In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Load the dataset
data = pd.read_csv("creditcard.csv") data
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount
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	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62
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	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69
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	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66
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	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50
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	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00

284807 rows × 31 columns

In [4]:
```
# Check basic statistics
print("\nDescriptive statistics:") data.describe()
```

Descriptive statistics:

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05	2.848070e+05	2.848
mean	94813.859575	1.759061e-12	-8.251130e-13	-9.654937e-13	8.321385e-13	1.649999e-13	4.248366e-13	-3.054600e-13	8.777971e-14	-1.179749e-12	...	3.405756e-13	-5.723197e-13	-9.725856e-13	1.46
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.05
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	2.283949e-01	-5.423504e-01	-1.618463e-01	-3.54
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	2.945017e-02	6.781943e-03	-1.119293e-02	4.09
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.39
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584

8 rows × 31 columns

In [5]:
```
# Check for missing values
print("\nMissing values:")
data.isnull().sum()
```

```
Missing values:
```

Out[5]:

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

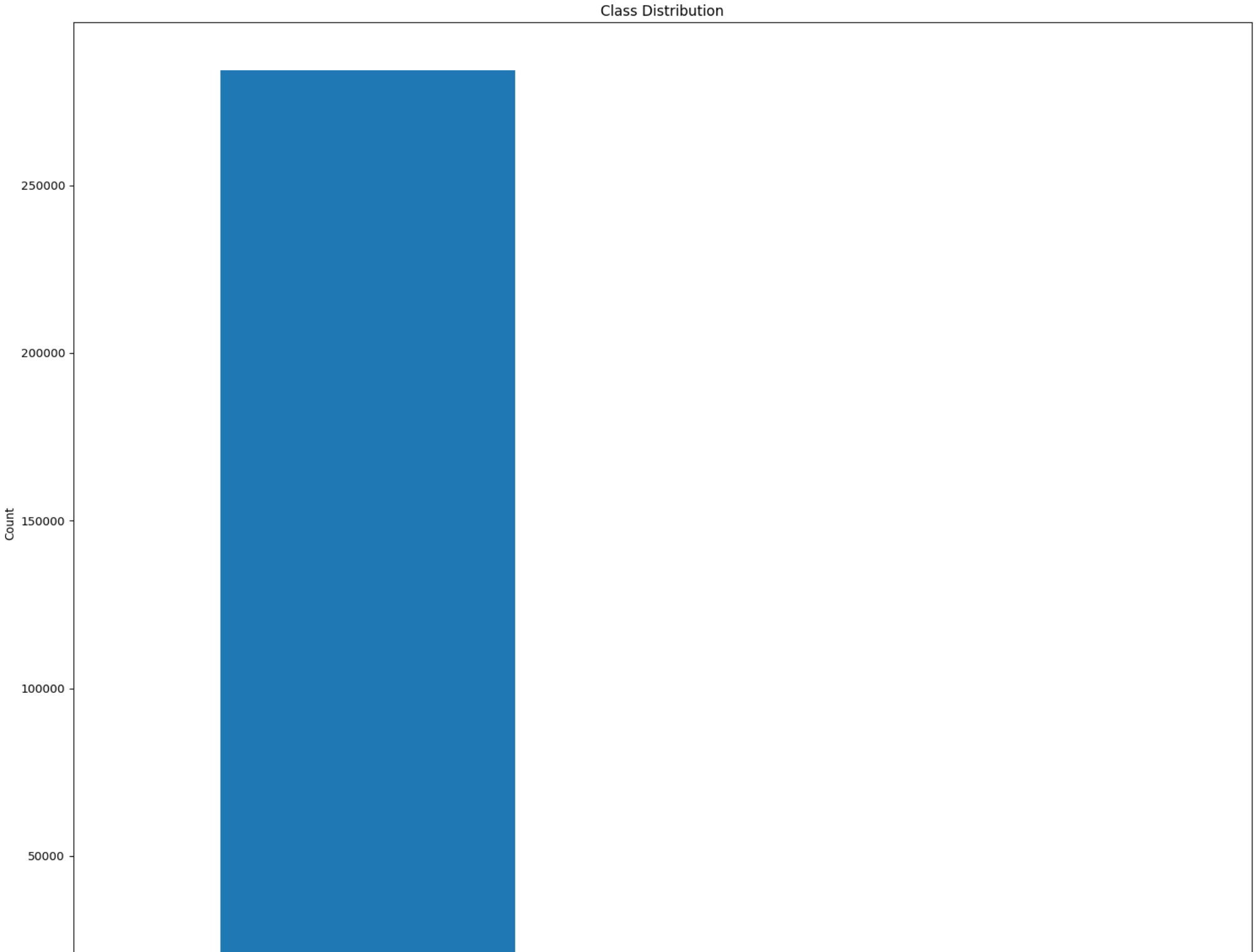
dtype: int64

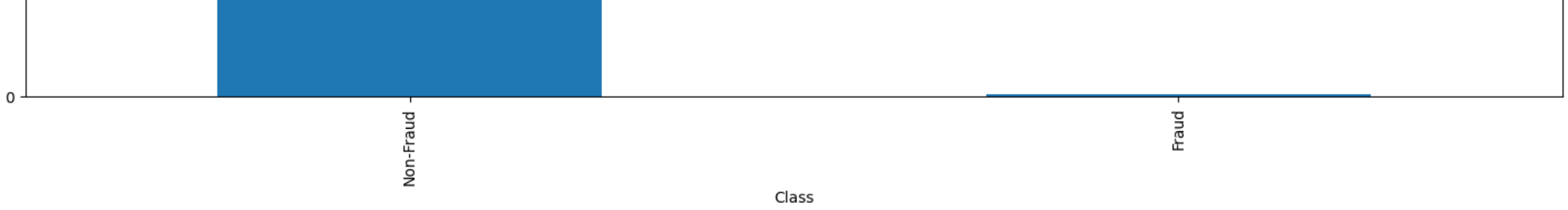
In [6]:
```
# Check class distribution
print("\nClass distribution:")
print(data['Class'].value_counts())
```

```
Class distribution:
0    284315
1       492
Name: Class, dtype: int64
```

In [7]

```
# Visualize class distribution
plt.figure(figsize=(18, 16))
data['Class'].value_counts().plot(kind='bar') plt.title('Class Distribution')
plt.xlabel('Class') plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['Non-Fraud', 'Fraud']) plt.show()
```


Class Distribution
Count
250000
200000
150000
100000
50000


0
Non-Fraud
Fraud
Class

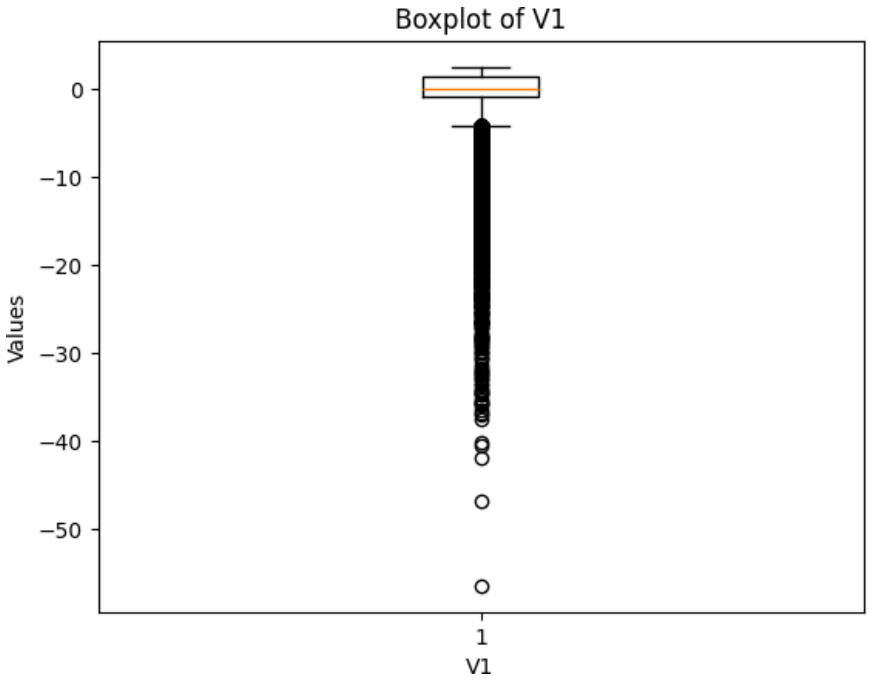
In [8]:

```
# Create a boxplot for 'V1'
df = data
plt.boxplot(df['V1'])

# Add title and labels
plt.title('Boxplot of V1') plt.xlabel('V1')
plt.ylabel('Values')

# Calculate upper and lower outliers
q1 = df['V1'].quantile(0.25)
q3 = df['V1'].quantile(0.75) iqr = q3 -
q1
upper_outliers = df[df['V1'] > (q3 + 1.5 * iqr)] lower_outliers =
df[df['V1'] < (q1 - 1.5 * iqr)]

# Count upper and lower outliers
count_upper_outliers = len(upper_outliers) count_lower_outliers =
len(lower_outliers)

print("Count of upper outliers:", count_upper_outliers) print("Count of lower
outliers:", count_lower_outliers)
```

In [ ]:

```
X = df.columns L =
X.tolist()
```

Count of upper outliers: 0
Count of lower outliers: 7062



In [10]:
```
print(L)
```

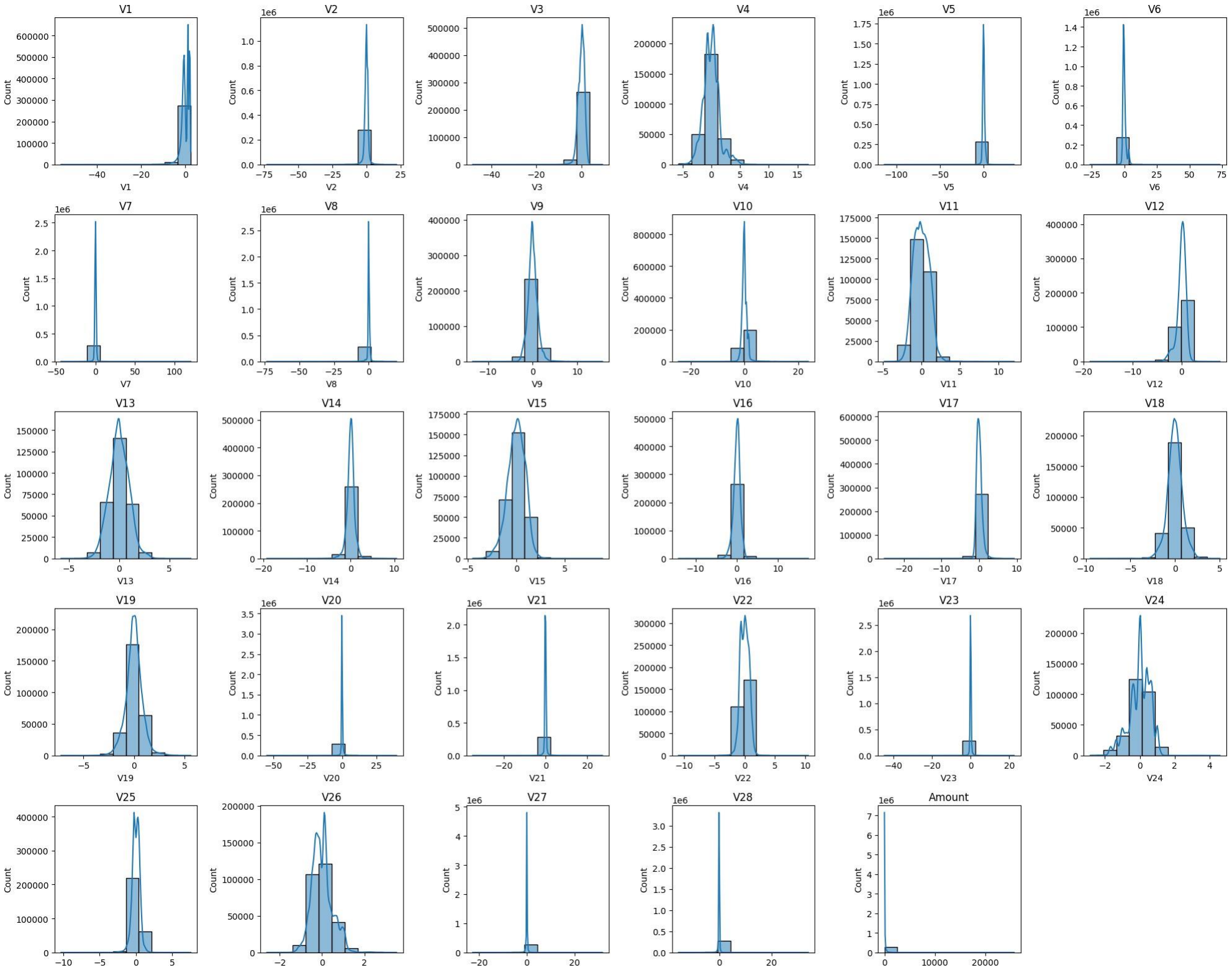
```
['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V 25', 'V26', 'V27', 'V28', 'Amount', 'Class']
```

In [11]:
```
dict1 = {}
for i in L:
    # Calculate upper and lower outliers
    q1 = df[i].quantile(0.25) q3 =
    df[i].quantile(0.75) iqr = q3 -
    q1
    upper_outliers = df[df[i] > (q3 + 1.5 * iqr)] lower_outliers
    = df[df[i] < (q1 - 1.5 * iqr)]

    # Count upper and lower outliers
    count_upper_outliers = len(upper_outliers) count_lower_outliers =
    len(lower_outliers)
    dict1[i] = count_upper_outliers, count_lower_outliers

print(dict1)
```

```
{'Time': (0, 0), 'V1': (0, 7062), 'V2': (5096, 8430), 'V3': (20, 3343), 'V4': (8905, 2243), 'V5': (8411, 3884), 'V6': (21213, 1752), 'V7': (4138, 4810), 'V8': (11897, 122
37), 'V9': (5844, 2439), 'V10': (6128, 3368), 'V11': (661, 119), 'V12': (769, 14579), 'V13': (2229, 1139), 'V14': (5392, 8757), 'V15': (429, 2465), 'V16': (1663, 6521),
'V17': (6663, 757), 'V18': (3547, 3986), 'V19': (5142, 5063), 'V20': (19087, 8683), 'V21': (8059, 6438), 'V22': (391, 926), 'V23': (10371, 8170), 'V24': (136, 4638), 'V2
5': (1679, 3688), 'V26': (4867, 729), 'V27': (19619, 19544), 'V28': (11800, 18542), 'Amount': (31904, 0), 'Class': (492, 0)}
```

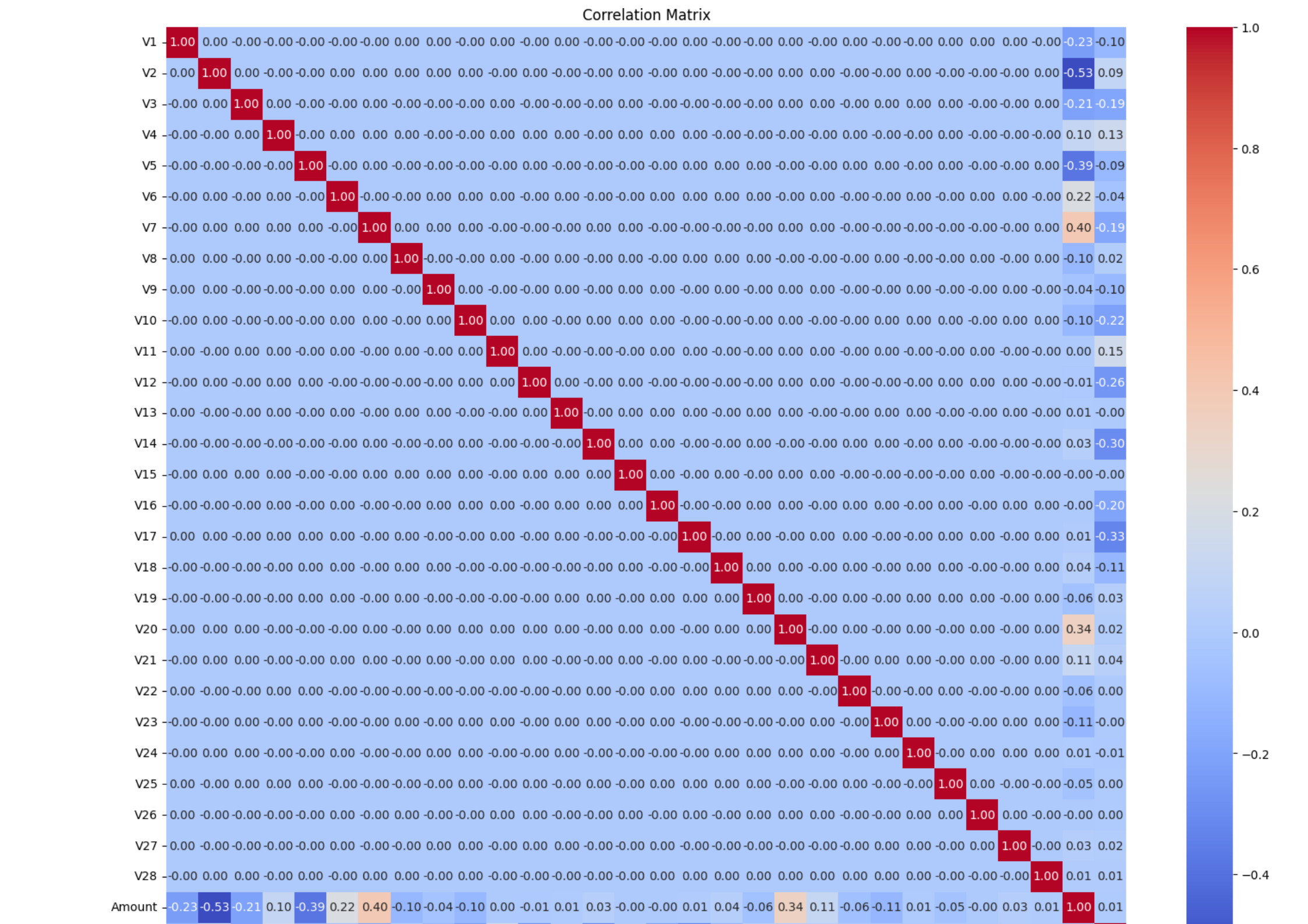
In [10]:

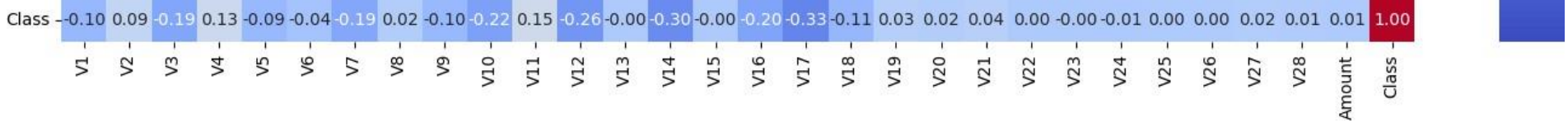
```
# Visualize distributions of numerical features
plt.figure(figsize=(20, 16))
for i, col in enumerate(data.columns[1:-1]): # Excluding 'Time' and 'Class' columns
    plt.subplot(5, 6, i+1)
    sns.histplot(data[col], bins=10, kde=True) plt.title(col)
plt.tight_layout() plt.show()
```


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
Count

In [11]:

```
# Visualize correlations among features
plt.figure(figsize=(18, 14))
corr = data.drop('Time', axis=1).corr()
sns.heatmap(corr, cmap='coolwarm', annot=True, fmt=".2f")
plt.title('Correlation Matrix')
plt.savefig('Correlation Natrix') plt.show()
```


Correlation Matrix
V1 1.00 0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 -0.23 -0.10
V2 0.00 1.00 0.00 -0.00 -0.00 0.00 0.00 0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.53 0.09
V3 -0.00 0.00 1.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 0.00 -0.00 -0.00 0.00 -0.21 -0.19
V4 -0.00 -0.00 0.00 1.00 -0.00 0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 0.10 0.13
V5 -0.00 -0.00 -0.00 -0.00 1.00 -0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.39 -0.09
V6 -0.00 0.00 -0.00 0.00 -0.00 1.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 0.22 -0.04
V7 -0.00 0.00 0.00 0.00 0.00 -0.00 1.00 0.00 0.00 0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.40 -0.19
V8 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 1.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.10 0.02
V9 0.00 0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 1.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.04 -0.10
V10 -0.00 0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 1.00 0.00 0.00 -0.00 0.00 0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 0.00 0.00 -0.10 -0.22
V11 0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 1.00 0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 0.15
V12 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 1.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 -0.01 -0.26
V13 0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 1.00 -0.00 0.00 0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 0.01 -0.00
V14 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 1.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 0.03 -0.30
V15 -0.00 0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 0.00 -0.00 0.00 0.00 0.00 1.00 0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 -0.00
V16 -0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 0.00 0.00 0.00 1.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.20
V17 0.00 0.00 -0.00 0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 1.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 0.01 -0.33
V18 -0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 1.00 0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 0.04 -0.11
V19 -0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 0.00 1.00 0.00 -0.00 0.00 -0.00 0.00 0.00 0.00 -0.00 0.00 -0.06 0.03
V20 0.00 0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.00 0.00 0.00 0.00 0.00 0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 1.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 0.34 0.02
V21 -0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 1.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 0.00 0.11 0.04
V22 0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 0.00 0.00 0.00 0.00 0.00 -0.00 1.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.06 0.00
V23 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 1.00 0.00 -0.00 -0.00 0.00 0.00 -0.11 -0.00
V24 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 1.00 -0.00 0.00 0.00 0.00 0.01 -0.01
V25 0.00 -0.00 0.00 0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 -0.00 1.00 0.00 -0.00 -0.00 -0.05 0.00
V26 0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 0.00 0.00 1.00 0.00 -0.00 -0.00 0.00
V27 0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 0.00 1.00 -0.00 0.03 0.02
V28 -0.00 0.00 0.00 -0.00 0.00 -0.00 0.00 0.00 -0.00 0.00 -0.00 -0.00 0.00 -0.00 -0.00 0.00 0.00 -0.00 0.00 0.00 0.00 0.00 0.00 0.00 -0.00 -0.00 -0.00 1.00 0.01 0.01
Amount -0.23 -0.53 -0.21 0.10 -0.39 0.22 0.40 -0.10 -0.04 -0.10 0.00 -0.01 0.01 0.03 -0.00 -0.00 0.01 0.04 -0.06 0.34 0.11 -0.06 -0.11 0.01 -0.05 -0.00 0.03 0.01 1.00 0.01
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4



In [16]:

```
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import joblib
# Separate features and target variable
X = data.drop(columns=['Time', 'Class']) # Drop 'Time' and 'Class' columns
y = data['Class']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)

# Feature engineering: No feature engineering is applied in this example

# Dealing with imbalanced data using SMOTE
smote = SMOTE(random_state=42)
X_resampled_, y_resampled_ = smote.fit_resample(X_train, y_train)

# # Feature scaling
# scaler = StandardScaler()
# X_train_scaled = scaler.fit_transform(X_resampled_)

# X_test_scaled = scaler.transform(X_test)

# Check the shape of the resampled data
print("Shape of X_train_resampled:", X_resampled_.shape) print("Shape of
y_train_resampled:", y_resampled_.shape) #print("Shape of
X_test_scaled:", X_test_scaled.shape)
print("Shape of y_test:", y_test.shape)
#joblib.dump(model, 'Capstone_project\logistic_regression_model1.pkl')
```

```
Shape of X_train_resampled: (511760, 29) Shape of
y_train_resampled: (511760,)
Shape of y_test: (28481,)
```

In [17]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import joblib

model = LogisticRegression()

# Train the model on the training data
model.fit(X_resampled_, y_resampled_)
joblib.dump(model, 'Capstone_project\logistic_regression_model.pkl')

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred) print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, y_pred)) print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9792142129840946

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     28435
           1       0.07      0.89      0.12        46

    accuracy                           0.98     28481
   macro avg       0.53      0.94      0.56     28481
weighted avg       1.00      0.98      0.99     28481


Confusion Matrix:
[[27848   587]
 [    5    41]]

C:\Users\Bhautik\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1): STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html) Please also refer to the documentation for
alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) n_iter_i = _check_optimize_result(
```

In [18]:

```
from sklearn.ensemble import RandomForestClassifier
# Initialize Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model
rf_classifier.fit(X_resampled_, y_resampled_)

# Make predictions
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred) print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9996488887328394

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28435
           1       0.93      0.85      0.89        46

    accuracy                           1.00     28481
   macro avg       0.96      0.92      0.94     28481
weighted avg       1.00      1.00      1.00     28481
```

In [19]:

```
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[28432     3]
 [    7    39]]
```

In [40]:

```
import joblib

# Load the saved model
model = joblib.load('Capstone_project\logistic_regression_model.pkl')

# Assuming X_test_scaled contains the scaled test data
# Let's select 5 records for prediction
X_new = X_test_scaled[:5]

# Make predictions on the new data
predictions = model.predict(X_new)

# Print the predictions
print("Predictions for the 5 records:") print(predictions)
```

```
Predictions for the 5 records:
[1 0 0 0 0]
```

In [41]:

```
import joblib

# Load the trained model
scaler = joblib.load('random_forest.pkl')

# Assuming X_new_scaled contains the scaled new data
# Let's select 5 records for prediction
X_new_scaled = scaler.transform(X_test[:5])

# Make predictions on the new data
predictions = model.predict(X_new_scaled)

# Print the predictions
print("Predictions for the 5 records:") print(predictions)
```

```
Predictions for the 5 records:
[1 0 0 0 0]
```

In [ ]: