Université Sultan Moulay Slimane
Ecole Supérieure de Technologie Béni Mellal
Département ITG
DUT- IDIA 1 - S2
M124 – Programmation en Python
Année universitaire: 2024-2025

Support de cours

Programmation en Python

Chapitre VI : Programmation Python pour la science des données

Pr. Siham BAKKOURI
Email: s.bakkouri@usms.ma

Plan

- INTRODUCTION À LA SCIENCE DES DONNÉES
- POURQUOI LA SCIENCE DES DONNÉES ?
- APPLICATIONS DE LA SCIENCE DES DONNÉES
- PYTHON EN SCIENCE DES DONNÉES
- MANIPULATION AVANCÉE DES DONNÉES AVEC `Pandas`
- VISUALISATION AVANCEE DES DONNEES
- TRAITEMENT NUMÉRIQUE AVEC `NumPy`

Introduction à la science des données

La **science des données** est un domaine interdisciplinaire qui combine informatique, statistiques et gestion des données pour extraire des informations utiles à partir de données brutes. Son objectif principal est de transformer ces données en connaissances exploitables pour aider à la prise de décision.

→ Elle repose sur plusieurs étapes

1. **Collecte des données** → récupérer des données depuis des bases de données, des capteurs, des sites web...
2. **Nettoyage et transformation** → organiser et corriger les données.
3. **Analyse et exploration** → comprendre les tendances et relations entre les données.
4. **Visualisation** → créer des graphiques et tableaux pour interpréter les résultats.
5. **Prise de décision** → utiliser ces analyses pour améliorer un produit, optimiser un service ou comprendre un phénomène.

Introduction à la science des données

→ Pourquoi la Science des Données est-elle Importante ?

- Les entreprises et organisations possèdent aujourd'hui **d'énormes volumes de données**.
- Ces données sont une **ressource précieuse** si elles sont bien exploitées. La science des données permet notamment de :
  - **Optimiser les performances** : améliorer la production, la logistique, la gestion des stocks.
  - **Prédire des tendances :** anticiper la demande d'un produit, détecter les fraudes bancaires.
  - **Personnaliser des services** : recommandations sur Netflix, Amazon, YouTube...
  - **Automatiser des tâches** : analyse automatique de documents, chatbot, détection d'anomalies.

Applications de la Science des Données

Voici quelques domaines où la science des données est **indispensable** :

- **Finance** → Détection de fraudes bancaires, prédiction de prix des actions.
- **Santé** → Analyse de dossiers médicaux, diagnostic basé sur des images médicales.
- **Marketing** → Publicités ciblées, recommandations de produits.
- **Transport** → Optimisation du trafic, systèmes de navigation intelligents.
- **Environnement** → Prévisions climatiques, détection de catastrophes naturelles.

Exemple

- Google Maps **analyse en temps réel** les embouteillages pour proposer l'itinéraire le plus rapide.
- Netflix **analyse l'historique** des utilisateurs pour recommander des films adaptés.

Python en science des données

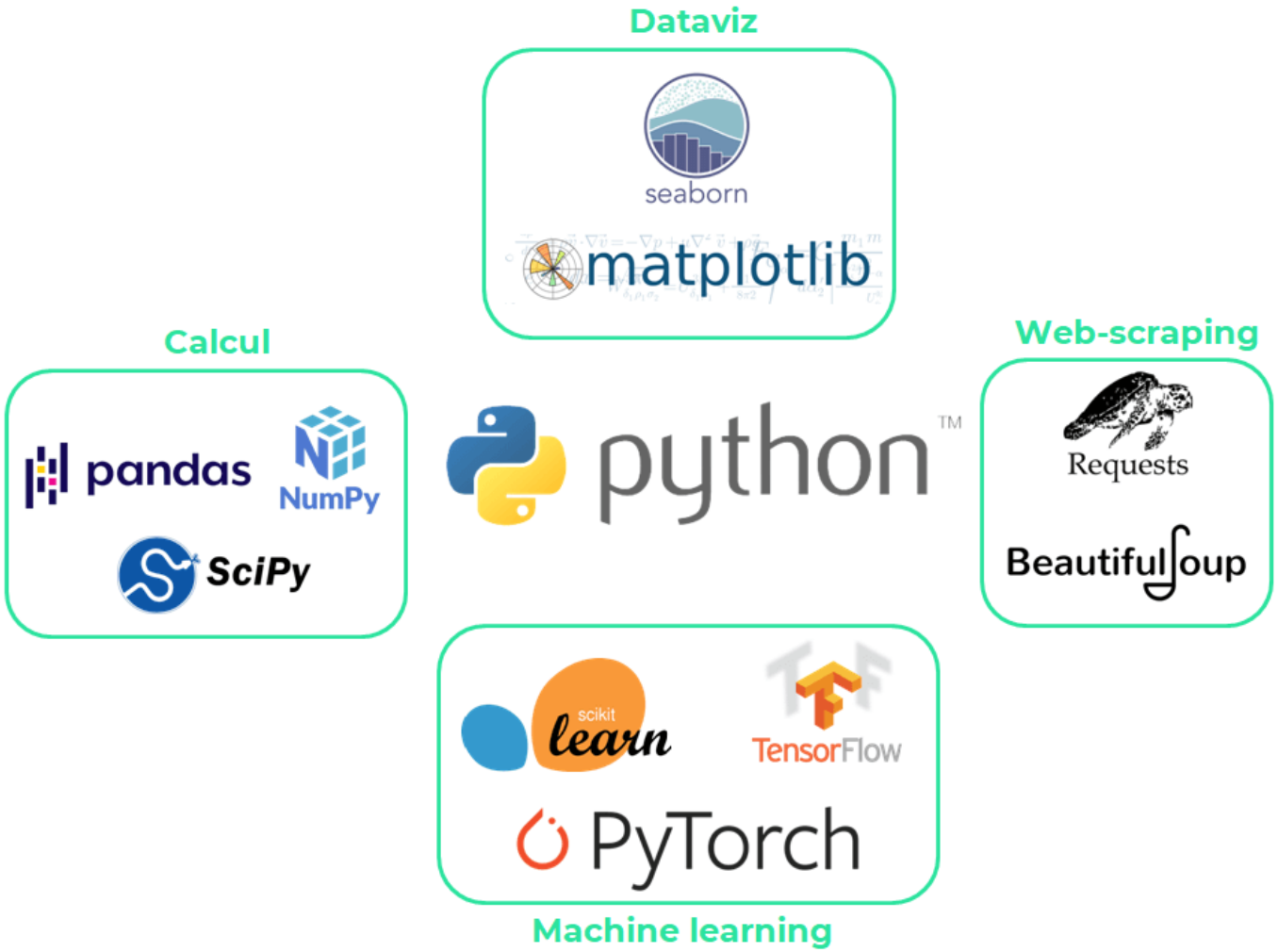
Python est **le langage le plus utilisé** en science des données, car il est :

- Simple et lisible
- Riche en bibliothèques adaptées
- Utilisé par les grandes entreprises et universités

→ **Les bibliothèques les plus courantes en science des données avec Python sont :**

- **Pandas** → Manipulation de tableaux de données.
- **NumPy** → Calculs numériques et matrices.
- **Matplotlib** / **Seaborn** → Visualisation et graphiques.

Python en science des données



→ Modules python pour la science des données

❑ Pandas

- ***Utilité*** : Manipulation et analyse des données tabulaires (semblables aux tableaux Excel).
- ***Installation*** :

```
pip install pandas
```

- ***Importation*** : `import pandas as pd`
- ***Fonctionnalités clés*** :
  - Lire/écrire des fichiers CSV, Excel.
  - Nettoyer les données, gérer les valeurs manquantes.
  - Grouper, filtrer, fusionner des DataFrames.

→ Modules python pour la science des données

❑ NumPy

- ***Utilité*** : Calculs scientifiques rapides, manipulation de tableaux multidimensionnels.
- ***Installation***

```
pip install numpy
```

- ***Importation*** : `import numpy as np`
- ***Fonctionnalités clés***
  - Calculs vectoriels, statistiques, opérations matricielles.
  - Efficace pour le traitement numérique massif.

→ Modules python pour la science des données

❑ Matplotlib

- ***Utilité*** : Création de graphiques 2D (courbes, histogrammes...)
- ***Installation***

```
pip install matplotlib
```

- ***Importation*** : `import matplotlib.pyplot as plt`
- ***Fonctionnalités clés***
  - Tracer des courbes, ajouter des légendes, titres.
  - Sauvegarder les graphiques au format image.

→ Modules python pour la science des données

❑ Seaborn

- ***Utilité*** : Visualisation statistique de haut niveau basée sur ***`Matplotlib`***.
- ***Installation***

```
pip install seaborn
```

- ***Importation*** : `import seaborn as sns`
- ***Fonctionnalités clés***
  - Graphiques avancés : heatmap, boxplot, pairplot...
  - Meilleur rendu esthétique que Matplotlib seul.

Avant de commencer les manipulations avec Pandas, nous allons travailler sur un exemple de fichier ***CSV*** nommé ***data.csv***. Ce fichier représente des données de ventes réalisées par des personnes dans un petit commerce.

❑ Qu'est-ce qu'un fichier CSV ?

Un fichier **CSV** (***Comma-Separated Values***) est un format simple de fichier texte dans lequel les données sont organisées en lignes et colonnes, chaque valeur étant séparée par un caractère spécifique (par défaut la ***`virgule ,`***).

❑ Structure du Fichier data.csv

Colonne	Description
ID	Identifiant unique de chaque ligne.
Nom	Prénom de la personne
Age	Âge de la personne
Prix	Prix du produit acheté en dirhams.
Date	Date de l'achat (format AAAA-MM-JJ).
Ventes	Nombre d'articles vendus (unités).

❑ Contenu du fichier `data.csv`

Le séparateur utilisé dans ce fichier est la **`virgule ,`**

```
ID,Nom,Age,Prix,Date,Ventes
1,Youssef,25,110.5,2023-01-01,10
2,Salma,33,200.0,2023-01-02,15
3,Hamza,28,,2023-01-03,12
4,Nadia,42,150.0,2023-01-04,13
5,Rachid,27,185.5,2023-01-05,
6,Laila,60,220.0,2023-01-06,17
2,Salma,33,200.0,2023-01-02,15
```

→ **Ce fichier contient**

- Des valeurs manquantes pour tester le nettoyage.
- Des lignes dupliquées pour pratiquer la suppression de doublons.
- Des types de données différents (texte, nombres, dates).

ID	Nom	Age	Prix	Date	Ventes
1	Youssef	25	110.5	2023-01-01	10
2	Salma	33	200.0	2023-01-02	15
3	Hamza	28		2023-01-03	12
4	Nadia	42	150.0	2023-01-04	13
5	Rachid	27	185.5	2023-01-05	
6	Laila	60	220.0	2023-01-06	17
2	Salma	33	200.0	2023-01-02	15

→ Chargement et exploration des données

❑ Lecture de fichiers CSV et Excel

```
import pandas as pd
df = pd.read_csv("data.csv")
df_excel = pd.read_excel("data.xlsx")
```

Remarque : Le séparateur par défaut utilisé par **`pandas.read_csv()`** est la ***virgule*** **`,`**. Si votre fichier CSV utilise un autre séparateur (comme `;` ou `|`), il faut le préciser en ajoutant l'argument **`sep=";"`**
par exemple : **`pd.read_csv("data.csv", sep=";")`**

→ Chargement et exploration des données

❑ Lecture de fichiers CSV ou Excel

Exemple

```
import pandas as pd
df = pd.read_csv("data.csv")
```

❑ Aperçu des données

Exemple

```
print(df.head())
print(df.info())
print(df.describe())
```

❑ Manipulation des index

Exemple

```
df.set_index("ID", inplace=True)
df.reset_index(inplace=True)
```

Remarque : Le séparateur par défaut utilisé par **`pandas.read_csv()`** est la ***virgule*** **,**.
Si votre fichier CSV utilise un autre séparateur (comme **`;`** ou **`|`**), il faut le préciser en ajoutant l'argument **`sep=";"`**
par exemple : **`pd.read_csv("data.csv", sep=";")`**

→ Chargement et exploration des données

- `pd.read_csv()` : Permet de lire un fichier CSV et de le convertir en DataFrame. Le fichier CSV contient des données que vous souhaitez manipuler dans Pandas.
- `df.head()` : Affiche les 5 premières lignes du DataFrame, ce qui vous permet de vérifier rapidement le contenu de votre fichier.
- `df.info()` : Affiche des informations essentielles sur les colonnes, les types de données, et le nombre de valeurs non-nulles dans chaque colonne.
- `df.describe()` : Affiche des statistiques descriptives pour les colonnes numériques, comme la moyenne, le minimum, le maximum et l'écart-type.
- `df.set_index("ID", inplace=True)` : Cette méthode permet de définir une colonne (ici ID) comme index du DataFrame. Cela peut être utile pour un meilleur accès aux données et pour des opérations plus rapides sur cette colonne.
- `df.reset_index(inplace=True)` : Cette méthode réinitialise l'index du DataFrame. Elle restaure la colonne d'origine (ici ID) comme une colonne régulière, tout en créant un nouvel index.

Manipulation avancée des données avec pandas

Exemple

```
print(df.head())
```

```
   ID      Nom  Age   Prix       Date  Ventes
0   1  Youssef   25  110.5  1/1/2023    10.0
1   2    Salma   33  200.0  1/2/2023    15.0
2   3    Hamza   28    NaN  1/3/2023    12.0
3   4    Nadia   42  150.0  1/4/2023    13.0
4   5   Rachid   27  185.5  1/5/2023     NaN
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ID      7 non-null      int64
 1   Nom     7 non-null      object
 2   Age     7 non-null      int64
 3   Prix    6 non-null      float64
 4   Date    7 non-null      object
 5   Ventes  6 non-null      float64
dtypes: float64(2), int64(2), object(2)
memory usage: 468.0+ bytes
None
```

```
print(df)
```

```
   ID      Nom  Age   Prix       Date  Ventes
0   1  Youssef   25  110.5  1/1/2023    10.0
1   2    Salma   33  200.0  1/2/2023    15.0
2   3    Hamza   28    NaN  1/3/2023    12.0
3   4    Nadia   42  150.0  1/4/2023    13.0
4   5   Rachid   27  185.5  1/5/2023     NaN
5   6    Laila   60  220.0  1/6/2023    17.0
6   2    Salma   33  200.0  1/2/2023    15.0
```

```
print(df.describe())
```

```
             ID        Age        Prix     Ventes
count  7.000000   7.000000    6.000000   6.000000
mean   3.285714  35.428571  177.666667  13.666667
std    1.799471  12.204605   40.318317   2.503331
min    1.000000  25.000000  110.500000  10.000000
25%    2.000000  27.500000  158.875000  12.250000
50%    3.000000  33.000000  192.750000  14.000000
75%    4.500000  37.500000  200.000000  15.000000
max    6.000000  60.000000  220.000000  17.000000
```

Exemple

```
df.set_index("ID", inplace=True)
print(df)
```

```
        Nom  Age   Prix      Date  Ventes
ID
1   Youssef   25  110.5  1/1/2023    10.0
2     Salma   33  200.0  1/2/2023    15.0
4     Nadia   42  150.0  1/4/2023    13.0
6     Laila   60  220.0  1/6/2023    17.0
2     Salma   33  200.0  1/2/2023    15.0
```

Le paramètre **`inplace=True`** dans `Pandas` signifie que l'opération sera effectuée directement sur l'objet d'origine sans créer de nouvelle copie.

➔ Autrement dit, les modifications sont appliquées sur le même DataFrame sans avoir à l'assigner à une nouvelle variable.

→ **Nettoyage des données**

❑ Détection des doublons

Exemple

```
df.drop_duplicates(inplace=True)
```

❑ Gestion des valeurs manquantes

Exemple

```
df.fillna(df.mean(), inplace=True)
```

```
df.dropna(inplace=True)
```

- **`df.dropna()`** : Supprime les lignes qui contiennent des valeurs manquantes (NaN).
- **`df.fillna()`** : Remplie les valeurs manquantes avec la moyenne des colonnes numériques, permettant de traiter les données incomplètes.
- **`df.drop_duplicates()`** : Supprime les lignes en double dans le DataFrame.

Exemple

```
df.drop_duplicates(inplace=True)
print(df)
```

```
   ID      Nom  Age   Prix       Date  Ventes
0   1  Youssef   25  110.5  1/1/2023    10.0
1   2    Salma   33  200.0  1/2/2023    15.0
2   3    Hamza   28    NaN  1/3/2023    12.0
3   4    Nadia   42  150.0  1/4/2023    13.0
4   5   Rachid   27  185.5  1/5/2023     NaN
5   6    Laila   60  220.0  1/6/2023    17.0
```

```
df.dropna(inplace=True)
print(df)
```

```
   ID      Nom  Age   Prix       Date  Ventes
0   1  Youssef   25  110.5  1/1/2023    10.0
1   2    Salma   33  200.0  1/2/2023    15.0
3   4    Nadia   42  150.0  1/4/2023    13.0
5   6    Laila   60  220.0  1/6/2023    17.0
```

Exemple

`select_dtypes(include=["number"])` : Sélectionne uniquement les colonnes de type numérique.

```
import pandas as pd
df = pd.read_csv("/content/data.csv")
print("Nombre de valeurs manquantes avant le nettoyage :")
print(df.isnull().sum())
df.fillna(df.select_dtypes(include=["number"]).mean(), inplace=True)
print("Nombre de valeurs manquantes après le nettoyage :")
print(df.isnull().sum())
```

```
Nombre de valeurs manquantes avant le nettoyage :
ID         0
Nom        0
Age        0
Prix       1
Date       0
Ventes     1
dtype: int64
Nombre de valeurs manquantes après le nettoyage :
ID         0
Nom        0
Age        0
Prix       0
Date       0
Ventes     0
dtype: int64
```

Nombre de valeurs manquantes par colonne

`mean()` : Calcule la moyenne de chaque colonne numérique.

→ Transformation et Fusion de Données

❑ Concaténation de DataFrames

- `concat()` permet de concaténer plusieurs DataFrames, soit horizontalement (ajout de colonnes), soit verticalement (ajout de lignes).

```
df_concat = pd.concat([df1, df2], axis=0, ignore_index=True)
df_concat = pd.concat([df1, df2], axis=1)
```

- **`axis=0`** : Concaténation verticale (ajouter des lignes).
- **`axis=1`** : Concaténation horizontale (ajouter des colonnes).
- **`ignore_index=True`** : Réinitialiser les index après la concaténation.

→ Transformation et Fusion de Données

❑ Fusion de DataFrames (Join/Merge)

- **`merge()`** permet de fusionner deux DataFrames sur une ou plusieurs colonnes en utilisant des méthodes de jointure (similaire à SQL).

```
# Fusionner deux DataFrames en utilisant une jointure interne sur la colonne 'ID'
df_merged = pd.merge(df1, df2, on="ID", how="inner")
```

- **`on="ID"`** : La colonne commune utilisée pour la fusion.
- **`how="inner"`** : Type de jointure (ici, "**`inner`**" signifie que seules les lignes avec des valeurs communes dans les deux DataFrames seront conservées).
  - **`how="left"`** : Jointure gauche, conserve toutes les lignes du DataFrame de gauche.
  - **`how="right"`** : Jointure droite, conserve toutes les lignes du DataFrame de droite.
  - **`how="outer"`** : Jointure extérieure, conserve toutes les lignes des deux DataFrames.

→ Transformation et Fusion de Données

❑ Transformation des colonnes

Parfois, il est nécessaire de transformer les données dans les colonnes pour les rendre plus utiles ou les nettoyer.

- **Créer une nouvelle colonne avec une transformation**

```
import pandas as pd
df = pd.read_csv("/content/data.csv")
# Créer une nouvelle colonne en appliquant une transformation sur une autre colonne
df['Prix_avec_TVA'] = df['Prix'] * 1.2
print(df)
```

```
   ID      Nom  Age   Prix      Date  Ventes  Prix_avec_TVA
0   1  Youssef   25  110.5  1/1/2023    10.0          132.6
1   2    Salma   33  200.0  1/2/2023    15.0          240.0
2   3    Hamza   28    NaN  1/3/2023    12.0            NaN
3   4    Nadia   42  150.0  1/4/2023    13.0          180.0
4   5   Rachid   27  185.5  1/5/2023     NaN          222.6
5   6    Laila   60  220.0  1/6/2023    17.0          264.0
6   2    Salma   33  200.0  1/2/2023    15.0          240.0
```

Cela ajoute une colonne Prix_avec_TVA qui représente le prix avec une TVA de 20%.

→ Transformation et Fusion de Données

❑ Transformation des colonnes

○ Appliquer une fonction à une colonne

```
# Appliquer une fonction lambda pour catégoriser l'âge
df['Age_category'] = df['Age'].apply(lambda x: 'Jeune' if x < 30 else 'Mature')
print(df)
```

	ID	Nom	Age	Prix	Date	Ventes	Prix_avec_TVA	Age_category
0	1	Youssef	25	110.5	1/1/2023	10.0	132.6	Jeune
1	2	Salma	33	200.0	1/2/2023	15.0	240.0	Mature
2	3	Hamza	28	NaN	1/3/2023	12.0	NaN	Jeune
3	4	Nadia	42	150.0	1/4/2023	13.0	180.0	Mature
4	5	Rachid	27	185.5	1/5/2023	NaN	222.6	Jeune
5	6	Laila	60	220.0	1/6/2023	17.0	264.0	Mature
6	2	Salma	33	200.0	1/2/2023	15.0	240.0	Mature

On crée une nouvelle colonne **`Age_category`** qui attribue la catégorie "**Jeune**" si l'âge est inférieur à 30 ans, sinon "**Mature**".

❑ Autre méthodes de pandas importantes

- `to_datetime()` : Convertir une colonne en format datetime.
- `astype()` : Convertir les types de données d'une colonne.
- `loc[]` : Accéder à un sous-ensemble du DataFrame par étiquette.
- `iloc[]` : Accéder à un sous-ensemble du DataFrame par position.
- `groupby()` : Grouper les données par une ou plusieurs colonnes et effectuer des calculs sur chaque groupe.
- `drop()` : supprime une colonne ou ligne.
- `rename()` : renomme les colonnes.

Exemple

```python
import pandas as pd

df = pd.read_csv("data.csv")
# Convertir la colonne Date en format datetime
df["Date"] = pd.to_datetime(df["Date"])
# Extraire l'année de la colonne "Date" et l'ajouter dans une nouvelle colonne "Année"
df["Année"] = df["Date"].dt.year

# Convertir le type de la colonne "Prix" en float
df["Prix"] = df["Prix"].astype(float)

print("\nAccès aux lignes où l'âge est supérieur à 30 :")
df_age_30 = df.loc[df["Age"] > 30]
print(df_age_30)

print("\nAccès aux 5 premières lignes :")
df_top_5 = df.iloc[:5]
print(df_top_5)

print("\nSomme des ventes par Client :")
df_grouped = df.groupby("Nom")["Ventes"].sum()
print(df_grouped)

# Supprimer la colonne "Age"
df.drop("Age", axis=1, inplace=True)
# Renommer la colonne "Nom" en "Client"
df.rename(columns={"Nom": "Client"}, inplace=True)
print(df)
```

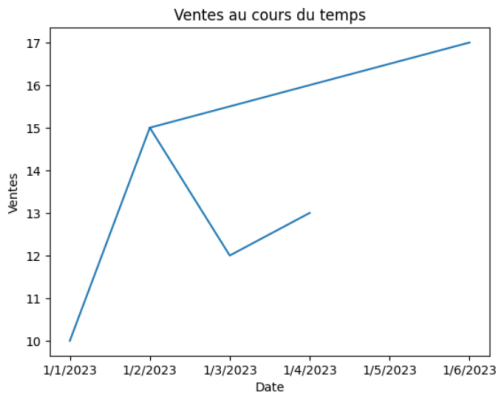
→ Graphiques avec Matplotlib

❑ Personnalisation des courbes

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

df = pd.read_csv("/content/data.csv")
# Créer un graphique linéaire avec Matplotlib
plt.plot(df['Date'], df['Ventes'])
plt.title("Ventes au cours du temps")
plt.xlabel("Date")
plt.ylabel("Ventes")
plt.show()
```

- `plt.plot()` : Crée un graphique linéaire montrant l'évolution des ventes au fil du temps.

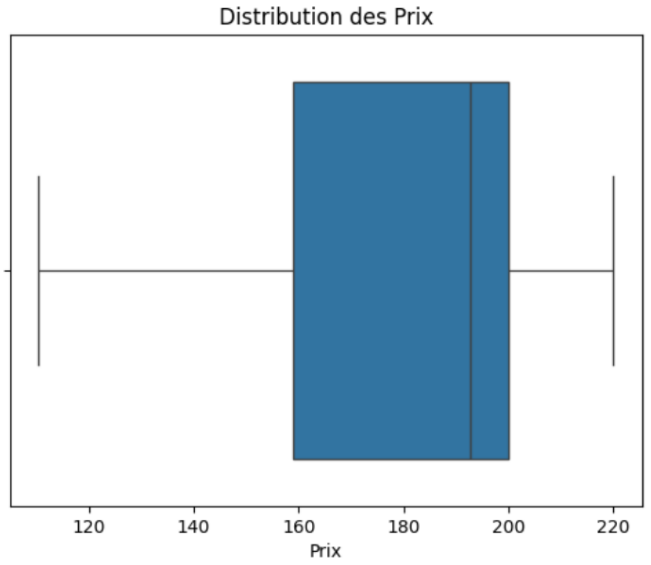


→ Visualisation avec Seaborn

❑ **Boxplot pour analyser la distribution des données :**

```
import seaborn as sns
# Créer un boxplot avec Seaborn
sns.boxplot(x=df['Prix'])
plt.title("Distribution des Prix")
plt.show()
```

`sns.boxplot()` : Crée un **`boxplot`** avec **`Seaborn`** pour visualiser la distribution des prix.



Traitement avancé des données avec NumPy

NumPy est une bibliothèque Python pour la manipulation de données numériques. Elle permet de travailler efficacement avec des tableaux multidimensionnels, offrant des fonctions mathématiques, logiques, de traitement de données et d'algèbre linéaire.

❑ Créer des tableaux avec NumPy

Un tableau NumPy est un objet similaire à une liste Python, mais optimisé pour les calculs mathématiques.

→ Création d'un tableau 1D (Vecteur)

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

```
[1 2 3 4 5]
```

→ Création d'un tableau 2D (Matrice)

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr_2d)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

❑ Opérations de base sur les tableaux

NumPy permet d'effectuer des opérations sur les tableaux de manière rapide et efficace.

→ Opérations élément par élément

Les opérations mathématiques entre les éléments d'un tableau NumPy sont très simples.

```
arr = np.array([1, 2, 3, 4, 5])
arr2 = np.array([5, 4, 3, 2, 1])
print("Addition: ",arr + arr2)
print("Multiplication: ",arr * arr2)
```

```
Addition:  [6 6 6 6 6]
Multiplication:  [5 8 9 8 5]
```

→ Opérations sur un tableau 2D

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print("Addition d'une constante")
print(arr_2d + 10)
print("\nMultiplication par une constante")
print(arr_2d * 2)
```

```
Addition d'une constante
[[11 12 13]
 [14 15 16]]

Multiplication par une constante
[[ 2  4  6]
 [ 8 10 12]]
```

❑ Indexation et slicing

On peut accéder aux éléments des tableaux NumPy de manière similaire aux listes Python, mais NumPy offre des capacités supplémentaires.

→ Indexation d'un tableau 1D

```
arr = np.array([10, 20, 30, 40, 50])

print("Accéder à un élément")
print(arr[2])
print("\nAccéder à un sous-tableau")
print(arr[1:4])
```

```
Accéder à un élément
30

Accéder à un sous-tableau
[20 30 40]
```

→ Indexation d'un tableau 2D

```
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])

print("Accéder à un élément (ligne 1, colonne 2)")
print(arr_2d[0, 1])

print("\nAccéder à une sous-matrice")
print(arr_2d[:2, 1:3])
```

```
Accéder à un élément (ligne 1, colonne 2)
2

Accéder à une sous-matrice
[[2 3]
 [5 6]]
```

❑ Fonctions utiles avec NumPy

NumPy propose une grande variété de fonctions pour effectuer des calculs sur les tableaux.

→ **`np.sum()`** : pour calculer la somme des éléments d'un tableau

```
arr = np.array([1, 2, 3, 4, 5])
print(np.sum(arr))
```

```
15
```

→ **`np.mean()`** : pour calculer la moyenne des éléments

```
arr = np.array([1, 2, 3, 4, 5])
print(np.mean(arr))
```

```
3.0
```

→ **`np.median()`** : pour calculer la médiane des éléments

```
arr = np.array([1, 2, 3, 4, 5])
print(np.median(arr))
```

```
3.0
```

❑ Fonctions utiles avec NumPy

→ `np.std()` : pour calculer l'écart-type

```
arr = np.array([1, 2, 3, 4, 5])
print(np.std(arr))
```

```
1.4142135623730951
```

❑ Reshaping des tableaux

NumPy permet de remodeler un tableau pour qu'il corresponde à une nouvelle forme (dimension).

→ Changer la forme d'un tableau

```
arr = np.array([1, 2, 3, 4, 5, 6])

# Remodeler un tableau 1D en tableau 2D
arr_reshaped = arr.reshape(2, 3)
print(arr_reshaped)
```

```
[[1 2 3]
 [4 5 6]]
```

❑ Générer des tableaux avec des fonctions NumPy

NumPy propose plusieurs fonctions pour créer des tableaux de différentes manières.

→ **`np.zeros(`*`shape, dtype=float`*`)`:** Crée un tableau rempli de zéros.

- ***`shape`*** : un entier ou un tuple d'entiers qui indique la forme (dimensions) du tableau. Exemple :
  `np.zeros(5)` crée un tableau 1D de 5 zéros.
  `np.zeros((3, 2))` crée un tableau 2D de 3 lignes et 2 colonnes.
- ***`dtype`*** (optionnel) : le type des données. Par défaut, float.

```
arr = np.zeros((2, 3))  # Tableau de 2x3 avec des zéros
print(arr)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

❑ Générer des tableaux avec des fonctions NumPy

→ ***np.ones(shape, dtype=float):*** Crée un tableau rempli de uns.

- ***shape*** : dimensions du tableau.
- ***dtype*** : type des données (par défaut float).

```
arr = np.ones((3, 2))  # Tableau de 3x2 avec des uns
print(arr)
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

→ **np.full(*shape, fill_value, dtype=None*):** Crée un tableau rempli d'une valeur donnée.

- ***shape*** : tuple ou entier → dimensions du tableau.
- ***fill_value*** : valeur avec laquelle remplir le tableau.
- ***dtype*** (optionnel) : type de la donnée.

```
arr = np.full((2, 2), 7)
print(arr)
```

```
[[7 7]
 [7 7]]
```

❑ Générer des tableaux avec des fonctions NumPy

→ **`np.arange(`***`start, stop, step, dtype=None`***`)`** : Crée un tableau 1D avec une séquence d'entiers ou de floats.

- ***`start`*** : valeur de départ.
- ***`stop`*** : valeur de fin (exclue).
- ***`step`*** : pas entre les valeurs.
- ***`dtype`*** (optionnel) : type des données.

```
arr = np.arange(0, 10, 2)
print(arr)
```

```
[0 2 4 6 8]
```

❑ Générer des tableaux avec des fonctions NumPy

→ **`np.linspace(`***`start, stop, num=50, endpoint=True`***`)`** : Crée un tableau 1D avec des valeurs réparties linéairement.

- ***`start`*** : valeur de départ.
- ***`stop`*** : valeur de fin.
- ***`num`*** : nombre total de valeurs.
- ***`endpoint`*** (bool) : si True (par défaut), inclut la valeur stop**.**

```
# Tableau de 5 valeurs espacées linéairement entre 0 et 10
arr = np.linspace(0, 10, 5)
print(arr)
```

```
[ 0.   2.5  5.   7.5 10. ]
```

❑ Générer des tableaux avec des fonctions NumPy

→ **`np.random.rand(d0, d1, ..., dn)`** : Génère un tableau avec des valeurs aléatoires entre 0 et 1 (distribution uniforme). *`d0, d1, ..., dn`* ➔ dimensions du tableau.

```
arr = np.random.rand(2, 3)
print(arr)
```

```
[[0.62321804 0.21035946 0.7791416 ]
 [0.20648241 0.2836284  0.44508456]]
```

→ **`np.random.randint(`***`low, high=None, size=None, dtype=int`***`)`** : Génère des entiers aléatoires dans un intervalle.

- ***`low`*** : borne inférieure (incluse).
- ***`high`*** : borne supérieure (exclue).
- ***`size`*** : forme du tableau.
- ***`dtype`*** : type des entiers (par défaut int).

```
arr = np.random.randint(10, 100, (2, 3))
print(arr)
```

```
[[79 77 22]
 [49 37 86]]
```

❑ Fusionner, empiler et diviser des tableaux

→ ***np.concatenate(..)*** **:** permet de fusionner ou empiler des tableaux le long d'un axe spécifié, à condition qu'ils aient la même forme sur les autres axes.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

arr_concat = np.concatenate((arr1, arr2))
print(arr_concat)
```

```
[1 2 3 4 5 6]
```

❑ Sauvegarder et charger des tableaux NumPy

NumPy permet de sauvegarder et de charger des tableaux facilement.

```
np.save('arr.npy', arr)
loaded_arr = np.load('arr.npy')
print(loaded_arr)
```

```
[10 20 30 10 20 30]
```

→ Documentation officielle de NumPy : Documentation NumPy