Clustering and PCA Assignment

==========

PART-1

Importing and taking a glimpse at the data

---------- Importing Data ----------

In [1]:

```
import pandas as pd
```

In [2]:

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

Out[2]:

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gd
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	5
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4(
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4-
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3!
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12

In [3]:

```
df.shape
```

Out[3]:

```
(167, 10)
```

In [4]:

```
df.describe()
```

Out[4]:

	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	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.555689
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.893172
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.100000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.300000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.100000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.800000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.800000

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 10 columns):
country       167 non-null object
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), object(1)
memory usage: 13.1+ KB
```

there are no null values

Note -

- While computng the principal components, we must not include the entire dataset. Model building is all about doing well on the data we haven't seen yet!
- So we'll calculate the PCs using the train data, and apply them later on the test data

In [85]:

```
df.drop_duplicates(inplace=True)
```

In [6]:

```
#from sklearn.model_selection import train_test_split
cols = df.columns
list1 = ['child_mort', 'exports', 'health', 'imports', 'income','inflation', 'life_expec',
```

In [7]:

```
#X_train,y_train, X_test,y_test = train_test_split(df, test_size = 0.2)
df_parameters = df[list1]
```

hecking outliers

In [8]:

```
df_parameters.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

Out[8]:

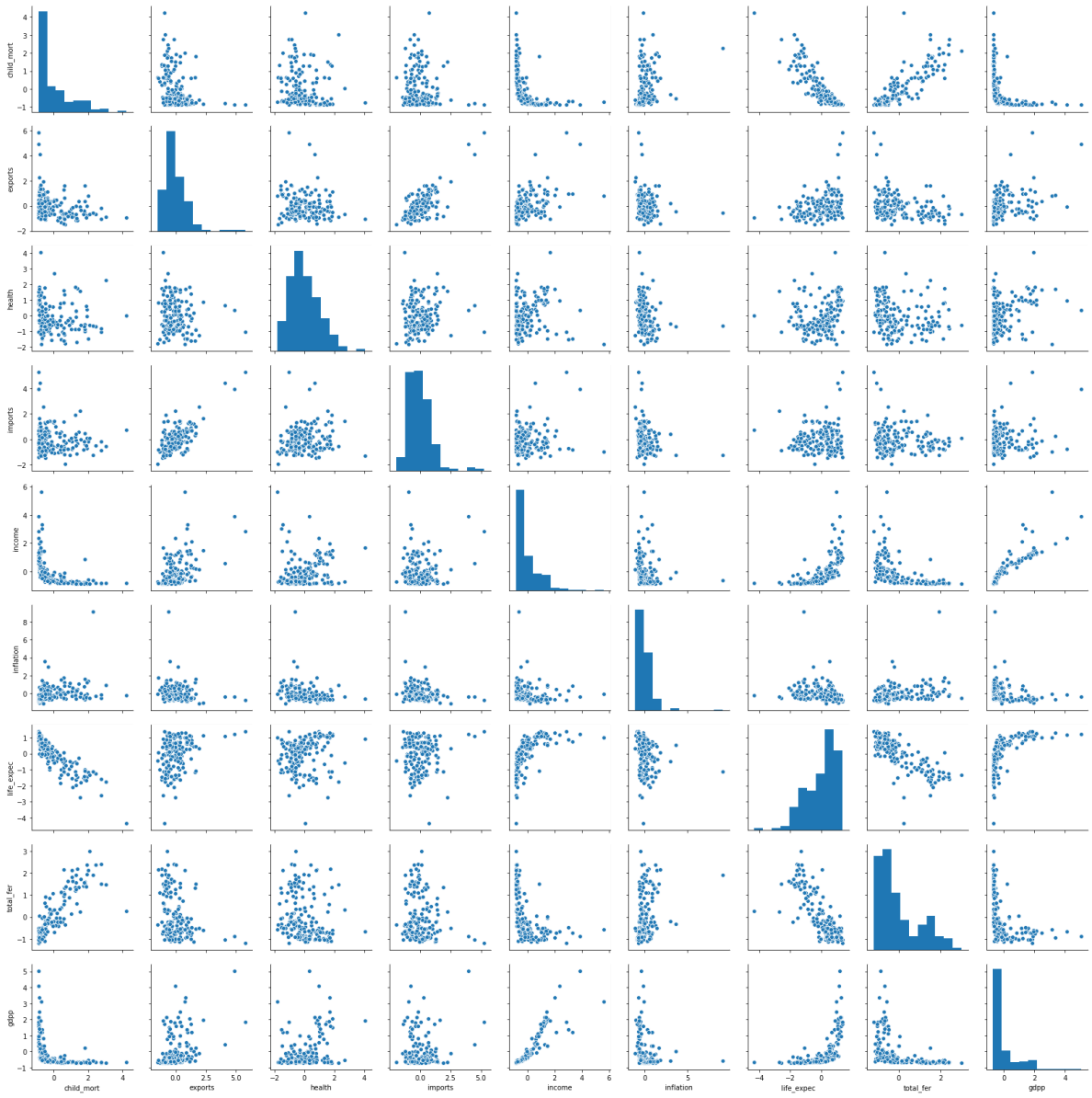
	child_mort	exports	health	imports	income	inflation	life_expec
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000
mean	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.555689
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.893172
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.100000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.300000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.100000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.800000
90%	100.220000	70.800000	10.940000	75.420000	41220.000000	16.640000	80.400000
95%	116.000000	80.570000	11.570000	81.140000	48290.000000	20.870000	81.400000
99%	153.400000	160.480000	13.474000	146.080000	84374.000000	41.478000	82.370000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.800000

In [89]:

```
sns.pairplot(df_parameters)
```

Out[89]:

```
<seaborn.axisgrid.PairGrid at 0x238b88282b0>
```



The above is a pair plot of the original data so as to get some visualisation

PART-2

Using PCA in order to reduce the dimensions

---------- Applying PCA ----------

Scaling the data

In [9]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data = scaler.fit_transform(df_parameters)
```

In [10]:

```
df_parameters = pd.DataFrame(data)
df_parameters.columns = list1
df_parameters.head()
```

Out[10]:

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	
0	1.291532	-1.138280	0.279088	-0.082455	-0.808245	0.157336	-1.619092	1.902882	-0.67
1	-0.538949	-0.479658	-0.097016	0.070837	-0.375369	-0.312347	0.647866	-0.859973	-0.48
2	-0.272833	-0.099122	-0.966073	-0.641762	-0.220844	0.789274	0.670423	-0.038404	-0.46
3	2.007808	0.775381	-1.448071	-0.165315	-0.585043	1.387054	-1.179234	2.128151	-0.51
4	-0.695634	0.160668	-0.286894	0.497568	0.101732	-0.601749	0.704258	-0.541946	-0.04

In [11]:

```
df_parameters.describe()
```

Out[11]:

	child_mort	exports	health	imports	income	inflation
count	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02
mean	-3.722904e-17	2.127373e-16	5.504579e-16	2.765585e-16	-7.977650e-17	1.063687e-16
std	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00	1.003008e+00
min	-8.871383e-01	-1.500192e+00	-1.827827e+00	-1.939940e+00	-8.603259e-01	-1.137852e+00
25%	-7.466190e-01	-6.333367e-01	-6.922106e-01	-6.914785e-01	-7.174558e-01	-5.666409e-01
50%	-4.717981e-01	-2.235279e-01	-1.810007e-01	-1.487432e-01	-3.738080e-01	-2.269504e-01
75%	5.926666e-01	3.747198e-01	6.515412e-01	4.913530e-01	2.942370e-01	2.816364e-01
max	4.221297e+00	5.813835e+00	4.047436e+00	5.266181e+00	5.611542e+00	9.129718e+00

Now applying PCA on the scaled variables

In [12]:

```
#Improting the PCA module
from sklearn.decomposition import PCA
pca = PCA(svd_solver='randomized', random_state=42)
```

In [13]:

```
pca.fit(df_parameters)
```

Out[13]:

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

In [14]:

```
pca.components_
```

Out[14]:

```
array([[-0.41951945,  0.28389698,  0.15083782,  0.16148244,  0.39844111,
        -0.19317293,  0.42583938, -0.40372896,  0.39264482],
       [ 0.19288394,  0.61316349, -0.24308678,  0.67182064,  0.02253553,
         -0.00840447, -0.22270674,  0.15523311, -0.0460224 ],
       [-0.02954353,  0.14476069, -0.59663237, -0.29992674,  0.3015475 ,
          0.64251951,  0.11391854,  0.01954925,  0.12297749],
       [ 0.37065326,  0.00309102,  0.4618975 , -0.07190746,  0.39215904,
          0.15044176, -0.20379723,  0.37830365,  0.53199457],
       [-0.16896968,  0.05761584,  0.51800037,  0.25537642, -0.2471496 ,
          0.7148691 ,  0.1082198 , -0.13526221, -0.18016662],
       [ 0.20062815, -0.05933283,  0.00727646, -0.03003154,  0.16034699,
          0.06628537, -0.60112652, -0.75068875,  0.01677876],
       [-0.07948854, -0.70730269, -0.24983051,  0.59218953,  0.09556237,
          0.10463252,  0.01848639,  0.02882643,  0.24299776],
       [-0.68274306, -0.01419742,  0.07249683, -0.02894642,  0.35262369,
         -0.01153775, -0.50466425,  0.29335267, -0.24969636],
       [ 0.3275418 , -0.12308207,  0.11308797,  0.09903717,  0.61298247,
         -0.02523614,  0.29403981, -0.02633585, -0.62564572]])
```

The above table shows all the principal components. For better Reference we will take a look at first 2 principal components for the columns

In [15]:

```
pcs_df = pd.DataFrame({'PC1':pca.components_[0],'PC2':pca.components_[1], 'Feature':list1})
pcs_df
```

Out[15]:

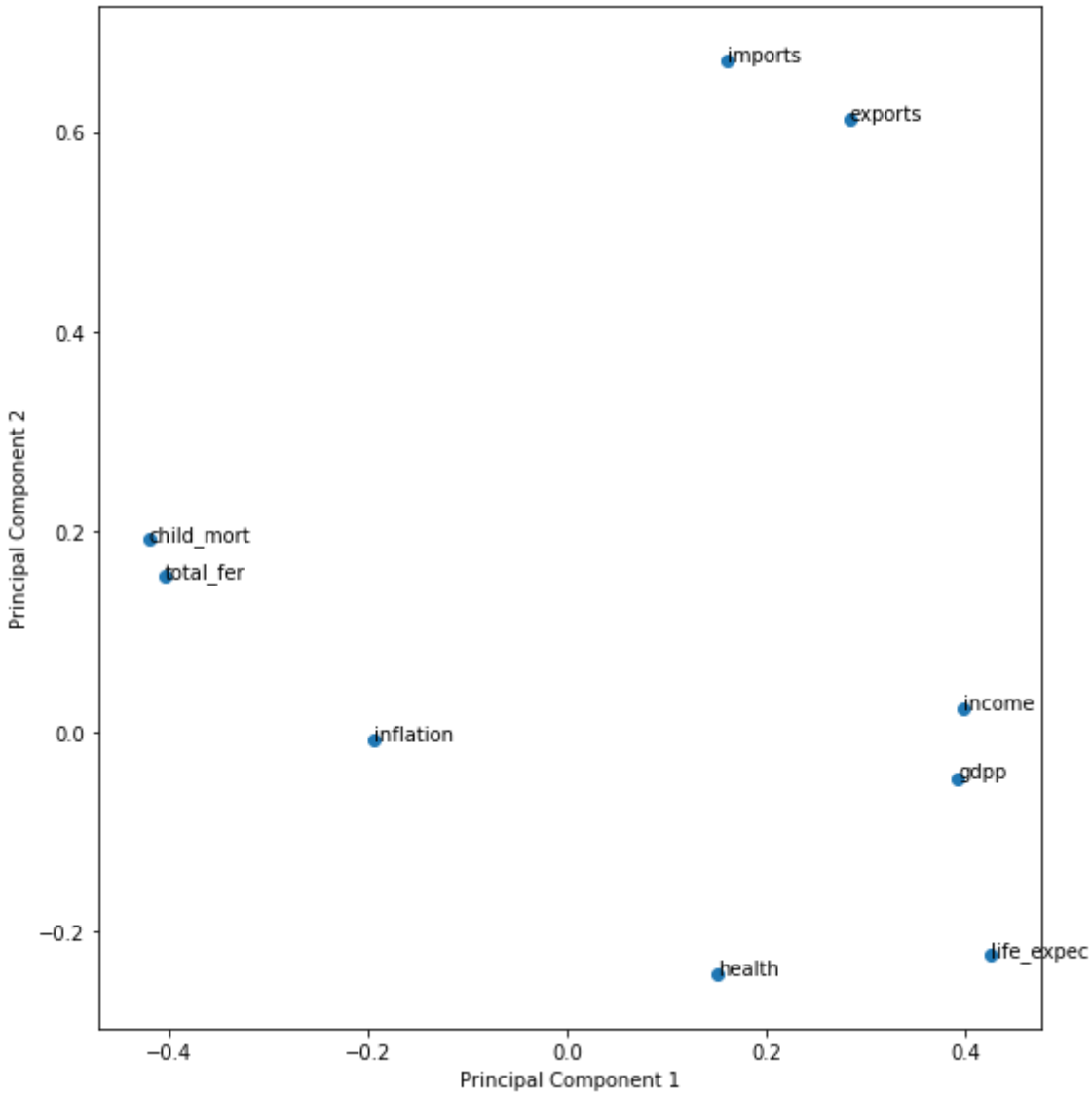
	PC1	PC2	Feature
0	-0.419519	0.192884	child_mort
1	0.283897	0.613163	exports
2	0.150838	-0.243087	health
3	0.161482	0.671821	imports
4	0.398441	0.022536	income
5	-0.193173	-0.008404	inflation
6	0.425839	-0.222707	life_expec
7	-0.403729	0.155233	total_fer
8	0.392645	-0.046022	gdpp

In [16]:

```
import matplotlib.pyplot as plt
import numpy as np
```

In [17]:

```
%matplotlib inline
fig = plt.figure(figsize = (8,8))
plt.scatter(pcs_df.PC1, pcs_df.PC2)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
for i, txt in enumerate(pcs_df.Feature):
    plt.annotate(txt, (pcs_df.PC1[i],pcs_df.PC2[i]))
plt.tight_layout()
plt.show()
```



We see that the fist component is in the direction where the 'charges' variables are heavy

- These 2 components also have the highest of the loadings

Looking at the screeplot to assess the number of needed principal components

In [18]:

```
pca.explained_variance_ratio_
```

Out[18]:

```
array([0.4595174 , 0.17181626, 0.13004259, 0.11053162, 0.07340211,
       0.02484235, 0.0126043 , 0.00981282, 0.00743056])
```

In [19]:

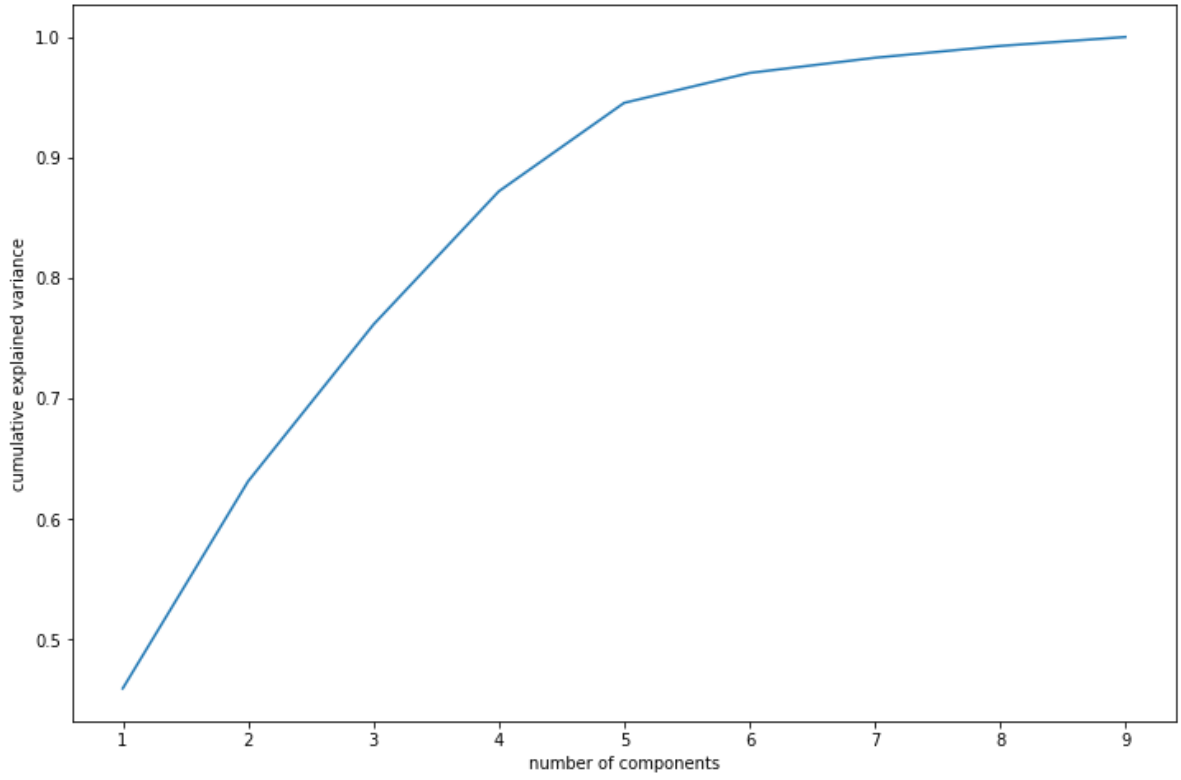
```
np.cumsum(pca.explained_variance_ratio_)
```

Out[19]:

```
array([0.4595174 , 0.63133365, 0.76137624, 0.87190786, 0.94530998,
       0.97015232, 0.98275663, 0.99256944, 1.        ])
```

In [20]:

```
#Making the screeplot - plotting the cumulative variance against the number of components
%matplotlib inline
fig = plt.figure(figsize = (12,8))
plt.plot(range(1,10),np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



In [21]:

```
#Using incremental PCA for efficiency - saves a lot of time on larger datasets
from sklearn.decomposition import IncrementalPCA
pca_final = IncrementalPCA(n_components=6)
```

In [22]:

```
df_train_pca = pca_final.fit_transform(df_parameters)
df_train_pca.shape
```

Out[22]:

```
(167, 6)
```

In [23]:

```
#creating correlation matrix for the principal components
corrmat = np.corrcoef(df_train_pca.transpose())
```

In [24]:

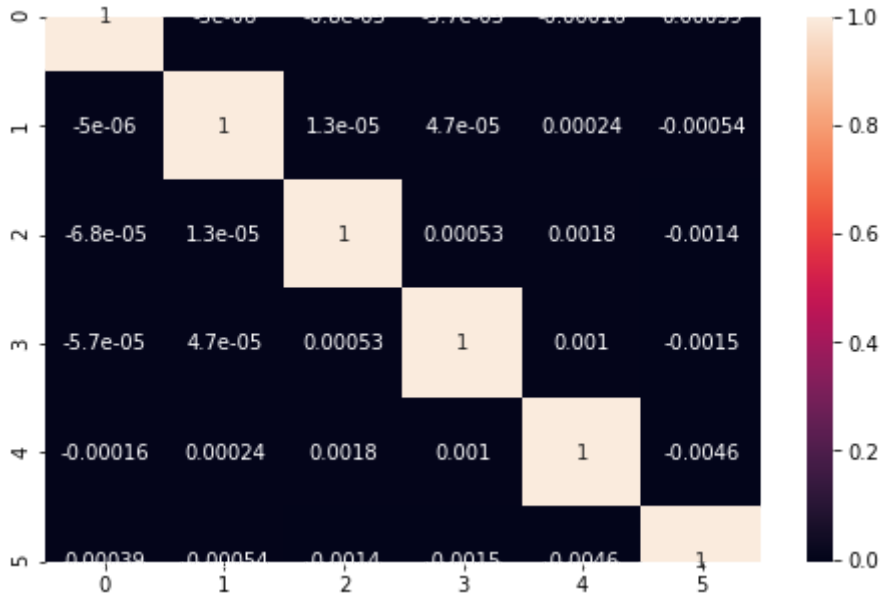
```
import seaborn as sns
```

In [25]:

```
#plotting the correlation matrix
%matplotlib inline
plt.figure(figsize = (8,5))
sns.heatmap(corrmat,annot = True)
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238b3d74eb8>
```



In [26]:

```
# 1s -> 0s in diagonals
corrmat_nodiag = corrmat - np.diagflat(corrmat.diagonal())
print("max corr:",corrmat_nodiag.max(), ", min corr: ", corrmat_nodiag.min(),)
# we see that correlations are indeed very close to 0
```

```
max corr: 0.0018111292855223682 , min corr:  -0.004636332644366443
```

In [27]:

```
df_train_pca
```

Out[27]:

```
array([[-2.91284554,  0.09595714, -0.72223179,  1.00236745, -0.1595579 ,
         0.26192351],
       [ 0.42989418, -0.58827943, -0.33242814, -1.16355837,  0.16470563,
        -0.07493261],
       [-0.28529746, -0.45523941,  1.22445743, -0.86375032,  0.1600654 ,
         0.39969066],
       ...,
       [ 0.49848683,  1.39039482, -0.2383915 , -1.07731727,  1.1732051 ,
        -0.05773731],
       [-1.88735011, -0.10945996,  1.10780542,  0.05884703,  0.62444524,
         0.53821244],
       [-2.86399796,  0.4865062 ,  0.22572025,  0.81914402, -0.26035771,
        -0.21918652]])
```

In [28]:

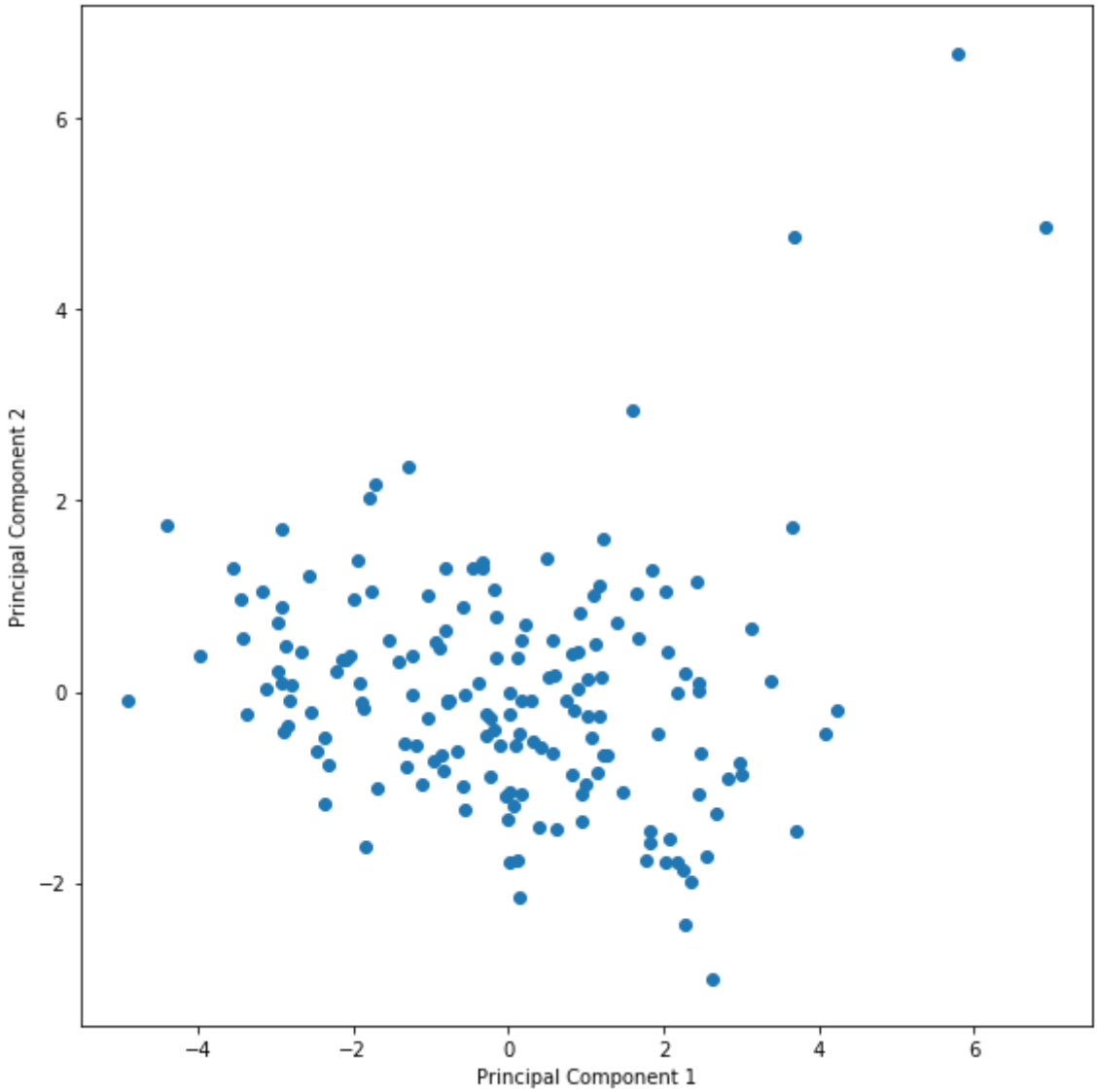
```
y= df['country']
y.head()
```

Out[28]:

```
0            Afghanistan
1                Albania
2                Algeria
3                 Angola
4    Antigua and Barbuda
Name: country, dtype: object
```

In [29]:

```
%matplotlib inline
fig = plt.figure(figsize = (8,8))
plt.scatter(df_train_pca[:,0], df_train_pca[:,1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.tight_layout()
plt.show()
```



It looks like most of the countries are clustered around the centre but a few are outside on the top of the chart

==========

PART-3

Now we remove the any outliers in the data

----------Outlier Treatment----------

In [30]:

```
df_pca_applied = pd.DataFrame(df_train_pca)
df_pca_applied.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

Out[30]:

	0	1	2	3	4	5
count	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02	1.670000e+02
mean	3.191060e-17	-1.063687e-17	-1.329608e-17	-9.307259e-18	-1.063687e-17	-5.983238e-18
std	2.039747e+00	1.247261e+00	1.085066e+00	1.000384e+00	8.151334e-01	4.741626e-01
min	-4.911158e+00	-2.997748e+00	-2.723471e+00	-1.592952e+00	-2.831723e+00	-3.119399e+00
25%	-1.382380e+00	-7.522276e-01	-5.248183e-01	-7.833816e-01	-3.866732e-01	-2.202167e-01
50%	2.249507e-02	-9.466945e-02	-2.415719e-01	-3.113193e-01	-5.417321e-02	-4.353130e-03
75%	1.224243e+00	5.534417e-01	4.081077e-01	7.168438e-01	2.968679e-01	2.377121e-01
90%	2.462520e+00	1.290204e+00	1.286505e+00	1.218526e+00	8.307979e-01	4.772825e-01
95%	3.103814e+00	1.719271e+00	1.811927e+00	1.786495e+00	1.136941e+00	6.436550e-01
99%	4.766567e+00	4.793678e+00	3.279557e+00	2.785117e+00	2.137630e+00	1.059049e+00
max	6.918044e+00	6.681787e+00	6.079148e+00	3.132961e+00	5.355856e+00	1.452548e+00

from percentile of 90% and above the values are increasing significantly, Removing values at 95% should be safe as but before that taking a look at countries which are 95 percentile bracket.

In [31]:

```
df_combined = df_pca_applied
```

In [32]:

```
df_combined['country'] = df['country']
```

In [33]:

```
list2 = list(df_combined[df_combined[0] > df_combined[0].quantile(0.95)].country)
list2
```

Out[33]:

```
['Belgium',
 'Ireland',
 'Luxembourg',
 'Malta',
 'Netherlands',
 'Norway',
 'Qatar',
 'Singapore',
 'Switzerland']
```

Above countries are top of the line countries and have all the parameters ['child_mort', 'exports', 'health', 'imports', 'income','inflation', 'life_expec', 'total_fer', 'gdpp']

In [34]:

```
df[df.country.isin(list2)].describe()
```

Out[34]:

	child_mort	exports	health	imports	income	inflation	life_expec	tot
count	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.0
mean	4.700000	105.044444	8.328889	88.933333	63022.222222	2.239889	80.900000	1.7
std	2.022993	56.908174	3.400296	55.114245	29783.543182	3.191136	1.031988	0.3
min	2.800000	39.700000	1.810000	23.800000	28300.000000	-3.220000	79.500000	1.1
25%	3.200000	64.000000	7.770000	53.300000	45500.000000	0.317000	80.300000	1.5
50%	4.500000	76.400000	9.190000	74.700000	55500.000000	1.880000	80.700000	1.7
75%	4.500000	153.000000	10.700000	142.000000	72100.000000	3.830000	81.300000	1.9
max	9.000000	200.000000	11.900000	174.000000	125000.000000	6.980000	82.700000	2.0

In [35]:

```
df.describe()
```

Out[35]:

	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	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.555689
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.893172
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.100000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.300000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.100000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.800000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.800000

taking a look at mean values of 95% and the mean values of rest of the dataframe, there are massive increase in values. so they dont have deficiency of any kind. Therefore removing 95% values

In [36]:

```
df_combined = df_combined[df_combined[0]<df_combined[0].quantile(0.95)]
```

In [37]:

```
df_combined_y = df_combined.pop('country')
df_combined = df_combined
```

Now dataframe that we can use is df_combined

==========

PART-4

Now we will use clustering algorithm to cluster the data. we will be using both K-means and Hierarchical clustering algorithm

---------- K-Means ----------

Finding the Optimal Number of Clusters

SSD

In [38]:

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

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

In [39]:

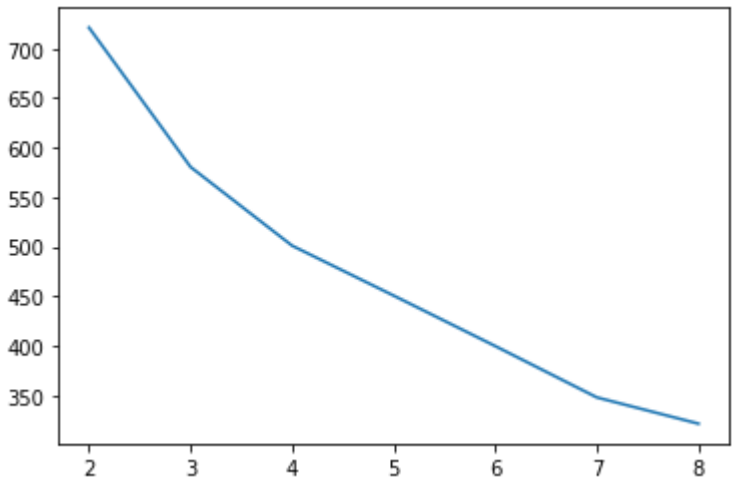
```
# elbow-curve/SSD
ssd = []
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]
for num_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(df_combined)

    ssd.append(kmeans.inertia_)

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

Out[39]:

```
[<matplotlib.lines.Line2D at 0x238b43c1fd0>]
```



Using elbow curve it can be seen that optimal number of `clusters are 3 or 5`

Silhouette Analysis

$$\text{silhouette score} = \frac{p - q}{max(p, q)}$$

p is the mean distance to the points in the nearest cluster that the data point is not a part of

q is the mean intra-cluster distance to all the points in its own cluster.

- The value of the silhouette score range lies between -1 to 1.
- A score closer to 1 indicates that the data point is very similar to other data points in the cluster,

- A score closer to -1 indicates that the data point is not similar to the data points in its cluster.

In [40]:

```
# silhouette analysis
range_n_clusters = [2, 3, 4, 5, 6, 7, 8]

for num_clusters in range_n_clusters:

    # intialise kmeans
    kmeans = KMeans(n_clusters=num_clusters, max_iter=50)
    kmeans.fit(df_combined)

    cluster_labels = kmeans.labels_

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

```
For n_clusters=2, the silhouette score is 0.34580198973508147
For n_clusters=3, the silhouette score is 0.29622026437619325
For n_clusters=4, the silhouette score is 0.3017000034615611
For n_clusters=5, the silhouette score is 0.2465800611676325
For n_clusters=6, the silhouette score is 0.23499546908966848
For n_clusters=7, the silhouette score is 0.23918547169594456
For n_clusters=8, the silhouette score is 0.24818228927135683
```

Silhouette score shows that `n_cluster = 2` is best, however if we compare `cluster = 3` and `cluster = 5`, then cluster 3 is better. But overall these values are not differentiated by very large amount

---

Creating clusters using K-Means Algorithm using `K=3`

In [41]:

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

Out[41]:

```
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 [42]:

```
# assign the label
# assign cluster labels
df_after_cluster = pd.DataFrame(kmeans.labels_,columns=['labels'])
df_after_cluster['country'] = df_combined_y
df_final_merged = df_after_cluster.merge(df,how='inner', on=['country'])
df_final_merged.head()
```

Out[42]:

	labels	country	child_mort	exports	health	imports	income	inflation	life_expec	total_
0	1	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5
1	0	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1
2	0	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2
3	1	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6
4	0	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2

In [43]:

```
df_final_merged['labels'].value_counts()
```

Out[43]:

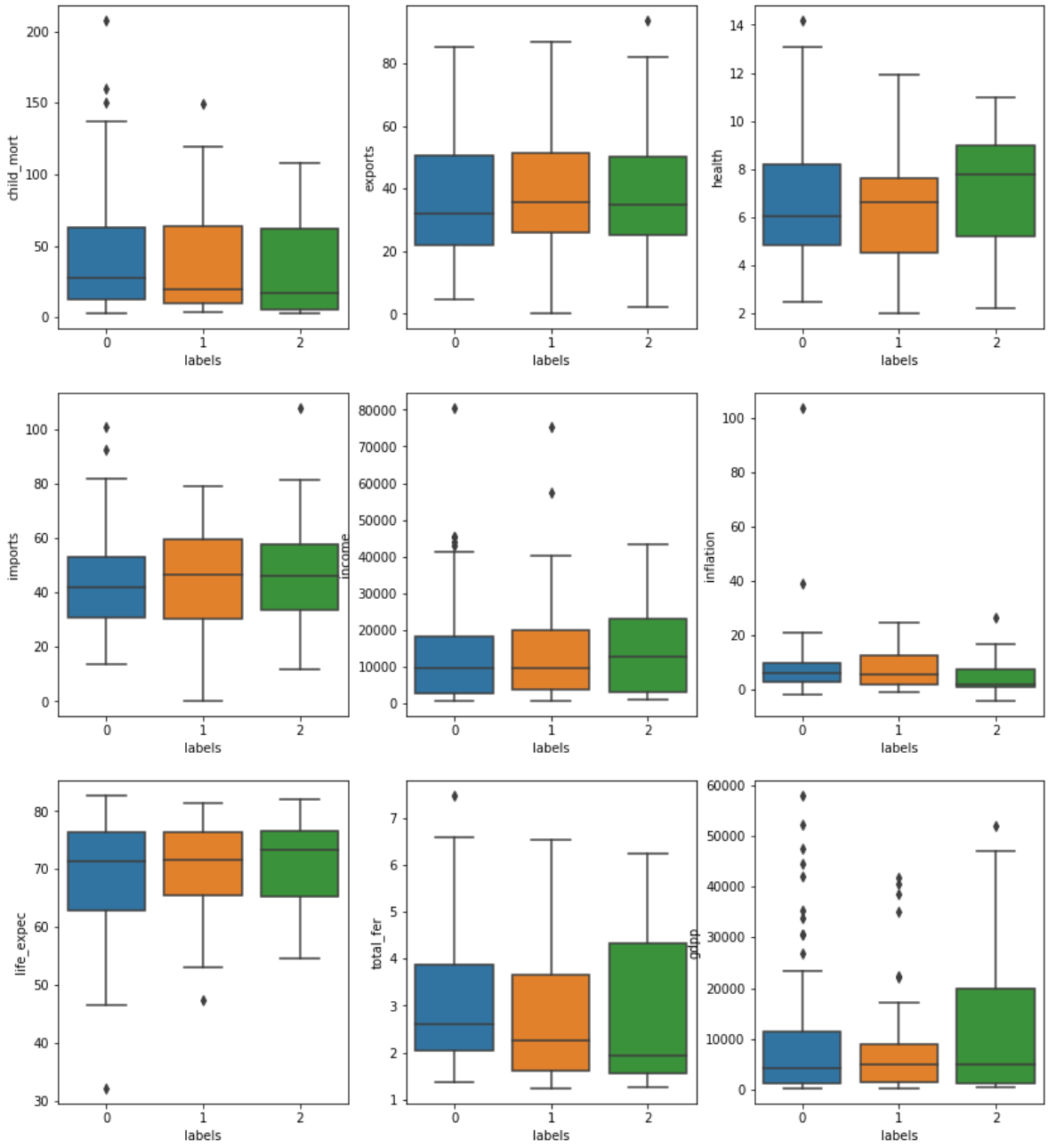
```
0    79
1    45
2    25
Name: labels, dtype: int64
```

In [44]:

```
fig,axs = plt.subplots(3,3,figsize=(14,16))
sns.boxplot('labels','child_mort', data =df_final_merged,ax= axs[0,0] )
sns.boxplot('labels','exports', data =df_final_merged,ax= axs[0,1] )
sns.boxplot('labels','health', data =df_final_merged,ax= axs[0,2] )
sns.boxplot('labels','imports', data =df_final_merged,ax= axs[1,0] )
sns.boxplot('labels','income', data =df_final_merged,ax= axs[1,1] )
sns.boxplot('labels','inflation', data =df_final_merged,ax= axs[1,2] )
sns.boxplot('labels','life_expec', data =df_final_merged,ax= axs[2,0] )
sns.boxplot('labels','total_fer', data =df_final_merged,ax= axs[2,1] )
sns.boxplot('labels','gdpp', data =df_final_merged,ax= axs[2,2] )
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238b45bf7b8>
```



The clusters formed with K-Means Algorithm with K=3 is not able to differentiate well amongst the classes

Creating clusters using K-Means Algorithm using K=5

In [45]:

```
# model with k=5
kmeans = KMeans(n_clusters=5, max_iter=50)
kmeans.fit(df_combined)
```

Out[45]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [46]:

```
# assign the label
# assign cluster labels
df_after_cluster = pd.DataFrame(kmeans.labels_,columns=['labels'])
df_after_cluster['country'] = df_combined_y
df_final_merged = df_after_cluster.merge(df,how='inner', on=['country'])
df_final_merged.head()
```

Out[46]:

	labels	country	child_mort	exports	health	imports	income	inflation	life_expec	total_
0	1	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5
1	2	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1
2	4	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2
3	1	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6
4	2	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2

In [47]:

```
df_final_merged['labels'].value_counts()
```

Out[47]:

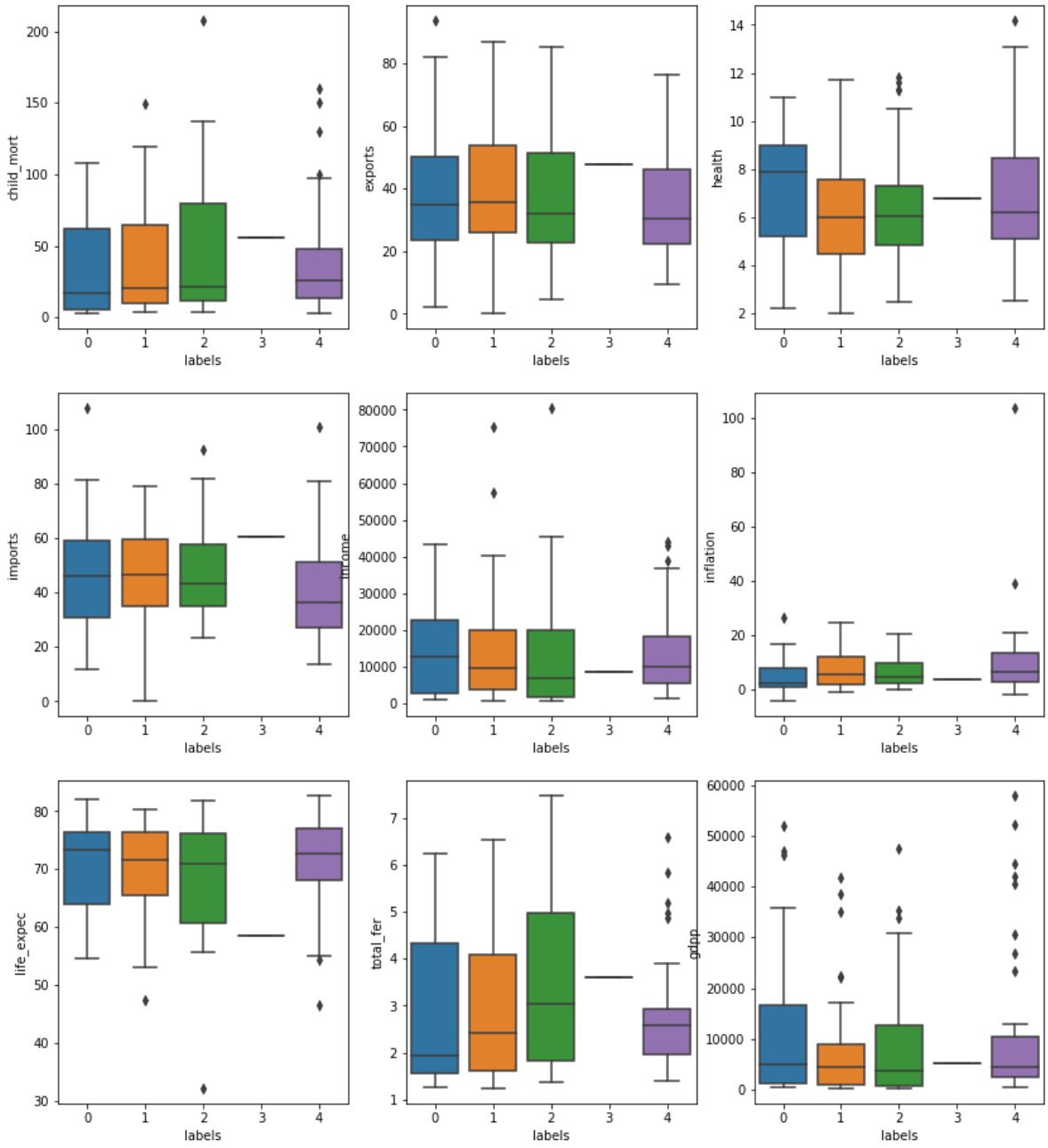
```
4    44
1    41
2    40
0    23
3     1
Name: labels, dtype: int64
```

In [48]:

```
fig,axs = plt.subplots(3,3,figsize=(14,16))
sns.boxplot('labels','child_mort', data =df_final_merged,ax= axs[0,0] )
sns.boxplot('labels','exports', data =df_final_merged,ax= axs[0,1] )
sns.boxplot('labels','health', data =df_final_merged,ax= axs[0,2] )
sns.boxplot('labels','imports', data =df_final_merged,ax= axs[1,0] )
sns.boxplot('labels','income', data =df_final_merged,ax= axs[1,1] )
sns.boxplot('labels','inflation', data =df_final_merged,ax= axs[1,2] )
sns.boxplot('labels','life_expec', data =df_final_merged,ax= axs[2,0] )
sns.boxplot('labels','total_fer', data =df_final_merged,ax= axs[2,1] )
sns.boxplot('labels','gdpp', data =df_final_merged,ax= axs[2,2] )
```

Out[48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238b4215f98>
```



The clusters formed with K-Means Algorithm with K=5 is not able to differentiate well amongst the classes

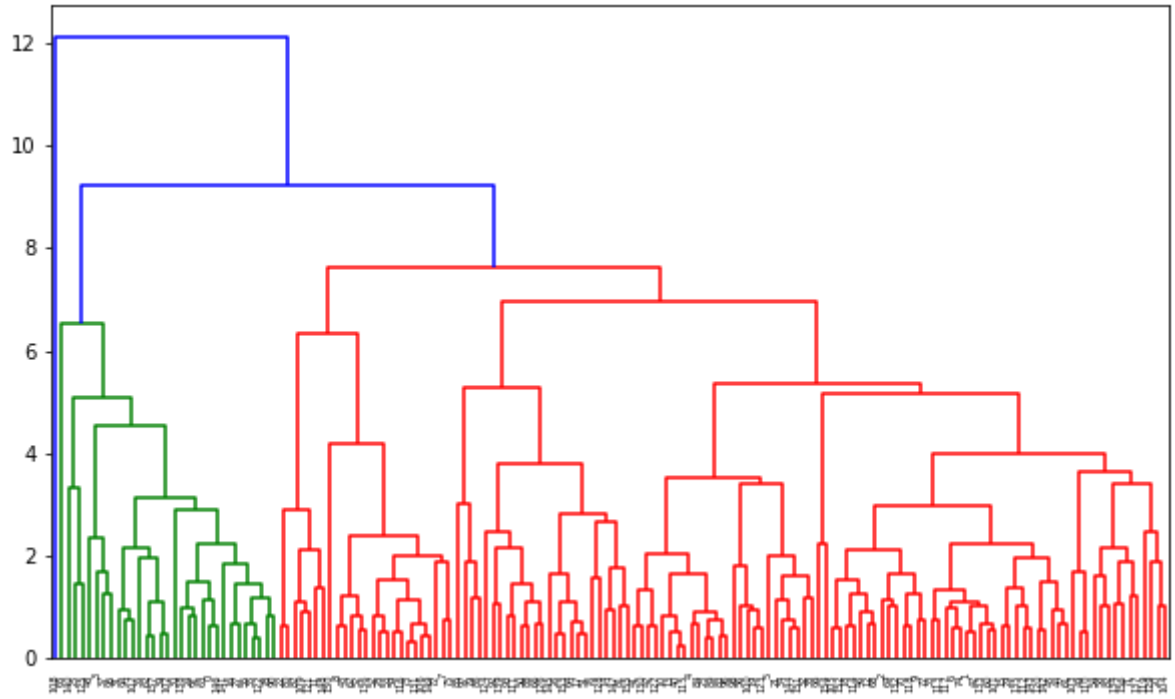
----------Hierarchical Clustering----------

complete linkage

Number of clusters = 5

In [49]:

```
# complete linkage
plt.figure(figsize = (10,6))
mergings = linkage(df_combined, method="complete", metric='euclidean')
dendrogram(mergings)
plt.show()
```



These cluster shows good division amongst cluster=2, cluster=3 and cluster=5

the algorithmns of kmeans and Hierarchical clusters shows 3 and 5 as optimal clusters

lets take 5 clustrers to divide the countries. the idea is that more number of clusters might help in taking out a deeper inference amonst parameters.

In [50]:

```
cluster_labels = cut_tree(mergings, n_clusters=5).reshape(-1, )
cluster_labels
```

Out[50]:

```
array([0, 1, 1, 0, 1, 1, 1, 2, 2, 1, 1, 2, 1, 1, 3, 1, 0, 3, 1, 1, 1, 1,
       2, 1, 0, 0, 3, 0, 2, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 2,
       1, 1, 1, 1, 0, 1, 3, 3, 2, 2, 1, 0, 1, 2, 1, 2, 1, 1, 0, 0, 3, 0,
       3, 2, 1, 1, 1, 1, 2, 2, 1, 2, 3, 1, 0, 3, 2, 3, 1, 1, 1, 3, 3, 2,
       3, 1, 0, 0, 3, 3, 0, 0, 1, 3, 1, 1, 1, 1, 0, 1, 1, 1, 2, 0, 4, 2,
       1, 3, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 0, 1, 3, 0, 3, 1, 3, 1, 1, 2,
       1, 1, 1, 1, 2, 1, 0, 3, 0, 0, 1, 1, 1, 3, 0, 3, 2, 2, 2, 1, 1, 3,
       1, 3, 1, 0])
```

In [51]:

```
# assign cluster labels
df_after_cluster = pd.DataFrame(df_combined_y)
df_after_cluster['labels'] = cluster_labels
df_final_merged = df_after_cluster.merge(df,how='inner', on=['country'])
df_final_merged.head()
```

Out[51]:

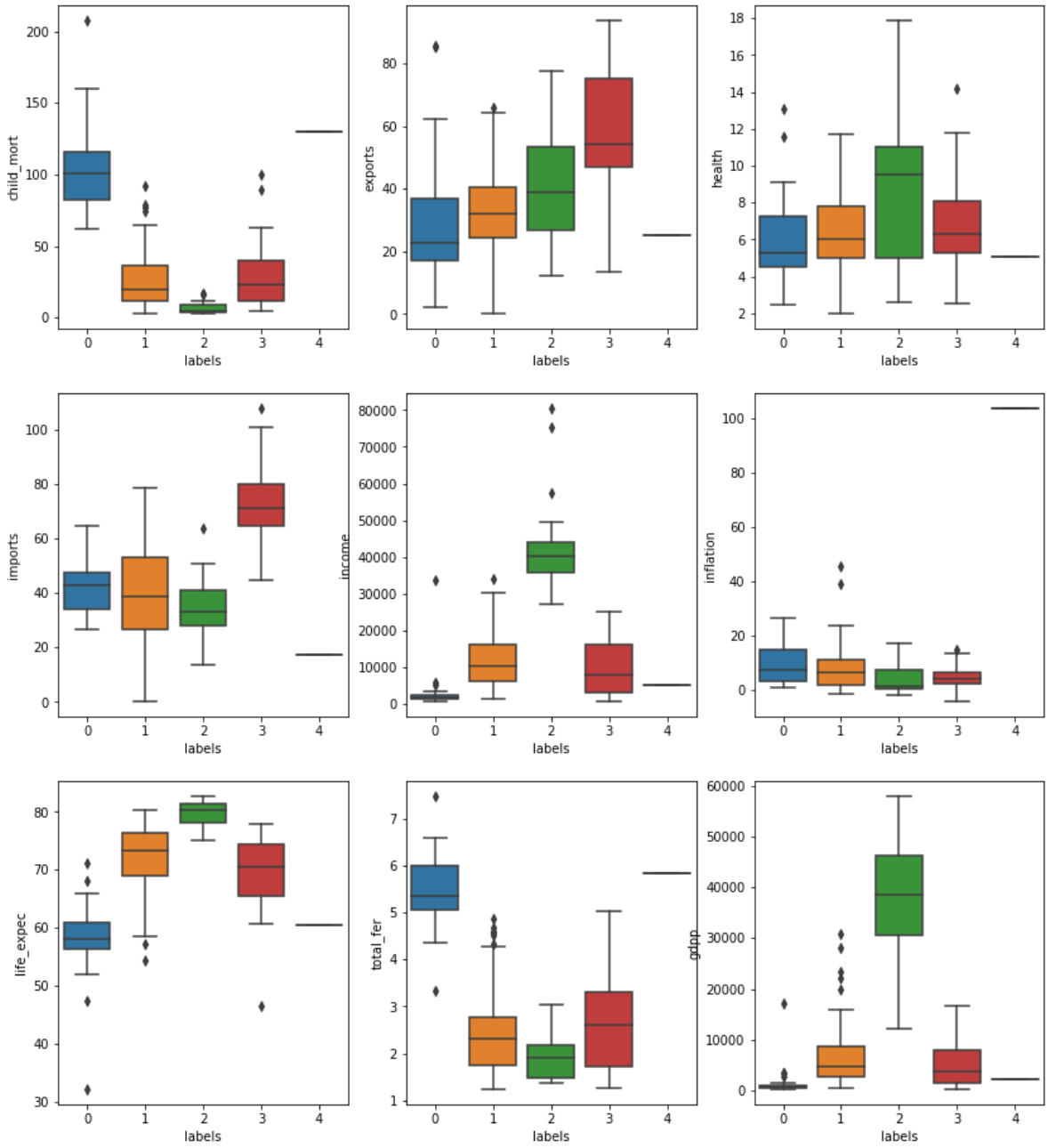
	country	labels	child_mort	exports	health	imports	income	inflation	life_expec	total_
0	Afghanistan	0	90.2	10.0	7.58	44.9	1610	9.44	56.2	5
1	Albania	1	16.6	28.0	6.55	48.6	9930	4.49	76.3	1
2	Algeria	1	27.3	38.4	4.17	31.4	12900	16.10	76.5	2
3	Angola	0	119.0	62.3	2.85	42.9	5900	22.40	60.1	6
4	Antigua and Barbuda	1	10.3	45.5	6.03	58.9	19100	1.44	76.8	2

In [52]:

```
fig,axs = plt.subplots(3,3,figsize=(14,16))
sns.boxplot('labels','child_mort', data =df_final_merged,ax= axs[0,0] )
sns.boxplot('labels','exports', data =df_final_merged,ax= axs[0,1] )
sns.boxplot('labels','health', data =df_final_merged,ax= axs[0,2] )
sns.boxplot('labels','imports', data =df_final_merged,ax= axs[1,0] )
sns.boxplot('labels','income', data =df_final_merged,ax= axs[1,1] )
sns.boxplot('labels','inflation', data =df_final_merged,ax= axs[1,2] )
sns.boxplot('labels','life_expec', data =df_final_merged,ax= axs[2,0] )
sns.boxplot('labels','total_fer', data =df_final_merged,ax= axs[2,1] )
sns.boxplot('labels','gdpp', data =df_final_merged,ax= axs[2,2] )
```

Out[52]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238b50a5080>
```



It looks like Hierarchical clusters with n = 5 is provideing a good differentiation

Lets try using `n_clusters = 3`

In [53]:

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

Out[53]:

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

In [54]:

```
# assign cluster labels
df_after_cluster = pd.DataFrame(df_combined_y)
df_after_cluster['labels'] = cluster_labels
df_final_merged = df_after_cluster.merge(df,how='inner', on=['country'])
df_final_merged.head()
```

Out[54]:

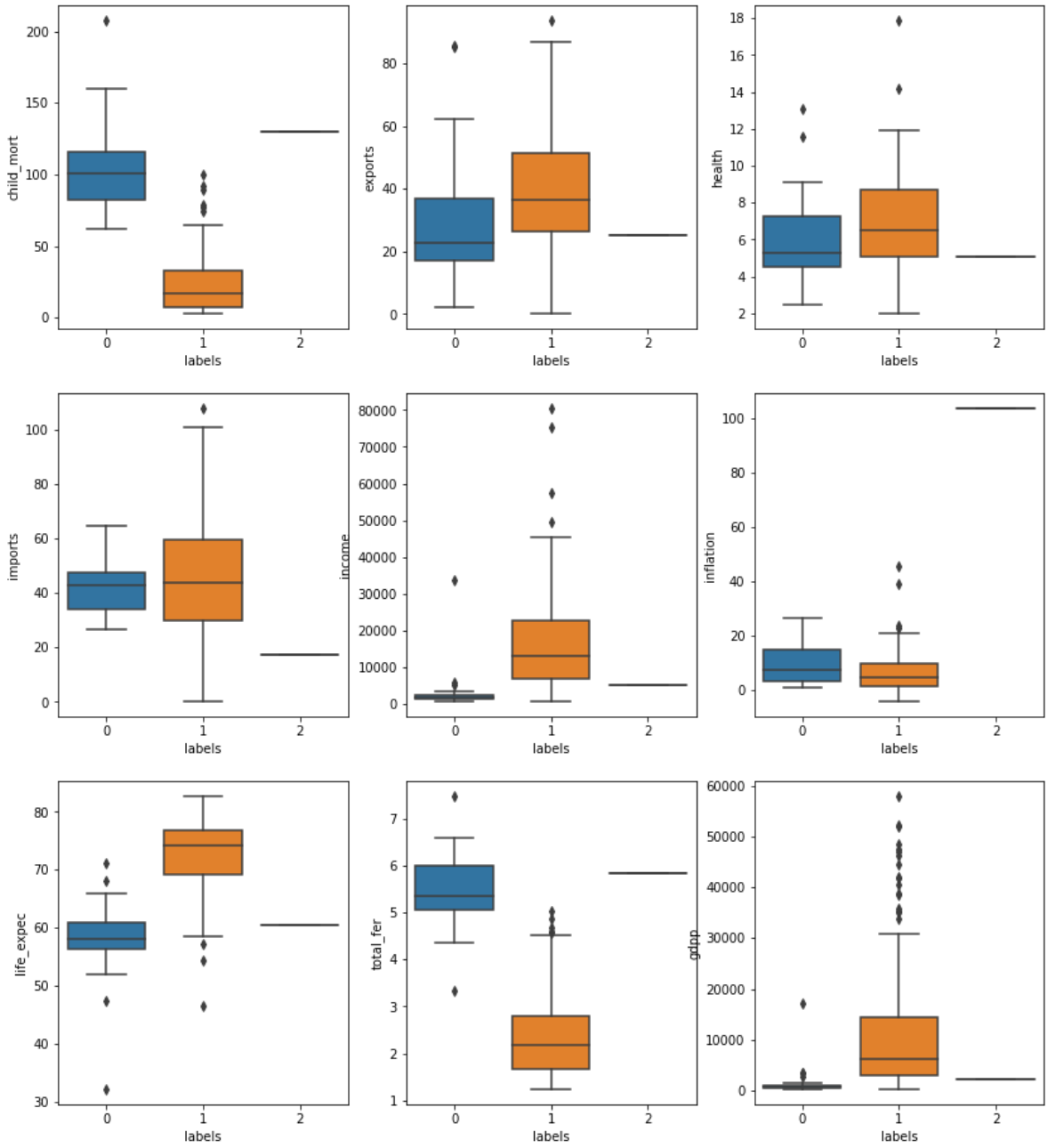
	country	labels	child_mort	exports	health	imports	income	inflation	life_expec	total_
0	Afghanistan	0	90.2	10.0	7.58	44.9	1610	9.44	56.2	5
1	Albania	1	16.6	28.0	6.55	48.6	9930	4.49	76.3	1
2	Algeria	1	27.3	38.4	4.17	31.4	12900	16.10	76.5	2
3	Angola	0	119.0	62.3	2.85	42.9	5900	22.40	60.1	6
4	Antigua and Barbuda	1	10.3	45.5	6.03	58.9	19100	1.44	76.8	2

In [55]:

```
fig,axs = plt.subplots(3,3,figsize=(14,16))
sns.boxplot('labels','child_mort', data =df_final_merged,ax= axs[0,0] )
sns.boxplot('labels','exports', data =df_final_merged,ax= axs[0,1] )
sns.boxplot('labels','health', data =df_final_merged,ax= axs[0,2] )
sns.boxplot('labels','imports', data =df_final_merged,ax= axs[1,0] )
sns.boxplot('labels','income', data =df_final_merged,ax= axs[1,1] )
sns.boxplot('labels','inflation', data =df_final_merged,ax= axs[1,2] )
sns.boxplot('labels','life_expec', data =df_final_merged,ax= axs[2,0] )
sns.boxplot('labels','total_fer', data =df_final_merged,ax= axs[2,1] )
sns.boxplot('labels','gdpp', data =df_final_merged,ax= axs[2,2] )
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238b5877080>
```



In [56]:

```
df_final_merged.labels.value_counts()
```

Out[56]:

```
1    126
0     31
2      1
Name: labels, dtype: int64
```

Hierarchical clustering has made 3 labels and they are divided in a way described above.

Label number 2 only have 1 country. So we will try KMeans clustering to see if we can create better cluster

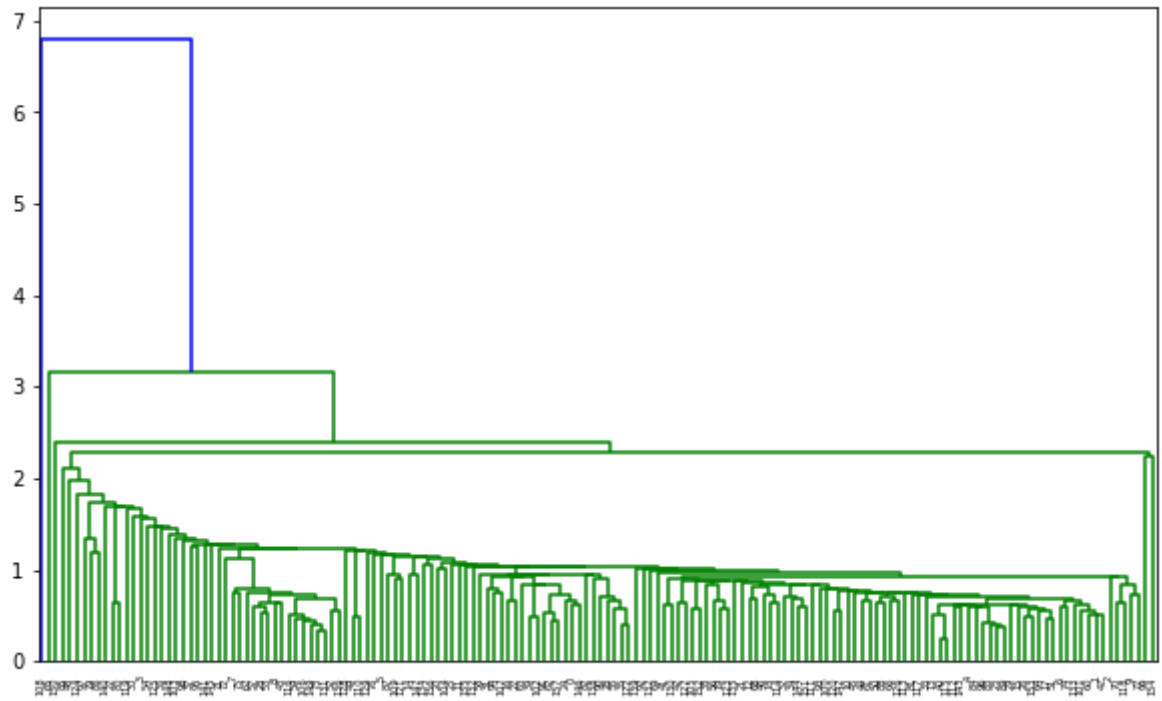
Looking at the charts above it looks like division when n=5 was better as compared to n=3

Single linkage

Number of clusters = 5

In [60]:

```
# single linkage
plt.figure(figsize = (10,6))
mergings = linkage(df_combined, method="single", metric='euclidean')
dendrogram(mergings)
plt.show()
```



These cluster shows good division amongst cluster=2, cluster=3 and cluster=5

the algorithmns of kmeans and Hierarchical clusters shows 3 and 5 as optimal clusters

lets take 5 clustrers to divide the countries. the idea is that more number of clusters might help in taking out a deeper inference amonst parameters.

In [61]:

```
cluster_labels = cut_tree(mergings, n_clusters=5).reshape(-1, )
cluster_labels
```

Out[61]:

```
array([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, 0, 0, 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, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0,
       0, 0, 0, 0, 0, 0, 0, 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, 4, 0, 0, 0,
       2, 0, 0, 0])
```

In [62]:

```
# assign cluster labels
df_after_cluster = pd.DataFrame(df_combined_y)
df_after_cluster['labels'] = cluster_labels
df_final_merged = df_after_cluster.merge(df,how='inner', on=['country'])
df_final_merged.head()
```

Out[62]:

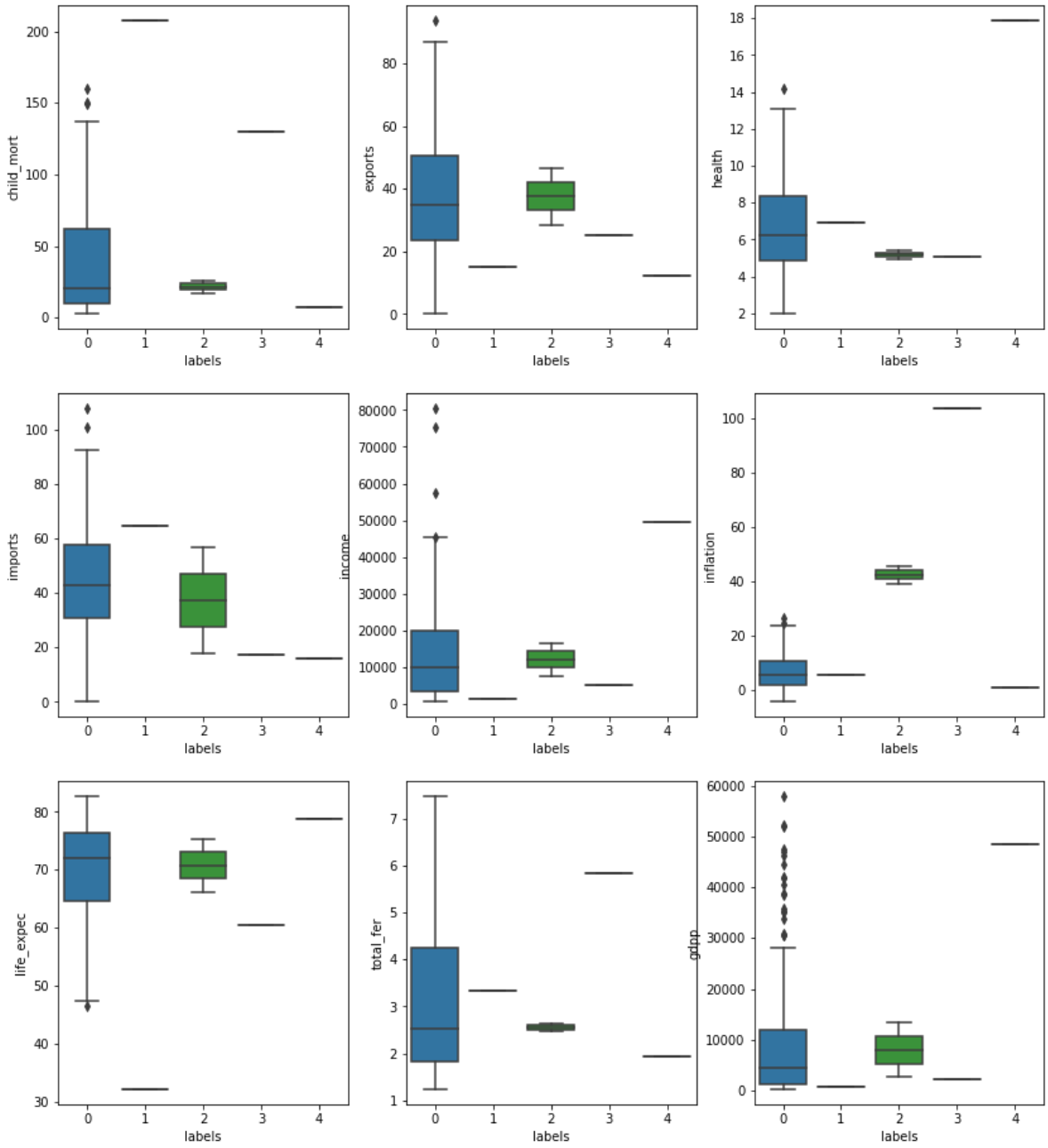
	country	labels	child_mort	exports	health	imports	income	inflation	life_expec	total_
0	Afghanistan	0	90.2	10.0	7.58	44.9	1610	9.44	56.2	5
1	Albania	0	16.6	28.0	6.55	48.6	9930	4.49	76.3	1
2	Algeria	0	27.3	38.4	4.17	31.4	12900	16.10	76.5	2
3	Angola	0	119.0	62.3	2.85	42.9	5900	22.40	60.1	6
4	Antigua and Barbuda	0	10.3	45.5	6.03	58.9	19100	1.44	76.8	2

In [63]:

```
fig,axs = plt.subplots(3,3,figsize=(14,16))
sns.boxplot('labels','child_mort', data =df_final_merged,ax= axs[0,0] )
sns.boxplot('labels','exports', data =df_final_merged,ax= axs[0,1] )
sns.boxplot('labels','health', data =df_final_merged,ax= axs[0,2] )
sns.boxplot('labels','imports', data =df_final_merged,ax= axs[1,0] )
sns.boxplot('labels','income', data =df_final_merged,ax= axs[1,1] )
sns.boxplot('labels','inflation', data =df_final_merged,ax= axs[1,2] )
sns.boxplot('labels','life_expec', data =df_final_merged,ax= axs[2,0] )
sns.boxplot('labels','total_fer', data =df_final_merged,ax= axs[2,1] )
sns.boxplot('labels','gdpp', data =df_final_merged,ax= axs[2,2] )
```

Out[63]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238b615df28>
```



Using single linkage, The clusters are not formed with good proportions.

Therefore for final analysis we will use Hierarchical clustering with 5 clusters using complete linkage

In [ ]:

```
```

==========

PART-5

Now we will Analyse the data using clusters formed with Hierarchical clustering algorithm.

Number of clusters = 5 and using complete linkage

---------- Analysis ----------

Lets again set the parameters for Hierarchical clustering

In [ ]:

```
```

In [67]:

```
# complete linkage
plt.figure(figsize = (10,6))
mergings = linkage(df_combined, method="complete", metric='euclidean')


cluster_labels = cut_tree(mergings, n_clusters=5).reshape(-1, )


# assign cluster labels
df_after_cluster = pd.DataFrame(df_combined_y)
df_after_cluster['labels'] = cluster_labels
df_final_merged = df_after_cluster.merge(df,how='inner', on=['country'])
```

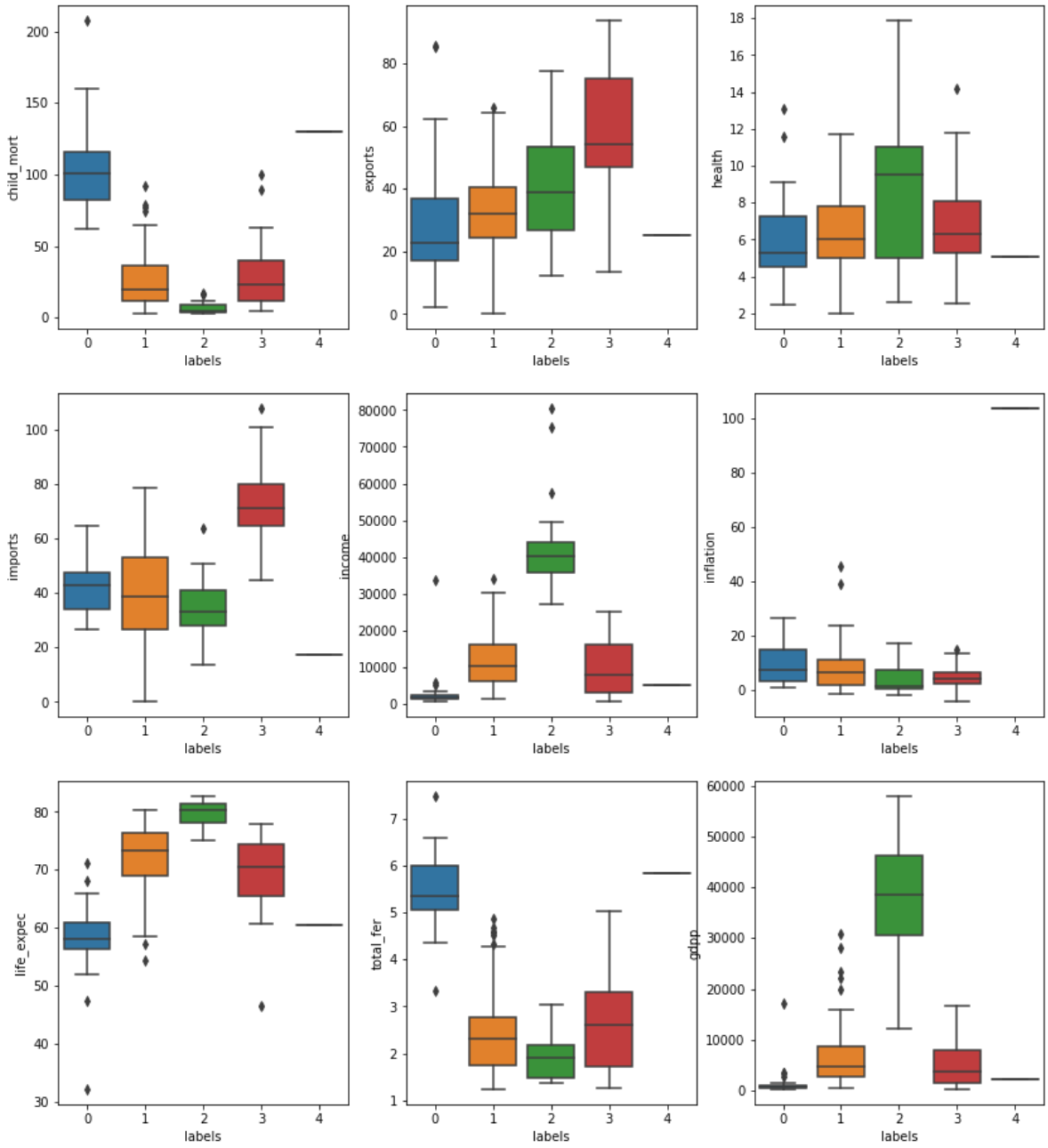
```
<Figure size 720x432 with 0 Axes>
```

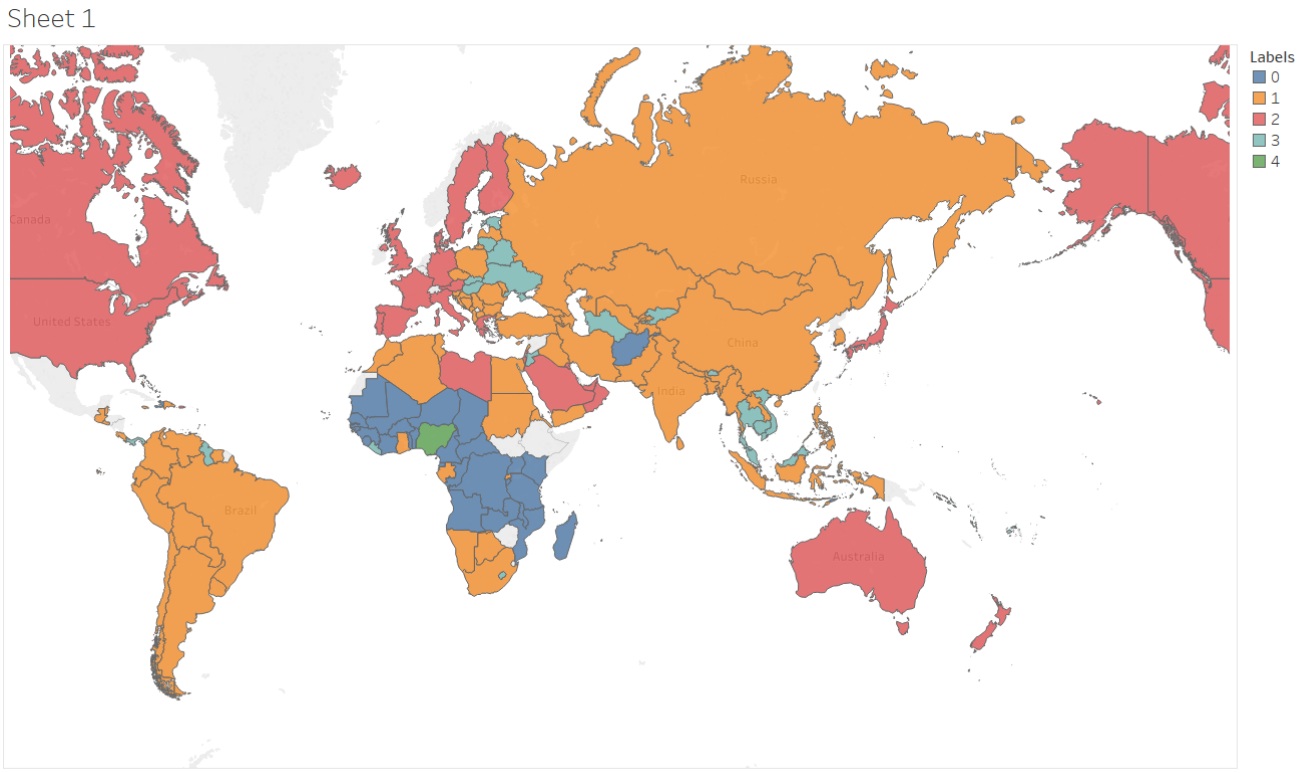
In [68]:

```
fig,axs = plt.subplots(3,3,figsize=(14,16))
sns.boxplot('labels','child_mort', data =df_final_merged,ax= axs[0,0] )
sns.boxplot('labels','exports', data =df_final_merged,ax= axs[0,1] )
sns.boxplot('labels','health', data =df_final_merged,ax= axs[0,2] )
sns.boxplot('labels','imports', data =df_final_merged,ax= axs[1,0] )
sns.boxplot('labels','income', data =df_final_merged,ax= axs[1,1] )
sns.boxplot('labels','inflation', data =df_final_merged,ax= axs[1,2] )
sns.boxplot('labels','life_expec', data =df_final_merged,ax= axs[2,0] )
sns.boxplot('labels','total_fer', data =df_final_merged,ax= axs[2,1] )
sns.boxplot('labels','gdpp', data =df_final_merged,ax= axs[2,2] )
```

Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x238b7b7c978>
```





The above image is generated using Tableau for the output labels.

NOTE : The colours in the Map and Boxplot are different for different labels

It shows an Interesting Pattern

Most of the south America and Asia are in the same clusters

Most of the North America, some parts of Europe and Australia are in the clusters

Most of the Africa follows a similar pattern

label - 3 have less number of countries in areas including europe and asia

And there is one country -- > Nigeria in Africa which worst of all

Child Mortality Rate And Total Fertility Rate-->

Child Mortality Rate is very low for countries in North America, Australia, some parts of Europe
And very High for countries like African countries including Nigeria
Asian and South American countries are in the middle

Income, Life expantancy And GDP follows a similar pattern

It is high for countries in North America, Australia, some parts of Europe
And low for countries like African countries including Nigeria
Asian and South American countries are in the middle

Exports And Imports

Highest for some parts of europe and some parts of south asia
Exports are very less for African countries and Imports are less for North America, Australia and some parts of europe

Health

It is very high for developed countries like North America, australia and some parts of europe
And the mean value of the rest of the world is very close to each other with lowest in africa

Inflation

It is Exceptionally high for Nigeria
Marginally low for eveloped countries
REst of the world is almost same

Most of the countries in Africa are in dire need of aid

In [91]:

```
df_final_merged[df_final_merged['labels']==0].country
```

Out[91]:

```
0                    Afghanistan
3                         Angola
16                         Benin
24                  Burkina Faso
25                       Burundi
27                      Cameroon
30     Central African Republic
31                          Chad
35                       Comoros
36              Congo, Dem. Rep.
37                   Congo, Rep.
39                 Cote d'Ivoire
48             Equatorial Guinea
55                        Gambia
62                        Guinea
63                 Guinea-Bissau
65                         Haiti
78                         Kenya
90                    Madagascar
91                        Malawi
94                          Mali
95                    Mauritania
102                   Mozambique
107                        Niger
122                      Senegal
125                 Sierra Leone
138                     Tanzania
140                  Timor-Leste
141                         Togo
146                       Uganda
157                       Zambia
Name: country, dtype: object
```

These are a few names of countries that are in dire need of aid

In [ ]:

```

```