Credit Card Fraud Detection

Importing Libraries

In [1]:
```python
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
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
import warnings
warnings.filterwarnings('ignore')
```

Importing Dataset

In [2]:
```python
data=pd.read_csv(r"creditcard.csv")
data
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0
...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0

284807 rows × 31 columns

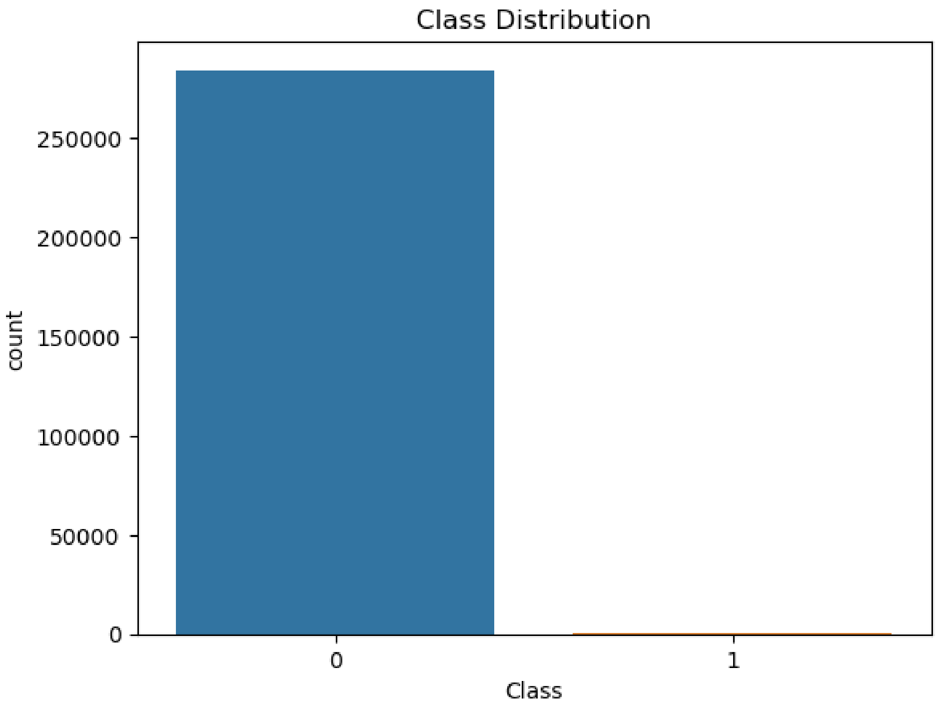
Dataset Information

In [3]:
```
data.info()
```

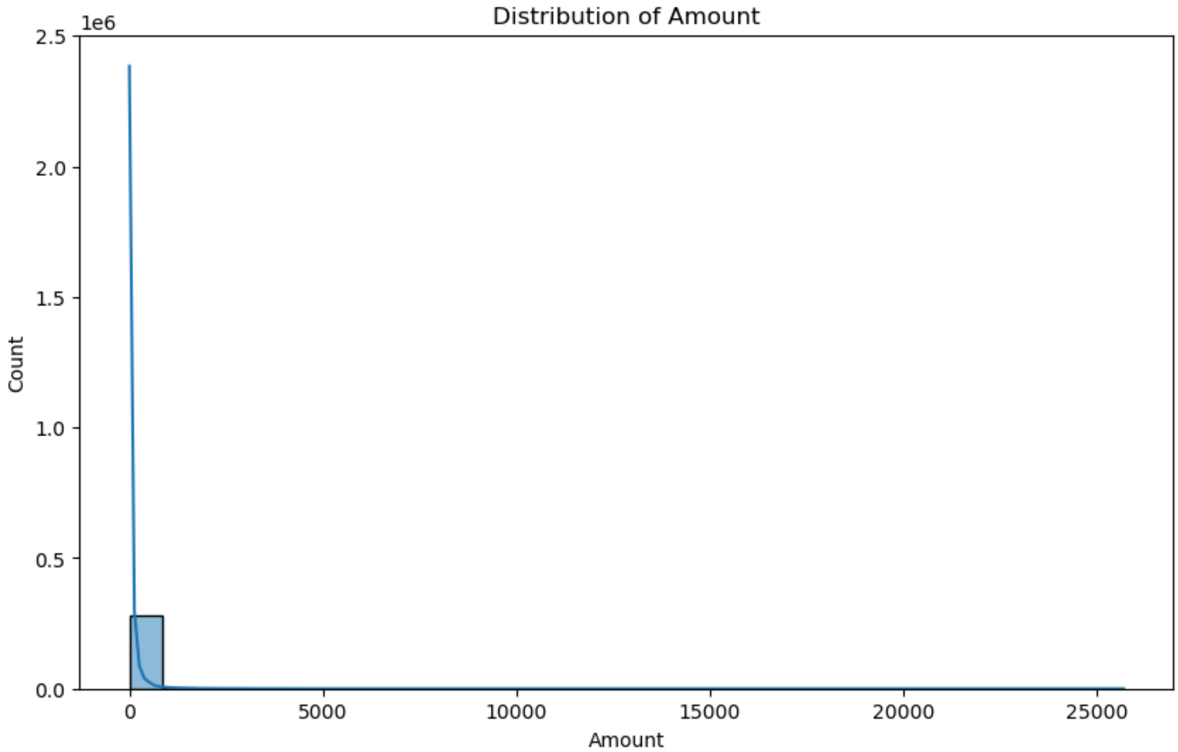
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Visualising Data

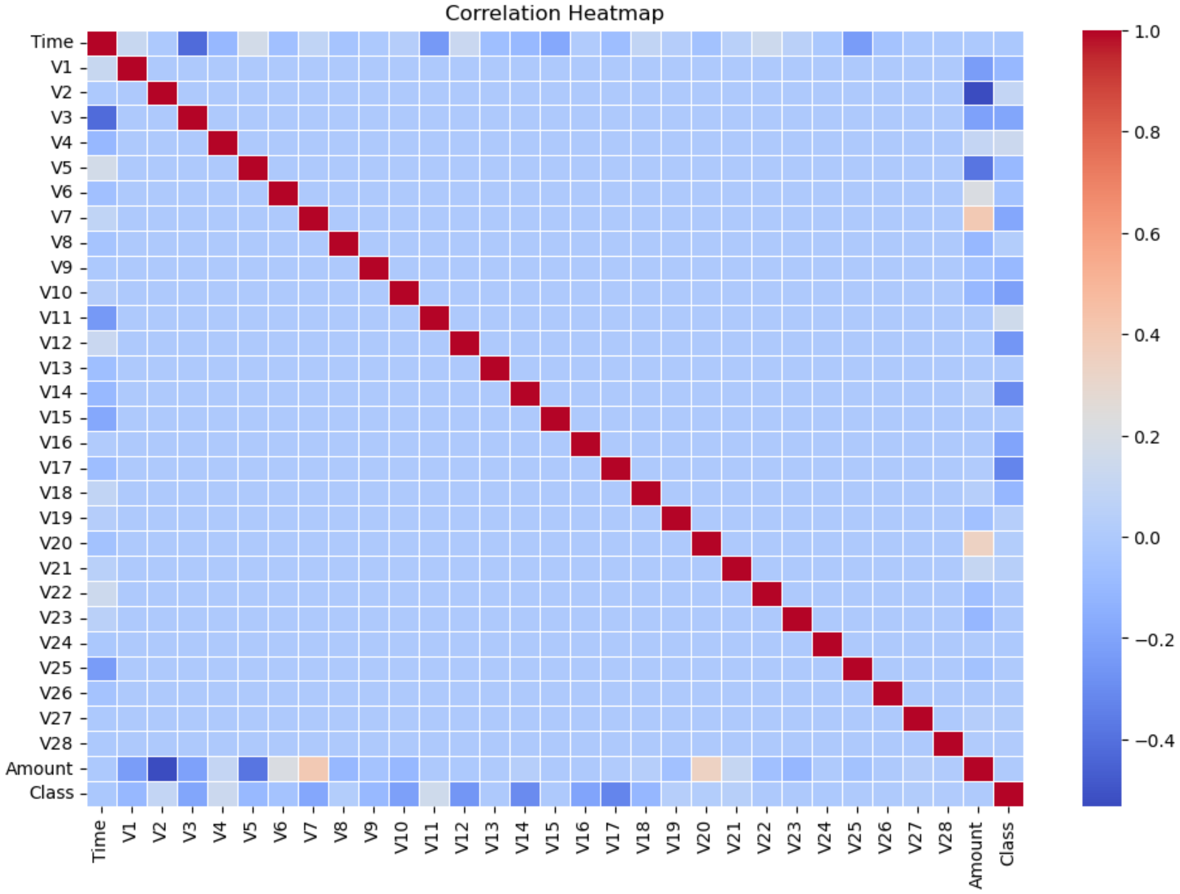
In [4]:
```
# Distribution of the 'Class' column (target variable)
sns.countplot(data=data, x='Class')
plt.title('Class Distribution')
plt.show()
```



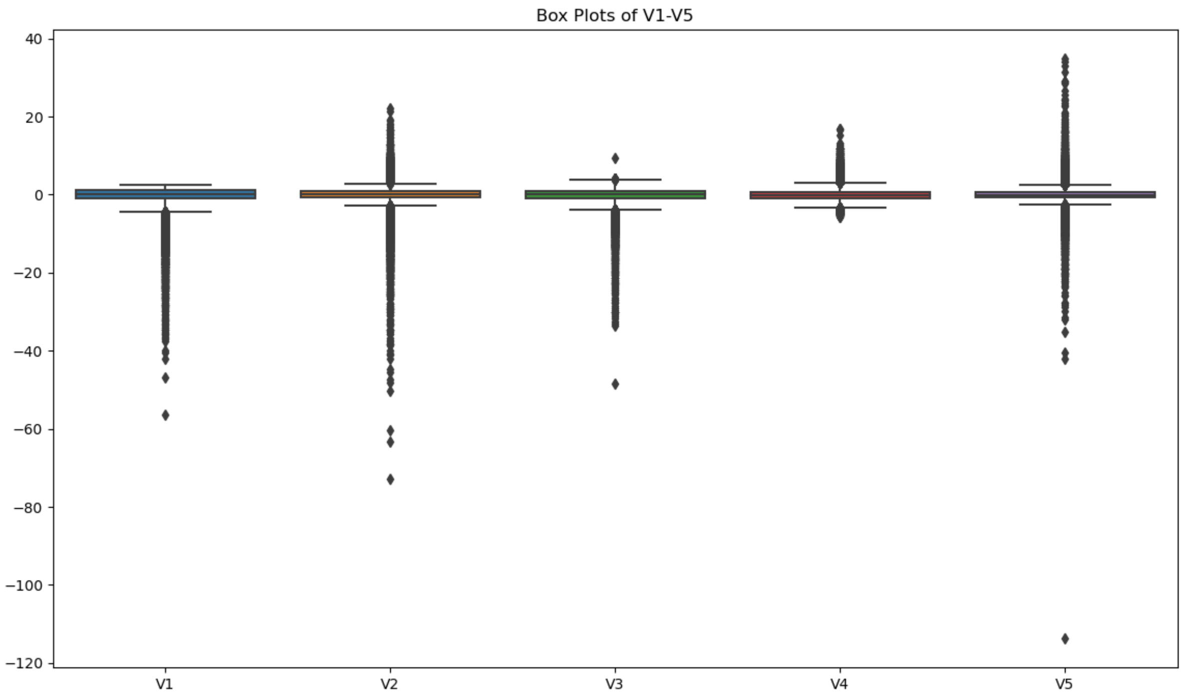
In [5]:
```
# Distribution of the 'Amount' column
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='Amount', bins=30, kde=True)
plt.title('Distribution of Amount')
plt.show()
```



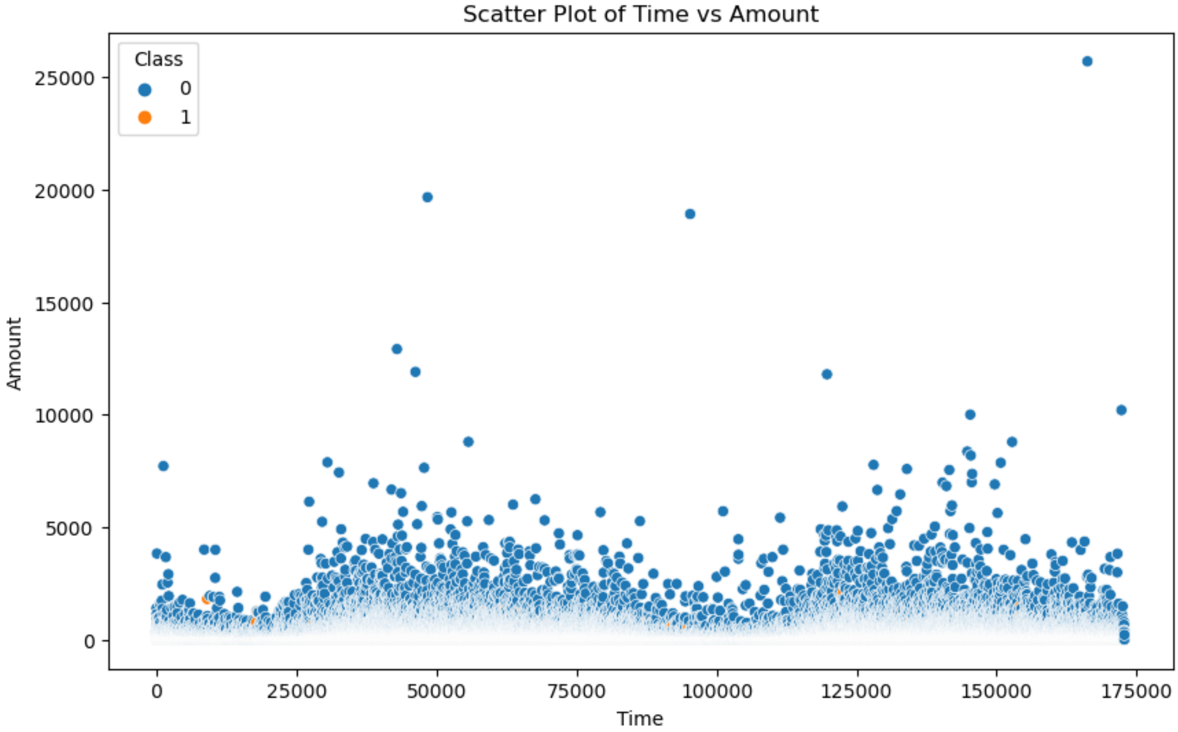
In [6]:
```
# Correlation heatmap of the numerical features
plt.figure(figsize=(12, 8))
corr_matrix = data.corr()
sns.heatmap(corr_matrix, cmap="coolwarm", annot=False, linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



In [7]:
```
# Box plots for some of the numerical features
plt.figure(figsize=(14, 8))
sns.boxplot(data=data[['V1', 'V2', 'V3', 'V4', 'V5']])
plt.title('Box Plots of V1-V5')
plt.show()
```



In [8]:
```
# Scatter plot for Time vs Amount
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='Time', y='Amount', hue='Class')
plt.title('Scatter Plot of Time vs Amount')
plt.show()
```



Training and Testing of Data

In [9]:
```
# Separate features and target
X = data.drop('Class', axis=1)
y = data['Class']
```

In [10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Logistic Regression Model

In [11]:
```
# Create a logistic regression model
logistic_reg = LogisticRegression(random_state=42)

# Fit the model on the training data
logistic_reg.fit(X_train, y_train)

# Make predictions on the test data
y_pred_lr = logistic_reg.predict(X_test)

# Evaluate the model
accuracy_lr = accuracy_score(y_test, y_pred_lr)
confusion_lr = confusion_matrix(y_test, y_pred_lr)
classification_rep_lr = classification_report(y_test, y_pred_lr)

print("Accuracy:", accuracy_lr)
print("Confusion Matrix:\n", confusion_lr)
print("Classification Report:\n", classification_rep_lr)
```

```
Accuracy: 0.9986482216214319
Confusion Matrix:
 [[56829    35]
 [   42    56]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.62      0.57      0.59        98

    accuracy                           1.00     56962
   macro avg       0.81      0.79      0.80     56962
weighted avg       1.00      1.00      1.00     56962
```

Random Forest Model

In [12]:
```
# Create a Random Forest classifier
random_forest = RandomForestClassifier(random_state=42)

# Fit the model on the training data
random_forest.fit(X_train, y_train)

# Make predictions on the test data
y_pred_rf = random_forest.predict(X_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
confusion_rf = confusion_matrix(y_test, y_pred_rf)
classification_rep_rf = classification_report(y_test, y_pred_rf)

print("Accuracy:", accuracy_rf)
print("Confusion Matrix:\n", confusion_rf)
print("Classification Report:\n", classification_rep_rf)
```

```
Accuracy: 0.9995611109160493
Confusion Matrix:
 [[56862     2]
 [   23    75]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.97      0.77      0.86        98

    accuracy                           1.00     56962
   macro avg       0.99      0.88      0.93     56962
weighted avg       1.00      1.00      1.00     56962
```

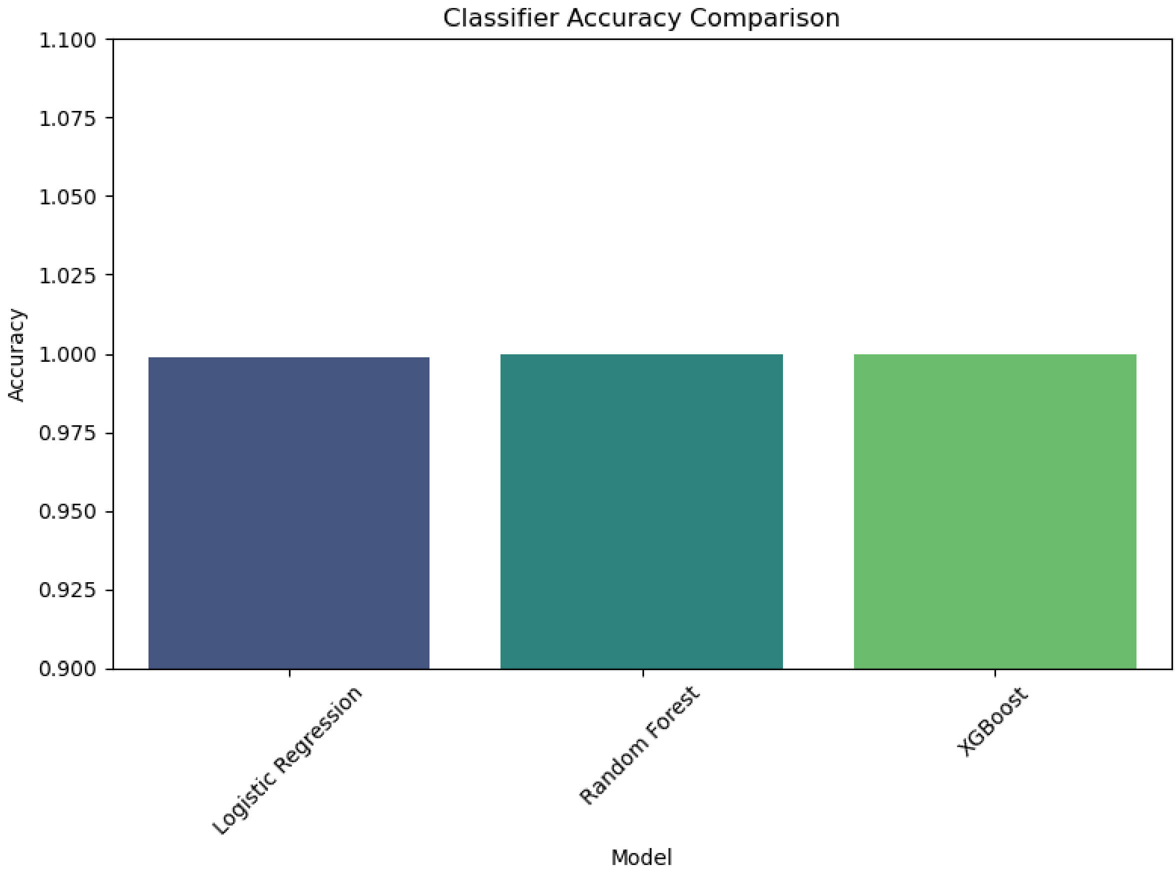
XG Boost Classifier

In [13]:
```
# Create an XGBoost classifier
xgb_classifier = XGBClassifier(random_state=42)

# Fit the model on the training data
xgb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred_xgb = xgb_classifier.predict(X_test)

# Evaluate the model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
confusion_xgb = confusion_matrix(y_test, y_pred_xgb)
classification_rep_xgb = classification_report(y_test, y_pred_xgb)

print("Accuracy:", accuracy_xgb)
print("Confusion Matrix:\n", confusion_xgb)
print("Classification Report:\n", classification_rep_xgb)
```

```
Accuracy: 0.9996137776061234
Confusion Matrix:
 [[56863     1]
 [   21    77]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.99      0.79      0.88        98

    accuracy                           1.00     56962
   macro avg       0.99      0.89      0.94     56962
weighted avg       1.00      1.00      1.00     56962
```

Result Comparison

In [14]:
```
# Create a DataFrame to store the results
results = pd.DataFrame({
    'Model': ['Logistic Regression', 'Random Forest', 'XGBoost'],
    'Accuracy': [accuracy_lr, accuracy_rf, accuracy_xgb]
})

# Create a bar plot to compare accuracies
plt.figure(figsize=(8, 6))
sns.barplot(data=results, x='Model', y='Accuracy', palette='viridis')
plt.title('Classifier Accuracy Comparison')
plt.ylim(0.9, 1.1)  # Adjust the y-axis limits for better visualization
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [15]:
```python
# Calculate precision, recall, and F1 Score for each model
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)

precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

precision_xgb = precision_score(y_test, y_pred_xgb)
recall_xgb = recall_score(y_test, y_pred_xgb)
f1_xgb = f1_score(y_test, y_pred_xgb)

# Create a DataFrame to store the metrics
metrics_df = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
    'Logistic Regression': [accuracy_lr, precision_lr, recall_lr, f1_lr],
    'Random Forest': [accuracy_rf, precision_rf, recall_rf, f1_rf],
    'XGBoost': [accuracy_xgb, precision_xgb, recall_xgb, f1_xgb]
})

# Set 'Metric' column as the index
metrics_df = metrics_df.set_index('Metric')

# Print the metrics table
print(metrics_df)
```

```
           Logistic Regression  Random Forest   XGBoost
Metric
Accuracy              0.998648       0.999561  0.999614
Precision             0.615385       0.974026  0.987179
Recall                0.571429       0.765306  0.785714
F1 Score              0.592593       0.857143  0.875000
```

Identifying the fraud

In [16]:
```
# Detect fraud using the trained model
fraud_predictions = random_forest.predict(X)  # Use the entire dataset for predictions

# Add the predictions to the original dataset
data['Fraud_Predictions'] = fraud_predictions

# You can now analyze the 'Fraud_Predictions' column to detect fraud cases
fraudulent_transactions = data[data['Fraud_Predictions'] == 1]
print("Detected fraud cases:\n", fraudulent_transactions)
```

```
Detected fraud cases:
             Time        V1        V2        V3        V4        V5        V6  \
472        347.0 -1.531271  1.399621 -0.587061  2.175002 -2.137637 -0.501576
541        406.0 -2.312227  1.951992 -1.609851  3.997906 -0.522188 -1.426545
623        472.0 -3.043541 -3.157307  1.088463  2.288644  1.359805 -1.064823
4920      4462.0 -2.303350  1.759247 -0.359745  2.330243 -0.821628 -0.075788
6108      6986.0 -4.397974  1.358367 -2.592844  2.679787 -1.128131 -1.706536
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V22       V23       V24  \
472    -1.215215  0.956862 -1.866561  ...  0.085267  0.403096  0.454438
541    -2.537387  1.391657 -2.770089  ... -0.035049 -0.465211  0.320198
623     0.325574 -0.067794 -0.270953  ...  0.435477  1.375966 -0.293803
4920    0.562320 -0.399147 -0.238253  ... -0.932391  0.172726 -0.087330
6108   -3.496197 -0.248778 -0.247768  ...  0.176968 -0.436207 -0.053502
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ... -0.319189  0.639419 -0.294885
280143 -1.413170  0.248525 -1.127396  ...  0.028234 -0.145640 -0.081049
280149 -2.234739  1.210158 -0.652250  ...  0.834108  0.190944  0.032070
281144 -2.208002  1.058733 -1.632333  ... -0.269209 -0.456108 -0.183659
281674  0.223050 -0.068384  0.577829  ... -0.295135 -0.072173 -0.450261

             V25       V26       V27       V28  Amount  Class  \
472     0.202522 -0.313118  0.527182  0.202575  204.03      0
541     0.044519  0.177840  0.261145 -0.143276    0.00      1
623     0.279798 -0.145362 -0.252773  0.035764  529.00      1
4920   -0.156114 -0.542628  0.039566 -0.153029  239.93      1
6108    0.252405 -0.657488 -0.827136  0.849573   59.00      1
...          ...       ...       ...       ...     ...    ...
279863  0.537503  0.788395  0.292680  0.147968  390.00      1
280143  0.521875  0.739467  0.389152  0.186637    0.76      1
280149 -0.739695  0.471111  0.385107  0.194361   77.89      1
281144 -0.328168  0.606116  0.884876 -0.253700  245.00      1
281674  0.313267 -0.289617  0.002988 -0.015309   42.53      1

        Fraud_Predictions
472                     1
541                     1
623                     1
4920                    1
6108                    1
...                   ...
279863                  1
280143                  1
280149                  1
281144                  1
281674                  1

[471 rows x 32 columns]
```

In [ ]: