**Salman Irfan**

**2024-Phd-Cs-5**

**CS-608 Advanced Techniques in Data Science**

**Course Project: Credit Card Fraud Detection**

**Submitted To**

**Prof. Dr. Muhammad Awais Hassan**

1. Select a Project Domain and Area: -

**Domain:** Financial services (Credit Card Transactions).

**Area:** Fraud detection in credit card transactions using machine learning.

1. Identify the Problem: -

Credit card companies need to detect fraudulent transactions with high accuracy while minimizing the number of false positives (legitimate transactions marked as fraud).

1. Ask Questions to Answer: -
2. Which transaction patterns or behaviors typically indicate fraud?

* **Why Ask**: Fraudulent transactions often follow certain unusual patterns, such as unusual spending amounts or multiple transactions within a short time frame.
* **Expected Insight**: Identify behaviors like high transaction volumes in a short time, geographical anomalies, or sudden changes in spending patterns.

1. What role does the transaction time play in identifying fraud?

* **Why Ask:** Fraud may occur during specific hours when detection is harder (e.g., late-night hours).
* **Expected Insight**: Certain times of day may show higher fraud rates, which could inform temporal-based fraud detection methods.

1. Are there specific amounts of transactions that are more prone to fraud?

* **Why Ask**: Fraudsters might target small amounts to evade detection or large amounts for bigger payouts.
* **Expected Insight:** By analyzing transaction amounts, we can determine if certain ranges are more susceptible to fraud.

1. How can we detect fraud without overwhelming the system with false positives?

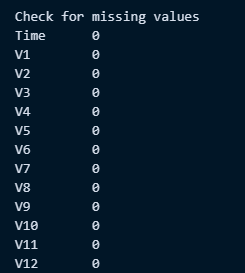
* **Why Ask**: False positives (legitimate transactions marked as fraud) can frustrate customers. We need to balance fraud detection with accuracy.
* Expected Insight: Use techniques like precision-recall trade-offs or resampling methods to improve model performance on rare events like fraud.

1. Collect or Find Data: -

* Data Source: Kaggle's credit card fraud detection dataset.
* Source: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?select=creditcard.csv>

1. Data Wrangling (Preprocessing): -
2. # *2. Data Wrangling (Preprocessing)*
3. # *Check for missing values*
4. print("Check for missing values")
5. print(df.isnull().sum())

Output: -



A screenshot of a computer program

Description automatically generated

**Handle Outliers: -**

# *handle Outliers*

print("handle outliers")

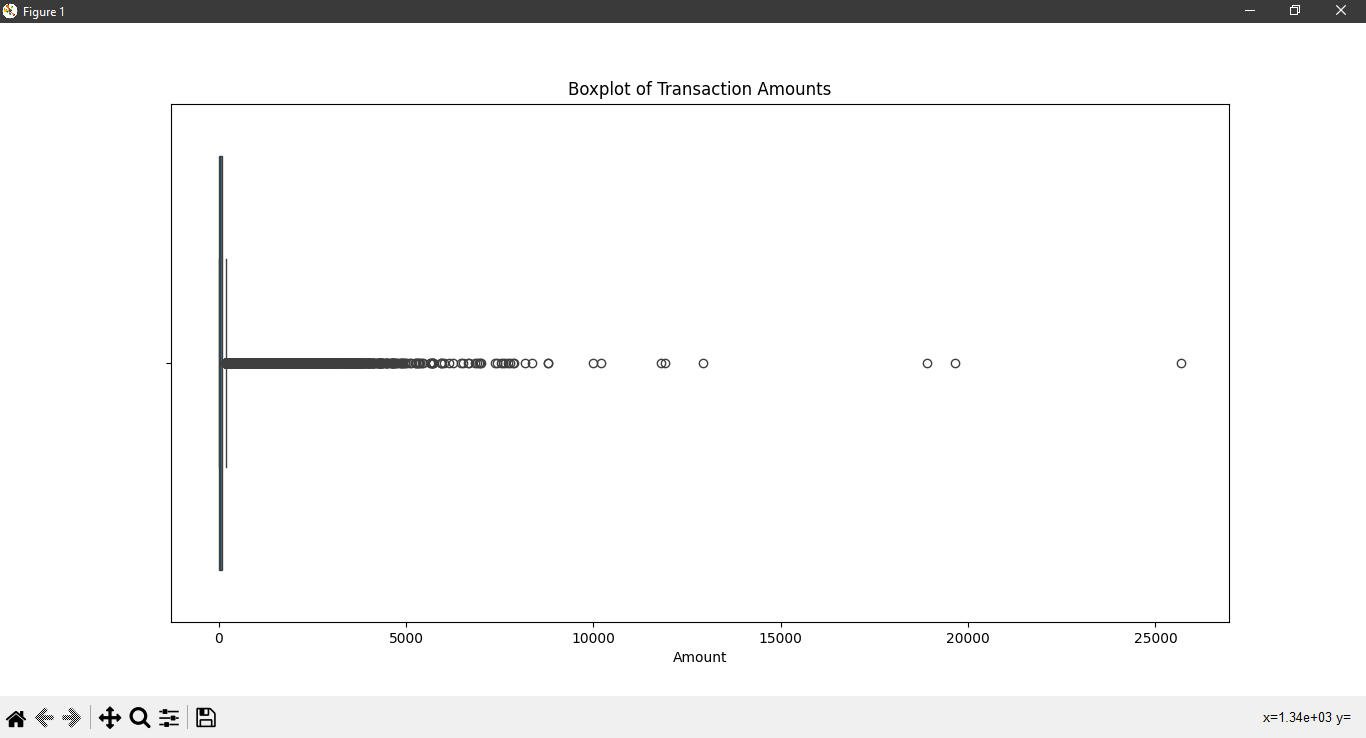
# *Analyze 'Amount' for outliers using a boxplot*

plt.figure(figsize=(10, 5))

sns.boxplot(x=df["Amount"])

plt.title("Boxplot of Transaction Amounts")

plt.show()



Outliers are important because they depict fraudulent transaction.

6. EDA (Exploratory Data Analysis): - => Feature Engineering

# *Standardize Features*

# *Standardize 'Time' and 'Amount'*

scaler = StandardScaler()

df["scaled\_time"] = scaler.fit\_transform(df[["Time"]])

df["scaled\_amount"] = scaler.fit\_transform(df[["Amount"]])

# *Drop original 'Time' and 'Amount' columns*

df = df.drop(["Time", "Amount"], axis=1)

# *EDA*

# *fraud vs non-fraud transaction*

print("fraud vs non-fraud transaction")

# *Check distribution of target variable*

fraud\_counts = df["Class"].value\_counts()

print(fraud\_counts)

# *Plot fraud vs. non-fraud distribution*

plt.figure(figsize=(5, 5))

sns.barplot(x=fraud\_counts.index, y=fraud\_counts.values, palette="viridis")

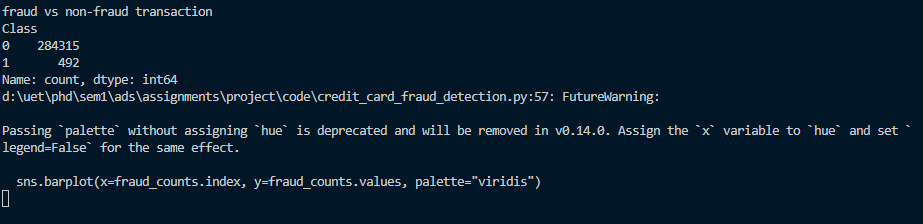
plt.title("Fraud vs Non-Fraud Transactions")

plt.xticks([0, 1], ["Non-Fraud", "Fraud"])

plt.ylabel("Count")

plt.show()

Output: -



A blue square with white text

Description automatically generated

# *Feature-Target Relationship*

print ("Feature-Target Relationship")

# *Plot correlation heatmap*

plt.figure(figsize=(15, 10))

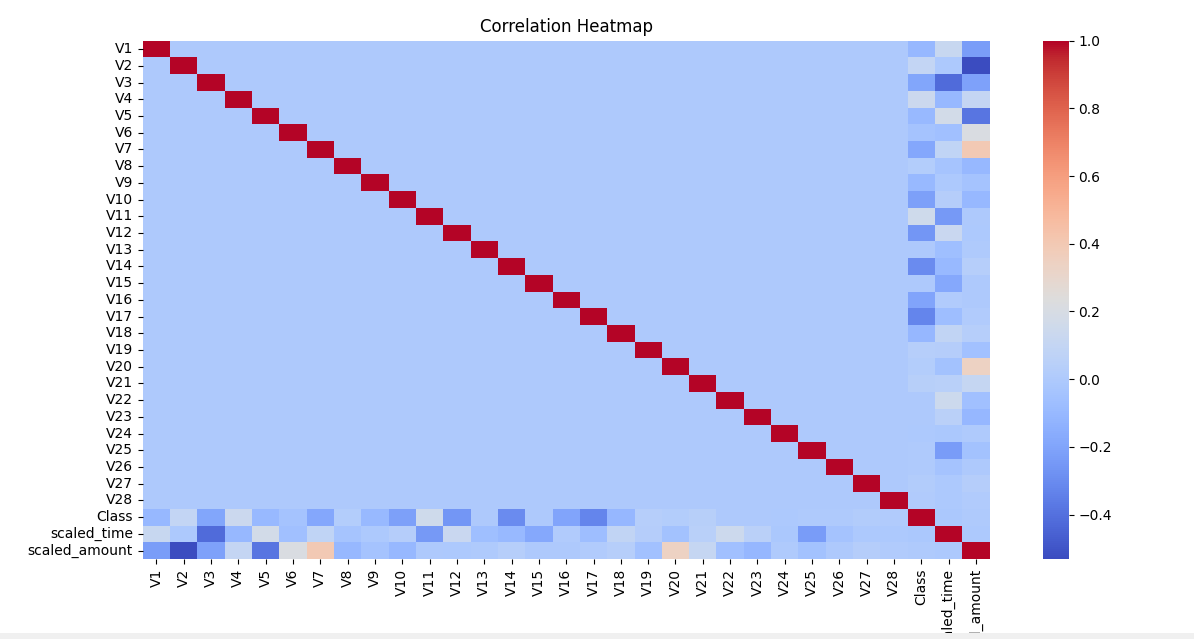
correlation = df.corr()

sns.heatmap(correlation, cmap="coolwarm", annot=False)

plt.title("Correlation Heatmap")

plt.show()

Output: -



# *Feature Analysis*

print ("Feature Analysis")

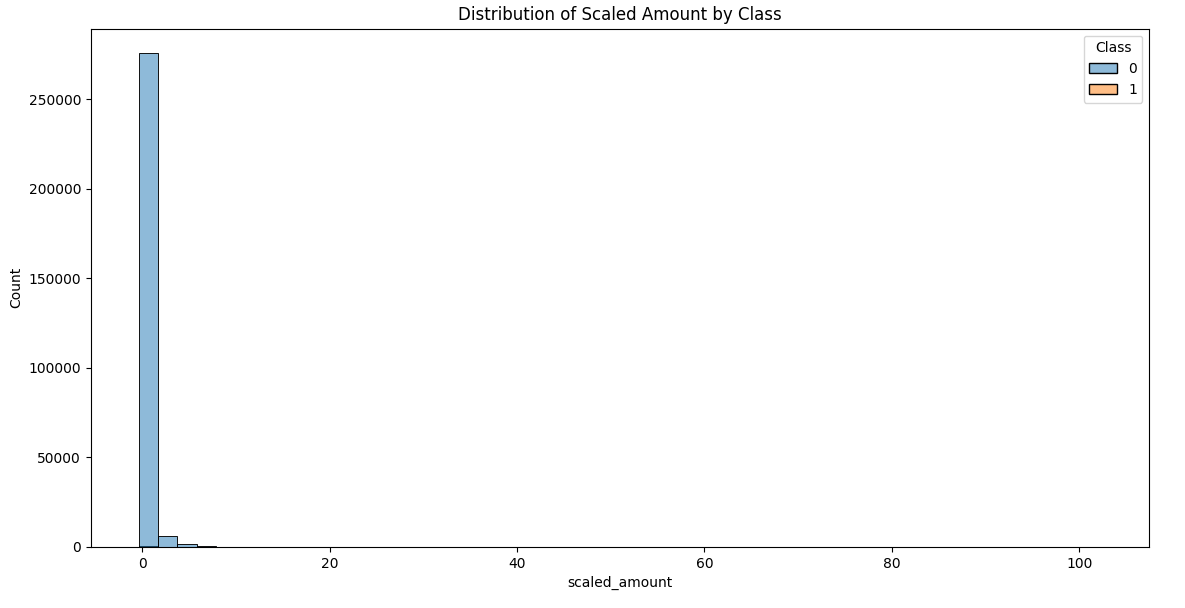
# *Analyze the relationship between scaled features and target*

sns.histplot(data=df, x="scaled\_amount", hue="Class", bins=50)

plt.title("Distribution of Scaled Amount by Class")

plt.show()

Output: -



sns.histplot(data=df, x="scaled\_time", hue="Class", kde=True, bins=50)

plt.title("Distribution of Scaled Time by Class")

plt.show()

Output: -

A graph of a graph

Description automatically generated with medium confidence

**7. Predictive Analysis: -, => split sy pehle smote krna hai, stratify bad mei krn hai.**

**SMOTE: -**

The code using **SMOTE (Synthetic Minority Oversampling Technique)** was applied to handle the **class imbalance** in the dataset.

**Understanding Class Imbalance**

* In the credit card fraud detection dataset, the target variable (Class) is highly imbalanced:
  + Non-fraudulent transactions (Class = 0) account for over 99.8%.
  + Fraudulent transactions (Class = 1) make up only 0.172%.
* This imbalance poses a challenge for machine learning models, which tend to be biased toward the majority class and perform poorly on the minority class (fraud).

**Why Handle Class Imbalance?**

Without addressing the class imbalance:

* **Models tend to prioritize accuracy**: A model could achieve over 99% accuracy by predicting only the majority class (non-fraudulent transactions) but fail to identify frauds.
* **Poor recall for the minority class**: The model would miss most fraudulent transactions, leading to high **false negatives**, which are costly in fraud detection.

**Why SMOTE?**

* **SMOTE (Synthetic Minority Oversampling Technique)** generates synthetic samples for the minority class (fraudulent transactions) by interpolating between existing samples.
* Unlike simple oversampling, which duplicates minority class samples, SMOTE creates synthetic data points, reducing the risk of overfitting.

**4. How SMOTE Helps**

* After applying SMOTE, the training set has a balanced number of samples for both classes, allowing the model to:
  + Learn patterns associated with fraud more effectively.
  + Improve recall and precision for the minority class.

**When to Use SMOTE**

* SMOTE is particularly useful when:
  + The dataset is heavily imbalanced.
  + The minority class is important, such as in fraud detection or medical diagnoses.
* SMOTE should be applied **only to the training set** to prevent data leakage into the test set.

**Model: Random Forest Classifier: -**

The Random Forest algorithm is an ensemble learning method primarily used for classification and regression tasks. It combines multiple decision trees (weak learners) to create a stronger predictive model.

* **Decision Tree Building**: Each subset is used to train a separate decision tree.
* **Feature Randomization**: At each split, only a random subset of features is considered to reduce correlation between trees.
* **Random Forest** is chosen here because it can handle imbalanced datasets with class\_weight.

print ("predictive analysis")

# *Separate features and target*

X = df.drop("Class", axis=1)

y = df["Class"]

# *Split into training and test sets*

*# Ensures that the training and test sets have the same distribution of fraudulent and non-fraudulent transactions (stratify=y).*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.3, random\_state=42, stratify=y

)

# *Apply SMOTE to handle class imbalance*

smote = SMOTE(random\_state=42)

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

# *Train a Random Forest model*

rf\_model = RandomForestClassifier(

    random\_state=42, n\_estimators=100, class\_weight="balanced"

)

rf\_model.fit(X\_train\_resampled, y\_train\_resampled)

# *Predict on the* test set

# This method applies the trained Random Forest model to the input data (X\_test) and predicts the output labels (y\_pred).

y\_pred = rf\_model.predict(X\_test)

# *Evaluate model performance*

print(classification\_report(y\_test, y\_pred))

# *Precision-Recall Curve*

# *Compute precision-recall values*

y\_pred\_proba = rf\_model.predict\_proba(X\_test)[:, 1]

precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_pred\_proba)

**Output: -**

A screenshot of a computer screen

Description automatically generated

**Precision-Recall Curve: -**

# *Precision-Recall Curve*

# *Compute precision-recall values*

y\_pred\_proba = rf\_model.predict\_proba(X\_test)[:, 1]

precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_pred\_proba)

# *Plot Precision-Recall Curve*

plt.figure(figsize=(10, 5))

plt.plot(recall, precision, marker=".", label="Random Forest")

plt.xlabel("Recall")

plt.ylabel("Precision")

plt.title("Precision-Recall Curve")

plt.legend()

plt.show()

# *Compute AUC-PR*

auc\_pr = roc\_auc\_score(y\_test, y\_pred\_proba)

print(f"AUC-PR: {auc\_pr}")

**Output: -**



