

# Python pour la data

## Récapitulatif



# Contenu

---

<b>Récapitulatif de la partie 1 : introduction .....</b>	<b>9</b>
Principales définitions .....	9
Les principaux usages du Python .....	9
Fonctionnement des cellules dans Jupyter Notebook .....	10
Barre d'accès rapide Jupyter .....	10
<b>Récapitulatif de la partie 2 : Les bases de Python.....</b>	<b>11</b>
Déclaration de Variable .....	11
Règles de Nommage des Variables .....	11
Commentaires .....	11
Types de Variables de base.....	11
Opérateurs Mathématiques Basiques .....	11
Concaténation de Strings.....	11
Échapper un Caractère.....	12
Méthodes Courantes pour les Strings.....	12
<b>Récapitulatif de la partie 3 : Les conditions.....</b>	<b>13</b>
Variables de Type Booléen.....	13
Opérateurs de Comparaison .....	13
Conditions if, elif, else.....	13
Short Hand if ... else (Opérateurs ternaires).....	14
If Imbriqués (Nested if).....	14
<b>Récapitulatif de la partie 4.1 : Les listes .....</b>	<b>15</b>
Déclaration de Liste .....	15
Accès et Modification des Éléments d'une Liste .....	15
Slicing (Découpage) de Listes .....	15
Listes Multidimensionnelles .....	15
Méthodes des Listes .....	15
Quelques exemples pratiques (non exhaustifs).....	16
<b>Récapitulatif de la partie 4.2 : Les Tuples .....</b>	<b>17</b>
Définition et Particularité des Tuples .....	17
Déclaration d'un Tuple.....	17
Accès aux Éléments du Tuple .....	17
Méthodes liées aux Tuples.....	17
Concaténation de Tuples.....	17
Conversion Liste $\rightleftharpoons$ Tuple .....	17
Utilisation de la Méthode .zip().....	17
<b>Récapitulatif de la partie 4.3 : Les Sets .....</b>	<b>18</b>
Définition et Particularité des Sets .....	18
Déclaration d'un Set.....	18
Manipulation des Sets .....	18
Méthodes liées aux Sets.....	18
Conversion Liste $\rightleftharpoons$ Set .....	18
Opérations Ensemblistes .....	18
Sous-Ensembles et Sur-Ensembles.....	18
Les Frozensets .....	18

<b>Récapitulatif de la partie 5 : Les dictionnaires .....</b>	<b>19</b>
Définition et Particularité des Dictionnaires.....	19
Déclaration d'un Dictionnaire.....	19
Accès aux Données d'un Dictionnaire .....	19
Modification d'un Dictionnaire .....	19
Suppression des Valeurs d'un Dictionnaire .....	19
Les Dictionnaires Imbriqués .....	19
Autres Méthodes Liées aux Dictionnaires .....	19
<b>Récapitulatif de la partie 6 : Les boucles.....</b>	<b>20</b>
Boucle while .....	20
break, continue, else.....	20
Opérateurs de Réaffectation .....	20
Boucle for .....	20
Boucle for sur Liste, Tuple, Dictionnaire .....	20
Fonction range().....	20
Méthodes enumerate et zip .....	21
Compréhensions de Listes.....	21
<b>Récapitulatif de la partie 7 : Les fonctions.....</b>	<b>22</b>
Définition et Déclaration d'une Fonction .....	22
Paramètres de Fonctions .....	22
Return .....	22
Portées Globales et Locales des Variables.....	22
*args et **kwargs.....	22
Les Fonctions Lambda .....	23
Décorateurs .....	23
Générateurs .....	23
<b>Récapitulatif de la partie 8 : La programmation orientée objet .....</b>	<b>24</b>
Définition et Création de Classes .....	24
Création d'Attributs .....	24
Création et Appel de Méthodes .....	24
Création d'Objets.....	24
Encapsulation .....	24
Attributs Protégés .....	24
Héritage Simple & Multiple .....	25
Le Polymorphisme .....	25
<b>Récapitulatif de la partie 9 : Les packages .....</b>	<b>26</b>
Introduction à la Notion de Package .....	26
Importer un Module Spécifique d'un Package .....	26
Utiliser des Alias .....	26
PIP et ses Fonctionnalités .....	26
Conseils d'utilisation .....	26
Navigation et Gestion de Fichiers dans le terminal.....	26
Gérer Python dans le terminal .....	27
La librairie Sys .....	27
La librairie Os .....	28
La librairie Sys .....	28
<b>Récapitulatif de la partie 10 - Packages math et random .....</b>	<b>30</b>
Package Math .....	30

Package Random.....	30
<b>Récapitulatif de la partie 11 – Datetime.....</b>	<b>31</b>
Date.....	31
Time .....	31
Datetime .....	31
Extraction d'Éléments d'une Date .....	31
Timedelta.....	31
Timezone .....	31
Replace .....	31
Strftime et Strptime .....	31
ISO Format .....	32
Fromisoformat.....	32
Timestamp .....	32
<b>Récapitulatif de la partie 12 – Ouverture de documents .....</b>	<b>33</b>
Ouverture et Manipulation de Document .....	33
Manipulation JSON.....	33
Requests .....	33
CSV .....	34
<b>Récapitulatif de la partie 13 – Gestion des erreurs.....</b>	<b>35</b>
Différence entre Erreurs et Exceptions .....	35
Try et Except.....	35
Les Différents Except Possibles .....	35
Raise .....	35
Finally .....	35
<b>Récapitulatif de la partie 14 – Les algorithmes .....</b>	<b>36</b>
Définition d'un Algorithme.....	36
Importance des Algorithmes en Programmation .....	36
Différence entre Algorithme et Programme.....	36
Notion de Big O Notation.....	36
Importance de la Complexité dans le Choix d'un Algorithme.....	36
Algorithmes de Tri .....	36
Algorithmes de Recherche .....	36
<b>Récapitulatif de la partie 15 – Les matrices .....</b>	<b>37</b>
Définition.....	37
Éléments de Matrice .....	37
Taille de Matrice.....	37
Types Spécifiques de Matrices .....	37
Addition et Soustraction de Matrices .....	37
Multiplication Scalaire.....	37
Multiplication Matricielle .....	37
<b>Récapitulatif de la partie 16 – Numpy : création et exploration de tableaux.....</b>	<b>38</b>
Création d'un Array Numpy .....	38
Attributs d'un Array.....	38
Accès aux Éléments par Indexation .....	38
Accès aux Éléments par Slicing .....	38
Effectuer des boucles sur les arrays.....	38

<b>Récapitulatif de la partie 17 – Numpy : Opérations arithmétiques de base.....</b>	<b>39</b>
Opérations de Base sur les Matrices .....	39
Produit Matriciel .....	39
Fonctions d'Agrégation .....	39
Fonctions Mathématiques Universelles (ufuncs).....	39
<b>Récapitulatif de la partie 18 – Numpy : data cleaning et modeling .....</b>	<b>40</b>
Chargement de Données .....	40
Gestion des Données Manquantes .....	40
Manipulation d'Arrays.....	40
Types de Données.....	40
Exemple de type de tableaux : .....	41
<b>Récapitulatif de la partie 19 – Numpy : fonctionnalités avancées.....</b>	<b>42</b>
Génération de Nombres Aléatoires .....	42
Contrôle de la Reproductibilité.....	42
Nouvelle API de Génération de Nombres Aléatoires (rng = np.random.default_rng()) .....	42
<b>Récapitulatif de la partie 20 – Pandas : construire un dataframe .....</b>	<b>43</b>
Créer et consulter une Series Pandas .....	43
Créer et consulter un DataFrame .....	43
Gestion de l'index.....	43
Créer un DataFrame à partir d'un fichier CSV .....	43
Créer un DataFrame à partir d'un fichier Excel.....	43
Créer un DataFrame à partir d'un fichier JSON.....	43
Exporter des DataFrame en CSV/Excel/JSON .....	43
<b>Récapitulatif de la partie 21 – Pandas : naviguer dans un dataframe .....</b>	<b>44</b>
Sélection de Certaines Lignes et Colonnes.....	44
La méthode loc (Sélection basée sur le label) .....	44
La méthode iloc (Sélection basée sur la position) .....	44
Slicing.....	44
<b>Récapitulatif de la partie 21 – Pandas : Data Cleaning .....</b>	<b>45</b>
Première Analyse d'un DataFrame .....	45
Création de DataFrame avec Certaines Colonnes .....	45
Renommer des Colonnes.....	45
Gestion et Modification des Types de Colonnes.....	45
Gestion des Valeurs Manquantes .....	45
Supprimer les Doublons .....	45
Suppression de Lignes via Filtrage.....	45
Transformation avec .apply() .....	45
Gestion des Paramètres de Pandas.....	46
<b>Récapitulatif de la partie 22 – Pandas : Data Modeling .....</b>	<b>47</b>
Filtrer des Données.....	47
Trier les Données .....	47
Création de Nouvelles Colonnes .....	47
<b>Récapitulatif de la partie 23 – Pandas : Gestion des dates.....</b>	<b>48</b>
Conversion de Colonne String en Date .....	48
Différents Formats de Dates pour les Conversions.....	48
Gestion du Format Unix .....	48
Extraire Composants d'une Date .....	48



Calcul de la Durée Entre Deux Dates .....	48
Ajouter ou Soustraire du Temps sur une Date .....	48
Gestion des Fuseaux Horaires .....	48
<b>Récapitulatif de la partie 24 – Pandas : Analyse des données .....</b>	<b>49</b>
Opérations Statistiques de Base .....	49
Comptage et Valeurs Uniques .....	49
Mesures de Distribution .....	49
Groupement par Catégories .....	49
Table Pivot .....	49
Segmentation des Données avec cut et qcut .....	49
Gestion des Fuseaux Horaires .....	50
<b>Récapitulatif de la partie 25 – Pandas : Combiner plusieurs sources de données ..</b>	<b>51</b>
Concat .....	51
Join .....	51
Merge .....	51
<b>Récapitulatif de la partie 26 – Data visualisation : Matplotlib .....</b>	<b>52</b>
Théorie sur la data visualisation .....	52
Structure Basique d'une Figure Matplotlib .....	53
Création de Graphiques Linéaires .....	53
Personnalisation des Lignes (Couleur, Style, etc.) .....	53
Ajout de Titres, de Légendes et d'Étiquettes d'Axes .....	54
Construction de Diagrammes à Barres et Histogrammes .....	54
Création de Nuages de Points .....	54
Graphiques à Aires et Aires Empilées .....	54
<b>Récapitulatif de la partie 27 – Data visualisation : Seaborn .....</b>	<b>55</b>
Composants d'un Graphique avec Seaborn .....	55
Les Thèmes de Seaborn .....	55
Distribution / Boxplot / Violon .....	55
<b>Récapitulatif de la partie 27 – Data visualisation : plotly express .....</b>	<b>56</b>
Graphique Linéaire (Line Plot) .....	56
Scatter Plot .....	56
Pie Chart .....	56
Sunburst Chart .....	56
Treemap .....	56
px.scatter_matrix .....	56
Funnel .....	56
Animation Temporelle .....	56
<b>Récapitulatif de la partie 28 – Data visualisation : données géographiques .....</b>	<b>57</b>
Création d'une carte choroplèth .....	57
Code ISO Alpha-3 .....	57
Création d'un scatter plot géographique avec des codes ISO Alpha-3 .....	57
Projections de cartes courantes .....	57
Longitude et Latitude .....	57
Fichier GeoJSON .....	58
<b>Récapitulatif de la partie 29 – SQL et Python .....</b>	<b>59</b>
Le format SQLite .....	59
Se connecter à une base de données SQLite en Python .....	59

Récupérer les résultats d'une requête SQLite dans un DataFrame pandas .....	59
Construction d'une requête SELECT .....	59
Faire des filtres dans une requête SQL .....	59
Utilisation de LIMIT et ORDER BY dans une requête SQL.....	59
Réaliser des GROUP BY dans une requête SQL .....	59
HAVING .....	59
Les JOINTURES SQL .....	60
<b>Récapitulatif de la partie 29 – WebScraping : BeautifulSoup .....</b>	<b>61</b>
Importation des bibliothèques.....	61
Envoyer une requête HTTP .....	61
Obtenir le contenu brut de la réponse (HTML de la page) .....	61
Afficher le code de statut de la réponse http .....	61
Les erreurs http courantes .....	61
Parser une page web .....	62
Trouver des éléments dans le HTML .....	63
Accéder aux attributs d'une balise.....	63
Obtenir le texte à l'intérieur d'une balise.....	63
<b>Récapitulatif de la partie 30 – WebScraping : Selenium .....</b>	<b>64</b>
Importation des Bibliothèques.....	64
Initialiser le WebDriver .....	64
Navigation vers une URL .....	64
Attendre qu'un Élément soit Présent .....	64
Interactions avec les Éléments .....	64
Sélectionner des Éléments.....	64
Récupérer des Attributs .....	65
Mode Headless (Sans Interface Graphique) .....	65
<b>Récapitulatif de la partie 31 – Les API.....</b>	<b>66</b>
Envoi d'une Requête API .....	66
Vérification de la Réponse .....	66
Conversion des Données JSON en DataFrame .....	66
Rappels des principaux statuts de requêtes .....	66
Bonnes Pratiques pour Utiliser une API.....	66
Gestion des données JSON dans un DataFrame .....	67
<b>Récapitulatif de la partie 32 – statistique descriptive .....</b>	<b>68</b>
Types de Données.....	68
Mesures de Tendances.....	68
Mesures de Dispersion.....	68
Mesures de Forme .....	68
Relations entre Variables .....	68
<b>Récapitulatif de la partie 33 – Data science .....</b>	<b>70</b>
Définition de la Data Science .....	70
Types de Données.....	70
Types de Modèles de Machine Learning.....	70
Data Architecture .....	70
Rôles en Data Science.....	71
Exemples de modèles de Machine Learning.....	71
Importance des Métriques d'Erreur .....	71

<b>Récapitulatif de la partie 34 – Régression linéaire .....</b>	<b>72</b>
Régression Linéaire .....	72
Méthode des Moindres Carrés .....	72
Descente du Gradient .....	72
Métriques d'Erreur : MAE et RMSE .....	72
Stratégies de Séparation des Données .....	72
Standardisation des valeurs .....	72
Dilemme Biais-Variance .....	73
Solutions pour Éviter l'Overfitting .....	73
Transformation « Dummy » .....	73



## Récapitulatif de la partie 1 : introduction

---

### Principales définitions

- **Python** : Langage de programmation interprété et de haut niveau, Python est apprécié pour sa syntaxe simple et sa grande lisibilité, et il est utilisé dans divers domaines comme la science des données, le développement web, et l'automatisation.
- **Kernel** : Un kernel est un programme qui exécute le code contenu dans un notebook Jupyter, interprétant le code écrit dans celui-ci.
- **Paquets** : Les paquets sont des ensembles de modules Python qui offrent des fonctionnalités supplémentaires, installables et gérables via des gestionnaires de paquets comme pip.
- **Environnements** : Un environnement est un espace isolé possédant sa propre installation de Python et de paquets, permettant une gestion indépendante des configurations de projet.
- **Jupyter Notebook** : Jupyter Notebook est une application web permettant de créer et de partager des documents contenant du code interactif, des visualisations, et du texte narratif.
- **Anaconda** : Anaconda est une distribution open source des langages Python et R, principalement utilisée pour la science des données et le machine learning, facilitant la gestion des paquets et des environnements.

### Les principaux usages du Python

- **Développement Web** : Python permet de créer des sites web grâce à des frameworks comme Django et Flask, en offrant une syntaxe claire, une sécurité renforcée et une grande adaptabilité.
- **Analyse de données et science des données** : Python est privilégié pour l'analyse et la visualisation de données avec des bibliothèques telles que Pandas, NumPy, et Matplotlib, offrant un écosystème adapté aux grands ensembles de données et une intégration facile avec les bases de données.
- **Apprentissage automatique et intelligence artificielle** : Avec des outils comme TensorFlow, PyTorch, et Scikit-learn, Python est utilisé pour construire des modèles d'IA et de machine learning grâce à sa flexibilité, ses performances, et sa grande communauté de développeurs.
- **Automatisation et scripting** : Python est fréquemment employé pour écrire des scripts automatisant des tâches répétitives, bénéficiant d'une vaste bibliothèque standard, et d'une portabilité entre différentes plateformes.

## Fonctionnement des cellules dans Jupyter Notebook

- **Cellules de code** : C'est ici que vous écrivez et exécutez votre code. Les cellules de code ont une barre à gauche indiquant si elles ont été exécutées et, si oui, leur ordre d'exécution dans le notebook.
- **Cellules markdown** : Utilisées pour ajouter du texte formaté, des images, des liens et des listes. Le markdown est un langage de balisage léger qui vous permet de styliser votre texte facilement.

## Barre d'accès rapide Jupyter

- Save (Sauvegarder) : Sauvegarde rapidement le notebook.
- + (Ajouter une cellule) : Ajoute une nouvelle cellule sous la sélection.
- Cut (Couper) : Coupe la cellule sélectionnée.
- Copy (Copier) : Copie la cellule sélectionnée.
- Paste (Coller) : Colle la cellule coupée ou copiée.
- Up Arrow (Flèche vers le haut) : Déplace la cellule sélectionnée vers le haut.
- Down Arrow (Flèche vers le bas) : Déplace la cellule sélectionnée vers le bas.
- Run (Exécuter) : Exécute la cellule et passe à la suivante.
- Stop (Arrêter) : Arrête l'exécution de la cellule.
- Restart the Kernel and Clear All Outputs : Redémarre le kernel et efface les résultats.
- Cell Type Dropdown (Type de cellule) : Change le type de cellule (code, markdown, raw).

## Récapitulatif de la partie 2 : Les bases de Python

---

### Déclaration de Variable

- Syntaxe: `nom_variable = valeur`
- Exemple: `age = 21`

### Règles de Nommage des Variables

- Doit commencer par une lettre (a-z, A-Z) ou un underscore (`_`)
- Peut contenir des lettres, des chiffres (0-9), ou des underscores
- Ne peut pas commencer par un chiffre
- Ne doit pas contenir de caractères spéciaux (sauf `'_'`)
- Ne doit pas être un mot réservé en Python (comme `if`, `else`, `for`, etc.)

### Commentaires

Utilisés pour expliquer le code, rendre le code plus lisible et empêcher l'exécution de certaines lignes de code.

- Commentaire sur une ligne : Utilisez `#` au début de la ligne.
- Commentaires multilignes : Utilisez `"""` ou `'''` au début et à la fin du commentaire.

### Types de Variables de base

- `int`: Nombres entiers (ex: `5`, `-3`)
- `float`: Nombres à virgule (ex: `3.14`, `-0.001`)
- `str`: Chaînes de caractères (ex: `"Bonjour"`, `'Python'`)
- `bool`: booléen (ex: `True`, `False`)

Pour modifier le type d'une variable, vous pouvez utiliser les méthodes `int()`, `float()`, `str()`

### Opérateurs Mathématiques Basiques

- Addition: `+` (ex: `3 + 4` donne `7`)
- Soustraction: `-` (ex: `5 - 2` donne `3`)
- Multiplication: `*` (ex: `3 * 4` donne `12`)
- Division: `/` (ex: `8 / 4` donne `2.0`)
- Division Entière: `//` (ex: `8 // 3` donne `2`)
- Modulo (Reste de la Division): `%` (ex: `8 % 3` donne `2`)
- Exponentiation: `**` (ex: `2 ** 3` donne `8`)

### Concaténation de Strings

- Avec `+`: `"Hello" + " " + "World"` donne `"Hello World"`
- Avec Literals: `f"{prenom} {nom}"` (ex: `prenom = "John"`; `nom = "Doe"` donne `"John Doe"`)

## Échapper un Caractère

- Utilisez le backslash (\) pour échapper des caractères spéciaux dans une chaîne.
- Exemple: "Il dit: \"Bonjour\""

## Méthodes Courantes pour les Strings

- `len()`: Retourne la longueur d'une chaîne (ex: `len("Python")` donne 6)
- `lower()`: Convertit la chaîne en minuscules (ex: `"PYTHON".lower()` donne `"python"`)
- `upper()`: Convertit la chaîne en majuscules (ex: `"python".upper()` donne `"PYTHON"`)
- `strip()`: Enlève les espaces au début et à la fin (ex: `" python ".strip()` donne `"python"`)
- `replace(old, new)`: Remplace une sous-chaîne par une autre (ex: `"python".replace("y", "i")` donne `"pithon"`)
- `split(separator)`: Divise la chaîne en liste, en utilisant le séparateur (ex: `"a,b,c".split(",")` donne `['a', 'b', 'c']`)

## Récapitulatif de la partie 3 : Les conditions

### Variables de Type Booléen

Définition : Peuvent prendre deux valeurs : True ou False.

```
a = True
b = False
```

### Opérateurs de Comparaison

- Égal à (==) : Vérifie si deux valeurs sont égales.
- Différent de (!=) : Vérifie si deux valeurs ne sont pas égales.
- Supérieur à (>) : Vérifie si la valeur de gauche est plus grande que celle de droite.
- Inférieur à (<) : Vérifie si la valeur de gauche est plus petite que celle de droite.
- Supérieur ou égal à (>=) : Vérifie si la valeur de gauche est supérieure ou égale à celle de droite.
- Inférieur ou égal à (<=) : Vérifie si la valeur de gauche est inférieure ou égale à celle de droite.
- And (and) : Vérifie si deux conditions ou plus sont toutes vraies.
- Or (or) : Vérifie si au moins une parmi plusieurs conditions est vraie.
- Not (not) : Inverse l'état de la condition (rend True si la condition est False et vice-versa).

### Conditions if, elif, else

- if : Exécute le bloc de code si la condition est vraie.

```
if condition:
    # bloc de code
```

- elif : Exécute un bloc de code si la condition précédente n'est pas vraie et que cette condition est vraie.

```
if condition1:
    # bloc de code
elif condition2:
    # bloc de code
```

- else : Exécute un bloc de code si aucune des conditions précédentes n'est vraie.

```
if condition:
    # bloc de code
else:
    # bloc de code
```

### Short Hand if ... else (Opérateurs ternaires)

Permet d'écrire des conditions en une seule ligne.

Syntaxe :

```
variable = valeur1 if condition else valeur2
```

### If Imbriqués (Nested if)

Utilisez un if à l'intérieur d'un autre if pour tester des conditions supplémentaires.

Exemple :

```
if condition1:  
    if condition2:  
        # bloc de code
```



## Récapitulatif de la partie 4.1 : Les listes

---

### Déclaration de Liste

- Créer une liste : `ma_liste = [1, 2, 3]`
- Liste vide : `liste_vide = []`

### Accès et Modification des Éléments d'une Liste

- Accéder à un élément : `element = ma_liste[index]`
- Modifier un élément : `ma_liste[index] = nouvelle_valeur`

### Slicing (Découpage) de Listes

- Accéder à une sous-liste : `sous_liste = ma_liste[debut:fin]`

### Listes Multidimensionnelles

- Liste de listes : `liste_multi = [[1, 2], [3, 4]]`
- Accéder à un élément : `element = liste_multi[i][j]`

### Méthodes des Listes

- `append(element)` : Ajoute un élément à la fin.
- `insert(index, element)` : Insère un élément à la position spécifiée.
- `remove(element)` : Supprime la première occurrence de l'élément.
- `pop([index])` : Supprime et retourne l'élément à l'index (dernier par défaut).
- `del ma_liste[index]` : Supprime l'élément à l'index spécifié.
- `clear()` : Vide la liste.
- `len(ma_liste)` : Retourne le nombre d'éléments.
- `count(element)` : Compte les occurrences de l'élément.
- `sum(ma_liste)` : Retourne la somme des éléments.
- `index(element)` : Trouve l'index de la première occurrence de l'élément.
- `element in ma_liste` : Vérifie si l'élément est dans la liste.
- `sort()` : Trie la liste en place.
- `sorted(ma_liste)` : Retourne une nouvelle liste triée.
- `reverse()` : Inverse la liste en place.
- `copy()` : Crée une copie superficielle de la liste.

## Quelques exemples pratiques (non exhaustifs)

```
# Ajout et suppression
ma_liste.append(4)
ma_liste.insert(2, 5)
ma_liste.remove(5)
dernier = ma_liste.pop()

# Tri et inversion
ma_liste.sort()
ma_liste_inverse = sorted(ma_liste, reverse=True)
ma_liste.reverse()

# Copie
copie_liste = ma_liste.copy()
```

## Récapitulatif de la partie 4.2 : Les Tuples

---

### Définition et Particularité des Tuples

- Définition : Un tuple est une collection ordonnée et immuable en Python.
- Particularité : Immuabilité (les éléments d'un tuple ne peuvent pas être modifiés après sa création).

### Déclaration d'un Tuple

- Tuple avec Plusieurs Éléments : `mon_tuple = (1, 2, 3)`
- Tuple Vide : `tuple_vide = ()`
- Singleton (tuple à un élément) : `singleton = (1,)` (la virgule est nécessaire)

### Accès aux Éléments du Tuple

- Accès par Index : `element = mon_tuple[0]` (accède au premier élément)
- Slicing : `sous_tuple = mon_tuple[1:3]` (extrait une portion du tuple)

### Méthodes liées aux Tuples

- Longueur (len) : `longueur = len(mon_tuple)` (donne le nombre d'éléments dans le tuple)
- Somme (sum) : `somme = sum(mon_tuple)` (calcule la somme des éléments, si numériques)
- Minimum (min) : `minimum = min(mon_tuple)` (trouve l'élément le plus petit)
- Maximum (max) : `maximum = max(mon_tuple)` (trouve l'élément le plus grand)

### Concaténation de Tuples

- Concaténation : `tuple_concatene = tuple1 + tuple2` (crée un nouveau tuple en combinant deux tuples)

### Conversion Liste ⇌ Tuple

- Liste en Tuple : `mon_tuple = tuple(ma_liste)`
- Tuple en Liste : `ma_liste = list(mon_tuple)`

### Utilisation de la Méthode .zip()

- Zip : Combine plusieurs itérables (comme des listes ou des tuples) en une séquence de tuples.

```
noms = ["Alice", "Bob"]
ages = [25, 30]
combine = list(zip(noms, ages)) # [('Alice', 25), ('Bob', 30)]
```

## Récapitulatif de la partie 4.3 : Les Sets

---

### Définition et Particularité des Sets

- Définition : Un set est une collection non ordonnée d'éléments uniques en Python.
- Particularité : Les sets sont mutables, ne peuvent pas contenir de doublons et ne maintiennent pas un ordre spécifique des éléments.

### Déclaration d'un Set

- Création d'un Set : `mon_set = {1, 2, 3}`
- Set Vide : `set_vide = set()`

### Manipulation des Sets

- Ajout d'Éléments : `mon_set.add(4)`
- Suppression d'Éléments :
  - `mon_set.remove(2)` (lève une `KeyError` si l'élément n'existe pas)
  - `mon_set.discard(3)` (ne lève pas d'erreur si l'élément n'existe pas)

### Méthodes liées aux Sets

- Longueur (len) : `len(mon_set)` (nombre d'éléments dans le set)
- Somme (sum) : `sum(mon_set)` (somme des éléments, si numériques)
- Minimum (min) et Maximum (max) : `min(mon_set)` et `max(mon_set)`

### Conversion Liste ⇌ Set

- Liste en Set : `set(ma_liste)`
- Set en Liste : `list(mon_set)`

### Opérations Ensemblistes

- Union : `set1 | set2` ou `set1.union(set2)`
- Intersection : `set1 & set2` ou `set1.intersection(set2)`
- Différence : `set1 - set2` ou `set1.difference(set2)`
- Différence Symétrique : `set1 ^ set2` ou `set1.symmetric_difference(set2)`

### Sous-Ensembles et Sur-Ensembles

- Sous-Ensemble : `set1.issubset(set2)`
- Sur-Ensemble : `set1.issuperset(set2)`

### Les Frozensets

- Un frozenset est une version immuable d'un set.
- Création : `mon_frozenset = frozenset([1, 2, 3])`

## Récapitulatif de la partie 5 : Les dictionnaires

---

### Définition et Particularité des Dictionnaires

- Définition : Un dictionnaire est une collection non ordonnée, modifiable et indexée de paires clé-valeur.
- Particularité : Les clés sont uniques dans un dictionnaire et ne peuvent pas être dupliquées.

### Déclaration d'un Dictionnaire

- Dictionnaire Vide : `mon_dict = {}` ou `mon_dict = dict()`
- Dictionnaire avec Paires Clé-Valeur : `mon_dict = {"clé1": "valeur1", "clé2": "valeur2"}`

### Accès aux Données d'un Dictionnaire

- Accès à une Valeur : `valeur = mon_dict["clé"]`
- `keys()` : Retourne toutes les clés : `cles = mon_dict.keys()`
- `values()` : Retourne toutes les valeurs : `valeurs = mon_dict.values()`
- `items()` : Retourne toutes les paires clé-valeur : `paires = mon_dict.items()`

### Modification d'un Dictionnaire

- Ajout/Modification : `mon_dict["nouvelle_clé"] = "nouvelle_valeur"`
- Mise à Jour avec un Autre Dictionnaire : `mon_dict.update(autre_dict)`

### Suppression des Valeurs d'un Dictionnaire

- Supprimer une Clé : `del mon_dict["clé"]`
- Supprimer et Retourner une Clé : `valeur = mon_dict.pop("clé")`
- Vider le Dictionnaire : `mon_dict.clear()`

### Les Dictionnaires Imbriqués

- Accès et Manipulation : `valeur_interne = mon_dict["clé_ext"]["clé_int"]`

### Autres Méthodes Liées aux Dictionnaires

- `copy()` : Crée une copie superficielle : `copie = mon_dict.copy()`
- `in` : Vérifie l'existence d'une clé : `"clé" in mon_dict`
- `len()` : Nombre de paires clé-valeur : `taille = len(mon_dict)`
- `fromkeys()` : Crée un dictionnaire à partir d'un ensemble de clés avec une valeur par défaut : `dict.fromkeys(["clé1", "clé2"], "valeur_default")`

## Récapitulatif de la partie 6 : Les boucles

---

### Boucle while

- Définition : Répète un bloc de code tant qu'une condition est vraie.
- Déclaration :

```
while condition:  
    # instructions
```

### break, continue, else

- **break** : Sort de la boucle immédiatement.
- **continue** : Passe à l'itération suivante de la boucle.
- **else** : Bloc exécuté après la fin de la boucle, sauf si sorti par **break**.

### Opérateurs de Réaffectation

- **+=** : Ajoute et réaffecte (ex. **x += 1**).
- **-=** : Soustrait et réaffecte.
- **\*=** : Multiplie et réaffecte.
- **/=** : Divise et réaffecte.
- **%=** : Modulo et réaffecte.
- **//=** : Division entière et réaffecte.
- **\*\*=** : Élève à la puissance et réaffecte.

### Boucle for

- Définition : Itère sur les éléments d'une séquence.
- Déclaration :

```
for variable in sequence:  
    # instructions
```

### Boucle for sur Liste, Tuple, Dictionnaire

- Liste : for item in [1, 2, 3]:
- Tuple : for item in (1, 2, 3):
- Dictionnaire :
  - Clés : for key in dict:
  - Valeurs : for value in dict.values():
  - Clés et valeurs : for key, value in dict.items():

### Fonction range()

- Utilisation : Génère une séquence de nombres.
- Exemples :
  - range(fin) : De 0 à fin - 1.



- `range(début, fin)` : De début à fin - 1.
- `range(début, fin, pas)` : De début à fin - 1, avec un intervalle de pas.

### **Méthodes enumerate et zip**

- `enumerate` : Ajoute un compteur aux itérations.
  - Exemple : `for index, value in enumerate(sequence):`
- `zip` : Itère simultanément sur plusieurs séquences.
  - Exemple : `for item1, item2 in zip(sequence1, sequence2):`

### **Compréhensions de Listes**

- Définition : Syntaxe concise pour créer des listes.
- Syntaxe : `[expression for item in iterable if condition]`.
- Exemple : `[x*2 for x in range(5)]` -> `[0, 2, 4, 6, 8]`

## Récapitulatif de la partie 7 : Les fonctions

### Définition et Déclaration d'une Fonction

- Définition : Un bloc de code réutilisable conçu pour effectuer une tâche spécifique.
- Syntaxe :

```
def nom_fonction(param1, param2, ...):  
    # Bloc de code  
    return resultat # Optionnel
```

### Paramètres de Fonctions

- Obligatoires: Doivent être fournis lors de l'appel de la fonction.

```
def fonction(param_obligatoire):  
    # Bloc de code
```

- Optionnels: Ont des valeurs par défaut et ne sont pas obligatoires.

```
def fonction(param_optionnel=42):  
    # Bloc de code
```

### Return

- Utilisation: Pour renvoyer une valeur de la fonction à l'appelant.
- Syntaxe:

```
return valeur
```

### Portées Globales et Locales des Variables

- Globale: Définie en dehors de toute fonction, accessible partout dans le script.
- Locale: Définie à l'intérieur d'une fonction, accessible uniquement dans cette fonction.
- Accès Global dans une fonction:

```
global variable_globale # Référence une variable globale
```

### \*args et \*\*kwargs

- \*args: Pour un nombre variable d'arguments positionnels.

```
def fonction(*args):  
    # args est un tuple des arguments positionnels
```

- \*\*kwargs: Pour un nombre variable d'arguments nommés.

```
def fonction(**kwargs):  
    # kwargs est un dictionnaire des arguments nommés
```

## Les Fonctions Lambda

- Définition: Fonctions anonymes définies en une seule ligne.
- Syntaxe:

```
lambda arguments: expression
```

## Décorateurs

Un décorateur est une fonction qui prend une autre fonction en argument et étend ou modifie son comportement sans la modifier directement.

```
def mon_decorateur(fonction):  
    def enveloppe():  
        # Actions avant l'appel de la fonction  
        resultat = fonction()  
        # Actions après l'appel de la fonction  
        return resultat  
    return enveloppe  
  
@mon_decorateur  
def ma_fonction():  
    print("Ma fonction est exécutée")
```

## Générateurs

Un générateur est une fonction qui permet de générer une séquence de valeurs, permettant d'itérer sur elles une par une.

Utilisez le mot-clé `yield` au lieu de `return` dans une fonction.

```
def mon_generateur():  
    yield 'a'  
    yield 'b'  
    yield 'c'  
  
for valeur in mon_generateur():  
    print(valeur)
```

## Récapitulatif de la partie 8 : La programmation orientée objet

### Définition et Création de Classes

- Définition : Modèle pour créer des objets ayant des attributs et méthodes communs.
- Syntaxe :

```
class MaClasse:  
    pass
```

### Création d'Attributs

- Attributs d'Instance : Uniques à chaque instance. Définis dans `__init__` avec `self`.

```
def __init__(self, attribut1):  
    self.attribut1 = attribut1
```

- Attributs de Classe : Partagés entre toutes les instances de la classe.

```
attribut_de_classe = "valeur"
```

### Création et Appel de Méthodes

- Méthodes d'Instance : Prend `self` comme premier argument. Opère sur des attributs d'instance.

```
def ma_methode(self):  
    print(self.attribut1)
```

- Méthodes de Classe : Décorateur `@classmethod`. Prend `cls` comme premier argument. Opère sur des attributs de classe.

```
@classmethod  
def ma_methode_de_classe(cls):  
    print(cls.attribut_de_classe)
```

### Création d'Objets

- Instanciation d'une classe pour créer un objet unique.

```
mon_objet = MaClasse()
```

### Encapsulation

- Objectif : Cacher certains détails internes de l'objet et exposer uniquement les méthodes nécessaires pour l'utilisation de cet objet.
- Accès : Utiliser des méthodes publiques pour accéder/modifier des attributs privés ou protégés.

### Attributs Protégés

- Précédés d'un underscore `_`. Convention pour indiquer qu'ils doivent être utilisés avec prudence.

```
self._attribut_protege = "valeur"
```

## Héritage Simple & Multiple

- Simple : Une classe enfant hérite d'une seule classe parent.

```
class Enfant(Parent):  
    pass
```

- Multiple : Une classe enfant hérite de plusieurs classes parents.

```
class Enfant(Parent1, Parent2):  
    pass
```

## Le Polymorphisme

- Permet à des méthodes de se comporter différemment en fonction de l'objet qui les appelle.
- Exemple : Redéfinir une méthode dans une classe enfant.

```
class Parent:  
    def ma_methode(self):  
        print("Appel depuis Parent")  
  
class Enfant(Parent):  
    def ma_methode(self):  
        print("Appel depuis Enfant")
```

## Récapitulatif de la partie 9 : Les packages

---

### Introduction à la Notion de Package

Un package en Python est un dossier qui contient des modules Python. Ces modules peuvent être des fichiers .py contenant des définitions de fonctions, de classes et de variables, ainsi que des instructions exécutables. Importer un package signifie rendre ces modules ou fonctions disponibles dans votre espace de travail actuel.

```
import math
```

### Importer un Module Spécifique d'un Package

```
from math import sqrt, cos, sin
```

### Utiliser des Alias

```
import numpy as np  
from math import sqrt as racine
```

### PIP et ses Fonctionnalités

- Lister les packages installés : `PIP list`
- Afficher les informations d'un package : `pip show nom_du_package`
- Installer un package : `pip install nom_du_package`
- Mettre à jour un package : `pip install --upgrade nom_du_package`
- Désinstaller un package : `pip uninstall nom_du_package`

### Conseils d'utilisation

- Environnements Virtuels : utiliser des environnements virtuels pour isoler les dépendances des projets.
- Gestion des Versions : Soyez attentifs aux versions des packages lors de l'installation pour éviter les incompatibilités.

### Navigation et Gestion de Fichiers dans le terminal

- Afficher le Répertoire Actuel :
  - Linux/MacOS : `pwd`
  - Windows : `cd`
- Lister les Fichiers et Dossiers :
  - Linux/MacOS : `ls`
  - Windows : `dir`
- Changer de Répertoire : `cd chemin/vers/le/dossier`
- Revenir au dossier parent : `cd ..`
- Créer un Dossier :
  - Linux/MacOS : `mkdir nom_du_dossier`



- Windows : md nom\_du\_dossier
- Créer un Fichier :
  - Linux/MacOS : touch nom\_du\_fichier.py
  - Windows : echo.> nom\_du\_fichier.py.
- Supprimer un Fichier :
  - Linux/MacOS : rm nom\_du\_fichier.py
  - Windows : del nom\_du\_fichier.py
- Renommer ou Déplacer un Fichier :
  - Linux/MacOS : mv ancien\_nom.py nouveau\_nom.py
  - Windows : rename ancien\_nom.py nouveau\_nom.py ou move ancien\_nom.py nouveau\_nom.py

## Gérer Python dans le terminal

- Exécuter un Script Python : python3 nom\_du\_script.py
- Pour obtenir de l'aide sur Python depuis le terminal : python3 -h
- Interpréteur Python Interactif :
  - Lancer l'interpréteur interactif : python3
  - Quitter l'interpréteur : exit() ou Ctrl+D (Linux/MacOS), Ctrl+Z puis Enter (Windows)

## La librairie Sys

- Accès aux Arguments de Ligne de Commande
  - sys.argv : Liste des arguments passés au script, où sys.argv[0] est le nom du script.

```
import sys
script_name = sys.argv[0]
first_argument = sys.argv[1] if len(sys.argv) > 1 else None
```

- Manipulation du Chemin de Recherche de Modules
  - sys.path : Liste des répertoires où Python cherche les modules lors de l'importation.

```
import sys
sys.path.append('/chemin/vers/mon/module')
```

- Sortie du Script
  - sys.exit() : Quitte le script Python avec un statut donné. sys.exit(0) pour une sortie sans erreur.

```
import sys
sys.exit("Erreur détectée, arrêt du script.")
```

- Version de Python
  - `sys.version` : Chaîne contenant la version de Python en cours d'utilisation.

```
import sys
print(sys.version)
```

## La librairie Os

### Navigation et Manipulation des Répertoires

- `os.getcwd()` : Retourne le répertoire de travail actuel.
- `os.chdir('/nouveau/chemin')` : Change le répertoire de travail actuel.

### Lister les Contenus d'un Répertoire

- `os.listdir('/chemin/optionnel')` : Liste les fichiers et dossiers dans le répertoire donné ou actuel.

### Création et Suppression de Dossiers

- `os.mkdir('monDossier')` : Crée un nouveau dossier.
- `os.rmdir('monDossier')` : Supprime le dossier spécifié.

### Exécution de Commandes du Système

- `os.system('commande')` : Exécute la commande spécifiée dans le shell du système.

### Manipulation des Variables d'Environnement

- `os.environ` : Un dictionnaire contenant les variables d'environnement. Utilisez `os.environ.get('NOM_VAR')` pour accéder en toute sécurité à une variable d'environnement.

## La librairie Sys

### `sys.argv`

Accéder aux arguments passés à un script Python via la ligne de commande.

```
import sys
print(sys.argv[1:])
```

### `if name == "main"`

Vérifie si le script est exécuté directement (et non importé comme module).

```
if __name__ == "__main__":
    # Le code ici ne s'exécute que si le script est exécuté directement.
```

## **sys.version**

Obtenir la version de l'interpréteur Python en cours d'utilisation.

## **sys.path**

Liste des répertoires dans lesquels Python recherche des modules à importer.

```
import sys
print(sys.path) # Affiche les chemins de recherche de modules.
sys.path.append('/chemin/vers/module/')
```

## **sys.getdefaultencoding()**

Obtenir l'encodage par défaut utilisé par l'interpréteur Python pour convertir les chaînes de caractères.

## Récapitulatif de la partie 10 - Packages math et random

---

### Package Math

- `math.ceil(x)` : Arrondit  $x$  à l'entier supérieur le plus proche.
- `math.floor(x)` : Arrondit  $x$  à l'entier inférieur le plus proche.
- `math.trunc(x)` : Tronque  $x$  en retirant la partie décimale.
- `math.pow(x, y)` : Calcule  $x$  élevé à la puissance  $y$ .
- `math.sqrt(x)` : Calcule la racine carrée de  $x$
- `math.pi` : Constante représentant la valeur de  $\pi$
- `math.inf` : Représente l'infini positif.

### Package Random

- `random.randint(a, b)` : Génère un entier aléatoire  $N$  tel que  $a \leq N \leq b$
- `random.random()` : Génère un flottant aléatoire entre 0.0 et 1.0.
- `random.choice(seq)` : Sélectionne un élément aléatoire d'une séquence non vide.
- `random.shuffle(seq)` : Mélange les éléments d'une liste en place
- `random.sample(population, k)` : Retourne une liste de  $k$  éléments uniques choisis parmi la population
- `random.seed(a=None)` : Initialise le générateur de nombres aléatoires avec une graine  $a$ .

## Récapitulatif de la partie 11 – Datetime

---

### Date

- Création : `datetime.date(année, mois, jour)`
- Aujourd'hui : `datetime.date.today()`

### Time

Création : `datetime.time(heure, minute, seconde, microseconde)`

### Datetime

- Création : `datetime.datetime(année, mois, jour, heure, minute, seconde, microseconde)`
- Maintenant : `datetime.datetime.now()`
- UTC maintenant : `datetime.datetime.utcnow()`

### Extraction d'Éléments d'une Date

- Année : `ma_date.year`
- Mois : `ma_date.month`
- Jour : `ma_date.day`
- Heure : `mon_datetime.hour`
- Minute : `mon_datetime.minute`
- Seconde : `mon_datetime.second`

### Timedelta

- Addition/Soustraction de jours : `ma_date + datetime.timedelta(days=10)`
- Addition/Soustraction de secondes : `mon_datetime + datetime.timedelta(seconds=30)`

### Timezone

Spécifier un fuseau horaire : `datetime.datetime.now(datetime.timezone.utc)`

### Replace

Modifier des éléments : `ma_date.replace(year=2024, month=12)`

### Strftime et Strptime

- Formatage en chaîne : `ma_date.strftime("%Y-%m-%d")`
- Conversion de chaîne en date : `datetime.datetime.strptime("2024-12-31", "%Y-%m-%d")`

## ISO Format

- Date en ISO 8601 : `ma_date.isoformat()`
- Datetime en ISO 8601 : `mon_datetime.isoformat()`

## Fromisoformat

Créer date/datetime à partir de chaîne ISO 8601 : `datetime.date.fromisoformat('2024-12-31')`

## Timestamp

- Obtenir un timestamp : `mon_datetime.timestamp()`
- Créer datetime à partir d'un timestamp :  
`datetime.datetime.fromtimestamp(1609459200)`



## Récapitulatif de la partie 12 – Ouverture de documents

---

### Ouverture et Manipulation de Document

- Ouverture de Fichier
  - Lecture: with open('fichier.txt', 'r') as f:
  - Écriture: with open('fichier.txt', 'w') as f:
  - Ajout: with open('fichier.txt', 'a') as f:
- Lecture de Contenu
  - Tout lire: contenu = f.read()
  - Lire une ligne: ligne = f.readline()
  - Lire toutes les lignes: lignes = f.readlines()
  - Boucler sur chaque ligne: for ligne in f:
- Suppression de Document
  - `os.remove('fichier.txt')`

### Manipulation JSON

- Équivalences JSON/Python
  - Objet JSON → Dictionnaire Python
  - Tableau JSON → Liste Python
  - String JSON → Chaîne de caractères Python
  - Nombre JSON → Int/Float Python
  - `true/false` JSON → `True/False` Python
  - `null` JSON → `None` Python
- Lecture et Écriture JSON
  - Lire JSON: `data = json.load(f)`
  - Écrire JSON (objet → fichier): `json.dump(data, f, indent=4)`
  - Convertir objet Python en chaîne JSON: `chaine = json.dumps(data, indent=4)`

### Requests

- Envoyer une Requête GET
  - `reponse = requests.get('https://exemple.com')`
  - Accéder au contenu: `contenu = reponse.text` ou `response.content`

## CSV

- Lire un Fichier CSV

```
with open('fichier.csv', 'r') as f:
    lecteur = csv.reader(f)
    en_tete = next(lecteur)
    for ligne in lecteur:
        print(ligne)
```

- Écrire dans un Fichier CSV

- Écrire une ligne:

```
with open('fichier.csv', 'w', newline='') as f:
    ecrivain = csv.writer(f)
    ecrivain.writerow(['Col1', 'Col2', 'Col3'])
```

- Écrire plusieurs lignes:

```
with open('fichier.csv', 'w', newline='') as f:
    ecrivain = csv.writer(f)
    ecrivain.writerows([[ 'Val1', 'Val2', 'Val3'], [ 'Val4', 'Val5',
'Val6']])
```

- Isoler l'Entête des Colonnes

- Utiliser `next(lecteur)` pour passer la première ligne d'en-tête lors de la lecture.

## Récapitulatif de la partie 13 – Gestion des erreurs

### Différence entre Erreurs et Exceptions

- Erreurs de Syntaxe : Problèmes dans le code qui empêchent Python de l'interpréter correctement. Exemples : oublis de deux-points, indentation incorrecte.
- Exceptions : Erreurs détectées pendant l'exécution du programme, même si la syntaxe est correcte. Peuvent être gérées et récupérées.

### Try et Except

- try : Bloc de code à surveiller pour les exceptions. Si une exception survient, le flux est transféré au bloc except.

```
try:  
    # Code qui pourrait lever une exception
```

- except : Spécifie un type d'exception à capturer et à gérer.

```
except TypeError:  
    # Traitement spécifique pour TypeError  
except Exception as e:  
    # Traitement pour toutes les autres exceptions capturées
```

### Les Différents Except Possibles

- Spécifiques : Capturer des types d'exceptions précis (e.g., ValueError, ZeroDivisionError).
- Générique : Utiliser except Exception as e pour capturer toute exception non capturée par les blocs except précédents.
- Tout Capturer : except: sans spécifier un type capture toutes les exceptions, y compris les exceptions système. À utiliser avec prudence.

### Raise

- Lever des Exceptions : Utilisez raise pour lever intentionnellement une exception dans votre code, soit pour signaler une erreur, soit pour valider les entrées.

```
if x < 0:  
    raise ValueError("x ne peut pas être négatif")
```

### Finally

- Bloc finally : S'exécute toujours après les blocs try et except, indépendamment de l'occurrence d'une exception. Utile pour le nettoyage et la libération de ressources.

```
finally:  
    # Code de nettoyage, exécuté quelle que soit l'issue du bloc try
```

## Récapitulatif de la partie 14 – Les algorithmes

---

### Définition d'un Algorithme

Algorithme : Une suite finie et non ambiguë d'instructions pour résoudre un problème ou accomplir une tâche.

### Importance des Algorithmes en Programmation

Les algorithmes sont essentiels pour résoudre des problèmes efficacement, réduire le temps d'exécution et l'utilisation de la mémoire.

### Différence entre Algorithme et Programme

- Algorithme : Concept ou idée pour résoudre un problème.
- Programme : Implémentation concrète d'un algorithme dans un langage de programmation.

### Notion de Big O Notation

Outil mathématique pour décrire la complexité d'un algorithme en termes de temps (complexité temporelle) ou d'espace (complexité spatiale) en fonction de la taille de l'entrée.

### Exemples de Complexités Courantes :

- $O(1)$  : Temps constant. Exemple : Accès à un élément d'un tableau par son index.
- $O(n)$  : Temps linéaire. Exemple : Parcourir une liste pour trouver un élément.
- $O(n^2)$  : Temps quadratique. Exemple : Tri à bulles.

### Importance de la Complexité dans le Choix d'un Algorithme

Choisir un algorithme avec une meilleure complexité peut grandement affecter la performance d'une application, surtout avec de grandes données.

### Algorithmes de Tri

- Tri par Sélection : Trouve le plus petit élément et le place au début, répète pour la sous-liste restante.
- Tri à Bulles : Compare les éléments adjacents et échange s'ils sont dans le mauvais ordre.

### Algorithmes de Recherche

- Recherche Linéaire : Parcourt chaque élément jusqu'à trouver la correspondance.
- Recherche Binaire : Sur une liste tirée, compare l'élément cible avec l'élément médian et continue dans la sous-liste appropriée.

## Récapitulatif de la partie 15 – Les matrices

---

### Définition

Matrice : Un tableau rectangulaire de nombres, de symboles, ou d'expressions, arrangés en lignes et colonnes.

### Éléments de Matrice

Élément  $a_{ij}$  : Représente l'élément de la matrice situé à la  $i$ -ème ligne et la  $j$ -ème colonne.

### Taille de Matrice

Notation  $m \times n$  : Indique une matrice avec  $m$  lignes et  $n$  colonnes.

### Types Spécifiques de Matrices

- Matrice Carrée : Une matrice avec le même nombre de lignes et de colonnes ( $n \times n$ ).
- Matrice Diagonale : Une matrice carrée où tous les éléments hors de la diagonale principale sont zéro.
- Matrice Unitaire (Identité) : Une matrice diagonale avec des 1 sur la diagonale principale.
- Matrice Nulle : Une matrice dont tous les éléments sont zéro.

### Addition et Soustraction de Matrices

- Condition : Les matrices doivent avoir la même dimension ( $m \times n$ ).
- Opération : Effectuée élément par élément pour correspondre les positions.

### Multiplication Scalaire

- Opération : Multiplier chaque élément de la matrice par un scalaire (nombre).
- Effet : Modifie la magnitude des éléments sans changer la dimension de la matrice.

### Multiplication Matricielle

- Condition : Le nombre de colonnes de la première matrice doit être égal au nombre de lignes de la seconde ( $A_{m \times n}$  et  $B_{n \times p}$ ).
- Résultat : Une nouvelle matrice avec la dimension  $m \times p$ .
- Calcul : L'élément  $c_{ij}$  est calculé comme le produit scalaire de la  $i$ -ème ligne de  $A$  et la  $j$ -ème colonne de  $B$ .

## Récapitulatif de la partie 16 – Numpy : création et exploration de tableaux

---

### Création d'un Array Numpy

- Créer un tableau Numpy: `arr = np.array([1, 2, 3])`

### Attributs d'un Array

- `shape`: Retourne les dimensions du tableau (ex: `arr.shape`)
- `size`: Nombre total d'éléments (ex: `arr.size`)
- `dtype`: Type de données des éléments (ex: `arr.dtype`)
- `ndim`: Nombre de dimensions (ex: `arr.ndim`)
- `itemsize`: Taille en octets de chaque élément (ex: `arr.itemsize`)

### Accès aux Éléments par Indexation

- Élément unique: `arr[2]` ou `arr[-1]`
- Pour un tableau multidimensionnel: `arr[0,1]`

### Accès aux Éléments par Slicing

- Slice d'une rangée: `arr[2:]` ou `arr[:2]`
- Slice d'un tableau 2D: `arr[1:, :2]`
- Slices avec pas: `arr[::2]`

### Effectuer des boucles sur les arrays

#### Méthode Classique

```
for i in range(arr.shape[0]):  
    print(arr[i])
```

Via `flatten` pour itérer sur chaque élément d'un tableau multidimensionnel comme s'il était 1D:

```
for val in arr.flatten():  
    print(val)
```

Via `ndenumerate` pour obtenir index et valeur

```
for idx, val in np.ndenumerate(arr):  
    print(idx, val)
```

Via `nditer` pour itérer sur chaque élément d'un tableau de n'importe quelle dimension

```
for val in np.nditer(arr):  
    print(val)
```

## Récapitulatif de la partie 17 – Numpy : Opérations arithmétiques de base

---

### Opérations de Base sur les Matrices

- Addition :  $C = A + B$  - Addition élément par élément de deux matrices A et B.
- Soustraction :  $C = A - B$  - Soustraction élément par élément.
- Multiplication élément par élément :  $C = A * B$  - Multiplie élément par élément (ne pas confondre avec le produit matriciel).
- Division :  $C = A / B$  - Division élément par élément.

### Produit Matriciel

- `.dot` :  $C = A.dot(B)$  ou  $C = np.dot(A, B)$  - Produit matriciel de A par B.
- `@` :  $C = A @ B$  - Autre syntaxe pour le produit matriciel.

### Fonctions d'Agrégation

- Somme : `np.sum(A)` - Somme de tous les éléments.
- Moyenne : `np.mean(A)` - Moyenne de tous les éléments.
- Minimum : `np.min(A)` - Plus petit élément.
- Maximum : `np.max(A)` - Plus grand élément.
- Médiane : `np.median(A)` - Valeur médiane.
- Écart type : `np.std(A)` - Mesure de la dispersion des données.
- Variance : `np.var(A)` - Moyenne des carrés des écarts à la moyenne.

### Fonctions Mathématiques Universelles (ufuncs)

- Sinus : `np.sin(A)` - Sinus des éléments de A (A en radians).
- Logarithme naturel : `np.log(A)` - Logarithme naturel (base e) des éléments de A.
- Racine carrée : `np.sqrt(A)` - Racine carrée de chaque élément.
- Exponentielle : `np.exp(A)` - e puissance les éléments de A.

## Récapitulatif de la partie 18 – Numpy : data cleaning et modeling

---

### Chargement de Données

- `np.loadtxt()`
  - Utilisé pour charger des données depuis un fichier texte.
  - Arguments clés :
    - **fname** : nom du fichier.
    - **delimiter** : délimiteur entre les données.
    - **skiprows** : nombre de lignes à ignorer au début du fichier.
    - **dtype** : type de données des éléments chargés.
- `np.genfromtxt()`
  - Similaire à **np.loadtxt** mais avec plus de flexibilité pour gérer les données manquantes.
  - Arguments clés :
    - **fname** : nom du fichier.
    - **delimiter** : délimiteur entre les données.
    - **skip\_header** : nombre de lignes à ignorer au début du fichier.
    - **filling\_values** : valeur utilisée pour remplacer les données manquantes.
    - **dtype** : type de données des éléments chargés.

### Gestion des Données Manquantes

- **np.isnan()** : Teste si chaque élément est NaN.
- **np.nan** : Représente une valeur "Not a Number".
- **np.nanmean()**, **np.nanmax()**, **np.nanmin()**, **np.nansum()** : Calculent respectivement la moyenne, le maximum, le minimum, et la somme des éléments d'un array, en ignorant les NaN.

### Manipulation d'Arrays

- **np.where()** : Retourne les indices des éléments qui satisfont une condition.
- **.copy()** : Crée une copie profonde d'un array.
- **.reshape()** : Change la forme d'un array sans en modifier les données.
- **.concatenate()** : Concatène une séquence d'arrays le long d'un axe existant.
- **.stack()** : Empile une séquence d'arrays le long d'un nouvel axe.
- **.sort()** : Trie les éléments d'un array, le long d'un axe spécifié.

### Types de Données

- **dtype** : Détermine le type de données d'un array Numpy (par exemple, **float64**, **int32**, **str**).



### Exemple de type de tableaux :

- `np.int8` : Entier signé 8 bits (-128 à 127)
- `np.int16` : Entier signé 16 bits (-32768 à 32767)
- `np.uint16` : Entier non signé 16 bits (0 à 65535)
- `np.uint32` : Entier non signé 32 bits (0 à 4294967295)
- `np.float16` : Flottant demi-précision
- `np.float32` : Flottant simple précision
- `np.bool_` : Valeur booléenne stockant True ou False
- `np.object_` : Objet Python
- `np.string_` : Chaîne de caractères ASCII à taille fixe
- `None` : pas de type spécifié

## Récapitulatif de la partie 19 – Numpy : fonctionnalités avancées

---

### Génération de Nombres Aléatoires

- `np.random.random(size=None)`
  - Génère des nombres aléatoires dans l'intervalle [0.0, 1.0).
  - `size` : Définit la forme de l'array de sortie.
- `np.random.rand(d0, d1, ..., dn)`
  - Génère des nombres aléatoires dans l'intervalle [0.0, 1.0) avec une forme spécifiée.
  - `d0, d1, ..., dn` : Dimensions de l'array de sortie.
- `np.random.randint(low, high=None, size=None, dtype='i')`
  - Génère des entiers aléatoires dans l'intervalle [low, high).
  - `low` : Borne inférieure (incluse).
  - `high` : Borne supérieure (exclusive).
  - `size` : Forme de l'array de sortie.

### Contrôle de la Reproductibilité

- `np.random.seed(seed=None)`
  - Initialise le générateur de nombres aléatoires avec une graine pour reproduire les résultats.
  - `seed` : Valeur de la graine.
- `np.random.default_rng(seed=None)`
  - Retourne un nouvel objet générateur de nombres aléatoires.
  - `seed` : Valeur de la graine pour initialiser le générateur.

### Nouvelle API de Génération de Nombres Aléatoires

#### (`rng = np.random.default_rng()`)

- `rng.integers(low, high=None, size=None, endpoint=False, dtype='i')`
  - Génère des entiers aléatoires, similaire à `np.random.randint` mais pour la nouvelle API.
- `rng.random(size=None, dtype=np.float64, out=None)`
  - Génère des nombres aléatoires flottants dans l'intervalle [0.0, 1.0).
- `rng.normal(loc=0.0, scale=1.0, size=None)`
  - Génère des échantillons à partir d'une distribution normale (Gaussienne).
- `rng.binomial(n, p, size=None)`
  - Génère des échantillons d'une distribution binomiale.

## Récapitulatif de la partie 20 – Pandas : construire un dataframe

---

### Créer et consulter une Series Pandas

```
import pandas as pd

# Créer une Series à partir d'une liste
data = [1, 2, 3, 4]
serie = pd.Series(data)
print(serie)

# Consulter les éléments d'une Series
print(serie[0]) # Premier élément
```

### Créer et consulter un DataFrame

```
data = {'nom': ['Alice', 'Bob'], 'âge': [25, 30]}
df = pd.DataFrame(data)
print(df)
```

### Gestion de l'index

```
df.set_index('nom', inplace=True)
print(df)
```

### Créer un DataFrame à partir d'un fichier CSV

```
df = pd.read_csv('chemin/vers/fichier.csv')
```

### Créer un DataFrame à partir d'un fichier Excel

```
df = pd.read_excel('chemin/vers/fichier.xlsx')
```

### Créer un DataFrame à partir d'un fichier JSON

```
df = pd.read_json('chemin/vers/fichier.json')
```

### Exporter des DataFrame en CSV/Excel/JSON

```
df.to_csv('chemin/vers/fichier_sortie.csv')
df.to_excel('chemin/vers/fichier_sortie.xlsx')
df.to_json('chemin/vers/fichier_sortie.json')
```

## Récapitulatif de la partie 21 – Pandas : naviguer dans un dataframe

---

### Sélection de Certaines Lignes et Colonnes

- Sélectionner une colonne par son nom : `df['nom_colonne']`
- Sélectionner plusieurs colonnes par leurs noms : `df[['nom_colonne1', 'nom_colonne2']]`
- Sélectionner des lignes par leur position : `df.iloc[indices_lignes]`
- Sélectionner des lignes par leur label : `df.loc[labels_lignes]`

### La méthode loc (Sélection basée sur le label)

- Sélectionner une seule ligne : `df.loc[label_ligne]`
- Sélectionner plusieurs lignes et/ou colonnes spécifiques : `df.loc[labels_lignes, labels_colonnes]`
- Slicing sur les lignes et sélection spécifique de colonnes : `df.loc[début:fin, ['nom_colonne1', 'nom_colonne2']]`
- Conditionnelle (par exemple, toutes les lignes où la colonne 'age' > 30) : `df.loc[df['age'] > 30]`

### La méthode iloc (Sélection basée sur la position)

- Sélectionner une seule ligne par sa position : `df.iloc[index]`
- Sélectionner plusieurs lignes et colonnes par leur position : `df.iloc[indices_lignes, indices_colonnes]`
- Slicing sur les lignes et les colonnes : `df.iloc[début_lignes:fin_lignes, début_colonnes:fin_colonnes]`

### Slicing

- Slicing de lignes : `df[début:fin]`
- Slicing conditionnel (par exemple, toutes les lignes où la colonne 'age' > 30 et < 50) : `df[(df['age'] > 30) & (df['age'] < 50)]`
- Utilisation de `slice()` pour des intervalles complexes : `df.loc[slice(début, fin), slice(début_col, fin_col)]`

## Récapitulatif de la partie 21 – Pandas : Data Cleaning

---

### Première Analyse d'un DataFrame

- **df.describe()** : Statistiques résumées pour les colonnes numériques.
- **df.head(n)** : Affiche les n premières lignes du DataFrame.
- **len(df)** : Nombre de lignes dans le DataFrame.
- **df.tail(n)** : Affiche les n dernières lignes du DataFrame.
- **df.info()** : Résumé des types de données, non-null values et mémoire utilisée.

### Création de DataFrame avec Certaines Colonnes

- **df\_new = df[['colonne1', 'colonne2']]** : Crée un nouveau DataFrame en sélectionnant certaines colonnes.

### Renommer des Colonnes

- **df.rename(columns={'ancien\_nom': 'nouveau\_nom'}, inplace=True)** : Renomme les colonnes spécifiées.

### Gestion et Modification des Types de Colonnes

- **df['colonne'].astype(float)** : Convertit le type de la colonne en float.
- **df['colonne'].astype(str)** : Convertit le type de la colonne en string.

### Gestion des Valeurs Manquantes

- **df.fillna(value)** : Remplace les valeurs NaN par "value".
- **df.dropna()** : Supprime les lignes contenant des valeurs NaN.

### Supprimer les Doublons

- **df.drop\_duplicates(inplace=True)** : Supprime les lignes en double dans le DataFrame.

### Suppression de Lignes via Filtrage

- **df = df[df['colonne'] != valeur]** : Supprime les lignes où la colonne a une valeur spécifique.

### Transformation avec .apply()

- **df['colonne'] = df['colonne'].apply(lambda x: x.replace('\$', '').astype(float))** : Supprime le symbole dollar et convertit en float.

## Gestion des Paramètres de Pandas

- **pd.set\_option('display.max\_rows', 10)** : Limite le nombre de lignes affichées à 10.
- **pd.set\_option('display.max\_columns', 10)** : Limite le nombre de colonnes affichées à 10.
- **pd.options.mode.chained\_assignment = None** : Option pour ignorer l'avertissement sur l'assignation en chaîne.

## Récapitulatif de la partie 22 – Pandas : Data Modeling

---

### Filtrer des Données

1. Filtrage sur valeurs numériques : Utilisez `df[df['Age'] > 30]` pour filtrer les lignes où l'âge est supérieur à 30.
2. Filtrage sur valeur textuelle : Pour filtrer où le nom contient 'John', utilisez `df[df['Name'].str.contains('John')]`.
3. Utiliser `isin` pour un ensemble de valeurs : Filtrer par statut peut se faire avec `df[df['Status'].isin(['Married', 'Single'])]`.
4. Combinaison de filtres avec AND (&) et OR (|) : Pour trouver les personnes de plus de 30 ans qui sont célibataires, utilisez `df[(df['Age'] > 30) & (df['Status'] == 'Single')]`.
5. Filtrage avec `query` : Une alternative à l'aide de `query` serait `df.query('Age > 30 & Status == "Single"')`.

### Trier les Données

1. Tri simple : Tri par âge en ordre ascendant se fait par `df.sort_values(by='Age')`.
2. Tri ascendant / descendant : Pour trier par âge en ordre descendant, utilisez `df.sort_values(by='Age', ascending=False)`.
3. Tri multicritères : Tri par département, puis par âge en ordre mixte peut être réalisé avec `df.sort_values(by=['Department', 'Age'], ascending=[True, False])`.
4. Gestion des valeurs manquantes lors du tri : Pour placer les valeurs NaN en premier, utilisez `df.sort_values(by='Age', na_position='first')`.
5. Tri via `key` : Tri insensible à la casse par nom s'effectue avec `df.sort_values(by='Name', key=lambda col: col.str.lower())`.

### Création de Nouvelles Colonnes

1. Concaténation de colonnes : Créez un nom complet avec `df['FullName'] = df['FirstName'] + ' ' + df['LastName']`.
2. Opérations mathématiques : Pour augmenter le salaire de 10%, utilisez `df['NewSalary'] = df['Salary'] * 1.1`.
3. Conditions avec `np.where` : Assignez des catégories d'âge avec `import numpy as np` puis `df['Category'] = np.where(df['Age'] > 50, 'Senior', 'Adult')`.
4. Utiliser `.apply` pour des transformations personnalisées : Enlever le signe dollar et convertir en float avec `df['Price'] = df['Price'].apply(lambda x: x.replace('$', '').astype(float))`.
5. Utiliser `.map` pour des transformations directes : Convertissez des codes en genre textuel avec `df['Gender'] = df['GenderCode'].map({1: 'Male', 2: 'Female'})`.

## Récapitulatif de la partie 23 – Pandas : Gestion des dates

---

### Conversion de Colonne String en Date

- Pour convertir une colonne de texte en date, utilisez `pd.to_datetime(df['date_column'])`. Pour spécifier un format particulier, comme année-mois-jour, utilisez `pd.to_datetime(df['date_column'], format='%Y-%m-%d %H:%M:%S')`.

### Différents Formats de Dates pour les Conversions

- Les formats courants incluent année-mois-jour `%Y-%m-%d`, et heure-minute-seconde `%H:%M:%S`. Pour un format complet incluant date et heure, utilisez `%Y-%m-%d %H:%M:%S`.

### Gestion du Format Unix

- Pour convertir un timestamp Unix en datetime, appliquez `pd.to_datetime(df['timestamp_column'], unit='s')`, où 's' indique que le timestamp est en secondes.

### Extraire Composants d'une Date

- Pour extraire des éléments comme l'année, le mois, ou l'heure d'une colonne datetime, utilisez des commandes comme `df['date_column'].dt.year`, `df['date_column'].dt.month`, ou `df['date_column'].dt.hour`. Pour le jour de l'année ou la semaine de l'année, utilisez `df['date_column'].dt.dayofyear` ou `df['date_column'].dt.weekofyear`.

### Calcul de la Durée Entre Deux Dates

- Pour calculer la différence entre deux dates, soustrayez-les directement : `df['duration'] = df['end_date'] - df['start_date']`.

### Ajouter ou Soustraire du Temps sur une Date

- Pour ajouter ou soustraire des jours à une date, utilisez `pd.DateOffset`. Par exemple, ajoutez cinq jours avec `df['new_date'] = df['date_column'] + pd.DateOffset(days=5)` ou soustrayez cinq jours avec `df['new_date'] = df['date_column'] - pd.DateOffset(days=5)`.

### Gestion des Fuseaux Horaires

- Pour localiser une date dans un fuseau horaire spécifique, commencez par la localiser en UTC avec `df['date_column'].dt.tz_localize('UTC')`, puis convertissez-la en un autre fuseau horaire comme `df['date_column'].dt.tz_convert('America/New_York')`.



## Récapitulatif de la partie 24 – Pandas : Analyse des données

---

### Opérations Statistiques de Base

- Pour calculer la somme d'une colonne 'Salary' : **df['Salary'].sum()**.
- Pour obtenir la moyenne : **df['Salary'].mean()**.
- Pour le minimum : **df['Salary'].min()**.
- Pour le maximum : **df['Salary'].max()**.
- Pour la médiane : **df['Salary'].median()**.
- Pour compter le nombre d'éléments : **df['Department'].count()**.

### Comptage et Valeurs Uniques

- Pour compter les valeurs uniques dans 'Department' : **df['Department'].nunique()**.
- Pour lister les valeurs uniques : **df['Department'].unique()**.
- Pour voir la fréquence des valeurs uniques : **df['Department'].value\_counts()**.

### Mesures de Distribution

- Pour calculer l'écart type : **df['Salary'].std()**.
- Pour l'asymétrie (skewness) : **df['Salary'].skew()**.
- Pour le kurtosis : **df['Salary'].kurt()**.
- Pour la covariance entre 'Age' et 'Salary' : **df[['Age', 'Salary']].cov()**.
- Pour la corrélation : **df[['Age', 'Salary']].corr()**.

### Groupement par Catégories

- Pour grouper par 'Department' et calculer la moyenne des salaires : **df.groupby('Department')['Salary'].mean()**.

### Table Pivot

- Pour créer une table pivot qui montre la moyenne des salaires par département : **df.pivot\_table(values='Salary', index='Department', aggfunc='mean')**.

### Segmentation des Données avec cut et qcut

- Pour diviser 'Age' en 4 catégories de largeur égale : **df['AgeGroup'] = pd.cut(df['Age'], bins=4)**.
- Pour diviser 'Salary' en quartiles : **df['SalaryQuantile'] = pd.qcut(df['Salary'], q=4)**.

## Gestion des Fuseaux Horaires

- Pour convertir les horaires UTC à un fuseau horaire spécifique, assurez-vous d'abord de localiser la date avec UTC puis convertissez-la, par exemple pour New York : **`df['DateTime'] = df['DateTime'].dt.tz_localize('UTC').dt.tz_convert('America/New_York')`**.

## Récapitulatif de la partie 25 – Pandas : Combiner plusieurs sources de données

---

### Concat

Pour combiner plusieurs DataFrames verticalement ou horizontalement, utilisez `pd.concat([df1, df2], axis=0)` pour un empilement vertical (par défaut), ou `pd.concat([df1, df2], axis=1)` pour un empilement horizontal. Ajoutez `ignore_index=True` si vous souhaitez réinitialiser l'index dans le DataFrame résultant.

### Join

Pour joindre deux DataFrames en utilisant les index comme clés, appliquez `df1.join(df2)` qui par défaut réalise une jointure gauche sur les index. Vous pouvez également spécifier `how='inner'`, `how='outer'`, ou `how='right'` pour modifier le type de jointure.

### Merge

Pour fusionner deux DataFrames sur une ou plusieurs colonnes en clé, employez `pd.merge(df1, df2, on='key_column')` pour une jointure interne par défaut. Utilisez `how='left'`, `how='right'`, ou `how='outer'` pour modifier le type de jointure, et `left_on` et `right_on` si les noms de colonnes clés diffèrent entre les deux DataFrames.

### **Théorie sur la data visualisation**

#### **1. Pourquoi la Data Viz ?**

- Simplification : Transforme des informations complexes pour une compréhension facilitée.
- Tendances : Permet de déceler des patterns et des tendances dans les données.
- Valeurs : Met en avant les valeurs inhérentes aux données.
- Relations : Révèle les relations entre différents jeux de données.

#### **2. Les 3 Piliers de la Data Visualisation**

- Qualité des Données : Assure que les données sont exactes et pertinentes.
- Choix du Graphique : Sélectionne le type de graphique le plus approprié pour représenter les données.
- Techniques de Mise en Forme : Utilise des méthodes de design pour améliorer la lisibilité et l'esthétique.

#### **3. Bonnes Pratiques en Data Viz**

- Couleurs comme Information : Utilise les couleurs pour différencier ou mettre en évidence les données.
- Esthétique : Évite les mises en forme textuelles trop complexes qui peuvent distraire.
- Redondance : Élimine toute information redondante pour une présentation claire.
- Échelles : N'utilise des échelles que quand elles apportent une valeur ajoutée.
- Simplicité : Préfère des visualisations simples pour une meilleure clarté.

#### **4. Attributs Pré-Attentifs en Data Viz**

- Orientation, Longueur, Largeur, Taille : Différencient les éléments rapidement.
- Forme, Courbure, Couleur, Position : Aident à organiser visuellement les informations.
- Marquage Supplémentaire, Encadrement, Groupement Spatial : Améliorent le focus sur des éléments spécifiques.

#### **5. Hiérarchie Visuelle en Data Viz**

- Position : Les éléments les plus importants sont placés où les regards se posent en premier.
- Couleur : Distingue les éléments par leur importance ou catégorie.
- Taille : Utilise la taille pour indiquer l'importance ou la quantité.
- Forme : Emploie des formes distinctes pour communiquer des types de données ou des actions.



## Ajout de Titres, de Légendes et d'Étiquettes d'Axes

Pour ajouter un titre, une légende et des étiquettes d'axes :

```
ax.set_title('Titre du graphique') # Titre du graphique
ax.set_xlabel('Étiquette de l'axe X') # Étiquette de l'axe X
ax.set_ylabel('Étiquette de l'axe Y') # Étiquette de l'axe Y
ax.legend(['Légende de la ligne']) # Ajoute une légende
```

## Construction de Diagrammes à Barres et Histogrammes

Pour créer un diagramme à barres :

```
ax.bar(categories, values) # categories et values sont des listes de
catégories et de valeurs respectives
```

Pour créer un histogramme :

```
ax.hist(data, bins=10) # data est une liste ou un array de valeurs, bins
spécifie le nombre de bacs
```

## Création de Nuages de Points

Pour créer un nuage de points :

```
ax.scatter(x, y, color='red') # x et y sont des listes ou des arrays de
valeurs, points rouges
```

## Graphiques à Aires et Aires Empilées

Pour créer un graphique à aires :

```
ax.fill_between(x, y1, y2, color='green', alpha=0.3) # Remplit la zone entre
y1 et y2 avec une couleur verte translucide
```

Pour créer un graphique à aires empilées :

```
ax.stackplot(x, y1, y2, y3, labels=['y1', 'y2', 'y3']) # Crée des aires
empilées avec légendes pour chaque série
ax.legend(loc='upper left') # Positionne la légende en haut à gauche
```

## Récapitulatif de la partie 27 – Data visualisation : Seaborn

### Composants d'un Graphique avec Seaborn

Figure : L'objet de base qui contient un ou plusieurs graphiques.

```
import matplotlib.pyplot as plt
import seaborn as sns
fig = plt.figure() # Crée une figure
```

Axes : Les sous-parties de la figure où les graphiques sont dessinés.

```
ax = fig.add_subplot(111) # Crée un axe dans la figure
```

Grille : Une disposition des axes dans une grille (utile pour les graphes multiples).

```
g = sns.FacetGrid(data, col="variable") # Crée une grille de graphes
```

### Les Thèmes de Seaborn

Seaborn offre plusieurs thèmes pour personnaliser l'apparence des graphiques.

Définir un thème

```
sns.set_theme(style="darkgrid") # Options : darkgrid, whitegrid, dark, white, ticks
```

Styles et contexts

```
sns.set_style("whitegrid") # Change le style de la grille
sns.set_context("talk") # Change le contexte (paper, notebook, talk, poster)
```

### Distribution / Boxplot / Violon

#### Distribution Plot

```
sns.displot(data, x="variable", kde=True) # kde=True ajoute une courbe de densité
```

#### Boxplot

```
sns.boxplot(x="variable1", y="variable2", data=data)
```

#### Violin Plot

```
sns.violinplot(x="variable1", y="variable2", data=data)
```

## Récapitulatif de la partie 27 – Data visualisation : plotly express

---

### Graphique Linéaire (Line Plot)

```
fig = px.line(data, x='x_variable', y='y_variable', title='Titre du Graphique Linéaire')
fig.show()
```

### Scatter Plot

```
fig = px.scatter(data, x='x_variable', y='y_variable', title='Titre du Scatter Plot')
fig.show()
```

### Pie Chart

```
fig = px.pie(data, names='category_variable', values='value_variable', title='Titre du Pie Chart')
fig.show()
```

### Sunburst Chart

```
fig = px.sunburst(data, path=['level_1', 'level_2'], values='value_variable', title='Titre du Sunburst Chart')
fig.show()
```

### Treemap

```
fig = px.treemap(data, path=['level_1', 'level_2'], values='value_variable', title='Titre du Treemap')
fig.show()
```

### px.scatter\_matrix

```
fig = px.scatter_matrix(data, dimensions=['dimension_1', 'dimension_2', 'dimension_3'], title='Titre de la Matrice de Dispersion')
fig.show()
```

### Funnel

```
fig = px.funnel(data, x='x_variable', y='y_variable', title='Titre du Funnel Chart')
fig.show()
```

### Animation Temporelle

```
fig = px.scatter(data, x='x_variable', y='y_variable', animation_frame='time_variable', title='Titre de l\'Animation Temporelle')
fig.show()
```



## Récapitulatif de la partie 28 – Data visualisation : données géographiques

---

### Création d'une carte choroplèth

Pour créer une carte choroplèth avec Plotly Express :

```
fig = px.choropleth(data, Locations='iso_alpha_3', color='variable',  
title='Titre de la Carte Choroplèthe')  
fig.show()
```

### Code ISO Alpha-3

Le code ISO Alpha-3 est un code de pays à trois lettres utilisé pour identifier les pays de manière standardisée, par exemple :

- États-Unis : USA
- France : FRA
- Japon : JPN

### Création d'un scatter plot géographique avec des codes ISO Alpha-3

```
fig = px.scatter_geo(data, Locations='iso_alpha_3', size='variable',  
title='Scatter Plot Géographique avec ISO Alpha-3')  
fig.show()
```

### Projections de cartes courantes

Plotly Express supporte plusieurs projections de cartes, y compris :

- Mercator : projection='mercator'
- Equirectangular : projection='equirectangular'
- Orthographic : projection='orthographic'
- Mollweide : projection='mollweide'

Exemple :

```
fig = px.scatter_geo(data, Locations='iso_alpha_3', projection='mercator',  
title='Carte avec Projection Mercator')  
fig.show()
```

### Longitude et Latitude

Les coordonnées de longitude et latitude représentent des points dans un système de coordonnées géographiques :

- Longitude : Coordonnée est-ouest (ex : -74.0060 pour New York)
- Latitude : Coordonnée nord-sud (ex : 40.7128 pour New York)

Pour créer un scatter plot géographique avec des données de longitude et latitude :

```
fig = px.scatter_geo(data, lat='latitude', lon='longitude', size='variable',  
title='Scatter Plot Géographique avec Longitude et Latitude')  
fig.show()
```

## Fichier GeoJSON

Un fichier GeoJSON est un format de fichier basé sur JSON qui contient des informations géographiques. Il peut représenter des objets géométriques comme des points, des lignes et des polygones, ainsi que des propriétés associées.

Exemple d'utilisation :

```
import json  
  
# Charger le fichier GeoJSON  
with open('path/to/geojson_file.geojson') as f:  
    geojson_data = json.load(f)  
  
# Créer la carte choroplèthe  
fig = px.choropleth(data, geojson=geojson_data, locations='location_column',  
color='variable', title='Carte Choroplèthe avec GeoJSON')  
fig.update_geos(fitbounds="locations", visible=False)  
fig.show()
```

## Récapitulatif de la partie 29 – SQL et Python

### Le format SQLite

SQLite est un moteur de base de données relationnelle léger qui ne nécessite pas de serveur séparé. Il stocke la base de données entière (définitions, tables, index, et les données elles-mêmes) dans un seul fichier cross-platform.

### Se connecter à une base de données SQLite en Python

Utilisez sqlite3 pour vous connecter à SQLite:

```
import sqlite3
conn = sqlite3.connect('chemin_vers_votre_fichier.db')
```

### Récupérer les résultats d'une requête SQLite dans un DataFrame pandas

Exécuter une requête et charger les résultats dans un DataFrame :

```
import pandas as pd
df = pd.read_sql_query("SELECT * FROM votre_table", conn)
```

### Construction d'une requête SELECT

Syntaxe basique pour sélectionner des données :

```
cursor = conn.cursor()
cursor.execute("SELECT colonne1, colonne2 FROM table")
```

### Faire des filtres dans une requête SQL

Utiliser WHERE pour filtrer les résultats :

```
cursor.execute("SELECT * FROM table WHERE colonne1 = 'valeur'")
```

### Utilisation de LIMIT et ORDER BY dans une requête SQL

Limiter le nombre de résultats et ordonner les données :

```
cursor.execute("SELECT * FROM table ORDER BY colonne1 DESC LIMIT 10")
```

### Réaliser des GROUP BY dans une requête SQL

Grouper des résultats et appliquer des fonctions d'agrégation :

```
cursor.execute("SELECT colonne1, COUNT(*) FROM table GROUP BY colonne1")
```

### HAVING

Filtrer les résultats après un GROUP BY :

```
cursor.execute("SELECT colonne1, COUNT(*) FROM table GROUP BY colonne1 HAVING COUNT(*) > 1")
```

## Les JOINTURES SQL

Joindre deux tables pour combiner des données liées :

```
cursor.execute('''SELECT a.colonne1, b.colonne2  
                  FROM table1  
                  a JOIN table2 b ON a.id = b.id''')
```

## Récapitulatif de la partie 29 – WebScraping : BeautifulSoup

### Importation des bibliothèques

```
from bs4 import BeautifulSoup
import requests
```

### Envoyer une requête HTTP

```
page = requests.get("http://quotes.toscrape.com/")
```

### Obtenir le contenu brut de la réponse (HTML de la page)

```
print(page.content)
```

### Afficher le code de statut de la réponse http

```
print(page.status_code)
```

### Les erreurs http courantes

Code HTTP	Signification	Causes les plus probables
200	OK	La requête a réussi et le serveur a retourné le contenu demandé.
201	Created	La requête a abouti à la création d'une nouvelle ressource.
204	No Content	La requête a réussi mais il n'y a pas de contenu à retourner.
301	Moved Permanently	La ressource demandée a été déplacée de manière permanente à une nouvelle URL.
302	Found	La ressource demandée réside temporairement sous une autre URL.
304	Not Modified	La ressource n'a pas été modifiée depuis la dernière requête. Utilisé pour la mise en cache.
400	Bad Request	La requête est mal formée ou contient des données invalides.
401	Unauthorized	L'authentification est nécessaire pour accéder à la ressource.
403	Forbidden	Le serveur comprend la requête, mais refuse de l'exécuter.
404	Not Found	La ressource demandée n'a pas été trouvée sur le serveur.

Code HTTP	Signification	Causes les plus probables
405	Method Not Allowed	La méthode HTTP utilisée n'est pas autorisée pour la ressource demandée.
408	Request Timeout	Le serveur a expiré en attendant que la requête soit terminée.
409	Conflict	La requête ne peut pas être traitée à cause d'un conflit avec l'état actuel de la ressource.
410	Gone	La ressource demandée n'est plus disponible et ne le sera plus.
429	Too Many Requests	Le client a envoyé trop de requêtes dans un temps donné.
500	Internal Server Error	Le serveur a rencontré une condition inattendue qui l'a empêché de traiter la requête.
501	Not Implemented	Le serveur ne supporte pas la fonctionnalité requise pour traiter la requête.
502	Bad Gateway	Le serveur, agissant comme une passerelle ou un proxy, a reçu une réponse invalide du serveur en amont.
503	Service Unavailable	Le serveur est actuellement incapable de traiter la requête à cause d'une surcharge ou d'une maintenance.
504	Gateway Timeout	Le serveur, agissant comme une passerelle ou un proxy, n'a pas reçu une réponse à temps du serveur en amont.

### Parser une page web

```

page = requests.get("http://quotes.toscrape.com/")
if page.status_code == 200:
    soup = BeautifulSoup(page.content, 'html.parser')

```

## Trouver des éléments dans le HTML

```
# Trouver la première occurrence d'une balise
title = soup.find('title')
print(title.get_text())

#Trouver toutes les occurrences d'une balise
quotes = soup.find_all('span', class_='text')
for quote in quotes:
    print(quote.get_text())

#Trouver un élément par ID
main_div = soup.find(id='main')
print(main_div)
```

## Accéder aux attributs d'une balise

```
image = soup.find('img')
image_src = image['src']
print(image_src)
```

## Obtenir le texte à l'intérieur d'une balise

```
text = soup.find('span', class_='text').get_text()
print(text)
```

## Récapitulatif de la partie 30 – WebScraping : Selenium

### Importation des Bibliothèques

```
from selenium import webdriver # Fournit les classes et méthodes nécessaires
pour contrôler un navigateur
from selenium.webdriver.chrome.service import Service # Gère le cycle de vie
du processus du driver
from selenium.webdriver.common.by import By # Permet de localiser des
éléments sur une page web
from selenium.webdriver.support.ui import WebDriverWait # Permet d'attendre
qu'une condition soit remplie
from selenium.webdriver.support import expected_conditions as EC # Fournit
des conditions pour les attentes explicites
```

### Initialiser le WebDriver

```
service = Service(executable_path='path_to_chromedriver') # Chemin vers
chromedriver
driver = webdriver.Chrome(service=service)
```

### Navigation vers une URL

```
driver.get('http://quotes.toscrape.com/js-delayed/')
```

### Attendre qu'un Élément soit Présent

```
element = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.CLASS_NAME, 'quote'))
)
```

### Interactions avec les Éléments

```
#Cliquer sur un Élément
element = driver.find_element(By.CLASS_NAME, 'header-search')
element.click()

#Envoyer du Texte dans un Champ
search_field = driver.find_element(By.CLASS_NAME, 'search-field')
search_field.send_keys('shoes')
```

### Sélectionner des Éléments

```
#Par ID
element = driver.find_element(By.ID, 'element_id')

#Par Nom de Classe
elements = driver.find_elements(By.CLASS_NAME, 'classname')

#Par Sélecteur CSS
element = driver.find_element(By.CSS_SELECTOR, 'div.quote span.text')
```



## Récupérer des Attributs

```
img_element = driver.find_element(By.TAG_NAME, 'img')
src = img_element.get_attribute('src')
print(src)
```

## Mode Headless (Sans Interface Graphique)

```
from selenium.webdriver.chrome.options import Options

chrome_options = Options()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu') # Nécessaire sur Windows
chrome_options.add_argument('--no-sandbox') # Nécessaire pour éviter les
problèmes de sandboxing

driver = webdriver.Chrome(service=service, options=chrome_options)
```

## Récapitulatif de la partie 31 – Les API

---

### Envoi d'une Requête API

```
url = "http://api.worldbank.org/v2/country/all/"
response = requests.get(url)
```

### Vérification de la Réponse

```
print(response) # Affiche l'objet réponse
print(response.status_code) # Affiche le code de statut HTTP
print(response.headers) # Affiche les en-têtes de la réponse
print(response.content) # Affiche le contenu brut de la réponse
```

### Conversion des Données JSON en DataFrame

```
data = response.json()
df = pd.DataFrame(data[1]) # Les données sont souvent dans la deuxième partie
                             du JSON
print(df.head()) # Affiche les premières lignes du DataFrame
```

Paramètres de la Requête : les paramètres dans une requête API modifient le contenu de la réponse. Par exemple, pour spécifier le format de la réponse en JSON et limiter les résultats par page.

```
params = {
    'format': 'json',
    'per_page': '10',
    'page': '1'
}
response = requests.get(url, params=params)
```

### Rappels des principaux statuts de requêtes

- 200 : OK - La requête a réussi.
- 400 : Bad Request - La requête est mal formée.
- 404 : Not Found - La ressource demandée n'a pas été trouvée.
- 500 : Internal Server Error - Une erreur est survenue sur le serveur.

### Bonnes Pratiques pour Utiliser une API

- Lire la Documentation : Toujours lire la documentation officielle de l'API pour comprendre les endpoints disponibles et leurs paramètres.
- Gérer les Limites de Taux : Respectez les limites de taux pour éviter les blocages.
- Vérifier les Codes de Statut : Toujours vérifier les codes de statut HTTP pour gérer les erreurs de manière appropriée.
- Sécuriser les Clés API : Ne partagez jamais vos clés API en public et utilisez des variables d'environnement pour les stocker.

## Gestion des données JSON dans un DataFrame

Lorsque vous travaillez avec des données JSON récupérées d'une API, il est courant de se retrouver avec des DataFrame où certaines colonnes contiennent des objets JSON. Pour extraire ces données dans un format tabulaire propre, vous pouvez utiliser la fonction `json_normalize` de pandas.

```
import pandas as pd
from pandas import json_normalize
data = {
    "countries": [
        {"name": "France", "capital": "Paris", "population": 67000000},
        {"name": "Germany", "capital": "Berlin", "population": 83000000}
    ]
}
df = json_normalize(data, 'countries')
print(df)
```

## Récapitulatif de la partie 32 – statistique descriptive

---

### Types de Données

1. Données Quantitatives
  - Discrètes : Valeurs spécifiques et dénombrables (ex. nombre d'étudiants par classe).
  - Continues : Valeurs pouvant prendre toute mesure dans un intervalle (ex. poids, température).
2. Données Qualitatives
  - Catégorise les données qui décrivent des attributs (ex. couleur des yeux, type de véhicule).

### Mesures de Tendance

1. Moyenne : La somme de toutes les valeurs divisée par le nombre total d'observations.
2. Médiane : La valeur qui divise l'ensemble des données en deux moitiés égales.
3. Mode : La valeur la plus fréquemment observée dans un ensemble de données.

### Mesures de Dispersion

1. Minimum et Maximum : Les valeurs les plus petites et les plus grandes dans un ensemble de données.
2. Étendue : La différence entre les valeurs maximale et minimale.
3. Quartiles :
  - Q1 (Premier quartile) : 25% des données sont inférieures à cette valeur.
  - Q2 (Médiane) : 50% des données sont inférieures à cette valeur.
  - Q3 (Troisième quartile) : 75% des données sont inférieures à cette valeur.
4. Écart Type : Mesure la variabilité ou la dispersion des valeurs autour de la moyenne.
5. Variance : L'écart type au carré, mesurant la dispersion des données.

### Mesures de Forme

1. Kurtosis : Indique le degré de concentration des valeurs dans la distribution autour de la moyenne (plus pointu ou plat par rapport à une distribution normale).
2. Skewness (Asymétrie) : Mesure l'asymétrie de la distribution autour de sa moyenne.

### Relations entre Variables

1. Covariance : Indique la direction de la relation linéaire entre deux variables.

## 2. Coefficient de Corrélation :

- Corrélation de Pearson : Mesure le degré de relation linéaire entre deux variables quantitatives.
- Valeurs entre -1 et 1, où 1 signifie une corrélation positive parfaite, -1 une corrélation négative parfaite, et 0 aucune corrélation.

## Récapitulatif de la partie 33 – Data science

---

### Définition de la Data Science

Champ interdisciplinaire qui utilise des méthodes scientifiques, des processus, des algorithmes et des systèmes pour extraire des connaissances et des insights à partir de données structurées et non structurées. Elle implique des techniques de statistiques, d'informatique et de visualisation pour analyser et interpréter des ensembles complexes de données.

### Types de Données

- **Données Structurées** : Données organisées et formatées de manière à être facilement recherchables par des algorithmes simples (ex. bases de données relationnelles).
- **Données Semi-structurées** : Données qui ne sont pas organisées dans un modèle de base de données mais qui possèdent une structure interne (ex. JSON, XML).
- **Données Non Structurées** : Données qui ne suivent pas un modèle ou format spécifique, rendant leur traitement et analyse plus complexes (ex. vidéos, emails, documents textuels)

### Types de Modèles de Machine Learning

- **Régression** : Modèles prédictifs visant à prédire une valeur continue.
- **Classification** : Modèles utilisés pour prédire la catégorie à laquelle appartient une observation. Cela peut être binaire (deux classes) ou multiclasse.
- **Clustering** : Technique non supervisée utilisée pour grouper un ensemble de données en clusters où les membres de chaque cluster sont plus similaires entre eux qu'avec ceux d'autres clusters.

### Data Architecture

- **Data Lake** : système de stockage centralisé qui permet de stocker des quantités massives de données brutes dans leur format natif jusqu'à ce qu'elles soient nécessaires. Les data lakes supportent les données structurées, semi-structurées et non structurées.
- **Data Warehouse** : Système utilisé pour le reporting et l'analyse de données qui est hautement structuré et formaté de manière spécifique pour les requêtes et l'analyse rapide.
- **Différence** : Le data lake est une large réserve de données brutes dont la structure n'est définie que lorsque les données sont lues, ce qui le rend plus flexible et capable de stocker de plus grandes variétés de données. Le data warehouse, en revanche, est hautement structuré, optimisé pour la rapidité des requêtes sur des données déjà transformées et organisées.

## Rôles en Data Science

- Data Engineer : Spécialiste technique chargé de la conception, de la construction, de l'installation, et de la maintenance des systèmes de données utilisés pour recueillir, stocker, et analyser les données à grande échelle.
- Data Analyst : Professionnel qui collecte, traite et effectue des analyses statistiques sur de grands ensembles de données pour identifier des tendances, développer des tableaux de bord et fournir des rapports qui aident à prendre des décisions opérationnelles.
- Data Scientist : Expert qui combine des compétences en statistiques, mathématiques et programmation pour interpréter et analyser des données complexes avec pour objectif de développer des insights guidant les décisions stratégiques.

## Exemples de modèles de Machine Learning

- Decision Tree (Arbre de Décision) : modèle prédictif qui segmente les données en branches pour aboutir à une décision ou prédiction. Chaque nœud de l'arbre représente une caractéristique, et chaque branche une décision, menant finalement à une prédiction finale.
- Random Forest : Un ensemble de nombreux arbres de décision, chaque arbre étant construit à partir d'un échantillon aléatoire des données. La prédiction finale est faite en prenant la moyenne (régression) ou le mode (classification) des prédictions de tous les arbres.
- Neural Network (Réseau de Neurones) : modèle composé de couches de neurones avec des poids ajustables. Les signaux passent à travers les couches et sont transformés par des fonctions d'activation.

## Importance des Métriques d'Erreur

Les métriques d'erreur quantifient la performance d'un modèle en mesurant l'écart entre les valeurs prédites et les valeurs réelles. Elles sont essentielles pour juger de la qualité et de la fiabilité d'un modèle.

- Pour la régression : MSE (Mean Squared Error), RMSE (Root Mean Squared Error), MAE (Mean Absolute Error)
- Pour la classification : Précision, Rappel, F1-Score, AUC-ROC

## Récapitulatif de la partie 34 – Régression linéaire

---

Pour la partie code pure, merci de vous référer au notebook du cours.

### Régression Linéaire

Modèle statistique qui tente de prédire la valeur d'une variable dépendante à partir d'une ou plusieurs variables indépendantes en ajustant une ligne droite (équation linéaire) à l'ensemble des données.

### Méthode des Moindres Carrés

Technique mathématique qui trouve la ligne de meilleur ajustement pour un ensemble de données en minimisant la somme des carrés des écarts (les "carrés des erreurs") entre les valeurs observées et celles prédites par le modèle linéaire.

### Descente du Gradient

Un algorithme d'optimisation utilisé pour minimiser une fonction de coût (souvent la somme des carrés des erreurs dans la régression linéaire) en ajustant itérativement les paramètres du modèle.

### Métriques d'Erreur : MAE et RMSE

- MAE (Mean Absolute Error) : La moyenne des valeurs absolues des écarts entre les prédictions et les observations réelles. C'est une mesure robuste aux outliers.
- RMSE (Root Mean Squared Error) : La racine carrée de la moyenne des carrés des écarts entre les prédictions et les valeurs réelles. Elle pénalise plus les grandes erreurs et est souvent utilisée pour évaluer la qualité des modèles de régression.

### Stratégies de Séparation des Données

- Split Simple : Divise les données en deux parties distinctes, généralement en ensembles d'entraînement et de test. Cette méthode attribue une proportion fixe des données à chaque ensemble.
- Split avec Randomisation : Similaire au split simple, mais inclut une étape de randomisation pour mélanger les données avant de les diviser. Cela aide à prévenir toute biais d'ordre dans les données.
- Méthode K-Fold : Divise les données en  $k$  sous-ensembles distincts (ou "folds"). Le modèle est entraîné sur  $k-1$  folds, avec le fold restant utilisé comme test. Ce processus est répété  $k$  fois, avec chaque fold utilisé exactement une fois comme ensemble de test.

### Standardisation des valeurs

Le processus de standardisation implique la mise à l'échelle des variables. Ce processus est important car il élimine les biais pouvant résulter de l'échelle des variables,



permettant ainsi une comparaison équitable et améliore souvent la performance des algorithmes.

- Min-Max Scaling (Z-Score) : Redimensionne les caractéristiques pour qu'elles se situent entre un minimum et un maximum donné, souvent 0 et 1.
- Z-Score Normalisation : Redimensionne les données pour qu'elles aient une moyenne de 0 et un écart-type de 1.

### **Dilemme Biais-Variance**

Le dilemme biais-variance décrit le problème de trouver le juste équilibre entre l'erreur de biais et l'erreur de variance lors de la construction d'un modèle de machine learning.

Le biais est une erreur due à des hypothèses simplificatrices dans le modèle qui peut causer une sous-performance sur les données d'entraînement et de test.

La variance est l'erreur due à la complexité excessive du modèle qui s'adapte trop bien aux données d'entraînement, mais échoue à généraliser sur de nouvelles données (overfitting).

### **Solutions pour Éviter l'Overfitting**

- Éliminer les caractéristiques qui ont une corrélation négligeable avec la variable cible peut aider à réduire la complexité du modèle et à éviter l'overfitting.
- Des caractéristiques hautement corrélées entre elles ne font qu'ajouter du bruit et de la redondance, ce qui peut conduire à une mauvaise performance sur de nouvelles données.
- Des variables catégorielles avec une faible fréquence ou une variance dans les catégories peuvent ne pas contribuer significativement à la capacité prédictive du modèle et peuvent être omises.
- Intentionnellement augmenter le biais (par exemple, en simplifiant le modèle) peut parfois améliorer la généralisation du modèle en réduisant la variance.

### **Transformation « Dummy »**

La transformation "dummy", ou encodage one-hot, convertit les variables catégorielles en une série de variables binaires (0 ou 1). Chaque catégorie de la variable originale devient une nouvelle variable binaire indiquant la présence (1) ou l'absence (0) de chaque catégorie possible dans les données (`pd.get_dummies()`)

Cela permet de transformer des données textuelles en un format utilisable par des algorithmes de machine learning tout en préservant l'information contenue dans les catégories