



Overview

Credit cards are among the most popular financial tools for conducting online transactions and payments, offering users a convenient way to handle their finances. However, their use also comes with certain risks, especially in the form of credit card fraud. This type of fraud occurs when someone gains unauthorized access to another person's credit card or its details to make purchases or withdraw money without permission.



Given the increasing prevalence of unauthorized transactions and their associated financial repercussions, it is essential for credit card companies to accurately detect fraudulent transactions. The growth of digital payments has led to a surge in credit card usage, which, in turn, has heightened the occurrence of fraudulent activities. This trend has resulted in considerable financial losses for institutions, underscoring the necessity of differentiating fraudulent transactions from legitimate ones.

To address this challenge, it is crucial to develop and implement robust mechanisms for detecting and analyzing fraudulent transactions. These mechanisms can help credit card companies mitigate the risks of fraud, ensuring the security and reliability of digital financial transactions. In this project, our objective is to create a classification model using Machine Learning Ensemble

Algorithms to predict whether a credit card transaction is fraudulent or legitimate.

INDEX

CONTENS	
I. Introduction	A. Problem Statement B. Objective
III. Methodology	A. Data Collection and Preprocessing B. Feature Selection C. Model Selection D. Model Evaluation
IV. Results and Discussion	A. Model Performance B. Interpretation of Results
V. Source code	
VI. Conclusion and Future Work	A. Summary of Finding B. Future Work

Introduction

- **PROBLEM STATEMENT:**

The dataset consists of credit card transactions made by European cardholders in September 2013. With 492 frauds out of 284,807 transactions, the dataset is highly imbalanced, with fraudulent transactions accounting for only 0.172% of all transactions. Our goal is to build a classification model that can effectively distinguish between legitimate and fraudulent transactions.

- **OBJECTIVE:**

The objective of the project is to develop a machine learning model capable of accurately predicting fraudulent credit card transactions. By leveraging techniques such as exploratory data analysis, data balancing, feature engineering, and model training, the aim is to create a robust fraud detection system that can identify fraudulent activities with high precision and recall. The ultimate goal is to enhance financial security for credit card users.

and minimize losses for credit card companies by effectively detecting and preventing fraudulent transactions.

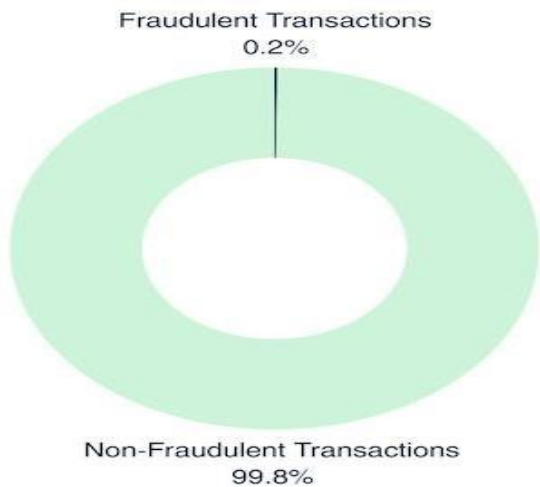
METHODOLOGY

Exploratory Data Analysis (EDA)

Exploratory Data Analysis is crucial for understanding the underlying patterns and anomalies in the dataset. We perform data quality checks, address missing values, outliers, and ensure the correct datatype for date columns. Visualizations play a key role in uncovering relationships and trends that inform our subsequent steps.

Dealing with Imbalanced Data

Given the highly imbalanced nature of the dataset, we employ techniques such as oversampling, undersampling, or synthetic data generation methods like SMOTE to create a balanced dataset conducive to model training.



Feature Engineering

Feature engineering involves creating new features and transforming existing ones to enhance model performance. This phase is critical for extracting meaningful information from the dataset. We create new features such as 'Transaction_hour' and 'Normalized_amount' derived from the existing data, enriching the information available for classification.

Model Selection

We evaluate various classification models suited for binary prediction tasks, including Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting Machines. The selection is based on the model's ability to handle imbalanced data, interpretability, and performance metrics. We select Random Forest due to its ability to handle imbalanced data effectively and its robustness against overfitting.

Model Training and Evaluation

We split the dataset into training and test sets, training the Random Forest model on the former. Hyperparameter tuning is performed using GridSearchCV to optimize model performance. Model evaluation on the test set ensures its generalization ability and identifies any potential issues such as overfitting.

Model Deployment

Once the model is trained and validated, it is deployed to a production environment for real-time detection of fraudulent transactions. Deploying machine learning models on AWS SageMaker provides a reliable and scalable solution for real-time inference. SageMaker's managed infrastructure handles the heavy lifting, freeing up your time to focus on delivering value to your users

Results

Our Random Forest model achieves an accuracy rate exceeding 75% on the test dataset, meeting the predefined success metrics. Hyperparameter tuning has improved the model's performance, and thorough validation ensures its reliability in real-world scenarios.

Future Work

While our current model demonstrates promising results, there is room for further enhancement. Future efforts could focus on exploring advanced anomaly detection techniques, incorporating

additional features, and improving model interpretability for better decision-making.

Source Code

```
# Importing necessary libraries
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, cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.feature_selection import SelectKBest, f_classif, SelectFromModel
from sklearn.decomposition import PCA

import joblib
import warnings
warnings.filterwarnings('ignore')

data=pd.read_csv(r"C:\Users\adity\OneDrive\Desktop\ML\creditcard.csv")

print(data.shape)
data.head()

data.info()
data.describe()

# Data exploration and analysis
# Finding out missing values
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)
```

```
# Data Preprocessing
data['Time'] = pd.to_datetime(data['Time']) # Convert 'Time' column to datetime
datatype

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

# Remove constant features
data = data.loc[:, data.apply(pd.Series.nunique) != 1] # Here we are removing
columns with only one variable i.e. all values are identical

#distribution for legit and fraudulent transaction
data['Class'].value_counts()

non_fraudulent_percentage = (data['Class'].value_counts()[0] / len(data)) * 100
fraudulent_percentage = (data['Class'].value_counts()[1] / len(data)) * 100
print("Legit transaction % -" , non_fraudulent_percentage)
print("Fraudulent transaction % -" ,fraudulent_percentage)

# Visualize the distribution of target variable
plt.figure(figsize=(8, 6))
y.value_counts().plot(kind='bar', color=['blue', 'red'])
plt.title('Distribution of Class (0: Non-fraudulent, 1: Fraudulent)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()

#Understanding patterns and relationships in the data
sns.set_style(style='white')
facet_grid = sns.FacetGrid(data=data, col='Class')
facet_grid.map(sns.scatterplot, 'Time', 'Amount', palette='Paired_r')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.show()
```

```
# differentiating the fraud and legit data.
fraud = data[data['Class'] == 1]
legit = data[data['Class'] == 0]

legit.Amount.describe()

fraud.Amount.describe()

# Lets check the fraudulent transactions occur more often during certain time
frame

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(legit.Time, legit.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()

# Feature Engineering
# Add new features
X['Transaction_hour'] = pd.to_datetime(X['Time'], unit='s').dt.hour
X['Normalized_amount'] = (X['Amount'] - X['Amount'].mean()) / X['Amount'].std() #
Normalizes the Amount column using z-score normalization

#X = X.drop(['Time'], axis=1)

# Import SMOTE
from imblearn.over_sampling import SMOTE
# Convert 'Time' feature to numerical format (e.g., seconds)
X['Time_seconds'] = (X['Time'] - X['Time'].min()).dt.total_seconds()

# Drop the original 'Time' feature
X = X.drop(['Time'], axis=1)

# Now, you can proceed with SMOTE oversampling
```

```

# Use SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)

# Visualize the distribution of 'Class' (target variable) after SMOTE
plt.figure(figsize=(8, 6))
y.value_counts().plot(kind='bar', color=['blue', 'red'])
plt.title('Distribution of Class after SMOTE (0: Non-fraudulent, 1: Fraudulent)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()

# Feature Selection
# Select features
selected_features = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
                    'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',
                    'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28',
                    'Transaction_hour', 'Normalized_amount']

from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif

# Perform PCA for dimensionality reduction
n_components = min(X.shape[0], X.shape[1]) # Number of components should be
less than or equal to the minimum of samples or features
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X)

# Perform feature selection on the PCA-transformed data
k_best_selector = SelectKBest(score_func=f_classif, k=5) # Adjust k as needed
X_k_best = k_best_selector.fit_transform(X_pca, y)

# Get the indices of selected features
selected_indices = k_best_selector.get_support(indices=True)

# Map selected PCA components back to original feature names

```

```
selected_features = [selected_features[i] for i in selected_indices]

print("Selected features using ANOVA F-test after PCA:")
print(selected_features)

X=X[selected_features]
X.head()

# Split the SMOTE-resampled data into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.linear_model import LogisticRegression

# Create a logistic regression model
logistic_model = LogisticRegression(random_state=42)

# Train the model on the training data
logistic_model.fit(X_train, y_train)

# Predict on the test data
y_pred_logistic = logistic_model.predict(X_test)

# Evaluate the model
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
conf_matrix_logistic = confusion_matrix(y_test, y_pred_logistic)
class_report_logistic = classification_report(y_test, y_pred_logistic)

print("Logistic Regression Model Evaluation:")
print("Accuracy:", accuracy_logistic)
print("Classification Report:\n", class_report_logistic)

LABELS = ['Legit', 'Fraud']
plt.figure(figsize=(4, 4))
sns.set(font_scale=1.1)
sns.heatmap(conf_matrix_logistic, cmap='Spectral', xticklabels=LABELS,
yticklabels=LABELS, annot=True, fmt='d')
plt.title('Confusion Matrix for Logistic Regression')
plt.ylabel('True Class')
plt.xlabel('Predicted Class')
```

```

plt.show()

# Model Selection and Training
model = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the model on the training data
model.fit(X_train, y_train)

# Cross-validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5)
print("Cross-validation Scores:", cv_scores)
print("Mean Cross-validation Score:", np.mean(cv_scores))

# Model Evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Random Forest Classifier Model Evaluation:")
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)

LABELS = ['Legit', 'Fraud']
plt.figure(figsize=(4, 4))
sns.set(font_scale=1.1)
sns.heatmap(conf_matrix, cmap='Spectral', xticklabels=LABELS, yticklabels=LABELS,
annot=True, fmt='d')
plt.title('Confusion Matrix for Random Forest Classifier Model')
plt.ylabel('True Class')
plt.xlabel('Predicted Class')
plt.show()

from sklearn.metrics import roc_auc_score

# Assuming model is your trained classifier
y_pred_proba = model.predict_proba(X_test)[:, 1] # Probability estimates of the
positive class

```

```
# Calculate ROC AUC score
roc_auc = roc_auc_score(y_test, y_pred_proba)
print("ROC AUC Score:", roc_auc)

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Calculate the false positive rate (FPR) and true positive rate (TPR)
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Calculate the area under the ROC curve (AUC)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', 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')
plt.legend(loc="lower right")
plt.show()

LABELS = ['Legit', 'Fraud']
plt.figure(figsize=(4, 4))
sns.set(font_scale=1.1)
sns.heatmap(conf_matrix, cmap='Spectral', xticklabels=LABELS, yticklabels=LABELS,
            annot=True, fmt='d')
plt.title('Confusion Matrix for Random Forest Classifier')
plt.ylabel('True Class')
plt.xlabel('Predicted Class')
plt.show()

# Hyperparameter Tuning using GridSearchCV
param_grid = {
    'n_estimators': [1, 5, 10],
    'max_depth': [None, 5, 10, 20],
```



```
'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(model, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
print("Best Hyperparameters:", best_params)

# Model Evaluation
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)

# Save the best model
joblib.dump(best_model, 'Credit_Card_fraud_detector.pkl')

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Additional Visualizations
# Visualize the distribution of 'Class' (target variable)
plt.figure(figsize=(8, 6))
data['Class'].value_counts().plot(kind='bar', color=['blue', 'red'])
plt.title('Distribution of Class (0: Non-fraudulent, 1: Fraudulent)')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.savefig('class_distribution.jpg') # Save the visualization as a JPEG image
plt.close()

# Creating a heatmap for correlation matrix
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.savefig('correlation_matrix.jpg') # Save the visualization as a JPEG image
plt.close()

# Scatter plot to visualize the actual vs. predicted classes for test data
plt.figure(figsize=(15, 6))
plt.scatter(range(len(y_test)), y_test, color='blue', marker='o', label='Actual')
plt.scatter(range(len(y_test)), y_pred, color='red', marker='x', label='Predicted')
plt.xlabel('Transaction Index')
plt.ylabel('Class (0: Non-fraudulent, 1: Fraudulent)')
plt.title('Actual vs. Predicted Classes for Test Data')
plt.legend()
plt.savefig('actual_vs_predicted.jpg') # Save the visualization as a JPEG image
plt.close()

# Plot the transaction volume over time
plt.figure(figsize=(12, 6))
plt.plot(data['Time'], data['Amount'], color='blue', alpha=0.5)
plt.title('Transaction Volume Over Time')
plt.xlabel('Time')
plt.ylabel('Transaction Amount')
plt.grid(True)
plt.savefig('transaction_volume_over_time.jpg') # Save the visualization as a JPEG image
plt.close()

# Provide links to download each image
print("Download Class Distribution visualization: class_distribution.jpg")
print("Download Correlation Matrix visualization: correlation_matrix.jpg")
print("Download Actual vs. Predicted visualization: actual_vs_predicted.jpg")
print("Download Transaction Volume Over Time visualization: transaction_volume_over_time.jpg")
```

Visualisation:

Accuracy: 0.9995435553526912

Confusion Matrix:

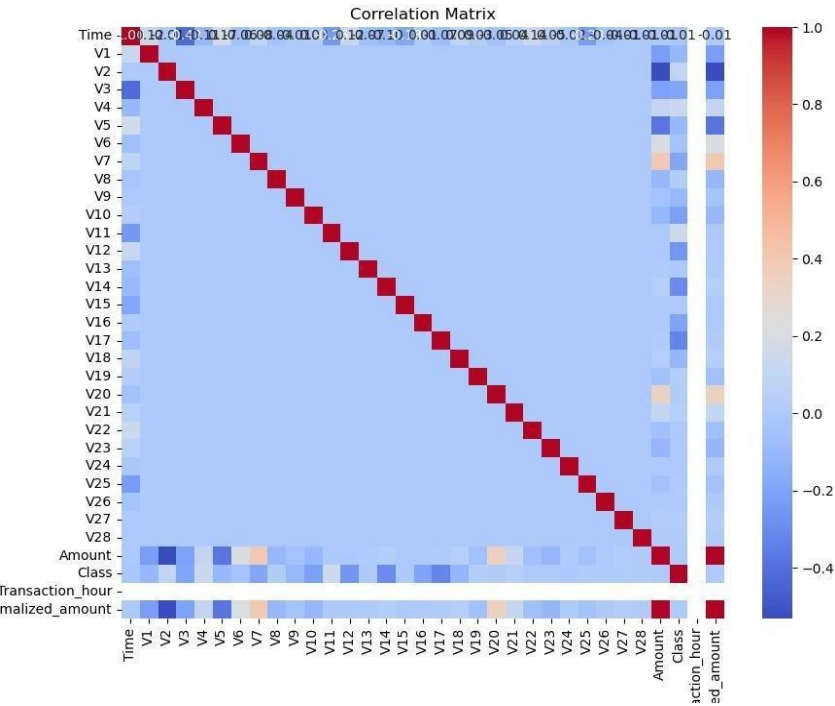
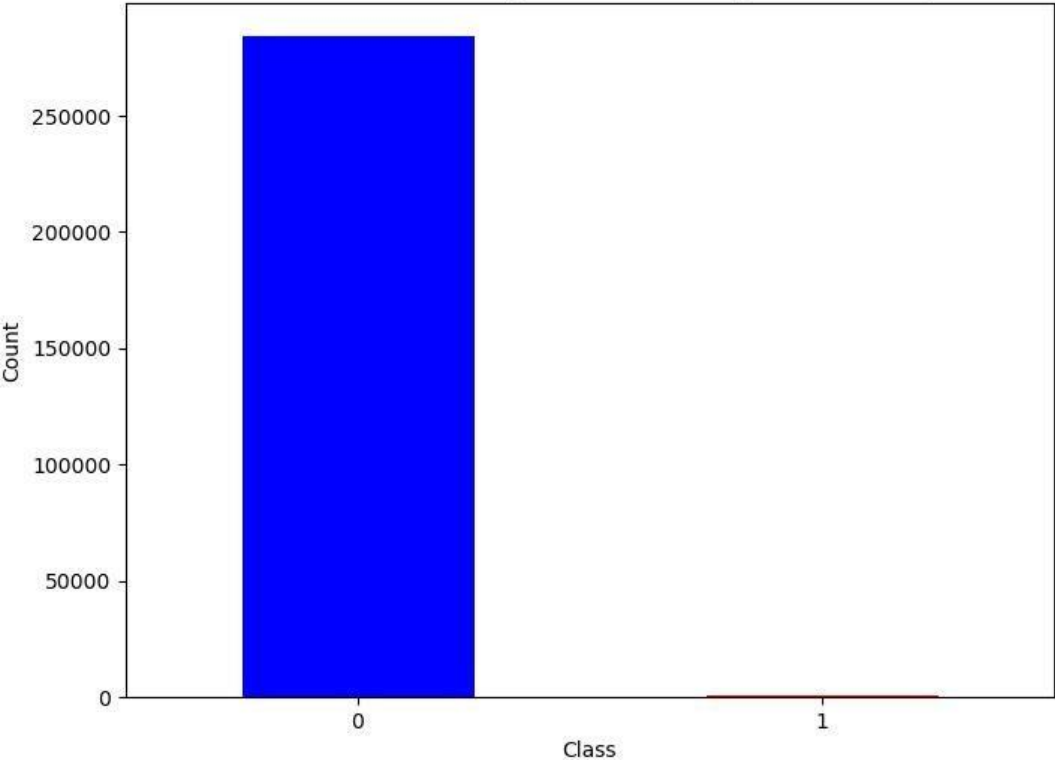
```
[[56860    4]
```

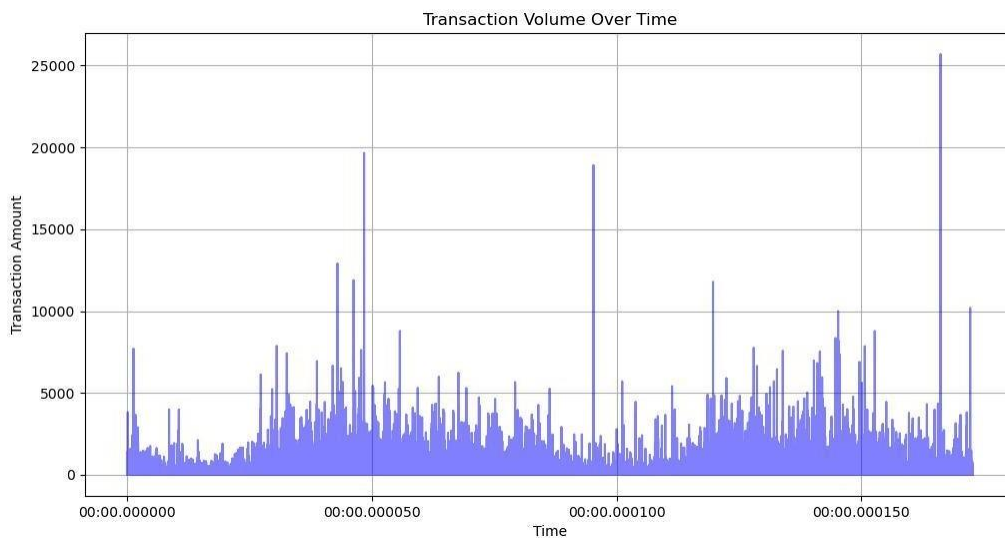
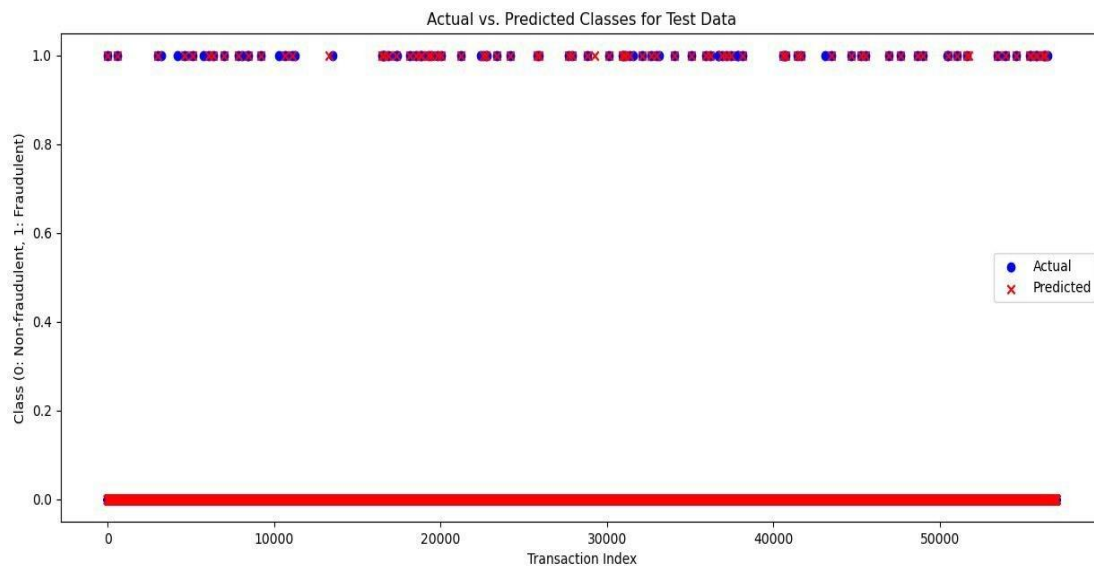
```
[   22   76]]
```

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.95	0.78	0.85	98
accuracy			1.00	56962
macro avg	0.97	0.89	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Distribution of Class (0: Non-fraudulent, 1: Fraudulent)





CONCLUSION

Through this project, we have developed a robust solution for credit card fraud detection using predictive modelling techniques. By accurately identifying fraudulent transactions, we aim to enhance the financial integrity of both consumers and institutions. This project reflects our commitment to leveraging data science for the benefit of our company and its stakeholders.

Credit card fraud poses a significant risk to both consumers and financial institutions. In this report, we outline our approach to tackling this challenge through predictive modeling techniques. Our objective is to develop a robust solution capable of accurately identifying fraudulent transactions while minimizing false positives.

The source code for the pipeline can be found in the attached files. For installation and execution instructions, please refer to the README file included in the zip archive.