KMeans

Target:- The assignment targets to implement K-Means and K-Medoid algorithms to cluster the dataset consists of socio-economic and health factors of countries and determine the overall development of the country

Starting of the program

Imported all important libraries numpy,pandas,random to generate random initial centroids,seaborn for graphing correlation values,matplotlib for plotting final graphs of clusters,KMeans from sklearn just to measure my accuracy

In [1]:
```
import numpy as np
import pandas as pd
import random
import math
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import silhouette_score
```

Data Cleaning

set na_values to na,N/a,string type and checked for errors found no errors

In [2]:
```
def data_cleaning():
    invalid_cells=['na','N/a',np.nan]
    df=pd.read_csv('C:\Python\Assignments\MLAssignment\Country_data.csv',
                   na_values=invalid_cells)
    df.isnull()
    df.set_index("country",
              inplace = True)
    return df
    pass
df=data_cleaning()
df.isnull().sum()
```

Out[2]:
```
child_mort    0
exports       0
health        0
imports       0
income        0
inflation     0
life_expec    0
total_fer     0
gdpp          0
dtype: int64
```

Creating usefull lists

created list of colors for using later while marking created cluster list to map clusters initial number of clusters were taken as 3 as given in assignment target was to cluster data into developped developping underdevelopped

In [3]:
```
colors=['fuchsia','aqua','gold','brown','g','black','r','b','orange','pink']
cluster=list('abcdefghijklmnop')
number_of_clusters=3
```

Scaling all of columns using MinMaxScaler

since gdpp has higher values squared difference of those values will be given heigher weightage automatically so scaling is done using MinMaxScaler from sklearn.preprocessing

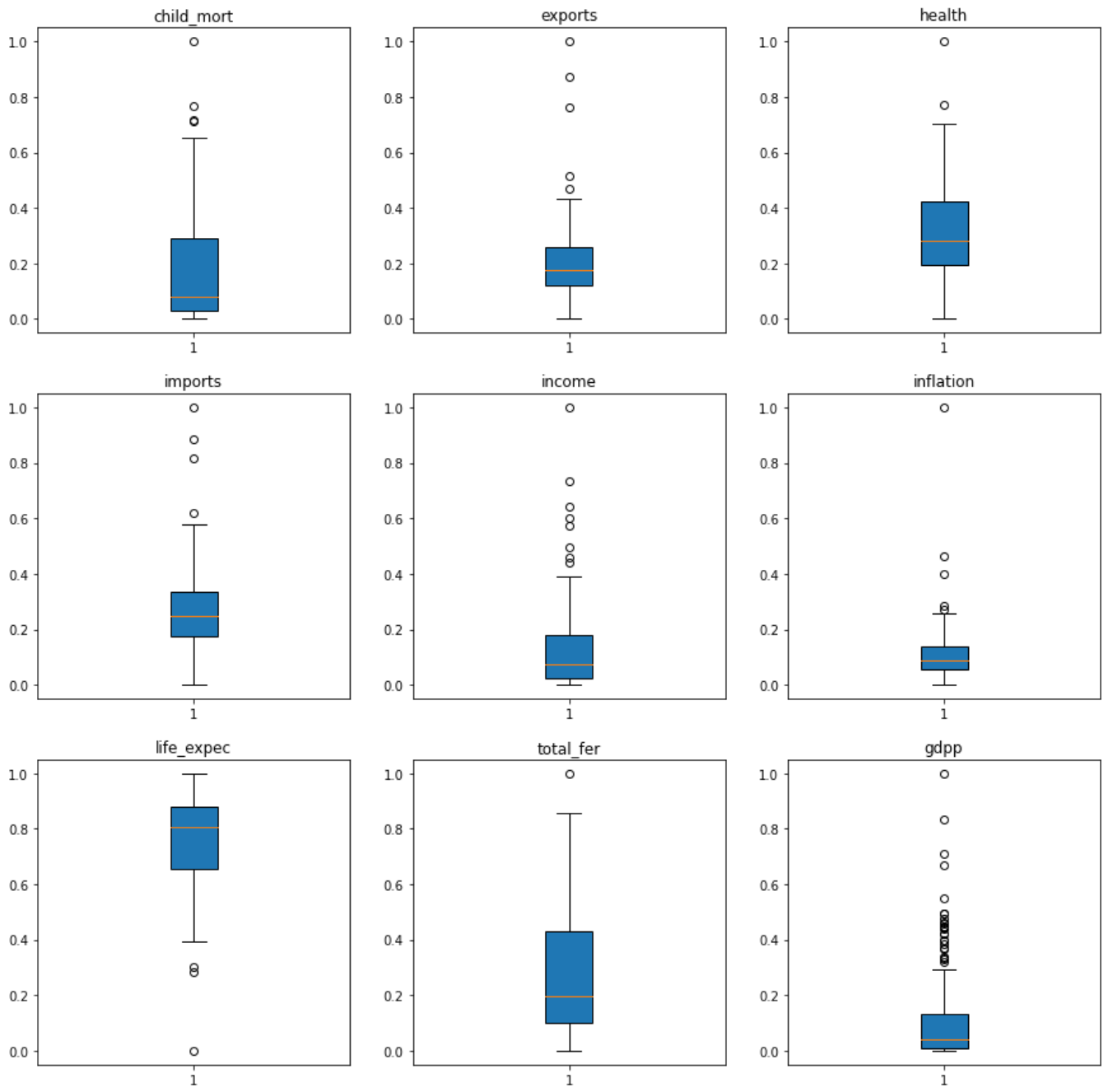
In [4]:
```
def scale_data():
    scaler=MinMaxScaler()
    for x in df.columns[0:9]:
        scaler.fit(np.array(df[x]).reshape(-1,1))
        df[x]=scaler.transform(np.array(df[x]).reshape(-1,1))
        df[x].round(2)
scale_data()
```

Creating boxplot

for visualizing outliers main figure of 15,15 size with 9 subplots for each feature

In [5]:
```
def boxPlot():
    ax=np.array([])
    fig, ax=plt.subplots(3,3,figsize=(15,15))
    ax=ax.reshape(1,9)


    for i in range(9):
        ax[0][i].boxplot(df[df.columns[i]],patch_artist=True)
        ax[0][i].set_title(df.columns[i])
    ax=ax.reshape(3,3)
    plt.show()
    plt.clf()
boxPlot()
```

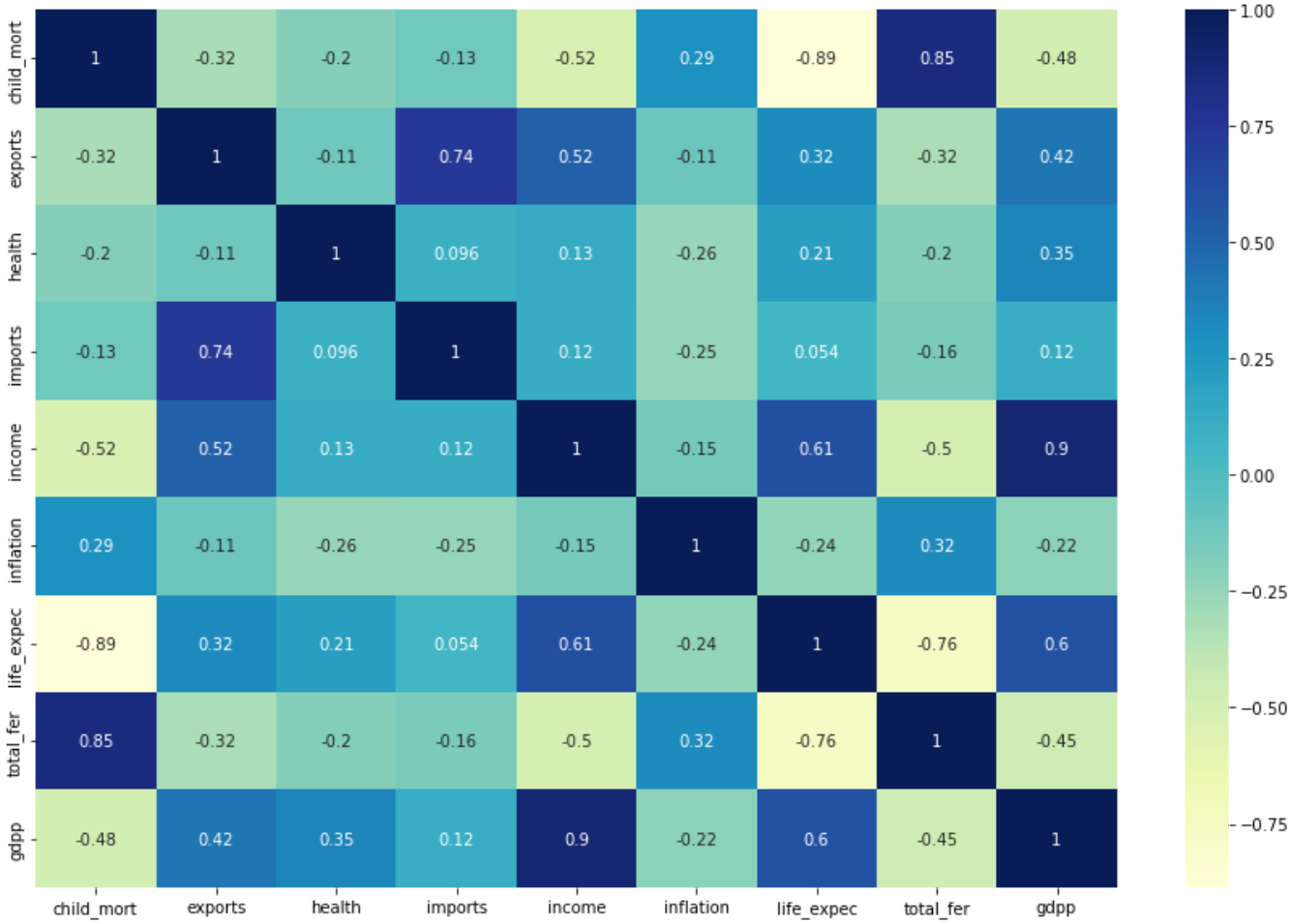



```
<Figure size 432x288 with 0 Axes>
```

Creating Correlation matrix

Seaborn was used to visualize correlation matrix instead of matplotlib because its taking lot of much time in matplotlib

In [6]:
```
def correlation_of_data():
    a=df[df.columns[0:9]].corr()
    plt.figure(figsize=(15,10))
    sns.heatmap(a,annot=True,cmap="YlGnBu")
    plt.show()
    plt.clf()
correlation_of_data()
```



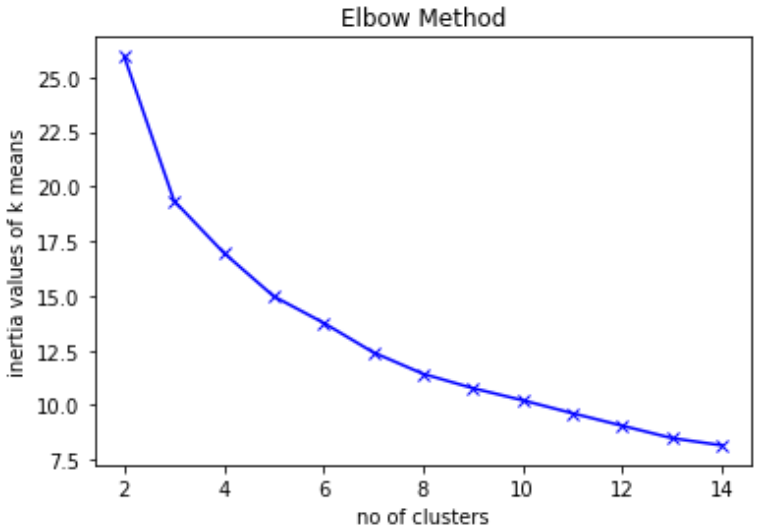

```
<Figure size 432x288 with 0 Axes>
```

Silhouetter score and Elbow Method

both silhouetter score and elbow drawings are plotted for different k values ranging from 2 to 15

In [7]:

```
def plot_scores():
    SSO={'silhouette_score':[],'elbow':[]}
    for i in range(2,15):
        kmeans=KMeans(n_clusters=i)
        a=kmeans.fit_predict(df[df.columns[0:9]])
        SSO['elbow'].append(kmeans.inertia_)
        SSO['silhouette_score'].append(silhouette_score(df[df.columns[0:9]],a))
    plt.title("Elbow Method")
    plt.plot(list(range(2,15)),SSO['elbow'],marker='x',color='b',label='elbow')
    plt.xlabel('no of clusters')
    plt.ylabel('inertia values of k means')
    plt.show()
    plt.clf()
    plt.title("Silhouette Score")
    plt.plot(list(range(2,15)),SSO['silhouette_score'],marker='x',
             color='r',label='silhouette_score')
    plt.xlabel('no of clusters')
    plt.ylabel('silhouette scores of k means')
    plt.show()
    plt.clf()
plot_scores()
```



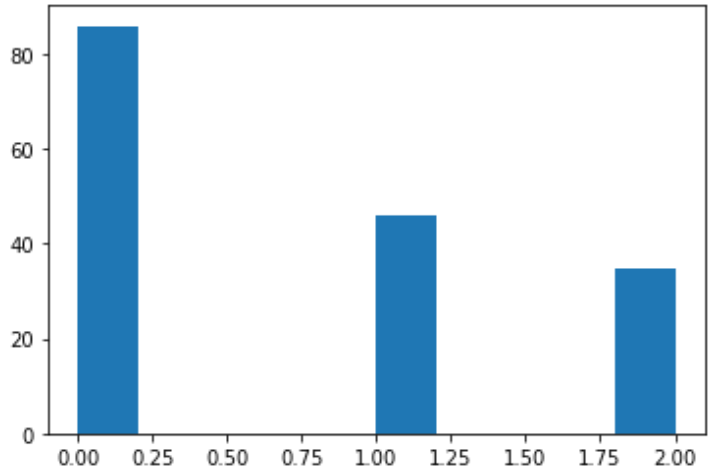

```
<Figure size 432x288 with 0 Axes>
```

Silhoutte Score for 3 clusters value is 0.335 for elbow method as we can see elbow point is at 3

Kmeans library

I have used Kmeans library just for checking if my final result is accurate or not

In [8]:
```
def kmeans_library():
    kmeans=KMeans(n_clusters=3)
    kmeans.fit(df[df.columns[0:9]])
    temp=kmeans.predict(df[df.columns[0:9]])
    centroid=kmeans.cluster_centers_.round(2)
    df['clusters']=temp
    plt.hist(df['clusters'])
    plt.show()
    del(df['clusters'])
kmeans_library()
```



Creating random centroid

created random centroids using random.sample method with a nparray of values from 0 to 1 of total 167 created using linspace

In [9]:
```
centroid_created =np.zeros((3,9))
def create_random_centroid():
    centroid=[]
    for i in range(number_of_clusters):
        centroid.append(random.sample(list(np.linspace(0,1,167).round(2)),9))
    global centroid_created
    centroid_created=np.array(centroid)
    print('creating new centroid \n')
create_random_centroid()
print(centroid_created)
```

```
creating new centroid

[[0.43 0.2  0.45 0.96 0.33 0.02 0.04 0.62 0.57]
 [0.19 0.7  0.35 0.8  0.05 0.42 0.48 0.92 0.24]
 [0.48 0.98 0.33 0.81 0.61 0.63 0.59 0.72 0.08]]
```

Adding new columns

added new columns to data frame containing euclidean distance calculated from each centre point to each country

In [10]:
```
def add_new_columns():
    for i in range(number_of_clusters):
        df[cluster[i]]=np.sqrt(sum((df[x]-centroid_created[i][ind])**2 for ind,
                                   x in enumerate(df.columns[0:9]))).round(2)
    df['min values']=df.iloc[:,9:(9+number_of_clusters)].idxmin(axis=1)
add_new_columns()
df[df.columns[9:]]
```

Out[10]:

country	a	b	c	min values
Afghanistan	1.08	0.98	1.35	b
Albania	1.40	1.27	1.50	b
Algeria	1.41	1.17	1.39	b
Angola	1.18	0.88	1.15	b
Antigua and Barbuda	1.32	1.17	1.39	b
...	...	...	...	...
Vanuatu	1.17	0.99	1.32	b
Venezuela	1.48	1.23	1.41	b
Vietnam	1.29	1.05	1.27	b
Yemen	1.25	0.98	1.28	b
Zambia	1.10	0.94	1.29	b

167 rows × 4 columns

Creating centroids

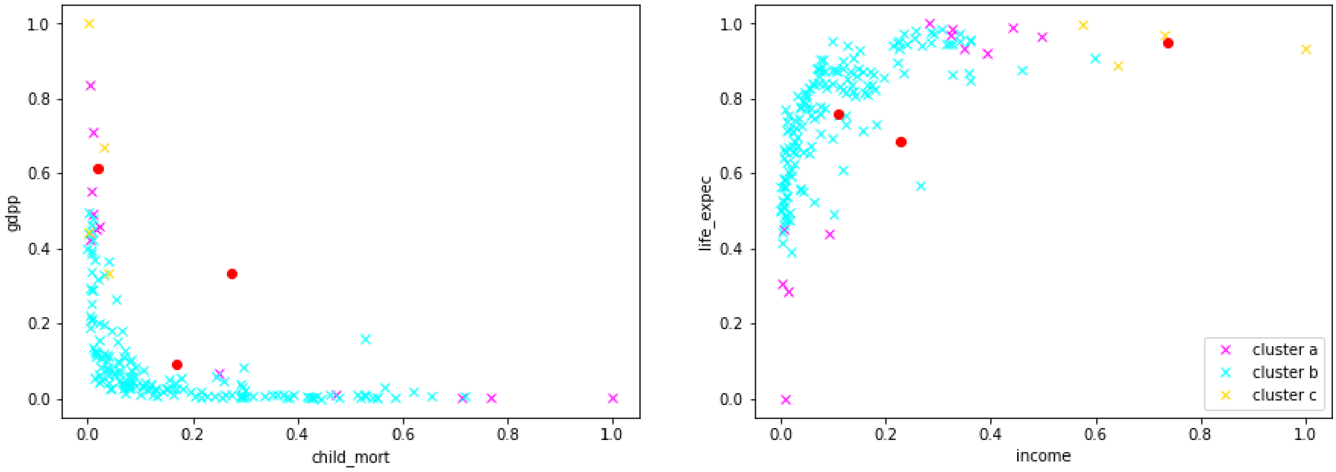
centroids were created using mean of selected clusters min values column contains cluster names for each country along with creating move_centroids method for creating new centroids each time it will also plot data along with three centroids to show changes in centroid movement for better visualisation of how kmeans works

For plotting i have used child_mort vs gdpp because they both have a correlation value of -48 meaning if more gdpp means less child_mort implying a developped nation Another subplot was made with income mort and life_expec

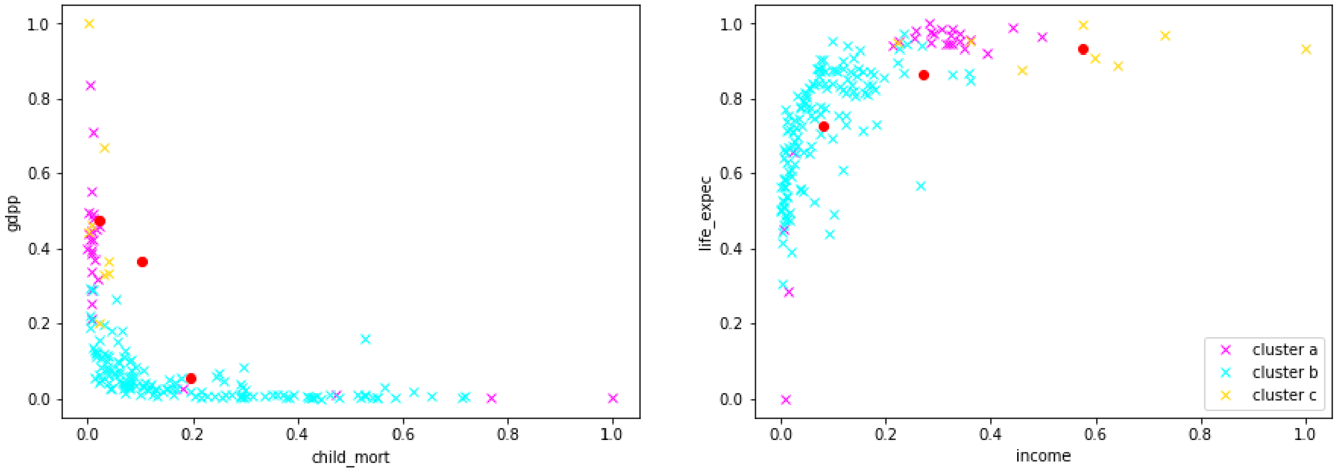
it will keep on recursing untill change in centroid becomes 0

In [11]:
```
old_centroid=None
def move_centroids():
    global centroid_created,old_centroid
    for i in range(number_of_clusters):
        centroid_created[i]=df[df['min values']==cluster[i]][df.columns[0:9]].mean(axis=0)
        if math.isnan(list(centroid_created[i])[0]):
            print("  initial centroid guessed is too far away creating new random centroid")
            create_random_centroid()
            add_new_columns()
    axs=[]
    fig, axs=plt.subplots(1,2,figsize=(15,5))
    for k in range(number_of_clusters):
        axs[0].plot(df[df['min values']==cluster[k]]['child_mort'],
                    df[df['min values']==cluster[k]]['gdpp'],'x',color=colors[k],
                    label=f'cluster {cluster[k]}')
        axs[1].plot(df[df['min values']==cluster[k]]['income'],
                    df[df['min values']==cluster[k]]['life_expec'],'x',color=colors[k],
                    label=f'cluster {cluster[k]}')
    axs[0].plot(centroid_created[:,0],centroid_created[:,8],'o',color='r')
    axs[1].plot(centroid_created[:,4],centroid_created[:,6],'o',color='r')
    axs[0].set_xlabel('child_mort')
    axs[0].set_ylabel('gdpp')
    axs[1].set_xlabel('income')
    axs[1].set_ylabel('life_expec')
    plt.legend()
```

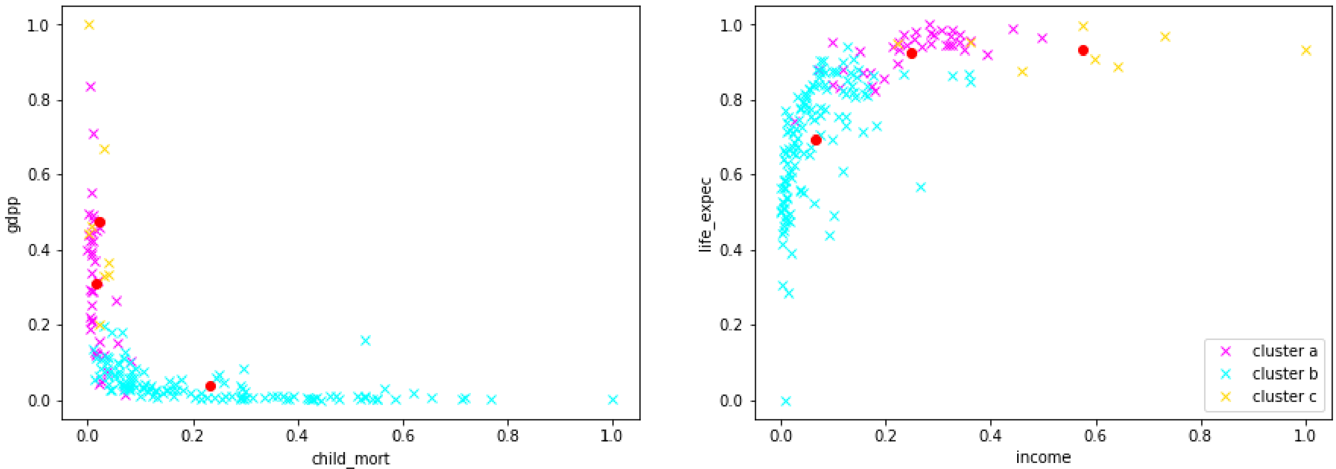
```
    plt.show()
    plt.clf()
    add_new_columns()
    if str(centroid_created)!=old_centroid:
        old_centroid=str(centroid_created)
        move_centroids()
    else:
        return
move_centroids()
```



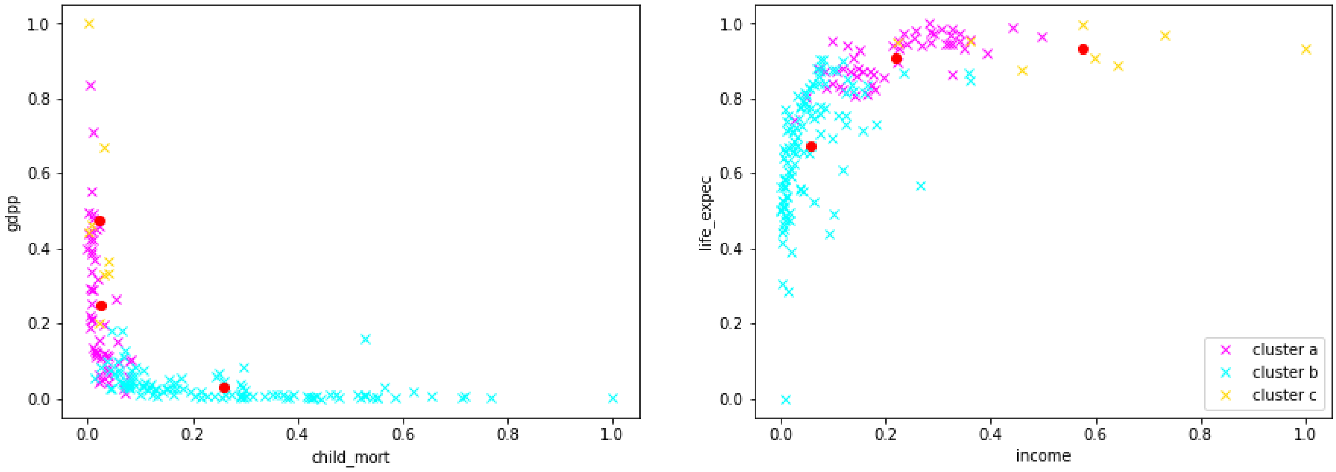
<Figure size 432x288 with 0 Axes>



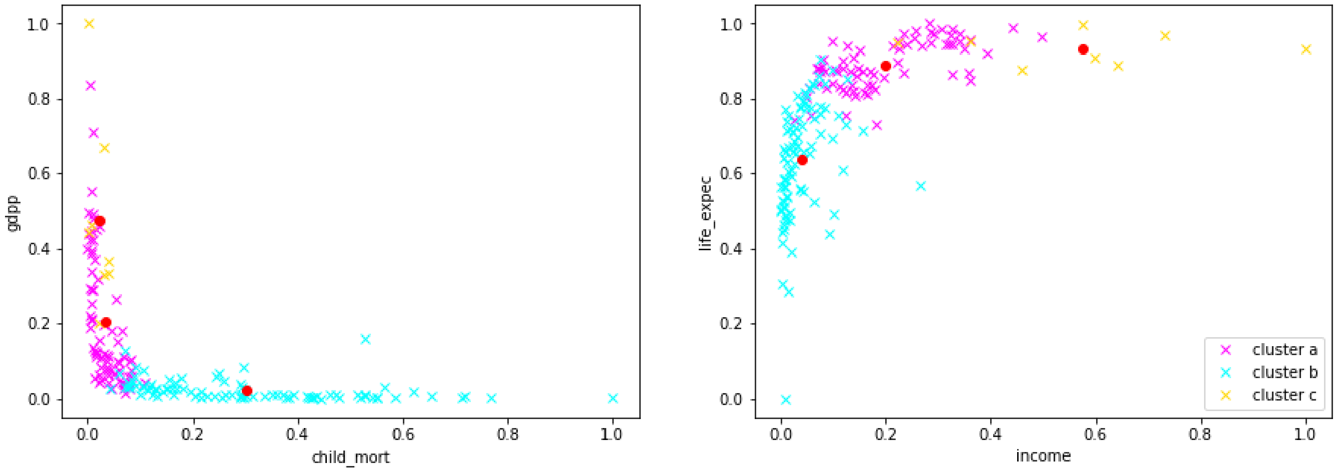
<Figure size 432x288 with 0 Axes>



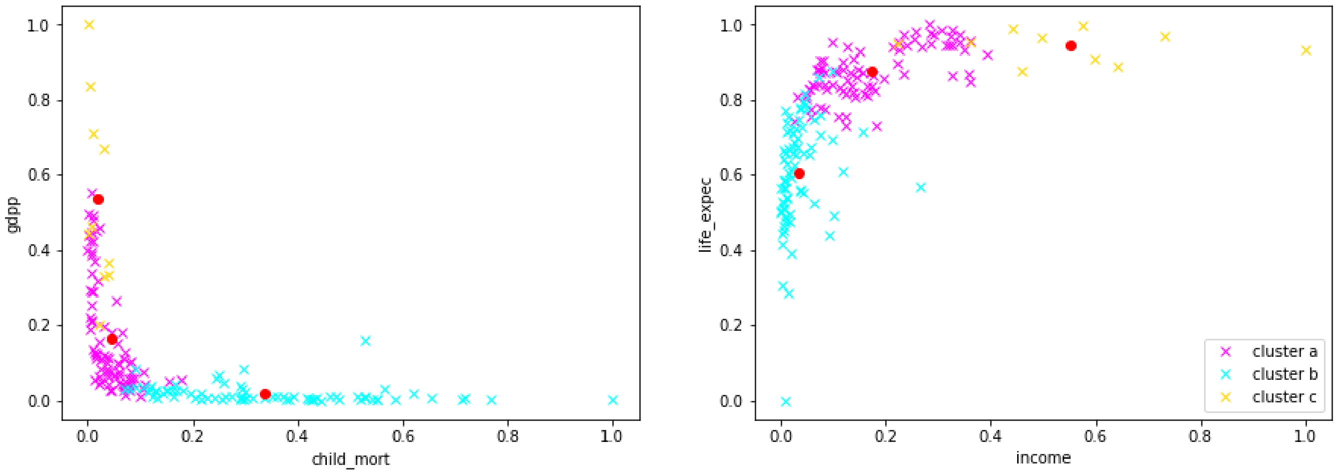
<Figure size 432x288 with 0 Axes>



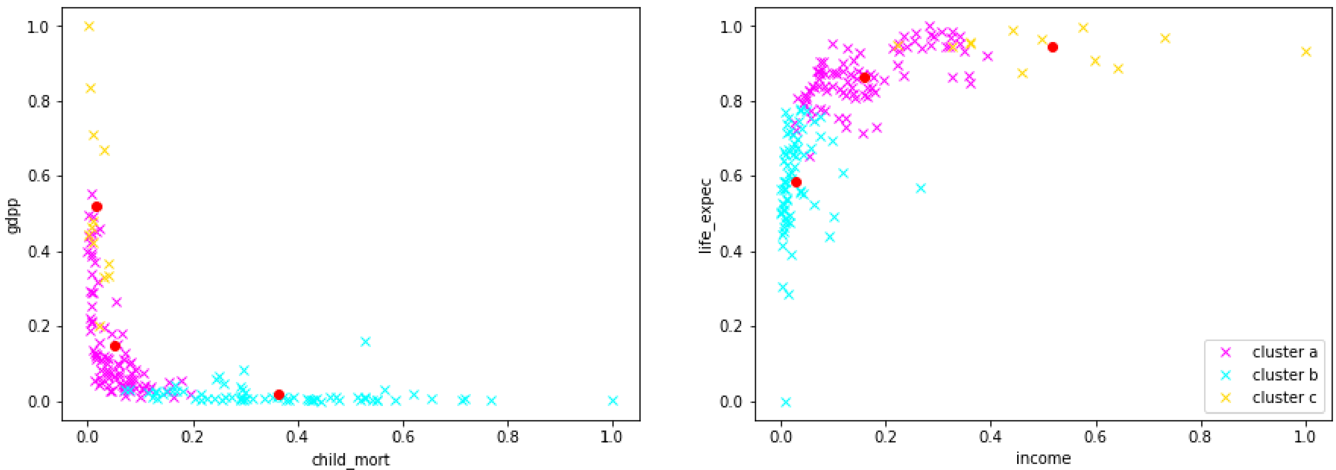
<Figure size 432x288 with 0 Axes>



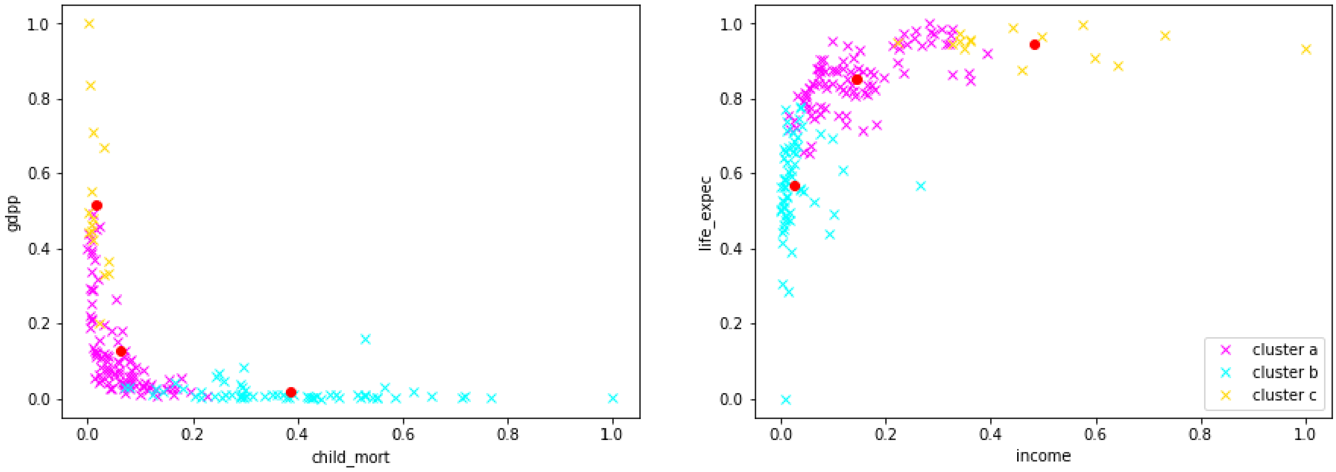
<Figure size 432x288 with 0 Axes>



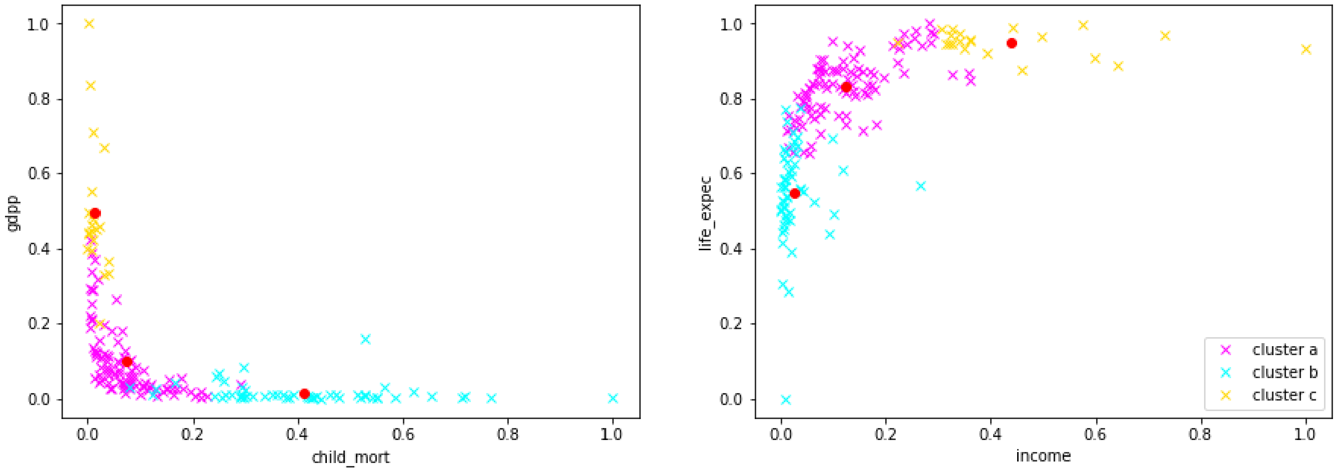
<Figure size 432x288 with 0 Axes>



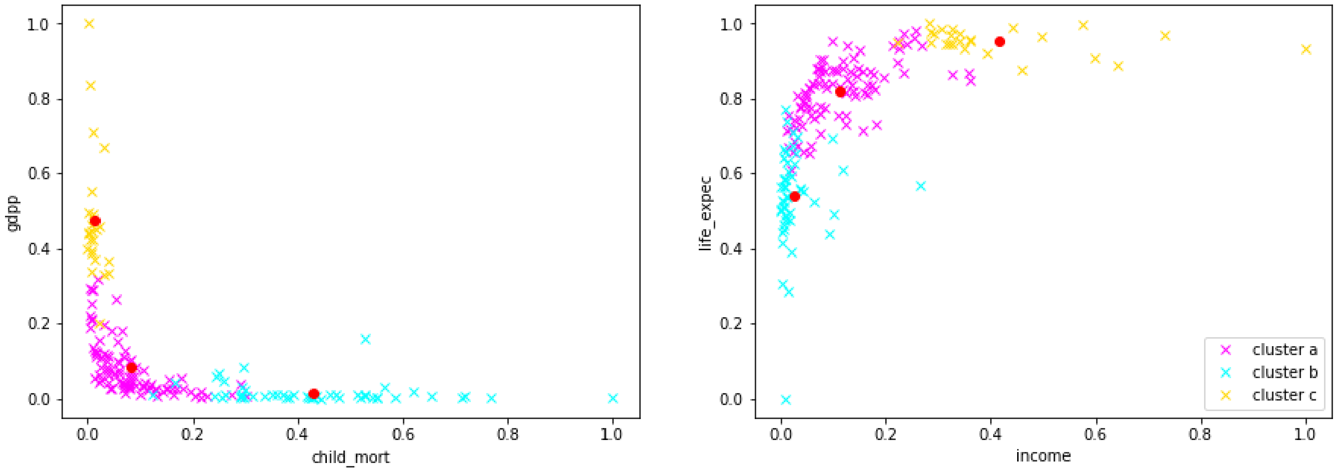
<Figure size 432x288 with 0 Axes>



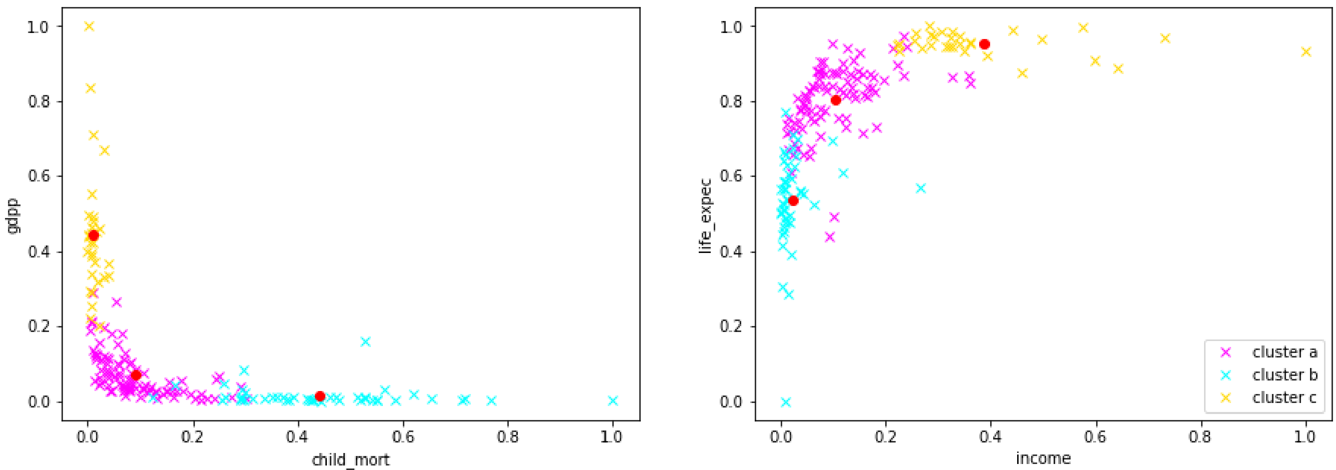
<Figure size 432x288 with 0 Axes>



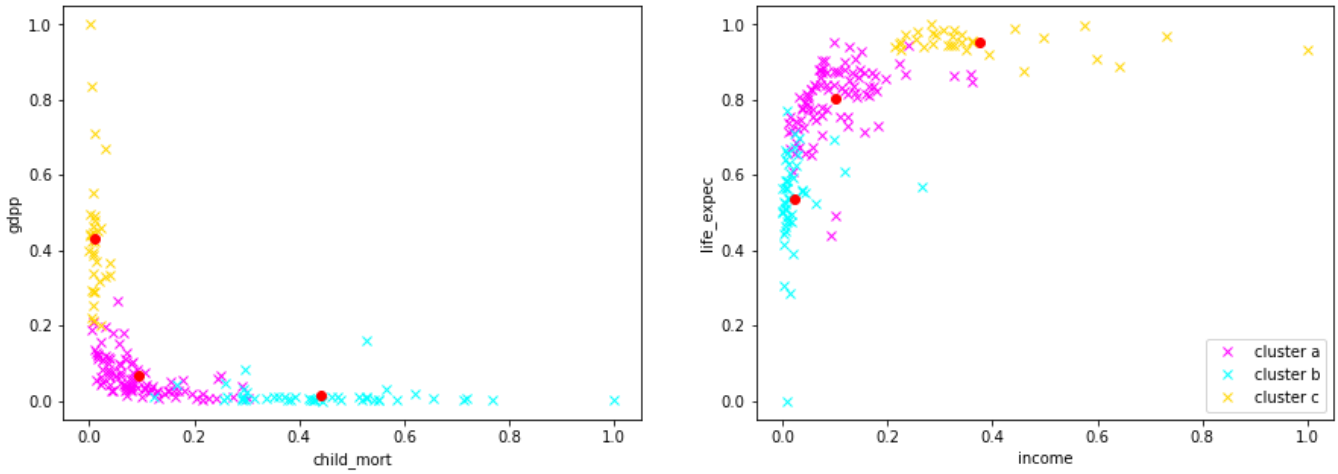
<Figure size 432x288 with 0 Axes>



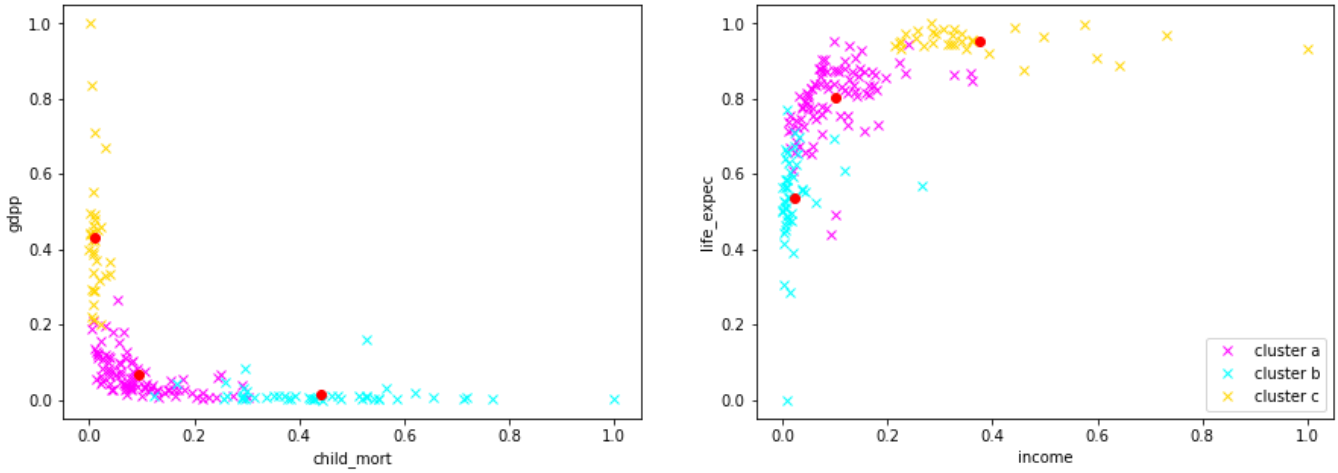
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>




```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

Final conlusion

cluster c has high child_mort, very low life_expec, income and gdpp with all other coinsiding features i will consider it as underdevelopped

cluster b has Normal child_mort, Normal Low life_expec, income and gdpp along with all other coinsiding deatuers i will considered it as developping

cluster a has Low child_mort, High life_expec, income and gdpp along with all other coinsiding deatuers i will considered it as developped

In [18]:
```
df['Nation status']=' '
df.loc[df['min values']=='b','Nation status']='underdevelopped'
df.loc[df['min values']=='a','Nation status']='developping'
df.loc[df['min values']=='c','Nation status']='developped'
display(df['Nation status'])
```

```
0      underdevelopped
1          developping
2          developping
3      underdevelopped
4          developping
              ...
162        developping
163        developping
164        developping
165    underdevelopped
166    underdevelopped
Name: Nation status, Length: 167, dtype: object
```

In [43]:
```
np.array(df[df['Nation status']=='underdevelopped']['country'])
```

Out[43]:
```
array(['Afghanistan', 'Angola', 'Benin', 'Burkina Faso', 'Burundi',
       'Cameroon', 'Central African Republic', 'Chad', 'Comoros',
       'Congo, Dem. Rep.', 'Congo, Rep.', "Cote d'Ivoire",
       'Equatorial Guinea', 'Eritrea', 'Gabon', 'Gambia', 'Ghana',
       'Guinea', 'Guinea-Bissau', 'Haiti', 'Iraq', 'Kenya', 'Kiribati',
       'Lao', 'Lesotho', 'Liberia', 'Madagascar', 'Malawi', 'Mali',
       'Mauritania', 'Mozambique', 'Namibia', 'Niger', 'Nigeria',
       'Pakistan', 'Rwanda', 'Senegal', 'Sierra Leone', 'Solomon Islands',
       'Sudan', 'Tanzania', 'Timor-Leste', 'Togo', 'Uganda', 'Yemen',
       'Zambia'], dtype=object)
```

In [21]:
```
df.reset_index(inplace=True)
np.array(df[df['Nation status']=='developping']['country'])
```

Out[21]:
```
array(['Albania', 'Algeria', 'Antigua and Barbuda', 'Argentina',
       'Armenia', 'Azerbaijan', 'Bahamas', 'Bahrain', 'Bangladesh',
       'Barbados', 'Belarus', 'Belize', 'Bhutan', 'Bolivia',
       'Bosnia and Herzegovina', 'Botswana', 'Brazil', 'Bulgaria',
       'Cambodia', 'Cape Verde', 'Chile', 'China', 'Colombia',
       'Costa Rica', 'Croatia', 'Czech Republic', 'Dominican Republic',
       'Ecuador', 'Egypt', 'El Salvador', 'Estonia', 'Fiji', 'Georgia',
       'Grenada', 'Guatemala', 'Guyana', 'Hungary', 'India', 'Indonesia',
       'Iran', 'Jamaica', 'Jordan', 'Kazakhstan', 'Kyrgyz Republic',
       'Latvia', 'Lebanon', 'Libya', 'Lithuania', 'Macedonia, FYR',
       'Malaysia', 'Maldives', 'Mauritius', 'Micronesia, Fed. Sts.',
       'Moldova', 'Mongolia', 'Montenegro', 'Morocco', 'Myanmar', 'Nepal',
       'Oman', 'Panama', 'Paraguay', 'Peru', 'Philippines', 'Poland',
       'Romania', 'Russia', 'Samoa', 'Saudi Arabia', 'Serbia',
       'Seychelles', 'Slovak Republic', 'South Africa', 'South Korea',
       'Sri Lanka', 'St. Vincent and the Grenadines', 'Suriname',
       'Tajikistan', 'Thailand', 'Tonga', 'Tunisia', 'Turkey',
       'Turkmenistan', 'Ukraine', 'Uruguay', 'Uzbekistan', 'Vanuatu',
       'Venezuela', 'Vietnam'], dtype=object)
```

In [44]:
```
np.array(df[df['Nation status']=='developped']['country'])
```

Out[44]:
```
array(['Australia', 'Austria', 'Belgium', 'Brunei', 'Canada', 'Cyprus',
       'Denmark', 'Finland', 'France', 'Germany', 'Greece', 'Iceland',
       'Ireland', 'Israel', 'Italy', 'Japan', 'Kuwait', 'Luxembourg',
       'Malta', 'Netherlands', 'New Zealand', 'Norway', 'Portugal',
       'Qatar', 'Singapore', 'Slovenia', 'Spain', 'Sweden', 'Switzerland',
       'United Arab Emirates', 'United Kingdom', 'United States'],
      dtype=object)
```

In [38]:
```
dfcopy=pd.read_csv('C:\Python\Assignments\MLAssignment\Country_data.csv')
dfcopy['Kmeans Nation status']=df['Nation status']
dfcopy.to_csv('C:\Python\Assignments\MLAssignment\country_data_Kmeans')
dfcopy
```

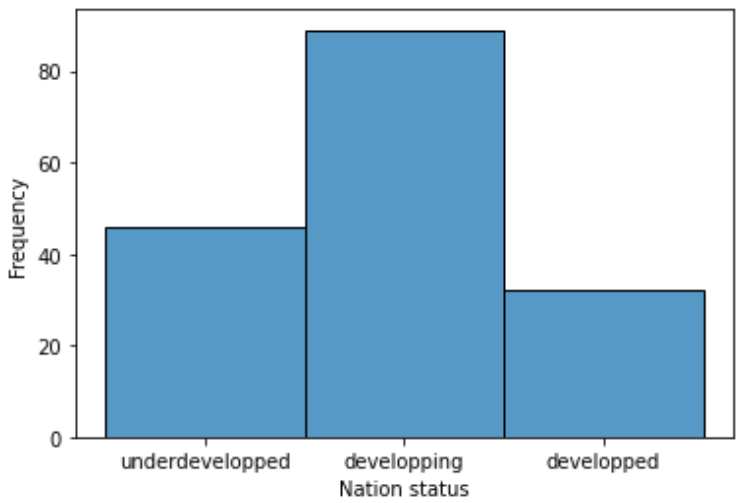
Out[38]:

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	Kmeans
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553	underdeve
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090	deve
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460	deve
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530	underdeve
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200	deve
...	...	...	...	...	...	...	...	...	...	...	
162	Vanuatu	29.2	46.6	5.25	52.7	2950	2.62	63.0	3.50	2970	deve
163	Venezuela	17.1	28.5	4.91	17.6	16500	45.90	75.4	2.47	13500	deve
164	Vietnam	23.3	72.0	6.84	80.2	4490	12.10	73.1	1.95	1310	deve
165	Yemen	56.3	30.0	5.18	34.4	4480	23.60	67.5	4.67	1310	underdeve
166	Zambia	83.1	37.0	5.89	30.9	3280	14.00	52.0	5.40	1460	underdeve

167 rows × 11 columns

In [35]:
```
sns.histplot(df['Nation status'],stat='frequency')
# plt.bar_label()
```

Out[35]:
```
<AxesSubplot:xlabel='Nation status', ylabel='Frequency'>
```



In [ ]: