PROJECT REPORT-5: Predicting Credit Card Fraud Detection

Data Collection: Gather transaction data in CSV format.

(Source: Kaggle)

Data Preprocessing:

Data loading

Subtask:

Load the credit card transaction data from the provided CSV file into a pandasDataFrame.

CODE:

```
import pandas as pd

try:
    df = pd.read_csv('synthetic_fraud_dataset.csv')
    display(df.head())
    print(df.shape)
except FileNotFoundError:
    print("Error: 'synthetic_fraud_dataset.csv' not found.")
    df = None
except pd.errors.ParserError:
    print("Error: Could not parse the CSV file.")
    df = None
except Exception as e:
    print(f"An unexpected error occurred: {e}")
    df = None
```

Data exploration

Subtask:

Explore the dataset to understand its structure, identify potential missing values, and examine the distribution of features, including the target variable (fraudulent transactions). Determine the data types of each column.

CODE:

```
# Check the dimensions of the DataFrame
print("DataFrame Shape:", df.shape)
```

```
# Get a concise summary of the DataFrame
df.info()

# Descriptive statistics for numerical features
print("\nDescriptive Statistics:\n", df.describe())

# Distribution of categorical features
for col in df.select_dtypes(include=['object']).columns:
    print(f"\nValue Counts for {col}:\n{df[col].value_counts()}")

# Correlation between numerical features and the target variable
print("\nCorrelation Matrix:\n", df.corr()['Fraud_Label'])
```

Data cleaning

Subtask:

Clean the data by handling missing values and removing duplicate rows.

```
CODE:
# Check for missing values
print("Missing values before cleaning:\n", df.isnull().sum())

# Handle missing values (if any)
# In this case, there are no missing values, so no imputation is
needed.

# Remove duplicate rows
df.drop_duplicates(inplace=True)

# Verify the cleaning
print("\nMissing values after cleaning:\n", df.isnull().sum())
print("\nShape of the DataFrame after cleaning:", df.shape)
print("\nDataFrame info after cleaning:")
df.info()
```

Data preparation

Subtask:

Prepare the data for model training by converting categorical features to numerical representations and scaling numerical features.

CODE:

```
from sklearn.preprocessing import OneHotEncoder, StandardScaler
import pandas as pd
```

```
import numpy as np

# Identify categorical and numerical features
categorical_cols = ['Transaction_Type', 'Device_Type', 'Location',
'Merchant_Category', 'Card_Type', 'Authentication_Method']
numerical_cols = ['Transaction_Amount', 'Account_Balance',
'IP_Address_Flag', 'Previous_Fraudulent_Activity',
'Daily_Transaction_Count', 'Avg_Transaction_Amount_7d',
'Failed_Transaction_Count_7d', 'Card_Age', 'Transaction_Distance',
'Risk_Score', 'Is_Weekend']

# Encode categorical features
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
encoded_features = encoder.fit_transform(df[categorical_cols])
encoded_df = pd.DataFrame(encoded_features,
columns=encoder.get_feature_names_out(categorical_cols))


# Scale numerical features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[numerical_cols])
scaled_df = pd.DataFrame(scaled_features, columns=numerical_cols)

# Concatenate encoded and scaled features
df_prepared = pd.concat([encoded_df, scaled_df, df['Fraud_Label']],
axis=1)

display(df_prepared.head())
```

Data splitting

Subtask:

Split the prepared data into training, validation, and testing sets.

CODE:

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = df_prepared.drop('Fraud_Label', axis=1)
y = df_prepared['Fraud_Label']

# Split data into training and temporary sets (validation + testing)
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.3, random_state=42, stratify=y)

# Split temporary set into validation and testing sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42, stratify=y_temp)
```

```
# Print the shapes of the resulting sets to verify the split
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_val shape:", X_val.shape)
print("y_val shape:", y_val.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

EDA:

Data visualization

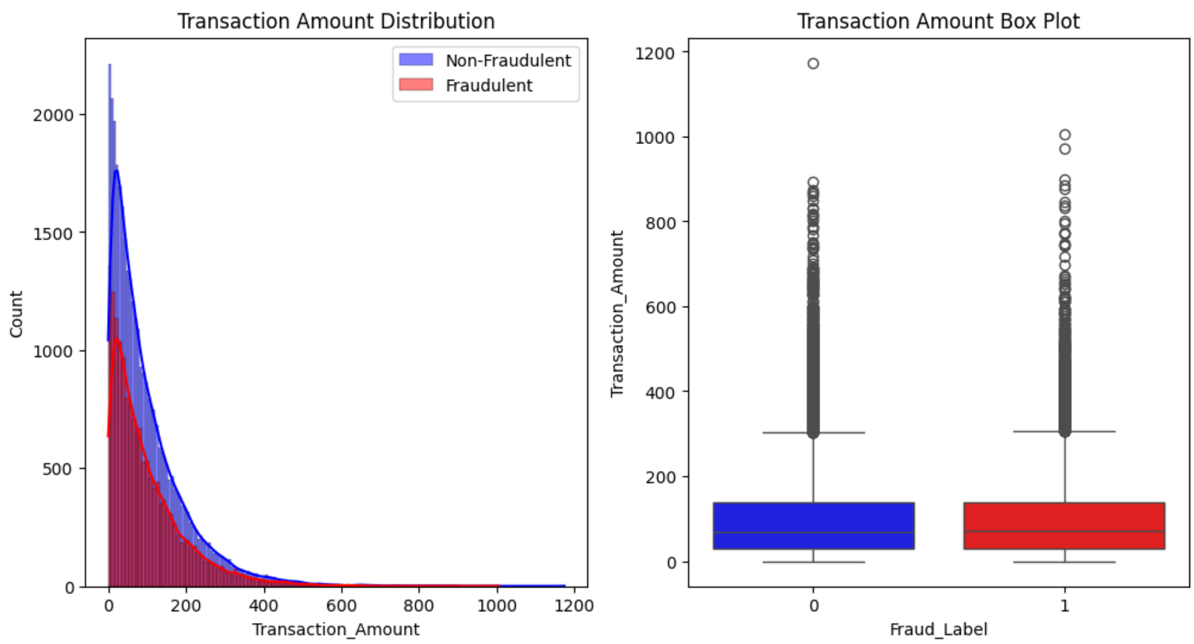
Subtask:

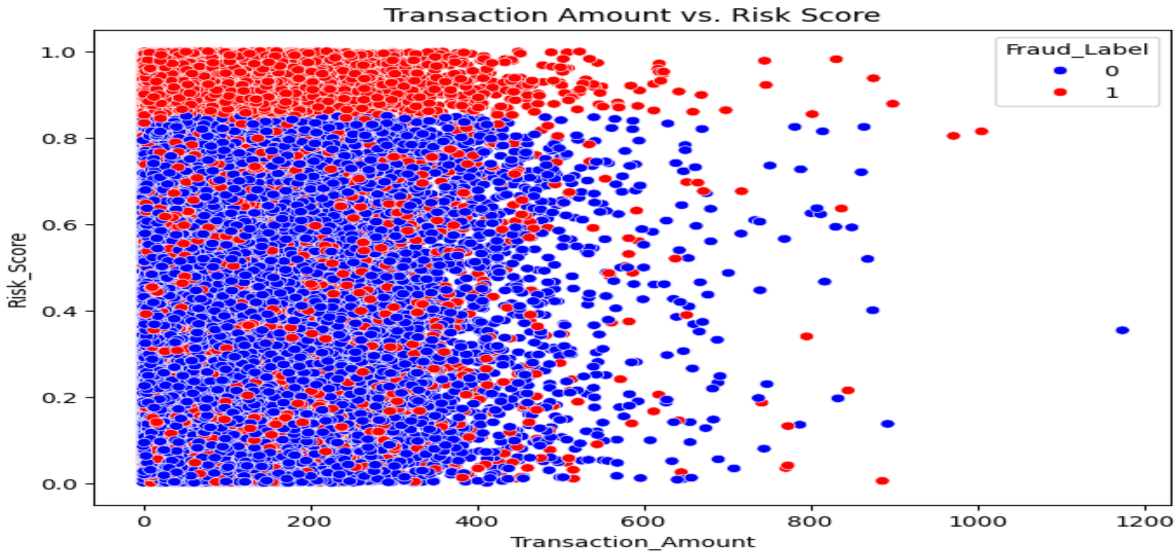
Create visualizations to illustrate the distributions of key features, the relationship between features and the target variable, and the model's performance.

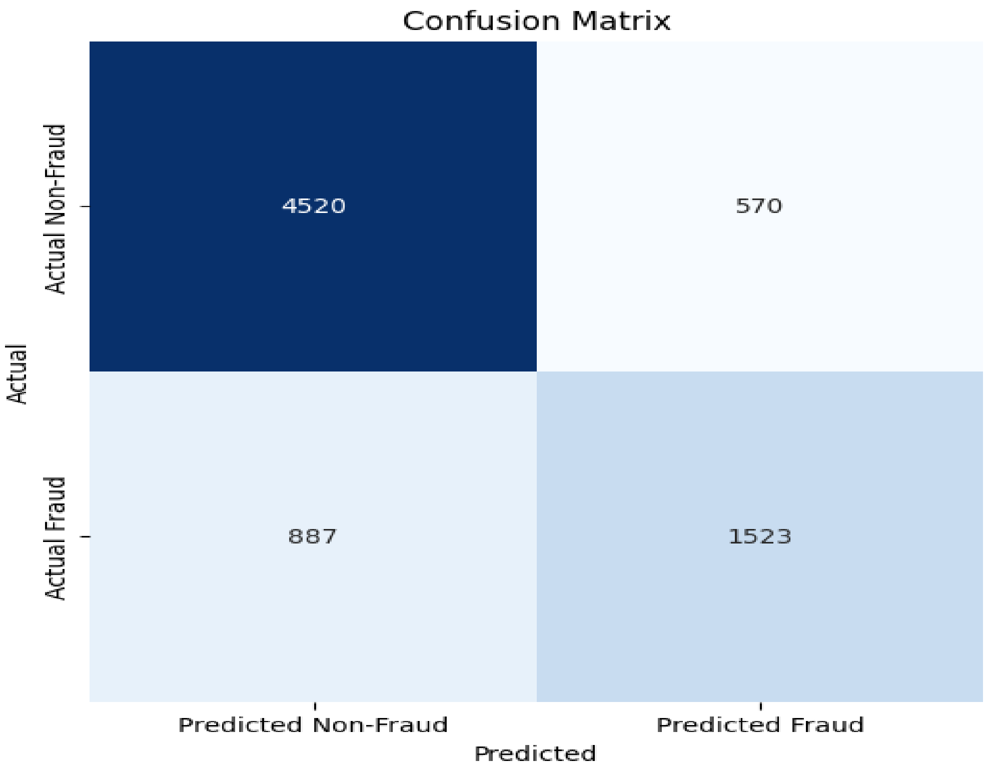
CODE:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_curve, auc

# Feature Distributions
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(df[df['Fraud_Label'] == 0]['Transaction_Amount'],
color='blue', label='Non-Fraudulent', kde=True)
sns.histplot(df[df['Fraud_Label'] == 1]['Transaction_Amount'],
color='red', label='Fraudulent', kde=True)
plt.title('Transaction Amount Distribution')
plt.legend()

plt.subplot(1, 2, 2)
sns.boxplot(x='Fraud_Label', y='Transaction_Amount', data=df,
palette={'0': 'blue', '1': 'red'})
plt.title('Transaction Amount Box Plot')
plt.show()

# Feature Relationships
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Transaction_Amount', y='Risk_Score',
hue='Fraud_Label', data=df, palette={'0': 'blue', '1': 'red'})
plt.title('Transaction Amount vs. Risk Score')
plt.show()

# Model Performance: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
```

```
            xticklabels=['Predicted Non-Fraud', 'Predicted Fraud'],
            yticklabels=['Actual Non-Fraud', 'Actual Fraud'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Model Performance: ROC Curve (already generated in the previous step)

# Additional Visualizations
plt.figure(figsize=(6, 4))
sns.countplot(x='Fraud_Label', data=df, palette=['blue', 'red'])
plt.title('Fraudulent vs. Non-Fraudulent Transactions')
plt.show()
```




Transaction Amount vs. Risk Score
Fraud_Label
0
1
Risk_Score
Transaction_Amount
0
200
400
600
800
1000
1200
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix
4520
570
887
1523
Actual Non-Fraud
Actual Fraud
Actual
Predicted Non-Fraud
Predicted Fraud
Predicted

Model training

Subtask:

Train a Logistic Regression model on the training data.

```
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000, random_state=42)

# Train the model using the training data
logreg_model.fit(X_train, y_train)
```

Model evaluation

Subtask:

Evaluate the performance of the optimized Logistic Regression model on the test set.

addCode
addText

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# Predict on the test set
y_pred = best_logreg_model.predict(X_test)

# Calculate evaluation metrics
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)
conf_matrix = confusion_matrix(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

# Print evaluation metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"ROC AUC: {roc_auc}")

# Generate ROC curve
y_prob = best_logreg_model.predict_proba(X_test)[:, 1]
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')  # Random classifier line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```