```
# Load the Data: Use a library like Pandas in Python to load the dataset from the CSV file.
```

In [6]:

```
import pandas as pd

# Load the CSV file into a DataFrame
file_path = "C:/Users/Deep/Documents/Projects Shivam/Final Project/Project-2-Prediction of Credit
data = pd.read_csv(file_path)

# Display the first few rows of the DataFrame to verify it was loaded correctly
print(data.head())
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

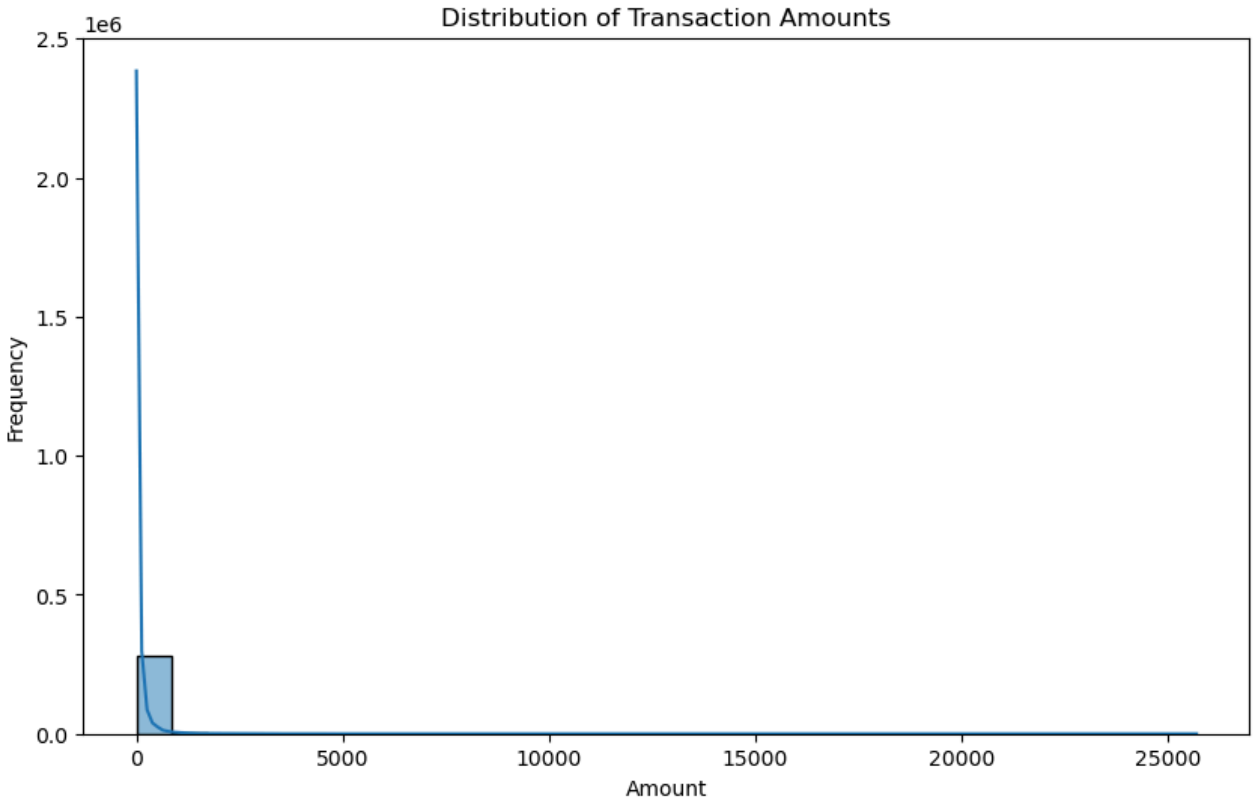
```
# Exploratory Data Analysis (EDA): Analyze the data to understand its structure, distributions, and relationships between variables.
```

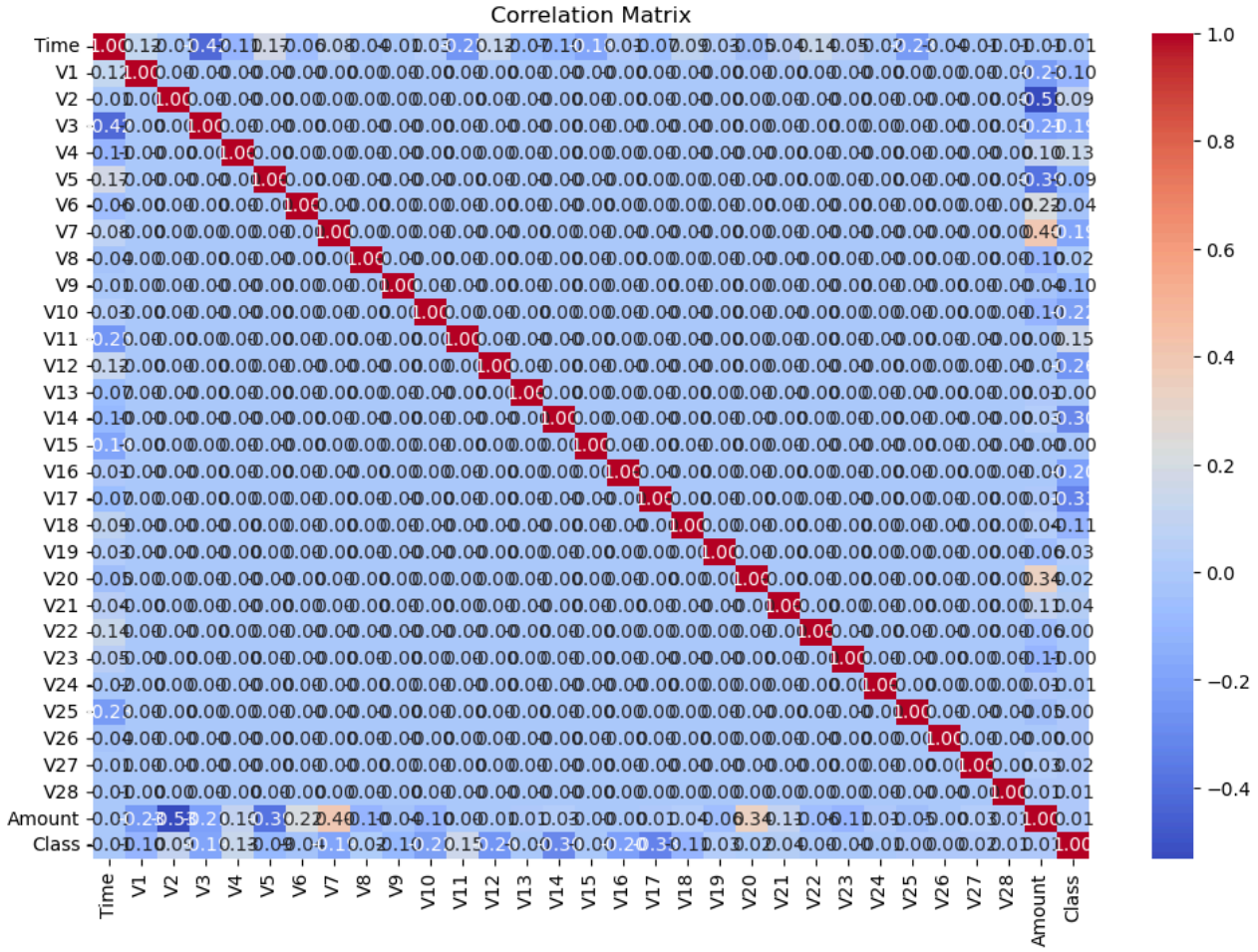
In [2]:

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

# Load the dataset
file_path = "C:/Users/Deep/Documents/Projects Shivam/Final Project/Project-2-Prediction of Credit
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
print(data.head())

# Check dimensions of the dataset
print("Dimensions:", data.shape)

# Summary statistics
print(data.describe())

# Data visualization
# Example: Histogram of transaction amounts
plt.figure(figsize=(10, 6))
sns.histplot(data['Amount'], bins=30, kde=True)
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.show()

# Example: Bar chart of class distribution
plt.figure(figsize=(6, 4))
sns.countplot(data['Class'])
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()

# Example: Correlation matrix heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), cmap='coolwarm', annot=True, fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
Dimensions: (284807, 31)
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.759061e-12 -8.251130e-13 -9.654937e-13  8.321385e-13
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.649999e-13  4.248366e-13 -3.054600e-13  8.777971e-14 -1.179749e-12
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ... -3.405756e-13 -5.723197e-13 -9.725856e-13  1.464150e-12
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean  -6.987102e-13 -5.617874e-13  3.332082e-12 -3.518874e-12      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

              Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
```

```
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```



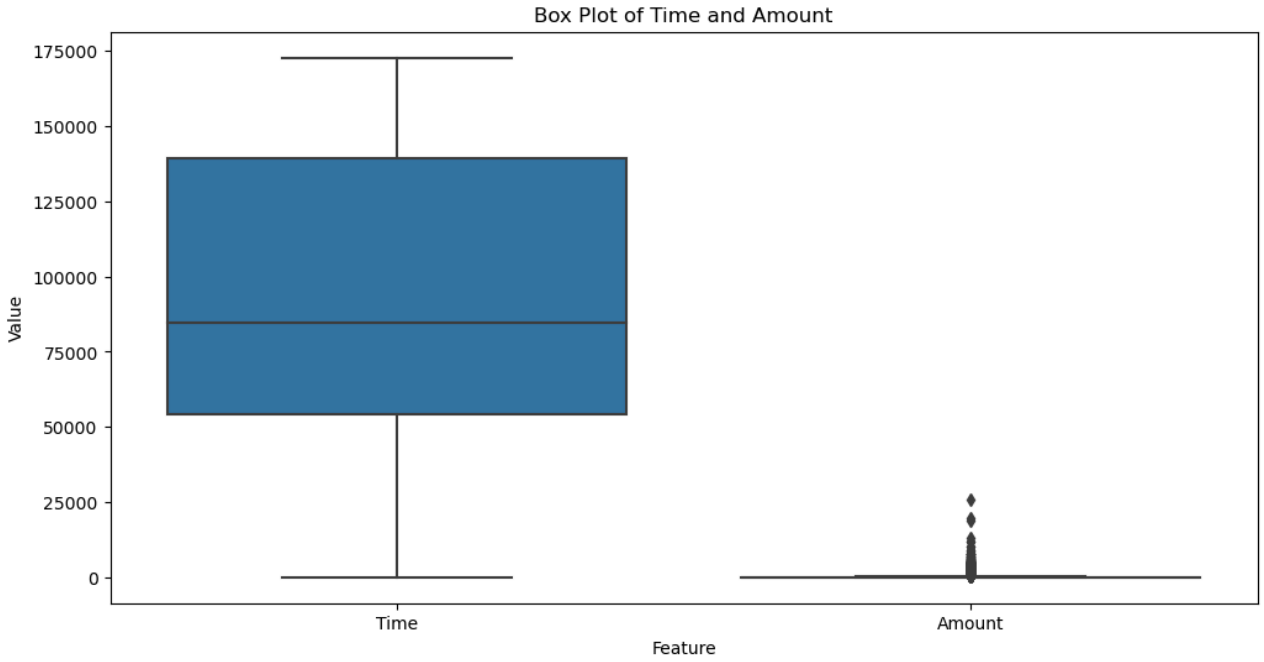



```
1 # Data Cleaning: Check for missing values, outliers, or inconsistencies in the data and handle them appropriately.
```

In [4]:

```
import numpy as np  # Add this line to import NumPy

# Check for missing values
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)

# Remove rows with missing values (if needed)
# data.dropna(inplace=True)

# Handle missing values by imputation (if needed)
# data.fillna(data.mean(), inplace=True)

# Identify outliers using z-score
from scipy import stats
z_scores = stats.zscore(data[['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10'
abs_z_scores = np.abs(z_scores)  # Use np.abs() here
outliers = (abs_z_scores > 3).all(axis=1)
print("Number of Outliers:", outliers.sum())

# Handle outliers by removing them (if needed)
# data = data[~outliers]

# Handle outliers by transforming the feature (if needed)
# data['Amount'] = np.log(data['Amount'] + 1)  # Log transformation

# Visualize outliers using box plots (optional)
plt.figure(figsize=(12, 6))
sns.boxplot(data=data[['Time', 'Amount']])
plt.title('Box Plot of Time and Amount')
plt.xlabel('Feature')
plt.ylabel('Value')
plt.show()
```

```
Missing Values:
 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
Number of Outliers: 0
```




```
1 # Feature Engineering: Create new features or transform existing ones to improve model performance.
```

In [5]:
```
# Example of feature engineering
import pandas as pd

# Extract time-related features
data['Hour'] = pd.to_datetime(data['Time'], unit='s').dt.hour
data['DayOfWeek'] = pd.to_datetime(data['Time'], unit='s').dt.dayofweek

# Transform amount using logarithmic transformation
data['LogAmount'] = np.log(data['Amount'] + 1)  # Adding 1 to avoid log(0)

# Create interaction features
data['V1_V2'] = data['V1'] * data['V2']

# Calculate aggregate statistics
data['MeanV'] = data[['V1', 'V2', 'V3', 'V4', 'V5']].mean(axis=1)
data['StdV'] = data[['V1', 'V2', 'V3', 'V4', 'V5']].std(axis=1)

# Drop original time and amount columns if needed
# data.drop(['Time', 'Amount'], axis=1, inplace=True)

# Perform feature selection if needed
# ...

# Display the updated dataset with engineered features
print(data.head())
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V27       V28  Amount  Class  Hour  \
0  0.098698  0.363787  ...  0.133558 -0.021053  149.62      0     0
1  0.085102 -0.255425  ... -0.008983  0.014724    2.69      0     0
2  0.247676 -1.514654  ... -0.055353 -0.059752  378.66      0     0
3  0.377436 -1.387024  ...  0.062723  0.061458  123.50      0     0
4 -0.270533  0.817739  ...  0.219422  0.215153   69.99      0     0

   DayOfWeek  LogAmount     V1_V2     MeanV      StdV
0          3   5.014760  0.098968  0.428719  1.531519
1          3   1.305626  0.317214  0.426532  0.451074
2          3   5.939276  1.820416 -0.209745  1.319366
3          3   4.824306  0.178979 -0.046421  1.108763
4          3   4.262539 -1.016624  0.252812  1.062912

[5 rows x 37 columns]
```

```
## Ignore next two installations
```

In [3]:
```
pip uninstall scikit-learn
```

```
^C
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
pip install scikit-learn
```

```
# Train/Test Split: Split the data into training and testing sets for model evaluation.
```

In [9]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the CSV file into a DataFrame
df = pd.read_csv("C:/Users/Deep/Documents/Projects Shivam/Final Project/Project-2-Prediction of C

# Assuming your target variable is named 'Class'
X = df.drop(columns=['Class'])  # Features (all columns except 'Class')
y = df['Class']                 # Target variable ('Class' column)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Displaying the shapes of the training and testing sets
print("Training set - Features:", X_train.shape, "Labels:", y_train.shape)
print("Testing set - Features:", X_test.shape, "Labels:", y_test.shape)

```

```
Training set - Features: (227845, 30) Labels: (227845,)
Testing set - Features: (56962, 30) Labels: (56962,)
```

```
# Model Selection: Choose appropriate machine learning algorithms for classification (e.g., logistic regression, random forest, support vector machines).
    ##Logistic Regression algorithm
```

In [11]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Step 1: Load the CSV data into a pandas DataFrame
data = pd.read_csv("C:/Users/Deep/Documents/Projects Shivam/Final Project/Project-2-Prediction of

# Step 2: Preprocess the data (if needed)
# For example, handle missing values, scale numerical features, encode categorical variables

# Step 3: Split the data into features (X) and target variable (y)
X = data.drop(columns=['Class'])  # Features (all columns except 'Class')
y = data['Class']                 # Target variable ('Class' column)

# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Train a Logistic Regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Step 6: Evaluate the model
y_pred = logistic_model.predict(X_test)
print(classification_report(y_test, y_pred))

```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.61      0.56      0.59        98

    accuracy                           1.00     56962
   macro avg       0.81      0.78      0.79     56962
weighted avg       1.00      1.00      1.00     56962


C:\Users\Deep\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modu
les/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-le
arn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

```
# Model Training: Train the selected models on the training data.
```

In [13]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Load the CSV data into a pandas DataFrame
data = pd.read_csv("C:/Users/Deep/Documents/Projects Shivam/Final Project/Project-2-Prediction of

# Split the data into features (X) and target variable (y)
X = data.drop(columns=['Class'])  # Features (all columns except 'Class')
y = data['Class']                 # Target variable ('Class' column)

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

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the Logistic Regression model with increased max_iter
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train_scaled, y_train)

# Predict on the testing set
y_pred = logistic_model.predict(X_test_scaled)

# Evaluate the model
print(classification_report(y_test, y_pred))

```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.85      0.56      0.67        98

    accuracy                           1.00     56962
   macro avg       0.92      0.78      0.84     56962
weighted avg       1.00      1.00      1.00     56962
```

```
# Hyperparameter Tuning: Fine-tune model parameters to optimize performance.
```

In [14]:

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# Load the CSV data into a pandas DataFrame
data = pd.read_csv("C:/Users/Deep/Documents/Projects Shivam/Final Project/Project-2-Prediction of

# Split the data into features (X) and target variable (y)
X = data.drop(columns=['Class'])  # Features (all columns except 'Class')
y = data['Class']                 # Target variable ('Class' column)

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

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define the hyperparameters grid
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'penalty': ['l1', 'l2']}

# Initialize Logistic Regression model
logistic_model = LogisticRegression(max_iter=1000)

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=logistic_model, param_grid=param_grid, cv=5, scoring='accura
grid_search.fit(X_train_scaled, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Evaluate the model with best hyperparameters
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_scaled)
print(classification_report(y_test, y_pred))

```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
C:\Users\Deep\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py:547: FitFailedWarni
ng:
30 fits failed out of a total of 60.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Deep\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py", line 895,
in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Deep\anaconda3\Lib\site-packages\sklearn\base.py", line 1474, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Deep\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py", line 1172, in
fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Deep\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py", line 67, in _c
heck_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or None penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Deep\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:1051: UserWarning: One
or more of the test scores are non-finite: [       nan 0.9990476         nan 0.99916171        nan
0.99919243
        nan 0.99919682        nan 0.99919243        nan 0.99920121]
  warnings.warn(

Best Hyperparameters: {'C': 100, 'penalty': 'l2'}
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.85      0.56      0.67        98

    accuracy                           1.00     56962
   macro avg       0.92      0.78      0.84     56962
weighted avg       1.00      1.00      1.00     56962
```

In [ ]:
```
1
```

In [ ]:
```
1
```