```
#load libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier

```

```
df = pd.read_csv('creditcard.csv')
```

```
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0

5 rows × 31 columns

```
df.describe()

```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
count	5974.000000	5974.000000	5974.000000	5974.000000	5974.000000	5974.000000	5974.000000	5974.000000	5974.000000	5974.000000	...
mean	2677.615501	-0.266159	0.285505	0.844231	0.104200	0.000709	0.194948	0.018324	-0.039006	0.396916	...
std	1765.025532	1.395405	1.208867	1.031448	1.442339	1.185900	1.365525	1.059870	1.304005	1.047749	...
min	0.000000	-12.168192	-15.732974	-12.389545	-4.657545	-32.092129	-7.465603	-12.968670	-23.632502	-3.336805	...
25%	1162.250000	-1.015749	-0.280054	0.295701	-0.839417	-0.609206	-0.677720	-0.492968	-0.189736	-0.264280	...
50%	2537.000000	-0.420703	0.346083	0.882882	0.161767	-0.083983	-0.142606	0.041761	0.037831	0.360826	...
75%	3781.750000	1.115402	0.941548	1.504158	1.071412	0.441406	0.605784	0.566306	0.343067	0.961662	...
max	6645.000000	1.685314	7.467017	4.101716	6.013346	10.658654	21.393069	34.303177	3.877662	9.272376	...

8 rows × 31 columns

```
# check for missing values
df.isnull().sum()
```

	0
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	1
V19	1
V20	1
V21	1
V22	1
V23	1
V24	1
V25	1
V26	1
V27	1
V28	1
Amount	1
Class	1

```
percentage_fraudulent = 100 * df[df.Class == 1].shape[0]/df[df.Class == 0].shape[0]
print('The percentage of fraudulent transactions is : %.2f percent' % percentage_fraudulent )
```

```
The percentage of fraudulent transactions is : 0.05 percent
```

```
features_list = list(df.columns)
features_list.remove("Class")
print(features_list)
```

```
['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
```

```
corr_Class = df.corr()['Class']
print(corr_Class)
```

```
Time     -0.011397
V1       -0.036739
V2       -0.001872
V3       -0.024730
```

```
V4        0.043025
V5        0.000083
V6       -0.017247
V7       -0.012015
V8        0.005970
V9       -0.031878
V10      -0.036398
V11       0.023579
V12      -0.042333
V13      -0.007063
V14      -0.063887
V15       0.011644
V16      -0.023274
V17      -0.070258
V18      -0.005445
V19      -0.005500
V20       0.021282
V21       0.008579
V22      -0.000547
V23       0.023886
V24      -0.001782
V25      -0.001859
V26      -0.005962
V27      -0.000569
V28      -0.007861
Amount    0.022274
Class     1.000000
Name: Class, dtype: float64
```

```
corr_Class = df.corr()['Class']
print(corr_Class)
```

```
Time      -0.011397
V1        -0.036739
V2        -0.001872
V3        -0.024730
V4         0.043025
V5         0.000083
V6        -0.017247
V7        -0.012015
V8         0.005970
V9        -0.031878
V10       -0.036398
V11        0.023579
V12       -0.042333
V13       -0.007063
V14       -0.063887
V15        0.011644
V16       -0.023274
V17       -0.070258
V18       -0.005445
V19       -0.005500
V20        0.021282
V21        0.008579
V22       -0.000547
V23        0.023886
V24       -0.001782
V25       -0.001859
V26       -0.005962
V27       -0.000569
V28       -0.007861
Amount     0.022274
Class      1.000000
Name: Class, dtype: float64
```

```
corr_Class = corr_Class[:-1].abs().sort_values(ascending = False)
corr_Class
```

	Class
V17	0.070258
V14	0.063887
V4	0.043025
V12	0.042333
V1	0.036739
V10	0.036398
V9	0.031878
V3	0.024730
V23	0.023886
V11	0.023579
V16	0.023274
Amount	0.022274
V20	0.021282
V6	0.017247
V7	0.012015
V15	0.011644
Time	0.011397
V21	0.008579
V28	0.007861
V13	0.007063
V8	0.005970
V26	0.005962
V19	0.005500
V18	0.005445
V2	0.001872
V25	0.001859
V24	0.001782
V27	0.000569
V22	0.000547
V5	0.000083

```
relevant_features = list(corr_Class[np.abs(corr_Class) > 0.09].index)
print(relevant_features)
```

```
[]
```

```
X = df.loc[:, relevant_features]
y = df['Class']
print(df['Class'].isnull().sum())
print('Shape of X: ' , X.shape)
print('Shape of y: ' , y.shape)
```

```
1
Shape of X:  (5974, 30)
Shape of y:  (5974,)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print('Shape of X_train: ' , X_train.shape)
print('Shape of y_train: ' , y_train.shape)
print('\nShape of X_test: ' , X_test.shape)
print('Shape of y_test: ' , y_test.shape)
```

```
Shape of X_train:  (4480, 0)
Shape of y_train:  (4480,)

Shape of X_test:  (1494, 0)
Shape of y_test:  (1494,)
```

```
print(X_test)
```

```
       Time        V1        V2        V3        V4        V5        V6  \
5956  6604 -0.965886  1.400465  0.546106 -1.072728  0.724583  0.082179
4332  3760  1.325005 -0.809991 -0.959412 -1.584234  1.419297  3.288968
2584  2113 -2.746065  3.022169 -0.134017 -0.386662 -0.244242  0.742847
1755  1356  1.057470  0.003263  0.339138  1.443896 -0.241933 -0.177943
4687  4112 -3.104906  1.210301  0.248620 -1.366092 -2.909295 -0.696190
...    ...       ...       ...       ...       ...       ...       ...
5142  4844  0.034083  1.126185  0.811825  0.537447 -0.128318  0.050313
3149  2726 -1.222026  0.142360  2.834213  0.098861  0.406521  0.071161
4774  4238 -0.519700  1.087853  1.475829  0.595549  0.135176  0.161461
5367  5314  1.297116  0.480839  0.231190  0.767106 -0.000046 -0.709100
1126   873 -1.535998  0.053411  0.657064 -0.699736  0.912905 -0.454612

            V7        V8        V9  ...       V20       V21       V22  \
5956 -0.139567 -2.800433  0.547775  ...  0.498242 -1.868922 -1.261180
4332 -1.117940  0.700098  0.355454  ...  0.302532 -0.254165 -0.919116
2584 -1.653331 -6.059684  0.366991  ... -1.072000  5.752131 -2.935378
1755  0.115970 -0.009037  0.343912  ... -0.103442 -0.072662 -0.116413
4687 -2.127176  2.164399 -0.098288  ... -0.622162  0.557374  0.894100
...        ...       ...       ...  ...       ...       ...       ...
5142 -0.985728 -2.731849  0.687019  ...  0.551764 -1.688356 -0.604620
3149  0.132806 -0.073322  0.558070  ... -0.196165  0.082535  0.507855
4774  1.215210 -1.401913  1.203209  ... -0.298942  0.616292 -0.032995
5367  0.009755 -0.272591  1.294087  ... -0.080749 -0.468757 -1.082688
1126 -0.213932  0.663058 -0.267338  ...  0.218537 -0.092907 -0.593964

           V23       V24       V25       V26       V27       V28  Amount
5956  0.014465 -1.098688 -0.215454  0.512105 -0.080029  0.096885   28.48
4332  0.046413  0.922049  0.342520 -0.512596 -0.027397  0.022321   81.48
2584  0.817265 -0.638756  0.153555  0.152716  0.342278 -0.076196    9.87
1755 -0.106484  0.074245  0.629171 -0.316821  0.026968  0.024700   63.65
4687 -0.098820  0.459199  0.024300 -0.289626 -0.798323 -0.272227   49.95
...        ...       ...       ...       ...       ...       ...     ...
5142  0.130605 -0.031477  0.661687  0.140126 -0.050927  0.131593    1.98
3149 -0.576997 -0.014051  0.385398 -0.396989 -0.497638 -0.341967    1.00
4774 -0.145831 -0.004458  0.093368 -0.435272 -0.330133 -0.463399  110.00
5367  0.069744 -0.243223  0.278078  0.087791 -0.047949  0.019847    1.98
1126 -0.200234 -0.815724 -0.085209  0.341124  0.187091 -0.051341    1.00

[1494 rows x 30 columns]
```

```
# %% [code]
X = df.loc[:, relevant_features]
y = df['Class']
print('Shape of X: ' , X.shape)
print('Shape of y: ' , y.shape)
```

```
Shape of X:  (5974, 0)
Shape of y:  (5974,)
```

```
# %% [code]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print('Shape of X_train: ' , X_train.shape)
print('Shape of y_train: ' , y_train.shape)
print('\nShape of X_test: ' , X_test.shape)
print('Shape of y_test: ' , y_test.shape)
```

```
Shape of X_train:  (4480, 0)
Shape of y_train:  (4480,)

Shape of X_test:  (1494, 0)
Shape of y_test:  (1494,)
```

```
print(relevant_features)
X = df.loc[:, relevant_features]
y = df['Class']
print('Shape of X: ' , X.shape)
print('Shape of y: ' , y.shape)
```

```
[]
Shape of X:  (5974, 0)
Shape of y:  (5974,)
```

```
print(df.head())
print(df.columns)
print(relevant_features)
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4     2 -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    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns]
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
[]
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
print('Shape of X_train before scaling:', X_train.shape)
```

```
Shape of X_train before scaling: (4480, 0)
```

```
print(X_train.shape)
print(X_train.head())

```

```
(4480, 0)
Empty DataFrame
Columns: []
Index: [1780, 4926, 5205, 4928, 304]
```

```
print(relevant_features)
print(all(feature in df.columns for feature in relevant_features))

```

```
[]
True
```

```
print(X_train.isnull().sum())
print(X_train.dtypes)

```

```
Series([], dtype: float64)
Series([], dtype: object)
```

```
relevant_features = list(corr_Class[corr_Class > 0.01].index)

```

```
relevant_features = df.columns.drop("Class")

```

```
X = df[relevant_features]
y = df["Class"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

print("X_train shape:", X_train.shape)
X_train_scaled = sc.fit_transform(X_train)

```

```
X_train shape: (4480, 30)
```

```
X_train_scaled = sc.fit_transform(X_train)
print('Shape of X_train_scaled: ' , X_train_scaled.shape)
```

```
Shape of X_train_scaled:  (4480, 30)
```

```
print(X_train_scaled)
print(y_train)
```

```
[[-0.74355793 -0.90473058  0.74980167 ...  1.35129608  0.69861793
  -0.28461563]
 [ 1.01650368 -3.44822996 -4.09195518 ...  3.15363869 -4.24182458
   0.10112522]
 [ 1.30633589 -0.73553053 -0.37454917 ...  0.46747216  0.09834108
   0.04066708]
 ...
 [ 1.33195956 -0.65644353  0.03036401 ... -0.06419891  0.25385364
  -0.30772673]
 [ 1.51417235 -0.47880532  0.46987828 ... -1.92805309 -0.18214841
  -0.01943368]
 [-1.15695318 -0.41984006  0.27536968 ...  0.39207649  1.06832549
  -0.31725708]]
1780    0.0
4926    0.0
5205    0.0
4928    0.0
304     0.0
       ...
3772    0.0
5191    0.0
5226    0.0
5390    0.0
860     0.0
Name: Class, Length: 4480, dtype: float64
```

```
print("Number of NaN values in y_train:", y_train.isnull().sum())
```

```
Number of NaN values in y_train: 1
```

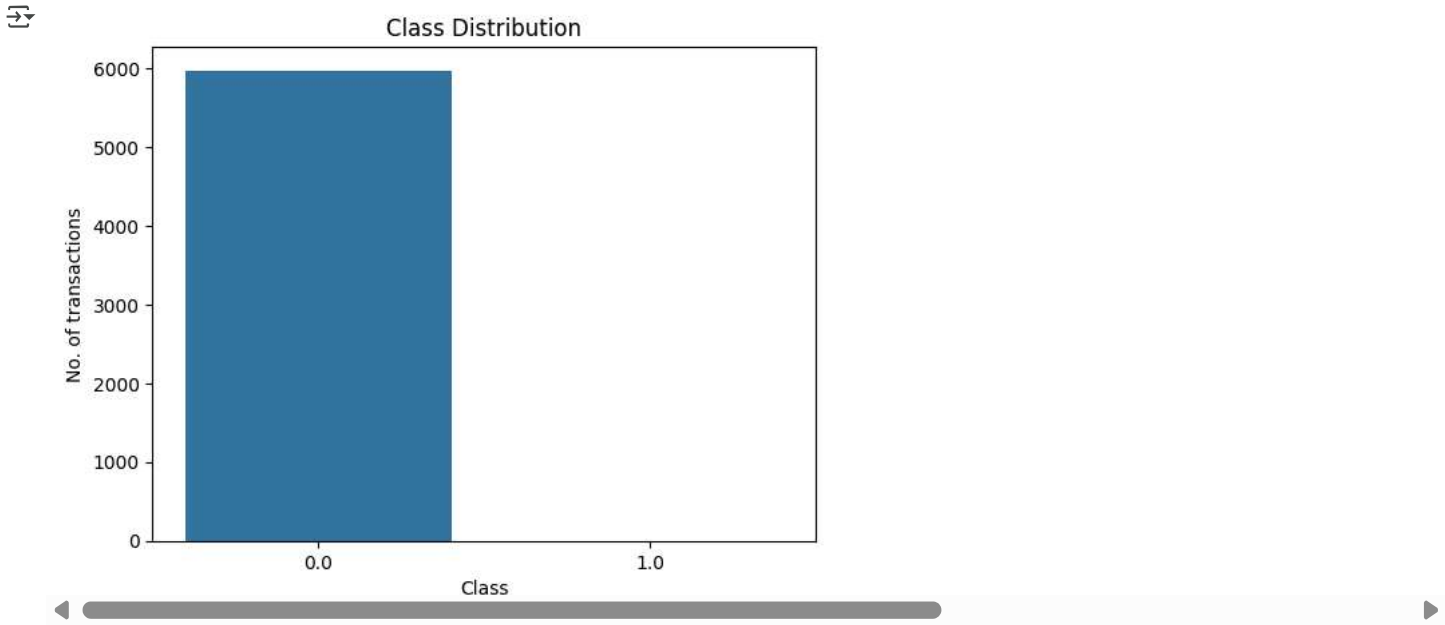
```
print("Number of NaN values in 'Class' column before handling:", df['Class'].isnull().sum())

# Remove rows where 'Class' has NaN values
df_cleaned = df.dropna(subset=['Class'])

# Verify that NaNs in 'Class' are removed
print("Number of NaN values in 'Class' column after handling:", df_cleaned['Class'].isnull().sum())

# Separate features (X) and target (y) from the cleaned DataFrame
X = df_cleaned.drop('Class', axis=1)
y = df_cleaned['Class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Scale the training data
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
print('Shape of X_train_scaled:', X_train_scaled.shape)
print('Shape of y_train:', y_train.shape)
```

```
Number of NaN values in 'Class' column before handling: 1
Number of NaN values in 'Class' column after handling: 0
```

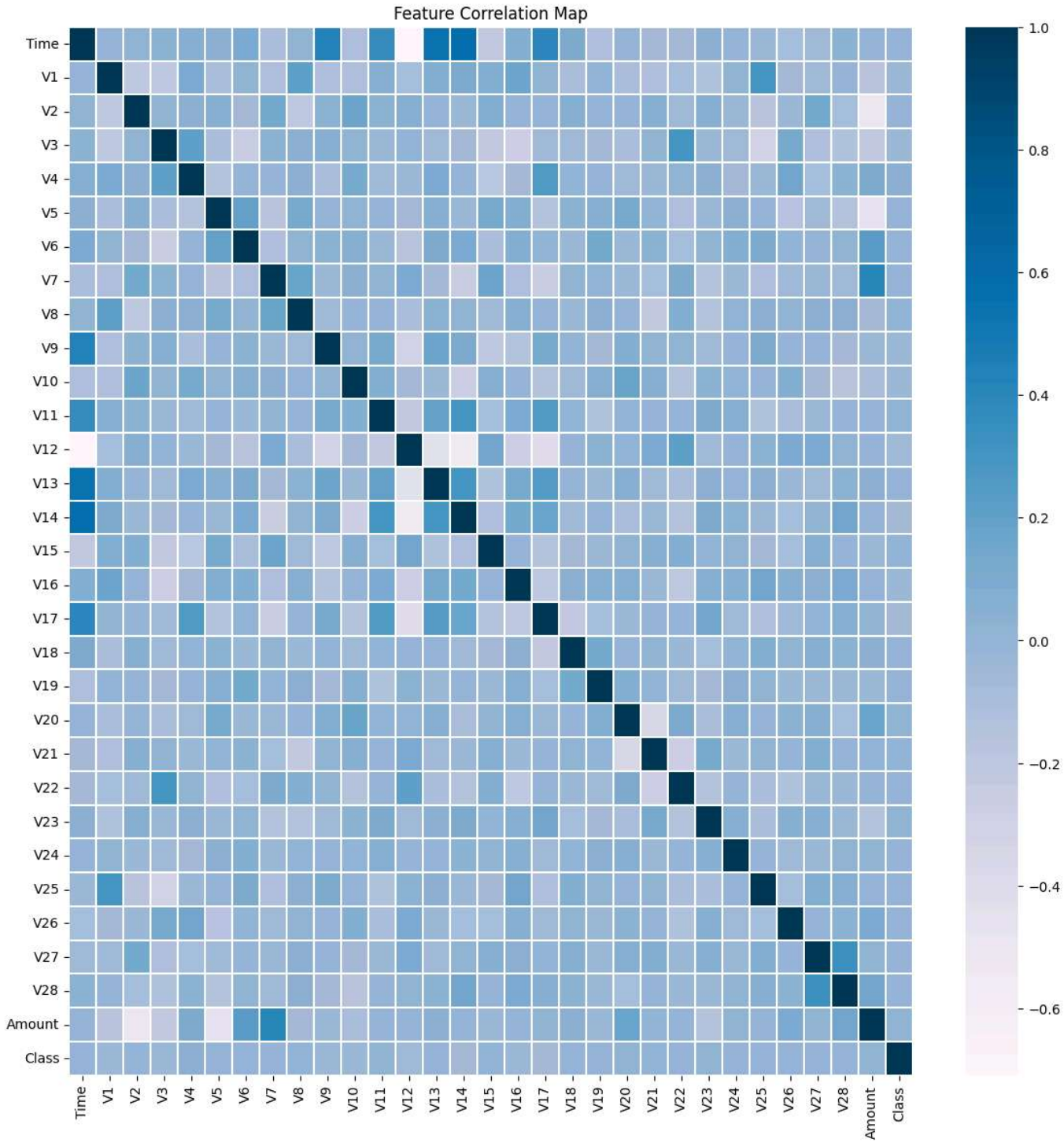
```
Shape of X_train_scaled: (4479, 30)
Shape of y_train: (4479,)
```

```
rf = RandomForestClassifier(max_depth=5, random_state=42)
rf.fit(X_train_scaled,y_train)

# We shouldn't forget to use the standard scaler to transform the features of the test set
X_test_scaled = sc.transform(X_test)
y_pred_rf = rf.predict(X_test_scaled)
```

```
sns.countplot(x = 'Class', data = df)
# df["Class"].value_counts().plot(kind = "bar")
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("No. of transactions")
plt.show()
```




```
plt.figure(figsize = (14, 14))
plt.title("Feature Correlation Map")
corr = df.corr()
sns.heatmap(corr, xticklabels = corr.columns, yticklabels = corr.columns, linewidths = .1, cmap = "PuBu")
plt.show()
```




```
# define predictors and target
target = "Class"
# all columns except class
predictors = df.columns.tolist()[:-1]
```

```
# split the data
train_df, test_df = train_test_split(df, test_size = 0.2, random_state = 42, shuffle = True)
train_df, valid_df = train_test_split(train_df, test_size = 0.2, random_state = 42, shuffle = True)
```

```
x_train, x_valid, x_test = train_df[predictors], valid_df[predictors], test_df[predictors]
y_train, y_valid, y_test = train_df[target].values, valid_df[target].values, test_df[target].values
```

```
clf = RandomForestClassifier(n_jobs = 4,
                             random_state = 42,
                             criterion = "gini",
                             n_estimators = 100,
                             verbose = False)
```

```
print("Number of NaN values in 'Class' column before handling:", df['Class'].isnull().sum())

# Remove rows where 'Class' has NaN values
df_cleaned = df.dropna(subset=['Class'])

# Verify that NaNs in 'Class' are removed
print("Number of NaN values in 'Class' column after handling:", df_cleaned['Class'].isnull().sum())

# Separate features (X) and target (y) from the cleaned DataFrame
X = df_cleaned.drop('Class', axis=1)
y = df_cleaned['Class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Scale the training data
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
print('Shape of X_train_scaled:', X_train_scaled.shape)
print('Shape of y_train:', y_train.shape)
```

```
Number of NaN values in 'Class' column before handling: 1
Number of NaN values in 'Class' column after handling: 0
Shape of X_train_scaled: (4479, 30)
Shape of y_train: (4479,)
```

```
print(train_df[target].isnull().sum())
print(np.isnan(train_df[target].values).sum())
```

```
1
1
```

```
print(f"Shape of train_df before dropping NaNs: {train_df.shape}")
train_df.dropna(subset=[target], inplace=True)
print(f"Shape of train_df after dropping NaNs: {train_df.shape}")
print(f"Number of NaNs in target after dropping: {train_df[target].isnull().sum()}")

# Now proceed with fitting the model
clf.fit(train_df[predictors], train_df[target].values)
```

```
Shape of train_df before dropping NaNs: (3823, 31)
Shape of train_df after dropping NaNs: (3822, 31)
Number of NaNs in target after dropping: 0
```

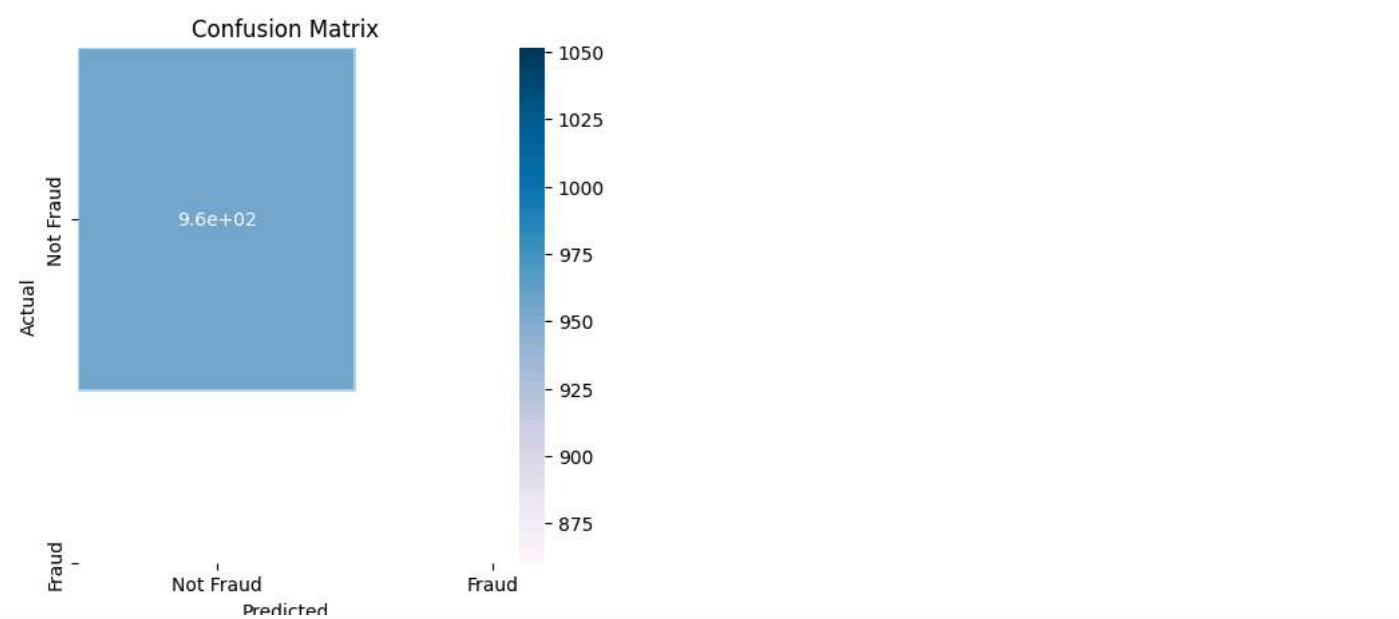
```
RandomForestClassifier
RandomForestClassifier(n_jobs=4, random_state=42, verbose=False)
```

```
preds = clf.predict(valid_df[predictors])
```

```
clf.score(test_df[predictors], test_df[target].values)
```

```
0.999163179916318
```

```
# confusion matrix
cm = pd.crosstab(valid_df[target].values, preds, rownames = ['Actual'], colnames = ['Predicted'])
fig, ax1 = plt.subplots(ncols = 1, figsize = (5, 5))
sns.heatmap(cm, xticklabels = ["Not Fraud", "Fraud"], yticklabels = ["Not Fraud", "Fraud"], annot = True, ax = ax1, linewidth = .2, line
plt.title("Confusion Matrix")
plt.show()
```




```
# area under curve
from sklearn.metrics import roc_auc_score
roc_auc_score(valid_df[target].values, preds)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:379: UndefinedMetricWarning: Only one class is present in y_true. RO
  warnings.warn(
nan
```

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(random_state = 42, algorithm = "SAMME", learning_rate = 0.8, n_estimators = 100)
```

```
print(f"Shape of train_df before dropping NaNs: {train_df.shape}")
train_df.dropna(subset=predictors, inplace=True)
print(f"Shape of train_df after dropping NaNs: {train_df.shape}")

# Now proceed with scaling and fitting
```

```
Shape of train_df before dropping NaNs: (3822, 31)
Shape of train_df after dropping NaNs: (3822, 31)
```

```
# Identify columns with a high percentage of NaNs (e.g., > 50%)
missing_percentage = train_df[predictors].isnull().sum() / len(train_df)
cols_to_drop = missing_percentage[missing_percentage > 0.5].index.tolist()
print(f"Columns to drop due to high missing values: {cols_to_drop}")
predictors = [col for col in predictors if col not in cols_to_drop]
train_df.drop(cols_to_drop, axis=1, inplace=True)
test_df.drop(cols_to_drop, axis=1, inplace=True, errors='ignore') # Handle cases where test_df might not have these columns
valid_df.drop(cols_to_drop, axis=1, inplace=True, errors='ignore')

# Now proceed with scaling and fitting using the remaining predictors
```

```
Columns to drop due to high missing values: []
```

```
from sklearn.impute import SimpleImputer

categorical_cols_with_nan = train_df[predictors].select_dtypes(include='object').columns[train_df[predictors].select_dtypes(include='obj

if categorical_cols_with_nan:
    imputer_categorical = SimpleImputer(strategy='most_frequent')
    train_df[categorical_cols_with_nan] = imputer_categorical.fit_transform(train_df[categorical_cols_with_nan])
    test_df[categorical_cols_with_nan] = imputer_categorical.transform(test_df[categorical_cols_with_nan])
    valid_df[categorical_cols_with_nan] = imputer_categorical.transform(valid_df[categorical_cols_with_nan])

# Now proceed with scaling (if applicable) and fitting
```

```
preds = clf.predict(x_valid)
```

```
# area under curve
roc_auc_score(y_valid, preds)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:379: UndefinedMetricWarning: Only one class is present in y_true. RO
  warnings.warn(
nan
```

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.14.1)
```

```
from sklearn.model_selection import train_test_split

# Assuming you already have X and y defined
X_train, X_valid, y_train, y_valid = train_test_split(
    X, y, test_size=0.2, random_state=42
)

```

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming you have your DataFrame 'df' loaded
df=pd.read_csv("creditcard.csv")
# Separate features (X) and target (y)
X = df.drop('Class', axis=1)
y = df['Class']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Now, use X_train_resampled and y_train_resampled for training your models
print("Shape of X_train before SMOTE:", X_train.shape)
print("Shape of y_train before SMOTE:", y_train.shape)
print("Shape of X_train after SMOTE:", X_train_resampled.shape)
print("Shape of y_train after SMOTE:", y_train_resampled.shape)

# Scaling the resampled training data
scaler = StandardScaler()
X_train_scaled_resampled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test) # Scale the test set using the fitted scaler

# Example of training a model on the resampled data
from sklearn.ensemble import RandomForestClassifier
rf_resampled = RandomForestClassifier(max_depth=5, random_state=0)
rf_resampled.fit(X_train_scaled_resampled, y_train_resampled)
y_pred_rf_resampled = rf_resampled.predict(X_test_scaled)

```

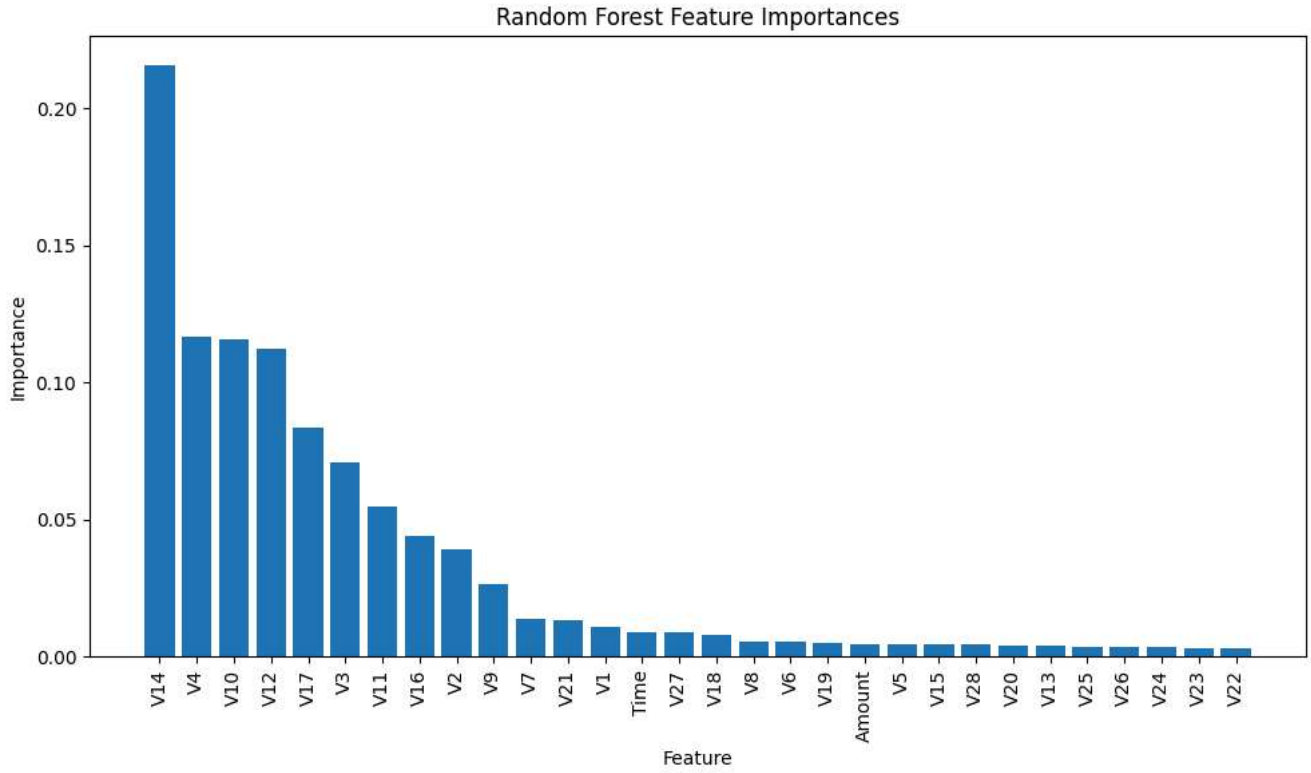
```
Exception ignored on calling ctypes callback function: <function ThreadpoolController._find_libraries_with_dl_iterate_phdr.<locals>.matc
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/threadpoolctl.py", line 1005, in match_library_callback
    self._make_controller_from_path(filepath)
  File "/usr/local/lib/python3.11/dist-packages/threadpoolctl.py", line 1187, in _make_controller_from_path
    lib_controller = controller_class(
                     ^^^^^^^^^^^^^^^^^
  File "/usr/local/lib/python3.11/dist-packages/threadpoolctl.py", line 114, in __init__
    self.dynlib = ctypes.CDLL(filepath, mode=_RTLD_NOLOAD)
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.11/ctypes/__init__.py", line 376, in __init__
    self._handle = _dlopen(self._name, mode)
                   ^^^^^^^^^^^^^^^^^^^^^^^^^
OSError: /usr/local/lib/python3.11/dist-packages/numpy.libs/libscipy_openblas64_-99b71e71.so: cannot open shared object file: No such fi
Shape of X_train before SMOTE: (213605, 30)
Shape of y_train before SMOTE: (213605,)
Shape of X_train after SMOTE: (426452, 30)
```

```
Shape of y_train after SMOTE: (426452,)
```

```
from sklearn.model_selection import train_test_split

# Drop rows with NaNs before split
X_clean = X.dropna()
y_clean = y[X_clean.index]

X_train, X_test, y_train, y_test = train_test_split(X_clean, y_clean, test_size=0.2, stratify=y_clean)

```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

df = pd.read_csv("creditcard.csv")

X = df.drop('Class', axis=1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

scaler = StandardScaler()
X_train_scaled_resampled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test)

# Train a Random Forest model to get feature importances
rf_feature_selection = RandomForestClassifier(n_estimators=100, random_state=42)
rf_feature_selection.fit(X_train_scaled_resampled, y_train_resampled)

# Get feature importances
importances = rf_feature_selection.feature_importances_
feature_names = X.columns
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

print("Feature Importances:")
print(feature_importance_df)

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.bar(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xticks(rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Random Forest Feature Importances')
plt.tight_layout()
plt.show()

# Select a subset of the most important features (e.g., top N)
top_n_features = feature_importance_df['Feature'][:10].tolist() # Select top 10
X_train_selected = X_train_resampled[top_n_features]
X_test_selected = X_test[top_n_features]

scaler_selected = StandardScaler()
X_train_scaled_selected = scaler_selected.fit_transform(X_train_selected)
X_test_scaled_selected = scaler_selected.transform(X_test_selected)

# Train and evaluate a model using the selected features
rf_selected = RandomForestClassifier(max_depth=5, random_state=0)
rf_selected.fit(X_train_scaled_selected, y_train_resampled)
y_pred_rf_selected = rf_selected.predict(X_test_scaled_selected)
print("\nClassification Report with Selected Features (Random Forest):")
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_rf_selected))
```

```
Feature Importances:
    Feature  Importance
14     V14    0.215406
4       V4    0.116516
10     V10    0.115802
12     V12    0.112384
17     V17    0.083300
3       V3    0.070909
11     V11    0.054797
16     V16    0.043997
2       V2    0.039316
9       V9    0.026457
7       V7    0.013947
21     V21    0.013179
1       V1    0.010606
0     Time    0.008776
27     V27    0.008667
18     V18    0.007708
8       V8    0.005464
6       V6    0.005427
19     V19    0.005192
29  Amount    0.004646
5       V5    0.004609
15     V15    0.004365
28     V28    0.004303
20     V20    0.003880
13     V13    0.003825
25     V25    0.003458
26     V26    0.003449
24     V24    0.003322
23     V23    0.003198
22     V22    0.003093
```



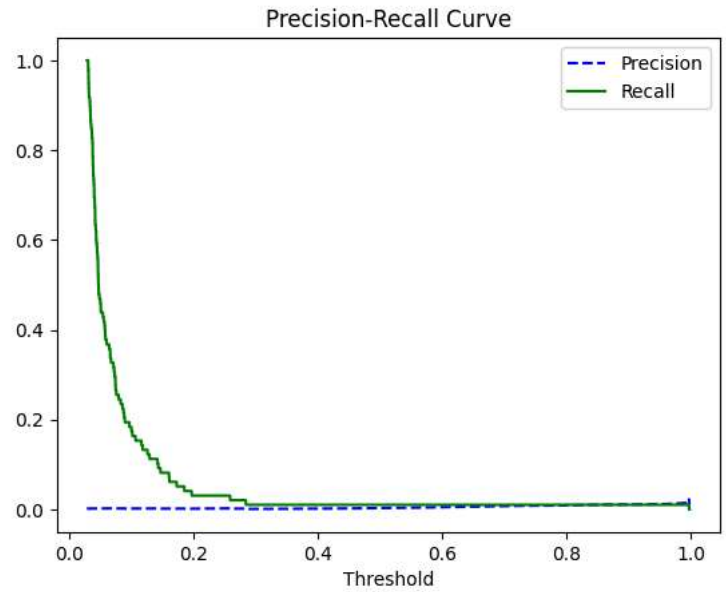

```
Classification Report with Selected Features (Random Forest):
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     71089
           1       0.17      0.90      0.28       113

    accuracy                           0.99     71202
   macro avg       0.58      0.95      0.64     71202
weighted avg       1.00      0.99      1.00     71202
```

```
print("X_test_scaled shape:", X_test_scaled.shape)
print("y_test shape:", y_test.shape)

```

```
X_test_scaled shape: (71202, 30)
y_test shape: (56962,)
```

```
# If X_test_scaled is the reduced set, you need the matching y_test
# Make sure you retained the right y_test when transforming X_test_scaled

# You can zip back them together temporarily:
if X_test_scaled.shape[0] < y_test.shape[0]:
    y_test = y_test[:X_test_scaled.shape[0]]
elif X_test_scaled.shape[0] > y_test.shape[0]:
    X_test_scaled = X_test_scaled[:y_test.shape[0]]

```

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

def plot_pr_curve(y_true, y_proba):
    precision, recall, thresholds = precision_recall_curve(y_true, y_proba)
    plt.plot(thresholds, precision[:-1], "b--", label="Precision")
    plt.plot(thresholds, recall[:-1], "g-", label="Recall")
    plt.xlabel("Threshold")
    plt.legend()
    plt.title("Precision-Recall Curve")
    plt.show()

def find_optimal_threshold(y_true, y_proba):
    precision, recall, thresholds = precision_recall_curve(y_true, y_proba)
    # You can choose the threshold that maximizes a specific metric, e.g., F1-score
    f1_scores = 2 * (precision[:-1] * recall[:-1]) / (precision[:-1] + recall[:-1] + 1e-8) # Avoid division by zero
    optimal_threshold_index = np.argmax(f1_scores)
    optimal_threshold = thresholds[optimal_threshold_index]
    print(f"Optimal Threshold: {optimal_threshold:.4f}")
    return optimal_threshold

# Assuming you have a trained model (e.g., rf_resampled) and test data (X_test_scaled, y_test)
y_proba_rf = rf_resampled.predict_proba(X_test_scaled)[:, 1] # Get probability of being fraudulent

plot_pr_curve(y_test, y_proba_rf)
optimal_threshold_rf = find_optimal_threshold(y_test, y_proba_rf)

# Make predictions using the optimal threshold
y_pred_optimal_rf = (y_proba_rf >= optimal_threshold_rf).astype(int)

# Evaluate the model with the optimal threshold
from sklearn.metrics import classification_report
print("Classification Report with Optimal Threshold (Random Forest):")
print(classification_report(y_test, y_pred_optimal_rf))
```




```
Optimal Threshold: 0.9975
Classification Report with Optimal Threshold (Random Forest):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.02      0.01      0.01        98

    accuracy                           1.00     56962
   macro avg       0.51      0.50      0.51     56962
weighted avg       1.00      1.00      1.00     56962
```

```
from sklearn.metrics import roc_curve, roc_auc_score, precision_recall_curve, average_precision_score, classification_report
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

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

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# --- Model 1: Random Forest without SMOTE ---
rf_no_smote = RandomForestClassifier(random_state=42)
rf_no_smote.fit(X_train_scaled, y_train)
y_pred_rf_no_smote = rf_no_smote.predict(X_test_scaled)
y_proba_rf_no_smote = rf_no_smote.predict_proba(X_test_scaled)[:, 1]

# --- Model 2: Random Forest with SMOTE ---
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)

rf_with_smote = RandomForestClassifier(random_state=42)
rf_with_smote.fit(X_train_resampled, y_train_resampled)
y_pred_rf_with_smote = rf_with_smote.predict(X_test_scaled)
y_proba_rf_with_smote = rf_with_smote.predict_proba(X_test_scaled)[:, 1]

# --- Plotting ROC Curves ---
plt.figure(figsize=(10, 6))
fpr_rf_no_smote, tpr_rf_no_smote, _ = roc_curve(y_test, y_proba_rf_no_smote)
auc_rf_no_smote = roc_auc_score(y_test, y_proba_rf_no_smote)
plt.plot(fpr_rf_no_smote, tpr_rf_no_smote, label=f'RF (No SMOTE), AUC = {auc_rf_no_smote:.2f}')

fpr_rf_with_smote, tpr_rf_with_smote, _ = roc_curve(y_test, y_proba_rf_with_smote)
```