CODE PYTHON :

Séance 2 :

```
#coding:utf8

import pandas as pd
import matplotlib.pyplot as plt

# Source des données :
https://www.data.gouv.fr/datasets/election-presidentielle-des-10-et-24-avril-2022-resu
ltats-definitifs-du-1er-tour/
with open("./data/resultats-elections-presidentielles-2022-1er-tour.csv","r") as fichier:
    contenu = pd.read_csv(fichier)

print(pd.DataFrame(contenu))

nombre_lignes = len(contenu)
nombre_colonnes = len(contenu.columns)

print("Nombre de lignes : ", nombre_lignes)
print("Nombre de colonnes : ", nombre_colonnes)

type_map = {
    'int64': 'int',
    'float64': 'float',
    'bool': 'bool',
    'object': 'str',
    'string': 'str'
}

for col in contenu.columns:
    dtype = str(contenu[col].dtype)
    type_logique = type_map.get(dtype, 'str')

print(contenu.head(0))

inscrits = contenu.loc[0, "Inscrits"]

colonnes_quantitatives = []

for col in contenu.columns:
    dtype = str(contenu[col].dtype)
    type_logique = type_map.get(dtype, "str")
    if type_logique in ["int", "float"]:
        colonnes_quantitatives.append(col)


# Affichage
for col in colonnes_quantitatives:
```

```
    print(f"{col}: {contenu[col].sum()}")

# Boucle sur les départements uniques
for dept in contenu["Code du département"].unique():
    # Filtrer les données du département
    df_dept = contenu[contenu["Code du département"] == dept]

    # Somme des inscrits et votants dans ce département
    inscrits = df_dept["Inscrits"].sum()
    votants = df_dept["Votants"].sum()

    # Créer le graphique
    plt.figure(figsize=(6, 4))
    plt.bar(["Inscrits", "Votants"], [inscrits, votants], color=["skyblue", "salmon"])
    plt.title(f"Département {dept} - Inscrits vs Votants")
    plt.ylabel("Nombre de personnes")

    # Sauvegarder dans un fichier image
    filename = f"diagrammes/departement_{dept}.png"
    plt.savefig(filename)
    plt.close()

# Boucle sur chaque département
for dept in contenu["Code du département"].unique():
    # Filtrer les données du département
    df_dept = contenu[contenu["Code du département"] == dept]
    valeurs = [
        df_dept["Abstentions"].sum(),
        df_dept["Blancs"].sum(),
        df_dept["Nuls"].sum(),
        df_dept["Exprimés"].sum()
        ]
    labels = ["Abstentions", "Blancs", "Nuls", "Exprimés"]
    valeurs_filtrees = [v for v in valeurs if v > 0]
    labels_filtres = [labels[i] for i in range(len(labels)) if valeurs[i] > 0]
    # Créer le graphique
    plt.figure(figsize=(6, 6))
    plt.pie(valeurs_filtrees, labels=labels_filtres, autopct='%1.1f%%', startangle=90)
    plt.title(f"Département {dept} - Répartition des votes")

    # Sauvegarder l'image
    filename = f"diagrammes_camembert/camembert_{dept}.png"
    plt.savefig(filename)
    plt.close()

plt.figure(figsize=(10, 6))
plt.hist(contenu["Inscrits"], bins=30, color="skyblue", edgecolor="black")
plt.title("Distribution des inscrits")
plt.xlabel("Nombre d'inscrits")
plt.ylabel("Fréquence")
```

```
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Enregistrer l'image dans un fichier PNG
plt.tight_layout()
plt.savefig("distribution_inscrits.png")
plt.close()

#print()
```

Séance 3 :

```
#coding:utf8

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Source des données :
https://www.data.gouv.fr/datasets/election-presidentielle-des-10-et-24-avril-2022-resu
ltats-definitifs-du-1er-tour/

# Sources des données : production de M. Forriez, 2016-2023

with open("data/resultats-elections-presidentielles-2022-1er-tour.csv",
encoding='utf-8') as fichier:
    df = pd.read_csv(fichier)


type_map = {
    'int64': 'int',
    'float64': 'float',
    'bool': 'bool',
    'object': 'str',
    'string': 'str'
}

colonnes_quantitatives = []

for col in df.columns:
    dtype = str(df[col].dtype)
    type_logique = type_map.get(dtype, "str")
    if type_logique in ["int", "float"]:
        colonnes_quantitatives.append(col)

liste_moyenne = []

# Affichage
for col in colonnes_quantitatives:
    moyenne = round(df[col].mean(), 2)
```

```
    liste_moyenne.append(f"{col}: {moyenne}")
print(liste_moyenne)

liste_median = []

for col in colonnes_quantitatives:
    median = df[col].median()
    liste_median.append(f"{col}: {median}")
print(liste_median)

liste_mode = []

for col in colonnes_quantitatives:
    mode_val= df[col].mode().tolist()
    liste_mode.append(f"{col}: {mode_val}")
print(liste_mode)

liste_ecart_type = []

for col in colonnes_quantitatives:
    ecart_type = round(df[col].std(), 2)
    liste_ecart_type.append(f"{col}: {ecart_type}")
print(liste_ecart_type)

liste_ecart_absolu = []

for col in colonnes_quantitatives:
    moyenne = df[col].mean()
    ecart_absolu = round(np.abs(df[col] - moyenne).mean(), 2)
    liste_ecart_absolu.append(f"{col}: {ecart_absolu}")
print(liste_ecart_absolu)

liste_etendue = []

for col in colonnes_quantitatives:
    etendue = round(df[col].max() - df[col].min(), 2)
    liste_etendue.append(f"{col}: {etendue}")
print(liste_etendue)

liste_interquartile = []

for col in colonnes_quantitatives:
    q3 = df[col].quantile(0.75)
    q1 = df[col].quantile(0.25)
    iqr = round(q3 - q1, 2)
    liste_interquartile.append(f"{col}: {iqr}")

liste_interdecile = []

for col in colonnes_quantitatives:
```

```
    d9 = df[col].quantile(0.9)
    d1 = df[col].quantile(0.1)
    idr = round(d9 - d1, 2)
    liste_interdecile.append(f"{col}: {idr}")

#for col in colonnes_quantitatives:
#    plt.figure(figsize=(6, 4))
#    plt.boxplot(df[col], vert=True, patch_artist=True)
#    plt.title(f'Boîte à moustaches - {col}')
#    plt.ylabel(col)
#    plt.savefig(f'img/boxplot_{col}.png', dpi=150, bbox_inches='tight')
#    plt.close()

with open("data/island-index.csv", encoding="utf-8") as fichier:
    df = pd.read_csv(fichier)

# Sélection et catégorisation de la colonne "Surface (km2)"

liste_surface = []

for surface in df["Surface (km²)"]:
    if 0 < surface <= 10:
        categorie = "]0, 10]"
    elif 10 < surface <= 25:
        categorie = "]10, 25]"
    elif 25 < surface <= 50:
        categorie = "]25, 50]"
    elif 50 < surface <= 100:
        categorie = "]50, 100]"
    elif 100 < surface <= 2500:
        categorie = "]100, 2500]"
    elif 2500 < surface <= 5000:
        categorie = "]2500, 5000]"
    elif 5000 < surface <= 10000:
        categorie = "]5000, 10000]"
    else:  # surface > 10000
        categorie = "]10000, +∞["

    liste_surface.append(categorie)

# Ajouter la liste comme nouvelle colonne dans le DataFrame
df["Classe_Surface"] = liste_surface

# Dénombrement des îles par catégorie
liste_comptage = df["Classe_Surface"].value_counts().sort_index()

plt.figure(figsize=(8, 5))
plt.bar(liste_comptage.index, liste_comptage.values, color='skyblue',
edgecolor='black')
```

```
plt.title("Répartition des îles par intervalle de surface")
plt.xlabel("Intervalle de surface (km²)")
plt.ylabel("Nombre d'îles")
plt.xticks(rotation=45, ha='right')

# Enregistrement de l'organigramme dans le dossier img
plt.tight_layout()
plt.savefig("img/organigramme_surface.png", dpi=150)
plt.close()

print("=== Nombre d'îles par intervalle de surface ===")
for categorie, nombre in liste_comptage.items():
    print(f"{categorie}: {nombre}")
```

Séance 4 :

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import (
    randint, binom, poisson, zipf, norm, lognorm,
    uniform, chi2, pareto
)

# --- Lois discrètes ---

# 1. Loi de Dirac
x = np.linspace(-5, 5, 100)
y = np.zeros_like(x)
y[np.argmin(np.abs(x - 0))] = 1
plt.figure()
plt.stem(x, y)
plt.title("Loi de Dirac (x0 = 0)")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 2. Loi uniforme discrète
a, b = 1, 6
x = np.arange(a, b+1)
pmf = randint.pmf(x, a, b+1)
plt.figure()
plt.stem(x, pmf)
plt.title(f"Loi uniforme discrète U({a},{b})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 3. Loi binomiale
n, p = 10, 0.5
x = np.arange(0, n+1)
pmf = binom.pmf(x, n, p)
```

```
plt.figure()
plt.stem(x, pmf)
plt.title(f"Loi binomiale B({n},{p})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 4️⃣ Loi de Poisson (discrète)
lam = 3
x = np.arange(0, 15)
pmf = poisson.pmf(x, lam)
plt.figure()
plt.stem(x, pmf)
plt.title(f"Loi de Poisson (λ={lam})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 5️⃣ Loi de Zipf-Mandelbrot
a = 2
x = np.arange(1, 10)
pmf = zipf.pmf(x, a)
plt.figure()
plt.stem(x, pmf)
plt.title(f"Loi de Zipf (a={a})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)


# --- Lois continues ---

# 6️⃣ Loi normale
mu, sigma = 0, 1
x = np.linspace(mu - 4*sigma, mu + 4*sigma, 300)
pdf = norm.pdf(x, mu, sigma)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi normale N({mu},{sigma**2})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 7️⃣ Loi log-normale
s, scale = 0.954, np.exp(0)
x = np.linspace(0, 5, 300)
pdf = lognorm.pdf(x, s, scale=scale)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi log-normale (s={s})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 8️⃣ Loi uniforme continue
```

```
a, b = 0, 1
x = np.linspace(a - 0.2, b + 0.2, 300)
pdf = uniform.pdf(x, a, b - a)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi uniforme continue U({a},{b})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 9️⃣ Loi du χ²
df = 5
x = np.linspace(0, 20, 300)
pdf = chi2.pdf(x, df)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi du χ² (ddl={df})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 🔟 Loi de Pareto
b = 3
x = np.linspace(1, 10, 300)
pdf = pareto.pdf(x, b)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi de Pareto (b={b})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# Affiche toutes les figures
plt.show()

# Fonction pour calculer la moyenne et l'écart type

def calcul_moyenne_et_ecart():
    resultats = {}

    # --- Loi de Dirac ---
    x0 = 0
    resultats["Dirac"] = {"moyenne": x0, "ecart_type": 0}

    # --- Lois discrètes ---
    resultats["Uniforme discrète"] = {"moyenne": randint.mean(1, 7),
                                      "ecart_type": randint.std(1, 7)}

    resultats["Binomiale"] = {"moyenne": binom.mean(n=10, p=0.5),
                              "ecart_type": binom.std(n=10, p=0.5)}

    resultats["Poisson"] = {"moyenne": poisson.mean(mu=3),
                            "ecart_type": poisson.std(mu=3)}
```

```
    resultats["Zipf-Mandelbrot"] = {"moyenne": zipf.mean(a=2),
                                    "ecart_type": zipf.std(a=2)}

    # --- Lois continues ---
    resultats["Normale"] = {"moyenne": norm.mean(loc=0, scale=1),
                            "ecart_type": norm.std(loc=0, scale=1)}

    resultats["Log-normale"] = {"moyenne": lognorm.mean(s=0.954, scale=np.exp(0)),
                                "ecart_type": lognorm.std(s=0.954, scale=np.exp(0))}

    resultats["Uniforme continue"] = {"moyenne": uniform.mean(loc=0, scale=1),
                                      "ecart_type": uniform.std(loc=0, scale=1)}

    resultats["Chi2"] = {"moyenne": chi2.mean(df=5),
                         "ecart_type": chi2.std(df=5)}

    resultats["Pareto"] = {"moyenne": pareto.mean(b=3),
                           "ecart_type": pareto.std(b=3)}

    print(resultats)

calcul_moyenne_et_ecart()
```

Séance 4 :

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import (
    randint, binom, poisson, zipf, norm, lognorm,
    uniform, chi2, pareto
)

# --- Lois discrètes ---

# 1) Loi de Dirac
x = np.linspace(-5, 5, 100)
y = np.zeros_like(x)
y[np.argmin(np.abs(x - 0))] = 1
plt.figure()
plt.stem(x, y)
plt.title("Loi de Dirac (x0 = 0)")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 2) Loi uniforme discrète
a, b = 1, 6
x = np.arange(a, b+1)
pmf = randint.pmf(x, a, b+1)
plt.figure()
```

```
plt.stem(x, pmf)
plt.title(f"Loi uniforme discrète U({a},{b})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 3️⃣ Loi binomiale
n, p = 10, 0.5
x = np.arange(0, n+1)
pmf = binom.pmf(x, n, p)
plt.figure()
plt.stem(x, pmf)
plt.title(f"Loi binomiale B({n},{p})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 4️⃣ Loi de Poisson (discrète)
lam = 3
x = np.arange(0, 15)
pmf = poisson.pmf(x, lam)

plt.figure()
plt.stem(x, pmf)
plt.title(f"Loi de Poisson (λ={lam})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)

# 5️⃣ Loi de Zipf-Mandelbrot
a = 2
x = np.arange(1, 10)
pmf = zipf.pmf(x, a)
plt.figure()
plt.stem(x, pmf)
plt.title(f"Loi de Zipf (a={a})")
plt.xlabel("x"); plt.ylabel("P(X=x)")
plt.grid(alpha=0.3)


# --- Lois continues ---

# 6️⃣ Loi normale
mu, sigma = 0, 1
x = np.linspace(mu - 4*sigma, mu + 4*sigma, 300)
pdf = norm.pdf(x, mu, sigma)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi normale N({mu},{sigma**2})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 7️⃣ Loi log-normale
```

```
s, scale = 0.954, np.exp(0)
x = np.linspace(0, 5, 300)
pdf = lognorm.pdf(x, s, scale=scale)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi log-normale (s={s})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 8️⃣ Loi uniforme continue
a, b = 0, 1
x = np.linspace(a - 0.2, b + 0.2, 300)
pdf = uniform.pdf(x, a, b - a)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi uniforme continue U({a},{b})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 9️⃣ Loi du χ²
df = 5
x = np.linspace(0, 20, 300)
pdf = chi2.pdf(x, df)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi du χ² (ddl={df})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# 🔟 Loi de Pareto
b = 3
x = np.linspace(1, 10, 300)
pdf = pareto.pdf(x, b)
plt.figure()
plt.plot(x, pdf)
plt.title(f"Loi de Pareto (b={b})")
plt.xlabel("x"); plt.ylabel("f(x)")
plt.grid(alpha=0.3)

# Affiche toutes les figures
plt.show()

# Fonction pour calculer la moyenne et l'écart type

def calcul_moyenne_et_ecart():
    resultats = {}

    # --- Loi de Dirac ---
    x0 = 0
    resultats["Dirac"] = {"moyenne": x0, "ecart_type": 0}
```

```
    # --- Lois discrètes ---
    resultats["Uniforme discrète"] = {"moyenne": randint.mean(1, 7),
                                      "ecart_type": randint.std(1, 7)}

    resultats["Binomiale"] = {"moyenne": binom.mean(n=10, p=0.5),
                              "ecart_type": binom.std(n=10, p=0.5)}

    resultats["Poisson"] = {"moyenne": poisson.mean(mu=3),
                            "ecart_type": poisson.std(mu=3)}

    resultats["Zipf-Mandelbrot"] = {"moyenne": zipf.mean(a=2),
                                    "ecart_type": zipf.std(a=2)}

    # --- Lois continues ---
    resultats["Normale"] = {"moyenne": norm.mean(loc=0, scale=1),
                            "ecart_type": norm.std(loc=0, scale=1)}

    resultats["Log-normale"] = {"moyenne": lognorm.mean(s=0.954, scale=np.exp(0)),
                                "ecart_type": lognorm.std(s=0.954, scale=np.exp(0))}

    resultats["Uniforme continue"] = {"moyenne": uniform.mean(loc=0, scale=1),
                                      "ecart_type": uniform.std(loc=0, scale=1)}

    resultats["Chi2"] = {"moyenne": chi2.mean(df=5),
                         "ecart_type": chi2.std(df=5)}

    resultats["Pareto"] = {"moyenne": pareto.mean(b=3),
                           "ecart_type": pareto.std(b=3)}

print(resultats)

calcul_moyenne_et_ecart()
```

Séance 5 :

```
#coding:utf8

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats
import math
import re
import os
from itertools import combinations

def ouvrirUnFichier(nom):

    with open(nom, "r", encoding='utf-8') as fichier:
```

```
        contenu = pd.read_csv(fichier)
    return contenu

def conversionLog(liste):
    log = []
    for element in liste:
        try:
            val = float(element)
            if val > 0:
                log.append(math.log(val))
            else:
                log.append(float('nan'))
        except Exception:
            log.append(float('nan'))
    return log

def ordreDecroissant(liste):
    copie = list(liste)
    copie.sort(reverse=True)
    return copie

def ordrePopulation(pop, etat):
    ordrepop = []
    for i in range(len(pop)):
        try:
            val = float(pop[i])
except Exception:
            val = float('nan')
        if not np.isnan(val):
            ordrepop.append([val, etat[i]])
    ordrepop = ordreDecroissant(ordrepop)

    resultat = []
    for idx, item in enumerate(ordrepop):
        resultat.append([idx + 1, item[1]])  # 1 = plus grand
    return resultat

def classementPays(ordre1, ordre2):

    map1 = {etat: rang for rang, etat in ordre1}
    map2 = {etat: rang for rang, etat in ordre2}
    classement = []

    for etat in map1:
        if etat in map2:
            classement.append([map1[etat], map2[etat], etat])
    classement.sort(key=lambda x: x[0])
    return classement

def clean_numeric_series(s):
```

```
    def clean_val(v):
        if pd.isna(v):
            return float('nan')
        st = str(v)
        st = st.replace('\xa0', ' ')
        st = re.sub(r'[a-zA-Zµ°/]+', '', st)
        st = st.replace(',', '.')
        st = re.sub(r'\s+', '', st)
        st = re.sub(r'[^\d\.\-]', '', st)
        if st == '' or st == '.' or st == '-' or st == '-.':
            return float('nan')
        try:
            return float(st)
        except:
            return float('nan')
    return [clean_val(x) for x in s]

def safe_plot_save(x, y, xlabel, ylabel, title, fname):
    plt.figure(figsize=(8,5))
    plt.plot(x, y, marker='o', linestyle='-')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.tight_layout()
    plt.savefig(fname, dpi=200)
    plt.close()
    print(f"Image sauvegardée: {fname}")

def spearman_kendall(list1, list2):

    a = np.array(list1, dtype=float)
    b = np.array(list2, dtype=float)
    mask = ~ (np.isnan(a) | np.isnan(b))
    a2 = a[mask]
    b2 = b[mask]
    if len(a2) < 2:
        return (float('nan'), float('nan'), float('nan'), float('nan'))
    rs = scipy.stats.spearmanr(a2, b2)
    kt = scipy.stats.kendalltau(a2, b2)
return (rs.correlation, rs.pvalue, kt.correlation, kt.pvalue)

def compare_rankings(rang1_values, rang2_values, labels=None):
    sp_cor, sp_p, kd_cor, kd_p = spearman_kendall(rang1_values, rang2_values)
    return {
        'spearman_corr': sp_cor,
        'spearman_p': sp_p,
        'kendall_corr': kd_cor,
        'kendall_p': kd_p
    }
```

```
def analyze_all_years_rank_concordance(df, etat_col, pop_prefix='Pop '):

    years = []
    for col in df.columns:
        if isinstance(col, str) and col.strip().startswith(pop_prefix.strip()):
            m = re.search(r'(\d{4})', col)
            if m:
                years.append(int(m.group(1)))
    years = sorted(set(years))
    if not years:
        return None
    rankings = {}
    etats = df[etat_col].astype(str).tolist()
    for y in years:
        colname = f"{pop_prefix}{y}"
        if colname not in df.columns:
            colname_alt = f"{pop_prefix}{y}"
            if colname_alt not in df.columns:
                continue
        values = clean_numeric_series(df[colname])
        ordpop = ordrePopulation(values, etats)
         maprank = {etat: r for r, etat in ordpop}
        ranks_list = [maprank.get(e, float('nan')) for e in etats]
        rankings[y] = ranks_list
    years_list = sorted(rankings.keys())

n = len(years_list)
    tau_mat = pd.DataFrame(index=years_list, columns=years_list, dtype=float)
    for i, y1 in enumerate(years_list):
        for j, y2 in enumerate(years_list):
            if i <= j:
                a = rankings[y1]
                b = rankings[y2]
                tau, p = scipy.stats.kendalltau(
                    np.array(a, dtype=float),
                    np.array(b, dtype=float),
                    nan_policy='omit'
                )
                tau_mat.loc[y1, y2] = float(tau) if not pd.isna(tau) else float('nan')
                tau_mat.loc[y2, y1] = tau_mat.loc[y1, y2]
    return tau_mat

def main():
    data_dir = "./data"

    print("=== Traitement des îles ===")
    df_iles = pd.DataFrame(ouvrirUnFichier(os.path.join(data_dir, "island-index.csv")))
    col_surface_name = None
    for col in df_iles.columns:
        if 'surface' in str(col).lower():
```

```
            col_surface_name = col
            break
    if col_surface_name is None:
        raise KeyError("Impossible de trouver la colonne 'Surface' dans
island-index.csv. Vérifie les en-têtes.")
    print("Colonne Surface identifiée :", col_surface_name)
    surfaces_raw = df_iles[col_surface_name].tolist()
    surfaces = clean_numeric_series(surfaces_raw)

    continents = [85545323.0, 37856841.0, 7768030.0, 7605049.0]
    print(f"Ajout de {len(continents)} surfaces de continents.")
    surfaces.extend(continents)

    surfaces = [float(x) if not pd.isna(x) else float('nan') for x in surfaces]

    surfaces_nonan = [x for x in surfaces if not math.isnan(x)]
    surfaces_ord = ordreDecroissant(surfaces_nonan)
    print(f"{len(surfaces_ord)} surfaces valides après nettoyage et ajout.")

rangs = list(range(1, len(surfaces_ord) + 1))

    safe_plot_save(rangs, surfaces_ord,
                   xlabel="Rang",
                   ylabel="Surface (km2)",
                   title="Loi rang-taille - surfaces (échelle linéaire)",
                   fname=os.path.join(".", "rang_taille_linear.png"))

    log_rangs = conversionLog(rangs)
    log_surfaces = conversionLog(surfaces_ord)
    safe_plot_save(log_rangs, log_surfaces,
                   xlabel="log(Rang)",
                   ylabel="log(Surface (km2))",
                   title="Loi rang-taille - surfaces (log-log)",
                   fname=os.path.join(".", "rang_taille_loglog.png"))

    commentaire_test_rangs = (
        "# Commentaire : Oui, il est possible de faire un test sur les rangs. "
        "On utilise les coefficients de Spearman et Kendall pour mesurer la corrélation "
        "ou la concordance entre deux classements (ou deux listes de rangs)."
    )
 print(commentaire_test_rangs)

    print("\n=== Traitement des populations (Le Monde HS) ===")
    df_monde = pd.DataFrame(ouvrirUnFichier(os.path.join(data_dir,
"Le-Monde-HS-Etats-du-monde-2007-2025.csv")))

    etat_col = None
    for col in df_monde.columns:
        if 'etat' in str(col).lower() or 'état' in str(col).lower():
etat_col = col
```

```
            break
    if etat_col is None:
        raise KeyError("Impossible de trouver la colonne 'État' dans le fichier
Le-Monde...")

    print("Colonne État identifiée :", etat_col)

    def find_col_by_contains(possible):
        for col in df_monde.columns:
            low = str(col).lower()
            if all(part.lower() in low for part in possible.split('|')):
                return col
        return None

    pop2007_col = None
    pop2025_col = None
    dens2007_col = None
    dens2025_col = None
    for col in df_monde.columns:
        low = str(col).lower()
        if 'pop' in low and '2007' in low:
            pop2007_col = col
        if 'pop' in low and '2025' in low:
            pop2025_col = col
        if 'dens' in low and '2007' in low:
            dens2007_col = col
        if 'dens' in low and '2025' in low:
            dens2025_col = col

    if pop2007_col is None:
        for col in df_monde.columns:
            if '2007' in str(col):
                if 'pop' in str(col).lower() or 'population' in str(col).lower():
                    pop2007_col = col
                    break
    if pop2025_col is None:
        for col in df_monde.columns:
            if '2025' in str(col):
                if 'pop' in str(col).lower() or 'population' in str(col).lower():
                    pop2025_col = col

break
    if dens2007_col is None:
        for col in df_monde.columns:
            if '2007' in str(col) and ('dens' in str(col).lower() or 'density' in str(col).lower()):
                dens2007_col = col
                break
    if dens2025_col is None:
        for col in df_monde.columns:
            if '2025' in str(col) and ('dens' in str(col).lower() or 'density' in str(col).lower()):
```

```
            dens2025_col = col
            break

    print("Pop2007 col:", pop2007_col)
    print("Pop2025 col:", pop2025_col)
    print("Dens2007 col:", dens2007_col)
    print("Dens2025 col:", dens2025_col)

    etats = df_monde[etat_col].astype(str).tolist()
    pop2007 = clean_numeric_series(df_monde[pop2007_col]) if pop2007_col else
[float('nan')] * len(etats)
    pop2025 = clean_numeric_series(df_monde[pop2025_col]) if pop2025_col else
[float('nan')] * len(etats)
    dens2007 = clean_numeric_series(df_monde[dens2007_col]) if dens2007_col else
[float('nan')] * len(etats)
    dens2025 = clean_numeric_series(df_monde[dens2025_col]) if dens2025_col else
[float('nan')] * len(etats)


    ord_pop2007 = ordrePopulation(pop2007, etats)
    ord_pop2025 = ordrePopulation(pop2025, etats)
    ord_dens2007 = ordrePopulation(dens2007, etats)
ord_dens2025 = ordrePopulation(dens2025, etats)

    classement_2007_pop_vs_dens = classementPays(ord_pop2007, ord_dens2007)

    rangs_pop_2007 = [item[0] for item in classement_2007_pop_vs_dens]
    rangs_dens_2007 = [item[1] for item in classement_2007_pop_vs_dens]
    etats_communs_2007 = [item[2] for item in classement_2007_pop_vs_dens]

    print(f"{len(etats_communs_2007)} pays présents dans les deux classements (pop
2007 & densité 2007).")

sp_corr, sp_p, kd_corr, kd_p = spearman_kendall(rangs_pop_2007,
rangs_dens_2007)
    print("\nRésultats 2007 (Population vs Densité) :")
    print(f"Spearman r = {sp_corr:.4f} (p = {sp_p:.4g})")
    print(f"Kendall tau = {kd_corr:.4f} (p = {kd_p:.4g})")


    df_cl_2007 = pd.DataFrame(classement_2007_pop_vs_dens,
columns=['rang_pop_2007','rang_dens_2007','etat'])
    df_cl_2007.to_csv("classement_pop_vs_dens_2007.csv", index=False,
encoding='utf-8')
    print("Fichier classement_pop_vs_dens_2007.csv généré.")

    print("\n(Bonus) Pour comparer 2 classements, utiliser compare_rankings(rang1,
rang2).")
```

```
    print("\n=== Bonus: Analyse des concordances annuelles (2007-2025) pour les
populations ===")
    tau_matrix = analyze_all_years_rank_concordance(df_monde, etat_col,
pop_prefix='Pop ')
    if tau_matrix is not None:
        tau_matrix.to_csv("kendall_tau_matrix_population_years_2007_2025.csv",
encoding='utf-8')
        print("Matrice Kendall tau (années) sauvegardée:
kendall_tau_matrix_population_years_2007_2025.csv")
    else:
        print("Aucune colonne population annuelle détectée avec le préfixe 'Pop '.
Vérifie les en-têtes du CSV.")

    print("\nTraitement terminé.")

if __name__ == "__main__":
    main()
```