In [1]:

```python
# William Barker
# DSC680
# Project 3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import IsolationForest
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_cu

# Set random seed for reproducibility
np.random.seed(42)
```

In [3]:

```python
# Load the dataset
df = pd.read_csv('creditcard_2023.csv')

# Check the shape and first few rows of the dataset
print(df.shape)
print(df.head())
```

```
(568630, 31)
   id        V1        V2        V3        V4        V5        V6        V7  \
0   0 -0.260648 -0.469648  2.496266 -0.083724  0.129681  0.732898  0.519014   
1   1  0.985100 -0.356045  0.558056 -0.429654  0.277140  0.428605  0.406466   
2   2 -0.260272 -0.949385  1.728538 -0.457986  0.074062  1.419481  0.743511   
3   3 -0.152152 -0.508959  1.746840 -1.090178  0.249486  1.143312  0.518269   
4   4 -0.206820 -0.165280  1.527053 -0.448293  0.106125  0.530549  0.658849   

         V8        V9  ...       V21       V22       V23       V24       V25  \
0 -0.130006  0.727159  ... -0.110552  0.217606 -0.134794  0.165959  0.126280   
1 -0.133118  0.347452  ... -0.194936 -0.605761  0.079469 -0.577395  0.190090   
2 -0.095576 -0.261297  ... -0.005020  0.702906  0.945045 -1.154666 -0.605564   
3 -0.065130 -0.205698  ... -0.146927 -0.038212 -0.214048 -1.893131  1.003963   
4 -0.212660  1.049921  ... -0.106984  0.729727 -0.161666  0.312561 -0.414116   

        V26       V27       V28    Amount  Class  
0 -0.434824 -0.081230 -0.151045  17982.10      0  
1  0.296503 -0.248052 -0.064512   6531.37      0  
2 -0.312895 -0.300258 -0.244718   2513.54      0  
3 -0.515950 -0.165316  0.048424   5384.44      0  
4  1.071126  0.023712  0.419117  14278.97      0  

[5 rows x 31 columns]
```

In [4]:

```python
# Summary statistics
print(df.describe())
```

```
                  id            V1            V2            V3            V4  \
count  568630.000000  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05   
mean   284314.500000 -1.109271e-14 -3.429498e-14 -1.209242e-14  3.825991e-15   
std    164149.486121  1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00   
min         0.000000 -3.495584e+00 -4.996657e+01 -3.183760e+00 -4.951222e+00   
25%    142157.250000 -5.652859e-01 -4.866777e-01 -6.492987e-01 -6.560203e-01   
50%    284314.500000 -9.363846e-02 -1.358939e-01  3.528579e-04 -7.376152e-02   
75%    426471.750000  8.326582e-01  3.435552e-01  6.285380e-01  7.070047e-01   
max    568629.000000  2.229046e+00  4.361865e+00  1.412583e+01  3.201536e+00   
```

```
                 V5            V6            V7            V8            V9  \
count  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05
mean   6.288281e-15 -2.751174e-14  1.240002e-14  8.208047e-15 -1.002980e-14
std    1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00
min   -9.952786e+00 -2.111111e+01 -4.351839e+00 -1.075634e+01 -3.751919e+00
25%   -2.934955e-01 -4.458712e-01 -2.835329e-01 -1.922572e-01 -5.687446e-01
50%    8.108788e-02  7.871758e-02  2.333659e-01 -1.145242e-01  9.252647e-02
75%    4.397368e-01  4.977881e-01  5.259548e-01  4.729905e-02  5.592621e-01
max    4.271689e+01  2.616840e+01  2.178730e+02  5.958040e+00  2.027006e+01

       ...           V21           V22           V23           V24  \
count  ...  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05
mean   ...  2.210679e-15 -8.767441e-16  4.376179e-16  6.825608e-16
std    ...  1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00
min    ... -1.938252e+01 -7.734798e+00 -3.029545e+01 -4.067968e+00
25%    ... -1.664408e-01 -4.904892e-01 -2.376289e-01 -6.515801e-01
50%    ... -3.743065e-02 -2.732881e-02 -5.968903e-02  1.590123e-02
75%    ...  1.479787e-01  4.638817e-01  1.557153e-01  7.007374e-01
max    ...  8.087080e+00  1.263251e+01  3.170763e+01  1.296564e+01

                V25           V26           V27           V28         Amount  \
count  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05  568630.000000
mean   2.545689e-15  1.781906e-15  2.817586e-15  2.891419e-15   12041.957635
std    1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00    6919.644449
min   -1.361263e+01 -8.226969e+00 -1.049863e+01 -3.903524e+01      50.010000
25%   -5.541485e-01 -6.318948e-01 -3.049607e-01 -2.318783e-01    6054.892500
50%   -8.193162e-03 -1.189208e-02 -1.729111e-01 -1.392973e-02   12030.150000
75%    5.500147e-01  6.728879e-01  3.340230e-01  4.095903e-01   18036.330000
max    1.462151e+01  5.623285e+00  1.132311e+02  7.725594e+01   24039.930000

          Class
count  568630.0
mean        0.5
std         0.5
min         0.0
25%         0.0
50%         0.5
75%         1.0
max         1.0

[8 rows x 31 columns]
```

In [5]:

```
# Check for missing values
print(df.isnull().sum())
```

```
id        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
```

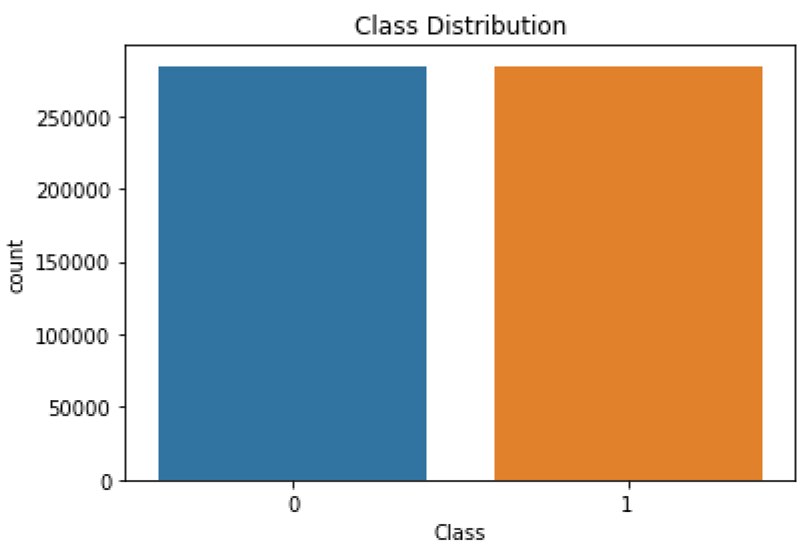
In [6]:

```
# Check the distribution of the 'Class' column
print(df['Class'].value_counts())
```

```
0    284315
1    284315
Name: Class, dtype: int64
```

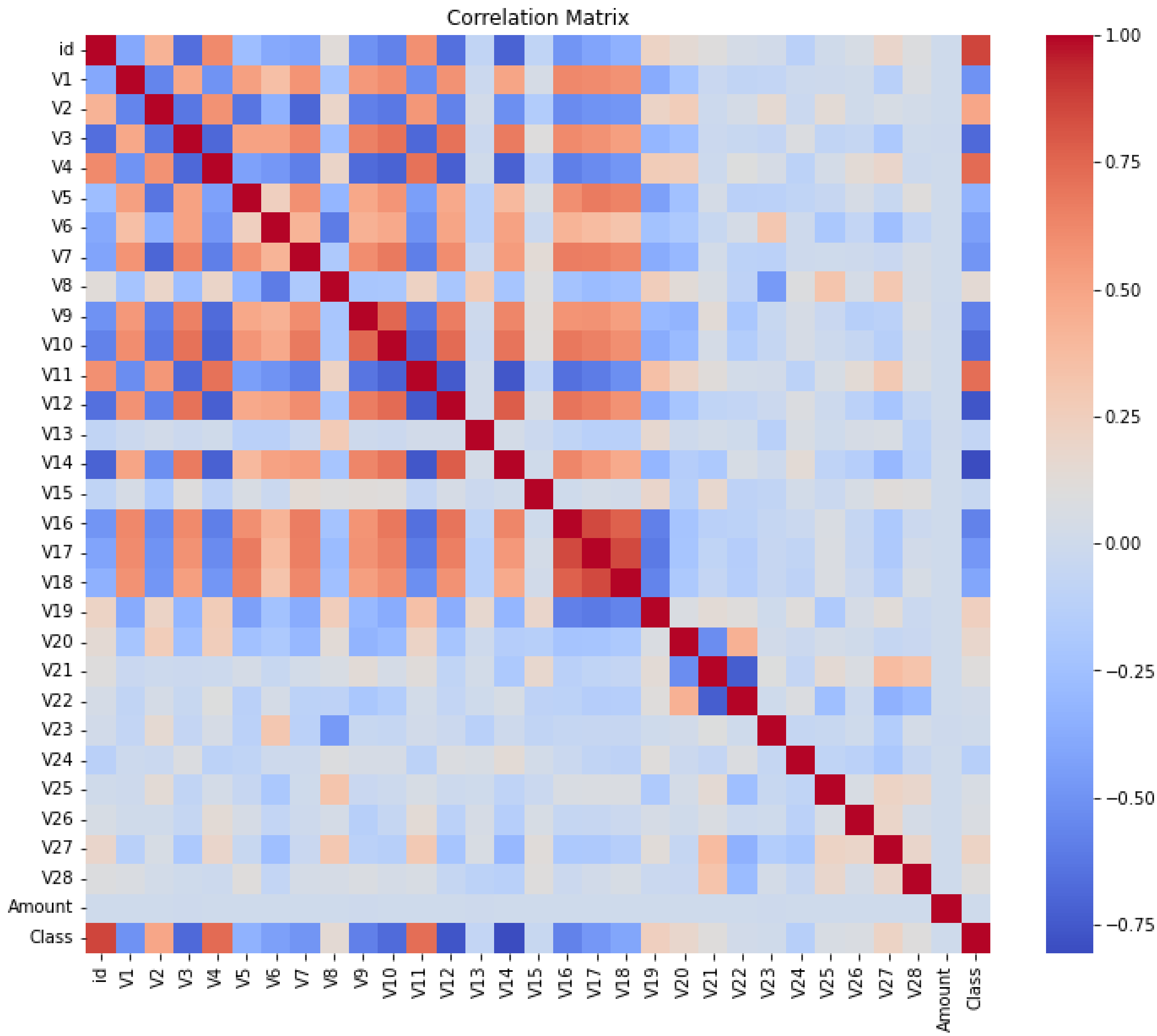
In [7]:

```
# Visualize class distribution
sns.countplot(x='Class', data=df)
plt.title('Class Distribution')
plt.show()
```



In [8]:

```
# Correlation matrix
corr_matrix = df.corr()

# Visualize the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, cmap='coolwarm', annot=False, fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



In [9]:

```
# Separate features and target variable
X = df.drop(columns=['id', 'Class'])
y = df['Class']

# Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,

# Feature scaling (only for Logistic Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [10]:

```
# Initialize Logistic Regression model
log_reg = LogisticRegression(class_weight='balanced', random_state=42)

# Train the model
log_reg.fit(X_train_scaled, y_train)
```

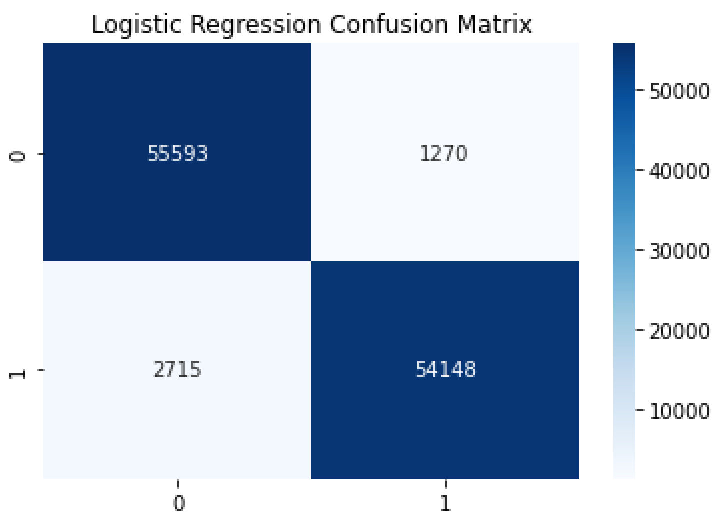
Out[10]:

```
▾ LogisticRegression
LogisticRegression(class_weight='balanced', random_state=42)
```

In [11]:
```
# Make predictions
y_pred_log_reg = log_reg.predict(X_test_scaled)
y_prob_log_reg = log_reg.predict_proba(X_test_scaled)[:, 1]

# Evaluation
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_log_reg))
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97     56863
           1       0.98      0.95      0.96     56863

    accuracy                           0.96    113726
   macro avg       0.97      0.96      0.96    113726
weighted avg       0.97      0.96      0.96    113726
```

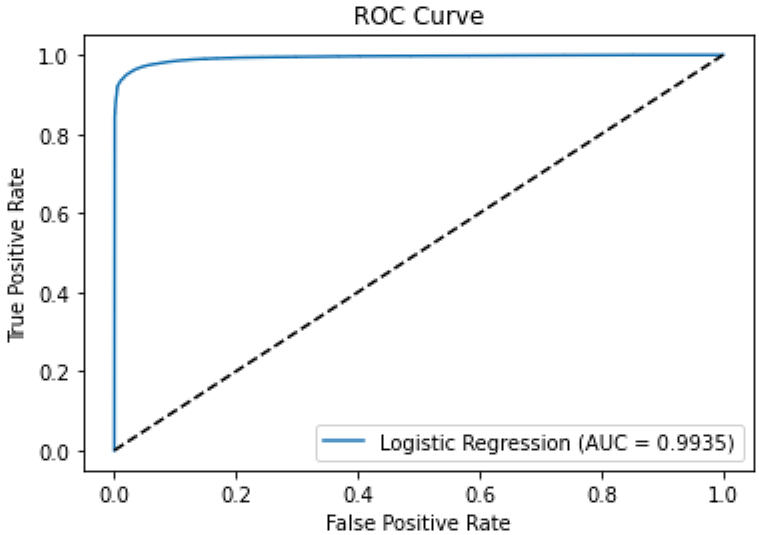
In [12]:
```
# Confusion Matrix
cm_log_reg = confusion_matrix(y_test, y_pred_log_reg)
sns.heatmap(cm_log_reg, annot=True, fmt='d', cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.show()
```



In [13]:
```
# ROC-AUC Score
roc_auc_log_reg = roc_auc_score(y_test, y_prob_log_reg)
print(f'ROC-AUC Score for Logistic Regression: {roc_auc_log_reg:.4f}')
```

```
ROC-AUC Score for Logistic Regression: 0.9935
```

In [14]:
```
# Plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob_log_reg)
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc_log_reg:.4f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



In [15]:
```
# Initialize Isolation Forest model
iso_forest = IsolationForest(contamination=0.01, random_state=42)
```

In [16]:
```
# Train the model (use training data only)
iso_forest.fit(X_train)
```

```
/Users/cameronbarker/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:439: UserWa
rning: X does not have valid feature names, but IsolationForest was fitted with feature na
mes
  warnings.warn(
```

Out[16]:
```
▾ IsolationForest
IsolationForest(contamination=0.01, random_state=42)
```

In [17]:
```
# Predict anomalies (Isolation Forest predicts -1 for anomalies, 1 for normal)
y_pred_iso_forest = iso_forest.predict(X_test)
```

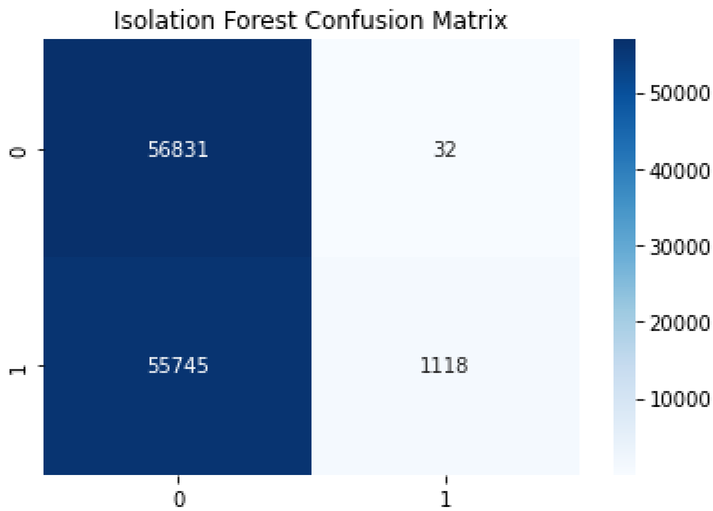
In [18]:
```
# Convert predictions to match the binary classification format (0: normal, 1: fraud)
y_pred_iso_forest = np.where(y_pred_iso_forest == -1, 1, 0)
```

In [19]:
```
# Evaluation
print("Isolation Forest Classification Report:")
print(classification_report(y_test, y_pred_iso_forest))
```

```
Isolation Forest Classification Report:
              precision    recall  f1-score   support

           0       0.50      1.00      0.67     56863
           1       0.97      0.02      0.04     56863

    accuracy                           0.51    113726
   macro avg       0.74      0.51      0.35    113726
weighted avg       0.74      0.51      0.35    113726
```

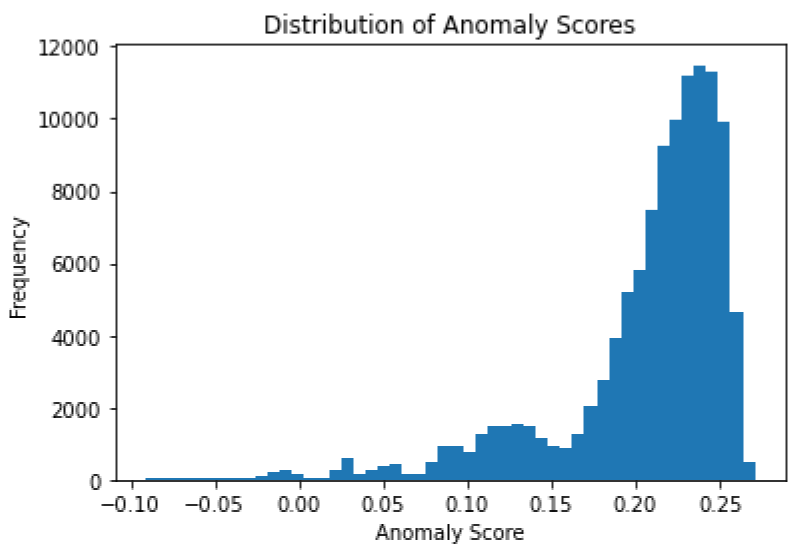
In [20]:
```
# Confusion Matrix
cm_iso_forest = confusion_matrix(y_test, y_pred_iso_forest)
sns.heatmap(cm_iso_forest, annot=True, fmt='d', cmap='Blues')
plt.title('Isolation Forest Confusion Matrix')
```

```
plt.show()

# Since Isolation Forest doesn't give probability estimates, ROC-AUC isn't applicable
```



In [21]:

```
# Get the anomaly scores for the test set
anomaly_scores = iso_forest.decision_function(X_test)

# Higher scores correspond to more "normal" data points, while lower scores are more "anor
plt.hist(anomaly_scores, bins=50)
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.title('Distribution of Anomaly Scores')
plt.show()
```



In [ ]:

```
```