Data Science Project

Credit Card Fraud Detection

Sonika Bhardwaj
Sonikabhardwaj05@gmail.com

Introduction:

In an era defined by digital transactions and financial interconnectivity, the prevalence of credit card fraud poses a significant threat to both financial institutions and consumers worldwide. The evolution of fraudulent tactics has become increasingly sophisticated, necessitating innovative approaches to detect and prevent unauthorized activities. To combat this challenge, the implementation of advanced machine learning models has emerged as a promising solution, enabling the identification of fraudulent patterns in real-time with enhanced accuracy and efficiency.

This project focuses on the development and deployment of a robust credit card fraud detection system leveraging state-of-the-art machine learning algorithms. The primary objective is to create a predictive model capable of swiftly analyzing vast volumes of transactional data to differentiate between legitimate and fraudulent activities. By harnessing the power of machine learning, this project aims to proactively identify anomalous behaviors and patterns, thereby fortifying the security measures surrounding credit card transactions.

The significance of this endeavor lies in its potential to mitigate financial losses for both financial institutions and consumers while safeguarding sensitive personal and financial information. By employing a data-driven approach, this project seeks to contribute to the ongoing efforts in enhancing cybersecurity measures within the realm of financial transactions, fostering trust and confidence in electronic payment systems.

Key components of this project include data preprocessing, feature engineering, model training and evaluation, and the deployment of an efficient and scalable fraud detection system. The success of this model will be measured by its ability to accurately distinguish between legitimate and fraudulent transactions while minimizing false positives and negatives, ultimately providing a reliable and adaptive defense against evolving fraudulent tactics.

Through this project, the aim is to contribute to the advancement of credit card fraud detection methodologies, empowering financial institutions and stakeholders with a proactive and resilient defense mechanism against illicit activities in the ever-evolving landscape of digital finance.

Credit Card Fraud Detection Project Steps

Overview

This project focuses on implementing a machine learning-driven solution for detecting credit card fraud. By leveraging advanced algorithms and data analysis techniques, the goal is to swiftly identify and prevent fraudulent transactions within electronic payment systems. Through meticulous model development and evaluation, this initiative aims to bolster security measures, minimizing financial losses and safeguarding consumer trust. The project's primary objective is to create an accurate, scalable, and real-time fraud detection system, contributing to the ongoing battle against evolving fraudulent activities in the realm of digital transactions.

Key Objectives

The key objectives of a credit card fraud detection project typically include:
Accuracy Improvement: Enhance the accuracy of fraud detection algorithms to effectively differentiate between legitimate and fraudulent transactions, reducing false positives and negatives.

Real-time Detection: Develop a system capable of detecting fraudulent activities in real-time to prevent unauthorized transactions swiftly.

Scalability: Build a scalable model capable of handling large volumes of transactional data without compromising efficiency or accuracy.

Reducing Financial Losses: Minimize financial losses incurred by both financial institutions and consumers due to fraudulent activities through early detection and prevention.

Adaptability and Robustness: Create a system that continuously adapts to new fraud patterns and techniques, ensuring ongoing effectiveness against evolving fraud tactics.

User-Friendly Integration: Integrate the fraud detection system seamlessly into existing payment processing systems without disrupting the user experience or transaction flow.

Compliance and Regulation: Ensure adherence to legal and regulatory frameworks related to financial transactions and data privacy while implementing fraud detection measures.

Methodologies

Libraries and Dependencies

Purpose

This section imports the necessary Python libraries and dependencies to facilitate various data manipulation, visualization, and machine learning tasks.

Code overview

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report , accuracy_score
from sklearn.ensemble import IsolationForest
import shap
import warnings
warnings.filterwarnings('ignore')
```

Explanation:

- NumPy and pandas: Used for efficient data manipulation.
- seaborn and matplotlib.pyplot: Employed for data visualization.
- scikit-learn (sklearn): Provides tools for machine learning tasks such as model training, evaluation, and preprocessing.
- shap: Utilized for interpreting model predictions.
- warnings: Used to suppress deprecation warnings for cleaner output.

Dataset Loading and Exploration

Purpose
This section loads the credit card transaction dataset and explores its structure, characteristics, and potential issues.

Code overview

```
df = pd.read_csv('creditcard.csv')
df.head()
df.tail()
df.isnull().sum()
```

Data Cleaning

Purpose

This section identifies and addresses data quality issues, including duplicate records.

Code overview

```
df =df.duplicated().sum()
print('Number of duplicate records:', duplicate_rows)
df =df.drop_duplicates()
print(df.isnull().sum())
```

Explanation:

df.duplicated().sum(): Calculates the number of duplicate rows.
df.drop_duplicates(): Removes duplicate records.
print(df.isnull().sum()): Verifies that there are no missing values after removing duplicates.

Class Distribution Analysis

This section identifies the class of the dataset.

Code overview

Df["Class"].value_counts()

```
0    284315
1       492
Name: Class, dtype: int64
```

Exploratory Data Analysis (EDA)

Purpose

This section visualizes the data distribution and relationships among features to gain insights.

Exploratory Data Analysis (EDA) is a crucial preliminary step in data analysis that involves examining and understanding the structure, patterns, relationships, anomalies, and insights within a dataset. It serves several important purposes in data analysis and machine learning:

Understanding the Data: EDA helps in gaining a deeper understanding of the dataset you're working with. It involves summarizing the main characteristics of the data, such as the number of data points, features, data types, and their distributions. This understanding is crucial for making informed decisions during the modeling process.

Identifying Patterns and Relationships: EDA helps in identifying patterns, trends, and relationships between variables. It involves statistical and visualization techniques to uncover correlations, dependencies, and associations between different features in the dataset.

Handling Missing Values and Outliers: EDA allows the detection and treatment of missing values and outliers in the data. Understanding the nature and extent of missing data or outlier values is important for deciding whether to impute missing values, remove outliers, or use specialized techniques to handle them.

Feature Selection and Engineering: Through EDA, you can identify important features that have a significant impact on the target variable. This process aids in feature selection, where irrelevant or redundant features may be removed, and feature engineering, where new informative features may be created based on insights gained from the data.

Model Selection and Assumptions: EDA helps in choosing appropriate modeling techniques by examining whether the assumptions of the intended models are met by the data. It provides insights into whether the relationships between variables are linear or nonlinear, whether there's multicollinearity, and more.

Communicating Results: EDA often involves the creation of visualizations and summary statistics that are useful for presenting findings and insights to stakeholders, colleagues, or clients. Visualizations are powerful tools for conveying complex information in an understandable manner.

Building Intuition: EDA allows analysts and data scientists to develop an intuition about the dataset. It helps in formulating hypotheses about relationships between variables and guides further investigation.

Code overview

 # Box plots for 'Amount' and 'Time' by class

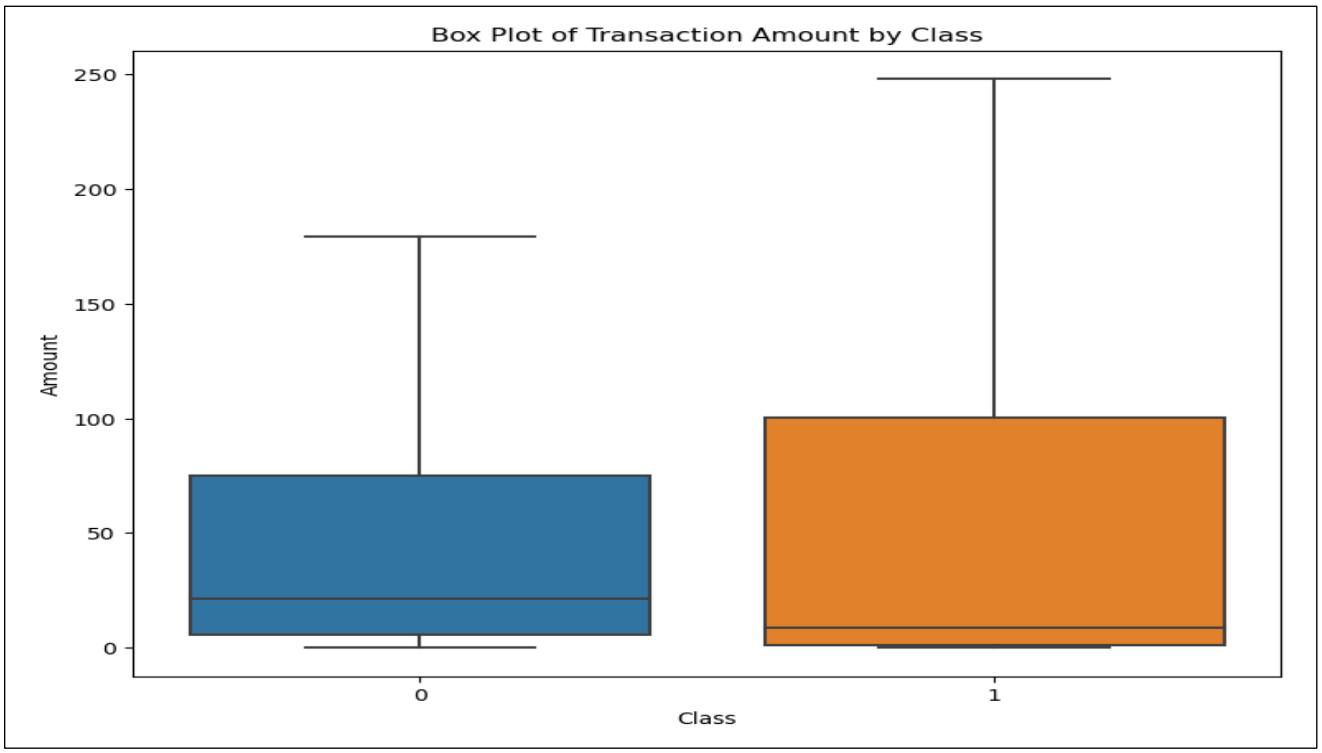
```
plt.figure(figsize=(9,7))
sns.boxplot(x='Class', y='Amount', data=df, showfliers=False)
plt.title('Box Plot of Transaction Amount by Class')
```

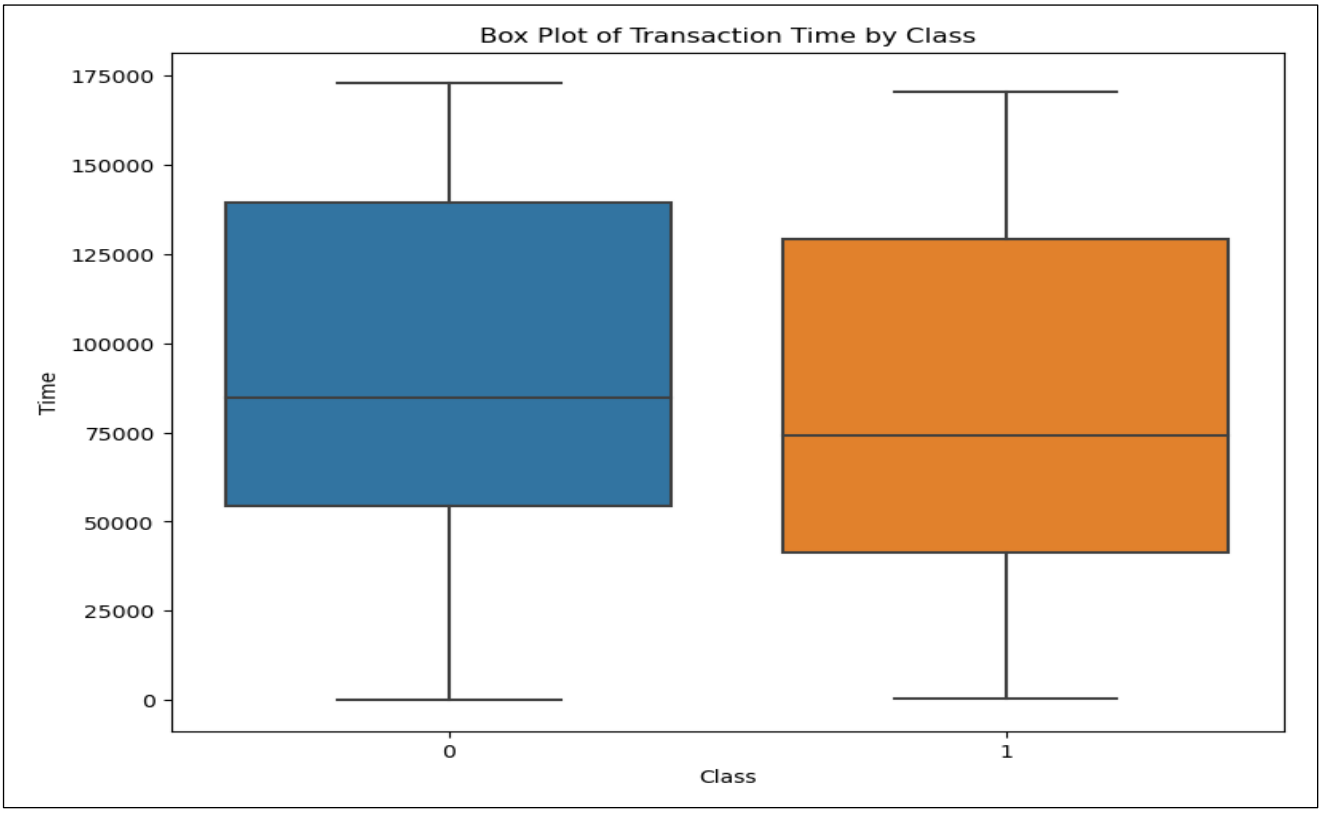
 # Box plots for 'Amount' and 'Time' by class

```
plt.figure(figsize=(9,7))
sns.boxplot(x='Class', y='Time', data=df, showfliers=False)
plt.title('Box Plot of Transaction Time by Class')
plt.show()
```

Explanation:

Box plots visualize the distribution of 'Amount' and 'Time' by transaction class.
Histograms explore the distributions of features V1 to V28.
A correlation heatmap illustrates the relationships between numerical features.





Correlation Heatmaps:

```
cor = df.corr()
fig = plt.figure(figsize =(12,10))
sns.heatmap(cor, vmax=.8, square = True)
plt.show()
```

Heatmaps are graphical representations of data where values in a matrix are represented using colors. They're used to visualize relationships, patterns, or distributions by assigning colors to data points in a two-dimensional array.

It can be used to:

Visualizing Correlation, Identifying Patterns in Large Datasets, Displaying Hierarchical Clustering, Understanding Relationships in Time-Series Data, Comparative Analysis , Feature Importance in Machine Learning

PairPlots:

Pairplots are used in data analysis and visualization to understand the relationship between multiple variables in a dataset. They are a grid of scatterplots arranged in a matrix format where each variable is plotted against every other variable.

Significance of Pairplots:

Exploring Relationships
Identifying Patterns

Detecting Outliers
Diagnosing Multicollinearity

PairPlots of all the features:

Codes for Pairplots:

1. plt.figure(figsize=(12,2))
sns.pairplot(df[['V1','V2','V3','V4','Class']],hue='Class')

2. plt.figure(figsize=(12,2))
sns.pairplot(df[['V1','V2','V3','V4','Class']],hue='Class')

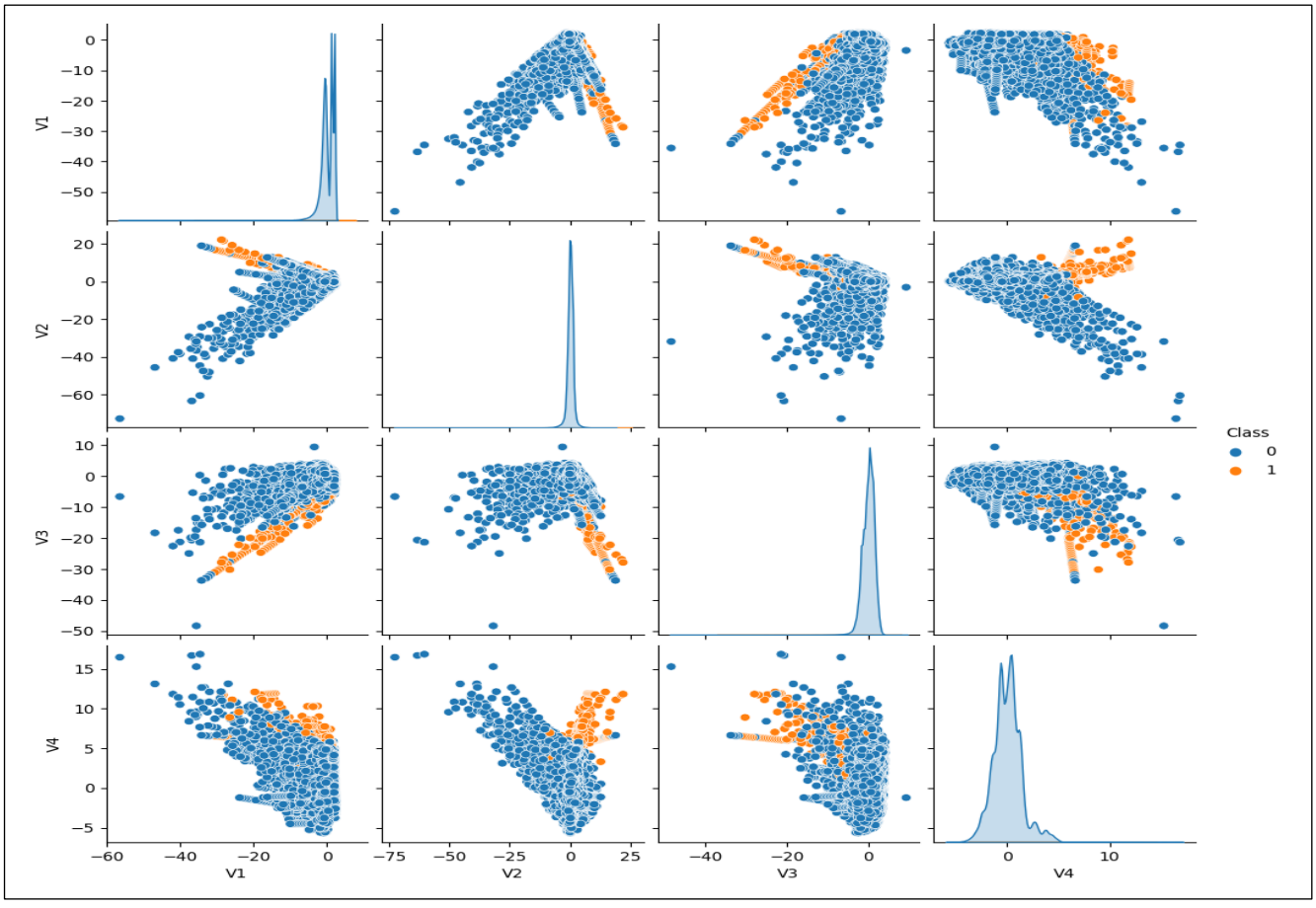
3. plt.figure(figsize=(12,2))
sns.pairplot(df[['V1','V2','V3','V4','Class']],hue='Class')

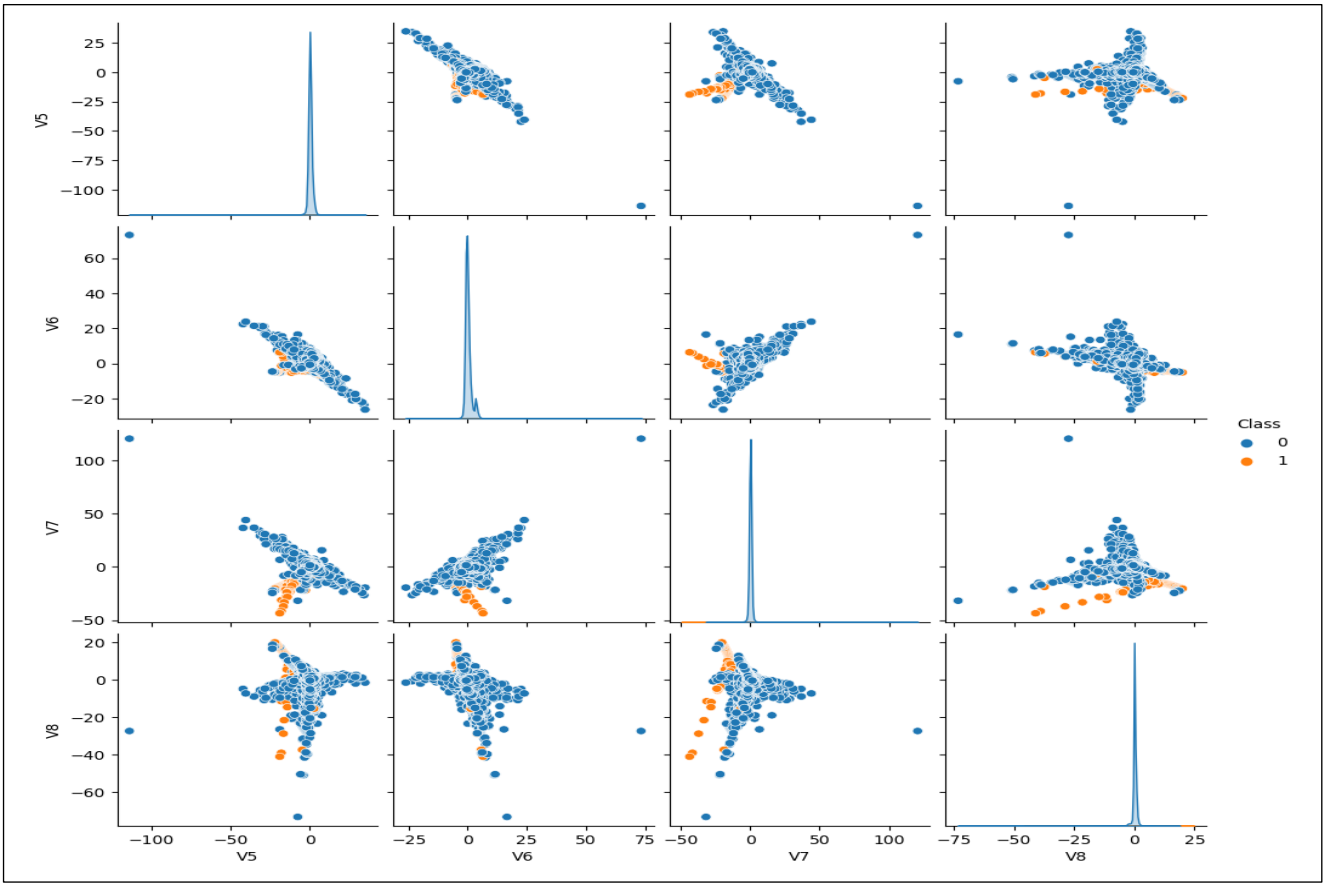
4. plt.figure(figsize=(12,2))
sns.pairplot(df[['V1','V2','V3','V4','Class']],hue='Class')

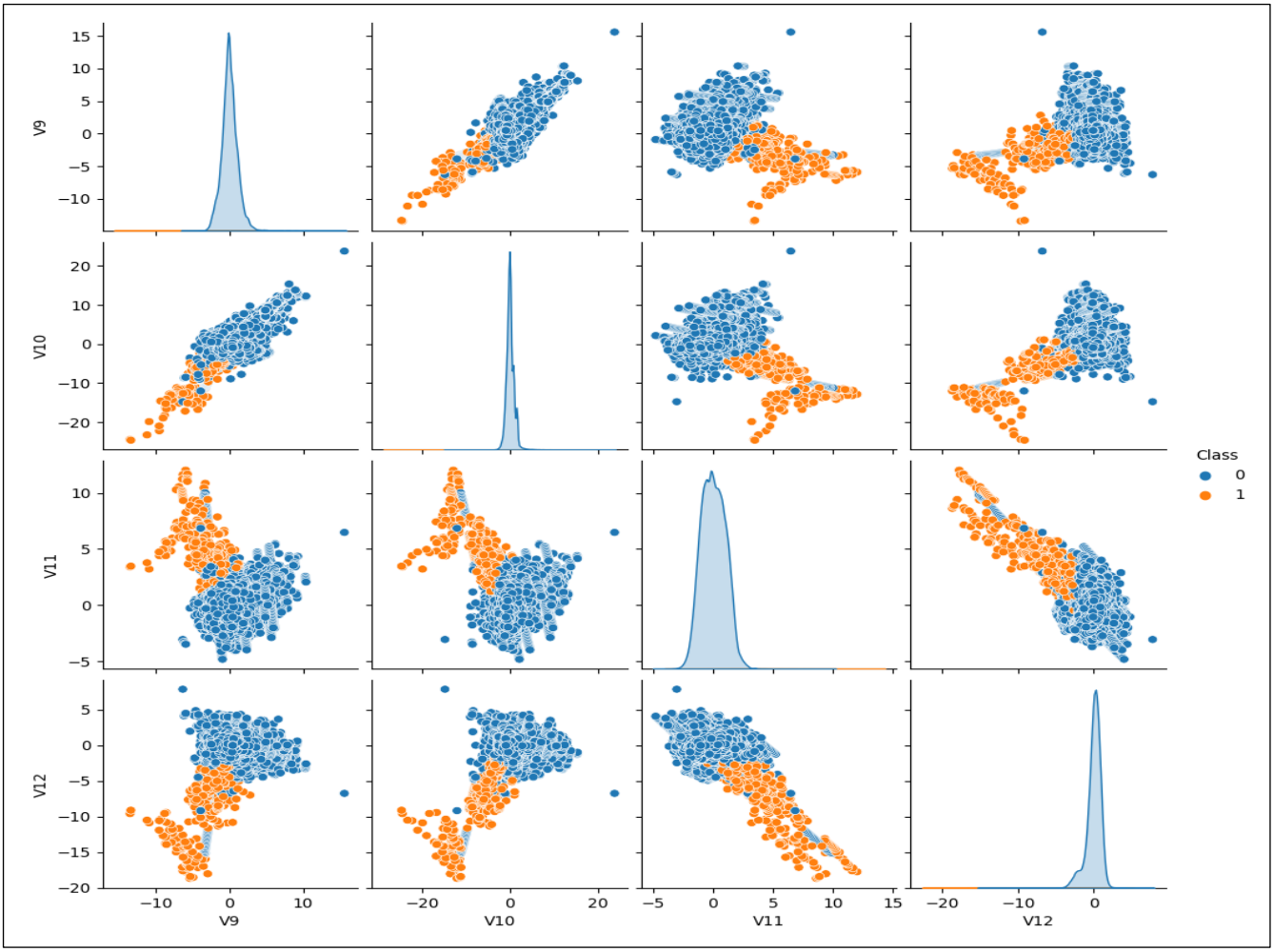
5. plt.figure(figsize=(12,2))
sns.pairplot(df[['V1','V2','V3','V4','Class']],hue='Class')

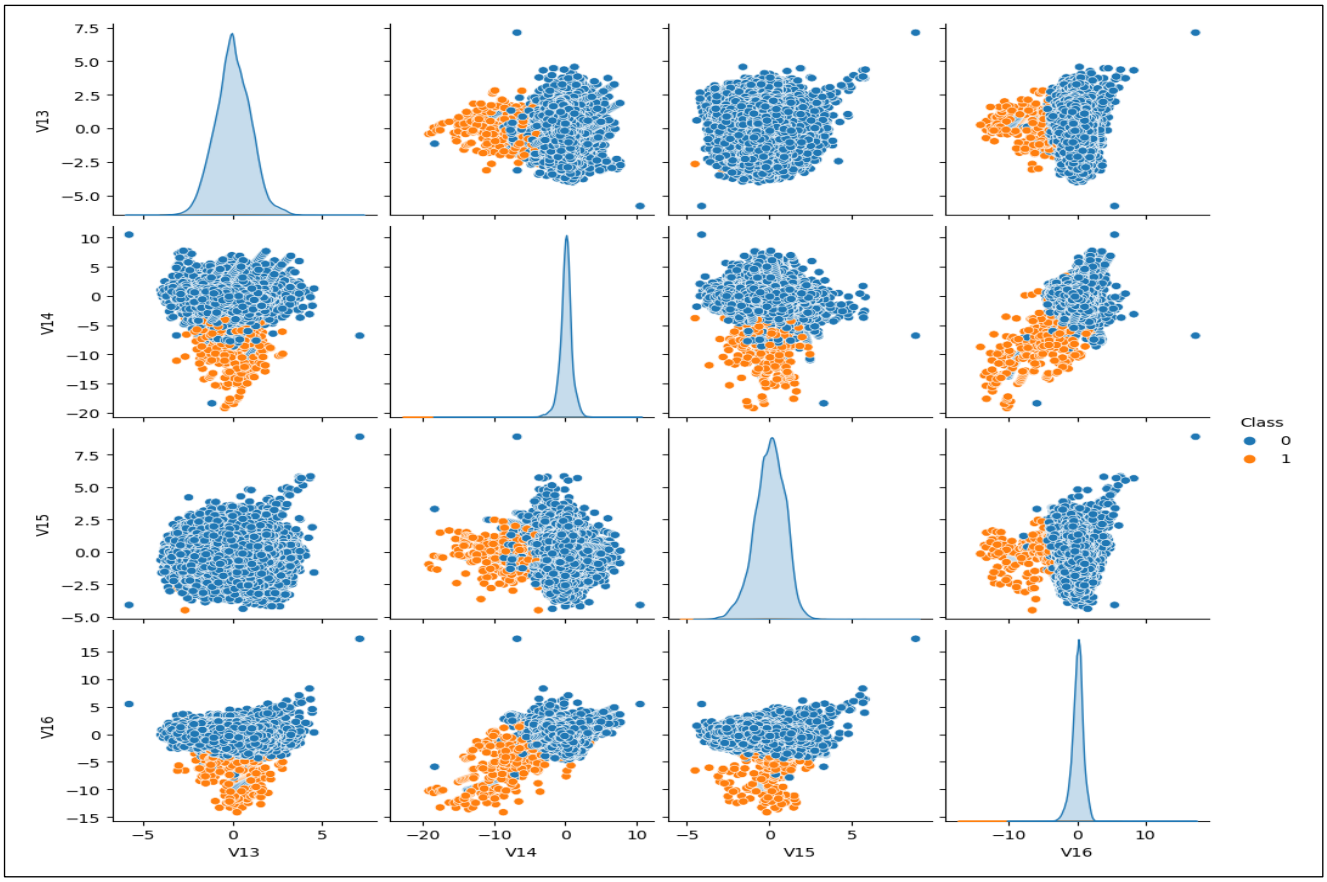
6. plt.figure(figsize=(12,2))
sns.pairplot(df[['V1','V2','V3','V4','Class']],hue='Class')

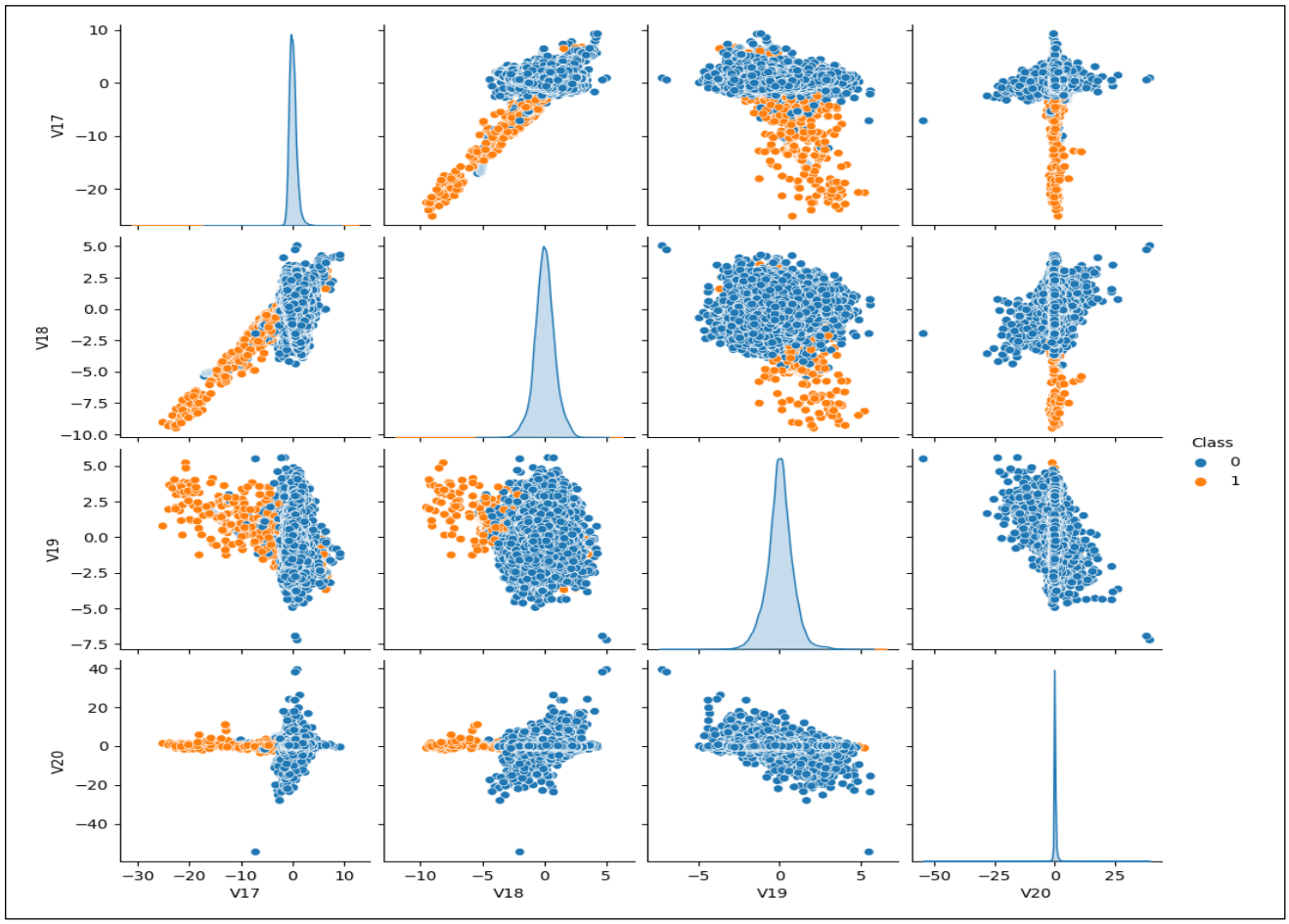
7. plt.figure(figsize=(12,2))
sns.pairplot(df[['V1','V2','V3','V4','Class']],hue='Class')

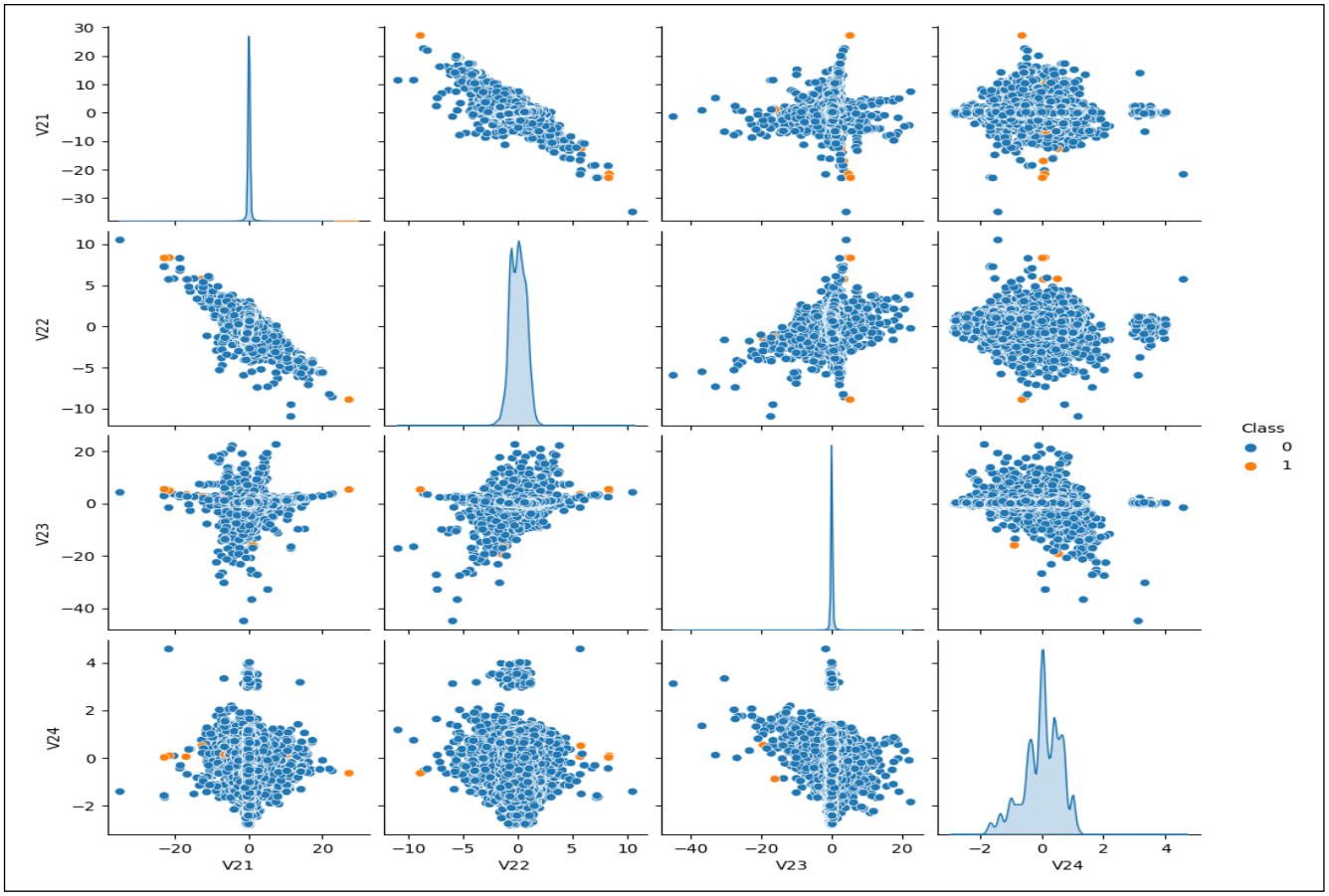


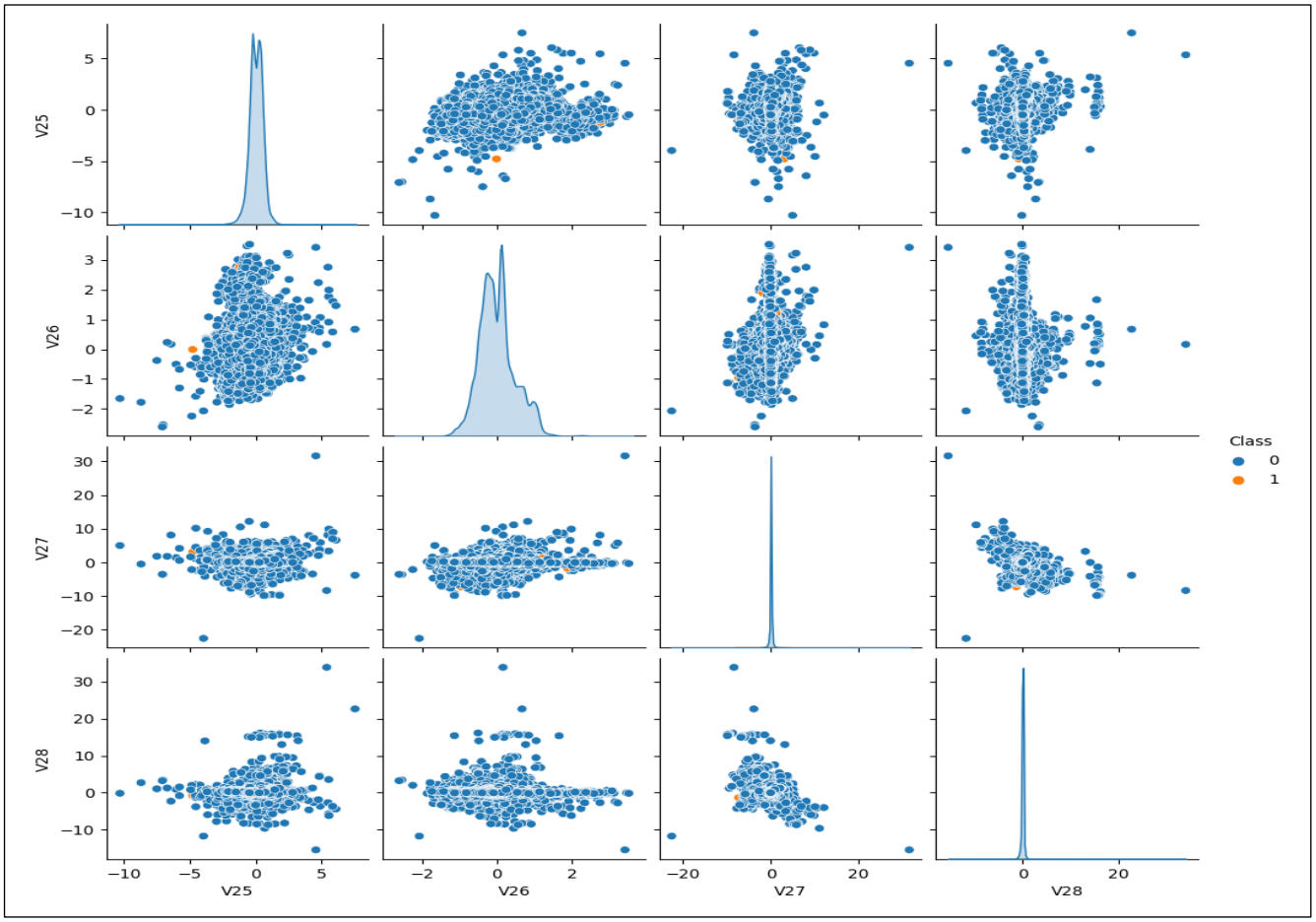

V5
V6
V7
V8
Class
0
1


V9
V10
V11
V12
Class
0
1


V13
V14
V15
V16
Class
0
1


V17
V18
V19
V20
Class
0
1


V21
V22
V23
V24
Class
0
1


V25
V26
V27
V28
Class
0
1

Statistical Analysis:

Statistical analysis plays a crucial role in predictive modeling for several reasons:
Data Understanding and Preprocessing
Feature Selection and Engineering
Model Selection
Model Interpretability
Model Evaluation
Assumptions Testing
Fine-tuning and Validation
Risk Assessment and Decision Making:

Code Overview:
df.describe()

Code Explanation:

This method in Pandas generates descriptive statistics summarizing the central tendency, dispersion, and shape of a DataFrame's numerical columns. It provides insights into the distribution and basic statistical properties of the data within each numerical column.
Here's what df.describe() displays:
Count: Number of non-null values present in each column.
Mean: The average value of the data.
Std (Standard Deviation): A measure of the amount of variation or dispersion in the data. It indicates how much the values deviate from the mean.
Min: The smallest value present in the column.
25th Percentile (Q1): Also known as the first quartile. It represents the value below which 25% of the data falls.
50th Percentile (Median or Q2): The median value, or the middle value when the data is sorted. Fifty percent of the data falls below and above this value.
75th Percentile (Q3): Also known as the third quartile. It represents the value below which 75% of the data falls.
Max: The largest value present in the column.

The statistical Analysis is as follows:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
count	283726.	283726.	283726.	283726.	283726.	283726.	283726.	283726.	283726.	283726.	283726.	...	283726.	283726.	283726.	283726.	283726.	283726.	283726.	283726.	283726.	283726.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class	
	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
mean	94811.077600	0.005917	-0.004135	0.001613	-0.002966	0.001828	-0.001139	0.001801	-0.000854	-0.001596	...		-0.000371	-0.000115	0.000198	0.000214	-0.000232	0.000149	0.000176	0.000547	88.472687	0.001667
std	47481.047891	1.948026	1.646703	1.508682	1.414184	1.377008	1.331931	1.227664	1.179054	1.095492	...		0.723909	0.724550	0.623702	0.605627	0.521220	0.482053	0.395744	0.328027	250.394437	0.040796
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307	-26.160506	-43.557242	-73.216718	-13.434066	...		-34.830382	-10.933144	-44.807735	-2.836627	-10.295397	-2.604551	-22.565679	-15.430084	0.000000	0.000000
25%	54204.750000	-0.915951	-0.600321	-0.889682	-0.850134	-0.689830	-0.769031	-0.552509	-0.208828	-0.644221	...		-0.228305	-0.542700	-0.161703	-0.354453	-0.317485	-0.326763	-0.070641	-0.052818	5.600000	0.000000

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class	
	00																				
50%	84692.500000	0.020384	0.063949	0.179963	-0.022248	-0.053468	-0.275168	0.040859	0.021898	-0.052596	...	-0.029441	0.006675	-0.011159	0.041016	0.016278	-0.052172	0.001479	0.011288	22.000000	0.000000
75%	139298.000000	1.316068	0.800283	1.026960	0.739647	0.612218	0.396792	0.570474	0.325704	0.595977	...	0.186194	0.528245	0.147748	0.439738	0.350667	0.240261	0.091208	0.078276	77.510000	0.000000
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666	73.301626	120.589494	20.007208	15.594995	...	27.202839	10.503090	22.528412	4.584549	7.519589	3.517346	31.612198	33.847808	25691.160000	1.000000

Model Building:

Model building refers to the process of creating and constructing a mathematical or computational representation (model) based on data to make predictions, infer patterns, or understand relationships between variables.

Splitting data into training and testing sets.
Implementation of Logistic Regression and Random Forest models.

Code Overview:

```
X = df.drop(['Class'], axis=1)
Y = df['Class']
print(X.shape)
print(Y.shape)
xData = X.values
yData = Y.values
```

Code Explanation:

This code Splittins data into training and testing sets. This step is very important for creating the model.
'X' will contain the independent variables or features for your machine learning model.
'Y' will contain the dependent variable or target variable for your model.

Training and Test Data Bifurcation:

We will divide the Datset into two main groups. One is for training model and other for testing the trained model's performance

'X' will contain the independent variables or features for your machine learning model.
'Y' will contain the dependent variable or target variable for your model.

Code Overview:

```
x_train,x_test,y_train,y_test = train_test_split(xData,yData, test_size = 0.3 , random_state = 42) #Building-a-Random-Forest-Model-using-a-scikit-Learn
```

Model Selection:

Logistic Regression

Logistic regression is a popular statistical method used for binary classification problems. There are several reasons why logistic regression might be selected as a model in various scenarios:

Interpretability: Logistic regression provides easily interpretable results. The model outputs probabilities that a given instance belongs to a particular class, and the coefficients associated with each feature can be interpreted as the impact of that feature on the likelihood of the outcome.

Efficiency with Linear Decision Boundaries: When the relationship between the features and the target variable is relatively linear or can be approximated as linear, logistic regression can perform well. It creates a linear decision boundary in feature space for binary classification tasks.

Low Variance, Less Prone to Overfitting: Logistic regression is less prone to overfitting, especially when there's a moderate amount of data and a limited number of features. It's less complex compared to some other machine learning models and tends to have lower variance.

Works Well with Small Datasets: Logistic regression can perform well even with smaller datasets compared to more complex models that might require larger amounts of data for training.

Computationally Inexpensive: Logistic regression tends to be computationally inexpensive compared to more complex algorithms. It can be trained quickly, making it efficient for datasets of moderate size.

Regularization Techniques: Regularization techniques such as L1 (Lasso) and L2 (Ridge) regularization can be applied to logistic regression to prevent overfitting and improve generalization.

Baseline Model: Logistic regression can serve as a baseline model against which more complex models can be compared. It provides a simple benchmark to evaluate the performance of other sophisticated models.

Code Overview:

```
log_reg = LogisticRegression(max_iter=1000)

log_reg.fit(x_train, y_train)
```

Predicting the output:

```
y_pred = log_reg.predict(x_test)
```

Model Evaluation

```
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy score: {accuracy*100:.2f}%")
```

Output:

```
Accuracy score: 99.91%
```

Report of the Classification Model:

A classification report is a tool used to evaluate the performance of a classification model. It provides a summary of various evaluation metrics that help assess how well the model is performing in predicting the different classes within a classification problem.

Code Overview:

```
print("Classification Report:")

print(classification_report(y_test, y_pred))
```

Output:

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

           0       1.00      1.00      1.00     84123
           1       0.86      0.60      0.70       144

    accuracy                           1.00     84267
   macro avg       0.93      0.80      0.85     84267
weighted avg       1.00      1.00      1.00     84267
```

Model Interpretability:

Code Overview:

```
feature_names = X.columns
coefficients = log_reg.coef_[0]
feature_importance_logistic = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})
feature_importance_logistic = feature_importance_logistic.sort_values(by='Coefficient', ascending=False)
print('\nLogistic Regression Coefficients:\n', feature_importance_logistic)
```

Output:

```
Logistic Regression Coefficients:
    Feature  Coefficient
7        V7     0.969509
22      V22     0.692359
21      V21     0.595796
```

1	V1	0.378607
12	V12	0.288421
20	V20	0.245841
23	V23	0.188124
4	V4	0.169694
26	V26	0.135452
18	V18	0.122559
28	V28	0.049391
19	V19	0.030146
0	Time	-0.000056
24	V24	-0.004189
29	Amount	-0.006551
6	V6	-0.030416
5	V5	-0.195569
27	V27	-0.208028
10	V10	-0.345390
16	V16	-0.429561
8	V8	-0.455938
2	V2	-0.478911
13	V13	-0.545726
17	V17	-0.545996
9	V9	-0.624116
11	V11	-0.686201
25	V25	-0.823343
14	V14	-0.843901
15	V15	-0.890329
3	V3	-1.181019

Explanation:

Model interpretability in logistic regression refers to the ease with which one can understand and explain how the model makes predictions based on its coefficients or parameters. Logisti c regression is known for its high interpretability due to its simplicity and transparent nature.

Random Forest:

Random Forest is a popular and powerful ensemble learning technique used in machine learning for both regression and classification tasks. There are several reasons why Random Forest might be chosen as a model:

High Predictive Accuracy: Random Forests generally yield high predictive accuracy. They work well for a variety of datasets and often outperform other algorithms in terms of predictive power.

Reduced Overfitting: By combining multiple decision trees (ensemble method), Random Forests can reduce overfitting compared to individual decision trees. The randomness injected during tree creation and feature selection helps to improve generalization.

Handles Large Datasets: Random Forests can handle large datasets with high dimensionality efficiently. They are robust against noise and perform well even with a large number of input features.

Feature Importance: They provide a measure of feature importance, indicating which features contribute more to the model's predictions. This feature ranking can aid in feature selection and understanding the dataset.

No Assumptions about Data Distribution: Random Forests don’t assume that the data follows a particular distribution. They can handle both numerical and categorical variables without the need for scaling or normalization.

Robustness to Outliers: Random Forests are robust to outliers and can handle missing values well, making them suitable for datasets with irregularities.

Parallelizable and Scalable: The training of individual trees in a Random Forest can be parallelized, which makes them scalable and efficient for large datasets.

Reduces Variance and Bias: Random Forests work by aggregating predictions from multiple trees. This ensemble approach helps to reduce variance and bias, leading to more stable and reliable predictions.

Code Overview:

```
cls = RandomForestClassifier()
cls.fit(x_train,y_train)
```

Checking the Accuracy of the model:

Code Overview:

```
acc = accuracy_score(y_test, y_pred)
print("Accuracy score : {}".format(acc*100))
```

Output:

```
Accuracy score : 99.96321217083793
```

Report of the model:

Code Overview:

```
print("Classification report : \n",classification_report(y_test,y_pred))
```

Output:

```
Classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     84123
```

```
              1       0.97      0.81      0.88       144

       accuracy                           1.00     84267
      macro avg       0.98      0.91      0.94     84267
   weighted avg       1.00      1.00      1.00     84267
```

SHAP Values: Random Forest

SHAP (SHapley Additive exPlanations) is a method used in machine learning for explaining individual predictions. It provides a unified approach to interpret the output of any machine learning model by attributing the prediction outcome to different features (input variables).

SHAP values are based on cooperative game theory and the concept of Shapley values, which originated from economics. These values offer a way to fairly distribute the "payout" among the features in a predictive model, considering their contributions to the prediction.

The key aspects of SHAP:

Local Interpretability: SHAP values aim to explain the prediction of a specific instance (local interpretation) rather than providing global model insights. They help to understand why a model made a particular prediction for an individual sample by attributing the prediction to the features.

Feature Importance: SHAP values provide an understanding of the impact of each feature on a particular prediction. They quantify how much each feature contributes to moving the model output from a base value (e.g., average model output) to the predicted value for a given instance.

Consistency and Additivity: SHAP values satisfy certain properties like consistency and additivity. Consistency implies that the sum of SHAP values for all features equals the difference between the model's output for a given instance and the average output of the model. Additivity means that the SHAP values add up to explain the prediction, ensuring fairness in feature attribution.

Visualizations: SHAP values can be visualized in various ways, such as summary plots (showing feature importance for an instance), force plots (illustrating how each feature contributes to the final prediction), and dependence plots (displaying the relationship between a feature and the SHAP values).

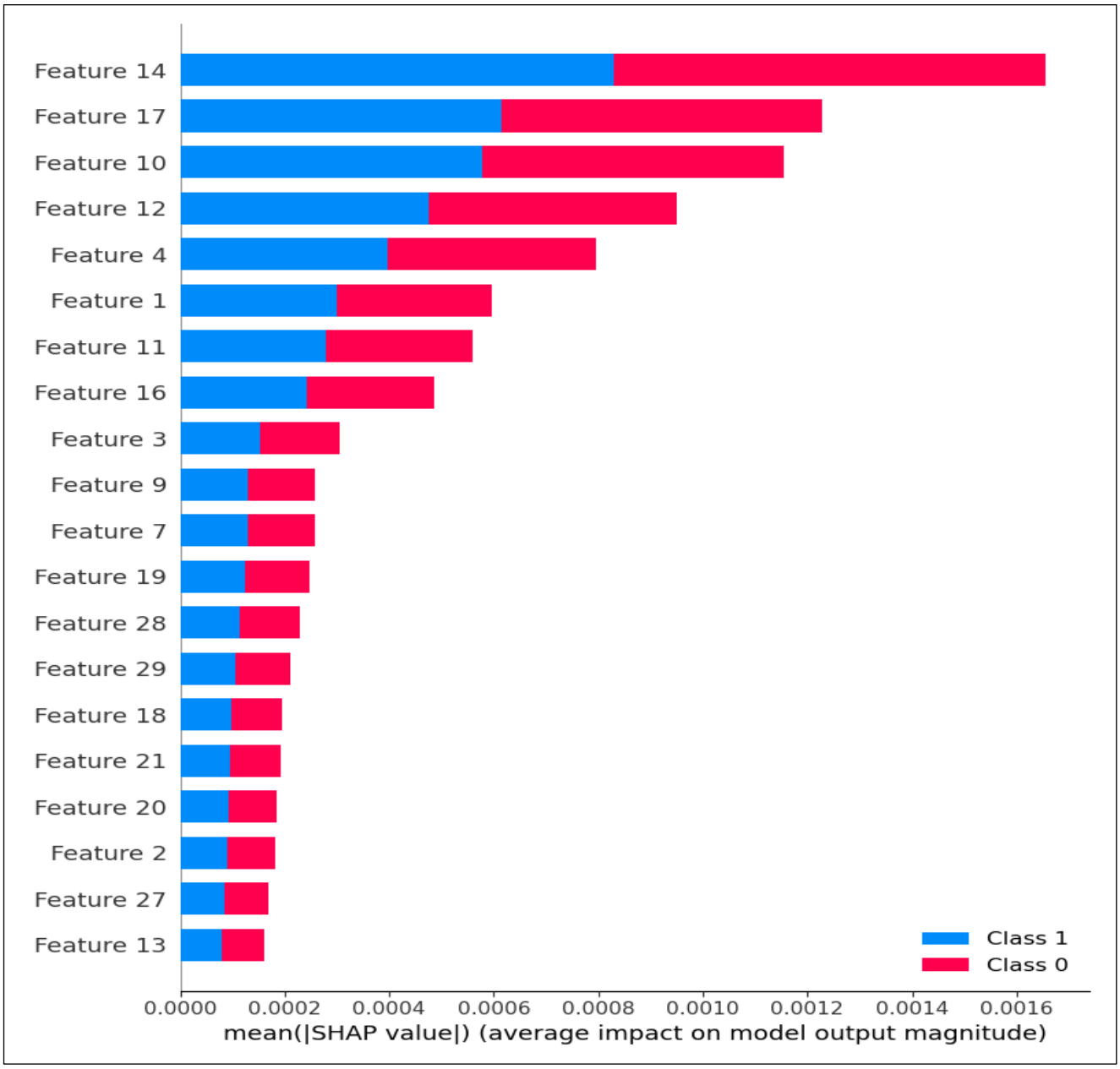
Applicability: SHAP values are model-agnostic, meaning they can be applied to any machine learning model, including tree-based models (Random Forests, Gradient Boosting), linear

models (Logistic Regression, Linear Regression), support vector machines, neural networks, and others.

SHAP values provide a powerful tool for explaining complex models and gaining insights into their decision-making processes. They are widely used in various domains, including healthcare, finance, and natural language processing, where interpretability of machine learning models is critical for decision-making, model debugging, and ensuring fairness and accountability.

Code Overview:

```
shap_values_rf = shap.TreeExplainer(cls).shap_values(x_train)
shap.summary_plot(shap_values_rf, x_train, plot_type='bar')
```



SHAP (SHapley Additive exPlanations) values for the Random Forest model are computed and visualized.

Model Serialization

Purpose:

This section saves the trained Logistic Regression and Random Forest models for future use.

Code Explanation:

```
import pickle

# Save Random Forest model to a file

with open('random_forestmodel.pkl', 'wb') as file:
    pickle.dump(cls.fit, file)
with open('logistic_regression.pkl', 'wb') as file:
    pickle.dump(cls.fit, file)
```

Explanation:
The trained models are saved to pickle files for later use.

Conclusion

The Credit Card Fraud Detection project has achieved its primary objective of developing accurate models for identifying fraudulent credit card transactions. Key insights and achievements from this project are summarized below:

Model Performance

High Accuracy: Both logistic regression and random forest models demonstrated exceptional accuracy in detecting fraudulent transactions across training and test datasets.
Comprehensive Metrics: Evaluation metrics, including AUC-ROC scores, provided a comprehensive understanding of the models' ability to distinguish between legitimate and fraudulent transactions.

Model Comparison

Thorough Comparison: A detailed comparison between logistic regression and random forest models was conducted, considering their strengths and weaknesses.
Performance Analysis: Classification reports and confusion matrices were employed to analyze and compare the performance of each model.

Interpretability

Feature Interpretation: Logistic regression coefficients and SHAP values were utilized to identify significant features, ensuring transparency in understanding the factors influencing model predictions.

Data Exploration

In-Depth Analysis: Extensive exploratory data analysis (EDA) enriched our understanding of legitimate and fraudulent transaction patterns.
Visualizations: Various visualizations such as box plots, histograms, and correlation heatmaps offered valuable insights into the dataset.

Data Preprocessing

Robust Techniques: Effective data preprocessing techniques, including outlier handling, feature scaling, and resolution of class imbalance through under-sampling, contributed to the models' effectiveness.

Model Serialization

Future Usability: Logistic regression and random forest models were serialized using the pickle library, ensuring they can be reused without the need for retraining.
In conclusion, this project not only delivered accurate fraud detection models but also provided crucial insights that can inform decision-making in the financial domain. The findings contribute significantly to ongoing efforts aimed at enhancing the security of credit card transactions.