```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.11/dist-packages (0.13.0)
Requirement already satisfied: numpy<3,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (2.0.2)
Requirement already satisfied: scipy<2,>=1.10.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.15.2)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.6.1
Requirement already satisfied: sklearn-compat<1,>=0.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (0.1.3)
Requirement already satisfied: joblib<2,>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn) (3.6.0)
```

Additional connection options

```
from google.colab import files
uploaded = files.upload()
```

Choose Files creditcard.csv

- **creditcard.csv**(text/csv) - 150828752 bytes, last modified: 5/8/2025 - 100% done

```
Saving creditcard.csv to creditcard (1).csv
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

from imblearn.over_sampling import SMOTE
```

```
df = pd.read_csv('creditcard.csv')
print(df.head())
print(df['Class'].value_counts())
```

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

```
scaler = StandardScaler()
df['scaled_Amount'] = scaler.fit_transform(df[['Amount']])
df['scaled_Time'] = scaler.fit_transform(df[['Time']])
df.drop(['Time', 'Amount'], axis=1, inplace=True)
```

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

```
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)



X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.3, random_state=42)



# Reduce dataset size for faster training (optional)
X_res = X_res.sample(n=10000, random_state=42)
y_res = y_res.loc[X_res.index]
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.3, random_state=42)




# Initialize and train the RandomForestClassifier model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Additional connection options

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

           0       0.98      1.00      0.99      1528
           1       1.00      0.98      0.99      1472

    accuracy                           0.99      3000
   macro avg       0.99      0.99      0.99      3000
weighted avg       0.99      0.99      0.99      3000
```