...	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0	1.035258	-0.295230
115	0.024330	0.294869	0.584800	-0.975926	...	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	0	1.035270	0.038798
260	-0.296827	0.708417	0.432454	-0.484782	...	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	0	1.035282	0.638020
708	-0.686180	0.679145	0.392087	-0.399126	...	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	0	1.035282	-0.166875
317	1.577006	-0.414650	0.486180	-0.915427	...	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	0	1.035329	2.711723

Then I applied the MinMaxScaler which Transforms features by scaling each feature to a given range, typically between 0 and 1. But, I changed the range and put it between -1 and 1. It preserves the relationships between data points.

df

V9	...	V25	V26	V27	V28	Amount	Class	TimeScaledRobust	AmountScaledRobust	scaled_timeMinMax	scaled_amountMinMax
0.363787	...	0.128539	-0.189115	0.133558	-0.021053	149.62	0	-0.995290	1.774718	-1.000000	-0.988352
-0.255425	...	0.167170	0.125895	-0.008983	0.014724	2.69	0	-0.995290	-0.268530	-1.000000	-0.999791
-1.514654	...	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0	-0.995279	4.959811	-0.999988	-0.970522
-1.387024	...	0.647376	-0.221929	0.062723	0.061458	123.50	0	-0.995279	1.411487	-0.999988	-0.990386
0.817739	...	-0.206010	0.502292	0.219422	0.215153	69.99	0	-0.995267	0.667362	-0.999977	-0.994551
...	...	...	...	...	...	...	...	...	...	...	...
1.914428	...	1.436807	0.250034	0.943651	0.823731	0.77	0	1.035258	-0.295230	0.999931	-0.999940
0.584800	...	-0.606624	-0.395255	0.068472	-0.053527	24.79	0	1.035270	0.038798	0.999942	-0.998070
0.432454	...	0.265745	-0.087371	0.004455	-0.026561	67.88	0	1.035282	0.638020	0.999954	-0.994716
0.392087	...	-0.569159	0.546668	0.108821	0.104533	10.00	0	1.035282	-0.166875	0.999954	-0.999222
0.486180	...	-0.473649	-0.818267	-0.002415	0.013649	217.00	0	1.035329	2.711723	1.000000	-0.983107

In order to understand which of the above scaling is better than the other, we have to complete the training phase and evaluation of the model based on those four columns. But, I trained these two scaled columns on a Machine Learning model



called Random Forest with RF classifier to evaluate the classification report and ROC-AUC score, which you will see below:

```
RobustScaler Classification Report:
              precision    recall  f1-score   support

         0           1.00       1.00       1.00      84984
         1           0.22       0.10       0.13        134

 accuracy          1.00          1.00          1.00      85118
 macro avg          0.61          0.55          0.57      85118
weighted avg          1.00          1.00          1.00      85118

RobustScaler ROC-AUC Score: 0.64358146081229
MinMaxScaler Classification Report:
              precision    recall  f1-score   support

         0           1.00       1.00       1.00      84984
         1           0.22       0.10       0.13        134

 accuracy          1.00          1.00          1.00      85118
 macro avg          0.61          0.55          0.57      85118
weighted avg          1.00          1.00          1.00      85118

MinMaxScaler ROC-AUC Score: 0.6398520494112324
```

Based on the outlier robustness and the slightly better ROC-AUC score, I chose **RobustScaler** for my fraud detection project. This choice ensures that the scaling process is less sensitive to outliers, which can be important in financial datasets where extreme values may occur.

## IV. Machine Learning Pipeline

In the machine learning part, first, I want to mention that I split the data then I applied the SMOTE. There is a reason why I think it should be explained. It is because the test set which will be cleared after the splitting part should maintain the integrity and also realism of the test set. In the splitting part I set the `test_size = 30` percent of the data(which is common among medium sized datasets) and I also used `random_state = 42` which guarantees that each run of the code may produce different train-test splits, as the random number generator will use a different seed

each time. I also used the stratify parameter which ensures that each split has the same proportion of classes as the original dataset. This helps in maintaining the representativeness of both training and test sets.

## IV.1. SMOTE

SMOTE which stands for Synthetic Minority Over-sampling Technique is a good example of oversampling in which the minority class is over-sampled by creating “synthetic” examples rather than by over-sampling with replacement. Steps of this method which is a classification for the dataset is as below:

1. Identify Nearest Neighbors
2. Generate Synthetic Samples
  - a. Calculate the difference(feature vector of the sample under consideration and its nearest neighbor)
  - b. Multiply by a random number(The number is between 0 and 1)
  - c. Create synthetic sample(This random point selection along the line segment between the sample and its neighbor creates diversity in the synthetic samples)

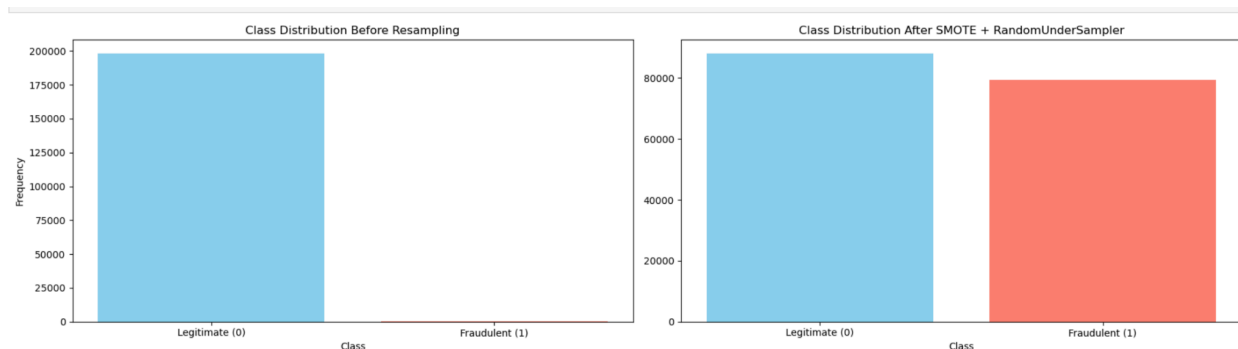
In my project, I used the combining of SMOKE with a RandomUnderSampler which means oversample the minority class to a specified ratio, followed by RandomUnderSampler to undersample the majority class to another specified ratio, because I think If the imbalance is moderate, using only SMOTE might be sufficient and simpler. Since the imbalance is very large, the combination of two methods could be more beneficial. Also we should consider the computational resources available. A very large dataset from SMOTE only can be resource-intensive.

```

oversample = SMOTE(sampling_strategy=0.4)
undersample = RandomUnderSampler(sampling_strategy=0.9)
oversampled_train_features, oversampled_train_labels = oversample.fit_resample(train_features, train_labels)
rebalanced_features, rebalanced_labels = undersample.fit_resample(oversampled_train_features, oversampled_train_labels)

```

In the project I define two variables called oversample and undersample, the first one which has a `sampling_strategy = 0.5` applied SMOTE which means to increase the number of samples in the minority class by generating synthetic examples, thus balancing the class distribution. The parameter means that after applying SMOTE, the minority class will have 50% of the number of samples of the majority class. For instance, if the majority class has 1000 samples, the minority class will be increased to 500 samples. The RandomUnderSampler's objective is to reduce the number of samples in the majority class by randomly removing examples, further balancing the class distribution. The `sampling_strategy = 0.9` means that after applying RandomUnderSampler, the majority class will be reduced to 90% of the number of samples of the minority class. For example, if the minority class has 500 samples after SMOTE, the majority class will be reduced to 450 samples. After these steps we have to apply this resampling to our data. Let's see the figure before SMOTE class labels and after SMOTE + RUS class labels.



After the resampling process, now I have to choose which feature between those 30 columns has more importance and could possibly create better results. So the next step is feature selection. I applied two feature selections for this dataset.

## IV.2. Feature Selection

### 1. Correlation Coefficient:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

$r$  = correlation coefficient  
 $x_i$  = values of the x-variable in a sample  
 $\bar{x}$  = mean of the values of the x-variable  
 $y_i$  = values of the y-variable in a sample  
 $\bar{y}$  = mean of the values of the y-variable

- a.
- b. From the above formula and the correlation graph in the code we figured that v3, v4, v10, v11, v12, v14, v16, v17 are the features that have a relationship(positive or negative) with the Class that we want to predict.

### 2. Recursive Feature Elimination:

- a. Recursive Feature Elimination (RFE) is a powerful and widely used feature selection method that aims to select the most important features of a dataset to enhance model performance. RFE works by recursively removing less important features and building models on increasingly smaller sets of features until the optimal subset is identified.
- b. The result from RFE on the logistic regression algorithm for our classification problem is: v14, v4, v26, v12, v24, v10, v22, v11, v6, and v23, respectively.

### 3. Sequential Feature Selection:

- a. *Forward-SFS* is a greedy procedure that iteratively finds the best new feature to add to the set of selected features. Concretely, we initially

start with zero features and find the one feature that **maximizes a cross-validated score** when an estimator is trained on this single feature. Once that first feature is selected, we repeat the procedure by adding a new feature to the set of selected features. The procedure stops when the desired number of selected features is reached, as determined by the `n_features_to_select` parameter.

- b. I implemented it with the K-NearestNeighbor algorithm, the result is: `v3, v4, v7, v14`(the output is in the shape of a boolean variable).

Between all the three methods that I applied two features are in common which are `v4, v14`.

Since resampling with SMOTE + RandomUnderSampling are parts of the Machine Learning pipeline I mentioned them here, because it makes more sense at the time.

### IV.3. Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. This approach helps improve the model's accuracy and robustness, especially for imbalanced datasets like this, where fraudulent transactions are significantly fewer than legitimate ones.

I defined a pipeline incorporating the Random Forest classifier with specific hyperparameters:

- `N_estimators = 100`: The number of trees in the forest.
- `Class_weight = "balanced"`: Adjusts weights inversely proportional to class frequencies in the input data.

- `Oob_score = True`: Uses out-of-bag samples to estimate the generalization accuracy.
- `Random_state = 42`: Ensures reproducibility.

The model was evaluated using 10-fold Stratified Cross-Validation to ensure that each fold had a similar distribution of classes, providing a reliable measure of model performance.

The model was trained on the rebalanced dataset and then tested on the unseen test dataset to assess its performance.

### Classification Report

The performance results of the random forest is the below picture:

Metrics	Results
Accuracy	0.999448
Precision	0.886179
Recall	0.767606
F1_score	0.822642

1. **Accuracy**: The proportion of correctly predicted instances out of the total instances. Our model achieved an accuracy of **0.999448**, indicating a very high overall correctness.
2. **Precision**: The proportion of true positive predictions among the total predicted positives. The model's precision was **0.886179**, reflecting its effectiveness in identifying fraudulent transactions.
3. **Recall**: The proportion of true positive predictions among the actual positives. The recall value was **0.767606**, indicating that the model

successfully identified a significant portion of the actual fraudulent transactions.

4. **F1-Score:** The harmonic mean of precision and recall, providing a balance between the two metrics. The F1-score was **0.822642**, demonstrating a strong overall performance in detecting fraud.

The confusion matrix output result would be

- **True Positives (110):** Fraudulent transactions correctly identified.
- **False Positives (14):** Legitimate transactions incorrectly identified as fraudulent.
- **True Negatives (84962):** Legitimate transactions correctly identified.
- **False Negatives (32):** Fraudulent transactions incorrectly identified as legitimate.

#### IV.4. Gradient Boosting

Gradient Boosting is a powerful ensemble learning technique widely used for classification and regression tasks, known for its high predictive accuracy. In the context of fraud detection, Gradient Boosting can effectively identify complex patterns and interactions in the data, making it a robust choice for this application. Gradient Boosting builds an additive model by sequentially training a series of weak learners, typically decision trees. Each new tree corrects the errors made by the previous ones, focusing on the instances that are harder to predict. The method combines these weak learners to form a strong predictive model. For finding the best values for hyperparameters `learning_rate` and `max_depth` for this algorithm I applied `RandomizedSearchCV` and the result

```
# Define the pipeline
```

```
pipe = Pipeline([
```

```

('classification', GradientBoostingClassifier(random_state=42))
])
param_grid = {
    'classification__learning_rate': [0.01, 0.1, 0.2],
    'classification__max_depth': [3],
}
# Define the cross-validation strategy
skf = StratifiedKFold(n_splits=3)

# Perform RandomizedSearchCV to find the best parameters
random_search = RandomizedSearchCV(pipe, param_grid, cv=skf,
    scoring='accuracy', n_iter=1, n_jobs=-1, random_state=42)
random_search.fit(rebalanced_features, rebalanced_labels)

# Print the best parameters and the best score
print("Best parameters found:", random_search.best_params_)
print("Best cross-validation accuracy:", random_search.best_score_)

```

The output of the code above after running in Google Colab is 0.01 for learning\_rate and the best cross-validation accuracy is 0.95057 which is high. Randomized Search Cross Validation was so time consuming and computationally expensive, that is why the number of iterations is 1 and I also decrease the number of cross validation to 3. Now we can train our gradient boosting model and see the result. But I played with max\_depth a little bit to see the difference in the result. And I chose 3 as my maximum depth for the algorithm.



```

# Define the Gradient Boosting Classifier with specified parameters
gb_model = GradientBoostingClassifier(n_estimators = 100, learning_rate =
0.01, max_depth = 5, random_state=42)
# Train the model
gb_model.fit(rebalanced_features, rebalanced_labels)
# Make predictions on the test set
y_pred = gb_model.predict(test_features)
y_pred_proba = gb_model.predict_proba(test_features)[:, 1]
# Evaluate the model
print("Classification Report:\n", classification_report(test_labels, y_pred))
roc_auc = roc_auc_score(test_labels, y_pred_proba)
print("ROC-AUC Score:", roc_auc)

```

Let's explain the result we obtained from the above algorithm

## Classification Report

1. **ROC-AUC Score: 0.9666:** This is an excellent score, indicating that the model is very good at distinguishing between the positive and negative classes.
2. **Precision, Recall, and F1-score for Class 0 (Legitimate Transactions):**
  - **Precision: 1.00:** Indicates that almost all transactions predicted as legitimate are actually legitimate.
  - **Recall: 0.99:** Indicates that almost all actual legitimate transactions are correctly identified.
  - **F1-score: 0.99:** A harmonic mean of precision and recall, indicating excellent performance for class 0.
3. **Precision, Recall, and F1-score for Class 1 (Fraudulent Transactions):**

- **Precision: 0.13:** Indicates a high number of false positives (legitimate transactions predicted as fraudulent).
  - **Recall: 0.87:** Indicates that most fraudulent transactions are correctly identified.
  - **F1-score: 0.22:** Indicates that there is room for improvement in balancing precision and recall for fraudulent transactions.
4. **Accuracy: 0.99:** High accuracy, but it is important to note that accuracy can be misleading in imbalanced datasets like fraud detection.
  5. The averages Macro Average and Weighted Average provide a balanced view of the model's performance across both classes, considering the class imbalance.

#### Confusion Matrix Breakdown

- **True Positives (TP):** 124
- **True Negatives (TN):** 82958
- **False Positives (FP):** 2018
- **False Negatives (FN):** 18

The high recall (0.87) for fraudulent transactions is a positive sign, saying that the model is effectively identifying most fraudulent transactions. However, the low precision (0.13) for fraudulent transactions suggests that the model is generating many false positives, which might be problematic for prediction on unseen data.

## IV.5. AdaBoost

Adaptive Boosting (AdaBoost) is a powerful ensemble learning technique that combines multiple weak classifiers to create a strong classifier. The core idea

behind AdaBoost is to focus on the instances that are difficult to classify by sequentially adjusting the weights of the training data. I need to break-down the code to explain what is actually going to happen to our data. The AdaBoost model was initialized with the following parameters:

- **Base Estimator:** SGDClassifier with a log\_loss function

The AdaBoost model was initialized with the following parameters:

- **Base Estimator:** SGDClassifier with a log\_loss function. This base estimator is suitable for classification tasks and helps in calculating the logarithmic loss.
- **Algorithm:** SAMME (Stagewise Additive Modeling using a Multiclass Exponential loss function), which allows using classifiers that do not necessarily produce probabilistic outputs.
- **Number of Estimators:** 100, meaning that the AdaBoost algorithm iterates 100 times to create a strong classifier.
- **Random State:** 42, ensuring reproducibility of the results.

The model was trained on the rebalanced training set created using SMOTE and RandomUnderSampler to address the class imbalance issue.

## Classification Report

**Precision:** 1.00 for the legitimate class and 0.03 for the fraudulent class. This indicates that while the model is highly precise for legitimate transactions, it struggles with correctly identifying fraudulent transactions.

- **Recall:** 0.95 for the legitimate class and 0.90 for the fraudulent class. The high recall for fraudulent transactions suggests that the model is effective at identifying most fraudulent transactions, but with a trade-off in precision.
- **F1-score:** 0.98 for the legitimate class and 0.06 for the fraudulent class. The low F1-score for fraudulent transactions reflects the model's difficulty in balancing precision and recall for this class.
- **Support:** The number of occurrences of each class in the dataset, with 84,976 legitimate and 142 fraudulent transactions.

**ROC-AUC Score:** 0.9566433919370108, which indicates that the model has a good ability to distinguish between the legitimate and fraudulent classes.

#### **Confusion Matrix Report:**

**True Negatives (TN):** 80943 legitimate transactions correctly identified.

**True Positives (TP):** 128 fraudulent transactions correctly identified.

**False Positives (FP):** 4033 legitimate transactions were incorrectly identified as fraudulent.

**False Negatives (FN):** 14 fraudulent transactions incorrectly identified as legitimate.

## **IV.6. Naive Bayes**

Naive Bayes is a probabilistic machine learning algorithm used for classification tasks. It is based on Bayes' Theorem, which describes the probability of an event based on prior knowledge of conditions related to the event. The Naive Bayes classifier assumes that the features are independent of each other given the class label, which is a strong and often unrealistic assumption in real-world scenarios. Despite this "naive" assumption, Naive Bayes classifiers have shown excellent performance in many complex real-world applications.

## Classification Report

**Precision:** 1.00 for the legitimate class and 0.05 for the fraudulent class. This indicates that the model has high precision for legitimate transactions but struggles with identifying fraudulent transactions correctly.

- **Recall:** 0.97 for the legitimate class and 0.82 for the fraudulent class. The high recall for fraudulent transactions suggests the model is effective at identifying most fraudulent transactions, but with a trade-off in precision.
- **F1-score:** 0.99 for the legitimate class and 0.10 for the fraudulent class. The low F1-score for fraudulent transactions reflects the model's difficulty in balancing precision and recall for this class.
- **Support:** The number of occurrences of each class in the dataset, with 84,976 legitimate and 142 fraudulent transactions.

**ROC-AUC Score:** 0.9541561942261743, indicating a strong ability of the model to distinguish between legitimate and fraudulent transactions.

**True Negatives (TN):** 82825

**True Positives (TP):** 117

**False Positives (FP):** 2151

**False Negatives (FN):** 25

The best algorithm between the above algorithms is Random Forest which gave promising results on our dataset.

## IV.7. Anomaly Detection

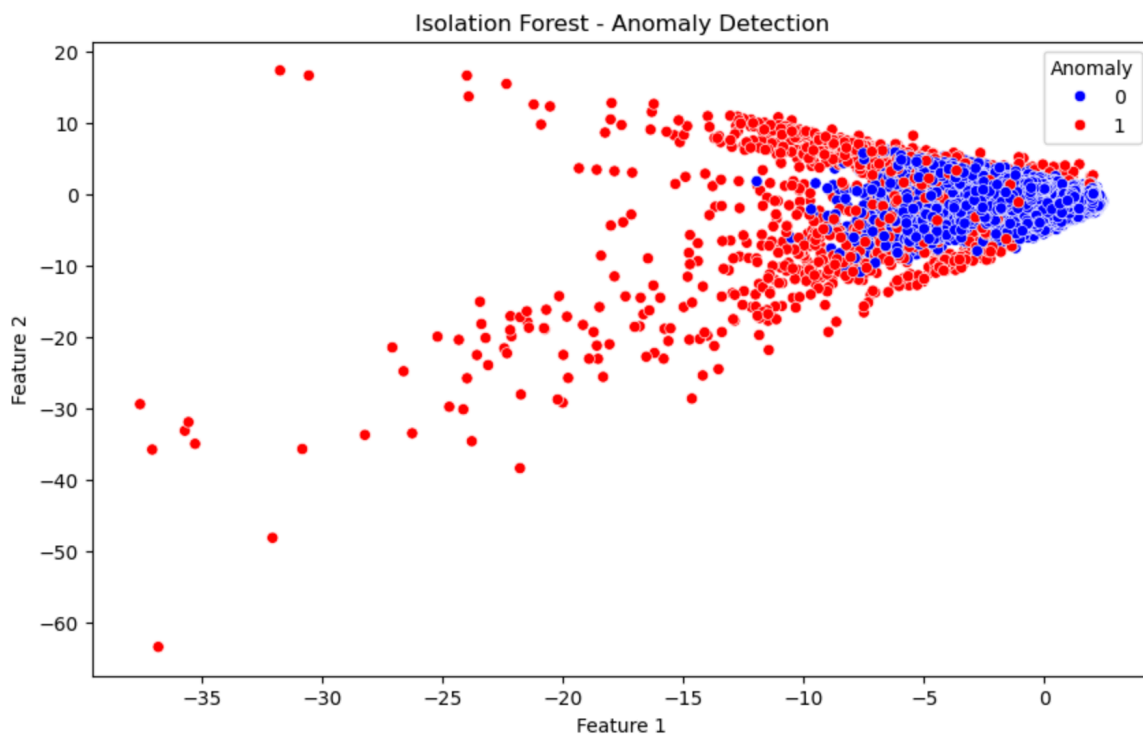
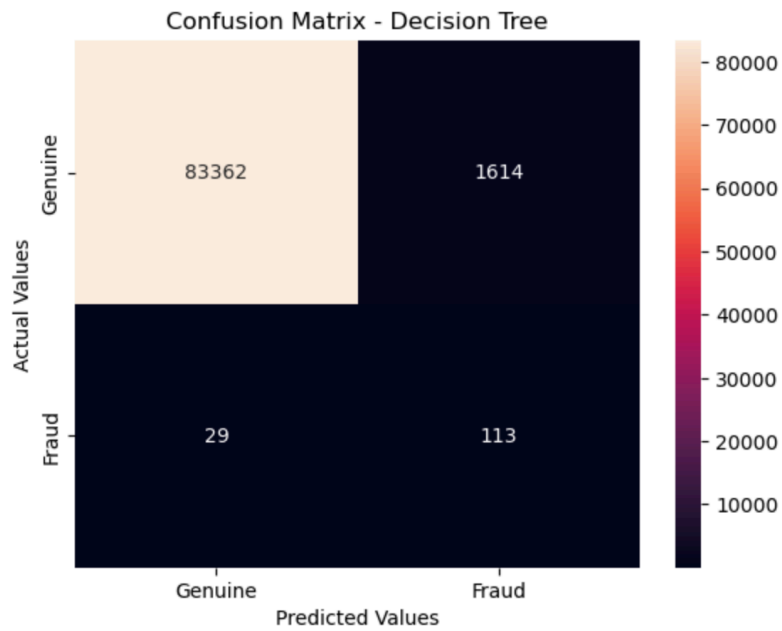
Anomalies are data points that do not conform to expected patterns in the dataset. Anomaly detection is used to identify rare events, unusual patterns, or outliers that do not fit the general data distribution. Since the initial dataset was heavily imbalanced, fraudulent transactions could potentially be considered outliers or anomalies in comparison to the rest of the data, therefore, treating the problem as an anomaly detection problem should also bring some interesting results. I used Isolation Forest and Local Outlier Factor and DBSCAN(It is a clustering model) for this dataset which will see the definition of each in a few lines. Isolation Forest is an ensemble learning algorithm specifically designed for anomaly detection. It works by isolating observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

#### IV.7.1. Isolation Forest

For Isolation Forest I have to calculate the contamination factor. A higher contamination factor value increases the sensitivity (more points are classified as outliers), while a lower value increases specificity (fewer points are classified as outliers).

```
#Model Creation
isof = IsolationForest(n_estimators=80,
                       max_samples=len(train_features),
                       contamination=float(0.02),
                       random_state=42,
                       verbose=2)
```

For contamination 0.02, the normal class had 1614 misclassified “Normal” transactions and 29 misclassified “Fraud” transactions. When increasing the contamination value, we see a better assimilation of the Fraud class, but an increase of False Positives from the Normal class by thousands.



Visualization Description

The scatter plot visualizes the distribution of data points across two selected features from the test set. The x-axis (Feature 1) and y-axis (Feature 2) represent the two chosen features for this analysis. Data points are colored based on the predictions made by the Isolation Forest model:

- **Blue points (0):** Represent normal transactions.
- **Red points (1):** Represent anomalous transactions, which are potential fraudulent activities.

#### IV.7.2. Local Outlier Factor(LOF)

The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method. It is designed to identify anomalous instances by comparing the local density of a point with that of its neighbors. The key concept behind LOF is that normal data points have a density similar to their neighbors, whereas outliers have a substantially lower density.

```

Classification Report:
              precision    recall  f1-score   support

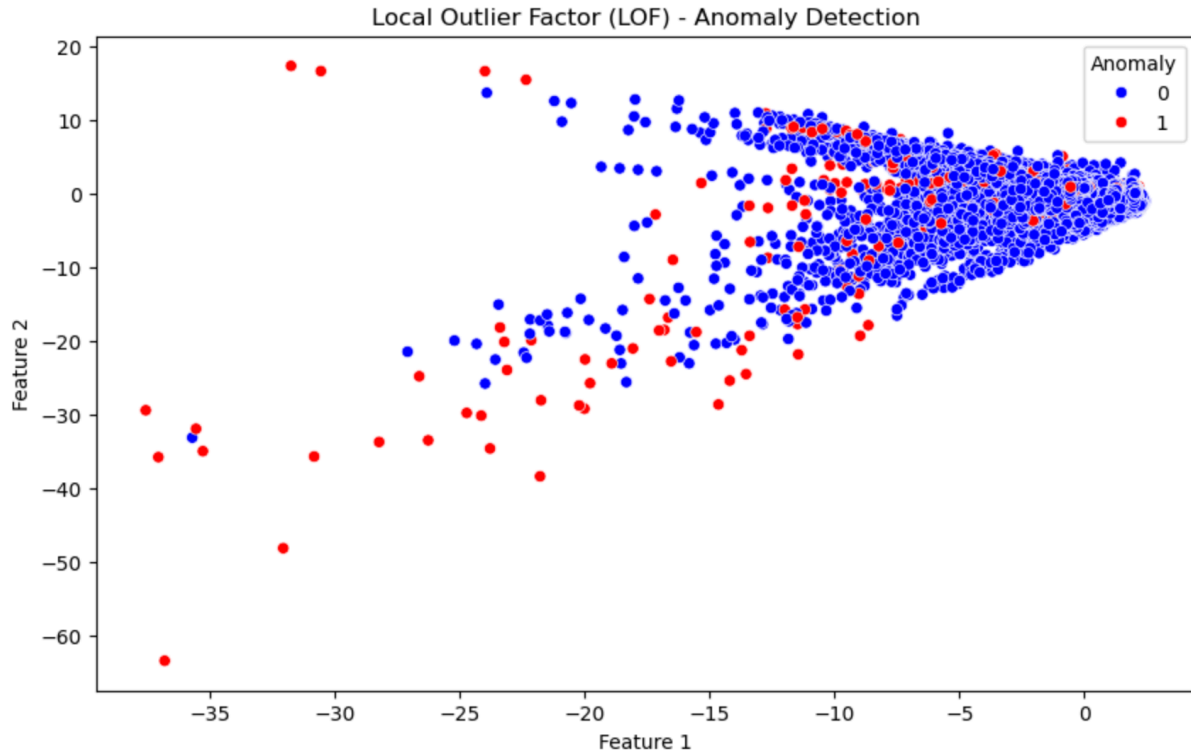
     0               1.00      0.95      0.97      84976
     1               0.00      0.14      0.01         142

 accuracy              0.95      85118
 macro avg              0.50      0.54      0.49      85118
 weighted avg           1.00      0.95      0.97      85118

Accuracy Score: 0.9474376747573956

```





The Local Outlier Factor (LOF) algorithm was applied to the dataset to detect anomalies. The classification report shows that while the model achieves perfect precision for legitimate transactions (Class 0), it has very low precision and recall for fraudulent transactions (Class 1), indicating poor performance in fraud detection. The overall accuracy of 0.95 is misleading due to the dataset's imbalance, heavily favoring legitimate transactions. The scatter plot visualizes the detected anomalies, with red points representing identified outliers. The plot shows that while the model correctly identifies some outliers, it also fails to detect many fraudulent transactions.

#### IV.7.3. DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised clustering algorithm that is particularly effective for identifying clusters of varying shapes and sizes within a dataset, along with detecting noise (outliers). The algorithm groups together points that are closely packed together, marking points that lie alone in low-density regions as outliers. The model was fitted on the scaled training features to identify clusters and outliers. The first result with epsilon 0.5 and min\_samples 5 wasn't intriguing at all. That's why I started to search for the best epsilon and min\_samples with Grid search and Silhouette Score. Silhouette Score is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It ranges from -1 to 1, where higher values indicate better-defined clusters. The best Silhouette Score that I got for the dataset is 0.6331 and the best epsilon was 0.6 and min\_samples is 25 but the result is not good at all because it detects 130334 points as anomalies. DBSCAN to the fraud detection dataset obtained limited success in accurately identifying fraudulent transactions. The result is below:

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	0.06	0.12	84976	
1	0.00	1.00	0.00	142	
accuracy			0.06	85118	
macro avg	0.50	0.53	0.06	85118	
weighted avg	1.00	0.06	0.11	85118	

**Precision:** Precision for class 0 (non-fraudulent transactions) is very high (1.00), indicating that most of the points identified as non-fraudulent are indeed

non-fraudulent. However, the precision for class 1 (fraudulent transactions) is 0.00, suggesting that the algorithm did not correctly identify any fraudulent transactions.

**Recall:** Recall for class 0 is 0.01, meaning that only 1% of the actual non-fraudulent transactions were correctly identified. The recall for class 1 is 1.00, indicating that all actual fraudulent transactions were identified as anomalies.

**F1-Score:** The F1-score for class 0 is 0.02, and for class 1, it is 0.00. The low F1-scores suggest a poor balance between precision and recall.

**Accuracy:** The overall accuracy is 0.01, reflecting the fact that the model correctly classified only a small fraction of the transactions.

## V. Deployment

For the deployment of the fraud detection model, I utilized the Random Forest algorithm, which demonstrated the best performance during the model selection process. The model was configured with the following parameters: `n_estimators=100`, `verbose=2`, `n_jobs=2`, `oob_score=True`, `class_weight="balanced"`, and `random_state=42`.

To provide a user-friendly interface for interacting with the model, I developed a web application using the Streamlit library in Python. The application offers users two distinct options for inputting their transaction data:

1. **Uploading a CSV File:** Users can choose to upload a CSV file containing their transaction data. This method is efficient for batch processing and allows users to quickly analyze multiple transactions at once. The application processes the uploaded file, performs predictions using the

trained Random Forest model, and immediately displays the results, indicating which transactions are classified as fraudulent.

- 2. Entering Data Manually:** For users who prefer or need to input data manually, the application provides an intuitive interface where they can enter values for each feature of their transactions. Users specify the number of transactions they want to analyze and input the corresponding feature values. The application then predicts the fraud status of each transaction based on the input data.


These functionalities ensure that the application is flexible and accessible, catering to different user needs. Below are screenshots showing the two options available in the application:

## Credit Card Fraud Detection

Choose how to input your data:

- ☒ Upload CSV file  
☐ Enter data manually

Choose a CSV file for prediction

 Drag and drop file here  
Limit 200MB per file • CSV

Browse files

Choose how to input your data:

☐ Upload CSV file

☒ Enter data manually

Enter the number of transactions:

1

- +

Please enter the values for each feature for 1 transactions (up to 12 decimal points):

Transaction 1

Feature 1 (Transaction 1)

0.000000000000

- +

Feature 2 (Transaction 1)

0.000000000000

- +

Feature 3 (Transaction 1)

0.000000000000

- +

Feature 4 (Transaction 1)

0.000000000000

- +

Feature 5 (Transaction 1)

0.000000000000

- +

Feature 6 (Transaction 1)

0.000000000000

- +