Clustering Countries for Strategic Aid Allocation

By: Parth Patel

Problem statement:

A humanitarian NGO dedicated to alleviating poverty and providing essential services to underdeveloped nations, has successfully raised $10 million to support its mission. The organisation now faces the critical task of allocating these funds strategically to maximise impact. The organisation must identify and prioritise the countries in the most dire need of aid, ensuring that limited resources are directed where they can have the greatest positive effect on poverty and community development. The goal is to identify the countries that are most vulnerable and in need of urgent support, enabling the NGO to deploy its resources effectively.

Data description :

Country: Name of the country

Child_mort: Death of children under 5 years of age per 1000 live births

Exports: Exports of goods and services per capita. Given as %age of the GDP per capita

Health: Total health spending per capita. Given as %age of GDP per capita

Imports: Imports of goods and services per capita. Given as %age of the GDP per capita

Income: Net income per person

Inflation: The measurement of the annual growth rate of the Total GDP

Life_expec: The average number of years a new born child would live

Total_fer: The number of children that would be born to each woman

Gdpp: The GDP per capita. Calculated as the Total GDP divided by the total population.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchC
from sklearn.metrics import classification_report,accuracy_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler,MinMaxScaler,OneHotEncoder,LabelEn
from sklearn.feature_selection import RFE
from scipy.stats import uniform
from sklearn.pipeline import make_pipeline
from sklearn.impute import KNNImputer
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
import regex as re
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
import scipy.stats as stats


import warnings
warnings.filterwarnings('ignore')


df=pd.read_csv("Country-data.csv")
```

```
df.head()
```

	country	child_mort	exports	health	imports	income	inflation	life_expe
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60
	Antigua							

```
df.tail()
```

	country	child_mort	exports	health	imports	income	inflation	life_exp
162	Vanuatu	29.2	46.6	5.25	52.7	2950	2.62	6:
163	Venezuela	17.1	28.5	4.91	17.6	16500	45.90	7!
164	Vietnam	23.3	72.0	6.84	80.2	4490	12.10	7:
165	Yemen	56.3	30.0	5.18	34.4	4480	23.60	6'
166	Zambia	83.1	37.0	5.89	30.9	3280	14.00	5:

```
df.info()
```

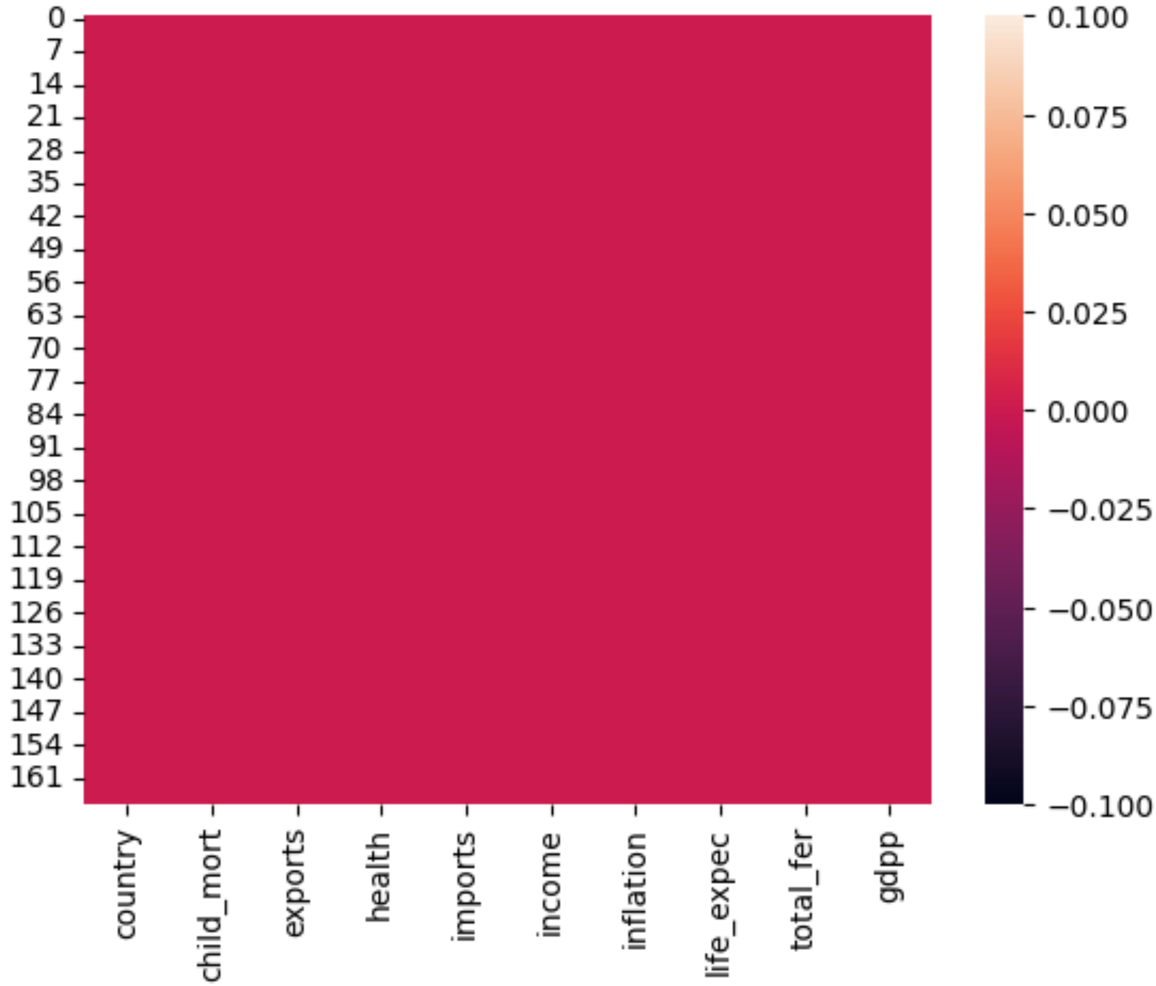
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   country     167 non-null    object
 1   child_mort  167 non-null    float64
 2   exports     167 non-null    float64
 3   health      167 non-null    float64
 4   imports     167 non-null    float64
 5   income      167 non-null    int64
 6   inflation   167 non-null    float64
 7   life_expec  167 non-null    float64
 8   total_fer   167 non-null    float64
 9   gdpp        167 non-null    int64
dtypes: float64(7), int64(2), object(1)
memory usage: 13.2+ KB
```

```
df.describe(include="all")
```

	country	child_mort	exports	health	imports	income	inf
count	167	167.000000	167.000000	167.000000	167.000000	167.000000	167
unique	167	NaN	NaN	NaN	NaN	NaN	
top	Afghanistan	NaN	NaN	NaN	NaN	NaN	
freq	1	NaN	NaN	NaN	NaN	NaN	
mean	NaN	38.270060	41.108976	6.815689	46.890215	17144.688623	7
std	NaN	40.328931	27.412010	2.746837	24.209589	19278.067698	10
min	NaN	2.600000	0.109000	1.810000	0.065900	609.000000	-4
25%	NaN	8.250000	23.800000	4.920000	30.200000	3355.000000	1
50%	NaN	19.300000	35.000000	6.320000	43.300000	9960.000000	5
75%	NaN	62.100000	51.350000	8.600000	58.750000	22800.000000	10
max	NaN	208.000000	200.000000	17.900000	174.000000	125000.000000	104

```
sns.heatmap(df.isna())#No Missing Data
```

```
<Axes: >
```




```
df.duplicated().sum() #No Duplicates
```

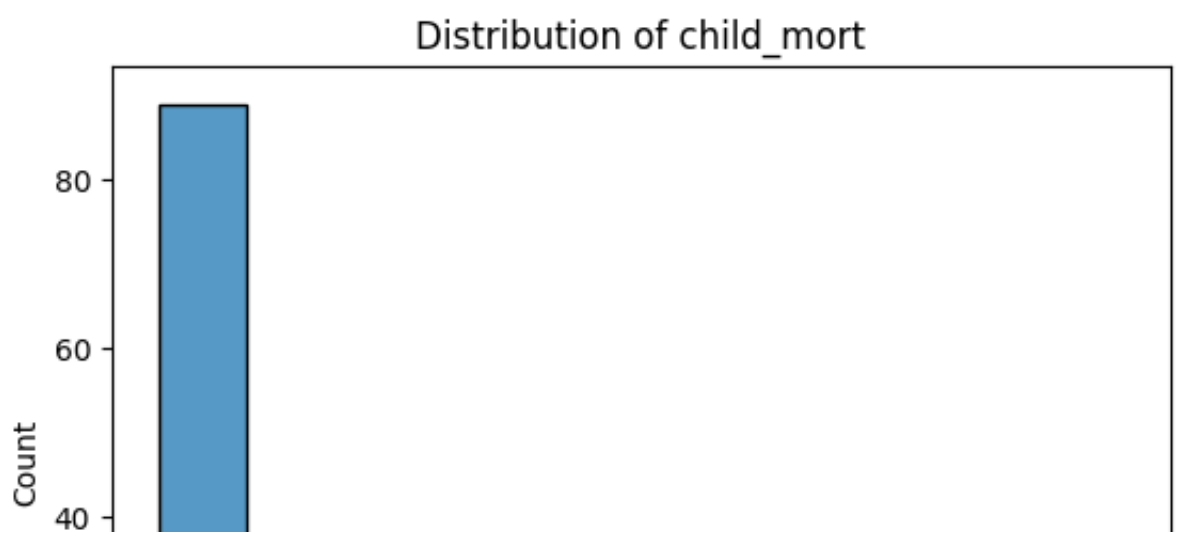
```
0
```

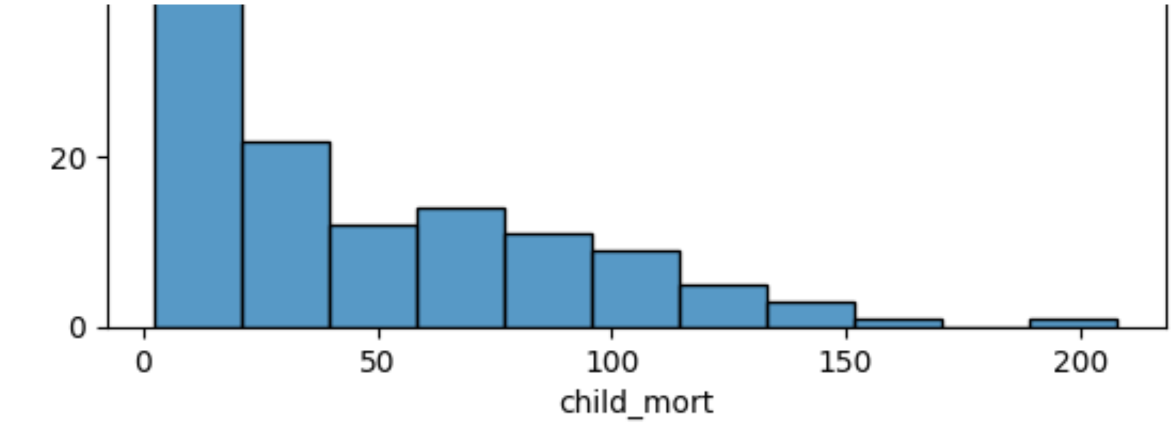
```
df.isna().sum()
```

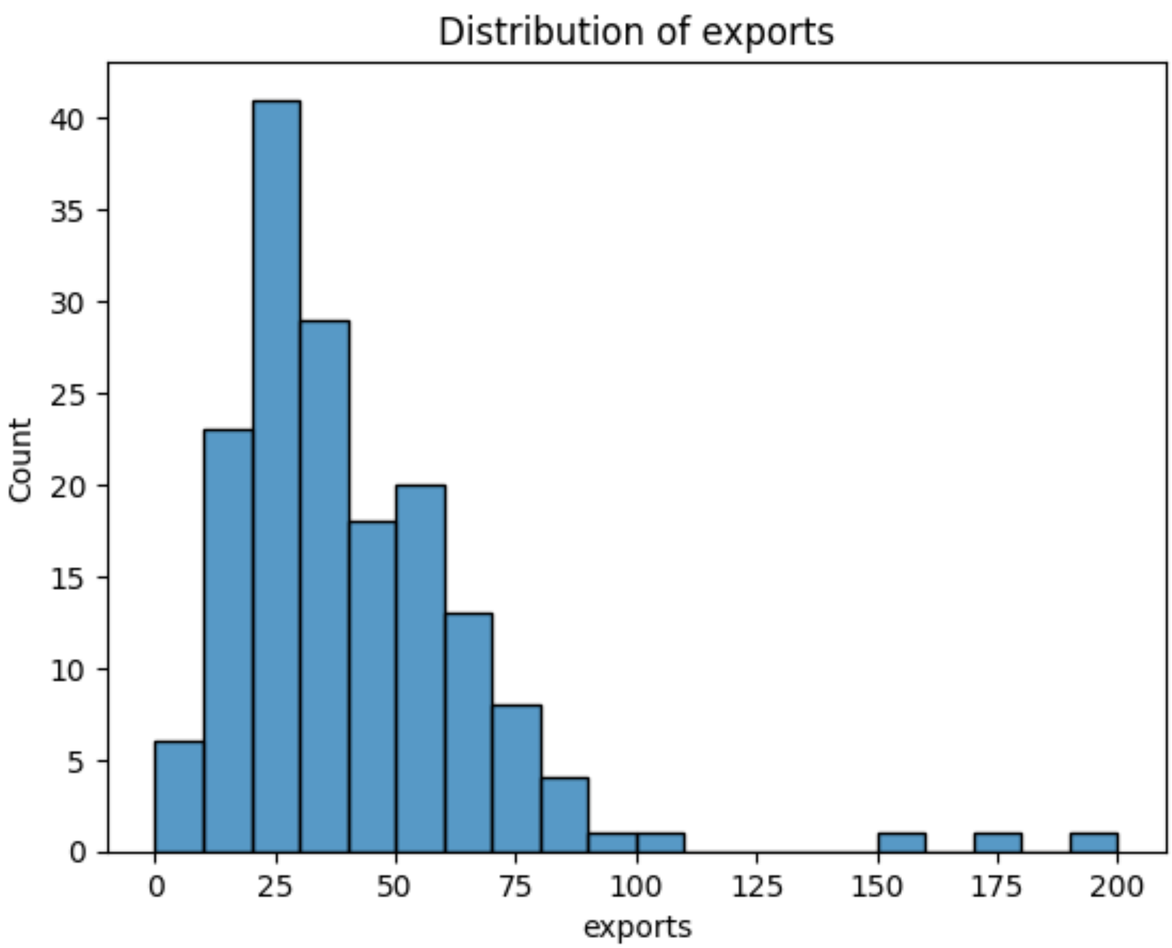
	0
country	0
child_mort	0
exports	0
health	0
imports	0
income	0
inflation	0
life_expec	0
total_fer	0
gdpp	0

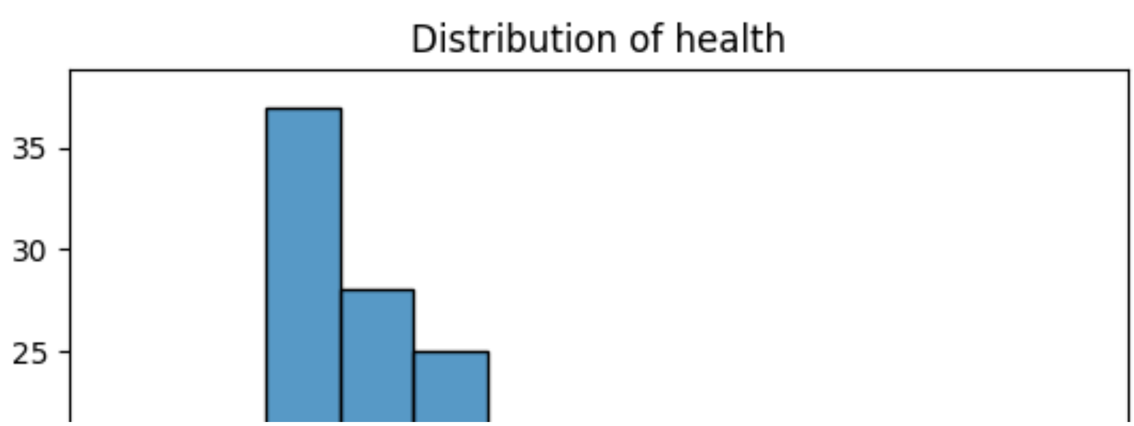
dtype: int64

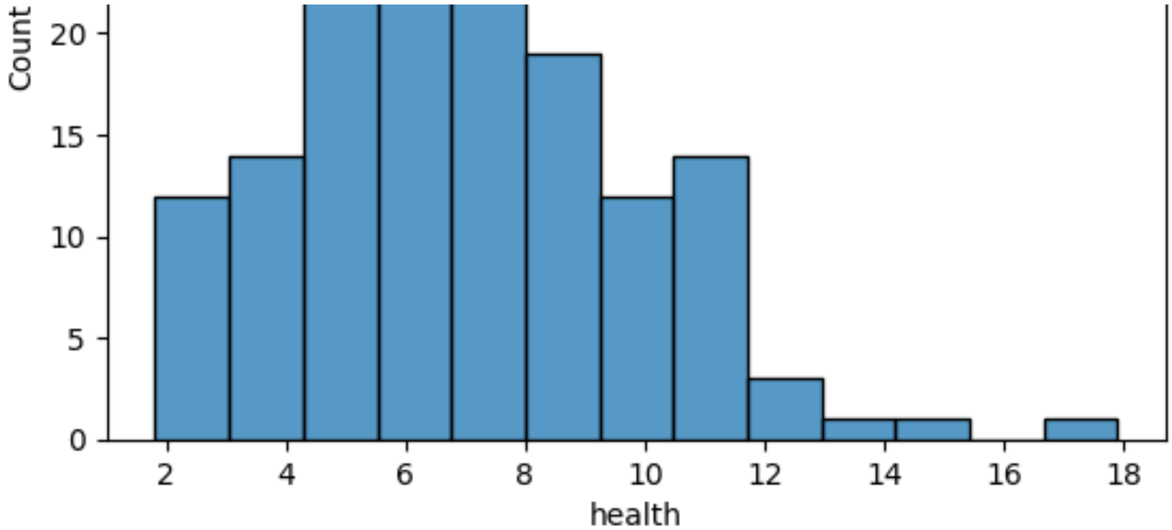
```
# Univariate Analysis
for col in df.columns:
  if df[col].dtype != 'object':
    plt.figure()
    sns.histplot(df[col])
    plt.title(f"Distribution of {col}")
    plt.show()
```

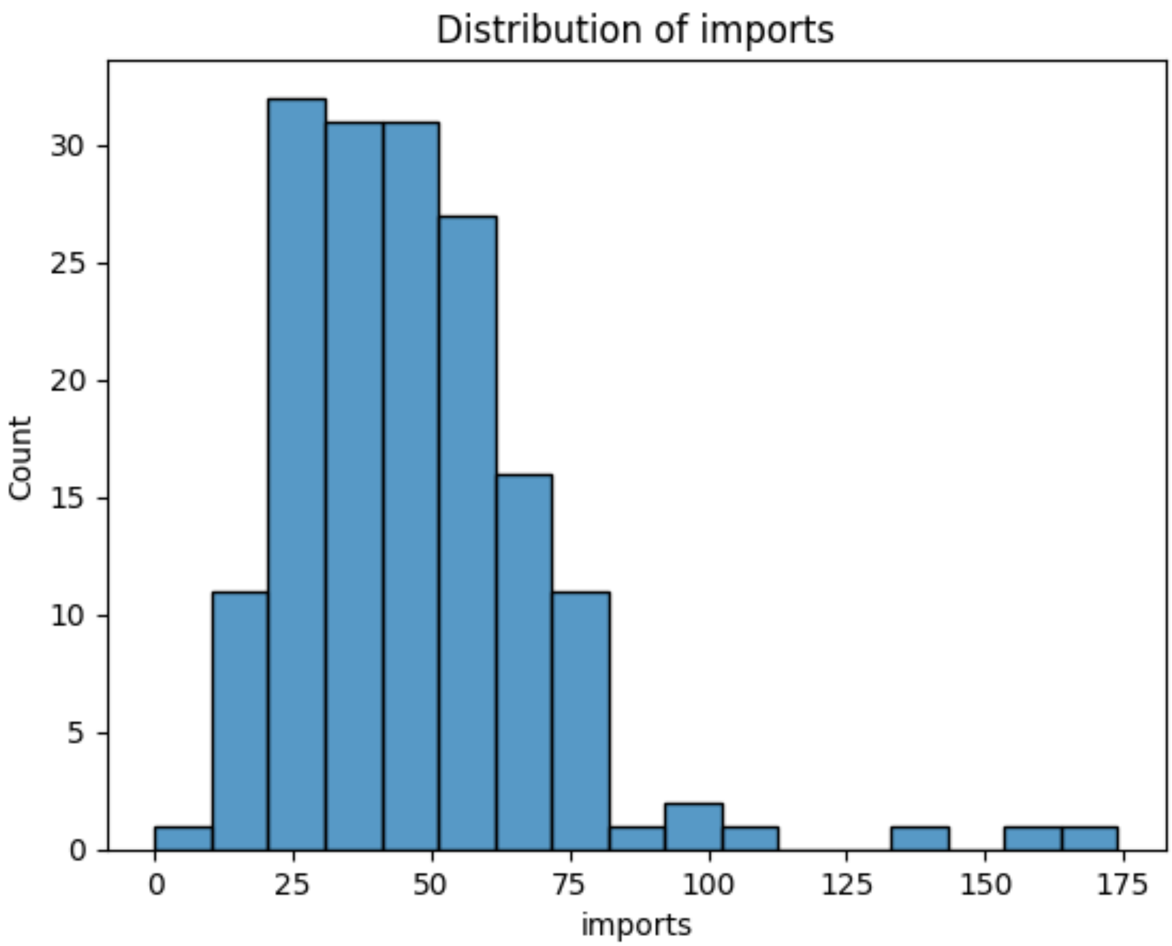


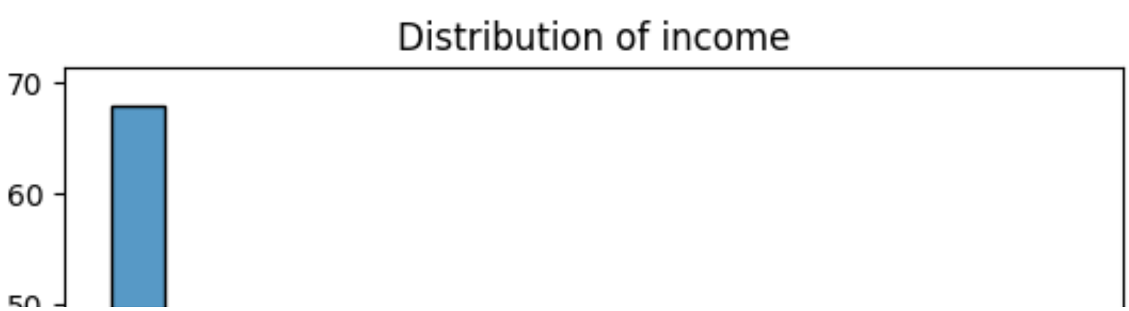

20
0
0
50
100
150
200
child_mort

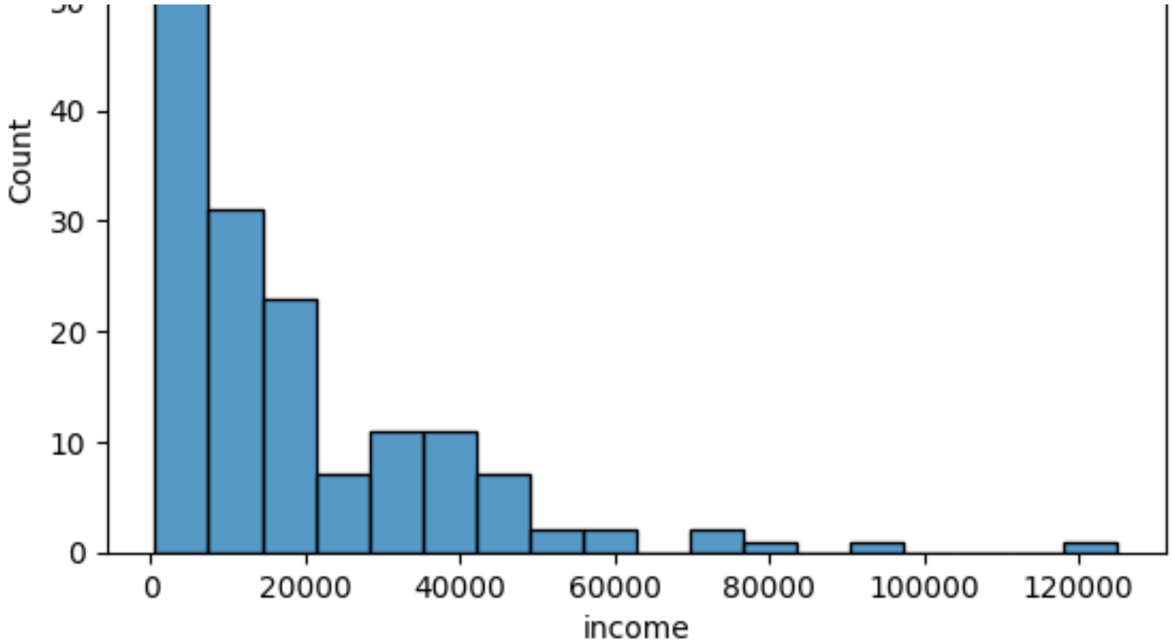

Distribution of exports
Count
exports
0
5
10
15
20
25
30
35
40
0
25
50
75
100
125
150
175
200


Distribution of health
35
30
25

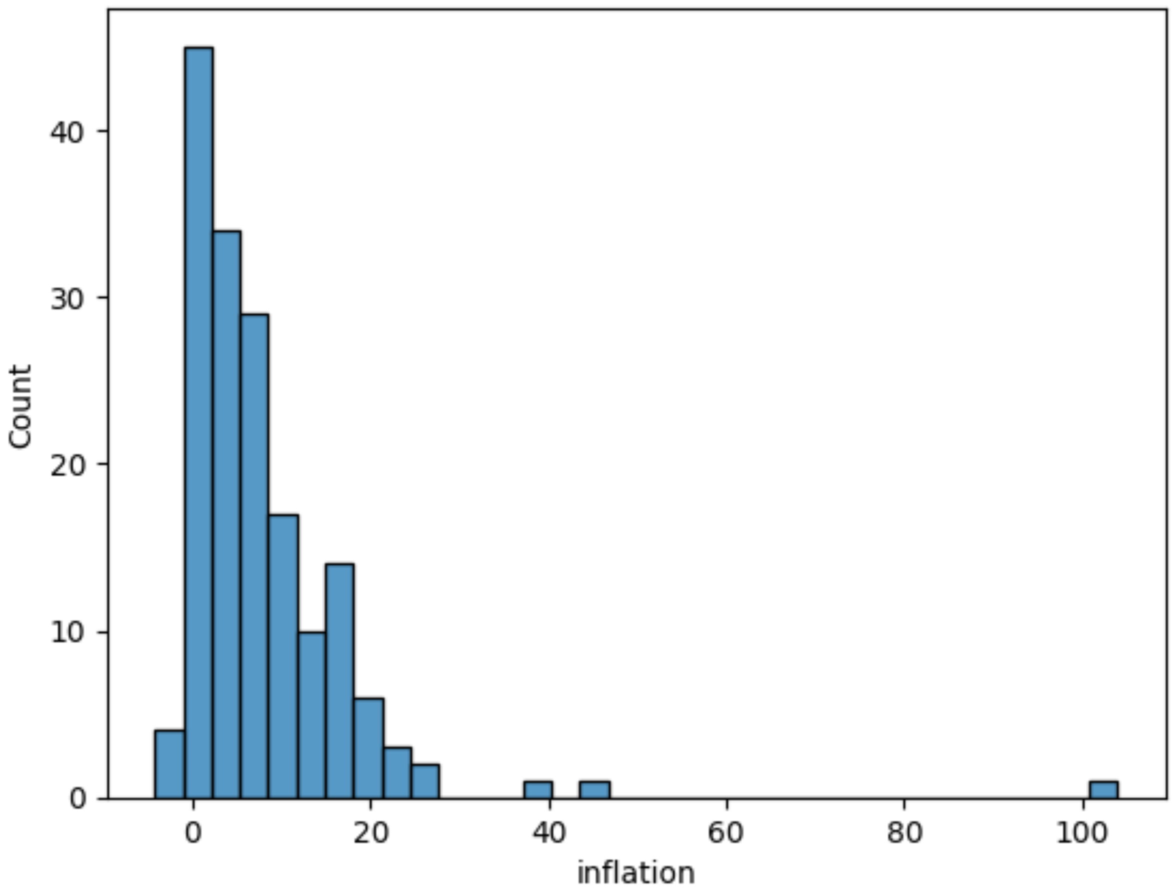

Count
20
15
10
5
0
2
4
6
8
10
12
14
16
18
health


Distribution of imports
Count
30
25
20
15
10
5
0
0
25
50
75
100
125
150
175
imports

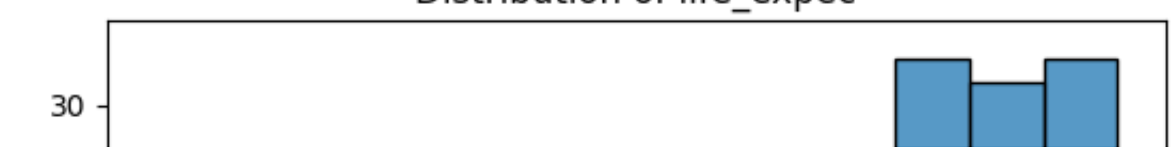

Distribution of income
70
60

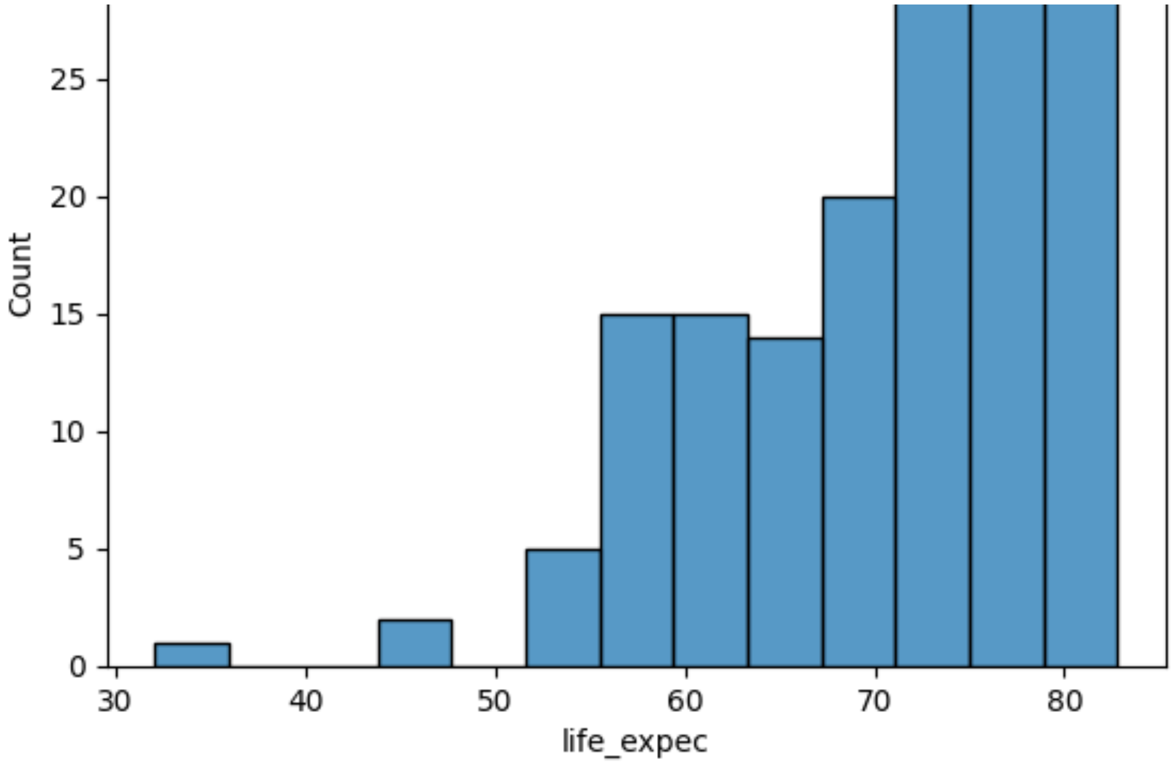


Distribution of inflation

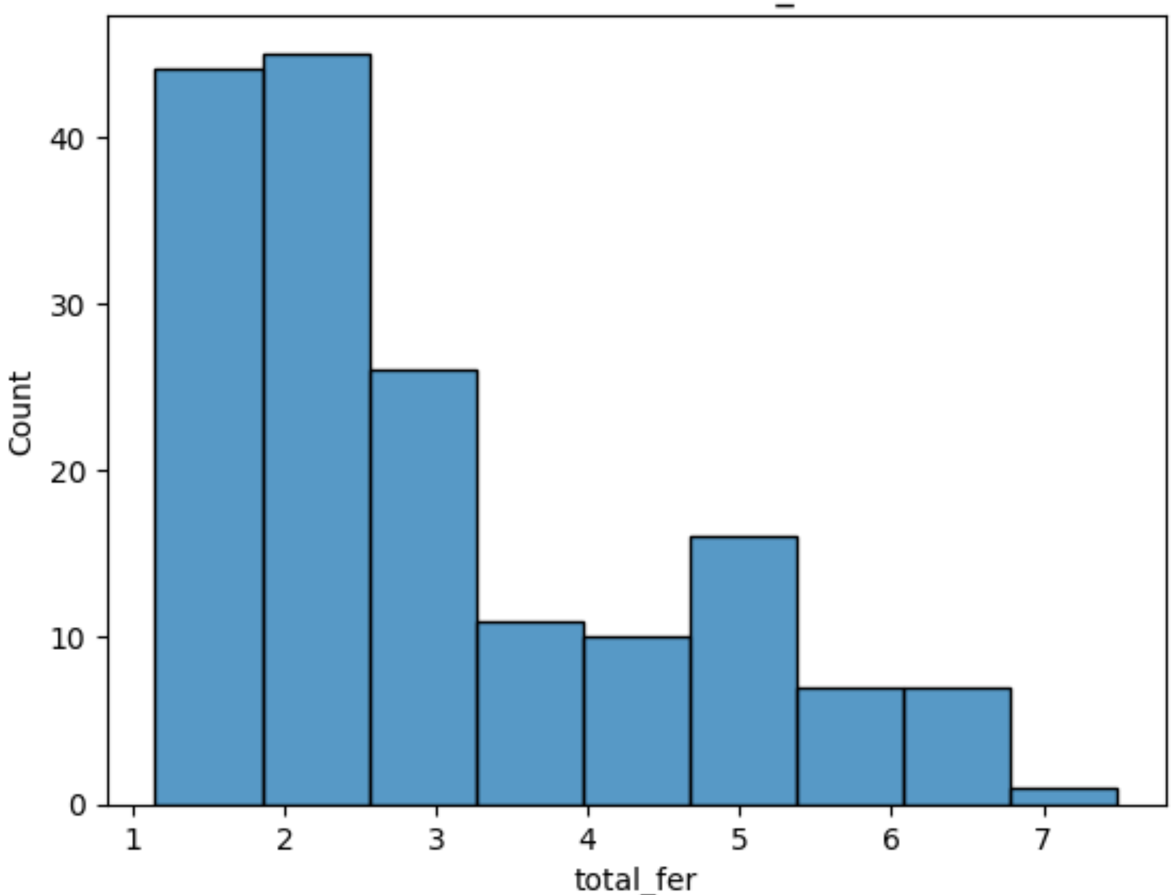


Distribution of life_expec

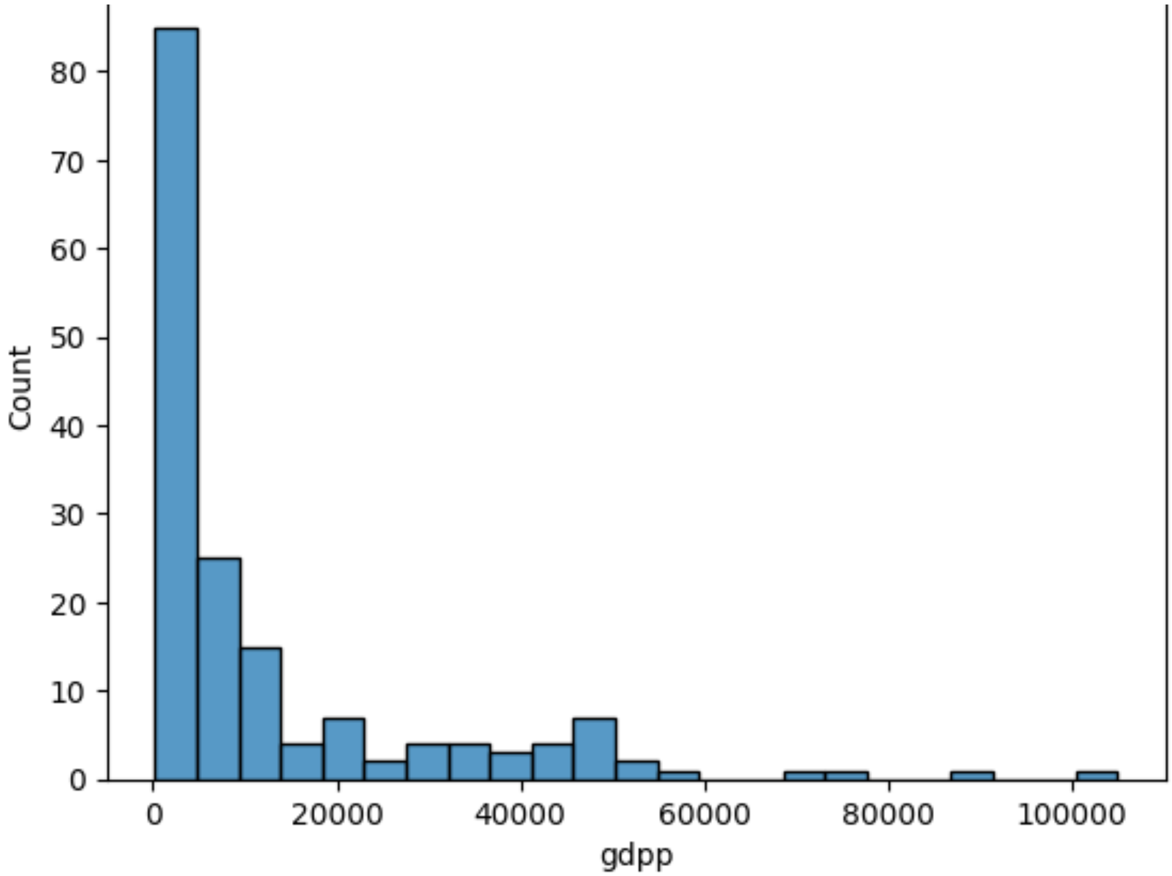




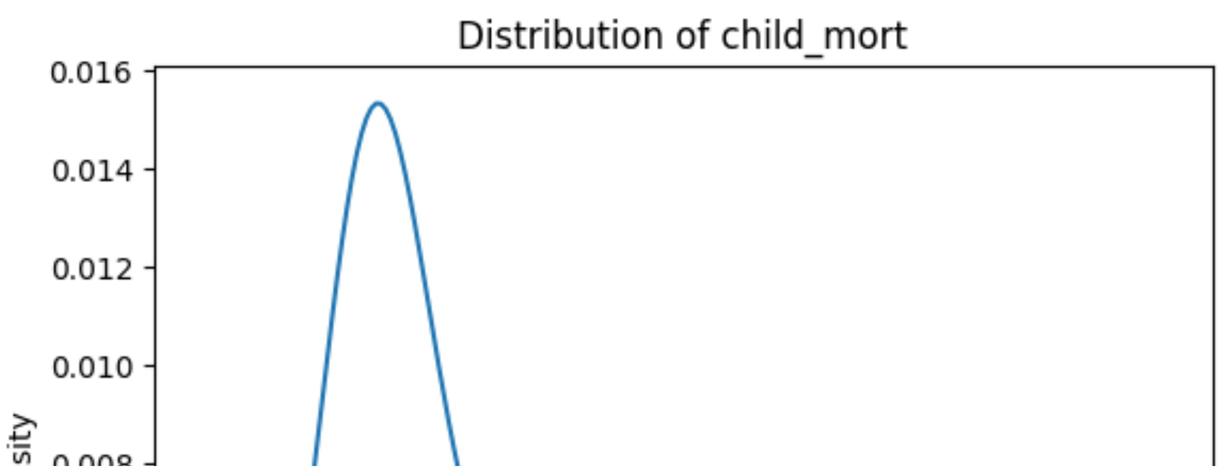
Distribution of total_fer

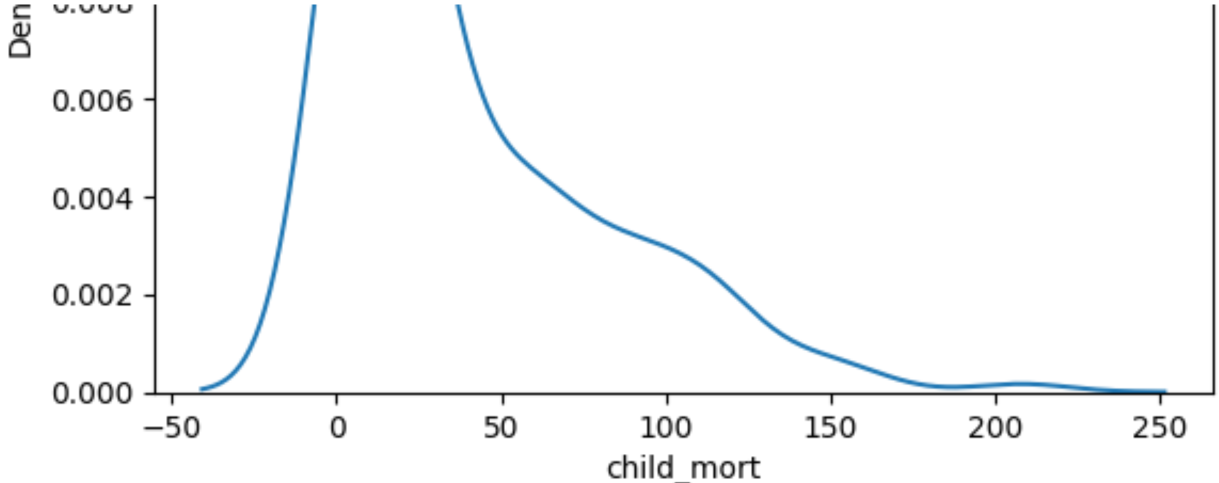


Distribution of gdpp

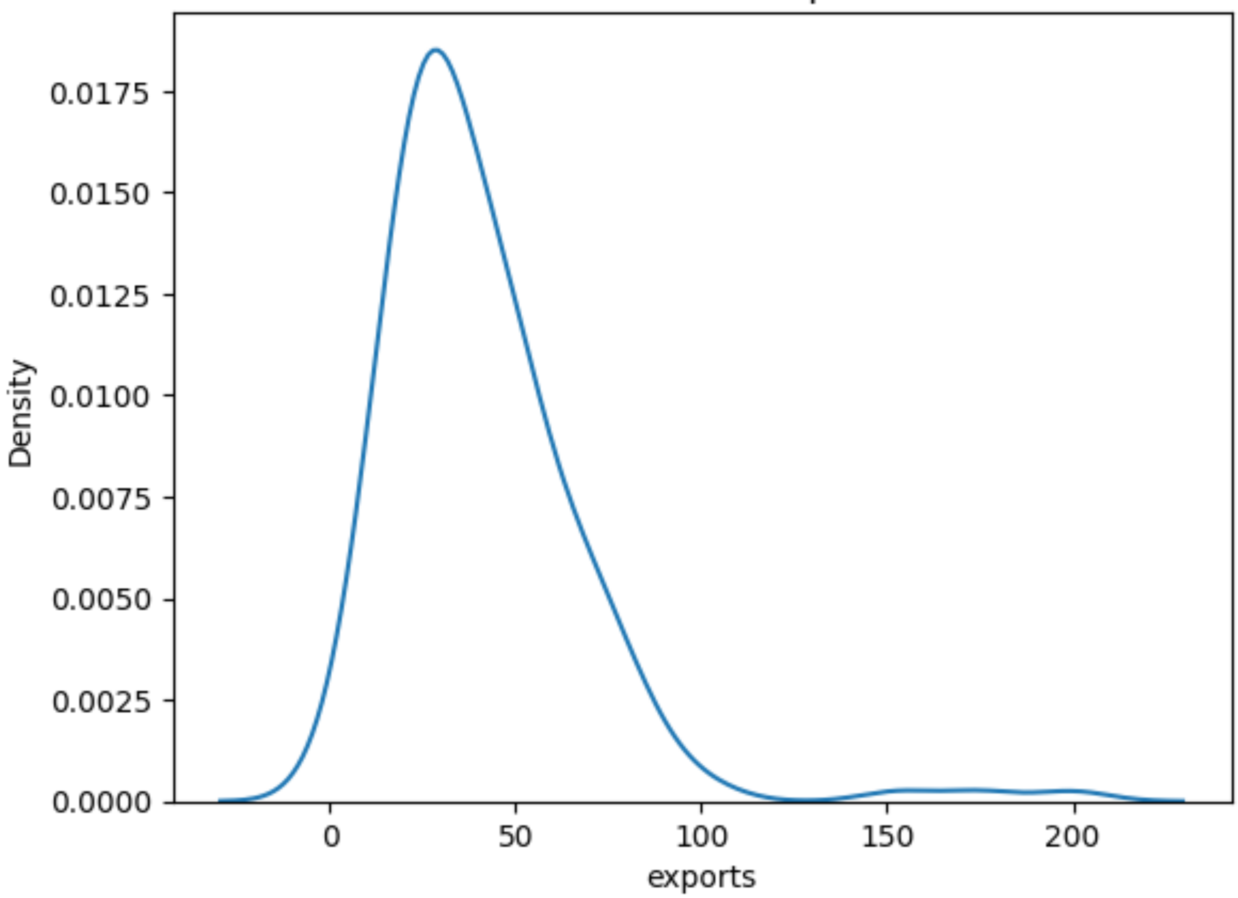



```
# Univariate Analysis
for col in df.columns:
  if df[col].dtype != 'object':
    plt.figure()
    sns.kdeplot(df[col])
    plt.title(f"Distribution of {col}")
    plt.show()
```

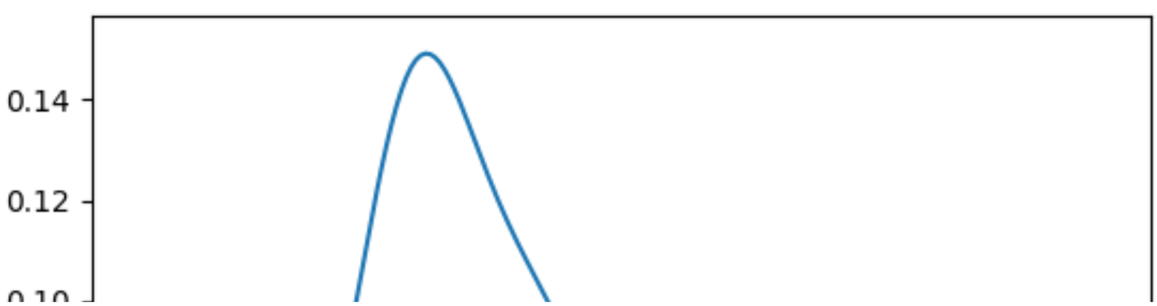


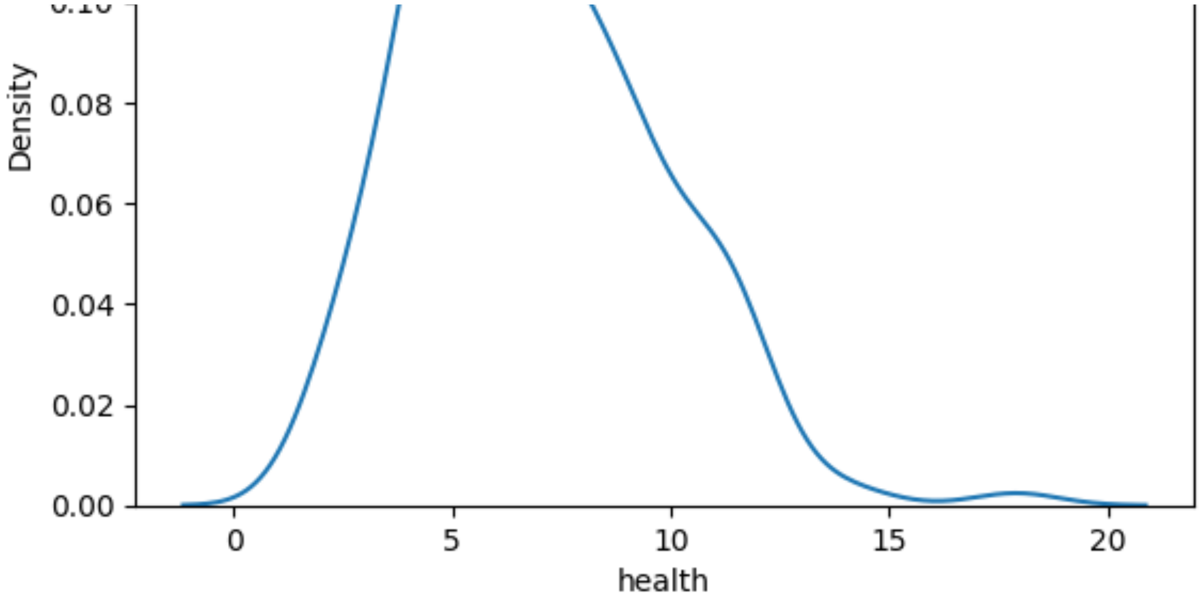


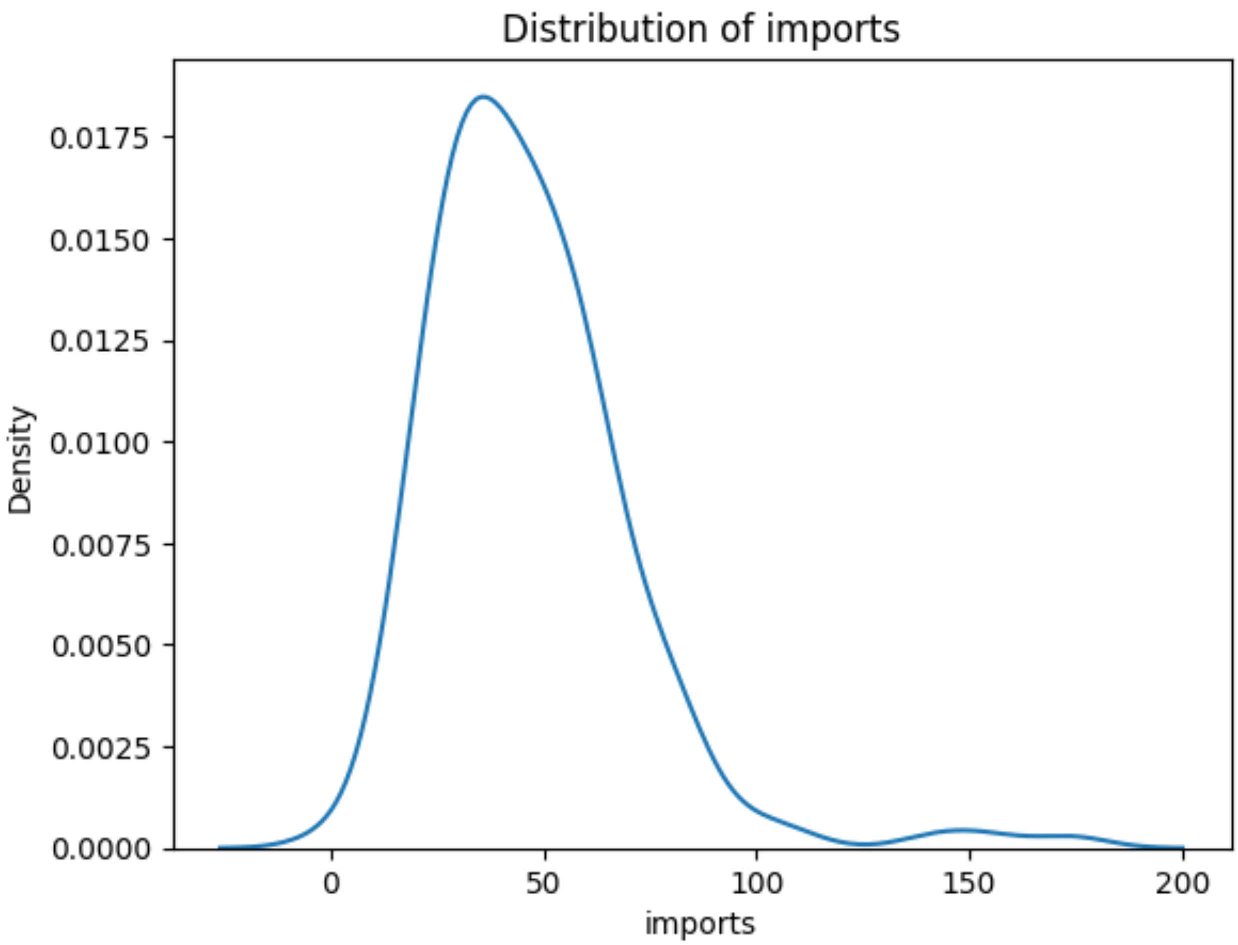
Distribution of exports

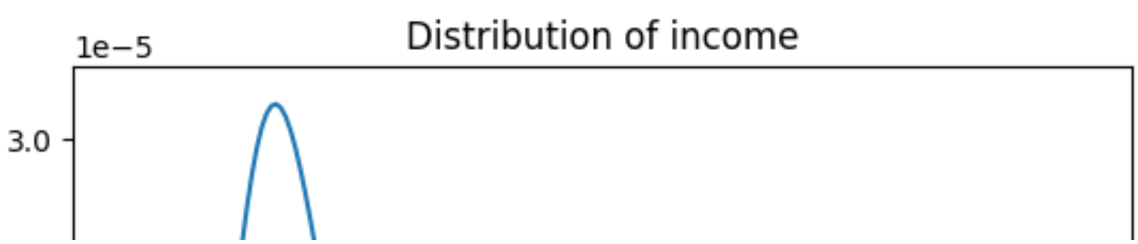


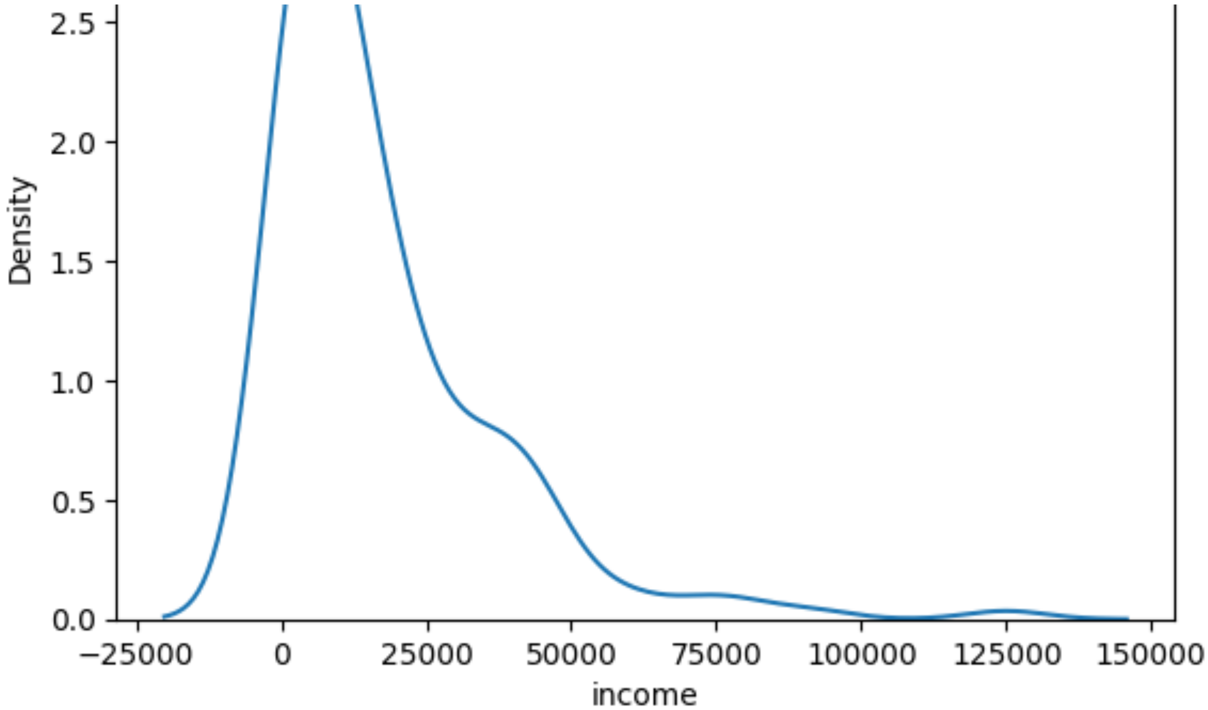
Distribution of health

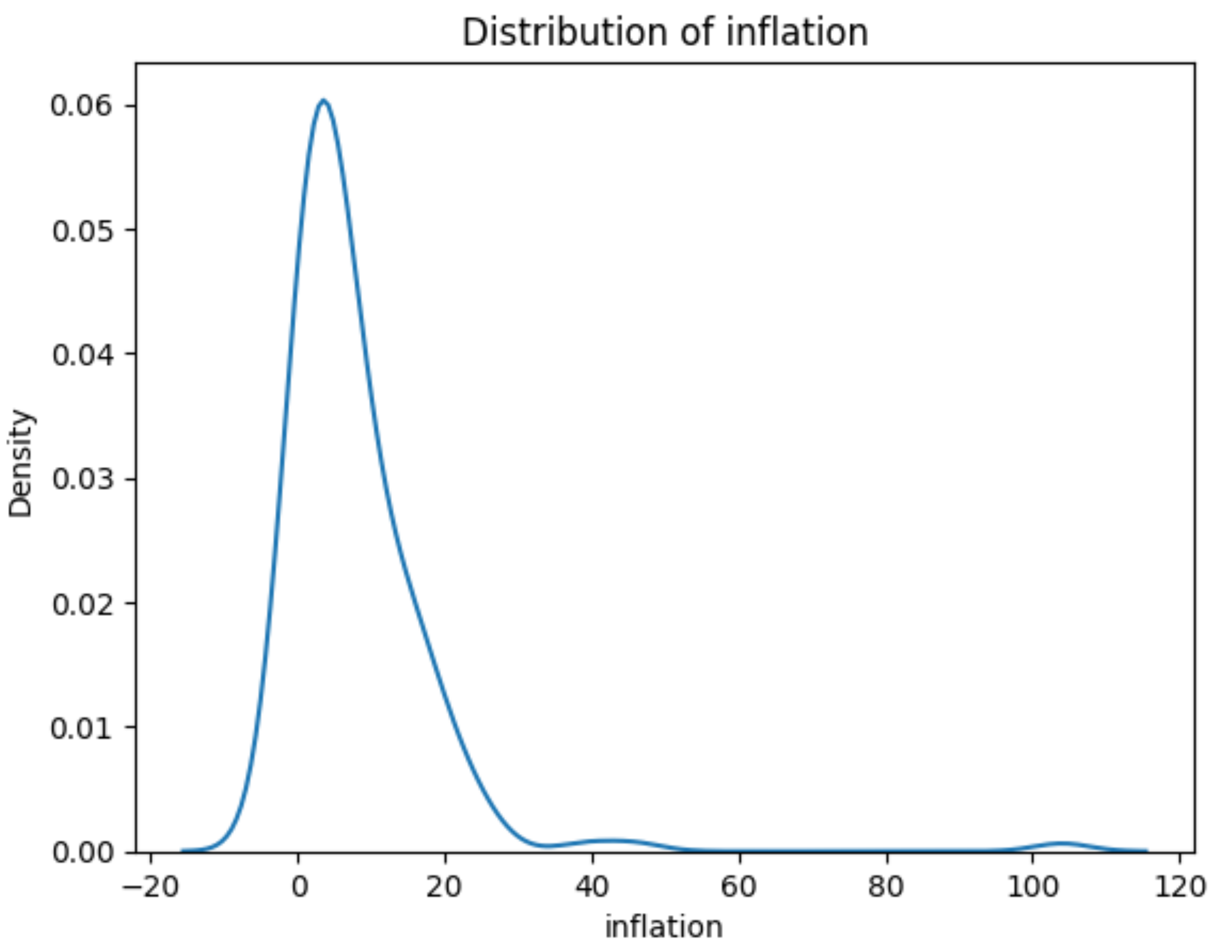


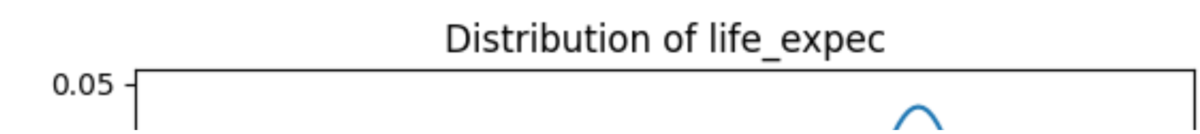

Density
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
health

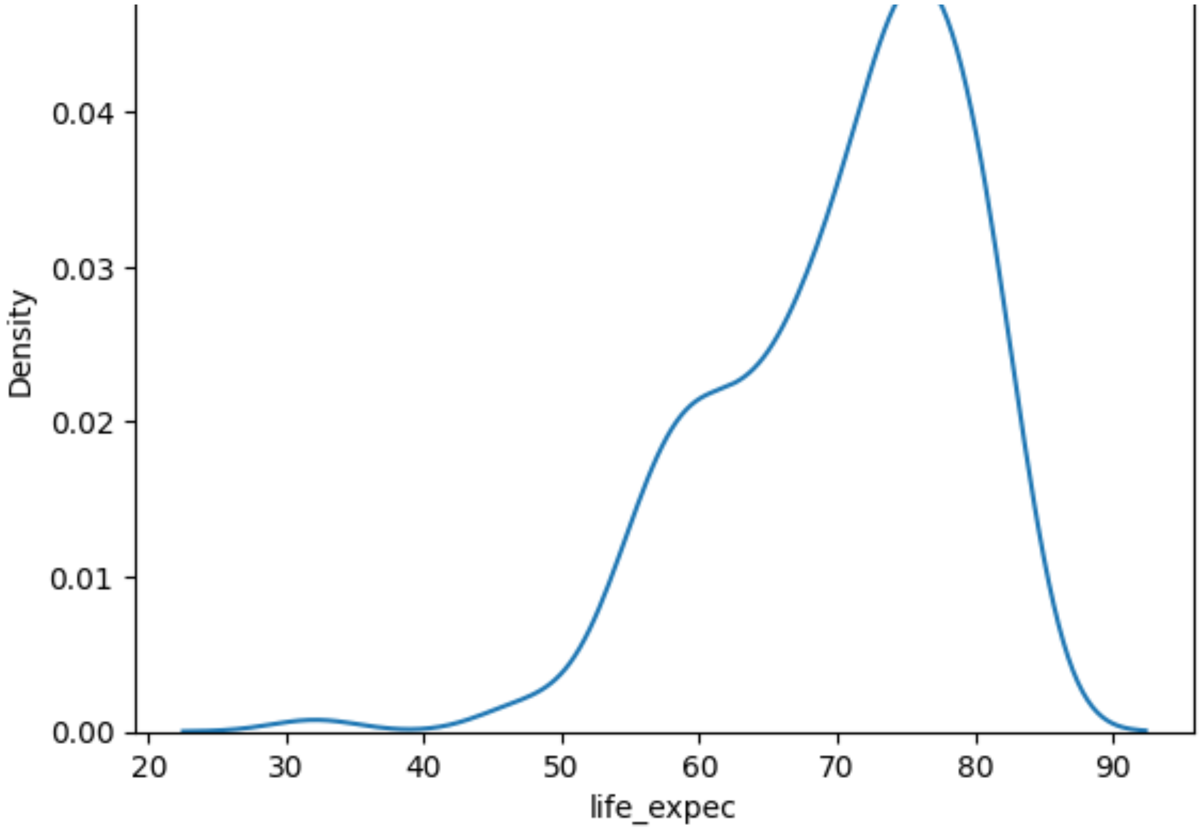

Distribution of imports
Density
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
50
100
150
200
imports

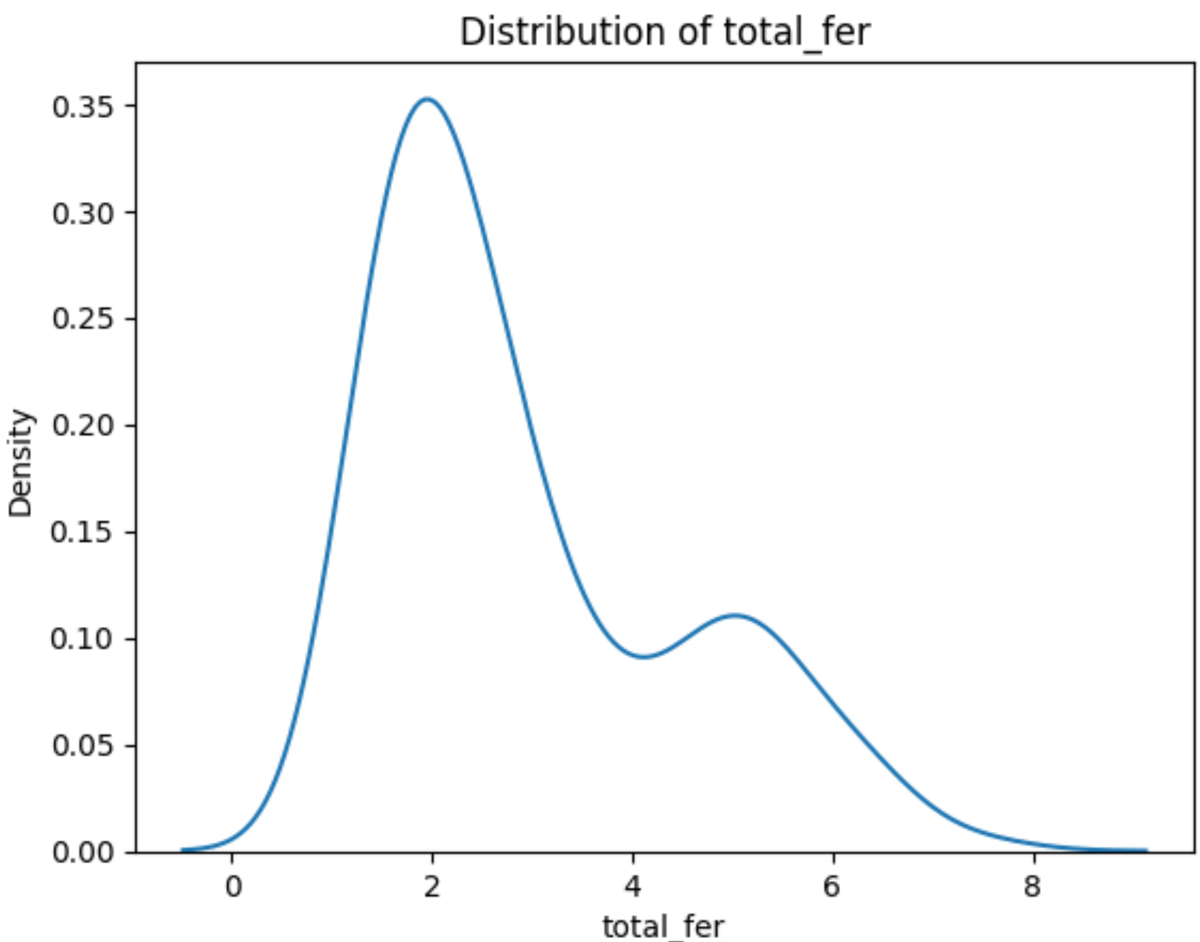

Distribution of income
1e−5
3.0

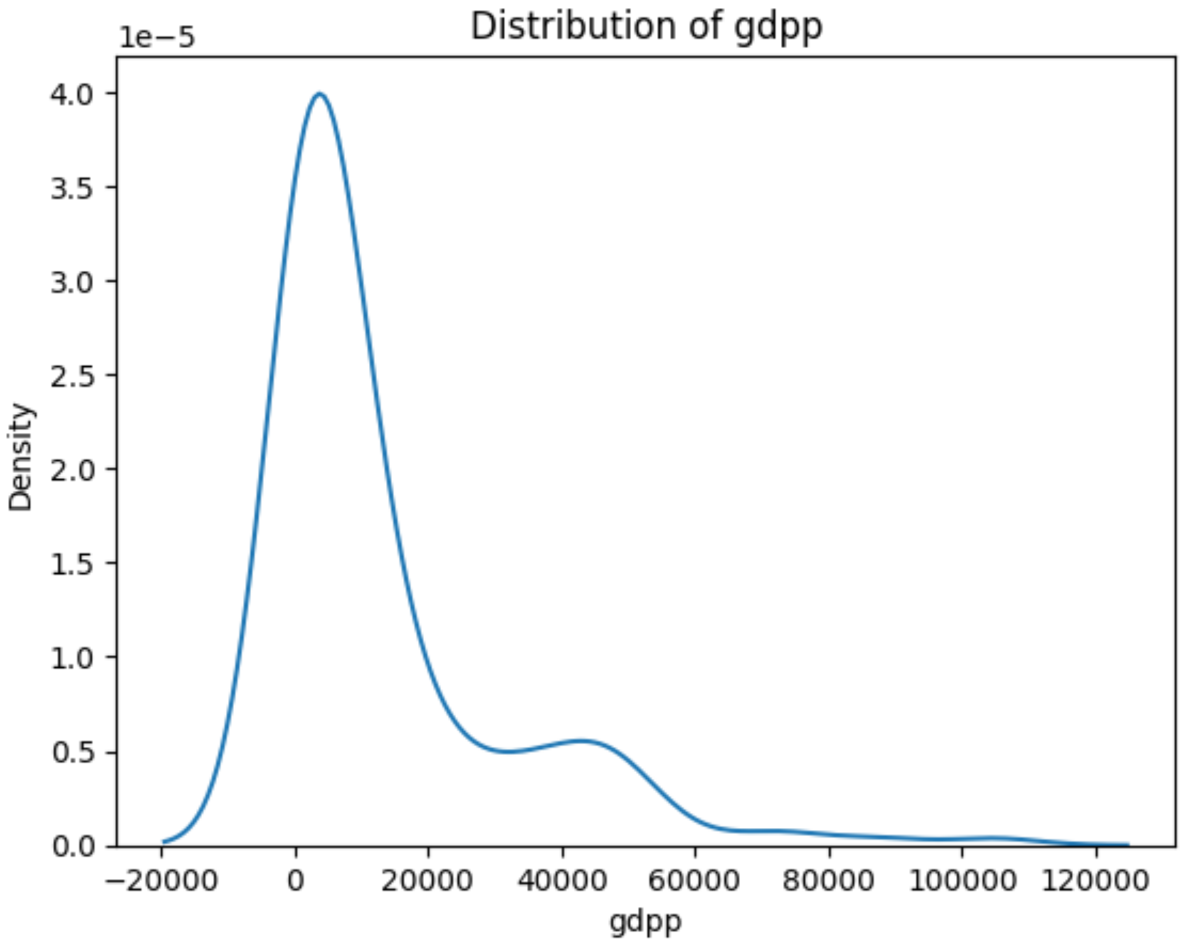

2.5
2.0
1.5
1.0
0.5
0.0
Density
−25000
0
25000
50000
75000
100000
125000
150000
income


Distribution of inflation
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
−20
0
20
40
60
80
100
120
inflation


Distribution of life_expec
0.05

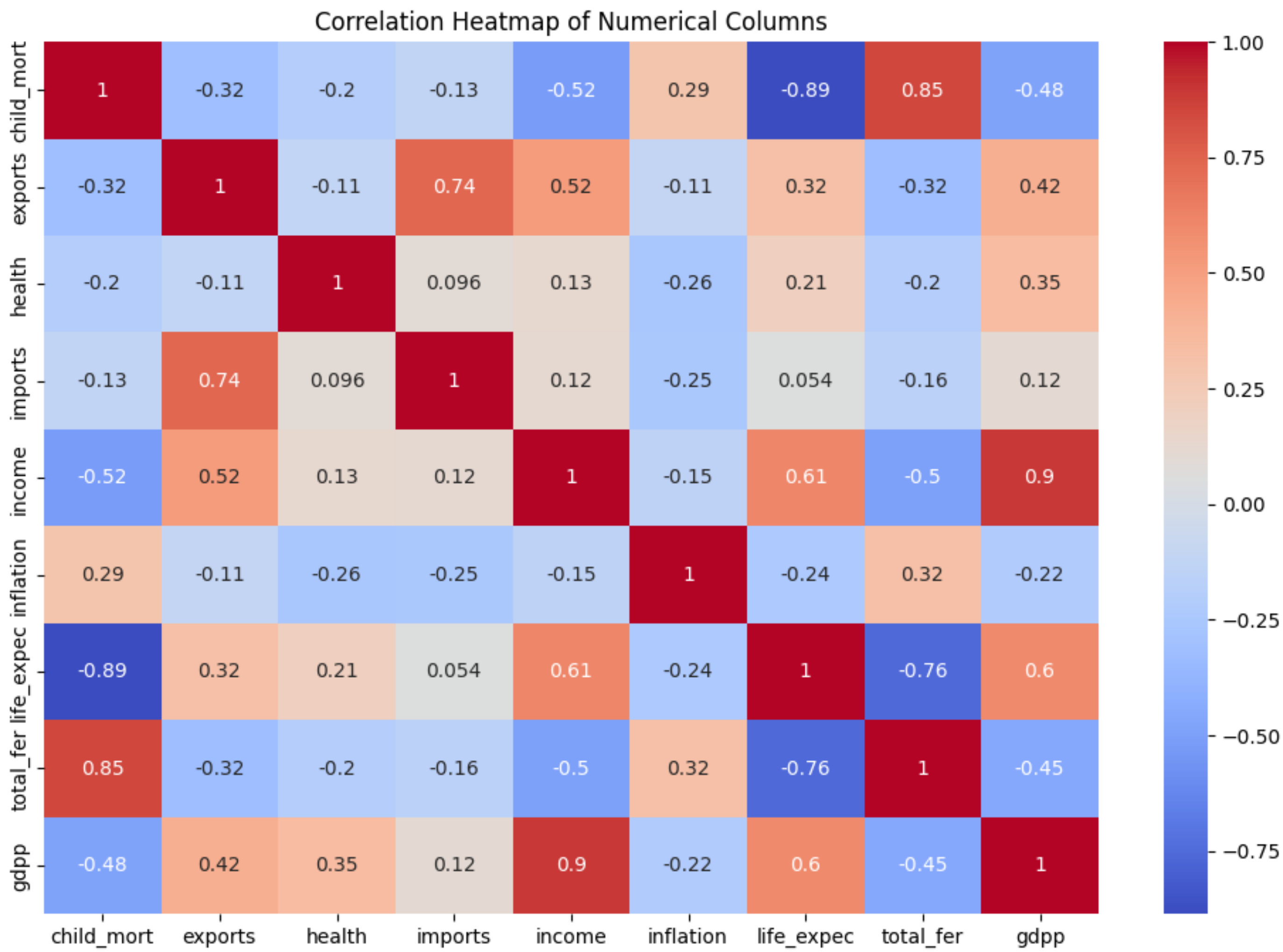

0.04
0.03
0.02
0.01
0.00
Density
20
30
40
50
60
70
80
90
life_expec


Distribution of total_fer
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
0
2
4
6
8
total_fer




```
# # Bivariate Analysis
# sns.pairplot(df)
# plt.show()
```

```
#Correlation Heatmap
df_corr = df[df.select_dtypes(include=['float', 'int']).columns]
plt.figure(figsize=(12, 8))
sns.heatmap(df_corr.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Numerical Columns')
plt.show()
```



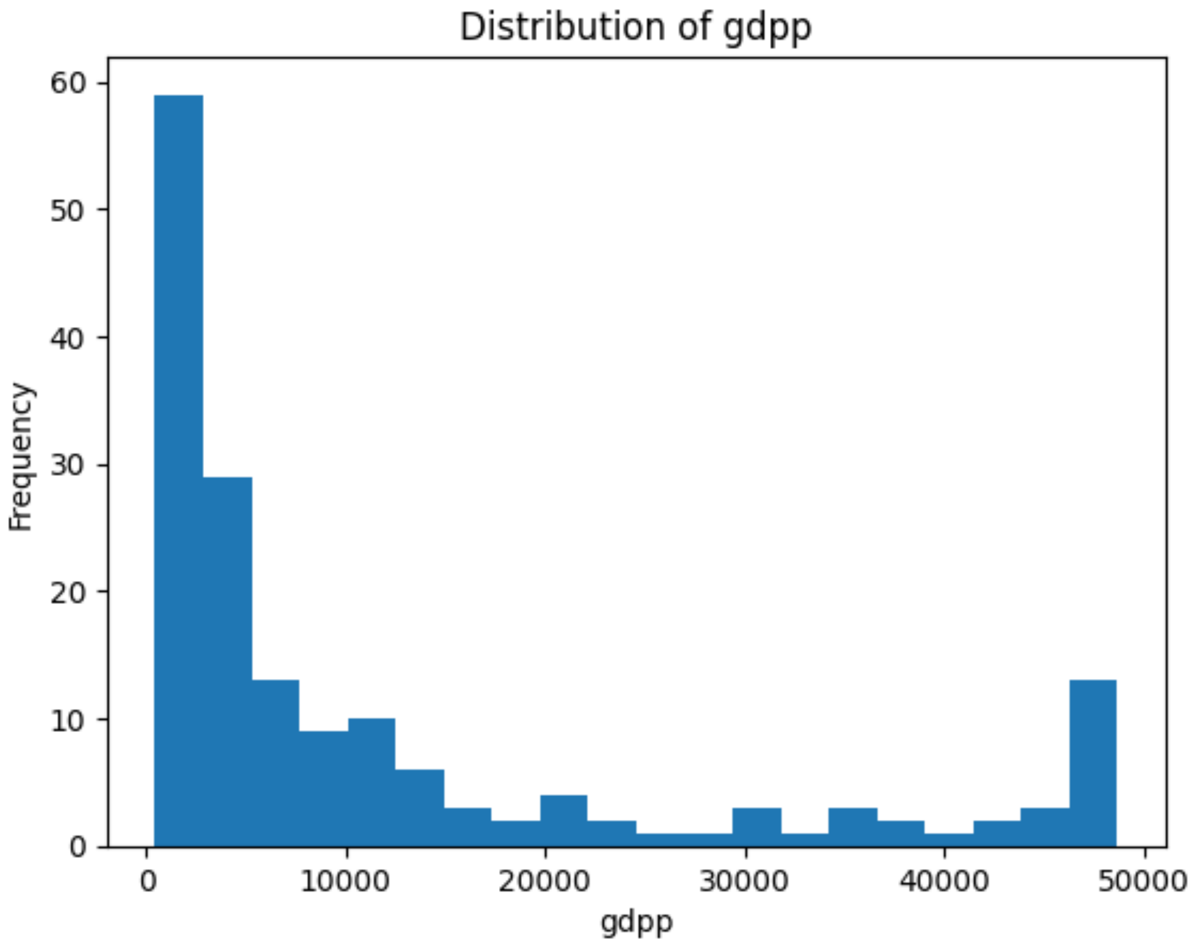
```
df['import_export_ratio'] = (df['exports'] / df['imports'])*100


# Outlier Detection using IQR
for col in df.columns:
  if df[col].dtype != 'object':
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"Outliers in {col}: {outliers.shape[0]}")
```

```
Outliers in child_mort: 4
Outliers in exports: 5
Outliers in health: 2
Outliers in imports: 4
Outliers in income: 8
Outliers in inflation: 5
Outliers in life_expec: 3
Outliers in total_fer: 1
Outliers in gdpp: 25
Outliers in import_export_ratio: 5
```

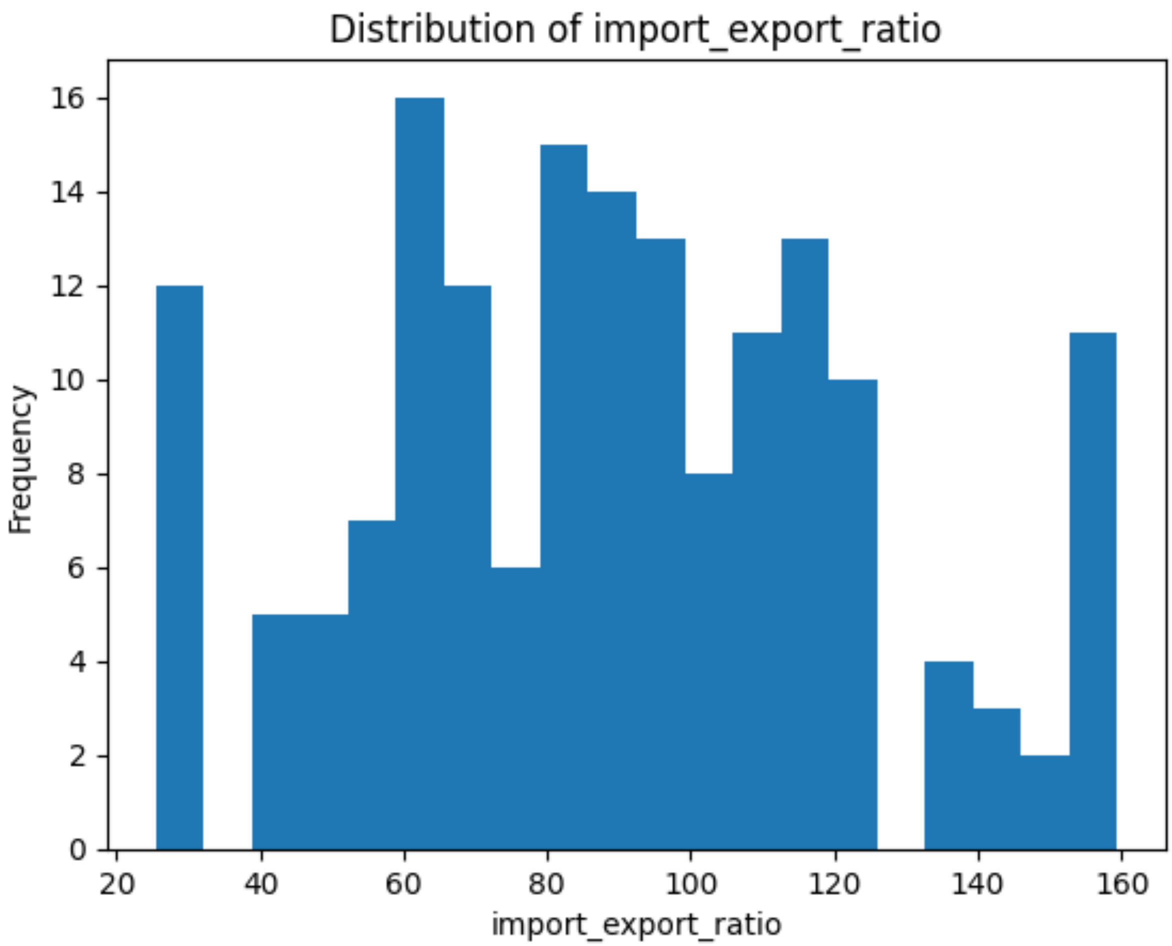
```
from scipy.stats.mstats import winsorize

# Apply winsorization to cap outliers
for col in df.columns:
  if df[col].dtype != 'object':
    df[col] = winsorize(df[col], limits=[0.05, 0.05])  # Cap 5% of outliers on bo

# Verify outlier handling (should show reduced or no outliers)
for col in df.columns:
  if df[col].dtype != 'object':
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"Outliers in {col} after winsorization: {outliers.shape[0]}")#gdp and
```

```
Outliers in child_mort after winsorization: 0
Outliers in exports after winsorization: 0
Outliers in health after winsorization: 0
Outliers in imports after winsorization: 0
Outliers in income after winsorization: 0
Outliers in inflation after winsorization: 0
Outliers in life_expec after winsorization: 0
Outliers in total_fer after winsorization: 0
Outliers in gdpp after winsorization: 25
Outliers in import_export_ratio after winsorization: 0
```

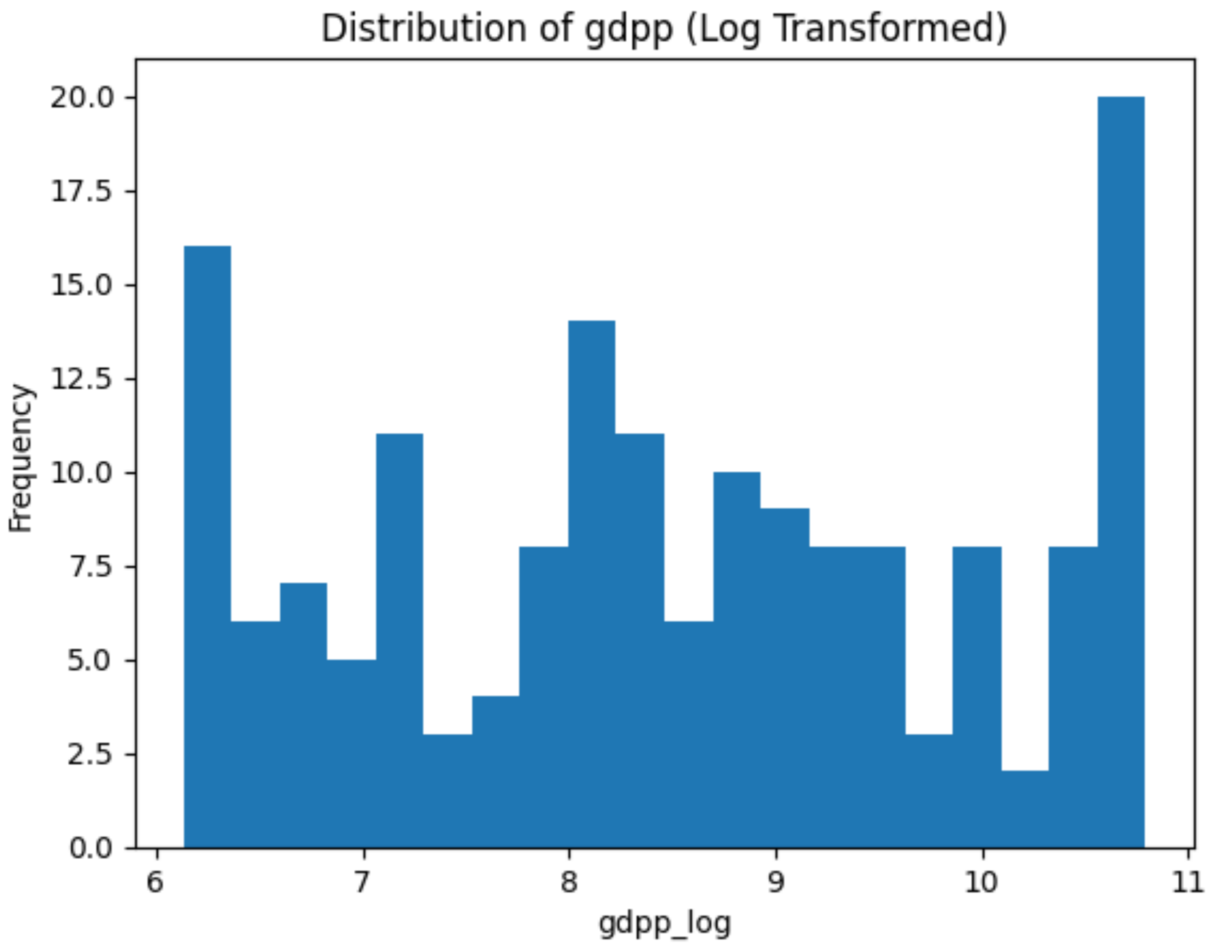
```
# Plot histogram of 'gdpp'
plt.hist(df['gdpp'], bins=20)
plt.title('Distribution of gdpp')
plt.xlabel('gdpp')
plt.ylabel('Frequency')
plt.show()
```



```
# Plot histogram of 'import_export_ratio'
plt.hist(df['import_export_ratio'], bins=20)
plt.title('Distribution of import_export_ratio')
plt.xlabel('import_export_ratio')
plt.ylabel('Frequency')
plt.show()
```



```
# Applying log transformation to 'gdpp'
df['gdpp_log'] = np.log1p(df['gdpp'])

# Plot histograms of transformed variables
plt.hist(df['gdpp_log'], bins=20)
plt.title('Distribution of gdpp (Log Transformed)')
plt.xlabel('gdpp_log')
plt.ylabel('Frequency')
plt.show()
```



```
# Perform t-test
result = stats.ttest_ind(df['health'], df['life_expec'])

# Print results
print(result)

# Interpret results
alpha = 0.05
if result.pvalue < alpha:
    print("Reject the null hypothesis. Increased health spending (% of GDP) leads
else:
    print("Fail to reject the null hypothesis. There is no significant relationsh
```

```
TtestResult(statistic=-97.99581526818585, pvalue=4.2279500495762e-247, df=332.
Reject the null hypothesis. Increased health spending (% of GDP) leads to high
```

```
# Calculate Pearson correlation coefficient and p-value
corr, p_value = stats.pearsonr(df['health'], df['life_expec'])

# Print results
print(f"Pearson correlation coefficient: {corr}")
print(f"P-value: {p_value}")

# Interpret results
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant correlation between
else:
    print("Fail to reject the null hypothesis. There is no significant correlatio
```

```
Pearson correlation coefficient: 0.24947600342499918
P-value: 0.0011488238927149217
Reject the null hypothesis. There is a significant correlation between health
```

```
# Perform t-test
result = stats.ttest_ind(df['total_fer'], df['income'])

# Print results
print(result)

# Interpret results
alpha = 0.05
if result.pvalue < alpha:
    print("Reject the null hypothesis. Countries with higher Total_fertility rate
else:
    print("Fail to reject the null hypothesis. There is no significant relationsh
```

```
TtestResult(statistic=-13.709791347305439, pvalue=3.2950919487094535e-34, df=
Reject the null hypothesis. Countries with higher Total_fertility rates have
```

```
# Calculate Pearson correlation coefficient and p-value
corr, p_value = stats.pearsonr(df['total_fer'], df['income'])

# Print results
print(f"Pearson correlation coefficient: {corr}")
print(f"P-value: {p_value}")

# Interpret results
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant correlation between
else:
    print("Fail to reject the null hypothesis. There is no significant correlatio
```

```
Pearson correlation coefficient: -0.5883441432027093
P-value: 6.239301897978759e-17
Reject the null hypothesis. There is a significant correlation between Total_
```

```
# Perform t-test
result = stats.ttest_ind(df['child_mort'], df['income'])

# Print results
print(result)

# Interpret results
alpha = 0.05
if result.pvalue < alpha:
    print("Reject the null hypothesis. Higher income levels are associated with l
else:
    print("Fail to reject the null hypothesis. There is no significant relationsh
```

```
TtestResult(statistic=-13.680373661937903, pvalue=4.268346375584393e-34, df=3
Reject the null hypothesis. Higher income levels are associated with lower ch
```

```
# Calculate Pearson correlation coefficient and p-value
corr, p_value = stats.pearsonr(df['child_mort'], df['income'])

# Print results
print(f"Pearson correlation coefficient: {corr}")
print(f"P-value: {p_value}")

# Interpret results
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant correlation between
else:
    print("Fail to reject the null hypothesis. There is no significant correlatio
```

```
Pearson correlation coefficient: -0.6353145168605605
P-value: 2.925297942217806e-20
Reject the null hypothesis. There is a significant correlation between Child_
```

```
# Perform t-test
result = stats.ttest_ind(df['inflation'], df['gdpp'])

# Print results
print(result)

# Interpret results
alpha = 0.05
if result.pvalue < alpha:
    print("Reject the null hypothesis. Higher inflation rates are associated with
else:
    print("Fail to reject the null hypothesis. There is no significant relationsh
```

```
TtestResult(statistic=-10.225772328311542, pvalue=1.6140069109571055e-21, df=3
Reject the null hypothesis. Higher inflation rates are associated with lower (
```

```
# Calculate Pearson correlation coefficient and p-value
corr, p_value = stats.pearsonr(df['inflation'], df['gdpp'])

# Print results
print(f"Pearson correlation coefficient: {corr}")
print(f"P-value: {p_value}")

# Interpret results
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis. There is a significant correlation between
else:
    print("Fail to reject the null hypothesis. There is no significant correlatio
```

```
Pearson correlation coefficient: -0.33293875847535354
P-value: 1.1011891306983982e-05
Reject the null hypothesis. There is a significant correlation between gdpp ar
```

ML Model

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   country              167 non-null    object 
 1   child_mort           167 non-null    float64
 2   exports              167 non-null    float64
 3   health               167 non-null    float64
 4   imports              167 non-null    float64
 5   income               167 non-null    int64  
 6   inflation            167 non-null    float64
 7   life_expec           167 non-null    float64
 8   total_fer            167 non-null    float64
 9   gdpp                 167 non-null    int64  
 10  import_export_ratio  167 non-null    float64
 11  gdpp_log             167 non-null    float64
dtypes: float64(9), int64(2), object(1)
memory usage: 15.8+ KB
```

```
# Define a function to categorize countries into regions
def assign_region(country):
  if country in ['Afghanistan', 'Bangladesh', 'Bhutan', 'India', 'Maldives', 'Nep
    return 'South Asia'
  elif country in ['Brunei', 'Cambodia', 'Indonesia', 'Laos', 'Malaysia', 'Myanma
    return 'Southeast Asia'
  elif country in ['China', 'Hong Kong', 'Japan', 'Macau', 'Mongolia', 'North Kor
    return 'East Asia'
  elif country in ['Kazakhstan', 'Kyrgyzstan', 'Tajikistan', 'Turkmenistan', 'Uzb
    return 'Central Asia'
  elif country in ['Bahrain', 'Cyprus', 'Egypt', 'Iran', 'Iraq', 'Israel', 'Jorda
    return 'Middle East'
  elif country in ['Algeria', 'Angola', 'Benin', 'Botswana', 'Burkina Faso', 'Bur
    return 'Africa'
  elif country in ['Albania', 'Andorra', 'Armenia', 'Austria', 'Azerbaijan', 'Bel
    return 'Europe'
  elif country in ['Antigua and Barbuda', 'Argentina', 'Bahamas', 'Barbados', 'Be
    return 'Americas'
  elif country in ['Australia', 'Fiji', 'Kiribati', 'Marshall Islands', 'Micrones
    return 'Oceania'
  else:
    return 'Other'

# Apply the function to create the 'regions' column
df['regions'] = df['country'].apply(assign_region)
```

```
df.regions.value_counts()
```

regions	count
Africa	43
Europe	40
Americas	26
Middle East	15
Southeast Asia	10
Other	9
South Asia	8
Oceania	8
East Asia	4
Central Asia	4

dtype: int64

```
# One Hot Encoding
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Ensure the 'regions' column exists
if 'regions' not in df.columns:
    raise KeyError("The column 'regions' does not exist in the DataFrame. Check y

# Ensure no missing values in the 'regions' column
df['regions'] = df['regions'].fillna('Unknown')  # Fills missing values if any

# Apply OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, drop='first')  # Use sparse_output f
encoded_regions = encoder.fit_transform(df[['regions']])

# Create a DataFrame from the encoded features
encoded_regions_df = pd.DataFrame(
    encoded_regions,
    columns=encoder.get_feature_names_out(['regions'])
```

```
)

# Concatenate the encoded features with the original DataFrame
df = pd.concat([df, encoded_regions_df], axis=1)

# Drop the original 'regions' column
df = df.drop('regions', axis=1)

print("One-hot encoding completed successfully!")
print(df.head())  # Display the updated DataFrame
```

```
One-hot encoding completed successfully!
              country  child_mort  exports  health  imports  income  \
0         Afghanistan        90.2     12.0    7.58     44.9    1610
1             Albania        16.6     28.0    6.55     48.6    9930
2             Algeria        27.3     38.4    4.17     31.4   12900
3              Angola       116.0     62.3    2.85     42.9    5900
4  Antigua and Barbuda        10.3     45.5    6.03     58.9   19100

   inflation  life_expec  total_fer   gdpp  ...  gdpp_log  regions_Americas  \
0       9.44        56.2       5.82    553  ...  6.317165               0.0
1       4.49        76.3       1.65   4090  ...  8.316545               0.0
2      16.10        76.5       2.89   4460  ...  8.403128               0.0
3      20.90        60.1       5.87   3530  ...  8.169336               0.0
4       1.44        76.8       2.13  12200  ...  9.409273               1.0

   regions_Central Asia  regions_East Asia  regions_Europe  \
0                   0.0                0.0             0.0
1                   0.0                0.0             1.0
2                   0.0                0.0             0.0
3                   0.0                0.0             0.0
4                   0.0                0.0             0.0

   regions_Middle East  regions_Oceania  regions_Other  regions_South Asia  \
0                  0.0              0.0            0.0                 1.0
1                  0.0              0.0            0.0                 0.0
2                  0.0              0.0            0.0                 0.0
3                  0.0              0.0            0.0                 0.0
4                  0.0              0.0            0.0                 0.0

   regions_Southeast Asia
0                     0.0
1                     0.0
2                     0.0
3                     0.0
4                     0.0

[5 rows x 21 columns]
```

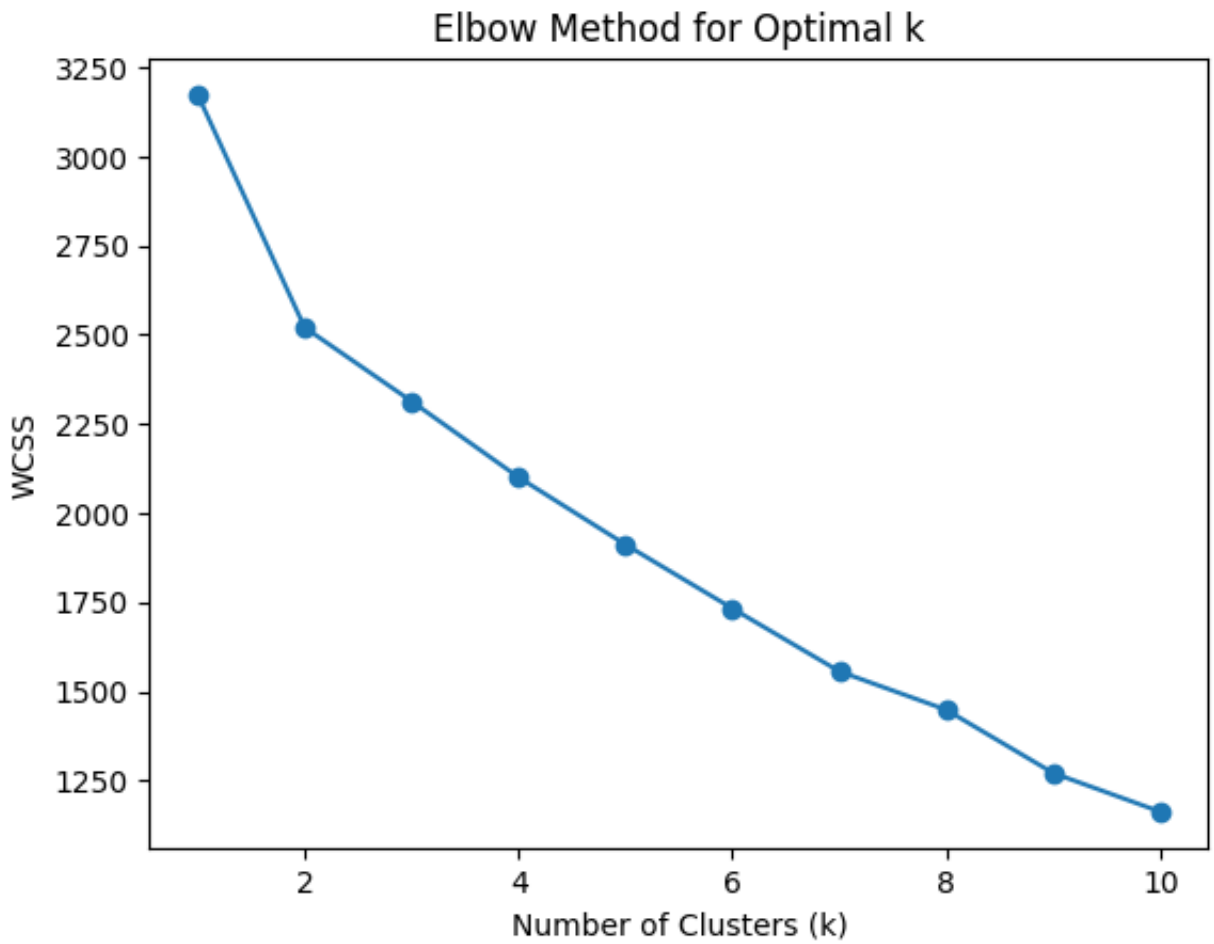
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   country                 167 non-null    object 
 1   child_mort              167 non-null    float64
 2   exports                 167 non-null    float64
 3   health                  167 non-null    float64
 4   imports                 167 non-null    float64
 5   income                  167 non-null    int64  
 6   inflation               167 non-null    float64
 7   life_expec              167 non-null    float64
 8   total_fer               167 non-null    float64
 9   gdpp                    167 non-null    int64  
 10  import_export_ratio     167 non-null    float64
 11  gdpp_log                167 non-null    float64
 12  regions_Americas        167 non-null    float64
 13  regions_Central Asia    167 non-null    float64
 14  regions_East Asia       167 non-null    float64
 15  regions_Europe          167 non-null    float64
 16  regions_Middle East     167 non-null    float64
 17  regions_Oceania         167 non-null    float64
 18  regions_Other           167 non-null    float64
 19  regions_South Asia      167 non-null    float64
 20  regions_Southeast Asia  167 non-null    float64
dtypes: float64(18), int64(2), object(1)
memory usage: 27.5+ KB
```

```
# Select numerical columns for scaling
numerical_cols = df.select_dtypes(include=['float', 'int']).columns

# Initialize the scaler
scaler = StandardScaler()  # or MinMaxScaler() depending on your preference

# Fit and transform the numerical columns
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

```
import matplotlib.pyplot as plt
# Assuming 'df' is your DataFrame with relevant features for clustering

# Select numerical features for clustering
X = df[['child_mort', 'exports', 'health', 'imports', 'income', 'inflation',
```

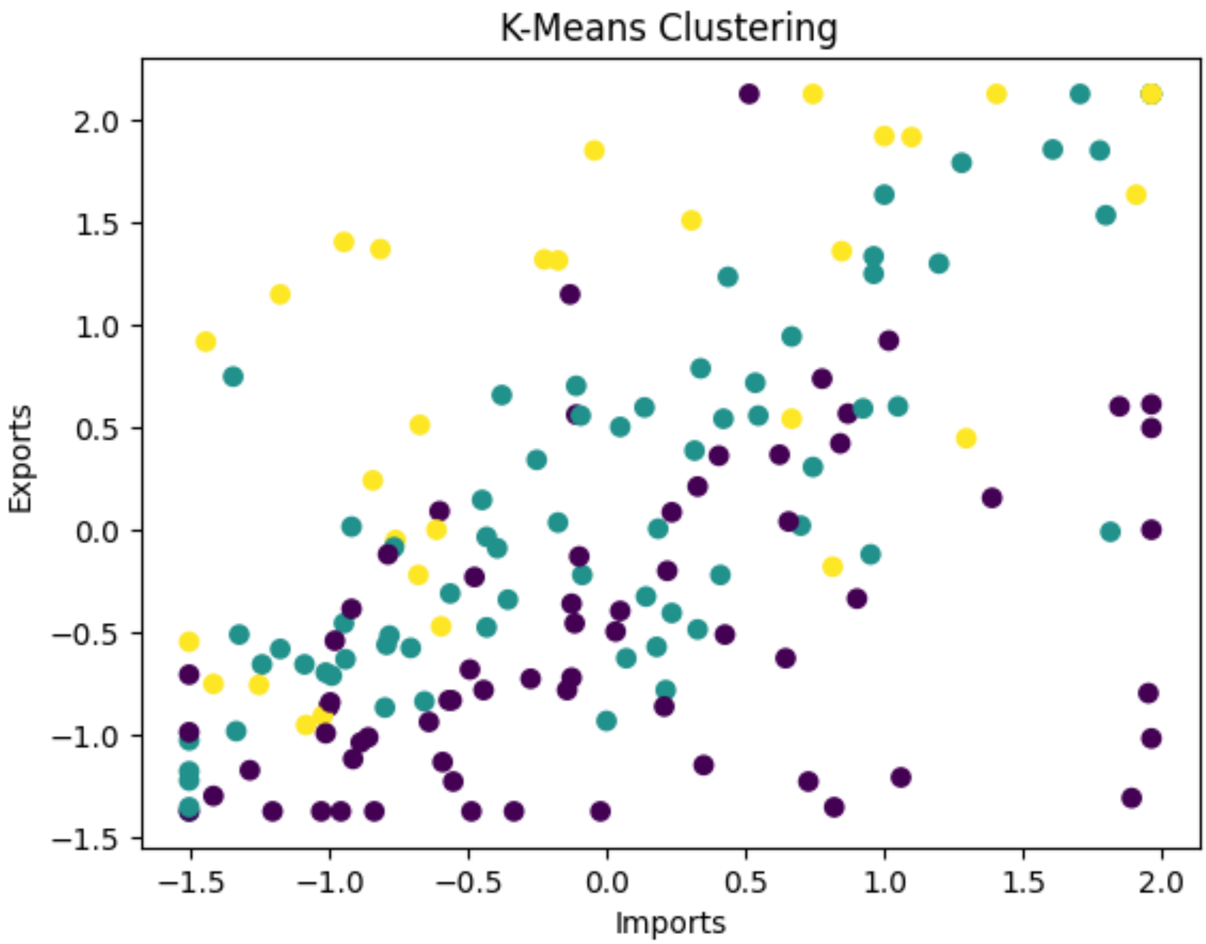
```
       'life_expec', 'total_fer','import_export_ratio', 'gdpp_log',
       'regions_Americas', 'regions_Central Asia', 'regions_East Asia',
       'regions_Europe', 'regions_Middle East', 'regions_Oceania',
       'regions_Other', 'regions_South Asia', 'regions_Southeast Asia']]

# Initialize list to store WCSS (Within-Cluster Sum of Squares)
wcss = []

# Try different values of k (number of clusters)
for i in range(1, 11):
  kmeans = KMeans(n_clusters=i, random_state=42)
  kmeans.fit(X)
  wcss.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.show()
```




```
# Based on the Elbow method, choose the optimal k (let's assume k=3)
optimal_k = 3

# Initialize KMeans with the optimal k
kmeans = KMeans(n_clusters=optimal_k, random_state=42)

# Fit the model to the data
kmeans.fit(X)

# Get cluster labels for each data point
df['Cluster'] = kmeans.labels_

# Print the cluster centers
print(kmeans.cluster_centers_)

# Visualize the clusters (example using two features)
plt.scatter(df['imports'], df['exports'], c=df['Cluster'], cmap='viridis')
plt.title('K-Means Clustering')
plt.xlabel('Imports')
```

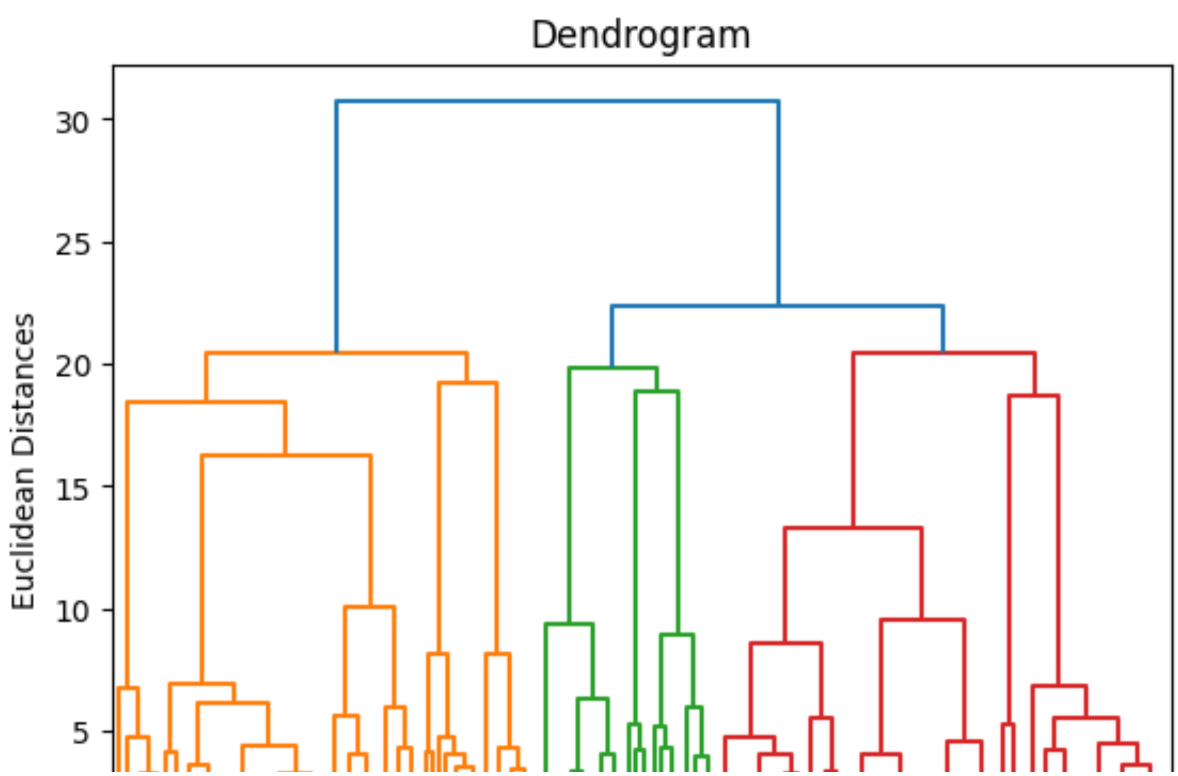
```
plt.ylabel('Exports')
plt.show()
```

```
[[ 9.26227057e-01 -5.03722974e-01 -2.74155474e-01 -1.87725906e-04
  -8.22984564e-01  2.47450284e-01 -9.71064453e-01  9.03330152e-01
  -5.94518621e-01 -9.52721222e-01 -2.64748342e-01  3.85784998e-02
  -5.90367951e-02 -5.61213533e-01 -3.14140431e-01  1.95014794e-01
   2.24022598e-01  2.64902070e-01 -7.53364873e-04]
 [-7.28442486e-01  1.52517322e-01  6.02429274e-01  1.91612419e-02
   5.25632345e-01 -4.93780426e-01  7.45235904e-01 -7.72863163e-01
   1.65374293e-01  6.95383809e-01  3.98034588e-01 -1.56652090e-01
   1.23643257e-01  7.77681610e-01 -3.14140431e-01 -9.05246477e-02
  -1.12135693e-01 -2.24308862e-01 -2.52377233e-01]
 [-3.68874627e-01  7.69107557e-01 -7.93387749e-01 -4.42903100e-02
   6.11523389e-01  5.99515359e-01  4.29826835e-01 -2.14089958e-01
   9.41884901e-01  5.05181843e-01 -3.37476075e-01  2.79362894e-01
  -1.56652090e-01 -5.61213533e-01  1.43457464e+00 -2.24308862e-01
  -2.38667185e-01 -6.82272787e-02  5.90562724e-01]]
```

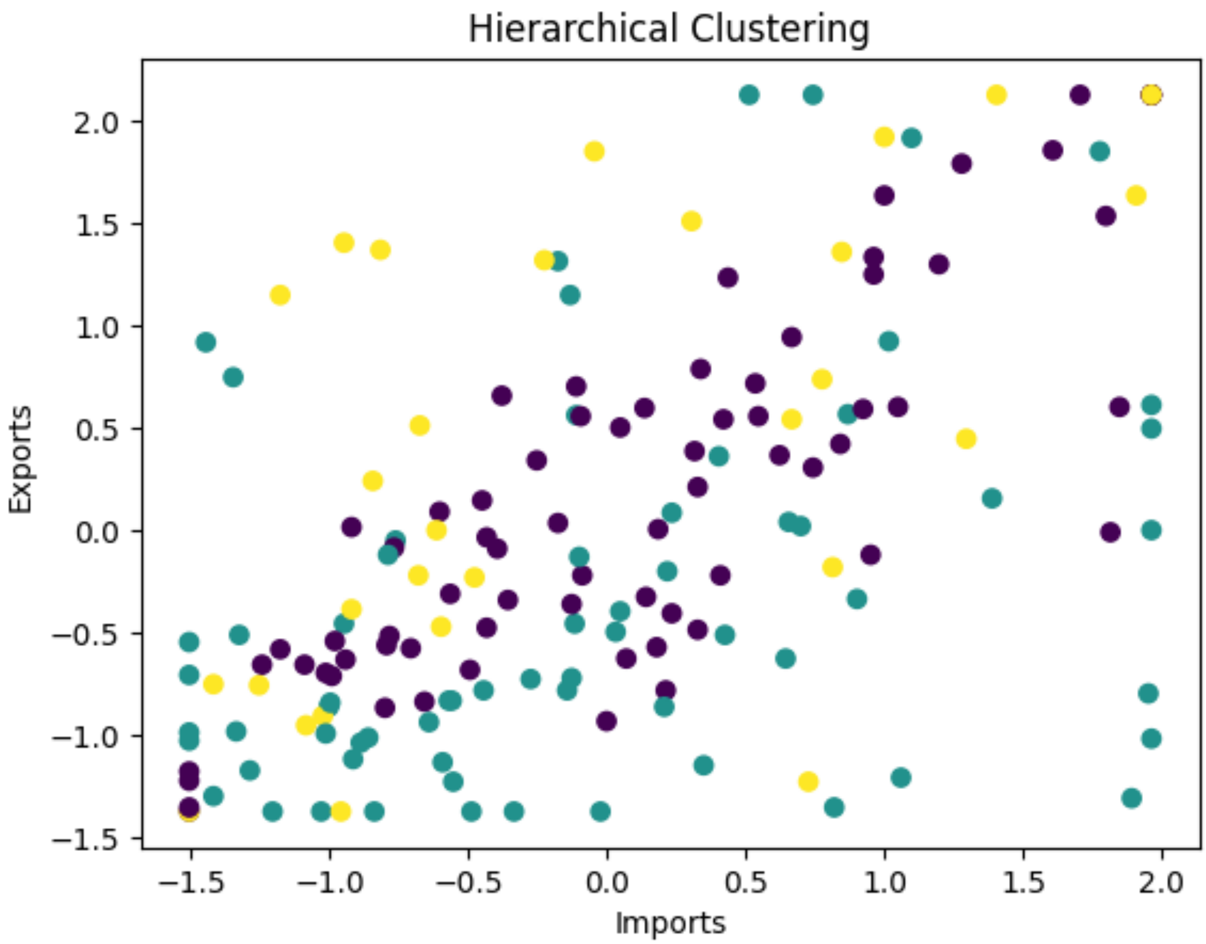



```
# Create a dendrogram to visualize the hierarchical structure
import scipy.cluster.hierarchy as sch
```

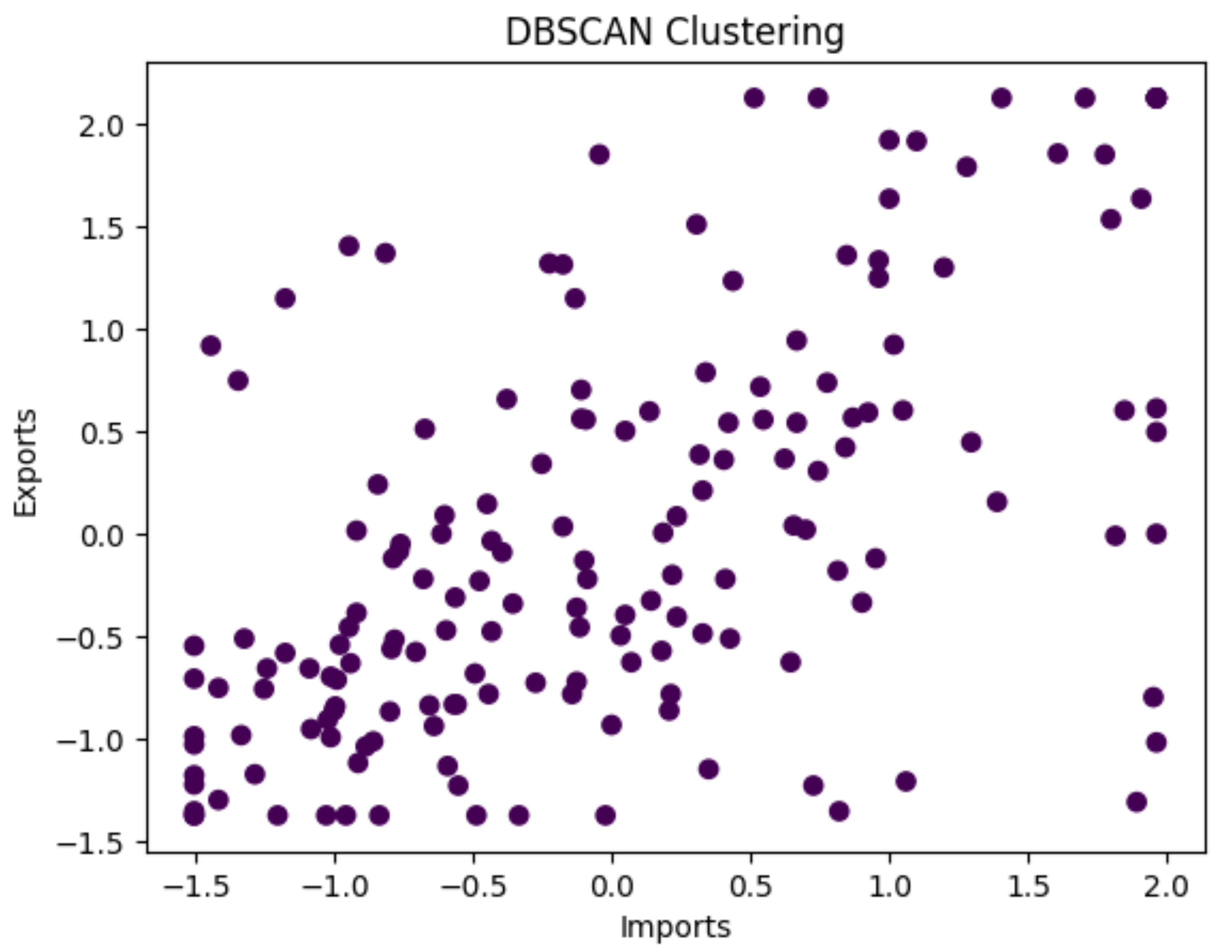
```
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

# Dendrogram to determine the number of clusters
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
plt.title('Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distances')
plt.show()

# Perform Agglomerative Clustering (choose the number of clusters based on the de
# Let's assume we want 3 clusters
hc = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')  #
y_hc = hc.fit_predict(X)

# Add cluster labels to your DataFrame
df['Cluster_HC'] = y_hc

# Visualize the clusters (example using two features)
plt.scatter(df['imports'], df['exports'], c=df['Cluster_HC'], cmap='viridis')
plt.title('Hierarchical Clustering')
plt.xlabel('Imports')
plt.ylabel('Exports')
plt.show()
```




0
Data Points


Hierarchical Clustering
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Exports
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Imports

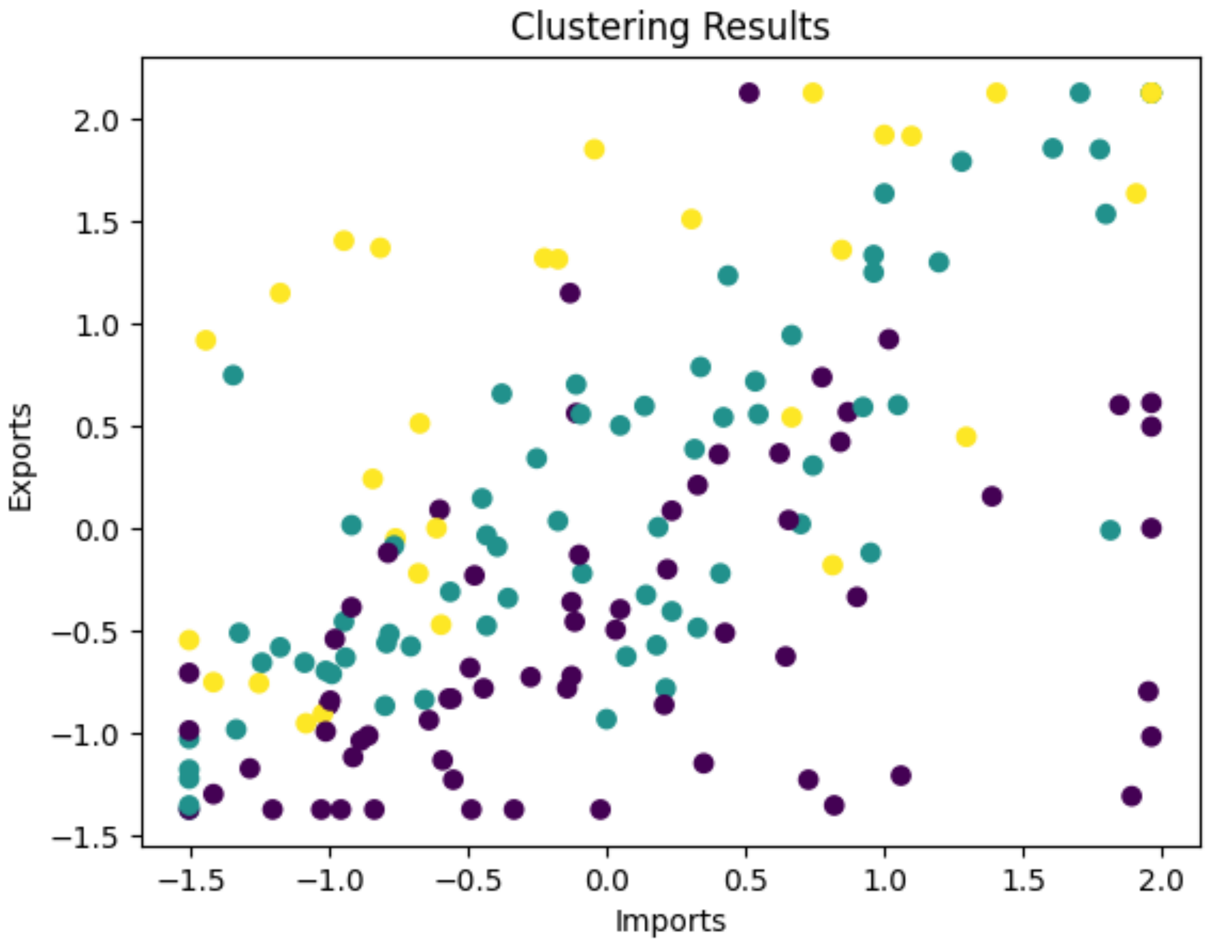
```
# Initialize DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Adjust eps and min_samples as needed

# Fit the model to the data
y_dbscan = dbscan.fit_predict(X)

# Add cluster labels to your DataFrame
df['Cluster_DBSCAN'] = y_dbscan

# Visualize the clusters (example using two features)
plt.scatter(df['imports'], df['exports'], c=df['Cluster_DBSCAN'], cmap='viridis')
plt.title('DBSCAN Clustering')
plt.xlabel('Imports')
plt.ylabel('Exports')
plt.show()
```



```
from sklearn.metrics import silhouette_score

# Calculate Silhouette Coefficient for KMeans
silhouette_kmeans = silhouette_score(X, df['Cluster'])
print("Silhouette Coefficient for KMeans:", silhouette_kmeans)

# Calculate Silhouette Coefficient for Agglomerative Clustering
silhouette_hc = silhouette_score(X, df['Cluster_HC'])
print("Silhouette Coefficient for Hierarchical Clustering:", silhouette_hc)

# Calculate Silhouette Coefficient for DBSCAN (excluding noise points)
core_samples_mask = np.zeros_like(y_dbscan, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True
labels_dbscan = y_dbscan[core_samples_mask]
X_dbscan = X[core_samples_mask]
if len(set(labels_dbscan)) > 1:  # Check if there are at least two clusters (excl
    silhouette_dbscan = silhouette_score(X_dbscan, labels_dbscan)
    print("Silhouette Coefficient for DBSCAN:", silhouette_dbscan)
else:
    print("Silhouette Coefficient for DBSCAN cannot be calculated as there are le
```

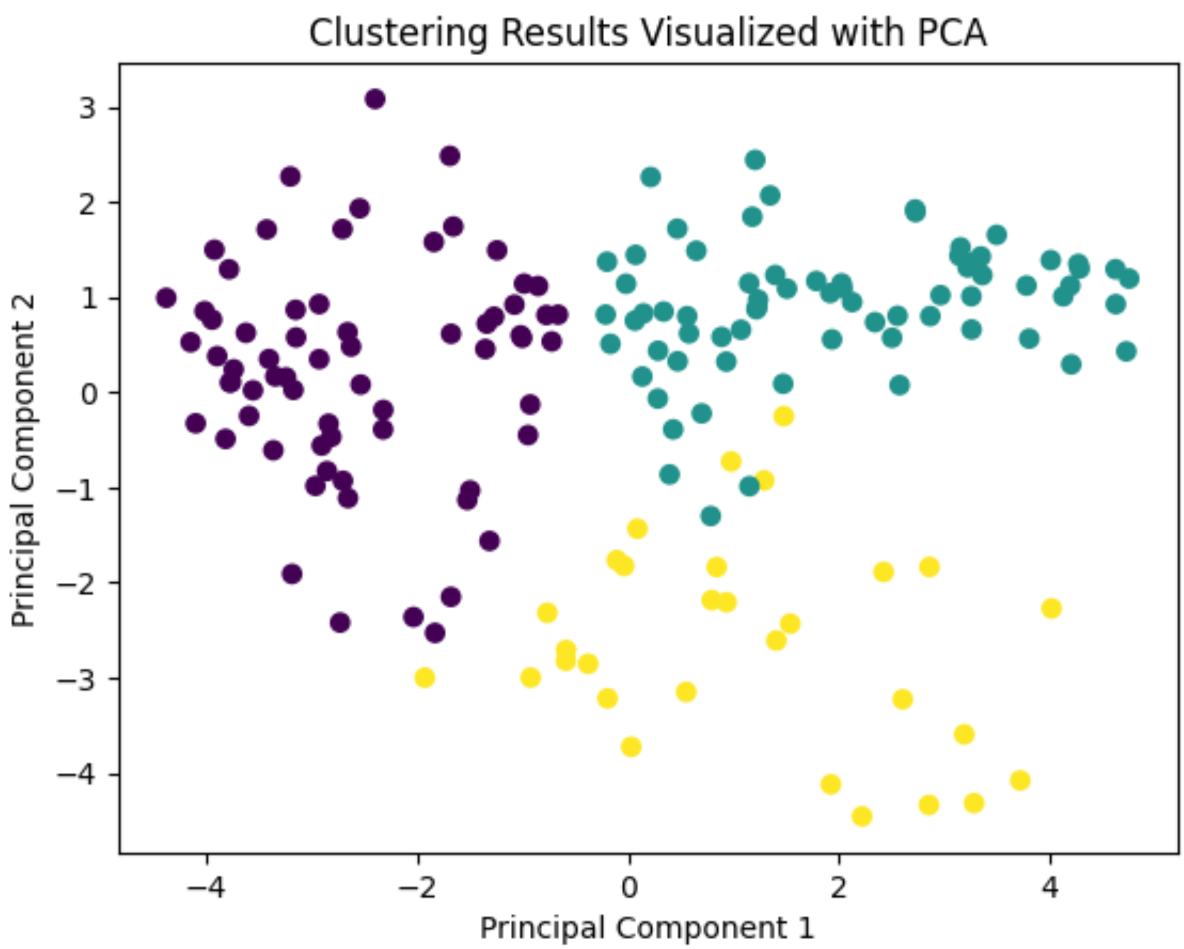
```
Silhouette Coefficient for KMeans: 0.1998189829076175
Silhouette Coefficient for Hierarchical Clustering: 0.18462409237755378
Silhouette Coefficient for DBSCAN cannot be calculated as there are less than
```

```
# Visualize the clusters (example using two features)
plt.scatter(df['imports'], df['exports'], c=df['Cluster'], cmap='viridis')
plt.title('Clustering Results')
plt.xlabel('Imports')
plt.ylabel('Exports')
plt.show()
```




```
from sklearn.decomposition import PCA

# Select numerical features for PCA
X = df.select_dtypes(include=['float', 'int'])

# Initialize PCA with 2 components
pca = PCA(n_components=2)

# Fit and transform the data
X_pca = pca.fit_transform(X)
```

```
# Create a DataFrame from the PCA results
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])

# Add cluster labels to the PCA DataFrame
pca_df['Cluster'] = df['Cluster']  # Replace 'Cluster' with the actual column nam

# Visualize the clusters using PCA
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['Cluster'], cmap='viridis')
plt.title('Clustering Results Visualized with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```




```
# Get the cluster centers
cluster_centers = kmeans.cluster_centers_

# Calculate the absolute values of the cluster centers
abs_centers = np.abs(cluster_centers)
```

```
# Sort features within each cluster based on their absolute values in the cluster
ordered_features = []
for i in range(3):  # Assuming optimal_k is the number of clusters
    ordered_features.append(np.argsort(abs_centers[i])[::-1])

# Print the most important features for each cluster
for i in range(3):
    print(f"Cluster {i}:")
    for feature_idx in ordered_features[i]:
        print(f"  - {X.columns[feature_idx]}")
```

```
Cluster 0:
  - life_expec
  - import_export_ratio
  - child_mort
  - total_fer
  - income
  - gdpp
  - regions_East Asia
  - exports
  - regions_Europe
  - health
  - regions_Other
  - gdpp_log
  - inflation
  - regions_Oceania
  - regions_Middle East
  - regions_Central Asia
  - regions_Americas
  - regions_South Asia
  - imports
Cluster 1:
  - regions_East Asia
  - total_fer
  - life_expec
  - child_mort
  - import_export_ratio
  - health
  - income
  - inflation
  - gdpp_log
  - regions_Europe
  - regions_South Asia
  - regions_Other
  - gdpp
  - regions_Americas
```

```
    - exports
    - regions_Central Asia
    - regions_Oceania
    - regions_Middle East
    - imports
  Cluster 2:
    - regions_Europe
    - gdpp
    - health
    - exports
    - income
    - inflation
    - regions_South Asia
    - regions_East Asia
    - import_export_ratio
    - life_expec
    - child_mort
    - gdpp_log
    - regions_Americas
    - regions_Oceania
    - regions_Middle East
    - total_fer
    - regions_Central Asia
    - regions_Other
```

```
# Get the features of the given country
country_name = input("Enter a country")  # Replace with the actual country name
country_data = df[df['country'] == country_name][['child_mort', 'exports', 'healt
       'life_expec', 'total_fer','import_export_ratio', 'gdpp_log',
       'regions_Americas', 'regions_Central Asia', 'regions_East Asia',
       'regions_Europe', 'regions_Middle East', 'regions_Oceania',
       'regions_Other', 'regions_South Asia', 'regions_Southeast Asia']].values

# Predict the cluster for the given country
country_cluster = kmeans.predict(country_data)[0]

# Find countries in the same cluster
similar_countries = df[df['Cluster'] == country_cluster]['country'].tolist()

# Remove the given country from the list of similar countries
similar_countries.remove(country_name)

print(f"Countries similar to {country_name} based on KMeans clustering: {similar_
```

```
Enter a countryIndia
Countries similar to India based on KMeans clustering: ['Afghanistan', 'Angola
```

```
import pickle
import os
#FIle Path
file_path = 'Clustering Countries for Strategic Aid Allocation.pkl'

# Save the KMeans model to the pickle file
with open(file_path, 'wb') as file:
  pickle.dump(kmeans, file)
```

Recommendations

1. **Invest in Healthcare:**
   - Focus on reducing high child mortality rates by funding immunization programs, enhancing maternal healthcare, and improving basic health infrastructure. These initiatives are critical for saving lives and fostering a healthier future for vulnerable populations.
2. **Collaborate to Strengthen Local Health Systems:**
   - Partner with local organizations to enhance healthcare facilities, supply medical resources, and train healthcare workers in regions with low life expectancy. A community-driven approach ensures sustainable improvements in health outcomes.
3. **Prioritize Long-Term Development Projects:**
   - Allocate funds toward clean water access, education, and renewable energy projects to address the root causes of poverty and foster long-term economic and social stability.
4. **Support Education and Skill Development:**
   - Promote early childhood education, health awareness campaigns, and vocational training programs in underserved areas. By providing scholarships and enhancing teacher training, these initiatives can empower communities and drive sustainable change.
5. **Enhance Nutrition and Food Security:**
   - Address high child mortality rates through sustainable agriculture programs, nutritional education, and food supplements. Implementing school meal initiatives can improve both health and educational outcomes, laying a strong foundation for future progress.