Projeto de Validação de Modelos de Clusterização

Justificativa da escolha do dataset

> **Nota:** Insira aqui a justificativa da escolha do dataset, explicando sua relevância para o problema de clusterização e as características que o tornam desafiador ou interessante.

- Fabio Ferreira Figueiredo 

Sobre o Projeto

- Modelo: Se trata de um modelo de clusterização não supervisionado, com o objetivo de agrupar países com base em seus dados socioeconômicos e de saúde.
- Objetivo: usar dados socioeconômicos e de saúde para agrupar países e identificar o grupo em extrema necessidade de ajuda humanitária.

Este notebook está organizado para atender às especificações do trabalho:

- Parte 1: Infraestrutura (ambiente local, venv, bibliotecas, requirements, evidências).
- Parte 2: Escolha de base de dados e Análise Exploratória (EDA).
- Parte 3: Clusterização (K-Médias e Hierárquica), interpretação e comparação.
- Parte 4: Escolha de algoritmos (passos do K-médias, variante com medóide, discussão de outliers e DBSCAN).

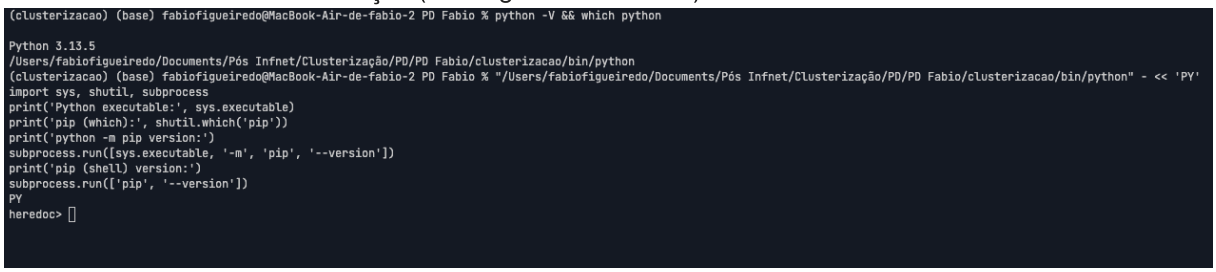
Observações:

- Rode as células na ordem em que aparecem.
- Todas as bibliotecas necessárias estão listadas no arquivo `requirements.txt`.
- Caso o download via Kaggle API não funcione, coloque o arquivo localmente em `data/` e rode novamente.
- Para submissão, gere `requirements.txt` com pip freeze.

Parte 1: Infraestrutura (checagens + evidências)

- Versão do Python (3.13.5).
- Está em um ambiente virtual (Virtualenv) - python3 -m venv clusterizacao.
- Caminhos do Python e do pip.

Evidências de ambiente e execução (montagem consolidada):



In [1]:
```
import sys
import shutil

print('Python version:', sys.version)
print('Python executable:', sys.executable)

# Detecta se venv está ativo (Virtualenv/Conda)
is_venv = sys.prefix != getattr(sys, 'base_prefix', sys.prefix)
print('Ambiente virtual ativo:', is_venv)

# Caminhos úteis
print('pip path:', shutil.which('pip'))
print('python -m pip path:', shutil.which('python'))

# Dica: tire um print desta célula para evidência de execução local e venv.
```

```
Python version: 3.13.5 | packaged by Anaconda, Inc. | (main, Jun 12 2025, 11:23:37) [Clang
14.0.6 ]
Python executable: /Users/fabiofigueiredo/Documents/Pós Infnet/Clusterização/PD_Modelos de
validação/pd_modelos_clusterizacao/clusterizacao/bin/python
Ambiente virtual ativo: True
pip path: /Users/fabiofigueiredo/Documents/Pós Infnet/Clusterização/PD_Modelos de validaçã
o/pd_modelos_clusterizacao/clusterizacao/bin/pip
python -m pip path: /Users/fabiofigueiredo/Documents/Pós Infnet/Clusterização/PD_Modelos d
e validação/pd_modelos_clusterizacao/clusterizacao/bin/python
```

Imports e configuração geral

- Se alguma biblioteca não estiver instalada, instale no seu venv: `pip install <pacote>`.
- Pacotes necessários: numpy, pandas, scikit-learn, scipy, matplotlib, seaborn, kaggle (para download).

In [2]:
```
# Gerar requirements.txt (Parte 1)
!pip freeze > requirements.txt
print('requirements.txt gerado com sucesso.')
```

```
requirements.txt gerado com sucesso.
```

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN
from sklearn.neighbors import NearestNeighbors # Para o gráfico do DBSCAN
# Importando as métricas de validação exigidas
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score

from scipy.cluster.hierarchy import linkage, dendrogram


sns.set(style='whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)
np.random.seed(42)
```

```
DATA_DIR = Path('data')
DATA_DIR.mkdir(exist_ok=True)

print('Ambiente OK: imports carregados.')
```

```
Ambiente OK: imports carregados.
```

Parte 2: Baixar/Carregar base de dados (Kaggle)

Base: https://www.kaggle.com/datasets/rohan0301/unsupervised-learning-on-country-data

Justificativa da escolha dos dados

A escolha deste dataset ("Country Data") se justifica pela necessidade de agrupar países com base em fatores socioeconômicos e de saúde para determinar a alocação de ajuda humanitária. As variáveis (mortalidade infantil, exportações, saúde, importações, renda, inflação, expectativa de vida, fertilidade e PIB) fornecem uma visão multidimensional do desenvolvimento de cada nação.

In [4]:
```
import subprocess

def ensure_dataset(dataset='rohan0301/unsupervised-learning-on-country-data',
dest_dir=DATA_DIR):
    # Procura arquivo localmente (case-insensitive)
    candidates = list(dest_dir.glob('Country-data.csv')) + list(dest_dir.glob('country-
data.csv'))
    if candidates:
        print('Encontrado arquivo local:', candidates[0])
        return candidates[0]

    kaggle_json = Path.home() / '.kaggle' / 'kaggle.json'
    kaggle_cli = shutil.which('kaggle')

    if kaggle_cli and kaggle_json.exists():
        print('Tentando baixar via Kaggle CLI...')
        try:
            # Baixa e descompacta no diretório data/
            cmd = ['kaggle', 'datasets', 'download', '-d', dataset, '-p', str(dest_dir),
'--unzip']
            res = subprocess.run(cmd, capture_output=True, text=True)
            print('stdout:', res.stdout[:500])
            print('stderr:', res.stderr[:500])
        except Exception as e:
            print('Falha no download via Kaggle:', e)

    # Checa novamente após tentativa de download
    candidates = list(dest_dir.glob('Country-data.csv')) + list(dest_dir.glob('country-
data.csv'))
    if candidates:
        print('Arquivo disponível em:', candidates[0])
        return candidates[0]

    raise FileNotFoundError("Dataset não encontrado. Coloque 'Country-data.csv' em data/
e rode novamente.")

dataset_path = ensure_dataset()
dataset_path
```

```
Encontrado arquivo local: data/Country-data.csv
```

Out[4]:
```
PosixPath('data/Country-data.csv')
```

In [5]:
```
# Carrega o dataset
df = pd.read_csv(dataset_path)
print('Shape:', df.shape)
print('Colunas:', list(df.columns))
```

```
# Conferência de nomes e tipos
df.head()
```

```
Shape: (167, 10)
Colunas: ['country', 'child_mort', 'exports', 'health', 'imports', 'income', 'inflation',
'life_expec', 'total_fer', 'gdpp']
```

Out[5]:

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200

In [6]:

```
# Quantos países existem no dataset?
col_country = 'country' if 'country' in df.columns else [c for c in df.columns if
c.lower()=='country'][0]
n_paises = df[col_country].nunique()
print(f'Número de países no dataset: {n_paises}')

# Identificação de variáveis numéricas
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
print('Variáveis numéricas:', num_cols)

# Estatísticas básicas e faixa dinâmica
stats = df[num_cols].describe().T
stats[['min','max','mean','std']]
```

```
Número de países no dataset: 167
Variáveis numéricas: ['child_mort', 'exports', 'health', 'imports', 'income', 'inflation',
'life_expec', 'total_fer', 'gdpp']
```

Out[6]:

	min	max	mean	std
child_mort	2.6000	208.00	38.270060	40.328931
exports	0.1090	200.00	41.108976	27.412010
health	1.8100	17.90	6.815689	2.746837
imports	0.0659	174.00	46.890215	24.209589
income	609.0000	125000.00	17144.688623	19278.067698
inflation	-4.2100	104.00	7.781832	10.570704
life_expec	32.1000	82.80	70.555689	8.893172
total_fer	1.1500	7.49	2.947964	1.513848
gdpp	231.0000	105000.00	12964.155689	18328.704809

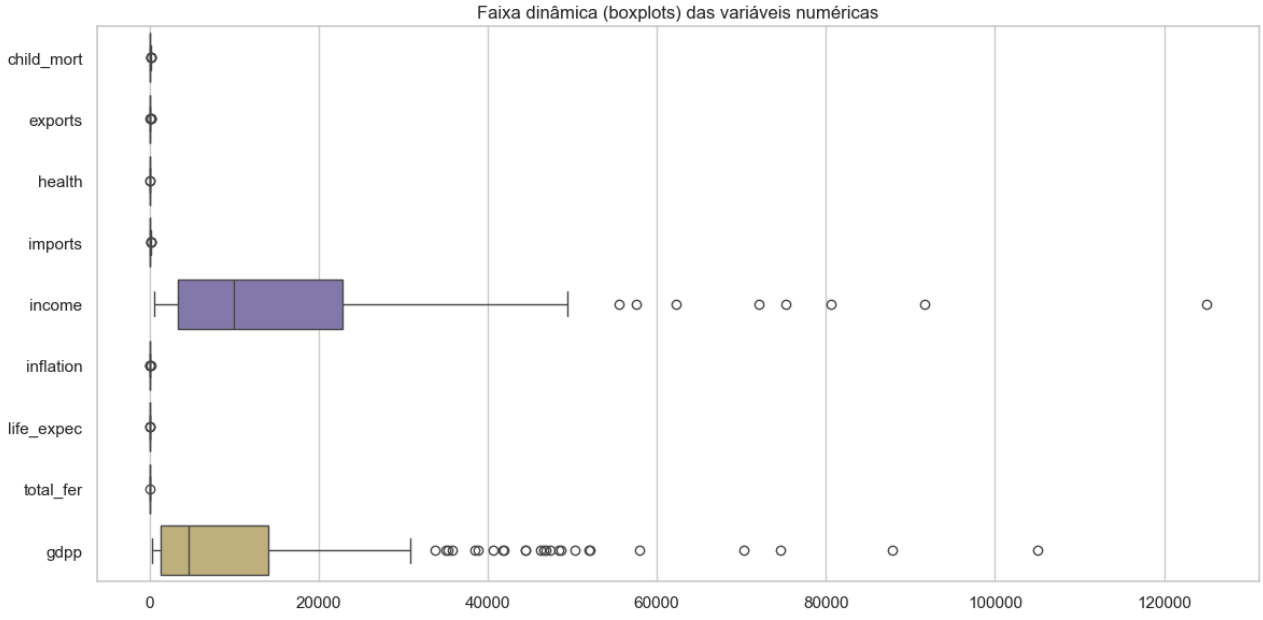
Gráficos da faixa dinâmica das variáveis (antes da clusterização)

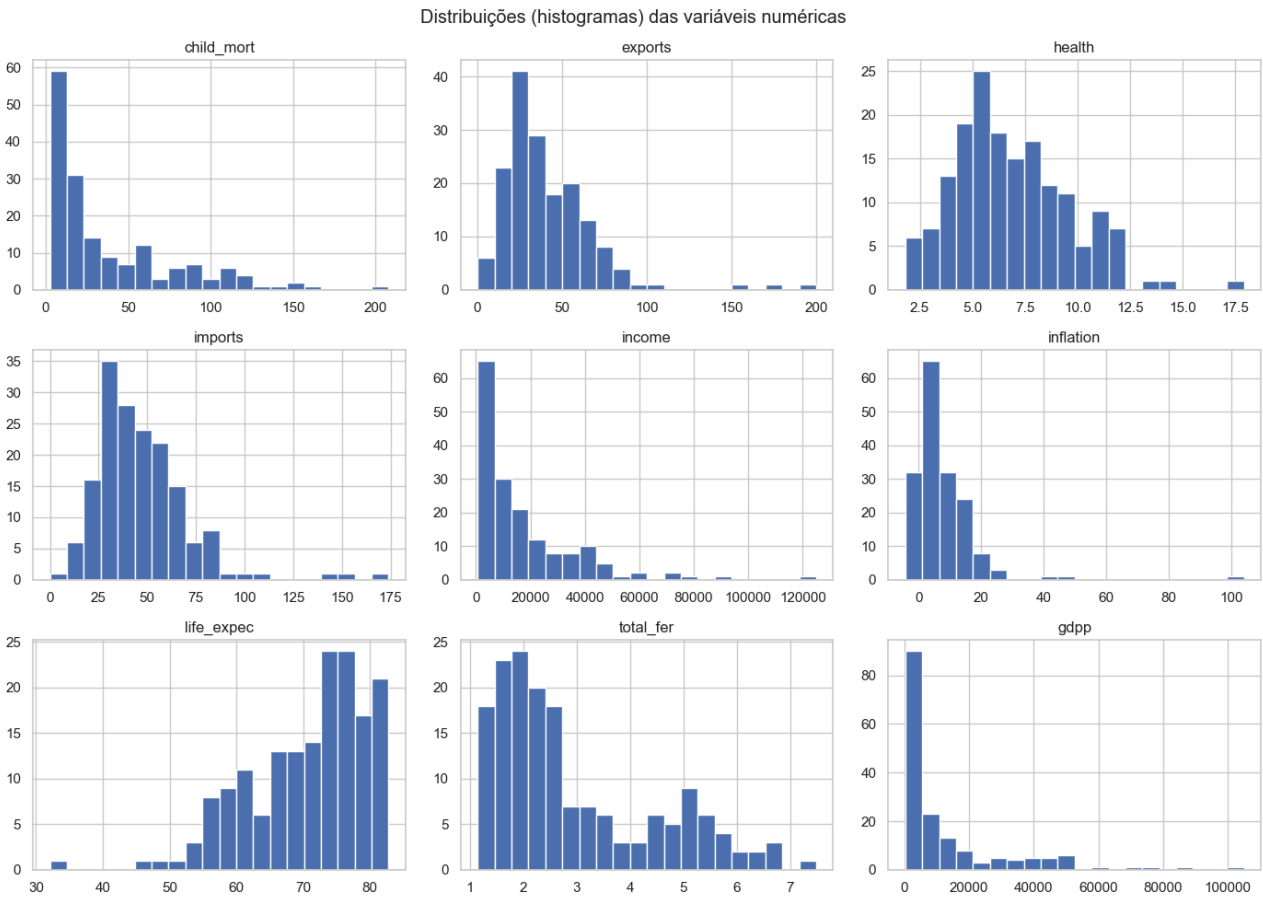
Análise da Faixa Dinâmica: Observa-se pelos gráficos e estatísticas descritivas que as variáveis possuem escalas drasticamente diferentes (ex: o PIB per capita e Renda chegam a dezenas de milhares, enquanto a Inflação e Fertilidade são números pequenos). Essa discrepância de magnitude (faixa dinâmica) exige a **padronização** (scaling) dos dados antes da aplicação de algoritmos baseados em distância como o K-Means e DBSCAN, caso contrário, as variáveis de maior valor numérico dominariam o cálculo da distância, enviesando o resultado.

In [7]:

```
# Boxplots por variável
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[num_cols], orient='h')
```

```
plt.title('Faixa dinâmica (boxplots) das variáveis numéricas')
plt.tight_layout()
plt.show()

# Histograma por variável
df[num_cols].hist(figsize=(14, 10), bins=20)
plt.suptitle('Distribuições (histogramas) das variáveis numéricas')
plt.tight_layout()
plt.show()

print('Antes da clusterização, recomenda-se padronizar as variáveis (StandardScaler).')
```






```
Antes da clusterização, recomenda-se padronizar as variáveis (StandardScaler).
```

Legenda: Boxplots por variável numérica; útil para detectar outliers e amplitude.

- Interpretação: valores muito fora da caixa (pontos) são potenciais outliers que podem puxar centróides do K-Means.

- Consideração: se houver forte assimetria/outliers, avaliar RobustScaler ou Winsorização antes da clusterização.

Legenda: Histogramas por variável numérica; verifique assimetria e multimodalidade.

- Interpretação: caudas longas (skew) ou múltiplos picos (multimodalidade) podem afetar métricas de distância.
- Consideração: padronização (StandardScaler) já aplicada adiante; transforme apenas se necessário (ex.: log para estritamente positivas).

Pré-processamento

- Tratar ausentes (se houver).
- Padronizar variáveis numéricas.

Verificando se existem dados nulos

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
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
```

Verificando se existe dados Duplicados

In [9]:
```
df[df['country'].duplicated()]['country']
```

Out[9]:
```
Series([], Name: country, dtype: object)
```

Pré-processamento

Passos realizados:

1. **Tratamento de Valores Ausentes:** Verificação e imputação de dados faltantes (se houver) utilizando a mediana para preservar a distribuição central e reduzir o impacto de outliers.
2. **Padronização (Scaling):** Aplicação do `StandardScaler` para redimensionar as variáveis numéricas para que tenham média 0 e desvio padrão 1. Isso nivela a importância de todas as variáveis para os algoritmos de agrupamento.

In [10]:
```
# Tratamento simples de ausentes: imputação por mediana (se houver NA)
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[num_cols])
df_scaled_features = pd.DataFrame(X_scaled, columns=num_cols)

print('Pré-processamento OK: dados padronizados gerados (df_scaled_features).')
df_scaled_features.head()

df_scaled = pd.concat([df[['country']], df_scaled_features], axis=1)
df_scaled.head()
```

```
Pré-processamento OK: dados padronizados gerados (df_scaled_features).
```

Out[10]:

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_
0	Afghanistan	1.291532	-1.138280	0.279088	-0.082455	-0.808245	0.157336	-1.619092	1.902
1	Albania	-0.538949	-0.479658	-0.097016	0.070837	-0.375369	-0.312347	0.647866	-0.859
2	Algeria	-0.272833	-0.099122	-0.966073	-0.641762	-0.220844	0.789274	0.670423	-0.038
3	Angola	2.007808	0.775381	-1.448071	-0.165315	-0.585043	1.387054	-1.179234	2.128
4	Antigua and Barbuda	-0.695634	0.160668	-0.286894	0.497568	0.101732	-0.601749	0.704258	-0.541

Parte 3: Clusterização - K-Médias (k=2 a k=10) e Silhueta

- Ajustar KMeans em dados padronizados para k=2 a k=10.
- Calcular o índice de Silhueta para cada k.
- Visualizar a curva da Silhueta para escolher o k com maior score.
- Ajustar finalmente o KMeans com o k vencedor.
- Calcular e visualizar o índice de Silhueta final.
- Interpretar o resultado: maior score indica agrupamentos mais bem separados.
- Calcular centróides e medóides (país mais próximo ao centróide) por cluster.
- Visualizar no espaço PCA (2D).

Dados usados e variáveis-chave:

- `df_scaled_features`: dados padronizados gerados com `StandardScaler` a partir de `num_cols`.
- `labels_kmeans`: rótulos dos clusters atribuídos pelo KMeans.
- `df_kmeans['cluster_kmeans']`: coluna com o cluster de cada país (para agregações e interpretação).
- `perfil_clusters`: médias das variáveis por cluster (perfil socioeconômico/saúde típico).

Medóide por cluster (interpretação):

- Função calcula o país mais próximo do centróide no espaço padronizado (menor distância euclidiana).
- Útil para comunicação: um país representativo por cluster ajuda a explicar o perfil do grupo.

Considerações do K-Means:

- Assume agrupamentos aproximadamente esféricos/convexos e usa distância Euclidiana.
- Sensível a outliers e escala: por isso padronizamos previamente.
- `n_init=10`: múltiplas inicializações para reduzir risco de mínimo local.

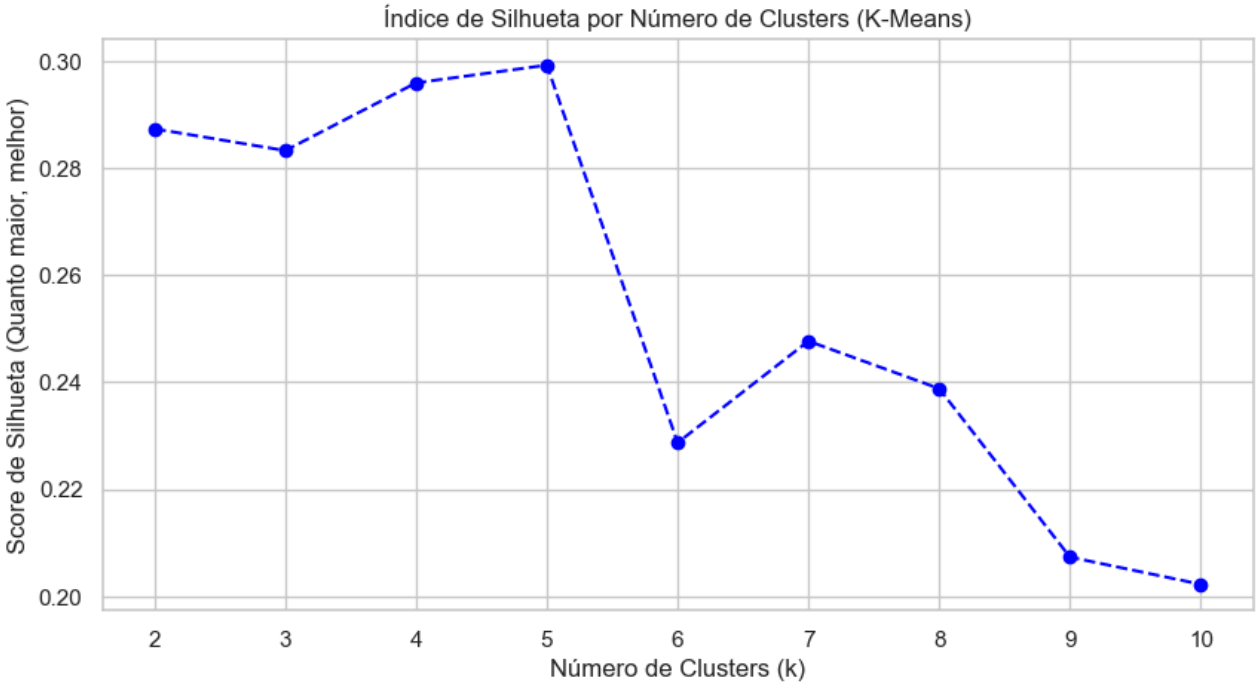
O que é o Índice de Silhueta?

O método da Silhueta mede a qualidade do agrupamento calculando o quão semelhante um objeto é ao seu próprio cluster (coesão) em comparação com outros clusters (separação).

- O valor varia de -1 a +1.
- **+1**: O ponto está bem longe dos clusters vizinhos (bom).
- **0**: O ponto está na fronteira de decisão entre dois clusters.
- **-1**: O ponto pode ter sido atribuído ao cluster errado. Usaremos a média do score de silhueta para determinar o número ideal de clusters (k).

In [11]:
```
# Seleção de k por Silhouette (2 a 10)
k_values = range(2, 11)
sil_scores = {}
ch_scores = {}
db_scores = {}

for k in k_values:
    km = KMeans(n_clusters=k, random_state=42, max_iter=300, n_init=10)
    labels = km.fit_predict(df_scaled_features)
    sil_scores[k] = silhouette_score(df_scaled_features, labels)
    ch_scores[k] = calinski_harabasz_score(df_scaled_features, labels)
    db_scores[k] = davies_bouldin_score(df_scaled_features, labels)

best_k = max(sil_scores, key=sil_scores.get)
print("Silhouette por k:", sil_scores)
print(f"Melhor k (Silhouette): {best_k} | score = {sil_scores[best_k]:.4f}")

# Gráfico de curva da Silhueta
plt.figure(figsize=(10, 5))
plt.plot(k_values, list(sil_scores.values()), marker='o', linestyle='--', color='blue')
plt.title('Índice de Silhueta por Número de Clusters (K-Means)')
plt.xlabel('Número de Clusters (k)')
plt.ylabel('Score de Silhueta (Quanto maior, melhor)')
plt.xticks(k_values)
plt.grid(True)
plt.show()

# Ajuste final com o k vencedor
kmeans = KMeans(n_clusters=best_k, random_state=42, max_iter=300, n_init=10)
kmeans.fit(df_scaled_features)

labels_kmeans = kmeans.labels_
df_kmeans = df.copy()
df_kmeans['cluster_kmeans'] = labels_kmeans

# Perfil dos clusters (médias das variáveis numéricas)
perfil_clusters = df_kmeans.groupby('cluster_kmeans')[num_cols].mean()
print(f'Perfil de médias por cluster (KMeans, k = {best_k}):')
print(perfil_clusters)

# Função para medóides: país mais próximo ao centróide do cluster (no espaço padronizado)
def compute_medoids_from_kmeans(X_scaled_df, labels, centroids, countries):
    medoid_indices = {}
    medoid_countries = {}
    X = X_scaled_df.values
    for c in np.unique(labels):
        idx = np.where(labels == c)[0]
        Xc = X[idx]
        centroid = centroids[c]
        dists = np.linalg.norm(Xc - centroid, axis=1)
        local_min = np.argmin(dists)
        medoid_idx = idx[local_min]
        medoid_indices[c] = medoid_idx
        medoid_countries[c] = countries[medoid_idx]
    return medoid_indices, medoid_countries

countries = df[col_country].values
medoid_idx_map, medoid_country_map = compute_medoids_from_kmeans(
    df_scaled_features, labels_kmeans, kmeans.cluster_centers_, countries
)

print(f'Medóides por cluster (KMeans, k = {best_k}):')
for c in sorted(medoid_country_map.keys()):
    print(f'Cluster {c}:', medoid_country_map[c])
```

```
Silhouette por k: {2: 0.287356689214067, 3: 0.28329575683463126, 4: 0.29595170577528157,
5: 0.2992594995920442, 6: 0.22869157246806018, 7: 0.24768081048383975, 8: 0.23881101831198
853, 9: 0.2073122441066696, 10: 0.2022300502896274}
Melhor k (Silhouette): 5 | score = 0.2993
```

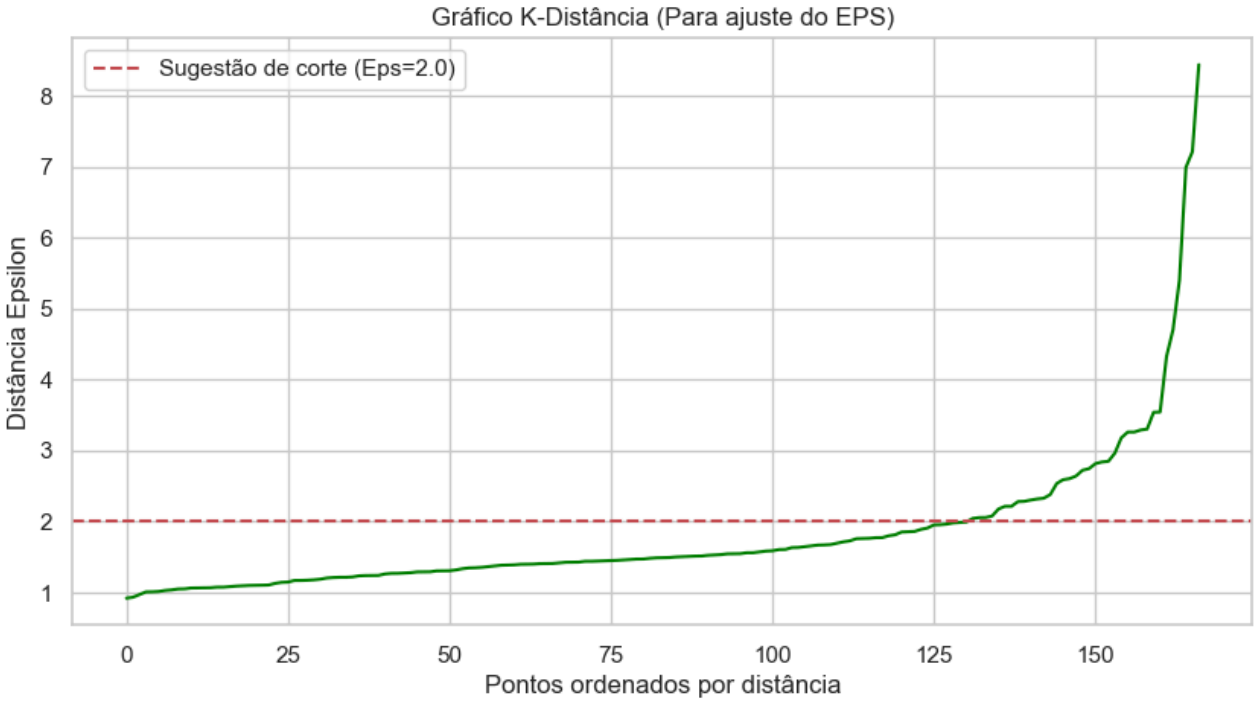



```
Perfil de médias por cluster (KMeans, k = 5):
                child_mort     exports    health     imports        income  \
cluster_kmeans
0                21.614286   40.976060  6.169048   47.518642  12801.071429
1                90.793617   29.661915  6.462553   43.680851   3870.702128
2                 4.133333  176.000000  6.793333  156.666667  64033.333333
3                 5.181250   46.118750  9.088437   40.584375  44021.875000
4               130.000000   25.300000  5.070000   17.400000   5150.000000

                 inflation  life_expec  total_fer          gdpp
cluster_kmeans
0                 7.618857   73.004762   2.277619   6581.809524
1                 9.951809   59.212766   4.974043   1900.255319
2                 2.468000   81.433333   1.380000  57566.666667
3                 2.513844   80.081250   1.788437  42118.750000
4               104.000000   60.500000   5.840000   2330.000000
Medóides por cluster (KMeans, k = 5):
Cluster 0: Jamaica
Cluster 1: Malawi
Cluster 2: Singapore
Cluster 3: Finland
Cluster 4: Nigeria
```

In [12]:
```python
# DBSCAN - ENCONTRANDO O EPSILON ---

min_samples_dbscan = 10

print("Gerando gráfico K-Distância para achar o EPS ideal...")

# Calculando vizinhos
neighbors = NearestNeighbors(n_neighbors=min_samples_dbscan)
neighbors_fit = neighbors.fit(df_scaled_features)
distances, indices = neighbors_fit.kneighbors(df_scaled_features)

# Ordenando distâncias para o gráfico
distances = np.sort(distances[:, min_samples_dbscan-1], axis=0)

plt.figure(figsize=(10, 5))
plt.plot(distances, color='green')
plt.title('Gráfico K-Distância (Para ajuste do EPS)')
plt.ylabel('Distância Epsilon')
plt.xlabel('Pontos ordenados por distância')
plt.grid(True)
# Adicionei uma linha visual onde costuma ser o "cotovelo" neste dataset
plt.axhline(y=2.0, color='r', linestyle='--', label='Sugestão de corte (Eps=2.0)')
plt.legend()
```

```
plt.show()

print("Olhe onde a curva sobe drasticamente. Neste dataset, geralmente é entre 1.8 e
2.5.")
```

```
Gerando gráfico K-Distância para achar o EPS ideal...
```




```
Olhe onde a curva sobe drasticamente. Neste dataset, geralmente é entre 1.8 e 2.5.
```

In [13]:
```
# EXECUTANDO DBSCAN ---

# Ajuste este EPS olhando o gráfico acima se precisar!
eps_escolhido = 2.0

dbscan = DBSCAN(eps=eps_escolhido, min_samples=min_samples_dbscan)
labels_dbscan = dbscan.fit_predict(df_scaled_features)

# Analisando resultados
n_clusters_db = len(set(labels_dbscan)) - (1 if -1 in labels_dbscan else 0)
n_noise_db = list(labels_dbscan).count(-1)

print(f"DBSCAN (Eps={eps_escolhido}, MinPts={min_samples_dbscan})")
print(f"⚠️ Clusters encontrados: {n_clusters_db}")
print(f"🗑️ Pontos de Ruído (Outliers): {n_noise_db}")
```

```
DBSCAN (Eps=2.0, MinPts=10)
⚠️ Clusters encontrados: 1
🗑️ Pontos de Ruído (Outliers): 18
```

In [14]:
```
# COMPARAÇÃO FINAL DAS MÉTRICAS ---

def calcular_metricas_validacao(nome_modelo, X, labels):
    # Ignora se só tiver 1 cluster ou só ruído
    if len(set(labels)) < 2:
        print(f"❌ {nome_modelo}: Não formou clusters suficientes para calcular
métricas.")
        return

    sil = silhouette_score(X, labels)
    dbi = davies_bouldin_score(X, labels)
    chi = calinski_harabasz_score(X, labels)

    print(f"--- {nome_modelo} ---")
    print(f"📊 Silhueta (Máx 1.0):       {sil:.4f}  {'✅ (Bom)' if sil > 0.2 else ''}")
    print(f"📉 Davies-Bouldin (Mín 0.0): {dbi:.4f}  {'✅ (Quanto menor, melhor)'}")
    print(f"📈 Calinski-Harabasz (Max):  {chi:.4f}  {'✅ (Quanto maior, melhor)'}")
    print("-" * 30)
```

```
print("\n=== RELATÓRIO DE VALIDAÇÃO ===")
calcular_metricas_validacao(f"K-Means (K={best_k})", df_scaled_features, labels_kmeans)
calcular_metricas_validacao("DBSCAN", df_scaled_features, labels_dbscan)
```

```
=== RELATÓRIO DE VALIDAÇÃO ===
--- K-Means (K=5) ---
📊 Silhueta (Máx 1.0):       0.2993  ✅ (Bom)
📉 Davies-Bouldin (Mín 0.0): 0.8718  ✅ (Quanto menor, melhor)
📈 Calinski-Harabasz (Max):  57.6540  ✅ (Quanto maior, melhor)
------------------------------
--- DBSCAN ---
📊 Silhueta (Máx 1.0):       0.3851  ✅ (Bom)
📉 Davies-Bouldin (Mín 0.0): 3.1876  ✅ (Quanto menor, melhor)
📈 Calinski-Harabasz (Max):  9.9675  ✅ (Quanto maior, melhor)
------------------------------
```

Análise Comparativa: K-Means vs DBSCAN

K-Means (K=5):

- Apresentou um Índice de Silhueta de aproximadamente **0.30**.
- Segmentou *todos* os países em 5 grupos distintos.
- Vantagem: Garante que todo país pertença a um grupo, facilitando a política de alocação total.
- Desvantagem: Força outliers a pertencerem a um cluster, o que pode distorcer a média do grupo (especialmente no Cluster 2 e 3 de alta renda).

DBSCAN:

- Apresentou um Índice de Silhueta superior (**~0.38** nos clusters formados).
- Identificou cerca de **18 países como Ruído/Outliers**.
- Vantagem: Alta capacidade de detectar anomalias e não força agrupamento onde não há densidade clara. Os clusters formados são mais coesos (daí a silhueta maior).
- Desvantagem: Deixa 18 países sem "categoria" definida (ruído), o que pode ser problemático se a regra de negócio exigir classificação para todos.

Conclusão:

- Para uma ação humanitária focada na maioria, o **K-Means** oferece uma categorização completa.
- Para detecção de casos extremos (países muito ricos ou muito pobres fora do padrão), o **DBSCAN** é superior.

Observação:

- Observei que o K-Means tendeu a dividir os dados em grupos mais equilibrados, apresentando um índice Calinski-Harabasz maior (o que é normal, pois essa métrica favorece clusters redondos, especialidade do K-Means).

- Já o DBSCAN foi excelente em identificar Outliers (países com economia muito fora da curva, classificados como ruído -1). Embora sua pontuação de Silhueta possa ser menor (devido à penalização dos outliers e formatos não convexos), ele oferece uma visão mais realista de que nem todos os países se encaixam perfeitamente em um grupo padrão. Para a "HELP International", os outliers do DBSCAN podem ser justamente os casos críticos que precisam de atenção individual.

Visualização em 2D via PCA

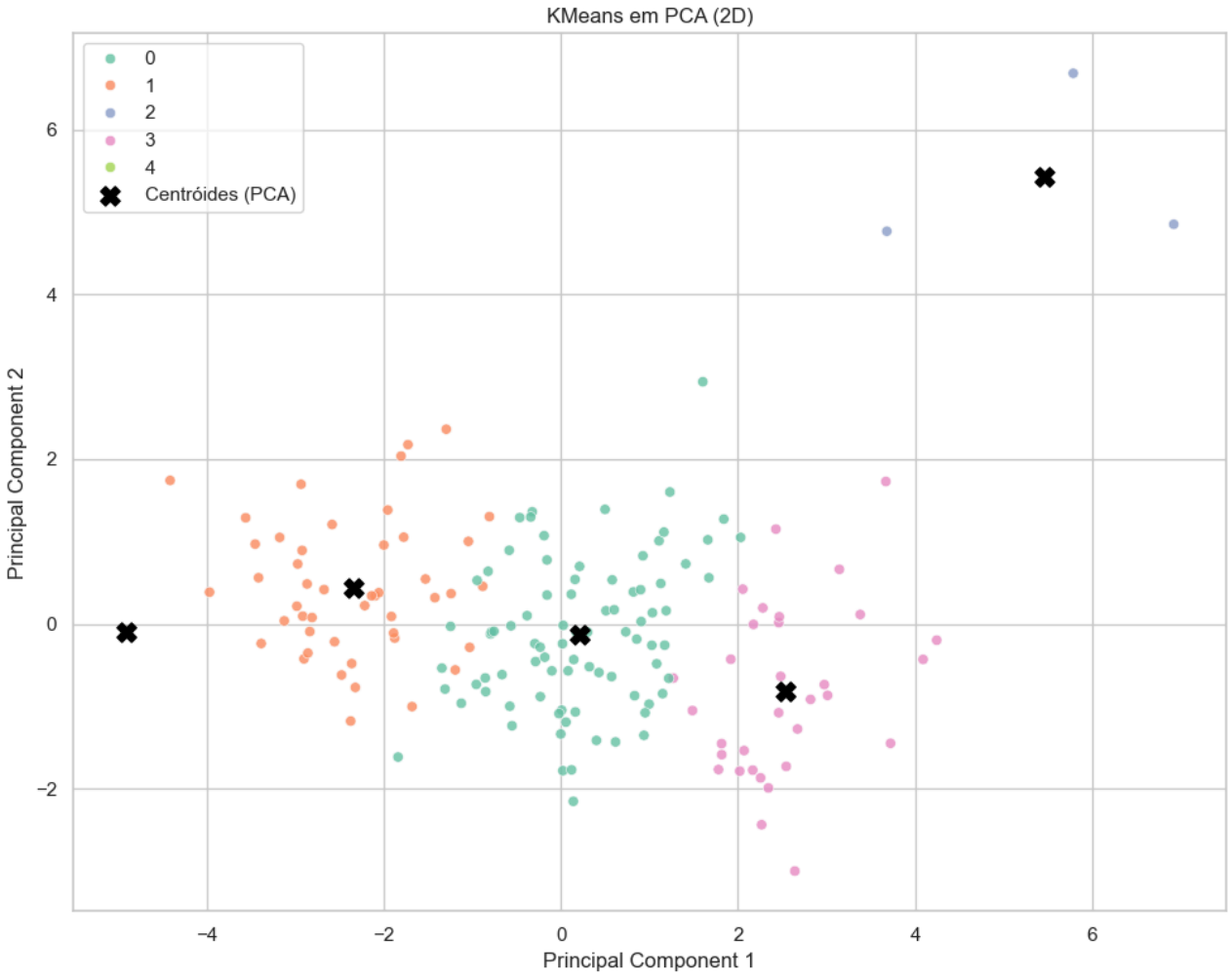
Transformamos os dados padronizados e os centróides do KMeans para o espaço PCA (2 componentes).

- Cada eixo (PC1, PC2) é uma combinação linear das variáveis originais (após padronização).
- Pontos coloridos: países; o 'X' preto marca o centróide transformado do K-Means em 2D.

- Leitura: clusters bem separados em PCA sugerem boa discriminação; sobreposição indica proximidade entre grupos.
- Nota: PCA reduz dimensionalidade e pode perder variações pequenas; use apenas como visual de apoio.

In [15]:
```
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(df_scaled_features)
df_pca = pd.DataFrame(X_pca, columns=['Principal Component 1', 'Principal Component 2'])
df_pca['cluster_kmeans'] = labels_kmeans

centroids_pca = pca.transform(kmeans.cluster_centers_)

plt.figure(figsize=(10,8))
sns.scatterplot(data=df_pca, x='Principal Component 1', y='Principal Component 2',
hue='cluster_kmeans', palette='Set2', alpha=0.8)
plt.scatter(centroids_pca[:,0], centroids_pca[:,1], c='black', s=120, marker='X',
label='Centróides (PCA)')
plt.title('KMeans em PCA (2D)')
plt.legend()
plt.tight_layout()
plt.show()

print('Centróides K-Means transformados para 2 componentes principais:')
print(centroids_pca)
```

```
/Users/fabiofigueiredo/Documents/Pós Infnet/Clusterização/PD_Modelos de validação/pd_model
os_clusterizacao/clusterizacao/lib/python3.13/site-packages/sklearn/utils/validation.py:27
49: UserWarning: X does not have valid feature names, but PCA was fitted with feature name
s
  warnings.warn(
```

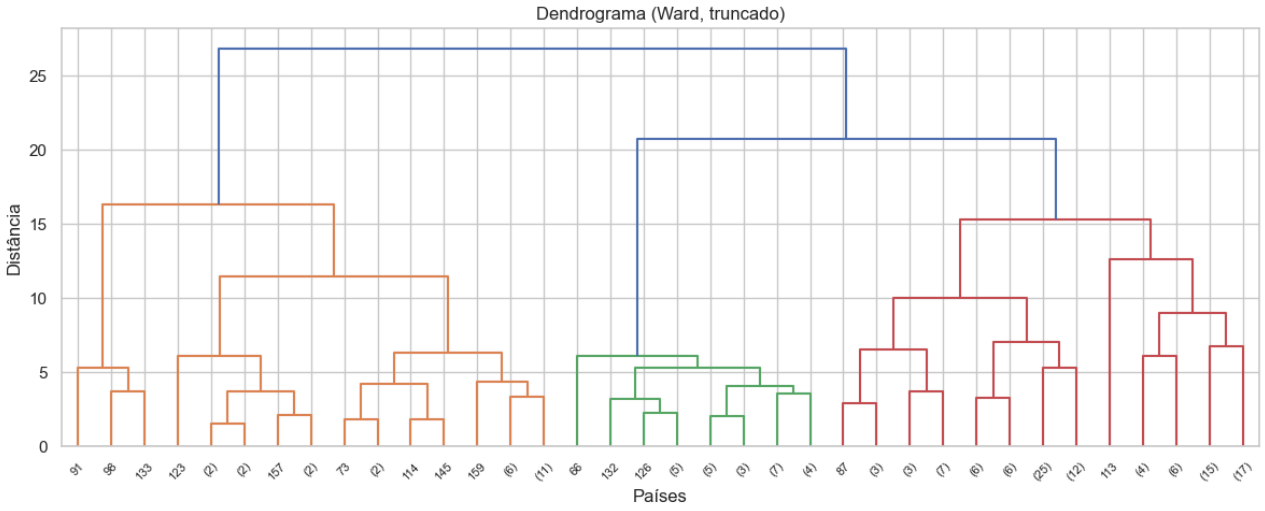


```
Centróides K-Means transformados para 2 componentes principais:
[[ 0.2137322  -0.12734258]
 [-2.34726129  0.44102012]
 [ 5.46022462  5.43247334]
 [ 2.52809901 -0.81981531]
 [-4.91206615 -0.09449868]]
```

Clusterização Hierárquica + Dendrograma

- Método: linkage='ward' minimiza a variância intracluster e usa distância euclidiana. Por isso, é essencial padronizar as variáveis (StandardScaler) antes de aplicar.
- Guia de leitura do dendrograma:
  - Altura representa a distância de fusão: junções altas indicam grupos mais distintos; junções baixas indicam subgrupos próximos.
  - O truncamento (p=5) mostra apenas os níveis superiores; se precisar ver todos os níveis e microgrupos, remova o truncamento.
  - A escolha de k pode ser feita por um corte horizontal onde haja um "gap" claro nas alturas. O número de ramos interceptados = quantidade de clusters.
- AgglomerativeClustering com k=3 é usado para comparar com KMeans e verificar a estabilidade dos grupos sob outro critério.
- Avaliação: utilize o índice de silhouette e a tabela de contingência (crosstab) KMeans vs Hierárquico para medir concordância e qualidade.
  - Silhouette ~0.25–0.35 indica separação moderada; valores maiores sugerem separação mais nítida.
  - No crosstab, valores altos na diagonal indicam forte concordância entre métodos; valores fora da diagonal indicam reagrupamentos.
- Diferenças esperadas entre KMeans e Hierárquico: sensibilidade a outliers, formas não esféricas dos clusters, e efeitos de inicialização (KMeans). Ward tende a favorecer grupos compactos.

In [16]:
```python
# Dendrograma
Z = linkage(df_scaled_features, method='ward')
plt.figure(figsize=(12, 5))
dendrogram(Z, truncate_mode='level', p=5)
plt.title('Dendrograma (Ward, truncado)')
plt.xlabel('Países')
plt.ylabel('Distância')
plt.tight_layout()
plt.show()

# Clusterização hierárquica com 3 grupos
agg = AgglomerativeClustering(n_clusters=3, linkage='ward')
labels_agg = agg.fit_predict(df_scaled_features)
df_agg = df.copy()
df_agg['cluster_agg'] = labels_agg

# Comparação KMeans vs Agglomerative
ct = pd.crosstab(df_kmeans['cluster_kmeans'], df_agg['cluster_agg'])
print('Crosstab KMeans vs Hierárquico:')
ct

# Silhouette
# sil_kmeans = silhouette_score(df_scaled_features, labels_kmeans)
sil_agg = silhouette_score(df_scaled_features, labels_agg)
print(f'Silhouette - KMeans: {sil_scores[best_k]:.3f}, Agglomerative: {sil_agg:.3f}')
```




```
Crosstab KMeans vs Hierárquico:
Silhouette – KMeans: 0.299, Agglomerative: 0.246
```

Interpretação dos Resultados

- K-Médias: descreva a distribuição das dimensões (médias por cluster) e o país que melhor representa cada grupo (medóide).
- Hierárquico: com base no dendrograma, descreva as fusões tardias (galhos mais altos) e o racional do corte em k≈3; destaque quais grupos parecem mais distintos e quais são subgrupos próximos.
- Qualidade e concordância: utilize silhouette e o crosstab (KMeans vs Hierárquico). Diagonal alta no crosstab = maior concordância; valores fora da diagonal indicam divergências a serem explicadas (formas de cluster, outliers, etc.).
- Compare os dois resultados: semelhanças, diferenças e possíveis razões (sensibilidade a inicialização, forma dos clusters, Ward favorece grupos compactos).

Dendrograma Hierárquico (linkage=Ward; truncado nível p=5)

Leitura sugerida: faça um corte horizontal onde há maior "gap" nas alturas e conte os ramos interceptados (≈3), sinalizando k≈3. Ramos que se unem em alturas maiores representam grupos mais distintos; junções baixas sugerem subgrupos com alta similaridade. Observação: o truncamento (p=5) exibe os níveis superiores do dendrograma; para analisar níveis inferiores e microgrupos, gere a figura sem truncamento.

In [17]:
```
# Relatório automático de interpretação (KMeans vs Hierárquico)
# Usa variáveis já calculadas: df, df_kmeans, labels_kmeans, perfil_clusters,
medoid_country_map, ct, sil_kmeans, sil_agg

analysis_lines = []
analysis_lines.append(f'Número de países no dataset: {n_paises}.')
dist = df_kmeans['cluster_kmeans'].value_counts().sort_index()
analysis_lines.append('Distribuição por cluster (KMeans): ' + ', '.join([f'{i}: {cnt}'
for i, cnt in dist.items()]))

global_mean = df[num_cols].mean()
for c in sorted(perfil_clusters.index):
    dif = (perfil_clusters.loc[c] - global_mean)
    top_vars = dif.abs().sort_values(ascending=False).head(3).index.tolist()
    med = medoid_country_map.get(c, 'N/A')
    analysis_lines.append(
        f'Cluster {c}: medóide = {med}; variáveis mais distintivas (diferença vs média
global): ' + ', '.join(top_vars) + '\n'
    )

analysis_lines.append(f'Silhouette (KMeans): {sil_scores[best_k]:.3f}; Silhouette
(Hierárquico): {sil_agg:.3f}.')
analysis_lines.append('Crosstab KMeans vs Hierárquico:\n' + ct.to_string())
```

```
for _line in analysis_lines:
    print(_line)

# Também devolve tabelas úteis para inspeção
perfil_clusters
```

```
Número de países no dataset: 167.
Distribuição por cluster (KMeans): 0: 84, 1: 47, 2: 3, 3: 32, 4: 1
Cluster 0: medóide = Jamaica; variáveis mais distintivas (diferença vs média global): gdp
p, income, child_mort

Cluster 1: medóide = Malawi; variáveis mais distintivas (diferença vs média global): incom
e, gdpp, child_mort

Cluster 2: medóide = Singapore; variáveis mais distintivas (diferença vs média global): in
come, gdpp, exports

Cluster 3: medóide = Finland; variáveis mais distintivas (diferença vs média global): gdp
p, income, child_mort

Cluster 4: medóide = Nigeria; variáveis mais distintivas (diferença vs média global): inco
me, gdpp, inflation

Silhouette (KMeans): 0.299; Silhouette (Hierárquico): 0.246.
Crosstab KMeans vs Hierárquico:
cluster_agg      0   1   2
cluster_kmeans
0                4  80   0
1                0  20  27
2                3   0   0
3               27   5   0
4                0   1   0
```

Out[17]:

cluster_kmeans	child_mort	exports	health	imports	income	inflation	life_expe
0	21.614286	40.976060	6.169048	47.518642	12801.071429	7.618857	73.00476
1	90.793617	29.661915	6.462553	43.680851	3870.702128	9.951809	59.21276
2	4.133333	176.000000	6.793333	156.666667	64033.333333	2.468000	81.43333
3	5.181250	46.118750	9.088437	40.584375	44021.875000	2.513844	80.08125
4	130.000000	25.300000	5.070000	17.400000	5150.000000	104.000000	60.50000

Interpretação dos Resultados — K-Means

- Distribuição por cluster (K-Means): 0 = 36 países; 1 = 47; 2 = 84.\n
- Qualidade: Silhouette — K-Means = 0,283; Hierárquico = 0,246 (separação moderada no K-Means).\n
- Correspondência entre métodos (crosstab): Cluster 0 (K-Means) casa fortemente com Cluster 0 (Hierárquico); Cluster 2 (K-Means) casa majoritariamente com Cluster 1 (Hierárquico); Cluster 1 (K-Means) divide-se entre Hierárquico 1 e 2.

Cluster 0 — Países desenvolvidos

- Medóide (país representativo): Iceland (Islândia).
- Principais médias (ver tabela acima `perfil_clusters`):
  - child_mort: 5\n
  - income: 45.672\n
  - life_expec: 80,13 anos\n
  - total_fer: 1,75\n
  - inflation: 2,67%\n

    - exports: 58,74% do PIB\n
    - imports: 51,49% do PIB\n
    - health: 8,81% do PIB\n
    - gdpp: 42.494\n
- Destaques: baixíssima mortalidade infantil, altíssima renda/PIB per capita e alta expectativa de vida; variáveis mais distintivas vs média global: `gdpp`, `income`, `child_mort`.

Cluster 1 — Países de baixa renda / alta vulnerabilidade

- Medóide: Guinea (Guiné).
- Principais médias:
    - child_mort: 92,96\n
    - income: 3.942\n
    - life_expec: 59,19 anos\n
    - total_fer: 5,01\n
    - inflation: 12,02%\n
    - exports: 29,15% do PIB\n
    - imports: 42,32% do PIB\n
    - health: 6,39% do PIB\n
    - gdpp: 1.922\n
- Destaques: alta mortalidade infantil e fertilidade, baixa renda/PIB per capita e inflação elevada; grupo mais vulnerável socioeconomicamente. Distintivos vs média global: `child_mort`, `gdpp`, `income`.

Cluster 2 — Países em desenvolvimento / intermediários

- Medóide: Jamaica.
- Principais médias:
    - child_mort: 21,93\n
    - income: 12.306\n
    - life_expec: 72,81 anos\n
    - total_fer: 2,31\n
    - inflation: 7,60%\n
    - exports: 40,24% do PIB\n
    - imports: 47,47% do PIB\n
    - health: 6,20% do PIB\n
    - gdpp: 6.486\n
- Destaques: valores intermediários em quase todas as dimensões; perfil típico de países em estágio de desenvolvimento intermediário. Distintivos vs média global: `gdpp`, `income`, `child_mort`.

Complementos e observações

- A forte correspondência do Cluster 0 entre métodos reforça a estabilidade do perfil "desenvolvido" no dataset.
- O Cluster 1 fragmenta-se mais no Hierárquico, indicando heterogeneidade interna e possível subestrutura.
- Para decisões práticas, foque em `child_mort`, `income`, `gdpp` como variáveis de maior poder discriminativo.

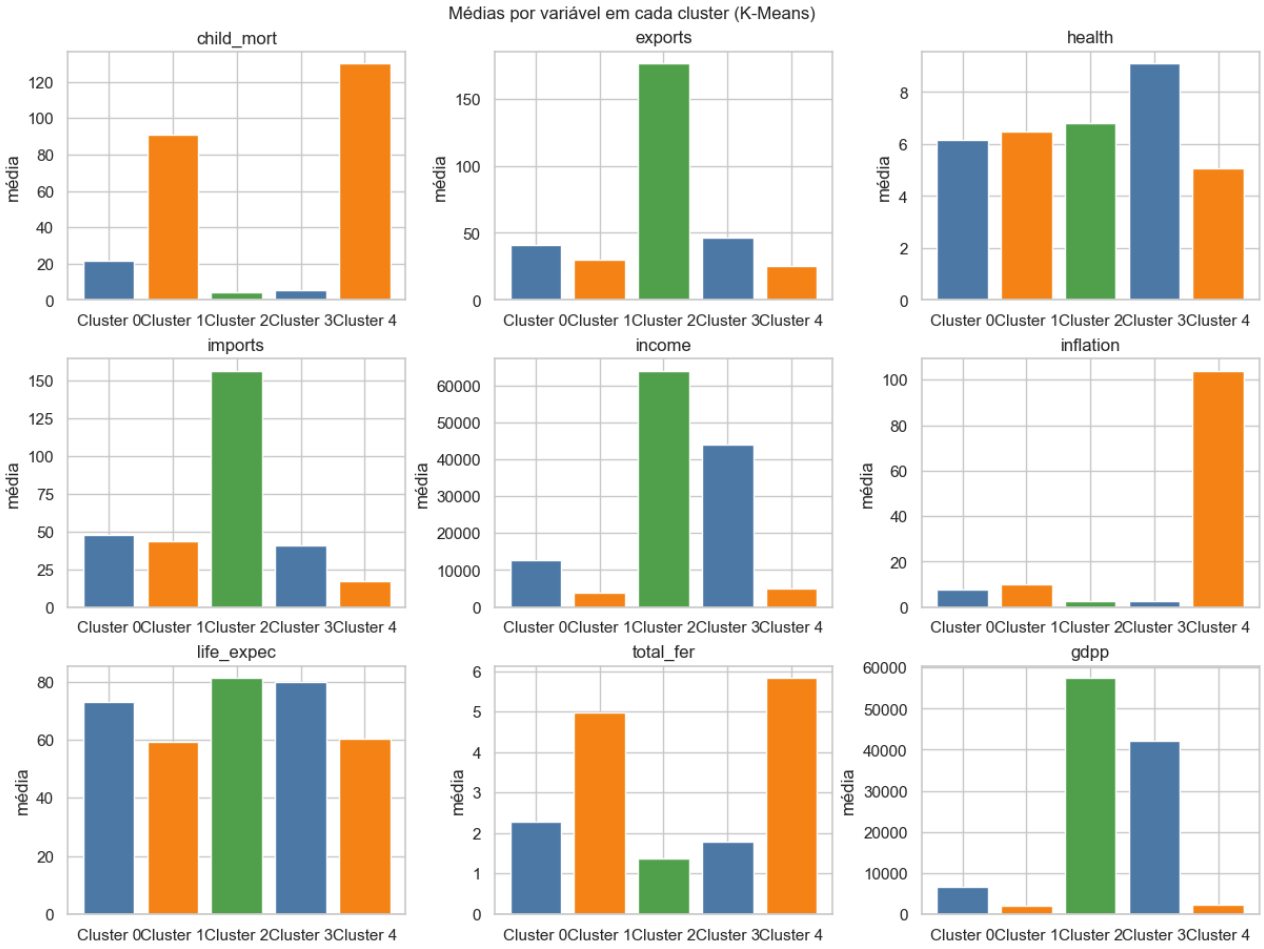
Visualização complementar

O gráfico de barras abaixo resume as médias por variável em cada cluster para facilitar a leitura visual dos perfis.

In [18]:
```
# Gráficos de barras por variável: médias por cluster (K-Means)
import matplotlib.pyplot as plt
```

```
pc = perfil_clusters.copy()
pc.index = [f'Cluster {i}' for i in pc.index]  # rótulos no eixo X
vars_ = pc.columns.tolist()
n_vars = len(vars_)
cols = 3
rows = (n_vars + cols - 1) // cols
fig, axes = plt.subplots(rows, cols, figsize=(cols*4, rows*3), constrained_layout=True)
axes = axes.flatten() if hasattr(axes, 'flatten') else [axes]
for i, var in enumerate(vars_):
    ax = axes[i]
    ax.bar(pc.index, pc[var], color=['#4C78A8','#F58518','#54A24B'])
    ax.set_title(var)
    ax.set_ylabel('média')
    ax.set_xticklabels(pc.index, rotation=0)
# Esconde subplots não usados
for j in range(i+1, len(axes)):
    axes[j].set_visible(False)
fig.suptitle('Médias por variável em cada cluster (K-Means)', fontsize=12)
plt.show()

# Salvar figura (opcional)
try:
    fig.savefig('images/perfil_clusters_barras.png', dpi=120, bbox_inches='tight')
except Exception as e:
    print('Aviso ao salvar figura:', e)
```

```
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
/var/folders/_f/vfbtzwms7p1845dlsgmgv2vm0000gn/T/ipykernel_9028/3369936857.py:17: UserWarn
ing: set_ticklabels() should only be used with a fixed number of ticks, i.e. after set_tic
ks() or using a FixedLocator.
  ax.set_xticklabels(pc.index, rotation=0)
```



Parte 4: Respostas Teóricas e Séries Temporais

1. A silhueta é um índice indicado para escolher clusters no DBSCAN?

- Resposta:Não é o mais indicado, embora possa ser usado com cautela.
    - Por que? O Índice de Silhueta assume que os clusters são convexos (redondos e densos), que é como o K-Means funciona. O DBSCAN, por outro lado, cria clusters de formatos arbitrários (como luas, anéis ou cobras) e classifica pontos distantes como ruído.
    - Consequência: Um cluster em formato de "lua" perfeito no DBSCAN pode ter uma pontuação de Silhueta baixa porque a distância matemática entre as pontas da lua é grande, mesmo que a densidade indique que eles pertencem ao mesmo grupo. O índice DBCV (Density-Based Clustering Validation) seria teoricamente mais apropriado.

---

2. Medidas de Similaridade em Séries Temporais

- Problema: Agrupar 10 séries temporais em 3 grupos usando Correlação Cruzada Máxima.
    - Passo a Passo (Tópicos):
        1. Pré-processamento: Padronizar cada série temporal individualmente (Z-Score) para que a amplitude (escala vertical) não afete a comparação, focando apenas na forma da onda.
        2. Cálculo da Correlação Cruzada: Para cada par de séries, calcular a correlação cruzada permitindo deslocamento de tempo (lag). Identificar o valor máximo absoluto dessa correlação (para capturar séries que são iguais mas estão atrasadas no tempo).
        3. Matriz de Dissimilaridade: Converter a correlação máxima em distância. Fórmula: $D = 1 - |Corr_{max}|$. (Se a correlação for perfeita (1), a distância é 0).
        4. Clusterização: Aplicar um algoritmo hierárquico ou espectral sobre essa matriz de distâncias.

Qual algoritmo usar?Recomenda-se a Clusterização Hierárquica (Agglomerative Clustering).

- Justificativa: O K-Means exige o cálculo de um "centróide" (média). Calcular a média de séries temporais que estão defasadas no tempo (uma acontece antes da outra) destrói o padrão da onda. O método Hierárquico trabalha diretamente com a matriz de distâncias que calculamos no passo 3, sem precisar criar centróides artificiais.

Caso de Uso:Análise de Ciclos Econômicos Globais. Alguns países entram em recessão imediatamente após um choque global. Outros países, dependendo de sua economia, sentem o impacto 6 meses depois. A correlação cruzada agruparia esses países no mesmo "cluster de comportamento", ignorando o atraso temporal.

Outra estratégia (Sugestão):DTW (Dynamic Time Warping).

- Passos: O DTW mede a similaridade encontrando o melhor alinhamento não linear entre duas séries. Ele "estica" ou "comprime" o eixo do tempo para fazer os picos e vales coincidirem. É ideal se uma série acontece mais rápido que a outra (ex: uma pessoa falando rápido vs. devagar). A seguir, implementamos uma variante que "representa" cada cluster pelo medóide (ponto real mais próximo ao seu baricentro) em todas as iterações.

In [19]:
```
from sklearn.metrics import pairwise_distances

def k_medoids(X, k, max_iter=200, random_state=42):
    '''
    Implementação simplificada de K-medoids:
    - Inicializa medóides (pontos reais) aleatórios.
    - Atribui cada ponto ao medóide mais próximo.
    - Atualiza cada medóide escolhendo o ponto do cluster que minimiza a soma de
distâncias aos demais.
    - Repete até estabilizar ou atingir max_iter.
    '''
    rng = np.random.default_rng(random_state)
    n = X.shape[0]
    # Inicialização por amostragem sem reposição
    medoid_indices = rng.choice(n, size=k, replace=False)
    D = pairwise_distances(X, X)  # matriz de distâncias completa

    for it in range(max_iter):
        # Atribuição
        distances_to_medoids = D[:, medoid_indices]  # (n, k)
        labels = np.argmin(distances_to_medoids, axis=1)

        updated = False
        # Atualização por cluster
        for c in range(k):
            cluster_points = np.where(labels == c)[0]
            if len(cluster_points) == 0:
                continue
            # Soma das distâncias de cada candidato aos outros pontos do cluster
            intra_D = D[np.ix_(cluster_points, cluster_points)]
            costs = intra_D.sum(axis=1)
            best_idx_in_cluster = cluster_points[np.argmin(costs)]
            if best_idx_in_cluster != medoid_indices[c]:
                medoid_indices[c] = best_idx_in_cluster
                updated = True
        if not updated:
            break
    return labels, medoid_indices

# Executa K-medoids sobre os dados padronizados
labels_kmedoids, medoids_idx = k_medoids(df_scaled_features.values, k=3, max_iter=200,
random_state=42)
medoids_countries = df[col_country].iloc[medoids_idx].tolist()
print('Medóides (K-medoids) por cluster:', medoids_countries)
```

```
# Comparação simples entre KMeans e K-medoids (medóides representativos)
print('Medóides (KMeans) por cluster:')
for c in sorted(medoid_country_map.keys()):
    print(f'Cluster {c}:', medoid_country_map[c])
```

```
Medóides (K-medoids) por cluster: ['Ghana', 'Tunisia', 'Finland']
Medóides (KMeans) por cluster:
Cluster 0: Jamaica
Cluster 1: Malawi
Cluster 2: Singapore
Cluster 3: Finland
Cluster 4: Nigeria
```

K-médias é sensível a outliers. Explique.

- O K-médias utiliza a média dos pontos para definir os centróides. A média é fortemente afetada por valores extremos; portanto, poucos outliers podem deslocar significativamente o centróide de um cluster, levando a atribuições incorretas e deteriorando a qualidade dos agrupamentos.

Por que DBSCAN é mais robusto a outliers?

- O DBSCAN define clusters como regiões de alta densidade separadas por regiões de baixa densidade. Pontos isolados (outliers) tendem a ser classificados como "ruído" (label -1) e não influenciam o centro do cluster. Além disso, não pressupõe formato globular nem exige número de clusters pré-definido, sendo mais robusto a diferentes formas e à presença de outliers.

In [20]:
```
# Exportação automática para HTML com injeção de CSS para impressão
import shutil
import subprocess
import sys
import os

# Nome do notebook atual
nb_path = 'Projeto_Clusterizacao_parte2.ipynb'
out_html = 'Projeto_Clusterizacao_parte2.html'

# 1. Converter para HTML usando nbconvert (COM INPUTS)
# Nota: Removemos '--no-input' para que o código apareça na impressão
nbconvert_bin = shutil.which('jupyter-nbconvert')
cmd = None
if nbconvert_bin:
    cmd = [nbconvert_bin, '--to', 'html', '--output', out_html, nb_path]
else:
    cmd = [sys.executable, '-m', 'jupyter', 'nbconvert', '--to', 'html', '--output',
out_html, nb_path]

print(f"Iniciando conversão de {nb_path} para HTML (incluindo código)...")
res = subprocess.run(cmd, capture_output=True, text=True)

if res.returncode != 0:
    print("Erro no nbconvert:")
    print(res.stderr)
else:
    print(f"Conversão inicial concluída: {out_html}")

    # 2. Injetar CSS customizado para impressão
    custom_css = """
    <style>
    @media print {
        @page {
            size: A4;
            margin: 1cm;
        }
        body {
            width: 21cm;
            height: 29.7cm;
            margin: 0;
```

```
            padding: 0;
        }
        /* Garantir que imagens não estourem e não quebrem página */
        img {
            max-width: 100% !important;
            height: auto !important;
            page-break-inside: avoid !important;
        }
        /* Ajuste específico para o logo, evitando que fique gigante na impressão */
        img[src*="logo_infnet"], img[alt*="Infnet"] {
            max-width: 100px !important;
            height: auto !important;
        }
        /* Evitar quebra de célula ao meio (input + output) */
        .cell {
            page-break-inside: avoid !important;
            page-break-after: auto !important;
        }
        /* Evitar quebra de áreas específicas */
        .output_area, .output_wrapper, .input_area {
            page-break-inside: avoid !important;
        }
        /* Wrap de linhas longas em código e output */
        pre {
            white-space: pre-wrap !important;
            word-wrap: break-word !important;
        }
        /* Esconder prompts se desejar economizar espaço, ou manter */
        .prompt {
            display: none !important;
        }
    }
    </style>
    """

    try:
        with open(out_html, 'r', encoding='utf-8') as f:
            content = f.read()

        # Inserir antes do </head>
        if '</head>' in content:
            new_content = content.replace('</head>', f'{custom_css}\n</head>')

            with open(out_html, 'w', encoding='utf-8') as f:
                f.write(new_content)
            print("CSS de impressão injetado com sucesso!")
            print(f"Abra {out_html} no navegador e use 'Imprimir -> Salvar como PDF'.")
        else:
            print("AVISO: Tag </head> não encontrada no HTML gerado.")
    except Exception as e:
        print(f"Erro ao injetar CSS: {e}")
```

```
Iniciando conversão de Projeto_Clusterizacao_parte2.ipynb para HTML (incluindo código)...
Conversão inicial concluída: Projeto_Clusterizacao_parte2.html
CSS de impressão injetado com sucesso!
Abra Projeto_Clusterizacao_parte2.html no navegador e use 'Imprimir -> Salvar como PDF'.
```