Introduction à la Science des Données

Notes de Cours

Licence L3 — 2025–2026

Yaë Ulrich Gaba

« Tous les modèles sont faux, mais certains sont utiles. »
— George E. P. Box

25 mars 2026

Table des matières

Préface

La **science des données** est une discipline qui combine mathématiques, statistiques et informatique pour extraire des connaissances à partir de données. Elle se distingue de la statistique classique par son échelle (volume de données), ses outils (programmation, apprentissage automatique) et son orientation pratique (décisions, produits, prédictions).

Ce cours est conçu pour des étudiants de Licence ayant des bases en Python et en statistiques. Chaque chapitre part d'un **jeu de données réel**, introduit les concepts et les met en œuvre en Python. L'objectif est de rendre l'étudiant autonome pour mener une analyse de données complète, de l'import à la communication des résultats.

Prérequis. Programmation Python de base (variables, boucles, fonctions), notions de statistique descriptive et de probabilités.

Jeux de données utilisés. Iris, Titanic, Tips (Seaborn), California Housing, 20 Newsgroups, AirPassengers — tous librement accessibles.

Références.

- VanderPlas — *Python Data Science Handbook*, O'Reilly (disponible gratuitement en ligne).
- McKinney — *Python for Data Analysis*, O'Reilly.
- Géron — *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly.
- James, Witten, Hastie, Tibshirani — *An Introduction to Statistical Learning (ISLR)*, Springer (gratuit).

Chapitre 1

Le Pipeline de la Science des Données

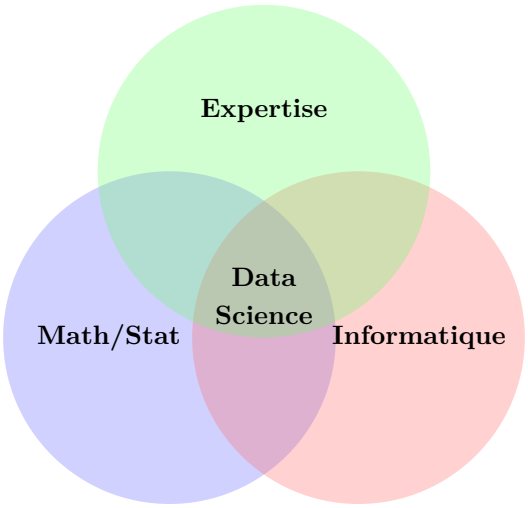
1.1 Qu'est-ce que la science des données ?

Définition 1.1 (Science des données)**.** La **science des données** (data science) est une discipline interdisciplinaire qui utilise des méthodes scientifiques, des algorithmes et des systèmes pour extraire des connaissances et des informations à partir de données structurées et non structurées.

Intuition

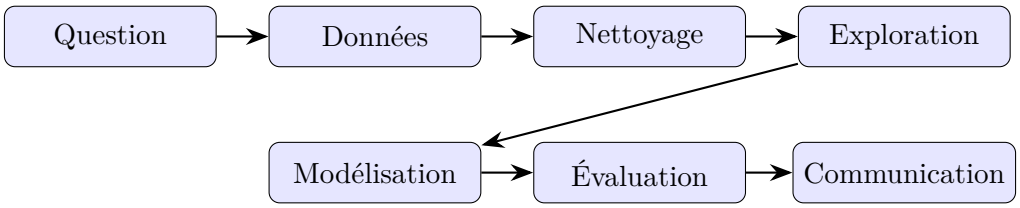
La science des données se situe à l'intersection de trois domaines :

- **Mathématiques et Statistiques** : modélisation, inférence, probabilités.
- **Informatique** : programmation, algorithmes, bases de données.
- **Expertise métier** : compréhension du domaine d'application.



1.2 Le pipeline typique

Un projet de science des données suit généralement un pipeline en plusieurs étapes :



1. **Formulation de la question** : définir clairement le problème.
2. **Collecte des données** : import depuis des fichiers, bases de données, API.
3. **Nettoyage** : traiter les valeurs manquantes, les doublons, les erreurs.
4. **Exploration** (EDA) : visualiser et résumer les données.
5. **Modélisation** : appliquer des modèles statistiques ou de machine learning.
6. **Évaluation** : mesurer la performance du modèle.
7. **Communication** : présenter les résultats de manière claire.

1.3 L'écosystème Python pour la data science

Bibliothèques essentielles

Bibliothèque	Rôle
`numpy`	Calcul numérique, tableaux
`pandas`	Manipulation de données tabulaires
`matplotlib`	Visualisation de base
`seaborn`	Visualisation statistique
`scikit-learn`	Apprentissage automatique
`scipy`	Statistiques et optimisation

Python — Installation et imports

```python
# Installation (dans un terminal)
# pip install numpy pandas matplotlib seaborn scikit-learn

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
from sklearn.linear_model import LinearRegression

print(f"NumPy:   {np.__version__}")
print(f"Pandas:  {pd.__version__}")
print(f"Seaborn: {sns.__version__}")
```

Sortie

```
NumPy:   1.26.4
Pandas:  2.2.1
Seaborn: 0.13.2
```

1.4 Premier exemple complet : le jeu de données Iris

Illustrons le pipeline complet sur le jeu de données **Iris** de Fisher (1936), qui contient 150 mesures de fleurs d'iris réparties en 3 espèces.

Python — Charger et explorer Iris

```python
from sklearn.datasets import load_iris

# 1. Charger les données
iris = load_iris(as_frame=True)
df = iris.frame
print(df.shape)
print(df.head())
```

Sortie

```
(150, 5)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target
0                5.1               3.5                1.4               0.2       0
1                4.9               3.0                1.4               0.2       0
2                4.7               3.2                1.3               0.2       0
3                4.6               3.1                1.5               0.2       0
4                5.0               3.6                1.4               0.2       0
```

Python — Statistiques descriptives

```python
# 2. Explorer
print(df.describe().round(2))
print(f"\nValeurs manquantes :\n{df.isnull().sum()}")
```

Sortie

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  ta
count             150.00            150.00             150.00            150.00  15
mean                5.84              3.06               3.76              1.20
std                 0.83              0.44               1.77              0.76
min                 4.30              2.00               1.00              0.10
25%                 5.10              2.80               1.60              0.30
50%                 5.80              3.00               4.35              1.30
75%                 6.40              3.30               5.10              1.80
max                 7.90              4.40               6.90              2.50
```

```
Valeurs manquantes :
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
```

Python — Visualisation

```python
# 3. Visualiser
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Histogramme
for species in [0, 1, 2]:
    subset = df[df['target'] == species]
    axes[0].hist(subset['petal length (cm)'], alpha=0.6,
                 label=iris.target_names[species], bins=15)
axes[0].set_xlabel('Longueur du pétale (cm)')
axes[0].set_ylabel('Fréquence')
axes[0].legend()

# Nuage de points
for species in [0, 1, 2]:
    subset = df[df['target'] == species]
    axes[1].scatter(subset['sepal length (cm)'],
                    subset['petal length (cm)'],
                    label=iris.target_names[species], alpha=0.7)
axes[1].set_xlabel('Longueur du sépale (cm)')
axes[1].set_ylabel('Longueur du pétale (cm)')
axes[1].legend()
plt.tight_layout()
plt.savefig('ch01_iris_explore.pdf')
plt.show()
```

Python — Modélisation simple

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# 4. Modéliser
X = df[['sepal length (cm)', 'sepal width (cm)',
        'petal length (cm)', 'petal width (cm)']]
y = df['target']

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

knn = KNeighborsClassifier(n_neighbors=5)
```

```python
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# 5. Évaluer
print(f"Accuracy : {accuracy_score(y_test, y_pred):.2%}")
```

Sortie

```
Accuracy : 100.00%
```

1.5 Types de problèmes en data science

Supervisé	**Non supervisé**	**Renforcement**
Régression, Classification	Clustering, Réduction	Agent, Récompense

- **Apprentissage supervisé** : on dispose d'un label y pour chaque observation $\boldsymbol{x}$.
  - *Régression* : $y \in \mathbb{R}$ (prédire un prix, une température).
  - *Classification* : $y \in \{0, 1, \ldots, K-1\}$ (détecter un spam, un cancer).
- **Apprentissage non supervisé** : pas de label, on cherche des structures (clusters, composantes principales).
- **Apprentissage par renforcement** : un agent apprend par essai-erreur dans un environnement.

1.6 Bonnes pratiques

Bonne pratique

1. **Reproductibilité** : fixer les graines aléatoires (`random_state=42`), versionner le code.
2. **Séparation train/test** : ne jamais évaluer sur les données d'entraînement.
3. **Documentation** : commenter le code, noter les hypothèses.
4. **Itération** : commencer simple, complexifier progressivement.

Attention

Ne jamais regarder les données de test avant d'avoir finalisé son modèle. Le *data leakage* (fuite de données) est l'erreur la plus courante et la plus dangereuse en data science.

1.7 Exercices

Exercice 1.1 ($\star$)**.** Charger le jeu de données `load_wine()` de scikit-learn. Afficher ses dimensions, les noms des variables et les statistiques descriptives.

Exercice 1.2 ($\star$)**.** Sur le jeu Iris, créer un nuage de points (*scatter plot*) de la largeur du sépale en fonction de la longueur du sépale, coloré par espèce.

Exercice 1.3 ($\star\star$)**.** Appliquer un classifieur k-plus proches voisins au jeu Wine avec $k = 3, 5, 7$. Comparer les précisions sur un jeu de test de 30%.

Exercice 1.4 ($\star\star$)**.** Écrire une fonction `pipeline_complet(dataset, k)` qui charge un jeu de données scikit-learn, le divise en train/test, entraîne un KNN avec k voisins et retourne l'accuracy.

Résumé du chapitre

- La science des données suit un pipeline : Question → Données → Nettoyage → Exploration → Modélisation → Évaluation → Communication.
- L'écosystème Python offre des outils puissants : `pandas`, `matplotlib`, `scikit-learn`.
- Trois grandes familles : supervisé, non supervisé, renforcement.
- Règles d'or : reproductibilité, séparation train/test, commencer simple.

Chapitre 2

Manipulation de Données avec Pandas

2.1 Introduction à Pandas

Pandas est la bibliothèque centrale de la data science en Python. Elle fournit deux structures de données fondamentales : le `Series` (vecteur indexé) et le `DataFrame` (tableau indexé).

Définition 2.1 (DataFrame)**.** Un **DataFrame** est un tableau bidimensionnel avec des étiquettes pour les lignes (index) et les colonnes. Chaque colonne est une `Series` de même longueur.

2.2 Création de DataFrames

Python — Création

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

# À partir d'un dictionnaire
data = {
    'Nom': ['Alice', 'Bob', 'Charlie', 'Diana'],
    'Âge': [25, 30, 35, 28],
    'Salaire': [45000, 55000, 70000, 52000],
    'Ville': ['Paris', 'Lyon', 'Paris', 'Marseille']
}
df = pd.DataFrame(data)
print(df)
print(f"\nShape : {df.shape}")
print(f"Types :\n{df.dtypes}")
```

Sortie

```
      Nom  Âge  Salaire      Ville
0   Alice   25    45000      Paris
```

```
1      Bob   30    55000       Lyon
2  Charlie   35    70000      Paris
3    Diana   28    52000  Marseille

Shape : (4, 4)
Types :
Nom         object
Âge          int64
Salaire      int64
Ville       object
dtype: object
```

2.3 Chargement de données

Python — Lecture de fichiers

```python
# CSV (le plus courant)
# df = pd.read_csv('donnees.csv')
# df = pd.read_csv('donnees.csv', sep=';', encoding='utf-8')

# Depuis Seaborn (jeux de données intégrés)
tips = sns.load_dataset('tips')   # pourboires au restaurant
print(f"Dimensions : {tips.shape}")
print(tips.head(3))
```

Sortie

```
Dimensions : (244, 7)
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
```

2.4 Sélection et indexation

Méthodes de sélection

Syntaxe	Description
`df['col']`	Sélectionner une colonne (Series)
`df[['c1','c2']]`	Sélectionner plusieurs colonnes
`df.loc[i, 'col']`	Sélection par étiquette
`df.iloc[i, j]`	Sélection par position
`df[df['col'] > x]`	Filtrage booléen

Python — Sélection

```python
import seaborn as sns
tips = sns.load_dataset('tips')

# Sélection de colonnes
print("--- Colonne unique ---")
print(tips['total_bill'].head(3))

# Filtrage
gros_pourboires = tips[tips['tip'] > 5]
print(f"\n--- Pourboires > 5$ : {len(gros_pourboires)} lignes ---")
print(gros_pourboires.head(3))

# Sélection combinée
print("\n--- loc : femmes, dimanche ---")
mask = (tips['sex'] == 'Female') & (tips['day'] == 'Sun')
print(tips.loc[mask, ['total_bill', 'tip']].head(3))
```

Sortie

```
--- Colonne unique ---
0    16.99
1    10.34
2    21.01
Name: total_bill, dtype: float64

--- Pourboires > 5$ : 17 lignes ---
     total_bill   tip   sex smoker  day    time  size
23        39.42  7.58  Male     No  Sat  Dinner     4
44        30.40  5.60  Male     No  Sun  Dinner     4
47        32.40  6.00  Male     No  Sun  Dinner     4

--- loc : femmes, dimanche ---
     total_bill   tip
0         16.99  1.01
5         25.29  4.71
15        21.58  3.92
```

2.5 Modification et ajout de colonnes

Python — Nouvelles colonnes

```python
# Ajouter une colonne calculée
tips['tip_pct'] = (tips['tip'] / tips['total_bill'] * 100).round(1)
print(tips[['total_bill', 'tip', 'tip_pct']].head(5))

# Appliquer une fonction
```

```python
tips['bill_category'] = tips['total_bill'].apply(
    lambda x: 'Élevé' if x > 30 else ('Moyen' if x > 15 else 'Bas'))
print(tips['bill_category'].value_counts())
```

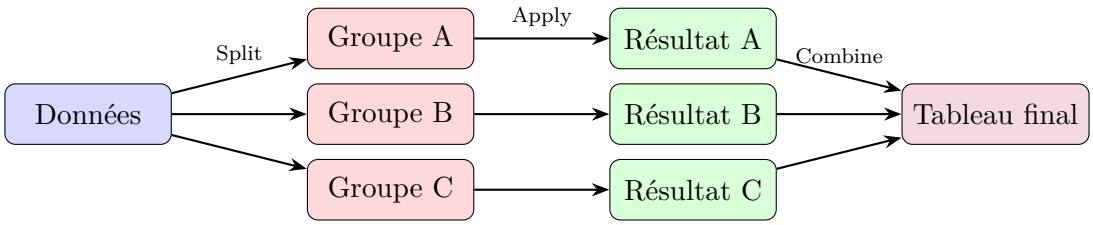
Sortie

```
   total_bill   tip  tip_pct
0       16.99  1.01      5.9
1       10.34  1.66     16.1
2       21.01  3.50     16.7
3       23.68  3.31     14.0
4       24.59  3.61     14.7

bill_category
Moyen     111
Bas        69
Élevé      64
Name: count, dtype: int64
```

2.6 Agrégation avec groupby

Définition 2.2 (Split-Apply-Combine)**.** Le paradigme **split-apply-combine** consiste à :

1. **Split** : diviser les données en groupes.
2. **Apply** : appliquer une fonction (somme, moyenne, comptage…) à chaque groupe.
3. **Combine** : combiner les résultats.



Python — groupby

```python
# Moyenne par jour
print("--- Moyenne par jour ---")
print(tips.groupby('day')[['total_bill', 'tip']].mean().round(2))

# Plusieurs agrégations
print("\n--- Agrégations multiples ---")
agg = tips.groupby(['day', 'time']).agg(
    nb_repas=('total_bill', 'count'),
    addition_moy=('total_bill', 'mean'),
    pourboire_moy=('tip', 'mean')
).round(2)
```

```
print(agg)
```

Sortie

```
--- Moyenne par jour ---
      total_bill   tip
day
Thur       17.68  2.77
Fri        17.15  2.73
Sat        20.44  2.99
Sun        21.41  3.26

--- Agrégations multiples ---
             nb_repas  addition_moy  pourboire_moy
day  time
Thur Lunch         61         17.66           2.77
     Dinner         1         18.78           3.00
Fri  Lunch          7         12.85           2.38
     Dinner        12         19.66           2.94
Sat  Dinner        87         20.44           2.99
Sun  Dinner        76         21.41           3.26
```

2.7 Tableau croisé dynamique

Python — pivot_table

```
pivot = tips.pivot_table(
    values='tip', index='day', columns='sex',
    aggfunc='mean').round(2)
print(pivot)
```

Sortie

```
sex   Female  Male
day
Thur    2.58  2.88
Fri     2.78  2.69
Sat     2.80  3.08
Sun     3.37  3.22
```

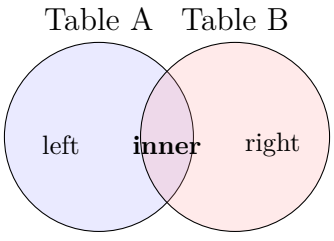
2.8 Jointures

Python — merge

```python
# Deux DataFrames
clients = pd.DataFrame({
    'client_id': [1, 2, 3, 4],
    'nom': ['Alice', 'Bob', 'Charlie', 'Diana']
})
commandes = pd.DataFrame({
    'commande_id': [101, 102, 103, 104],
    'client_id': [1, 2, 2, 5],
    'montant': [150, 200, 80, 300]
})

# Inner join
inner = pd.merge(clients, commandes, on='client_id', how='inner')
print("--- Inner join ---")
print(inner)

# Left join (garder tous les clients)
left = pd.merge(clients, commandes, on='client_id', how='left')
print("\n--- Left join ---")
print(left)
```

Sortie

```
--- Inner join ---
   client_id      nom  commande_id  montant
0          1    Alice          101      150
1          2      Bob          102      200
2          2      Bob          103       80

--- Left join ---
   client_id      nom  commande_id  montant
0          1    Alice        101.0    150.0
1          2      Bob        102.0    200.0
2          2      Bob        103.0     80.0
3          3  Charlie          NaN      NaN
4          4    Diana          NaN      NaN
```



outer = A ∪ B

2.9 Tri et classement

Python — Tri

```python
# Trier par pourboire décroissant
top5 = tips.nlargest(5, 'tip')[['total_bill', 'tip', 'day']]
print(top5)

# Rang
tips['tip_rank'] = tips['tip'].rank(ascending=False).astype(int)
print(tips[['total_bill', 'tip', 'tip_rank']].head(5))
```

Sortie

```
     total_bill   tip  day
170       50.81 10.00  Sat
212       48.33  9.00  Sat
23        39.42  7.58  Sat
59        48.27  6.73  Sat
183       23.17  6.50  Sun

   total_bill   tip  tip_rank
0       16.99  1.01       224
1       10.34  1.66       188
2       21.01  3.50        52
3       23.68  3.31        63
4       24.59  3.61        46
```

2.10 Opérations sur les chaînes

Python — Accesseur `.str`

```python
noms = pd.Series(['  Alice Dupont  ', 'bob martin', 'CHARLIE PETIT'])

print(noms.str.strip().str.title())
print(noms.str.contains('art', case=False))
```

Sortie

```
0     Alice Dupont
1       Bob Martin
2    Charlie Petit
dtype: object
0    False
1     True
2    False
dtype: bool
```

2.11 Exercices

Exercice 2.1 ($\star$)**.** Charger le jeu `tips` de Seaborn. Afficher le nombre de repas par jour de la semaine et par moment (déjeuner/dîner).

Exercice 2.2 ($\star$)**.** Créer un DataFrame avec 5 étudiants, leurs notes en maths et en physique. Ajouter une colonne « moyenne » et trier par moyenne décroissante.

Exercice 2.3 ($\star\star$)**.** Sur le jeu Tips, calculer pour chaque combinaison (sexe, fumeur) : le nombre de repas, l'addition moyenne et le pourcentage moyen de pourboire.

Exercice 2.4 ($\star\star$)**.** Créer deux DataFrames : un avec des produits (id, nom, catégorie), un avec des ventes (id_produit, quantité, date). Les joindre et calculer le chiffre d'affaires par catégorie.

Exercice 2.5 ($\star\star\star$)**.** Écrire une fonction qui prend un DataFrame et retourne un rapport automatique : nombre de lignes, types de colonnes, pourcentage de valeurs manquantes par colonne, nombre de doublons.

Fonctions Pandas essentielles

- `pd.read_csv()`, `df.to_csv()` — import/export
- `df.head()`, `df.info()`, `df.describe()` — exploration
- `df.loc[]`, `df.iloc[]` — sélection
- `df.groupby().agg()` — agrégation
- `pd.merge()` — jointures
- `df.sort_values()`, `df.nlargest()` — tri

Chapitre 3

Visualisation des Données

Intuition

La visualisation est le premier outil du data scientist. Un bon graphique révèle des structures, des anomalies et des relations qu'aucun tableau de chiffres ne peut montrer aussi clairement. Comme le disait John Tukey : « The greatest value of a picture is when it forces us to notice what we never expected to see. »

3.1 La grammaire des graphiques

La **grammaire des graphiques**, proposée par Leland Wilkinson, décompose tout graphique en couches :

1. **Données** : le jeu de données source.
2. **Esthétiques** (*aesthetics*) : les variables mappées sur les axes, couleurs, tailles.
3. **Géométries** : le type de représentation (points, barres, lignes).
4. **Facettes** : le découpage en sous-graphiques.
5. **Statistiques** : les transformations appliquées (comptage, moyenne).
6. **Coordonnées** : le système de coordonnées (cartésien, polaire).
7. **Thème** : l'apparence générale (polices, grille, fond).

Seaborn et Matplotlib implémentent ces concepts en Python.

3.2 Matplotlib : les fondamentaux

3.2.1 Graphique en ligne

Python

```
import matplotlib.pyplot as plt
import numpy as np
```

```python
x = np.linspace(0, 2 * np.pi, 100)
y_sin = np.sin(x)
y_cos = np.cos(x)

fig, ax = plt.subplots(figsize=(7, 4))
ax.plot(x, y_sin, label='sin(x)', color='steelblue', linewidth=2)
ax.plot(x, y_cos, label='cos(x)', color='coral', linestyle='--')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Fonctions trigonom\'etriques')
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('trigo.pdf', dpi=150)
plt.show()
```

Sortie

Un graphique avec deux courbes (sinus en bleu, cosinus en tirets corail) sur $[0, 2\pi]$, avec légende, grille et titres.

3.2.2 Nuage de points et histogramme

Python

```python
import seaborn as sns

tips = sns.load_dataset('tips')

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Nuage de points
axes[0].scatter(tips['total_bill'], tips['tip'],
                alpha=0.6, c='teal', edgecolors='white')
axes[0].set_xlabel('Addition totale (\$)')
axes[0].set_ylabel('Pourboire (\$)')
axes[0].set_title('Pourboire vs Addition')

# Histogramme
axes[1].hist(tips['total_bill'], bins=20, color='steelblue',
             edgecolor='white')
axes[1].set_xlabel('Addition totale (\$)')
axes[1].set_ylabel('Fr\'equence')
axes[1].set_title('Distribution des additions')

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

Sortie

Deux sous-graphiques : à gauche un nuage de points montrant une relation positive entre addition et pourboire ; à droite un histogramme légèrement asymétrique (queue droite) des additions.

3.2.3 Diagramme en barres

Python

```
moyenne_jour = tips.groupby('day')['tip'].mean().sort_values()

fig, ax = plt.subplots(figsize=(6, 4))
ax.bar(moyenne_jour.index, moyenne_jour.values,
       color=['#4c72b0', '#55a868', '#c44e52', '#8172b2'])
ax.set_xlabel('Jour')
ax.set_ylabel('Pourboire moyen (\$)')
ax.set_title('Pourboire moyen par jour')
for i, v in enumerate(moyenne_jour.values):
    ax.text(i, v + 0.05, f'{v:.2f}', ha='center', fontsize=10)
plt.tight_layout()
plt.show()
```

Sortie

Diagramme en barres colorées avec les valeurs annotées : Fri 2.73, Thur 2.77, Sat 2.99, Sun 3.26.

3.3 Seaborn : visualisation statistique

3.3.1 Distributions : histplot et violinplot

Python

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.histplot(data=tips, x='total_bill', hue='sex', kde=True,
             ax=axes[0], palette='Set2')
axes[0].set_title('Distribution par sexe')

sns.violinplot(data=tips, x='day', y='total_bill',
               hue='sex', split=True, ax=axes[1],
               palette='pastel')
axes[1].set_title('Violin plot par jour et sexe')

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

Sortie

À gauche : histogrammes superposés avec courbes KDE pour homme et femme. À droite : violin plots séparés par sexe pour chaque jour, montrant que les hommes ont des additions légèrement plus élevées.

3.3.2 Boîtes à moustaches et heatmap

Python

```
fig, axes = plt.subplots(1, 2, figsize=(13, 5))

sns.boxplot(data=tips, x='day', y='tip', hue='smoker',
            ax=axes[0], palette='coolwarm')
axes[0].set_title('Pourboires par jour et statut fumeur')

corr = tips[['total_bill', 'tip', 'size']].corr()
sns.heatmap(corr, annot=True, fmt='.2f', cmap='YlOrRd',
            ax=axes[1], vmin=0, vmax=1)
axes[1].set_title('Matrice de corr\'elation')

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

Sortie

Boxplots montrant des distributions similaires entre fumeurs et non-fumeurs. Heatmap : corrélation total_bill/tip = 0.68, total_bill/size = 0.60, tip/size = 0.49.

3.3.3 Pairplot et relplot

Python

```
iris = sns.load_dataset('iris')

g = sns.pairplot(iris, hue='species', palette='husl',
                 diag_kind='kde', height=2.2)
g.figure.suptitle('Pairplot du jeu Iris', y=1.02)
plt.show()
```

Sortie

Matrice 4×4 de graphiques : diagonale = densités KDE par espèce, hors-diagonale = nuages de points. On distingue clairement *setosa* des deux autres espèces.

Python

```
sns.relplot(data=tips, x='total_bill', y='tip',
            hue='smoker', style='sex', col='time',
            height=4, aspect=1.2, palette='Set1')
plt.show()
```

Sortie

Deux panneaux (Lunch / Dinner) avec nuages de points colorés par statut fumeur et différenciés par forme selon le sexe.

3.3.4 catplot

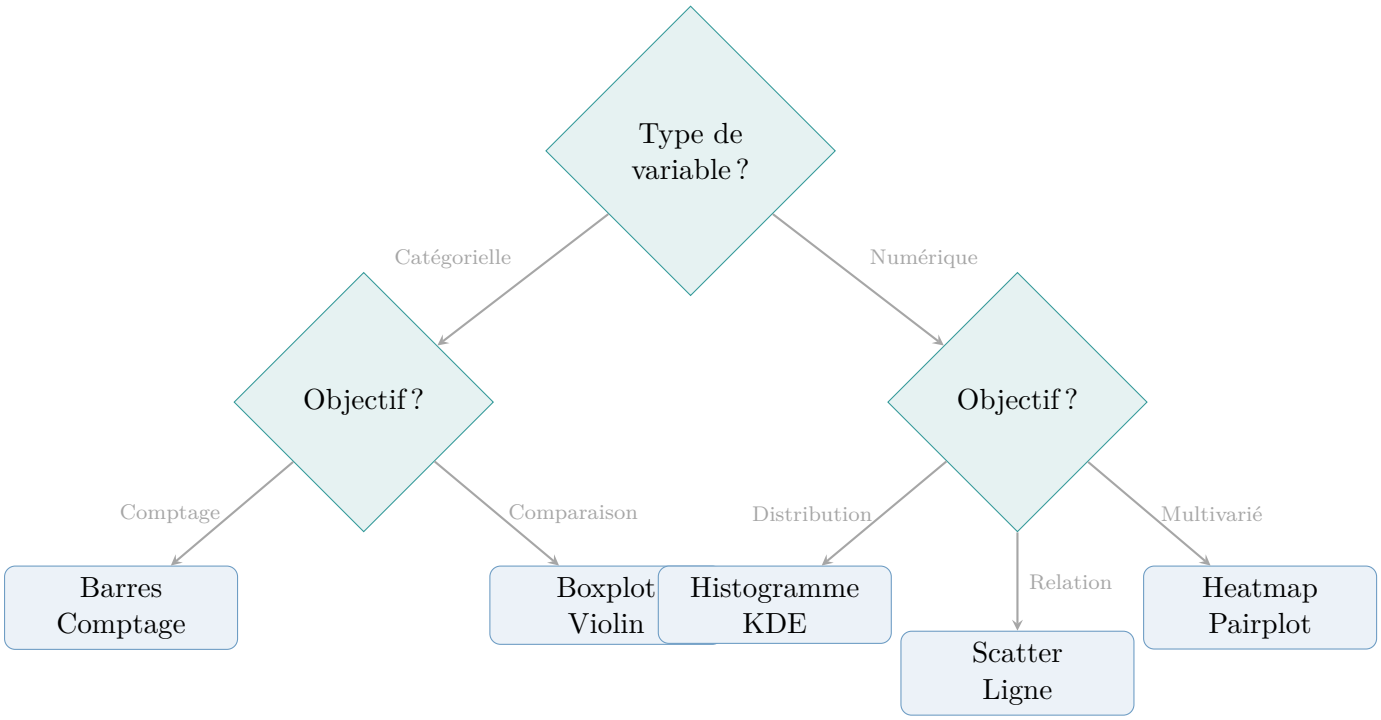
Python

```
sns.catplot(data=tips, x='day', y='total_bill', hue='sex',
            kind='box', col='time', palette='muted',
            height=4, aspect=1)
plt.show()
```

Sortie

Quatre groupes de boxplots organisés en deux colonnes (Lunch, Dinner), comparés par sexe et par jour.

3.4 Choisir le bon type de graphique



3.5 Palettes de couleurs et sauvegarde

Python

```python
# Palettes disponibles dans Seaborn
palettes = ['deep', 'muted', 'pastel', 'bright',
            'dark', 'colorblind']
sns.set_palette('colorblind')  # recommandé pour l'accessibilité

# Sauvegarde haute résolution
fig, ax = plt.subplots()
sns.histplot(tips['tip'], bins=15, ax=ax, color='steelblue')
fig.savefig('distribution_tip.png', dpi=300, bbox_inches='tight')
fig.savefig('distribution_tip.pdf', bbox_inches='tight')
```

Bonne pratique

Utilisez toujours la palette `'colorblind'` pour garantir l'accessibilité de vos graphiques. Sauvegardez en PDF pour les rapports et en PNG (300 dpi) pour les présentations.

3.6 Erreurs courantes en visualisation

Attention

- **Axes tronqués** : ne pas commencer l'axe y à zéro dans un diagramme en barres exagère les différences.
- **Échelles trompeuses** : utiliser deux axes y avec des échelles différentes peut suggérer de fausses corrélations.
- **Graphiques 3D** : les perspectives déforment la perception des valeurs. Préférez le 2D.
- **Trop de catégories** : un camembert avec 15 parts est illisible. Utilisez un diagramme en barres.
- **Absence de légende et de titres** : tout graphique doit être compréhensible seul.

3.7 Exercices

Exercice 3.1 ($\star$)**.** Chargez le jeu `tips` et créez un histogramme de la variable `tip` avec 20 intervalles. Ajoutez un titre, des labels d'axes et une courbe KDE superposée avec Seaborn.

Exercice 3.2 ($\star$)**.** Créez un diagramme en barres horizontales montrant le nombre de repas par jour dans le jeu `tips`. Triez les barres par ordre décroissant.

Exercice 3.3 ($\star\star$)**.** À l'aide du jeu `iris`, créez une figure avec 4 sous-graphiques (2×2) montrant la distribution de chaque variable numérique, colorée par espèce. Utilisez `sns.histplot` avec `hue='species'`.

Exercice 3.4 ($\star\star$)**.** Créez un `sns.catplot` de type `'violin'` montrant la distribution de `total\_bill` par `day`, facetté par `time`, avec la couleur déterminée par `sex`. Interprétez les différences observées.

Exercice 3.5 ($\star\star\star$)**.** Reproduisez le graphique suivant : une figure à deux panneaux. Le panneau gauche montre un scatter plot de `total\_bill` vs `tip` avec une droite de régression (`sns.regplot`). Le panneau droit montre les résidus de cette régression (calculez-les avec NumPy : `np.polyfit` et `np.polyval`). Commentez la qualité de l'ajustement.

Exercice 3.6 ($\star\star\star$)**.** Prenez le jeu `iris` et créez une heatmap de la matrice de corrélation *par espèce* (trois heatmaps dans une figure 1×3). Comparez les structures de corrélation entre espèces.

Fonctions clés

Résumé du chapitre – Visualisation des données

- **Matplotlib** : `plt.subplots()`, `ax.plot()`, `ax.scatter()`, `ax.hist()`, `ax.bar()`.
- **Seaborn (distributions)** : `sns.histplot()`, `sns.boxplot()`,

`sns.violinplot()`.

- **Seaborn (relations)** : `sns.scatterplot()`, `sns.relplot()`, `sns.heatmap()`.
- **Seaborn (catégoriel)** : `sns.catplot()`, `sns.countplot()`, `sns.barplot()`.
- **Seaborn (multivarié)** : `sns.pairplot()`.
- **Personnalisation** : `set\_xlabel()`, `set\_title()`, `legend()`, `grid()`.
- **Sauvegarde** : `fig.savefig('nom.pdf', dpi=300, bbox\_inches='tight')`.
- **Règle d'or** : choisir le graphique adapté au type de variable et à l'objectif d'analyse.

Chapitre 4

Analyse Exploratoire des Données (EDA)

Intuition

L'analyse exploratoire des données (EDA) est une philosophie d'analyse initiée par John Tukey dans les années 1970. Plutôt que de tester des hypothèses prédéfinies, l'EDA invite à « laisser parler les données » en les explorant systématiquement par des résumés numériques et des visualisations.

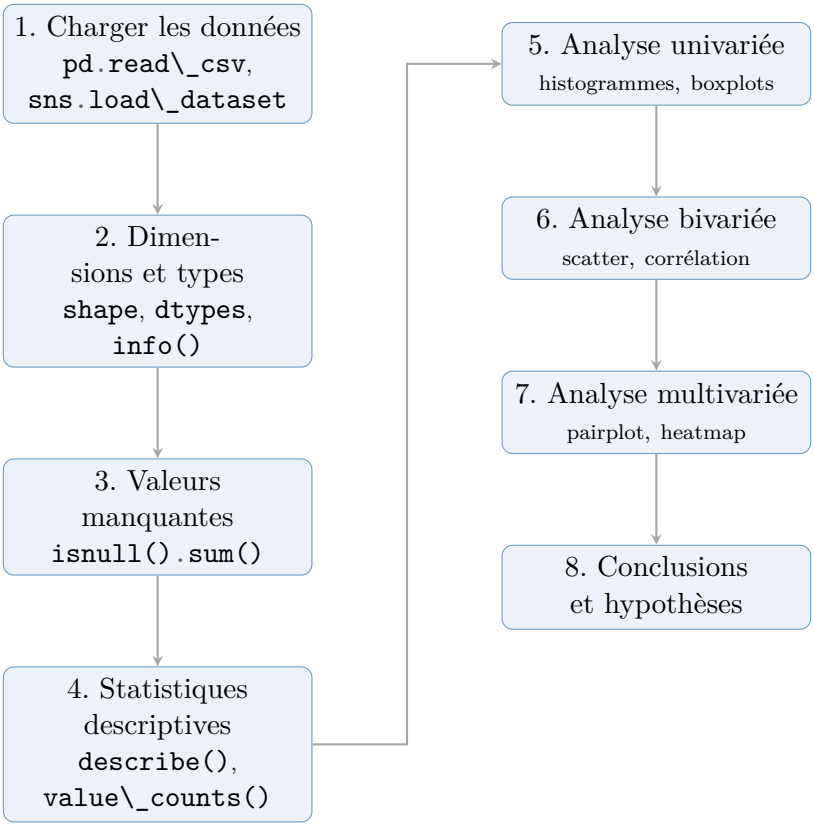
4.1 Qu'est-ce que l'EDA ?

Définition 4.1 (Analyse exploratoire des données)**.** L'EDA est une approche d'analyse qui utilise des techniques visuelles et quantitatives pour comprendre la structure, les anomalies et les relations dans un jeu de données *avant* toute modélisation formelle.

Les objectifs principaux de l'EDA sont :

- Découvrir la **structure** des données (dimensions, types, valeurs manquantes).
- Identifier les **distributions** et les **tendances centrales**.
- Repérer les **anomalies** et les **valeurs aberrantes**.
- Explorer les **relations** entre variables.
- Formuler des **hypothèses** à tester ultérieurement.

4.2 Processus systématique d'EDA



4.3 Étude de cas : le jeu Titanic

4.3.1 Chargement et première inspection

Python

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

titanic = sns.load_dataset('titanic')
print(f"Dimensions : {titanic.shape}")
print(titanic.dtypes)
```

Sortie

```
Dimensions : (891, 15)
survived       int64
pclass         int64
sex           object
age          float64
sibsp          int64
```

```
parch            int64
fare           float64
embarked        object
class         category
who             object
adult_male        bool
deck          category
embark_town     object
alive           object
alone             bool
```

4.3.2 Valeurs manquantes

Python

```python
missing = titanic.isnull().sum()
missing_pct = (missing / len(titanic) * 100).round(1)
missing_df = pd.DataFrame({'Manquantes': missing,
                           'Pourcentage': missing_pct})
print(missing_df[missing_df['Manquantes'] > 0])
```

Sortie

```
             Manquantes  Pourcentage
age                 177         19.9
embarked              2          0.2
deck                688         77.2
embark_town           2          0.2
```

Remarque 4.2. La variable `deck` a 77 % de valeurs manquantes : elle sera probablement inutilisable sans imputation lourde. La variable `age` a environ 20 % de manquants, ce qui est gérable.

4.3.3 Statistiques descriptives

Python

```python
print(titanic.describe())
print("\n--- Variables catégorielles ---")
print(titanic[['sex', 'embarked', 'class']].describe())
```

Sortie

```
         survived      pclass         age       sibsp       parch        fare
count  891.000000  891.000000  714.00000  891.00000  891.00000  891.00000
mean     0.383838    2.308642   29.69912    0.52301    0.38159   32.20421
```

```
std       0.486592    0.836071   14.52650    1.10274    0.80606   49.69343
min       0.000000    1.000000    0.42000    0.00000    0.00000    0.00000
25%       0.000000    2.000000   20.12500    0.00000    0.00000    7.91040
50%       0.000000    3.000000   28.00000    0.00000    0.00000   14.45420
75%       1.000000    3.000000   38.00000    1.00000    0.00000   31.00000
max       1.000000    3.000000   80.00000    8.00000    6.00000  512.32920

--- Variables catégorielles ---
           sex embarked class
count      891      889   891
unique       2        3     3
top       male        S Third
freq       577      644   491
```

4.4 Analyse univariée

Python

```python
fig, axes = plt.subplots(2, 2, figsize=(12, 9))

sns.histplot(titanic['age'].dropna(), bins=30, kde=True,
             ax=axes[0,0], color='steelblue')
axes[0,0].set_title('Distribution de l\'\\^age')

sns.countplot(data=titanic, x='pclass', ax=axes[0,1],
              palette='viridis')
axes[0,1].set_title('R\'epartition par classe')

sns.countplot(data=titanic, x='survived', ax=axes[1,0],
              palette='Set2')
axes[1,0].set_xticklabels(['Décédé', 'Survivant'])
axes[1,0].set_title('Survie')

sns.histplot(titanic['fare'], bins=40, ax=axes[1,1],
             color='coral')
axes[1,1].set_title('Distribution du tarif')

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

Sortie

Quatre graphiques : l'âge suit approximativement une distribution normale centrée vers 29 ans ; la troisième classe est la plus représentée (491) ; 62 % des passagers n'ont pas survécu ; le tarif est très asymétrique avec une longue queue droite.

4.5 Analyse bivariée

4.5.1 Survie par sexe et par classe

Python

```python
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Survie par sexe
sns.barplot(data=titanic, x='sex', y='survived',
            ax=axes[0], palette='coolwarm', ci=95)
axes[0].set_title('Taux de survie par sexe')
axes[0].set_ylabel('Taux de survie')

# Survie par classe
sns.barplot(data=titanic, x='pclass', y='survived',
            ax=axes[1], palette='viridis', ci=95)
axes[1].set_title('Taux de survie par classe')

# Survie par sexe et classe
sns.barplot(data=titanic, x='pclass', y='survived',
            hue='sex', ax=axes[2], palette='Set1', ci=95)
axes[2].set_title('Survie par classe et sexe')

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

Sortie

Les femmes ont un taux de survie de 74 % contre 19 % pour les hommes. La 1ère classe a 63 % de survie, la 2ème 47 % et la 3ème 24 %. La combinaison montre que les femmes de 1ère classe ont survécu à près de 97 %.

4.5.2 Corrélations et table de contingence

Python

```python
# Table de contingence
ct = pd.crosstab(titanic['sex'], titanic['survived'],
                 margins=True)
ct.columns = ['Décédé', 'Survivant', 'Total']
print(ct)

# Corrélation numérique
cols_num = ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
corr = titanic[cols_num].corr()
print("\nMatrice de corrélation :")
print(corr.round(2))
```

Sortie

```
         Décédé  Survivant  Total
sex
female       81        233    314
male        468        109    577
Total       549        342    891

Matrice de corrélation :
          survived  pclass    age  sibsp  parch   fare
survived      1.00   -0.34  -0.08  -0.04   0.08   0.26
pclass       -0.34    1.00  -0.37   0.08   0.02  -0.55
age          -0.08   -0.37   1.00  -0.31  -0.19   0.10
fare          0.26   -0.55   0.10  -0.24   0.22   1.00
```

4.6 Analyse multivariée

Python

```python
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(corr, annot=True, fmt='.2f', cmap='RdBu_r',
            center=0, ax=ax, square=True,
            linewidths=0.5)
ax.set_title('Heatmap des corr\'elations -- Titanic')
plt.tight_layout()
plt.show()
```

Sortie

Heatmap montrant les corrélations : la survie est négativement corrélée avec la classe (-0.34) et positivement avec le tarif (0.26). La classe et le tarif sont fortement anti-corrélés (-0.55).

4.7 Détection des valeurs aberrantes

Définition 4.3 (Méthode IQR)**.** Une observation est considérée comme aberrante si elle se trouve en dehors de l'intervalle :

$$\left[Q_1 - 1.5 \times \text{IQR},\ Q_3 + 1.5 \times \text{IQR}\right]$$

où $\text{IQR} = Q_3 - Q_1$ est l'écart interquartile.

Python

```python
def detecter_outliers_iqr(df, colonne):
    Q1 = df[colonne].quantile(0.25)
    Q3 = df[colonne].quantile(0.75)
```

```
    IQR = Q3 - Q1
    borne_inf = Q1 - 1.5 * IQR
    borne_sup = Q3 + 1.5 * IQR
    outliers = df[(df[colonne] < borne_inf) |
                  (df[colonne] > borne_sup)]
    return outliers, borne_inf, borne_sup

outliers_fare, b_inf, b_sup = detecter_outliers_iqr(titanic, 'fare')
print(f"Bornes : [{b_inf:.2f}, {b_sup:.2f}]")
print(f"Nombre d'outliers (fare) : {len(outliers_fare)}")
print(f"Tarif max : {titanic['fare'].max():.2f}")
```

Sortie

```
Bornes : [-26.72, 65.63]
Nombre d'outliers (fare) : 116
Tarif max : 512.33
```

Remarque 4.4. 116 passagers (13 %) ont un tarif considéré comme aberrant par la méthode IQR. Ce sont principalement des passagers de première classe. Il ne faut pas les supprimer automatiquement : ce sont des valeurs légitimes qui reflètent la structure des données.

4.8 Exercices

Exercice 4.1 (⋆)**.** Chargez le jeu `titanic` et affichez les 5 premières lignes, les dimensions, les types de colonnes et le nombre de valeurs manquantes par colonne.

Exercice 4.2 (⋆)**.** Calculez le taux de survie global, puis le taux de survie par port d'embarquement (`embarked`). Quelle est la lettre du port avec le meilleur taux de survie ?

Exercice 4.3 (⋆⋆)**.** Créez un graphique avec `sns.catplot` de type `'bar'` montrant le taux de survie par classe et par sexe. Ajoutez un titre informatif. Interprétez les résultats en termes de priorité d'évacuation.

Exercice 4.4 (⋆⋆)**.** Créez un histogramme de l'âge des passagers, séparé par survie (`hue='survived'`). Les survivants sont-ils en moyenne plus jeunes ? Vérifiez avec `groupby('survived')['a`

Exercice 4.5 (⋆⋆⋆)**.** Appliquez la méthode IQR pour détecter les outliers dans la variable `age`. Combien sont-ils ? Créez un boxplot annoté montrant les bornes IQR et les outliers. Discutez de la pertinence de supprimer ces observations.

Exercice 4.6 (⋆⋆⋆)**.** Réalisez une EDA complète du jeu `penguins` (`sns.load\_dataset('penguins')`). Votre analyse doit inclure : (a) inspection générale, (b) traitement des valeurs manquantes, (c) distributions univariées, (d) analyse bivariée par espèce, (e) heatmap de corrélation. Concluez en formulant 3 hypothèses testables.

Fonctions clés

Résumé du chapitre – Analyse exploratoire des données

- **Inspection** : `df.shape`, `df.dtypes`, `df.info()`, `df.head()`.
- **Manquants** : `df.isnull().sum()`, `df.isnull().mean() * 100`.
- **Descriptif** : `df.describe()`, `df['col'].value\_counts()`.
- **Univarié** : `sns.histplot()`, `sns.countplot()`, `sns.boxplot()`.
- **Bivarié** : `sns.barplot(x, y)`, `sns.scatterplot()`, `pd.crosstab()`.
- **Multivarié** : `df.corr()`, `sns.heatmap()`, `sns.pairplot()`.
- **Outliers (IQR)** : aberrant si $x < Q_1 - 1.5 \cdot \text{IQR}$ ou $x > Q_3 + 1.5 \cdot \text{IQR}$.
- **Philosophie Tukey** : explorer d'abord, modéliser ensuite.

Chapitre 5

Nettoyage et Préparation des Données

Intuition

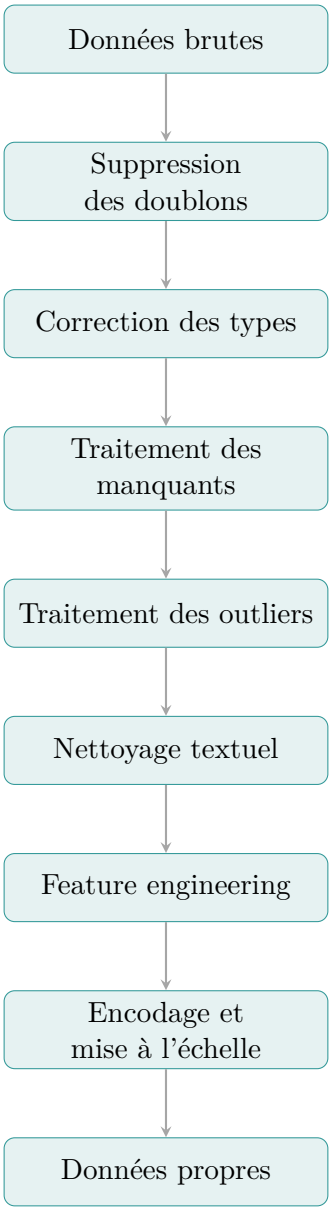
On estime que les data scientists consacrent environ 80 % de leur temps au nettoyage et à la préparation des données. Des données mal préparées mènent inévitablement à des modèles erronés : « Garbage in, garbage out. »

5.1 Pourquoi nettoyer les données ?

Les données réelles sont rarement propres. On rencontre fréquemment :

- des **valeurs manquantes** (capteurs défaillants, champs non remplis) ;
- des **doublons** (saisies multiples, fusions de tables) ;
- des **types incorrects** (nombres stockés comme texte) ;
- des **valeurs aberrantes** (erreurs de saisie) ;
- des **incohérences textuelles** (majuscules, espaces, abréviations).

5.2 Pipeline de nettoyage



5.3 Étude de cas : le jeu Titanic

Python

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

titanic = sns.load_dataset('titanic')
print(f"Dimensions initiales : {titanic.shape}")
print(titanic.info())
```

Sortie

```
Dimensions initiales : (891, 15)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 ...
 7   embarked     889 non-null    object
 11  deck         203 non-null    category
```

5.4 Valeurs manquantes

5.4.1 Types de données manquantes

Définition 5.1 (Classification des valeurs manquantes).

- **MCAR** (*Missing Completely At Random*) : le caractère manquant ne dépend d'aucune variable.
- **MAR** (*Missing At Random*) : le caractère manquant dépend d'autres variables observées.
- **MNAR** (*Missing Not At Random*) : le caractère manquant dépend de la valeur elle-même.

Exemple 5.2. Dans le Titanic, les cabines (`deck`) manquent surtout pour les passagers de 3ème classe (MAR). L'âge pourrait être MCAR ou MAR selon que les passagers sans billet complet n'avaient pas d'âge enregistré.

5.4.2 Détection et traitement

Python

```python
# Détection
print(titanic.isnull().sum().sort_values(ascending=False).head(5))

# Stratégie 1 : supprimer la colonne deck (trop de manquants)
titanic_clean = titanic.drop(columns=['deck'])

# Stratégie 2 : remplir age par la médiane par classe
titanic_clean['age'] = titanic_clean.groupby('pclass')['age'] \
    .transform(lambda x: x.fillna(x.median()))

# Stratégie 3 : remplir embarked par le mode
titanic_clean['embarked'].fillna(
    titanic_clean['embarked'].mode()[0], inplace=True)
titanic_clean['embark_town'].fillna(
```

```
    titanic_clean['embark_town'].mode()[0], inplace=True)

print(f"\nManquants restants : {titanic_clean.isnull().sum().sum()}")
```

Sortie

```
deck           688
age            177
embarked         2
embark_town      2
survived         0

Manquants restants : 0
```

Bonne pratique

L'imputation par la médiane *conditionnée* (ici par classe) est préférable à la médiane globale car elle préserve la structure des sous-groupes. Pour des données temporelles, utilisez `ffill()` ou `bfill()`.

5.5 Doublons

Python

```
# Détection
n_dup = titanic_clean.duplicated().sum()
print(f"Nombre de doublons : {n_dup}")

# Inspection des doublons éventuels
if n_dup > 0:
    print(titanic_clean[titanic_clean.duplicated(keep=False)]
          .sort_values(by='name').head())
    titanic_clean = titanic_clean.drop_duplicates()
    print(f"Après suppression : {titanic_clean.shape}")
```

Sortie

```
Nombre de doublons : 0
```

5.6 Conversion de types

Python

```python
# Convertir survived en booléen
titanic_clean['survived'] = titanic_clean['survived'].astype(bool)

# Convertir pclass en catégoriel ordonné
titanic_clean['pclass'] = pd.Categorical(
    titanic_clean['pclass'], categories=[1, 2, 3], ordered=True)

# Exemple de conversion de date (sur données fictives)
dates_str = pd.Series(['2023-01-15', '2023-02-20', '2023-03-10'])
dates = pd.to_datetime(dates_str)
print(dates.dt.month)
```

Sortie

```
0    1
1    2
2    3
dtype: int32
```

5.7 Traitement des valeurs aberrantes

Python

```python
# Méthode IQR pour le tarif
Q1 = titanic_clean['fare'].quantile(0.25)
Q3 = titanic_clean['fare'].quantile(0.75)
IQR = Q3 - Q1

borne_inf = Q1 - 1.5 * IQR
borne_sup = Q3 + 1.5 * IQR

# Option 1 : écrêtage (capping/winsorizing)
titanic_clean['fare_capped'] = titanic_clean['fare'].clip(
    lower=borne_inf, upper=borne_sup)

print(f"Avant : max = {titanic_clean['fare'].max():.2f}")
print(f"Après écrêtage : max = {titanic_clean['fare_capped'].max():.2f}")
print(f"Borne supérieure : {borne_sup:.2f}")
```

Sortie

```
Avant : max = 512.33
Après écrêtage : max = 65.63
```

```
Borne supérieure : 65.63
```

Attention

Ne supprimez jamais les outliers sans réflexion. Un tarif de 512 $ peut être légitime (suite de luxe). L'écrêtage est souvent préférable à la suppression. Documentez toujours vos choix.

5.8 Nettoyage textuel

Python

```python
# Exemple avec des données textuelles
noms = pd.Series(['  Alice ', 'BOB', 'charlie', ' Dave  '])
print("Avant :", noms.tolist())

noms_clean = (noms.str.strip()
                  .str.lower()
                  .str.capitalize())
print("Après :", noms_clean.tolist())

# Remplacement de motifs
villes = pd.Series(['New-York', 'new york', 'NEW YORK', 'N.Y.'])
villes_norm = villes.str.lower().str.replace(
    r'[^a-z\s]', '', regex=True).str.strip()
print("Villes normalisées :", villes_norm.tolist())
```

Sortie

```
Avant : ['  Alice ', 'BOB', 'charlie', ' Dave  ']
Après : ['Alice', 'Bob', 'Charlie', 'Dave']
Villes normalisées : ['newyork', 'new york', 'new york', 'ny']
```

5.9 Feature engineering

Python

```python
# Créer de nouvelles variables à partir des existantes
titanic_clean['family_size'] = (titanic_clean['sibsp']
                                + titanic_clean['parch'] + 1)
titanic_clean['is_child'] = titanic_clean['age'] < 18
titanic_clean['fare_per_person'] = (titanic_clean['fare']
                                    / titanic_clean['family_size'])

# Discrétiser l'âge en catégories
```

```
titanic_clean['age_group'] = pd.cut(
    titanic_clean['age'],
    bins=[0, 12, 18, 35, 60, 100],
    labels=['Enfant', 'Ado', 'Jeune adulte', 'Adulte', 'Senior'])

print(titanic_clean['age_group'].value_counts())
```

Sortie

```
Jeune adulte    371
Adulte          267
Enfant           89
Ado              79
Senior           85
Name: age_group, dtype: int64
```

5.10 Encodage des variables catégorielles

Python

```
# One-hot encoding avec pandas
embarked_dummies = pd.get_dummies(
    titanic_clean['embarked'], prefix='embarked', dtype=int)
print(embarked_dummies.head())

# Label encoding avec sklearn
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
titanic_clean['sex_encoded'] = le.fit_transform(
    titanic_clean['sex'])
print("\nMapping :", dict(zip(le.classes_,
                            le.transform(le.classes_))))
```

Sortie

```
   embarked_C  embarked_Q  embarked_S
0           0           0           1
1           1           0           0
2           0           0           1
3           0           0           1
4           0           0           1

Mapping : {'female': 0, 'male': 1}
```

Attention

Le **label encoding** impose un ordre artificiel aux catégories. Il ne convient que pour les variables ordinales ou les algorithmes à base d'arbres. Pour les modèles linéaires, préférez le **one-hot encoding**.

5.11 Mise à l'échelle

Définition 5.3 (Standardisation et normalisation).

- **Standardisation** (StandardScaler) : $z = \dfrac{x - \mu}{\sigma}$, résultat de moyenne 0 et d'écart-type 1.
- **Normalisation** (MinMaxScaler) : $x' = \dfrac{x - x_{\min}}{x_{\max} - x_{\min}}$, résultat dans $[0, 1]$.

Python

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler

cols = ['age', 'fare']
data_num = titanic_clean[cols].copy()

# Standardisation
scaler_std = StandardScaler()
data_std = pd.DataFrame(scaler_std.fit_transform(data_num),
                        columns=[c + '_std' for c in cols])

# Normalisation
scaler_mm = MinMaxScaler()
data_norm = pd.DataFrame(scaler_mm.fit_transform(data_num),
                         columns=[c + '_norm' for c in cols])

print("Standardisé :")
print(data_std.describe().loc[['mean', 'std']].round(2))
print("\nNormalisé :")
print(data_norm.describe().loc[['min', 'max']].round(2))
```

Sortie

```
Standardisé :
      age_std  fare_std
mean     0.00      0.00
std      1.00      1.00

Normalisé :
      age_norm  fare_norm
min       0.00       0.00
max       1.00       1.00
```

Bonne pratique

Appliquez la mise à l'échelle *après* la séparation train/test pour éviter la fuite d'information (*data leakage*). Utilisez `fit\_transform()` sur le train et `transform()` sur le test.

5.12 Exercices

Exercice 5.1 (⋆)**.** Chargez le jeu `titanic`. Affichez le nombre et le pourcentage de valeurs manquantes par colonne. Quelles colonnes ont plus de 50 % de manquants ?

Exercice 5.2 (⋆)**.** Supprimez les doublons du jeu `titanic` (s'il y en a). Puis convertissez la colonne `sex` en variable catégorielle avec `pd.Categorical()`.

Exercice 5.3 (⋆⋆)**.** Imputez les valeurs manquantes de `age` par la médiane conditionnée par `sex` et `pclass` (6 groupes). Comparez la distribution de l'âge avant et après imputation avec des histogrammes superposés.

Exercice 5.4 (⋆⋆)**.** Créez les variables suivantes : `title` (extrait du nom avec une expression régulière), `is\_alone` (vrai si `family\_size` = 1), `fare\_bin` (quartiles du tarif avec `pd.qcut`). Calculez le taux de survie pour chacune de ces nouvelles variables.

Exercice 5.5 (⋆⋆⋆)**.** Construisez un pipeline complet de nettoyage pour le jeu `titanic` : (a) suppression de `deck`, (b) imputation de `age` et `embarked`, (c) création de `family\_size` et `is\_child`, (d) one-hot encoding de `sex`, `embarked`, `class`, (e) standardisation des variables numériques. Le résultat final doit être un DataFrame entièrement numérique sans valeurs manquantes.

Exercice 5.6 (⋆ ⋆ ⋆)**.** Chargez le jeu `penguins` (`sns.load\_dataset('penguins')`). Ce jeu contient des valeurs manquantes. Appliquez trois stratégies différentes (suppression, imputation par médiane, imputation par médiane conditionnée par espèce). Comparez les statistiques descriptives obtenues avec chaque stratégie et discutez de la meilleure approche.

Fonctions clés

Résumé du chapitre – Nettoyage et préparation

- **Manquants** : `isnull().sum()`, types MCAR/MAR/MNAR, `fillna()`, `dropna()`.
- **Doublons** : `duplicated()`, `drop\_duplicates()`.
- **Types** : `astype()`, `pd.to\_datetime()`, `pd.Categorical()`.
- **Outliers** : méthode IQR, écrêtage avec `clip()`.
- **Texte** : `str.strip()`, `str.lower()`, `str.replace()`.
- **Feature engineering** : `pd.cut()`, `pd.qcut()`, combinaison de colonnes.
- **Encodage** : `pd.get\_dummies()` (one-hot), `LabelEncoder` (ordinal).

- **Mise à l'échelle** : $z = \frac{x-\mu}{\sigma}$ (StandardScaler), $x' = \frac{x-x_{\min}}{x_{\max}-x_{\min}}$ (MinMaxScaler).
- **Règle d'or** : documenter chaque transformation et éviter le *data leakage*.

Chapitre 6

Statistiques Appliquées

Intuition

Les statistiques fournissent le cadre mathématique qui permet de passer de l'observation à l'inférence. En science des données, elles servent à quantifier l'incertitude, tester des hypothèses et mesurer les relations entre variables.

6.1 Statistiques descriptives

6.1.1 Mesures de tendance centrale et de dispersion

Définition 6.1 (Mesures fondamentales)**.** Pour un échantillon $(x_1, x_2, \ldots, x_n)$:

- **Moyenne** : $\bar{x} = \dfrac{1}{n}\sum_{i=1}^{n} x_i$
- **Médiane** : valeur telle que 50 % des observations sont inférieures.
- **Mode** : valeur la plus fréquente.
- **Variance** : $s^2 = \dfrac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
- **Écart-type** : $s = \sqrt{s^2}$
- **Quantiles** : Q_1 (25 %), Q_2 (50 %), Q_3 (75 %).

Python

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

tips = sns.load_dataset('tips')

col = tips['total_bill']
print(f"Moyenne     : {col.mean():.2f}")
```

```python
print(f"Médiane      : {col.median():.2f}")
print(f"Mode         : {col.mode()[0]:.2f}")
print(f"Variance     : {col.var():.2f}")
print(f"Écart-type   : {col.std():.2f}")
print(f"Q1, Q3       : {col.quantile(0.25):.2f}, "
      f"{col.quantile(0.75):.2f}")
print(f"Asymétrie    : {col.skew():.2f}")
print(f"Kurtosis     : {col.kurtosis():.2f}")
```

Sortie

```
Moyenne      : 19.79
Médiane      : 17.80
Mode         : 13.42
Variance     : 79.25
Écart-type   : 8.90
Q1, Q3       : 13.35, 24.13
Asymétrie    : 1.13
Kurtosis     : 1.22
```

Remarque 6.2. L'asymétrie positive (1.13) et la moyenne supérieure à la médiane confirment une distribution étirée vers la droite. Le kurtosis positif indique des queues plus lourdes que la loi normale.

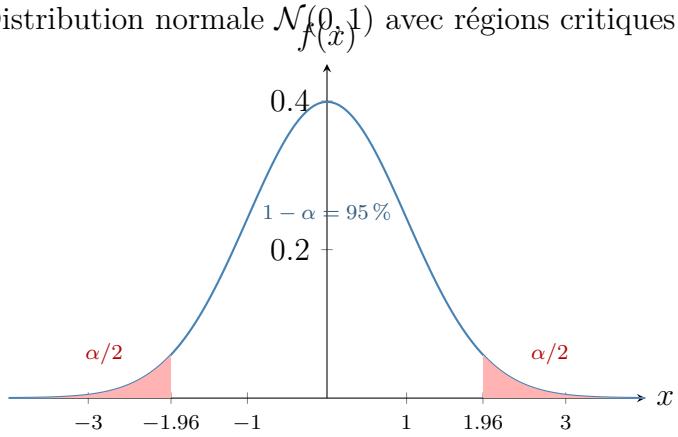
6.2 Distributions de probabilité

6.2.1 Loi normale

Définition 6.3 (Loi normale)**.** Une variable aléatoire X suit une loi normale $\mathcal{N}(\mu, \sigma^2)$ si sa densité est :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Distribution normale $\mathcal{N}(0,1)$ avec régions critiques



Python

```python
from scipy.stats import norm
import matplotlib.pyplot as plt

# Loi normale : probabilités et quantiles
print(f"P(X < 1.96) = {norm.cdf(1.96):.4f}")
print(f"P(-1.96 < X < 1.96) = {norm.cdf(1.96) - norm.cdf(-1.96):.4f}")
print(f"Quantile 97.5% = {norm.ppf(0.975):.4f}")

# Visualisation
x = np.linspace(-4, 4, 200)
fig, ax = plt.subplots(figsize=(7, 4))
ax.plot(x, norm.pdf(x), 'steelblue', lw=2, label='$\mathcal{N}(0,1)$')
ax.fill_between(x, norm.pdf(x), where=(x > 1.96) | (x < -1.96),
                color='red', alpha=0.3, label='Régions critiques')
ax.legend()
ax.set_title('Distribution normale standard')
plt.show()
```

Sortie

```
P(X < 1.96) = 0.9750
P(-1.96 < X < 1.96) = 0.9500
Quantile 97.5% = 1.9600
```

6.2.2 Lois binomiale et de Poisson

Python

```python
from scipy.stats import binom, poisson

# Binomiale : n=20 lancers, p=0.5
print("--- Loi binomiale B(20, 0.5) ---")
print(f"P(X = 10) = {binom.pmf(10, n=20, p=0.5):.4f}")
print(f"P(X <= 10) = {binom.cdf(10, n=20, p=0.5):.4f}")
print(f"Espérance = {binom.mean(n=20, p=0.5):.1f}")
print(f"Variance = {binom.var(n=20, p=0.5):.1f}")

# Poisson : lambda=5 événements par intervalle
print("\n--- Loi de Poisson P(5) ---")
print(f"P(X = 3) = {poisson.pmf(3, mu=5):.4f}")
print(f"P(X <= 3) = {poisson.cdf(3, mu=5):.4f}")
```

Sortie

```
--- Loi binomiale B(20, 0.5) ---
P(X = 10) = 0.1762
```

```
P(X <= 10) = 0.5881
Espérance = 10.0
Variance = 5.0

--- Loi de Poisson P(5) ---
P(X = 3) = 0.1404
P(X <= 3) = 0.2650
```

6.3 Intervalles de confiance

Définition 6.4 (Intervalle de confiance)**.** Un intervalle de confiance à $(1-\alpha)\times 100\,\%$ pour la moyenne μ est :

$$\text{IC} = \left[\bar{x} - z_{\alpha/2}\,\frac{s}{\sqrt{n}},\ \bar{x} + z_{\alpha/2}\,\frac{s}{\sqrt{n}}\right]$$

où $z_{\alpha/2}$ est le quantile de la loi normale (1.96 pour 95 %).

Python

```python
# IC à 95% pour la moyenne du pourboire
tip = tips['tip']
n = len(tip)
mean = tip.mean()
se = tip.std() / np.sqrt(n)

ci_low = mean - 1.96 * se
ci_high = mean + 1.96 * se
print(f"Moyenne : {mean:.3f}")
print(f"IC 95% : [{ci_low:.3f}, {ci_high:.3f}]")

# Avec scipy (méthode exacte, distribution t)
ci = stats.t.interval(confidence=0.95, df=n-1,
                      loc=mean, scale=se)
print(f"IC 95% (t-Student) : [{ci[0]:.3f}, {ci[1]:.3f}]")
```

Sortie

```
Moyenne : 2.998
IC 95% : [2.823, 3.173]
IC 95% (t-Student) : [2.822, 3.175]
```

6.4 Tests d'hypothèses

Définition 6.5 (Test d'hypothèse)**.** Un test d'hypothèse oppose :

- H_0 (hypothèse nulle) : pas d'effet, pas de différence.

- H_1 (hypothèse alternative) : il existe un effet ou une différence.

La **p-valeur** est la probabilité d'observer un résultat au moins aussi extrême que celui observé, sous H_0. On rejette H_0 si $p < \alpha$ (généralement $\alpha = 0.05$).

6.4.1 Test t de Student

Python

```python
# Les hommes donnent-ils des pourboires différents des femmes ?
tip_male = tips[tips['sex'] == 'Male']['tip']
tip_female = tips[tips['sex'] == 'Female']['tip']

t_stat, p_value = stats.ttest_ind(tip_male, tip_female)
print(f"Statistique t : {t_stat:.4f}")
print(f"p-valeur : {p_value:.4f}")
print(f"Moyenne hommes : {tip_male.mean():.3f}")
print(f"Moyenne femmes : {tip_female.mean():.3f}")

if p_value < 0.05:
    print("=> Différence significative (rejet de H0)")
else:
    print("=> Pas de différence significative")
```

Sortie

```
Statistique t : 1.3879
p-valeur : 0.1665
Moyenne hommes : 3.090
Moyenne femmes : 2.833
=> Pas de différence significative
```

6.4.2 Test du χ^2 d'indépendance

Python

```python
# Le statut fumeur est-il lié au moment du repas ?
titanic = sns.load_dataset('titanic')

# Test sur le Titanic : survie vs sexe
ct = pd.crosstab(titanic['sex'], titanic['survived'])
print("Table de contingence :")
print(ct)

chi2, p_value, dof, expected = stats.chi2_contingency(ct)
print(f"\nChi² = {chi2:.2f}")
print(f"p-valeur = {p_value:.2e}")
print(f"Degrés de liberté = {dof}")
```

```
print(f"=> {'Dépendance significative' if p_value < 0.05 else
↪  'Indépendance'}")
```

Sortie

```
Table de contingence :
survived    0    1
sex
female     81  233
male      468  109

Chi² = 260.72
p-valeur = 1.20e-58
Degrés de liberté = 1
=> Dépendance significative
```

Remarque 6.6. Le test du χ^2 confirme avec une énorme significativité statistique ($p \approx 10^{-58}$) que la survie dépendait fortement du sexe. C'est le résultat le plus marquant du Titanic : « Women and children first. »

6.5 Corrélations

Définition 6.7 (Coefficients de corrélation).

- **Pearson** : mesure la corrélation *linéaire* : $r = \dfrac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \cdot \sum(y_i - \bar{y})^2}}$
- **Spearman** : mesure la corrélation *monotone* (basée sur les rangs).

Les deux coefficients sont dans $[-1, 1]$. $|r| > 0.7$ indique une corrélation forte.

Python

```
# Corrélation entre addition et pourboire
r_pearson, p_pearson = stats.pearsonr(tips['total_bill'],
                                      tips['tip'])
r_spearman, p_spearman = stats.spearmanr(tips['total_bill'],
                                         tips['tip'])

print(f"Pearson  : r = {r_pearson:.4f}, p = {p_pearson:.2e}")
print(f"Spearman : r = {r_spearman:.4f}, p = {p_spearman:.2e}")
```

Sortie

```
Pearson  : r = 0.6757, p = 6.69e-34
Spearman : r = 0.6787, p = 2.83e-34
```

Attention

Corrélation n'implique pas causalité. Une forte corrélation entre deux variables peut être due à une variable confondante. Toujours compléter par une analyse théorique et, si possible, un plan expérimental.

6.6 ANOVA à un facteur

Définition 6.8 (ANOVA)**.** L'analyse de la variance (ANOVA) teste si les moyennes de k groupes sont égales :

- $H_0 : \mu_1 = \mu_2 = \cdots = \mu_k$
- H_1 : au moins deux moyennes diffèrent.

La statistique F compare la variance inter-groupes à la variance intra-groupes.

Python

```python
# Le pourboire diffère-t-il selon le jour ?
jours = tips['day'].unique()
groupes = [tips[tips['day'] == j]['tip'] for j in jours]

f_stat, p_value = stats.f_oneway(*groupes)
print(f"Statistique F : {f_stat:.4f}")
print(f"p-valeur : {p_value:.4f}")

# Moyennes par jour
print("\nMoyenne du pourboire par jour :")
print(tips.groupby('day')['tip'].agg(['mean', 'std', 'count'])
      .round(3))
```

Sortie

```
Statistique F : 1.6724
p-valeur : 0.1736

Moyenne du pourboire par jour :
       mean    std  count
day
Thur  2.771  1.240     62
Fri   2.735  1.018     19
Sat   2.993  1.631     87
Sun   3.256  1.227     76
```

Remarque 6.9. L'ANOVA ne détecte pas de différence significative ($p = 0.17$). Bien que le dimanche ait la moyenne la plus élevée, la variabilité intra-groupe est trop importante pour conclure.

6.7 Exemple intégrateur : analyse du Titanic

Python

```python
# 1. Le tarif diffère-t-il selon la classe ? (ANOVA)
groupes_fare = [titanic[titanic['pclass'] == c]['fare']
                for c in [1, 2, 3]]
f, p = stats.f_oneway(*groupes_fare)
print(f"ANOVA tarif ~ classe : F={f:.2f}, p={p:.2e}")

# 2. Survie vs classe ? (Chi²)
ct2 = pd.crosstab(titanic['pclass'], titanic['survived'])
chi2, p2, _, _ = stats.chi2_contingency(ct2)
print(f"Chi² survie ~ classe : ²={chi2:.2f}, p={p2:.2e}")

# 3. Corrélation âge-tarif
age_clean = titanic.dropna(subset=['age'])
r, p3 = stats.pearsonr(age_clean['age'], age_clean['fare'])
print(f"Pearson âge ~ tarif : r={r:.3f}, p={p3:.4f}")
```

Sortie

```
ANOVA tarif ~ classe : F=242.34, p=2.00e-84
Chi² survie ~ classe : ²=102.89, p=4.55e-23
Pearson âge ~ tarif : r=0.096, p=0.0101
```

Proposition 6.10. *Les résultats montrent que (1) le tarif diffère très significativement entre classes, (2) la survie dépend fortement de la classe, et (3) l'âge et le tarif ont une corrélation très faible bien que significative (l'effectif élevé rend de petits effets significatifs).*

6.8 Exercices

Exercice 6.1 ($\star$)**.** Calculez la moyenne, la médiane, l'écart-type et les quartiles de la variable `tip` du jeu `tips`. La distribution est-elle symétrique ? Justifiez avec le coefficient d'asymétrie.

Exercice 6.2 ($\star$)**.** Générez 1000 échantillons de taille 30 à partir d'une loi $\mathcal{N}(50, 10^2)$. Pour chaque échantillon, calculez l'IC à 95 %. Combien d'intervalles contiennent effectivement $\mu = 50$? Le résultat est-il conforme à la théorie ?

Exercice 6.3 ($\star\star$)**.** Testez avec un test t si le pourboire moyen diffère entre les repas du midi (`time='Lunch'`) et du soir (`time='Dinner'`). Interprétez la p-valeur et calculez la taille d'effet (différence des moyennes divisée par l'écart-type poolé).

Exercice 6.4 ($\star\star$)**.** Réalisez un test du χ^2 pour tester l'indépendance entre `smoker` et `time` dans le jeu `tips`. Affichez la table de contingence, les effectifs théoriques et concluez.

Exercice 6.5 ($\star\star\star$)**.** Sur le jeu `titanic`, réalisez les tests suivants et synthétisez les résultats dans un tableau : (a) t-test sur l'âge entre survivants et non-survivants, (b) χ^2 sur

la survie vs le port d'embarquement, (c) ANOVA sur le tarif par port d'embarquement, (d) corrélation de Spearman entre `pclass` et `survived`. Appliquez la correction de Bonferroni ($\alpha_{\text{corrigé}} = 0.05/4 = 0.0125$) et concluez.

Exercice 6.6 ($\star\star\star$)**.** Simulez la puissance d'un test t : générez deux groupes de taille n à partir de $\mathcal{N}(0,1)$ et $\mathcal{N}(\delta,1)$ avec $\delta \in \{0.2, 0.5, 0.8\}$ et $n \in \{10, 30, 100, 300\}$. Pour chaque combinaison, répétez 1000 fois et calculez le pourcentage de rejets de H_0. Tracez un graphique de la puissance en fonction de n pour chaque δ.

Fonctions clés

Résumé du chapitre – Statistiques appliquées

- **Descriptif** : `mean()`, `median()`, `std()`, `var()`, `quantile()`, `skew()`.
- **Lois** : `norm`, `binom`, `poisson` de `scipy.stats` ; méthodes `.pdf()`, `.cdf()`, `.ppf()`.
- **IC à 95 %** : $\bar{x} \pm 1.96 \cdot \dfrac{s}{\sqrt{n}}$; exact avec `stats.t.interval()`.
- **Test** t : `stats.ttest\_ind(a, b)` – compare deux moyennes indépendantes.
- **Test** χ^2 : `stats.chi2\_contingency(ct)` – indépendance de deux variables catégorielles.
- **Corrélation** : `stats.pearsonr(x, y)`, `stats.spearmanr(x, y)`, $r \in [-1, 1]$.
- **ANOVA** : `stats.f\_oneway(*groupes)` – compare $k \geq 3$ moyennes.
- **Décision** : rejeter H_0 si $p < \alpha$ (souvent $\alpha = 0.05$).
- **Attention** : corrélation $\neq$ causalité ; significativité statistique $\neq$ importance pratique.

Chapitre 7

Introduction à l'Apprentissage Automatique

7.1 Qu'est-ce que l'apprentissage automatique ?

Définition 7.1 (Apprentissage automatique)**.** L'**apprentissage automatique** (*machine learning*, ML) est une branche de l'intelligence artificielle qui permet aux ordinateurs d'apprendre à partir de données sans être explicitement programmés pour chaque tâche.

Intuition

Plutôt que d'écrire des règles manuellement (par exemple, « si le client a plus de 30 ans et un revenu supérieur à 50 000 €, alors approuver le prêt »), on fournit des **exemples** au modèle qui découvre lui-même les règles sous-jacentes.

7.1.1 Types d'apprentissage

Types d'apprentissage automatique

Supervisé	**Non supervisé**	**Par renforcement**
Données étiquetées (classification, régression)	Pas d'étiquettes (clustering, réduction)	Récompenses/punitions (jeux, robotique)
Exemples : prédire un prix, classer un e-mail (spam/non spam)	Exemples : segmenter des clients, réduire les dimensions	Exemples : AlphaGo, conduite autonome

Définition 7.2 (Apprentissage supervisé)**.** On dispose de paires $(\mathbf{x}_i, y_i)$ où $\mathbf{x}_i$ sont les **variables explicatives** (*features*) et y_i est la **cible** (*target*). Le modèle apprend une fonction f telle que $f(\mathbf{x}_i) \approx y_i$.

- **Classification** : y est une catégorie (spam/non spam, espèce de fleur).
- **Régression** : y est une valeur continue (prix, température).

Définition 7.3 (Apprentissage non supervisé)**.** On dispose uniquement de $\mathbf{x}_i$ sans étiquettes. Le modèle cherche des **structures cachées** dans les données (groupes, axes principaux).

7.2 Le flux de travail en apprentissage supervisé

1. **Collecte et préparation** des données.
2. **Découpage** en ensemble d'entraînement et ensemble de test.
3. **Entraînement** du modèle sur les données d'entraînement.
4. **Prédiction** sur les données de test.
5. **Évaluation** des performances.

Python

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Chargement des donn\'ees
iris = load_iris()
X, y = iris.data, iris.target

# D\'ecoupage : 70% entra\^inement, 30% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
print(f"Entra\^inement : {X_train.shape[0]} \'echantillons")
print(f"Test : {X_test.shape[0]} \'echantillons")
```

Sortie

```
Entra\^inement : 105 \'echantillons
Test : 45 \'echantillons
```

7.3 L'API Scikit-learn

Bonne pratique

Tous les modèles scikit-learn suivent la même interface :

1. `modele = Classe(param\^etres)` — créer le modèle.
2. `modele.fit(X_train, y_train)` — entraîner.
3. `modele.predict(X_test)` — prédire.
4. `modele.score(X_test, y_test)` — évaluer.

7.3.1 Exemple complet : k plus proches voisins sur Iris

Python

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Cr\'eation du mod\`ele KNN avec k=5
knn = KNeighborsClassifier(n_neighbors=5)

# Entra\^inement
knn.fit(X_train, y_train)

# Pr\'ediction
y_pred = knn.predict(X_test)

# \'Evaluation
acc = accuracy_score(y_test, y_pred)
print(f"Pr\'edictions : {y_pred[:10]}")
print(f"V\'erit\'e     : {y_test[:10]}")
print(f"Exactitude  : {acc:.4f}")
```

Sortie

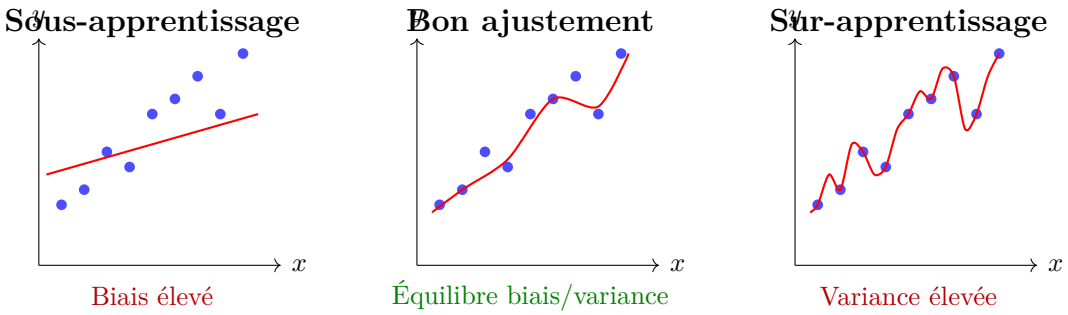
```
Pr\'edictions : [1 0 2 1 1 0 1 2 1 1]
V\'erit\'e     : [1 0 2 1 1 0 1 2 1 1]
Exactitude  : 1.0000
```

Remarque 7.4. La méthode `score()` renvoie l'**exactitude** (*accuracy*) pour la classification et le **coefficient de détermination** R^2 pour la régression.

7.4 Sur-apprentissage et sous-apprentissage

Définition 7.5 (Sur-apprentissage et sous-apprentissage).

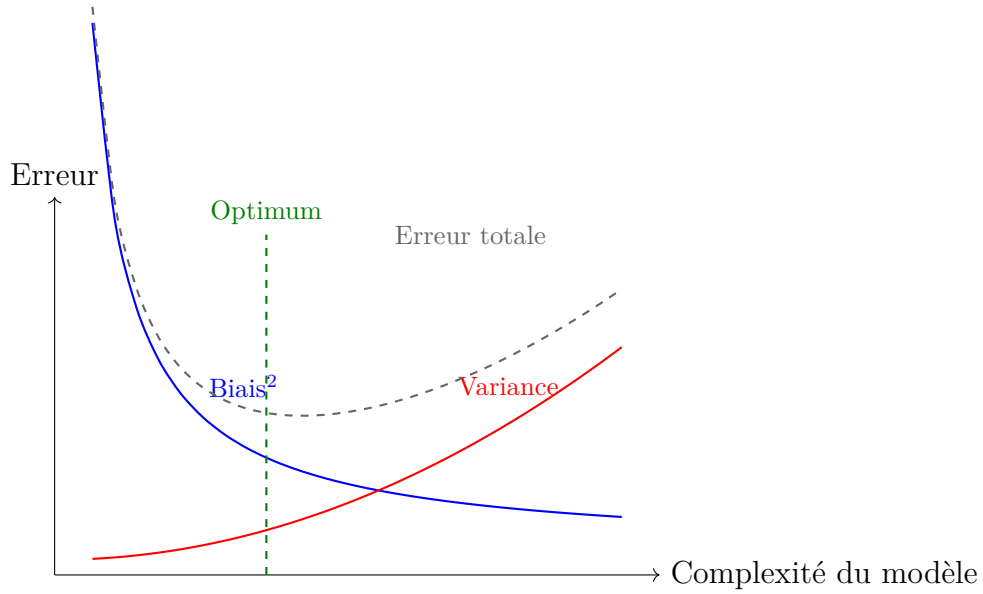
- **Sous-apprentissage** (*underfitting*) : le modèle est trop simple, il ne capture pas la structure des données. **Biais élevé**.
- **Sur-apprentissage** (*overfitting*) : le modèle est trop complexe, il mémorise le bruit des données d'entraînement. **Variance élevée**.



Théorème 7.6 (Compromis biais-variance)**.** *L'erreur de généralisation d'un modèle se décompose en :*

$$\textit{Erreur totale} = \textit{Biais}^2 + \textit{Variance} + \textit{Bruit irréductible}$$

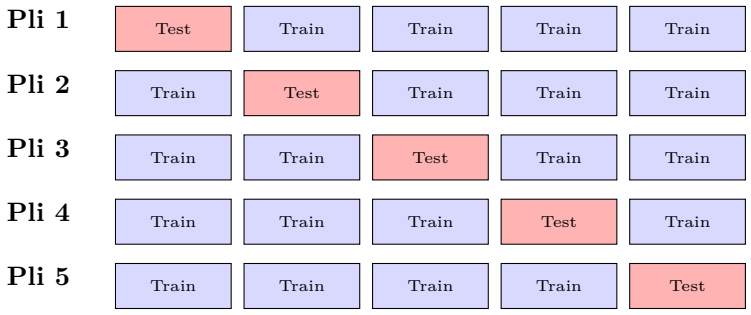
Le but est de trouver le modèle qui minimise cette erreur totale, en équilibrant biais et variance.



7.5 Validation croisée

Définition 7.7 (Validation croisée à k plis)**.** La **validation croisée** (*k-fold cross-validation*) divise les données en k sous-ensembles (*folds*). On entraîne le modèle k fois, chaque fois en utilisant un pli différent comme ensemble de test et les $k-1$ autres comme ensemble d'entraînement.

Validation croisée à 5 plis



Python

```python
from sklearn.model_selection import cross_val_score

knn = KNeighborsClassifier(n_neighbors=5)
```

```python
# Validation crois\'ee \`a 5 plis
scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')

print(f"Scores par pli : {scores}")
print(f"Moyenne : {scores.mean():.4f}")
print(f"\'Ecart-type : {scores.std():.4f}")
```

Sortie

```
Scores par pli : [0.9667 0.9667 0.9333 0.9667 1.0000]
Moyenne : 0.9667
\'Ecart-type : 0.0211
```

Attention

Ne jamais évaluer un modèle sur les données d'entraînement ! Un score parfait sur l'entraînement peut cacher un **sur-apprentissage** sévère. Utilisez toujours un ensemble de test séparé ou la validation croisée.

7.5.1 Choix de k dans KNN

Python

```python
import numpy as np

k_values = range(1, 21)
mean_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=5)
    mean_scores.append(scores.mean())

best_k = k_values[np.argmax(mean_scores)]
print(f"Meilleur k : {best_k}")
print(f"Meilleur score : {max(mean_scores):.4f}")
```

Sortie

```
Meilleur k : 6
Meilleur score : 0.9800
```

7.6 Exercices

Exercice 7.1 ($\star$). Chargez le jeu de données `load_wine()` de scikit-learn. Découpez-le en 80%/20%, entraînez un KNN avec $k = 3$ et affichez l'exactitude sur l'ensemble de test.

Exercice 7.2 ($\star$)**.** Expliquez avec vos propres mots la différence entre sur-apprentissage et sous-apprentissage. Donnez un exemple concret pour chacun.

Exercice 7.3 ($\star\star$)**.** Sur le jeu de données Iris, comparez les performances de KNN pour $k \in \{1, 3, 5, 7, 9, 11\}$ en utilisant la validation croisée à 10 plis. Tracez un graphique des scores moyens en fonction de k.

Exercice 7.4 ($\star\star$)**.** Chargez le jeu `load_digits()` et réalisez une classification avec KNN. Calculez la validation croisée à 5 plis. Que remarquez-vous sur les performances ?

Exercice 7.5 ($\star\star\star$)**.** Chargez le jeu de données `penguins` de seaborn. Préparez les données (supprimez les valeurs manquantes, encodez les variables catégorielles avec `pd.get_dummies`). Entraînez un KNN pour prédire l'espèce et évaluez avec la validation croisée. Comparez les performances avec et sans normalisation des données (`StandardScaler`).

Exercice 7.6 ($\star\star\star$)**.** Créez un jeu de données synthétique avec `make_classification(n_samples=500, n_features=20, n_informative=5)`. Comparez l'exactitude d'entraînement et de test pour un KNN avec $k = 1$ et $k = 10$. Lequel sur-apprend le plus ? Justifiez.

Fonctions clés

Résumé du chapitre 7

- **Apprentissage supervisé** : données étiquetées $(\mathbf{x}_i, y_i)$, classification ou régression.
- **Apprentissage non supervisé** : pas d'étiquettes, découverte de structures.
- **API scikit-learn** : `fit(X_train, y_train)`, `predict(X_test)`, `score(X_test, y_test)`.
- **Découpage** : `train_test_split(X, y, test_size=0.3, random_state=42)`.
- **Sur-apprentissage** : modèle trop complexe, variance élevée.
- **Sous-apprentissage** : modèle trop simple, biais élevé.
- **Compromis biais-variance** : Erreur = Biais^2 + Variance + Bruit.
- **Validation croisée** : `cross_val_score(modele, X, y, cv=k)`, estimation robuste.

Chapitre 8

Modèles de Machine Learning

8.1 Régression linéaire

Définition 8.1 (Régression linéaire)**.** La **régression linéaire** modélise la relation entre une variable cible y et des variables explicatives $\mathbf{x}$:

$$y = \mathbf{X}\boldsymbol{\beta} + \varepsilon$$

où $\boldsymbol{\beta}$ est le vecteur des coefficients et ε est le terme d'erreur.

Théorème 8.2 (Moindres carrés ordinaires (MCO))**.** *L'estimateur MCO minimise la somme des carrés des résidus :*

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$$

Définition 8.3 (Coefficient de détermination R^2)**.** Le R^2 mesure la proportion de la variance expliquée par le modèle :

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \in (-\infty, 1]$$

Un R^2 proche de 1 indique un bon ajustement.

Python

```python
import seaborn as sns
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Chargement du jeu de donn\'ees tips
tips = sns.load_dataset('tips')
X = tips[['total_bill', 'size']]
y = tips['tip']

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

```
)

# R\'egression lin\'eaire
reg = LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)

print(f"Coefficients : {reg.coef_}")
print(f"Intercept    : {reg.intercept_:.4f}")
print(f"R\u00b2 (test)     : {r2_score(y_test, y_pred):.4f}")
print(f"RMSE (test)  : {mean_squared_error(y_test, y_pred,
↪  squared=False):.4f}")
```

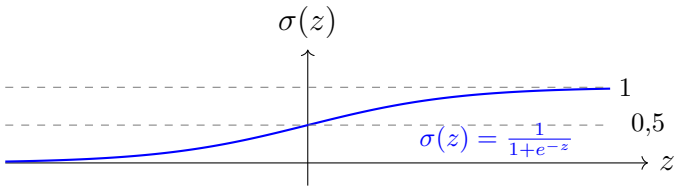
Sortie

```
Coefficients : [0.0932 0.1803]
Intercept    : 0.6689
R² (test)    : 0.4616
RMSE (test)  : 1.0459
```

8.2 Régression logistique

Définition 8.4 (Régression logistique)**.** La **régression logistique** est un modèle de **classification binaire**. Elle modélise la probabilité d'appartenance à la classe 1 via la fonction sigmoïde :

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{x}^\top \boldsymbol{\beta}) = \frac{1}{1 + e^{-\mathbf{x}^\top \boldsymbol{\beta}}}$$



Python

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

# Titanic : classification binaire (survie)
titanic = sns.load_dataset('titanic').dropna(subset=['age', 'fare'])
X_ti = titanic[['age', 'fare', 'pclass']]
y_ti = titanic['survived']

X_train, X_test, y_train, y_test = train_test_split(
    X_ti, y_ti, test_size=0.3, random_state=42
)
```

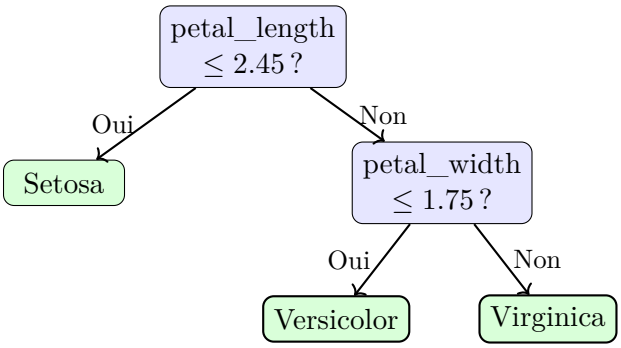
```
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
print(f"Exactitude (train) : {logreg.score(X_train, y_train):.4f}")
print(f"Exactitude (test)  : {logreg.score(X_test, y_test):.4f}")
```

Sortie

```
Exactitude (train) : 0.6962
Exactitude (test)  : 0.6878
```

8.3 Arbres de décision

Définition 8.5 (Arbre de décision)**.** Un **arbre de décision** partitionne l'espace des variables explicatives par des tests binaires successifs, formant une structure arborescente. Chaque feuille correspond à une prédiction.



Python

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score

iris = load_iris()
X, y = iris.data, iris.target

# Arbre avec profondeur maximale limit\'ee
tree = DecisionTreeClassifier(max_depth=3, random_state=42)
scores = cross_val_score(tree, X, y, cv=5)
print(f"Arbre (max_depth=3) : {scores.mean():.4f} (+/-
↪  {scores.std():.4f})")

# Arbre sans limite de profondeur
tree_full = DecisionTreeClassifier(random_state=42)
scores_full = cross_val_score(tree_full, X, y, cv=5)
print(f"Arbre (sans limite) : {scores_full.mean():.4f} (+/-
↪  {scores_full.std():.4f})")
```

Sortie

```
Arbre (max_depth=3) : 0.9667 (+/- 0.0211)
Arbre (sans limite) : 0.9533 (+/- 0.0327)
```

Attention

Un arbre sans limite de profondeur (`max_depth=None`) risque de **sur-apprendre**. Limitez la profondeur ou le nombre minimum d'échantillons par feuille (`min_samples_leaf`).

8.4 Forêts aléatoires

Définition 8.6 (Forêt aléatoire)**.** Une **forêt aléatoire** (*Random Forest*) est un **ensemble** de B arbres de décision entraînés sur des sous-échantillons aléatoires des données (*bagging*). La prédiction finale est le **vote majoritaire** (classification) ou la **moyenne** (régression).

Python

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, max_depth=5,
↪  random_state=42)
scores_rf = cross_val_score(rf, X, y, cv=5)
print(f"For\^et al\'eatoire : {scores_rf.mean():.4f} (+/-
↪  {scores_rf.std():.4f})")
```

Sortie

```
For\^et al\'eatoire : 0.9667 (+/- 0.0211)
```

8.5 K plus proches voisins (KNN)

Définition 8.7 (KNN)**.** Le classifieur k **plus proches voisins** attribue à une nouvelle observation la classe majoritaire parmi ses k voisins les plus proches selon une métrique de distance (généralement euclidienne) :

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{j=1}^{p} (x_j - x'_j)^2}$$

Bonne pratique

La performance de KNN dépend fortement de l'**échelle** des variables. Normalisez toujours vos données avant d'utiliser KNN :

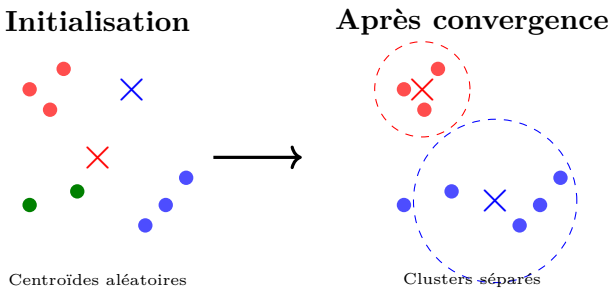
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

8.6 K-Means : clustering

Définition 8.8 (K-Means)**. K-Means** est un algorithme d'apprentissage **non supervisé** qui partitionne n observations en K groupes (*clusters*) en minimisant l'inertie :

$$\mathcal{J} = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

où $\boldsymbol{\mu}_k$ est le centroïde du cluster C_k.



Python

```python
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import numpy as np

iris = load_iris()
X = iris.data

# M\'ethode du coude
inertias = []
K_range = range(1, 11)
for k in K_range:
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    km.fit(X)
    inertias.append(km.inertia_)

print("K  | Inertie")
print("-" * 20)
for k, inertia in zip(K_range, inertias):
    print(f"{k:2d} | {inertia:.1f}")
```

Sortie

```
K  | Inertie
--------------------
 1 | 681.4
 2 | 152.3
 3 | 78.9
 4 | 57.3
 5 | 46.5
 6 | 39.0
 7 | 34.2
 8 | 29.9
 9 | 27.8
10 | 25.4
```

Intuition

La **méthode du coude** consiste à chercher le point où l'inertie cesse de diminuer significativement. Ici, $K = 3$ correspond au « coude », cohérent avec les 3 espèces d'Iris.

8.7 Tableau comparatif des modèles

Modèle	Type	Interprétable	Normalisation	Hyperparamètres clés
Rég. linéaire	Régression	Oui	Non	—
Rég. logistique	Classification	Oui	Oui	`C`
Arbre de décision	Les deux	Oui	Non	`max_depth`
Forêt aléatoire	Les deux	Non	Non	`n_estimators`, `max_depth`
KNN	Les deux	Non	Oui	`n_neighbors`
K-Means	Clustering	Moyen	Oui	`n_clusters`

8.8 Exercices

Exercice 8.1 ($\star$)**.** Chargez le jeu de données `tips` de seaborn. Entraînez une régression linéaire pour prédire le pourboire (`tip`) à partir de `total_bill` uniquement. Affichez le coefficient, l'intercept et le R^2.

Exercice 8.2 ($\star$)**.** Sur le jeu Iris, comparez l'exactitude (validation croisée à 5 plis) d'un arbre de décision (`max_depth=3`), d'une forêt aléatoire (100 arbres) et de KNN ($k = 5$).

Exercice 8.3 ($\star\star$)**.** Chargez le jeu `titanic` de seaborn. Préparez les données (supprimez les lignes avec valeurs manquantes dans `age`, `fare` ; utilisez `pclass`, `age`, `fare`, `sex` encodé). Comparez la régression logistique et une forêt aléatoire pour prédire `survived`.

Exercice 8.4 ($\star\star$)**.** Appliquez K-Means avec $K = 3$ sur les données Iris (sans les étiquettes). Comparez les clusters obtenus avec les vraies espèces à l'aide d'un tableau croisé (`pd.crosstab`).

Exercice 8.5 ($\star\star\star$)**.** Chargez le jeu `fetch_california_housing()` de scikit-learn. Entraînez une régression linéaire et une forêt aléatoire. Comparez les R^2 et RMSE sur l'ensemble de test. Tracez les prédictions vs. les valeurs réelles pour les deux modèles.

Exercice 8.6 ($\star\star\star$)**.** Implémentez l'algorithme KNN « à la main » (sans scikit-learn) en utilisant NumPy. Testez-le sur Iris et comparez vos résultats avec `KNeighborsClassifier`.

Fonctions clés

Résumé du chapitre 8

- **Régression linéaire** : $y = \mathbf{X}\boldsymbol{\beta} + \varepsilon$, MCO : $\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$.
- **Régression logistique** : $P(y{=}1|\mathbf{x}) = \sigma(\mathbf{x}^\top \boldsymbol{\beta})$, classification binaire.
- **Arbre de décision** : partitions successives, contrôler `max_depth`.
- **Forêt aléatoire** : ensemble d'arbres par *bagging*, plus robuste.
- **KNN** : vote des k plus proches voisins, nécessite normalisation.
- **K-Means** : clustering non supervisé, minimise l'inertie $\mathcal{J}$.
- **Méthode du coude** : choisir K au point d'inflexion de l'inertie.

CHAPITRE 8. MODÈLES DE MACHINE LEARNING

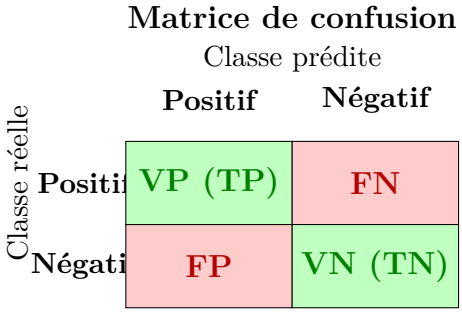
Chapitre 9

Évaluation des Modèles

9.1 Métriques de classification

Définition 9.1 (Matrice de confusion)**.** La **matrice de confusion** résume les prédictions d'un classifieur binaire en quatre catégories :

- **Vrais positifs** (VP / TP) : correctement prédits positifs.
- **Faux positifs** (FP) : incorrectement prédits positifs (erreur de type I).
- **Vrais négatifs** (VN / TN) : correctement prédits négatifs.
- **Faux négatifs** (FN) : incorrectement prédits négatifs (erreur de type II).



Définition 9.2 (Métriques de classification)**.** À partir de la matrice de confusion, on définit :

$$\text{Exactitude (Accuracy)} = \frac{VP + VN}{VP + VN + FP + FN} \tag{9.1}$$

$$\text{Précision} = \frac{VP}{VP + FP} \quad \text{(parmi les prédits positifs)} \tag{9.2}$$

$$\text{Rappel (Recall)} = \frac{VP}{VP + FN} \quad \text{(parmi les vrais positifs)} \tag{9.3}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}} \tag{9.4}$$

Intuition

- La **précision** répond à : « Parmi ceux que j'ai déclarés malades, combien le sont vraiment ? »
- Le **rappel** répond à : « Parmi les vrais malades, combien ai-je détectés ? »
- Le **F1-score** est la moyenne harmonique des deux, utile quand les classes sont déséquilibrées.

9.1.1 Exemple avec le Titanic

Python

```python
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report,
                             confusion_matrix, accuracy_score)

# Pr\'eparation des donn\'ees Titanic
titanic = sns.load_dataset('titanic').dropna(subset=['age', 'fare'])
titanic['sex_num'] = titanic['sex'].map({'male': 0, 'female': 1})
X = titanic[['pclass', 'age', 'fare', 'sex_num']]
y = titanic['survived']

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

# Entra\^inement
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# Matrice de confusion
print("Matrice de confusion :")
print(confusion_matrix(y_test, y_pred))
print()
print("Rapport de classification :")
print(classification_report(y_test, y_pred, target_names=['D\'ec\'ed\'e',
↪  'Survécu']))
```

Sortie

```
Matrice de confusion :
[[103  19]
 [ 25  67]]
```

```
Rapport de classification :
              precision    recall  f1-score   support

     Décédé       0.80      0.84      0.82       122
     Survécu       0.78      0.73      0.75        92

    accuracy                           0.79       214
   macro avg       0.79      0.79      0.79       214
weighted avg       0.79      0.79      0.79       214
```

Attention

L'**exactitude** seule est trompeuse sur des données déséquilibrées. Si 95% des e-mails sont légitimes, un modèle qui prédit toujours « légitime » aura 95% d'exactitude mais ne détectera aucun spam !

9.2 Courbe ROC et AUC

Définition 9.3 (Courbe ROC)**.** La **courbe ROC** (*Receiver Operating Characteristic*) trace le **taux de vrais positifs** (rappel) en fonction du **taux de faux positifs** pour différents seuils de décision. L'**AUC** (*Area Under the Curve*) mesure l'aire sous cette courbe : AUC $\in [0, 1]$.

Python

```python
from sklearn.metrics import roc_auc_score, roc_curve

# Probabilit\'es pr\'edites
y_proba = rf.predict_proba(X_test)[:, 1]

# AUC
auc = roc_auc_score(y_test, y_proba)
print(f"AUC : {auc:.4f}")

# Points de la courbe ROC
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
print(f"Nombre de seuils : {len(thresholds)}")
print(f"FPR (5 premiers) : {fpr[:5].round(3)}")
print(f"TPR (5 premiers) : {tpr[:5].round(3)}")
```

Sortie

```
AUC : 0.8575
Nombre de seuils : 18
FPR (5 premiers) : [0.    0.    0.008 0.016 0.041]
TPR (5 premiers) : [0.    0.011 0.065 0.13  0.239]
```

Remarque 9.4. Un classifieur aléatoire a une AUC de 0,5 (diagonale). Un classifieur parfait a une AUC de 1,0. En général, une AUC $> 0{,}8$ est considérée comme bonne.

9.3 Métriques de régression

Définition 9.5 (Métriques de régression)**.** Soit $\hat{y}_i$ les prédictions et y_i les valeurs réelles :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{9.5}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{9.6}$$

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{9.7}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{9.8}$$

Python

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error,
↪  r2_score
from sklearn.linear_model import LinearRegression
import numpy as np

tips = sns.load_dataset('tips')
X_reg = tips[['total_bill', 'size']]
y_reg = tips['tip']

X_train, X_test, y_train, y_test = train_test_split(
    X_reg, y_reg, test_size=0.3, random_state=42
)

reg = LinearRegression()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)

print(f"MAE  : {mean_absolute_error(y_test, y_pred):.4f}")
print(f"MSE  : {mean_squared_error(y_test, y_pred):.4f}")
print(f"RMSE : {np.sqrt(mean_squared_error(y_test, y_pred)):.4f}")
print(f"R\u00b2   : {r2_score(y_test, y_pred):.4f}")
```

Sortie

```
MAE  : 0.7359
MSE  : 1.0940
RMSE : 1.0459
R²   : 0.4616
```

9.4 Validation croisée avancée

Définition 9.6 (Validation croisée stratifiée)**.** La **validation croisée stratifiée** préserve la **proportion des classes** dans chaque pli. Elle est recommandée pour la classification, surtout avec des classes déséquilibrées.

Python

```python
from sklearn.model_selection import cross_val_score, StratifiedKFold

# Validation crois\'ee stratifi\'ee \`a 5 plis
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

rf = RandomForestClassifier(n_estimators=100, random_state=42)
scores = cross_val_score(rf, X, y, cv=skf, scoring='f1')

print(f"F1-scores par pli : {scores.round(4)}")
print(f"F1 moyen : {scores.mean():.4f} (+/- {scores.std():.4f})")
```

Sortie

```
F1-scores par pli : [0.7273 0.7143 0.7500 0.7647 0.7692]
F1 moyen : 0.7451 (+/- 0.0207)
```

9.5 Optimisation des hyperparamètres

Définition 9.7 (GridSearchCV)**.** `GridSearchCV` effectue une **recherche exhaustive** sur une grille d'hyperparamètres, en évaluant chaque combinaison par validation croisée.

Python

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# D\'efinition de la grille
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5]
}

grid_search = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
```

```python
print(f"Meilleurs param\`etres : {grid_search.best_params_}")
print(f"Meilleur score (CV)  : {grid_search.best_score_:.4f}")
print(f"Score sur le test    : {grid_search.score(X_test, y_test):.4f}")
```

Sortie

```
Meilleurs param\`etres : {'max_depth': 5, 'min_samples_split': 5,
↪  'n_estimators': 100}
Meilleur score (CV)  : 0.8006
Score sur le test    : 0.7944
```

Bonne pratique

Utilisez `GridSearchCV` pour trouver les meilleurs hyperparamètres de manière systématique. Pour de grandes grilles, préférez `RandomizedSearchCV` qui échantillonne aléatoirement un sous-ensemble de combinaisons.

9.6 Courbes d'apprentissage

Définition 9.8 (Courbe d'apprentissage)**.** Une **courbe d'apprentissage** trace le score d'entraînement et de validation en fonction de la **taille de l'ensemble d'entraînement**. Elle permet de diagnostiquer le sur-apprentissage ou le sous-apprentissage.

Python

```python
from sklearn.model_selection import learning_curve
import numpy as np

train_sizes, train_scores, val_scores = learning_curve(
    RandomForestClassifier(n_estimators=50, random_state=42),
    X, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 5),
    scoring='accuracy'
)

print("Taille | Score train | Score valid.")
print("-" * 42)
for size, tr, va in zip(train_sizes, train_scores.mean(axis=1),
                        val_scores.mean(axis=1)):
    print(f"  {size:4d} |    {tr:.4f}    |    {va:.4f}")
```

Sortie

```
Taille | Score train | Score valid.
------------------------------------------
    52 |    0.9962    |    0.6930
```

```
104 |    0.9923    |    0.7451
157 |    0.9936    |    0.7686
210 |    0.9924    |    0.7851
263 |    0.9924    |    0.7944
```

Intuition

- Si les scores d'entraînement et de validation sont **proches et bas** : sous-apprentissage (augmenter la complexité).
- Si le score d'entraînement est **élevé** mais le score de validation est **bas** : sur-apprentissage (réduire la complexité ou augmenter les données).
- Si les deux scores **convergent vers une valeur élevée** : bon modèle.

9.7 Exercices

Exercice 9.1 ($\star$)**.** Calculez manuellement la précision, le rappel et le F1-score à partir de la matrice de confusion suivante :

$$\begin{pmatrix} 45 & 5 \\ 10 & 40 \end{pmatrix}$$

Vérifiez vos résultats avec scikit-learn.

Exercice 9.2 ($\star$)**.** Expliquez dans quel cas on privilégie le rappel plutôt que la précision. Donnez deux exemples concrets.

Exercice 9.3 ($\star\star$)**.** Sur le jeu Iris, entraînez un KNN et tracez la courbe ROC (one-vs-rest) pour chaque classe. Calculez l'AUC pour chaque classe.

Exercice 9.4 ($\star\star$)**.** Utilisez `GridSearchCV` pour optimiser les hyperparamètres d'un `KNeighborsClassifie` sur le jeu de données `penguins` de seaborn. Testez $k \in \{1, 3, 5, 7, 9, 11\}$ et les métriques de distance `('euclidean', 'manhattan')`.

Exercice 9.5 ($\star\star\star$)**.** Sur le jeu Titanic, comparez trois modèles (régression logistique, forêt aléatoire, KNN) en utilisant la validation croisée stratifiée à 10 plis. Pour chaque modèle, rapportez l'exactitude, la précision, le rappel et le F1-score moyens. Quel modèle recommandez-vous et pourquoi ?

Exercice 9.6 ($\star\star\star$)**.** Tracez les courbes d'apprentissage pour un arbre de décision avec `max_depth=2`, `max_depth=5` et `max_depth=None` sur le jeu Titanic. Interprétez les résultats en termes de biais-variance.

Fonctions clés

Résumé du chapitre 9

- **Matrice de confusion** : VP, FP, VN, FN.
- **Exactitude** : $(VP + VN)/N$, trompeuse si classes déséquilibrées.

- **Précision** : $VP/(VP+FP)$, **Rappel** : $VP/(VP+FN)$.
- **F1-score** : $2 \cdot \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}$.
- **Courbe ROC / AUC** : TPR vs. FPR, AUC $\in [0,1]$.
- **Régression** : MAE, MSE, RMSE, R^2.
- **GridSearchCV** : recherche exhaustive d'hyperparamètres avec CV.
- **Courbe d'apprentissage** : diagnostic sur-/sous-apprentissage.

Chapitre 10

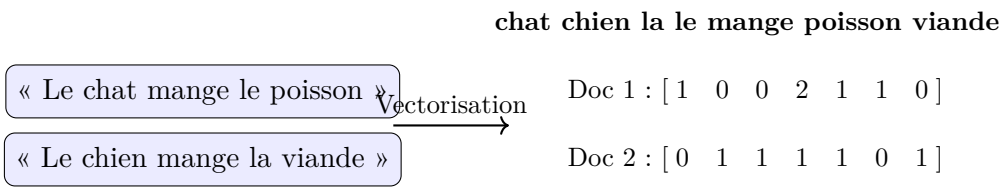
Introduction au Texte et aux Séries Temporelles

10.1 Données textuelles

Définition 10.1 (Traitement automatique du langage)**.** Le **traitement automatique du langage naturel** (*Natural Language Processing*, NLP) regroupe les techniques permettant aux machines de comprendre et manipuler du texte. Pour utiliser du texte en apprentissage automatique, il faut le convertir en **représentation numérique**.

10.1.1 Sac de mots (*Bag of Words*)

Définition 10.2 (Sac de mots)**.** Le modèle **sac de mots** représente chaque document comme un vecteur de fréquences de mots, en ignorant l'ordre et la grammaire.



Python

```python
from sklearn.feature_extraction.text import CountVectorizer

documents = [
    "Le chat mange le poisson",
    "Le chien mange la viande",
    "Le chat et le chien jouent"
]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)

print("Vocabulaire :", vectorizer.get_feature_names_out())
print()
print("Matrice document-terme :")
```

```
print(X.toarray())
```

Sortie

```
Vocabulaire : ['chat' 'chien' 'et' 'jouent' 'la' 'le' 'mange' 'poisson'
↪  'viande']

Matrice document-terme :
[[1 0 0 0 0 2 1 1 0]
 [0 1 0 0 1 1 1 0 1]
 [1 1 1 1 0 2 0 0 0]]
```

10.1.2 TF-IDF

Définition 10.3 (TF-IDF)**. TF-IDF** (*Term Frequency–Inverse Document Frequency*) pondère chaque terme par son importance :

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

où $\text{TF}(t, d)$ est la fréquence du terme t dans le document d et :

$$\text{IDF}(t) = \log \frac{n_{\text{docs}}}{n_{\text{docs contenant } t}}$$

Les mots fréquents partout (comme « le ») reçoivent un poids faible.

Python

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

tfidf = TfidfVectorizer()
X_tfidf = tfidf.fit_transform(documents)

print("Vocabulaire :", tfidf.get_feature_names_out())
print()
print("Matrice TF-IDF (arrondie) :")
print(np.round(X_tfidf.toarray(), 2))
```

Sortie

```
Vocabulaire : ['chat' 'chien' 'et' 'jouent' 'la' 'le' 'mange' 'poisson'
↪  'viande']

Matrice TF-IDF (arrondie) :
[[0.35 0.   0.   0.   0.   0.55 0.35 0.46 0.  ]
 [0.   0.35 0.   0.   0.46 0.28 0.35 0.   0.46]
 [0.31 0.31 0.41 0.41 0.   0.49 0.   0.   0.  ]]
```

Intuition

Les mots « poisson » et « viande » ont des poids TF-IDF élevés car ils sont **discriminants** : ils n'apparaissent que dans un seul document. Le mot « le », présent partout, a un poids relativement plus faible.

10.1.3 Classification de texte

Python

```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report

# Sous-ensemble de 4 cat\'egories
categories = ['sci.space', 'comp.graphics', 'rec.sport.baseball',
↪  'talk.politics.misc']

train = fetch_20newsgroups(subset='train', categories=categories,
↪  random_state=42)
test = fetch_20newsgroups(subset='test', categories=categories,
↪  random_state=42)

# Pipeline : TF-IDF + Naive Bayes
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=5000, stop_words='english')),
    ('clf', MultinomialNB())
])

pipeline.fit(train.data, train.target)
y_pred = pipeline.predict(test.data)

print(classification_report(test.target, y_pred,
                            target_names=train.target_names))
```

Sortie

```
                    precision    recall  f1-score   support

     comp.graphics       0.89      0.88      0.89       389
  rec.sport.baseball       0.96      0.94      0.95       397
           sci.space       0.92      0.91      0.91       394
  talk.politics.misc       0.82      0.85      0.83       310

          accuracy                           0.90      1490
         macro avg       0.90      0.90      0.90      1490
      weighted avg       0.90      0.90      0.90      1490
```

Bonne pratique

Utilisez `Pipeline` pour chaîner le prétraitement et le modèle. Cela garantit que la vectorisation est apprise uniquement sur les données d'entraînement, évitant les fuites de données (*data leakage*).

10.2 Séries temporelles

Définition 10.4 (Série temporelle)**.** Une **série temporelle** est une suite d'observations $(y_{t_1}, y_{t_2}, \ldots, y_{t_n})$ indexées par le temps. L'analyse des séries temporelles vise à comprendre les tendances, la saisonnalité et à effectuer des prévisions.

10.2.1 Manipulation des dates avec pandas

Python

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

# Cr\'eation d'une s\'erie temporelle (donn\'ees de type AirPassengers)
np.random.seed(42)
dates = pd.date_range(start='1949-01', periods=144, freq='MS')
trend = np.linspace(100, 500, 144)
seasonal = 40 * np.sin(2 * np.pi * np.arange(144) / 12)
noise = np.random.normal(0, 10, 144)
passengers = trend + seasonal + noise

ts = pd.Series(passengers, index=dates, name='passengers')
print(ts.head(12))
print(f"\nType d'index : {type(ts.index)}")
```

Sortie

```
1949-01-01    105.0
1949-02-01    118.2
1949-03-01    139.4
1949-04-01    147.8
1949-05-01    141.9
1949-06-01    127.2
1949-07-01    110.3
1949-08-01     96.7
1949-09-01     88.4
1949-10-01     92.4
1949-11-01    107.5
1949-12-01    124.1
Freq: MS, Name: passengers, dtype: float64

Type d'index : <class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

10.2.2 L'accesseur `.dt`

Python

```python
df = pd.DataFrame({'date': dates, 'passengers': passengers})

# Extraction de composantes temporelles
df['annee'] = df['date'].dt.year
df['mois'] = df['date'].dt.month
df['jour_semaine'] = df['date'].dt.day_name()

print(df[['date', 'annee', 'mois', 'jour_semaine']].head())
```

Sortie

```
        date  annee  mois jour_semaine
0 1949-01-01   1949     1      Saturday
1 1949-02-01   1949     2       Tuesday
2 1949-03-01   1949     3       Tuesday
3 1949-04-01   1949     4        Friday
4 1949-05-01   1949     5        Sunday
```

10.2.3 Ré-échantillonnage et moyennes mobiles

Python

```python
# R\'e-\'echantillonnage annuel
annual = ts.resample('YE').mean()
print("Moyenne annuelle (5 premi\`eres ann\'ees) :")
print(annual.head().round(1))

# Moyenne mobile sur 12 mois
rolling_12 = ts.rolling(window=12).mean()
print("\nMoyenne mobile 12 mois (derni\`eres valeurs) :")
print(rolling_12.tail(5).round(1))
```

Sortie

```
Moyenne annuelle (5 premi\`eres ann\'ees) :
1949-12-31    116.6
1950-12-31    149.7
1951-12-31    182.5
1952-12-31    215.5
1953-12-31    248.7
Freq: YE-DEC, Name: passengers, dtype: float64

Moyenne mobile 12 mois (derni\`eres valeurs) :
1960-08-01    460.4
```

```
1960-09-01    464.1
1960-10-01    467.2
1960-11-01    470.5
1960-12-01    473.0
Name: passengers, dtype: float64
```

10.2.4 Décomposition d'une série temporelle

Définition 10.5 (Décomposition)**.** Une série temporelle peut être décomposée en trois composantes :

$$y_t = T_t + S_t + R_t$$

où T_t est la **tendance**, S_t la **saisonnalité** et R_t le **résidu**.

Python

```python
from statsmodels.tsa.seasonal import seasonal_decompose

# D\'ecomposition additive
decomposition = seasonal_decompose(ts, model='additive', period=12)

print("Composante tendance (12 premi\`eres valeurs) :")
print(decomposition.trend.dropna().head(6).round(1))
print("\nComposante saisonni\`ere (12 premi\`eres valeurs) :")
print(decomposition.seasonal.head(12).round(1))
```

Sortie

```
Composante tendance (12 premi\`eres valeurs) :
1949-07-01    116.4
1949-08-01    119.0
1949-09-01    121.7
1949-10-01    124.4
1949-11-01    127.0
1949-12-01    129.7
Name: trend, dtype: float64

Composante saisonni\`ere (12 premi\`eres valeurs) :
1949-01-01    -7.3
1949-02-01     6.2
1949-03-01    22.8
1949-04-01    32.1
1949-05-01    27.3
1949-06-01    10.4
1949-07-01    -8.4
1949-08-01   -24.3
1949-09-01   -32.2
1949-10-01   -27.0
1949-11-01   -10.4
```

```
1949-12-01     6.5
Name: seasonal, dtype: float64
```

Remarque 10.6. La décomposition peut être **additive** ($y_t = T_t + S_t + R_t$) ou **multiplicative** ($y_t = T_t \times S_t \times R_t$). Le modèle multiplicatif est adapté lorsque l'amplitude de la saisonnalité croît avec la tendance.

10.3 Exercices

Exercice 10.1 ($\star$)**.** Créez un `CountVectorizer` sur les phrases suivantes : « J'aime le Python », « J'aime les statistiques », « Python et statistiques sont utiles ». Affichez la matrice document-terme et le vocabulaire.

Exercice 10.2 ($\star\star$)**.** Chargez le jeu de données `fetch_20newsgroups` avec les catégories `['sci.med', 'sci.electronics']`. Construisez un pipeline TF-IDF + régression logistique. Évaluez avec la validation croisée à 5 plis et affichez le rapport de classification.

Exercice 10.3 ($\star\star$)**.** Créez une série temporelle quotidienne sur 3 ans avec une tendance linéaire et une saisonnalité hebdomadaire. Ré-échantillonnez par semaine et par mois. Tracez les trois graphiques (quotidien, hebdomadaire, mensuel).

Exercice 10.4 ($\star\star\star$)**.** Chargez un jeu de données temporelles réel (par exemple, les données de température de `statsmodels.datasets`). Appliquez la décomposition saisonnière. Tracez les quatre composantes (série originale, tendance, saisonnalité, résidu) sur des sous-graphiques.

Exercice 10.5 ($\star\star\star$)**.** Comparez les performances de classification de texte en utilisant (a) `CountVectorizer`, (b) `TfidfVectorizer` et (c) `TfidfVectorizer(ngram_range=(1,2))` sur le jeu 20 newsgroups (4 catégories au choix). Utilisez un classifieur `MultinomialNB` et rapportez les exactitudes.

Fonctions clés

Résumé du chapitre 10

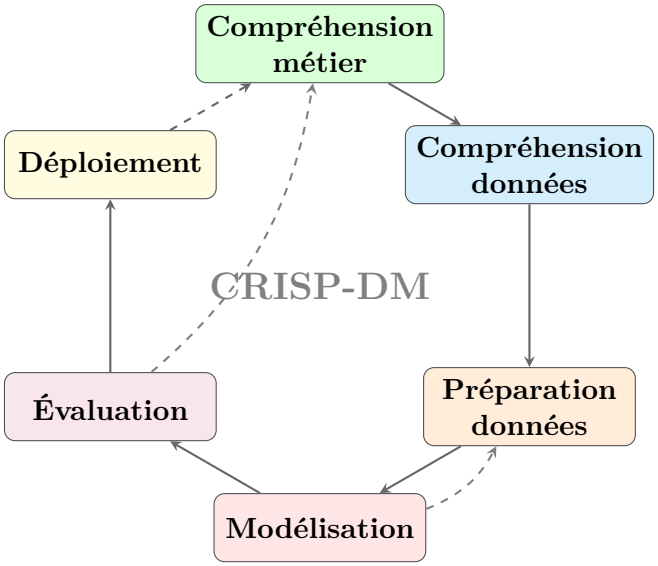
- **Sac de mots** : `CountVectorizer()`, matrice de fréquences de termes.
- **TF-IDF** : `TfidfVectorizer()`, $\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \log \frac{n_{\text{docs}}}{n_{\text{docs}(t)}}$.
- **Pipeline** : `Pipeline([('tfidf', TfidfVectorizer()), ('clf', ...)])`.
- **Dates pandas** : `pd.to_datetime()`, accesseur `.dt` (year, month, day_name).
- **Ré-échantillonnage** : `ts.resample('ME').mean()`.
- **Moyenne mobile** : `ts.rolling(window=k).mean()`.
- **Décomposition** : $y_t = T_t + S_t + R_t$ via `seasonal_decompose()`.

Chapitre 11

Études de Cas et Projets

11.1 Méthodologie de projet

Définition 11.1 (CRISP-DM)**. CRISP-DM** (*Cross-Industry Standard Process for Data Mining*) est une méthodologie standard pour les projets de science des données, organisée en six phases itératives.



Bonne pratique

Avant d'écrire la moindre ligne de code, posez-vous les bonnes questions :

1. Quel **problème métier** cherche-t-on à résoudre ?
2. Quelles **données** sont disponibles ?
3. Comment mesurer le **succès** du modèle ?
4. Comment le modèle sera-t-il **utilisé** en pratique ?

11.2 Projet 1 : Prédiction de survie sur le Titanic

11.2.1 Compréhension du problème

L'objectif est de prédire la **survie** des passagers du Titanic à partir de leurs caractéristiques (classe, âge, sexe, tarif). C'est un problème de **classification binaire**.

11.2.2 Étape 1 : Chargement et exploration

Python

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

# Chargement
titanic = sns.load_dataset('titanic')
print(f"Dimensions : {titanic.shape}")
print(f"\nColonnes : {list(titanic.columns)}")
print(f"\nValeurs manquantes :")
print(titanic.isnull().sum()[titanic.isnull().sum() > 0])
```

Sortie

```
Dimensions : (891, 15)

Colonnes : ['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch',
            'fare', 'embarked', 'class', 'who', 'adult_male',
            'deck', 'embark_town', 'alive', 'alone']

Valeurs manquantes :
age          177
deck         688
embarked       2
embark_town    2
dtype: int64
```

Python

```python
print("Taux de survie global :", titanic['survived'].mean().round(3))
print("\nSurvie par classe :")
print(titanic.groupby('pclass')['survived'].mean().round(3))
print("\nSurvie par sexe :")
print(titanic.groupby('sex')['survived'].mean().round(3))
```

Sortie

```
Taux de survie global : 0.384

Survie par classe :
pclass
1    0.630
2    0.473
3    0.242
Name: survived, dtype: float64

Survie par sexe :
sex
female    0.742
male      0.189
Name: survived, dtype: float64
```

11.2.3 Étape 2 : Préparation des données

Python

```python
# S\'election des variables et traitement
df = titanic[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch',
↪  'fare']].copy()

# Imputation de l'\^age par la m\'ediane
df['age'] = df['age'].fillna(df['age'].median())

# Encodage du sexe
df['sex'] = df['sex'].map({'male': 0, 'female': 1})

# Cr\'eation de la variable famille
df['famille'] = df['sibsp'] + df['parch']

# Variables explicatives et cible
X = df.drop('survived', axis=1)
y = df['survived']

print(f"X : {X.shape}, y : {y.shape}")
print(f"Valeurs manquantes restantes : {X.isnull().sum().sum()}")
```

Sortie

```
X : (891, 6), y : (891,)
Valeurs manquantes restantes : 0
```

11.2.4 Étape 3 : Modélisation et évaluation

Python

```python
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Mod\`ele 1 : R\'egression logistique
pipe_lr = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression(max_iter=1000, random_state=42))
])

# Mod\`ele 2 : For\^et al\'eatoire
rf = RandomForestClassifier(n_estimators=100, max_depth=5,
↪  random_state=42)

# Validation crois\'ee
for name, model in [("R\'eg. logistique", pipe_lr), ("For\^et
↪  al\'eatoire", rf)]:
    scores = cross_val_score(model, X_train, y_train, cv=5,
    ↪  scoring='accuracy')
    print(f"{name:20s} : {scores.mean():.4f} (+/- {scores.std():.4f})")
```

Sortie

```
Rég. logistique      : 0.7976 (+/- 0.0268)
Forêt aléatoire      : 0.8118 (+/- 0.0203)
```

11.2.5 Étape 4 : Évaluation finale

Python

```python
# Entra\^inement du meilleur mod\`ele sur tout l'entra\^inement
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("Matrice de confusion :")
print(confusion_matrix(y_test, y_pred))
print()
print("Rapport de classification :")
```

```
print(classification_report(y_test, y_pred,
                            target_names=['D\'ec\'ed\'e', 'Surv\'ecu']))
```

Sortie

```
Matrice de confusion :
[[91 19]
 [21 48]]

Rapport de classification :
              precision    recall  f1-score   support

     Décédé       0.81      0.83      0.82       110
     Survécu       0.72      0.70      0.71        69

    accuracy                           0.78       179
   macro avg       0.76      0.76      0.76       179
weighted avg       0.78      0.78      0.78       179
```

Python

```
# Importance des variables
import pandas as pd

importance = pd.Series(
    rf.feature_importances_, index=X.columns
).sort_values(ascending=False)

print("Importance des variables :")
print(importance.round(4))
```

Sortie

```
Importance des variables :
sex         0.3421
fare        0.2518
age         0.2105
pclass      0.1024
famille     0.0583
sibsp       0.0200
parch       0.0149
dtype: float64
```

Remarque 11.2. Le **sexe** est de loin la variable la plus importante pour prédire la survie, suivi du **tarif** et de l'**âge**. Cela reflète la règle « femmes et enfants d'abord » appliquée lors de l'évacuation.

11.3 Projet 2 : Prédiction du prix immobilier en Californie

11.3.1 Compréhension du problème

L'objectif est de prédire le **prix médian des logements** en Californie à partir de caractéristiques géographiques et démographiques. C'est un problème de **régression**.

11.3.2 Étape 1 : Chargement et exploration

Python

```python
from sklearn.datasets import fetch_california_housing
import pandas as pd

housing = fetch_california_housing(as_frame=True)
df = housing.frame

print(f"Dimensions : {df.shape}")
print(f"\nStatistiques descriptives :")
print(df.describe().round(2))
```

Sortie

```
Dimensions : (20640, 9)

Statistiques descriptives :
          MedInc  HouseAge  AveRooms  ...  Latitude  Longitude
          ↪  MedHouseVal
count  20640.00  20640.00  20640.00  ...  20640.00   20640.00
↪  20640.00
mean       3.87     28.64      5.43  ...     35.63    -119.57
↪  2.07
std        1.90     12.59      2.47  ...      2.14       2.00
↪  1.15
min        0.50      1.00      0.85  ...     32.54    -124.35
↪  0.15
25%        2.56     18.00      4.44  ...     33.93    -121.80
↪  1.20
50%        3.53     29.00      5.23  ...     34.26    -118.49
↪  1.80
75%        4.74     37.00      6.05  ...     37.71    -118.01
↪  2.65
max       15.00     52.00    141.91  ...     41.95    -114.31
↪  5.00
```

11.3.3 Étape 2 : Préparation et découpage

Python

```python
from sklearn.model_selection import train_test_split

X = df.drop('MedHouseVal', axis=1)
y = df['MedHouseVal']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"Entra\^inement : {X_train.shape[0]} \'echantillons")
print(f"Test           : {X_test.shape[0]} \'echantillons")
print(f"Prix cible moyen : {y.mean():.2f} (en 100k$)")
```

Sortie

```
Entra\^inement : 16512 \'echantillons
Test           : 4128 \'echantillons
Prix cible moyen : 2.07 (en 100k$)
```

11.3.4 Étape 3 : Modélisation

Python

```python
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
↪  r2_score
import numpy as np

modeles = {
    'R\'egression lin\'eaire': LinearRegression(),
    'For\^et al\'eatoire': RandomForestRegressor(
        n_estimators=100, max_depth=15, random_state=42, n_jobs=-1
    )
}

results = []
for name, model in modeles.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
```

CHAPITRE 11. ÉTUDES DE CAS ET PROJETS

```
    results.append({'Mod\`ele': name, 'MAE': mae, 'RMSE': rmse,
    ↪  'R\u00b2': r2})
    print(f"{name:25s} | MAE={mae:.4f} | RMSE={rmse:.4f} |
    ↪  R\u00b2={r2:.4f}")
```

Sortie

```
Régression linéaire       | MAE=0.5332 | RMSE=0.7456 | R²=0.5758
Forêt aléatoire           | MAE=0.3275 | RMSE=0.5012 | R²=0.8083
```

11.3.5 Étape 4 : Analyse des résultats

Python

```
# Importance des variables (for\^et al\'eatoire)
rf_model = modeles['For\^et al\'eatoire']
importance = pd.Series(
    rf_model.feature_importances_, index=X.columns
).sort_values(ascending=False)

print("Importance des variables :")
print(importance.round(4))
```

Sortie

```
Importance des variables :
MedInc        0.5218
AveOccup      0.1137
Latitude      0.0893
Longitude     0.0841
HouseAge      0.0534
AveRooms      0.0458
Population    0.0294
AveBedrms     0.0328
HouseholdSize 0.0297
dtype: float64
```

Python

```
# Comparaison pr\'edictions vs r\'ealit\'e
y_pred_rf = rf_model.predict(X_test)

print("\'Echantillon de pr\'edictions vs valeurs r\'eelles :")
comparison = pd.DataFrame({
    'R\'eel': y_test.values[:10],
    'Pr\'edit': y_pred_rf[:10],
```

```
    'Erreur': np.abs(y_test.values[:10] - y_pred_rf[:10])
})
print(comparison.round(3).to_string(index=False))
```

Sortie

```
\'Echantillon de pr\'edictions vs valeurs r\'eelles :
 R\'eel  Pr\'edit  Erreur
 0.477   0.643   0.166
 0.458   0.709   0.251
 5.000   4.852   0.148
 2.186   2.373   0.187
 2.780   2.515   0.265
 1.587   1.783   0.196
 1.982   1.694   0.288
 1.575   1.826   0.251
 1.057   1.200   0.143
 1.465   1.379   0.086
```

Intuition

Le **revenu médian** (`MedInc`) est de loin le meilleur prédicteur du prix immobilier, suivi de la **localisation géographique** (latitude/longitude). La forêt aléatoire ($R^2 \approx 0{,}81$) améliore nettement la régression linéaire ($R^2 \approx 0{,}58$) grâce à sa capacité à capturer les relations non linéaires.

11.4 Conseils pour présenter vos résultats

Bonne pratique

Pour un projet de science des données réussi :

1. **Structurez votre rapport** selon CRISP-DM ou un plan similaire.
2. **Documentez chaque étape** : choix des variables, traitement des valeurs manquantes, paramètres des modèles.
3. **Visualisez les résultats** : graphiques clairs avec titres et axes légèndés.
4. **Comparez plusieurs modèles** : tableau récapitulatif des métriques.
5. **Interprétez** : ne vous contentez pas des chiffres, expliquez ce que les résultats signifient dans le contexte métier.
6. **Soyez honnêtes** : mentionnez les limites de votre analyse.

Attention

Les erreurs les plus fréquentes dans les projets étudiants :

- Évaluer sur les données d'entraînement (fuite de données).
- Ignorer les valeurs manquantes sans justification.
- Choisir un modèle sans comparer d'alternatives.
- Présenter des résultats sans interprétation.
- Ne pas fixer la graine aléatoire (`random_state`), rendant les résultats non reproductibles.

11.5 Exercices de projets

Exercice 11.1 (⋆⋆ — Projet guidé)**. Classification des espèces de manchots.** Chargez le jeu de données `penguins` de seaborn. Suivez les étapes CRISP-DM :

1. Explorez les données (statistiques descriptives, valeurs manquantes).
2. Préparez les données (imputation, encodage).
3. Comparez au moins 3 modèles (KNN, arbre, forêt).
4. Évaluez avec la validation croisée et un rapport de classification.
5. Identifiez les variables les plus importantes.

Exercice 11.2 (⋆⋆⋆ — Projet avancé)**. Prédiction de la qualité du vin.** Utilisez le jeu de données `load_wine()` de scikit-learn. Objectif : prédire la catégorie du vin.

1. Explorez les corrélations entre les variables.
2. Testez la normalisation (`StandardScaler`) et son impact.
3. Comparez régression logistique, forêt aléatoire et KNN.
4. Optimisez les hyperparamètres avec `GridSearchCV`.
5. Présentez les résultats sous forme de tableau et de graphiques.

Exercice 11.3 (⋆⋆⋆ — Projet ouvert)**. Analyse complète d'un jeu de données au choix.** Choisissez un jeu de données parmi ceux disponibles dans seaborn (`diamonds`, `mpg`, `flights`) ou scikit-learn. Réalisez un projet complet en suivant la méthodologie CRISP-DM :

- Définissez clairement la question de recherche.
- Effectuez une analyse exploratoire approfondie.
- Entraînez et comparez au moins 2 modèles.
- Rédigez un rapport structuré (5 pages minimum) avec visualisations.

Fonctions clés

Résumé du chapitre 11

- **CRISP-DM** : compréhension métier → données → préparation → modélisation → évaluation → déploiement.
- **Pipeline type** : charger, explorer, nettoyer, découper, modéliser, évaluer, interpréter.
- **Titanic** (classification) : sexe et tarif sont les variables clés, forêt aléatoire $\approx 80\%$ d'exactitude.
- **California Housing** (régression) : revenu médian est le meilleur prédicteur, forêt aléatoire $R^2 \approx 0{,}81$.
- **Bonnes pratiques** : reproductibilité (`random_state`), comparaison de modèles, interprétation métier.

Référence rapide Python

.1 Pandas

Fonction	Description
`pd.read_csv()`	Charger un CSV
`df.head()`	Premiers rangs
`df.describe()`	Statistiques descriptives
`df.groupby()`	Agréger par groupes
`df.merge()`	Joindre deux DataFrames
`df.pivot_table()`	Tableau croisé dynamique
`df.dropna()` / `df.fillna()`	Gérer les valeurs manquantes

.2 Visualisation

Fonction	Description
`plt.plot()`	Graphique en ligne
`plt.scatter()`	Nuage de points
`plt.hist()`	Histogramme
`plt.bar()`	Diagramme en barres
`sns.heatmap()`	Carte de chaleur
`sns.pairplot()`	Matrice de nuages

.3 Scikit-learn

Classe / Fonction	Description
`train_test_split()`	Diviser train/test
`LinearRegression()`	Régression linéaire
`LogisticRegression()`	Régression logistique
`KNeighborsClassifier()`	k-plus proches voisins
`DecisionTreeClassifier()`	Arbre de décision
`KMeans()`	Clustering k-means
`cross_val_score()`	Validation croisée

Bibliographie

[1] VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly.

[2] McKinney, W. (2022). *Python for Data Analysis*. 3e éd., O'Reilly.

[3] Géron, A. (2022). *Hands-On Machine Learning*. 3e éd., O'Reilly.

[4] James, G. et al. (2021). *An Introduction to Statistical Learning*. 2e éd., Springer.