Credit Card Fraud Detection

Step 1: Installing and importing required libraries

In [1]:

```
pip install pandas scikit-learn imbalanced-learn matplotlib seaborn
```

```
Requirement already satisfied: pandas in c:\users\kaurm\coding stuff\lib\site-pack
ages (1.4.2)
Requirement already satisfied: scikit-learn in c:\users\kaurm\coding stuff\lib\sit
e-packages (1.0.2)
Requirement already satisfied: imbalanced-learn in c:\users\kaurm\coding stuff\lib
\site-packages (0.12.4)
Requirement already satisfied: matplotlib in c:\users\kaurm\coding stuff\lib\site-
packages (3.5.1)
Requirement already satisfied: seaborn in c:\users\kaurm\coding stuff\lib\site-pac
kages (0.11.2)
Requirement already satisfied: numpy>=1.18.5 in c:\users\kaurm\coding stuff\lib\si
te-packages (from pandas) (1.22.4)
Requirement already satisfied: pytz>=2020.1 in c:\users\kaurm\coding stuff\lib\sit
e-packages (from pandas) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\kaurm\coding stu
ff\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\kaurm\coding stuff
\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\kaurm\coding stuff\lib\sit
e-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: scipy>=1.1.0 in c:\users\kaurm\coding stuff\lib\sit
e-packages (from scikit-learn) (1.7.3)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\kaurm\coding stuff\lib
\site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\kaurm\coding stuff\li
b\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: cycler>=0.10 in c:\users\kaurm\coding stuff\lib\sit
e-packages (from matplotlib) (0.11.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\kaurm\coding stuff\lib\si
te-packages (from matplotlib) (9.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\kaurm\coding stuff\li
b\site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: packaging>=20.0 in c:\users\kaurm\coding stuff\lib
\site-packages (from matplotlib) (21.3)
Requirement already satisfied: six>=1.5 in c:\users\kaurm\coding stuff\lib\site-pa
ckages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Step 2: Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE #for dealing with class imbalance, Mr. GPT
import matplotlib.pyplot as plt
import seaborn as sns
```

Step 3: Load Dataset

Dataset used: Credit Card Fraud Detection from Kaggle

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=download (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=download)

In [2]:

```
# Load dataset
df = pd.read_csv('creditcard.csv')

# Display first few rows of the dataset
print(df.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]
```

Step 4: Data Preprocessing

4.1 Handling Missing Values

In [3]:

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

```
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
```

4.2 Scale Features

In [5]:
```
#Scale the 'Amount' feature
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])

#check first few rows after scaling
print(df.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  0.244964      0
1  0.125895 -0.008983  0.014724 -0.342475      0
2 -0.139097 -0.055353 -0.059752  1.160686      0
3 -0.221929  0.062723  0.061458  0.140534      0
4  0.502292  0.219422  0.215153 -0.073403      0

[5 rows x 31 columns]
```

4.3 Handle Class Imbalance

In [6]:
```
# Split data into features (X) and target (y)
X = df.drop('Class', axis=1)  # All columns except the target column 'Class'
y = df['Class']  # Target column

# Split the data into training and testing sets (70% training, 30% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Apply SMOTE to handle class imbalance
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Verify the class distribution after applying SMOTE
print("Class distribution after SMOTE: ")
print(pd.Series(y_train_res).value_counts())
```

```
Class distribution after SMOTE:
0    199008
1    199008
Name: Class, dtype: int64
```

Step 5: Train the Model

In this step, we are also evaluating each model by using the classification_report to show metrics like

1. Precision: How many selected items are relevant (how many fraudulent transactions we flagged are actually fraudulent).

2. Recall: How many relevant items are selected (how many actual fraudulent transactions we identified).

3. F1-Score: Harmonic mean of precision and recall.

4. Accuracy: Overall correctness, but might not be the best metric for imbalanced datasets.

5.1 Logistic Regression (Linear Model)

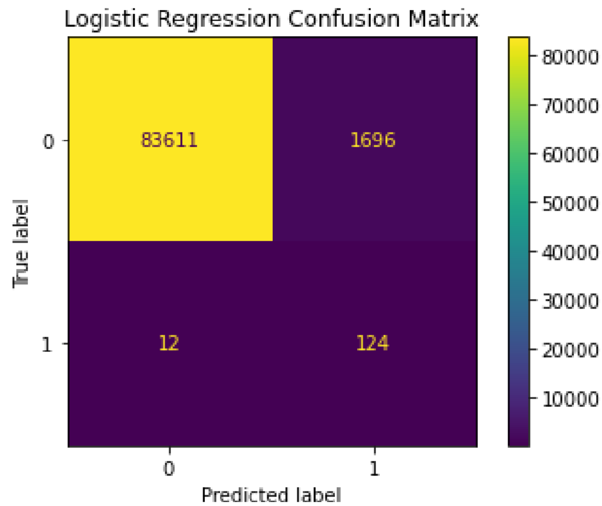
In [7]:

```
# Initialize Logistic Regression model
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train_res, y_train_res)

# Predict using the Logistic Regression model
lr_preds = lr_model.predict(X_test)

# Evaluate Logistic Regression model performance
print("Logistic Regression Performance:")
print(classification_report(y_test, lr_preds))
```

```
Logistic Regression Performance:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     85307
           1       0.07      0.91      0.13       136

    accuracy                           0.98     85443
   macro avg       0.53      0.95      0.56     85443
weighted avg       1.00      0.98      0.99     85443
```

In [8]:
```python
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix for Logistic Regression
plt.figure(figsize=(10, 8))  # Optional: adjust the figure size
ConfusionMatrixDisplay.from_estimator(lr_model, X_test, y_test)
plt.title('Logistic Regression Confusion Matrix')
plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```



5.2 Random Forest Classifier (Non-linear Model)

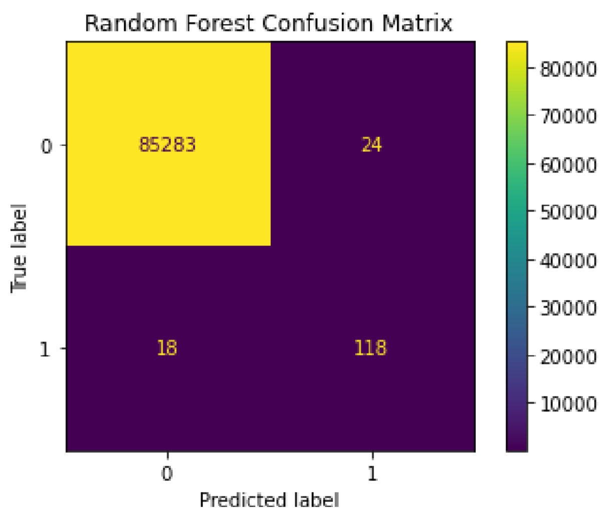
In [9]:
```python
# Initialize Random Forest model
rf_model = RandomForestClassifier(n_estimators=50, random_state=42, n_jobs=-1)
rf_model.fit(X_train_res, y_train_res)

# Predict using the Random Forest model
rf_preds = rf_model.predict(X_test)

# Evaluate Random Forest model performance
print("Random Forest Performance:")
print(classification_report(y_test, rf_preds))
```

```
Random Forest Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.83      0.87      0.85       136

    accuracy                           1.00     85443
   macro avg       0.92      0.93      0.92     85443
weighted avg       1.00      1.00      1.00     85443
```

In [10]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix for Random Forest
plt.figure(figsize=(10, 8))  # Optional: adjust the figure size
ConfusionMatrixDisplay.from_estimator(rf_model, X_test, y_test)
plt.title('Random Forest Confusion Matrix')
plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```



5.3 Support Vector Machine (Non-linear Model)

In [11]:

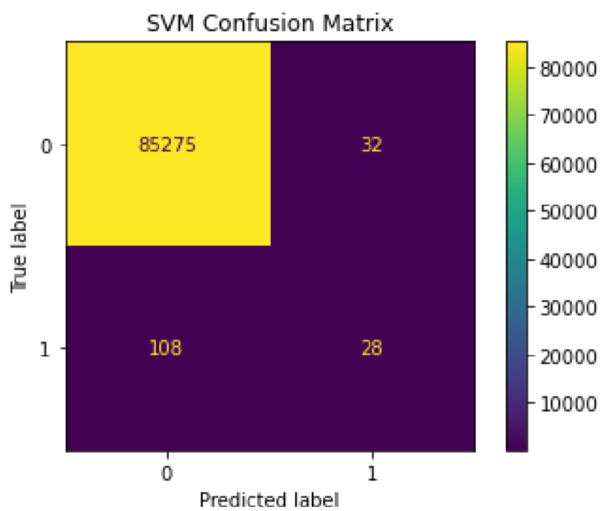
```python
from sklearn.svm import SVC
# Reduce the training set size for faster testing (only for debugging)
X_train_subset = X_train_res[:10000]
y_train_subset = y_train_res[:10000]


# Initialize SVM model with a linear kernel
svm_model = SVC(kernel='linear', random_state=42, C=0.1)

# Train the model on a smaller subset
svm_model.fit(X_train_subset, y_train_subset)

# Predict using the SVM model
svm_preds = svm_model.predict(X_test)

# Evaluate SVM model performance
print("SVM Performance:")
print(classification_report(y_test, svm_preds))
```

```
SVM Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.47      0.21      0.29       136

    accuracy                           1.00     85443
   macro avg       0.73      0.60      0.64     85443
weighted avg       1.00      1.00      1.00     85443
```

In [12]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix for Support Vector Machine (SVM)
plt.figure(figsize=(10, 8))  # Optional: adjust the figure size
ConfusionMatrixDisplay.from_estimator(svm_model, X_test, y_test)
plt.title('SVM Confusion Matrix')
plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```



Step 6: Hyperparameter Tuning

In [ ]:

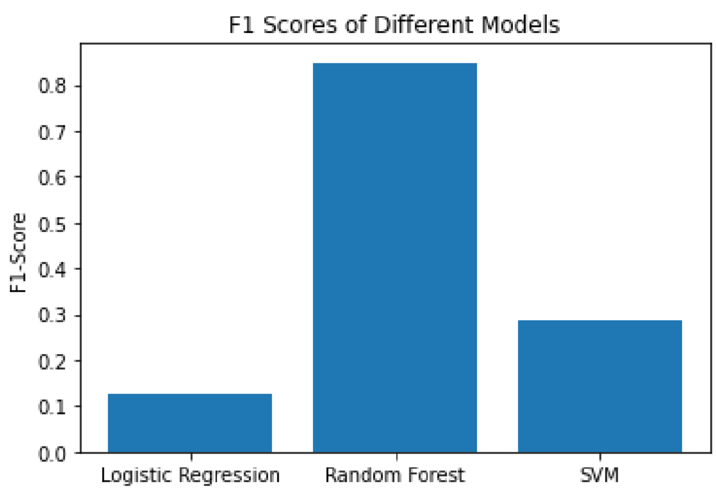
```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# Define the parameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': np.arange(50, 201, 50),
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10]
}

# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Perform Randomized Search
random_search = RandomizedSearchCV(rf_model, param_distributions=param_dist, n_iter
random_search.fit(X_train_res, y_train_res)

# Print the best parameters
print("Best parameters found: ", random_search.best_params_)
```

Step 7: Visualization of Results

7.1 F1-Score

In [17]:

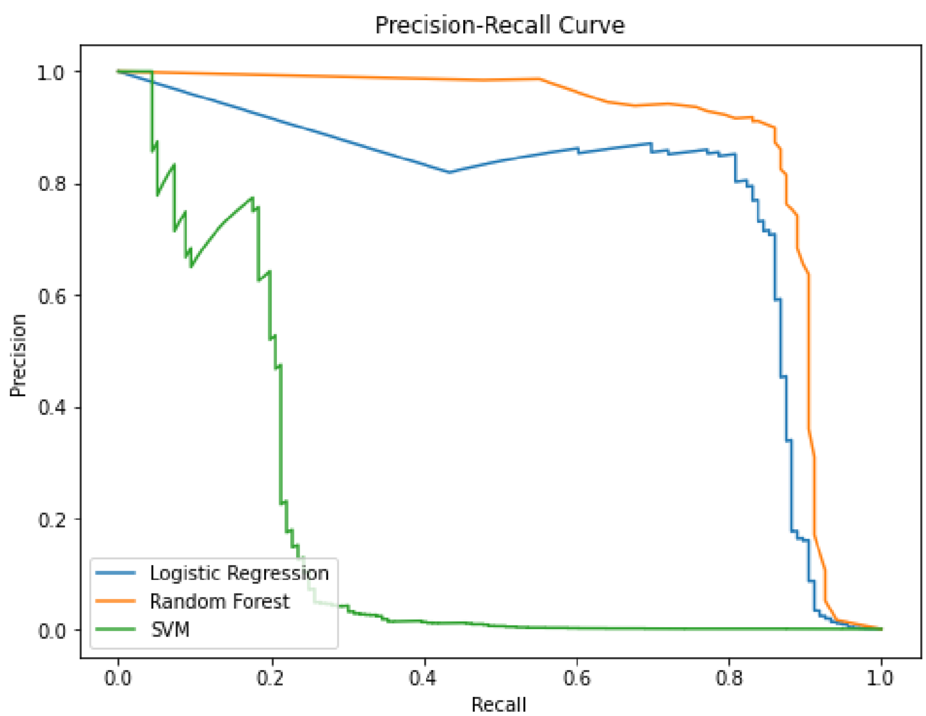
```
# Plotting the F1-scores for each model
models = ['Logistic Regression', 'Random Forest', 'SVM']
f1_scores = [classification_report(y_test, lr_preds, output_dict=True)['1']['f1-sco
             classification_report(y_test, rf_preds, output_dict=True)['1']['f1-sco
             classification_report(y_test, svm_preds, output_dict=True)['1']['f1-sc

# models = ['Logistic Regression', 'Random Forest']
# f1_scores = [classification_report(y_test, lr_preds, output_dict=True)['1']['f1-s
#              classification_report(y_test, rf_preds, output_dict=True)['1']['f1-s

plt.bar(models, f1_scores)
plt.title('F1 Scores of Different Models')
plt.ylabel('F1-Score')
plt.show()
```



7.2 Precision-Recall Curve

In [19]:

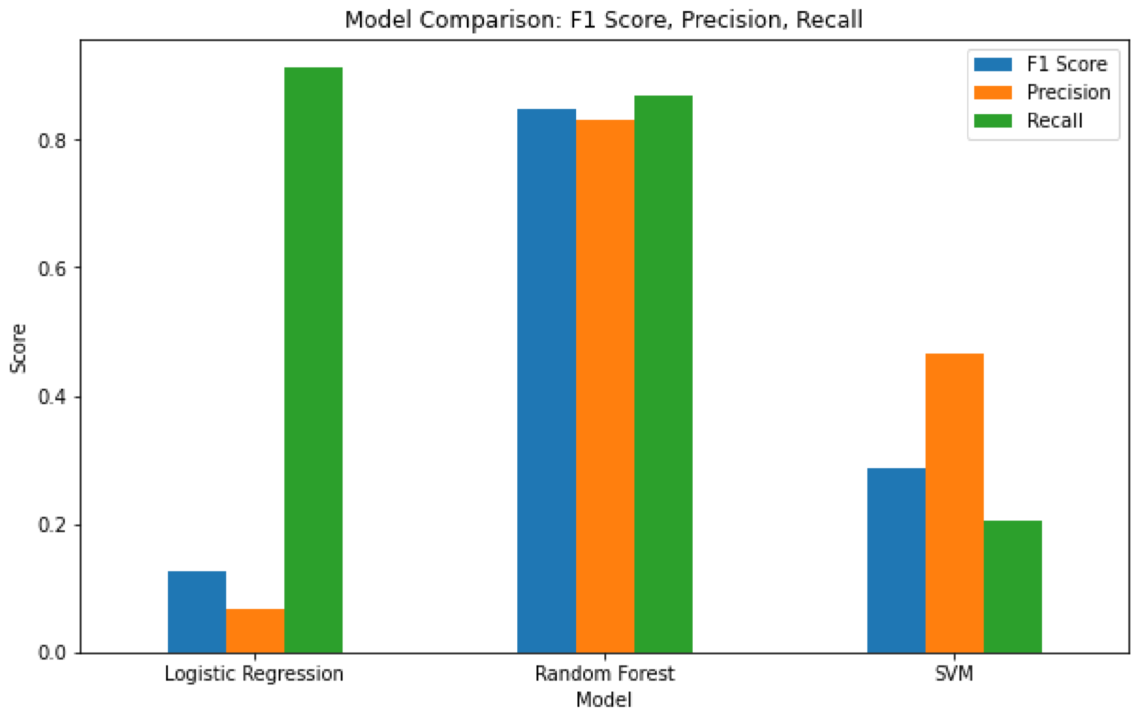
```
from sklearn.metrics import precision_recall_curve
from sklearn.preprocessing import label_binarize

# Binarize the target labels (important for multi-class classification, but works f
y_test_bin = label_binarize(y_test, classes=[0, 1])

# Calculate precision-recall for each model
precision_lr, recall_lr, _ = precision_recall_curve(y_test_bin, lr_model.predict_pr
precision_rf, recall_rf, _ = precision_recall_curve(y_test_bin, rf_model.predict_pr
precision_svm, recall_svm, _ = precision_recall_curve(y_test_bin, svm_model.decisic

# Plot Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recall_lr, precision_lr, label=f'Logistic Regression')
plt.plot(recall_rf, precision_rf, label=f'Random Forest')
plt.plot(recall_svm, precision_svm, label=f'SVM')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```



7.3 F1-Score, Precision, Recall

In [21]:

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

# Get metrics for each model
f1_lr = f1_score(y_test, lr_preds)
f1_rf = f1_score(y_test, rf_preds)
f1_svm = f1_score(y_test, svm_preds)

precision_lr = precision_score(y_test, lr_preds)
precision_rf = precision_score(y_test, rf_preds)
precision_svm = precision_score(y_test, svm_preds)

recall_lr = recall_score(y_test, lr_preds)
recall_rf = recall_score(y_test, rf_preds)
recall_svm = recall_score(y_test, svm_preds)

# Create a DataFrame to hold the values
# metrics_df = pd.DataFrame({
#     'Model': ['Logistic Regression', 'Random Forest', 'SVM'],
#     'F1 Score': [f1_lr, f1_rf],
#     'Precision': [precision_lr, precision_rf],
#     'Recall': [recall_lr, recall_rf]
# })

# Create a DataFrame to hold the values
metrics_df = pd.DataFrame({
    'Model': ['Logistic Regression', 'Random Forest', 'SVM'],
    'F1 Score': [f1_lr, f1_rf, f1_svm],
    'Precision': [precision_lr, precision_rf, precision_svm],
    'Recall': [recall_lr, recall_rf, recall_svm]
})

# Plot bar chart
metrics_df.set_index('Model').plot(kind='bar', figsize=(10, 6))
plt.title('Model Comparison: F1 Score, Precision, Recall')
plt.ylabel('Score')
plt.xticks(rotation=0)
plt.show()
```

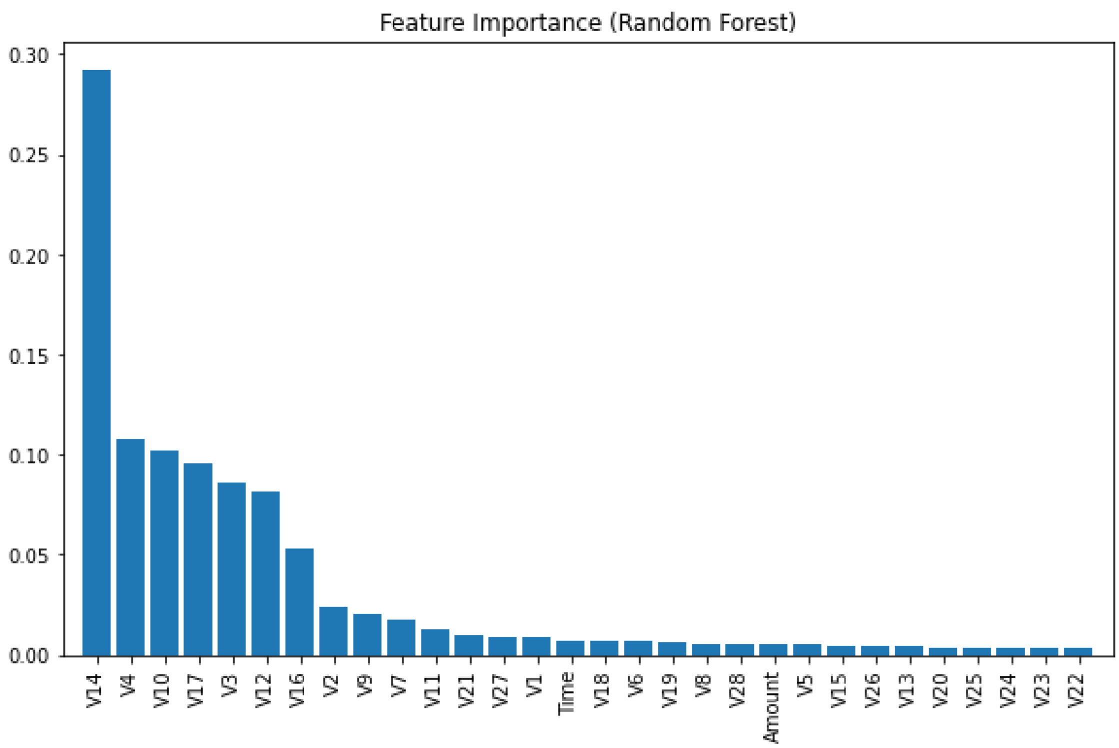


7.4 Feature Importance (for Random Forest)

Feature importance gives insight into which features have the most impact on predictions. This can help us understand your model better and possibly improve it by focusing on the most important features.

In [22]:

```python
# Get feature importances from Random Forest
importances = rf_model.feature_importances_

# Sort the feature importances in descending order
indices = importances.argsort()[::-1]

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.title('Feature Importance (Random Forest)')
plt.bar(range(X_test.shape[1]), importances[indices], align='center')
plt.xticks(range(X_test.shape[1]), X_test.columns[indices], rotation=90)
plt.xlim([-1, X_test.shape[1]])
plt.show()
```



Step 8: Tuned Random Forest

In [16]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Best hyperparameters
rf_model = RandomForestClassifier(
    n_estimators=100,
    min_samples_split=5,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)

# Train the model
rf_model.fit(X_train_res, y_train_res)

# Predict on test data
rf_preds = rf_model.predict(X_test)

# Evaluate performance
print("Random Forest Performance (Tuned):")
print(classification_report(y_test, rf_preds))
```

```
Random Forest Performance (Tuned):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.83      0.87      0.85       136

    accuracy                           1.00     85443
   macro avg       0.91      0.93      0.92     85443
weighted avg       1.00      1.00      1.00     85443
```