projeto_final

November 20, 2022

Projeto de Disciplina

Algoritmos não-supervisionados para clusterização [22E4_2]

Bruno Meletti

0.1 "Installs & Imports"...

[1]:
```
# %pip install scikit-learn-extra
# %pip install plotly
```

[2]:
```
import opendatasets as od
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
import scipy
import scipy.cluster.hierarchy as sch
import numpy as np
import plotly.express as px

# from tqdm.auto import tqdm
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from yellowbrick.cluster import KElbowVisualizer
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn_extra.cluster import KMedoids
from sklearn.metrics import pairwise_distances_argmin_min

# tqdm.pandas(); # <-- Necessário?
```

0.2 Configurações

[3]:
```
download_dataset = False
color_map="Blues"
clusters_number = 3
initializations_number = 10
```

```
maximum_iteration = 300
seed = 42
np.random.seed(seed)
```

0.3 Funções

[4]:
```
# Exibe gráfico de correlação de colunas do dataframe original
def plot_columns_correlation(df):
    correlation_dataframe = df.corr()
    fig, ax = plt.subplots(figsize=(10, 8))
    sb.heatmap(correlation_dataframe, square=True, linewidths=.5,␣
 ↪center=0,annot=True, cmap=color_map);

# Gera o gráfico de cotovelo para análise de número de clusters.
def plot_elbow_analysis(df):
    distortions = []

    K = range(1,10)

    for k in K:
        kmeanModel = KMeans(n_clusters= k, n_init=initializations_number,␣
 ↪max_iter=maximum_iteration, random_state= seed)
        # kmeanModel = KMeans(n_clusters=k)
        kmeanModel.fit(df)
        distortions.append(kmeanModel.inertia_)

    plt.figure(figsize=(10, 5))
    plt.plot(K, distortions, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Distorção')
    plt.title('Método do cotovelo mostrando o k ideal')
    plt.show()

# Exibe o gráfico de distância (euclidiana) entre as colunas do dataframe
def plot_distance_matrix_chart(df, show_annot):
    dist_matrix = scipy.spatial.distance_matrix(df, df, p=2)
    dist_matrix_df = pd.DataFrame(dist_matrix, index=df.index, columns=df.index)

    fig, ax = plt.subplots(figsize=(40, 20))
    sb.heatmap(dist_matrix_df, center=0, square=True, annot=show_annot,␣
 ↪cmap=color_map).set(xlabel='', ylabel='');

# Clusteriza por kmeans.
# Retorna:
# O dataframe de dimensões reduzidas
# Os lables dos clusters de cada linha do dataframe
# Os centroides de cada um dos clusters
```

```
def get_pca_kmeans_cluster(df):
    pca = PCA(2)
    pca_df = pca.fit_transform(df)

    #  kmeans = KMeans(n_clusters=n_clusters, n_init=10, max_iter=300)
    # labels = kmeans.fit_predict(dataset)

    kmeans = KMeans(n_clusters= clusters_number, n_init=initializations_number,␣
 ↪max_iter=maximum_iteration, random_state= seed)

    return pca_df, kmeans.fit_predict(pca_df), kmeans.cluster_centers_

# Clusteriza por kmeans.
# Retorna:
# O dataframe de dimensões reduzidas
# Os lables dos clusters de cada linha do dataframe
# Os centroides de cada um dos clusters
def get_kmeans_cluster(df):
    kmeans = KMeans(n_clusters= clusters_number, n_init=initializations_number,␣
 ↪max_iter=maximum_iteration, random_state= seed)
    # kmeans = KMeans(n_clusters= clusters_number, random_state= seed)

    return df, kmeans.fit_predict(df), kmeans.cluster_centers_

# Gera dataframe contendo a coluna de clusterização por kmeans
def cluster(kmeans_cluster_labels, df):
    kmeans_label = kmeans_cluster_labels

    clustered_df = df.copy()
    clustered_df['kmeans_cluster'] = kmeans_label

    return clustered_df

# Gera o gráfico de clusters
def plot_pca_cluster(pca_df, kmeans_cluster_labels, centroids = None,␣
 ↪show_centroids = False):
    unique_labels = np.unique(kmeans_cluster_labels)
    # labels = ['A', 'B', 'C']

    fig, ax = plt.subplots(figsize=(10, 5))

    for i in unique_labels:
        plt.scatter(pca_df[kmeans_cluster_labels == i , 0] ,␣
 ↪pca_df[kmeans_cluster_labels == i , 1], edgecolors='w', linewidths=1, label=␣
 ↪'Cluster {}'.format(i), s=100, alpha=0.8)
        # ax = plt.scatter(pca_df[label == i , 0] , pca_df[label == i , 1],␣
 ↪edgecolors='w', linewidths=labels[1], label= i, s=100, alpha=0.8)
```

```
    if(show_centroids):
        plt.scatter(centroids[:,0] , centroids[:,1], s = 100, color = 'k',␣
 ↪marker='x', linewidths=3)

    plt.legend()
    plt.show()

# Gera o gráfico de clusters
def plot_cluster(df, kmeans_cluster_labels, centroids = None, show_centroids =␣
 ↪False, show_points_label = False):

    def plotlabel(xvar, yvar, label):
        ax.text(xvar+0.002, yvar, label)

    # unique_labels = np.unique(kmeans_cluster_labels)
    fig, ax = plt.subplots(figsize=(10, 5))

    pca = PCA(2)
    pca_data = pd.DataFrame(pca.fit_transform(df), columns=['PC1','PC2'])
    pca_data['cluster'] = pd.Categorical(kmeans_cluster_labels)
    pca_data['country'] = df.index

    # print(pca_data)
    fig = plt.figure(figsize=(30,10))
    # ax = sb.scatterplot(x = 'sepal_length', y = 'sepal_width', data=df)
    ax = sb.scatterplot(x="PC1", y="PC2", hue="cluster", data=pca_data, ax=ax,␣
 ↪s=100, edgecolors='w', linewidths=1, alpha=0.8) #, label= 'Cluster {}'.
 ↪format(i))
    # sb.scatterplot(x="PC1", y="PC2", hue="cluster", data=pca_data, ax=ax,␣
 ↪s=100, edgecolors='w', linewidths=1, alpha=0.8) #, label= 'Cluster {}'.
 ↪format(i))

    # Exibe o label do ponto
    if(show_points_label):
        pca_data.apply(lambda x: plotlabel(x['PC1'],  x['PC2'], x['country']),␣
 ↪axis=1)

    if(show_centroids):
        plt.scatter(centroids[:,0] , centroids[:,1], s = 100, color = 'k',␣
 ↪marker='x', linewidths=3)

    # plt.legend()

# Gera dendrograma
def plot_open_dendrogram(df):
```

```
    plt.figure(figsize=(20, 7));
    plt.grid(False);
    dendrogram = sch.dendrogram(sch.linkage(df, method='ward'), labels=df.
 ↪index);
    plt.title('Dendrograma');
    plt.ylabel('Distância Euclidiana');

# Recupera o cluster aglomerativo
def get_agglomerative_pca_clustering(df, cluster_number = None):
    if(cluster_number == None):
        agglomerative_clustering =␣
 ↪AgglomerativeClustering(distance_threshold=0, n_clusters=cluster_number)
    else:
        agglomerative_clustering =␣
 ↪AgglomerativeClustering(n_clusters=cluster_number)

    # TODO: validar se é necessário utilizar PCA para clustes aglomerativo
    pca = PCA(2)
    pca_df = pca.fit_transform(df)

    return agglomerative_clustering.fit(pca_df)
    # return agglomerative_clustering.fit(df)

# Recupera o cluster aglomerativo
def get_agglomerative_clustering(df, cluster_number = None):
    if(cluster_number == None):
        agglomerative_clustering =␣
 ↪AgglomerativeClustering(distance_threshold=0, n_clusters=cluster_number)
    else:
        agglomerative_clustering =␣
 ↪AgglomerativeClustering(n_clusters=cluster_number)

    return agglomerative_clustering.fit(df)

# Gera dendrograma hierárquico
def plot_hierarchical_dendrogram(df, agglomerative_clustering, **kwargs):
    counts = np.zeros(agglomerative_clustering.children_.shape[0])

    n_samples = len(agglomerative_clustering.labels_)

    for i, merge in enumerate(agglomerative_clustering.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
```

```
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [agglomerative_clustering.children_, agglomerative_clustering.
 ↪distances_, counts]
    ).astype(float)

    plt.figure(figsize=(20, 7));
    plt.grid(False);
    plt.title("Dendrograma de Cluster Hierárquico")

    sch.dendrogram(linkage_matrix, **kwargs)

    plt.xlabel("Número de pontos no nó (ou índice de ponto se não houver␣
 ↪parênteses).")
    plt.show()

# Gera o gráfico de barras horizontais com os dados do cluster.
def plot_row_chart(df, columns, axis):
    df.plot(y=columns, use_index=True, grid=False,
        kind="barh", width=0.01*len(df), ax=axis, stacked=True)
         # linewidth=0.01*len(df),
        # , figsize=(40, 15))

    axis.set_title('Cluster {} ({} elementos)'.format(df['kmeans_cluster'][0],␣
 ↪len(df)))
    # ax.set_xticks([])
    # plt.show()

# Gera o gráfico de barras com os dados do cluster.
def plot_bar_chart(df, columns, axis):
    df.plot(y=columns, use_index=True, grid=False,
        kind="bar", width=0.01*len(df), ax=axis, stacked=True)

# Exibe gráfico de tabelas
# Tipos
# 1- KMeans
# 2- Hierarchical
# 3- Both
def plot_table(source_df, clusters, type):
    dic = {}

    match type:
        case 1:
            filter_columns = 'kmeans_cluster'
            dic_label = 'KMeans cluster'
        case 2:
```

```
            filter_columns = 'hierarchical_cluster'
            dic_label = 'Hierarchical Cluster'

    if(type != 3):
        for num in range(clusters):
            filtered_df = source_df[(source_df[filter_columns] == num)]
            dic['{} {} - ({} itens)'.format(dic_label, num, len(filtered_df))]␣
 ↪= filtered_df.index.values.tolist()
    else:
        for num in range(clusters):
            kmeans_filtered_df = source_df[(source_df['kmeans_cluster'] == num)]
            hierarchical_filtered_df =␣
 ↪source_df[(source_df['hierarchical_cluster'] == num)]
            dic['KMeans cluster {} - ({} itens)'.format(num,␣
 ↪len(kmeans_filtered_df))] = kmeans_filtered_df.index.values.tolist()
            dic['Hierarchical cluster {} - ({} itens)'.format(num,␣
 ↪len(hierarchical_filtered_df))] = hierarchical_filtered_df.index.values.
 ↪tolist()

    temp_df = pd.DataFrame.from_dict(dic, orient='index')
    temp_df = temp_df.transpose()

    #define figure and axes
    fig, ax = plt.subplots()

    #hide the axes
    fig.patch.set_visible(False)
    ax.axis('off')
    ax.axis('tight')

    #create table
    table = ax.table(cellText=temp_df.values, colLabels=temp_df.columns,␣
 ↪loc='center')
    if(type != 3):
        table.set_fontsize(12)
        table.scale(2,2)
    else:
        table.set_fontsize(14)
        table.scale(6,2)
    ax.axis('off')

    #display table
    # fig.tight_layout()
    plt.show()

# Gera mapa de calor agrupado hierarquicamente.
def plot_cluster_heatmap(df):
```

```
    plt.rcParams["figure.figsize"] = [0, 0]
    plt.rcParams["figure.autolayout"] = True
    sb.set_theme(color_codes=True)

    g = sb.clustermap(df, figsize=(20, 20), cmap=color_map, )
    plt.show()

# Gera o gráfico de mapa mundi identificando os clusters por cor.
# Tipos
# 1- KMeans
# 2- Hierarchical
def plot_map(df, type):
    temp_df = df.copy()

    clusters_name = {0: 'Cluster 0', 1: 'Cluster 1', 2: 'Cluster 2'}
    # clusters_name = {0: 'Cluster A', 1: 'Cluster B', 2: 'Cluster C', 3:␣
 ↪'Cluster D'}

    title = ''

    if(type == 1):
        title = 'KMeans Cluster'
        temp_df['labeled_cluster'] = temp_df['kmeans_cluster'].
 ↪map(clusters_name)
    else:
        title = 'Hierarchical Cluster'
        temp_df['labeled_cluster'] = temp_df['hierarchical_cluster'].
 ↪map(clusters_name)

    fig = px.choropleth(
        temp_df,
        locationmode= 'country names',
        locations= temp_df.index,
        range_color=(0, 12),
        color= 'labeled_cluster',
        title= title,
        width= 700,
        height= 400,
    )

    fig.show()

# Exibe os clusters e objeto central por KMedoids
def get_KMedoids(df, clusters_number):
    X = df.values

    kmedoids = KMedoids(n_clusters=clusters_number, random_state=seed).fit(X)
```

```
    print('Label   Medoid        Index')
    print('---------------------------')
    for index in kmedoids.medoid_indices_:
        label = kmedoids.labels_[index]
        medoid = X[index]

        print(f'{label:<7} {medoid}  {index}')

# Calcula a distância dos pontos para o centroid
def k_mean_distance(data, cx, cy, i_centroid, cluster_labels):
        distances = [np.sqrt((x-cx)**2+(y-cy)**2) for (x, y) in␣
 ↪data[cluster_labels == i_centroid]]
        return distances

# Recupera o item mais próximo do centroid de cada cluster
def get_cluster_nearest_centroid_point(df, cluster_number):
    X = df.values
    pca = PCA(2).fit(X)
    data2D = pca.transform(X)

    kmeans = KMeans(n_clusters= clusters_number, n_init=initializations_number,␣
 ↪max_iter=maximum_iteration, random_state= seed)
    # kmeans = KMeans(n_clusters= clusters_number, random_state= seed)
    clusters=kmeans.fit_predict(data2D)
    centroids = kmeans.cluster_centers_

    distances = []
    for i, (cx, cy) in enumerate(centroids):
        mean_distance = k_mean_distance(data2D, cx, cy, i, clusters)
        distances.append(mean_distance)

    min_index = np.argmin(distances[cluster_number])

    return df.iloc[[min_index]]

# def get_cluster_nearest_centroid_point_old(df, cluster_number):
#     distances = []
#     for i, (cx, cy) in enumerate(centroids):
#         mean_distance = get_kmeans_distance(pca_df, cx, cy, i, cluster_labels)
#         distances.append(mean_distance)

#     min_index = np.argmin(distances[cluster_number])

#     return df.iloc[[min_index]]
```

0.4 Análise e Pré-processamento

[5]:
```
# Download do dataset no Kaggle.
# Nota: Insira as credenciais no prompt...
if download_dataset:
    od.download("https://www.kaggle.com/datasets/rohan0301/
 ↪unsupervised-learning-on-country-data")
```

Dicionário de Dados

Colunas	Descrição
country	Nome do país
child_mort	Morte de crianças menores de 5 anos por 1000 nascidos vivos
exports	Exportações de bens e serviços per capita. Dado em % do PIB per capita.
health	Gastos totais com saúde per capita. Dado em % do PIB per capita.
imports	Importações de bens e serviços per capita. Dado em % do PIB per capita.
income	Rendimentos líquidos por pessoa
inflation	Inflação. Medição da taxa de crescimento anual do PIB Total.
life_expec	Espectativa de vida. Número médio de anos que um recém-nascido viveria se os padrões atuais de mortalidade permanecessem os mesmos.
total_fer	Número de filhos que nasceriam de cada mulher se as taxas atuais de idade-fertilidade permanecerem as mesmas.
gdpp	PIB per capita. Calculado como o PIB total dividido pela população total.

0.4.1 Análise dos Dados

[6]:
```
# Carrega o dataset e exibe os dados
# Optei por transformar a coluna "coutry" em index do dataframe por motivos␣
 ↪práticos
original_df = pd.read_csv("unsupervised-learning-on-country-data/Country-data.
 ↪csv", index_col='country')
#country_df = pd.read_csv("unsupervised-learning-on-country-data/Country-data.
 ↪csv")
original_df.head()
```

[6]:

country	child_mort	exports	health	imports	income	inflation	\
Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	
Albania	16.6	28.0	6.55	48.6	9930	4.49	

```
Algeria                    27.3     38.4    4.17     31.4   12900      16.10
Angola                    119.0     62.3    2.85     42.9    5900      22.40
Antigua and Barbuda        10.3     45.5    6.03     58.9   19100       1.44

                     life_expec  total_fer   gdpp
country
Afghanistan                56.2       5.82    553
Albania                    76.3       1.65   4090
Algeria                    76.5       2.89   4460
Angola                     60.1       6.16   3530
Antigua and Barbuda        76.8       2.13  12200
```

Quantos países existem no dataset?

[7]:
```
# Extrai a lista de países
country_names = original_df.index

print('Temos {} países no dataset. \nSão eles: {}'.format(len(country_names),␣
  ↪', '.join(country_names.tolist())))
```

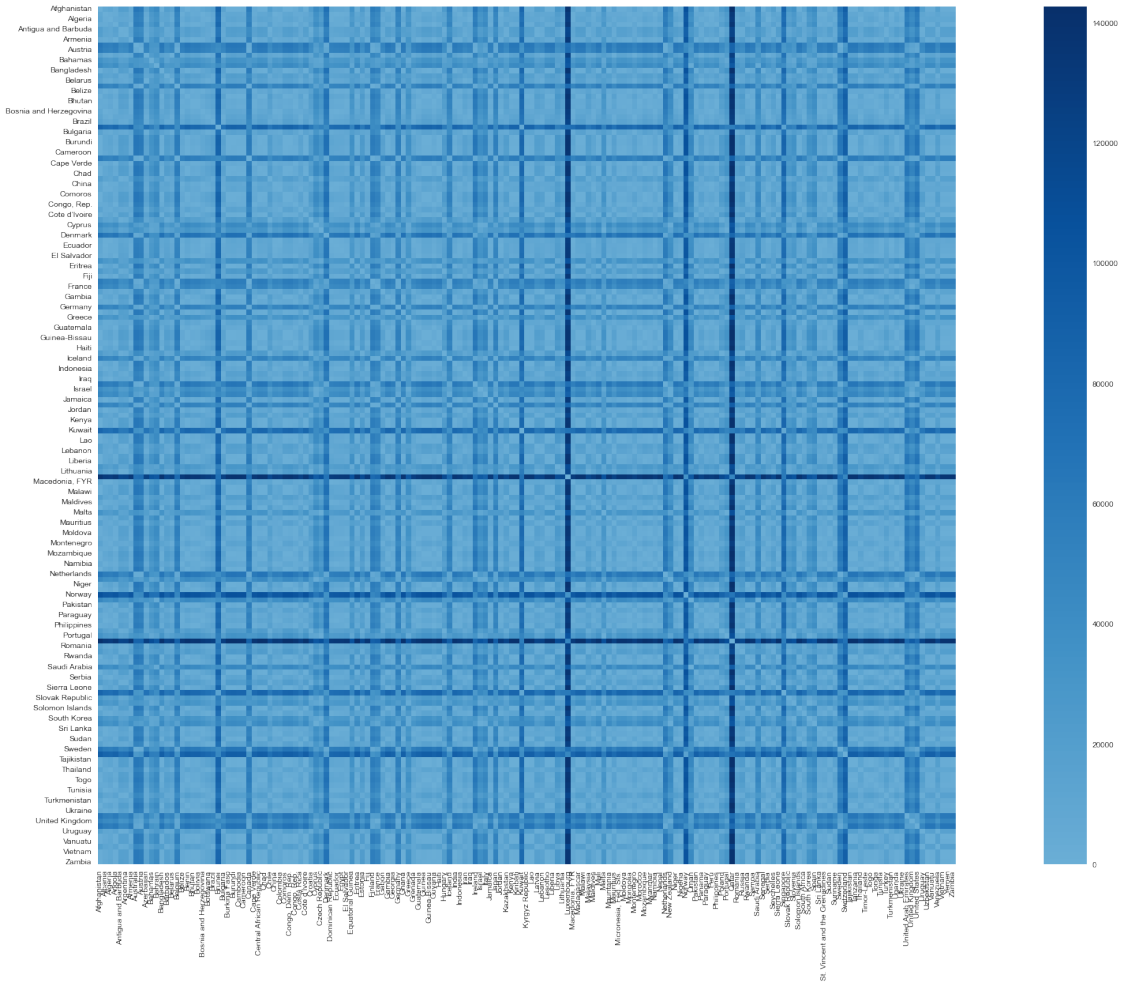
```
Temos 167 países no dataset.
São eles: Afghanistan, Albania, Algeria, Angola, Antigua and Barbuda, Argentina,
Armenia, Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Barbados,
Belarus, Belgium, Belize, Benin, Bhutan, Bolivia, Bosnia and Herzegovina,
Botswana, Brazil, Brunei, Bulgaria, Burkina Faso, Burundi, Cambodia, Cameroon,
Canada, Cape Verde, Central African Republic, Chad, Chile, China, Colombia,
Comoros, Congo, Dem. Rep., Congo, Rep., Costa Rica, Cote d'Ivoire, Croatia,
Cyprus, Czech Republic, Denmark, Dominican Republic, Ecuador, Egypt, El
Salvador, Equatorial Guinea, Eritrea, Estonia, Fiji, Finland, France, Gabon,
Gambia, Georgia, Germany, Ghana, Greece, Grenada, Guatemala, Guinea, Guinea-
Bissau, Guyana, Haiti, Hungary, Iceland, India, Indonesia, Iran, Iraq, Ireland,
Israel, Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya, Kiribati, Kuwait,
Kyrgyz Republic, Lao, Latvia, Lebanon, Lesotho, Liberia, Libya, Lithuania,
Luxembourg, Macedonia, FYR, Madagascar, Malawi, Malaysia, Maldives, Mali, Malta,
Mauritania, Mauritius, Micronesia, Fed. Sts., Moldova, Mongolia, Montenegro,
Morocco, Mozambique, Myanmar, Namibia, Nepal, Netherlands, New Zealand, Niger,
Nigeria, Norway, Oman, Pakistan, Panama, Paraguay, Peru, Philippines, Poland,
Portugal, Qatar, Romania, Russia, Rwanda, Samoa, Saudi Arabia, Senegal, Serbia,
Seychelles, Sierra Leone, Singapore, Slovak Republic, Slovenia, Solomon Islands,
South Africa, South Korea, Spain, Sri Lanka, St. Vincent and the Grenadines,
Sudan, Suriname, Sweden, Switzerland, Tajikistan, Tanzania, Thailand, Timor-
Leste, Togo, Tonga, Tunisia, Turkey, Turkmenistan, Uganda, Ukraine, United Arab
Emirates, United Kingdom, United States, Uruguay, Uzbekistan, Vanuatu,
Venezuela, Vietnam, Yemen, Zambia
```

Mostre através de gráficos a faixa dinâmica das variáveis que serão usadas nas tarefas de clusterização. Analise os resultados mostrados.

[8]:
```
# Exibe a correlação de colunas do dataframe
plot_columns_correlation(original_df)
```

Como pode ser visto no gráfico acima, existem colunas com correlações fortes que poderiam ser analisadas de forma destacada do dataset completo, permitindo análises mais específicas.

[9]:
```
# Exibe o gráfico de distância (euclidiana) entre as colunas do dataframe
plot_distance_matrix_chart(original_df, False)
```



No gráfico acima, o volume de dados não nos permite ter uma visão precisa da relação dos das colunas.

[10]:
```
# Exibe a estrutura dos dados
# original_df.info()

# Verifica se existem colunas com dados ausentes
# original_df.isnull().sum()

# Verifica a presença de países duplicados
duplicated_analysis_series = original_df.duplicated()
duplicate_countries = len(duplicated_analysis_series[duplicated_analysis_series
  ↪== True])

# Resumo do dataframe original.
print(
```

```
    '- {} países distintos (linhas) e {} colunas no dataframe. \n      A coluna␣
 ↪"country" foi transformada em index. \n- {} colunas contém "missing values".␣
 ↪\n- {} países estão duplicados.'.format(
        len(original_df), original_df.shape[1], original_df.isnull().sum().
 ↪sum(), duplicate_countries
    )
)
```

```
- 167 países distintos (linhas) e 9 colunas no dataframe.
      A coluna "country" foi transformada em index.
- 0 colunas contém "missing values".
- 0 países estão duplicados.
```

Olhando para um cenário global, países como México, Cuba, Porto Rico, Groelândia, Etiópia, entre outros, estão ausentes do dataset.

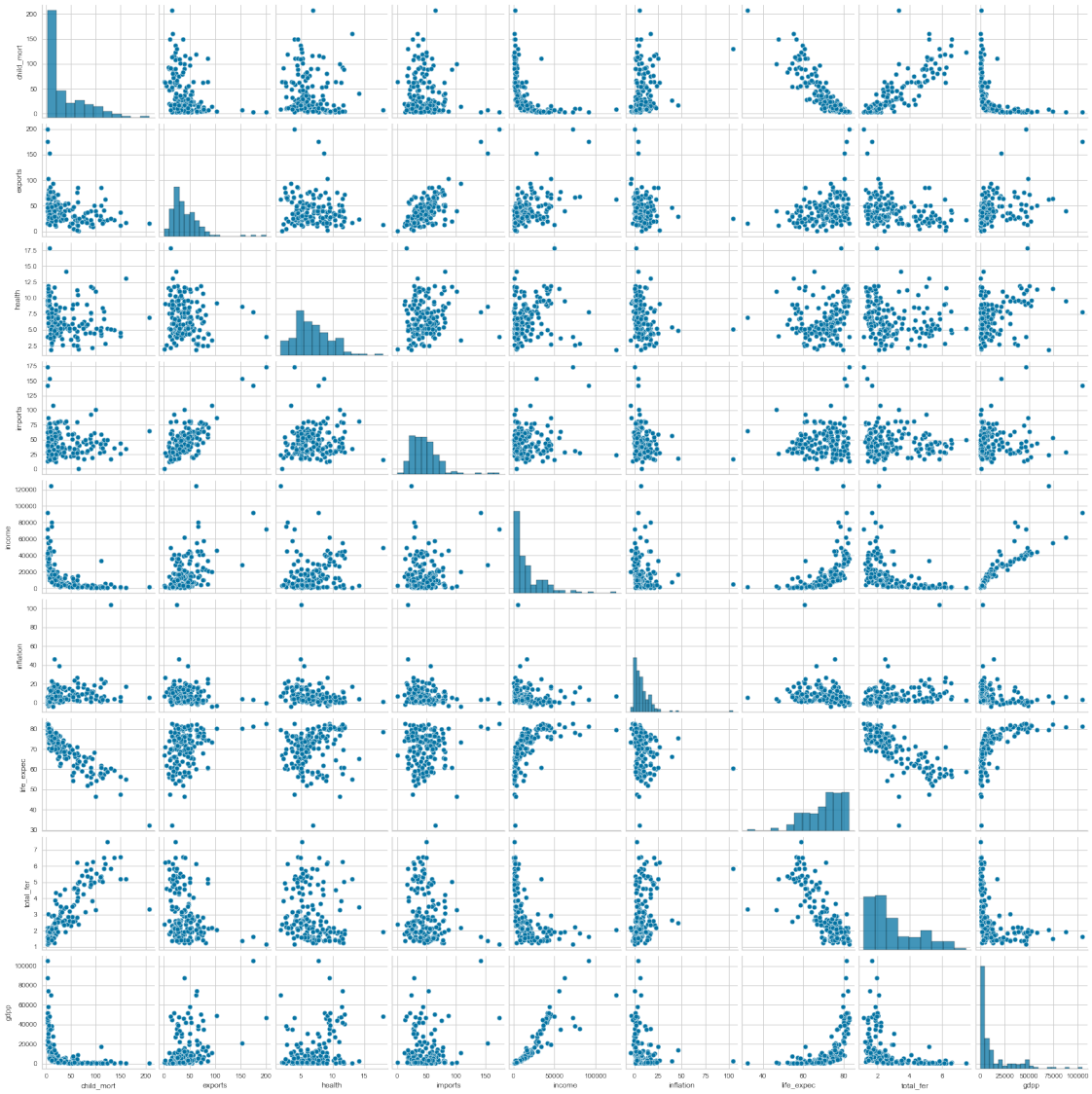
[11]:
```
# Exibe descrição estatística
original_df.describe().T
```

[11]:

	count	mean	std	min	25%	50%	\
child_mort	167.0	38.270060	40.328931	2.6000	8.250	19.30	
exports	167.0	41.108976	27.412010	0.1090	23.800	35.00	
health	167.0	6.815689	2.746837	1.8100	4.920	6.32	
imports	167.0	46.890215	24.209589	0.0659	30.200	43.30	
income	167.0	17144.688623	19278.067698	609.0000	3355.000	9960.00	
inflation	167.0	7.781832	10.570704	-4.2100	1.810	5.39	
life_expec	167.0	70.555689	8.893172	32.1000	65.300	73.10	
total_fer	167.0	2.947964	1.513848	1.1500	1.795	2.41	
gdpp	167.0	12964.155689	18328.704809	231.0000	1330.000	4660.00	

	75%	max
child_mort	62.10	208.00
exports	51.35	200.00
health	8.60	17.90
imports	58.75	174.00
income	22800.00	125000.00
inflation	10.75	104.00
life_expec	76.80	82.80
total_fer	3.88	7.49
gdpp	14050.00	105000.00

[12]:
```
# Exibe o gráfico de relação dos dados do dataframe por pares de colunas.
sb.pairplot(data=original_df);

# g = sb.PairGrid(original_df, vars=original_df.columns);
# g.map_diag(sb.kdeplot, lw=2);
# g.map_offdiag(plt.scatter);
```



O que deve ser feito com os dados antes da etapa de clusterização?

Os dados devem ser normalizados e os outliers, caso existam, devem ser removidos.

0.4.2 Pré-processamento dos Dados

A coluna de nome dos países foi removida e transformada em index na leitura dos dados do arquivo csv

[13]:
```
# Normaliza os dados gerando um novo dataframe
normalized_mean_std_df = (original_df - original_df.mean(axis=0))/ original_df.
  ↪std(axis=0)
normalized_min_max_df = (original_df-original_df.min())/(original_df.
  ↪max()-original_df.min())
```

```
standard_scaler_normalized = StandardScaler().fit_transform(original_df)
min_max_normalized = MinMaxScaler().fit_transform(original_df)
```

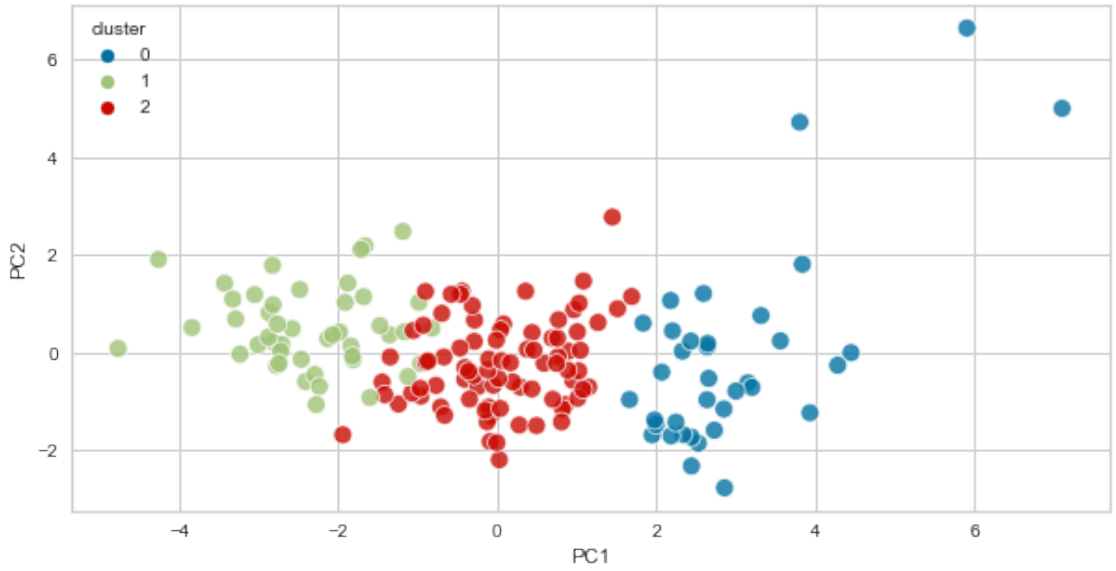
[14]:
```
# Define o dataframe de trabalho segundo a normalização escolhida
normalized_df = normalized_mean_std_df
# normalized_df = normalized_min_max_df
```

0.5 Clusterização

KMeans

[15]:
```
# Como o enunciado já determina o número de clusters, este se torna irrelevante.

# Gera o gráfico de cotovelo para análise de número de clusters.
# plot_elbow_analysis(original_df)
```

[16]:
```
# Clusteriza os dados por kmeans e exibe o gráfico de clusters
df, cluster_labels, centroids = get_kmeans_cluster(normalized_df)
clustered_df = cluster(cluster_labels, normalized_df)
plot_cluster(clustered_df, cluster_labels, centroids)
```




```
<Figure size 2160x720 with 0 Axes>
```

[17]:
```
# Separa os dados em dataframes por cluster
kmeans_cluster_0_df =  clustered_df[(clustered_df['kmeans_cluster'] == 0)]
kmeans_cluster_1_df =  clustered_df[(clustered_df['kmeans_cluster'] == 1)]
kmeans_cluster_2_df =  clustered_df[(clustered_df['kmeans_cluster'] == 2)]
```

Para os resultados do K-Médias, interprete cada um dos clusters obtidos citando a distribuição das dimensões em cada grupo.

[18]:
```
print('Distribuição de elementos por clusters KMeans: \n- Cluster 0: {} itens␣
↪\n- Cluster 1: {} itens \n- Cluster 2: {} itens'.format(
   len(kmeans_cluster_0_df), len(kmeans_cluster_1_df),␣
↪len(kmeans_cluster_2_df)))
```

```
Distribuição de elementos por clusters KMeans:
- Cluster 0: 36 itens
- Cluster 1: 47 itens
- Cluster 2: 84 itens
```

[19]:
```
(kmeans_cluster_0_df.describe().T)
```

[19]:

	count	mean	std	min	25%	50%	75%	max
child_mort	36.0	-0.824968	0.054277	-0.884478	-0.860922	-0.844804	-0.806370	-0.681150
exports	36.0	0.643146	1.529650	-1.047314	-0.416204	0.337116	0.978258	5.796402
health	36.0	0.725230	1.156972	-1.822346	0.334680	0.957214	1.441407	4.035299
imports	36.0	0.190067	1.521876	-1.375084	-0.763756	-0.323848	0.661299	5.250390
income	36.0	1.479792	1.081645	0.417848	0.793924	1.214090	1.529215	5.594716
inflation	36.0	-0.483467	0.385757	-1.040785	-0.693457	-0.623594	-0.391112	0.843668
life_expec	36.0	1.076341	0.204173	0.555967	1.005750	1.101329	1.219397	1.376822
total_fer	36.0	-0.789502	0.246428	-1.187678	-0.997765	-0.748400	-0.654269	0.054190
gdpp	36.0	1.611150	1.036139	0.198369	0.966290	1.540526	1.937444	5.021405
kmeans_cluster	36.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

[20]:
```
(kmeans_cluster_1_df.describe().T)
```

[20]:

	count	mean	std	min	25%	50%
child_mort	47.0	1.356139	0.827575	-0.033972	0.633043	1.287660
exports	47.0	-0.436221	0.662505	-1.419413	-0.881328	-0.631438
health	47.0	-0.155516	0.969120	-1.680365	-0.835757	-0.420734
imports	47.0	-0.188636	0.732468	-1.226382	-0.705928	-0.272215
income	47.0	-0.684834	0.292653	-0.857746	-0.817234	-0.792335
inflation	47.0	0.400905	1.467259	-0.652448	-0.356819	0.107672
life_expec	47.0	-1.278335	0.724547	-4.324181	-1.563637	-1.243166
total_fer	47.0	1.360851	0.687904	-0.236460	1.048346	1.395144
gdpp	47.0	-0.602431	0.161283	-0.694711	-0.677307	-0.658375

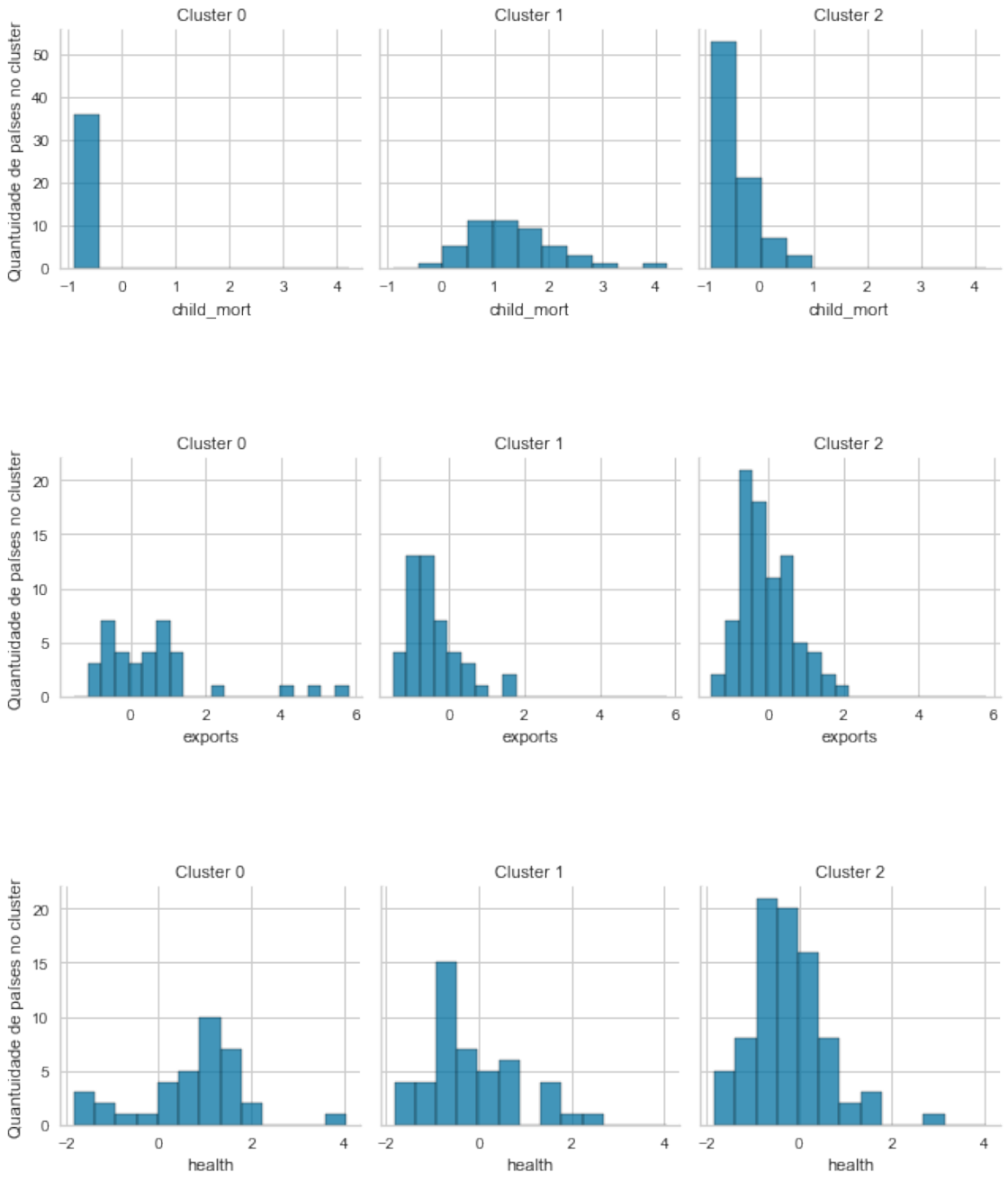
```
kmeans_cluster   47.0  1.000000  0.000000  1.000000  1.000000  1.000000

                      75%       max
child_mort       1.803418  4.208640
exports         -0.106120  1.630345
health           0.469380  2.287835
imports          0.105734  2.235056
income          -0.698705  0.858764
inflation        0.834208  9.102343
life_expec      -0.866472  0.061206
total_fer        1.774971  3.000326
gdpp            -0.627112  0.225648
kmeans_cluster   1.000000  1.000000
```

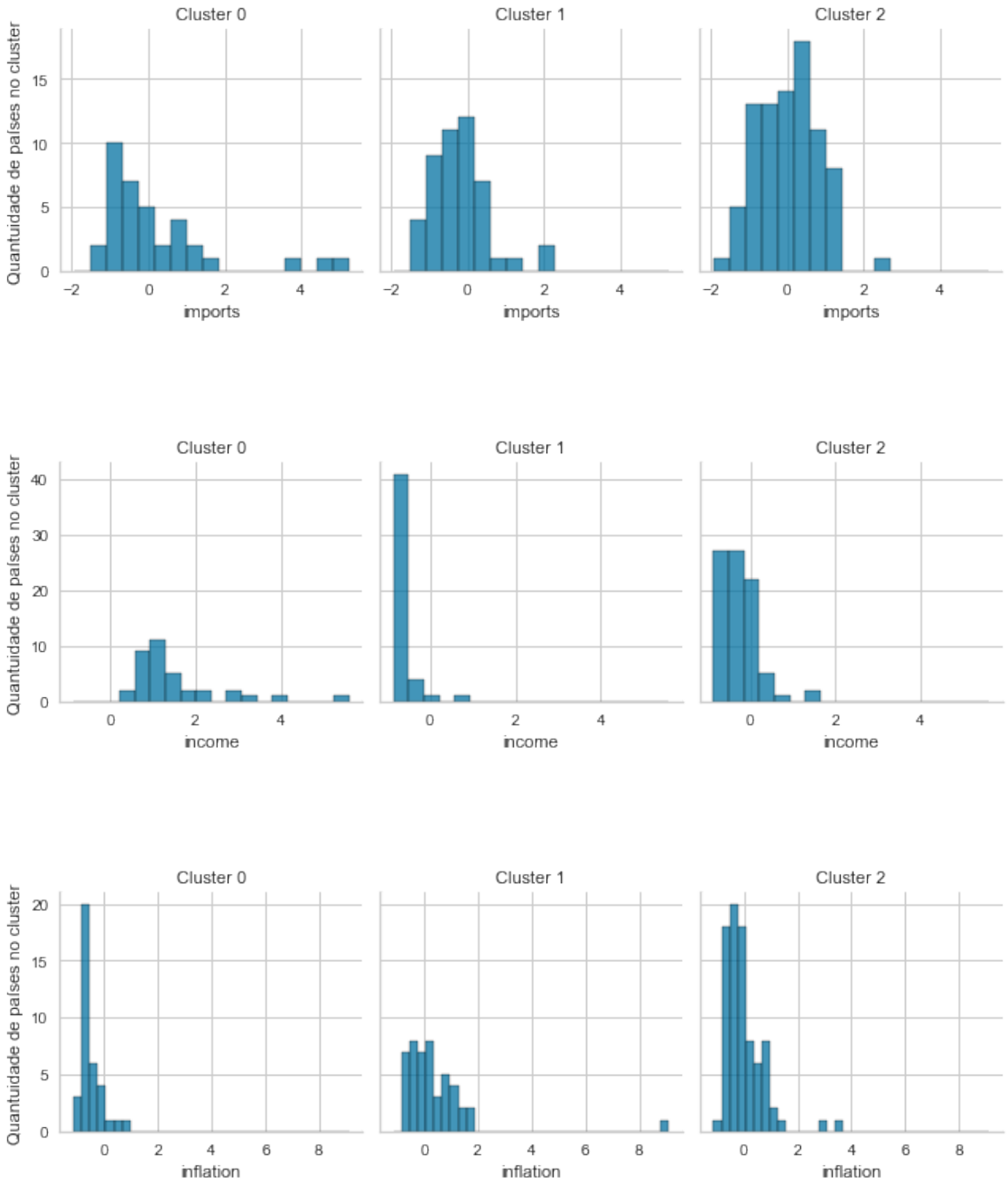
[21]:
```
(kmeans_cluster_2_df.describe().T)
```

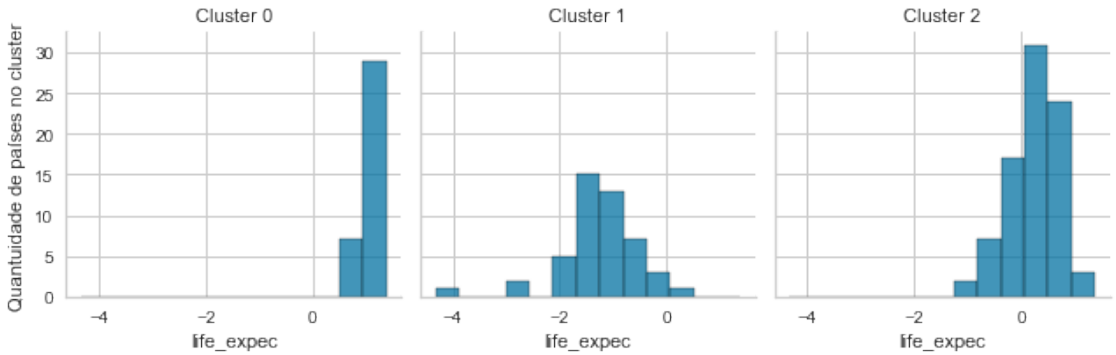
[21]:
```
                count      mean       std       min       25%       50%  \
child_mort       84.0 -0.405235  0.341143 -0.837366 -0.660074 -0.493940
exports          84.0 -0.031558  0.694191 -1.495694 -0.523821 -0.138953
health           84.0 -0.223798  0.788992 -1.764097 -0.713799 -0.322439
imports          84.0  0.024089  0.830410 -1.934123 -0.608859  0.072690
income           84.0 -0.251015  0.424320 -0.797004 -0.545163 -0.361275
inflation        84.0 -0.017116  0.742573 -1.134440 -0.512202 -0.187010
life_expec       84.0  0.253970  0.445369 -0.995785 -0.039996  0.342320
total_fer        84.0 -0.423070  0.460388 -1.121622 -0.791337 -0.470962
gdpp             84.0 -0.353418  0.273966 -0.675015 -0.545410 -0.452523
kmeans_cluster   84.0  2.000000  0.000000  2.000000  2.000000  2.000000

                      75%       max
child_mort      -0.245979  0.647920
exports          0.375420  1.922188
health           0.199069  2.688296
imports          0.559067  2.524198
income          -0.025661  1.465671
inflation        0.212206  3.606020
life_expec       0.623435  1.106952
total_fer       -0.188569  0.919535
gdpp            -0.212462  0.820344
kmeans_cluster   2.000000  2.000000
```

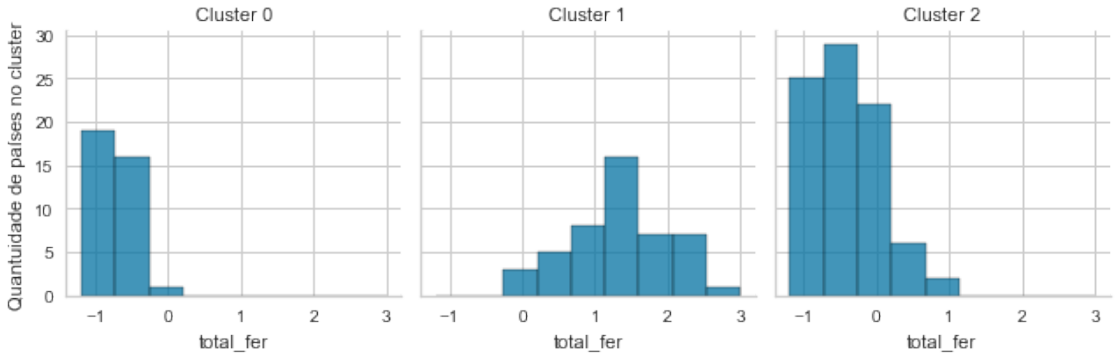
[23]:
```
columns = ['child_mort', 'exports', 'health', 'imports', 'income', 'inflation',␣
 ↪'life_expec', 'total_fer', 'gdpp']

for i in range(len(columns)):
    g = sb.displot(data=clustered_df, x=columns[i], col='kmeans_cluster',␣
 ↪height=3, aspect=1)
    g.set_axis_labels(columns[i], 'Quantuidade de países no cluster')
```

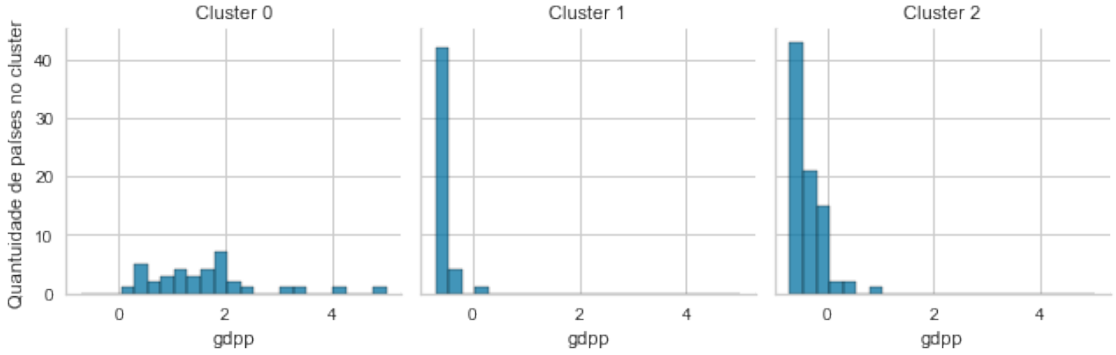
```
g.set_titles('Cluster {col_name}')
```




Cluster 0
Cluster 1
Cluster 2
Quantuidade de palses no cluster
imports
income
inflation







Para os resultados do K-Médias, o país, de acordo com o algoritmo, melhor representa o seu agrupamento.

[24]:
```
# Identifica o país, de acordo com o algoritmo, melhor representa o seu cluster
print('- Cluster 0: {} \n- Cluster 1: {} \n- Cluster 2: {}'.format(
    ''.join(get_cluster_nearest_centroid_point(kmeans_cluster_0_df, 0).index),
    ''.join(get_cluster_nearest_centroid_point(kmeans_cluster_1_df, 1).index),
    ''.join(get_cluster_nearest_centroid_point(kmeans_cluster_2_df, 2).index))
```

```
)
```

- Cluster 0: Austria
- Cluster 1: Afghanistan
- Cluster 2: Argentina

Justificativa: são os países cujas distâcias são mais próximas do centroid de cada cluster.

Clusterização Hierárquica

[25]:
```
# clustered_df['hierarchical_cluster'] =␣
 ↪(get_agglomerative_pca_clustering(normalized_df, clusters_number)).labels_
clustered_df['hierarchical_cluster'] =␣
 ↪(get_agglomerative_clustering(normalized_df, clusters_number)).labels_
clustered_df.head()
```

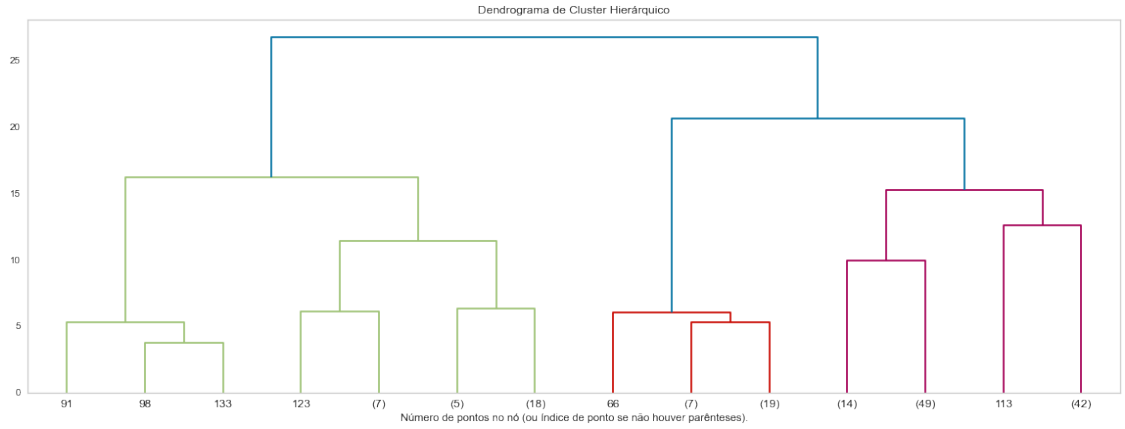
[25]:

country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp	kmeans_cluster	hierarchical_cluster
Afghanistan	1.287660	-1.134867	0.278251	-0.082208	-0.805822	0.156864	-1.614237	1.897176	-0.677143	1	2
Albania	-0.537333	-0.478220	-0.096725	0.070624	-0.374243	-0.311411	0.645924	-0.857394	-0.484167	2	1
Algeria	-0.272015	-0.098824	-0.963176	-0.639838	-0.220182	0.786908	0.668413	-0.038289	-0.463980	2	1
Angola	2.001787	0.773056	-1.443729	-0.164820	-0.583289	1.382894	-1.175698	2.121770	-0.514720	1	1
Antigua and Barbuda	-0.693548	0.160186	-0.286034	0.496076	0.101427	-0.599944	0.702147	-0.540321	-0.041692	2	1

Para os resultados da Clusterização Hierárquica, apresente o dendograma e interprete os resultados

[26]:
```
# # Exibe o dendrograma "aberto", exibindo todos os países
# # plot_open_dendrogram(normalized_df)

# # Exibe o dendrograma hierárquico
# plot_hierarchical_dendrogram(normalized_df,␣
 ↪get_agglomerative_pca_clustering(normalized_df), truncate_mode="level", p=3)
```

```
plot_hierarchical_dendrogram(normalized_df,␣
↪get_agglomerative_clustering(normalized_df), truncate_mode="level", p=3)
```



Este dendrograma foi criado usando-se uma partição final de 3 agrupamentos, que ocorre em um nível de similaridade de aproximadamente 15.

Se a linha de corte for mais alta, haveria menos agrupamentos finais, mas o nível de similaridade seria menor.

Se a linha de corte for mais baixa, o nível de similaridade seria maior, mas haveria mais agrupamentos finais.

Clusterização Hierárquica X KMeans

Compare os dois resultados, aponte as semelhanças e diferenças e interprete.

[34]:
```
clustered_df[['kmeans_cluster', 'hierarchical_cluster']].head()
```

[34]:

country	kmeans_cluster	hierarchical_cluster
Afghanistan	1	2
Albania	2	1
Algeria	2	1
Angola	1	1
Antigua and Barbuda	2	1

[28]:
```
# Tipos
# 1- KMeans
# 2- Hierarchical
# 3- Both

# Exibe a tabela de clusters por KMeans
#plot_table(clustered_df, clusters_number, 1);
```

```
# Exibe a tabela de clusters hierárquicos
#plot_table(clustered_df, clusters_number, 2);

# Exibe tabela comparando os resultados da clusterização hierarquica x KMeans
plot_table(clustered_df, clusters_number, 3);
```

KMeans cluster 0 - (36 itens)	Hierarchical cluster 0 - (34 itens)	KMeans cluster 1 - (47 itens)	Hierarchical cluster 1 - (106 itens)	KMeans cluster 2 - (84 itens)	Hierarchical cluster 2 - (27 itens)
Australia	Australia	Afghanistan	Albania	Albania	Afghanistan
Austria	Austria	Angola	Algeria	Algeria	Benin
Bahrain	Bahrain	Benin	Angola	Antigua and Barbuda	Burkina Faso
Belgium	Belgium	Botswana	Antigua and Barbuda	Argentina	Burundi
Brunei	Brunei	Burkina Faso	Argentina	Armenia	Cameroon
Canada	Canada	Burundi	Armenia	Azerbaijan	Central African Republic
Cyprus	Denmark	Cameroon	Azerbaijan	Bahamas	Chad
Czech Republic	Finland	Central African Republic	Bahamas	Bangladesh	Comoros
Denmark	France	Chad	Bangladesh	Barbados	Congo, Dem. Rep.
Finland	Germany	Comoros	Barbados	Belarus	Cote d'Ivoire
France	Greece	Congo, Dem. Rep.	Belarus	Belize	Gambia
Germany	Iceland	Congo, Rep.	Belize	Bhutan	Guinea
Greece	Ireland	Cote d'Ivoire	Bhutan	Bolivia	Guinea-Bissau
Iceland	Israel	Equatorial Guinea	Bolivia	Bosnia and Herzegovina	Haiti
Ireland	Italy	Eritrea	Bosnia and Herzegovina	Brazil	Kenya
Israel	Japan	Gabon	Botswana	Bulgaria	Madagascar
Italy	Kuwait	Gambia	Brazil	Cambodia	Malawi
Japan	Libya	Ghana	Bulgaria	Cape Verde	Mali
Kuwait	Luxembourg	Guinea	Cambodia	Chile	Mozambique
Luxembourg	Malta	Guinea-Bissau	Cape Verde	China	Niger
Malta	Netherlands	Haiti	Chile	Colombia	Rwanda
Netherlands	New Zealand	Iraq	China	Costa Rica	Senegal
New Zealand	Norway	Kenya	Colombia	Croatia	Sierra Leone
Norway	Oman	Kiribati	Congo, Rep.	Dominican Republic	Tanzania
Portugal	Portugal	Lao	Costa Rica	Ecuador	Togo
Qatar	Qatar	Lesotho	Croatia	Egypt	Uganda
Singapore	Saudi Arabia	Liberia	Cyprus	El Salvador	Zambia
Slovak Republic	Singapore	Madagascar	Czech Republic	Estonia	
Slovenia	Spain	Malawi	Dominican Republic	Fiji	
South Korea	Sweden	Mali	Ecuador	Georgia	
Spain	Switzerland	Mauritania	Egypt	Grenada	
Sweden	United Arab Emirates	Mozambique	El Salvador	Guatemala	
Switzerland	United Kingdom	Namibia	Equatorial Guinea	Guyana	
United Arab Emirates	United States	Niger	Eritrea	Hungary	
United Kingdom		Nigeria	Estonia	India	
United States		Pakistan	Fiji	Indonesia	
		Rwanda	Gabon	Iran	
		Senegal	Georgia	Jamaica	
		Sierra Leone	Ghana	Jordan	
		South Africa	Grenada	Kazakhstan	
		Sudan	Guatemala	Kyrgyz Republic	
		Tanzania	Guyana	Latvia	
		Timor-Leste	Hungary	Lebanon	
		Togo	India	Libya	
		Uganda	Indonesia	Lithuania	
		Yemen	Iran	Macedonia, FYR	
		Zambia	Iraq	Malaysia	
			Jamaica	Maldives	
			Jordan	Mauritius	
			Kazakhstan	Micronesia, Fed. Sts.	
			Kiribati	Moldova	
			Kyrgyz Republic	Mongolia	
			Lao	Montenegro	
			Latvia	Morocco	
			Lebanon	Myanmar	
			Lesotho	Nepal	
			Liberia	Oman	
			Lithuania	Panama	
			Macedonia, FYR	Paraguay	
			Malaysia	Peru	
			Maldives	Philippines	
			Mauritania	Poland	
			Mauritius	Romania	
			Micronesia, Fed. Sts.	Russia	
			Moldova	Samoa	
			Mongolia	Saudi Arabia	
			Montenegro	Serbia	
			Morocco	Seychelles	
			Myanmar	Solomon Islands	
			Namibia	Sri Lanka	
			Nepal	St. Vincent and the Grenadines	
			Nigeria	Suriname	
			Pakistan	Tajikistan	
			Panama	Thailand	
			Paraguay	Tonga	
			Peru	Tunisia	
			Philippines	Turkey	
			Poland	Turkmenistan	
			Romania	Ukraine	
			Russia	Uruguay	
			Samoa	Uzbekistan	
			Serbia	Vanuatu	
			Seychelles	Venezuela	
			Slovak Republic	Vietnam	
			Slovenia		
			Solomon Islands		
			South Africa		
			South Korea		
			Sri Lanka		
			St. Vincent and the Grenadines		
			Sudan		
			Suriname		
			Tajikistan		
			Thailand		
			Timor-Leste		
			Tonga		
			Tunisia		
			Turkey		
			Turkmenistan		
			Ukraine		
			Uruguay		
			Uzbekistan		
			Vanuatu		
			Venezuela		
			Vietnam		
			Yemen		

[29]:
```
# Separa os dados em dataframes por cluster
hierarchical_cluster_0_df =  clustered_df[(clustered_df['hierarchical_cluster']␣
 ↪== 0)]
hierarchical_cluster_1_df =  clustered_df[(clustered_df['hierarchical_cluster']␣
 ↪== 1)]
hierarchical_cluster_2_df =  clustered_df[(clustered_df['hierarchical_cluster']␣
 ↪== 2)]
```

[30]:
```
print('Distribuição de elementos por clusters \n\n A distribuição por KMeans␣
 ↪está mais balanceada do que a distribuição hierárquica.\n- Cluster 0: \n  -␣
 ↪KMeans: {} itens \n  - Hierárquica: {} items \n- Cluster 1: \n  - KMeans: {}␣
 ↪itens \n  - Hierárquica: {} items \n- Cluster 2: \n  - KMeans: {} itens \n ␣
 ↪- Hierárquica: {} items'.format(
    len(kmeans_cluster_0_df), len(hierarchical_cluster_0_df),␣
 ↪len(kmeans_cluster_1_df),  len(hierarchical_cluster_1_df),␣
 ↪len(kmeans_cluster_2_df), len(hierarchical_cluster_2_df)))
```

```
Distribuição de elementos por clusters

 A distribuição por KMeans está mais balanceada do que a distribuição
hierárquica.
- Cluster 0:
  - KMeans: 36 itens
  - Hierárquica: 34 items
- Cluster 1:
  - KMeans: 47 itens
  - Hierárquica: 106 items
- Cluster 2:
  - KMeans: 84 itens
  - Hierárquica: 27 items
```

[31]:
```
# Gera um dataframe final com os dados originais rotulados.
clusterized_original_df = original_df.copy()
clusterized_original_df['kmeans_cluster'] = clustered_df['kmeans_cluster']
clusterized_original_df['hierarchical_cluster'] =␣
 ↪clustered_df['hierarchical_cluster']

# Agrupa os dados para análise
kmeans_cluster_description = clusterized_original_df.
 ↪groupby("kmeans_cluster")[clusterized_original_df.columns.tolist()]
# kmeans_cluster_description = clusterized_original_df.
 ↪groupby("kmeans_cluster")['child_mort', 'exports', 'health', 'imports',␣
 ↪'income', 'inflation', 'life_expec', 'total_fer', 'gdpp', 'kmeans_cluster']
kmeans_n_clients = kmeans_cluster_description.size()
kmeans_cluster_description = kmeans_cluster_description.mean()
kmeans_cluster_description['n_clients'] = kmeans_n_clients
```

```
hierarchical_cluster_description = clusterized_original_df.
 ↪groupby("hierarchical_cluster")[clusterized_original_df.columns.tolist()]
# hierarchical_cluster_description = clusterized_original_df.
 ↪groupby("hierarchical_cluster")['child_mort', 'exports', 'health',␣
 ↪'imports', 'income', 'inflation', 'life_expec', 'total_fer', 'gdpp',␣
 ↪'hierarchical_cluster']
hierarchical_n_clients = hierarchical_cluster_description.size()
hierarchical_cluster_description = hierarchical_cluster_description.mean()
hierarchical_cluster_description['n_clients'] = hierarchical_n_clients
```

[32]:
```
kmeans_cluster_description[['child_mort', 'exports', 'health', 'imports',␣
 ↪'income', 'inflation', 'life_expec', 'total_fer', 'gdpp']]
```

[32]:

kmeans_cluster	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	5.000000	58.738889	8.807778	51.491667	45672.222222	2.671250	80.127778	1.752778	42494.444444
1	92.961702	29.151277	6.388511	42.323404	3942.404255	12.019681	59.187234	5.008085	1922.382979
2	21.927381	40.243917	6.200952	47.473404	12305.595238	7.600905	72.814286	2.307500	6486.452381

Interpretação dos Clusters KMeans

- Cluster 0:
  - Menor mortalidade de crianças menores de 5 anos por 1.000 nascidos vivos
  - Maiores exportações de bens e serviços per capita.
  - Maior gasto total em saúde per capita.
  - Maiores importações de bens e serviços per capita.
  - Maior renda líquida por pessoa
  - Menor taxa de inflação
  - Maior expectativa de vida
  - Maior O PIB per capita
- Cluster 1:
  - Maior mortalidade de crianças menores de 5 anos por 1.000 nascidos vivos
  - Menores exportações de bens e serviços per capita.
  - Menores importações de bens e serviços per capita.
  - Menor renda líquida por pessoa
  - Maior taxa inflação
  - Menor expectativa de vida
  - Menor O PIB per capita.
- Cluster 2:
  - Menor gasto total com saúde per capita.

[33]:
```
hierarchical_cluster_description[['child_mort', 'exports', 'health', 'imports',␣
↪'income', 'inflation', 'life_expec', 'total_fer', 'gdpp']]
```

[33]:

hierarchical_cluster	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	5.961765	58.508824	8.501176	48.902941	47588.235294	4.115500	79.982353	1.888529	43170.588235
1	31.617925	39.990368	6.353679	48.085527	11341.886792	9.120604	70.921698	2.654623	6407.367925
2	105.070370	23.589630	6.507037	39.662963	1589.740741	7.142778	57.248148	5.433704	667.888889

Interpretação dos Clusters Hierárquicos

- Cluster 0:
  - Menor mortalidade de crianças menores de 5 anos por 1.000 nascidos vivos
  - Maiores exportações de bens e serviços per capita
  - Maior gasto total em saúde per capita
  - Maiores importações de bens e serviços per capita
  - Maior renda líquida por pessoa
  - Menor taxa de inflação
  - Maior expectativa de vida
  - Maior O PIB per capita
- Cluster 1:
  - Menor Gasto total com saúde per capita
  - Maior taxa de inflação
- Cluster 2:
  - Maior mortalidade de crianças menores de 5 anos por 1.000 nascidos vivos
  - Menores exportações de bens e serviços per capita
  - Menores importações de bens e serviços per capita
  - Menor renda líquida por pessoa
  - Menor expectativa de vida
  - Menor O PIB per capita

Comparando as duas interpretações, pode-se dizer que. Apesar dos rótulos dos clustes 1 e 2 parecerem invertidos, a distribuição dos pontos me parece coerente.

0.5.1 Escolha de algoritmos

1. Escreva em tópicos as etapas do algoritmo de K-médias até sua convergência.

Resposta

1. Inicializar k centróides em pontos aleatórios
2. Para cada ponto, encontrar o centróide mais próximo
3. Calcular o baricentro dos pontos de cada centróide
4. Mover o centróide na direção do seu baricentro
5. Repetir, a partir do passo 2, até a convergência do algorítimo

O algoritmo converge quando o movimento for menor que um valor pré-definido, quando os elementos de cada cluster não mudam ou quando o número de iterações pré-especificado for atingido.

---

2. O algoritmo de K-médias converge até encontrar os centróides que melhor descrevem os clusters encontrados (até o deslocamento entre as interações dos centróides ser mínimo).

Lembrando que o centróide é o baricentro do cluster em questão e não representa, em via de regra, um dado existente na base.

Refaça o algoritmo apresentado na questão 1 a fim de garantir que o cluster seja representado pelo dado mais próximo ao seu baricentro em todas as iterações do algoritmo. *Obs: nesse novo algoritmo, o dado escolhido será chamado medóide. (rever a aula do kmedoids)*

Resposta

Medoid é definido como o objeto no cluster cuja dissimilaridade média para todos os objetos no cluster é mínima, ou seja, é um ponto central do cluster.

1. Selecionar k **objetos** aleatóreos no conjunto de dados e definir como medóids. *Em comparação com KMeans, ao invés de sortear k pontos aleatóreos, k objetos serão selecionados do dataset.*
2. Calcular a matriz de dissimilaridade.
3. Atribuir cada ponto ao seu medóide mais próximo.
4. Para cada cluster, verificar se existe algum ponto mais central do que o medóide inicialmente escolhido. (Dissimilaridade dele para todos os outros pontos do cluster é menor).
5. Repetir a partir do passo 3 até a convergência do algorítimo.

---

3. O algoritmo de K-médias é sensível a outliers nos dados. Explique.

Resposta

Sim. Porque outliers são pontos isolados que se diferenciam drasticamente de todos os outros. Como KMeans é baseado em médias, os outliers afetam o cálculo exercendo peso e fazendo com que o centroid seja deslocado de um ponto razoável, ficando longe dos pontos na qual ele deveria estar ligado.

---

4. Por que o algoritmo de DBScan é mais robusto à presença de outliers?

Resposta

Porque é um algorítimo baseado em densidade. Onde, dado o conjunto de pontos no espaço, agrupa os pontos próximos marcando como ruído os pontos isolados em região de baixa densidade. Os ruídos recebem o rótulo -1.