CLUSTERING

Clustering is a way to group similar items together without using labels. Imagine sorting a box of mixed toys into groups—like all the cars in one pile and all the dolls in another. That's clustering!

Common Clustering Methods

K-Means : You pick how many groups (clusters) you want, and the algorithm tries to put similar items in each group.

Hierarchical Clustering : This method creates a tree-like structure that shows how items can be grouped together step-by-step.

DBSCAN : It looks for areas where items are close together and groups them, while ignoring items that are far away.

Gaussian Mixture Models (GMM) : It assumes the data comes from a mix of different bell-shaped curves and groups items based on that.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
data = pd.read_csv("C:/Users/Acer/Downloads/dataset - Sheet1(2).csv")
data
```

Out[4]:

	GDP per Capita	Life Expectancy	Literacy Rate	Unemployment Rate	Access to Clean Water	Poverty Rate	Country
0	1548.814	55.68434	68.20993	15.046950	61.35474	38.558030	Country 1
1	1715.189	50.18790	60.97101	21.778170	62.98282	30.117140	Country 2
2	1602.763	56.17635	68.37945	17.700080	65.69965	33.599780	Country 3
3	1544.883	56.12096	60.96098	22.351940	65.90873	37.299910	Country 4
4	1423.655	56.16934	69.76459	24.621890	65.74325	31.716300	Country 5
...	...	...	...	...	...	...	...
95	3490.305	78.81720	89.06733	9.610557	86.33461	14.979620	Country 96
96	3989.410	72.72437	90.52078	5.447473	94.80580	8.621891	Country 97
97	3065.304	73.79057	87.71653	7.029712	93.71786	9.706489	Country 98
98	3783.234	73.74296	89.55444	5.121566	90.02721	8.782452	Country 99
99	3288.398	77.48788	89.01714	6.713055	94.22348	14.795270	Country 100

100 rows × 7 columns

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   GDP per Capita        100 non-null    float64
 1   Life Expectancy       100 non-null    float64
 2   Literacy Rate         100 non-null    float64
 3   Unemployment Rate     100 non-null    float64
 4   Access to Clean Water  100 non-null    float64
 5   Poverty Rate          100 non-null    float64
 6   Country               100 non-null    object 
dtypes: float64(6), object(1)
memory usage: 5.6+ KB
```

In [6]:
```
data.shape
```

Out[6]:
```
(100, 7)
```

In [7]:
```
data.describe()
```

Out[7]:

	GDP per Capita	Life Expectancy	Literacy Rate	Unemployment Rate	Access to Clean Water	Poverty Rate
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	2499.546730	64.747678	77.746925	13.287461	77.815276	23.413789
std	809.626188	9.149988	11.245180	5.869658	10.642503	10.504556
min	1020.218000	50.187900	60.201080	5.039421	60.691670	5.360338
25%	1797.300500	56.442397	67.103633	8.397232	68.378357	13.951227
50%	2495.138000	64.441300	78.232965	11.782680	77.274045	26.099410
75%	3139.029750	72.760908	87.957125	18.253077	87.927922	32.024588
max	3990.345000	79.920110	94.944010	24.621890	94.805800	39.443720

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
GDP per Capita          0
Life Expectancy         0
Literacy Rate           0
Unemployment Rate       0
Access to Clean Water   0
Poverty Rate            0
Country                 0
dtype: int64
```

In [9]:
```
list(data)
```

Out[9]:
```
['GDP per Capita',
 'Life Expectancy',
 'Literacy Rate',
 'Unemployment Rate',
 'Access to Clean Water',
 'Poverty Rate',
 'Country']
```

In [10]:
```
features = data.drop('Country',axis=1)
features
```

Out[10]:

	GDP per Capita	Life Expectancy	Literacy Rate	Unemployment Rate	Access to Clean Water	Poverty Rate
0	1548.814	55.68434	68.20993	15.046950	61.35474	38.558030
1	1715.189	50.18790	60.97101	21.778170	62.98282	30.117140
2	1602.763	56.17635	68.37945	17.700080	65.69965	33.599780
3	1544.883	56.12096	60.96098	22.351940	65.90873	37.299910
4	1423.655	56.16934	69.76459	24.621890	65.74325	31.716300
...	...	...	...	...	...	...
95	3490.305	78.81720	89.06733	9.610557	86.33461	14.979620
96	3989.410	72.72437	90.52078	5.447473	94.80580	8.621891
97	3065.304	73.79057	87.71653	7.029712	93.71786	9.706489
98	3783.234	73.74296	89.55444	5.121566	90.02721	8.782452
99	3288.398	77.48788	89.01714	6.713055	94.22348	14.795270

100 rows × 6 columns

In [11]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler_features= scaler.fit_transform(features)
```

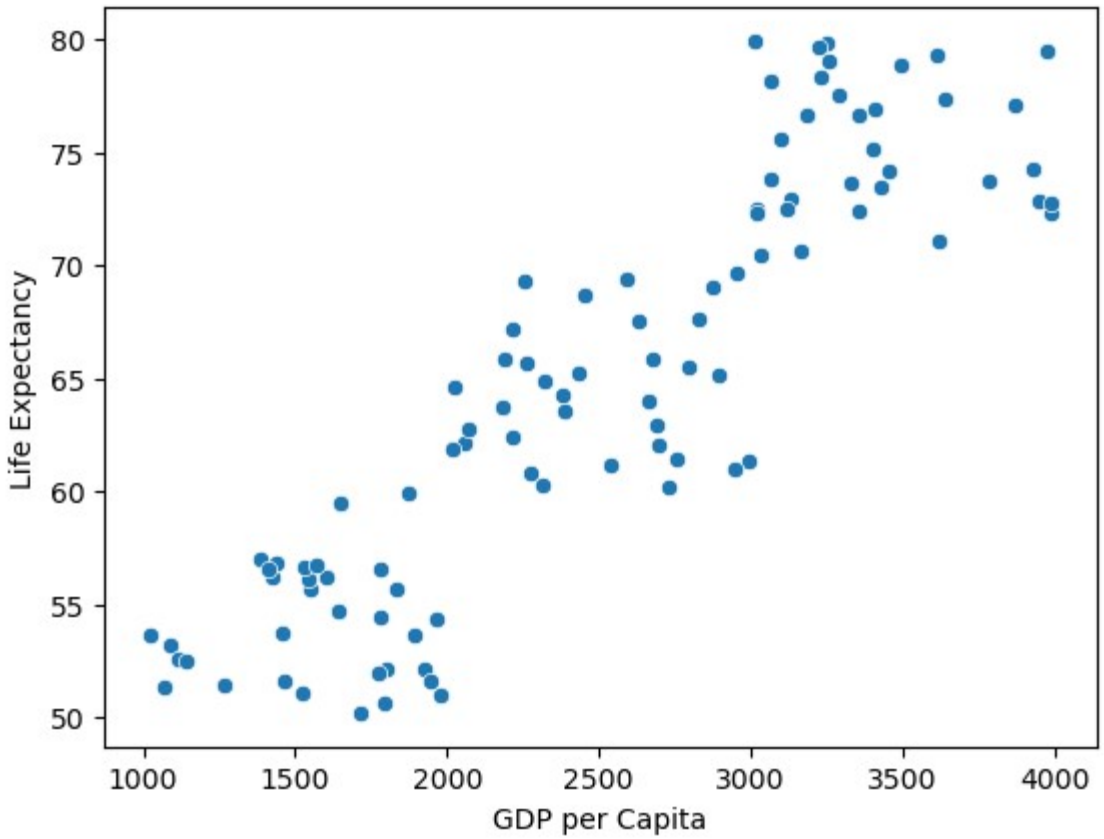
In [12]:
```
features.corr()
```

Out[12]:

	GDP per Capita	Life Expectancy	Literacy Rate	Unemployment Rate	Access to Clean Water	Poverty Rate
GDP per Capita	1.000000	0.890564	0.896906	-0.861979	0.881120	-0.874749
Life Expectancy	0.890564	1.000000	0.901298	-0.886980	0.901824	-0.886039
Literacy Rate	0.896906	0.901298	1.000000	-0.899111	0.880738	-0.892937
Unemployment Rate	-0.861979	-0.886980	-0.899111	1.000000	-0.874084	0.866927
Access to Clean Water	0.881120	0.901824	0.880738	-0.874084	1.000000	-0.912326
Poverty Rate	-0.874749	-0.886039	-0.892937	0.866927	-0.912326	1.000000

In [13]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(x='GDP per Capita', y='Life Expectancy',data=data)
plt.show()
```



WHAT IS THE RANDOM INITIALIZATION TRAP ?

RANDOM START: K-means begins by randomly selecting initial centroids from your data.

IMPACT: Different initial chooices can lead to different final clusters, which means your results can vary with run.

HOW TO MITIGATE IT

1.n_clusters - what it is : Number of clusters you want to divide your data into.

Example: If you set n_clusters=3, the algorithm will find 3 clusters in your data.

2.init - What it is : Methos for initializing the centroids (starting points) of the clusters.

Options : 'k-means++' : A smarter way to pick initail centroids .'random': Picks random points from the data.

Example: Inir 'k-means++' oftern gives better results than random

3.n_init - What it is : Number of times to run the K-means algorithm with different initial centroids.

Purpose : Help the best clustering result by tring multiple random starts.

Example: n init=10 means k-means will run 10 times with different initial centroids and pick the best result.

4. max_iter - What it is : Maximum number of iterations for each run of the K-means algoritm.

Purpose: Limits how long the algorithm will runto prevent it from running forever.

Example : max iter=300 means the algorithm will stop if it doesnt converage withoin 300 iterations

In [14]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
data['Cluster'] = kmeans.fit_predict(scaler_features)
```

```
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [15]:
```
data
```

Out[15]:

	GDP per Capita	Life Expectancy	Literacy Rate	Unemployment Rate	Access to Clean Water	Poverty Rate	Country	Cluster
0	1548.814	55.68434	68.20993	15.046950	61.35474	38.558030	Country 1	1
1	1715.189	50.18790	60.97101	21.778170	62.98282	30.117140	Country 2	1
2	1602.763	56.17635	68.37945	17.700080	65.69965	33.599780	Country 3	1
3	1544.883	56.12096	60.96098	22.351940	65.90873	37.299910	Country 4	1
4	1423.655	56.16934	69.76459	24.621890	65.74325	31.716300	Country 5	1
...	...	...	...	...	...	...	...	...
95	3490.305	78.81720	89.06733	9.610557	86.33461	14.979620	Country 96	0
96	3989.410	72.72437	90.52078	5.447473	94.80580	8.621891	Country 97	0
97	3065.304	73.79057	87.71653	7.029712	93.71786	9.706489	Country 98	0
98	3783.234	73.74296	89.55444	5.121566	90.02721	8.782452	Country 99	0
99	3288.398	77.48788	89.01714	6.713055	94.22348	14.795270	Country 100	0

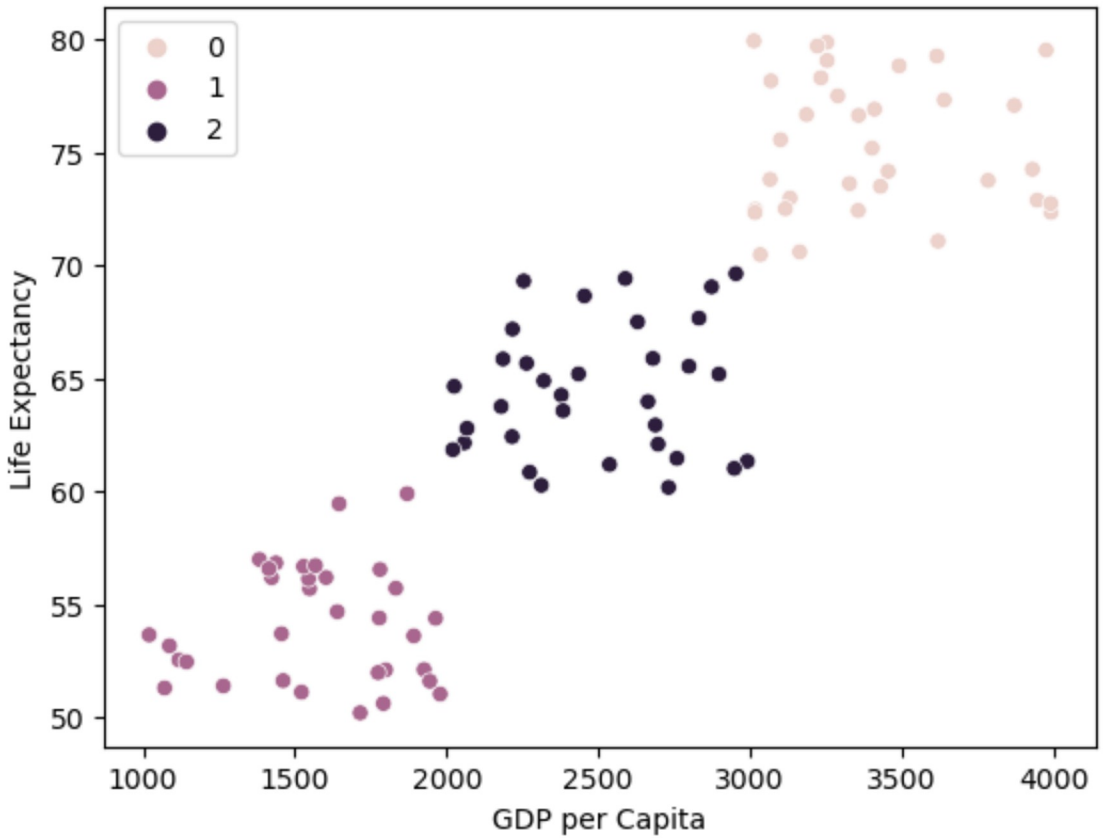
100 rows × 8 columns

In [16]:
```
labels = kmeans.labels_
labels
```

Out[16]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 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, 0])
```

In [17]:
```
sns.scatterplot(x='GDP per Capita', y='Life Expectancy',data=data, hue=data['Cluster'])
plt.legend()
```

Out[17]:
```
<matplotlib.legend.Legend at 0x207045c3f50>
```



Evaluate Clustering

SILHOUETTE SCORE : Measure how well the clusters are seperated

In [18]:
```
from sklearn.metrics import silhouette_score

silhouette_avg = silhouette_score(scaler_features ,labels)
print('silhouette_avg',silhouette_avg)
```

```
silhouette_avg 0.631640138719835
```

HOW CAN WE KNOW THE NUMBER OF CLUSTER ??

ELBOW METHOD

In [19]:
```
scaler = StandardScaler()
scaler_features = scaler.fit_transform(features)
```

In [20]:
```
# Apply the Elbow Method to determine the optimal number of cluster
inertia =[]
k_values = range(1,11)
# Check from 1 to 10 clusters

for k in k_values:
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(scaler_features)
    inertia.append(kmeans.inertia_)
```

```
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Acer\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: KMeans is known to have a mem
ory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the envi
ronment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [21]:
```
k_values
```

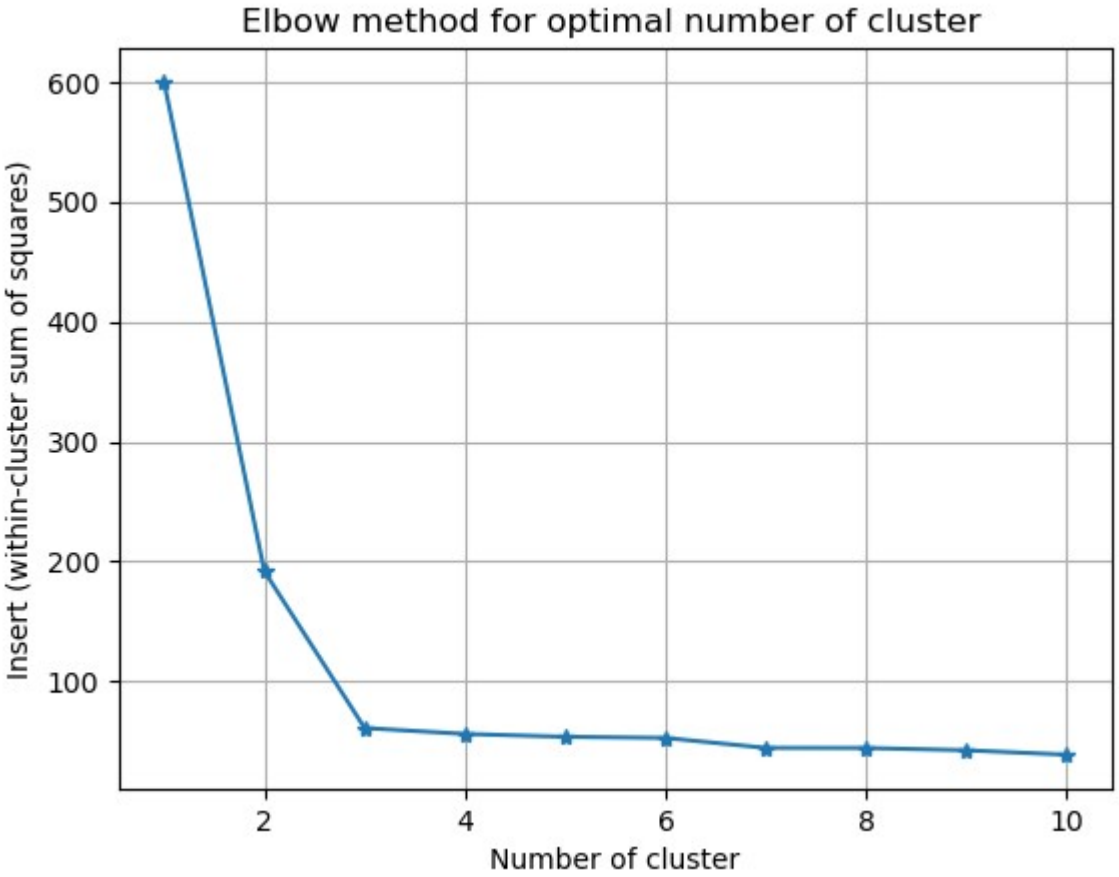
Out[21]:
```
range(1, 11)
```

In [22]:
```
inertia
```

Out[22]:
```
[600.0000000000002,
 191.78640738571227,
 60.95739675119526,
 56.04130161759119,
 53.64811048663857,
 52.69011529940316,
 44.3980876275986,
 44.31072952149389,
 42.41222559254021,
 38.72492395273803]
```

In [23]:
```
import matplotlib.pyplot as plt
# Plot the Elbow curve

plt.plot(k_values , inertia , marker ='*')
plt.xlabel("Number of cluster")
plt.ylabel("Insert (within-cluster sum of squares)")
plt.title("Elbow method for optimal number of cluster ")
plt.grid(True)
plt.show()
```



AGGLOMERATIVE HIERARCHICAL CLUSTERING

is a type of hierarchical clustering method used to group data points into clusters based on their similarity. It is called "agglomerative" because it starts with each data point as its own cluster and then iteratively merges the most similar clusters until a single cluster is formed or a stopping criterion is met. Here's an overview of the process and how dendrograms are used to visualize it: Agglomerative Hierarchical Clustering Process

INITIALIZATION: Begin with each data point as a separate cluster. If you have n data points, you start with n clusters.

COMPUTER DISTANCE: Calculate the distance (or similarity) between each pair of clusters. Common distance measures include Euclidean distance, Manhattan distance, and cosine similarity.

MERGE CLUSTERS: Find the pair of clusters with the smallest distance and merge them into a single cluster.

UPDATE DISTANCES: After merging, update the distance matrix to reflect the new distances between the merged cluster and the remaining clusters. Different methods can be used to calculate the distance between clusters, such as:

1-Single Linkage: Distance between the closest pair of points in the two clusters.

2-Complete Linkage: Distance between the farthest pair of points in the two clusters.

3-Average Linkage: Average distance between all pairs of points in the two clusters.

4-Ward's Method: Minimizes the total variance within clusters.

5-Repeat: Continue merging clusters until only one cluster remains or a desired number of clusters is achieved.

DENDROGRAM

A dendrogram is a tree-like diagram that illustrates the arrangement of the clusters produced by hierarchical clustering. It provides a visual representation of how clusters are merged at each step of the agglomerative process.

In [29]:
```
from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n_clusters=3 , metric='euclidean',linkage='ward')
data['Cluster'] = hc.fit_predict(scaler_features)
data
```

Out[29]:

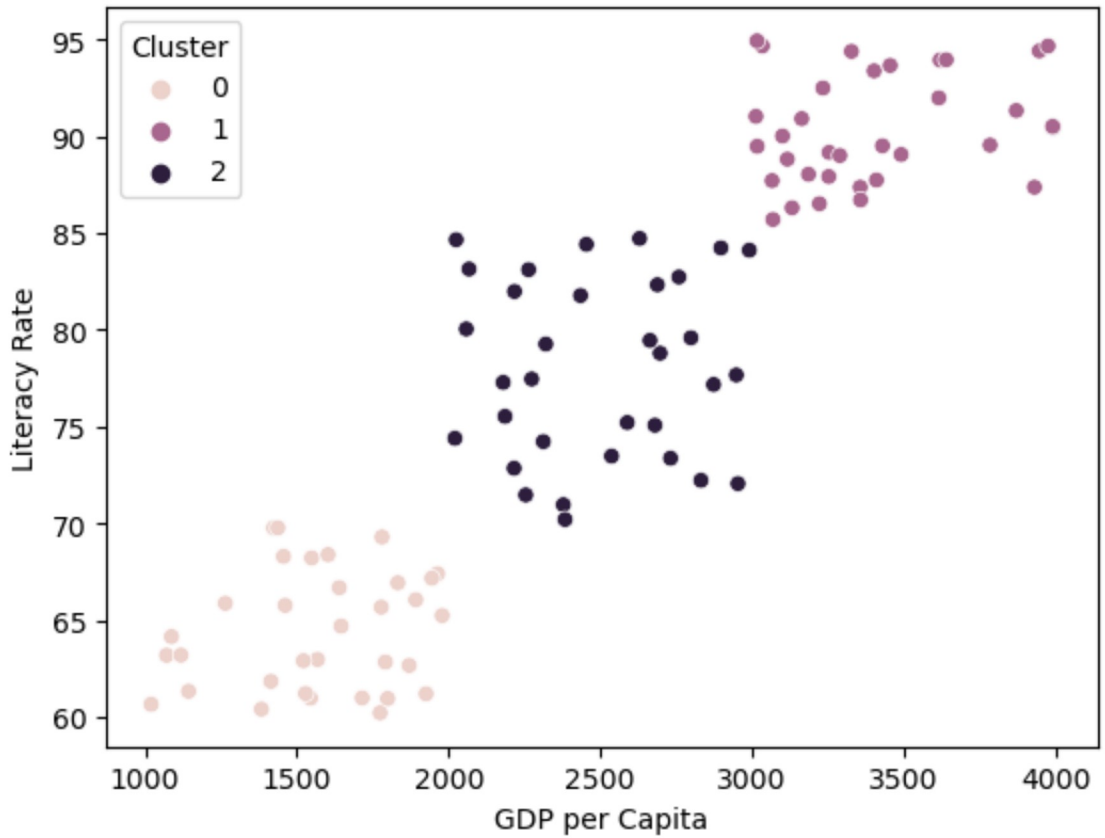
	GDP per Capita	Life Expectancy	Literacy Rate	Unemployment Rate	Access to Clean Water	Poverty Rate	Country	Cluster
0	1548.814	55.68434	68.20993	15.046950	61.35474	38.558030	Country 1	0
1	1715.189	50.18790	60.97101	21.778170	62.98282	30.117140	Country 2	0
2	1602.763	56.17635	68.37945	17.700080	65.69965	33.599780	Country 3	0
3	1544.883	56.12096	60.96098	22.351940	65.90873	37.299910	Country 4	0
4	1423.655	56.16934	69.76459	24.621890	65.74325	31.716300	Country 5	0
...	...	...	...	...	...	...	...	...
95	3490.305	78.81720	89.06733	9.610557	86.33461	14.979620	Country 96	1
96	3989.410	72.72437	90.52078	5.447473	94.80580	8.621891	Country 97	1
97	3065.304	73.79057	87.71653	7.029712	93.71786	9.706489	Country 98	1
98	3783.234	73.74296	89.55444	5.121566	90.02721	8.782452	Country 99	1
99	3288.398	77.48788	89.01714	6.713055	94.22348	14.795270	Country 100	1

100 rows × 8 columns

In [35]:
```
import seaborn as sns

sns.scatterplot(x ="GDP per Capita", y="Literacy Rate",data=data,hue='Cluster')
```

Out[35]:
```
<Axes: xlabel='GDP per Capita', ylabel='Literacy Rate'>
```



In [36]:
```
from scipy.cluster.hierarchy import dendrogram , linkage
import matplotlib.pyplot as plt

# Perform hierarchcal clustering
z= linkage(scaler_features , method='ward')
```

In [48]:
```
lab=data['Country'].tolist()
lab
# data['Country'] is in the series form so add tolist to make it to list form
```

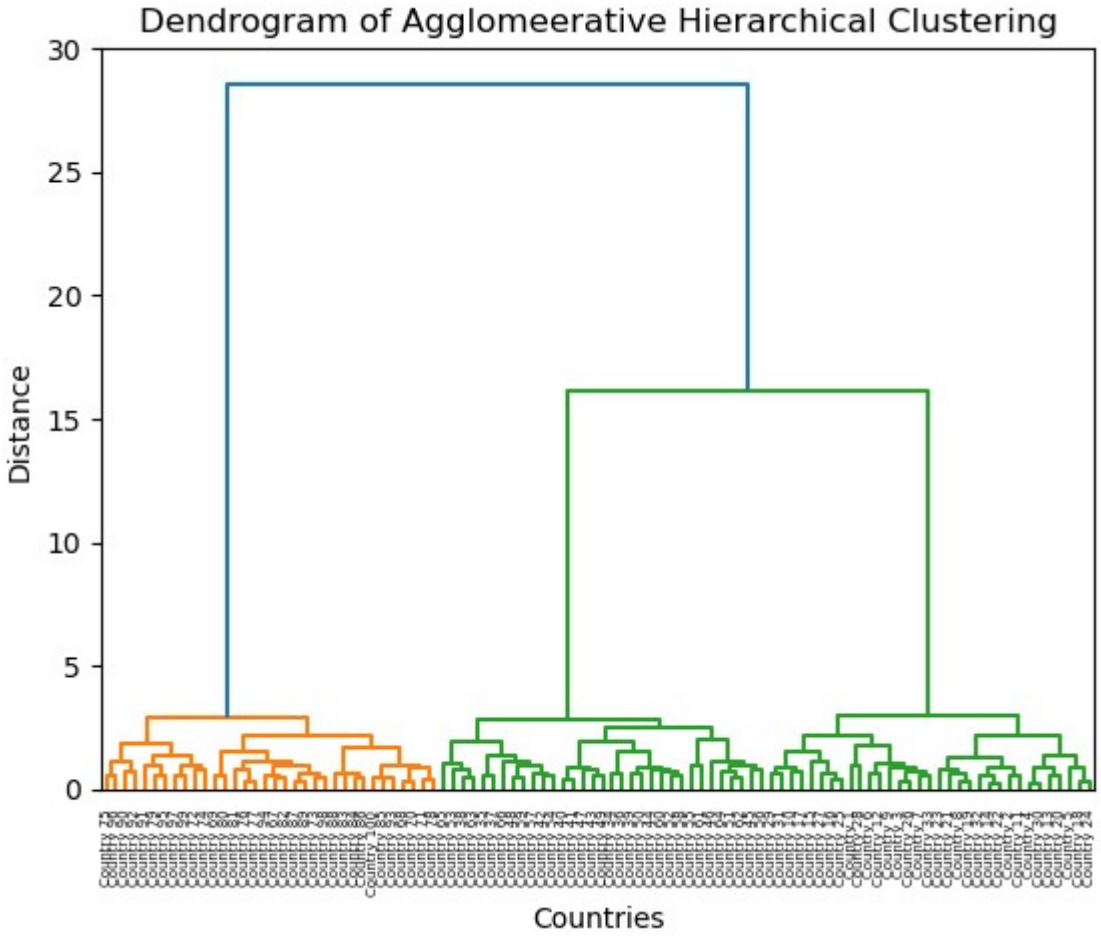
Out[48]:
```
['Country 1',
 'Country 2',
 'Country 3',
 'Country 4',
 'Country 5',
 'Country 6',
 'Country 7',
 'Country 8',
 'Country 9',
 'Country 10',
 'Country 11',
 'Country 12',
 'Country 13',
 'Country 14',
 'Country 15',
 'Country 16',
 'Country 17',
 'Country 18',
 'Country 19',
 'Country 20',
 'Country 21',
 'Country 22',
 'Country 23',
 'Country 24',
 'Country 25',
 'Country 26',
 'Country 27',
 'Country 28',
 'Country 29',
 'Country 30',
 'Country 31',
 'Country 32',
 'Country 33',
 'Country 34',
 'Country 35',
 'Country 36',
 'Country 37',
 'Country 38',
 'Country 39',
 'Country 40',
 'Country 41',
 'Country 42',
```

```
'Country 43',
'Country 44',
'Country 45',
'Country 46',
'Country 47',
'Country 48',
'Country 49',
'Country 50',
'Country 51',
'Country 52',
'Country 53',
'Country 54',
'Country 55',
'Country 56',
'Country 57',
'Country 58',
'Country 59',
'Country 60',
'Country 61',
'Country 62',
'Country 63',
'Country 64',
'Country 65',
'Country 66',
'Country 67',
'Country 68',
'Country 69',
'Country 70',
'Country 71',
'Country 72',
'Country 73',
'Country 74',
'Country 75',
'Country 76',
'Country 77',
'Country 78',
'Country 79',
'Country 80',
'Country 81',
'Country 82',
'Country 83',
'Country 84',
```

```
 'Country 85',
 'Country 86',
 'Country 87',
 'Country 88',
 'Country 89',
 'Country 90',
 'Country 91',
 'Country 92',
 'Country 93',
 'Country 94',
 'Country 95',
 'Country 96',
 'Country 97',
 'Country 98',
 'Country 99',
 'Country 100']
```

In [47]:
```
# Plot the dendrogram

dendrogram(z, labels=lab, leaf_rotation=90)
plt.title("Dendrogram of Agglomeerative Hierarchical Clustering")
plt.xlabel("Countries")
plt.ylabel("Distance")
plt.show()
```



CONCLUSION

In this analysis, we looked at how GDP per Capita, Life Expectancy, and Literacy Rate relate to each other across different countries. By using clustering methods like KMeans and Hierarchical Clustering, we found groups of countries that share similar economic and health characteristics. We verified the strength of these clusters using metrics like the silhouette score and the Elbow Method, which helped us decide how many clusters to use. The dendrogram also showed how closely related some countries are. Overall, this analysis provides useful insights that could help shape research and policies aimed at addressing global socioeconomic issues.