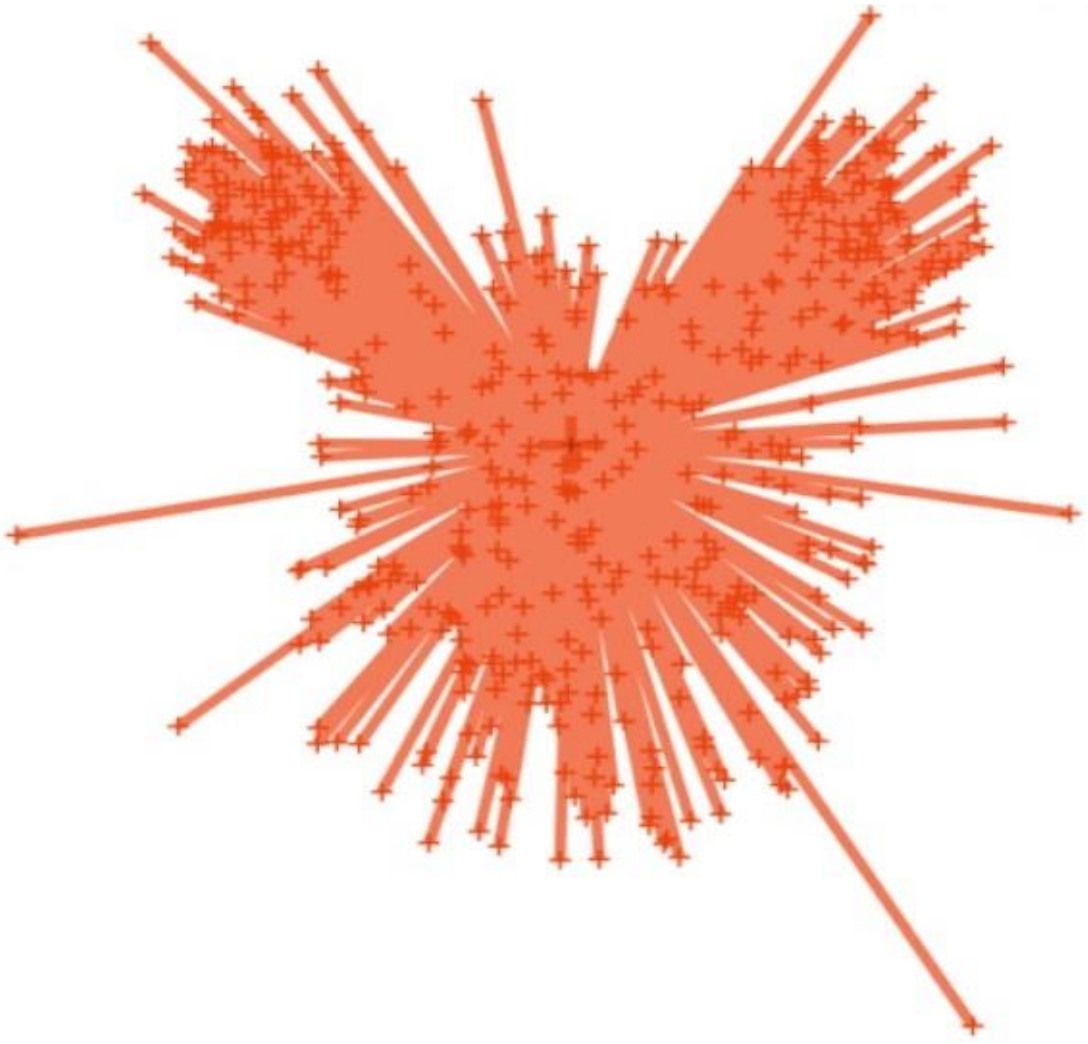
Unsupervised Learning: Clustering countries based on socio-economic factors using HELP International's Country Dataset

Author: Syed Bokhari

Date of Submission: 18th December 2022



Introduction:

The dataset used in this project comes from an NGO, HELP International. The dataset contains socio-economic data on countries of the world. The dataset can be found at the link: https://www.kaggle.com/datasets/rohan0301/unsupervised-learning-on-country-data

The dataset has data on 167 countries. It captures 9 socio-economic indicators for each of these countries. All indicators are in numeric format. A brief summary of the dataset's columns is provided below:

country(obj): The name of the country

child_mort(float): Child mortality for the country. This measure represents the number of deaths in children under 5 years of age for every 1000 live births.

exports(float): Total exports per capita. Given as %age of GDP per capita.

health(float): Total health spending per capita. Given as %age of GDP per capita.

imports(float): Total imports per capita. Given as %age of GDP per capita.

income(int): Net income per person.

inflation(float): This measures the annual growth rate of the total GDP.

life_expec(float): The average number of years each newborn child is expected to live based on current mortality patterns.

total_fer(float): The number of children that will be born to each woman based on current age-fertility rates.

gdpp(int): GDP per capita. It is calculated as GDP divided by population.

Objectives:

This project attempts to make use of multiple clustering approaches to cluster the countries in the dataset. The idea behind attempting this is to find clusters of countries according to socioeconomic and health indicators, so that countries that suffer from the most problems in these areas may be accurately determined. This can be useful for organisations that focus on alleviating these issues such as NGOs, UN and development agencies. It can also be useful for researchers and practitioners who can use such clustering results as starting points for investigations. For instance, similarities and differences between countries in a cluster as well as countries in other clusters can provide actionable insights and identify issues that can be investigated further.

The project aims to apply clustering techniques to resolve the business case discussed above. Although the project produces some results, the exact evaluation of the results is not part of it. Instead, the project remains primarily concerned with making use of unsupervised learning techniques. Some discussion of the results obtained will be done but in context of its aim.

Data Exploration:

The dataset was initially in csv format but was loaded into a Pandas dataframe. Pandas is a very useful library in Python for data preprocessing. It chiefly makes use of Dataframes, a built-in data structure, for performing data preprocessing and analysis in an efficient manner.

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

```
df.head()
```

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200

With the dataset successfully loaded in the dataframe, some Exploratory Data Analysis was conducted on it.

```
df.shape
```

```
(167, 10)
```

The dataset is not large at all.

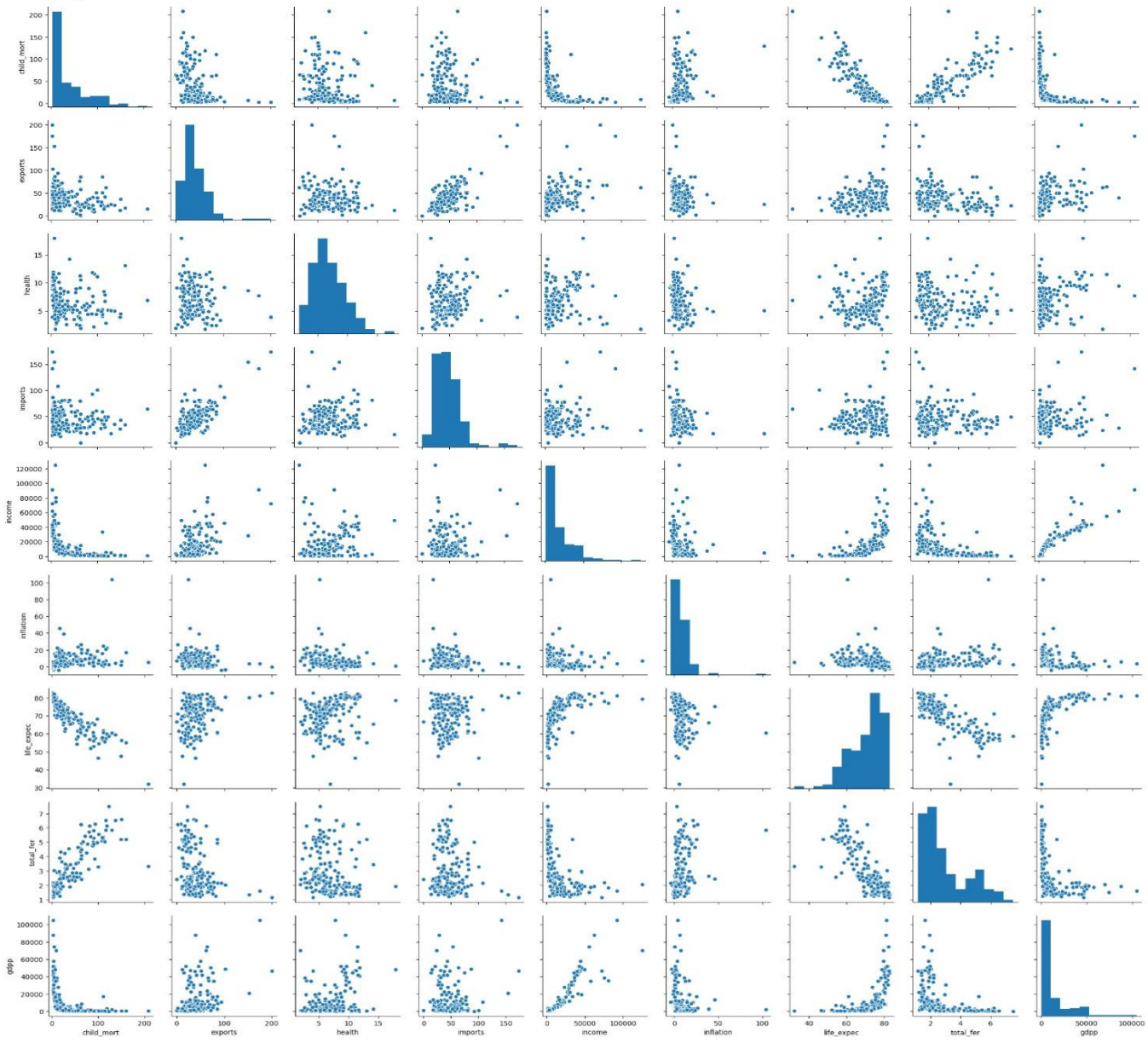
```
df.info()
```

```
<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
```

All the features or columns in the dataset are either integers or floats except for the first column which is a string representing the name of a country.

```
sns.pairplot(df)
```

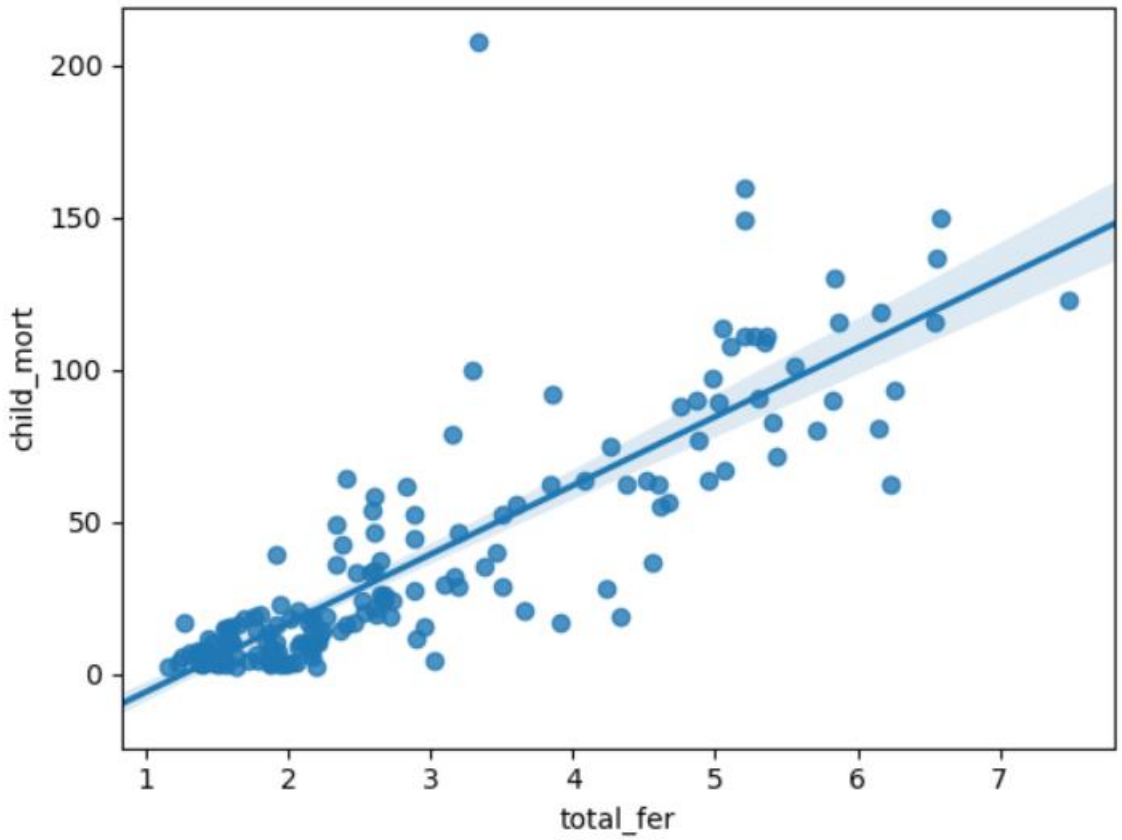
```
<seaborn.axisgrid.PairGrid at 0x7f8c70ee1050>
```



As the number of features is small, the pairplot functionality can be leveraged. The pairplots provide significant insights into the data. For instance, child mortality and total fertility appear to be positively linearly correlated.

```
sns.regplot(x=df['total_fer'], y=df['child_mort'])
```

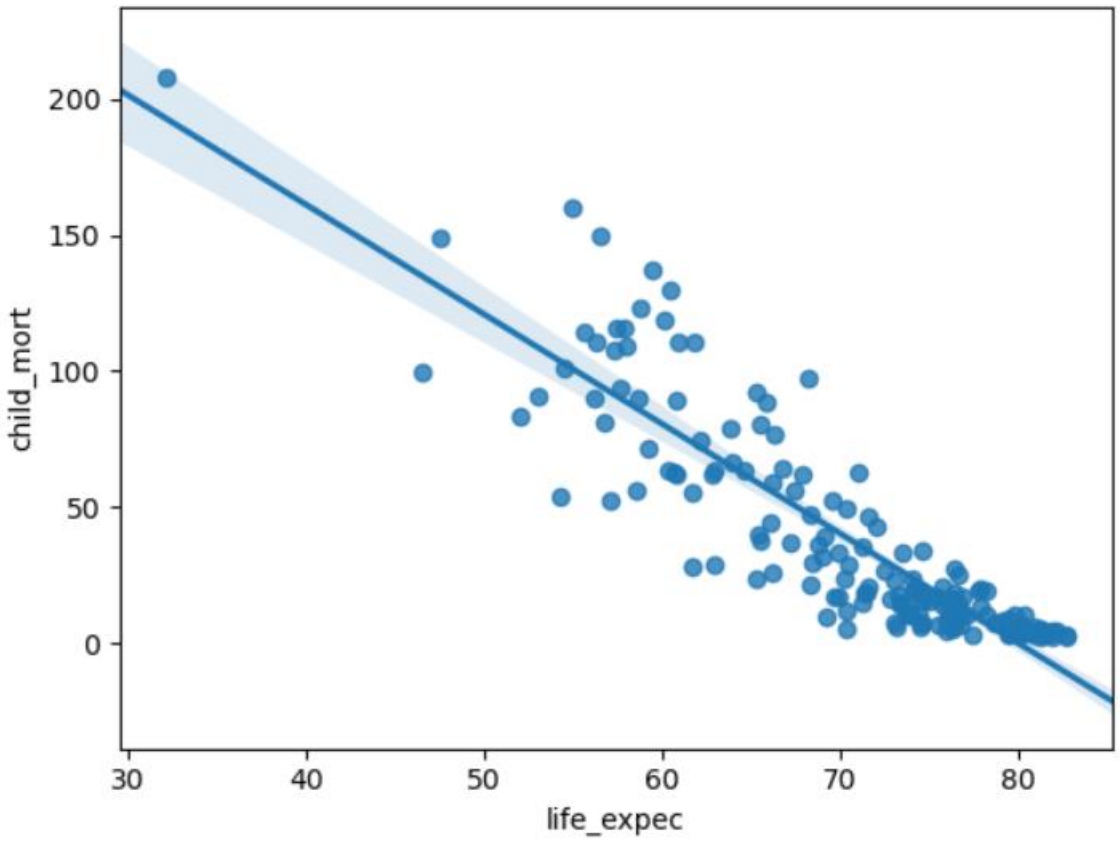
```
<AxesSubplot:xlabel='total_fer', ylabel='child_mort'>
```



On the other hand, life expectancy and child mortality appear to be negatively linearly correlated.

```
sns.regplot(x=df['life_expec'], y=df['child_mort'])
```

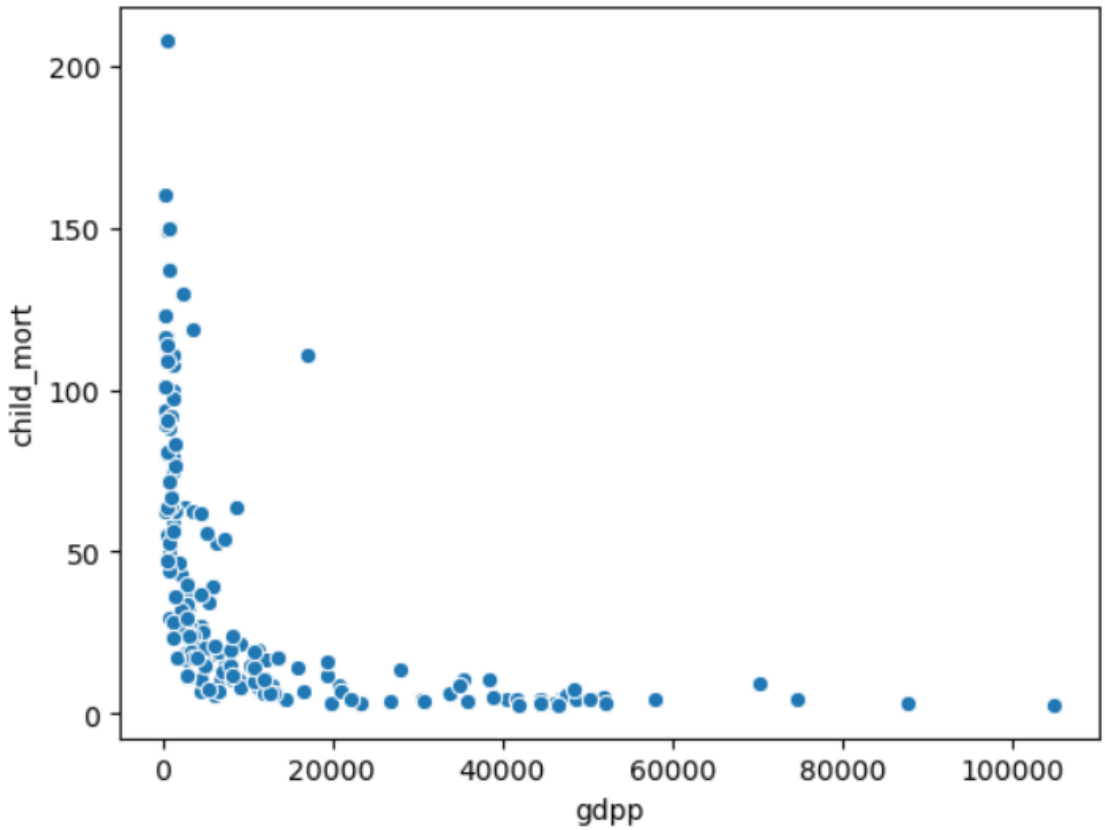
```
<AxesSubplot:xlabel='life_expec', ylabel='child_mort'>
```



There are some non-linear relationships in the data as well. GDP per person (gdpp) appears to be negative exponentially related to child mortality.

```
sns.scatterplot(x=df['gdpp'], y=df['child_mort'])
```

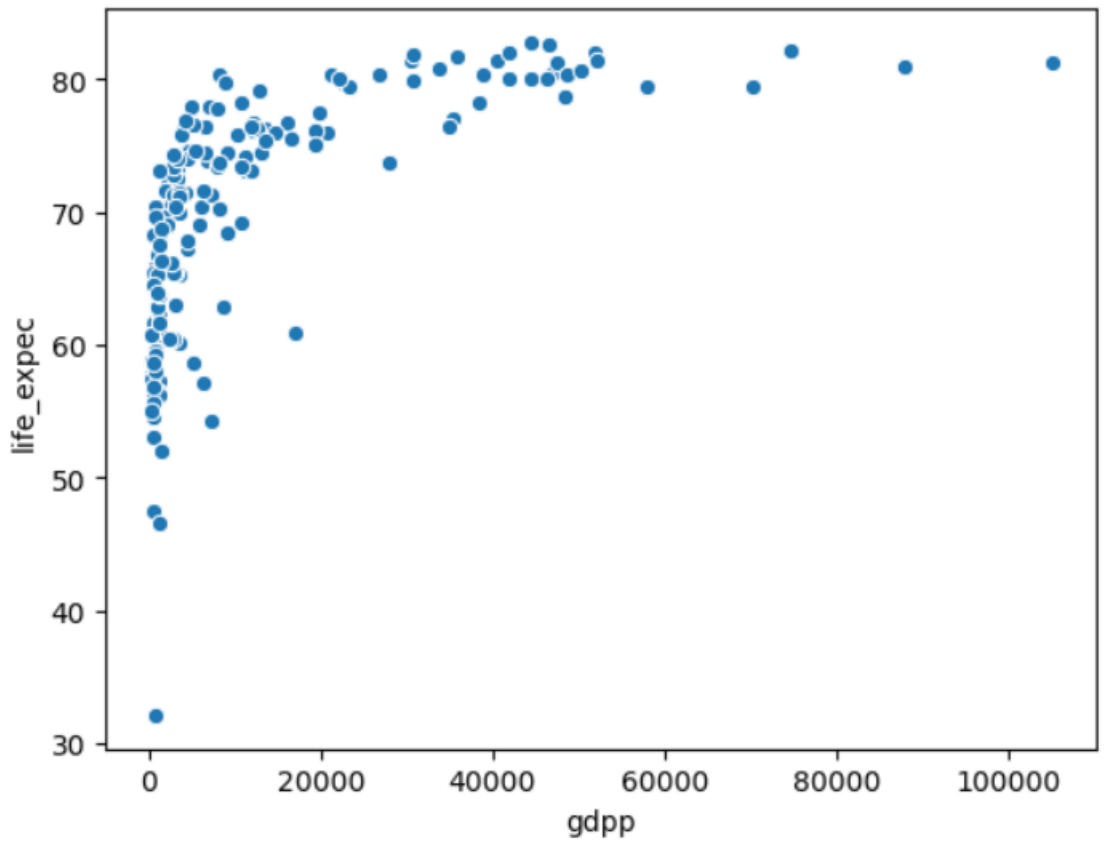
```
<AxesSubplot:xlabel='gdpp', ylabel='child_mort'>
```



A similar relationship exists between life expectancy and gdpp.

```
sns.scatterplot(x=df['gdpp'], y=df['life_expec'])
```

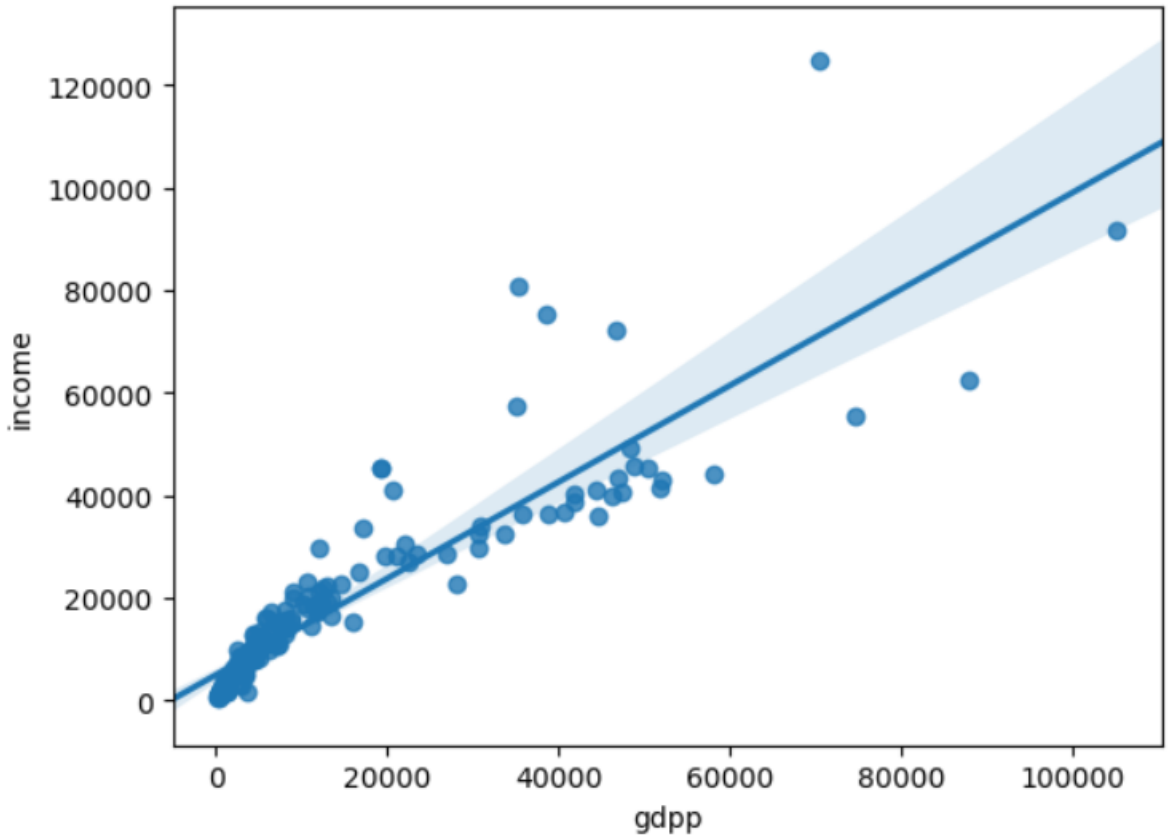
```
<AxesSubplot:xlabel='gdpp', ylabel='life_expec'>
```



There also appears to be some redundant information in the dataset. Income and gdpp are very closely related to each other.

```
sns.regplot(x=df['gdpp'], y=df['income'])
```

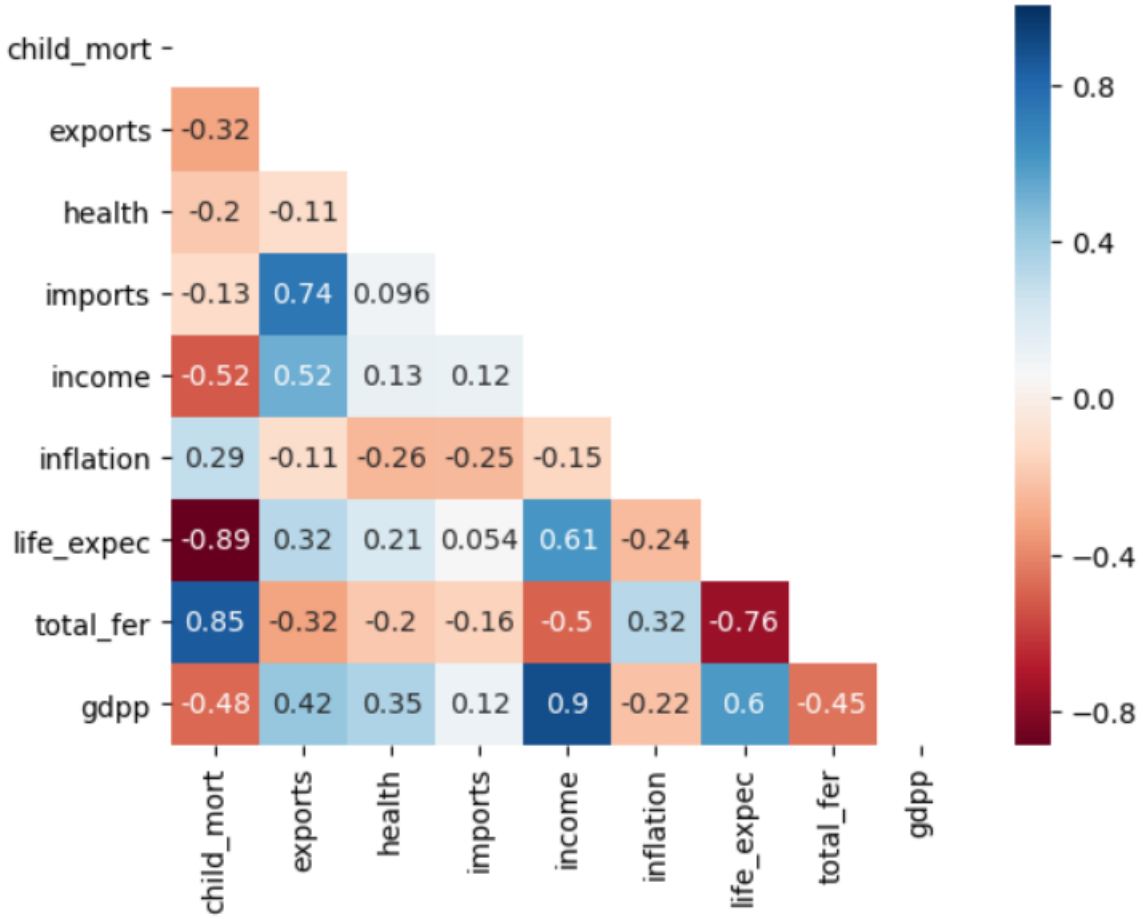
```
<AxesSubplot:xlabel='gdpp', ylabel='income'>
```



This can be confirmed by plotting a heatmap of the features in the dataset.

```
sns.heatmap(df.corr(), cmap='RdBu', annot=True, mask=np.triu(df.corr()))
```

```
<AxesSubplot:>
```



Finally, the range of values in the dataset are noted.

```
df.describe()
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000
mean	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.555689	2.947964	12964.155689
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.893172	1.513848	18328.704809
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.100000	1.150000	231.000000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.300000	1.795000	1330.000000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.100000	2.410000	4660.000000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.800000	3.880000	14050.000000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.800000	7.490000	105000.000000

There is considerable variation in the range of values of features in the dataset. In some features, it goes from hundreds to thousands; while in others it ranges between 1 and 10.

Also, there are no missing values in the dataset.

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

```
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
```

Feature Engineering:

The dataset is relatively clean and lacks any missing values. This is very convenient as missing data can cause many issues. Most of the data is in numeric form already which is very good as well since most Machine Learning models require their inputs to be in numeric form. Only the country column is in string form. Since this column is only an identifier, it was dropped for the preparing the feature set.

```
df2 = df.drop("country", axis=1)
df2.head()
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460
3	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530
4	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200

Next, the EDA indicated that the range of values in the dataset's features differed considerably from each other. Unsupervised learning models make extensive use of

distance-based metrics which can cause problems with such a dataset. Therefore, the feature set was standardized.

```
scaler = MinMaxScaler()
X = scaler.fit_transform(df2)
X[:5]
```

```
array([[0.42648491, 0.04948197, 0.35860783, 0.25776487, 0.00804721,
        0.12614361, 0.47534517, 0.73659306, 0.00307343],
       [0.06815969, 0.13953104, 0.29459291, 0.27903729, 0.07493307,
        0.08039922, 0.87179487, 0.07886435, 0.03683341],
       [0.12025316, 0.1915594 , 0.14667495, 0.18014926, 0.0988094 ,
        0.1876906 , 0.87573964, 0.27444795, 0.04036499],
       [0.56669912, 0.31112456, 0.06463642, 0.24626626, 0.04253523,
        0.24591073, 0.55226824, 0.79022082, 0.03148832],
       [0.03748783, 0.22707876, 0.2622747 , 0.33825512, 0.14865223,
        0.05221329, 0.8816568 , 0.15457413, 0.11424181]])
```

With this step, the feature set was ready for unsupervised learning.

Clustering:

With the feature set prepared, the clustering can proceed.

K-Means:

The first technique used for clustering was the K-Means technique. K-Means is a hard clustering technique which requires that the number of clusters be pre-defined. The number of clusters was chosen to be 2, turning the business question into a binary one: Which countries have bad socioeconomic indicators and which countries have good socioeconomic indicators?

```
num_clusters = 2
km = KMeans(n_clusters=num_clusters)
km.fit(df2)
km.labels_
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

The result successfully divided the dataset into two clusters. However, this posed a question: Was 2 clusters the best option for this data? To determine the optimal number

of clusters for the dataset. The inertia of the dataset was used. Inertia is a measure of dispersion of the data in the dataset.

```
km.inertia_
```

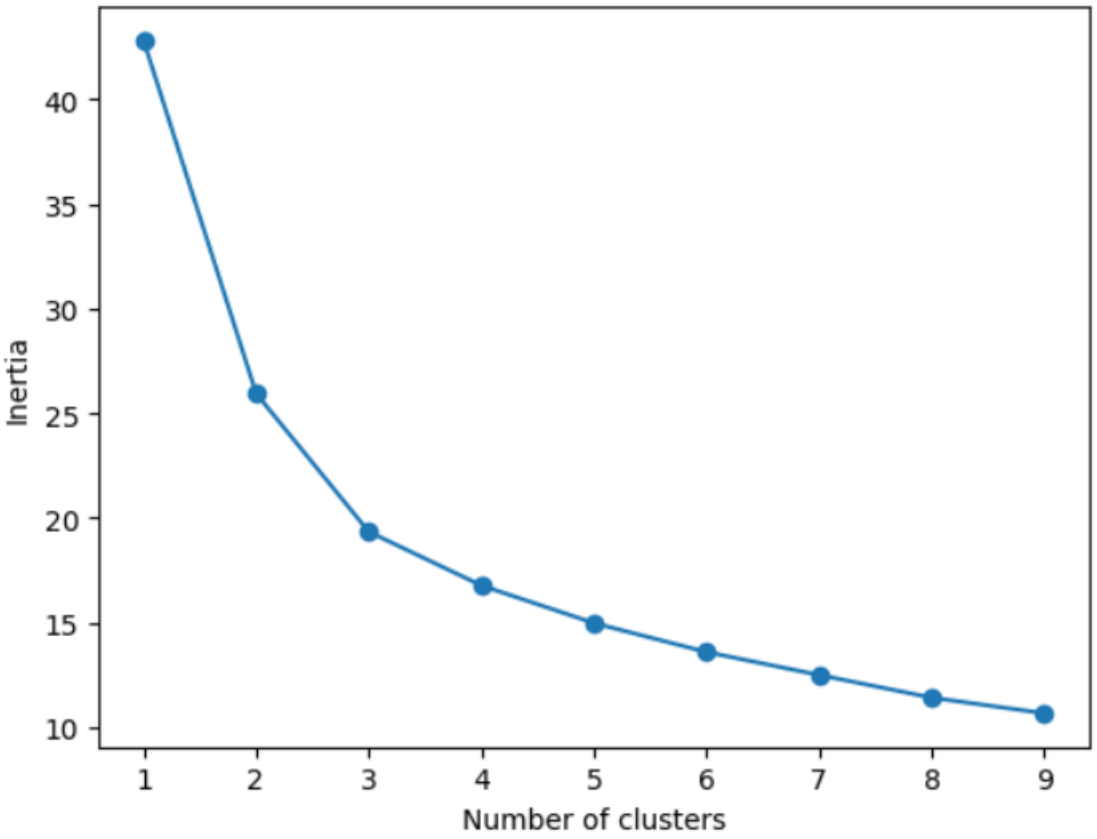
```
36528387934.3221
```

The inertia along with the elbow method was used to determine which number of clusters is the optimal one. The elbow method plots the inertia of the results for each number of clusters and where the ‘elbow’ of the plot occurs, the optimal number of clusters may be determined. The ‘elbow’ occurs when the decrease in inertia due to increasing number of clusters is not sharp.

```
num_clusters = range(1,10)
inertia=[]
for num in num_clusters:
    km = KMeans(init='k-means++', random_state=1, n_clusters=num)
    km.fit(X)
    inertia.append(km.inertia_)
```

```
plt.plot(num_clusters, inertia)
plt.scatter(num_clusters, inertia)
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
```

```
Text(0, 0.5, 'Inertia')
```



The elbow plot shown here indicates that while the change in inertia was a sharp decrease in the case of increasing the number of clusters to 2 and 3, it becomes a smooth fall for number of clusters beyond that. Visually, 3 appears to be the optimal number of clusters.

Applying the k-means algorithm with 3 clusters:

```
k_opt = KMeans(init='k-means++', random_state=1, n_clusters=3)
k_opt.fit(X)
k_opt.labels_
```

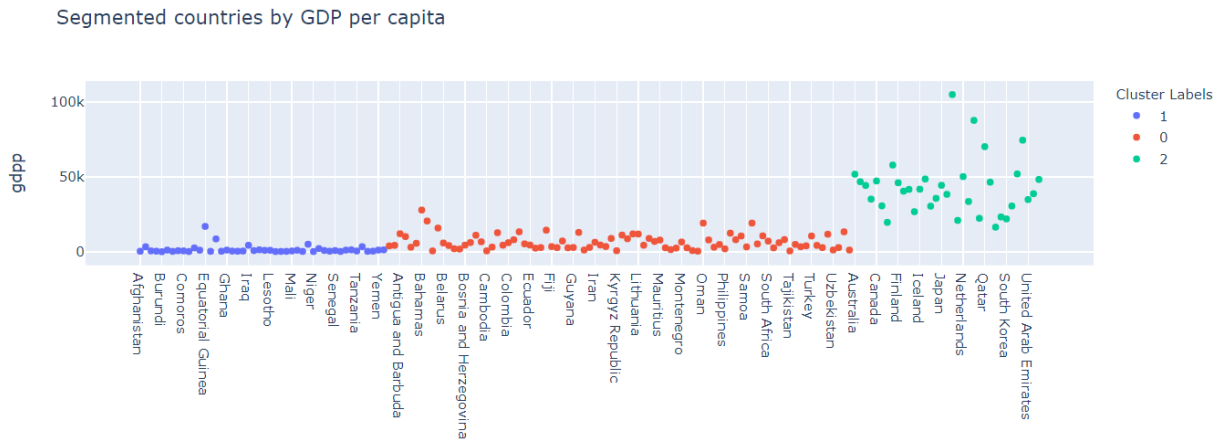
```
array([1, 0, 0, 1, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0,
       0, 2, 0, 1, 1, 0, 1, 2, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 2, 2,
       2, 0, 0, 0, 0, 1, 1, 0, 0, 2, 2, 1, 1, 0, 2, 1, 2, 0, 0, 1, 1, 0,
       1, 0, 2, 0, 0, 0, 1, 2, 2, 2, 0, 2, 0, 0, 1, 1, 2, 0, 1, 0, 0, 1,
       1, 0, 0, 2, 0, 1, 1, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       2, 2, 1, 1, 2, 0, 1, 0, 0, 0, 0, 0, 2, 2, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 2, 2, 2, 1, 0, 2, 2, 0, 0, 1, 0, 2, 2, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 2, 2, 2, 0, 0, 0, 0, 0, 1, 1], dtype=int32)
```

The labels indicate that the algorithm successfully divided the dataset into three clusters. Now, applying these labels to the original dataset reveals which countries were clustered together.

```
df3 = df.copy()
df3['Cluster Labels'] = k_opt.labels_
df3['Cluster Labels'] = df3['Cluster Labels'].astype('str')
```

The labels were converted into string type from integers because this allows them to be plotted as categorical variables in Plotly, a visualizing library for data in Python.

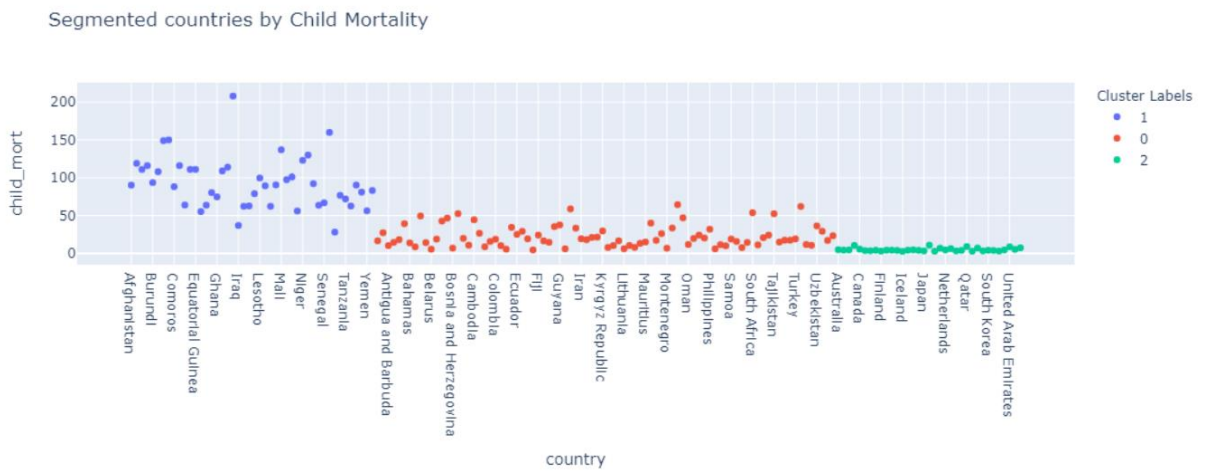
```
px.scatter(df3, x='country', y='gdpp', color='Cluster Labels', title='Segmented countries by GDP per capita')
```



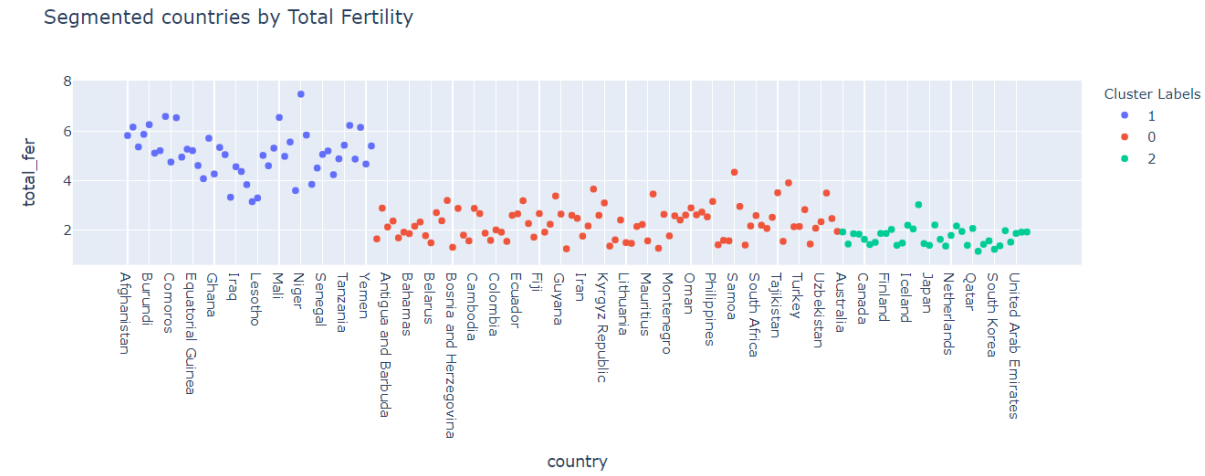
Notice that countries with a higher gdpp are mostly clustered together. Also, countries with very low gdpp are clustered together a well. This is accurate as countries with higher gdpp tend to have better socioeconomic conditions.

Additional features for clustered data are plotted as well.

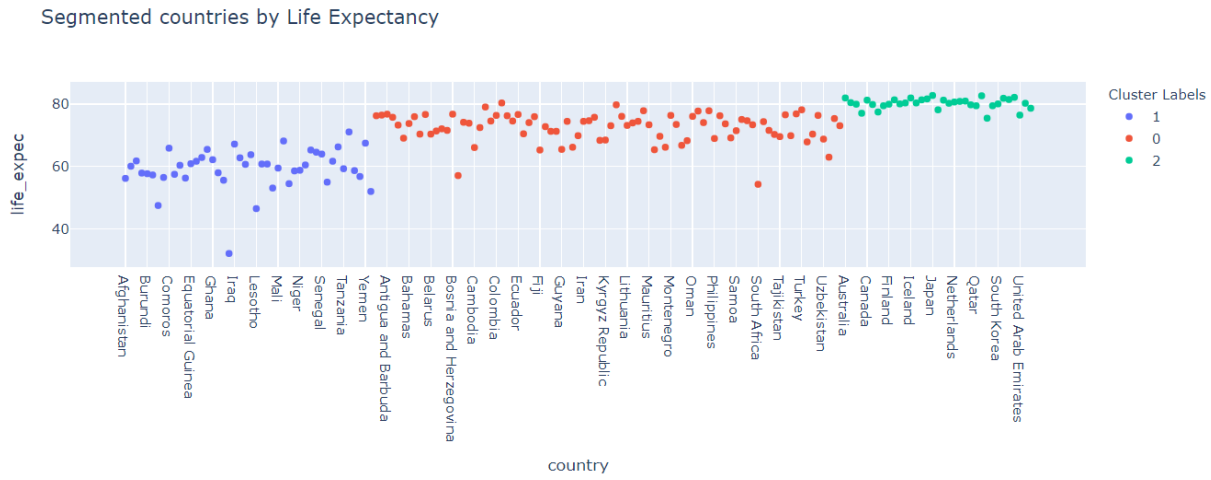
```
px.scatter(df3, x='country', y='child_mort', color='Cluster Labels', title='Segmented countries by Child Mortality')
```




```
px.scatter(df3, x='country', y='total_fer', color='Cluster Labels', title='Segmented countries by Total Fertility')
```



```
px.scatter(df3, x='country', y='life_expec', color='Cluster Labels', title='Segmented countries by Life Expectancy')
```



Note that countries with very low child mortality are clustered together while the same holds true for countries with very high child mortality. Total fertility is somewhat mixed when it comes to countries with lower total fertility. However, countries with a higher total fertility have also been clustered together. Finally, there is a clear trend in life expectancy with countries clustered together although there are outliers.

All in all, it can be seen that K-Means has done a relatively decent job in clustering countries together. For the business case, this will allow the development agencies and NGOs to safely ignore countries in cluster 2 and spare most of their efforts in cluster 1. For cluster 0, countries will have to be evaluated individually though as these countries have problems as well but they are less severe in comparison to countries in cluster 1.

DBSCAN:

Next DBSCAN, another clustering algorithm is used. This algorithm does not require any input for the number of clusters. However, it has two parameters that need to be tuned: a neighborhood distance represented by eps and the number of samples in this neighborhood to consider represented by min_samples.

```
db = DBSCAN(eps=0.25, min_samples=12, metric='euclidean')
db_mol = db.fit(X)
db_mol.labels_
```

```
array([ 1,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        1,  0,  0,  0, -1,  0, -1,  0,  1, -1,  0,  1,  0,  0, -1, -1,  0,
        0,  0, -1,  1, -1,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0, -1, -1,
        0,  0,  0,  0, -1,  1,  0,  0,  1,  0,  0,  0,  1,  1,  0, -1,  0,
        0,  0,  0,  0, -1, -1,  0,  0,  0,  0,  0,  0,  1, -1, -1,  0,  0,
        0,  0, -1, -1,  0,  0, -1,  0,  1,  1,  0,  0,  1, -1, -1,  0, -1,
        0, -1,  0,  0,  1,  0,  0,  0,  0,  0, -1, -1, -1, -1, -1,  0,  0,
        0,  0,  0,  0, -1,  0,  0, -1,  0, -1,  1,  0, -1, -1, -1,  0,  0,
       -1, -1,  0,  0,  0,  0,  1,  0,  0, -1,  0,  1,  0, -1,  1,  0,  0,
        0, -1,  1,  0, -1,  0, -1,  0,  0,  0, -1,  0, -1,  1])
```

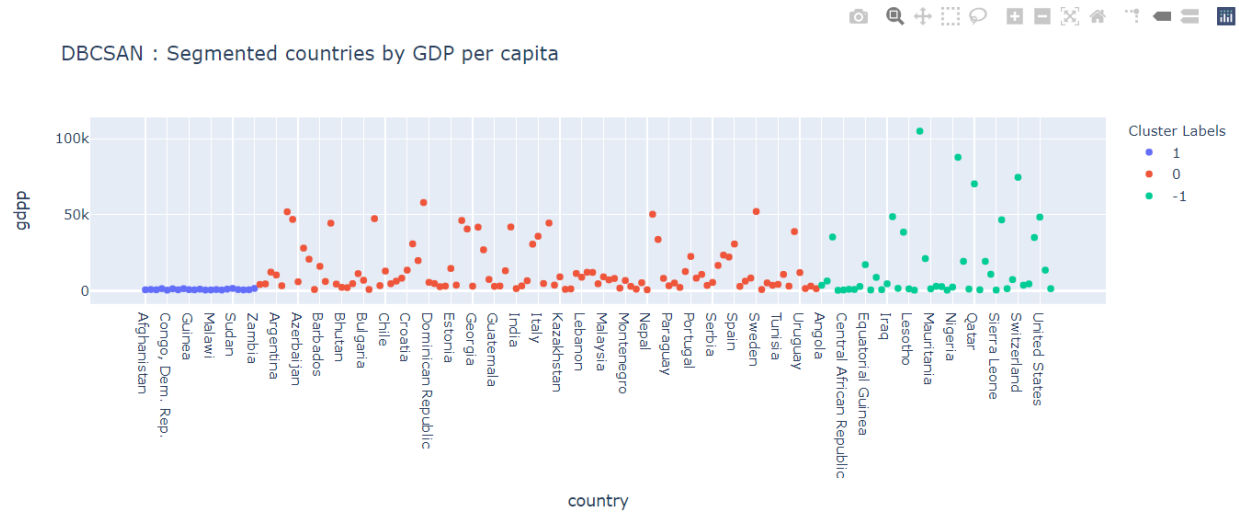
After applying the algorithm, the labels are applied to the dataset.

```
df_db = df.copy()
df_db['Cluster Labels'] = db_mol.labels_
df_db['Cluster Labels'] = df_db['Cluster Labels'].astype('str')
df_db.head()
```

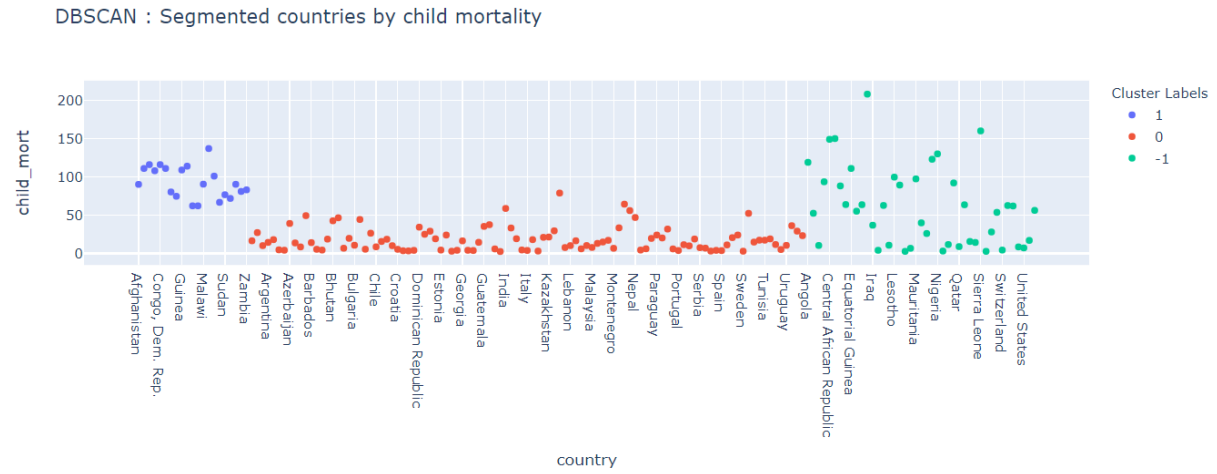
	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Labels
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553	1
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090	0
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460	0
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530	-1
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200	0

And then plotted.

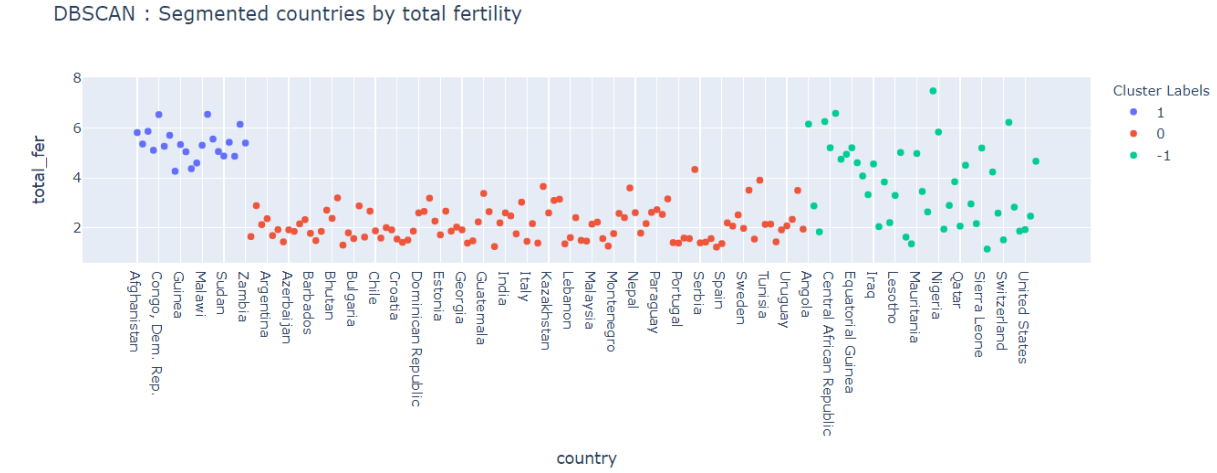
```
px.scatter(df_db, x='country', y='gdpp', color='Cluster Labels', title='DBCSAN : Segmented countries by GDP per capita')
```



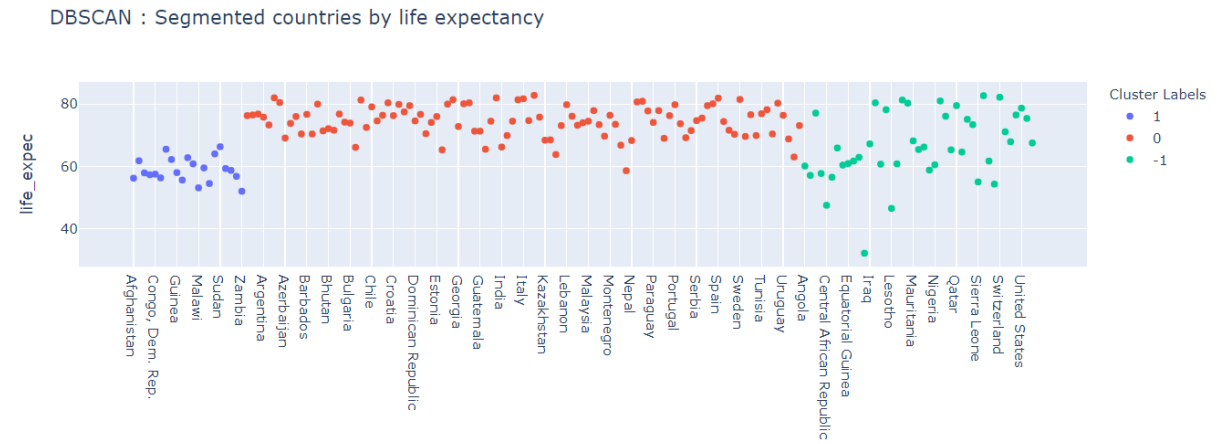
```
px.scatter(df_db, x='country', y='child_mort', color='Cluster Labels', title='DBSCAN : Segmented countries by child mortality')
```




```
px.scatter(df_db, x='country', y='total_fer', color='Cluster Labels', title='DBSCAN : Segmented countries by total fertility')
```




```
px.scatter(df_db, x='country', y='life_expec', color='Cluster Labels', title='DBSCAN : Segmented countries by life expectancy')
```



The results are quite mixed. Although the algorithm seems to have done a decent job in clustering countries with lower socioeconomic conditions, it seems to have run into problems even there. This may be because DBSCAN is capable of sifting noise from data but in this case, it is not helping the business case as many countries even with poor socioeconomic conditions are not added. Choosing the right parameters for the algorithm is very difficult as well. The above results were acquired after considerable experimentation.

Hierarchical Agglomerative Clustering:

Afterwards, the Hierarchical Agglomerative Clustering (HAC) is used to cluster the data. This algorithm also takes the number of clusters as input. As it is already known that 3 is the most optimal number of clusters for reducing inertia, this is passed as input to the algorithm. Furthermore, the algorithm also takes multiple approaches to creating clusters. For the purposes of this project, 'ward' linkage approach is taken. This approach tends to minimize inertia.

```
hac3 = AgglomerativeClustering(n_clusters = 3, affinity='euclidean', linkage='ward')
hac3_fit = hac3.fit(X)
hac3_fit.labels_
```

```
array([1, 2, 2, 1, 2, 2, 2, 0, 0, 2, 2, 0, 2, 2, 2, 0, 2, 1, 2, 2, 2, 2,
       2, 0, 2, 1, 1, 2, 1, 0, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 1, 2, 2, 2,
       0, 2, 2, 2, 2, 1, 1, 2, 2, 0, 0, 1, 1, 2, 0, 1, 0, 2, 2, 1, 1, 2,
       1, 2, 0, 2, 2, 2, 2, 0, 0, 0, 2, 0, 2, 2, 1, 2, 0, 2, 1, 2, 2, 2,
       2, 0, 2, 0, 2, 1, 1, 2, 2, 1, 0, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 2,
       0, 0, 1, 1, 0, 0, 1, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 1, 2, 2,
       1, 0, 2, 2, 2, 2, 2, 0, 2, 2, 1, 2, 0, 0, 2, 1, 2, 1, 1, 2, 2, 2,
       1, 1, 2, 0, 0, 0, 2, 2, 2, 2, 2, 1, 1])
```

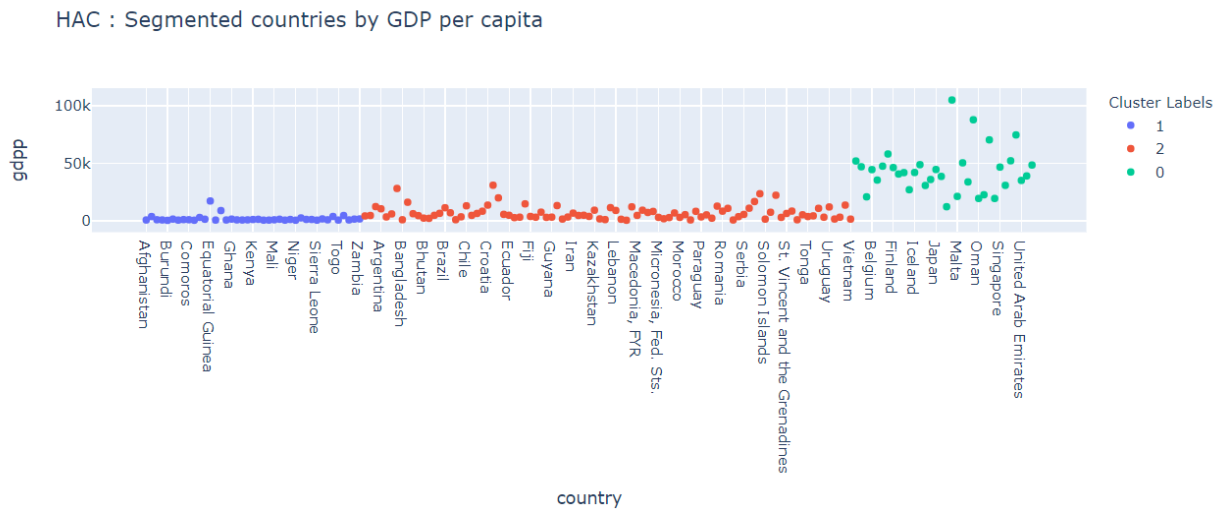
Next, labels are added to the dataset.

```
df_hac3 = df.copy()
df_hac3['Cluster Labels'] = hac3_fit.labels_
df_hac3['Cluster Labels'] = df_hac3['Cluster Labels'].astype('str')
df_hac3.head()
```

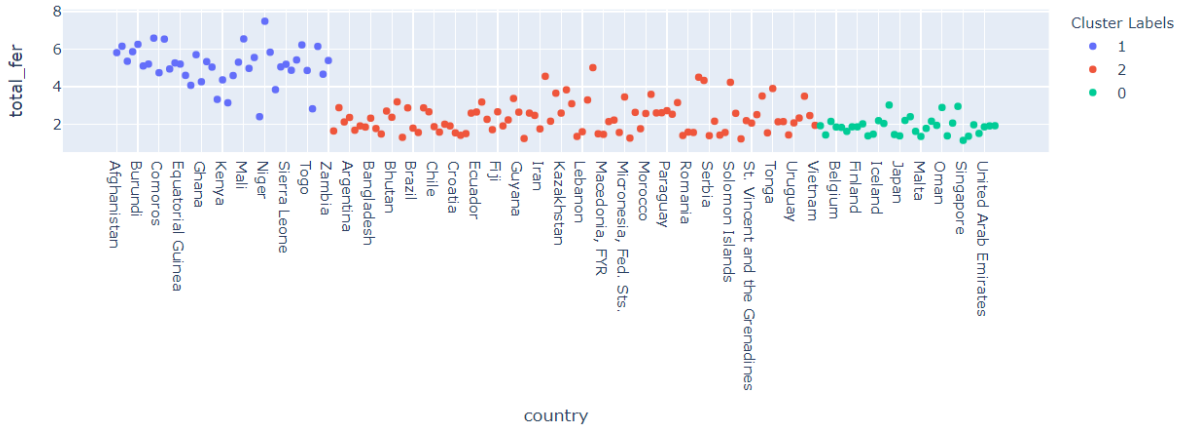
	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Labels
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553	1
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090	2
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460	2
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530	1
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200	2

And then the results are plotted.

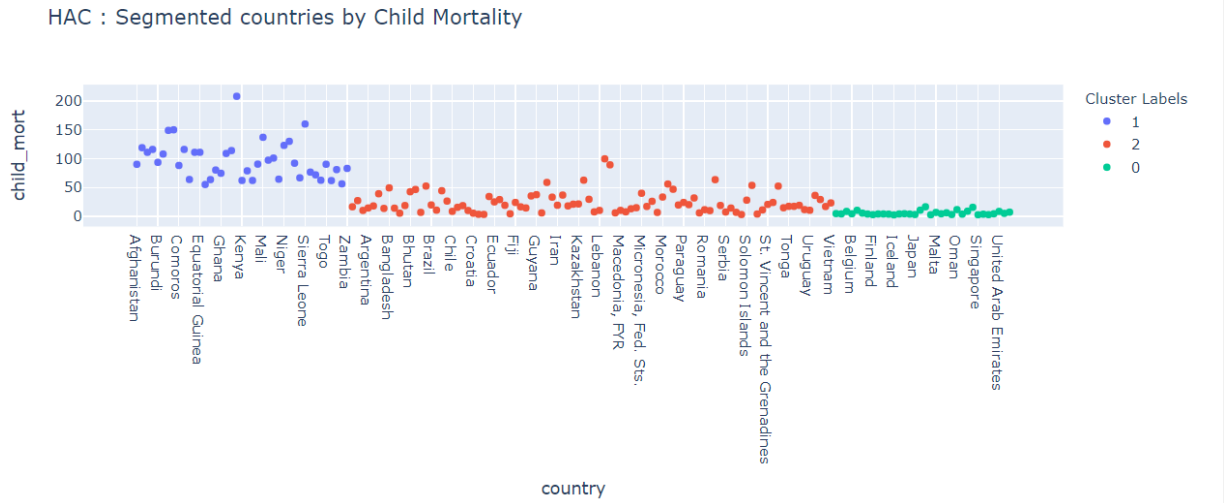
```
px.scatter(df_hac3, x='country', y='gdpp', color='Cluster Labels', title='HAC :_Segmented_countries_by_GDP_per_capita')
```



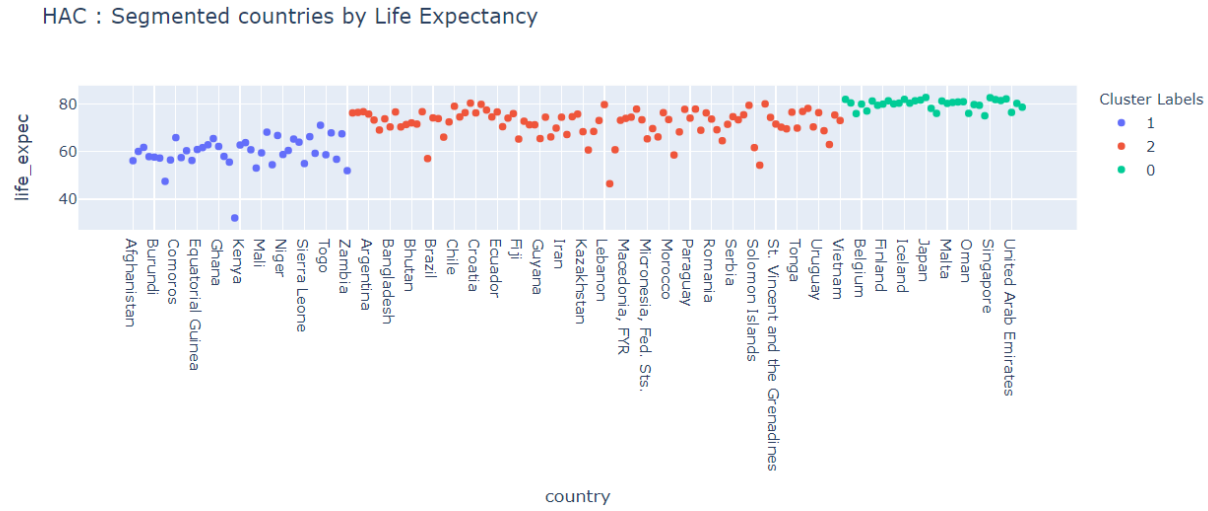

```
px.scatter(df_hac3, x='country', y='total_fer', color='Cluster Labels', title='HAC : Segmented countries by Total Fertility')
```



```
px.scatter(df_hac3, x='country', y='child_mort', color='Cluster Labels', title='HAC : Segmented countries by Child Mortality')
```




```
px.scatter(df_hac3, x='country', y='life_expec', color='Cluster Labels', title='HAC : Segmented countries by Life Expectancy')
```



The results given by HAC are almost the same as those given by K-Means. Both algorithms had no trouble clustering countries according to their socioeconomic indicators.

Mean Shift:

The final clustering technique to be considered is Mean Shift clustering. This technique is similar to DBSCAN in that it does not need the number of clusters to be specified beforehand. However, like DBSCAN, it needs some parameters to be tuned. In this case,

it is bandwidth. For bandwidth, quantile and number of samples (n_samples) need to be specified beforehand.

Applying Mean Shift,

```
band = estimate_bandwidth(X, quantile=0.56, n_samples=6)
band
```

```
0.6522663639810129
```

```
ms = MeanShift(bandwidth=band, bin_seeding=True)
ms_mod = ms.fit(X)
ms_mod.labels_
```

```
array([0, 0, 0, 1, 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, 1, 0, 0, 0, 0, 1, 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, 0,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 2, 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])
```

And then adding resulting cluster labels to the data:

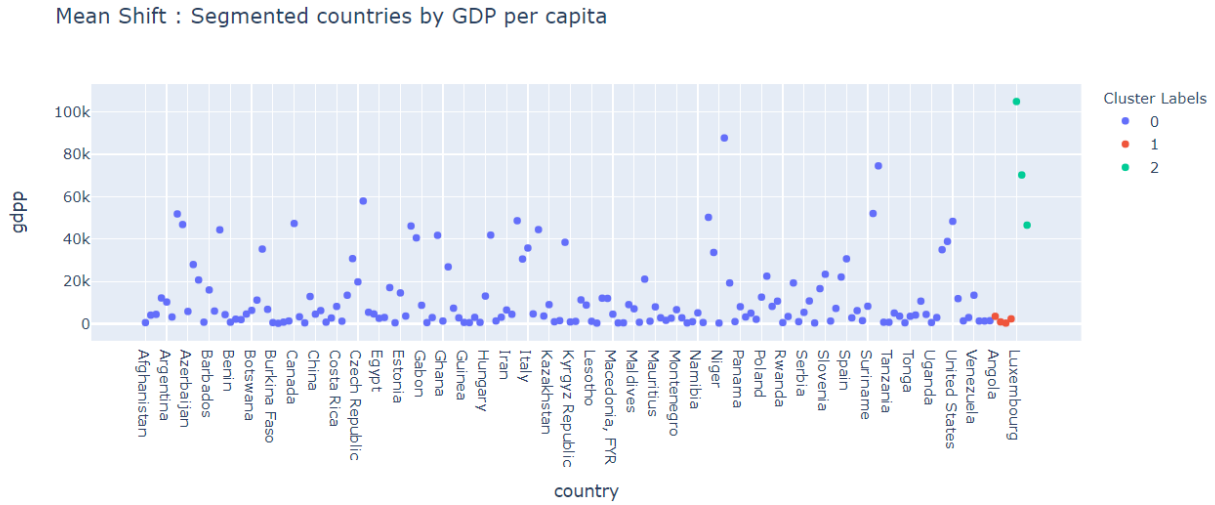
```
df_ms = df.copy()
df_ms['Cluster Labels'] = ms_mod.labels_
df_ms['Cluster Labels'] = df_ms['Cluster Labels'].astype('str') # This is to plot the values in Plotly
df_ms.head()
```

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Labels
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553	0
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090	0
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460	0
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530	1
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200	0

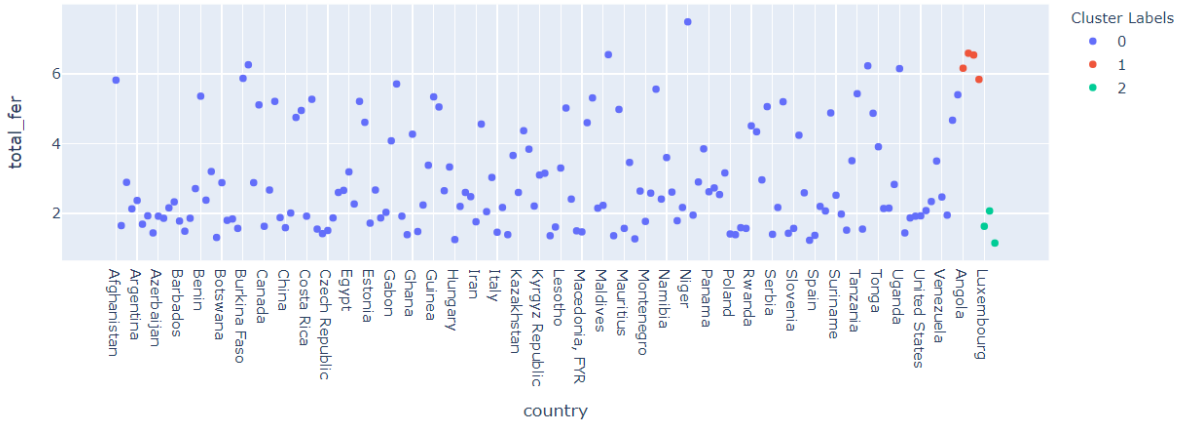
```
px.scatter(df_ms, x='country', y='gdpp', color='Cluster Labels', title='Mean Shift : Segmented countries by GDP per capita')
```

The results can be plotted,

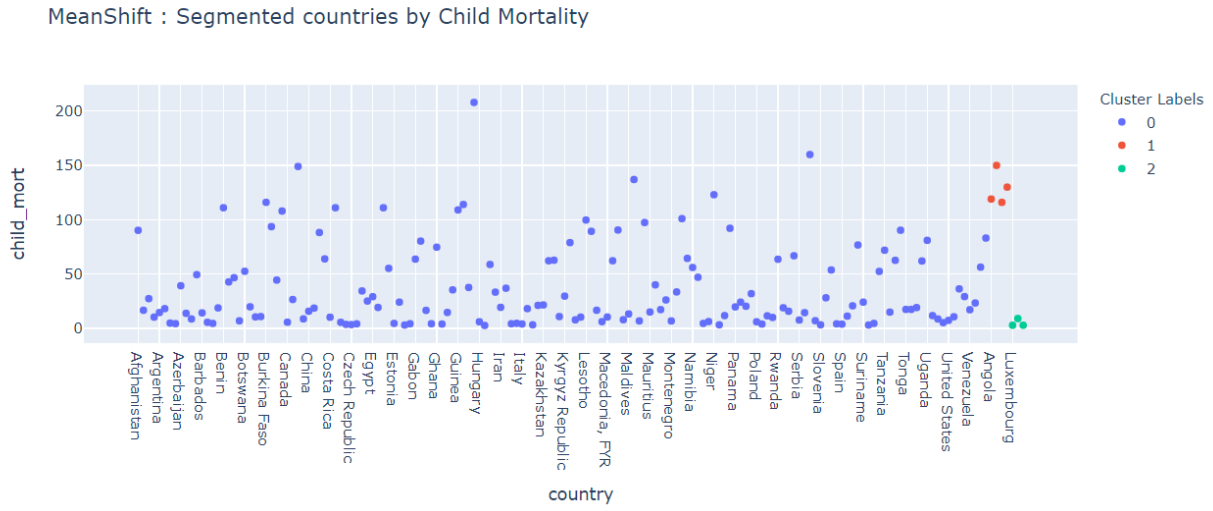
```
px.scatter(df_ms, x='country', y='gdpp', color='Cluster Labels', title='Mean Shift : Segmented countries by GDP per capita')
```



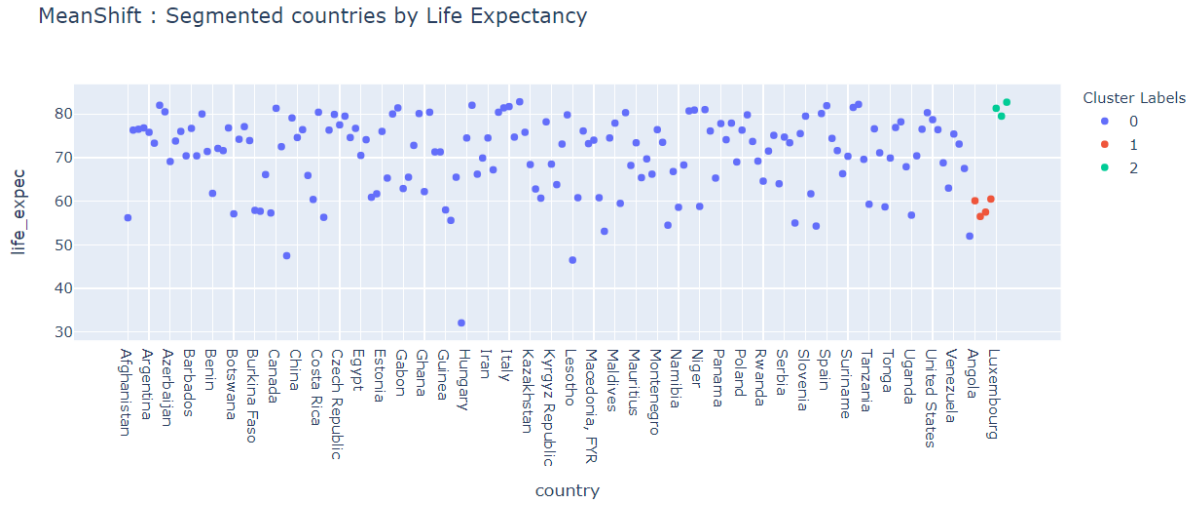

```
px.scatter(df_ms, x='country', y='total_fer', color='Cluster Labels', title='Mean Shift : Segmented countries by Total Fertility')
```



```
px.scatter(df_ms, x='country', y='child_mort', color='Cluster Labels', title='MeanShift : Segmented countries by Child Mortality')
```




```
px.scatter(df_ms, x='country', y='life_expec', color='Cluster Labels', title='MeanShift : Segmented countries by Life Expectancy')
```



As the results clearly illustrate, Mean Shift has a lot of trouble distinguishing between the points to create clusters. What it outputted for three clusters cannot be considered a good result. Although outputting multiple clusters may improve the overall clusters, it defeats the ultimate purpose of the technique to solve the business problem. Once more, the parameters are too unintuitive and shifting them to get the correct combination is difficult.

Results:

The results of applying multiple clustering techniques indicated that techniques such as K-Means and Hierarchical Agglomerative Clustering, which take the number of clusters

to segment the data in as an input, did far better than techniques such as DBSCAN and Mean Shift. Part of the problem is that techniques like DBSCAN and Mean Shift require input parameters that are not very easy to understand. They cannot be set intuitively. It is possible that parameter combinations of these techniques exist that provide better results. For the purposes of the project however, it is clear that techniques which take the number of clusters as input are superior in clustering the data properly largely because there are fewer parameters to shift.

It should be noted that it is not always possible to find the right number of clusters beforehand. Also, the dataset was too small. For larger datasets, the results produced by DBSCAN and Mean Shift may be better than those achieved here. However, there is no easy way to tune the parameters of these two algorithms. In the course of the project, it took a lot of iterations with different combinations of parameters to get the results. Making use of some metric like inertia in the case of K-Means might make it possible to receive better results by quickly honing in on the right combinations.

Key Findings:

The key findings from the project are as follows:

1. Algorithms like K-Means and Hierarchical Agglomerative Clustering, which take in the number of clusters to segment the data in as input, are easier to use in practice.
2. Algorithms like Mean Shift and DBSCAN, which have non-intuitive input parameters, are difficult to use as the ideal combination is difficult to find.
3. K-Means has an easy way to find the right number of combinations. The elbow method is quite useful for finding the 'right' number of combinations beforehand.
4. If some metric such as inertia could be used to compare different combinations for Mean Shift and DBSCAN, their results could be improved substantially by honing in on the right input parameter combinations quickly.

Issues:

There were several issues with the project which are as follows:

1. The dataset was very small. There are many more socioeconomic indicators which could have been used in addition to those present in the dataset.
2. The dataset had data for one specific year only. It would have been better to have data for multiple years. This could have provided a trend for the indicators which

would help more with the business case. Countries whose indicators show a downward trend should pose a greater concern.
3. Several techniques used in the project such as Hierarchical Agglomerative Clustering have multiple hyperparameters that could have been changed. The changed parameters could have produced different results.

Future Steps:

Some future steps for dealing with the issues discussed are outlined below:

1. A more comprehensive dataset with more indicators for multiple years could be used for a more thorough analysis.
2. Multiple hyperparameters for the different techniques considered here could be tuned for more results.
3. Additional clustering techniques could be added to the project.

Bonus:

As an added bonus, a different clustering technique in addition to the ones mentioned above will be used.

K-Medoids:

K-Medoids is a variation of the K-Means algorithm. K-Means algorithm finds the specified k number of clusters by initializing k centroids in the data and moving them around the data to get the required results. K-Medoids uses the same approach. However, the centroids created by K-Means are not necessarily data points but may be any point in the feature space. The centroids generated by K-Medoids must necessarily be points in the dataset.

K-Medoids is not available in the python Scikit-Learn package. Instead, it is present in the Scikit-Learn-Extra package. More details can be found here: https://scikit-learn-extra.readthedocs.io/en/stable/generated/sklearn_extra.cluster.KMedoids.html

```
!pip install scikit-learn-extra
```

Next, K-Medoids is initialized and the labels generated.

```
from sklearn_extra.cluster import KMedoids
```

```
k_med = KMedoids(random_state=1, n_clusters=3)
k_med.fit(X)
k_med.labels_
```

```
array([1, 2, 2, 1, 2, 0, 2, 0, 0, 2, 0, 0, 2, 0, 2, 0, 2, 1, 2, 2, 0, 2,
       0, 0, 2, 1, 1, 2, 1, 0, 2, 1, 1, 0, 2, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 2, 2, 2, 2, 1, 1, 2, 2, 0, 0, 1, 1, 2, 0, 1, 0, 2, 2, 1, 1, 2,
       1, 0, 0, 2, 2, 2, 2, 0, 0, 0, 2, 0, 2, 2, 1, 2, 0, 2, 2, 0, 0, 1,
       1, 2, 0, 0, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 0, 2, 0, 2, 1, 2, 2, 2,
       0, 0, 1, 1, 0, 2, 1, 2, 2, 2, 2, 0, 0, 0, 0, 0, 1, 2, 2, 1, 0, 2,
       1, 0, 0, 0, 2, 2, 0, 0, 2, 2, 1, 2, 0, 0, 2, 1, 2, 1, 1, 2, 2, 0,
       2, 1, 2, 0, 0, 0, 0, 2, 2, 2, 2, 1, 1])
```

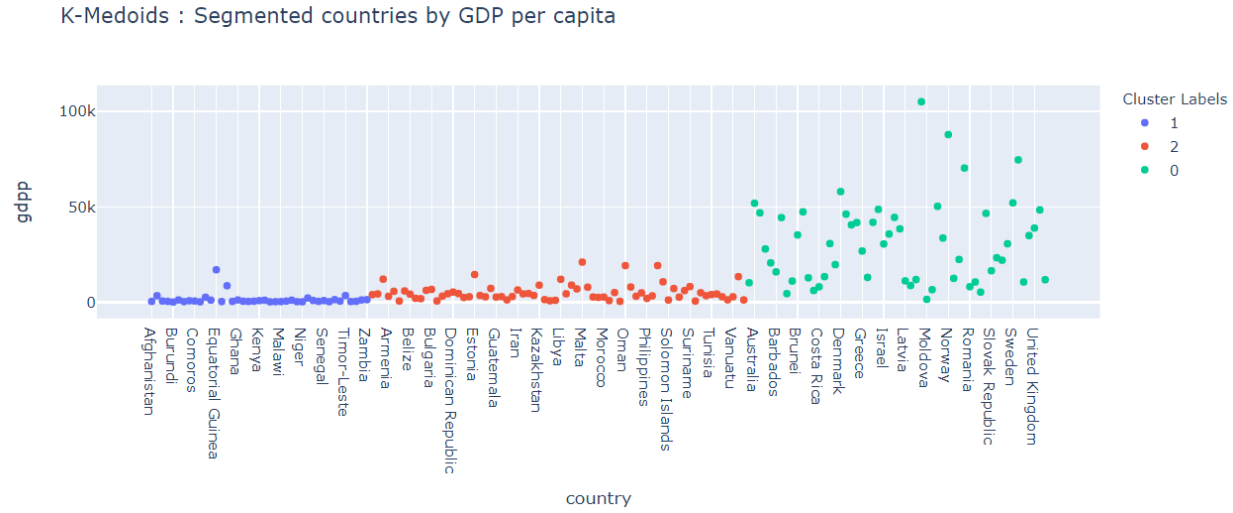
The labels are then added to the dataset.

```
df_med = df.copy()
df_med['Cluster Labels'] = k_med.labels_
df_med['Cluster Labels'] = df_med['Cluster Labels'].astype('str')
df_med.head()
```

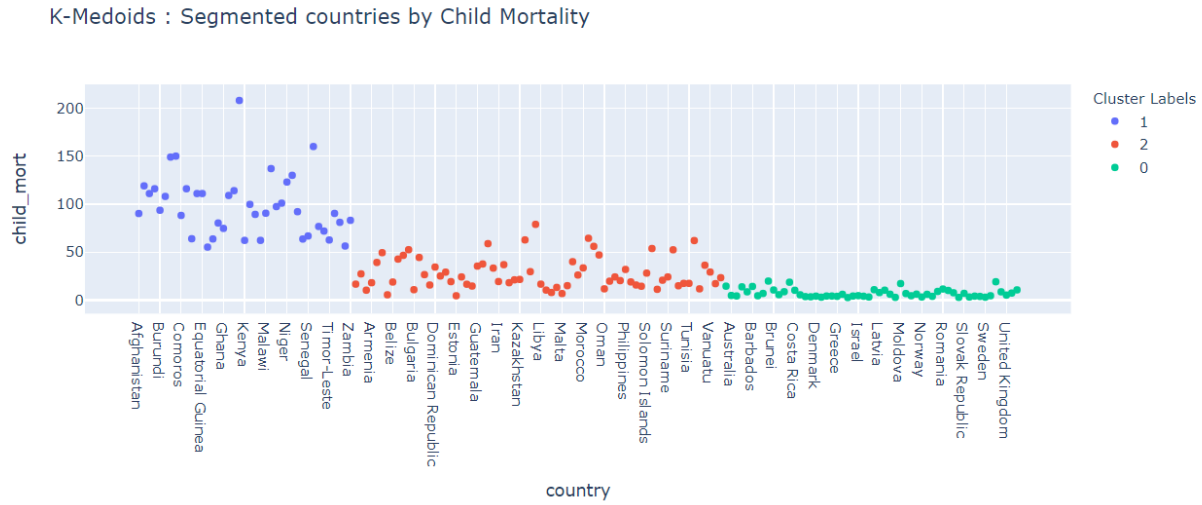
	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Cluster Labels
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553	1
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090	2
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460	2
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530	1
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200	2

Finally, the results are plotted.

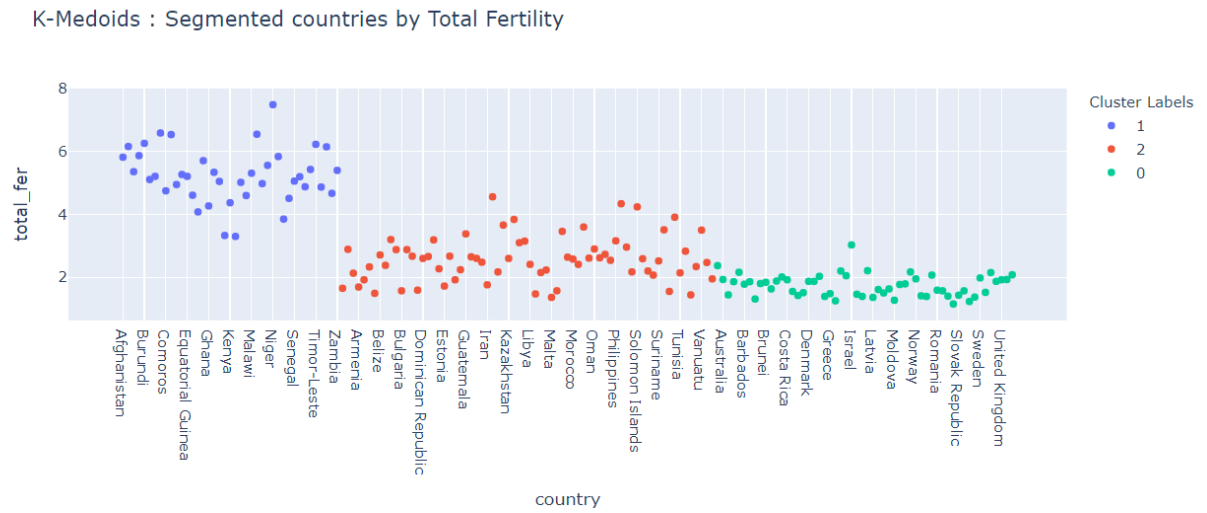
```
px.scatter(df_med, x='country', y='gdpp', color='Cluster Labels', title='K-Medoids : Segmented countries by GDP per capita')
```



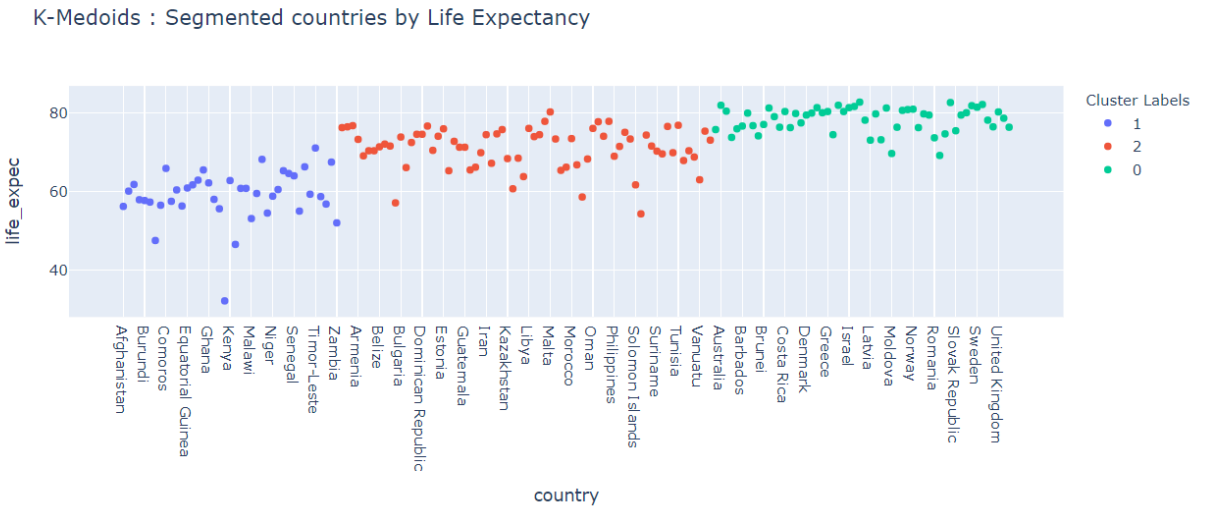

```
px.scatter(df_med, x='country', y='child_mort', color='Cluster Labels', title='K-Medoids : Segmented countries by Child Mortality')
```



```
px.scatter(df_med, x='country', y='total_fer', color='Cluster Labels', title='K-Medoids : Segmented countries by Total Fertility')
```




```
px.scatter(df_med, x='country', y='life_expec', color='Cluster Labels', title='K-Medoids : Segmented countries by Life Expectancy')
```



The results indicate that K-Medoids has done a decent job of clustering the data. The results also appear to be somewhat different from those obtained by K-Means and Hierarchical Agglomerative Clustering. While the results for countries with lower socioeconomic conditions don't seem different, there is some difference in the countries with somewhat better socioeconomic conditions.