In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [16]:
```python
df=pd.read_csv('creditbn.csv')
#df=pd.read_csv('credits.csv')
```

In [17]:
```python
df
```

Out[17]:

	Time	V1	V2	V3	V4	V5	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095
...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649

284807 rows × 31 columns

In [18]:
```python
df.head()
```

Out[18]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.2395
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.0788
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.7914
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.2376
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.5929

5 rows × 31 columns

In [19]:
```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 [20]:

```
df.describe()
```

Out[20]:

	Time	V1	V2	V3	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e
mean	94813.859575	1.759061e-12	-8.251130e-13	-9.654937e-13	8.321385e
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e

8 rows × 31 columns

In [7]:

```
df.count
```

Out[7]:

```
<bound method DataFrame.count of             Time         V1         V2
V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
...          ...        ...        ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278
...          ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount
\
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
...          ...       ...       ...       ...       ...       ...     ...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
...       ...
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]>
```

In [8]:
```
df.columns
```

Out[8]:
```
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',
       'Class'],
      dtype='object')
```

In [9]:
```
df.dtypes
```

Out[9]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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 [21]:
```
df.shape
```

Out[21]:
```
(284807, 31)
```

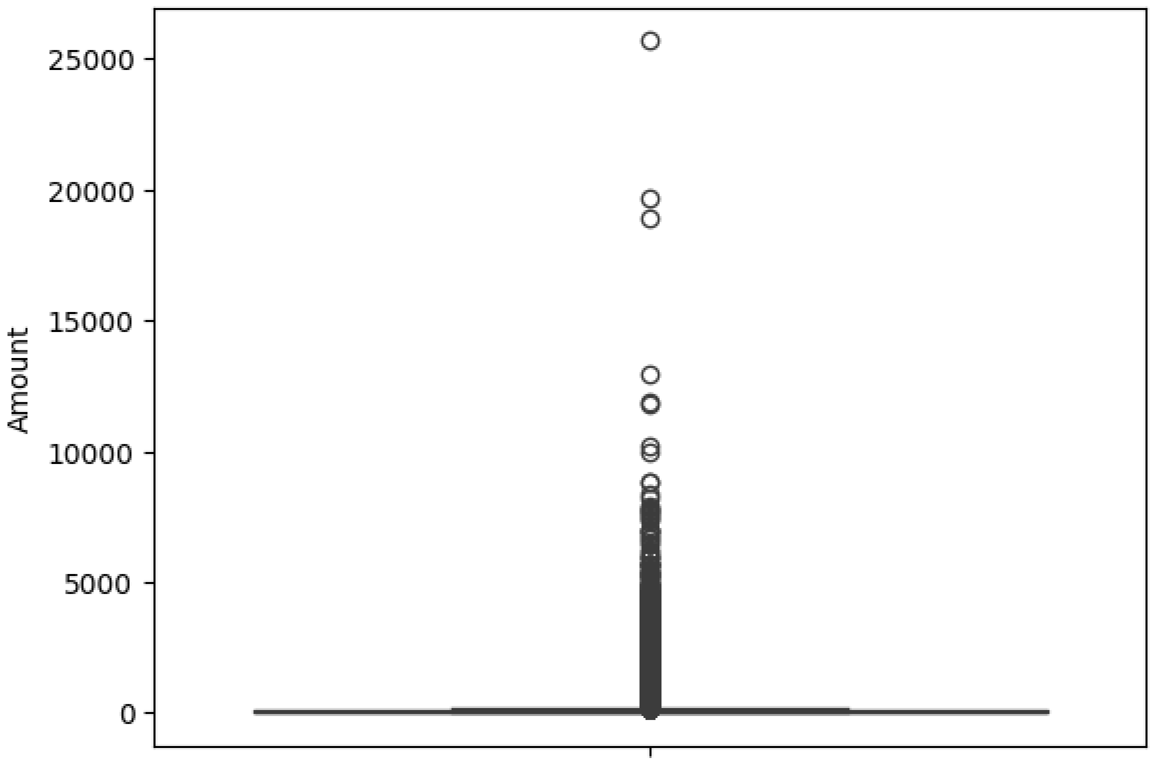
check the descriptive summary of data

In [13]:
```
df.describe
```

```
Out[13]: <bound method NDFrame.describe of             Time         V1         V2        
V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321   
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018   
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198   
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309   
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193   
...          ...        ...        ...       ...       ...       ...   
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473   
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229   
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515   
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961   
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546   

              V6        V7        V8        V9  ...       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838   
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672   
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679   
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274   
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278   
...          ...       ...       ...       ...  ...       ...       ...   
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864   
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384   
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229   
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049   
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078   

             V23       V24       V25       V26       V27       V28  Amount  
\
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62   
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69   
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66   
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50   
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99   
...          ...       ...       ...       ...       ...       ...     ...   
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77   
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79   
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88   
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00   
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00   

        Class  
0           0  
1           0  
2           0  
3           0  
4           0  
...       ...  
284802      0  
284803      0  
284804      0  
284805      0  
284806      0  

[284807 rows x 31 columns]>
```

In [22]:
```python
sns.boxplot(y= 'Amount', data=df)
```

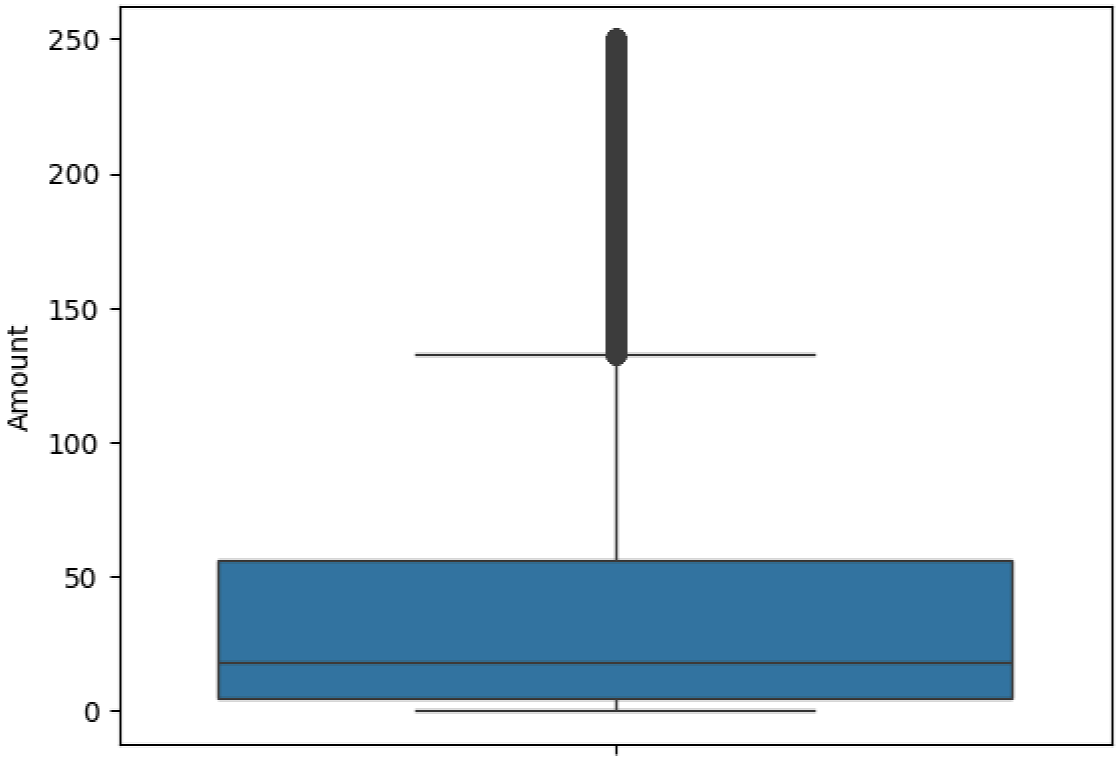
Out[22]: <Axes: ylabel='Amount'>



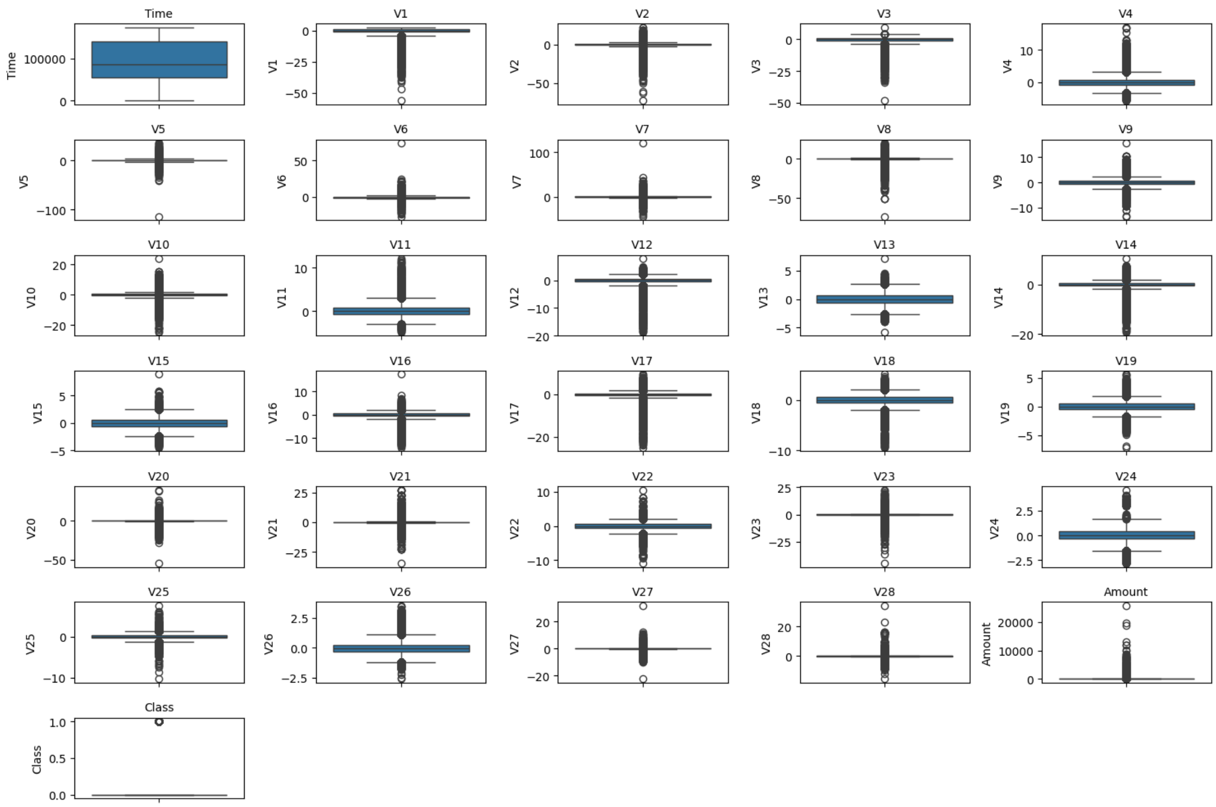
visually inspecting a feature with reduced range

In [25]:
```python
sns.boxplot(y= 'Amount', data=df[df['Amount']<250])
```

Out[25]: <Axes: ylabel='Amount'>



In [38]:
```
plt.figure(figsize= (15,10))
for i in range(len(df.columns)):
    plt.subplot(7,5, i+1)
    sns.boxplot(y= df.columns[i], data=df)
    plt.title(df.columns[i], fontsize=10)
    plt.tight_layout()  #add parenthis
plt.show()
```



plt.tight_layout() = it automatically adjusts the spacing between subplots so that:

how to detect outliers

In [42]:
```
# identify the column we want to analyses for  outliers
column_to_analyze= 'Amount'

#calculate the 25th and 75th parcentile(Q1, Q3)
Q1= df[column_to_analyze].quantile(0.25)
Q2= df[column_to_analyze].quantile(0.50)
Q3= df[column_to_analyze].quantile(0.75)

IQR= Q3- Q1

# calculate the lower bound and upper bound
Upper_bound= Q3 + 1.5*IQR
Lower_bound=  Q1 - 1.5*IQR

outliers= df[(df[column_to_analyze]< Lower_bound) | (df[column_to_analyze]>
outliers

# print the outliers shape
print(outliers.shape[0])
```

```
31904
```

In [43]:
```
outliers
```

Out[43]:

	Time	V1	V2	V3	V4	V5	V
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.8004
20	16.0	0.694885	-1.361819	1.029221	0.834159	-1.191209	1.3091
51	36.0	-1.004929	-0.985978	-0.038039	3.710061	-6.631951	5.1221
64	42.0	-0.522666	1.009923	0.276470	1.475289	-0.707013	0.3552
85	55.0	-4.575093	-4.429184	3.402585	0.903915	3.002224	-0.4910
...	...	...	...	...	...	...	
284735	172727.0	-1.661169	-0.565425	0.294268	-1.549156	-2.301359	2.3659
284748	172738.0	1.634178	-0.486939	-1.975967	0.495364	0.263635	-0.7130
284753	172743.0	1.465737	-0.618047	-2.851391	1.425282	0.893893	-0.9583
284757	172745.0	-1.757643	-0.982659	1.091540	-1.409539	-0.662159	0.0469
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.6496

31904 rows × 31 columns

outlier detect for entire dataset

In [47]:
```
Q1= df[column_to_analyze].quantile(0.25)
Q3= df[column_to_analyze].quantile(0.75)
# calculate the lower bound and upper bound
Upper_bounds= Q3 + 1.5*IQR
Lower_bounds=  Q1 - 1.5*IQR
```

In [50]:
```
((df>Upper_bounds) | (df<Lower_bounds)).sum()  # count the outliers for the
```

```
Out[50]: Time      284544
V1             0
V2             0
V3             0
V4             0
V5             1
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     31904
Class          0
dtype: int64
```

checking the count of outliers in entire dataframe

```
In [53]: ((df>Upper_bounds) | (df<Lower_bounds)).sum().sum()
```

```
Out[53]: np.int64(316449)
```

handiling outliers

```
In [56]: # Removal- we can remove outliers from the dataset
data1= df[(df[column_to_analyze] >= Lower_bound) & (df[column_to_analyze] <=
print("Rows before removing the outliers", df.shape[0])
print(" ")
print("Rows after removing the outliers", data1.shape[0])
print(" ")
print("Rows total lost", df.shape[0]- data1.shape[0])
```

```
Rows before removing the outliers 284807

Rows after removing the outliers 252903

Rows total lost 31904
```

check the no. of rows that have only one outlier in our dataset

In [58]:
```
df[(( df> Upper_bounds) | (df< Lower_bounds))].any(axis=1).sum()
```

Out[58]:
```
np.int64(284567)
```

In [62]:
```
outlier_rows = ((df > Upper_bounds) | (df < Lower_bounds)).any(axis=1).sum()
print("Number of rows with at least one outlier:", outlier_rows)
```

```
Number of rows with at least one outlier: 284567
```

2nd method

replacing the outliers with their median value in our dataset

In [63]:
```
data2= df.copy()
data2.loc[data2[column_to_analyze]>= Upper_bound, column_to_analyze]= Q2
data2.loc[data2[column_to_analyze]<= Lower_bound, column_to_analyze]= Q2

# check the no. of rows containing outliers before the formula uses
data2[(data2[column_to_analyze]< Lower_bound) | (data2[column_to_analyze]> U
```

Out[63]:
```
0
```

or - It uses when we find the rows that have the outliers

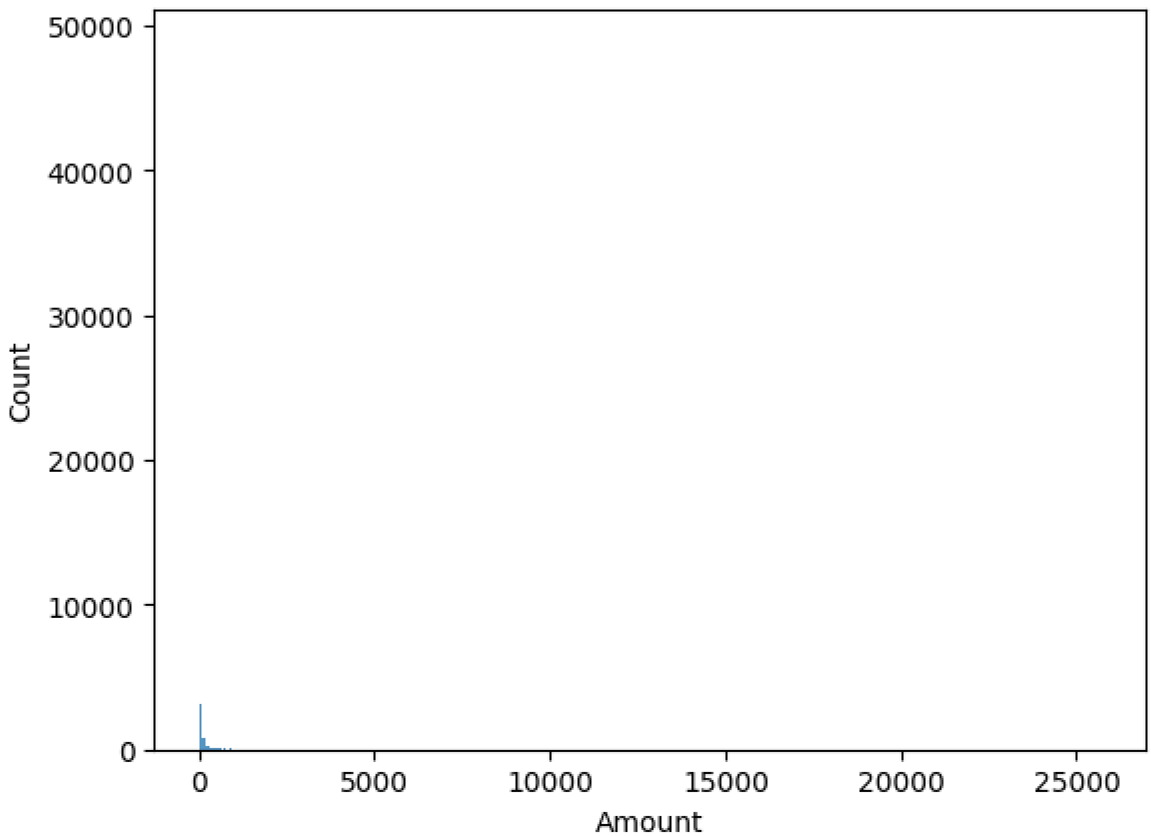
and- It uses when we keep only the normal(non-outlier) values

3rd method - Transformatio method

Transformation is a way to reduce the influence of extreme values (outliers) by applying mathematical functions that “compress” the data range.

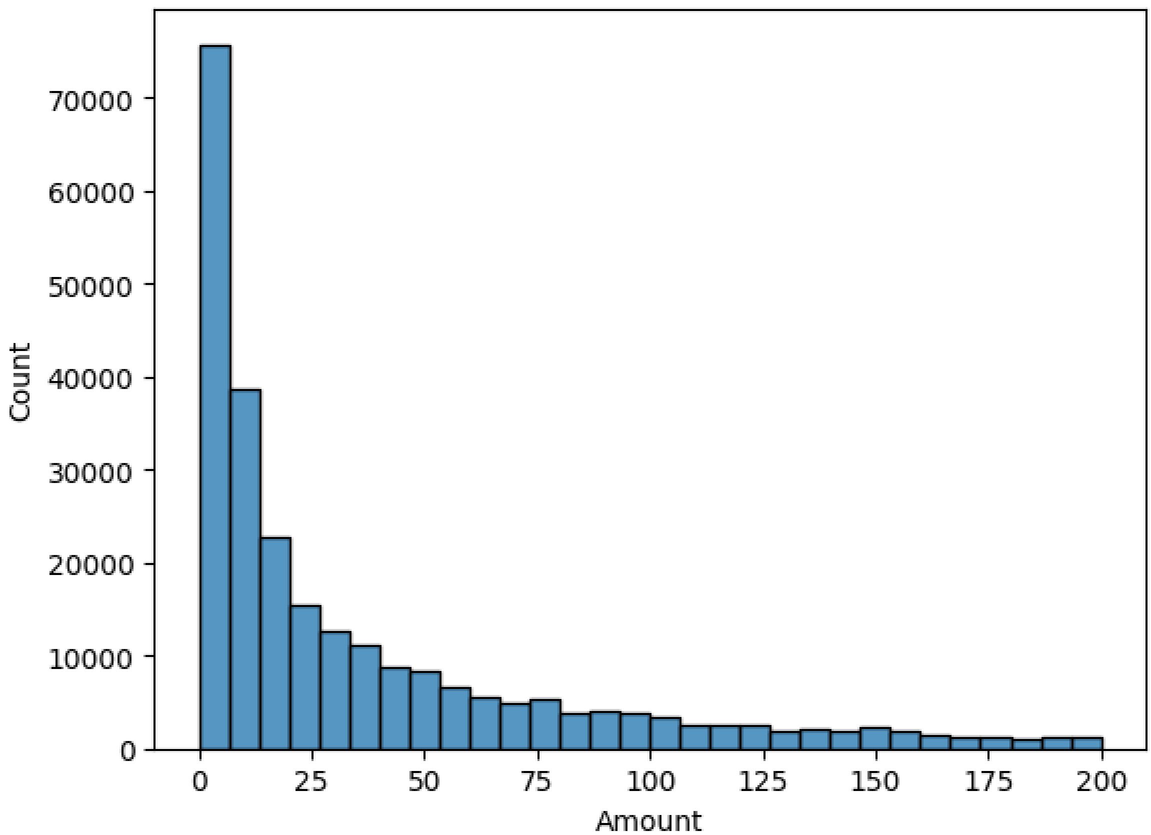
It helps make the data more symmetric or normally distributed — especially useful before applying algorithms like linear regression, logistic regression, or clustering

In [65]:
```
# Distribution of original features:
sns.histplot(x= 'Amount', data= df)
```



In [69]:
```
sns.histplot(x= 'Amount', data= df[df['Amount']< 200], bins=30)  #we can see
```

Out[69]: <Axes: xlabel='Amount', ylabel='Count'>



bins=30

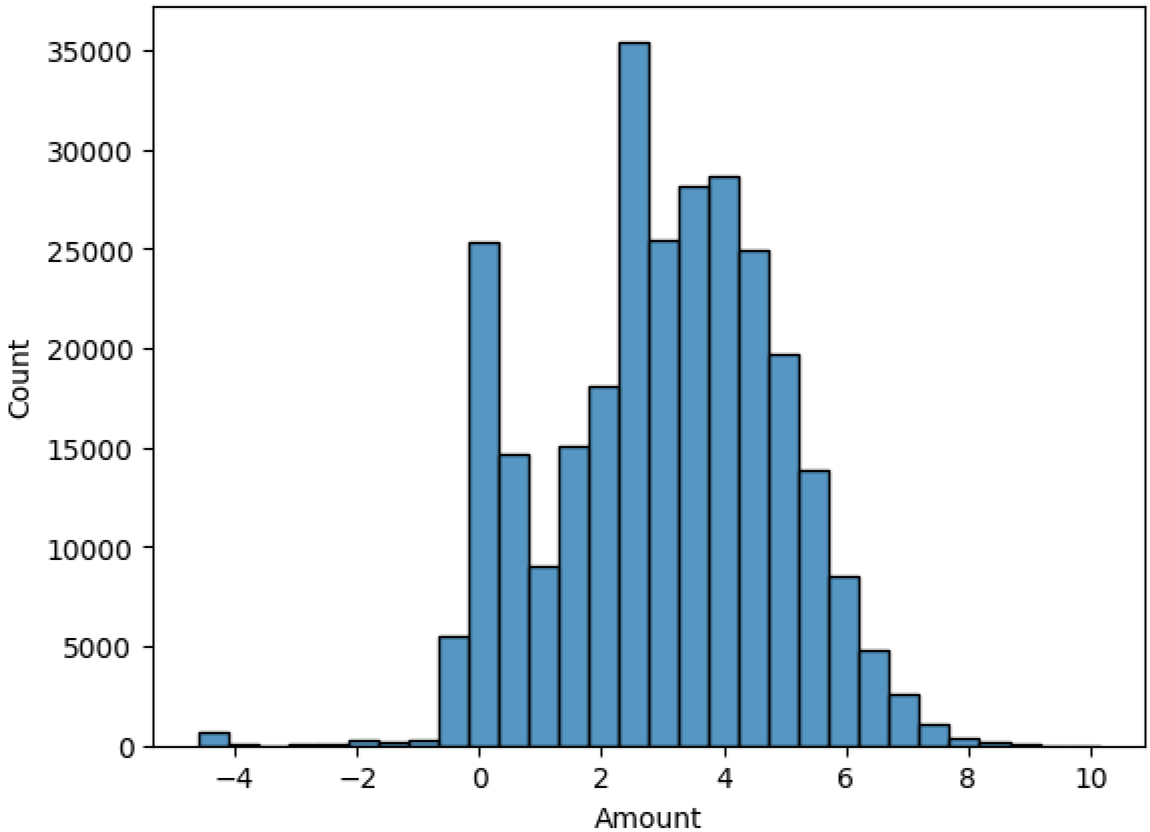
it signifies- This divides the x-axis into 30 intervals (bins). Each bar in the histogram shows how many data points fall within that interval.

Each bin represents a range of values, and the height of the bar shows how many data points fall within that range.

In [70]:
```
# applying the logarithm transformation to the "Amount column'
data3= df.copy()
data3['Amount']= np.log(df['Amount'])
sns.histplot(x= 'Amount', data= data3[data3['Amount']< 200], bins=30)
```

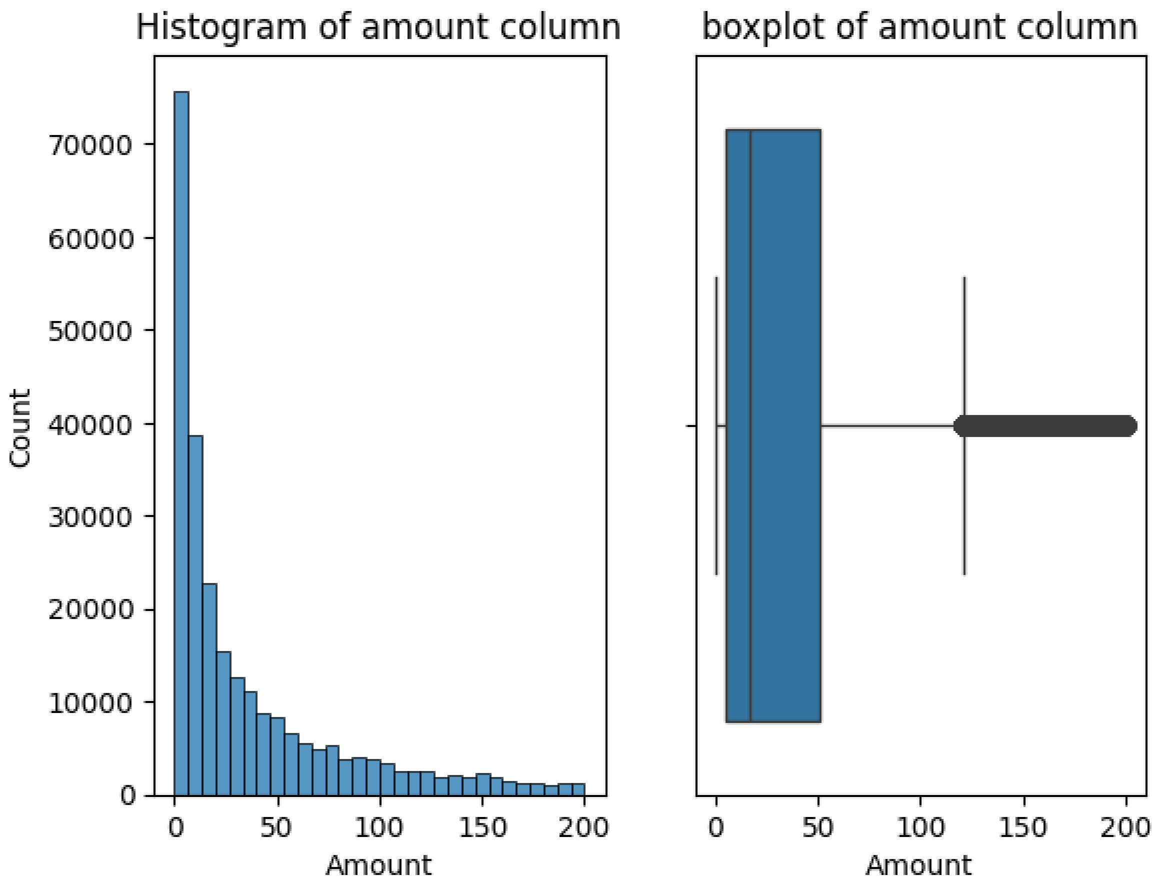
```
C:\Users\shaw3\AppData\Local\Programs\Python\Python312\Lib\site-packages\pan
das\core\arraylike.py:399: RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[70]: <Axes: xlabel='Amount', ylabel='Count'>



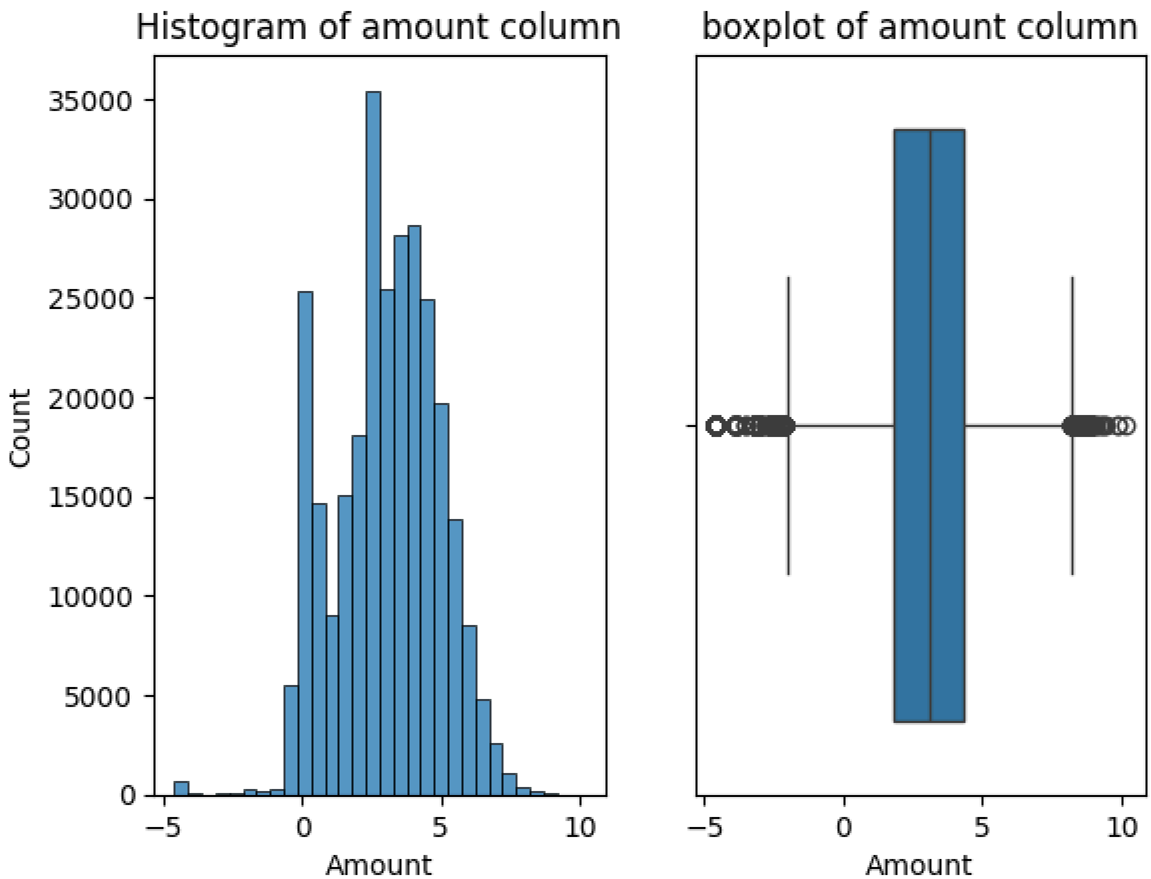
In [75]:
```
# Before applying the transformation
fig, axes= plt.subplots(1,2)
sns.histplot(x= 'Amount', data= df[df['Amount']< 200], bins=30, ax= axes[0])
axes[0].set_title('Histogram of amount column')

sns.boxplot(x= 'Amount', data= df[df['Amount']< 200],  ax= axes[1])
axes[1].set_title('boxplot of amount column')
```

Out[75]: Text(0.5, 1.0, 'boxplot of amount column')




```
In [76]: # after applying the transformation
         fig, axes= plt.subplots(1,2)
         sns.histplot(x= 'Amount', data= data3[data3['Amount']< 200], bins=30, ax= ax
         axes[0].set_title('Histogram of amount column')

         sns.boxplot(x= 'Amount', data= data3[data3['Amount']< 200],  ax= axes[1])
         axes[1].set_title('boxplot of amount column')
```

```
Out[76]: Text(0.5, 1.0, 'boxplot of amount column')
```



It only includes rows where Amount < 200 — this removes extreme outliers from the plot so we can see the main data distribution clearly.

we’re using the object-oriented (subplots) approach, that's why we use subplots — especially when we want to make side-by-side plots (like histogram + boxplot).

4th method

Winsorization method

Winsorization is a transformation method used to limit extreme values in our dataset instead of removing them.

Any value above a high threshold is replaced with that threshold, and any value below a low threshold is replaced with that lower limit.

In [83]:
```
data4= df.copy()
print(Lower_bound, Upper_bound)
```

```
-101.7475 184.5125
```

In [82]:
```
data4['Amount']= df['Amount'].clip(lower=Lower_bound, upper= Upper_bound)
```

The .clip() function in Pandas is used to limit (cap) the values of a Series or DataFrame. It’s exactly how we use this for implement Winsorization manually.

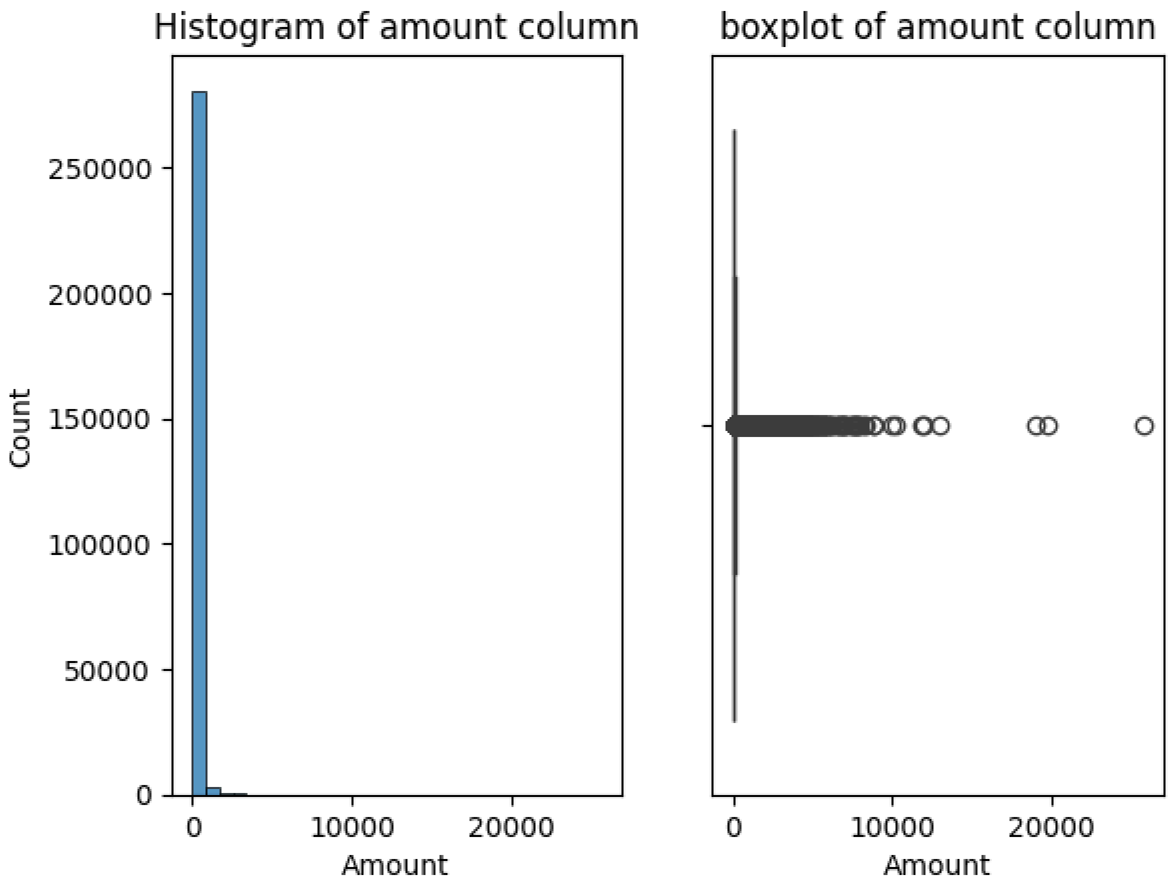
In [87]:
```
ok= df['Amount'].max()
ok    # check the maximum value before treatment
```

Out[87]:
```
np.float64(25691.16)
```

In [88]:
```
# after treatment
data4['Amount'].max()
```

Out[88]:
```
np.float64(25691.16)
```

In [89]:
```
# visually inspect the data after treatment
fig, axes= plt.subplots(1,2)
sns.histplot(x= 'Amount', data= data4, bins=30, ax= axes[0])
axes[0].set_title('Histogram of amount column')

sns.boxplot(x= 'Amount', data= data4,  ax= axes[1])
axes[1].set_title('boxplot of amount column')
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

This notebook was converted with convert.ploomber.io