INTRODUCTION A DATASCIENCE

khalil lakhdhar

PROTECH-IT khalillakhdhar@gmail.com

Contents

Présentation de NumPy

Qu'est-ce que NumPy ?

NumPy est une bibliothèque Python utilisée pour travailler avec des tableaux.

Il a également des fonctions pour travailler dans le domaine de l'algèbre linéaire, de la transformée de Fourier et des matrices.

NumPy a été créé en 2005 par Travis Oliphant. C'est un projet open source et vous pouvez l'utiliser librement.

NumPy signifie Python numérique.

Pourquoi utiliser NumPy ?

En Python, nous avons des listes qui servent à des tableaux, mais elles sont lentes à traiter.

NumPy vise à fournir un objet tableau jusqu'à 50 fois plus rapide que les listes Python traditionnelles.

L'objet tableau dans NumPy s'appelle `ndarray`, il fournit de nombreuses fonctions de support qui facilitent le travail `ndarray`.

Les tableaux sont très fréquemment utilisés en science des données, où la vitesse et les ressources sont très importantes.

Data Science: est une branche de l'informatique où nous étudions comment stocker, utiliser et analyser des données pour en extraire des informations.

Pourquoi NumPy est-il plus rapide que les listes ?

Les tableaux NumPy sont stockés à un endroit continu dans la mémoire contrairement aux listes, de sorte que les processus peuvent y accéder et les manipuler très efficacement.

Ce comportement est appelé localité de référence en informatique.

C'est la principale raison pour laquelle NumPy est plus rapide que les listes. Il est également optimisé pour fonctionner avec les dernières architectures de processeur.

Dans quelle langue NumPy est-il écrit ?

NumPy est une bibliothèque Python et est écrite partiellement en Python, mais la plupart des parties qui nécessitent un calcul rapide sont écrites en C ou C++.

Premiers pas avec NumPy

Installation de NumPy

```
pip install numpy
```

Si cette commande échoue, utilisez une distribution python sur laquelle NumPy est déjà installé, comme Anaconda, Spyder, etc.

Importer NumPy

Une fois NumPy installé, importez-le dans vos applications en ajoutant le `import`mot clé :

```
import numpy
```

Maintenant, NumPy est importé et prêt à être utilisé.

```
import numpy

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

print(arr)
```

NumPy comme np

NumPy est généralement importé sous l' `np` alias.

alias : en Python, les alias sont un nom alternatif pour faire référence à la même chose.

Créez un alias avec le `as`mot-clé lors de l'importation :

```
import numpy as np
```

Désormais, le package NumPy peut être appelé au `np`lieu de `numpy`.

Exemple

```
import numpy as np

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

print(arr)
```

Vérification de la version NumPy

La chaîne de version est stockée sous `__version__` l'attribut.

Exemple

```
import numpy as np

print(np.__version__)
```

NumPy Création de tableaux

Créer un objet NumPy ndarray

NumPy est utilisé pour travailler avec des tableaux. L'objet tableau dans NumPy est appelé `ndarray`.

Nous pouvons créer un `ndarray`objet NumPy en utilisant la `array()`fonction.

Exemple

```
import numpy as np

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

print(arr)

print(type(arr))
```

type() : cette fonction Python intégrée nous indique le type de l'objet qui lui est transmis. Comme dans le code ci-dessus, il montre
que `arr`c'est `numpy.ndarray`le type.

Pour créer un `ndarray`, nous pouvons passer une liste, un tuple ou tout autre objet de type tableau dans la `array()` méthode, et il sera converti en
un `ndarray`:

Exemple

Utilisez un tuple pour créer un tableau NumPy :

```
import numpy as np

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

print(arr)
```

Dimensions dans les tableaux

Une dimension dans les tableaux correspond à un niveau de profondeur de tableau (tableaux imbriqués).

tableau imbriqué : ce sont des tableaux qui ont des tableaux comme éléments.

Tableaux 0-D

Les tableaux 0-D, ou scalaires, sont les éléments d'un tableau. Chaque valeur d'un tableau est un tableau 0-D.

Exemple

Créer un tableau 0-D avec la valeur 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

Tableaux 1-D

Un tableau qui a des tableaux 0-D comme éléments est appelé tableau unidimensionnel ou 1-D.

Ce sont les tableaux les plus courants et les plus basiques.

Exemple

Créez un tableau 1D contenant les valeurs 1,2,3,4,5 :

```
import numpy as np

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

print(arr)
```

Tableaux 2D

Un tableau qui a des tableaux 1-D comme éléments est appelé un tableau 2-D.

Ceux-ci sont souvent utilisés pour représenter des matrices ou des tenseurs du 2e ordre.

NumPy a un sous-module entier dédié aux opérations matricielles appelé `numpy.mat`

Exemple

Créez un tableau 2D contenant deux tableaux avec les valeurs 1,2,3 et 4,5,6 :

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

Tableaux 3D

Un tableau qui a des tableaux 2D (matrices) comme éléments est appelé tableau 3D.

Ceux-ci sont souvent utilisés pour représenter un tenseur d'ordre 3.

Exemple

Créez un tableau 3D avec deux tableaux 2D, contenant tous deux deux tableaux avec les valeurs 1,2,3 et 4,5,6 :

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

Vérifier le nombre de dimensions ?

NumPy Arrays fournit l' `ndim`attribut qui renvoie un entier qui nous indique le nombre de dimensions du tableau.

Exemple

Vérifiez le nombre de dimensions des tableaux :

```
import numpy as np
```

```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Tableaux dimensionnels supérieurs

Un tableau peut avoir n'importe quel nombre de dimensions.

Lorsque le tableau est créé, vous pouvez définir le nombre de dimensions à l'aide de l' `ndmin`argument.

Exemple

Créez un tableau à 5 dimensions et vérifiez qu'il a 5 dimensions :

```
import numpy as np

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

print(arr)
print('number of dimensions :', arr.ndim)
```

Dans ce tableau, la dimension la plus interne (5ème dim) a 4 éléments, la 4ème dim a 1 élément qui est le vecteur, la 3ème dim a 1 élément qui est la matrice avec le vecteur, la 2ème dim a 1 élément qui est un tableau 3D et 1st dim a 1 élément qui est un tableau 4D.

Indexation de tableau NumPy

Accéder aux éléments du tableau

L'indexation d'un tableau est identique à l'accès à un élément de tableau.

Vous pouvez accéder à un élément de tableau en vous référant à son numéro d'index.

Les index dans les tableaux NumPy commencent par 0, ce qui signifie que le premier élément a l'index 0 et le second a l'index 1, etc.

Exemple

Obtenez le premier élément du tableau suivant :

```
import numpy as np

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

print(arr[0])
```

Exemple

Obtenez le deuxième élément du tableau suivant.

```
import numpy as np

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

print(arr[1])
```

Exemple

Obtenez les troisième et quatrième éléments du tableau suivant et ajoutez-les.

```
import numpy as np

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

print(arr[2] + arr[3])
```

Accéder aux tableaux 2D

Pour accéder aux éléments des tableaux 2D, nous pouvons utiliser des entiers séparés par des virgules représentant la dimension et l'indice de l'élément.

Considérez les tableaux 2D comme un tableau avec des lignes et des colonnes, où la dimension représente la ligne et l'index représente la colonne.

Exemple

Accédez à l'élément de la première ligne, deuxième colonne :

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

Exemple

Accédez à l'élément de la 2e ligne, 5e colonne :

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('5th element on 2nd row: ', arr[1, 4])
```

Accéder aux tableaux 3D

Pour accéder aux éléments des tableaux 3D, nous pouvons utiliser des entiers séparés par des virgules représentant les dimensions et l'indice de l'élément.

Exemple expliqué

`arr[0, 1, 2]`imprime la valeur `6`.

Et c'est pourquoi :

Le premier nombre représente la première dimension, qui contient deux tableaux :
[[1, 2, 3], [4, 5, 6]]
et :
[[7, 8, 9], [10, 11, 12]]
Puisque nous avons sélectionné `0`, il nous reste le premier tableau :
[[1, 2, 3], [4, 5, 6]]

Le deuxième nombre représente la deuxième dimension, qui contient également deux tableaux :
[1, 2, 3]
et :
[4, 5, 6]
Puisque nous avons sélectionné `1`, il nous reste le deuxième tableau :
[4, 5, 6]

Le troisième nombre représente la troisième dimension, qui contient trois valeurs :
4
5
6
Puisque nous avons sélectionné `2`, nous nous retrouvons avec la troisième valeur :
6

Exercice:

Créer un tableau numpy de 10 éléments puis récupérez au clavier un entier X

Et créez et affichez deux tableaux inf (contenant les élément inférieurs à X et sup contenant les éléments supérieurs et afficher le nombre des occurrences de X (sans comparaison directe)

Indexation négative

Utilisez l'indexation négative pour accéder à un tableau à partir de la fin.

Exemple

Imprimez le dernier élément de la 2ème dim :

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

Tableaux de recherche NumPy

Recherche de tableaux

Vous pouvez rechercher une certaine valeur dans un tableau et renvoyer les index qui correspondent.

Pour rechercher un tableau, utilisez la `where()`méthode.

Exemple

Trouvez les index dont la valeur est 4 :

```
import numpy as np

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

```
x = np.where(arr == 4)

print(x)
```

L'exemple ci-dessus renverra un tuple :`(array([3, 5, 6],)`

Ce qui signifie que la valeur 4 est présente aux index 3, 5 et 6.

Exemple

Trouvez les index où les valeurs sont paires :

```
import numpy as np

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

x = np.where(arr%2 == 0)

print(x)
```

Exemple

Trouvez les index où les valeurs sont impaires :

```
import numpy as np

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

x = np.where(arr%2 == 1)

print(x)
```

Recherche triée

Il existe une méthode appelée `searchsorted()`qui effectue une recherche binaire dans le tableau et renvoie l'index où la valeur spécifiée serait insérée pour maintenir l'ordre de recherche.

La `searchsorted()`méthode est supposée être utilisée sur des tableaux triés.

Exemple

Trouvez les index où la valeur 7 doit être insérée :

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print(x)
```

Explication de l'exemple : Le chiffre 7 doit être inséré sur l'index 1 pour conserver l'ordre de tri.

La méthode commence la recherche à partir de la gauche et renvoie le premier index où le nombre 7 n'est plus supérieur à la valeur suivante.

Rechercher du côté droit

Par défaut, l'index le plus à gauche est renvoyé, mais nous pouvons choisir `side='right'`de renvoyer l'index le plus à droite à la place.

Exemple

Trouvez les index où la valeur 7 doit être insérée, en partant de la droite :

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7, side='right')
```

```
print(x)
```

Explication de l'exemple : Le chiffre 7 doit être inséré sur l'index 2 pour conserver l'ordre de tri.

La méthode commence la recherche à partir de la droite et renvoie le premier index où le nombre 7 n'est plus inférieur à la valeur suivante.

Valeurs multiples

Pour rechercher plusieurs valeurs, utilisez un tableau avec les valeurs spécifiées.

Exemple

Trouvez les index où les valeurs 2, 4 et 6 doivent être insérées :

```
import numpy as np

arr = np.array([1, 3, 5, 7])

x = np.searchsorted(arr, [2, 4, 6])

print(x)
```

La valeur de retour est un tableau : `[1 2 3]`contenant les trois index où 2, 4, 6 seraient insérés dans le tableau d'origine pour maintenir l'ordre.

Nombres aléatoires dans NumPy

Un nombre aléatoire ne signifie PAS un nombre différent à chaque fois. Aléatoire signifie quelque chose qui ne peut pas être prédit logiquement.

Pseudo aléatoire et vrai aléatoire.

Les ordinateurs fonctionnent sur des programmes, et les programmes sont un ensemble définitif d'instructions. Cela signifie donc qu'il doit également y avoir un algorithme pour générer un nombre aléatoire.

S'il existe un programme pour générer un nombre aléatoire, il peut être prédit, il n'est donc pas vraiment aléatoire.

Les nombres aléatoires générés par un algorithme de génération sont appelés *pseudo-aléatoires* .

Peut-on faire des nombres vraiment aléatoires ?

Oui. Afin de générer un nombre vraiment aléatoire sur nos ordinateurs, nous devons obtenir les données aléatoires d'une source extérieure. Cette source extérieure est généralement nos frappes au clavier, les mouvements de la souris, les données sur le réseau, etc.

Nous n'avons pas vraiment besoin de nombres aléatoires, à moins qu'ils ne soient liés à la sécurité (par exemple, les clés de cryptage) ou que la base d'application soit le caractère aléatoire (par exemple, les roulettes numériques).

Générer un nombre aléatoire

NumPy propose le `random`module pour travailler avec des nombres aléatoires.

Exemple

Générez un entier aléatoire de 0 à 100 :

```
from numpy import random

x = random.randint(100)

print(x)
```

Générer un flottant aléatoire

La `rand()`méthode du module random renvoie un flottant aléatoire entre 0 et 1.

Exemple

Générez un flottant aléatoire de 0 à 1 :

```
from numpy import random

x = random.rand()

print(x)
```

Générer un tableau aléatoire

Dans NumPy, nous travaillons avec des tableaux et vous pouvez utiliser les deux méthodes des exemples ci-dessus pour créer des tableaux aléatoires.

Entiers

La `randint()`méthode prend un `size` paramètre dans lequel vous pouvez spécifier la forme d'un tableau.

Exemple

Générez un tableau 1-D contenant 5 entiers aléatoires de 0 à 100 :

```
from numpy import random

x=random.randint(100, size=(5))

print(x)
```

Exemple

Générez un tableau 2D avec 3 lignes, chaque ligne contenant 5 entiers aléatoires de 0 à 100 :

```
from numpy import random

x = random.randint(100, size=(3, 5))

print(x)
```

Flotteurs

La `rand()`méthode vous permet également de spécifier la forme du tableau.

Exemple

Générez un tableau 1-D contenant 5 flottants aléatoires :

```
from numpy import random

x = random.rand(5)

print(x)
```

Exemple

Générez un tableau 2D avec 3 lignes, chaque ligne contenant 5 nombres aléatoires :

```
from numpy import random

x = random.rand(3, 5)

print(x)
```

Générer un nombre aléatoire à partir d'un tableau

La `choice()`méthode vous permet de générer une valeur aléatoire basée sur un tableau de valeurs.

La `choice()`méthode prend un tableau comme paramètre et renvoie aléatoirement l'une des valeurs.

Exemple

Renvoie l'une des valeurs d'un tableau :

```
from numpy import random

x = random.choice([3, 5, 7, 9])

print(x)
```

La `choice()`méthode vous permet également de renvoyer un *tableau* de valeurs.

Ajoutez un `size`paramètre pour spécifier la forme du tableau.

Exemple

Générez un tableau 2D composé des valeurs du paramètre tableau (3, 5, 7 et 9) :

```
from numpy import random

x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

Exercice:

Rempliseez un tableau tab avec des entiers aléatoire entre 5 et 100 de 2 dimensions

Et afficher les éléments paires

Pandas :

Présentation

C'est quoi Pandas ?

Pandas est une bibliothèque Python utilisée pour travailler avec des ensembles de données.

Il a des fonctions d'analyse, de nettoyage, d'exploration et de manipulation des données.

Le nom "Pandas" fait référence à la fois à "Panel Data" et à "Python Data Analysis" et a été créé par Wes McKinney en 2008.

Pourquoi utiliser les pandas ?

Pandas nous permet d'analyser les mégadonnées et de tirer des conclusions basées sur des théories statistiques.

Les pandas peuvent nettoyer des ensembles de données désordonnés et les rendre lisibles et pertinents.

Les données pertinentes sont très importantes en science des données.

:}

Data Science: est une branche de l'informatique où nous étudions comment stocker, utiliser et analyser des données pour en extraire des informations.

Que peuvent faire les pandas ?

Pandas vous donne des réponses sur les données. Comme:

- Existe-t-il une corrélation entre deux colonnes ou plus ?
- Qu'est-ce que la valeur moyenne ?
- Valeur max?
- Valeur mini ?

Les pandas sont également capables de supprimer les lignes qui ne sont pas pertinentes ou qui contiennent des valeurs erronées, comme des valeurs vides ou NULL. C'est ce qu'on appelle *nettoyer* les données.

Pandas pour commencer

Installation de Pandas

```
pip install pandas
```

Si cette commande échoue, utilisez une distribution python sur laquelle Pandas est déjà installé, comme Anaconda, Spyder, etc.

Importer des pandas

Une fois Pandas installé, importez-le dans vos applications en ajoutant le `import`mot clé :

```
import pandas
```

Maintenant, Pandas est importé et prêt à être utilisé.

Exemple

```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
```

```
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

Pandas as pd

Les pandas sont généralement importés sous l' `pd` alias.

alias : en Python, les alias sont un nom alternatif pour faire référence à la même chose.

Créez un alias avec le `as`mot-clé lors de l'importation :

```
import pandas as pd
```

Désormais, le package Pandas peut être appelé au `pd`lieu de `pandas`.

Exemple

```
import pandas as pd

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pd.DataFrame(mydataset)

print(myvar)
```

Vérification de la version Pandas

La chaîne de version est stockée sous `__version__` l'attribut.

Exemple

```
import pandas as pd

print(pd.__version__)
```

Pandas Series

Qu'est-ce qu'une série ?

Une série Pandas est comme une colonne dans un tableau.

C'est un tableau unidimensionnel contenant des données de tout type.

Exemple

Créez une série Pandas simple à partir d'une liste :

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Étiquettes

Si rien d'autre n'est spécifié, les valeurs sont étiquetées avec leur numéro d'index. La première valeur a l'indice 0, la deuxième valeur a l'indice 1, etc.

Cette étiquette peut être utilisée pour accéder à une valeur spécifiée.

Exemple

Renvoie la première valeur de la série :

```
print(myvar[0])
```

Créer des étiquettes

Avec l' `index`argument, vous pouvez nommer vos propres étiquettes.

Exemple

Créez vos propres étiquettes :

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

Lorsque vous avez créé des étiquettes, vous pouvez accéder à un élément en vous référant à l'étiquette.

Exemple

Renvoie la valeur de "y":

```
print(myvar["y"])
```

Objets clé/valeur en série

Vous pouvez également utiliser un objet clé/valeur, comme un dictionnaire, lors de la création d'une série.

Exemple

Créez une série Pandas simple à partir d'un dictionnaire :

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories)

print(myvar)
```

Remarque : Les clés du dictionnaire deviennent les libellés.

Pour sélectionner uniquement certains éléments du dictionnaire, utilisez l' `index` argument et spécifiez uniquement les éléments que vous souhaitez inclure dans la série.

Exemple

Créez une série en utilisant uniquement les données de "jour1" et "jour2" :

```
import pandas as pd

calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

DataFrames

Les ensembles de données dans Pandas sont généralement des tables multidimensionnelles, appelées DataFrames.

La série est comme une colonne, un DataFrame est la table entière.

Exemple

Créez un DataFrame à partir de deux séries :

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

myvar = pd.DataFrame(data)

print(myvar)
```

Pandas DataFrames

Qu'est-ce qu'un DataFrame ?

Un Pandas DataFrame est une structure de données à 2 dimensions, comme un tableau à 2 dimensions ou un tableau avec des lignes et des colonnes.

Exemple

Créez un DataFrame Pandas simple :

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
```

```
#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Résultat

```
   durée calorique
0 420 50
1 380 40
2 390 45
```

Localiser la ligne

Comme vous pouvez le voir dans le résultat ci-dessus, le DataFrame est comme un tableau avec des lignes et des colonnes.

Les pandas utilisent l' `loc`attribut pour renvoyer une ou plusieurs lignes spécifiées

Exemple

Renvoyer la ligne 0 :

```
#refer to the row index:
print(df.loc[0])
```

Résultat

```
calories 420
durée 50
Nom : 0, dtype : int64
```

Remarque : cet exemple renvoie une Pandas **Series** .

Exemple

Renvoyer les lignes 0 et 1 :

```
#use a list of indexes:
print(df.loc[[0, 1]])
```

Résultat

```
   durée calorique
0 420 50
1 380 40
```

Remarque : lors de l'utilisation de `[]`, le résultat est un Pandas **DataFrame** .

Index nommés

Avec l' `index`argument, vous pouvez nommer vos propres index.

Exemple

Ajoutez une liste de noms pour donner un nom à chaque ligne :

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

Résultat

```
      durée calorique
jour1 420 50
```

```
  jour2 380 40
  jour3 390 45
```

Localiser les index nommés

Utilisez l'index nommé dans l' `loc`attribut pour renvoyer la ou les lignes spécifiées.

Exemple

Retour "jour2":

```
#refer to the named index:
print(df.loc["day2"])
```

Résultat

```
  calories 380
  durée 40
  Nom : 0, dtype : int64
```

Charger des fichiers dans un DataFrame

Si vos ensembles de données sont stockés dans un fichier, Pandas peut les charger dans un DataFrame.

Exemple

Chargez un fichier séparé par des virgules (fichier CSV) dans un DataFrame :

```
import pandas as pd

df = pd.read_csv('data.csv')
```

```
print(df)
```

Pandas Read CSV

Lire les fichiers CSV

Un moyen simple de stocker de grands ensembles de données consiste à utiliser des fichiers CSV (fichiers séparés par des virgules).

Les fichiers CSV contiennent du texte brut et sont un format bien connu qui peut être lu par tout le monde, y compris les Pandas.

Dans nos exemples, nous utiliserons un fichier CSV appelé 'data.csv'.

Exemple

Chargez le CSV dans un DataFrame :

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

Astuce : utilisez `to_string()`pour imprimer l'intégralité du DataFrame.

Si vous avez un DataFrame volumineux avec de nombreuses lignes, Pandas ne renverra que les 5 premières lignes et les 5 dernières lignes :

Exemple

Imprimez le DataFrame sans la `to_string()` méthode :

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df)
```

max_rows

Le nombre de lignes renvoyées est défini dans les paramètres d'option de Pandas.

Vous pouvez vérifier le nombre maximal de lignes de votre système avec l' `pd.options.display.max_rows`instruction.

Exemple

Vérifiez le nombre maximal de lignes renvoyées :

```
import pandas as pd

print(pd.options.display.max_rows)
```

Dans mon système, le nombre est 60, ce qui signifie que si le DataFrame contient plus de 60 lignes, l' `print(df)`instruction renverra uniquement les en-têtes et les 5 premières et dernières lignes.

Vous pouvez modifier le nombre maximal de lignes avec la même instruction.

Exemple

Augmentez le nombre maximum de lignes pour afficher l'intégralité du DataFrame :

```
import pandas as pd

pd.options.display.max_rows = 9999

df = pd.read_csv('data.csv')

print(df)
```

Pandas Read JSON (RestAPI)

Les grands ensembles de données sont souvent stockés ou extraits au format JSON.

JSON est du texte brut, mais a le format d'un objet, et est bien connu dans le monde de la programmation, y compris Pandas.

Dans nos exemples, nous utiliserons un fichier JSON appelé 'data.json'.

Exemple

Chargez le fichier JSON dans un DataFrame :

```
import pandas as pd

df = pd.read_json('data.json')

print(df.to_string())
```

Astuce : utilisez `to_string()`pour imprimer l'intégralité du DataFrame.

Dictionnaire au format JSON

JSON = Dictionnaire Python

Les objets JSON ont le même format que les dictionnaires Python.

Si votre code JSON n'est pas dans un fichier, mais dans un dictionnaire Python, vous pouvez le charger directement dans un DataFrame :

Exemple

Chargez un dictionnaire Python dans un DataFrame :

```
import pandas as pd
```

```
data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}

df = pd.DataFrame(data)

print(df)
```

Pandas - Analyzing DataFrames

Affichage des données

L'une des méthodes les plus utilisées pour obtenir un aperçu rapide du DataFrame est la `head()`méthode .

La `head()`méthode renvoie les en-têtes et un nombre spécifié de lignes, en commençant par le haut.

Exemple

Obtenez un aperçu rapide en imprimant les 10 premières lignes du DataFrame :

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```

Dans nos exemples, nous utiliserons notre fichier CSV appelé 'data.csv'.

Remarque : si le nombre de lignes n'est pas spécifié, la `head()`méthode renverra les 5 premières lignes.

Exemple

Imprimez les 5 premières lignes du DataFrame :

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head())
```

Il existe également une `tail()`méthode pour visualiser les *dernières* lignes du DataFrame.

La `tail()`méthode renvoie les en-têtes et un nombre spécifié de lignes, en commençant par le bas.

Exemple

Imprimez les 5 dernières lignes du DataFrame :

```
print(df.tail())
```

Informations sur les données

L'objet DataFrames a une méthode appelée `info()`, qui vous donne plus d'informations sur l'ensemble de données.

Exemple

Imprimer des informations sur les données :

```
print(df.info())
```

Résultat

```
<classe 'pandas.core.frame.DataFrame'>
RangeIndex : 169 entrées, 0 à 168
Colonnes de données (4 colonnes au total) :
 # Colonne Non-Null Count Dtype
--- ------ -------------- -----
 0 Durée 169 int64 non nul
 1 Impulsion 169 int64 non nul
 2 Maxpulse 169 int64 non nul
 3 Calories 164 flottant non nul64
dtypes : float64(1), int64(3)
utilisation de la mémoire : 5,4 Ko
Aucun
```

Résultat expliqué

Le résultat nous indique qu'il y a 169 lignes et 4 colonnes :

```
RangeIndex : 169 entrées, 0 à 168
Colonnes de données (4 colonnes au total) :
```

Et le nom de chaque colonne, avec le type de données :

```
 # Colonne Non-Null Count Dtype
--- ------ -------------- -----
 0 Durée 169 int64 non nul
 1 Impulsion 169 int64 non nul
 2 Maxpulse 169 int64 non nul
 3 Calories 164 flottant non nul64
```

Valeurs nulles

La `info()`méthode nous indique également combien de valeurs non nulles sont présentes dans chaque colonne, et dans notre ensemble de données, il semble qu'il y ait 164 des 169 valeurs non nulles dans la colonne "Calories".

Ce qui signifie qu'il y a 5 lignes sans aucune valeur dans la colonne "Calories", pour une raison quelconque.

Pandas - Nettoyage des données

Nettoyage des données

Le nettoyage des données signifie réparer les mauvaises données dans votre ensemble de données.

Les mauvaises données peuvent être :

- Cellules vides
- Données au mauvais format
- Données erronées
- Doublons

Dans ce didacticiel, vous apprendrez à les gérer tous.

Notre ensemble de données

Dans les prochains chapitres, nous utiliserons cet ensemble de données :

```
    Durée Date Pouls Maxpulse Calories
0 60 '2020/12/01' 110 130 409.1
1 60 '2020/12/02' 117 145 479.0
2 60 '2020/12/03' 103 135 340.0
3 45 '2020/12/04' 109 175 282.4
4 45 '2020/12/05' 117 148 406.0
5 60 '2020/12/06' 102 127 300.0
6 60 '2020/12/07' 110 136 374.0
7 450 '2020/12/08' 104 134 253,3
8 30 '2020/12/09' 109 133 195.1
9 60 '2020/12/10' 98 124 269.0
10 60 '2020/12/11' 103 147 329.3
11 60 '2020/12/12' 100 120 250.7
12 60 '2020/12/12' 100 120 250.7
13 60 '2020/12/13' 106 128 345,3
14 60 '2020/12/14' 104 132 379,3
15 60 '2020/12/15' 98 123 275.0
16 60 '2020/12/16' 98 120 215.2
17 60 '2020/12/17' 100 120 300.0
18 45 '2020/12/18' 90 112 NaN
19 60 '2020/12/19' 103 123 323.0
20 45 '2020/12/20' 97 125 243.0
21 60 '2020/12/21' 108 131 364.2
22 45 NaN 100 119 282,0
23 60 '2020/12/23' 130 101 300.0
24 45 '2020/12/24' 105 132 246.0
25 60 '2020/12/25' 102 126 334.5
26 60 2020/12/26 100 120 250,0
27 60 '2020/12/27' 92 118 241.0
28 60 '2020/12/28' 103 132 NaN
29 60 '2020/12/29' 100 132 280.0
30 60 '2020/12/30' 102 129 380.3
31 60 '2020/12/31' 92 115 243.0
```

L'ensemble de données contient des cellules vides ("Date" dans la ligne 22 et "Calories" dans les lignes 18 et 28).

L'ensemble de données contient un format incorrect ("Date" dans la ligne 26).

L'ensemble de données contient des données erronées ("Duration" dans la ligne 7).

L'ensemble de données contient des doublons (lignes 11 et 12).

Pandas - Nettoyage des cellules vides

Cellules vides

Les cellules vides peuvent potentiellement vous donner un résultat erroné lorsque vous analysez des données.

Supprimer les lignes

Une façon de traiter les cellules vides consiste à supprimer les lignes contenant des cellules vides.

C'est généralement correct, car les ensembles de données peuvent être très volumineux et la suppression de quelques lignes n'aura pas un grand impact sur le résultat.

Exemple

Renvoie un nouveau bloc de données sans cellules vides :

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

Remarque : Par défaut, la `dropna()`méthode renvoie un *nouveau* DataFrame et ne modifiera pas l'original.

Si vous souhaitez modifier le DataFrame d'origine, utilisez l' `inplace = True`argument :

Exemple

Supprimez toutes les lignes contenant des valeurs NULL :

```
import pandas as pd

df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df.to_string())
```

Remarque : Désormais, le `dropna(inplace = True)`ne renverra PAS un nouveau DataFrame, mais il supprimera toutes les lignes contenant des valeurs NULL du DataFrame d'origine.

Remplacer les valeurs vides

Une autre façon de gérer les cellules vides consiste à insérer une *nouvelle* valeur à la place.

De cette façon, vous n'avez pas à supprimer des lignes entières simplement à cause de certaines cellules vides.

La `fillna()`méthode nous permet de remplacer les cellules vides par une valeur :

Exemple

Remplacez les valeurs NULL par le nombre 130 :

```
import pandas as pd

df = pd.read_csv('data.csv')

df.fillna(130, inplace = True)
```

Remplacer uniquement pour les colonnes spécifiées

L'exemple ci-dessus remplace toutes les cellules vides dans l'ensemble du bloc de données.

Pour remplacer uniquement les valeurs vides d'une colonne, spécifiez le *nom de la colonne* pour le DataFrame :

Exemple

Remplacez les valeurs NULL dans les colonnes "Calories" par le nombre 130 :

```
import pandas as pd

df = pd.read_csv('data.csv')

df["Calories"].fillna(130, inplace = True)
```

Pandas - Suppression des doublons

Découverte des doublons

Les lignes en double sont des lignes qui ont été enregistrées plus d'une fois.

```
    Durée Date Pouls Maxpulse Calories
0 60 '2020/12/01' 110 130 409.1
1 60 '2020/12/02' 117 145 479.0
2 60 '2020/12/03' 103 135 340.0
3 45 '2020/12/04' 109 175 282.4
4 45 '2020/12/05' 117 148 406.0
5 60 '2020/12/06' 102 127 300.0
6 60 '2020/12/07' 110 136 374.0
7 450 '2020/12/08' 104 134 253,3
8 30 '2020/12/09' 109 133 195.1
```

```
 9 60 '2020/12/10' 98 124 269.0
 10 60 '2020/12/11' 103 147 329.3
 11 60 '2020/12/12' 100 120 250.7
 12 60 '2020/12/12' 100 120 250.7
 13 60 '2020/12/13' 106 128 345,3
 14 60 '2020/12/14' 104 132 379,3
 15 60 '2020/12/15' 98 123 275.0
 16 60 '2020/12/16' 98 120 215.2
 17 60 '2020/12/17' 100 120 300.0
 18 45 '2020/12/18' 90 112 NaN
 19 60 '2020/12/19' 103 123 323.0
 20 45 '2020/12/20' 97 125 243.0
 21 60 '2020/12/21' 108 131 364.2
 22 45 NaN 100 119 282,0
 23 60 '2020/12/23' 130 101 300.0
 24 45 '2020/12/24' 105 132 246.0
 25 60 '2020/12/25' 102 126 334.5
 26 60 20201226 100 120 250,0
 27 60 '2020/12/27' 92 118 241.0
 28 60 '2020/12/28' 103 132 NaN
 29 60 '2020/12/29' 100 132 280.0
 30 60 '2020/12/30' 102 129 380.3
 31 60 '2020/12/31' 92 115 243.0
```

En examinant notre ensemble de données de test, nous pouvons supposer que les lignes 11 et 12 sont des doublons.

Pour découvrir les doublons, nous pouvons utiliser la `duplicated()`méthode.

La `duplicated()`méthode renvoie une valeur booléenne pour chaque ligne :

Exemple

Renvoie `True`pour chaque ligne qui est un doublon, sinon`False` :

```
print(df.duplicated())
```

Suppression des doublons

Pour supprimer les doublons, utilisez la `drop_duplicates()`méthode.

Exemple

Supprimez tous les doublons :

```
df.drop_duplicates(inplace = True)
```

N'oubliez pas : le `(inplace = True)`s'assurera que la méthode ne renvoie PAS un *nouveau* DataFrame, mais il supprimera tous les doublons du DataFrame *d'origine* .