



TORNADE.IO
WEB - CYBERSÉCURITÉ



EMILIE

BOUT

Co-Fondateur de [tornade.io](#) et de [learning.tornade.io](#)

Docteur en cybersécurité à Tornade.io

emilie@tornade.io

Traitement de données & visualisation en Python

27/10/2025 - Next-U

Quel est l'usage principal de Pandas en Python ?

1. Créer des graphiques interactifs

2. Manipuler et analyser des données tabulaires

3. Se connecter à des bases de données SQL

4. Développer des applications web

Une `DataFrame` peut contenir à la fois des colonnes numériques et des colonnes textuelles.

1. Vrai

2. Faux

Lequel de ces graphiques est le plus adapté pour montrer l'évolution d'une variable dans le temps ?

1. Histogramme

2. Diagramme en barres

3. Courbe (line plot)

4. Carte

Python avancé pour la donnée.

Partie 1- 60 minutes



- Concevoir des scripts python pour manipuler la donnée.
- Lire, écrire des fichiers de formats variés (CSV,JSON)
- Nettoyer et filtrer des données avec Pandas

Pourquoi le traitement de donnée ?

1. Obtenir des informations exploitables

3. Automatiser des tâches

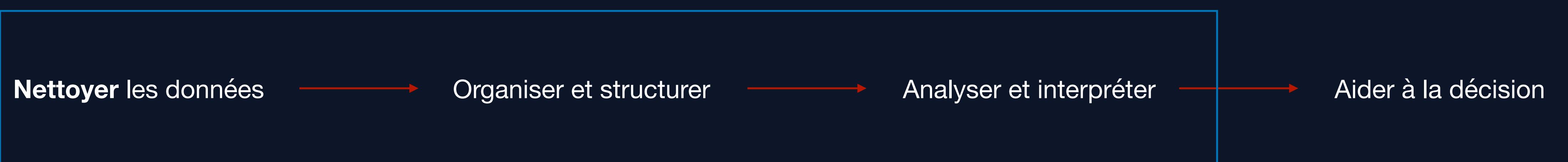
2. Aider à la prise de décision

4. Préparer les données pour l'IA / le machine learning

Pourquoi le traitement de donnée ?

“Les données brutes nourrissent le traitement,
le traitement nourrit l’analyse,
et l’analyse nourrit l’intelligence.”

Pourquoi le traitement de données ?



Pandas et NumPy : les fondations de l'analyse de données en Python

1. Introduction à Pandas

2. Manipuler des données avec Pandas

3. Manipuler des fichiers volumineux avec Pandas

Pandas

- **Pandas (Panel Data)** est une bibliothèque Python pour **analyser, nettoyer et manipuler des données**, surtout quand elles viennent de fichiers comme CSV, Excel, JSON, SQL, etc.
- Elle est très utilisée en **data science, finance, analyse métier**, etc.
- Open-Source
- Basée sur **Numpy** qui est aussi une bibliothèque mathématique permettant d'implémenter de façon efficace de l'algèbre linéaire et des calculs standards.

Cette bibliothèque permet de :

- Lire / écrire des fichiers (CSV, Excel, JSON...)
- Nettoyer les données (supprimer des lignes, renommer des colonnes...)
- Faire des statistiques (moyennes, sommes, regroupements...)
- Préparer la visualisation des données avec Matplotlib / Seaborn



Pandas et NumPy : les fondations de l'analyse de données en Python

1. Introduction à Pandas

2. Manipuler des données avec Pandas

3. Manipuler des fichiers volumineux avec Pandas

Structure de donnée: DataFrame

- **Pandas** utilise comme structure de donnée des Dataframe.
- Grosso-modo un **DataFrame**, c'est une **table en 2 dimensions** (comme une feuille Excel) de ce style où chaque **colonne** est une **Série Pandas** (type pd.Series), comme une liste de valeurs du même type.

Ici une ligne correspond à un enregistrement (ex. une transaction) et une colonne à un attribut / variable (ex. action,prix, quantité).

Tableau 1

action	quantite	prix
AAPL	10	145.0
TSLA	8	700.0
GOOGL	5	2800.0

Structure de donnée: DataFrame

- Création d'un DataFrame a partir d'une liste de dictionnaire.

Code

```
import pandas as pd

transactions = [
    {"action": "AAPL", "quantite": 10, "prix": 145.0},
    {"action": "GOOGL", "quantite": 5, "prix": 2800.0},
    {"action": "TSLA", "quantite": 8, "prix": 700.0} ]

df = pd.DataFrame(transactions)

print(df)
```

Résultat

	action	quantite	prix
0	AAPL	10	145.0
1	GOOGL	5	2800.0
2	TSLA	8	700.0

Structure de donnée: DataFrame

- Les fonctions de bases pour explorer un DataFrame

Tableau 1

Fonction	Description
<code>df.head(n)</code>	Affiche les n premières lignes (par défaut n=5)
<code>df.tail(n)</code>	Affiche les n dernières lignes
<code>df.shape</code>	Dimensions (lignes, colonnes)
<code><u>df.info()</u></code>	Infos sur colonnes, type de données, valeurs non-null
<code>df.describe()</code>	Statistiques descriptives pour colonnes numériques
<code>df.columns</code>	Liste des noms de colonnes
<code>df.index</code>	Indices des lignes

Structure de donnée: DataFrame

- Les fonctions de bases pour modifier les données

A) Ajouter une colonne

```
df["total"] = df["quantite"] * df["prix"]
```

B) Supprimer une colonne

```
df = df.drop(columns="total")
```

C) Renommer une colonne

```
df = df.rename(columns={"quantite": "volume"})
```

D) Supprimer des lignes

```
df = df.drop(index=1) # supprime ligne d'index 1
```

Structure de donnée: DataFrame

- **Exercice :** Crée un DataFrame `df` à partir des données suivantes :

Nom	Âge	Ville	Note
Alice.	25	Paris	15.5
Bob	30	Lyon	12.0
Clara	22	Marseille	17.0
David	28	Lille	13.5

Affiche :

- les **5 premières lignes**
- le **nombre de lignes et de colonnes**
- la **liste des colonnes**
- le **type de chaque colonne**

Structure de donnée: DataFrame

- **Correction :** Crée un DataFrame df à partir des données suivantes :

```
# 1. Création du DataFrame
data = {
    "Nom": ["Alice", "Bob", "Clara", "David"],
    "Âge": [25, 30, 22, 28],
    "Ville": ["Paris", "Lyon", "Marseille", "Lille"],
    "Note": [15.5, 12.0, 17.0, 13.5]
}
df = pd.DataFrame(data)
```

Affiche :

- les 5 premières lignes - `print(df.head())`
- le nombre de lignes et de colonnes. `print("Shape :", df.shape)`
- la liste des colonnes. `print("Colonnes :", df.columns.tolist())`
- le type de chaque colonne `print(df.info())`

Rappel les formats de fichiers:

CSV:

- **Comma-Separated Values** (valeurs séparées par des virgules).
- Fichier texte simple où chaque ligne représente un enregistrement et chaque valeur est séparée par un délimiteur (virgule, point-virgule, tabulation...).
- Facile à lire, compatible avec de nombreux logiciels, idéal pour les fichiers pas trop volumineux
- Pas adapté pour des structures complexes.
- Extension: .csv
-

nom,age,ville
Alice,30,Paris
Bob,25,Lyon

Rappel les formats de fichiers:

JSON:

- **JavaScript Object Notation (valeurs séparées par des virgules).**
- **Fichier texte qui stocke des données sous forme d'objets et de listes, idéal pour des données structurées ou hiérarchiques**
- **Gère facilement les données imbriquées (ex: commandes d'un client)**
- **Très utilisé pour les API et lisibles pour les machines et humains facilement**
- **Moins efficace que Parquet pour les fichiers volumineux mais mieux que CVS**
- **Peut consommer beaucoup de mémoire si mal géré**
- **Extension: .json**

```
[  
  
  {“nom”: “Alice”, “age”: 30, “ville”: “Paris”},  
  
  {“nom”: “Bob”, “age”: 25, “ville”: “Lyon”}  
  
]
```

Rappel les formats de fichiers:

Parquet:

- Parquet est un format binaire colonne-orienté conçu pour le stockage de données volumineux
- Optimisé pour l'écriture et la lecture rapide
- Très rapide sur les calculs de gros volumes
- Conserve les types de données.
- Moins lisible pour l'humain
- Gère facilement les données imbriquées (ex: commandes d'un client)
- Très utilisé pour les API et lisibles pour les machines et humains facilement
- Moins efficace que Parquet pour les fichiers volumineux mais mieux que CVS
- Peut consommer beaucoup de mémoire si mal géré

Les fichiers avec Pandas ...

1) Avant de charger : inspecter le fichier

- Vérifie le format : CSV / Excel / JSON / Parquet / compressé (.gz, .zip) ?
- Ouvre rapidement le fichier dans un éditeur texte si c'est possible pour voir le séparateur (;, ,, \t), s'il y a une ligne d'en-tête, s'il y a des métadonnées en début, encodage (UTF-8, ISO-8859-1), et exemples de valeurs manquantes (NA, -, ?, \N).
- Ça évite la plupart des erreurs de lecture (colonnes qui fusionnent, mauvaise encodage, etc.).

Les fichiers avec Pandas ...

- **Création d'un DataFrame a partir d'un fichier CSV et son exportation**

```
import pandas as pd  
  
df = pd.read_csv("transactions.csv")  
  
print(df)  
  
df.to_csv("transactions_clean.csv", index=False)
```

Structure de donnée: DataFrame

- Les fonctions de bases pour accéder au données

A) Sélection d'une colonne.

```
df["action"] # retourne une Series  
df.action    # même chose si nom de colonne simple
```

B) Sélection de plusieurs colonnes

```
df[["action", "prix"]]
```

C) Sélection par lignes

```
df.iloc[0]      # 1ère ligne par index  
df.iloc[0:2]    # 1ère et 2ème lignes
```

D) Filtrer avec conditions

```
df[df["prix"] > 500]  
df[(df["prix"] > 500) & (df["quantite"] > 5)]
```

Structure de donnée: DataFrame

- Les fonctions essentielles pour le traitement de données

Tableau 1

Fonction	Description
<code>df.dropna()</code>	Supprimer les lignes avec valeurs manquantes
<code>df.fillna(valeur)</code>	Remplacer les valeurs manquantes
<code>df.drop_duplicates()</code>	Supprimer les doublons
<code>df.astype(type)</code>	Changer le type d'une colonne
<code>df.replace(old, new)</code>	Remplacer certaines valeurs
<code>df.str.strip()</code>	Nettoyer les espaces en début/fin de texte (pour les colonnes string)
<code>df.str.lower()/str.upper()</code>	Normaliser les textes
<code>df.duplicated()</code>	Identifier les doublons
<code>df.sort_values(by="colonne")</code>	Trier par une colonne
<code>df.reset_index(drop=True)</code>	Réinitialiser les index
<code>df.astype(str).str.replace(...)</code>	Nettoyage avancé de chaînes

Pandas et Numpy

- **Sous le capot de Pandas c'est Numpy (Numerical Python) qui permet de stocker les données efficacement. Chaque colonne d'un DataFrame est en réalité un tableau NumPy (pd.Series) sous le capot. De ce fait tu peux appliquer toutes les fonctions de Numpy sur chaque pd.Series (colones) d'un DataFrame.**

```
print(np.sum(df["quantite"]))    # Affiche la somme des quantité  
  
print(np.mean(df["prix"]))       # Affiche la moyenne des prix  
  
print(np.max(df["prix"]))        # Affiche le prix maximum  
  
print(np.min(df["prix"]))        # Affiche le prix minimum
```

Exercice:

A partir d'un fichier CSV fictif contenant des transactions boursières réaliser les opérations suivantes:

- Charger le CSV dans un DataFrame.
- Afficher les 5 premières lignes et les infos générales (info()).
- Supprimer les doublons.
- Supprimer les lignes avec valeurs manquantes.
- Créer une colonne total = quantite * prix.
- Ajouter une colonne prix_log = log(prix) avec NumPy.
- Filtrer pour ne garder que les transactions où prix > 100.
- Trier le DataFrame par prix décroissant.
- Calculer le total des quantités par action.
- Exporter le DataFrame final nettoyé en CSV transactions_clean.csv

Identifier les problèmes courants

Corriger les problèmes classiques :

- doublons
- valeurs manquantes
- types de données incorrects
- incohérences simples
- Renommer des colonnes

Identifier les problèmes courants

Corriger les problèmes classiques :

- Doublons

```
df.duplicated().sum()
```

- Supprimer des doublons

```
df = df.drop_duplicates()
```

Identifier les problèmes courants

Corriger les problèmes classiques :

- Nombre de valeurs manquantes

```
df.isnull().sum()
```

- Remplir les valeurs manquantes (version texte)

```
df["ville"] = df["ville"].fillna("inconnue")
```

- Remplir les valeurs manquantes (version avec la dernière valeur listée)

```
df = df.fillna(method="ffill")
```

Identifier les problèmes courants

Corriger les problèmes classiques :

- Convertir une colonne en numérique

```
df["revenu"] = pd.to_numeric(df["revenu"], errors="coerce")
```

- Convertir une colonne en date

```
df["date_commande"] = pd.to_datetime(df["date_commande"], errors="coerce", dayfirst=True)
```

- Nettoyer du texte

```
df["ville"] = df["ville"].str.strip().str.lower()
```

Identifier les problèmes courants

Étape	Fonction principale	Objectif
Lecture du CSV	<code>pd.read_csv()</code>	Charger les données
Inspection	<code>head(), info(), isnull()</code>	Comprendre la structure
Doublons	<code>drop_duplicates()</code>	Éliminer les lignes en double
Valeurs manquantes	<code>fillna(), dropna()</code>	Corriger les trous
Conversion de types	<code>to_numeric(), to_datetime()</code>	Uniformiser les formats
Nettoyage texte	<code>.str.strip(), .str.lower()</code>	Standardiser les chaînes
Export	<code>to_csv(), to_parquet()</code>	Sauvegarder le dataset propre

Identifier les problèmes courants

Exercices: Charger et Nettoyer le fichier ventes.csv (Penser a visualiser en amont + doublons, conversion etc)

Correction: le fichier vente.py

Pandas et fichiers volumineux:

Un fichier volumineux :

- Quand on parle de gros fichiers, il s'agit de fichiers dont la taille dépasse la RAM disponible (ex: plusieurs Go).
- Pandas ne peut pas charger tout le fichier en mémoire d'un coup → risque de plantage ou ralentissement

Objectif : apprendre à traiter ces fichiers avec Pandas par morceaux et à optimiser la mémoire

Pandas et fichiers volumineux:

1- Les chunks :

- ***chunksize* permet de lire un fichier en morceaux (chunks) de taille définie.**
- **Chaque chunk est un DataFrame Pandas classique.**
- **On peut traiter chaque chunk indépendamment et agréger les résultats.**

```
import pandas as pd

chunksize = 100000 # 100k lignes par chunk

total_rows = 0

for chunk in pd.read_csv("gros_fichier.csv",
chunksize=chunksize):

    # Traitement du chunk
    total_rows += len(chunk)

print(f"Nombre total de lignes : {total_rows}")
```

Pandas et fichiers volumineux:

1- Les chunks :

- Exemple : somme d'une colonne

```
import pandas as pd
total = 0

for chunk in pd.read_csv("gros_fichier.csv", chunksize=50000, usecols=['quantite']):

    total += chunk['quantite'].sum()

print(f"Total : {total}")
```

Pandas et fichiers volumineux:

1- Les chunks :

- Exemple : filtrage

```
chunkszie = 50000

filtered_list = []

for chunk in pd.read_csv("gros_fichier.csv", chunkszie=chunkszie):

    filtered_chunk = chunk[chunk['ville'] == 'Paris']

    filtered_list.append(filtered_chunk)

df_filtered = pd.concat(filtered_list, ignore_index=True)
```

Pandas et fichiers volumineux:

2- Optimisation mémoire - Choisir les bon types :

- Les colonnes int64 → int32 ou int16
- Les colonnes float64 → float32
- Les colonnes textes avec peu de valeurs uniques → category

```
df = pd.read_csv("data.csv",
                 dtype={'col1':'int32','col2':'float32','col3':'category'})
```

Pandas et fichiers volumineux:

2- Optimisation mémoire - Charger seulement les colonnes utiles

```
df = pd.read_csv("data.csv", usecols=['col1','col2'])
```

Pandas et fichiers volumineux:

2- Optimisation mémoire - Libérer la mémoire

- `del df` supprime la **référence** à l'objet dans le namespace actuel (libérer un DataFrame qui n'est plus utile).

- Forcer le garbage collector

Python utilise un **garbage collector (GC)** pour gérer la mémoire automatiquement.

- Il supprime les objets **qui ne sont plus référencés**.
- Mais parfois, notamment avec des objets volumineux ou des cycles de références, il peut être utile de le forcer. Il parcourt les objets orphelins et les libère physiquement de la RAM.

```
import gc

del df      # supprime la référence

gc.collect() # force le garbage collector à libérer la
            # mémoire
```

Pandas et fichiers volumineux:

Bonnes pratiques:

- Toujours limiter les colonnes et lignes si possible.
- Utiliser chunksize pour les fichiers volumineux.
- Convertir les colonnes en types optimisés (int32, float32, category).
- Préférer les formats Parquet ou Feather pour la performance.
- Profiler la mémoire régulièrement :

```
print(df.memory_usage(deep=True))
```

Pandas et fichiers volumineux:

Exercices:

Calculer des statistiques et filtrer les données sans charger tout le fichier en mémoire.

- Lecture par chunks (100000) et afficher les 5 premières lignes de chaque chunk ainsi que leur taille
- Calculer la quantité totale vendue pour le produit « ProduitA » (Faire le calcul chunk par chunk et sommer le résultat)
Résultat attendu: 1673156
- Convertir les colonnes numériques en types optimisés (`int32, float32`)
- Supprimer les chunks après traitement pour éviter la saturation RAM
- Forcer le garbage collector

Pandas et fichiers volumineux:

Exercices:

Calculer des statistiques et filtrer les données sans charger tout le fichier en mémoire.

- **Lecture par chunks (100000) et afficher les 5 premières lignes de chaque chunk ainsi que leur taille**

```
import pandas as pd  
  
chunksize = 100_000  
  
for chunk in pd.read_csv("ventes_gros.csv", chunksize=chunksize):  
  
    print(len(chunk))  
  
    print(chunk.head())
```

Pandas et fichiers volumineux:

Exercices:

Calculer des statistiques et filtrer les données sans charger tout le fichier en mémoire.

- Calculer la quantité totale vendue pour le produit « ProduitA » (Faire le calcul chunk par chunk et sommer le résultat)

```
total_quantite = 0

for chunk in pd.read_csv("fichier_volumineux.csv", chunksize=chunksize,
usecols=['Produit','Quantité']):

    filtered = chunk[chunk['Produit'] == 'ProduitA']

    total_quantite += filtered['Quantité'].sum()

print(f"Quantité totale vendue de ProduitA : {total_quantite}")
```

Pandas et fichiers volumineux:

Exercices:

Calculer des statistiques et filtrer les données sans charger tout le fichier en mémoire.

- Convertir les colonnes numériques en types optimisés (`int32`, `float32`)

```
for chunk in pd.read_csv("fichier_volumineux.csv", chunksize=chunksizes):  
  
    print(chunk.memory_usage(deep=True))  
  
    chunk['Quantité'] = chunk['Quantité'].astype('int32')  
  
    chunk['Prix_Unitaire'] = chunk['Prix_Unitaire'].astype('float32')  
  
    print(chunk.memory_usage(deep=True))
```

Pandas et fichiers volumineux:

Exercices:

Calculer des statistiques et filtrer les données sans charger tout le fichier en mémoire.

- Supprimer les chunks après traitement pour éviter la saturation RAM
- Forcer le garbage collector

```
import gc

for chunk in pd.read_csv("fichier_volumineux.csv", chunksize=chunksize):

    # traitement
    del chunk
    gc.collect()
```

Visualisation des données avec Matplotlib

Partie 2- 30 -40 minutes



- Identifier les tendances, relations et pattern
- Déetecter les anomalies
- Transformer les chiffres en information visuelle claire

Pourquoi visualiser de la donnée?

1. Identifier les tendances et pattern

2. Détecter les anomalies

3. Communiquer clairement les résultats

Une Anomalie ?

Une **anomalie** (ou **outlier**) est une observation qui **s'écarte significativement** du comportement attendu d'une série de données.

Tableau 1

Type	Description	Exemple
Point isolé	Une valeur unique très différente des autres	Température 40°C alors que la moyenne est 15°C
Anomalie collective	Plusieurs points qui forment un motif inhabituel	Plusieurs jours consécutifs de pluie intense inhabituelle
Changement de tendance	Le comportement global change brusquement	Température qui chute soudainement de 20°C à 5°C
Saisonnalité inhabituelle	Variation qui ne correspond pas au cycle attendu	Pic de pluie en plein été dans une zone sèche

Une Anomalie ?

Pourquoi détecter des anomalies ?

- Identifier des **erreurs de mesure** ou de saisie
- Déetecter des **événements exceptionnels** (climatiques, économiques, industriels...)
- Améliorer la **qualité des données** pour le Machine Learning
- Déetecter des **fraudes** ou comportements inhabituels

Méthodes statistiques simples:

- Ecart Type une valeur est considérée comme anormale si elle est trop loin de la moyenne
- **Interquartile Range (IQR)** : basée sur le 1er et 3e quartile

```
moyenne = df["temperature"].mean()  
ecart_type = df["temperature"].std()  
anomalies = df[(df["temperature"] > moyenne + 3*ecart_type) |  
               (df["temperature"] < moyenne - 3*ecart_type)]
```

```
Q1 = df["temperature"].quantile(0.25)  
Q3 = df["temperature"].quantile(0.75)  
IQR = Q3 - Q1  
anomalies = df[(df["temperature"] < Q1 - 1.5*IQR) | (df["temperature"] > Q3 + 1.5*IQR)]
```

Visualiser des données

Matplotlib:

Matplotlib est une bibliothèque Python pour créer des graphes et visualisations 2D.

- Elle est très utilisée pour explorer les données, vérifier des tendances et communiquer les résultats.
- Le module principal pour tracer s'appelle pyplot.

```
import matplotlib.pyplot as plt
```

Les bases de Matplotlib

Création d'un graphique simple avec `plot()` pour des courbes :

```
plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.show()
```

Création d'un graphique en points avec `scatter()`:

```
plt.scatter([1, 2, 3, 4], [10, 20, 25, 30])
plt.show()
```

Les bases de Matplotlib

Création d'un graphique histogramme:

```
data = [1, 2, 2, 3, 3, 3, 4, 4, 5]
plt.hist(data, bins=5)
plt.show()
```

Création d'un graphique en bar:

```
x = ['A', 'B', 'C']
y = [5, 7, 3]
plt.bar(x, y)
plt.show()
```

Visualiser des données

Objectif :

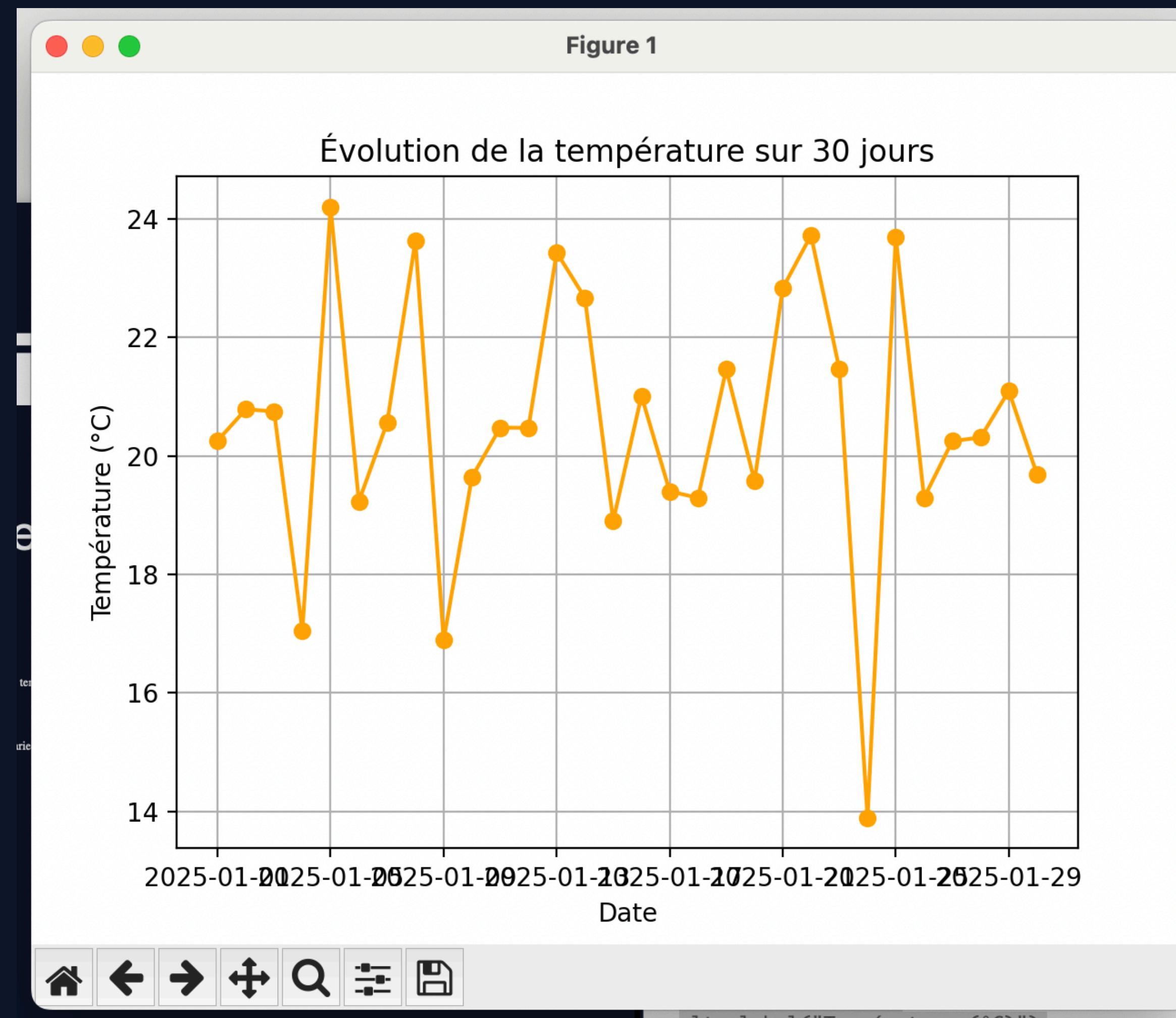
Apprendre à représenter des données qui évoluent dans le temps

Exemple :

Visualiser un DataFrame avec des dates et une valeur qui varie (ex. température).

```
# Création de données temporelles simulées  
  
dates = pd.date_range(start="2025-01-01", periods=30, freq="D")  
  
temperature = np.random.normal(20, 2, size=30) # température  
autour de 20°C  
  
df = pd.DataFrame({"date": dates, "temperature": temperature})  
  
# Tracé  
  
plt.plot(df["date"], df["temperature"], color="orange",  
marker="o")  
plt.title("Évolution de la température sur 30 jours")  
plt.xlabel("Date")  
plt.ylabel("Température (°C)")  
plt.grid(True)  
plt.show()
```

Visualiser des données



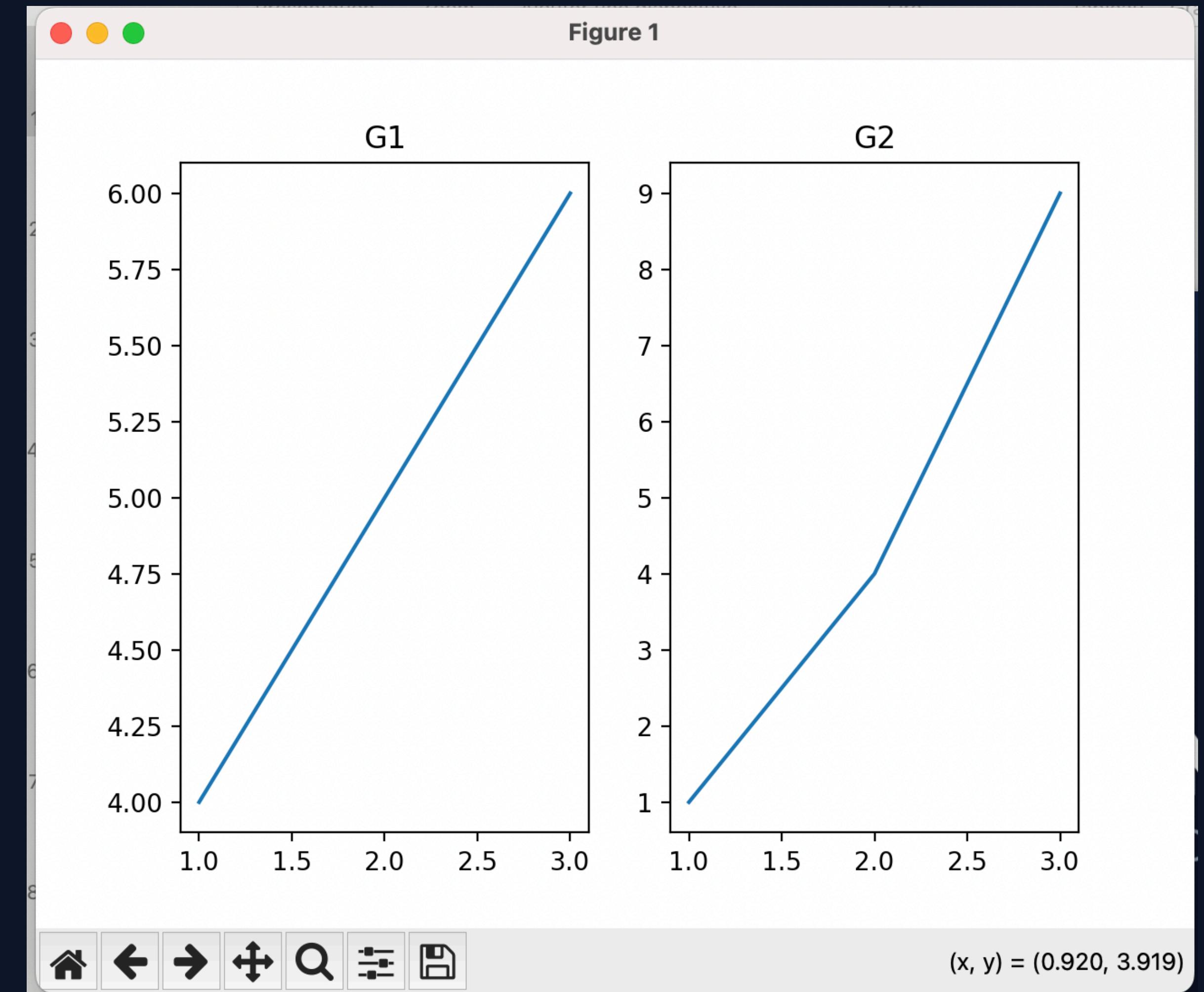
Créer des graphiques multiples

Objectif plusieurs graphiques sur la même figure:

```
plt.subplot(1,2,1) # 1 ligne, 2 colonnes, 1er
graphique
plt.plot([1,2,3],[4,5,6])
plt.title("G1")

plt.subplot(1,2,2)
plt.plot([1,2,3],[1,4,9])
plt.title("G2")

plt.show()
```



Créer des graphiques multiples

Objectif graphique simple.

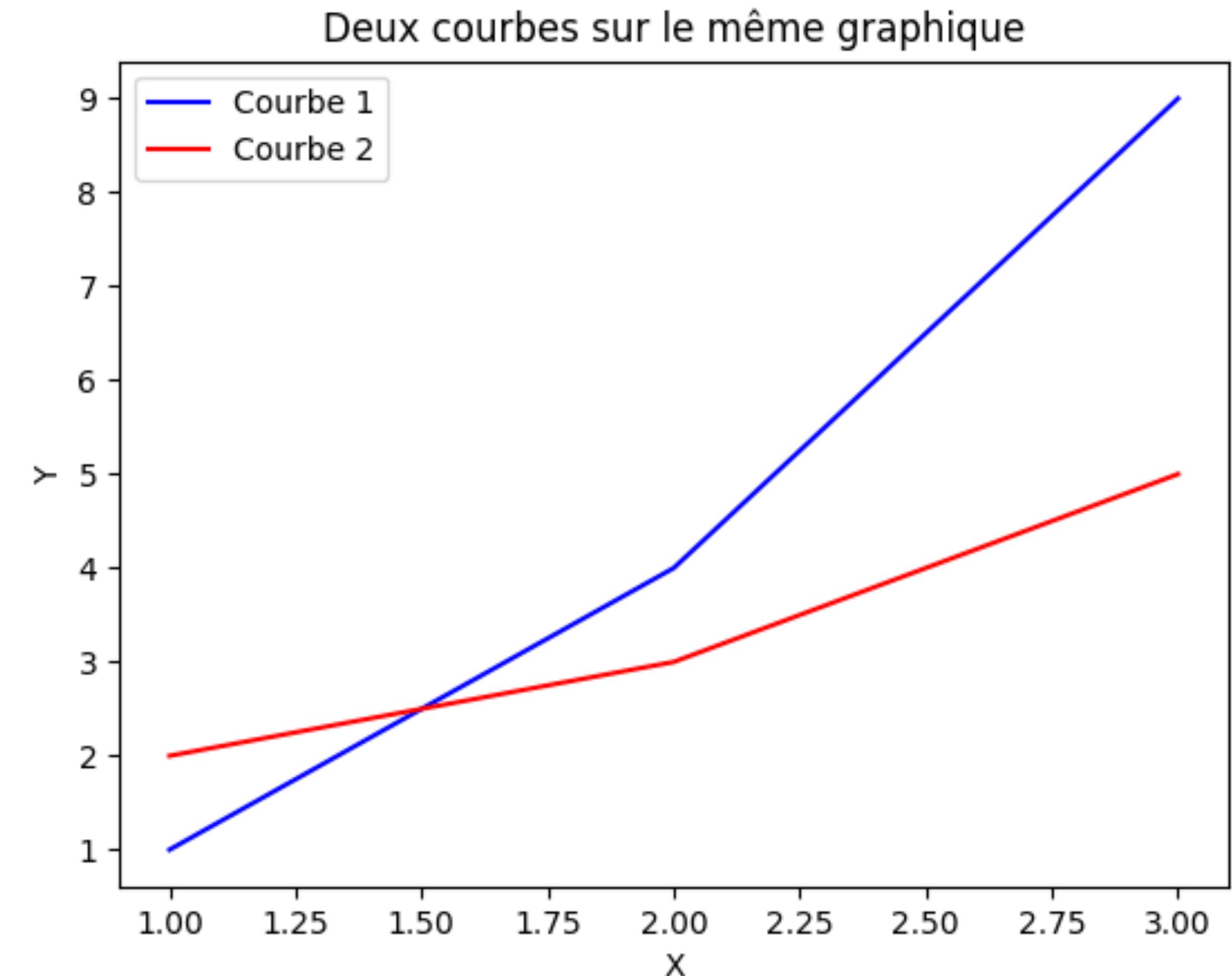
```
# Création de la figure et des axes
fig, ax = plt.subplots()

# Tracé de la première courbe
ax.plot([1, 2, 3], [1, 4, 9], label="Courbe 1",
color='blue')

# Tracé de la deuxième courbe
ax.plot([1, 2, 3], [2, 3, 5], label="Courbe 2",
color='red')

# Titre et légende
ax.set_title("Deux courbes sur le même graphique")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend() # Affiche la légende

# Affichage
plt.show()
```



Storytelling et UX en Data

Partie 2- 20 -30 minutes



- Comprendre le rôle du storytelling dans la visualisation de données.
- Créer des visualisations claires qui racontent une histoire.
- Identifier les bonnes pratiques pour capter l'attention et faciliter la prise de décision.

Introduction au storytelling

Le **storytelling** en data consiste à utiliser les données pour **raconter une histoire**

Pourquoi ?

- Informer : transmettre un message précis.
- Convaincre : soutenir une décision.
- Engager : capter l'attention de l'audience.

Exemple concret :

- Mauvais : “Les ventes ont augmenté de 5% ce mois-ci.”
- Bon : “Grâce à la nouvelle campagne marketing, nos ventes ont augmenté de 5%, avec un pic notable dans la région X.”

• Mauvais : “Les ventes ont augmenté de 5% ce mois-ci.”

Chaque graphique doit avoir un “**fil rouge**” qui guide le lecteur.

Astuces

Lisibilité ?

- Titre clair, explicite et orienté message.
- Axes bien labellisés, unités visibles.
- Couleurs contrastées et harmonieuses.

Simplicité:

- Éviter les graphiques surchargés (trop de séries ou d'éléments).
- Montrer uniquement les informations essentielles.
- Supprimer le superflu (“chartjunk”).

Navigation et hiérarchie

- Organiser les graphiques dans un ordre logique.
- Mettre en avant les KPI ou insights principaux.
- Utiliser des couleurs ou tailles pour hiérarchiser l'information.
- Mauvais : “Les ventes ont augmenté de 5% ce mois-ci.”

Accessibilité ?

- Contraste suffisant pour daltoniens.
- Police lisible.
- Eviter les abréviations obscures.

KPI ?

KPI = Key Performance Indicator.

- C'est un indicateur clé qui mesure la performance d'une activité ou d'un processus par rapport à un objectif.

Caractéristiques d'un bon KPI:

- Mesurable et quantifiable
- Pertinent par rapport à l'objectif
- Actionnable : il permet de prendre des décisions
- Comparatif dans le temps ou par segment

Exemples :

- Nombre de ventes par mois
- Taux de réussite d'une campagne de phishing
- Temps moyen de traitement d'un ticket client

Qu'est-ce qu'un insight ?

Définition : Un insight est une **conclusion ou découverte actionable** tirée de l'analyse des données.

Règles pour un bon insight :

- Basé sur des données fiables (KPI, métriques)
- Actionnable : il permet de prendre des décisions
- Contextualisé : replacé dans le temps, la zone, le segment

Exemple concret (Phishing) :

- KPI : Taux de réussite des tentatives de phishing = 25% pour les emails.
- Analyse : C'est plus élevé que pour les SMS (10%).
- Insight : Les emails représentent un risque majeur → prioriser les formations et alertes pour ce vecteur.

KPI et insight ?

Martine et le support client

Martine travaille au service support. Elle veut suivre la performance de son équipe.

Question : Quel KPI est le plus pertinent ?

- a) Le nombre total de tickets reçus
- b) Le temps moyen de résolution des tickets
- c) Le nombre d'emails envoyés aux clients
- d) Le nombre de cafés bus dans la journée

KPI et insight ?

Julien et la cybersécurité

Julien analyse les attaques de phishing dans son entreprise.

Question : Quel KPI est le plus utile pour suivre l'efficacité des actions de prévention ?

- a) Nombre total de mails reçus
- b) Nombre d'employés formés à la cybersécurité
- c) Taux de clic sur les liens de phishing lors des tests internes
- d) Taille moyenne des pièces jointes

KPI et insight ?

Sophie et le marketing

Sophie gère une campagne d'emailing.

Question : Quel KPI suivre pour mesurer le succès de la campagne ?

- a) Nombre d'emails envoyés
- b) Taux d'ouverture et taux de clics
- c) Nombre d'abonnés sur Instagram
- d) Temps passé à créer le visuel

MERCI !

FIN

emilie@TORNADE.IO



TORNADE.IO
WEB - CYBERSÉCURITÉ