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)*

**TABLE OF CONTENTS**

[Chapter 1: INTRODUCTION 7](#_Toc190713854)

[Chapter 2: THEOREM OF WORKING WITH THE DATASET AND MODEL 7](#_Toc190713855)

[2.1 Dataset overview 7](#_Toc190713856)

[2.2 Model Training Pipeline 8](#_Toc190713857)

[2.3 Undersampling: 8](#_Toc190713858)

[2.4 Oversampling: 8](#_Toc190713859)

[2.5 Combination Techniques (Optional): 9](#_Toc190713860)

[2.6 K-Nearest Neighbors(KNN): 9](#_Toc190713861)

[2.7 Random Forest: 9](#_Toc190713862)

[2.8 Support Vector Machine(SVM): 9](#_Toc190713863)

[2.9 Model Evaluation: 9](#_Toc190713864)

[Chapter 3: READ AND UNDERSTANDING DATA 10](#_Toc190713865)

[3.1 Import Libraries: 10](#_Toc190713866)

[3.2 Understanding our data: 2](#_Toc190713867)

[3.3 Check The Data: 2](#_Toc190713868)

[3.4 Glimpse the data: 3](#_Toc190713869)

[3.6 Data Preprocessing 7](#_Toc190713870)

[CHAPTER 4: UNDERSAMPLING THE DATASET 8](#_Toc190713871)

[4.1 Using Random Under Sampling 8](#_Toc190713872)

[4.2 Using NearMiss UnderSampling 13](#_Toc190713873)

[**4.2.1** **TRAIN-TEST-SPLIT** 14](#_Toc190713874)

[4.3 Using for Random Under Sampling 14](#_Toc190713875)

[**4.3.1** **Train, test, split** 14](#_Toc190713876)

[4.4 Building LogisticRegression with RUS 16](#_Toc190713877)

[4.5 Using Decision Tree Method 17](#_Toc190713878)

[4.6 Using K-Nearest Neighbors Classifier 18](#_Toc190713879)

[4.7 Using Support Vector Machine 18](#_Toc190713880)

[4.8 Using NearMiss Under Sampling Method 19](#_Toc190713881)

[**4.8.1** **Train Test Split** 19](#_Toc190713882)

[4.9 Using LogisticRegression Model 20](#_Toc190713883)

[4.10 Using Decision Tree Classifier 20](#_Toc190713884)

[4.11 Using the SVM 21](#_Toc190713885)

[4.12 Using Clustered Centroid 21](#_Toc190713886)

[**4.12.1** **Train test split** 21](#_Toc190713887)

[4.13 Using Decision Tree Model 22](#_Toc190713888)

[4.14 Using SVM 23](#_Toc190713889)

[5.1 Using traditional SMOTE approach 23](#_Toc190713890)

[5.2 Alternative way to implement 26](#_Toc190713891)

[5.3 SMOTE Oversampling Method 27](#_Toc190713892)

[5.3.1 Result Logistic Regression with SMOTE Oversampling: 27](#_Toc190713893)

[5.3.2 Result Decision Tree with SMOTE Oversampling 27](#_Toc190713894)

[5.3.3 Result K-Nearest Neighbors with SMOTE Oversampling 28](#_Toc190713895)

[5.4 Using Random Over-sampler Method (ROS) 28](#_Toc190713896)

[**5.4.1** Result Logistic Regression with ROS: 28](#_Toc190713897)

[5.4.2 Result with Decision Tree with ROS 28](#_Toc190713898)

[5.4.3 Result with K-Nearest Neighbors with ROS: 29](#_Toc190713899)

[5.5 Using ADASYN: 29](#_Toc190713900)

[***5.5.1*** Result Logistic Regression with ADASYN: 29](#_Toc190713901)

[5.5.2 Result Decision Tree Method with ADASYN 29](#_Toc190713902)

[5.5.3 Result K-Nearest Neighbors with ADASYN: 29](#_Toc190713903)

[References 30](#_Toc190713904)

[1 https://blog.tomorrowmarketers.org/thuat-toan-phan-loai-classification-machine-learning/ 30](#_Toc190713905)

[2 https://www.kaggle.com/code/chanchal24/credit-card-fraud-detection 30](#_Toc190713906)

[3 https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive-models 30](#_Toc190713907)

**CREDIT CARD FRAUD DETECTION  
WITH IMBALANCED DATA**

# **Chapter 1: INTRODUCTION**

Former introduction about team, we are Phan An Duy and Lam Ngoc Hai, both from class 18H50301 and for this report, we’ll be covering the topic of **CREDIT CARD FRAUD DETECTION WITH IMBALANCED DATA**. 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 tend to be biased towards the dataset.

Why we are choosing this dataset, beside from the requirement from the deans of the faculty require our project, this project is one of the major issue throughout the economy of the day to day task. The project are the comprehensive of most the dataset.

**Chapter 2: THEOREM OF WORKING WITH THE DATASET AND MODEL**

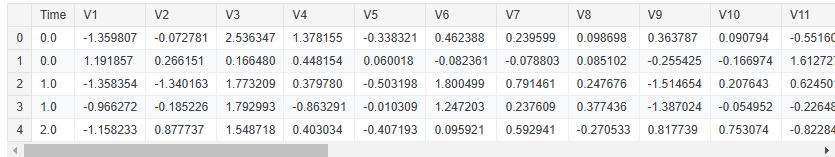
* 1. **Dataset overview**

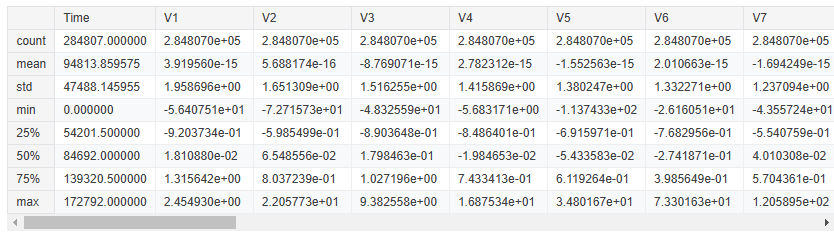
This dataset was set to be a model training for today’s issue about the fraud credit card transaction, the credit card transaction from the early days are mostly fraudulent and non-fraudulent. The dataset built overtime, the dataset tendency to built to last most of the time. In the dataset from 2023, the fraud and non-fraud are in with the margin of 50/50 now the fraud dataset only cover about 0.17%. It show that overtime the transaction are much more secure during these day. It is our job to deal with this dataset, implementing about Machine Learning, Training the model to predict hand testing out the accuracy.

* 1. **Model Training Pipeline**

For today’s project, we’ll be focusing on how to implementing the reading the dataset, finding the anomalies, reading out, ploting out the dataset to see what. Using the pandas as the dataframe reader. Reading out the dataset to get what is happening on the dataset, with how many field, are there any not a number to be dropping out, missing values, null values, anything that may affect the dataset training.

After looking the dataset, we need to specifiy the scaling method throughout the form.

****

****

* 1. **Undersampling:**
* **NearMiss:** Reduces the majority class by selecting instances that are close to the minority class.
* **Random Undersampling:** Randomly removes instances from the majority class.
* **ClusterCentroid:** Uses clustering to reduce the number of majority class instances.
  1. **Oversampling:**
* **SMOTE (Synthetic Minority Over-sampling Technique):** Creates synthetic instances for the minority class.
* **ADASYN (Adaptive Synthetic Sampling):** Similar to SMOTE but focuses on more difficult-to-learn instances.
* **Random Oversampling:** Randomly duplicates instances of the minority class.
  1. **Combination Techniques (Optional):**
* **SMOTENC:** A variant of SMOTE that handles continuous and categorical features.
* **SMOTETomek:** Combines SMOTE with Tomek links to remove instances that are borderline and overlap between classes.
* **SMOTEENN:** Combines SMOTE with Edited Nearest Neighbors to clean the dataset by removing noisy examples.

These techniques are applied only to the training set to ensure that the model learns from balanced data without introducing bias from the oversampling process into the test set.

* 1. **K-Nearest Neighbors(KNN):**

A straightforward algorithm that predicts the class of a data point based on the majority class among its k-nearest neighbors. It works well with balanced data and is easy to understand and implement.

* 1. **Random Forest:**

An ensemble method that combines multiple decision trees to improve classification performance and robustness. It is particularly effective for handling non-linear relationships and providing feature importance.

* 1. **Support Vector Machine(SVM):**

A powerful classifier that finds the optimal hyperplane to separate different classes. SVMs are effective in high-dimensional spaces and can handle non-linear classification with the use of kernel functions.

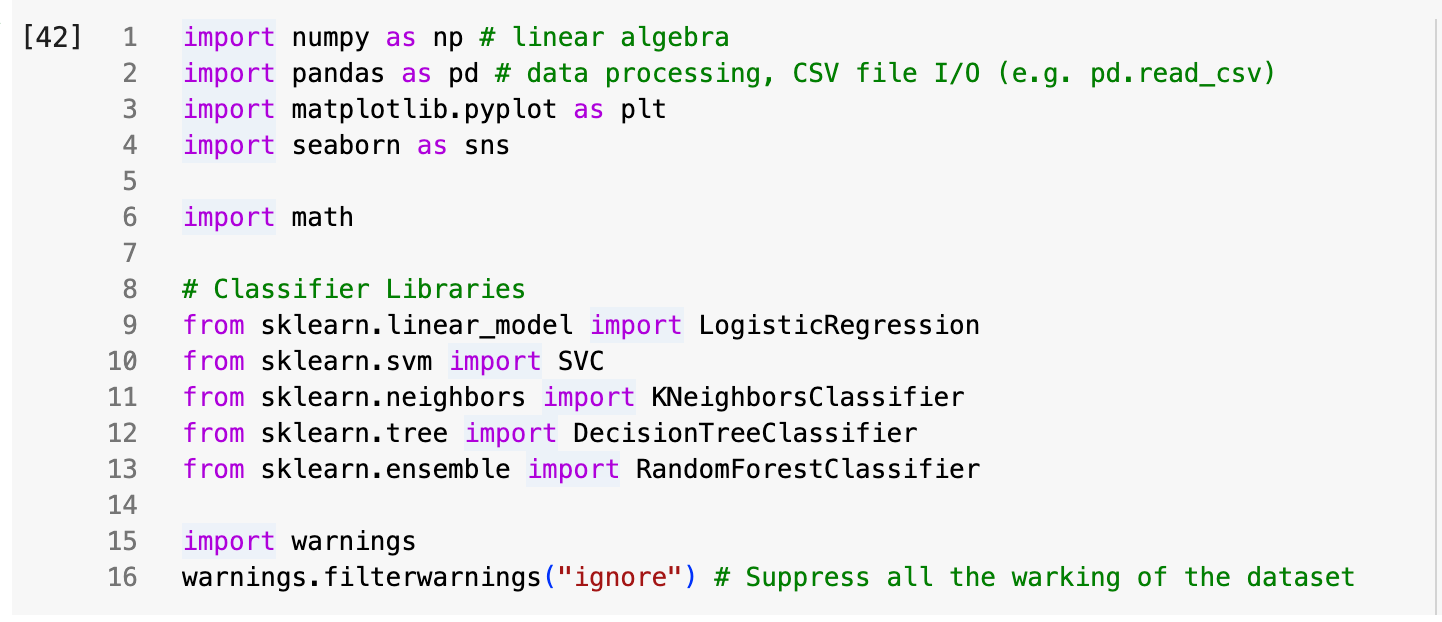
* 1. **Model Evaluation:**

The performance of these models is evaluated on the test set using metrics such as accuracy, precision, recall, and F1-score. These metrics help in understanding how well the model performs in distinguishing between the classes.

By following this methodology, we ensure that the classification model is robust, accurate, and capable of effectively identifying both majority and minority classes. This chapter lays the groundwork for the practical implementation of data preprocessing, sampling techniques, and model evaluation, leading to the development of a reliable and balanced classification model.

# **Chapter 3: READ AND UNDERSTANDING DATA**

## **3.1 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.

**3.2 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.

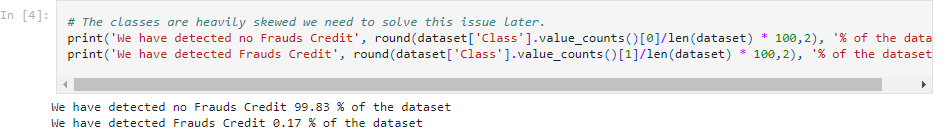
* id: Unique identifier for each transaction
* V1-V28: Anonymized features representing various transaction attributes (e.g., time, location, etc.)
* Amount: The transaction amount
* Class: Binary label indicating whether the transaction is fraudulent (1) or not (0)

To read dataset we use this command



## **3.3 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.

From the dataset, there are 99.83 % of non-fraudulent and 0.17% of the fraudulent credit cards. The data are imbalance due to the lack of non-secure transaction

**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*

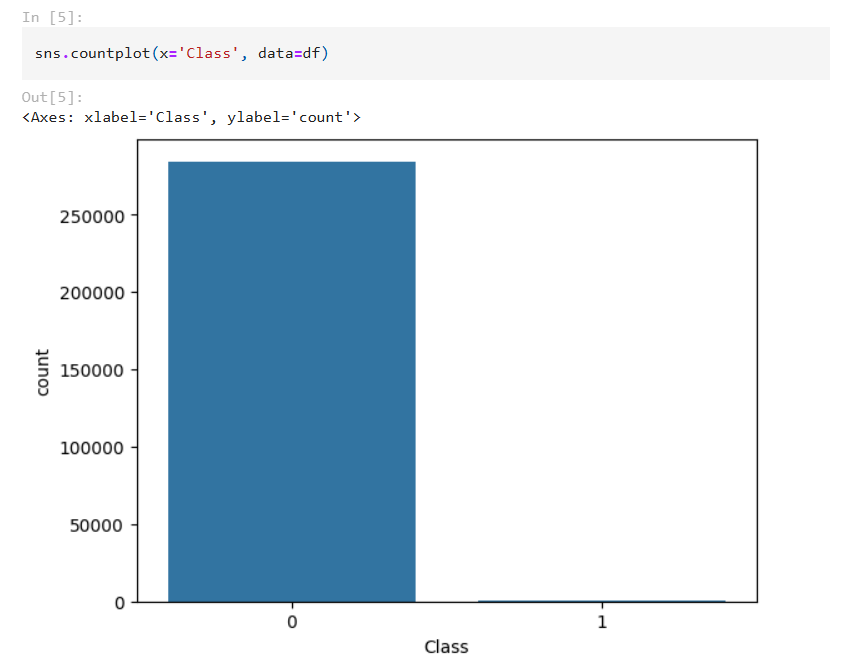
Assuming the dataset has 10,000 transactions, with 9,950 non-fraudulent and 50 fraudulent transactions, the output would be:



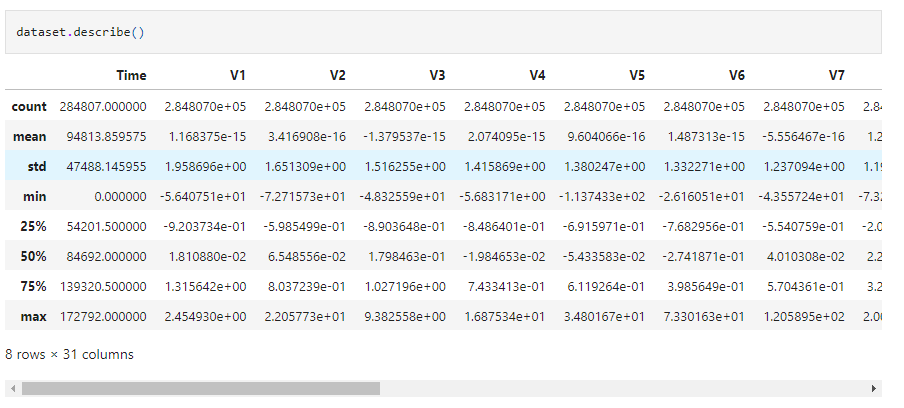
**3.4 Glimpse the data:**

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



As the label are 0 and 1 are Non-Fraudulent and Fraudulent respectively, and the dataset are in the bias of the Non-fraud side, with since today the transaction are in the secure side, the dataset for fraud is minimal

**3.5 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**:

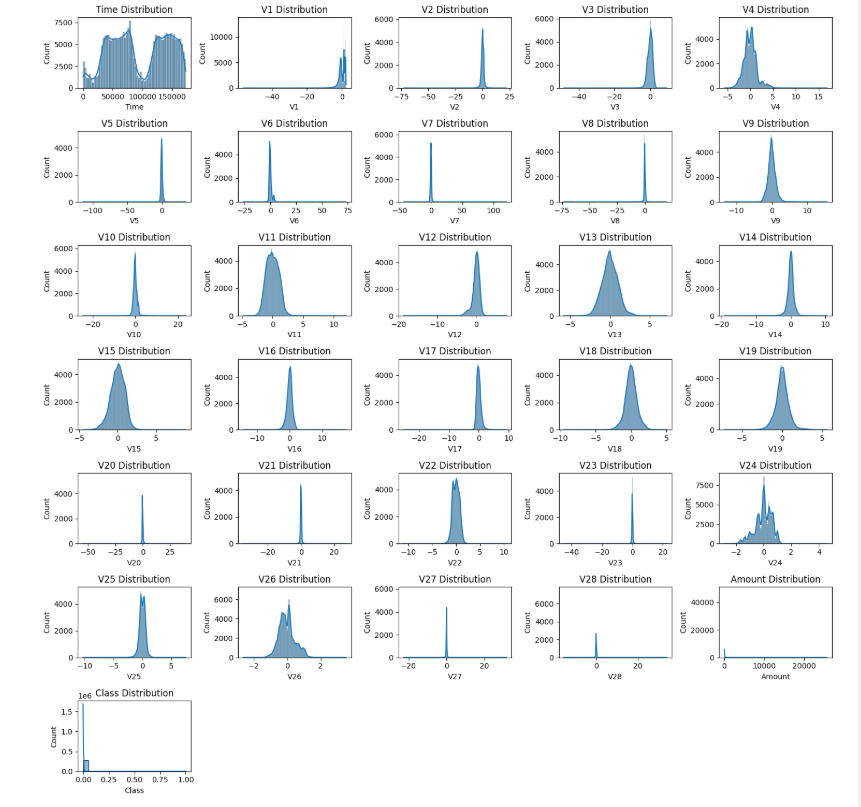
* The loop iterates through each column in the dataset.
* plt.subplot(7,5,t):

This creates a subplot in a 7x5 grid at the position specified by t.

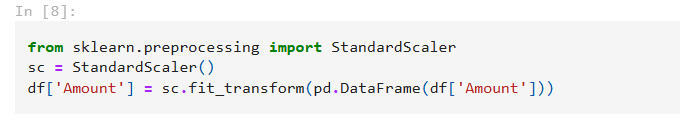
* sns.histplot(dataset[i], kde=True):
* This creates a histogram for the column i using Seaborn's histplot function.
* kde=True adds a Kernel Density Estimate (KDE) curve to the histogram, which represents the data's probability density function.
* plt.title(i+' Distribution'):
* This sets the title 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.

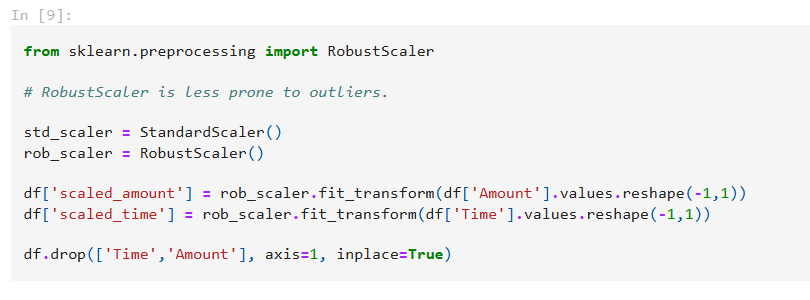
**Show the Plot**: This displays the entire figure with all the subplots.



## **3.6 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.

These will scale down to match the label ready for the dataset sampling and training



*Code sample of scaling the data set, using fit and transform.*

**CHAPTER 4: UNDERSAMPLING THE DATASET**

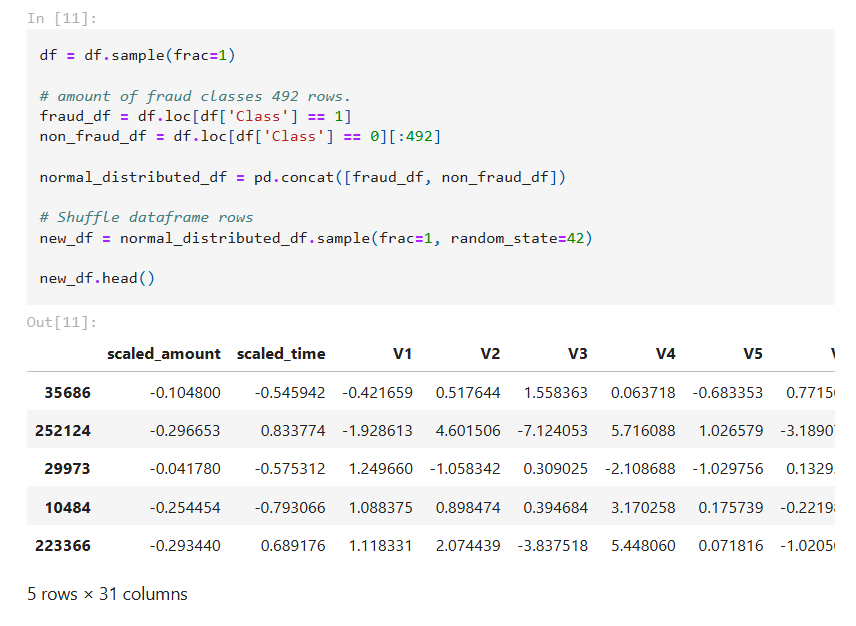
Undersampling the dataset is a technique used to address class imbalance, which is a common issue in machine learning. This approach involves reducing the number of instances in the majority class to create a more balanced dataset.

**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.

**Based** on the dataset, we can see the non fraudulent are massive, so for the first undersampling method, we will be using the Random Undersampling Method, Later we will use the NearMiss Method to test the differentials.

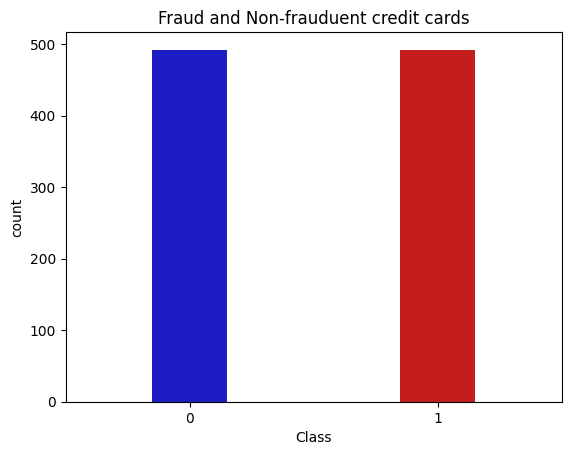
**4.1 Using Random Under Sampling**

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.



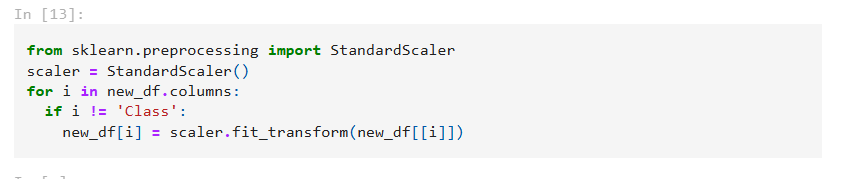
*Code sample using Random Under Sampling*

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, after under sampling the dataset by dropping the non-fraud scale down to the same size as the fraud.

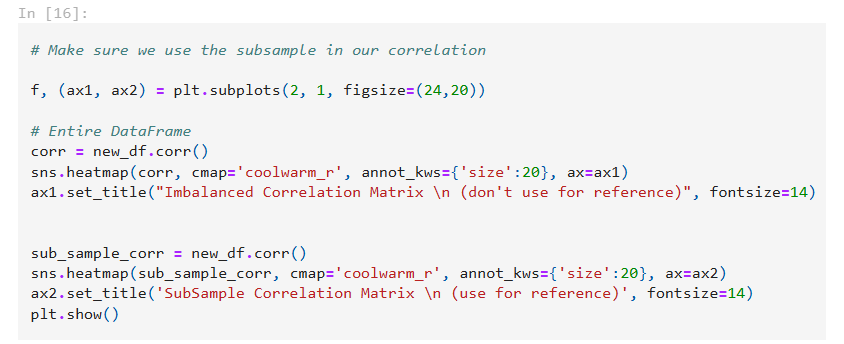


*After scale down, the dataset are now equal to the train test*

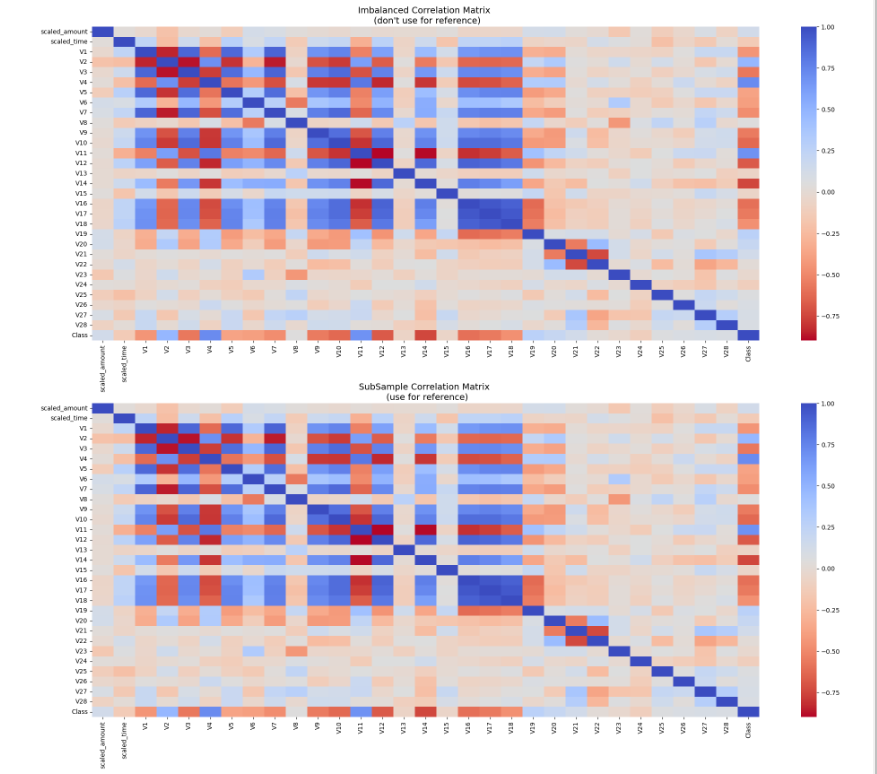
This code will generate a bar plot showing the count of each class (fraud and non-fraud) in the new\_df DataFrame. The sns.countplot function creates the plot, and plt.show() displays it.



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.

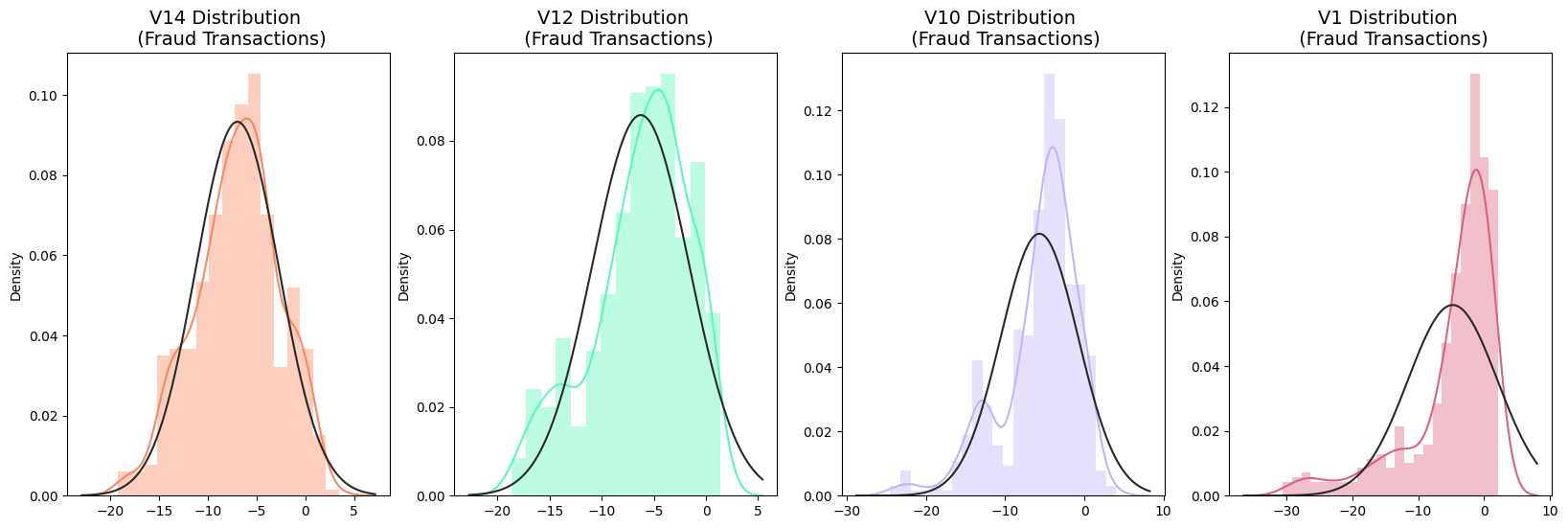


This code creates a figure with two subplots: one showing the correlation matrix for the entire DataFrame and the other showing the correlation matrix for the subsample. The heatmaps visualize the correlation values between different features, and the titles indicate which plot should be used for reference.



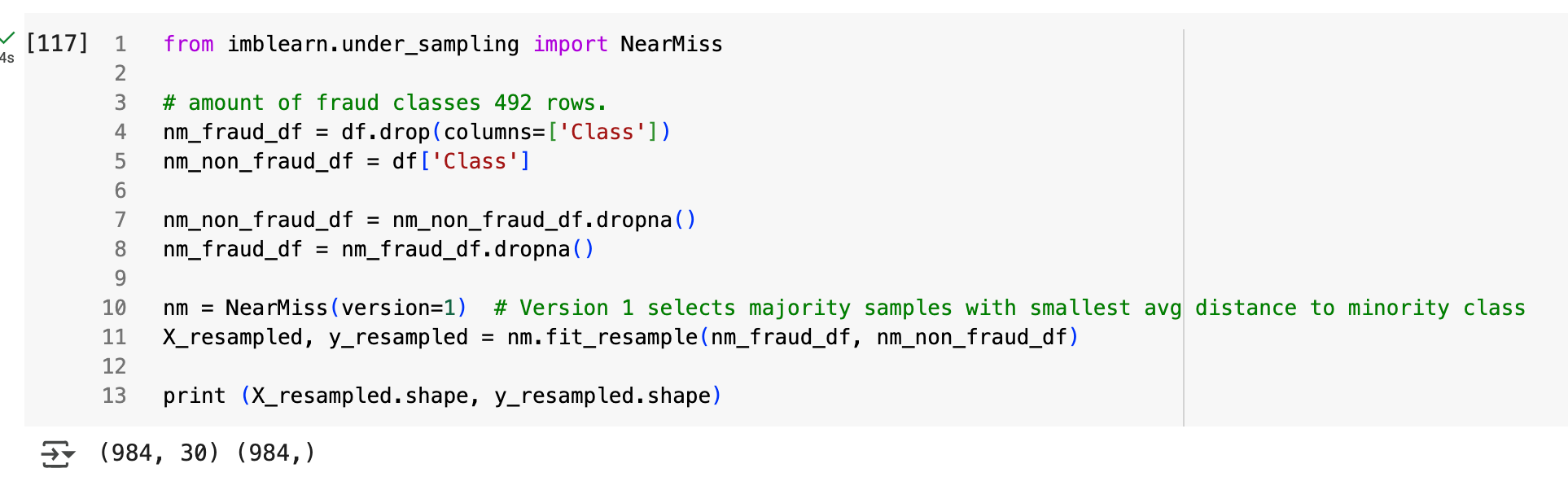


This code creates a figure with four subplots, each showing the distribution of a different feature ('V14', 'V12', 'V10', 'V1') for fraud transactions. Each distribution is fitted with a normal distribution, and the subplots are arranged horizontally.



4.2 **Using NearMiss UnderSampling**

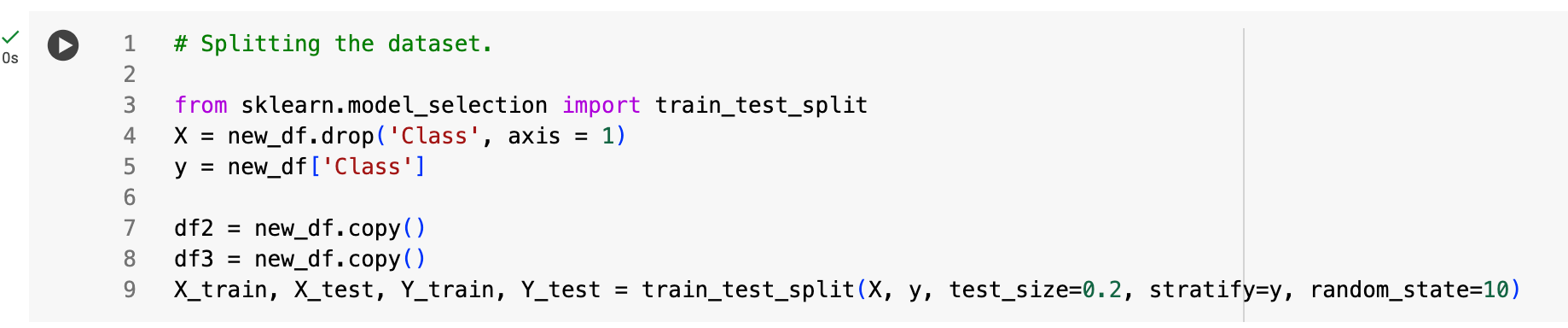
Near Miss UnderSampling is another method to scale down the dataset as it shown in the code below



Using the library to test with the fill dataset, create a new dataset after scaled down, balanced out. But it may cost the fact that some of the field have prone to accommodate the NaN, so on line 7-8 will dropna() all from the dataset.

* + 1. **TRAIN-TEST-SPLIT**
  1. **Using for Random Under Sampling**
     1. **Train, test, split**

These method for training, processing the dataset, to match the usage need. This first segment for train test split will apply for the RUS method



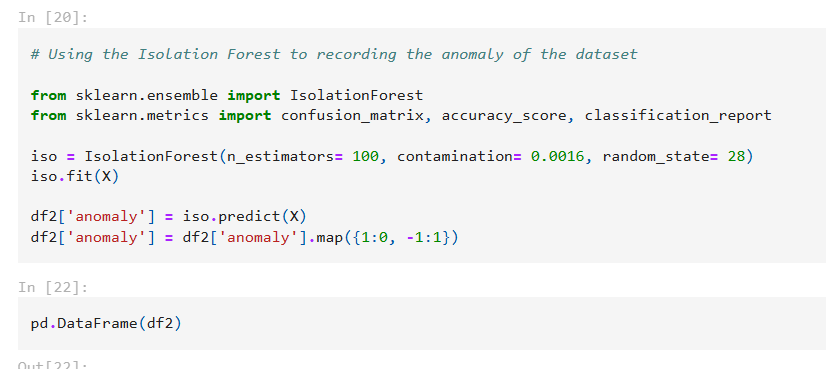
*Data will be trained and store as X\_train, X\_test, Y\_train, Y\_test*

The code performs the following steps:

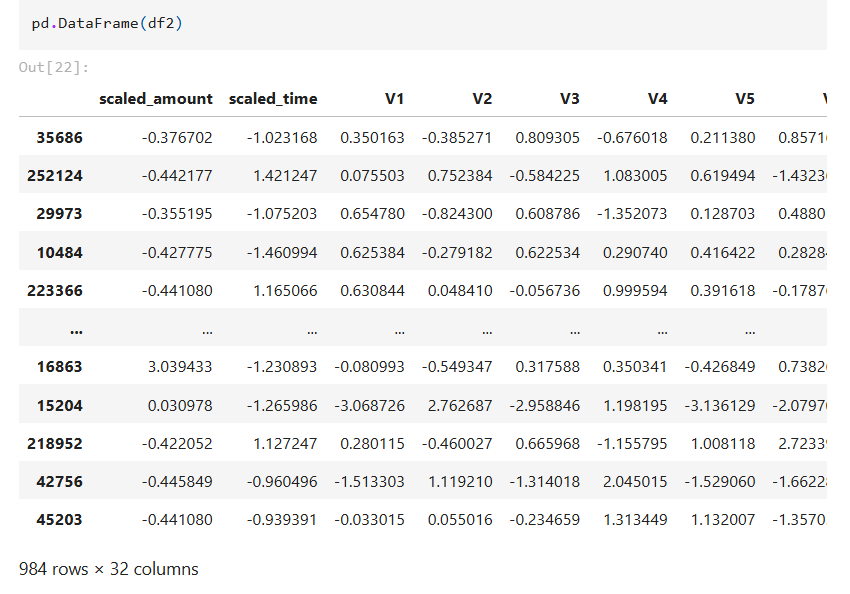
1. Imports the necessary library.
2. Defines the features (X) and target (y) variables.
3. Creates copies of the dataset.
4. Splits the dataset into training and testing sets , with different test sizes and random seeds.

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.



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.

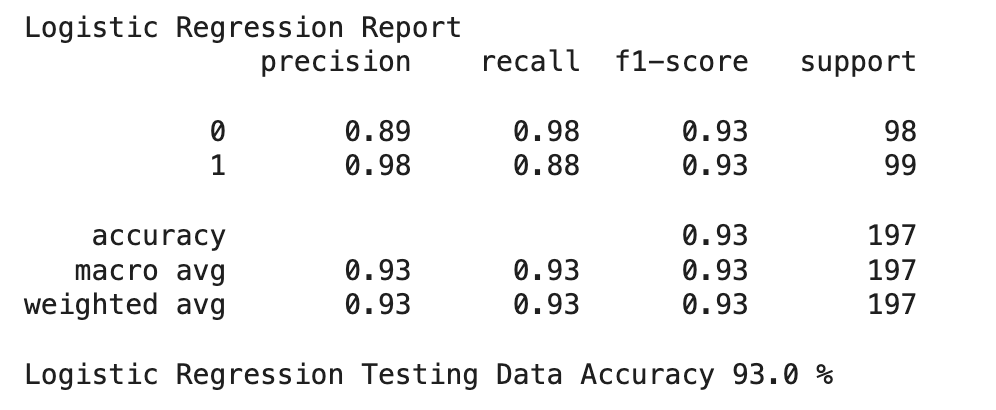


* 1. **Building LogisticRegression with RUS**

****

***Coding method for Logistic Regression***

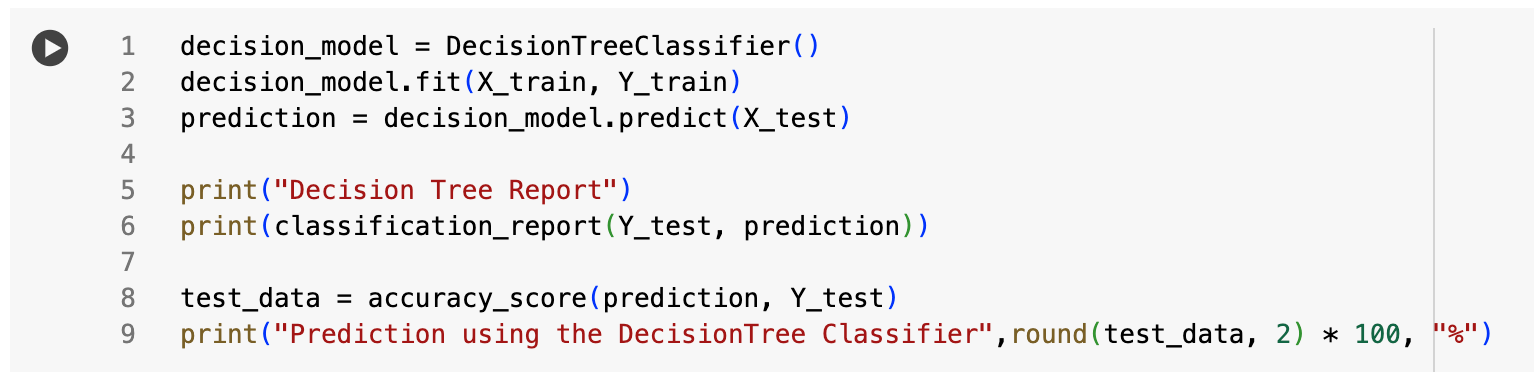
This method will fit the LogisticRegression for the RUS sampling, as the result will log as the classification report here



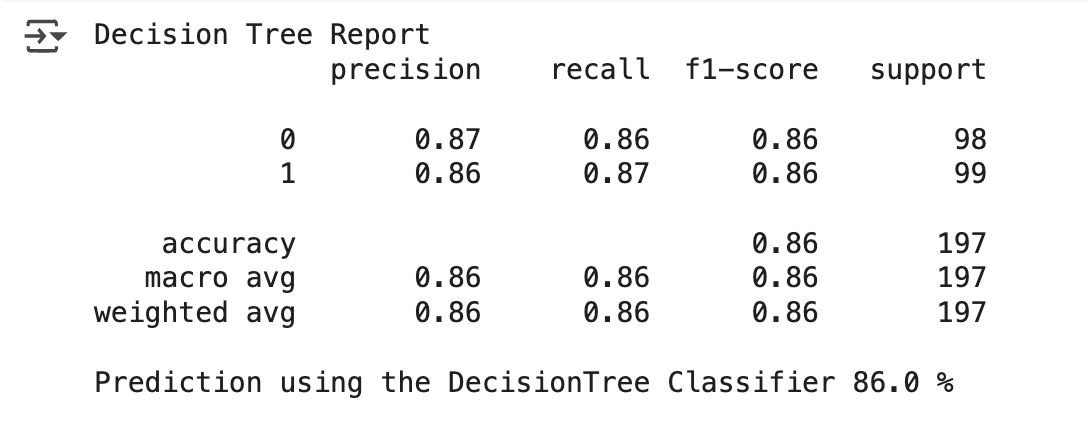
As the Logistic Regression method training out, the accuracy are 93% of the efficiency

* 1. **Using Decision Tree Method**

Decision Tree is a supervised learning algorithm that can solve both regression and classification problems. As with this method, the code below are representing it



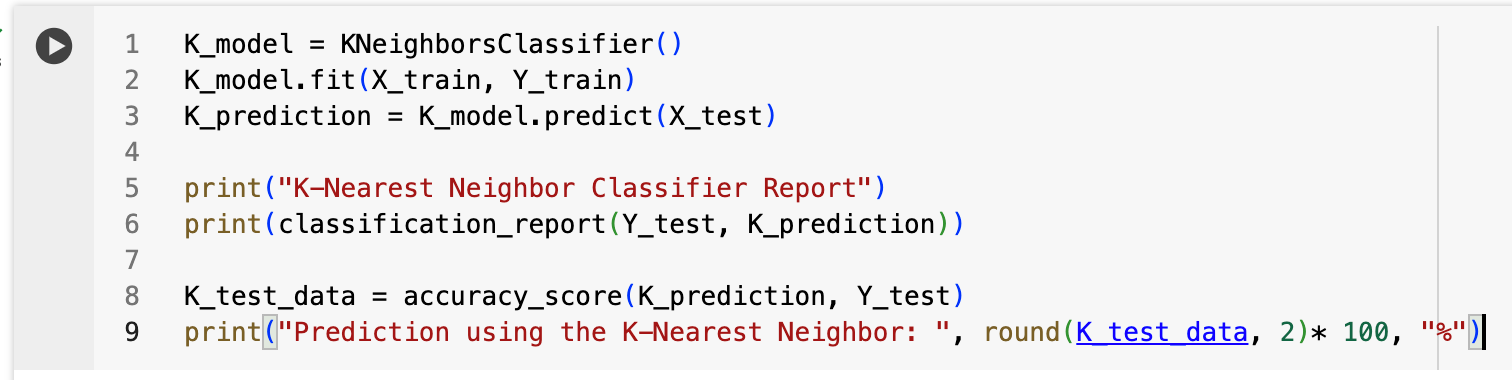
The prediction testing out for this method are



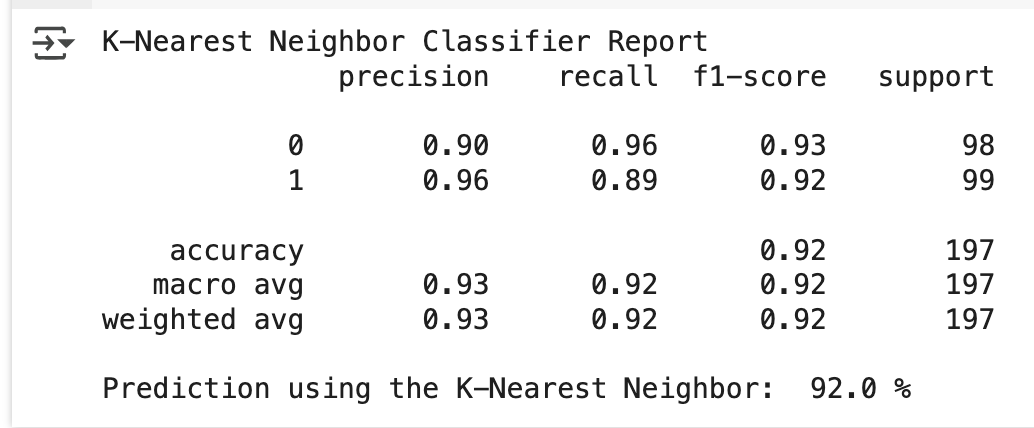
* 1. **Using K-Nearest Neighbors Classifier**

K-nearest neighbor is one of the simplest (and in some cases effective) supervised-learning algorithms in Machine Learning. During training, the algorithm does not learn anything from the training data (which is why it is classified as lazy learning )

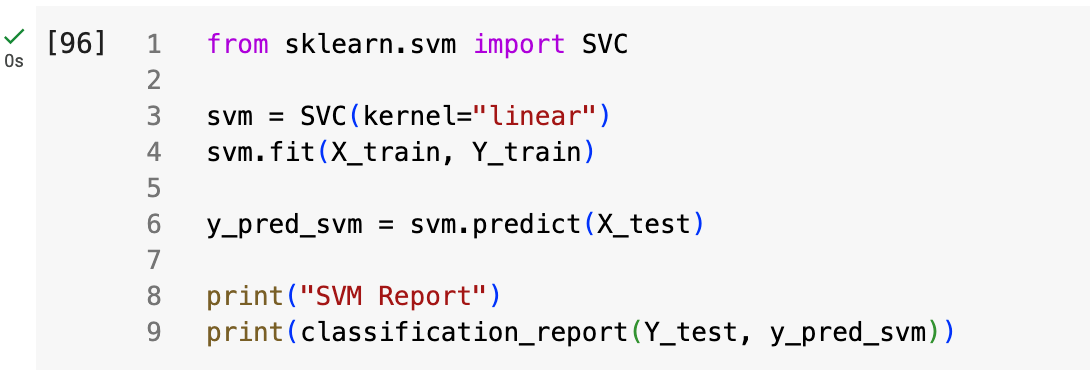
For this the code below representing this method



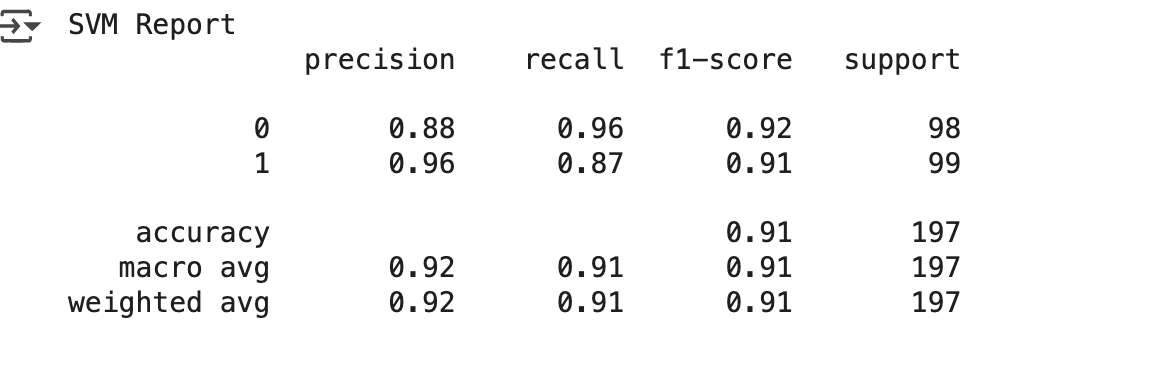
And the Result is:



* 1. **Using Support Vector Machine**

****

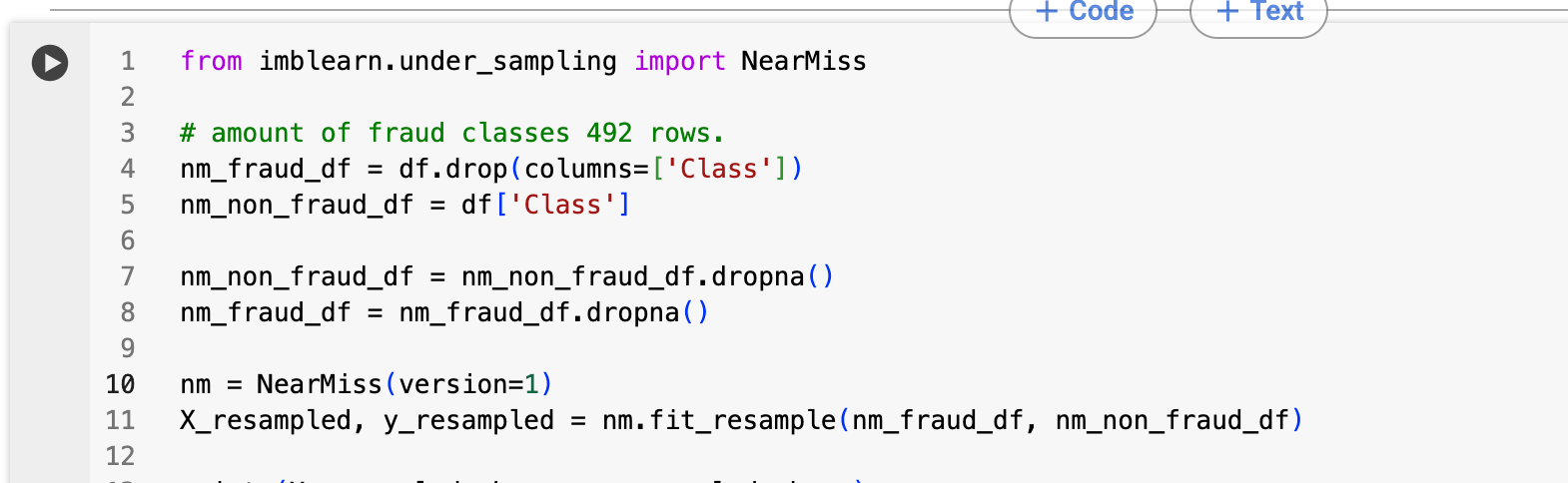
And the result for using SVM are



The test for the SVM are in the relatively in 92%. After using the RUS Sample

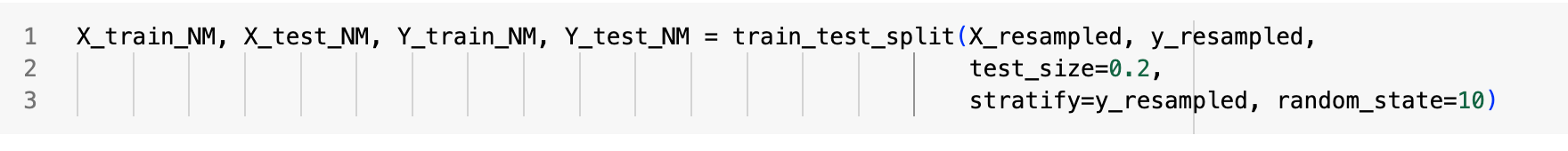
* 1. **Using NearMiss Under Sampling Method**
     1. **Train Test Split**

Beside of using the RUS, we have the method of using the Near Miss as well, the code below are the sampling of using the NearMiss

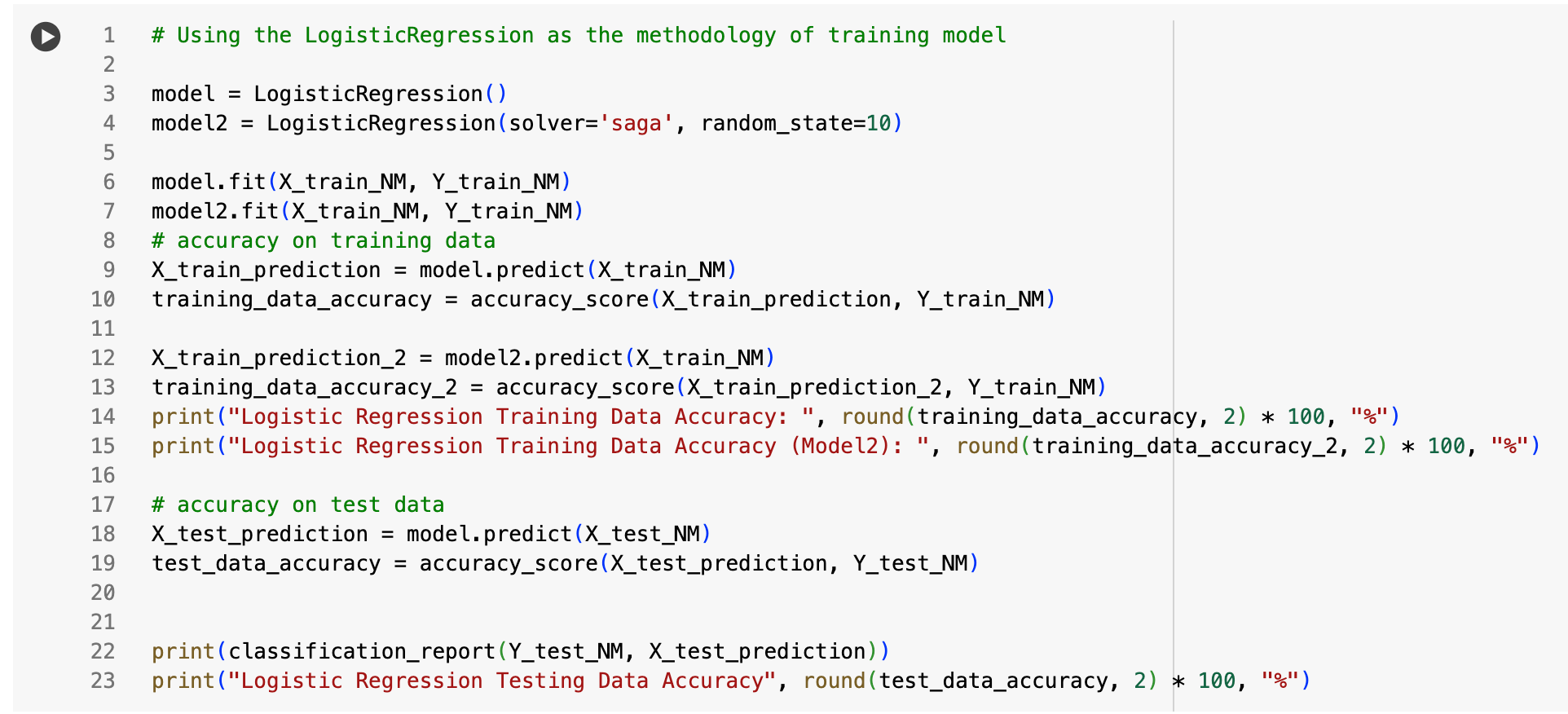


We will process as well with the droping NaN section of the dataset

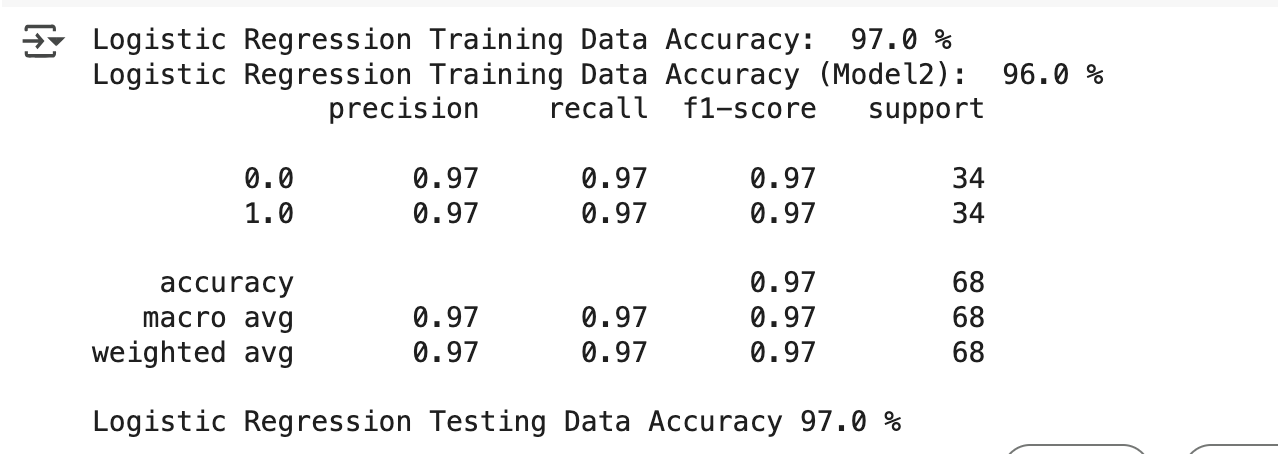
After that, we’ll train-test-split with this new dataset



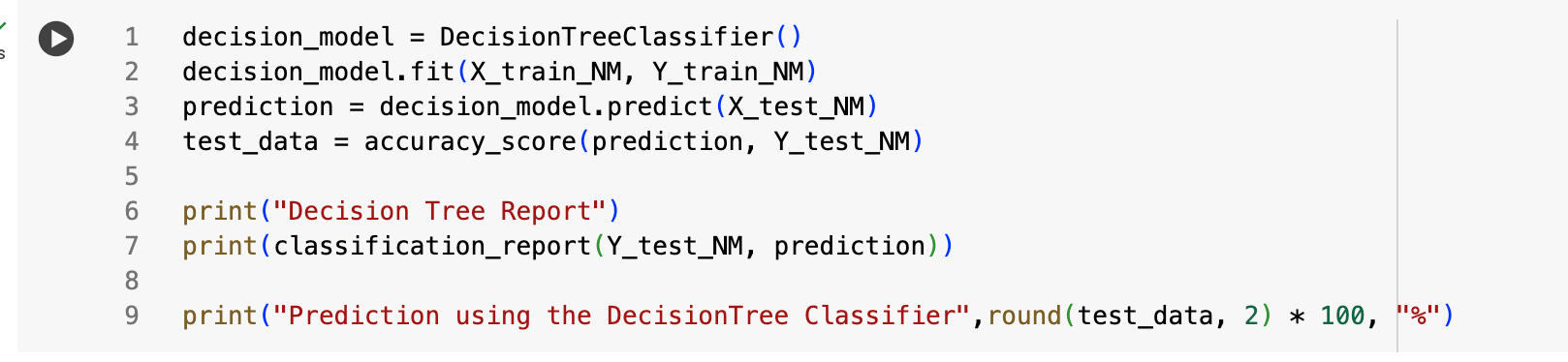
* 1. **Using LogisticRegression Model**

****

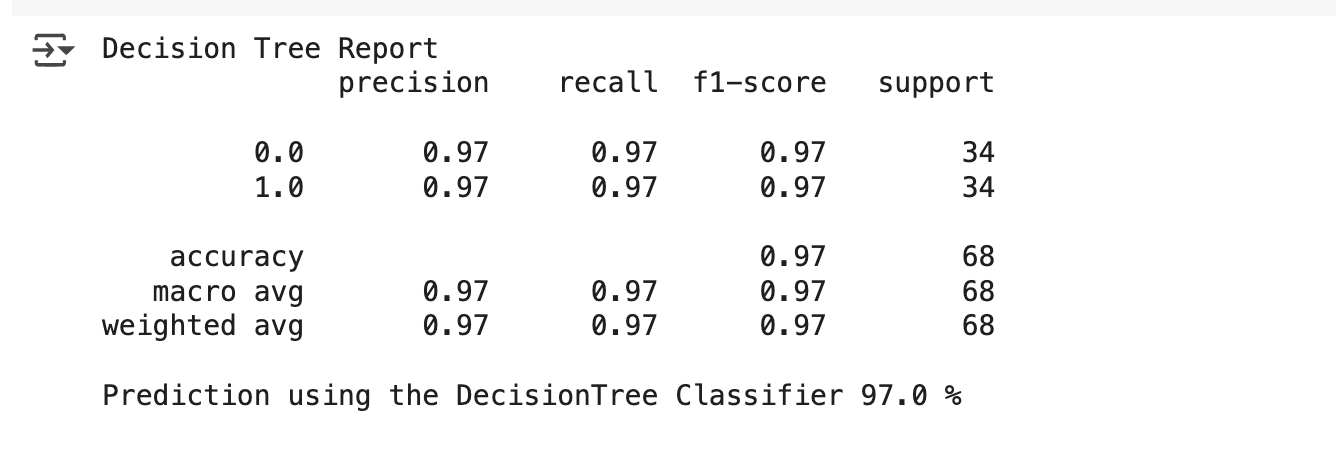
After running the model, we can see the report below



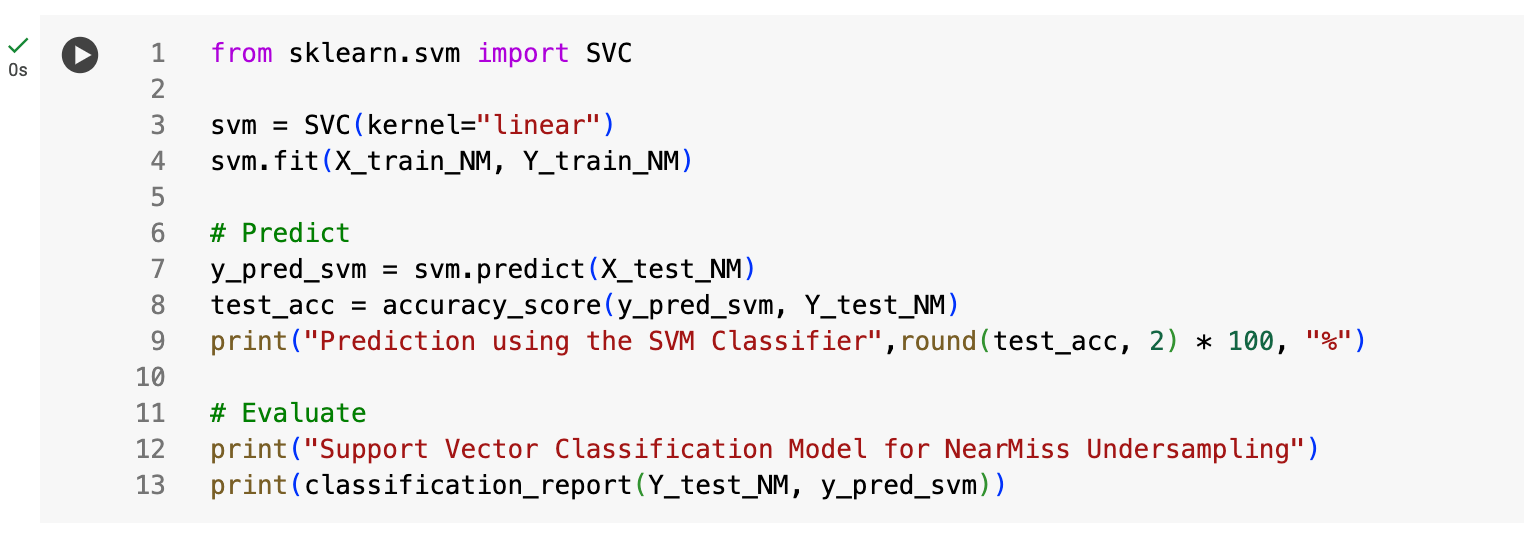
**4.10 Using Decision Tree Classifier**

****

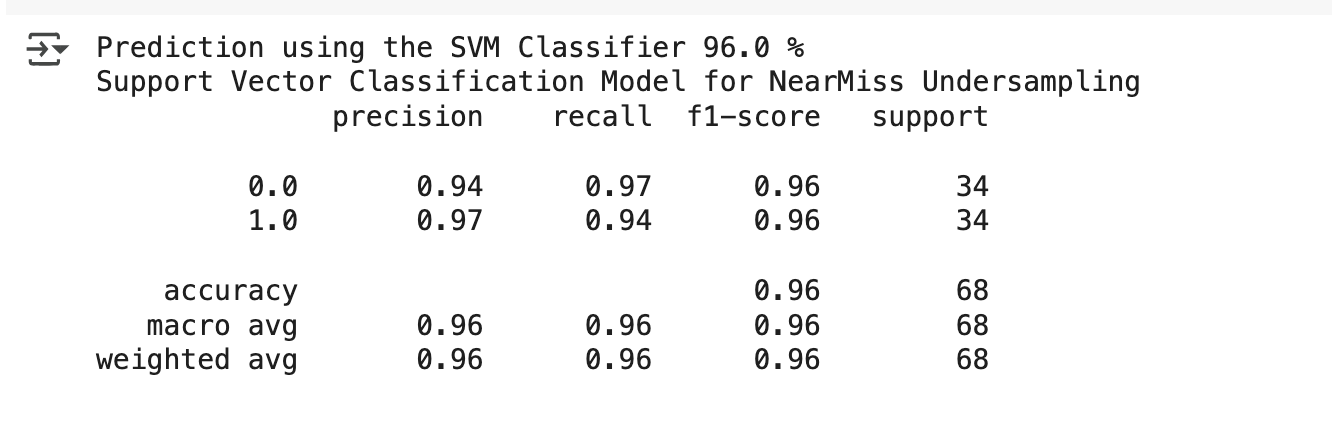
And the result are

****

**4.11 Using the SVM**



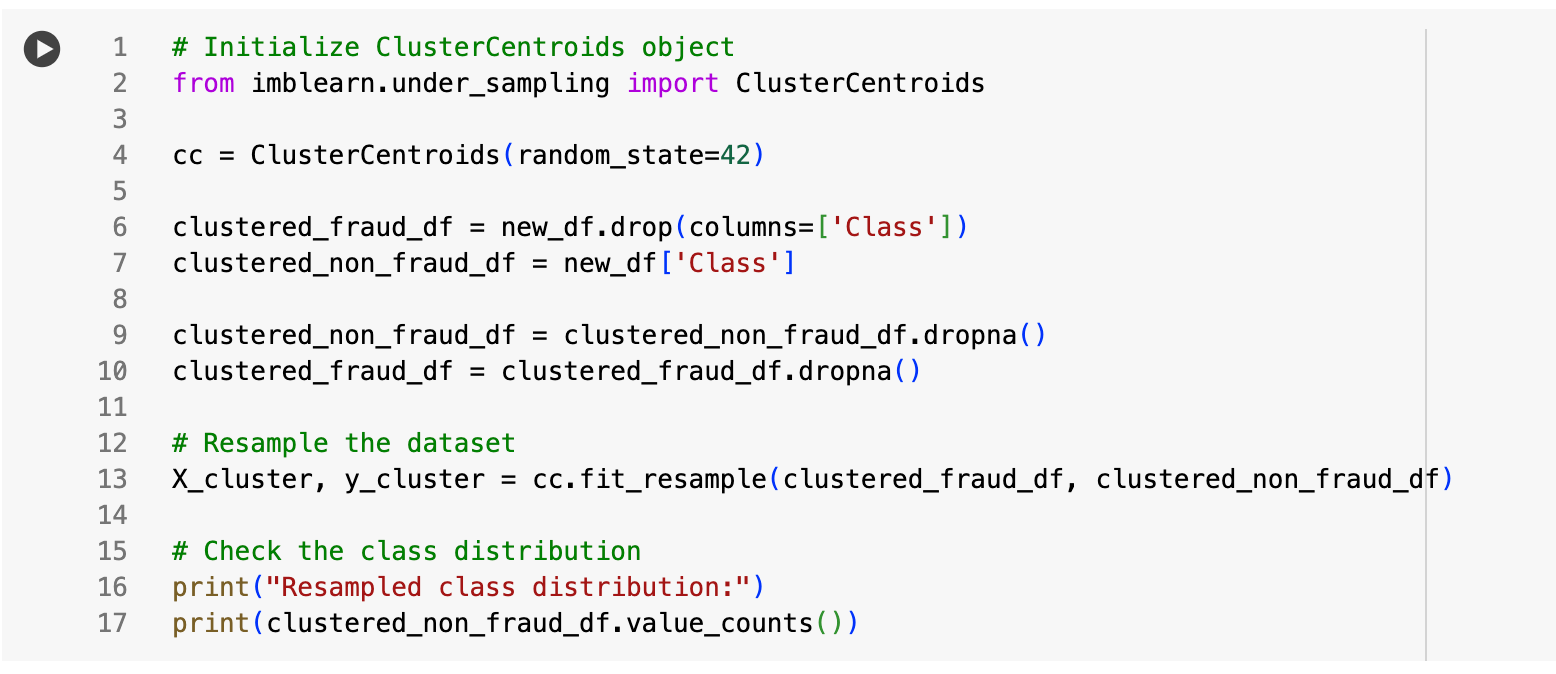
And the result are:



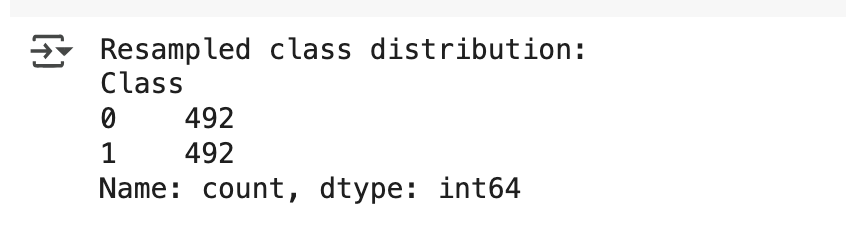
**4.12 Using Clustered Centroid**

* + 1. **Train test split**

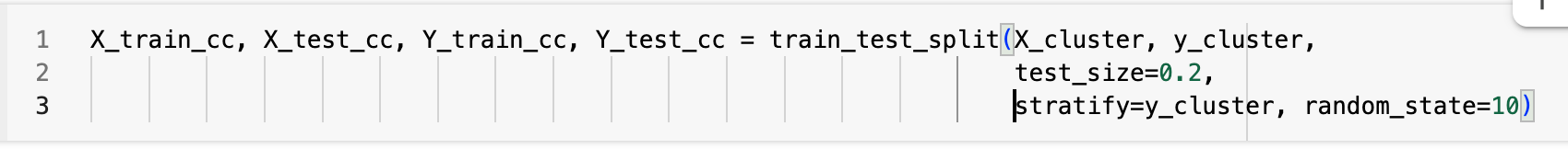
This section will use the Clustered Centroid method for undersampling

****

And after sampling we get the new dataset as well:

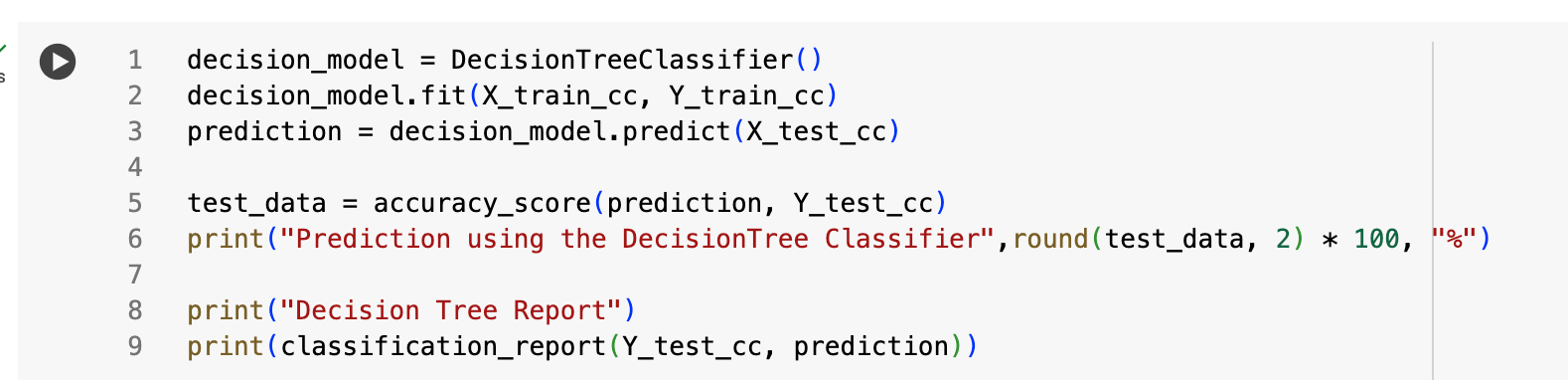


After that we will train-test-split with the dataset

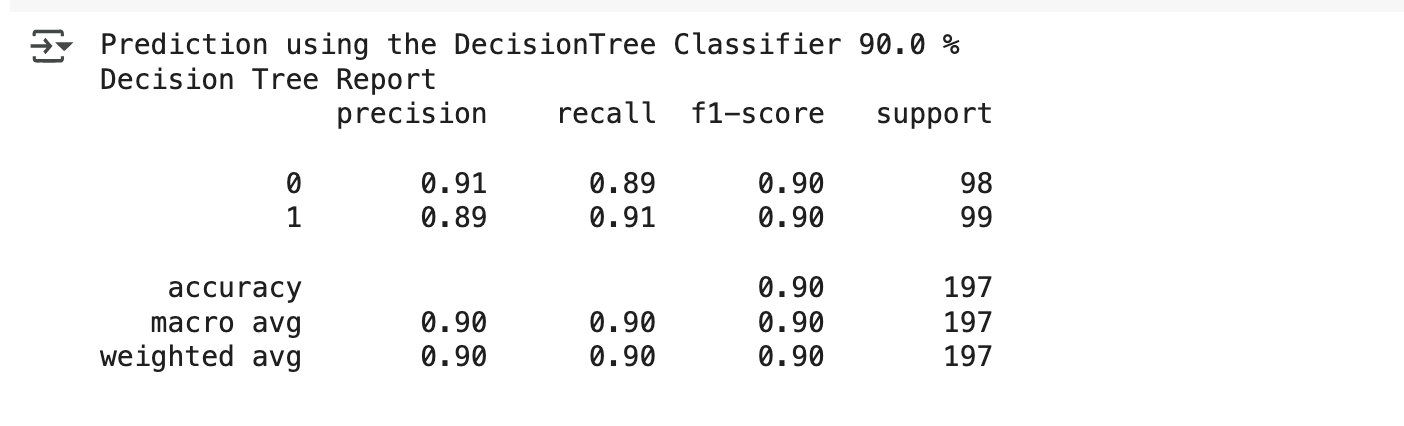


* 1. **Using Decision Tree Model**

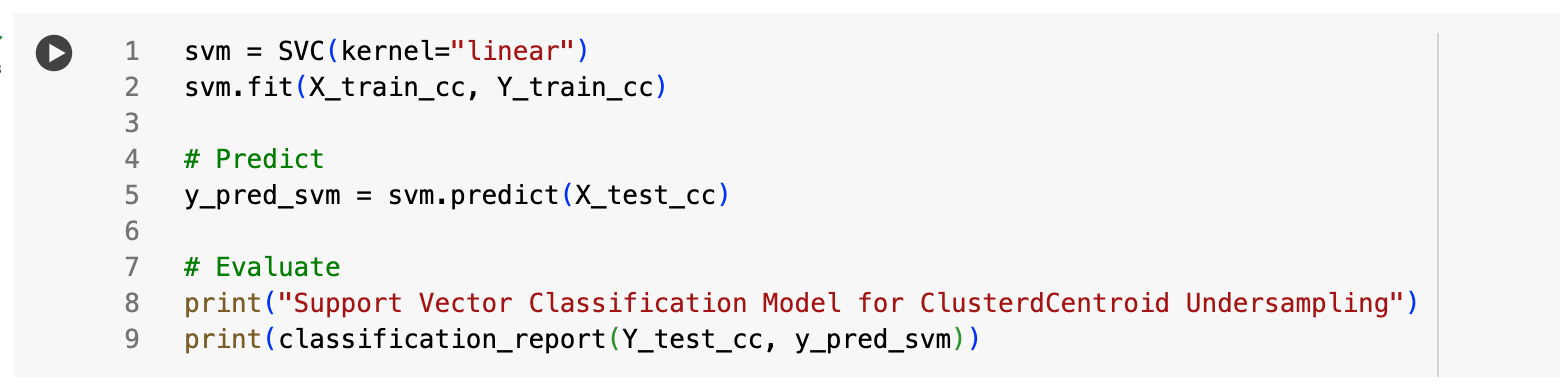
The code below show the function to use this method:



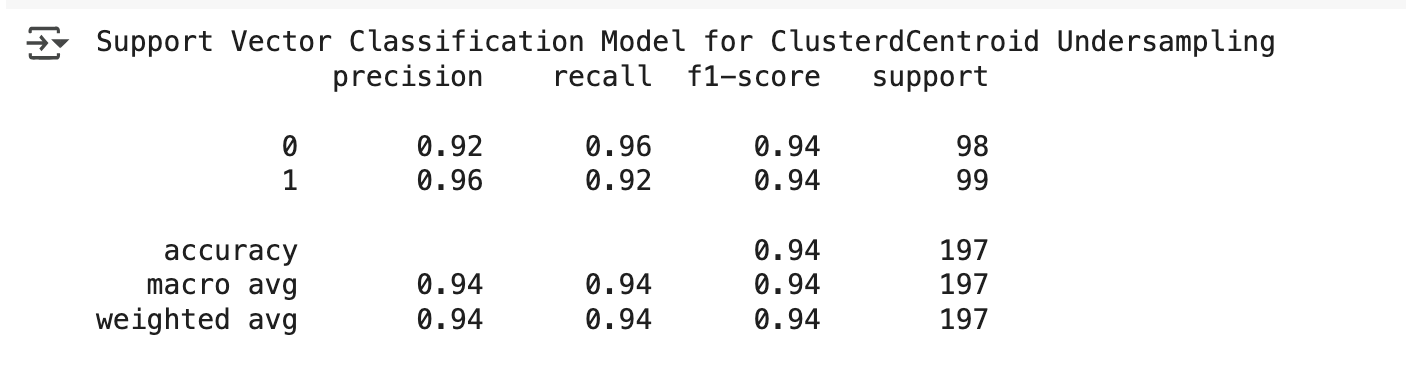
And the result are



**4.14 Using SVM**



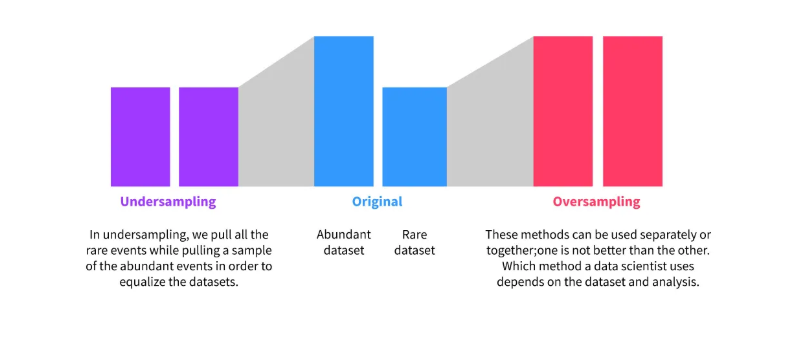
And the result are

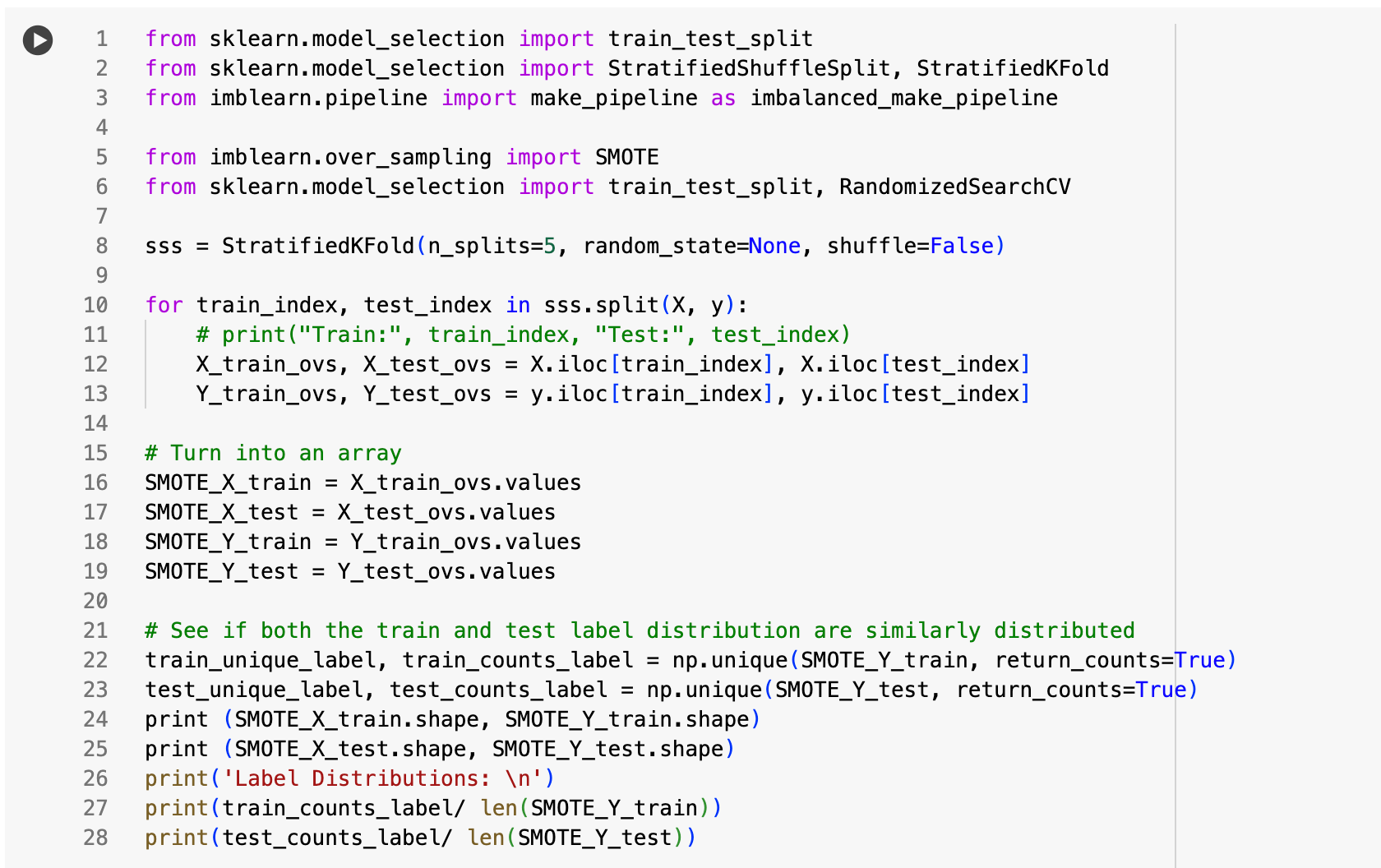


**CHAPTER 5: UNDERSTANDING THE OVER SAMPLING METHOD**

5.1 **Using traditional SMOTE approach**

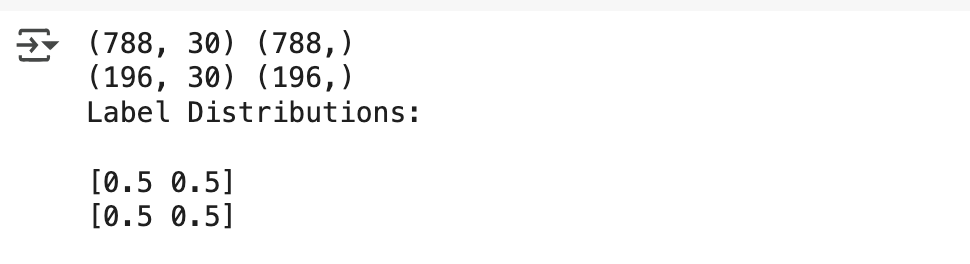
The dataset we work imbalanced, because of the inequality of the fraudalent - non-fraudulent. We need to Scale the dataset to, with SMOTE (to balanced out the imbalanced one), by multiplying the fraudent one to match the scale of the non-fraudulent. Oversampling in here are using the SMOTE, ADASYN





* 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.

The result of using SMOTE Oversampling are



* **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.

**After declare the data validation using SMOTE, the code will train and put as the accuracy are below**

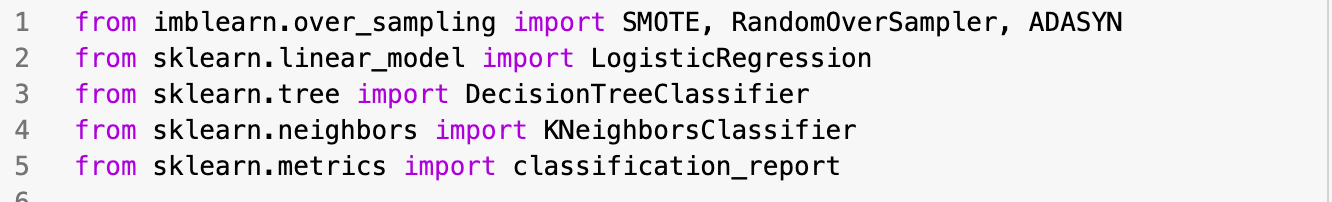
****

**And the result are:**

****

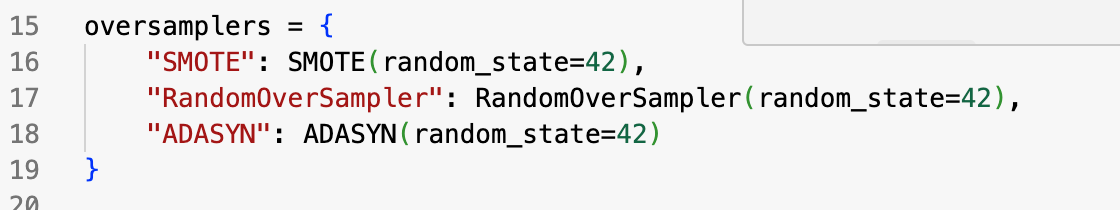
**5.2 Alternative way to implement**

**The alternative way is to use as an array method with library to use the Oversampling method**

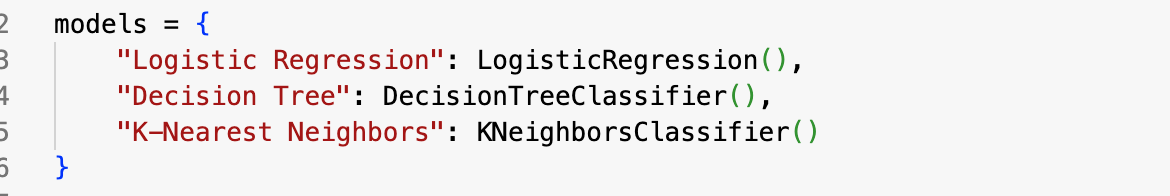
****

*Import all the library for the oversampling the method*

After import all the method sampling, the next step is to clustered in an object:



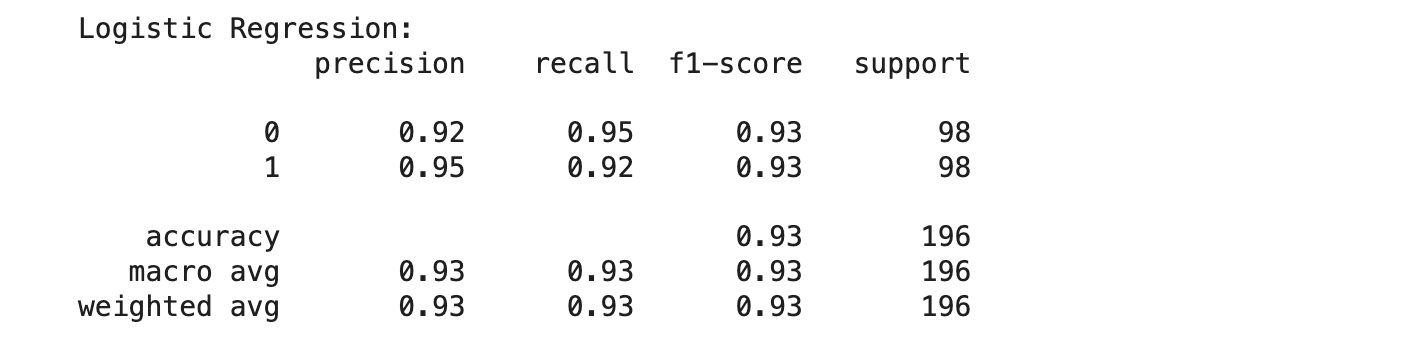
And put aside with the modeler:



From this one, we’ll import all with a for loop and run those model to get the classification report

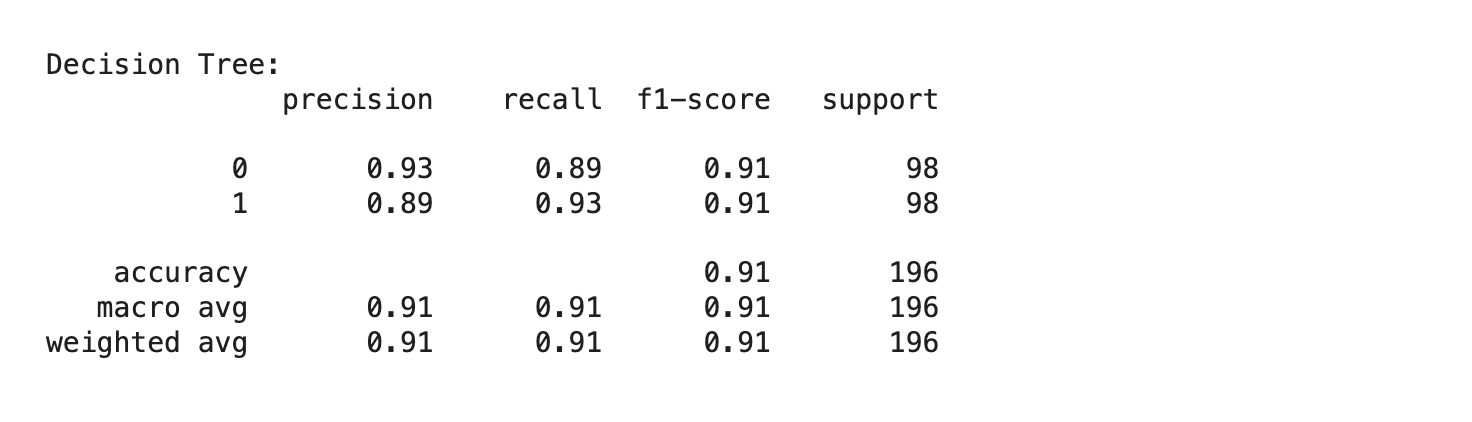
* 1. **SMOTE Oversampling Method**

5.3.1 Result Logistic Regression with SMOTE Oversampling:



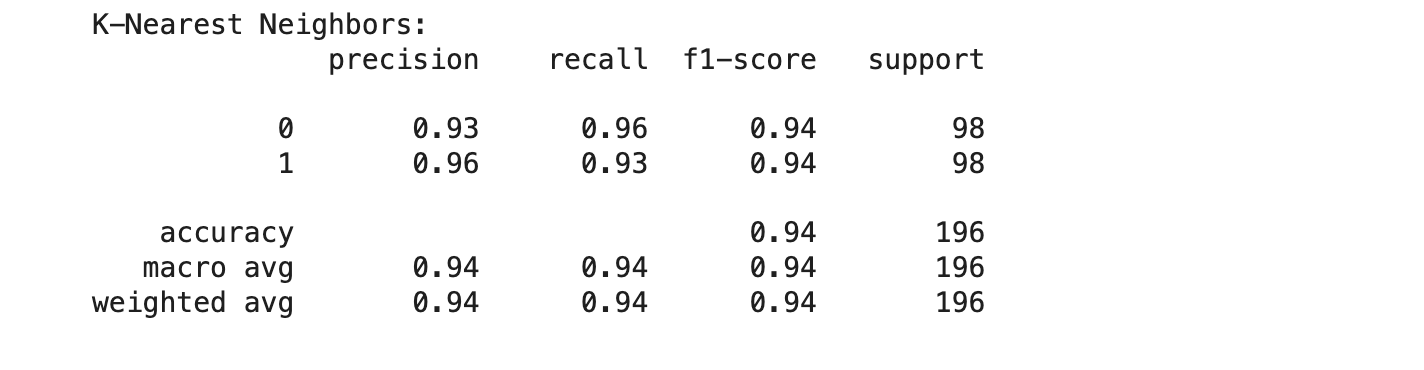
***The result for using the Logistic Regression are 93%***

* + 1. Result Decision Tree with SMOTE Oversampling



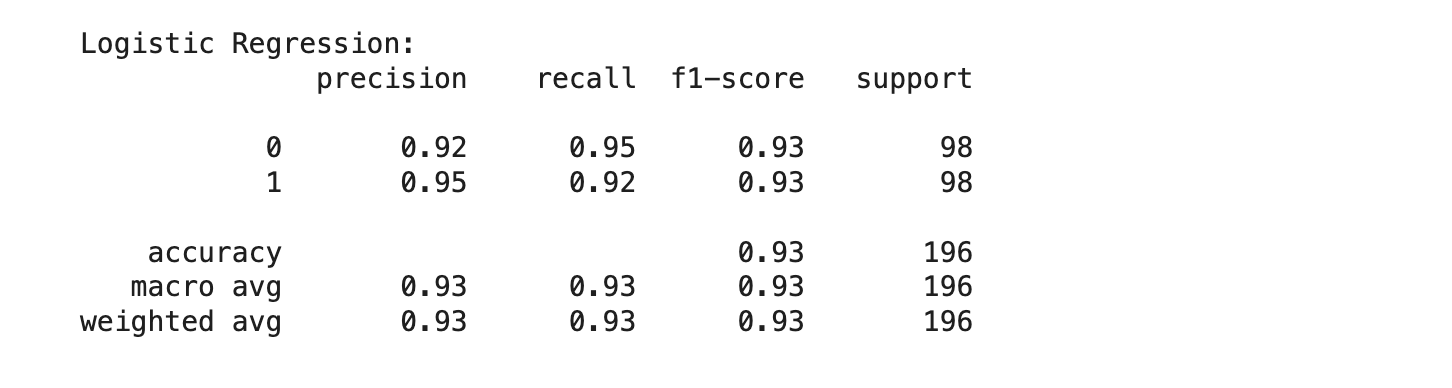
***The result for using the Decision Tree are 91%***

* + 1. Result K-Nearest Neighbors with SMOTE Oversampling

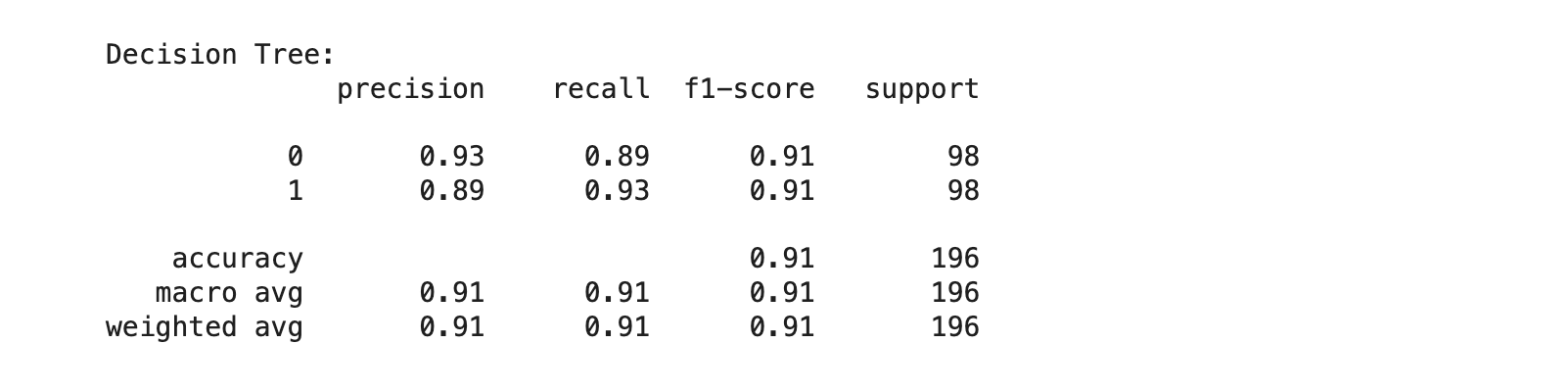


***The result for using the K-Nearest Neighbors are 94%***

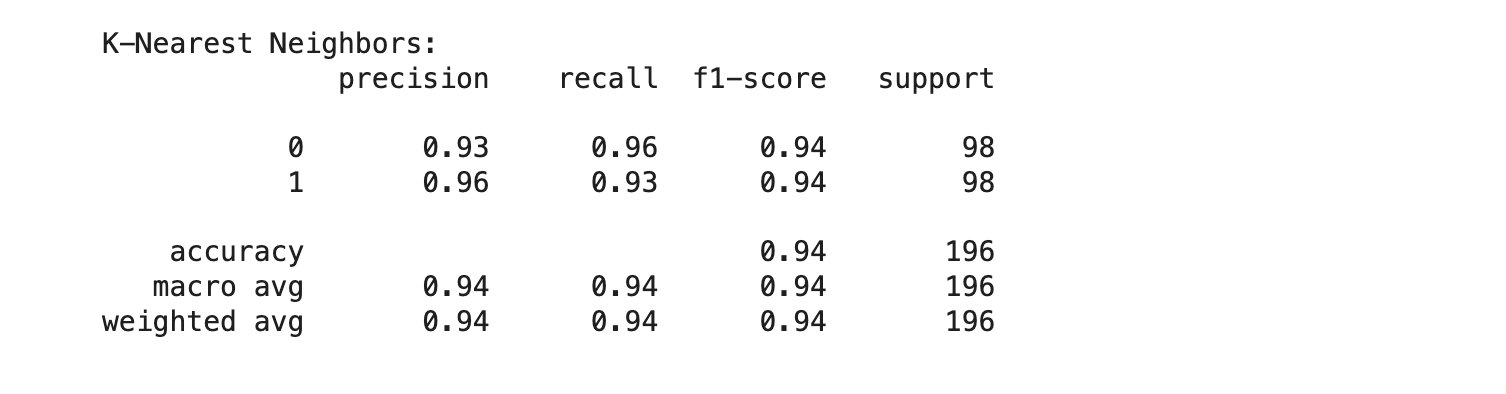
* 1. **Using Random Over-sampler Method (ROS)**
     1. Result Logistic Regression with ROS:

******

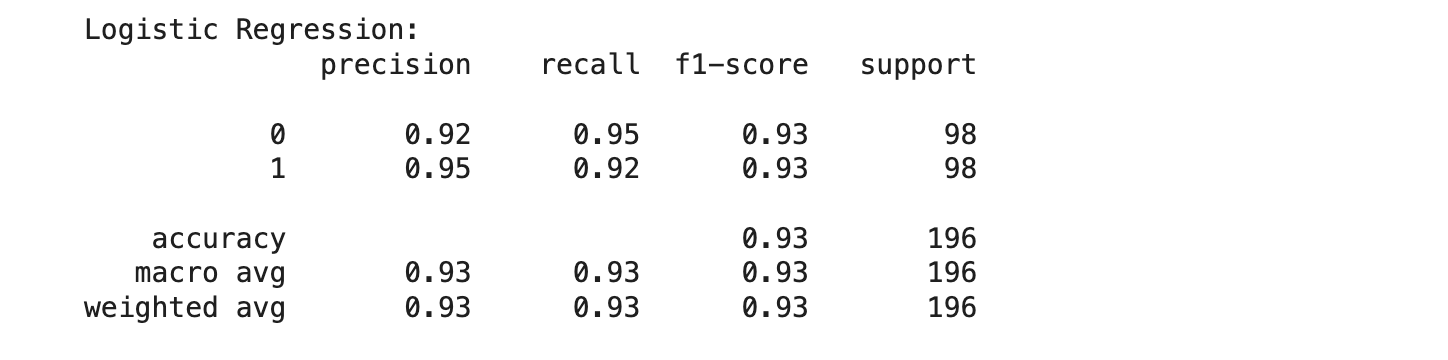
* + 1. Result with Decision Tree with ROS



* + 1. Result with K-Nearest Neighbors with ROS:



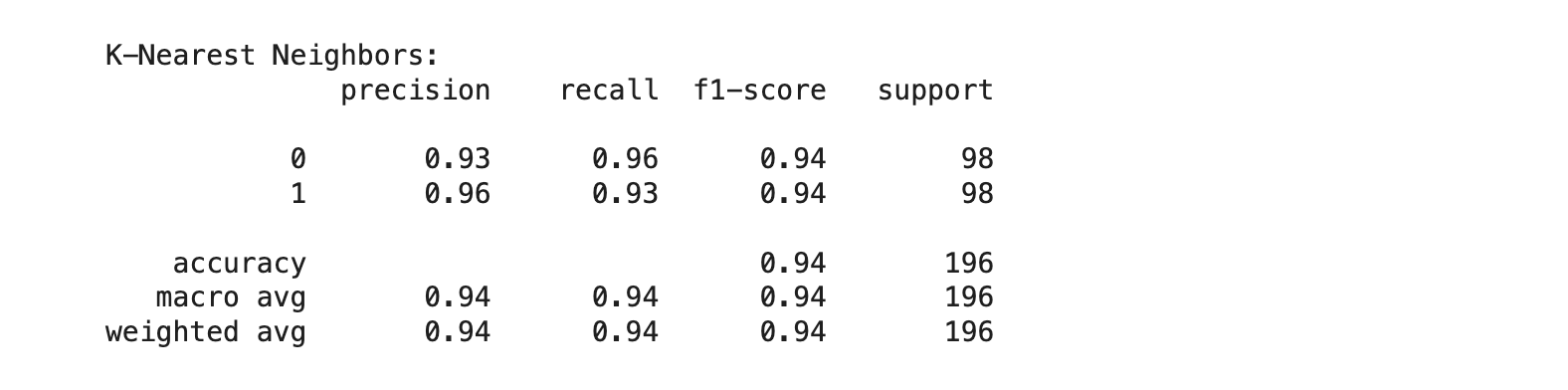
* 1. Using ADASYN:
     1. Result Logistic Regression with ADASYN:

******

* + 1. Result Decision Tree Method with ADASYN



* + 1. Result K-Nearest Neighbors with ADASYN:

******

**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>