

```
In [4]: # Repositorio git
# https://github.com/nilosaj/posgrad2024_alg_cluster
```

Bibliotecas e leitura do Dataset

```
In [5]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

2 Escolha da base de dados

2.1 Leitura do Dataset

```
In [6]: country_data_df = pd.read_csv('./data/Country-data.csv')
country_data_df
```

```
Out[6]:
```

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

167 rows x 10 columns

```
In [7]: # ajuste dos tipos de dados
cols = country_data_df.columns.drop('country')
country_data_df[cols] = country_data_df[cols].apply(pd.to_numeric, errors='c
country_data_df
```

Out [7]:

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

167 rows × 10 columns

2.2 Total de países

```
In [8]: print(f"Total de países: {country_data_df['country'].unique().size}")
```

Total de países: 167

2.3 Faixa dinâmica e análise dos resultados

```
In [9]: # revisão ddos daods minimos
country_data_df.min()
```

```
Out[9]: country      Afghanistan
child_mort          2.6
exports            0.109
health             1.81
imports           0.0659
income              609
inflation          -4.21
life_expec         32.1
total_fer           1.15
gdpp                231
dtype: object
```

```
In [10]: # revisão dos dados máximos
country_data_df.max()
```

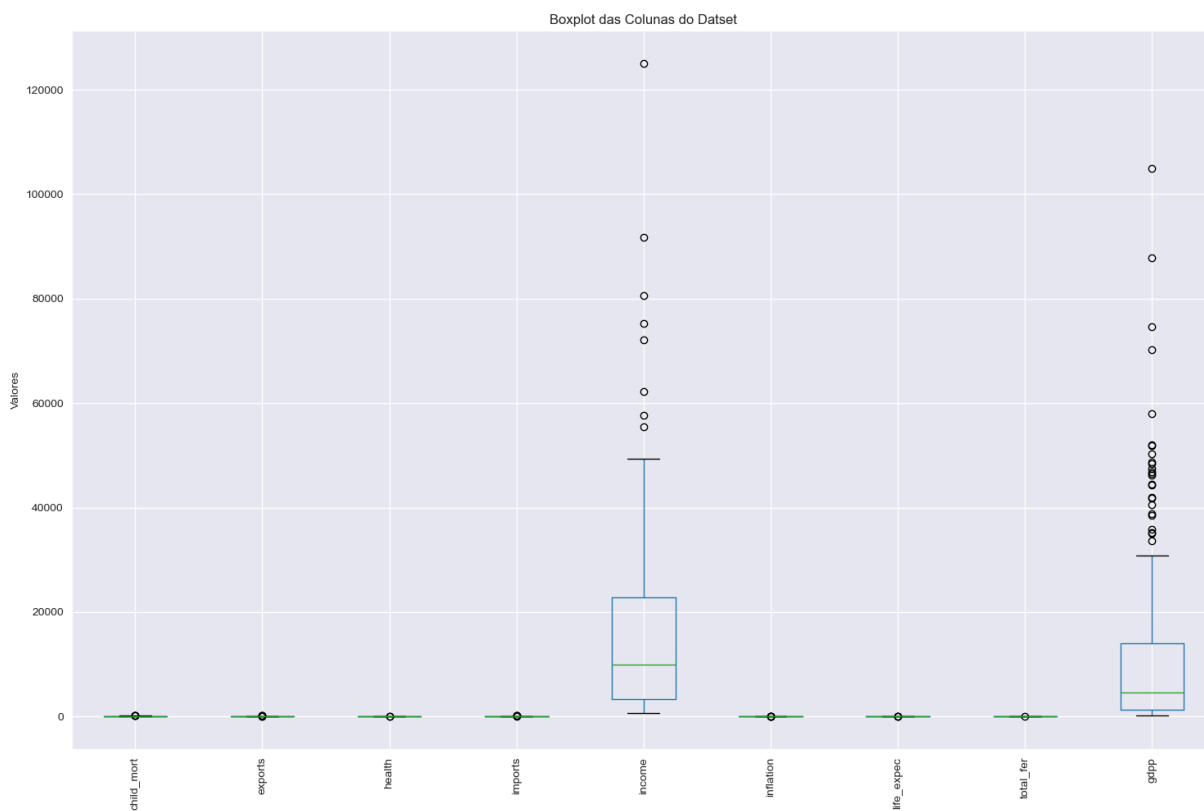
```
Out[10]: country      Zambia
child_mort    208.0
exports      200.0
health       17.9
imports      174.0
income      125000
inflation    104.0
life_expec   82.8
total_fer    7.49
gdpp        105000
dtype: object
```

```
In [11]: cols_analysis = country_data_df.select_dtypes(include=['number'])
fig, ax = plt.subplots(figsize=(15, 10))

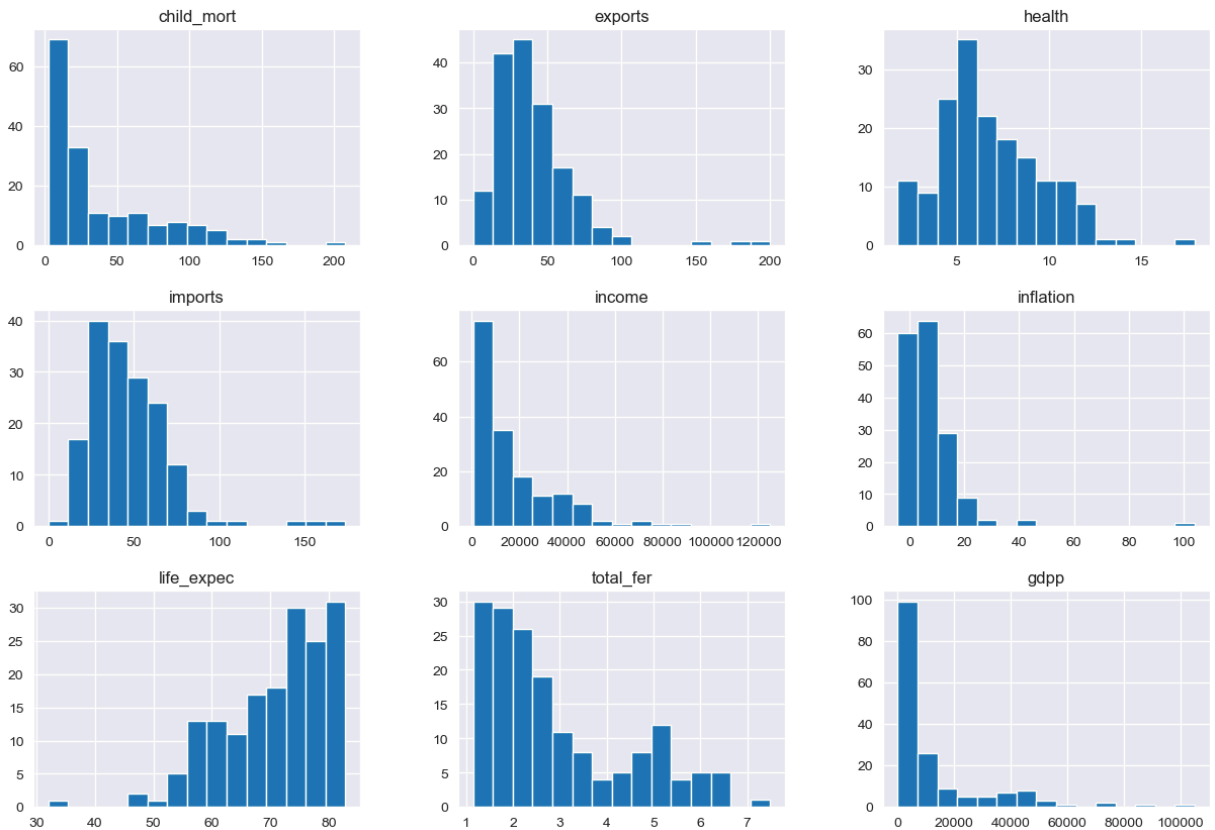
cols_analysis.boxplot(ax=ax)
ax.set_title('Boxplot das Colunas do Dataset')
ax.set_ylabel('Valores')
ax.set_xticklabels(cols_analysis.columns, rotation=90)

plt.tight_layout()
plt.show()

country_data_df.iloc[:, 1:].hist(bins=15, figsize=(15, 10))
plt.suptitle('Distribuição das Variáveis')
plt.show()
```



Distribuição das Variáveis



2.4 Pré processamento dos dados

```
In [12]: from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scale_data = scaler.fit_transform(cols_analysis)
normalized_df = pd.DataFrame(scale_data, columns=cols_analysis.columns)
normalized_df.head()
```

```
Out[12]:
```

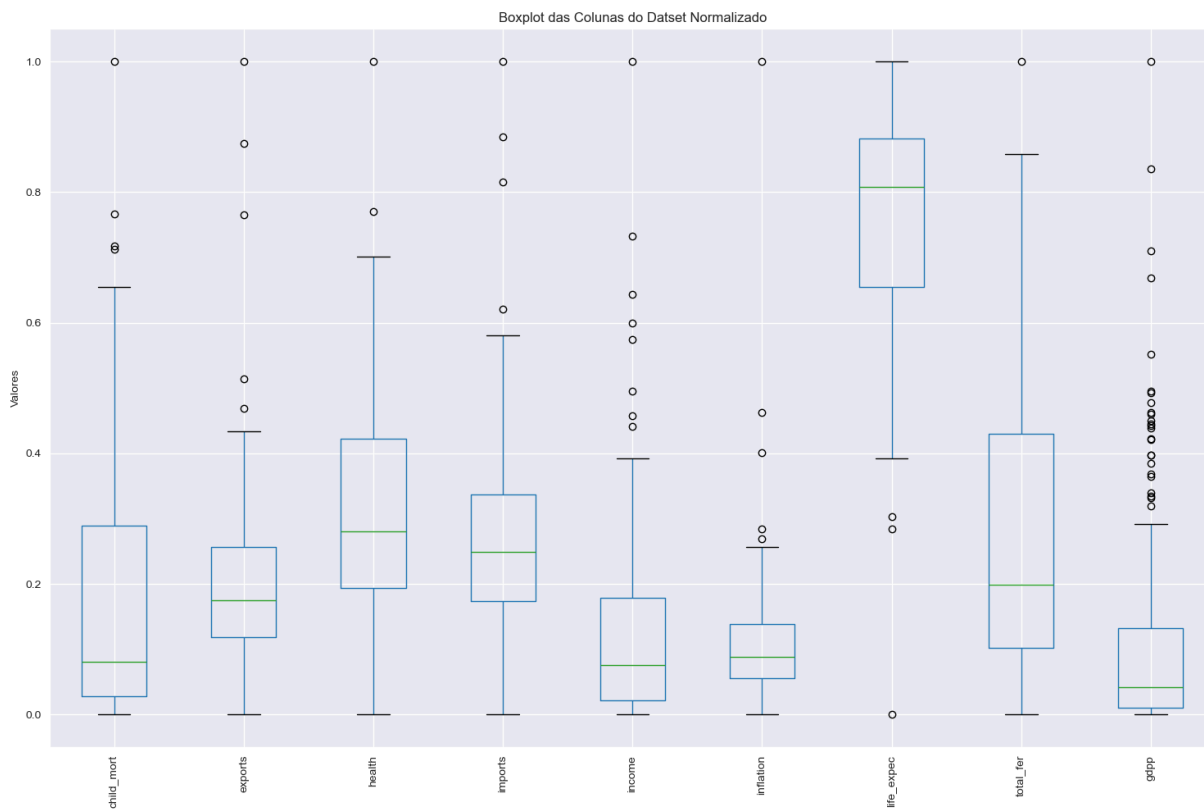
	child_mort	exports	health	imports	income	inflation	life_expec	total_fer
0	0.426485	0.049482	0.358608	0.257765	0.008047	0.126144	0.475345	0.7365
1	0.068160	0.139531	0.294593	0.279037	0.074933	0.080399	0.871795	0.0788
2	0.120253	0.191559	0.146675	0.180149	0.098809	0.187691	0.875740	0.2744
3	0.566699	0.311125	0.064636	0.246266	0.042535	0.245911	0.552268	0.7902
4	0.037488	0.227079	0.262275	0.338255	0.148652	0.052213	0.881657	0.1545

```
In [13]: scaled_cols_analysis = normalized_df.select_dtypes(include=['number'])
fig, ax = plt.subplots(figsize=(15, 10))

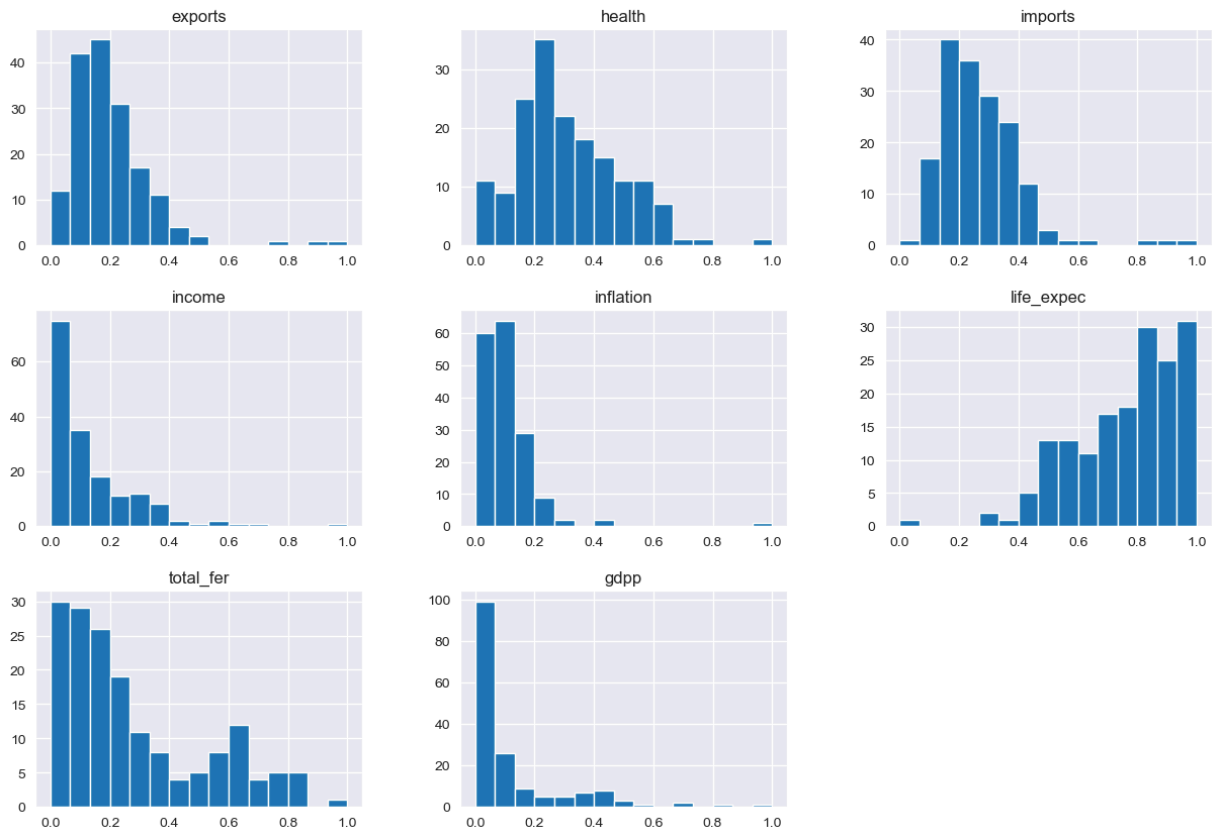
scaled_cols_analysis.boxplot(ax=ax)
ax.set_title('Boxplot das Colunas do Datset Normalizado')
ax.set_ylabel('Valores')
ax.set_xticklabels(cols_analysis.columns, rotation=90)
```

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

scaled_cols_analysis.iloc[:, 1:].hist(bins=15, figsize=(15, 10))
plt.suptitle('Distribuição das Variáveis')
plt.show()
```



Distribuição das Variáveis



3 Clusterização

3.1.a K-Means

```
In [14]: from sklearn.cluster import KMeans
```

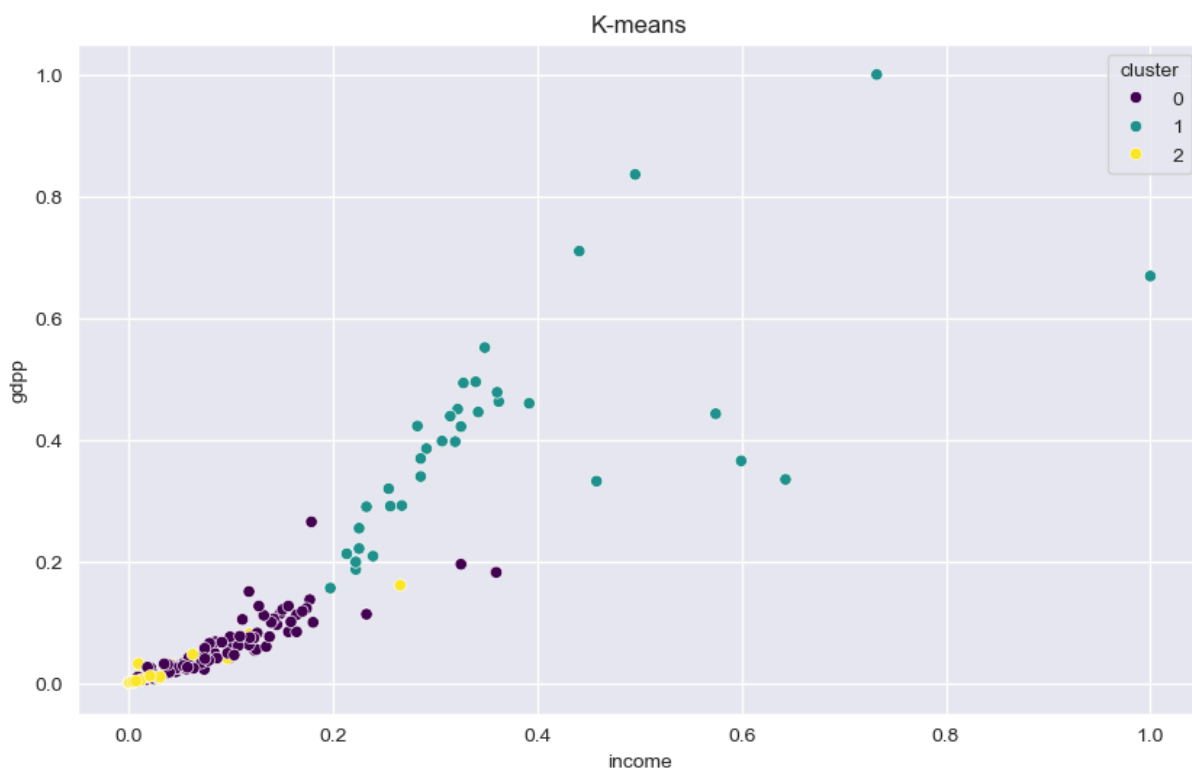
```
In [15]: kmeans = KMeans(n_clusters=3, random_state=42)
normalized_df['cluster'] = kmeans.fit_predict(normalized_df)
normalized_df['cluster']
```

/Users/nilosaj/miniconda3/lib/python3.12/site-packages/threadpoolctl.py:121
 4: RuntimeWarning:
 Found Intel OpenMP ('libiomp') and LLVM OpenMP ('libomp') loaded at
 the same time. Both libraries are known to be incompatible and this
 can cause random crashes or deadlocks on Linux when loaded in the
 same Python program.
 Using threadpoolctl may cause crashes or deadlocks. For more
 information and possible workarounds, please see
https://github.com/joblib/threadpoolctl/blob/master/multiple_openmp.md
 warnings.warn(msg, RuntimeWarning)

```
Out[15]: 0      2
         1      0
         2      0
         3      2
         4      0
         ..
        162     0
        163     0
        164     0
        165     2
        166     2
        Name: cluster, Length: 167, dtype: int32
```

```
In [16]: colonas_usadas = ['income', 'gdpp']
#colonas_usadas = ['child_mort', 'health']
#colonas_usadas = ['child_mort', 'life_expec']
#colonas_usadas = ['exports', 'imports']

plt.figure(figsize=(10, 6))
plt.title('K-means')
sns.scatterplot(data=normalized_df, x=colonas_usadas[0], y=colonas_usadas[1])
plt.xlabel(colonas_usadas[0])
plt.ylabel(colonas_usadas[1])
plt.show()
```

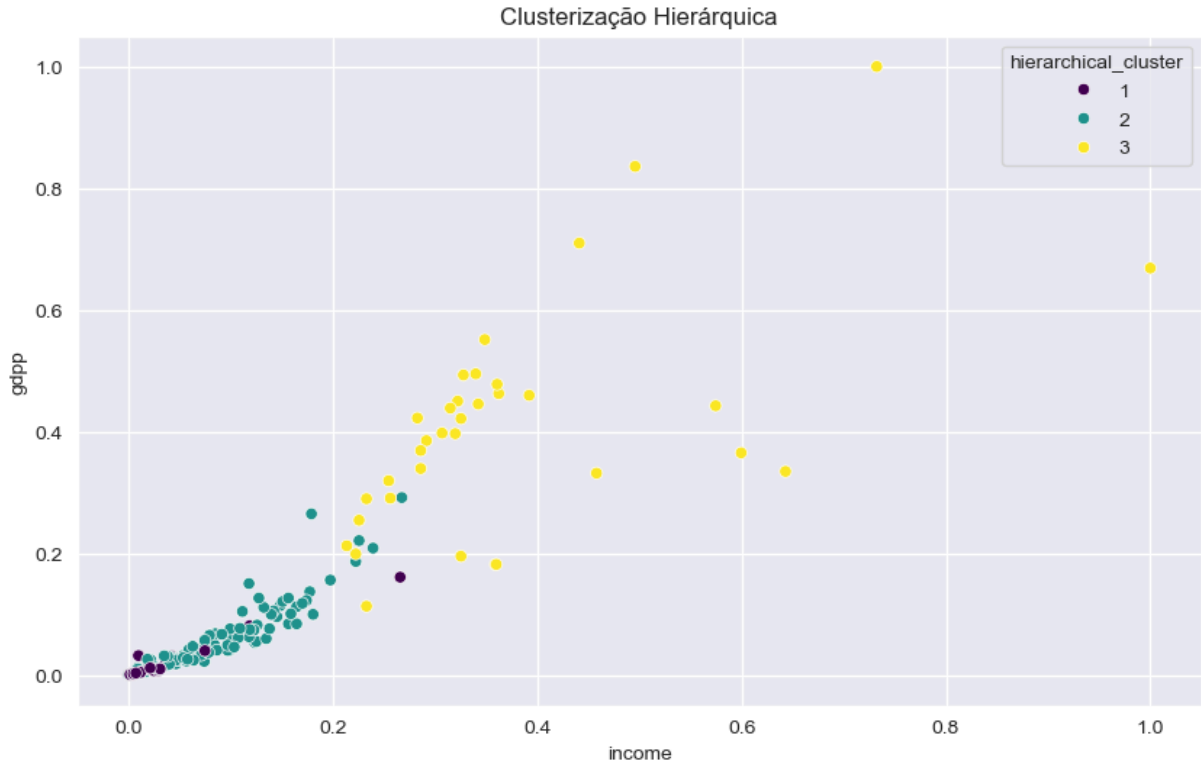


3.1.b Clusterização Hierárquica

```
In [17]: from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

link = linkage(normalized_df.drop(columns=['cluster']), method='ward')
```

```
normalized_df['hierarchical_cluster'] = fcluster(link, 3, criterion='maxclus
plt.figure(figsize=(10, 6))
sns.scatterplot(data=normalized_df, x=colunas_usadas[0], y=colunas_usadas[1]
plt.title('Clusterização Hierárquica')
plt.xlabel(colunas_usadas[0])
plt.ylabel(colunas_usadas[1])
plt.show()
```



3.2 Analise dos Dados

```
In [18]: # Grupo 0
g0_df = normalized_df[normalized_df['cluster'] == 0].agg(['min', 'max', 'mean', 'std'])
g0_df
```

```
Out[18]:
```

	child_mort	exports	health	imports	income	inflation	life_expect	total_fert
max	0.300876	0.46871	0.770044	0.620546	0.360082	0.463081	0.952663	0.500000
mean	0.096839	0.20199	0.275476	0.271713	0.096755	0.109138	0.796005	0.180000
min	0.009250	0.00000	0.009944	0.000000	0.009414	0.000000	0.437870	0.010000

```
In [19]: # Grupo 1
g1_df = normalized_df[normalized_df['cluster'] == 1].agg(['min', 'max', 'mean', 'std'])
g1_df
```



```
Out [19]:
```

	child_mort	exports	health	imports	income	inflation	life_expec	total
max	0.039922	1.000000	1.00000	1.000000	1.000000	0.193235	1.000000	0.29
mean	0.011184	0.291771	0.44173	0.295760	0.363321	0.062333	0.949620	0.09
min	0.000000	0.061489	0.00000	0.077812	0.197691	0.009149	0.856016	0.00

```
In [20]: # Grupo 2
g2_df = normalized_df[normalized_df['cluster'] == 2].agg(['min', 'max', 'mea
g2_df
```

```
Out [20]:
```

	child_mort	exports	health	imports	income	inflation	life_expec	to
max	1.000000	0.428689	0.701678	0.580301	0.266024	1.000000	0.769231	1.0
mean	0.448556	0.143746	0.278392	0.243711	0.023687	0.151784	0.537322	0.6
min	0.166991	0.010461	0.024239	0.098509	0.000000	0.047084	0.000000	0.3

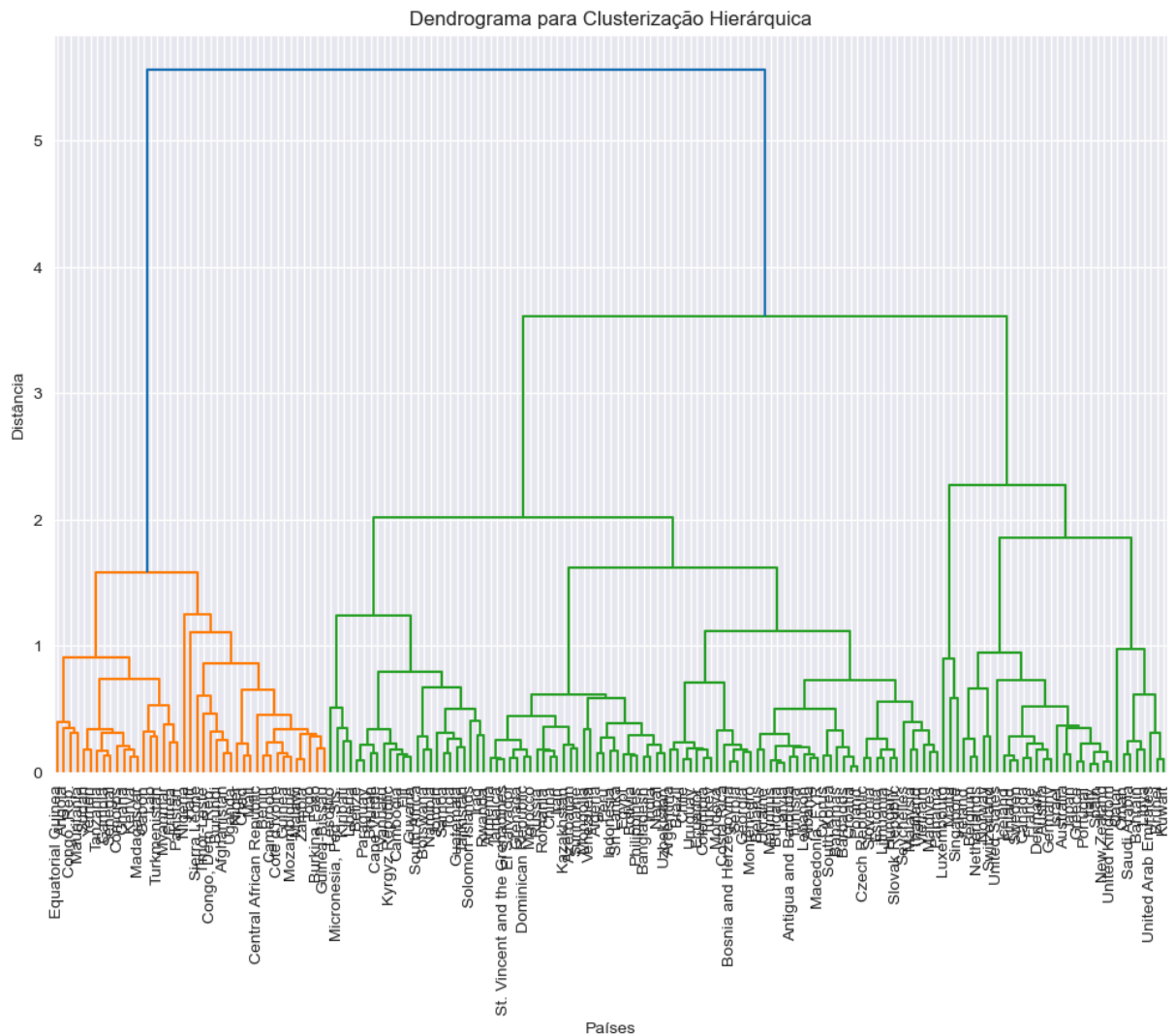
```
In [21]: # child_mort = 1 < 0 < 2
# exports = 1 > 0 > 2
# health = 1 > 2 > 0
# imports = 1 > 0 > 2
# income = 1 > 0 > 2
# inflation =
# life_expect = 1 < 0 < 2
# total_fer = 2 > 0 > 1
# gdpp = 1 > 0 > 2
```

Considerando legenda da visualização do K-Means

- Cluster 1 é referente a países mais ricos , com maior qualidade de vida , maior renda , menor inflação e menor mortalidade infantil
- Cluster 2 é referente a países com menor renda, menor qualidade de vida, maior inflação e maior mortalidade infantil
- Cluster 0 é referente a países com renda intermediária, qualidade de vida intermediária, inflação intermediária e mortalidade infantil intermediária

3.3 Dendrograma

```
In [22]: plt.figure(figsize=(12, 8))
dendrogram(link, labels=country_data_df['country'].values, leaf_rotation=90,
plt.title('Dendrograma para Clusterização Hierárquica')
plt.xlabel('Países')
plt.ylabel('Distância')
plt.show()
```



- Grupo laranja indica um cluster países com menor renda ,menor qualidade de vida e mais problemas economicos
- Grupo verde agrupa os clusters de países onde as analises da economia e qualidade de vida são intermediárias e altas.
- Pelo dendrograma podemos ver que o numero de clusters ideal é verificado pelo corte entre os níveis 3 e 4, sendo assim temos 3 clusters.

3.4 Conclusão

Analizando os dados do K-means e da clusterização podemos confirmar que existem 3 grupos de países, sendo eles. O Dedograma trás uma analise mais detalhada de quão proximos estão os países e como eles se agrupam.

4 Escolha de algoritmos

4.1 Algoritmo K-means

1. Definir numero de clusters Indica o numero de clusters que serão criados
2. Inicializar centróides Os centróides são pontos aleatórios dentro do espaço de dados onde os clusters serão criados
3. Atribuir cada ponto ao cluster mais próximo Calcular a distância entre cada ponto e o centróide e atribuir o ponto ao cluster mais próximo (centróide mais proximo)
4. Recalcular centróides Considerando os pontos atribuidos a cada cluster, recalcular os centróides
5. Repetir passos 3 e 4 até convergir Repetir os passos 3 e 4 até que os centróides não se movam mais

4.2 Ajuste no Algoritmo

1. Definir numero de clusters Indica o numero de clusters que serão criados
2. Inicializar centróides Os centróides são pontos aleatórios dentro do espaço de dados onde os clusters serão criados
3. Atribuir cada ponto ao cluster mais próximo Calcular a distância entre cada ponto e o centróide e atribuir o ponto ao cluster mais próximo (centróide mais proximo)
4. Recalcular centróides Considerando os pontos atribuidos a cada cluster, considere o ponto mais próximo do centroide como medóide. considere o medóide como novo centroide do cluster
5. Repetir passos 3 e 4 até convergir Repetir os passos 3 e 4 até que os centróides não se movam mais

4.3 Algoritmo K-Means é sensível a outliers?

O K-means é sensível a outliers, pois ao definir os centróides por meio das médias um ponto que esteja muito distante dos outros vai influenciar e deslocar este centróide para um ponto extremamente aquém do ideal e atrapalhar a convergência do algoritmo.

4.4 DbScan é sensível a outliers?

O DBScan é menos sensível pois isola os outliers e os classifica como ruído. o DBscan considera a densidade dos pontos na hora de identificar os clusters, por isso consegue isolar estes outliers.