Credit Card Fraud Detection

- Anonymized credit card transactions labeled as fraudulent or genuine

In [115]:

```
from IPython.display import Image
Image("G:/ML portfolio projects/Own Projects/Credit Card Fraud Detection//1.jpg")
```

Out[115]:

Load the Dataset

In [1]:

```
import pandas as pd
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[1]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [109]:

```
Image("G:/ML portfolio projects/Own Projects/Credit Card Fraud Detection//2.jpg")
```

Out[109]:

Exploratory Data Analysis (EDA)

In [2]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
df.describe()
```

Out[2]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e·
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e·
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e·
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e·

8 rows × 31 columns

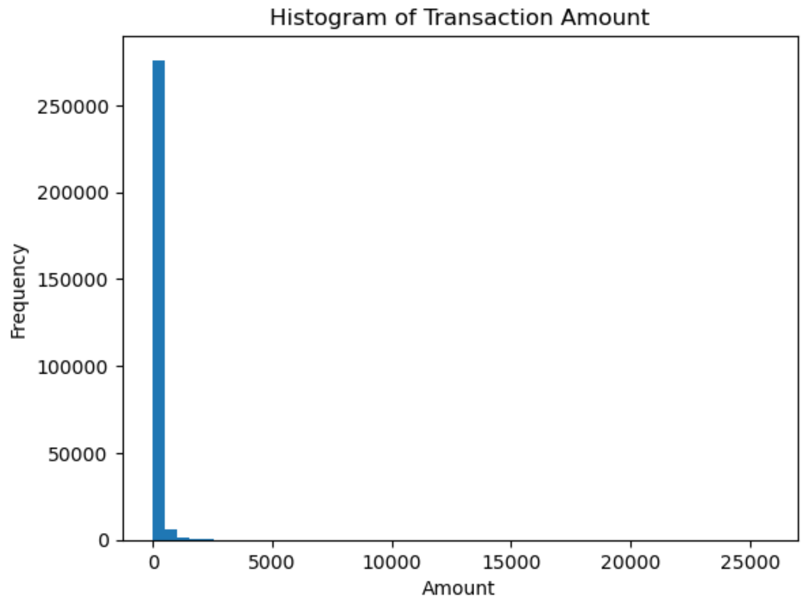
In [3]:

```python
class_counts = df['Class'].value_counts()
print(class_counts)
```

```
0    284315
1       492
Name: Class, dtype: int64
```

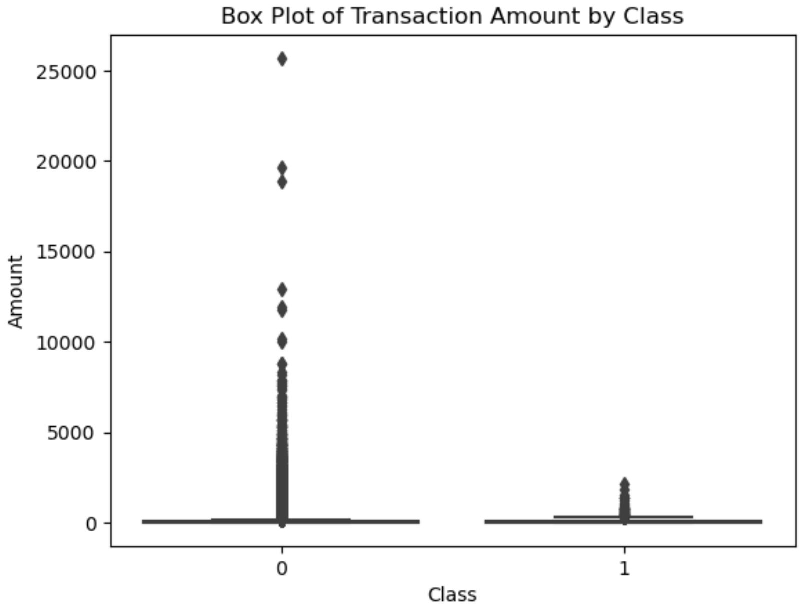
In [4]:

```python
plt.hist(df['Amount'], bins=50)
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.title('Histogram of Transaction Amount')
plt.show()
```



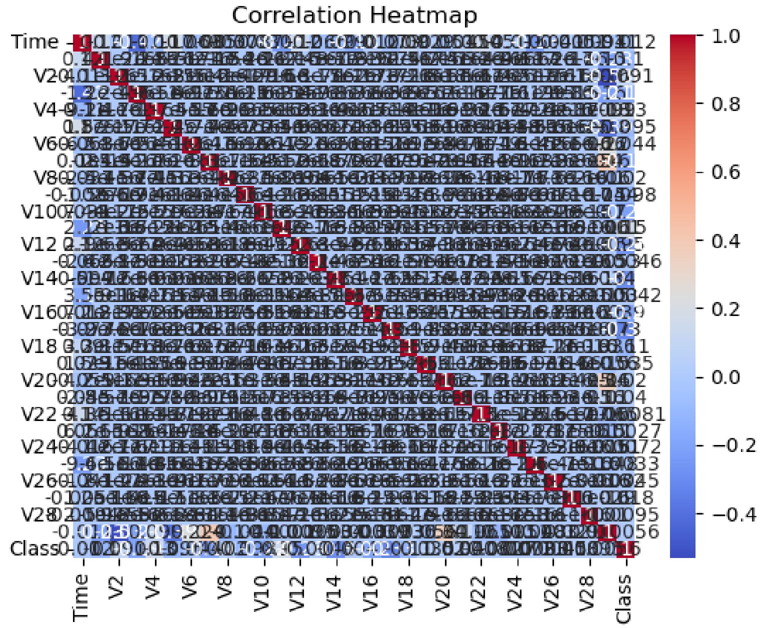
In [5]:

```
sns.boxplot(x='Class', y='Amount', data=df)
plt.xlabel('Class')
plt.ylabel('Amount')
plt.title('Box Plot of Transaction Amount by Class')
plt.show()
```



In [6]:

```
corr = df.corr()
sns.heatmap(corr, cmap='coolwarm', annot=True)
plt.title('Correlation Heatmap')
plt.show()
```



Data Preprocessing

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
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
```

In [8]:

```
Q1 = df['Amount'].quantile(0.25)
Q3 = df['Amount'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

df = df[(df['Amount'] >= lower_bound) & (df['Amount'] <= upper_bound)]
```

In [9]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df[['Amount', 'Time']] = scaler.fit_transform(df[['Amount', 'Time']])
```

In [110]:

```
Image("G:/ML portfolio projects/Own Projects/Credit Card Fraud Detection//3.png")
```

Out[110]:

Data Splitting

In [10]:

```
from sklearn.model_selection import train_test_split
```

In [11]:

```
X = df.drop('Class', axis=1)
y = df['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Handling Class Imbalance

- Oversampling with SMOTE
- Ensemble Methods using Random Forest

In [12]:

```
!pip install imbalanced-learn

from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
Requirement already satisfied: imbalanced-learn in c:\users\somnath\anacon
da3\lib\site-packages (0.11.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\somnath\an
aconda3\lib\site-packages (from imbalanced-learn) (3.1.0)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\somnath\ana
conda3\lib\site-packages (from imbalanced-learn) (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\somnath\anaconda3
\lib\site-packages (from imbalanced-learn) (1.24.3)
Requirement already satisfied: joblib>=1.1.1 in c:\users\somnath\anaconda3
\lib\site-packages (from imbalanced-learn) (1.3.1)
Requirement already satisfied: scipy>=1.5.0 in c:\users\somnath\anaconda3
\lib\site-packages (from imbalanced-learn) (1.9.1)
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

## One alternative strategy among many other alternatives.
```

In [111]:

```
Image("G:/ML portfolio projects/Own Projects/Credit Card Fraud Detection//4.png")
```

Out[111]:

Model Selection

Logistic Regression

In [13]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score
```

In [14]:

```
logistic_regression = LogisticRegression(random_state=42)
logistic_regression.fit(X_train_resampled, y_train_resampled)
```

Out[14]:

```
▾        LogisticRegression
LogisticRegression(random_state=42)
```

In [64]:

```
y_pred_lr = logistic_regression.predict(X_test)
```

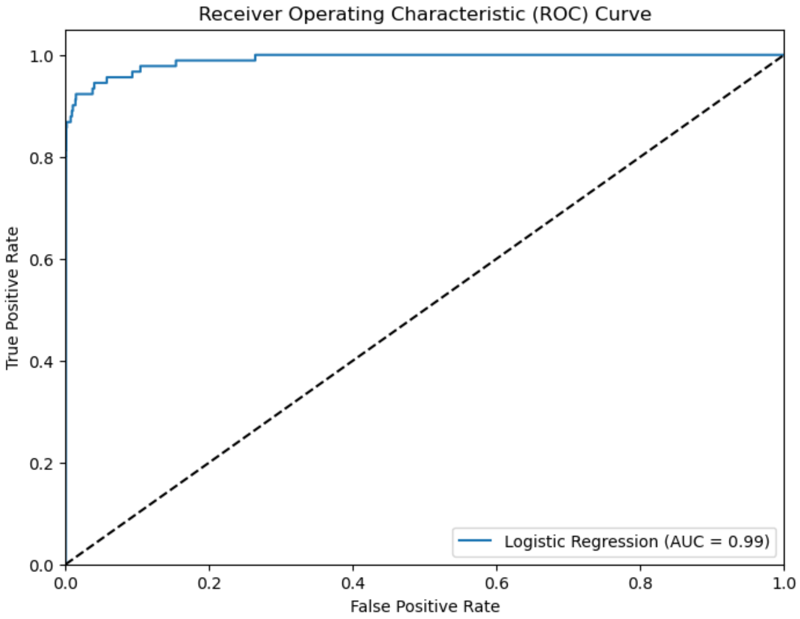
In [65]:

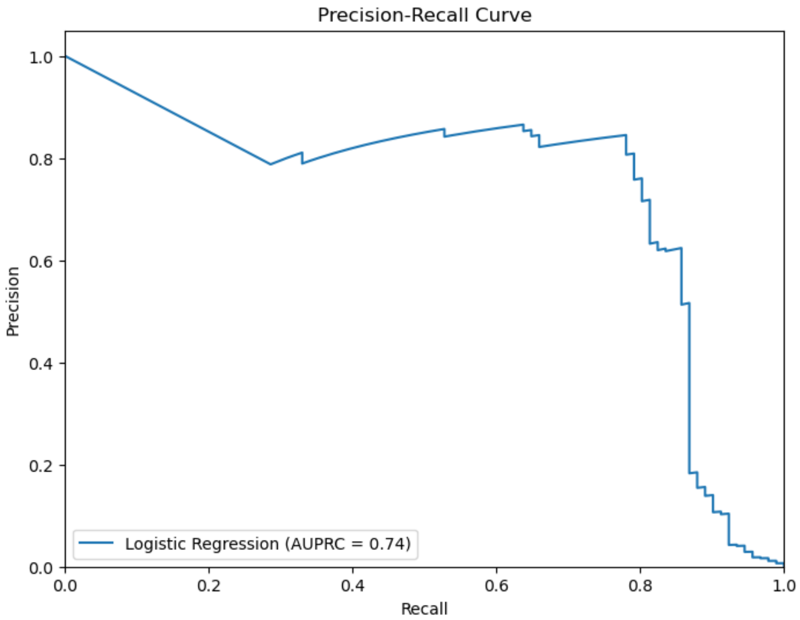
```
print(classification_report(y_test, y_pred_lr))
print("AUC: ", roc_auc_score(y_test,y_pred_lr))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     50490
           1       0.08      0.92      0.15        91

    accuracy                           0.98     50581
   macro avg       0.54      0.95      0.57     50581
weighted avg       1.00      0.98      0.99     50581

AUC:  0.9522494934259641
```

In [91]:

```
from sklearn.metrics import roc_curve, precision_recall_curve, auc
import matplotlib.pyplot as plt

# Get the predicted probabilities for the positive class (fraud) from the logistic regres
y_pred_prob_lr = logistic_regression.predict_proba(X_test)[:, 1]

# Compute the false positive rate, true positive rate, and threshold for the ROC curve
fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_prob_lr)

# Compute the precision, recall, and threshold for the Precision-Recall curve
precision, recall, thresholds_pr = precision_recall_curve(y_test, y_pred_prob_lr)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Compute the area under the Precision-Recall curve
pr_auc = auc(recall, precision)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='Logistic Regression (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Plot the Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label='Logistic Regression (AUPRC = %0.2f)' % pr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```





Decision Tree

In [17]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [18]:

```
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train_resampled, y_train_resampled)
```

Out[18]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [66]:

```
y_pred_dt = decision_tree.predict(X_test)
```

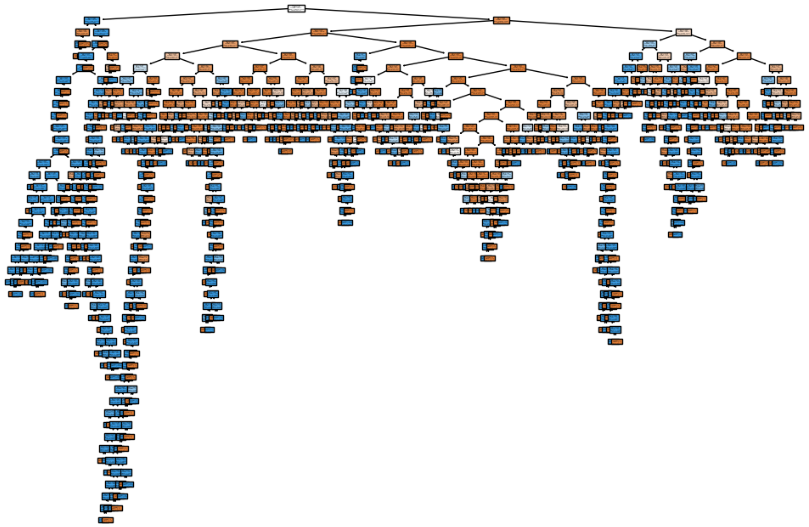
In [67]:

```
print(classification_report(y_test, y_pred_dt))
print("AUC: ", roc_auc_score(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     50490
           1       0.48      0.78      0.59        91

    accuracy                           1.00     50581
   macro avg       0.74      0.89      0.80     50581
weighted avg       1.00      1.00      1.00     50581

AUC:  0.8893473628767746
```

In [93]:

```
from sklearn.tree import plot_tree

# Get the list of feature names
feature_names = X.columns.tolist()

# Visualize the Decision Tree model
plt.figure(figsize=(12, 8))
plot_tree(decision_tree, feature_names=feature_names, class_names=['Genuine', 'Fraud'], f
plt.show()
```



Random Forest

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [22]:

```
random_forest = RandomForestClassifier(random_state=42)
random_forest.fit(X_train_resampled, y_train_resampled)
```

Out[22]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [68]:

```
y_pred_rf = random_forest.predict(X_test)
```

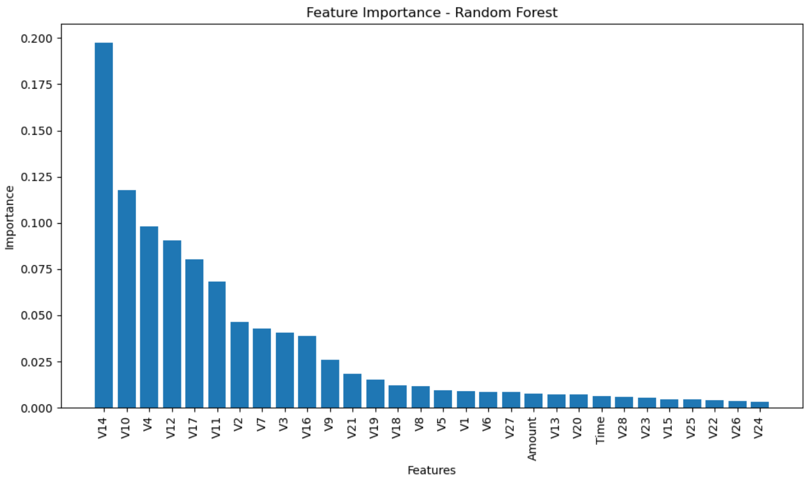
In [69]:

```
print(classification_report(y_test, y_pred_rf))
print("AUC: ", roc_auc_score(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     50490
           1       0.91      0.85      0.87        91

    accuracy                           1.00     50581
   macro avg       0.95      0.92      0.94     50581
weighted avg       1.00      1.00      1.00     50581

AUC:  0.9229976994682878
```

In [94]:

```
import matplotlib.pyplot as plt

# Get feature importances from the Random Forest model
feature_importances = random_forest.feature_importances_

# Sort feature importances in descending order
sorted_indices = feature_importances.argsort()[::-1]
sorted_importances = feature_importances[sorted_indices]

# Get the names of the sorted features
sorted_feature_names = X_train.columns[sorted_indices]

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.bar(range(len(feature_importances)), sorted_importances)
plt.xticks(range(len(feature_importances)), sorted_feature_names, rotation=90)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance - Random Forest')
plt.tight_layout()
plt.show()
```



Gradient Boosting Machines (GBM)

In [25]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [26]:

```
gbm = GradientBoostingClassifier(random_state=42)
gbm.fit(X_train_resampled, y_train_resampled)
```

Out[26]:

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

In [70]:

```
y_pred_gbm = gbm.predict(X_test)
```

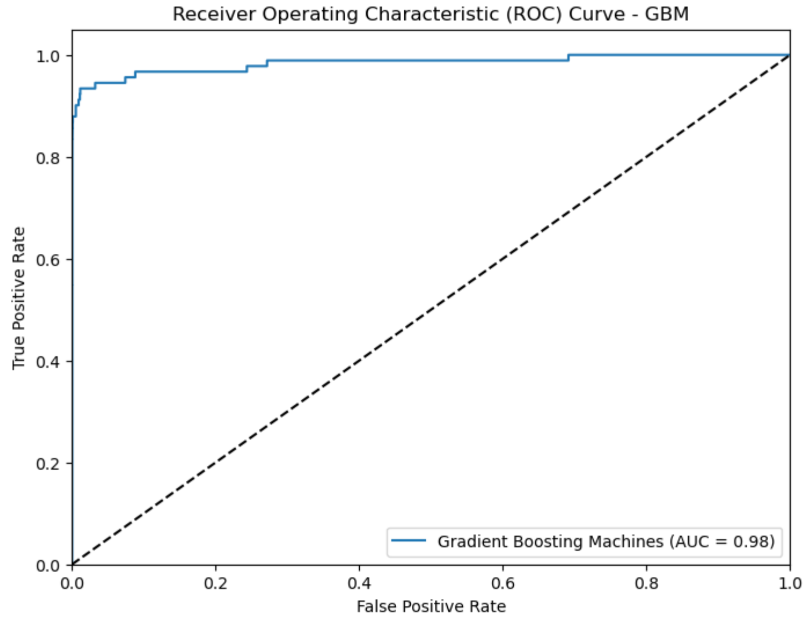
In [71]:

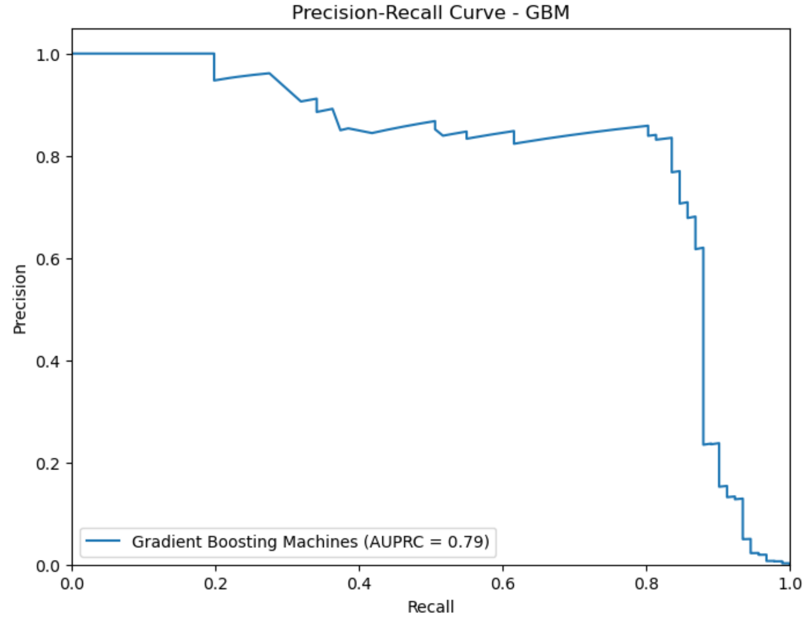
```
print(classification_report(y_test, y_pred_gbm))
print("AUC: ", roc_auc_score(y_test, y_pred_gbm))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     50490
           1       0.18      0.90      0.30        91

    accuracy                           0.99     50581
   macro avg       0.59      0.95      0.65     50581
weighted avg       1.00      0.99      0.99     50581

AUC:  0.9468160379925086
```

In [96]:

```
from sklearn.metrics import roc_curve, precision_recall_curve, auc
import matplotlib.pyplot as plt

# Get the predicted probabilities for the positive class (fraud)
y_pred_prob_gbm = gbm.predict_proba(X_test)[:, 1]

# Compute the false positive rate, true positive rate, and threshold for the ROC curve
fpr_gbm, tpr_gbm, thresholds_roc_gbm = roc_curve(y_test, y_pred_prob_gbm)

# Compute the precision, recall, and threshold for the Precision-Recall curve
precision_gbm, recall_gbm, thresholds_pr_gbm = precision_recall_curve(y_test, y_pred_prob

# Compute the area under the ROC curve
roc_auc_gbm = auc(fpr_gbm, tpr_gbm)

# Compute the area under the Precision-Recall curve
pr_auc_gbm = auc(recall_gbm, precision_gbm)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_gbm, tpr_gbm, label='Gradient Boosting Machines (AUC = %0.2f)' % roc_auc_gbm
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve - GBM')
plt.legend(loc="lower right")
plt.show()

# Plot the Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall_gbm, precision_gbm, label='Gradient Boosting Machines (AUPRC = %0.2f)' %
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - GBM')
plt.legend(loc="lower left")
plt.show()
```





Support Vector Machines (SVM)

In [29]:

```
from sklearn.svm import SVC
```

In [30]:

```
svm = SVC(random_state=42)
svm.fit(X_train_resampled, y_train_resampled)
```

Out[30]:

```
▾ SVC
SVC(random_state=42)
```

In [72]:

```
y_pred_svm = svm.predict(X_test)
```

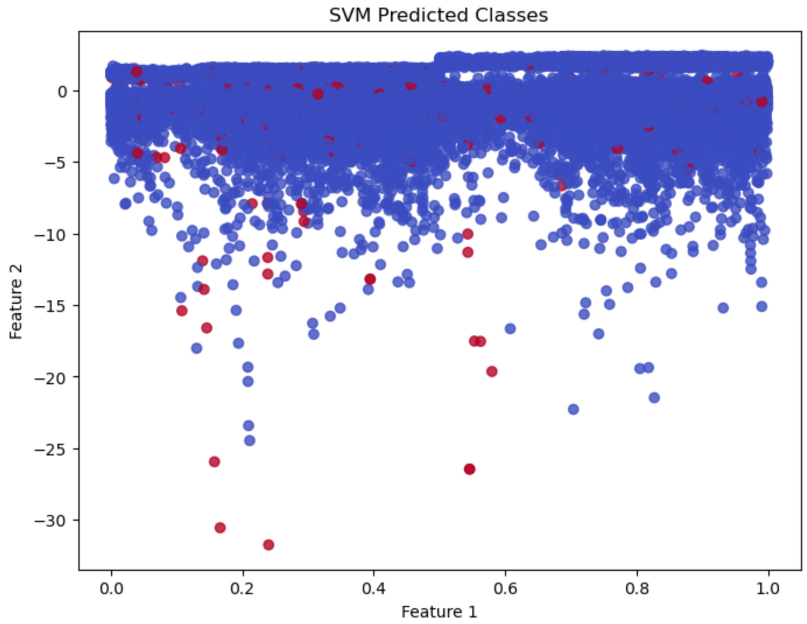
In [73]:

```
print(classification_report(y_test, y_pred_svm))
print("AUC: ", roc_auc_score(y_test, y_pred_svm))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     50490
           1       0.12      0.91      0.22        91

    accuracy                           0.99     50581
   macro avg       0.56      0.95      0.61     50581
weighted avg       1.00      0.99      0.99     50581

AUC:  0.950270535564653
```

In [100]:

```
import numpy as np
import matplotlib.pyplot as plt

# Extract the two features for visualization
X_vis = X_test.iloc[:, :2].values

# Make predictions on the test data using the SVM model
y_pred_svm = svm.predict(X_test)

# Create a scatter plot of the predicted classes
plt.figure(figsize=(8, 6))
plt.scatter(X_vis[:, 0], X_vis[:, 1], c=y_pred_svm, cmap='coolwarm', alpha=0.8)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('SVM Predicted Classes')
plt.show()
```



Artificial Neural Networks (ANNs)

In [33]:

```
from tensorflow import keras
from tensorflow.keras import layers
```

In [34]:

```
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train_resampled.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_train_resampled, y_train_resampled, epochs=10, batch_size=16)
```

```
Epoch 1/10
25252/25252 [==============================] - 43s 2ms/step - loss: 0.0111
- accuracy: 0.9969
Epoch 2/10
25252/25252 [==============================] - 37s 1ms/step - loss: 0.0039
- accuracy: 0.9992
Epoch 3/10
25252/25252 [==============================] - 34s 1ms/step - loss: 0.0029
- accuracy: 0.9994
Epoch 4/10
25252/25252 [==============================] - 39s 2ms/step - loss: 0.0024
- accuracy: 0.9995
Epoch 5/10
25252/25252 [==============================] - 40s 2ms/step - loss: 0.0021
- accuracy: 0.9996
Epoch 6/10
25252/25252 [==============================] - 36s 1ms/step - loss: 0.0021
- accuracy: 0.9996
Epoch 7/10
25252/25252 [==============================] - 39s 2ms/step - loss: 0.0018
- accuracy: 0.9996
Epoch 8/10
25252/25252 [==============================] - 34s 1ms/step - loss: 0.0016
- accuracy: 0.9997
Epoch 9/10
25252/25252 [==============================] - 40s 2ms/step - loss: 0.0016
- accuracy: 0.9997
Epoch 10/10
25252/25252 [==============================] - 35s 1ms/step - loss: 0.0016
- accuracy: 0.9997
```

Out[34]:

```
<keras.src.callbacks.History at 0x259a8c4adf0>
```

In [74]:

```
y_pred_ann = model.predict(X_test)
y_pred_ann = (y_pred_ann > 0.5).astype(int)
```

```
1581/1581 [==============================] - 5s 1ms/step
```

In [75]:

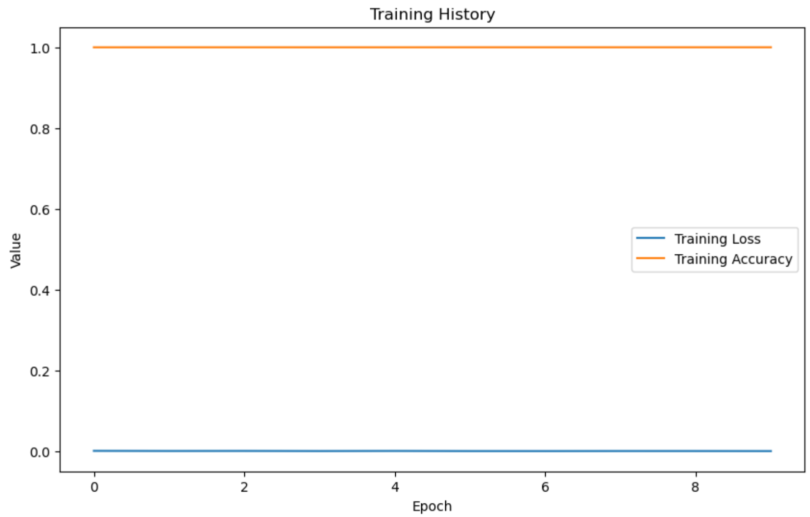
```
print(classification_report(y_test, y_pred))
print("AUC: ", roc_auc_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     50490
           1       0.08      0.92      0.15        91

    accuracy                           0.98     50581
   macro avg       0.54      0.95      0.57     50581
weighted avg       1.00      0.98      0.99     50581

AUC:  0.9522593963770435
```

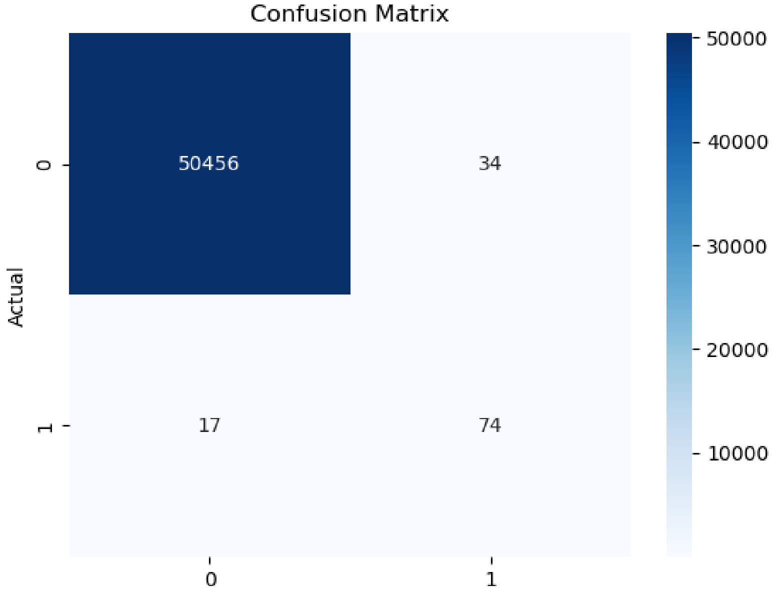
In [102]:

```
from tensorflow import keras
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Train the model and capture the history
history = model.fit(X_train_resampled, y_train_resampled, epochs=10, batch_size=16)

# Plot the training history
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Value')
plt.title('Training History')
plt.legend()
plt.show()

# Generate predictions
y_pred_ann = model.predict(X_test)
y_pred_ann = (y_pred_ann > 0.5).astype(int)

# Generate classification report and AUC score
print(classification_report(y_test, y_pred_ann))
print("AUC: ", roc_auc_score(y_test, y_pred_ann))

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred_ann)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Epoch 1/10
25252/25252 [==============================] - 35s 1ms/step - loss: 0.0017
- accuracy: 0.9997
Epoch 2/10
25252/25252 [==============================] - 34s 1ms/step - loss: 0.0014
- accuracy: 0.9998
Epoch 3/10
25252/25252 [==============================] - 39s 2ms/step - loss: 0.0015
- accuracy: 0.9998
Epoch 4/10
25252/25252 [==============================] - 34s 1ms/step - loss: 0.0012
- accuracy: 0.9997
Epoch 5/10
25252/25252 [==============================] - 37s 1ms/step - loss: 0.0015
- accuracy: 0.9997
Epoch 6/10
25252/25252 [==============================] - 34s 1ms/step - loss: 0.0011
- accuracy: 0.9998
Epoch 7/10
25252/25252 [==============================] - 32s 1ms/step - loss: 0.0011
- accuracy: 0.9998
Epoch 8/10
25252/25252 [==============================] - 35s 1ms/step - loss: 0.0013
- accuracy: 0.9998
Epoch 9/10
25252/25252 [==============================] - 37s 1ms/step - loss: 0.0012
- accuracy: 0.9998
Epoch 10/10
25252/25252 [==============================] - 39s 2ms/step - loss: 0.0011
- accuracy: 0.9998
```



```
1581/1581 [==============================] - 1s 895us/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     50490
           1       0.69      0.81      0.74        91

    accuracy                           1.00     50581
   macro avg       0.84      0.91      0.87     50581
weighted avg       1.00      1.00      1.00     50581

AUC:  0.9062567062567063
```




```python
from sklearn.ensemble import IsolationForest
```

In [39]:

```python
isolation_forest = IsolationForest(random_state=42)
isolation_forest.fit(X_train)
```

Out[39]:

```
▾ IsolationForest
IsolationForest(random_state=42)
```

In [78]:

```python
y_pred_IF = isolation_forest.predict(X_test)
y_pred_IF = (y_pred_IF == -1).astype(int)
```

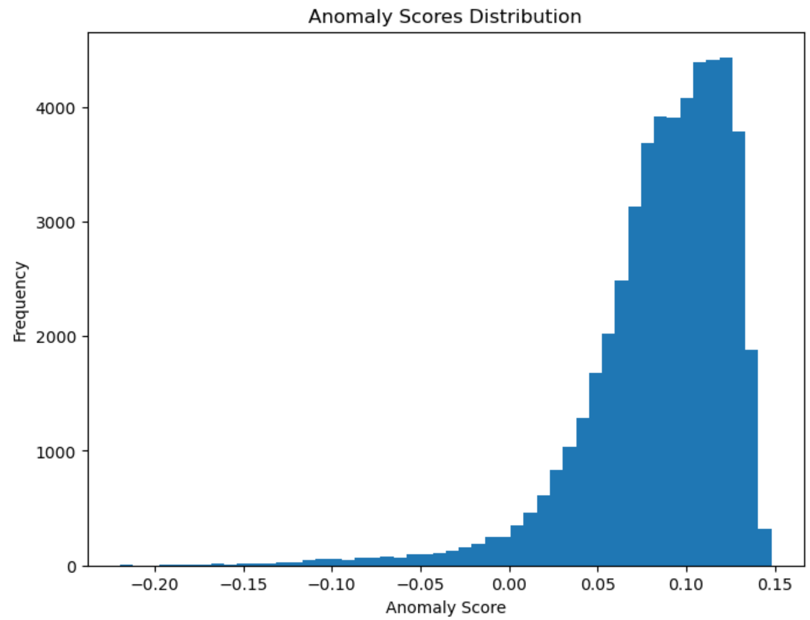
In [79]:

```python
print(classification_report(y_test, y_pred_IF))
print("AUC: ", roc_auc_score(y_test, y_pred_IF))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     50490
           1       0.04      0.79      0.07        91

    accuracy                           0.96     50581
   macro avg       0.52      0.88      0.53     50581
weighted avg       1.00      0.96      0.98     50581

AUC:  0.8778781131722307
```

In [103]:

```python
import matplotlib.pyplot as plt

# Get the anomaly scores for the test data using the Isolation Forest model
anomaly_scores = isolation_forest.decision_function(X_test)

# Plot the anomaly scores
plt.figure(figsize=(8, 6))
plt.hist(anomaly_scores, bins=50)
plt.xlabel('Anomaly Score')
plt.ylabel('Frequency')
plt.title('Anomaly Scores Distribution')
plt.show()
```



One-Class SVM

In [42]:

```
from sklearn.svm import OneClassSVM
```

In [43]:

```
one_class_svm = OneClassSVM()
one_class_svm.fit(X_train)
```

Out[43]:

```
▾ OneClassSVM
OneClassSVM()
```

In [76]:

```
y_pred_OCSVM = one_class_svm.predict(X_test)
y_pred_OCSVM = (y_pred_OCSVM == -1).astype(int)
```

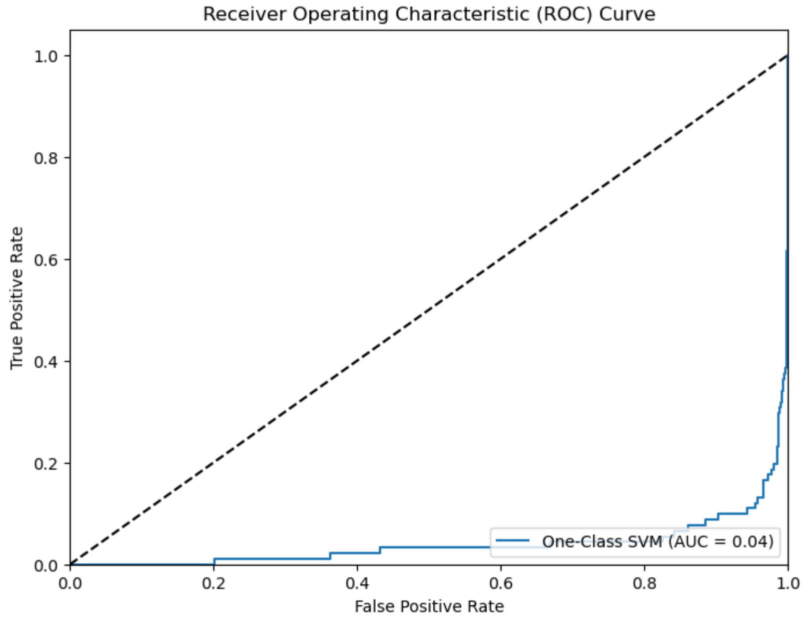
In [77]:

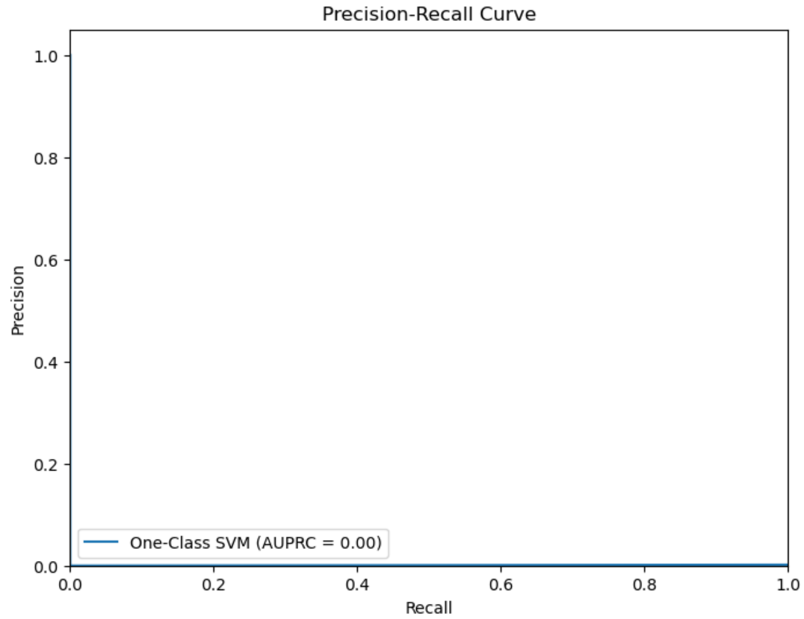
```
print(classification_report(y_test, y_pred_OCSVM))
print("AUC: ", roc_auc_score(y_test, y_pred_OCSVM))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67     50490
           1       0.00      0.97      0.01        91

    accuracy                           0.50     50581
   macro avg       0.50      0.73      0.34     50581
weighted avg       1.00      0.50      0.67     50581

AUC:  0.7337343484402309
```

In [104]:

```
from sklearn.metrics import roc_curve, precision_recall_curve, auc
import matplotlib.pyplot as plt

# Calculate the decision function scores for the test data
decision_scores = one_class_svm.decision_function(X_test)

# Compute the false positive rate, true positive rate, and threshold for the ROC curve
fpr, tpr, thresholds_roc = roc_curve(y_test, decision_scores)

# Compute the precision, recall, and threshold for the Precision-Recall curve
precision, recall, thresholds_pr = precision_recall_curve(y_test, decision_scores)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Compute the area under the Precision-Recall curve
pr_auc = auc(recall, precision)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label='One-Class SVM (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Plot the Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label='One-Class SVM (AUPRC = %0.2f)' % pr_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```





Model Comparison and Selection

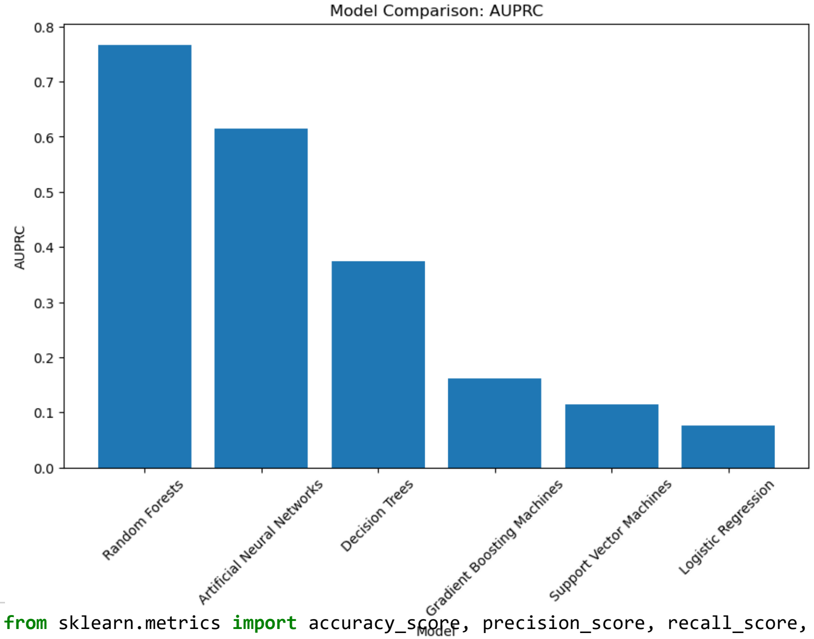
In [80]:

```python
import pandas as pd
from sklearn.metrics import precision_score, recall_score, f1_score, average_precision_sc
import matplotlib.pyplot as plt

# Create a DataFrame for model comparison
model_comparison = pd.DataFrame(columns=['Model', 'Precision', 'Recall', 'F1 Score', 'AUF

# Logistic Regression
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)
auprc_lr = average_precision_score(y_test, y_pred_lr)

model_comparison = model_comparison.append({
    'Model': 'Logistic Regression',
    'Precision': precision_lr,
    'Recall': recall_lr,
    'F1 Score': f1_lr,
    'AUPRC': auprc_lr
}, ignore_index=True)

# Decision Trees
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)
auprc_dt = average_precision_score(y_test, y_pred_dt)

model_comparison = model_comparison.append({
    'Model': 'Decision Trees',
    'Precision': precision_dt,
    'Recall': recall_dt,
    'F1 Score': f1_dt,
    'AUPRC': auprc_dt
}, ignore_index=True)

# Random Forests
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)
auprc_rf = average_precision_score(y_test, y_pred_rf)

model_comparison = model_comparison.append({
    'Model': 'Random Forests',
    'Precision': precision_rf,
    'Recall': recall_rf,
    'F1 Score': f1_rf,
    'AUPRC': auprc_rf
}, ignore_index=True)

# Gradient Boosting Machines (GBM)
precision_gbm = precision_score(y_test, y_pred_gbm)
recall_gbm = recall_score(y_test, y_pred_gbm)
f1_gbm = f1_score(y_test, y_pred_gbm)
auprc_gbm = average_precision_score(y_test, y_pred_gbm)

model_comparison = model_comparison.append({
    'Model': 'Gradient Boosting Machines',
    'Precision': precision_gbm,
    'Recall': recall_gbm,
    'F1 Score': f1_gbm,
    'AUPRC': auprc_gbm
}, ignore_index=True)

# Support Vector Machines (SVM)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
auprc_svm = average_precision_score(y_test, y_pred_svm)

model_comparison = model_comparison.append({
    'Model': 'Support Vector Machines',
    'Precision': precision_svm,
    'Recall': recall_svm,
    'F1 Score': f1_svm,
    'AUPRC': auprc_svm
}, ignore_index=True)

# Artificial Neural Networks (ANNs)
precision_ann = precision_score(y_test, y_pred_ann)
recall_ann = recall_score(y_test, y_pred_ann)
f1_ann = f1_score(y_test, y_pred_ann)
auprc_ann = average_precision_score(y_test, y_pred_ann)

model_comparison = model_comparison.append({
    'Model': 'Artificial Neural Networks',
    'Precision': precision_ann,
    'Recall': recall_ann,
    'F1 Score': f1_ann,
    'AUPRC': auprc_ann
}, ignore_index=True)

# Isolation Forest
precision_ann = precision_score(y_test, y_pred_IF)
recall_ann = recall_score(y_test, y_pred_IF)
f1_ann = f1_score(y_test, y_pred_IF)
auprc_ann = average_precision_score(y_test, y_pred_IF)

model_comparison = model_comparison.append({
    'Model': 'Artificial Neural Networks',
    'Precision': precision_ann,
    'Recall': recall_ann,
    'F1 Score': f1_ann,
    'AUPRC': auprc_ann
}, ignore_index=True)

# One-Class SVM
precision_ann = precision_score(y_test, y_pred_OCSVM)
recall_ann = recall_score(y_test, y_pred_OCSVM)
f1_ann = f1_score(y_test, y_pred_OCSVM)
auprc_ann = average_precision_score(y_test, y_pred_OCSVM)

model_comparison = model_comparison.append({
    'Model': 'Artificial Neural Networks',
    'Precision': precision_ann,
    'Recall': recall_ann,
    'F1 Score': f1_ann,
    'AUPRC': auprc_ann
}, ignore_index=True)
```

```python
# Sort the DataFrame by AUPRC
model_comparison = model_comparison.sort_values(by='AUPRC', ascending=False)

# Print the model comparison table
print(model_comparison)

# Bar plot of AUPRC scores
plt.figure(figsize=(10, 6))
plt.bar(model_comparison['Model'], model_comparison['AUPRC'])
plt.xlabel('Model')
plt.ylabel('AUPRC')
plt.title('Model Comparison: AUPRC')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:14: Futu
reWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:28: Futu
reWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:42: Futu
reWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:56: Futu
reWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:70: Futu
reWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:84: Futu
reWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:98: Futu
reWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
C:\Users\SOMNATH\AppData\Local\Temp\ipykernel_13788\3241063033.py:112: Fut
ureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  model_comparison = model_comparison.append({
```

	Model	Precision	Recall	F1 Score	AUPRC
2	Random Forests	0.905882	0.846154	0.875000	0.766793
5	Artificial Neural Networks	0.755102	0.813187	0.783069	0.614375
1	Decision Trees	0.479730	0.780220	0.594142	0.374690
3	Gradient Boosting Machines	0.178649	0.901099	0.298182	0.161159
4	Support Vector Machines	0.124625	0.912088	0.219287	0.113827
0	Logistic Regression	0.082192	0.923077	0.150943	0.076008
6	Artificial Neural Networks	0.038668	0.791209	0.073733	0.030970
7	Artificial Neural Networks	0.003477	0.967033	0.006929	0.003421




```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, aver

# Make predictions on the test data using the Random Forests model
y_pred_test_rf = random_forest.predict(X_test)

# Calculate evaluation metrics
accuracy_rf = accuracy_score(y_test, y_pred_test_rf)
precision_rf = precision_score(y_test, y_pred_test_rf)
recall_rf = recall_score(y_test, y_pred_test_rf)
f1_rf = f1_score(y_test, y_pred_test_rf)
auprc_rf = average_precision_score(y_test, y_pred_test_rf)

# Print the evaluation metrics
print("Random Forests Model Evaluation:")
print("Accuracy: {:.4f}".format(accuracy_rf))
print("Precision: {:.4f}".format(precision_rf))
print("Recall: {:.4f}".format(recall_rf))
print("F1 Score: {:.4f}".format(f1_rf))
print("AUPRC: {:.4f}".format(auprc_rf))
```

```
Random Forests Model Evaluation:
Accuracy: 0.9996
Precision: 0.9059
Recall: 0.8462
F1 Score: 0.8750
AUPRC: 0.7668
```

In [118]:

```
Image("G:/ML portfolio projects/Own Projects/Credit Card Fraud Detection//5.png")
```

Out[118]:

In [ ]:

```

```