Dataset Overview: Credit Card Transactions

The dataset contains **284,807 entries** and **31 columns**, representing credit card transactions.
Each row corresponds to a single transaction with various features.
The goal of the analysis is to **detect fraudulent transactions**.

Columns Overview

- **Time**:
  The elapsed time (in seconds) since the first transaction in the dataset.

- **V1 to V28**:
  These are **anonymized features** (represented as V1, V2, ..., V28) generated from **PCA (Principal Component Analysis)** transformation to ensure privacy.

  - We do not have direct information about the original variables.
  - These features are critical in distinguishing between **legitimate** and **fraudulent** transactions.
- **Amount**:
  The transaction amount for the credit card transaction.

- **Class**:
  The **target variable**:

  - `1` indicates a **fraudulent** transaction.
  - `0` indicates a **legitimate** transaction.

Key Note

- The columns **V1 to V28** are anonymized and derived features, meaning we do not have direct access to their original meanings.
  However, they are essential in enabling the model to distinguish fraud patterns.

- The **Class** column is the target variable used in training classification models:

  - `1` = Fraud
  - `0` = Legitimate
- This dataset is commonly used for building **classification models** to detect fraudulent transactions based on the provided features.

IMPORTING IMPORTANT LIBRARIES

In [44]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import plotly.express as px
import seaborn as sns
from pyspark.sql import SparkSession
```

A SMALL ETL PRROCESS

In [6]:
```
def ETL(filepath, output_path):
    spark = SparkSession.builder.appName("CreditCardFraudDetection").getOrCre

    df = spark.read.csv(filepath, header=True)

    df.createOrReplaceTempView("creditcard_table")

    cleandf = spark.sql(
        """
        select * from creditcard_table
        where Class is not Null
        """

    )

    clean_df.write.mode("overwrite").csv(output_path, header=True)


    spark.stop()


    return clean_df.toPandas()



def run_ETL():
    df = ETL("creditcard.csv", "updated.csv")
```

WEEKLY DAG

> assuming csv updating as it connects live from SQl, so as CSv updated it will every week

In [9]:
```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime

default_args = {
    'owner': 'airflow',
    'start_date': datetime(2025, 4, 26),
    'retries': 1,
}

dag = DAG(
    'creditcard_etl_dag',
    default_args=default_args,
    schedule_interval='@weekly',
```

```
    catchup=False,
)


etl_task = PythonOperator(
    task_id='run_creditcard_etl',
    python_callable=run_ETL,
    dag=dag,
)

etl_task
```

```
/var/folders/64/4lr7sg2s2hg6k9zng9qjnc380000gn/T/ipykernel_20229/1640569
/var/folders/64/4lr7sg2s2hg6k9zng9qjnc380000gn/T/ipykernel_20229/1640569
```

Out[9]:

```
<Task(PythonOperator): run_creditcard_etl>
```

Success: This box indicates a successful action.

Manually!

In [2]:

```
from pyspark.sql import SparkSession

def ETL(filepath):
    return (SparkSession.builder.appName("CCF_ETL").getOrCreate()
            .read.csv(filepath, header=True, inferSchema=True)
            .filter("Class IS NOT NULL")
            .toPandas())


df = ETL("creditcard.csv")
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLog
Level(newLevel).
25/05/01 19:30:40 WARN NativeCodeLoader: Unable to load native-hadoop librar
y for your platform... using builtin-java classes where applicable
25/05/01 19:30:40 WARN Utils: Service 'SparkUI' could not bind on port 4040.
Attempting port 4041.
25/05/01 19:30:44 WARN package: Truncated the string representation of a pla
n since it was too large. This behavior can be adjusted by setting 'spark.sq
l.debug.maxToStringFields'.
```

In [3]:

```
df.head(2)
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510

2 rows × 31 columns

In [6]:

```
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  int32  
dtypes: float64(30), int32(1)
memory usage: 66.3 MB
```

No missing values, it shows data is not raw! and as we seing V1-V28 it shows data has beeen processed

---

Outliers

In [143…

```
sns.set_style('whitegrid')
sns.set_palette('Dark2')
```

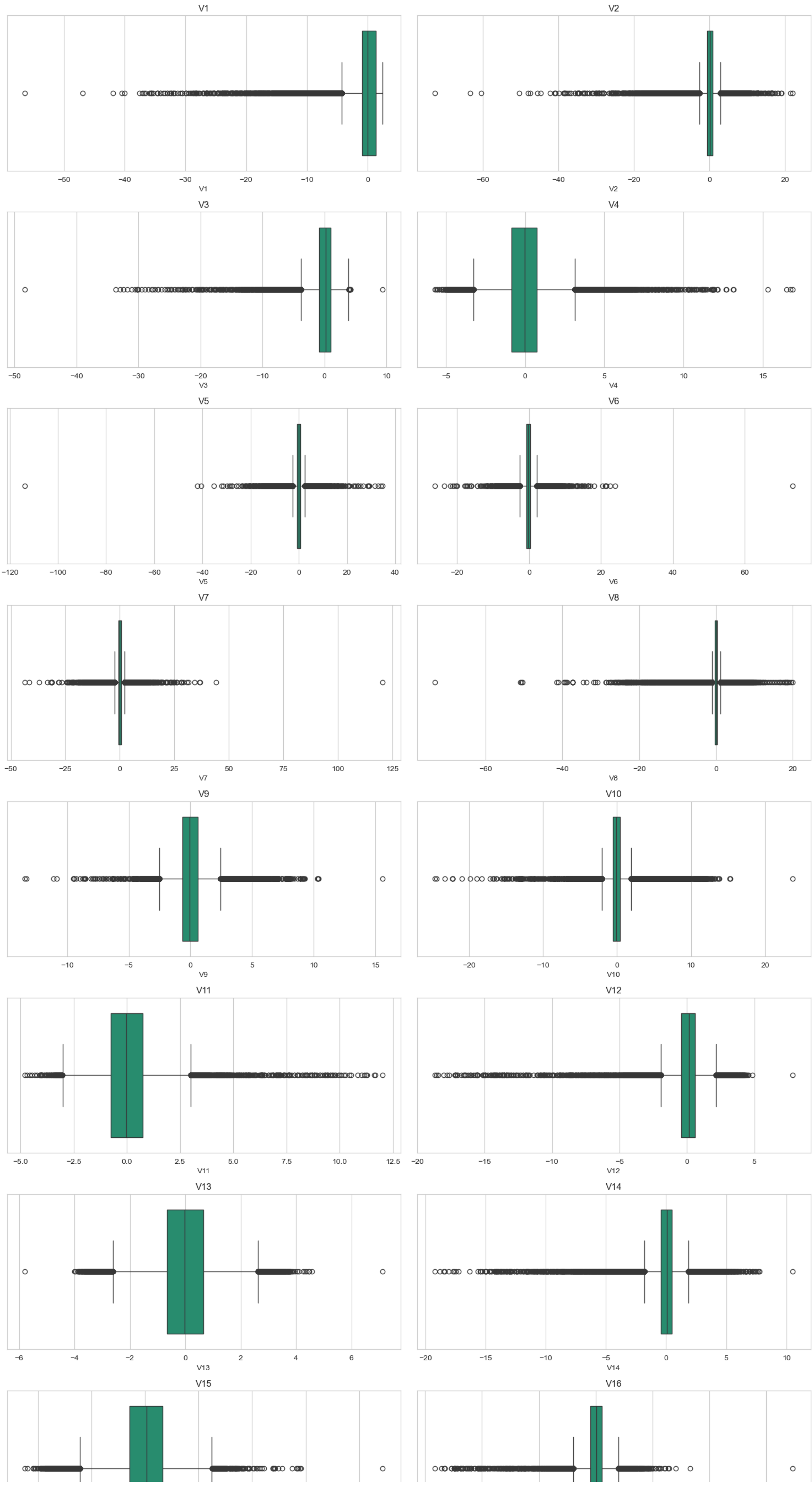
In [145…

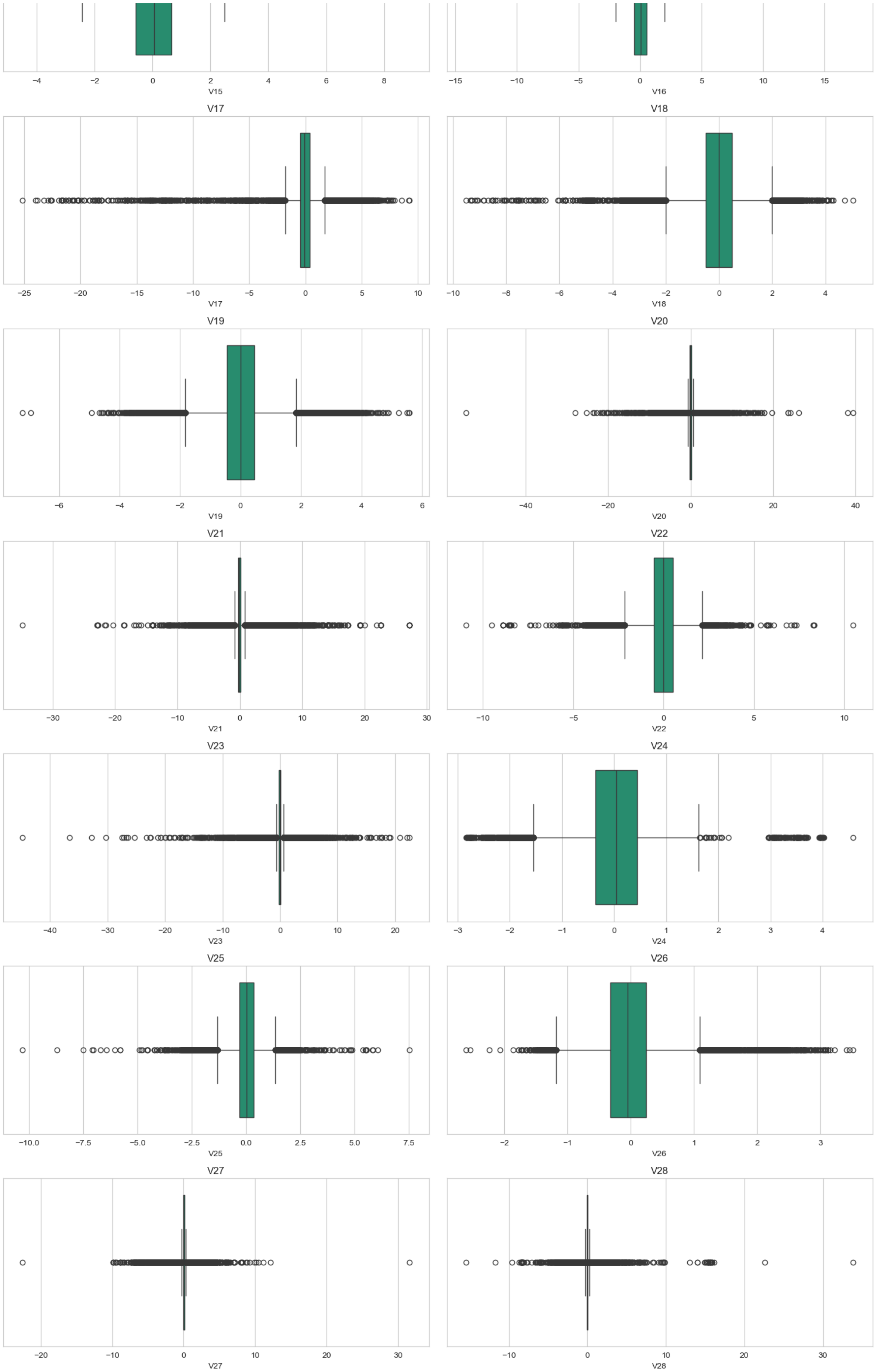
```
fig, axes = plt.subplots(nrows=14, ncols=2, figsize=(15, 50))
axes = axes.flatten()

v_cols = [f'V{i}' for i in range(1, 29)]

for i, col in enumerate(v_cols):
    sns.boxplot(x=df[col], ax=axes[i])
    axes[i].set_title(col)

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

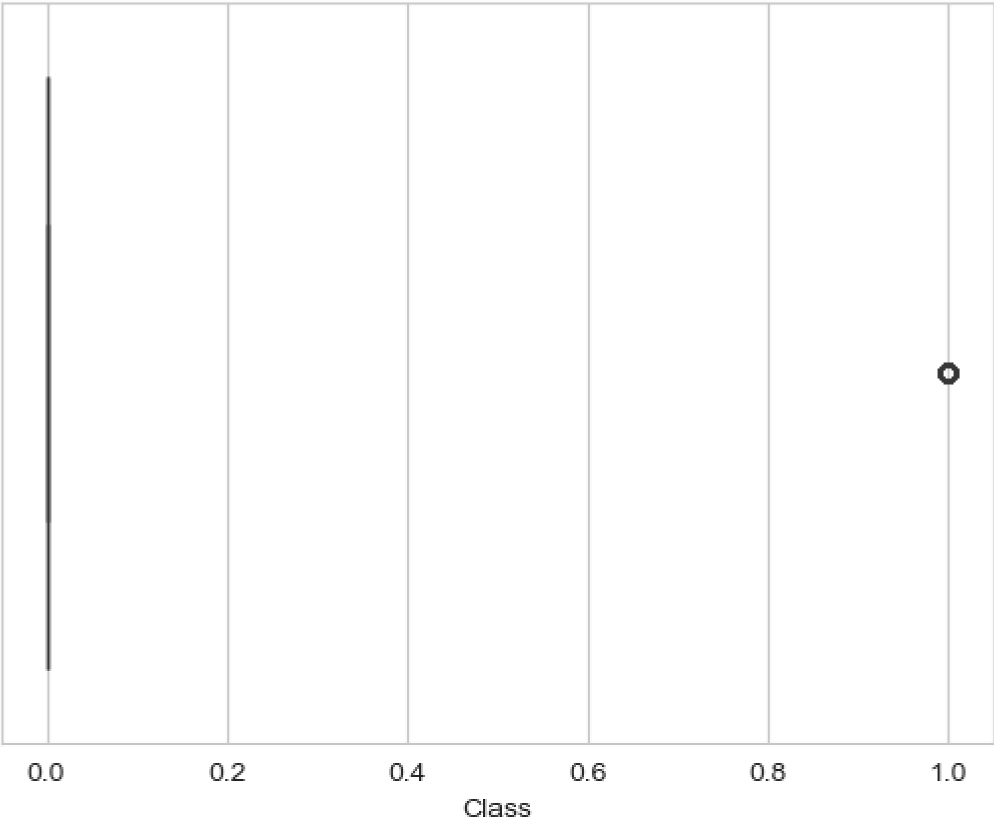

V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16



In [147…

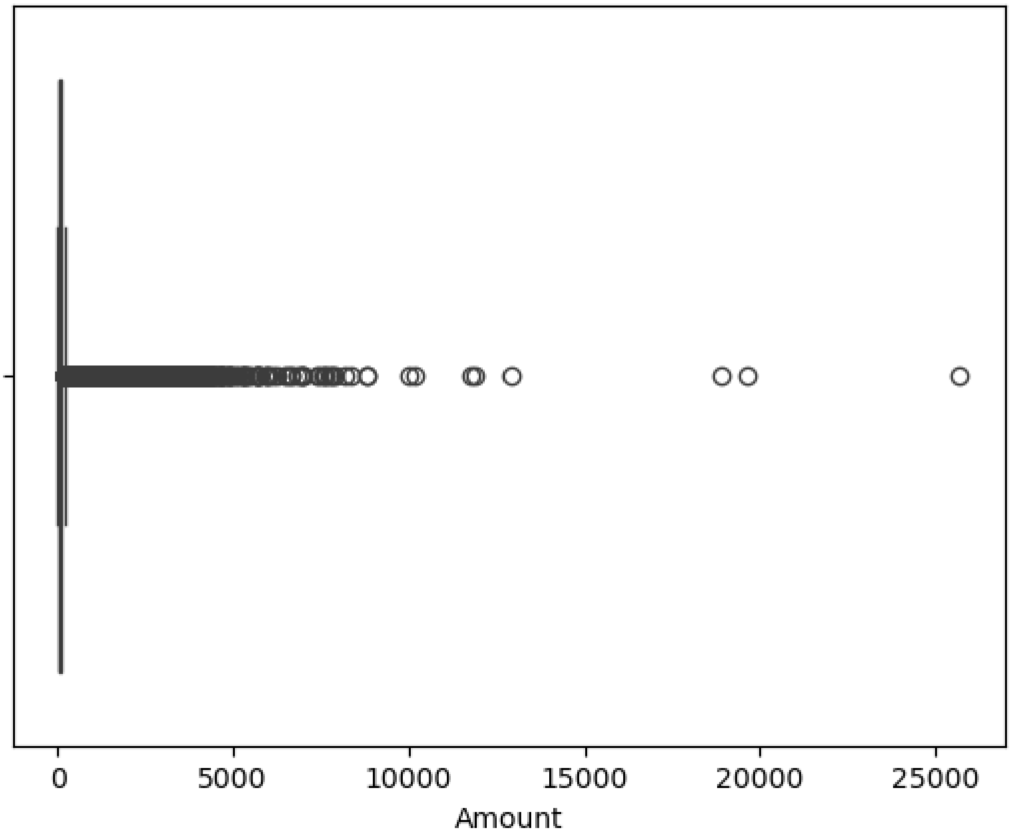
```
sns.boxplot(x=df['Class'])
```

Out[147]: <Axes: xlabel='Class'>



In [48]:
```
sns.boxplot(x=df['Amount'])
```

Out[48]: <Axes: xlabel='Amount'>



Warning:Since the data is classified (i.e., labeled for categories), removing outliers could distort class boundaries or important rare cases. Therefore, we will not remove outliers!

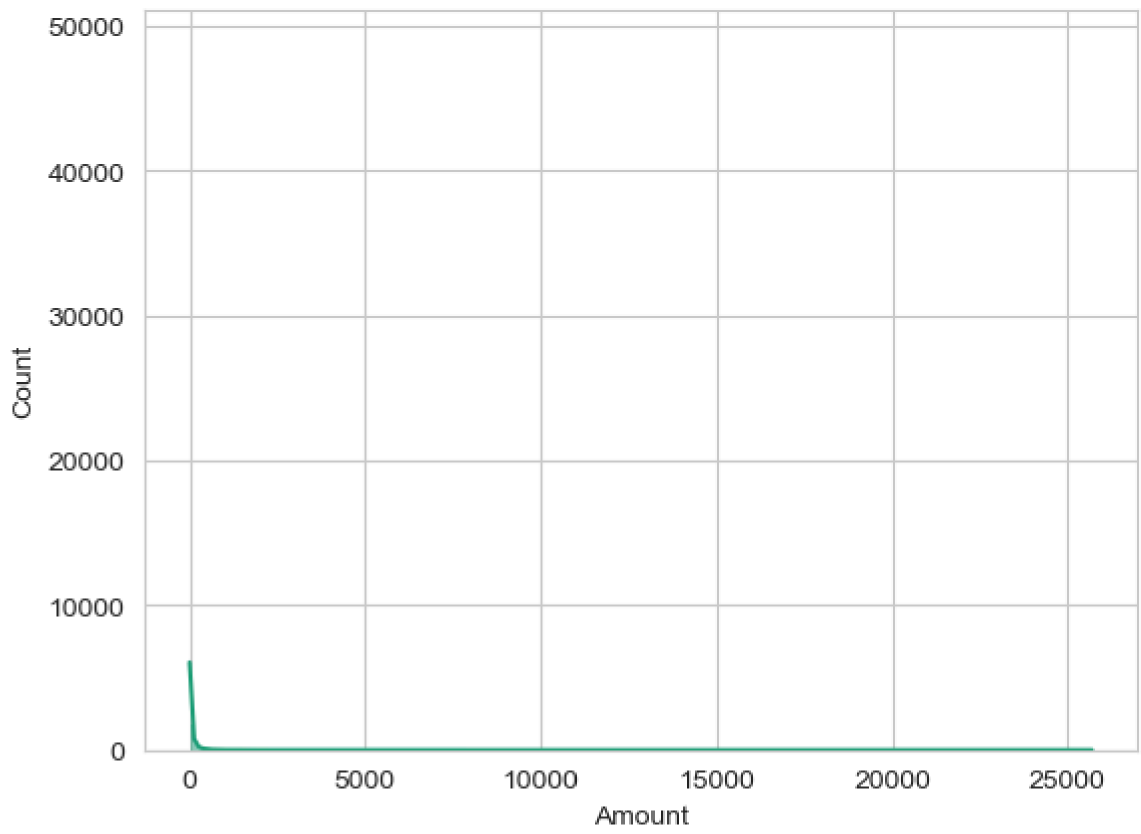
Checking Distribution and trends

In [52]:
```python
# %matplotlib widget
```

In [53]:
```python
sns.set_style('whitegrid')
sns.set_palette('Dark2')
```

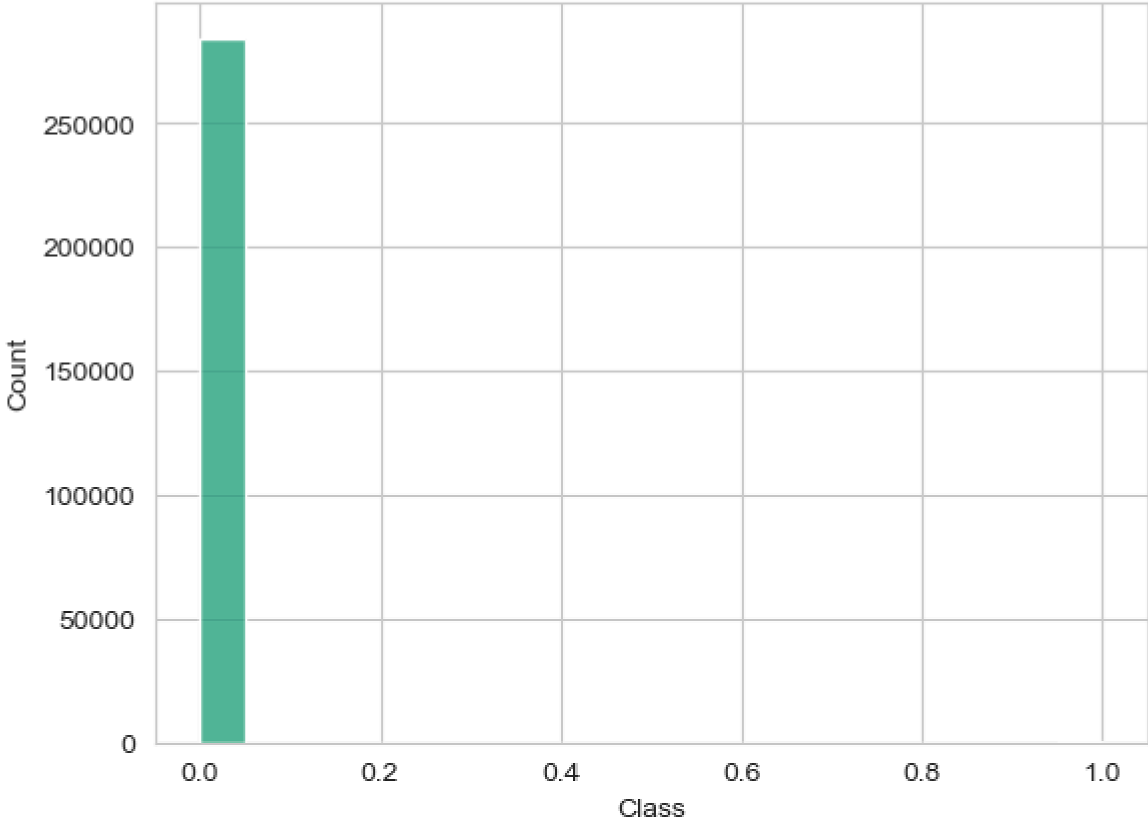
In [54]:
```python
sns.histplot(x=df['Amount'], kde=True)
```

Out[54]:
```
<Axes: xlabel='Amount', ylabel='Count'>
```



In [55]:
```python
sns.histplot(x=df['Class'])
```

Out[55]:
```
<Axes: xlabel='Class', ylabel='Count'>
```



Descriptive Statistics

Class - Fraudant

In [58]:
```
classdist = df.groupby('Class')['Time'].count()
print(f" The count non-Fraudent and Fraudent is {classdist.tolist()}")
classdist_percentage = (classdist / classdist.sum()) * 100
print(f" The Percentage difference between non-Fraudent and Fraudent is {cla
```

```
 The count non-Fraudent and Fraudent is [284315, 492]
 The Percentage difference between non-Fraudent and Fraudent is [99.82725143
693798, 0.1727485630620034]
```

so it's less than 1 percent who is fraudent!

Note: Have to Apply SMOTE while training

Amount

In [61]:
```
# Removing Outliers to remove effect of it on average
q3 = df['Amount'].quantile(0.75)
q1 = df['Amount'].quantile(0.25)
IQR = q3 - q1

lower_bound = q1 - 1.5 * IQR
upper_bound = q3 + 1.5 * IQR

Amount_wh_outliers = df[(df['Amount'] >= lower_bound) & (df['Amount'] <= upp
```

In [62]:
```
skewness = Amount_wh_outliers['Amount'].skew()
print(f"Skewness of 'Amount': {skewness}")
```

```
Skewness of 'Amount': 1.5750079526382468
```

since it's greater 0 so it means postive skewed so we have to use median instead of mean to average amount

In [64]:
```
print(F"the average amount is {Amount_wh_outliers['Amount'].median()}")
print(F"what if i don't remove outliers, the average amount will be {df['Amo
```

```
the average amount is 16.0
what if i don't remove outliers, the average amount will be 0        149.62
1           2.69
2         378.66
3         123.50
4          69.99
           ...
284802      0.77
284803     24.79
284804     67.88
284805     10.00
284806    217.00
Name: Amount, Length: 284807, dtype: float64
```

TIME

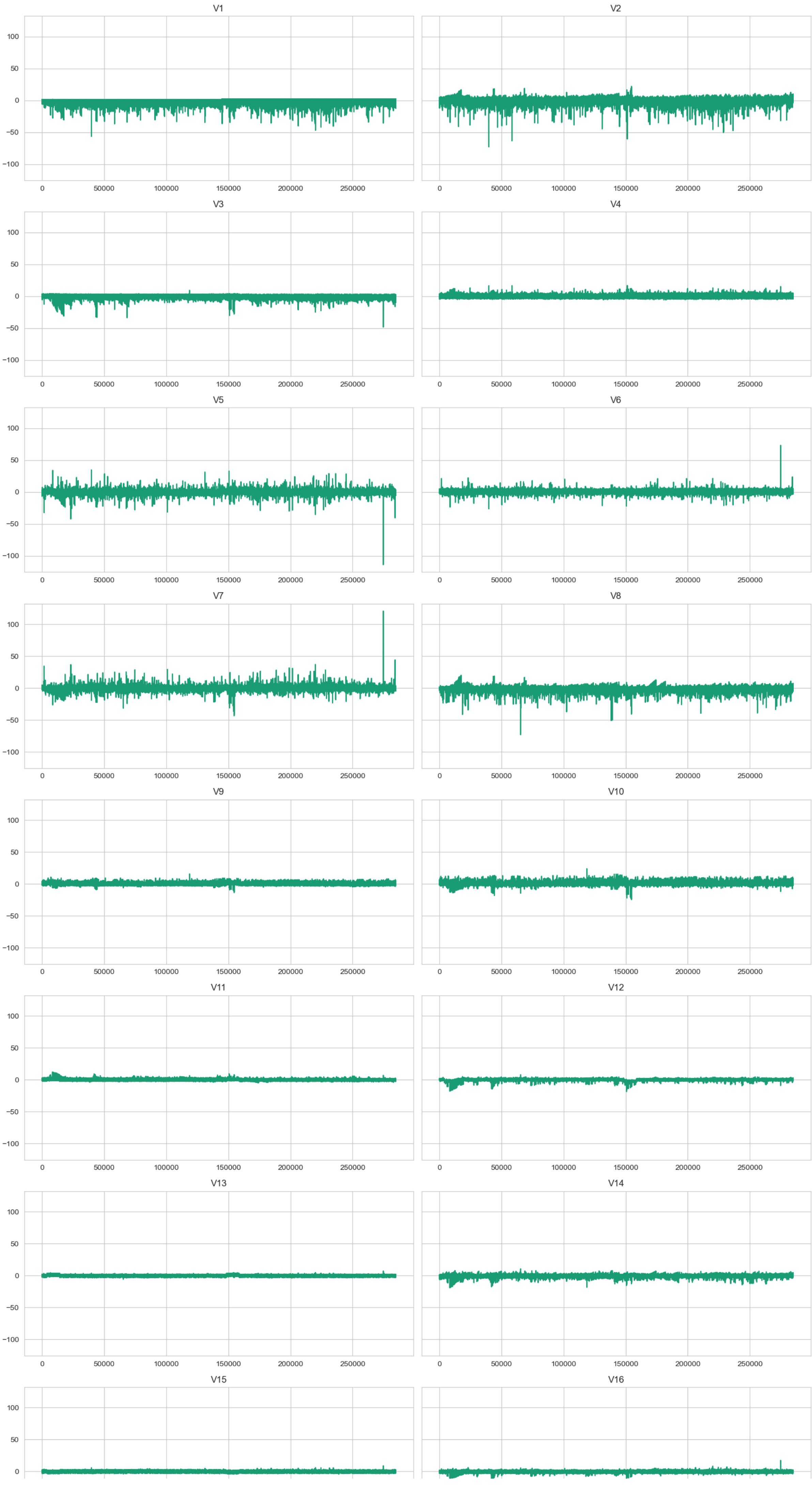
In [66]:
```
dff = df.copy()
dff['RollingMean'] = dff['Time'].rolling(window=5).mean()
dff['RollingMeanChange'] = dff['RollingMean'].diff()
average_change_per_row = dff['RollingMeanChange'].abs().mean()
print(f"Average change in Time is {average_change_per_row}")
```

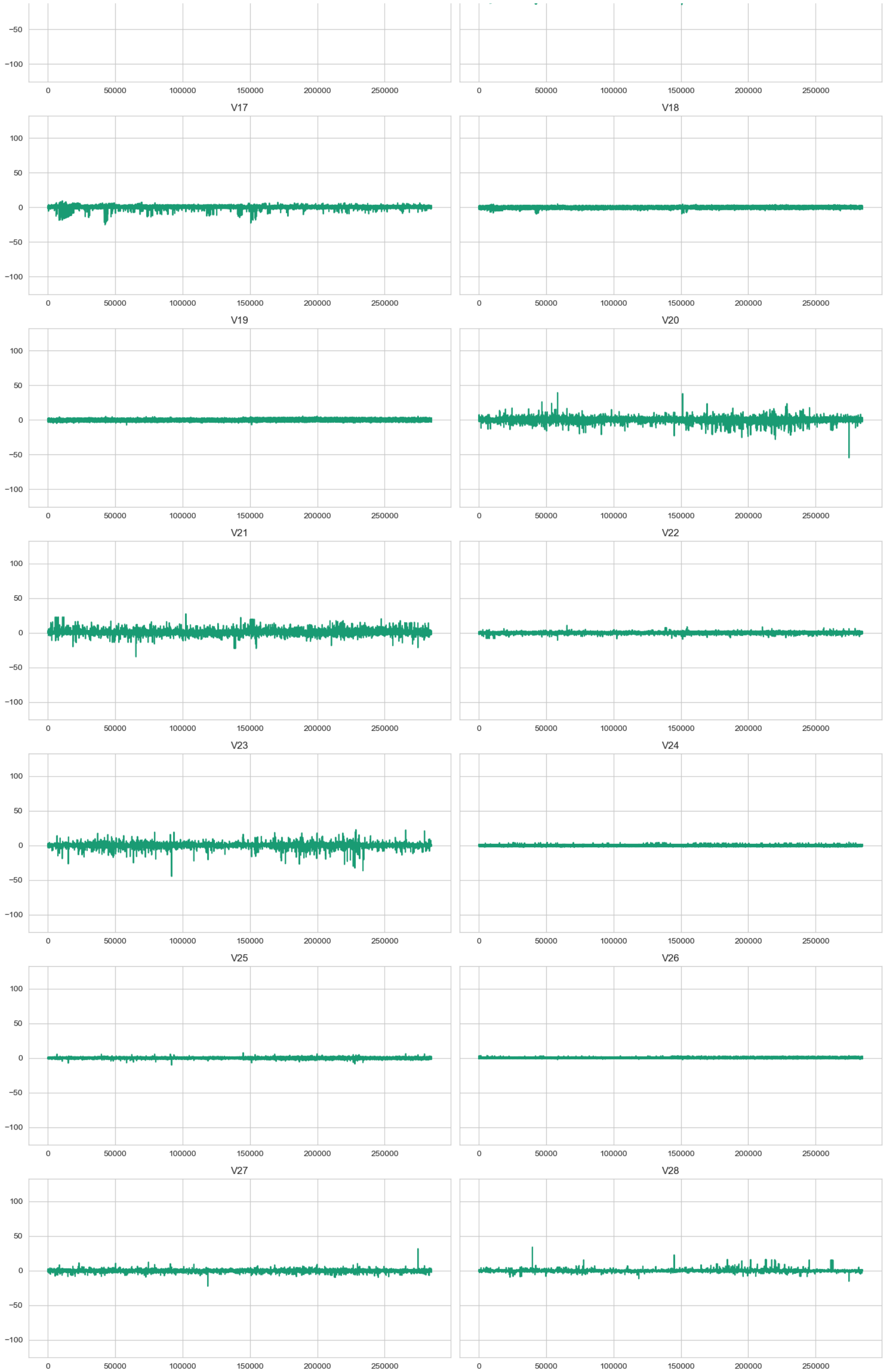
```
Average change in Time is 0.6066930709756253
```

Trend Analysis

In [157…
```
v_cols = [f'V{i}' for i in range(1, 29)]
fig, axes = plt.subplots(nrows=14, ncols=2, figsize=(15, 50), sharey=True)
axes = axes.flatten()
for i, col in enumerate(v_cols):
    df[col].plot(ax=axes[i])
    axes[i].set_title(col)

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


V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
100
50
0
−50
−100
0
50000
100000
150000
200000
250000



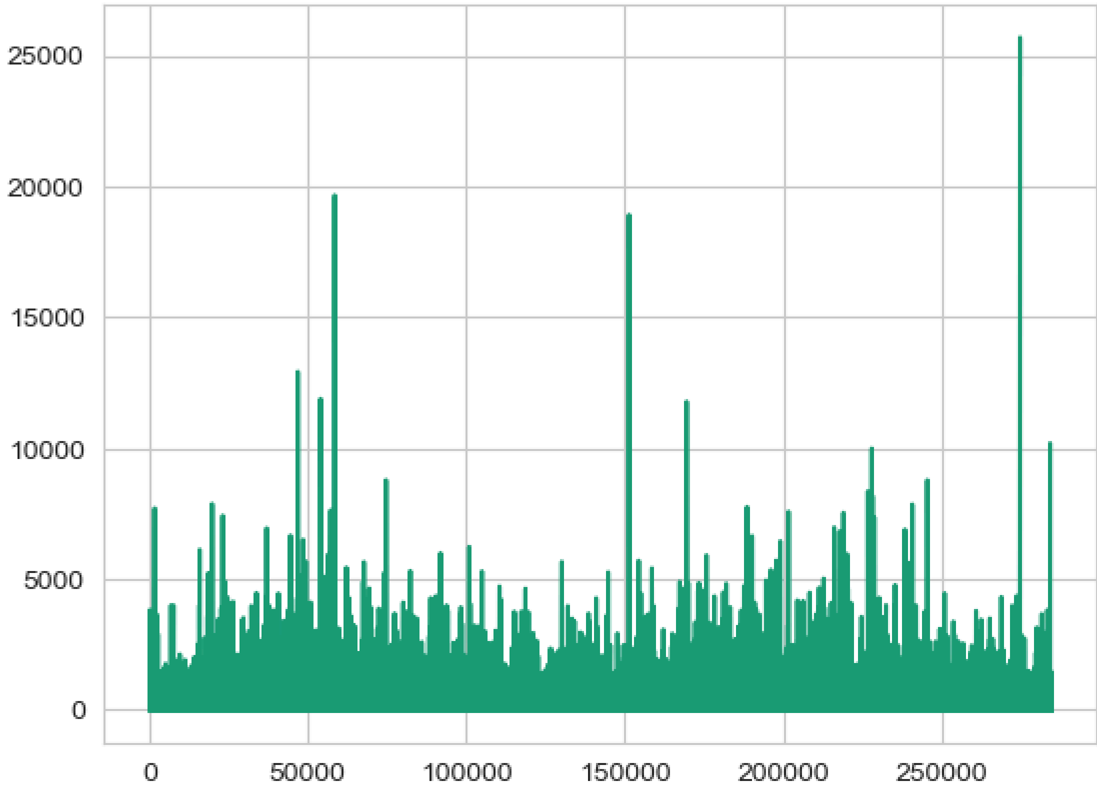
all Vs ranges from -50 to 50

In [163…

```
df['Amount'].plot()
```

Out[163]:

```
<Axes: >
```



Fraudent - Distribution

In [166…

```python
fraudent_df = df[df['Class'] == 1]
fraudent_df
```

Out[166]:

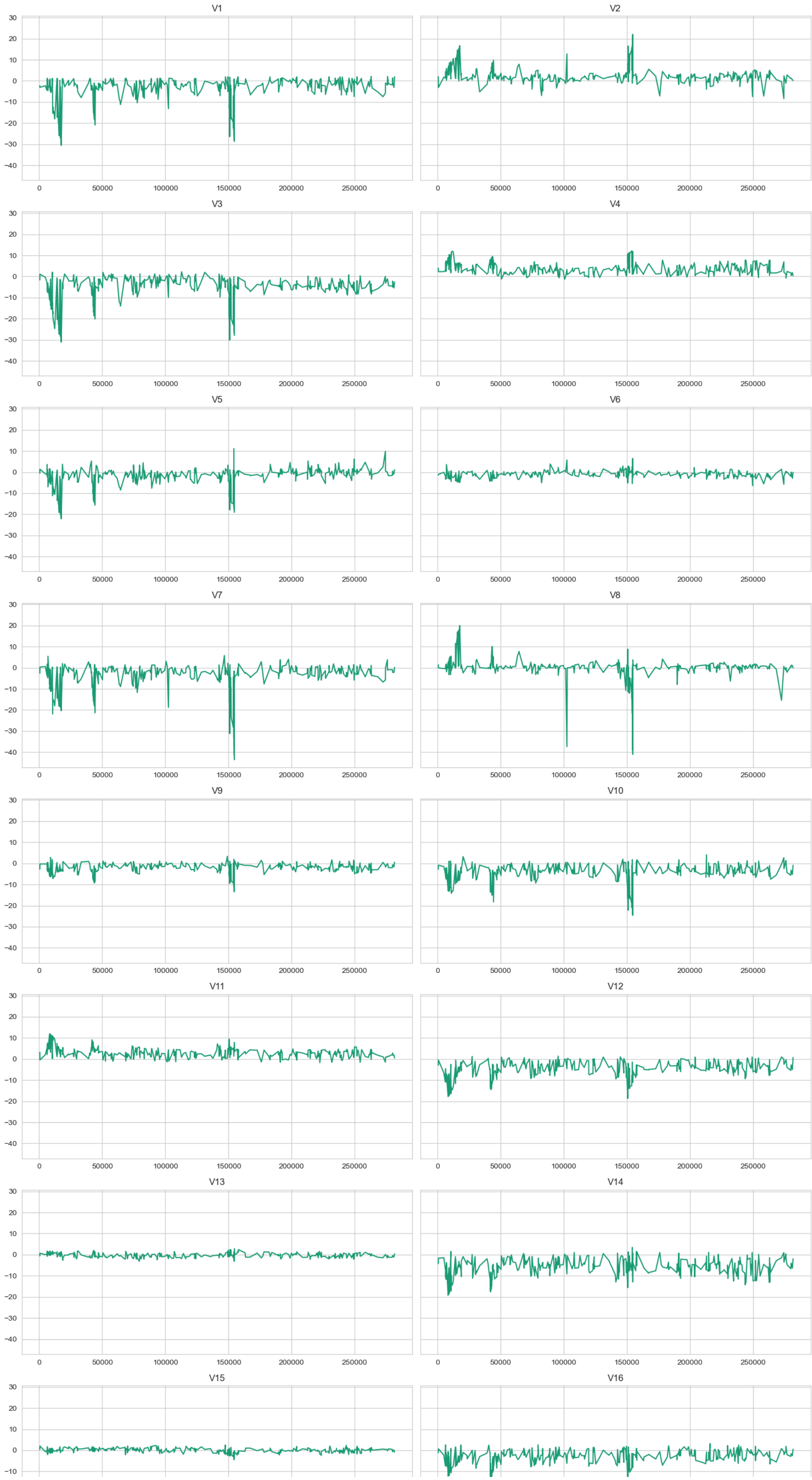
	Time	V1	V2	V3	V4	V5	V6	
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.53
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.32
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.56
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.49
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.71
...	...	...	...	...	...	...	...	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.88
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.41
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.23
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.20
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.22

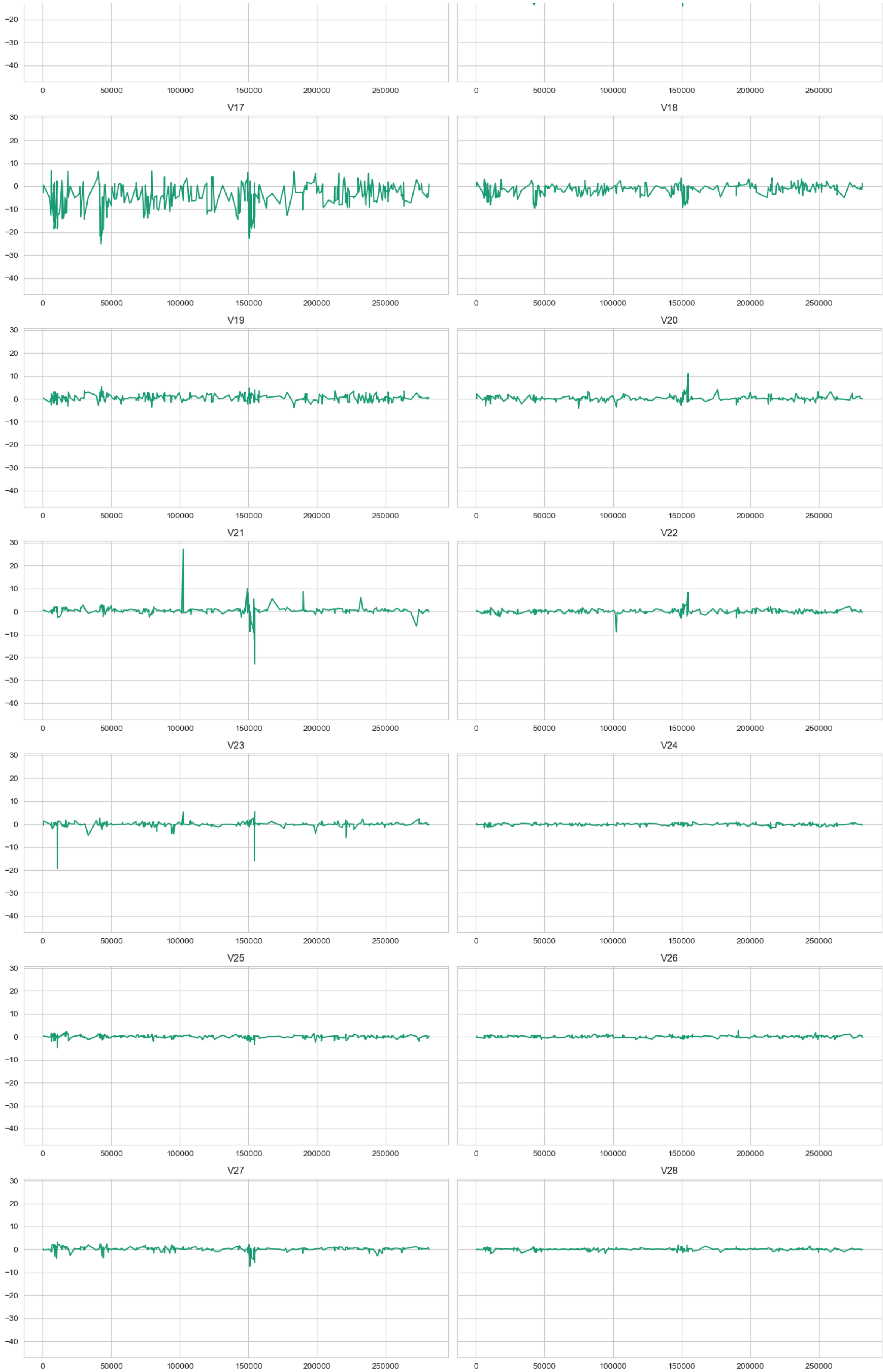
492 rows × 31 columns

In [173…

```python
v_cols = [f'V{i}' for i in range(1, 29)]
fig, axes = plt.subplots(nrows=14, ncols=2, figsize=(15, 50), sharey=True)
axes = axes.flatten()
for i, col in enumerate(v_cols):
    fraudent_df[col].plot(ax=axes[i], x=fraudent_df['Time'])
    axes[i].set_title(col)

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


V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
30
20
10
0
-10
-20
-30
-40
0
50000
100000
150000
200000
250000



In [193…

```
variation_data = []

for i, col in enumerate(v_cols):
    total_std = df[col].std()
    fraud_std = fraudent_df[col].std()
    variation_data.append({
        "Column": col,
        "Total Std Dev": total_std,
```

```
        "Fraud Std Dev": fraud_std
    })

variation_df = pd.DataFrame(variation_data)
variation_df['difference'] =  variation_df['Fraud Std Dev'] - variation_df['
print(variation_df)
```

```
   Column  Total Std Dev  Fraud Std Dev  difference
0      V1       1.958696       6.783687    4.824991
1      V2       1.651309       4.291216    2.639907
2      V3       1.516255       7.110937    5.594682
3      V4       1.415869       2.873318    1.457449
4      V5       1.380247       5.372468    3.992221
5      V6       1.332271       1.858124    0.525852
6      V7       1.237094       7.206773    5.969679
7      V8       1.194353       6.797831    5.603478
8      V9       1.098632       2.500896    1.402263
9     V10       1.088850       4.897341    3.808491
10    V11       1.020713       2.678605    1.657891
11    V12       0.999201       4.654458    3.655257
12    V13       0.995274       1.104518    0.109244
13    V14       0.958596       4.278940    3.320344
14    V15       0.915316       1.049915    0.134599
15    V16       0.876253       3.865035    2.988782
16    V17       0.849337       6.970618    6.121281
17    V18       0.838176       2.899366    2.061190
18    V19       0.814041       1.539853    0.725813
19    V20       0.770925       1.346635    0.575710
20    V21       0.734524       3.869304    3.134780
21    V22       0.725702       1.494602    0.768900
22    V23       0.624460       1.579642    0.955182
23    V24       0.605647       0.515577   -0.090071
24    V25       0.521278       0.797205    0.275927
25    V26       0.482227       0.471679   -0.010548
26    V27       0.403632       1.376766    0.973133
27    V28       0.330083       0.547291    0.217208
```

so there is unusaul pattern in fraudents, more variations

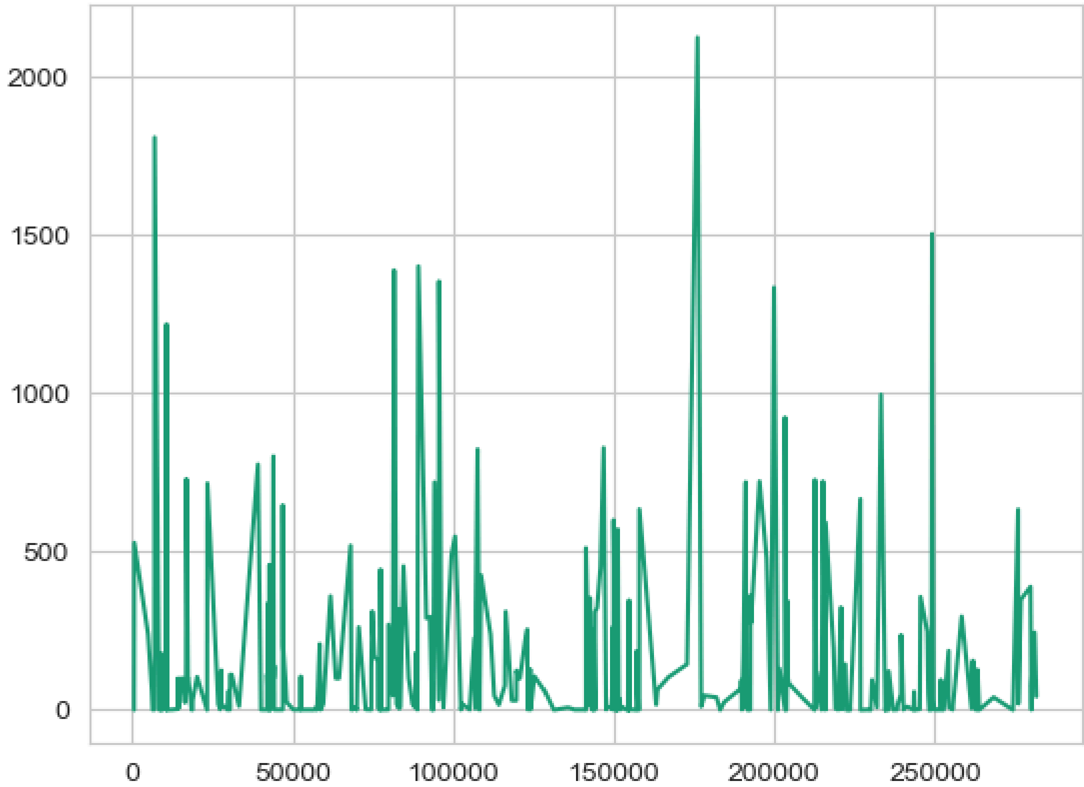
due to privacy it is not ceeared but it shows more transactions, atm check, online card payments this shows fraudent

In [171…

```
fraudent_df['Amount'].plot()
```

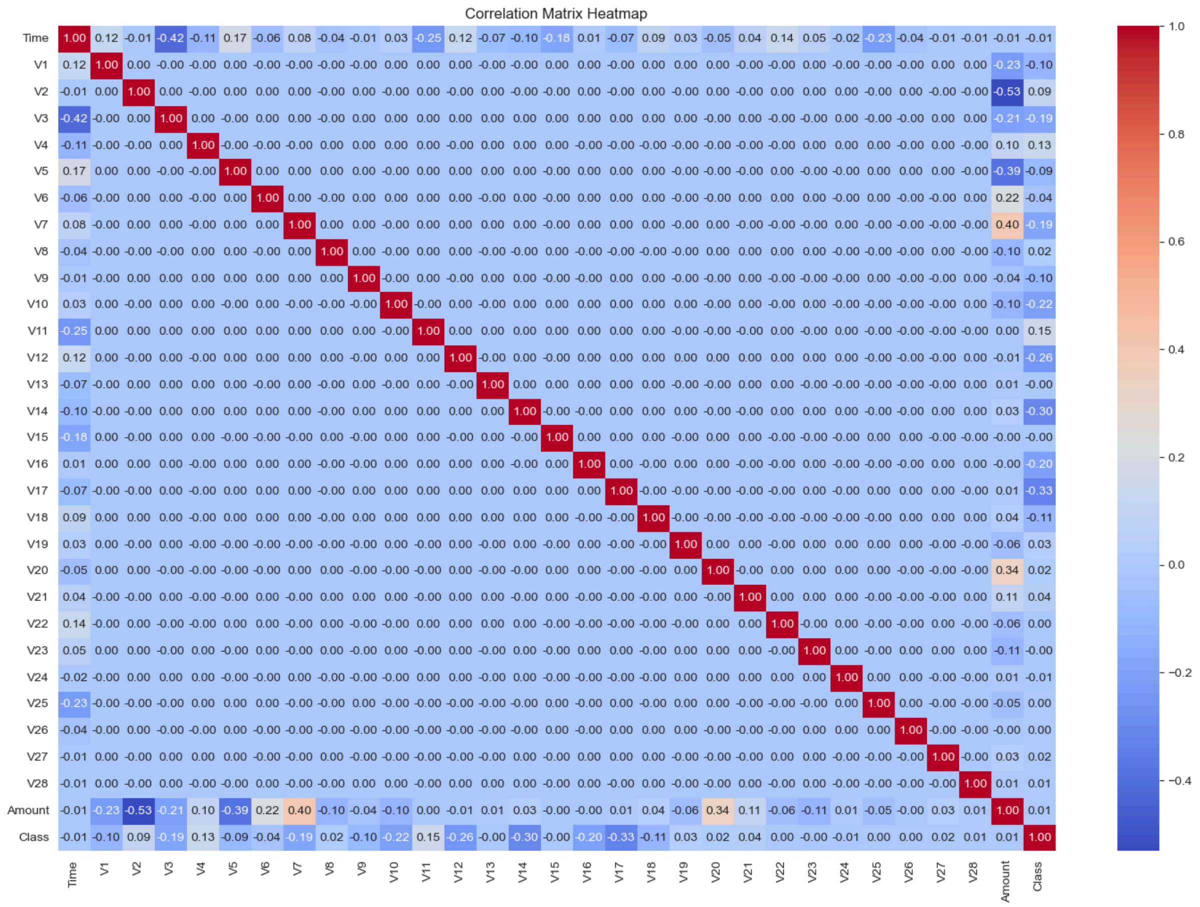
Out[171]:

```
<Axes: >
```



Correlation

In [78]:
```
corr_matrix = df.corr()
plt.figure(figsize=(18, 12))
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True, fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()
```



📊 Challenges in Visualizing Correlation with the Target Variable Visualizing correlations between the target variable (Class) and other features in the dataset presents two primary challenges:

> Severe Class Imbalance: The dataset exhibits a significant class imbalance, with approximately 99% of the instances belonging to the non-fraudulent class and only 1% to the fraudulent class. This disparity can obscure meaningf ul patterns and affect the performance of traditional machine learning models.
>
> Non-Linear Relationships: The relationship between the features and the target variable is not linear. Standard linear correlation measures may fail to capture these complex associations, leading to misleading interpretations.

Model Applying and Comparison

In [103…

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

In [105…

```
X.columns
```

Out[105]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V2
0',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'],
      dtype='object')
```

In [107…

```
Y.head()
```

Out[107]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int32
```

In [109…
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train1, Y_test = train_test_split(X, Y, test_size=0.2, ra
```

In [111…
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train, Y_train = smote.fit_resample(X_train, Y_train1)

from collections import Counter
print("Before SMOTE:", Counter(Y_train1))
print("After SMOTE:", Counter(Y_train))
```

```
Before SMOTE: Counter({0: 227451, 1: 394})
After SMOTE: Counter({0: 227451, 1: 227451})
```

Started with Random Forest Classifier

because less computative cost and easily fit!

In [124…
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42,verbose=1,
model.fit(X_train, Y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  4.1min finished
```

Out[124]:
```
▾ RandomForestClassifier
RandomForestClassifier(n_jobs=1, random_state=42, verbose=1)
```

In [125…
```
from sklearn.metrics import accuracy_score, classification_report
Y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(Y_test, Y_pred))
print("Classification Report:\n" ,classification_report(Y_test, Y_pred))
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
Accuracy: 0.9995259997893332
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.87      0.85      0.86        98

    accuracy                           1.00     56962
   macro avg       0.94      0.92      0.93     56962
weighted avg       1.00      1.00      1.00     56962

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.3s finished
```

Gradient Bossting to see how it works

In [130…
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score
gb_model = GradientBoostingClassifier(random_state=42)
```

```
gb_model.fit(X_train, Y_train)
rf_probs = model.predict_proba(X_test)[:, 1]
gb_probs = gb_model.predict_proba(X_test)[:, 1]
rf_auc = roc_auc_score(Y_test, rf_probs)
gb_auc = roc_auc_score(Y_test, gb_probs)
print(f"Random Forest AUC: {rf_auc:.4f}")
print(f"Gradient Boosting AUC: {gb_auc:.4f}")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
Random Forest AUC: 0.9849
Gradient Boosting AUC: 0.9844
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.3s finished
```

Functional API of keras,

since it can capture complex relations

In [131…

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
inputs = Input(shape=(X_train.shape[1],))
x = Dense(128, activation='relu')(inputs)
x = Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(32, activation='relu')(x)
outputs = Dense(1, activation='sigmoid')(x)
dl_model = Model(inputs=inputs, outputs=outputs)
dl_model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy',
metrics=['AUC'])
dl_model.fit(X_train_scaled, Y_train,
epochs=25, batch_size=32,
validation_split=0.2, verbose=1)
# 5. Predict and evaluate AUC
dl_probs = dl_model.predict(X_test_scaled).ravel()
dl_auc = roc_auc_score(Y_test, dl_probs)
print(f"Deep Learning (Functional API) AUC: {dl_auc:.4f}")
```

```
Epoch 1/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 607us/step - AUC: 0.9948 - loss: 0.0666
- val_AUC: 0.0000e+00 - val_loss: 0.0070
Epoch 2/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 607us/step - AUC: 0.9996 - loss: 0.0153
- val_AUC: 0.0000e+00 - val_loss: 0.0088
Epoch 3/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 598us/step - AUC: 0.9997 - loss: 0.0108
- val_AUC: 0.0000e+00 - val_loss: 0.0035
Epoch 4/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 597us/step - AUC: 0.9997 - loss: 0.0097
- val_AUC: 0.0000e+00 - val_loss: 0.0026
Epoch 5/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 608us/step - AUC: 0.9998 - loss: 0.0089
- val_AUC: 0.0000e+00 - val_loss: 6.5465e-04
Epoch 6/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 597us/step - AUC: 0.9998 - loss: 0.0074
- val_AUC: 0.0000e+00 - val_loss: 0.0031
Epoch 7/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 608us/step - AUC: 0.9998 - loss: 0.0071
- val_AUC: 0.0000e+00 - val_loss: 0.0017
Epoch 8/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 600us/step - AUC: 0.9998 - loss: 0.0070
- val_AUC: 0.0000e+00 - val_loss: 0.0025
Epoch 9/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 598us/step - AUC: 0.9998 - loss: 0.0070
- val_AUC: 0.0000e+00 - val_loss: 0.0020
Epoch 10/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 610us/step - AUC: 0.9998 - loss: 0.0059
- val_AUC: 0.0000e+00 - val_loss: 0.0024
Epoch 11/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 606us/step - AUC: 0.9998 - loss: 0.0057
- val_AUC: 0.0000e+00 - val_loss: 0.0063
Epoch 12/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 602us/step - AUC: 0.9998 - loss: 0.0058
- val_AUC: 0.0000e+00 - val_loss: 0.0017
Epoch 13/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 616us/step - AUC: 0.9998 - loss: 0.0049
- val_AUC: 0.0000e+00 - val_loss: 0.0012
Epoch 14/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 610us/step - AUC: 0.9998 - loss: 0.0050
- val_AUC: 0.0000e+00 - val_loss: 2.8047e-04
Epoch 15/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 612us/step - AUC: 0.9998 - loss: 0.0051
- val_AUC: 0.0000e+00 - val_loss: 0.0011
Epoch 16/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 605us/step - AUC: 0.9998 - loss: 0.0052
- val_AUC: 0.0000e+00 - val_loss: 0.0017
Epoch 17/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 645us/step - AUC: 0.9999 - loss: 0.0042
- val_AUC: 0.0000e+00 - val_loss: 0.0011
Epoch 18/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 8s 724us/step - AUC: 0.9998 - loss: 0.0048
- val_AUC: 0.0000e+00 - val_loss: 0.0014
Epoch 19/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 9s 784us/step - AUC: 0.9999 - loss: 0.0042
- val_AUC: 0.0000e+00 - val_loss: 0.0011
Epoch 20/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 8s 670us/step - AUC: 0.9999 - loss: 0.0040
- val_AUC: 0.0000e+00 - val_loss: 8.7408e-04
Epoch 21/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 638us/step - AUC: 0.9998 - loss: 0.0047
- val_AUC: 0.0000e+00 - val_loss: 0.0010
Epoch 22/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 599us/step - AUC: 0.9999 - loss: 0.0047
- val_AUC: 0.0000e+00 - val_loss: 5.7594e-04
```

```
Epoch 23/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 598us/step - AUC: 0.9998 - loss: 0.0044
- val_AUC: 0.0000e+00 - val_loss: 4.7365e-04
Epoch 24/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 594us/step - AUC: 0.9998 - loss: 0.0041
- val_AUC: 0.0000e+00 - val_loss: 6.5735e-04
Epoch 25/25
11373/11373 ━━━━━━━━━━━━━━━━━━━━ 7s 654us/step - AUC: 0.9998 - loss: 0.0040
- val_AUC: 0.0000e+00 - val_loss: 8.5520e-04
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 1s 329us/step
Deep Learning (Functional API) AUC: 0.9734
```

In [132…

```python
print(f"Random Forest AUC: {rf_auc:.4f}")
print(f"Deep Learning (Functional API) AUC: {dl_auc:.4f}")
```

```
Random Forest AUC: 0.9849
Deep Learning (Functional API) AUC: 0.9734
```

In [133…

```python
from sklearn.metrics import accuracy_score, classification_report
Y_pred = dl_model.predict(X_test)
Y_pred_class = (Y_pred >= 0.5).astype(int)
accuracy = accuracy_score(Y_test, Y_pred_class)
print("Accuracy:", accuracy)
report = classification_report(Y_test, Y_pred_class)
print("Classification Report:\n"
, report)
```

```
1781/1781 ━━━━━━━━━━━━━━━━━━━━ 0s 237us/step
Accuracy: 0.9977528878901724
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.00      0.00      0.00        98

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962
```

Deployement

In [135…

```python
testdf = pd.read_csv('test.csv')
```

In [136…

```python
predictions = dl_model.predict(testdf)
testdf['Predictions'] = predictions
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 749us/step
```

In [137…

```python
checking = testdf.groupby('Predictions')['Time'].count()
checking
```

Out[137]:

```
Predictions
0.0    449
1.0     51
Name: Time, dtype: int64
```

In [202…

```python
testdf_F = testdf[testdf['Predictions'] == 1]
testdf_F.describe()
```

Out[202]:

	Time	V1	V2	V3	V4	V5	
count	51.000000	51.000000	51.000000	51.000000	51.000000	51.000000	51.0000
mean	14277.339140	43.278081	34.504639	28.804562	24.269724	19.937705	17.7656
std	8876.906755	51.080969	62.192787	57.210912	49.287418	55.242408	57.484
min	853.589186	-67.699390	-67.595013	-69.953842	-68.091074	-65.794454	-69.1484
25%	6918.328914	11.585473	-17.854826	-22.609622	-13.420239	-22.159383	-24.122
50%	14078.037730	54.438810	39.354406	34.283146	18.669800	12.536631	4.755
75%	20063.739685	86.145422	88.693485	77.012996	65.739820	78.551294	66.2109
max	33548.032750	116.209869	119.242429	114.942698	111.429045	111.573785	117.252

8 rows × 31 columns

Success: This box indicates a successful action.

In [140…

```
input_values = {
'Time': float(input("Enter Time: ")),
'V1': float(input("Enter V1: ")),
'V2': float(input("Enter V2: ")),
'V3': float(input("Enter V3: ")),
'V4': float(input("Enter V4: ")),
'V5': float(input("Enter V5: ")),
'V6': float(input("Enter V6: ")),
'V7': float(input("Enter V7: ")),
'V8': float(input("Enter V8: ")),
'V9': float(input("Enter V9: ")),
'V10': float(input("Enter V10: ")),
'V11': float(input("Enter V11: ")),
'V12': float(input("Enter V12: ")),
'V13': float(input("Enter V13: ")),
'V14': float(input("Enter V14: ")),
'V15': float(input("Enter V15: ")),
'V16': float(input("Enter V16: ")),
'V17': float(input("Enter V17: ")),
'V18': float(input("Enter V18: ")),
'V19': float(input("Enter V19: ")),
'V20': float(input("Enter V20: ")),
'V21': float(input("Enter V21: ")),
'V22': float(input("Enter V22: ")),
'V23': float(input("Enter V23: ")),
'V24': float(input("Enter V24: ")),
'V25': float(input("Enter V25: ")),
'V26': float(input("Enter V26: ")),
'V27': float(input("Enter V27: ")),
'V28': float(input("Enter V28: ")),
'Amount': float(input("Enter Amount: "))
}
input_df = pd.DataFrame([input_values])
scaled_input = scaler.transform(input_df)
prediction = model.predict(scaled_input)
print("Prediction:", "Fraudulent (1)" if prediction[0] == 1 else "Legitimat
```

```
Prediction: Legitimate (0)
```

```
/Users/ahadmoeen/anaconda3/lib/python3.11/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: