CREDIT CARD FRAUD DETECTION

DATA COLLECTION AND DATA PREPARATION

Import Libraries

In [1]:
```
#import library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine
from sklearn.preprocessing import StandardScaler
```

Database connection and load data

In [2]:
```
#create SQLAlchemy engine
engine = create_engine('postgresql+psycopg2://postgres:admin@localhost:5432/fraud_detectionDB')
```

Deliverable: Data successfully stored in PostgreSQL, schema applied

In [3]:
```
# read data into pandas
df = pd.read_sql_query("SELECT * FROM cc_data", engine)
```

In [4]:
```
# View top rows
print(df.head())
```

```
   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]
```

Data Profiling

In [5]:
```
print("Dataset Information")
print(df.info())
```

```
Dataset Information
<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
None
```

In [6]:
```
# List all columns in the dataset.
print("Columns in Dataset")
print(df.columns)
```

```
Columns in Dataset
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')
```

In [11]:
```
print("Data shape:", df.shape)
```

```
Data shape: (284807, 31)
```

In [12]:
```
# Count and print missing values per column
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
```

Data Cleaning

In [13]:
```python
# Check unique values & class balance

print("Unique values in 'Class':", df['class'].unique())
print("Class distribution (counts):")
print(df['class'].value_counts())
print("Class distribution (percentage):")
print(df['class'].value_counts(normalize=True) * 100)
```

```
Unique values in 'Class': [0 1]
Class distribution (counts):
class
0    284315
1       492
Name: count, dtype: int64
Class distribution (percentage):
class
0    99.827251
1     0.172749
Name: proportion, dtype: float64
```

RESULT

So there are 284,315 normal transactions and 492 fraud transaction. The dataset is heavily imbalanced.

In [48]:
```python
# Simple textual report
# --------------------------
fraud_count = df['class'].value_counts()[1]
normal_count = df['class'].value_counts()[0]

print("Quick Report Summary:")
print(f"Total transactions: {len(df)}")
print(f"Normal transactions: {normal_count}")
print(f"Fraud transactions: {fraud_count}")
print(f"Fraud Rate: {100 * fraud_count / len(df):.4f}%")
```

```
Quick Report Summary:
Total transactions: 284807
Normal transactions: 284315
Fraud transactions: 492
Fraud Rate: 0.1727%
```

Data Validation

In [50]:
```python
# Data validation on SQL and Python
df = pd.read_sql("SELECT COUNT(*) FROM cc_data", engine)
print("SQL:", df)
print(f" Row count validation passed: {df.shape[0]} rows match SQL.")
```

```
SQL:     count
0  284807
 Row count validation passed: 1 rows match SQL.
```

Report Genration

1. Dataset Identification ✅
2. Data Import into SQL ✅
3. Normalization into relational tables : our table is flat table. No normalization. As data didn't require normalization as each row is an independent transaction. ✅
4. Initial SQL Profiling (SQL+Python) ✅
5. Data Cleaning & Transformation (Python): Nulls (none found), Duplicates (kept fraud class and non-fraud duplicates), Cleaning on Task 3 ✅
6. Data Validation ✅
7. Final Deliverable : Cleaned dataset on task 3 ✅

EXPLORATORY DATA ANALYSIS AND VISUALIZATION

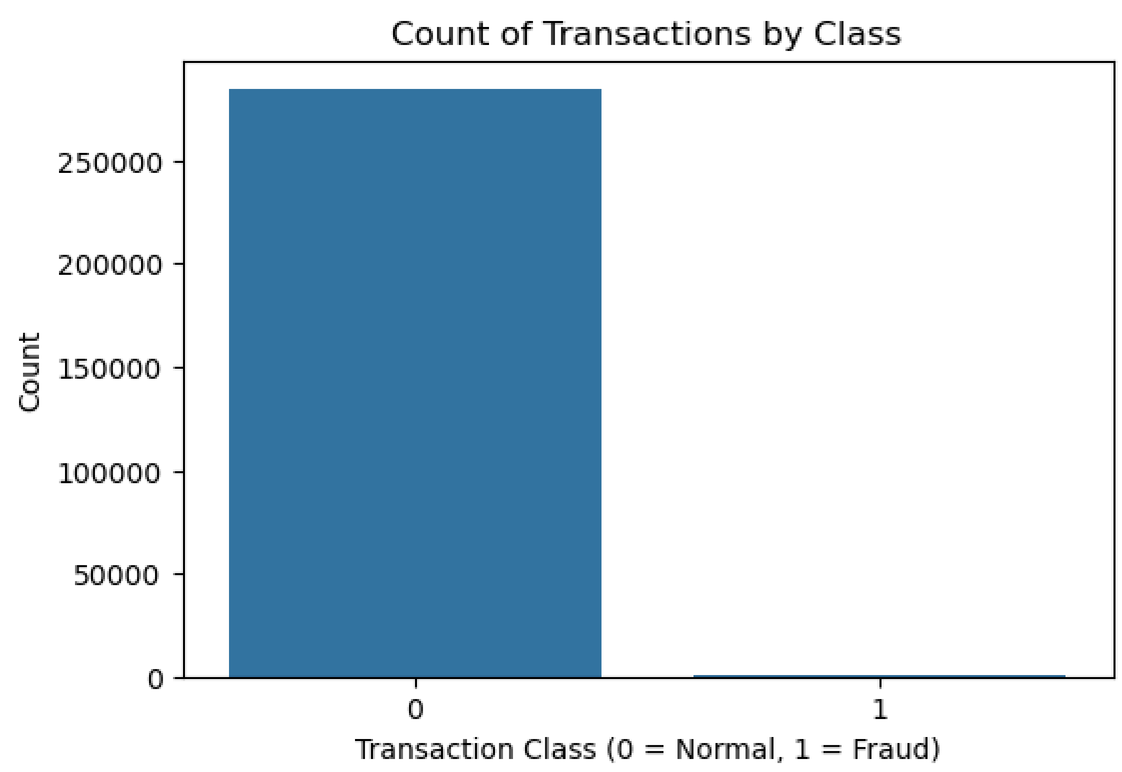
Summary Statistics

In [14]:
```python
# Explorartory data analysis

print("Statistical summary:")
print(df.describe())
```

```
Statistical summary:
                time            v1            v2            v3            v4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 v5            v6            v7            v8            v9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.205498e-16 -2.406306e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           v21           v22           v23           v24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.656562e-16 -3.568593e-16  2.610582e-16  4.473066e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                v25           v26           v27           v28         amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   5.213180e-16  1.683537e-15 -3.659966e-16 -1.223710e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

Univariate Analysis

In [15]:
```python
# Distribution of normal transaction and fraud transaction

plt.figure(figsize=(6, 4))
sns.countplot(x='class', data=df)
plt.title('Count of Transactions by Class')
plt.xlabel('Transaction Class (0 = Normal, 1 = Fraud)')
plt.ylabel('Count')
plt.show()
```



In [16]:
```python
# Count and print unique transaction amounts
print(df["amount"].value_counts())
```

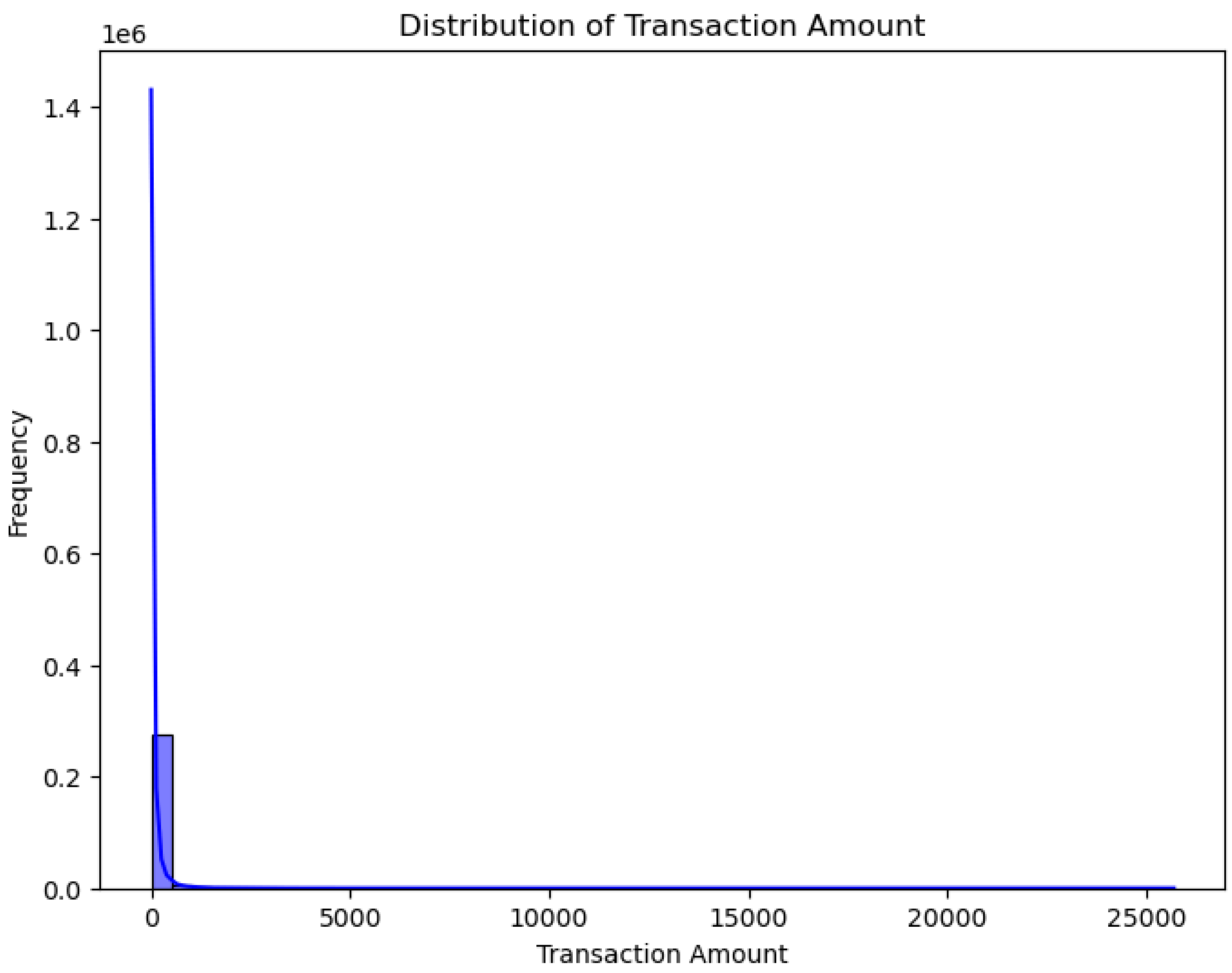
```
amount
1.00      13688
1.98       6044
0.89       4872
9.99       4747
15.00      3280
          ...
202.24        1
252.85        1
615.52        1
180.93        1
807.48        1
Name: count, Length: 32767, dtype: int64
```

In [17]:
```python
# Show percentage distribution of transaction amounts
df["amount"].value_counts(normalize=True)
```

Out[17]:
```
amount
1.00      0.048061
1.98      0.021221
0.89      0.017106
9.99      0.016667
15.00     0.011517
            ...
202.24    0.000004
252.85    0.000004
615.52    0.000004
180.93    0.000004
807.48    0.000004
Name: proportion, Length: 32767, dtype: float64
```

In [18]:
```python
# Distribution of transaction Amount

plt.figure(figsize=(8, 6))
```

```python
sns.histplot(df['amount'], bins=50, kde=True, color='blue')
plt.title('Distribution of Transaction Amount')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.show()
```



In [19]:
```python
# Count and print unique transaction times (duration)
print(df["time"].value_counts())
```

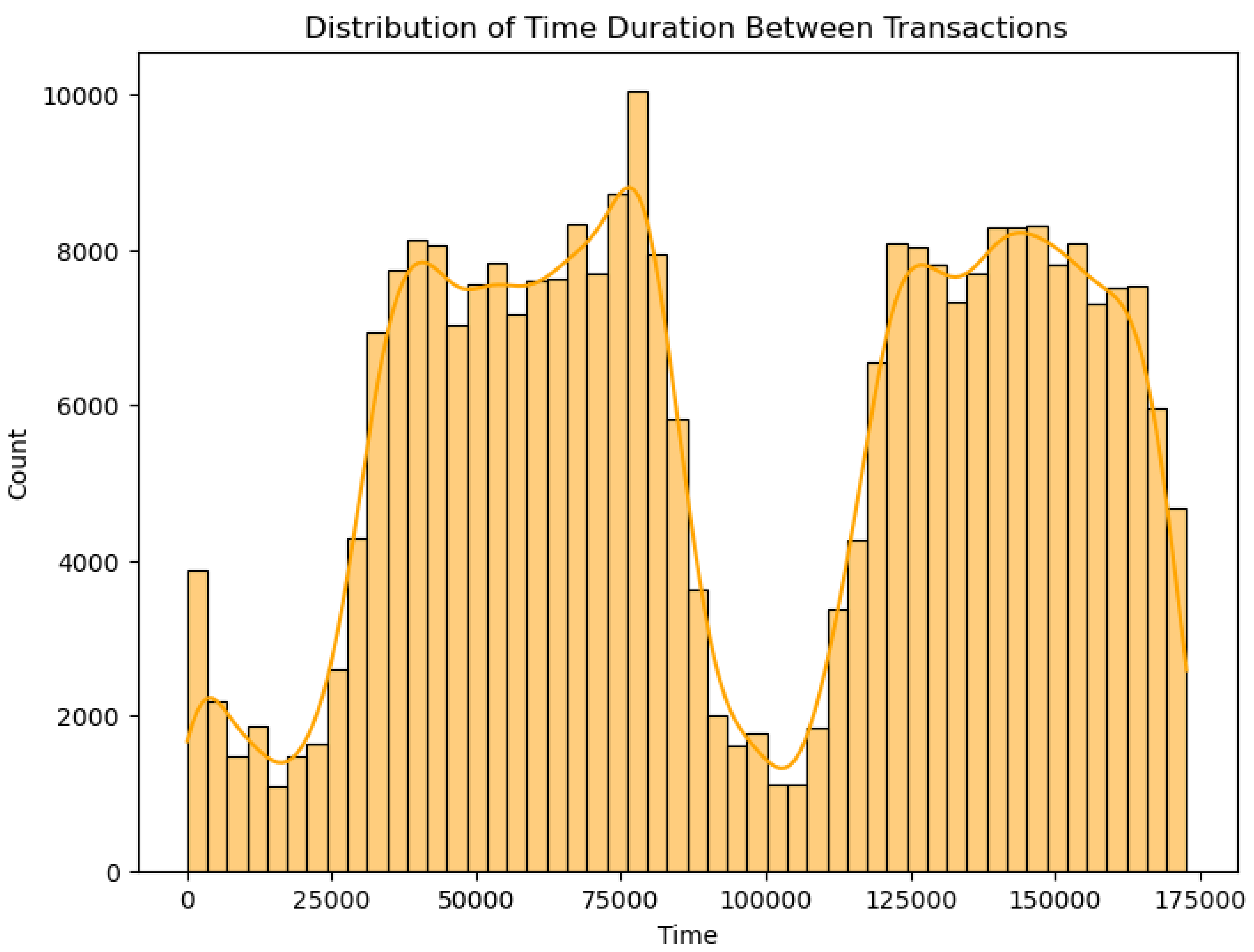
```
time
163152.0    36
64947.0     26
68780.0     25
3767.0      21
3770.0      20
            ..
172760.0     1
172758.0     1
172757.0     1
172756.0     1
172754.0     1
Name: count, Length: 124592, dtype: int64
```

In [20]:
```python
# Show percentage distribution of transaction times (duration)
df["time"].value_counts(normalize=True)
```

Out[20]:
```
time
163152.0    0.000126
64947.0     0.000091
68780.0     0.000088
3767.0      0.000074
3770.0      0.000070
              ...
172760.0    0.000004
172758.0    0.000004
172757.0    0.000004
172756.0    0.000004
172754.0    0.000004
Name: proportion, Length: 124592, dtype: float64
```

In [21]:
```python
# Distribution of Time duration between transactions

plt.figure(figsize=(8,6))
sns.histplot(df["time"], bins=50, kde=True, color="orange")
plt.title("Distribution of Time Duration Between Transactions")
plt.xlabel("Time")
plt.ylabel("Count")
```

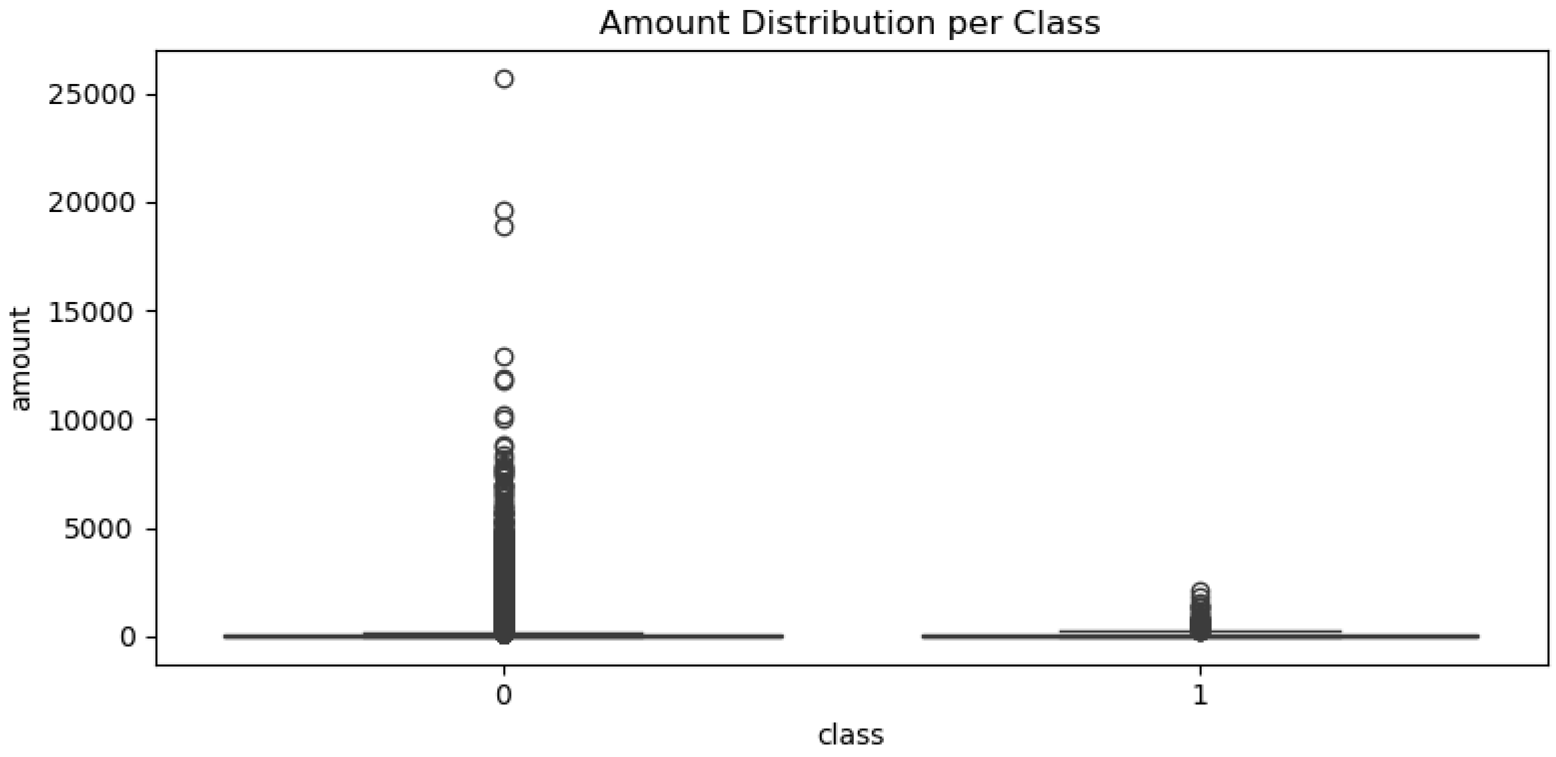
Out[21]:
```
Text(0, 0.5, 'Count')
```

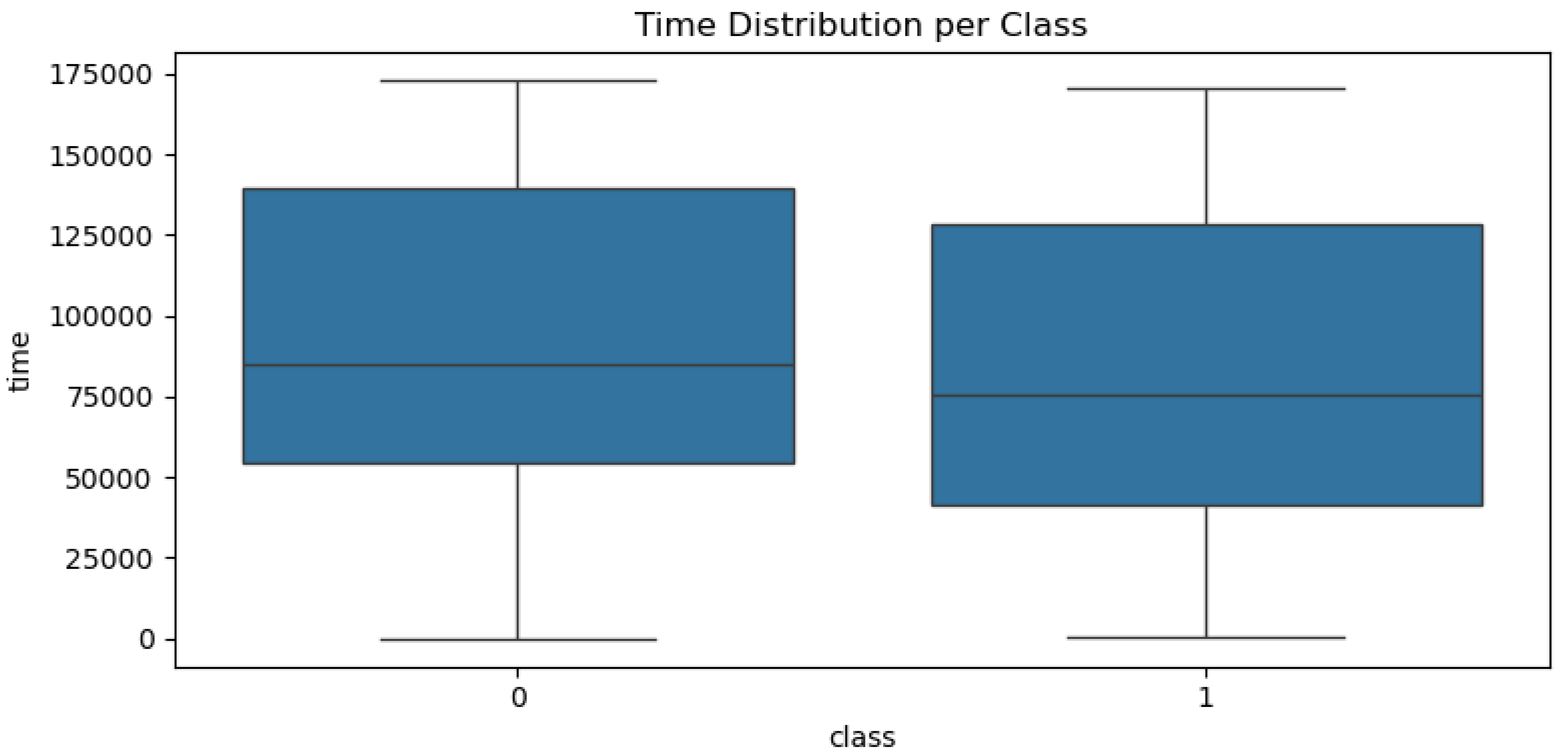


Bivariate/ Multivariate Analysis

In [23]:
```python
# Distribution of Time and Amount per Class (0=Normal, 1=fraud case)

# Boxplot Figure 1: Amount vs Class

plt.figure(figsize=(8,4))
sns.boxplot(x="class", y="amount", data=df)
plt.title("Amount Distribution per Class")
```

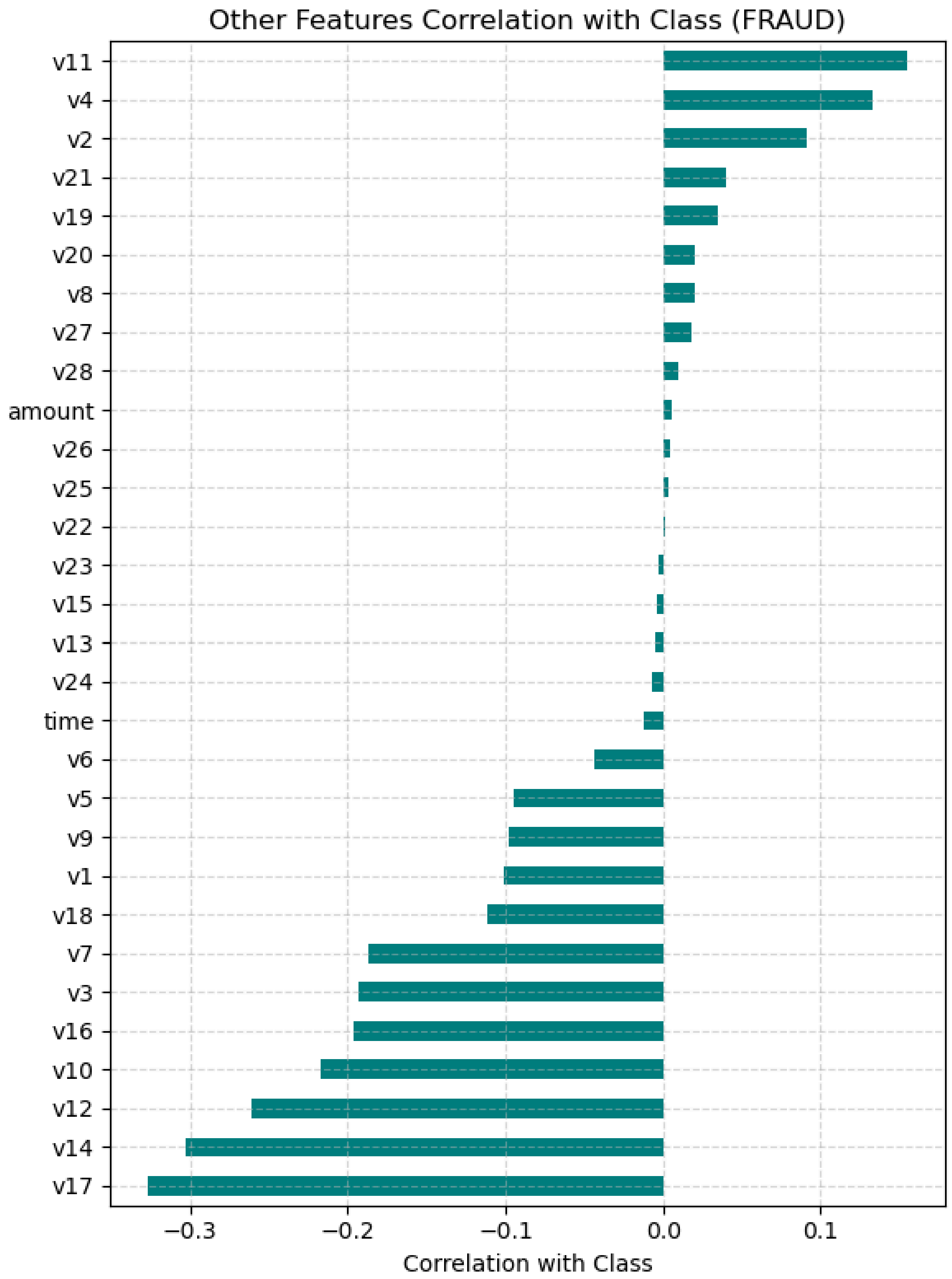
```python
plt.tight_layout()
plt.show()

# Boxplot Figure 2: Time vs Class

plt.figure(figsize=(8,4))
sns.boxplot(x="class", y="time", data=df)
plt.title("Time Distribution per Class")
plt.tight_layout()
plt.show()
```





In [19]:

```python
# Correlation between Features and Class 1 (Fraud)

correlations = df.corr()["class"].drop("class").sort_values()

plt.figure(figsize=(6, 8))
correlations.plot(kind="barh", color="teal")

plt.title("Other Features Correlation with Class (FRAUD)")
plt.xlabel("Correlation with Class")
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



DATA PREPARATION

In [27]:

```python
# 1.Handling missing values.

df.isnull().sum()
```

```
Out[27]: 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 [28]:
```python
# 2.Handling duplicated values.

# Show all rows that are duplicates (keeping and showing all occurrences)
df[df.duplicated(keep=False)]
```

Out[28]:

	time	v1	v2	v3	v4	v5	v6	v7	v8	v9	...	v21	v22	v23	v24	v25	v26	v27	v28	amount
32	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	...	0.046949	0.208105	-0.185548	0.001031	0.098816	-0.552904	-0.073288	0.023307	6.14
33	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	...	0.046949	0.208105	-0.185548	0.001031	0.098816	-0.552904	-0.073288	0.023307	6.14
34	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	...	0.049526	0.206537	-0.187108	0.000753	0.098117	-0.553471	-0.078306	0.025427	1.77
35	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	...	0.049526	0.206537	-0.187108	0.000753	0.098117	-0.553471	-0.078306	0.025427	1.77
112	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	...	0.102520	0.605089	0.023092	-0.626463	0.479120	-0.166937	0.081247	0.001192	1.18
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
283485	171627.0	-1.457978	1.378203	0.811515	-0.603760	-0.711883	-0.471672	-0.282535	0.880654	0.052808	...	0.284205	0.949659	-0.216949	0.083250	0.044944	0.639933	0.219432	0.116772	11.93
284190	172233.0	-2.667936	3.160505	-3.355984	1.007845	-0.377397	-0.109730	-0.667233	2.309700	-1.639306	...	0.391483	0.266536	-0.079853	-0.096395	0.086719	-0.451128	-1.183743	-0.222200	55.66
284191	172233.0	-2.667936	3.160505	-3.355984	1.007845	-0.377397	-0.109730	-0.667233	2.309700	-1.639306	...	0.391483	0.266536	-0.079853	-0.096395	0.086719	-0.451128	-1.183743	-0.222200	55.66
284192	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.745886	2.325616	-1.634651	...	0.402639	0.259746	-0.086606	-0.097597	0.083693	-0.453584	-1.205466	-0.213020	36.74
284193	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.745886	2.325616	-1.634651	...	0.402639	0.259746	-0.086606	-0.097597	0.083693	-0.453584	-1.205466	-0.213020	36.74

1854 rows × 31 columns

In [29]:
```python
# Show duplicates based only on Time, Amount, and Class features

df[df.duplicated(subset=["time", "amount", "class"], keep=False)]
```

Out[29]:

	time	v1	v2	v3	v4	v5	v6	v7	v8	v9	...	v21	v22	v23	v24	v25	v26	v27	v28	amo
32	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	...	0.046949	0.208105	-0.185548	0.001031	0.098816	-0.552904	-0.073288	0.023307	[illegible]
33	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	...	0.046949	0.208105	-0.185548	0.001031	0.098816	-0.552904	-0.073288	0.023307	[illegible]
34	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	...	0.049526	0.206537	-0.187108	0.000753	0.098117	-0.553471	-0.078306	0.025427	[illegible]
35	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	...	0.049526	0.206537	-0.187108	0.000753	0.098117	-0.553471	-0.078306	0.025427	[illegible]
108	73.0	1.162281	1.248178	-1.581317	1.475024	1.138357	-1.020373	0.638387	-0.136762	-0.805505	...	-0.124012	-0.227150	-0.199185	-0.289757	0.776244	-0.283950	0.056747	0.084706	[illegible]
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284193	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.745886	2.325616	-1.634651	...	0.402639	0.259746	-0.086606	-0.097597	0.083693	-0.453584	-1.205466	-0.213020	[illegible]
284248	172273.0	-0.765414	1.343887	-0.306101	-0.645545	-0.067358	-1.172196	0.516073	0.342927	0.368227	...	-0.289752	-0.709882	0.173594	-0.064594	-0.420300	0.159895	0.330875	0.150175	[illegible]
284251	172273.0	2.061056	-0.077031	-1.068720	0.422266	-0.181192	-1.227747	0.160285	-0.314824	0.596385	...	-0.292782	-0.727558	0.345553	0.036287	-0.312041	0.196591	-0.073152	-0.060601	[illegible]
284328	172348.0	2.064806	0.008284	-2.226901	0.926502	1.119908	0.178604	0.349210	-0.010441	0.262333	...	0.006761	0.087820	-0.095232	-0.452246	0.532753	-0.468378	-0.036697	-0.079256	[illegible]
284329	172348.0	-1.351689	1.969541	-2.145252	-0.866654	0.438384	-0.124297	-0.245481	1.404284	-0.342847	...	-0.296305	-1.007138	0.104401	-0.523060	-0.148007	0.175197	0.068956	-0.026791	[illegible]

8736 rows × 31 columns

DATA CLEANING

In [30]:
```python
# Identify all rows that are duplicates (showing every occurrence)

all_duplicates = df[df.duplicated(keep=False)]
print("All Duplicate Rows (Full Duplicate Set):", all_duplicates.shape)
```

```
All Duplicate Rows (Full Duplicate Set): (1854, 31)
```

In [31]:
```python
# Identify duplicates based only on 'time', 'amount', and 'class'

partial_duplicates = df[df.duplicated(subset=["time", "amount", "class"], keep=False)]
print("Duplicates based on ['time', 'amount', 'class']:", partial_duplicates.shape)
```

```
Duplicates based on ['time', 'amount', 'class']: (8736, 31)
```

In [32]:
```python
# Count total number of strictly duplicated rows (ignores first appearance)

strict_duplicate_count = df.duplicated().sum()
print("Strict Duplicate Count (excluding firsts):", strict_duplicate_count)
```

```
Strict Duplicate Count (excluding firsts): 1081
```

In [33]:
```python
# Count total number of duplicated rows including all repeated instances

full_duplicate_count = df.duplicated(keep=False).sum()
print("Full Duplicate Count (all duplicates marked):", full_duplicate_count)
```

```
Full Duplicate Count (all duplicates marked): 1854
```

In [34]:
```python
# Number of unique duplicate patterns (dropping repeats among duplicates)
```

```
unique_duplicate_patterns = df[df.duplicated(keep=False)].drop_duplicates().shape[0]
print("Unique Duplicate Patterns (after dropping repeated copies):", unique_duplicate_patterns)
```

```
Unique Duplicate Patterns (after dropping repeated copies): 773
```

In [35]:
```
# Class-wise breakdown of the duplicate rows

class_distribution_among_duplicates = df[df.duplicated(keep=False)]["class"].value_counts()
print("Class distribution among duplicates:\n", class_distribution_among_duplicates)
```

```
Class distribution among duplicates:
 class
0    1822
1      32
Name: count, dtype: int64
```

Observation

There are 1,822 duplicats on Class 0 (normal) and 32 duplicats on Class 1 (fraud).

Since Class 1 has few duplicats, I decided not too drop the duplicats.

I will only drop the duplicats from Class 0.

In [36]:
```
# Separate data into fraud and normal classes
fraud = df[df["class"] == 1]
normal = df[df["class"] == 0]

# Drop duplicates from the normal class only (retain all fraud cases)
normal_cleaned = normal.drop_duplicates()
```

In [37]:
```
# Combine cleaned normal data with unaltered fraud data
df_cleaned = pd.concat([fraud, normal_cleaned], ignore_index=True)

# Check number of duplicates remaining in the cleaned dataset
remaining_duplicates = df_cleaned.duplicated().sum()
print("Remaining Duplicates in Cleaned Data:", remaining_duplicates)
```

```
Remaining Duplicates in Cleaned Data: 19
```

In [38]:
```
# Class distribution among these remaining duplicates

remaining_duplicate_classes = df_cleaned[df_cleaned.duplicated(keep=False)]["class"].value_counts()
print("Class distribution in remaining duplicates:\n", remaining_duplicate_classes)
```

```
Class distribution in remaining duplicates:
 class
1    32
Name: count, dtype: int64
```

In [41]:
```
# Final class breakdown (percentage-wise) after cleaning

final_class_distribution = df_cleaned["class"].value_counts(normalize=True) * 100
print("Class percentage after cleaning:\n", final_class_distribution)
```

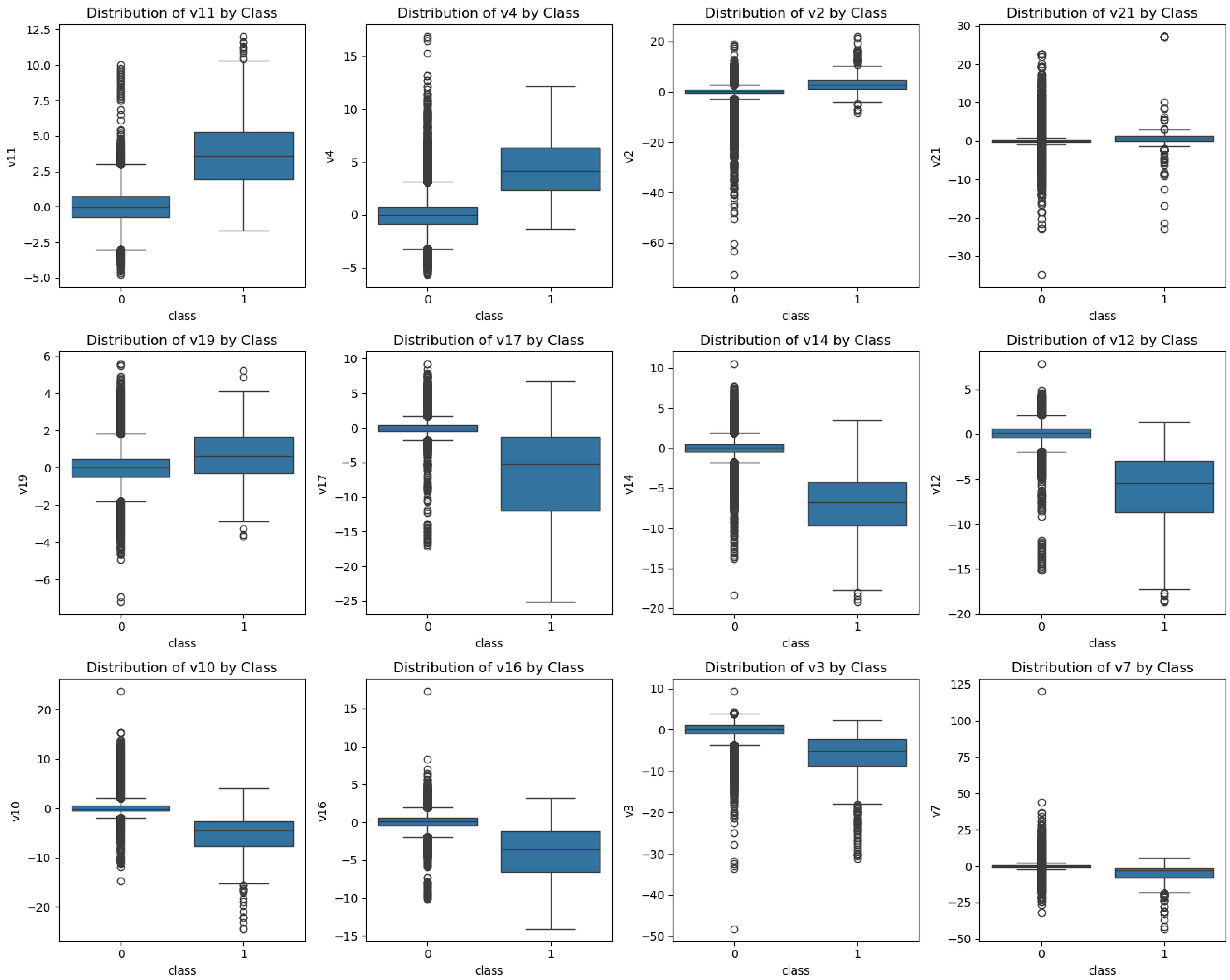
```
Class percentage after cleaning:
 class
0    99.826605
1     0.173395
Name: proportion, dtype: float64
```

Result

- Preserved all fraud records (even duplicates)
- Dropped duplicates only from the normal class (Class 0)
- Increased integrity without distorting fraud signals

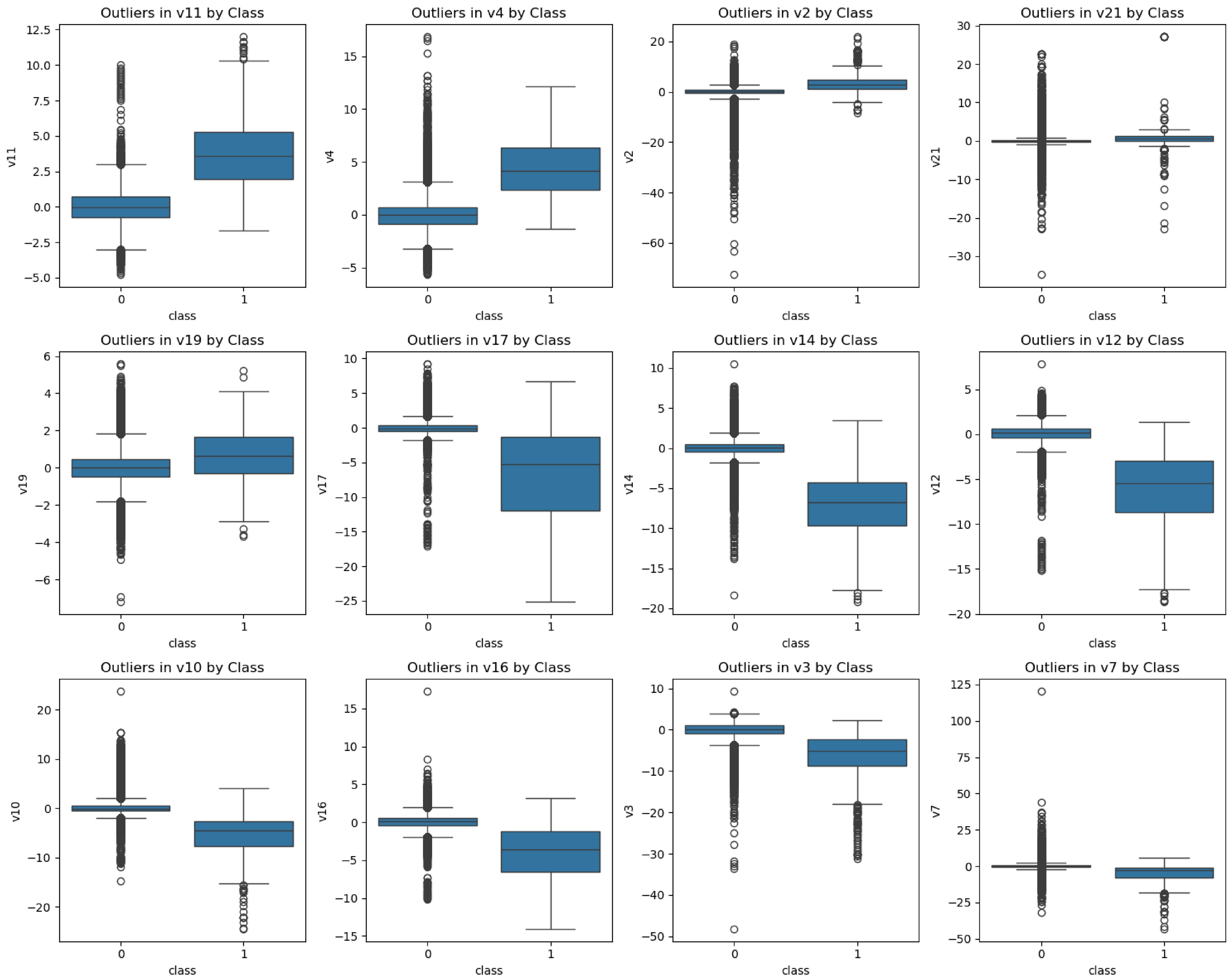
Outliers and Anomaly Detection

In [42]:
```
#CHECKING CLASS DISTRIBUTION:

# A. Feature Distribution between Fraud and Normal Transactions

# Selected features based on Strongest Positive and Strongest Negative correlations with Class

selected_features = ['v11', 'v4', 'v2', 'v21', 'v19', # Strong Positive
                     'v17', 'v14', 'v12', 'v10', 'v16', 'v3', 'v7']  # Strong Negative

plt.figure(figsize=(15, 12))
for i, feature in enumerate(selected_features, 1):
    plt.subplot(3, 4, i)
    sns.boxplot(data=df_cleaned, x='class', y=feature)
    plt.title(f'Distribution of {feature} by Class')
plt.tight_layout()
plt.show()
```



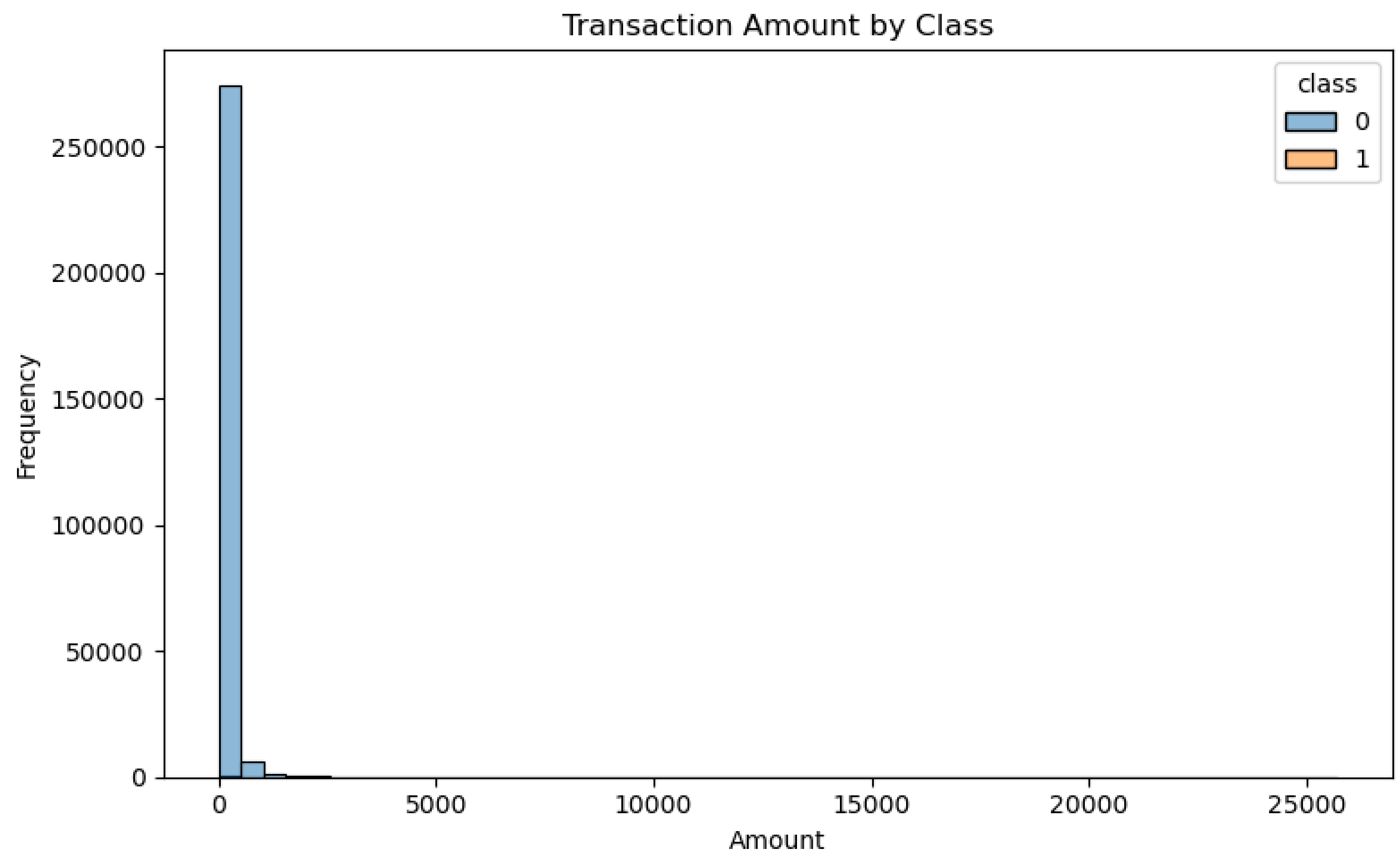
In [43]:

```python
# B. Outlier Visualization with Boxplots

plt.figure(figsize=(15, 12))
for i, feature in enumerate(selected_features, 1):
    plt.subplot(3, 4, i)
    sns.boxplot(data=df_cleaned, x='class', y=feature)
    plt.title(f'Outliers in {feature} by Class')
plt.tight_layout()
plt.show()
```



In [44]:

```
# C. Histogram for Transaction Amount

plt.figure(figsize=(8, 5))
sns.histplot(df_cleaned, x='amount', hue='class', bins=50)
plt.title('Transaction Amount by Class', fontsize=12)
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



RESULT

- I have decided not to drop any outliers.
- Outliers were retained because they may represent real fraudulent behavior, which is inherently anomalous.
- Removing them could reduce the model's ability to detect rare but significant fraud patterns.

Power BI Visualization

In [51]:

```
print("Cleaned Dataset Summary for Power BI")
print("—————————————————————————————————————")
print(f" Total Records After Cleaning: {df_cleaned.shape[0]}")
print(f" Total Columns: {df_cleaned.shape[1]}")
print(f" Normal Transactions: {df_cleaned['class'].value_counts().get(0, 0)}")
print(f" Fraud Transactions: {df_cleaned['class'].value_counts().get(1, 0)}")
print(f" Fraud Rate: {round(df_cleaned['class'].value_counts(normalize=True).get(1, 0) * 100, 4)}%")
```

```python
print(f" Missing Values: {df_cleaned.isnull().sum().sum()} (should be 0)")
print(f" Remaining Duplicates: {df_cleaned.duplicated().sum()}")
```

```
Cleaned Dataset Summary for Power BI
──────────────────────────────────────
 Total Records After Cleaning: 283745
 Total Columns: 31
 Normal Transactions: 283253
 Fraud Transactions: 492
 Fraud Rate: 0.1734%
 Missing Values: 0 (should be 0)
 Remaining Duplicates: 19
```

In [53]:

```python
# Export to CSV
df_cleaned.to_csv("cleaned_dataset.csv", index=False)
```

Power BI Summary

- **Total (raw)**: 284,807
- **Normal**: 284,315 → 🔻 283,254 (after cleaning)
- **Fraud**: 492 (unchanged)
- **Fraud Rate**: 0.1727% → 0.1734%
- **Avg. Amount**: 88.35
- **Max / Min Amount**: 25,691.16 / 0.00
- No missing values
- Duplicates: 1,854 (cleaned from normal only)
- Final Records: 283,746 ✅