Projet 11 Etude de marché pour la poule qui chante

OBJECTIFS DE CE NOTEBOOK

- Mettre en œuvre un algorithme K-means et une Classification Ascendante Hiérarchique (CAH)

- Réaliser une Analyse en Composantes Principales (ACP) combinée à une analyse de clustering

- Identifier la segmentation de pays la plus cohérente et significative pour notre étude de marché

Sommaire

Etape 1 - Importation des librairies et chargement des données

1.1 - Importation des librairies

In [6]:
```
# Importation de la librairie Pandas
import pandas as pd
# Importation de la librairie Matplotlib
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib.collections import LineCollection
# Importation de la librairie Seaborn
import seaborn as sns
# Importation de la librairie Plotly
import plotly.express as px
# Importation de la librairie Numpy
import numpy as np
#  Importation K-means
from sklearn.preprocessing import MaxAbsScaler
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
import plotly.graph_objects as go
#   Importation PCA
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
#  Importation CAH
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
```

1.2 - chargement des données

In [8]:
```
df = pd.read_csv("donnees_finale.csv")
```

In [9]:
```
df
```

Out[9]:

	pays	population	production_hab	importation_hab	disponibilite_hab	pib_ha
0	Afrique du Sud	57792000	28.84	8.89	36.65	1475
1	Albanie	2882000	4.51	13.19	16.31	1944
2	Algérie	42228000	6.51	0.05	6.56	1583
3	Allemagne	83124000	18.21	10.13	20.92	6758
4	Angola	30809000	1.36	8.99	10.35	792
...	...	...	...	...	...	
105	Uruguay	3449000	9.57	0.87	9.57	3300
106	Viet Nam	95545000	9.61	3.05	12.62	1390
107	Zambie	17351000	2.82	0.69	3.46	384
108	Équateur	17084000	19.90	0.00	19.96	1519
109	Îles Salomon	652000	0.00	9.20	4.60	269

110 rows × 9 columns

1.3 - Scaling des données

Dans un premier temps on va realiser un ACP pour cela on va concerver uniquement les valeurs numeriques et nous allons standardiser les données grâce à StandardScaler()

In [12]:
```
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Sélectionner uniquement les colonnes numériques depuis df
colonnes_numeriques = df.select_dtypes(include=["number"])

# Sauvegarder la colonne "pays" à part
pays = df["pays"]

# Application du StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(colonnes_numeriques)

# Conversion en DataFrame avec les mêmes noms de colonnes
X_scaled = pd.DataFrame(X_scaled, columns=colonnes_numeriques.columns)

# Arrondir les valeurs pour plus de lisibilité
X_scaled = X_scaled.round(2)

# Ajouts de la colonne "Pays" pour conserver l'identité des pays
```

```
X_scaled.insert(0, "pays", df["pays"])

# Afficher un aperçu
print("\nStatistiques descriptives :")
print(X_scaled.describe())
```

```
Statistiques descriptives :
       population  production_hab  importation_hab  disponibilite_hab  \
count  110.000000    1.100000e+02       110.000000       1.100000e+02
mean    -0.000091   -2.422305e-17         0.000091       4.037175e-18
std      1.004042    1.004645e+00         1.004805       1.004691e+00
min     -0.310000   -1.050000e+00        -0.690000      -1.240000e+00
25%     -0.280000   -9.075000e-01        -0.667500      -8.925000e-01
50%     -0.230000   -1.900000e-01        -0.390000      -8.500000e-02
75%     -0.070000    5.675000e-01         0.177500       6.100000e-01
max      9.760000    3.210000e+00         5.040000       3.340000e+00

          pib_hab   stabilite  indice_perf_logistique      douane
count  110.000000  110.000000            1.100000e+02  110.000000
mean    -0.000364    0.001273            1.211152e-17   -0.000364
std      1.004921    1.004558            1.003980e+00    1.004578
min     -1.060000   -3.330000           -1.660000e+00   -1.040000
25%     -0.840000   -1.070000           -8.900000e-01   -0.820000
50%     -0.310000    0.050000           -1.800000e-01   -0.405000
75%      0.585000    1.180000            7.100000e-01    0.642500
max      4.060000    1.180000            2.140000e+00    3.270000
```

In [13]:
```
#Affichage des données standardisé avec les pays
X_scaled.head()
```

Out[13]:

	pays	population	production_hab	importation_hab	disponibilite_hab	pib_hab
0	Afrique du Sud	0.12	0.59	0.12	0.96	-0.54
1	Albanie	-0.29	-0.80	0.51	-0.27	-0.36
2	Algérie	0.01	-0.68	-0.69	-0.87	-0.50
3	Allemagne	0.31	-0.02	0.23	0.00	1.46
4	Angola	-0.08	-0.98	0.13	-0.64	-0.80

In [14]:
```
#Affichage des données standardisé sans les pays
X_scaled_no_pays = X_scaled.drop(columns=["pays"])
X_scaled_no_pays
```

Out[14]:

	population	production_hab	importation_hab	disponibilite_hab	pib_hab	stabilite
0	0.12	0.59	0.12	0.96	-0.54	-1.07
1	-0.29	-0.80	0.51	-0.27	-0.36	0.05
2	0.01	-0.68	-0.69	-0.87	-0.50	-1.07
3	0.31	-0.02	0.23	0.00	1.46	1.18
4	-0.08	-0.98	0.13	-0.64	-0.80	-1.07
...	...	...	...	...	...	...
105	-0.28	-0.51	-0.61	-0.68	0.15	1.18
106	0.40	-0.51	-0.41	-0.50	-0.57	0.05
107	-0.18	-0.89	-0.63	-1.05	-0.96	0.05
108	-0.18	0.08	-0.69	-0.05	-0.53	0.05
109	-0.30	-1.05	0.15	-0.98	-1.00	0.05

110 rows × 8 columns

Étape 2 - Détermination du nombre optimal de clusters

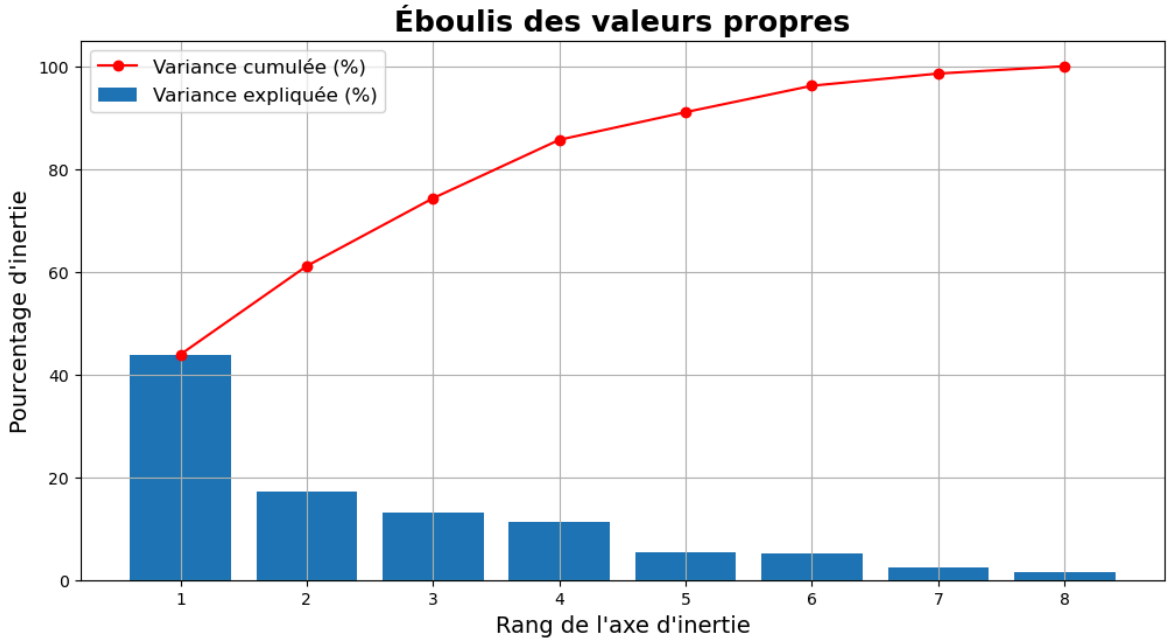
2.1 Réduction de dimension (ACP)

L'Analyse en Composantes Principales (ACP) est une méthode de réduction de dimensionnalité. Elle transforme des variables corrélées en nouvelles variables non corrélées (composantes principales). L'ACP permet de résumer l'information.

In [18]:
```
# Calcul de la variance expliquée et cumulée
# Appliquer ACP
pca = PCA()
pca.fit(X_scaled_no_pays)

scree = pca.explained_variance_ratio_ * 100
scree_cum = np.cumsum(scree)

# Définir la liste des composantes
composantes = [f"{i+1}" for i in range(len(scree))]

# Création d'un DataFrame pour visualiser la variance
df_scree = pd.DataFrame({
    "Composante": composantes,
    "Variance expliquée (%)": scree,
    "Variance cumulée (%)": scree_cum
})
print(df_scree.round(2))
```

```
# Création du graphique
plt.figure(figsize=(12,6))
plt.bar(composantes, scree, label="Variance expliquée (%)")
plt.plot(composantes, scree_cum, c="red", marker='o', label="Variance cumulée (%
plt.xlabel("Rang de l'axe d'inertie", fontsize=14)
plt.ylabel("Pourcentage d'inertie", fontsize=14)
plt.title("Éboulis des valeurs propres", fontsize=18, fontweight="bold")
plt.legend(fontsize=12)
plt.grid()

# Affichage du graphique
plt.show()
```

	Composante	Variance expliquée (%)	Variance cumulée (%)
0	1	43.89	43.89
1	2	17.21	61.10
2	3	13.20	74.31
3	4	11.38	85.68
4	5	5.38	91.06
5	6	5.14	96.20
6	7	2.37	98.58
7	8	1.42	100.00



On remarque ici que plus de 80% de la variance est comprise dans les 4 premiers composants et 90% se trouve à la 5eme. On va donc choisir 4 composantes.

In [20]:
```
# Appliquer l'ACP avec 4 composantes sur les données sans la colonne "pays"
pca = PCA(n_components=4, random_state=42)
X_pca = pca.fit_transform(X_scaled_no_pays)

# Créer un DataFrame avec les 4 composantes principales
df_pca = pd.DataFrame(
    X_pca,
    columns=["PC1", "PC2", "PC3", "PC4"],
    index=X_scaled_no_pays.index
)

# Aperçu du DataFrame
print(df_pca.head())
```

```
# (Optionnel) Afficher la variance expliquée
print("\nVariance expliquée par composante :", pca.explained_variance_ratio_)
print("Variance totale expliquée :", round(pca.explained_variance_ratio_.sum() *
```

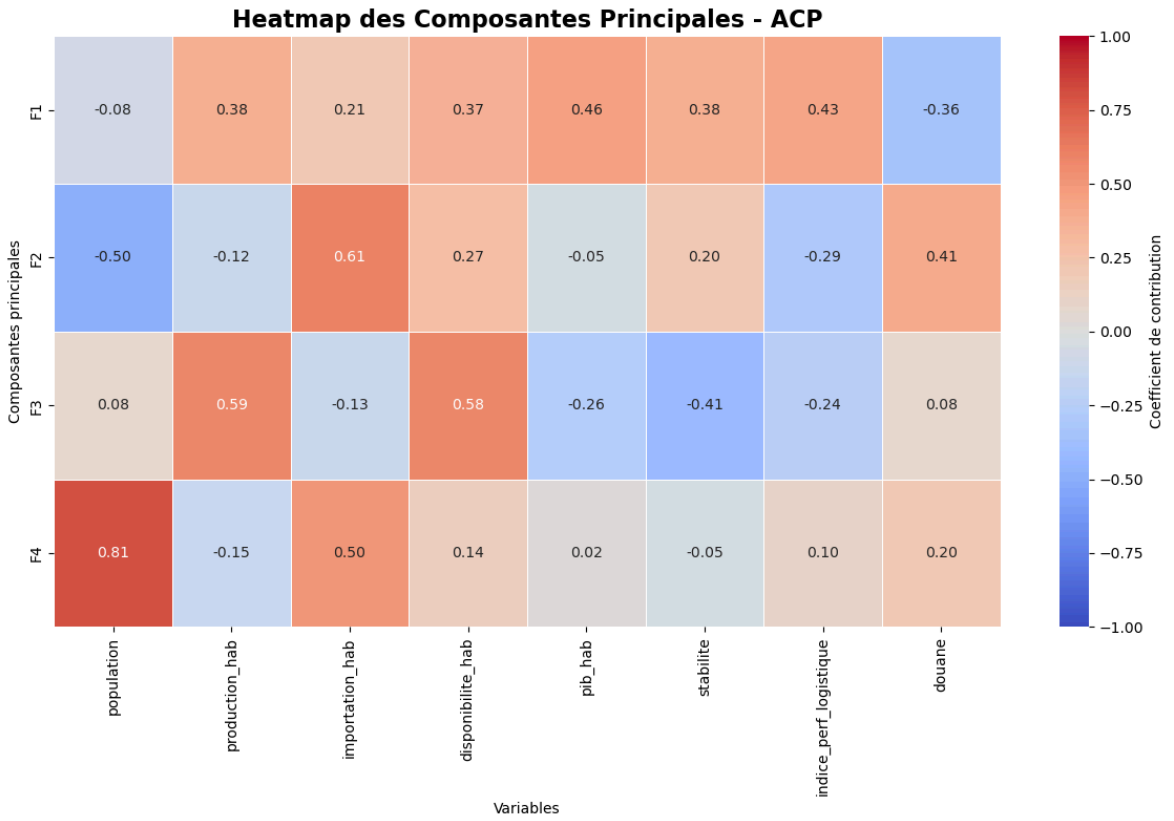
```
        PC1       PC2       PC3       PC4
0  0.524131 -0.379132  1.168612  0.358733
1 -0.436653  0.334393 -0.509343 -0.201380
2 -2.050733 -0.156346  0.034434 -0.243796
3  2.271623 -0.756096 -1.407575  0.373367
4 -2.320519  0.670917  0.117518  0.078474

Variance expliquée par composante : [0.43891712 0.17211053 0.13202334 0.11378077]
Variance totale expliquée : 85.68 %
```

In [21]:
```
# Matrice des composantes principales du PCA
pcs = pca.components_
pcs = pd.DataFrame(pcs)

# Attribution des noms de colonnes et d'index aux composantes principales
pcs.columns = X_scaled_no_pays.columns
pcs.index = [f"F{i+1}" for i in range(pca.n_components_)]

print("Matrice des composantes principales (Loadings) :")
print(f"Dimensions : {pcs.shape[0]} composantes x {pcs.shape[1]} variables")
display(pcs.round(3))
```

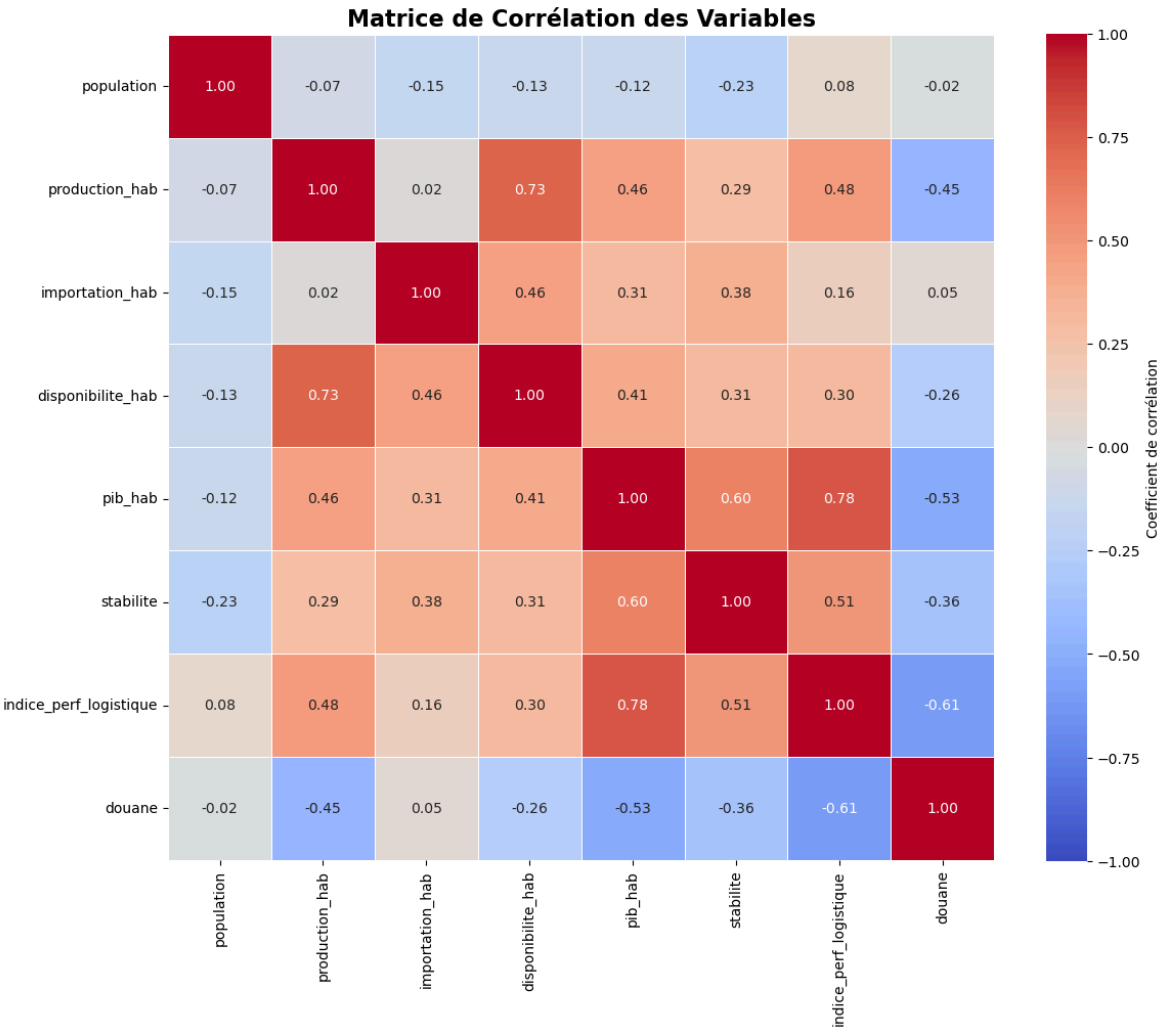
```
Matrice des composantes principales (Loadings) :
Dimensions : 4 composantes x 8 variables
```

	population	production_hab	importation_hab	disponibilite_hab	pib_hab	stabilite	in
F1	-0.081	0.384	0.209	0.366	0.459	0.376	
F2	-0.501	-0.119	0.609	0.272	-0.053	0.196	
F3	0.078	0.586	-0.134	0.580	-0.261	-0.408	
F4	0.808	-0.146	0.504	0.144	0.016	-0.049	

In [22]:
```
# Heatmap des composantes principales
plt.figure(figsize=(12, 8))

# Créer la heatmap avec les composantes
sns.heatmap(pcs,
            annot=True,
            cmap="coolwarm",
            center=0,
            vmin=-1,
            vmax=1,
            fmt=".2f",
            linewidths=0.5,
            cbar_kws={'label': 'Coefficient de contribution'})

plt.title("Heatmap des Composantes Principales - ACP", fontsize=16, fontweight='
plt.xlabel("Variables")
plt.ylabel("Composantes principales")
plt.tight_layout()
plt.show()
```



In [23]:
```
# Graphique de corrélation (matrice de corrélation)
plt.figure(figsize=(12, 10))

# Calculer la matrice de corrélation
correlation_matrix = X_scaled_no_pays.corr()

# Heatmap de la matrice de corrélation
sns.heatmap(correlation_matrix,
            annot=True,
            cmap="coolwarm",
            center=0,
            vmin=-1,
            vmax=1,
            fmt=".2f",
            linewidths=0.5,
            square=True,
            cbar_kws={'label': 'Coefficient de corrélation'})

plt.title("Matrice de Corrélation des Variables", fontsize=16, fontweight='bold'
plt.tight_layout()
plt.show()
```



2.2 Méthode du coude (K-means)

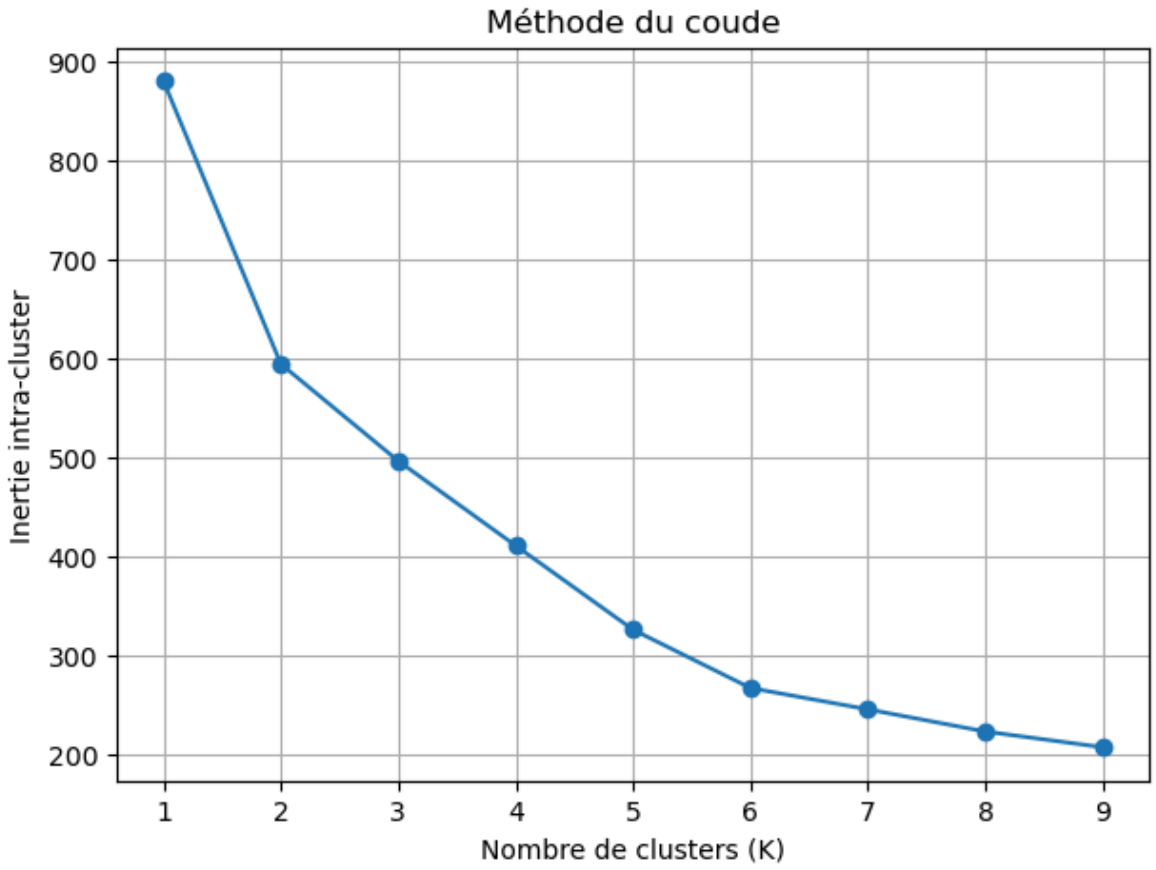
In [ ]:

KMEANS

In [26]:
```
# Création des cluster pour K-Means
X_cluster = X_pca
# Liste vide pour enregistrer les inerties
inerties = []

# Plage de valeurs pour K
K_range = range(1, 10)

# Boucle sur les différentes valeurs de K
for k in K_range:
    # Instancier K-Means avec un nombre de clusters donné
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)

    # Entraîner le modèle sur les données sans la colonne "pays"
    kmeans.fit(X_scaled_no_pays)

    # Enregistrer l'inertie
    inerties.append(kmeans.inertia_)
```

```python
# Tracé de la méthode du coude
plt.figure(figsize=(7, 5))
plt.plot(K_range, inerties, marker='o')
plt.title("Méthode du coude")
plt.xlabel("Nombre de clusters (K)")
plt.ylabel("Inertie intra-cluster")
plt.grid(True)
plt.show()
```

```
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
```



On va retenir 5 clusters pour la suite de notre analyse

In [28]:
```
# Choix du nombre de clusters
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(X_cluster)
```

```
C:\Users\PC\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment var
iable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [29]:
```
#Ajout d'une colonne cluster pour chaque pays
X_scaled_no_pays["cluster"] = clusters
X_scaled_no_pays
```

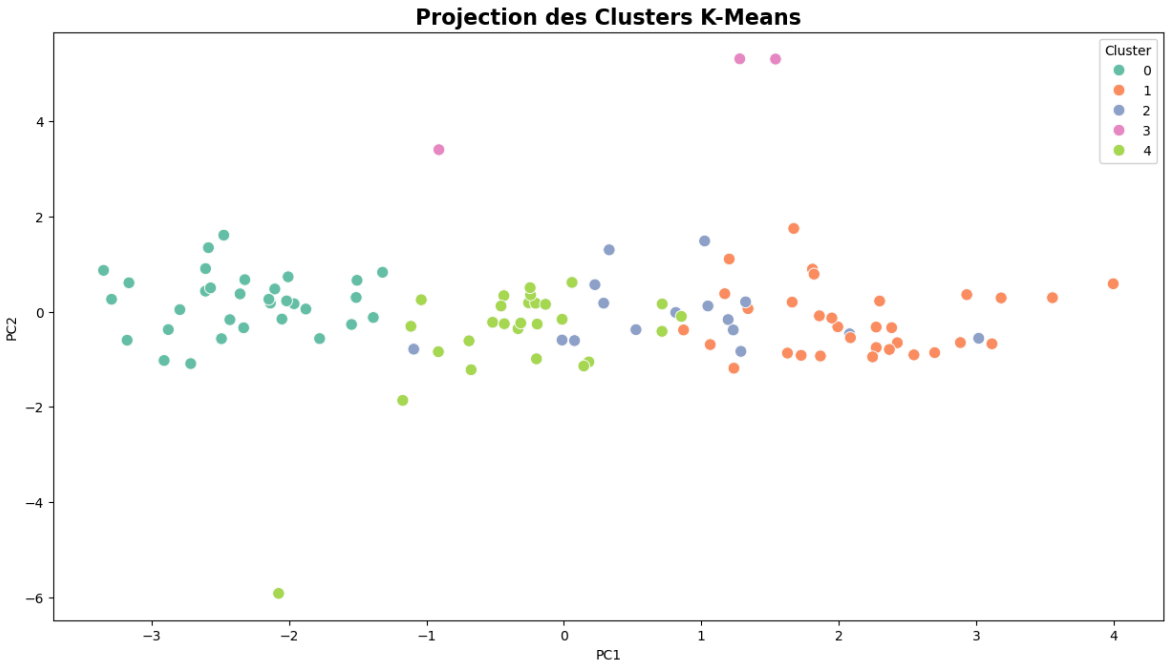
Out[29]:

	population	production_hab	importation_hab	disponibilite_hab	pib_hab	stabilite
0	0.12	0.59	0.12	0.96	-0.54	-1.07
1	-0.29	-0.80	0.51	-0.27	-0.36	0.05
2	0.01	-0.68	-0.69	-0.87	-0.50	-1.07
3	0.31	-0.02	0.23	0.00	1.46	1.18
4	-0.08	-0.98	0.13	-0.64	-0.80	-1.07
...	...	...	...	...	...	...
105	-0.28	-0.51	-0.61	-0.68	0.15	1.18
106	0.40	-0.51	-0.41	-0.50	-0.57	0.05
107	-0.18	-0.89	-0.63	-1.05	-0.96	0.05
108	-0.18	0.08	-0.69	-0.05	-0.53	0.05
109	-0.30	-1.05	0.15	-0.98	-1.00	0.05

110 rows × 9 columns

2.2 Projection des clusters

In [31]:
```
# Sélectionner uniquement les colonnes numériques
X_numeric = X_scaled_no_pays.drop(columns=["cluster", "cluster_cah"], errors='ig

# Appliquer PCA pour réduire les dimensions à 2
pca = PCA(n_components=2, random_state=42)
principal_components = pca.fit_transform(X_numeric)

# Créer un DataFrame avec les 2 premières composantes et les clusters K-Means
pca_df = pd.DataFrame(data=principal_components, columns=["PC1", "PC2"])
pca_df ["cluster"] = X_scaled_no_pays["cluster"]


# Afficher le graphique
plt.figure(figsize=(15, 8))
sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue="cluster", palette="Set2", s=
plt.title("Projection des Clusters K-Means", fontsize=16, fontweight='bold')
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="Cluster")
plt.show()
```



In [32]:
```
from sklearn.decomposition import PCA

#Variance expliquée par chaque composante
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_numeric)

print("Variance expliquée par chaque composante :", pca.explained_variance_ratio
print("Variance totale expliquée :", sum(pca.explained_variance_ratio_))
```

```
Variance expliquée par chaque composante : [0.43891712 0.17211053]
Variance totale expliquée : 0.6110276512610198
```

60% de l'information est conservée avec 2 composantes

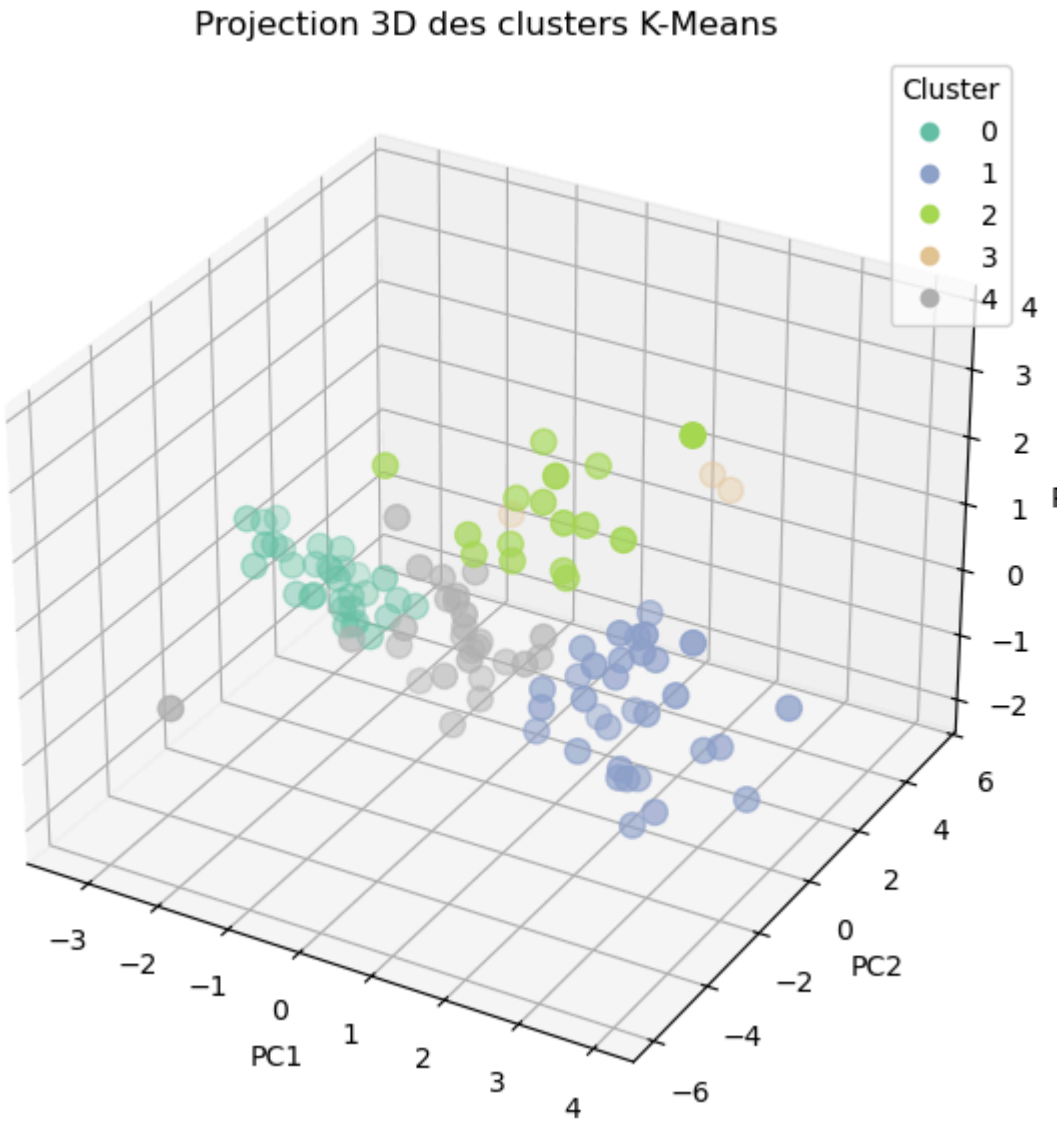
In [34]:
```
# Variance expliquée par chaque composante
pca = PCA(n_components=3)
X_pca_3d = pca.fit_transform(X_numeric)
print("Variance expliquée :", pca.explained_variance_ratio_)
print("Variance totale :", sum(pca.explained_variance_ratio_))
```

```
Variance expliquée : [0.43891712 0.17211053 0.13202334]
Variance totale : 0.743050993788561
```

74% de l'information est conservée avec 3 composantes

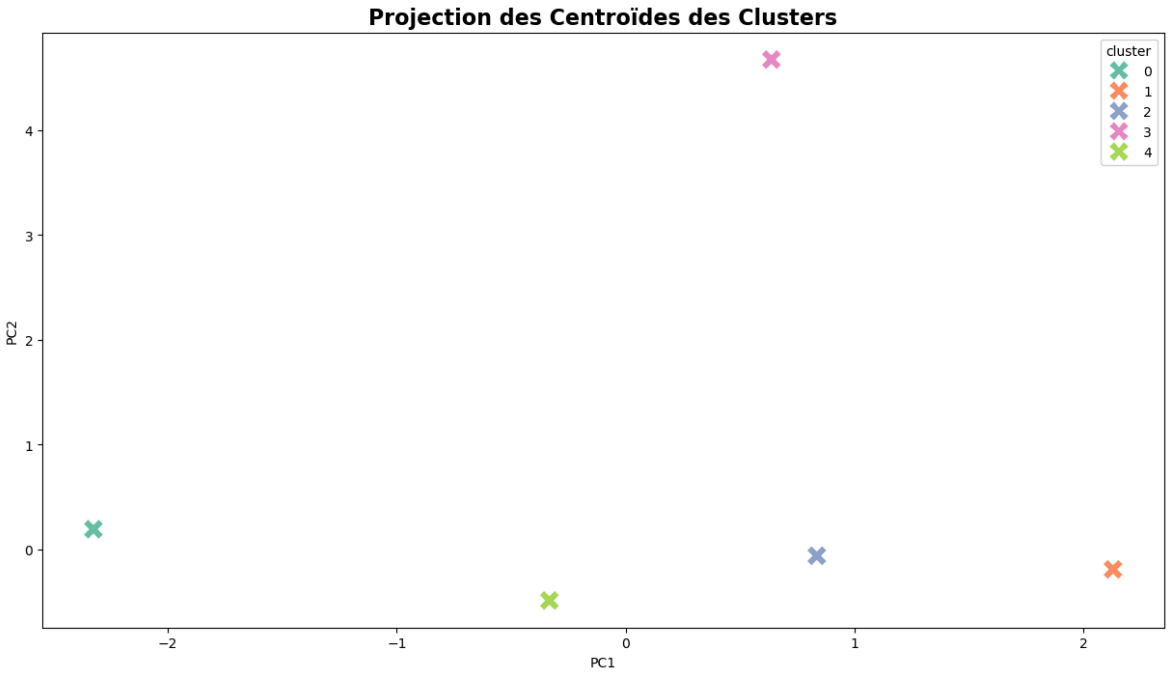
In [36]:
```
#Projection en 3D des clusters de Kmeans
from mpl_toolkits.mplot3d import Axes3D

pca_3d = PCA(n_components=3)
X_pca_3d = pca_3d.fit_transform(X_numeric)
df_3d = pd.DataFrame(X_pca_3d, columns=["PC1", "PC2", "PC3"])
df_3d["cluster"] = X_scaled_no_pays["cluster"]

fig = plt.figure(figsize=(10,7))
```

```
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(df_3d["PC1"], df_3d["PC2"], df_3d["PC3"], c=df_3d["cluster"
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
plt.title("Projection 3D des clusters K-Means")
plt.legend(*scatter.legend_elements(), title="Cluster")
plt.show()
```



2.3 Analyse des clusters

In [38]:
```
# Calculer les centroïdes
centroids = pca_df.groupby("cluster")[["PC1", "PC2"]].mean()

# Tracer un scatter plot des centroïdes
plt.figure(figsize=(15, 8))
sns.scatterplot(data=centroids, x="PC1", y="PC2", hue=centroids.index, palette="
plt.title("Projection des Centroïdes des Clusters", fontsize=16, fontweight='bol
plt.show()
```



In [39]:
```
# Ajouter la colonne cluster de X_scaled_no_pays vers X_scaled
X_scaled["cluster_kmeans"] = X_scaled_no_pays["cluster"]
```

In [40]:
```
# Affichage du nombre de pays par clusters
print("Nombre de pays par cluster (K-Means) :")
cluster_counts = X_scaled["cluster_kmeans"].value_counts().sort_index()
for cluster, count in cluster_counts.items():
    print(f"Cluster {cluster} : {count} pays")
```

```
Nombre de pays par cluster (K-Means) :
Cluster 0 : 32 pays
Cluster 1 : 32 pays
Cluster 2 : 16 pays
Cluster 3 : 3 pays
Cluster 4 : 27 pays
```

In [41]:
```
# Affichage des pays par cluster
print("\nPays dans chaque cluster :")
for cluster in sorted(X_scaled["cluster_kmeans"].unique()):
    pays_du_cluster = X_scaled[X_scaled["cluster_kmeans"] == cluster]["pays"].to
    print(f"\nCluster {cluster} : {pays_du_cluster}")
```

```
Pays dans chaque cluster :

Cluster 0 : ['Algérie', 'Angola', 'Azerbaïdjan', 'Bangladesh', 'Burkina Faso', 'B
énin', 'Cambodge', 'Cameroun', "Côte d'Ivoire", 'Gambie', 'Ghana', 'Guinée', 'Gui
née-Bissau', 'Kenya', 'Lesotho', 'Libéria', 'Madagascar', 'Mali', 'Mauritanie',
'Niger', 'Nigéria', 'Népal', 'Ouganda', 'Pakistan', 'Rwanda', 'République centraf
ricaine', 'Sierra Leone', 'Sénégal', 'Tchad', 'Togo', 'Zambie', 'Îles Salomon']

Cluster 1 : ['Allemagne', 'Arabie saoudite', 'Australie', 'Autriche', 'Belgique',
'Canada', 'Chypre', 'Croatie', 'Danemark', 'Espagne', 'Estonie', 'Finlande', 'Fra
nce', 'Grèce', 'Hongrie', 'Irlande', 'Islande', 'Italie', 'Japon', 'Koweït', 'Let
tonie', 'Lituanie', 'Malte', 'Norvège', 'Nouvelle-Zélande', 'Oman', 'Pays-Bas',
'Pologne', 'Portugal', 'Slovénie', 'Suisse', 'Suède']

Cluster 2 : ['Afrique du Sud', 'Argentine', 'Brésil', 'Chili', 'Colombie', 'Guyan
a', 'Israël', 'Jamaïque', 'Malaisie', 'Maurice', 'Mexique', 'Myanmar', 'Panama',
'Pérou', 'République dominicaine', 'Trinité-et-Tobago']

Cluster 3 : ['Bahamas', 'Gabon', 'Grenade']

Cluster 4 : ['Albanie', 'Bosnie-Herzégovine', 'Botswana', 'Bulgarie', 'Costa Ric
a', 'El Salvador', 'Guatemala', 'Géorgie', 'Honduras', 'Inde', 'Indonésie', 'Jord
anie', 'Kazakhstan', 'Macédoine du Nord', 'Mongolie', 'Monténégro', 'Namibie', 'N
icaragua', 'Paraguay', 'Philippines', 'Roumanie', 'Thaïlande', 'Turquie', 'Ukrain
e', 'Uruguay', 'Viet Nam', 'Équateur']
```

Cluster 3 contient seulement 3 pays : Bahamas, Gabon, Grenade:

Ce qui indique qu'ils ont des caractéristiques particulières par rapport aux autres pays.
Ces pays sont regroupés parce qu'ils sont hors norme par rapport aux clusters principaux ce sont des petits pays ou des pays avec des indicateurs extrêmes (outliers)

In [43]:
```
#Affichage de la moyenne par cluster kmean
moyennes_par_cluster_k = X_scaled.groupby('cluster_kmeans').mean(numeric_only=Tr
moyennes_par_cluster_k
```

Out[43]:

cluster_kmeans	population	production_hab	importation_hab	disponibilite_hab	pib_h
0	-0.056875	-0.915000	-0.469375	-0.994062	-0.9075
1	-0.156250	0.517188	0.410937	0.329375	1.2650
2	-0.005625	1.403750	-0.303750	1.405625	-0.1056
3	-0.303333	-0.553333	4.196667	2.303333	-0.1633
4	0.289259	-0.298889	-0.216667	-0.301111	-0.3444

Pays du cluster 1 triés par distance au centroïde

Afin d'identifier les pays qui se démarquent au sein du cluster 1, nous avons construit un graphique classant les pays du plus proche au plus éloigné du centroïde. Cette

représentation permet ainsi de visualiser rapidement quels pays sont les plus typiques ou représentatifs du cluster

In [46]:
```python
# Variables utilisées pour le clustering (quantitatives)
variables = [
    "population", "production_hab", "importation_hab",
    "disponibilite_hab", "pib_hab", "stabilite",
    "indice_perf_logistique", "douane"
]

# Choix du cluster à analyser
cluster_num = 1

# Filtrer les pays du cluster
cluster_data = X_scaled[X_scaled["cluster_kmeans"] == cluster_num].copy()

# Calcul du centroïde (moyenne de chaque variable)
centroid = cluster_data[variables].mean()
```

In [47]:
```python
# Fonction distance euclidienne
def distance(row):
    return np.linalg.norm(row[variables] - centroid)

cluster_data["distance_au_centroid"] = cluster_data.apply(distance, axis=1)
```

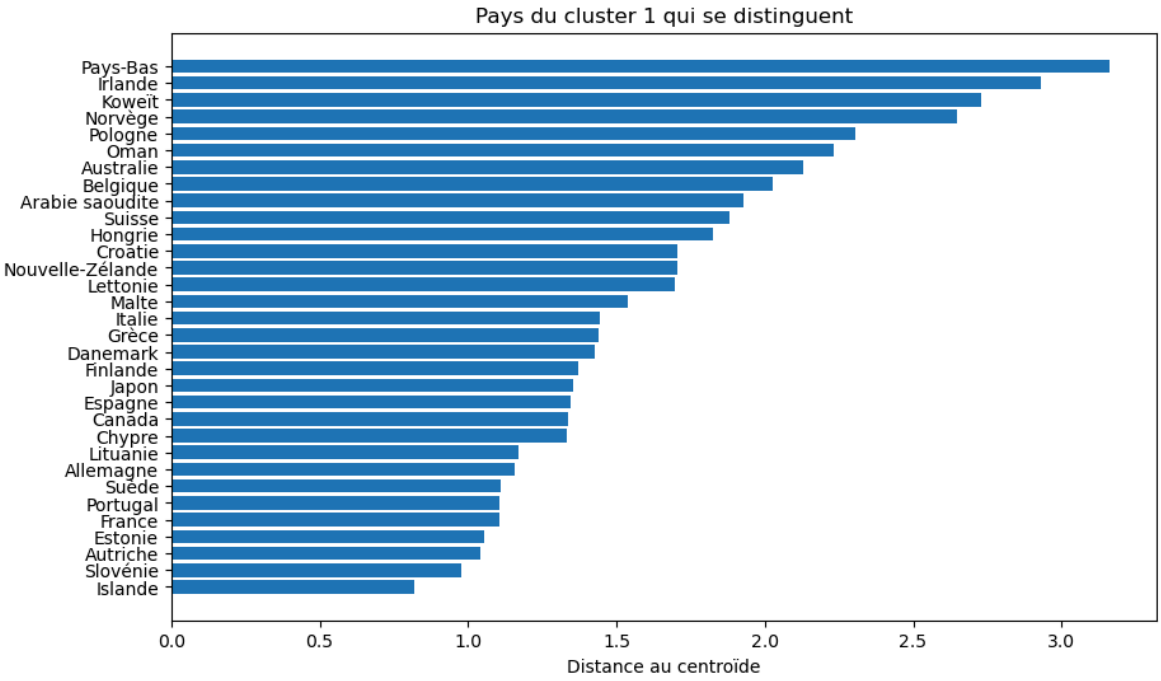
In [48]:
```python
#Affichage de la liste des ^pays du cluster 1 triés par distance au centroïde
cluster_data = cluster_data.sort_values("distance_au_centroid", ascending=False)

print(f"\nPays du cluster {cluster_num} triés par distance au centroïde :\n")
print(cluster_data[["pays", "distance_au_centroid"]])
```

Pays du cluster 1 triés par distance au centroïde :

	pays	distance_au_centroid
85	Pays-Bas	3.161405
48	Irlande	2.931680
57	Koweït	2.729631
77	Norvège	2.647368
87	Pologne	2.305264
80	Oman	2.230249
7	Australie	2.131079
12	Belgique	2.026515
5	Arabie saoudite	1.928903
96	Suisse	1.880347
45	Hongrie	1.823741
26	Croatie	1.707258
78	Nouvelle-Zélande	1.706046
59	Lettonie	1.696419
66	Malte	1.539990
51	Italie	1.442322
38	Grèce	1.438969
28	Danemark	1.427471
32	Finlande	1.372140
53	Japon	1.354181
30	Espagne	1.347040
21	Canada	1.335978
23	Chypre	1.330890
61	Lituanie	1.169909
3	Allemagne	1.158916
97	Suède	1.110304
88	Portugal	1.105053
33	France	1.104329
31	Estonie	1.054567
8	Autriche	1.041642
95	Slovénie	0.977669
49	Islande	0.816891

Pays du cluster 1 : du plus proche au plus eloigné par rapport au centroïde

In [50]:
```
#Affichage du graphique

plt.figure(figsize=(10,6))
plt.barh(cluster_data["pays"], cluster_data["distance_au_centroid"])
plt.xlabel("Distance au centroïde")
plt.title(f"Pays du cluster {cluster_num} qui se distinguent")
plt.gca().invert_yaxis()
plt.show()
```

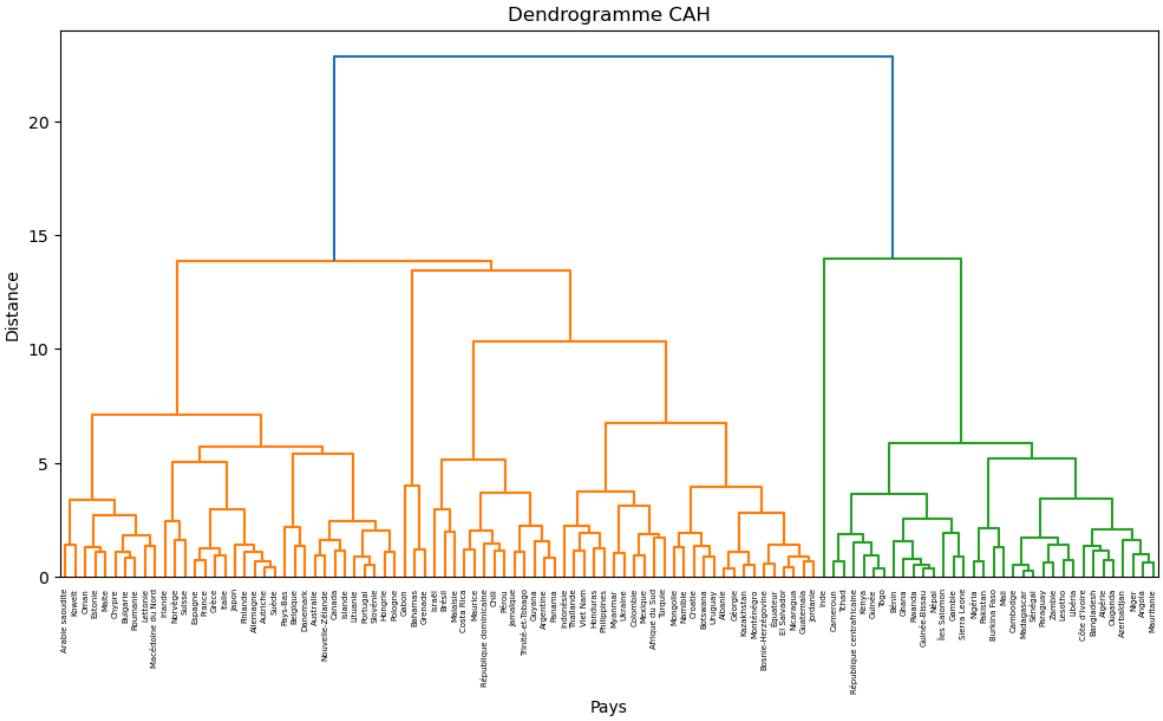


D'après ce graphique, l'Islande se démarque comme le pays le plus proche du centroïde, suivie par la Slovénie, l'Autriche et l'Estonie

Etape 3 - Classification Ascendante Hiérarchique (CAH)

3.1 Dendrogramme

In [54]:
```
# Données numériques seulement pour CAH
X = X_scaled_no_pays.drop(columns=["cluster"], errors='ignore')

# Calcul des liaisons hiérarchiques (méthode de Ward)
Z = linkage(X, method='ward')

# # Tracer le dendrogramme
plt.figure(figsize=(12, 6))
dendrogram(Z, labels=df["pays"].values, leaf_rotation=90)
plt.title("Dendrogramme CAH")
plt.xlabel("Pays")
plt.ylabel("Distance")
plt.show()

# Découper le dendrogramme pour obtenir 5 clusters
clusters_cah = fcluster(Z, t=5, criterion='maxclust')

# Ajouter les clusters à X_scaled
X_scaled["cluster_cah"] = clusters_cah
```



In [55]:
```
#Affichage de la moyenne par cluster CAH
moyennes_par_cluster_c = X_scaled.groupby('cluster_cah').mean(numeric_only=True)
moyennes_par_cluster_c
```

Out[55]:

cluster_cah	population	production_hab	importation_hab	disponibilite_hab	pib_hab
1	-0.159706	0.460882	0.443235	0.326176	1.192941
2	-0.303333	-0.553333	4.196667	2.303333	-0.163333
3	-0.034615	0.432308	-0.288718	0.405641	-0.252308
4	-0.063030	-0.908182	-0.475758	-0.991515	-0.894545
5	9.760000	-0.900000	-0.690000	-1.100000	-0.750000

3.2 CAH

In [57]:
```
# Création d'un DataFrame temporaire avec les pays et les clusters CAH
df_cah = df.copy()
df_cah["cluster_cah"] = X_scaled["cluster_cah"].values

# Liste des pays par cluster CAH
print("\nPays dans chaque cluster (CAH) :")
pays_par_cluster_cah = df_cah.groupby("cluster_cah")["pays"].apply(list)
for cluster_id, pays in pays_par_cluster_cah.items():
    print(f"\nCluster {cluster_id} ({len(pays)} pays) : {pays}")
```

```
Pays dans chaque cluster (CAH) :

Cluster 1 (34 pays) : ['Allemagne', 'Arabie saoudite', 'Australie', 'Autriche',
'Belgique', 'Bulgarie', 'Canada', 'Chypre', 'Danemark', 'Espagne', 'Estonie', 'Fi
nlande', 'France', 'Grèce', 'Hongrie', 'Irlande', 'Islande', 'Italie', 'Japon',
'Koweït', 'Lettonie', 'Lituanie', 'Macédoine du Nord', 'Malte', 'Norvège', 'Nouve
lle-Zélande', 'Oman', 'Pays-Bas', 'Pologne', 'Portugal', 'Roumanie', 'Slovénie',
'Suisse', 'Suède']

Cluster 2 (3 pays) : ['Bahamas', 'Gabon', 'Grenade']

Cluster 3 (39 pays) : ['Afrique du Sud', 'Albanie', 'Argentine', 'Bosnie-Herzégov
ine', 'Botswana', 'Brésil', 'Chili', 'Colombie', 'Costa Rica', 'Croatie', 'El Sal
vador', 'Guatemala', 'Guyana', 'Géorgie', 'Honduras', 'Indonésie', 'Israël', 'Jam
aïque', 'Jordanie', 'Kazakhstan', 'Malaisie', 'Maurice', 'Mexique', 'Mongolie',
'Monténégro', 'Myanmar', 'Namibie', 'Nicaragua', 'Panama', 'Philippines', 'Péro
u', 'République dominicaine', 'Thaïlande', 'Trinité-et-Tobago', 'Turquie', 'Ukrai
ne', 'Uruguay', 'Viet Nam', 'Équateur']

Cluster 4 (33 pays) : ['Algérie', 'Angola', 'Azerbaïdjan', 'Bangladesh', 'Burkina
Faso', 'Bénin', 'Cambodge', 'Cameroun', "Côte d'Ivoire", 'Gambie', 'Ghana', 'Guin
ée', 'Guinée-Bissau', 'Kenya', 'Lesotho', 'Libéria', 'Madagascar', 'Mali', 'Mauri
tanie', 'Niger', 'Nigéria', 'Népal', 'Ouganda', 'Pakistan', 'Paraguay', 'Rwanda',
'République centrafricaine', 'Sierra Leone', 'Sénégal', 'Tchad', 'Togo', 'Zambi
e', 'Îles Salomon']

Cluster 5 (1 pays) : ['Inde']
```

3.3 Analyse des clusters

In [59]:
```python
# Compter le nombre de pays par cluster CAH
nombre_par_cluster_cah = X_scaled["cluster_cah"].value_counts().sort_index()
print("Nombre de pays par cluster (CAH) :")
for cluster, count in nombre_par_cluster_cah.items():
    print(f"Cluster {cluster} : {count} pays")
```

```
Nombre de pays par cluster (CAH) :
Cluster 1 : 34 pays
Cluster 2 : 3 pays
Cluster 3 : 39 pays
Cluster 4 : 33 pays
Cluster 5 : 1 pays
```

3.4 Comparaison avec K-means

La différence entre la prévision des clusters par K-Means et par CAH s'explique principalement par le principe de construction des méthodes : K-Means minimise la variance autour de centres fixes, favorisant des clusters sphériques et équilibrés, alors que la CAH fusionne les points progressivement selon la distance, permettant des clusters de tailles et formes inégales. Les outliers comme l'Inde ou les petits pays insulaires sont traités différemment selon la méthode, ce qui explique les divergences dans l'affectation des pays aux clusters.

Dans le cadre de notre analyse on va préférer utiliser les cluster du Kmeans pour avoir des groupes plus homogène et représentatif.

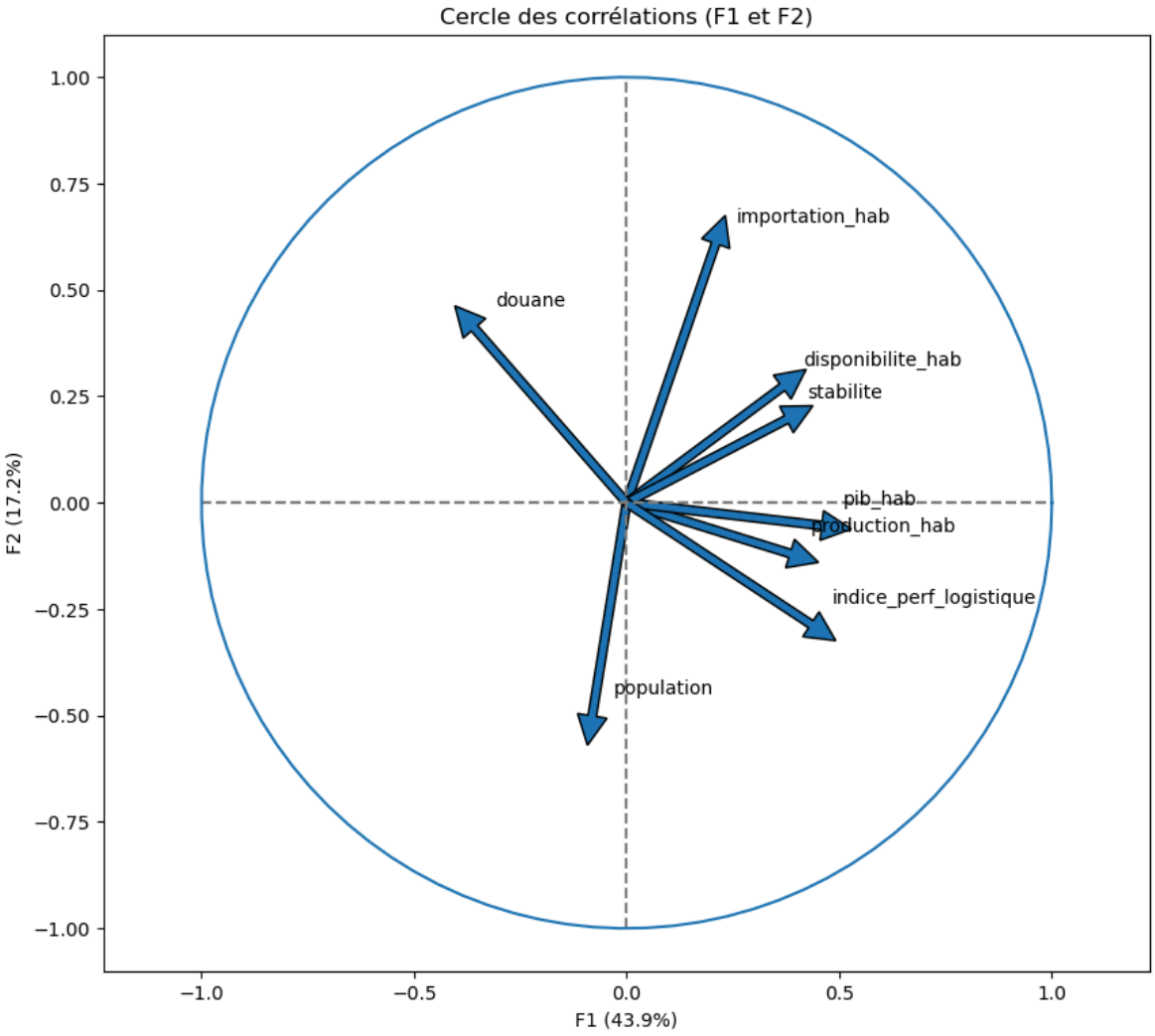
Etape 4 - Analyses

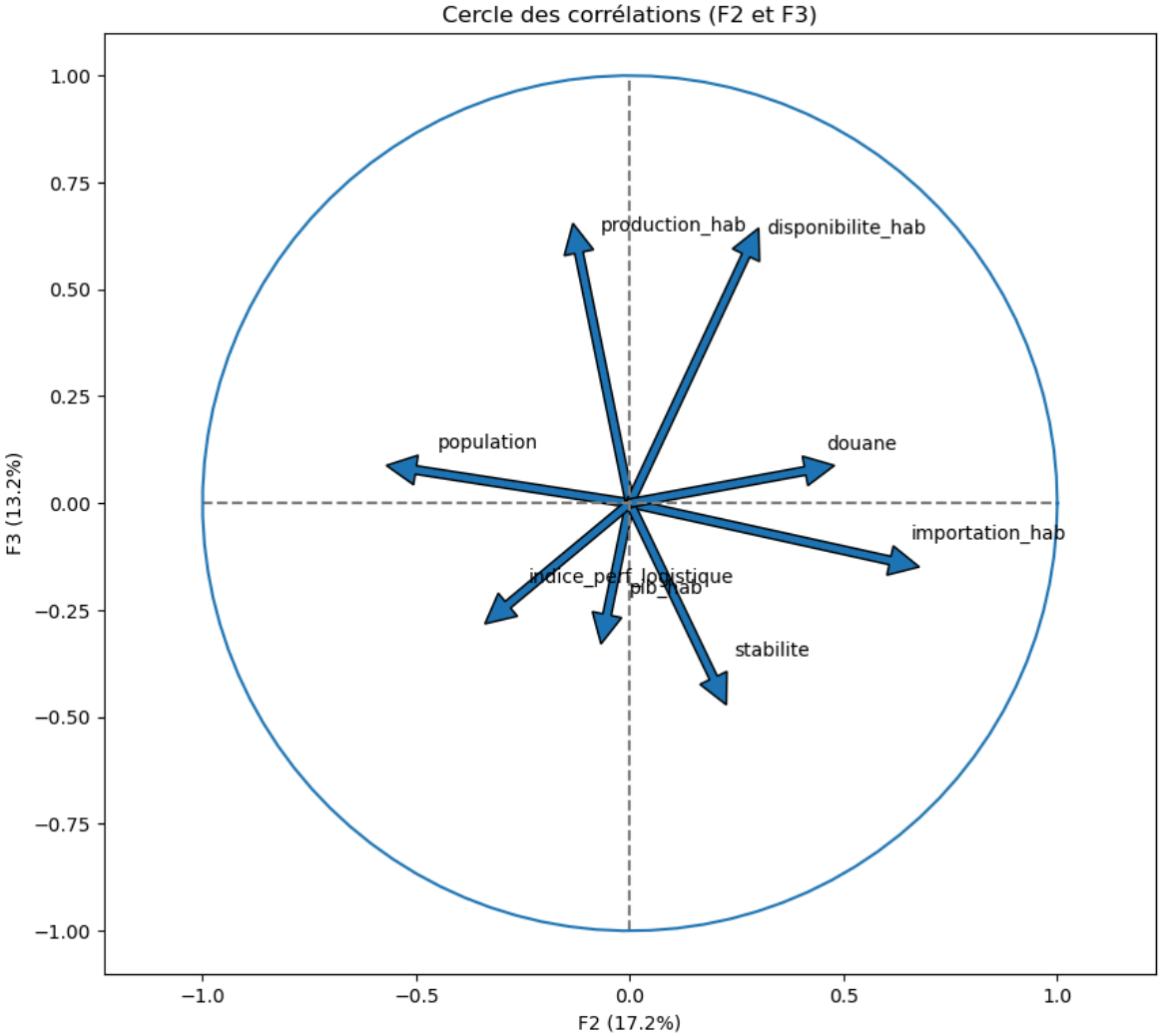
4.1 Cercles des corrélations

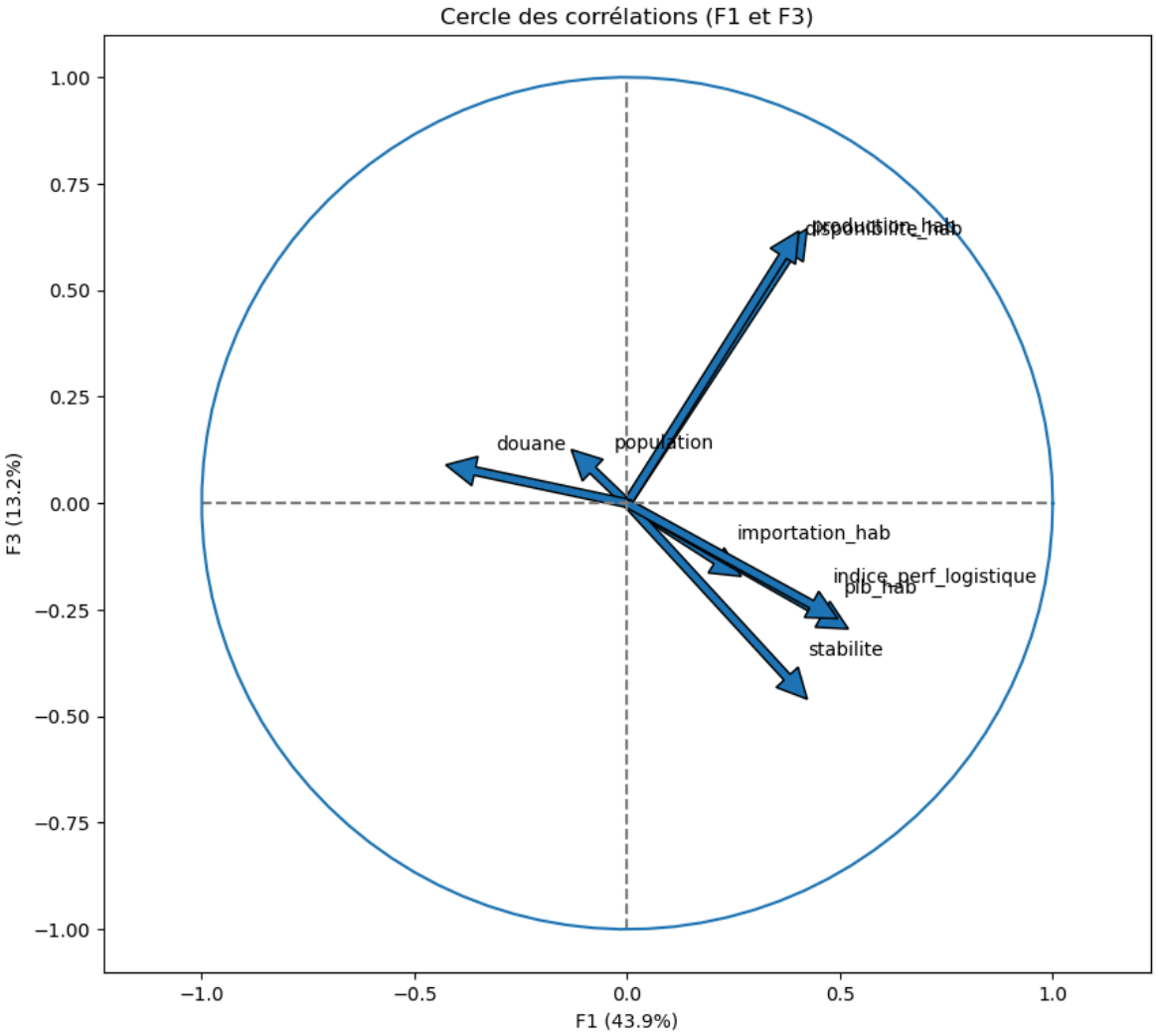
In [64]:
```
# Affichage du cercle des corrélations
def correlation_graph(pca, x_y, features):
    # Extrait x et y
    x,y = x_y

    # Taille de l'image (en inches)
    fig, ax = plt.subplots(figsize=(10, 9))

    # Pour chaque composante :
    for i in range(0, pca.components_.shape[1]):
        # Les flèches
        ax.arrow(0,0,
                pca.components_[x, i],
                pca.components_[y, i],
                head_width=0.07,
                head_length=0.07,
                width=0.02, )

        # Les labels
        plt.text(pca.components_[x, i] + 0.05,
                pca.components_[y, i] + 0.05,
                features[i])

    # Affichage des lignes horizontales et verticales
    plt.plot([-1, 1], [0, 0], color='grey', ls='--')
    plt.plot([0, 0], [-1, 1], color='grey', ls='--')

    # Nom des axes, avec le pourcentage d'inertie expliqué
    plt.xlabel('F{} ({}%)'.format(x+1, round(100*pca.explained_variance_ratio_[x
    plt.ylabel('F{} ({}%)'.format(y+1, round(100*pca.explained_variance_ratio_[y

    # J'ai copié collé le code sans le lire
    plt.title("Cercle des corrélations (F{} et F{})".format(x+1, y+1))

    # Le cercle
    an = np.linspace(0, 2 * np.pi, 100)
    plt.plot(np.cos(an), np.sin(an))

    # Axes et display
    plt.axis('equal')
    plt.show()

# Utilisation de la fonction - Affichage de tous les cercles possibles
print("Cercles des corrélations :")
```

```python
# Premier cercle : F1 vs F2
correlation_graph(pca, [0,1], X_scaled_no_pays.columns)

# Deuxième cercle : F2 vs F3
correlation_graph(pca, [1,2], X_scaled_no_pays.columns)

# Troisième cercle : F1 vs F3
correlation_graph(pca, [0,2], X_scaled_no_pays.columns)

# Cercles supplémentaires si il y a plus de composantes
if pca.n_components_ > 3:
    correlation_graph(pca, [2,3], X_scaled_no_pays.columns)  # F3 vs F4
    correlation_graph(pca, [0,3], X_scaled_no_pays.columns)  # F1 vs F4
    correlation_graph(pca, [1,3], X_scaled_no_pays.columns)  # F2 vs F4
```

Cercles des corrélations :




Cercle des corrélations (F2 et F3)
production_hab
disponibilite_hab
population
douane
importation_hab
indice_perf_logistique
pib_hab
stabilite
F3 (13.2%)
F2 (17.2%)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0

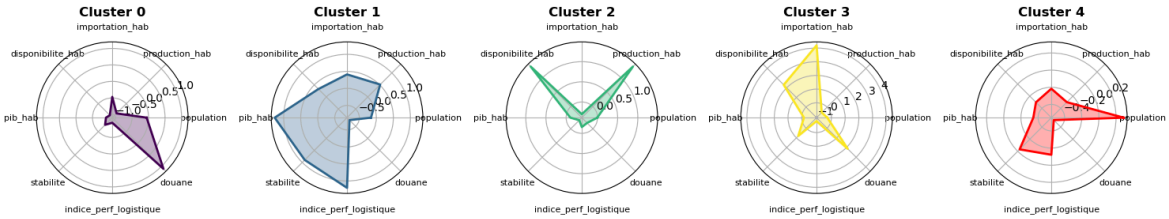


4.2 Projection des individus

In [66]:
```
# Affichage du radar chart par cluster

# Sélectionner uniquement les colonnes numériques pour le groupby (exclure les c
colonnes_numeriques = X_scaled.select_dtypes(include=[np.number]).columns
colonnes_numeriques = [col for col in colonnes_numeriques if col not in ["cluste

# Calculer la moyenne de chaque variable par cluster
cluster_means = X_scaled.groupby("cluster_kmeans")[colonnes_numeriques].mean()

# Utiliser les moyennes
cluster_means_scaled = cluster_means

# Définir les variables et les clusters
categories = cluster_means.columns
num_vars = len(categories)
clusters = cluster_means.index

# Palette de couleurs "Set2"
set2_colors = ["#440154", "#31688e", "#35b779", "#fde725", "#ff0000"]

# Création du graphique radar pour chaque cluster
fig, axes = plt.subplots(1, len(clusters), figsize=(15, 6), subplot_kw=dict(pola
fig.suptitle("Profil des clusters", fontsize=16, fontweight='bold')
```

```
for i, cluster in enumerate(clusters):
    values = cluster_means_scaled.loc[cluster].values
    values = np.concatenate((values, [values[0]]))
    angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()
    angles += angles[:1]
    ax = axes[i]
    color = set2_colors[i % len(set2_colors)]
    ax.fill(angles, values, alpha=0.3, label=f'Cluster {cluster}', color=color)
    ax.plot(angles, values, linewidth=2, color=color)
    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(categories, fontsize=8)
    ax.set_title(f'Cluster {cluster}', fontsize=12, fontweight='bold')
    ax.grid(True)

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

Profil des clusters



Profil général des clusters : Tous les clusters sont structurés autour des mêmes variables clés: Importation/hab, PIB/hab, disponibilité, stabilité et performance logistique.

Spécificité du cluster 1 : Ce cluster présente le profil le plus équilibré et complet, ce qui en fait le plus représentatif. On va vérifier cela avec un affichage du profil détaillé des clusters pour pouvoir mieux les comparer.

4.3 Choix du cluster

In [69]:
```
#Affichage du meilleur cluster

print("="*60)
print("PROFIL DÉTAILLÉ DES CLUSTERS K-MEANS")
print("="*60)

def get_level(value):
    """Détermine le niveau (MIN, Faible, Moyen, Élevé, MAX)"""
    if value < -1:
        return "MIN"
    elif value < -0.5:
        return "Faible"
    elif value < 0.5:
```

```
        return "Moyen"
    elif value < 1:
        return "Élevé"
    else:
        return "MAX"

def profile_clusters(data, cluster_col='cluster_kmeans', key_vars=['population',

    # Colonnes numériques à analyser
    numeric_cols = [col for col in data.select_dtypes(include=[np.number]).colum
                    if col != cluster_col]

    # Statistiques moyennes par cluster
    cluster_stats = data.groupby(cluster_col)[numeric_cols].mean()

    for cluster_id in sorted(cluster_stats.index):
        print(f"## Cluster {cluster_id}")
        cluster_data = cluster_stats.loc[cluster_id]

        # Afficher les variables clés
        for var in key_vars:
            if var in cluster_data:
                print(f"- {var.replace('_', ' ').title()} : {get_level(cluster_d

        # Variables avec valeurs extrêmes
        max_vars = cluster_data.drop(key_vars, errors='ignore').nlargest(top_n).
        min_vars = cluster_data.drop(key_vars, errors='ignore').nsmallest(top_n)

        for var in max_vars:
            print(f"- {var.replace('_', ' ').title()} : MAX")

        for var in min_vars:
            print(f"- {var.replace('_', ' ').title()} : MIN")

        # Afficher les pays du cluster
        pays_cluster = data[data[cluster_col] == cluster_id]['pays'].tolist()
        print(f"- Pays : {', '.join(pays_cluster)}")

# Exemple d'utilisation
profile_clusters(X_scaled)
```

==
PROFIL DÉTAILLÉ DES CLUSTERS K-MEANS
==
Cluster 0
- Population : Moyen
- Pib Hab : Faible
- Cluster Cah : MAX
- Douane : MAX
- Importation Hab : MAX
- Disponibilite Hab : MIN
- Indice Perf Logistique : MIN
- Production Hab : MIN
- Pays : Algérie, Angola, Azerbaïdjan, Bangladesh, Burkina Faso, Bénin, Cambodge, Cameroun, Côte d'Ivoire, Gambie, Ghana, Guinée, Guinée-Bissau, Kenya, Lesotho, Libéria, Madagascar, Mali, Mauritanie, Niger, Nigéria, Népal, Ouganda, Pakistan, Rwanda, République centrafricaine, Sierra Leone, Sénégal, Tchad, Togo, Zambie, Îles Salomon
Cluster 1
- Population : Moyen
- Pib Hab : MAX
- Indice Perf Logistique : MAX
- Cluster Cah : MAX
- Stabilite : MAX
- Douane : MIN
- Disponibilite Hab : MIN
- Importation Hab : MIN
- Pays : Allemagne, Arabie saoudite, Australie, Autriche, Belgique, Canada, Chypre, Croatie, Danemark, Espagne, Estonie, Finlande, France, Grèce, Hongrie, Irlande, Islande, Italie, Japon, Koweït, Lettonie, Lituanie, Malte, Norvège, Nouvelle-Zélande, Oman, Pays-Bas, Pologne, Portugal, Slovénie, Suisse, Suède
Cluster 2
- Population : Moyen
- Pib Hab : Moyen
- Cluster Cah : MAX
- Disponibilite Hab : MAX
- Production Hab : MAX
- Importation Hab : MIN
- Stabilite : MIN
- Douane : MIN
- Pays : Afrique du Sud, Argentine, Brésil, Chili, Colombie, Guyana, Israël, Jamaïque, Malaisie, Maurice, Mexique, Myanmar, Panama, Pérou, République dominicaine, Trinité-et-Tobago
Cluster 3
- Population : Moyen
- Pib Hab : Moyen
- Importation Hab : MAX
- Disponibilite Hab : MAX
- Douane : MAX
- Indice Perf Logistique : MIN
- Production Hab : MIN
- Stabilite : MIN
- Pays : Bahamas, Gabon, Grenade
Cluster 4
- Population : Moyen
- Pib Hab : Moyen
- Cluster Cah : MAX
- Stabilite : MAX
- Indice Perf Logistique : MAX
- Douane : MIN
- Disponibilite Hab : MIN

```
- Production Hab : MIN
- Pays : Albanie, Bosnie-Herzégovine, Botswana, Bulgarie, Costa Rica, El Salvado
r, Guatemala, Géorgie, Honduras, Inde, Indonésie, Jordanie, Kazakhstan, Macédoine
du Nord, Mongolie, Monténégro, Namibie, Nicaragua, Paraguay, Philippines, Roumani
e, Thaïlande, Turquie, Ukraine, Uruguay, Viet Nam, Équateur
```

Le profil détaillés de chaque cluster nous permet de confirmer que le cluster 1 est le plus pertinent pour la poursuite de l'étude de marché export. Avec 4 variables en MAX contre 3 pour les autres clusters.

Cluster 1 :

Allemagne, Arabie saoudite, Australie, Autriche, Belgique, Canada, Chypre, Croatie, Danemark, Espagne, Estonie, Finlande, France, Grèce, Hongrie, Irlande, Islande, Italie, Japon, Koweït, Lettonie, Lituanie, Malte, Norvège, Nouvelle-Zélande, Oman, Pays-Bas, Pologne, Portugal, Slovénie, Suisse, Suède

4.4 Conclusion

L’analyse en composantes principales (ACP) nous a conduit à retenir quatre dimensions essentielles pour représenter les données. Grâce à la méthode du coude, nous avons déterminé un regroupement optimal en cinq clusters. La projection des centroïdes permet d’identifier le "profil moyen" de chaque groupe de pays, tout en révélant une bonne séparation entre les clusters, signe d’une segmentation pertinente.

Les centroïdes incarnent le type représentatif de chaque cluster. L’examen approfondi des groupes montre que le Cluster 1 constitue une cible stratégique prioritaire, caractérisée par un PIB élevé et une forte stabilité. À l’inverse, le Cluster 0 représente un marché de volume, avec une population importante mais un pouvoir d’achat limité.

En conclusion le cluster 1 semble être le meilleur choix