Importing Library

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

In [2]:
```python
df = pd.read_csv('creditcard.csv')
```

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

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431
...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057

284807 rows × 31 columns

In [4]:
```python
df.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278

5 rows × 31 columns

In [5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
df.describe()
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2
mean	94813.859575	1.759061e-12	-8.251130e-13	-9.654937e-13	8.321385e-13	1.649999e-13	4.248366e-13	-3.054600e-13	8
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2

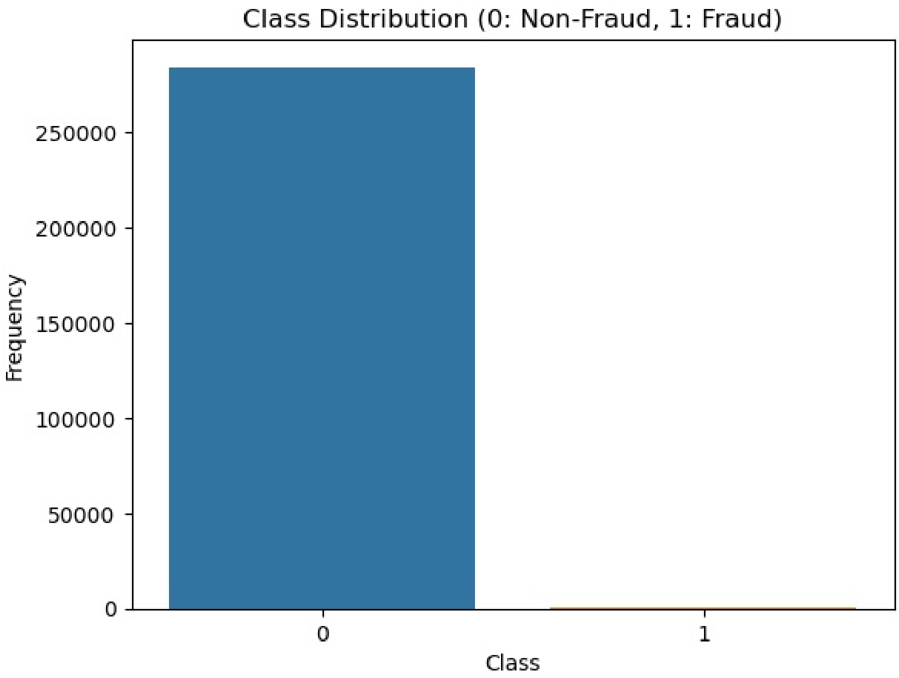
8 rows × 31 columns

In [8]:
```python
df.shape
```

Out[8]:
```
(284807, 31)
```

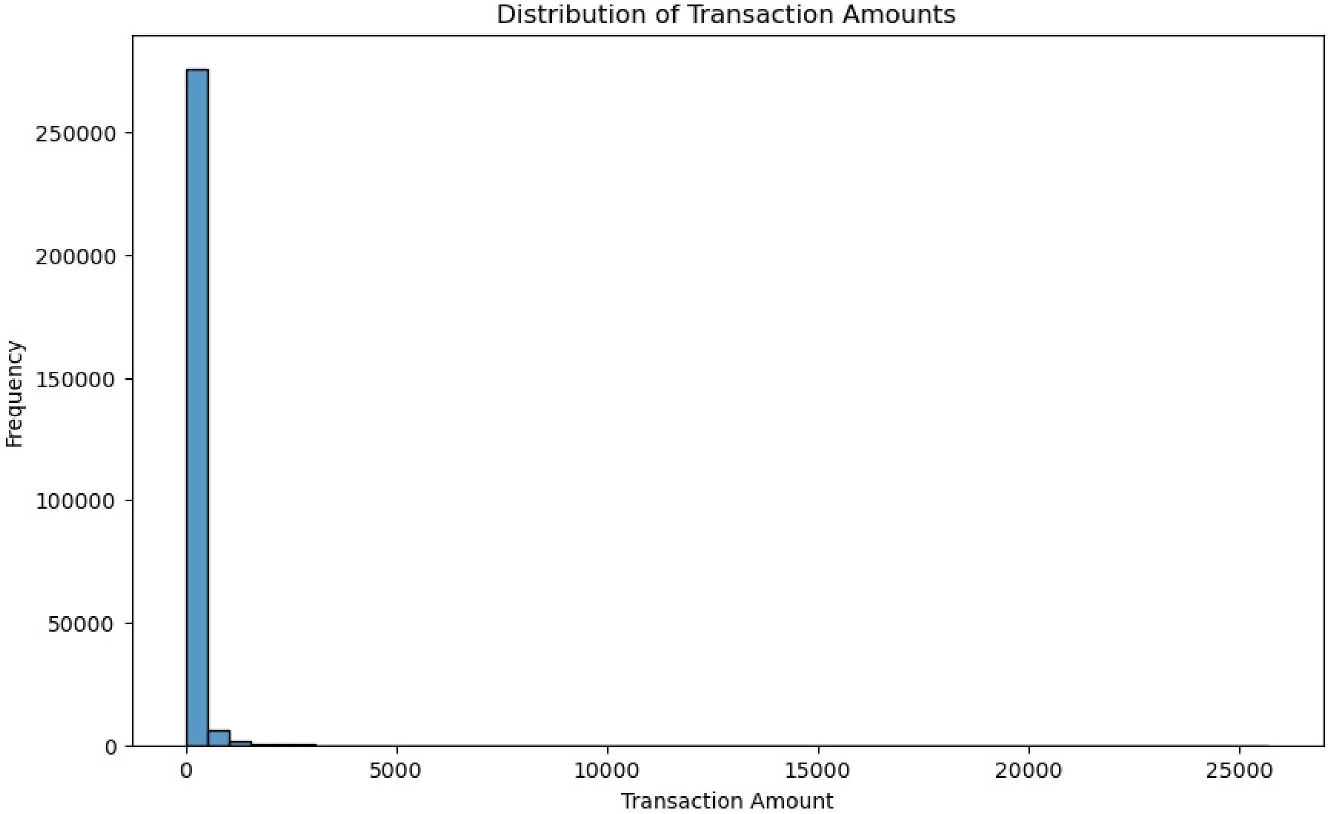
In [9]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

In [10]:
```python
# Visualize the class distribution (fraudulent vs non-fraudulent transactions)
class_counts = df['class'].value_counts()
sns.barplot(x=class_counts.index, y=class_counts.values)
plt.title("Class Distribution (0: Non-Fraud, 1: Fraud)")
plt.xlabel("Class")
plt.ylabel("Frequency")
plt.show()
```



In [11]:
```python
# Display the distribution of transaction amounts
plt.figure(figsize=(10,6))
sns.histplot(df['Amount'], bins=50, kde=False)
plt.title("Distribution of Transaction Amounts")
plt.xlabel("Transaction Amount")
plt.ylabel("Frequency")
plt.show()
```

```
C:\Users\Nihira Khare\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [13]:
```
# Features and target variable
X = df.drop(columns=['class'])
y = df['class']
```

In [14]:
```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

In [15]:
```
# Apply feature scaling to the 'Amount' and 'Time' columns
scaler = StandardScaler()
```

In [16]:
```
# Scaling 'Amount' and 'Time' features
X_train[['Amount', 'Time']] = scaler.fit_transform(X_train[['Amount', 'Time']])
X_test[['Amount', 'Time']] = scaler.transform(X_test[['Amount', 'Time']])
```

In [17]:
```
X_train.head(), X_test.head()
```

Out[17]:
```
(             Time        V1        V2        V3        V4        V5        V6  \
 265518  1.411588  1.946747 -0.752526 -1.355130 -0.661630  1.502822  4.024933
 180305  0.623141  2.035149 -0.048880 -3.058693  0.247945  2.943487  3.298697
 42664  -1.130680 -0.991920  0.603193  0.711976 -0.992425 -0.825838  1.956261
 198723  0.794699  2.285718 -1.500239 -0.747565 -1.668119 -1.394143 -0.350339
 82325  -0.748102 -0.448747 -1.011440  0.115903 -3.454854  0.715771 -0.147490

               V7        V8        V9  ...       V20       V21       V22  \
 265518 -1.479661  1.139880  1.406819  ... -0.134435  0.076197  0.297537
 180305 -0.002192  0.674782  0.045826  ... -0.227279  0.038628  0.228197
 42664  -2.212603 -5.037523  0.000772  ...  1.280856 -2.798352  0.109526
 198723 -1.427984  0.010010 -1.118447  ... -0.490642 -0.139670  0.077013
 82325   0.504347 -0.113817 -0.044782  ... -0.275297 -0.243245 -0.173298

              V23       V24       V25       V26       V27       V28    Amount
 265518  0.307915  0.690980 -0.350316 -0.388907  0.077641 -0.032248 -0.322494
 180305  0.035542  0.707090  0.512885 -0.471198  0.002520 -0.069002 -0.339764
 42664  -0.436530 -0.932803  0.826684  0.913773  0.038049  0.185340  0.346693
 198723  0.208310 -0.538236 -0.278032 -0.162068  0.018045 -0.063005 -0.327360
 82325  -0.006692 -1.362383 -0.292234 -0.144622 -0.032580 -0.064194 -0.008281

 [5 rows x 30 columns],
             Time        V1        V2        V3        V4        V5        V6  \
 263020  1.387182 -0.674466  1.408105 -1.110622 -1.328366  1.388996 -1.308439
 11378  -1.580138 -2.829816 -2.765149  2.537793 -1.074580  2.842559 -2.153536
 147283 -0.138120 -3.576495  2.318422  1.306985  3.263665  1.127818  2.865246
 219439  0.986536  2.060386 -0.015382 -1.082544  0.386019 -0.024331 -1.074935
 36939  -1.182272  1.209965  1.384303 -1.343531  1.763636  0.662351 -2.113384

               V7        V8        V9  ...       V20       V21       V22  \
 263020  1.885879 -0.614233  0.311652  ...  0.394322  0.080084  0.810034
 11378  -1.795519 -0.250020  3.073504  ... -0.515765 -0.295555  0.109305
 147283  1.444125 -0.718922  1.874046  ...  2.034786 -1.060151  0.016867
 219439  0.207792 -0.338140  0.455091  ... -0.192024 -0.281684 -0.639426
 36939   0.854039 -0.475963 -0.629658  ...  0.009083 -0.164015 -0.328294

              V23       V24       V25       V26       V27       V28    Amount
 263020 -0.224327  0.707899 -0.135837  0.045102  0.533837  0.291319 -0.259954
 11378  -0.813272  0.042996 -0.027660 -0.910247  0.110802 -0.511938 -0.304426
 147283 -0.132058 -1.483996 -0.296011  0.062823  0.552411  0.509764 -0.048286
 219439  0.331818 -0.067584 -0.283675  0.203529 -0.063621 -0.060077 -0.347741
 36939  -0.154631  0.619449  0.818998 -0.330525  0.046884  0.104527 -0.345707

 [5 rows x 30 columns])
```

In [18]:
```python
from sklearn.linear_model import LogisticRegression
```

In [19]:
```python
# Initialize the model
model_lr = LogisticRegression(max_iter=1000, random_state=42)
```

In [20]:
```python
# Train the model
model_lr.fit(X_train, y_train)
```

Out[20]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

In [21]:
```python
# Predict on the test set
y_pred = model_lr.predict(X_test)
```

In [22]:
```python
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

In [23]:
```python
# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[56851    13]
 [   34    64]]
```

In [24]:
```python
# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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

           0       1.00      1.00      1.00     56864
           1       0.83      0.65      0.73        98

    accuracy                           1.00     56962
   macro avg       0.92      0.83      0.87     56962
weighted avg       1.00      1.00      1.00     56962
```

In [25]:
```
# ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_pred)
print("ROC-AUC Score:", roc_auc)
```

```
ROC-AUC Score: 0.8264163044227257
```

Handling imbalanced data for Logistic Regression

In [26]:
```
from imblearn.over_sampling import SMOTE
```

In [27]:
```
# Initialize SMOTE
smote = SMOTE(random_state=42)
```

In [28]:
```
# Apply SMOTE to the training data
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
```

In [29]:
```
# Train the model on the balanced data
model_lr.fit(X_train_sm, y_train_sm)
```

Out[29]:
```
▾ LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```