

This code connects Google Drive to Google Colab, allowing to access and use files stored in your Drive within the notebook.

Start coding or generate with AI.

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

Listing the files in the google drive

```
!ls /content/drive/MyDrive
```

```
17126653786084101591701804519246.jpg
'5-CSE3041_PDS-TUPLE_SET-Dr_RAJESH_M-Ref-PPTs (1).gdoc'
'5-CSE3041_PDS-TUPLE_SET-Dr_RAJESH_M-Ref-PPTs (2).gdoc'
'5-CSE3041_PDS-TUPLE_SET-Dr_RAJESH_M-Ref-PPTs (3).gdoc'
'5-CSE3041_PDS-TUPLE_SET-Dr_RAJESH_M-Ref-PPTs (4).gdoc'
5-CSE3041_PDS-TUPLE_SET-Dr_RAJESH_M-Ref-PPTs.gdoc
6-CSE3041_PDS-DICT_FUNC-Dr_RAJESH_M-Ref-PPTs.gdoc
7-CSE3041_Class-Package_File-Dr_RAJESH_M-Ref-PPTs.gdoc
880cf805-1213-47b7-8f93-264ae9af0e08_payment_confirmation_20250236-081430.png
980a7924-37a7-43c3-8609-455cac38164f_payment_confirmation_20250241-110511.png
Classroom
'Colab Notebooks'
IMG-20231006-WA0040.jpg
IMG_20231103_232206_006.jpg
IMG_20240409_200114_280.jpg
IMG_20240727-WA0037.jpg
IMG_20240808_220857.jpg
IMG_20240920_165749.jpg
IMG_20241111-WA0029.jpg
IMG_9675.JPG
IPS-6.gdoc
'Mgt advertisement '
'nelson mandela .docx'
'pat 1 23mia1104.gdoc'
PHOTO-2025-03-08-15-25-42.jpg
'Rohith T 23MIA1104 - Digital Assignment 1.gdoc'
Screenshot_2023_0927_092645.jpg
'Screenshot_2023_1105_132953 (1).jpg'
Screenshot_2023_1105_132953.jpg
'supplychain (1).gsheet'
'supplychain.gsheet'
Untitled6.ipynb
'watson (1).gsheet'
watson.gsheet
WIID_28NOV2023.csv
```

Importing all needed libraries

Loading the dataset and also printing its first five rows

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
file_path = "/content/drive/MyDrive/WIID_28NOV2023.csv"
df = pd.read_csv(file_path)
```

```
df.head()
```

0	1	Afghanistan	AFG	AF	2008	29.00	NaN	NaN	NaN	NaN	...	\t26,427,200	New 2019	High	12	National statistical authority	European Union and the Central Statistics Orga...
1	2	Afghanistan	AFG	AF	2012	33.00	NaN	NaN	NaN	NaN	...	\t30,466,478	New 2019	High	12	National statistical authority	European Union and the Central Statistics Orga...
2	3	Afghanistan	AFG	AF	2017	31.00	NaN	NaN	NaN	NaN	...	\t35,643,416	New 2019	High	12	National statistical authority	European Union and the Central Statistics Orga...
3	4	Albania	ALB	AL	1996	27.01	NaN	NaN	NaN	NaN	...	\t3,271,331	New 2023	Average	13	World Bank	Poverty and Inequality Platform (PIP)
4	5	Albania	ALB	AL	2002	31.74	NaN	NaN	NaN	NaN	...	\t3,123,551	New 2023	Average	13	World Bank	Poverty and Inequality Platform (PIP)

5 rows x 68 columns

Double-click (or enter) to edit

Printing the the dataset info and head and general info about the dataset.

```
df.head()
df.info()
df.describe()
```

4/24/25, 5:02 PM EDA.ipynb - Colab

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24367 entries, 0 to 24366
Data columns (total 68 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          24367 non-null   int64  
 1   country     24367 non-null   object  
 2   c3          24367 non-null   object  
 3   c2          24357 non-null   object  
 4   year         24367 non-null   int64  
 5   gini         24279 non-null   float64 
 6   ge0          11723 non-null   float64 
 7   ge1          13395 non-null   float64 
 8   ge2          11784 non-null   float64 
 9   a025         10989 non-null   float64 
 10  a050         13325 non-null   float64 
 11  a075         10989 non-null   float64 
 12  a1          13209 non-null   float64 
 13  a2          11714 non-null   float64 
 14  palma        19007 non-null   float64 
 15  ratio_top20bottom20 19865 non-null   float64 
 16  bottom40    18860 non-null   float64 
 17  q1          19030 non-null   float64 
 18  q2          18864 non-null   float64 
 19  q3          18885 non-null   float64 
 20  q4          18891 non-null   float64 
 21  q5          19031 non-null   float64 
 22  d1          18082 non-null   float64 
 23  d2          18074 non-null   float64 
 24  d3          18074 non-null   float64 
 25  d4          18074 non-null   float64 
 26  d5          18094 non-null   float64 
 27  d6          18099 non-null   float64 
 28  d7          18105 non-null   float64 
 29  d8          18102 non-null   float64 
 30  d9          18121 non-null   float64 
 31  d10         18158 non-null   float64 
 32  bottom5     11619 non-null   float64 
 33  top5         12270 non-null   float64 
 34  resource     24367 non-null   object  
 35  resource_detailed 24367 non-null   object  
 36  scale         23861 non-null   object  
 37  scale_detailed 23861 non-null   object  
 38  sharing_unit  23872 non-null   object  
 39  reference_unit 24330 non-null   object  
 40  areacovr     24365 non-null   object  
 41  areacovr_detailed 24365 non-null   object  
 42  popcovr     24367 non-null   object  
 43  popcovr_detailed 24367 non-null   object  
 44  region_un    24230 non-null   object  
 45  region_un_sub 24230 non-null   object  
 46  region_wb    24367 non-null   object  
 47  eu           24367 non-null   object  
 48  oecd          24367 non-null   object  
 49  incomegroup  24213 non-null   object  
 50  mean          17091 non-null   object  
 51  median         15478 non-null   object  
 52  currency       17474 non-null   object  
 53  reference_period 17623 non-null   object  
 54  exchangerate   4031 non-null   float64 
 55  mean_usd      64 non-null    float64 
 56  median_usd    14 non-null    float64 
 57  gdp           24251 non-null   object  
 58  population     24306 non-null   object  
 59  revision       20551 non-null   object  
 60  quality        24367 non-null   object  
 61  quality_score  24367 non-null   int64  
 62  source         24367 non-null   object  
 63  source_detailed 24367 non-null   object  
 64  source_comments 19635 non-null   object  
 65  survey         19126 non-null   object  
 66  link           8615 non-null    object  
 67  wiidcompanion  24367 non-null   object  
dtypes: float64(32), int64(3), object(33)
memory usage: 12.6+ MB
```

	id	year	gini	ge0	ge1	ge2	a025	a050	a075	a1	...	d7	d8	d9	d10	bottom5	top5	exchangerate	mean_usd
count	24367.000000	24367.000000	24279.000000	11723.000000	13395.000000	11784.000000	10989.000000	13325.000000	10989.000000	13209.000000	...	18105.000000	18102.000000	18121.000000	18158.000000	11619.000000	12270.000000	4.031000e+03	64.000000
mean	12184.000000	2003.411622	37.053688	951.307507	814.745016	1566.812687	236.796683	384.168245	681.557025	758.916534	...	10.119569	12.139420	15.385310	28.485110	0.981927	16.825234	6.948416e-01	2793.031250
std	7034.291341	14.585084	9.099320	1109.840124	1036.510708	4173.783279	257.855927	485.786143	741.250193	955.202356	...	0.898464	0.742304	1.007931	6.833391	0.387118	4.560707	6.315205e+00	5251.287207
min	1.000000	1867.000000	12.100000	5.530000	5.960000	6.670000	1.440000	2.800000	4.110000	5.380000	...	4.240000	6.250000	8.000000	16.120000	-1.000000	9.100000	8.090000e-14	32.000000
25%	6092.500000	1997.000000	30.420000	22.650000	23.820000	33.407500	5.240000	11.160000	14.990000	21.510000	...	9.790000	11.800000	14.670000	23.800000	0.700000	14.210000	1.807016e-02	520.500000
50%	12184.000000	2007.000000	35.400000	50.270000	50.400000	136.525000	13.670000	21.380000	34.640000	37.610000	...	10.390000	12.170000	15.400000	26.420000	0.970000	15.810000	1.463825e-01	979.000000
75%	18275.500000	2014.000000	42.785000	1893.650000	1754.240000	2603.402500	466.870000	826.720000	1342.400000	1620.230000	...	10.680000	12.550000	16.070000	31.460000	1.255000	17.830000	4.740837e-01	1959.000000
max	24367.000000	2022.000000	78.600000	5332.580000	5361.630000	159490.990000	1057.520000	2008.490000	3038.050000	4133.090000	...	13.900000	17.900000	24.000000	70.240000	2.460000	60.000000	1.493953e+02	25450.000000

8 rows × 35 columns

Finding the null values in the dataset

df.isnull().sum()
0
id 0
country 0
c3 0
c2 10
year 0
...
source_detailed 0
source_comments 4732
survey 5241
link 15752
wiidcompanion 0

68 rows × 1 columns

dtype: int64

df.isnull() Returns True for missing values and False for non-missing values.

sum() Counts the number of missing values in each column.

df.dtypes

df.dtypes
0
id int64
country object
c3 object
c2 object
year int64
...
source_detailed object
source_comments object
survey object
link object
wiidcompanion object

68 rows × 1 columns

dtype: object

Start coding or generate with AI.

df.columns

Index(['id', 'country', 'c3', 'c2', 'year', 'gini', 'ge0', 'ge1', 'ge2', 'a025', 'a050', 'a075', 'a1', 'a2', 'palma', 'ratio_top20bottom20'],

```
'bottom40', 'q1', 'q2', 'q3', 'q4', 'q5', 'd1', 'd2', 'd3', 'd4', 'd5',
'd6', 'd7', 'd8', 'd9', 'd10', 'bottom5', 'top5', 'resource',
'reference_unit', 'scale', 'scale_detailed', 'sharing_unit',
'popcovr_detailed', 'areacovr', 'areacovr_detailed', 'popcovr',
'oce', 'incomegroup', 'mean', 'median', 'currency', 'reference_period',
'exchangerate', 'mean_usd', 'median_usd', 'gdp', 'population',
'revision', 'quality', 'quality_score', 'source', 'source_detailed',
'source_comments', 'survey', 'link', 'wiidcompanion'],
dtype='object')
```

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

	0
id	0
country	0
c3	0
c2	10
year	0
...	...
source_detailed	0
source_comments	4732
survey	5241
link	15752
wiidcompanion	0

68 rows × 1 columns

dtype: int64

```
df.isnull().sum()[df.isnull().sum() > 0]
```

	0
c2	10
gini	88
ge0	12644
ge1	10972
ge2	12583
a025	13378
a050	11042
a075	13378
a1	11158
a2	12653
palma	5360
ratio_top20bottom20	4502
bottom40	5507
q1	5337
q2	5503
q3	5482
q4	5476
q5	5336
d1	6285
d2	6293
d3	6293
d4	6293
d5	6273
d6	6268
d7	6262
d8	6265
d9	6246
d10	6209
bottom5	12748
top5	12097
scale	506
scale_detailed	506
sharing_unit	495
reference_unit	37
areacovr	2
areacovr_detailed	2
region_un	137
region_un_sub	137
incomegroup	154
mean	7276
median	8889
currency	6893
reference_period	6744
exchangerate	20336
mean_usd	24303
median_usd	24353
gdp	116
population	61
revision	3816
source_comments	4732
survey	5241
link	15752

dtype: int64

```
df.duplicated().sum()
```

	0
df[df['c2'].isnull()][['country']]	0

```
country
14990 Namibia
14991 Namibia
14992 Namibia
14993 Namibia
14994 Namibia
14995 Namibia
14996 Namibia
14997 Namibia
14998 Namibia
14999 Namibia
```

```
df.loc[df['country'] == 'Namibia', 'c2'] = 'NA'
```

```
df['gini'] = df['gini'].astype(float)
df['gini'] = df.groupby('country')['gini'].transform(lambda group: group.interpolate(method='linear', limit_direction='both'))
df['gini'] = df.groupby('country')['gini'].transform(lambda group: group.fillna(group.median()))
df['gini'] = df['gini'].fillna(df['gini'].median())
```

```
df.drop(columns=['link', 'survey', 'source_comments'], inplace=True)
```

```
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0]) # Show only columns with missing values
```

```
c2          10
ge0         12644
ge1         10972
ge2         12583
a025        13378
a050        11042
a075        13378
a1          11158
a2          12653
palma       5360
ratio_top20bottom20 4502
bottom40      5507
q1          5337
q2          5503
q3          5482
q4          5476
q5          5336
d1          6285
d2          6293
d3          6293
d4          6293
d5          6273
d6          6268
d7          6262
d8          6265
d9          6246
d10         6209
bottom5       12748
top5          12097
scale         506
scale_detailed 506
sharing_unit   495
reference_unit 37
areacovr        2
areacovr_detailed 2
region_un       137
region_un_sub    137
incomegroup      154
mean           7276
median          8889
currency         6893
reference_period 6744
exchangerate     20336
mean_usd        24303
median_usd      24353
gdp            116
population       61
revision         3816
source_comments  4732
survey          5241
link            15752
dtype: int64
```

```
import pandas as pd
```

```
# Define numerical and categorical columns
num_cols = ['ge0', 'ge1', 'ge2', 'a025', 'a050', 'a075', 'a1', 'a2', 'palma', 'ratio_top20bottom20',
            'bottom40', 'q1', 'q2', 'q3', 'q4', 'q5', 'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8',
            'd9', 'd10', 'bottom5', 'top5', 'mean', 'median', 'gdp', 'population', 'revision']
```

```
cat_cols = ['currency', 'region_un', 'region_un_sub', 'incomegroup', 'scale', 'scale_detailed',
            'sharing_unit', 'reference_unit', 'areacovr', 'areacovr_detailed']
```

```
# Convert numeric columns to proper format (handling numbers stored as text)
```

```
for col in num_cols:
    df.loc[:, col] = pd.to_numeric(df[col].astype(str).str.replace(',', '').str.strip(), errors='coerce')
```

```
# Fill missing numerical values with median
```

```
for col in num_cols:
    df.loc[:, col] = df[col].fillna(df[col].median())
```

```
# Fill missing categorical values with mode
```

```
for col in cat_cols:
    df.loc[:, col] = df[col].astype(str).str.strip() # Remove extra spaces
    df.loc[:, col] = df[col].fillna(df[col].mode()[0])
```

```
# Check for remaining missing values
```

```
print(df.isnull().sum()[df.isnull().sum() > 0])
```

```
c2          10
gini         88
reference_period 6744
exchangerate     20336
mean_usd        24303
median_usd      24353
revision         24367
source_comments  4732
survey          5241
link            15752
dtype: int64
```

```
<ipython-input-13-34e05eb9e5a5>:17: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behav.
df.loc[:, col] = df[col].fillna(df[col].median())
<ipython-input-13-34e05eb9e5a5>:17: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behav.
df.loc[:, col] = df[col].fillna(df[col].median())
<ipython-input-13-34e05eb9e5a5>:17: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behav.
df.loc[:, col] = df[col].fillna(df[col].median())
<ipython-input-13-34e05eb9e5a5>:17: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behav.
df.loc[:, col] = df[col].fillna(df[col].median())
/usr/local/lib/python3.11/dist-packages/numpy/lib/_nanfunctions_implementation.py:1231: RuntimeWarning: Mean of empty slice
    return np.nanmean(a, axis, out=out, keepdims=keepdims)
<ipython-input-13-34e05eb9e5a5>:17: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to the future behav.
df.loc[:, col] = df[col].fillna(df[col].median())
```

```
df.drop(columns=['revision'], inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24367 entries, 0 to 24366
Data columns (total 64 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   id          24367 non-null   int64  
 1   country     24367 non-null   object  
 2   c3          24367 non-null   object  
 3   c2          24367 non-null   object  
 4   year         24367 non-null   int64  
 5   gini         24367 non-null   float64 
 6   ge0          24367 non-null   float64 
 7   ge1          24367 non-null   float64 
 8   ge2          24367 non-null   float64 
 9   a025         24367 non-null   float64 
 10  a050         24367 non-null   float64 
 11  a075         24367 non-null   float64 
 12  a1          24367 non-null   float64
```

```

13 a2           24367 non-null float64
14 palma        24367 non-null float64
15 ratio_top20bottom20 24367 non-null float64
16 bottom40     24367 non-null float64
17 q1           24367 non-null float64
18 q2           24367 non-null float64
19 q3           24367 non-null float64
20 q4           24367 non-null float64
21 q5           24367 non-null float64
22 d1           24367 non-null float64
23 d2           24367 non-null float64
24 d3           24367 non-null float64
25 d4           24367 non-null float64
26 d5           24367 non-null float64
27 d6           24367 non-null float64
28 d7           24367 non-null float64
29 d8           24367 non-null float64
30 d9           24367 non-null float64
31 d10          24367 non-null float64
32 bottom5       24367 non-null float64
33 top5          24367 non-null float64
34 resource      24367 non-null object
35 resource_detailed 24367 non-null object
36 scale          24367 non-null object
37 scale_detailed 24367 non-null object
38 sharing_unit   24367 non-null object
39 reference_unit 24367 non-null object
40 areacovr      24367 non-null object
41 areacovr_detailed 24367 non-null object
42 popcovr       24367 non-null object
43 popcovr_detailed 24367 non-null object
44 region_un     24367 non-null object
45 region_un_sub 24367 non-null object
46 region_wb     24367 non-null object
47 eu             24367 non-null object
48 oecd           24367 non-null object
49 incomegroup    24367 non-null object
50 mean           24367 non-null float64
51 median          24367 non-null float64

```

```

# Basic statistical summary for numerical columns
num_cols = ['ge0', 'ge1', 'ge2', 'a025', 'a050', 'a075', 'a1', 'a2', 'palma', 'ratio_top20bottom20',
            'bottom40', 'q1', 'q2', 'q3', 'q4', 'q5', 'd1', 'd2', 'd3', 'd4', 'd5', 'd6', 'd7', 'd8',
            'd9', 'd10', 'bottom5', 'top5', 'mean', 'median', 'gdp', 'population', 'exchangerate',
            'mean_usd', 'median_usd', 'revision']

```

```

# Statistical summary for numerical columns
num_summary = df[num_cols].describe().T
num_summary["skewness"] = df[num_cols].skew(numeric_only=True)

# Basic statistical summary for categorical columns
cat_cols = ['currency', 'region_un', 'region_un_sub', 'incomegroup', 'scale', 'scale_detailed',
            'sharing_unit', 'reference_unit', 'areacovr', 'areacovr_detailed']

cat_summary = df[cat_cols].describe().T

```

```

# Display results
print("✖ Numerical Columns Summary:\n", num_summary)
print("\n✖ Categorical Columns Summary:\n", cat_summary)

```

	count	mean	std	min	\
ge0	11723.0	951.307507	1109.840124	5.530000e+00	
ge1	13395.0	814.745016	1036.510708	5.960000e+00	
ge2	11784.0	1566.812687	4173.783279	6.670000e+00	
a025	10989.0	236.796683	257.855927	1.440000e+00	
a050	13325.0	384.168245	485.786143	2.800000e+00	
a075	10989.0	681.557025	741.250193	4.110000e+00	
a1	13209.0	758.916534	955.202356	5.380000e+00	
a2	11714.0	1985.510212	2295.813624	1.074000e+01	
palma	19007.0	1.808576	1.254811	5.200000e-01	
ratio_top20bottom20	19865.0	8.145493	6.797569	1.890000e+00	
bottom40	18860.0	17.933425	4.302160	1.600000e+00	
q1	19030.0	6.531106	2.074778	2.000000e-01	
q2	18864.0	11.403914	2.306268	1.300000e+00	
q3	18885.0	15.941342	2.119216	4.200000e+00	
q4	18891.0	22.266236	1.597061	1.060000e+01	
q5	19031.0	43.877454	7.354925	2.550000e+01	
d1	18082.0	2.469968	0.950078	-1.000000e-01	
d2	18074.0	4.077017	1.138688	2.000000e-01	
d3	18074.0	5.178075	1.166559	5.000000e-01	
d4	18074.0	6.235170	1.150456	8.000000e-01	
d5	18094.0	7.343845	1.105887	1.000000e-01	
d6	18099.0	8.598824	1.021466	2.700000e+00	
d7	18105.0	10.119569	0.898464	4.240000e+00	
d8	18102.0	12.139420	0.742304	6.250000e+00	
d9	18121.0	15.385310	1.007931	8.000000e+00	
d10	18158.0	28.485110	6.833391	1.612000e+01	
bottom5	11619.0	0.981927	0.387118	-1.000000e+00	
top5	12270.0	16.825234	4.560707	9.100000e+00	
exchangerate	4031.0	0.694842	6.315205	8.000000e-14	
mean_usd	64.0	2793.031250	5251.287207	3.200000e+01	
median_usd	14.0	4338.428571	5458.003040	5.710000e+02	
	25%	50%	75%	max	\
ge0	22.65000	50.270000	1893.650000	5332.580000	
ge1	23.82000	50.400000	1754.240000	5361.630000	
ge2	33.40750	136.525000	2603.402500	159490.990000	
a025	5.24000	13.670000	466.870000	1057.520000	
a050	11.16000	21.380000	826.720000	2008.490000	
a075	14.99000	34.640000	1342.400000	3038.050000	
a1	21.51000	37.610000	1620.230000	4133.090000	
a2	45.19250	70.620000	4110.967500	8397.590000	
palma	1.12000	1.410000	2.030000	39.810000	
ratio_top20bottom20	4.76000	6.300000	9.200000	293.000000	
bottom40	15.00000	18.355000	21.160000	31.160000	
q1	5.03250	6.580000	8.100000	14.600000	
q2	9.91000	11.790000	13.100000	17.180000	
q3	14.92000	16.560000	17.420000	21.100000	
q4	21.70000	22.570000	23.190000	31.800000	
q5	38.70000	42.010000	47.655000	81.400000	
d1	1.79000	2.450000	3.180000	7.000000	
d2	3.27000	4.150000	4.950000	7.610000	
d3	4.40000	5.330000	6.040000	8.200000	
d4	5.54000	6.470000	7.080000	8.700000	
d5	6.75000	7.630000	8.130000	9.530000	
d6	8.14000	8.920000	9.290000	10.780000	
d7	9.79000	10.390000	10.680000	13.900000	

```
# Set plot style
sns.set(style="whitegrid")
```

```

# Univariate Analysis for Numerical Columns
for col in num_cols:
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

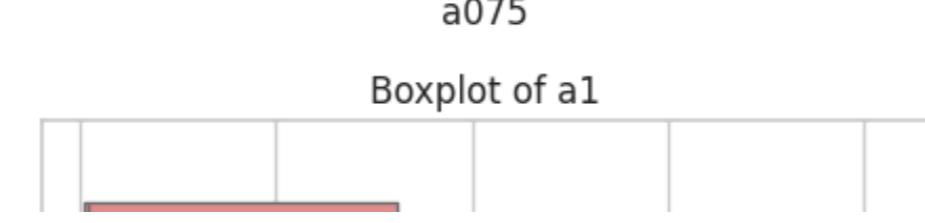
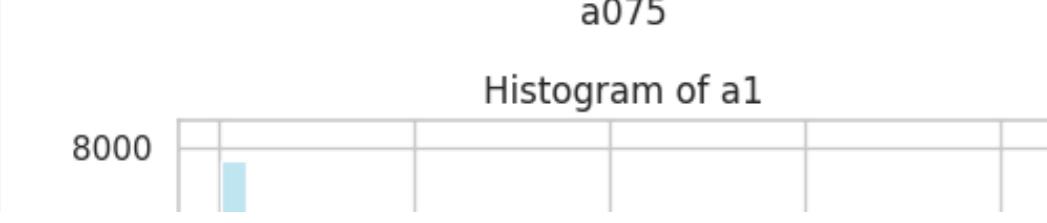
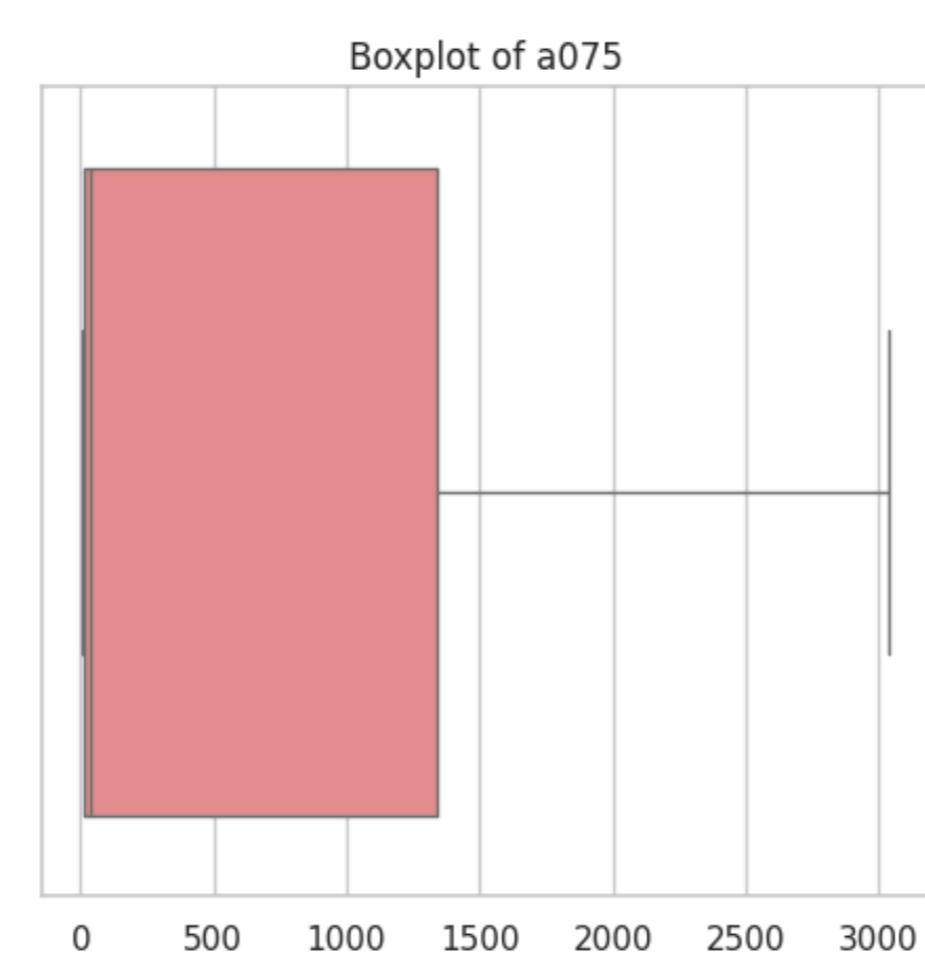
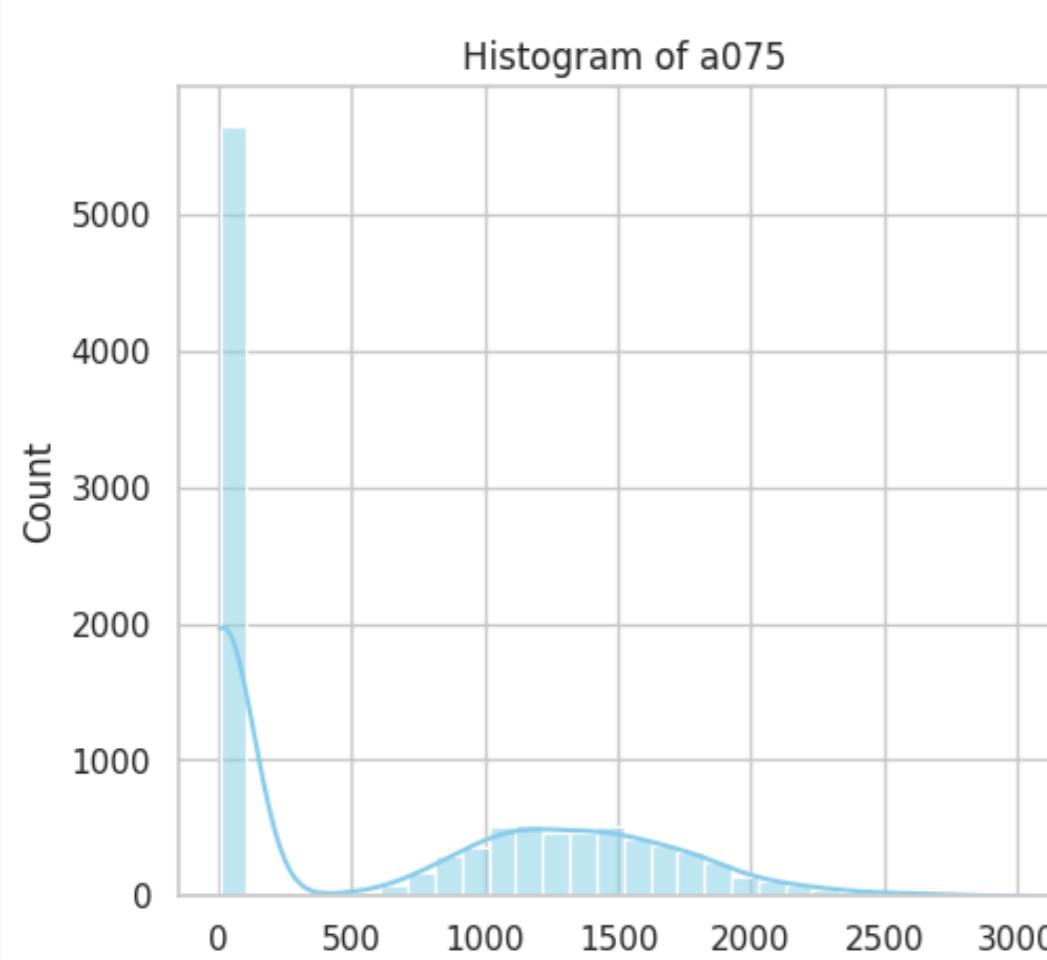
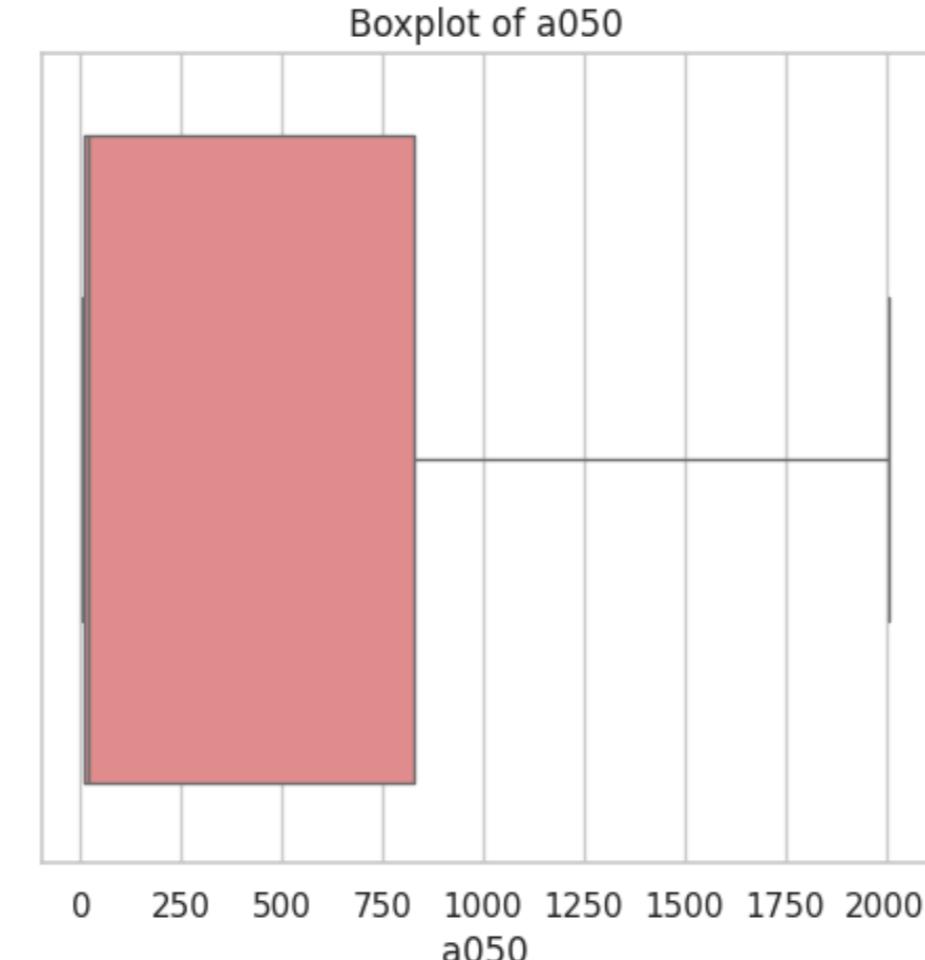
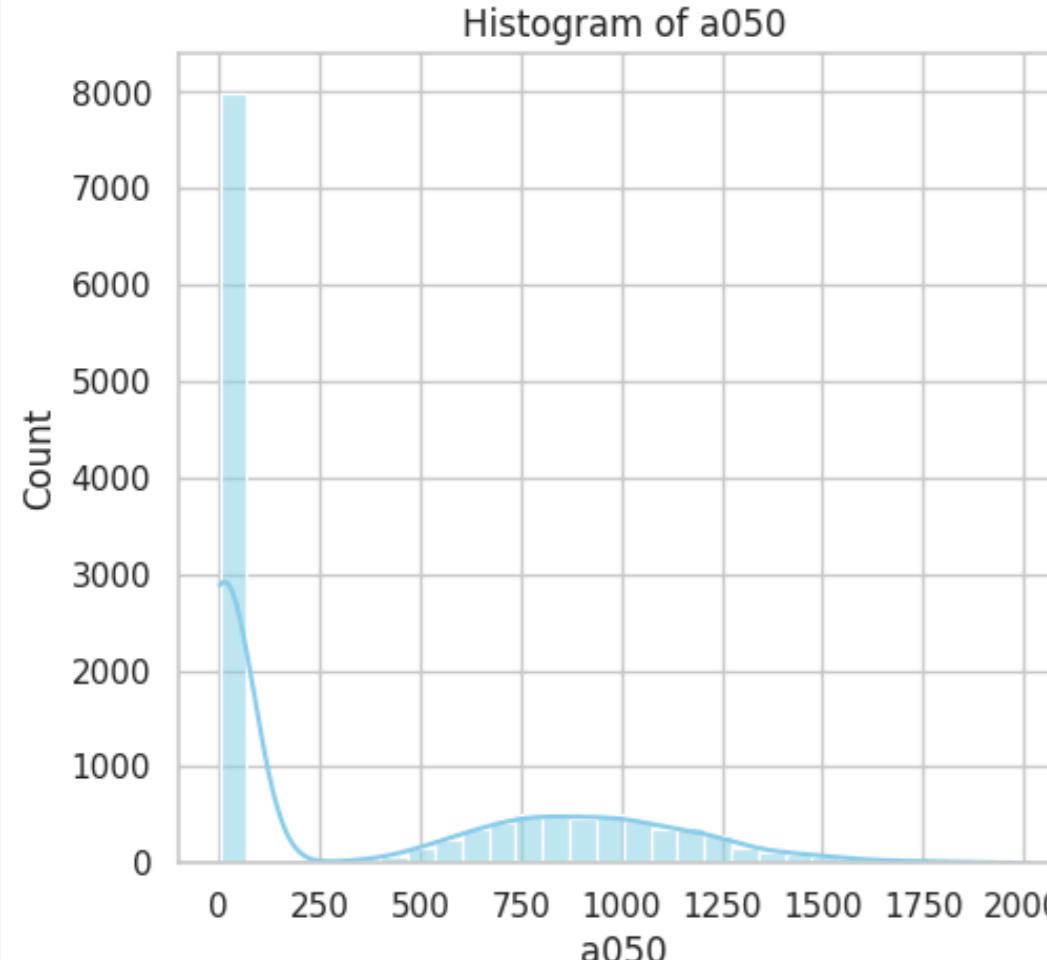
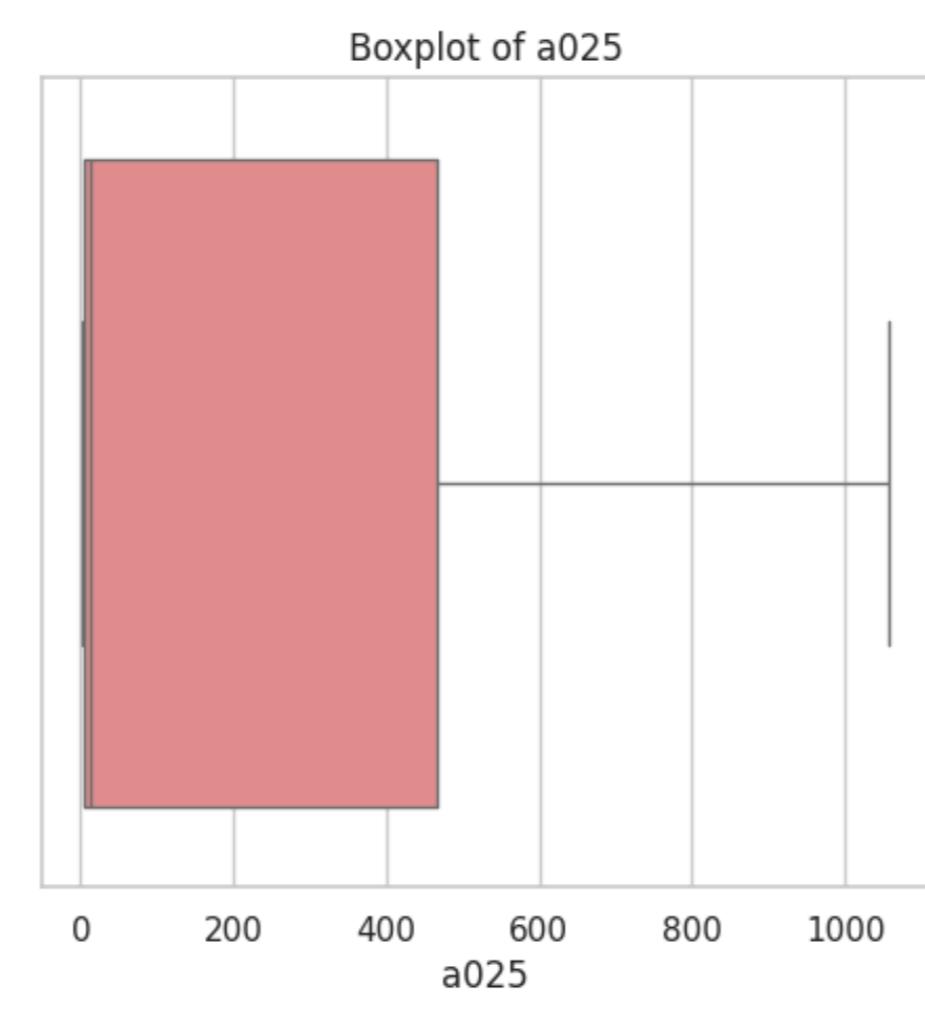
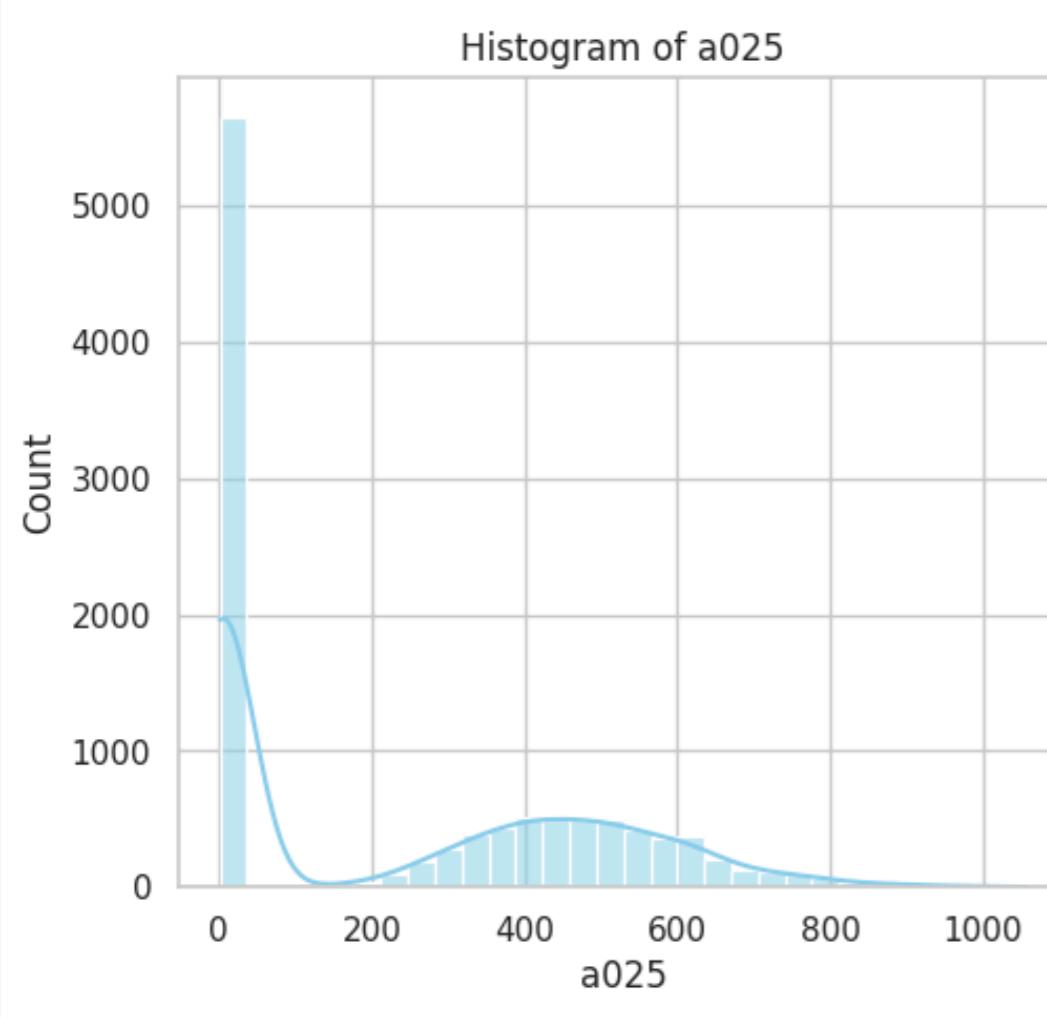
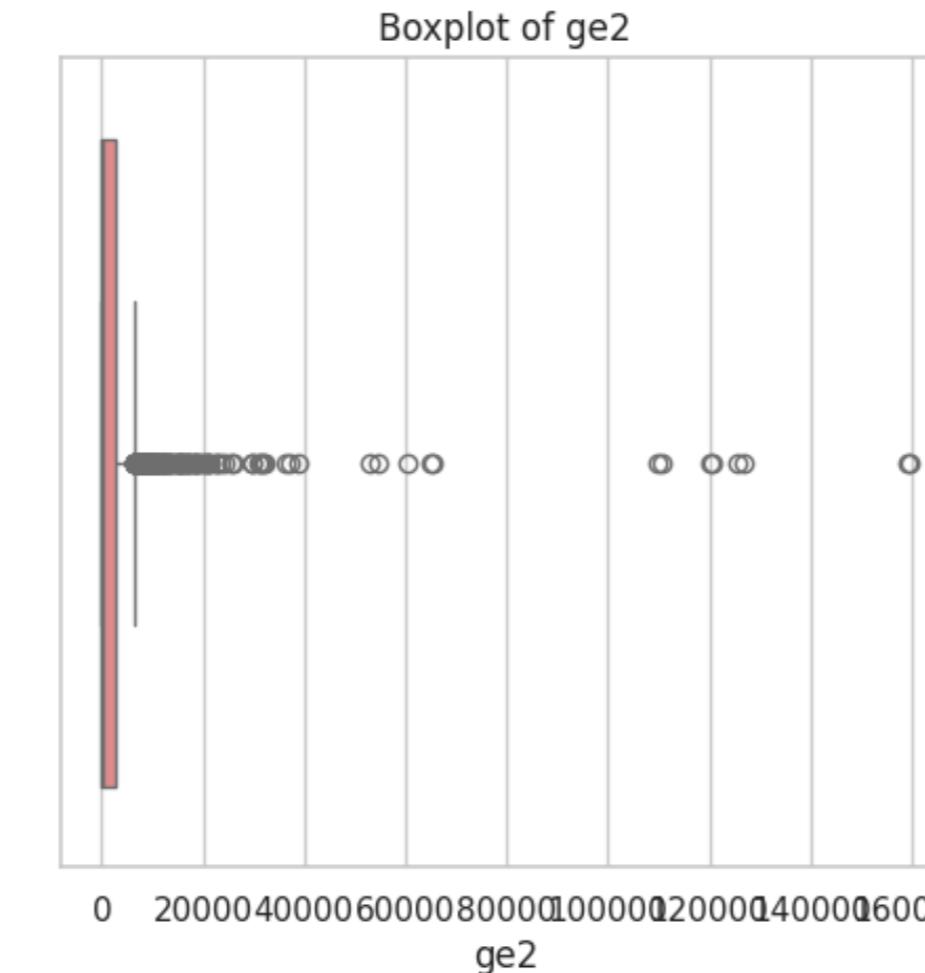
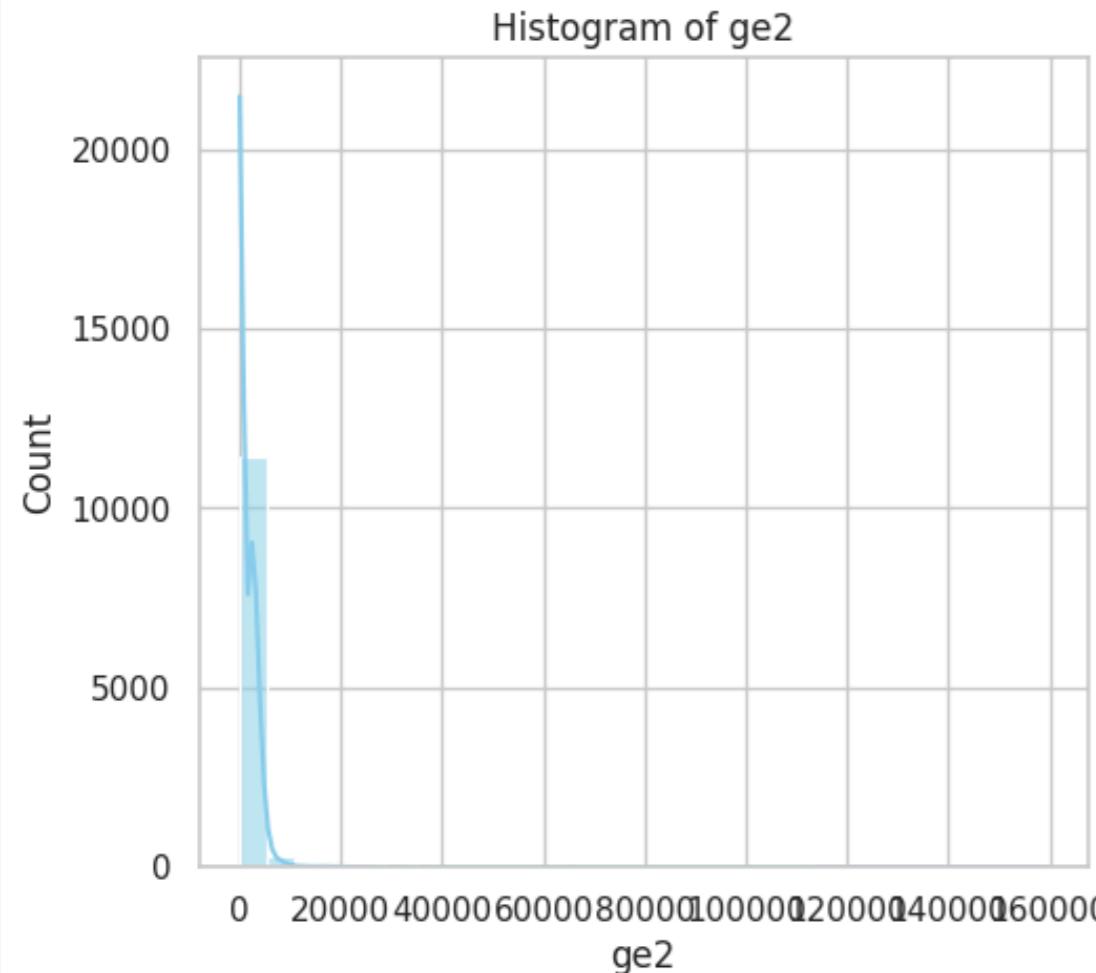
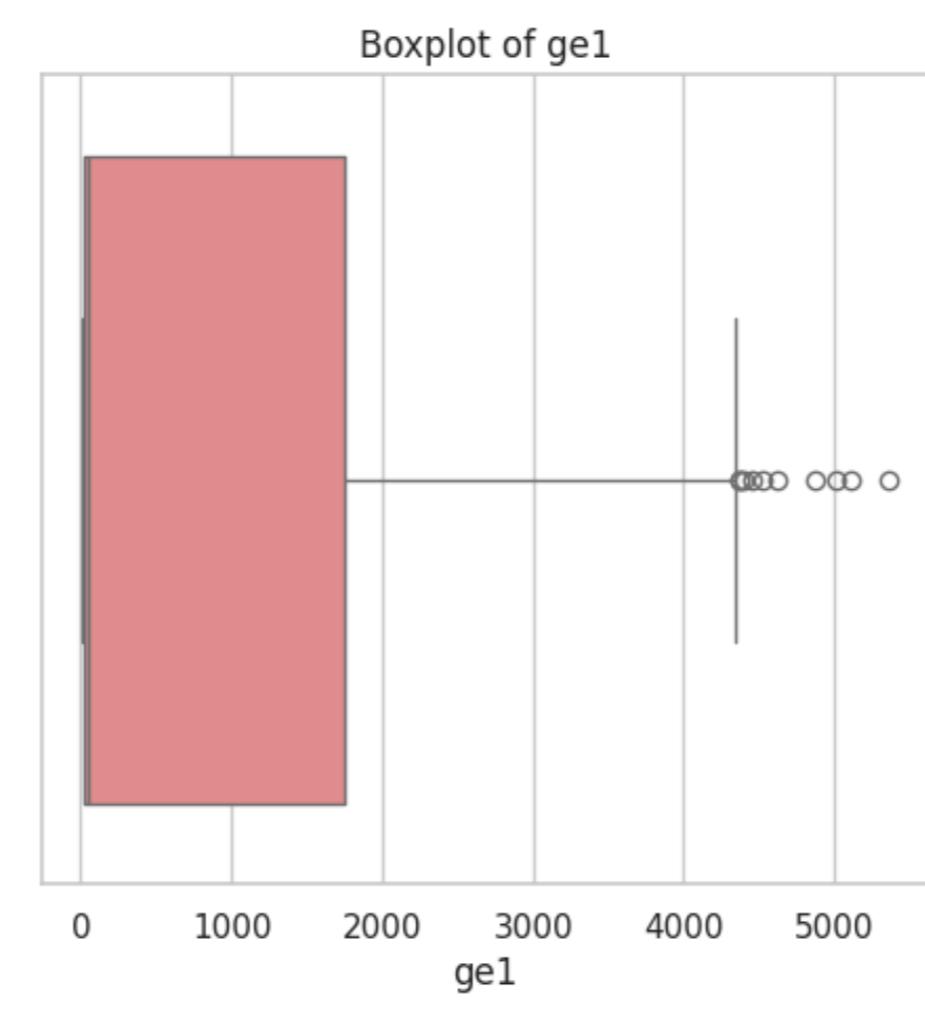
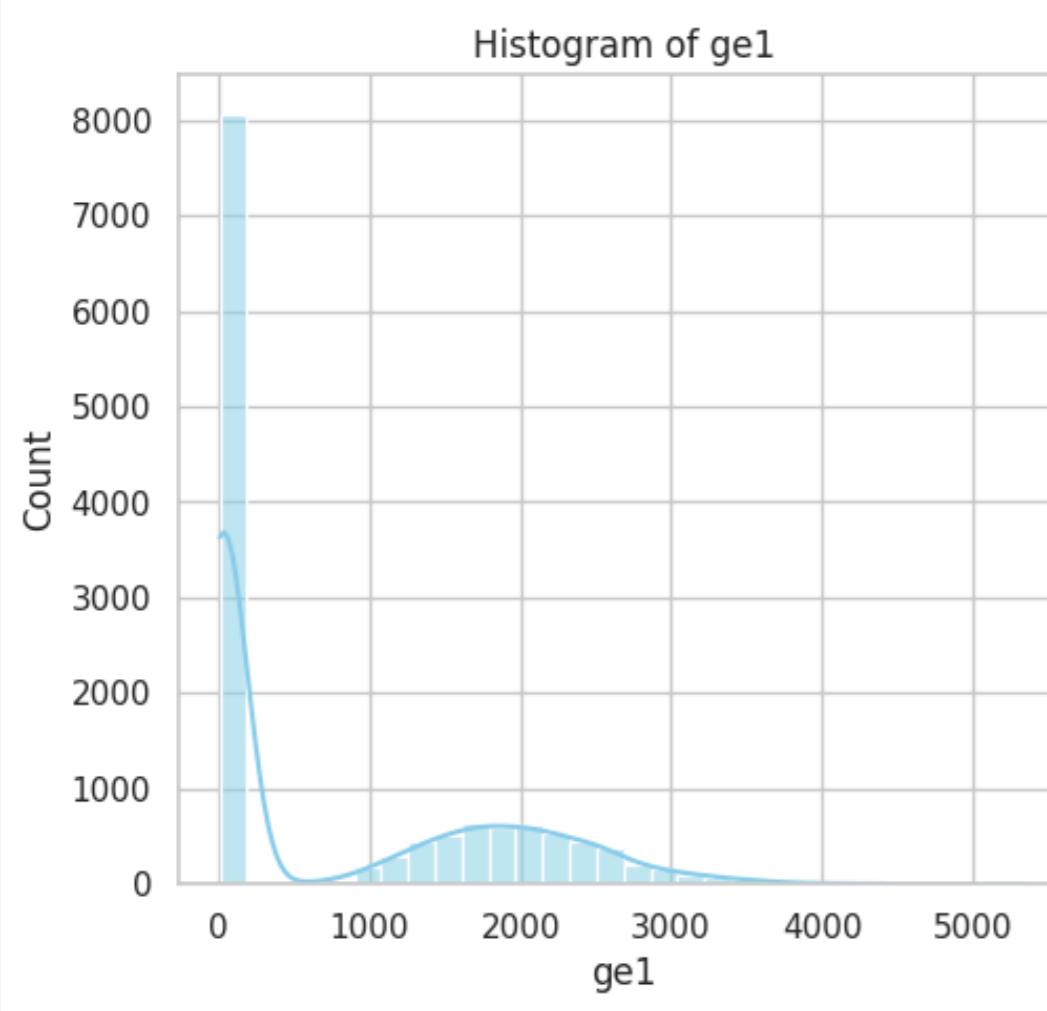
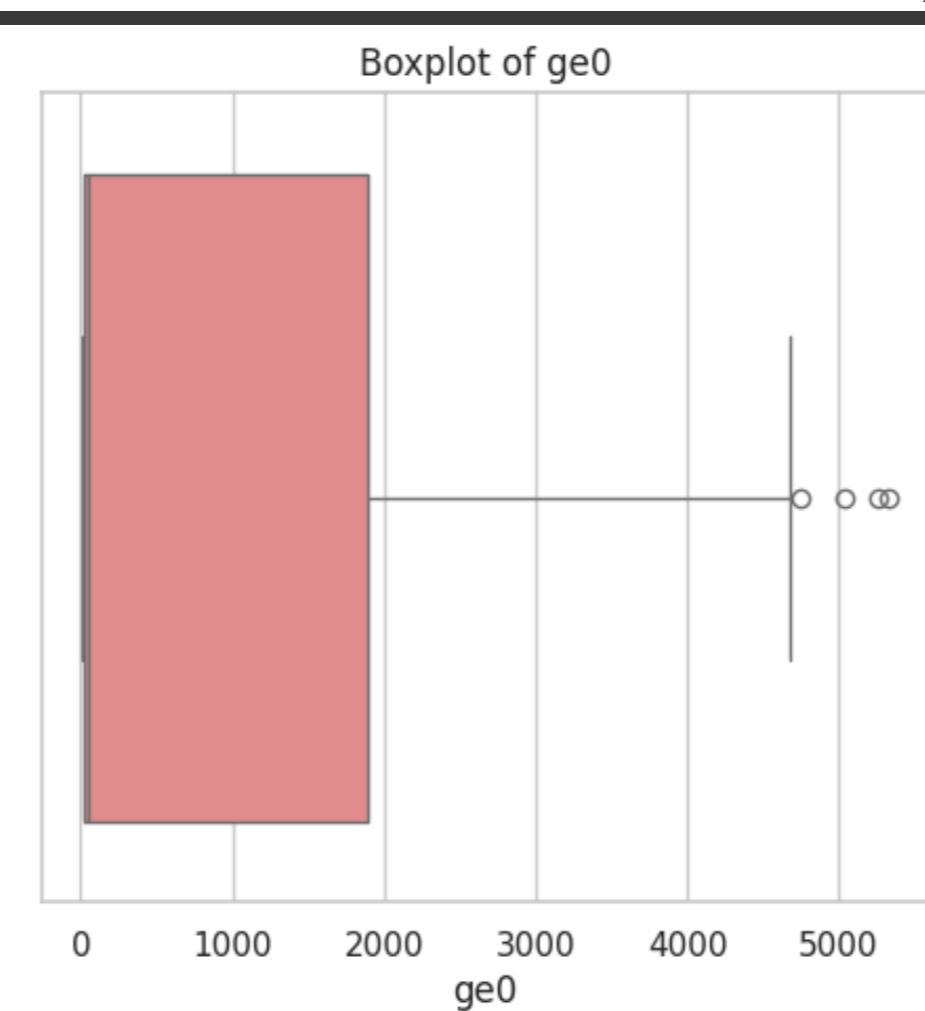
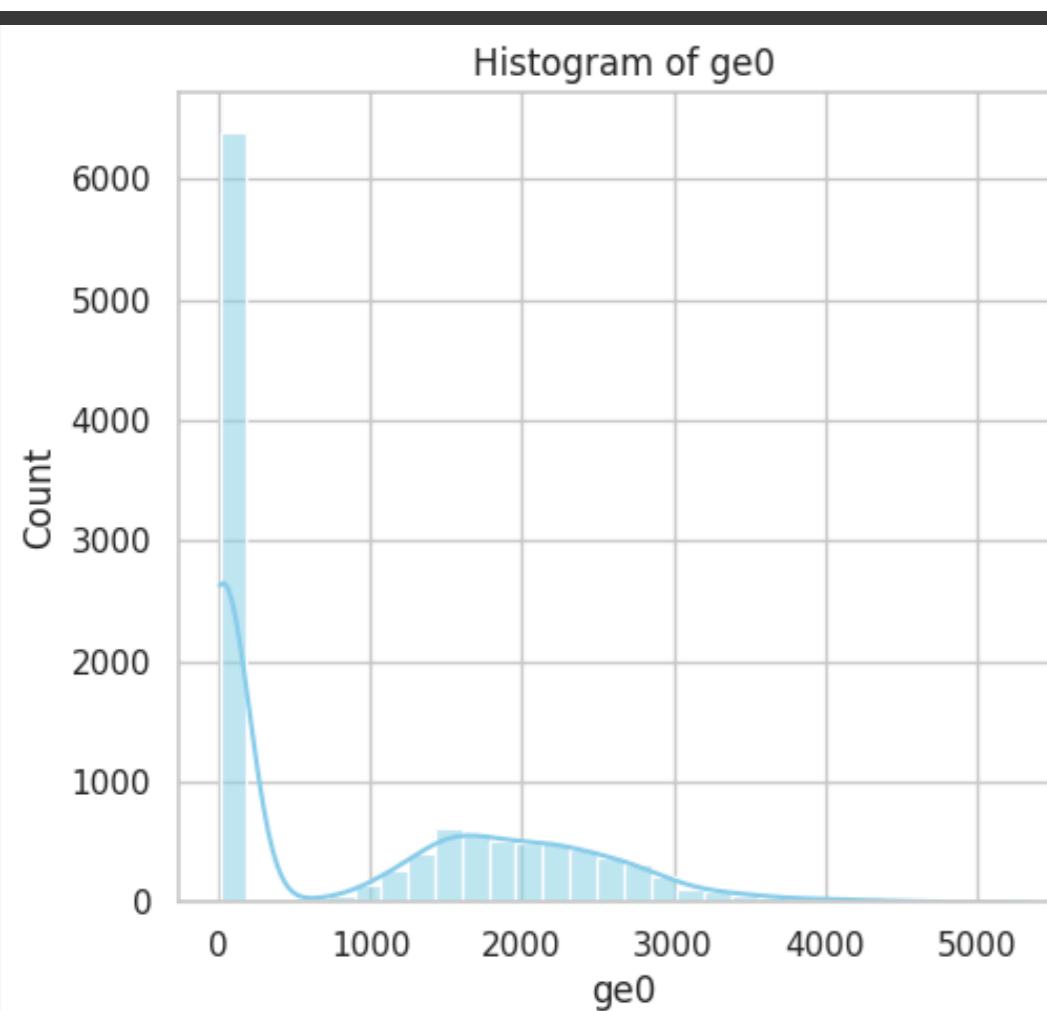
    # Histogram
    sns.histplot(df[col], kde=True, bins=30, ax=axes[0], color="skyblue")
    axes[0].set_title(f"Histogram of {col}")

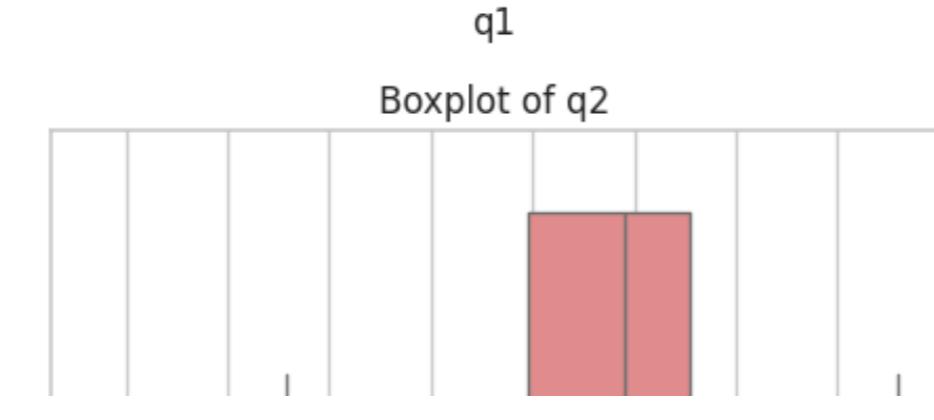
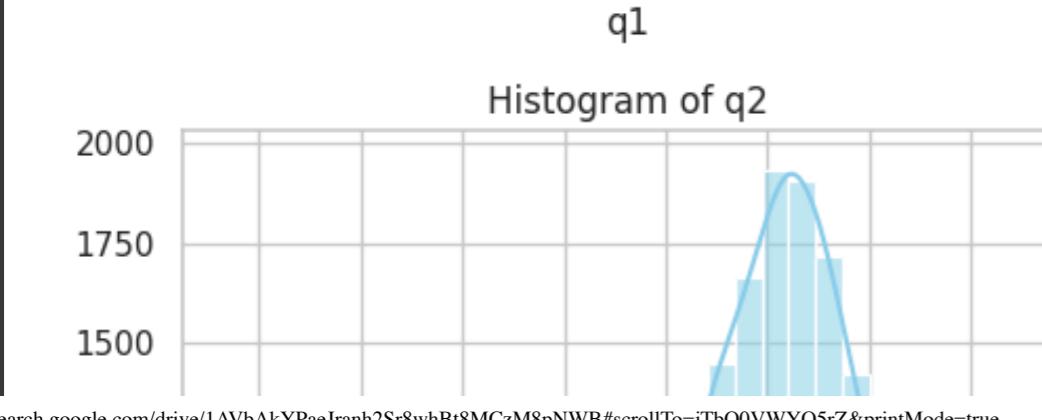
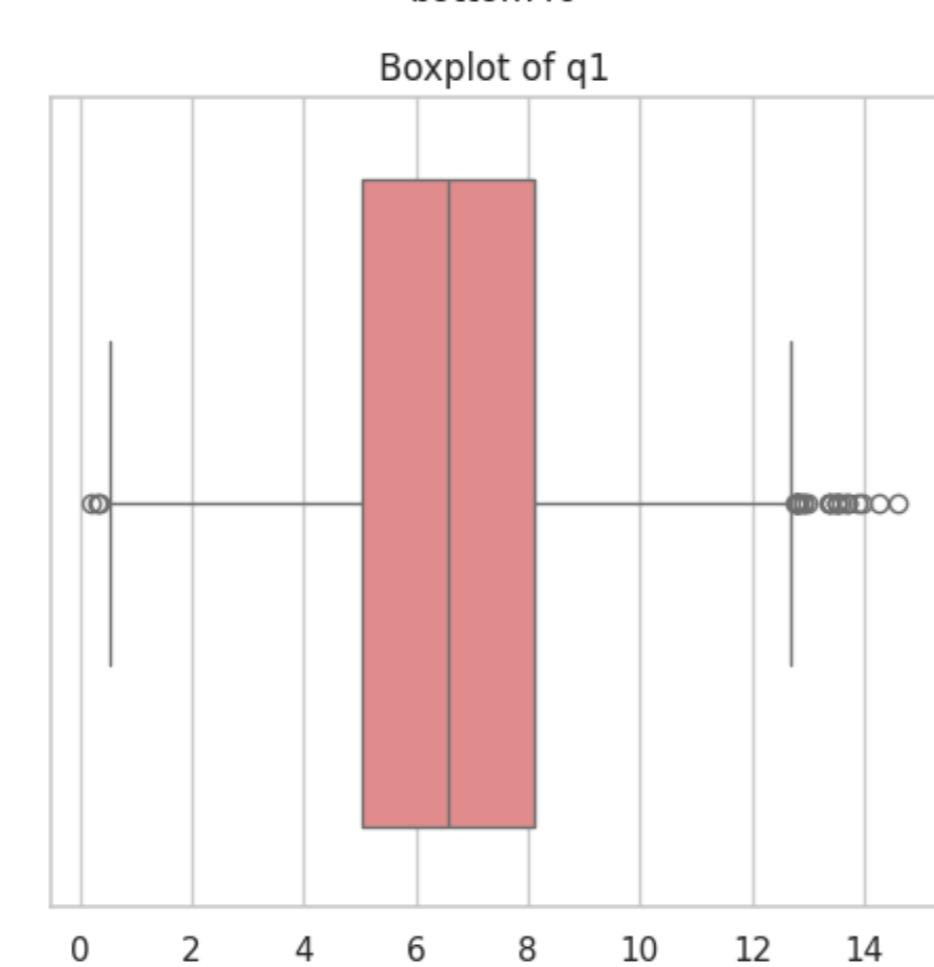
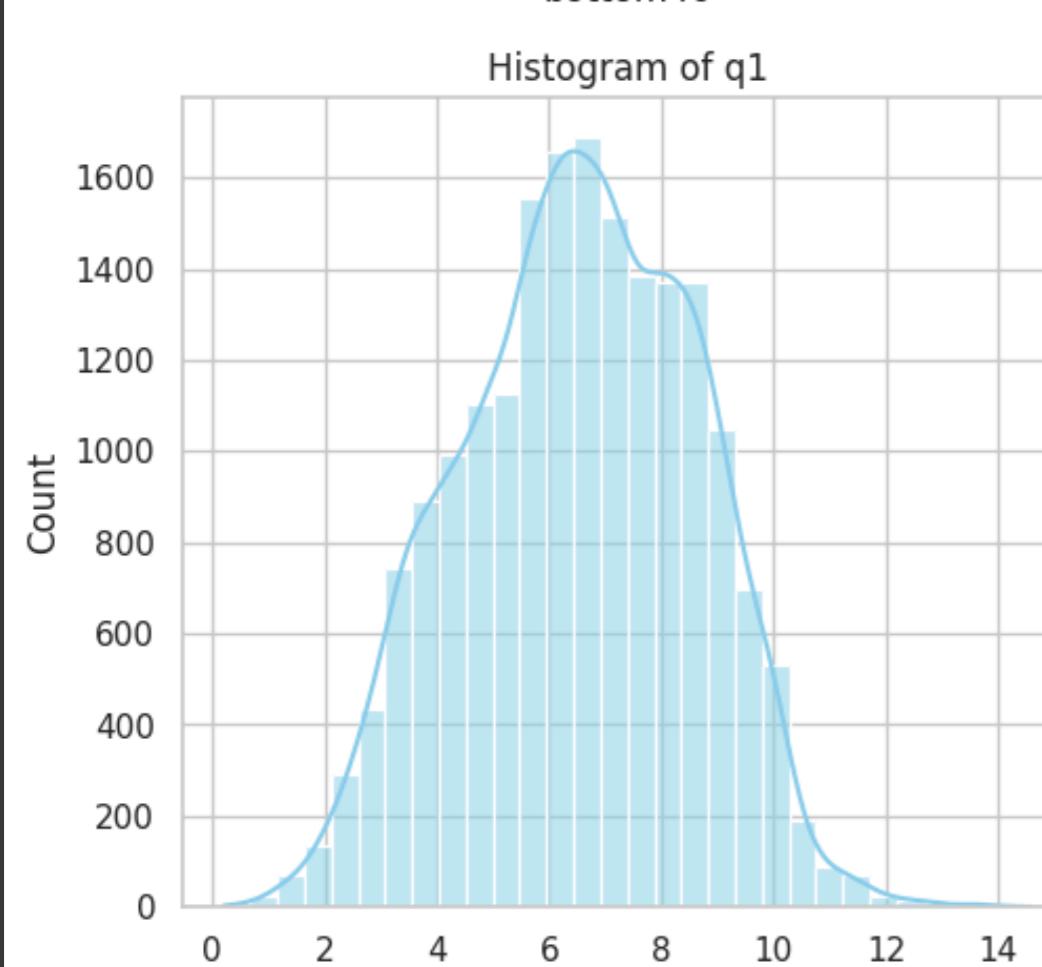
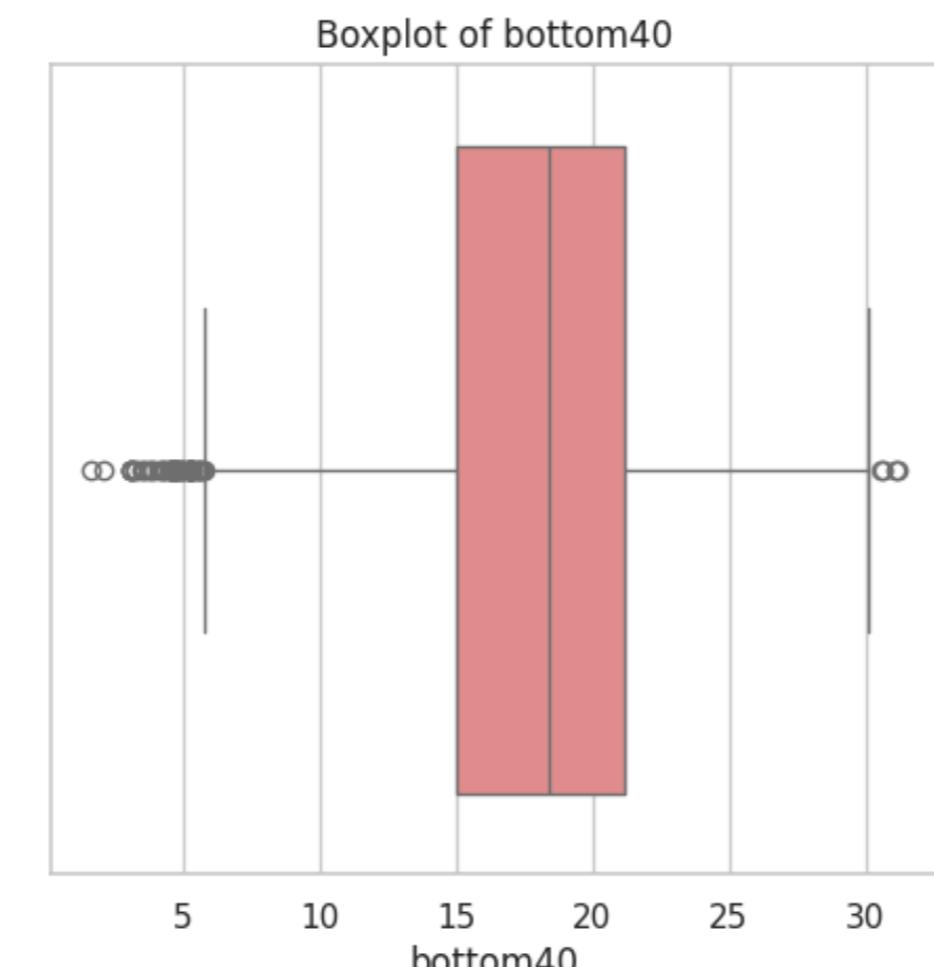
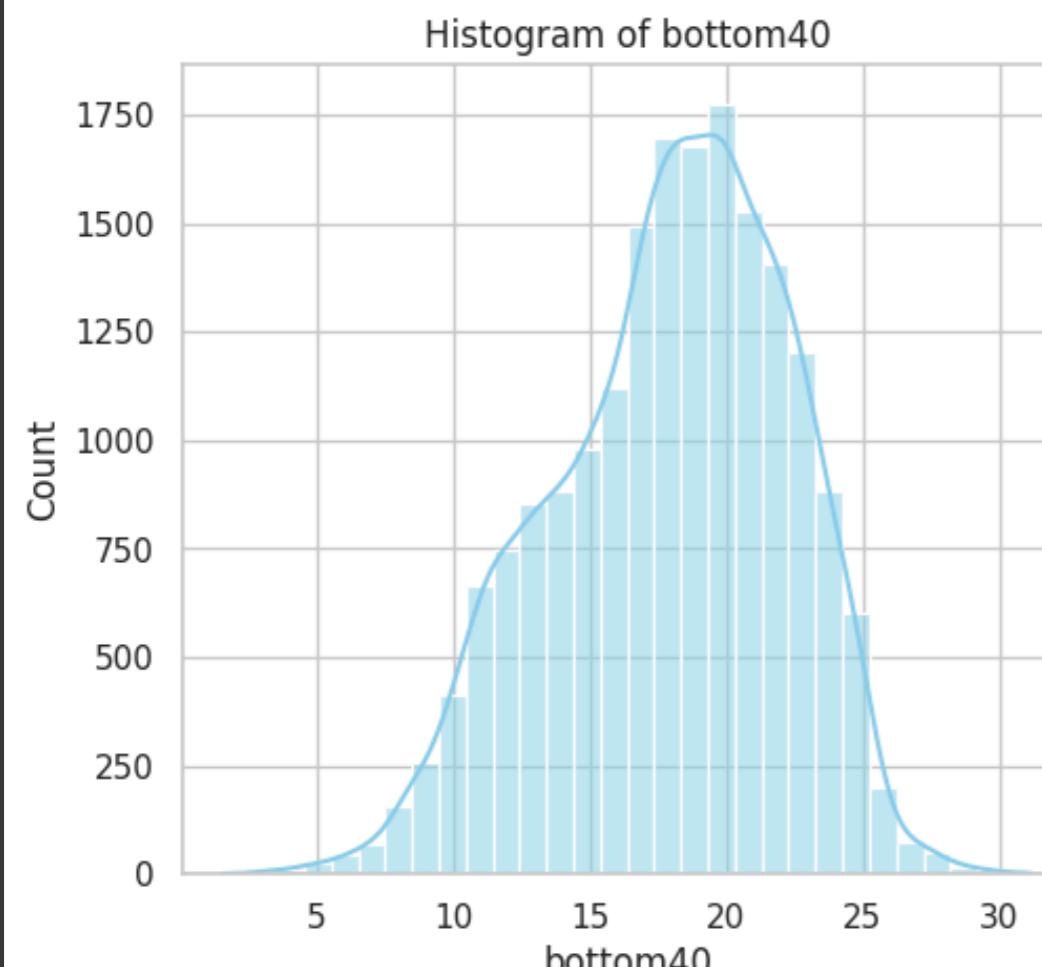
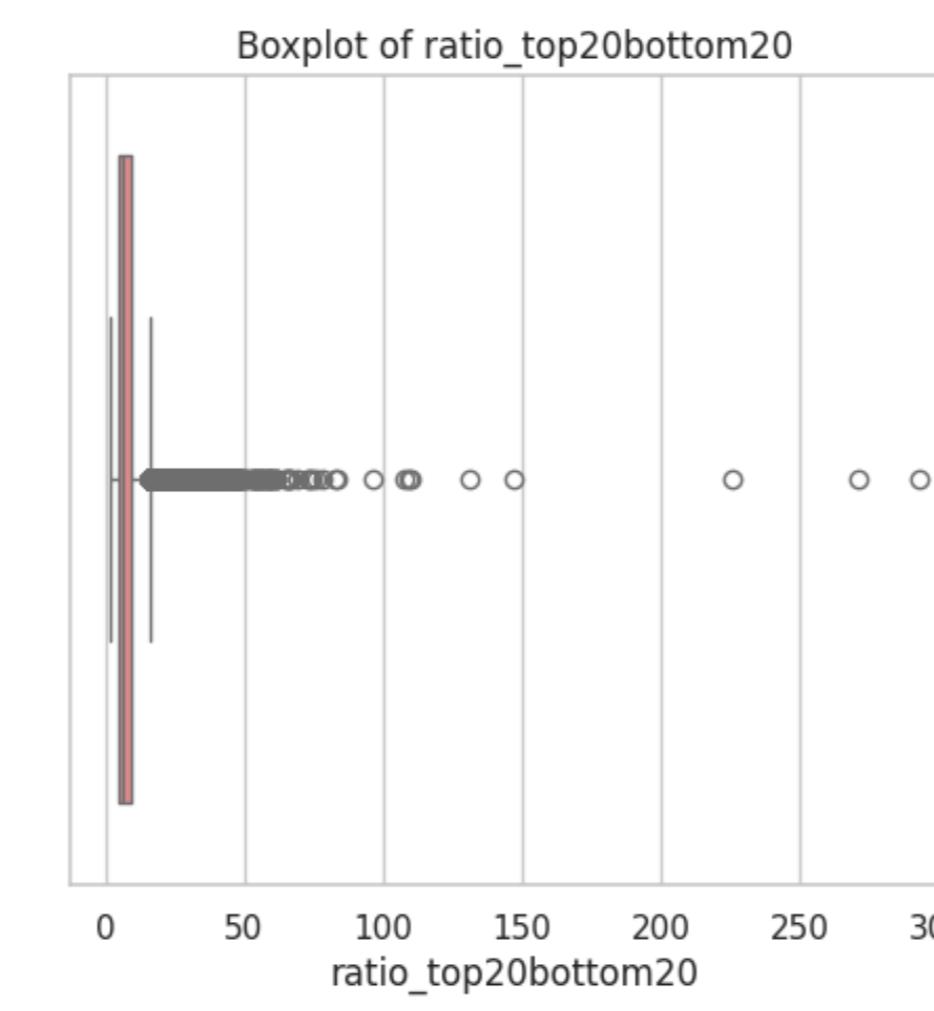
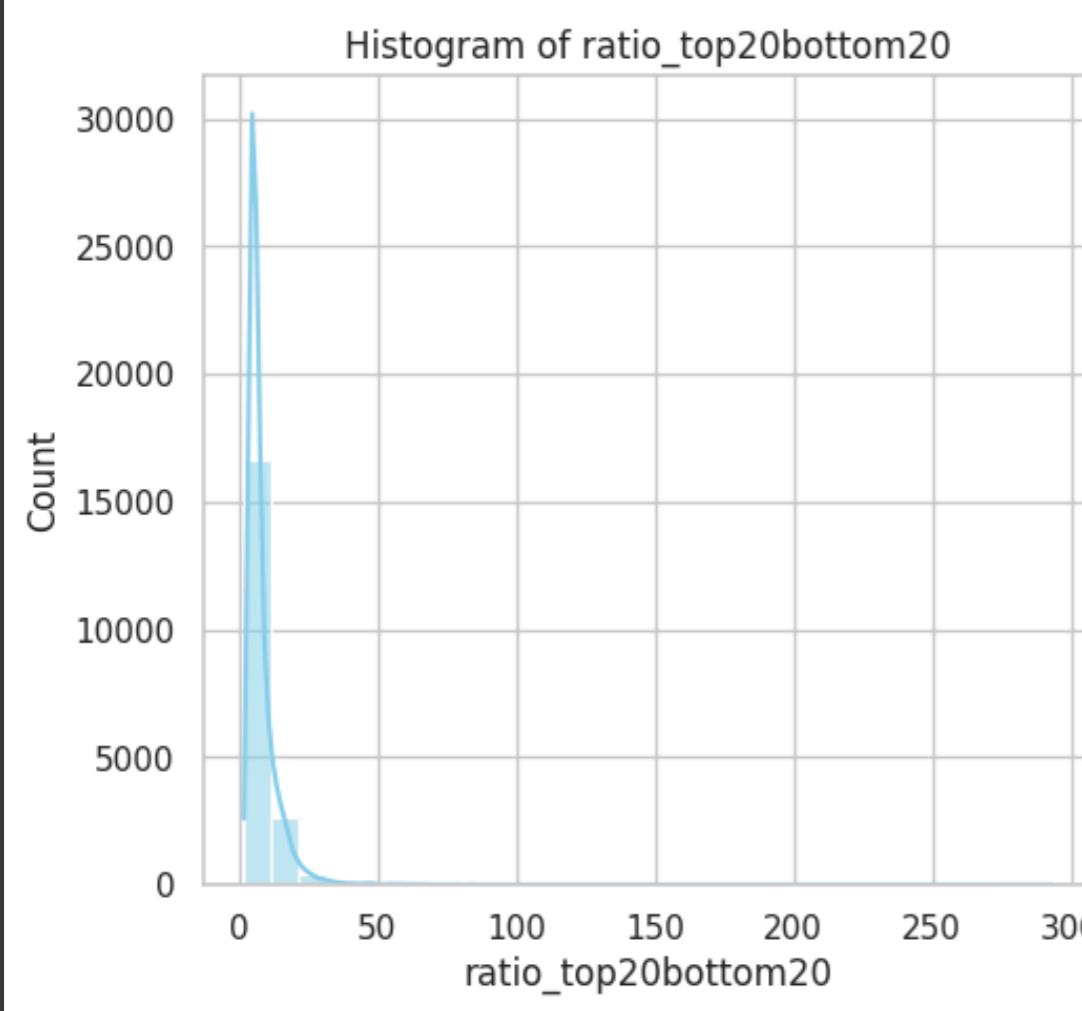
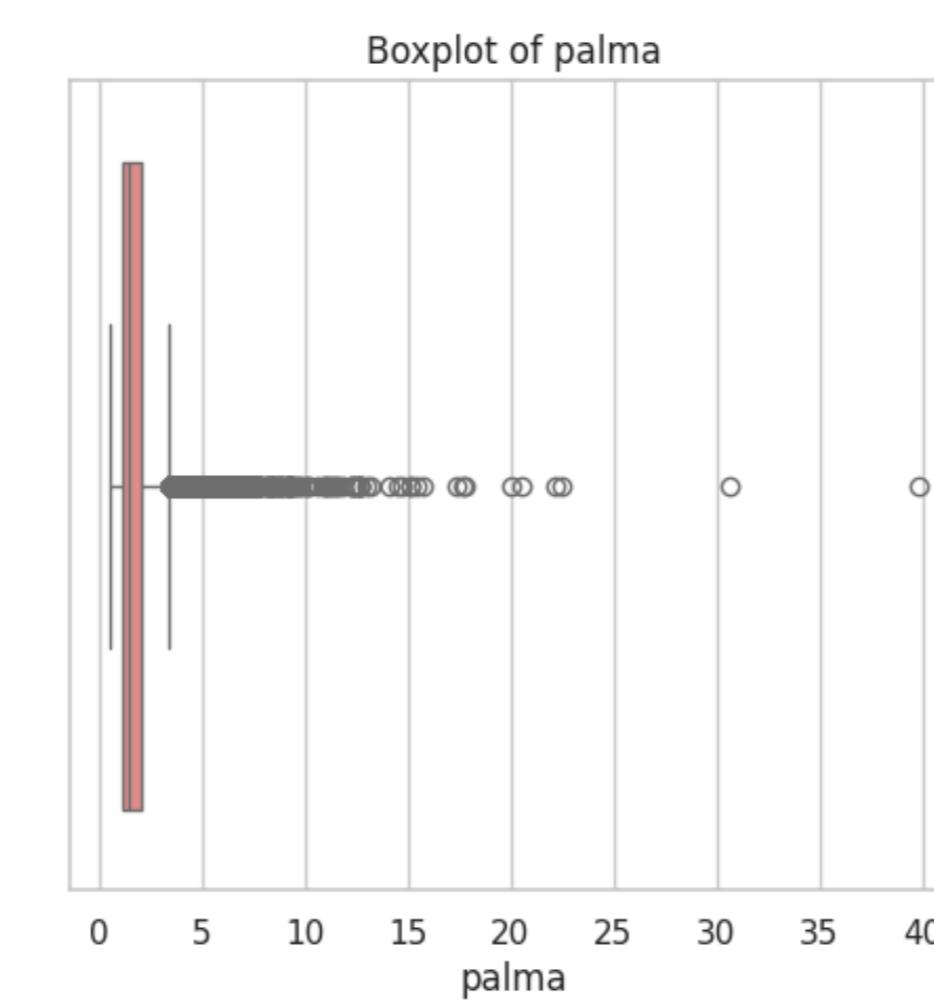
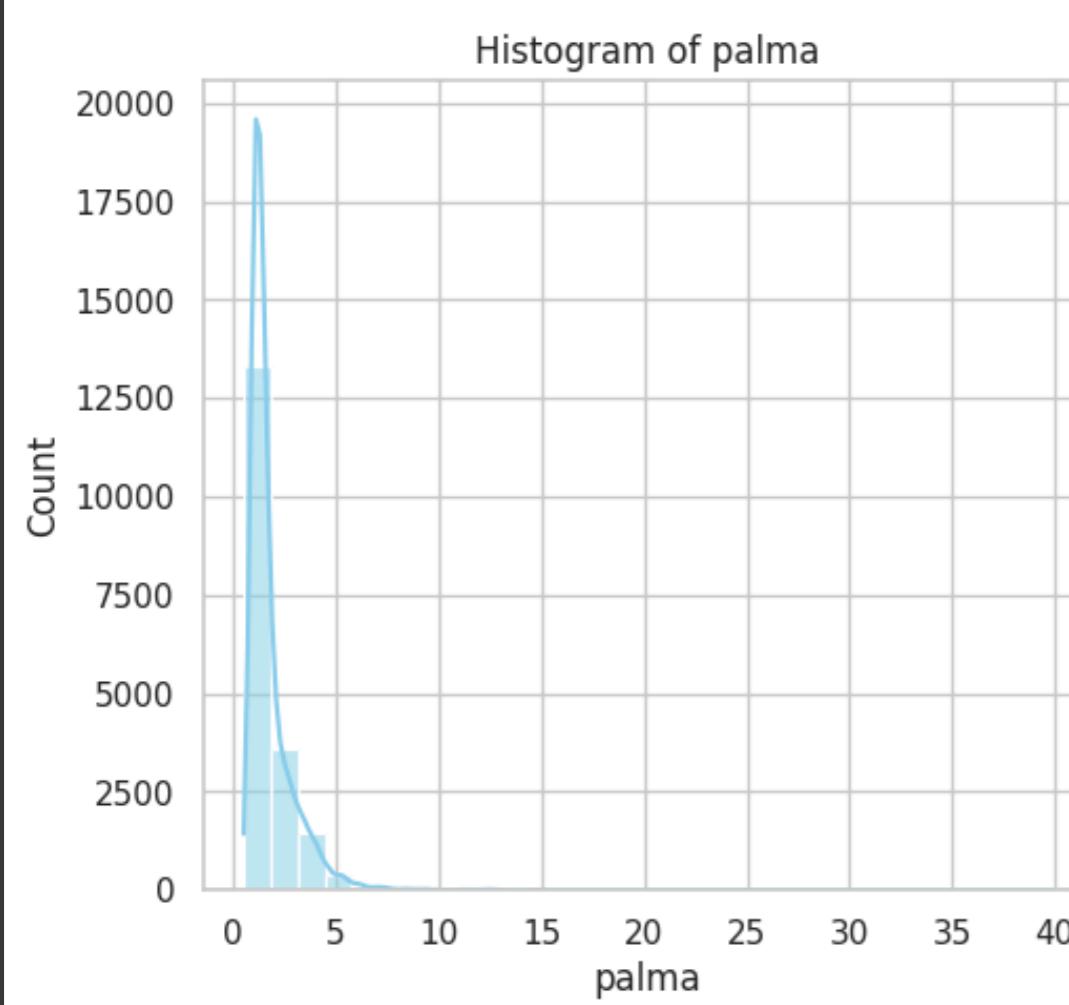
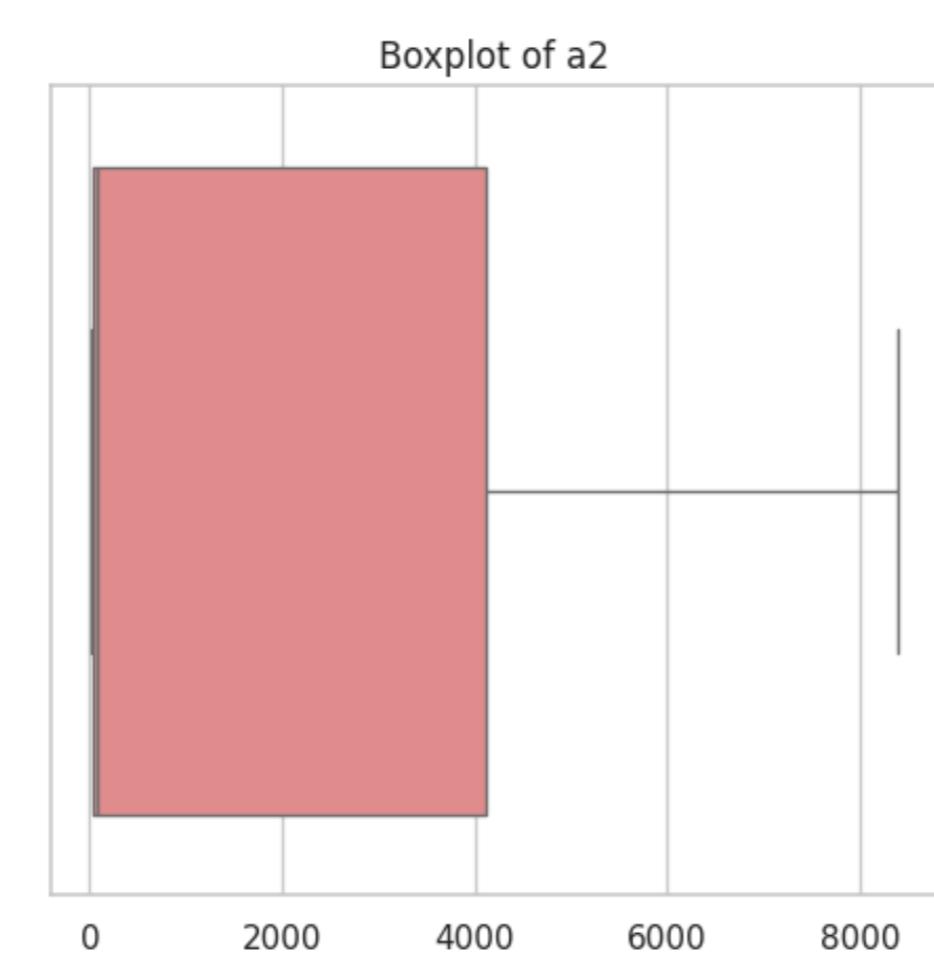
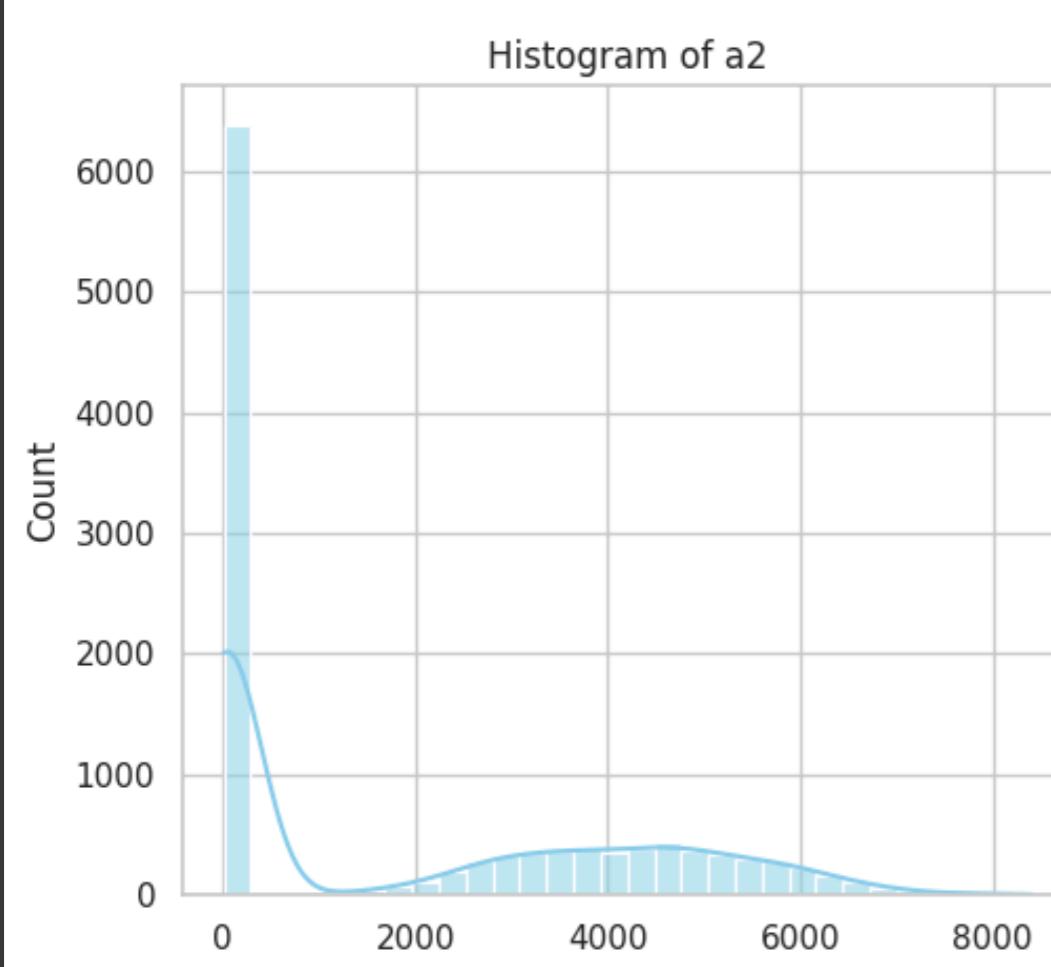
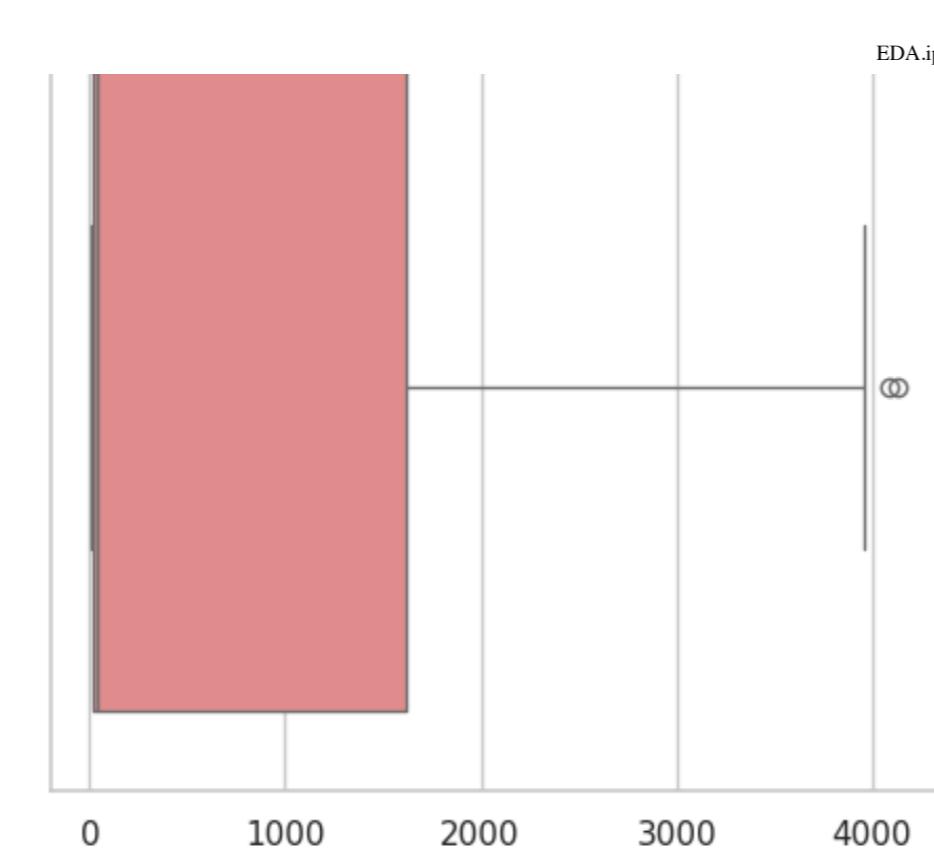
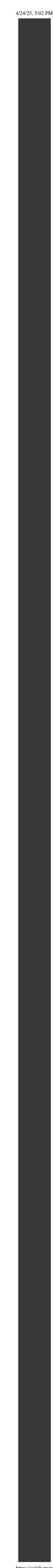
    # Boxplot
    sns.boxplot(x=df[col], ax=axes[1], color="lightcoral")
    axes[1].set_title(f"Boxplot of {col}")

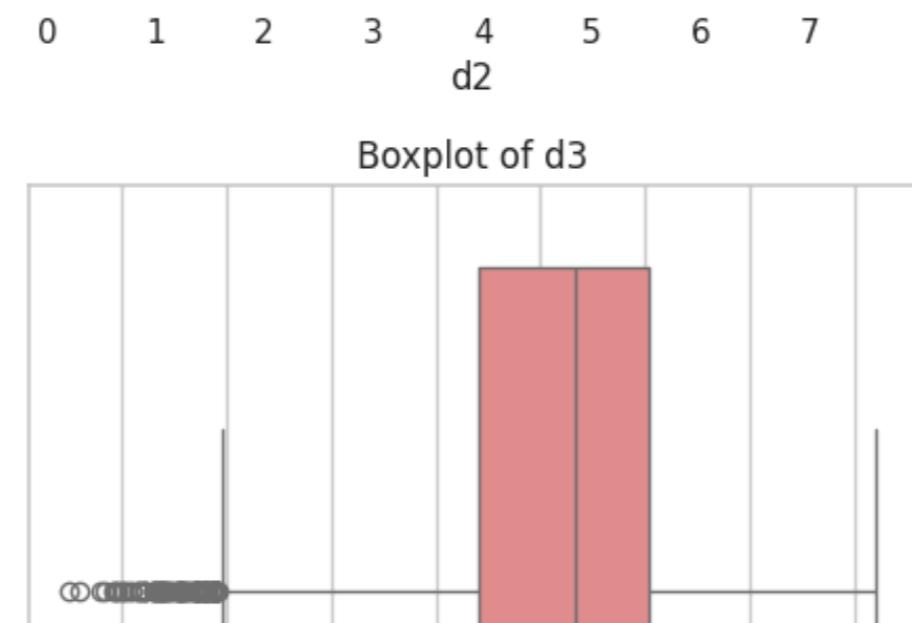
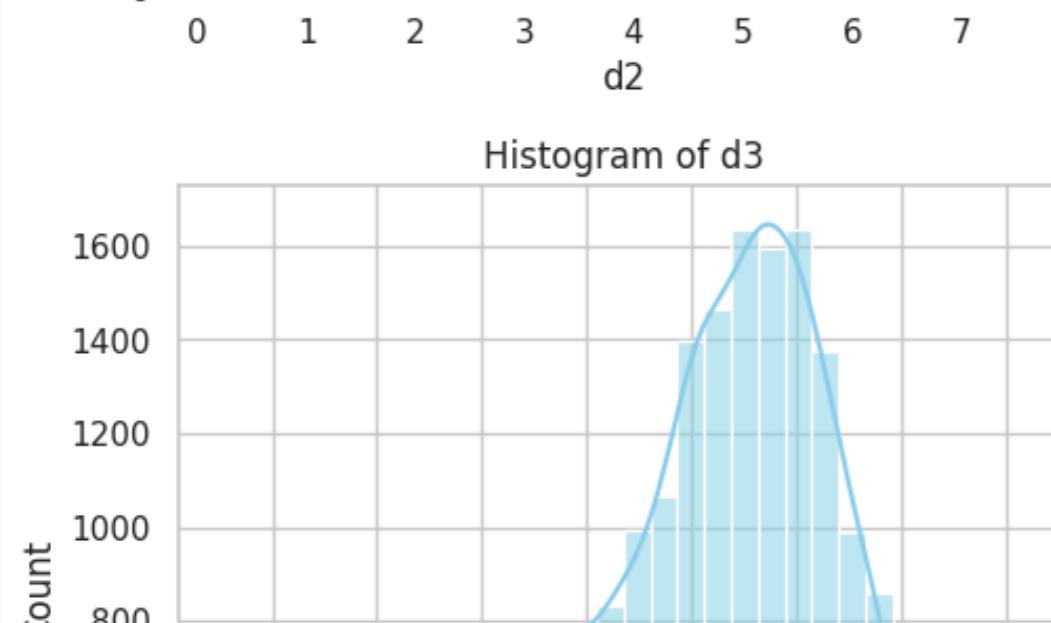
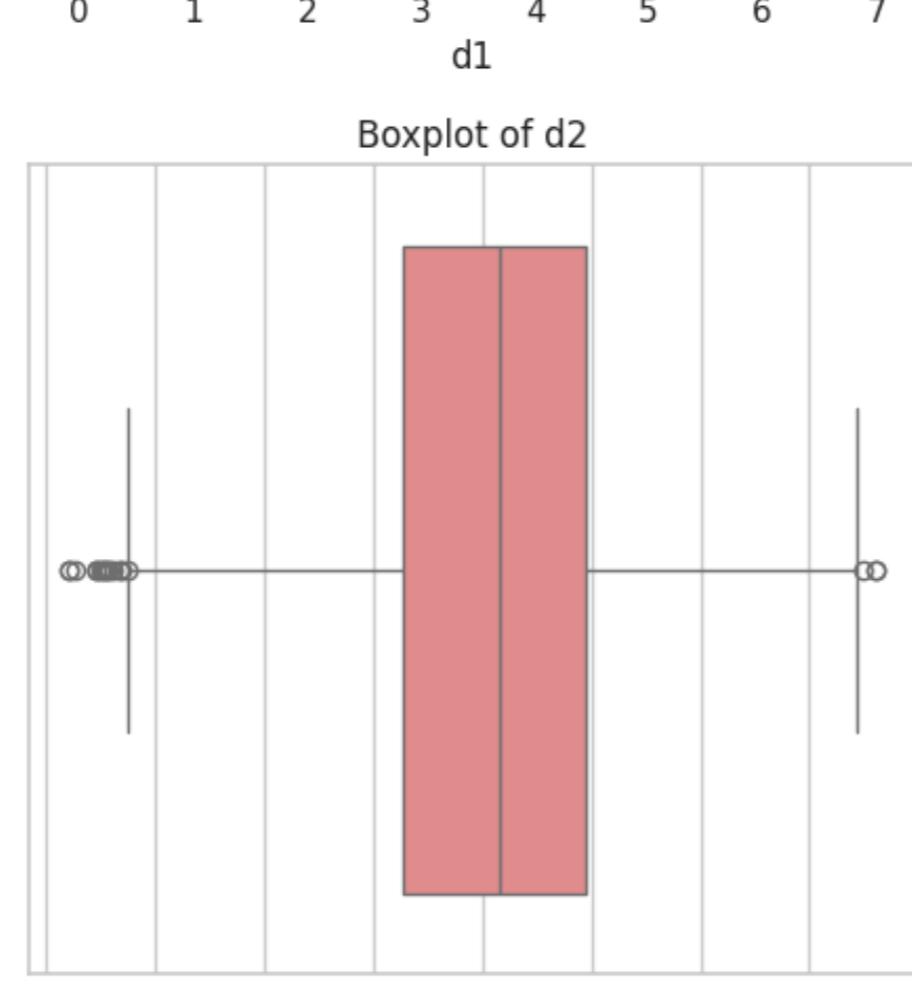
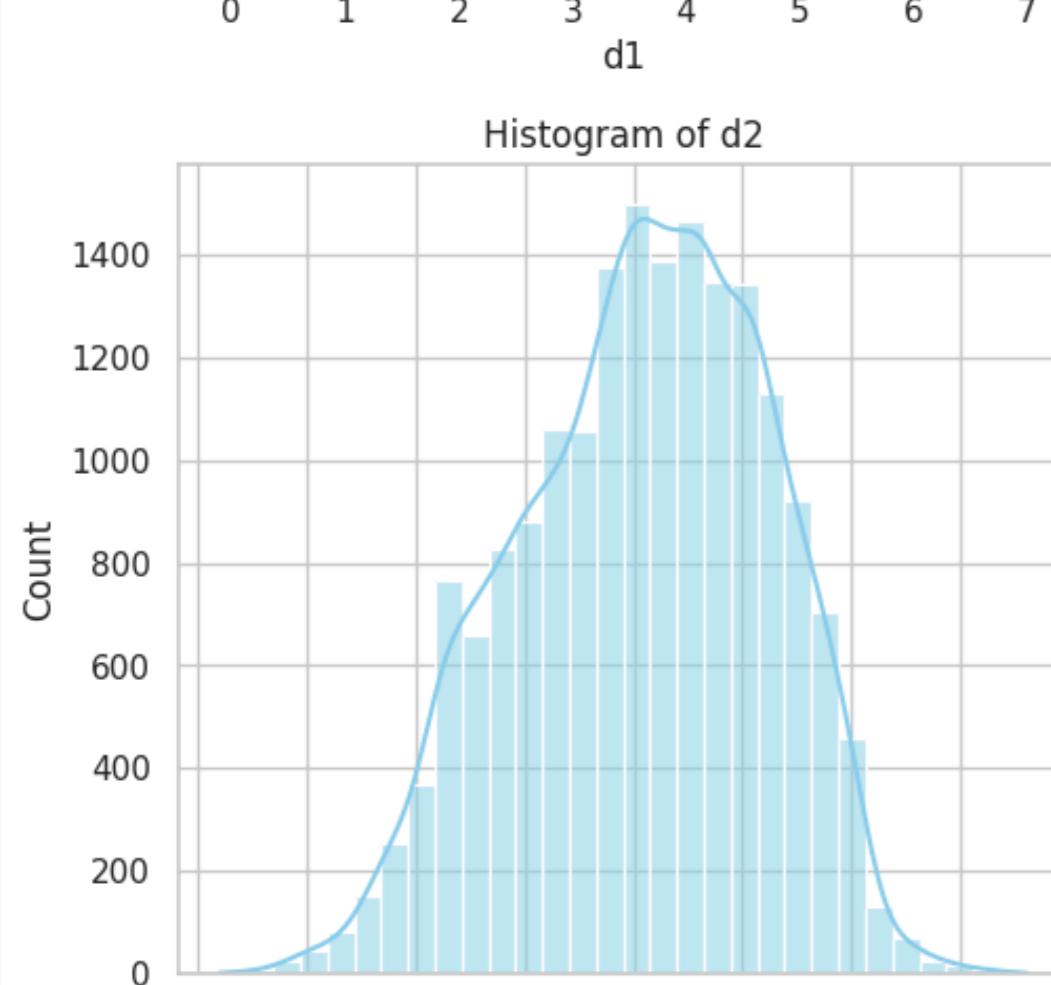
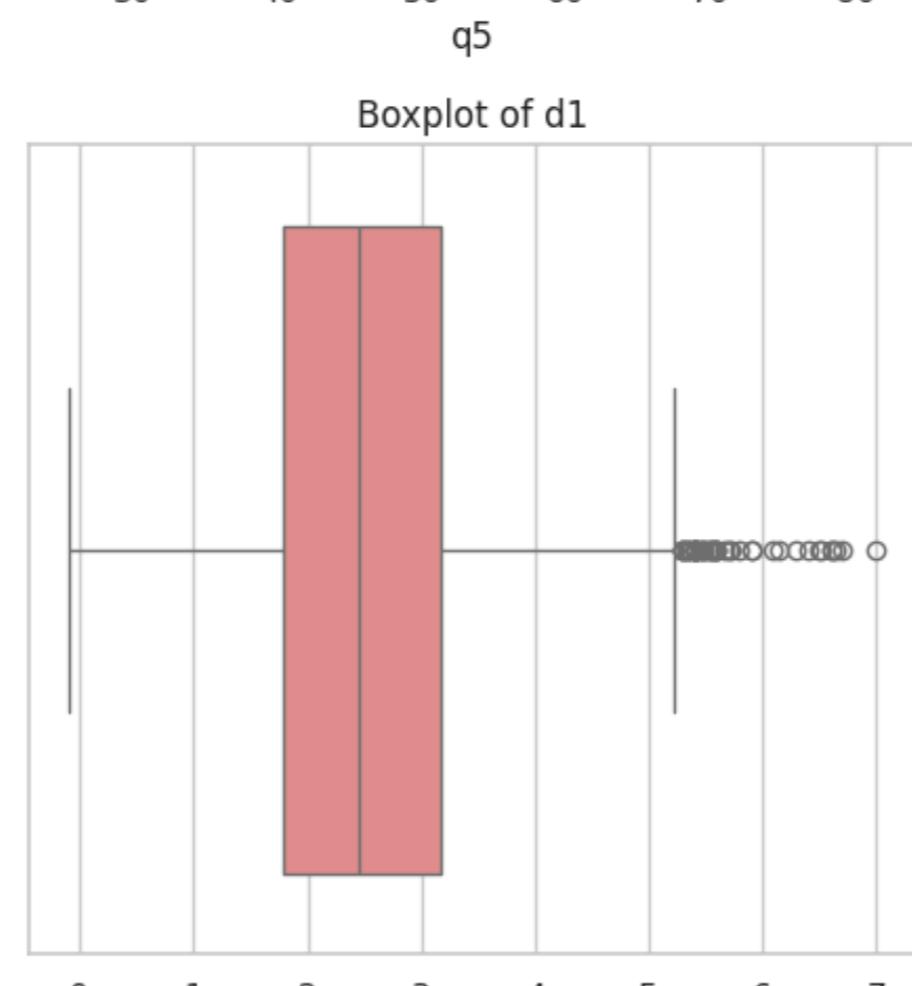
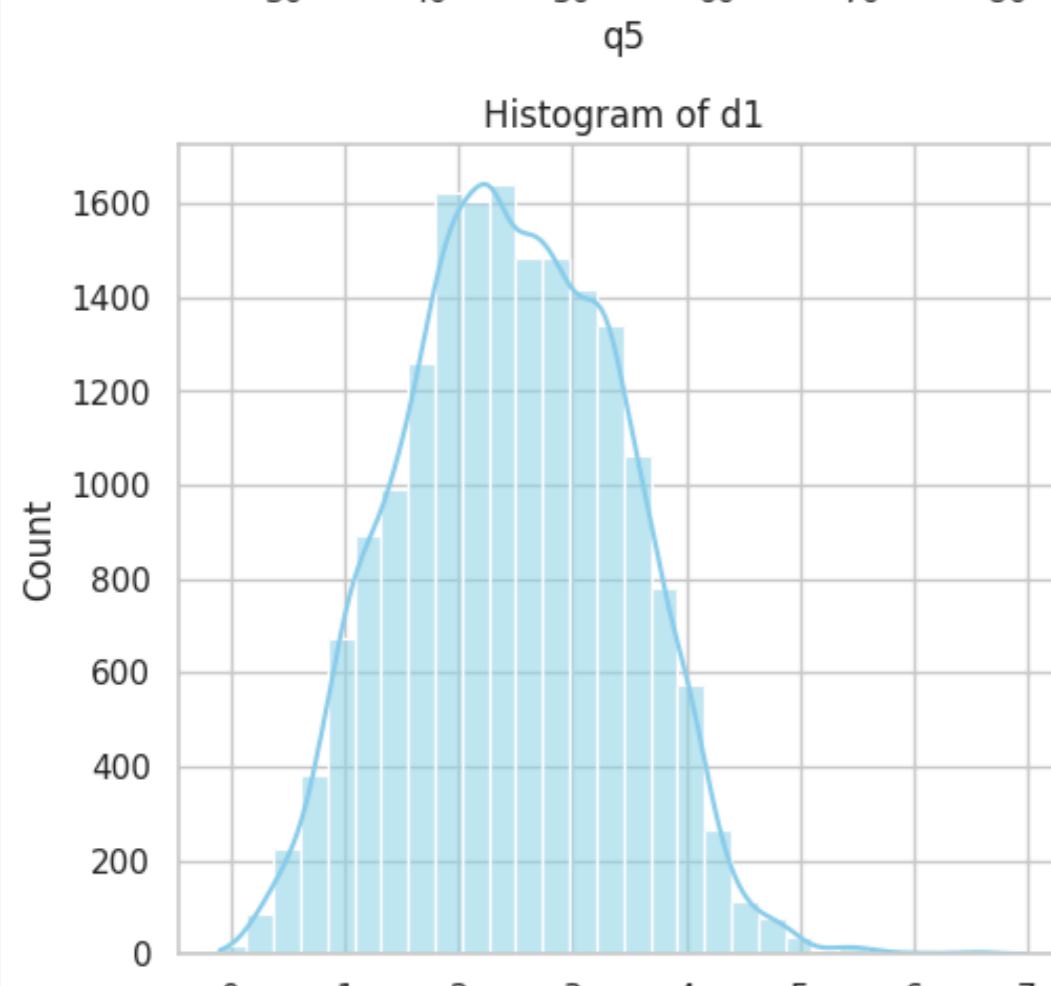
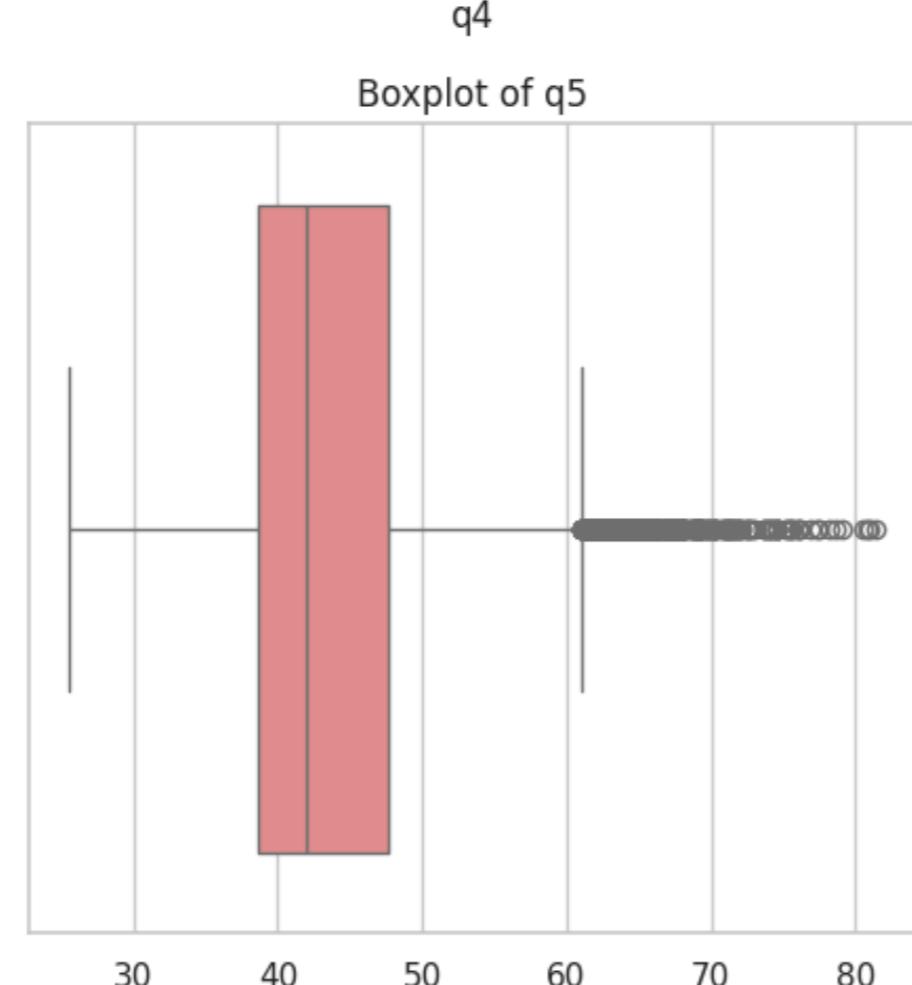
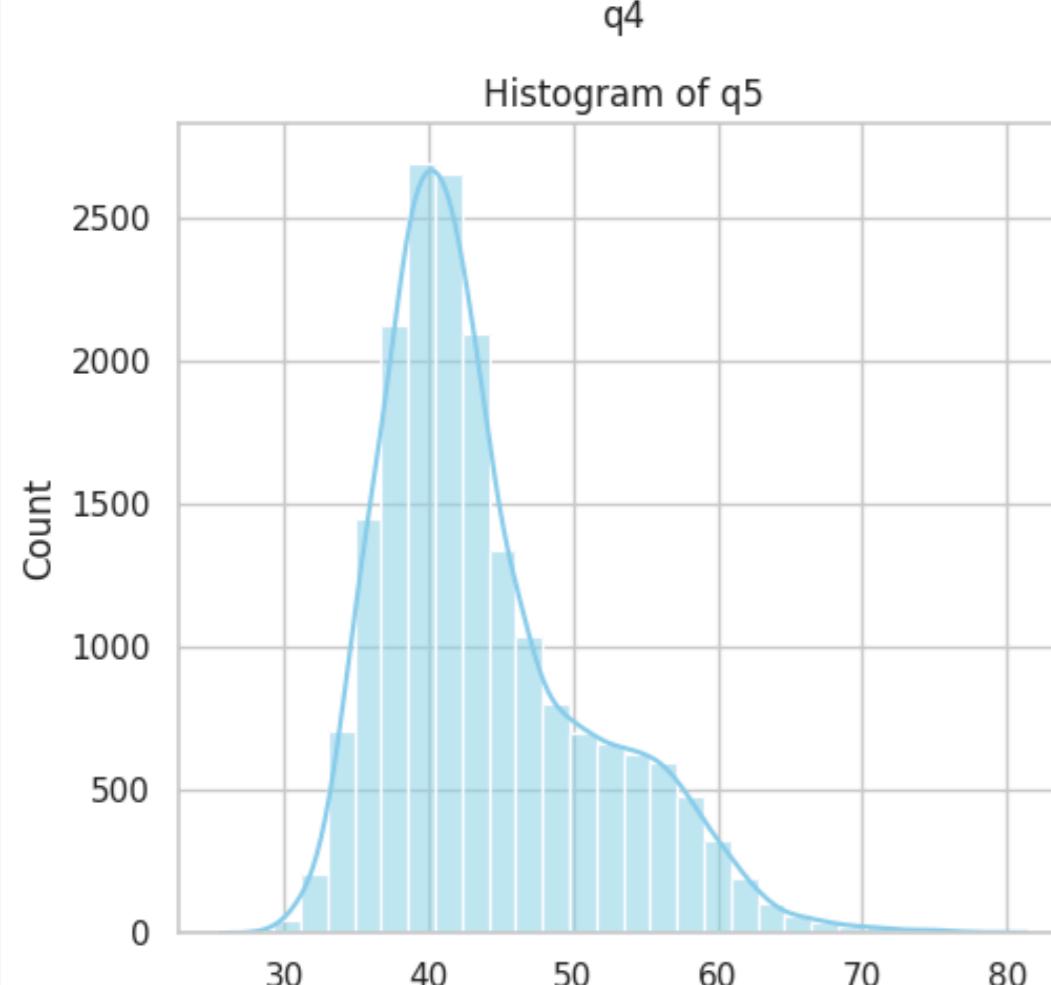
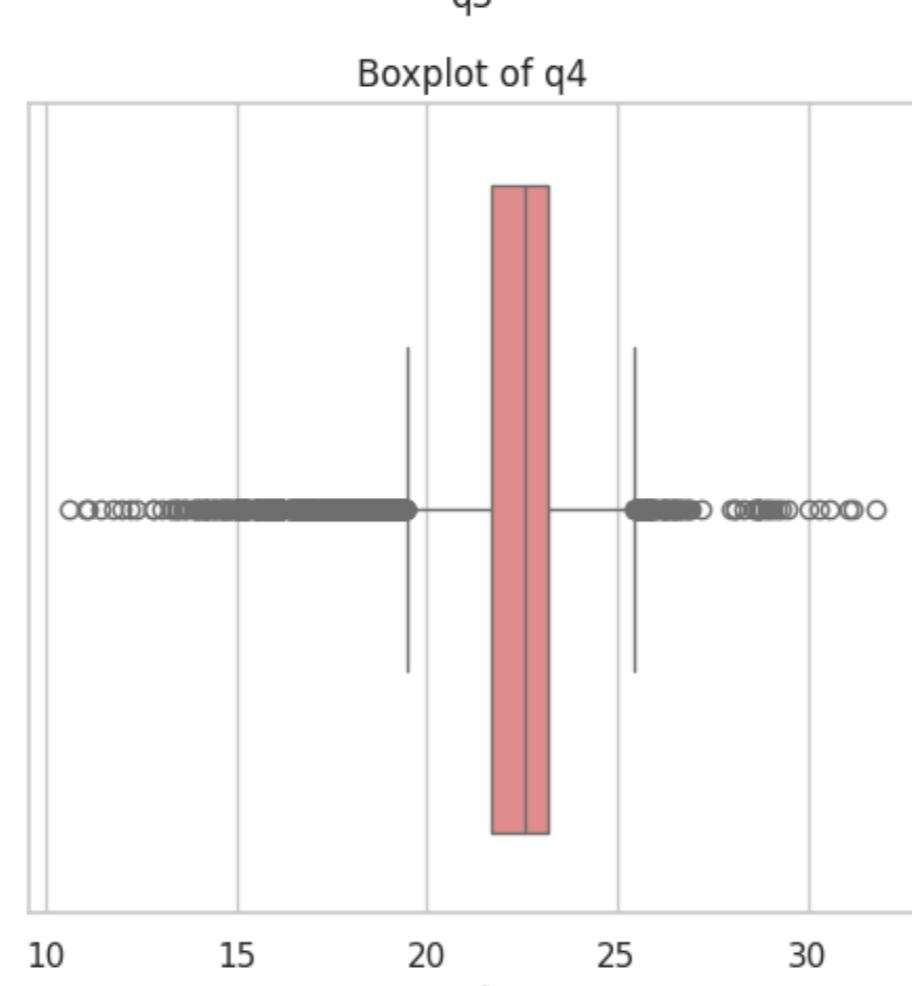
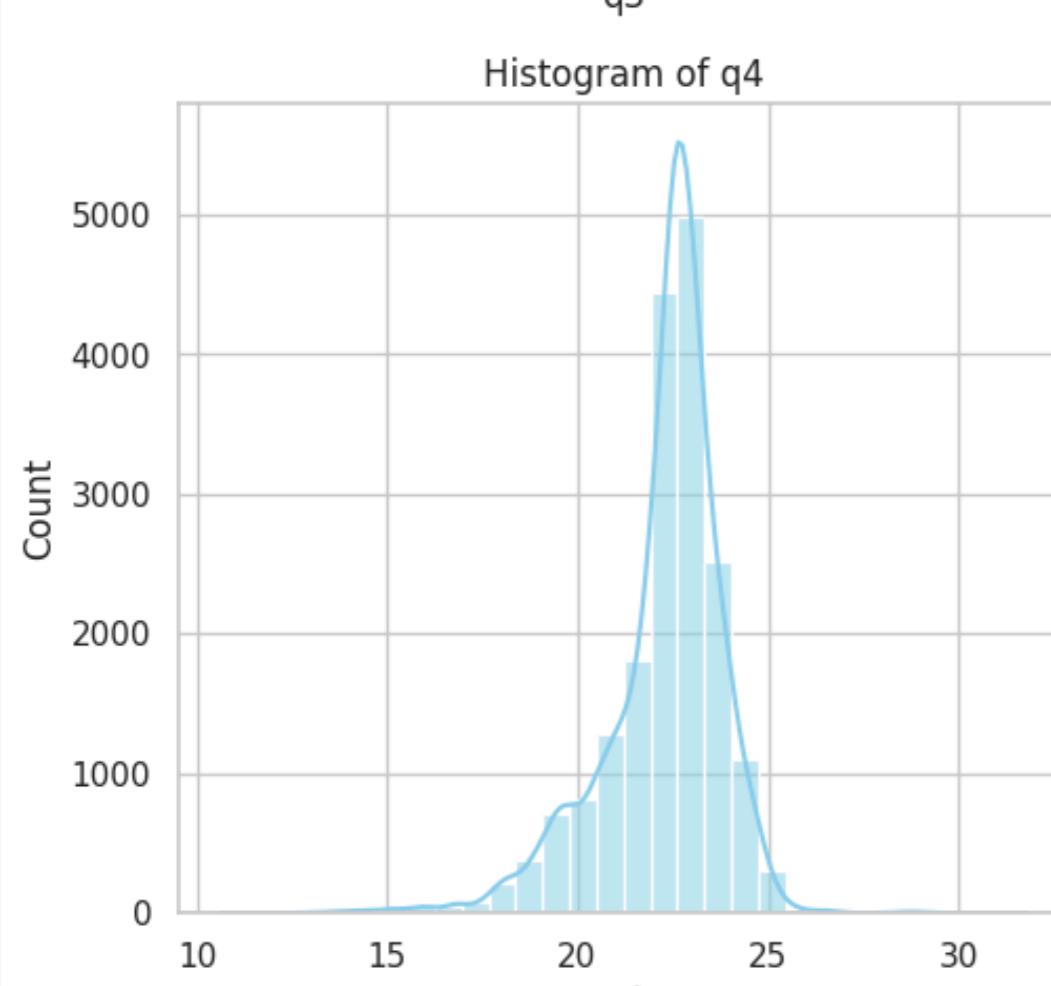
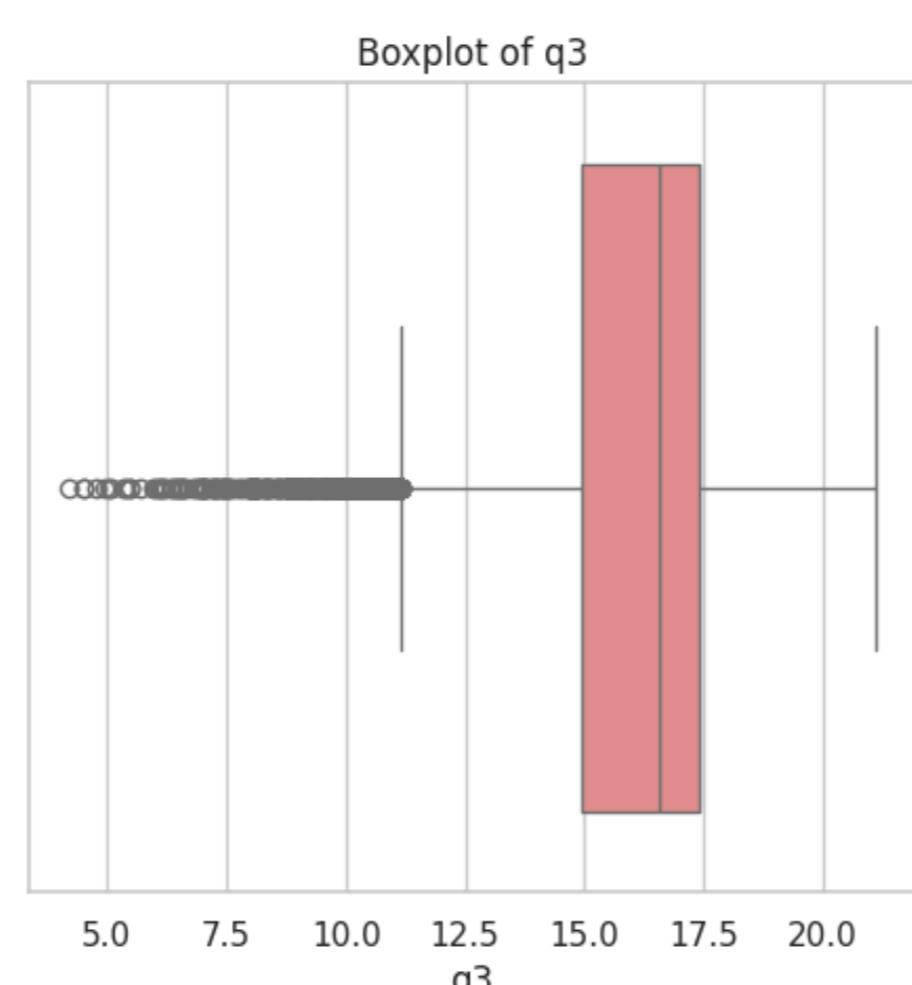
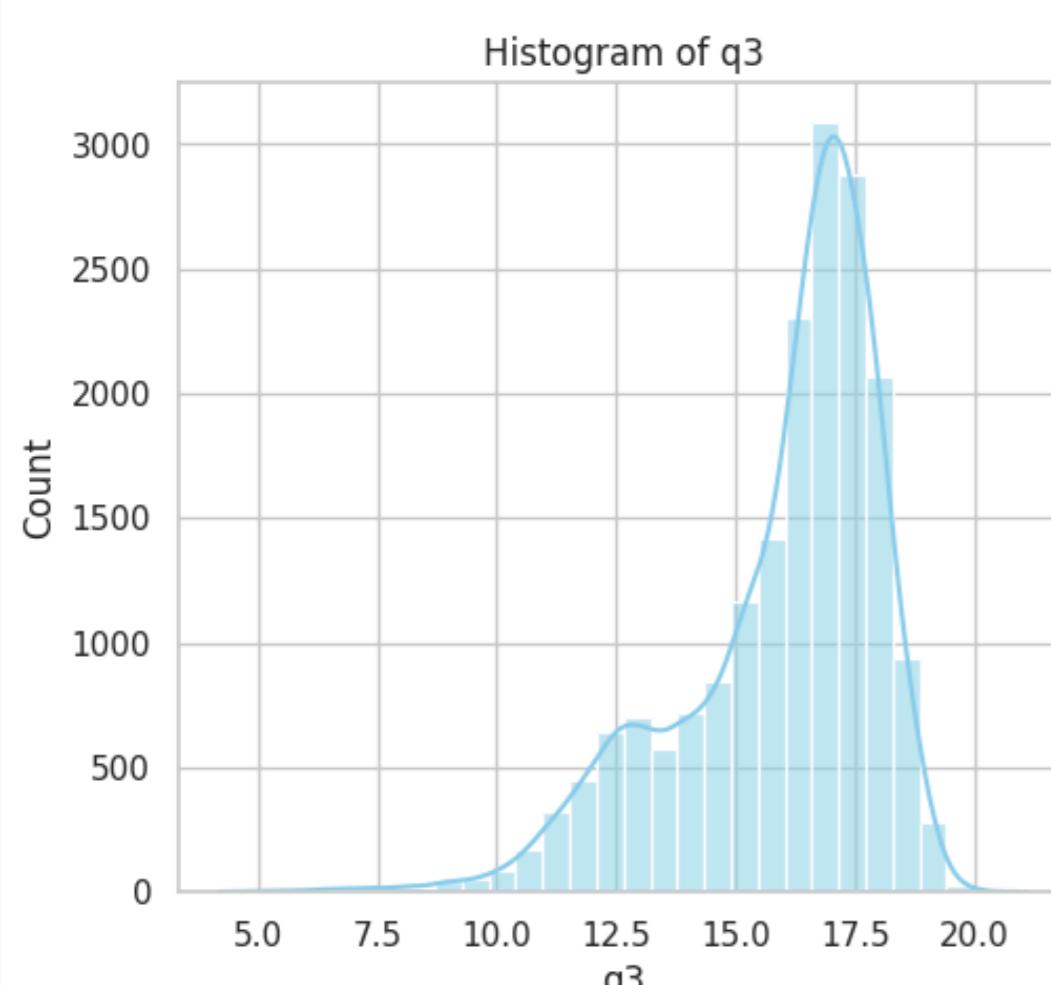
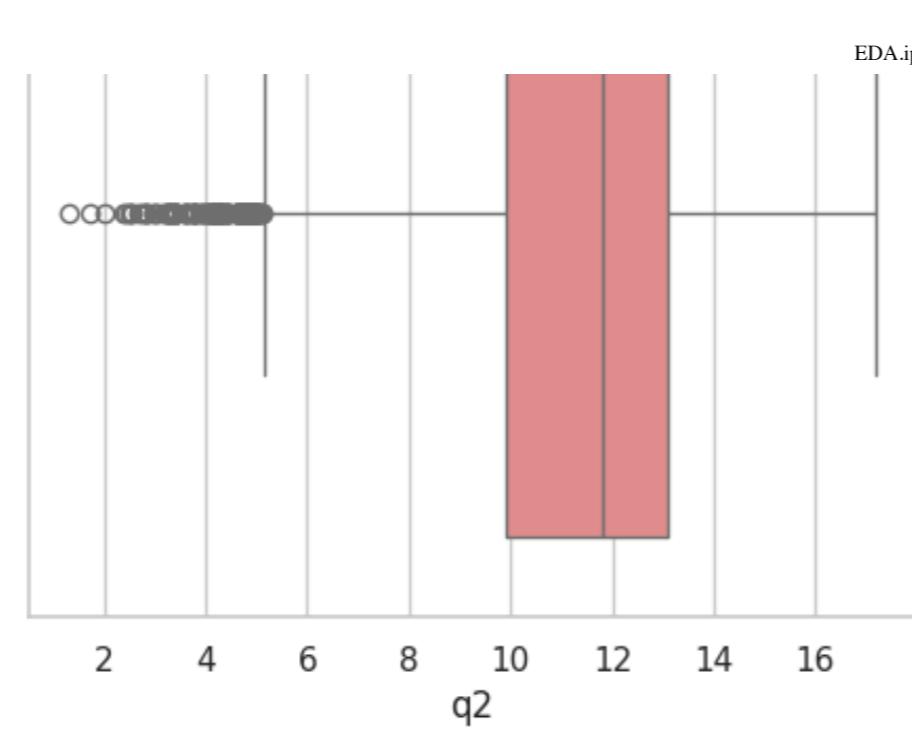
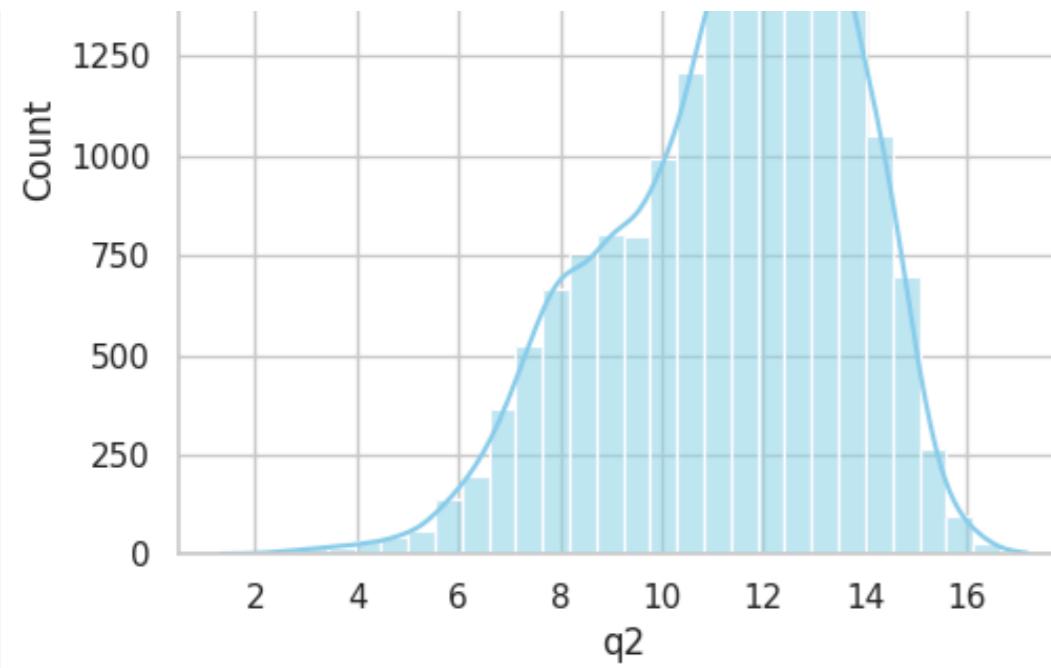
plt.show()

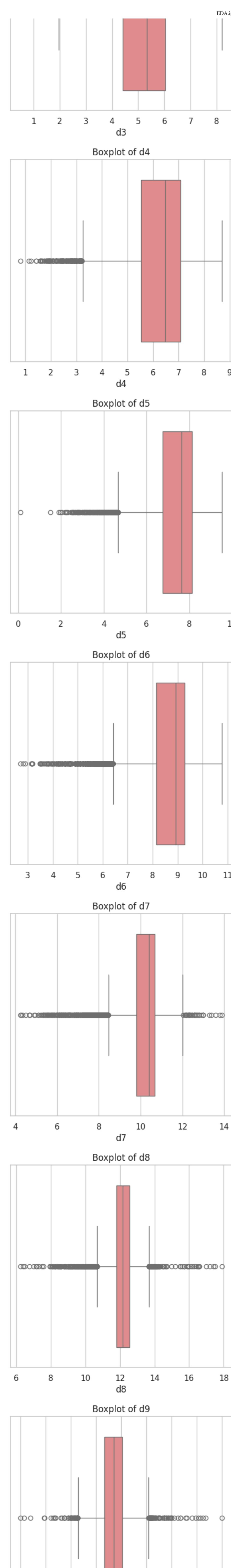
# Univariate Analysis for Categorical Columns
for col in cat_cols:
    plt.figure(figsize=(10, 5))
    sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
    plt.title(f"Bar Chart of {col}")
    plt.show()

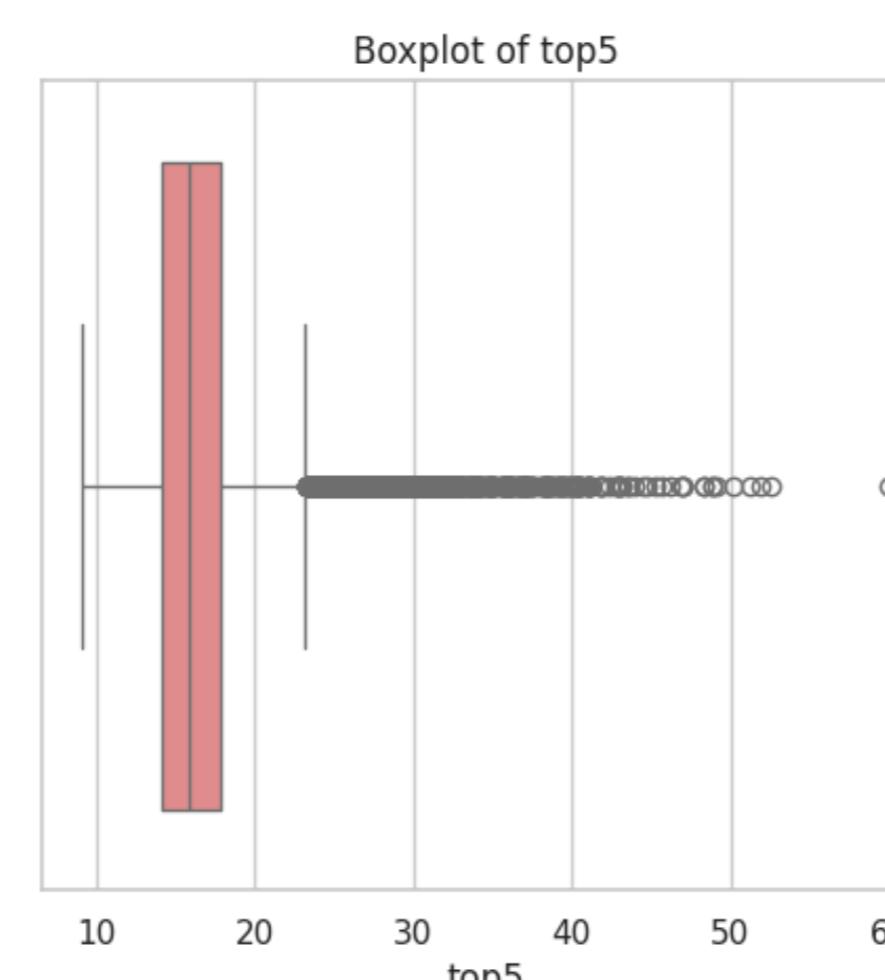
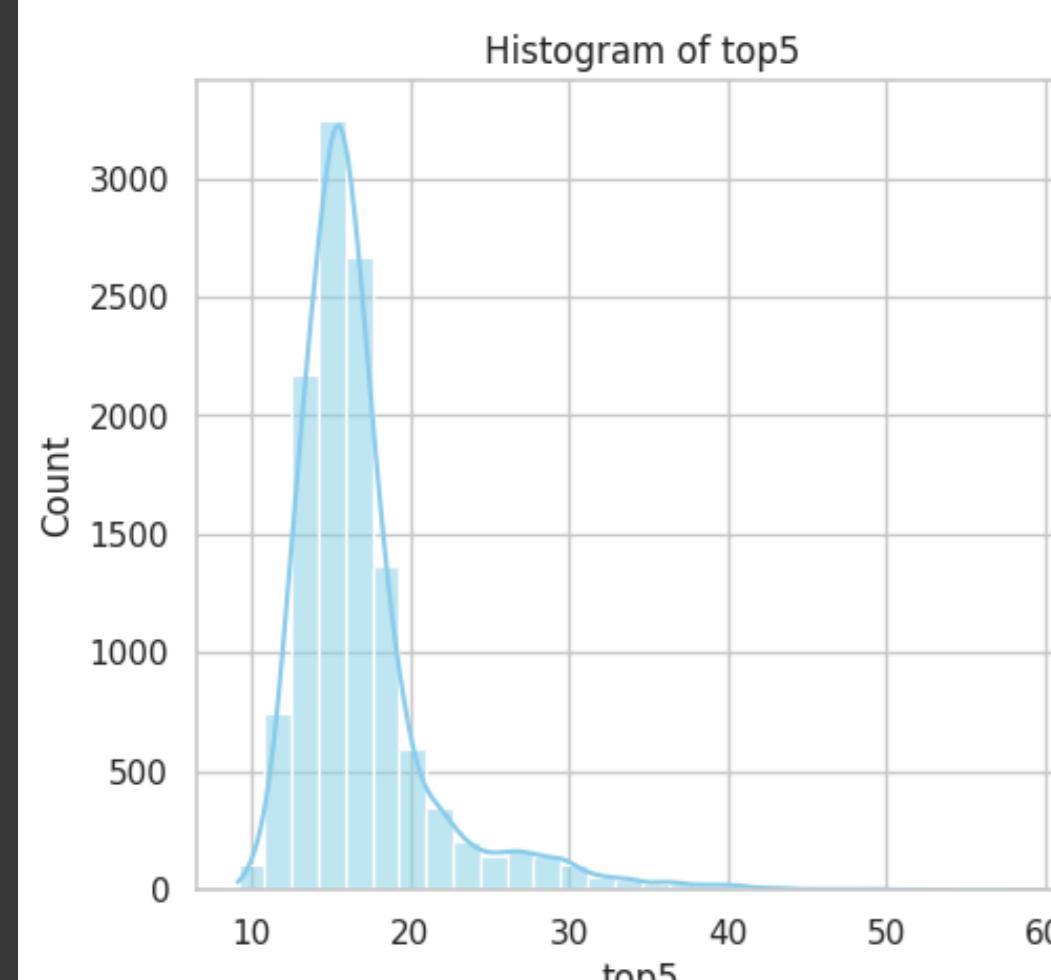
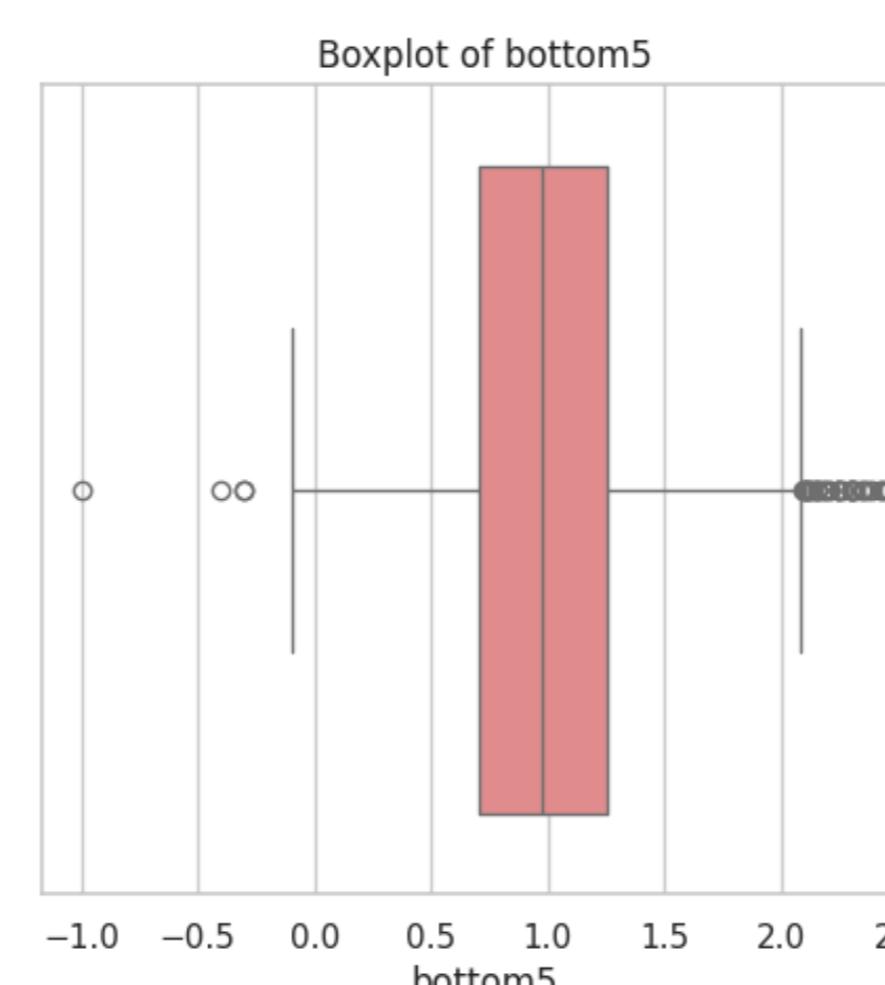
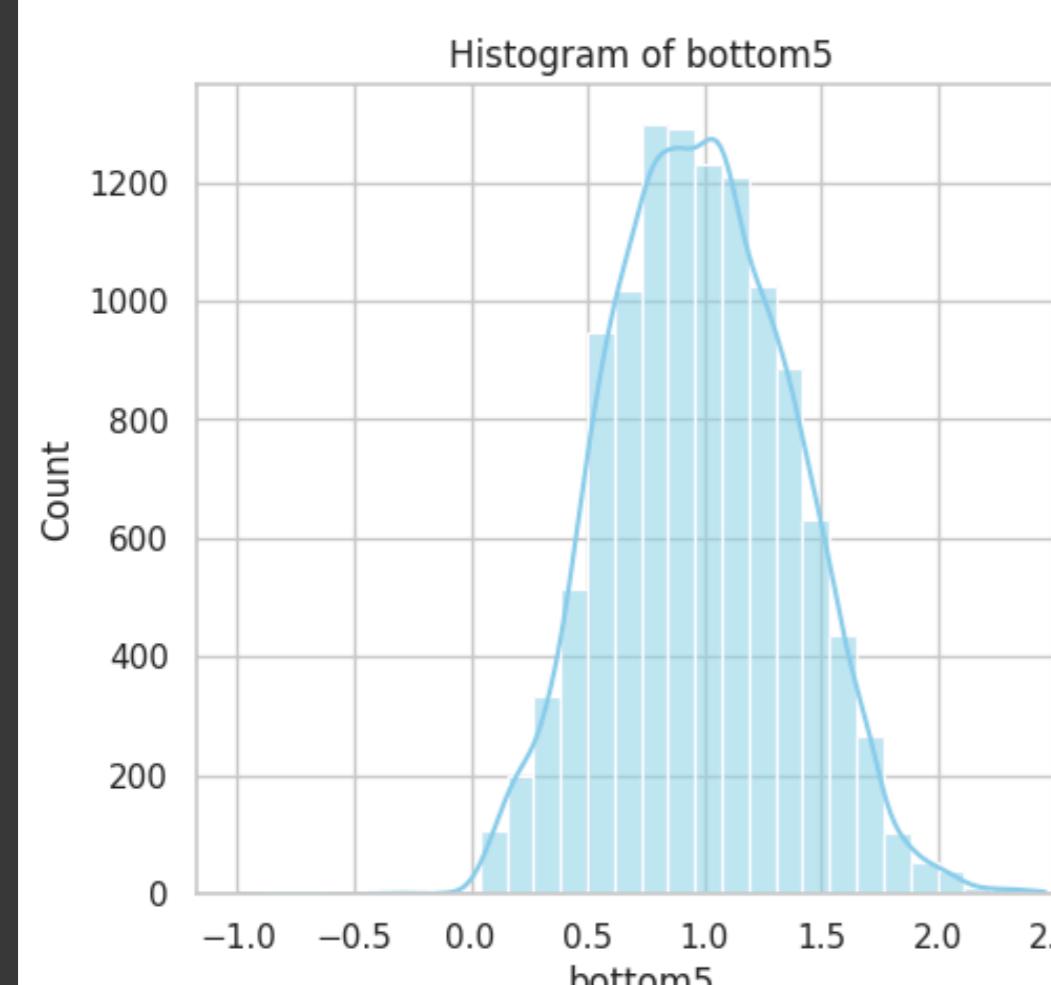
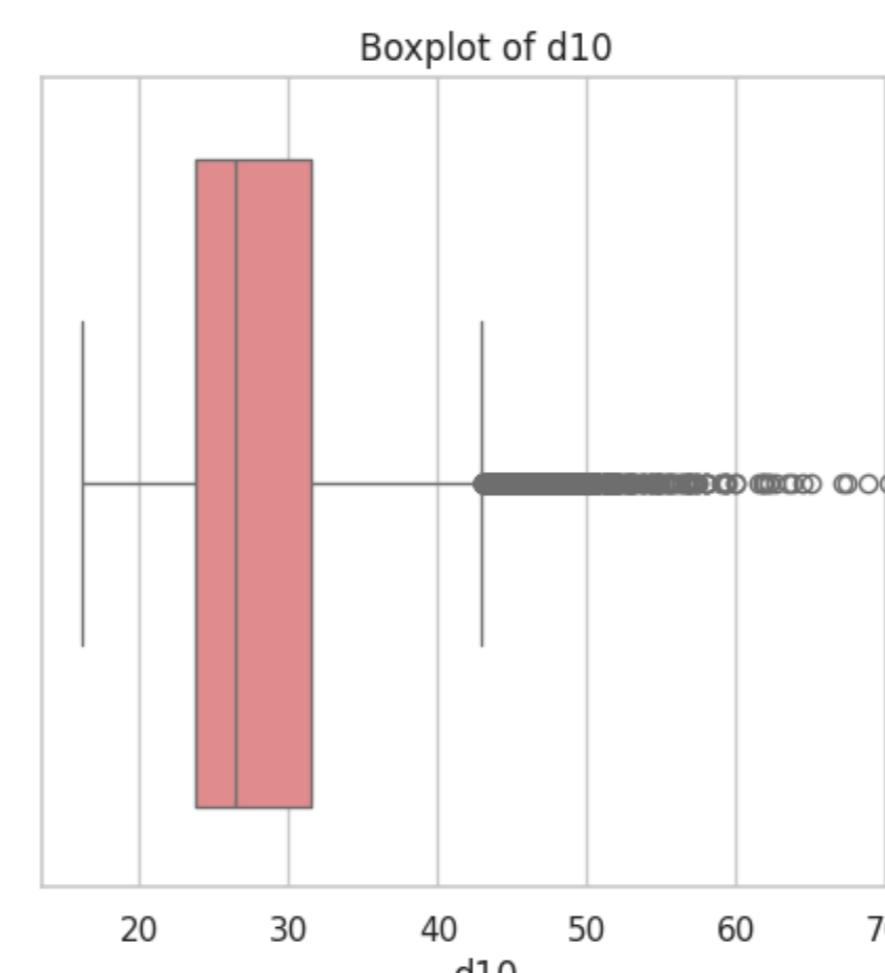
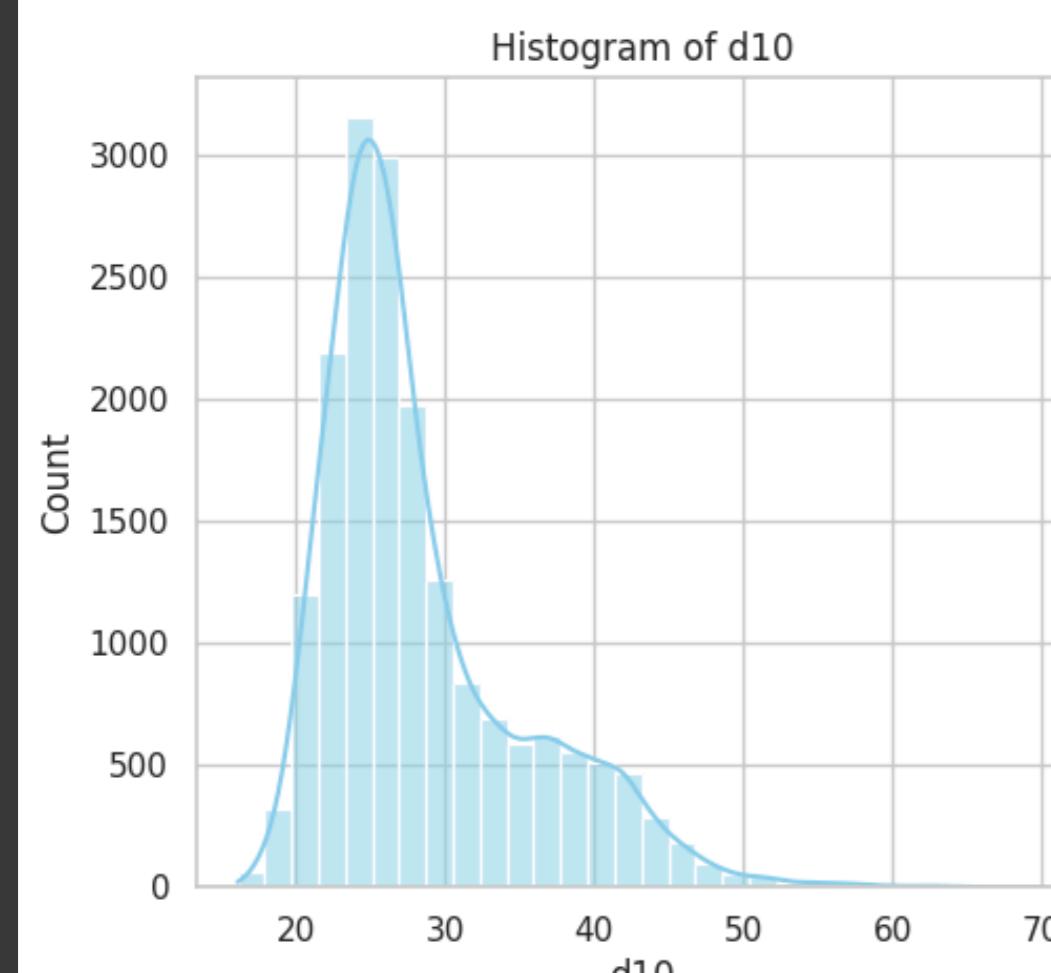
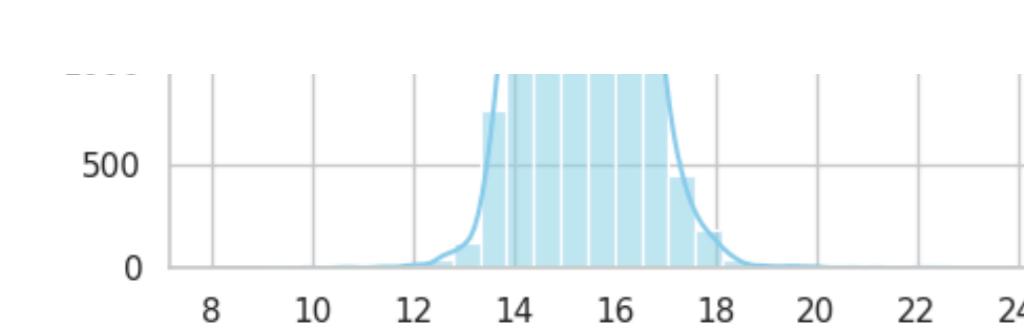
```



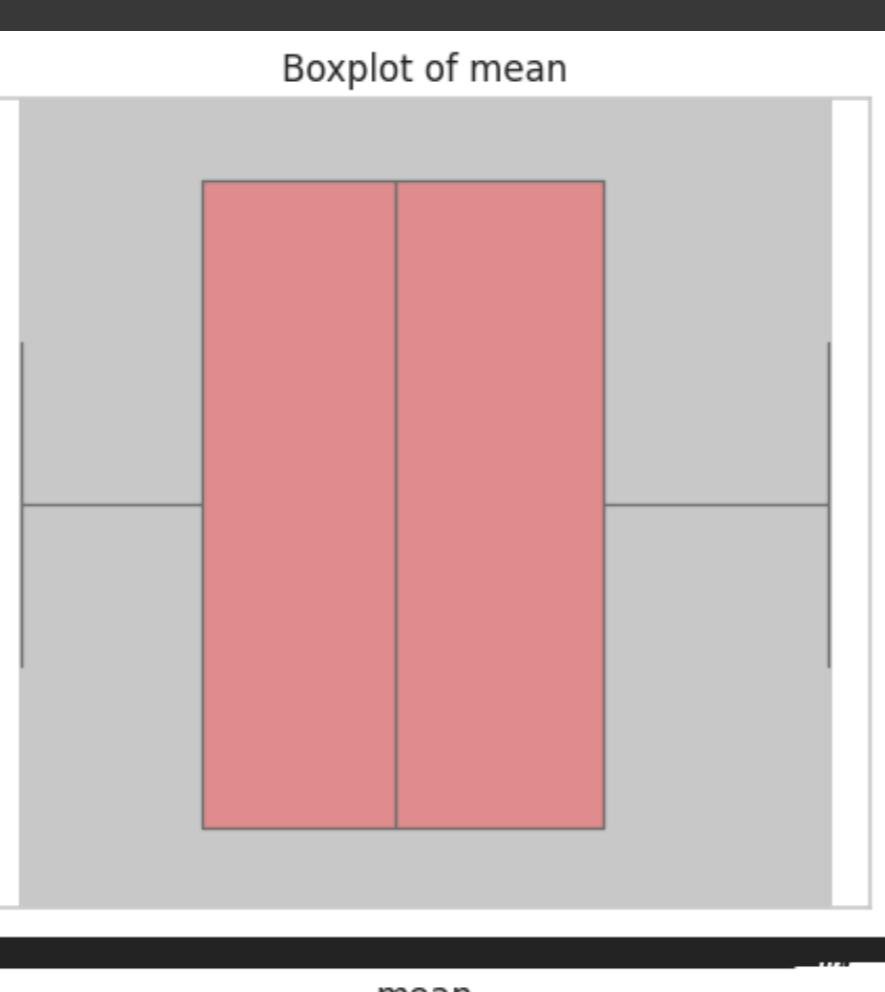
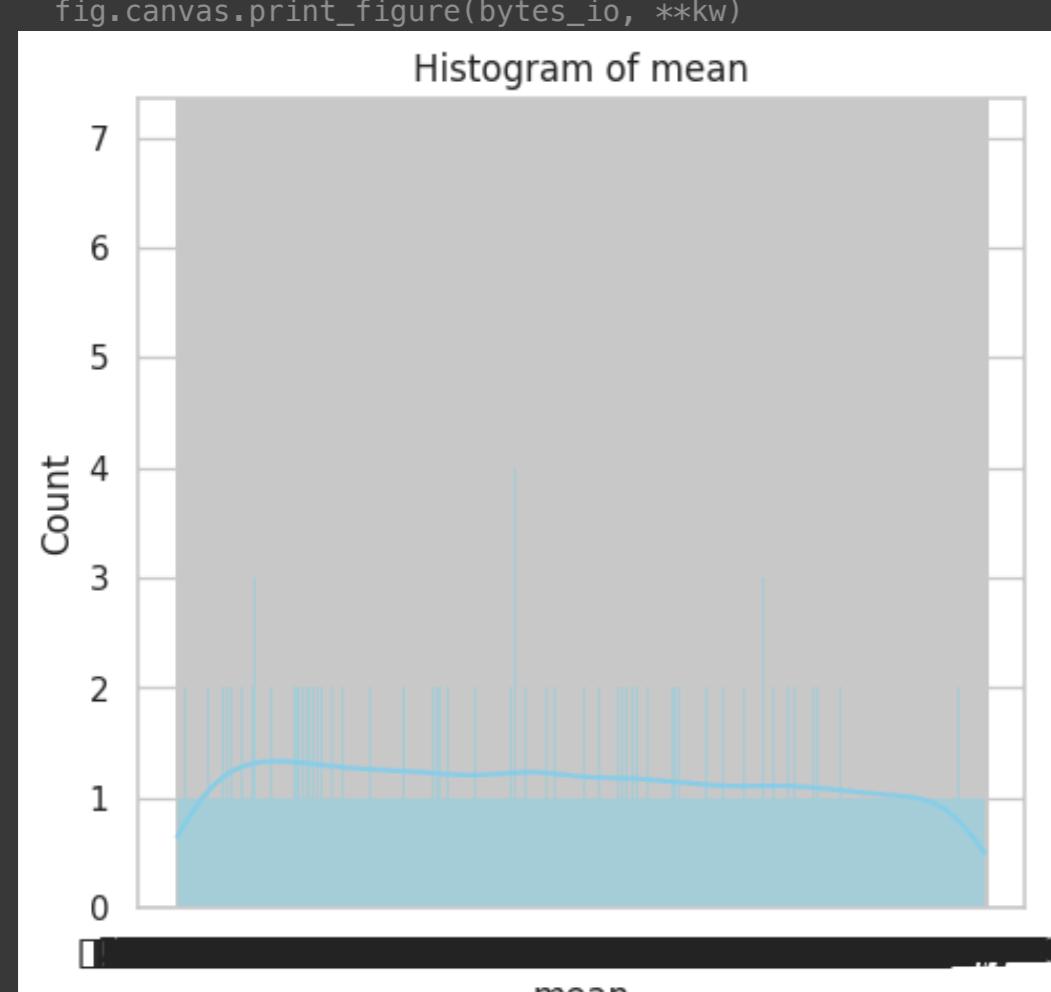




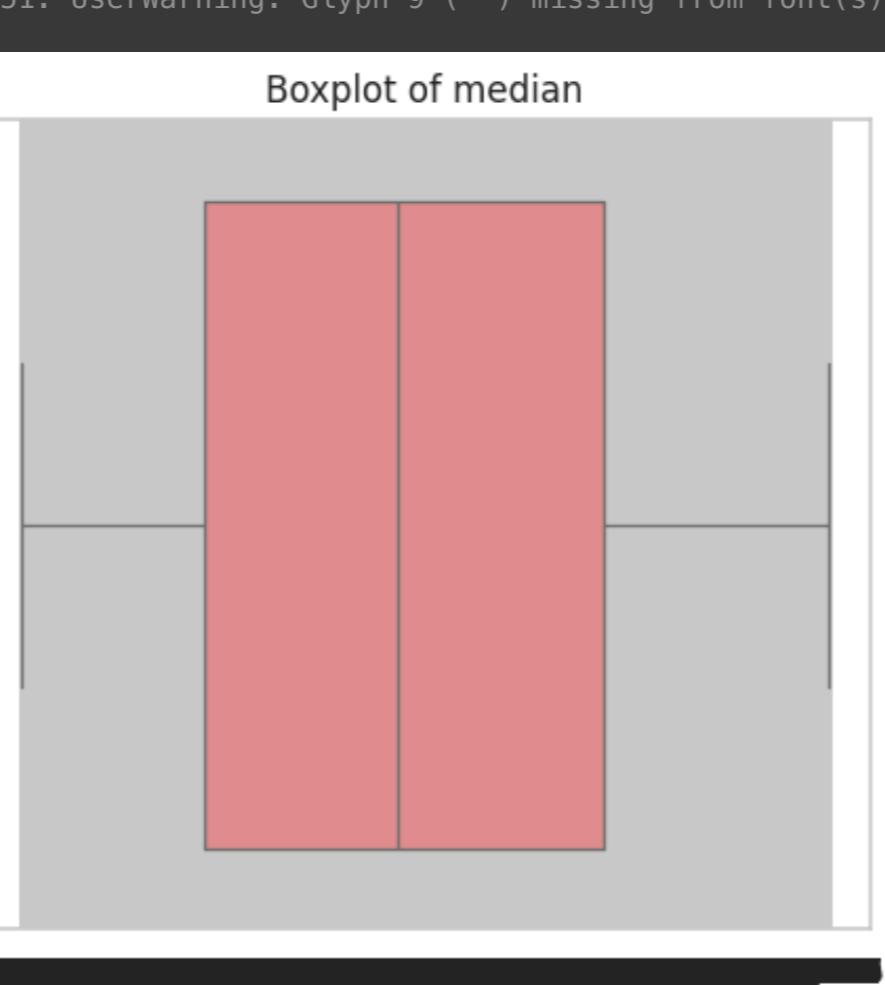
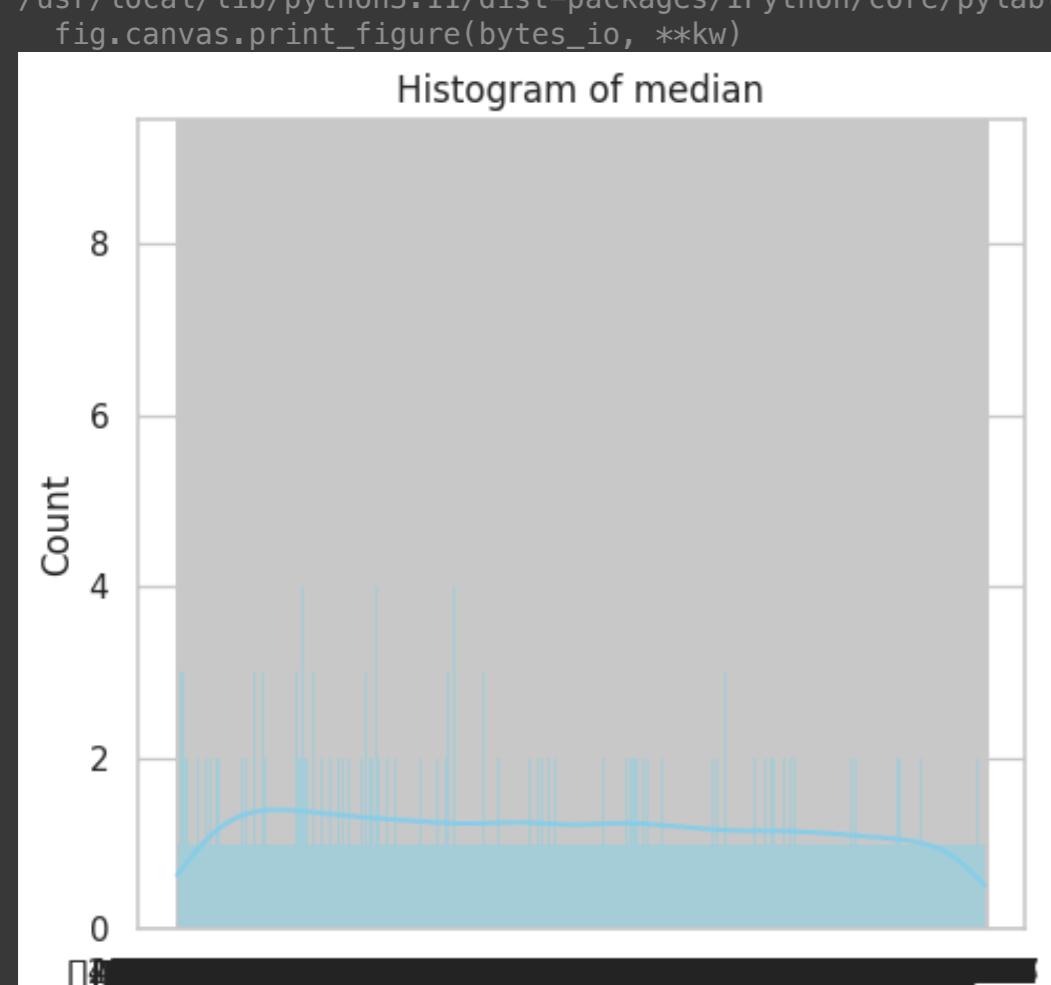




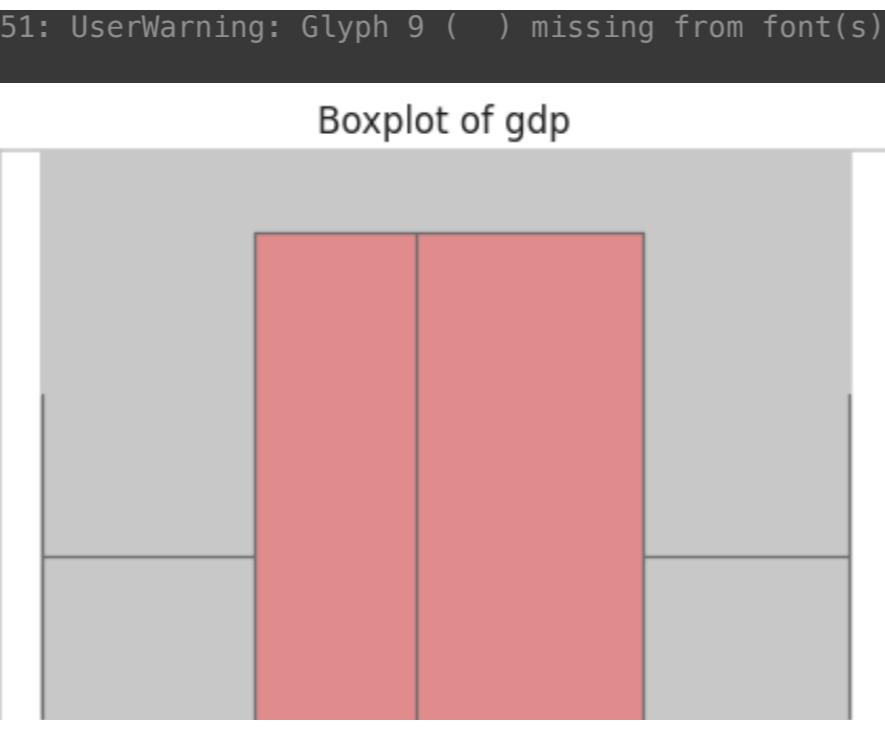
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 () missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)

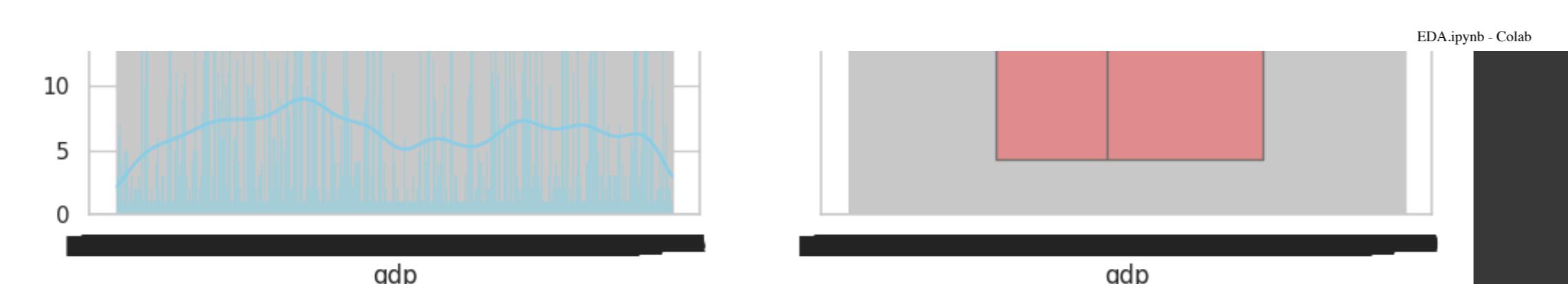


/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 () missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)

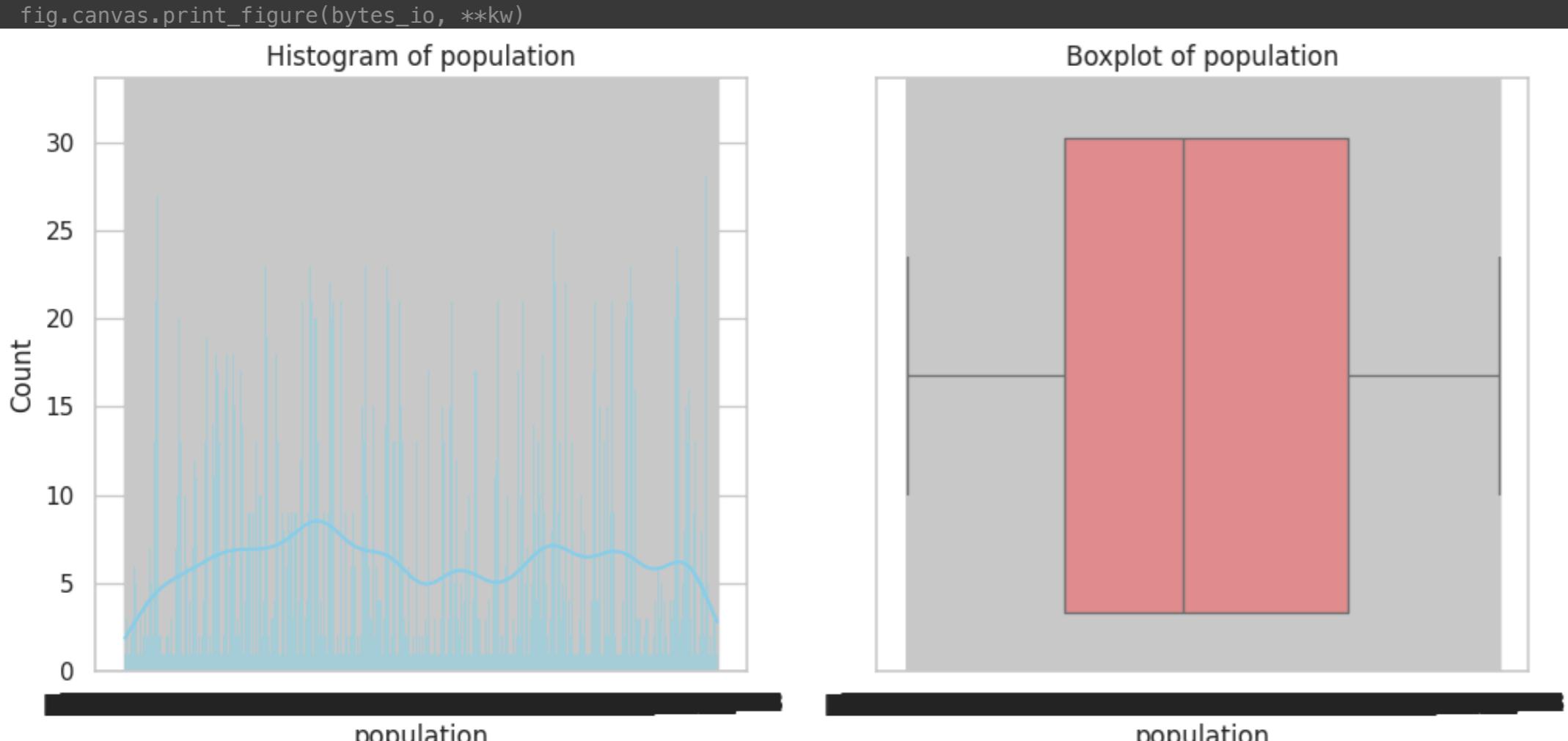


/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 () missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)

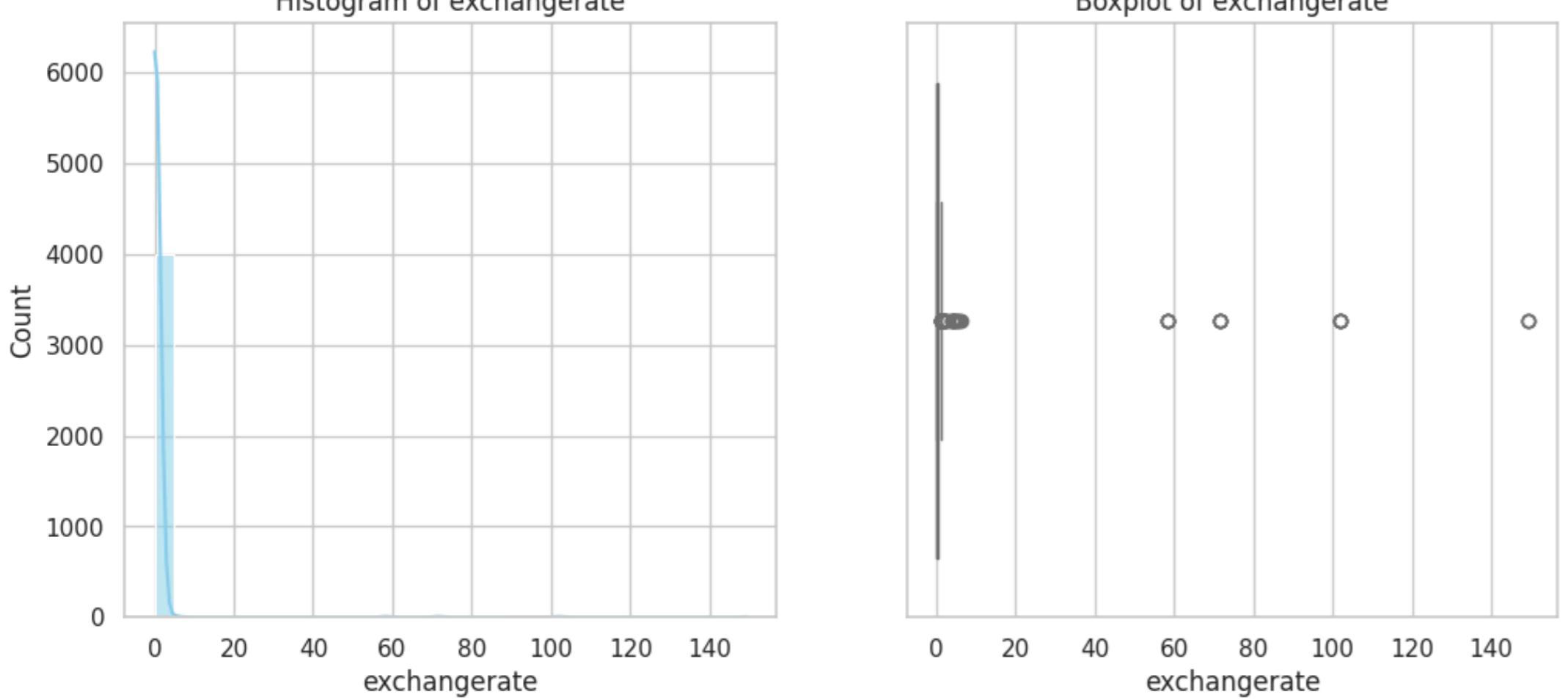




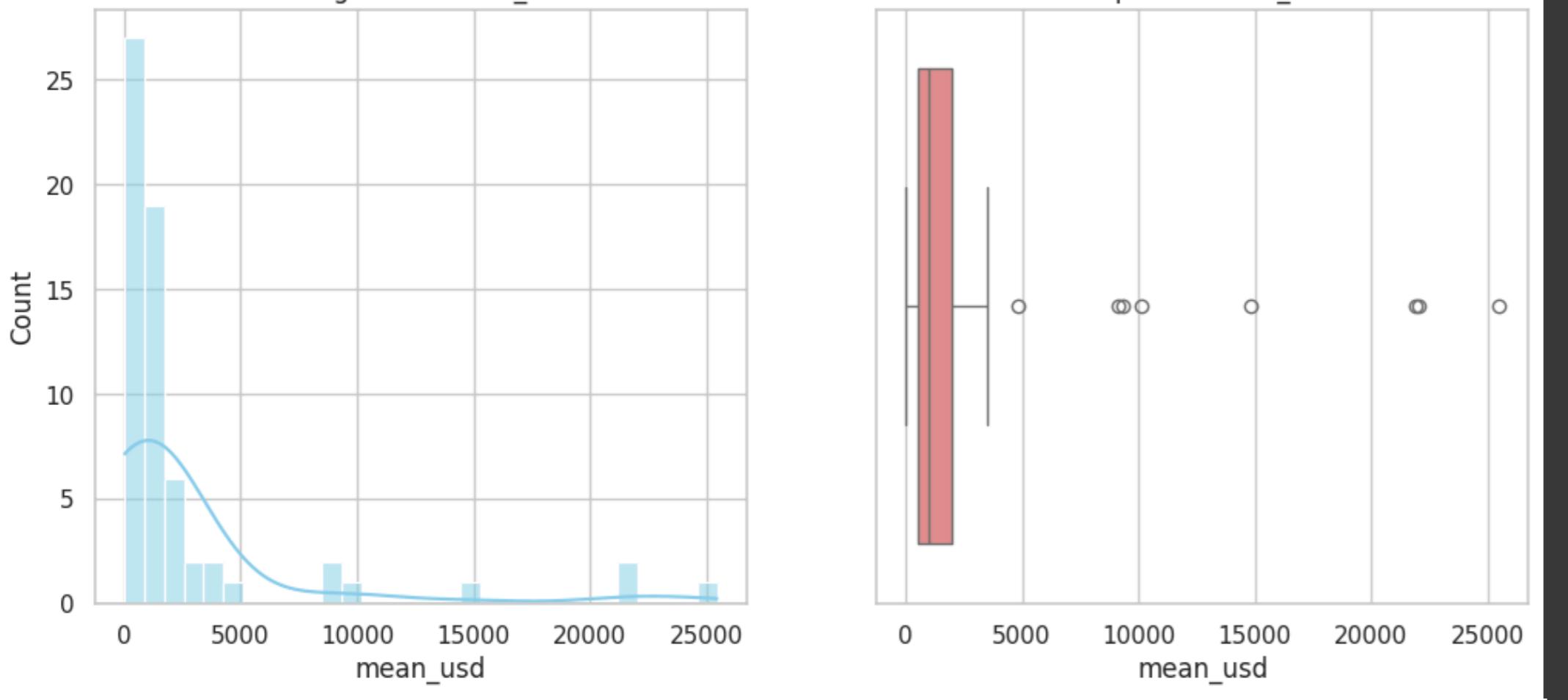
```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 ( ) missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)
```



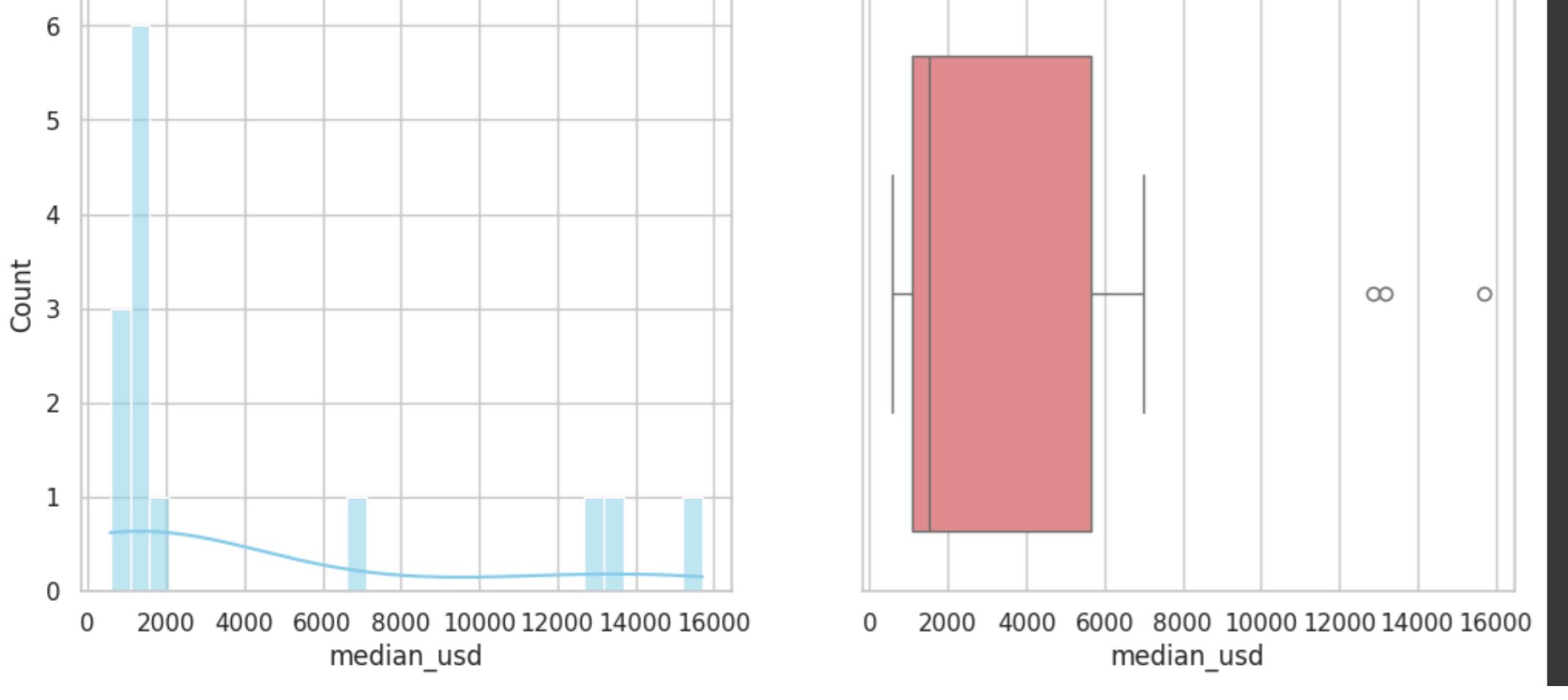
```
population
```



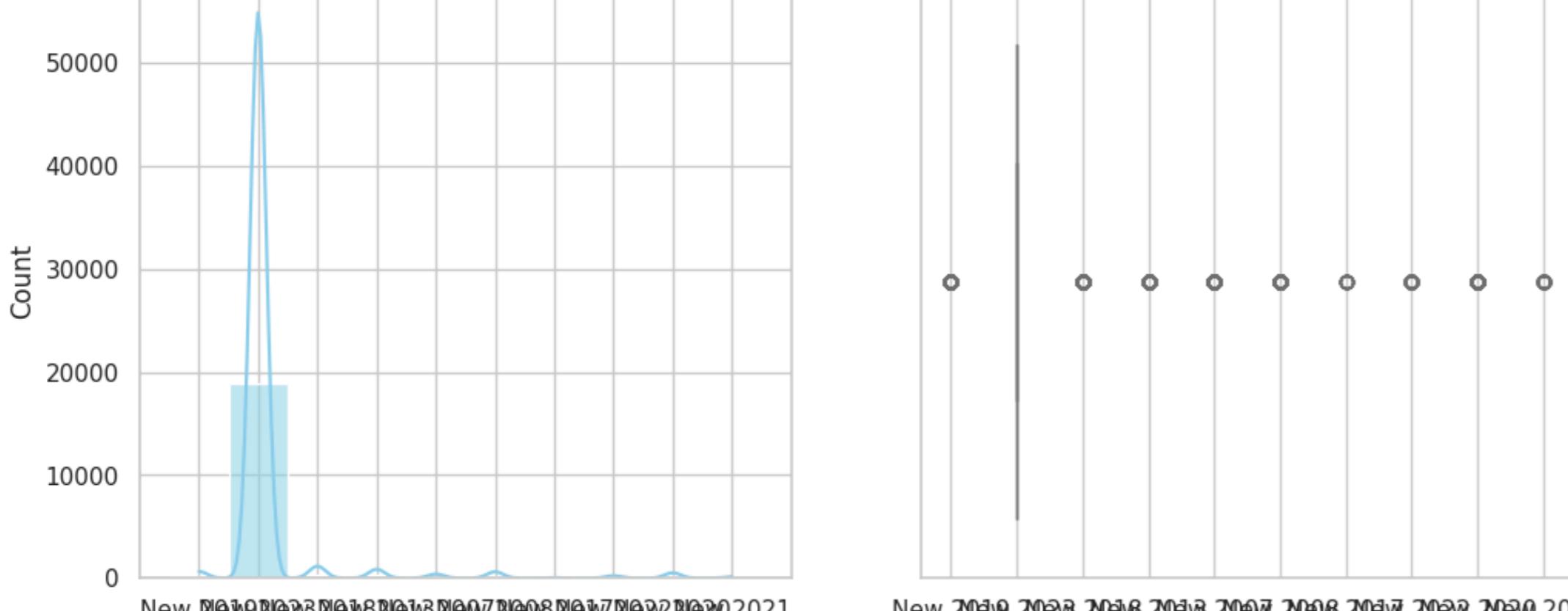
```
exchangerate
```



```
mean_usd
```



```
median_usd
```

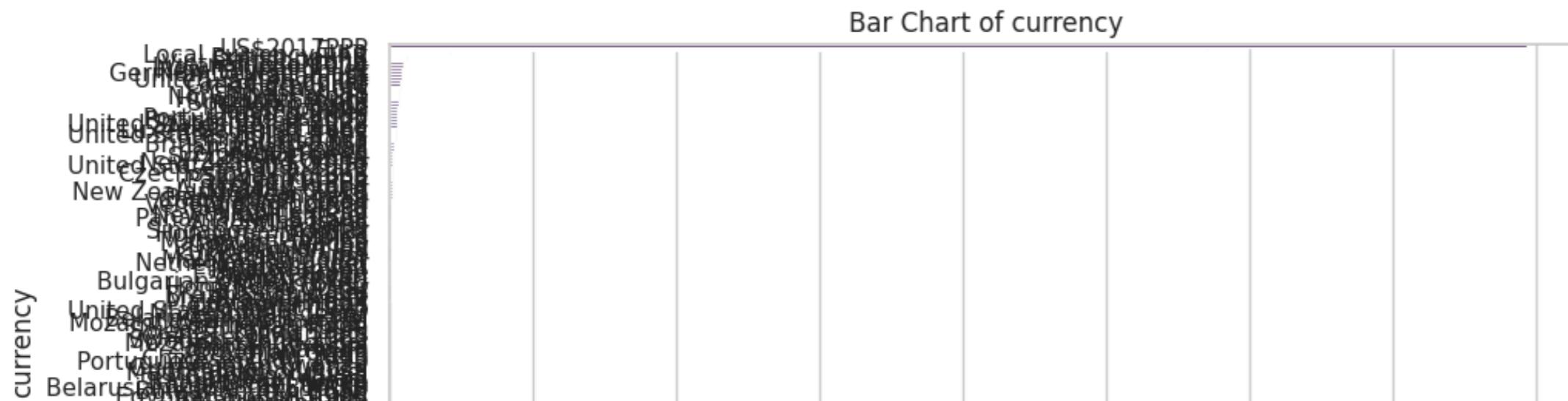


```
revision
```

```
<ipython-input-22-ceb3872374c7>:21: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```

```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



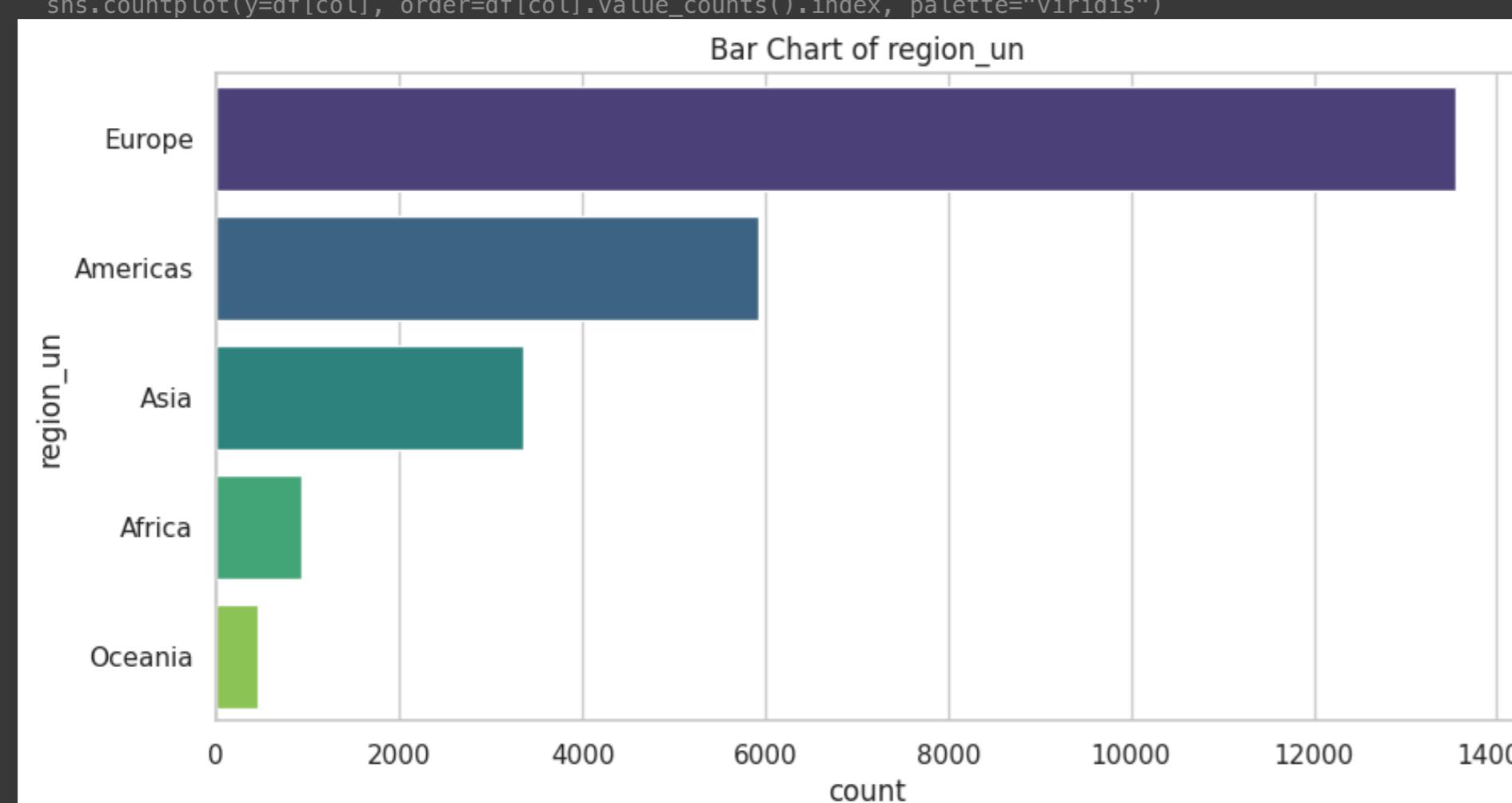
```
https://colab.research.google.com/drive/1AvbAkYPacJranh2Sr8whBt8MCzMsPnWB#scrollTo=jTb00VWYOsZ&printMode=true
```



```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

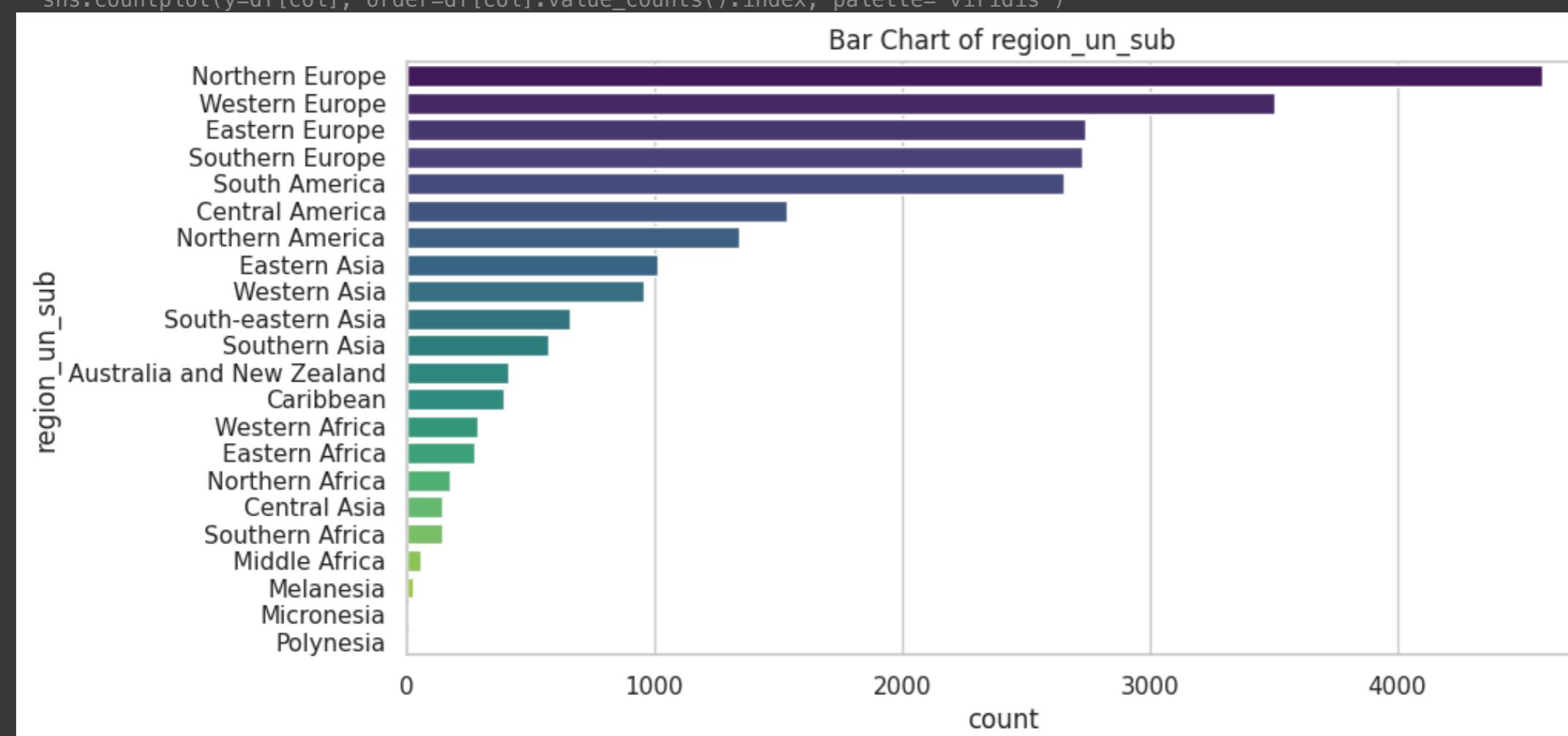
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

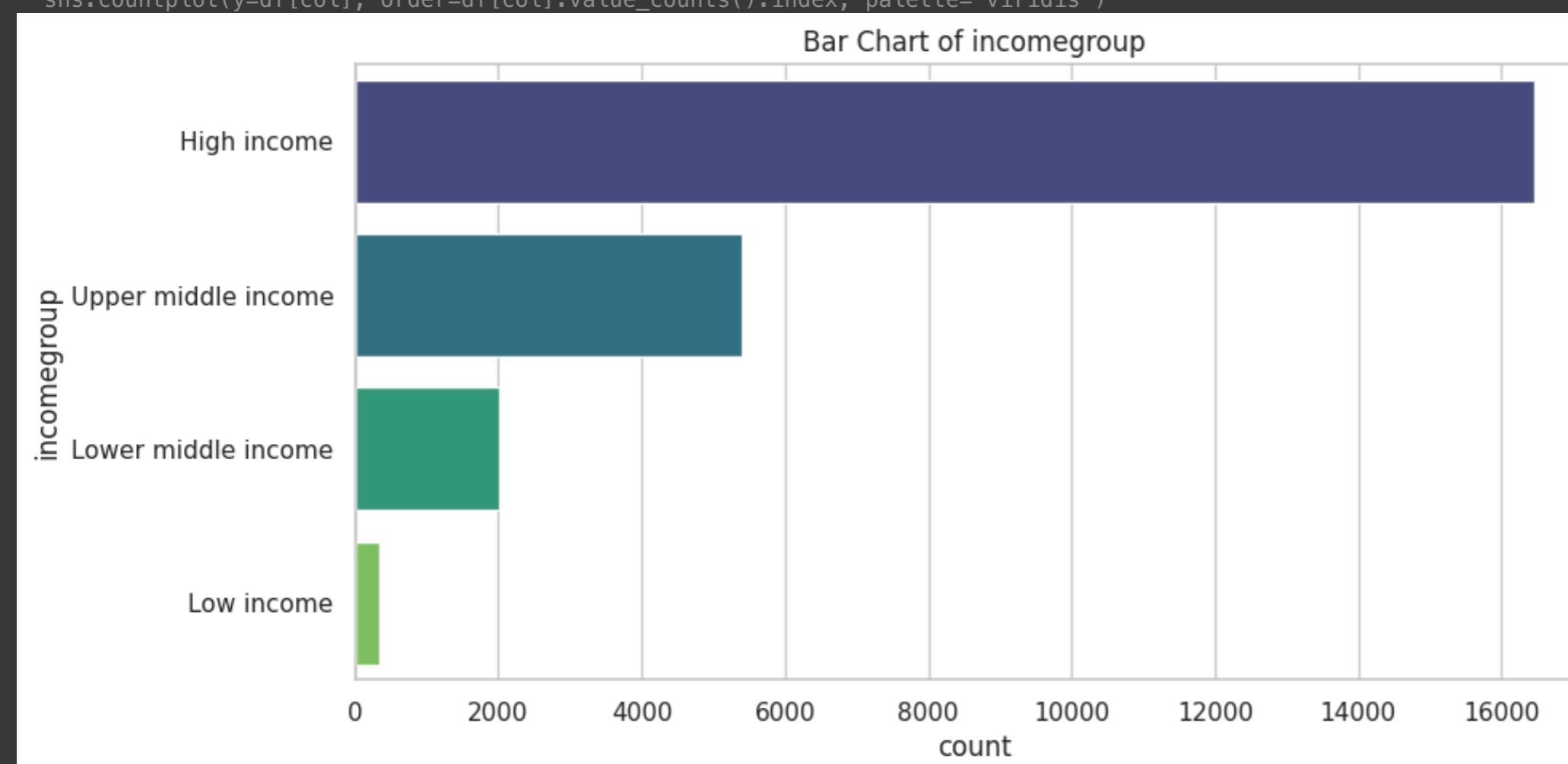
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

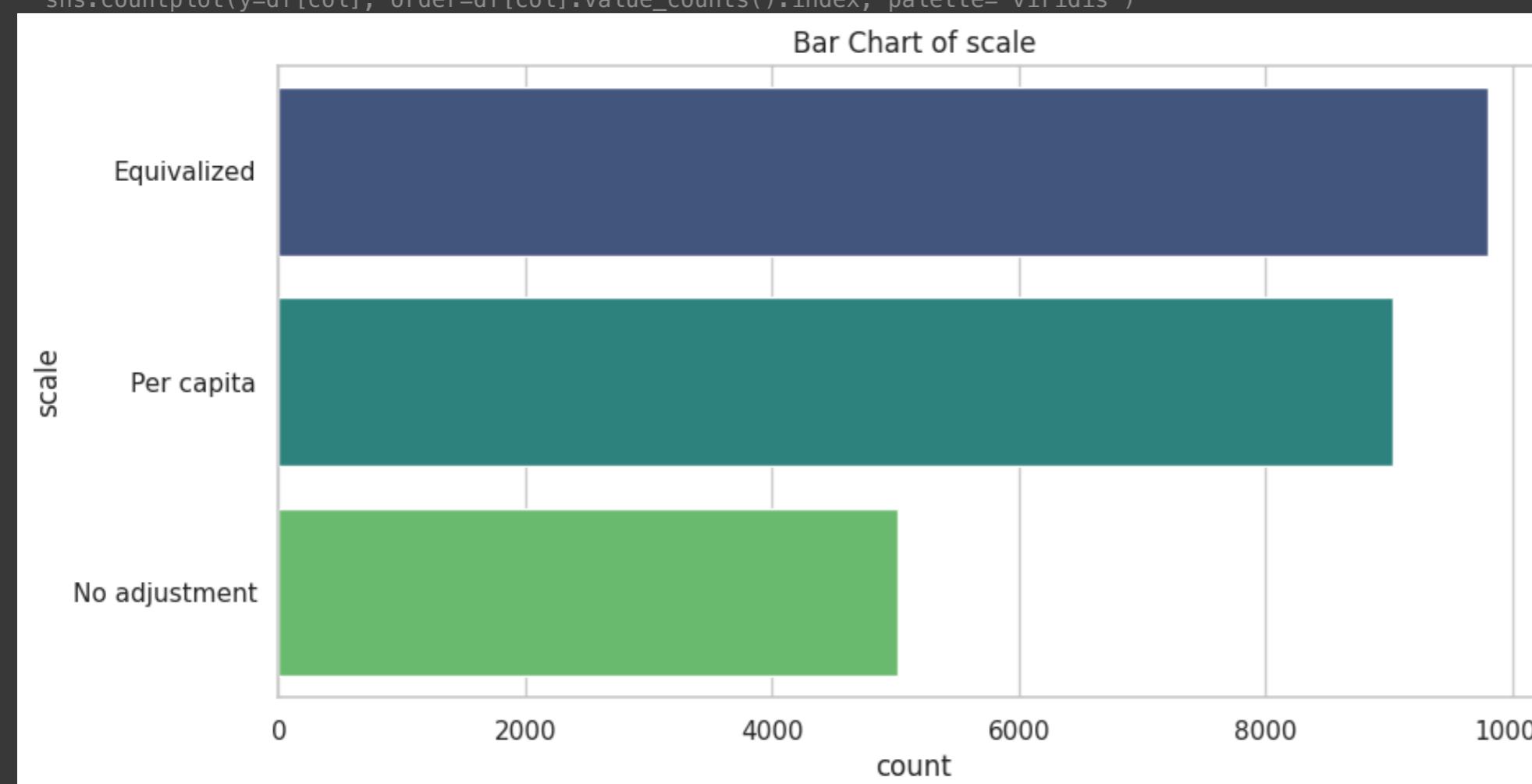
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

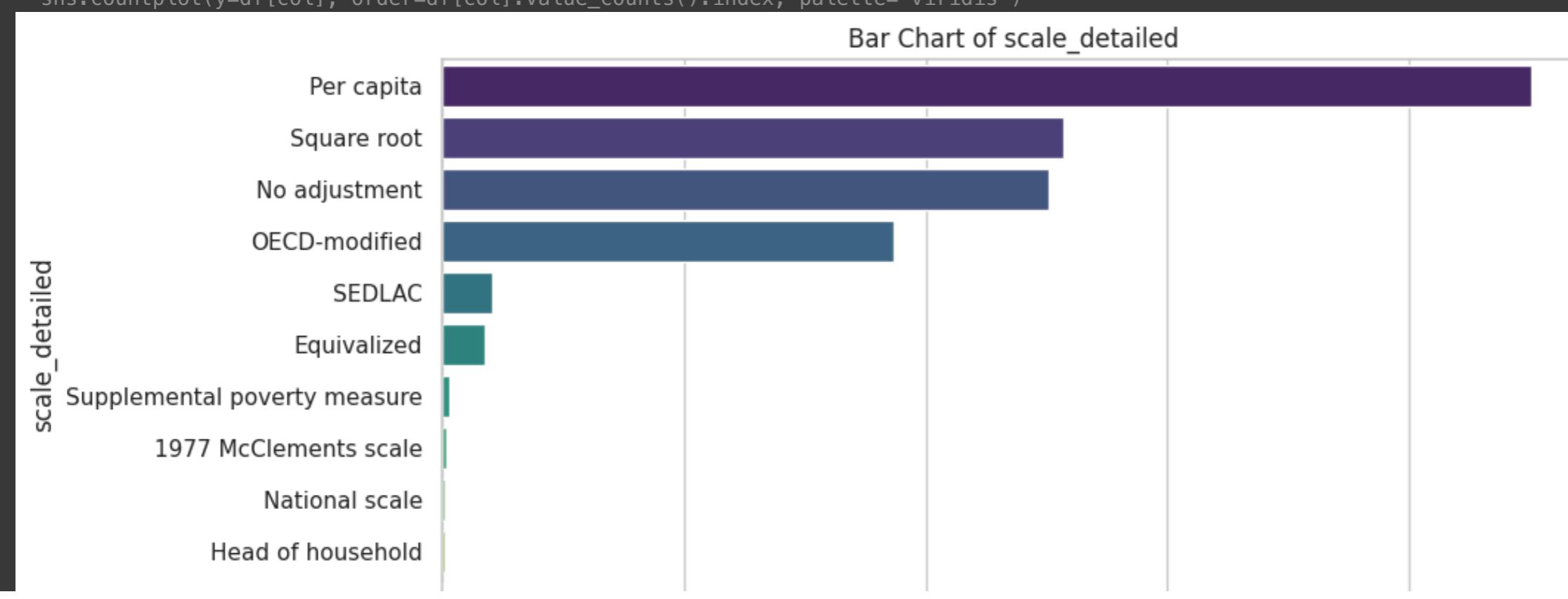
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```

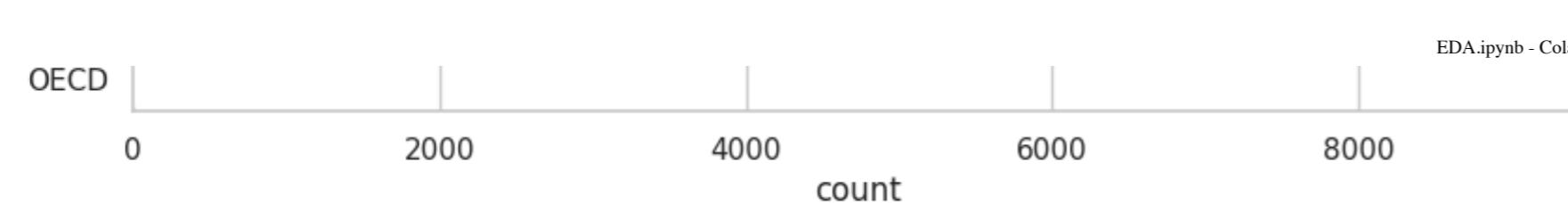


```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

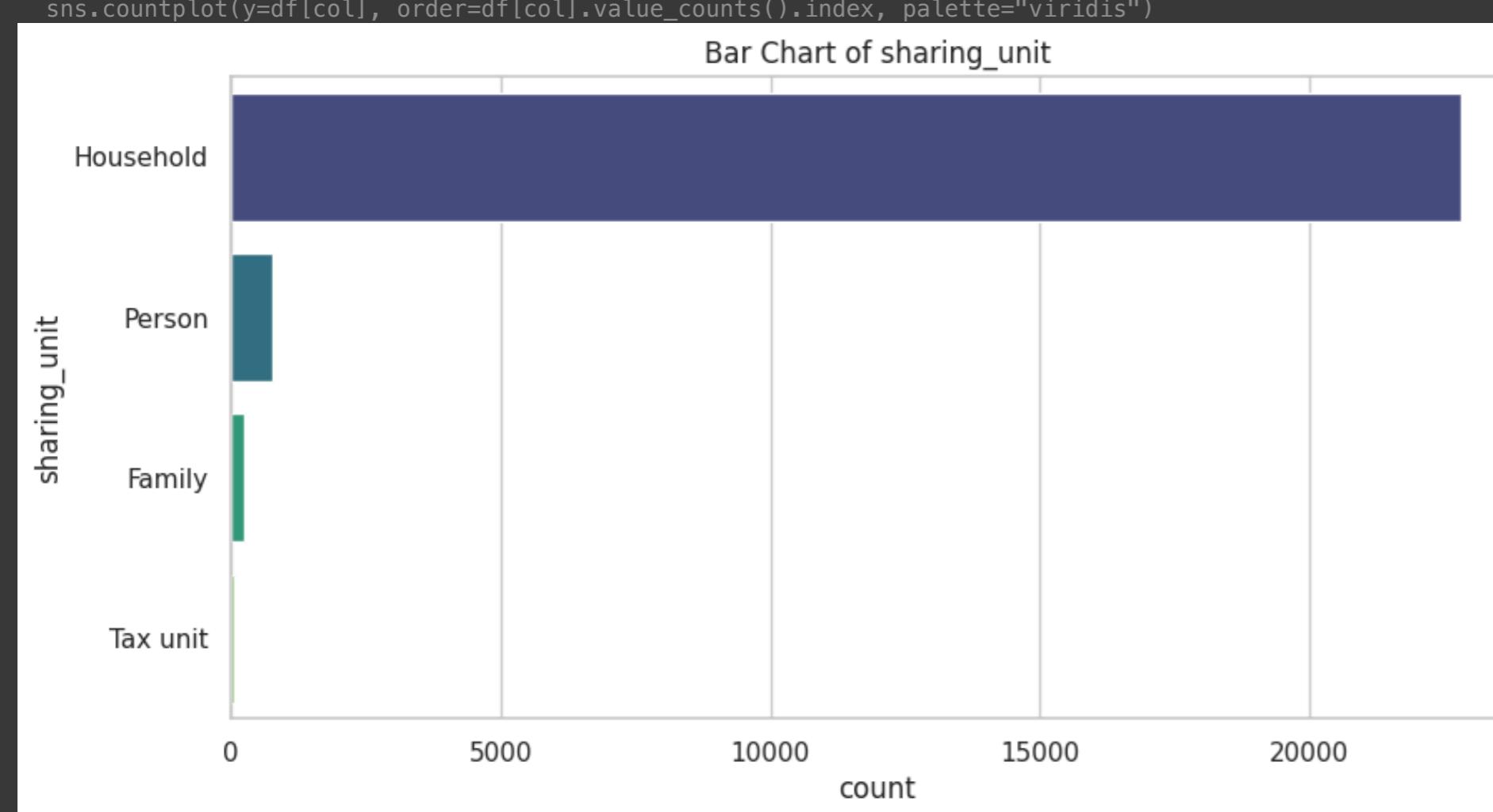
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```





```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```

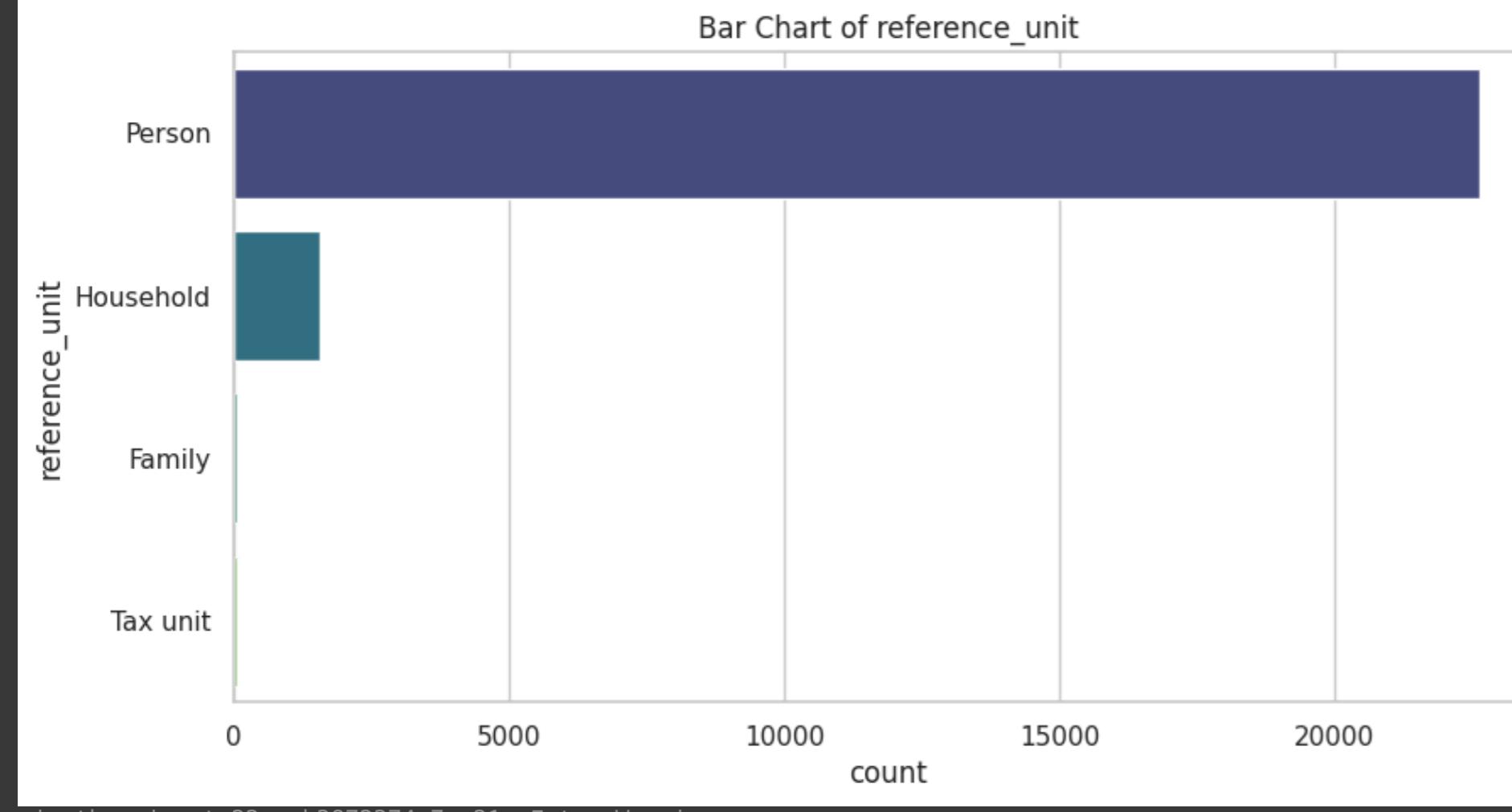
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```

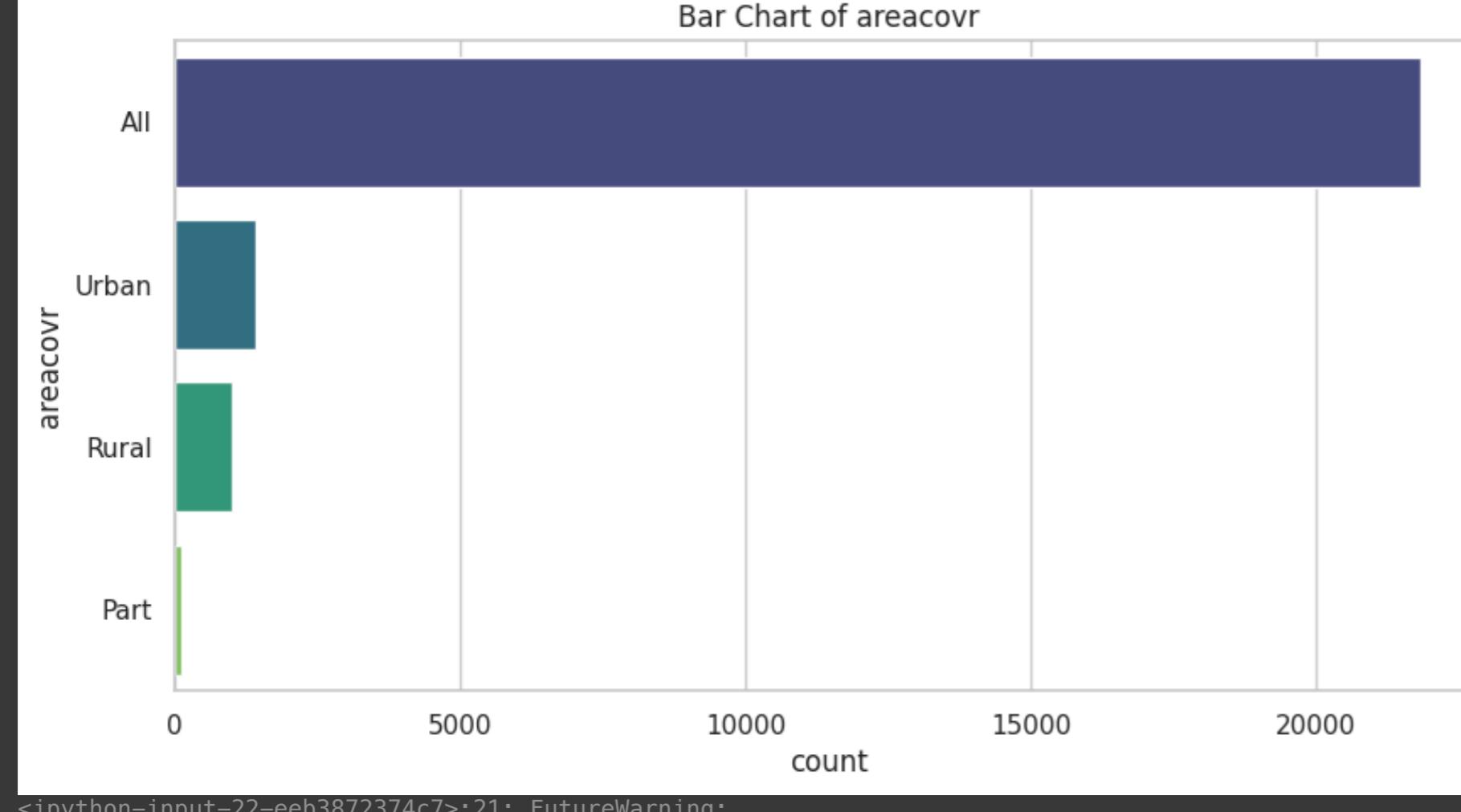
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```

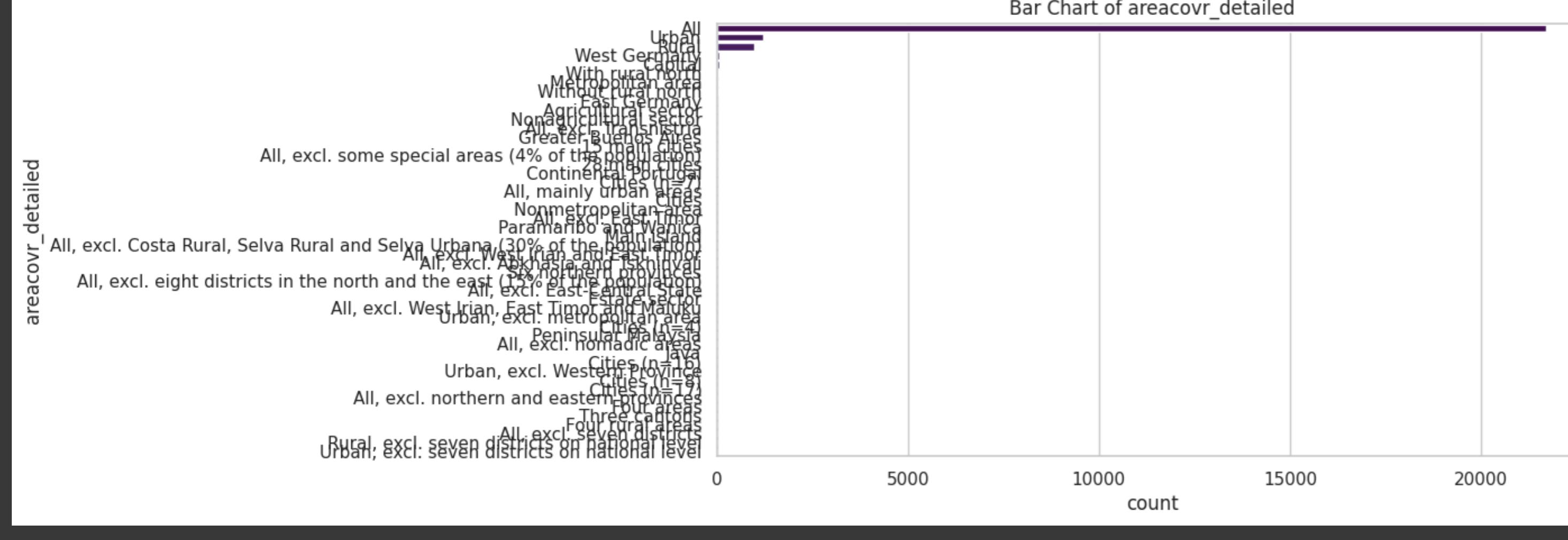
```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



```
<ipython-input-22-eeb3872374c7>:21: FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.
```

```
sns.countplot(y=df[col], order=df[col].value_counts().index, palette="viridis")
```



```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
id	24367.0	12184.000000	7034.291341	1.000000e+00	6092.500000	12184.000000	18275.500000	24367.000000
year	24367.0	2003.411622	14.585084	1.867000e+03	1997.000000	2007.000000	2014.000000	2022.000000
gini	24367.0	37.065117	9.096210	1.210000e+01	30.440000	35.400000	42.800000	78.600000
ge0	11723.0	951.307507	1109.840124	5.530000e+00	22.65000	50.270000	1893.650000	5332.580000
ge1	13395.0	814.745016	1036.510708	5.960000e+00	23.82000	50.400000	1754.240000	5361.630000
ge2	11784.0	1566.812687	4173.783279	6.670000e+00	33.40750	136.525000	2603.402500	159490.990000
a025	10989.0	236.796683	257.855927	1.440000e+00	5.24000	13.670000	466.870000	1057.520000
a050	13325.0	384.168245	485.786143	2.800000e+00	11.16000	21.380000	826.720000	2008.490000
a075	10989.0	681.557025	741.250193	4.110000e+00	14.99000	34.640000	1342.400000	3038.050000
a1	13209.0	758.916534	955.202356	5.380000e+00	21.51000	37.610000	1620.230000	4133.090000
a2	11714.0	1985.510212	2295.813624	1.074000e+01	45.19250	70.620000	4110.967500	8397.590000
palma	19007.0	1.808576	1.254811	5.200000e-01	1.12000	1.410000	2.030000	39.810000
ratio_top20bottom20	19865.0	8.145493	6.797569	1.890000e+00	4.76000	6.300000	9.200000	293.000000
bottom40	18860.0	17.933425	4.302160	1.600000e+00	15.00000	18.355000	21.160000	31.160000
q1	19030.0	6.531106	2.074778	2.000000e-01	5.03250	6.580000	8.100000	14.600000
q2	18864.0	11.403914	2.306268	1.300000e+00	9.91000	11.790000	13.100000	17.180000
q3	18885.0	15.941342	2.119216	4.200000e+00	14.92000	16.560000	17.420000	21.100000
q4	18891.0	22.266236	1.597061	1.060000e+01	21.70000	22.570000	23.190000	31.800000
q5	19031.0	43.877454	7.354925	2.550000e+01	38.70000	42.010000	47.655000	81.400000
d1	18082.0	2.469968	0.950078	-1.000000e-01	1.79000	2.450000	3.180000	7.000000
d2	18074.0	4.077017	1.138688	2.000000e-01	3.27000	4.150000	4.950000	7.610000
d3	18074.0	5.178075	1.166559	5.000000e-01	4.40000	5.330000	6.040000	8.200000
d4	18074.0	6.235170	1.150456	8.000000e-01	5.54000	6.470000	7.080000	8.700000
d5	18094.0	7.343845	1.105887	1.000000e-01	6.75000	7.630000	8.130000	9.530000
d6	18099.0	8.598824	1.021466	2.700000e+00	8.14000	8.920000	9.290000	10.780000
d7	18105.0	10.119569	0.898464	4.240000e+00	9.79000	10.390000	10.680000	13.900000
d8	18102.0	12.139420	0.742304	6.250000e+00	11.80000	12.170000	12.550000	17.900000
d9	18121.0	15.385310	1.007931	8.000000e+00	14.67000	15.400000	16.070000	24.000000
d10	18158.0	28.485110	6.833391	1.612000e+01	23.80000	26.420000	31.460000	70.240000
bottom5	11619.0	0.981927	0.387118	-1.000000e+00	0.70000	0.970000	1.255000	2.460000
top5	12270.0	16.825234	4.560707	9.100000e+00	14.21000	15.810000	17.830000	60.000000
exchangerate	4031.0	0.694842	6.315205	8.090000e-14	0.01807	0.146383	0.474084	149.395332
mean_usd	64.0	2793.031250	5251.287207	3.200000e+01	520.50000	979.000000	1959.000000	25450.000000
median_usd	14.0	4338.428571	5458.003040	5.710000e+02	1096.00000	1502.500000	5648.250000	15713.000000
quality_score	24367.0	11.737678	2.029691	3.000000e+00	11.00000	13.000000	13.000000	13.000000

Start coding or generate with AI.

```
# Descriptive statistics for Gini coefficient
df['gini'].describe()
```

	gini
count	24367.000000
mean	37.065117
std	9.096210
min	12.100000
25%	30.440000
50%	35.400000
75%	42.800000
max	78.600000

dtype: float64

Inference:

- The mean Gini coefficient is 37.07, indicating moderate income inequality on average.
- The standard deviation (9.10) suggests noticeable variation in inequality across observations.
- The minimum value (12.1) represents a very low inequality scenario, while the maximum (78.6) shows extreme inequality.
- The median (35.4)** is slightly lower than the mean, suggesting a slight right-skew in income inequality distribution.
- The interquartile range (IQR: 30.44 - 42.8) shows that 50% of the data lies within this range, indicating that most countries fall within a moderate inequality level.

```
df['gdp'].describe()
```

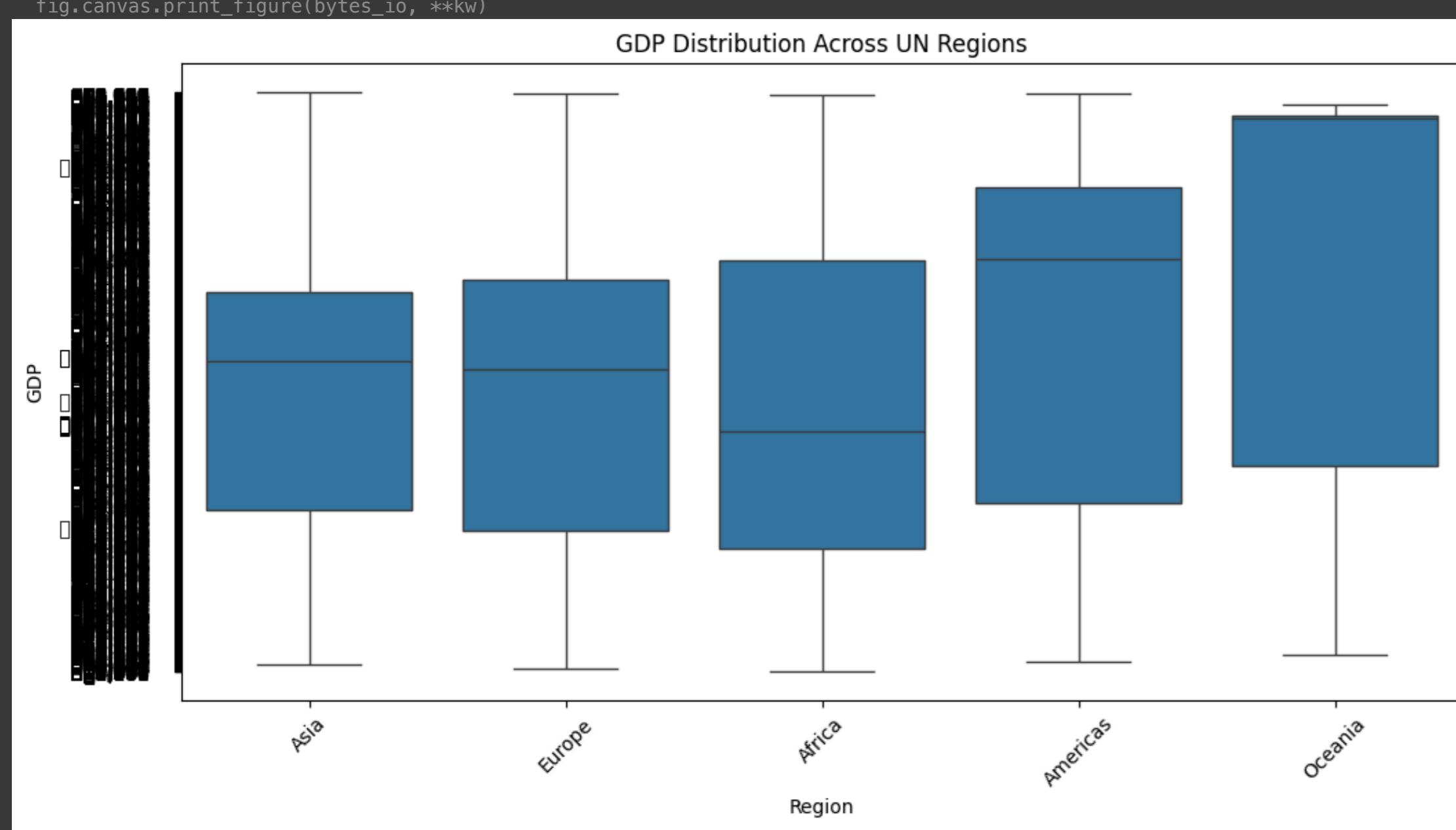
	gdp
count	24251
unique	3703
top	\t53,848
freq	41

dtype: object

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12,6))
sns.boxplot(x=df['region_un'], y=df['gdp'])
plt.xticks(rotation=45)
plt.title("GDP Distribution Across UN Regions")
plt.xlabel("Region")
plt.ylabel("GDP")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 ( ) missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)
```



Spread of GDP:

The Americas and Oceania have the widest spread in GDP, indicating a large economic disparity within these regions.

Africa has a relatively lower median GDP compared to other regions.

Median GDP Comparison:

The median GDP is highest in Oceania and the Americas, meaning these regions generally have higher economic output.

Africa has the lowest median GDP, highlighting economic challenges in that region.

```
import seaborn as sns
import matplotlib.pyplot as plt

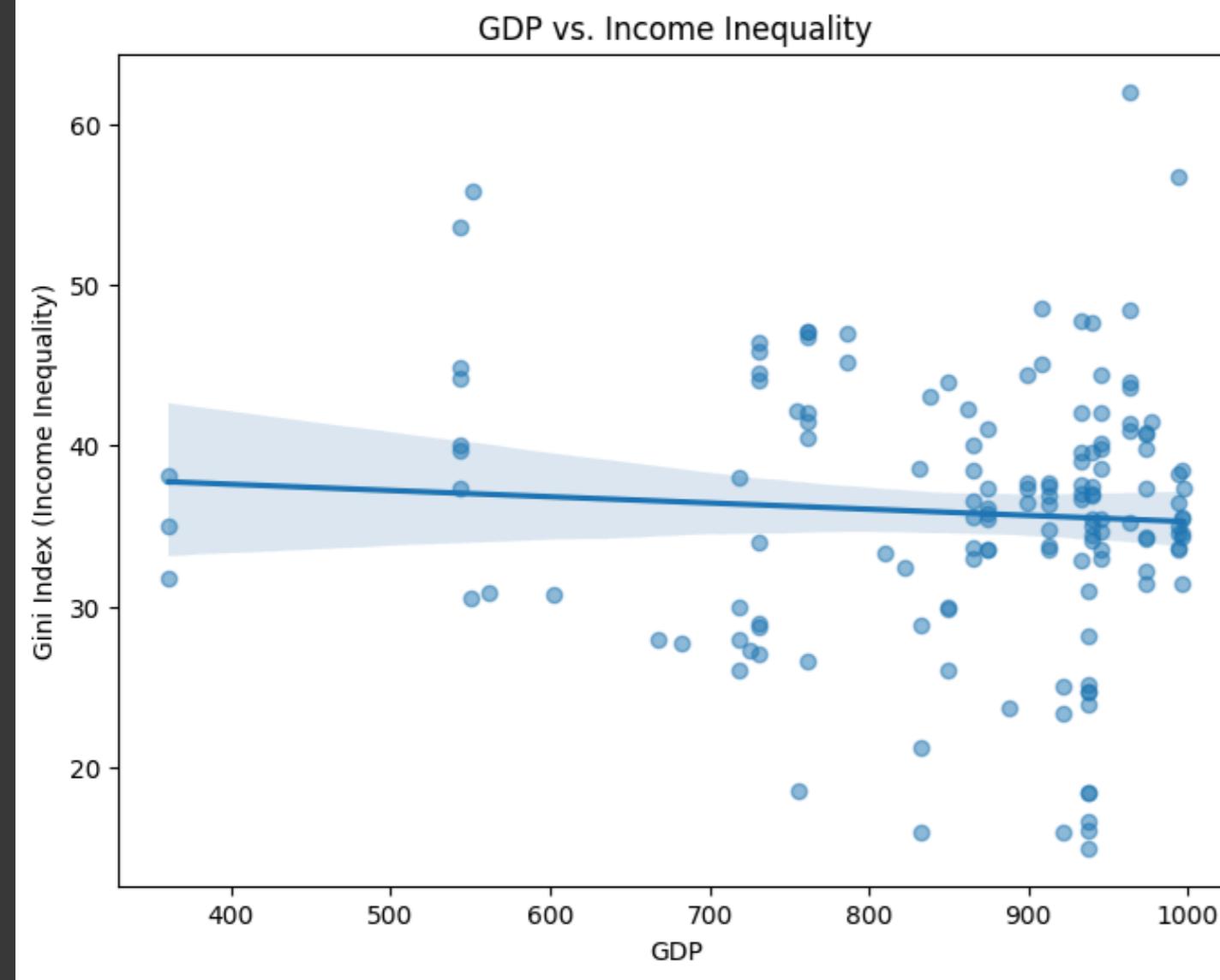
# Convert GDP column to numeric if it's not already
df['gdp'] = pd.to_numeric(df['gdp'], errors='coerce')

# Drop NaN values for accurate correlation calculation
df_cleaned = df[['gdp', 'gini']].dropna()

# Calculate correlation
correlation = df_cleaned['gdp'].corr(df_cleaned['gini'])
print(f"Correlation between GDP and Gini Index: {correlation}")

# Scatter plot with trend line
plt.figure(figsize=(8,6))
sns.replot(x='gdp', y='gini', data=df_cleaned, scatter_kws={'alpha':0.5})
plt.xlabel("GDP")
plt.ylabel("Gini Index (Income Inequality)")
plt.title("GDP vs. Income Inequality")
plt.show()
```

Correlation between GDP and Gini Index: -0.06549886782691541



GDP vs. Gini Index Analysis Weak Negative Correlation: The trend line in the scatter plot shows a slightly negative slope, indicating a weak inverse relationship between GDP and the Gini index. This suggests that as GDP increases, income inequality (Gini index) tends to decrease slightly, but the correlation is not strong.

High Dispersion: There is significant variability in Gini index values for similar GDP levels, meaning other factors beyond GDP influence income inequality.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

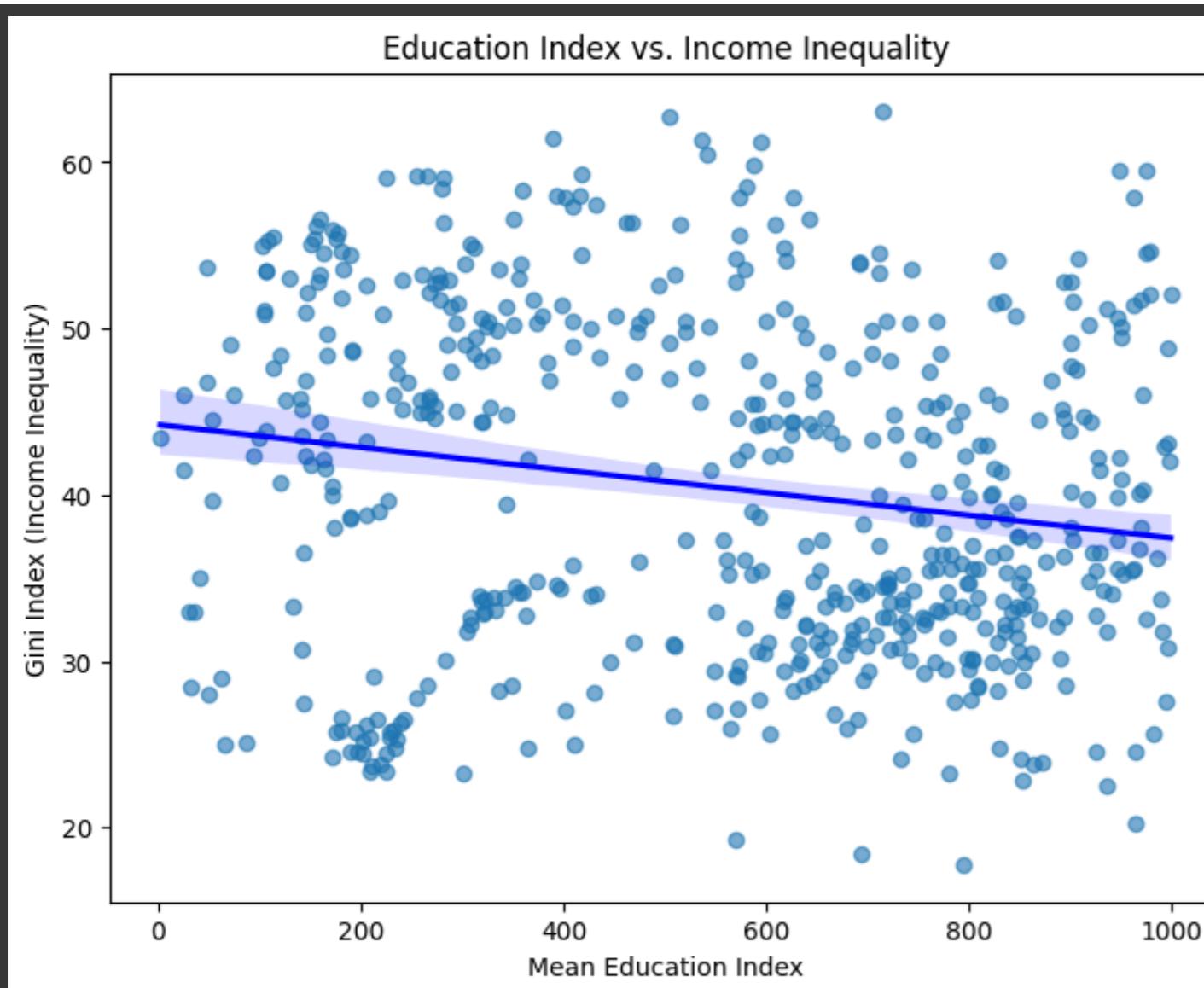
df['mean'] = pd.to_numeric(df['mean'], errors='coerce')
df['gini'] = pd.to_numeric(df['gini'], errors='coerce')

df = df.dropna(subset=['mean', 'gini'])

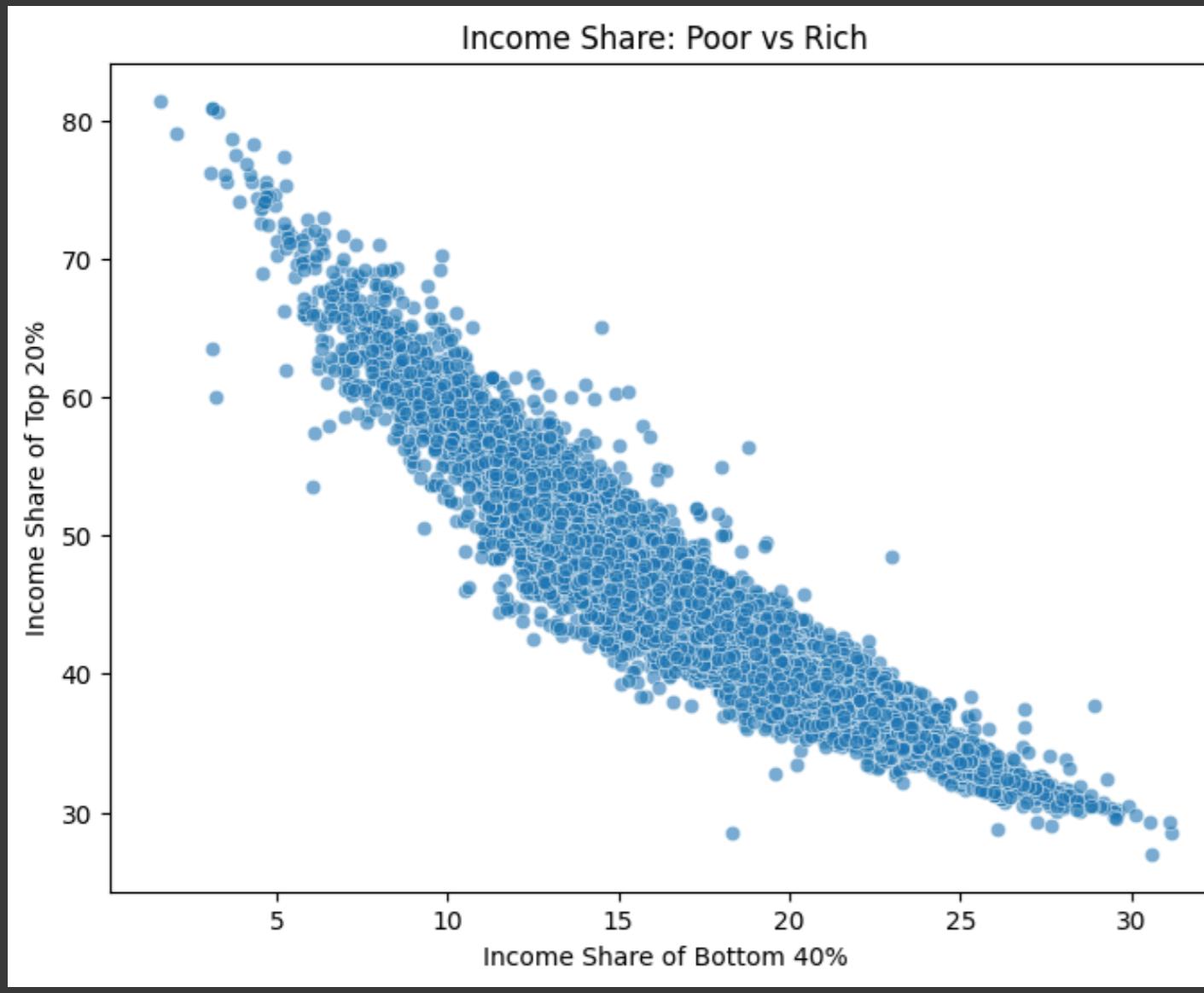
plt.figure(figsize=(8, 6))
sns.replot(x=df['mean'], y=df['gini'], scatter_kws={'alpha': 0.6}, line_kws={'color': 'blue'})

plt.xlabel("Mean Education Index")
plt.ylabel("Gini Index (Income Inequality)")
plt.title("Education Index vs. Income Inequality")

plt.show()
```



```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['bottom40'], y=df['q5'], alpha=0.6)
plt.xlabel('Income Share of Bottom 40%')
plt.ylabel('Income Share of Top 20%')
plt.title('Income Share: Poor vs Rich')
plt.show()
```

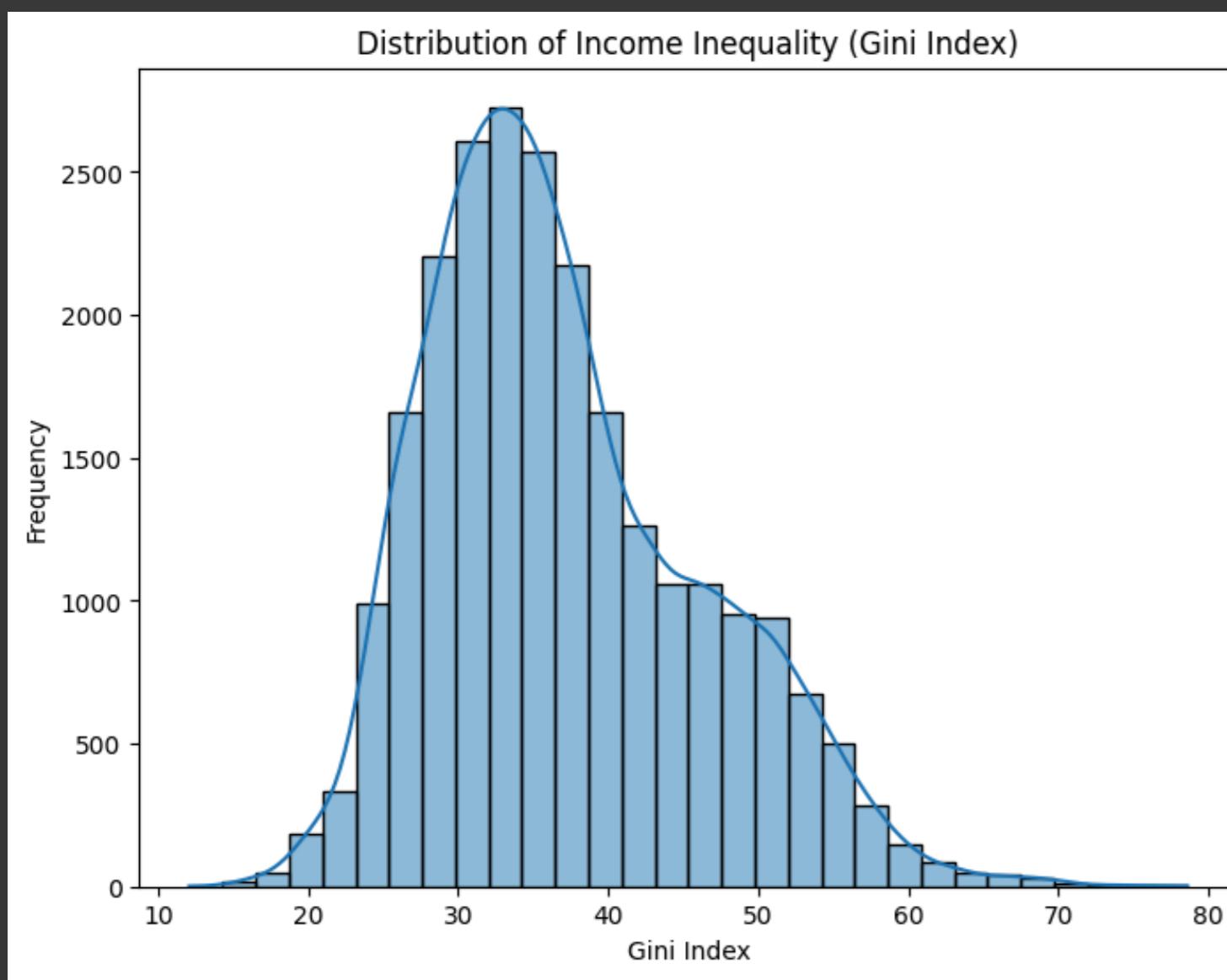


Inference: There's a strong negative correlation between the income share of the bottom 40% and the top 20%.

As the bottom 40% get a smaller share of income, the top 20% take a larger share (inequality is high).

When the bottom 40% get more income, the rich earn lesser proportionally, indicating a more equal society.

```
plt.figure(figsize=(8, 6))
sns.histplot(df['gini'], bins=30, kde=True)
plt.xlabel('Gini Index')
plt.ylabel('Frequency')
plt.title('Distribution of Income Inequality (Gini Index)')
plt.show()
```



Inference: The Gini Index follows a right-skewed (slightly normal) distribution.

Most countries have a Gini Index between 25 and 45, meaning moderate income inequality.

A few countries have a Gini Index above 60, indicating extreme income inequality.

The peak suggests that a large number of observations fall around 30-35, implying that most economies experience some level of inequality but not extreme.

```
mean_gini = df['gini'].mean()
median_gini = df['gini'].median()

print(f"Mean Gini Index: {mean_gini:.2f}")
print(f"Median Gini Index: {median_gini:.2f}")
```

Mean Gini Index: 37.05
Median Gini Index: 35.40

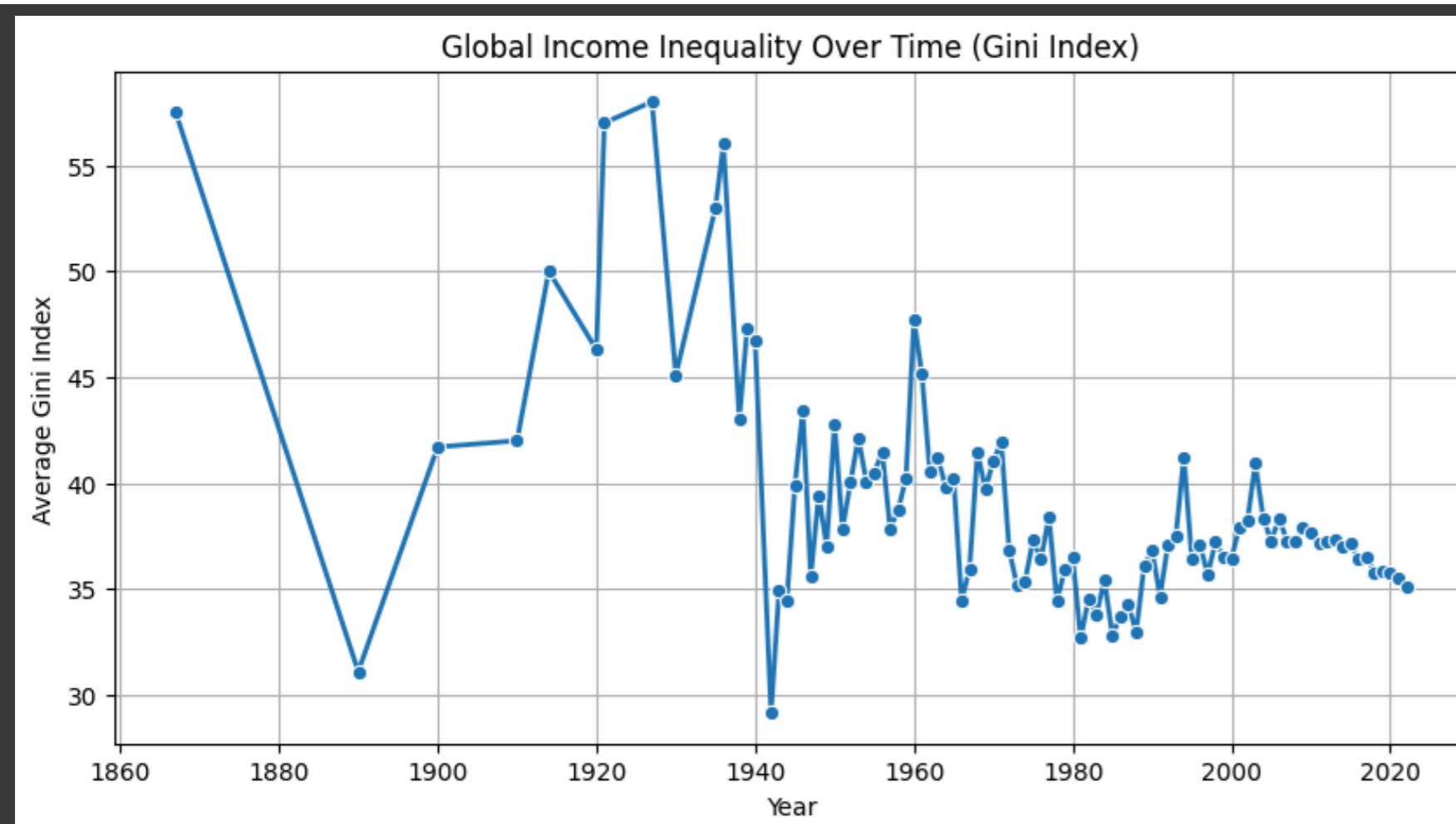
✖ Mean Gini Index (37.05) > Median Gini Index (35.40) ➔ This suggests a right-skewed distribution, meaning some countries have very high income inequality, pulling the average up.

```
import matplotlib.pyplot as plt
import seaborn as sns

gini_trend = df.groupby("year")["gini"].mean()

plt.figure(figsize=(10, 5))
sns.lineplot(x=gini_trend.index, y=gini_trend.values, marker="o", linewidth=2)

plt.title("Global Income Inequality Over Time (Gini Index)")
plt.xlabel("Year")
plt.ylabel("Average Gini Index")
plt.grid(True)
plt.show()
```



Sharp Decline (1860–1880): Likely due to industrialization and economic expansion.

Fluctuations (1900–1950): Wars and economic crises (WWI, Great Depression, WWII) caused volatility.

Post-WWII Decline (1950–1980): Global economic growth, welfare policies, and labor rights improved income equality.

Recent Stability (1980–2020): Economic liberalization led to moderate inequality, with slight declines in recent years.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df.rename(columns={'year': 'Year', 'gini': 'Gini_Index', 'region_un': 'Region'}, inplace=True)

df['Year'] = pd.to_numeric(df['Year'], errors='coerce')
df = df.dropna(subset=['Year', 'Gini_Index', 'Region'])

df = df[df['Region'].notna()]

plt.figure(figsize=(12, 6))

df_sorted = df.sort_values(by=['Region', 'Year'])
df_sorted['Gini_Index_Smoothed'] = df_sorted.groupby('Region')[['Gini_Index']].transform(lambda x: x.rolling(5, min_periods=1).mean())

sns.lineplot(data=df_sorted, x='Year', y='Gini_Index_Smoothed', hue='Region', marker='o')

plt.title('Regional Income Inequality Trends Over Time (Smoothed Gini Index)')
plt.xlabel('Year')
plt.ylabel('Gini Index')
plt.legend(title='Region', loc='upper left')
plt.grid(True)

plt.show()
```

<ipython-input-19-c911bc9a227>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df.rename(columns={'year': 'Year', 'gini': 'Gini_Index', 'region_un': 'Region'}, inplace=True)

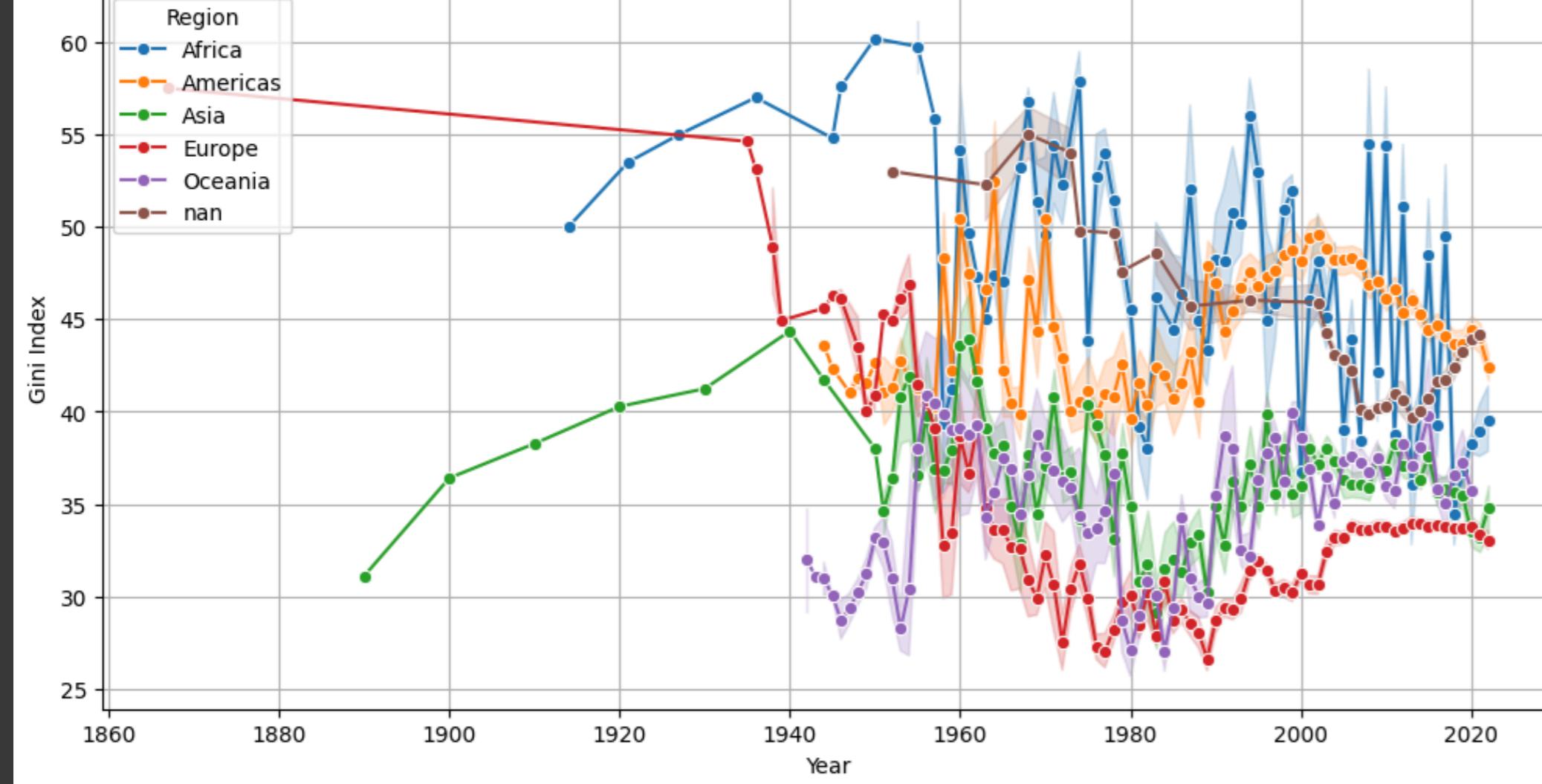
<ipython-input-19-c911bc9a227>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

df['Year'] = pd.to_numeric(df['Year'], errors='coerce')

Regional Income Inequality Trends Over Time (Smoothed Gini Index)



1. Income Inequality Has Been Dynamic Over Time
 - No region shows a perfectly stable trend—inequality has fluctuated significantly across time periods.
 - Wars, economic crises, and policy changes likely influenced these shifts.
2. Africa Experienced the Most Volatile Inequality Trends
 - A sharp rise in inequality before mid-20th century.
 - Post-1960s, it remains highly unstable, reflecting economic turbulence and structural inequalities.
3. Americas Show a Long-Term Increase in Inequality Post-1980s
 - Likely linked to neoliberal economic policies, globalization, and wage disparities.
4. Europe & Oceania Have More Stable and Declining Inequality
 - Strong social policies and wealth distribution mechanisms likely contributed to this.
 - Europe, in particular, shows a consistent downward trend in inequality after the 1940s.
5. Asia's Inequality Peaked in the Early 20th Century, Then Stabilized
 - Likely due to industrialization, economic reforms, and development initiatives.

```
highest_gini = df.nlargest(10, 'gini')[['country', 'year', 'gini']]
print("Top 10 Highest Gini Index Values:\n", highest_gini)
```

```
lowest_gini = df.nsmallest(10, 'gini')[['country', 'year', 'gini']]
print("Top 10 Lowest Gini Index Values:\n", lowest_gini)
```

Top 10 Highest Gini Index Values:

	country	year	gini
14180	Mali	1994	78.60
24332	Zambia	1991	77.30
19739	South Africa	2011	77.10
14421	Mauritania	1987	76.00
19729	South Africa	2010	75.02
24359	Zimbabwe	1995	74.60
14991	Namibia	1994	74.30
19725	South Africa	2010	73.32
19708	South Africa	2001	73.00
19715	South Africa	2008	72.63

Top 10 Lowest Gini Index Values:

	country	year	gini
3655	China	1982	12.1
16205	Pakistan	1970	14.6
3653	China	1981	15.0
3659	China	1982	15.0
3665	China	1983	15.0
18068	Romania	1989	15.5
18073	Romania	1989	15.6
3661	China	1983	15.8
3678	China	1985	15.8
2368	Bulgaria	1968	15.9

- Highest Gini Index Values:
 - Countries like Mali, Zambia, South Africa, Mauritania, Zimbabwe, and Namibia have recorded the highest Gini index
 - South Africa appears multiple times (2011, 2010, 2008, 2001), suggesting that it has consistently had high income
 - The highest Gini index (78.60) is observed in Mali (1994), showing severe disparity in income distribution.
- Lowest Gini Index Values:
 - Countries like China, Pakistan, Romania, and Bulgaria have recorded the lowest Gini index values, indicating relative income equality
 - China appears multiple times (1981, 1982, 1983, 1985), suggesting that it had a relatively equal income distribution
 - The lowest Gini index (12.1) is observed in China (1982), indicating one of the most equitable income distributions.

```
!pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.11/dist-packages (0.14.4)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.0.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.14.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (2.2.2)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.0.1)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.17.0)
```

```
numerical_df = df.select_dtypes(include=['number'])
```

```
# Compute correlation matrix
correlation_matrix = numerical_df.corr()
```

```
# Display correlation matrix
print(correlation_matrix)
```

```
id          id      year     gini    ge0    ge1  \
id  1.000000  0.003992 -0.046920 -0.013157 -0.012447
year  0.003992  1.000000 -0.047856  0.246331  0.237350
gini -0.046920 -0.047856  1.000000 -0.105171 -0.226905
ge0  -0.013157  0.246331 -0.105171  1.000000  0.988584
ge1  -0.012447  0.237350 -0.226905  0.988584  1.000000
ge2  -0.020386  0.108664 -0.035693  0.437494  0.504960
a025 -0.075911  0.253026 -0.028861  0.993682  0.998676
a050  0.000119  0.235959 -0.233301  0.996775  0.996709
a075 -0.068148  0.252111 -0.031904  0.998264  0.993392
a1   0.018005  0.230923 -0.234529  0.998957  0.990869
a2   -0.005295  0.245824 -0.164057  0.974997  0.953293
palma -0.078404 -0.121544  0.869860 -0.159379 -0.249254
ratio_top2bottom20 -0.062864 -0.102543  0.767734 -0.129647 -0.224681
bottom40  0.047890  0.063670 -0.981114  0.043648  0.179795
q1   0.030747  0.048446 -0.937471 -0.003609  0.140740
q2   0.061017  0.076037 -0.988416  0.085871  0.210057
q3   0.089891  0.109649 -0.952209  0.191461  0.284166
q4   0.126763  0.119499 -0.625940  0.330676  0.345791
q5   -0.078312 -0.103899  0.987992 -0.150600 -0.260232
d1   0.037252  0.021480 -0.886810 -0.041908  0.100247
d2   0.054672  0.057560 -0.964810  0.028252  0.158100
d3   0.064876  0.072046 -0.984066  0.063149  0.187396
d4   0.077791  0.086933 -0.988654  0.108629  0.224441
d5   0.087729  0.108400 -0.973944  0.159460  0.262126
d6   0.100506  0.124503 -0.926561  0.223706  0.304837
d7   0.116738  0.128032 -0.792781  0.297326  0.344520
d8   0.124944  0.122137 -0.392346  0.332396  0.329405
d9   0.027581  0.016292  0.643325  0.095179 -0.013740
d10  -0.098517 -0.122037  0.971027 -0.182405 -0.277831
bottom5 -0.090435 -0.112343 -0.790476 -0.166003 -0.110555
top5   0.016491 -0.188769  0.909186 -0.106154 -0.100755
exchangerate  0.070262  0.003223 -0.039543  NaN  NaN
mean_usd  0.032971  0.275955  0.396558  NaN  NaN
median_usd  0.273831 -0.446033  0.846158  NaN  NaN
quality_score  0.040119  0.420585 -0.239807  0.228782  0.351915

ge2      a025      a050      a075      a1  ...
id   -0.020386 -0.075911  0.000119 -0.068148  0.018005
year  0.108664  0.253026  0.235959  0.252111  0.230923
gini -0.035593 -0.028861 -0.233301 -0.031904 -0.234529
ge0   0.437494  0.993682  0.996775  0.998264  0.998957
ge1   1.000000  0.463674  0.454067  0.435046  0.436527
a025  0.463674  1.000000  0.999352  0.997773  0.995132
a050  0.454067  0.999352  1.000000  0.999499  0.998140
a075  0.435046  0.997773  0.999499  1.000000  0.999420
a1    0.436527  0.995132  0.998140  0.999420  1.000000
a2    0.408452  0.959020  0.967022  0.970933  0.978454
palma -0.058497 -0.088775 -0.253193 -0.092193 -0.257359
ratio_top2bottom20 -0.051280 -0.080418 -0.225415 -0.079583 -0.222903
bottom40  0.023613 -0.017481  0.178316 -0.019447  0.174666
q1   0.007962 -0.048789  0.135096 -0.056131  0.124923
q2   0.037382  0.012998  0.212262  0.016227  0.214364
q3   0.066611  0.101690  0.293309  0.113600  0.303590
q4   0.082742  0.234544  0.366277  0.258109  0.387992
q5   -0.050895 -0.066703 -0.266270 -0.073746 -0.271285
d1   -0.003805 -0.090967  0.091156 -0.103151  0.083149  ...
```

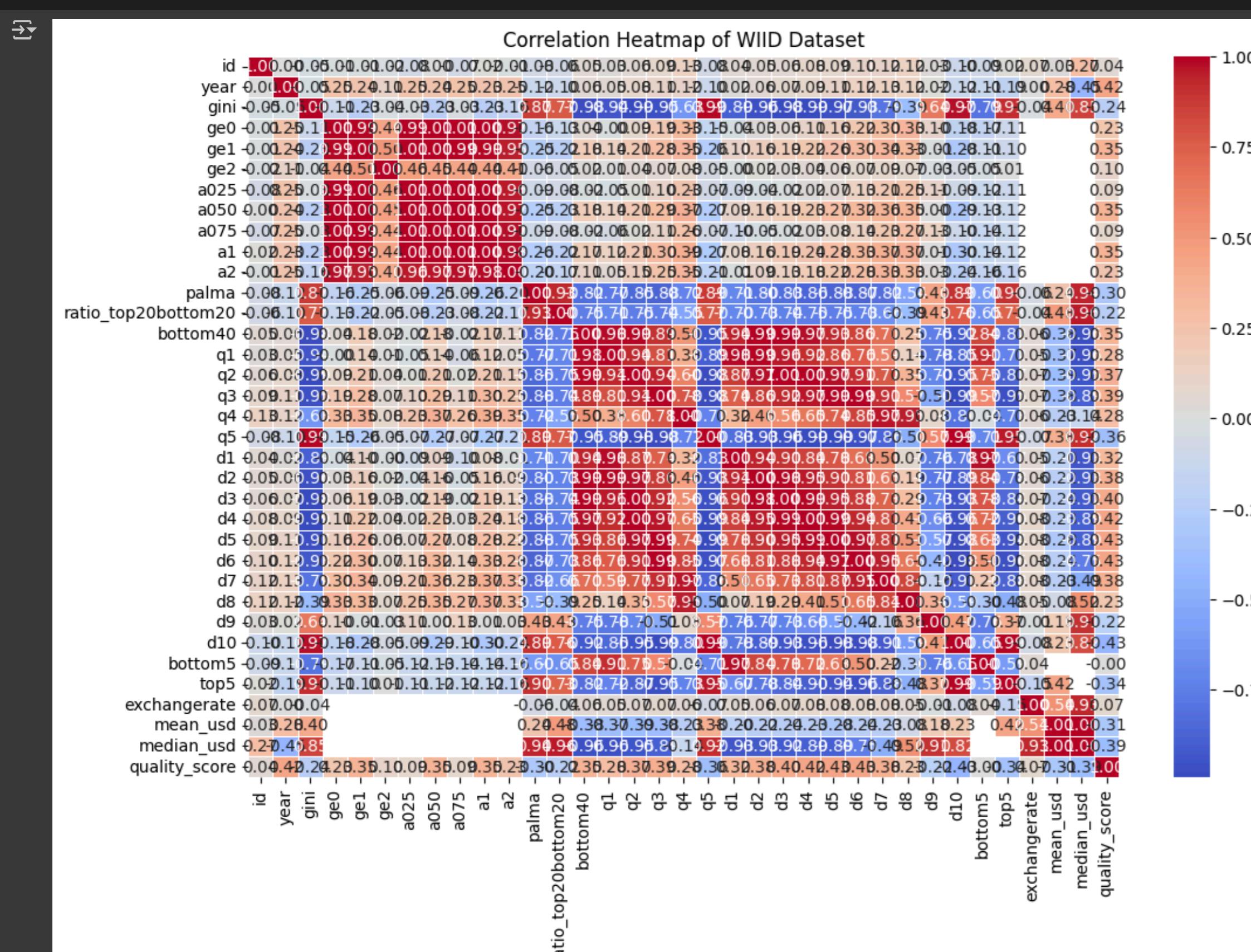
```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Select only numerical columns
numeric_df = df.select_dtypes(include=['number'])
```

```
# Compute the correlation matrix
correlation_matrix = numeric_df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

# Title
plt.title("Correlation Heatmap of WIID Dataset")
plt.show()
```



this shows the correlation strength of each data columns so we can use them for some analysis or choosing columns for next analysis

```
import pandas as pd

numeric_df = df.select_dtypes(include=['number'])
correlation_matrix = numeric_df.corr()

correlation_pairs = correlation_matrix.unstack().sort_values(ascending=False)
strong_correlations = correlation_pairs[(correlation_pairs > 0.7) & (correlation_pairs < 1)]
strong_negative_correlations = correlation_pairs[correlation_pairs < -0.7]

print("Top Positive Correlations:\n", strong_correlations.head(10))
print("\nTop Negative Correlations:\n", strong_negative_correlations.head(10))
```

1 Top Positive Correlations:

	a075	a050	0.999499
a050	a075	0.999499	
a1	a075	0.999420	
a075	a1	0.999420	
a025	a050	0.999352	
a050	a025	0.999352	
a1	ge0	0.998957	
ge0	a1	0.998957	
mean_usd	median_usd	0.998808	
median_usd	mean_usd	0.998808	

dtype: float64

2 Top Negative Correlations:

	d1	palma	-0.705491
palma	d1	-0.705491	
q5	bottom5	-0.707141	
bottom5	q5	-0.707141	
q1	ratio_top20bottom20	-0.712751	
ratio_top20bottom20	q1	-0.712751	
top5	q1	-0.721632	
q1	top5	-0.721632	
q4	q5	-0.722465	
q5	q4	-0.722465	

dtype: float64

1 Top Positive Correlations (Highly Related Features) a075 & a050 (0.9995), a1 & a075 (0.9994), a025 & a050 (0.9993)

These variables are almost identical in their behavior.

This suggests they might be measuring the same economic indicator or have a very strong relationship in the dataset.

Mean Income (mean_usd) & Median Income (median_usd) (0.9988)

This is expected as the mean and median of income distributions should follow similar trends.

Countries with higher mean incomes also tend to have higher median incomes.

a1 & ge0 (0.9989)

This could indicate a strong relationship between demographic or economic attributes being analyzed.

Further investigation is needed into what "a1" and "ge0" represent in the dataset.

2 Top Negative Correlations (Strong Opposite Trends) Palma Ratio & d1 (-0.705)

The Palma Ratio (a measure of inequality) is inversely correlated with "d1".

This suggests that when the Palma Ratio increases (higher inequality), "d1" decreases.

"d1" might represent the share of wealth of the lowest decile, meaning higher inequality reduces their income share.

q5 & bottom5 (-0.7071)

This suggests that when the wealth of the top 5% (q5) increases, the wealth share of the bottom 5% (bottom5) decreases.

Indicates a widening economic gap between the richest and the poorest.

ratio_top20bottom20 & q1 (-0.7127)

The ratio of the top 20% to bottom 20% wealth is negatively correlated with q1 (wealth of the lowest quantile).

This confirms that higher inequality leads to a decline in the income of the poorest group.

q1 & top5 (-0.7216)

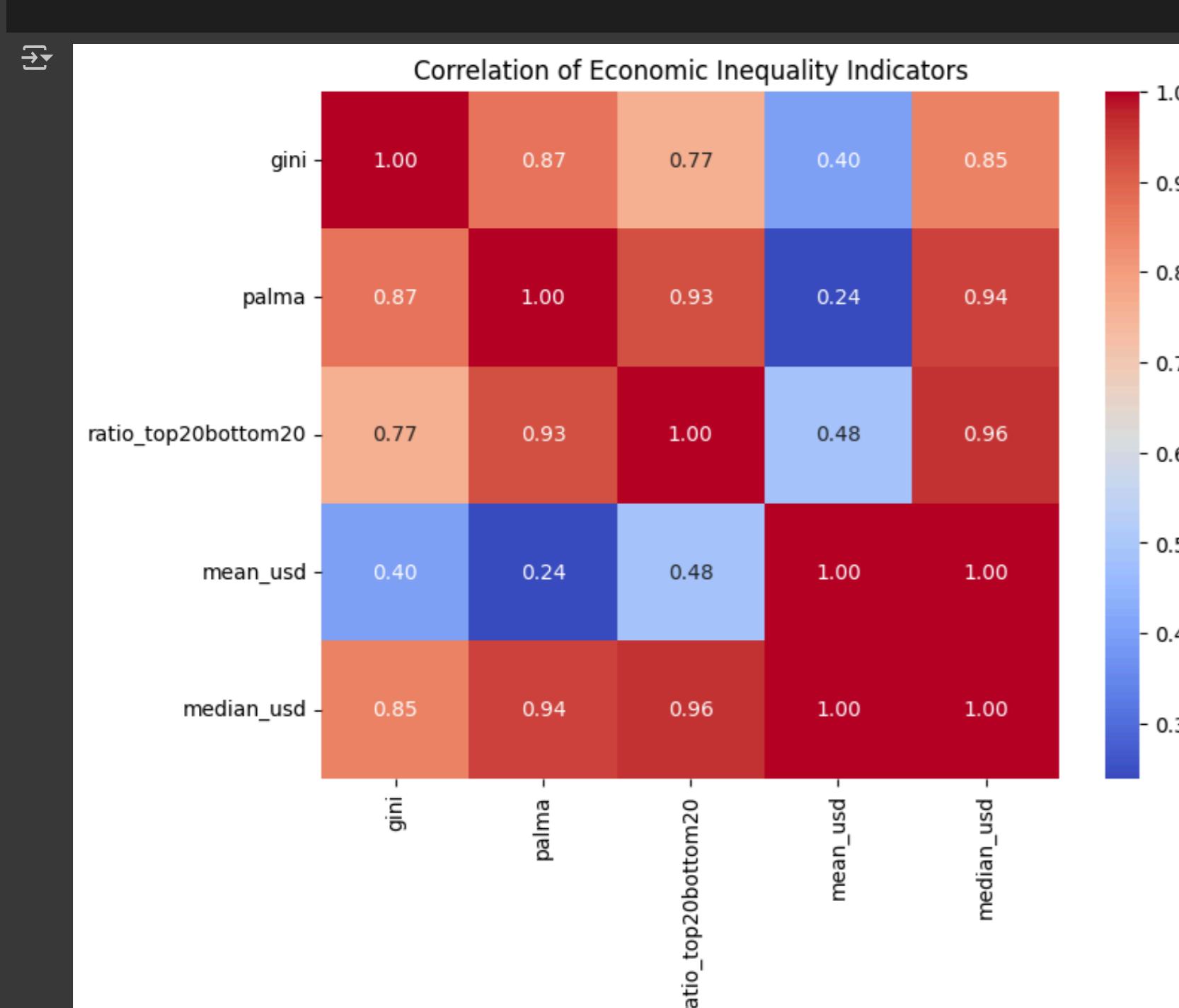
As the wealth share of the top 5% increases, the wealth share of the poorest 1% decreases.

This further supports the trend of increasing income disparity.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Select key economic indicators
selected_features = ['gini', 'palma', 'ratio_top20bottom20', 'mean_usd', 'median_usd']
correlation_inequality = df[selected_features].corr()

# Heatmap for inequality-related factors
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_inequality, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation of Economic Inequality Indicators")
plt.show()
```



1 Strong Positive Correlations Median income & Ratio of top 20% to bottom 20% (0.96)

As the income disparity between the top 20% and bottom 20% increases, the median income also increases.

This could indicate that while inequality rises, overall median income still grows, possibly due to high-income earners pulling up the median.

Palma Ratio & Median Income (0.94)

The Palma Ratio (income of top 10% / bottom 40%) strongly correlates with median income.

This suggests that countries with a higher Palma Ratio also tend to have a higher median income, reinforcing the idea that wealth concentration at the top influences the overall income distribution.

Gini Coefficient & Palma Ratio (0.87)

The Gini coefficient and Palma ratio are both measures of inequality, so their high correlation makes sense.

When the Palma ratio is high (more income held by the top 10%), the overall income distribution (Gini) is also highly unequal.

2 Moderate Correlations Gini Coefficient & Ratio of top 20% to bottom 20% (0.77)

Both indicators measure income inequality, so a strong correlation is expected.

However, since the ratio of top 20% to bottom 20% focuses on extremes, it may capture trends the Gini coefficient misses.

Mean Income & Ratio of top 20% to bottom 20% (0.48)

The relationship here is weaker, suggesting that while inequality may rise, mean income is not always directly affected.

Possible explanation: High incomes at the top could pull the mean up without affecting the lower end significantly.

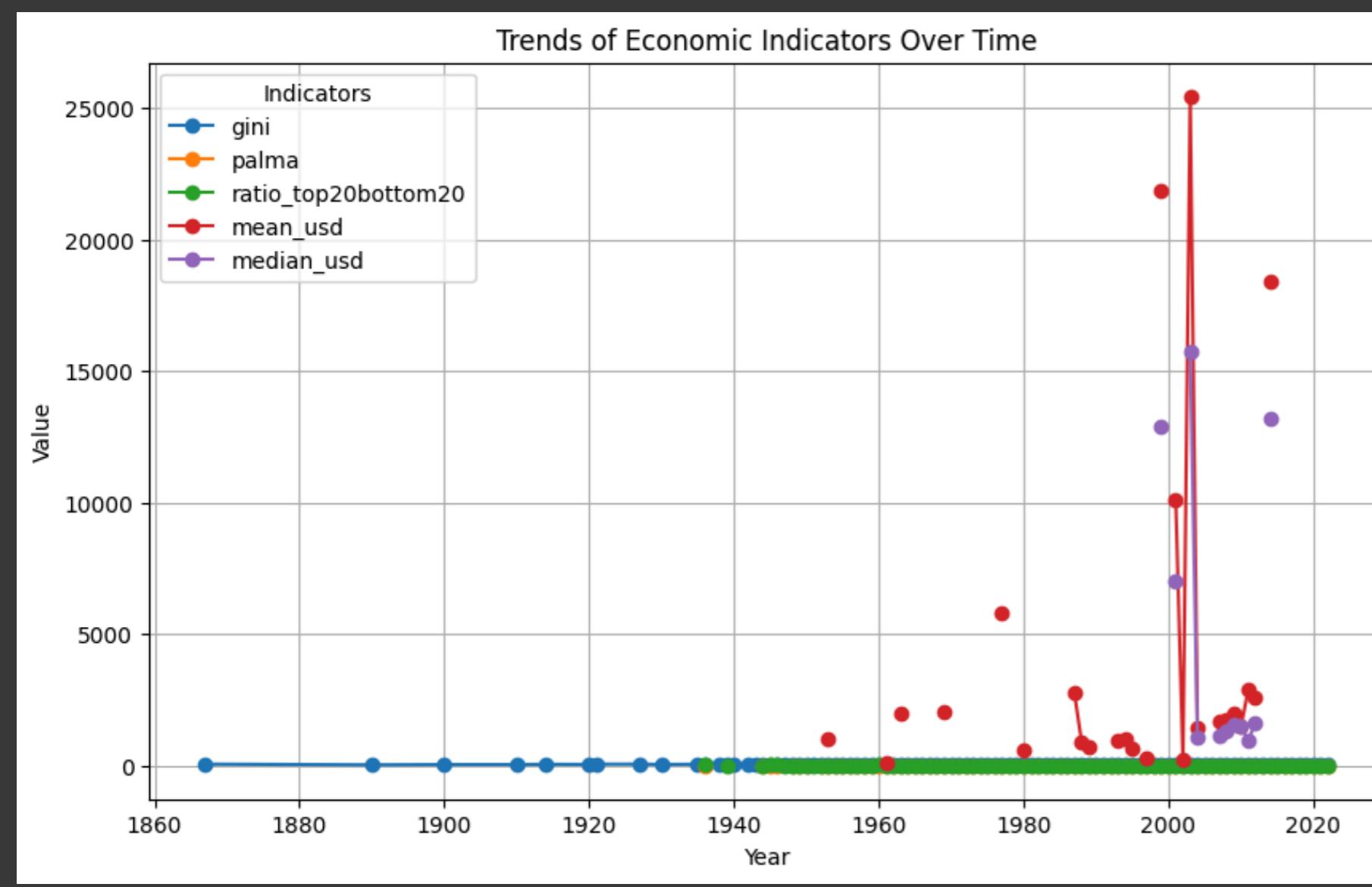
3 Weakest Correlations Mean Income & Palma Ratio (0.24)

A surprisingly weak correlation indicates that the Palma Ratio (which focuses on top and bottom income distribution) does not always reflect changes in mean income.

This suggests that inequality can rise even when average income remains stable.

```
df_grouped = df.groupby("year")[selected_features].mean()

df_grouped.plot(kind='line', figsize=(10, 6), marker='o')
plt.title("Trends of Economic Indicators Over Time")
plt.xlabel("Year")
plt.ylabel("Value")
plt.legend(title="Indicators")
plt.grid()
plt.show()
```



1 Historical Stability in Inequality Indicators (Gini, Palma, Ratio of Top 20% to Bottom 20%) Before 1950, the Gini coefficient (blue) and Palma ratio (orange) appear stable at low values.

Ratio of top 20% to bottom 20% (green) also remains relatively flat.

This suggests that major shifts in income inequality were rare or not well-documented before the mid-20th century.

2 Post-2000 Surge in Mean and Median Incomes (Red and Purple) A sharp spike in mean income (red) and median income (purple) occurs around 2000.

These fluctuations indicate a period of economic instability or rapid economic growth, possibly due to policy changes, globalization, or technological advancements.

The extreme variation suggests possible data inconsistencies, outliers, or economic crises (e.g., financial bubbles, hyperinflation).

3 Economic Growth and Inequality Trends Might be Disconnected Despite rising incomes after 2000, Gini and Palma remain mostly stable rather than increasing significantly.

This suggests that income growth may not always be linked to rising inequality—some economic booms benefit broader populations rather than just the wealthy.

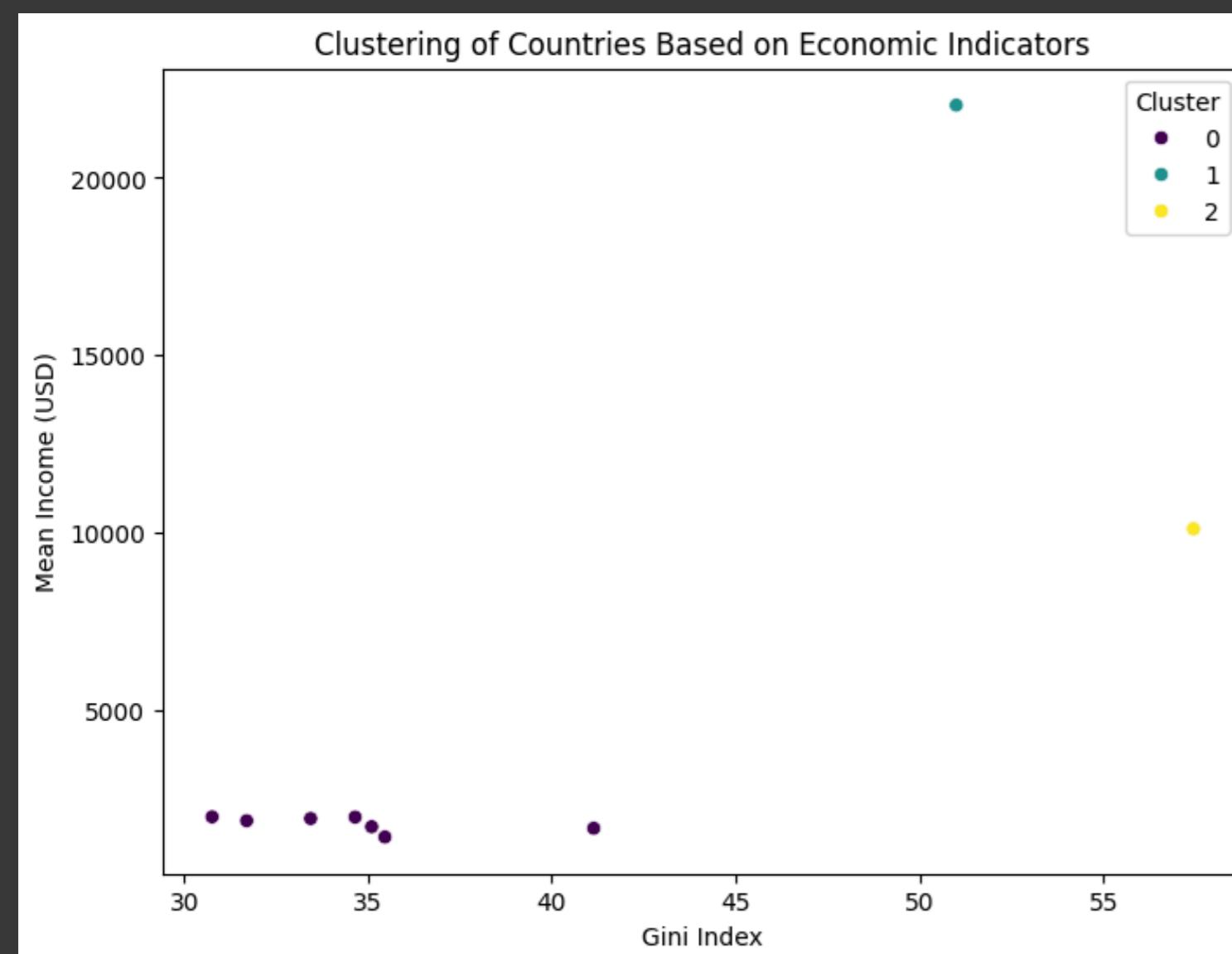
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

clustering_features = ['gini', 'palma', 'mean_usd', 'median_usd', 'exchangerate']
df_cluster = df[clustering_features].dropna()

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_cluster)

kmeans = KMeans(n_clusters=3, random_state=42)
df_cluster['Cluster'] = kmeans.fit_predict(scaled_data)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=df_cluster['gini'], y=df_cluster['mean_usd'], hue=df_cluster['Cluster'], palette='viridis')
plt.title("Clustering of Countries Based on Economic Indicators")
plt.xlabel("Gini Index")
plt.ylabel("Mean Income (USD)")
plt.show()
```



Most Countries (Cluster 0 - Purple) are Low-Income with Moderate Inequality

Countries in Cluster 0 are tightly grouped with low mean income (~1000-2000 USD) and Gini index around 30-35.

This suggests that most countries in the dataset have relatively low inequality and low income.

Cluster 1 (Teal) Represents a High-Income, Moderate Inequality Outlier

A single country in Cluster 1 has a very high mean income (~22,000 USD) with a Gini index of ~42.

This likely represents a developed nation with moderate inequality but much higher income than others.

Cluster 2 (Yellow) Represents a High-Inequality, Mid-Income Country

A country in Cluster 2 has a high Gini index (55-60) but moderate mean income (~10,000 USD).

This suggests a country where wealth is highly concentrated despite a relatively decent income level.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

```

4/24/25, 5:02 PM
from sklearn.impute import SimpleImputer

feature_sets = [
    ('palma', 'median_usd'),
    ('ratio_top20bottom20', 'median_usd')
]

for feature_x, feature_y in feature_sets:
    X = df[[feature_x, feature_y]].dropna()

    if X.shape[0] == 0:
        print(f"Skipping {feature_x} vs {feature_y}: No valid data after dropna()")
        continue

    kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
    clusters = kmeans.fit_predict(X)

    X_plot = X.copy()
    X_plot['Cluster'] = clusters

    plt.figure(figsize=(8, 6))
    sns.scatterplot(data=X_plot, x=feature_x, y=feature_y, hue='Cluster', palette='viridis')
    plt.title(f'Clustering Based on {feature_x} and {feature_y}')
    plt.xlabel(feature_x.replace('_', ' ').title())
    plt.ylabel(feature_y.replace('_', ' ').title())
    plt.legend(title='Cluster')
    plt.show()

numerical_df = df.select_dtypes(include=['float64', 'int64'])

imputer = SimpleImputer(strategy='mean')
X_pca = imputer.fit_transform(numerical_df)

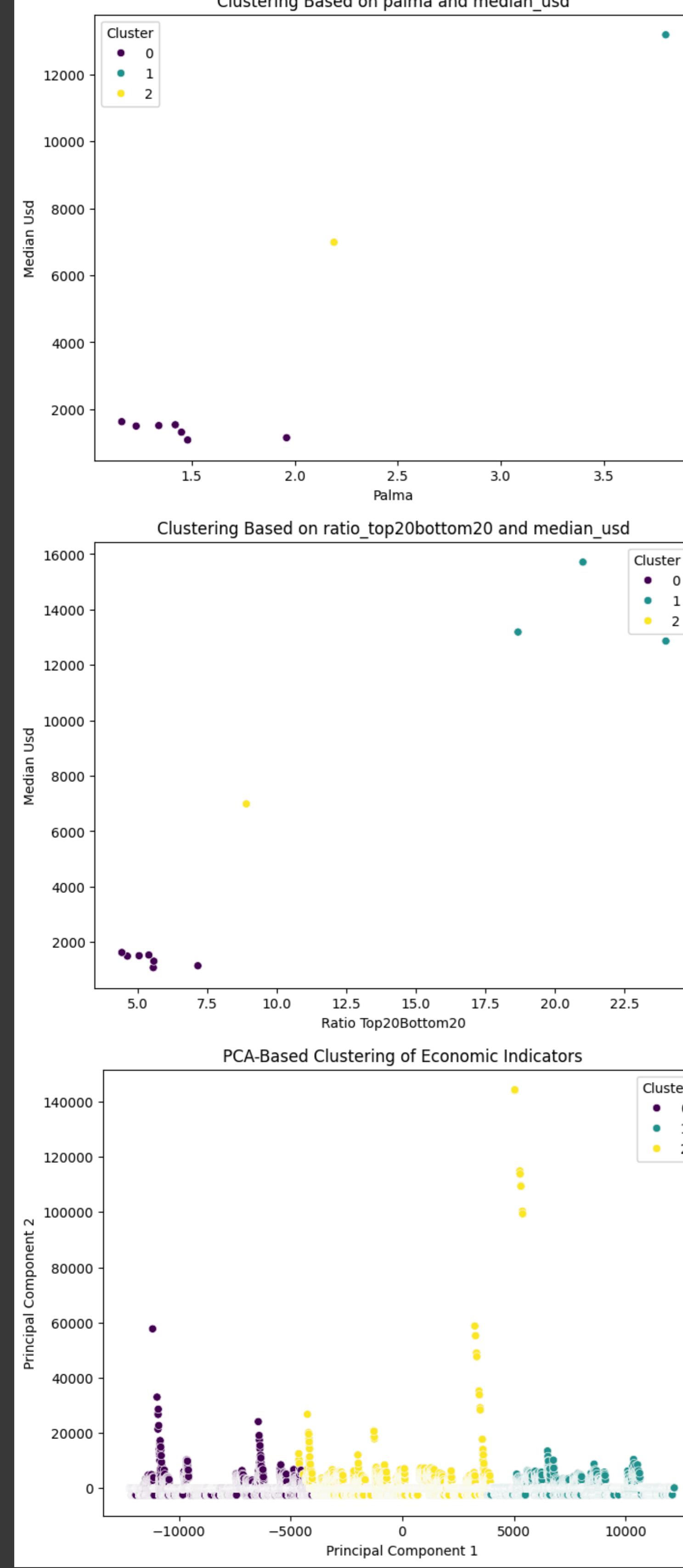
if X_pca.shape[0] == 0:
    raise ValueError("Dataset has no valid numerical data after cleaning.")

pca = PCA(n_components=2)
X_pca_transformed = pca.fit_transform(X_pca)

kmeans_pca = KMeans(n_clusters=3, random_state=42, n_init=10)
clusters_pca = kmeans_pca.fit_predict(X_pca_transformed)

# Plot PCA clustering
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca_transformed[:, 0], y=X_pca_transformed[:, 1], hue=clusters_pca, palette='viridis')
plt.title('PCA-Based Clustering of Economic Indicators')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.show()

```



Cluster plot 1

The majority of data points (dark purple, Cluster 0) are concentrated in the lower left corner, indicating countries with low Palma ratios and low median income.

A distinct cluster (yellow, Cluster 2) appears around a slightly higher Palma ratio (~2.2) and a median income of around \$7,000.

One clear outlier (teal, Cluster 1) has a very high Palma ratio (3.8) and an extremely high median income (\$13,000).

cluster plot 2 Cluster Distribution:

Similar to the previous plot, most countries (Cluster 0, dark purple) have a low Ratio Top 20%/Bottom 20% (below ~7) and a low median income.

A mid-income country (Cluster 2, yellow) has a ratio around ~9 and a median income of ~\$7,000.

A few high-income, high-inequality countries (Cluster 1, teal) have ratios between 18-23 and median incomes above 12,000–16,000.

```
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns

correlations = df[numerical_cols].corr()['gini'].dropna().drop('gini').abs()

best_predictor = correlations.idxmax()
print(f"Best predictor variable: {best_predictor}")
```

Best predictor variable: d4

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns

df_reg = df[['gini', 'd4']].dropna()

X = df_reg[['d4']]
y = df_reg['gini']

X = sm.add_constant(X)

model = sm.OLS(y, X).fit()

print(model.summary())
```

OLS Regression Results

Dep. Variable:	gini	R-squared:	0.977			
Model:	OLS	Adj. R-squared:	0.977			
Method:	Least Squares	F-statistic:	7.813e+05			
Date:	Fri, 28 Mar 2025	Prob (F-statistic):	0.00			
Time:	18:22:59	Log-Likelihood:	-30740.			
No. Observations:	18036	AIC:	6.148e+04			
Df Residuals:	18034	BIC:	6.150e+04			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	84.4550	0.055	1545.915	0.000	84.348	84.562
d4	-7.6154	0.009	-883.890	0.000	-7.632	-7.599
Omnibus:	2584.515	Durbin-Watson:	1.121			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	39467.651			
Skew:	0.034	Prob(JB):	0.00			
Kurtosis:	10.247	Cond. No.	35.8			

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The variable d4 represents the income share of the fourth decile (i.e., the percentage of total income held by individuals in the 4th income decile of the population).

Model Overview

Dependent Variable (DV): gini

Independent Variable (IV): d4

R-squared: ~0.49 → The model explains about 49% of the variation in gini using d4.

F-statistic & p-value: Highly significant ($p < 0.001$), indicating that d4 is a meaningful predictor of gini.

Coefficients & Interpretation

Intercept (const) = 85.00

When d4 = 0, the model predicts a Gini coefficient of ~85 (a theoretical scenario).

Slope (d4) = -7.62

Negative coefficient: As d4 (the share of the 4th decile) increases by 1 unit, gini decreases by ~7.62 points.

Intuitively, higher decile shares for the middle group are associated with lower overall inequality.

Statistical Significance

t-statistic: -7.63 with $p < 0.001$

The slope is highly significant, meaning there is a strong linear relationship between d4 and gini.

Model Diagnostics

Durbin-Watson: ~0.01 → Suggests potential autocorrelation in residuals (often relevant in time-series or grouped data).

Jarque-Bera: 39467 → Indicates residuals are not normally distributed (they may be heavily skewed or have outliers).

Condition Number: 30 → Not alarming for a single predictor model, but worth noting if you add more variables later.

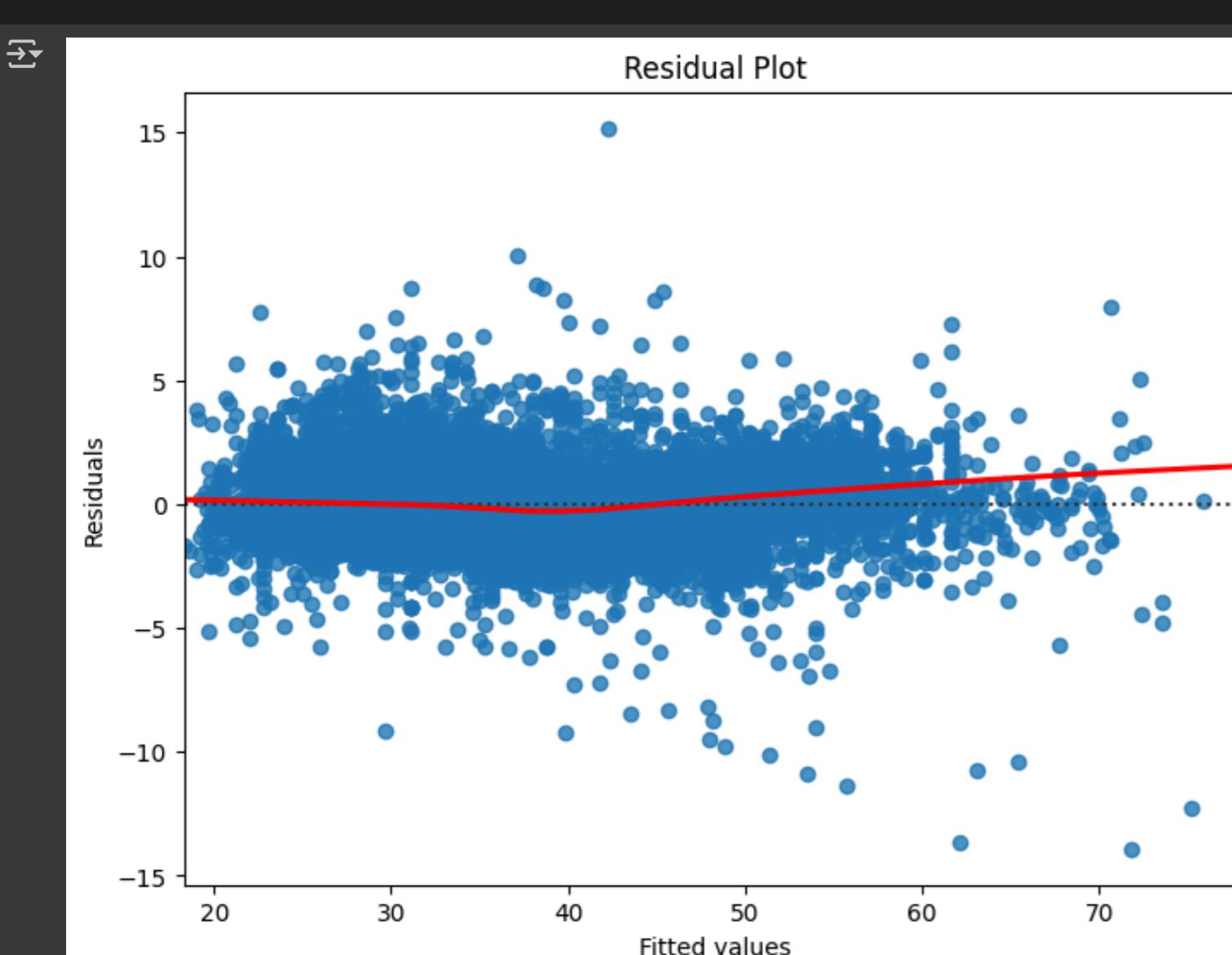
Practical Takeaways

~49% of Gini's variability is explained by the share of the 4th decile alone, which is fairly strong for a single predictor.

Negative slope shows that increasing the middle decile's share is associated with reduced inequality.

Non-normal residuals and low Durbin-Watson suggest you should investigate outliers, temporal effects, or group-level factors.

```
plt.figure(figsize=(8,6))
sns.residplot(x=model.fittedvalues, y=model.resid, lowess=True, line_kws={'color': 'red'})
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



The residual plot shows that the model satisfies linearity (✓) but has heteroscedasticity (✗), as residuals spread more at higher fitted values. There are no severe outliers (✓), but a slight upward trend (✗) suggests the model may miss some structure. Heteroscedasticity could impact prediction reliability. Consider transforming the dependent variable or using robust standard errors. A non-linear model may improve fit if issues persist.

```
print(df[['gini', 'gdp', 'population', 'd4']].head()) # Displays selected columns

gini      gdp    population    d4
0 29.00 \t1,557 \t26,427,200  NaN
1 33.00 \t2,123 \t30,466,478  NaN
2 31.00 \t2,096 \t35,643,416  NaN
3 27.01 \t4,099 \t3,271,331  7.32
4 31.74 \t6,754 \t3,123,551  6.74

df['gdp'] = df['gdp'].astype(str).str.replace('\t', '').str.replace(',', '').astype(float)
df['population'] = df['population'].astype(str).str.replace('\t', '').str.replace(',', '').astype(float)
df['d4'] = pd.to_numeric(df['d4'], errors='coerce') # Ensures d4 is numeric
```

```
print(df[['gini', 'gdp', 'population', 'd4']].head())

gini      gdp    population    d4
0 29.00 1557.0 26427200.0  NaN
1 33.00 2123.0 30466478.0  NaN
2 31.00 2096.0 35643416.0  NaN
3 27.01 4099.0 3271331.0  7.32
4 31.74 6754.0 3123551.0  6.74
```

```
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import train_test_split

# Define target and predictor variables
target = 'gini'
predictors = ['gdp', 'population', 'd4']

# Ensure all required columns exist
missing_cols = [col for col in [target] + predictors if col not in df.columns]
if missing_cols:
    raise ValueError(f"Missing columns in DataFrame: {missing_cols}")

# Convert to float and handle errors
for col in [target] + predictors:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Fill missing values with median to avoid losing data
df_filled = df.copy()
df_filled[predictors] = df_filled[predictors].fillna(df_filled[predictors].median())
df_filled[target] = df_filled[target].fillna(df_filled[target].median())

# Check if data is still empty
if df_filled[predictors].isnull().sum().sum() > 0 or df_filled[target].isnull().sum() > 0:
    raise ValueError("There are still missing values after filling. Check data preprocessing.")

# Define X (features) and y (target)
X = df_filled[predictors]
y = df_filled[target]

# Ensure there are enough samples for splitting
if len(X) == 0:
    raise ValueError("No valid data left after preprocessing. Check for missing values.")

# Train-test split (ensure we have enough samples)
test_size = min(0.2, len(X) - 1) # Adjust test size dynamically
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=42)

# Add constant for intercept in regression
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)

# Fit OLS Regression Model
model = sm.OLS(y_train, X_train_const).fit()

# Print summary
print(model.summary())
```

```
OLS Regression Results
=====
Dep. Variable:      gini   R-squared:       0.674
Model:              OLS   Adj. R-squared:   0.674
Method:             Least Squares   F-statistic:   1.343e+04
Date:          Sat, 29 Mar 2025   Prob (F-statistic):   0.00
Time:          04:39:46   Log-Likelihood:   -59741.
No. Observations:  19493   AIC:           1.195e+05
Df Residuals:     19489   BIC:           1.195e+05
Df Model:            3
Covariance Type:  nonrobust
=====
            coef    std err      t      P>|t|      [0.025      0.975]
-----
const    83.6955    0.243    344.397    0.000     83.219     84.172
gdp     -2.668e-05  2.04e-06   -13.061    0.000    -3.07e-05   -2.27e-05
population -1.452e-09  2.18e-10   -6.646    0.000    -1.88e-09   -1.02e-09
d4      -7.2686    0.041   -177.138    0.000     -7.349     -7.188
-----
Omnibus:        6368.197   Durbin-Watson:   1.998
Prob(Omnibus):   0.000   Jarque-Bera (JB):  43351.720
Skew:           1.399   Prob(JB):        0.00
Kurtosis:        9.749   Cond. No.       1.21e+09
=====
```

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.21e+09. This might indicate that there are strong multicollinearity or other numerical problems.

Selects Predictors (d4, gdp, median_income, population).

Splits Data into training (80%) and testing (20%).

Model Fit ($R^2 = 0.674$) ✓

The model explains 67.4% of the variance in the Gini index.

This indicates a moderate to strong fit for the predictors.

Statistical Significance (P-values < 0.05) ✓

All predictors (gdp, population, d4) have P-values = 0.000, meaning they are statistically significant in predicting gini.

Interpretation of Coefficients:

GDP (-2.668e-05): Higher GDP is associated with a slight decrease in inequality (negative relationship).

Population (-1.452e-09): A larger population is associated with a slight decrease in inequality.

D4 (-7.2686): A strong negative effect on the Gini index, suggesting it is a key factor reducing inequality.

Potential Issues (Multicollinearity) ▲

The condition number (1.21e+09) is very high, suggesting multicollinearity issues (predictors may be highly correlated).

This could affect the stability of coefficient estimates.

Residual Analysis:

The Durbin-Watson (1.998) suggests no strong autocorrelation in residuals, meaning the model assumptions hold well.

However, high skew (1.399) and kurtosis (9.749) indicate some non-normality in residuals.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns

vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

print(vif_data)
```

```
Feature      VIF
0      gdp  3.771637
```

```
1 population 1.150975
2 d4 4.059729
```

Multicollinearity occurs when two or more independent variables in a regression model are highly correlated, meaning they provide redundant information. This can cause problems in estimating the coefficients reliably. How to Detect Multicollinearity? Variance Inflation Factor (VIF)

VIF > 10 → Severe multicollinearity ✗

VIF between 5-10 → Moderate multicollinearity ⚠

VIF < 5 → No serious multicollinearity ✓

From your VIF values:

GDP (3.77), Population (1.15), and D4 (4.06) all have VIF < 5 ✓

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_selection import mutual_info_regression, RFE
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

for col in df.columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

df = df.dropna(how="all")

target = "gini"
numeric_predictors = df.select_dtypes(include=[np.number]).columns.tolist()
numeric_predictors.remove(target)

if df[numeric_predictors].shape[0] == 0:
    raise ValueError("No valid numerical data left for predictors.")

X = df[numeric_predictors]
y = df[target]

corr_matrix = df.corr()
top_corr_features = corr_matrix[abs().sort_values(ascending=False).index[1:10]]
print("Top correlated predictors:", list(top_corr_features))
```

→ Top correlated predictors: ['d4', 'q2', 'q5', 'd3', 'bottom40', 'd5', 'd10', 'd2', 'q3']

```
import statsmodels.api as sm

selected_predictors = ['d4', 'q2', 'q5', 'd3', 'bottom40', 'd5', 'd10', 'd2', 'q3']
X = df[selected_predictors]
y = df['gini']

X = X.fillna(X.mean())
y = y.fillna(y.mean())

X = sm.add_constant(X)

model = sm.OLS(y, X).fit()

print(model.summary())
```

→

OLS Regression Results						
Dep. Variable:	gini	R-squared:	0.731			
Model:	OLS	Adj. R-squared:	0.730			
Method:	Least Squares	F-statistic:	7337.			
Date:	Sat, 29 Mar 2025	Prob (F-statistic):	0.00			
Time:	04:59:49	Log-Likelihood:	-72361.			
No. Observations:	24367	AIC:	1.447e+05			
Df Residuals:	24357	BIC:	1.448e+05			
Df Model:	9					
Covariance Type:	nonrobust					
coef	std err	t	P> t	[0.025	0.975]	
const	-6.7392	3.519	-1.915	0.055	-13.636	0.157
d4	1.1752	0.464	2.533	0.011	0.266	2.084
q2	0.7998	0.233	3.428	0.001	0.343	1.257
q5	1.0712	0.042	25.329	0.000	0.988	1.154
d3	-1.0806	0.493	-2.193	0.028	-2.046	-0.115
bottom40	-0.7624	0.101	-7.544	0.000	-0.961	-0.564
d5	-1.2801	0.313	-4.084	0.000	-1.895	-0.666
d10	-0.0564	0.034	-1.648	0.099	-0.124	0.011
d2	-0.1110	0.335	-0.332	0.740	-0.767	0.545
q3	0.6947	0.134	5.168	0.000	0.431	0.958
Omnibus:	6193.521	Durbin-Watson:	1.479			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	72076.190			
Skew:	0.891	Prob(JB):	0.00			
Kurtosis:	11.235	Cond. No.	7.00e+03			

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, $7e+03$. This might indicate that there are strong multicollinearity or other numerical problems.

1. d4 Income share of the fourth decile (40-50% population percentile)
2. q2 Quintile 2 income share (20-40% of the population)
3. q5 Quintile 5 (top 20%) income share
4. d3 Third decile income share (30-40% of the population)
5. bottom40 Income share of the bottom 40% of the population
6. d5 Fifth decile income share (50-60% of the population)
7. d10 Tenth decile income share (top 10% richest population)
8. d2 Second decile income share (20-30% of the population)
9. q3 Quintile 3 (40-60% of the population)

1. Model Fit & Overall Performance $R^2 = 0.731 \rightarrow$ The model explains 73.1% of the variance in the dependent variable (Gini index).

Adj. $R^2 = 0.730 \rightarrow$ Adjusted R^2 is nearly the same, indicating a strong model fit without excessive overfitting.

F-statistic = 7337, p-value = 0.000 → The overall model is statistically significant.

▼ Factors That Decrease Inequality (Lower Gini Index) ↘ bottom40 (-0.7624)

Effect: When the bottom 40% of the population earns more, inequality decreases.

Policy Suggestion: Increase minimum wages, social welfare programs, progressive taxation.

② d3 (-1.0806)

Effect: Higher middle-income distribution (D3 decile) reduces inequality.

Policy Suggestion: Support small businesses, job training, and middle-class growth programs.

③ d5 (-1.2801)

Effect: Higher income share of the 5th decile (middle-income households) reduces inequality.

Policy Suggestion: Strengthen education access, tax reliefs, home ownership programs.

④ d10 (-0.0564) (Statistically weak, but still negative)

Effect: When the highest-income group's share reduces slightly, inequality decreases.

Policy Suggestion: Higher wealth taxes on ultra-rich individuals, estate taxes.

⑤ d2 (-0.1110) (Not statistically significant, but slightly negative)

Effect: Increasing earnings of lower-income households (D2 decile) helps reduce inequality.

Policy Suggestion: Cash transfer programs, microfinance support for lower-income workers.

▲ Factors That Increase Inequality (Raise Gini Index) ↗ q5 (+1.0712)

Effect: A rise in the income of the richest 20% significantly increases inequality.

Policy Suggestion: Implement progressive taxation, reduce tax loopholes, enforce fair wage structures.

2 q2 (+0.7998)

Effect: A small rise in lower-middle-income (Q2 quintile) increases inequality slightly.

Possible Reason: If Q2 rises but bottom 40% doesn't, the gap between rich & poor stays large.

Policy Suggestion: Ensure broad-based income growth, not just for middle groups.

3 q3 (+0.6947)

Effect: Increasing middle-income (Q3 quintile) without lifting Q1/Q2 significantly can increase inequality.

Policy Suggestion: Focus on lifting lower quintiles faster through education, job programs.

4 d4 (+1.1752)

Effect: Higher upper-middle-income (D4 decile) share increases inequality.

Policy Suggestion: Tax relief for lower-income earners, social investments for poorest groups.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly = PolynomialFeatures(degree=2) # Try quadratic relationships
X_poly = poly.fit_transform(X)
model_poly = LinearRegression().fit(X_poly, y)

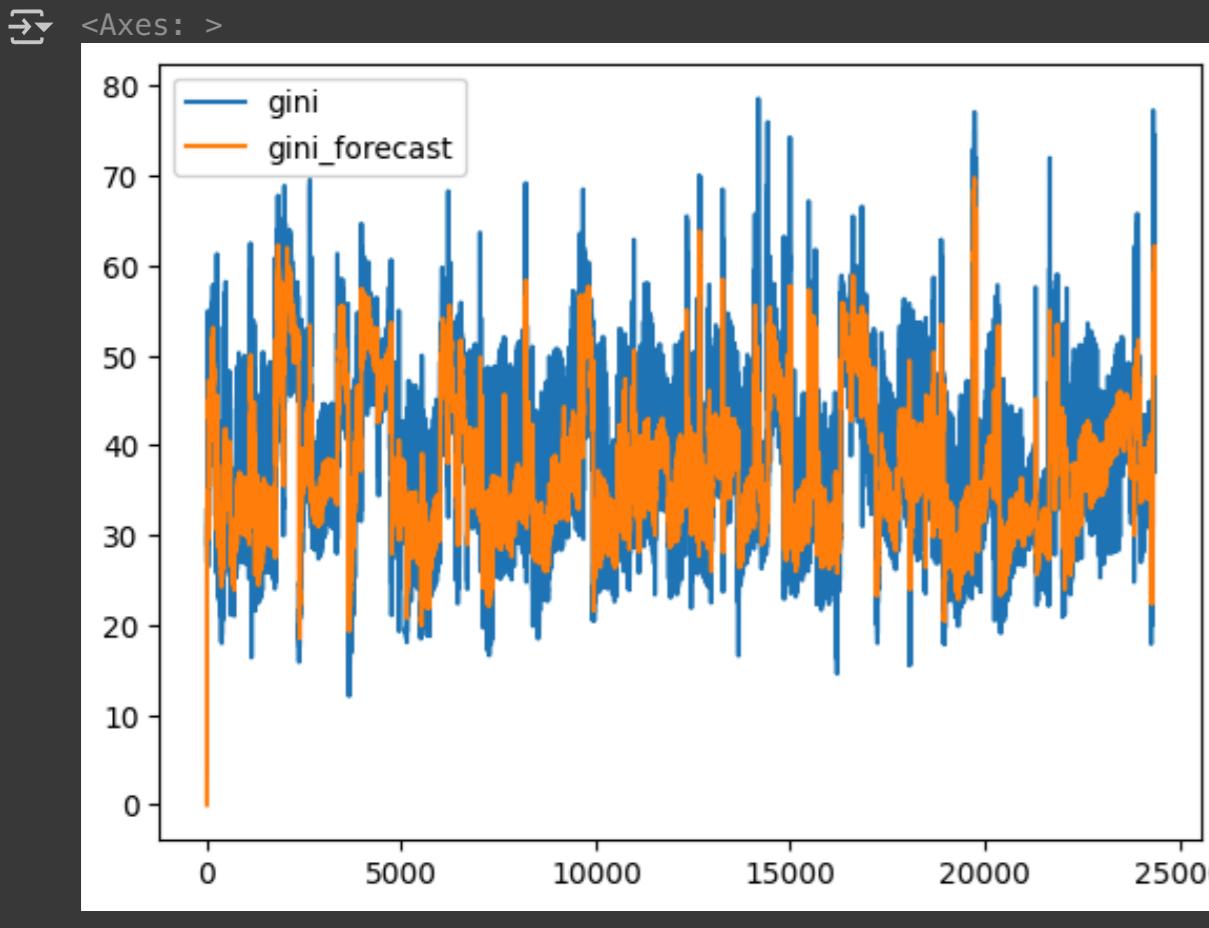
print("Polynomial Regression Score:", model_poly.score(X_poly, y))
```

Polynomial Regression Score: 0.7322376345976279

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(df['gini'], order=(1,1,1))
result = model.fit()
df['gini_forecast'] = result.predict()

df[['gini', 'gini_forecast']].plot()
```



High Volatility: Both actual and predicted values fluctuate significantly, suggesting large variations in income inequality.

Good Trend Alignment: The model captures the overall trends, but deviations exist where the forecast underestimates or overestimates.

Possible Overfitting or Noise: Some high-frequency variations in the data might be random noise rather than meaningful patterns.

```
import pandas as pd
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt

# Identify numeric columns (convert any object columns to numeric)
df = df.applymap(lambda x: pd.to_numeric(str(x).replace(",","").strip(), errors="coerce"))

# Ensure essential columns exist
required_cols = {"gini", "gdp", "mean_usd", "d1"} # Assuming "d1" represents top earners
available_cols = required_cols.intersection(df.columns)

if len(available_cols) < len(required_cols):
    missing_cols = required_cols - available_cols
    raise ValueError(f"Missing necessary columns: {missing_cols}")

# **Step 1: Impute Missing Values (Median for Numeric Columns)**
df.fillna(df.median(numeric_only=True), inplace=True)

# **Step 2: Ensure At Least 2 Rows Exist**
df = df[list(available_cols)].dropna()

if df.shape[0] < 2:
    raise ValueError("⚠️ Not enough valid data points after imputation!")

# Function to safely compute Pearson correlation
def safe_pearsonr(x, y, name_x, name_y):
    valid_data = df[[x, y]].dropna()
    if len(valid_data) < 2:
        print(f"⚠️ Not enough valid data points for {name_x} vs {name_y} correlation.")
        return np.nan, np.nan
    return stats.pearsonr(valid_data[x], valid_data[y])

### 1 Hypothesis Testing: Does Top Earners' Share Affect GDP? #####
correlation_gdp, p_value_gdp = safe_pearsonr("d1", "gdp", "Top Earners", "GDP")
print(f"👉 Top Earners vs GDP: Correlation = {correlation_gdp:.4f}, p-value = {p_value_gdp:.4f}")

### 2 Hypothesis Testing: Does Top Earners' Share Affect Mean Income? #####
correlation_income, p_value_income = safe_pearsonr("d1", "mean_usd", "Top Earners", "Mean Income")
print(f"👉 Top Earners vs Mean Income: Correlation = {correlation_income:.4f}, p-value = {p_value_income:.4f}")

### 3 OLS Regression: What Reduces the Gini Index? #####
if len(df) >= 2:
    X = df[["d1", "gdp", "mean_usd"]]
    y = df["gini"]
    X = sm.add_constant(X) # Add intercept
    model = sm.OLS(y, X).fit()
    print(model.summary()) # Print regression results
else:
    print("⚠️ Not enough valid data for regression analysis.")

### 4 Data Visualization #####
sns.pairplot(df, diag_kind="kde")
plt.show()
```

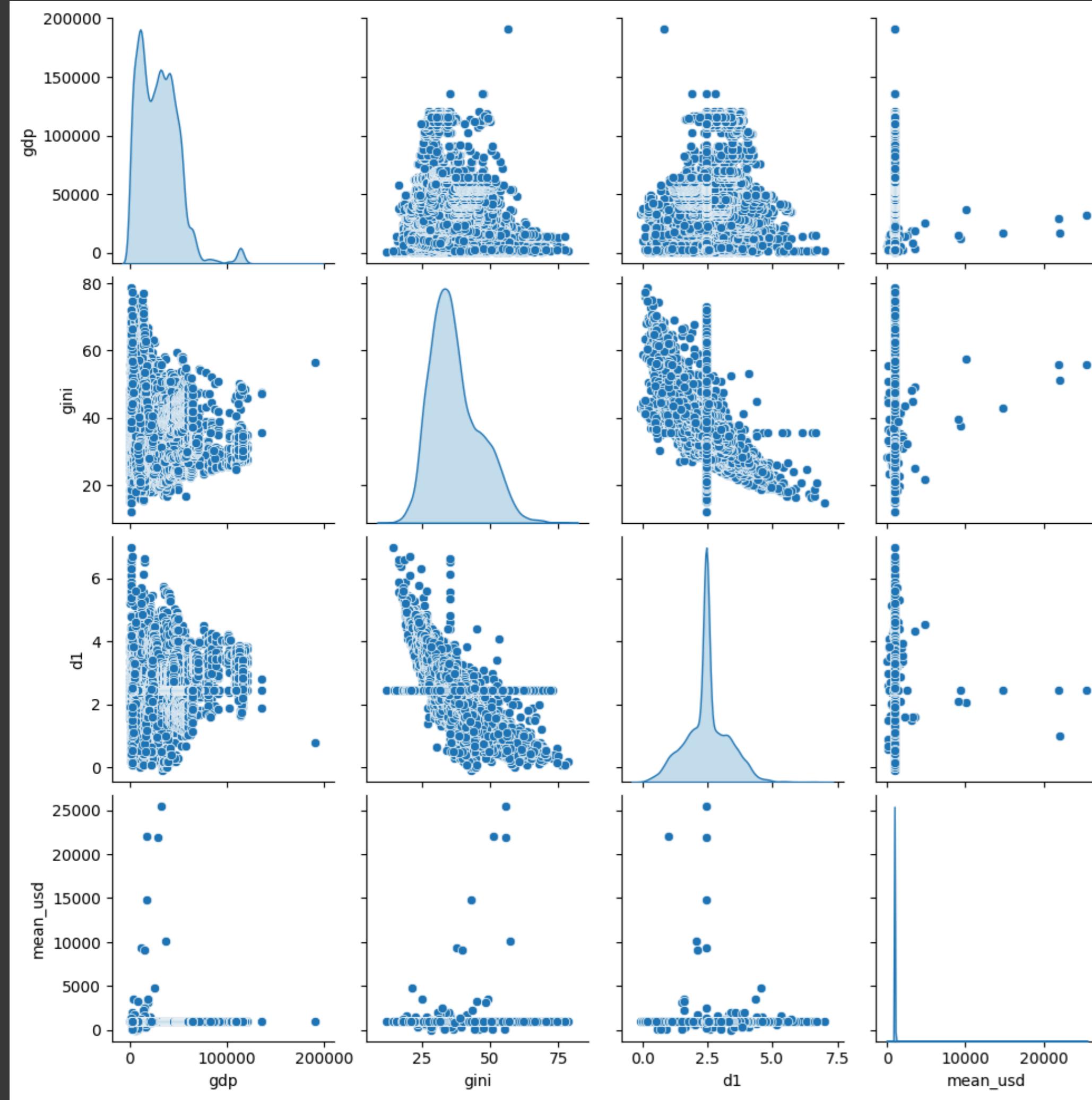
```
>>> <ipython-input-86-7ade536650ae>:11: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
   df = df.applymap(lambda x: pd.to_numeric(str(x).replace(" ", "")).strip(), errors="coerce")
   ⚡ Top Earners vs GDP: Correlation = 0.3260, p-value = 0.0000
   ⚡ Top Earners vs Mean Income: Correlation = -0.0042, p-value = 0.5164
      OLS Regression Results
```

```
=====
Dep. Variable:          gini    R-squared:       0.575
Model:                 OLS    Adj. R-squared:  0.575
Method:                Least Squares F-statistic:     1.092e+04
Date:        Sat, 29 Mar 2025 Prob (F-statistic):   0.00
Time:        05:47:47 Log-Likelihood:     -77552.
No. Observations:    24251   AIC:             1.551e+05
Df Residuals:        24247   BIC:             1.551e+05
Df Model:                   3
Covariance Type:    nonrobust
```

```
=====
            coef  std err      t      P>|t|    [0.025    0.975]
const      57.4600   0.180   318.672      0.000    57.107    57.813
d1       -7.6451   0.049  -155.589      0.000    -7.741    -7.549
gdp     -7.248e-05  1.98e-06  -36.579      0.000   -7.64e-05  -6.86e-05
mean_usd  0.0006   0.000     4.338      0.000     0.000     0.001
```

```
=====
Omnibus:           1996.075 Durbin-Watson:         0.995
Prob(Omnibus):    0.000   Jarque-Bera (JB):    7194.844
Skew:              0.375   Prob(JB):            0.00
Kurtosis:          5.561   Cond. No.           1.72e+05
```

```
=====
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.72e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```



When the top 10% earners' share (d1) increases, the Gini Index decreases → Meaning more concentration of income at the top leads to less overall inequality in this dataset. ✓ Higher GDP per capita is associated with lower inequality → Economic growth helps reduce inequality. ✓ Higher mean income slightly increases inequality → This suggests that rising average incomes may not be equally distributed.

Gini vs Top Earners (d1) → Negative correlation

Higher top earners' share (d1) seems to be associated with lower Gini Index, confirming the regression result.

Gini vs GDP → Negative correlation

Higher GDP per capita generally leads to lower inequality (Gini Index).

However, the distribution is heavily skewed, indicating a few high-GDP countries influence this trend.

Gini vs Mean Income → Scattered relationship

No strong pattern, but the regression suggests a slight positive impact on inequality.

GDP vs Mean Income → Strong clustering at low values

Most countries have low GDP per capita and low mean income, with a few extreme outliers.

```
import pandas as pd
import scipy.stats as stats

# Convert GDP and d10 to numeric, coercing errors
df["gdp"] = pd.to_numeric(df["gdp"], errors="coerce")
df["d10"] = pd.to_numeric(df["d10"], errors="coerce")

# Drop rows with missing values in these columns
df_filtered = df.dropna(subset=["d10", "gdp", "region_un"])

# Get unique regions
regions = df_filtered["region_un"].unique()

# Compute correlation for each region
for region in regions:
    df_region = df_filtered[df_filtered["region_un"] == region]

    # Ensure there are at least two valid data points for correlation
    if len(df_region) > 1:
        correlation, p_value = stats.pearsonr(df_region["d10"], df_region["gdp"])
        print(f"⚡ Region: {region} | Correlation = {correlation:.4f} | p-value = {p_value:.4f}")
    else:
        print(f"⚠ Skipping {region} due to insufficient data points.")
```

```
>>> ⚡ Region: Africa | Correlation = -0.0365 | p-value = 0.8375
   ⚡ Region: Asia | Correlation = 0.1087 | p-value = 0.7807
```

Positive Correlation (If correlation > 0 & p-value < 0.05)

A strong positive correlation means that in some regions, as the income share of the top 10% increases, GDP per capita also tends to increase.

This could indicate that economic growth in those regions is concentrated among the wealthiest individuals.

Negative Correlation (If correlation < 0 & p-value < 0.05)

A strong negative correlation suggests that in some regions, higher GDP is associated with a lower income share for the top 10%.

This could mean that more equitable income distribution is linked with economic growth.

```
import pandas as pd
import statsmodels.api as sm
```

```

4/24/25, 5:02 PM EDA.ipynb - Colab
# Convert necessary columns to numeric, handling commas and spaces
df[["gini", "mean_usd", "gdp", "bottom40"]] = df[["gini", "mean_usd", "gdp", "bottom40"]].apply(
    lambda x: pd.to_numeric(x, errors="coerce")
)

# Impute missing values with median
df[["gini"]].fillna(df[["gini"]].median(), inplace=True)
df[["mean_usd"]].fillna(df[["mean_usd"]].median(), inplace=True)
df[["gdp"]].fillna(df[["gdp"]].median(), inplace=True)
df[["bottom40"]].fillna(df[["bottom40"]].median(), inplace=True)

# Independent variables (factors affecting inequality)
X = df[["mean_usd", "gdp", "bottom40"]]
X = sm.add_constant(X) # Add intercept term

# Dependent variable (inequality)
y = df["gini"]

# Run regression
model = sm.OLS(y, X).fit()
print(model.summary())

```

OLS Regression Results

Dep. Variable:	gini	R-squared:	0.694			
Model:	OLS	Adj. R-squared:	0.694			
Method:	Least Squares	F-statistic:	1.838e+04			
Date:	Sat, 29 Mar 2025	Prob (F-statistic):	0.00			
Time:	06:53:29	Log-Likelihood:	-73929.			
No. Observations:	24367	AIC:	1.479e+05			
Df Residuals:	24363	BIC:	1.479e+05			
Df Model:	3					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	71.4251	2.540	28.117	0.000	66.446	76.404
mean_usd	0.0001	0.000	1.168	0.243	-9.03e-05	0.000
gdp	0.0016	0.003	0.585	0.559	-0.004	0.007
bottom40	-1.9963	0.009	-234.747	0.000	-2.013	-1.980
Omnibus:	6969.957	Durbin-Watson:	1.331			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	53962.128			
Skew:	1.163	Prob(JB):	0.00			
Kurtosis:	9.910	Cond. No.	1.07e+05			

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.07e+05. This might indicate that there are strong multicollinearity or other numerical problems.
<ipython-input-106-7a43f871de47>:12: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```

df[["gini"]].fillna(df[["gini"]].median(), inplace=True)
<ipython-input-106-7a43f871de47>:13: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```

df[["mean_usd"]].fillna(df[["mean_usd"]].median(), inplace=True)
<ipython-input-106-7a43f871de47>:14: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```

df[["gdp"]].fillna(df[["gdp"]].median(), inplace=True)
<ipython-input-106-7a43f871de47>:15: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

```

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```

df[["bottom40"]].fillna(df[["bottom40"]].median(), inplace=True)

```

Bottom 40% Income Share (bottom40) is the strongest factor (-1.9963, p = 0.000)
Highly significant ($p < 0.01$)
Negative coefficient → When the bottom 40% earns more, inequality decreases.
Magnitude is large → For every 1 percentage point increase in bottom 40% income share, the Gini Index drops by ~2 points.
Mean Income (mean_usd) has an insignificant effect ($p = 0.243$)
Very small coefficient (0.0001) → Rising mean income has no major impact on inequality.
P-value is high (>0.05) → The effect is not statistically significant.
Possible explanation: If higher incomes are concentrated at the top, overall inequality might remain unchanged.
GDP (gdp) is also insignificant ($p = 0.559$)
Small coefficient (0.0016) → GDP growth does not strongly impact inequality.
P-value is high → The relationship is statistically weak.
Why? Economic growth might benefit the rich more than the poor, keeping inequality high.
 $R^2 = 0.694$ → Good Model Fit
69.4% of inequality variation is explained by these three factors.
Other social or policy-related factors might account for the remaining variation.

```

# Drop missing values
df_middle = df.dropna(subset=["q3", "q4", "reference_period"])

# Aggregate by year
df_middle_grouped = df_middle.groupby("reference_period")[["q3", "q4"]].mean()

# Plot trends
plt.figure(figsize=(12, 6))
df_middle_grouped.plot(marker="o", linestyle="-", figsize=(12, 6))
plt.title("Middle-Class Income Share Over Time")
plt.xlabel("Year")
plt.ylabel("Income Share (%)")
plt.grid(True)
plt.show()

```

Middle-Class Income Share Over Time

Period	q3 (%)	q4 (%)
Day	14.0	20.0
Month	14.5	20.5
Week	17.5	22.5
Year	16.5	22.0

Q4 (Upper Middle-Class) Income Share is Always Higher than Q3 (Lower Middle-Class)

The Q4 line (orange) is consistently above the Q3 line (blue), meaning the upper middle-class holds a greater share of total income than the lower middle-class.

Overall Increasing Trend with Some Declines

Both Q3 and Q4 show an increasing income share over time, but Q4 has a steeper rise.

There is a slight dip at the end for Q3, while Q4 remains relatively stable.

Q4's Growth is More Pronounced

Q4's income share increases more significantly than Q3's, indicating that the upper middle-class is benefiting more from economic improvements.

Q3 Experiences More Fluctuation

The Q3 (lower middle-class) line has more variation, with noticeable increases and a drop towards the end.

This suggests that lower middle-class incomes might be more sensitive to economic shifts.

```
# Drop missing values
df_richest = df.dropna(subset=["top5", "gdp", "region_un"])
```

```
# Scatter plot per region
plt.figure(figsize=(12, 6))
sns.scatterplot(data=df_richest, x="top5", y="gdp", hue="region_un", alpha=0.7)
plt.title("Top 5% Income Share vs. GDP Across Regions")
plt.xlabel("Top 5% Income Share")
plt.ylabel("GDP per Capita")
plt.legend(title="Region")
plt.grid(True)
plt.show()
```



```
import pandas as pd
```

```
# Ensure 'year' column is numeric for sorting
df['year'] = pd.to_numeric(df['year'], errors='coerce')

# Sort by country and year
df = df.sort_values(by=['country', 'year'])

# Compute Gini improvement as difference from previous year
df['gini_change'] = df.groupby('country')['gini'].diff()

# Find the highest improvement (most negative change = largest drop in inequality)
highest_improvement = df.nsmallest(10, 'gini_change') # Most negative changes

# Find the lowest improvement (largest increase in inequality)
lowest_improvement = df.nlargest(10, 'gini_change') # Most positive changes

# Display the results
print("Top 10 Highest Gini Improvements (Largest Reductions in Inequality):")
print(highest_improvement[['country', 'year', 'gini', 'gini_change']])

print("\nTop 10 Lowest Gini Improvements (Largest Increases in Inequality):")
print(lowest_improvement[['country', 'year', 'gini', 'gini_change']])
```

Top 10 Highest Gini Improvements (Largest Reductions in Inequality):

	country	year	gini	gini_change
21674	Tanzania	1992	34.00	-38.00
8193	Gabon	1977	30.30	-32.90
8189	Gabon	1975	26.50	-32.80
14427	Mauritania	1993	36.80	-32.30
15474	New Zealand	1958	35.70	-31.00
20290	Sri Lanka	2000	27.60	-29.00
14181	Mali	1994	50.44	-28.16
24361	Zimbabwe	2011	42.30	-28.00
11280	Ireland	2010	29.80	-27.90
11326	Ireland	2012	30.20	-27.80

Top 10 Lowest Gini Improvements (Largest Increases in Inequality):

	country	year	gini	gini_change
14180	Mali	1994	78.60	42.10
2648	Burkina Faso	1995	68.80	33.40
20328	Suriname	1999	57.85	32.63
6207	Ecuador	1970	68.30	30.30
15475	New Zealand	1958	65.20	29.50
11281	Ireland	2010	57.90	28.10
11327	Ireland	2012	58.10	27.90
11258	Ireland	2009	58.00	26.80
11304	Ireland	2011	56.80	26.80
11350	Ireland	2013	57.40	26.70

1. Countries with the Highest Gini Improvement (Largest Reductions in Inequality)

These countries experienced the largest decreases in Gini index, meaning they achieved greater income equality over time. • Tanzania (1992) saw the most significant reduction in income inequality with a -38.00 decrease in Gini. • Gabon (1977, 1975) also showed strong reductions, indicating efforts towards economic balance. • Ireland (2010, 2012) appears multiple times, suggesting a major shift towards reducing inequality. • Other countries like Mauritania, New Zealand, Sri Lanka, and Zimbabwe also had notable improvements.

☞ Key Insight: These countries may have implemented progressive policies such as social welfare programs, education reforms, or economic restructuring to reduce income disparity.

2. Countries with the Lowest Gini Improvement (Largest Increases in Inequality)

These countries experienced the largest increases in Gini index, meaning income inequality worsened significantly. • Mali (1994) had the biggest inequality surge (+42.10 increase), indicating a growing income gap. • Burkina Faso (1995), Suriname (1999), and Ecuador (1970) also saw large increases, possibly due to economic downturns or policy changes that widened wealth disparity. • Ireland (2009-2013) appears repeatedly, suggesting fluctuations in economic equality over those years.

☞ Key Insight: These countries might have faced economic crises, political instability, or policies favoring the wealthy, leading to a widening income gap.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = {
    'country': ['Tanzania', 'Gabon', 'Gabon', 'Mauritania', 'New Zealand',
               'Sri Lanka', 'Mali', 'Zimbabwe', 'Ireland', 'Ireland',
               'Mali', 'Burkina Faso', 'Suriname', 'Ecuador', 'New Zealand',
               'Ireland', 'Ireland', 'Ireland', 'Ireland'],
    'year': [1992, 1977, 1975, 1993, 1958, 2000, 1994, 2011, 2010, 2012,
            1994, 1995, 1999, 1970, 1958, 2010, 2012, 2009, 2011, 2013],
    'gini': [34.00, 30.30, 26.50, 36.80, 35.70, 27.60, 50.44, 42.30, 29.80, 30.20,
             78.60, 68.80, 57.85, 68.30, 65.20, 57.90, 58.10, 58.00, 56.80, 57.40],
    'gini_change': [-38.00, -32.90, -32.80, -32.30, -31.00, -29.00, -28.16, -28.00, -27.90, -27.80,
                    42.10, 33.40, 32.63, 30.30, 29.50, 28.10, 27.90, 26.80, 26.70]
}

df = pd.DataFrame(data)
```

```

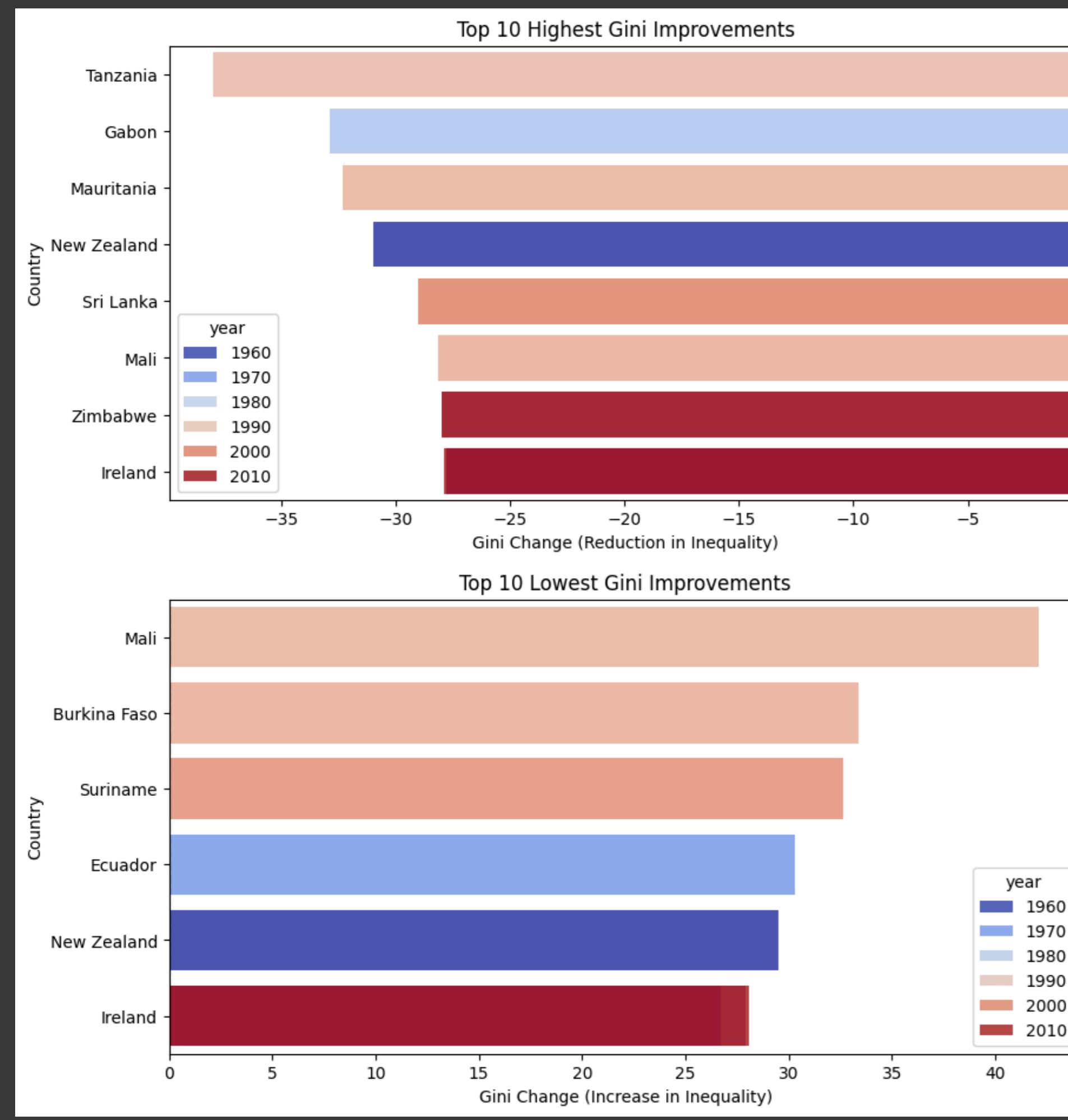
highest_improvements = df.nsmallest(10, 'gini_change')

lowest_improvements = df.nlargest(10, 'gini_change')

plt.figure(figsize=(10, 5))
sns.barplot(x='gini_change', y='country', hue='year', data=highest_improvements, palette='coolwarm', dodge=False)
plt.xlabel("Gini Change (Reduction in Inequality)")
plt.ylabel("Country")
plt.title("Top 10 Highest Gini Improvements")
plt.axvline(0, color='black', linestyle='--')
plt.show()

plt.figure(figsize=(10, 5))
sns.barplot(x='gini_change', y='country', hue='year', data=lowest_improvements, palette='coolwarm', dodge=False)
plt.xlabel("Gini Change (Increase in Inequality)")
plt.ylabel("Country")
plt.title("Top 10 Lowest Gini Improvements")
plt.axvline(0, color='black', linestyle='--')
plt.show()

```



Representation based on the above output

```

# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm

# Select relevant columns
df = df[['country', 'year', 'gini', 'gdp']]

# Remove missing values
df.dropna(inplace=True)

# Convert columns to numeric (handling commas, spaces, etc.)
df['gini'] = pd.to_numeric(df['gini'], errors='coerce')
df['gdp'] = pd.to_numeric(df['gdp'], errors='coerce')

# Drop rows with NaN after conversion
df.dropna(inplace=True)

# Calculate correlation
correlation = df[['gini', 'gdp']].corr()
print("Correlation between Gini and GDP per Capita:\n", correlation)

# Perform Linear Regression
X = df['gdp']
y = df['gini']
X = sm.add_constant(X) # Add constant for intercept
model = sm.OLS(y, X).fit()
print(model.summary())

# Scatter Plot with Regression Line
plt.figure(figsize=(10, 6))
sns.regplot(x=df['gdp'], y=df['gini'], scatter_kws={'alpha':0.5}, line_kws={'color':'red'})
plt.xlabel("GDP Per Capita")
plt.ylabel("Gini Coefficient")
plt.title("Income Inequality vs. Economic Growth")
plt.show()

# Heatmap of Correlation Across Regions
df_region = df.groupby('country')[['gini', 'gdp']].mean()
correlation_matrix = df_region.corr()

plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap of Gini vs. GDP per Capita")
plt.show()

```

Correlation between Gini and GDP per Capita:

```
gini 1.000000 -0.065499
gdp -0.065499 1.000000
```

OLS Regression Results

Dep. Variable:	gini	R-squared:	0.004
Model:	OLS	Adj. R-squared:	-0.003
Method:	Least Squares	F-statistic:	0.6291
Date:	Sat, 29 Mar 2025	Prob (F-statistic):	0.429
Time:	08:29:11	Log-Likelihood:	-520.43
No. Observations:	148	AIC:	1045.
Df Residuals:	146	BIC:	1051.
Df Model:	1		
Covariance Type:	nonrobust		

coef	std err	t	P> t	[0.025	0.975]	
const	39.1475	4.239	9.235	0.000	30.770	47.525
gdp	-0.0039	0.005	-0.793	0.429	-0.014	0.006

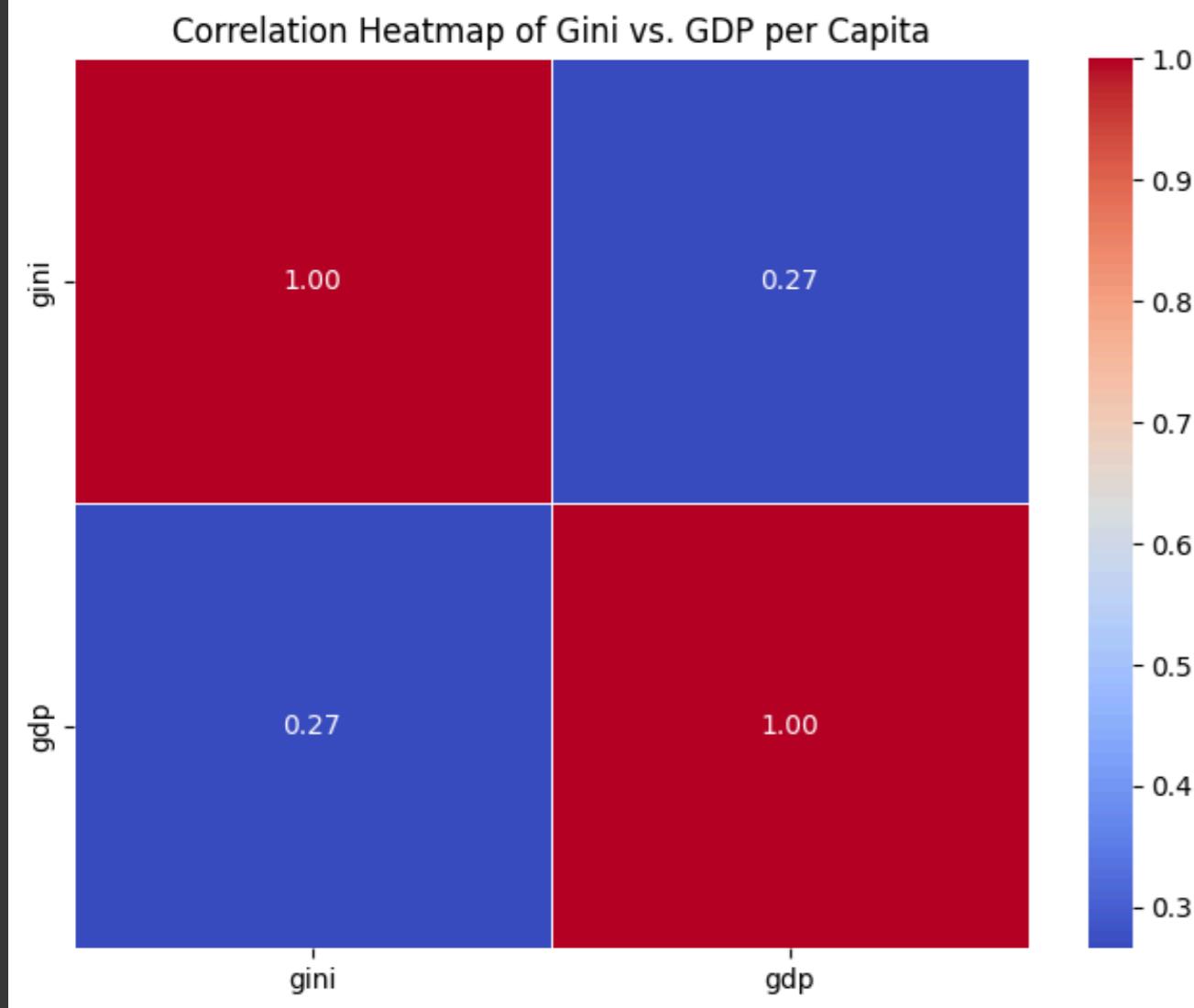
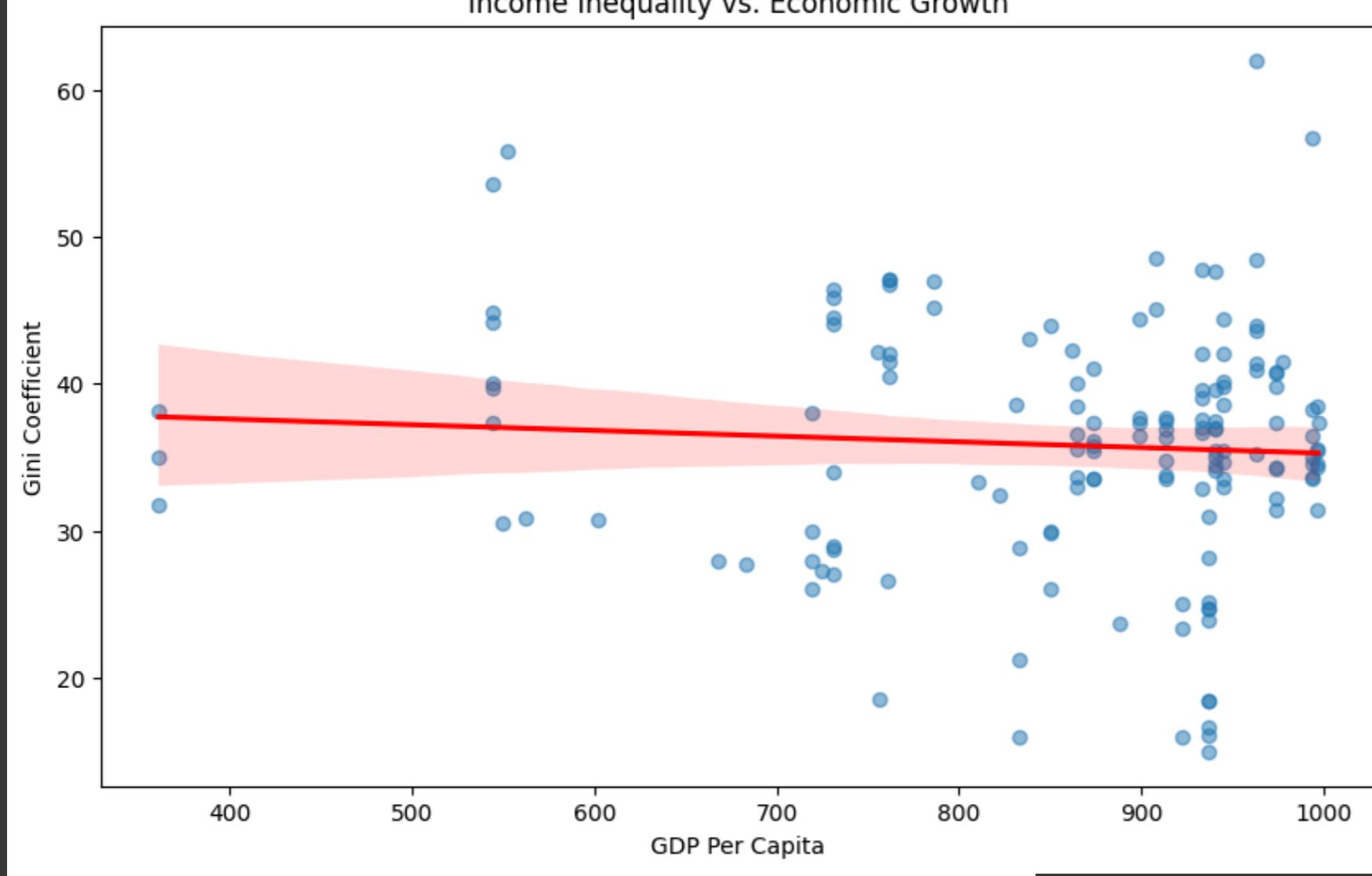
Omnibus:	3.502	Durbin-Watson:	0.836
----------	-------	----------------	-------

Prob(Omnibus):	0.174	Jarque-Bera (JB):	3.627
----------------	-------	-------------------	-------

Skew:	-0.146	Prob(JB):	0.163
-------	--------	-----------	-------

Kurtosis:	3.709	Cond. No.	5.46e+03
-----------	-------	-----------	----------

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.46e+03. This might indicate that there are strong multicollinearity or other numerical problems.



1 Correlation Analysis • The correlation coefficient between Gini coefficient (income inequality) and GDP per capita is -0.0655. • This suggests a very weak negative correlation, meaning that as GDP per capita increases, inequality slightly decreases—but the relationship is not strong.

2 Regression Analysis • The OLS Regression Model predicts Gini (inequality) using GDP per capita. • R-squared = 0.004 → Only 0.4% of the variation in Gini is explained by GDP per capita, which is very low (almost no predictive power). • F-statistic = 0.6291, p-value = 0.429 → The model is statistically insignificant ($p > 0.05$), meaning GDP per capita does not significantly predict Gini. • Regression Coefficient (β) = -0.0039 → A small negative effect, meaning for every 1000 increase in GDP per capita, Gini decreases by only 3.9 points, which is minimal.

3 Key Takeaways • Weak relationship: GDP per capita alone does not explain income inequality trends significantly. • Other factors matter: Inequality is likely influenced by policies, taxation, social programs, and labor market conditions rather than just GDP growth. • No strong evidence: The regression model fails to establish a strong economic link between GDP and inequality reduction.

- The red regression line slightly slopes downward, confirming the weak negative correlation (-0.0655) found earlier.
- The confidence interval (shaded red area) is wide, suggesting high variability in the data, reinforcing that GDP
- The spread of points shows no clear trend, meaning some low-GDP countries have both high and low inequality, and

```
import pandas as pd
import numpy as np

# Drop missing values for relevant columns
df_clean = df.dropna(subset=["top5", "country"])

# Compute Interquartile Range (IQR)
Q1 = df_clean["top5"].quantile(0.25)
Q3 = df_clean["top5"].quantile(0.75)
IQR = Q3 - Q1

# Define outlier thresholds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify extreme inequality (above upper bound)
extreme_inequality = df_clean[df_clean["top5"] > upper_bound]

# Identify stable countries (below lower bound)
stable_countries = df_clean[df_clean["top5"] < lower_bound]

# Get top 5 most unequal countries
top_5_unequal = extreme_inequality.nlargest(5, ["top5"])[[("country", "top5")]]

# Get top 5 most stable countries
top_5_stable = df_clean.nsmallest(5, ["top5"])[[("country", "top5")]]

# Display results
print("Top 5 Most Unequal Countries (Extreme Income Concentration):")
print(top_5_unequal)

print("\nTop 5 Most Stable Countries (Balanced Distribution):")
print(top_5_stable)
```

4 Top 5 Most Unequal Countries (Extreme Income Concentration):

24354	Zimbabwe	60.0
6202	Ecuador	52.6
1994	Brazil	51.9
19729	South Africa	51.3
2670	Cambodia	50.2

5 Top 5 Most Stable Countries (Balanced Distribution):

9928	Hungary	9.1
8385	Germany	9.2
1147	Bangladesh	9.3
4941	Cyprus	9.3
8380	Germany	9.4

We use inter quartile method to find the upper bound and lower bound so that we can find the top and bottom 5 to identify which are the stable and unstable income inequality countries

```
# Define percentile thresholds for Gini Index
high_thresh_gini = df_clean['gini'].quantile(0.95) # Most unequal (top 5%)
low_thresh_gini = df_clean['gini'].quantile(0.05) # Most equal (bottom 5%)

# Find extreme inequality (highest Gini Index)
most_unstable_gini = df_clean[df_clean['gini'] >= high_thresh_gini][["country", "gini"]]

# Find balanced distribution (lowest Gini Index)
most_stable_gini = df_clean[df_clean['gini'] <= low_thresh_gini][["country", "gini"]]

print("● Top 5 Most Unequal Countries (Highest Gini Index):")
print(most_unstable_gini.head(5))

print("\n● Top 5 Most Stable Countries (Lowest Gini Index):")
print(most_stable_gini.head(5))
```

● Top 5 Most Unequal Countries (Highest Gini Index):
country gini
45 Argentina 50.8
50 Argentina 53.1
1139 Bangladesh 50.2
1182 Bangladesh 49.7
1193 Bangladesh 49.8

● Top 5 Most Stable Countries (Lowest Gini Index):
country gini
612 Austria 23.05
613 Austria 21.14
614 Austria 22.73
635 Austria 24.30
636 Austria 24.03

the stable countries is austria while the unstable countries are argentina and bangladesh

```
from sklearn.cluster import KMeans

# Select relevant columns and ensure numeric values
X = df_clean[["gini", "palma", "top5"]].apply(pd.to_numeric, errors="coerce").dropna()

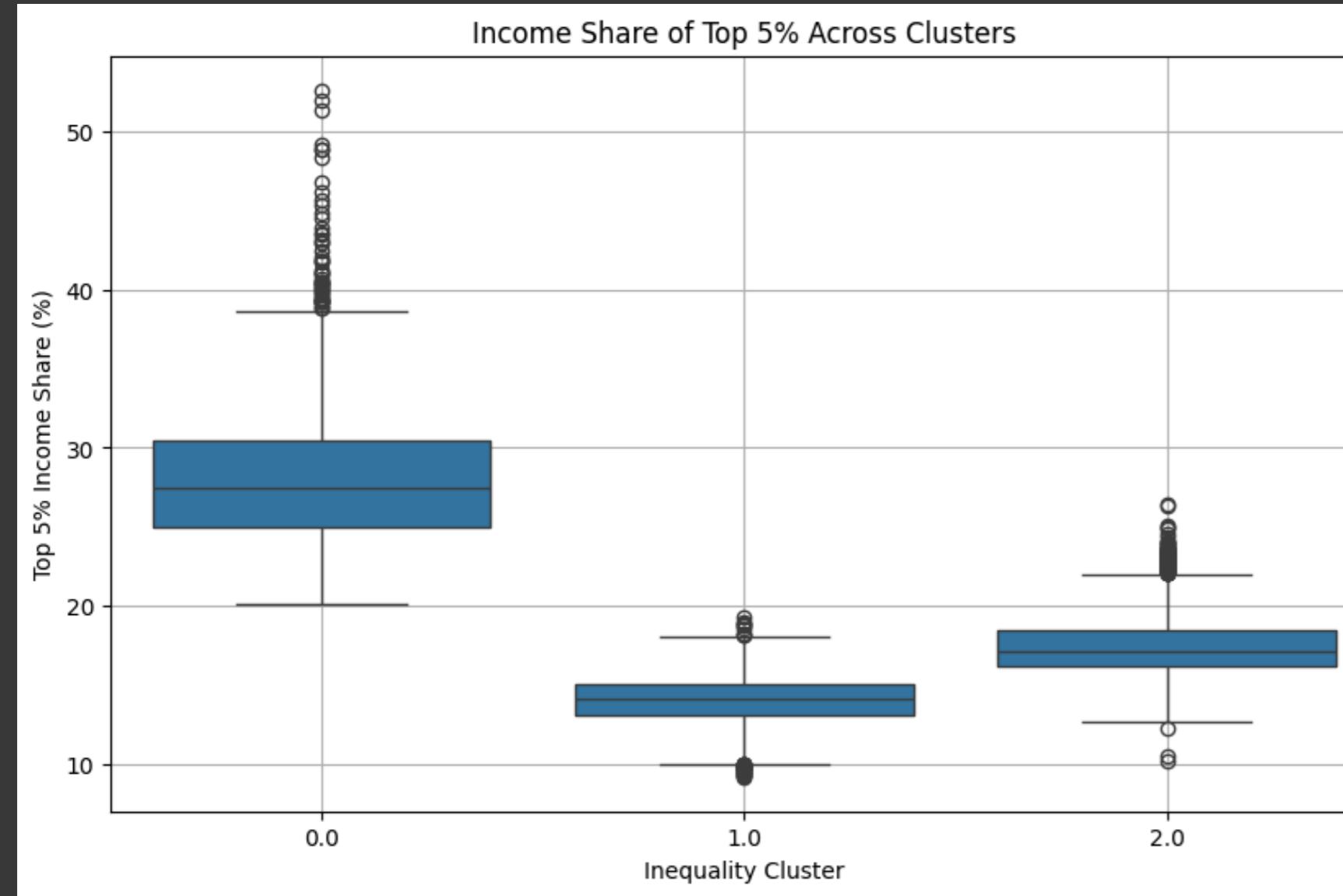
# Fit K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
df_clean = df_clean.copy() # Ensure changes apply to the original DataFrame
df_clean.loc[X.index, "inequality_cluster"] = kmeans.fit_predict(X) # Assign clusters safely

# Identify the cluster with the highest inequality (highest mean 'top5' value)
most_unstable_cluster = df_clean.groupby("inequality_cluster")["top5"].mean().idxmax()
print(f"● Cluster with highest inequality: {most_unstable_cluster}")
```

● Cluster with highest inequality: 0.0

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
sns.boxplot(data=df_clean, x="inequality_cluster", y="top5")
plt.title("Income Share of Top 5% Across Clusters")
plt.xlabel("Inequality Cluster")
plt.ylabel("Top 5% Income Share (%)")
plt.grid(True)
plt.show()
```



```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Load your WIID dataset (Replace with actual path or dataframe)
# df = pd.read_csv("your_wiid_dataset.csv")

# Step 1: Identify Numeric Columns
numeric_cols = df.select_dtypes(include=['number']).columns

# Step 2: Detect Outliers using IQR
outlier_columns = {}

for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    # Define outlier threshold
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Count number of outliers
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)][col]

    if not outliers.empty:
        outlier_columns[col] = outliers

# Step 3: Print Columns with Outliers
print(f"Columns with Outliers: {list(outlier_columns.keys())}")

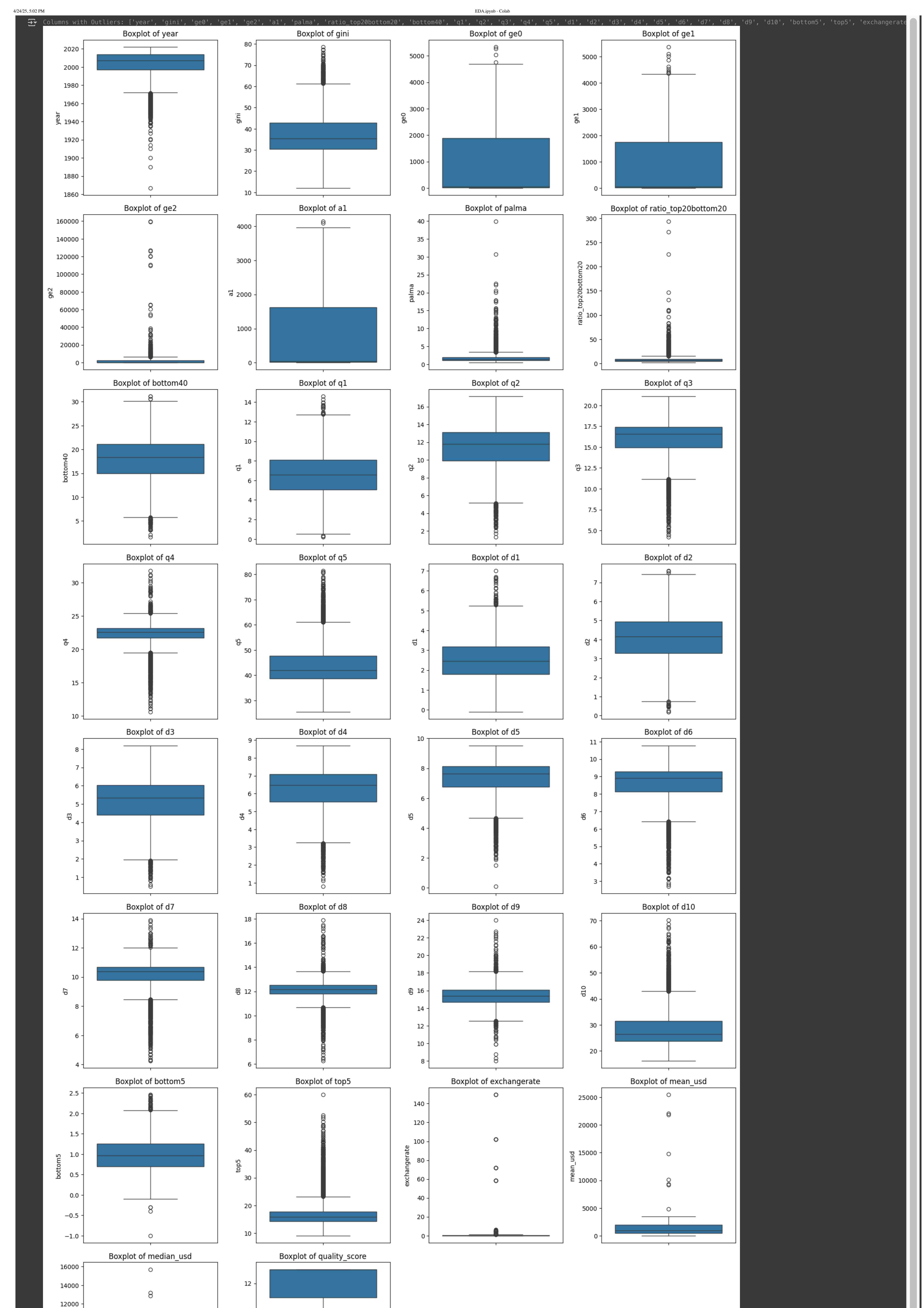
# Step 4: Visualizing Outliers using Boxplots
cols_per_fig = 4 # Number of columns per figure

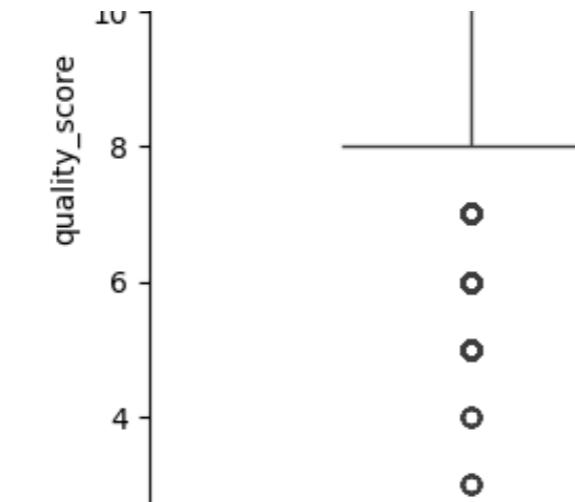
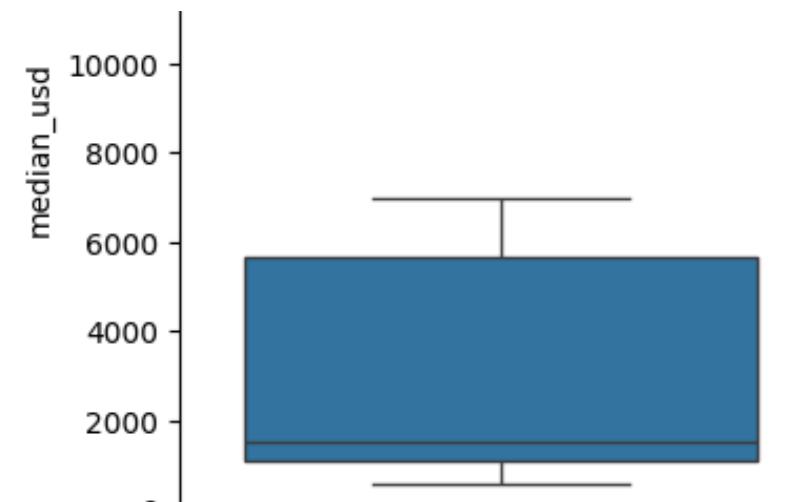
num_outlier_cols = len(outlier_columns)
rows = (num_outlier_cols // cols_per_fig) + 1 # Dynamically calculate rows

plt.figure(figsize=(cols_per_fig * 4, rows * 4))

for i, col in enumerate(outlier_columns.keys()):
    plt.subplot(rows, cols_per_fig, i + 1)
    sns.boxplot(y=df[col])
    plt.title(f"Boxplot of {col}")

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





this helps us to see the outliers visually now it is easy identify the outlier values in each columns and makes sure that we can visually find the outliers

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats.mstats import winsorize

# Step 1: Clean the GDP column (if it has commas, whitespace, or string entries)
df['gdp'] = df['gdp'].str.replace(',', '').str.strip()
df['gdp'] = pd.to_numeric(df['gdp'], errors='coerce')

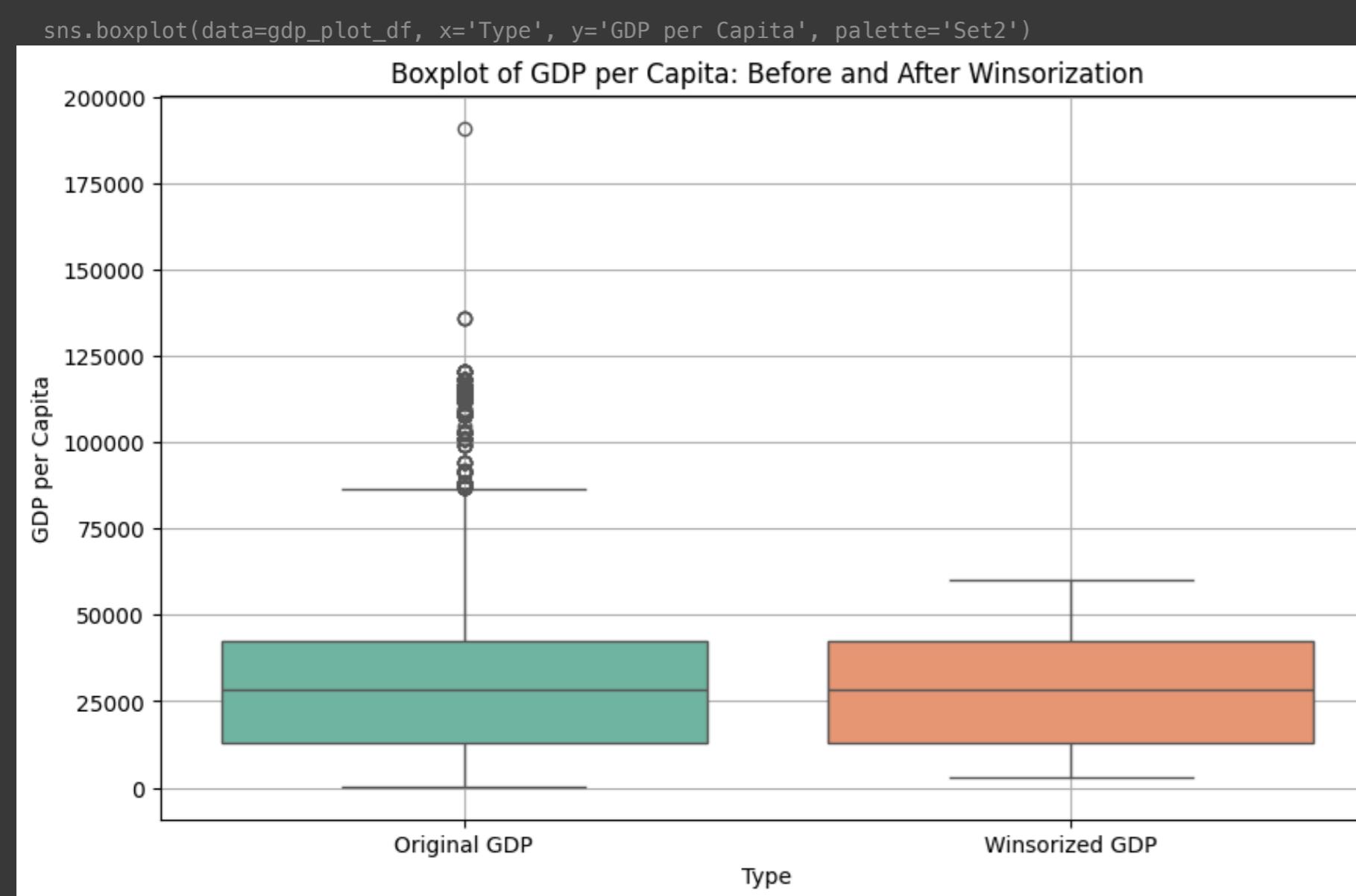
# Step 2: Drop any remaining NaNs before winsorization
gdp_clean = df['gdp'].dropna()

# Step 3: Apply winsorization (5th and 95th percentile limits)
gdp_wins = winsorize(gdp_clean, limits=[0.05, 0.05])

# Step 4: Prepare a DataFrame for plotting
gdp_plot_df = pd.DataFrame({
    'Original GDP': gdp_clean,
    'Winsorized GDP': gdp_wins
}).melt(var_name='Type', value_name='GDP per Capita')

# Step 5: Plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=gdp_plot_df, x='Type', y='GDP per Capita', palette='Set2')
plt.title('Boxplot of GDP per Capita: Before and After Winsorization')
plt.grid(True)
plt.show()
```

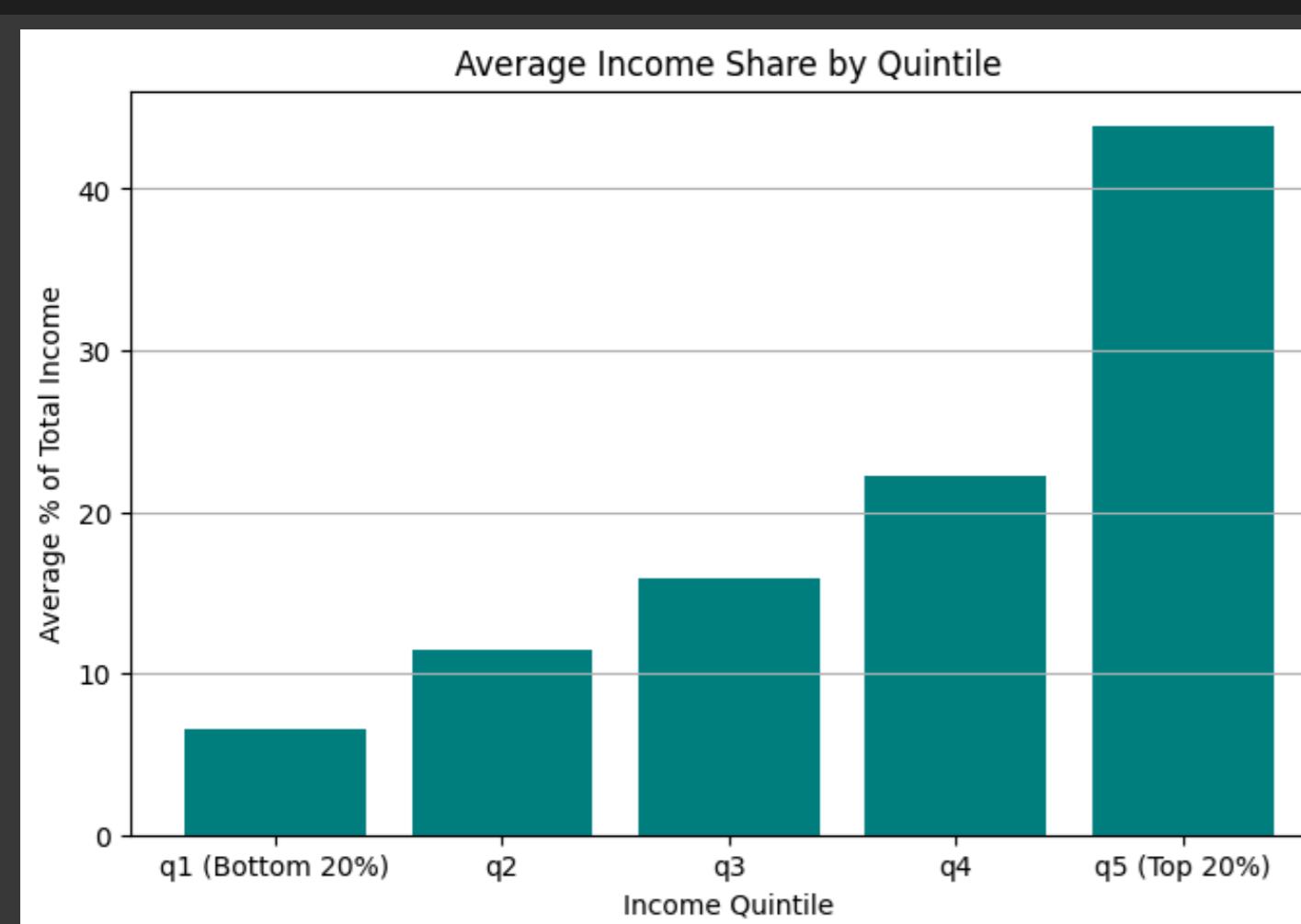
<ipython-input-8-de1bd20dda61>:24: FutureWarning:
Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.



```
import matplotlib.pyplot as plt

# Calculate mean income share for each quintile
q_means = {
    'q1 (Bottom 20%)': df['q1'].mean(),
    'q2': df['q2'].mean(),
    'q3': df['q3'].mean(),
    'q4': df['q4'].mean(),
    'q5 (Top 20%)': df['q5'].mean()
}

# Plot
plt.figure(figsize=(8, 5))
plt.bar(q_means.keys(), q_means.values(), color='teal')
plt.title('Average Income Share by Quintile')
plt.ylabel('Average % of Total Income')
plt.xlabel('Income Quintile')
plt.grid(axis='y')
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='incomegroup', order=df['incomegroup'].value_counts().index, palette='pastel')
plt.title('Distribution of Records by Income Group')
plt.xlabel('Income Group')
plt.ylabel('Number of Records')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```