Techniques avancées pour NumPy

Maîtriser NumPy : Le compagnon indispensable pour les Data Lovers · https://moncoachdata.com/tutos/techniques-avancees-pour-numpy/

Ce tuto présente deux **techniques avancées pour NumPy** : les tableaux masqués, pour ignorer les valeurs invalides ou manquantes, et les tableaux structurés, pour stocker des données de types différents dans un même tableau.

Les tableaux masqués

Les tableaux masqués (*masked arrays*) te permettent de travailler avec des données dont certains éléments sont invalides ou manquants. Un masque est un tableau de booléens qui indique quels éléments sont à ignorer. Tu peux ainsi faire des calculs sur les valeurs valides uniquement.

Dans cet exemple, des températures contiennent des valeurs manquantes codées par `-999`. Sans traitement, elles faussent complètement la moyenne :

```python
import numpy as np
import numpy.ma as ma

# Températures avec valeurs manquantes (-999 représente une valeur manquante)
temperatures = np.array([22.5, 23.0, -999, 24.5, -999, 26.0, 27.2, -999, 28.5])

# Moyenne sans tenir compte des valeurs manquantes : résultat faux
mean_temperature = np.mean(temperatures)
print("Température moyenne (sans traitement des valeurs manquantes) :", mean_temperature)

# Créer un masque pour les valeurs manquantes (-999)
mask = (temperatures == -999)

# Créer un tableau masqué
masked_temperatures = ma.masked_array(temperatures, mask=mask)

# Moyenne en excluant les valeurs manquantes
mean_temperature = ma.mean(masked_temperatures)
print("Température moyenne (sans les valeurs manquantes) :", mean_temperature)
```

Python

```
Température moyenne (sans traitement des valeurs manquantes) : -316.14444444444445
Température moyenne (sans les valeurs manquantes) : 25.28333333333333
```

Pour masquer directement une valeur précise, tu peux aussi utiliser `ma.masked_values(temperatures, -999)`.

Les tableaux structurés

Les tableaux structurés te permettent de travailler avec des données qui ressemblent à un tableau à colonnes nommées. Chaque élément peut contenir des champs de types différents. Pour en créer un :

- définis le type de données avec `np.dtype`, sous la forme d'une liste de tuples `(nom du champ, type)` ;
- passe ce type au paramètre `dtype` lors de la création du tableau.

```python
# Définir les types de données des champs
dt = np.dtype([('name', 'U20'), ('age', int), ('salary', float)])

# Créer un tableau structuré
employees = np.array([('Alice', 30, 50000.0), ('Bob', 25, 60000.0)], dtype=dt)

# Accéder au champ 'name' du premier employé
print(employees['name'][0])

# Accéder au champ 'age' de tous les employés
print(employees['age'])
```

Python

```
Alice
[30 25]
```

Le type `'U20'` correspond à une chaîne de caractères Unicode de 20 caractères au maximum. Avec `'S20'`, tu obtiendrais des chaînes d'octets (`b'Alice'`), moins pratiques à manipuler.

Pour des données de ce genre, Pandas et son DataFrame est souvent plus confortable. La suite de la série porte sur l'optimisation des performances avec NumPy. Documentation : tableaux masqués dans NumPy.

Pour aller plus loin, consulte aussi notre guide des 45 méthodes NumPy.

Maîtriser NumPy, c'est surtout savoir l'utiliser dans une vraie analyse. Le cours Analyse et visualisation de données combine NumPy, pandas et Matplotlib sur des jeux de données réels.

