```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```python
df=pd.read_csv("earthquake_data_tsunami.csv")
```

```python
df.head(5)
```

```
   magnitude  cdi  mmi  sig  nst   dmin   gap    depth  latitude
longitude  \
0        7.0    8    7  768  117  0.509  17.0   14.000   -9.7963
159.596
1        6.9    4    4  735   99  2.229  34.0   25.000   -4.9559
100.738
2        7.0    3    3  755  147  3.125  18.0  579.000  -20.0508    -
178.346
3        7.3    5    5  833  149  1.865  21.0   37.000  -19.2918    -
172.129
4        6.6    0    2  670  131  4.998  27.0  624.464  -25.5948
178.278

   Year  Month  tsunami
0  2022     11        1
1  2022     11        0
2  2022     11        1
3  2022     11        1
4  2022     11        1
```

```python
df.shape
```

```
(782, 13)
```

```python
df.isna().sum()
```

```
magnitude    0
cdi          0
mmi          0
sig          0
nst          0
dmin         0
gap          0
depth        0
latitude     0
longitude    0
Year         0
Month        0
tsunami      0
dtype: int64
```

```
df.drop_duplicates()
```

```
     magnitude  cdi  mmi  sig  nst   dmin   gap    depth  latitude
longitude  \
0          7.0    8    7  768  117  0.509  17.0   14.000   -9.7963
159.596
1          6.9    4    4  735   99  2.229  34.0   25.000   -4.9559
100.738
2          7.0    3    3  755  147  3.125  18.0  579.000  -20.0508   -
178.346
3          7.3    5    5  833  149  1.865  21.0   37.000  -19.2918   -
172.129
4          6.6    0    2  670  131  4.998  27.0  624.464  -25.5948
178.278
..         ...  ...  ...  ...  ...    ...   ...      ...       ...
...
777        7.7    0    8  912  427  0.000   0.0   60.000   13.0490
-88.660
778        6.9    5    7  745    0  0.000   0.0   36.400   56.7744   -
153.281
779        7.1    0    7  776  372  0.000   0.0  103.000  -14.9280
167.170
780        6.8    0    5  711   64  0.000   0.0   33.000    6.6310
126.899
781        7.5    0    7  865  324  0.000   0.0   33.000    6.8980
126.579

     Year  Month  tsunami
0    2022     11        1
1    2022     11        0
2    2022     11        1
3    2022     11        1
4    2022     11        1
..    ...    ...      ...
777  2001      1        0
778  2001      1        0
779  2001      1        0
780  2001      1        0
781  2001      1        0

[782 rows x 13 columns]
```

```
df.describe()
```

```
        magnitude         cdi         mmi          sig         nst  \
count  782.000000  782.000000  782.000000   782.000000  782.000000
mean     6.941125    4.333760    5.964194   870.108696  230.250639
std      0.445514    3.169939    1.462724   322.465367  250.188177
min      6.500000    0.000000    1.000000   650.000000    0.000000
25%      6.600000    0.000000    5.000000   691.000000    0.000000
```

```
50%      6.800000    5.000000    6.000000   754.000000  140.000000
75%      7.100000    7.000000    7.000000   909.750000  445.000000
max      9.100000    9.000000    9.000000  2910.000000  934.000000

             dmin         gap       depth    latitude   longitude  \
count  782.000000  782.000000  782.000000  782.000000  782.000000
mean     1.325757   25.038990   75.883199    3.538100   52.609199
std      2.218805   24.225067  137.277078   27.303429  117.898886
min      0.000000    0.000000    2.700000  -61.848400 -179.968000
25%      0.000000   14.625000   14.000000  -14.595600  -71.668050
50%      0.000000   20.000000   26.295000   -2.572500  109.426000
75%      1.863000   30.000000   49.750000   24.654500  148.941000
max     17.654000  239.000000  670.810000   71.631200  179.662000

              Year       Month    tsunami
count   782.000000  782.000000  782.000000
mean   2012.280051    6.563939    0.388747
std       6.099439    3.507866    0.487778
min    2001.000000    1.000000    0.000000
25%    2007.000000    3.250000    0.000000
50%    2013.000000    7.000000    0.000000
75%    2017.000000   10.000000    1.000000
max    2022.000000   12.000000    1.000000
```

```
df.columns
```

```
Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap',
'depth',
       'latitude', 'longitude', 'Year', 'Month', 'tsunami'],
      dtype='object')
```

```
cols=['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap', 'depth',
       'latitude', 'longitude', 'Year', 'Month', 'tsunami']
```

```
for i in cols:
    print(df[i].value_counts())
```

```
magnitude
6.50    131
6.60    115
6.70     98
6.80     78
6.90     77
7.00     49
7.10     43
7.30     31
7.20     30
7.60     22
7.50     22
7.40     18
7.70     16
```

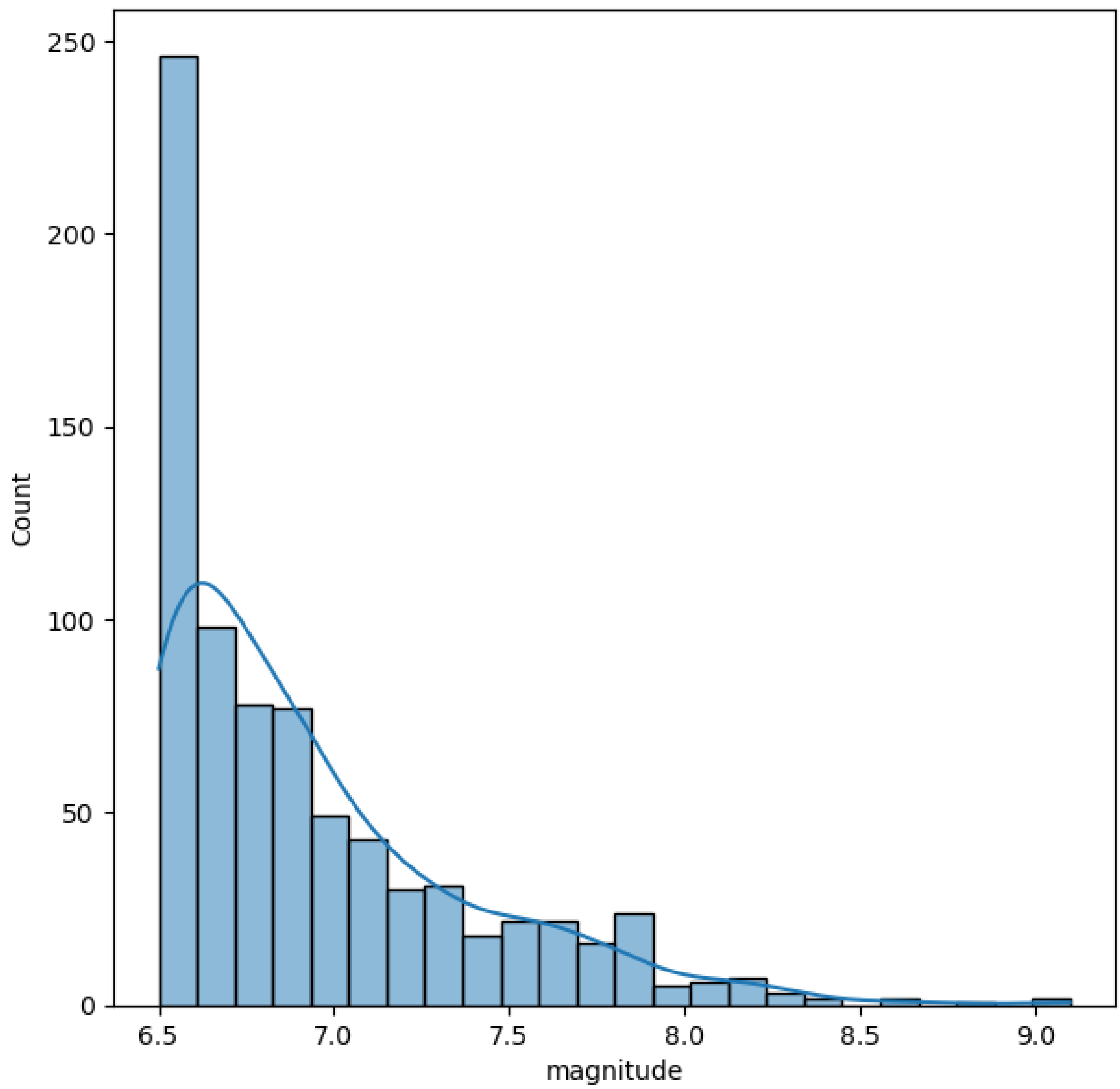
```
7.80     15
7.90      9
8.10      6
8.20      6
8.00      5
8.30      3
8.60      2
9.10      2
8.40      2
8.80      1
8.16      1
Name: count, dtype: int64
cdi
0    212
5    107
7     97
8     86
6     77
9     66
4     62
3     47
1     14
2     14
Name: count, dtype: int64
mmi
7    209
6    203
5    142
4     87
8     68
3     40
9     28
2      4
1      1
Name: count, dtype: int64
sig
650     50
670     41
691     36
711     25
776     18
        ..
1013     1
1026     1
1024     1
1750     1
1441     1
Name: count, Length: 339, dtype: int64
nst
```

```
0      365
282      4
398      4
518      4
385      3
      ... 
192      1
178      1
215      1
64       1
175      1
Name: count, Length: 312, dtype: int64
dmin
0.000    405
2.705      2
2.045      2
0.828      2
0.289      2
        ... 
2.977      1
3.158      1
8.454      1
5.293      1
3.968      1
Name: count, Length: 369, dtype: int64
gap
0.0     70
18.0    23
16.0    22
22.0    22
12.0    20
        ..
26.7     1
19.1     1
19.4     1
28.7     1
38.4     1
Name: count, Length: 256, dtype: int64
depth
10.000     92
35.000     25
20.000     25
33.000     23
12.000     21
           ..
131.800     1
51.798      1
60.000      1
36.400      1
```

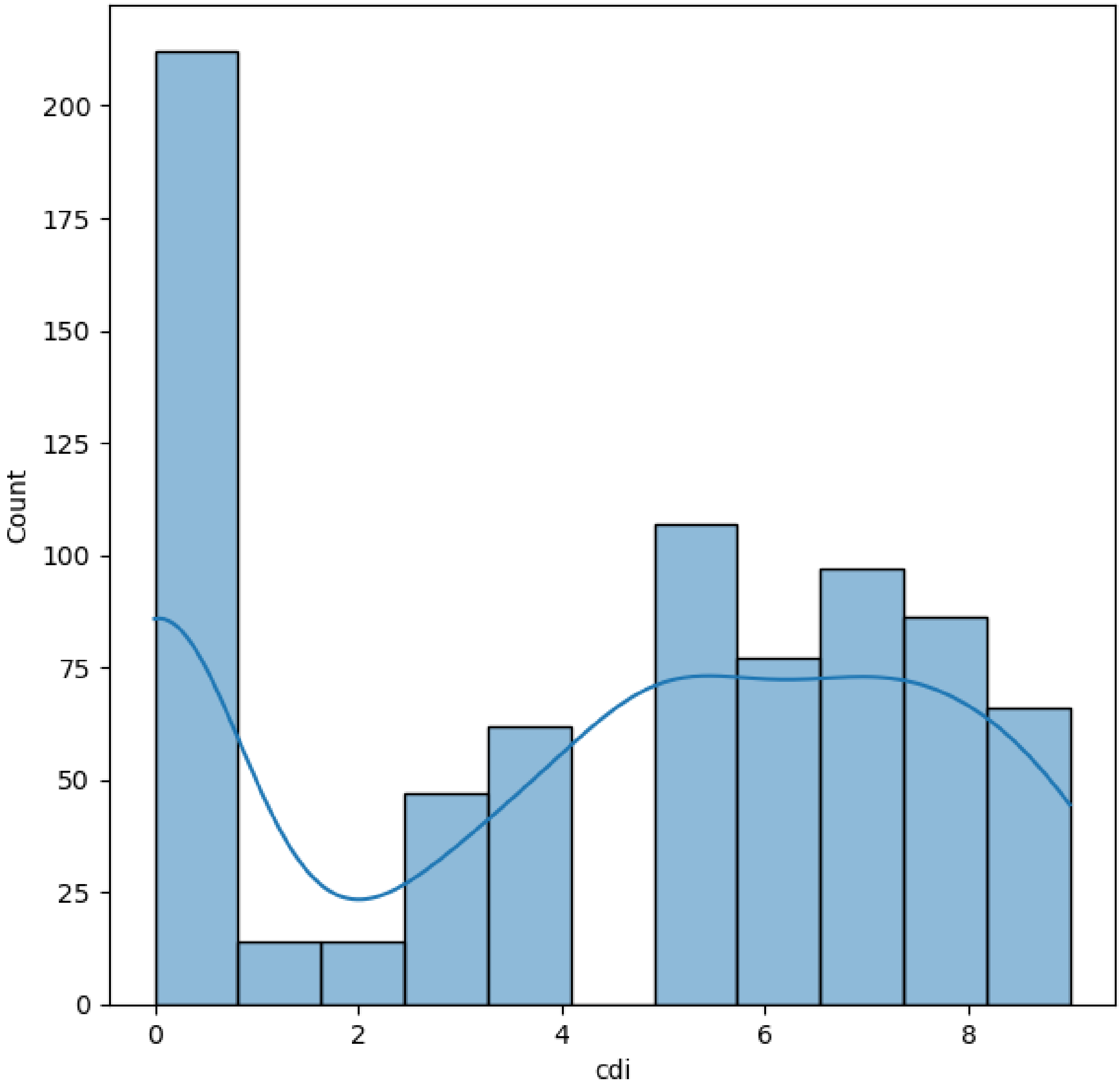
```
103.000     1
Name: count, Length: 303, dtype: int64
latitude
 52.5020    2
 52.4800    2
 38.2760    2
 1.2710     2
-9.7963     1
           ..
 13.0490    1
 56.7744    1
-14.9280    1
 6.6310     1
-17.5430    1
Name: count, Length: 778, dtype: int64
longitude
 168.143    2
-167.736    2
-168.080    2
 168.892    2
 107.419    2
           ..
-88.660     1
-153.281    1
 167.170    1
 126.899    1
-71.649     1
Name: count, Length: 777, dtype: int64
Year
2013    53
2015    53
2014    48
2016    43
2018    43
2021    42
2010    41
2022    40
2007    37
2017    36
2011    34
2019    33
2004    32
2003    31
2012    31
2001    28
2005    28
2020    27
2009    26
2006    26
```

```
2008    25
2002    25
Name: count, dtype: int64
Month
11    80
9     80
4     77
1     70
10    69
8     68
3     63
2     63
5     58
7     56
12    56
6     42
Name: count, dtype: int64
tsunami
0    478
1    304
Name: count, dtype: int64
```

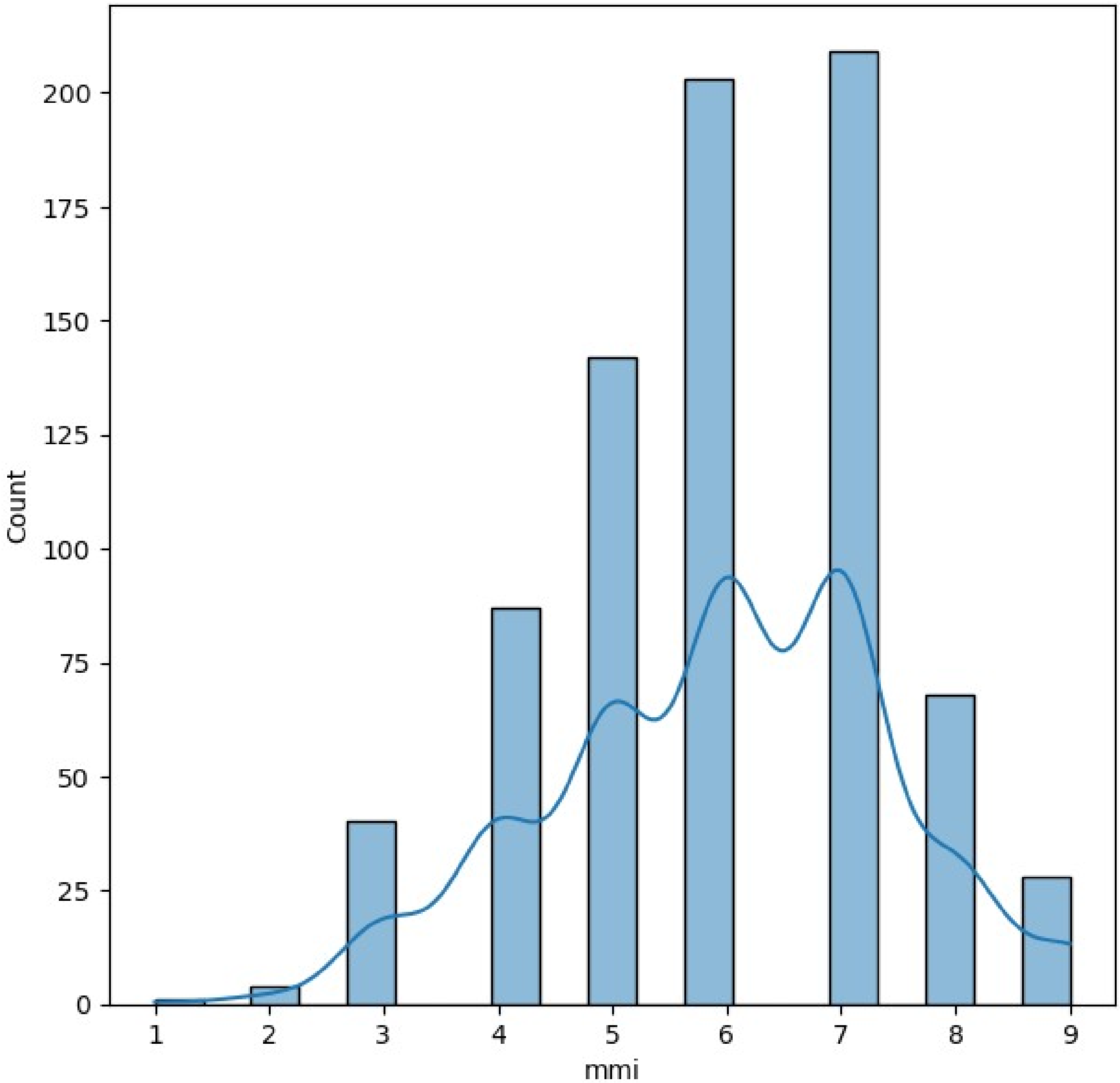
```
cols=['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap', 'depth',
       'latitude', 'longitude', 'Year', 'Month']
for col in cols:
    plt.figure(figsize=(7,7))
    sns.histplot(x=df[col],kde=True)
```



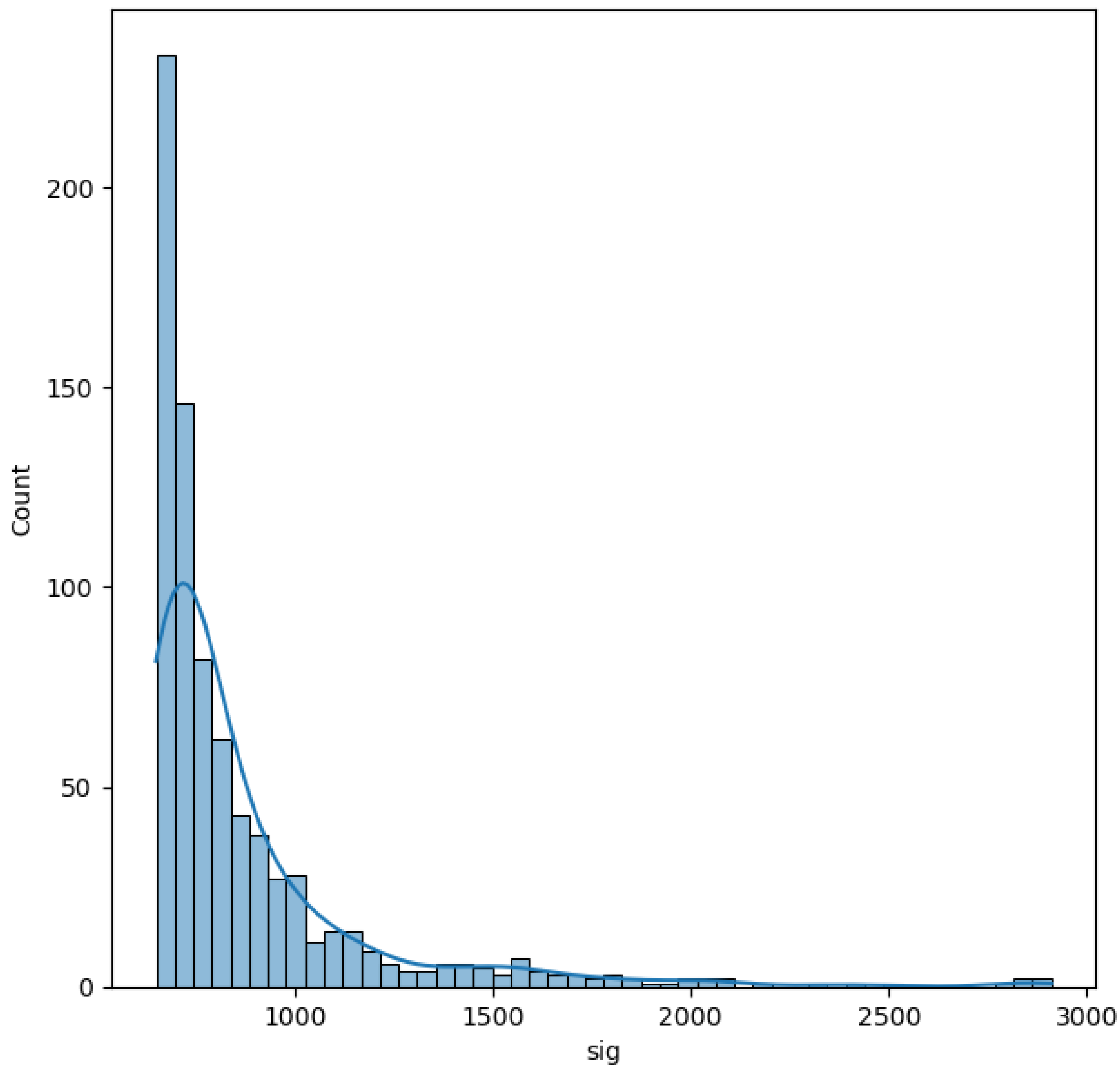
250
200
150
100
50
0
Count
6.5
7.0
7.5
8.0
8.5
9.0
magnitude



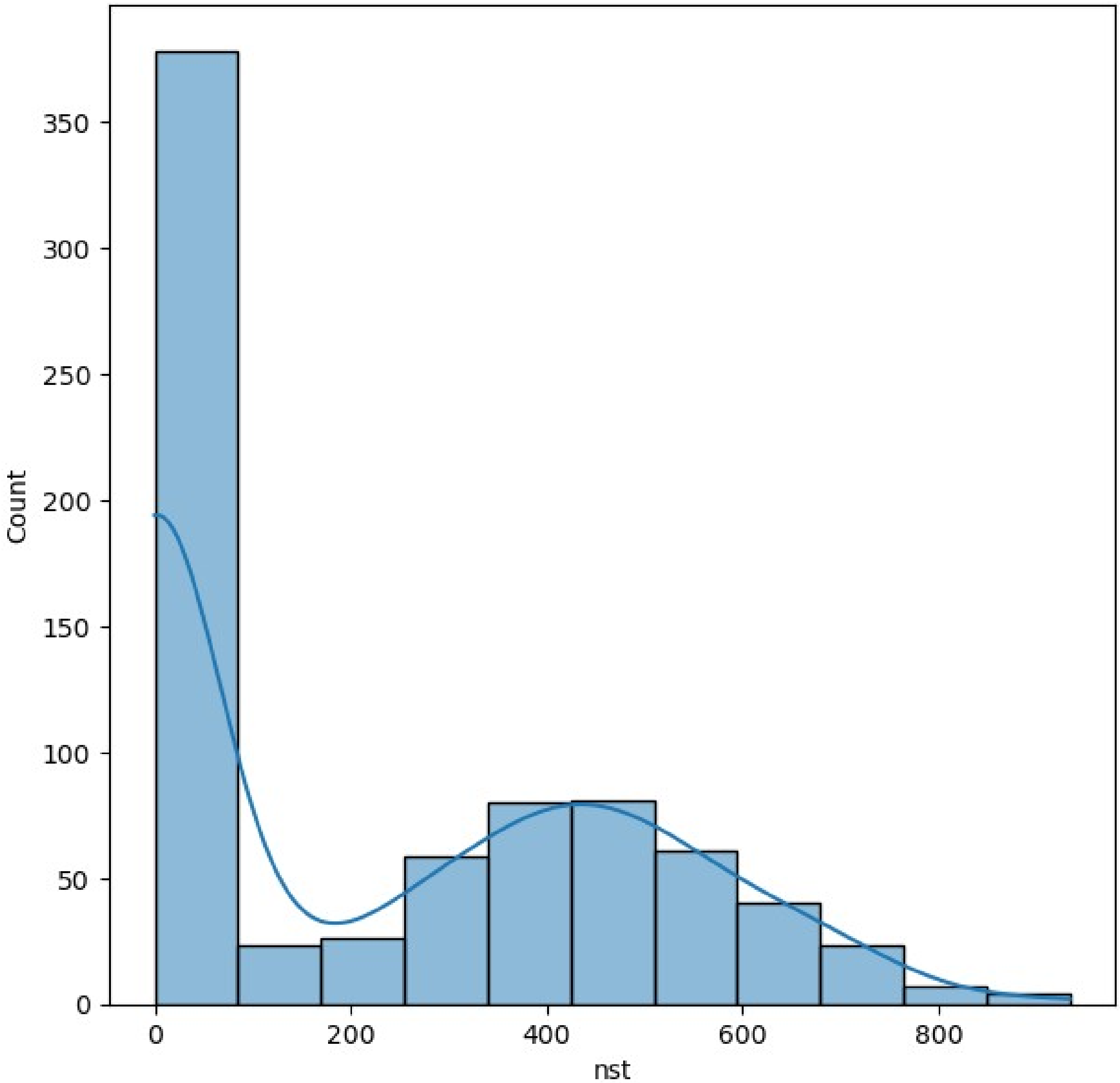
200
175
150
125
100
75
50
25
0
Count
0
2
4
6
8
cdi



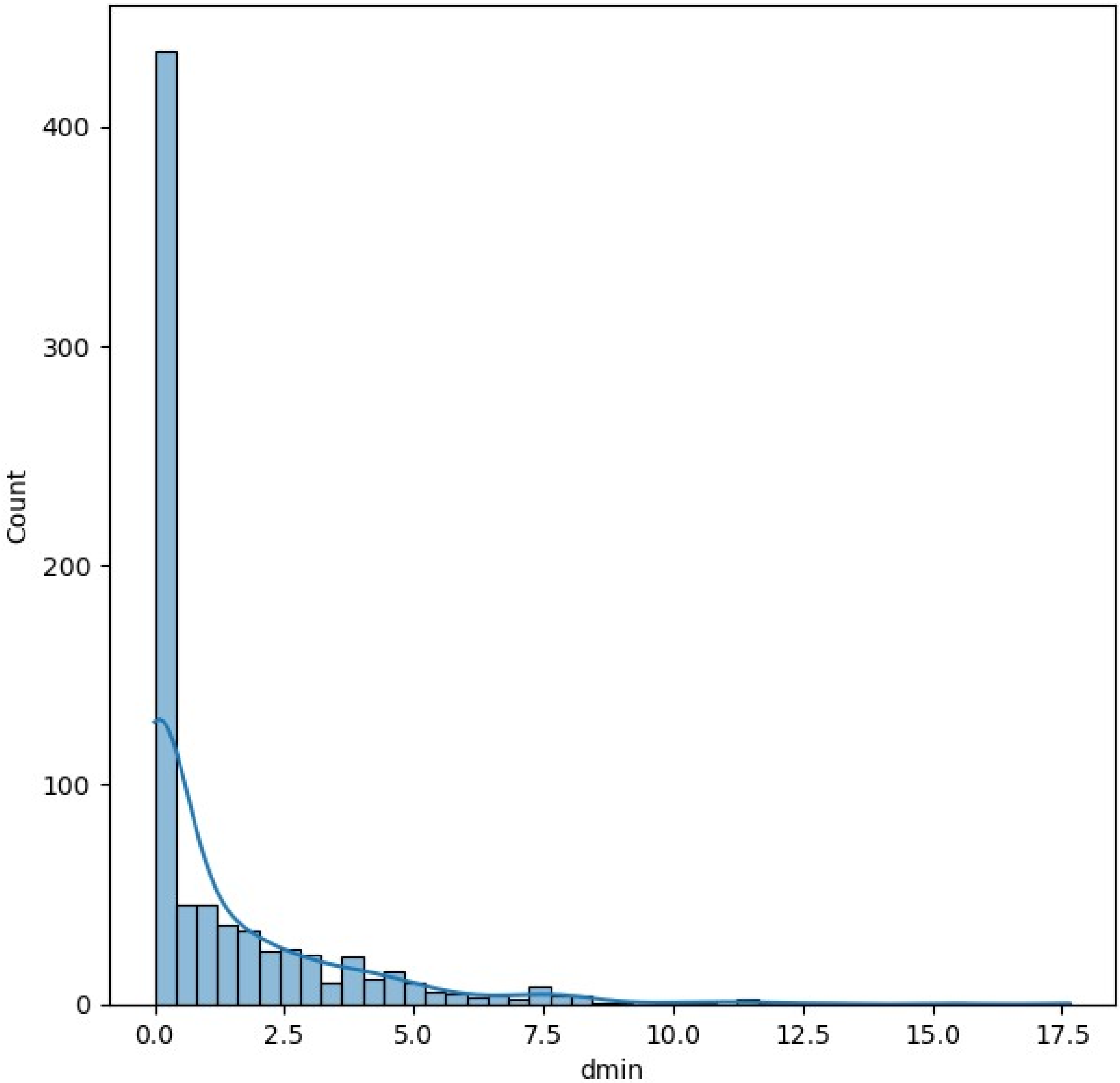
Count
0
25
50
75
100
125
150
175
200
1
2
3
4
5
6
7
8
9
mmi

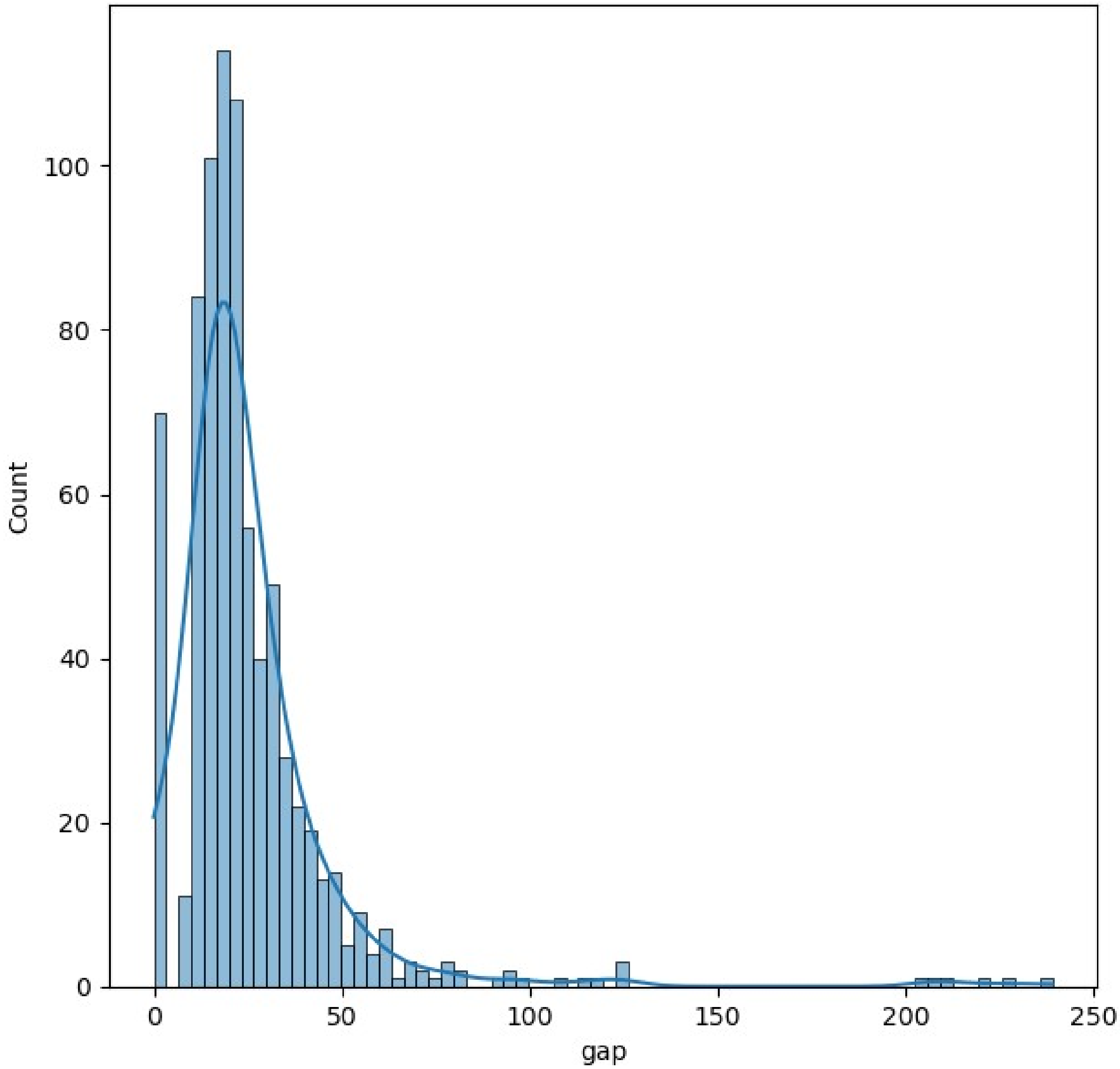


Count
200
150
100
50
0
1000
1500
2000
2500
3000
sig

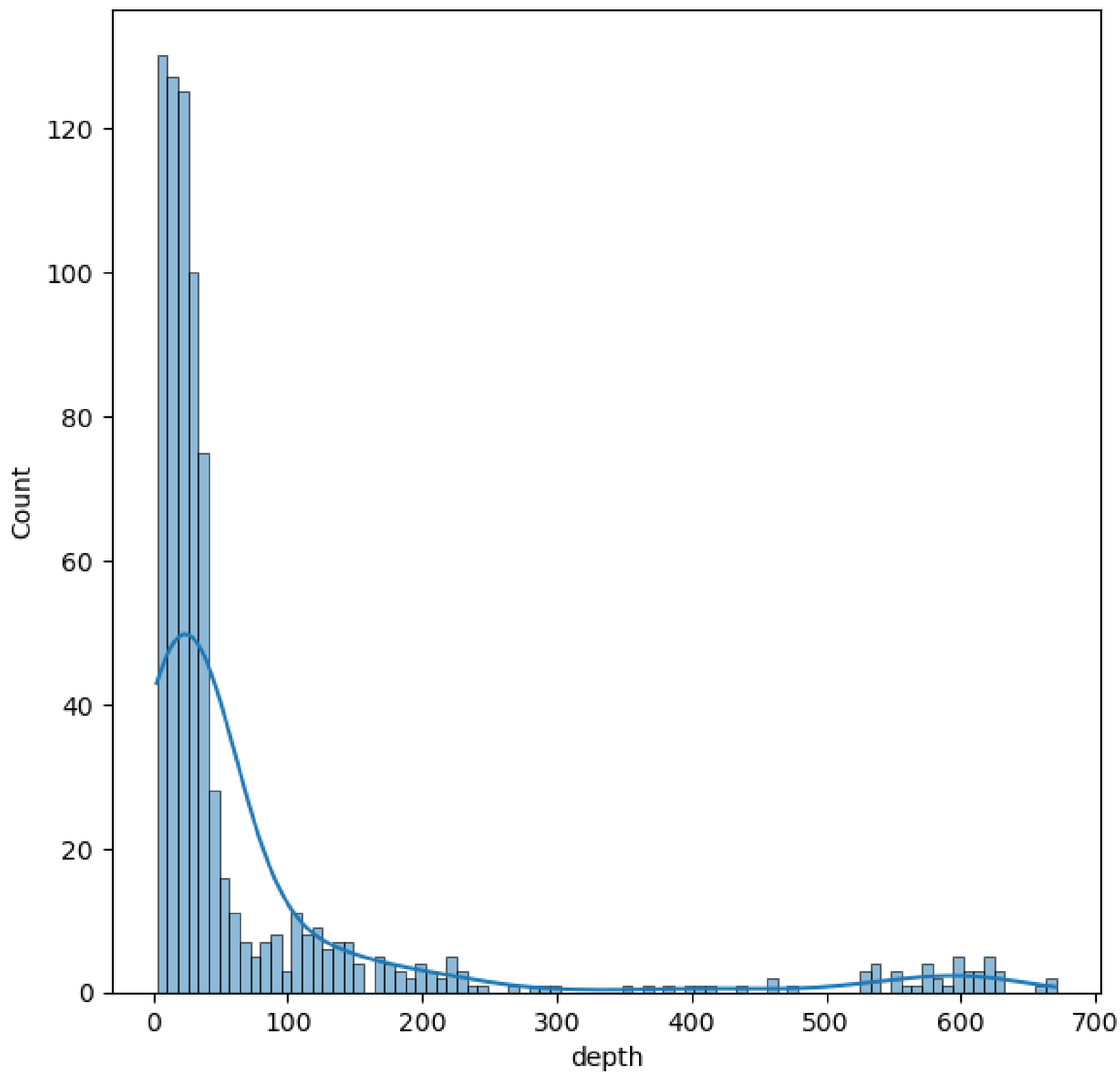


Count
nst
0
50
100
150
200
250
300
350
0
200
400
600
800

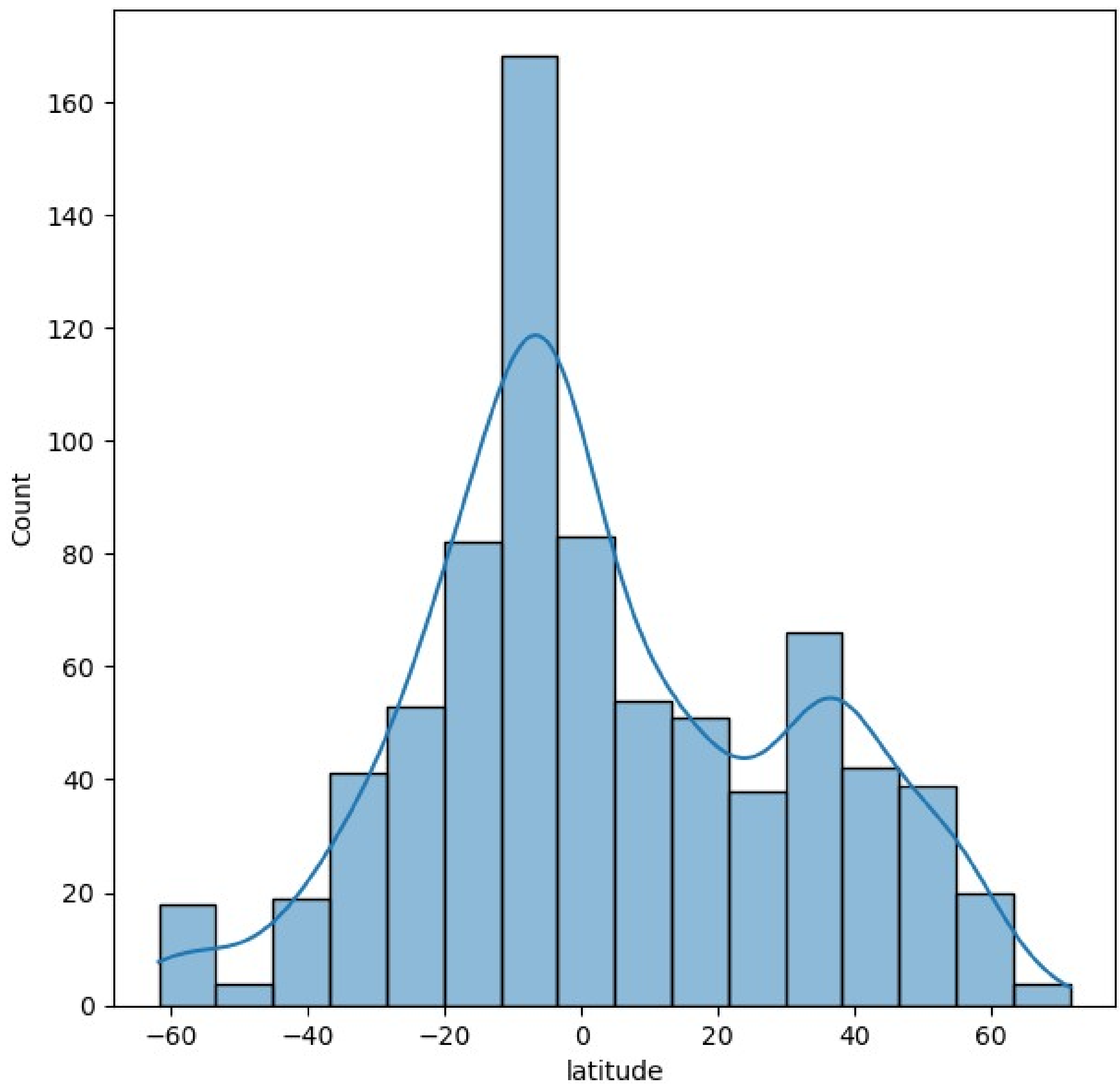

400
300
200
100
0
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
dmin



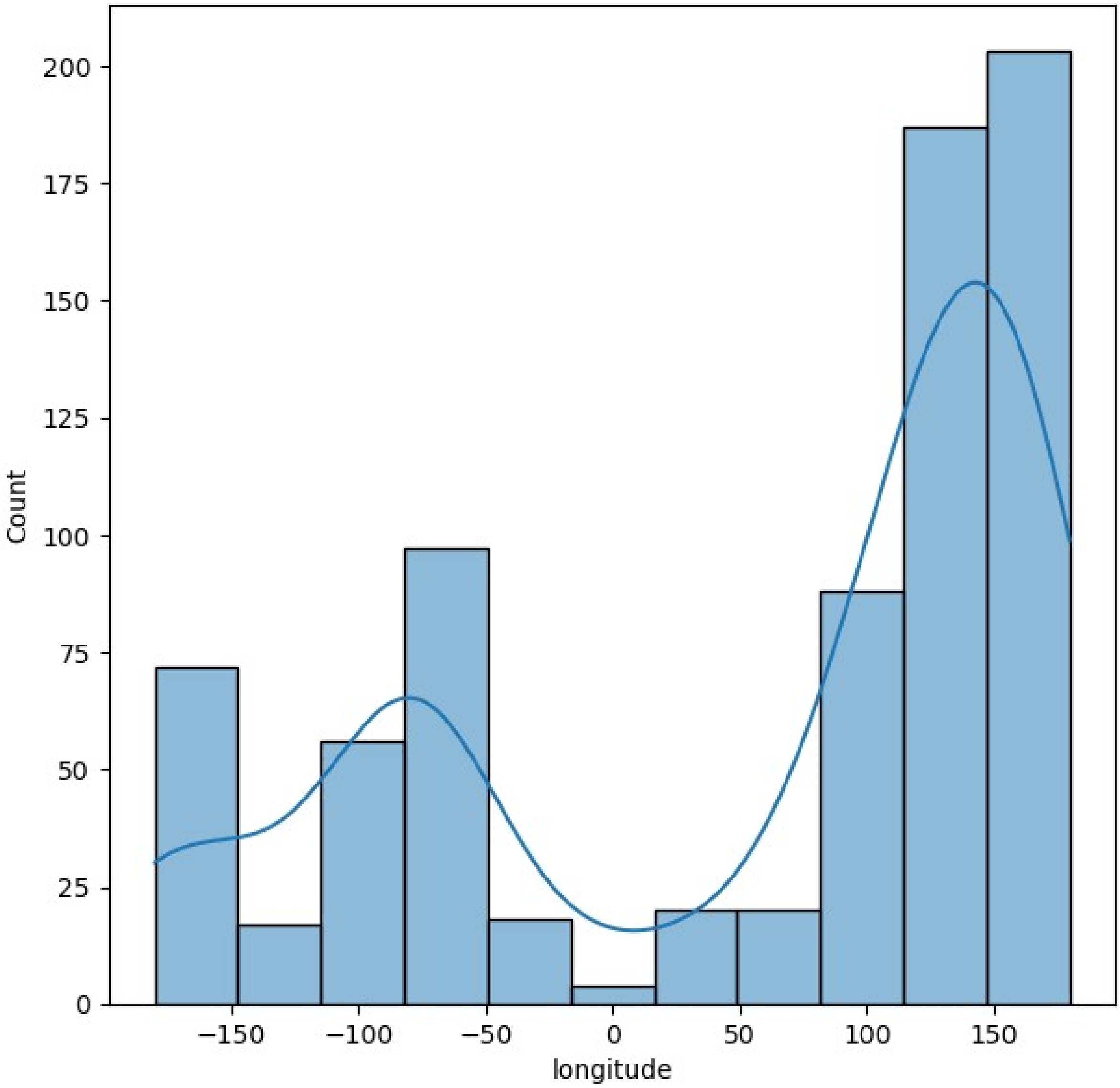
Count
gap
0
20
40
60
80
100
0
50
100
150
200
250



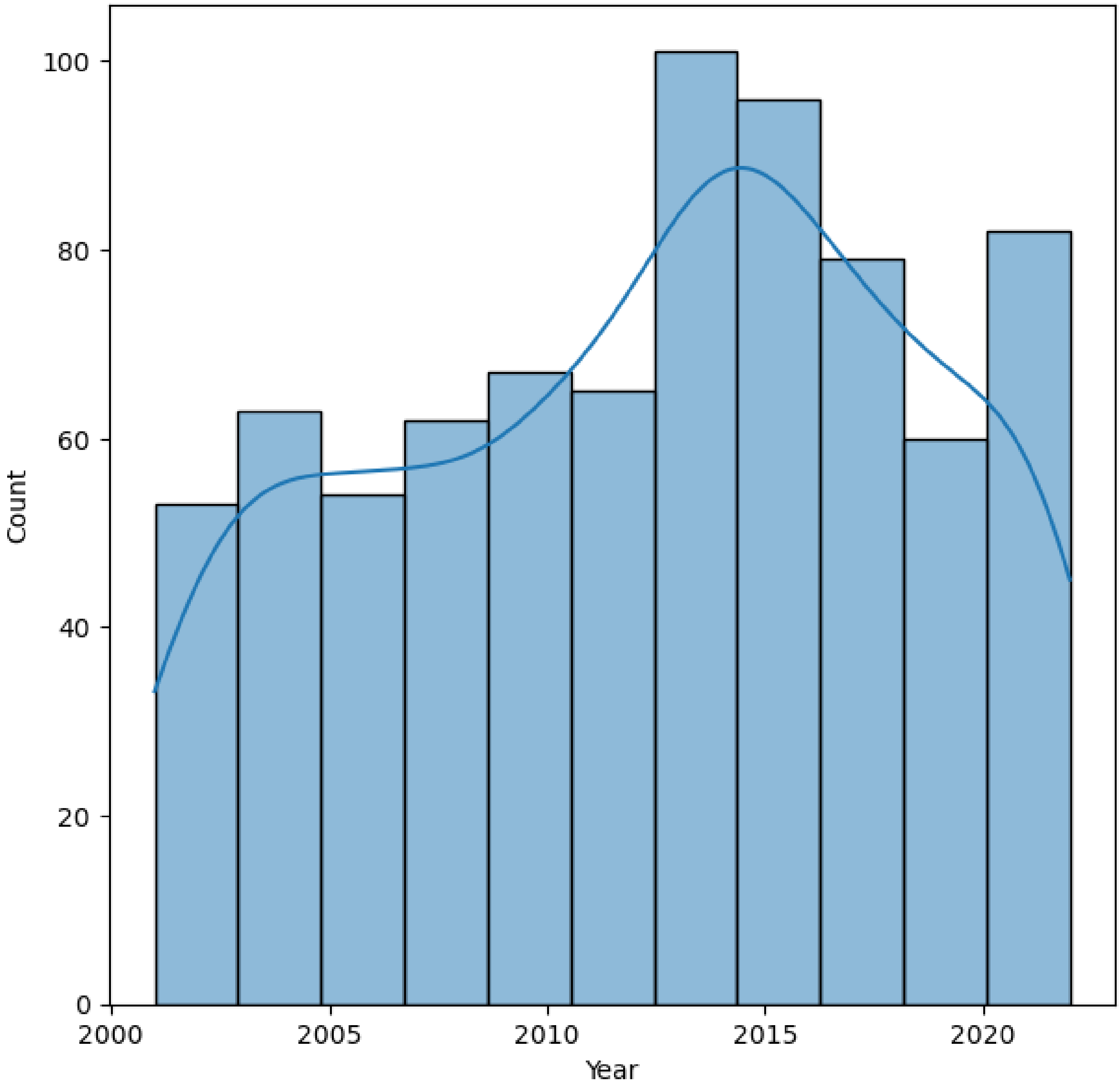
Count
depth
0
20
40
60
80
100
120
0
100
200
300
400
500
600
700



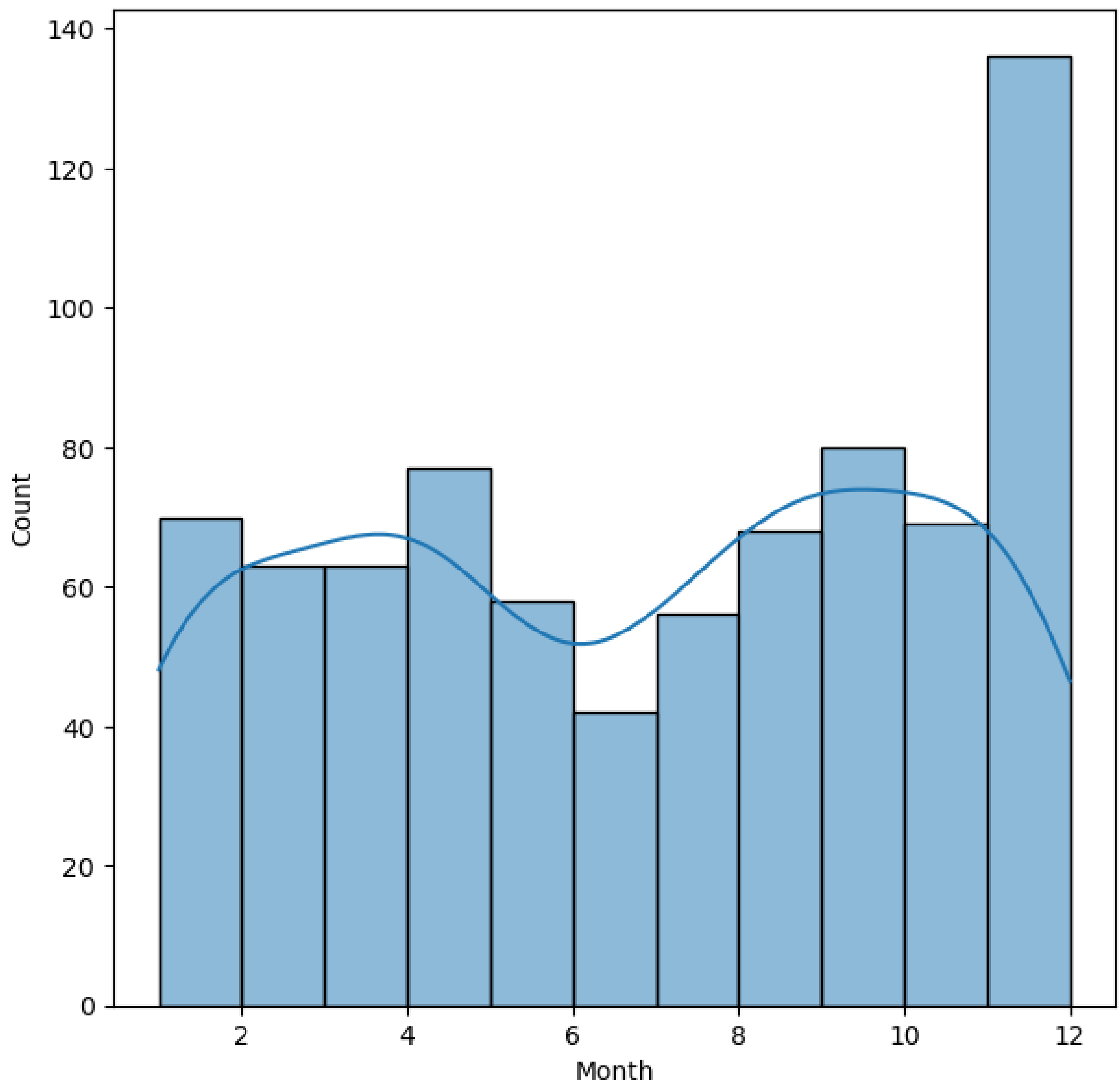
Count
0
20
40
60
80
100
120
140
160
−60
−40
−20
0
20
40
60
latitude



Count
longitude
200
175
150
125
100
75
50
25
0
−150
−100
−50
0
50
100
150

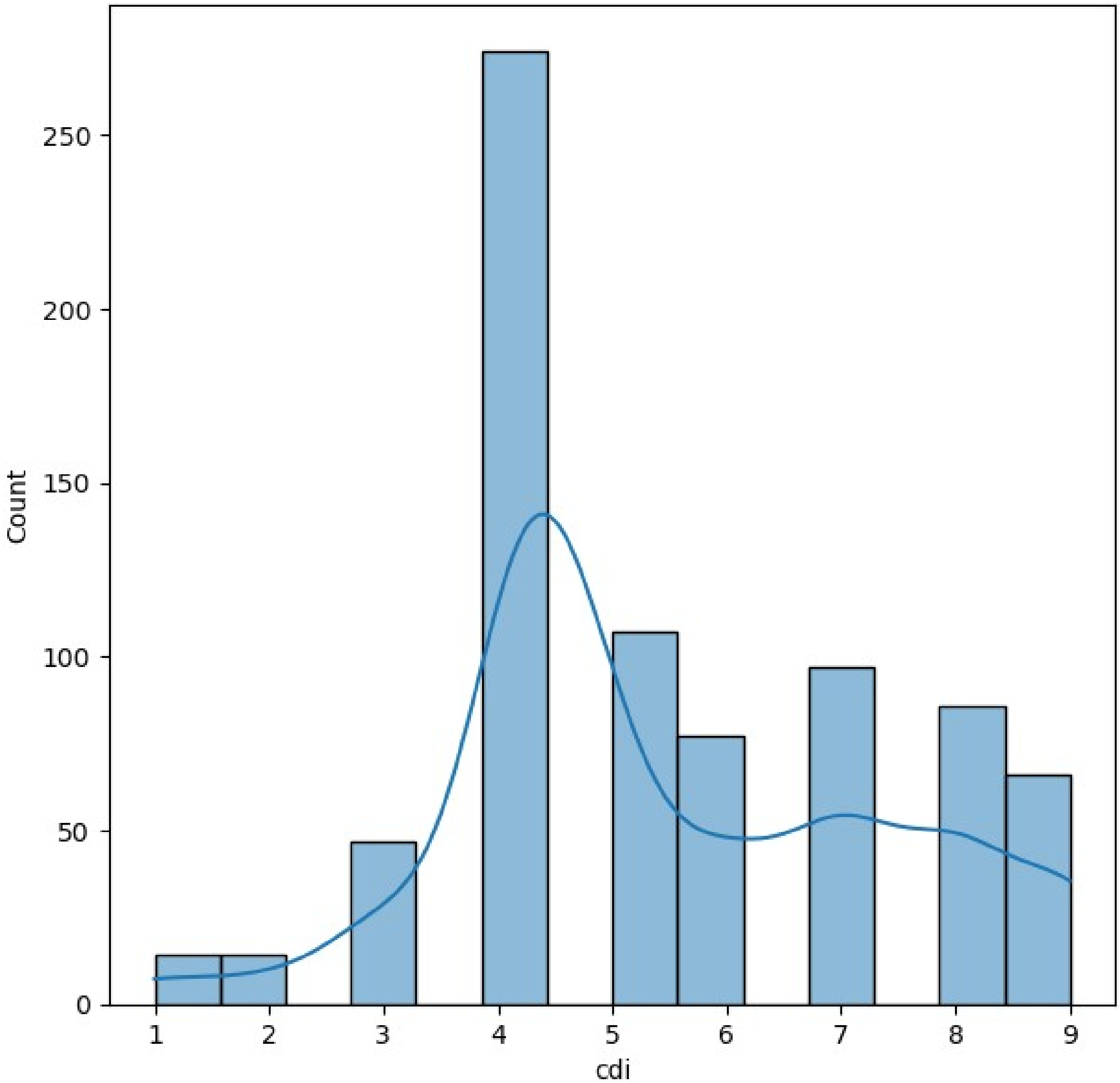


100
80
60
40
20
0
Count
2000
2005
2010
2015
2020
Year



caping

```
df['cdi'].min()

0

cdi_mean=df['cdi'].mean()

df['cdi']=df['cdi'].replace(0,cdi_mean)

plt.figure(figsize=(7,7))
sns.histplot(x=df['cdi'],kde=True)

<Axes: xlabel='cdi', ylabel='Count'>
```




```
df['sig'].min()
```

```
650
```

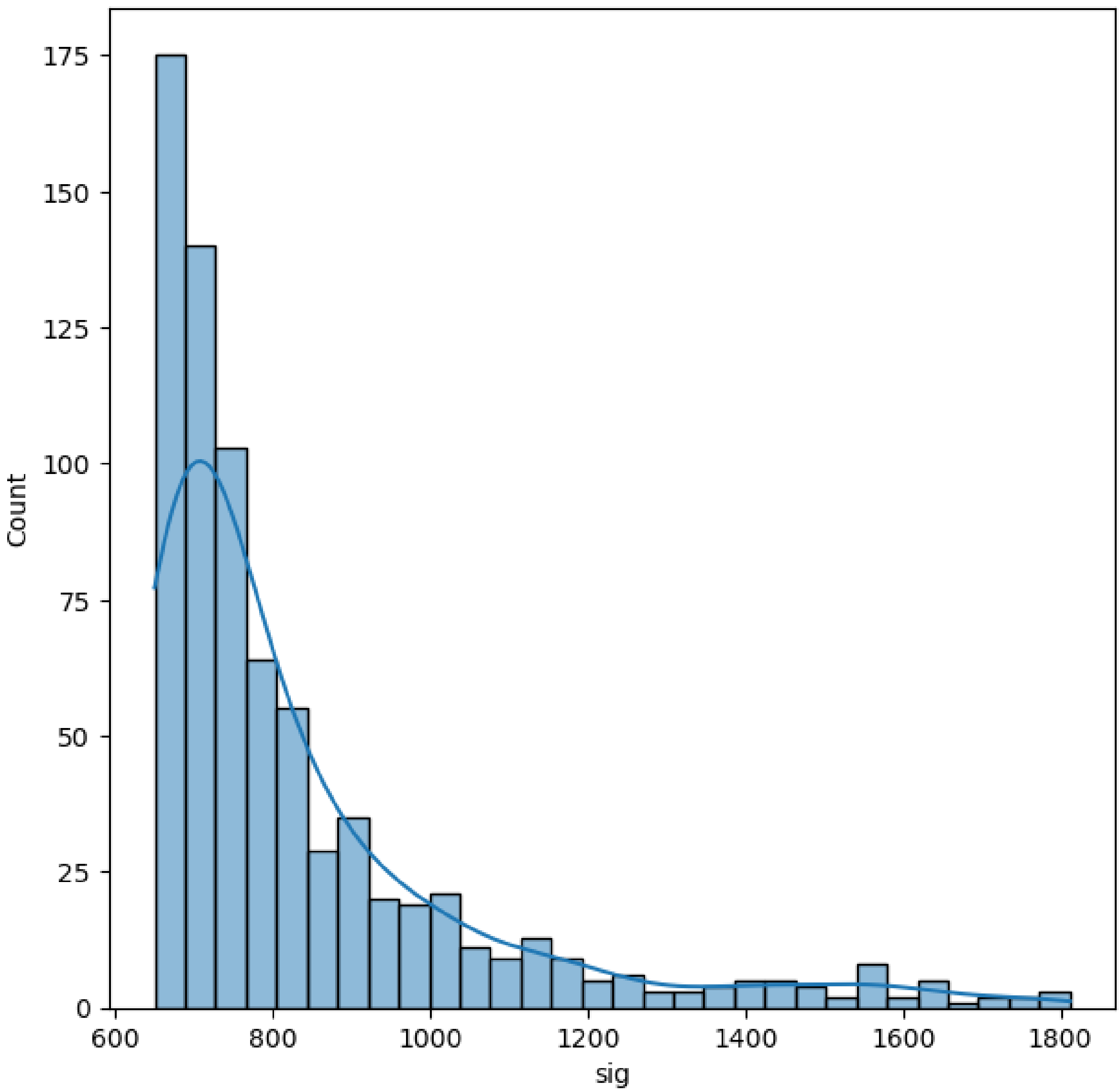
```
df['sig'].value_counts().sort_values(ascending=False)
```

```
sig
650     50
670     41
691     36
711     25
776     18
        ..
824      1
1360     1
```

```
1024      1
1750      1
1441      1
Name: count, Length: 339, dtype: int64
```

```
z_scores = (df['sig'] - df['sig'].mean()) / df['sig'].std()

df = df[(np.abs(z_scores) < 3)]

plt.figure(figsize=(7,7))
sns.histplot(x=df['sig'],kde=True,bins=30)
```

```
<Axes: xlabel='sig', ylabel='Count'>
```




```
df['sig'].value_counts().sort_values(ascending=False)
```

```
sig
650     50
670     41
691     36
711     25
776     18
        ..
1046     1
848      1
1560     1
914      1
1248     1
Name: count, Length: 322, dtype: int64
```

```
df['gap'].min()
```

```
0.0
```

```
df['gap'].min()
```

```
0.0
```

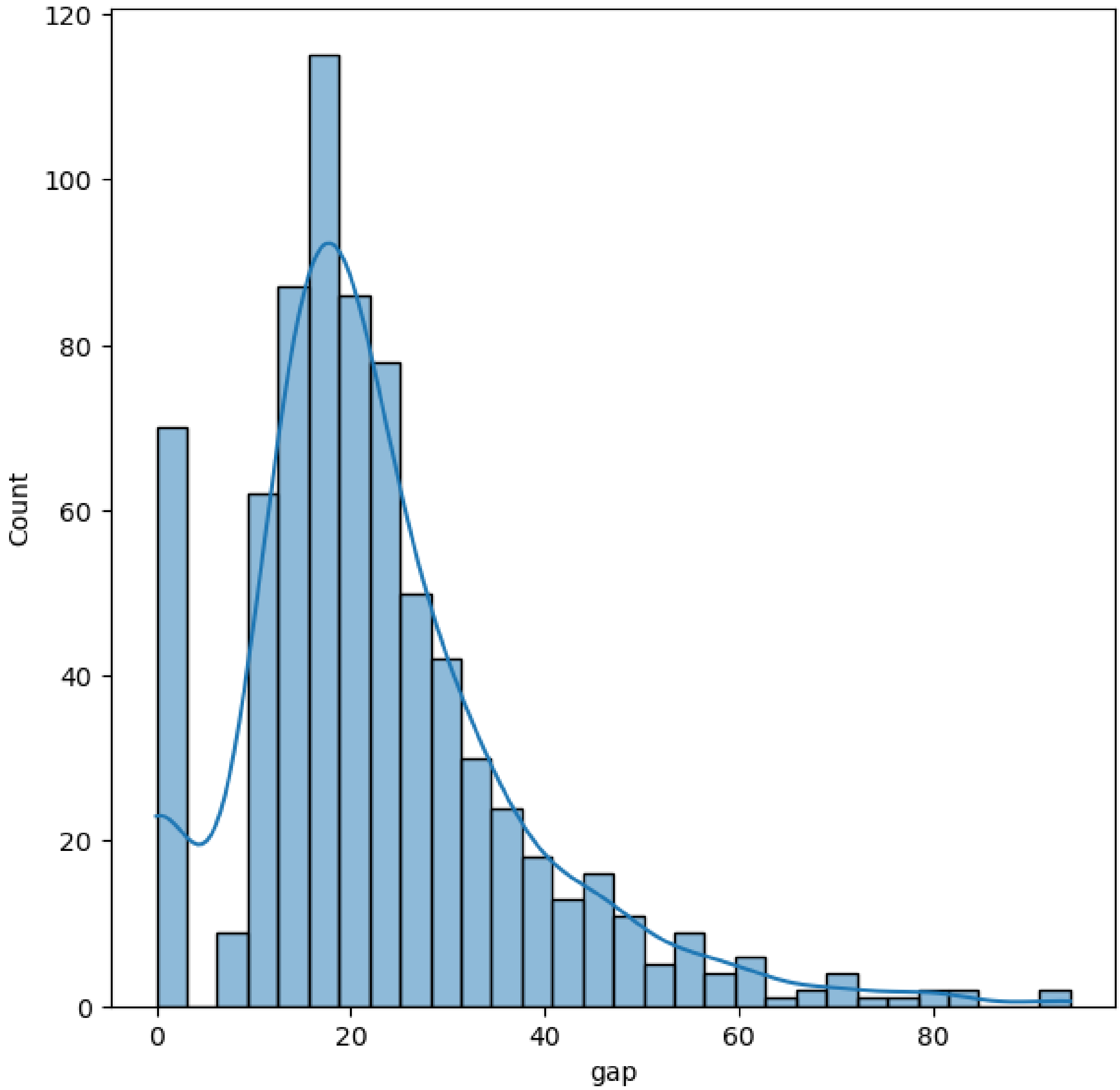
```
df['gap'].value_counts().sort_values(ascending=False)
```

```
gap
0.0     70
18.0    23
16.0    22
22.0    21
12.0    19
        ..
26.7     1
19.1     1
19.4     1
28.7     1
38.4     1
Name: count, Length: 253, dtype: int64
```

```
z_scores = (df['gap'] - df['gap'].mean()) / df['gap'].std()
```

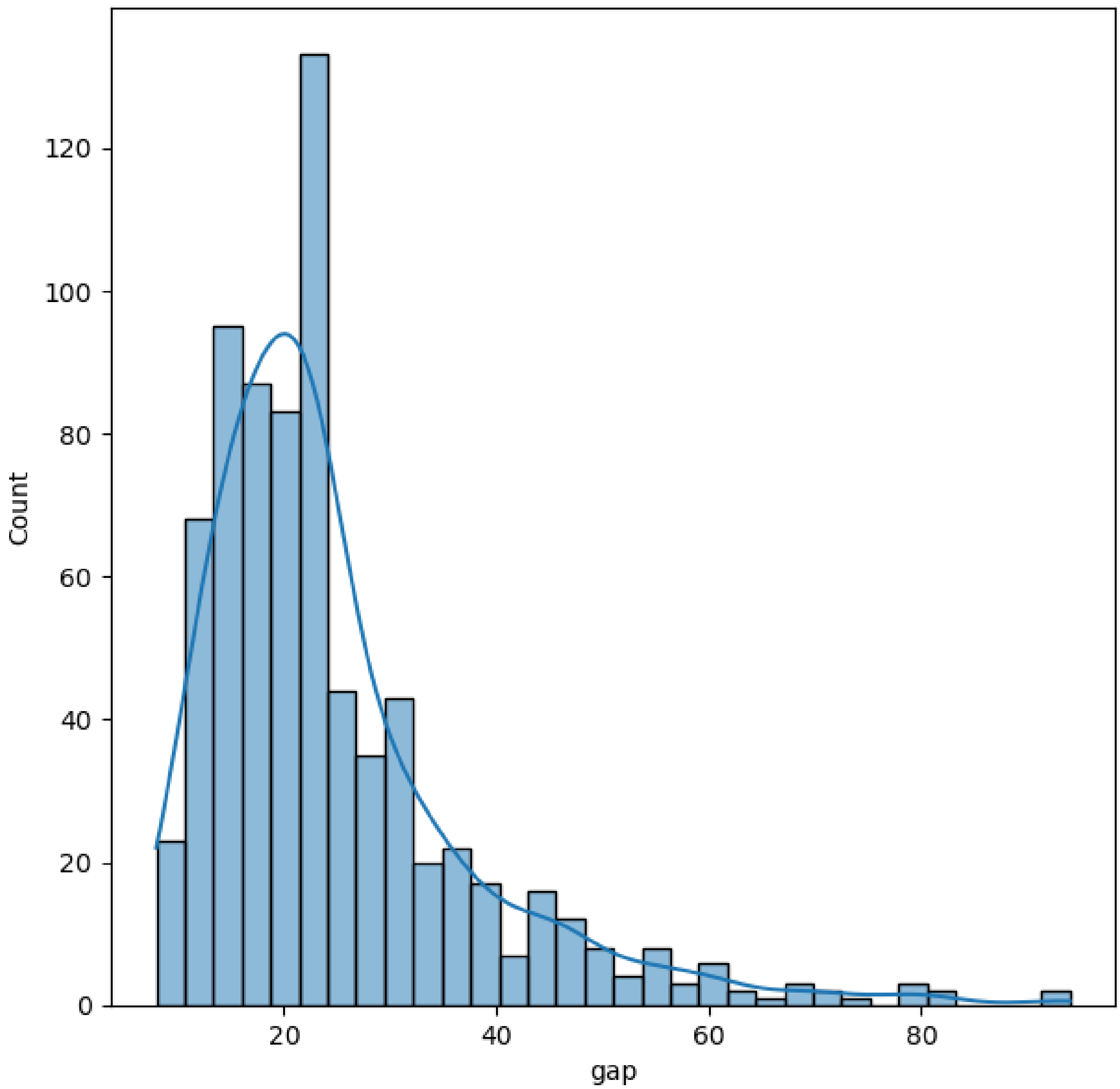
```
df = df[(np.abs(z_scores) < 3)]
plt.figure(figsize=(7,7))
sns.histplot(x=df['gap'],kde=True)
```

```
<Axes: xlabel='gap', ylabel='Count'>
```

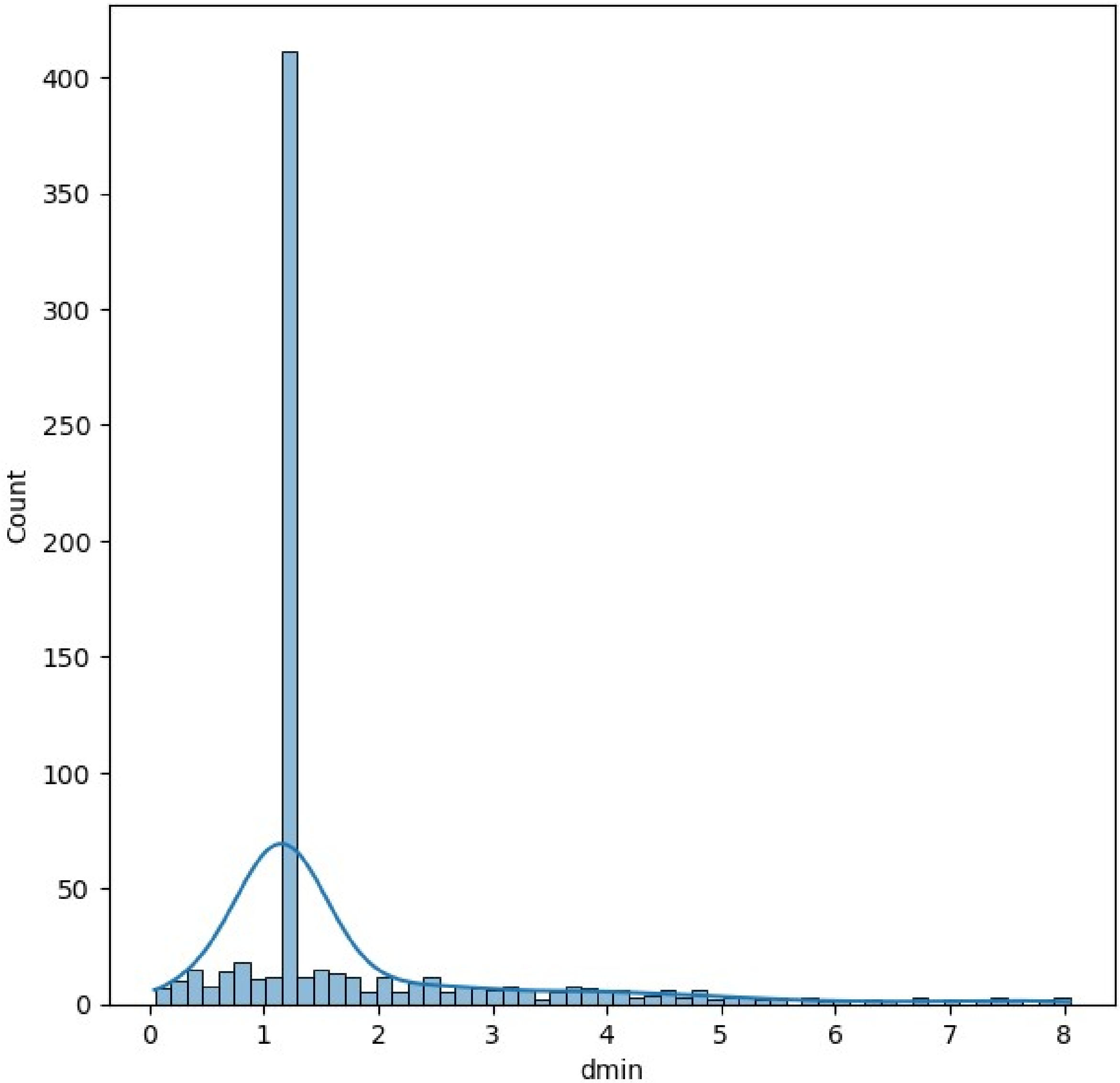



```
df['gap']=df['gap'].replace(0,df['gap'].mean())

plt.figure(figsize=(7,7))
sns.histplot(x=df['gap'],kde=True)
```

```
<Axes: xlabel='gap', ylabel='Count'>
```



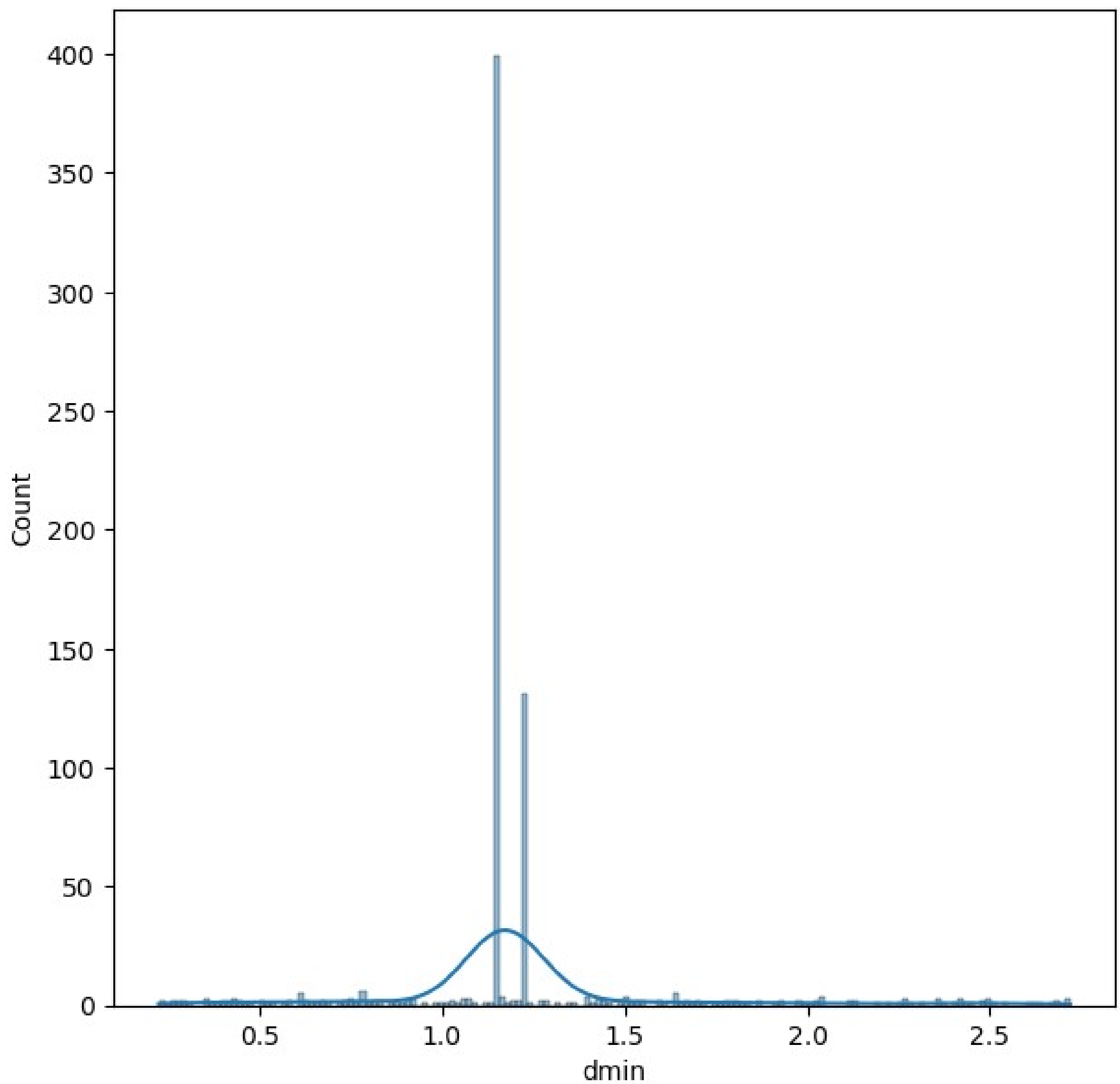

```
z_scores = (df['dmin'] - df['dmin'].mean()) / df['dmin'].std()

df = df[(np.abs(z_scores) < 3)]
df['dmin']=df['dmin'].replace(0,df['dmin'].mean())
plt.figure(figsize=(7,7))
sns.histplot(x=df['dmin'],kde=True)

<Axes: xlabel='dmin', ylabel='Count'>
```




```
df['dmin'].mean()

np.float64(1.779944720507443)

df['dmin'].median()

1.157278546195652

df['dmin'].dtype

dtype('float64')

df = df[df['dmin'] != 0]
Q1 = df['dmin'].quantile(0.25)
Q3 = df['dmin'].quantile(0.75)
IQR = Q3 - Q1
```

```python
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Compute mean of non-outlier values
mean_value = df.loc[(df['dmin'] >= lower_bound) & (df['dmin'] <=
upper_bound), 'dmin'].mean()

# Replace outliers with mean
df['dmin'] = np.where((df['dmin'] < lower_bound) | (df['dmin'] >
upper_bound),
                       mean_value, df['dmin'])

plt.figure(figsize=(7,7))
sns.histplot(x=df['dmin'],kde=True)
```

```
<Axes: xlabel='dmin', ylabel='Count'>
```




```
mean_nst=df['nst'].mean()

counts, bins = np.histogram(df['nst'], bins=30)  # change bins= as
needed

# Show all ranges and their counts
for i in range(len(counts)):
    print(f"Bin {i+1}: Range ({bins[i]:.2f}, {bins[i+1]:.2f}) ->
Count: {counts[i]}")
```

```
Bin 1: Range (0.00, 31.13) -> Count: 336
Bin 2: Range (31.13, 62.27) -> Count: 0
Bin 3: Range (62.27, 93.40) -> Count: 5
Bin 4: Range (93.40, 124.53) -> Count: 5
Bin 5: Range (124.53, 155.67) -> Count: 12
```

```
Bin 6: Range (155.67, 186.80) -> Count: 7
Bin 7: Range (186.80, 217.93) -> Count: 5
Bin 8: Range (217.93, 249.07) -> Count: 13
Bin 9: Range (249.07, 280.20) -> Count: 20
Bin 10: Range (280.20, 311.33) -> Count: 20
Bin 11: Range (311.33, 342.47) -> Count: 26
Bin 12: Range (342.47, 373.60) -> Count: 21
Bin 13: Range (373.60, 404.73) -> Count: 33
Bin 14: Range (404.73, 435.87) -> Count: 35
Bin 15: Range (435.87, 467.00) -> Count: 29
Bin 16: Range (467.00, 498.13) -> Count: 27
Bin 17: Range (498.13, 529.27) -> Count: 28
Bin 18: Range (529.27, 560.40) -> Count: 21
Bin 19: Range (560.40, 591.53) -> Count: 18
Bin 20: Range (591.53, 622.67) -> Count: 14
Bin 21: Range (622.67, 653.80) -> Count: 19
Bin 22: Range (653.80, 684.93) -> Count: 10
Bin 23: Range (684.93, 716.07) -> Count: 16
Bin 24: Range (716.07, 747.20) -> Count: 4
Bin 25: Range (747.20, 778.33) -> Count: 4
Bin 26: Range (778.33, 809.47) -> Count: 4
Bin 27: Range (809.47, 840.60) -> Count: 0
Bin 28: Range (840.60, 871.73) -> Count: 1
Bin 29: Range (871.73, 902.87) -> Count: 0
Bin 30: Range (902.87, 934.00) -> Count: 3
```

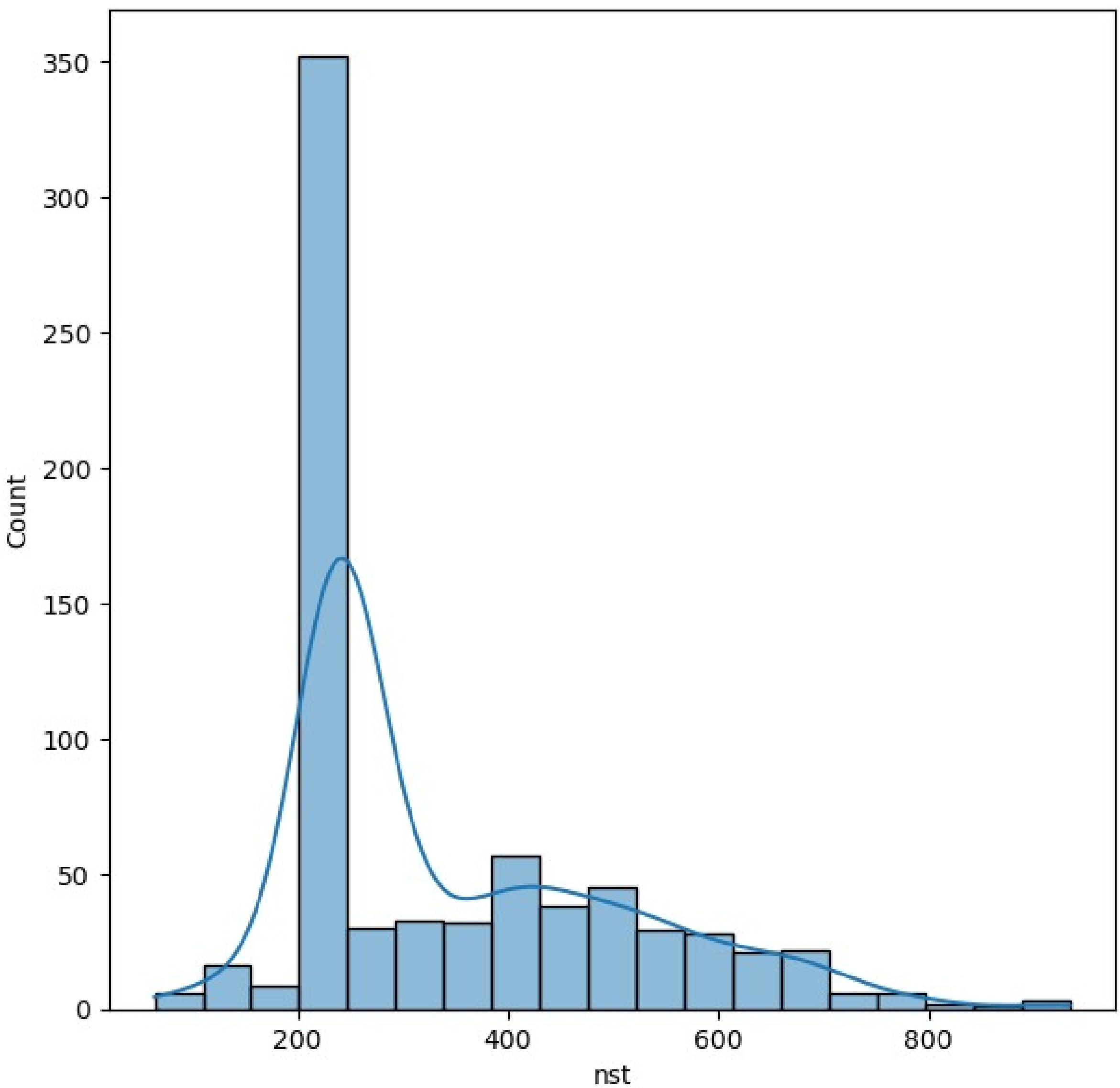
```
df['nst']=np.where((df['nst'] <= 31.13) & (df['nst'] >= 0),mean_nst
                       , df['nst'])

counts, bins = np.histogram(df['nst'], bins=30)  # change bins= as
needed

# Show all ranges and their counts
for i in range(len(counts)):
    print(f"Bin {i+1}: Range ({bins[i]:.2f}, {bins[i+1]:.2f}) ->
Count: {counts[i]}")
```

```
Bin 1: Range (64.00, 93.00) -> Count: 5
Bin 2: Range (93.00, 122.00) -> Count: 5
Bin 3: Range (122.00, 151.00) -> Count: 11
Bin 4: Range (151.00, 180.00) -> Count: 8
Bin 5: Range (180.00, 209.00) -> Count: 3
Bin 6: Range (209.00, 238.00) -> Count: 11
Bin 7: Range (238.00, 267.00) -> Count: 349
Bin 8: Range (267.00, 296.00) -> Count: 22
Bin 9: Range (296.00, 325.00) -> Count: 23
Bin 10: Range (325.00, 354.00) -> Count: 20
Bin 11: Range (354.00, 383.00) -> Count: 19
Bin 12: Range (383.00, 412.00) -> Count: 35
Bin 13: Range (412.00, 441.00) -> Count: 29
```

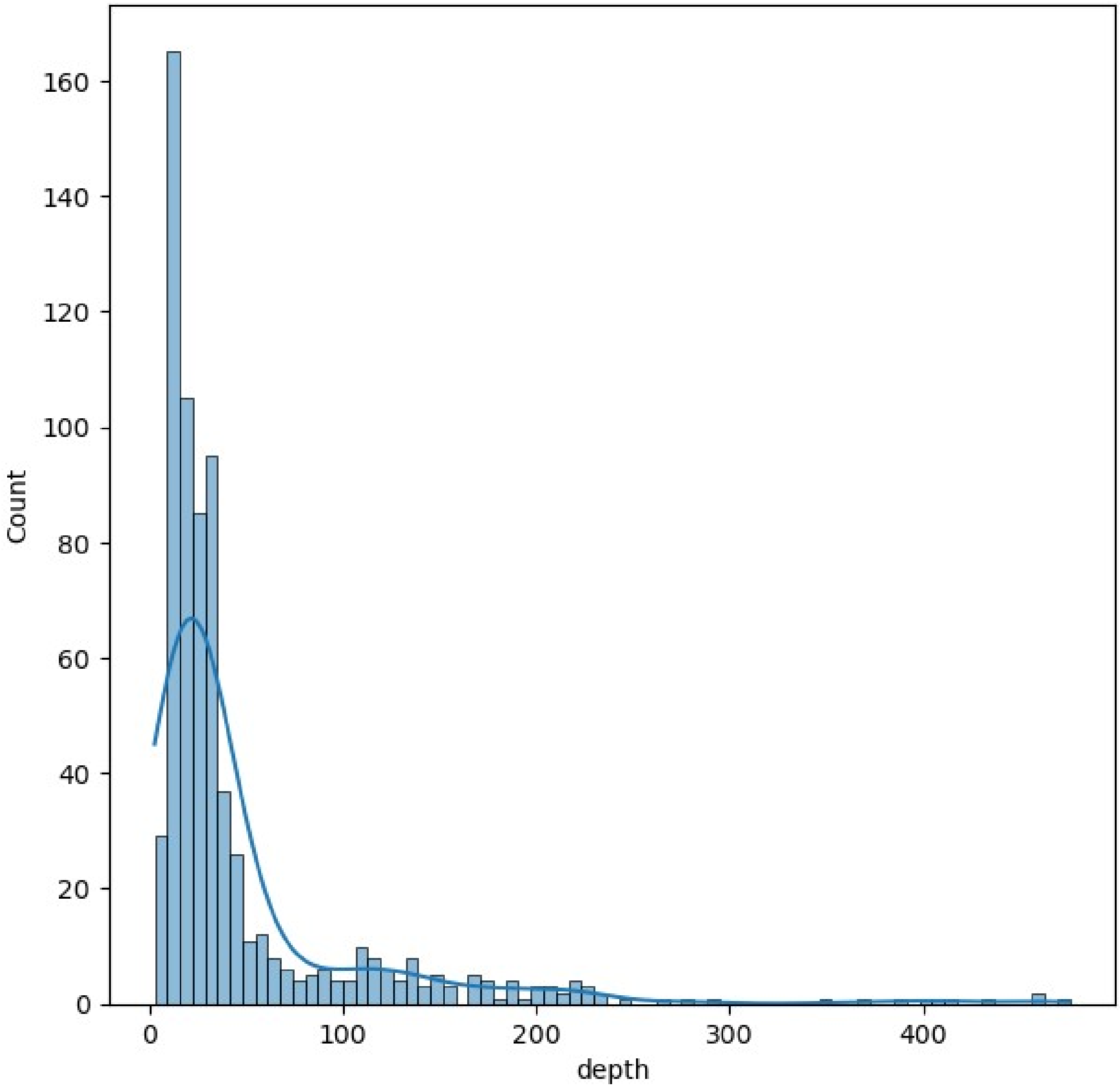
```
Bin 14: Range (441.00, 470.00) -> Count: 28
Bin 15: Range (470.00, 499.00) -> Count: 26
Bin 16: Range (499.00, 528.00) -> Count: 25
Bin 17: Range (528.00, 557.00) -> Count: 24
Bin 18: Range (557.00, 586.00) -> Count: 16
Bin 19: Range (586.00, 615.00) -> Count: 16
Bin 20: Range (615.00, 644.00) -> Count: 13
Bin 21: Range (644.00, 673.00) -> Count: 15
Bin 22: Range (673.00, 702.00) -> Count: 13
Bin 23: Range (702.00, 731.00) -> Count: 6
Bin 24: Range (731.00, 760.00) -> Count: 4
Bin 25: Range (760.00, 789.00) -> Count: 4
Bin 26: Range (789.00, 818.00) -> Count: 2
Bin 27: Range (818.00, 847.00) -> Count: 0
Bin 28: Range (847.00, 876.00) -> Count: 1
Bin 29: Range (876.00, 905.00) -> Count: 0
Bin 30: Range (905.00, 934.00) -> Count: 3
```

```python
plt.figure(figsize=(7,7))
sns.histplot(x=df['nst'],kde=True)
```

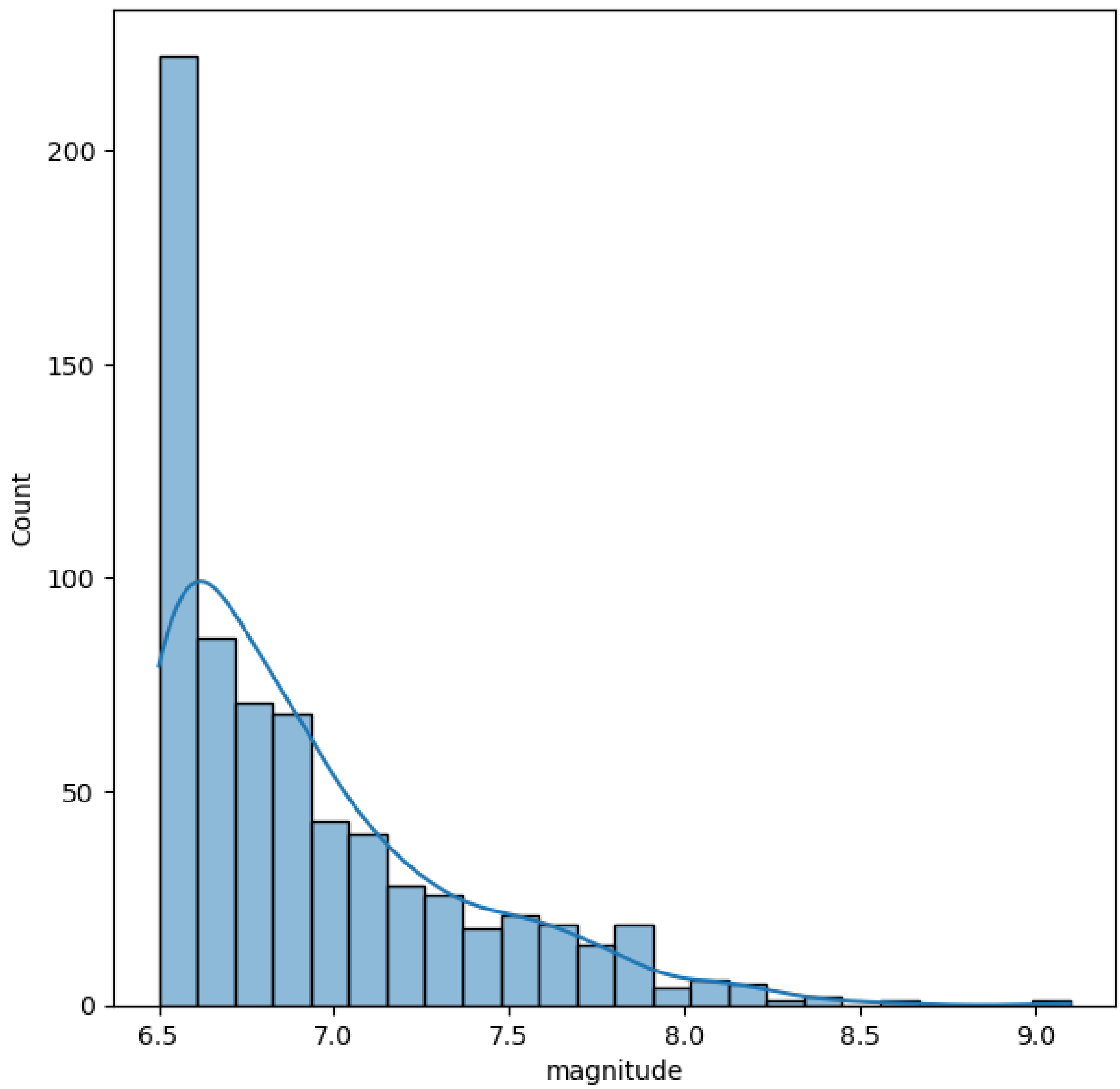
```
<Axes: xlabel='nst', ylabel='Count'>
```

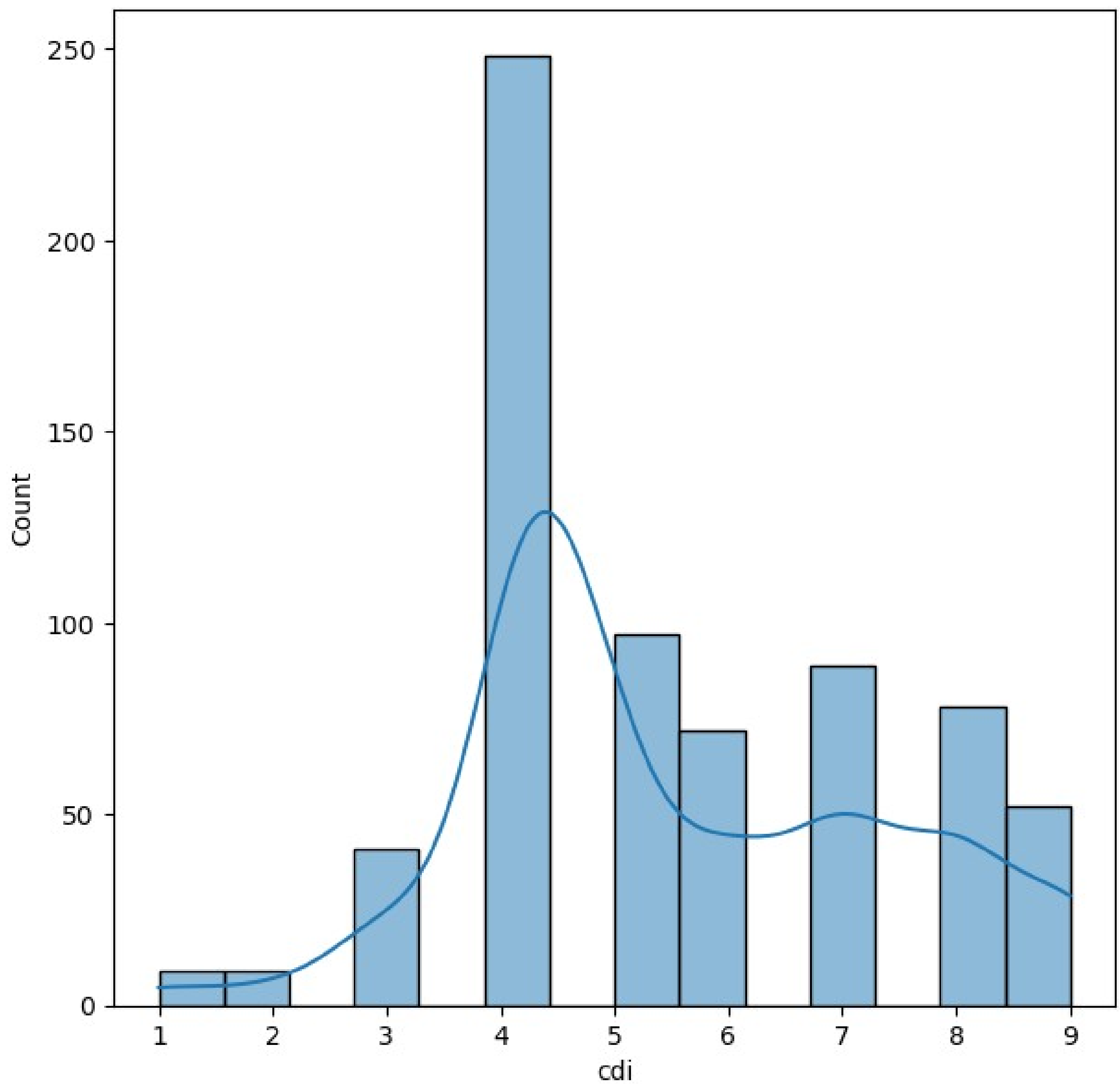


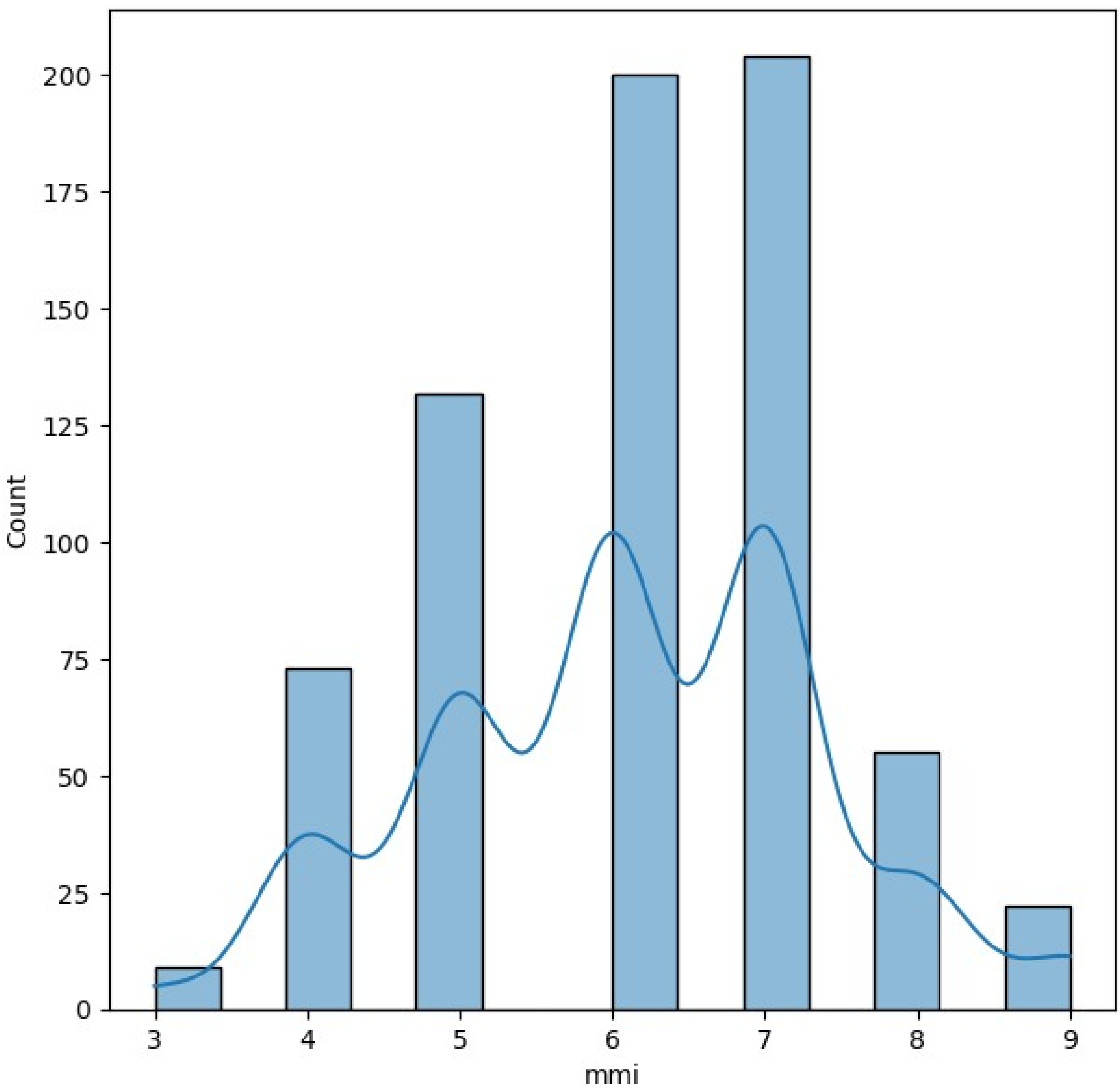

```
z_scores = (df['depth'] - df['depth'].mean()) / df['depth'].std()

df = df[(np.abs(z_scores) < 3)]
df['depth']=df['depth'].replace(0,df['depth'].mean())
plt.figure(figsize=(7,7))
sns.histplot(x=df['depth'],kde=True)

<Axes: xlabel='depth', ylabel='Count'>
```



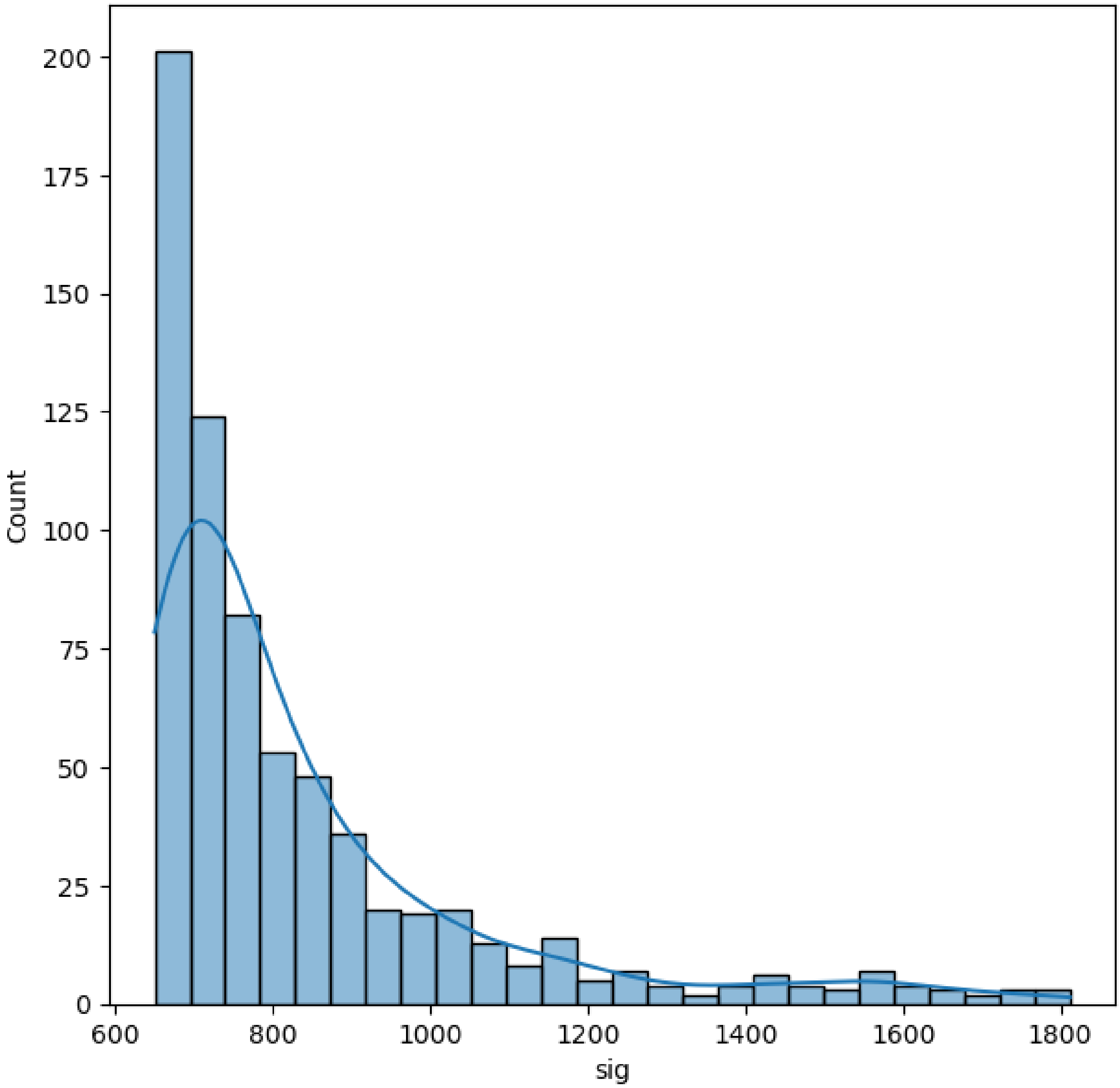

```
cols=['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap', 'depth',
       'latitude', 'longitude', 'Year', 'Month']
for col in cols:
    plt.figure(figsize=(7,7))
    sns.histplot(x=df[col],kde=True)
```


Count
0
50
100
150
200
6.5
7.0
7.5
8.0
8.5
9.0
magnitude

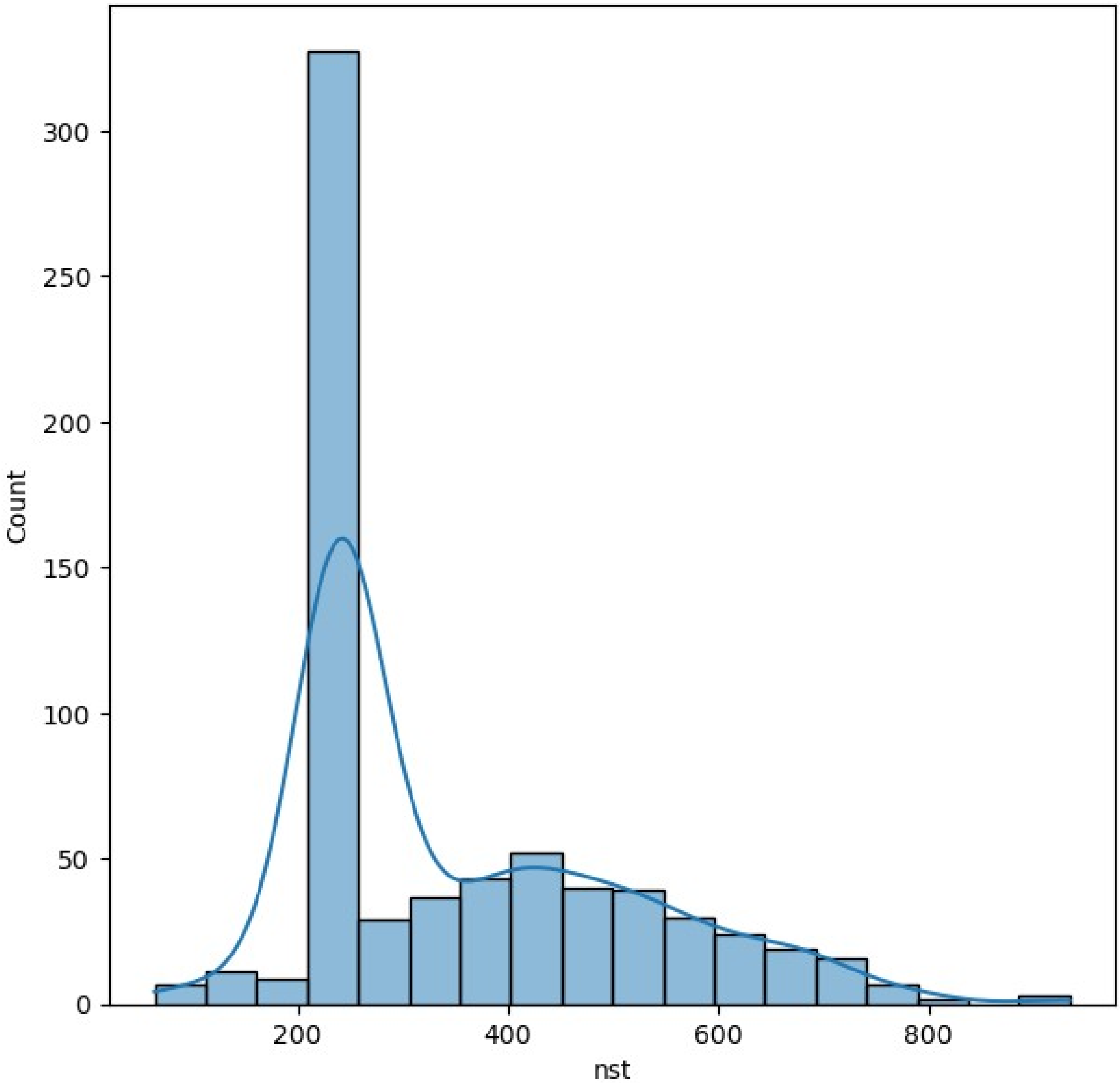

250
200
150
100
50
0
Count
1
2
3
4
5
6
7
8
9
cdi



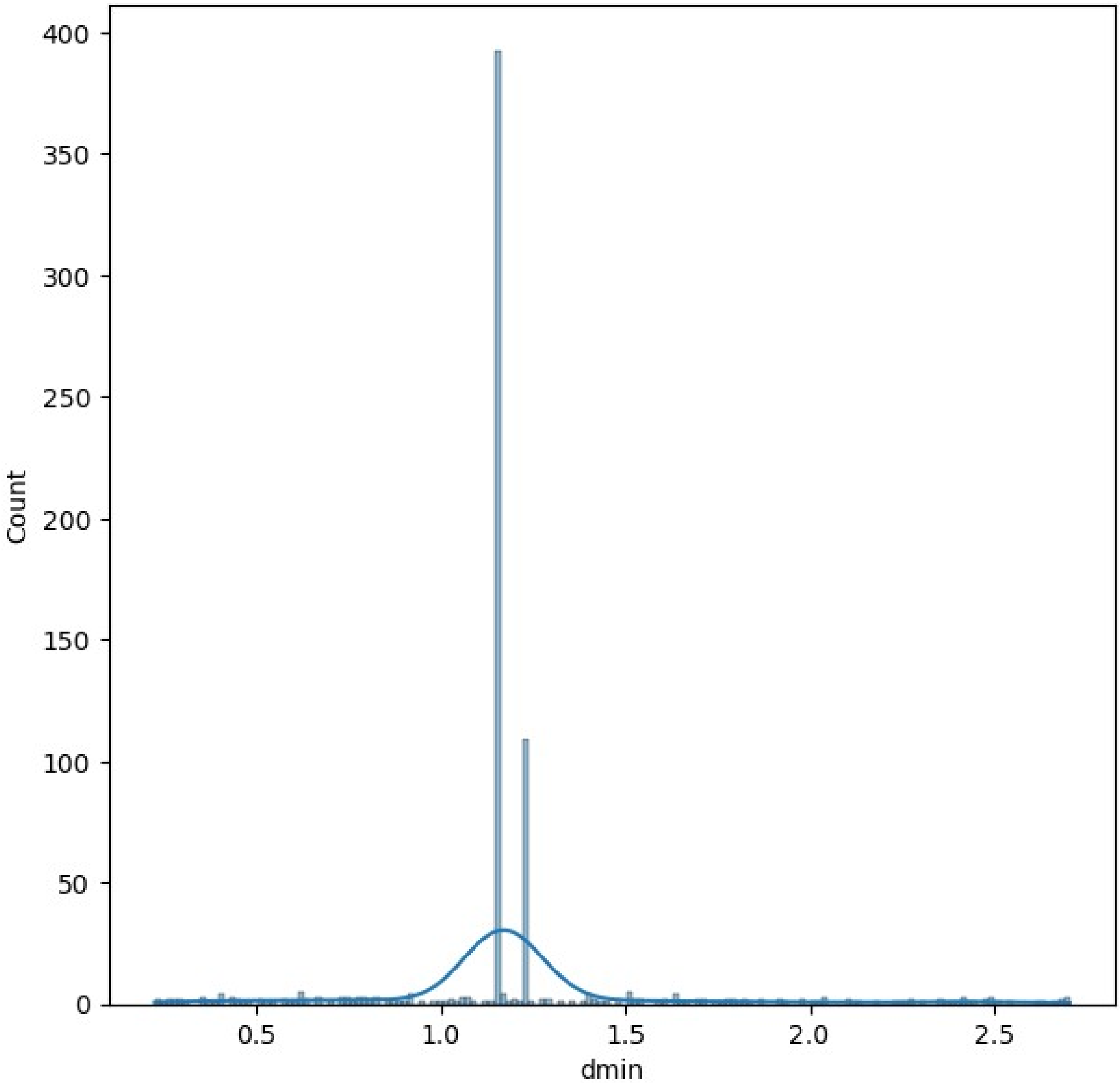
200
175
150
125
100
75
50
25
0
Count
3
4
5
6
7
8
9
mmi

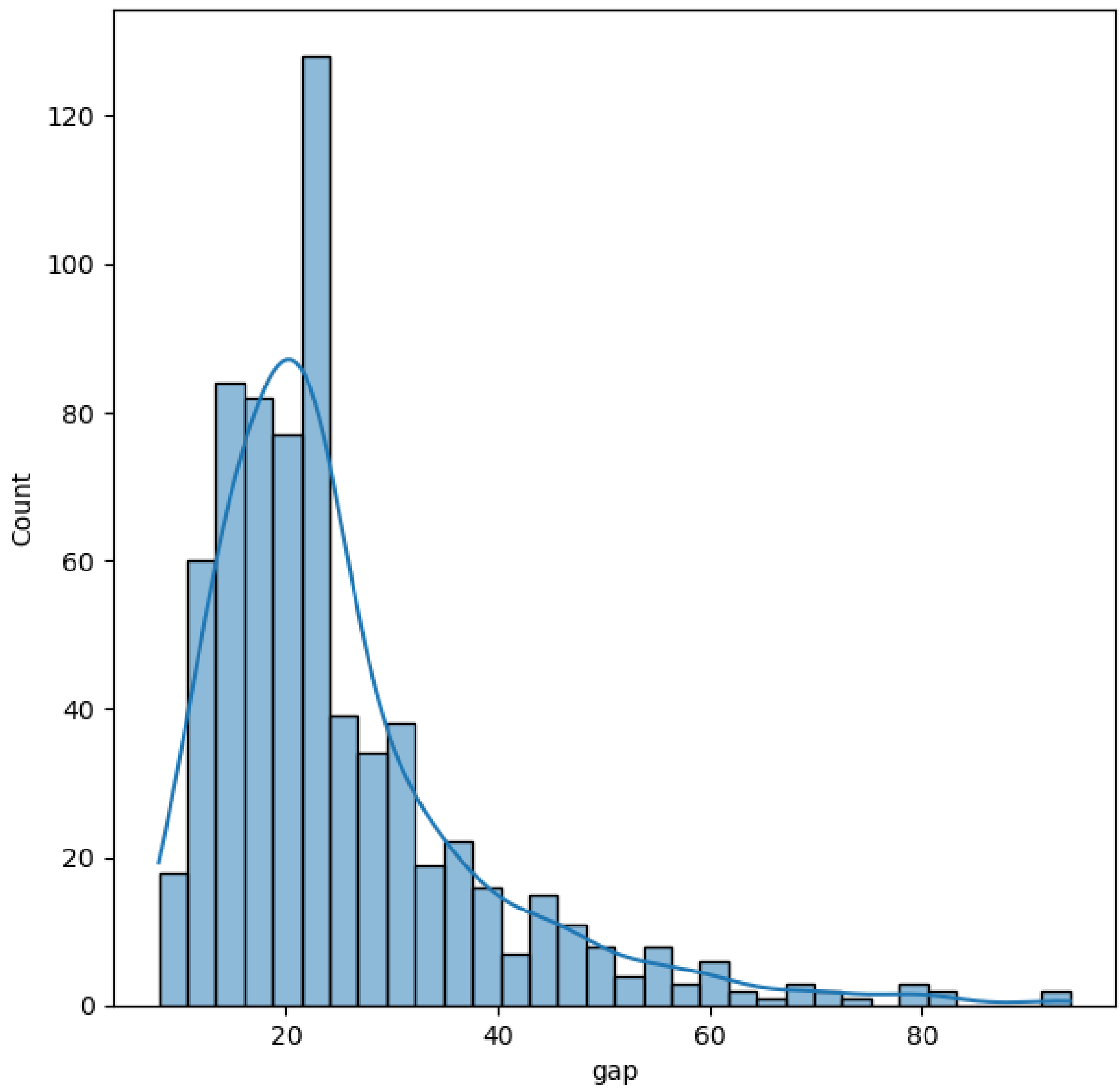


Count
200
175
150
125
100
75
50
25
0
600
800
1000
1200
1400
1600
1800
sig

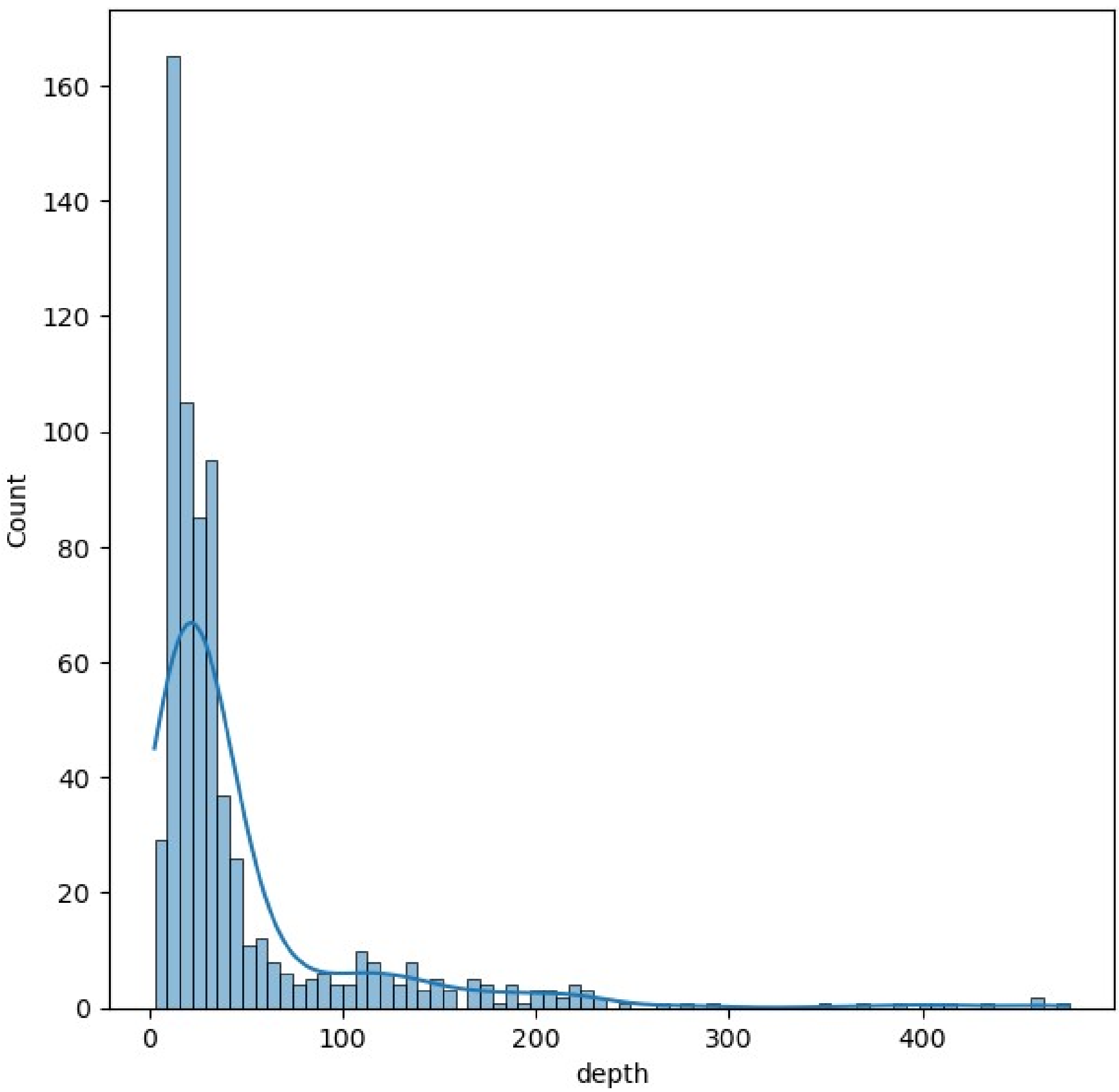


Count
nst
0
50
100
150
200
250
300
200
400
600
800

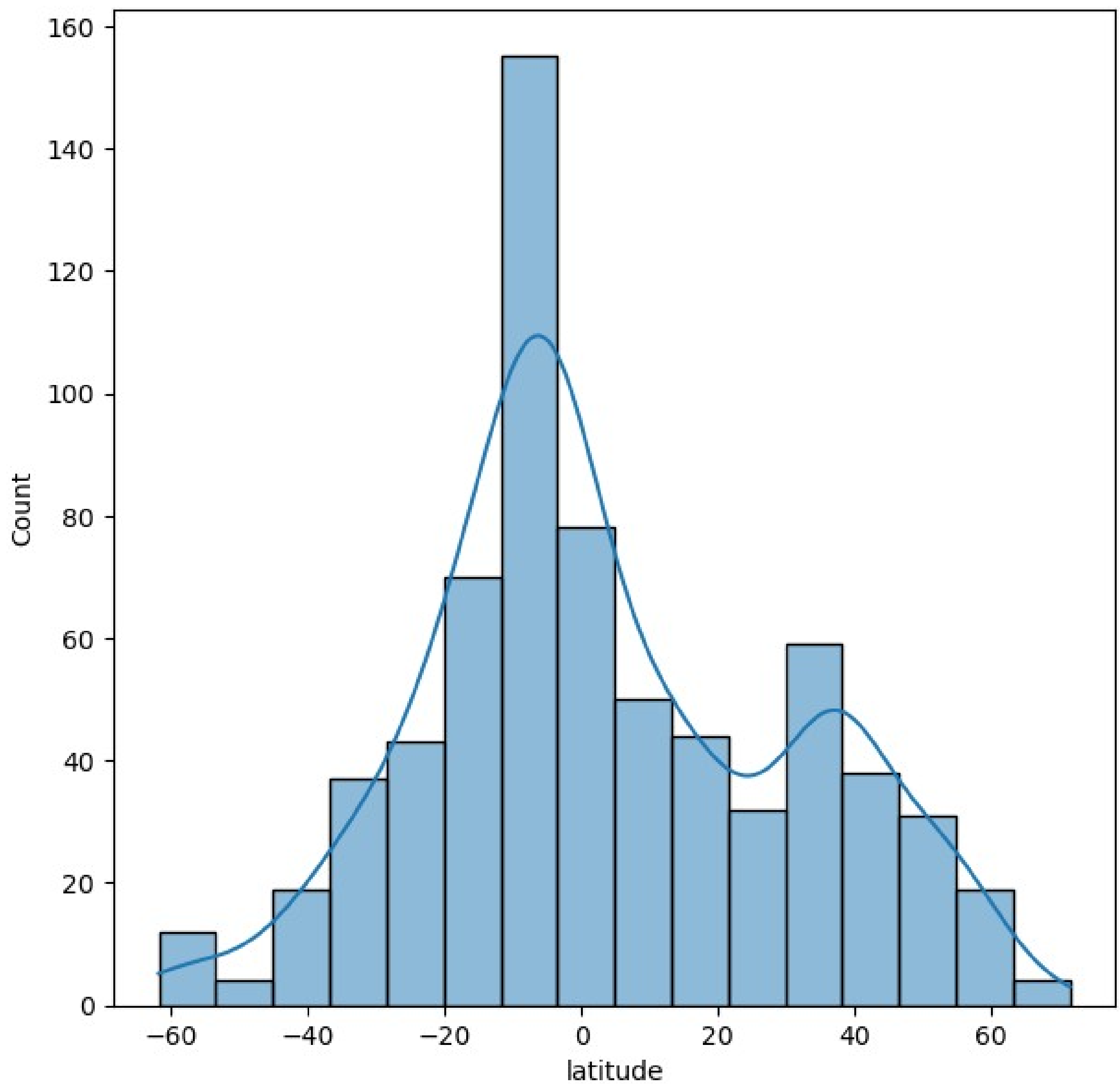

400
350
300
250
200
150
100
50
0
Count
0.5
1.0
1.5
2.0
2.5
dmin



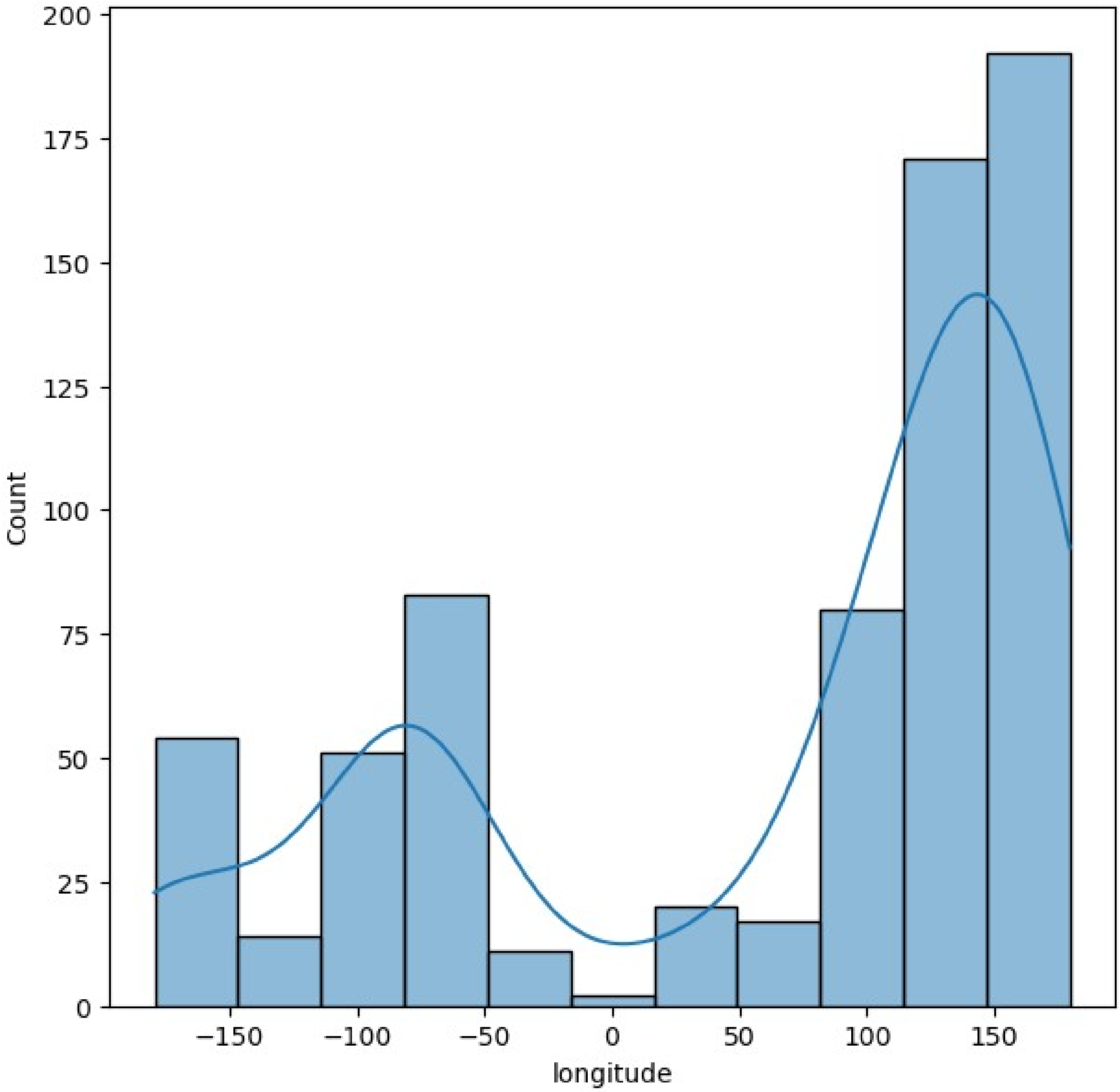
120
100
80
60
40
20
0
Count
20
40
60
80
gap

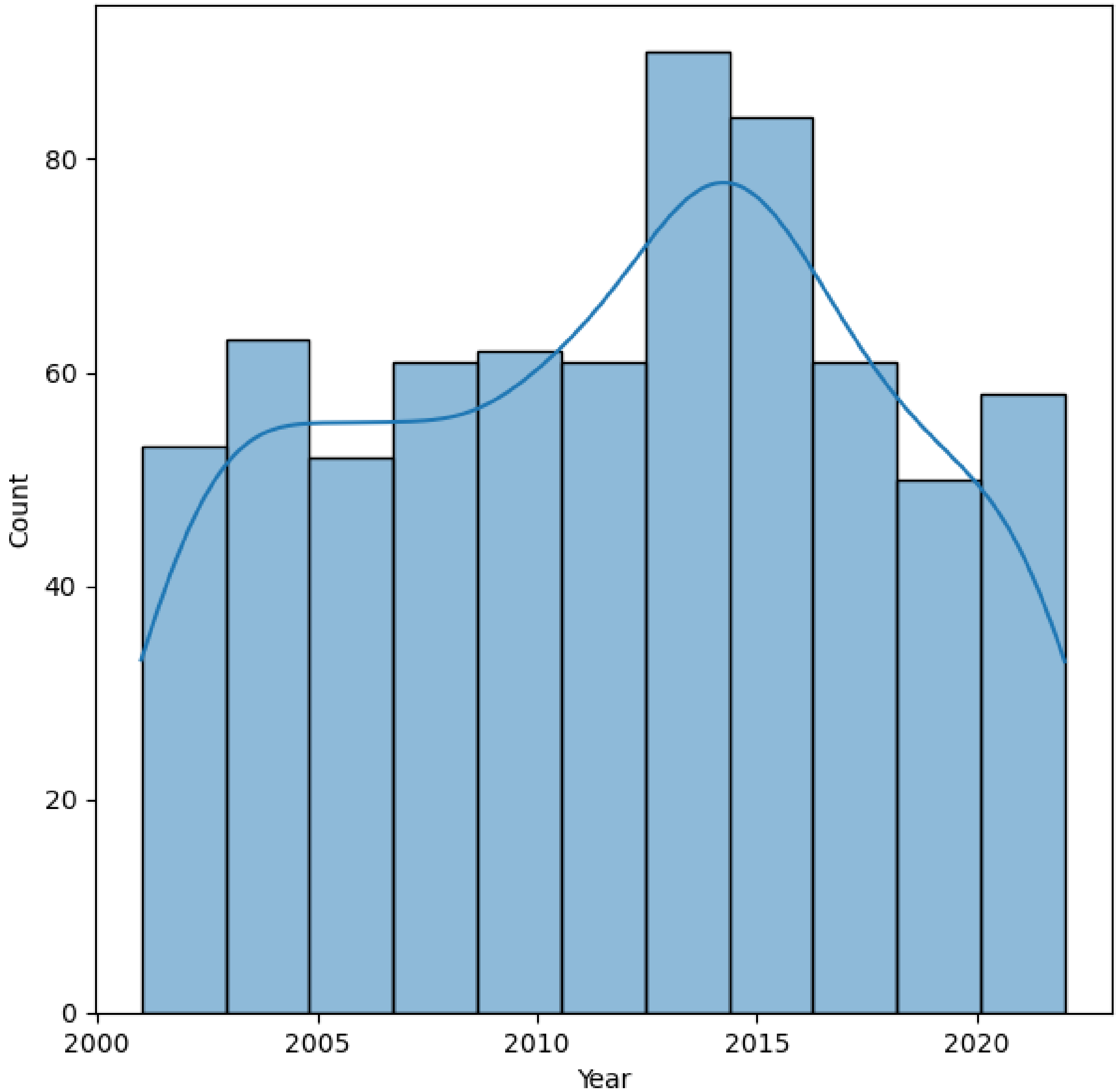


160
140
120
100
80
60
40
20
0
Count
0
100
200
300
400
depth

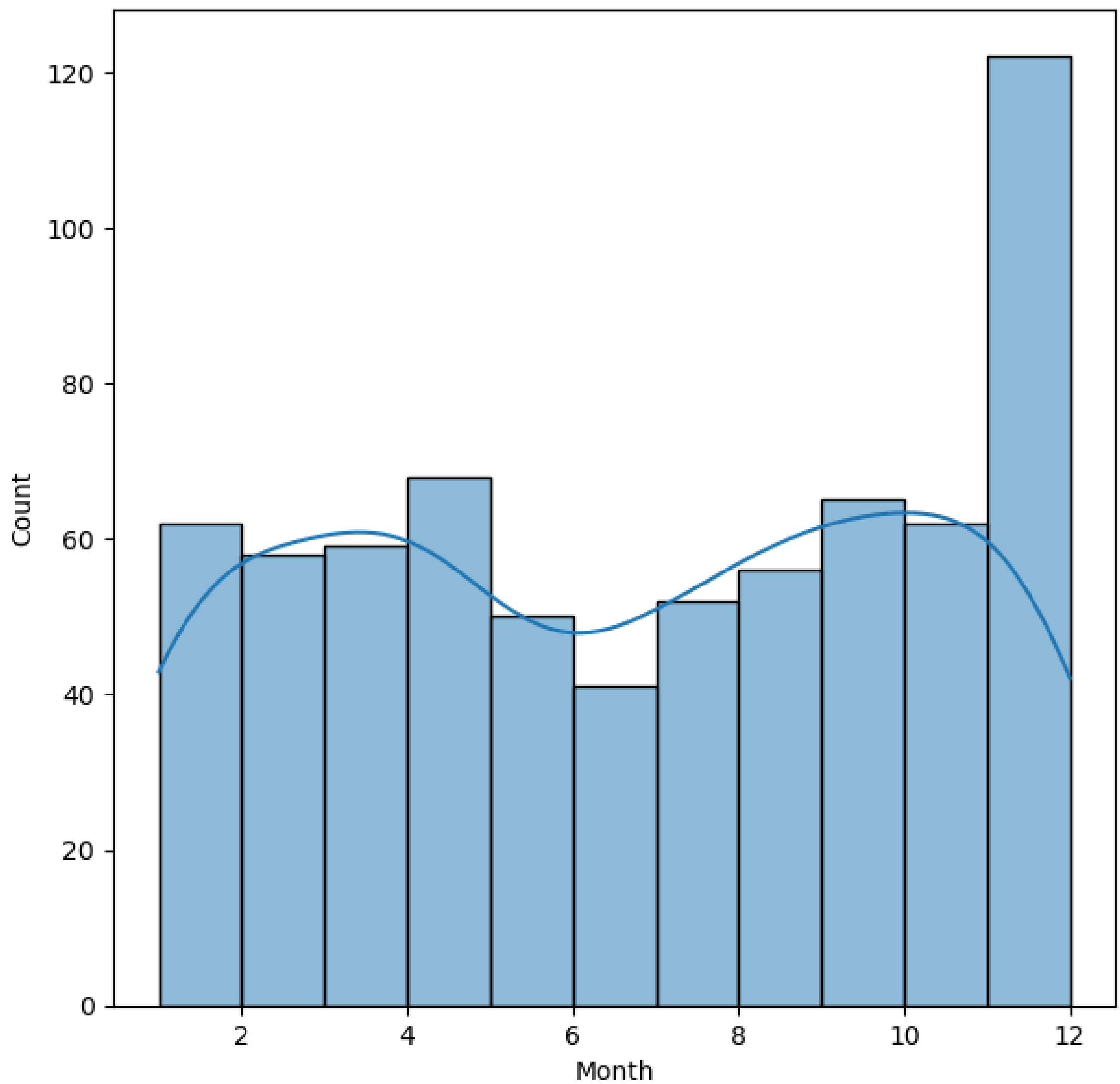


160
140
120
100
80
60
40
20
0
Count
−60
−40
−20
0
20
40
60
latitude

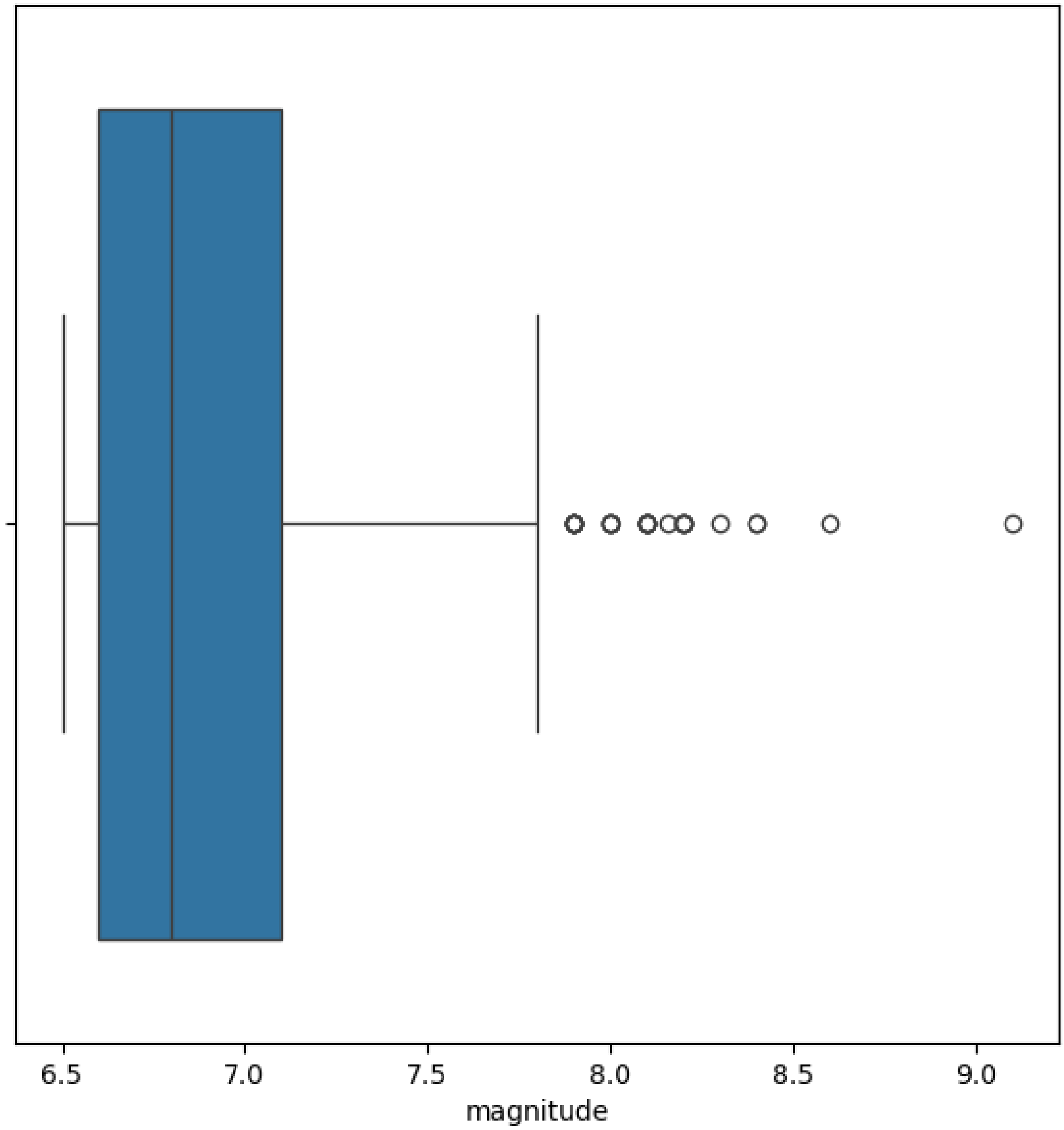

200
175
150
125
100
75
50
25
0
Count
−150
−100
−50
0
50
100
150
longitude



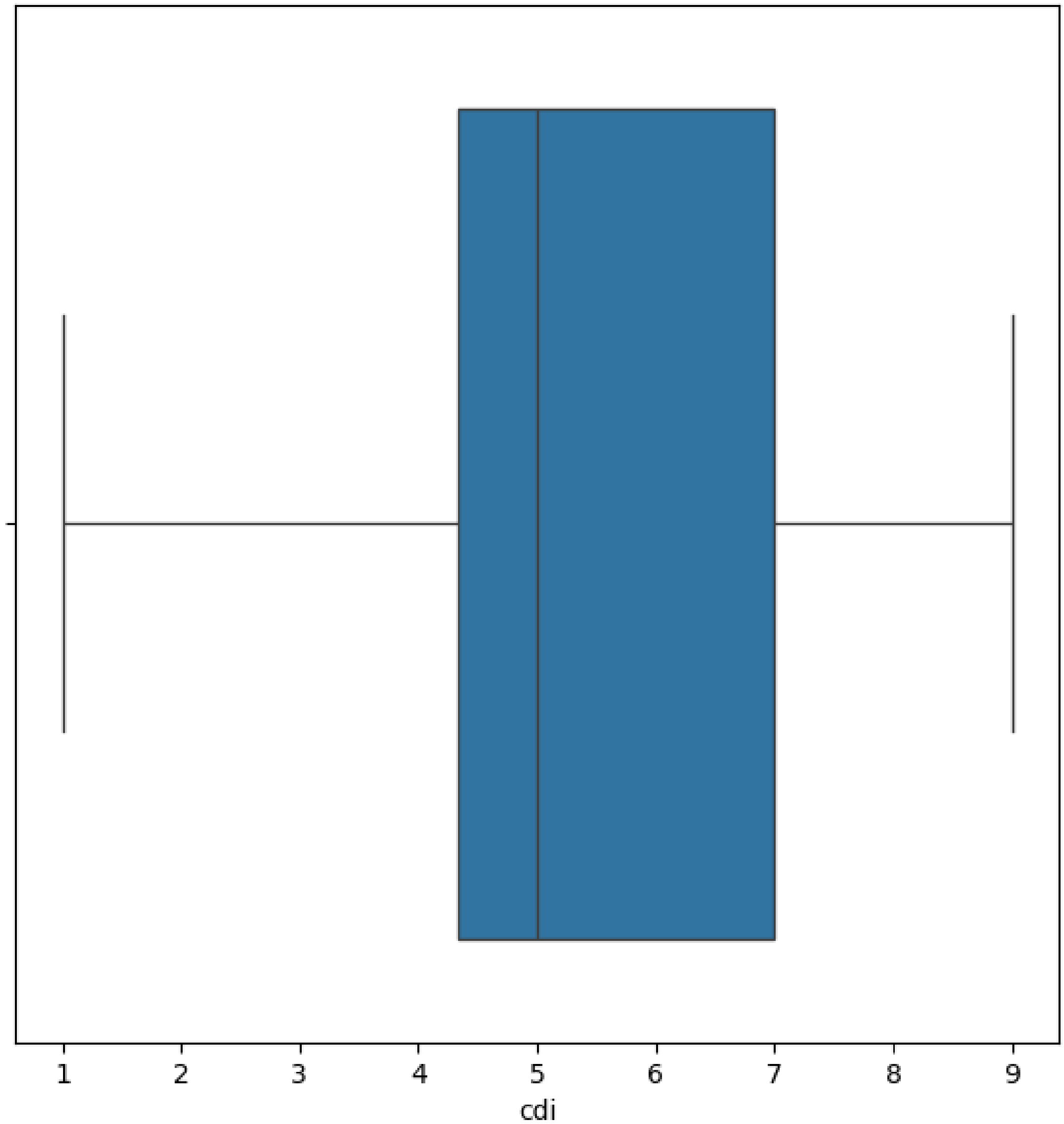
0
20
40
60
80
Count
2000
2005
2010
2015
2020
Year



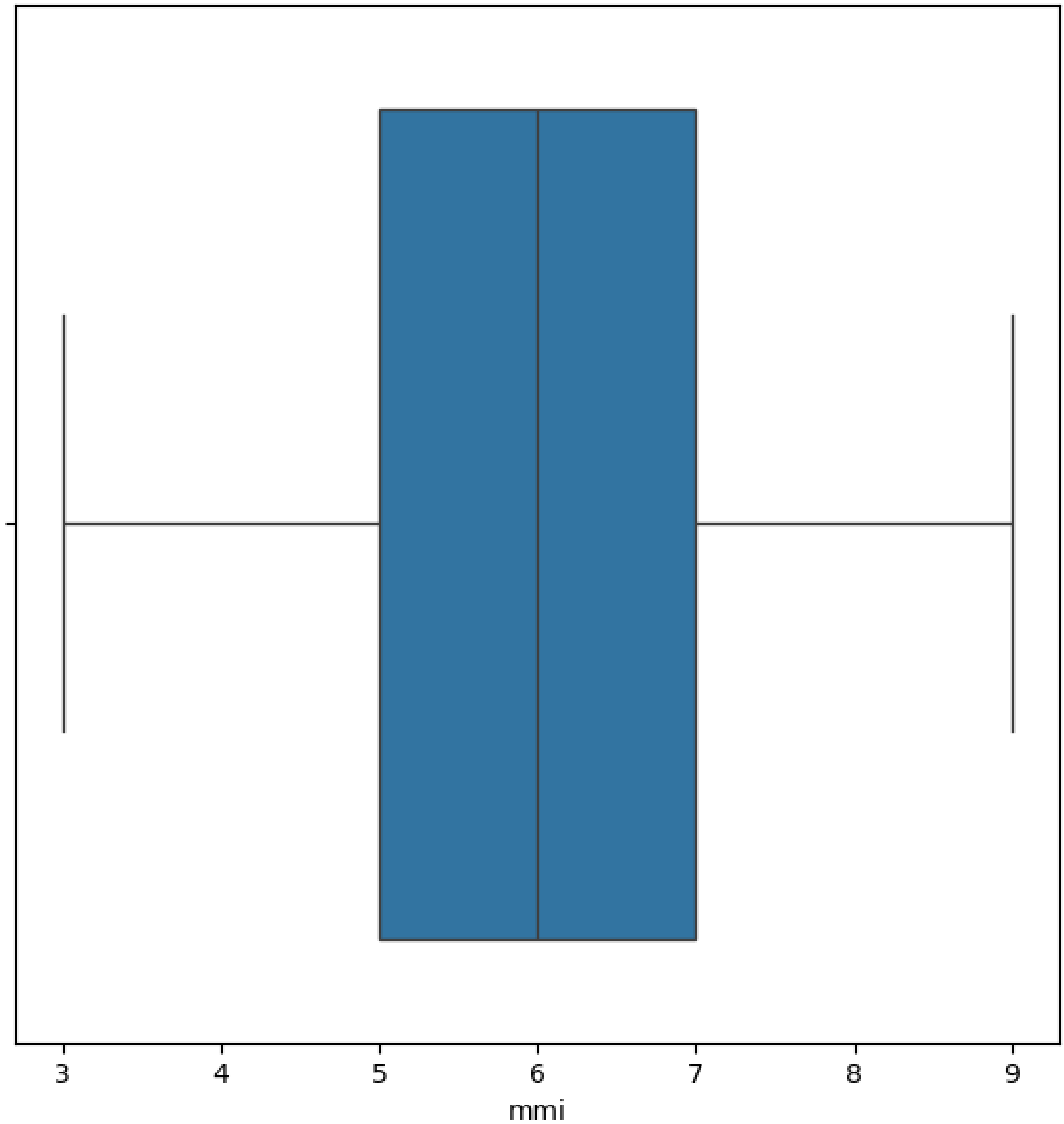

```
cols=['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap', 'depth',
       'latitude', 'longitude', 'Year', 'Month']
for col in cols:
    plt.figure(figsize=(7,7))
    sns.boxplot(x=df[col])
```

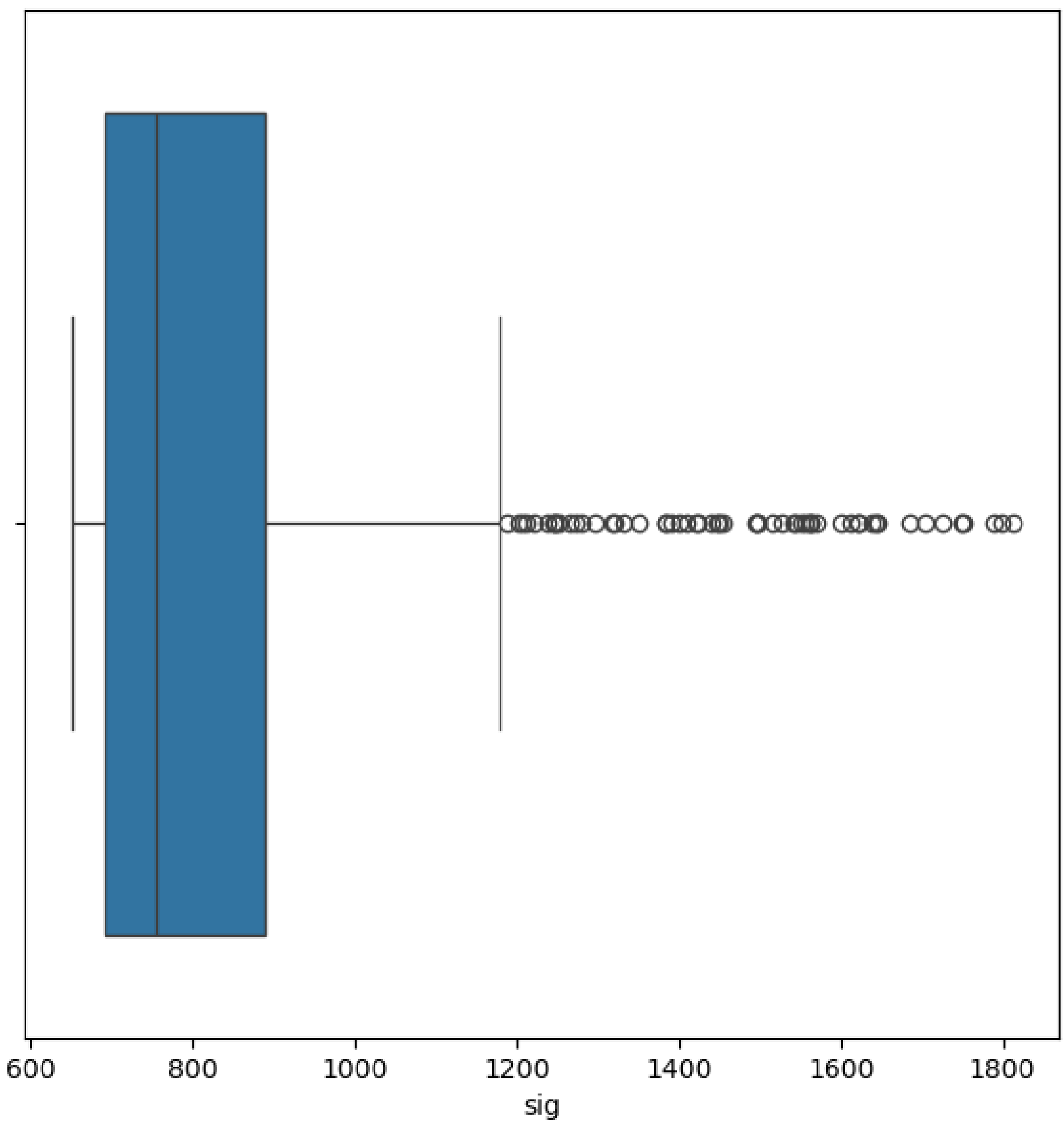


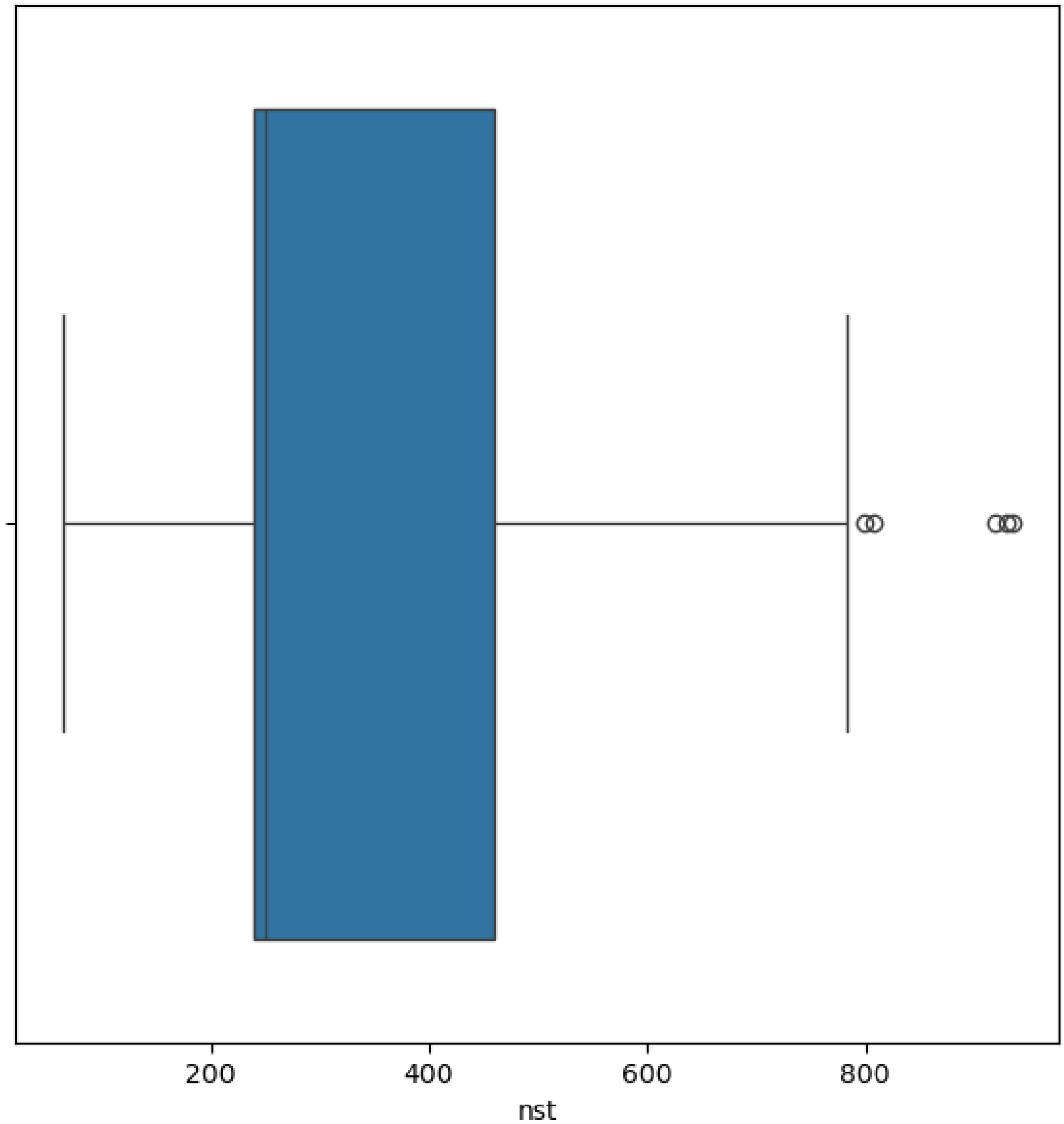
6.5
7.0
7.5
8.0
8.5
9.0
magnitude

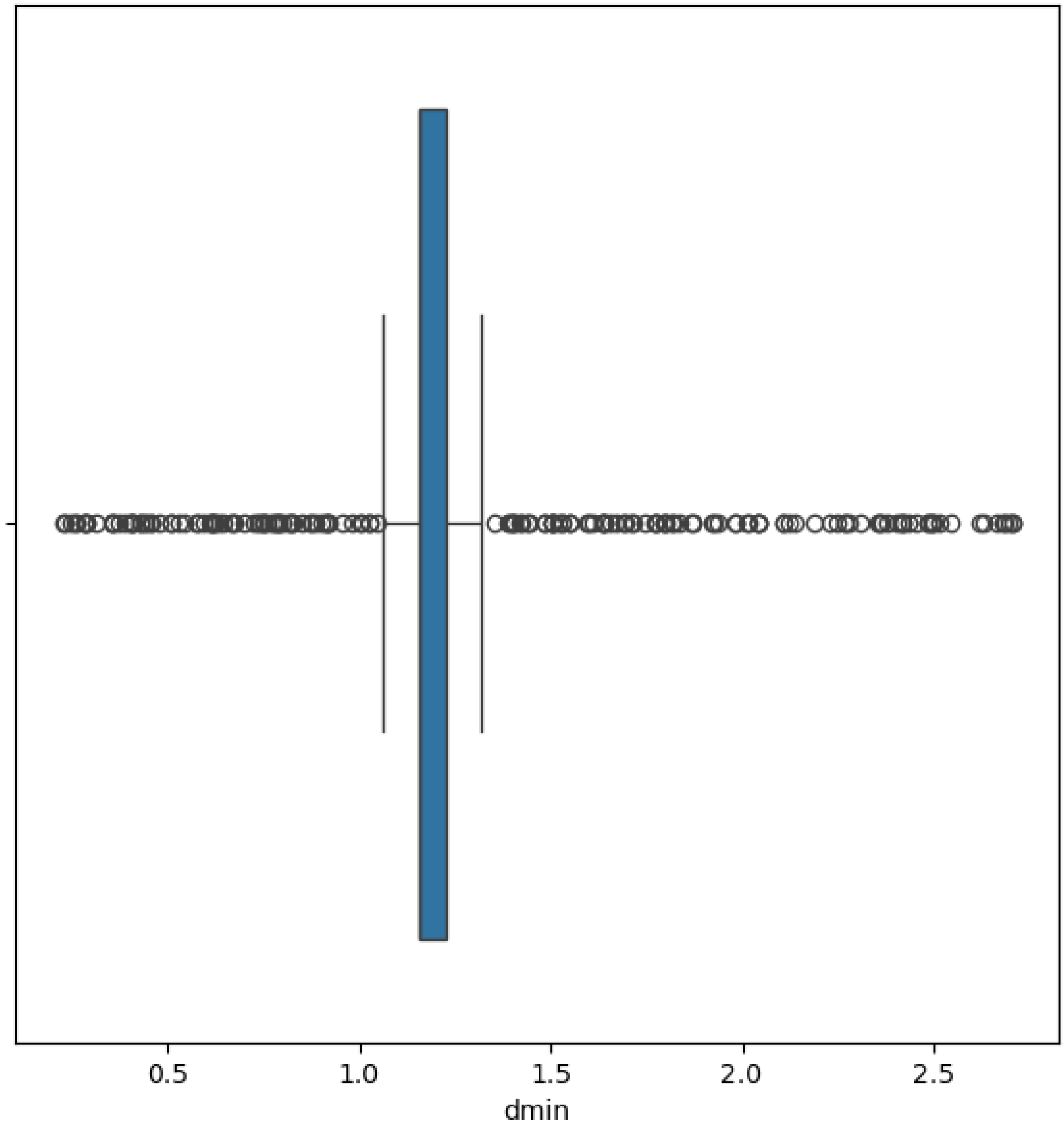


1
2
3
4
5
6
7
8
9
cdi

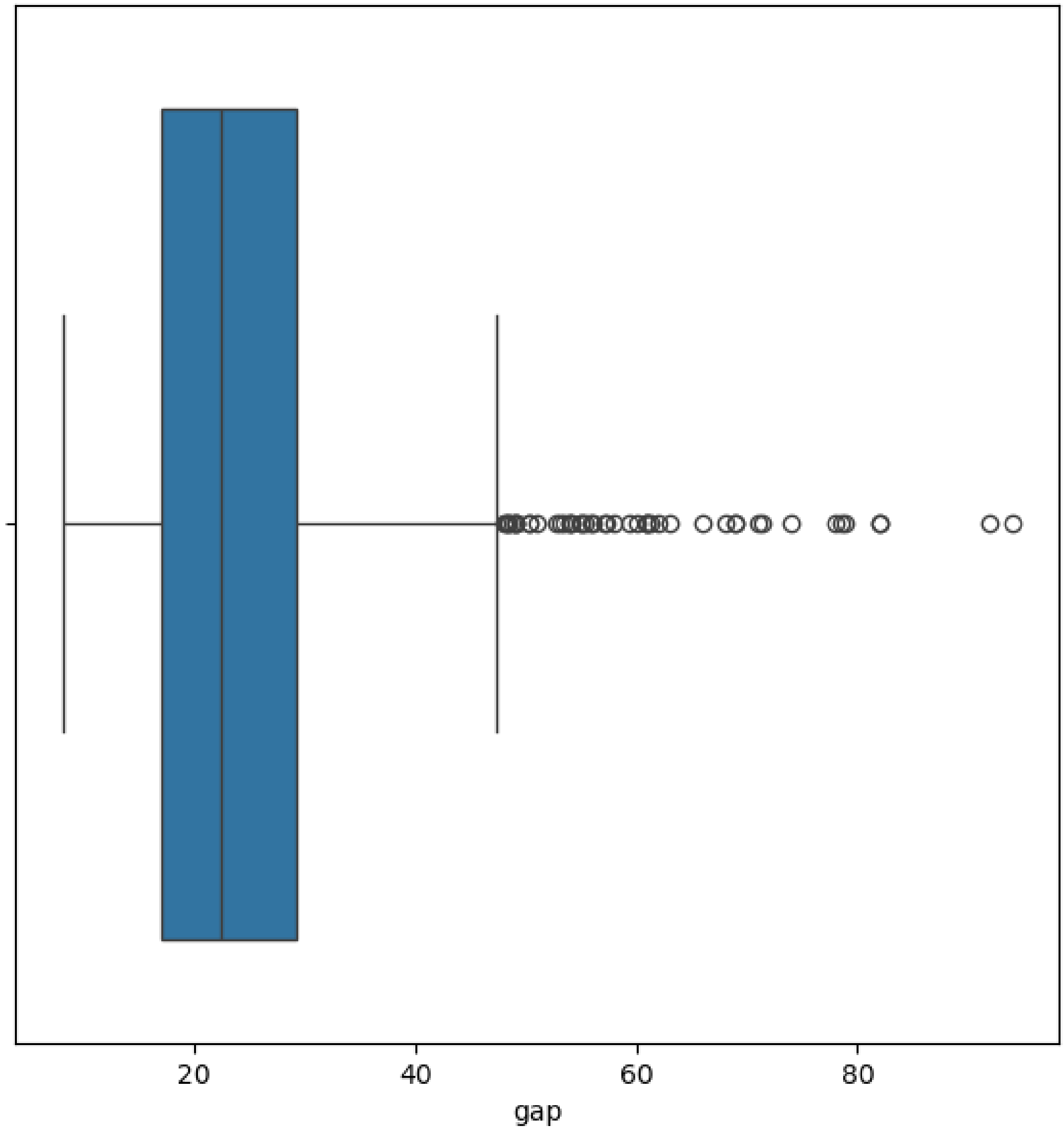

3
4
5
6
7
8
9
mmi


600
800
1000
1200
1400
1600
1800
sig

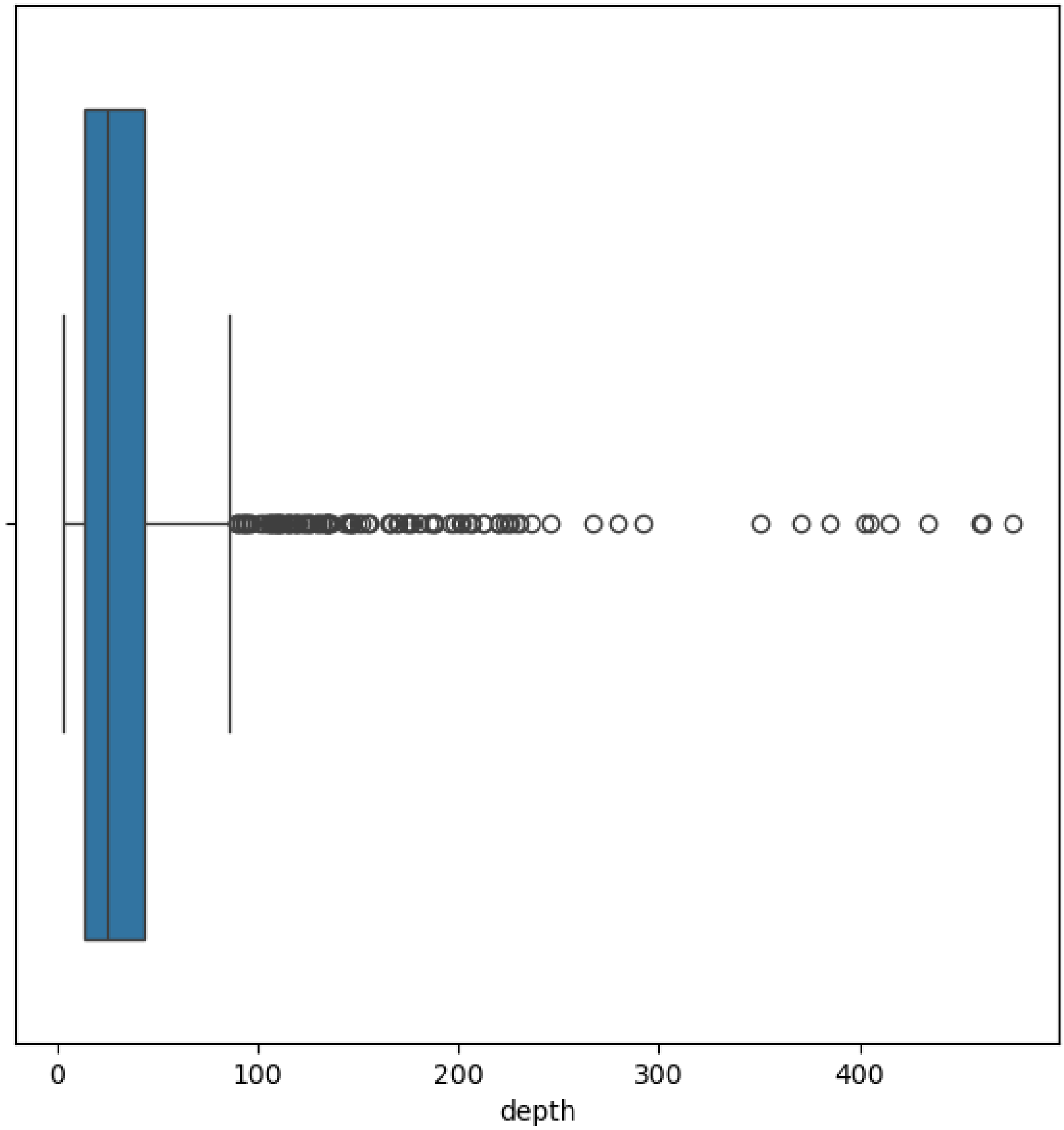

200
400
600
800
nst



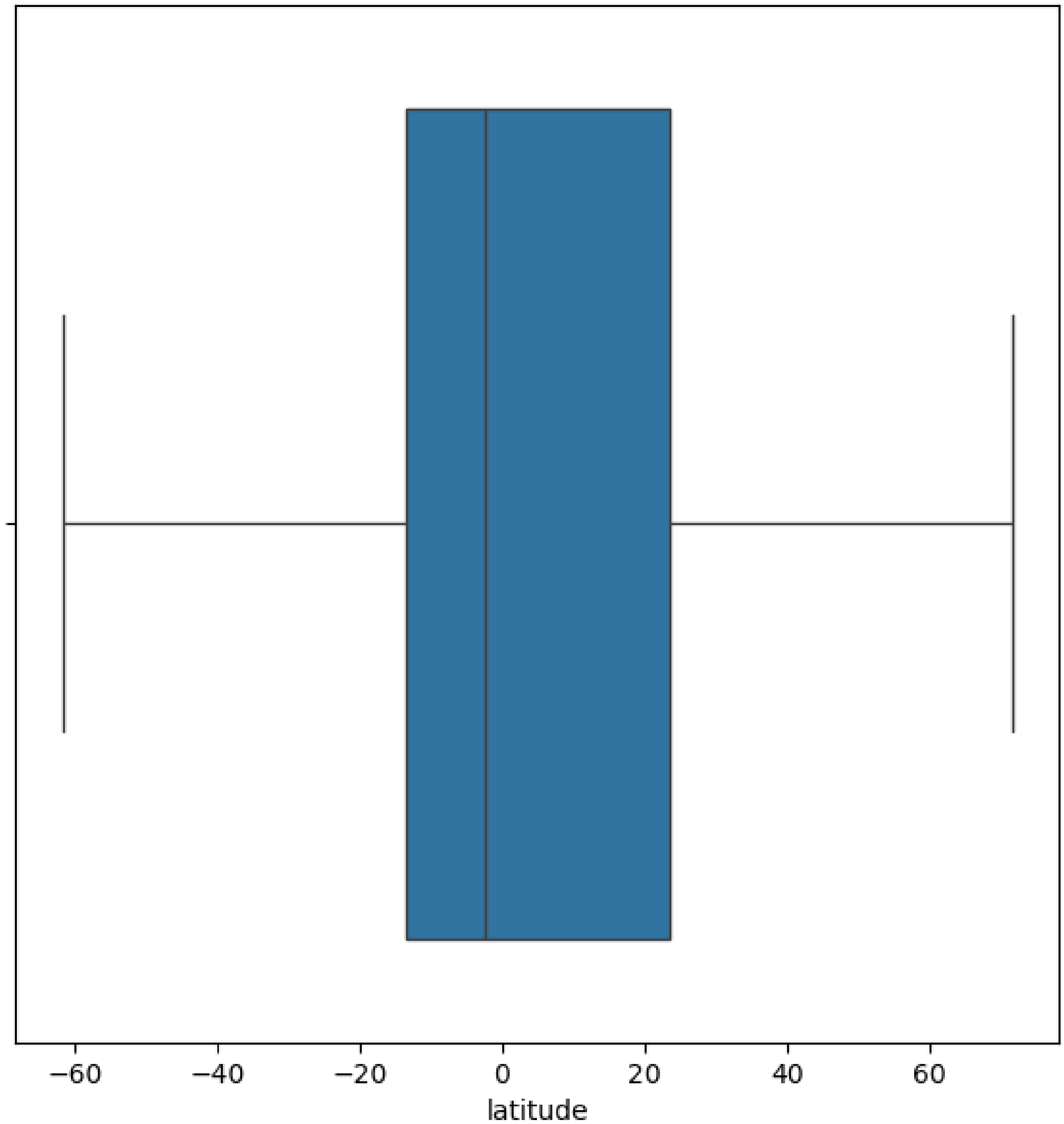
0.5
1.0
1.5
2.0
2.5
dmin

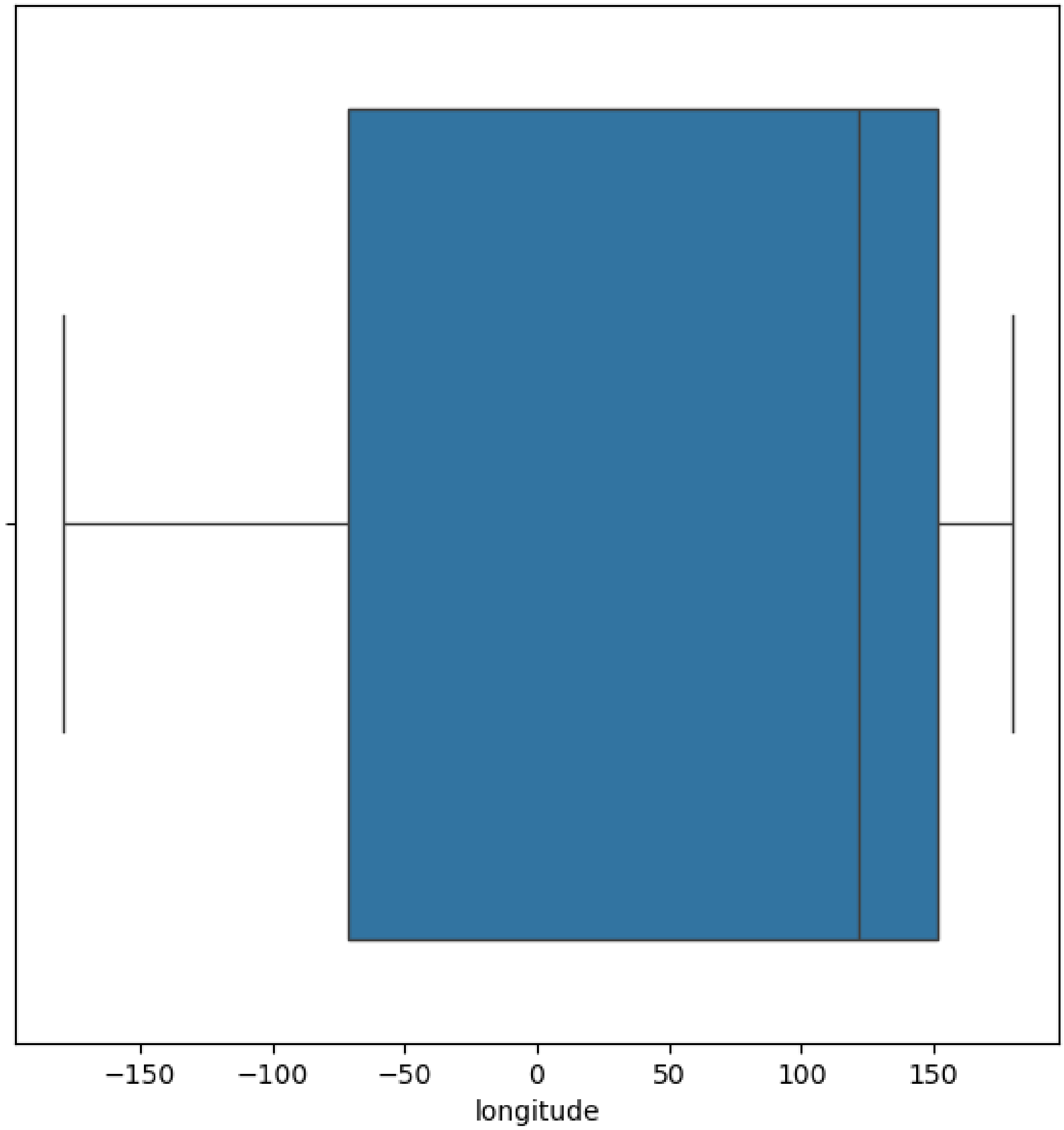


20
40
60
80
gap

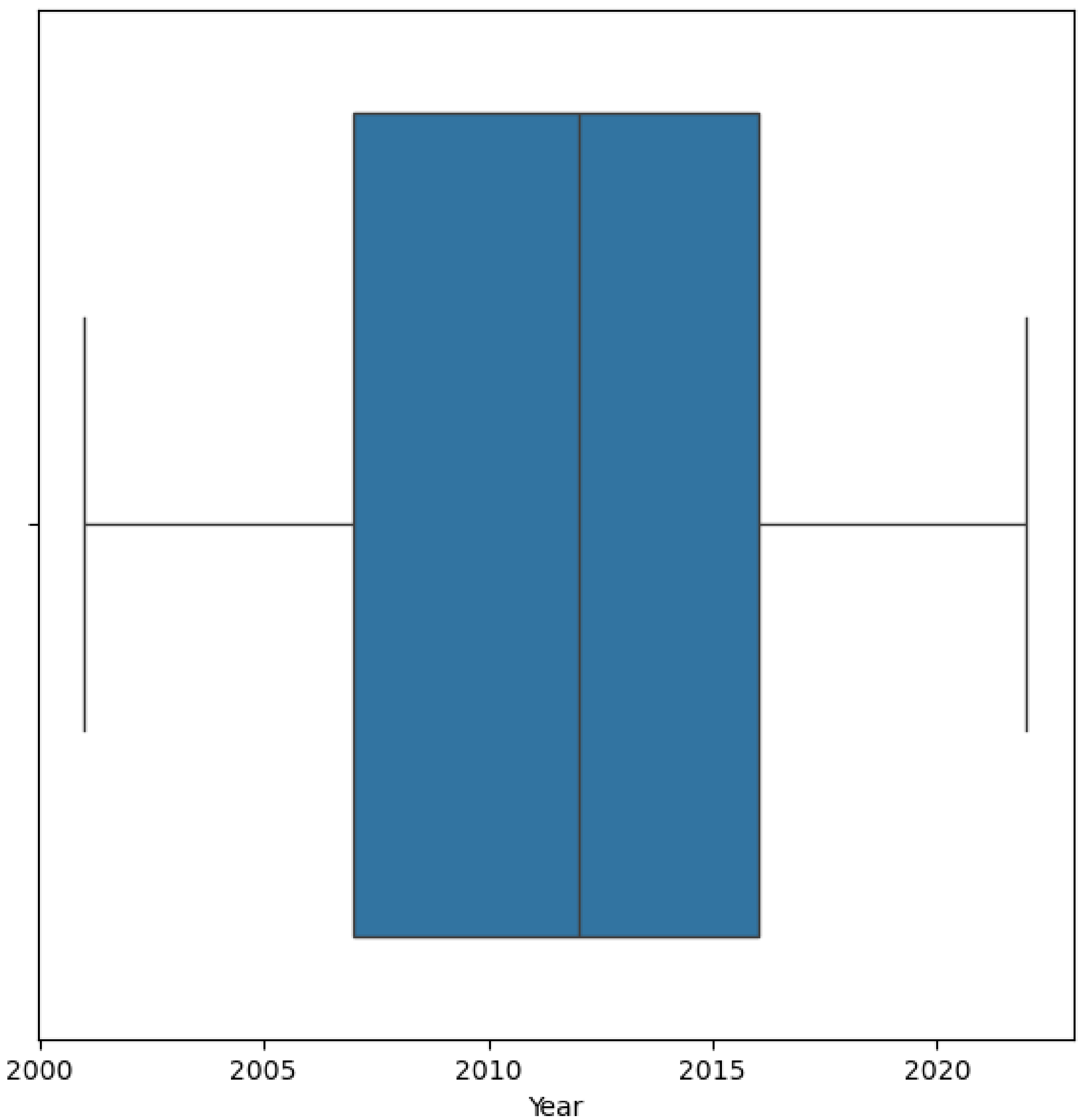


0
100
200
300
400
depth

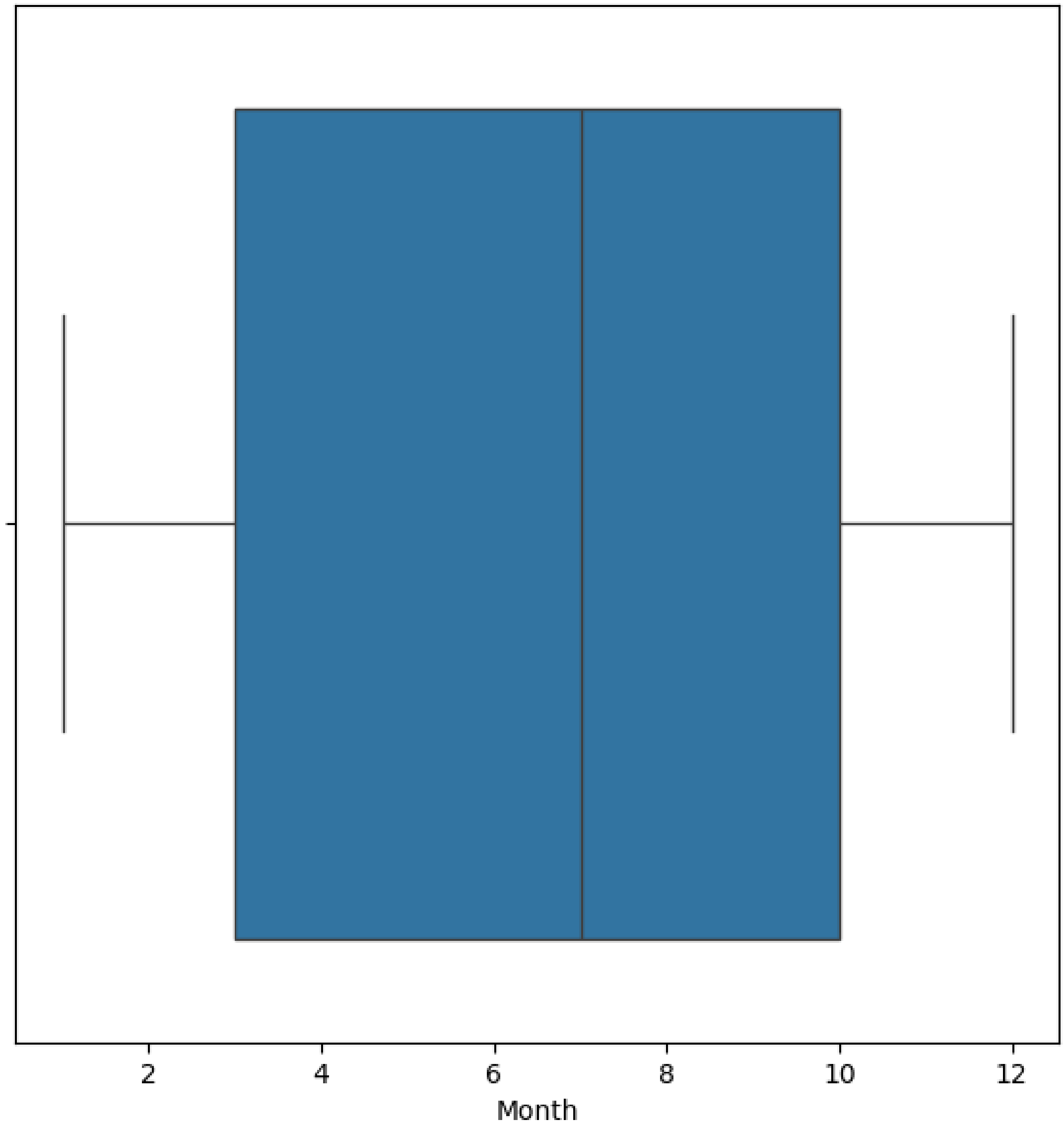

−60
−40
−20
0
20
40
60
latitude



−150
−100
−50
0
50
100
150
longitude



2000
2005
2010
2015
2020
Year




```python
df.head(10)
```

```
    magnitude  cdi  mmi   sig    nst      dmin   gap   depth
latitude  \
0         7.0  8.0    7   768  117.0  0.509000  17.0  14.000   -
9.7963
1         6.9  4.0    4   735   99.0  2.229000  34.0  25.000   -
4.9559
3         7.3  5.0    5   833  149.0  1.865000  21.0  37.000  -
19.2918
7         6.7  7.0    6   797  145.0  1.151000  37.0  20.000
7.6712
8         6.8  8.0    7  1179  175.0  2.137000  92.0  20.000
18.3300
9         7.6  9.0    8  1799  271.0  1.153000  69.0  26.943
```

```
18.3667
10         6.9  9.0    9   887  215.0  0.401000  34.0   10.000
23.1444
11         6.5  7.0    7   756  178.0  0.430000  54.0   10.000
23.0290
12         7.0  7.0    5   761  192.0  1.226396  45.0  137.000  -
21.2077
13         7.6  8.0    8   965  272.0  1.226396  12.0  116.000   -
6.2237

    longitude  Year  Month  tsunami
0    159.5960  2022     11        1
1    100.7380  2022     11        0
3   -172.1290  2022     11        1
7    -82.3396  2022     10        1
8   -102.9130  2022      9        1
9   -103.2520  2022      9        1
10   121.3070  2022      9        1
11   121.3480  2022      9        1
12   170.2390  2022      9        1
13   146.4710  2022      9        1
```

```
df_clean=df.copy()
```

```
from sklearn.preprocessing import StandardScaler
cols=[  'sig', 'nst', 'gap', 'depth',
       'latitude', 'longitude']
scaler=StandardScaler()
df_clean[cols]=scaler.fit_transform(df_clean[cols])
df_clean
```

```
     magnitude      cdi  mmi       sig       nst      dmin
gap  \
0          7.0  8.00000    7 -0.306481 -1.501622  0.509000 -0.622362

1          6.9  4.00000    4 -0.449875 -1.615822  2.229000  0.680963

3          7.3  5.00000    5 -0.024040 -1.298600  1.865000 -0.315697

7          6.7  7.00000    6 -0.180469 -1.323978  1.151000  0.910962

8          6.8  8.00000    7  1.479419 -1.133645  2.137000  5.127603

..         ...      ...  ...       ...       ...       ...       ...

777        7.7  4.33376    8  0.319236  0.465150  1.157279 -0.190191

778        6.9  5.00000    7 -0.406422 -0.725430  1.157279 -0.190191

779        7.1  4.33376    7 -0.271719  0.116207  1.157279 -0.190191
```

```
780         6.8  4.33376    5 -0.554161 -1.837877  1.157279 -0.190191

781         7.5  4.33376    7  0.115009 -0.188326  1.157279 -0.190191


        depth  latitude  longitude  Year  Month  tsunami
0   -0.519222 -0.505578   0.871407  2022     11        1
1   -0.354250 -0.325076   0.359030  2022     11        0
3   -0.174281 -0.859673  -2.016359  2022     11        1
7   -0.429237  0.145799  -1.234715  2022     10        1
8   -0.429237  0.543274  -1.413813  2022      9        1
..        ...       ...        ...   ...    ...      ...
777  0.170659  0.346341  -1.289736  2001      1        0
778 -0.183280  1.976897  -1.852281  2001      1        0
779  0.815548 -0.696944   0.937341  2001      1        0
780 -0.234271  0.107009   0.586769  2001      1        0
781 -0.234271  0.116966   0.583984  2001      1        0

[695 rows x 13 columns]
```

```
df_clean.columns
```

```
Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap',
'depth',
       'latitude', 'longitude', 'Year', 'Month', 'tsunami'],
      dtype='object')
```

```
from scipy.stats import pearsonr

# ----------------------------------
# Pearson Correlation Calculation
# ----------------------------------

# List of features to check against target
selected_features = ['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin',
'gap', 'depth',
       'latitude', 'longitude', 'Year', 'Month']

correlations = {
    feature: pearsonr(df_clean[feature], df_clean['tsunami'])[0]
    for feature in selected_features
}
correlation_df = pd.DataFrame(list(correlations.items()),
columns=['Feature', 'Pearson Correlation'])
correlation_df.sort_values(by='Pearson Correlation', ascending=False)
```

```
      Feature  Pearson Correlation
10       Year             0.674124
1         cdi             0.121926
7       depth             0.059754
5        dmin             0.049814
```

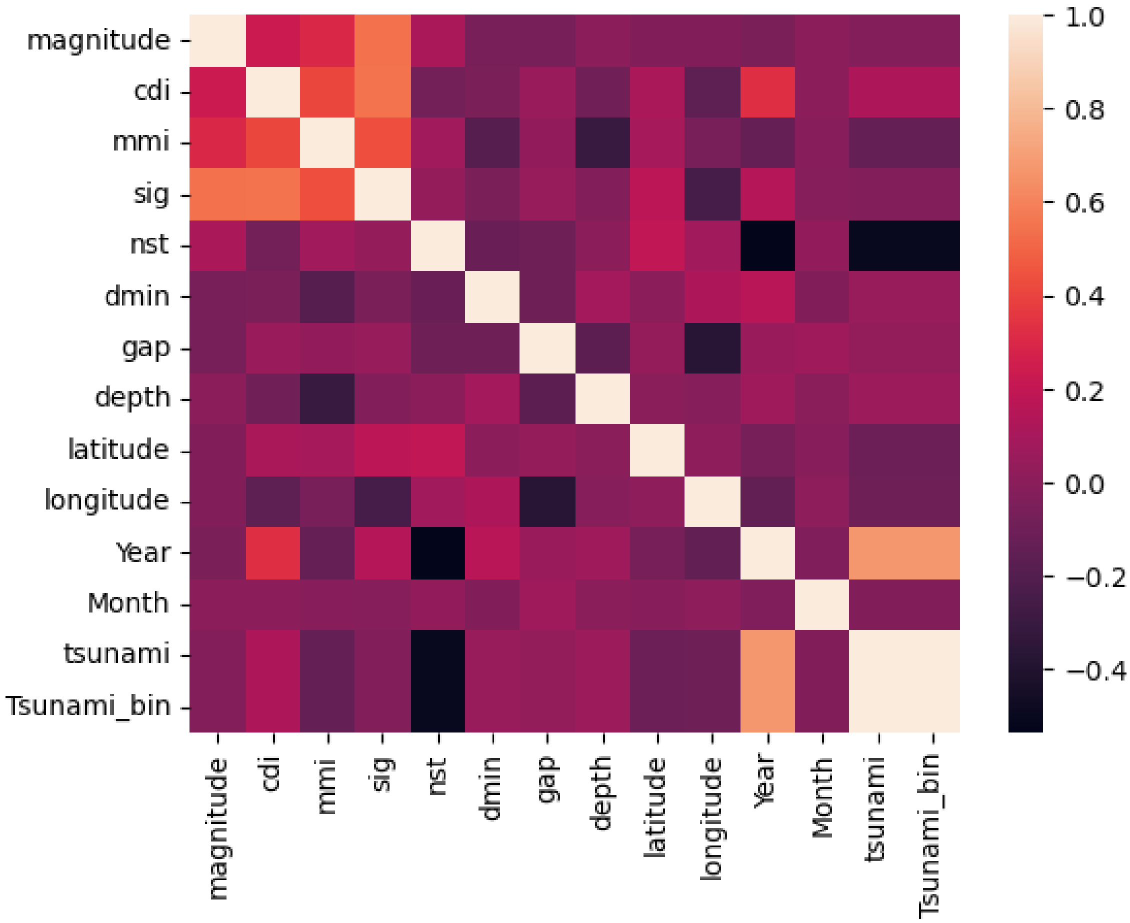
```
6          gap            0.031907
0    magnitude           -0.020328
3          sig           -0.029271
11       Month           -0.037051
9    longitude           -0.108383
8     latitude           -0.109183
2          mmi           -0.138401
4          nst           -0.501840
```

```
from scipy.stats import chi2_contingency

alpha = 0.05

df_clean['Tsunami_bin'] = df_clean['tsunami']
chi2_results = {}

for col in cols:
    contingency = pd.crosstab(df_clean[col], df_clean['Tsunami_bin'])
    chi2_stat, p_val, _, _ = chi2_contingency(contingency)
    decision = 'Reject Null (Keep Feature)' if p_val < alpha else
'Accept Null (Drop Feature)'
    chi2_results[col] = {
        'chi2_statistic': chi2_stat,
        'p_value': p_val,
        'Decision': decision
    }

chi2_df = pd.DataFrame(chi2_results).T
chi2_df = chi2_df.sort_values(by='p_value')
chi2_df
```

	chi2_statistic	p_value	Decision
gap	383.613923	0.0	Reject Null (Keep Feature)
nst	410.799659	0.000003	Reject Null (Keep Feature)
depth	276.307847	0.104676	Accept Null (Drop Feature)
sig	315.1489	0.289625	Accept Null (Drop Feature)
latitude	695.0	0.460792	Accept Null (Drop Feature)
longitude	692.864611	0.472876	Accept Null (Drop Feature)

```
sns.heatmap(df_clean.corr())
```

```
<Axes: >
```




```
df_clean.drop(columns=['Tsunami_bin'],inplace=True)

df_clean.head(5)
```

```
   magnitude  cdi  mmi       sig       nst   dmin       gap     depth  \
0        7.0  8.0    7 -0.306481 -1.501622  0.509 -0.622362 -0.519222

1        6.9  4.0    4 -0.449875 -1.615822  2.229  0.680963 -0.354250

3        7.3  5.0    5 -0.024040 -1.298600  1.865 -0.315697 -0.174281

7        6.7  7.0    6 -0.180469 -1.323978  1.151  0.910962 -0.429237

8        6.8  8.0    7  1.479419 -1.133645  2.137  5.127603 -0.429237


   latitude  longitude  Year  Month  tsunami
0 -0.505578   0.871407  2022     11        1
1 -0.325076   0.359030  2022     11        0
3 -0.859673  -2.016359  2022     11        1
```

```
7  0.145799  -1.234715  2022     10        1
8  0.543274  -1.413813  2022      9        1
```

Logistic Regression

```
x=df_clean.drop(columns=['tsunami','longitude','Month','Year','gap','d
min'],axis=1)
y=df_clean['tsunami']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
x.columns

Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'depth', 'latitude'],
dtype='object')

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train,y_train)
y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score, confusion_matrix ,
classification_report
acc=accuracy_score(y_test,y_pred)
acc*100

87.05035971223022

confusion_matrix(y_test,y_pred)

array([[69,  8],
       [10, 52]])

print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.87      0.90      0.88        77
           1       0.87      0.84      0.85        62

    accuracy                           0.87       139
   macro avg       0.87      0.87      0.87       139
weighted avg       0.87      0.87      0.87       139


X_test
```

```
     magnitude      cdi  mmi       sig       nst     depth  latitude
459        7.0  6.00000    6  0.323581  0.744305  1.475434 -0.425205
754        6.8  4.33376    7 -0.554161  0.198685 -0.489227 -0.498743
322        6.7  7.00000    4 -0.610649 -0.725430 -0.399242 -1.544183
413        7.1  4.33376    5 -0.271719  1.245515 -0.579211 -0.557289
390        6.5  5.00000    6 -0.693209  2.184490 -0.279263  0.298385
..         ...      ...  ...       ...       ...       ...       ...
596        6.5  5.00000    6 -0.710590  0.179651 -0.504224 -0.506984
34         6.6  8.00000    5 -0.714935 -0.725430 -0.444235  1.822170
208        6.5  4.33376    4 -0.819222 -0.725430 -0.504224 -2.243990
606        8.1  9.00000    8  0.810250  0.541284 -0.369248 -0.455970
419        7.1  5.00000    5  0.019413  1.036149  1.595414 -0.383887

[139 rows x 7 columns]
```

Custom Testing

```
# data_inp = []
# features = ['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'depth',
'latitude']

# for col in features:
#     val = float(input(f'Enter the {col}: '))
#     data_inp.append(val)

# data = {features[i]: [data_inp[i]] for i in range(len(features))}
# X_custom = pd.DataFrame(data)
# print("\nYour input data:")
# print(X_custom)

# from sklearn.linear_model import LogisticRegression
# model = LogisticRegression()

# model.fit(X_train,y_train)
# y_pred_custom= model.predict(X_custom)
# if(y_pred_custom):
#     print("Tsunami")
# else:
#     print("No Tsunami")
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)

df_clean.columns

Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap',
'depth',
```

```
       'latitude', 'longitude', 'Year', 'Month', 'tsunami'],
      dtype='object')
```

```
x=df_clean.drop(columns=['tsunami','Month','mmi','cdi','dmin','gap','depth','magnitude'],axis=1)
y=df_clean['tsunami']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
x.columns
```

```
Index(['sig', 'nst', 'latitude', 'longitude', 'Year'], dtype='object')
```

```
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier()
```

```
y_knn_predict=knn.predict(X_test)
```

```
acc=accuracy_score(y_test,y_knn_predict)
acc*100
```

```
93.5251798561151
```

```
confusion_matrix(y_test,y_knn_predict)
```

```
array([[70,  7],
       [ 2, 60]])
```

```
print(classification_report(y_test,y_knn_predict))
```

```
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        77
           1       0.90      0.97      0.93        62

    accuracy                           0.94       139
   macro avg       0.93      0.94      0.93       139
weighted avg       0.94      0.94      0.94       139
```

Custom Testing

```
# data_inp = []
# features = ['sig', 'nst', 'latitude', 'longitude', 'Year']

# for col in features:
#     val = float(input(f'Enter the {col}: '))
#     data_inp.append(val)

# data = {features[i]: [data_inp[i]] for i in range(len(features))}
# X_custom = pd.DataFrame(data)
```

```
# print("\nYour input data:")
# print(X_custom)

# from sklearn.neighbors import KNeighborsClassifier
# knn=KNeighborsClassifier(n_neighbors=5)

# knn.fit(X_train,y_train)
# y_pred_custom= knn.predict(X_custom)
# if(y_pred_custom):
#     print("Tsunami")
# else:
#     print("No Tsunami")
```

Naive Bayes

```
df_clean.columns
```

```
Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap',
'depth',
       'latitude', 'longitude', 'Year', 'Month', 'tsunami'],
      dtype='object')
```

```
x=df_clean.drop(columns=['tsunami','Month','dmin','gap','depth','latit
ude','longitude','magnitude','cdi','mmi'],axis=1)
y=df_clean['tsunami']
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
x.columns
```

```
Index(['sig', 'nst', 'Year'], dtype='object')
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred_gnb = gnb.fit(X_train, y_train).predict(X_test)
```

```
acc=accuracy_score(y_test,y_pred_gnb)
acc*100
```

```
88.48920863309353
```

```
confusion_matrix(y_test,y_pred_gnb)
```

```
array([[65, 12],
       [ 4, 58]])
```

```
print(classification_report(y_test,y_pred_gnb))
```

```
              precision    recall  f1-score   support

           0       0.94      0.84      0.89        77
```

```
           1       0.83      0.94      0.88        62

    accuracy                           0.88       139
   macro avg       0.89      0.89      0.88       139
weighted avg       0.89      0.88      0.89       139
```

Custom Testing

```
# data_inp = []
# features = ['sig', 'nst', 'Year']

# for col in features:
#     val = float(input(f'Enter the {col}: '))
#     data_inp.append(val)

# data = {features[i]: [data_inp[i]] for i in range(len(features))}
# X_custom = pd.DataFrame(data)

# print("\nYour input data:")
# print(X_custom)

# from sklearn.naive_bayes import GaussianNB
# gnb = GaussianNB()
# y_pred_custom = gnb.fit(X_train, y_train).predict(X_custom)
# if(y_pred_custom):
#     print("Tsunami")
# else:
#     print("No Tsunami")
```

Decision Tree

```
df_clean.columns
```

```
Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap',
'depth',
       'latitude', 'longitude', 'Year', 'Month', 'tsunami'],
      dtype='object')
```

```
df_clean.head(5)
```

```
   magnitude  cdi  mmi       sig       nst   dmin       gap     depth
\
0        7.0  8.0    7 -0.306481 -1.501622  0.509 -0.622362 -0.519222

1        6.9  4.0    4 -0.449875 -1.615822  2.229  0.680963 -0.354250

3        7.3  5.0    5 -0.024040 -1.298600  1.865 -0.315697 -0.174281
```

```
7        6.7  7.0    6 -0.180469 -1.323978  1.151  0.910962 -0.429237

8        6.8  8.0    7  1.479419 -1.133645  2.137  5.127603 -0.429237
```

```
   latitude  longitude  Year  Month  tsunami
0 -0.505578   0.871407  2022     11        1
1 -0.325076   0.359030  2022     11        0
3 -0.859673  -2.016359  2022     11        1
7  0.145799  -1.234715  2022     10        1
8  0.543274  -1.413813  2022      9        1
```

```python
x=df_clean.drop(columns=['tsunami','magnitude','sig','dmin'],axis=1)
y=df_clean['tsunami']
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
x.columns
```

```
Index(['cdi', 'mmi', 'nst', 'gap', 'depth', 'latitude', 'longitude',
'Year',
       'Month'],
      dtype='object')
```

```python
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_tree_pred=clf.predict(X_test)
```

```python
acc=accuracy_score(y_test,y_tree_pred)
acc*100
```

```
93.5251798561151
```

```python
confusion_matrix(y_test,y_tree_pred)
```

```
array([[73,  4],
       [ 5, 57]])
```

```python
print(classification_report(y_test,y_tree_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.94        77
           1       0.93      0.92      0.93        62

    accuracy                           0.94       139
   macro avg       0.94      0.93      0.93       139
weighted avg       0.94      0.94      0.94       139
```

Custom Testing

```
# data_inp = []
# features = ['cdi', 'mmi', 'nst', 'gap', 'depth', 'latitude',
'longitude', 'Year','Month']
# for col in features:
#     val = float(input(f'Enter the {col}: '))
#     data_inp.append(val)

# data = {features[i]: [data_inp[i]] for i in range(len(features))}
# X_custom = pd.DataFrame(data)

# print("\nYour input data:")
# print(X_custom)

# from sklearn.tree import DecisionTreeClassifier
# clf = DecisionTreeClassifier(random_state=42)
# clf.fit(X_train, y_train)
# y_pred_custom=clf.predict(X_custom)
# if(y_pred_custom):
#     print("Tsunami")
# else:
#     print("No Tsunami")
```

Support Vector Machine

```
df_clean.columns
```

```
Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'dmin', 'gap',
'depth',
       'latitude', 'longitude', 'Year', 'Month', 'tsunami'],
      dtype='object')
```

```
x=df_clean.drop(columns=['tsunami','Year','dmin','latitude','longitude
','Month','gap'],axis=1)
y=df_clean['tsunami']
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
x.columns
```

```
Index(['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'depth'],
dtype='object')
```

```
from sklearn import svm
SVM = svm.SVC(kernel='poly')
SVM.fit(X_train, y_train)
y_svm_pred=SVM.predict(X_test)
```

```
acc=accuracy_score(y_test,y_svm_pred)
acc*100
```

```
84.89208633093526
```

```python
confusion_matrix(y_test,y_svm_pred)
```

```
array([[66, 11],
       [10, 52]])
```

```python
print(classification_report(y_test,y_svm_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86        77
           1       0.83      0.84      0.83        62

    accuracy                           0.85       139
   macro avg       0.85      0.85      0.85       139
weighted avg       0.85      0.85      0.85       139
```

Custom Testing

```python
# data_inp = []
# features = ['magnitude', 'cdi', 'mmi', 'sig', 'nst', 'depth']
# for col in features:
#     val = float(input(f'Enter the {col}: '))
#     data_inp.append(val)

# data = {features[i]: [data_inp[i]] for i in range(len(features))}
# X_custom = pd.DataFrame(data)

# print("\nYour input data:")
# print(X_custom)

# from sklearn import svm
# SVM = svm.SVC(kernel='poly')
# SVM.fit(X_train, y_train)
#y_pred_custom=SVM.predict(X_custom)
# if(y_pred_custom):
#     print("Tsunami")
# else:
#     print("No Tsunami")
```