IBM Machine Learning

Course 4: Supervised Learning

Topic: Country data

1. Introduction

1.1 Data

This data set contains socioeconomic and health information of different countries. The dataset can be found in kaggle: https://www.kaggle.com/rohan0301/unsupervised-learning-on-country-data. The metadata as below:

Column Name	Descritption
country	Name of the country
child_mort	Death of children under 5 years of age per 100 live births
exports	Exports of goods and services per capita. Given as %age of the GDP per capita
health	Total health spending per capita. Given as %age of GDP per capita
imports	Imports of goods and services per capita. Given as %age of the GDP per capita
Income	Net income per person
Inflation	The measurement of the annual growth rate of the Total GDP
life_expec	The average number of years a new born child would live if the current mortality patterns are to remain
total_fer	The number of children that would be born to each woman if the current age-fertility rates remain
gdpp	The GDP per capita. Calculated as the Total GDP divided by the total population

1.2 Objective

This analysis is to categories the countries to determine which countries need more aid.

2. Explorative Data Analysis

In [20]:
```
## import the library and dataset
import warnings
warnings.filterwarnings('ignore')

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

In [2]:
```
data=pd.read_csv("data/Country-data.csv")
```

2.1 The columns and datatypes

In [3]:
```
## the size of data
data.shape
```

Out[3]:
```
(167, 10)
```

In [4]:
```
## the name of each column
data.columns.tolist()
```

Out[4]:
```
['country',
 'child_mort',
 'exports',
 'health',
 'imports',
 'income',
 'inflation',
 'life_expec',
 'total_fer',
 'gdpp']
```

In [5]:
```
## datatype
data.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
```

In [6]:
```
## data the distribution of each column
data.describe()
```

Out[6]:

	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

All the value are positive except for inflation. We need to be careful about the negative value when applying transformation.

2.2 Data Transformation

As health, import and export columns are percent of GDP, we would like to convert them to absolute value.

In [7]:
```
data['health']=data['health']*data['gdpp']/100
data['imports']=data['imports']*data['gdpp']/100
data['exports']=data['exports']*data['gdpp']/100
```

In [8]:
```
## The numeric data
numeric_columns=[x for x in data.columns if x!='country']
```

In [9]:
```
numeric_columns
```

Out[9]:
```
['child_mort',
 'exports',
 'health',
 'imports',
 'income',
 'inflation',
 'life_expec',
 'total_fer',
 'gdpp']
```

In [10]:
```
## The skewness of numeric data
skew_columns = data[numeric_columns].skew().sort_values(ascending=False)
skew_columns
```

Out[10]:
```
exports       6.720171
imports       6.618500
inflation     5.154049
health        2.526029
income        2.231480
gdpp          2.218051
child_mort    1.450774
total_fer     0.967092
life_expec   -0.970996
dtype: float64
```

In [11]:
```
skew_columns = skew_columns.loc[skew_columns.abs() > 1]
skew_columns
```

Out[11]:
```
exports       6.720171
imports       6.618500
inflation     5.154049
health        2.526029
income        2.231480
gdpp          2.218051
child_mort    1.450774
dtype: float64
```

In [12]:
```
# Perform log transform on skewed columns, for inflation we need to do it separately
for col in skew_columns.index.tolist():
    if col=='inflation':
        data[col] = np.log1p(data[col]-data[col].min())
    data[col] = np.log1p(data[col])
```

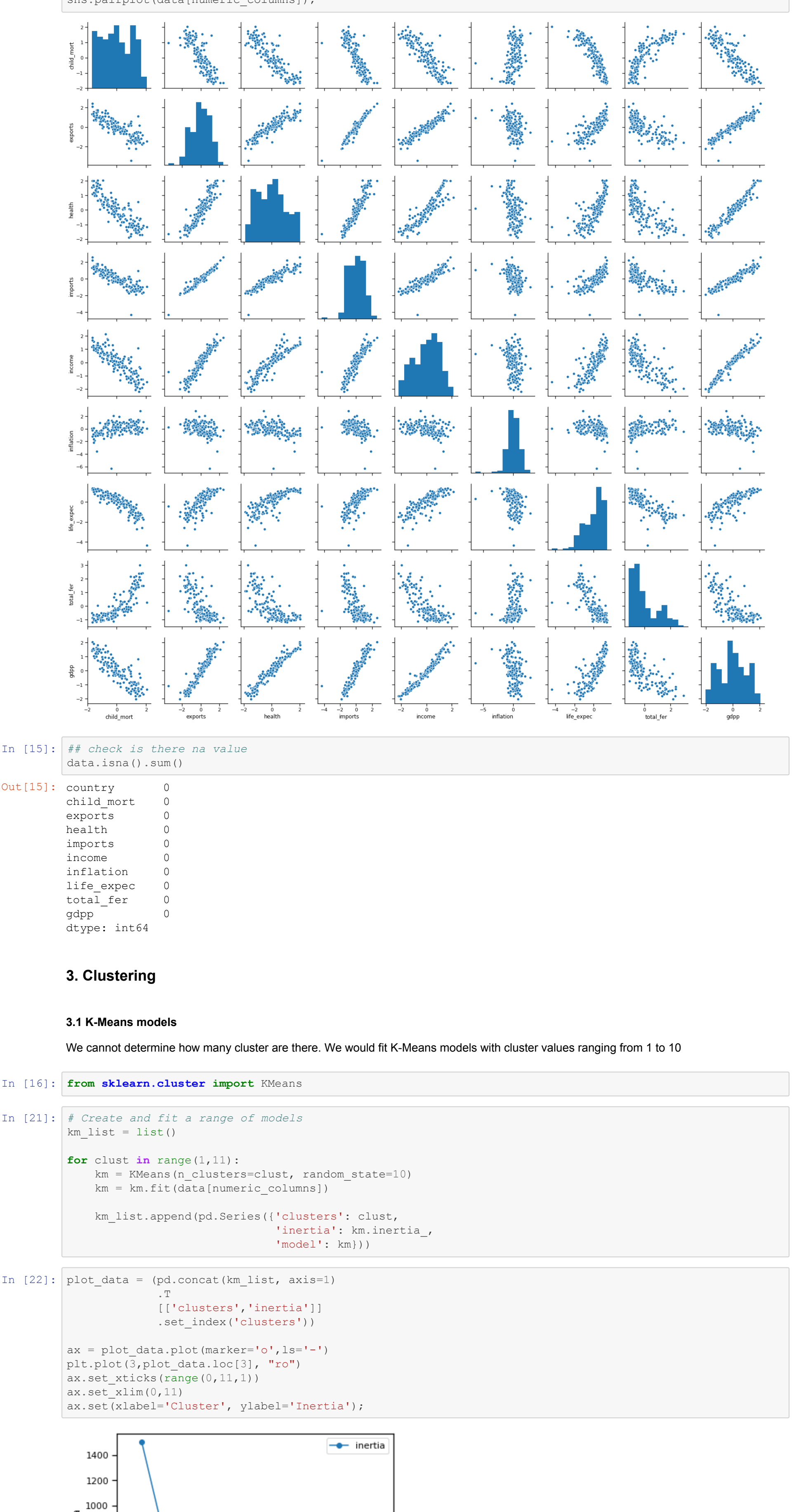
2.3 Feature scaling

In [13]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
data[numeric_columns] = sc.fit_transform(data[numeric_columns])

data.head(4)
```

Out[13]:

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	Afghanistan	1.278012	-1.744684	-1.210000	-1.281862	-1.413035	0.547657	-1.619092	1.902882	-1.460301
1	Albania	-0.239708	-0.186626	-0.068390	-0.306310	0.071660	-0.075626	0.647866	-0.859973	-0.122835
2	Algeria	0.198464	0.021291	-0.310450	-0.217929	0.285240	1.056361	0.670423	-0.038404	-0.064916
3	Angola	1.531161	0.150499	-0.663244	-0.170672	-0.353257	1.361207	-1.179234	2.128151	-0.221309

In [14]:
```
## the pairplot of the transformed and scaled data
sns.set_context('notebook')
sns.pairplot(data[numeric_columns]);
```



In [15]:
```
## check is there na value
data.isna().sum()
```

Out[15]:
```
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
```

3. Clustering

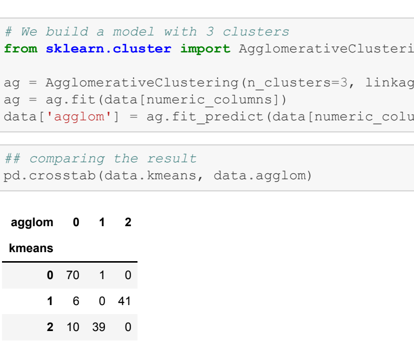
3.1 K-Means models

We cannot determine how many cluster are there. We would fit K-Means models with cluster values ranging from 1 to 10

In [16]:
```
from sklearn.cluster import KMeans
```

In [21]:
```
# Create and fit a range of models
km_list = list()

for clust in range(1,11):
    km = KMeans(n_clusters=clust, random_state=10)
    km = km.fit(data[numeric_columns])

    km_list.append(pd.Series({'clusters': clust,
                              'inertia': km.inertia_,
                              'model': km}))
```

In [22]:
```
plot_data = (pd.concat(km_list, axis=1)
             .T
             [['clusters','inertia']]
             .set_index('clusters'))

ax = plot_data.plot(marker='o',ls='-')
plt.plot(3,plot_data.loc[3],"ro")
ax.set_xticks(range(1,11,1))
ax.set_xlim(0,11)
ax.set(xlabel='Cluster', ylabel='Inertia');
```



From the elbow method, we found that cluster the data into 3 groups may be good.

In [23]:
```
data['kmeans'] = km_list[2].model.predict(data[numeric_columns])
```

3.2 Hierarchical clustering

In [24]:
```
# We build a model with 3 clusters
from sklearn.cluster import AgglomerativeClustering

ag = AgglomerativeClustering(n_clusters=3, linkage='ward', compute_full_tree=True)
ag = ag.fit(data[numeric_columns])
data['agglom'] = ag.fit_predict(data[numeric_columns])
```

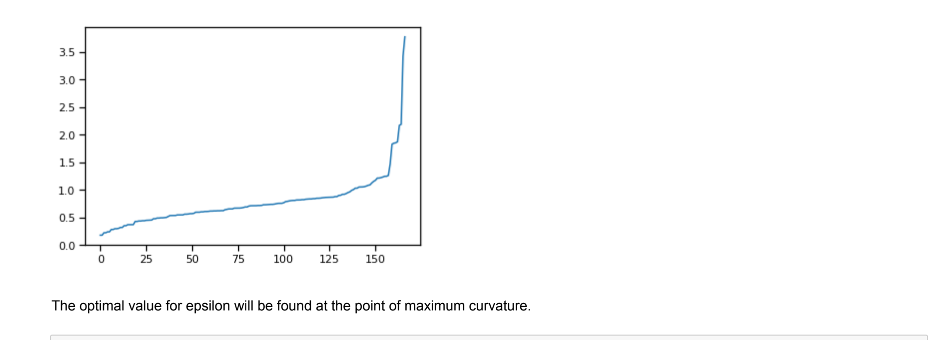
In [25]:
```
## comparing the result
pd.crosstab(data.kmeans, data.agglom)
```

Out[25]:

agglom	0	1	2
kmeans			
0	70	1	0
1	6	0	41
2	10	39	0

Create the dendrogram from agglomerative clustering.

In [26]:
```
from scipy.cluster import hierarchy

Z = hierarchy.linkage(ag.children_, method='ward')

fig, ax = plt.subplots(figsize=(15,5))

den = hierarchy.dendrogram(Z, orientation='top',
                           p=30, truncate_mode='lastp',
                           show_leaf_counts=True, ax=ax)
```



3.3 Density-Based Spatial Clustering of Applications with Noise.

In [39]:
```
## We compute the euclidian distance between every two records

from scipy.spatial.distance import cdist

all_distance=pd.DataFrame(cdist(data[numeric_columns], data[numeric_columns], 'euclid'))
```

We would like to determine the optimal eps. We can calculate the distance from each point to its closest neighbour using the NearestNeighbors.

In [140]:
```
from sklearn.neighbors import NearestNeighbors

neigh = NearestNeighbors(n_neighbors=2)

nbrs = neigh.fit(data[numeric_columns])

distances, indices = nbrs.kneighbors(data[numeric_columns])
```

In [142]:
```
## we sort and plot results
distances = np.sort(distances, axis=0)

distances = distances[:,1]

plt.plot(distances)
```

Out[142]:
```
[<matplotlib.lines.Line2D at 0x19560c10>]
```

The optimal value for epsilon will be found at the point of maximum curvature.

In [211]:
```
## We train our model, selecting 2.1 for eps and setting min_samples to 5.
from sklearn.cluster import DBSCAN
dbscan=DBSCAN(eps=2.1, min_samples=5).fit(data[numeric_columns])
```

In [212]:
```
dbscan.labels_
```

Out[212]:
```
array([ 0,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,
        0,  1,  1, -1,  1,  1,  1, -1,  1,  0,  1,  1, -1,  0,  1,
        1,  1,  0, -1, -1,  1,  0,  1,  1,  1,  0, -1, -1,  1, -1,
       [illegible]
      dtype=int64)
```

In [214]:
```
## number of cluster
len(set(dbscan.labels_)) - (1 if -1 in dbscan.labels_ else 0)
```

Out[214]:
```
3
```

4. PCA and visualization

4.1 plot the data after dimension reduction

As we are not able to visualize the dimensions higher than 3d. We would reduce the dimension of data for visualization of clustering.

In [112]:
```
from sklearn.decomposition import PCA

PCAmod = PCA(n_components=2)
reduced_data=pd.DataFrame(PCAmod.fit_transform(data[numeric_columns]), columns=["PC1", "PC2"])
```

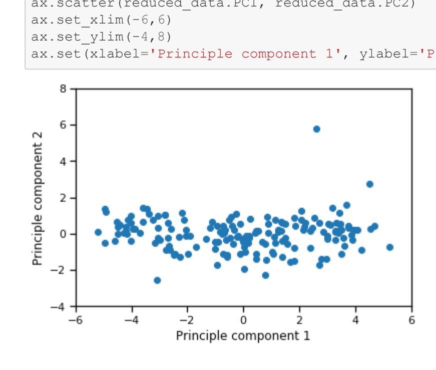
In [113]:
```
## Reduce the dimension to 2
reduced_data
```

Out[113]:

	PC1	PC2
0	-4.207070	0.130489
1	0.457792	0.099568
2	-0.115430	-1.092748
3	-2.279833	-1.295628
4	1.873019	0.421531
...	...	...
162	-1.101959	0.676816
163	0.760962	-2.254905
164	-0.890625	-0.476336
165	-2.448284	-1.094761
166	-3.167094	-0.530944

167 rows × 2 columns

In [124]:
```
## plot the data after dimension reduction
fig, ax=plt.subplots()
ax.scatter(reduced_data.PC1, reduced_data.PC2)
ax.set_xlim(-6,6)
ax.set_ylim(-6,6)
ax.set(xlabel="Principle component 1", ylabel="Principle component 2");
```

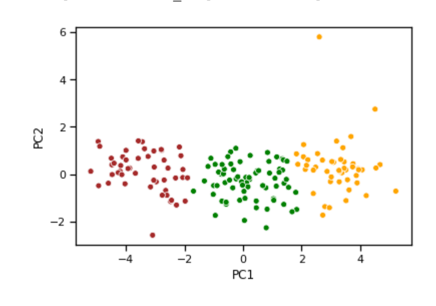


4.2 plot the data after dimension reduction with K-means labels

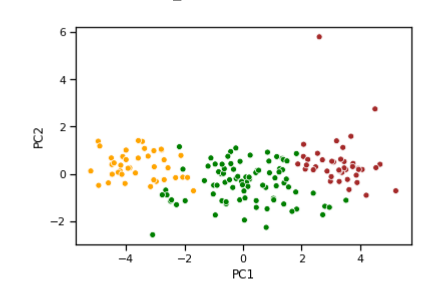
In [126]:
```
## create the column for clusters
reduced_data["kmeans"]=data["kmeans"]
```

In [18]:
```
import seaborn as sns

sns.scatterplot(data=reduced_data, x="PC1", y="PC2", hue="kmeans", legend=False, palette=["green","orange","brown"])
```

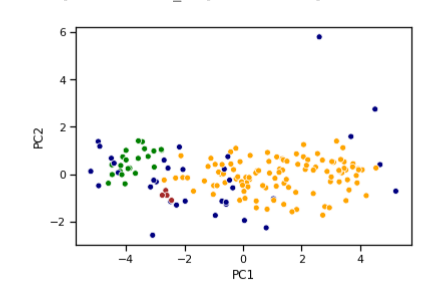
Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17e65e10>
```



4.3 plot the data after dimension reduction with agglomerative labels



4.4 plot the data after dimension reduction with DBSCAN labels



5. Result

As we can see that K-Mean and Agglomerative clustering generate very similar result. We generate two lists of country that may be positively or negatively influenced. We can find common country in these two lists that may be the countries need more help.

6. Discussion

Unlike k-means, DBSCAN will figure out the number of clusters. DBSCAN works by determining whether the minimum number of points are close enough to one another to be considered part of a single cluster. DBSCAN is very sensitive to scale since epsilon is a fixed value for the maximum distance between two points.

As we have only 9 columns of socioeconomic and health information. It may not well distinguish the different countries. More data may be collected to further segment the countries in different clusters.

In [ ]: