**Credit Card Fraud Detection Project**

**1. Project Overview**

The Credit Card Fraud Detection project aims to develop a robust machine learning model to identify and classify fraudulent transactions in credit card data. The project involves comprehensive data analysis, preprocessing, model selection, hyperparameter tuning, and evaluation.

**1.1 Key Objectives**

* Develop a machine learning model capable of accurately detecting fraudulent credit card transactions.
* Explore and analyse the dataset to gain insights into the distribution and patterns of legitimate and fraudulent transactions.
* Compare and evaluate the performance of logistic regression and random forest models for fraud detection.
* Ensure interpretability of the models through feature importance analysis using coefficients and SHAP values.

**2. Libraries and Dependencies**

**2.1 Purpose**

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

**2.2 Code overview**

import NumPy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import StandardScaler

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

from sklearn.ensemble import IsolationForest

import shap

import warnings

import pickle

warnings.filterwarnings("ignore", category=DeprecationWarning)

**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.

**3. Dataset Loading and Exploration**

**3.1 Purpose**

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

**3.2 Code overview**

credit\_card\_data = pd.read\_csv('creditcard.csv')

credit\_card\_data.head()

credit\_card\_data.isnull().sum()

**4. Data Cleaning**

**4.1 Purpose**

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

**4.2 Code overview**

duplicate\_rows = credit\_card\_data.duplicated().sum()

print('Number of duplicate records:', duplicate\_rows)

credit\_card\_data = credit\_card\_data.drop\_duplicates()

print(credit\_card\_data.isnull().sum())

**Explanation:**

credit\_card\_data.duplicated().sum(): Calculates the number of duplicate rows.

credit\_card\_data.drop\_duplicates(): Removes duplicate records.

print(credit\_card\_data.isnull().sum()): Verifies that there are no missing values after removing duplicates.

**4.3 Class Distribution Analysis**

**4.4 Code overview**

print(credit\_card\_data['Class'].value\_counts())

0 283253

1 473

Name: Class, dtype: int64

**5. Exploratory Data Analysis (EDA)**

**5.1 Purpose**

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

**5.2 Code overview**

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

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

sns.boxplot(x='Class', y='Amount', data=credit\_card\_data, showfliers=False)

plt.title('Box Plot of Transaction Amount by Class')

plt.show()

A diagram of a box

Description automatically generated

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

sns.boxplot(x='Class', y='Time', data=credit\_card\_data, showfliers=False)

plt.title('Box Plot of Transaction Time by Class')

plt.show()

A diagram of a diagram

Description automatically generated with medium confidence

**# Histograms for feature distributions (V1 to V28)**

plt.figure(figsize=(14, 12))

for i in range(1, 29):

plt.subplot(7, 4, i)

sns.histplot(credit\_card\_data[f'V{i}'], bins=30, kde=True)

plt.title(f'Distribution of V{i}')

plt.tight\_layout()

plt.show()

A collage of different graphs

Description automatically generated

**# Correlation Heatmap**

plt.figure(figsize=(16, 10))

correlation\_matrix = credit\_card\_data.corr()

sns.heatmap(correlation\_matrix, cmap='coolwarm', annot=True, fmt=".2f", linewidths=.5)

plt.title('Correlation Heatmap with Numerical Annotations')

plt.show()

A graph of numbers and symbols

Description automatically generated with medium confidence

**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.

**6. Data Preprocessing**

**6.1 Purpose**

This section addresses outliers, scales features, and performs under-sampling to handle class imbalances.

**6.2 Code overview**

**# Outlier handling using Isolation Forest for 'Amount' and 'Time'**

outlier\_detector = IsolationForest(contamination=0.01, random\_state=1)

credit\_card\_data['Outlier'] = outlier\_detector.fit\_predict(credit\_card\_data[['Amount', 'Time']])

credit\_card\_data = credit\_card\_data[credit\_card\_data['Outlier'] == 1].drop(columns='Outlier')

**# Scaling 'Amount' using StandardScaler**

scaler = StandardScaler()

credit\_card\_data['Amount'] = scaler.fit\_transform(credit\_card\_data[['Amount']])

**# Under-sampling**

Legit = credit\_card\_data[credit\_card\_data['Class'] == 0]

fraud = credit\_card\_data[credit\_card\_data['Class'] == 1]

Legit\_sample = Legit.sample(n=492)

new\_dataset = pd.concat([Legit\_sample, fraud], axis=0)

**Explanation:**

* Isolation Forest is used to handle outliers in 'Amount' and 'Time.'
* 'Amount' is scaled using StandardScaler.
* Under-sampling is performed to balance the class distribution.

**7. Model Selection: Logistic Regression**

**7.1 Purpose**

This section trains a Logistic Regression model, optimizes hyperparameters, and assesses its performance.

**7.2 Code overview**

# Model training

logistic\_model = LogisticRegression(random\_state=2, max\_iter=1000)

# Hyperparameter tuning using GridSearchCV

param\_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

grid\_search = GridSearchCV(LogisticRegression(random\_state=2, max\_iter=1000), param\_grid, cv=5)

grid\_search.fit(X\_train, Y\_train)

# Get the best hyperparameter

best\_C = grid\_search.best\_params\_['C']

# Model training with the best hyperparameter

logistic\_model = LogisticRegression(C=best\_C, random\_state=2, max\_iter=1000)

logistic\_model.fit(X\_train, Y\_train)

**Explanation:**

* A Logistic Regression model is initialized.
* GridSearchCV is employed for hyperparameter tuning.
* The best hyperparameter 'C' is identified, and the model is trained with the optimal parameter.

**7.3 Cross-Validation**

# Cross-validation

cross\_val\_scores = cross\_val\_score(logistic\_model, X\_train, Y\_train, cv=5, scoring='accuracy')

print('Cross-Validation Scores:', cross\_val\_scores)

**Explanation:**

Cross-validation assesses the model's generalization performance.

Cross-Validation Scores: [0.92810458 0.94117647 0.96732026 0.93464052 0.94078947]

**7.4 Model Evaluation: Logistic Regression**

# Accuracy score on training data

X\_train\_prediction\_logistic = logistic\_model.predict(X\_train)

training\_data\_accuracy\_logistic = accuracy\_score(X\_train\_prediction\_logistic, Y\_train)

print('Logistic Regression Training data accuracy:', training\_data\_accuracy\_logistic)

# Accuracy score on test data

X\_test\_prediction\_logistic = logistic\_model.predict(X\_test)

test\_data\_accuracy\_logistic = accuracy\_score(X\_test\_prediction\_logistic, Y\_test)

print('Logistic Regression Test data accuracy:', test\_data\_accuracy\_logistic)

**Explanation:**

The accuracy scores on training and test data are calculated for evaluation.

Logistic Regression Training data accuracy: 0.9489528795811518

Logistic Regression Test data accuracy: 0.9427083333333334

**7.5 Additional Metrics: Logistic Regression**

# AUC-ROC scores

logistic\_train\_auc = roc\_auc\_score(Y\_train, logistic\_model.predict\_proba(X\_train)[:, 1])

logistic\_test\_auc = roc\_auc\_score(Y\_test, logistic\_model.predict\_proba(X\_test)[:, 1])

print('Logistic Regression Train AUC:', logistic\_train\_auc)

print('Logistic Regression Test AUC:', logistic\_test\_auc)

**Explanation:**

AUC-ROC scores provide insights into the model's ability to discriminate between classes.

Logistic Regression Train AUC: 0.9799866943752872

Logistic Regression Test AUC: 0.9829477571413056

**8. Model Comparison: Random Forest**

**8.1 Purpose**

This section introduces a Random Forest model as an alternative, evaluating its performance and comparing it with Logistic Regression.

**8.2 Code overview**

# Model training: Random Forest

rf\_model = RandomForestClassifier(random\_state=2)

rf\_model.fit(X\_train, Y\_train)

Explanation:

A Random Forest model is trained.

**8.3 Additional Metrics: Random Forest**

# AUC-ROC scores

rf\_train\_auc = roc\_auc\_score(Y\_train, rf\_model.predict\_proba(X\_train)[:, 1])

rf\_test\_auc = roc\_auc\_score(Y\_test, rf\_model.predict\_proba(X\_test)[:, 1])

print('Random Forest Train AUC:', rf\_train\_auc)

print('Random Forest Test AUC:', rf\_test\_auc)

**Explanation:**

AUC-ROC scores provide insights into the Random Forest model's discriminative performance.

Random Forest Train AUC: 1.0

Random Forest Test AUC: 0.9841425002715326

**8.4 Model Evaluation: Random Forest**

**8.4.1 Purpose**

This section provides a detailed evaluation of the Logistic Regression and random forest’s model's performance on the test dataset. The classification report and confusion matrix offer insights into the model's ability to correctly classify instances and identify potential areas for improvement.

**8.4.2 Code Explanation**

**# Classification report and confusion matrix**

print('\nClassification Report (Logistic Regression):\n', classification\_report(Y\_test, logistic\_model.predict(X\_test)))

print('\nConfusion Matrix (Logistic Regression):\n', confusion\_matrix(Y\_test, logistic\_model.predict(X\_test)))

Classification Report (Logistic Regression):

precision recall f1-score support

0 0.93 0.96 0.95 99

1 0.96 0.92 0.94 93

accuracy 0.94 192

macro avg 0.94 0.94 0.94 192

weighted avg 0.94 0.94 0.94 192

Confusion Matrix (Logistic Regression):

[[95 4]

[ 7 86]]

# Classification report and confusion matrix

print('\nClassification Report (Random Forest):\n', classification\_report(Y\_test, rf\_model.predict(X\_test)))

print('\nConfusion Matrix (Random Forest):\n', confusion\_matrix(Y\_test, rf\_model.predict(X\_test)))

**Explanation:**

A detailed classification report and confusion matrix are provided for the Random Forest model.

Classification Report (Random Forest):

precision recall f1-score support

0 0.93 0.99 0.96 99

1 0.99 0.92 0.96 93

accuracy 0.96 192

macro avg 0.96 0.96 0.96 192

weighted avg 0.96 0.96 0.96 192

Confusion Matrix (Random Forest):

[[98 1]

[ 7 86]]

**9. Model Interpretability**

**9.1 Purpose**

This section explores the interpretability of the models, analysing Logistic Regression coefficients and SHAP values for Random Forest.

**9.2 Code overview**

# Logistic Regression coefficients

feature\_names = X.columns

coefficients = logistic\_model.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)

**Explanation:**

Logistic Regression coefficients are extracted and displayed to identify important features.

# Interpretation of Logistic Regression Coefficients

**9.2.1 Overview**

Logistic Regression coefficients reveal the impact of each feature on the model's predictions. Positive coefficients indicate a positive relationship with the target (fraudulent transactions), while negative coefficients suggest a negative relationship.

**9.2.2 Key Findings**

1. V4 (Coefficient: 0.677197): V4 has the highest positive coefficient, signifying a strong positive impact on the likelihood of predicting a fraudulent transaction.

2. V17 (Coefficient: 0.314554): V17 also has a positive coefficient, contributing significantly to the model's prediction of fraudulent transactions.

3. V5 (Coefficient: 0.278490): V5 exhibits a positive relationship with the target, influencing the model's predictions.

4. Amount (Coefficient: 0.201437): The transaction amount (Amount) has a positive impact on predicting fraudulent transactions.

5. V18 (Coefficient: 0.192923): V18 contributes positively to the model's prediction of fraud.

**9.2.3 Interpretation**

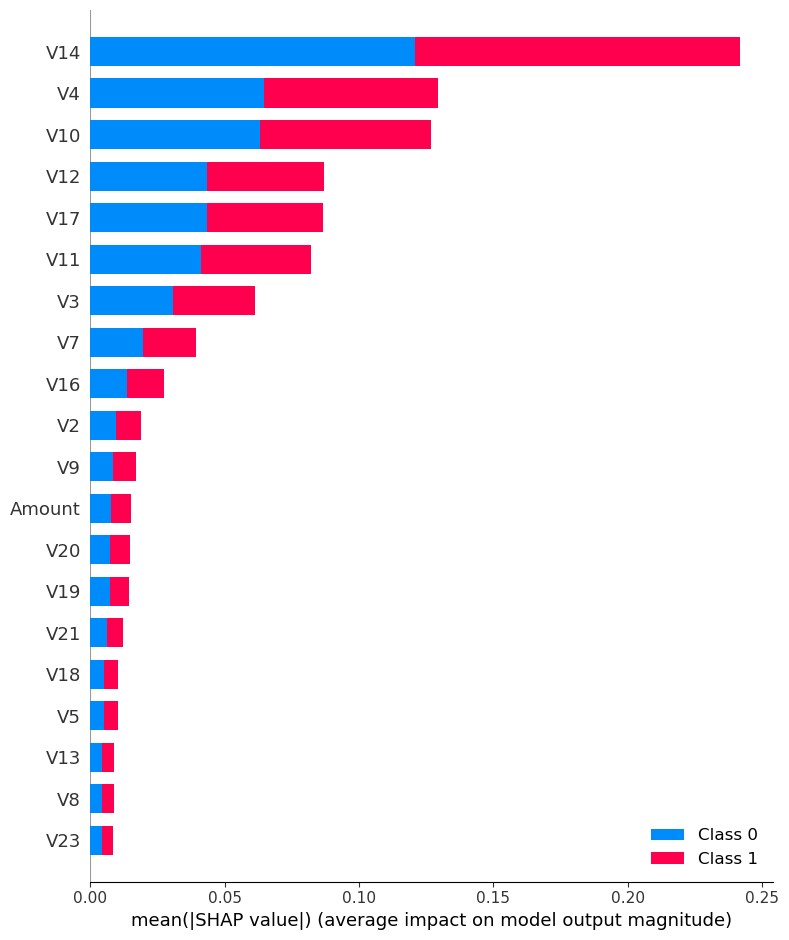
The positive coefficients indicate features that increase the likelihood of a transaction being classified as fraudulent. Conversely, features with negative coefficients contribute to the likelihood of a transaction being classified as legitimate.

**9.3 SHAP Values: Random Forest**

# SHAP values for Random Forest

shap\_values\_rf = shap.TreeExplainer(rf\_model).shap\_values(X\_train)

shap.summary\_plot(shap\_values\_rf, X\_train, plot\_type='bar')



**Explanation:**

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

**10. Model Serialization**

**10.1 Purpose**

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

**10.2 Code Explanation**

# Save Logistic Regression model to a file

with open('logistic\_model.pkl', 'wb') as file:

pickle.dump(logistic\_model, file)

# Save Random Forest model to a file

with open('random\_forest\_model.pkl', 'wb') as file:

pickle.dump(rf\_model, 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:

**1. 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.

**2. 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.

**3. Interpretability**

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

**4. 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.

**5. 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.

**6. 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.