```
# Step 1: Install Necessary Libraries (if not already installed)
# Uncomment the following line to install libraries if needed
!pip install pandas numpy scikit-learn imbalanced-learn
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.12.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

```
# Step 2: Import Standard Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Step 3: Load the Dataset from GitHub Repository
url = "https://raw.githubusercontent.com/IndulekhaKP/credit-card-fraud-detection/main/data/creditcard.csv"
data = pd.read_csv(url)
print(f"Dataset shape: {data.shape}")
data.head(100)
```

```
Dataset shape: (284807, 31)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	64.0	-0.658305	0.406791	2.037461	-0.291298	0.147910	-0.350857	0.945373	-0.172560	0.025133	...	-0.156096	-0.238805	0.089877
96	64.0	0.959602	0.370711	0.888613	2.343244	0.352491	1.365515	-0.277771	0.516053	-0.700929	...	-0.155547	-0.403239	0.356504
97	67.0	-0.653445	0.160225	1.592256	1.296832	0.997175	-0.343000	0.469937	-0.132470	-0.197794	...	0.038363	0.336449	-0.014883
98	67.0	-1.494668	0.837241	2.628211	3.145414	-0.609098	0.258495	-0.012189	0.102136	-0.286164	...	-0.140047	0.355044	0.332720
99	68.0	1.232996	0.189454	0.491040	0.633673	-0.511574	-0.990609	0.066240	-0.196940	0.075921	...	-0.251566	-0.770139	0.125998

100 rows × 31 columns

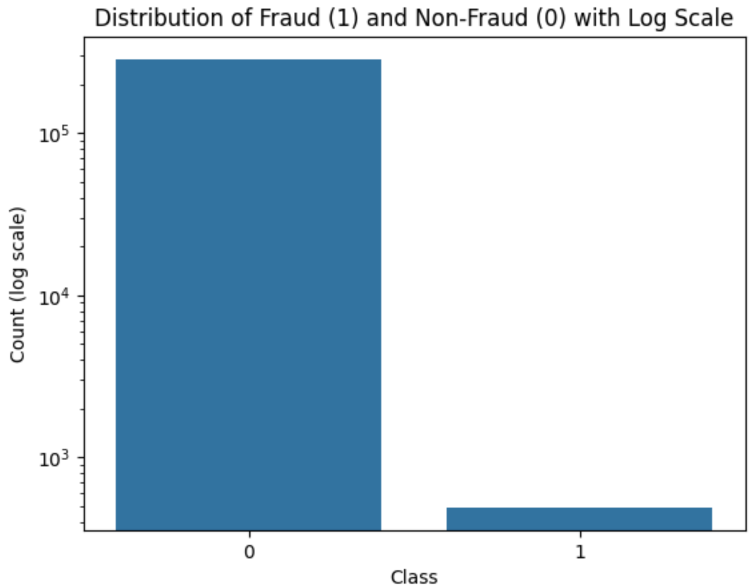
```
# Step 4: Check for Missing Values
print("Missing values per column:\n", data.isnull().sum())
```

```
Missing values per column:
 Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
```

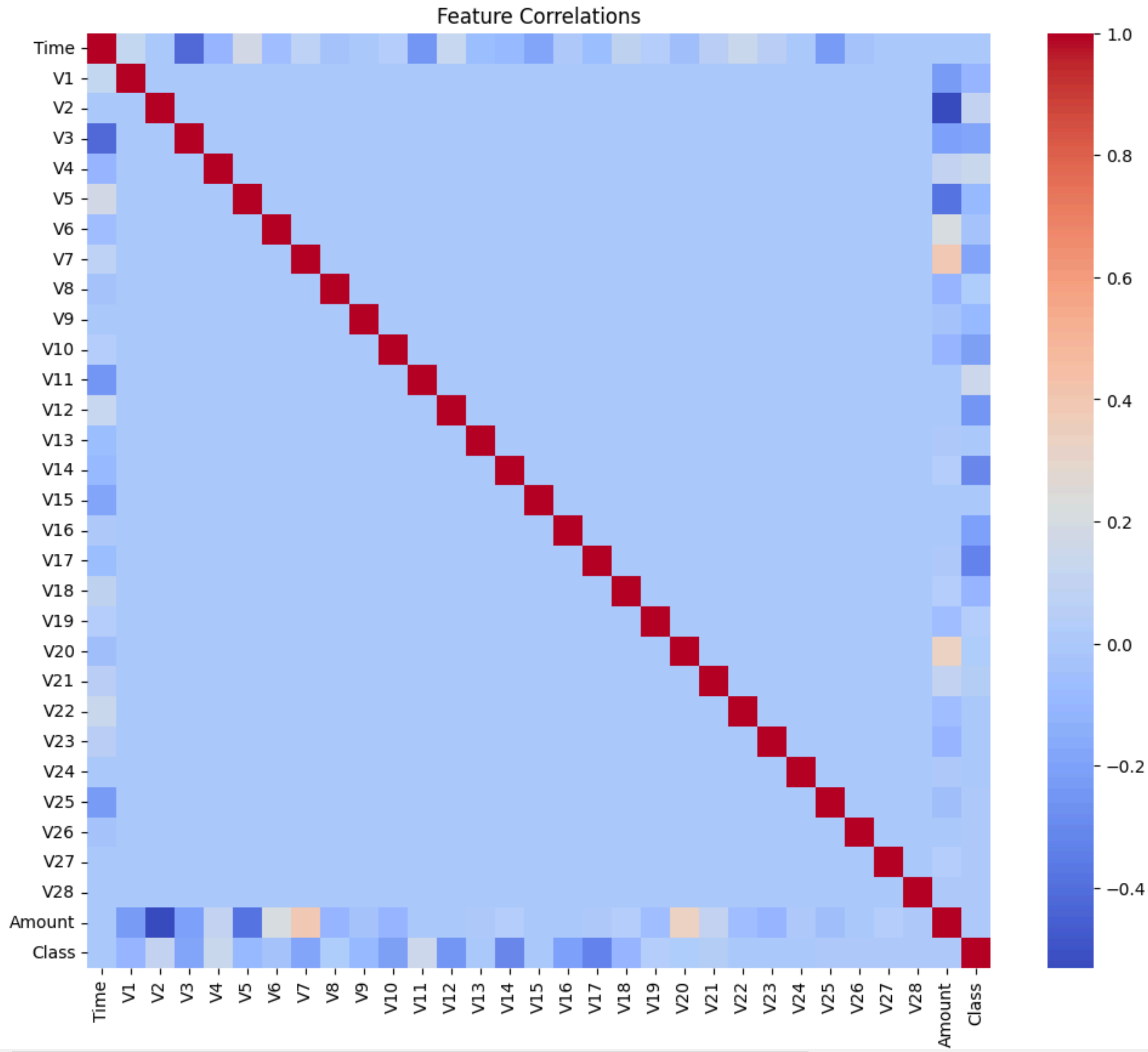
```
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
# Step 5: Visualize the Class Distribution (Fraud and Non-Fraud)
sns.countplot(x='Class', data=data)
plt.title("Distribution of Fraud (1) and Non-Fraud (0)")
plt.show()
```


```
# Step 6: Visualize the Class Distribution with a Log Scale
sns.countplot(x='Class', data=data)
plt.yscale('log')
plt.title("Distribution of Fraud (1) and Non-Fraud (0) with Log Scale")
plt.ylabel("Count (log scale)")
plt.show()
```




```
# Plot Correlation Heatmap to Identify Feature Relationships
plt.figure(figsize=(12, 10))
correlation_matrix = data.corr()
sns.heatmap(correlation_matrix, cmap="coolwarm", annot=False)
plt.title("Feature Correlations")
plt.show()
```



Start coding or generate with AI.

```
# Step 7: Double-Check for Missing Values
print("Missing values per column:\n", data.isnull().sum())
```

```
Missing values per column:
 Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
```

```
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
# Step 8: Define Features (X) and Target Variable (y)
X = data.drop(columns=['Class'])  # Features
y = data['Class']  # Target variable (Fraud or Non-Fraud)
```

```
# Step 9: Split the Data into Training and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Step 10: Scale the Training and Test Data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# Step 11: Initialize Logistic Regression with Class Weights
model_logreg = LogisticRegression(class_weight='balanced', random_state=42, max_iter=2000)
```

```
# Step 12: Train the Logistic Regression Model
model_logreg.fit(X_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(class_weight='balanced', max_iter=2000, random_state=42)
```

```
# Step 13: Predict on the Test Data
y_pred = model_logreg.predict(X_test_scaled)
```

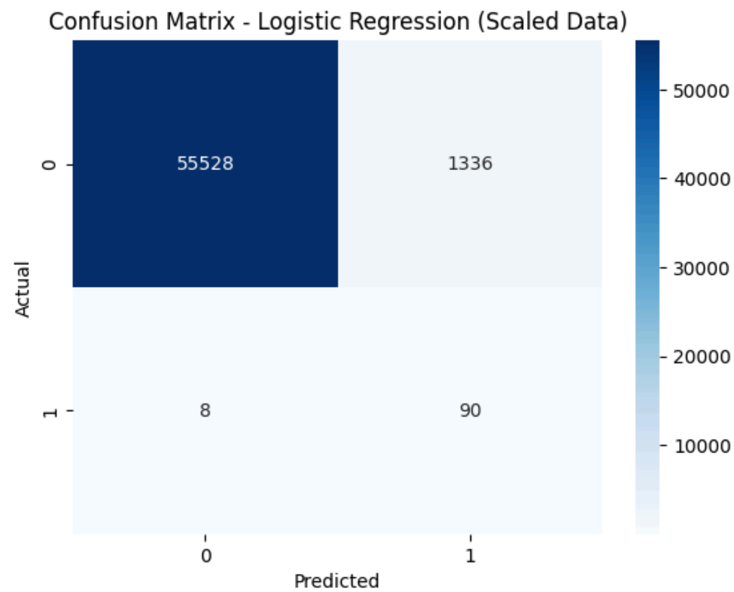
```
# Step 14: Evaluate the Model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nAccuracy Score:\n", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[55528  1336]
 [    8    90]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.98      0.99     56864
           1       0.06      0.92      0.12        98

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


Accuracy Score:
 0.9764053228468101
```

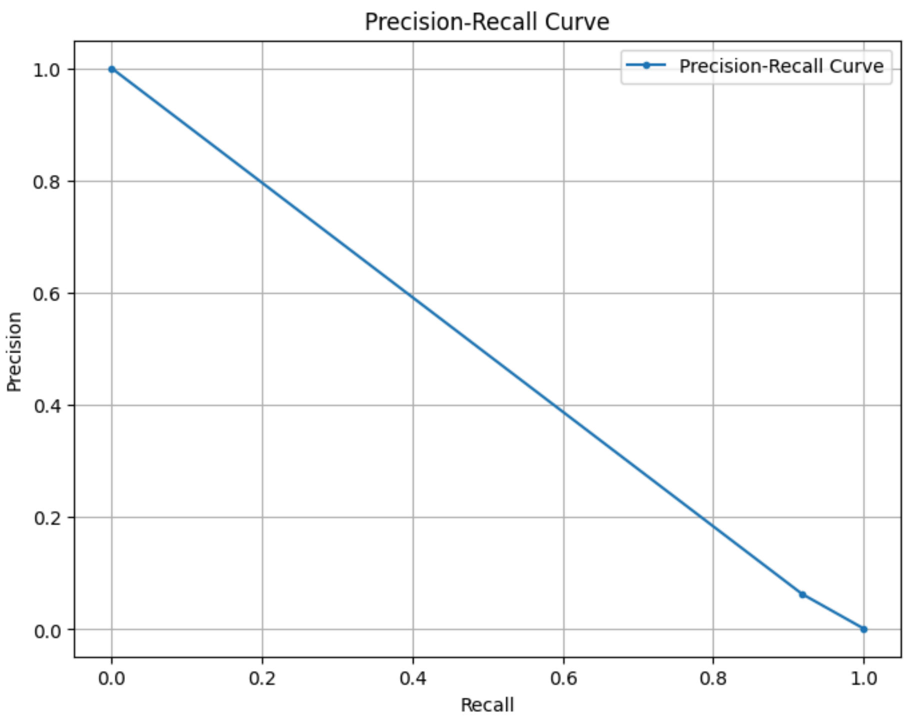
```
# Step 15: Generate Plots
```

```
# Plot the confusion matrix to visualize model performance
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Logistic Regression (Scaled Data)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
plt.show()
```




```
# Plot Precision-Recall Curve
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_test, y_pred)
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.', label='Precision-Recall Curve')
plt.title("Precision-Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.legend()
plt.grid()
plt.show()
```




```
# Plot ROC Curve
from sklearn.metrics import roc_curve, auc

y_pred_prob = model_logreg.predict_proba(X_test_scaled)[:, 1]
```

```
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid()
plt.show()
```