Analyse de données avec Python

pour les professionnels de la santé

Un cours pratique de 30 heures

Yaé Ulrich Gaba

AIRINA Labs

2026

Table des matières

Préface

Ce cours est conçu pour les professionnels de la santé — médecins, infirmiers, épidémiologistes, chercheurs en santé publique et analystes de politiques de santé — qui ont besoin d'analyser des données mais qui ont peu ou pas d'expérience en programmation. Vous n'avez pas besoin de devenir développeur. Vous devez devenir quelqu'un capable de charger un jeu de données, le nettoyer, l'explorer, tester une hypothèse, construire un modèle prédictif simple et communiquer clairement les résultats.

Nous utilisons Python parce qu'il est gratuit, largement adopté dans la recherche en santé, et possède un écosystème de bibliothèques spécifiquement conçues pour les types d'analyses dont les professionnels de la santé ont besoin : analyse de survie, mesures épidémiologiques, cartographie géospatiale et prédiction clinique.

Chaque chapitre suit le même rythme : une brève explication du concept, un exemple appliqué sur des données de santé réelles, et des exercices à compléter dans des notebooks Jupyter. Tous les jeux de données utilisés dans ce cours sont librement disponibles auprès de l'OMS, du CDC, du UCI Machine Learning Repository et d'autres sources ouvertes.

Ce que ce cours n'est pas. Ce n'est pas un cours de statistiques (bien que nous utilisions les statistiques). Ce n'est pas un cours d'informatique (bien que nous écrivions du code). C'est un *cours pratique d'analyse de données* pour des personnes dont l'expertise principale est la santé, et non la programmation.

Prérequis. Aucun, au-delà de compétences informatiques de base. Une familiarité avec les statistiques descriptives (moyenne, médiane, écart-type) est utile mais sera révisée.

Logiciels. Tout le code s'exécute dans des notebooks Jupyter, soit localement (Anaconda), soit dans le cloud (Google Colab — gratuit, aucune installation requise).

Yaé Ulrich Gaba
Cotonou, 2026

Chapitre 1

Bases de Python pour les professionnels de la santé

« L'objectif n'est pas d'apprendre Python. L'objectif est de répondre à des questions de santé avec des données. Python n'est que l'outil. »

1.1 Pourquoi Python pour les données de santé ?

Trois raisons pour lesquelles les professionnels de la santé devraient apprendre Python plutôt que de se fier uniquement à Excel ou SPSS :

1. **Reproductibilité.** Un script Python documente chaque étape de votre analyse. Quand un examinateur demande « comment avez-vous traité les valeurs de laboratoire manquantes ? », vous montrez le code — pas une description de clics dans des menus.
2. **Échelle.** Excel a du mal avec 100 000 dossiers de patients. Python en gère des millions sans ralentir.
3. **Écosystème.** Des bibliothèques comme `lifelines` (analyse de survie), `geopandas` (cartographie des maladies) et `scikit-learn` (modèles prédictifs) offrent des outils que SPSS ne propose pas.

Contexte clinique

Vous n'aurez pas besoin de mémoriser la syntaxe. Les professionnels de la santé utilisent Python comme les cliniciens utilisent un stéthoscope — vous apprenez les parties dont vous avez besoin et vous cherchez le reste. Chaque bloc de code de ce cours peut être copié dans un notebook Jupyter et exécuté immédiatement.

1.2 Configurer votre environnement

1.2.1 Google Colab (recommandé)

Aucune installation. Rendez-vous sur `https://colab.research.google.com`, connectez-vous avec un compte Google et créez un nouveau notebook. Toutes les bibliothèques que nous utilisons sont préinstallées.

1.2.2 Installation locale avec Anaconda

Si vous préférez travailler hors ligne :

1. Téléchargez Anaconda : https://www.anaconda.com/download
2. Installez avec les paramètres par défaut.
3. Ouvrez **Jupyter Notebook** depuis Anaconda Navigator.

1.3 Votre premier notebook

Un notebook Jupyter est un document qui mélange texte, code et résultats. Chaque *cellule* contient soit :

- **Du code** — des instructions Python qui produisent un résultat.
- **Du Markdown** — du texte formaté (explications, titres, notes).

Appuyez sur `Shift + Entrée` pour exécuter une cellule.

```
# Votre premiere cellule Python
print("Bonjour, donnees de sante !")
```

1.4 Variables et types

Une *variable* stocke une valeur. En données de santé, vous travaillez constamment avec quatre types :

```
patient_age = 45              # int (entier)
temperature = 37.2            # float (nombre decimal)
diagnosis = "Type 2 Diabetes" # str (chaine de caracteres / texte)
is_hospitalized = True        # bool (True ou False)
```

Astuce Python

Les noms de variables doivent être descriptifs. Utilisez `patient_age`, pas `x`. Votre futur vous (et vos collaborateurs) vous remercieront.

1.5 Collections : listes et dictionnaires

1.5.1 Listes

Une liste contient plusieurs valeurs dans un ordre donné :

```
blood_pressures = [120, 135, 128, 142, 118]
print(f"Nombre de mesures : {len(blood_pressures)}")
print(f"Premiere mesure : {blood_pressures[0]}")  # l'indexation commence \`a 0
print(f"Derniere mesure : {blood_pressures[-1]}")
```

1.5.2 Dictionnaires

Un dictionnaire associe des clés à des valeurs — comme un dossier patient :

```
patient = {
    "id": "PAT-0042",
    "age": 58,
    "sex": "F",
    "diagnosis": "Hypertension",
    "systolic_bp": 148,
    "medications": ["Amlodipine", "Hydrochlorothiazide"]
}
print(patient["diagnosis"])  # Hypertension
```

1.6 Structures de contrôle

1.6.1 Conditions

```
systolic = 148

if systolic >= 140:
    category = "Hypertension de stade 2"
elif systolic >= 130:
    category = "Hypertension de stade 1"
elif systolic >= 120:
    category = "Elevee"
else:
    category = "Normale"

print(f"Categorie de PA : {category}")
```

1.6.2 Boucles

```
patients = [
    {"id": "P001", "age": 34, "hba1c": 5.4},
    {"id": "P002", "age": 67, "hba1c": 7.8},
    {"id": "P003", "age": 52, "hba1c": 6.1},
]

for p in patients:
    if p["hba1c"] >= 6.5:
        print(f"{p['id']}: Diabetique (HbA1c = {p['hba1c']})")
    elif p["hba1c"] >= 5.7:
        print(f"{p['id']}: Pre-diabetique (HbA1c = {p['hba1c']})")
    else:
        print(f"{p['id']}: Normal (HbA1c = {p['hba1c']})")
```

1.7 Fonctions

Une fonction encapsule un calcul réutilisable :

```python
def bmi(weight_kg, height_m):
    """Calculer l'indice de masse corporelle."""
    return weight_kg / (height_m ** 2)

def bmi_category(bmi_value):
    """Classifier l'IMC selon les categories de l'OMS."""
    if bmi_value < 18.5:
        return "Insuffisance ponderale"
    elif bmi_value < 25.0:
        return "Poids normal"
    elif bmi_value < 30.0:
        return "Surpoids"
    else:
        return "Obesite"

value = bmi(82, 1.75)
print(f"IMC : {value:.1f} ({bmi_category(value)})")
```

Exercice

1. Écrivez une fonction `classify_bp(systolic, diastolic)` qui retourne une catégorie de pression artérielle selon les recommandations AHA/ACC.
2. Créez une liste de 5 dictionnaires patients avec les champs `weight_kg` et `height_m`. Parcourez-les, calculez l'IMC et affichez la catégorie pour chacun.
3. Écrivez une fonction `egfr(creatinine, age, sex)` qui estime le débit de filtration glomérulaire selon l'équation CKD-EPI.

1.8 Importer des bibliothèques

La puissance de Python vient de ses bibliothèques. Voici celles que nous utiliserons tout au long de ce cours :

```python
import pandas as pd          # manipulation de donnees
import numpy as np           # calcul numerique
import matplotlib.pyplot as plt  # graphiques
import seaborn as sns        # visualisation statistique
from scipy import stats      # tests statistiques
```

Attention

Si vous obtenez `ModuleNotFoundError`, installez la bibliothèque manquante :

```
pip install pandas matplotlib seaborn scipy
```

Dans Google Colab, préfixez avec `!` : `!pip install lifelines`

1.9 Résumé du chapitre

- Python est gratuit, reproductible et dispose de bibliothèques spécifiques à la santé.
- Les notebooks Jupyter mélangent code, texte et résultats dans un seul document.
- Les variables stockent des valeurs ; les listes et dictionnaires les organisent.
- Les fonctions rendent votre analyse réutilisable et lisible.
- `pandas`, `matplotlib`, `seaborn` et `scipy` sont les bibliothèques fondamentales pour l'analyse de données de santé.

Chapitre 2

Données de santé avec Pandas

« 80 % de l'analyse de données consiste à mettre les données dans une forme où l'on peut leur poser des questions. »

2.1 Qu'est-ce qu'un DataFrame ?

Un DataFrame est un tableau — les lignes sont des observations (patients, pays, points temporels), les colonnes sont des variables (âge, diagnostic, valeurs de laboratoire). Si vous avez utilisé Excel, vous comprenez déjà le concept. Pandas vous donne la puissance d'Excel avec la reproductibilité de Python.

```
import pandas as pd

# Creer un petit DataFrame clinique
data = {
    "patient_id": ["P001", "P002", "P003", "P004", "P005"],
    "age": [45, 67, 34, 52, 71],
    "sex": ["M", "F", "F", "M", "F"],
    "systolic_bp": [128, 158, 112, 145, 162],
    "hba1c": [5.4, 7.8, 5.1, 6.3, 8.2],
    "diagnosis": ["None", "T2DM", "None", "Pre-DM", "T2DM"]
}
df = pd.DataFrame(data)
df
```

2.2 Charger des données de santé réelles

2.2.1 Fichiers CSV

La plupart des jeux de données de santé sont fournis sous forme de fichiers CSV (valeurs séparées par des virgules) :

```
# Donnees d'esperance de vie de l'OMS (Gapminder)
url =
↪  "https://raw.githubusercontent.com/datasets/gapminder/main/data/gapminder.csv"
```

```
gapminder = pd.read_csv(url)
print(f"Dimensions : {gapminder.shape}")  # (lignes, colonnes)
gapminder.head()
```

2.2.2 Fichiers Excel

```
df = pd.read_excel("patient_records.xlsx", sheet_name="Lab Results")
```

2.2.3 Depuis l'Observatoire mondial de la santé de l'OMS

```
# Ratio de mortalite maternelle par pays
url = "https://apps.who.int/gho/athena/api/GHO/MDG_0000000026?format=csv"
mmr = pd.read_csv(url)
mmr.head()
```

Contexte clinique

L'API GHO de l'OMS fournit plus de 1 000 indicateurs de santé pour 194 États membres. Vous pouvez parcourir les indicateurs sur https://www.who.int/data/gho/data/indicators. Chaque indicateur possède un code (par ex. `MDG_0000000026` pour le ratio de mortalité maternelle) que vous intégrez dans l'URL de l'API.

2.3 Explorer vos données

La première chose à faire avec tout jeu de données de santé — avant toute analyse — est de *le regarder*.

```
# Exploration de base
print(gapminder.shape)           # dimensions
print(gapminder.columns.tolist())  # noms des colonnes
print(gapminder.dtypes)          # types de donnees
gapminder.describe()             # statistiques descriptives
gapminder.info()                 # comptage des non-nuls et memoire
```

2.3.1 Sélectionner des colonnes

```
# Une seule colonne (retourne une Series)
ages = gapminder["lifeExp"]

# Plusieurs colonnes (retourne un DataFrame)
subset = gapminder[["country", "year", "lifeExp"]]
```

2.3.2 Filtrer des lignes

```
# Patients avec HbA1c >= 6.5 (seuil diabetique)
diabetic = df[df["hba1c"] >= 6.5]

# Pays africains dans Gapminder
africa = gapminder[gapminder["continent"] == "Africa"]

# Conditions multiples : patientes de plus de 60 ans avec PA elevee
high_risk = df[(df["sex"] == "F") & (df["age"] > 60) & (df["systolic_bp"] >=
↪  140)]
```

> **Attention**
>
> Utilisez `&` (et), `|` (ou), `~` (non) avec des parenthèses autour de chaque condition. Les mots-clés Python `and`/`or` ne fonctionnent pas avec les DataFrames.

2.4 Tri et classement

```
# Pays avec l'esperance de vie la plus elevee en 2007
recent = gapminder[gapminder["year"] == 2007]
top10 = recent.sort_values("lifeExp", ascending=False).head(10)
print(top10[["country", "lifeExp"]])
```

2.5 Regroupement et agrégation

Le regroupement est la façon de répondre à des questions comme « quelle est l'espérance de vie moyenne par continent ? » ou « combien de patients dans chaque catégorie de diagnostic ? »

```
# Esperance de vie moyenne par continent (2007)
recent.groupby("continent")["lifeExp"].mean().sort_values(ascending=False)
```

```
# Agregations multiples
recent.groupby("continent")["lifeExp"].agg(["mean", "median", "std", "count"])
```

```
# Nombre de patients par diagnostic
df.groupby("diagnosis")["patient_id"].count()
```

2.6 Créer de nouvelles colonnes

```python
# IMC a partir du poids et de la taille
df["bmi"] = df["weight_kg"] / (df["height_m"] ** 2)

# Groupe d'age
df["age_group"] = pd.cut(df["age"],
                         bins=[0, 18, 40, 60, 100],
                         labels=["<18", "18-39", "40-59", "60+"])

# Indicateur binaire
df["hypertensive"] = (df["systolic_bp"] >= 140).astype(int)
```

2.7 Sauvegarder vos résultats

```python
# Sauvegarder en CSV
df.to_csv("cleaned_patients.csv", index=False)

# Sauvegarder en Excel
df.to_excel("cleaned_patients.xlsx", index=False, sheet_name="Patients")
```

2.8 Travaux pratiques : analyse de l'espérance de vie (OMS)

Exercice

En utilisant le jeu de données Gapminder :

1. Chargez les données et affichez les statistiques de base.
2. Filtrez les pays africains uniquement. Combien de pays uniques ?
3. Calculez l'espérance de vie moyenne par décennie (années 1950, 1960, ..., 2000) pour l'Afrique par rapport à l'Europe.
4. Trouvez les 5 pays africains avec l'espérance de vie la plus basse en 2007.
5. Créez une colonne `life_exp_category` : « Basse » (< 55), « Moyenne » (55–70), « Élevée » (> 70).
6. Regroupez par continent et `life_exp_category`. Combien de pays dans chaque catégorie ?
7. Sauvegardez les données de l'Afrique uniquement dans un fichier CSV.

2.9 Résumé du chapitre

- Un DataFrame est un tableau. Lignes = observations, colonnes = variables.
- `pd.read_csv()` charge les données ; `.head()`, `.describe()`, `.info()` les explorent.
- Filtrez avec des conditions booléennes : `df[df["col"] > valeur]`.
- `.groupby()` + `.agg()` répond aux questions « par groupe ».
- `pd.cut()` crée des variables catégorielles à partir de variables continues.
- Sauvegardez toujours les données nettoyées pour la reproductibilité.

Chapitre 3

Nettoyage des données en contexte sanitaire

« Déchets en entrée, déchets en sortie. En données de santé, des déchets en entrée peuvent signifier de mauvais traitements, de mauvaises politiques et de mauvaises conclusions sur des vies humaines. »

3.1 Pourquoi le nettoyage des données est essentiel en santé

Chaque jeu de données de santé comporte des problèmes. Des valeurs de laboratoire sont saisies avec de mauvaises unités. Des patients apparaissent deux fois sous des identifiants différents. Les dates sont enregistrées comme `03/04/2023` — est-ce le 3 avril ou le 4 mars ? Une glycémie de 5000 mg/dL est clairement une erreur, mais une glycémie de 500 mg/dL pourrait être une véritable urgence diabétique.

Le nettoyage des données n'est pas un prétraitement optionnel. C'est une tâche de raisonnement clinique. Vous devez décider :

- Cette valeur est-elle *erronée* ou simplement *inhabituelle* ?
- Cet enregistrement doit-il être *corrigé*, *signalé* ou *supprimé* ?
- Mes décisions de nettoyage vont-elles *biaiser* l'analyse ?

Contexte clinique

Dans une étude de 2019 sur les dossiers médicaux électroniques d'un grand réseau hospitalier américain, les chercheurs ont constaté que 18 % des valeurs de laboratoire présentaient au moins un problème de qualité — doublons, valeurs physiologiquement impossibles ou unités incohérentes. Les décisions de nettoyage ont modifié la prévalence estimée de la maladie rénale chronique de plus de 3 points de pourcentage.

3.2 Données manquantes : types et mécanismes

3.2.1 Les trois mécanismes

Toutes les données manquantes ne se valent pas. Le mécanisme détermine ce que vous pouvez faire.

1. **MCAR — Manquantes complètement au hasard (*Missing Completely At Random*).** Le fait que la donnée manque n'a aucun lien avec les données. Exemple : un tube de sang est renversé et l'échantillon est perdu. La probabilité de perdre l'échantillon est indépendante de l'état de santé du patient.

2. **MAR — Manquantes au hasard (*Missing At Random*).** Le fait que la donnée manque dépend de variables *observées* mais pas de la valeur manquante elle-même. Exemple : les patients jeunes ont moins souvent un dosage du cholestérol (car les cliniciens le prescrivent moins fréquemment pour les jeunes patients). Une fois l'âge pris en compte, la donnée manquante est aléatoire.

3. **MNAR — Manquantes non au hasard (*Missing Not At Random*).** Le fait que la donnée manque dépend de la *valeur non observée* elle-même. Exemple : les patients très malades sont trop souffrants pour se présenter aux visites de suivi, donc leurs données de résultats manquent *parce qu'ils* sont malades. C'est le cas le plus difficile.

⚠ Attention

Le mécanisme MNAR est fréquent dans les données de santé et dangereux. Si vous supprimez tous les patients dont les données de suivi sont manquantes, vous excluez systématiquement les patients les plus malades — votre analyse surestimera le succès du traitement. Il n'existe pas de correction statistique pour le MNAR ; vous avez besoin de connaissances cliniques pour évaluer la direction et l'ampleur probables du biais.

3.2.2 Exemples cliniques de chaque mécanisme

Type	Exemple	Conséquence de la suppression
MCAR	Échantillon de sang hémolysé en raison de vibrations lors du transport	Aucun biais, mais taille d'échantillon réduite
MAR	HbA1c non prescrite pour les patients non diabétiques	Biais si vous étudiez l'HbA1c sans tenir compte de l'indication
MNAR	Les patients sévèrement déprimés ne remplissent pas les questionnaires de suivi	Sous-estimation de la sévérité de la dépression dans les données restantes

3.3 Détecter les valeurs manquantes avec pandas

3.3.1 Détection de base

```python
import pandas as pd
import numpy as np

# Creer un jeu de donnees clinique avec des valeurs manquantes
data = {
    "patient_id": ["P001", "P002", "P003", "P004", "P005", "P006"],
    "age": [45, 67, 34, np.nan, 71, 52],
    "sex": ["M", "F", "F", "M", np.nan, "F"],
    "systolic_bp": [128, 158, np.nan, 145, 162, np.nan],
    "hba1c": [5.4, 7.8, 5.1, np.nan, 8.2, 6.3],
    "smoking": ["Never", np.nan, "Current", "Former", np.nan, "Never"]
}
df = pd.DataFrame(data)

# Compter les valeurs manquantes par colonne
print(df.isnull().sum())

# Pourcentage de valeurs manquantes par colonne
print((df.isnull().sum() / len(df) * 100).round(1))

# Lignes avec au moins une valeur manquante
print(f"Lignes avec donnees manquantes : {df.isnull().any(axis=1).sum()} /
↪  {len(df)}")
```

Astuce Python

`.isnull()` et `.isna()` sont identiques dans pandas. Utilisez celui qui vous semble le plus lisible. Les deux détectent `NaN`, `None` et `NaT` (datetime manquant).

3.3.2 Visualiser les données manquantes avec msno

La bibliothèque `missingno` fournit des résumés visuels instantanés des schémas de données manquantes :

```python
import missingno as msno
import matplotlib.pyplot as plt

# Graphique matriciel : barres blanches = manquant
msno.matrix(df)
plt.title("Schema des donnees manquantes")
plt.tight_layout()
plt.show()

# Diagramme en barres : nombre de valeurs non nulles par colonne
msno.bar(df)
```

```
plt.tight_layout()
plt.show()

# Carte thermique : correlations entre les donnees manquantes de differentes
↪  colonnes
# Si deux colonnes tendent a etre manquantes ensemble, elles afficheront une
↪  correlation elevee
msno.heatmap(df)
plt.tight_layout()
plt.show()
```

Attention

Si `missingno` n'est pas installé, exécutez `!pip install missingno` dans votre notebook. Dans Google Colab, il n'est pas préinstallé.

3.3.3 Schémas de co-absence

Quand deux variables sont souvent manquantes ensemble, cela suggère une cause commune. Par exemple, si `systolic_bp` et `diastolic_bp` sont manquantes pour les mêmes patients, cela signifie probablement que la mesure de PA n'a pas été effectuée du tout — et non que chaque valeur a été perdue indépendamment.

```
# Verifier quelles colonnes sont manquantes ensemble
missing_pairs = df.isnull().corr()
print(missing_pairs)
```

3.4 Stratégies d'imputation

3.4.1 Quand imputer ou quand supprimer

- **Supprimer des lignes** si les données manquantes sont MCAR et la fraction manquante est faible ($< 5\,\%$).
- **Supprimer des colonnes** si $> 50\,\%$ des valeurs sont manquantes et la variable n'est pas critique.
- **Imputer** lorsque vous devez conserver la taille de l'échantillon et que vous disposez d'une stratégie raisonnable.
- **Signaler + analyse de sensibilité** pour les données MNAR.

```
# Supprimer les lignes avec toute valeur manquante
df_complete = df.dropna()
print(f"Lignes restantes : {len(df_complete)} / {len(df)}")

# Supprimer les lignes uniquement si des colonnes specifiques sont manquantes
df_partial = df.dropna(subset=["age", "systolic_bp"])
```

3.4.2 Imputation par la moyenne/médiane pour les variables continues

```
# Imputation par la mediane pour les valeurs de laboratoire (robuste aux
↪  valeurs aberrantes)
df["systolic_bp"] = df["systolic_bp"].fillna(df["systolic_bp"].median())
df["hba1c"] = df["hba1c"].fillna(df["hba1c"].median())

# Imputation par la moyenne pour l'age (distribution approximativement
↪  symetrique)
df["age"] = df["age"].fillna(df["age"].mean())
```

Contexte clinique

Utilisez la **médiane** plutôt que la moyenne pour les valeurs de laboratoire comme la créatinine, la glycémie et les triglycérides. Ces distributions sont typiquement asymétriques à droite — quelques valeurs très élevées tirent la moyenne vers le haut, ce qui en fait un mauvais représentant du patient « typique ». La médiane résiste à ces valeurs aberrantes.

3.4.3 Imputation par le mode pour les variables catégorielles

```
# Imputation par le mode pour le sexe et le statut tabagique
df["sex"] = df["sex"].fillna(df["sex"].mode()[0])
df["smoking"] = df["smoking"].fillna(df["smoking"].mode()[0])
```

3.4.4 Imputation spécifique par groupe

Parfois, la bonne valeur d'imputation dépend d'un groupe. Imputer la PA systolique avec la médiane globale ignore le fait que les patients âgés ont une PA plus élevée :

```
# Imputer systolic_bp avec la mediane du groupe d'age du patient
df["age_group"] = pd.cut(df["age"], bins=[0, 40, 60, 100],
                         labels=["<40", "40-59", "60+"])
df["systolic_bp"] = df.groupby("age_group")["systolic_bp"].transform(
    lambda x: x.fillna(x.median())
)
```

3.4.5 Imputation par KNN

L'imputation par les k plus proches voisins trouve les k patients les plus similaires (basés sur les autres variables) et utilise leurs valeurs pour combler la valeur manquante :

```
from sklearn.impute import KNNImputer
```

```
# Selectionner les colonnes numeriques pour l'imputation KNN
numeric_cols = ["age", "systolic_bp", "hba1c"]
imputer = KNNImputer(n_neighbors=5)
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])
```

Astuce Python

L'imputation KNN respecte les relations entre variables. Si un patient avec un IMC élevé, un âge avancé et un HbA1c élevé a une PA systolique manquante, KNN imputera à partir de patients similaires — qui ont probablement aussi une PA élevée. C'est plus réaliste que d'utiliser la médiane globale.

3.4.6 Créer un indicateur de donnée manquante

Pour les variables importantes, créez un indicateur afin de pouvoir tester ultérieurement si le fait d'être manquant est associé aux résultats :

```
# Avant d'imputer, creer un indicateur
df["bp_was_missing"] = df["systolic_bp"].isnull().astype(int)

# Puis imputer
df["systolic_bp"] = df["systolic_bp"].fillna(df["systolic_bp"].median())
```

3.5 Gestion des doublons

3.5.1 Détecter les doublons

```
# Charger un jeu de donnees avec des doublons deliberes
patients = pd.DataFrame({
    "patient_id": ["P001", "P002", "P003", "P002", "P004", "P003"],
    "visit_date": ["2023-01-15", "2023-01-16", "2023-01-17",
                   "2023-01-16", "2023-01-18", "2023-02-20"],
    "systolic_bp": [128, 158, 112, 158, 145, 118],
    "diagnosis": ["HTN", "T2DM", "None", "T2DM", "HTN", "None"]
})

# Doublons exacts (toutes les colonnes identiques)
print(f"Doublons exacts : {patients.duplicated().sum()}")
print(patients[patients.duplicated(keep=False)])  # afficher toutes les copies

# Doublons bases sur patient_id uniquement
print(f"\nIdentifiants patients en double :
↪  {patients.duplicated(subset='patient_id').sum()}")
```

3.5.2 Décider ce qui constitue un doublon

Dans les données de santé, « doublon » n'est pas toujours évident :

Scénario	Action appropriée
Même patient, même visite, valeurs identiques	Vrai doublon — supprimer une copie
Même patient, même visite, valeurs différentes	Erreur de saisie — vérifier quelle valeur est correcte
Même patient, dates de visite différentes	Mesures répétées — conserver tout (données longitudinales)
Identifiants différents, mêmes données démographiques	Possible double enregistrement — signaler pour révision manuelle

```
# Supprimer les doublons exacts
patients_clean = patients.drop_duplicates()
print(f"Apres suppression des doublons exacts : {len(patients_clean)} lignes")

# Ne garder que la premiere visite par patient
first_visits = patients.drop_duplicates(subset="patient_id", keep="first")
print(f"Premieres visites uniquement : {len(first_visits)} lignes")

# Ne garder que la derniere visite par patient
last_visits = patients.drop_duplicates(subset="patient_id", keep="last")
```

Contexte clinique

Dans les registres cliniques, les doublons de dossiers patients constituent un problème de sécurité majeur. Un patient enregistré sous deux identifiants peut recevoir des prescriptions contradictoires. La déduplication dans les systèmes de production utilise l'appariement probabiliste sur le nom, la date de naissance et l'adresse — bien au-delà de ce que `drop_duplicates()` peut faire.

3.6 Données incohérentes

3.6.1 Formats de codes CIM

La Classification internationale des maladies utilise des codes comme `E11.9` (Diabète de type 2 sans complication). Dans des données désordonnées, vous trouverez la même affection enregistrée comme `E11.9`, `E119`, `e11.9`, ou même `250.00` (l'ancien code CIM-9).

```
diagnoses = pd.Series(["E11.9", "e11.9", "E119", "E11.9 ", " e11.9"])

# Standardiser : majuscules, supprimer les espaces, assurer le format avec
↪  point
cleaned = (diagnoses
           .str.strip()
           .str.upper()
           .str.replace(r"^([A-Z]\d{2})(\d+)$", r"\1.\2", regex=True))
print(cleaned)
```

3.6.2 Formats de dates

```
dates = pd.Series(["2023-01-15", "01/15/2023", "15-Jan-2023",
                   "Jan 15, 2023", "20230115"])

# pd.to_datetime gere plusieurs formats automatiquement
parsed = pd.to_datetime(dates, format="mixed", dayfirst=False)
print(parsed)

# Standardiser au format ISO
print(parsed.dt.strftime("%Y-%m-%d"))
```

Attention

La date `03/04/2023` correspond au 4 mars aux États-Unis mais au 3 avril en Europe. Vérifiez toujours la source. Si votre jeu de données mélange les conventions, vous obtiendrez des erreurs silencieuses — la date sera interprétée sans avertissement, mais elle sera *fausse*. Utilisez `dayfirst=True` pour les dates au format européen.

3.6.3 Conversions d'unités

Différents laboratoires rapportent les résultats dans différentes unités. Les confusions les plus fréquentes :

Analyte	Unité US	Unité SI	Conversion
Glycémie	mg/dL	mmol/L	diviser par 18,018
Cholestérol	mg/dL	mmol/L	diviser par 38,67
Créatinine	mg/dL	µmol/L	multiplier par 88,42
Hémoglobine	g/dL	g/L	multiplier par 10

```
# Standardiser la glycemie en mmol/L
# Supposer que les valeurs > 30 sont en mg/dL (la glycemie en mmol/L depasse
↪  rarement 30)
def standardize_glucose(value, unit=None):
    """Convertir la glycemie en mmol/L. Inferer l'unite si non fournie."""
    if unit == "mg/dL" or (unit is None and value > 30):
        return value / 18.018
    return value  # deja en mmol/L

df["glucose_mmol"] = df["glucose"].apply(
    lambda x: standardize_glucose(x)
)
```

Astuce Python

Quand un jeu de données n'a pas de colonne d'unité, utilisez la plage de valeurs pour inférer l'unité. La glycémie à jeun en mg/dL est typiquement de 70 à 300 ; en mmol/L, de 3,9 à 16,7. Si vous voyez une valeur de 126, c'est presque certainement

en mg/dL. Si vous voyez 7,0, c'est presque certainement en mmol/L.

3.6.4 Standardiser les champs textuels

```
# Champ sexe avec des entrees incoherentes
sex_col = pd.Series(["Male", "male", "M", "m", "MALE", "Female",
                     "female", "F", "f", "FEMALE"])

# Mapper vers des valeurs standard
sex_mapping = {"male": "M", "m": "M", "female": "F", "f": "F"}
sex_clean = sex_col.str.strip().str.lower().map(sex_mapping)
print(sex_clean)
```

3.7 Validation des données

Après le nettoyage, validez que vos données ont un sens clinique.

3.7.1 Vérifications de plage

```
# Definir des plages cliniquement plausibles
ranges = {
    "age": (0, 120),
    "systolic_bp": (60, 300),
    "diastolic_bp": (30, 200),
    "heart_rate": (20, 300),
    "hba1c": (2.0, 20.0),
    "glucose_mgdl": (10, 1500),
    "bmi": (8, 80),
    "temperature_c": (25, 45),
}

def validate_range(df, column, low, high):
    """Signaler les valeurs en dehors de la plage plausible."""
    out_of_range = df[(df[column] < low) | (df[column] > high)]
    if len(out_of_range) > 0:
        print(f"ATTENTION : {len(out_of_range)} valeurs dans '{column}' "
              f"en dehors de [{low}, {high}]")
        print(out_of_range[["patient_id", column]])
    return out_of_range

# Appliquer les verifications de plage
for col, (low, high) in ranges.items():
    if col in df.columns:
        validate_range(df, col, low, high)
```

3.7.2 Validation croisée entre champs

Certaines erreurs ne deviennent visibles que lorsque vous comparez les colonnes :

```
# La diastolique doit etre inferieure a la systolique
bp_errors = df[df["diastolic_bp"] >= df["systolic_bp"]]
if len(bp_errors) > 0:
    print(f"ATTENTION : {len(bp_errors)} lignes ou diastolique >= systolique")

# L'indicateur de grossesse ne doit etre a 1 que pour les patientes
preg_errors = df[(df["pregnant"] == 1) & (df["sex"] == "M")]
if len(preg_errors) > 0:
    print(f"ATTENTION : {len(preg_errors)} patients masculins signales comme
    ↪ enceints")

# La date de deces doit etre posterieure a la date d'admission
date_errors = df[df["death_date"] < df["admission_date"]]
if len(date_errors) > 0:
    print(f"ATTENTION : {len(date_errors)} deces enregistres avant
    ↪ l'admission")
```

Contexte clinique

Toute valeur « impossible » n'est pas forcément une erreur. Une fréquence cardiaque de 250 bpm est physiologiquement extrême mais peut survenir lors d'une tachycardie supraventriculaire. Une glycémie de 800 mg/dL est rare mais réelle en cas d'acido-cétose diabétique. Les connaissances cliniques vous indiquent s'il faut supprimer, signaler ou conserver.

3.7.3 Rapport de validation automatisé

```
def data_quality_report(df):
    """Generer un resume des problemes de qualite des donnees."""
    report = []
    for col in df.columns:
        n_missing = df[col].isnull().sum()
        pct_missing = n_missing / len(df) * 100
        n_unique = df[col].nunique()

        row = {
            "column": col,
            "dtype": str(df[col].dtype),
            "n_missing": n_missing,
            "pct_missing": round(pct_missing, 1),
            "n_unique": n_unique,
        }

        if df[col].dtype in ["float64", "int64"]:
            row["min"] = df[col].min()
```

```
            row["max"] = df[col].max()
            row["mean"] = round(df[col].mean(), 2)

        report.append(row)

    return pd.DataFrame(report)

quality = data_quality_report(df)
print(quality.to_string(index=False))
```

3.8 Un pipeline de nettoyage complet

Voici un pipeline réaliste appliqué à des données de type NHANES :

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer

# ------------------------------------------------------------
# Etape 1 : Charger les donnees
# ------------------------------------------------------------
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
df = pd.read_csv(url)
print(f"Donnees brutes : {df.shape}")

# ------------------------------------------------------------
# Etape 2 : Standardiser les noms de colonnes
# ------------------------------------------------------------
df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")

# ------------------------------------------------------------
# Etape 3 : Supprimer les doublons exacts
# ------------------------------------------------------------
n_before = len(df)
df = df.drop_duplicates()
print(f"Doublons supprimes : {n_before - len(df)}")

# ------------------------------------------------------------
# Etape 4 : Evaluer les donnees manquantes
# ------------------------------------------------------------
missing_pct = (df.isnull().sum() / len(df) * 100).round(1)
print("Pourcentages de valeurs manquantes :")
print(missing_pct[missing_pct > 0])

# ------------------------------------------------------------
# Etape 5 : Valider les plages
# ------------------------------------------------------------
# L'esperance de vie doit etre entre 20 et 90
outliers = df[(df["lifeexp"] < 20) | (df["lifeexp"] > 90)]
```

```
print(f"Valeurs aberrantes d'esperance de vie : {len(outliers)}")

# ------------------------------------------------------------
# Etape 6 : Sauvegarder les donnees nettoyees
# ------------------------------------------------------------
df.to_csv("gapminder_cleaned.csv", index=False)
print(f"Donnees nettoyees sauvegardees : {df.shape}")
```

3.9 Travaux pratiques : nettoyage d'un jeu de données clinique désordonné

Exercice

Téléchargez ou créez un jeu de données clinique désordonné avec les erreurs délibérées suivantes, puis nettoyez-le étape par étape :

```
import pandas as pd
import numpy as np

# Donnees cliniques desordonnees avec des erreurs deliberees
messy = pd.DataFrame({
    "patient_id": ["P001", "P002", "P003", "P004", "P002",
                   "P005", "P006", "P007", "P008", "P009"],
    "age": [45, 67, -3, 52, 67, 200, 38, np.nan, 71, 44],
    "sex": ["M", "F", "Female", "m", "F",
            "Male", np.nan, "F", "X", "M"],
    "systolic_bp": [128, 158, 112, 450, 158,
                    135, 122, np.nan, 162, 118],
    "diastolic_bp": [82, 160, 74, 92, 160,
                     88, 78, np.nan, 98, 76],
    "glucose_mgdl": [95, 7.2, 110, 180, 95,
                     88, 102, 6.8, 220, 5.5],
    "hba1c": [5.4, 7.8, 5.1, 6.3, 7.8,
              np.nan, 5.8, 6.1, 55, 5.2],
    "visit_date": ["2023-01-15", "01/16/2023", "2023-01-17",
                   "2023/01/18", "2023-01-16",
                   "15-Jan-2023", "2023-01-20", "2023-01-21",
                   "2023-01-22", "Jan 23, 2023"],
    "icd_code": ["I10", "E119", "e11.9", "I10 ", " i10",
                 "E11.9", "I10", "e119", "I10", "E11.9"]
})
```

Tâches :

1. Générez un rapport de qualité des données : comptez les valeurs manquantes, vérifiez les types de données, calculez les statistiques descriptives.
2. Identifiez et supprimez les doublons exacts (P002 apparaît deux fois avec des données identiques).

3. Corrigez la colonne `sex` : standardisez en « M » et « F ». Décidez quoi faire avec « X » et `NaN`.
4. Validez `age` : signalez ou corrigez les valeurs impossibles (-3 et 200). Mettez-les à `NaN`.
5. Validez `systolic_bp` : une valeur de 450 est impossible. Mettez-la à `NaN`.
6. Vérifiez que diastolique $<$ systolique pour tous les patients. Signalez les violations.
7. Corrigez `glucose_mgdl` : certaines valeurs (7,2 ; 6,8 ; 5,5) sont clairement en mmol/L. Convertissez-les en mg/dL en multipliant par 18,018.
8. Corrigez `hba1c` : la valeur 55 est probablement en mmol/mol (IFCC) plutôt qu'en % (NGSP). Convertissez avec : NGSP $= 0{,}0915 \times$ IFCC $+ 2{,}15$.
9. Standardisez `visit_date` au format ISO (AAAA-MM-JJ).
10. Standardisez `icd_code` : majuscules, sans espaces, avec séparateur point.
11. Imputez les valeurs manquantes restantes : médiane pour les variables continues, mode pour les catégorielles.
12. Générez un rapport final de qualité des données et comparez avec le rapport initial.
13. Sauvegardez le jeu de données nettoyé sous `cleaned_clinical_data.csv`.

3.10 Résumé du chapitre

- Les données manquantes en santé suivent trois mécanismes : MCAR, MAR et MNAR. Le mécanisme détermine la stratégie appropriée.
- Utilisez `.isnull().sum()` pour des comptages rapides et `missingno` pour des visualisations des schémas.
- Options d'imputation : moyenne/médiane pour les données continues, mode pour les catégorielles, KNN pour l'imputation multivariée.
- Les doublons dans les données de santé peuvent être de vrais doublons, des erreurs de saisie ou des visites répétées — chacun nécessite une réponse différente.
- Le codage incohérent (formats CIM, formats de dates, unités) est omniprésent et doit être standardisé.
- Validez les données par rapport à des plages cliniquement plausibles *après* le nettoyage.
- Créez toujours un rapport de qualité des données avant et après le nettoyage pour documenter chaque décision.

Chapitre 4

Statistiques descriptives et mesures épidémiologiques

« Avant d'ajuster un modèle, décrivez les données. Avant de décrire les données, regardez les données. L'épidémiologie commence par le dénombrement. »

4.1 Tendance centrale : qu'est-ce qui est « typique » ?

4.1.1 Moyenne, médiane et quand utiliser chacune

```
import pandas as pd
import numpy as np

# Glycemie a jeun de 10 patients (mg/dL)
glucose = pd.Series([92, 98, 104, 95, 88, 110, 97, 340, 101, 93])

print(f"Moyenne :  {glucose.mean():.1f} mg/dL")
print(f"Mediane : {glucose.median():.1f} mg/dL")
```

La moyenne est de 121,8 mg/dL. La médiane est de 97,5 mg/dL. Un patient avec une glycémie de 340 (acidocétose diabétique) tire la moyenne vers le haut de près de 25 mg/dL. La médiane n'est pas affectée.

Contexte clinique

Règle générale pour les données de santé : utilisez la **médiane** lorsque la distribution est asymétrique (la plupart des valeurs de laboratoire, durée de séjour, dépenses de santé). Utilisez la **moyenne** lorsque la distribution est approximativement symétrique (taille, pression artérielle diastolique chez les adultes en bonne santé).

4.1.2 Mode

Le mode est la valeur la plus fréquente. C'est la seule mesure de tendance centrale pour les données catégorielles :

```
diagnoses = pd.Series(["HTN", "T2DM", "HTN", "None", "HTN",
                       "T2DM", "None", "HTN", "COPD", "HTN"])
print(f"Diagnostic le plus frequent : {diagnoses.mode()[0]}")
print(f"Frequence : {diagnoses.value_counts().iloc[0]}/{len(diagnoses)}")
```

4.2 Dispersion : quelle est la variabilité des données ?

4.2.1 Écart-type et variance

```
# Mesures de pression arterielle de deux cliniques
clinic_a = pd.Series([120, 122, 118, 125, 121, 119, 123, 120])
clinic_b = pd.Series([98, 145, 110, 155, 102, 160, 115, 135])

print(f"Clinique A : moyenne={clinic_a.mean():.1f}, ET={clinic_a.std():.1f}")
print(f"Clinique B : moyenne={clinic_b.mean():.1f}, ET={clinic_b.std():.1f}")
```

Les deux cliniques ont des moyennes similaires ($\approx$121), mais la clinique B présente une variabilité bien plus élevée. Un médecin ne regardant que la moyenne manquerait le fait que la clinique B a des patients avec une pression artérielle dangereusement élevée *et* anormalement basse.

4.2.2 Écart interquartile et percentiles

L'écart interquartile (EIQ) est l'intervalle entre le 25^{e} et le 75^{e} percentile. Comme la médiane, il est robuste aux valeurs aberrantes.

```
# Duree de sejour hospitalier (jours) --- typiquement asymetrique a droite
los = pd.Series([2, 3, 3, 4, 4, 5, 5, 6, 7, 8, 12, 28, 45])

q25 = los.quantile(0.25)
q75 = los.quantile(0.75)
iqr = q75 - q25

print(f"Duree de sejour mediane : {los.median():.0f} jours")
print(f"EIQ : {q25:.0f}--{q75:.0f} jours (etendue : {iqr:.0f})")
print(f"Duree de sejour moyenne : {los.mean():.1f} jours")  # gonflee par 28 et
↪ 45
```

Astuce Python

La méthode `.describe()` de pandas vous donne le nombre, la moyenne, l'écart-type, le minimum, les 25 %, 50 %, 75 % et le maximum en un seul appel. C'est le moyen le plus rapide d'obtenir un aperçu statistique de chaque colonne numérique.

4.2.3 Coefficient de variation

Le coefficient de variation (CV) exprime l'écart-type en pourcentage de la moyenne. Il est utile pour comparer la variabilité entre des mesures ayant des unités ou des échelles différentes :

```
# Comparer la variabilite de la glycemie (mg/dL) et de l'hemoglobine (g/dL)
glucose = pd.Series([95, 102, 88, 110, 97, 105, 92, 98])
hemoglobin = pd.Series([13.2, 14.1, 12.8, 13.5, 14.0, 13.8, 12.5, 13.9])

cv_glucose = (glucose.std() / glucose.mean()) * 100
cv_hb = (hemoglobin.std() / hemoglobin.mean()) * 100

print(f"CV glycemie : {cv_glucose:.1f}%")
print(f"CV hemoglobine : {cv_hb:.1f}%")
```

4.3 Tableaux de fréquences et tableaux croisés

4.3.1 Tableaux de fréquences simples

```
# Charger les donnees Gapminder
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
gapminder = pd.read_csv(url)
recent = gapminder[gapminder["year"] == 2007]

# Creer des categories d'esperance de vie
recent = recent.copy()
recent["le_cat"] = pd.cut(recent["lifeExp"],
                          bins=[0, 55, 70, 85],
                          labels=["Basse (<55)", "Moyenne (55-70)",
                                  "Elevee (>70)"])

# Tableau de frequences
freq = recent["le_cat"].value_counts().sort_index()
print(freq)

# Avec proportions
freq_table = pd.DataFrame({
    "n": freq,
    "pct": (freq / freq.sum() * 100).round(1),
    "cum_pct": (freq.cumsum() / freq.sum() * 100).round(1)
})
print(freq_table)
```

4.3.2 Tableaux croisés

```
# Tableau croise : categorie d'esperance de vie par continent
```

```python
ct = pd.crosstab(recent["continent"], recent["le_cat"], margins=True)
print(ct)

# Avec pourcentages en ligne (pourcentage au sein de chaque continent)
ct_pct = pd.crosstab(recent["continent"], recent["le_cat"],
                     normalize="index").round(3) * 100
print(ct_pct)
```

Contexte clinique

Les tableaux croisés sont l'outil de base de l'épidémiologie descriptive. Chaque « Tableau 1 » dans un article clinique est essentiellement un tableau croisé des caractéristiques des patients par groupe d'étude. Avec pandas, `pd.crosstab()` les construit en une seule ligne.

4.4 Mesures épidémiologiques : dénombrer la maladie

4.4.1 Prévalence

La prévalence est la *proportion* d'une population qui présente une affection à un moment donné :

$$\text{Prévalence} = \frac{\text{Nombre de cas existants}}{\text{Population totale}} \times 100$$

```python
# Prevalence du diabete a partir d'une enquete de sante
n_total = 5000
n_diabetic = 625

prevalence = n_diabetic / n_total * 100
print(f"Prevalence du diabete : {prevalence:.1f}%")

# Avec intervalle de confiance a 95% (approximation normale)
from scipy import stats

p = n_diabetic / n_total
se = np.sqrt(p * (1 - p) / n_total)
ci_low = (p - 1.96 * se) * 100
ci_high = (p + 1.96 * se) * 100
print(f"IC a 95% : [{ci_low:.1f}%, {ci_high:.1f}%]")
```

4.4.2 Taux d'incidence

L'incidence mesure les *nouveaux* cas sur une période donnée. Le taux d'incidence utilise les personnes-temps au dénominateur :

$$\text{Taux d'incidence} = \frac{\text{Nombre de nouveaux cas}}{\text{Total personnes-temps à risque}}$$

```python
# Cohorte tuberculose : 200 patients suivis pendant des durees variables
new_tb_cases = 18
total_person_years = 850  # somme du temps de suivi de tous les patients

incidence_rate = new_tb_cases / total_person_years
print(f"Taux d'incidence TB : {incidence_rate:.4f} par personne-annee")
print(f"                    = {incidence_rate * 1000:.1f} pour 1 000
↪  personnes-annees")

# IC a 95% pour le taux d'incidence (approximation de Poisson)
ir_se = np.sqrt(new_tb_cases) / total_person_years
ci_low = (incidence_rate - 1.96 * ir_se) * 1000
ci_high = (incidence_rate + 1.96 * ir_se) * 1000
print(f"IC a 95% : [{ci_low:.1f}, {ci_high:.1f}] pour 1 000 personnes-annees")
```

Attention

La prévalence et l'incidence répondent à des questions différentes. La prévalence indique la *charge* de morbidité (combien de personnes sont malades en ce moment). L'incidence indique le *risque* de tomber malade. Une maladie peut avoir une faible incidence mais une prévalence élevée si les patients survivent longtemps (par ex. le diabète de type 2). Une maladie avec une incidence élevée et une faible prévalence tue rapidement (par ex. Ebola).

4.4.3 Taux de mortalité

```python
# Taux de mortalite brut
deaths = 1250
population = 500_000
mortality_rate = deaths / population * 100_000
print(f"Taux de mortalite brut : {mortality_rate:.1f} pour 100 000")

# Taux de letalite (TL)
total_cases = 3200
deaths_among_cases = 128
cfr = deaths_among_cases / total_cases * 100
print(f"Taux de letalite : {cfr:.1f}%")
```

Contexte clinique

Le taux de létalité (TL) est souvent mal interprété lors des épidémies. Au début d'une épidémie, le dénominateur (nombre total de cas) est sous-estimé car de nombreux cas bénins ne sont pas testés. Cela fait paraître le TL plus élevé que le véritable taux de mortalité par infection (TMI). Au début de la COVID-19, le TL rapporté était de ∼3–4 %, mais les études de séroprévalence ont ensuite estimé le TMI à ∼0,5–1 %.

4.5 Mesures d'association

4.5.1 Risque relatif

Le risque relatif compare la probabilité de maladie entre les groupes exposé et non exposé :

$$\text{RR} = \frac{\text{Risque chez les exposés}}{\text{Risque chez les non-exposés}} = \frac{a/(a+b)}{c/(c+d)}$$

```
# Tabagisme et cancer du poumon (cohorte hypothetique)
# Exposes (fumeurs) :      a=80 cancers, b=920 sans cancer
# Non-exposes (non-fumeurs) : c=10 cancers, d=990 sans cancer
a, b, c, d = 80, 920, 10, 990

risk_exposed = a / (a + b)
risk_unexposed = c / (c + d)
rr = risk_exposed / risk_unexposed

print(f"Risque chez les fumeurs :     {risk_exposed:.3f}
↪  ({risk_exposed*100:.1f}%)")
print(f"Risque chez les non-fumeurs : {risk_unexposed:.3f}
↪  ({risk_unexposed*100:.1f}%)")
print(f"Risque relatif : {rr:.2f}")

# IC a 95% pour log(RR)
import math
se_ln_rr = math.sqrt(1/a - 1/(a+b) + 1/c - 1/(c+d))
ln_rr = math.log(rr)
ci_low = math.exp(ln_rr - 1.96 * se_ln_rr)
ci_high = math.exp(ln_rr + 1.96 * se_ln_rr)
print(f"IC a 95% : [{ci_low:.2f}, {ci_high:.2f}]")
```

4.5.2 Rapport des cotes (*odds ratio*)

Le rapport des cotes est utilisé dans les études cas-témoins où l'on ne peut pas calculer directement le risque :

$$\text{OR} = \frac{a \times d}{b \times c}$$

```
# Etude cas-temoins : obesite et gonarthrose
# Cas (arthrose) :     a=120 obeses, c=80 non-obeses
# Temoins (pas d'arthrose) : b=60 obeses,  d=140 non-obeses
a, b, c, d = 120, 60, 80, 140

odds_ratio = (a * d) / (b * c)
print(f"Rapport des cotes : {odds_ratio:.2f}")

# IC a 95%
se_ln_or = math.sqrt(1/a + 1/b + 1/c + 1/d)
```

```
ln_or = math.log(odds_ratio)
ci_low = math.exp(ln_or - 1.96 * se_ln_or)
ci_high = math.exp(ln_or + 1.96 * se_ln_or)
print(f"IC a 95% : [{ci_low:.2f}, {ci_high:.2f}]")
```

Contexte clinique

Le rapport des cotes approxime le risque relatif lorsque la maladie est rare ($< 10\,\%$ de prévalence dans les deux groupes). Pour les événements fréquents, l'OR exagère l'association. Un OR de 2,0 pour une affection avec 40 % de prévalence de base ne signifie pas « deux fois le risque » — le véritable RR serait plus faible.

4.5.3 Fonctions réutilisables

```
def risk_ratio(a, b, c, d, alpha=0.05):
    """Calculer le risque relatif avec intervalle de confiance.

    a = exposes avec maladie
    b = exposes sans maladie
    c = non-exposes avec maladie
    d = non-exposes sans maladie
    """
    from scipy.stats import norm
    z = norm.ppf(1 - alpha / 2)

    rr = (a / (a + b)) / (c / (c + d))
    se = math.sqrt(1/a - 1/(a+b) + 1/c - 1/(c+d))
    ci = (math.exp(math.log(rr) - z * se),
          math.exp(math.log(rr) + z * se))

    return {"RR": round(rr, 3), "CI_low": round(ci[0], 3),
            "CI_high": round(ci[1], 3)}


def odds_ratio(a, b, c, d, alpha=0.05):
    """Calculer le rapport des cotes avec intervalle de confiance."""
    from scipy.stats import norm
    z = norm.ppf(1 - alpha / 2)

    o_r = (a * d) / (b * c)
    se = math.sqrt(1/a + 1/b + 1/c + 1/d)
    ci = (math.exp(math.log(o_r) - z * se),
          math.exp(math.log(o_r) + z * se))

    return {"OR": round(o_r, 3), "CI_low": round(ci[0], 3),
            "CI_high": round(ci[1], 3)}


# Exemple
print(risk_ratio(80, 920, 10, 990))
```

```
print(odds_ratio(120, 60, 80, 140))
```

4.6 Taux standardisés sur l'âge

Les taux bruts peuvent être trompeurs lorsqu'on compare des populations avec des structures d'âge différentes. Un pays avec une population plus âgée aura un taux de mortalité brut plus élevé même s'il est « en meilleure santé » à chaque âge. La standardisation sur l'âge élimine ce facteur de confusion.

4.6.1 Standardisation directe

```
# Comparer les taux de mortalite bruts et standardises entre deux regions
# Region A (population jeune), Region B (population agee)

data_a = pd.DataFrame({
    "age_group": ["0-14", "15-44", "45-64", "65+"],
    "population": [50000, 80000, 40000, 10000],
    "deaths": [50, 80, 200, 300]
})
data_a["rate"] = data_a["deaths"] / data_a["population"] * 100000

data_b = pd.DataFrame({
    "age_group": ["0-14", "15-44", "45-64", "65+"],
    "population": [15000, 40000, 50000, 45000],
    "deaths": [20, 50, 280, 1500]
})
data_b["rate"] = data_b["deaths"] / data_b["population"] * 100000

# Population standard (ex. : population standard mondiale de l'OMS)
standard_pop = pd.DataFrame({
    "age_group": ["0-14", "15-44", "45-64", "65+"],
    "std_weight": [0.26, 0.42, 0.22, 0.10]
})

# Taux bruts
crude_a = data_a["deaths"].sum() / data_a["population"].sum() * 100000
crude_b = data_b["deaths"].sum() / data_b["population"].sum() * 100000
print(f"Taux brut Region A : {crude_a:.1f} pour 100 000")
print(f"Taux brut Region B : {crude_b:.1f} pour 100 000")

# Taux standardises sur l'age
asr_a = (data_a["rate"] * standard_pop["std_weight"]).sum()
asr_b = (data_b["rate"] * standard_pop["std_weight"]).sum()
print(f"Taux standardise Region A : {asr_a:.1f} pour 100 000")
print(f"Taux standardise Region B : {asr_b:.1f} pour 100 000")
```

> **⚠ Attention**
>
> Indiquez toujours quelle population standard vous avez utilisée (population standard mondiale de l'OMS, standard européen, standard US 2000). Des standards différents donnent des résultats différents. Si vous comparez vos taux avec des taux publiés, vous devez utiliser la même population standard.

4.7 Travaux pratiques : calcul de statistiques descriptives avec des données réelles

4.7.1 Statistiques descriptives sur les données Gapminder

```python
import pandas as pd
import numpy as np
from scipy import stats

# Charger Gapminder
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
df = pd.read_csv(url)

# Statistiques descriptives par continent (2007)
recent = df[df["year"] == 2007].copy()
summary = recent.groupby("continent")["lifeExp"].agg(
    ["count", "mean", "median", "std",
     lambda x: x.quantile(0.25),
     lambda x: x.quantile(0.75)]
)
summary.columns = ["n", "mean", "median", "sd", "Q1", "Q3"]
print(summary.round(1))
```

4.7.2 Calcul de la prévalence du paludisme par région

```python
# Cas estimes de paludisme (Our World in Data)
url = ("https://raw.githubusercontent.com/owid/"
       "owid-datasets/master/datasets/"
       "Malaria%20incidence%20-%20IHME/Malaria%20incidence%20-%20IHME.csv")
# Alternative : utiliser directement les donnees du GHO de l'OMS
# url = "https://apps.who.int/gho/athena/api/GHO/MALARIA_EST_CASES?format=csv"

# Si l'URL ci-dessus ne fonctionne pas, utiliser le GitHub d'OWID
url2 = ("https://raw.githubusercontent.com/owid/etl/master/etl/steps/"
        "data/garden/who/2024-07-30/gho.py")  # metadonnees uniquement

# Pour une source fiable, utiliser le jeu de donnees OWID pretraite sur le
↪  paludisme
owid_url = ("https://raw.githubusercontent.com/owid/"
```

CHAPITRE 4. STATISTIQUES DESCRIPTIVES ET MESURES ÉPIDÉMIOLOGIQUES

```
            "owid-datasets/master/datasets/"
            "Estimated%20malaria%20incidence%20rate%20"
            "(per%201%2C000%20population%20at%20risk)%20-%20WHO/"
            "Estimated%20malaria%20incidence%20rate%20"
            "(per%201%2C000%20population%20at%20risk)%20-%20WHO.csv")

# Donnees de paludisme simulees pour la pratique
np.random.seed(42)
countries = (["Nigeria", "DRC", "Mozambique", "Uganda", "Ghana"] * 3 +
             ["India", "Indonesia", "Myanmar", "Pakistan", "Ethiopia"] * 3)
regions = (["West Africa"] * 5 + ["Central Africa"] * 5 +
           ["East Africa"] * 5 +
           ["South Asia"] * 5 + ["Southeast Asia"] * 5 +
           ["East Africa"] * 5)

malaria = pd.DataFrame({
    "country": countries,
    "region": ["West Africa", "Central Africa", "East Africa",
               "East Africa", "West Africa"] * 6,
    "year": [2018]*5 + [2019]*5 + [2020]*5 +
            [2018]*5 + [2019]*5 + [2020]*5,
    "population_at_risk": np.random.randint(10_000_000, 200_000_000, 30),
    "estimated_cases": np.random.randint(500_000, 50_000_000, 30)
})

# Calculer la prevalence pour 1 000 personnes a risque
malaria["prevalence_per_1000"] = (
    malaria["estimated_cases"] / malaria["population_at_risk"] * 1000
)

# Resume par region
region_summary = malaria.groupby("region").agg(
    total_cases=("estimated_cases", "sum"),
    total_pop=("population_at_risk", "sum"),
    n_countries=("country", "nunique")
)
region_summary["prevalence_per_1000"] = (
    region_summary["total_cases"] / region_summary["total_pop"] * 1000
)
print(region_summary.round(1))

# Tendance par annee
year_trend = malaria.groupby("year").agg(
    total_cases=("estimated_cases", "sum"),
    total_pop=("population_at_risk", "sum")
)
year_trend["prevalence_per_1000"] = (
    year_trend["total_cases"] / year_trend["total_pop"] * 1000
)
print(year_trend.round(1))
```

Exercice

En utilisant le jeu de données Gapminder et les fonctions épidémiologiques définies ci-dessus :

1. Calculez les statistiques descriptives (moyenne, médiane, ET, EIQ) pour l'espérance de vie, le PIB par habitant et la population pour chaque continent en 2007.

2. Créez un tableau croisé continent / catégorie d'espérance de vie (Basse/Moyenne/Élevée). Calculez les pourcentages en ligne.

3. Une cohorte de 10 000 personnes a été suivie pendant 5 ans. Durant cette période, 450 ont développé un diabète de type 2. Calculez l'incidence cumulée et le taux d'incidence (en supposant un suivi moyen de 4,5 années par personne).

4. Dans une étude cas-témoins de 200 cas et 200 témoins, 140 cas et 80 témoins étaient exposés à un facteur de risque. Calculez le rapport des cotes et son IC à 95 %. Interprétez le résultat.

5. Le pays A a un taux de mortalité brut de 350 pour 100 000 et le pays B de 580 pour 100 000. En utilisant les taux spécifiques par âge et la population standard fournis ci-dessus, calculez les taux standardisés. Le classement change-t-il après standardisation ?

6. En utilisant les données de Our World in Data ou de l'OMS, calculez le taux d'incidence du paludisme pour 5 pays africains sur 3 ans. Quel pays a la charge la plus élevée ? La tendance est-elle croissante ou décroissante ?

7. Écrivez une fonction `prevalence_ci(cases, total, confidence=0.95)` qui retourne la prévalence et son intervalle de confiance. Testez-la avec : 250 patients diabétiques sur 2 000 dépistés.

4.8 Résumé du chapitre

- Utilisez la **médiane** pour les distributions asymétriques (valeurs de laboratoire, coûts, durée de séjour) et la **moyenne** pour les distributions symétriques.

- L'**EIQ** est une mesure robuste de dispersion ; l'**écart-type** est sensible aux valeurs aberrantes.

- **Prévalence** = cas existants / population. **Taux d'incidence** = nouveaux cas / personnes-temps.

- Le **risque relatif** compare les risques dans les études de cohorte ; le **rapport des cotes** est utilisé dans les études cas-témoins.

- L'OR approxime le RR uniquement lorsque l'événement est rare ($< 10\,\%$).

- La **standardisation sur l'âge** élimine le facteur de confusion lié à l'âge lors de la comparaison de populations.

CHAPITRE 4. STATISTIQUES DESCRIPTIVES ET MESURES ÉPIDÉMIOLOGIQUES

- Rapportez toujours les intervalles de confiance en plus des estimations ponctuelles.
- `pd.crosstab()` construit les tableaux croisés qui forment la base de l'analyse épidémiologique.

Chapitre 5

Visualisation pour la santé

« La plus grande valeur d'une image, c'est lorsqu'elle nous force à remarquer ce que nous ne nous attendions pas à voir. » — John Tukey

5.1 Principes de la visualisation de données de santé

Une bonne figure de santé répond à une question. Une mauvaise décore une diapositive. Avant d'écrire le moindre code de graphique, posez-vous ces questions :

1. **Quel est le message ?** « L'incidence du paludisme diminue en Afrique subsaharienne » — pas « voici des données ».
2. **Qui est le public ?** Les cliniciens ont besoin de graphiques différents de ceux des décideurs politiques.
3. **Quelle comparaison compte ?** Entre groupes ? Au fil du temps ? Par rapport à un seuil ?

Règles applicables à toute figure de santé :

- Étiquetez les axes avec les unités (« PA systolique (mmHg) », pas « PA »).
- Utilisez des palettes adaptées aux daltoniens.
- N'utilisez pas de graphiques 3D, de camemberts ou de doubles axes Y, sauf raison très justifiée.
- Incluez les tailles d'échantillon dans les titres ou les annotations.
- Commencez l'axe Y à zéro pour les diagrammes en barres. Pour les courbes, zéro est optionnel si la plage est étroite.

Contexte clinique

Dans les publications cliniques, les figures sont souvent la première chose (et parfois la seule) que les évaluateurs examinent attentivement. Une figure claire et bien étiquetée peut porter un article. Une figure trompeuse peut le couler. The Lancet et le BMJ ont tous deux des directives spécifiques pour les figures — étudiez-les avant de soumettre.

5.2 Mise en place : matplotlib et seaborn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Definir un style propre pour tous les graphiques
sns.set_theme(style="whitegrid", font_scale=1.1)
plt.rcParams["figure.figsize"] = (8, 5)
plt.rcParams["figure.dpi"] = 100

# Palette adaptee aux daltoniens
cb_palette = ["#0072B2", "#D55E00", "#009E73",
              "#CC79A7", "#F0E442", "#56B4E9"]
sns.set_palette(cb_palette)
```

Astuce Python

La palette de six couleurs ci-dessus provient de l'article de Bang Wong « Points of view : Color blindness » (Nature Methods, 2011). Elle est distinguable par les personnes atteintes des formes les plus courantes de déficience de la vision des couleurs. Utilisez-la comme palette par défaut.

5.3 Histogrammes et courbes de densité

Les histogrammes montrent la *distribution* d'une seule variable. En santé, cela vous indique si une mesure suit une distribution normale, est asymétrique ou bimodale.

```
# Charger les donnees Gapminder
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
gapminder = pd.read_csv(url)
recent = gapminder[gapminder["year"] == 2007]

# Histogramme de l'esperance de vie
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Gauche : histogramme
axes[0].hist(recent["lifeExp"], bins=20, edgecolor="white", alpha=0.8)
axes[0].set_xlabel("Esperance de vie (annees)")
axes[0].set_ylabel("Nombre de pays")
axes[0].set_title("Distribution de l'esperance de vie, 2007")
axes[0].axvline(recent["lifeExp"].median(), color="red",
                linestyle="--", label=f'Mediane :
                ↪ {recent["lifeExp"].median():.1f}')
axes[0].legend()
```

```
# Droite : estimation de densite par noyau (version lissee)
sns.kdeplot(data=recent, x="lifeExp", hue="continent",
            fill=True, alpha=0.3, ax=axes[1])
axes[1].set_xlabel("Esperance de vie (annees)")
axes[1].set_title("Esperance de vie par continent, 2007")

plt.tight_layout()
plt.show()
```

```
# Exemple clinique : distribution de la glycemie a jeun
np.random.seed(42)
normal_glucose = np.random.normal(95, 10, 800)
prediabetic = np.random.normal(115, 8, 150)
diabetic = np.random.normal(180, 40, 50)
all_glucose = np.concatenate([normal_glucose, prediabetic, diabetic])

fig, ax = plt.subplots(figsize=(10, 5))
ax.hist(all_glucose, bins=40, edgecolor="white", alpha=0.7)
ax.axvline(100, color="orange", linestyle="--", linewidth=2,
           label="Seuil normal (100 mg/dL)")
ax.axvline(126, color="red", linestyle="--", linewidth=2,
           label="Seuil diabetique (126 mg/dL)")
ax.set_xlabel("Glycemie a jeun (mg/dL)")
ax.set_ylabel("Nombre de patients")
ax.set_title("Distribution de la glycemie a jeun (n=1 000)")
ax.legend()
plt.tight_layout()
plt.show()
```

5.4 Boîtes à moustaches

Les boîtes à moustaches comparent les distributions entre groupes. La boîte montre l'EIQ (25^{e}–75^{e} percentile), la ligne est la médiane, et les moustaches s'étendent à 1,5×EIQ. Les points au-delà des moustaches sont des valeurs aberrantes.

```
# Pression arterielle par continent
fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(data=recent, x="continent", y="lifeExp", ax=ax)
ax.set_xlabel("Continent")
ax.set_ylabel("Esperance de vie (annees)")
ax.set_title("Esperance de vie par continent, 2007 (n={})".format(len(recent)))
plt.tight_layout()
plt.show()
```

```
# Exemple clinique : IMC par sexe et groupe d'age
np.random.seed(42)
```

```
n = 500
clinical = pd.DataFrame({
    "sex": np.random.choice(["Male", "Female"], n),
    "age_group": np.random.choice(["18-39", "40-59", "60+"], n,
                                  p=[0.3, 0.4, 0.3]),
    "bmi": np.random.normal(27, 5, n).clip(15, 50)
})

fig, ax = plt.subplots(figsize=(10, 6))
sns.boxplot(data=clinical, x="age_group", y="bmi", hue="sex",
            order=["18-39", "40-59", "60+"], ax=ax)
ax.axhline(25, color="orange", linestyle="--", alpha=0.7,
           label="Seuil surpoids")
ax.axhline(30, color="red", linestyle="--", alpha=0.7,
           label="Seuil obesite")
ax.set_xlabel("Groupe d'age")
ax.set_ylabel("IMC (kg/m2)")
ax.set_title(f"IMC par sexe et groupe d'age (n={n})")
ax.legend()
plt.tight_layout()
plt.show()
```

Attention

Les boîtes à moustaches masquent la forme de la distribution. Une distribution bimodale apparaît identique à une distribution unimodale dans une boîte à moustaches. Envisagez d'utiliser un **graphique en violon** (`sns.violinplot`) ou un **strip plot** (`sns.stripplot`) superposé à la boîte pour les petits échantillons.

5.5 Diagrammes en barres

Les diagrammes en barres comparent des *effectifs* ou des *taux* entre catégories. Ils sont le choix standard pour comparer les prévalences de maladies.

```
# Comparaison de l'esperance de vie : 15 pays les plus bas (2007)
bottom15 = recent.nsmallest(15, "lifeExp")

fig, ax = plt.subplots(figsize=(10, 7))
ax.barh(bottom15["country"], bottom15["lifeExp"], color="#D55E00")
ax.set_xlabel("Esperance de vie (annees)")
ax.set_title("15 pays avec l'esperance de vie la plus basse, 2007")
ax.invert_yaxis()  # le plus eleve en haut
for i, v in enumerate(bottom15["lifeExp"]):
    ax.text(v + 0.5, i, f"{v:.1f}", va="center", fontsize=9)
plt.tight_layout()
plt.show()
```

```
# Comparaison de prevalence de maladies par pays (simule)
```

```
diseases = pd.DataFrame({
    "country": ["Nigeria", "DRC", "Tanzania", "Kenya", "Ghana",
                "South Africa", "Ethiopia", "Uganda"],
    "malaria_prev": [27.3, 31.2, 7.5, 5.8, 15.2, 0.5, 1.2, 9.7],
    "hiv_prev": [1.3, 0.7, 4.7, 4.9, 1.7, 19.0, 1.0, 5.4],
})

fig, ax = plt.subplots(figsize=(10, 6))
x = np.arange(len(diseases))
width = 0.35
ax.bar(x - width/2, diseases["malaria_prev"], width,
       label="Paludisme", color="#0072B2")
ax.bar(x + width/2, diseases["hiv_prev"], width,
       label="VIH", color="#D55E00")
ax.set_xticks(x)
ax.set_xticklabels(diseases["country"], rotation=45, ha="right")
ax.set_ylabel("Prevalence (%)")
ax.set_title("Prevalence du paludisme vs VIH par pays")
ax.legend()
plt.tight_layout()
plt.show()
```

5.6 Courbes temporelles et courbes épidémiques

Les courbes temporelles montrent comment une mesure évolue dans le temps. Elles sont essentielles pour suivre les tendances des maladies et la progression des épidémies.

5.6.1 Tendances temporelles

```
# Tendances de l'esperance de vie par continent
continent_trend = gapminder.groupby(
    ["year", "continent"])["lifeExp"].mean().reset_index()

fig, ax = plt.subplots(figsize=(10, 6))
for continent in continent_trend["continent"].unique():
    subset = continent_trend[continent_trend["continent"] == continent]
    ax.plot(subset["year"], subset["lifeExp"],
            marker="o", markersize=4, label=continent)

ax.set_xlabel("Annee")
ax.set_ylabel("Esperance de vie moyenne (annees)")
ax.set_title("Tendances de l'esperance de vie par continent, 1952-2007")
ax.legend(title="Continent")
plt.tight_layout()
plt.show()
```

5.6.2 Courbes épidémiques

```
# Simuler une courbe epidemique (forme de type SIR)
np.random.seed(42)
days = pd.date_range("2023-01-01", periods=120, freq="D")
# Simuler les nouveaux cas quotidiens avec un pic de type SIR
t = np.arange(120)
peak = 45
cases = np.maximum(0, 200 * np.exp(-0.5 * ((t - peak) / 15) ** 2)
                   + np.random.normal(0, 10, 120)).astype(int)

epi = pd.DataFrame({"date": days, "new_cases": cases})
epi["rolling_7d"] = epi["new_cases"].rolling(7).mean()
epi["cumulative"] = epi["new_cases"].cumsum()

fig, axes = plt.subplots(2, 1, figsize=(12, 8), sharex=True)

# Haut : cas quotidiens + moyenne sur 7 jours
axes[0].bar(epi["date"], epi["new_cases"], alpha=0.4,
            color="#0072B2", label="Cas quotidiens")
axes[0].plot(epi["date"], epi["rolling_7d"], color="#D55E00",
             linewidth=2, label="Moyenne sur 7 jours")
axes[0].set_ylabel("Nouveaux cas par jour")
axes[0].set_title("Courbe epidemique : nouveaux cas quotidiens")
axes[0].legend()

# Bas : cas cumules
axes[1].plot(epi["date"], epi["cumulative"], color="#009E73",
             linewidth=2)
axes[1].set_xlabel("Date")
axes[1].set_ylabel("Cas cumules")
axes[1].set_title("Nombre cumule de cas")

plt.tight_layout()
plt.show()
```

Contexte clinique

Lors des épidémies, la courbe épidémique (« epi curve ») est la visualisation la plus importante en santé publique. Sa forme révèle le mode de transmission : une épidémie de source ponctuelle présente un pic aigu ; une source continue produit un plateau ; une transmission de personne à personne génère des vagues successives. La moyenne glissante sur 7 jours lisse les délais de déclaration et les effets de week-end.

5.7 Nuages de points

Les nuages de points montrent la relation entre deux variables continues.

```
# PIB par habitant vs esperance de vie (le graphique classique de Gapminder)
```

```
fig, ax = plt.subplots(figsize=(10, 7))
for continent in recent["continent"].unique():
    subset = recent[recent["continent"] == continent]
    ax.scatter(subset["gdpPercap"], subset["lifeExp"],
               s=subset["pop"] / 1e6,  # taille de bulle = population
               alpha=0.6, label=continent)

ax.set_xscale("log")
ax.set_xlabel("PIB par habitant (echelle log, USD)")
ax.set_ylabel("Esperance de vie (annees)")
ax.set_title("Richesse vs Sante, 2007 (taille de bulle = population)")
ax.legend(title="Continent")
plt.tight_layout()
plt.show()
```

```
# Nuage de points clinique : IMC vs PA systolique avec droite de regression
np.random.seed(42)
n = 200
bmi = np.random.normal(28, 5, n).clip(18, 45)
sbp = 80 + 1.8 * bmi + np.random.normal(0, 12, n)

fig, ax = plt.subplots(figsize=(8, 6))
ax.scatter(bmi, sbp, alpha=0.5, s=30, color="#0072B2")

# Ajouter la droite de regression
from numpy.polynomial.polynomial import polyfit
b, m = polyfit(bmi, sbp, 1)
x_line = np.linspace(bmi.min(), bmi.max(), 100)
ax.plot(x_line, b + m * x_line, color="#D55E00", linewidth=2,
        label=f"y = {m:.1f}x + {b:.1f}")

# Coefficient de correlation
from scipy.stats import pearsonr
r, p = pearsonr(bmi, sbp)
ax.set_xlabel("IMC (kg/m2)")
ax.set_ylabel("PA systolique (mmHg)")
ax.set_title(f"IMC vs PA systolique (r={r:.2f}, p<0.001, n={n})")
ax.legend()
plt.tight_layout()
plt.show()
```

Astuce Python

Les fonctions `sns.regplot()` et `sns.lmplot()` de seaborn ajoutent automatiquement des droites de régression et des bandes de confiance. Utilisez `sns.lmplot()` lorsque vous souhaitez séparer par une variable catégorielle (par ex. régression par sexe).

5.8 Courbes de survie de Kaplan-Meier

L'analyse de survie suit le temps jusqu'à un événement (décès, rechute, infection). La courbe de Kaplan-Meier estime la probabilité de survivre au-delà de chaque point temporel.

```
# Installer lifelines si necessaire : !pip install lifelines
from lifelines import KaplanMeierFitter

# Donnees de survie simulees : deux groupes de traitement
np.random.seed(42)
n_per_group = 100

# Groupe traitement : survie plus longue
treatment_time = np.random.exponential(24, n_per_group).clip(0, 60)
treatment_event = np.random.binomial(1, 0.7, n_per_group)

# Groupe temoin : survie plus courte
control_time = np.random.exponential(15, n_per_group).clip(0, 60)
control_event = np.random.binomial(1, 0.8, n_per_group)

fig, ax = plt.subplots(figsize=(10, 6))

# Ajuster et tracer le groupe traitement
kmf_treatment = KaplanMeierFitter()
kmf_treatment.fit(treatment_time, event_observed=treatment_event,
                  label="Traitement (n=100)")
kmf_treatment.plot_survival_function(ax=ax, ci_show=True)

# Ajuster et tracer le groupe temoin
kmf_control = KaplanMeierFitter()
kmf_control.fit(control_time, event_observed=control_event,
                label="Temoin (n=100)")
kmf_control.plot_survival_function(ax=ax, ci_show=True)

ax.set_xlabel("Temps (mois)")
ax.set_ylabel("Probabilite de survie")
ax.set_title("Courbes de survie de Kaplan-Meier par groupe de traitement")
ax.set_ylim(0, 1.05)

# Ajouter les lignes de survie mediane
for kmf, color in [(kmf_treatment, "#0072B2"), (kmf_control, "#D55E00")]:
    median = kmf.median_survival_time_
    if not np.isinf(median):
        ax.axhline(0.5, color="gray", linestyle=":", alpha=0.5)
        ax.axvline(median, color=color, linestyle=":", alpha=0.5)

plt.tight_layout()
plt.show()

# Afficher la survie mediane
print(f"Survie mediane (traitement) : "
```

```
      f"{kmf_treatment.median_survival_time_:.1f} mois")
print(f"Survie mediane (temoin) : "
      f"{kmf_control.median_survival_time_:.1f} mois")
```

Attention

Si `lifelines` n'est pas installé, exécutez `!pip install lifelines` dans votre notebook. Il n'est pas préinstallé dans Google Colab.

Contexte clinique

Les courbes de Kaplan-Meier gèrent les observations **censurées** — patients encore en vie à la fin de l'étude ou perdus de vue. Sans censure adéquate, vous sous-estimeriez la survie. Les marques verticales sur la courbe indiquent les observations censurées.

5.9 Cartes thermiques

Les cartes thermiques affichent des matrices de valeurs à l'aide de l'intensité des couleurs. Deux usages courants en santé :

5.9.1 Matrices de corrélation

```
# Correlation entre indicateurs de sante
np.random.seed(42)
n = 300
health = pd.DataFrame({
    "IMC": np.random.normal(27, 5, n),
    "PA_Systolique": np.random.normal(130, 18, n),
    "PA_Diastolique": np.random.normal(82, 10, n),
    "Glycemie": np.random.normal(105, 25, n),
    "Cholesterol": np.random.normal(200, 40, n),
    "HbA1c": np.random.normal(5.8, 1.2, n),
    "Freq_Cardiaque": np.random.normal(72, 12, n),
    "Age": np.random.normal(55, 15, n).clip(18, 90)
})

# Ajouter des correlations realistes
health["PA_Systolique"] += 0.8 * health["IMC"]
health["Glycemie"] += 15 * health["HbA1c"]
health["PA_Diastolique"] += 0.4 * health["PA_Systolique"]
health["Cholesterol"] += 0.5 * health["IMC"]

corr = health.corr()

fig, ax = plt.subplots(figsize=(9, 8))
mask = np.triu(np.ones_like(corr, dtype=bool))  # masquer le triangle superieur
sns.heatmap(corr, mask=mask, annot=True, fmt=".2f",
            cmap="RdBu_r", center=0, vmin=-1, vmax=1,
```

```
            square=True, ax=ax)
ax.set_title("Matrice de correlation des indicateurs de sante")
plt.tight_layout()
plt.show()
```

5.9.2 Co-occurrence de maladies

```
# Co-occurrence de pathologies chroniques
conditions = ["Hypertension", "Diabete", "BPCO",
              "Insuffisance cardiaque", "IRC", "Obesite"]
np.random.seed(42)
cooccurrence = np.random.randint(5, 60, (6, 6))
cooccurrence = (cooccurrence + cooccurrence.T) // 2  # rendre symetrique
np.fill_diagonal(cooccurrence, [320, 180, 95, 75, 110, 210])

co_df = pd.DataFrame(cooccurrence,
                     index=conditions, columns=conditions)

fig, ax = plt.subplots(figsize=(9, 8))
sns.heatmap(co_df, annot=True, fmt="d", cmap="YlOrRd",
            square=True, ax=ax)
ax.set_title("Matrice de co-occurrence des maladies\n"
             "(diagonale = total des cas, hors diagonale = co-occurrence)")
plt.tight_layout()
plt.show()
```

5.10 Travaux pratiques : construire un tableau de bord santé à 4 panneaux

Combiner plusieurs graphiques dans une seule figure est essentiel pour les rapports et publications. Voici un tableau de bord complet à 4 panneaux utilisant des données réelles :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_theme(style="whitegrid", font_scale=1.0)

# Charger les donnees
url = ("https://raw.githubusercontent.com/datasets/"
       "gapminder/main/data/gapminder.csv")
df = pd.read_csv(url)
recent = df[df["year"] == 2007].copy()

# Creer la figure a 4 panneaux
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
```

```
fig.suptitle("Tableau de bord sante mondiale, 2007",
             fontsize=16, fontweight="bold", y=1.02)

# ---------------------------------------------------------
# Panneau A : Distribution de l'esperance de vie par continent
# ---------------------------------------------------------
ax = axes[0, 0]
sns.boxplot(data=recent, x="continent", y="lifeExp", ax=ax,
            palette="Set2")
ax.set_xlabel("")
ax.set_ylabel("Esperance de vie (annees)")
ax.set_title("A. Esperance de vie par continent")
ax.tick_params(axis="x", rotation=30)

# ---------------------------------------------------------
# Panneau B : PIB vs Esperance de vie
# ---------------------------------------------------------
ax = axes[0, 1]
for continent in recent["continent"].unique():
    subset = recent[recent["continent"] == continent]
    ax.scatter(subset["gdpPercap"], subset["lifeExp"],
               s=subset["pop"] / 2e6, alpha=0.6,
               label=continent)
ax.set_xscale("log")
ax.set_xlabel("PIB par habitant (USD, echelle log)")
ax.set_ylabel("Esperance de vie (annees)")
ax.set_title("B. Richesse vs sante")
ax.legend(fontsize=8, title="Continent", title_fontsize=8)

# ---------------------------------------------------------
# Panneau C : Tendances de l'esperance de vie
# ---------------------------------------------------------
ax = axes[1, 0]
trend = df.groupby(["year", "continent"])["lifeExp"].mean().reset_index()
for continent in trend["continent"].unique():
    subset = trend[trend["continent"] == continent]
    ax.plot(subset["year"], subset["lifeExp"],
            marker="o", markersize=3, label=continent)
ax.set_xlabel("Annee")
ax.set_ylabel("Esperance de vie moyenne (annees)")
ax.set_title("C. Tendances de l'esperance de vie, 1952-2007")
ax.legend(fontsize=8, title="Continent", title_fontsize=8)

# ---------------------------------------------------------
# Panneau D : 10 pays avec l'esperance de vie la plus basse
# ---------------------------------------------------------
ax = axes[1, 1]
bottom10 = recent.nsmallest(10, "lifeExp")
ax.barh(bottom10["country"], bottom10["lifeExp"], color="#D55E00")
ax.set_xlabel("Esperance de vie (annees)")
ax.set_title("D. 10 pays avec l'esperance de vie la plus basse")
ax.invert_yaxis()
```

```
for i, v in enumerate(bottom10["lifeExp"]):
    ax.text(v + 0.3, i, f"{v:.1f}", va="center", fontsize=8)

plt.tight_layout()
plt.savefig("health_dashboard_2007.png", dpi=150, bbox_inches="tight")
plt.show()
print("Tableau de bord sauvegarde sous health_dashboard_2007.png")
```

Exercice

1. Modifiez le tableau de bord pour afficher les données de 2002 au lieu de 2007. L'histoire change-t-elle ?

2. Ajoutez un 5^{e} panneau montrant un histogramme du PIB par habitant (transformé en log) avec une ligne verticale à la médiane mondiale. Utilisez `fig, axes = plt.subplots(2, 3)` ou `plt.subplot2grid()`.

3. Créez un tableau de bord clinique pour une cohorte simulée de 500 patients avec les panneaux suivants :

   - Histogramme de l'IMC avec les seuils de surpoids/obésité
   - Boîte à moustaches de la PA systolique par sexe
   - Nuage de points âge vs HbA1c avec une ligne de seuil diabétique à 6,5 %
   - Diagramme en barres de la fréquence des diagnostics

4. En utilisant Our World in Data (https://github.com/owid/owid-datasets), téléchargez un jeu de données sur le paludisme ou la tuberculose. Construisez un tableau de bord épidémique avec :

   - Courbe d'incidence au fil du temps pour 5 pays
   - Diagramme en barres de l'incidence de l'année la plus récente
   - Nuage de points de l'incidence vs le PIB
   - Carte thermique de la corrélation entre indicateurs de santé

5. Créez un graphique de Kaplan-Meier comparant la survie entre 3 groupes de traitement (médicament A, médicament B, placebo) à partir de données simulées. Ajoutez des annotations de survie médiane.

6. Construisez une visualisation « avant/après intervention » : montrez un indicateur de santé (par ex. cas de paludisme) sous forme de courbe avec une ligne verticale marquant la date d'intervention. Coloriez les périodes avant et après intervention avec des couleurs différentes.

5.11 Résumé du chapitre

- Étiquetez toujours les axes avec les unités, incluez les tailles d'échantillon et utilisez des palettes adaptées aux daltoniens.

- Les **histogrammes** montrent les distributions ; ajoutez des lignes de seuil clinique pour donner du contexte.
- Les **boîtes à moustaches** comparent les groupes ; envisagez des graphiques en violon pour plus de détail.
- Les **diagrammes en barres** comparent des effectifs ou des taux ; utilisez des barres horizontales pour de nombreuses catégories.
- Les **courbes temporelles** montrent les tendances ; ajoutez des moyennes glissantes sur 7 jours pour les courbes épidémiques.
- Les **nuages de points** révèlent les relations ; ajoutez des droites de régression et rapportez les valeurs de r.
- Les **courbes de Kaplan-Meier** sont la norme pour les données de survie ; utilisez la bibliothèque `lifelines`.
- Les **cartes thermiques** affichent les matrices de corrélation et les schémas de co-occurrence.
- Les figures multi-panneaux (`plt.subplots()`) combinent des graphiques reliés en une histoire cohérente.
- Sauvegardez les figures de qualité publication avec `plt.savefig("fichier.png", dpi=300, bbox_inches="tight")`.

CHAPITRE 5. VISUALISATION POUR LA SANTÉ

Chapitre 6

Tests d'hypothèses

« L'objectif d'un test statistique n'est pas de prouver que quelque chose est vrai. C'est de mesurer à quel point nous devrions être surpris si rien ne se passait. »

6.1 La logique des tests d'hypothèses

Chaque question clinique peut être formulée comme un test entre deux énoncés concurrents :

- **Hypothèse nulle (H_0) :** Il n'y a *aucun* effet, aucune différence, aucune association. Le médicament ne fonctionne pas. Les deux groupes ont la même pression artérielle.
- **Hypothèse alternative (H_1) :** Il *y a* un effet, une différence ou une association. Le médicament abaisse le cholestérol. Le groupe traitement a une pression artérielle plus basse que le groupe témoin.

Nous collectons des données, calculons une statistique de test et demandons : *si H_0 était vraie, quelle est la probabilité d'observer des données aussi extrêmes ?* Cette probabilité est la **valeur** p.

Contexte clinique

La formulation clinique importe. Un statisticien écrit $H_0 : \mu_1 = \mu_2$. Un clinicien écrit « il n'y a pas de différence de pression artérielle systolique entre le groupe statine et le groupe placebo ». Les deux disent la même chose — utilisez la formulation qui rend la question de recherche claire.

6.1.1 Les étapes d'un test d'hypothèse

1. **Énoncer les hypothèses** (H_0 et H_1).
2. **Choisir le niveau de signification** α (généralement 0,05).
3. **Collecter les données** et calculer la **statistique de test**.
4. **Calculer la valeur** p — la probabilité d'observer un résultat au moins aussi extrême que le vôtre, en supposant H_0 vraie.

5. **Décider :** si $p < \alpha$, rejeter H_0. Sinon, ne pas rejeter H_0.

```
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
```

6.2 Valeurs p : ce qu'elles signifient et ce qu'elles ne signifient pas

6.2.1 Ce qu'est une valeur p

Une valeur p est la probabilité d'observer des données aussi extrêmes (ou plus extrêmes) que celles collectées, *en supposant* H_0 vraie. Un $p = 0{,}03$ signifie : si le médicament n'a vraiment aucun effet, il y a 3 % de chances de voir des résultats aussi extrêmes par le seul hasard.

6.2.2 Ce qu'une valeur p n'est PAS

- Ce n'est **pas** la probabilité que H_0 soit vraie.
- Ce n'est **pas** la probabilité que le résultat soit dû au hasard.
- Elle ne mesure **pas** la taille de l'effet.
- $p = 0{,}049$ n'est pas fondamentalement différent de $p = 0{,}051$.

⚠ Attention

Une petite valeur p ne signifie pas un effet *important* ou *cliniquement significatif*. Une étude avec 100 000 patients peut trouver $p < 0{,}001$ pour une différence de pression artérielle de 0,5 mmHg — statistiquement significatif mais cliniquement sans intérêt. Rapportez toujours la **taille d'effet** en plus des valeurs p.

6.2.3 Erreurs de type I et de type II

- **Erreur de type I (faux positif) :** Vous rejetez H_0 alors qu'elle est vraie. Vous concluez que le médicament fonctionne alors que ce n'est pas le cas. Contrôlée par α.
- **Erreur de type II (faux négatif) :** Vous ne rejetez pas H_0 alors que H_1 est vraie. Vous manquez un vrai effet. Liée à la **puissance** $(1 - \beta)$.

```
# Visualiser la logique : distribution d'echantillonnage sous H0
np.random.seed(42)
null_distribution = np.random.normal(loc=0, scale=1, size=10000)
```

```
fig, ax = plt.subplots(figsize=(9, 4))
ax.hist(null_distribution, bins=60, density=True, alpha=0.7, color="steelblue")
ax.axvline(1.96, color="red", linestyle="--", label="Valeur critique
↪  (alpha=0.05)")
ax.axvline(-1.96, color="red", linestyle="--")
ax.fill_betweenx([0, 0.45], 1.96, 3.5, alpha=0.3, color="red", label="Zone de
↪  rejet")
ax.fill_betweenx([0, 0.45], -3.5, -1.96, alpha=0.3, color="red")
ax.set_xlabel("Statistique de test (z)")
ax.set_ylabel("Densite")
ax.set_title("Test bilateral : zones de rejet sous H0")
ax.legend()
plt.tight_layout()
plt.show()
```

6.3 Test t à un échantillon

Question : Le cholestérol total moyen de nos patients est-il différent de la moyenne nationale de 200 mg/dL ?

```
# Donnees de cholesterol simulees de la clinique (mg/dL)
np.random.seed(42)
cholesterol = np.random.normal(loc=212, scale=35, size=80)

# Test t a un echantillon : la moyenne est-elle differente de 200 ?
t_stat, p_value = stats.ttest_1samp(cholesterol, popmean=200)
print(f"Moyenne echantillon : {cholesterol.mean():.1f} mg/dL")
print(f"Statistique t : {t_stat:.3f}")
print(f"Valeur p : {p_value:.4f}")

if p_value < 0.05:
    print("Rejeter H0 : le cholesterol moyen differe de 200 mg/dL")
else:
    print("Ne pas rejeter H0 : pas de preuve que la moyenne differe de 200
    ↪  mg/dL")
```

Contexte clinique

Le test t à un échantillon compare une moyenne d'échantillon à une valeur de référence connue. En pratique clinique, la référence peut être une moyenne nationale, un seuil de recommandation ou une valeur de base avant traitement.

6.4 Test t à deux échantillons

Question : La pression artérielle systolique est-elle différente entre le groupe traitement et le groupe témoin ?

```python
# Donnees d'essai clinique simulees
np.random.seed(42)
treatment = np.random.normal(loc=128, scale=15, size=60)
control = np.random.normal(loc=138, scale=16, size=55)

# Test t a deux echantillons independants
t_stat, p_value = stats.ttest_ind(treatment, control)
print(f"Moyenne traitement : {treatment.mean():.1f} mmHg")
print(f"Moyenne temoin :     {control.mean():.1f} mmHg")
print(f"Difference :         {control.mean() - treatment.mean():.1f} mmHg")
print(f"Statistique t :      {t_stat:.3f}")
print(f"Valeur p :           {p_value:.4f}")
```

6.4.1 Vérification des hypothèses

Le test t suppose une normalité approximative et une égalité des variances. Vérifiez les deux :

```python
# Normalite : test de Shapiro-Wilk (H0 : les donnees sont normalement
↪  distribuees)
_, p_treat = stats.shapiro(treatment)
_, p_ctrl = stats.shapiro(control)
print(f"Shapiro-Wilk p (traitement) : {p_treat:.4f}")
print(f"Shapiro-Wilk p (temoin) :     {p_ctrl:.4f}")

# Egalite des variances : test de Levene
_, p_levene = stats.levene(treatment, control)
print(f"Test de Levene p : {p_levene:.4f}")

# Si les variances sont inegales, utiliser le test t de Welch :
t_stat, p_value = stats.ttest_ind(treatment, control, equal_var=False)
print(f"Test t de Welch valeur p : {p_value:.4f}")
```

Astuce Python

Mettez `equal_var=False` dans `ttest_ind()` pour utiliser le test t de Welch. Il ne suppose pas l'égalité des variances et constitue souvent le choix le plus sûr par défaut.

6.5 Test t apparié

Question : Un programme d'exercice de 12 semaines réduit-il la pression artérielle systolique ?

Chaque patient est mesuré *avant* et *après*. Le test t apparié utilise les différences intra-patient.

```python
# Donnees simulees avant/apres intervention
```

```
np.random.seed(42)
n_patients = 45
bp_before = np.random.normal(loc=142, scale=12, size=n_patients)
# Apres intervention : reduction moyenne de ~8 mmHg avec variation individuelle
bp_after = bp_before - np.random.normal(loc=8, scale=6, size=n_patients)

# Test t apparie
t_stat, p_value = stats.ttest_rel(bp_before, bp_after)
differences = bp_before - bp_after
print(f"PA moyenne avant :   {bp_before.mean():.1f} mmHg")
print(f"PA moyenne apres :   {bp_after.mean():.1f} mmHg")
print(f"Reduction moyenne :  {differences.mean():.1f} mmHg")
print(f"Statistique t :      {t_stat:.3f}")
print(f"Valeur p :           {p_value:.6f}")
```

```
# Visualiser avant/apres
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Lignes appariees
for i in range(n_patients):
    axes[0].plot([0, 1], [bp_before[i], bp_after[i]],
                 color="gray", alpha=0.4)
axes[0].plot([0, 1], [bp_before.mean(), bp_after.mean()],
             color="red", linewidth=3, marker="o", markersize=8)
axes[0].set_xticks([0, 1])
axes[0].set_xticklabels(["Avant", "Apres"])
axes[0].set_ylabel("PA systolique (mmHg)")
axes[0].set_title("Trajectoires individuelles des patients")

# Distribution des differences
axes[1].hist(differences, bins=15, edgecolor="black", alpha=0.7,
↪  color="steelblue")
axes[1].axvline(0, color="red", linestyle="--", label="Aucun changement")
axes[1].axvline(differences.mean(), color="green", linestyle="--",
                label=f"Moyenne = {differences.mean():.1f}")
axes[1].set_xlabel("Reduction de PA (mmHg)")
axes[1].set_ylabel("Frequence")
axes[1].set_title("Distribution des changements de PA")
axes[1].legend()

plt.tight_layout()
plt.show()
```

Attention

Une erreur courante est d'utiliser un test t à deux échantillons *indépendants* sur des données avant/après. Cela ignore l'appariement et réduit la puissance statistique. Si les mêmes patients sont mesurés deux fois, utilisez toujours un test t **apparié**.

6.6 Test du chi-deux d'indépendance

Question : La prévalence du diabète est-elle associée à la catégorie d'obésité ?

Le test du chi-deux fonctionne avec des variables *catégorielles*. Il compare les effectifs observés à ceux attendus si les variables étaient indépendantes.

```
# Donnees de tableau croise simulees
np.random.seed(42)
n = 500
bmi_cat = np.random.choice(["Normal", "Surpoids", "Obese"],
                           size=n, p=[0.3, 0.4, 0.3])
# La probabilite de diabete augmente avec l'obesite
diabetes_prob = {"Normal": 0.08, "Surpoids": 0.18, "Obese": 0.35}
diabetes = [np.random.binomial(1, diabetes_prob[b]) for b in bmi_cat]

df_chi = pd.DataFrame({"bmi_category": bmi_cat, "diabetes": diabetes})
df_chi["diabetes_label"] = df_chi["diabetes"].map({0: "Non", 1: "Oui"})

# Tableau de contingence
contingency = pd.crosstab(df_chi["bmi_category"], df_chi["diabetes_label"])
print(contingency)
print()

# Test du chi-deux
chi2, p_value, dof, expected = stats.chi2_contingency(contingency)
print(f"Statistique du chi-deux : {chi2:.2f}")
print(f"Degres de liberte :       {dof}")
print(f"Valeur p :                {p_value:.4f}")
print(f"\nFrequences attendues (si independance) :")
print(pd.DataFrame(expected, index=contingency.index,
                   columns=contingency.columns).round(1))
```

Contexte clinique

Le test du chi-deux exige que les effectifs attendus par cellule soient d'au moins 5. Si vous avez de petites cellules (par ex. une maladie rare), utilisez le test exact de Fisher : `stats.fisher_exact(table)` pour les tableaux 2×2.

6.7 Test U de Mann-Whitney

Quand vos données ne sont pas normalement distribuées — fréquent pour la durée de séjour, les données de coût ou les petits échantillons — utilisez le test U de Mann-Whitney. Il compare les *rangs* des observations plutôt que les moyennes.

```
# Duree de sejour : chirurgical vs medical (donnees asymetriques a droite)
np.random.seed(42)
los_surgical = np.random.exponential(scale=7, size=40)
los_medical = np.random.exponential(scale=4.5, size=45)
```

```python
# Test U de Mann-Whitney
u_stat, p_value = stats.mannwhitneyu(los_surgical, los_medical,
                                     alternative="two-sided")
print(f"Duree de sejour mediane (chirurgical) : {np.median(los_surgical):.1f}
↪  jours")
print(f"Duree de sejour mediane (medical) :     {np.median(los_medical):.1f}
↪  jours")
print(f"Statistique U :                         {u_stat:.0f}")
print(f"Valeur p :                              {p_value:.4f}")
```

```python
# Visualiser les distributions asymetriques
fig, ax = plt.subplots(figsize=(8, 4))
ax.hist(los_surgical, bins=15, alpha=0.6, label="Chirurgical", color="coral")
ax.hist(los_medical, bins=15, alpha=0.6, label="Medical", color="steelblue")
ax.set_xlabel("Duree de sejour (jours)")
ax.set_ylabel("Frequence")
ax.set_title("Duree de sejour par service")
ax.legend()
plt.tight_layout()
plt.show()
```

Astuce Python

Règle générale pour le choix d'un test :

- Données normales, deux groupes → test t
- Données non normales, deux groupes → test U de Mann-Whitney
- Données catégorielles → test du chi-deux
- Mesures appariées → test t apparié (ou test de Wilcoxon si non normal)

6.8 Correction pour tests multiples

6.8.1 Le problème

Si vous testez 20 hypothèses à $\alpha = 0{,}05$, vous attendez $20 \times 0{,}05 = 1$ faux positif *même s'il ne se passe rien*. En génomique, vous pourriez tester 20 000 gènes simultanément. Sans correction, vous obtiendriez 1 000 « découvertes » fallacieuses.

```python
# Simulation : 1000 tests ou H0 est vraie pour tous
np.random.seed(42)
n_tests = 1000
p_values = []
for _ in range(n_tests):
    # Deux echantillons aleatoires de la MEME distribution (H0 est vraie)
```

```
    a = np.random.normal(0, 1, 30)
    b = np.random.normal(0, 1, 30)
    _, p = stats.ttest_ind(a, b)
    p_values.append(p)

p_values = np.array(p_values)
false_positives = (p_values < 0.05).sum()
print(f"Tests effectues : {n_tests}")
print(f"Faux positifs (p < 0.05) : {false_positives}")
print(f"Taux de faux positifs : {false_positives/n_tests:.1%}")
```

6.8.2 Correction de Bonferroni

La correction la plus simple : diviser α par le nombre de tests.

```
from statsmodels.stats.multitest import multipletests

# Correction de Bonferroni
rejected_bonf, pvals_corrected_bonf, _, _ = multipletests(
    p_values, alpha=0.05, method="bonferroni"
)
print(f"Bonferroni : {rejected_bonf.sum()} resultats significatifs")
```

6.8.3 Benjamini-Hochberg (FDR)

Moins conservative : contrôle le *taux de fausses découvertes* (la proportion d'hypothèses rejetées qui sont des faux positifs).

```
# Correction FDR de Benjamini-Hochberg
rejected_fdr, pvals_corrected_fdr, _, _ = multipletests(
    p_values, alpha=0.05, method="fdr_bh"
)
print(f"FDR (BH) : {rejected_fdr.sum()} resultats significatifs")
```

Contexte clinique

En génomique et protéomique, la correction FDR est le standard. Quand vous lisez un article rapportant « 542 gènes différentiellement exprimés à FDR $< 0{,}05$ », les auteurs ont utilisé Benjamini-Hochberg (ou une méthode similaire) pour contrôler les milliers de tests simultanés.

Attention

Bonferroni est très conservative — elle minimise les faux positifs mais peut manquer de vrais effets (erreur de type II élevée). Le FDR est un bon compromis pour les analyses exploratoires. Pour les essais cliniques confirmatoires, Bonferroni est souvent préférée car les faux positifs sont plus coûteux.

6.9 Taille d'effet : signification clinique vs statistique

6.9.1 d de Cohen

Le d de Cohen mesure la taille de la différence en *unités d'écart-type* :

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s_{\text{groupé}}}$$

- $|d| < 0{,}2$: effet négligeable
- $0{,}2 \leq |d| < 0{,}5$: petit effet
- $0{,}5 \leq |d| < 0{,}8$: effet moyen
- $|d| \geq 0{,}8$: grand effet

```
def cohens_d(group1, group2):
    """Calculer le d de Cohen pour deux groupes independants."""
    n1, n2 = len(group1), len(group2)
    var1, var2 = group1.var(ddof=1), group2.var(ddof=1)
    pooled_std = np.sqrt(((n1 - 1) * var1 + (n2 - 1) * var2) / (n1 + n2 - 2))
    return (group1.mean() - group2.mean()) / pooled_std

# Avec nos donnees PA traitement vs temoin
d = cohens_d(treatment, control)
print(f"d de Cohen : {d:.2f}")
print(f"Interpretation : {'grand' if abs(d) >= 0.8 else 'moyen' if abs(d) >=
↪  0.5 else 'petit' if abs(d) >= 0.2 else 'negligeable'} effet")
```

6.9.2 Pourquoi la taille d'effet est importante

```
# Grand echantillon + effet minuscule = valeur p significative
np.random.seed(42)
huge_group1 = np.random.normal(120.0, 15, size=50000)
huge_group2 = np.random.normal(120.5, 15, size=50000)  # seulement 0.5 mmHg de
↪  difference

t, p = stats.ttest_ind(huge_group1, huge_group2)
d = cohens_d(huge_group1, huge_group2)
print(f"Difference de moyennes : {huge_group2.mean() - huge_group1.mean():.2f}
↪  mmHg")
print(f"Valeur p : {p:.6f}")
print(f"d de Cohen : {d:.3f}")
print("Statistiquement significatif ? Oui. Cliniquement pertinent ? Non.")
```

Contexte clinique

Dans un rapport d'essai clinique, la FDA et l'EMA attendent à la fois des valeurs p *et* des intervalles de confiance pour l'effet du traitement. Un médicament qui abaisse l'HbA1c de 0,01 % avec $p < 0{,}001$ ne sera pas approuvé. L'effet doit être **cliniquement significatif**.

6.10 Travaux pratiques : tests d'hypothèses avec des données de type Framingham

```
# Charger un jeu de donnees cardiovasculaire de type Framingham
url = ("https://raw.githubusercontent.com/dsrscientist/"
       "dataset-for-machine-learning/master/framingham.csv")
fram = pd.read_csv(url)
print(f"Dimensions : {fram.shape}")
print(f"Colonnes : {fram.columns.tolist()}")
fram.head()
```

```
# Exploration rapide
fram.describe()
```

```
# Supprimer les lignes avec valeurs manquantes par simplicite
fram_clean = fram.dropna()
print(f"Jeu de donnees nettoye : {fram_clean.shape[0]} patients")
```

Exercice

En utilisant le jeu de données Framingham :

1. **Test t à un échantillon.** Le cholestérol total moyen de cette cohorte diffère-t-il de la moyenne nationale américaine de 200 mg/dL ?
2. **Test t à deux échantillons.** Comparez la pression artérielle systolique entre les patients ayant développé une coronaropathie (`TenYearCHD = 1`) et ceux qui ne l'ont pas fait. Rapportez la valeur p, la différence de moyennes et le d de Cohen.
3. **Test du chi-deux.** Créez un tableau de contingence 2×2 de `currentSmoker` vs `TenYearCHD`. Le tabagisme est-il associé au risque de coronaropathie à 10 ans ?
4. **Test U de Mann-Whitney.** Comparez `cigsPerDay` entre les groupes coronaropathie et non-coronaropathie. Pourquoi le test de Mann-Whitney est-il plus approprié qu'un test t ici ?
5. **Tests multiples.** Exécutez des tests t comparant coronaropathie vs non-coronaropathie pour toutes les variables continues (age, totChol, sysBP,

diaBP, BMI, heartRate, glucose). Appliquez les corrections de Bonferroni et FDR. Quelles variables restent significatives après chaque correction ?

6. **Interprétation clinique.** Pour la variable avec la plus grande taille d'effet (d de Cohen), discutez si la différence est cliniquement significative.

6.11 Résumé du chapitre

- Un test d'hypothèse mesure à quel point les données seraient surprenantes si H_0 était vraie.
- La valeur p n'est *pas* la probabilité que H_0 soit vraie.
- Test t à un échantillon : comparer une moyenne d'échantillon à une valeur de référence.
- Test t à deux échantillons : comparer les moyennes entre deux groupes indépendants.
- Test t apparié : comparer les moyennes des mêmes sujets mesurés deux fois.
- Test du chi-deux : tester l'association entre deux variables catégorielles.
- Test U de Mann-Whitney : alternative non paramétrique quand les données ne sont pas normales.
- La correction pour tests multiples (Bonferroni, FDR) empêche les fausses découvertes.
- La taille d'effet (d de Cohen) distingue la signification statistique de la signification clinique.
- Rapportez toujours les valeurs p et les tailles d'effet dans la recherche clinique.

Chapitre 7

Régression pour les résultats de santé

« La corrélation vous dit que deux choses évoluent ensemble. La régression vous dit de combien — et vous permet d'ajuster pour les facteurs de confusion. »

7.1 De la corrélation à la prédiction

Au chapitre 6, nous avons demandé « y a-t-il une différence ? ». Maintenant nous demandons « pouvons-nous prédire une variable à partir d'une autre ? » La régression est l'outil de base de la recherche clinique : elle apparaît dans presque chaque étude épidémiologique, chaque analyse d'essai clinique et chaque modèle de prédiction du risque.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

7.2 Régression linéaire simple

Question : Comment la pression artérielle systolique évolue-t-elle avec l'âge ?

```
# Donnees cliniques simulees
np.random.seed(42)
n = 200
age = np.random.uniform(25, 80, n)
# Relation reelle : PAS = 90 + 0.7 * age + bruit
sbp = 90 + 0.7 * age + np.random.normal(0, 12, n)

df = pd.DataFrame({"age": age, "systolic_bp": sbp})

# Ajuster avec scipy
slope, intercept, r_value, p_value, std_err = stats.linregress(df["age"],
↪  df["systolic_bp"])
```

```
print(f"Ordonnee a l'origine : {intercept:.2f}")
print(f"Pente : {slope:.2f} mmHg par annee d'age")
print(f"R-carre : {r_value**2:.3f}")
print(f"Valeur p : {p_value:.2e}")
```

```
# Visualiser
fig, ax = plt.subplots(figsize=(8, 5))
ax.scatter(df["age"], df["systolic_bp"], alpha=0.5, s=20, color="steelblue")
x_line = np.linspace(25, 80, 100)
ax.plot(x_line, intercept + slope * x_line, color="red", linewidth=2,
        label=f"PAS = {intercept:.1f} + {slope:.2f} x Age")
ax.set_xlabel("Age (annees)")
ax.set_ylabel("PA systolique (mmHg)")
ax.set_title("Regression lineaire simple : PAS vs Age")
ax.legend()
plt.tight_layout()
plt.show()
```

Contexte clinique

La pente de 0,7 signifie qu'en moyenne, la PA systolique augmente de 0,7 mmHg pour chaque année d'âge supplémentaire. Cela ne signifie *pas* que le vieillissement *cause* une PA plus élevée — cela signifie que les deux sont linéairement associés dans cet échantillon.

7.3 Régression linéaire multiple

Question : Prédire la PA systolique à partir de l'âge, de l'IMC et du statut tabagique simultanément.

```
import statsmodels.api as sm

# Donnees simulees etendues
np.random.seed(42)
n = 300
age = np.random.uniform(25, 80, n)
bmi = np.random.normal(27, 5, n)
smoking = np.random.binomial(1, 0.25, n)  # 25% fumeurs
# Vrai modele : PAS = 70 + 0.6*age + 0.8*IMC + 5*tabac + bruit
sbp = 70 + 0.6 * age + 0.8 * bmi + 5 * smoking + np.random.normal(0, 10, n)

df = pd.DataFrame({
    "age": age, "bmi": bmi, "smoking": smoking, "systolic_bp": sbp
})

# Ajuster la regression multiple avec statsmodels
X = sm.add_constant(df[["age", "bmi", "smoking"]])
model = sm.OLS(df["systolic_bp"], X).fit()
```

```
print(model.summary())
```

7.3.1 Interprétation des coefficients en contexte clinique

```
# Extraire et afficher les coefficients
coef_df = pd.DataFrame({
    "Coefficient": model.params,
    "IC 95% inf": model.conf_int()[0],
    "IC 95% sup": model.conf_int()[1],
    "Valeur p": model.pvalues
}).round(4)
print(coef_df)
```

Chaque coefficient répond à : « toutes les autres variables étant constantes, quel est le changement attendu de PAS pour une augmentation d'une unité de ce prédicteur ? »

- **Coefficient âge** $\approx 0{,}6$: chaque année d'âge supplémentaire est associée à 0,6 mmHg de PAS en plus, *après ajustement sur l'IMC et le tabagisme.*
- **Coefficient IMC** $\approx 0{,}8$: chaque unité d'IMC supplémentaire est associée à 0,8 mmHg de PAS en plus, *après ajustement sur l'âge et le tabagisme.*
- **Coefficient tabagisme** ≈ 5 : les fumeurs ont, en moyenne, 5 mmHg de PAS en plus que les non-fumeurs, *après ajustement sur l'âge et l'IMC.*

Astuce Python

L'expression « après ajustement sur » (ou « en contrôlant pour ») est la différence clé entre la régression simple et la régression multiple. C'est pourquoi les épidémiologistes utilisent la régression multiple — pour séparer l'effet d'une variable des facteurs de confusion potentiels.

7.4 Facteurs de confusion

Un facteur de confusion est une variable associée à la fois à l'exposition et au résultat, déformant la relation apparente entre eux.

```
# Exemple : consommation de cafe et maladie cardiaque
# Les deux sont associes au tabagisme (le facteur de confusion)
np.random.seed(42)
n = 500
smoking = np.random.binomial(1, 0.3, n)
coffee = 1.5 + 2 * smoking + np.random.normal(0, 1.5, n)  # les fumeurs boivent
↪  plus de cafe
heart_disease_risk = 0.5 * smoking + np.random.normal(0, 0.3, n)  # le tabac
↪  cause la MC
```

```
# Non ajuste : le cafe semble predire la maladie cardiaque
r_unadjusted, p_unadjusted = stats.pearsonr(coffee, heart_disease_risk)
print(f"Correlation non ajustee (cafe vs risque MC) : r = {r_unadjusted:.3f}, p
↪  = {p_unadjusted:.4f}")

# Regression ajustee : l'effet du cafe disparait
df_conf = pd.DataFrame({"coffee": coffee, "smoking": smoking, "hd_risk":
↪  heart_disease_risk})
X = sm.add_constant(df_conf[["coffee", "smoking"]])
model_adj = sm.OLS(df_conf["hd_risk"], X).fit()
print(f"\nModele ajuste :")
print(f"  Coefficient cafe :  {model_adj.params['coffee']:.4f} (p =
↪  {model_adj.pvalues['coffee']:.4f})")
print(f"  Coefficient tabac : {model_adj.params['smoking']:.4f} (p =
↪  {model_adj.pvalues['smoking']:.4f})")
```

Attention

Sans ajustement sur le tabagisme, vous concluriez à tort que le café augmente le risque de maladie cardiaque. C'est l'une des erreurs les plus courantes dans les études observationnelles. Réfléchissez toujours aux facteurs de confusion qui pourraient se cacher dans vos données.

7.5 Vérification des hypothèses de la régression

```
# Diagnostics des residus pour notre modele PAS
residuals = model.resid
fitted = model.fittedvalues

fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Residus vs valeurs ajustees
axes[0].scatter(fitted, residuals, alpha=0.4, s=15)
axes[0].axhline(0, color="red", linestyle="--")
axes[0].set_xlabel("Valeurs ajustees")
axes[0].set_ylabel("Residus")
axes[0].set_title("Residus vs valeurs ajustees")

# Histogramme des residus
axes[1].hist(residuals, bins=25, edgecolor="black", alpha=0.7)
axes[1].set_xlabel("Residus")
axes[1].set_title("Distribution des residus")

# Graphique Q-Q
stats.probplot(residuals, dist="norm", plot=axes[2])
axes[2].set_title("Graphique Q-Q")

plt.tight_layout()
plt.show()
```

Contexte clinique

Les quatre hypothèses clés de la régression linéaire : (1) linéarité, (2) indépendance des résidus, (3) homoscédasticité (variance constante), (4) normalité des résidus. Les graphiques ci-dessus vous aident à vérifier les hypothèses 1, 3 et 4. Une violation n'invalide pas toujours les résultats, mais doit vous rendre prudent.

7.6 Régression logistique

Quand le résultat est binaire (maladie : oui/non), la régression linéaire est inappropriée. La régression logistique modélise la *probabilité* du résultat.

Question : Prédire si un patient a le diabète (oui/non) à partir de l'âge, l'IMC et la glycémie.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Charger le jeu de donnees Pima Indians Diabetes
url = ("https://raw.githubusercontent.com/jbrownlee/Datasets/"
       "master/pima-indians-diabetes.data.csv")
columns = ["pregnancies", "glucose", "blood_pressure", "skin_thickness",
           "insulin", "bmi", "diabetes_pedigree", "age", "outcome"]
pima = pd.read_csv(url, names=columns)
print(f"Dimensions : {pima.shape}")
print(f"Prevalence du diabete : {pima['outcome'].mean():.1%}")
pima.head()
```

```
# Ajuster la regression logistique avec statsmodels (pour des resultats
↪  detailles)
X = sm.add_constant(pima[["age", "bmi", "glucose"]])
y = pima["outcome"]

logit_model = sm.Logit(y, X).fit()
print(logit_model.summary())
```

7.7 Rapports des cotes issus de la régression logistique

Les coefficients de la régression logistique sont des log-odds. Exponentionnez-les pour obtenir les **rapports des cotes** :

```
# Rapports des cotes avec intervalles de confiance a 95%
odds_ratios = np.exp(logit_model.params)
```

```
ci = np.exp(logit_model.conf_int())

or_df = pd.DataFrame({
    "Rapport des cotes": odds_ratios,
    "IC 95% inf": ci[0],
    "IC 95% sup": ci[1],
    "Valeur p": logit_model.pvalues
}).round(4)
print(or_df)
```

Contexte clinique

Un rapport des cotes de 1,04 pour la glycémie signifie : pour chaque augmentation de 1 mg/dL de glycémie, les cotes de diabète augmentent de 4 %, en maintenant l'âge et l'IMC constants. Un OR > 1 signifie un risque accru ; OR < 1 un risque réduit ; OR $= 1$ aucune association.

7.8 Évaluation du modèle : matrice de confusion et au-delà

```
# Ajuster avec scikit-learn pour la prediction
features = ["age", "bmi", "glucose"]
X = pima[features]
y = pima["outcome"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

clf = LogisticRegression(max_iter=1000, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
# Matrice de confusion
cm = confusion_matrix(y_test, y_pred)
print("Matrice de confusion :")
print(cm)
print()

# Metriques detaillees
print(classification_report(y_test, y_pred,
                            target_names=["Pas de diabete", "Diabete"]))
```

7.8.1 Sensibilité, spécificité, VPP, VPN

```
tn, fp, fn, tp = cm.ravel()
```

```
sensitivity = tp / (tp + fn)     # taux de vrais positifs (rappel)
specificity = tn / (tn + fp)     # taux de vrais negatifs
ppv = tp / (tp + fp)             # valeur predictive positive (precision)
npv = tn / (tn + fn)             # valeur predictive negative

print(f"Sensibilite (rappel) : {sensitivity:.3f}")
print(f"  -> Parmi ceux AVEC diabete, {sensitivity:.1%} ont ete correctement
↪  identifies")
print(f"Specificite :          {specificity:.3f}")
print(f"  -> Parmi ceux SANS diabete, {specificity:.1%} ont ete correctement
↪  identifies")
print(f"VPP (precision) :      {ppv:.3f}")
print(f"  -> Parmi ceux PREDITS positifs, {ppv:.1%} ont reellement le diabete")
print(f"VPN :                  {npv:.3f}")
print(f"  -> Parmi ceux PREDITS negatifs, {npv:.1%} sont reellement indemnes")
```

```
# Visualiser la matrice de confusion
fig, ax = plt.subplots(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Pas de diabete", "Diabete"],
            yticklabels=["Pas de diabete", "Diabete"], ax=ax)
ax.set_xlabel("Predit")
ax.set_ylabel("Reel")
ax.set_title("Matrice de confusion")
plt.tight_layout()
plt.show()
```

Attention

En dépistage clinique, manquer une maladie (faux négatif) est souvent plus dangereux qu'une fausse alerte (faux positif). Un test de dépistage du cancer doit avoir une **sensibilité élevée** même si la spécificité est modérée. Un test de confirmation doit avoir une **spécificité élevée**.

7.9 Introduction à l'analyse de survie

Parfois, le résultat n'est pas « oui/non » mais « temps jusqu'à l'événement » — temps jusqu'au décès, jusqu'à la rechute, jusqu'à la réadmission hospitalière. C'est l'**analyse de survie**.

7.9.1 Pourquoi ne pas simplement utiliser la régression logistique ?

Deux raisons :

1. **Censure.** Certains patients quittent l'étude avant que l'événement ne survienne. On sait qu'ils ont survécu *au moins* aussi longtemps, mais pas leur véritable temps d'événement.

2. **Le temps compte.** Un médicament qui retarde la rechute de 3 ans est meilleur qu'un médicament qui la retarde de 3 mois, même si les deux échouent finalement.

7.9.2 Courbes de Kaplan-Meier

```
# Installer lifelines si necessaire : pip install lifelines
from lifelines import KaplanMeierFitter, CoxPHFitter

# Donnees de survie simulees
np.random.seed(42)
n = 200
treatment = np.random.binomial(1, 0.5, n)
# Le groupe traitement survit plus longtemps
time = np.where(treatment == 1,
                np.random.exponential(24, n),    # traitement : moyenne 24 mois
                np.random.exponential(16, n))    # temoin : moyenne 16 mois
event = np.random.binomial(1, 0.75, n)  # 25% censures

surv_df = pd.DataFrame({
    "time": time, "event": event, "treatment": treatment
})
```

```
# Courbes de Kaplan-Meier
fig, ax = plt.subplots(figsize=(8, 5))

for label, group_df in surv_df.groupby("treatment"):
    kmf = KaplanMeierFitter()
    kmf.fit(group_df["time"], event_observed=group_df["event"],
            label="Traitement" if label == 1 else "Temoin")
    kmf.plot_survival_function(ax=ax)

ax.set_xlabel("Temps (mois)")
ax.set_ylabel("Probabilite de survie")
ax.set_title("Courbes de survie de Kaplan-Meier")
plt.tight_layout()
plt.show()
```

7.9.3 Modèle de Cox à risques proportionnels

Le modèle de Cox est la « régression » de l'analyse de survie. Il estime comment les covariables affectent le risque instantané (hazard) à chaque point temporel :

```
# Modele de Cox a risques proportionnels
cph = CoxPHFitter()
cph.fit(surv_df, duration_col="time", event_col="event",
        formula="treatment")
cph.print_summary()
```

```python
# Rapport de risque (hazard ratio)
hr = np.exp(cph.params_["treatment"])
print(f"Rapport de risque pour le traitement : {hr:.3f}")
if hr < 1:
    print(f"Le traitement reduit le risque de {(1-hr)*100:.1f}%")
else:
    print(f"Le traitement augmente le risque de {(hr-1)*100:.1f}%")
```

Contexte clinique

Un rapport de risque (*hazard ratio*) de 0,65 signifie que le groupe traitement a un risque 35 % plus faible de l'événement à tout moment donné, par rapport au groupe témoin. Les rapports de risque sont le mode standard de présentation des résultats dans les essais oncologiques, les études cardiovasculaires et toute étude avec un critère de jugement de type temps-jusqu'à-l'événement.

7.10 Travaux pratiques : prédiction du risque de diabète

Exercice

En utilisant le jeu de données Pima Indians Diabetes (https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv) :

1. **Préparation des données.** Chargez le jeu de données. Remplacez les valeurs zéro dans `glucose`, `blood_pressure`, `bmi`, `skin_thickness` et `insulin` par `NaN` (ce sont des zéros biologiquement impossibles). Supprimez les lignes avec des valeurs manquantes.

2. **Régression logistique simple.** Ajustez un modèle prédisant le diabète à partir de la glycémie seule. Quel est le rapport des cotes ? Interprétez-le.

3. **Régression logistique multiple.** Ajustez un modèle avec l'âge, l'IMC, la glycémie, la pression artérielle et la fonction de pédigree du diabète. Quelles variables sont statistiquement significatives ? Rapportez les rapports des cotes avec IC à 95 %.

4. **Comparaison de modèles.** Séparez les données 70/30. Ajustez les deux modèles sur l'ensemble d'entraînement. Comparez sensibilité, spécificité et exactitude sur l'ensemble de test. Quel modèle est meilleur pour le *dépistage* ?

5. **Confusion.** Ajustez un modèle avec la glycémie seule, puis ajoutez l'âge. Le coefficient de la glycémie change-t-il ? Qu'est-ce que cela vous dit sur la confusion ?

6. **Seuils cliniques.** Au lieu du seuil par défaut de 0,5, essayez 0,3 et 0,7. Comment la sensibilité et la spécificité changent-elles ? Quand préféreriez-vous un seuil plus bas ?

7. **Bonus : analyse de survie.** En utilisant les données de survie simulées ci-dessus, ajoutez « âge » et « tabagisme » comme covariables au modèle de Cox. Lequel a le plus grand rapport de risque ?

7.11 Résumé du chapitre

- La régression linéaire simple modélise la relation entre un prédicteur et un résultat continu.
- La régression multiple ajuste pour les facteurs de confusion — l'expression « après ajustement sur » reflète cela.
- Les coefficients représentent le changement attendu du résultat pour une variation d'une unité du prédicteur, les autres variables étant constantes.
- La régression logistique prédit les résultats binaires (maladie oui/non) et produit des rapports des cotes.
- L'évaluation du modèle nécessite la matrice de confusion : sensibilité, spécificité, VPP et VPN.
- Une sensibilité élevée est essentielle pour le dépistage ; une spécificité élevée pour la confirmation.
- L'analyse de survie (Kaplan-Meier, modèle de Cox) traite les données de temps-jusqu'à-l'événement avec censure.
- Vérifiez toujours les hypothèses de la régression et rapportez les intervalles de confiance, pas seulement les valeurs p.

Chapitre 8

Apprentissage automatique pour la prédiction clinique

« Un modèle qui fonctionne parfaitement sur vos données d'entraînement et échoue sur le patient suivant n'est pas un modèle — c'est un souvenir. »

8.1 Quand utiliser l'AA vs les statistiques traditionnelles

L'apprentissage automatique (AA) et les statistiques traditionnelles répondent à des questions différentes :

- **Statistiques traditionnelles** (régression, tests t) : « Quels facteurs de risque sont associés aux maladies cardiaques, et dans quelle mesure ? » L'objectif est l'*inférence* — comprendre les relations.
- **Apprentissage automatique** : « Etant donné les données de ce patient, quelle est sa probabilité de maladie cardiaque ? » L'objectif est la *prédiction* — la précision sur de nouveaux patients.

Contexte clinique

Utilisez la régression logistique quand vous devez expliquer le *pourquoi* (rapports des cotes, intervalles de confiance, valeurs p pour chaque facteur de risque). Utilisez l'AA quand vous avez besoin de la meilleure *prédiction* possible et que l'interprétabilité est secondaire. En pratique, de nombreuses études cliniques utilisent les deux : la régression pour comprendre la biologie, l'AA pour construire l'outil de dépistage.

8.1.1 Quand NE PAS utiliser l'AA

- Petits jeux de données (< 200 patients) : l'AA sur-ajuste facilement. Restez avec la régression logistique.
- Quand vous avez besoin d'une approbation réglementaire : la FDA exige des modèles interprétables pour la plupart des outils d'aide à la décision clinique.

- Quand une règle simple fonctionne : si « glycémie > 126 » classe correctement 90 % des diabétiques, vous n'avez pas besoin d'une forêt aléatoire.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report, confusion_matrix,
                             roc_curve, roc_auc_score, RocCurveDisplay)
```

8.2 Chargement des données cliniques

Nous utilisons le jeu de données UCI Heart Disease — un benchmark classique en AA clinique.

```
# Jeu de donnees UCI Heart Disease
url = ("https://raw.githubusercontent.com/raphaelfontenelle/"
       "heart-disease-uci/main/heart.csv")
heart = pd.read_csv(url)
print(f"Dimensions : {heart.shape}")
print(f"Prevalence de maladie cardiaque : {heart['target'].mean():.1%}")
heart.head()
```

```
# Descriptions des colonnes
column_info = {
    "age": "Age en annees",
    "sex": "1 = homme, 0 = femme",
    "cp": "Type de douleur thoracique (0-3)",
    "trestbps": "Pression arterielle au repos (mmHg)",
    "chol": "Cholesterol serique (mg/dL)",
    "fbs": "Glycemie a jeun > 120 mg/dL (1 = vrai)",
    "restecg": "Resultats ECG au repos (0-2)",
    "thalach": "Frequence cardiaque maximale atteinte",
    "exang": "Angor induit par l'exercice (1 = oui)",
    "oldpeak": "Depression ST induite par l'exercice",
    "slope": "Pente du segment ST au pic d'exercice",
    "ca": "Nombre de vaisseaux principaux colores par fluoroscopie (0-3)",
    "thal": "Thalassemie (1 = normal, 2 = defaut fixe, 3 = reversible)",
    "target": "Maladie cardiaque (1 = oui, 0 = non)"
}
for col, desc in column_info.items():
    print(f"  {col:12s} : {desc}")
```

8.3 Séparation entraînement/test et validation croisée

8.3.1 Pourquoi séparer les données ?

Si vous entraînez un modèle sur toutes vos données puis l'évaluez sur les mêmes données, l'exactitude est trompeusement élevée. Le modèle a « vu les réponses ». Pour estimer la performance en conditions réelles, vous devez évaluer sur des données que le modèle n'a *jamais vues*.

```
# Definir les variables et la cible
feature_cols = ["age", "sex", "cp", "trestbps", "chol", "fbs",
                "restecg", "thalach", "exang", "oldpeak", "slope", "ca",
                ↪  "thal"]
X = heart[feature_cols]
y = heart["target"]

# Separation 70/30 (stratifiee pour preserver l'equilibre des classes)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
print(f"Ensemble d'entrainement : {X_train.shape[0]} patients")
print(f"Ensemble de test :        {X_test.shape[0]} patients")
print(f"Prevalence entrainement : {y_train.mean():.1%}")
print(f"Prevalence test :         {y_test.mean():.1%}")
```

8.3.2 Validation croisée

Une seule séparation entraînement/test peut être chanceuse ou malchanceuse. La validation croisée (VC) fournit une estimation plus robuste en faisant tourner l'ensemble de test :

```
# Validation croisee a 5 plis avec regression logistique
lr = LogisticRegression(max_iter=1000, random_state=42)
cv_scores = cross_val_score(lr, X, y, cv=5, scoring="accuracy")
print(f"Exactitude VC : {cv_scores.mean():.3f} +/- {cv_scores.std():.3f}")
print(f"Par pli : {cv_scores.round(3)}")
```

Attention

N'utilisez jamais les résultats du test pour choisir votre modèle ou ajuster les hyperparamètres. L'ensemble de test ne doit être utilisé qu'**une seule fois**, à la toute fin. Si vous consultez les résultats du test pendant la sélection du modèle, vous entraînez effectivement sur l'ensemble de test.

8.4 Arbres de décision

Un arbre de décision sépare les patients en groupes à l'aide de questions oui/non, formant un organigramme que les cliniciens peuvent suivre.

```
# Ajuster un arbre de decision
dt = DecisionTreeClassifier(max_depth=4, random_state=42)
dt.fit(X_train, y_train)

# Evaluer
y_pred_dt = dt.predict(X_test)
print("Resultats de l'arbre de decision :")
print(classification_report(y_test, y_pred_dt,
                            target_names=["Pas de maladie", "Maladie"]))
```

```
# Visualiser l'arbre
fig, ax = plt.subplots(figsize=(20, 10))
plot_tree(dt, feature_names=feature_cols,
          class_names=["Pas de maladie", "Maladie"],
          filled=True, rounded=True, fontsize=9, ax=ax)
plt.title("Arbre de decision pour la prediction de maladie cardiaque")
plt.tight_layout()
plt.show()
```

Contexte clinique

Les arbres de décision sont populaires en médecine d'urgence car ils produisent des *règles de décision clinique* — des organigrammes simples qu'un médecin peut suivre au chevet du patient sans ordinateur. Le score HEART, les critères de Wells et les règles d'Ottawa pour la cheville sont tous conceptuellement similaires aux arbres de décision.

8.4.1 Le danger des arbres profonds

```
# Arbre sur-ajuste (pas de limite de profondeur)
dt_overfit = DecisionTreeClassifier(random_state=42)  # pas de max_depth
dt_overfit.fit(X_train, y_train)

print(f"Exactitude entrainement (sur-ajuste) : {dt_overfit.score(X_train,
↪  y_train):.3f}")
print(f"Exactitude test (sur-ajuste) :         {dt_overfit.score(X_test,
↪  y_test):.3f}")
print(f"Exactitude entrainement (profondeur=4) : {dt.score(X_train,
↪  y_train):.3f}")
print(f"Exactitude test (profondeur=4) :         {dt.score(X_test,
↪  y_test):.3f}")
```

Astuce Python

Quand l'exactitude d'entraînement est bien supérieure à celle du test, votre modèle **sur-ajuste** — il a mémorisé les données d'entraînement au lieu d'apprendre des motifs généraux. Limitez la profondeur de l'arbre avec `max_depth` ou exigez un nombre minimum d'échantillons par feuille avec `min_samples_leaf`.

8.5 Forêts aléatoires

Une forêt aléatoire combine des centaines d'arbres de décision, chacun entraîné sur un sous-ensemble aléatoire des données et des variables. La prédiction finale est le vote majoritaire.

```
# Ajuster une foret aleatoire
rf = RandomForestClassifier(n_estimators=200, max_depth=6,
                            random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
print("Resultats de la foret aleatoire :")
print(classification_report(y_test, y_pred_rf,
                            target_names=["Pas de maladie", "Maladie"]))
```

```
# Comparaison par validation croisee
models = {
    "Regression logistique": LogisticRegression(max_iter=1000,
    ↪  random_state=42),
    "Arbre de decision (profondeur=4)": DecisionTreeClassifier(max_depth=4,
    ↪  random_state=42),
    "Foret aleatoire": RandomForestClassifier(n_estimators=200, max_depth=6,
                                              random_state=42, n_jobs=-1)
}

print("Exactitude VC a 5 plis :")
for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5, scoring="accuracy")
    print(f"  {name:40s}: {scores.mean():.3f} +/- {scores.std():.3f}")
```

8.6 Courbes ROC et AUC

La courbe ROC (*Receiver Operating Characteristic*) trace la sensibilité en fonction de (1 – spécificité) pour tous les seuils de classification possibles. L'AUC (*Area Under the Curve*) résume la capacité discriminante en un seul nombre.

- AUC = 0,5 : pas mieux que le hasard.
- AUC = 0,7–0,8 : discrimination acceptable.

- AUC = 0,8–0,9 : excellente discrimination.
- AUC > 0,9 : exceptionnelle (rare en pratique clinique).

```
# Obtenir les probabilites predites
lr.fit(X_train, y_train)
y_prob_lr = lr.predict_proba(X_test)[:, 1]
y_prob_rf = rf.predict_proba(X_test)[:, 1]

# Calculer les courbes ROC
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)
auc_lr = roc_auc_score(y_test, y_prob_lr)
auc_rf = roc_auc_score(y_test, y_prob_rf)

# Tracer
fig, ax = plt.subplots(figsize=(7, 6))
ax.plot(fpr_lr, tpr_lr, label=f"Regression logistique (AUC = {auc_lr:.3f})",
        linewidth=2)
ax.plot(fpr_rf, tpr_rf, label=f"Foret aleatoire (AUC = {auc_rf:.3f})",
        linewidth=2)
ax.plot([0, 1], [0, 1], "k--", alpha=0.5, label="Hasard (AUC = 0.500)")
ax.set_xlabel("Taux de faux positifs (1 - Specificite)")
ax.set_ylabel("Taux de vrais positifs (Sensibilite)")
ax.set_title("Courbes ROC : prediction de maladie cardiaque")
ax.legend(loc="lower right")
plt.tight_layout()
plt.show()
```

Contexte clinique

En pratique clinique, les courbes ROC servent à comparer des tests diagnostiques. Lors de l'évaluation d'un nouveau biomarqueur pour le sepsis, vous tracez sa courbe ROC par rapport aux biomarqueurs existants (CRP, procalcitonine). Le test avec l'AUC la plus élevée a la meilleure capacité discriminante globale.

8.7 Importance des variables

Question : Quels facteurs de risque comptent le plus pour prédire la maladie cardiaque ?

```
# Importance des variables de la foret aleatoire
importances = rf.feature_importances_
importance_df = pd.DataFrame({
    "Feature": feature_cols,
    "Importance": importances
}).sort_values("Importance", ascending=True)
```

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.barh(importance_df["Feature"], importance_df["Importance"],
↪  color="steelblue")
ax.set_xlabel("Importance des variables (Gini)")
ax.set_title("Quels facteurs de risque predisent la maladie cardiaque ?")
plt.tight_layout()
plt.show()
```

```
# Top 5 des variables
print("Top 5 des variables les plus importantes :")
for _, row in importance_df.tail(5).iloc[::-1].iterrows():
    print(f"  {row['Feature']:12s}: {row['Importance']:.3f}")
```

Attention

L'importance des variables dans les forêts aléatoires indique quelles variables sont utiles pour la *prédiction*, pas quelles variables *causent* le résultat. Une variable peut être importante pour la prédiction parce qu'elle est un proxy d'autre chose. Ne confondez pas importance prédictive et importance causale.

8.8 Calibration

Un modèle dit « ce patient a 70 % de risque de maladie cardiaque ». Cela signifie-t-il que parmi tous les patients auxquels le modèle attribue 70 %, environ 70 % ont effectivement la maladie ? Si oui, le modèle est **bien calibré**.

```
from sklearn.calibration import calibration_curve

# Graphique de calibration pour la foret aleatoire
prob_true, prob_pred = calibration_curve(y_test, y_prob_rf, n_bins=8)

fig, ax = plt.subplots(figsize=(6, 6))
ax.plot(prob_pred, prob_true, "o-", linewidth=2, label="Foret aleatoire")
ax.plot([0, 1], [0, 1], "k--", label="Calibration parfaite")
ax.set_xlabel("Probabilite predite moyenne")
ax.set_ylabel("Proportion observee")
ax.set_title("Graphique de calibration")
ax.legend()
plt.tight_layout()
plt.show()
```

```
# Score de Brier : plus bas = meilleur (0 = parfait)
from sklearn.metrics import brier_score_loss

brier_lr = brier_score_loss(y_test, y_prob_lr)
brier_rf = brier_score_loss(y_test, y_prob_rf)
```

```
print(f"Score de Brier (Regression logistique) : {brier_lr:.4f}")
print(f"Score de Brier (Foret aleatoire) :       {brier_rf:.4f}")
```

Contexte clinique

La calibration est essentielle dans les modèles de prédiction clinique. Si un calculateur de risque de diabète dit à un patient « vous avez 40 % de chances de développer un diabète dans 10 ans », ce chiffre doit être exact — il guide les décisions thérapeutiques. Le Framingham Risk Score et le QRISK sont régulièrement recalibrés pour différentes populations.

8.9 Sur-ajustement : le danger des petits jeux de données cliniques

```
# Demontrer le sur-ajustement avec des tailles de jeux de donnees variables
train_sizes = [30, 50, 100, 150, 200, len(X_train)]
results = []

for size in train_sizes:
    if size > len(X_train):
        continue
    X_sub = X_train.iloc[:size]
    y_sub = y_train.iloc[:size]

    rf_temp = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_temp.fit(X_sub, y_sub)

    train_acc = rf_temp.score(X_sub, y_sub)
    test_acc = rf_temp.score(X_test, y_test)
    results.append({"n_train": size, "train_acc": train_acc, "test_acc":
    ↪  test_acc})

results_df = pd.DataFrame(results)

fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(results_df["n_train"], results_df["train_acc"], "o-",
        label="Exactitude entrainement", linewidth=2)
ax.plot(results_df["n_train"], results_df["test_acc"], "s-",
        label="Exactitude test", linewidth=2)
ax.set_xlabel("Nombre de patients d'entrainement")
ax.set_ylabel("Exactitude")
ax.set_title("Courbe d'apprentissage : plus de donnees reduit le
↪  sur-ajustement")
ax.legend()
ax.set_ylim(0.5, 1.05)
plt.tight_layout()
plt.show()
```

Astuce Python

Signes de sur-ajustement : (1) l'exactitude d'entraînement est bien supérieure à celle du test, (2) la performance varie fortement entre les plis de validation croisée, (3) ajouter des variables dégrade la performance sur le test. Le remède : plus de données, modèles plus simples, régularisation ou validation croisée pour la sélection du modèle.

8.10 Travaux pratiques : pipeline de prédiction de maladie cardiaque

```
# Pipeline complet des donnees brutes au modele evalue
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Pipeline complet
pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
    ("classifier", RandomForestClassifier(n_estimators=200, max_depth=6,
                                          random_state=42))
])

# Validation croisee du pipeline complet
cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring="roc_auc")
print(f"AUC VC du pipeline : {cv_scores.mean():.3f} +/- {cv_scores.std():.3f}")

# Ajustement final et evaluation
pipeline.fit(X_train, y_train)
y_prob_final = pipeline.predict_proba(X_test)[:, 1]
y_pred_final = pipeline.predict(X_test)

print(f"\nAUC test finale : {roc_auc_score(y_test, y_prob_final):.3f}")
print(f"\n{classification_report(y_test, y_pred_final, target_names=['Pas de
↪  maladie', 'Maladie'])}")
```

8.11 Éthique : équité dans les modèles de prédiction clinique

Attention

Les modèles de prédiction clinique peuvent perpétuer et amplifier les inégalités de santé. Un modèle entraîné principalement sur une population peut mal fonctionner sur une autre. Avant de déployer tout modèle d'AA clinique, demandez-vous :

- **Qui figure dans les données d'entraînement ?** Si 90 % des patients sont

des hommes blancs, le modèle peut ne pas fonctionner pour les femmes ou les groupes minoritaires.

- **La performance diffère-t-elle entre sous-groupes ?** Vérifiez l'AUC, la sensibilité et la spécificité séparément par sexe, origine ethnique et groupe d'âge.
- **Quelles sont les conséquences des erreurs ?** Un faux négatif pour le risque cardiaque chez une jeune femme est plus dangereux qu'un faux positif.
- **Le modèle est-il transparent ?** Les cliniciens et les patients méritent de savoir comment une prédiction a été faite.

```python
# Verifier la performance du modele par sexe
for sex_val, sex_label in [(1, "Homme"), (0, "Femme")]:
    mask = X_test["sex"] == sex_val
    if mask.sum() < 10:
        print(f"{sex_label}: trop peu d'echantillons ({mask.sum()})")
        continue

    auc_sub = roc_auc_score(y_test[mask], y_prob_final[mask])
    sens = (y_pred_final[mask] == 1)[y_test[mask] == 1].mean()
    spec = (y_pred_final[mask] == 0)[y_test[mask] == 0].mean()

    print(f"{sex_label:8s}: AUC = {auc_sub:.3f}, "
          f"Sensibilite = {sens:.3f}, Specificite = {spec:.3f}, "
          f"n = {mask.sum()}")
```

Contexte clinique

En 2019, un algorithme commercial largement utilisé pour l'allocation des ressources de santé s'est révélé biaisé contre les patients noirs. L'algorithme utilisait les *coûts* de santé comme proxy des *besoins* de santé, mais comme les patients noirs avaient des coûts plus faibles en raison de barrières systémiques à l'accès, l'algorithme sous-estimait systématiquement leurs besoins. Cela a affecté des millions de patients. La leçon : auditez toujours votre modèle pour les disparités.

Exercice

En utilisant le jeu de données UCI Heart Disease :

1. **Base de référence.** Ajustez un modèle de régression logistique avec toutes les variables. Rapportez l'exactitude VC et l'AUC.
2. **Arbre de décision.** Ajustez un arbre de décision avec `max_depth=3`. Visualisez l'arbre. Un clinicien peut-il suivre cette règle de décision au chevet du patient ?
3. **Forêt aléatoire.** Ajustez une forêt aléatoire avec 200 arbres. Comparez son AUC à la régression logistique et à l'arbre de décision. L'amélioration vaut-elle

la perte d'interprétabilité ?

4. **Sélection de variables.** Entraînez une forêt aléatoire en utilisant uniquement les 5 variables les plus importantes. Comment la performance se compare-t-elle à l'utilisation des 13 variables ?

5. **Comparaison ROC.** Tracez les courbes ROC des trois modèles sur la même figure. Quel modèle choisiriez-vous pour un outil de dépistage ? Pour un test de confirmation ?

6. **Analyse des seuils.** Pour la forêt aléatoire, tracez la sensibilité et la spécificité en fonction du seuil de classification (de 0 à 1). À quel seuil sont-elles égales ?

7. **Calibration.** Tracez les courbes de calibration pour la régression logistique et la forêt aléatoire. Quel modèle est mieux calibré ?

8. **Audit d'équité.** Comparez l'AUC, la sensibilité et la spécificité entre patients masculins et féminins. Y a-t-il des disparités préoccupantes ? Qu'est-ce qui pourrait les expliquer ?

9. **Déploiement clinique.** Rédigez un paragraphe (dans une cellule Markdown) décrivant comment vous valideriez ce modèle avant de l'utiliser dans un véritable hôpital. Considérez : la validation externe, les exigences réglementaires, le flux de travail clinique et le consentement du patient.

8.12 Résumé du chapitre

- Utilisez les statistiques traditionnelles (régression) pour l'inférence et l'AA pour la prédiction.
- Séparez toujours les données en ensembles d'entraînement et de test. N'évaluez jamais sur les données d'entraînement.
- La validation croisée donne des estimations de performance plus robustes qu'une seule séparation.
- Les arbres de décision sont interprétables et adaptés aux cliniciens mais sujets au sur-ajustement.
- Les forêts aléatoires améliorent l'exactitude en agrégeant de nombreux arbres.
- Les courbes ROC et l'AUC comparent la capacité discriminante de différents modèles.
- L'importance des variables révèle quels facteurs de risque guident les prédictions (mais pas la causalité).
- La calibration vérifie si les probabilités prédites correspondent aux résultats observés.

- Le sur-ajustement est le danger central de l'AA sur les petits jeux de données cliniques — validez toujours.
- Les audits d'équité sont essentiels : vérifiez la performance du modèle par sous-groupes démographiques avant le déploiement.

Chapitre 9

Données géospatiales et temporelles de santé

« Une épidémie est une histoire racontée dans le temps et dans l'espace. Si vous pouvez tracer les deux, vous pouvez voir d'où elle vient et où elle va. »

9.1 Séries temporelles en santé

Une *série temporelle* est une suite de mesures enregistrées à intervalles réguliers : comptage quotidien des cas, bilan hebdomadaire des décès, doses mensuelles de vaccination. Les séries temporelles de santé ont trois composantes :

- **Tendance.** La direction à long terme — l'incidence augmente-t-elle ou diminue-t-elle ?
- **Saisonnalité.** Des cycles réguliers — le paludisme culmine pendant les saisons des pluies, la grippe culmine en hiver.
- **Bruit.** La fluctuation aléatoire au jour le jour qui masque le signal.

Contexte clinique

Les courbes épidémiques (« epi curves ») sont les séries temporelles les plus importantes en santé publique. Lors d'une épidémie, la forme de la courbe épidémique vous indique le mode de transmission : une exposition ponctuelle produit un pic aigu ; une source continue produit un plateau ; une transmission de personne à personne génère des vagues successives.

9.2 Tracer des séries temporelles avec pandas

9.2.1 Analyse et indexation des dates

Pandas gère les dates nativement. L'essentiel est d'analyser les colonnes de dates et de les définir comme index :

```
import pandas as pd
import matplotlib.pyplot as plt

# Exemple : cas quotidiens de paludisme dans un hopital de district
dates = pd.date_range("2023-01-01", periods=365, freq="D")
import numpy as np
np.random.seed(42)
# Simuler un schema saisonnier : pic pendant la saison des pluies
↪  (juin-septembre)
day_of_year = dates.dayofyear
seasonal = 50 + 40 * np.sin(2 * np.pi * (day_of_year - 90) / 365)
cases = np.maximum(0, seasonal + np.random.normal(0, 12, 365)).astype(int)

df = pd.DataFrame({"date": dates, "cases": cases})
df["date"] = pd.to_datetime(df["date"])
df = df.set_index("date")
df.head()
```

9.2.2 Moyennes glissantes

Les fluctuations quotidiennes rendent les tendances difficiles à voir. Une *moyenne glissante* lisse la courbe :

```
df["cases_7d"] = df["cases"].rolling(window=7).mean()
df["cases_14d"] = df["cases"].rolling(window=14).mean()

fig, ax = plt.subplots(figsize=(12, 5))
ax.bar(df.index, df["cases"], alpha=0.3, label="Cas quotidiens",
↪  color="steelblue")
ax.plot(df.index, df["cases_7d"], color="red", linewidth=2, label="Moyenne 7
↪  jours")
ax.plot(df.index, df["cases_14d"], color="darkgreen", linewidth=2,
↪  label="Moyenne 14 jours")
ax.set_xlabel("Date")
ax.set_ylabel("Cas de paludisme")
ax.set_title("Cas quotidiens de paludisme avec moyennes glissantes")
ax.legend()
plt.tight_layout()
plt.show()
```

Astuce Python

Le paramètre `window` définit le nombre de jours sur lequel moyenner. Une fenêtre de 7 jours élimine les effets jour de la semaine ; une fenêtre de 14 jours lisse de manière plus agressive. Choisissez en fonction du niveau de bruit de vos données.

9.2.3 Décomposition de tendance

La bibliothèque `statsmodels` peut décomposer une série temporelle en composantes de tendance, saisonnalité et résidus :

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df["cases"], model="additive", period=30)

fig, axes = plt.subplots(4, 1, figsize=(12, 10), sharex=True)
result.observed.plot(ax=axes[0], title="Observe")
result.trend.plot(ax=axes[1], title="Tendance")
result.seasonal.plot(ax=axes[2], title="Saisonnalite")
result.resid.plot(ax=axes[3], title="Residus")
plt.tight_layout()
plt.show()
```

> **Attention**
>
> Le paramètre `period` doit correspondre à la longueur de cycle attendue. Pour des données quotidiennes avec une saisonnalité mensuelle, utilisez `period=30`. Pour des données hebdomadaires avec une saisonnalité annuelle, utilisez `period=52`. Une mauvaise période produira des décompositions trompeuses.

9.3 Courbes de cas COVID-19

9.3.1 Chargement des données JHU

Le jeu de données COVID-19 de l'université Johns Hopkins (CSSE) est l'un des jeux de données de santé publique les plus cités de l'histoire :

```
# JHU CSSE COVID-19 cas confirmes (mondial, serie temporelle)
url = ("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/"
       "master/csse_covid_19_data/csse_covid_19_time_series/"
       "time_series_covid19_confirmed_global.csv")
confirmed = pd.read_csv(url)
print(f"Dimensions : {confirmed.shape}")
confirmed.head()
```

9.3.2 Reformater du format large au format long

Les données JHU sont en format large (une colonne par date). Nous devons les reformater :

```
# Se concentrer sur 5 pays africains
countries = ["South Africa", "Nigeria", "Kenya", "Egypt", "Morocco"]
africa = confirmed[confirmed["Country/Region"].isin(countries)]
```

```python
# Grouper par pays (certains pays ont des lignes par province)
africa = africa.groupby("Country/Region").sum(numeric_only=True)
africa = africa.drop(columns=["Lat", "Long"], errors="ignore")

# Reformater : large -> long
africa_long = africa.T
africa_long.index = pd.to_datetime(africa_long.index)
africa_long.index.name = "date"
africa_long.head()
```

9.3.3 Calcul des cas quotidiens et moyennes sur 7 jours

```python
# Cumulatif -> nouveaux cas quotidiens
daily = africa_long.diff().clip(lower=0)  # clip supprime les corrections
↪  negatives

# Moyenne glissante sur 7 jours
daily_7d = daily.rolling(7).mean()

# Tracer
fig, ax = plt.subplots(figsize=(14, 6))
for country in countries:
    ax.plot(daily_7d.index, daily_7d[country], linewidth=2, label=country)
ax.set_xlabel("Date")
ax.set_ylabel("Nouveaux cas quotidiens (moyenne 7 jours)")
ax.set_title("Cas quotidiens de COVID-19 dans 5 pays africains")
ax.legend()
plt.tight_layout()
plt.show()
```

Contexte clinique

La méthode `.diff()` convertit les comptages cumulatifs en incréments quotidiens. Des valeurs négatives apparaissent parfois en raison de corrections de données (par ex. un pays révise son total à la baisse). Borner à zéro est l'approche standard, bien que cela gonfle légèrement les jours suivants.

9.4 Introduction aux données géospatiales

Les données géospatiales lient des mesures à des localisations sur Terre. Deux formats courants :

- **Shapefiles** (`.shp`) — le format SIG traditionnel. Un shapefile est en réalité un ensemble de fichiers (`.shp`, `.shx`, `.dbf`, `.prj`) qui définissent ensemble des géométries de polygones et des attributs.
- **GeoJSON** (`.geojson`) — un seul fichier JSON. Plus facile à partager, lire et utiliser sur le web.

Les deux formats stockent des *géométries* (frontières de pays, limites de districts, emplacements d'hôpitaux) et des *attributs* (population, nombre de cas, taux de mortalité) dans la même structure.

9.5 geopandas pour la cartographie des maladies

`geopandas` étend pandas avec des capacités spatiales. Un `GeoDataFrame` est un DataFrame avec une colonne spéciale `geometry` qui contient les formes.

9.5.1 Installation

```
# Dans Google Colab ou en local
!pip install geopandas mapclassify
```

9.5.2 Charger une carte du monde

```
import geopandas as gpd

# Jeu de donnees Natural Earth (integre dans geopandas)
world = gpd.read_file(gpd.datasets.get_path("naturalearth_lowres"))
print(f"Colonnes : {world.columns.tolist()}")
print(f"SCR : {world.crs}")  # Systeme de coordonnees de reference
world.head()
```

9.5.3 Cartes choroplèthes

Une carte *choroplèthe* colorie les régions selon une variable. C'est l'outil standard pour cartographier la charge de morbidité par pays ou district :

```
# Filtrer l'Afrique
africa_map = world[world["continent"] == "Africa"].copy()

# Tracer l'estimation de population
fig, ax = plt.subplots(figsize=(10, 10))
africa_map.plot(column="pop_est", cmap="YlOrRd", legend=True,
                legend_kwds={"label": "Population", "shrink": 0.6},
                edgecolor="black", linewidth=0.5, ax=ax)
ax.set_title("Estimations de population des pays africains")
ax.set_axis_off()
plt.tight_layout()
plt.show()
```

Astuce Python

Bonnes palettes de couleurs pour les données de santé : `"YlOrRd"` (séquentielle, de faible à forte sévérité), `"RdYlGn_r"` (divergente, rouge = mauvais, vert = bon), `"Blues"` (séquentielle, neutre). Évitez les palettes arc-en-ciel (`"jet"`) — elles trompent l'œil.

9.6 Cartographie de la prévalence du paludisme par pays africain

Combinons des données de santé réelles avec la cartographie géospatiale. Nous utiliserons les données d'incidence du paludisme du GHO de l'OMS :

```
# GHO de l'OMS : incidence estimee du paludisme (pour 1000 personnes a risque)
url = "https://apps.who.int/gho/athena/api/GHO/MALARIA_INCIDENCE?format=csv"
malaria = pd.read_csv(url)
print(malaria.columns.tolist())
malaria.head()
```

```
# Filtrer l'annee la plus recente et les donnees au niveau pays
latest_year = malaria["YEAR (DISPLAY)"].max()
malaria_latest = malaria[malaria["YEAR (DISPLAY)"] == latest_year].copy()

# Garder les colonnes pertinentes
malaria_latest = malaria_latest[["COUNTRY (DISPLAY)", "Numeric"]].copy()
malaria_latest.columns = ["country", "incidence_per_1000"]
malaria_latest.head()
```

```
# Fusionner avec la carte de l'Afrique
# Faire correspondre les noms de pays (certains necessitent une correction
↪  manuelle)
name_map = {
    "United Republic of Tanzania": "Tanzania",
    "Democratic Republic of the Congo": "Dem. Rep. Congo",
    "Central African Republic": "Central African Rep.",
    "South Sudan": "S. Sudan",
    "Cote d'Ivoire": "Cote d'Ivoire",
    "Equatorial Guinea": "Eq. Guinea",
}
malaria_latest["country"] = malaria_latest["country"].replace(name_map)

africa_malaria = africa_map.merge(malaria_latest, left_on="name",
                                  right_on="country", how="left")
```

```
fig, ax = plt.subplots(figsize=(10, 10))
```

```
africa_malaria.plot(column="incidence_per_1000", cmap="YlOrRd",
                    legend=True, missing_kwds={"color": "lightgrey"},
                    legend_kwds={"label": "Incidence pour 1 000",
                                 "shrink": 0.6},
                    edgecolor="black", linewidth=0.5, ax=ax)
ax.set_title(f"Incidence du paludisme en Afrique ({latest_year})")
ax.set_axis_off()
plt.tight_layout()
plt.show()
```

Contexte clinique

La charge du paludisme en Afrique subsaharienne représente environ 95 % des décès mondiaux dus au paludisme. Les cartes choroplèthes révèlent immédiatement la concentration géographique et aident à cibler les programmes d'intervention tels que les moustiquaires imprégnées d'insecticide (MII) et la pulvérisation intradomiciliaire à effet rémanent (PIR).

9.7 Combiner analyse temporelle et spatiale

9.7.1 Petits multiples : une carte par période

Une alternative pratique à l'animation est l'approche des *petits multiples* — une carte par période côte à côte :

```
# Exemple : cas cumules de COVID-19 par pays a 3 moments
dates_to_plot = ["2020-06-01", "2021-01-01", "2021-06-01"]

fig, axes = plt.subplots(1, 3, figsize=(18, 7))
for ax, date_str in zip(axes, dates_to_plot):
    date_col = pd.to_datetime(date_str).strftime("%-m/%-d/%y")
    if date_col in africa.columns:
        data_col = africa[date_col]
    else:
        # Trouver la date la plus proche
        all_dates = pd.to_datetime(africa.columns, errors="coerce")
        nearest = all_dates[all_dates <= pd.to_datetime(date_str)][-1]
        data_col = africa[nearest.strftime("%-m/%-d/%y")]

    temp = africa_map.copy()
    temp = temp.merge(data_col.reset_index(), left_on="name",
                      right_on="Country/Region", how="left")
    temp.plot(column=data_col.name, cmap="Reds", legend=True,
              edgecolor="black", linewidth=0.5, ax=ax,
              missing_kwds={"color": "lightgrey"})
    ax.set_title(date_str)
    ax.set_axis_off()

plt.suptitle("Cas cumules de COVID-19 en Afrique", fontsize=14)
```

```
plt.tight_layout()
plt.show()
```

9.7.2 Cartes animées (option avancée)

Pour les présentations, des GIFs animés peuvent montrer la propagation épidémique au fil du temps. L'idée : créer une image par pas de temps, puis combiner avec `imageio` ou `matplotlib.animation` :

```
import matplotlib.animation as animation

fig, ax = plt.subplots(figsize=(10, 10))

def update(frame_date):
    ax.clear()
    # Fusionner les donnees de cas pour cette date avec la carte
    temp = africa_map.copy()
    # ... logique de fusion similaire a ci-dessus ...
    temp.plot(column="cases", cmap="Reds", ax=ax, edgecolor="black",
              linewidth=0.5, missing_kwds={"color": "lightgrey"},
              vmin=0, vmax=max_cases)
    ax.set_title(f"Cas COVID-19 - {frame_date}")
    ax.set_axis_off()

# Creer l'animation (une image par mois)
# ani = animation.FuncAnimation(fig, update, frames=monthly_dates,
#                               interval=500)
# ani.save("covid_africa.gif", writer="pillow")
```

Attention

Les cartes animées sont visuellement impressionnantes mais peuvent être difficiles à interpréter scientifiquement. Pour les publications, les petits multiples ou les instantanés statiques avec un curseur temporel (dans un tableau de bord) sont préférés. Utilisez les animations pour les présentations et la communication avec le public.

9.8 Travaux pratiques : tableau de bord COVID-19 pour 5 pays africains

Exercice

Construisez un tableau de bord COVID-19 multi-panneaux pour l'Afrique du Sud, le Nigéria, le Kenya, l'Égypte et le Maroc. Votre tableau de bord doit comporter quatre panneaux :

Panneau 1 — Cas confirmés au fil du temps.

1. Chargez les données de séries temporelles JHU CSSE pour les cas confirmés : https://raw.githubusercontent.com/CSSEGISandData/COVID-19/

`master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv`

2. Filtrez les 5 pays. Calculez les nouveaux cas quotidiens et les moyennes glissantes sur 7 jours.

3. Tracez les 5 pays sur un même axe avec une légende.

Panneau 2 — Décès au fil du temps.

1. Chargez la série temporelle des décès JHU : https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_deaths_global.csv

2. Calculez les décès quotidiens (moyenne 7 jours) et tracez.

Panneau 3 — Progression de la vaccination.

1. Chargez le jeu de données de vaccination de Our World in Data : https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/vaccinations/vaccinations.csv

2. Filtrez les 5 pays. Tracez `people_fully_vaccinated_per_hundred` au fil du temps.

Panneau 4 — Carte choroplèthe.

1. Avec `geopandas`, créez une carte choroplèthe du total des cas confirmés (ou des cas par million) à travers tous les pays africains à la dernière date disponible.

2. Mettez en évidence les 5 pays cibles avec des bordures plus épaisses.

Mise en page :

```
fig, axes = plt.subplots(2, 2, figsize=(18, 14))
# axes[0, 0] -> Panneau 1 : Cas
# axes[0, 1] -> Panneau 2 : Deces
# axes[1, 0] -> Panneau 3 : Vaccination
# axes[1, 1] -> Panneau 4 : Carte choroletheh
```

Bonus :

- Ajoutez un tableau sous la figure avec les statistiques récapitulatives par pays : total des cas, total des décès, taux de létalité (%), couverture vaccinale (%).

- Normalisez les cas et décès pour 100 000 habitants pour permettre une comparaison équitable.

9.9 Résumé du chapitre

- Les séries temporelles de santé ont trois composantes : tendance, saisonnalité et bruit.
- Les moyennes glissantes (`.rolling()`) lissent les fluctuations quotidiennes ; `seasonal_decompose(` sépare formellement les composantes.
- Le jeu de données JHU CSSE fournit des séries temporelles mondiales de cas, décès et guérisons COVID-19. Utilisez `.diff()` pour convertir les comptages cumulatifs en nouveaux cas quotidiens.
- Les données géospatiales se présentent sous forme de shapefiles (`.shp`) ou de GeoJSON (`.geojson`). `geopandas` lit les deux.
- Les cartes choroplèthes colorient les régions selon une variable de santé — utilisez `.plot(column=...)` avec une palette de couleurs appropriée.
- La fusion de données de santé avec des données cartographiques nécessite de faire correspondre les noms de pays — vérifiez toujours les incohérences.
- Les petits multiples (une carte par période) sont plus rigoureux scientifiquement que les animations.

Chapitre 10

Projet de synthèse

« La meilleure façon d'apprendre l'analyse de données, c'est d'analyser des données. Pas des données jouets — des données réelles, avec un vrai désordre, sur de vraies questions de santé. »

10.1 Consignes du projet

Le projet de synthèse est votre opportunité d'appliquer toutes les compétences de ce cours à une vraie question de santé. Vous travaillerez individuellement ou en binômes.

10.1.1 Calendrier

- **Semaine 1.** Choisir un projet, télécharger les données, effectuer l'exploration initiale.
- **Semaine 2.** Compléter l'analyse, générer les figures et tableaux, rédiger le rapport.
- **Semaine 3.** Finaliser le rapport et préparer une présentation orale de 5 minutes.

10.1.2 Attendus

Votre projet doit inclure :

1. **Une question de santé claire.** Pas « explorer les données » mais « la prévalence du diabète est-elle associée à l'IMC après ajustement sur l'âge et le sexe ? »
2. **Des données publiques réelles.** Pas de jeux de données simulés. Toutes les sources de données doivent être citées.
3. **Du code reproductible.** Un notebook Jupyter qui s'exécute du début à la fin sans erreur.
4. **Au moins 3 visualisations.** Pertinentes, étiquetées et interprétables.
5. **Au moins 1 test statistique ou modèle.** Avec une interprétation adéquate.
6. **Un rapport écrit.** 5 à 8 pages suivant le modèle ci-dessous.
7. **Une présentation orale.** 5 minutes avec diapositives.

Attention

Intégrité académique : vous pouvez utiliser des outils d'IA (ChatGPT, GitHub Copilot) pour aider à écrire du code, mais vous devez comprendre chaque ligne. Lors de la présentation orale, on vous demandera d'expliquer votre code et d'interpréter vos résultats. Copier-coller du code que vous ne comprenez pas sera évident.

10.2 Projets suggérés

Choisissez l'un des cinq projets ci-dessous, ou proposez le vôtre (sous réserve de l'approbation de l'enseignant). Chaque projet spécifie l'objectif, le(s) jeu(x) de données, les analyses requises et les livrables attendus.

10.2.1 Projet 1 : Analyse des facteurs de risque du diabète

Objectif. Identifier les facteurs de risque les plus importants du diabète de type 2 et construire un modèle prédictif.

Jeux de données.

- **Pima Indians Diabetes Dataset** (UCI/Kaggle) :
  `https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv`
  768 patientes, 8 variables (grossesses, glycémie, pression artérielle, épaisseur cutanée, insuline, IMC, fonction de pédigree du diabète, âge), issue binaire.

- **CDC BRFSS** (Behavioral Risk Factor Surveillance System) :
  `https://www.cdc.gov/brfss/annual_data/annual_data.htm`
  Enquête annuelle de plus de 400 000 adultes américains. Téléchargez le CSV de l'année la plus récente.

Analyses requises.

1. Charger et nettoyer le jeu de données Pima. Gérer les zéros dans glycémie, pression artérielle, IMC (ce sont des valeurs manquantes codées en 0).

2. Analyse exploratoire : distributions, corrélations, boîtes à moustaches par statut diabétique.

3. Régression logistique : quelles variables sont des prédicteurs significatifs ? Rapporter les rapports des cotes avec IC à 95 %.

4. Comparer la régression logistique avec un classifieur par forêt aléatoire. Rapporter exactitude, sensibilité, spécificité et AUC-ROC.

5. (Bonus) Répéter l'analyse des facteurs de risque sur les données BRFSS et comparer les résultats.

Livrables.

- Carte thermique de corrélation de toutes les variables.

- Courbes ROC comparant régression logistique et forêt aléatoire.
- Tableau des rapports des cotes avec intervalles de confiance.
- Interprétation écrite : quels facteurs de risque sont les plus importants cliniquement ?

Contexte clinique

Le jeu de données Pima Indians est historiquement important mais soulève des considérations éthiques. Les données ont été collectées auprès de la communauté Akimel O'odham (Pima), qui présente l'un des taux les plus élevés de diabète de type 2 au monde. Lors de la présentation, reconnaissez la population et évitez de généraliser les résultats à d'autres groupes.

10.2.2 Projet 2 : Analyse de l'impact de la COVID-19 dans les pays africains

Objectif. Analyser comment la COVID-19 a affecté différemment les pays africains et explorer les corrélations avec la capacité du système de santé et les facteurs socio-économiques.

Jeux de données.

- **Données JHU CSSE COVID-19** :
  https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_time_series
  Cas confirmés, décès et guérisons mondiaux (séries temporelles quotidiennes).
- **Our World in Data** (vaccination, dépistage) :
  https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/owid-covid-data.csv
  Jeu de données complet avec cas, décès, vaccinations, dépistage, données hospitalières et indicateurs démographiques.
- **Banque mondiale** (PIB, dépenses de santé, lits d'hôpital) :
  https://data.worldbank.org/
  Codes d'indicateurs : `SH.XPD.CHEX.PC.CD` (dépenses de santé par habitant), `SH.MED.BEDS.ZS` (lits d'hôpital pour 1 000), `NY.GDP.PCAP.CD` (PIB par habitant).
- **Africa CDC** :
  https://africacdc.org/covid-19/
  Tableaux de bord et rapports au niveau continental.

Analyses requises.

1. Sélectionner 10 à 15 pays africains de différentes sous-régions (Afrique de l'Ouest, de l'Est, australe, du Nord, centrale).
2. Tracer les courbes épidémiques (moyenne glissante 7 jours) pour chaque pays. Identifier les vagues.
3. Calculer les indicateurs clés : total des cas par million, décès par million, taux de létalité, taux de vaccination.

4. Fusionner avec les indicateurs de la Banque mondiale. Calculer les corrélations entre les résultats COVID-19 et la capacité du système de santé (dépenses de santé, lits d'hôpital, densité de médecins).

5. Construire une carte choroplèthe des taux de létalité à travers l'Afrique.

6. (Bonus) Exécuter une régression multiple : les dépenses de santé par habitant prédisent-elles le taux de létalité après ajustement sur l'âge médian et le PIB ?

Livrables.

- Courbes épidémiques pour tous les pays sélectionnés (petits multiples ou graphique groupé).
- Nuage de points des dépenses de santé vs taux de létalité.
- Carte choroplèthe de l'Afrique montrant la charge COVID-19.
- Tableau récapitulatif avec les indicateurs clés pour tous les pays.

10.2.3 Projet 3 : Déterminants de la mortalité maternelle

Objectif. Étudier quels facteurs du système de santé et socio-économiques sont associés à la mortalité maternelle à travers les pays.

Jeux de données.

- **GHO de l'OMS** — Ratio de mortalité maternelle (RMM) :
  `https://apps.who.int/gho/athena/api/GHO/MDG_0000000026?format=csv`
- **GHO de l'OMS** — Accouchements assistés par du personnel qualifié :
  `https://apps.who.int/gho/athena/api/GHO/MDG_0000000025?format=csv`
- **GHO de l'OMS** — Couverture en soins prénatals (4+ visites) :
  `https://apps.who.int/gho/athena/api/GHO/WHS4_154?format=csv`
- **Banque mondiale** — Taux d'alphabétisation des femmes, PIB par habitant, dépenses de santé :
  `https://data.worldbank.org/`
- **Programme DHS** (Enquêtes démographiques et de santé) — indicateurs au niveau des enquêtes :
  `https://www.statcompiler.com/`
  Fournit des données infranationales sur la santé maternelle, l'utilisation de la contraception et les soins prénatals pour les pays d'Afrique et d'Asie.

Analyses requises.

1. Charger les données de RMM et filtrer l'année la plus récente. Identifier les 20 pays avec le RMM le plus élevé.

2. Fusionner avec les données d'accouchements assistés, de couverture en soins prénatals et d'indicateurs socio-économiques.

3. Analyse exploratoire : nuages de points du RMM vs chaque prédicteur. Calculer les corrélations de Pearson et de Spearman.

4. Régression linéaire multiple : quels facteurs prédisent significativement le RMM ? Vérifier les hypothèses du modèle (graphiques des résidus, normalité).

5. Créer une carte choroplèthe du RMM à travers l'Afrique.

6. (Bonus) Comparer les tendances du RMM au fil du temps (2000–2020) pour 5 pays ayant mis en place des programmes spécifiques de santé maternelle.

Livrables.

- Nuages de points avec droites de régression (RMM vs accouchements assistés, RMM vs dépenses de santé).
- Tableau de régression avec coefficients, erreurs standard, valeurs p et R^2.
- Carte choroplèthe de la mortalité maternelle en Afrique.
- Discussion des implications politiques : quelles interventions pourraient réduire le RMM ?

Contexte clinique

La mortalité maternelle est l'une des inégalités de santé les plus criantes au monde. L'Afrique subsaharienne représente environ 70 % des décès maternels mondiaux. La plupart de ces décès sont évitables grâce à du personnel qualifié à l'accouchement, des soins obstétricaux d'urgence et un nombre adéquat de visites prénatales.

10.2.4 Projet 4 : Comparaison de modèles de prédiction des maladies cardiaques

Objectif. Comparer plusieurs modèles d'apprentissage automatique pour la prédiction des maladies cardiaques et identifier les prédicteurs cliniques les plus importants.

Jeu de données.

- **UCI Heart Disease Dataset** (Cleveland) :
  `https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data`
  303 patients, 13 variables cliniques (âge, sexe, type de douleur thoracique, PA au repos, cholestérol, glycémie à jeun, ECG au repos, fréquence cardiaque max, angor d'effort, dépression ST, pente, nombre de vaisseaux, thalassémie), cible binaire (présence de maladie cardiaque).
- Noms des colonnes : `age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak, slope, ca, thal, target`.

Analyses requises.

1. Charger et nettoyer les données. Gérer les valeurs manquantes (codées comme `?` dans le fichier original).

2. Analyse exploratoire : distributions de chaque variable selon le statut de maladie cardiaque.
3. Séparation entraînement-test (80/20). Utiliser une séparation stratifiée pour maintenir l'équilibre des classes.
4. Entraîner et évaluer 4 modèles :
   - Régression logistique
   - Arbre de décision
   - Forêt aléatoire
   - K plus proches voisins (KNN)
5. Pour chaque modèle : rapporter exactitude, précision, rappel, score F1 et AUC-ROC.
6. Tracer les courbes ROC des 4 modèles sur la même figure.
7. Importance des variables : quelles variables cliniques sont les prédicteurs les plus puissants ?

Livrables.

- Tableau comparatif des métriques de performance des modèles.
- Courbes ROC de tous les modèles sur un seul graphique.
- Diagramme en barres de l'importance des variables (de la forêt aléatoire).
- Matrice de confusion du modèle le plus performant.
- Interprétation clinique : les principaux prédicteurs sont-ils en accord avec les recommandations en cardiologie ?

Astuce Python

Utilisez `sklearn.model_selection.cross_val_score` avec une validation croisée à 5 plis au lieu d'une seule séparation entraînement-test. Cela donne une estimation plus fiable de la performance du modèle, surtout avec un petit jeu de données comme celui-ci (303 patients).

10.2.5 Projet 5 : Charge du paludisme et efficacité des interventions

Objectif. Analyser les tendances de la charge du paludisme dans les pays africains et évaluer si les interventions clés (moustiquaires, pulvérisation intradomiciliaire, traitement) sont associées à une baisse de l'incidence.

Jeux de données.

- **GHO de l'OMS** — Incidence du paludisme (pour 1 000 personnes à risque) : `https://apps.who.int/gho/athena/api/GHO/MALARIA_INCIDENCE?format=csv`

- **GHO de l'OMS** — Taux de mortalité du paludisme : https://apps.who.int/gho/athena/api/GHO/MALARIA_DEATHS_PER100000?format=csv
- **Rapport mondial sur le paludisme de l'OMS** — couverture des interventions (utilisation des MII, couverture PIR, traitement ACT) : https://www.who.int/teams/global-malaria-programme/reports/world-malaria-report Les tableaux de données annexes sont disponibles en téléchargement Excel.
- **The Malaria Atlas Project** : https://malariaatlas.org/ Estimations de prévalence infranationales et cartes raster.
- **Programme DHS** : https://www.statcompiler.com/ Données d'enquêtes sur la possession et l'utilisation de MII, la couverture PIR et le recours aux soins.

Analyses requises.

1. Charger les données d'incidence et de mortalité du paludisme. Filtrer les pays d'Afrique subsaharienne.
2. Tracer les tendances d'incidence (2000–2022) pour les 10 pays les plus touchés.
3. Fusionner avec les données de couverture des interventions (utilisation de MII, PIR, accès aux ACT).
4. Calculer les corrélations entre les changements de couverture des interventions et les changements d'incidence au fil du temps (c'est-à-dire : l'augmentation de l'utilisation des MII prédit-elle une diminution de l'incidence ?).
5. Créer une carte choroplèthe de l'incidence actuelle du paludisme en Afrique.
6. Régression linéaire : prédire le changement d'incidence du paludisme à partir du changement de couverture des interventions, en ajustant sur le PIB par habitant et l'urbanisation.
7. (Bonus) Créer une comparaison avant/après pour les pays ayant intensifié la distribution de MII après les subventions du Fonds mondial.

Livrables.

- Graphique de séries temporelles de l'incidence du paludisme pour 10 pays (2000–2022).
- Nuage de points du changement d'utilisation de MII vs changement d'incidence avec droite de régression.
- Carte choroplèthe de l'incidence du paludisme en Afrique.
- Tableau de régression avec interprétation.
- Discussion politique : quelles interventions montrent l'association la plus forte avec la baisse du paludisme ?

Contexte clinique

Entre 2000 et 2015, la mortalité due au paludisme en Afrique a diminué d'environ 44 %, largement grâce à l'intensification des MII, de la PIR et des thérapies combinées à base d'artémisinine (ACT). Depuis 2015, les progrès stagnent, ce qui rend cette analyse opportune et pertinente pour les politiques de santé.

10.3 Modèle de rapport

Votre rapport écrit doit suivre cette structure (5 à 8 pages, hors annexe de code) :

10.3.1 1. Introduction (0,5 à 1 page)

- Contexte sanitaire : pourquoi cette question est-elle importante ?
- Question de recherche ou objectif spécifique.
- Brève description du (des) jeu(x) de données utilisé(s).

10.3.2 2. Description des données (1 page)

- Source(s) et citation(s).
- Nombre d'observations et de variables.
- Variables clés : nom, type, unité et signification clinique.
- Tableau de statistiques descriptives (`.describe()`).
- Données manquantes : combien, quelles variables, comment traitées.

10.3.3 3. Méthodes (1 page)

- Étapes de nettoyage (gestion des valeurs manquantes, recodage des variables, traitement des valeurs aberrantes).
- Tests statistiques utilisés et pourquoi (par ex. « nous avons utilisé le test du chi-deux car les deux variables sont catégorielles »).
- Modèles utilisés (régression logistique, forêt aléatoire, etc.) avec hyperparamètres.
- Stratégie de validation (séparation entraînement-test, validation croisée).

10.3.4 4. Résultats (1,5 à 2 pages)

- Figures et tableaux avec légendes.
- Rapporter les statistiques de test, valeurs p, intervalles de confiance.
- Métriques de performance du modèle.
- Ne pas interpréter ici — présentez seulement les chiffres.

10.3.5 5. Discussion (1 à 1,5 page)

- Que signifient les résultats cliniquement ?
- Comment vos résultats se comparent-ils à la littérature publiée ?
- Qu'est-ce qui vous a surpris ?
- Quelles sont les implications pratiques ?

10.3.6 6. Limites (0,5 page)

- Problèmes de qualité des données (données manquantes, erreurs de mesure, biais de sélection).
- Généralisabilité : le jeu de données représente-t-il la population qui vous intéresse ?
- Quelles analyses feriez-vous avec plus de temps ou de données ?

Attention

Chaque figure doit avoir un titre, des axes étiquetés et une légende expliquant ce que le lecteur doit y voir. Chaque tableau doit avoir une ligne d'en-tête et une légende. Les figures sans étiquettes sont la raison la plus fréquente de perte de points.

10.4 Consignes de présentation

Vous ferez une présentation orale de 5 minutes suivie de 2 à 3 minutes de questions.

10.4.1 Structure

1. **Diapositive 1 : Titre.** Titre du projet, votre nom, date.
2. **Diapositive 2 : Motivation.** Pourquoi cette question de santé est-elle importante ? Un ou deux faits clés.
3. **Diapositive 3 : Données.** Quelles données avez-vous utilisées ? Combien d'observations ? Quelles sont les variables clés ?
4. **Diapositives 4–5 : Résultats clés.** Vos 2 ou 3 figures ou tableaux les plus importants. Guidez le public à travers chacun.
5. **Diapositive 6 : Conclusions.** Un ou deux messages à retenir. Que doit retenir le public ?

10.4.2 Conseils

- **5 minutes, c'est court.** Entraînez-vous avec un chronomètre. Éliminez tout ce qui n'est pas essentiel.
- **Un message par diapositive.** Ne surchargez pas les diapositives de texte.
- **Parlez au public, pas à l'écran.**
- **Anticipez les questions.** Connaissez vos limites et soyez prêt à en discuter.
- **Pas de code sur les diapositives.** Montrez les résultats, pas le code qui les a produits.

Astuce Python

Sauvegardez vos figures en PNG haute résolution pour les diapositives :

```
fig.savefig("figure1.png", dpi=300, bbox_inches="tight")
```

Cela évite les captures d'écran floues et garantit que vos graphiques sont professionnels dans une présentation.

10.5 Barème de notation

Le projet de synthèse est noté sur 100 points :

Composante	Points	Critères
Question de recherche	10	Claire, spécifique et pertinente pour la santé
Gestion des données	15	Chargement, nettoyage, traitement des valeurs manquantes et documentation adéquats
Analyse exploratoire	15	Statistiques descriptives, distributions et au moins 3 visualisations bien étiquetées
Analyse statistique	20	Choix et application corrects des tests ou modèles ; interprétation adéquate des valeurs p, intervalles de confiance et tailles d'effet
Qualité du rapport	20	Suit le modèle ; rédaction claire ; figures avec titres, étiquettes et légendes ; limites honnêtement discutées
Qualité du code	10	Notebook reproductible qui s'exécute du début à la fin ; commentaires expliquant les étapes clés ; pas de code inutile
Présentation orale	10	Claire, dans la limite de temps, répond aux questions avec assurance
Total	**100**	

10.5.1 Limites de notes

- **A (90–100) :** Excellente analyse avec interprétation perspicace. Code propre et bien documenté. Va au-delà des exigences minimales.
- **B (75–89) :** Analyse solide avec méthodologie correcte. Problèmes mineurs de présentation ou d'interprétation.
- **C (60–74) :** Répond aux exigences de base mais présente des lacunes notables (par ex. visualisations manquantes, interprétation superficielle, erreurs de code).
- **D/F (< 60) :** Analyse incomplète, erreurs méthodologiques majeures ou code non reproductible.

10.6 Pour démarrer : une liste de vérification

Exercice

Avant de partir aujourd'hui, complétez cette liste de vérification :

1. Choisissez votre projet (1 à 5 ou une proposition personnalisée).
2. Téléchargez votre jeu de données principal. Chargez-le dans un notebook Jupyter et exécutez `.head()`, `.shape`, `.info()` et `.describe()`.
3. Rédigez votre question de recherche en une phrase.
4. Listez 3 visualisations que vous prévoyez de créer.
5. Listez 1 test statistique ou modèle que vous prévoyez d'utiliser.
6. Identifiez les problèmes potentiels : données manquantes, noms de colonnes désordonnés, multiples fichiers à fusionner.
7. Créez un dossier de projet avec la structure :

```
projet_synthese/
    data/
        raw/          # fichiers originaux telecharges
        cleaned/      # donnees traitees
    notebooks/
        01_exploration.ipynb
        02_analyse.ipynb
    figures/
    rapport.pdf
```

8. Soumettez votre choix de projet et votre question de recherche à l'enseignant avant la fin de la semaine.

10.7 Résumé du chapitre

- Le projet de synthèse intègre toutes les compétences du cours : chargement, nettoyage, exploration, visualisation, analyse statistique et interprétation des données.
- Cinq projets suggérés couvrent le diabète, la COVID-19, la mortalité maternelle, les maladies cardiaques et le paludisme — tous utilisant des jeux de données publics et réels.
- Chaque projet exige une question de santé claire, du code reproductible, au moins 3 visualisations et au moins 1 test statistique ou modèle.
- Le rapport suit une structure standard : Introduction, Données, Méthodes, Résultats, Discussion, Limites.
- Les présentations durent 5 minutes — concentrez-vous sur la motivation, les résultats clés et les conclusions.
- Le barème de notation récompense la clarté de la réflexion et l'honnêteté de l'interprétation plutôt que la complexité méthodologique.

Annexe A : Guide d'installation de Python

Option 1 : Google Colab (recommandé pour les débutants)

Rendez-vous sur https://colab.research.google.com. Connectez-vous avec un compte Google. Aucune installation nécessaire. Toutes les bibliothèques utilisées dans ce cours sont préinstallées.

Option 2 : Anaconda (installation locale)

Téléchargez Anaconda depuis https://www.anaconda.com/download. Installez avec les paramètres par défaut. Ouvrez Jupyter Notebook depuis Anaconda Navigator.

Bibliothèques requises

```
pip install pandas matplotlib seaborn scipy scikit-learn lifelines geopandas
```

Annexe B : Sources de données de santé

Source	Description	URL
OMS GHO	Indicateurs de santé mondiaux (190+ pays, 1000+ indicateurs)	who.int/data/gho
Gapminder	Santé, revenu, population par pays et année	gapminder.org/data
Our World in Data	COVID-19, mortalité, charge de morbidité, vaccination	ourworldindata.org
CDC NHANES	Enquête nationale de santé américaine (laboratoire, questionnaires, examens)	cdc.gov/nchs/nhanes
UCI ML Repository	Jeux de données d'AA sélectionnés (maladies cardiaques, diabète, cancer)	archive.ics.uci.edu
Banque mondiale	Dépenses de santé, espérance de vie, mortalité maternelle	data.worldbank.org
Africa CDC	Surveillance des maladies et données d'épidémies en Afrique	africacdc.org
JHU COVID-19	Cas, décès et guérisons COVID-19 mondiaux (séries temporelles)	github.com/CSSEGISandData
Programme DHS	Enquêtes démographiques et de santé (50+ pays africains)	dhsprogram.com