VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**LAM NGOC HAI – 518H0347**

**PHAN AN DUY - 518H0616**

**CREDIT CARD FRAUD DETECTION WITH IMBALANCED DATA**

**INFORMATION TECHNOLOGY PROJECT 2**

**COMPUTER SCIENCE**

**HO CHI MINH , YEAR 2025**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**LAM NGOC HAI – 518H0347**

**PHAN AN DUY - 518H0616**

**CREDIT CARD FRAUD DETECTION WITH IMBALANCED DATA**

**INFORMATION TECHNOLOGY PROJECT 2**

**COMPUTER SCIENCE**

Advised by

**Mr., Trinh Hung Cuong**

**HO CHI MINH , YEAR 2025**

**ACKNOWLEDGMENT**

I would like to express my heartfelt gratitude to Mr. Trinh Hung Cuong for his invaluable guidance and support throughout this project. His insights and encouragement have been instrumental in shaping this work, and I am deeply thankful for his contributions..

Finally, I would want to express my gratitude to all the other people and organizations that have supported the project in various ways even though they aren't listed here.

This project would not have been possible without the cooperation, hard work, and dedication of those involved. I sincerely hope that we can continue to collaborate and accomplish similar feats in the future.

*Ho Chi Minh City, day 08 month 02 year 2025*

*Author*

*(Signature and full name)*

Hai, Duy

LAM NGOC HAI

PHAN AN DUY

**DECLARATION OF AUTHORSHIP**

I hereby declare that this project report is my own work and that all sources of information and data have been duly acknowledged. I affirm that I have not received unauthorized assistance in preparing this report and that all content and ideas presented are original, except where explicitly stated otherwise.

By signing this declaration, I take full responsibility for the authenticity and integrity of this work.

**I will take full responsibility for any fraud detected in my thesis**. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

*Ho Chi Minh City, day 08 month 02 year 2025*

*Author*

*Hai , Duy*

*Lam Ngoc Hai*

*Phan An Duy*

*(signature and full name)*

**CREDIT CARD FRAUD DETECTION WITH IMBALANCED DATA**

**Introduction**

Credit card fraud detection with imbalanced data refers to challenge of identifying fraudulent transaction when the dataset contains a significantly higher number of legitimate transactions compared to fraudulent ones. This imbalance make it difficult for machine learning models to accurately detect fraud, as they tent to be biased towards the majority class( Legitimate transactions)

**Why we need to balanced data?**

Balancing data is crucial in credit card fraud detection for several reasons:

1. **Improved Model Performance:** Machine learning models can struggle with imbalanced data because they tend to predict the majority class ( Legitimate transactions) more often. Balancing the data helps the model learn to identify both classes more accurately.
2. **Reduced Bias:** When the data is imbalanced, the model can become biased towards the majority class. Balancing the data mitigates this bias, ensuring that the model pays attention to the minority class ( Fraudulent transactions).
3. **Enhanced Detection of Fraud:** Since fraudulent transactions are rare, they can be easily overlooked by the model. Balancing the data increase the model’s ability to detect these rare events, improving the overall effectiveness of fraud detection.
4. **Better Evaluation Metrics:** Standard evaluation metrics like accuracy can be misleading with imbalanced data. Balancing the data allows for more meaningful metrics like Precision, Recall, and F-1 Score, which provide a clearer picture of the model’s performance.

**Challenges of Imbalanced Data**

Imbalanced data can lead to several issues when training machine learning models:

1. Bias towards Majority Class: Models trained on imbalanced data tend to be biased towards the majority class, resulting in poor detection of the minority class.
2. Metric Misleading: Traditional evaluation metrics like accuracy can be misleading. A model that classifies all transactions as legitimate may have high accuracy but will fail to detect fraud.

**Techniques to Address Imbalanced Data**

Various techniques are employed to handle imbalanced datasets:

1. **Resampling Methods:**

* **Oversampling:** This involves increasing the number of instances in the minority class. Techniques like Synthetic Minority Over-sampling Technique (SMOTE) generate synthetic examples.
* **Undersampling:** This reduces the number of instances in the majority class. Random undersampling selects a subset of the majority class to balance the dataset.

1. **Cost-Sensitive Learning:**

* This approach assigns a higher cost to misclassifying the minority class (fraudulent transactions), encouraging the model to focus more on detecting fraud.

1. **Ensemble Models:**

* Combining multiple models, such as Random Forest or Gradient Boosting, can improve detection performance by leveraging the strengths of different algorithms.

1. **Anomaly Detection:**

* Since fraud is rare, treating it as an anomaly detection problem can be effective. Techniques like Isolation Forest and One-Class SVM can help identify outliers that may represent fraudulent transactions.

1. **Feature Engineering:**

* Creating new features that capture patterns of fraudulent behavior can improve model performance. For example, features like transaction frequency, location patterns, and purchase amounts can provide valuable insights.

**Evaluation Metrics**

When dealing with imbalanced data, it's crucial to use appropriate evaluation metrics:

1. **Precision:** The ratio of true positives to the sum of true positives and false positives.
2. **Recall:** The ratio of true positives to the sum of true positives and false negatives.
3. **F1-Score:** The harmonic mean of precision and recall.
4. **Area Under the ROC Curve (AUC-ROC):** Measures the model's ability to distinguish between classes.

**Conclusion**

Credit card fraud detection with imbalanced data is a complex but critical task. By employing techniques like resampling, cost-sensitive learning, ensemble models, anomaly detection, and careful feature engineering, it's possible to build robust models that effectively identify fraudulent transactions. Appropriate evaluation metrics are essential to ensure the model's performance is accurately assessed.

# **CONTENTS**

[**CONTENTS** 7](#_Toc190638533)

[**LIST OF FIGURES** 8](#_Toc190638534)

[**Chapter 1: Read And Understanding Data** 1](#_Toc190638535)

[**1** **Import Libraries:** 1](#_Toc190638536)

[**2** **Understanding our data:** 2](#_Toc190638537)

[**3** **Check The Data:** 3](#_Toc190638538)

[**4** **Glimpse the data:** 4](#_Toc190638539)

[**5** **Result Meaning** 6](#_Toc190638540)

[**6** **Data Preprocessing** 7](#_Toc190638541)

[**Chapter 2: Undersampling The Dataset** 8](#_Toc190638542)

[**1** **Train-Test-Split: Working On The Dataset** 15](#_Toc190638543)

[**2** **Understanding The Sampling Method** 17](#_Toc190638544)

[**Chapter 3: Using Model To Balance Data** 18](#_Toc190638545)

[**1** **Using Logistic Regression** 18](#_Toc190638546)

[**2** **Using Decision Tree Classifier** 21](#_Toc190638547)

[**3** **Using The KNN Classifier** 22](#_Toc190638548)

[**References** 33](#_Toc190638549)

[1 https://blog.tomorrowmarketers.org/thuat-toan-phan-loai-classification-machine-learning/ 33](#_Toc190638550)

[2 https://www.kaggle.com/code/chanchal24/credit-card-fraud-detection 33](#_Toc190638551)

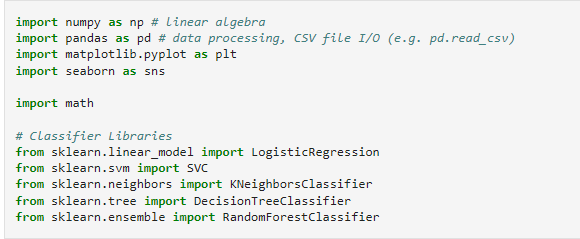
[3 https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive-models 33](#_Toc190638552)

# 

# **LIST OF FIGURES**

# **Chapter 1: Read And Understanding Data**

1. **Import Libraries:**



**Importing libraries in Python is essential for several reasons**:

1. **Code Reusability:** Libraries contain pre-written code that you can use in your projects. Instead of writing code from scratch, you can import libraries that offer the functionalities you need, saving time and effort.
2. **Access to Advanced Features:** Libraries provide access to advanced features and tools that would be difficult to implement on your own. For example, libraries like NumPy and pandas offer powerful data manipulation capabilities, while libraries like TensorFlow and PyTorch provide tools for machine learning and deep learning.
3. **Standardization:** Using well-established libraries ensures that your code follows industry standards and best practices. This can make your code more reliable and easier to understand for others who may work on your project.
4. **Community Support:** Popular libraries often have strong community support, including extensive documentation, tutorials, and forums. This makes it easier to learn how to use the library and get help when you encounter issues.
5. **Efficiency:** Libraries are often optimized for performance. By using them, you can leverage optimized algorithms and data structures that can significantly improve the efficiency of your code.
6. **Understanding our data:**

Reading out the dataset, fully formatted. For this is to having a glimpse of what is coming about the dataset, about the imbalance column, value, data, and numbers. From there we will have a look on what we are working on the dataset.

Also on this dataset, we'll try to cover it all the rows and columns, to have a better sense on what we are working on.

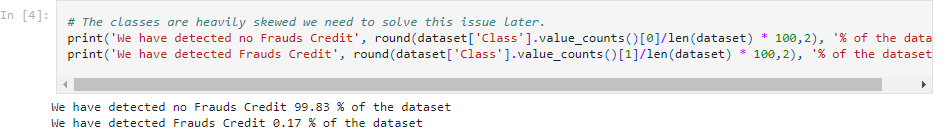
* id: Unique identifier for each transaction
* V1-V28: Anonymized features representing various transaction attributes (e.g., time, location, etc.)
* Amount: The transaction amount
* Class: Binary label indicating whether the transaction is fraudulent (1) or not (0)

To read dataset we use this command



1. **Check The Data:**

To begin with the dataset, let’s see how to dataset went.



**Printing Non-Fraudulent Transactions:**

* dataset['Class'].value\_counts(): This function counts the occurrences of each class in the 'Class' column of the dataset. It returns a Series with the count of each class (0 for non-fraudulent and 1 for fraudulent).
* value\_counts()[0]: This retrieves the count of non-fraudulent transactions (class 0).
* len(dataset): This returns the total number of transactions in the dataset.
* dataset['Class'].value\_counts()[0]/len(dataset) \* 100: This calculates the percentage of non-fraudulent transactions in the dataset.
* round(..., 2): This rounds the percentage to two decimal places.
* print('We have detected no Frauds Credit', ..., '% of the dataset'): This prints the percentage of non-fraudulent transactions along with a message.

**Printing Fraudulent Transactions:**

* **Similar to the first print statement, but this one is for fraudulent transactions.**
* **value\_counts()[1]: This retrieves the count of fraudulent transactions (class 1).**
* **The rest of the calculation follows the same steps to determine the percentage of fraudulent transactions.**

Example Output

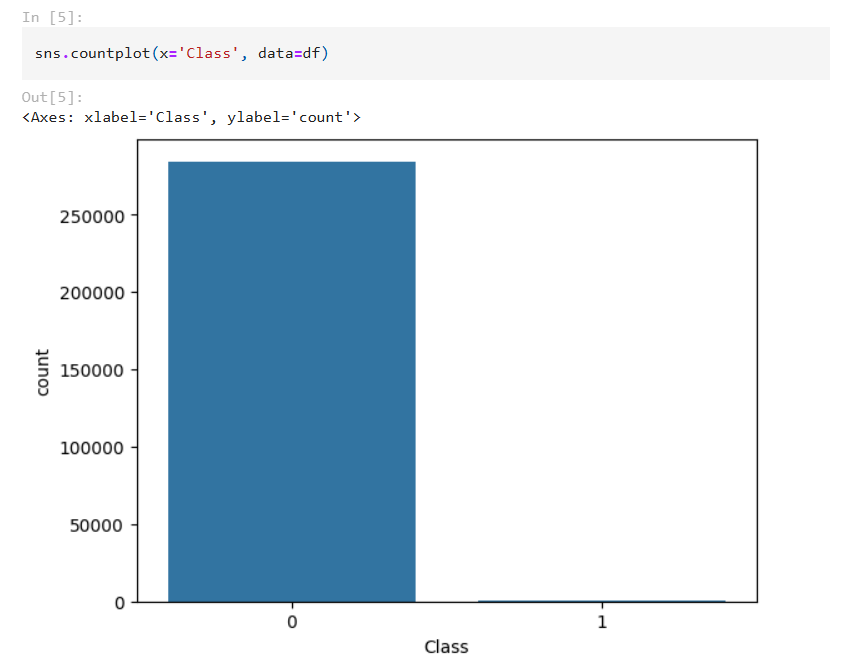
Assuming the dataset has 10,000 transactions, with 9,950 non-fraudulent and 50 fraudulent transactions, the output would be:



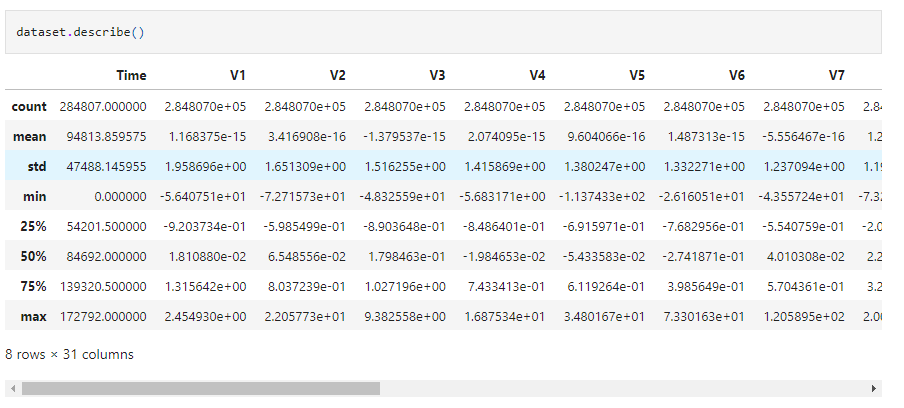
1. **Glimpse the data:**

As at the time we detection, with non format all the dataset we can find the data value seem to be off set to no Frauds Credit, with the majority of 50% of the value and only 50% are the Frauds one

Definition: Because most of the transaction need to be secure during all the transaction, with many transaction need to be secure before can other transaction can be made. So the value contain the fraud only 50% of the all the transaction made it. The dataset are validate to use as a research purposeAutomotive Industry



This dataset collected with the 50% and 50%. It's pretty rare and not common to these hence the transaction are common to have it more secure. This data maybe bit old because the date taken from 2023. These day the transaction are more secure, the fraudulent rate found are less than before.

****

1. **Result Meaning**



**Set Figure Size:** This sets the size of the entire figure to 15x15 inches.

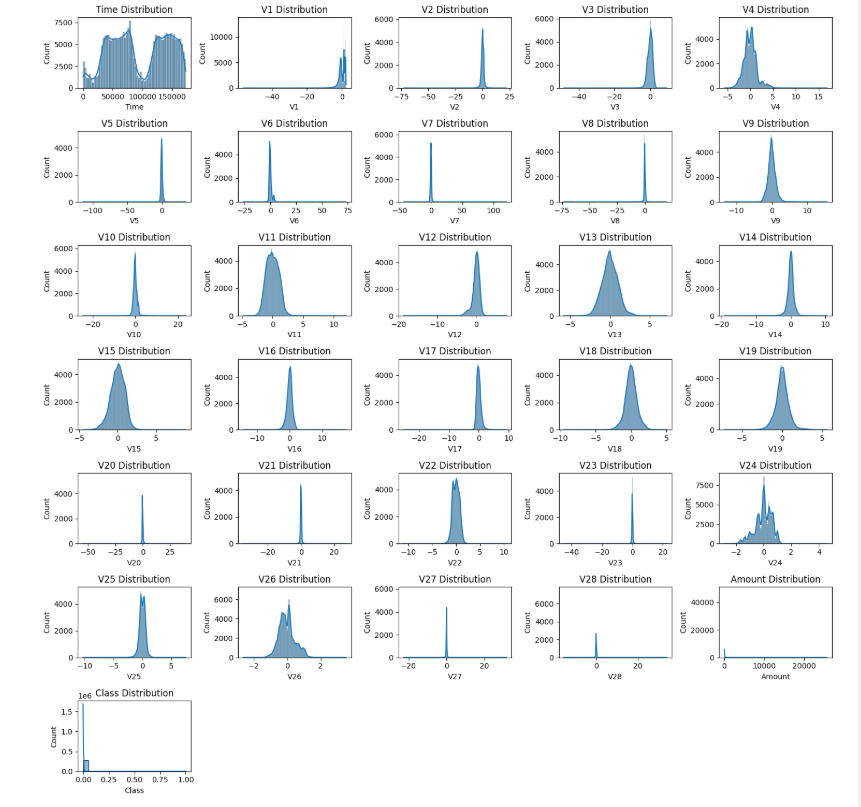
**Initialize Counter**: A counter t is initialized to 1. This counter is used to specify the position of the subplot within the figure.

**Loop Through Columns**:

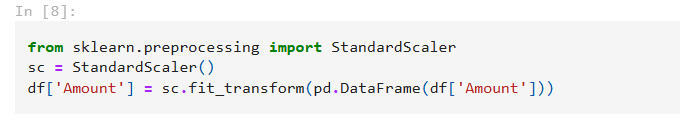
* The loop iterates through each column in the dataset.
* plt.subplot(7,5,t):
* This creates a subplot in a 7x5 grid at the position specified by t.
* sns.histplot(dataset[i], kde=True):
* This creates a histogram for the column i using Seaborn's histplot function.
* kde=True adds a Kernel Density Estimate (KDE) curve to the histogram, which represents the data's probability density function.
* plt.title(i+' Distribution'):
* This sets the title of the subplot to indicate the column name followed by "Distribution".
* t+= 1 increments the counter t by 1 for the next subplot position.

**Adjust Layout**: This adjusts the layout of the subplots to ensure there is minimal overlap and they fit neatly within the figure.

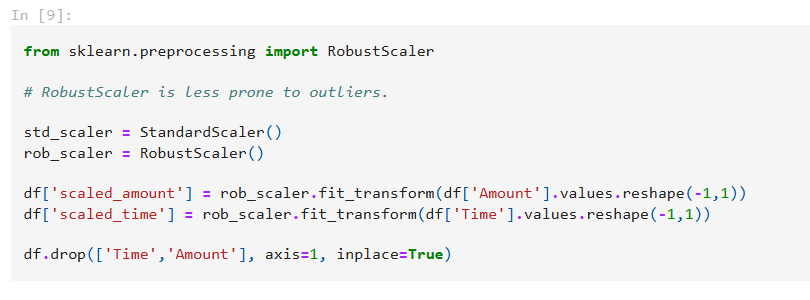
**Show the Plot**: This displays the entire figure with all the subplots.



1. **Data Preprocessing**

****

This code helps standardize the 'Amount' column in the DataFrame so that its values have a mean of 0 and a standard deviation of 1.



This code imports the necessary scaler, creates instances of the scalers, scales the 'Amount' and 'Time' columns using the RobustScaler, and then removes the original columns from the DataFrame.

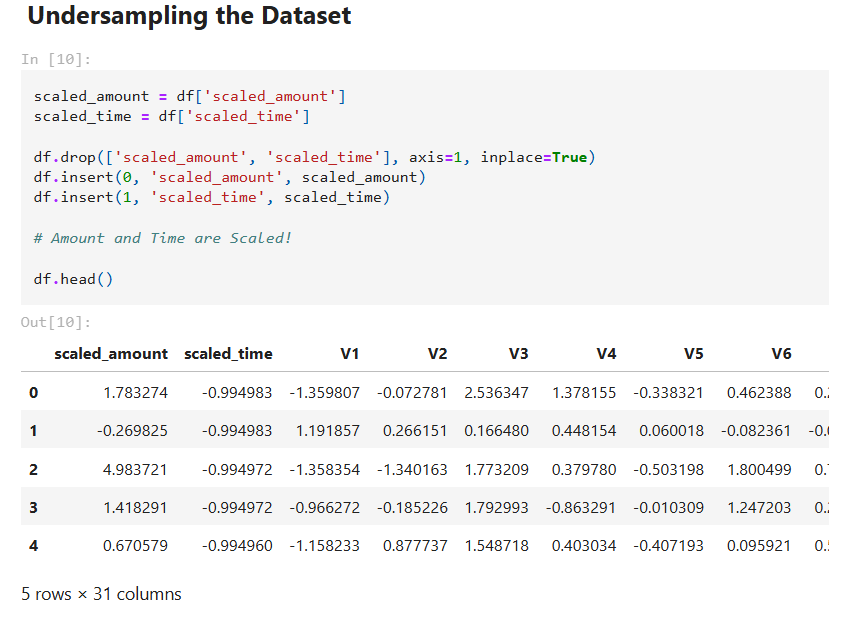
**Chapter 2: Undersampling The Dataset**

Undersampling the dataset is a technique used to address class imbalance, which is a common issue in machine learning. This approach involves reducing the number of instances in the majority class to create a more balanced dataset.

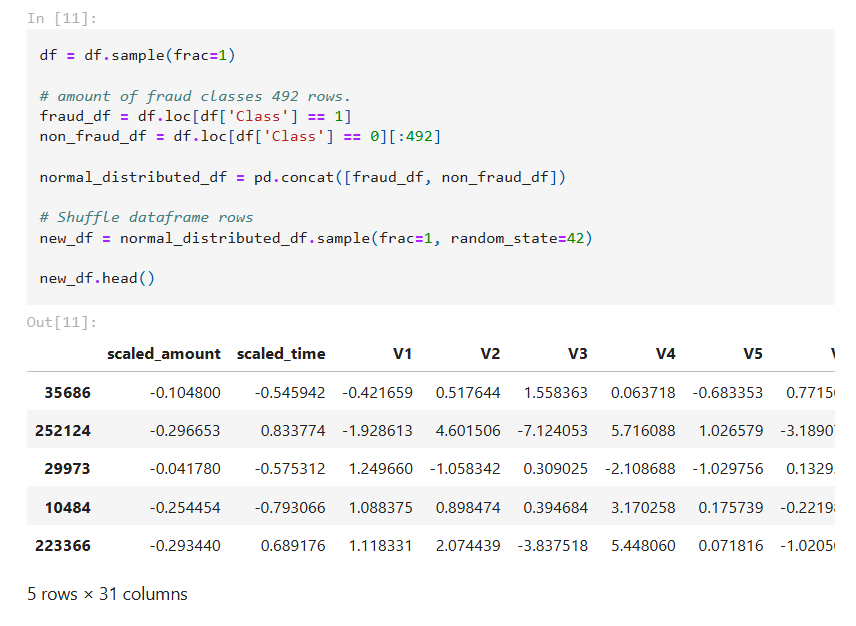
**Usage:** Undersampling is applied when the dataset contains a significantly larger number of instances for one class compared to another. By randomly selecting and removing instances from the majority class, we can create a more balanced distribution of classes. This is particularly useful for algorithms that are sensitive to class imbalance, as it helps them to better detect patterns and avoid being biased towards the majority class.

**Why Use It:** The primary reason for using undersampling is to improve model performance. In imbalanced datasets, the model can become biased towards the majority class, leading to poor performance in identifying minority class instances. By undersampling the majority class, we can ensure that the model is exposed to a more representative sample of both classes. This helps in achieving better accuracy, recall, and precision for the minority class, thus providing a more reliable and fair evaluation of the model’s performance.

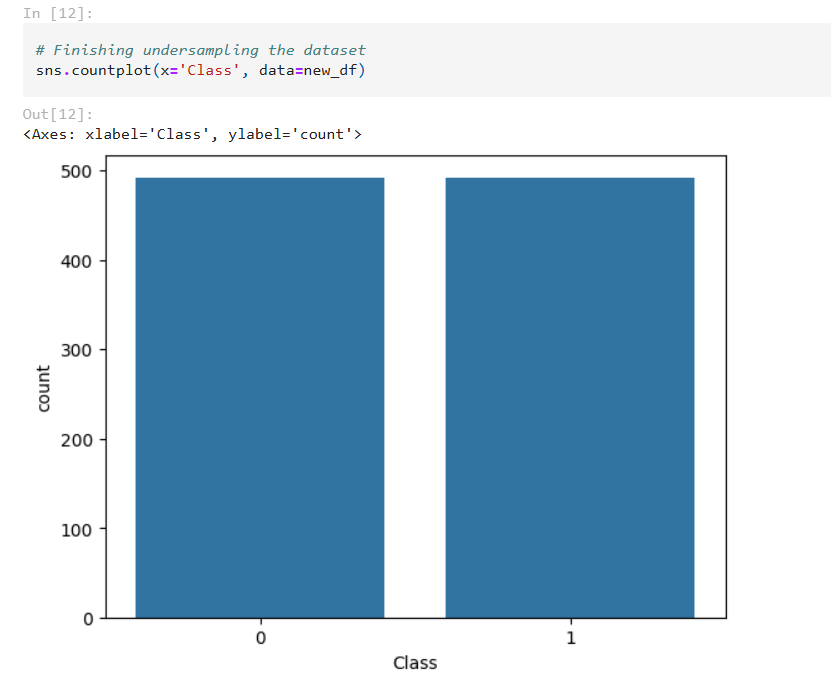
Overall, undersampling helps to mitigate the effects of class imbalance, leading to more accurate and equitable model predictions.



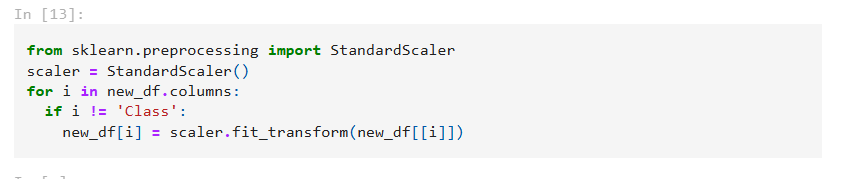
This code rearranges the DataFrame df by first extracting the scaled columns, dropping them from their original positions, and then reinserting them at the beginning of the DataFrame. This ensures that the DataFrame has the 'scaled\_amount' and 'scaled\_time' columns in the first two positions.



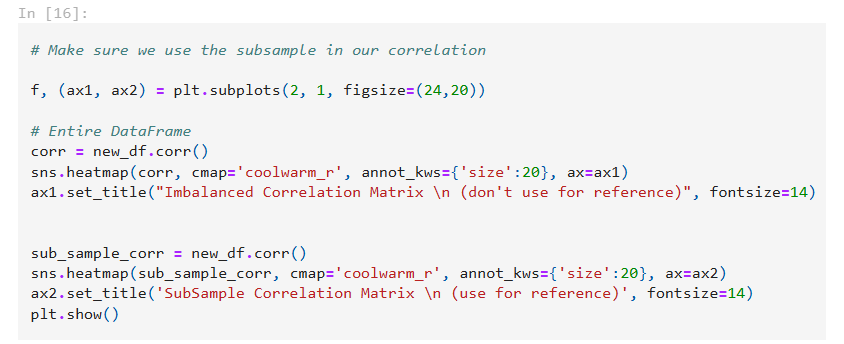
This code creates a balanced dataset with an equal number of fraud and non-fraud samples, concatenates them into a single DataFrame, shuffles the rows to ensure randomness, and then displays the first few rows of the shuffled DataFrame.



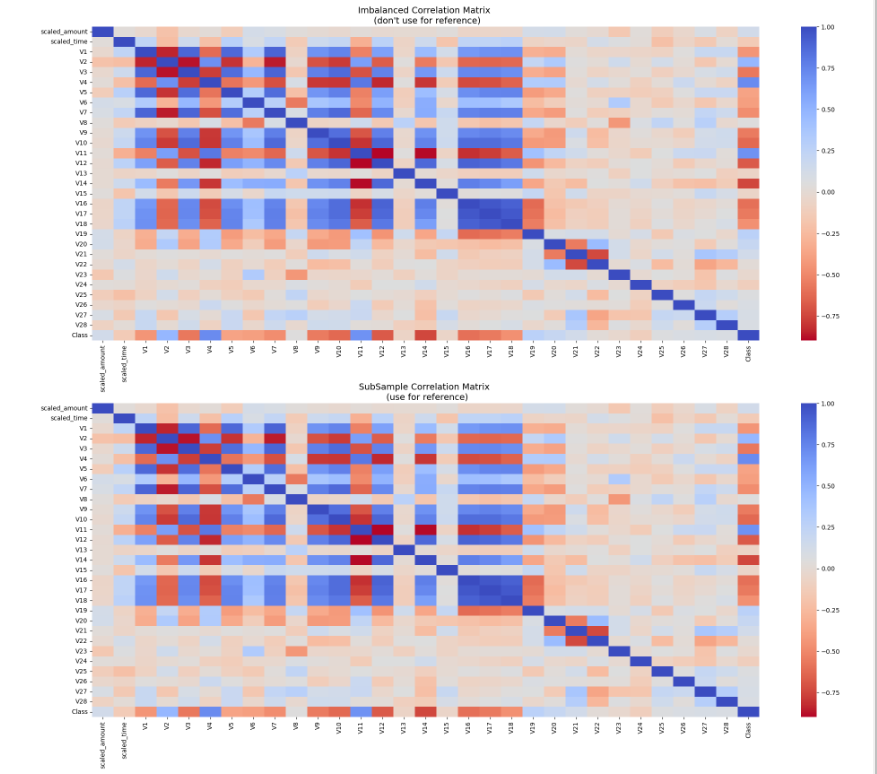
This code will generate a bar plot showing the count of each class (fraud and non-fraud) in the new\_df DataFrame. The sns.countplot function creates the plot, and plt.show() displays it.



This code standardizes all columns in the DataFrame new\_df except for the 'Class' column, ensuring that the feature values have a mean of 0 and a standard deviation of 1.

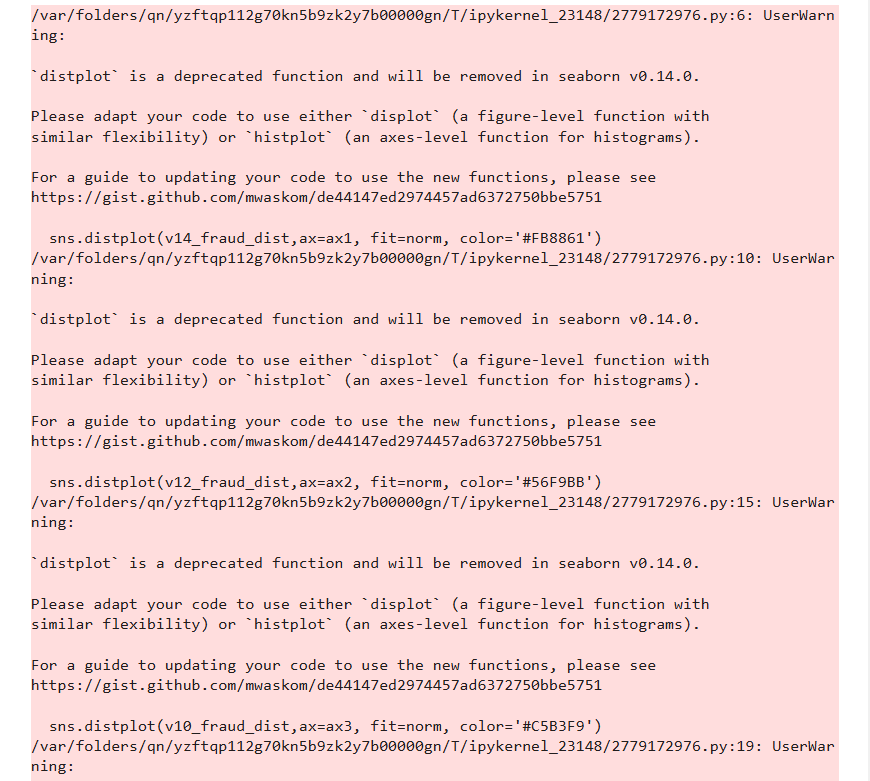


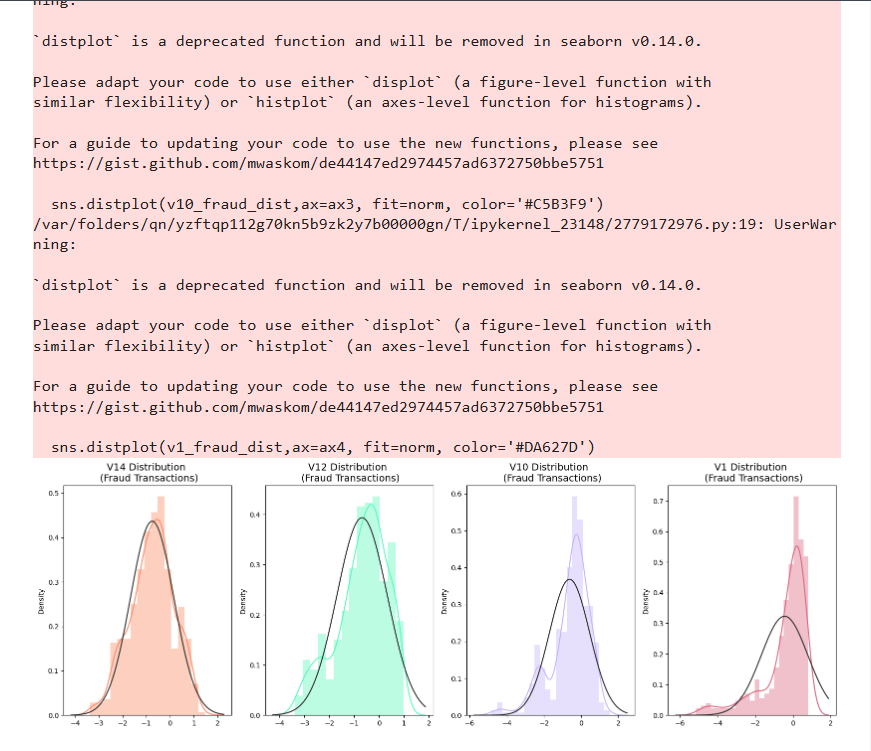
This code creates a figure with two subplots: one showing the correlation matrix for the entire DataFrame and the other showing the correlation matrix for the subsample. The heatmaps visualize the correlation values between different features, and the titles indicate which plot should be used for reference.





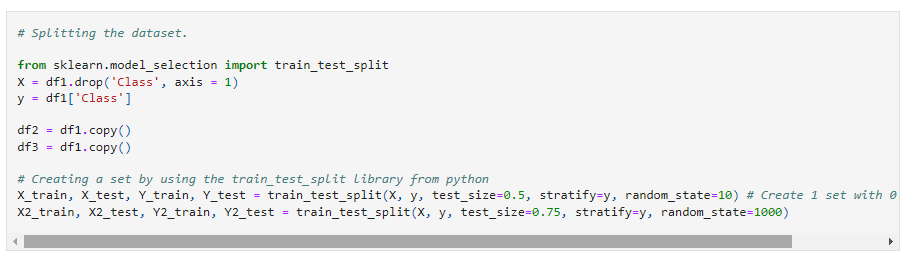
This code creates a figure with four subplots, each showing the distribution of a different feature ('V14', 'V12', 'V10', 'V1') for fraud transactions. Each distribution is fitted with a normal distribution, and the subplots are arranged horizontally.





1. **Train-Test-Split: Working On The Dataset**

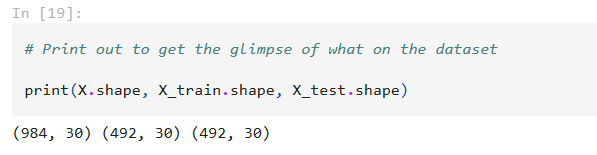
These method for training, processing the dataset, to match the usage need. By splitting the dataset, we'll have a glimpse of what the dataset working on



The code performs the following steps:

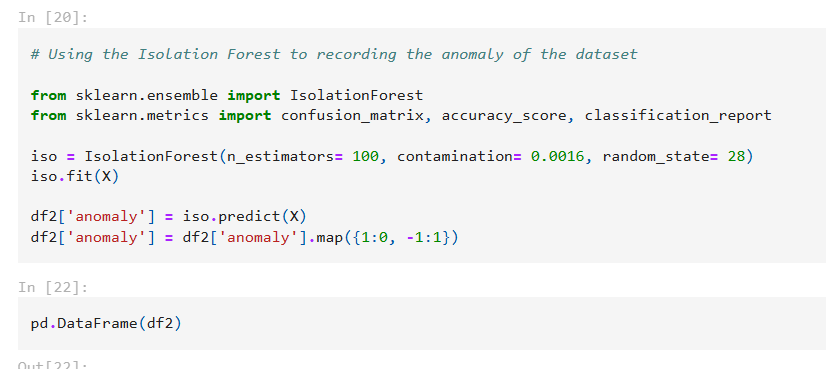
1. Imports the necessary library.
2. Defines the features (X) and target (y) variables.
3. Creates copies of the dataset.
4. Splits the dataset into training and testing sets twice, with different test sizes and random seeds.

This is useful for preparing the data for model training and evaluation while ensuring that the class distribution is preserved.

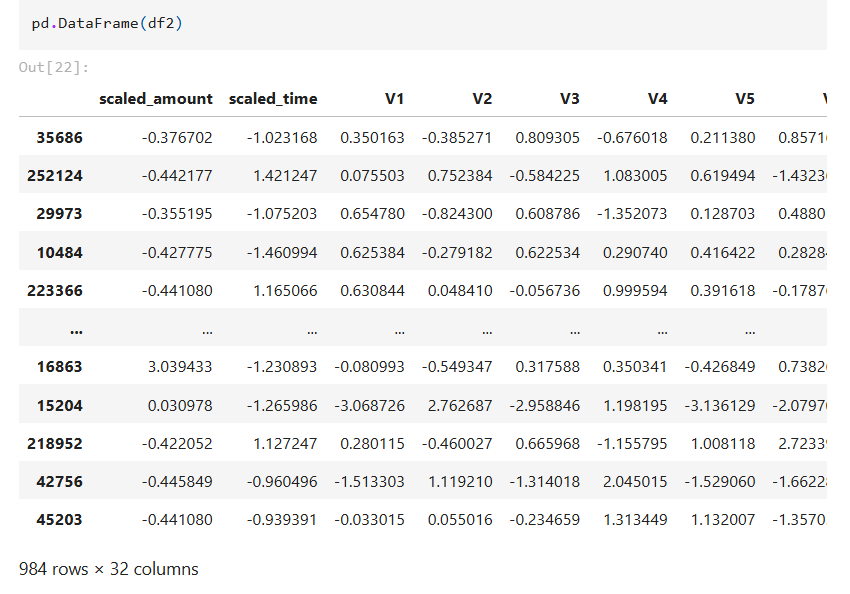


This code is helpful for understanding the dimensions of your datasets, especially when you're working with train-test splits and need to ensure they are correctly partitioned.

With the above the dataset, the testing method function will take it out 31743 non fraudulent, and 30 transaction



This code uses the Isolation Forest algorithm to detect anomalies in the dataset X, predicting anomalies and mapping the results to a new column 'anomaly' in the DataFrame df2.



1. **Understanding The Sampling Method**

The dataset we work imbalanced, because of the inequality of the fraudalent - non-fraudulent. We need to Scale the dataset to, with SMOTE (to balanced out the imbalanced one), by multiplying the fraudent one to match the scale of the non-fraudulent.

**Understanding SMOTE:**

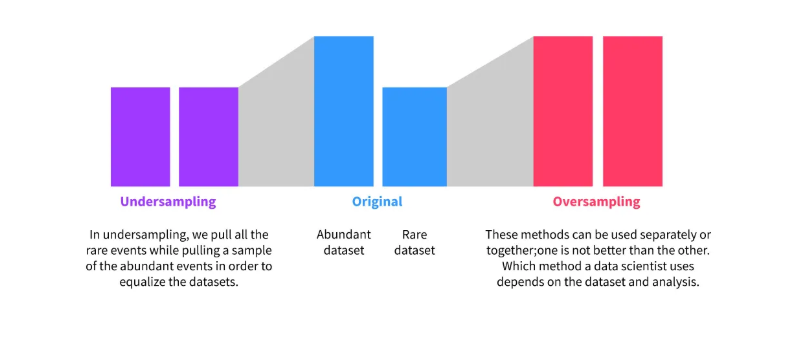
**Addressing Class Imbalance**: SMOTE (Synthetic Minority Over-sampling Technique) generates new instances for the minority class, aiming to balance the dataset with the majority class.

**Generation of Synthetic Points**: By examining the closest neighbors within the minority class, SMOTE identifies distances and then creates synthetic data points between these neighbors.

**Preserving Data Integrity**: Unlike random undersampling which might remove useful data, SMOTE retains all the original data points, thus preserving the overall dataset integrity and providing more information for training.

**Accuracy vs. Training Time**: While SMOTE often improves accuracy compared to random undersampling, it does come with a trade-off. Since no data is discarded, the dataset size increases, leading to longer training times.

Reference:https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets



**Chapter 3: Using Model To Balance Data**

1. **Using Logistic Regression**

**Logistic Regression** is a powerful tool used for binary classification tasks. It is particularly effective when the goal is to model the relationship between a binary dependent variable and one or more independent variables, using probabilities derived from a logistic function.

**Why Use Logistic Regression for Balanced Data**:

* **Interpretability**: The coefficients in logistic regression provide clear insights into how each feature influences the outcome, making it easier to understand the model's decisions.
* **Balanced Performance**: Logistic regression is well-suited for datasets where the classes are balanced, ensuring that the model learns an unbiased decision boundary between the classes.
* **Probabilistic Predictions**: The model provides probabilities for class membership, which can be useful in various decision-making scenarios where the level of confidence matters.

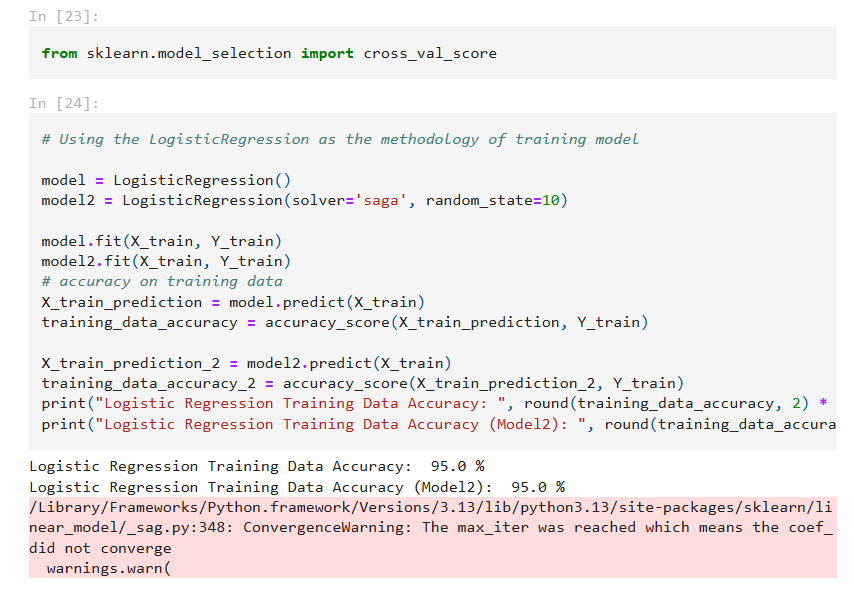
**When to Use Logistic Regression**:

* **Binary Outcomes**: Ideal for problems with two possible outcomes (e.g., success/failure, fraud/no fraud).
* **Balanced Classes**: Best applied when the classes in the dataset are roughly equal in number.

**Results**:

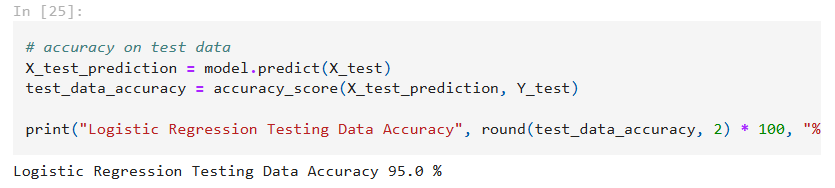
* **Balanced Performance**: Logistic regression can provide balanced metrics, ensuring fair identification of both classes.
* **Insightful Predictions**: The model offers insights into which features are most influential in determining the outcome, aiding in better decision-making.

This logistic regression is the based model that most of use will be using during the first section



* **Import Libraries:** The code first imports LogisticRegression from sklearn.linear\_model and accuracy\_score from sklearn.metrics.
* **Initialize and Train Models:**
* model: A Logistic Regression model with default parameters.
* model2: A Logistic Regression model using the 'saga' solver and a specific random state to ensure reproducibility.
* Both models are fitted to the training data X\_train and Y\_train.
* **Predict and Calculate Accuracy:**
* Predictions for the training data are made using both models.
* The accuracy\_score function compares the predicted labels with the actual labels (Y\_train) to calculate accuracy.
* **Print Results:** The training accuracy for both models is printed, rounded to two decimal places and multiplied by 100 to represent it as a percentage.

This helps you understand the effectiveness of your model in detecting fraudulent transactions in the test dataset.



After testing with multiple dataset and model training with Logistic Regression can learning up-to 0.99% with this dataset, and compare with the training data accuracy of 0.998%

The margin of those result are comparably small. So with Logistic Regression, the data result are fairly accurate

1. **Using Decision Tree Classifier**

**Decision Tree Classifier** is a widely used algorithm for classification tasks in machine learning. It works by splitting the dataset into branches based on the values of the input features, creating a tree-like structure where each node represents a decision based on a feature, and each branch represents an outcome.

**Why Use Decision Tree Classifier for Balanced Data**:

* **Easy to Interpret**: Decision trees are straightforward to visualize and understand, making it clear how decisions are made and which features play a key role.
* **Non-Linear Relationships**: They can capture complex, non-linear patterns in the data that simpler models might miss.
* **No Preprocessing Required**: They handle both numerical and categorical data without needing feature scaling or normalization.
* **Versatile**: Suitable for various types of data and can handle multiclass classification as well.

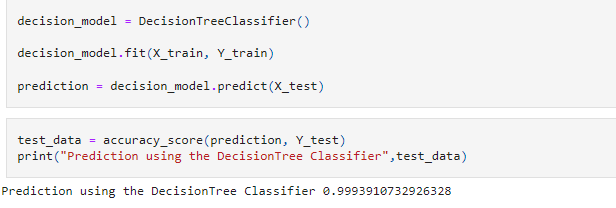
**When to Use**:

* **Classification Problems**: Ideal for problems where you need to assign data points to specific categories.
* **Balanced Data**: Works best when the dataset has a balanced class distribution. For imbalanced data, additional techniques like undersampling, oversampling, or using class weights are recommended.

**Results**:

* **Accuracy**: Measures how well the model correctly classifies data points.
* **Classification Report**: Provides detailed metrics like precision, recall, and F1-score for each class.
* **Confusion Matrix**: Shows the true positive, true negative, false positive, and false negative predictions.

The Decision Tree is a way to split out the training the dataset, each of every node will split to 2 type of statement, testing out all of the testing purpose to find out hte model working for the dataset.



The code initializes a Decision Tree Classifier, trains it using the training data, makes predictions on the test data, calculates the accuracy of these predictions, and prints the accuracy score. This process helps evaluate how well the Decision Tree Classifier can detect fraudulent transactions in the test dataset.

1. **Using The KNN Classifier**

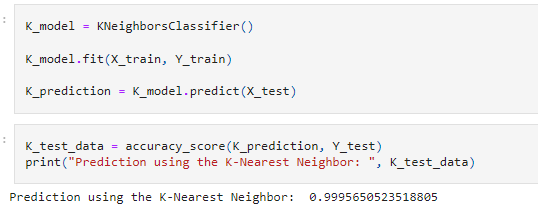
**K-Nearest Neighbors (KNN) Classifier** is a simple yet powerful machine learning algorithm used for classification tasks. It works by finding the k-nearest data points (neighbors) to a query point and making a prediction based on the majority class among these neighbors.

**Why Use KNN Classifier for Balanced Data:**

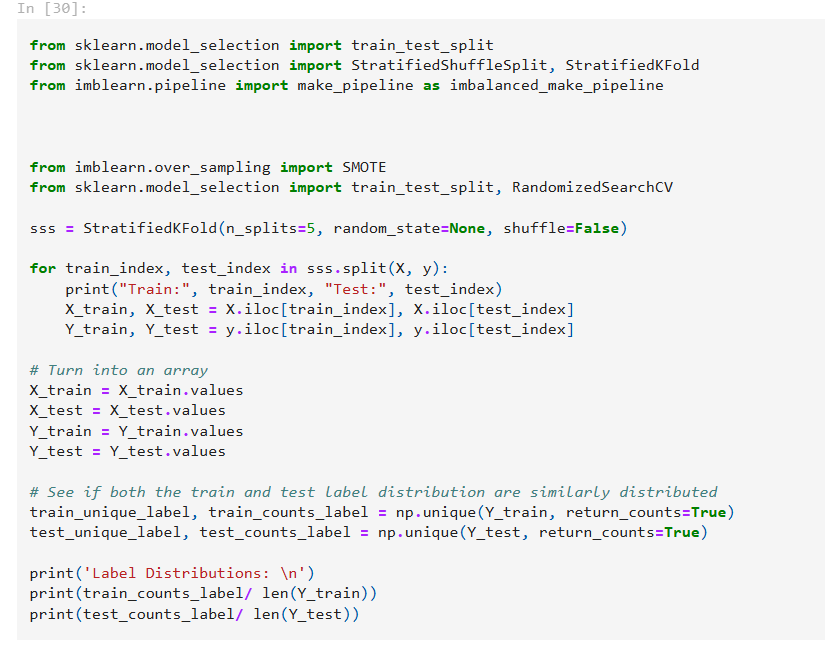
* **Simplicity**: KNN is straightforward to understand and implement. It doesn't require any assumptions about the data distribution.
* **Effectiveness**: When the dataset is balanced, KNN can be very effective as it relies on the majority class within the neighborhood for making predictions.
* **Non-parametric**: KNN is a non-parametric method, meaning it doesn't make any underlying assumptions about the data's distribution.

**When to Use:**

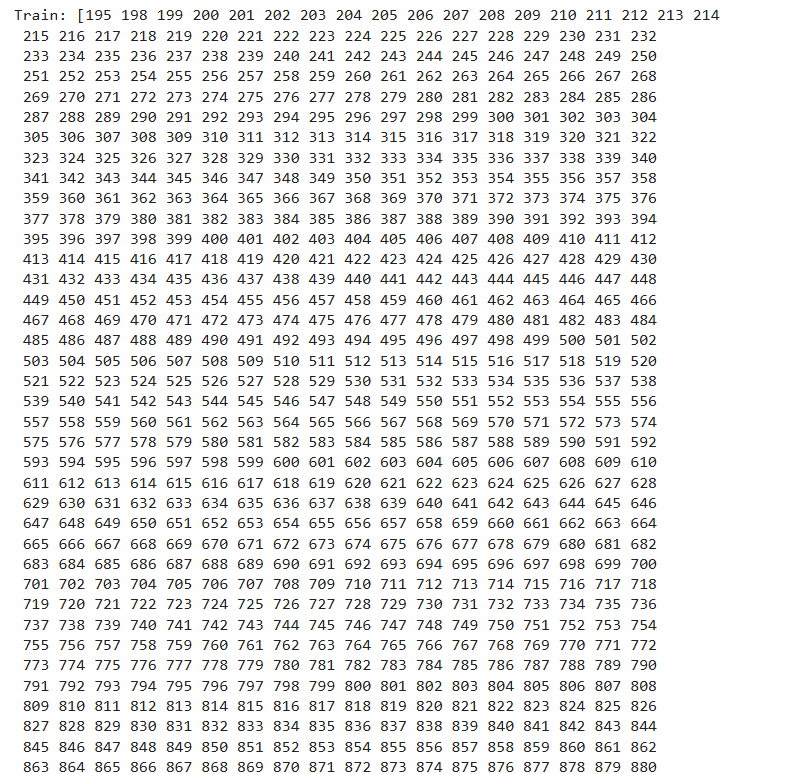
* **Classification Tasks**: KNN is suitable for problems where the objective is to classify data points into predefined categories.
* **Balanced Datasets**: It's most effective when the classes in the dataset are balanced because it uses the majority class within the neighborhood for prediction.
* **Small to Medium-sized Datasets**: KNN works best with smaller datasets as its computational cost increases with the size of the dataset.

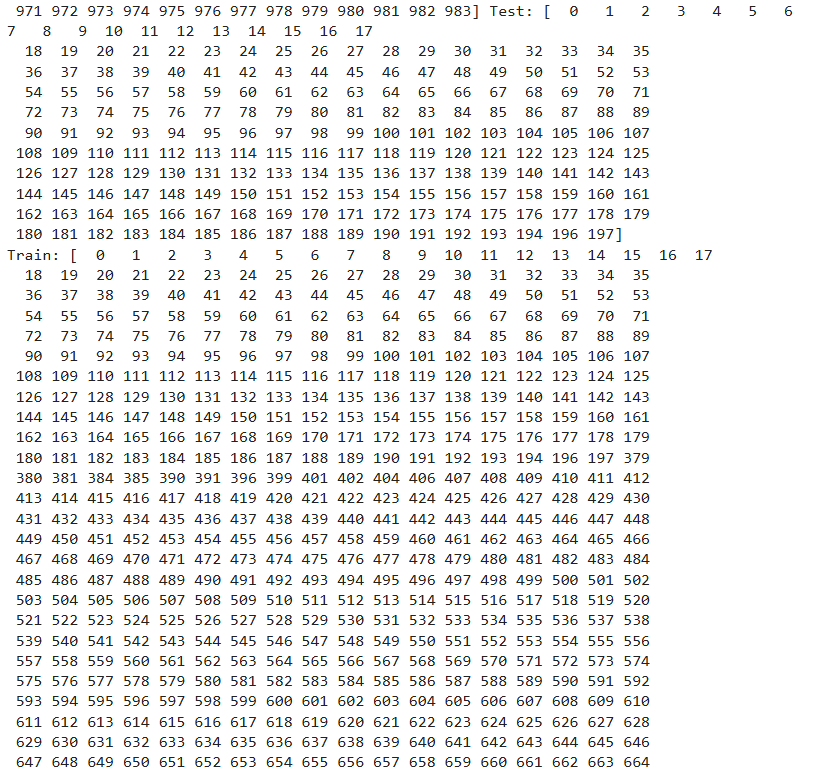


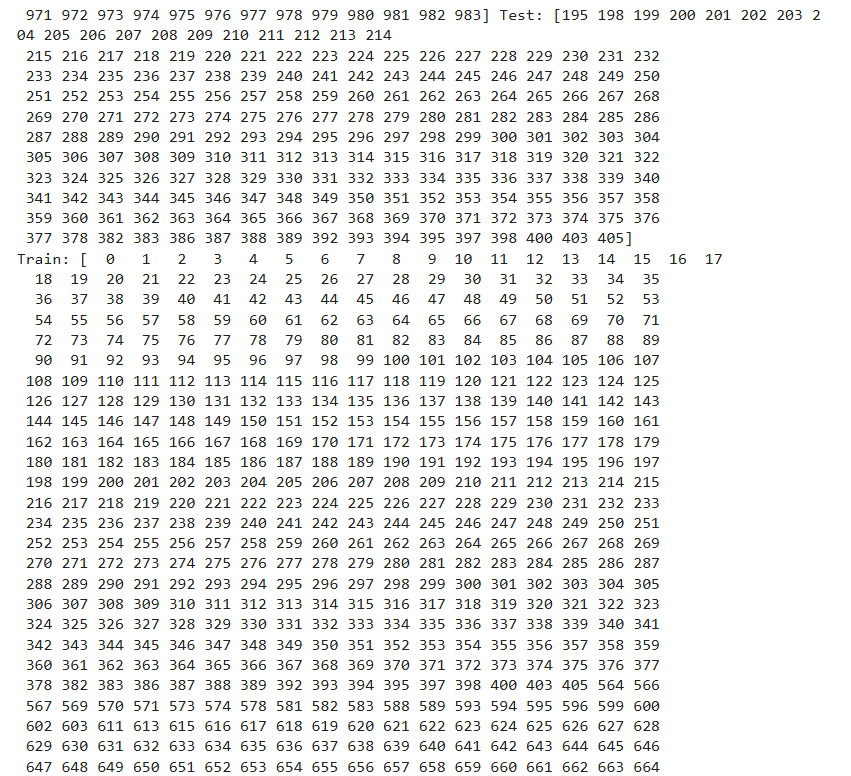
The code initializes a K-Nearest Neighbors (KNN) classifier, trains it using the training data, makes predictions on the test data, calculates the accuracy of these predictions, and prints the accuracy score. This process helps evaluate how well the KNN classifier can detect fraudulent transactions in the test dataset.

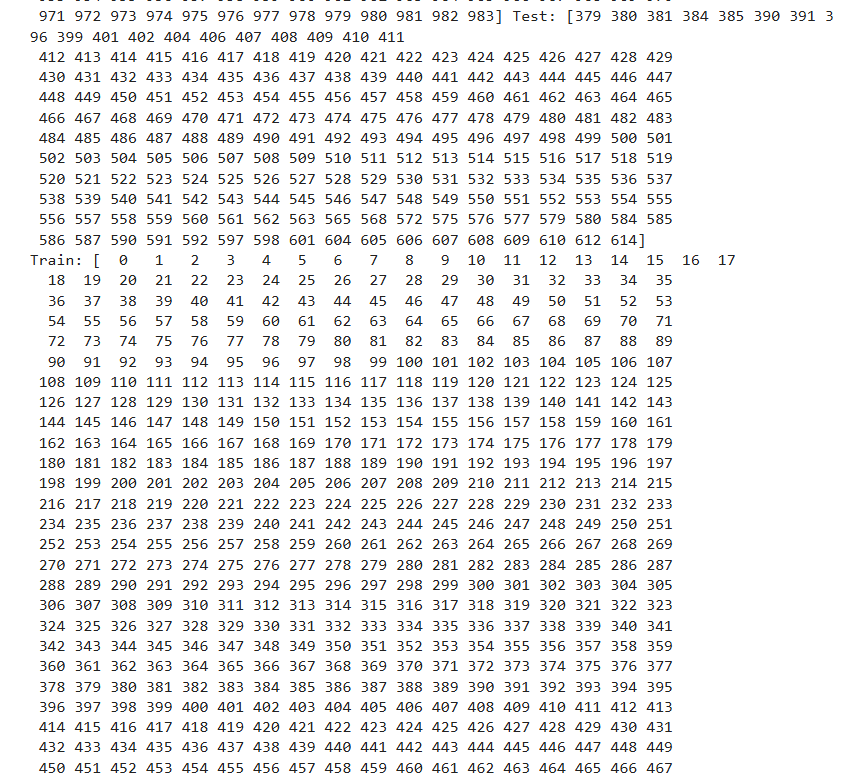


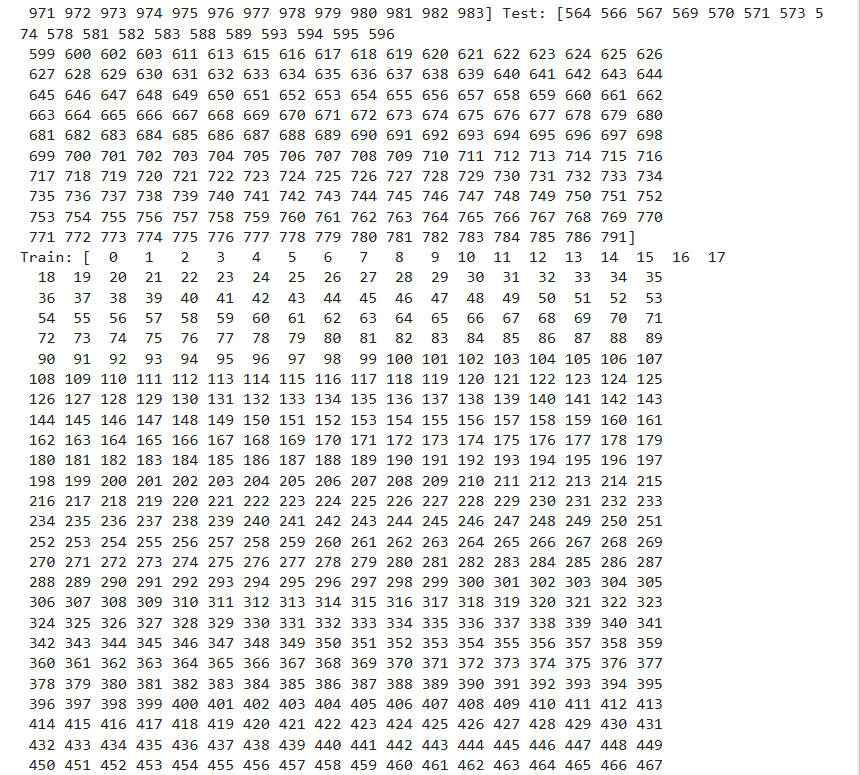
* The code imports necessary libraries and modules.
* It initializes StratifiedKFold for 5 splits.
* The loop splits the data into training and testing sets using stratified sampling to ensure balanced label distribution.
* The training and testing sets are converted to arrays.
* Finally, it checks and prints the label distribution in both training and testing sets to ensure they are similarly distributed.

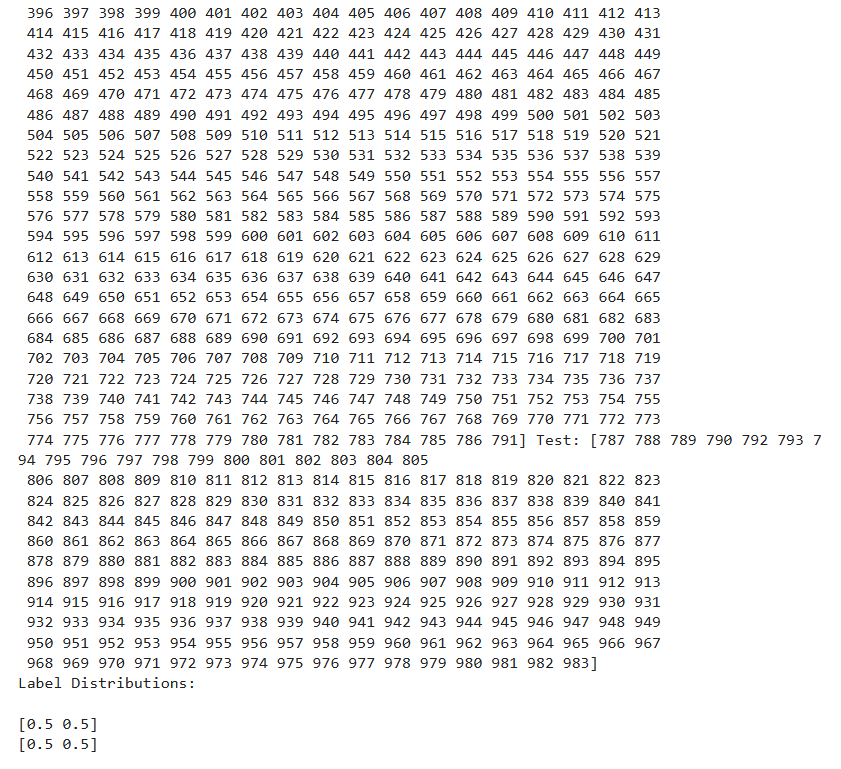
****

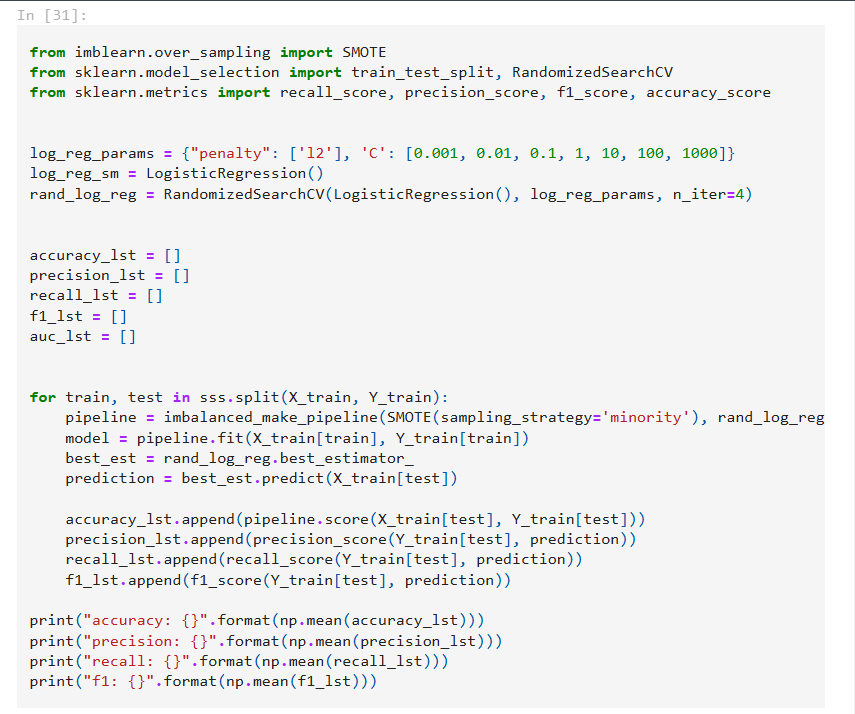
****

****

****

****

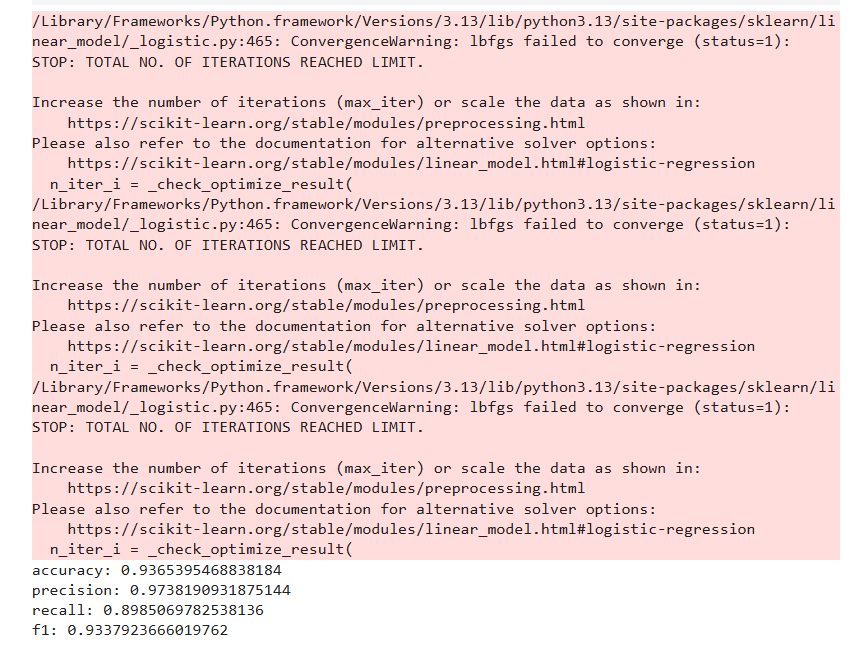
****

****

* **Goal:** Train and evaluate a Logistic Regression model to detect anomalies using imbalanced datasets and improve performance through hyperparameter tuning and cross-validation.
* **Steps:**

1. **Import Libraries:** Bring in necessary libraries for data preprocessing, model training, and evaluation.
2. **Set Up Parameters:** Define hyperparameters for Logistic Regression and initialize models.
3. **Cross-Validation:** Use StratifiedKFold for cross-validation, ensuring balanced label distribution.
4. **Handle Imbalanced Data:** Apply SMOTE (Synthetic Minority Over-sampling Technique) to handle class imbalance.
5. **Train and Evaluate:** Train the model, make predictions, and calculate performance metrics (accuracy, precision, recall, F1-score) for each fold.
6. **Report Results:** Print the average performance metrics.

* **Purpose:** To build a reliable anomaly detection model and ensure fair evaluation using cross-validation and proper handling of imbalanced data.

****

**References**

1. <https://blog.tomorrowmarketers.org/thuat-toan-phan-loai-classification-machine-learning/>
2. <https://www.kaggle.com/code/chanchal24/credit-card-fraud-detection>
3. <https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive-models>