Numpy Tp

Djebabla Ammar

Contents

1 Primitives importantes de Numpy

Numpy offre une variété de fonctions utiles pour manipuler des données. Voici 20 primitives courantes regroupées par catégories, avec un exemple pour chacune.

1.1 Statistiques de base

- `mean()` : Calcule la moyenne des valeurs.

```
import numpy as np
data = np.array([1, 2, 3, 4])
mean_value = data.mean()
print("Moyenne :", mean_value)  # Output: 2.5
```

Listing 1: Exemple : Calcul de la moyenne

- `sum()` : Calcule la somme des valeurs.

```
data = np.array([1, 2, 3, 4])
total = data.sum()
print("Somme :", total)  # Output: 10
```

Listing 2: Exemple : Calcul de la somme

- `std()` : Calcule l'écart type.

```
data = np.array([1, 2, 3, 4])
std_dev = data.std()
print(" cart  type :", std_dev)  # Output: 1.118...
```

Listing 3: Exemple : Calcul de l'écart type

- `var()` : Calcule la variance.

```
data = np.array([1, 2, 3, 4])
variance = data.var()
print("Variance :", variance)  # Output: 1.25
```

Listing 4: Exemple : Calcul de la variance

1.2 Manipulation des tableaux

- `reshape()` : Change la forme d'un tableau.

```
data = np.array([1, 2, 3, 4, 5, 6])
reshaped = data.reshape(2, 3)
print(reshaped)
# Output:
# [[1 2 3]
#  [4 5 6]]
```

Listing 5: Exemple : Reshape d'un tableau

- `flatten()` : Transforme un tableau multidimensionnel en une seule dimension.

```
data = np.array([[1, 2], [3, 4]])
flattened = data.flatten()
print(flattened)  # Output: [1 2 3 4]
```

Listing 6: Exemple : Flatten d'un tableau

- `transpose()` : Transpose un tableau.

```python
data = np.array([[1, 2], [3, 4]])
transposed = data.transpose()
print(transposed)
# Output:
# [[1 3]
#  [2 4]]
```

Listing 7: Exemple : Transposition

- `hstack()` et `vstack()` : Combine des tableaux horizontalement ou verticalement.

```python
data1 = np.array([1, 2])
data2 = np.array([3, 4])
h_combined = np.hstack([data1, data2])  # [1 2 3 4]
v_combined = np.vstack([data1, data2])
# [[1 2]
#  [3 4]]
```

Listing 8: Exemple : hstack et vstack

- `-1` : Utilisé pour accéder au dernier élément ou dimension dans un tableau.

```python
data = np.array([10, 20, 30, 40])
last_element = data[-1]
print(last_element)  # Output: 40
```

Listing 9: Exemple : Utilisation de -1 pour accéder au dernier élément

- `::` : Utilisé pour effectuer des tranches (slicing) dans un tableau.

```python
data = np.array([10, 20, 30, 40, 50])
slice_data = data[::2]  # S lectionne tous les deux    lments
print(slice_data)  # Output: [10 30 50]
```

Listing 10: Exemple : Utilisation de :: pour une tranche

- `copy()` : Crée une copie d'un tableau, indépendante du tableau original.

```python
data = np.array([1, 2, 3, 4])
copied_data = data.copy()
copied_data[0] = 99
print("Original :", data)  # Output: [1 2 3 4]
print("Copi  :", copied_data)  # Output: [99  2  3  4]
```

Listing 11: Exemple : Utilisation de copy()

La fonction `copy()` permet de créer une copie profonde d'un tableau. Cela signifie que les modifications apportées à la copie n'affecteront pas l'original, contrairement à l'assignation directe où les deux variables pointent vers la même mémoire. Le `copy()` est essentiel pour éviter les effets de bord quand on travaille avec des pointeurs en Python.

1.3 Affichage des premiers et derniers éléments du dataset

Pour afficher les 10 premiers éléments d'un dataset NumPy, on utilise `array[:10]`. Pour afficher les 10 derniers éléments, on utilise `array[-10:]`.

python import numpy as np

Création d'un dataset NumPy dataset = np.arange(1, 101)

Affichage des 10 premiers et 10 derniers éléments print("Les 10 premiers éléments :", dataset[:10]) print("Les 10 derniers éléments :", dataset[-10:])

1.4 Informations supplémentaires sur le dataset

Voici d'autres informations utiles que vous pouvez afficher pour mieux comprendre un dataset NumPy :

- **Taille** : Le nombre total d'éléments dans le dataset avec `dataset.size`.
- **Forme** : Les dimensions du dataset avec `dataset.shape`.
- **Type de données** : Le type des éléments du dataset avec `dataset.dtype`.
- **Valeur minimale** : La valeur la plus petite avec `dataset.min()`.
- **Valeur maximale** : La valeur la plus grande avec `dataset.max()`.
- **Moyenne** : La moyenne des éléments avec `dataset.mean()`.

python Taille du dataset print("Taille du dataset :", dataset.size)
Forme du dataset print("Forme du dataset :", dataset.shape)
Type de données print("Type de données :", dataset.dtype)
Valeur minimale et maximale print("Valeur minimale :", dataset.min()) print("Valeur maximale :", dataset.max())
Moyenne print("Moyenne :", dataset.mean())

1.5 Indexation avancée

- `Fancy Indexing` : Sélectionne des éléments en utilisant une liste d'indices.

```
data = np.array([10, 20, 30, 40, 50])
indices = [0, 2, 4]
selected = data[indices]
print(selected)  # Output: [10 30 50]
```

Listing 12: Exemple : Fancy Indexing

- `masking` : Sélectionne des éléments en fonction d'une condition.

```
data = np.array([1, 2, 3, 4, 5])
filtered = data[data > 3]
print(filtered)  # Output: [4 5]
```

Listing 13: Exemple : Masking

- `where()` : Renvoie les indices où une condition est remplie.

```
data = np.array([10, 20, 30, 40, 50])
condition = data > 30
result = np.where(condition)
print(result)  # Output: (array([3, 4]),)
```

Listing 14: Exemple : Utilisation de where()

La fonction `where()` renvoie les indices des éléments qui satisfont la condition spécifiée. Cela est utile pour effectuer des opérations conditionnelles sur un tableau.

`Sélection d'une catégorie spécifique` : Utilisation de conditions pour filtrer un tableau en fonction d'une catégorie spécifique.

```
import numpy as np
data = np.array([
    {"Nom": "Alice", "Classe": "Guerrier", "Niveau": 5},
    {"Nom": "Bob", "Classe": "Mage", "Niveau": 3},
    {"Nom": "Charlie", "Classe": "Guerrier", "Niveau": 7},
])

# S lectionner tous les guerriers
guerriers = data[data["Classe"] == "Guerrier"]
print(guerriers)
```

Listing 15: Exemple : Sélectionner une catégorie spécifique

1.6 Charger un fichier CSV avec `genfromtxt`

Voici un exemple de code pour charger un fichier CSV en utilisant la fonction `genfromtxt`. Cette fonction permet de lire un fichier CSV et de charger les données sous forme de tableau `NumPy`.

```
import numpy as np

# Charger le fichier CSV
data = np.genfromtxt('votre_fichier.csv', delimiter=',', names=True, dtype=None, encoding=None)

# Afficher les 5 premiers    lments
print("Premiers    lments   :")
print(data[:5])

# Afficher les 5 derniers    lments
print("\nDerniers    lments   :")
print(data[-5:])

# Afficher les noms des colonnes (features)
print("\nNoms des colonnes:")
print(data.dtype.names)
```

Listing 16: Charger un fichier CSV

1.7 Agrégations et calculs par groupes

- `unique()` : Renvoie les valeurs uniques d'un tableau.

```
data = np.array([1, 2, 2, 3, 3, 3])
unique_values = np.unique(data)
print(unique_values)  # Output: [1 2 3]
```

Listing 17: Exemple : Valeurs uniques

- `sort()` : Trie un tableau.

```
data = np.array([4, 2, 3, 1])
sorted_data = np.sort(data)
print(sorted_data)  # Output: [1 2 3 4]
```

Listing 18: Exemple : Tri d'un tableau

- `argsort()` : Renvoie les indices pour trier un tableau.

```
data = np.array([4, 2, 3, 1])
sorted_indices = np.argsort(data)
print(sorted_indices)  # Output: [3 1 2 0]
```

Listing 19: Exemple : Indices de tri

1.8 Calculs mathématiques avancés

- `dot()` : Calcule le produit scalaire ou matriciel.

```
data1 = np.array([1, 2])
data2 = np.array([3, 4])
dot_product = np.dot(data1, data2)
print(dot_product)  # Output: 11
```

Listing 20: Exemple : Produit matriciel

- `corrcoef()` : Calcule la corrélation entre deux tableaux.

```
data1 = np.array([1, 2, 3])
data2 = np.array([4, 5, 6])
correlation = np.corrcoef(data1, data2)
print(correlation)
# Output: Matrice de corr lation
```

Listing 21: Exemple : Corrélation

- `linspace()` : Génère des valeurs régulièrement espacées dans un intervalle donné.

```
data = np.linspace(0, 10, 5)
print(data)  # Output: [ 0.   2.5  5.   7.5 10. ]
```

Listing 22: Exemple : Génération d'une gamme de valeurs

2 Primitives importantes de Matplotlib

Matplotlib est une bibliothèque de visualisation de données en Python, utilisée pour créer des graphiques statiques, animés ou interactifs. Voici les primitives les plus courantes regroupées par catégorie, avec un exemple pour chacune.

2.1 1. Création de graphiques de base

- `plot()` : Crée un graphique linéaire.
- `scatter()` : Crée un graphique de dispersion.
- `bar()` : Crée un diagramme en barres.
- `hist()` : Crée un histogramme.

Exemple : Création d'un graphique linéaire avec `plot()`

```
import matplotlib.pyplot as plt

# Donn es
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

# Cr ation du graphique
plt.plot(x, y)

#   tiquettes   des axes
plt.xlabel("X - Axe")
plt.ylabel("Y - Axe")

# Affichage
plt.title("Exemple de graphique lin aire")
plt.show()
```

Listing 23: Création d'un graphique linéaire avec Matplotlib

2.2 2. Gestion des axes et légendes

- `xlabel()` : Ajoute une étiquette à l'axe des abscisses.
- `ylabel()` : Ajoute une étiquette à l'axe des ordonnées.
- `title()` : Ajoute un titre au graphique.
- `legend()` : Ajoute une légende au graphique.

Exemple : Ajouter des étiquettes, un titre et une légende

```
import matplotlib.pyplot as plt

# Donn es
x = [1, 2, 3, 4]
y1 = [10, 20, 25, 30]
y2 = [5, 15, 20, 25]

# Cr ation des graphiques
plt.plot(x, y1, label="S rie 1")
plt.plot(x, y2, label="S rie 2")

# Configuration des axes et de la l gende
plt.xlabel("X - Axe")
plt.ylabel("Y - Axe")
plt.title("Exemple avec une l gende")
plt.legend()

# Affichage
plt.show()
```

Listing 24: Ajout d'étiquettes et de légendes

2.3 3. Graphique de dispersion (*Scatter Plot*)

- `scatter()` : Permet de tracer un graphique de dispersion.

Exemple : Création d'un graphique de dispersion

```
import matplotlib.pyplot as plt

# Donn es
x = [1, 2, 3, 4, 5]
y = [2, 4, 1, 3, 7]

# Cr ation du graphique de dispersion
plt.scatter(x, y)

# Configuration des axes
plt.xlabel("Nombre de parties jou es")
plt.ylabel("Nombre de victoires")
plt.title("Relation entre parties jou es et victoires")

# Affichage
plt.show()
```

Listing 25: Création d'un graphique de dispersion

2.4 4. Autres primitives utiles

- `savefig()` : Enregistre le graphique sous forme de fichier.
- `grid()` : Ajoute une grille au graphique.
- `xticks()` et `yticks()` : Personnalise les graduations des axes.

Exemple : Ajouter une grille et enregistrer le graphique

```
import matplotlib.pyplot as plt

# Donn es
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

# Cr ation du graphique
```

```
plt.plot(x, y)

# Configuration
plt.grid(True)
plt.title("Graphique avec grille")
plt.xlabel("X - Axe")
plt.ylabel("Y - Axe")

# Enregistrement
plt.savefig("graphique.png")

# Affichage
plt.show()
```

Listing 26: Ajout d'une grille et enregistrement

2.5 Résumé

Ces primitives constituent une base solide pour créer et personnaliser des graphiques avec Matplotlib. N'hésitez pas à explorer la documentation officielle pour découvrir des fonctionnalités plus avancées.

3 Exercice : Analyse des données des joueurs

3.1 Dataset pour les joueurs de MMORPG

Vous travaillez sur un dataset fictif qui regroupe des informations détaillées sur des joueurs de MMORPG. Ce dataset représente un groupe de gamers passionnés par ce type de jeu. Pour explorer les données et les utiliser dans vos analyses, cliquez sur le lien suivant ::

Dataset des joueurs de MMORPG. https://www.kaggle.com/datasets/ammardjebabla/gamers
Le dataset contient les colonnes suivantes :

- `pseudo` : Le pseudo du joueur.
- `nb_parties` : Nombre de parties jouées.
- `nb_victoires` : Nombre de victoires.
- `team` : Équipe du joueur.
- `class` : Classe du personnage (*guerrier, archer, etc.*).

3.2 Questions

1. **Déterminer le meilleur joueur dans la classe `Guerrier`.** *Indice : Filtrez le dataset pour ne garder que les joueurs de la classe `Guerrier`, puis trouvez celui ayant le maximum de victoires.*
   - **Primitives utiles :**
     - Utiliser `numpy.where` ou `numpy.argmax` pour filtrer et trouver le joueur avec le maximum de victoires dans la classe `Guerrier`.
2. **Déterminer le meilleur joueur dans la classe `Archer`.** *Indice : Répétez la même méthode que pour la classe `Guerrier`, mais en filtrant sur `Archer`.*
   - **Primitives utiles :**
     - Utiliser `numpy.where` ou `numpy.argmax` pour filtrer et trouver le joueur avec le maximum de victoires dans la classe `Archer`.
3. **Afficher toutes les équipes (`team`) disponibles dans le dataset.** *Indice : Répétez la même méthode que pour les classes, mais appliquez-la à la colonne `team`.*
   - **Primitives utiles :**
     - Utiliser `numpy.unique` pour obtenir les équipes uniques.
4. **Déterminer la meilleure équipe (`team`) ayant le maximum de victoires cumulées.** *Indice : Regroupez les données par équipe, puis calculez la somme des victoires pour chaque équipe.*
   - **Primitives utiles :**
     - Utiliser `numpy.unique` pour identifier les équipes uniques.
     - Utiliser `numpy.sum` pour calculer la somme des victoires par équipe.
     - Utiliser `numpy.argmax` pour trouver l'équipe avec le maximum de victoires cumulées.
5. **Identifier le meilleur joueur global (le joueur avec le plus de victoires).** *Indice : Trouvez le joueur ayant la valeur maximale dans la colonne `nb_victoires`.*
   - **Primitives utiles :**
     - Utiliser `numpy.argmax` pour trouver le joueur avec le maximum de victoires.
6. **Afficher toutes les classes disponibles dans le dataset.** *Indice : Utilisez une fonction permettant d'identifier les valeurs uniques dans une colonne spécifique.*
   - **Primitives utiles :**
     - Utiliser `numpy.unique` pour obtenir les classes uniques.

3.3 Évaluation à soumettre

7. **Y a-t-il une relation entre le nombre de parties jouées et le nombre de victoires ?** *Indice : Utilisez une mesure statistique comme la corrélation pour évaluer la relation entre ces deux colonnes.*
   - **Primitives utiles :**
     * Utiliser `numpy.corrcoef` pour calculer la corrélation entre les colonnes `nb_parties` et `nb_victoires`.
8. **Y a-t-il une corrélation entre la classe (`class`) et le nombre de victoires ?** *Indice : Identifiez si certaines classes ont tendance à avoir plus de victoires en moyenne. Une analyse qualitative ou des regroupements peuvent aider.*
   - **Primitives utiles :**
     * Utiliser `numpy.unique` pour identifier les classes uniques.
     * Calculer les moyennes des victoires pour chaque classe en utilisant `numpy.mean`.
9. **Représenter graphiquement la relation entre le nombre de parties jouées et le nombre de victoires.** *Indice : Créez un graphique de dispersion (scatter plot) avec `matplotlib`. Assurez-vous que les axes sont correctement étiquetés.*
   - **Primitives utiles :**
     * Utiliser `matplotlib.pyplot.scatter` pour créer un graphique de dispersion.
     * Utiliser `matplotlib.pyplot.xlabel` et `matplotlib.pyplot.ylabel` pour étiqueter les axes.