**Fraud Detection System**

**Prepared By** : SHAILESH K R

**Submitted To** : M MUVENDIRAN (Infosys Springboard Internship Mentor)

**Subject** : A Report on Fraud Detection System Model until Logistic Regression.

Introduction

Fraud detection is a critical aspect of financial security and data analysis. This project utilizes machine learning and data visualization techniques to analyze credit card transactions, identifying fraudulent activities with precision. The dataset includes transaction details, features indicating the likelihood of fraud, and labels for classifying transactions as fraudulent or legitimate.

Project Description

This project involves advanced data handling and visualization techniques, providing insights into transaction patterns and anomalies. The objective is to analyze the features contributing to fraud detection, visualize their distributions, and identify potential outliers. The project incorporates methods for understanding class distribution and relationships between variables to aid machine learning algorithms.

Data Exploration

1. import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

The code imports key libraries for data analysis and visualization. **Seaborn** and **Matplotlib** handle advanced and basic plotting, respectively. **Pandas** is used for efficient data manipulation, while **NumPy** supports numerical computations and array operations. Together, they enable comprehensive data exploration and visualization.

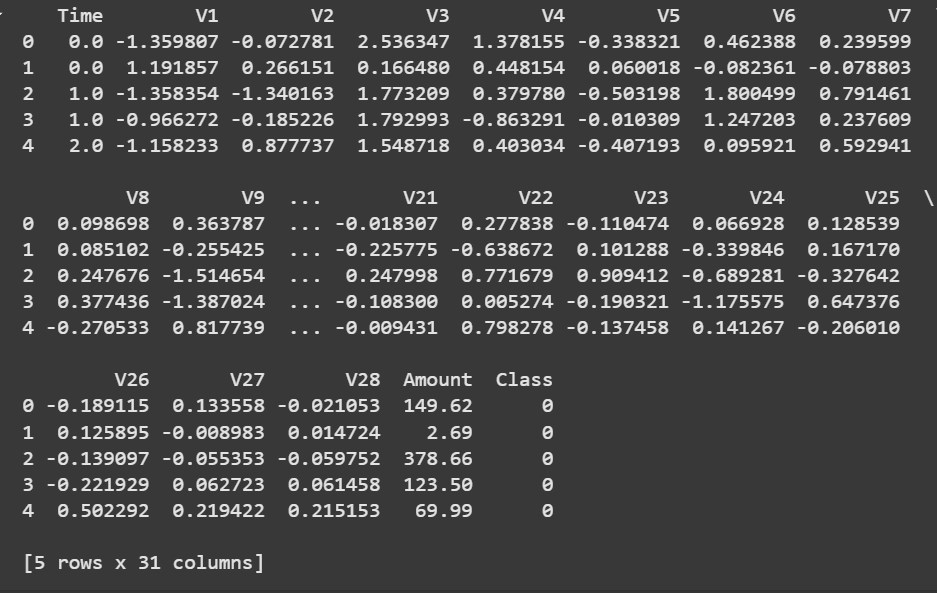
2) file\_path = '/content/creditcard.csv'

data = pd.read\_csv(file\_path)

The code loads the creditcard.csv dataset into a pandas DataFrame using the pd.read\_csv function. It specifies the file location through the file\_path variable and imports the data in a structured tabular format. This step prepares the dataset for subsequent analysis, visualization, and modeling.

3) print(data.head())

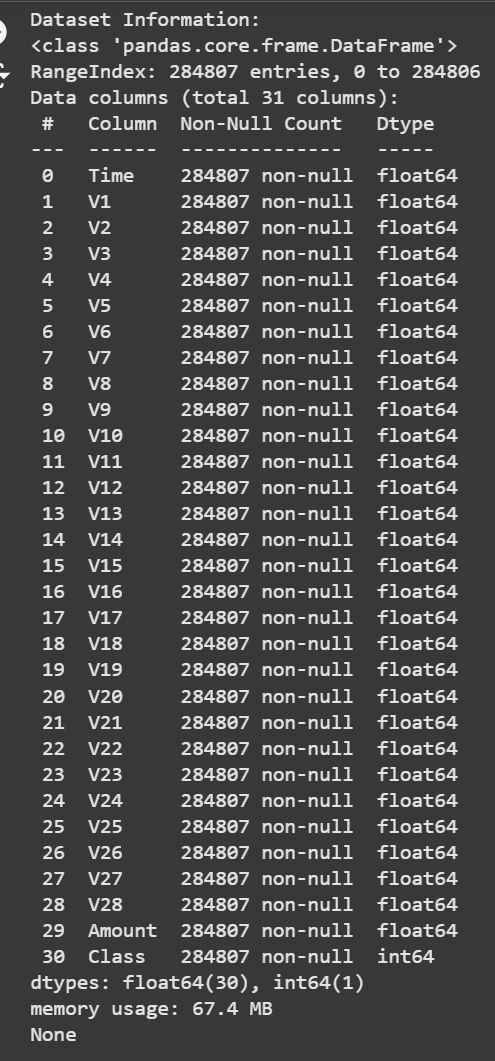
**Output :**



The code displays the first five rows of the dataset using the data.head() function. This provides a quick preview of the dataset, including its structure, column names, and a sample of the data values. It is a useful step to understand the dataset's format and contents before diving into detailed analysis.

4) print(data.info())

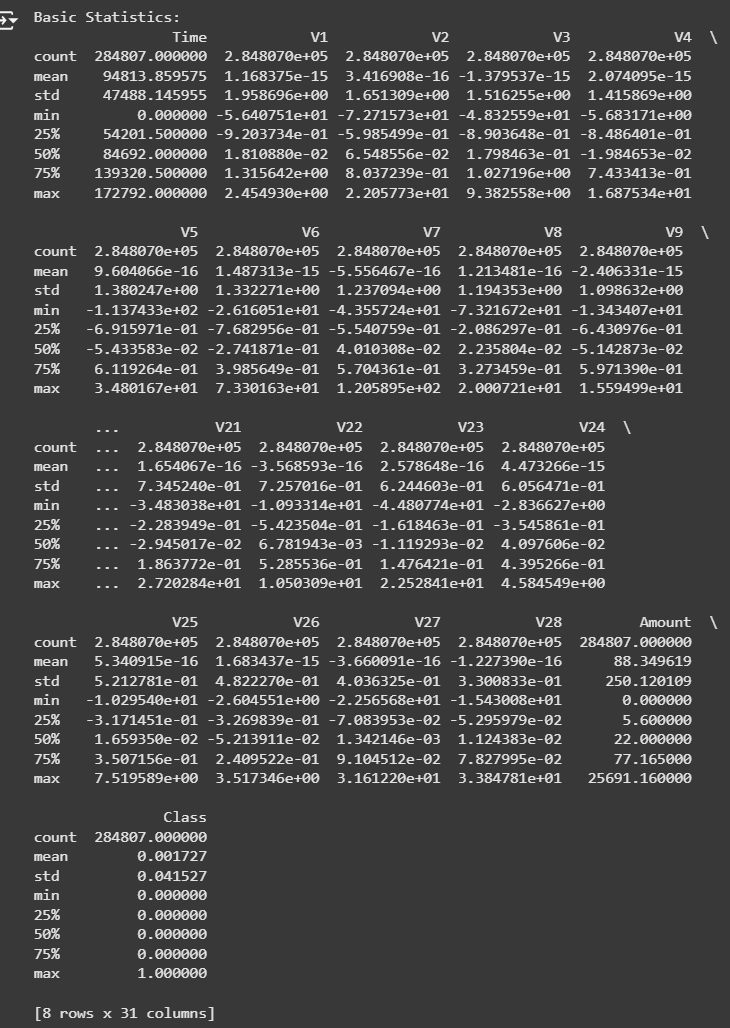
**Output:**



The data.info() function provides a concise summary of the dataset, including the number of rows and columns, column names, and data types. It also indicates the non-null count for each column, helping to identify any missing data. This step is essential for understanding the dataset's structure and preparing it for analysis.

5) print(data.describe())

**Output:**



The data.describe() function generates summary statistics for the numerical columns in the dataset. It includes metrics like count, mean, standard deviation, minimum, maximum, and quartiles (25th, 50th, and 75th percentiles). This helps in understanding the data's distribution, variability, and potential anomalies.

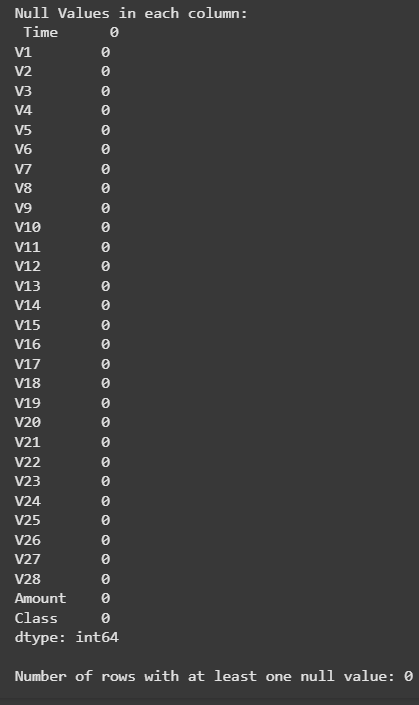
6) null\_values = data.isnull().sum()

6) print("\nNull Values in each column:\n", null\_values)

rows\_with\_null = data[data.isnull().any(axis=1)]

print("\nNumber of rows with at least one null value:", len(rows\_with\_null))

**Output:**



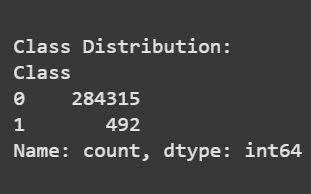
This code checks for missing data in the dataset. It calculates the total number of null values in each column using data.isnull().sum() and prints the results. Additionally, it identifies rows with at least one null value using data.isnull().any(axis=1) and displays their count. This step ensures data completeness before proceeding with analysis or modeling.

7) class\_counts = data['Class'].value\_counts()

print("\nClass Distribution:")

print(class\_counts)

**Output:**



The code calculates the distribution of the target variable Class using data['Class'].value\_counts(). It displays the count of each class, where typically 0 represents non-fraudulent transactions and 1 represents fraudulent ones. This step is crucial for understanding the dataset's class imbalance, which can significantly impact model training and evaluation.

8) def find\_outliers\_iqr(data, column):

Q1 = data[column].quantile(0.25)

Q3 = data[column].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

return len(data[(data[column] < lower\_bound) | (data[column] > upper\_bound)])

columns\_to\_check = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'Amount']

total\_outliers = sum(find\_outliers\_iqr(data, col) for col in columns\_to\_check)

print(f"Total number of outliers: {total\_outliers}")

**Output:**



This code defines a function, find\_outliers\_iqr, that uses the Interquartile Range (IQR) method to detect outliers in a specified column. The function calculates the lower and upper bounds based on 1.5 times the IQR and counts the data points outside these bounds. It then iterates over selected columns (V1 to V10 and Amount) to calculate the total number of outliers across these features. Finally, it prints the total count of outliers, providing insight into the dataset's anomalies.

Data Visualisations and Insights

9) plt.figure(figsize=(12, 10))

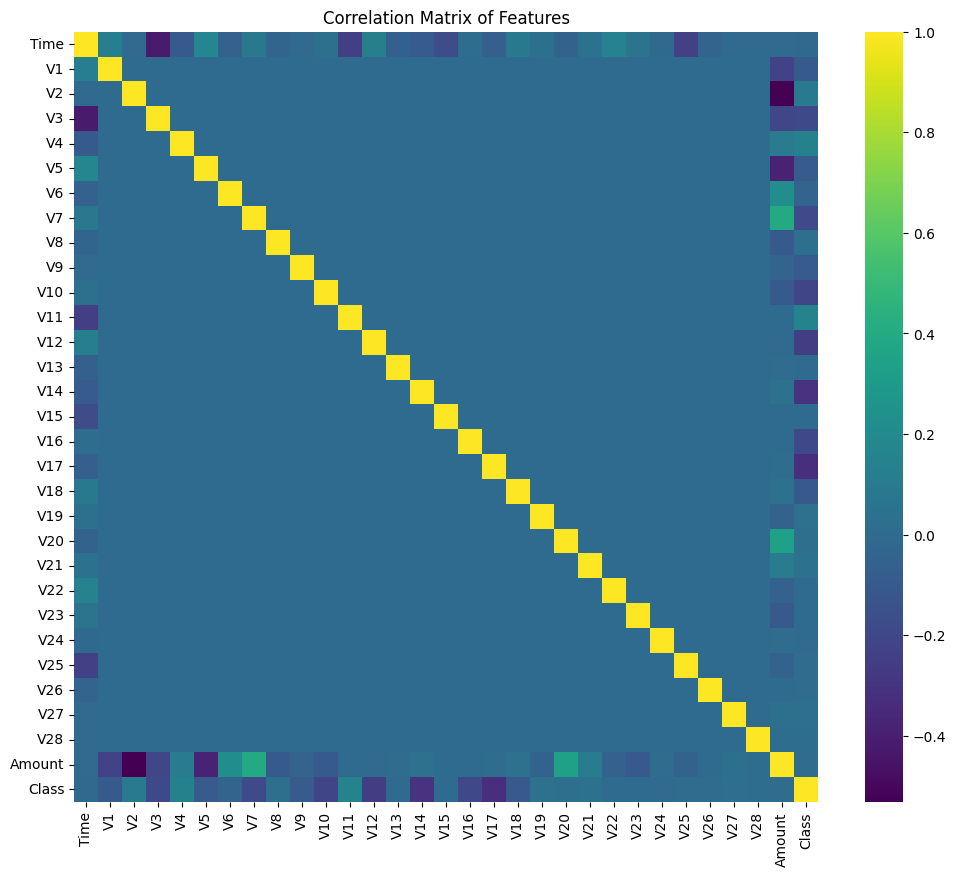
correlation\_matrix = data.corr()

sns.heatmap(correlation\_matrix, annot=False, cmap='viridis', fmt=".2f")

plt.title('Correlation Matrix of Features')

plt.show()

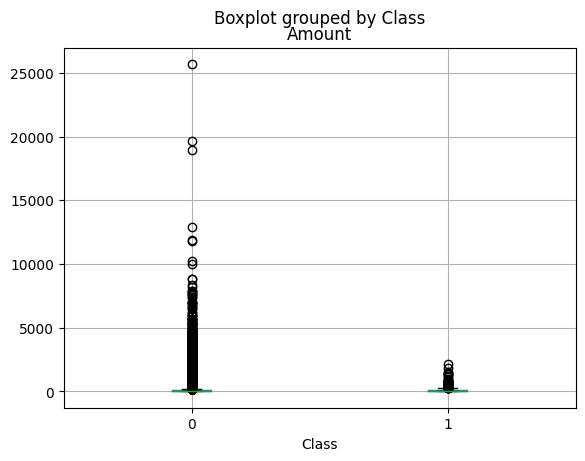
**Output:**



This code generates a heatmap to visualize the correlation matrix of the dataset's numerical features. The data.corr() function calculates the pairwise correlation coefficients between features, and sns.heatmap displays these correlations in a heatmap format. The annot=False argument disables annotations within the heatmap, while the cmap='viridis' argument sets the color scheme. This visualization helps identify relationships between features, which can be useful for feature selection and understanding the data's structure.

10) data.boxplot(column=['Amount'], by='Class')

**Output:**



This code creates a boxplot to visualize the distribution of the Amount feature, grouped by the Class variable (fraud vs. non-fraud). The boxplot() function helps identify the spread, median, and potential outliers in the transaction amounts for both fraudulent (Class 1) and non-fraudulent (Class 0) transactions. It is a useful visualization to compare the transaction amount distributions between the two classes.

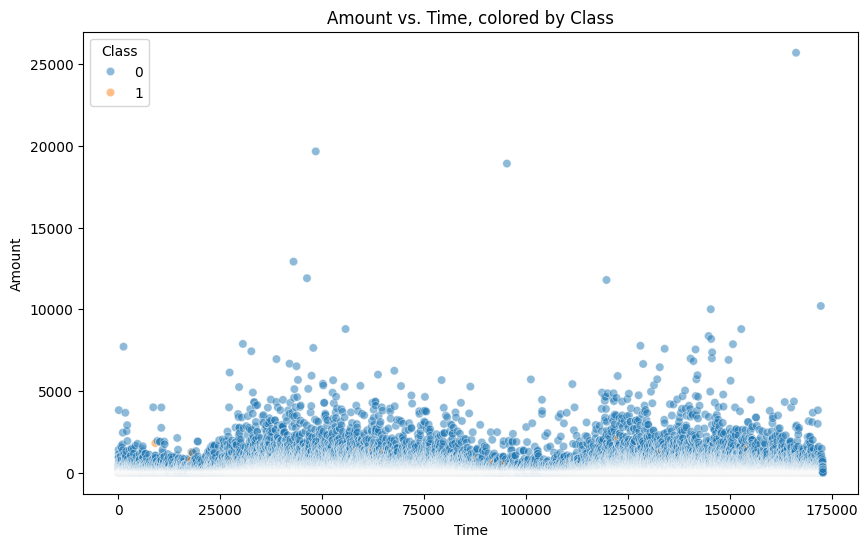
11) plt.figure(figsize=(10, 6))

sns.scatterplot(x='Time', y='Amount', hue='Class', data=data, alpha=0.5)

plt.title('Amount vs. Time, colored by Class')

plt.show()

**Output:**



This code generates a scatter plot to visualize the relationship between the Time and Amount features, with points colored based on the Class variable (fraudulent vs. non-fraudulent transactions). The sns.scatterplot function plots Amount on the y-axis and Time on the x-axis, while hue='Class' colors the points to distinguish between the two classes. The alpha=0.5 parameter adjusts the transparency of the points. This plot helps observe any trends or patterns in the transaction amounts over time, highlighting differences between fraudulent and non-fraudulent transactions.

12) plt.figure(figsize=(10, 6))

sns.kdeplot(data.loc[data['Class'] == 0, 'Amount'], label='Non-Fraud', shade=True)

sns.kdeplot(data.loc[data['Class'] == 1, 'Amount'], label='Fraud', shade=True)

plt.title('Distribution of Transaction Amount by Class')

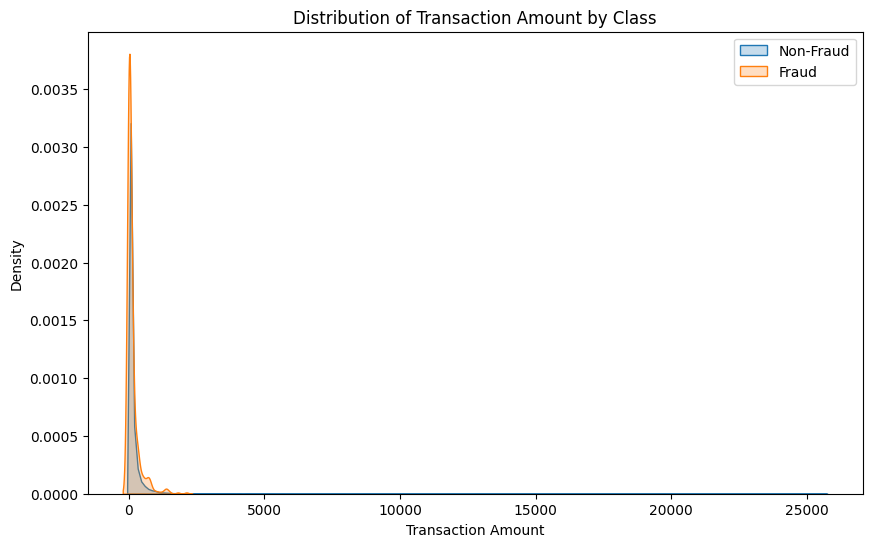
plt.xlabel('Transaction Amount')

plt.ylabel('Density')

plt.legend()

plt.show()

**Output:**



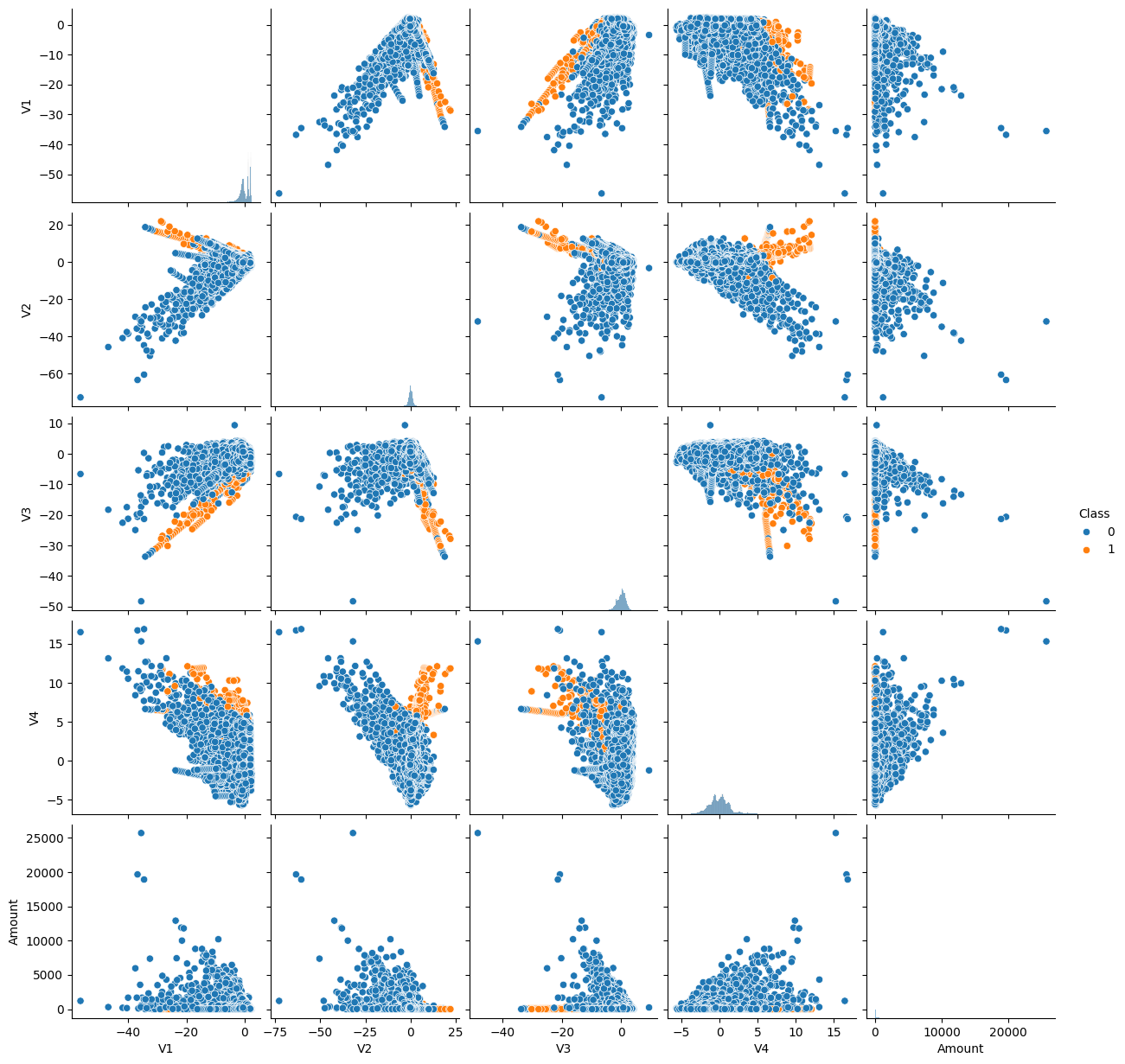
This code generates a Kernel Density Estimate (KDE) plot to visualize the distribution of transaction amounts (Amount) for both non-fraudulent (Class 0) and fraudulent (Class 1) transactions. The sns.kdeplot function is used to estimate the probability density function of the Amount feature, with shading to highlight the areas under the curves. The label parameter differentiates between the two classes in the legend. This plot helps compare the distribution of transaction amounts across the two classes, revealing any differences in patterns between fraudulent and non-fraudulent transactions.

13) selected\_features = ['V1', 'V2', 'V3', 'V4', 'Amount', 'Class']

sns.pairplot(data[selected\_features], hue='Class', diag\_kind='hist')

plt.show()

**Output:**



This code creates a pairplot to visualize the relationships between selected features (V1, V2, V3, V4, Amount, and Class). The sns.pairplot function plots pairwise relationships between the features, and the hue='Class' argument colors the points by the Class variable, differentiating between fraudulent and non-fraudulent transactions. The diag\_kind='hist' argument sets histograms along the diagonal to show the distribution of each feature. This visualization helps to understand how features correlate with each other and how they are distributed across the two classes.

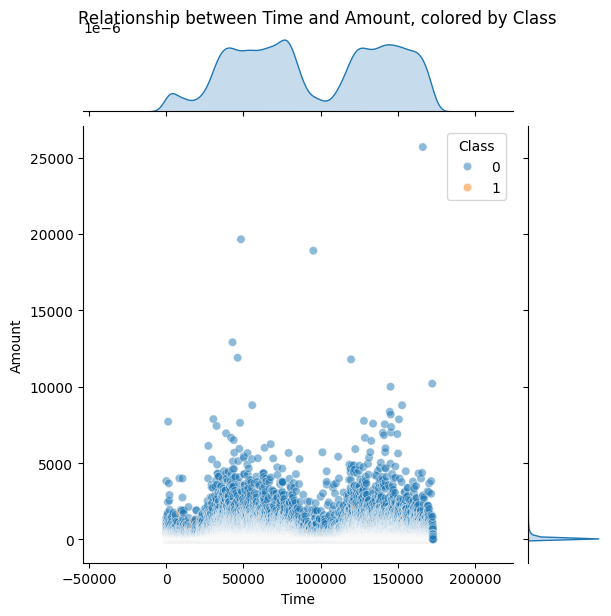
14) plt.figure(figsize=(10, 6))

sns.jointplot(x='Time', y='Amount', data=data, hue='Class', kind='scatter', alpha=0.5)

plt.suptitle('Relationship between Time and Amount, colored by Class', y=1.02)

plt.show()

**Output:**



This code generates a jointplot to visualize the relationship between Time and Amount, with points colored by the Class variable (fraud vs. non-fraud). The sns.jointplot function creates a scatter plot to display the relationship between these two features, with the hue='Class' argument differentiating the classes by color. The alpha=0.5 parameter adjusts the transparency of the points. This plot is useful for observing how the transaction time and amount interact, and whether there are any visible patterns or trends between fraudulent and non-fraudulent transactions.

15) for col in ['V1', 'V2', 'V3', 'Amount']:

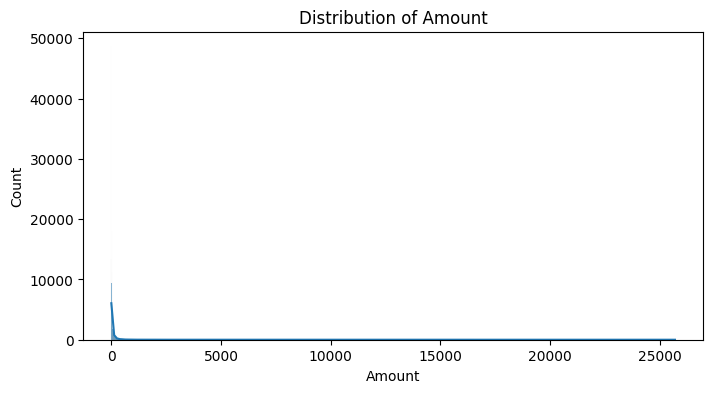
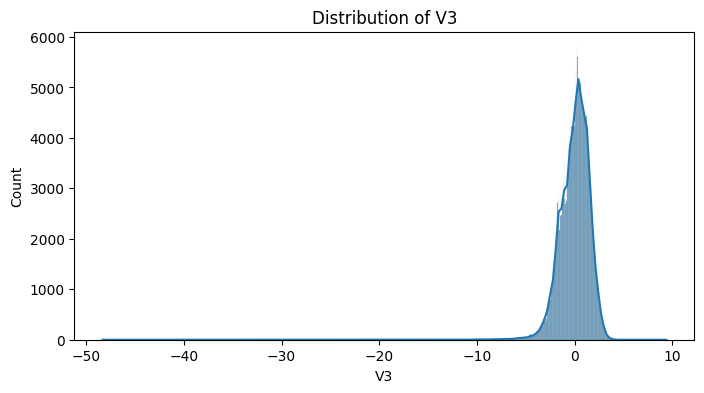
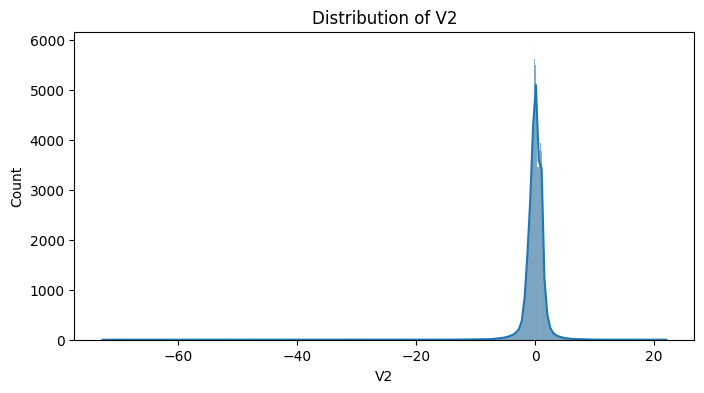
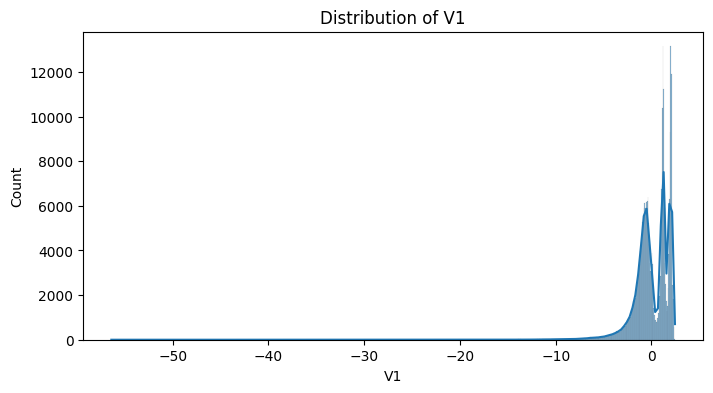
plt.figure(figsize=(8, 4))

sns.histplot(data[col], kde=True)

plt.title(f'Distribution of {col}')

plt.show()

**Output:**



This code generates individual histograms with Kernel Density Estimate (KDE) plots for the selected features (V1, V2, V3, and Amount). The sns.histplot function is used to visualize the distribution of each feature, with the kde=True argument adding a smooth curve to represent the probability density. Each plot is displayed with a title corresponding to the feature being visualized. This is useful for understanding the distribution, skewness, and potential outliers in the features, which can inform further analysis or preprocessing steps.

15) plt.figure(figsize=(6, 4))

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

plt.title('Class Distribution (Fraud vs. Non-Fraud)')

plt.show()

**Output:**



Data Preprocessing

Data preprocessing is a critical step in building a robust fraud detection system. It ensures the dataset is clean, consistent, and ready for analysis or model training. The following steps from the code highlight the preprocessing techniques used in this project

1) X = data.iloc[:, data.columns != 'Class']

y = data.iloc[:, data.columns == 'Class']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

X\_train\_cv = X\_train.copy()

y\_train\_cv = y\_train.copy()

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train['normalizedAmount'] = scaler.fit\_transform(X\_train['Amount'].values.reshape(-1, 1))

X\_test['normalizedAmount'] = scaler.transform(X\_test['Amount'].values.reshape(-1, 1))

X\_train = X\_train.drop(['Amount'], axis=1)

X\_test = X\_test.drop(['Amount'], axis=1)

y.head()

X\_train.shape

X\_test.shape

Y\_train.shape

Y\_test.shape

**Output:**

**(199364, 29)**

(85443, 29)

(199364, 1)

(85443, 1)

This code splits the dataset into features (X) and labels (y), then further divides them into training and testing sets using a 70-30 split. It normalizes the "Amount" feature using StandardScaler, creating a new column normalizedAmount for scaled values, and subsequently drops the original "Amount" column. The resulting training and testing datasets are prepared for model training and evaluation, ensuring consistent feature scaling.

Data Modelling

The model is built using a RandomForestClassifier, trained on the preprocessed training dataset (X\_train and y\_train) to classify transactions. It is evaluated on the test set using metrics like accuracy, precision, recall, and a confusion matrix. Hyperparameter tuning with GridSearchCV optimizes the model’s performance through cross-validation.

1) from imblearn.over\_sampling import SMOTE

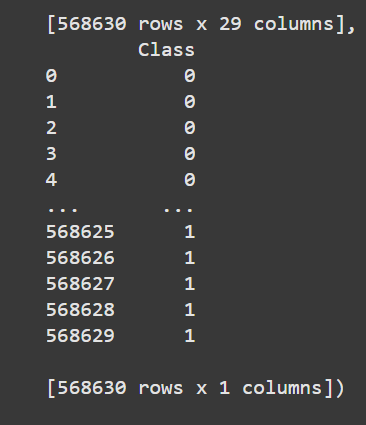
smote = SMOTE()

X\_resample, y\_resample = smote.fit\_resample(X, y)

y\_resample

This code applies **SMOTE (Synthetic Minority Oversampling Technique)** to address class imbalance in the dataset. By generating synthetic samples for the minority class, it balances the feature set (X) and target labels (y). The resampled datasets, X\_resample and y\_resample, contain an equal representation of both classes, enhancing the model's ability to detect fraudulent transactions effectively.

**Output:**



LOGISTIC REGRESSION

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc, accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, classification\_report, confusion\_matrix

from sklearn.linear\_model import LogisticRegression

import seaborn as sns

# Visualizations for Logistic Regression

y\_pred\_logreg = log\_reg.predict(X\_test)

y\_prob\_logreg = log\_reg.predict\_proba(X\_test)[:, 1]

# 1. ROC Curve

fpr\_logreg, tpr\_logreg, \_ = roc\_curve(y\_test, y\_prob\_logreg)

roc\_auc\_logreg = auc(fpr\_logreg, tpr\_logreg)

plt.figure(figsize=(8, 6))

plt.plot(fpr\_logreg, tpr\_logreg, color='darkorange', lw=2, label=f'ROC curve (area = {roc\_auc\_logreg:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve - Logistic Regression')

plt.legend(loc="lower right")

plt.show()

# 2. Confusion Matrix

cm\_logreg = confusion\_matrix(y\_test, y\_pred\_logreg)

plt.figure(figsize=(6, 6))

sns.heatmap(cm\_logreg, annot=True, fmt='d', cmap='Blues',

xticklabels=['Non-Fraud', 'Fraud'], yticklabels=['Non-Fraud', 'Fraud'])

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix - Logistic Regression')

plt.show()

# 3. Accuracy, Precision, Recall, F1-Score

accuracy\_logreg = accuracy\_score(y\_test, y\_pred\_logreg)

precision\_logreg = precision\_score(y\_test, y\_pred\_logreg)

recall\_logreg = recall\_score(y\_test, y\_pred\_logreg)

f1\_logreg = f1\_score(y\_test, y\_pred\_logreg)

print(f"Logistic Regression - Accuracy: {accuracy\_logreg:.4f}")

print(f"Logistic Regression - Precision: {precision\_logreg:.4f}")

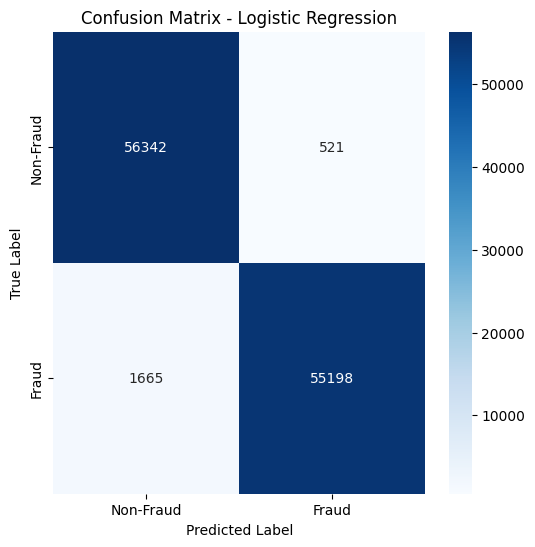
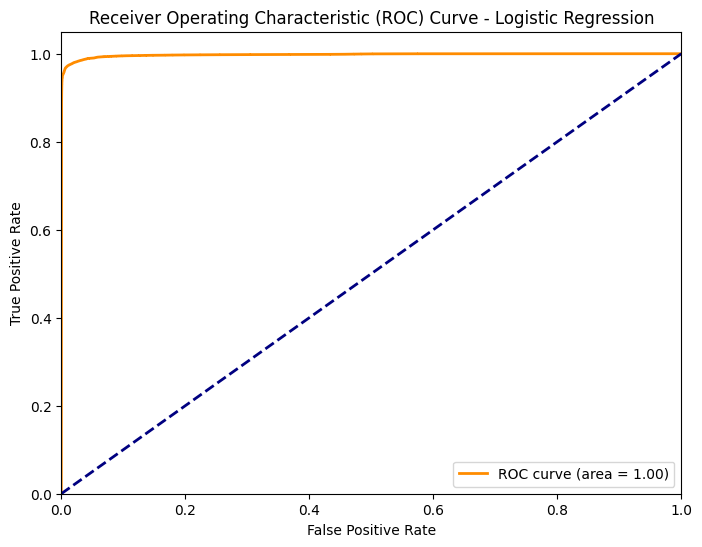
print(f"Logistic Regression - Recall: {recall\_logreg:.4f}")

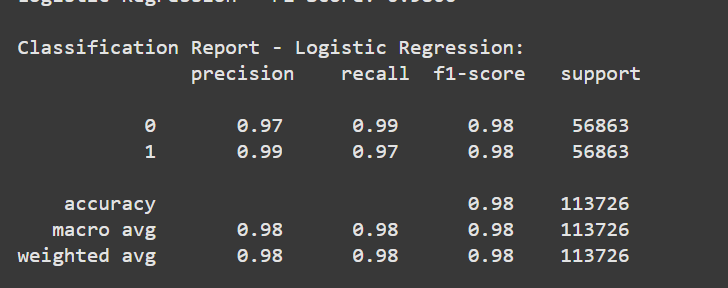
print(f"Logistic Regression - F1-Score: {f1\_logreg:.4f}")

# 4. Classification Report

print("\nClassification Report - Logistic Regression:\n", classification\_report(y\_test, y\_pred\_logreg))

**Output:**





This code evaluates the performance of a Logistic Regression model by generating key visualizations and performance metrics. It plots the ROC curve to assess classification accuracy, displays a confusion matrix to show true vs. predicted values, and calculates key metrics such as accuracy, precision, recall, and F1-score. Finally, it prints a classification report for a detailed overview of the model's performance.

Conclusion

The fraud detection system leverages statistical analysis and visualization techniques to identify patterns and anomalies in transaction data. By addressing class imbalances and detecting outliers, this project lays the groundwork for developing robust machine learning models for automated fraud detection. The analysis reveals key insights into fraudulent transaction characteristics, offering significant value for enhancing financial security.

Colab Link  
Access the notebook for this analysis   
<https://colab.research.google.com/drive/11qJiuS_Y9KOc47SqhwzFgUA2EGVXrXdD#scrollTo=XtLvt89oDkwO>