Section C

```
#@title Importing necessary libraries

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

#@title Reading the Dataset

df = pd.read_csv('Country-data.csv').drop(['country'], axis=1)
original_df = pd.read_csv('Country-data.csv')

df

     child_mort  exports  health  imports  income  inflation
life_expec  \
0          90.2     10.0    7.58     44.9    1610       9.44
56.2
1          16.6     28.0    6.55     48.6    9930       4.49
76.3
2          27.3     38.4    4.17     31.4   12900      16.10
76.5
3         119.0     62.3    2.85     42.9    5900      22.40
60.1
4          10.3     45.5    6.03     58.9   19100       1.44
76.8
..          ...      ...     ...      ...     ...        ...          .
..
162        29.2     46.6    5.25     52.7    2950       2.62
63.0
163        17.1     28.5    4.91     17.6   16500      45.90
75.4
164        23.3     72.0    6.84     80.2    4490      12.10
73.1
165        56.3     30.0    5.18     34.4    4480      23.60
67.5
166        83.1     37.0    5.89     30.9    3280      14.00
52.0

     total_fer   gdpp
0         5.82    553
1         1.65   4090
2         2.89   4460
3         6.16   3530
4         2.13  12200
```

```
..          ...    ...
162        3.50   2970
163        2.47  13500
164        1.95   1310
165        4.67   1310
166        5.40   1460

[167 rows x 9 columns]
```

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

```
child_mort    0
exports       0
health        0
imports       0
income        0
inflation     0
life_expec    0
total_fer     0
gdpp          0
dtype: int64
```

```
print(df.dtypes)
```

```
child_mort    float64
exports       float64
health        float64
imports       float64
income          int64
inflation     float64
life_expec    float64
total_fer     float64
gdpp            int64
dtype: object
```

```
for cols in df.columns:
  print(f'{cols}: {df[cols].unique()}')
  print()
```

```
child_mort: [ 90.2  16.6  27.3 119.   10.3  14.5  18.1   4.8   4.3
39.2  13.8   8.6
  49.4  14.2   5.5   4.5  18.8 111.   42.7  46.6   6.9  52.5  19.8
10.5
  10.8 116.   93.6  44.4 108.    5.6  26.5 149.  150.    8.7  15.7
18.6
  88.2  63.9  10.2   3.6   3.4   4.1  34.4  25.1  29.1  19.2  55.2
24.1
   3.    4.2  63.7  80.3  16.5  74.7   3.9  14.6  35.4 109.  114.
37.6
 208.    6.    2.6  58.8  33.3  19.3  36.9   4.6   4.    3.2  21.1
21.5
```

```
  62.2  62.7  29.6  78.9   7.8  99.7  89.3   6.1   2.8  10.4  90.5
7.9
  13.2 137.    6.8  97.4  15.   40.   17.2  26.1  33.5 101.   64.4
56.
  47.    6.2 123.  130.   11.7  92.1  19.7  20.3  31.9   9.   11.5
10.
  63.6  18.9  66.8   7.6  14.4 160.    7.   28.1  53.7   3.8  11.2
20.7
  76.7  52.4  71.9  14.9  62.6  90.3  17.4  19.1  62.   81.    5.2
7.3
  10.6  36.3  29.2  17.1  23.3  56.3  83.1]

exports: [1.00e+01 2.80e+01 3.84e+01 6.23e+01 4.55e+01 1.89e+01
2.08e+01 1.98e+01
 5.13e+01 5.43e+01 3.50e+01 6.95e+01 1.60e+01 3.95e+01 5.14e+01
7.64e+01
 5.82e+01 2.38e+01 4.25e+01 4.12e+01 2.97e+01 4.36e+01 1.07e+01
6.74e+01
 5.02e+01 1.92e+01 8.92e+00 5.41e+01 2.22e+01 2.91e+01 3.27e+01
1.18e+01
 3.68e+01 3.77e+01 2.63e+01 1.59e+01 1.65e+01 4.11e+01 8.51e+01
3.32e+01
 5.06e+01 3.76e+01 6.60e+01 5.05e+01 2.27e+01 2.79e+01 2.13e+01
2.69e+01
 8.58e+01 4.79e+00 7.51e+01 5.78e+01 3.87e+01 2.68e+01 5.77e+01
4.23e+01
 2.95e+01 2.21e+01 2.58e+01 3.03e+01 1.49e+01 1.53e+01 8.18e+01
5.34e+01
 2.26e+01 2.43e+01 2.44e+01 3.94e+01 1.03e+02 2.52e+01 3.13e+01
1.50e+01
 4.83e+01 4.42e+01 2.07e+01 1.33e+01 6.67e+01 5.16e+01 3.54e+01
5.37e+01
 3.58e+01 1.91e+01 6.56e+01 6.53e+01 1.75e+02 3.98e+01 2.50e+01
2.28e+01
 8.69e+01 7.76e+01 1.53e+02 5.07e+01 5.12e+01 2.35e+01 3.92e+01
4.67e+01
 3.70e+01 3.22e+01 3.15e+01 1.09e-01 4.78e+01 9.58e+00 7.20e+01
2.53e+01
 3.97e+01 6.57e+01 1.35e+01 7.00e+01 5.51e+01 2.78e+01 3.48e+01
4.01e+01
 2.99e+01 3.26e+01 2.92e+01 1.20e+01 4.96e+01 2.49e+01 3.29e+01
9.38e+01
 1.68e+01 2.00e+02 7.63e+01 6.43e+01 4.93e+01 2.86e+01 4.94e+01
2.55e+01
 1.96e+01 1.97e+01 5.25e+01 4.62e+01 6.40e+01 1.87e+01 6.65e+01
2.20e+00
 4.02e+01 1.24e+01 2.04e+01 1.71e+01 4.71e+01 7.77e+01 2.82e+01
3.17e+01
 4.66e+01 2.85e+01 3.00e+01]
```

```
health: [ 7.58  6.55  4.17  2.85  6.03  8.1   4.4   8.73 11.    5.88
7.89  4.97
  3.52  7.97  5.61 10.7   5.2   4.1   4.84 11.1   8.3   9.01  2.84
6.87
  6.74 11.6   5.68  5.13 11.3   4.09  3.98  4.53  7.96  5.07  7.59
4.51
  7.91  2.46 10.9   5.3   7.76  5.97  7.88 11.4   6.22  8.06  4.66
6.91
  4.48  2.66  4.86  8.95 11.9   3.5   5.69 10.1   5.22 10.3   5.86
6.85
  4.93  8.5   5.38  7.33  9.4   4.05  2.61  5.6   8.41  9.19  7.63
9.53
  4.81  9.49  8.04  4.29  4.75  2.63  6.18  4.47  6.68  7.03 11.8
3.88
  7.04  7.77  7.09  3.77  6.59  4.39  6.33  4.98  8.65  4.41  6.
14.2
 11.7   5.44  9.11  5.21  1.97  6.78  5.25  5.16  9.48  2.77  2.2
5.87
  5.08  3.61  7.46  1.81  5.58 10.5   6.47  5.66 10.4   3.4  13.1
3.96
  8.79  9.41  8.55  8.94  6.93  9.54  2.94  6.32  7.01  9.63 11.5
5.98
  6.01  9.12  7.65  6.21  2.5   7.72  3.66  9.64 17.9   8.35  5.81
4.91
  6.84  5.18  5.89]

imports: [4.49e+01 4.86e+01 3.14e+01 4.29e+01 5.89e+01 1.60e+01
4.53e+01 2.09e+01
 4.78e+01 2.07e+01 4.37e+01 5.09e+01 2.18e+01 4.87e+01 6.45e+01
7.47e+01
 5.75e+01 3.72e+01 7.07e+01 3.43e+01 5.13e+01 1.18e+01 2.80e+01
5.30e+01
 2.96e+01 3.92e+01 5.95e+01 2.70e+01 3.10e+01 6.18e+01 2.65e+01
4.35e+01
 3.13e+01 2.26e+01 1.78e+01 5.17e+01 4.96e+01 5.47e+01 3.50e+01
4.33e+01
 3.81e+01 6.29e+01 4.36e+01 3.33e+01 3.24e+01 2.66e+01 4.66e+01
2.33e+01
 6.87e+01 6.39e+01 3.74e+01 2.81e+01 1.89e+01 4.27e+01 5.28e+01
3.71e+01
 4.59e+01 3.07e+01 4.92e+01 3.63e+01 4.32e+01 3.52e+01 7.91e+01
6.47e+01
 7.65e+01 2.71e+01 2.24e+01 1.94e+01 3.41e+01 8.65e+01 3.29e+01
2.72e+01
 1.36e+01 6.90e+01 2.99e+01 3.36e+01 7.99e+01 3.04e+01 8.17e+01
4.93e+01
 5.51e+01 6.02e+01 1.01e+02 9.26e+01 4.21e+01 6.72e+01 1.42e+02
5.81e+01
```

```
 4.30e+01 3.49e+01 7.10e+01 6.54e+01 3.51e+01 1.54e+02 6.12e+01
6.22e+01
 8.10e+01 7.85e+01 5.67e+01 6.27e+01 4.62e+01 6.59e-02 6.07e+01
3.64e+01
 6.36e+01 4.91e+01 1.74e+01 2.85e+01 4.12e+01 7.82e+01 5.15e+01
2.38e+01
 3.66e+01 3.88e+01 2.11e+01 3.00e+01 5.31e+01 3.30e+01 4.03e+01
4.79e+01
 1.08e+02 3.45e+01 1.74e+02 7.78e+01 8.12e+01 2.74e+01 2.68e+01
5.71e+01
 1.72e+01 3.84e+01 4.07e+01 5.33e+01 5.86e+01 2.91e+01 6.08e+01
2.78e+01
 5.73e+01 6.03e+01 5.53e+01 2.55e+01 4.45e+01 2.86e+01 5.11e+01
3.08e+01
 1.58e+01 2.54e+01 5.27e+01 1.76e+01 8.02e+01 3.44e+01 3.09e+01]

income: [  1610   9930  12900   5900  19100  18700   6700  41400
43200  16000
   22900  41100   2440  15300  16200   7880   1820   6420   5410   9720
   13300  14500  80600   1430    764   2520   2660  40700   5830    888
    1930  19400   9530  10900   1410    609   5190  13000   2690  20100
   33900  28300  44000  11100   9350   9860   7300  33700   1420  22700
    7350  39800  36900  15400   1660   6730  40400   3060  28700  11200
    6710   1190   1390   5840   1500  22300  38800   4410   8430  17400
   12700  45700  29600  36200   8000  35800   9470   2480   1730  75200
    2790   3980  18300  16300   2380    700  21100  91700  11400   1030
   10500   1870   3320  15900   3340   3910   7710  14000   6440    918
    3720   8460   1990  45500  32300    814   5150  62300  45300   4280
    7290   9960   5600  21800  27200 125000  17800  23100   1350   5400
   45400   2180  20400   1220  72100  25200   1780  12000  30400  32500
    8560   9920   3370  14200  42900  55500   2110   2090  13500   1850
    1210   4980  10400  18000   9940   1540   7820  57600  49400  17100
    4240   2950  16500   4490   4480   3280]

inflation: [ 9.44e+00  4.49e+00  1.61e+01  2.24e+01  1.44e+00
2.09e+01  7.77e+00
   1.16e+00  8.73e-01  1.38e+01 -3.93e-01  7.44e+00  7.14e+00  3.21e-01
   1.51e+01  1.88e+00  1.14e+00  8.85e-01  5.99e+00  8.78e+00  1.40e+00
   8.92e+00  8.41e+00  1.67e+01  1.11e+00  6.81e+00  1.23e+01  3.12e+00
   1.91e+00  2.87e+00  5.05e-01  2.01e+00  6.39e+00  8.96e+00  6.94e+00
   3.86e+00  3.87e+00  2.08e+01  2.07e+01  6.57e+00  5.39e+00  8.21e-01
  -1.43e+00  3.22e+00  5.44e+00  7.47e+00  1.01e+01  2.65e+00  2.49e+01
   1.16e+01  1.74e+00  4.23e+00  3.51e-01  1.05e+00  1.66e+01  4.30e+00
   8.55e+00  7.58e-01  6.73e-01  4.80e-01  5.14e+00  2.97e+00  5.73e+00
   5.45e+00  2.33e+00  5.47e+00  8.98e+00  1.53e+01  1.59e+01 -3.22e+00
   1.77e+00  3.19e-01  9.81e+00 -1.90e+00  8.43e+00  1.95e+01  2.09e+00
   1.52e+00  1.12e+01  1.00e+01  9.20e+00 -8.12e-01  2.38e-01  4.15e+00
   1.42e+01  2.38e+00  3.62e+00  2.04e+00  8.79e+00  1.21e+01  7.27e+00
   2.88e+00  4.37e+00  3.83e+00  1.89e+01  1.13e+00  3.80e+00  1.11e+01
```

```
  3.92e+01  1.60e+00  9.76e-01  7.64e+00  7.04e+00  3.56e+00  8.48e-01
  3.73e+00  2.55e+00  1.04e+02  5.95e+00  1.56e+01  1.09e+01  2.59e+00
  6.10e+00  5.71e+00  4.22e+00  1.66e+00  6.43e-01  6.98e+00  3.53e+00
  2.61e+00  1.72e+00  1.72e+01  1.85e+00  5.88e+00 -4.21e+00 -4.60e-02
  4.85e-01 -9.87e-01  6.35e+00  3.16e+00  1.60e-01  2.28e+01  4.44e+00
  1.96e+01  7.20e+00  9.91e-01  3.17e-01  1.25e+01  9.25e+00  4.08e+00
  2.65e+01  1.18e+00  3.68e+00  3.82e+00  7.01e+00  2.31e+00  1.06e+01
  1.34e+01  1.57e+00  1.22e+00  4.91e+00  1.65e+01  2.62e+00  4.59e+01
  2.36e+01  1.40e+01]

life_expec: [56.2 76.3 76.5 60.1 76.8 75.8 73.3 82.  80.5 69.1 73.8
76.  70.4 76.7
 80.  71.4 61.8 72.1 71.6 57.1 74.2 77.1 73.9 57.9 57.7 66.1 57.3 81.3
 72.5 47.5 56.5 79.1 74.6 76.4 65.9 57.5 60.4 80.4 56.3 79.9 77.5 79.5
 70.5 74.1 60.9 61.7 65.3 81.4 62.9 65.5 72.8 80.1 62.2 71.3 58.  55.6
 32.1 74.5 66.2 69.9 67.2 81.7 74.7 82.8 68.4 62.8 60.7 78.2 68.5 63.8
 73.1 79.8 46.5 60.8 76.1 73.2 74.  53.1 77.9 59.5 80.3 68.2 73.4 65.4
 69.7 73.5 54.5 66.8 58.6 68.3 80.7 80.9 58.8 60.5 81.  77.8 69.  73.7
 69.2 64.6 71.5 75.1 64.  55.  82.7 75.5 54.3 81.9 74.4 66.3 70.3 81.5
 82.2 69.6 59.3 76.6 71.1 58.7 76.9 67.9 56.8 78.7 68.8 63.  75.4 67.5
 52. ]

total_fer: [5.82 1.65 2.89 6.16 2.13 2.37 1.69 1.93 1.44 1.92 1.86
2.16 2.33 1.78
 1.49 2.71 5.36 2.38 3.2  1.31 2.88 1.8  1.84 1.57 5.87 6.26 5.11 1.63
 2.67 5.21 6.59 1.88 1.59 2.01 4.75 6.54 4.95 5.27 1.55 1.42 1.51 1.87
 2.6  2.66 3.19 2.27 4.61 1.72 2.03 4.08 5.71 1.39 4.27 1.48 2.24 3.38
 5.34 5.05 2.65 3.33 1.25 2.2  2.48 1.76 4.56 2.05 3.03 1.46 2.17 3.66
 4.37 3.84 2.21 3.1  3.15 1.36 1.61 3.3  5.02 2.41 1.5  1.47 4.6  5.31
 2.15 2.23 6.55 4.98 3.46 1.27 2.64 1.77 2.58 5.56 3.6  2.61 1.79 7.49
 5.84 1.95 2.9  3.85 2.62 2.73 2.54 3.16 1.41 2.07 4.51 4.34 2.96 5.06
 1.4  5.2  1.15 1.43 4.24 2.59 1.23 1.37 4.88 2.52 1.98 1.52 3.51 5.43
 6.23 4.87 3.91 2.14 2.83 6.15 2.08 2.34 3.5  2.47 4.67 5.4 ]

gdpp: [   553   4090   4460   3530  12200  10300   3220  51900  46900
5840
  28000  20700    758  16000   6030  44400   4340   2180   1980   4610
   6350  11200  35300   6840    575    231    786   1310  47400   3310
    446    897  12900   4560   6250    769    334   2740   8200   1220
  13500  30800  19800  58000   5450   4660   2600   2990  17100    482
  14600   3650  46200  40600   8750    562   2960  41800  26900   7370
   2830    648    547   3040    662  13100  41900   1350   3110   6530
   4500  48700  30600  35800   4680  44500   3680   9070    967   1490
  38500    880   1140  11300   8860   1170    327  12100  12000 105000
   4540    413    459   7100    708  21100   1200   8000   2860   1630
   2650   6680    419    988   5190    592  50300  33700    348   2330
  87800  19300   1040   8080   3230   5020   2130  12600  22500  70300
   8230  10700    563   3450   1000   5410  10800    399  46600  16600
  23400   1290   7280  22100  30700   2810   6230   1480   8300  52100
  74600    738    702   5080   3600    488   3550   4140   4440    595
```

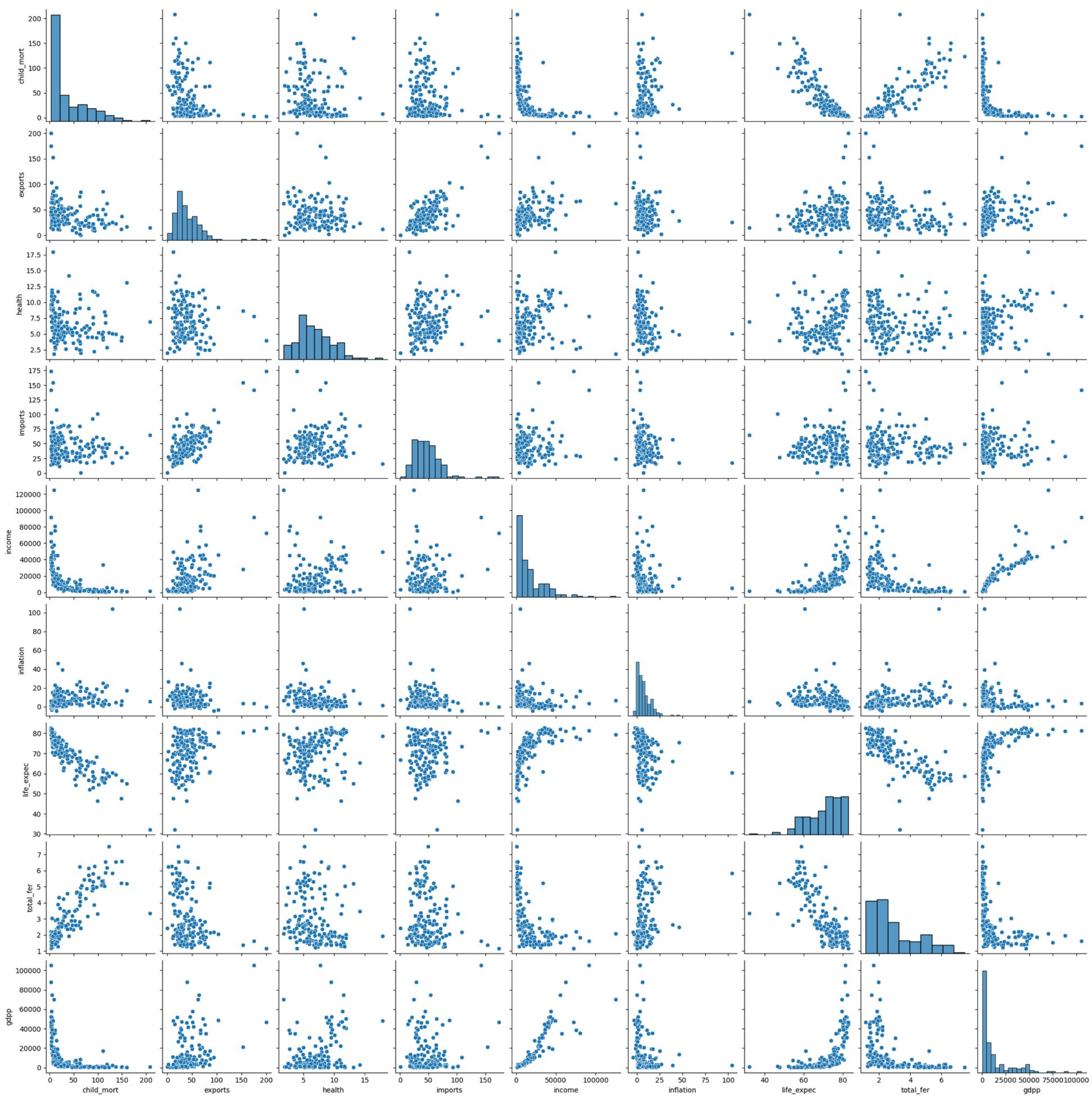
```
   2970  35000  38900  48400  11900   1380   1460]
```

```python
#@title EDA

df.describe()
```

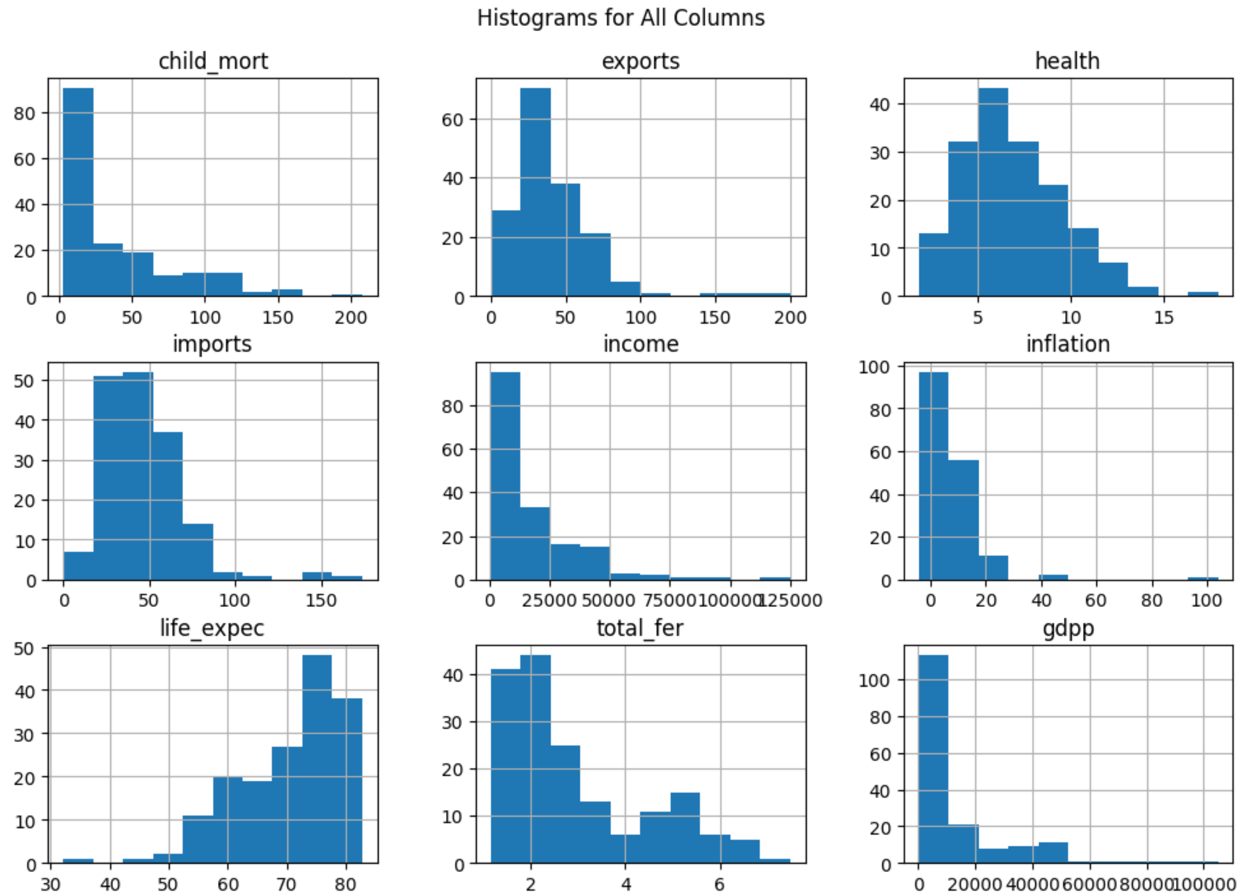
```
       child_mort     exports      health     imports        
income  \
count  167.000000  167.000000  167.000000  167.000000     167.000000

mean    38.270060   41.108976    6.815689   46.890215   17144.688623

std     40.328931   27.412010    2.746837   24.209589   19278.067698

min      2.600000    0.109000    1.810000    0.065900     609.000000

25%      8.250000   23.800000    4.920000   30.200000    3355.000000

50%     19.300000   35.000000    6.320000   43.300000    9960.000000

75%     62.100000   51.350000    8.600000   58.750000   22800.000000

max    208.000000  200.000000   17.900000  174.000000  125000.000000


        inflation  life_expec   total_fer           gdpp
count  167.000000  167.000000  167.000000     167.000000
mean     7.781832   70.555689    2.947964   12964.155689
std     10.570704    8.893172    1.513848   18328.704809
min     -4.210000   32.100000    1.150000     231.000000
25%      1.810000   65.300000    1.795000    1330.000000
50%      5.390000   73.100000    2.410000    4660.000000
75%     10.750000   76.800000    3.880000   14050.000000
max    104.000000   82.800000    7.490000  105000.000000
```

```python
#@title Pairplot

import seaborn as sns

sns.pairplot(df)
```

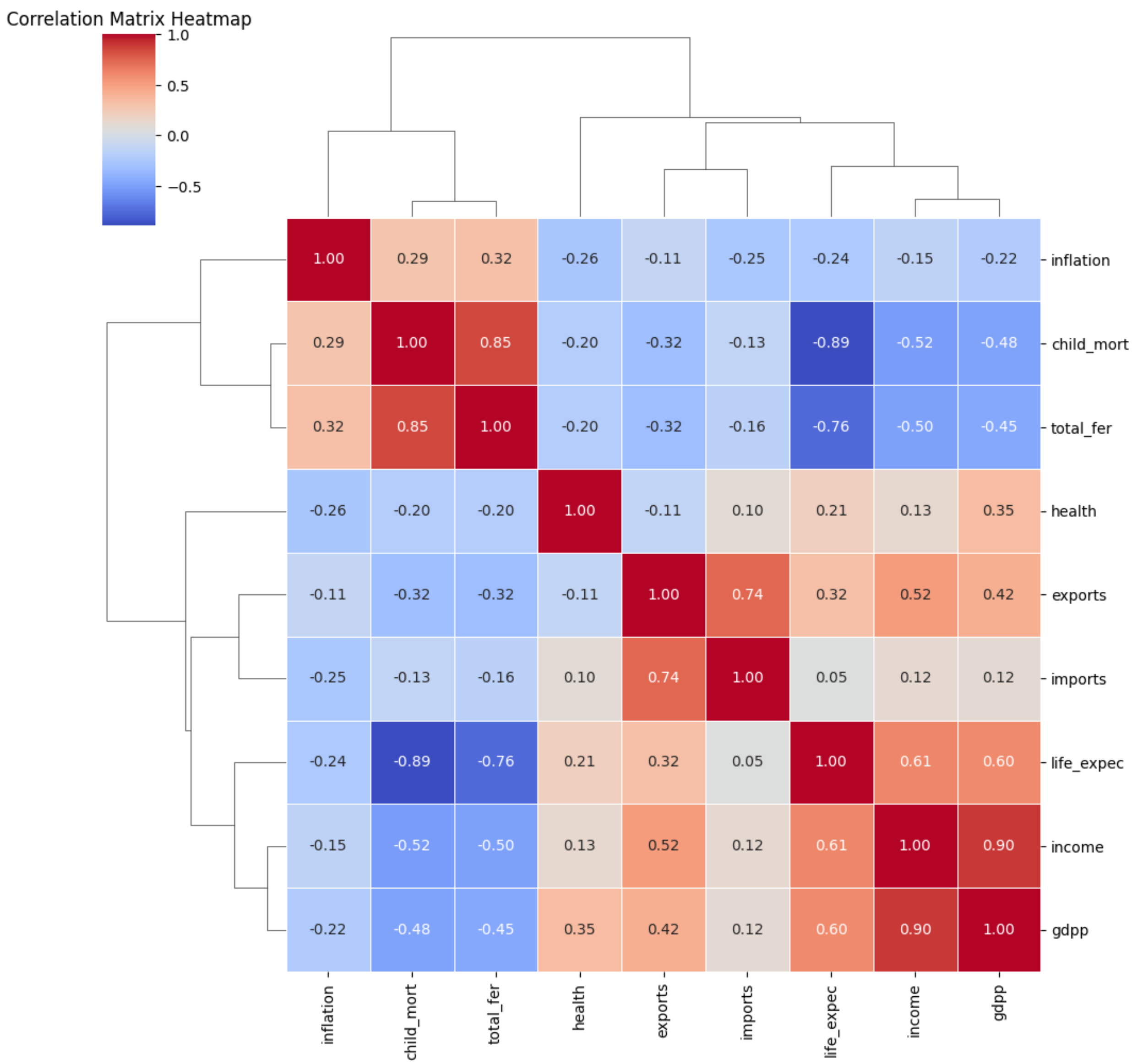
```
<seaborn.axisgrid.PairGrid at 0x7d20ca763580>
```



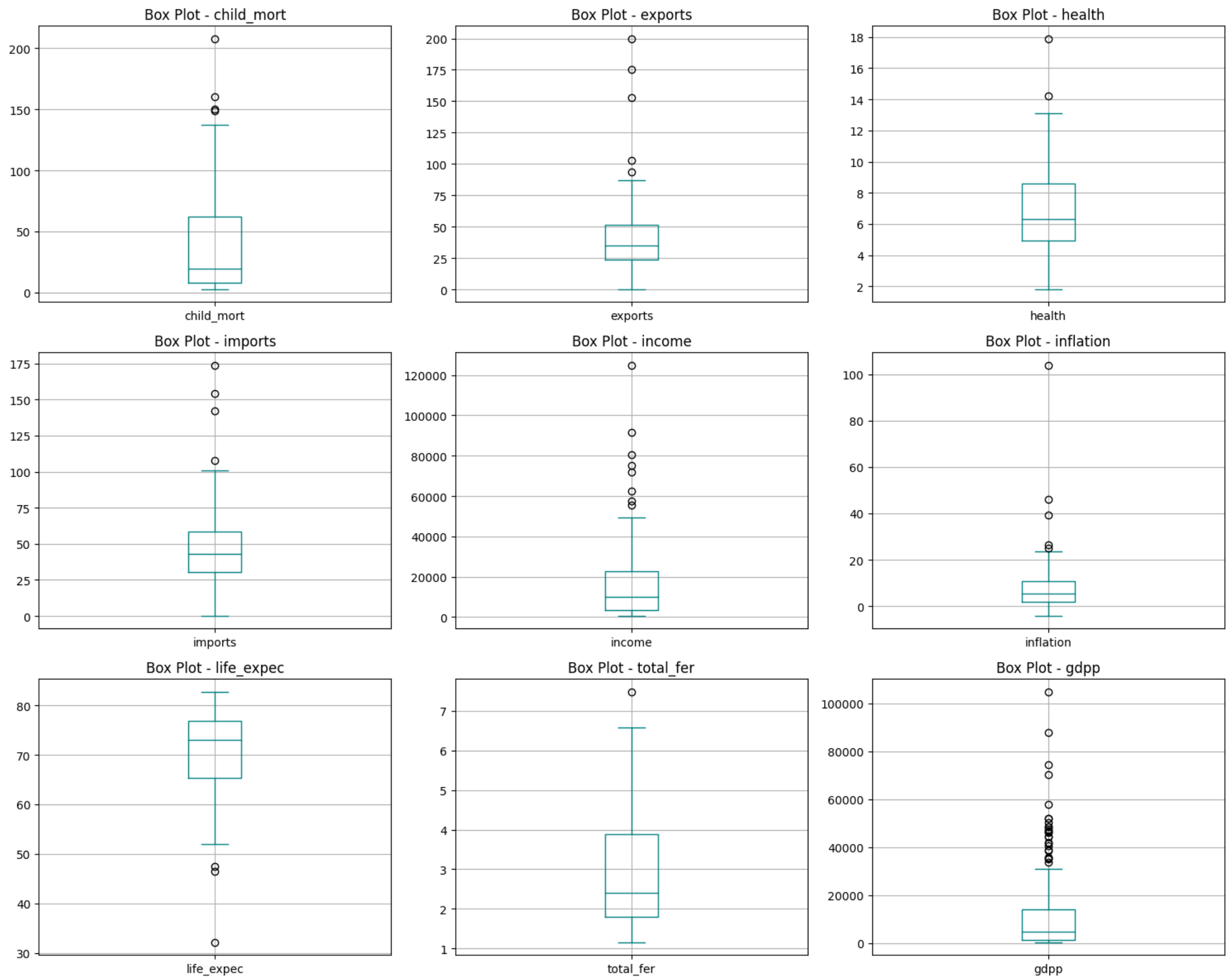

```
#@title Histogram

df.hist(bins=10, figsize=(12, 8))
plt.suptitle('Histograms for All Columns', x=0.5, y=0.95, ha='center',
fontsize='large')
plt.show()
```




```
#@title Heatmap

correlation_matrix = df.corr()

plt.figure(figsize=(10, 8))
sns.clustermap(correlation_matrix, dendrogram_ratio = (0.2, 0.2),
annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()

<Figure size 1000x800 with 0 Axes>
```




```
#@title Boxplot
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 12))

axes = axes.flatten()

# Plot box plots for each feature
for i, feature in enumerate(df.columns):
    df.boxplot(column=feature, ax=axes[i], color = 'teal')
    axes[i].set_title(f'Box Plot - {feature}')

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




```
#@title Normalising the data

# from sklearn.preprocessing import MinMaxScaler

# scaler = MinMaxScaler()
# scaled_data = MinMaxScaler().fit_transform(df)
# scaled_data_df = pd.DataFrame(scaled_data, columns=df.columns)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

df_scaled
```

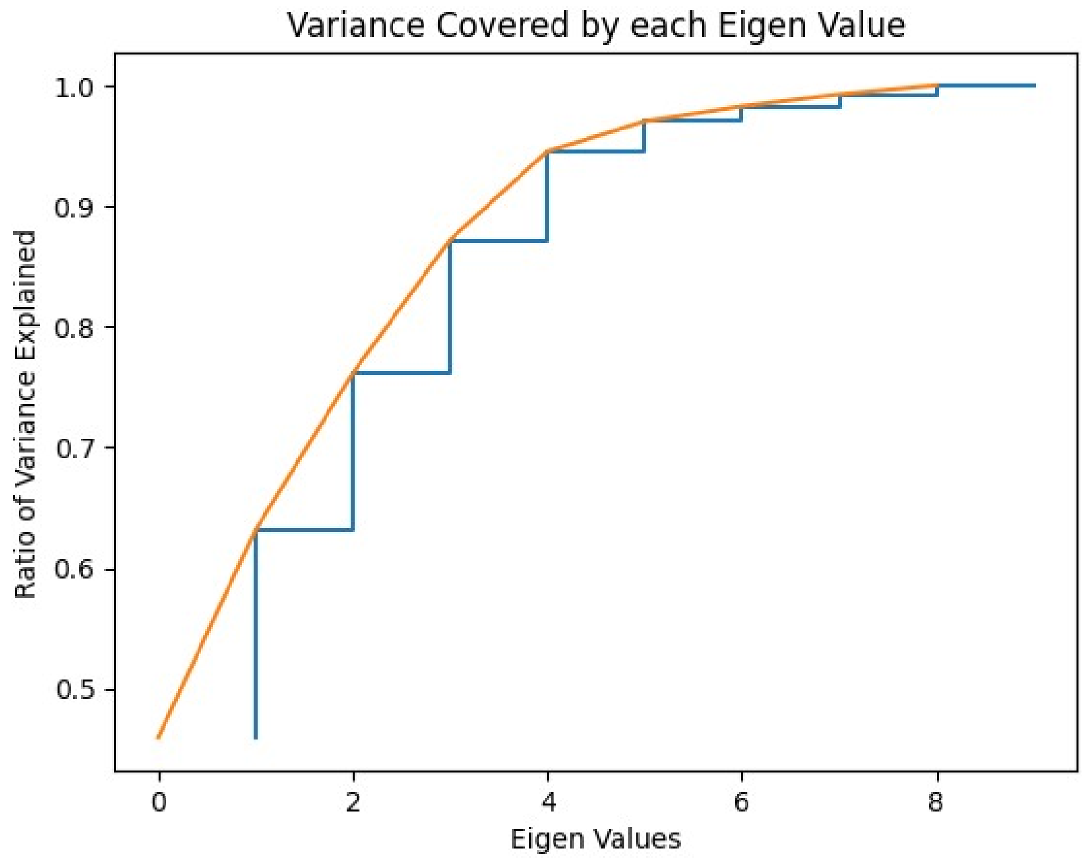
```
array([[ 1.29153238, -1.13827979,  0.27908825, ..., -1.61909203,
         1.90288227, -0.67917961],
       [-0.5389489 , -0.47965843, -0.09701618, ...,  0.64786643,
        -0.85997281, -0.48562324],
       [-0.27283273, -0.09912164, -0.96607302, ...,  0.67042323,
```

```
         -0.0384044 , -0.46537561],
        ...,
       [-0.37231541,  1.13030491,  0.0088773 , ...,  0.28695762,
         -0.66120626, -0.63775406],
       [ 0.44841668, -0.40647827, -0.59727159, ..., -0.34463279,
          1.14094382, -0.63775406],
       [ 1.11495062, -0.15034774, -0.33801514, ..., -2.09278484,
          1.6246091 , -0.62954556]])
```

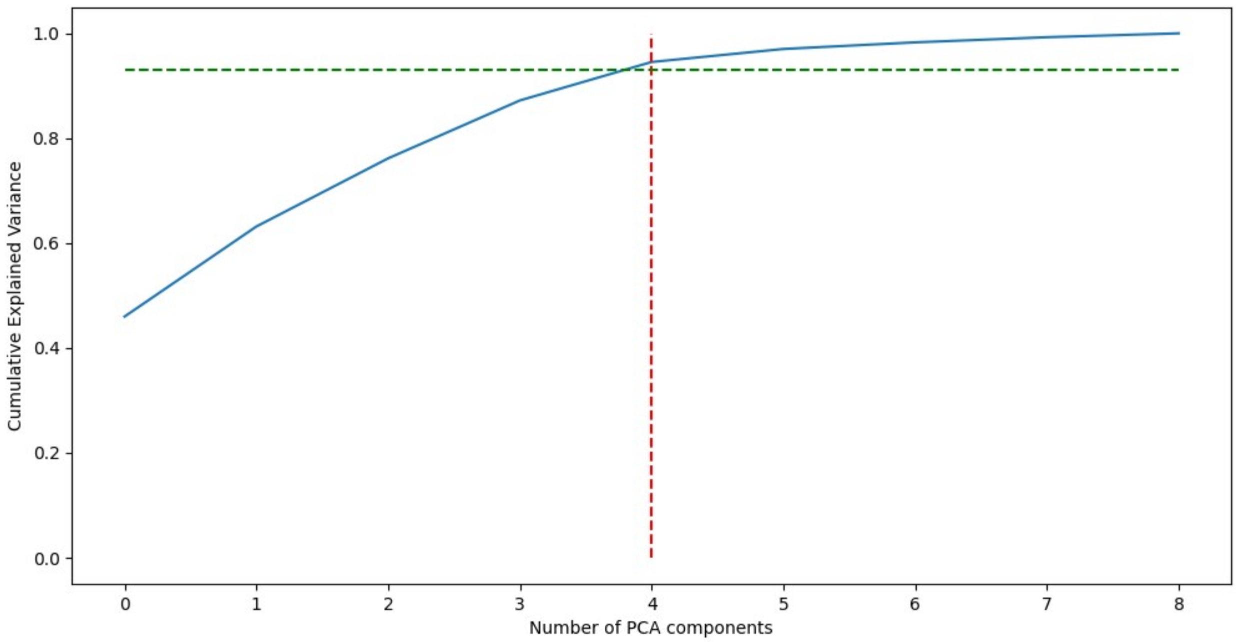
```
#@title PCA

from sklearn.decomposition import PCA

# Applying PCA
pca = PCA()
pca.fit_transform(df_scaled)
```

```
array([[-2.91302459,  0.09562058, -0.7181185 , ...,  0.38300026,
          0.41507602, -0.01414844],
       [ 0.42991133, -0.58815567, -0.3334855 , ...,  0.24891887,
         -0.22104247,  0.17331578],
       [-0.28522508, -0.45517441,  1.22150481, ..., -0.08721359,
         -0.18416209,  0.08403718],
        ...,
       [ 0.49852439,  1.39074432, -0.23852611, ..., -0.14362677,
         -0.21759009, -0.03652231],
       [-1.88745106, -0.10945301,  1.10975159, ...,  0.06025631,
          0.08949452, -0.09604924],
       [-2.86406392,  0.48599799,  0.22316658, ..., -0.44218462,
          0.66433809, -0.44148176]])
```

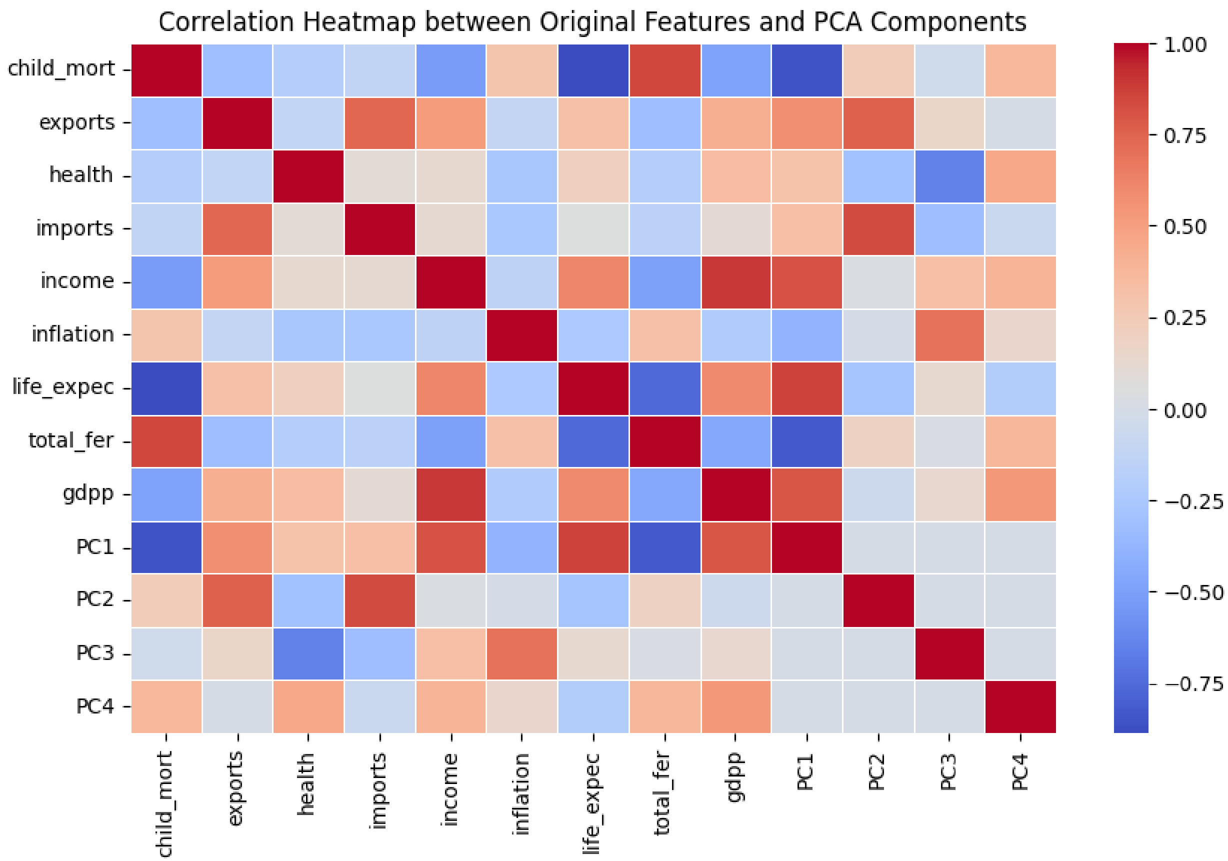
```
#@title Variance of each principal component

plt.step(list(range(1,10)), np.cumsum(pca.explained_variance_ratio_))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Eigen Values')
plt.ylabel('Ratio of Variance Explained')
plt.title('Variance Covered by each Eigen Value')
plt.show()
```




```
fig = plt.figure(figsize = (12,6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.vlines(x=4, ymax=1, ymin=0, colors="r", linestyles="--")
plt.hlines(y=0.93, xmax=8, xmin=0, colors="g", linestyles="--")
plt.xlabel('Number of PCA components')
plt.ylabel('Cumulative Explained Variance')

Text(0, 0.5, 'Cumulative Explained Variance')
```

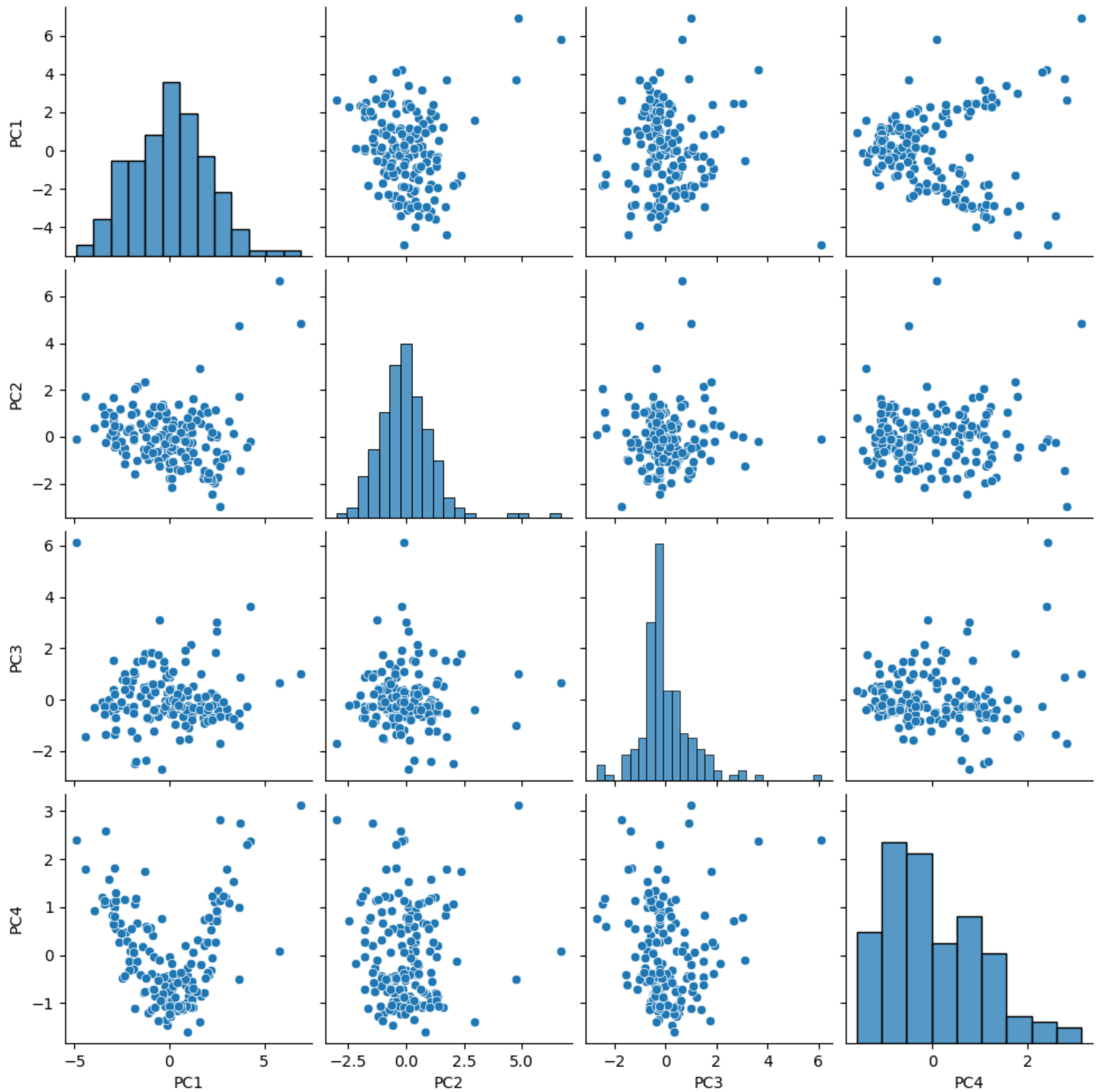



```
#@title Preferred PCA (n=4)

num_components = 4

pca_df = PCA(n_components=num_components)
pca_result = pca_df.fit_transform(df_scaled)

#@title Final Heatmap
columns_pca = [f'PC{i+1}' for i in range(num_components)]
df_pca = pd.DataFrame(data=pca_result, columns=columns_pca)

# df_pca

# Concatenating the original features and principal components
df_combined = pd.concat([df, df_pca], axis=1)

# df_combined

# Calculating the correlation matrix
correlation_matrix = df_combined.corr()

# Ploting the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False,
fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap between Original Features and PCA
Components')
plt.show()
```



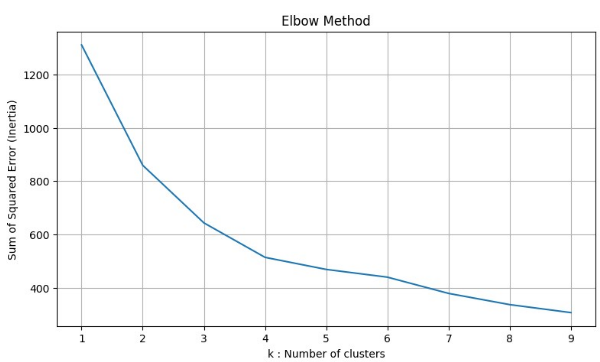

```
df_combined

     child_mort  exports  health  imports  income  inflation
life_expec  \
0          90.2     10.0    7.58     44.9    1610       9.44
56.2
1          16.6     28.0    6.55     48.6    9930       4.49
76.3
2          27.3     38.4    4.17     31.4   12900      16.10
76.5
3         119.0     62.3    2.85     42.9    5900      22.40
60.1
4          10.3     45.5    6.03     58.9   19100       1.44
76.8
..          ...      ...     ...      ...     ...        ...         .
..
162        29.2     46.6    5.25     52.7    2950       2.62
63.0
163        17.1     28.5    4.91     17.6   16500      45.90
75.4
164        23.3     72.0    6.84     80.2    4490      12.10
73.1
165        56.3     30.0    5.18     34.4    4480      23.60
```

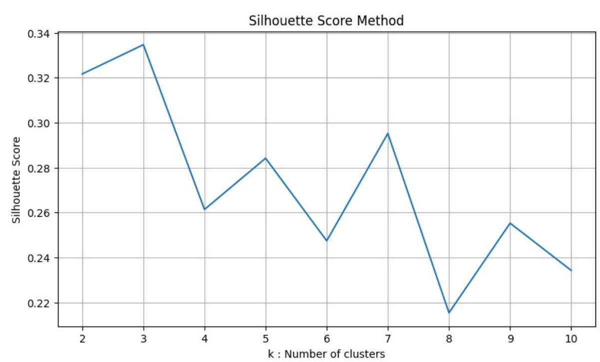
```
67.5
166        83.1      37.0     5.89      30.9    3280       14.00
52.0

     total_fer   gdpp        PC1        PC2        PC3        PC4
0         5.82    553 -2.913025  0.095621 -0.718118  1.005255
1         1.65   4090  0.429911 -0.588156 -0.333486 -1.161059
2         2.89   4460 -0.285225 -0.455174  1.221505 -0.868115
3         6.16   3530 -2.932423  1.695555  1.525044  0.839625
4         2.13  12200  1.033576  0.136659 -0.225721 -0.847063
..         ...    ...        ...        ...        ...        ...
162       3.50   2970 -0.820631  0.639570 -0.389923 -0.706595
163       2.47  13500 -0.551036 -1.233886  3.101350 -0.115311
164       1.95   1310  0.498524  1.390744 -0.238526 -1.074098
165       4.67   1310 -1.887451 -0.109453  1.109752  0.056257
166       5.40   1460 -2.864064  0.485998  0.223167  0.816364

[167 rows x 13 columns]

#@title Scatter-Plots of PCA

sns.pairplot(df_pca)
plt.show()
```




```python
#@title Elbow and Silhouette Method

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

inertia = [] #sum of square error
sil = []
kmax = 10
fig = plt.subplots(nrows=1, ncols=2, figsize=(20, 5))

# Elbow Method
plt.subplot(1, 2, 1)
```

```
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000,
n_init='auto').fit(df_pca)
    inertia.append(kmeans.inertia_)
sns.lineplot(x=list(range(1, 10)), y=inertia)
plt.title('Elbow Method')
plt.xlabel("k : Number of clusters")
plt.ylabel("Sum of Squared Error (Inertia)")
plt.grid(True)

# Silhouette Score Method
plt.subplot(1, 2, 2)
for k in range(2, kmax + 1):
    kmeans = KMeans(n_clusters=k, n_init='auto').fit(df_pca)
    labels = kmeans.labels_
    sil.append(silhouette_score(df_pca, labels, metric='euclidean'))
sns.lineplot(x=range(2, kmax + 1), y=sil)
plt.title('Silhouette Score Method')
plt.xlabel("k : Number of clusters")
plt.ylabel("Silhouette Score")
plt.grid(True)

plt.show()
```






```
#@title Applying K-Means Clustering

kmeans = KMeans(n_clusters=4, init='random', n_init=10, max_iter=100,
random_state=1)
labels = kmeans.fit_predict(df_scaled)

# labels.shape
print(labels)
```

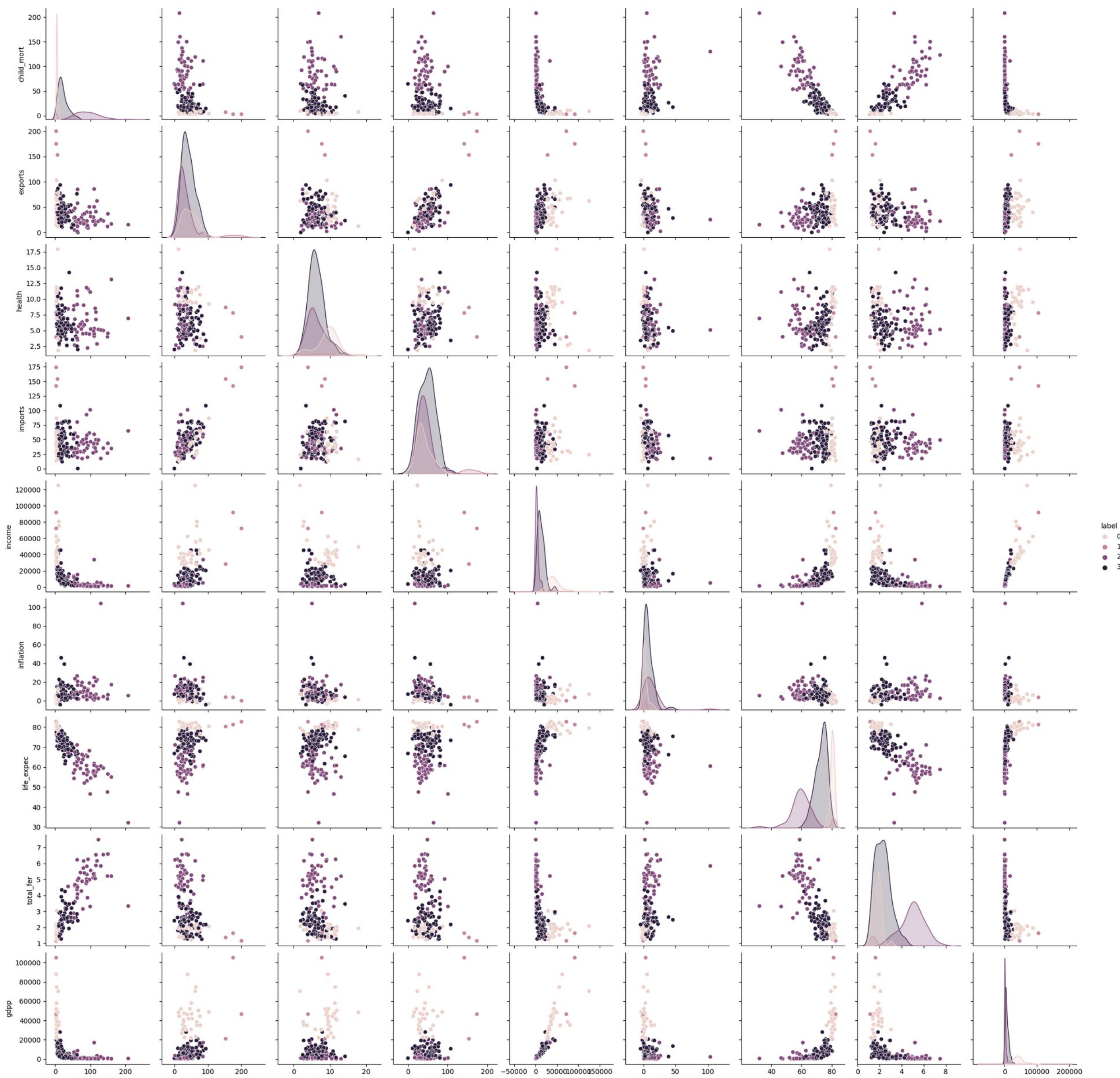
```
[2 3 3 2 3 3 3 0 0 3 3 3 3 3 3 0 3 2 3 3 3 2 3 0 3 2 2 3 2 0 3 2 2 3 3
3 2
 2 2 3 2 3 0 3 0 3 3 3 3 2 2 3 3 0 0 2 2 3 0 2 0 3 3 2 2 3 2 3 0 3 3 3
2 0
 0 0 3 0 3 3 2 2 0 3 2 3 3 2 2 3 3 1 3 2 2 3 3 2 1 2 3 3 3 3 3 3 2 3 2
```

```
3 0
 0 2 2 0 3 2 3 3 3 3 3 0 0 3 3 2 3 3 2 3 3 2 1 3 0 3 2 0 0 3 3 2 3 0 0
3 2
 3 2 2 3 3 3 3 2 3 0 0 0 3 3 3 3 3 2 2]
```

```python
#@title Concatinating the Labels

# Assign the label
df_labelled=pd.concat([original_df,pd.Series(kmeans.labels_,name='label')],axis=1)
df_labelled
```

```
                  country  child_mort  exports  health  imports  income  \
0             Afghanistan        90.2     10.0    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       119.0     62.3    2.85     42.9    5900   

4     Antigua and Barbuda        10.3     45.5    6.03     58.9   19100   

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

162               Vanuatu        29.2     46.6    5.25     52.7    2950   

163             Venezuela        17.1     28.5    4.91     17.6   16500   

164               Vietnam        23.3     72.0    6.84     80.2    4490   

165                 Yemen        56.3     30.0    5.18     34.4    4480   

166                Zambia        83.1     37.0    5.89     30.9    3280   


     inflation  life_expec  total_fer   gdpp  label
0         9.44        56.2       5.82    553      2
1         4.49        76.3       1.65   4090      3
2        16.10        76.5       2.89   4460      3
3        22.40        60.1       6.16   3530      2
4         1.44        76.8       2.13  12200      3
..         ...         ...        ...    ...    ...
162       2.62        63.0       3.50   2970      3
163      45.90        75.4       2.47  13500      3
164      12.10        73.1       1.95   1310      3
165      23.60        67.5       4.67   1310      2
166      14.00        52.0       5.40   1460      2

[167 rows x 11 columns]
```

```
df_labelled['label'].value_counts()

3    87
2    47
0    30
1     3
Name: label, dtype: int64

#@title Pairplots - Hue('label)

sns.pairplot(df_labelled, hue ='label')

<seaborn.axisgrid.PairGrid at 0x7d20beddc700>
```



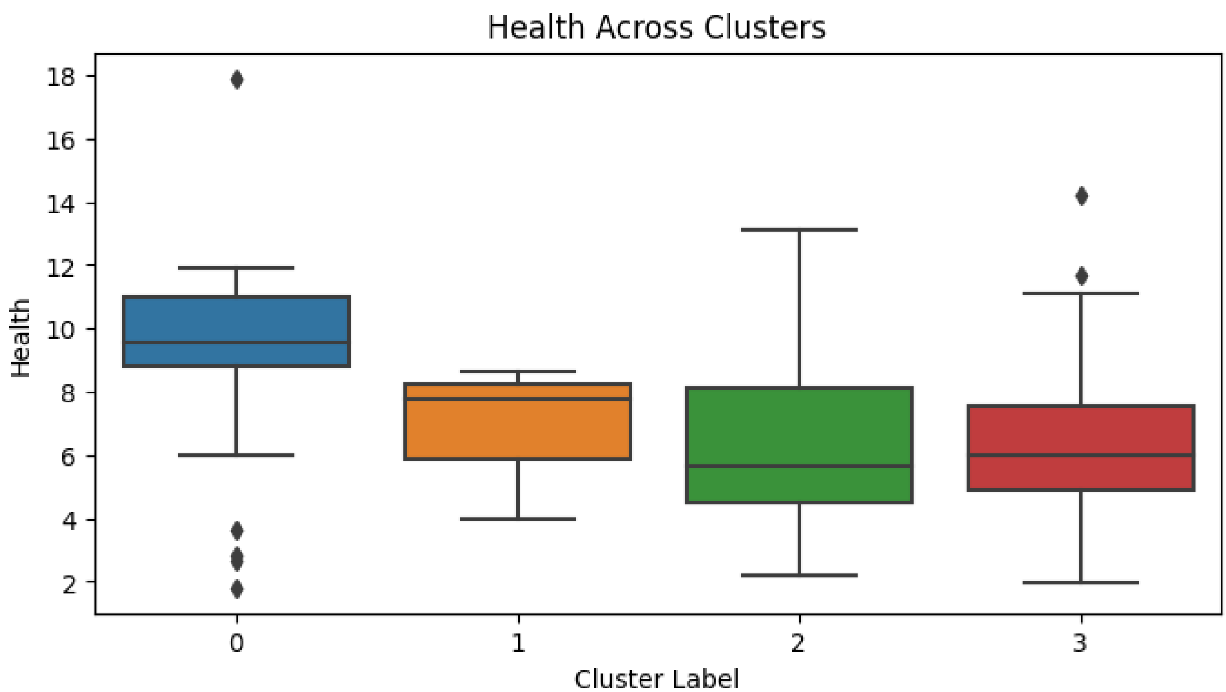
```python
# Print countries with label 0
print("Countries with label 0 (Help Needed):")
print(original_df[df_labelled['label'] == 0]['country'])

# Print countries with label 1
print("\nCountries with label 1 (Might Need Help):")
print(original_df[df_labelled['label'] == 1]['country'])

# Print countries with label 2
print("\nCountries with label 2 (No Help Needed):")
print(original_df[df_labelled['label'] == 2]['country'])

# Print countries with label 3
print("\nCountries with label 3 (Help Needed - More than other
countries):")
print(original_df[df_labelled['label'] == 3]['country'])
```

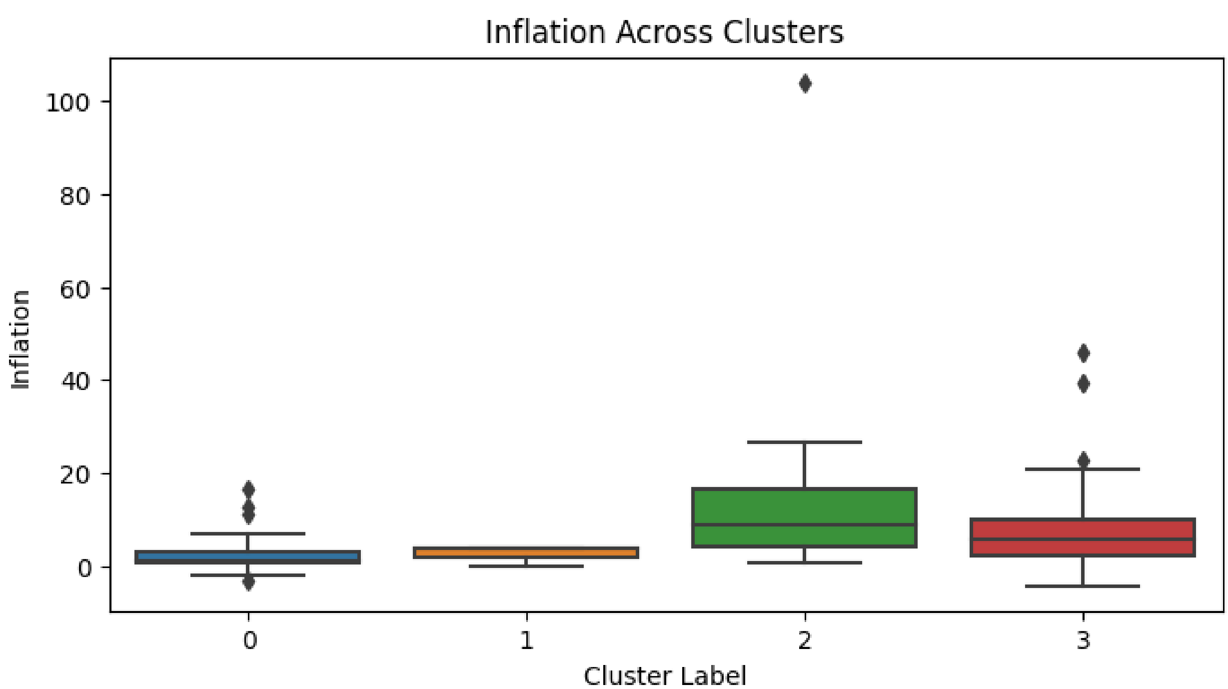
```
Countries with label 0 (Help Needed):
7                 Australia
8                   Austria
15                  Belgium
23                   Brunei
29                   Canada
42                   Cyprus
44                  Denmark
53                  Finland
54                   France
58                  Germany
60                   Greece
68                  Iceland
73                  Ireland
74                   Israel
75                    Italy
77                    Japan
82                   Kuwait
110             Netherlands
111             New Zealand
114                  Norway
122                Portugal
123                   Qatar
135                Slovenia
138             South Korea
139                   Spain
144                  Sweden
145             Switzerland
157    United Arab Emirates
158          United Kingdom
159           United States
Name: country, dtype: object
```

```
Countries with label 1 (Might Need Help):
91     Luxembourg
98          Malta
133     Singapore
Name: country, dtype: object

Countries with label 2 (No Help Needed):
0                   Afghanistan
3                        Angola
17                        Benin
21                     Botswana
25                 Burkina Faso
26                      Burundi
28                     Cameroon
31     Central African Republic
32                         Chad
36                      Comoros
37              Congo, Dem. Rep.
38                   Congo, Rep.
40                Cote d'Ivoire
49            Equatorial Guinea
50                      Eritrea
55                        Gabon
56                       Gambia
59                        Ghana
63                       Guinea
64                Guinea-Bissau
66                        Haiti
72                         Iraq
80                        Kenya
81                     Kiribati
84                          Lao
87                      Lesotho
88                      Liberia
93                   Madagascar
94                       Malawi
97                         Mali
99                   Mauritania
106                  Mozambique
108                     Namibia
112                       Niger
113                     Nigeria
116                    Pakistan
126                      Rwanda
129                     Senegal
132                Sierra Leone
137                South Africa
142                       Sudan
147                    Tanzania
```

```
149                  Timor-Leste
150                         Togo
155                       Uganda
165                        Yemen
166                       Zambia
Name: country, dtype: object

Countries with label 3 (Help Needed - More than other countries):
1                  Albania
2                  Algeria
4      Antigua and Barbuda
5                Argentina
6                  Armenia
              ...
160                Uruguay
161             Uzbekistan
162                Vanuatu
163              Venezuela
164                Vietnam
Name: country, Length: 87, dtype: object
```

```
#@title Boxplot of specific features

# Box plot for a specific features
plt.figure(figsize=(8, 4))
sns.boxplot(x='label', y='health', data=df_labelled)
plt.title('Health Across Clusters')
plt.xlabel('Cluster Label')
plt.ylabel('Health')
plt.show()

plt.figure(figsize=(8, 4))
sns.boxplot(x='label', y='inflation', data=df_labelled)
plt.title('Inflation Across Clusters')
plt.xlabel('Cluster Label')
plt.ylabel('Inflation')
plt.show()
```





Section B

```
import numpy as np

# formula of convolution used
```

```
# C(i,j)=∑m∑n I(i+m,j+n)·K(m,n)+b

def convolution_forward(input_data, kernel, stride, padding):

    # Parameters:
    # - input_data: Input data (numpy array).
    # - kernel: Convolutional kernel (numpy array).
    # - stride: Stride for the convolution operation.
    # - padding: Padding for the convolution operation.

    # Returns:
    # - output: Result of the convolution operation.
    # - cache: Tuple containing information needed for the backward
pass.


    # Adding padding to the input data
    padded_data = np.pad(input_data, ((padding, padding), (padding,
padding), (0, 0)), mode='constant')

    # Calculating the output dimensions
    output_height = (padded_data.shape[0] - kernel.shape[0]) // stride
+ 1
    output_width = (padded_data.shape[1] - kernel.shape[1]) // stride
+ 1

    # Initializing the output
    output = np.zeros((output_height, output_width, kernel.shape[2]))

    # Performing the convolution
    for i in range(0, output.shape[0], stride):
        for j in range(0, output.shape[1], stride):
            window = padded_data[i:i+kernel.shape[0],
j:j+kernel.shape[1]]
            output[i//stride, j//stride] = np.sum(window * kernel,
axis=(0, 1))

    cache = (input_data, kernel, stride, padding, padded_data)

    return output, cache

def convolution_backward(doutput, cache):

    # Parameters:
    # - doutput: Gradient of the loss with respect to the output of
the convolution.
    # - cache: Tuple containing information needed for the backward
pass.

    # Returns:
```

```
    # - dinput: Gradient of the loss with respect to the input of the
convolution.
    # - dkernel: Gradient of the loss with respect to the
convolutional kernel.

    input_data, kernel, stride, padding, padded_data = cache

    # Initializing gradients
    dinput = np.zeros_like(input_data)
    dkernel = np.zeros_like(kernel)

    # Iterating over the input data and update gradients
    for i in range(0, doutput.shape[0], stride):
        for j in range(0, doutput.shape[1], stride):
            window = padded_data[i:i+kernel.shape[0],
j:j+kernel.shape[1]]
            dinput[i:i+kernel.shape[0], j:j+kernel.shape[1]] +=
np.sum(doutput[i//stride, j//stride, None, None, None] * kernel,
axis=2)
            dkernel += window * doutput[i//stride, j//stride, None,
None, :]

    return dinput, dkernel

def max_pooling_forward(input_data, pool_size, stride):

    # Parameters:
    # - input_data: Input data (numpy array).
    # - pool_size: Size of the pooling window.
    # - stride: Stride for the pooling operation.

    # Returns:
    # - output: Result of the max pooling operation.
    # - cache: Tuple containing information needed for the backward
pass.


    # Calculating the output dimensions
    output_height = (input_data.shape[0] - pool_size[0]) // stride[0]
+ 1
    output_width = (input_data.shape[1] - pool_size[1]) // stride[1] +
1

    # Initializing the output
    output = np.zeros((output_height, output_width,
input_data.shape[2]))

    # Performing the max pooling
    for i in range(0, output.shape[0], stride[0]):
        for j in range(0, output.shape[1], stride[1]):
```

```
            window = input_data[i:i+pool_size[0], j:j+pool_size[1]]
            output[i//stride[0], j//stride[1]] = np.max(window,
axis=(0, 1))

    cache = (input_data, pool_size, stride)

    return output, cache

def max_pooling_backward(doutput, cache):

    # Parameters:
    # - doutput: Gradient of the loss with respect to the output of
the max pooling.
    # - cache: Tuple containing information needed for the backward
pass.

    # Returns:
    # - dinput: Gradient of the loss with respect to the input of the
max pooling.


    input_data, pool_size, stride = cache

    # Initializing the gradient
    dinput = np.zeros_like(input_data)

    # Iterating over the input data and updating the gradient
    for i in range(0, doutput.shape[0], stride[0]):
        for j in range(0, doutput.shape[1], stride[1]):
            window = input_data[i:i+pool_size[0], j:j+pool_size[1]]
            max_values = np.max(window, axis=(0, 1))
            mask = (window == max_values)
            dinput[i:i+pool_size[0], j:j+pool_size[1]] += mask *
doutput[i//stride[0], j//stride[1], None, None, :]

    return dinput


# Testing the convolution function
input_data = np.random.rand(5, 5, 3)
kernel = np.random.rand(3, 3, 3)
stride = 3
padding = 3

conv_output, conv_cache = convolution_forward(input_data, kernel,
stride, padding)
doutput = np.random.rand(conv_output.shape[0], conv_output.shape[1],
conv_output.shape[2])

dinput, dkernel = convolution_backward(doutput, conv_cache)
```

```python
print("Convolution Forward Output:")
print(conv_output)
print("\nConvolution Backward dInput:")
print(dinput)
print("\nConvolution Backward dKernel:")
print(dkernel)
```

```
Convolution Forward Output:
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]

Convolution Backward dInput:
[[[1.13895999 0.53084404 0.58382703]
  [0.78980571 0.16703837 1.15712571]
  [0.5905336  0.65521735 0.63545153]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[1.13895999 0.53084404 0.58382703]
  [0.78980571 0.16703837 1.15712571]
  [0.5905336  0.65521735 0.63545153]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[1.13895999 0.53084404 0.58382703]
  [0.78980571 0.16703837 1.15712571]
  [0.5905336  0.65521735 0.63545153]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]]
```

```
Convolution Backward dKernel:
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]
```

```
# Testing the max pooling function
input_data_pooling = np.random.rand(6, 6, 3)
pool_size = (2, 2)
stride_pooling = (2, 2)

pool_output, pool_cache = max_pooling_forward(input_data_pooling,
pool_size, stride_pooling)
doutput_pooling = np.random.rand(pool_output.shape[0],
pool_output.shape[1], pool_output.shape[2])

dinput_pooling = max_pooling_backward(doutput_pooling, pool_cache)

print("\nMax Pooling Forward Output:")
print(pool_output)
print("\nMax Pooling Backward dInput:")
print(dinput_pooling)
```

```
Max Pooling Forward Output:
[[[0.70647021 0.98811828 0.74839217]
  [0.74007592 0.73242949 0.8870404 ]
  [0.         0.         0.        ]]

 [[0.37380854 0.84405131 0.96607071]
  [0.93866523 0.76954821 0.95618808]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]]

Max Pooling Backward dInput:
[[[0.         0.         0.        ]
  [0.         0.54365997 0.94862622]
  [0.         0.         0.        ]
  [0.70439626 0.         0.83915886]
  [0.         0.         0.        ]
```

```
  [0.         0.         0.        ]]

 [[0.74769064 0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.61889394 0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.05255551]
  [0.         0.26862296 0.05531794]
  [0.44751067 0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.53887602 0.99571689 0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]

 [[0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]
  [0.         0.         0.        ]]]
```