In [1]:

```
import pandas as pd

master_df = pd.read_csv("Country-data.csv")
data_df = master_df.set_index("country").copy()
data_df.head()
```

Out[1]:

country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	g
Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	
Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4
Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4
Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3
Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12

Understanding the structure of the dataframe

In [2]:

```
## Checking if there are any NULL or NAN values using the info() function
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 167 entries, Afghanistan to Zambia
Data columns (total 9 columns):
child_mort    167 non-null float64
exports       167 non-null float64
health        167 non-null float64
imports       167 non-null float64
income        167 non-null int64
inflation     167 non-null float64
life_expec    167 non-null float64
total_fer     167 non-null float64
gdpp          167 non-null int64
dtypes: float64(7), int64(2)
memory usage: 13.0+ KB
```

There are no columns with the missing values (NULL or NAN) as described above

In [3]:

```
## check the description of the data, and see the types of data present
data_df.describe()
```

Out[3]:

	child_mort	exports	health	imports	income	inflation	life_expe
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.00000
mean	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.55568
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.89317
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.10000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.30000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.10000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.80000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.80000

Seems all the data is numeric (except country names , which is ok). No need to include the dummy variables as there is no conversion is required.

However the range of the values are very diverse - definitely require standardization or normalization of the data, which will be done after checking the outliers

Checking the outliers

In [4]:

```
## Check the outliers at 25%,50%,75%,90%,95% and 99%
data_df.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

Out[4]:

	child_mort	exports	health	imports	income	inflation	life_expe
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.00000
mean	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.55568
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.89317
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.10000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.30000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.10000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.80000
90%	100.220000	70.800000	10.940000	75.420000	41220.000000	16.640000	80.40000
95%	116.000000	80.570000	11.570000	81.140000	48290.000000	20.870000	81.40000
99%	153.400000	160.480000	13.474000	146.080000	84374.000000	41.478000	82.37000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.80000

From the distribution shown above, you can see that there no outliner in your data. The numbers are gradually increasing.

Feature Standardization

In [5]:

```
## Normalizing all the columns with continuous values and round them to nearest 1 decim
al value
normalized_df= round(((data_df-data_df.mean())/data_df.std()),1)
normalized_df.describe()
```

Out[5]:

	child_mort	exports	health	imports	income	inflation	life_expec
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000
mean	-0.002994	-0.000599	0.001198	-0.001198	0.001796	-0.001198	0.002395
std	1.000146	1.000572	0.998733	0.999216	1.002796	0.998794	0.999816
min	-0.900000	-1.500000	-1.800000	-1.900000	-0.900000	-1.100000	-4.300000
25%	-0.750000	-0.600000	-0.700000	-0.700000	-0.700000	-0.600000	-0.600000
50%	-0.500000	-0.200000	-0.200000	-0.100000	-0.400000	-0.200000	0.300000
75%	0.600000	0.400000	0.650000	0.500000	0.300000	0.300000	0.700000
max	4.200000	5.800000	4.000000	5.300000	5.600000	9.100000	1.400000

In [6]:

```
normalized_df.head()
```

Out[6]:

country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gd
Afghanistan	1.3	-1.1	0.3	-0.1	-0.8	0.2	-1.6	1.9	-
Albania	-0.5	-0.5	-0.1	0.1	-0.4	-0.3	0.6	-0.9	-
Algeria	-0.3	-0.1	-1.0	-0.6	-0.2	0.8	0.7	-0.0	-
Angola	2.0	0.8	-1.4	-0.2	-0.6	1.4	-1.2	2.1	-
Antigua and Barbuda	-0.7	0.2	-0.3	0.5	0.1	-0.6	0.7	-0.5	-

Check the Corelation between the features before performing PCA

In [7]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
corr_df = normalized_df.reset_index()
corr_df_visual = corr_df.drop(columns = "country")
corrmat = np.corrcoef(corr_df_visual)
plt.figure(figsize=[15,15])
sns.heatmap(corrmat)
plt.show()
```

```
<Figure size 1500x1500 with 2 Axes>
```

In [8]:

```
corr_df_visual.head()
```

Out[8]:

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	1.3	-1.1	0.3	-0.1	-0.8	0.2	-1.6	1.9	-0.7
1	-0.5	-0.5	-0.1	0.1	-0.4	-0.3	0.6	-0.9	-0.5
2	-0.3	-0.1	-1.0	-0.6	-0.2	0.8	0.7	-0.0	-0.5
3	2.0	0.8	-1.4	-0.2	-0.6	1.4	-1.2	2.1	-0.5
4	-0.7	0.2	-0.3	0.5	0.1	-0.6	0.7	-0.5	-0.0

As seen above, the data is normalized. The max values are within the range of 1 to 10

PCA on the normalized data

In [9]:

```
normalized_df.shape
```

Out[9]:

```
(167, 9)
```

In [10]:

```
from sklearn.decomposition import PCA
pca = PCA(random_state=42)
pca.fit(normalized_df)
```

Out[10]:

```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=42,
    svd_solver='auto', tol=0.0, whiten=False)
```

Components of PCA

In [11]:

```
pca.components_
```

Out[11]:

```
array([[-4.19025219e-01,  2.84110724e-01,  1.52225652e-01,
         1.59666368e-01,  3.98531407e-01, -1.94223694e-01,
         4.25557504e-01, -4.03167524e-01,  3.93496630e-01],
       [ 1.88744400e-01,  6.14184622e-01, -2.48148074e-01,
         6.71254207e-01,  2.36944206e-02, -4.08472256e-03,
        -2.20526004e-01,  1.52580871e-01, -5.00237222e-02],
       [-2.44187029e-02,  1.40425121e-01, -5.87946196e-01,
        -3.05044484e-01,  3.10347581e-01,  6.44095376e-01,
         1.10349895e-01,  2.21666347e-02,  1.30800279e-01],
       [ 3.76365234e-01,  1.86122577e-03,  4.68885648e-01,
        -6.14968974e-02,  3.90592763e-01,  1.39912744e-01,
        -2.06066741e-01,  3.77945277e-01,  5.26560263e-01],
       [-1.66559279e-01,  5.50340764e-02,  5.17935818e-01,
         2.55234120e-01, -2.45934160e-01,  7.15798262e-01,
         1.11250469e-01, -1.36490168e-01, -1.78802389e-01],
       [ 1.96076449e-01, -5.25393433e-02,  9.42019662e-03,
        -3.51305654e-02,  1.51123141e-01,  6.33574967e-02,
        -6.05237094e-01, -7.50917100e-01,  2.07349930e-02],
       [-1.46425178e-02, -7.01680543e-01, -2.58550102e-01,
         5.91014454e-01,  5.72385441e-02,  1.03465032e-01,
         6.09860955e-02, -7.03999107e-03,  2.71170875e-01],
       [-6.75969500e-01, -8.86299510e-02,  5.50338988e-02,
         3.22396664e-02,  3.80683924e-01, -4.67755417e-04,
        -4.89560655e-01,  2.93239720e-01, -2.46093890e-01],
       [ 3.51011709e-01, -1.27811124e-01,  1.07201998e-01,
         1.01854506e-01,  6.00178847e-01, -2.71935279e-02,
         3.05860991e-01, -4.79153496e-02, -6.17889574e-01]])
```

In [12]:

```
pca.explained_variance_ratio_
```

Out[12]:

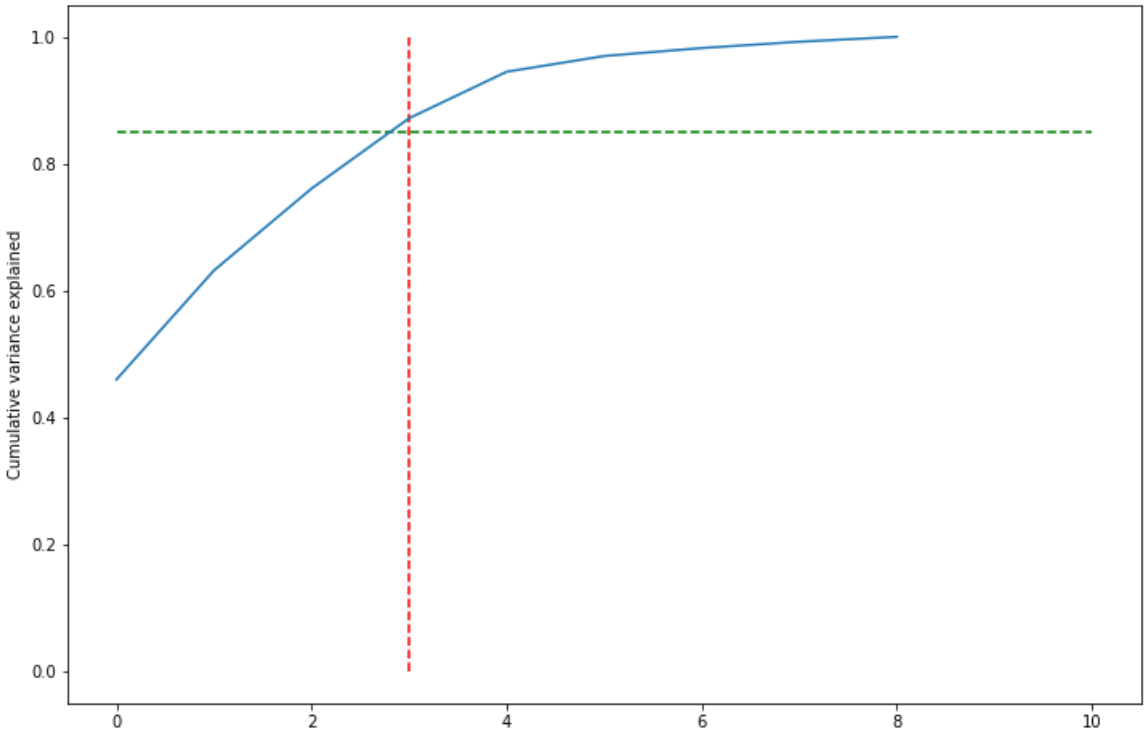
```
array([0.45992541, 0.17198798, 0.12899513, 0.11078812, 0.07330956,
       0.02476818, 0.01257505, 0.00995536, 0.00769522])
```

In [13]:

```
var_cumu = np.cumsum(pca.explained_variance_ratio_)
```

In [14]:

```
fig = plt.figure(figsize=[12,8])
plt.vlines(x=3, ymax=1, ymin=0, colors="r", linestyles="--")
plt.hlines(y=0.85, xmax=10, xmin=0, colors="g", linestyles="--")
plt.plot(var_cumu)
plt.ylabel("Cumulative variance explained")
plt.show()
```



As can be seen - 4 PCA components are enough to explain the 95% of the data Now to perform an Incremental PCA with just 4 components

In [15]:

```
from sklearn.decomposition import IncrementalPCA
pca_final = IncrementalPCA(n_components=3)
df_train_pca = pca_final.fit_transform(normalized_df)
df_train_pca.shape
```

Out[15]:

```
(167, 3)
```

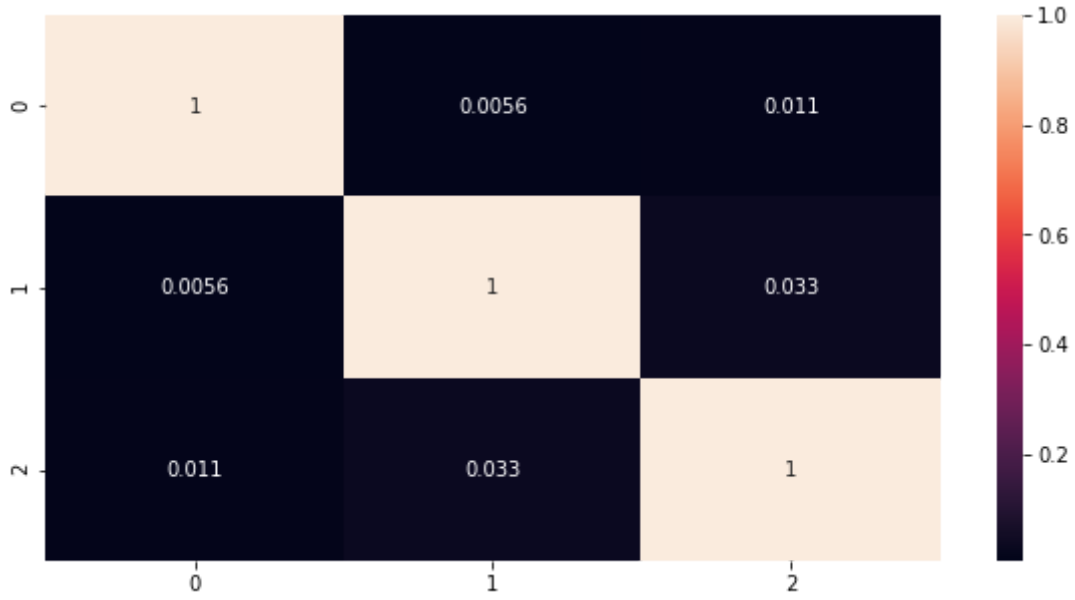
Plotting the correlation Heatmap to see that the selected 4 PCs have very less or no corelation|

In [16]:

```
corrmat = np.corrcoef(df_train_pca.transpose())
plt.figure(figsize=[10,5])
sns.heatmap(corrmat, annot=True)
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x189b5a31908>
```



In [17]:

```
data_pca = pd.DataFrame(df_train_pca)
data_pca['Country'] = master_df["country"]
data_pca = data_pca[['Country', 0, 1, 2]]
data_pca_df = data_pca.set_index('Country')
data_pca_df.head()
```

Out[17]:

	0	1	2
Country			
Afghanistan	-2.912245	-0.002626	1.156287
Albania	0.385131	-0.826425	-0.567307
Algeria	-0.268748	-0.115479	-1.579527
Angola	-2.917331	2.186826	-0.325058
Antigua and Barbuda	1.032632	-0.018046	-0.310583

K-means Clustering

In [18]:

```
# k-means with some arbitrary k
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

kmeans = KMeans(n_clusters=4, max_iter=50)
kmeans.fit(data_pca_df)
```

Out[18]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [19]:

```
kmeans.labels_
```

Out[19]:

```
array([2, 1, 1, 2, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 2, 1, 1, 0, 2,
       1, 0, 1, 2, 2, 1, 2, 0, 1, 2, 2, 1, 1, 1, 2, 2, 2, 0, 2, 0, 0, 0,
       0, 1, 1, 1, 1, 2, 2, 0, 1, 0, 0, 1, 2, 1, 0, 2, 0, 1, 1, 2, 2, 1,
       2, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 2, 2, 0, 1, 2, 0, 0, 2,
       2, 1, 0, 3, 1, 2, 2, 1, 1, 2, 3, 2, 1, 2, 0, 1, 0, 1, 2, 1, 2, 1,
       0, 0, 2, 2, 0, 1, 2, 0, 1, 1, 1, 0, 0, 0, 1, 1, 2, 1, 1, 2, 0, 0,
       2, 3, 0, 0, 2, 2, 0, 0, 1, 1, 2, 1, 0, 0, 1, 2, 1, 2, 2, 1, 1, 1,
       1, 2, 1, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2])
```

Finding the optimal number of clusters SSD

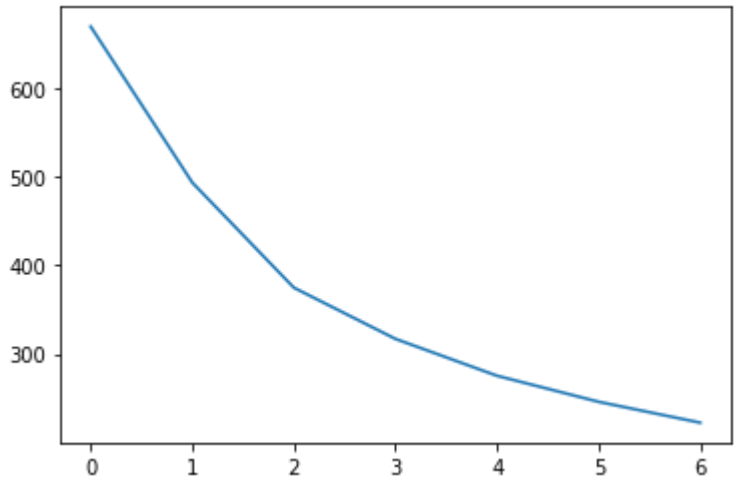
In [20]:

```
# elbow-curve/SSD and Silhoutte analysis
ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(data_pca_df)

    ssd.append(kmeans.inertia_)

    cluster_labels = kmeans.labels_

    # silhouette score
    silhouette_avg = silhouette_score(data_pca_df, cluster_labels)
    print("For n_clusters={0}, the silhouette score is {1}".format(num_clusters, silhou
ette_avg))

# plot the SSDs for each n_clusters
# ssd
plt.plot(ssd)
```

```
For n_clusters=2, the silhouette score is 0.36817626729893277
For n_clusters=3, the silhouette score is 0.33469489743096464
For n_clusters=4, the silhouette score is 0.3340724890564839
For n_clusters=5, the silhouette score is 0.32652408728128773
For n_clusters=6, the silhouette score is 0.29997014222753915
For n_clusters=7, the silhouette score is 0.27163826957698955
For n_clusters=8, the silhouette score is 0.2779654614961132
```

Out[20]:

```
[<matplotlib.lines.Line2D at 0x189b4381550>]
```



In [21]:

```
# final model with k=3
kmeans = KMeans(n_clusters=3, max_iter=50)
kmeans.fit(data_pca_df)
```

Out[21]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [22]:

```
kmeans.labels_
```

Out[22]:

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

In [23]:

```
# assign the label
data_pca_df['cluster_id'] = kmeans.labels_
data_pca_df.columns = ['PC1', 'PC2', 'PC3', 'cluster_id']
data_pca_df.head()
```

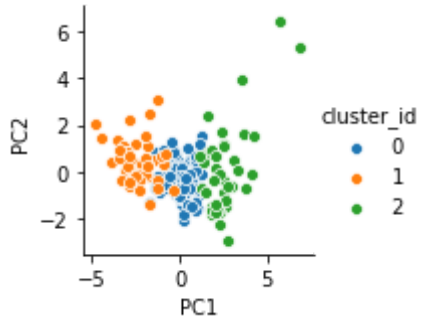
Out[23]:

Country	PC1	PC2	PC3	cluster_id
Afghanistan	-2.912245	-0.002626	1.156287	1
Albania	0.385131	-0.826425	-0.567307	0
Algeria	-0.268748	-0.115479	-1.579527	0
Angola	-2.917331	2.186826	-0.325058	1
Antigua and Barbuda	1.032632	-0.018046	-0.310583	0

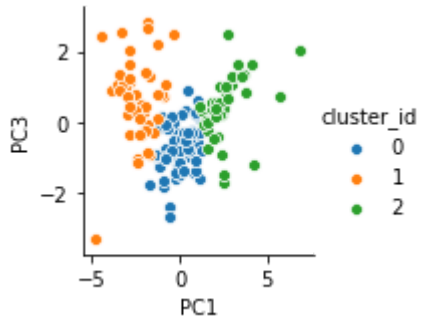
In [24]:

```
plt.figure(figsize=[10,10])
sns.pairplot(data=data_pca_df, x_vars=["PC1"], y_vars=["PC2"], hue = "cluster_id")
plt.show()

plt.figure(figsize=[10,10])
sns.pairplot(data=data_pca_df, x_vars=["PC1"], y_vars=["PC3"], hue = "cluster_id")
plt.show()

plt.figure(figsize=[10,10])
sns.pairplot(data=data_pca_df, x_vars=["PC2"], y_vars=["PC3"], hue = "cluster_id")
plt.show()
```

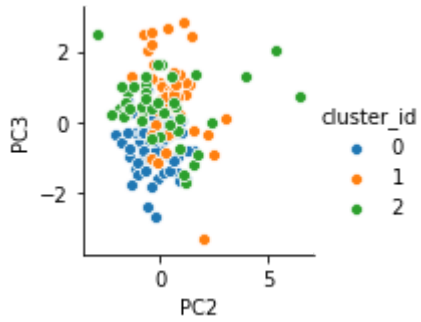
```
<Figure size 720x720 with 0 Axes>
```




```
<Figure size 720x720 with 0 Axes>
```




```
<Figure size 720x720 with 0 Axes>
```



Adding the clusters and the selective three columns (gdpp, child_mort and income) from the original data to a new DF for the purpose of analyzing the data

In [25]:

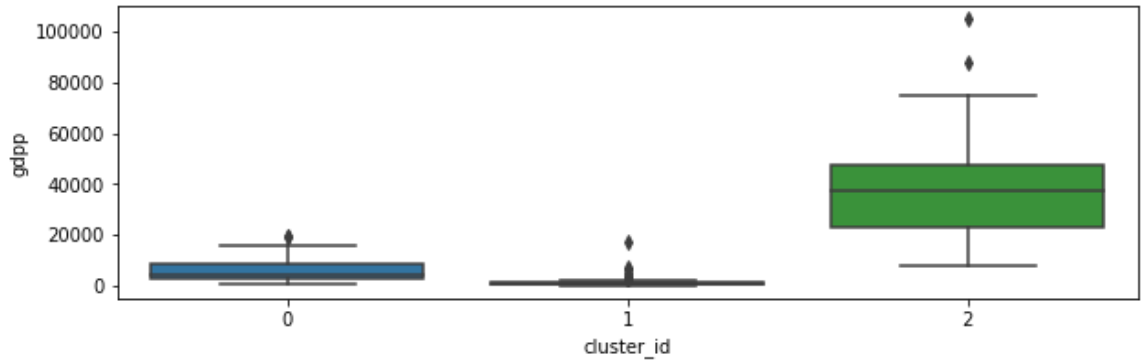
```
mixed_df_k = pd.DataFrame()
mixed_df_k['country'] = master_df['country']
mixed_df_k['income'] = list(master_df['income'])
mixed_df_k['gdpp'] = list(master_df['gdpp'])
mixed_df_k['child_mort'] = list(master_df['child_mort'])
mixed_df_k['cluster_id'] = kmeans.labels_
mixed_df_k.head()
```

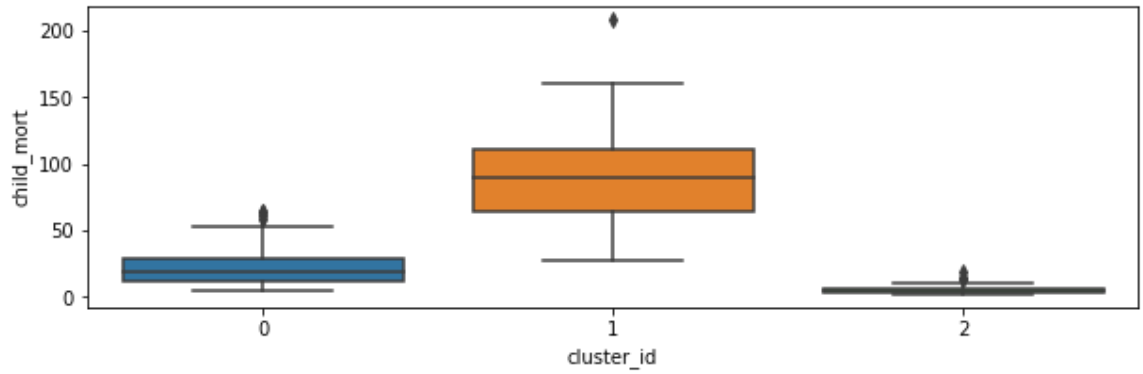
Out[25]:

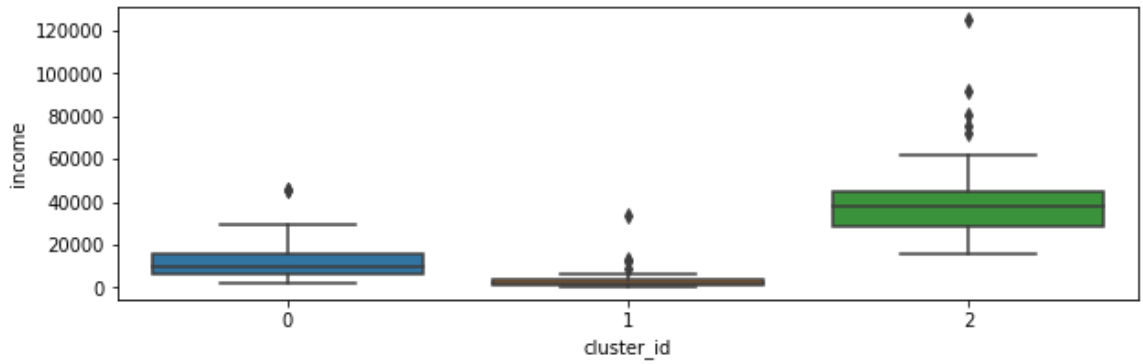
	country	income	gdpp	child_mort	cluster_id
0	Afghanistan	1610	553	90.2	1
1	Albania	9930	4090	16.6	0
2	Algeria	12900	4460	27.3	0
3	Angola	5900	3530	119.0	1
4	Antigua and Barbuda	19100	12200	10.3	0

In [26]:

```
plt.figure(figsize=[10,10])
plt.subplot(3,1,1)
sns.boxplot(x='cluster_id', y='gdpp', data=mixed_df_k)
plt.show()

plt.figure(figsize=[10,10])
plt.subplot(3,1,2)
sns.boxplot(x='cluster_id', y='child_mort', data=mixed_df_k)
plt.show()

plt.figure(figsize=[10,10])
plt.subplot(3,1,3)
sns.boxplot(x='cluster_id', y='income', data=mixed_df_k)
plt.show()
```







From K clustering, Finding the top 5 countries that need the aid with the below criteria:

- low gdp and low incomes
- high child mortality rate

As per the visualizations above, cluster '0' meets this criteria very closely

In [37]:

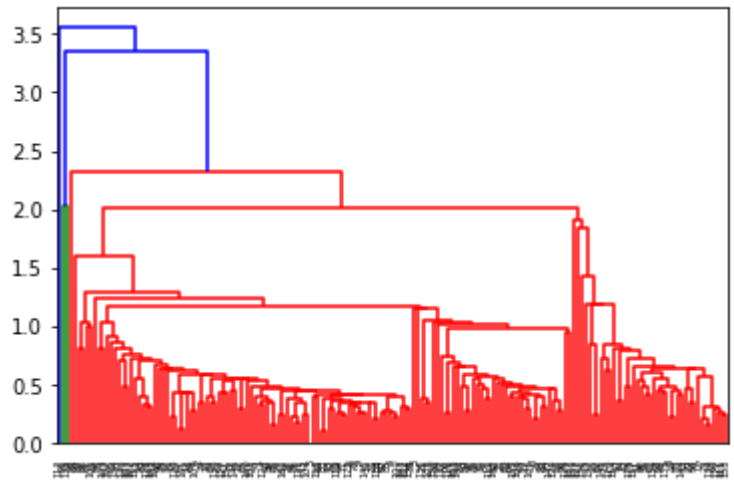
```
df_cluster_zero_k = mixed_df_k[mixed_df_k['cluster_id'] == 1]

income_sorted_df = df_cluster_zero_k.sort_values(by = 'income')
income_last_20 = income_sorted_df.head(20)
## list of 20 countries with least income
income_set = set(list(income_last_20['country']))

gdp_sorted_df = df_cluster_zero_k.sort_values(by = 'gdpp')
gdp_last_20 = gdp_sorted_df.head(20)
## list of 20 countries with least income
gdp_set = set(list(gdp_last_20['country']))


childmort_sorted_df = df_cluster_zero_k.sort_values(by = 'child_mort', ascending = Fals
e)
childmort_top_20 = childmort_sorted_df.head(20)
## list of 20 countries with least income
childmort_set = set(list(childmort_top_20['country']))

## finding the common countries in all three sets income_set, gdp_set and childmort_set
income_gdp_set = income_set.intersection(gdp_set)
top_n_countries_k = income_gdp_set.intersection(childmort_set)

print("The top countries that required aid are: ")
print(list(top_n_countries_k))
```

```
The top countries that required aid are:
['Congo, Dem. Rep.', 'Niger', 'Guinea-Bissau', 'Burkina Faso', 'Guinea',
'Burundi', 'Central African Republic', 'Sierra Leone', 'Mozambique', 'Hait
i']
```

Hirarchical Clustering

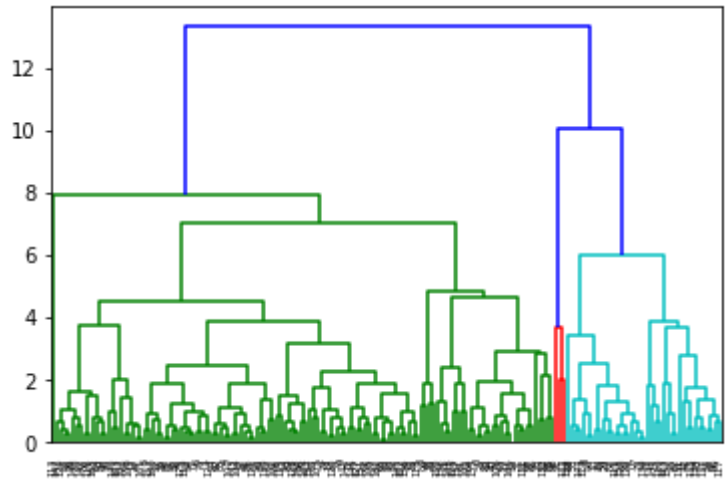
In [28]:

```
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree

# single linkage
mergings = linkage(data_pca_df, method="single", metric='euclidean')
dendrogram(mergings)
plt.show()
```



In [29]:

```
# Complete Linkage

mergings = linkage(data_pca_df, method="complete", metric='euclidean')
dendrogram(mergings)
plt.show()
```



In [30]:

```
# 3 clusters
cluster_labels = cut_tree(mergings, n_clusters=3).reshape(-1, )
cluster_labels
```

Out[30]:

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

In [31]:

```
# assign the label
data_pca_df['cluster_id'] = cluster_labels
data_pca_df.head()
```

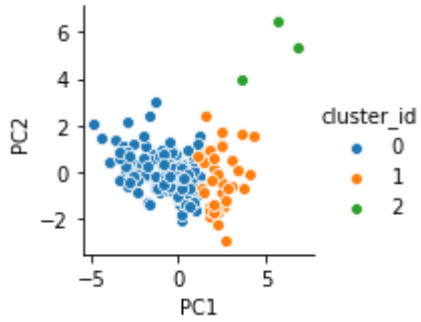
Out[31]:

Country	PC1	PC2	PC3	cluster_id
Afghanistan	-2.912245	-0.002626	1.156287	0
Albania	0.385131	-0.826425	-0.567307	0
Algeria	-0.268748	-0.115479	-1.579527	0
Angola	-2.917331	2.186826	-0.325058	0
Antigua and Barbuda	1.032632	-0.018046	-0.310583	0

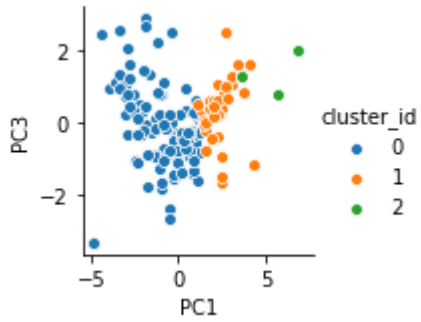
In [32]:

```
plt.figure(figsize=[10,10])
sns.pairplot(data=data_pca_df, x_vars=["PC1"], y_vars=["PC2"], hue = "cluster_id")
plt.show()

plt.figure(figsize=[10,10])
sns.pairplot(data=data_pca_df, x_vars=["PC1"], y_vars=["PC3"], hue = "cluster_id")
plt.show()

plt.figure(figsize=[10,10])
sns.pairplot(data=data_pca_df, x_vars=["PC2"], y_vars=["PC3"], hue = "cluster_id")
plt.show()
```

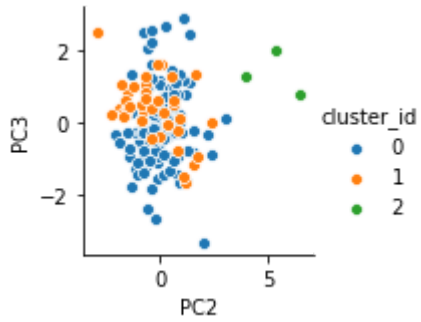
```
<Figure size 720x720 with 0 Axes>
```




```
<Figure size 720x720 with 0 Axes>
```




```
<Figure size 720x720 with 0 Axes>
```



In [33]:

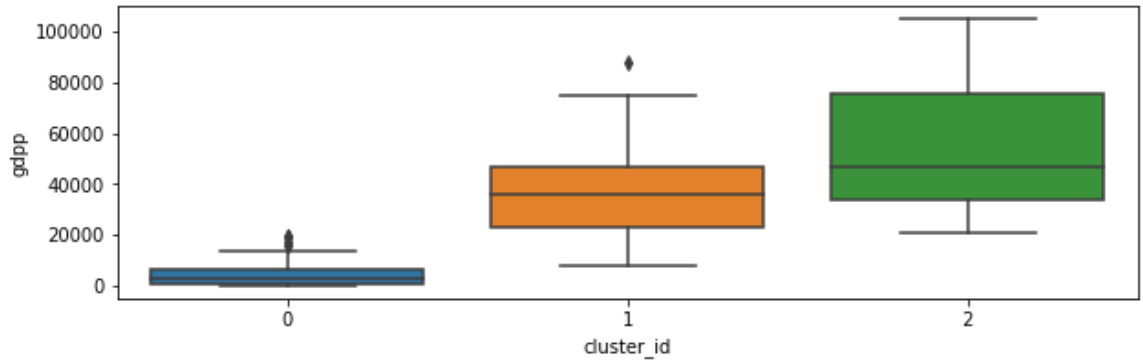
```
mixed_df_h = pd.DataFrame()
mixed_df_h['country'] = master_df['country']
mixed_df_h['income'] = list(master_df['income'])
mixed_df_h['gdpp'] = list(master_df['gdpp'])
mixed_df_h['child_mort'] = list(master_df['child_mort'])
mixed_df_h['cluster_id'] = cluster_labels
mixed_df_h.head()
```

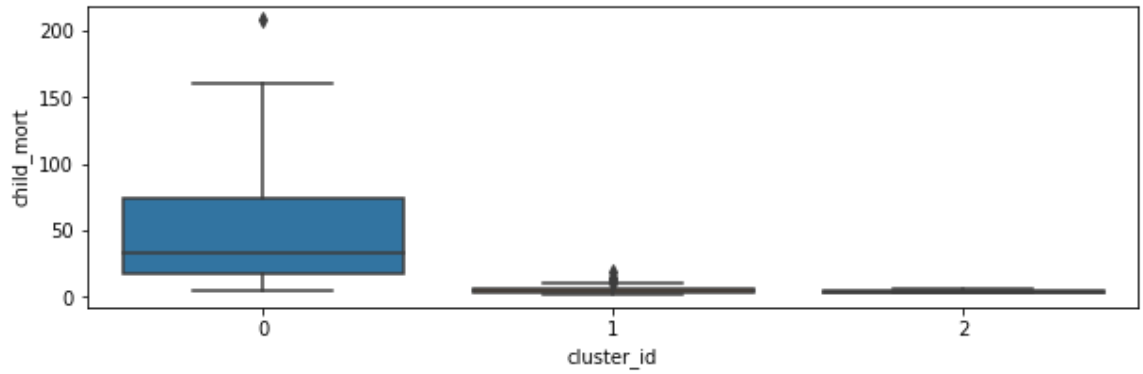
Out[33]:

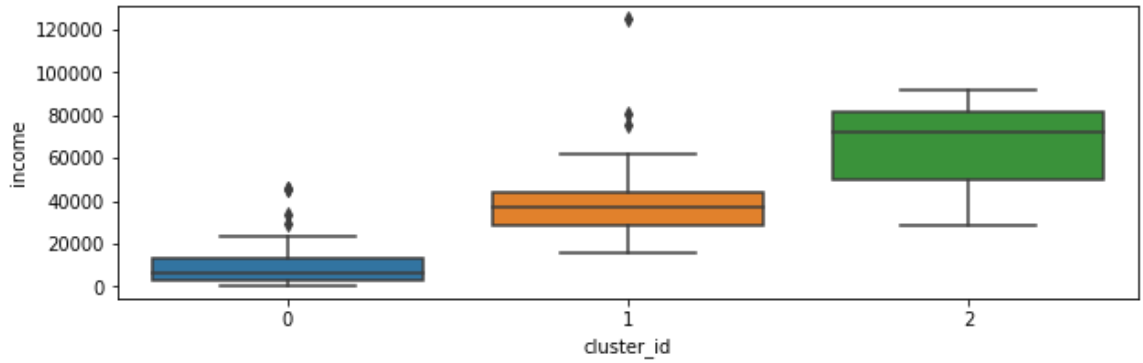
	country	income	gdpp	child_mort	cluster_id
0	Afghanistan	1610	553	90.2	0
1	Albania	9930	4090	16.6	0
2	Algeria	12900	4460	27.3	0
3	Angola	5900	3530	119.0	0
4	Antigua and Barbuda	19100	12200	10.3	0

In [34]:

```
plt.figure(figsize=[10,10])
plt.subplot(3,1,1)
sns.boxplot(x='cluster_id', y='gdpp', data=mixed_df_h)
plt.show()

plt.figure(figsize=[10,10])
plt.subplot(3,1,2)
sns.boxplot(x='cluster_id', y='child_mort', data=mixed_df_h)
plt.show()

plt.figure(figsize=[10,10])
plt.subplot(3,1,3)
sns.boxplot(x='cluster_id', y='income', data=mixed_df_h)
plt.show()
```







From Hirarchical clustering, Finding the top 5 countries that need the aid with the below criteria:

- low gdp and low incomes
- high child mortality rate

As per the visualizations above, cluster '0' meets this criteria very closely

In [35]:

```
df_cluster_zero_h = mixed_df_h[mixed_df_h['cluster_id'] == 0]

income_sorted_df = df_cluster_zero_h.sort_values(by = 'income')
income_last_20 = income_sorted_df.head(20)
## list of 20 countries with least income
income_set = set(list(income_last_20['country']))

gdp_sorted_df = df_cluster_zero_h.sort_values(by = 'gdpp')
gdp_last_20 = gdp_sorted_df.head(20)
## list of 20 countries with least income
gdp_set = set(list(gdp_last_20['country']))


childmort_sorted_df = df_cluster_zero_h.sort_values(by = 'child_mort', ascending = Fals
e)
childmort_top_20 = childmort_sorted_df.head(20)
## list of 20 countries with least income
childmort_set = set(list(childmort_top_20['country']))

## finding the common countries in all three sets income_set, gdp_set and childmort_set
income_gdp_set = income_set.intersection(gdp_set)
top_n_countries_h = income_gdp_set.intersection(childmort_set)

print("The top countries that required aid are: ")
print(list(top_n_countries_h))
```

```
The top countries that required aid are:
['Congo, Dem. Rep.', 'Niger', 'Guinea-Bissau', 'Burkina Faso', 'Guinea',
'Burundi', 'Central African Republic', 'Sierra Leone', 'Mozambique', 'Hait
i']
```

Let us see if there are common countries filteres between k-clusters and Hirarchical clusters

In [39]:

```
common_k_h_country_list = top_n_countries_h.intersection(top_n_countries_k)
common_k_h_country_list
```

Out[39]:

```
{'Burkina Faso',
 'Burundi',
 'Central African Republic',
 'Congo, Dem. Rep.',
 'Guinea',
 'Guinea-Bissau',
 'Haiti',
 'Mozambique',
 'Niger',
 'Sierra Leone'}
```

In [ ]:

```

```