2_clustering_visualisations_072025

August 25, 2025

ETUDE DE MARCHE_ACP ET CLUSTERING

I_ANALYSE EN COMPOSANTES PRINCIPALES (ACP)

1_IMPORT DES LIBRAIRIES

[3]:
```
import pandas as pd
import numpy as np

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.preprocessing import MinMaxScaler


import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection

import seaborn as sns
```

[4]:
```
# on 'charge' seaborn
sns.set()
```

[5]:
```
#fichier final importé
df = pd.read_csv("top_100_pays_avec_distance_corrigee.csv", decimal=".",␣
 ↪index_col=0)
df_copy = df.copy()
df
```

[5]:

Zone	Disponibilité alimentaire en quantité (kg/personne/an)	\
Indonésie	7.19	
Pakistan	5.86	
Nigéria	1.01	
Bangladesh	1.50	
Fédération de Russie	30.98	
…	…	
Norvège	19.05	
Congo	21.53	

Costa Rica	26.52
Irlande	25.82
Libéria	10.67

Zone	Disponibilité de protéines en quantité (g/personne/jour)
Indonésie	2.42
Pakistan	1.97
Nigéria	0.31
Bangladesh	0.47
Fédération de Russie	10.44
…	…
Norvège	7.81
Congo	7.45
Costa Rica	7.93
Irlande	11.60
Libéria	3.74

Zone	Disponibilité intérieure	Exportations - Quantité
Indonésie	2323.0	0.0
Pakistan	1282.0	4.0
Nigéria	202.0	0.0
Bangladesh	250.0	-2.5
Fédération de Russie	4556.0	115.0
…	…	…
Norvège	102.0	0.0
Congo	110.0	0.0
Costa Rica	134.0	3.0
Irlande	128.0	93.0
Libéria	50.0	0.0

Zone	Importations - Quantité	PIB par habitant
Indonésie	1.0	11899.3
Pakistan	2.0	5191.9
Nigéria	0.0	5849.1
Bangladesh	0.0	6020.3
Fédération de Russie	226.0	36011.8
…	…	…
Norvège	2.0	85144.1
Congo	104.0	7426.4
Costa Rica	17.0	22525.4
Irlande	99.0	85225.2
Libéria	48.0	1677.5

Zone	Stabilité politique	Population	ratio_export_import	\
Indonésie	-0.50	264650963.0	0.000000e+00	
Pakistan	-2.40	207906209.0	1.999999e+00	
Nigéria	-2.00	190873244.0	0.000000e+00	
Bangladesh	-1.25	159685424.0	-2.500000e+06	
Fédération de Russie	-0.64	145530082.0	5.088496e-01	
…	…	…	…	
Norvège	1.15	5296326.0	0.000000e+00	
Congo	-0.53	5110695.0	0.000000e+00	
Costa Rica	0.60	4949954.0	1.764706e-01	
Irlande	0.99	4753279.0	9.393939e-01	
Libéria	-0.32	4702226.0	0.000000e+00	

Zone	conso_par_habitant	score_agro_eco	Distance_France_km
Indonésie	0.008778	4761.727	12061.04
Pakistan	0.006166	2077.798	6040.23
Nigéria	0.001058	2339.343	4383.89
Bangladesh	0.001566	2408.195	7854.91
Fédération de Russie	0.031306	14413.822	5480.82
…	…	…	…
Norvège	0.019259	34063.700	1427.78
Congo	0.021523	2976.860	6107.36
Costa Rica	0.027071	9018.296	8895.84
Irlande	0.026929	34098.123	852.03
Libéria	0.010633	674.105	4904.59

[100 rows x 12 columns]

[6]:
```
df.isna().sum()
```

[6]:
```
Disponibilité alimentaire en quantité (kg/personne/an)      0
Disponibilité de protéines en quantité (g/personne/jour)    0
Disponibilité intérieure                                    0
Exportations - Quantité                                     0
Importations - Quantité                                     0
PIB par habitant                                            0
Stabilité politique                                         0
Population                                                  0
ratio_export_import                                         0
conso_par_habitant                                          0
score_agro_eco                                              0
Distance_France_km                                          0
dtype: int64
```

[7]:
```
df = df.sort_index()
display(df.head())
```

Zone	Disponibilité alimentaire en quantité (kg/personne/an)	Disponibilité de protéines en quantité (g/personne/jour)	Disponibilité intérieure	Exportations - Quantité	Importations - Quantité	PIB par habitant	Stabilité politique	Population	ratio_export_import	conso_par_habitant	score_agro_eco	Distance_France_km
Afghanistan	1.53	0.54	57.0	-1.53	29.0	2956.8	-2.79	36296113.0	-0.052759	0.001570	1182.342	5432.92
Afrique du Sud	35.69	14.11	2118.0	63.00	514.0	14823.6	-0.28	57009756.0	0.122568	0.037152	5940.063	8892.99
Algérie	6.38	1.97	277.0	0.00	2.0	13805.4	-0.92	41389189.0	0.000000	0.006693	5523.798	2315.90
Allemagne	19.47	7.96	1739.0	646.00	842.0	61563.6	0.57	82658409.0	0.767221			
Angola	10.56	3.60	319.0	0.00	277.0	9050.0	-0.39	29816766.0	0.000000			

```
Allemagne                0.021038       24631.452             634.06
Angola                   0.010699        3623.051            6892.07
```

II *ANALYSE DES COMPOSANTES* ACP

DATA SPLIT

[10]:
```
X = df
names = df.index
features = df.columns
```

SCALING

[12]:
```
# On instancie
scaler = StandardScaler()

# On fit
scaler.fit(X)

# On transforme
X_scaled = scaler.transform(X)
X_scaled
```

[12]:
```
array([[-1.1527478 , -1.10678106, -0.65038922, …, -1.14493121,
        -0.88564632,  0.01568966],
       [ 1.34903842,  1.52767236,  1.79096906, …,  1.31552545,
        -0.34748185,  1.03668623],
       [-0.79754689, -0.82916364, -0.38978814, …, -0.79073114,
        -0.39456722, -0.90407981],
       …,
       [ 1.91882462,  1.66162762, -0.22987384, …,  1.74947214,
         2.15149395, -0.03660733],
       [ 0.14941142, -0.01766656, -0.31397692, …,  0.15129068,
        -0.3603015 ,  1.22012256],
       [-1.25528002, -1.20385008, -0.70132488, …, -1.24442769,
        -0.91749999,  0.02026044]])
```

[13]:
```
# On espère avoir une moyenne à 0 et un écart type à 1 :
idx = ["mean", "std"]
pd.DataFrame(X_scaled).describe().round(2).loc[idx, :]
```

[13]:

	0	1	2	3	4	5	6	7	8	9	10	11
mean	0.00	0.00	0.00	-0.00	0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00	0.00
std	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01

ACP

[15]:
```
# Calcul des composantes principales
n_components = 11
```

```
# On instancie notre ACP
pca = PCA(n_components = n_components)

# On l'entraine sur nos données actuelles
pca.fit(X_scaled)
```

[15]:
```
PCA(n_components=11)
```

EXplained Variance & Scree Plot

Intéressons nous maintenant à la variance captée par chaque nouvelle composante. Grace à scikit-learn on peut utiliser l'attribut explained_variance_ratio_ :

[17]:
```
pca.explained_variance_ratio_
```

[17]:
```
array([0.41648543, 0.17415414, 0.11839333, 0.08361902, 0.06267424,
       0.0564808 , 0.04556793, 0.02456168, 0.01380381, 0.00334215,
       0.00091748])
```

La première composante (PC1) résume à elle seule 45 % de l'information totale. Les deux premières composantes résument ensemble 62.6 % (45.32 + 17.27). Les trois premières composantes → 73.8 % de la variance totale.

[19]:
```
# Enregistrement dans une variable
scree = (pca.explained_variance_ratio_*100).round(2)
scree
```

[19]:
```
array([41.65, 17.42, 11.84,  8.36,  6.27,  5.65,  4.56,  2.46,  1.38,
        0.33,  0.09])
```

Les 2 premières composantes captent donc 45+17 = 63% de la variance, les 3 premières 45+17+11 = 74% de la variance etc...

Dans le jargon, cela s'appelle une somme cumulée. Et pour faire une somme cumulée numpy dispose de la fonction cumsum :

[21]:
```
scree_cum = scree.cumsum().round()
scree_cum
```

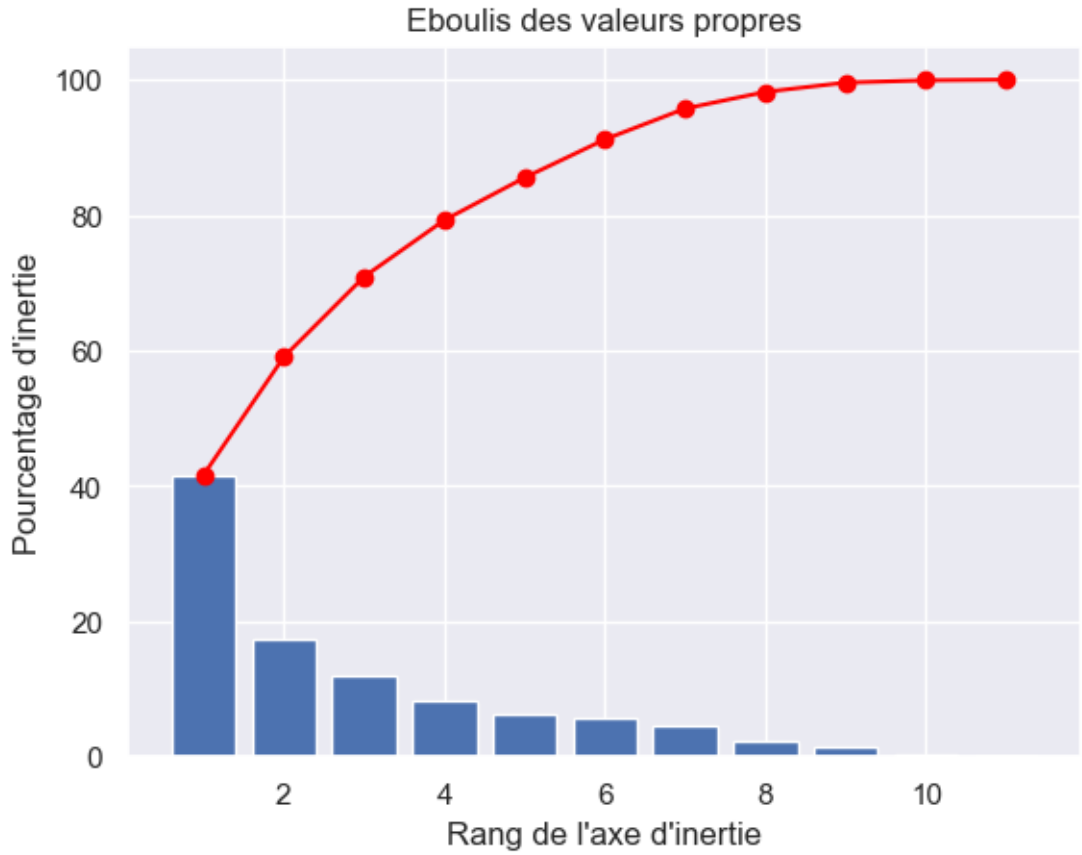
[21]:
```
array([ 42.,  59.,  71.,  79.,  86.,  91.,  96.,  98., 100., 100., 100.])
```

[22]:
```
# Définir une variable avec la liste de nos composantes
x_list = range(1, n_components+1)
list(x_list)
```

[22]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

[23]:
```
# Affichage du graphique scree plot
plt.bar(x_list, scree)
plt.plot(x_list, scree.cumsum(), c="red", marker="o")
```

```
plt.xlabel("Rang de l'axe d'inertie")
plt.ylabel("Pourcentage d'inertie")
plt.title("Eboulis des valeurs propres")

plt.show(block=False)
```



En bleu est représenté la variance de chaque nouvelle composante. En rouge, on retrouve la variance cumulée. Près de 80% de la variance est comprise dans les 4 premières composantes et près de 90% dans les 5 premières

COMPONENTS

Calcul pour obtenir ces composantes. La formule de ce calcul nous est donnée par l'attribut components_. Cette variable est généralement nommée pcs :

[27]:
```
pcs = pca.components_
pcs
```

[27]:
```
array([[ 3.94007949e-01,  4.06943669e-01,  2.09110072e-01,
         2.16685961e-01,  2.50400209e-01,  3.83489222e-01,
```

```
         2.87075539e-01, -3.33460904e-02, -3.62282605e-02,
         3.78805751e-01,  3.83570044e-01, -5.70431580e-02],
       [ 1.16359760e-01,  6.77059806e-02,  5.29736239e-01,
        -2.00572965e-02,  2.39656771e-01, -2.14320384e-01,
        -3.01798420e-01,  5.24390790e-01, -3.37397906e-02,
         1.33833495e-01, -2.14213785e-01,  4.08019145e-01],
       [ 3.20622501e-01,  2.64001682e-01, -1.57302308e-01,
        -3.68768999e-01, -3.26544713e-01, -2.12790574e-01,
         1.65504652e-02, -3.84888330e-01,  1.32335680e-01,
         3.56704917e-01, -2.12580368e-01,  4.20123754e-01],
       [-4.55069141e-02, -3.69285067e-02,  3.43498669e-02,
         2.01460982e-01,  1.01673450e-01, -8.58411479e-03,
         8.53558905e-02,  7.70173153e-03,  9.63810504e-01,
        -5.86244355e-02, -8.60021192e-03,  6.78333750e-02],
       [-1.89205619e-01, -2.10599629e-01,  6.10431024e-02,
        -5.31449926e-01,  1.00197135e-01,  2.29685737e-01,
         5.39731208e-01,  1.60210039e-01, -3.22172731e-03,
        -1.70612591e-01,  2.29547997e-01,  4.14849841e-01],
       [-6.21324173e-02, -9.02042709e-02, -1.56542403e-01,
         6.16489537e-01,  2.16992028e-01, -1.46570461e-01,
         2.74877185e-01, -2.54527940e-01, -2.22736162e-01,
        -1.22511591e-01, -1.46549004e-01,  5.37057167e-01],
       [ 8.29926891e-03,  1.05241587e-02, -3.13879936e-02,
        -3.31815075e-01,  8.05715048e-01, -1.27602277e-01,
        -1.34463209e-01, -4.05333828e-01,  7.15425831e-03,
        -4.28554392e-02, -1.27565729e-01, -1.50791846e-01],
       [-4.58966706e-02, -9.07091213e-02,  5.57810970e-01,
         1.31234629e-02, -1.32223146e-01, -3.18798133e-01,
         5.42506901e-01, -1.77260786e-01, -3.69987640e-02,
         7.92548848e-02, -3.18711749e-01, -3.52508567e-01],
       [ 1.70687916e-01,  2.55025658e-01, -5.26029486e-01,
        -3.70790105e-02,  1.69103233e-01, -2.63531728e-01,
         3.73712883e-01,  5.30765482e-01, -7.48448924e-04,
         1.55737711e-02, -2.63365860e-01, -1.96781112e-01],
       [-4.33343521e-01, -3.35625222e-01, -1.67124670e-01,
         4.14431964e-02,  9.95735828e-02,  4.17117942e-03,
         2.15070269e-03,  7.78310122e-02,  3.25677859e-03,
         8.08411562e-01,  3.96925633e-03, -1.92537009e-02],
       [-6.83807583e-01,  7.21277932e-01,  6.87082300e-02,
        -1.42675746e-02, -2.44932810e-02, -1.31778678e-02,
        -4.30363817e-03, -4.20202835e-02, -7.25829413e-03,
        -4.37276665e-02, -1.34910467e-02,  5.03621621e-02]])
```

[28]:
```
#Affichons la même chose mais version pandas :
pcs = pd.DataFrame(pcs)
pcs
```

[28]:

	0	1	2	3	4	5	6
0	0.394008	0.406944	0.209110	0.216686	0.250400	0.383489	0.287076
1	0.116360	0.067706	0.529736	-0.020057	0.239657	-0.214320	-0.301798
2	0.320623	0.264002	-0.157302	-0.368769	-0.326545	-0.212791	0.016550
3	-0.045507	-0.036929	0.034350	0.201461	0.101673	-0.008584	0.085356
4	-0.189206	-0.210600	0.061043	-0.531450	0.100197	0.229686	0.539731
5	-0.062132	-0.090204	-0.156542	0.616490	0.216992	-0.146570	0.274877
6	0.008299	0.010524	-0.031388	-0.331815	0.805715	-0.127602	-0.134463
7	-0.045897	-0.090709	0.557811	0.013123	-0.132223	-0.318798	0.542507
8	0.170688	0.255026	-0.526029	-0.037079	0.169103	-0.263532	0.373713
9	-0.433344	-0.335625	-0.167125	0.041443	0.099574	0.004171	0.002151
10	-0.683808	0.721278	0.068708	-0.014268	-0.024493	-0.013178	-0.004304

	7	8	9	10	11
0	-0.033346	-0.036228	0.378806	0.383570	-0.057043
1	0.524391	-0.033740	0.133833	-0.214214	0.408019
2	-0.384888	0.132336	0.356705	-0.212580	0.420124
3	0.007702	0.963811	-0.058624	-0.008600	0.067833
4	0.160210	-0.003222	-0.170613	0.229548	0.414850
5	-0.254528	-0.222736	-0.122512	-0.146549	0.537057
6	-0.405334	0.007154	-0.042855	-0.127566	-0.150792
7	-0.177261	-0.036999	0.079255	-0.318712	-0.352509
8	0.530765	-0.000748	0.015574	-0.263366	-0.196781
9	0.077831	0.003257	0.808412	0.003969	-0.019254
10	-0.042020	-0.007258	-0.043728	-0.013491	0.050362

[29]:
```
pcs.columns = features
pcs.index = [f"F{i}" for i in x_list]
pcs.round(2)
```

[29]:

	Disponibilité alimentaire en quantité (kg/personne/an)	Disponibilité de protéines en quantité (g/personne/jour)
F1	0.39	0.41
F2	0.12	0.07
F3	0.32	0.26
F4	-0.05	
F5	-0.19	
F6	-0.06	
F7	0.01	
F8	-0.05	
F9	0.17	
F10	-0.43	
F11	-0.68	

F4	-0.04
F5	-0.21
F6	-0.09
F7	0.01
F8	-0.09
F9	0.26
F10	-0.34
F11	0.72

	Disponibilité intérieure	Exportations - Quantité	\
F1	0.21	0.22	
F2	0.53	-0.02	
F3	-0.16	-0.37	
F4	0.03	0.20	
F5	0.06	-0.53	
F6	-0.16	0.62	
F7	-0.03	-0.33	
F8	0.56	0.01	
F9	-0.53	-0.04	
F10	-0.17	0.04	
F11	0.07	-0.01	

	Importations - Quantité	PIB par habitant	Stabilité politique	\
F1	0.25	0.38	0.29	
F2	0.24	-0.21	-0.30	
F3	-0.33	-0.21	0.02	
F4	0.10	-0.01	0.09	
F5	0.10	0.23	0.54	
F6	0.22	-0.15	0.27	
F7	0.81	-0.13	-0.13	
F8	-0.13	-0.32	0.54	
F9	0.17	-0.26	0.37	
F10	0.10	0.00	0.00	
F11	-0.02	-0.01	-0.00	

	Population	ratio_export_import	conso_par_habitant	score_agro_eco	\
F1	-0.03	-0.04	0.38	0.38	
F2	0.52	-0.03	0.13	-0.21	
F3	-0.38	0.13	0.36	-0.21	
F4	0.01	0.96	-0.06	-0.01	
F5	0.16	-0.00	-0.17	0.23	
F6	-0.25	-0.22	-0.12	-0.15	
F7	-0.41	0.01	-0.04	-0.13	
F8	-0.18	-0.04	0.08	-0.32	
F9	0.53	-0.00	0.02	-0.26	
F10	0.08	0.00	0.81	0.00	
F11	-0.04	-0.01	-0.04	-0.01	

```
     Distance_France_km
F1                -0.06
F2                 0.41
F3                 0.42
F4                 0.07
F5                 0.41
F6                 0.54
F7                -0.15
F8                -0.35
F9                -0.20
F10               -0.02
F11                0.05
```

Les résultats ont été arrondis pour simplifier l'analyse

Alors, comment calcule t-on la première composante F1 ?

et bien c'est assez simple :

F1 = (0.40 *disponibilité alimentaire) + (0.41* disponibilité de protéines) + … + (0.38 * score agro-eco)

et F2 ?

F2 = (0.04 * disponibilité alimentaire) + (0.01 * disponibilité de protéine) + … + (-0.17 * score agro-eco)

[32]:
```
#nouvelle affichage du dataframe
pcs.T
```

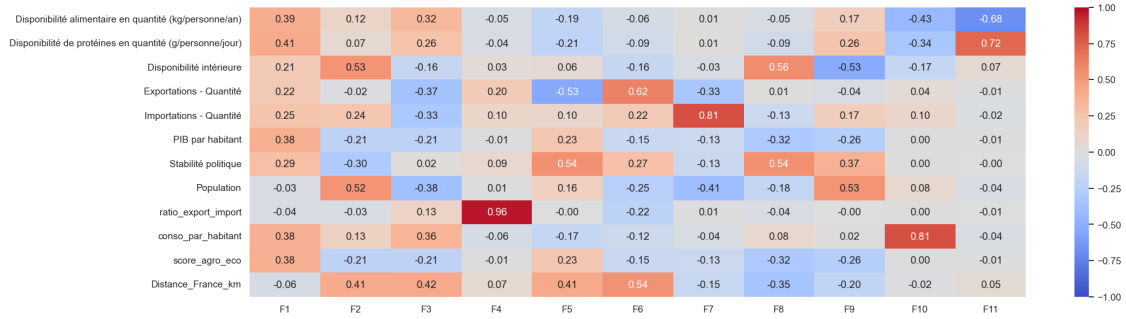
[32]:
```
                                                    F1        F2  \
Disponibilité alimentaire en quantité (kg/perso…  0.394008  0.116360
Disponibilité de protéines en quantité (g/perso…  0.406944  0.067706
Disponibilité intérieure                           0.209110  0.529736
Exportations - Quantité                            0.216686 -0.020057
Importations - Quantité                            0.250400  0.239657
PIB par habitant                                   0.383489 -0.214320
Stabilité politique                                0.287076 -0.301798
Population                                        -0.033346  0.524391
ratio_export_import                               -0.036228 -0.033740
conso_par_habitant                                 0.378806  0.133833
score_agro_eco                                     0.383570 -0.214214
Distance_France_km                                -0.057043  0.408019

                                                    F3        F4  \
Disponibilité alimentaire en quantité (kg/perso…  0.320623 -0.045507
Disponibilité de protéines en quantité (g/perso…  0.264002 -0.036929
Disponibilité intérieure                          -0.157302  0.034350
Exportations - Quantité                           -0.368769  0.201461
```

```
Importations - Quantité                              -0.326545  0.101673
PIB par habitant                                     -0.212791 -0.008584
Stabilité politique                                   0.016550  0.085356
Population                                           -0.384888  0.007702
ratio_export_import                                   0.132336  0.963811
conso_par_habitant                                    0.356705 -0.058624
score_agro_eco                                       -0.212580 -0.008600
Distance_France_km                                    0.420124  0.067833

                                                           F5        F6  \
Disponibilité alimentaire en quantité (kg/perso… -0.189206 -0.062132
Disponibilité de protéines en quantité (g/perso… -0.210600 -0.090204
Disponibilité intérieure                              0.061043 -0.156542
Exportations - Quantité                              -0.531450  0.616490
Importations - Quantité                               0.100197  0.216992
PIB par habitant                                      0.229686 -0.146570
Stabilité politique                                   0.539731  0.274877
Population                                            0.160210 -0.254528
ratio_export_import                                  -0.003222 -0.222736
conso_par_habitant                                   -0.170613 -0.122512
score_agro_eco                                        0.229548 -0.146549
Distance_France_km                                    0.414850  0.537057

                                                           F7        F8  \
Disponibilité alimentaire en quantité (kg/perso…  0.008299 -0.045897
Disponibilité de protéines en quantité (g/perso…  0.010524 -0.090709
Disponibilité intérieure                             -0.031388  0.557811
Exportations - Quantité                              -0.331815  0.013123
Importations - Quantité                               0.805715 -0.132223
PIB par habitant                                     -0.127602 -0.318798
Stabilité politique                                  -0.134463  0.542507
Population                                           -0.405334 -0.177261
ratio_export_import                                   0.007154 -0.036999
conso_par_habitant                                   -0.042855  0.079255
score_agro_eco                                       -0.127566 -0.318712
Distance_France_km                                   -0.150792 -0.352509

                                                           F9       F10  \
Disponibilité alimentaire en quantité (kg/perso…  0.170688 -0.433344
Disponibilité de protéines en quantité (g/perso…  0.255026 -0.335625
Disponibilité intérieure                             -0.526029 -0.167125
Exportations - Quantité                              -0.037079  0.041443
Importations - Quantité                               0.169103  0.099574
PIB par habitant                                     -0.263532  0.004171
Stabilité politique                                   0.373713  0.002151
Population                                            0.530765  0.077831
ratio_export_import                                  -0.000748  0.003257
```

```
conso_par_habitant                                  0.015574  0.808412
score_agro_eco                                     -0.263366  0.003969
Distance_France_km                                 -0.196781 -0.019254

                                                          F11
Disponibilité alimentaire en quantité (kg/perso… -0.683808
Disponibilité de protéines en quantité (g/perso…  0.721278
Disponibilité intérieure                           0.068708
Exportations - Quantité                           -0.014268
Importations - Quantité                           -0.024493
PIB par habitant                                  -0.013178
Stabilité politique                               -0.004304
Population                                        -0.042020
ratio_export_import                               -0.007258
conso_par_habitant                                -0.043728
score_agro_eco                                    -0.013491
Distance_France_km                                 0.050362
```

[34]:
```
# Représentation plus visuelle
fig, ax = plt.subplots(figsize=(20, 6))
sns.heatmap(pcs.T, vmin=-1, vmax=1, annot=True, cmap="coolwarm", fmt="0.2f")
```

[34]: <Axes: >

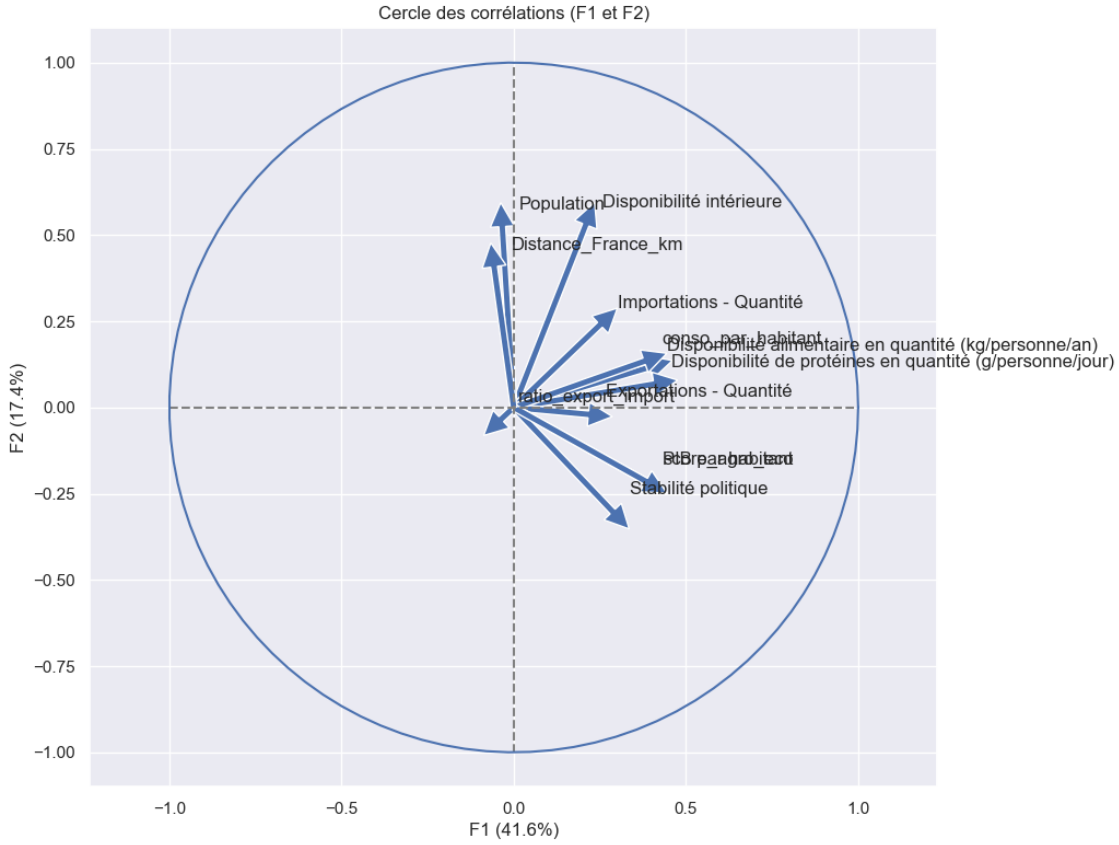


CORRELATION GRAPH

[42]:
```
# Définissons nos axes x et y. Nous allons utiliser les 2 premières composantes.
  ↪ Comme - en code - on commence à compter à partir de 0, cela nous donne :
x, y = 0,1
```

[44]:
```
# Partie graphique
fig, ax = plt.subplots(figsize=(10, 9))
for i in range(0, pca.components_.shape[1]):
    ax.arrow(0,
             0,  # Start the arrow at the origin
```

```
            pca.components_[0, i],  #0 for PC1
            pca.components_[1, i],  #1 for PC2
            head_width=0.07,
            head_length=0.07,
            width=0.02,              )

    plt.text(pca.components_[0, i] + 0.05,
            pca.components_[1, 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],1)))
plt.ylabel('F{} ({}%)'.format(y+1, round(100*pca.
 ↪explained_variance_ratio_[y],1)))

plt.title("Cercle des corrélations (F{} et F{})".format(x+1, y+1))


an = np.linspace(0, 2 * np.pi, 100)
plt.plot(np.cos(an), np.sin(an))  # Add a unit circle for scale
plt.axis('equal')
plt.show(block=False)
```



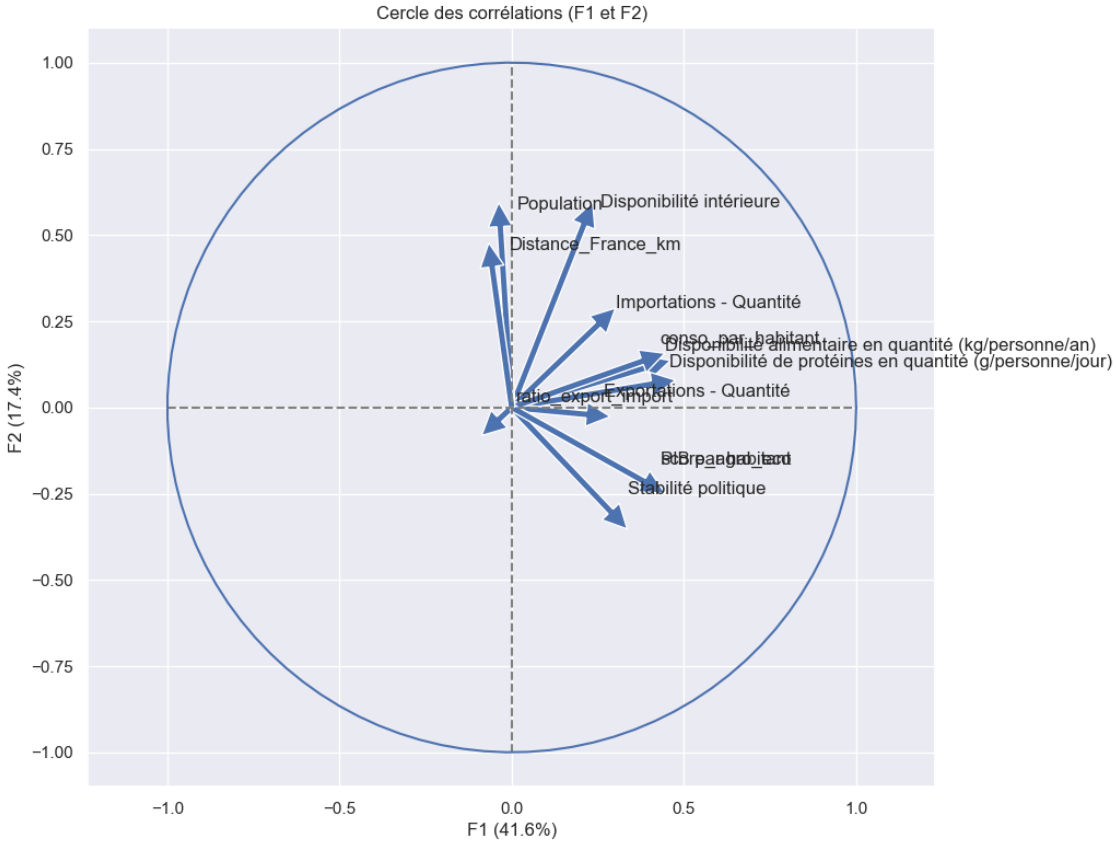
[46]:
```
# On en fait une fonction
def correlation_graph(pca,
                      x_y,
                      features) :
    """Affiche le graphe des correlations

    Positional arguments :
    -----------------------------------
    pca : sklearn.decomposition.PCA : notre objet PCA qui a été fait
    x_y : list ou tuple : le couple x,y des plans à afficher, exemple [0,1]␣
 ↪pour F1, F2
    features : list ou tuple : la liste des features (ie des dimensions) à␣
 ↪représenter
    """

    # 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],1)))
    plt.ylabel('F{} ({}%)'.format(y+1, round(100*pca.
↪explained_variance_ratio_[y],1)))

    # 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))  # Add a unit circle for scale

    # Axes et display
    plt.axis('equal')
    plt.show(block=False)
```

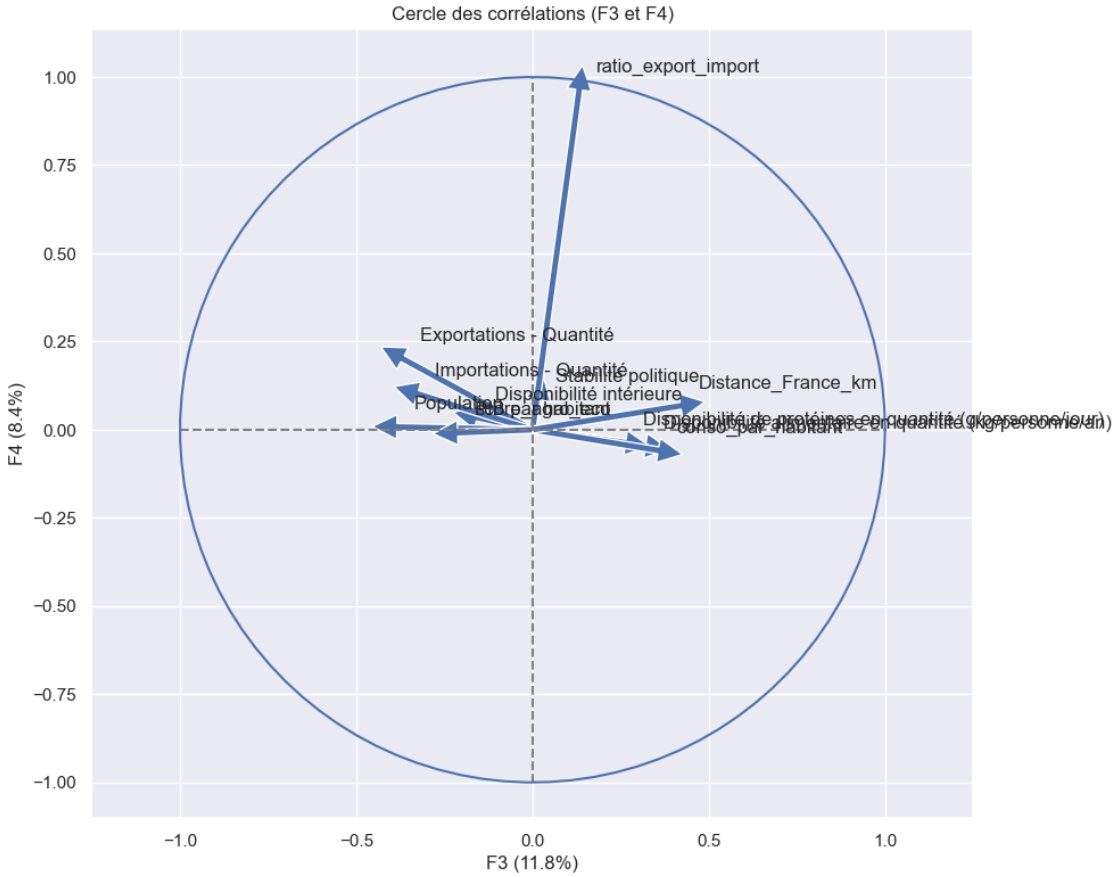
[51]:
```
# Essayons cette fonction pour F1 et F2
#-- ATTENTION -- Encore une fois Pour F1 et F2 il faut bien préciser 0 et 1
x_y = (0,1)
x_y
```

[51]: (0, 1)

[52]:
```
correlation_graph(pca, x_y, features)
```



[53]:
```
# Essayons pour F3 et F4
correlation_graph(pca, (2,3), features)
```



PROJECTION

Travail sur la projection des dimensions

[58]:
```
# Calcul des coordonnées des individus dans le nouvel espace
X_proj = pca.transform(X_scaled)
X_proj[:5]
```

[58]:
```
array([[-3.16849099e+00,  4.43342641e-01, -3.46115157e-01,
        -3.10302960e-01, -1.16409567e+00, -4.26927943e-01,
         4.60249356e-01, -1.23755978e+00, -8.28158188e-01,
        -2.07572267e-02,  4.98832714e-02],
       [ 2.12267259e+00,  2.60218657e+00,  8.75479626e-01,
         2.19896764e-02, -1.37206494e-01,  2.60408066e-01,
         1.10203600e+00,  4.64088845e-01,  2.12165711e-01,
        -1.50156877e-01,  2.46720131e-01],
       [-1.71323711e+00, -5.16447921e-01, -6.39400463e-01,
        -2.45745391e-01, -3.29774709e-01, -6.54781089e-01,
         2.18654385e-02,  6.76680984e-02, -6.52933937e-02,
        -1.86605133e-03, -6.08080865e-02],
```

```
       [ 3.75022991e+00,  3.80751063e-01, -3.95074178e+00,
         8.61687234e-01, -5.04122870e-01,  1.17333018e+00,
         7.19807735e-01,  7.79535615e-02,  7.80131946e-02,
         3.30853351e-01, -7.35506100e-02],
       [-1.09790459e+00,  2.27845156e-01, -4.68789151e-03,
        -1.49109408e-02,  3.75058820e-01,  5.43663784e-01,
         8.81233544e-01, -5.27841469e-02,  1.96630159e-01,
         5.19641804e-02,  8.14890030e-04]])
```

[61]:
```
# Rappel
x_y
```

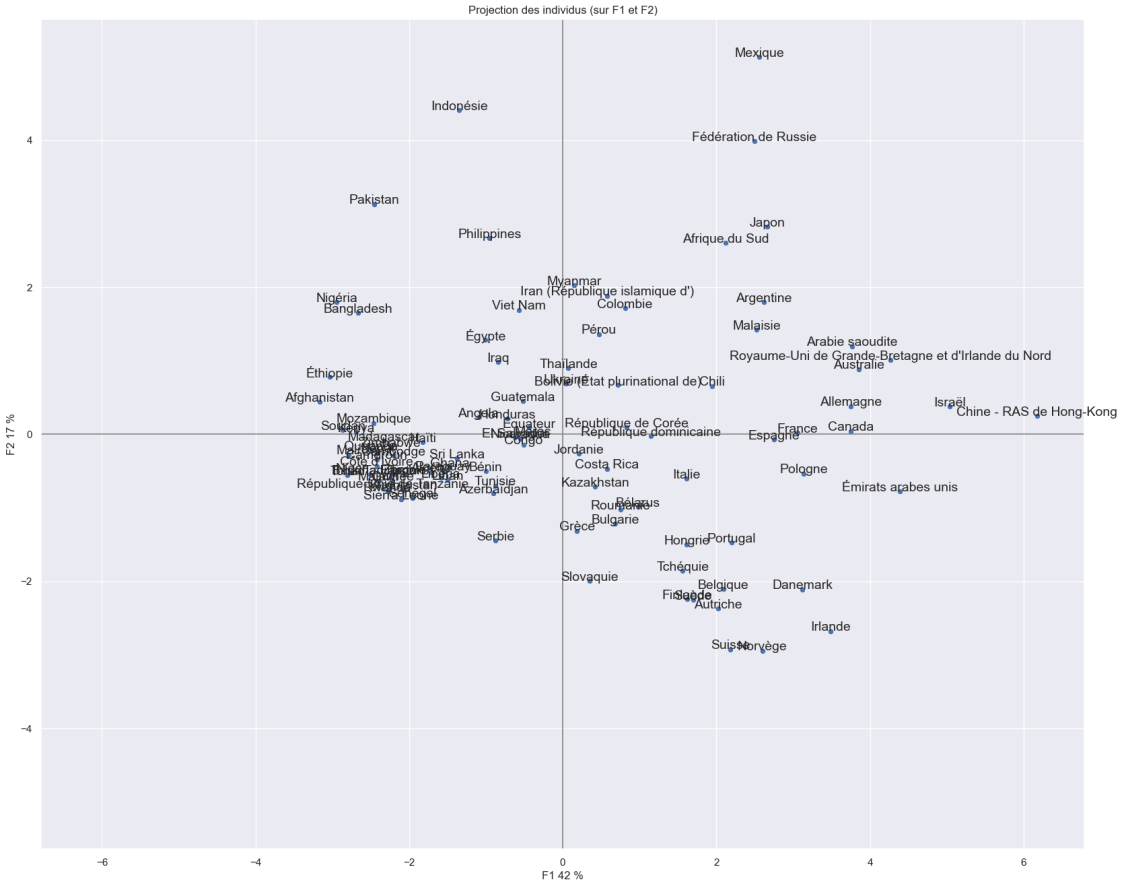
[61]: (0, 1)

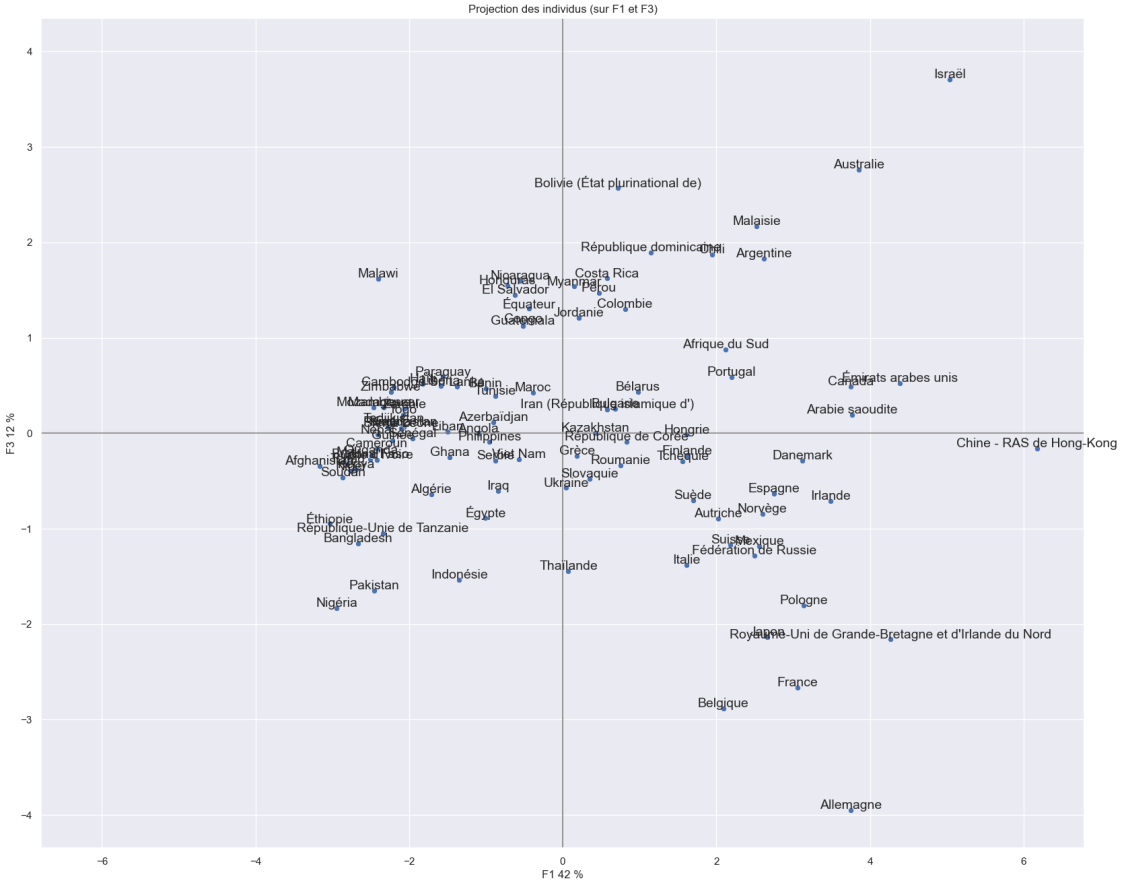
[62]:
```
def display_factorial_planes(   X_projected,
                                x_y,
                                pca=None,
                                labels = None,
                                clusters=None,
                                alpha=1,
                                figsize=[10,8],
                                marker="." ):
    """
    Affiche la projection des individus

    Positional arguments :
    -----------------------------------
    X_projected : np.array, pd.DataFrame, list of list : la matrice des points␣
↪projetés
    x_y : list ou tuple : le couple x,y des plans à afficher, exemple [0,1]␣
↪pour F1, F2

    Optional arguments :
    -----------------------------------
    pca : sklearn.decomposition.PCA : un objet PCA qui a été fit, cela nous␣
↪permettra d'afficher la variance de chaque composante, default = None
    labels : list ou tuple : les labels des individus à projeter, default = None
    clusters : list ou tuple : la liste des clusters auquel appartient chaque␣
↪individu, default = None
    alpha : float in [0,1] : paramètre de transparence, 0=100% transparent,␣
↪1=0% transparent, default = 1
    figsize : list ou tuple : couple width, height qui définit la taille de la␣
↪figure en inches, default = [10,8]
    marker : str : le type de marker utilisé pour représenter les individus,␣
↪points croix etc etc, default = "."
    """
```

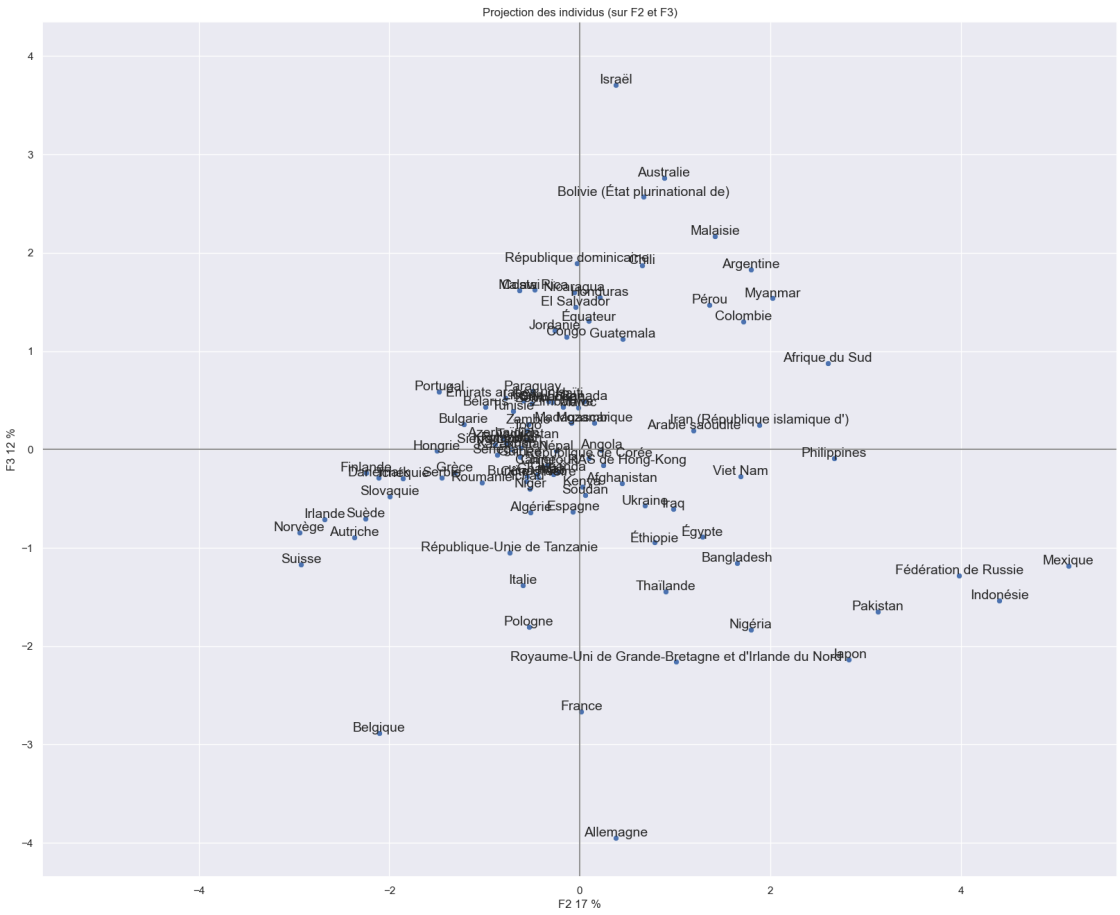
```
    # Transforme X_projected en np.array
    X_ = np.array(X_projected)

    # On définit la forme de la figure si elle n'a pas été donnée
    if not figsize:
        figsize = (7,6)

    # On gère les labels
    if  labels is None :
        labels = []
    try :
        len(labels)
    except Exception as e :
        raise e

    # On vérifie la variable axis
    if not len(x_y) ==2 :
        raise AttributeError("2 axes sont demandées")
    if max(x_y )>= X_.shape[1] :
        raise AttributeError("la variable axis n'est pas bonne")

    # on définit x et y
    x, y = x_y

    # Initialisation de la figure
    fig, ax = plt.subplots(1, 1, figsize=figsize)

    # On vérifie s'il y a des clusters ou non
    c = None if clusters is None else clusters

    # Les points
    # plt.scatter(   X_[:, x], X_[:, y], alpha=alpha,
    #                     c=c, cmap="Set1", marker=marker)
    sns.scatterplot(data=None, x=X_[:, x], y=X_[:, y], hue=c)

    # Si la variable pca a été fournie, on peut calculer le % de variance de␣
↪chaque axe
    if pca :
        v1 = str(round(100*pca.explained_variance_ratio_[x]))  + " %"
        v2 = str(round(100*pca.explained_variance_ratio_[y]))  + " %"
    else :
        v1=v2= ''

    # Nom des axes, avec le pourcentage d'inertie expliqué
    ax.set_xlabel(f'F{x+1} {v1}')
    ax.set_ylabel(f'F{y+1} {v2}')
```

```
    # Valeur x max et y max
    x_max = np.abs(X_[:, x]).max() *1.1
    y_max = np.abs(X_[:, y]).max() *1.1

    # On borne x et y
    ax.set_xlim(left=-x_max, right=x_max)
    ax.set_ylim(bottom= -y_max, top=y_max)

    # Affichage des lignes horizontales et verticales
    plt.plot([-x_max, x_max], [0, 0], color='grey', alpha=0.8)
    plt.plot([0,0], [-y_max, y_max], color='grey', alpha=0.8)

    # Affichage des labels des points
    if len(labels) :
        # j'ai copié collé la fonction sans la lire
        for i,(_x,_y) in enumerate(X_[:,[x,y]]):
            plt.text(_x, _y+0.05, labels[i], fontsize='14',␣
 ↪ha='center',va='center')

   # Titre et display
    plt.title(f"Projection des individus (sur F{x+1} et F{y+1})")
    plt.show()
```

[63]:
```
X_proj = pca.transform(X_scaled)
display_factorial_planes(X_proj, (0,1), pca, labels=names, figsize=(20,16),␣
 ↪marker="o")
display_factorial_planes(X_proj, (0,2), pca, labels=names, figsize=(20,16),␣
 ↪marker="o")
display_factorial_planes(X_proj, (1,2), pca, labels=names, figsize=(20,16),␣
 ↪marker="o")
```


Projection des individus (sur F1 et F2)
Mexique
Indonésie
Fédération de Russie
Pakistan
Japon
Philippines
Afrique du Sud
Myanmar
Iran (République islamique d')
Nigéria
Bangladesh
Viet Nam
Colombie
Argentine
Malaisie
Égypte
Pérou
Arabie saoudite
Royaume-Uni de Grande-Bretagne et d'Irlande du Nord
Iraq
Thaïlande
Australie
Éthiopie
Ukraine
Bolivie (État plurinational de)
Chili
Afghanistan
Guatemala
Allemagne
Israël
Chine - RAS de Hong-Kong
Mozambique
Honduras
Équateur
République de Corée
France
Canada
Haïti
République dominicaine
Espagne
Jordanie
Sri Lanka
Costa Rica
Pologne
Italie
Émirats arabes unis
Kazakhstan
Tunisie
Azerbaïdjan
Bélarus
Bulgarie
Grèce
Serbie
Hongrie
Portugal
Tchéquie
Slovaquie
Belgique
Danemark
Suède
Autriche
Irlande
Suisse
Norvège
F2 17 %
F1 42 %
4
2
0
−2
−4
−6
−4
−2
0
2
4
6


Projection des individus (sur F1 et F3)
F3 12 %
F1 42 %
Israël
Australie
Bolivie (État plurinational de)
Malaisie
République dominicaine
Chili
Argentine
Malawi
Nicaragua
Costa Rica
Honduras
Myanmar
El Salvador
Pérou
Équateur
Colombie
Jordanie
Afrique du Sud
Portugal
Émirats arabes unis
Canada
Bélarus
Maroc
Tunisie
Arabie saoudite
Azerbaïdjan
Kazakhstan
Hongrie
Chine - RAS de Hong-Kong
Philippines
République de Corée
Grèce
Finlande
Danemark
Ghana
Viet Nam
Roumanie
Slovaquie
Ukraine
Algérie
Iraq
Suède
Espagne
Irlande
Égypte
Autriche
Norvège
Éthiopie
République-Unie de Tanzanie
Bangladesh
Suisse
Mexique
Fédération de Russie
Italie
Thaïlande
Indonésie
Pakistan
Nigéria
Pologne
Japon
Royaume-Uni de Grande-Bretagne et d'Irlande du Nord
France
Belgique
Allemagne



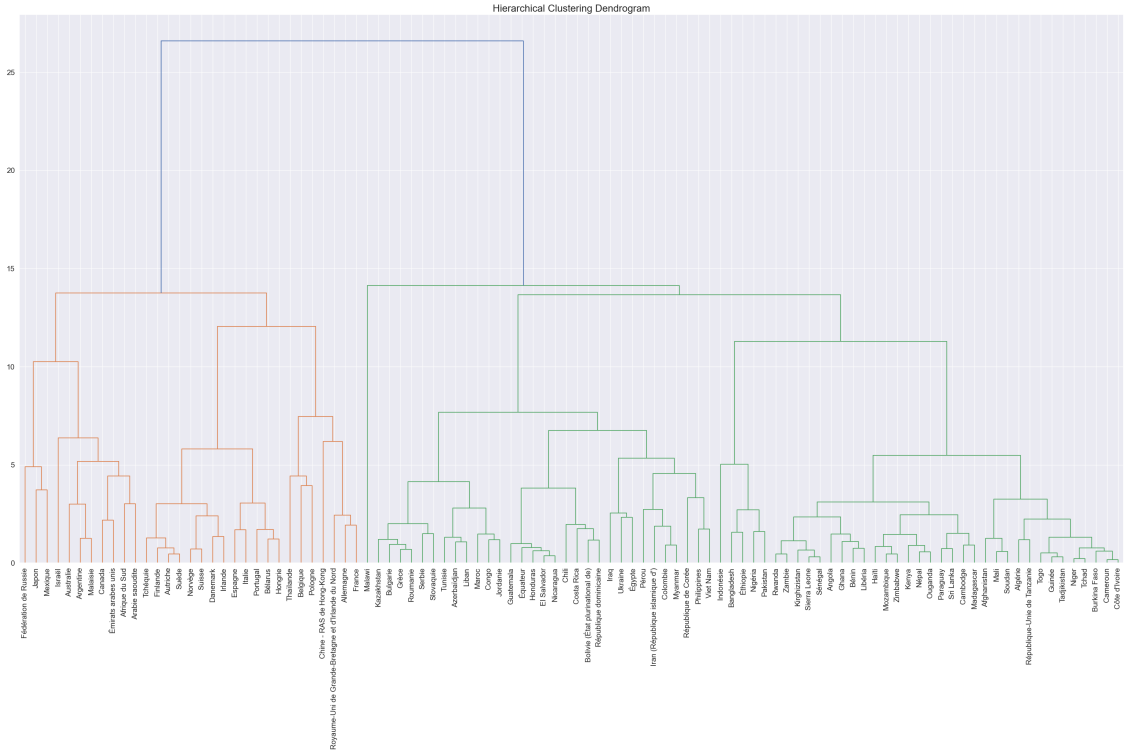
III_ CLASSIFICATION ASCENDANTE HIERARCHIQUE

[65]:
```
# Préparation des données pour le clustering
X = df.values
features = df.index
```

[66]:
```
# On instancie
scaler = StandardScaler()

# On fit
scaler.fit(X)

# On transforme
X_scaled = scaler.transform(X)
```

[67]:
```
# Clustering hiérarchique
Z = linkage(X_scaled, "ward")
```

[68]:
```
def plot_dendogram(Z, names):
    fig, ax = plt.subplots(1, 1, figsize=(40, 20))

    _ = dendrogram(Z, ax=ax, labels=names, orientation = "top")

    plt.title("Hierarchical Clustering Dendrogram", fontsize=20)
    ax.tick_params(axis='x', which='major', labelsize=15)
    ax.tick_params(axis='y', which='major', labelsize=15)
    plt.show()

plot_dendogram(Z, names)
```



[69]:
```
from sklearn.cluster import AgglomerativeClustering

agglo = AgglomerativeClustering(n_clusters=4)
labels = agglo.fit_predict(X)  # ou X_scaled si tu n'as pas utilisé l'ACP

df['cluster'] = labels
```

III – Détermination du nombre optimal de clusters

Méthode du coude (Elbow Method)

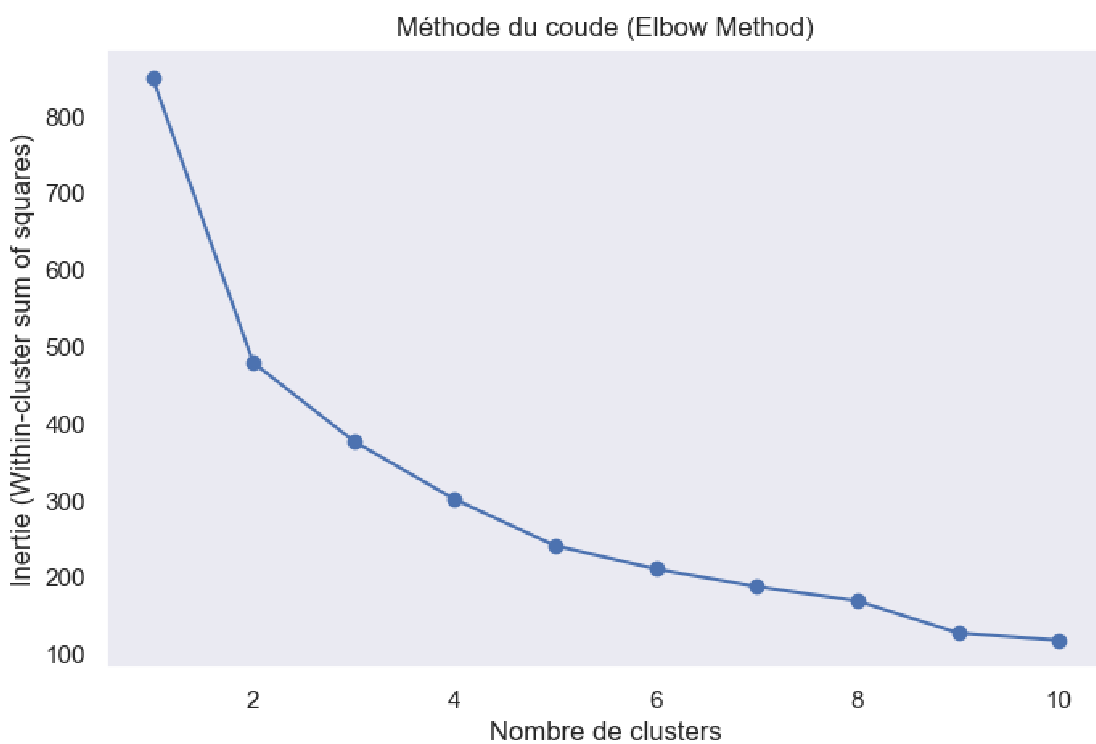
But : Identifier à partir de combien de clusters l'amélioration du score d'inertie (somme des distances intra-cluster) devient marginale.

[74]:

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# On utilise les composantes principales : ici, on prend les 2 ou 3 premières
X_pca = pca.transform(X_scaled)[:, :3]  # 3 premières composantes ( 74% de la␣
↪variance)

inertias = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_pca)
    inertias.append(kmeans.inertia_)

# Affichage du graphique "elbow"
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertias, marker='o')
plt.title("Méthode du coude (Elbow Method)")
plt.xlabel("Nombre de clusters")
plt.ylabel("Inertie (Within-cluster sum of squares)")
plt.grid()
plt.show()
```

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

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



Interprétation : Le “coude” du graphique correspond à un changement de pente marqué : c’est là que l’ajout d’un cluster supplémentaire n’apporte plus beaucoup d’information. Ce nombre de clusters sera utilisé pour le k-means.

nombre de clusters choisis 4

2. Méthode du score de silhouette

But : Mesurer la qualité de la séparation des clusters. Plus le score est proche de 1, mieux les points sont regroupés.

[82]:
```
from sklearn.metrics import silhouette_score

silhouette_scores = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_pca)
    score = silhouette_score(X_pca, labels)
    silhouette_scores.append(score)

# Affichage du score silhouette en fonction de k
plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), silhouette_scores, marker='s', color='orange')
plt.title("Score de silhouette en fonction du nombre de clusters")
plt.xlabel("Nombre de clusters")
plt.ylabel("Score de silhouette")
plt.grid()
plt.show()
```

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

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

Interprétation : Le nombre de clusters qui maximise le score de silhouette est celui où les groupes sont les mieux formés (dense et bien séparés).

IV – Clustering (k-means)

[87]:
```
# Choix du nombre de clusters optimal trouvé (à adapter selon résultats␣
↪précédents)
optimal_k = 3

kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(X_pca)

# Ajout des clusters dans le DataFrame original
df["cluster"] = clusters
```

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

V – Analyse descriptive des clusters

[91]:
```
# Moyenne des indicateurs par cluster
df.groupby("cluster").mean(numeric_only=True).round(2)
```

[91]:

cluster	Disponibilité alimentaire en quantité (kg/personne/an)	Disponibilité de protéines en quantité (g/personne/jour)	Disponibilité intérieure	Exportations - Quantité	Importations - Quantité	PIB par habitant	Stabilité politique	Population	ratio_export_import	conso_par_habitant	score_agro_eco	Distance_France_km
0	8.50	2.85	157.13	1.54	38.10	7427.28	-0.70	28226850.15	1061538.33	0.01	2973.25	5589.76
1	24.98	8.45	1848.11	93.78	212.56	18859.50	-0.54	86664654.11	24.37	0.03	7551.13	8798.60
2	27.85	10.78	638.97	183.53	217.47	50868.14	0.55	23028500.60	100001.58	0.03	20355.78	2964.42

Analyse des clusters : Cluster 0 : Faibles disponibilités alimentaires et protéiques. PIB faible (7915) et forte instabilité politique (-0.74). Très faible score agro-éco. Pays probablement pauvres, peu stables, dépendants des importations.

Cluster 1 : Meilleures disponibilités que le cluster 0, mais moins élevées que le cluster 2. PIB modéré (24442), stabilité politique négative (-0.32). Score agro-éco intermédiaire. Ces pays ont une production et un commerce alimentaire moyen, plus développés que ceux du cluster 0.

Cluster 2 : Disponibilités alimentaires et protéines élevées. PIB élevé (54982) et bonne stabilité politique (0.74). Score agro-éco très élevé. Ce cluster correspond sans doute à des pays riches, politiquement stables et autosuffisants, avec fortes exportations et une balance commerciale alimentaire plus équilibrée (ratio export/import proche de 1). Et distance la plus proche de la France.

VI – Liste des pays par groupe

[94]:
```
for k in range(optimal_k):
    print(f"\nCluster {k} :")
    display(df[df["cluster"] == k].index.tolist())
```

```
Cluster 0 :
```

```
['Afghanistan',
 'Algérie',
 'Angola',
 'Azerbaïdjan',
 'Bangladesh',
 'Burkina Faso',
 'Bénin',
 'Cambodge',
 'Cameroun',
 'Congo',
 'Costa Rica',
 "Côte d'Ivoire",
 'El Salvador',
 'Ghana',
 'Grèce',
 'Guatemala',
 'Guinée',
 'Haïti',
 'Honduras',
 'Iraq',
 'Jordanie',
 'Kenya',
 'Kirghizistan',
 'Liban',
 'Libéria',
 'Madagascar',
 'Malawi',
 'Mali',
 'Maroc',
 'Mozambique',
 'Nicaragua',
 'Niger',
 'Nigéria',
 'Népal',
 'Ouganda',
 'Paraguay',
 'Rwanda',
 'République-Unie de Tanzanie',
 'Serbie',
 'Sierra Leone',
```

```
 'Soudan',
 'Sri Lanka',
 'Sénégal',
 'Tadjikistan',
 'Tchad',
 'Togo',
 'Tunisie',
 'Zambie',
 'Zimbabwe',
 'Égypte',
 'Équateur',
 'Éthiopie']
```

```
Cluster 1 :
```

```
['Afrique du Sud',
 'Argentine',
 'Bolivie (État plurinational de)',
 'Chili',
 'Colombie',
 'Fédération de Russie',
 'Indonésie',
 "Iran (République islamique d')",
 'Japon',
 'Malaisie',
 'Mexique',
 'Myanmar',
 'Pakistan',
 'Philippines',
 'Pérou',
 'Thaïlande',
 'Ukraine',
 'Viet Nam']
```

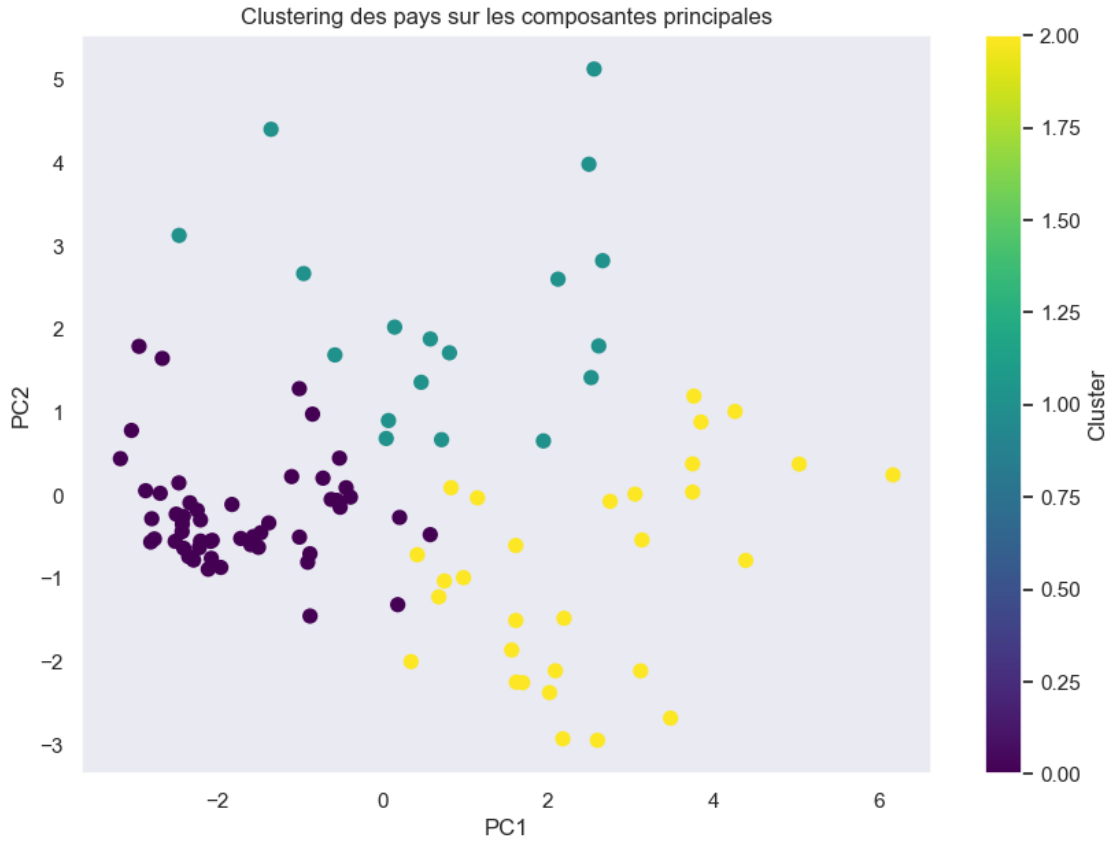
```
Cluster 2 :
```

```
['Allemagne',
 'Arabie saoudite',
 'Australie',
 'Autriche',
 'Belgique',
 'Bulgarie',
 'Bélarus',
 'Canada',
 'Chine - RAS de Hong-Kong',
 'Danemark',
 'Espagne',
 'Finlande',
```

```
'France',
'Hongrie',
'Irlande',
'Israël',
'Italie',
'Kazakhstan',
'Norvège',
'Pologne',
'Portugal',
'Roumanie',
"Royaume-Uni de Grande-Bretagne et d'Irlande du Nord",
'République de Corée',
'République dominicaine',
'Slovaquie',
'Suisse',
'Suède',
'Tchéquie',
'Émirats arabes unis']
```

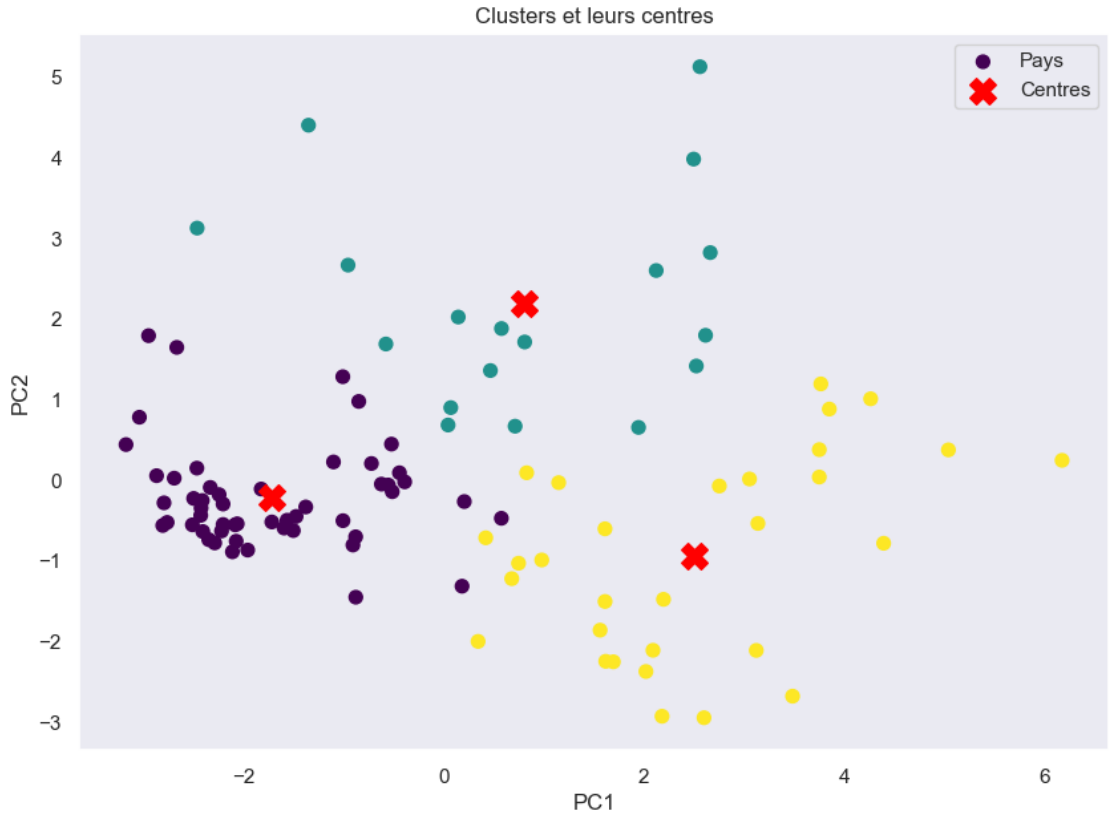
VII – Visualisation des clusters sur les composantes principales

[96]:
```
plt.figure(figsize=(10, 7))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis', s=50)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Clustering des pays sur les composantes principales")
plt.colorbar(label="Cluster")
plt.grid()
plt.show()
```



VIII – Visualisation des centres de clusters

[98]:
```
centroids = kmeans.cluster_centers_

plt.figure(figsize=(10, 7))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis', s=50,␣
 ↪label='Pays')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='X',␣
 ↪label='Centres')
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("Clusters et leurs centres")
plt.legend()
plt.grid()
plt.show()
```



[99]:
```
#score_prometteur" dans df pour identifier les pays les plus attractifs à␣
 ↪cibler pour l'exportation de volaille
# Étape 1 : Ajouter la distance dans les variables utilisées
variables_utiles = df[['PIB par habitant', 'Stabilité politique', 'Importations␣
 ↪- Quantité',
                       'ratio_export_import', 'Distance_France_km']].copy()

# Étape 2 : Inverser les variables où un faible score est meilleur
variables_utiles['ratio_export_import'] = 1 /␣
 ↪(variables_utiles['ratio_export_import'] + 1e-6)
variables_utiles['Distance_France_km'] = 1 /␣
 ↪(variables_utiles['Distance_France_km'] + 1e-6)

# Étape 3 : Normaliser
scaler = MinMaxScaler()
variables_normalisées = scaler.fit_transform(variables_utiles)

# Étape 4 : Calcul du score
df['score_prometteur'] = variables_normalisées.mean(axis=1)
```

[100]:
```
df.columns
```

[100]:
```
Index(['Disponibilité alimentaire en quantité (kg/personne/an)',
       'Disponibilité de protéines en quantité (g/personne/jour)',
       'Disponibilité intérieure', 'Exportations - Quantité',
       'Importations - Quantité', 'PIB par habitant', 'Stabilité politique',
       'Population', 'ratio_export_import', 'conso_par_habitant',
       'score_agro_eco', 'Distance_France_km', 'cluster', 'score_prometteur'],
      dtype='object')
```

[105]:
```
df['Pays'] = df.index
```

[106]:
```
top5_pays = df.sort_values(by='score_prometteur', ascending=False).head(6)
print(top5_pays[['Pays', 'score_prometteur', 'cluster']])
```

```
                                          Pays  score_prometteur  cluster
Zone
Norvège                                Norvège          0.628342        2
France                                  France          0.570507        2
Belgique                              Belgique          0.557499        2
Allemagne                            Allemagne          0.545497        2
Chine - RAS de Hong-Kong  Chine - RAS de Hong-Kong          0.503979        2
Japon                                    Japon          0.497712        1
```

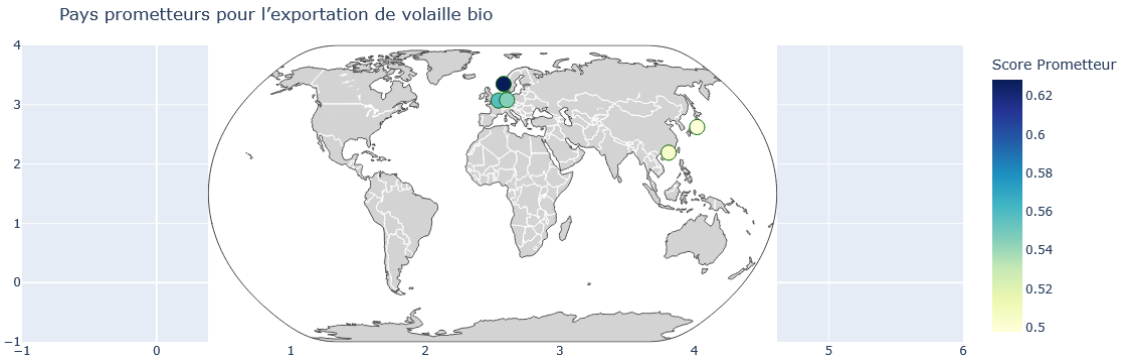
[107]:
```
top5_pays = df[df['Pays'] != 'France'] \
              .sort_values(by='score_prometteur', ascending=False) \
              .head(5)

print(top5_pays[['Pays', 'score_prometteur', 'cluster']])
```

```
                                          Pays  score_prometteur  cluster
Zone
Norvège                                Norvège          0.628342        2
Belgique                              Belgique          0.557499        2
Allemagne                            Allemagne          0.545497        2
Chine - RAS de Hong-Kong  Chine - RAS de Hong-Kong          0.503979        2
Japon                                    Japon          0.497712        1
```

[108]:
```
import plotly.graph_objects as go
import pandas as pd

# Données avec URLs de drapeaux (wikimedia ou other CDN)
data = pd.DataFrame({
    'Pays': ['Norvège', 'Belgique', 'Allemagne', 'Hong Kong (RAS Chine)',␣
 ↪'Japon'],
    'Latitude': [60.472, 50.850, 51.165, 22.3193, 36.2048],
    'Longitude': [8.4689, 4.3517, 10.4515, 114.1694, 138.2529],
    'Score': [0.628342, 0.557499, 0.545497, 0.503979, 0.497712],
```

```
    'Drapeau': [
        'https://flagcdn.com/w80/no.png',
        'https://flagcdn.com/w80/be.png',
        'https://flagcdn.com/w80/de.png',
        'https://flagcdn.com/w80/hk.png',
        'https://flagcdn.com/w80/jp.png'
    ]
})

# Créer la carte de fond
fig = go.Figure()

# Ajout des marqueurs pour les scores (coloration)
fig.add_trace(go.Scattergeo(
    lon = data['Longitude'],
    lat = data['Latitude'],
    text = data['Pays'] + "<br>Score: " + data['Score'].round(3).astype(str),
    mode = 'markers',
    marker=dict(
        size=15,
        color=data['Score'],
        colorscale='YlGnBu',
        colorbar=dict(title="Score Prometteur"),
        line=dict(width=1, color='darkgreen')
    ),
    hoverinfo='text'
))

# Ajout des images de drapeaux
for i, row in data.iterrows():
    fig.add_layout_image(
        dict(
            source=row['Drapeau'],
            xref="x",
            yref="y",
            x=row['Longitude'],
            y=row['Latitude'],
            sizex=5,
            sizey=5,
            xanchor="center",
            yanchor="middle",
            layer="above"
        )
    )

# Mise en forme de la carte
fig.update_geos(
```

```
    projection_type="natural earth",
    showcountries=True,
    countrycolor="white",
    showland=True,
    landcolor="lightgrey"
)

fig.update_layout(
    title="Pays prometteurs pour l'exportation de volaille bio",
    margin=dict(l=0, r=0, t=50, b=0)
)

fig.show()
```



[ ]: