Entrée [1]:
```
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
from sklearn.cluster import KMeans
import os
os.environ["OMP_NUM_THREADS"] = "1"  # Évite les fuites de mémoire sur Windo
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from pandas.plotting import parallel_coordinates
import plotly.graph_objects as go
import seaborn as sns
from thefuzz import fuzz, process
from fuzzywuzzy import process
```

Entrée [2]:
```
dispo = pd.read_csv("DisponibiliteAlimentaire_2017.csv",sep=",")
pop = pd.read_csv("Population_2000_2018.csv",sep=",")
abord = pd.read_csv("Cout et abordabilité.csv",sep=",")
dist = pd.read_excel("Distance.xlsx")
dtf = pd.read_excel("DTF.xlsx")
kfc = pd.read_excel("KFC.xlsx")
pib = pd.read_excel("PIB.xlsx", header=3)
elec = pd.read_csv("Electricité.csv",sep=",", header=5)
stabpol = pd.read_excel("Stab Pol.xlsx")
trad = pd.read_excel("traduction anglais français.xlsx")
```

Disponibilité Alimentaire

Entrée [3]:
```
dispo.head()
```

Out[3]:

	Code Domaine	Domaine	Code zone	Zone	Code Élément	Élément	Code Produit	Produit	Code année	An
0	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5511	Production	2511	Blé et produits	2017	2
1	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5611	Importations - Quantité	2511	Blé et produits	2017	2
2	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5072	Variation de stock	2511	Blé et produits	2017	2
3	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5911	Exportations - Quantité	2511	Blé et produits	2017	2
4	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5301	Disponibilité intérieure	2511	Blé et produits	2017	2

Entrée [4]:
```
dispo.shape
```

Out[4]:
```
(176600, 14)
```

Entrée [5]:
```
dispo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 176600 entries, 0 to 176599
Data columns (total 14 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   Code Domaine            176600 non-null  object 
 1   Domaine                 176600 non-null  object 
 2   Code zone               176600 non-null  int64  
 3   Zone                    176600 non-null  object 
 4   Code Élément            176600 non-null  int64  
 5   Élément                 176600 non-null  object 
 6   Code Produit            176600 non-null  int64  
 7   Produit                 176600 non-null  object 
 8   Code année              176600 non-null  int64  
 9   Année                   176600 non-null  int64  
 10  Unité                   176600 non-null  object 
 11  Valeur                  176600 non-null  float64
 12  Symbole                 176600 non-null  object 
 13  Description du Symbole  176600 non-null  object 
dtypes: float64(1), int64(5), object(8)
memory usage: 18.9+ MB
```

Entrée [6]:
```
dispo.isna().mean()
```

Out[6]:
```
Code Domaine              0.0
Domaine                   0.0
Code zone                 0.0
Zone                      0.0
Code Élément              0.0
Élément                   0.0
Code Produit              0.0
Produit                   0.0
Code année                0.0
Année                     0.0
Unité                     0.0
Valeur                    0.0
Symbole                   0.0
Description du Symbole    0.0
dtype: float64
```

Entrée [7]:
```
dispo.duplicated().sum()
```

Out[7]:
```
0
```

Entrée [8]:
```
dispo.nunique()
```

Out[8]:
```
Code Domaine                1
Domaine                     1
Code zone                 174
Zone                      174
Code Élément               17
Élément                    17
Code Produit               98
Produit                    98
Code année                  1
Année                       1
Unité                       4
Valeur                   7250
Symbole                     2
Description du Symbole      2
dtype: int64
```

Population

Entrée [9]:
```
pop.head()
```

Out[9]:

	Code Domaine	Domaine	Code zone	Zone	Code Élément	Élément	Code Produit	Produit	Code année
0	OA	Séries temporelles annuelles	2	Afghanistan	511	Population totale	3010	Population-Estimations	2000
1	OA	Séries temporelles annuelles	2	Afghanistan	511	Population totale	3010	Population-Estimations	2001
2	OA	Séries temporelles annuelles	2	Afghanistan	511	Population totale	3010	Population-Estimations	2002
3	OA	Séries temporelles annuelles	2	Afghanistan	511	Population totale	3010	Population-Estimations	2003
4	OA	Séries temporelles annuelles	2	Afghanistan	511	Population totale	3010	Population-Estimations	2004

Entrée [10]:
```
pop.shape
```

Out[10]:
```
(4411, 15)
```

Entrée [11]:
```
pop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4411 entries, 0 to 4410
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Code Domaine            4411 non-null   object 
 1   Domaine                 4411 non-null   object 
 2   Code zone               4411 non-null   int64  
 3   Zone                    4411 non-null   object 
 4   Code Élément            4411 non-null   int64  
 5   Élément                 4411 non-null   object 
 6   Code Produit            4411 non-null   int64  
 7   Produit                 4411 non-null   object 
 8   Code année              4411 non-null   int64  
 9   Année                   4411 non-null   int64  
 10  Unité                   4411 non-null   object 
 11  Valeur                  4411 non-null   float64
 12  Symbole                 4411 non-null   object 
 13  Description du Symbole  4411 non-null   object 
 14  Note                    258 non-null    object 
dtypes: float64(1), int64(5), object(9)
memory usage: 517.0+ KB
```

Entrée [12]:
```
pop.isna().mean()
```

Out[12]:
```
Code Domaine              0.00000
Domaine                   0.00000
Code zone                 0.00000
Zone                      0.00000
Code Élément              0.00000
Élément                   0.00000
Code Produit              0.00000
Produit                   0.00000
Code année                0.00000
Année                     0.00000
Unité                     0.00000
Valeur                    0.00000
Symbole                   0.00000
Description du Symbole    0.00000
Note                      0.94151
dtype: float64
```

Entrée [13]:
```
pop.duplicated().sum()
```

Out[13]:
```
0
```

Entrée [14]:
```
pop.nunique()
```

Out[14]:
```
Code Domaine                1
Domaine                     1
Code zone                 238
Zone                      238
Code Élément                1
Élément                     1
Code Produit                1
Produit                     1
Code année                 19
Année                      19
Unité                       1
Valeur                   4398
Symbole                     2
Description du Symbole      2
Note                        1
dtype: int64
```

Entrée [15]:
```
print(pop['Note'].unique())
```

```
[nan
 'UNDESA, Population Division – World Population Prospects, the 2017 Revis
ion']
```

Entrée [16]:
```
pop = pop[pop['Année'] == 2017]
```

Entrée [17]:
```
pop = pop[['Code zone','Zone','Unité','Valeur','Symbole','Description du Sy
pop
```

Out[17]:

	Code zone	Zone	Unité	Valeur	Symbole	Description du Symbole
17	2	Afghanistan	1000 personnes	36296.113	X	Sources internationales sûres
36	202	Afrique du Sud	1000 personnes	57009.756	X	Sources internationales sûres
55	3	Albanie	1000 personnes	2884.169	X	Sources internationales sûres
74	4	Algérie	1000 personnes	41389.189	X	Sources internationales sûres
93	79	Allemagne	1000 personnes	82658.409	X	Sources internationales sûres
...	...	...	...	...	...	...
4333	236	Venezuela (République bolivarienne du)	1000 personnes	29402.484	X	Sources internationales sûres
4352	237	Viet Nam	1000 personnes	94600.648	X	Sources internationales sûres
4371	249	Yémen	1000 personnes	27834.819	X	Sources internationales sûres
4390	251	Zambie	1000 personnes	16853.599	X	Sources internationales sûres
4409	181	Zimbabwe	1000 personnes	14236.595	X	Sources internationales sûres

236 rows × 6 columns

Entrée [18]:
```
print(pop['Unité'].unique())
```

```
['1000 personnes']
```

Entrée [19]:
```
pop['Valeur'] = pop['Valeur'] /1000
pop = pop[['Zone', 'Valeur']]
```

Entrée [20]:
```
pop = pop.rename(columns={'Valeur': 'Population en M'})
pop
```

Out[20]:

	Zone	Population en M
17	Afghanistan	36.296113
36	Afrique du Sud	57.009756
55	Albanie	2.884169
74	Algérie	41.389189
93	Allemagne	82.658409
...	...	...
4333	Venezuela (République bolivarienne du)	29.402484
4352	Viet Nam	94.600648
4371	Yémen	27.834819
4390	Zambie	16.853599
4409	Zimbabwe	14.236595

236 rows × 2 columns

Entrée [21]:
```
doubpop = pop['Zone'][pop['Zone'].duplicated()]
doubpop
```

Out[21]:
```
Series([], Name: Zone, dtype: object)
```

Entrée [22]:
```
pop
```

Out[22]:

	Zone	Population en M
17	Afghanistan	36.296113
36	Afrique du Sud	57.009756
55	Albanie	2.884169
74	Algérie	41.389189
93	Allemagne	82.658409
...	...	...
4333	Venezuela (République bolivarienne du)	29.402484
4352	Viet Nam	94.600648
4371	Yémen	27.834819
4390	Zambie	16.853599
4409	Zimbabwe	14.236595

236 rows × 2 columns

Disponibilité Alimentaire

Entrée [23]:
```
print(dispo['Produit'].unique())
```

```
['Blé et produits' 'Riz et produits' 'Orge et produits' 'Maïs et produits'
 'Seigle et produits' 'Avoine' 'Millet et produits' 'Sorgho et produits'
 'Céréales, Autres' 'Pommes de Terre et produits' 'Ignames' 'Racines nda'
 'Sucre, canne' 'Sucre, betterave' 'Sucre Eq Brut' 'Edulcorants Autres'
 'Miel' 'Haricots' 'Pois' 'Légumineuses Autres et produits'
 'Noix et produits' 'Soja' 'Arachides Decortiquees' 'Graines de tournesol'
 'Graines Colza/Moutarde' 'Graines de coton' 'Coco (Incl Coprah)' 'Sésame'
 'Olives' 'Plantes Oleiferes, Autre' 'Huile de Soja' "Huile d'Arachide"
 'Huile de Tournesol' 'Huile de Colza&Moutarde' 'Huile Graines de Coton'
 'Huile de Palmistes' 'Huile de Palme' 'Huile de Coco' 'Huile de Sésame'
 "Huile d'Olive" 'Huile de Son de Riz' 'Huile de Germe de Maïs'
 'Huil Plantes Oleif Autr' 'Tomates et produits' 'Oignons'
 'Légumes, Autres' 'Oranges, Mandarines' 'Citrons & Limes et produits'
 'Pamplemousse et produits' 'Agrumes, Autres' 'Bananes'
 'Pommes et produits' 'Ananas et produits' 'Dattes' 'Raisin'
 'Fruits, Autres' 'Café et produits' 'Feve de Cacao et produits' 'Thé'
 'Poivre' 'Piments' 'Girofles' 'Épices, Autres' 'Vin' 'Bière'
 'Boissons Fermentés' 'Boissons Alcooliques' 'Alcool, non Comestible'
 'Viande de Bovins' "Viande d'Ovins/Caprins" 'Viande de Suides'
 'Viande de Volailles' 'Viande, Autre' 'Abats Comestible' 'Beurre, Ghee'
 'Crème' 'Graisses Animales Crue' 'Oeufs' 'Lait - Excl Beurre'
 'Poissons Eau Douce' 'Aliments pour enfants' 'Miscellanees'
 'Manioc et produits' 'Patates douces' 'Palmistes' 'Bananes plantains'
 'Huiles de Poissons' 'Huiles de Foie de Poisso' 'Perciform'
 'Poissons Pelagiques' 'Poissons Marins, Autres' 'Crustacés'
 'Cephalopodes' 'Mollusques, Autres' 'Animaux Aquatiques Autre'
 'Plantes Aquatiques' 'Sucre non centrifugé' 'Viande de Anim Aquatiq']
```

Entrée [24]:
```
print(dispo['Élément'].unique())
```

```
['Production' 'Importations - Quantité' 'Variation de stock'
 'Exportations - Quantité' 'Disponibilité intérieure'
 'Aliments pour animaux' 'Semences' 'Pertes' 'Résidus' 'Nourriture'
 'Disponibilité alimentaire en quantité (kg/personne/an)'
 'Disponibilité alimentaire (Kcal/personne/jour)'
 'Disponibilité de protéines en quantité (g/personne/jour)'
 'Disponibilité de matière grasse en quantité (g/personne/jour)'
 'Traitement' 'Autres utilisations (non alimentaire)'
 'Alimentation pour touristes']
```

Entrée [25]:
```
print(dispo['Unité'].unique())
```

```
['Milliers de tonnes' 'kg' 'Kcal/personne/jour' 'g/personne/jour']
```

Entrée [26]:
```
dispo = dispo[dispo['Produit'] == 'Viande de Volailles']
dispo.head()
```

Out[26]:

	Code Domaine	Domaine	Code zone	Zone	Code Élément	Élément	Code Produit	Produit	Code année
651	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5511	Production	2734	Viande de Volailles	2017
652	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5611	Importations - Quantité	2734	Viande de Volailles	2017
653	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5072	Variation de stock	2734	Viande de Volailles	2017
654	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5301	Disponibilité intérieure	2734	Viande de Volailles	2017
655	FBS	Nouveaux Bilans Alimentaire	2	Afghanistan	5123	Pertes	2734	Viande de Volailles	2017

Entrée [27]:
```
dispo = dispo[dispo['Élément'].isin(['Production', 'Importations - Quantité
                                     'Exportations - Quantité',
                                     'Disponibilité alimentaire en quantité (kg/perso
dispo = dispo.pivot(index=['Zone', 'Produit', 'Année'],
                    columns='Élément',
                    values='Valeur')
dispo.head()
```

Out[27]:

			Élément	Disponibilité alimentaire en quantité (kg/personne/an)	Exportations - Quantité	Importations - Quantité	Production
Zone	**Produit**	**Année**					
Afghanistan	**Viande de Volailles**	**2017**		1.53	NaN	29.0	28.0
Afrique du Sud	**Viande de Volailles**	**2017**		35.69	63.0	514.0	1667.0
Albanie	**Viande de Volailles**	**2017**		16.36	0.0	38.0	13.0
Algérie	**Viande de Volailles**	**2017**		6.38	0.0	2.0	275.0
Allemagne	**Viande de Volailles**	**2017**		19.47	646.0	842.0	1514.0

Entrée [28]:
```
dispo = dispo.rename(columns={
    'Disponibilité alimentaire en quantité (kg/personne/an)': 'dispo alim (
    'Exportations - Quantité': 'Exportation (mT)',
    'Importations - Quantité': 'Importation (mT)',
    'Production': 'Production (mT)'
})
dispo
```

Out[28]:

		Élément	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)
Zone	**Produit**	**Année**				
Afghanistan	**Viande de Volailles**	**2017**	1.53	NaN	29.0	28.0
Afrique du Sud	**Viande de Volailles**	**2017**	35.69	63.0	514.0	1667.0
Albanie	**Viande de Volailles**	**2017**	16.36	0.0	38.0	13.0
Algérie	**Viande de Volailles**	**2017**	6.38	0.0	2.0	275.0
Allemagne	**Viande de Volailles**	**2017**	19.47	646.0	842.0	1514.0
...	**...**	**...**	...	...	...	...
Émirats arabes unis	**Viande de Volailles**	**2017**	43.47	94.0	433.0	48.0
Équateur	**Viande de Volailles**	**2017**	19.31	0.0	0.0	340.0
États-Unis d'Amérique	**Viande de Volailles**	**2017**	55.68	3692.0	123.0	21914.0
Éthiopie	**Viande de Volailles**	**2017**	0.13	NaN	1.0	14.0
Îles Salomon	**Viande de Volailles**	**2017**	4.45	0.0	6.0	0.0

172 rows × 4 columns

Entrée [29]:
```
dispo = dispo.reset_index()
dispo = dispo.drop(columns=['Produit','Année'])
dispo
```

Out[29]:

Élément	Zone	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)
0	Afghanistan	1.53	NaN	29.0	28.0
1	Afrique du Sud	35.69	63.0	514.0	1667.0
2	Albanie	16.36	0.0	38.0	13.0
3	Algérie	6.38	0.0	2.0	275.0
4	Allemagne	19.47	646.0	842.0	1514.0
...	...	...	...	...	...
167	Émirats arabes unis	43.47	94.0	433.0	48.0
168	Équateur	19.31	0.0	0.0	340.0
169	États-Unis d'Amérique	55.68	3692.0	123.0	21914.0
170	Éthiopie	0.13	NaN	1.0	14.0
171	Îles Salomon	4.45	0.0	6.0	0.0

172 rows × 5 columns

Entrée [30]:
```
doubdispo = dispo['Zone'][dispo['Zone'].duplicated()]
print(doubdispo)
```

```
Series([], Name: Zone, dtype: object)
```

Entrée [31]:
```
def find_best_match(row, choices, scorer=fuzz.ratio, seuil=80):
    best_match = process.extractOne(row, choices, scorer=scorer)
    return best_match[0] if best_match and best_match[1] >= seuil else None

pop['Zone'] = pop['Zone'].apply(lambda x: find_best_match(x, dispo['Zone'].u
df = pop.merge(dispo, on='Zone', how='left')
df
```

Out[31]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)
0	Afghanistan	36.296113	1.53	NaN	29.0	28.0
1	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0
2	Albanie	2.884169	16.36	0.0	38.0	13.0
3	Algérie	41.389189	6.38	0.0	2.0	275.0
4	Allemagne	82.658409	19.47	646.0	842.0	1514.0
...	...	...	...	...	...	...
231	Venezuela (République bolivarienne du)	29.402484	20.28	0.0	25.0	600.0
232	Viet Nam	94.600648	12.33	1.0	291.0	918.0
233	Yémen	27.834819	8.53	0.0	78.0	168.0
234	Zambie	16.853599	3.42	1.0	12.0	49.0
235	Zimbabwe	14.236595	4.68	NaN	6.0	69.0

236 rows × 6 columns

Entrée [32]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                         64
Population en M               0
dispo alim (kg/pers/an)      64
Exportation (mT)            101
Importation (mT)             66
Production (mT)              68
dtype: int64
```

Entrée [33]:
```
df['Exportation (mT)'] = df['Exportation (mT)'].fillna(0)
df['Importation (mT)'] = df['Importation (mT)'].fillna(0)
df['Production (mT)'] = df['Production (mT)'].fillna(0)
df
```

Out[33]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)
0	Afghanistan	36.296113	1.53	0.0	29.0	28.0
1	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0
2	Albanie	2.884169	16.36	0.0	38.0	13.0
3	Algérie	41.389189	6.38	0.0	2.0	275.0
4	Allemagne	82.658409	19.47	646.0	842.0	1514.0
...	...	...	...	...	...	...
231	Venezuela (République bolivarienne du)	29.402484	20.28	0.0	25.0	600.0
232	Viet Nam	94.600648	12.33	1.0	291.0	918.0
233	Yémen	27.834819	8.53	0.0	78.0	168.0
234	Zambie	16.853599	3.42	1.0	12.0	49.0
235	Zimbabwe	14.236595	4.68	0.0	6.0	69.0

236 rows × 6 columns

Entrée [34]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        64
Population en M              0
dispo alim (kg/pers/an)     64
Exportation (mT)             0
Importation (mT)             0
Production (mT)              0
dtype: int64
```

Entrée [35]:
```
df = df.dropna(subset=['dispo alim (kg/pers/an)'])
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
dtype: int64
```

Abordabilité alimentaire

Entrée [36]:
```
abord.head()
```

Out[36]:

	Code Domaine	Domaine	Code zone (M49)	Zone	Code Élément	Élément	Code Produit	Produit	Code année	Anne
0	CAHD	Coût et abordabilité d'une alimentation saine\...	4	Afghanistan	6132	Valeur	7006	Number of people unable to afford a healthy di...	2017	20
1	CAHD	Coût et abordabilité d'une alimentation saine\...	710	Afrique du Sud	6132	Valeur	7006	Number of people unable to afford a healthy di...	2017	20
2	CAHD	Coût et abordabilité d'une alimentation saine\...	8	Albanie	6132	Valeur	7006	Number of people unable to afford a healthy di...	2017	20
3	CAHD	Coût et abordabilité d'une alimentation saine\...	12	Algérie	6132	Valeur	7006	Number of people unable to afford a healthy di...	2017	20
4	CAHD	Coût et abordabilité d'une alimentation saine\...	276	Allemagne	6132	Valeur	7006	Number of people unable to afford a healthy di...	2017	20

Entrée [37]:
```
abord.shape
```

Out[37]:
```
(203, 15)
```

Entrée [38]:
```
abord.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 203 entries, 0 to 202
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   Code Domaine            203 non-null    object
 1   Domaine                 203 non-null    object
 2   Code zone (M49)         203 non-null    int64 
 3   Zone                    203 non-null    object
 4   Code Élément            203 non-null    int64 
 5   Élément                 203 non-null    object
 6   Code Produit            203 non-null    int64 
 7   Produit                 203 non-null    object
 8   Code année              203 non-null    int64 
 9   Année                   203 non-null    int64 
 10  Édition                 203 non-null    object
 11  Unité                   203 non-null    object
 12  Valeur                  150 non-null    object
 13  Symbole                 203 non-null    object
 14  Description du Symbole  203 non-null    object
dtypes: int64(5), object(10)
memory usage: 23.9+ KB
```

Entrée [39]:
```
abord.isna().mean()
```

Out[39]:
```
Code Domaine              0.000000
Domaine                   0.000000
Code zone (M49)           0.000000
Zone                      0.000000
Code Élément              0.000000
Élément                   0.000000
Code Produit              0.000000
Produit                   0.000000
Code année                0.000000
Année                     0.000000
Édition                   0.000000
Unité                     0.000000
Valeur                    0.261084
Symbole                   0.000000
Description du Symbole    0.000000
dtype: float64
```

Entrée [40]:
```
abord.duplicated().sum()
```

Out[40]:
```
0
```

Entrée [41]:
```
abord.nunique()
```

Out[41]:
```
Code Domaine               1
Domaine                    1
Code zone (M49)          203
Zone                     203
Code Élément               1
Élément                    1
Code Produit               1
Produit                    1
Code année                 1
Année                      1
Édition                    1
Unité                      1
Valeur                    82
Symbole                    2
Description du Symbole     2
dtype: int64
```

Entrée [42]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(abord.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Code Domaine               0
Domaine                    0
Code zone (M49)            0
Zone                       0
Code Élément               0
Élément                    0
Code Produit               0
Produit                    0
Code année                 0
Année                      0
Édition                    0
Unité                      0
Valeur                    53
Symbole                    0
Description du Symbole     0
dtype: int64
```

Entrée [43]:
```
abord1 = abord[abord['Description du Symbole'] == 'Valeur manquante']
abord1 = abord1['Zone']
abord1
```

Out[43]:
```
0                                    Afghanistan
5                                        Andorre
7                              Antigua-et-Barbuda
8                                 Arabie saoudite
14                                       Bahamas
15                                       Bahreïn
17                                       Barbade
22                                      Bermudes
28                               Brunéi Darussalam
33                                      Cambodge
37                         Chine - RAS de Hong-Kong
38                             Chine - RAS de Macao
48                                          Cuba
51                                     Dominique
53                                   El Salvador
56                                      Érythrée
68                                       Géorgie
72                                     Groenland
75                               Guinée équatoriale
81                                       Îles Cook
82                                   Îles Marshall
83                                    Îles Salomon
98                                       Kiribati
99                                         Koweït
104                                         Libye
118                 Micronésie (États fédérés de)
124                                         Nauru
129                                         Nioué
131                              Nouvelle-Calédonie
132                                Nouvelle-Zélande
133                                         Oman
137                                        Palaos
140                          Papouasie-Nouvelle-Guinée
146                                Polynésie française
147                                     Porto Rico
150                          République arabe syrienne
157    République populaire démocratique de Corée
163                                Saint-Kitts-et-Nevis
164                    Saint-Vincent-et-les Grenadines
165                                           Samoa
166                                 Samoa américaines
172                                       Singapour
175                                         Somalie
177                                     Soudan du Sud
186                                       Timor-Leste
188                                          Tokélaou
189                                             Tonga
192                                      Turkménistan
194                                            Tuvalu
195                                           Ukraine
197                                          Vanuatu
198          Venezuela (République bolivarienne du)
200                                             Yémen
Name: Zone, dtype: object
```

Entrée [44]:
```
print(abord['Unité'].unique())
```

```
['millions de No']
```

Entrée [45]:
```
print(abord['Produit'].unique())
```

```
['Number of people unable to afford a healthy diet (NUA), million']
```

Entrée [46]:
```
print(abord['Domaine'].unique())
```

```
['Coût et abordabilité d’une alimentation saine\r\n (CoAHD)']
```

Entrée [47]:
```
abord = abord[['Zone', 'Valeur']]
abord = abord.rename(columns={'Valeur': 'Abordabilité alimentaire en M'})
abord
```

Out[47]:

	Zone	Abordabilité alimentaire en M
0	Afghanistan	NaN
1	Afrique du Sud	35.2
2	Albanie	0.7
3	Algérie	7.7
4	Allemagne	2.3
...	...	...
198	Venezuela (République bolivarienne du)	NaN
199	Viet Nam	10.6
200	Yémen	NaN
201	Zambie	13.6
202	Zimbabwe	11.1

203 rows × 2 columns

Entrée [48]:
```
doubabord = abord['Zone'][abord['Zone'].duplicated()]
print(doubabord)
```

```
Series([], Name: Zone, dtype: object)
```

Entrée [49]:
```
def find_best_match(row, choices, scorer=fuzz.ratio, seuil=80):
    if pd.isna(row):  # Gérer les valeurs NaN/None
        return None
    best_match = process.extractOne(str(row), choices, scorer=scorer)  # Co
    return best_match[0] if best_match and best_match[1] >= seuil else None

# Appliquer le matching et faire la jointure
pop['Zone'] = pop['Zone'].apply(lambda x: find_best_match(x, abord['Zone'].
df = df.merge(abord, on='Zone', how='left')
```

Entrée [50]:
```
df
```

Out[50]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	Abordabilité alimentaire en M
0	Afghanistan	36.296113	1.53	0.0	29.0	28.0	NaN
1	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	35.2
2	Albanie	2.884169	16.36	0.0	38.0	13.0	0.7
3	Algérie	41.389189	6.38	0.0	2.0	275.0	7.7
4	Allemagne	82.658409	19.47	646.0	842.0	1514.0	2.3
...	...	...	...	...	...	...	...
167	Venezuela (République bolivarienne du)	29.402484	20.28	0.0	25.0	600.0	NaN
168	Viet Nam	94.600648	12.33	1.0	291.0	918.0	10.6
169	Yémen	27.834819	8.53	0.0	78.0	168.0	NaN
170	Zambie	16.853599	3.42	1.0	12.0	49.0	13.6
171	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	11.1

172 rows × 7 columns

Entrée [51]:
```
df = df.dropna(subset=['Abordabilité alimentaire en M'])
df
```

Out[51]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	Abordabilité alimentaire en M
1	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	35.2
2	Albanie	2.884169	16.36	0.0	38.0	13.0	0.7
3	Algérie	41.389189	6.38	0.0	2.0	275.0	7.7
4	Allemagne	82.658409	19.47	646.0	842.0	1514.0	2.3
5	Angola	29.816766	10.56	0.0	277.0	42.0	19
...	...	...	...	...	...	...	...
163	Turquie	81.116450	20.64	429.0	3.0	2192.0	9.7
165	Uruguay	3.436641	9.12	3.0	3.0	33.0	1.1
168	Viet Nam	94.600648	12.33	1.0	291.0	918.0	10.6
170	Zambie	16.853599	3.42	1.0	12.0	49.0	13.6
171	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	11.1

142 rows × 7 columns

Entrée [52]:
```
df = df.copy()
df['Abordabilité alimentaire en M'] = pd.to_numeric(df['Abordabilité aliment
df['Population en M'] = pd.to_numeric(df['Population en M'], errors='coerce
```

Entrée [53]:
```
df['% pouvant manger'] = (1 - (df['Abordabilité alimentaire en M'] / df['Po
df['% pouvant manger'] = df['% pouvant manger'].round(1)
df
```

Out[53]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	Abordabilité alimentaire en M	p r
1	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	35.2	
2	Albanie	2.884169	16.36	0.0	38.0	13.0	0.7	
3	Algérie	41.389189	6.38	0.0	2.0	275.0	7.7	
4	Allemagne	82.658409	19.47	646.0	842.0	1514.0	2.3	
5	Angola	29.816766	10.56	0.0	277.0	42.0	19.0	
...	...	...	...	...	...	...	...	
163	Turquie	81.116450	20.64	429.0	3.0	2192.0	9.7	
165	Uruguay	3.436641	9.12	3.0	3.0	33.0	1.1	
168	Viet Nam	94.600648	12.33	1.0	291.0	918.0	10.6	
170	Zambie	16.853599	3.42	1.0	12.0	49.0	13.6	
171	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	11.1	

142 rows × 8 columns

Entrée [54]:
```
df.drop('Abordabilité alimentaire en M', axis=1, inplace=True)
```

Entrée [55]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            7
dtype: int64
```

Entrée [56]:
```
lignes_non_renseignees = df[df.isnull().any(axis=1)]
lignes_non_renseignees
```

Out[56]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger
63	Grenade	0.110874	45.70	0.0	7.0	1.0	NaN
77	Islande	0.334393	29.95	0.0	2.0	10.0	NaN
90	Liban	6.819373	10.74	4.0	13.0	64.0	NaN
93	Luxembourg	0.591910	18.33	1.0	11.0	0.0	NaN
98	Maldives	0.496402	13.50	0.0	12.0	0.0	NaN
100	Malte	0.437933	27.17	0.0	9.0	4.0	NaN
139	Sainte-Lucie	0.180954	56.69	0.0	10.0	1.0	NaN

Entrée [57]:
```
df = df.drop(lignes_non_renseignees.index)
df
```

Out[57]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger
1	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3
2	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7
3	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4
4	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2
5	Angola	29.816766	10.56	0.0	277.0	42.0	36.3
...	...	...	...	...	...	...	...
163	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0
165	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0
168	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8
170	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3
171	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0

135 rows × 7 columns

Entrée [58]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
dtype: int64
```

Entrée [59]:
```
print(df['Zone'].unique())
```

```
['Afrique du Sud' 'Albanie' 'Algérie' 'Allemagne' 'Angola' 'Argentine'
 'Arménie' 'Australie' 'Autriche' 'Azerbaïdjan' 'Bangladesh' 'Bélarus'
 'Belgique' 'Belize' 'Bénin' 'Bolivie (État plurinational de)'
 'Bosnie-Herzégovine' 'Botswana' 'Brésil' 'Bulgarie' 'Burkina Faso'
 'Cabo Verde' 'Cameroun' 'Canada' 'Chili' 'Chine, continentale'
 'Chine, Taiwan Province de' 'Chypre' 'Colombie' 'Congo' 'Costa Rica'
 "Côte d'Ivoire" 'Croatie' 'Danemark' 'Djibouti' 'Égypte'
 'Émirats arabes unis' 'Équateur' 'Espagne' 'Estonie' 'Eswatini'
 "États-Unis d'Amérique" 'Éthiopie' 'Fédération de Russie' 'Fidji'
 'Finlande' 'France' 'Gabon' 'Gambie' 'Ghana' 'Grèce' 'Guatemala' 'Guinée'
 'Guinée-Bissau' 'Guyana' 'Haïti' 'Honduras' 'Hongrie' 'Inde' 'Indonésie'
 "Iran (République islamique d')" 'Iraq' 'Irlande' 'Israël' 'Italie'
 'Jamaïque' 'Japon' 'Jordanie' 'Kazakhstan' 'Kenya' 'Kirghizistan'
 'Lesotho' 'Lettonie' 'Libéria' 'Lituanie' 'Macédoine du Nord'
 'Madagascar' 'Malaisie' 'Malawi' 'Mali' 'Maroc' 'Maurice' 'Mauritanie'
 'Mexique' 'Mongolie' 'Monténégro' 'Mozambique' 'Myanmar' 'Namibie'
 'Népal' 'Nicaragua' 'Niger' 'Nigéria' 'Norvège' 'Ouganda' 'Ouzbékistan'
 'Pakistan' 'Panama' 'Paraguay' 'Pérou' 'Philippines' 'Pologne' 'Portugal'
 'République centrafricaine' 'République de Corée' 'République de Moldova'
 'République démocratique populaire lao' 'République dominicaine'
 'République-Unie de Tanzanie' 'Roumanie'
 "Royaume-Uni de Grande-Bretagne et d'Irlande du Nord" 'Rwanda'
 'Sao Tomé-et-Principe' 'Sénégal' 'Serbie' 'Sierra Leone' 'Slovaquie'
 'Slovénie' 'Soudan' 'Sri Lanka' 'Suède' 'Suisse' 'Suriname' 'Tadjikistan'
 'Tchad' 'Tchéquie' 'Thaïlande' 'Togo' 'Trinité-et-Tobago' 'Tunisie'
 'Turquie' 'Uruguay' 'Viet Nam' 'Zambie' 'Zimbabwe']
```

DTF

Entrée [60]:
```
dtf.head()
```

Out[60]:

	Unnamed: 0	Economy	DB 2019	DB 2020
0	NaN	Afghanistan	44.2	44.1
1	NaN	Afrique du Sud	66.7	67.0
2	NaN	Albanie	67.0	67.7
3	NaN	Algérie	48.5	48.6
4	NaN	Allemagne	79.3	79.7

Entrée [61]:
```
dtf.shape
```

Out[61]:
```
(213, 4)
```

Entrée [62]:
```
dtf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213 entries, 0 to 212
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  0 non-null      float64
 1   Economy     213 non-null    object 
 2   DB 2019     213 non-null    float64
 3   DB 2020     213 non-null    float64
dtypes: float64(3), object(1)
memory usage: 6.8+ KB
```

Entrée [63]:
```
dtf['Evolution DTF(%)'] = ((dtf['DB 2020'] - dtf['DB 2019']) / dtf['DB 2019
dtf
```

Out[63]:

	Unnamed: 0	Economy	DB 2019	DB 2020	Evolution DTF(%)
0	NaN	Afghanistan	44.2	44.1	-0.226244
1	NaN	Afrique du Sud	66.7	67.0	0.449775
2	NaN	Albanie	67.0	67.7	1.044776
3	NaN	Algérie	48.5	48.6	0.206186
4	NaN	Allemagne	79.3	79.7	0.504414
...	...	...	...	...	...
208	NaN	Venezuela, RB	32.1	30.2	-5.919003
209	NaN	Viet Nam	68.6	69.8	1.749271
210	NaN	Yémen, République du	30.7	31.8	3.583062
211	NaN	Zambie	65.7	66.9	1.826484
212	NaN	Zimbabwe	50.5	54.5	7.920792

213 rows × 5 columns

Entrée [64]:
```
dtf = dtf[['Economy','DB 2020']]
dtf.head()
```

Out[64]:

	Economy	DB 2020
0	Afghanistan	44.1
1	Afrique du Sud	67.0
2	Albanie	67.7
3	Algérie	48.6
4	Allemagne	79.7

Entrée [65]:
```
dtf = dtf.rename(columns={'Economy': 'Zone'})
dtf.head()
```

Out[65]:

	Zone	DB 2020
0	Afghanistan	44.1
1	Afrique du Sud	67.0
2	Albanie	67.7
3	Algérie	48.6
4	Allemagne	79.7

Entrée [66]:
```
def find_best_match(row, choices, scorer=fuzz.ratio, seuil=80):
    if pd.isna(row):  # Gérer les valeurs NaN/None
        return None
    best_match = process.extractOne(str(row), choices, scorer=scorer)  # Co
    return best_match[0] if best_match and best_match[1] >= seuil else None

pop['Zone'] = pop['Zone'].apply(lambda x: find_best_match(x, dtf['Zone'].un
df = df.merge(dtf, on='Zone', how='left')
```

Entrée [67]:
```
df
```

Out[67]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
130	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
131	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
132	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
133	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
134	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

135 rows × 8 columns

Entrée [68]:
```
lignes_non_renseignees = df[df.isnull().any(axis=1)]
lignes_non_renseignees
```

Out[68]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
26	Chine, Taiwan Province de	23.674546	33.17	8.0	161.0	652.0	94.1	NaN
54	Guyana	0.775222	38.34	0.0	0.0	31.0	61.3	NaN
112	Sao Tomé-et-Principe	0.207089	9.47	0.0	2.0	1.0	51.7	NaN

Entrée [69]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
DB 2020                     3
dtype: int64
```

Stabilité Politique

Entrée [70]:
```
stabpol
```

Out[70]:

	codeindyr	code	countryname	year	indicator	estimate	stddev	nsource	pctr
0	AFGcc1996	AFG	Afghanistan	1996	cc	-1.291705	0.340507	2	4.30
1	ALBcc1996	ALB	Albania	1996	cc	-0.893903	0.315914	3	19.354
2	DZAcc1996	DZA	Algeria	1996	cc	-0.566741	0.262077	4	33.333
3	ASMcc1996	ASM	American Samoa	1996	cc	..	..	..	
4	ADOcc1996	ADO	Andorra	1996	cc	1.318143	0.480889	1	87.096
...	...	...	...	...	...	...	...	...	
32095	VIRva2023	VIR	Virgin Islands (U.S.)	2023	va	..	..	..	
32096	WBGva2023	WBG	West Bank and Gaza	2023	va	-1.118067	0.149837	6	18.137
32097	YEMva2023	YEM	Yemen, Rep.	2023	va	-1.550217	0.131432	8	6.372
32098	ZMBva2023	ZMB	Zambia	2023	va	-0.047946	0.118482	12	45.098
32099	ZWEva2023	ZWE	Zimbabwe	2023	va	-1.092633	0.118235	13	19.117

32100 rows × 11 columns

Entrée [71]:
```
stabpol.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32100 entries, 0 to 32099
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   codeindyr     32100 non-null  object
 1   code          32100 non-null  object
 2   countryname   32100 non-null  object
 3   year          32100 non-null  int64 
 4   indicator     32100 non-null  object
 5   estimate      32100 non-null  object
 6   stddev        32100 non-null  object
 7   nsource       32100 non-null  object
 8   pctrank       32100 non-null  object
 9   pctranklower  32100 non-null  object
 10  pctrankupper  32100 non-null  object
dtypes: int64(1), object(10)
memory usage: 2.7+ MB
```

Entrée [72]:
```
stabpol = stabpol[stabpol['indicator'] == 'pv']
stabpol = stabpol[stabpol['year'] == 2017]
stabpol
```

Out[72]:

	codeindyr	code	countryname	year	indicator	estimate	stddev	nsource	pct
23540	AFGpv2017	AFG	Afghanistan	2017	pv	-2.794976	0.225672	6	0.47
23541	ALBpv2017	ALB	Albania	2017	pv	0.373771	0.213936	8	59.52
23542	DZApv2017	DZA	Algeria	2017	pv	-0.919614	0.219212	7	15.71
23543	ASMpv2017	ASM	American Samoa	2017	pv	1.184324	0.344771	1	91.90
23544	ADOpv2017	ADO	Andorra	2017	pv	1.39289	0.309248	2	97.14
...	...	...	...	...	...	...	...	...	
23749	VIRpv2017	VIR	Virgin Islands (U.S.)	2017	pv	0.981491	0.284397	2	82.85
23750	WBGpv2017	WBG	West Bank and Gaza	2017	pv	-1.650646	0.237187	4	8.095
23751	YEMpv2017	YEM	Yemen, Rep.	2017	pv	-2.934317	0.22367	6	
23752	ZMBpv2017	ZMB	Zambia	2017	pv	0.142043	0.218073	7	52.38
23753	ZWEpv2017	ZWE	Zimbabwe	2017	pv	-0.710431	0.213936	8	20.95

214 rows × 11 columns

Entrée [73]:
```
trad = trad[['Code alpha-3','Nom en français du pays ou territoire']]
trad = trad.rename(columns={'Code alpha-3': 'code'})
trad
```

Out[73]:

	code	Nom en français du pays ou territoire
0	AFG	Afghanistan
1	ZAF	Afrique du Sud
2	ALA	Îles Åland
3	ALB	Albanie
4	DZA	Algérie
...	...	...
244	VNM	Viet Nam
245	WLF	Wallis-et-Futuna
246	YEM	Yémen
247	ZMB	Zambie
248	ZWE	Zimbabwe

249 rows × 2 columns

Entrée [74]:
```
stabpol = pd.merge(stabpol, trad, on='code', how='left')
stabpol.head()
```

Out[74]:

	codeindyr	code	countryname	year	indicator	estimate	stddev	nsource	pctrank
0	AFGpv2017	AFG	Afghanistan	2017	pv	-2.794976	0.225672	6	0.47619
1	ALBpv2017	ALB	Albania	2017	pv	0.373771	0.213936	8	59.523811
2	DZApv2017	DZA	Algeria	2017	pv	-0.919614	0.219212	7	15.714286
3	ASMpv2017	ASM	American Samoa	2017	pv	1.184324	0.344771	1	91.904762
4	ADOpv2017	ADO	Andorra	2017	pv	1.39289	0.309248	2	97.14286

Entrée [75]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(stabpol.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
codeindyr                              0
code                                   0
countryname                            0
year                                   0
indicator                              0
estimate                               0
stddev                                 0
nsource                                0
pctrank                                0
pctranklower                           0
pctrankupper                           0
Nom en français du pays ou territoire  6
dtype: int64
```

Entrée [76]:
```
lnr = stabpol[stabpol.isnull().any(axis=1)]
lnr
```

Out[76]:

	codeindyr	code	countryname	year	indicator	estimate	stddev	nsource	pctrar
4	ADOpv2017	ADO	Andorra	2017	pv	1.39289	0.309248	2	97.142
44	COGpv2017	COG	Congo, Rep.	2017	pv	-0.533615	0.222478	6	27.6190
102	KSVpv2017	KSV	Kosovo	2017	pv	-0.244493	0.237187	4	38.0952
138	ANTpv2017	ANT	Netherlands Antilles (former)	2017	pv	..	..	..	
191	TMPpv2017	TMP	Timor-Leste	2017	pv	0.066383	0.237187	4	49.0476
210	WBGpv2017	WBG	West Bank and Gaza	2017	pv	-1.650646	0.237187	4	8.0952

Entrée [77]:
```
stabpol = stabpol[['Nom en français du pays ou territoire','estimate']]
stabpol = stabpol.rename(columns={'Nom en français du pays ou territoire':
stabpol
```

Out[77]:

	Zone	Stabilité Politique
0	Afghanistan	-2.794976
1	Albanie	0.373771
2	Algérie	-0.919614
3	Samoa américaines	1.184324
4	NaN	1.39289
...	...	...
209	Îles Vierges des États-Unis	0.981491
210	NaN	-1.650646
211	Yémen	-2.934317
212	Zambie	0.142043
213	Zimbabwe	-0.710431

214 rows × 2 columns

Entrée [78]:
```
stabpol['Zone'] = stabpol['Zone'].astype(str)
```

Entrée [79]:
```
def find_best_match(row, choices, scorer=fuzz.ratio, seuil=90):
    if pd.isna(row):  # Gérer les valeurs NaN/None
        return None
    best_match = process.extractOne(str(row), choices, scorer=scorer)  # Co
    return best_match[0] if best_match and best_match[1] >= seuil else None

pop['Zone'] = pop['Zone'].apply(lambda x: find_best_match(x, stabpol['Zone'
df = df.merge(stabpol, on='Zone', how='left')
```

Entrée [80]:
```
df
```

Out[80]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
130	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
131	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
132	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
133	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
134	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

135 rows × 9 columns

Entrée [81]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
DB 2020                     3
Stabilité Politique         7
dtype: int64
```

Entrée [82]:
```
lignes_non_renseignees = df[df.isnull().any(axis=1)]
lignes_non_renseignees
```

Out[82]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	D 20
11	Bélarus	9.450231	27.98	152.0	21.0	463.0	96.8	74
21	Cabo Verde	0.537498	17.62	0.0	12.0	1.0	62.8	55
26	Chine, Taiwan Province de	23.674546	33.17	8.0	161.0	652.0	94.1	Na
29	Congo	5.110695	21.53	0.0	104.0	7.0	21.7	36
54	Guyana	0.775222	38.34	0.0	0.0	31.0	61.3	Na
73	Libéria	4.702226	10.67	0.0	48.0	15.0	34.1	43
105	République de Moldova	4.059684	16.14	0.0	16.0	54.0	90.1	74
106	République démocratique populaire lao	6.953035	10.91	0.0	0.0	0.0	39.6	50
112	Sao Tomé-et-Principe	0.207089	9.47	0.0	2.0	1.0	51.7	Na

PIB

Entrée [83]:
```
pib
```

Out[83]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	
0	Aruba	ABW	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	NaN	NaN	NaN	
1	NaN	AFE	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	186.132432	186.947182	197.408105	22
2	Afghanistan	AFG	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	NaN	NaN	NaN	
3	NaN	AFW	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	121.938353	127.452629	133.825452	13
4	Angola	AGO	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	
261	Kosovo	XKX	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	NaN	NaN	NaN	
262	Yémen, Rép. du	YEM	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	NaN	NaN	NaN	
263	Afrique du Sud	ZAF	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	532.147504	545.657512	563.423009	60
264	Zambie	ZMB	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	221.559849	209.693206	202.281031	20
265	Zimbabwe	ZWE	PIB par habitant ($ US courants)	NY.GDP.PCAP.CD	276.419762	279.016467	275.545586	27

266 rows × 68 columns

Entrée [84]:
```
pib = pib[['Country Name','2017',]]
pib = pib.rename(columns={'Country Name':'Zone','2017':'PIB'})
pib['PIB'] = pib['PIB'].round(1)
pib.head()
```

Out[84]:

	Zone	PIB
0	Aruba	28440.1
1	NaN	1520.2
2	Afghanistan	525.5
3	NaN	1560.2
4	Angola	2437.3

Entrée [85]:
```
pib['Zone'] = pib['Zone'].astype(str)
```

Entrée [86]:
```
def find_best_match(row, choices, scorer=fuzz.ratio, seuil=80):
    if pd.isna(row):  # Gérer les valeurs NaN/None
        return None
    best_match = process.extractOne(str(row), choices, scorer=scorer)  # Co
    return best_match[0] if best_match and best_match[1] >= seuil else None

pop['Zone'] = pop['Zone'].apply(lambda x: find_best_match(x, pib['Zone'].un
df = df.merge(pib, on='Zone', how='left')
```

Entrée [87]:
```
df
```

Out[87]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
130	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
131	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
132	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
133	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
134	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

135 rows × 10 columns

Entrée [88]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
DB 2020                     3
Stabilité Politique         7
PIB                         7
dtype: int64
```

Entrée [89]:
```
lignes_non_renseignees = df[df.isnull().any(axis=1)]
lignes_non_renseignees
```

Out[89]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	D 20
11	Bélarus	9.450231	27.98	152.0	21.0	463.0	96.8	74
15	Bolivie (État plurinational de)	11.192855	36.00	1.0	1.0	533.0	85.7	51
21	Cabo Verde	0.537498	17.62	0.0	12.0	1.0	62.8	55
26	Chine, Taiwan Province de	23.674546	33.17	8.0	161.0	652.0	94.1	Na
29	Congo	5.110695	21.53	0.0	104.0	7.0	21.7	36
54	Guyana	0.775222	38.34	0.0	0.0	31.0	61.3	Na
60	Iran (République islamique d')	80.673883	27.52	45.0	6.0	2174.0	91.8	58
70	Kirghizistan	6.189733	3.10	0.0	25.0	7.0	59.6	67
73	Libéria	4.702226	10.67	0.0	48.0	15.0	34.1	43
105	République de Moldova	4.059684	16.14	0.0	16.0	54.0	90.1	74
106	République démocratique populaire lao	6.953035	10.91	0.0	0.0	0.0	39.6	50
112	Sao Tomé-et-Principe	0.207089	9.47	0.0	2.0	1.0	51.7	Na
116	Slovaquie	5.447900	13.90	35.0	63.0	71.0	79.8	75

KFC

Entrée [90]:
```
kfc = kfc [['Pays','Restaurants']]
kfc = kfc.rename(columns={'Pays':'Zone','Restaurants':'KFC'})
kfc
```

Out[90]:

	Zone	KFC
0	Afrique du Sud	955
1	Albanie	8
2	Algérie	2
3	Allemagne	189
4	Angola	9
...	...	...
102	Ukraine	48
103	Viet Nam	136
104	Yémen	1
105	Zambie	4
106	Zimbabwe	6

107 rows × 2 columns

Entrée [91]:
```
def find_best_match(row, choices, scorer=fuzz.ratio, seuil=80):
    if pd.isna(row):  # Gérer les valeurs NaN/None
        return None
    best_match = process.extractOne(str(row), choices, scorer=scorer)  # Co
    return best_match[0] if best_match and best_match[1] >= seuil else None

pop['Zone'] = pop['Zone'].apply(lambda x: find_best_match(x, kfc['Zone'].un
df = df.merge(kfc, on='Zone', how='left')
```

Entrée [92]:
```
df
```

Out[92]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
130	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
131	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
132	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
133	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
134	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

135 rows × 11 columns

Entrée [93]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                         0
Population en M              0
dispo alim (kg/pers/an)      0
Exportation (mT)             0
Importation (mT)             0
Production (mT)              0
% pouvant manger             0
DB 2020                      3
Stabilité Politique          7
PIB                          7
KFC                         52
dtype: int64
```

Entrée [94]:
```
lignes_non_renseignees = df[df.isnull().any(axis=1)]
lignes_non_renseignees
```

Out[94]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	I 20
5	Argentine	43.937140	42.24	207.0	8.0	2161.0	91.4	5
11	Bélarus	9.450231	27.98	152.0	21.0	463.0	96.8	7
13	Belize	0.375769	25.69	0.0	0.0	20.0	46.8	5
14	Bénin	11.175198	14.40	0.0	123.0	18.0	15.9	5
15	Bolivie (État plurinational de)	11.192855	36.00	1.0	1.0	533.0	85.7	5
16	Bosnie-Herzégovine	3.351525	16.70	8.0	10.0	61.0	94.0	6
18	Brésil	207.833823	48.03	4223.0	3.0	14201.0	73.0	5
20	Burkina Faso	19.193234	2.27	0.0	0.0	46.0	32.8	5
21	Cabo Verde	0.537498	17.62	0.0	12.0	1.0	62.8	5
22	Cameroun	24.566073	3.16	0.0	0.0	81.0	48.7	4
24	Chili	18.470439	36.36	115.0	155.0	712.0	51.8	7
26	Chine, Taiwan Province de	23.674546	33.17	8.0	161.0	652.0	94.1	N
28	Colombie	48.909839	31.99	2.0	78.0	1564.0	68.7	7
29	Congo	5.110695	21.53	0.0	104.0	7.0	21.7	3
30	Costa Rica	4.949954	26.52	3.0	17.0	133.0	85.9	6
34	Djibouti	0.944099	2.68	0.0	3.0	0.0	36.4	6
37	Équateur	16.785361	19.31	0.0	0.0	340.0	76.8	5
42	Éthiopie	106.399924	0.13	0.0	1.0	14.0	38.9	4
43	Fédération de Russie	145.530082	30.98	115.0	226.0	4444.0	96.9	7
44	Fidji	0.877459	28.02	0.0	3.0	23.0	54.4	6
48	Gambie	2.213889	3.53	0.0	16.0	2.0	54.8	5
51	Guatemala	16.914970	20.58	7.0	129.0	235.0	54.5	6
52	Guinée	12.067519	4.08	0.0	37.0	13.0	61.9	4
53	Guinée-Bissau	1.828145	2.16	0.0	4.0	3.0	34.4	4
54	Guyana	0.775222	38.34	0.0	0.0	31.0	61.3	N
55	Haïti	10.982366	8.91	0.0	89.0	9.0	23.5	4
56	Honduras	9.429013	21.73	1.0	12.0	193.0	59.7	5
60	Iran (République islamique d')	80.673883	27.52	45.0	6.0	2174.0	91.8	5
65	Jamaïque	2.920848	51.10	1.0	31.0	128.0	82.9	6
70	Kirghizistan	6.189733	3.10	0.0	25.0	7.0	59.6	6
73	Libéria	4.702226	10.67	0.0	48.0	15.0	34.1	4
79	Mali	18.512430	2.83	0.0	1.0	48.0	41.1	5

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	I 20
82	Mauritanie	4.282570	5.14	0.0	24.0	5.0	51.0	5
85	Monténégro	0.627563	15.98	0.0	8.0	4.0	84.1	7
87	Myanmar	53.382523	30.37	0.0	3.0	1662.0	53.7	4
90	Nicaragua	6.384846	21.59	0.0	6.0	143.0	74.9	5
91	Niger	21.602382	0.94	0.0	3.0	19.0	14.8	5
93	Norvège	5.296326	19.05	0.0	2.0	101.0	98.1	8
97	Panama	4.106769	33.82	0.0	20.0	198.0	58.6	6
98	Paraguay	6.867061	5.99	4.0	1.0	45.0	78.2	5
99	Pérou	31.444298	13.47	1.0	60.0	1465.0	66.6	6
103	République centrafricaine	4.596023	2.29	0.0	4.0	7.0	8.6	3
104	République de Corée	51.096415	16.70	6.0	137.0	838.0	89.6	8
105	République de Moldova	4.059684	16.14	0.0	16.0	54.0	90.1	7
106	République démocratique populaire lao	6.953035	10.91	0.0	0.0	0.0	39.6	5
107	République dominicaine	10.513104	35.26	6.0	42.0	339.0	72.4	6
108	République-Unie de Tanzanie	54.660339	1.88	0.0	2.0	105.0	22.2	5
112	Sao Tomé-et-Principe	0.207089	9.47	0.0	2.0	1.0	51.7	N
115	Sierra Leone	7.488423	3.97	0.0	14.0	17.0	38.6	4
116	Slovaquie	5.447900	13.90	35.0	63.0	71.0	79.8	7
122	Suriname	0.570496	31.06	5.0	18.0	10.0	82.5	4
124	Tchad	15.016753	0.45	0.0	1.0	6.0	34.7	3
127	Togo	7.698474	7.06	0.0	16.0	40.0	35.1	6
128	Trinité-et-Tobago	1.384059	54.54	0.0	23.0	61.0	63.9	6
131	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	6

Entrée [95]:
```
df['KFC'] = df['KFC'].fillna(0)
df
```

Out[95]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
130	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
131	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
132	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
133	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
134	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

135 rows × 11 columns

Entrée [96]:
```
lignes_non_renseignees = df[df.isnull().any(axis=1)]
lignes_non_renseignees
```

Out[96]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	D 20
11	Bélarus	9.450231	27.98	152.0	21.0	463.0	96.8	74
15	Bolivie (État plurinational de)	11.192855	36.00	1.0	1.0	533.0	85.7	51
21	Cabo Verde	0.537498	17.62	0.0	12.0	1.0	62.8	55
26	Chine, Taiwan Province de	23.674546	33.17	8.0	161.0	652.0	94.1	Na
29	Congo	5.110695	21.53	0.0	104.0	7.0	21.7	36
54	Guyana	0.775222	38.34	0.0	0.0	31.0	61.3	Na
60	Iran (République islamique d')	80.673883	27.52	45.0	6.0	2174.0	91.8	58
70	Kirghizistan	6.189733	3.10	0.0	25.0	7.0	59.6	67
73	Libéria	4.702226	10.67	0.0	48.0	15.0	34.1	43
105	République de Moldova	4.059684	16.14	0.0	16.0	54.0	90.1	74
106	République démocratique populaire lao	6.953035	10.91	0.0	0.0	0.0	39.6	50
112	Sao Tomé-et-Principe	0.207089	9.47	0.0	2.0	1.0	51.7	Na
116	Slovaquie	5.447900	13.90	35.0	63.0	71.0	79.8	75

Entrée [97]:
```
df = df.drop(lignes_non_renseignees.index)
df
```

Out[97]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
130	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
131	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
132	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
133	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
134	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

122 rows × 11 columns

Distance avec la France

Entrée [98]:
```
dist
```

Out[98]:

	Pays	Distance km	Ville
0	Afghanistan	5,572	Kaboul
1	Afrique du Sud	8,665	Pretoria
2	Albanie	1,599	Tirana
3	Algérie	1,336	Alger
4	Allemagne	877	Berlin
...	...	...	...
205	Venezuela	7,601	Caracas
206	Viet Nam	9,178	Hanoï
207	Yémen	5,300	Sanaa
208	Zambie	7,575	Lusaka
209	Zimbabwe	7,939	Harare

210 rows × 3 columns

Entrée [99]:
```
dist = dist.rename(columns={'Pays':'Zone'})
dist = dist[['Zone', 'Distance km']]
dist['Distance km'] = dist['Distance km'].astype(str).str.replace(',', '').
dist['Distance km'] = pd.to_numeric(dist['Distance km'])
dist
```

Out[99]:

	Zone	Distance km
0	Afghanistan	5572.0
1	Afrique du Sud	8665.0
2	Albanie	1599.0
3	Algérie	1336.0
4	Allemagne	877.0
...	...	...
205	Venezuela	7601.0
206	Viet Nam	9178.0
207	Yémen	5300.0
208	Zambie	7575.0
209	Zimbabwe	7939.0

210 rows × 2 columns

Entrée [100]:
```
dist['Zone'] = dist['Zone'].str.strip()
```

Entrée [101]:
```
def find_best_match(row, choices, scorer=fuzz.ratio, seuil=80):
    if pd.isna(row):  # Gérer les valeurs NaN/None
        return None
    best_match = process.extractOne(str(row), choices, scorer=scorer)  # C
    return best_match[0] if best_match and best_match[1] >= seuil else Non

pop['Zone'] = pop['Zone'].apply(lambda x: find_best_match(x, dist['Zone'].
df = df.merge(dist, on='Zone', how='left')
```

Entrée [102]:
```
df
```

Out[102]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
117	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
118	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
119	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
120	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
121	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

122 rows × 12 columns

Entrée [103]:
```
print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
DB 2020                     0
Stabilité Politique         0
PIB                         0
KFC                         0
Distance km                 2
dtype: int64
```

Entrée [104]:
```
lignes_non_renseignees = df[df.isnull().any(axis=1)]
lignes_non_renseignees
```

Out[104]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020	S P
35	Eswatini	1.124805	6.46	0.0	2.0	6.0	28.9	59.5	-0
41	France	64.842509	22.90	501.0	506.0	1750.0	98.1	76.8	0

Entrée [105]:
```
df = df.drop(lignes_non_renseignees.index)
df
```

Out[105]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
117	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
118	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
119	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
120	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
121	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

120 rows × 12 columns

Entrée [106]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 120 entries, 0 to 121
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Zone                     120 non-null    object
 1   Population en M          120 non-null    float64
 2   dispo alim (kg/pers/an)  120 non-null    float64
 3   Exportation (mT)         120 non-null    float64
 4   Importation (mT)         120 non-null    float64
 5   Production (mT)          120 non-null    float64
 6   % pouvant manger         120 non-null    float64
 7   DB 2020                  120 non-null    float64
 8   Stabilité Politique      120 non-null    object
 9   PIB                      120 non-null    float64
 10  KFC                      120 non-null    float64
 11  Distance km              120 non-null    float64
dtypes: float64(10), object(2)
memory usage: 12.2+ KB
```

Entrée [107]:
```
df
```

Out[107]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
117	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
118	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
119	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
120	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
121	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

120 rows × 12 columns

Entrée [108]:
```
colonnes_a_convertir = df.columns.drop('Zone')

for colonne in colonnes_a_convertir:
    df[colonne] = pd.to_numeric(df[colonne], errors='coerce')

print("Types des colonnes après conversion:")
print(df.dtypes)

print("\nNombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Types des colonnes après conversion:
Zone                         object
Population en M             float64
dispo alim (kg/pers/an)     float64
Exportation (mT)            float64
Importation (mT)            float64
Production (mT)             float64
% pouvant manger            float64
DB 2020                     float64
Stabilité Politique         float64
PIB                         float64
KFC                         float64
Distance km                 float64
dtype: object

Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
DB 2020                     0
Stabilité Politique         0
PIB                         0
KFC                         0
Distance km                 0
dtype: int64
```

Entrée [109]:
```
print("Nombre de valeurs manquantes par colonne:")
print(df.isnull().sum())

#Les valeurs manquantes par la moyenne de chaque colonne
#On ne traite pas la colonne 'Zone' car c'est une colonne catégorielle
colonnes_numeriques = df.columns.drop('Zone')
for colonne in colonnes_numeriques:
    df[colonne] = df[colonne].fillna(df[colonne].mean())

#Vérification qu'il n'y a plus de valeurs manquantes
print("\nAprès traitement - Nombre de valeurs manquantes par colonne:")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
DB 2020                     0
Stabilité Politique         0
PIB                         0
KFC                         0
Distance km                 0
dtype: int64

Après traitement - Nombre de valeurs manquantes par colonne:
Zone                        0
Population en M             0
dispo alim (kg/pers/an)     0
Exportation (mT)            0
Importation (mT)            0
Production (mT)             0
% pouvant manger            0
DB 2020                     0
Stabilité Politique         0
PIB                         0
KFC                         0
Distance km                 0
dtype: int64
```

Entrée [110]:
```
X = df.select_dtypes(include=[np.number])
pays = df['Zone']
```

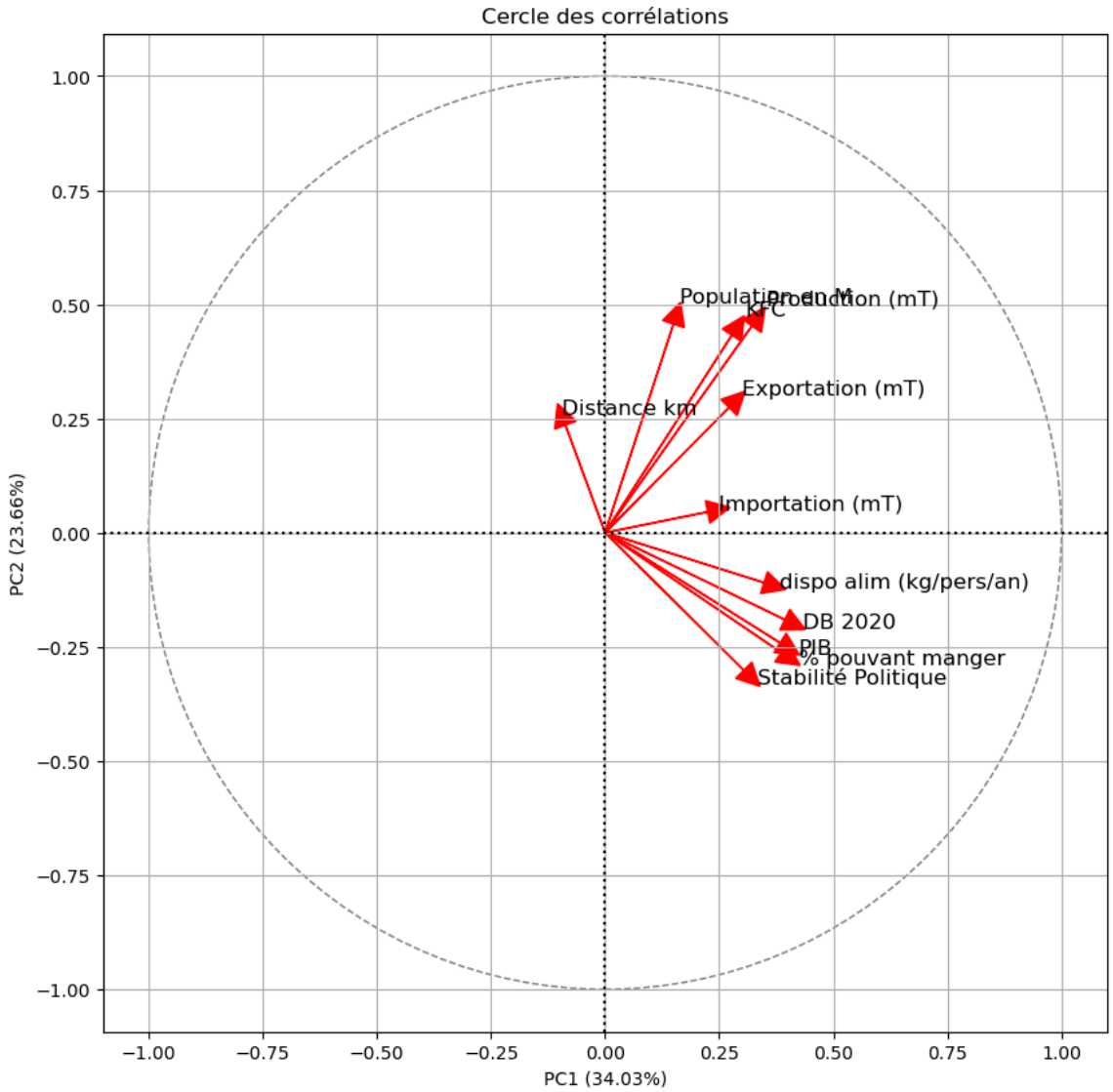
Entrée [111]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

Entrée [112]:
```
n_components = 4
pca = PCA(n_components=n_components)
pca_result = pca.fit_transform(X_scaled)
```

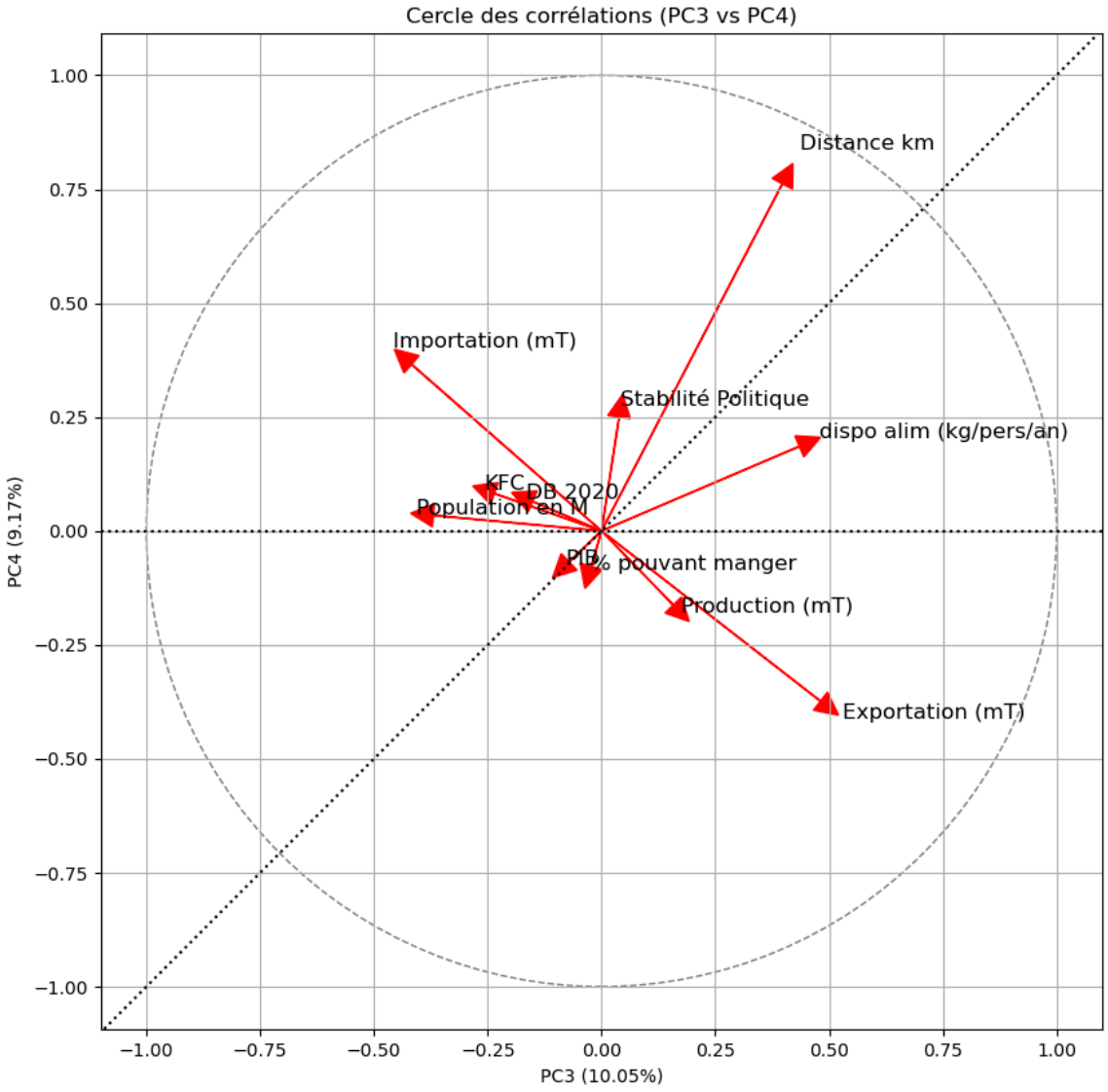
Entrée [113]:
```
print("Variance expliquée par composante:")
print(pca.explained_variance_ratio_)
print("\nVariance expliquée cumulative:")
print(np.cumsum(pca.explained_variance_ratio_))
```

```
Variance expliquée par composante:
[0.34033497 0.23655195 0.10054207 0.09169691]

Variance expliquée cumulative:
[0.34033497 0.57688692 0.67742899 0.7691259 ]
```

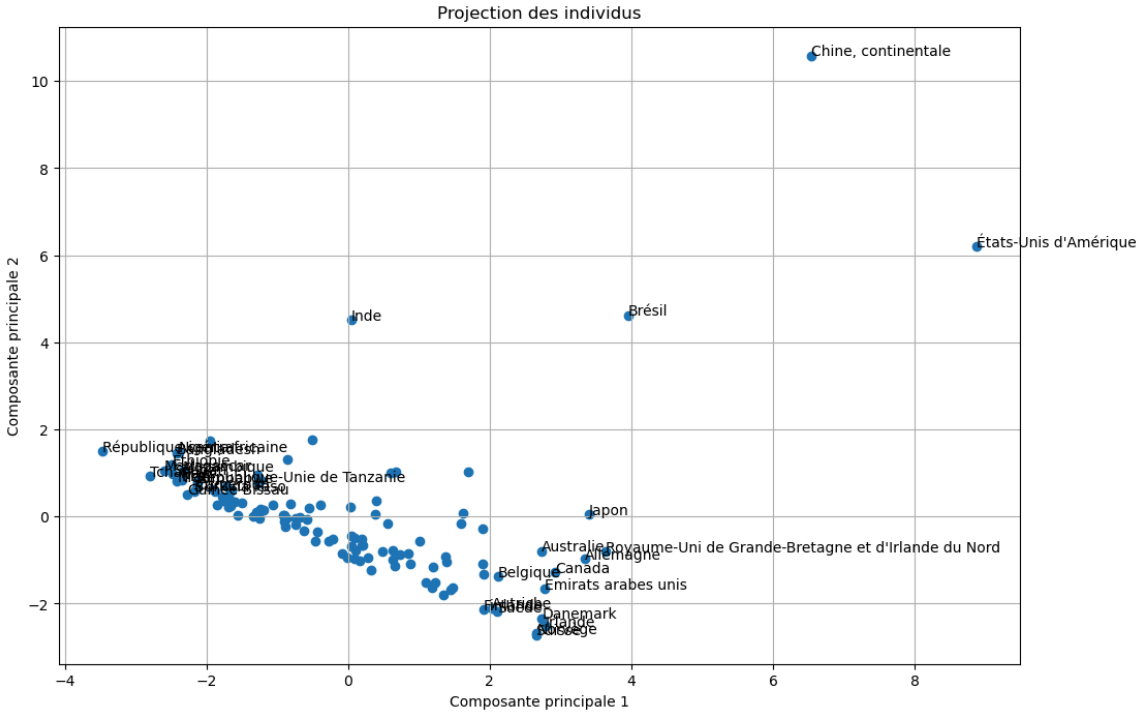
Entrée [114]:
```
plt.figure(figsize=(10, 10))
circle = plt.Circle((0,0), radius=1, fill=False, color='gray', ls='--')
plt.axhline(y=0, color='k', ls=':')
plt.axvline(x=0, color='k', ls=':')
plt.gca().add_patch(circle)

for i, (x, y) in enumerate(zip(pca.components_[0], pca.components_[1])):
    plt.arrow(0, 0, x, y, head_width=0.05, head_length=0.05, fc='r', ec='r
    plt.text(x*1.1, y*1.1, X_scaled.columns[i], fontsize=12)

plt.grid(True)
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.2%})')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.2%})')
plt.title('Cercle des corrélations')
plt.axis('equal')
plt.show()
```



Entrée [115]:
```
plt.figure(figsize=(10, 10))
circle = plt.Circle((0, 0), radius=1, fill=False, color='gray', ls='--')
plt.gca().add_patch(circle)
plt.axhline(y=0, color='k', ls=':')
plt.axline((0, 0), slope=1, color='k', ls=':')

for i, (x, y) in enumerate(zip(pca.components_[2], pca.components_[3])):  #
    plt.arrow(0, 0, x, y, head_width=0.05, head_length=0.05, fc='r', ec='r
    plt.text(x*1.1, y*1.1, X_scaled.columns[i], fontsize=12)

plt.grid(True)
plt.xlabel(f'PC3 ({pca.explained_variance_ratio_[2]:.2%})')
plt.ylabel(f'PC4 ({pca.explained_variance_ratio_[3]:.2%})')
plt.title('Cercle des corrélations (PC3 vs PC4)')
plt.axis('equal')
plt.show()
```

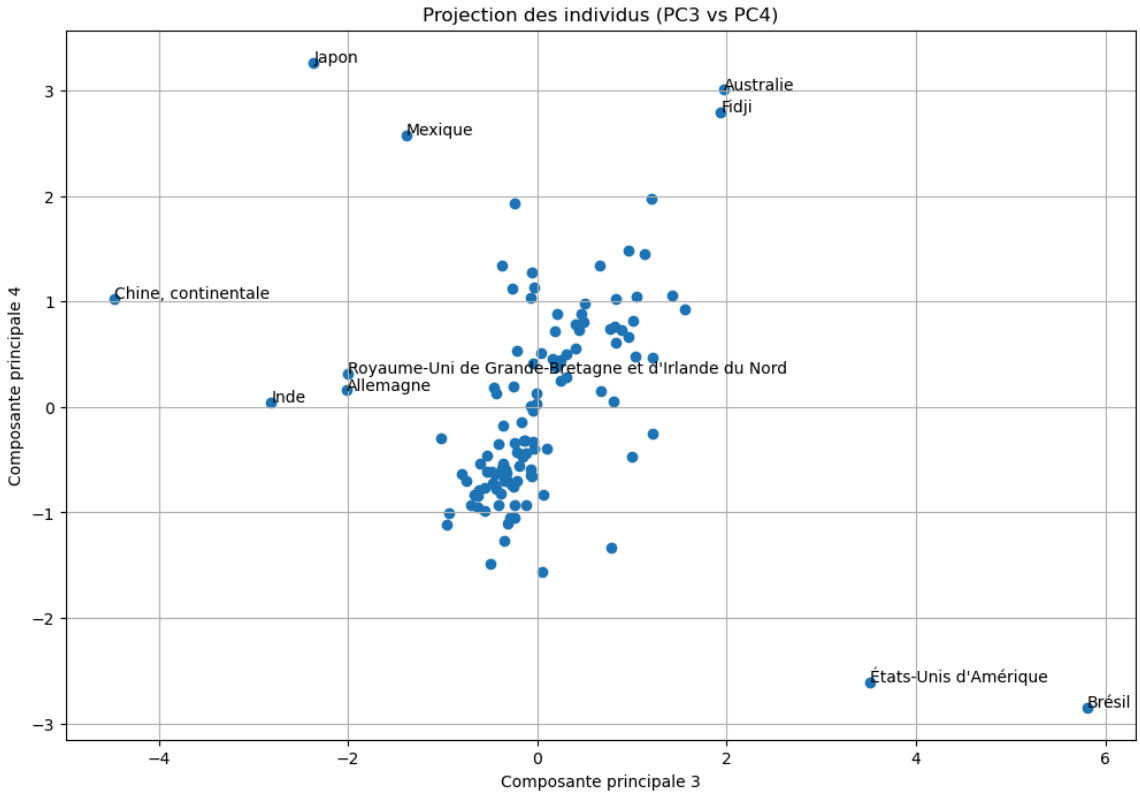


Entrée [116]:
```
plt.figure(figsize=(12, 8))
plt.scatter(pca_result[:, 0], pca_result[:, 1])
plt.xlabel(f'Composante principale 1')
plt.ylabel(f'Composante principale 2')
plt.title('Projection des individus')
for i, txt in enumerate(pays):

    if abs(pca_result[i, 0]) > 2 or abs(pca_result[i, 1]) > 2:
        plt.annotate(txt, (pca_result[i, 0], pca_result[i, 1]))
plt.grid(True)
plt.show()
```

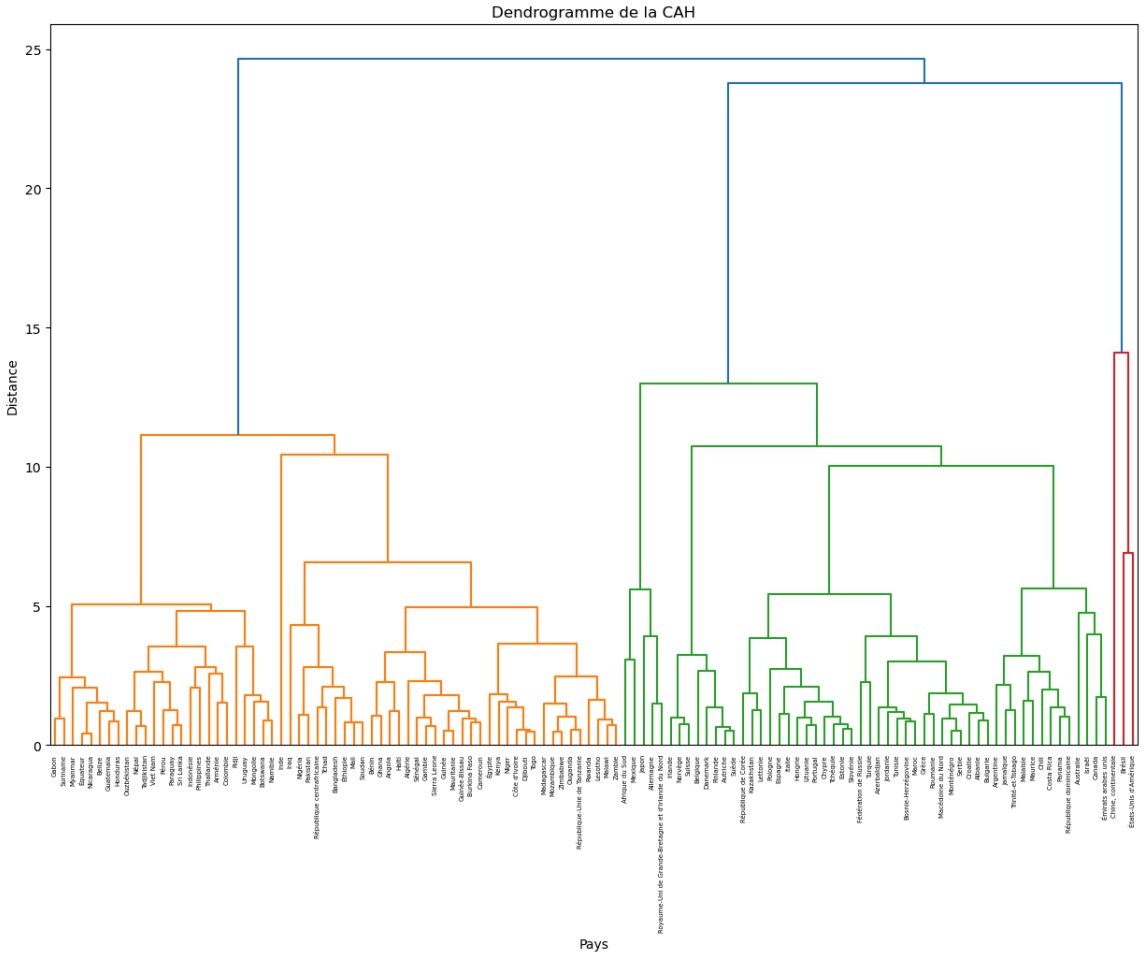


Entrée [117]:

```
plt.figure(figsize=(12, 8))
plt.scatter(pca_result[:, 2], pca_result[:, 3])
plt.xlabel(f'Composante principale 3')
plt.ylabel(f'Composante principale 4')
plt.title('Projection des individus (PC3 vs PC4)')

for i, txt in enumerate(pays):
    if abs(pca_result[i, 2]) > 2 or abs(pca_result[i, 3]) > 2:
        plt.annotate(txt, (pca_result[i, 2], pca_result[i, 3]))

plt.grid(True)
plt.show()
```

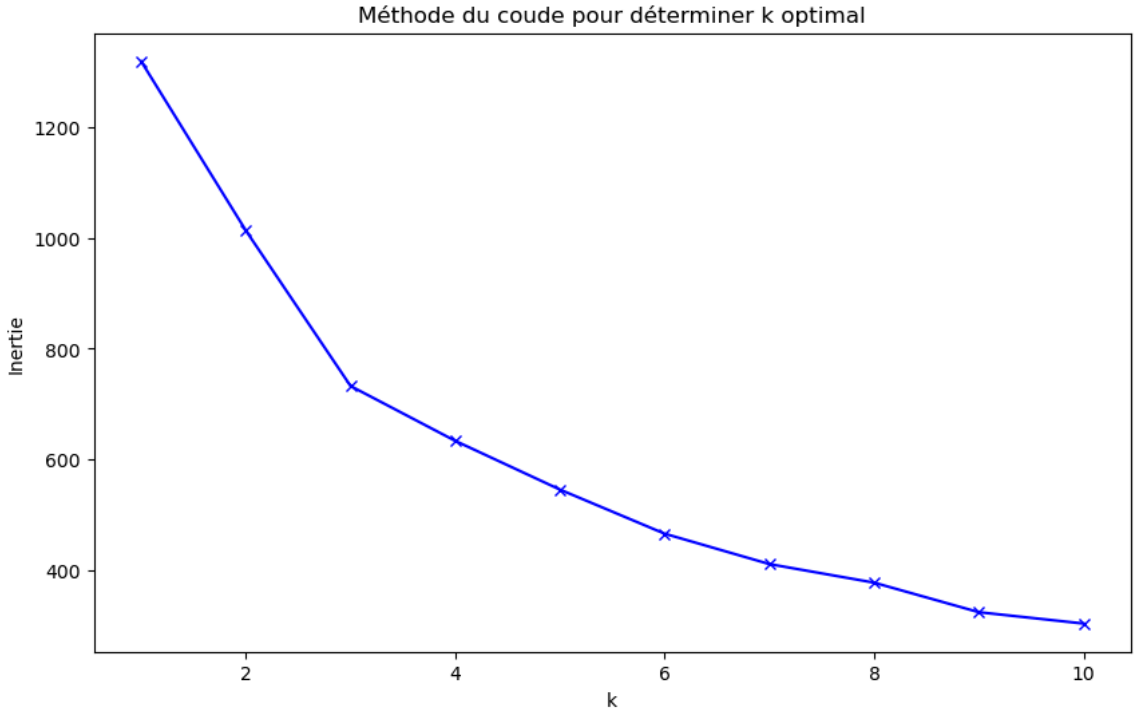


Entrée [118]:
```
linkage_matrix = linkage(X_scaled, method='ward')

plt.figure(figsize=(15, 10))
dendrogram(linkage_matrix, labels=pays.values, leaf_rotation=90)
plt.title('Dendrogramme de la CAH')
plt.xlabel('Pays')
plt.ylabel('Distance')
plt.show()
```



Entrée [119]:
```
inertias = []
K = range(1, 11)
for k in K:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Visualisation de la méthode du coude
plt.figure(figsize=(10, 6))
plt.plot(K, inertias, 'bx-')
plt.xlabel('k')
plt.ylabel('Inertie')
plt.title('Méthode du coude pour déterminer k optimal')
plt.show()
```

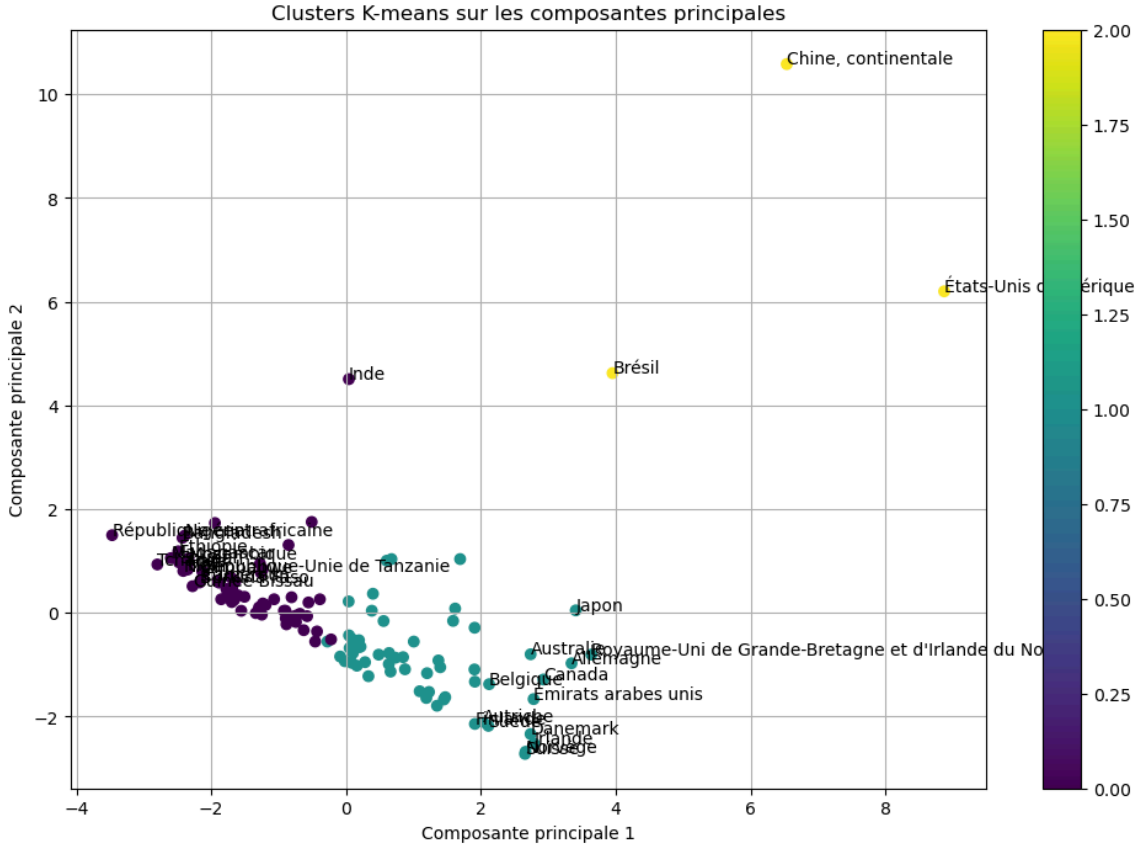
```
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



Entrée [120]:
```
k_optimal = 3  #méthode du coude
kmeans = KMeans(n_clusters=k_optimal, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
```

```
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:141
2: FutureWarning: The default value of `n_init` will change from 10 to 'au
to' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\maxen\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:143
6: 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 set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Entrée [121]:
```
plt.figure(figsize=(12, 8))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1], c=clusters, cmap=
plt.xlabel('Composante principale 1')
plt.ylabel('Composante principale 2')
plt.title('Clusters K-means sur les composantes principales')
plt.colorbar(scatter)
for i, txt in enumerate(pays):
    if abs(pca_result[i, 0]) > 2 or abs(pca_result[i, 1]) > 2:
        plt.annotate(txt, (pca_result[i, 0], pca_result[i, 1]))
plt.grid(True)
plt.show()
```



Entrée [122]:
```
resultats_clusters = pd.DataFrame({
    'Pays': pays,
    'Cluster': clusters
})
print("\nComposition des clusters:")
print(resultats_clusters.groupby('Cluster')['Pays'].apply(list))
```

```
Composition des clusters:
Cluster
0    [Algérie, Angola, Arménie, Bangladesh, Belize,...
1    [Afrique du Sud, Albanie, Allemagne, Argentine...
2    [Brésil, Chine, continentale, États-Unis d'Amé...
Name: Pays, dtype: object
```

Entrée [123]:
```
df['Cluster'] = clusters
df
```

Out[123]:

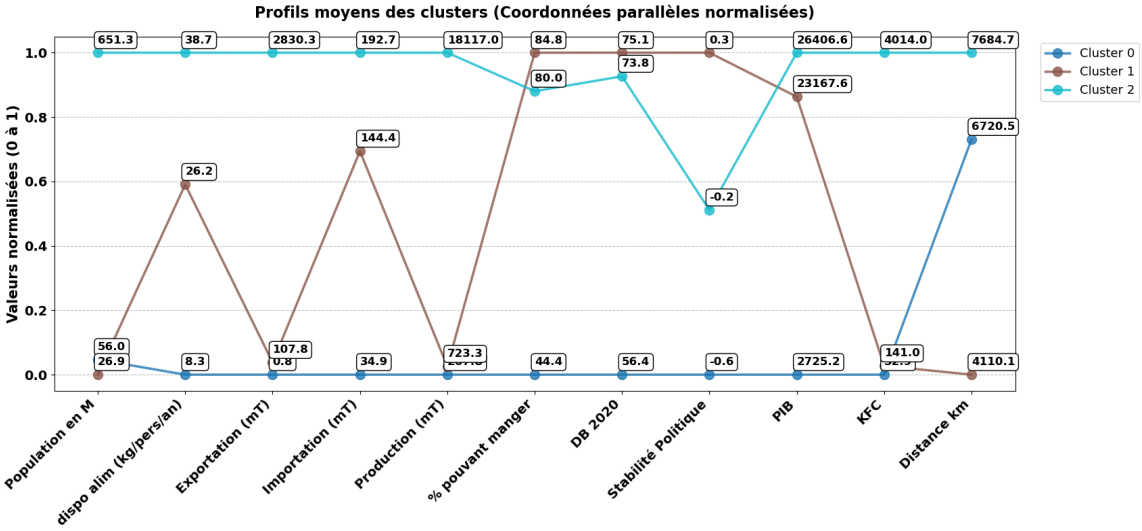
	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Algérie	41.389189	6.38	0.0	2.0	275.0	81.4	48.6
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
4	Angola	29.816766	10.56	0.0	277.0	42.0	36.3	41.3
...	...	...	...	...	...	...	...	...
117	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
118	Uruguay	3.436641	9.12	3.0	3.0	33.0	68.0	61.5
119	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8
120	Zambie	16.853599	3.42	1.0	12.0	49.0	19.3	66.9
121	Zimbabwe	14.236595	4.68	0.0	6.0	69.0	22.0	54.5

120 rows × 13 columns

Entrée [124]:
```
def create_parallel_coordinates(df):
    """
    Crée un graphique en coordonnées parallèles avec normalisation par vari
    et affichage des valeurs en grand format.
    """
    # Sélection des variables
    variables = [
        'Population en M', 'dispo alim (kg/pers/an)', 'Exportation (mT)',
        'Importation (mT)', 'Production (mT)', '% pouvant manger',
        'DB 2020', 'Stabilité Politique', 'PIB', 'KFC', 'Distance km'
    ]

    # Calcul des moyennes par cluster
    cluster_means = df.groupby('Cluster')[variables].mean()

    # Normalisation par variable (Min-Max Scaling sur chaque colonne indépe
    normalized_data = cluster_means.apply(lambda x: (x - x.min()) / (x.max

    # Création du graphique
    plt.figure(figsize=(18, 8))

    # Positions des axes verticaux
    x = np.arange(len(variables))

    # Génération des couleurs par cluster
    num_clusters = len(cluster_means.index)
    colors = plt.cm.get_cmap("tab10", num_clusters)

    # Tracer les lignes pour chaque cluster
    for idx, cluster in enumerate(cluster_means.index):
        values = normalized_data.loc[cluster].values
        plt.plot(x, values, '-o',
                 color=colors(idx),
                 label=f'Cluster {cluster}',
                 linewidth=2.5, markersize=10, alpha=0.8)

    # Personnalisation des axes et labels
    plt.xticks(x, variables, rotation=45, ha='right', fontsize=14, fontweig
    plt.yticks(fontsize=14, fontweight='bold')
    plt.grid(axis='y', linestyle='--', alpha=0.7)

    # Affichage des valeurs réelles à côté des points avec une taille plus
    for idx, cluster in enumerate(cluster_means.index):
        real_values = cluster_means.loc[cluster].values
        norm_values = normalized_data.loc[cluster].values

        for j, (norm_v, real_v) in enumerate(zip(norm_values, real_values)
            plt.annotate(f'{real_v:.1f}',
                         (x[j], norm_v),
                         xytext=(0, 10), textcoords='offset points',
                         fontsize=12, fontweight='bold', color='black',
                         bbox=dict(facecolor='white', edgecolor='black', bo

    # Personnalisation supplémentaire
    plt.ylabel('Valeurs normalisées (0 à 1)', fontsize=15, fontweight='bold
    plt.title('Profils moyens des clusters (Coordonnées parallèles normalis
    plt.legend(bbox_to_anchor=(1.02, 1), loc='upper left', fontsize=13)

    # Ajustement des marges pour éviter le chevauchement
    plt.tight_layout(rect=[0, 0, 0.95, 1])
```

```
    return plt.gcf()

# Création et affichage du graphique
fig = create_parallel_coordinates(df)
plt.show()
```

```
C:\Users\maxen\AppData\Local\Temp\ipykernel_12160\2899308841.py:27: Matplo
tlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib
3.7 and will be removed two minor releases later. Use ``matplotlib.colorma
ps[name]`` or ``matplotlib.colormaps.get_cmap(obj)`` instead.
  colors = plt.cm.get_cmap("tab10", num_clusters)
```



Entrée [125]:
```
pays_cluster = df[df['Cluster'] == 1].reset_index(drop=True)
pays_cluster
```

Out[125]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DE 2020
0	Afrique du Sud	57.009756	35.69	63.0	514.0	1667.0	38.3	67.0
1	Albanie	2.884169	16.36	0.0	38.0	13.0	75.7	67.7
2	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
3	Argentine	43.937140	42.24	207.0	8.0	2161.0	91.4	59.0
4	Australie	24.584620	47.65	42.0	16.0	1269.0	97.2	81.2
5	Autriche	8.819901	18.20	78.0	110.0	148.0	97.7	78.7
6	Azerbaïdjan	9.845320	13.11	0.0	27.0	104.0	99.0	76.7
7	Belgique	11.419748	12.65	656.0	338.0	463.0	98.2	75.0
8	Bosnie-Herzégovine	3.351525	16.70	8.0	10.0	61.0	94.0	65.4
9	Bulgarie	7.102444	22.06	45.0	108.0	107.0	90.1	72.0
10	Canada	36.732095	39.02	163.0	182.0	1417.0	96.7	79.6
11	Chili	18.470439	36.36	115.0	155.0	712.0	51.8	72.6
12	Chypre	1.179678	25.08	1.0	14.0	25.0	91.5	73.4
13	Colombie	48.909839	31.99	2.0	78.0	1564.0	68.7	70.1
14	Costa Rica	4.949954	26.52	3.0	17.0	133.0	85.9	69.2
15	Croatie	4.182857	10.77	17.0	24.0	61.0	73.7	73.6
16	Danemark	5.732274	28.98	139.0	133.0	173.0	98.3	85.3
17	Émirats arabes unis	9.487203	43.47	94.0	433.0	48.0	96.8	80.9
18	Espagne	46.647428	30.39	212.0	205.0	1515.0	89.1	77.9
19	Estonie	1.319390	21.26	11.0	21.0	20.0	92.4	80.6
20	Fédération de Russie	145.530082	30.98	115.0	226.0	4444.0	96.9	78.2
21	Finlande	5.511371	18.76	12.0	16.0	129.0	98.2	80.2
22	Grèce	10.569450	15.32	29.0	79.0	246.0	68.8	68.4
23	Hongrie	9.729823	25.27	210.0	58.0	493.0	68.1	73.4
24	Irlande	4.753279	25.82	93.0	99.0	110.0	97.9	79.6
25	Israël	8.243848	67.39	3.0	0.0	629.0	80.6	76.7
26	Italie	60.673701	18.88	183.0	97.0	1315.0	89.8	72.9
27	Jamaïque	2.920848	51.10	1.0	31.0	128.0	82.9	69.7
28	Japon	127.502725	18.50	10.0	1069.0	2215.0	92.9	78.0
29	Jordanie	9.785843	28.07	10.0	64.0	210.0	86.7	69.0
30	Kazakhstan	18.080019	18.27	6.0	174.0	171.0	90.6	79.6
31	Lettonie	1.951097	20.89	20.0	43.0	33.0	74.4	80.3
32	Lituanie	2.845414	28.19	68.0	44.0	118.0	78.9	81.6
33	Macédoine du Nord	2.081996	19.73	1.0	40.0	2.0	76.0	80.7

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DE 2020
34	Malaisie	31.104646	39.21	44.0	68.0	1724.0	95.8	81.5
35	Maroc	35.581255	20.96	1.0	3.0	762.0	87.1	73.4
36	Maurice	1.264499	37.93	0.0	2.0	48.0	84.2	81.5
37	Mexique	124.777324	32.52	9.0	972.0	3249.0	72.0	72.4
38	Monténégro	0.627563	15.98	0.0	8.0	4.0	84.1	73.8
39	Norvège	5.296326	19.05	0.0	2.0	101.0	98.1	82.6
40	Panama	4.106769	33.82	0.0	20.0	198.0	58.6	66.6
41	Pologne	37.953180	30.30	1025.0	55.0	2351.0	82.6	76.4
42	Portugal	10.288527	30.58	44.0	82.0	361.0	81.5	76.5
43	République de Corée	51.096415	16.70	6.0	137.0	838.0	89.6	84.0
44	République dominicaine	10.513104	35.26	6.0	42.0	339.0	72.4	60.0
45	Roumanie	19.653969	19.37	69.0	146.0	392.0	47.6	73.3
46	Royaume-Uni de Grande-Bretagne et d'Irlande du...	66.727461	31.94	359.0	779.0	1814.0	96.7	83.5
47	Serbie	8.829628	10.16	7.0	12.0	85.0	80.7	75.7
48	Slovénie	2.076394	24.41	29.0	20.0	72.0	95.2	76.5
49	Suède	9.904896	16.60	23.0	84.0	157.0	96.0	82.0
50	Suisse	8.455804	15.72	4.0	51.0	91.0	98.8	76.6
51	Tchéquie	10.641034	21.96	27.0	116.0	163.0	95.3	76.3
52	Thaïlande	69.209810	12.95	796.0	2.0	1676.0	78.3	80.1
53	Trinité-et-Tobago	1.384059	54.54	0.0	23.0	61.0	63.9	61.3
54	Tunisie	11.433443	17.03	4.0	0.0	213.0	92.1	68.7
55	Turquie	81.116450	20.64	429.0	3.0	2192.0	88.0	76.8
56	Viet Nam	94.600648	12.33	1.0	291.0	918.0	88.8	69.8

Entrée [126]:
```
pays_cluster = df[
    (df['Stabilité Politique'] > 0) &
    (df['Distance km'] < 1500) &
    (df['% pouvant manger'] > 90) &
    (df['Cluster'] == 1)
]
pays_cluster
```

Out[126]:

	Zone	Population en M	dispo alim (kg/pers/an)	Exportation (mT)	Importation (mT)	Production (mT)	% pouvant manger	DB 2020
3	Allemagne	82.658409	19.47	646.0	842.0	1514.0	97.2	79.7
8	Autriche	8.819901	18.20	78.0	110.0	148.0	97.7	78.7
11	Belgique	11.419748	12.65	656.0	338.0	463.0	98.2	75.0
28	Danemark	5.732274	28.98	139.0	133.0	173.0	98.3	85.3
55	Irlande	4.753279	25.82	93.0	99.0	110.0	97.9	79.6
84	Norvège	5.296326	19.05	0.0	2.0	101.0	98.1	82.6
99	Royaume-Uni de Grande-Bretagne et d'Irlande du...	66.727461	31.94	359.0	779.0	1814.0	96.7	83.5
104	Slovénie	2.076394	24.41	29.0	20.0	72.0	95.2	76.5
108	Suisse	8.455804	15.72	4.0	51.0	91.0	98.8	76.6
112	Tchéquie	10.641034	21.96	27.0	116.0	163.0	95.3	76.3