Kmeans_task

June 28, 2024

0.1 Categorising countries

0.1.1 Data Source

The data used in this task was orginally sourced from Help.NGO. This international non-governmental organisation specialises in emergency response, preparedness, and risk mitigation.

0.1.2 Dataset Attributes

- country: name of the country
- child_mort: death of children under 5 years of age per 1000 live births
- exports: exports of goods and services per capita. Given as a percentage of the GDP per capita
- health: total health spending per capita. Given as a percentage of GDP per capita
- imports: imports of goods and services per capita. Given as a percentage 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 remain the same
- total_fer: the number of children that would be born to each woman if the current age-fertility rates remains the same
- gdpp: the GDP per capita. Calculated as the Total GDP divided by the total population.

0.2 Objective

To group countries using socio-economic and health factors to determine the development status of the country.

[1]:
```
#%pip install yellowbrick
```

[2]:
```
# Import libraries
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings(action='ignore', category=FutureWarning)
import warnings
warnings.filterwarnings('ignore')
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score,  silhouette_samples
from sklearn.preprocessing import MinMaxScaler
from yellowbrick.cluster import SilhouetteVisualizer
```

[3]:
```python
# Random state seed
rseed = 42
```

0.3 Load and explore data

[4]:
```python
# Import the dataset
df = pd.read_csv('Country-data.csv')
data = pd.read_csv('Country-data.csv')
```

[5]:
```python
# Check the shape
df.shape
```

[5]: (167, 10)

[6]:
```python
# Check datatypes & counts
df.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
```

[7]:
```python
# Get descriptive statistics
df.describe()
```

[7]:
```
       child_mort     exports      health     imports         income  \
count  167.000000  167.000000  167.000000  167.000000     167.000000
mean    38.270060   41.108976    6.815689   46.890215   17144.688623
std     40.328931   27.412010    2.746837   24.209589   19278.067698
min      2.600000    0.109000    1.810000    0.065900     609.000000
25%      8.250000   23.800000    4.920000   30.200000    3355.000000
50%     19.300000   35.000000    6.320000   43.300000    9960.000000
75%     62.100000   51.350000    8.600000   58.750000   22800.000000
max    208.000000  200.000000   17.900000  174.000000  125000.000000

        inflation  life_expec   total_fer           gdpp
count  167.000000  167.000000  167.000000     167.000000
mean     7.781832   70.555689    2.947964   12964.155689
std     10.570704    8.893172    1.513848   18328.704809
min     -4.210000   32.100000    1.150000     231.000000
25%      1.810000   65.300000    1.795000    1330.000000
50%      5.390000   73.100000    2.410000    4660.000000
75%     10.750000   76.800000    3.880000   14050.000000
max    104.000000   82.800000    7.490000  105000.000000
```

[8]:
```
df.head()
```

[8]:
```
               country  child_mort  exports  health  imports  income  \
0          Afghanistan        90.2     10.0    7.58     44.9    1610
1              Albania        16.6     28.0    6.55     48.6    9930
2              Algeria        27.3     38.4    4.17     31.4   12900
3               Angola       119.0     62.3    2.85     42.9    5900
4  Antigua and Barbuda        10.3     45.5    6.03     58.9   19100

   inflation  life_expec  total_fer   gdpp
0       9.44        56.2       5.82    553
1       4.49        76.3       1.65   4090
2      16.10        76.5       2.89   4460
3      22.40        60.1       6.16   3530
4       1.44        76.8       2.13  12200
```

[9]:
```
# Identify any missing data
df.isnull().sum()
```

[9]:
```
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
```

0.4 Preprocessing and Feature Selection

[10]:
```
# Drop any non-numeric features (columns)
#df=df.select_dtypes(include='number')
df=df.select_dtypes(exclude= 'object')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   child_mort  167 non-null    float64
 1   exports     167 non-null    float64
 2   health      167 non-null    float64
 3   imports     167 non-null    float64
 4   income      167 non-null    int64
 5   inflation   167 non-null    float64
 6   life_expec  167 non-null    float64
 7   total_fer   167 non-null    float64
 8   gdpp        167 non-null    int64
dtypes: float64(7), int64(2)
memory usage: 11.9 KB
```

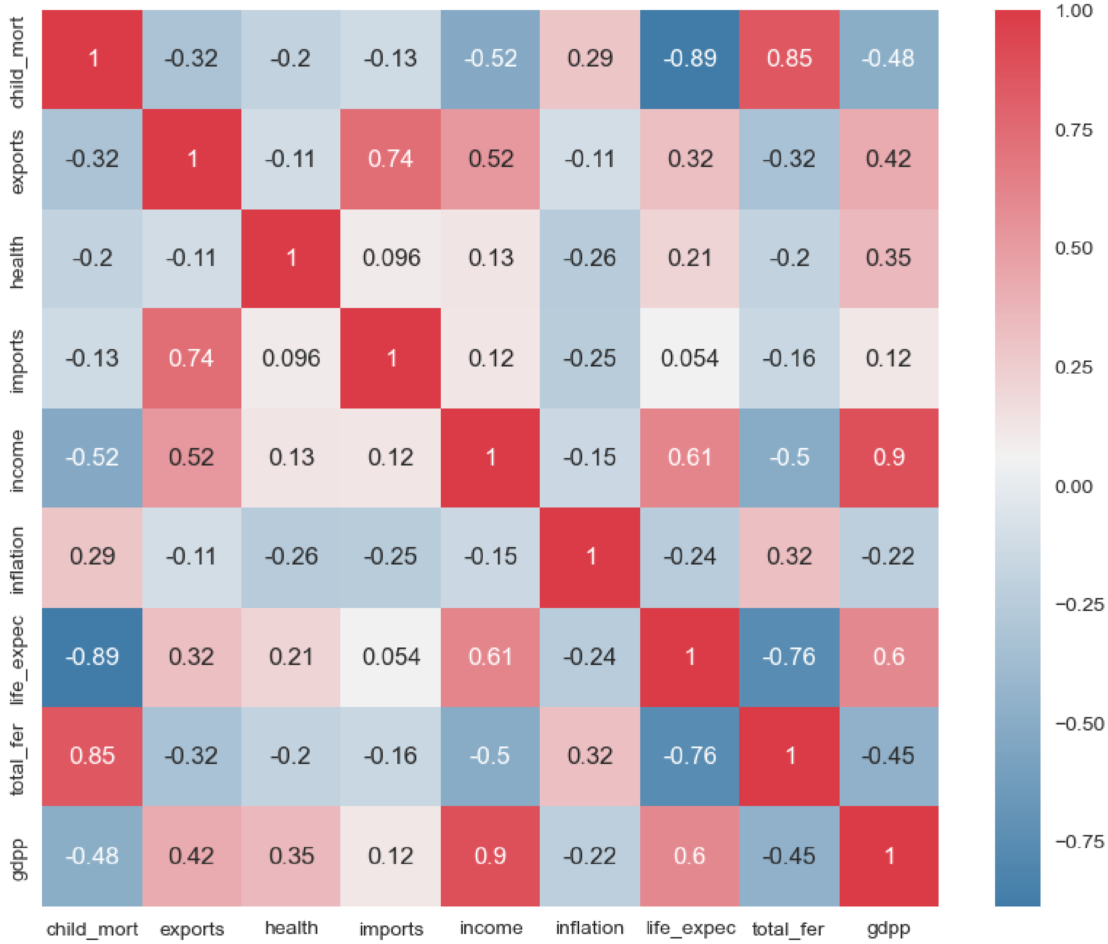
[11]:
```
# Create a correlation map of features to explore relationships between features
df.corr()
```

[11]:
```
            child_mort   exports    health   imports    income  inflation  \
child_mort    1.000000 -0.318093 -0.200402 -0.127211 -0.524315   0.288276
exports      -0.318093  1.000000 -0.114408  0.737381  0.516784  -0.107294
health       -0.200402 -0.114408  1.000000  0.095717  0.129579  -0.255376
imports      -0.127211  0.737381  0.095717  1.000000  0.122406  -0.246994
income       -0.524315  0.516784  0.129579  0.122406  1.000000  -0.147756
inflation     0.288276 -0.107294 -0.255376 -0.246994 -0.147756   1.000000
life_expec   -0.886676  0.316313  0.210692  0.054391  0.611962  -0.239705
total_fer     0.848478 -0.320011 -0.196674 -0.159048 -0.501840   0.316921
gdpp         -0.483032  0.418725  0.345966  0.115498  0.895571  -0.221631

            life_expec  total_fer      gdpp
child_mort   -0.886676   0.848478 -0.483032
exports       0.316313  -0.320011  0.418725
health        0.210692  -0.196674  0.345966
imports       0.054391  -0.159048  0.115498
income        0.611962  -0.501840  0.895571
```

```
inflation    -0.239705   0.316921 -0.221631
life_expec    1.000000  -0.760875  0.600089
total_fer    -0.760875   1.000000 -0.454910
gdpp          0.600089  -0.454910  1.000000
```

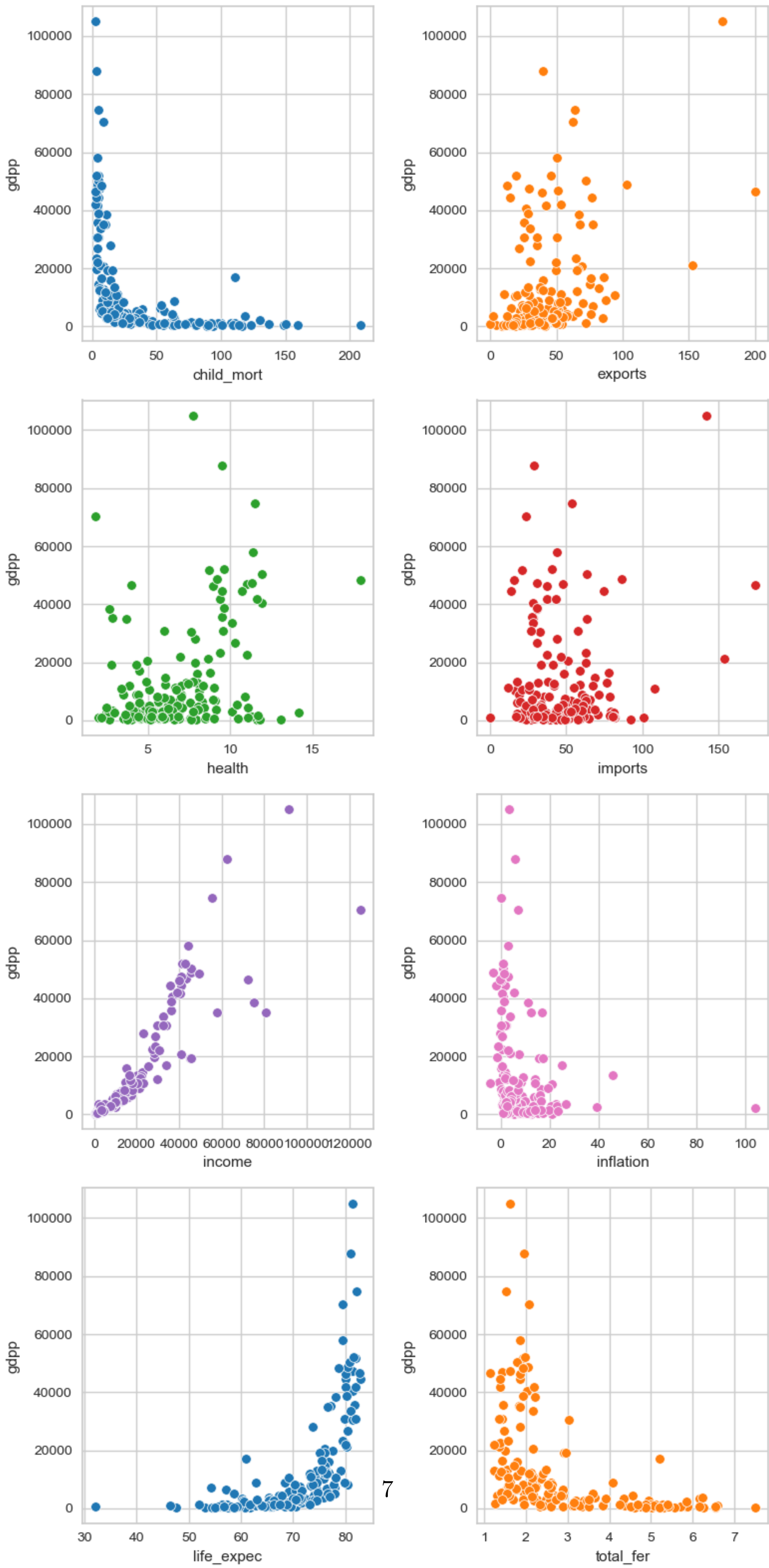
[12]:
```python
plt.figure(figsize = (15,5))
f, ax = plt.subplots(figsize=(10, 8))
corr = df.corr()
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool_), cmap=sns.
 ↪diverging_palette(240,10,as_cmap=True),
            square=True, ax=ax, annot=True)
```

[12]:
```
<Axes: >

<Figure size 1500x500 with 0 Axes>
```

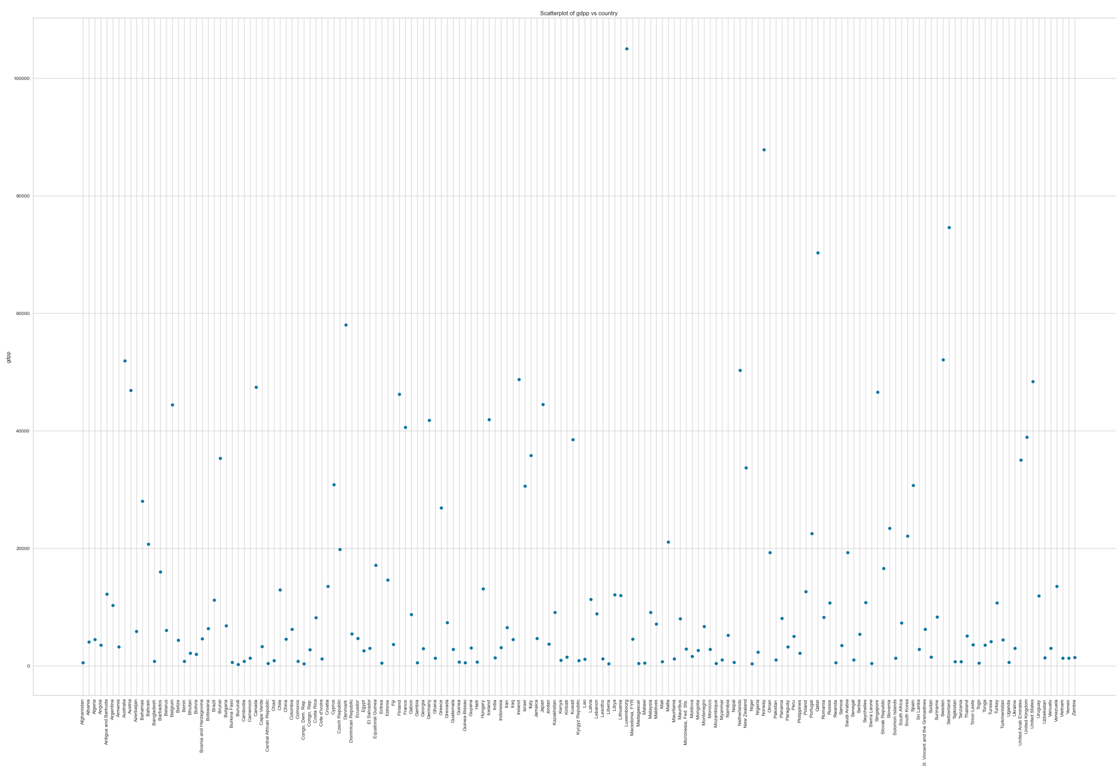


[13]:
```python
# function definition for scatterplot visualisation

def plot(array, y_var):
# define the figure and subplots
    fig, axes = plt.subplots(nrows=int(len(array)/2), ncols=2, figsize=(8, 16))

# array to 1D
    axes = axes.ravel()

# list of colours for each subplot, otherwise all subplots will be one color
    colours = [
        'tab:blue','tab:orange','tab:green','tab:red','tab:purple','tab:
 ↪pink','tab:blue','tab:orange'
              ]

    for col, colour, ax in zip(array, colours, axes):
        sns.scatterplot(x=col, y=y_var, data=df, color=colour, ax=ax)

    fig.tight_layout()
    plt.show()
```

[14]:
```python
# Explore the continuous independent features against gdpp using scatter plots.
gdpp = df.columns.tolist()
gdpp.remove('gdpp')
plot(gdpp,'gdpp')
```

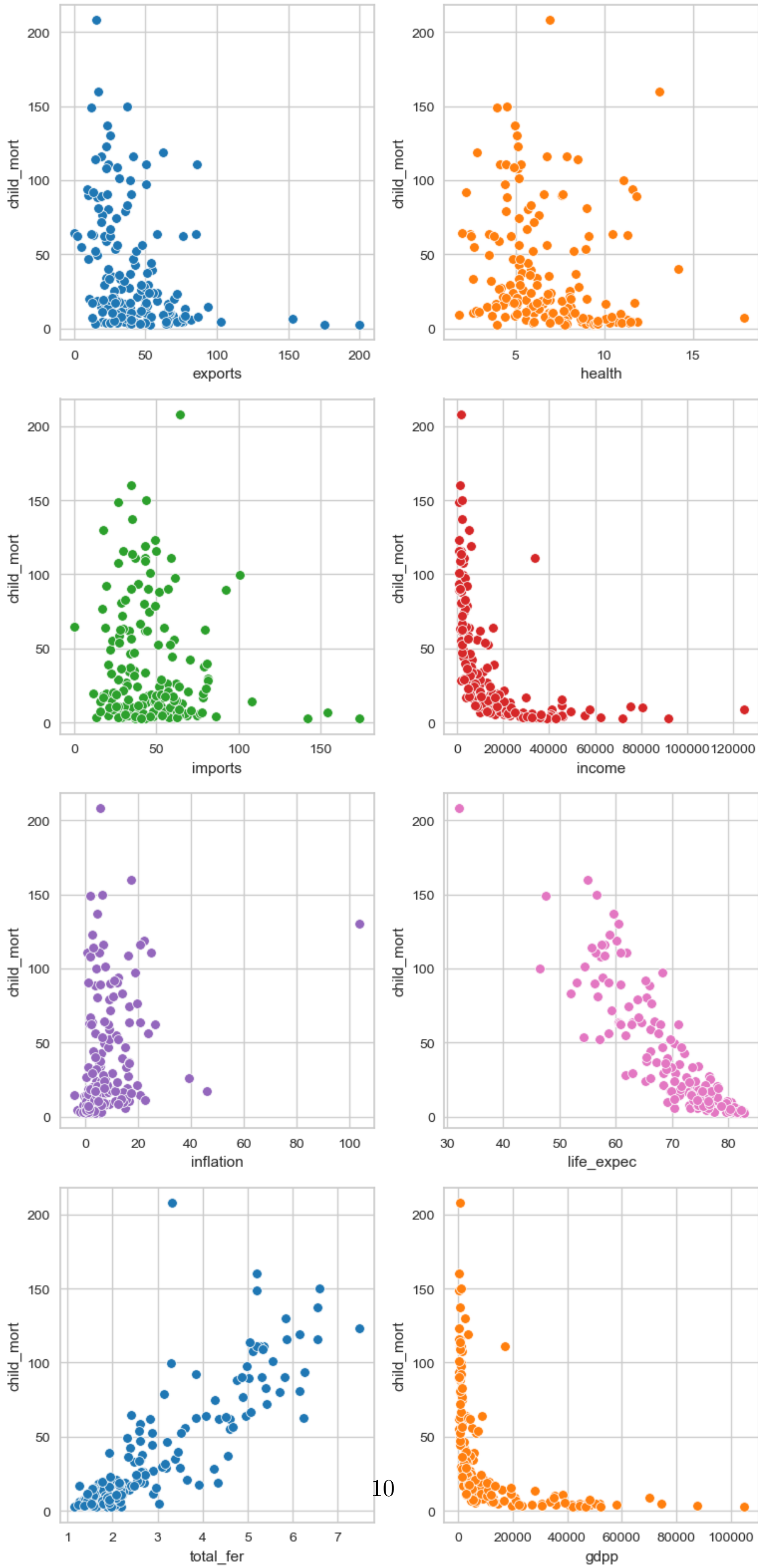

gdpp
child_mort
exports
health
imports
income
inflation
life_expec
total_fer

[15]:
```
# Since the question asks to plot 9 scatterplot, that means it is expected of␣
↪us to plot a scatterplot between gdpp
# and country. Since country name is unique in the dataset, I cant apply␣
↪groupby method on it. The only thing I can
# think of is to plot the graph like the way I have done it, so that y-label␣
↪ticks become bit clearer.

plt.figure(figsize = (40,25))
labels_plot= data['country']
g = sns.scatterplot(x=data.country, y=data.gdpp)
plt.title("Scatterplot of gdpp vs country")
g.set_xticklabels(labels= labels_plot, rotation=90)
# Show the plot
plt.show()
```



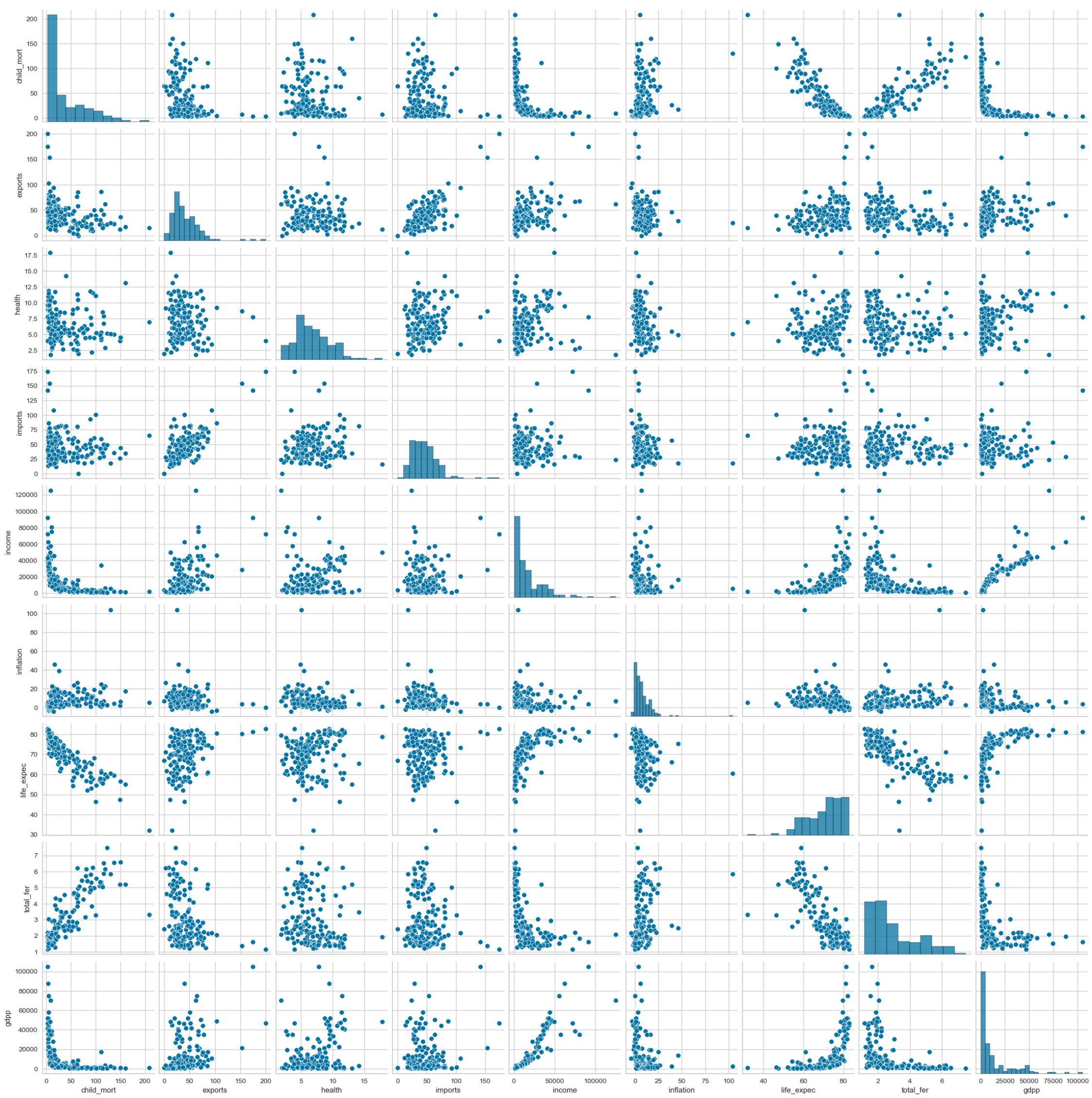
[29]:
```
# Explore the continuous independent features against chilkd_mort using scatter␣
↪plots.
child_mort = df.columns.tolist()
child_mort.remove('child_mort')
```

```
plot(child_mort,'child_mort')
```


child_mort
exports
health
imports
income
inflation
life_expec
total_fer
gdpp

[16]:
```
# pair plot
sns.pairplot(df)
```

[16]: <seaborn.axisgrid.PairGrid at 0x16aa9e3d0>



Note the peaks in the diagonal graphs that are distinct from each other or only overlap slightly. Looking at the scatter plot distributions may also give you some indication of features that would be good candidates for clustering the data.

0.4.1 Scaling the Data

[17]:
```
# Normalising the data using MinMaxScaler and naming the normalised dataframe␣
↪"df_scaled"

scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df.to_numpy())
df_scaled = pd.DataFrame(df_scaled, columns=df.columns.tolist())
print("Scaled Dataset Using MinMaxScaler")
df_scaled.head()
```

```
Scaled Dataset Using MinMaxScaler
```

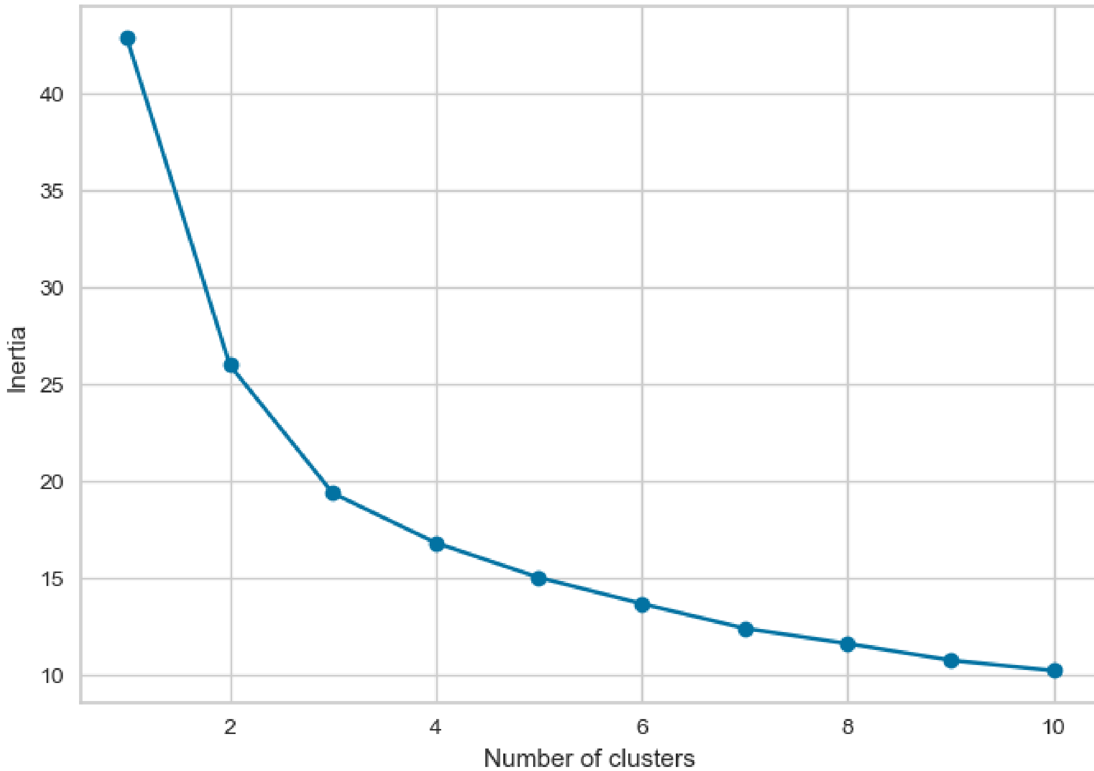
[17]:
```
   child_mort   exports    health   imports    income  inflation  life_expec  \
0    0.426485  0.049482  0.358608  0.257765  0.008047   0.126144    0.475345
1    0.068160  0.139531  0.294593  0.279037  0.074933   0.080399    0.871795
2    0.120253  0.191559  0.146675  0.180149  0.098809   0.187691    0.875740
3    0.566699  0.311125  0.064636  0.246266  0.042535   0.245911    0.552268
4    0.037488  0.227079  0.262275  0.338255  0.148652   0.052213    0.881657

   total_fer      gdpp
0   0.736593  0.003073
1   0.078864  0.036833
2   0.274448  0.040365
3   0.790221  0.031488
4   0.154574  0.114242
```

0.5 K-Means Clustering

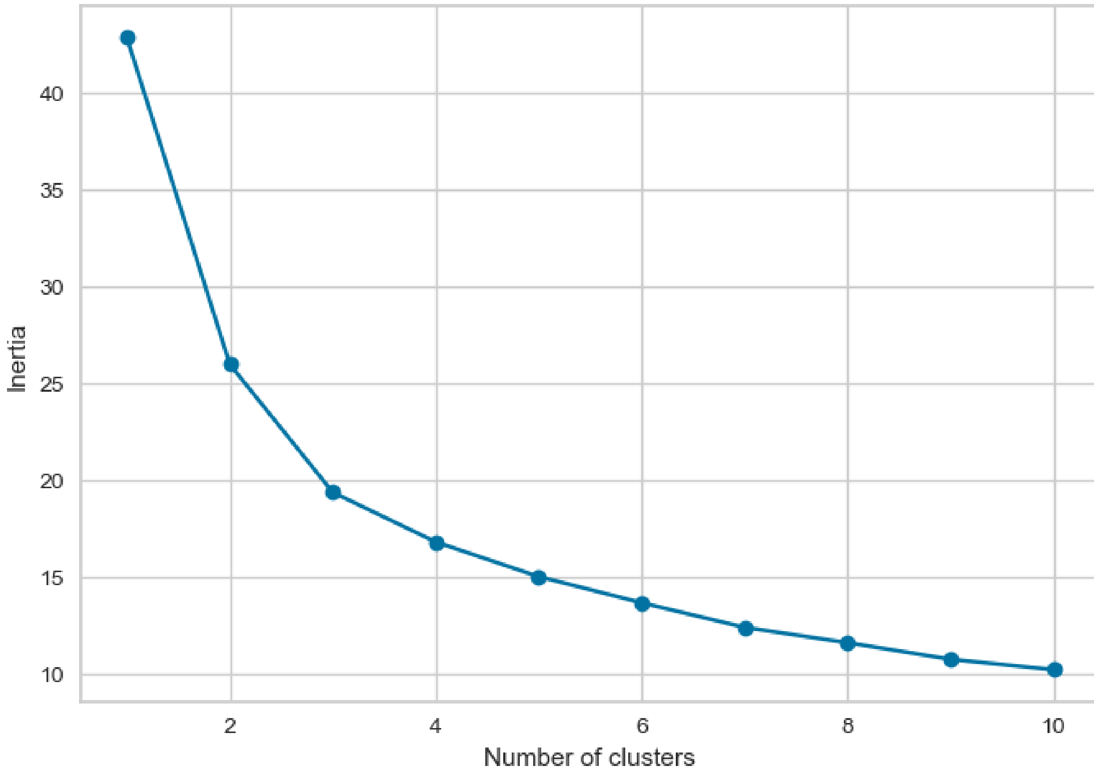
0.5.1 Selecting K

[18]:
```
# Plotting elbow curve
def eval_Kmeans(x, i, r):
    kmeans = KMeans(n_clusters=i, random_state=r, max_iter=500)
    kmeans.fit(x)
    return kmeans.inertia_

def elbow_Kmeans(x, max_k=10, r=42):
    within_cluster_vars = [eval_Kmeans(x, k, r) for k in range(1, max_k+1)]
    plt.plot(range(1, 11), within_cluster_vars, marker='o')
    plt.xlabel('Number of clusters')
    plt.ylabel('Inertia')
    plt.show()

# Plot elbow curve using scaled dataset
elbow_Kmeans(df_scaled, 10, rseed)
```

```

```



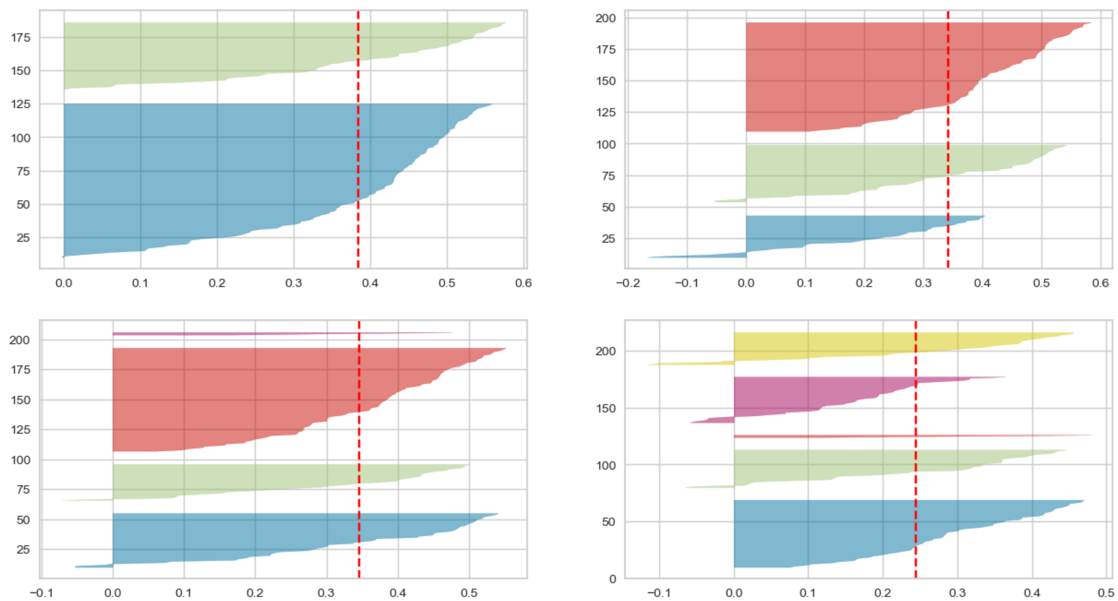
[19]:
```
# Another way to evaluate inertia and plot elbow method is:
inertia_scores=[]

for i in range(1, 11):
    inertia = eval_Kmeans(df_scaled, i, rseed)
    inertia_scores.append(inertia)

plt.plot(range(1, 11), inertia_scores, '-o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



[20]:
```
# Calculating Silhouette score and plotting

fig, ax = plt.subplots(2, 2, figsize=(15,8))
silhouette_avg = []
range_n_clusters= [2, 3, 4, 5]

for i in range_n_clusters:

# Create KMeans instances for different number of clusters
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=100,␣
 ↪random_state=rseed)
    km.fit(df_scaled)
    km_labels=km.labels_
    sil_score = round(silhouette_score(df_scaled, km.labels_,␣
 ↪metric='euclidean'), 4)
    print( "For n_clusters(k) =",
            i,
            "The Silhouette_score is :",
             sil_score
        )
    silhouette_avg.append(sil_score)
```

```
    q, mod = divmod(i, 2)

# Create SilhouetteVisualizer instance with KMeans instance
# Fit the visualizer

    visualizer = SilhouetteVisualizer(km, colors='yellowbrick', ax=ax[q-1][mod])
    visualizer.fit(df_scaled)
```

```
For n_clusters(k) = 2 The Silhouette_score is : 0.3845
For n_clusters(k) = 3 The Silhouette_score is : 0.3427
For n_clusters(k) = 4 The Silhouette_score is : 0.346
For n_clusters(k) = 5 The Silhouette_score is : 0.2449
```

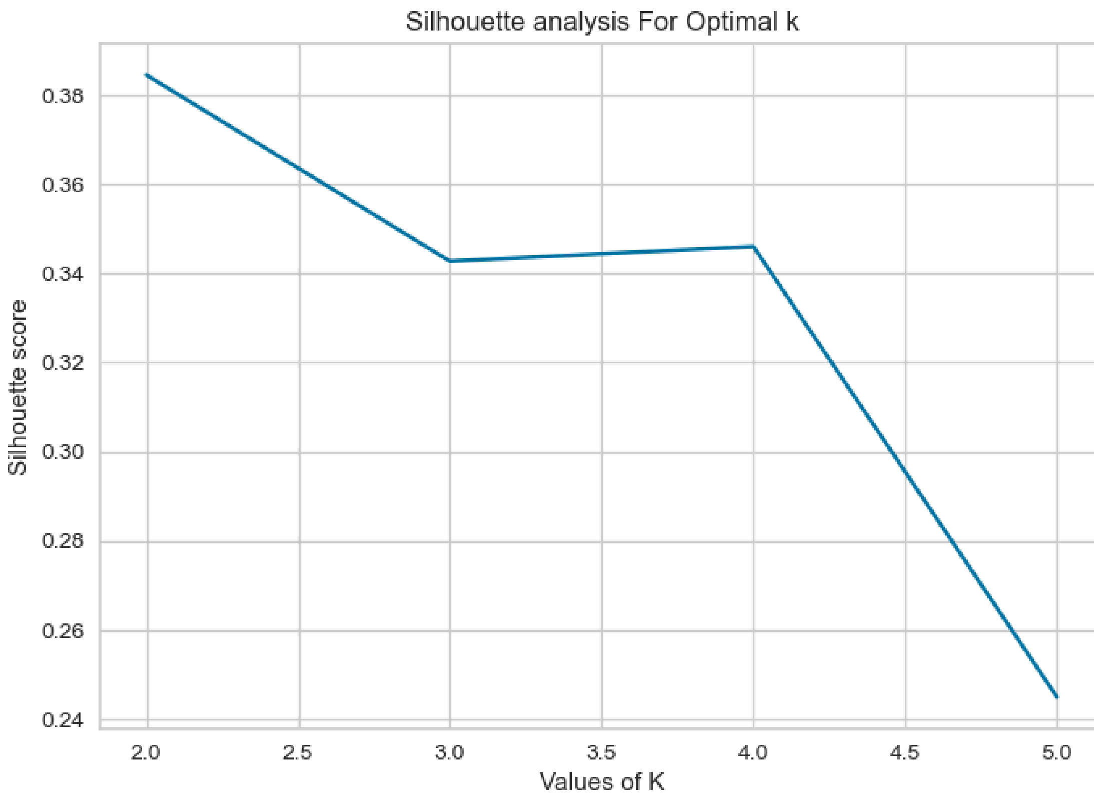


The silhouette score is maximum (0.3845) for K=2, but that's not sufficient to select the optimal K.

The following conditions should be checked to pick the right 'K' using the Silhouette plots: 1. For a particular K, all the clusters should have a Silhouette score greater than the average score of the data set represented by the red-dotted line. The x-axis represents the Silhouette score. Here for all values of k, all clusters have silhouette score greater than the average score of the data.

2. There shouldn't be wide fluctuations in the size of the clusters. The width of the clusters represents the number of data points. For K=4 and k=5, the pink clusters are close to none when compared to the rest of the clusters. For k=2 and k=3 one cluster size is relatively bigger than the other two clusters but since Silhouette score is the greatest for k=2, it should be the best selection here.

[21]:
```
# Visual plot of Silhouette score vs number of clusters
plt.plot(range_n_clusters,silhouette_avg,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Silhouette score')
plt.title('Silhouette analysis For Optimal k')
plt.show()
```



Based on the elbow and silhouette score method, I choose K=2.

0.6 Fitting a K-Means Model with the selected K value

[22]:
```
# Remember to set the random_state to rseed
# Model training with k=2
kmeans_2 = KMeans(n_clusters=2, init='k-means++', random_state=rseed)

# The initialization is not random here. I have used the k-means++␣
↪initialization here which generally produces
# better results as using the K-Means++ algorithm, we optimize the step where␣
↪we randomly pick the cluster centroid.
# We are more likely to find a solution that is competitive with the optimal␣
↪K-Means solution while using the
```

```
# K-Means++ initialization.

kmeans_2.fit(df_scaled)
labels = kmeans_2.labels_
print("KMeans Clustering inertia is:",round(kmeans_2.inertia_,4),'\n')
print("The number of iterations required to converge:", kmeans_2.n_iter_, '\n')
centroids= kmeans_2.cluster_centers_
print("KMeans cluster centres:", centroids, '\n')
print("First five predicted labels:", labels[:5])
```

```
KMeans Clustering inertia is: 25.9424

The number of iterations required to converge: 4

KMeans cluster centres: [[0.0648944  0.23165798 0.32481623 0.27959735 0.17995717
0.09512402
  0.85293818 0.14939356 0.16801166]
 [0.42105313 0.14473182 0.27992054 0.24557475 0.0259766  0.14652065
  0.54368256 0.58882291 0.0158251 ]]

First five predicted labels: [1 0 0 1 0]
```

[23]:
```
# Count the number of records in each cluster
pred = kmeans_2.predict(df_scaled)
print("KMeans predicted values:", pred)
records = pd.DataFrame(df_scaled)
records['cluster'] = pred
records['cluster'].value_counts()
```

```
KMeans predicted values: [1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1
0 1 0 0 1 1 0 0 0 1
 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0
 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1
 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1]
```

[23]:
```
cluster
0    116
1     51
Name: count, dtype: int64
```

0.7 Predictions

[24]:
```
# Adding the predicted cluster label column to the original dataframe
df_scaled['label'] =  kmeans_2.labels_
df_scaled.head()
```

[24]:

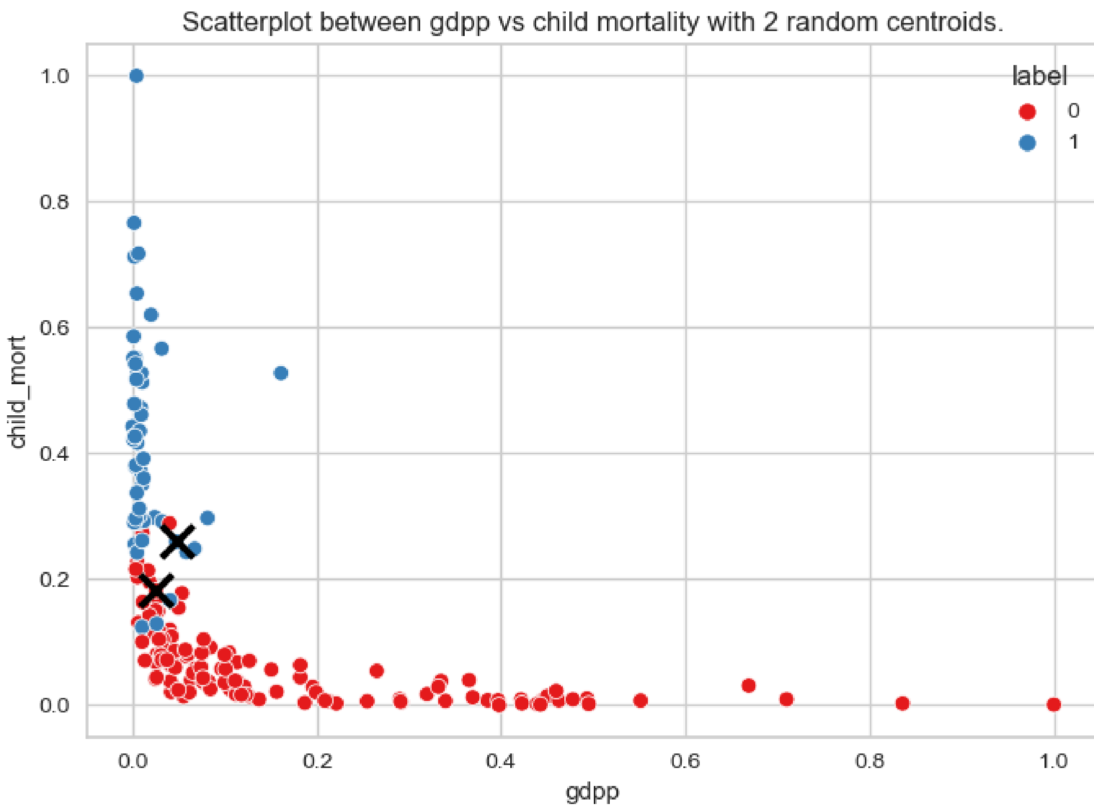
	child_mort	exports	health	imports	income	inflation	life_expec
0	0.426485	0.049482	0.358608	0.257765	0.008047	0.126144	0.475345
1	0.068160	0.139531	0.294593	0.279037	0.074933	0.080399	0.871795
2	0.120253	0.191559	0.146675	0.180149	0.098809	0.187691	0.875740
3	0.566699	0.311125	0.064636	0.246266	0.042535	0.245911	0.552268
4	0.037488	0.227079	0.262275	0.338255	0.148652	0.052213	0.881657

	total_fer	gdpp	label
0	0.736593	0.003073	1
1	0.078864	0.036833	0
2	0.274448	0.040365	0
3	0.790221	0.031488	1
4	0.154574	0.114242	0

0.8 Visualisation of clusters

[25]:

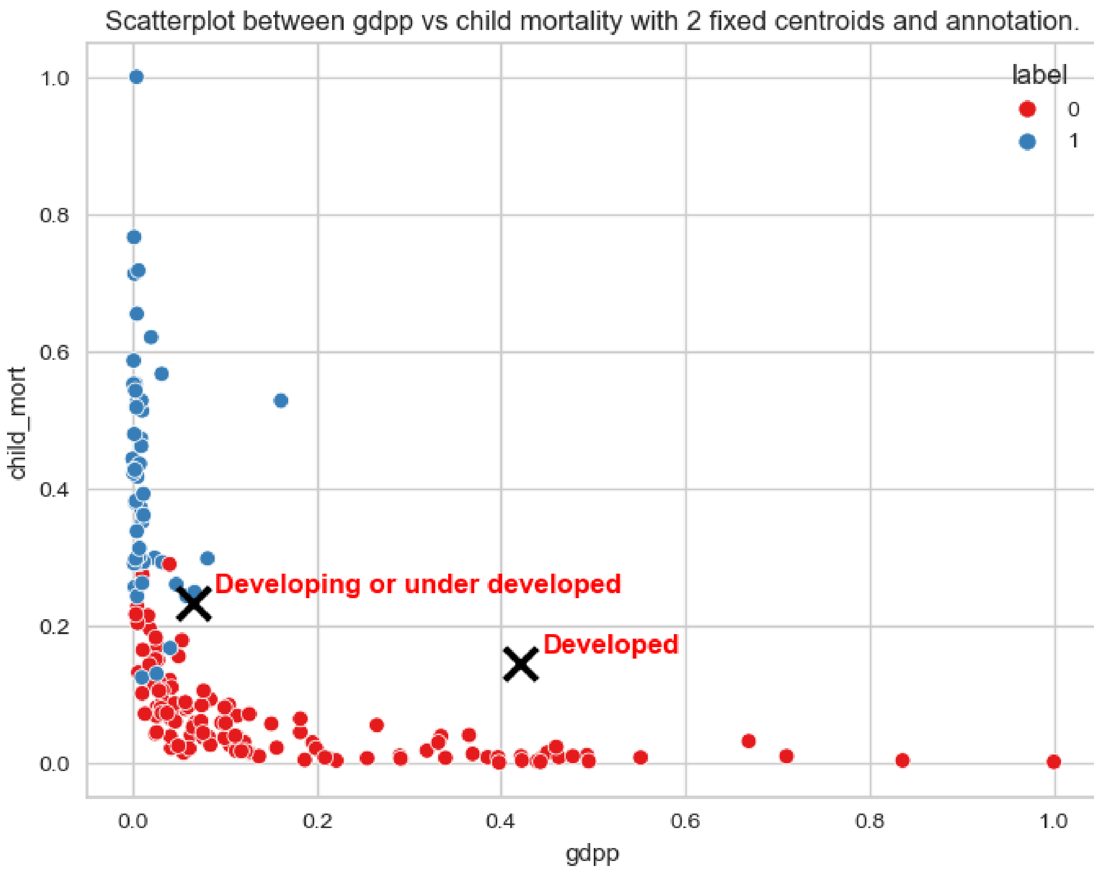
```
# Visualisation of clusters with 2 random centroids selected from the sample:␣
↪child mortality vs gdpp

K=2
centroids1 = df_scaled.sample(n=K, random_state=rseed)
sns.scatterplot(x=df_scaled['gdpp'],y=df_scaled['child_mort'],␣
↪hue=df_scaled['label'],palette='Set1')
plt.scatter(centroids1['gdpp'],centroids1['child_mort'],marker='x', s=200,␣
↪linewidths=3, color='black')
plt.xlabel('gdpp')
plt.ylabel('child_mort')
plt.title("Scatterplot between gdpp vs child mortality with 2 random centroids.
↪")
plt.show()
```



[26]:

```
# Visualisation of clusters with 2 fixed centroids and annotation: child␣
↪mortality vs gdpp

cluster_labels = ['Developing or under developed','Developed']
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df_scaled['gdpp'],y=df_scaled['child_mort'],␣
↪hue=df_scaled['label'],palette='Set1')
plt.scatter(centroids[:,0], centroids[:,1], marker='x', s=200, linewidths=3,␣
↪color='black')

# Annotate the cluster centroids with labels

for i, label in enumerate(cluster_labels):
    plt.annotate(label, (centroids[i, 0], centroids[i, 1]),
                 xytext=(10, 5), textcoords='offset points',
                 fontsize=12, color='red', fontweight='bold')

plt.xlabel('gdpp')
plt.ylabel('child_mort')
plt.title("Scatterplot between gdpp vs child mortality with 2 fixed centroids␣
↪and annotation.")
```

```
plt.show()
```



0.9 Cluster number 1:

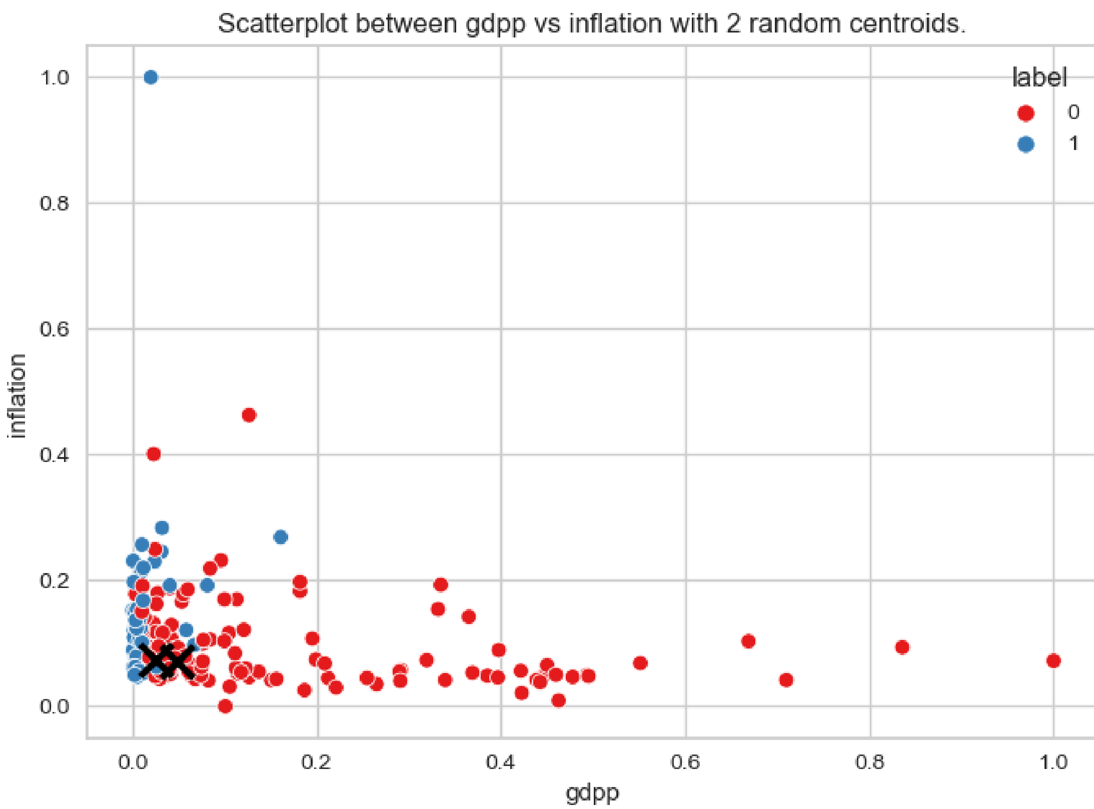
has countries with low child mortality rate and gdpp in the range of 0 to 1 (Scaled data) but most of the countries lie in gdpp range which is less then 0.5. Since their gdpp is high and chilkd mortality rate is low, we can say these must be developed countries.

0.10 Cluster number 2:

has countries with high child mortality rate and low gdpp. So these must be under developed countries.

[27]:
```
# Visualisation of clusters with 2 random centroids selected from the sample:␣
↪inflation vs gdpp

sns.scatterplot(x=df_scaled['gdpp'],y=df_scaled['inflation'],␣
↪hue=df_scaled['label'],palette='Set1')
plt.scatter(centroids1['gdpp'],centroids1['inflation'], marker='x', s=200,␣
↪linewidths=3, color='black')
```

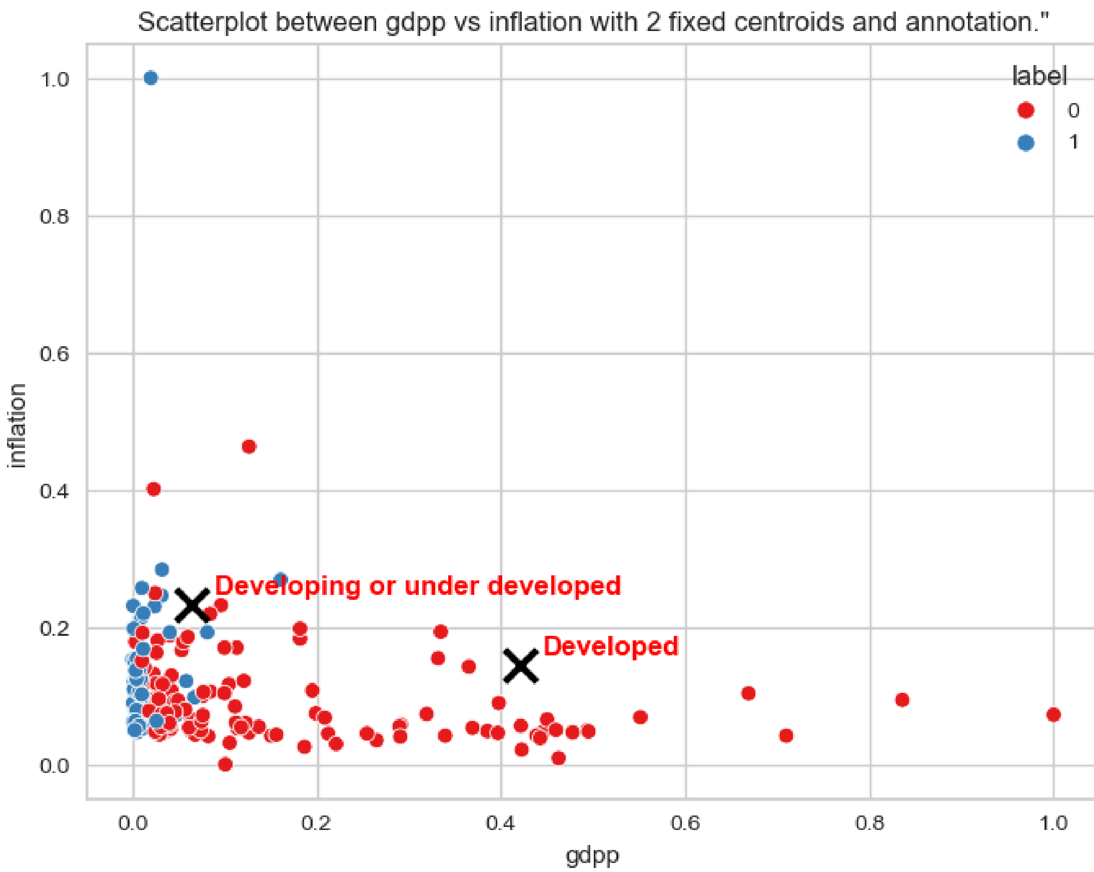
```
plt.xlabel('gdpp')
plt.ylabel('inflation')
plt.title("Scatterplot between gdpp vs inflation with 2 random centroids.")
plt.show()
```



[28]:

```
# Visualisation of clusters with 2 fixed centroids and annotations: inflation␣
↪vs gdpp

cluster_labels = ['Developing or under developed','Developed']
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df_scaled['gdpp'],y=df_scaled['inflation'],␣
↪hue=df_scaled['label'],palette='Set1')
plt.scatter(centroids[:,0], centroids[:,1], marker='x', s=200, linewidths=3,␣
↪color='black')

# Annotate the cluster centroids with labels

for i, label in enumerate(cluster_labels):
    plt.annotate(label, (centroids[i, 0], centroids[i, 1]),
                 xytext=(10, 5), textcoords='offset points',
                 fontsize=12, color='red', fontweight='bold')
```

```
plt.xlabel('gdpp')
plt.ylabel('inflation')
plt.title('Scatterplot between gdpp vs inflation with 2 fixed centroids and␣
↪annotation."')
plt.show()
```



0.11 Cluster number 1:

has countries with gdpp in the range of 0 to 1 (Scaled data) but most of the countries lie in gdpp range which is less then 0.5. the inflation rate in countries which with low gdpp is higher than the countries with high gdpp. So countries with low gdpp and high inflation must be the under developed countries.
Cluster number 2: has countries with lowest gdpp and varying inflation rate. The countries which are close to 0 inflation and 0 gdpp must be developing countries but countries which have high inflation and gdpp close to 0 must be the underdeveloped countries.

Label the groups of countries in the plots you created based on child mortality, GDPP and inflation. You may use terms such as: least developed, developing and developed, or low, low-middle, upper-middle and high income. Alternatively, simply rank them from highest to lowest. Justify the labels you assign to each group.

Answer here:

0.12 Conclusion:

0.13 Cluster number 1:

has countries with gdpp in the range of 0 to 1 (Scaled data) but most of the countries lie in gdpp range which is less then 0.5. The inflation rate in countries with low gdpp is higher than the countries with high gdpp and also they have high child mortality rate. So countries with low gdpp and high inflation and high child mortality rate must be the under developed countries. The same cluster has countries with high gdpp, comparatively low inflation and very low child mortalty rate, making them the developed countries. **Under developed countries:** low gdpp, high inflation, high child mortality rate **Developing countries:** comparatively higher gdpp, comparatively lower infaltion rate than under developed countries and lower child mortality rate. **Developed countries:** Majority of the countries in this cluster are **developed countries** with high gdpp, lower inflation and low child mortality rate

0.14 Cluster number 2:

has countries with lowest gdpp and varying inflation rate. The countries which are close to 0 inflation and 0 gdpp and comparatively low child mortality rate must be developing countries but countries which have high inflation and gdpp close to 0 and high child mortaloity rate must be the underdeveloped countries. **Under developed countries:** lowest gdpp and higher infaltion rate **Developing countries:** low gdpp, comparatively low infaltion and low child mortality rate. Most of the countries in this cluster are **developing countries.**

I just wanted to check since my centroids werent aligned within the clusters and I did some research on it which says if data is multidimensional and we are plotting it in 2D, even if centroids don't seem within the clusters, doesn't mean they arent in the clusters. For that reason a few articles suggest selecting random centroids and thats why I have plotted the graph with fixed and random centroids both. Is that the right approach? Also is there any other reason for us to select random centroids?