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.

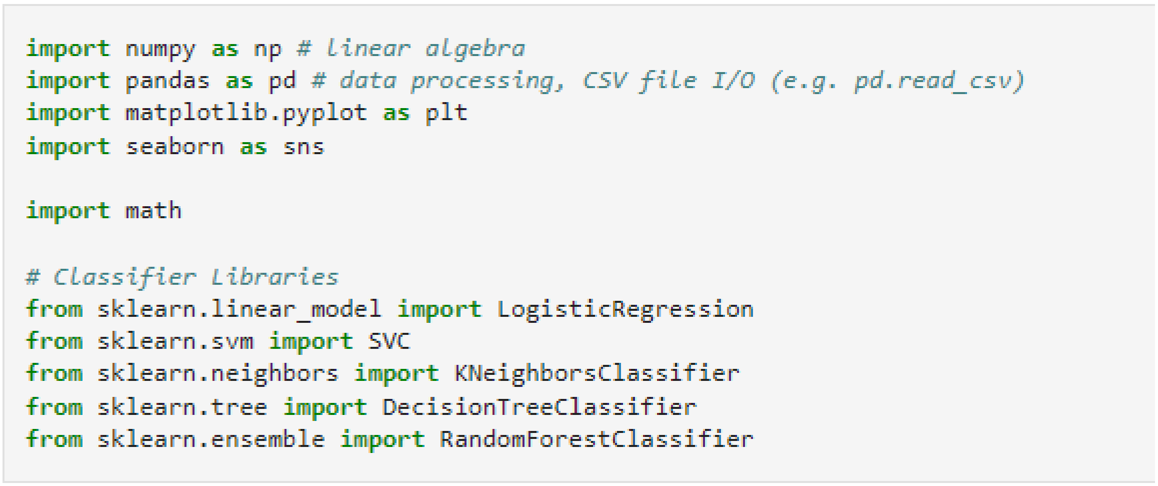
Conclusion

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:



```python
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.

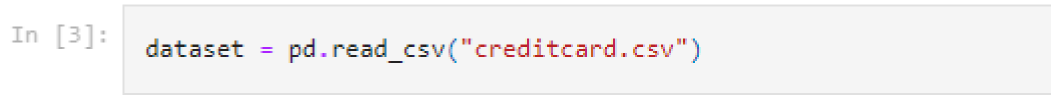
2 Understanding our data:

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)

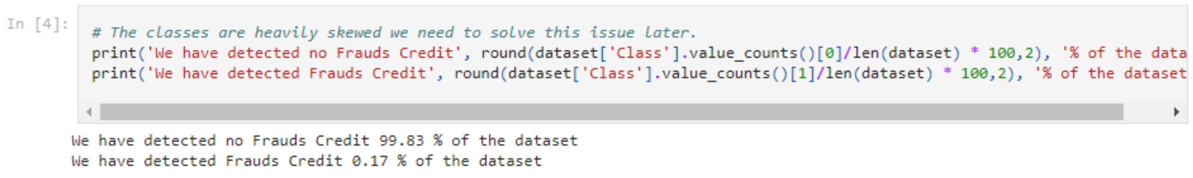
To read dataset we use this command



```
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
```

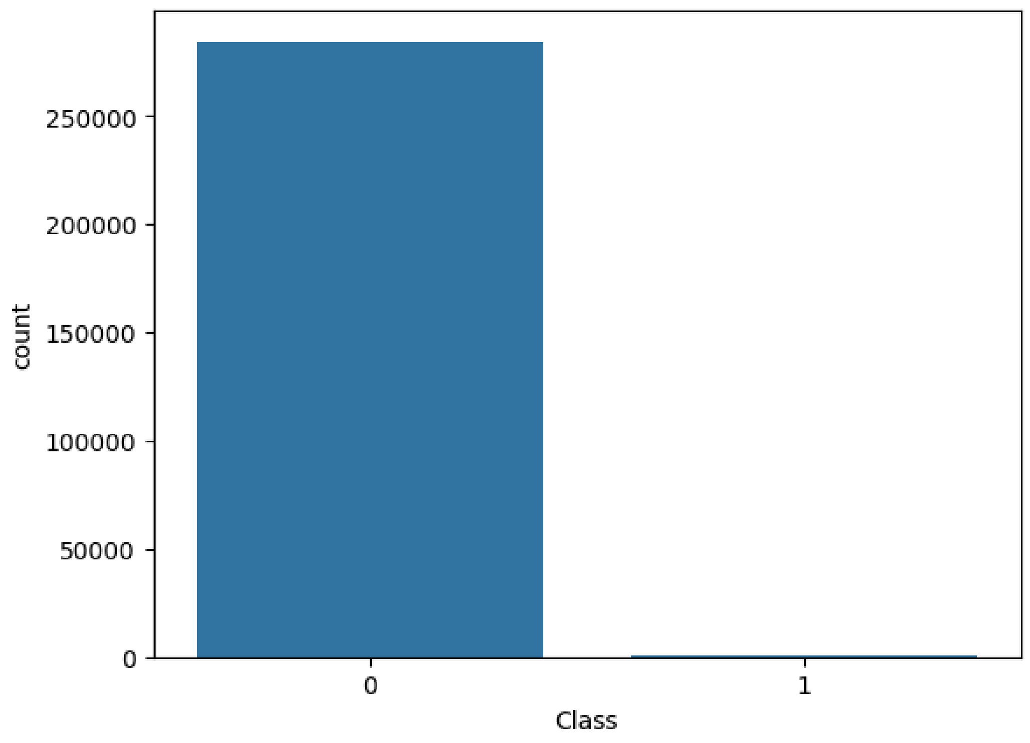
4 Glimpse the data:

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

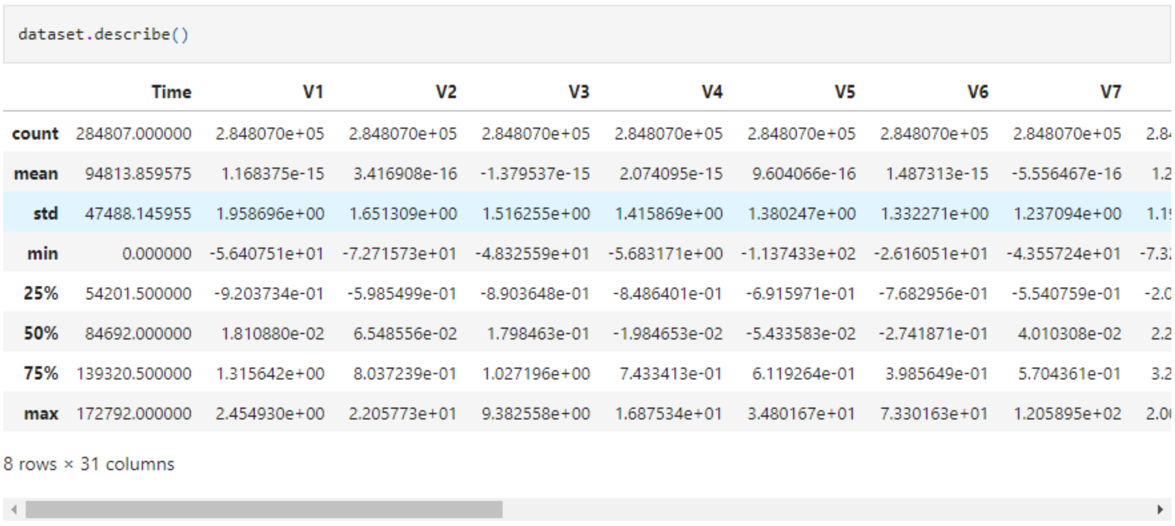
Definition: Because most of the transaction need to be secure during all the transaction, with many transaction need to be secure before can other transaction can be made. So the value contain the fraud only 50% of the all the transaction made it. The dataset are validate to use as a research purposeAutomotive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

```
sns.countplot(x='Class', data=dataset)
```

```
<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.

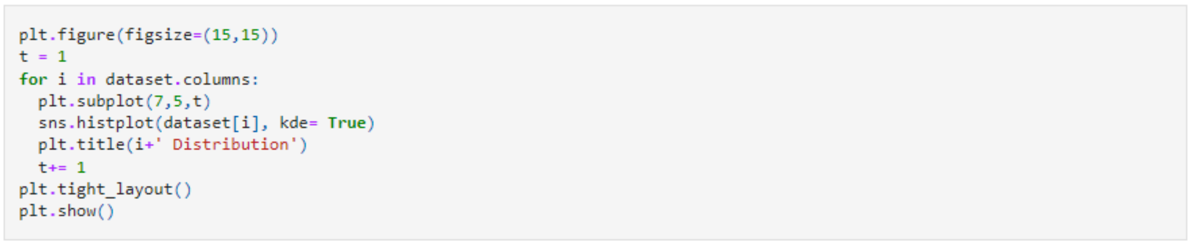


```
dataset.describe()
```

	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()
```

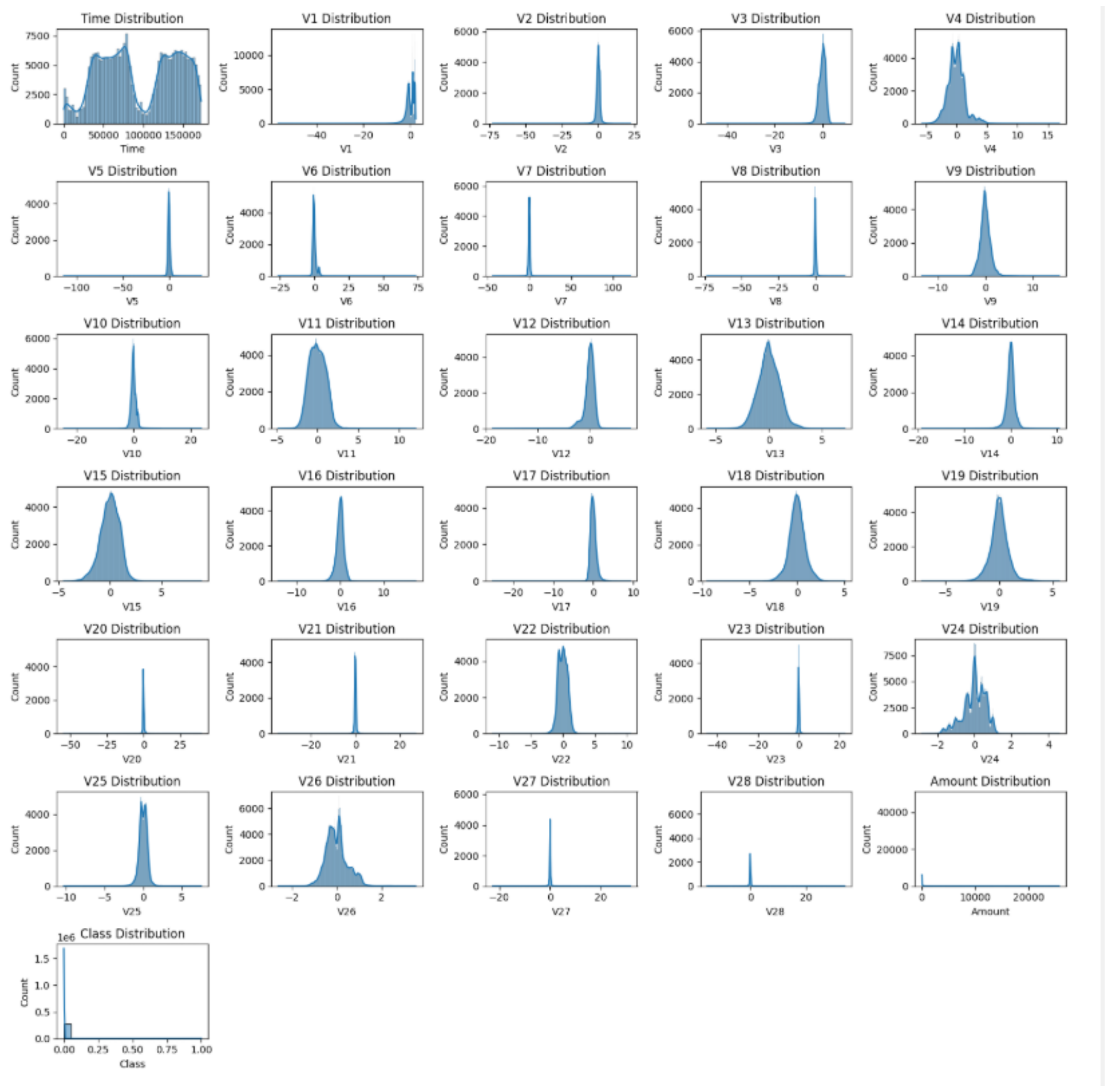
Set Figure Size: This sets the size of the entire figure to 15x15 inches.
Initialize Counter: A counter t is initialized to 1. This counter is used to specify the position of the subplot within the figure.
Loop Through Columns:

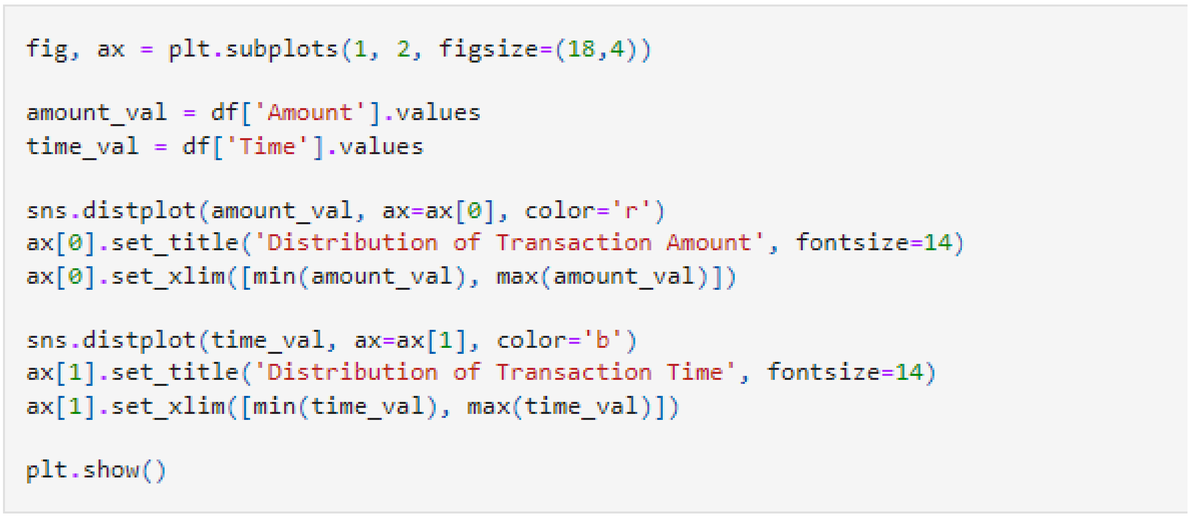
- The loop iterates through each column in the dataset.
- plt.subplot(7,5,t):
   - This creates a subplot in a 7x5 grid at the position specified by t.
- sns.histplot(dataset[i], kde=True):
   - This creates a histogram for the column i using Seaborn's histplot function.

- 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.





```
fig, ax = plt.subplots(1, 2, figsize=(18,4))

amount_val = df['Amount'].values
time_val = df['Time'].values

sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Time', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])

plt.show()
```

Creating Subplots:

- plt.subplots(1, 2, figsize=(18,4)): Creates a figure with a 1x2 grid of subplots (1 row, 2 columns). The size of the entire figure is set to 18x4 inches.
- fig: Represents the entire figure.
- ax: An array of Axes objects corresponding to the subplots. Here, ax[0] will refer to the first subplot and ax[1] to the second.

Extracting Values:

- df['Amount'].values: Extracts the values from the 'Amount' column of the dataframe df.
- df['Time'].values: Extracts the values from the 'Time' column of the dataframe df.

Plotting Distribution of Transaction Amount:

- sns.distplot(amount_val, ax=ax[0], color='r'): Creates a distribution plot (histogram with a KDE curve) for the amount_val data in the first subplot (ax[0]). The plot color is set to red (color='r').
- ax[0].set_title(): Sets the title of the first subplot to 'Distribution of Transaction Amount' with a font size of 14.
- ax[0].set_xlim(): Sets the x-axis limits of the first subplot to the minimum and maximum values of amount_val.

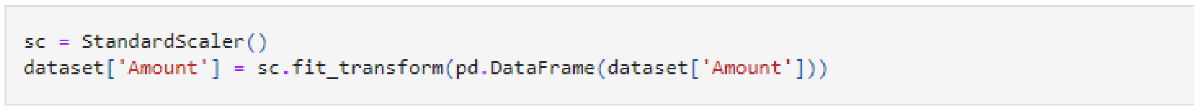
Plotting Distribution of Transaction Time:

- sns.distplot(): Creates a distribution plot for the time_val data in the second subplot (ax[1]). The plot color is set to blue (color='b').
- ax[1].set_title(): Sets the title of the second subplot to 'Distribution of Transaction Time' with a font size of 14.

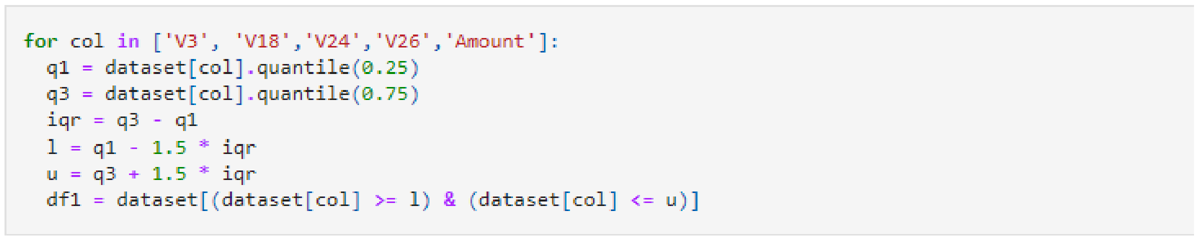
- ax[1].set_xlim([min(time_val), max(time_val)]): Sets the x-axis limits of the second subplot to the minimum and maximum values of time_val.

Displaying the Plot:
This displays the figure with the two subplots.

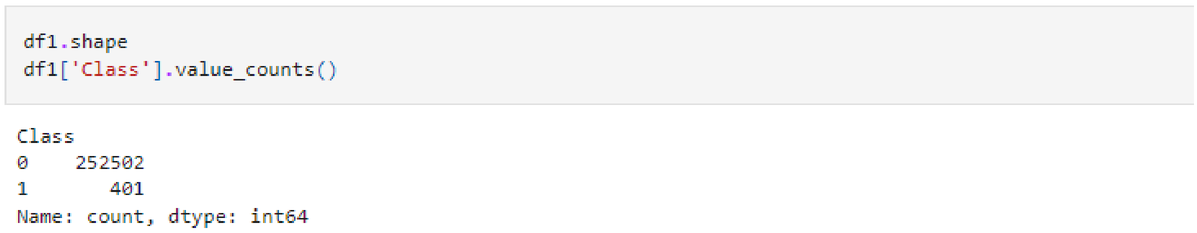
6 Data Preprocessing



```
sc = StandardScaler()
dataset['Amount'] = sc.fit_transform(pd.DataFrame(dataset['Amount']))
```

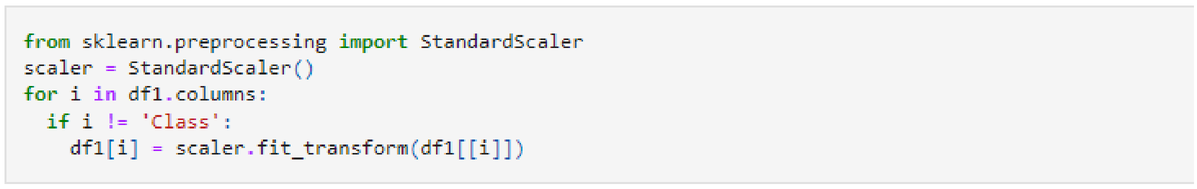


```
for col in ['V3', 'V18','V24','V26','Amount']:
  q1 = dataset[col].quantile(0.25)
  q3 = dataset[col].quantile(0.75)
  iqr = q3 - q1
  l = q1 - 1.5 * iqr
  u = q3 + 1.5 * iqr
  df1 = dataset[(dataset[col] >= l) & (dataset[col] <= u)]
```



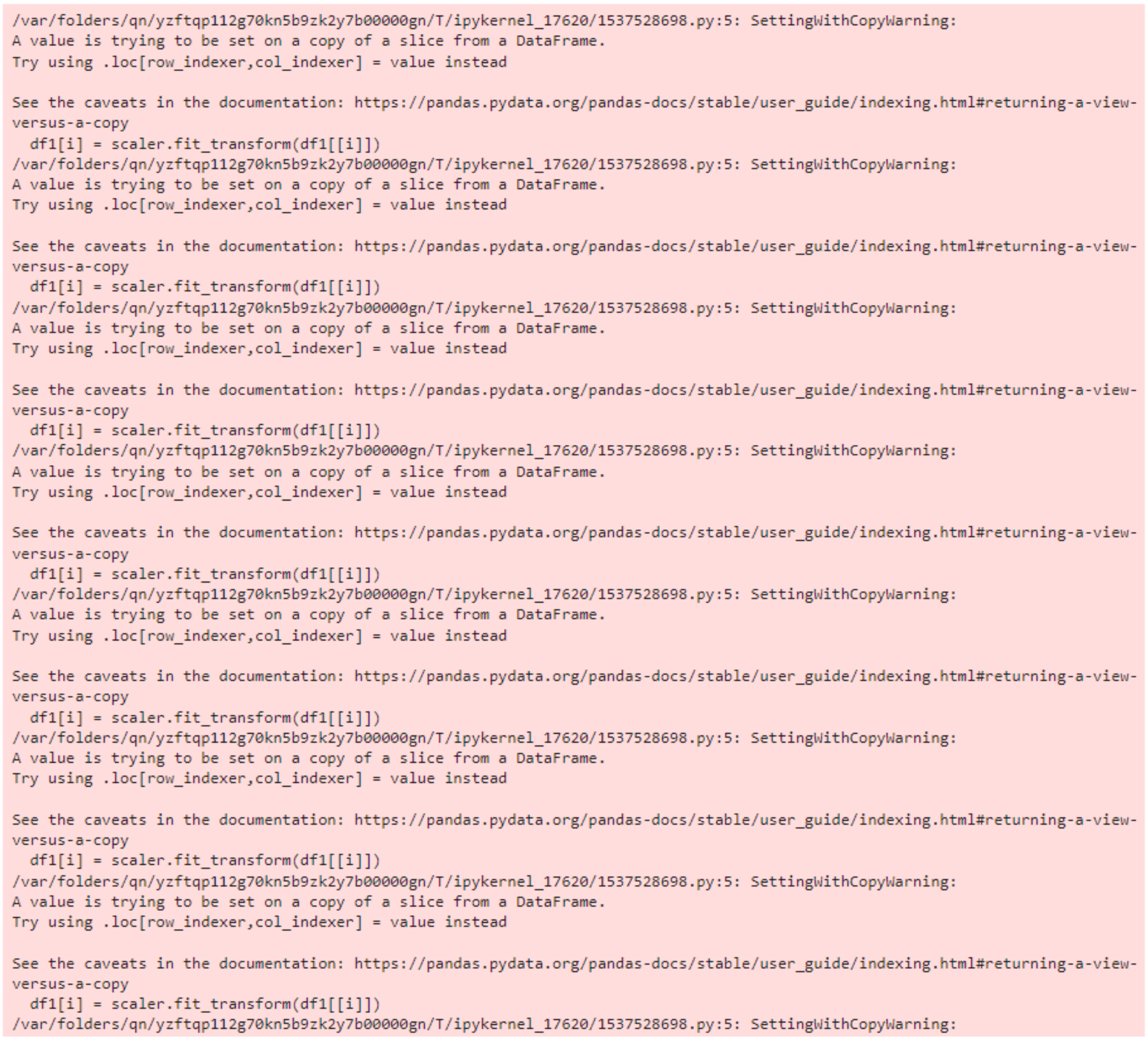
```
df1.shape
df1['Class'].value_counts()
```

```
Class
0    252502
1       401
Name: count, dtype: int64
```



```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
for i in df1.columns:
  if i != 'Class':
    df1[i] = scaler.fit_transform(df1[[i]])
```

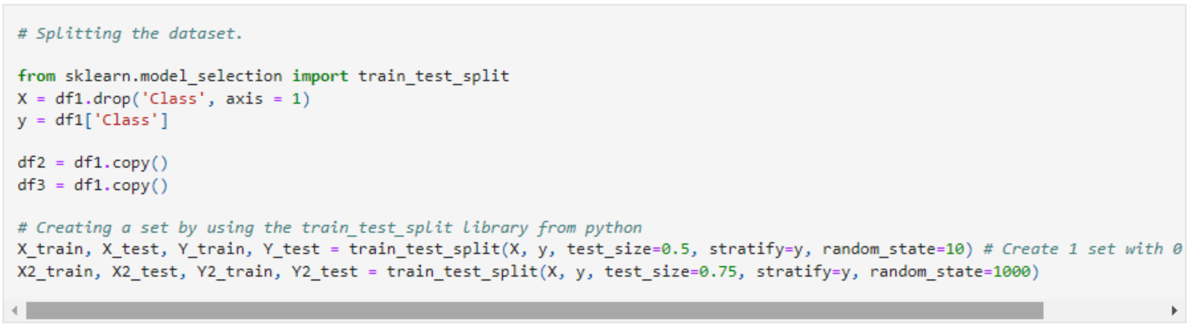
This code standardizes the 'Amount' column, removes outliers from selected columns, checks the shape and class distribution of the filtered dataset, and finally standardizes the remaining columns.



```
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_17620/1537528698.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
versus-a-copy
  df1[i] = scaler.fit_transform(df1[[i]])
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_17620/1537528698.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
versus-a-copy
  df1[i] = scaler.fit_transform(df1[[i]])
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_17620/1537528698.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
versus-a-copy
  df1[i] = scaler.fit_transform(df1[[i]])
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_17620/1537528698.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
versus-a-copy
  df1[i] = scaler.fit_transform(df1[[i]])
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_17620/1537528698.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
versus-a-copy
  df1[i] = scaler.fit_transform(df1[[i]])
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_17620/1537528698.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
versus-a-copy
  df1[i] = scaler.fit_transform(df1[[i]])
/var/folders/qn/yzftqp112g70kn5b9zk2y7b00000gn/T/ipykernel_17620/1537528698.py:5: SettingWithCopyWarning:
```

7 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



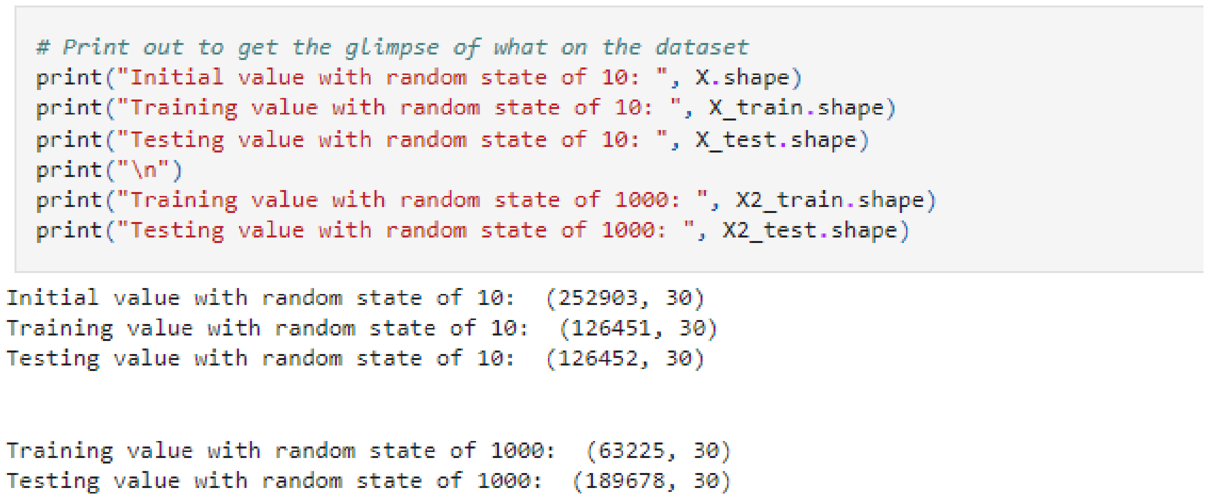
```python
# Splitting the dataset.

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.



```
# Print out to get the glimpse of what on the dataset
print("Initial value with random state of 10: ", X.shape)
print("Training value with random state of 10: ", X_train.shape)
print("Testing value with random state of 10: ", X_test.shape)
print("\n")
print("Training value with random state of 1000: ", X2_train.shape)
print("Testing value with random state of 1000: ", X2_test.shape)
```

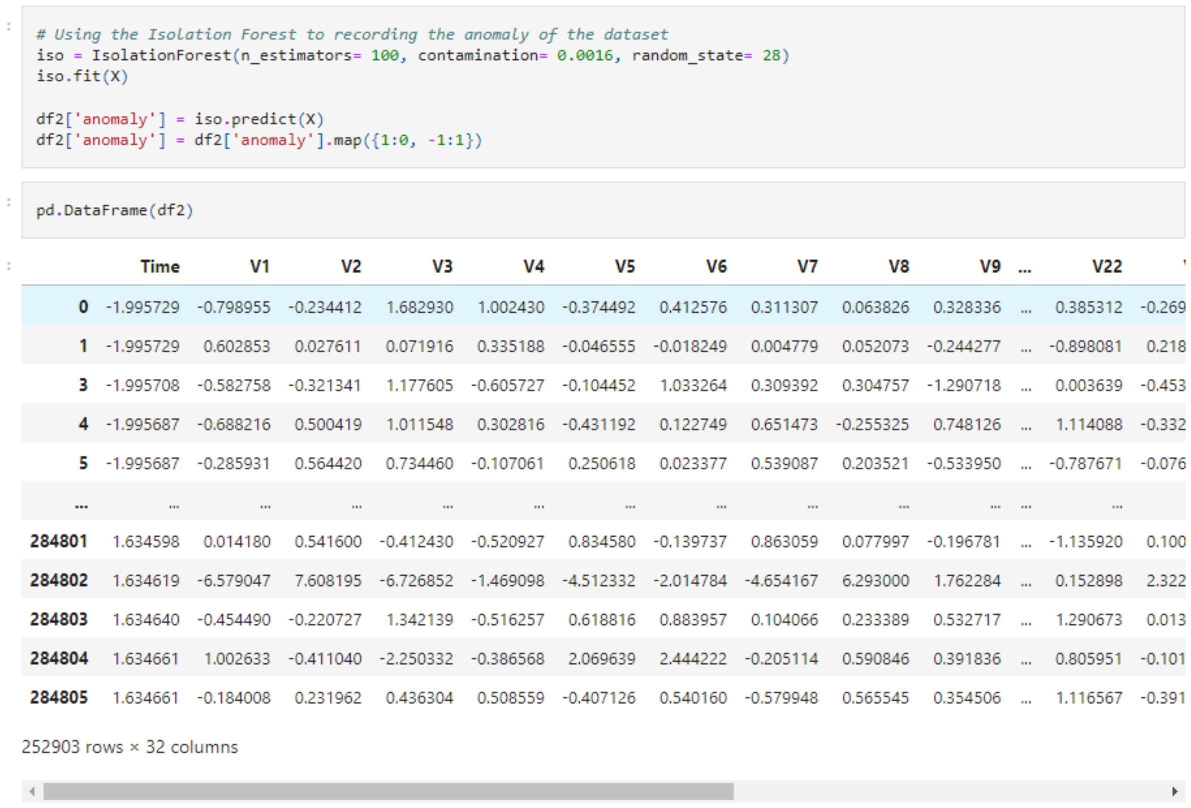
```
Initial value with random state of 10:  (252903, 30)
Training value with random state of 10:  (126451, 30)
Testing value with random state of 10:  (126452, 30)


Training value with random state of 1000:  (63225, 30)
Testing value with random state of 1000:  (189678, 30)
```

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

8 Using the Isolation Forest Model

The IsolationForest algorithm works by isolating observations. It does this by randomly picking a feature and then choosing a split value between the feature's maximum and minimum values. This method, known as recursive partitioning, can be visualized as a tree structure. The number of splits needed to isolate a sample corresponds to the path length from the root node to a terminal node. Averaging this path length over a forest of random trees gives a measure of normality, which forms the basis of the decision function.

Ref:https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html



```
# Using the Isolation Forest to recording the anomaly of the dataset
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})
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V22	
0	-1.995729	-0.798955	-0.234412	1.682930	1.002430	-0.374492	0.412576	0.311307	0.063826	0.328336	...	0.385312	-0.269
1	-1.995729	0.602853	0.027611	0.071916	0.335188	-0.046555	-0.018249	0.004779	0.052073	-0.244277	...	-0.898081	0.218
3	-1.995708	-0.582758	-0.321341	1.177605	-0.605727	-0.104452	1.033264	0.309392	0.304757	-1.290718	...	0.003639	-0.453
4	-1.995687	-0.688216	0.500419	1.011548	0.302816	-0.431192	0.122749	0.651473	-0.255325	0.748126	...	1.114088	-0.332
5	-1.995687	-0.285931	0.564420	0.734460	-0.107061	0.250618	0.023377	0.539087	0.203521	-0.533950	...	-0.787671	-0.076
...	...	...	...	...	...	...	...	...	...	...	...	...	
284801	1.634598	0.014180	0.541600	-0.412430	-0.520927	0.834580	-0.139737	0.863059	0.077997	-0.196781	...	-1.135920	0.100
284802	1.634619	-6.579047	7.608195	-6.726852	-1.469098	-4.512332	-2.014784	-4.654167	6.293000	1.762284	...	0.152898	2.322
284803	1.634640	-0.454490	-0.220727	1.342139	-0.516257	0.618816	0.883957	0.104066	0.233389	0.532717	...	1.290673	0.013
284804	1.634661	1.002633	-0.411040	-2.250332	-0.386568	2.069639	2.444222	-0.205114	0.590846	0.391836	...	0.805951	-0.101
284805	1.634661	-0.184008	0.231962	0.436304	0.508559	-0.407126	0.540160	-0.579948	0.565545	0.354506	...	1.116567	-0.391

252903 rows × 32 columns

9 Using Logistic Regression

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

```
# Using the LogisticRegression as the methodology of training model

model = LogisticRegression()
model2 = LogisticRegression(solver='liblinear', random_state=10, penalty='elasticnet')

model.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)

print("Logistic Regression Training Data Accuracy: ", training_data_accuracy)
```

```
Logistic Regression Training Data Accuracy:  0.9991696388324331
```

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

print(test_data_accuracy)
```

```
0.999161737259988
```

The code creates two logistic regression models, trains the first one, makes predictions on the same data, calculates the accuracy of these predictions, and prints the accuracy.

The code evaluates the logistic regression model on the test dataset by:

1. Making predictions using the test features (X_test).
2. Calculating the accuracy of these predictions by comparing them with the true test labels (Y_test).
3. Printing the accuracy score, which indicates how well the model performs on unseen data.

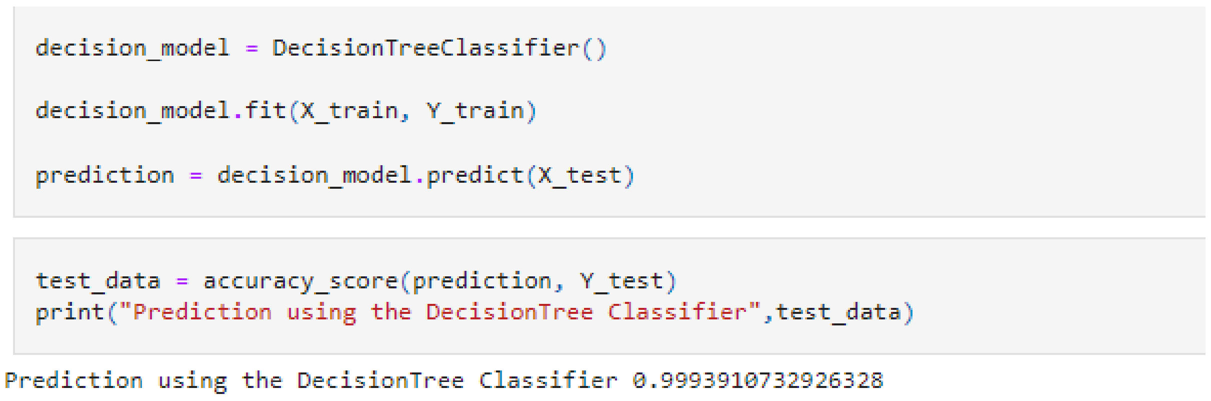
This helps you understand the effectiveness of your model in detecting fraudulent transactions in the test dataset.

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

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

10 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)
```

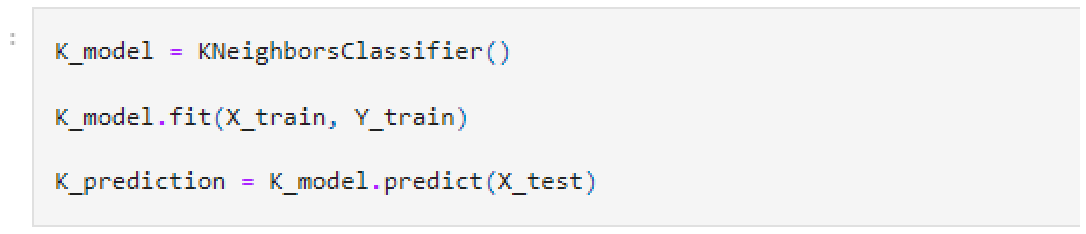
```
test_data = accuracy_score(prediction, Y_test)
print("Prediction using the DecisionTree Classifier",test_data)
```

```
Prediction using the DecisionTree Classifier 0.9993910732926328
```

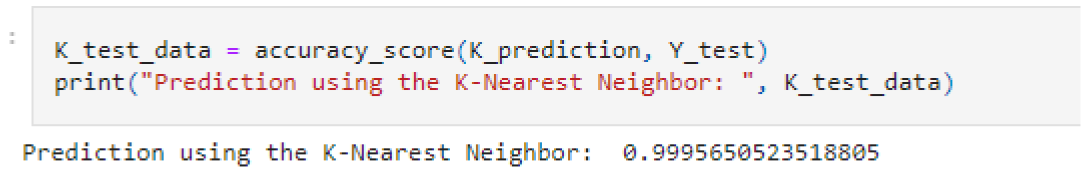
The code initializes a Decision Tree Classifier, trains it using the training data, makes predictions on the test data, calculates the accuracy of these predictions, and prints the accuracy score. This process helps evaluate how well the Decision Tree Classifier can detect fraudulent transactions in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

11 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)
```



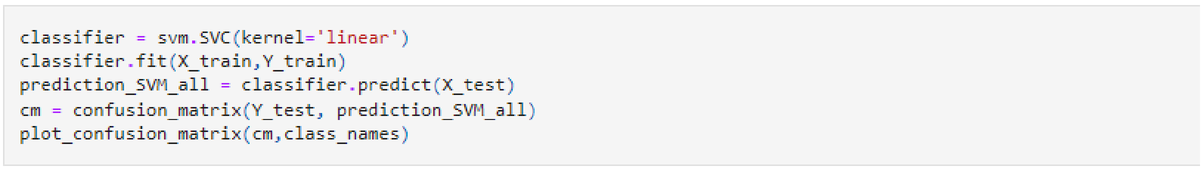
```
K_test_data = accuracy_score(K_prediction, Y_test)
print("Prediction using the K-Nearest Neighbor: ", K_test_data)
```

```
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.

12 Using the Support Vector Machine

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



```
classifier = svm.SVC(kernel='linear')
classifier.fit(X_train,Y_train)
prediction_SVM_all = classifier.predict(X_test)
cm = confusion_matrix(Y_test, prediction_SVM_all)
plot_confusion_matrix(cm,class_names)
```

The code initializes a Support Vector Machine (SVM) classifier with a linear kernel, trains it using the training data, makes predictions on the test data, calculates the confusion matrix, and plots the confusion matrix to visualize the performance of the model. This process helps evaluate how well the SVM classifier can detect fraudulent transactions in the test dataset.

By after creating a classification 'bout using the SVM, next we will rebalance our data to get the classification (supervised)

```
classifier_b = svm.SVC(kernel='linear',class_weight={0:0.60, 1:0.40})
classifier_b.fit(X_train, y_train)
```

The code initializes an SVM classifier with a linear kernel and custom class weights to address class imbalance. It then trains the classifier using the training data. The class weights ensure that the model pays more attention to the minority class (fraudulent transactions) during training, which can help improve detection performance.

Instead stick with the linear model, why we don't change it up go poly for polynomial kernel

```
classifier_b = svm.SVC(kernel='poly',class_weight={0:0.60, 1:0.40})
classifier_b.fit(X_train, y_train)
prediction_SVM_b_all = classifier_b.predict(X_test_all)
cm = confusion_matrix(y_test_all, prediction_SVM_b_all)
plot_confusion_matrix(cm,class_names)
```

This code initializes an SVM classifier with a polynomial kernel and custom class weights, trains it using the training data, makes predictions on the test data, calculates the confusion matrix, and plots the confusion matrix to visualize the performance of the model.

```
report= classification_report(Y_test, prediction_SVM_b_all)
print(report)

mean_abs_error = mean_absolute_error(y_test_all,prediction_SVM_b_all)
mean_abs_percentage_error = np.mean(np.abs((y_test_all - prediction_SVM_b_all) // y_test_all))
mse= mean_squared_error(y_test_all,prediction_SVM_b_all)
print("Mean absolute error : {} \nMean Absolute Percentage error : {}\nMean Squared Error : {}".format(mean_abs_error,mean_
```

The code generates and prints a classification report, calculates and prints three error metrics (mean absolute error, mean absolute percentage error, and mean squared error) based on the actual and predicted labels. These metrics help evaluate the performance of the SVM classifier.

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