**MODULE 2 : Manipulation et Prétraitement des Données**

Le **manipulation et prétraitement des données** est une étape essentielle en science des données et en apprentissage automatique. Elle permet de nettoyer, transformer et structurer les données pour en faciliter l'analyse et améliorer la performance des modèles.

**2.1 Manipulation des Données**

La manipulation des données consiste à modifier, organiser et extraire des informations utiles. Voici quelques techniques courantes :

* **Chargement des données** : Lecture de fichiers CSV, JSON, Excel, bases de données, etc.
* **Filtrage et sélection** : Sélection des colonnes et des lignes pertinentes.
* **Tri et regroupement** : Classement des données et agrégation par catégories.
* **Fusion et jointure** : Combinaison de plusieurs sources de données.

**Exemple avec Python (Pandas)** :

*import pandas as pd*

*# Chargement des données*

*df = pd.read\_csv("data.csv")*

*# Sélection des colonnes pertinentes*

*df = df[['Nom', 'Age', 'Salaire']]*

*# Filtrage des données*

*df = df[df['Age'] > 30]*

*# Regroupement et agrégation*

*df\_grouped = df.groupby("Nom").mean()*

**2.2 Prétraitement des Données**

Le prétraitement prépare les données pour l'analyse et l'apprentissage automatique.

**Nettoyage des Données**

* Gestion des valeurs manquantes (suppression ou imputation).

Les valeurs manquantes sont fréquentes dans les jeux de données et peuvent fausser les analyses ou réduire la performance des modèles. Voici différentes approches pour les traiter.

### ****1. Identifier les valeurs manquantes****

Avant de traiter les valeurs manquantes, il faut d'abord les détecter.

*import pandas as pd*

*# Chargement des données*

*df = pd.read\_csv("data.csv")*

*# Afficher le nombre de valeurs manquantes par colonne*

*print(df.isnull().sum())*

*# Afficher les lignes contenant des valeurs manquantes*

*print(df[df.isnull().any(axis=1)])*

### ****2. Suppression des valeurs manquantes****

Si les valeurs manquantes sont peu nombreuses, on peut simplement supprimer les lignes ou les colonnes concernées.

#### **Supprimer les lignes contenant des valeurs manquantes**

*df\_cleaned = df.dropna()*

#### **Supprimer les colonnes contenant des valeurs manquantes**

*df\_cleaned = df.dropna(axis=1)*

***Remarque* :** Cette approche peut entraîner une perte d'informations importante si trop de données sont supprimées.

### ****3. Imputation des valeurs manquantes****

Quand supprimer des données n’est pas une option, on peut remplacer les valeurs manquantes par des estimations.

#### **Remplacer par une valeur fixe**

*df.fillna(0, inplace=True) # Remplace les valeurs manquantes par 0*

*df.fillna("Inconnu", inplace=True) # Pour les variables catégoriques*

#### **Remplacer par la moyenne, la médiane ou le mode**

*df['Salaire'].fillna(df['Salaire'].mean(), inplace=True) # Moyenne*

*df['Salaire'].fillna(df['Salaire'].median(), inplace=True) # Médiane*

*df['Sexe'].fillna(df['Sexe'].mode()[0], inplace=True) # Mode (valeur la plus fréquente)*

### ****4. Imputation avancée avec Scikit-Learn****

L'imputation basée sur un modèle peut être plus précise.

#### **Imputation avec KNN (K-Nearest Neighbors)**

*from sklearn.impute import KNNImputer*

*imputer = KNNImputer(n\_neighbors=5)*

*df[['Age', 'Salaire']] = imputer.fit\_transform(df[['Age', 'Salaire']])*

#### **Imputation avec Régression**

Une régression peut prédire les valeurs manquantes à partir des autres variables du dataset.

### ****5. Gérer les valeurs manquantes dans un modèle de Machine Learning****

Certains algorithmes comme **XGBoost, LightGBM et CatBoost** gèrent directement les valeurs manquantes, donc il n’est pas toujours nécessaire de les imputer.

* L'approche dépend du volume et de la nature des valeurs manquantes.
* Si peu de valeurs sont manquantes, la suppression peut suffire.
* L’imputation par moyenne/médiane est simple mais peut biaiser(Effet de distorsion ou deviation) les données.
* Des méthodes avancées comme **KNN** ou **régression** peuvent être plus efficaces.
* Suppression des doublons.

Les doublons peuvent fausser les analyses et influencer négativement les performances des modèles. Il est donc important de les détecter et de les supprimer.

### ****1. Détecter les doublons****

Avant de supprimer les doublons, il faut d'abord les identifier.

#### **Vérifier s’il y a des doublons**

*import pandas as pd*

*# Chargement des données*

*df = pd.read\_csv("data.csv")*

*# Vérifier le nombre total de doublons*

*print(df.duplicated().sum())*

Cette commande compte le nombre de lignes dupliquées.

#### **Afficher les lignes en double**

*df[df.duplicated()]*

Cette commande affiche toutes les lignes qui apparaissent plusieurs fois.

### ****2. Supprimer les doublons****

#### **Supprimer les doublons sur toutes les colonnes**

*df\_cleaned = df.drop\_duplicates()*

Cette commande supprime les doublons en gardant la première occurrence.

#### **Supprimer les doublons sur des colonnes spécifiques**

Si seuls certains critères doivent être uniques (exemple : "Nom" et "Email"), on peut filtrer par ces colonnes :

*df\_cleaned = df.drop\_duplicates(subset=['Nom', 'Email'])*

#### **Garder la dernière occurrence au lieu de la première**

*df\_cleaned = df.drop\_duplicates(keep='last')*

Par défaut, keep='first' garde la première occurrence, et keep='last' garde la dernière.  
Si on veut supprimer **toutes** les occurrences en double (y compris la première apparition), on peut utiliser :

*df\_cleaned = df.drop\_duplicates(keep=False)*

### ****3. Suppression des doublons et réindexation****

Après suppression des doublons, il peut être utile de réindexer le DataFrame :

*df\_cleaned.reset\_index(drop=True, inplace=True)*

Cela évite d’avoir des numéros d’index sautés après suppression des doublons.

### ****Conclusion****

* Vérifier la présence de doublons avant de les supprimer.
* Choisir la bonne méthode en fonction du besoin (suppression totale ou partielle).
* Réindexer les données après suppression pour garder un DataFrame propre.
* Correction des incohérences.

Les incohérences dans les données peuvent provenir d'erreurs humaines, de fautes de frappe, de divergences dans les formats ou de données mal saisies. Il est crucial de les identifier et de les corriger pour assurer la fiabilité des analyses et des modèles.

## **1. Identifier les incohérences**

Avant de corriger les incohérences, il faut d'abord les détecter.

### *****Rechercher les valeurs uniques dans une colonne*****

Si une colonne doit contenir des valeurs standardisées (ex. : "Homme" ou "Femme"), vérifier les valeurs présentes :

*print(df['Sexe'].unique())*

Si on trouve des variantes comme "homme", "H", "M", "femme", cela signifie qu’il y a des incohérences.

### *****Détecter les erreurs typographiques*****

On peut afficher les valeurs les plus fréquentes pour voir les erreurs :

*print(df['Ville'].value\_counts())*

Cela permet d'identifier des variantes incorrectes comme "Paris ", "paris", "PAR1S".

### *****Rechercher des valeurs aberrantes (outliers)*****

Si une colonne doit contenir des valeurs dans un certain intervalle (ex. : âge entre 0 et 120 ans), on peut vérifier :

*print(df[df['Age'] < 0]) # Âges négatifs*

*print(df[df['Salaire'] > 1\_000\_000]) # Valeurs trop élevées*

## **2. Correction des incohérences**

### *****Normaliser les formats de texte*****

*df['Sexe'] = df['Sexe'].str.lower().str.strip() # Convertir en minuscule et supprimer les espaces*

*df['Sexe'] = df['Sexe'].replace({'homme': 'H', 'femme': 'F', 'h': 'H', 'f': 'F'}) # Harmoniser les valeurs*

### *****Corriger les fautes de frappe avec***** *fuzzywuzzy*

*from fuzzywuzzy import process*

*# Correction des erreurs typographiques dans la colonne "Ville"*

*df['Ville'] = df['Ville'].apply(lambda x: process.extractOne(x, ['Paris', 'Lyon', 'Marseille'])[0])*

### *****Remplacer les valeurs aberrantes par des valeurs plausibles*****

# Remplacer les âges négatifs par la médiane

*df.loc[df['Age'] < 0, 'Age'] = df['Age'].median()*

# Truncature des salaires trop élevés

*df.loc[df['Salaire'] > 1\_000\_000, 'Salaire'] = df['Salaire'].quantile(0.99) # Remplace par le 99e centile*

### *****Convertir les formats de dates et corriger les erreurs*****

*df['Date\_naissance'] = pd.to\_datetime(df['Date\_naissance'], errors='coerce') # Convertir en format date*

errors='coerce' remplace les dates invalides par NaT (valeur manquante).

## **Conclusion**

* Détecter les incohérences avant de les corriger.
* Standardiser les textes et corriger les fautes de frappe.
* Identifier et traiter les valeurs aberrantes.
* Assurer la cohérence des formats (dates, nombres, catégories).
* Encodage des variables catégoriques.

Les modèles de Machine Learning fonctionnent généralement mieux avec des valeurs numériques. Les variables catégoriques (ex. : "Rouge", "Bleu", "Vert" ou "Femme", "Homme") doivent donc être converties en nombres à l'aide de méthodes d'encodage.

## **1. Identifier les variables catégoriques**

Avant d’encoder, on peut repérer les variables non numériques dans un DataFrame :

*import pandas as pd*

*df = pd.read\_csv("data.csv")*

*# Afficher les types de colonnes*

*print(df.dtypes)*

*# Lister les colonnes catégoriques*

*cat\_cols = df.select\_dtypes(include=['object']).columns*

*print(cat\_cols)*

## **2. Méthodes d'encodage**

### *****Encodage Ordinal (Label Encoding)*****

* Convient aux variables catégoriques avec un ordre logique (ex. : "Faible", "Moyen", "Élevé").
* Chaque catégorie est convertie en un nombre unique.

*from sklearn.preprocessing import LabelEncoder*

*encoder = LabelEncoder()*

*df['Niveau'] = encoder.fit\_transform(df['Niveau'])*

**Exemple** :

| **Niveau** | **Encodé** |
| --- | --- |
| Faible | 0 |
| Moyen | 1 |
| Élevé | 2 |

Ce type d'encodage peut introduire un biais si la variable n’a pas d’ordre réel.

### *****Encodage One-Hot (One-Hot Encoding)*****

* Convient aux variables sans ordre logique (ex. : "Rouge", "Bleu", "Vert").
* Crée une colonne binaire pour chaque catégorie.

*df = pd.get\_dummies(df, columns=['Couleur'], drop\_first=True)*

**Exemple** :

| **Couleur** | **Rouge** | **Vert** |
| --- | --- | --- |
| Bleu | 0 | 0 |
| Rouge | 1 | 0 |
| Vert | 0 | 1 |

**Inconvénient** : Augmente la dimensionnalité du dataset si la variable a beaucoup de catégories.

### ****Encodage Fréquentiel (Frequency Encoding)****

Remplace chaque catégorie par sa fréquence d’apparition dans le dataset.

*df['Categorie\_freq'] = df['Categorie'].map(df['Categorie'].value\_counts() / len(df))*

**Exemple** :

| **Catégorie** | **Fréquence** |
| --- | --- |
| A | 0.4 |
| B | 0.35 |
| C | 0.25 |

### ****Encodage avec Target Encoding (Mean Encoding)****

* Remplace chaque catégorie par la moyenne de la variable cible (utile pour les modèles supervisés).

*df['Categorie\_encoded'] = df.groupby('Categorie')['Revenu'].transform('mean')*

**Exemple** :

| **Catégorie** | **Revenu Moyen** |
| --- | --- |
| A | 30,000 |
| B | 50,000 |
| C | 70,000 |

Peut entraîner du surapprentissage si mal utilisé.

## **Conclusion**

* **Label Encoding** : Pour les catégories ordinales.
* **One-Hot Encoding** : Pour les catégories nominales avec peu de valeurs uniques.
* **Frequency Encoding** : Pour éviter la haute dimensionnalité.
* **Target Encoding** : Utile pour les modèles supervisés, mais risque de surapprentissage.

**Manipulation des Données avec Pandas**

Pandas est une bibliothèque Python essentielle pour la manipulation et l'analyse des données. Elle permet de charger, filtrer, trier et transformer facilement des jeux de données.

## **1. Chargement des Données**

Pandas permet de lire différents formats de fichiers comme **CSV, Excel, JSON, SQL**, etc.

### ****Lire un fichier CSV****

*import pandas as pd*

*df = pd.read\_csv("data.csv") # Charger un fichier CSV*

*print(df.head()) # Afficher les 5 premières lignes*

### ****Lire un fichier Excel****

*df = pd.read\_excel("data.xlsx", sheet\_name="Feuille1")*

### ****Lire un fichier JSON****

df = pd.read\_json("data.json")

## **2. Exploration des Données**

### *****Afficher des informations générales*****

*print(df.info()) # Type de données et valeurs manquantes*

*print(df.describe()) # Statistiques sur les colonnes numériques*

### *****Aperçu des données*****

*print(df.head(3)) # Afficher les 3 premières lignes*

*print(df.tail(3)) # Afficher les 3 dernières lignes*

*print(df.sample(3)) # Afficher 3 lignes aléatoires*

### *****Vérifier les valeurs uniques d'une colonne*****

*print(df['Categorie'].unique()) # Valeurs uniques dans la colonne "Categorie"*

*print(df['Categorie'].value\_counts()) # Nombre d'occurrences par catégorie*

## **3. Sélection et Filtrage des Données**

### *****Sélection de colonnes*****

*df\_subset = df[['Nom', 'Age', 'Salaire']] # Sélectionner certaines colonnes*

### *****Filtrer selon une condition*****

*df\_filtre = df[df['Age'] > 30] # Garde les lignes où l'âge est > 30*

*df\_filtre = df[(df['Age'] > 30) & (df['Sexe'] == 'Homme')] # Condition multiple*

### *****Sélection avec***** *loc* *****et***** *iloc*

print(df.loc[2]) # Sélectionner la ligne avec l'index 2

print(df.iloc[2, 1]) # Sélectionner l'élément à la 2e ligne et 1re colonne

## **4. Transformation des Données**

### *****Renommer des colonnes*****

*df.rename(columns={'AncienNom': 'NouveauNom'}, inplace=True)*

### *****Ajouter une nouvelle colonne*****

*df['Salaire\_Annuel'] = df['Salaire'] \* 12 # Création d’une colonne calculée*

### ****Remplacer des valeurs****

*df['Sexe'] = df['Sexe'].replace({'H': 'Homme', 'F': 'Femme'})*

### ****Modifier le type de données****

*df['Date'] = pd.to\_datetime(df['Date']) # Convertir en format date*

*df['Age'] = df['Age'].astype(int) # Convertir en entier*

## **5. Traitement des Valeurs Manquantes et des Doublons**

### *****Vérifier les valeurs manquantes*****

*print(df.isnull().sum()) # Nombre de valeurs manquantes par colonne*

### *****Supprimer les valeurs manquantes*****

*df\_cleaned = df.dropna() # Supprime les lignes contenant des valeurs NaN*

### *****Remplacer les valeurs manquantes*****

*df['Salaire'].fillna(df['Salaire'].mean(), inplace=True) # Remplacer par la moyenne*

### *****Supprimer les doublons*****

*df = df.drop\_duplicates() # Supprime les lignes dupliquées*

## **6. Regroupement et Agrégation**

### *****Regrouper les données et calculer des statistiques*****

*df\_grouped = df.groupby('Categorie')['Salaire'].mean() # Moyenne des salaires par catégorie*

*print(df\_grouped)*

### *****Appliquer plusieurs fonctions d'agrégation*****

*df\_agg = df.groupby('Categorie').agg({'Salaire': ['mean', 'min', 'max']})*

*print(df\_agg)*

## **7. Trier et Réorganiser les Données**

### *****Trier les données*****

*df\_sorted = df.sort\_values(by='Salaire', ascending=False) # Trier par salaire décroissant*

### *****Réindexer le DataFrame*****

*df.reset\_index(drop=True, inplace=True) # Réinitialiser l'index*

## **Conclusion**

* **Charger** et explorer les données facilement avec Pandas.
* **Filtrer** et **transformer** les informations en fonction des besoins.
* **Gérer les valeurs manquantes et les doublons** pour un dataset propre.
* **Regrouper et analyser** les données avec des agrégations efficaces.

# **Visualisation des Données en Python**

La visualisation des données est essentielle pour comprendre les tendances, détecter les valeurs aberrantes et explorer les relations entre les variables.

## **1. Graphiques de base avec Matplotlib**

Matplotlib est la bibliothèque de base pour la création de graphiques en Python.

### *****Importer Matplotlib*****

*import matplotlib.pyplot as plt*

*import numpy as np*

*import pandas as pd*

*# Exemple de dataset*

*df = pd.DataFrame({*

*'Catégorie': ['A', 'B', 'C', 'D'],*

*'Valeurs': [23, 45, 56, 78]*

*})*

### *****Histogramme*****

*Utilisé pour afficher la distribution d'une variable numérique.*

*plt.hist(df['Valeurs'], bins=5, color='skyblue', edgecolor='black')*

*plt.xlabel("Valeurs")*

*plt.ylabel("Fréquence")*

*plt.title("Histogramme des valeurs")*

*plt.show()*

### *****Graphique en barres*****

Utile pour comparer des catégories.

*plt.bar(df['Catégorie'], df['Valeurs'], color=['red', 'blue', 'green', 'purple'])*

*plt.xlabel("Catégorie")*

*plt.ylabel("Valeurs")*

*plt.title("Graphique en barres")*

*plt.show()*

### *****Nuage de points (*Scatter Plot)****

Utile pour visualiser la relation entre deux variables.

*x = np.random.rand(50)*

*y = np.random.rand(50)*

*plt.scatter(x, y, color='blue', alpha=0.5)*

*plt.xlabel("Variable X")*

*plt.ylabel("Variable Y")*

*plt.title("Nuage de points")*

*plt.show()*

### *****Courbe (Ligne)*****

Utilisée pour montrer l’évolution des valeurs.

*plt.plot(df['Catégorie'], df['Valeurs'], marker='o', linestyle='-', color='blue')*

*plt.xlabel("Catégorie")*

*plt.ylabel("Valeurs")*

*plt.title("Courbe des valeurs")*

*plt.show()*

## **2. Graphiques avancés avec Seaborn**

Seaborn est une bibliothèque basée sur Matplotlib qui permet de créer des visualisations plus complexes et plus esthétiques.

*import seaborn as sns*

### *****Histogramme amélioré (Distribution Plot)*****

*sns.histplot(df['Valeurs'], bins=5, kde=True, color='blue')*

*plt.title("Distribution des valeurs")*

*plt.show()*

### *****Boxplot (Boîte à moustaches)*****

Utile pour détecter les valeurs aberrantes.

*sns.boxplot(x=df['Valeurs'], color='orange')*

*plt.title("Boxplot des valeurs")*

*plt.show()*

### *****Graphique en barres amélioré*****

*sns.barplot(x='Catégorie', y='Valeurs', data=df, palette='viridis')*

*plt.title("Graphique en barres avec Seaborn")*

*plt.show()*

### *****Nuage de points avec régression*****

*df = pd.DataFrame({'X': np.random.rand(50), 'Y': np.random.rand(50)})*

*sns.regplot(x='X', y='Y', data=df)*

*plt.title("Nuage de points avec régression")*

*plt.show()*

### *****Matrice de corrélation (Heatmap)*****

*# Création d'un dataset aléatoire*

*df\_corr = pd.DataFrame(np.random.rand(10, 4), columns=['A', 'B', 'C', 'D'])*

*# Calcul de la corrélation*

*corr = df\_corr.corr()*

*# Affichage de la heatmap*

*sns.heatmap(corr, annot=True, cmap="coolwarm", linewidths=0.5)*

*plt.title("Matrice de corrélation")*

*plt.show()*

## **3. Interprétation des Graphiques pour l’Analyse Exploratoire**

Voici quelques observations que l’on peut tirer des visualisations :

* **Histogramme** → Permet d’identifier la distribution des données (normale, asymétrique, bimodale, etc.).
* **Boxplot** → Utile pour détecter les valeurs aberrantes et voir la dispersion des données.
* **Nuage de points** → Permet d’examiner les relations entre deux variables (corrélation positive ou négative).
* **Graphique en barres** → Compare des catégories et montre leur répartition.
* **Heatmap (corrélation)** → Met en évidence les relations fortes entre les variables d’un dataset.

## **Conclusion**

**Matplotlib** pour des graphiques simples et personnalisables.  
**Seaborn** pour des visualisations plus avancées et esthétiques.  
**L’interprétation des graphiques** permet de mieux comprendre les tendances et anomalies dans un dataset.

# **Les Fonctions NumPy Utiles en Data Science**

NumPy est l'une des bibliothèques les plus utilisées en data science pour la manipulation des données numériques. Voici une liste des fonctions les plus couramment utilisées en data science, avec des exemples pratiques.

## **1. Création de Tableaux et Manipulation des Données**

### ****Créer un tableau NumPy à partir d’une liste ou d’une séquence****

*import numpy as np*

*# Créer un tableau simple*

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

*# Créer un tableau avec des séquences*

*arr\_range = np.arange(0, 10, 2) # [0, 2, 4, 6, 8]*

*# Créer un tableau avec des nombres également espacés*

*arr\_linspace = np.linspace(0, 1, 5) # [0.0, 0.25, 0.5, 0.75, 1.0]*

### *****Générer des matrices spéciales*****

*# Matrice de zéros*

*zeros\_array = np.zeros((3, 3))*

*# Matrice de uns*

*ones\_array = np.ones((2, 4))*

*# Matrice identité*

*identity\_matrix = np.eye(3)*

### *****Utiliser des valeurs aléatoires pour les simulations*****

*# Générer des nombres aléatoires entre 0 et 1*

*rand\_array = np.random.rand(3, 3)*

*# Générer des entiers aléatoires*

*rand\_int\_array = np.random.randint(1, 100, (3, 3))*

*# Fixer la graine pour la reproductibilité*

*np.random.seed(42)*

## **2. Statistiques et Calculs Mathématiques**

### *****Calcul des mesures statistiques*****

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

*# Moyenne, médiane, écart-type*

*mean = np.mean(arr) # 30.0*

*median = np.median(arr) # 30.0*

*std\_dev = np.std(arr) # 14.142*

### *****Calcul de la somme, produit et cumul*****

*# Somme des éléments*

*sum\_arr = np.sum(arr) # 150*

*# Produit des éléments*

*prod\_arr = np.prod(arr) # 12000000*

*# Somme cumulée*

*cumsum\_arr = np.cumsum(arr) # [10 30 60 100 150]*

### *****Calcul de la variance et des percentiles*****

*variance = np.var(arr) # 200.0*

*percentiles = np.percentile(arr, [25, 50, 75]) # [20.0 30.0 40.0]*

### *****Max, min, et index des valeurs extrêmes*****

*max\_value = np.max(arr) # 150*

*min\_value = np.min(arr) # 10*

*# Trouver l'indice de la valeur maximale*

*index\_max = np.argmax(arr) # 4*

## **3. Manipulation des Tableaux**

### *****Redimensionner (reshape) un tableau*****

*arr = np.arange(1, 10) # Créer un tableau [1, 2, 3, 4, 5, 6, 7, 8, 9]*

*# Redimensionner en 3x3*

*reshaped\_arr = arr.reshape(3, 3)*

### *****Aplatir un tableau multidimensionnel*****

arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

flattened\_arr = arr\_2d.flatten() # [1 2 3 4 5 6 7 8 9]

### *****Indexation et slicing (découper un tableau)*****

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

*# Slicing*

*slice\_arr = arr[1:4] # [20, 30, 40]*

*# Indexation conditionnelle*

*filtered\_arr = arr[arr > 25] # [30, 40, 50]*

### *****Concaténer et empiler des tableaux*****

*arr1 = np.array([1, 2, 3])*

*arr2 = np.array([4, 5, 6])*

*# Concaténer*

*concat\_arr = np.concatenate((arr1, arr2)) # [1 2 3 4 5 6]*

*# Empiler verticalement*

*vstack\_arr = np.vstack((arr1, arr2)) # [[1 2 3], [4 5 6]]*

*# Empiler horizontalement*

*hstack\_arr = np.hstack((arr1, arr2)) # [1 2 3 4 5 6]*

## **4. Opérations sur les Matrices**

### *****Produit matriciel (dot product)*****

*A = np.array([[1, 2], [3, 4]])*

*B = np.array([[5, 6], [7, 8]])*

*# Produit matriciel*

*mat\_mul = np.dot(A, B) # [[19 22], [43 50]]*

### *****Transposer une matrice*****

*transposed\_A = A.T # [[1 3], [2 4]]*

### *****Inverse d'une matrice (utilisation de***** *linalg******)*****

# Matrice carrée

matrix = np.array([[1, 2], [3, 4]])

# Inverse de la matrice

inv\_matrix = np.linalg.inv(matrix) # [[-2. 1. ], [1.5 -0.5]]

### *****Calcul du déterminant d'une matrice*****

*det\_matrix = np.linalg.det(matrix) # -2.0*

## **5. Algèbre Linéaire et Fonctions Avancées**

### *****Valeurs et vecteurs propres*****

*eig\_vals, eig\_vecs = np.linalg.eig(matrix) # Valeurs et vecteurs propres*

### *****Résolution de systèmes d’équations linéaires*****

*# Résoudre Ax = b*

*A = np.array([[2, 1], [1, 3]])*

*b = np.array([8, 18])*

*x = np.linalg.solve(A, b) # Solution du système*

### *****Calculs sur les matrices (Trace)*****

*trace\_matrix = np.trace(A) # Somme des éléments diagonaux*

## **6. Trigonometric Functions et Fonctions Avancées**

### *****Fonctions trigonométriques*****

*angles = np.array([0, np.pi/2, np.pi])*

*# Sinus, cosinus, tangente*

*sin\_vals = np.sin(angles) # [0. 1. 0.]*

*cos\_vals = np.cos(angles) # [1. 0. -1.]*

*tan\_vals = np.tan(angles) # [0. 1. 0.]*

### *****Fonctions exponentielles et logarithmes*****

*# Exponentielle*

*exp\_vals = np.exp(arr) # [2.71828, 7.38906, ...]*

*# Logarithme naturel*

*log\_vals = np.log(arr) # [2.30258, 2.99573, ...]*

*# Logarithme en base 10*

*log10\_vals = np.log10(arr) # [1. 1.3010, 1.4771, ...]*

## **7. Applications en Data Science**

### *****Gestion des valeurs manquantes*****

*# Création d’un tableau avec des NaN*

*arr = np.array([1, np.nan, 3, np.nan, 5])*

*# Remplacer les NaN par une valeur (par exemple la moyenne)*

*arr[np.isnan(arr)] = np.nanmean(arr) # Remplacer NaN par la moyenne*

### *****Normalisation et standardisation des données*****

*# Normaliser les données entre 0 et 1*

*arr\_minmax = (arr - np.min(arr)) / (np.max(arr) - np.min(arr))*

*# Standardiser les données (moyenne 0, écart-type 1)*

*arr\_standardized = (arr - np.mean(arr)) / np.std(arr)*

## **Conclusion**

NumPy est un outil essentiel en **Data Science** pour la manipulation et le calcul de données numériques. Il est extrêmement utile pour les tâches de manipulation de matrices, les statistiques, l'algèbre linéaire, et plus encore.