In [1]:
```
import pandas as pd
import numpy as np
```

```
C:\Users\Dell\AppData\Roaming\Python\Python311\site-packages\pandas\core
\arrays\masked.py:60: UserWarning: Pandas requires version '1.3.6' or new
er of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
df=pd.read_csv("C:\\Users\\Dell\\Downloads\\archive (2)\\creditcard.csv")
```

In [3]:
```
df
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.(
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.1
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.(
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.2
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.(
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.5

284807 rows × 31 columns

In [4]:
```
#Data preprocessing
#Handle missing values
```

In [5]:
```
#check 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
```

In [6]:
```
df=df.dropna()
```

In [7]:
```
# Features and labels
X = df.drop(columns=['Class', 'Time', 'Amount'])  # Dropping 'Time' and 'A
y = df['Class']
```

In [8]:
```
from sklearn.preprocessing import StandardScaler
```

In [9]:
```
# Normalize 'Amount' feature
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
```

In [10]:
```
from sklearn.model_selection import train_test_split
```

In [11]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
```

In [12]:
```
from sklearn.linear_model import LogisticRegression
```

In [13]:
```
# Train Logistic Regression model with class_weight='balanced'
lr_model = LogisticRegression(class_weight='balanced', random_state=42)
lr_model.fit(X_train, y_train)
```

Out[13]:

```
LogisticRegression(class_weight='balanced', random_state=42)
```

In [14]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [15]:
```
# Predict on the test set
y_pred_lr = lr_model.predict(X_test)
```

In [16]:
```
# Evaluate model performance
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr))
```

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

           0       1.00      0.98      0.99     85295
           1       0.06      0.88      0.12       148

    accuracy                           0.98     85443
   macro avg       0.53      0.93      0.55     85443
weighted avg       1.00      0.98      0.99     85443
```

In [17]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [18]:
```
# Train Random Forest model with class_weight='balanced'
rf_model = RandomForestClassifier(class_weight='balanced', random_state=4
rf_model.fit(X_train, y_train)
```

Out[18]:

```
RandomForestClassifier(class_weight='balanced', random_state=42)
```

In [19]:
```
# Predict on the test set
y_pred_rf = rf_model.predict(X_test)
```

In [20]:
```
# Evaluate Random Forest model
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

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

           0       1.00      1.00      1.00     85295
           1       0.97      0.70      0.82       148

    accuracy                           1.00     85443
   macro avg       0.99      0.85      0.91     85443
weighted avg       1.00      1.00      1.00     85443
```

In [21]:
```
# Confusion matrix
print("Confusion Matrix (Logistic Regression):")
```

```
print(confusion_matrix(y_test, y_pred_lr))
```

```
Confusion Matrix (Logistic Regression):
[[83396  1899]
 [   18   130]]
```

In [22]:
```
print("Confusion Matrix (Random Forest):")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Confusion Matrix (Random Forest):
[[85292     3]
 [   44   104]]
```

In [ ]:
```

```