Rapport d’activité

Analyse de données - Parcours débutant

Lison Blanc

Master 1 GEAD, parcours Sociétés Cultures et Territoires, Sorbonne Université

Table des matières

Séance 1

Objectifs

- Comprendre le format et les objectifs du cours.
- Se familiariser avec les outils : Python, GitHub, Docker.
- Installer les outils et préparer l'espace de travail pour le semestre.

Installation des outils

Après le cours de présentation du semestre, du calendrier et du découpage des séances, la première étape restait encore la plus importante : préparer ma machine et installer tous les outils. Une fois les comptes *GitHub* et *Docker* créés, il s'agissait de les installer et de les raccorder à mon ordinateur et mon espace de travail local.
Dans un souci de transparence, j'annonce que cette étape a été faite avec de l'aide : celle du livret distribué en cours le matin même, et celle d'un de mes proches, plus compétent dans le domaine. Ainsi, une fois le livret épluché pour s'assurer des consignes et étapes précises, nous nous sommes penchés sur l'installation de *Notepad++, Docker Desktop*, ou encore *GitHub Desktop*. Une fois tout cela initié, nous avons créé un lien vers ma page *GitHub* ainsi qu'un « *Readme* » sur ma page d'accueil, par pure convention, comme me l'a expliqué mon cousin. Ce *Readme* n'est pas réellement nécessaire à l'exercice, il n'a pas été rempli comme il aurait dû pour une belle page *GitHub*, mais il m'a permis de comprendre quelques mécanismes de la plateforme. De là, j'ai pu opérer mon *git init* et mon *git add* en parallèle de la création de mon *repository* sur la plateforme, cela me permet aujourd'hui d'ajouter au fur et à mesure mes codes et mes images, graphiques et autres productions de code grâce à *GitHub Desktop* et sa gestion des *git commit* et *git push* via un interface.
Une fois tout cela fait et après m'être assurée de bien comprendre toutes les logiques de fonctionnement, mon ordinateur était prêt à fonctionner. Je n'ai d'ailleurs eu aucun problème depuis mon installation côté *Docker* et *GitHub*, et *Python* fonctionne correctement.

Séance 2

Objectifs

- Manipuler un fichier C.S.V.
- Faire des sorties graphiques
- Utiliser les bibliothèques Pandas (données) et Matplotlib (graphiques)
  N.B. pd et plt sont des alias qui remplacent respectivement pandas et matplotlib.pyplot.
- Calculer des effectifs
- Calculer des fréquences
- Faire des graphiques (diagrammes en bâton et circulaires, et histogrammes)

Questions de cours

1. Quel est le positionnement de la géographie par rapport aux statistiques ?

La géographie occupe une position singulière vis-à-vis des statistiques : discipline productrice d'un grand volume de données, elle a historiquement sous-estimé la rigueur mathématique inhérente aux méthodes statistiques. Beaucoup de géographes n'ont pas reçu de formation mathématique approfondie et peuvent donc mal utiliser les outils statistiques, ce qui conduit parfois à des interprétations discutables.
Néanmoins, la production massive d'informations géographiques rend indispensable le recours aux méthodes statistiques pour classer, résumer, visualiser et modéliser les phénomènes spatiaux. L'émergence et le développement de l'analyse spatiale ont rapproché la géographie d'une posture plus nomothétique et scientifique : plutôt que de rester une simple méthode descriptive, la géographie moderne tend à intégrer systématiquement les techniques statistiques pour dégager des régularités à différentes échelles.

2. Le hasard existe-t-il en géographie ?

La question du hasard en géographie relève tant de la philosophie que de la pratique.
Sur le plan philosophique, deux positions coexistent : le déterminisme où tout événement a une cause identifiable et l'idée que le hasard existe mais qu'il peut correspondre à des causes encore inconnues.
En pratique statistique, on admet l'aléa : il est illusoire de prévoir précisément chaque réalisation individuelle, mais il est possible d'identifier des tendances et des comportements probabilistes à une échelle plus globale. D'autre part, on distingue deux types d'aléas : le « hasard bénin », qui peut souvent être approché par des lois telles que la loi normale, et le « hasard sauvage », associé à des événements extrêmes et à des lois de queue lourde (par exemple de type paretien). En géographie, l'acceptation de l'aléa permet de dégager des certitudes globales malgré des variations locales imprévisibles.

3. Quels sont les types d'information géographique.

L'information géographique se présente principalement sous deux formes :

- Les **séries attributaires**, qui renseignent sur les caractéristiques d'entités spatiales : population, variables sociales ou économiques, mesures physiques telles que températures ou précipitations…
- Les **séries géométriques**, qui décrivent la morphologie et la géométrie des ensembles spatiaux.

Les premières constituent la base attributaire d'un SIG, tandis que les secondes portent sur les aspects géométriques et seront traitées par des méthodes spécifiques.

4. Quels sont les besoins de la géographie au niveau de l'analyse de données ?

La géographie a besoin d'un ensemble d'étapes et d'outils pour exploiter correctement les données : définir des nomenclatures claires et hiérarchisées avant le recueil, documenter les jeux de données avec des métadonnées complètes (définitions, dates et lieux d'observation, modalités d'échantillonnage, etc.) pour évaluer la fiabilité des sources, résumer et visualiser les distributions (statistique descriptive), identifier la loi de probabilité adaptée et construire des modèles explicatifs ou prédictifs (statistique inférentielle), réduire la dimensionnalité quand nécessaire (analyses factorielles) et classer ou segmenter les observations (classifications).
Enfin, il est essentiel de confronter les résultats statistiques à la méthodologie de collecte et aux connaissances disciplinaires pour éviter des interprétations abusives.

5. Quelles sont les différences entre la statistique descriptive et la statistique explicative ?

La statistique descriptive a pour objectif de rendre compte, de manière synthétique et visuelle, des caractéristiques d'un jeu de données : mesures de position et de dispersion, détection d'outliers, représentations graphiques (histogrammes, boîtes à moustaches, etc.), et préparation des comparaisons. Elle sert d'étape exploratoire indispensable. La statistique explicative (ou statistique mathématique / inférentielle), en revanche, vise à expliquer ou prédire : elle cherche à relier une variable réponse à un ensemble de variables explicatives par des modèles (régressions, ANOVA, modèles généralisés, régression logistique, analyse discriminante, etc.), permet de tester des hypothèses et d'estimer la fiabilité des inférences.
En résumé : la descriptive décrit et synthétise, l'explicative modélise et teste.

6. Quelles sont les types de visualisations de données en géographie ? Comment choisir celles-ci ?

Parmi les représentations couramment utilisées :

- Pour les **variables continues** : histogrammes, polygones de fréquence, courbes cumulative décroissante, boîtes à moustaches, lissage normal

- Pour les **variables discrètes** : diagrammes en bâtons
- Pour les **variables qualitatives** : diagrammes camembert

Mais il existe également d'autres options : diagrammes à rectangles horizontaux, polygones, représentations de proximités issues d'analyses factorielles.

Le choix d'une visualisation dépend du type de variable (qualitative, quantitative, discrète, continue) et de l'objectif (montrer la distribution, comparer des groupes, repérer des valeurs extrêmes, visualiser des proximités multivariées). Par exemple, un histogramme apparait plus adapté pour explorer la forme d'une distribution continue, un secteur pour présenter la composition relative d'une variable catégorielle, et une boîte à moustaches pour comparer la dispersion et détecter des outliers entre plusieurs groupes.

7. Quelles sont les méthodes d'analyse de données possibles ?

Il est possible de classer les méthodes en trois grandes familles :

- **Méthodes descriptives et multidimensionnelles** : analyse en composantes principales (ACP) pour réduire la dimensionnalité et visualiser des variables quantitatives ; analyse factorielle des correspondances (AFC) et correspondances multiples (ACM) pour des variables qualitatives, analyses mixtes (AFDM, AFM) pour jeux hétérogènes, classifications (classification ascendante hiérarchique, nuées dynamiques) pour regrouper individus ou variables, analyses des proximités pour cartographier des similarités.

- **Méthodes explicatives** : régression simple et multiple (pour variable quantitative), ANOVA et modèles linéaires généraux, régression logistique et analyse discriminante (pour réponses qualitatives), segmentation pour la typologie.

- **Méthodes de prévision** : modèles de séries temporelles qui relient le présent à ses valeurs passées (modèles autorégressifs, etc.) afin d'effectuer des prévisions. Ces familles sont complémentaires et souvent combinées dans une démarche d'analyse.

8. Comment définiriez-vous : (a) population statistique ? (b) individu statistique ? (c) caractères statistiques ? (d) modalités statistiques ? Quels sont les types de caractères? Existe-t-il une hiérarchie entre eux ?

- **Population statistique** : ensemble des unités sur lequel porte l'étude (par exemple l'ensemble des habitants d'un territoire).

- **Individu statistique** : unité élémentaire de la population, aussi appelée unité spatiale quand elle est localisable (ville, commune, salarié, etc.).

- **Caractères statistiques** : propriétés observées sur chaque individu (âge, revenu, altitude, catégorie socio-professionnelle, etc.).

- **Modalités** : valeurs possibles d'un caractère. Elles doivent être mutuellement exclusives et collectivement exhaustives (chaque individu appartient exactement à une modalité).

Types de caractères :
- **Qualitatifs** : nominal (catégories sans ordre) et ordinal (catégories ordonnées).
- **Quantitatifs** : discrets (valeurs entières isolées, par ex. nombre d'enfants) et continus (valeurs prises dans un intervalle, par ex. âge ou salaire ; on distingue variables d'intervalle et variables de rapport selon la signification du zéro).

Concernant une hiérarchie, il n'existe pas de hiérarchie de « valeur » entre ces types, mais on distingue au niveau des unités spatiales les unités primaires (données non agrégées, «atomes ») et les unités secondaires (données agrégées, « molécules »), ce qui constitue une forme de hiérarchie pertinente pour l'analyse spatiale et l'agrégation des données.

9. Comment mesurer une amplitude et une densité ?

Amplitude d'une classe : pour une classe d'intervalle $]a, b]$, l'amplitude est la longueur $b - a$.

Densité d'une classe : rapport entre l'effectif $n_i$de la classe et son amplitude ; formellement $d = \frac{n_i}{b-a}$. La densité permet de comparer des classes de largeur

10. À quoi servent les formules de Sturges et de Yule ?

Ces formules sont des règles empiriques destinées à guider le choix du nombre de classes k lors de la discrétisation d'une variable continue (préparation d'un histogramme). Elles visent à éviter un découpage trop fin (trop de classes, perte de lisibilité) ou trop grossier (trop peu de classes, perte d'information). Deux formules usuelles : la formule de Sturges $k \approx 1 + 3{,}2222 \times \log_{10} n$et la formule de Yule $k \approx 2{,}5\sqrt[4]{n}$, qui donnent des valeurs approximatives en fonction de la taille de l'échantillon n.

11. Comment définir un effectif ? Comment calculer une fréquence et une fréquence cumulée ? Qu'est-ce qu'une distribution statistique ?

- **Effectif** n_i: nombre d'observations correspondant à une modalité ou à une classe.
- **Fréquence relative** f_i: proportion d'observations pour la modalité i, calculée par $f_i = \frac{n_i}{n}$ où n est l'effectif total.
- **Fréquence cumulée** jusqu'à la k-ième modalité : somme des fréquences des modalités inférieures ou égales à k, soit $F_k = \sum_{i=1}^{k} \quad f_i = \frac{1}{n}\sum_{i=1}^{k} \quad n_i$. Sur l'ensemble des modalités, la somme des fréquences vaut 1.

- **Distribution statistique** : représentation empirique de la répartition des effectifs ou fréquences sur les modalités ou classes. Elle constitue le lien observable entre les données et les lois de probabilité théoriques, et sert de base à la description et au choix des modèles statistiques.

Mise en œuvre avec Python

Etapes 1 à 4 : Introduction

Avant toute tentative de manipulation et de codage, la première étape de cet exercice consiste à télécharger les fichiers de la séance 2 et à les placer au bon endroit dans notre dossier local relié à GitHub. Une fois cela fait, avant d'ouvrir le fichier main.py sur Notepad++ et de passer aux choses sérieuses, il faut se « positionner » dans le dossier de la séance. Pour cela, il suffit de se rendre dans le dossier Séance-02, dans le dossier src, et de sélectionner l'option « ouvrir dans le terminal ». Ainsi nous voici arrivés :

```
PS C:\Users\33664\Desktop\Sorbonne U\S1\Blanc-2025-2026-Analyse-de-donnees\Seance-2\src>
```

Une fois ces premières étapes effectuées nous pouvons ouvrir le fichier main.py et trouver notre base de code. Cette base nous permet de lire un fichier CSV contenant les résultats du premier tour des élections présidentielles de 2022 par département. Il constitue le cœur de notre travail.

Etapes 5 à 13 : Codage

Il s'agit dans un premier temps de vérifier si le fichier CSV est bien lu par notre programme et si ses données et colonnes que l'on retrouve sont cohérentes (étape 5). On utilise ainsi la commande suivante :

```
# Etape 5 - afficher le contenu du tableau
print(contenu)
```

Une fois jouée dans le terminal grâce à la commande `Docker-compose run python`, on obtient bien le résultat suivant :

```
[107 rows x 56 columns]
```

Et les 5 premières lignes affichées sont bien les départements de l'Ain, l'Aisne et l'Allier, et les dernières lignes concernent les ressortissants français hors de France ainsi que les Outre-Mer.

Ensuite, nous cherchons à calculer le nombre de lignes et de colonnes (étape 6). Nous allons alors nous aider des commandes suivantes :

```
# Etape 6 - Calculer le nombre de lignes et de colonnes
print("Nombre de lignes :", len(contenu))
print("Nombre de colonnes :", len(contenu.columns))
```

Une fois jouée dans le terminal, nous obtenons le résultat suivant :

```
Nombre de lignes : 107
Nombre de colonnes : 56
```

Ainsi, chaque ligne correspond à un département (métropole, outre-mer, zones consulaire, expatriés, etc), le nombre 107 semble alors cohérent. Les colonnes quant à elles correspondent aux variables (résultat ou métadonnée).

Dans un prochain temps nous souhaitons identifier la nature statistique de nos variables en en listant les types, selon la classification suivante : int, float, str, bool (étape 7). Voici à quoi ressemble nos lignes de code :

```
# Etape 7 - Liste sur le type de chaque colonne
print(contenu.dtypes)
```

Le terminal affiche alors le résultat suivant :

```
Code du département       object
Libellé du département    object
Inscrits                   int64
Abstentions              float64
Votants                  float64
Blancs                   float64
Nuls                     float64
Exprimés                 float64
Sexe                      object
Nom                       object
Prénom                    object
Voix                     float64
Sexe.1                    object
Nom.1                     object
Prénom.1                  object
Voix.1                   float64
Sexe.2                    object
Nom.2                     object
Prénom.2                  object
Voix.2                   float64
Sexe.3                    object
Nom.3                     object
Prénom.3                  object
Voix.3                   float64
Sexe.4                    object
Nom.4                     object
Prénom.4                  object
Voix.4                   float64
Sexe.5                    object
Nom.5                     object
Prénom.5                  object
Voix.5                   float64
Sexe.6                    object
Nom.6                     object
Prénom.6                  object
Voix.6                   float64
Sexe.7                    object
Nom.7                     object
Prénom.7                  object
Voix.7                   float64
Sexe.8                    object
Nom.8                     object
Prénom.8                  object
Voix.8                   float64
Sexe.9                    object
Nom.9                     object
Prénom.9                  object
Voix.9                   float64
Sexe.10                   object
Nom.10                    object
Prénom.10                 object
Voix.10                  float64
Sexe.11                   object
Nom.11                    object
Prénom.11                 object
Voix.11                  float64
dtype: object
```

On retrouve ainsi des variables qualitatives ou « object », comme le code du département, le libellé du département, le nom, le prénom, etc. Mais également des variables quantitatives : « int64 » ou « float64 », comme les inscrits, les votants, l'abstention, les voix, etc. Cette impression dans notre terminal nous indique que les colonnes ont bien été reconnues comme numériques, ce qui nous permettra de réaliser des opérations de sommes et des graphiques.
Nous voulons maintenant accéder à un aperçu du tableau (étape 8). Les lignes suivantes nous permettent de le réaliser :

```
# Etape 8 Liste sur le type de chaque colonne
```

```
print("Aperçu du tableau:")
print(contenu.head)
```

Une fois exécutées, ces lignes nous donnent le résultat suivant :

```
Aperçu du tableau:
<bound method NDFrame.head of     Code du département          Libellé du département  Inscrits  ...         Nom.11  Prénom.11  Voix
.11
0                   01                           Ain    438109  ...  DUPONT-AIGNAN    Nicolas   8998.0
1                   02                         Aisne    373544  ...  DUPONT-AIGNAN    Nicolas   5790.0
2                   03                        Allier    249991  ...  DUPONT-AIGNAN    Nicolas   4216.0
3                   04          Alpes-de-Haute-Provence    128075  ...  DUPONT-AIGNAN    Nicolas   2504.0
4                   05                  Hautes-Alpes    113519  ...  DUPONT-AIGNAN    Nicolas   2142.0
..                 ...                           ...       ...  ...           ...        ...      ...
102                 ZP                Polynésie française    205576  ...  DUPONT-AIGNAN    Nicolas   1969.0
103                 ZS         Saint-Pierre-et-Miquelon      5045  ...  DUPONT-AIGNAN    Nicolas     82.0
104                 ZW                 Wallis et Futuna      9528  ...  DUPONT-AIGNAN    Nicolas    244.0
105                 ZX    Saint-Martin/Saint-Barthélemy     24414  ...  DUPONT-AIGNAN    Nicolas    339.0
106                 ZZ  Français établis hors de France   1435746  ...  DUPONT-AIGNAN    Nicolas   7074.0
```

Delà, notre objectif est d'isoler la colonne des inscrits pour l'afficher (étape 9). Nous utiliserons la commande suivante :

```
# Etape 9 - Sélectionner la colonne "Inscrits"
print("Nombre des inscrits par départements :")
print(contenu.Inscrits)
```

Pour le résultat suivant :

```
Nombre des inscrits par départements :
0       438109
1       373544
2       249991
3       128075
4       113519
        ...
102     205576
103       5045
104       9528
105      24414
106    1435746
Name: Inscrits, Length: 107, dtype: int64
```

Ainsi, les effectifs d'inscrits sont bien extraits et cohérents avec les totaux départementaux

A partir de notre colonne, nous voulons maintenant calculer la somme des colonnes numériques et seulement numériques (étapes 10). Pour ce faire nous jouerons le code suivant :

```
# Etape 10 - Calculer les effectifs de chaque colonnes
print("\nNombre des inscrits par département :")
print(contenu["Inscrits"])
print("\nSommes colonnes numériques:\n", contenu.select_dtypes(include=["int64","flot64"]).sum())
```

La dernière ligne (normalement sur une seule ligne sans retour à la ligne) nous permet de ne sélectionner que les colonnes numériques, en filtrant automatiquement les colonnes selon leur type de données.

Le résultat suivant s'affiche ainsi :

```
Sommes colonnes numériques :
 Inscrits       48747876.0
Abstentions     12824169.0
Votants         35923707.0
Blancs            543609.0
Nuls              247151.0
Exprimés        35132947.0
Voix              197094.0
Voix.1            802422.0
Voix.2           9783058.0
Voix.3           1101387.0
Voix.4           8133828.0
Voix.5           2485226.0
Voix.6           7712520.0
Voix.7            616478.0
Voix.8           1627853.0
Voix.9           1679001.0
Voix.10           268904.0
Voix.11           725176.0
dtype: float64
```

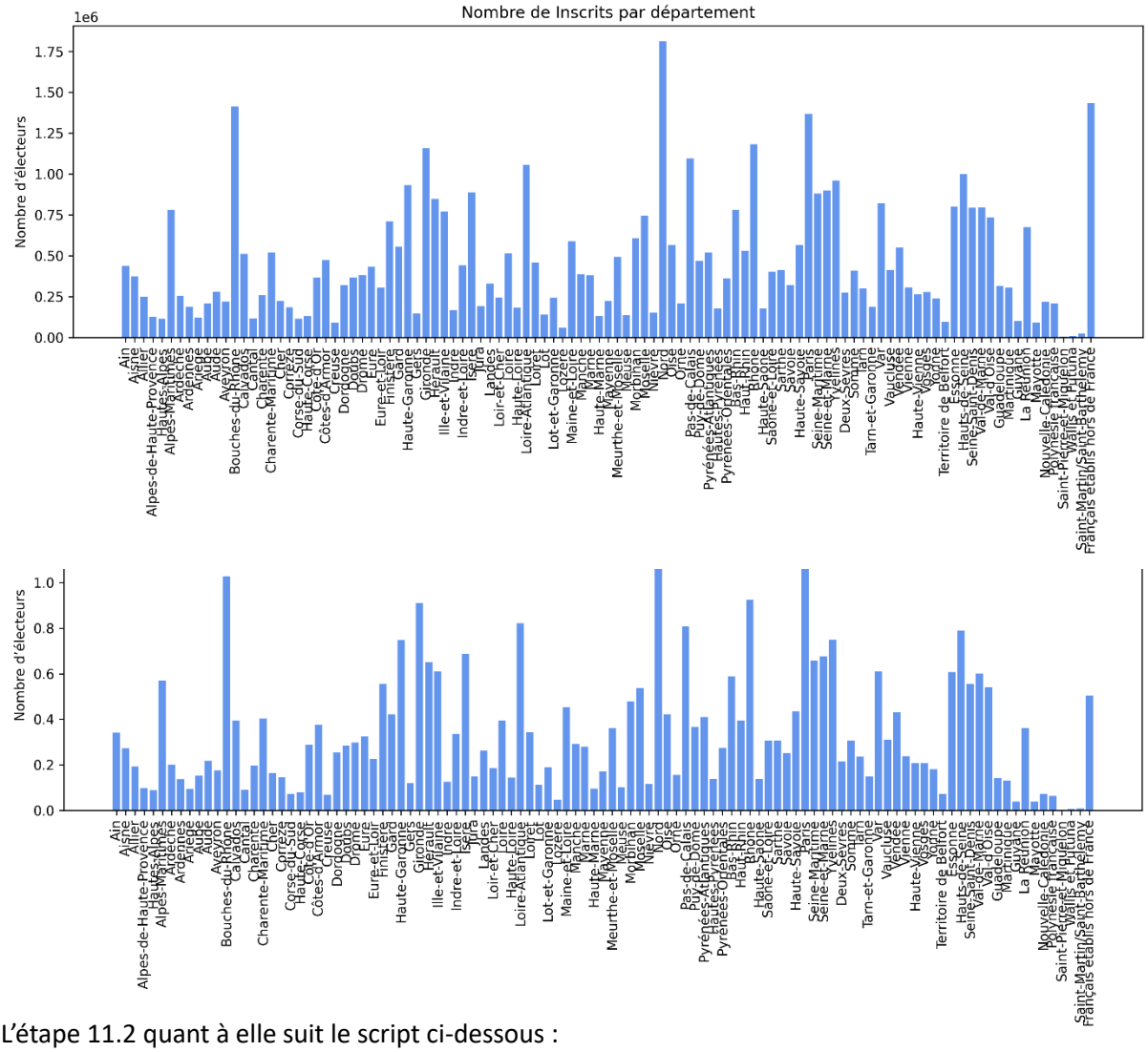
Ainsi, le nombre d'inscrit en 2022 (environ 48,7 millions[1]), correspond bien au nombre d'électeurs affiché par notre programme. Les totaux des candidats (colonnes « Voix.X ») s'additionnent également correctement à la somme des exprimés.

Dans un premier temps, l'étape 11 a été interprétée dans le sens où l'on nous demandait deux diagrammes en barres, un pour les inscrits, un pour les votants (étape 11). Ces diagrammes nous auraient permis de comparer visuellement les inscrits et les votants par département. Cependant, après comparaison avec les résultats du script de Flavie Gherardi (M1 GAED, SCT), il apparait clairement que l'étape 11 attendait un diagramme par département avec une colonne pour les inscrits et une pour les votants, et ce dans chaque diagramme. Ne voulant pas supprimer les premières lignes de codes écrites, nous pouvons ici retrouver une étape 11.1 pour les 2 diagrammes et une étape 11.2 pour les diagrammes par département. L'étape 11.2 a été réalisée à l'aide des lignes suivantes :

```
# Etape 11 - Diagrammes en barres : inscrits & votants
#Etape 11.1 - Diagrammes globaux
os.makedirs("images", exist_ok=True)
for c in ["Inscrits", "Votants"]:
    plt.figure(figsize=(12,6))
    plt.bar(contenu["Libellé du département"], contenu[c], color="cornflowerblue")
    plt.xticks(rotation=90)
    plt.title(f"Nombre de {c} par département")
    plt.ylabel("Nombre d'électeurs")
    plt.tight_layout()
    plt.savefig(f"images/{c}.png", dpi=300)
    plt.close()
print("→ Diagrammes 'Inscrits.png' et 'Votants.png' enregistrés")
```

[1] https://www.archives-resultats-elections.interieur.gouv.fr/resultats/presidentielle-2022/FE.php

Ainsi, dans un dossier « image », les deux fichiers suivants ont été créés :

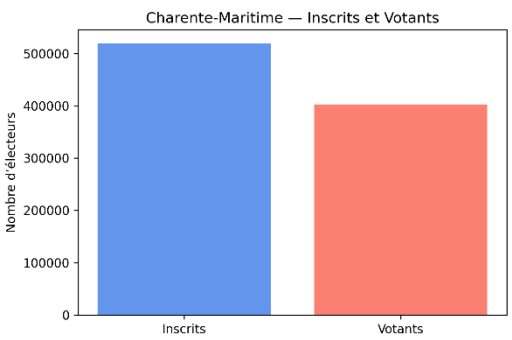


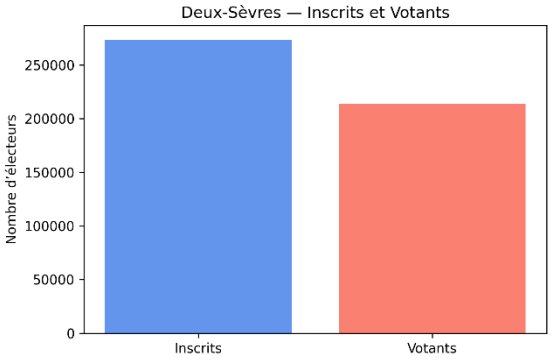
L'étape 11.2 quant à elle suit le script ci-dessous :

```
#Etape 11.2 - Diagramme par département
os.makedirs("images_par_dept", exist_ok=True)

for i, dept in enumerate(contenu["Libellé du département"]):
    inscrits = contenu.loc[i, "Inscrits"]
    votants = contenu.loc[i, "Votants"]

    plt.figure(figsize=(6,4))
    plt.bar(["Inscrits", "Votants"], [inscrits, votants], color=["cornflowerblue", "salmon"])
    plt.title(f"{dept} — Inscrits et Votants")
    plt.ylabel("Nombre d'électeurs")
    plt.tight_layout()

    # sauvegarde sécurisée
    filename = safe_filename(dept) + ".png"
    plt.savefig(f"images_par_dept/{filename}", dpi=300)
    plt.close()
```

```
print("→ Diagrammes individuels par département enregistrés dans 'images_par_dept/'")
```

Elle a quant à elle produit un dossier intitulé « images_par_dep » dans lequel on retrouve bien un fichier png par département. Ils prennent d'ailleurs la forme suivante :

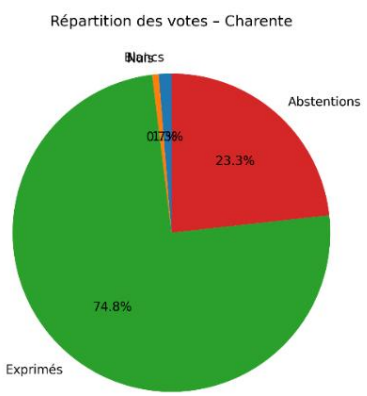


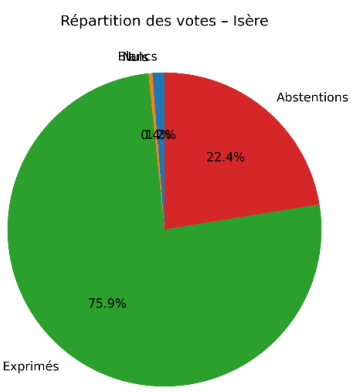


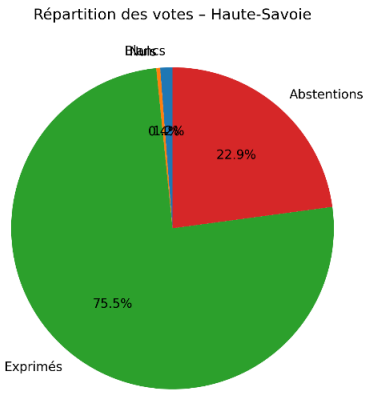
Dans l'étape suivante (étape 12) nous voulons réaliser des diagrammes circulaires comprenant les catégories de votes : blancs, nuls, exprimés, et abstention, pour chaque département. Voici le code utilisé :

```
# Etape 12 - Diagrammes circulaires : blancs, nuls, exprimés, abstentions
os.makedirs("images_pie", exist_ok=True)
cols_pie = ["Blancs", "Nuls", "Exprimés", "Abstentions"]
for _, row in contenu.iterrows():
    dep = re.sub(r"[^\w\-]", "_", row["Libellé du département"])
    plt.figure(figsize=(5,5))
    plt.pie([row[c] for c in cols_pie], labels=cols_pie, autopct="%1.1f%%", startangle=90)
    plt.title(f"Répartition des votes – {row['Libellé du département']}")
    plt.tight_layout()
    plt.savefig(f"images_pie/{row['Code du département']}_{dep}.png", dpi=300)
    plt.close()
print("→ Diagrammes circulaires enregistrés dans /images_pie")
```

Il nous permet d'obtenir un dossier « images_pie » contenant 107 fichiers, un par département, ce qui correspond bien au nombre de lignes dans le tableau. Chaque graphique affiche les parts respectives de nos catégories mentionnées plus haut. Voici un exemple de quelques graphiques sélectionnés au hasard :



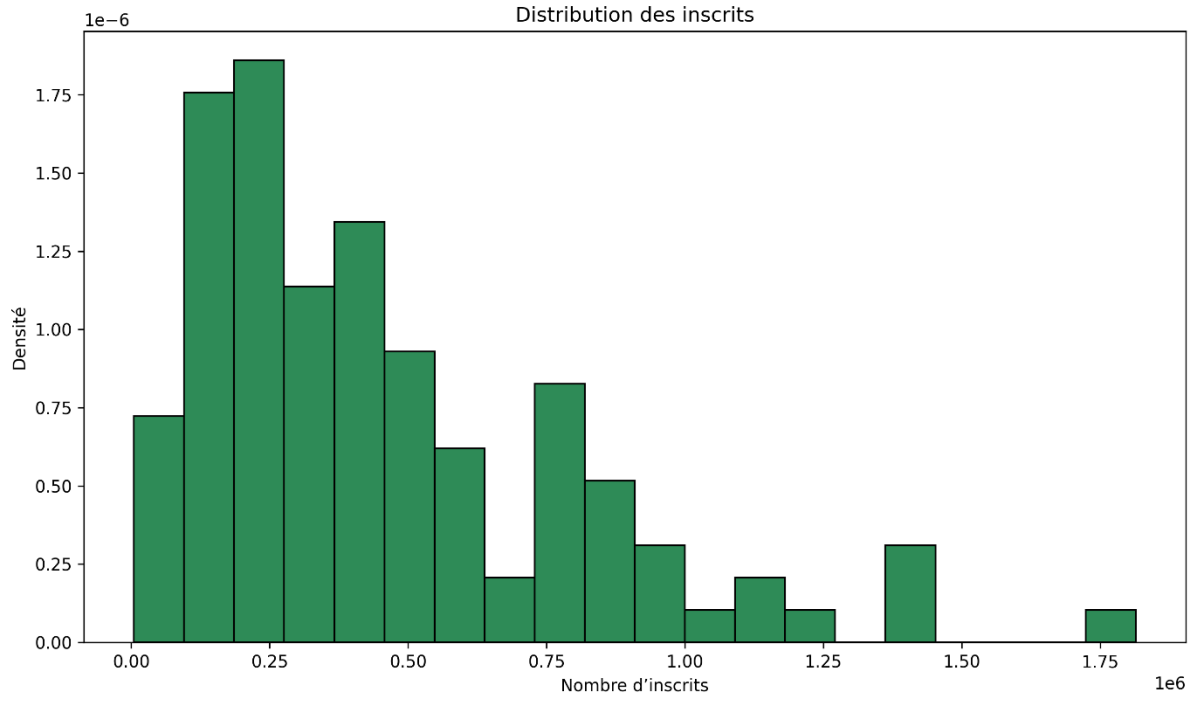




Nous voulons à présent une troisième et dernière représentation graphique pour visualiser la répartition du nombre d'inscrits entre les départements (étape 13). Nous allons construire un histogramme de la distribution des inscrits à l'aide du code suivant :

```
# Etape 13 - Histogramme de la distribution des inscrits
os.makedirs("images_hist", exist_ok=True)
plt.figure(figsize=(10,6))
plt.hist(contenu["Inscrits"], bins=20, color="seagreen", edgecolor="black", density=True)
plt.title("Distribution des inscrits")
plt.xlabel("Nombre d'inscrits")
plt.ylabel("Densité")
plt.tight_layout()
plt.savefig("images_hist/histogramme_inscrits.png", dpi=300)
plt.close()
print("→ Histogramme enregistré dans /images_hist")
```

Ces commandes nous créent un dossier « images_hist » qui contient le fameux histogramme que voici :



Etape 14 : Bonus

L'étape complémentaire 14 a pour objectif la visualisation de la répartition des voix entre les différents candidats, d'abord pour chaque département, puis pour l'ensemble du territoire Français. C'est une visualisation qui permet de rendre compte graphiquement de la part de chaque candidat dans les suffrages exprimés, tant localement que nationalement. Dans cet objectif, on utilise le code suivant :

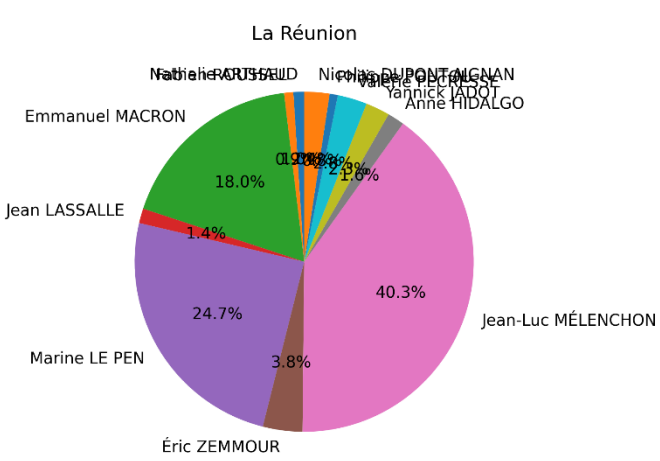
```
#  Etape 14 Bonus - diagrammes circulaires des voix par candidat
os.makedirs("images_voix", exist_ok=True)
vcols = [c for c in contenu.columns if c.startswith("Voix")]
def noms(r): return [r.get(f'Prénom.{i}', r['Prénom']) + " " + r.get(f'Nom.{i}', r['Nom']) for i in
range(len(vcols))]
```

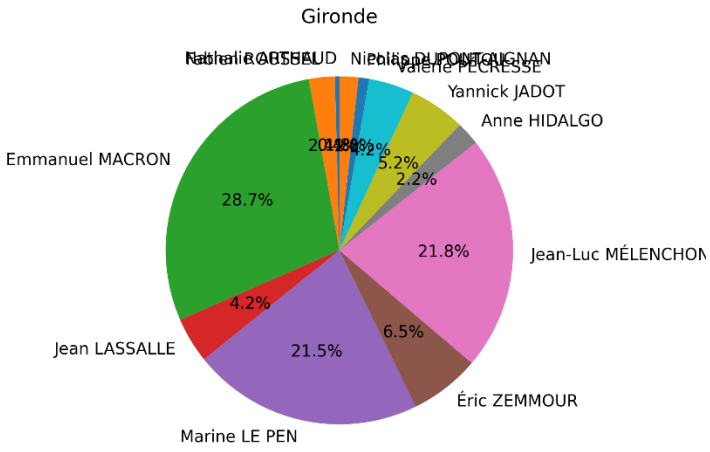
```
for _, r in contenu.iterrows():
    dep = re.sub(r"[^\w\-]", "_", r["Libellé du département"])
    plt.pie(r[vcols], labels=noms(r), autopct='%1.1f%%', startangle=90)
    plt.title(r['Libellé du département'])
    plt.savefig(f"images_voix/{r['Code du département']}_{dep}.png", dpi=300)
    plt.close()
plt.pie(contenu[vcols].sum(), labels=noms(contenu.iloc[0]), autopct='%1.1f%%', startangle=90)
plt.title("France entière")
plt.savefig("images_voix/France.png", dpi=300)
plt.close()
print("→ Diagrammes voix par candidat enregistrés (départements + France)")
```

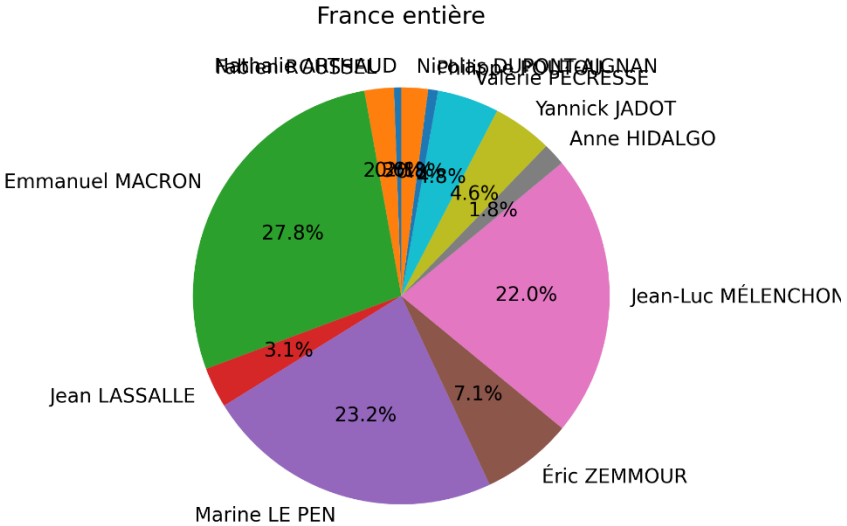
Nous obtenons comme résultat un fichier intitulé « images_voix » contenant 108 items : un diagramme circulaire par département et un diagramme pour la France entière. Comme pour les autres représentations graphiques, le terminal nous a d'ailleurs confirmé la création de ces éléments, de la manière suivante :

```
→ Diagrammes voix par candidat enregistrés (départements + France)
```

Voici quelques-uns de ces diagrammes départementaux, ainsi que celui à l'échelle de la France.







Séance 3

Objectifs

- Découvrir les méthodes de Pandas permettant de calculer les paramètres d'une série statistique
- Tracer une boîte de dispersion

Questions de cours

1. Quel caractère est le plus général : le caractère quantitatif ou le caractère qualitatif ? Justifier pourquoi.

Le caractère qualitatif est le plus général, car il permet de décrire des catégories ou des modalités, comme la couleur, le genre, un type de produit, *etc*, sans nécessiter de mesure numérique. D'autre part, les caractères quantitatifs sont un cas particulier des caractères qualitatifs. Ils expriment une valeur numérique mesurable, comme la taille, l'âge, le revenu. Ainsi, il apparait que tout caractère quantitatif peut être transformé en qualitatif, par regroupement en classes, mais l'inverse n'est pas toujours possible.

2. Que sont les caractères quantitatifs discrets et caractères quantitatifs continus ? Pourquoi les distinguer ?

- **Les caractères quantitatifs discrets** : prend un nombre fini ou dénombrable de valeurs, comme un nombre d'enfants, nombre de voitures, etc
- **Les caractères quantitatifs continus** : peut prendre toute valeur dans un intervalle réel, comme la taille, le poids, la durée)

Il est important de les distinguer car les méthodes de traitements statistiques appliquées aux deux types peuvent différer. Les caractères discrets s'appréhendent en se comptant, alors que les continus se mesurent. Les distinguer permet de faire le choix dans les graphiques et les paramètres statistiques.

3. Paramètres de position
    - Pourquoi existe-t-il plusieurs types de moyenne ?

Il existe plusieurs types de moyennes car un seul type ne saurait suffire à toujours représenter fidèlement une série de données. On retrouve ainsi les moyennes suivantes :

- **Arithmétique** : pour des valeurs homogènes
- **Pondérée** : quand certaines valeurs ont plus d'importance que d'autres
- **Géométrique ou harmonique** : plus spécifique pour certains contextes comme les taux de croissance, les vitesses moyennes, etc.

Chaque moyenne traduit donc une réalité différente selon la structure et la nature des données impliquées.

- Pourquoi calculer une médiane ?

La médiane est utile car elle permet de partager la population en deux groupes égaux en se débarrassant de l'influence des valeurs extrêmes. Elle représente ainsi une tendance centrale « plus robuste » que la moyenne en présence de valeurs « atypiques ».

- Quand est-il possible de calculer un mode ?

Le mode est calculable pour tout type de variable, soit-elle qualitative ou quantitative, à condition qu'une modalité ou valeur apparaisse le plus souvent. Il est particulièrement pertinent pour les variables qualitatives ou les données discrètes.

4. <u>Paramètres de concentration</u>
   - Quel est l'intérêt de la médiale et de l'indice de C. Gini ?

La médiale, ou médiane de la distribution cumulée des revenus, par exemple, mesure la concentration en partageant la distribution en deux moitiés égales. L'indice de Gini quant à lui, quantifie le niveau d'inégalité ou de répartition inégale dans une population. Un indice de Gini proche de 0 indique une égalité parfaite, alors qu'un indice proche de 1 traduit une forte concentration en inégalités, cela illustre par exemple que quelques individus seulement détiennent la majorité des richesses.

5. <u>Paramètres de dispersion</u>
   - Pourquoi calculer une variance à la place de l'écart à la moyenne ? Pourquoi la remplacer par l'écart type ?

La variance mesure la dispersion globale des données autour de la moyenne. L'écart à la moyenne simple quant à lui, se compense, c'est-à-dire que les écarts positifs et négatifs s'annulent. En les élevant au carré, on évite cette compensation. L'écart type est la racine carrée de la variance, ce qui permet de revenir à l'unité d'origine de la variable et ce qui la rend plus interprétable que la variance.

- Pourquoi calculer l'étendue ?

L'étendue, ou valeur max – valeur min, donne une idée rapide de la dispersion totale. Elle est simple à calculer mais sensible aux valeurs extrêmes. Elle complète donc d'autres mesures plus robustes comme l'écart interquartile.

- Pourquoi créer un quantile ? Quel(s) est (sont) le(s) quantile(s) le(s) plus utilisé(s) ?

Les quantiles servent à découper la distribution en parties égales pour analyser la répartition des valeurs. Les plus utilisés sont la médiane (quantile 0,5), les quartiles (Q1, Q2, Q3) et le déciles et centiles selon le niveau de précision souhaité. Ils permettent d'étudier la dispersion, la concentration et les positions relatives dans la distribution.

- Pourquoi construire une boîte de dispersion ? Comment l'interpréter ?

La boîte de dispersion ou boîte à moustache, ou encore boxplot, permet de visualiser la répartition, les quartiles, la médiane et les valeurs extrêmes. Elle aide à : identifier la symétrie ou l'asymétrie de la distribution, repérer les valeurs aberrantes et comparer plusieurs groupes de données.

6. Paramètres de forme
   - Quelle différence faites-vous entre les moments centrés et les moments absolus ? Pourquoi les utiliser ?

- **Les moments centrés** mesurent la distribution autour de la moyenne, en tenant compte du signe.
  - Le moment d'ordre 2 = variance
  - Le moment d'ordre 3 = asymétrie (*skewness*)
  - Le moment d'ordre 4 = aplatissement (*kurtosis*)
- **Les moments absolus** ignorent le signe des écarts. On les utilise pour analyser la forme de la distribution et détecter des asymétries ou concentrations particulières.

- Pourquoi vérifier la symétrie d'une distribution et comment faire ?

La symétrie indique si la moyenne, la médiane et le mode coïncident :
- Si la distribution est symétrique, les trois valeurs sont proches
- Si elle est asymétrique à droite, la moyenne > médiane > mode
- Si elle est asymétrique à gauche, moyenne < médiane < mode.

On peut la vérifier visuellement, grâce à un histogramme ou encore une boîte de dispersion, mais également numériquement, par le calcul du coefficient d'asymétrie.

Mise en œuvre avec Python.

Etapes 1 à 3 : Introduction

Avant toute manipulation, il est nécessaire de préparer l'environnement de travail. Il faut ainsi commencer par créer, dans le dossier src, un sous-dossier intitulé data, dans lequel on place le fichier *resultats-elections-presidentielles-2022-1er-tour.csv* provenant du GitHub de la séance 3. Dans le même dossier src, on ajoute également le fichier *main.py* de la séance, puis on l'ouvre dans notre éditeur de code, ici *Notepad++.*

Enfin, avant d'exécuter notre script, il convient de se positionner dans le bon répertoire via le terminal. Il suffit ici, depuis le dossier src d'effectuer un clic droit et de sélectionner « ouvrir dans le terminal ». Cette manipulation nous situe automatiquement au bon endroit :

```
PS C:\Users\33664\Desktop\Sorbonne U\S1\Blanc-2025-2026-Analyse-de-donnees\Seance-3\src>
```

Etapes 4 à 10 : Codage

La première étape de programmation consiste à lire le fichier CSV à l'aide de la méthode *read_csv()* de *Pandas*, intégrée à une instruction *with*, afin d'assurer une bonne gestion du fichier (étape 4). On sélectionne ensuite les colonnes quantitatives utiles à nos analyses statistiques : inscrits, votants, blancs, nuls, exprimés, abstentions :

```
# Etape 4 – Lire le CSV des résultats (with + détection du séparateur) et sélectionner colonnes quantitatives
with open(os.path.join(DATA_DIR, "resultats-elections-presidentielles-2022-1er-tour.csv"), "r", encoding="utf-8") as f:
    contenu = pd.read_csv(f, sep=None, engine="python")
colonnes_quanti = [« Inscrits », « Votants », « Blancs », « Nuls », « Exprimés », « Abstentions »]
num = contenu[colonnes_quanti].copy()
# Définir cols pour les boucles suivantes
cols = num.columns
```

Cette première lecture du fichier permet de constituer une base de données propre contenant uniquement les variables quantitatives nécessaires à la suite des calculs.

Nous calculons maintenant plusieurs paramètres statistiques fondamentaux pour chacune des colonnes sélectionnées (étape 5) : la moyenne, la médiane, le mode, l'écart-type, l'écart absolu moyen, et l'étendue. Voici maintenant l'extrait de code correspondant :

```
# Etape 5 - Calculer moyennes, médianes, modes, écart-type, écart absolu à la moyenne, étendue
moyennes = num.mean().round(2)
medianes = num.median().round(2)
modes = num.mode().iloc[0].round(2)                    # première ligne = mode principal
ecarts_type = num.std(ddof=0).round(2)                 # ddof=0 -> population
ecarts_abs_moy = num.apply(lambda s: np.abs(s - s.mean()).mean()).round(2)  # écart absolu moyen
etendues = (num.max() - num.min()).round(2)            # étendue = max - min

# (affichage facultatif pour contrôle)
print("\nMoyennes :\n", moyennes)
print("\nMédianes :\n", medianes)
print("\nModes :\n", modes)
print("\nÉcarts type :\n", ecarts_type)
print("\nÉcarts absolus moyens :\n", ecarts_abs_moy)
print("\nÉtendues :\n", etendues)
```

Une fois ces lignes exécutées, le terminal affiche les résultats suivants :

```
Moyennes :
 Inscrits       455587.63
Votants         335735.58
Blancs           5080.46
Nuls             2309.82
Exprimés        328345.30
Abstentions     119852.05

Médianes :
 Inscrits       366859.0
Votants         274372.0
Blancs           4001.0
Nuls             2039.0
Exprimés        268568.0
Abstentions      95369.0

Modes :
 Inscrits       5045.0
Votants          2773.0
Blancs          4577.0
Nuls             17.0
Exprimés         2701.0
Abstentions     2272.0

Écarts type :
 Inscrits       349359.73
Votants         257183.52
Blancs           3476.17
Nuls             1494.35
Exprimés        252570.01
Abstentions     116469.70

Étendues :
 Inscrits       1808861.0
Votants         1297100.0
Blancs           17389.0
Nuls             8236.0
Exprimés        1272080.0
Abstentions      929183.0
```

Ensuite, nous souhaitons présenter ces résultats de manière claire dans le terminal (étape 6), nous allons alors faire tourner le code suivant :

```
# Etape 6 - Afficher la liste des paramètres
print("\n--- Paramètres (par colonne quantitative) ---\n")
for c in cols:
    print(f"{c} : Moyenne={moyennes[c]}, Médiane={medianes[c]}, Mode={modes[c]}, "
```

```
        f"Écart-type={ecarts_type[c]}, Écart abs. moy.={ecarts_abs_moy[c]}, Étendue={etendues[c]}")
```

Une fois lancé, le résultat suivant s'affiche dans notre terminal :

```
--- Paramètres (par colonne quantitative) ---

Inscrits : Moyenne=455587.63, Médiane=366859.0, Mode=5045.0, Écart-type=349359.73, Écart abs. moy.=272240.72, Étendue=18
08861.0
Votants : Moyenne=335735.58, Médiane=274372.0, Mode=2773.0, Écart-type=257183.52, Écart abs. moy.=201517.17, Étendue=129
7100.0
Blancs : Moyenne=5080.46, Médiane=4001.0, Mode=4577.0, Écart-type=3476.17, Écart abs. moy.=2817.95, Étendue=17389.0
Nuls : Moyenne=2309.82, Médiane=2039.0, Mode=17.0, Écart-type=1494.35, Écart abs. moy.=1131.99, Étendue=8236.0
Exprimés : Moyenne=328345.3, Médiane=268568.0, Mode=2701.0, Écart-type=252570.01, Écart abs. moy.=197762.2, Étendue=1272
080.0
Abstentions : Moyenne=119852.05, Médiane=95369.0, Mode=2272.0, Écart-type=116469.7, Écart abs. moy.=74959.07, Étendue=92
9183.0
```

Il nous est maintenant demandé de calculer la distance interquartile (IQR) et la distance interdécile (IDR) à l'aide de la méthode quantile() (étape 7). Cela va permettre d'affiner la mesure de la dispersion et nous allons utiliser le code suivant :

```
# Etape 7 - Calculer IQR et interdécile (avec quantile())
q1 = num.quantile(0.25)
q3 = num.quantile(0.75)
d1 = num.quantile(0.10)
d9 = num.quantile(0.90)

iqr = (q3 - q1).round(2)
idr = (d9 - d1).round(2)

print("\n--- IQR et Interdécile (par colonne) ---\n")
for c in cols:
    print(f"{c} : IQR={iqr[c]}, Interdécile={idr[c]}")
```

Nous pouvons exécuter ce programme dans le terminal, ce qui nous donnera le résultat suivant :

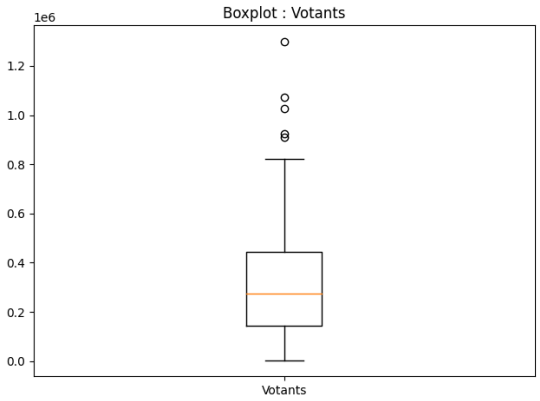
```
--- IQR et Interdécile (par colonne) ---

Inscrits : IQR=401050.0, Interdécile=793988.8
Votants : IQR=301770.5, Interdécile=602687.2
Blancs : IQR=4852.5, Interdécile=8845.8
Nuls : IQR=1917.0, Interdécile=3240.6
Exprimés : IQR=296870.5, Interdécile=590169.2
Abstentions : IQR=106489.0, Interdécile=193676.2
```

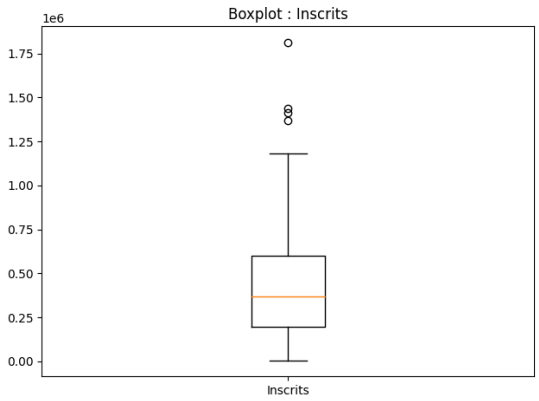
Dans cette prochaine étape, nous avons pour objectif de réaliser une boîte à moustache, ou boxplot, pour chaque variable quantitative (étape 8). Cette représentation permettra de visualiser la répartition des valeurs et de détecter les éventuelles valeurs extrêmes. Nous pouvons utiliser les lignes de code suivantes :

```
# Etape 8 - Boîte à moustache par colonne quantitative
for col in cols:
```

```
    plt.figure()
    plt.title(f"Boxplot : {col}")
    plt.boxplot(num[col].dropna(), labels=[col])
    plt.tight_layout()
    safe_name = col.replace(" ", "_").replace("/", "_")
    plt.savefig(os.path.join(IMG_DIR, f"boxplot_{safe_name}.png"))
    plt.close()
print(f"\nBoxplots sauvegardés dans '{IMG_DIR}/'")
```

Une fois exécuté, ce code à bien créé un dossier image intitulé « img » dans lequel sont stockées les boîtes à moustache, dont voici deux exemples :





Pour cette nouvelle étape, il convient tout d'abord d'importer un nouveau fichier, le fichier island-index.csv (étape 9). Il contient les informations sur les îles du monde, comme leur nom, leur surface, leur localisation, etc. pour lire ce dossier, nous allons utiliser la formule suivante :

```
# Etape 9 - Lire island-index.csv
island_path = os.path.join(DATA_DIR, "island-index.csv")
with open(island_path, "r", encoding="utf-8") as f:
    islands = pd.read_csv(f, sep=None, engine="python")
```

Il nous est ensuite demandé de sélectionner la colonne « Surface (km²) » pour classer les îles dans des intervalles de taille croissantes : 8 intervalles, allant de 0 à supérieur à 10 000 km2 (étape 10). Nous allons ici nous aider de la fonction pd.cut(). Le code suivant permet d'effectuer ce classement et d'en dénombrer les effectifs :

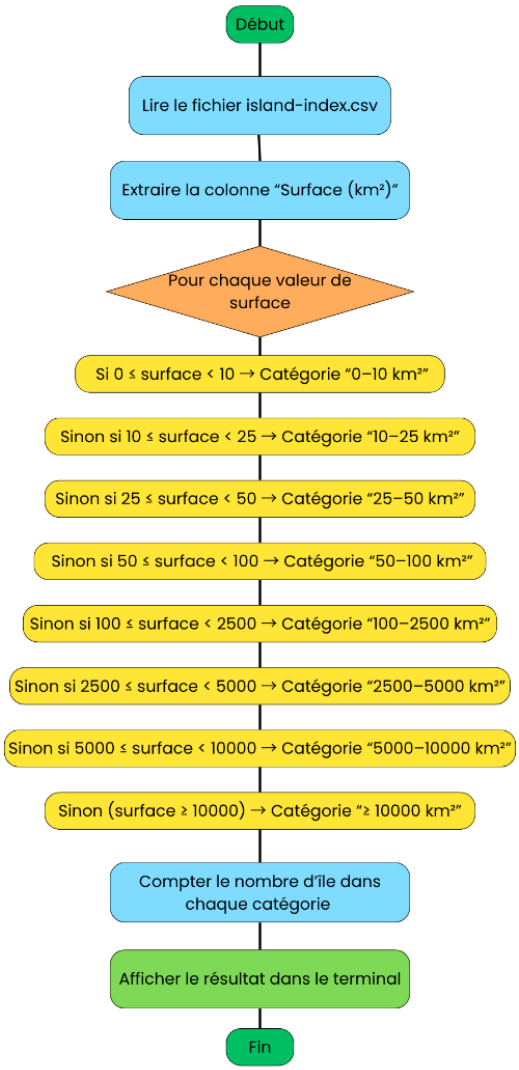
```
# Etape 10 - Sélectionner la colonne 'Surface (km2)' et catégoriser selon les intervalles demandés
col_name = next((c for c in islands.columns if "Surface" in c and "km" in c), None)
if col_name is None:
    raise ValueError("Colonne 'Surface (km2)' introuvable dans island-index.csv")

surface = pd.to_numeric(islands[col_name], errors="coerce")  # convertir proprement en float

bins = [0, 10, 25, 50, 100, 2500, 5000, 10000, np.inf]
labels = [
```

```
    "0-10", "10-25", "25-50", "50-100",
    "100-2500", "2500-5000", "5000-10000", ">=10000"
]
cats = pd.cut(surface, bins=bins, labels=labels, right=True, include_lowest=True)
counts = cats.value_counts().reindex(labels).fillna(0).astype(int)

print("\n--- Décompte des îles par intervalle de surface (km²) ---\n")
print(counts)
```

Une fois enregistré et le script joué dans le terminal, on obtient le résultat suivant :

```
--- Décompte des îles par intervalle de surface (km²) ---

0-10          78423
10-25          2327
25-50          1164
50-100          788
100-2500       1346
2500-5000        60
5000-10000       40
>=10000          71
```

Voici un organigramme pour illustrer ma solution à la dernière question :



Etape 11 : Bonus

Dans cette dernière étape bonus nous cherchons à exporter les listes calculées en deux formats (étape 11) : un fichier CSV et un fichier Excel, ils seront stockés dans un dossier intitulé « exports ». Ainsi, nous allons utiliser le script suivant :

```
# Etape 11 Bonus - Sortie des paramètres statistiques au format CSV et Excel
parametres = pd.DataFrame({
    "Moyenne": moyennes,
    "Médiane": medianes,
    "Mode": modes,
    "Écart type": ecarts_type,
    "Écart absolu moyen": ecarts_abs_moy,
    "Étendue": etendues,
    "IQR": iqr,
    "IDR": idr
}, index=num.columns)

os.makedirs("exports", exist_ok=True)

parametres.to_csv("exports/parametres_statistiques.csv", sep=";", encoding="utf-8")
parametres.to_excel("exports/parametres_statistiques.xlsx")

print("→ Exports réalisés dans le dossier /exports")
```

Un dossier « export » ainsi que nos deux fichiers ont bien été créés dans le dossier src de la Séance-03 et le terminal affiche ce message de confirmation :

```
→ Exports réalisés dans le dossier /exports
```

Ces fichiers contiennent donc l'ensemble des paramètres statistiques calculés, permettant une réutilisation ou une présentation ultérieure sous forme tabulaire.

Séance 4

Objectifs

- Savoir afficher une distribution statistique. Ce savoir est utilisé pour comparer une distribution observée avec une distribution théorique.

Questions de cours

1. Quels critères mettriez-vous en avant pour choisir entre une distribution statistique avec des variables discrètes et une distribution statistique avec des variables continues ?

Il est possible de mettre en avant plusieurs critères pour choisir entre une distribution à variables discrètes et une distribution à variables continues.
Le premier critère repose sur la nature de la variable : Si elle ne peut prendre que des valeurs entières et dénombrables, comme le nombre d'habitants, de naissances, d'occurrences d'un événement etc, il s'agit d'une variable discrète. On utilisera alors des lois telles que Bernoulli, binomiale ou Poisson, adaptées aux phénomènes de comptage. Si au contraire la variable peut prendre toute valeur dans un intervalle continu, comme l'altitude, le revenu, la température, la distance, etc, elle est continue et se modélise par des lois comme la normale, la log-normale, l'exponentielle ou la Weibull.

Le second critère tient à la forme empirique de la distribution. Une distribution en « marches » ou en valeurs isolées indique une variable discrète, tandis qu'une densité fluide et continue correspond à une variable continue.

Enfin, le choix du modèle dépend aussi du processus génératif du phénomène : les phénomènes résultant d'un grand nombre d'effets indépendants suivent souvent la loi normale, ceux issus de processus multiplicatifs la log-normale, et les comptages d'événements rares la Poisson. L'échelle d'observation et la précision des données peuvent également influer sur ce choix, comme une mesure arrondie pouvant rendre une variable continue apparemment discrète.

2. Expliquez selon vous quelles sont les lois les plus utilisées en géographie ?

En géographie, plusieurs lois statistiques sont particulièrement mobilisées pour décrire la répartition et la dynamique des phénomènes spatiaux. Parmi les plus courantes il est possible de retrouver :

- **La loi de Zipf ou loi rang-taille** : elle est utilisée pour représenter la hiérarchie urbaine et la répartition des tailles de villes. Elle met en évidence la structure inégale du système urbain.
- **La loi log-normale ou loi de Gibrat** : elle s'applique aux phénomènes de croissance proportionnelle, comme l'évolution des populations ou des revenus.
- **La loi de Pareto** : elle est typique des distributions « à queue lourde », décrit les fortes inégalités ou les structures spatiales où quelques grandes entités concentrant une part importante des ressources.

- **La loi normale ou de Gauss** : elle est très répandue pour les phénomènes aléatoires résultant d'une multitude de petits effets indépendants, comme par exemple les erreurs de mesure, variations naturelles, etc.
- **La loi de Poisson**, utilisée pour modéliser les comptages d'événements rares et indépendants dans l'espace, comme par exemple la répartition d'accidents, de points d'intérêt, etc.

Ces lois sont essentielles en géographie car elles permettent de relier les formes observées : répartition, hiérarchie, concentration, aux mécanismes sous-jacents des phénomènes spatiaux.

Mise en œuvre avec Python

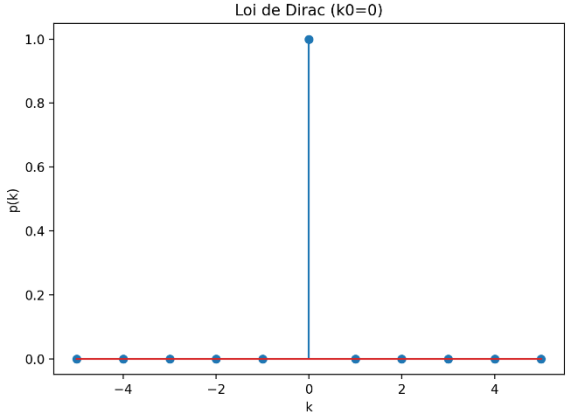
Avant de commencer toute manipulation avec Python, il convient d'importer les bibliothèques nécessaires, ici le dossier src de la séance, disponible sur GitHub. Il nous faut également créer un dossier prêt à réceptionner les représentations graphiques que nous allons générer (étape 1). Pour ce faire nous utiliserons le code suivant :

```
# Etape 1 - Préparation et configuration
IMG_DIR = "img_seance4"
os.makedirs(IMG_DIR, exist_ok=True)
```

Ensuite, l'objectif est d'abord de représenter les distributions statistiques de variables discrètes suivantes : Dirac, uniforme discrète, binomiale, Poisson et Zipf-Mandelbrot (étape 2). Pour la loi de Dirac par exemple, nous pouvons utiliser le programme suivant :

```
# Etape 2 - Lois discrètes
def plot_dirac(k0=0, kmin=None, kmax=None, save="dirac.png"):
    if kmin is None: kmin = k0 - 5
    if kmax is None: kmax = k0 + 5
    k = np.arange(kmin, kmax + 1)
    pmf = (k == k0).astype(int)
    fig = plt.figure()
    plt.stem(k, pmf, use_line_collection=True)
    plt.title(f"Loi de Dirac (k0={k0})")
    plt.xlabel("k")
    plt.ylabel("p(k)")
    save_fig(fig, save)
```

Voici le résultat obtenu dans notre dossier « img_seance4 » :

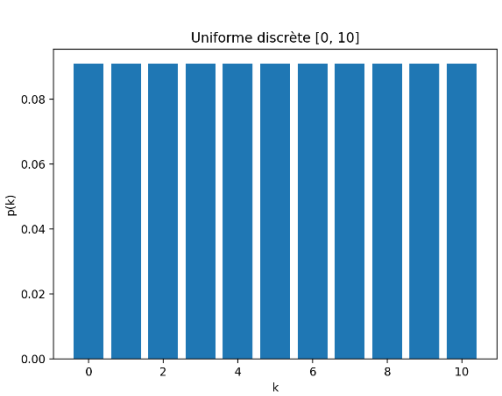


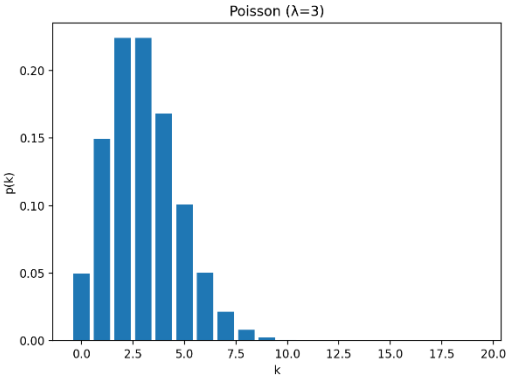
Nous allons pouvoir ainsi coder les autres distributions statistiques de manière similaire :

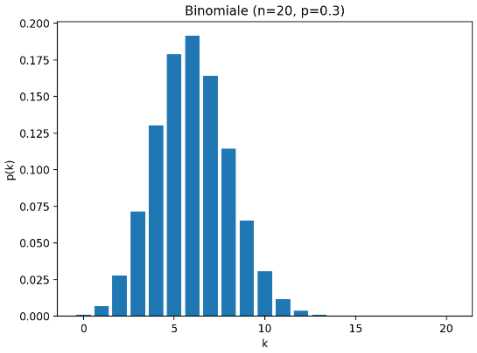
```
# Loi uniforme discrète
def plot_uniform_discrete(a=0, b=10, save="uniform_discrete.png"):
    k = np.arange(a, b + 1)
    pmf = np.ones_like(k) / len(k)
    fig = plt.figure()
    plt.bar(k, pmf)
    plt.title(f"Uniforme discrète [{a}, {b}]")
    plt.xlabel("k")
    plt.ylabel("p(k)")
    save_fig(fig, save)

# Loi binomiale
def plot_binomiale(n=20, p=0.3, save="binomiale.png"):
    k = np.arange(0, n + 1)
    dist = scipy.stats.binom(n, p)
    fig = plt.figure()
    plt.bar(k, dist.pmf(k))
    plt.title(f"Binomiale (n={n}, p={p})")
    plt.xlabel("k")
    plt.ylabel("p(k)")
    save_fig(fig, save)

# Loi de Poisson
def plot_poisson(mu=3, save="poisson.png"):
    k = np.arange(0, 20)
    dist = scipy.stats.poisson(mu)
    fig = plt.figure()
    plt.bar(k, dist.pmf(k))
    plt.title(f"Poisson (λ={mu})")
    plt.xlabel("k")
    plt.ylabel("p(k)")
    save_fig(fig, save)

# Loi de Zipf-Mandelbrot
def zipf_mandelbrot_pmf(s=1.5, q=1.0, kmax=100):
    k = np.arange(1, kmax + 1)
    probs = (k + q) ** (-s)
    probs /= probs.sum()
    return k, probs

def plot_zipf_mandelbrot(s=1.5, q=1.0, kmax=100, save="zipf_mandelbrot.png"):
    k, pmf = zipf_mandelbrot_pmf(s, q, kmax)
    fig = plt.figure()
    plt.bar(k, pmf)
    plt.title(f"Zipf-Mandelbrot (s={s}, q={q})")
    plt.xlabel("k")
```

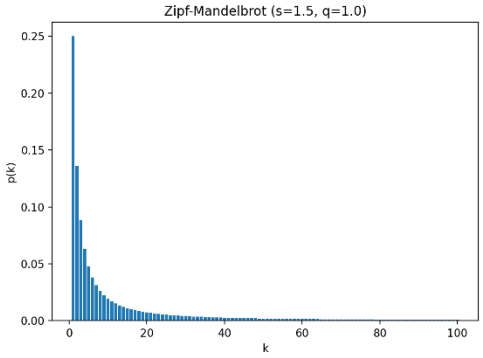
```
    plt.ylabel("p(k)")
    save_fig(fig, save)
```

Une fois cela exécuté, voici les résultats graphiques obtenus :







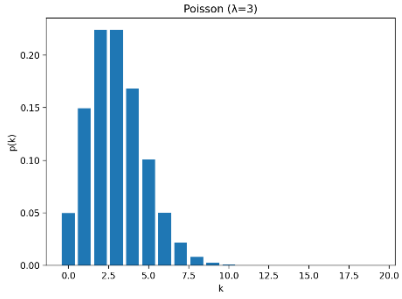


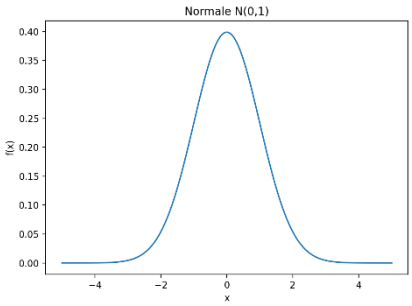
On souhaite maintenant obtenir les distributions statistiques des variables continues : loi de Poisson, loi normale, loi log-normale, loi uniforme, loi du χ 2 et loi de Pareto (étape 3). Nous allons ainsi exécuter le programme suivant :

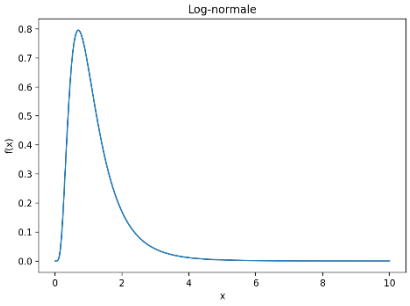
```
# Etape 3 - Lois continues

def plot_pdf(frozen, x, title, save):
    fig = plt.figure()
    plt.plot(x, frozen.pdf(x))
    plt.title(title)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    save_fig(fig, save)

# Lois continues principales
def plot_continues():
    # Poisson (en PMF, même si discrète)
    plot_poisson(3, save="poisson_continu.png")

    # Loi normale
    x = np.linspace(-5, 5, 400)
```

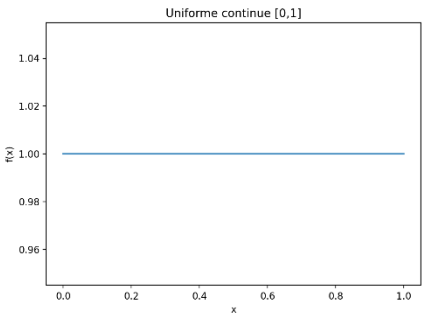
```
plot_pdf(scipy.stats.norm(0, 1), x, "Normale N(0,1)", "normale.png")

# Loi log-normale
x = np.linspace(0.01, 10, 400)
plot_pdf(scipy.stats.lognorm(s=0.6), x, "Log-normale", "lognormale.png")

# Loi uniforme continue
x = np.linspace(0, 1, 200)
plot_pdf(scipy.stats.uniform(0, 1), x, "Uniforme continue [0,1]", "uniforme_continue.png")

# Loi du chi²
x = np.linspace(0, 20, 400)
plot_pdf(scipy.stats.chi2(4), x, "Chi² (df=4)", "chi2.png")

# Loi de Pareto
x = np.linspace(1, 10, 400)
plot_pdf(scipy.stats.pareto(2.5), x, "Pareto (b=2.5)", "pareto.png")
```

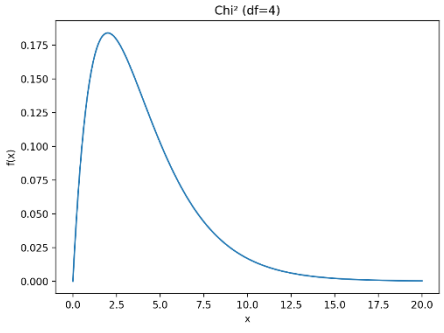
Ces lignes de code nous génèrent les représentations graphiques qui affichent la fonction de densité correspondante :

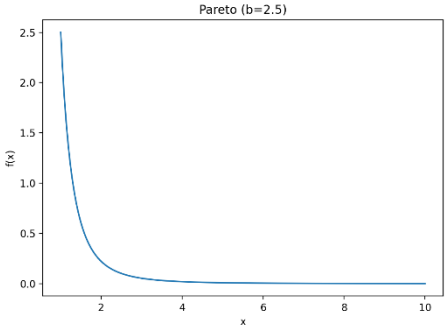












Enfin, nous souhaitons, à l'aide de fonctions informatiques, calculer la moyenne et l'écart-type de chacune de ces distributions (étape 4 et 5). Nous allons alors utiliser les codes ci-dessous pour les calculer et les exécuter :

```
# Etape 4 - Fonctions moyenne et écart-type

def mean_std_from_frozen(dist):
    return float(dist.mean()), float(dist.std())
```

```
def mean_std_from_pmf(k, pmf):
    m = (k * pmf).sum()
    v = ((k - m) ** 2 * pmf).sum()
    return float(m), float(np.sqrt(v))

# Etape 5 - Exécution principale
if __name__ == '__main__':

    # Lois discrètes
    plot_dirac()
    plot_uniform_discrete()
    plot_binomiale()
    plot_poisson()
    plot_zipf_mandelbrot()

    # Lois continues
    plot_continues()

    # Calculs de moyenne / écart-type exemples
    print("\n--- Moyennes et écarts-types ---\n")
    print("Dirac(0) :", mean_std_from_pmf(np.array([0]), np.array([1])))
    print("Uniforme discrète 0..10 :", mean_std_from_pmf(np.arange(0,11), np.ones(11)/11))
    print("Binomiale(20,0.3) :", mean_std_from_frozen(scipy.stats.binom(20,0.3)))
    print("Poisson(3) :", mean_std_from_frozen(scipy.stats.poisson(3)))
    k, pmf = zipf_mandelbrot_pmf()
    print("Zipf-Mandelbrot :", mean_std_from_pmf(k, pmf))
    print("Normale(0,1) :", mean_std_from_frozen(scipy.stats.norm(0,1)))
    print("Log-normale :", mean_std_from_frozen(scipy.stats.lognorm(0.6)))
    print("Pareto(2.5) :", mean_std_from_frozen(scipy.stats.pareto(2.5)))

    print(f"\nImages sauvegardées dans '{IMG_DIR}'")
```

Ainsi, l'exécution du programme principal nous affiche le résultat suivant dans le terminal :

```
--- Moyennes et écarts-types ---

Dirac(0) : (0.0, 0.0)
Uniforme discrète 0..10 : (5.0, 3.1622776601683795)
Binomiale(20,0.3) : (6.0, 2.0493901531919194)
Poisson(3) : (3.0, 1.7320508075688772)
Zipf-Mandelbrot : (11.511222116286445, 18.02179592797163)
Normale(0,1) : (0.0, 1.0)
Log-normale : (1.1972173631218102, 0.7881013869316227)
Pareto(2.5) : (1.6666666666666667, 1.4907119849998598)
```

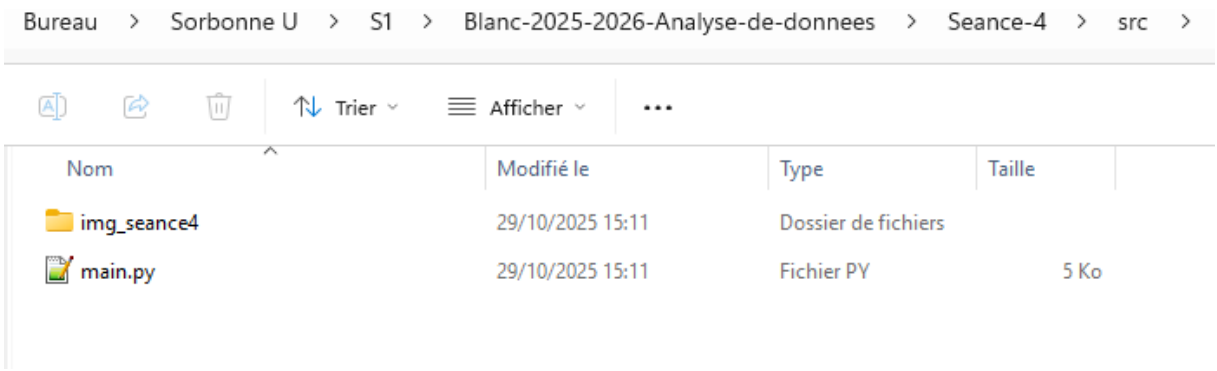
Sans rentrer dans l'analyse, ces valeurs nous confirment tout de même les propriétés théoriques des lois que nous utilisons depuis le début de la séance. La Dirac est centrée en 0 avec écart-type nul,

l’uniforme discrète présente une moyenne égale au centre de son intervalle, les distributions de Poisson et binomiale respectent leurs définitions (moyenne = λ ou n.p) et les lois continues affichent bien les valeurs attendues pour leurs paramètres standards.

Pour finir, le terminal affiche la phrase suivante :

```
Images sauvegardées dans 'img_seance4'
```

Il nous indique et nous rassure ainsi sur l’emplacement des manifestations graphiques visuelles de nos lignes de codes. Cela confirme bien que le dossier, ainsi que les graphiques, se sont générés automatiquement.



Séance 5

Objectifs

- Manipuler harmonieusement les fonctions natives avec les méthodes Pandas
- Comprendre les trois théories permettant de valider un résultat en analyse de données

Questions de cours

1. Comment définir l'échantillonnage. Pourquoi ne pas utiliser la population en entier ? Quelles sont les méthodes d'échantillonnage ? Comment les choisir ?

L'échantillonnage consiste à prélever une partie représentative d'une population mère, afin d'en déduire des informations sur l'ensemble. Il n'est pas toujours possible d'utiliser une population entière dans une étude exhaustive car cela peut être matériellement impossible ou trop coûteux.
L'objectif est de généraliser les résultats de l'échantillon à la population tout en maîtrisant l'erreur d'échantillonnage. On distingue ainsi deux grandes familles :

- **Méthodes aléatoires** : tirage au sort, simple, avec ou sans remise. Elles garantissent une représentativité et permettent l'application rigoureuse des lois statistiques.
- **Méthodes non aléatoires** : échantillonnage systématique, méthode des quotas ou méthodes de type Monte Carlo, adaptées aux situations où la base de sondage n'est pas complète.

Le choix de la méthode dépend du but de l'étude, du coût, de la taille de la population, ou encore du niveau de précision souhaité.

2. Comment définir un estimateur et une estimation ?

Un estimateur est une fonction mathématique des données observées permettant d'évaluer un paramètre inconnu de la population comme une moyenne, une variance, une proportion *etc*. L'estimation, quant à elle, est la valeur numérique obtenue à partir de l'échantillon. Par exemple, la moyenne de l'échantillon $\bar{X}$ est un estimateur de la moyenne de la population μ, et la valeur calculée $\bar{x}$ est une estimation.

3. Comment distingueriez-vous l'intervalle de fluctuation et l'intervalle de confiance ?

L'intervalle de fluctuation correspond à la zone dans laquelle la fréquence observée est susceptible de se situer, lorsque la proportion théorique p est connue. Il s'agit d'un outil d'échantillonnage : il mesure les variations attendues dues au hasard.
L'intervalle de confiance, de son côté, correspond à l'estimation du paramètre inconnu, comme μ ou p, avec un niveau de certitude donné. Il s'agit d'un outil d'estimation.
Ainsi, le premier est calculé à partir d'un paramètre connu, le second sert à estimer un paramètre inconnu.

4. Qu'est-ce qu'un biais dans la théorie de l'estimation ?

Le biais d'un estimateur est la différence entre l'espérance mathématique de l'estimateur et la vraie valeur du paramètre. Un estimateur est « sans biais » si l'équation suivante se vérifie :

$$E(\hat{\theta}) - \theta = 0$$

Si ce n'est pas le cas, alors il est « biaisé ». Le biais traduit une erreur systématique dans l'estimation. Pour qu'un estimateur soit bon, il doit être sans biais, convergent et de variance minimale.

5. Comment appelle-t-on une statistique travaillant sur la population totale ? Faites le lien avec la notion de données massives 1 ?

Lorsqu'une statistique porte sur l'ensemble de la population, on parle de statistique exhaustive. Elle ne repose alors plus sur des inférences, mais sur une mesure complète des paramètres. Avec les données massives ou big data, une tendance se rapprochant de cette logique émerge : les bases de données géantes permettent souvent une observation quasi exhaustive, réduisant la part d'inférence au profit de l'analyse descriptive directe.

6. Quels sont les enjeux autour du choix d'un estimateur ?

Lors du choix d'un estimateur il est important d'opérer un arbitrage entre :
- La fidélité, ou absence de biais
- La précision, ou faible variance
- La robustesse, ou résistance aux valeurs aberrantes
- L'efficacité, ou minimisation de l'erreur quadratique moyenne

Selon les théorèmes de Rao-Blackwell et Lehmann-Scheffé un estimateur sans biais et de variance minimale est considéré comme optimal.

7. Quelles sont les méthodes d'estimation d'un paramètre ? Comment en sélectionner une ?

Il existe deux méthodes principales pour estimer un paramètre :
- **L'estimation ponctuelle** : une seule valeur estimée du paramètre, la moyenne de l'échantillon par exemple.
- **L'estimation par intervalle** : une plage de valeurs, ou intervalle de confiance, dans laquelle le paramètre a une forte probabilité de se trouver.

Les méthodes peuvent ainsi être par vraisemblance, par moments, ou encore Bayésienne, c'est-à-dire incorporant une connaissance a priori. Le choix dépend du modèle probabiliste, de la nature du paramètre et de la disponibilité des données.

8. Quels sont les tests statistiques existants ? À quoi servent-ils ? Comment créer un test ?

Les tests statistiques permettent de vérifier la validité d'une hypothèse sur un paramètre de la population. Ils servent à décider, avec un risque d'erreur maîtrisé, si une observation confirme ou infirme une hypothèse. On en retrouve plusieurs types, d'un côté les tests paramétriques (Student, Khi-

deux, Fisher, *etc*.), ils supposent une loi de distribution continue. De l'autre côté, l'on retrouve les tests non paramétrique, utilisés quand la distribution n'est pas connue. La création d'un test suit d'autre part la procédure suivante :

1) Formulation des hypothèses H_0(nulle) et H_1(alternative)
2) Choix d'un statistique test
3) Détermination du seuil de signification α
4) Décision selon la valeur observée

9. Que pensez-vous des critiques de la statistique inférentielle ?

La statistique inférentielle est l'objet de différentes critiques pour différentes raisons :

- Sa dépendance aux hypothèses théoriques comme la normalité, l'indépendance, l'homogénéité des variances, souvent non vérifiées en pratique
- Sa sensibilité aux biais d'échantillonnage ou aux valeurs aberrantes
- Ses interprétations abusives des seuils de signification
- Et enfin, à l'ère du big data, certains estiment que l'inférence devient moins pertinente alors qu'il est maintenant possible d'observer une population statistique entière.

Cependant, l'inférence statistique demeure une méthode clé pour tirer des conclusions généralisables à partir d'informations partielles. Ces critiques ne sauraient ainsi remettre en cause cet avantage scientifique que l'inférence a de son côté. Les possibilités qu'elle offre en termes de reproductibilité de la statistique restent tout de même intéressantes. A mon sens, ces critiques devraient mener à une réflexivité et une lecture critique lors de l'analyse des résultats de l'inférence, plutôt qu'à son complet abandon. Tout repose maintenant plus dans l'utilisation fait de cette méthode que dans si oui ou non il est pertinent de l'appliquer.

Mise en œuvre avec Python

Avant de lancer tout programme, il convient, comme pour les autres séances, de placer les fichiers CSV nécessaires dans le dossier data, puis d'ouvrir le fichier main.py. Comme d'habitude, il convient également de se positionner dans le répertoire src via le terminal pour pouvoir exécuter correctement le script. Une fois tout en place, lançons l'analyse.

Partie 1 : Théorie de l'échantillonnage

La première partie du script va nous permettre une analyse de 100 échantillons simulés afin de comprendre comment varient les proportions observées d'un échantillon à l'autre. Cela va permettre de vérifier si les résultats restent cohérents avec la population mère.
Notre première étape consiste alors à charger les données et afficher les titres souhaités (étape 1.0), notre script affiche ainsi :

```
print("Résultat sur le calcul d'un intervalle de fluctuation")
donnees = pd.DataFrame(ouvrirUnFichier("./data/Echantillonnage-100-Echantillons.csv"))
```

Et dès le lancement, le terminal nous annonce le début du calcul :

```
Résultat sur le calcul d'un intervalle de fluctuation
```

A partir de là, nous pouvons nous pencher dans un premier temps sur notre calcul de la moyenne des effectifs observés sur l'ensemble des échantillons (étape 1.1). Dans un souci de clarté dans le terminal, nous voulons également un titre de partie mis en évidence (étape 1.0). Notre script python affiche :

```
#Etape 1.0 - Théorie de l'échantillonnage
print("\n" + "-"*60)
print("Partie 1 — Théorie de l'échantillonnage (intervalles de fluctuation)")
print("-"*60 + "\n")

#Etape 1.1 - Calcul des moyennes par colonnes
moyennes = donnees.mean().round(0)
print("\nMoyennes observées sur 100 échantillons :\n", moyennes)
```

Le résultat dans le terminal sera le suivant :

```
------------------------------------------------------------
Partie 1 — Théorie de l'échantillonnage (intervalles de fluctuation)
------------------------------------------------------------

Moyennes observées sur 100 échantillons :
 Pour           391.0
Contre          416.0
Sans opinion    193.0
dtype: float64
```

Cela nous permet alors de visualiser la structure moyenne des échantillons.
Nous voulons maintenant comparer les moyennes avec la population mère (étape 1.2). Pour cela, nous emploierons les lignes de code suivantes :

```
#Etape 1.2 - Calcul des fréquences observées dans échantillons
somme_moyennes = moyennes.sum()
frequences_echant = (moyennes / somme_moyennes).round(2)
print("\nFréquences observées dans les échantillons :\n", frequences_echant)
```

Pour le résultat suivant dans notre terminal :

```
Fréquences observées dans les échantillons :
 Pour           0.39
Contre          0.42
Sans opinion    0.19
dtype: float64
```

Ces valeurs apparaissent très proches de celles de la population réelle, bon indicateur pour la suite de nos analyses

Dans un second temps, nous allons pouvoir effectuer un rappel des fréquences réelles de la population (étape 1.3). Pour ce faire, nous allons exécuter le code suivant :

```
#Etape 1.3 - Fréquences réelle population mère
pop_totale = 2185
pop_pour, pop_contre, pop_sans = 852, 911, 422
frequences_reelles = pd.Series({
    "Pour": round(pop_pour / pop_totale, 2),
    "Contre": round(pop_contre / pop_totale, 2),
    "Sans opinion": round(pop_sans / pop_totale, 2)
})
print("\nFréquences réelles de la population mère :\n", frequences_reelles)
```

Le terminal nous propose le résultat suivant et nous allons voir que cette simulation reflète bien la population :

```
Fréquences réelles de la population mère :
 Pour          0.39
Contre         0.42
Sans opinion    0.19
dtype: float64
```

Nous voulons ensuite calculer des intervalles de fluctuation à 95% (étape 1.4). C'est dans ces intervalles que devraient se situer les fréquences issues d'un échantillon. Le script suivant va nous permettre ce calcul :

```
#Etape 1.4 - Calcul intervalles fluctuation
zC = 1.96
n = len(donnees)

intervalle_fluctuation = {}

for cat, p_obs in frequences_echant.items():
    marge = zC * math.sqrt((p_obs * (1 - p_obs)) / n)
    borne_inf = round(p_obs - marge, 3)
    borne_sup = round(p_obs + marge, 3)
    intervalle_fluctuation[cat] = (borne_inf, borne_sup)

print("\nIntervalles de fluctuation à 95 % :\n")
for cat, (inf, sup) in intervalle_fluctuation.items():
    print(f"{cat} : [{inf}, {sup}]")
```

Le terminal nous affiche les résultats recherchés : des intervalles qui indiquent ce que l'on peut attendre d'un échantillonnage à répétition.

```
Intervalles de fluctuation à 95 % :

Pour : [0.294, 0.486]
Contre : [0.323, 0.517]
Sans opinion : [0.113, 0.267]
```

Nous cherchons maintenant à vérifier la cohérence avec la population. Il s'agit de savoir si les valeurs réelles de la population mère se situent bien dans nos intervalles (étape 1.5). Le script suivant a été utilisé :

```
#Etape 1.5 - Comparaison fréquences observée et réelles
print("\nComparaison avec les valeurs réelles :\n")
for cat in frequences_reelles.index:
    fr_real = frequences_reelles[cat]
    inf, sup = intervalle_fluctuation[cat]
    if inf <= fr_real <= sup:
        conclusion = "La fréquence réelle est comprise dans l'intervalle"
    else:
        conclusion = "La fréquence réelle est en dehors de l'intervalle"
    print(f"{cat} : fréquence réelle = {fr_real} → {conclusion}")
```

Le terminal nous affiche le résultat suivant, tout en nous rassurant sur la cohérence de notre analyse. Ainsi, les échantillons simulés reflètent bien la population.

```
Comparaison avec les valeurs réelles :

Pour : fréquence réelle = 0.39 → La fréquence réelle est comprise dans l'intervalle
Contre : fréquence réelle = 0.42 → La fréquence réelle est comprise dans l'intervalle
Sans opinion : fréquence réelle = 0.19 → La fréquence réelle est comprise dans l'intervalle
```

Partie 2 : Théorie de l'estimation, intervalles de confiance

Dans cette deuxième partie portée sur les intervalles de confiance, nous n'allons plus répéter notre analyse sur 100 échantillons, mais proposer de se concentrer sur un seul échantillon. On cherche alors à encadrer les proportions observées.
Mais avant toute chose, comme pour l'étape 1, nous voulons afficher un résultat ordonné dans notre terminal, la première étape consiste alors à afficher clairement que nous commençons la partie 2 (étape 2.0) :

```
#Etape 2.0 - Théorie de l'estimation
print("\n" + "-"*60)
print("Partie 2 — Théorie de l'estimation (intervalles de confiance)")
print("-"*60 + "\n")
```

Voilà ce que nous affiche maintenant notre terminal :

```
------------------------------------------------------------
```

```
Partie 2 — Théorie de l'estimation (intervalles de confiance)
-------------------------------------------------------
```

Une fois cela fait, nous pouvons nous pencher sur l'extraction du premier échantillon (étape 2.1) :

```
#Etape 2.1 - Sélection 1er échantillon
premier_ech = donnees.iloc[0]
ligne = list(premier_ech.astype(int))
colonnes = list(donnees.columns)
print("Premier échantillon (ligne 0) :")
for nom, val in zip(colonnes, ligne):
    print(f"{nom} : {val}")
```

Le terminal nous confirme bien que nous avons extrait l'échantillon qui servira de base pour les intervalles de confiance.

```
Premier échantillon (ligne 0) :
Pour : 395
Contre : 396
Sans opinion : 209
```

Nous pouvons également venir mettre en évidence la taille totale et la fréquence de ce premier échantillon (étape 2.2), via le code suivant :

```
#Etape 2.2 - Calcul taille échantillon et fréquences
n = sum(ligne)
print(f"\nEffectif total de l'échantillon : n = {n}")

frequences = {nom: round(val / n, 2) for nom, val in zip(colonnes, ligne)}
print("\nFréquences observées sur cet échantillon :")
for nom, freq in frequences.items():
    print(f"{nom} : {freq}")
```

Notre terminal affiche les informations suivantes, base importante pour une analyse.

```
Effectif total de l'échantillon : n = 1000

Fréquences observées sur cet échantillon :
Pour : 0.4
Contre : 0.4
Sans opinion : 0.21
```

Vient maintenant le moment de calculer nos fameux intervalles de confiance à 95% (étape 2.3). Nous allons pouvoir composer avec le script suivant :

```
#Etape 2.3 - Calcul intervalle de confiance
zC = 1.96
ic_95 = {}
```

```
for nom, val in zip(colonnes, ligne):
    p = val / n
    marge = zC * math.sqrt((p * (1 - p)) / n)
    borne_inf = max(0.0, round(p - marge, 3))
    borne_sup = min(1.0, round(p + marge, 3))
    ic_95[nom] = (borne_inf, borne_sup)

print("\nIntervalles de confiance à 95 % :")
for nom, (inf, sup) in ic_95.items():
    print(f"{nom} : [{inf}, {sup}]")
```

Le terminal nous renvoie au résultat suivant :

```
Intervalles de confiance à 95 % :
Pour : [0.365, 0.425]
Contre : [0.366, 0.426]
Sans opinion : [0.184, 0.234]
```

Ces résultats nous montrent des intervalles déjà bien plus précis que les intervalles de fluctuation précédemment calculés.

Nous voulons à présent effectuer une comparaison avec les valeurs réelles et les intervalles de fluctuation (étape 2.4). Le script à retrouver ci-dessous croisera alors 3 informations : la fréquence réelle, l'intervalle de confiance et l'intervalle de fluctuation.

```
#Etape 2.4 - Comparaison fréquences réelles et intervalles
print("\nComparaison avec les fréquences réelles et les intervalles de fluctuation :\n")

for nom in colonnes:
    fr_real = frequences_reelles.get(nom, None)
    inf_ic, sup_ic = ic_95[nom]
    inf_fluct, sup_fluct = intervalle_fluctuation[nom]

    if inf_ic <= fr_real <= sup_ic:
        conclusion_ic = "fréquence réelle DANS l'IC"
    else:
        conclusion_ic = "fréquence réelle HORS de l'IC"

    if inf_fluct <= fr_real <= sup_fluct:
        conclusion_fluct = "fréquence réelle DANS l'intervalle de fluctuation"
    else:
        conclusion_fluct = "fréquence réelle HORS de l'intervalle de fluctuation"

    print(f"{nom} : réelle={fr_real} → {conclusion_ic} ; {conclusion_fluct}")
```

Dans le résultat affiché dans notre terminal, nous allons voir que les trois approches s'accordent :

Comparaison avec les fréquences réelles et les intervalles de fluctuation :

Pour : réelle=0.39 → fréquence réelle DANS l'IC ; fréquence réelle DANS l'intervalle de fluctuation
Contre : réelle=0.42 → fréquence réelle DANS l'IC ; fréquence réelle DANS l'intervalle de fluctuation
Sans opinion : réelle=0.19 → fréquence réelle DANS l'IC ; fréquence réelle DANS l'intervalle de fluctuation

Nous cherchons maintenant à opérer une vérification sur les 5 premiers échantillons (étapes 2.5), via le script que voici :

```
#Etape 2.5 - Verification plusieurs lignes
print("\nAnalyse rapide sur les 5 premiers échantillons (IC 95 %) :")

for i in range(min(5, len(donnees))):
    ligne_i = list(donnees.iloc[i].astype(int))
    n_i = sum(ligne_i)
    print(f"\nÉchantillon {i} — taille n = {n_i}")
    for nom, val in zip(colonnes, ligne_i):
        p_i = val / n_i
        marge_i = zC * math.sqrt((p_i * (1 - p_i)) / n_i)
        borne_inf_i = max(0.0, round(p_i - marge_i, 3))
        borne_sup_i = min(1.0, round(p_i + marge_i, 3))
        print(f"  {nom} : p = {round(p_i,3)} → IC95% [{borne_inf_i}, {borne_sup_i}]")
```

Nous pouvons alors constater dans le terminal un résultat qui révèle une grande stabilité entre les différents échantillons :

```
Analyse rapide sur les 5 premiers échantillons (IC 95 %) :

Échantillon 0 — taille n = 1000
  Pour : p = 0.395 → IC95% [0.365, 0.425]
  Contre : p = 0.396 → IC95% [0.366, 0.426]
  Sans opinion : p = 0.209 → IC95% [0.184, 0.234]

Échantillon 1 — taille n = 1000
  Pour : p = 0.379 → IC95% [0.349, 0.409]
  Contre : p = 0.432 → IC95% [0.401, 0.463]
  Sans opinion : p = 0.189 → IC95% [0.165, 0.213]

Échantillon 2 — taille n = 1000
  Pour : p = 0.384 → IC95% [0.354, 0.414]
  Contre : p = 0.426 → IC95% [0.395, 0.457]
  Sans opinion : p = 0.19 → IC95% [0.166, 0.214]

Échantillon 3 — taille n = 1000
  Pour : p = 0.395 → IC95% [0.365, 0.425]
  Contre : p = 0.407 → IC95% [0.377, 0.437]
  Sans opinion : p = 0.198 → IC95% [0.173, 0.223]

Échantillon 4 — taille n = 1000
  Pour : p = 0.389 → IC95% [0.359, 0.419]
  Contre : p = 0.413 → IC95% [0.382, 0.444]
  Sans opinion : p = 0.198 → IC95% [0.173, 0.223]
```

Partie 3 : Théorie de la décision, test de Shapiro-Wilk

Dans cette dernière partie, il s'agit de vérifier si les valeurs suivent une loi normale afin de savoir si l'on peut utiliser des tests paramétriques ou non.
Nous allons ainsi commencer, comme pour les étapes précédentes, par « nettoyer » nos résultats dans le terminal en affichant un titre (étape 3.0) et par charger nos fichiers pour le test (étape 3.1) :

```
#Etape 3.0 - Théorie de la décision
print("\n" + "-"*60)
print("Partie 3 — Théorie de la décision (test de Shapiro-Wilk)")
print("-"*60 + "\n")

import scipy.stats as stats

#Etape 3.1 - changement fichier CSV
fichiers = ["./data/Loi-normale-Test-1.csv", "./data/Loi-normale-Test-2.csv"]
donnees_tests = {}

for f in fichiers:
    try:
        df = pd.read_csv(f, header=None)
        valeurs = pd.to_numeric(df[0], errors='coerce').dropna().tolist()
        donnees_tests[f] = valeurs
        print(f"{f} chargé avec succès, {len(valeurs)} valeurs numériques.")
    except Exception as e:
        print(f"Erreur lors du chargement de {f} : {e}")
```

Voici alors nos résultats dans le terminal, ils vont nous permettre de poursuivre notre analyse :

```
------------------------------------------------------------
Partie 3 — Théorie de la décision (test de Shapiro-Wilk)
------------------------------------------------------------

./data/Loi-normale-Test-1.csv chargé avec succès, 2000 valeurs numériques.
./data/Loi-normale-Test-2.csv chargé avec succès, 2000 valeurs numériques.
```

Nous pouvons maintenant nous pencher sur l'application de notre test Shapiro-Wilk (étape 3.2). Nous utiliserons alors le script suivant :

```
#Etape 3.2 - Test de Shapiro-Wilks
print("\nRésultats du test de Shapiro-Wilk :")
for nom_fichier, valeurs in donnees_tests.items():
    stat, p_value = stats.shapiro(valeurs)
    print(f"\nFichier : {nom_fichier}")
```

```
    print(f"  Statistique W = {stat:.4f}")
    print(f"  p-value = {p_value:.4f}")

    if p_value > 0.05:
        print("Distribution NORMALE (hypothèse de normalité non rejetée)")
    else:
        print("Distribution NON NORMALE (hypothèse de normalité rejetée)")
```

Les résultats du test se présentent dans le terminal de cette façon :

```
Résultats du test de Shapiro-Wilk :

Fichier : ./data/Loi-normale-Test-1.csv
  Statistique W = 0.9639
  p-value = 0.0000
Distribution NON NORMALE (hypothèse de normalité rejetée)

Fichier : ./data/Loi-normale-Test-2.csv
  Statistique W = 0.2609
  p-value = 0.0000
Distribution NON NORMALE (hypothèse de normalité rejetée)
```

Dans notre dernière étape, nous allons par notre script rappeler la règle d'interprétation (étape 3.3) :

```
#Etape 3.3 - Interprétation
print("\nInterprétation :")
print("Le test de Shapiro-Wilk vérifie si l'échantillon suit une loi normale.")
print("- Si p-value > 0.05 : la distribution peut être considérée comme normale.")
print("- Si p-value ≤ 0.05 : la distribution ne suit pas la loi normale.")
print("Ces résultats permettront de décider quel test statistique utiliser pour l'analyse ultérieure.\n")
```

Ainsi le terminal nous rappelle bien que :

```
- Si p-value > 0.05 : la distribution peut être considérée comme normale.
- Si p-value ≤ 0.05 : la distribution ne suit pas la loi normale.
Ces résultats permettront de décider quel test statistique utiliser pour l'analyse ultérieure.
```

Dans notre séance, les deux fichiers ici utilisés ne suivent pas une loi normale. Il faudra alors utiliser des tests plus adaptés.

Partie Bonus

Il nous est indiqué que l'un des fichiers ne suit pas une loi normale. D'après les distributions étudiées lors de la séance précédente, la distribution la plus probable serait celle de la loi exponentielle. Cette distribution est asymétrique et décroissante, ce qui correspond au comportement observé dans les valeurs du fichier et explique le rejet de l'hypothèse de normalité.

Séance 6

Objectifs

- Manipuler des fonctions locales et comprendre la nécessité de factoriser son code en une liste de fonctions ou de procédures exécutant une tâche unique
- Créer des fonctions locales spécifiques au traitement d'un problème
- Comprendre l'analyse de variables qualitatives ordinales

Questions de cours

1. Qu'est-ce qu'une statistique ordinale ? À quelle autre statistique catégorielle s'oppose-telle ? Quel type de variables utilise-t-elle ? En quoi cela peut matérialiser une hiérarchie spatiale ?

Une statistique ordinale correspond à l'ensemble des méthodes fondées sur le classement d'objets ou d'individus, c'est-à-dire sur l'ordre des observations plutôt que sur leurs valeurs absolues. Elle s'appuie sur les rangs, notés $X(1) \leq \cdots \leq X(n)$, obtenus en ordonnant une série d'observations. Elle s'oppose ainsi aux statistiques nominales, qui classent les individus dans des catégories sans relation d'ordre.
La statistique ordinale utilise des variables éponymes, elles aussi dites « ordinales ». Il s'agit de variables qualitatives pour lesquelles un ordre naturel peut être établi : croissant ou décroissant. L'ordre croissant est l'ordre privilégié dans la plupart des applications, sauf pour quelques cas particuliers comme la loi rang-taille
Ce type de statistique matérialise aisément une hiérarchie spatiale, car de nombreux phénomènes géographiques produisent spontanément des classements : taille des villes, intensité de phénomènes physiques comme des crues, séismes, *etc*, dynamisme socio-économique, *etc*. Le classement permet alors d'identifier les entités « en tête », « moyennes » ou « en queue », rendant visible l'organisation hiérarchique d'un ou des territoires.

2. Quel ordre est à privilégier dans les classifications ?

Dans les classifications, l'ordre à privilégier est l'ordre dit « croissant », aussi appelé « ordre naturel ». Il facilité l'analyse des rangs, la détection des valeurs aberrantes et l'étude de certaines distributions, comme par exemple la plus grande valeur d'une série.

3. Quelle est la différence entre une corrélation des rangs et une concordance de classements ?

La corrélation des rangs vise à mesurer la similarité entre deux séries ordonnées en comparant les rangs attribués à chaque individu. Il est possible d'identifier deux outils principaux : le coefficient de *Spearman* et le coefficient τ *de Kendall*. Ils permettent de savoir si les rangs sont proches, inverses ou indépendants.

La concordance de classements, quant à elle, s'intéresse au nombre de paires concordantes et discordantes entre deux classements. Elle repose ainsi sur l'examen du respect ou non de l'ordre naturel entre les couples de rangs. La concordance est dite « complète » si toutes les paires vont dans le même sens ; de l'autre côté, elle est dite « nulle » si le nombre de concordances et de discordances s'équilibre. Ainsi, la corrélation mesure une proximité globale entre deux ordres, là où la concordance mesure la cohérence paire par paire entre deux classements.

4. Quelle est la différence entre les tests de Spearman et de Kendall ?

Ces deux tests sont utilisés pour la comparaison de classement mais il est possible d'en distinguer les logiques. Le test de Spearman utilise directement les rangs et calcul une corrélation à partir de la différence entre deux séries : (ui−vi)2. La formule finale de ce test prend la forme suivante :

$$r_s = 1 - \frac{6}{n\left(n^2-1\right)} \sum_{i=1}^{n} \left(u_i - v_i\right)^2$$

Ce test est sensible à la présence ou non « d'ex aequo » et la distribution peut être assimilée à une loi normale pou n > 30.

Le test de Kendall quant à lui repose sur le nombre de paires concordantes et discordantes. Ainsi le coefficient s'écrit de la façon suivante :

$$\tau = \frac{2S_c}{n\left(n-1\right)}$$

Ce test peut être qualifié de « plus simple » conceptuellement, il compare l'ordre de chaque paire d'individu. Il a également l'avantage de se généraliser plus facilement à p classements.

Ainsi, plus globalement, Spearman mesure la proximité des rangs de manière quantitative, tandis que Kendall mesure la cohérence de l'ordre de manière qualitative. Les deux tests sont complémentaires et utiles pour l'analyse des hiérarchies spatiales lorsque plusieurs classements coexistent.

5. À quoi servent les coefficients de Goodman-Kruskal et de Yule ?

Le coefficient de Goodman-Kruskal mesure la force d'association d'ordre entre deux variables ordinales, en comparant le nombre de paires concordantes (Na) et discordantes (Nd). Son équation prend la forme suivante :

$$\Gamma = \frac{Na - Nd}{Na + Nd}$$

Pour sa lecture, il est important de savoir qu'il varie entre -1 et +1 et s'interprète comme un indicateur de concordance. On retrouve ainsi : Γ = +1 comme concordance parfaite, Γ = -1 comme inversion totale et Γ = 0 comme une absence d'association observable. Ce coefficient se rapproche conceptuellement du τ de Kendall.

Le Coefficient de Yule ou Q de Yule, quant à lui, est un cas particulier du Γ, réservé aux tableaux de contingence en 2 x 2. Il s'exprime sous la forme suivante :

$$Q = \frac{ad - bc}{ad + bc}$$

Il permet de mesure l'association entre deux variables dites dichotomiques, comme oui et non, absent et présent, etc. A l'instar de Γ, il varie entre −1 et +1 et indique : + 1 comme une association positive parfaite, - 1 comme une association négative parfaite et 0 comme une absence d'association.

En somme, Goodman-Kruskal propose une mesure générale de l'association d'ordre fondée sur les couples classés, tandis que Yule propose un outil plus ciblé pour des associations binaires. Leur usage permet de quantifier la force d'une relation entre des variables catégorielles et d'interpréter rigoureusement les hiérarchies ou les dépendances observées dans des données géographiques.

Mise en œuvre avec Python

Cette 6e et dernière séance ne faisant pas exception, nous allons la commencer comme toutes les autres, en plaçant les fichiers CSV nécessaires dans le dossier data du répertoire src, ouvrant notre main.py et en nous positionnant dans le bon chemin de dossier sur le terminal, pour pouvoir exécuter notre code sereinement.

Partie 1 : Analyse loi rang-taille

Dans cette première partie, il nous est demandé une analyse des surfaces des îles et continents afin de représenter la loi rang-taille. La première étape consiste alors à charger le fichier CVS en utilisant la fonction locale ouvrirUnFichier() (étape 1.0). Cette étape nous permet de visualiser les premières lignes pour vérifier la bonne importation. Ainsi, notre script prend la forme suivante :

```
#Etape 1.0 - Chargement du fichier island-index.csv
print("\n" + "-"*60)
print("Étape 1.0 — Importation du fichier island-index.csv")
print("-"*60)

iles = pd.DataFrame(ouvrirUnFichier("./data/island-index.csv"))
print("\nFichier chargé avec succès :")
print(iles.head())
```

Le résultat attendu s'affiche dans le terminal :

```
------------------------------------------------------------
Étape 1.0 — Importation du fichier island-index.csv
------------------------------------------------------------

Fichier chargé avec succès :
           Type            Identifiant Toponyme Code ISO 1  ... Trait de côte (km) Surface (km²)   Latitude  Longitude
ILE NORMALE     1           Aruba Island      ABW        NaN  ...         105.706524    181.938366 -69.970276  12.509315
ILE NORMALE     3  Baia dos Tigres Island      AGO        NaN  ...          70.472444     88.570958  11.704911 -16.595546
ILE NORMALE     5                    NaN      AGO        NaN  ...          52.638245     26.792477  12.299087  -6.118821
ILE NORMALE     6                    NaN      AGO        NaN  ...          12.593973      2.988454  12.291456  -6.155205
ILE NORMALE     7                    NaN      AGO        NaN  ...           2.249531      0.090183  12.341880 -14.393714

[5 rows x 10 columns]
```

Nous voulons maintenant isoler la colonne « Surface (km2) » et y ajouter les surfaces de continents (étape 1.1). Pour ce faire, il est d'abord important de forcer le typage en float et d'enlever l'unité. Nous utiliserons les lignes de code suivantes :

```
#Etape 1.1 - Extraction colonne surfaces et ajout continents
print("\n" + "-"*60)
print("Étape 1.1 — Extraction des surfaces des îles")
print("-"*60)

surfaces = list(iles["Surface (km²)"])
surfaces = [float(x) for x in surfaces]   # typage forcé en float

print("Nombre de surfaces initiales :", len(surfaces))

continents = [
   85545323,   # Asie / Afrique / Europe combinés
   37856841,   # Amérique
   7768030,    # Antarctique
   7605049     # Australie
]

surfaces.extend([float(v) for v in continents])

print("Nouveau nombre de surfaces :", len(surfaces))
```

Le résultat suivant apparait dans notre terminal :

```
------------------------------------------------------------
Étape 1.1 — Extraction des surfaces des îles
------------------------------------------------------------
Nombre de surfaces initiales : 84219
Nouveau nombre de surfaces : 84223
```

Maintenant que cela est fait, pour visualiser correctement la loi rang–taille, nous trions la liste des surfaces par ordre décroissant à l'aide de la fonction locale ordreDecroissant() (étape 1.2) :

```
#Etape 1.2 - Ordre décroissant
print("\n" + "-"*60)
print("Étape 1.2 — Tri décroissant des surfaces")
print("-"*60)

surfaces_triees = ordreDecroissant(surfaces)
print("Extrait des surfaces triées (10 premières valeurs) :")
print(surfaces_triees[:10])
```

Le résultat attendu s'écrit dans le terminal, et les dix premières valeurs sont affichées :

```
------------------------------------------------------------
```

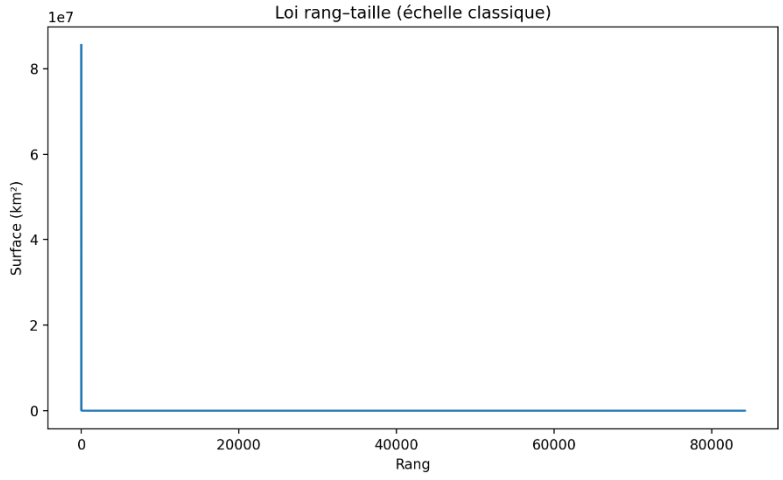
```
Étape 1.2 — Tri décroissant des surfaces
------------------------------------------------------------
Extrait des surfaces triées (10 premières valeurs) :
[85545323.0, 37856841.0, 7768030.0, 7605049.0, 2117507.755175175, 785274.519913497,
725098.0491459599, 590547.3794915207, 515224.18422131287, 429003.2172921432]
```

Il s'agit à présent de générer la représentation graphique de la loi rang-taille sur une échelle classique (étape 1.3). Le script suivant va nous permettre de générer notre image :

```
#Etape 1.3 - Visualisation loi rang-taille
print("\n" + "-"*60)
print("Étape 1.3 — Visualisation loi rang–taille")
print("-"*60)

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

plt.figure(figsize=(8,5))
plt.plot(rangs, surfaces_triees)
plt.title("Loi rang–taille (échelle classique)")
plt.xlabel("Rang")
plt.ylabel("Surface (km²)")
plt.tight_layout()
plt.savefig("rang_taille_classique.png", dpi=200)
plt.close()

print("Image générée : rang_taille_classique.png")
```

Et, en effet, le terminal nous indique que l'image a été générée, nous pouvons la retrouver dans nos dossiers. Voici le message du terminal ainsi que la représentation graphique :

```
------------------------------------------------------------
Étape 1.3 — Visualisation loi rang–taille
------------------------------------------------------------
Image générée : rang_taille_classique.png
```

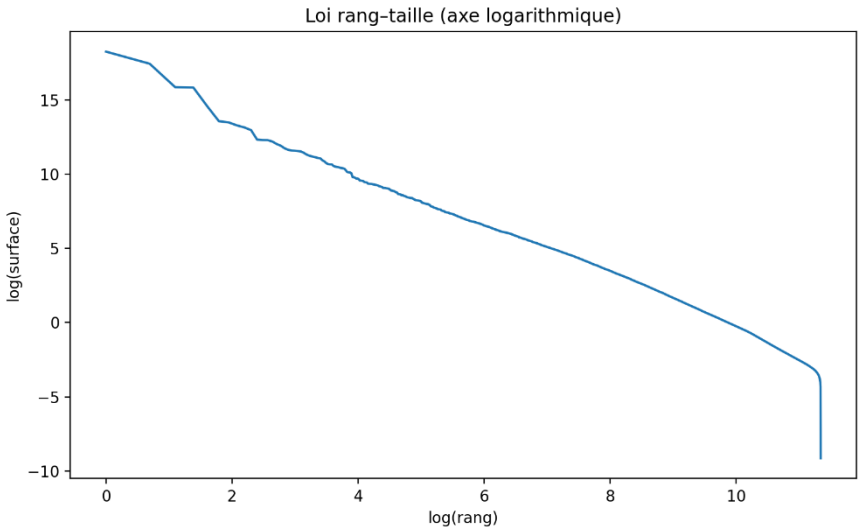


Il nous est maintenant demandé, dans un souci d'amélioration de lisibilité, de transformer les axes en échelle logarithmique (étape 1.4). Voici le code utilisé pour cette étape :

```
#Etape 1.4 - Conversion logarithmique données
print("\n" + "-"*60)
print("Étape 1.4 — Conversion logarithmique des axes")
print("-"*60)

log_rangs = conversionLog(rangs)
log_surfaces = conversionLog(surfaces_triees)

plt.figure(figsize=(8,5))
plt.plot(log_rangs, log_surfaces)
plt.title("Loi rang–taille (axe logarithmique)")
plt.xlabel("log(rang)")
plt.ylabel("log(surface)")
plt.tight_layout()
plt.savefig("rang_taille_log.png", dpi=200)
plt.close()

print("Image générée : rang_taille_log.png")
```

De la même manière, le terminal nous indique que l'image est générée, et l'on peut la retrouver au même endroit que l'image précédente :

```
------------------------------------------------------------
Étape 1.4 — Conversion logarithmique des axes
------------------------------------------------------------
Image générée : rang_taille_log.png
```



La question de la possibilité d'un test sur les rangs nous a été posée, comme demandée, la réponse est laissée en commentaire dans les lignes de code (étape 1.5), la voici :

```
#Etape 1.5 Test statistique possible ?
# Non. Les rangs ne sont pas des variables aléatoires : ce sont des valeurs
# déterministes obtenues après un tri. Ils ne suivent donc pas une distribution
# probabiliste permettant d'appliquer un test statistique (normalité, KS, etc.)
# On ne peut tester que les valeurs (surfaces), jamais les rangs eux-mêmes.
```

Partie 2 : Populations du monde (rangs et corrélation)

Il nous est à présent demandé de comparer les classements par population et par densité pour 195 Etats du monde. Mais avant toute chose préparons notre terrain ! Pour importer notre fichier (étape 2.0), nous utiliserons le script suivant :

```
# Étape 2.0 — Chargement du fichier populations
print("\n" + "-"*60)
print("Étape 2.0 — Importation du fichier Le-Monde-HS-Etats-du-monde-2007-2025.csv")
print("-"*60)

monde = pd.DataFrame(ouvrirUnFichier("./data/Le-Monde-HS-Etats-du-monde-2007-2025.csv"))
print("\nFichier chargé avec succès :")
print(monde.head())
```

Voici alors ce que nous montre le terminal, indiquant que les données ont bien été chargées :

```
------------------------------------------------------------
Étape 2.0 — Importation du fichier Le-Monde-HS-Etats-du-monde-2007-2025.csv
------------------------------------------------------------

Fichier chargé avec succès :
  Code ISO_3  Numéro Continent rattaché         State  ... Densité 2022  Densité 2023  Densité 2024  Densité 2025
0        afg       1              Asie   Afghanistan  ...         60.95         62.94         64.62         65.24
1        zaf       2            Afrique  South Africa  ...         49.10         48.20         49.43         52.37
2        alb       3            Europe       Albania  ...         98.97         96.55         96.55         93.10
3        dza       4            Afrique      Algeria  ...         18.71         18.83         19.13         19.63
4        deu       5            Europe       Germany  ...        234.73        233.61        233.33        236.69

[5 rows x 45 columns]
```

Il nous est ensuite demandé de sélectionner les colonnes suivantes : « État », « Pop 2007 », « Pop 2025 », « Densité 2007 » et « Densité 2025 » (étape 2.1). Nous isolons ces colonnes car elles s'avèrent pertinentes pour une étude statistique, c'est ce script qui va nous y autoriser :

```
#Etape 2.1 - Isolation colonnes pertinentes
print("\n" + "-"*60)
print("Étape 2.1 — Colonnes à analyser : État, Pop 2007, Pop 2025, Densité 2007, Densité 2025")
print("-"*60)

colonnes_analyse = ["État", "Pop 2007", "Pop 2025", "Densité 2007", "Densité 2025"]
donnees = monde[colonnes_analyse]

etats = list(donnees["État"])
pop2007 = [float(x) for x in donnees["Pop 2007"]]
pop2025 = [float(x) for x in donnees["Pop 2025"]]
```

```
dens2007 = [float(x) for x in donnees["Densité 2007"]]
dens2025 = [float(x) for x in donnees["Densité 2025"]]

print(f"Nombre d'États : {len(etats)}")
```

Nous souhaitons à présent trier nos populations et densités de manière décroissante avec la fonction locale « ordrePopulation() » (étape 2.2). Voici le code utilisé :

```
print("\n" + "-"*60)
print("Étape 2.2 — Classement décroissant des populations et densités")
print("-"*60)

ordre_pop2007 = ordrePopulation(pop2007, etats)
ordre_pop2025 = ordrePopulation(pop2025, etats)
ordre_dens2007 = ordrePopulation(dens2007, etats)
ordre_dens2025 = ordrePopulation(dens2025, etats)

print("Extrait classement Pop 2007 (5 premiers) :", ordre_pop2007[:5])
print("Extrait classement Densité 2007 (5 premiers) :", ordre_dens2007[:5])
```

Le terminal, une fois cette partie du code jouée, nous extrait, à titre d'exemple les 5 premiers pays en termes de population et de densité :

```
------------------------------------------------------------
Étape 2.2 — Classement décroissant des populations et densités
------------------------------------------------------------
Extrait classement Pop 2007 (5 premiers) : [[1, 'Chine'], [2, 'Inde'], [3, 'États-Unis'], [4, 'Indonésie'], [5, 'Brésil']]
Extrait classement Densité 2007 (5 premiers) : [[1, 'Singapour'], [2, 'Malte'], [3, 'Bangladesh'], [4, 'Maldives'], [5, 'Bahreïn']]
```

Il nous faut à présent préparer notre comparaison des classements (étapes 2.3). Nous allons ainsi préparer la liste combinée pour comparer les rangs population / densité à l'aide de la fonction « classementPays() » :

```
#Etape 2.3 - Préparation comparaison classement pays
print("\n" + "-"*60)
print("Étape 2.3 — Préparation de la comparaison des classements")
print("-"*60)

comparaison_pop_dens = classementPays(ordre_pop2007, ordre_dens2007)
comparaison_pop_dens.sort()  # tri par rapport au classement de 2007
```

Le terminal nous indique bien que l'étape a été faite en imprimant :

```
------------------------------------------------------------
Étape 2.3 — Préparation de la comparaison des classements
------------------------------------------------------------
```

Les prochaines étapes consistent à calculer la corrélation de rang et de concordance. Nous allons ainsi dans un premier temps isoler des colonnes pour corrélation (étape 2.4). Puis nous pourrons calculer nos coefficients de Spearman er de Kendall (étape 2.5), pour analyser la similitude des classements. Voici le script qui va nous permettre de réaliser ces deux étapes de calcul de corrélation et de concordance :

```
#Etape 2.4 - Isolation deux colonnes pour corrélation
print("\n" + "-"*60)
print("Étape 2.4 — Isolation des rangs Pop 2007 et Densité 2007")
print("-"*60)

rangs_pop = []
rangs_dens = []

for element in comparaison_pop_dens:
    rangs_pop.append(element[0])
    rangs_dens.append(element[1])

print("Extrait rangs Pop 2007 :", rangs_pop[:10])
print("Extrait rangs Densité 2007 :", rangs_dens[:10])

#Etape 2.5 - Calcul corrélations rang
print("\n" + "-"*60)
print("Étape 2.5 — Corrélation de rang (Spearman) et concordance (Kendall)")
print("-"*60)

from scipy.stats import spearmanr, kendalltau

spearman_coef, spearman_p = spearmanr(rangs_pop, rangs_dens)
kendall_coef, kendall_p = kendalltau(rangs_pop, rangs_dens)

print(f"Coefficient de corrélation de rang Spearman : {spearman_coef:.4f} (p-value = {spearman_p:.4f})")
print(f"Coefficient de concordance de rang Kendall : {kendall_coef:.4f} (p-value = {kendall_p:.4f})")
```

Le terminal nous indique ainsi que nos deux étapes ont été réalisées et nous donne les résultats de nos deux coefficients :

```
------------------------------------------------------------
Étape 2.4 — Isolation des rangs Pop 2007 et Densité 2007
------------------------------------------------------------
Extrait rangs Pop 2007 : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Extrait rangs Densité 2007 : [39, 14, 123, 48, 134, 28, 3, 159, 38, 16]

------------------------------------------------------------
Étape 2.5 — Corrélation de rang (Spearman) et concordance (Kendall)
------------------------------------------------------------
Coefficient de corrélation de rang Spearman : 0.0973 (p-value = 0.2056)
Coefficient de concordance de rang Kendall : 0.0693 (p-value = 0.1786)
```

Pour finir, le commentaire de ces résultats (étape 2.5) a été intégré à même le script python via un « # », une commande a été utilisée pour qu'il s'affiche également dans le terminal. Voici ce commentaire :

```
# Commentaire :
# Ces coefficients indiquent le degré de similarité entre le classement par population et le classement par densité.
# Valeurs proches de 1 => forte concordance ; proches de 0 => classement indépendant ; valeurs négatives => classement inverse.
```

Retours sur le cours

Comme beaucoup de personnes lors de l'annonce du contenu du cours, ou de la course pour le choix des « niveaux » (bien qu'ils n'aient aucune signification réelle), ou encore lors de la première séance où nous ont été présentés les outils, exercices et objectifs, j'ai un peu paniqué. Comme beaucoup d'étudiant.e.s, j'avais arrêté les mathématiques au lycée, je choisis des méthodes qualitatives dès que j'en ai l'occasion et j'utilise Excel à reculons. Pourtant, comme pour tout ce qui concerne les statistiques, et ce depuis la spé mathématique au lycée, je sais que le plus important est de ne pas se laisser impressionner.

J'ai d'abord mobilisé l'aide de ma famille, plus proche du monde des mathématiques que moi, il faut l'admettre. J'ai essayé de me faciliter la vie autant que possible en installant tous les outils nécessaires pour éviter les problèmes par la suite. Et c'est ce qui s'est passé : je n'ai rencontré aucun problème technique. Le fait de ne pas avoir un Mac a probablement aussi aidé.

Au fil du temps, je peux dire que je ne trouve pas ce cours infaisable, même s'il reste impressionnant. Après un moment, j'ai eu la sensation de comprendre ce que je faisais et, et est-ce que j'aurais pris du plaisir à organiser mes lignes de code et leur présentation dans le terminal ? Peut-être. Peu après avoir terminé la séance 5, je me suis rendu compte qu'avec tous les outils installés sur mon ordinateur, et gratuitement en plus, je pouvais rapidement écrire un script et obtenir automatiquement tous mes tris à plat et mes futurs Chi^2. C'est là que l'intérêt du cours m'est apparu : j'ai commencé à le trouver pertinent, voire presque amusant (autant que peut l'être la statistique, bien sûr).

Cependant, je dois reconnaître que le début n'a pas été facile. Tout au long du semestre, alors que j'avançais dans mes séances, le cours posait de plus en plus de problèmes à certains de mes collègues. Selon moi, le format mérite d'être questionné. Pour la plupart, ce cours représente un vrai saut dans le grand bain, dans l'inconnu total. J'admets qu'un peu plus d'accompagnement et de cours théoriques aurait pu être intéressant et rassurant.

C'est pourquoi je me questionne sur la pertinence de la pédagogie inversée. Bien qu'elle favorise l'autonomie et la débrouillardise, elle peut aussi conduire au déni ou à l'évitement, notamment pour des étudiant.e.s déjà effrayé.e.s par Excel. J'ai conscience que les contraintes de salles, de matériel (ordinateurs de la Sorbonne) et d'effectifs rendent le cours plus complexe. Peut-être que ce format est le seul capable de s'adapter à ces contraintes techniques et humaines. Pourtant, je maintiens que l'autonomie totale n'est pas toujours la meilleure solution.

L'objectif des séances présentielles était de pouvoir répondre à nos questions et soucis, certes. Mais parfois, il n'était même pas possible de savoir par où commencer. Ainsi, si je devais retenir deux points de vigilance essentiels, cela serait :

- L'installation de tous les outils sur sa machine.
- La compréhension, plus que l'apprentissage, d'un langage comme Python.