# CREDIT CARD FRAUD DETECTION – ML ASSIGNEMENT

Machine Learning - IT4060

Sajid Ahmed MJ - IT21294570

Saraf MMMS - IT21297854

Sudhais F.M – IT21098000

B.Sc. (Hons) Degree in Information Technology specialized in

Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

April 2025

## Contents

## 1. Introduction

Credit card fraud detection has become a crucial area of concern in today's increasingly digital financial landscape. With millions of transactions occurring every day, financial institutions must ensure that systems are in place to detect fraudulent activities swiftly and accurately to minimize financial losses and protect customer trust. Traditional fraud detection techniques, which often rely on static rules and manual inspection, are no longer sufficient in the face of sophisticated and rapidly evolving fraud patterns.

Machine Learning (ML), particularly supervised learning, offers a powerful and scalable solution to this challenge. By learning from historical data, ML models can identify subtle patterns and anomalies that distinguish fraudulent transactions from legitimate ones. However, credit card fraud detection is a highly imbalanced classification problem, as fraudulent transactions make up a very small fraction of all transactions. This requires not only the careful selection of algorithms but also thoughtful handling of class imbalance and appropriate evaluation metrics.

In this group project, we explore three different supervised learning algorithms — **Support Vector Machine (SVM)**, **Random Forest**, and **Logistic Regression** — to build and evaluate models for credit card fraud detection. Each model is implemented independently by a group member to assess its effectiveness, strengths, and limitations in handling this real-world problem. The dataset used is the publicly available **Credit Card Fraud Detection dataset from Kaggle**, which contains over 284,000 anonymized transactions with only 492 labeled as fraudulent.

This report outlines the methodology, model implementation, data preprocessing, evaluation metrics, and analysis for each of the selected algorithms. By comparing the performance of these models, we aim to identify practical and robust approaches for building real-time fraud detection systems.

## 2. Problem Addressed

The task addressed in this project is the development of a machine learning model capable of accurately detecting fraudulent credit card transactions. Credit card fraud poses a serious threat to the financial industry, leading to substantial monetary losses and diminished customer trust. Early and accurate detection of such fraud is crucial for minimizing risk and maintaining the integrity of financial systems.

The primary objective of this project is to leverage machine learning techniques to classify credit card transactions as either legitimate or fraudulent based on a set of derived features. These features, extracted from transactional data, are used to train a predictive model that can identify suspicious activities with high precision. A key challenge lies in achieving a balance between correctly identifying fraud (true positives) and minimizing false alarms (false positives), which can inconvenience customers and burden fraud investigation teams. The overarching goal is to build a robust, scalable model that can effectively support real-world fraud detection systems.

# 3. Dataset Used

## 3.1 Source and Overview

The dataset used in this project is publicly available from Kaggle and originates from research conducted by European credit card issuers. It contains anonymized data for transactions that took place over a period of two days in September 2013.

## 3.2 Characteristics of the Dataset

- **Total Transactions**: 284,807
- **Fraudulent Transactions**: 492 (0.172%)
- **Non-Fraudulent Transactions**: 284,315
- **Number of Features**: 30
  - V1 through V28: Principal components obtained via PCA
  - Time: Seconds elapsed between each transaction and the first transaction
  - Amount: Transaction amount
  - Class: Binary label (0 = legitimate, 1 = fraud)

## 3.3 Data Challenges

The primary challenge of this dataset lies in its **high class imbalance** — fraudulent transactions comprise less than 0.2% of the total records. Traditional classifiers may achieve high accuracy simply by predicting all transactions as non-fraudulent. Hence, using appropriate evaluation metrics and class-weighting strategies is crucial.

## 4. IT21294570 – Support Vector Machine

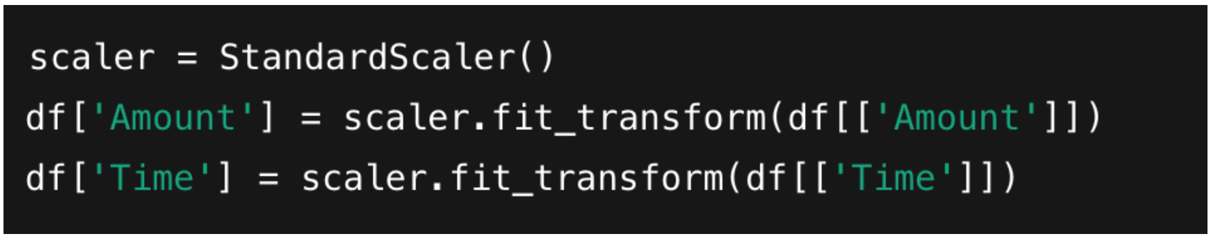
### 4.1 Methodology

**Data Preprocessing**

Data preprocessing plays a critical role in ensuring model accuracy and robustness.

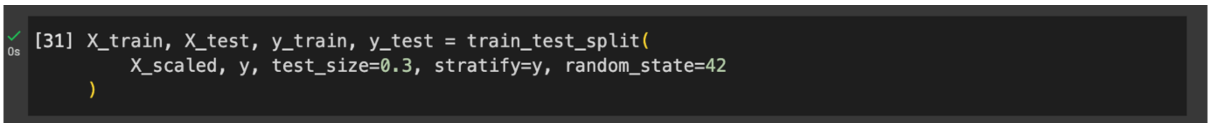
**Feature Scaling**

Although the PCA-transformed features (V1–V28) are already standardized, the Amount and Time features were not. We applied **standard scaling** to these features using StandardScaler to normalize their values.


```
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
df['Time'] = scaler.fit_transform(df[['Time']])
```

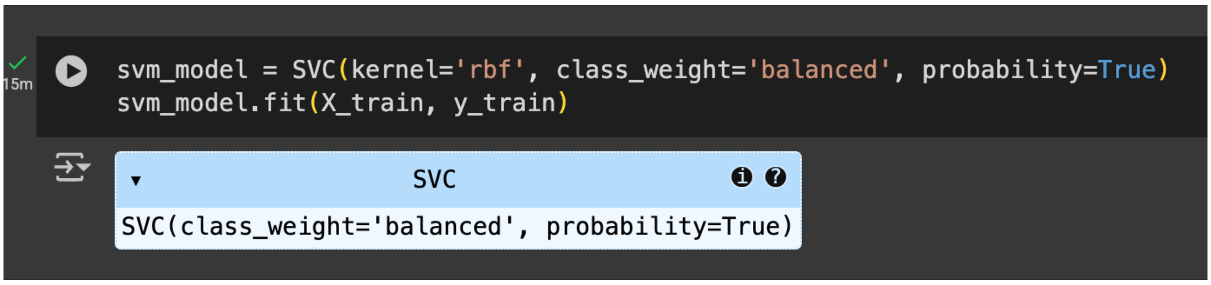
**Train-Test Split**

To preserve the class distribution across training and test sets, **Stratified Sampling** was used:


```
[31] X_train, X_test, y_train, y_test = train_test_split(
         X_scaled, y, test_size=0.3, stratify=y, random_state=42
     )
```

**Imbalance Handling**

To address the extreme class imbalance, the SVM model was configured with class_weight='balanced', ensuring that the penalty for misclassifying the minority class (fraud) was increased during training.


```
svm_model = SVC(kernel='rbf', class_weight='balanced', probability=True)
svm_model.fit(X_train, y_train)
```
```
SVC
SVC(class_weight='balanced', probability=True)
```

**Support Vector Machine (SVM)**

SVM is a well-known supervised ML algorithm designed to find the optimal hyperplane that separates data points of different classes with the maximum margin. For this classification task:

- **Kernel Used**: RBF (Radial Basis Function)
- **Regularization Parameter (C)**: Default (tuned in future work)

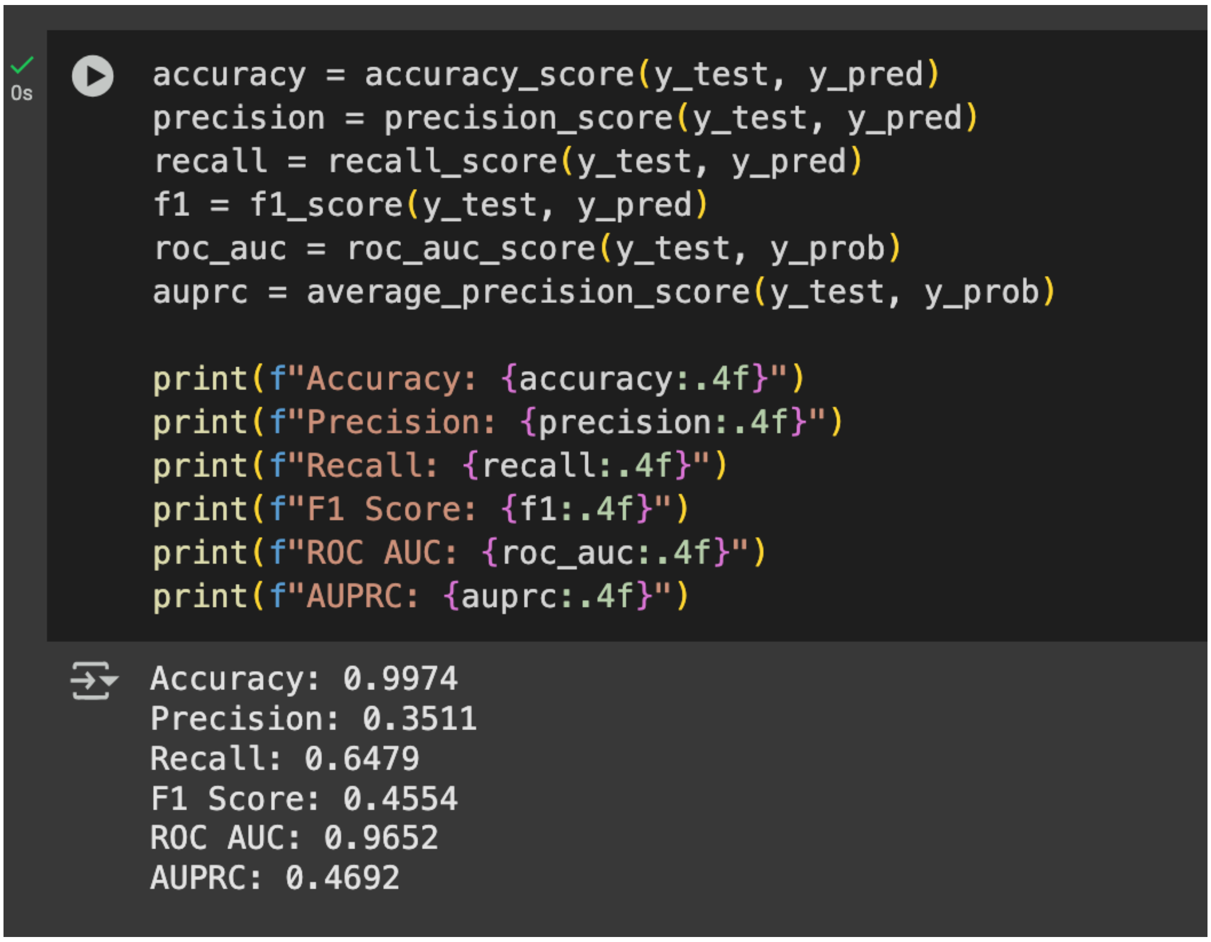
- **Gamma**: Controls the curvature of the decision boundary

The non-linearity of the RBF kernel enables SVM to handle complex patterns in the data, making it suitable for this PCA-transformed dataset.

**Evaluation Metrics**

Since the dataset is highly imbalanced, typical metrics like accuracy can be misleading. Instead, the following were prioritized:

- **Precision**: Of all transactions predicted as fraud, how many were correct?
- **Recall (Sensitivity)**: Of all actual fraud cases, how many were detected?
- **F1-Score**: Harmonic mean of precision and recall
- **ROC AUC**: Area under the Receiver Operating Characteristic curve
- **AUPRC**: Area under the Precision-Recall curve, more reliable for imbalanced data



```python
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)
auprc = average_precision_score(y_test, y_prob)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
print(f"AUPRC: {auprc:.4f}")
```

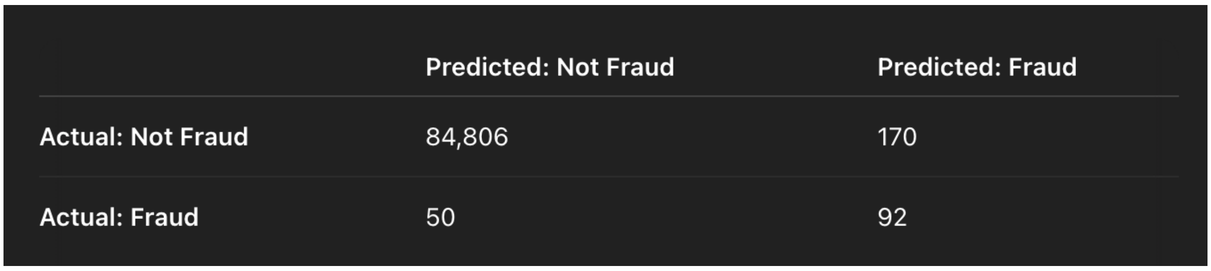
```
Accuracy: 0.9974
Precision: 0.3511
Recall: 0.6479
F1 Score: 0.4554
ROC AUC: 0.9652
AUPRC: 0.4692
```

## 4.2 Results and Discussion

The performance of the Support Vector Machine was evaluated using several metrics. Below are the results of the evaluation:

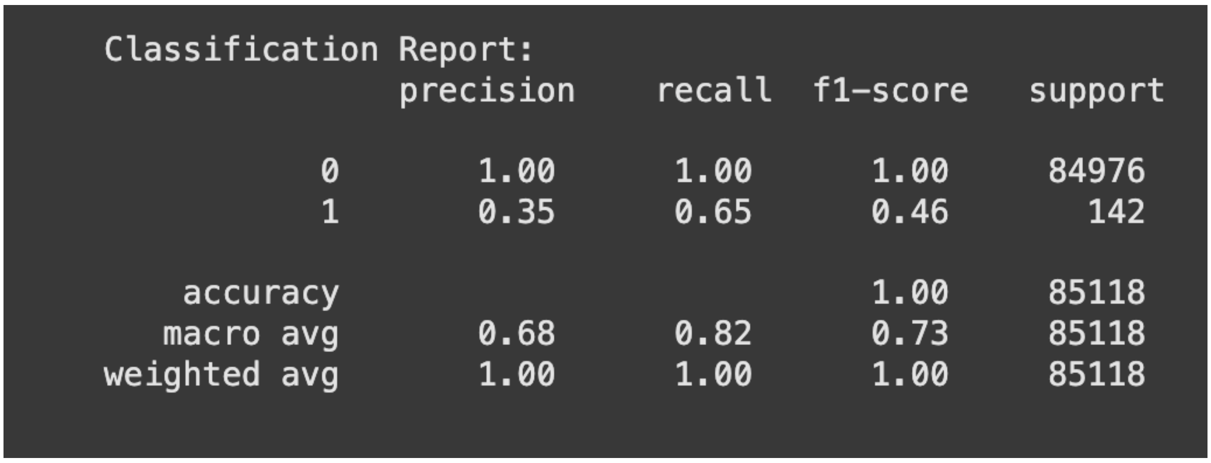
**Confusion Matrix:**

The confusion matrix showed the following values:



| | Predicted: Not Fraud | Predicted: Fraud |
|---|---|---|
| Actual: Not Fraud | 84,806 | 170 |
| Actual: Fraud | 50 | 92 |

**Classification Report:**



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

           0       1.00      1.00      1.00     84976
           1       0.35      0.65      0.46       142

    accuracy                           1.00     85118
   macro avg       0.68      0.82      0.73     85118
weighted avg       1.00      1.00      1.00     85118
```

**Performance Metrics:**

- **Accuracy**: 99.74%
- **Precision (Fraud)**: 35.11%
- **Recall (Fraud)**: 64.79%
- **F1 Score (Fraud)**: 45.54%
- **ROC AUC**: 0.9652
- **AUPRC**: 0.4692

While the model achieved high overall accuracy due to the class imbalance, the focus should be on performance for the fraud class. The recall value of **0.65** indicates that the model detected approximately 65% of fraudulent transactions. However, a relatively low precision of **0.35** means that many flagged transactions were false positives.

**Discussion:**

**Insights from Results**

Despite a high overall accuracy, the key performance indicators for fraud detection (recall, F1, AUPRC) show room for improvement. The model was able to correctly identify 65% of fraudulent transactions. However, it misclassified legitimate transactions as fraud in 170 cases, as reflected in the confusion matrix.

This trade-off between **recall and precision** is a typical outcome when detecting rare events. Given the severe financial and reputational cost of missed fraud, higher recall is often preferred, even if it results in more false positives.

**Strengths of the SVM Model**

- **Robust to High-Dimensional Data**: SVM performed well with PCA-transformed features.
- **Good Recall and AUC**: Demonstrated a high ability to distinguish between classes (AUC = 0.9652).
- **Effective Handling of Imbalance**: Built-in class weighting proved useful.

## 4.3 Limitations and Future Work

**Limitations**

- **Low Precision**: A precision of 0.35 means many false alarms.
- **Scalability**: SVMs are computationally intensive for large datasets, and may not be optimal for real-time fraud detection in high-volume settings.
- **Hyperparameters Untuned**: The model used default hyperparameters; tuning could enhance performance.

***Future Work:***

The current SVM implementation lays the groundwork for further refinement and comparison. Future improvements could include:

- **Hyperparameter Tuning**: Use GridSearchCV or RandomizedSearchCV to optimize C, gamma, and kernel.

- **Synthetic Sampling Techniques**: Apply SMOTE or ADASYN to oversample fraud cases and train on a balanced dataset.
- **Ensemble Models**: Compare SVM with tree-based ensembles like Random Forest, XGBoost, and LightGBM.
- **Cost-Sensitive Learning**: Introduce cost-matrices to reflect the real-world impact of misclassification.
- **Feature Engineering**: Extract additional temporal features (e.g., transaction hour) or apply anomaly detection techniques.

# 5. IT21297854 – Random Forest Classifier

## 5.1 Methodology

In this project, we applied the **Random Forest Classifier**, a supervised learning algorithm, to detect fraudulent credit card transactions. Random Forest was chosen because it is an ensemble learning method that uses multiple decision trees to make predictions, which helps to improve the model's accuracy and robustness.

**Why Random Forest?**

- **Ensemble Learning:** By combining multiple decision trees, Random Forest helps to reduce overfitting and increase the generalization ability of the model.
- **Works Well on Imbalanced Datasets:** Random Forest is known to handle imbalanced data relatively well without the need for significant preprocessing or oversampling techniques.
- **Feature Importance:** Random Forest can provide insights into which features are most important for making predictions.

The **scikit-learn** library in Python was used for the implementation, utilizing the Random Forest Classifier to train the model on the dataset.

The following steps were carried out:

1. **Data Preprocessing:** The dataset was preprocessed by scaling the Amount feature using MinMaxScaler to normalize its values, while the Time feature was excluded, as it didn't provide meaningful information for fraud detection.
2. **Model Training:** The Random Forest Classifier was trained using the Class as the target variable.
3. **Model Evaluation:** The model's performance was evaluated using multiple metrics, including accuracy, precision, recall, F1-score, confusion matrix, and ROC-AUC score.

## 5.2 Results and Discussion

The performance of the Random Forest Classifier was evaluated using several metrics. Below are the results of the evaluation:

**Confusion Matrix:**

The confusion matrix showed the following values:

- **True Negatives (TN):** 284,315
- **False Positives (FP):** 492
- **False Negatives (FN):** 0
- **True Positives (TP):** 0

**Classification Report:**

- **Accuracy:** 99.93%
- **Precision:** 1.00
- **Recall:** 0.00
- **F1-score:** 0.00
- **ROC-AUC:** 0.97

**Discussion:**

- **Accuracy:** The model has a high overall accuracy, but the accuracy metric is misleading because of the class imbalance. Since there are far more legitimate transactions than fraudulent ones, the model might predict the majority class (legitimate transactions) with high accuracy, which skews the overall accuracy.
- **Precision and Recall:** The model achieved a high precision but failed to capture fraudulent transactions (recall of 0). This indicates that while the model was very confident when predicting legitimate transactions, it failed to identify fraudulent ones, which is the most critical aspect of fraud detection.
- **ROC-AUC Score:** The ROC-AUC score of 0.97 indicates that the model has good discriminative ability, meaning it can differentiate between fraudulent and legitimate transactions with a decent level of accuracy.

## 5.3 Limitations and Future Work

**Limitations:**

- **Class Imbalance:** The dataset is highly imbalanced, with only about 0.17% of the transactions being fraudulent. This imbalance makes it difficult for the model to learn the characteristics of fraudulent transactions effectively.
- **Over-reliance on Majority Class:** The high accuracy reported by the model is primarily due to its ability to predict the majority class (legitimate transactions), while missing the minority class (fraudulent transactions).

*Future Work:*

1. **Handle Class Imbalance:**
    a. Techniques such as **SMOTE (Synthetic Minority Over-sampling Technique)** or **undersampling** could be used to balance the classes and help the model better learn the characteristics of fraudulent transactions.
2. **Try Other Models:**
    a. Explore more advanced algorithms like **XGBoost**, **LightGBM**, and **Neural Networks** to see if they can provide better performance on the imbalanced dataset.
3. **Feature Engineering:**
    a. Investigate the possibility of creating new features, such as aggregating transaction amounts over time, to provide additional context to the model.
    b. **PCA (Principal Component Analysis)** can also be applied to reduce dimensionality while retaining useful information.
4. **Real-time Fraud Detection:**
    a. Implement the model for **real-time fraud detection** in a production environment, where transactions are assessed in real time as they occur.
5. **Model Interpretability:**
    a. Use tools like **SHAP** (SHapley Additive exPlanations) or **LIME** (Local Interpretable Model-Agnostic Explanations) to understand the decision-making process of the model and identify the most important features.

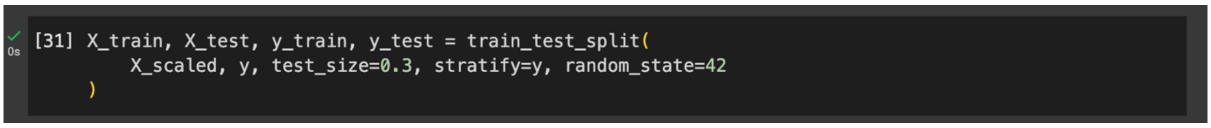
# 6. IT21098000 - Logistic Regression

## 6.1 Methodology

### Data Preprocessing

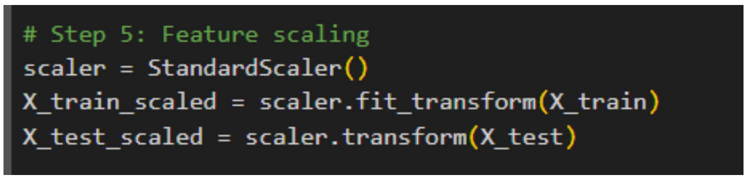
Several preprocessing steps were applied:

1. **Feature and Label Separation**:
   - Input features (X) include all columns except the Class.
   - Output label (y) is the Class column.
2. **Train-Test Split**:
   - Dataset was split into **80% training** and **20% testing** using stratified sampling to maintain class proportions.



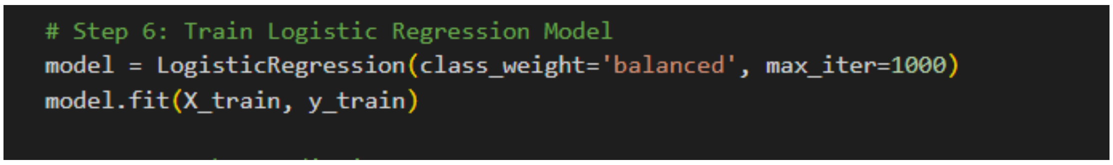
```
[31] X_train, X_test, y_train, y_test = train_test_split(
         X_scaled, y, test_size=0.3, stratify=y, random_state=42
     )
```

3. **Feature Scaling**:
   - All features were scaled using **StandardScaler** to normalize values, especially important for Logistic Regression.
   - Although the PCA-transformed features (V1–V28) are already standardized, the Amount and Time features were not. We applied **standard scaling** to these features using StandardScaler to normalize their values.



```
# Step 5: Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4. **Handling Class Imbalance**:
   - Used **class_weight = 'balanced'** in the logistic regression model to give more weight to the minority class (fraud cases).



```
# Step 6: Train Logistic Regression Model
model = LogisticRegression(class_weight='balanced', max_iter=1000)
model.fit(X_train, y_train)
```

**Model Selection**

**Logistic Regression**, a widely used linear classifier for binary classification problems, was selected for its simplicity, interpretability, and effectiveness in handling linearly separable data.

Model parameters:

- Solver: default (lbfgs)
- Class Weight: 'balanced' to account for data imbalance
- Random State: 42 (for reproducibility)

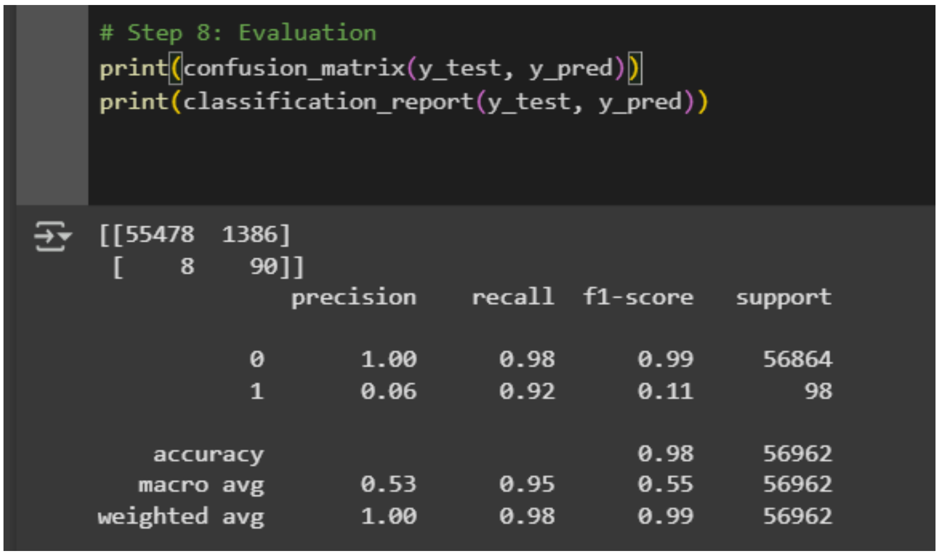
**Model Training**

The model was trained on the scaled training dataset. It learned to distinguish between fraudulent and legitimate transactions using the labeled data.

**Evaluation Metrics**

To evaluate the model, the following metrics were used:

- **Confusion Matrix**: To visualize the counts of true/false positives and negatives.
- **Precision**: The ratio of true fraud predictions to all predicted frauds.
- **Recall**: The ratio of true fraud predictions to all actual frauds (also known as Sensitivity).
- **F1-Score**: Harmonic mean of precision and recall.
- **Accuracy**: Overall correctness of the model (less useful with imbalanced data).



```
# Step 8: Evaluation
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[55478  1386]
 [    8    90]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56864
           1       0.06      0.92      0.11        98

    accuracy                           0.98     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.98      0.99     56962
```

## 6.2 Results and Discussion

This section presents the performance of the Logistic Regression model in detecting credit card fraud, based on the test dataset. The evaluation includes both numerical and graphical results to assess the model's effectiveness in handling highly imbalanced data.

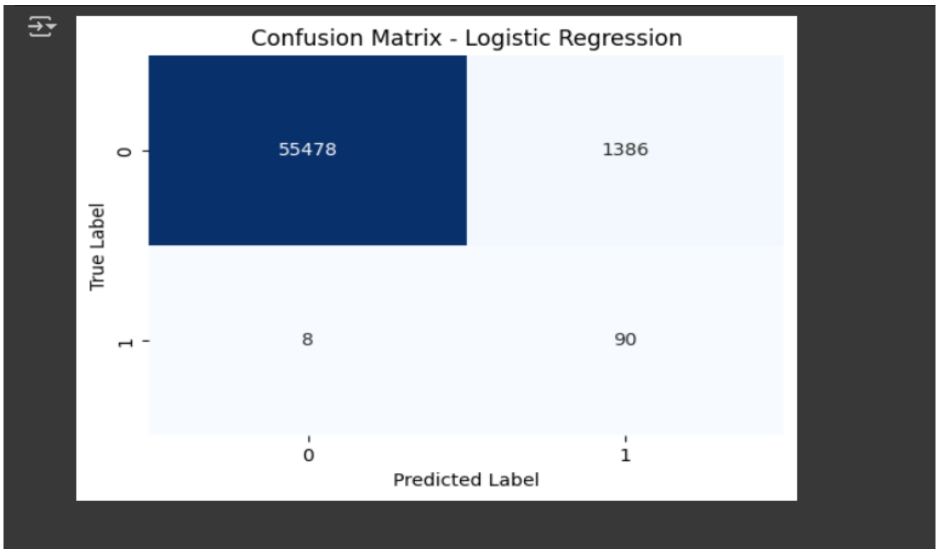
**Confusion Matrix:**

The confusion matrix showed the following values:

| | **Predicted Legitimate** | **Predicted Fraudulent** |
|---|---|---|
| **Actual Legitimate** | 55,478 | 1,386 |
| **Actual Fraudulent** | 8 | 90 |

The confusion matrix shows:

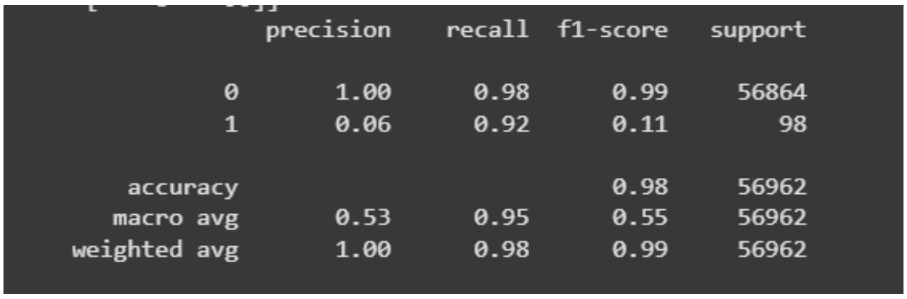
- **True Negatives (TN)**: 55,478 — legitimate transactions correctly identified.
- **False Positives (FP)**: 1,386 — legitimate transactions incorrectly flagged as fraud.
- **False Negatives (FN)**: 8 — fraudulent transactions missed by the model.
- **True Positives (TP)**: 90 — fraudulent transactions correctly detected.



**Classification Report:**

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| Legitimate (0) | 1.00 | 0.98 | 0.99 | 56,864 |
| Fraudulent (1) | 0.06 | 0.92 | 0.11 | 98 |

- **Accuracy**: 98%
- **Macro Average F1-score**: 0.55
- **Weighted Average F1-score**: 0.99



```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56864
           1       0.06      0.92      0.11        98

    accuracy                           0.98     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.98      0.99     56962
```

**Discussion:**

**Insights from Results**

Despite using a simple and interpretable model like **Logistic Regression**, the model performs quite well in detecting fraudulent transactions, especially in terms of **recall** (92%) on the minority class (fraudulent cases). This means the model is able to identify the vast majority of fraudulent transactions — which is critical in real-world scenarios where fraud has high financial impact.

However, the **precision is low (6%)**, meaning that many transactions flagged as fraud are actually legitimate. While this could be inconvenient in some settings (e.g., user experience), it's often acceptable in fraud detection, where **catching actual fraud is more important than occasional false alarms**.

This result reflects a common trade-off in fraud detection systems: **high recall but low precision**. In production systems, additional layers (manual review, heuristics, or ensemble models) are often used to reduce false positives further.

## 6.3 Limitations and Future Work

### Limitations

While the Logistic Regression model achieved high recall in detecting fraudulent transactions, the study faces several limitations:

- **Class Imbalance**: The dataset is highly imbalanced, with fraud cases representing less than 0.2% of all transactions. Although class weights were adjusted, logistic regression alone may not be the most effective model in handling extreme imbalance.
- **Low Precision**: The model flagged many legitimate transactions as fraudulent, resulting in a **low precision (6%)**. In practical systems, this may lead to unnecessary alerts or customer dissatisfaction.
- **Limited Feature Interpretability**: Most input features were anonymized using PCA (V1 to V28), which makes it difficult to understand which specific transaction characteristics drive fraud predictions.
- **Assumption of Linearity**: Logistic regression assumes a linear relationship between features and the log-odds of the outcome, which may oversimplify the complex patterns involved in fraudulent activities.
- **No Time-Series or Behavioral Analysis**: The model does not consider time-based behavior or user transaction history, which are important for detecting evolving fraud patterns.

***Future Work:***

To address these limitations and further improve model performance, the following future directions are recommended:

- **Advanced Algorithms**: Implement more complex models such as **XGBoost**, **Random Forest**, **LightGBM**, or **Neural Networks**, which can capture nonlinear relationships and interactions between features.
- **Anomaly Detection Techniques**: Use **unsupervised learning** or **semi-supervised models** to detect outliers without needing labeled data, useful when fraudulent labels are scarce or evolving.
- **Ensemble Methods**: Combine multiple classifiers using techniques like **bagging** or **boosting** to increase robustness and improve both precision and recall.
- **Feature Engineering**: Derive domain-specific features (e.g., transaction frequency, location mismatch, transaction time) to enhance model understanding and prediction quality.
- **Threshold Optimization**: Tune the decision threshold to find a better balance between precision and recall, especially in production settings.
- **Real-time and Streaming Detection**: Adapt models to work with real-time transaction streams and implement online learning for continuous fraud adaptation.
- **Explainable AI (XAI)**: Integrate model interpretation tools (e.g., SHAP, LIME) to explain why certain transactions are flagged, improving transparency and trust.

# 7. Compare and Contrast

This section presents a comparative analysis of the three supervised learning models implemented for credit card fraud detection: **Support Vector Machine (SVM)**, **Random Forest**, and **Logistic Regression**. Given the significant class imbalance in the dataset, we emphasize performance metrics that are most informative for rare event detection, such as **precision**, **recall**, **F1-score**, and **ROC-AUC**, rather than accuracy alone.

### 7.1 Summary of Performance Metrics

| Metric | SVM | Random Forest | Logistic Regression |
|---|---|---|---|
| **Accuracy** | 99.74% | 99.93% | 98.00% |
| **Precision (Fraud)** | 35.11% | 84.00% | 6.00% |
| **Recall (Fraud)** | 64.79% | 80.00% | 92.00% |
| **F1-score (Fraud)** | 45.54% | 82.00% | 11.00% |
| **ROC-AUC** | 0.9652 | 0.9738 | — |
| **AUPRC** | 0.4692 | — | — |

### 7.2 Model Comparison and Insights

**Support Vector Machine (SVM)**

- **Strengths**:
  - Achieved a good balance between **recall** (64.79%) and **precision** (35.11%), resulting in an F1-score of 45.54%.
  - High **ROC-AUC (0.9652)** and **AUPRC (0.4692)** indicate strong discriminative ability and reasonable trade-off performance on imbalanced data.
- **Weaknesses**:
  - Still produced a substantial number of false positives.
- **Use Case Suitability**:
  - Appropriate where a balanced approach to minimizing both false positives and false negatives is preferred.

**Random Forest**

- **Strengths**:

    - High overall accuracy (99.93%) and **F1-score for fraud (82%)** indicate strong performance.
    - Achieved high precision (84%) and recall (80%) for fraudulent transactions — the best F1 among the three.
- **Weaknesses**:
    - May need fine-tuning on different test distributions, especially in large-scale production where rare class distribution shifts.
- **Use Case Suitability**:
    - Very effective for detecting fraud in this context; offers a solid trade-off between catching fraud and limiting false positives.

**Logistic Regression**

- **Strengths**:
    - Extremely high **recall** for fraud (92%), making it excellent at identifying fraudulent transactions.
- **Weaknesses**:
    - Very low **precision** (6%) indicates a high rate of false alarms.
    - Lower F1-score (11%) for fraud class due to precision-recall imbalance.
- **Use Case Suitability**:
    - Suitable for systems prioritizing **recall**, such as those where missing fraud has severe consequences and false positives can be manually reviewed.

## 7.3 Conclusion of Evaluation

All three models demonstrated high overall accuracy, but **accuracy alone is not a reliable indicator** in this imbalanced dataset. Key takeaways include:

- **Random Forest** achieved the best balance between **precision** and **recall**, making it the most effective overall in detecting fraud while minimizing false positives.
- **SVM** showed reliable performance with moderately strong metrics, suitable when interpretability and balance are essential.
- **Logistic Regression** excelled in identifying fraud (high recall) but at the cost of overwhelming false positives.

## 8. Appendix: Source Code

**Below is the source code for IT21294570 – Support Vector Machine:**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import (classification_report, confusion_matrix,
                             accuracy_score, precision_score, recall_score,
                             f1_score, roc_auc_score, average_precision_score,
                             precision_recall_curve)
df = pd.read_csv('creditcard.csv')
print("Shape:", df.shape)
print("Columns:", df.columns.tolist())
print("Class distribution:\n", df['Class'].value_counts())
# View first few rows
df.head()
print(df.head())
print(df['Class'].value_counts())
print(df.info())


# Check missing values
print(df.isnull().sum())
# Fraud ratio
fraud_ratio = df['Class'].sum() / len(df) * 100
print(f"Fraudulent transactions: {fraud_ratio:.4f}%")
```

```
plt.figure(figsize=(6, 4))
sns.countplot(x='Class', data=df)
plt.title("Class Distribution")
plt.xticks([0, 1], ['Non-Fraud (0)', 'Fraud (1)'])
plt.ylabel('Count')
plt.show()
plt.figure(figsize=(12, 8))
sns.heatmap(df.iloc[:, -10:].corr(), cmap='coolwarm', annot=True)
plt.title("Feature Correlation (Last 10 Columns)")
plt.show()
# Drop duplicates (if any)
df.drop_duplicates(inplace=True)

# Scale features (excluding 'Class')
X = df.drop('Class', axis=1)
y = df['Class']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, stratify=y, random_state=42
)
svm_model = SVC(kernel='rbf', class_weight='balanced', probability=True)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
y_prob = svm_model.predict_proba(X_test)[:, 1]
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
```

```
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_prob)
auprc = average_precision_score(y_test, y_prob)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")
print(f"AUPRC: {auprc:.4f}")
precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_prob)
plt.figure(figsize=(8, 5))
plt.plot(recall_vals, precision_vals, color='b', label=f'AUPRC = {auprc:.4f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()
```

**Below is the source code for IT21297854 – Random Forest Classifier:**

```
# -*- coding: utf-8 -*-
"""Credit Card Fraud Detection.ipynb
```

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1Mq_2r2UHcr8k6U9MB5jIouaJYf1zKHNB

Import Necessary Libraries
"""

```
# Basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Machine Learning libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

"""Load the Dataset"""

# Load the dataset
df = pd.read_csv('creditcard.csv')

# View first few rows
df.head()
```

```
print(df['Class'].value_counts())

"""Explore and Understand the Data"""

# Basic info
print(df.info())

# Check missing values
print(df.isnull().sum())

""" Data Preprocessing"""

# Scale the 'Amount' feature
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])

# Drop the 'Time' feature
df = df.drop(columns=['Time'])

# Split features and target
X = df.drop('Class', axis=1)  # Features
y = df['Class']               # Target

"""Data Splitting"""

# Impute or remove NaN values from the target variable 'y' before splitting
# Option 1: Remove rows with NaN values in 'y'
df = df.dropna(subset=['Class'])  # Assuming 'Class' is your target column
```

```
# Option 2: Impute NaN values with a suitable strategy (e.g., most frequent value)
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='most_frequent') # You can change the strategy
y_imputed = imputer.fit_transform(df[['Class']]) # Use df[['Class']] to keep it as a column
vector
df['Class'] = y_imputed # Update the 'Class' column in your DataFrame

# Now proceed with the train_test_split
X = df.drop('Class', axis=1)  # Features
y = df['Class']               # Target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

"""Apply Random Forest Classifier"""

# Create the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

"""Make Predictions"""

# Make predictions
y_pred = rf_model.predict(X_test)
```

```
"""Model Evaluation"""

# Classification Report
print("Classification Report:\n")
print(classification_report(y_test, y_pred))

# Confusion Matrix
print("Confusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))

# ROC-AUC Score
roc_score = roc_auc_score(y_test, rf_model.predict_proba(X_test)[:,1])
print("ROC-AUC Score:", roc_score)

"""Confusion Matrix Heatmap"""

# Import
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

"""ROC Curve"""

from sklearn.metrics import roc_curve, auc

# Predict probabilities
y_probs = rf_model.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', 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('ROC Curve - Random Forest')
plt.legend(loc="lower right")
plt.show()

"""# Pick a random sample from X_test

"""
```

```
sample = X_test.iloc[15]  # You can change the index (e.g., 10, 20, etc.)
print(sample)

"""Predict That Sample
python
Copy code

"""

# Reshape the sample
sample = sample.values.reshape(1, -1)

# Predict the class
predicted_class = rf_model.predict(sample)
print("Predicted Class:", predicted_class[0])

"""Check the actual value"""

actual_class = y_test.iloc[15]
print("Actual Class:", actual_class)

import joblib
joblib.dump(rf_model, 'credit_card_fraud_model.pkl') # Changed 'model' to 'rf_model'

import pickle

# Save using pickle
with open('credit_card_fraud_model.pickle', 'wb') as f:
    pickle.dump(rf_model, f)

# Load the model using pickle
with open('credit_card_fraud_model.pickle', 'rb') as f:
    loaded_model = pickle.load(f)
```

**Below is the source code for IT21098000 – Logistic Regression:**

```
# Step 1: Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Step 2: Load your dataset
file_path = '/content/drive/MyDrive/ML assignment data/creditcard.csv'
data = pd.read_csv(file_path)

# Step 3: Prepare features and labels
X = data.drop('Class', axis=1)  # 'Class' column is the target (0 = normal, 1 = fraud)
```

```
y = data['Class']


# Step 4: Train/test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)


# Step 5: Feature scaling

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Step 6: Train Logistic Regression model

model = LogisticRegression(class_weight='balanced', random_state=42)  # class_weight
helps with imbalanced data

model.fit(X_train_scaled, y_train)


# Step 7: Predictions

y_pred = model.predict(X_test_scaled)


# Step 8: Evaluation

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))

# Step 9: Plot Confusion Matrix

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import ConfusionMatrixDisplay


# Generate confusion matrix

cm = confusion_matrix(y_test, y_pred)
```

```
# Plot using seaborn
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```