TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

LAM NGOC HAI – 518H0347

PHAN AN DUY - 518H0616

Code: ĐA2-08

FRAUD DETECTION CREDIT CARD WITH IMBALANCED DATA

INFORMATION TECHNOLOGY PROJECT 2
COMPUTER SCIENCE

HO CHI MINH , YEAR 2025

TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

LAM NGOC HAI – 518H0347

PHAN AN DUY - 518H0616

Code: ĐA2-08

FRAUD DETECTION CREDIT CARD 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)

LAM NGOC HAI

PHAN AN DUY

This thesis was carried out at Ton Duc Thang University.

Advisor: ....

....

(Title, full name and signature)

This is defended at the Undergraduate Thesis Examination Committee was hold at Ton Duc Thang University on … /…/……

Confirmation of the Chairman of the Undergraduate Thesis Examination Committee and the Dean of the faculty after receiving the modified thesis (if any).

CHAIRMAN	**DEAN OF FACULTY**
…………………………..	**……………………………….**

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.

2. **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.

3. **Ensemble Models:**

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

4. **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.

5. **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.

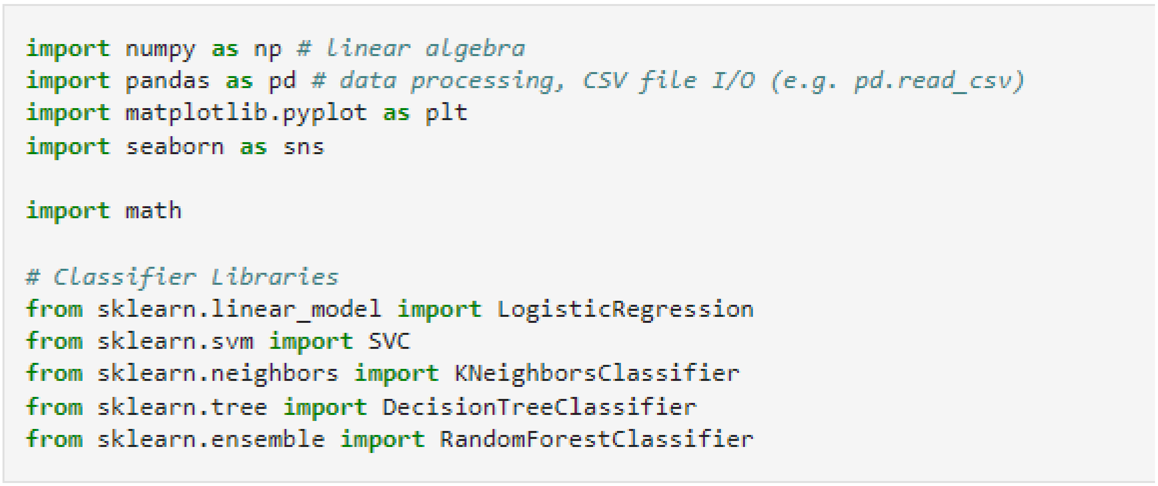
Conclusion

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

LIST OF FIGURES

1 Import Libraries:



```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

import math

# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

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.

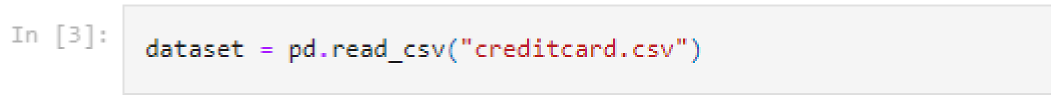
2 Understanding our data:

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)

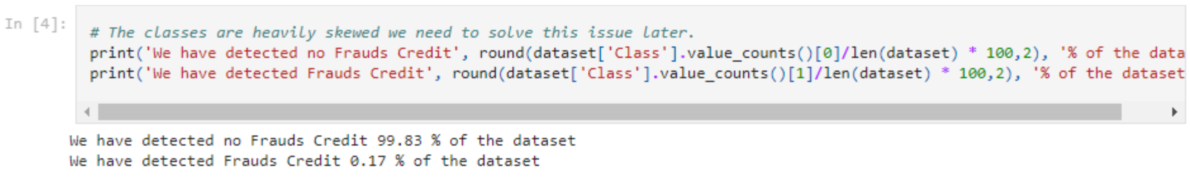
To read dataset we use this command



```
In [3]:  dataset = pd.read_csv("creditcard.csv")
```

3 Check The Data:

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



```
In [4]:  # The classes are heavily skewed we need to solve this issue later.
         print('We have detected no Frauds Credit', round(dataset['Class'].value_counts()[0]/len(dataset) * 100,2), '% of the data
         print('We have detected Frauds Credit', round(dataset['Class'].value_counts()[1]/len(dataset) * 100,2), '% of the dataset

         We have detected no Frauds Credit 99.83 % of the dataset
         We have detected Frauds Credit 0.17 % of the dataset
```

Printing Non-Fraudulent Transactions:

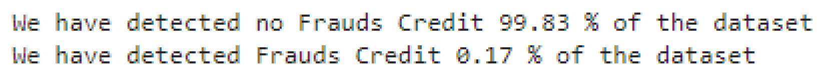
- dataset['Class'].value_counts(): This function counts the occurrences of each class in the 'Class' column of the dataset. It returns a Series with the count of each class (0 for non-fraudulent and 1 for fraudulent).
- value_counts()[0]: This retrieves the count of non-fraudulent transactions (class 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:



```
We have detected no Frauds Credit 99.83 % of the dataset
We have detected Frauds Credit 0.17 % of the dataset
```

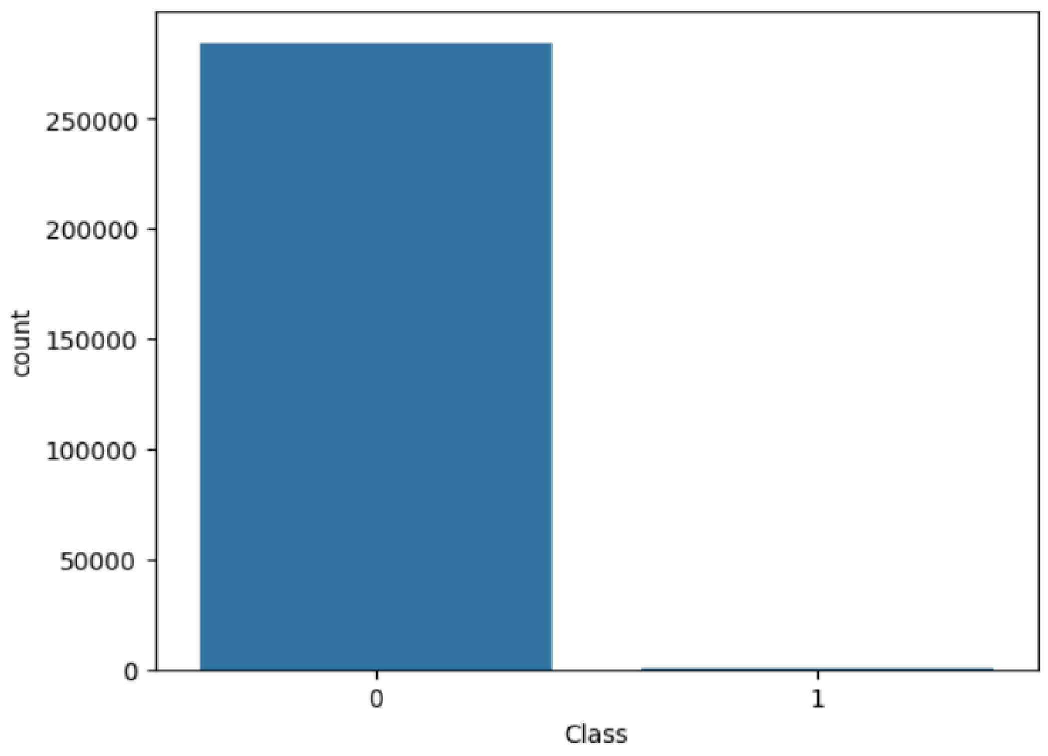
4 Glimpse the data:

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

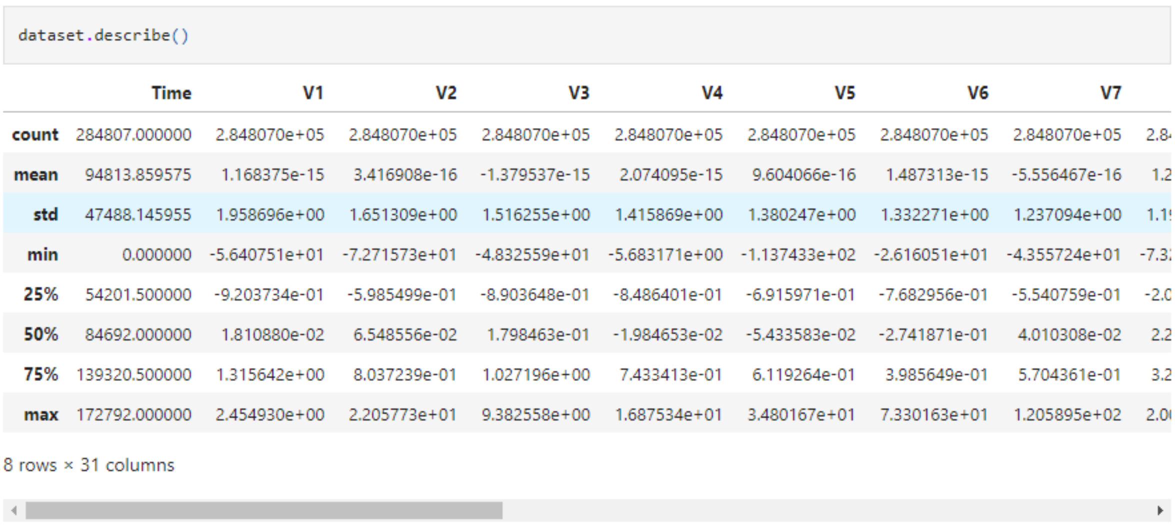
Definition: Because most of the transaction need to be secure during all the transaction, with many transaction need to be secure before can other transaction can be made. So the value contain the fraud only 50% of the all the transaction made it. The dataset are validate to use as a research purposeAutomotive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

```
In [5]:
sns.countplot(x='Class', data=df)
```

```
Out[5]:
<Axes: xlabel='Class', ylabel='count'>
```



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.

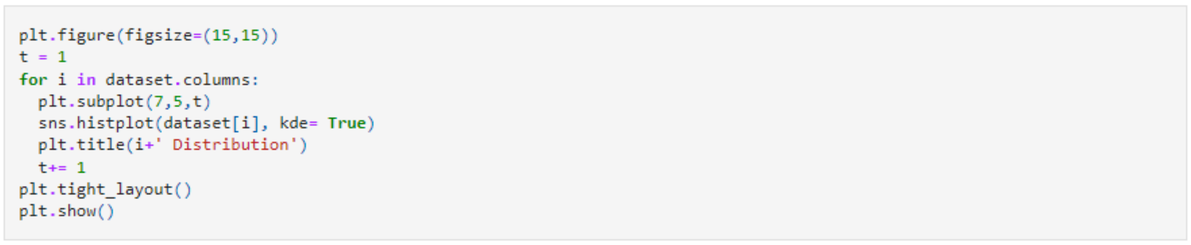


```
dataset.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.2
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.1
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.3
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.2
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.2
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.0

8 rows × 31 columns

5 Result Meaning


```
plt.figure(figsize=(15,15))
t = 1
for i in dataset.columns:
  plt.subplot(7,5,t)
  sns.histplot(dataset[i], kde= True)
  plt.title(i+' Distribution')
  t+= 1
plt.tight_layout()
plt.show()
```

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.

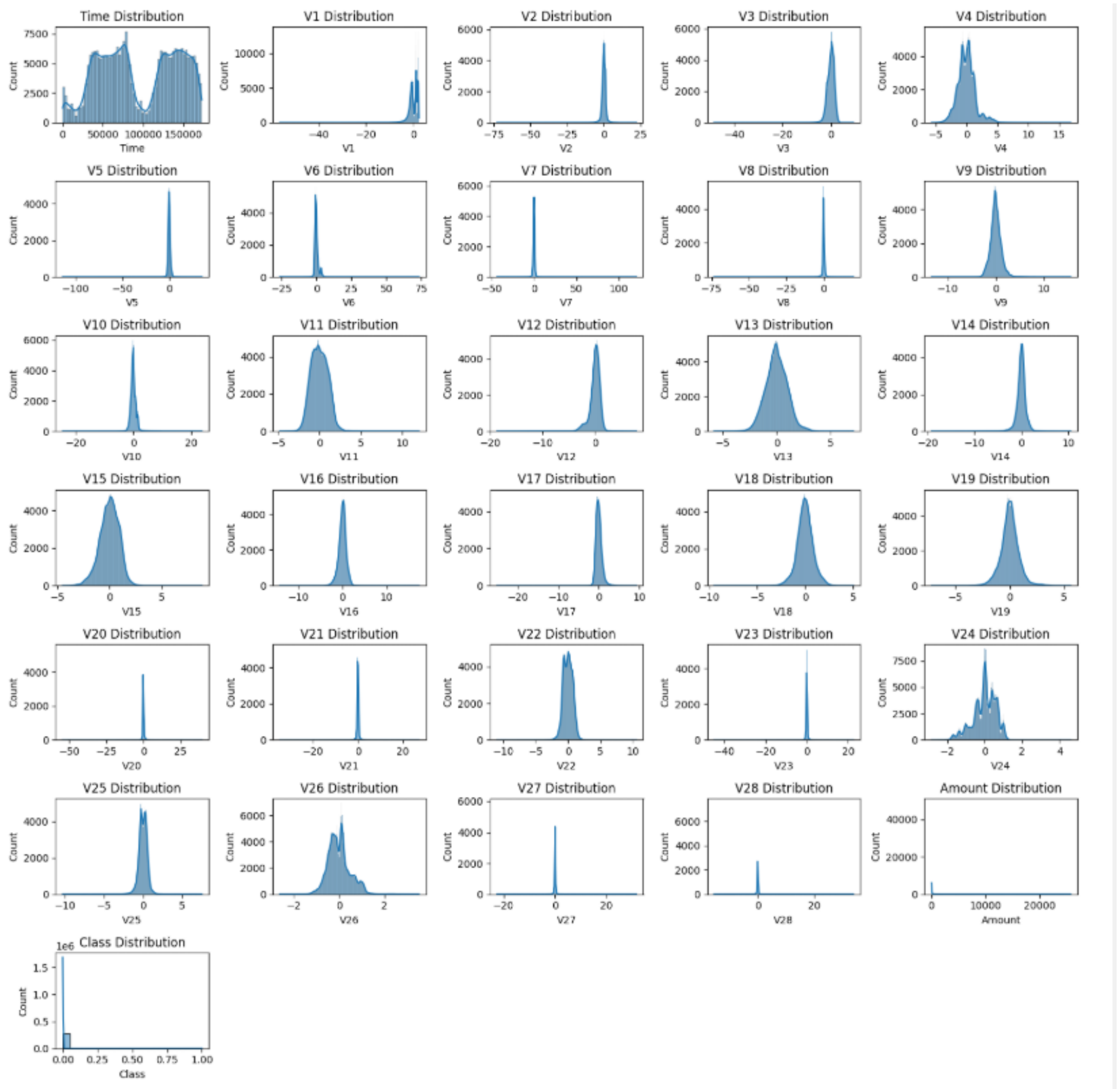
  - kde=True adds a Kernel Density Estimate (KDE) curve to the histogram, which represents the data's probability density function.

- 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.



6 Data Preprocessing

In [8]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
df['Amount'] = sc.fit_transform(pd.DataFrame(df['Amount']))
```

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.

In [9]:

```python
from sklearn.preprocessing import RobustScaler

# RobustScaler is less prone to outliers.

std_scaler = StandardScaler()
rob_scaler = RobustScaler()

df['scaled_amount'] = rob_scaler.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rob_scaler.fit_transform(df['Time'].values.reshape(-1,1))

df.drop(['Time','Amount'], axis=1, inplace=True)
```

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.

7 Undersampling The Dataset

Undersampling the Dataset

In [10]:

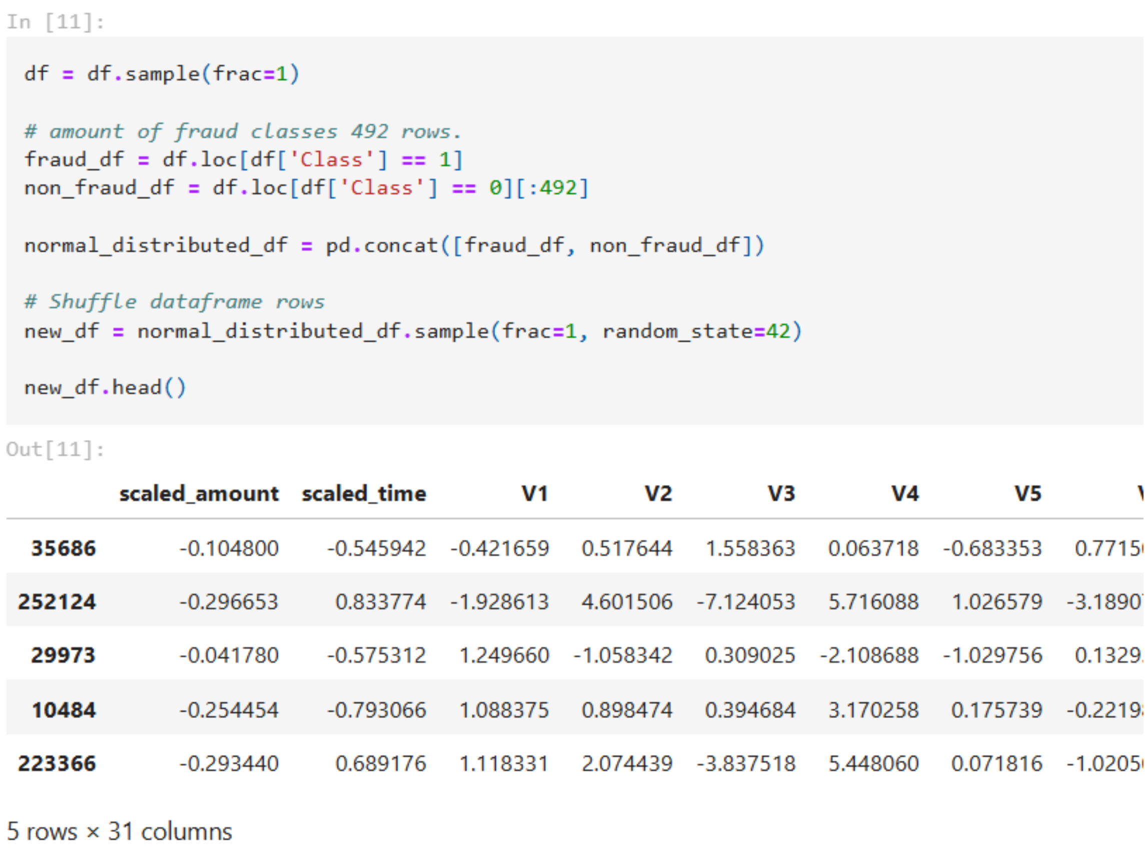
```python
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']

df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(1, 'scaled_time', scaled_time)

# Amount and Time are Scaled!

df.head()
```

Out[10]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.

5 rows × 31 columns

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.



```
In [11]:
df = df.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)

new_df.head()
```

Out[11]:

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	
35686	-0.104800	-0.545942	-0.421659	0.517644	1.558363	0.063718	-0.683353	0.7715
252124	-0.296653	0.833774	-1.928613	4.601506	-7.124053	5.716088	1.026579	-3.1890
29973	-0.041780	-0.575312	1.249660	-1.058342	0.309025	-2.108688	-1.029756	0.1329
10484	-0.254454	-0.793066	1.088375	0.898474	0.394684	3.170258	0.175739	-0.2219
223366	-0.293440	0.689176	1.118331	2.074439	-3.837518	5.448060	0.071816	-1.0205

5 rows × 31 columns

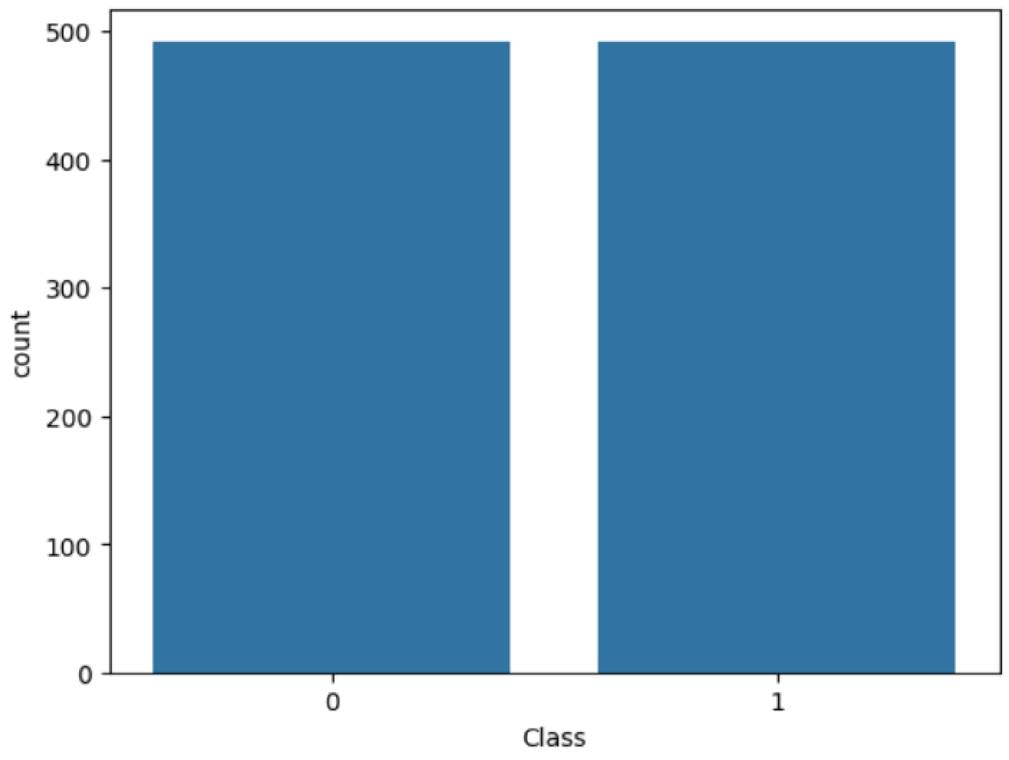
This code creates a balanced dataset with an equal number of fraud and non-fraud samples, concatenates them into a single DataFrame, shuffles the rows to ensure randomness, and then displays the first few rows of the shuffled DataFrame.

In [12]:

```
# Finishing undersampling the dataset
sns.countplot(x='Class', data=new_df)
```

Out[12]:

```
<Axes: xlabel='Class', ylabel='count'>
```



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.

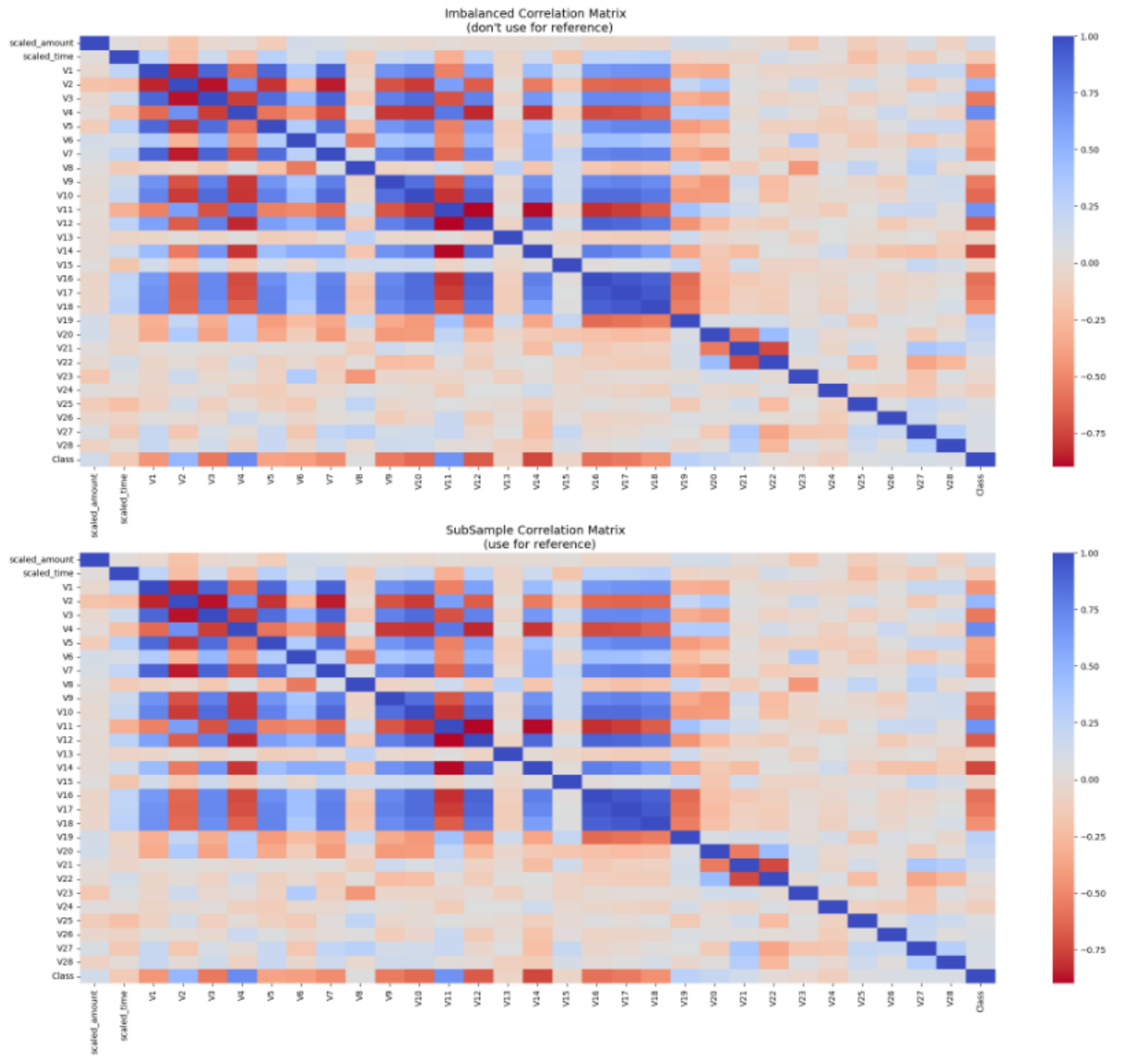
In [13]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
for i in new_df.columns:
  if i != 'Class':
    new_df[i] = scaler.fit_transform(new_df[[i]])
```

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.

```
In [16]:
# Make sure we use the subsample in our correlation

f, (ax1, ax2) = plt.subplots(2, 1, figsize=(24,20))

# Entire DataFrame
corr = new_df.corr()
sns.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax1)
ax1.set_title("Imbalanced Correlation Matrix \n (don't use for reference)", fontsize=14)


sub_sample_corr = new_df.corr()
sns.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=ax2)
ax2.set_title('SubSample Correlation Matrix \n (use for reference)', fontsize=14)
plt.show()
```

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.



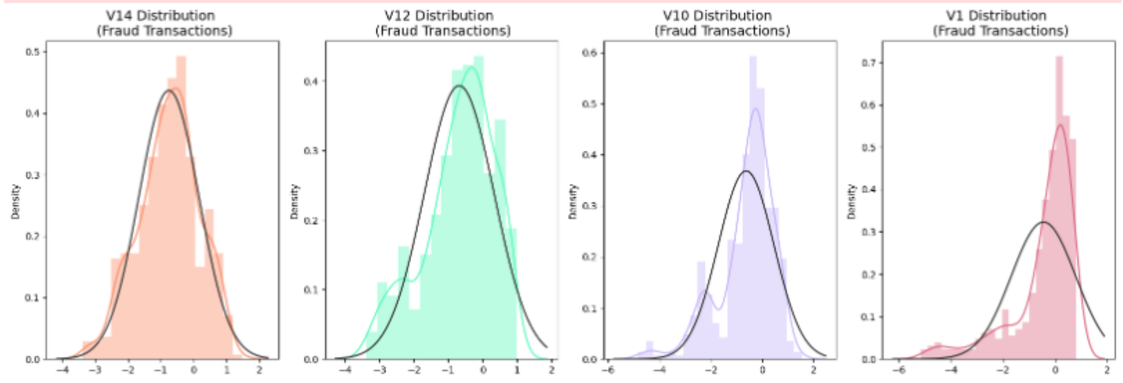
In [17]:

```
from scipy.stats import norm

f, (ax1, ax2, ax3, ax4) = plt.subplots(1,4, figsize=(20, 6))

v14_fraud_dist = new_df['V14'].loc[new_df['Class'] == 1].values
sns.distplot(v14_fraud_dist,ax=ax1, fit=norm, color='#FB8861')
ax1.set_title('V14 Distribution \n (Fraud Transactions)', fontsize=14)

v12_fraud_dist = new_df['V12'].loc[new_df['Class'] == 1].values
sns.distplot(v12_fraud_dist,ax=ax2, fit=norm, color='#56F9BB')
ax2.set_title('V12 Distribution \n (Fraud Transactions)', fontsize=14)


v10_fraud_dist = new_df['V10'].loc[new_df['Class'] == 1].values
sns.distplot(v10_fraud_dist,ax=ax3, fit=norm, color='#C5B3F9')
ax3.set_title('V10 Distribution \n (Fraud Transactions)', fontsize=14)

v1_fraud_dist = new_df['V1'].loc[new_df['Class'] == 1].values
sns.distplot(v1_fraud_dist,ax=ax4, fit=norm, color='#DA627D')
ax4.set_title('V1 Distribution \n (Fraud Transactions)', fontsize=14)

plt.show()
```

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.

```
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_23148/2779172976.py:6: UserWarn
ing:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v14_fraud_dist,ax=ax1, fit=norm, color='#FB8861')
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_23148/2779172976.py:10: UserWar
ning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v12_fraud_dist,ax=ax2, fit=norm, color='#56F9BB')
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_23148/2779172976.py:15: UserWar
ning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v10_fraud_dist,ax=ax3, fit=norm, color='#C5B3F9')
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_23148/2779172976.py:19: UserWar
ning:
```

```
ning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v10_fraud_dist,ax=ax3, fit=norm, color='#C5B3F9')
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_23148/2779172976.py:19: UserWar
ning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(v1_fraud_dist,ax=ax4, fit=norm, color='#DA627D')
```



8 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

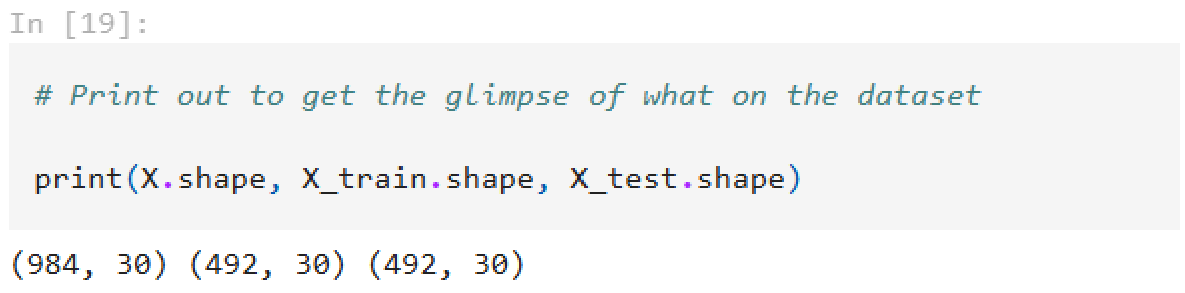
```
# Splitting the dataset.

from sklearn.model_selection import train_test_split
X = df1.drop('Class', axis = 1)
y = df1['Class']

df2 = df1.copy()
df3 = df1.copy()

# Creating a set by using the train_test_split library from python
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.5, stratify=y, random_state=10) # Create 1 set with 0
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X, y, test_size=0.75, stratify=y, random_state=1000)
```

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.

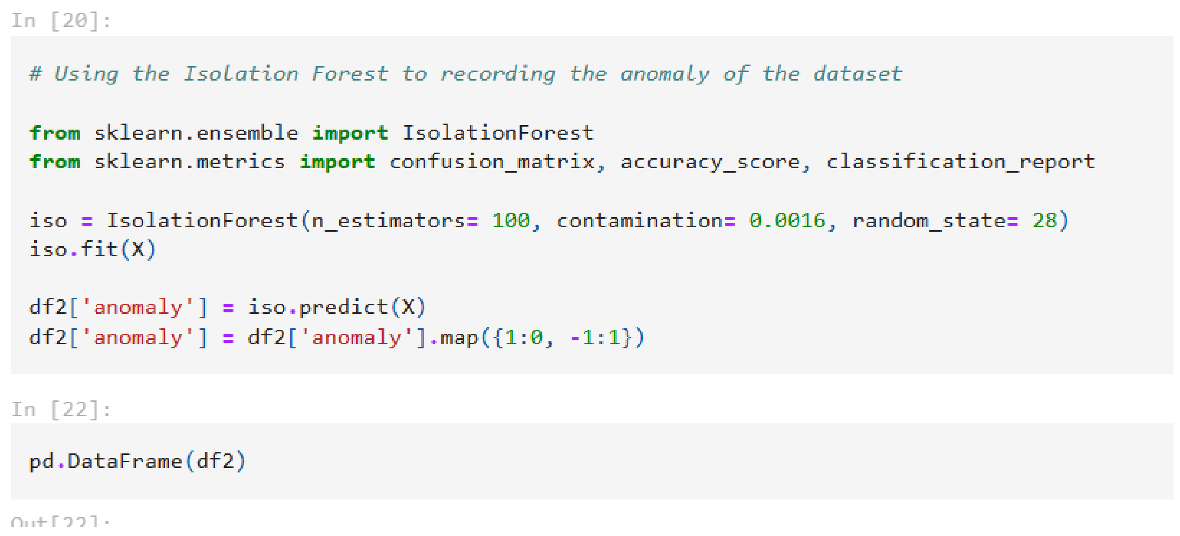
This is useful for preparing the data for model training and evaluation while ensuring that the class distribution is preserved.



```
In [19]:
# Print out to get the glimpse of what on the dataset

print(X.shape, X_train.shape, X_test.shape)

(984, 30) (492, 30) (492, 30)
```

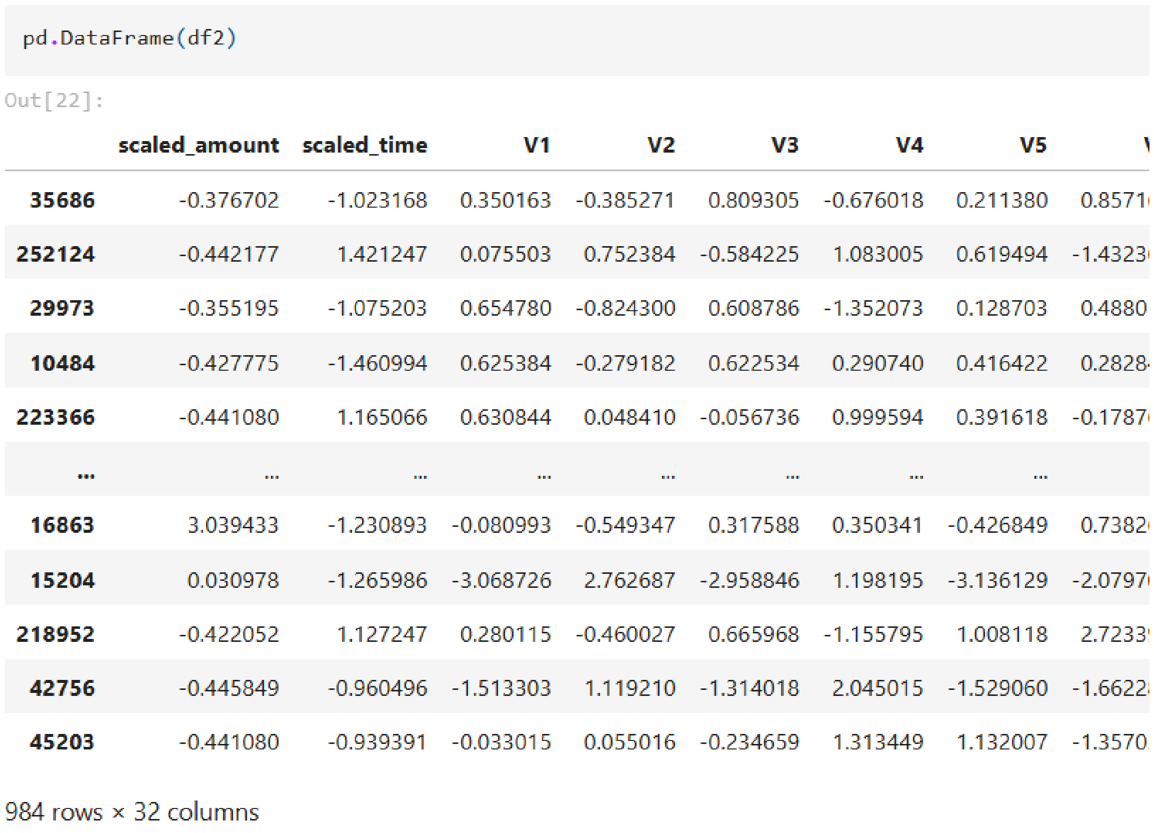
This code is helpful for understanding the dimensions of your datasets, especially when you're working with train-test splits and need to ensure they are correctly partitioned.

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



```
In [20]:
# Using the Isolation Forest to recording the anomaly of the dataset

from sklearn.ensemble import IsolationForest
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

iso = IsolationForest(n_estimators= 100, contamination= 0.0016, random_state= 28)
iso.fit(X)

df2['anomaly'] = iso.predict(X)
df2['anomaly'] = df2['anomaly'].map({1:0, -1:1})

In [22]:
pd.DataFrame(df2)

Out[22]:
```

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.



```
pd.DataFrame(df2)
```

```
Out[22]:
```

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	
35686	-0.376702	-1.023168	0.350163	-0.385271	0.809305	-0.676018	0.211380	0.8571
252124	-0.442177	1.421247	0.075503	0.752384	-0.584225	1.083005	0.619494	-1.4323
29973	-0.355195	-1.075203	0.654780	-0.824300	0.608786	-1.352073	0.128703	0.4880
10484	-0.427775	-1.460994	0.625384	-0.279182	0.622534	0.290740	0.416422	0.2828
223366	-0.441080	1.165066	0.630844	0.048410	-0.056736	0.999594	0.391618	-0.1787
...	...	...	...	...	...	...	...	
16863	3.039433	-1.230893	-0.080993	-0.549347	0.317588	0.350341	-0.426849	0.7382
15204	0.030978	-1.265986	-3.068726	2.762687	-2.958846	1.198195	-3.136129	-2.0797
218952	-0.422052	1.127247	0.280115	-0.460027	0.665968	-1.155795	1.008118	2.7233
42756	-0.445849	-0.960496	-1.513303	1.119210	-1.314018	2.045015	-1.529060	-1.6622
45203	-0.441080	-0.939391	-0.033015	0.055016	-0.234659	1.313449	1.132007	-1.3570

984 rows × 32 columns

9 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:

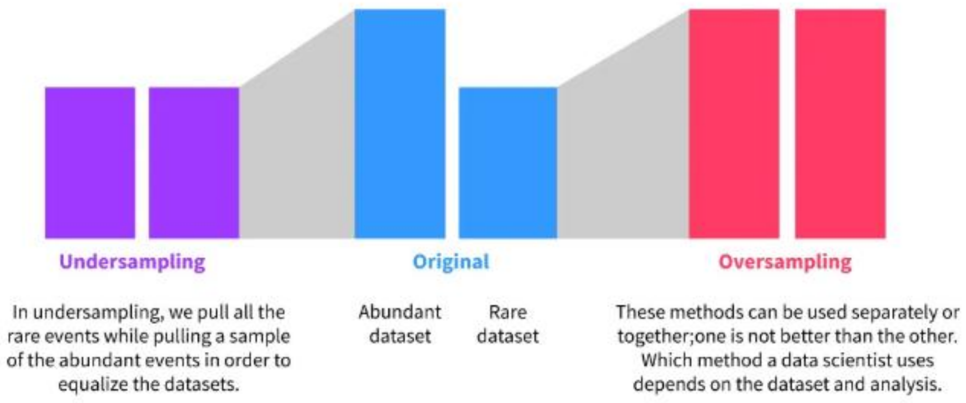
Addressing Class Imbalance: SMOTE (Synthetic Minority Over-sampling Technique) generates new instances for the minority class, aiming to balance the dataset with the majority class.

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.

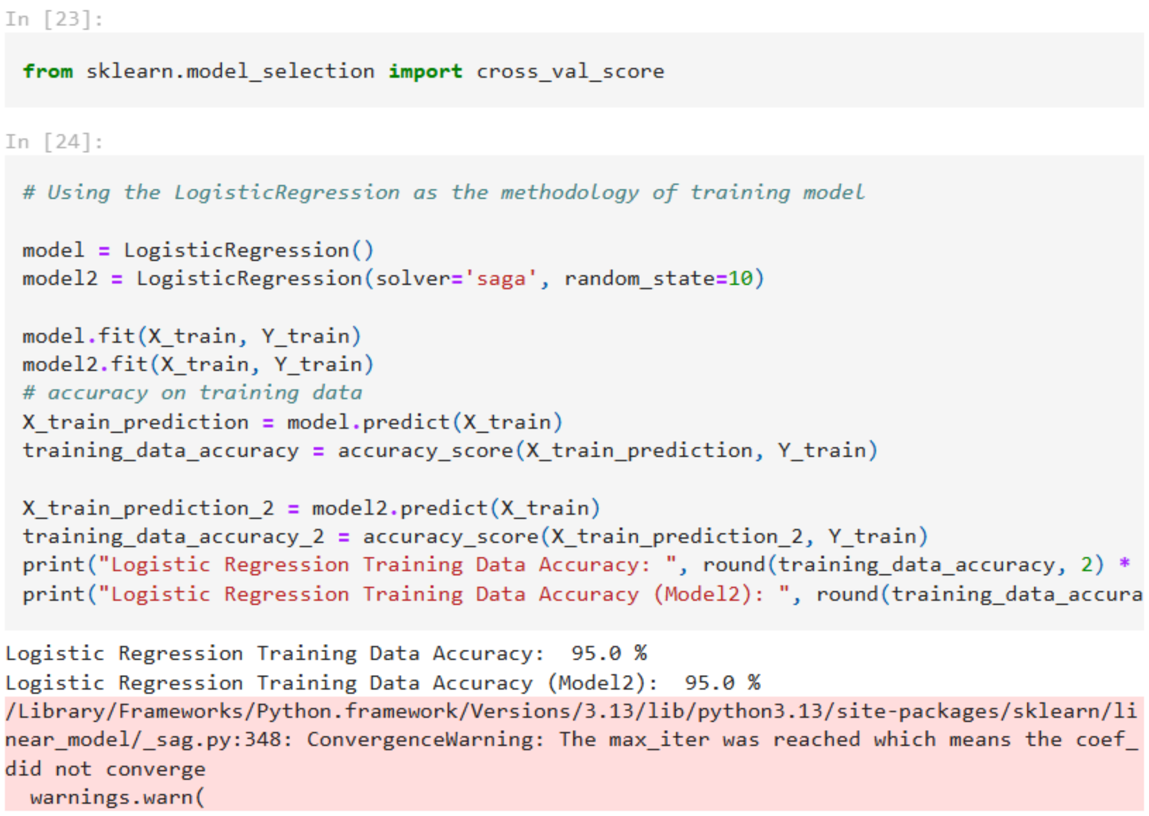
Accuracy vs. Training Time: While SMOTE often improves accuracy compared to random undersampling, it does come with a trade-off. Since no data is discarded, the dataset size increases, leading to longer training times.

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


Undersampling
In undersampling, we pull all the rare events while pulling a sample of the abundant events in order to equalize the datasets.
Original
Abundant dataset
Rare dataset
Oversampling
These methods can be used separately or together;one is not better than the other. Which method a data scientist uses depends on the dataset and analysis.

10 Using Logistic Regression

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



```
In [23]:
from sklearn.model_selection import cross_val_score
```

```
In [24]:
# Using the LogisticRegression as the methodology of training model

model = LogisticRegression()
model2 = LogisticRegression(solver='saga', random_state=10)

model.fit(X_train, Y_train)
model2.fit(X_train, Y_train)
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

X_train_prediction_2 = model2.predict(X_train)
training_data_accuracy_2 = accuracy_score(X_train_prediction_2, Y_train)
print("Logistic Regression Training Data Accuracy: ", round(training_data_accuracy, 2) *
print("Logistic Regression Training Data Accuracy (Model2): ", round(training_data_accura
```

```
Logistic Regression Training Data Accuracy:  95.0 %
Logistic Regression Training Data Accuracy (Model2):  95.0 %
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/sklearn/li
near_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
```

- **Import Libraries:** The code first imports LogisticRegression from sklearn.linear_model and accuracy_score from sklearn.metrics.

- **Initialize and Train Models:**

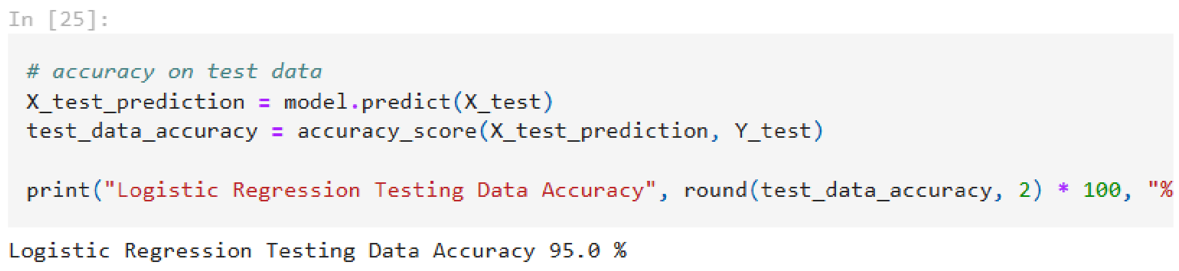
  - model: A Logistic Regression model with default parameters.
  - model2: A Logistic Regression model using the 'saga' solver and a specific random state to ensure reproducibility.
  - Both models are fitted to the training data X_train and Y_train.

- **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.



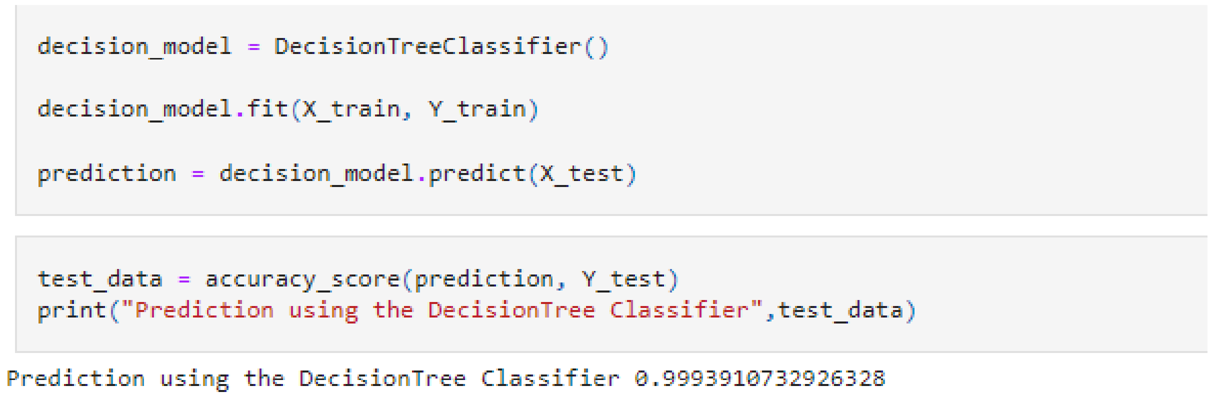
```
In [25]:

# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print("Logistic Regression Testing Data Accuracy", round(test_data_accuracy, 2) * 100, "%

Logistic Regression Testing Data Accuracy 95.0 %
```

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

11 Using Decision Tree Classifier

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.



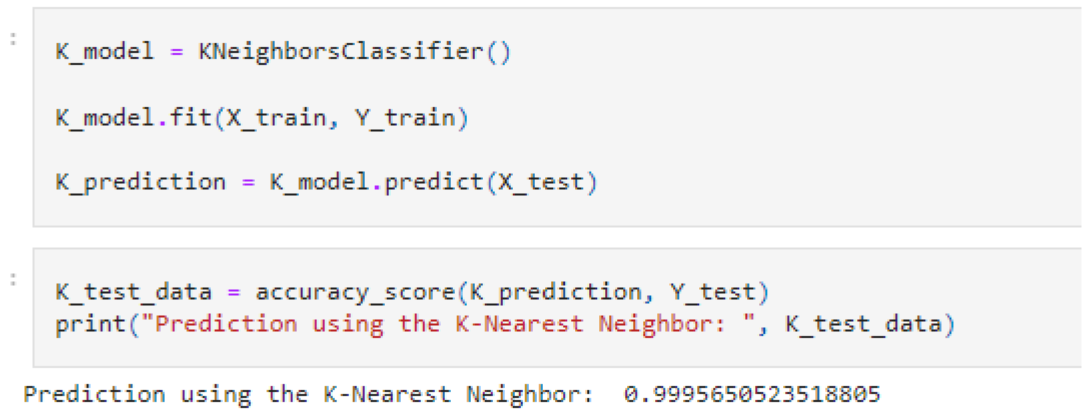
```
decision_model = DecisionTreeClassifier()

decision_model.fit(X_train, Y_train)

prediction = decision_model.predict(X_test)
```

```
test_data = accuracy_score(prediction, Y_test)
print("Prediction using the DecisionTree Classifier",test_data)

Prediction using the DecisionTree Classifier 0.9993910732926328
```

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.

12 Using the KNN Classifier

Nearest neighbor methods operate by identifying a set number of training samples that are closest to a new data point based on distance metrics. The label for the new point is then predicted using these nearby samples. The number of nearest neighbors can be specified as a fixed value (k-nearest neighbor learning) or can be determined based on the local point density (radius-based neighbor learning).



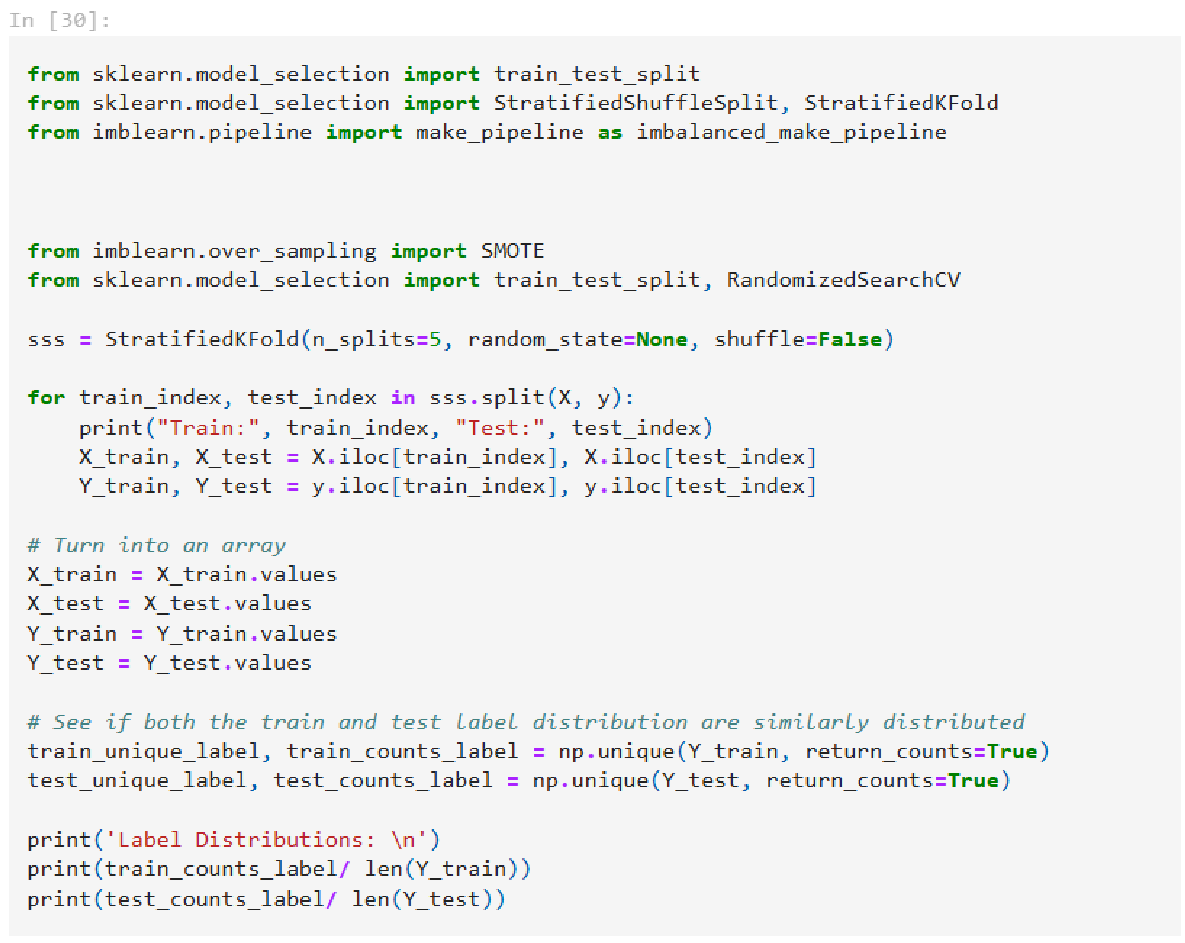
```
K_model = KNeighborsClassifier()

K_model.fit(X_train, Y_train)

K_prediction = K_model.predict(X_test)
```

```
K_test_data = accuracy_score(K_prediction, Y_test)
print("Prediction using the K-Nearest Neighbor: ", K_test_data)
```

```
Prediction using the K-Nearest Neighbor:  0.9995650523518805
```

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.

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

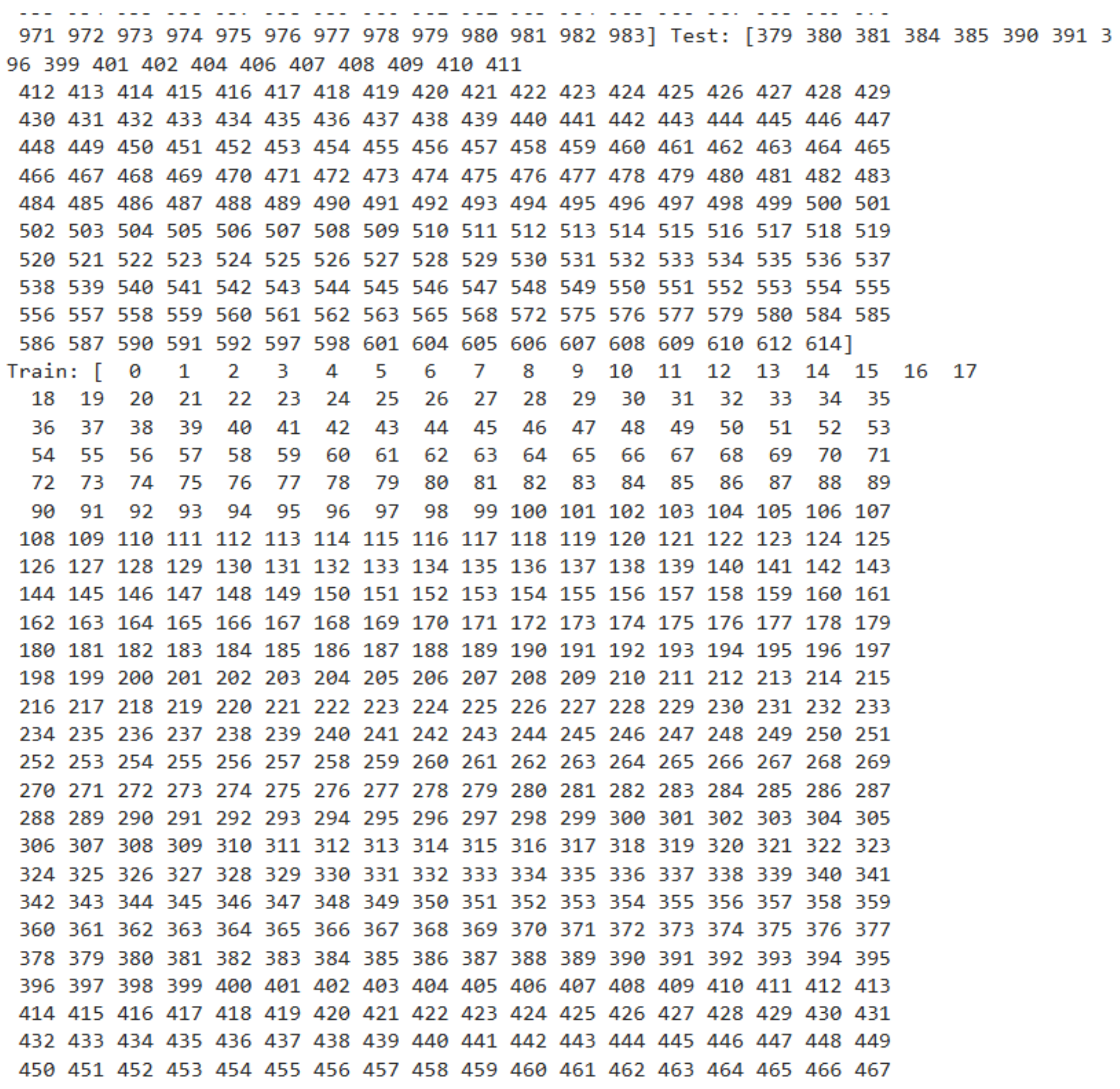


```
In [30]:
```

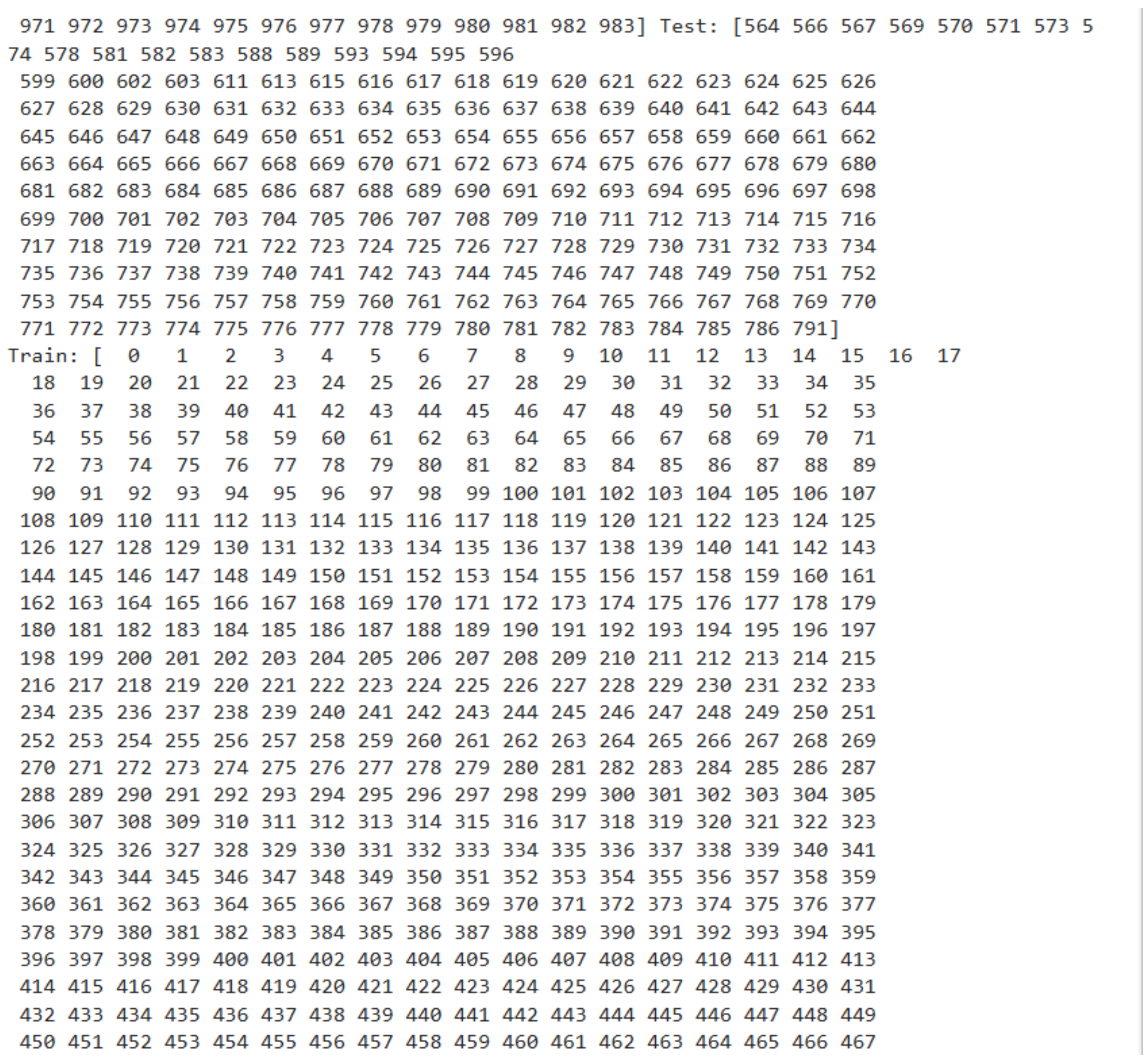
```python
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit, StratifiedKFold
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline



from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, RandomizedSearchCV

sss = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

for train_index, test_index in sss.split(X, y):
    print("Train:", train_index, "Test:", test_index)
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    Y_train, Y_test = y.iloc[train_index], y.iloc[test_index]

# Turn into an array
X_train = X_train.values
X_test = X_test.values
Y_train = Y_train.values
Y_test = Y_test.values

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(Y_train, return_counts=True)
test_unique_label, test_counts_label = np.unique(Y_test, return_counts=True)

print('Label Distributions: \n')
print(train_counts_label/ len(Y_train))
print(test_counts_label/ len(Y_test))
```

- 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.

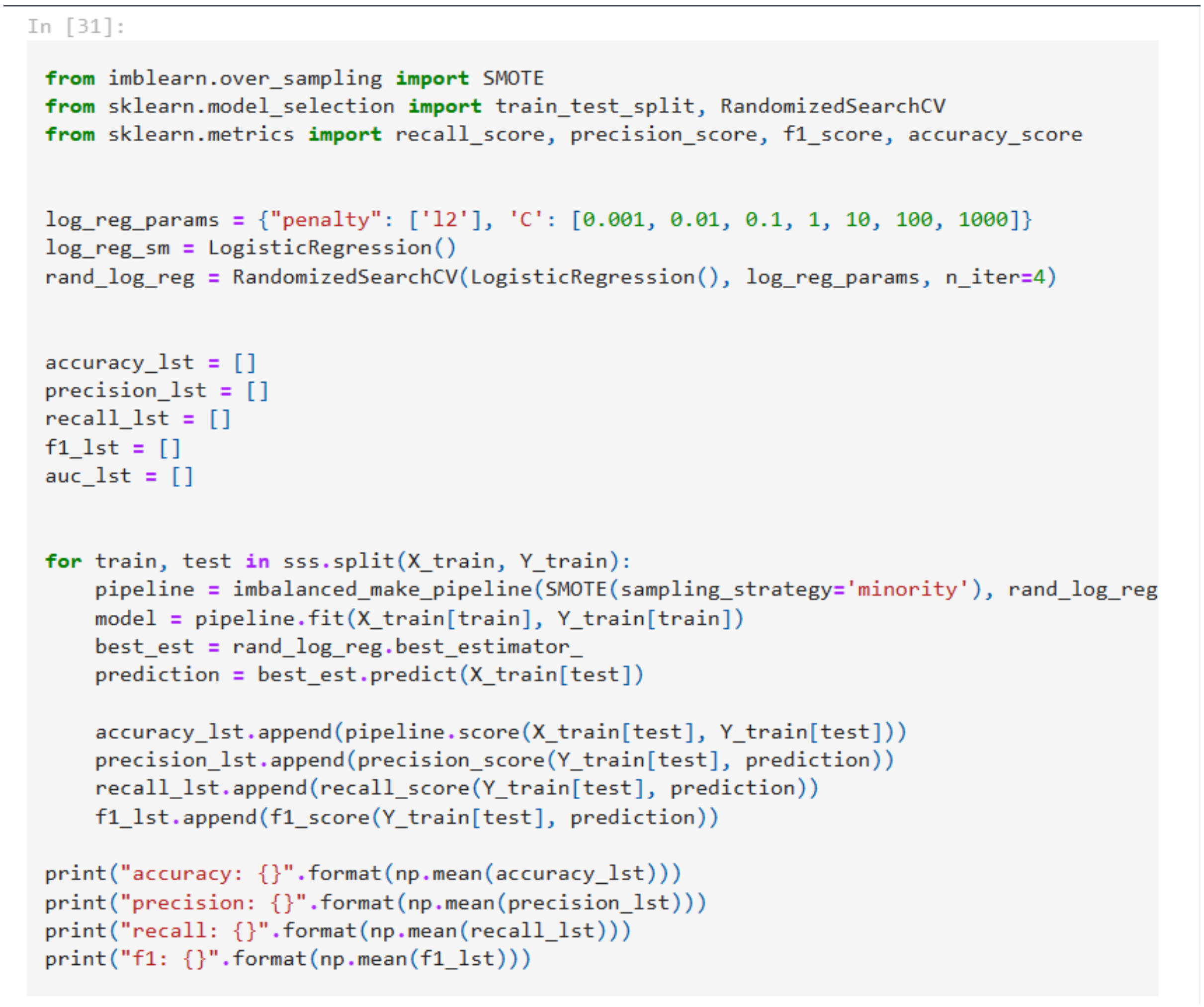
```
Train: [195 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214
 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232
 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250
 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268
 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286
 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304
 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322
 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358
 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376
 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394
 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412
 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430
 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448
 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466
 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484
 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502
 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520
 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538
 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556
 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574
 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592
 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610
 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628
 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646
 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664
 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682
 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700
 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718
 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736
 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754
 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772
 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790
 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808
 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826
 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844
 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862
 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880
 971 972 973 974 975 976 977 978 979 980 981 982 983] Test: [   0   1   2   3   4   5   6
7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 196 197]
Train: [   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 196 197 379
 380 381 384 385 390 391 396 399 401 402 404 406 407 408 409 410 411 412
 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430
 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448
 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466
 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484
 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502
 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520
 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538
 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556
 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574
 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592
 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610
 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628
 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646
 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664
 971 972 973 974 975 976 977 978 979 980 981 982 983] Test: [195 198 199 200 201 202 203 2
04 205 206 207 208 209 210 211 212 213 214
 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232
 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250
 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268
 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286
 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304
 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322
 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358
 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376
 377 378 382 383 386 387 388 389 392 393 394 395 397 398 400 403 405]
Train: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 382 383 386 387 388 389 392 393 394 395 397 398 400 403 405 564 566
 567 569 570 571 573 574 578 581 582 583 588 589 593 594 595 596 599 600
 602 603 611 613 615 616 617 618 619 620 621 622 623 624 625 626 627 628
 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646
 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664
```



```
 971 972 973 974 975 976 977 978 979 980 981 982 983] Test: [379 380 381 384 385 390 391 3
96 399 401 402 404 406 407 408 409 410 411
 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429
 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447
 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465
 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483
 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501
 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519
 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537
 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555
 556 557 558 559 560 561 562 563 565 568 572 575 576 577 579 580 584 585
 586 587 590 591 592 597 598 601 604 605 606 607 608 609 610 612 614]
Train: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
```



```
 971 972 973 974 975 976 977 978 979 980 981 982 983] Test: [564 566 567 569 570 571 573 5
74 578 581 582 583 588 589 593 594 595 596
 599 600 602 603 611 613 615 616 617 618 619 620 621 622 623 624 625 626
 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644
 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662
 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680
 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698
 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716
 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734
 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752
 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770
 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 791]
Train: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503
 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521
 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539
 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557
 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593
 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611
 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629
 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647
 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665
 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683
 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701
 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719
 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737
 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755
 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773
 774 775 776 777 778 779 780 781 782 783 784 785 786 791] Test: [787 788 789 790 792 793 7
94 795 796 797 798 799 800 801 802 803 804 805
 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823
 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841
 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859
 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877
 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895
 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913
 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931
 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949
 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967
 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983]
Label Distributions:

[0.5 0.5]
[0.5 0.5]
```



```
In [31]:
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import recall_score, precision_score, f1_score, accuracy_score


log_reg_params = {"penalty": ['l2'], 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}
log_reg_sm = LogisticRegression()
rand_log_reg = RandomizedSearchCV(LogisticRegression(), log_reg_params, n_iter=4)


accuracy_lst = []
precision_lst = []
recall_lst = []
f1_lst = []
auc_lst = []


for train, test in sss.split(X_train, Y_train):
    pipeline = imbalanced_make_pipeline(SMOTE(sampling_strategy='minority'), rand_log_reg
    model = pipeline.fit(X_train[train], Y_train[train])
    best_est = rand_log_reg.best_estimator_
    prediction = best_est.predict(X_train[test])

    accuracy_lst.append(pipeline.score(X_train[test], Y_train[test]))
    precision_lst.append(precision_score(Y_train[test], prediction))
    recall_lst.append(recall_score(Y_train[test], prediction))
    f1_lst.append(f1_score(Y_train[test], prediction))

print("accuracy: {}".format(np.mean(accuracy_lst)))
print("precision: {}".format(np.mean(precision_lst)))
print("recall: {}".format(np.mean(recall_lst)))
print("f1: {}".format(np.mean(f1_lst)))
```

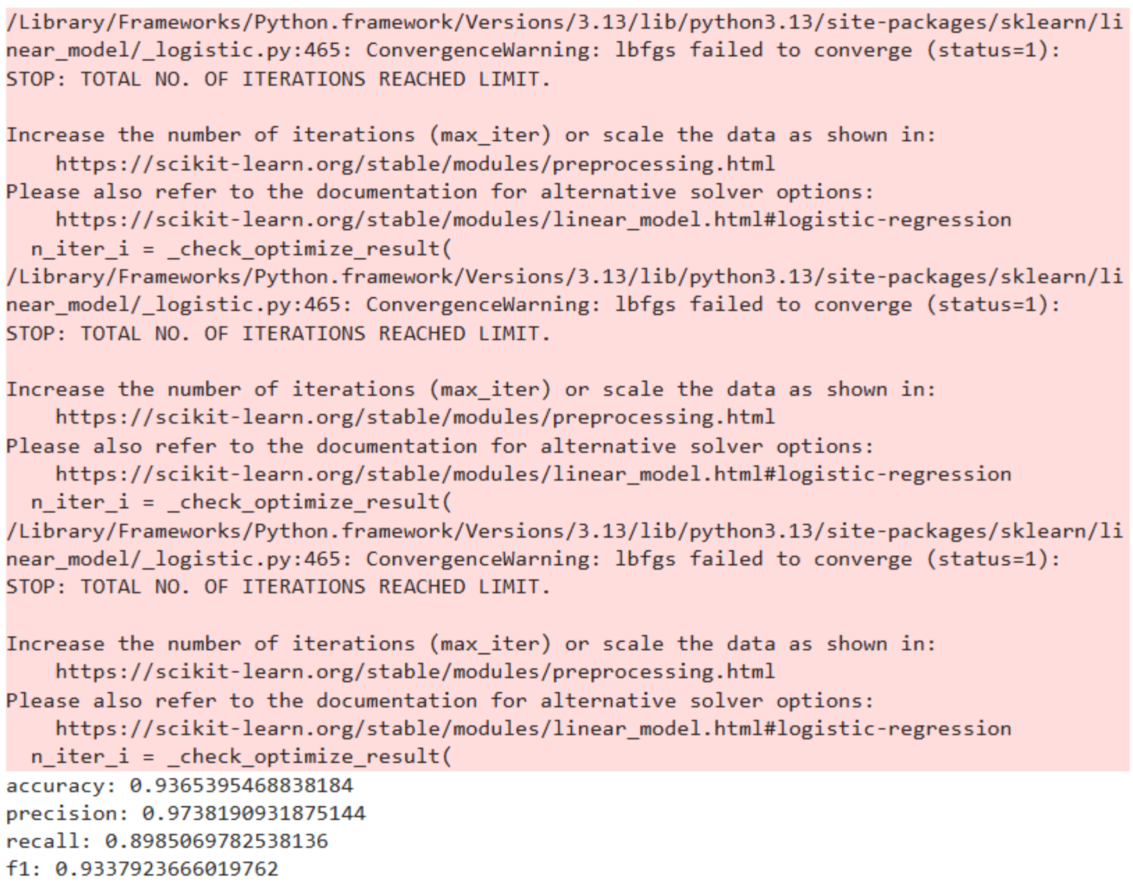
- **Goal:** Train and evaluate a Logistic Regression model to detect anomalies using imbalanced datasets and improve performance through hyperparameter tuning and cross-validation.

- **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.



```
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/sklearn/li
near_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/sklearn/li
near_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/Library/Frameworks/Python.framework/Versions/3.13/lib/python3.13/site-packages/sklearn/li
near_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
accuracy: 0.9365395468838184
precision: 0.9738190931875144
recall: 0.8985069782538136
f1: 0.9337923666019762
```

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