Data Frame en Python

La fonction `pandas.DataFrame()` : La fonction `pandas.DataFrame()` en Python sert à créer un objet **DataFrame**, une structure de données bidimensionnelle (tableau) très utilisée pour la manipulation et l'analyse de données. Elle fait partie de la bibliothèque **Pandas**, l'une des bibliothèques les plus populaires en Python pour le traitement des données.

Qu'est-ce qu'un DataFrame ? Un **DataFrame** est une structure de données en forme de tableau, similaire à une table dans une base de données ou une feuille de calcul Excel. Il contient des lignes et des colonnes, et chaque colonne peut avoir un type de données différent (par exemple, des entiers, des chaînes de caractères, des flottants, etc.).

Utilité de `pandas.DataFrame()` : Un DataFrame() permet de :

- Stocker des données sous forme de tableau avec des étiquettes pour les lignes et les colonnes.
- Manipuler facilement les données : sélection, ajout, suppression de colonnes ou de lignes, filtrage, etc.
- Effectuer des calculs statistiques, des analyses de données, et des visualisations.

Exemple d'utilisation de `pandas.DataFrame()` Voici un exemple de création d'un DataFrame à partir de données financières :

```
import pandas as pd

# Créer un DataFrame avec des données financières
data = {
    'Entreprise': ['Apple', 'Tesla', 'Amazon', 'Microsoft', 'Google'],
    'Prix': [150, 750, 3500, 300, 2800],
    'Volume': [1000000, 900000, 1300000, 1200000, 1500000],
    'Rendement': [0.02, 0.015, 0.01, 0.025, 0.03]
}

# Créer un DataFrame à partir du dictionnaire
df = pd.DataFrame(data)
```

```

# Afficher le DataFrame
print(df)
```

Résultat : Le résultat de l'exécution du code ci-dessus donnera un tableau avec les informations financières des entreprises :

	Entreprise	Prix	Volume	Rendement
0	Apple	150	1000000	0.020
1	Tesla	750	900000	0.015
2	Amazon	3500	1300000	0.010
3	Microsoft	300	1200000	0.025
4	Google	2800	1500000	0.030

Ajout d'une colonne dans un DataFrame : Dans Pandas, il est très facile d'ajouter une nouvelle colonne à un **DataFrame**. Voici comment vous pouvez ajouter une colonne au DataFrame en effectuant une opération entre les colonnes existantes.

```
# Ajouter une colonne 'Capitalisation' au DataFrame
df['Capitalisation'] = df['Prix'] * df['Volume']

# Afficher le DataFrame mis à jour
print(df)
```

Filtrer les entreprises avec un prix supérieur à 500: Dans Pandas, il est possible de filtrer un DataFrame en fonction d'une condition appliquée sur une ou plusieurs colonnes. Voici comment filtrer les entreprises dont le prix est supérieur à 500.

```
# Filtrer les entreprises avec un prix supérieur à 500
df_filtre = df[df['Prix'] > 500]

# Afficher le DataFrame filtré
print(df_filtre)
```

Résultat : Le DataFrame filtré sera affiché ainsi :

```
  Entreprise  Prix   Volume  Rendement
1      Tesla   750   900000      0.015
2     Amazon  3500  1300000      0.010
4     Google  2800  1500000      0.030
```

Explication

- `df['Prix']` fait référence à la colonne `Prix` dans le `DataFrame` `df`.
- L'expression `df['Prix'] > 500` crée un masque booléen qui renvoie `True` pour les lignes où la condition est remplie (prix supérieur à 500), et `False` sinon.
- `df[df['Prix'] > 500]` applique ce masque booléen au `DataFrame`, sélectionnant uniquement les lignes où le prix est supérieur à 500.

Filtrer les entreprises avec un rendement inférieur à 0.020 :

```
# Filtrer les entreprises avec un rendement inférieur à 0.020
df_filtre = df[df['Rendement'] < 0.020]

# Afficher le DataFrame filtré
print(df_filtre)

```

Résultat : Le DataFrame filtré sera affiché ainsi :

```
  Entreprise  Prix   Volume  Rendement
1      Tesla   750   900000      0.015
2     Amazon  3500  1300000      0.010
```

Accéder à la colonne 'Prix' :

```
# Accéder à la colonne 'Prix'
colonne_prix = df['Prix']
print(colonne_prix)
```

Accéder à une ligne dans un DataFrame: Pour accéder à une ligne spécifique dans un `DataFrame` Pandas, on peut utiliser les méthodes suivantes :

1. Utilisation de `iloc` (par index positionnel)

```
# Accéder à la première ligne (index 0)
ligne_premiere = df.iloc[0]
print(ligne_premiere)
```

Explication :

- `iloc` permet d'accéder à une ligne en utilisant sa position (index numérique).
- Cela renvoie une `Série` contenant toutes les colonnes et leurs valeurs pour cette ligne.

Résultat pour `df.iloc[0]` :

```
Entreprise       Apple
Prix               150
Volume         1000000
Name: 0, dtype: object
```

2. Utilisation de `loc` (par étiquette d'index):

```
# Accéder à une ligne par son étiquette d'index (si définie)
ligne_par_etiquette = df.loc[0]
print(ligne_par_etiquette)
```

Explication :

- `loc` permet d'accéder à une ligne en utilisant son étiquette d'index.
- Par défaut, les étiquettes d'index correspondent aux positions numériques.

Qu'est-ce qu'une étiquette d'index dans Pandas ?

L'étiquette d'index (ou simplement l'index) est un identifiant unique associé à chaque ligne d'un `DataFrame` ou d'une `Série` dans Pandas. Elle permet de référencer les lignes de manière explicite et efficace.

1. Par défaut : Lorsque vous créez un `DataFrame` sans spécifier d'index, Pandas attribue un index numérique par défaut, commençant à `0`.

```
import pandas as pd

data = {
    'Entreprise': ['Apple', 'Tesla', 'Amazon'],
    'Prix': [150, 750, 3500]
}
df = pd.DataFrame(data)

print(df)
```

Résultat :

```
   Entreprise  Prix
0       Apple   150
1       Tesla   750
2      Amazon  3500
```

Ici, les étiquettes d'index sont `0, 1, 2`.

2. Personnaliser l'étiquette d'index : Vous pouvez définir des étiquettes d'index personnalisées, comme des noms ou des identifiants uniques.

```
df = pd.DataFrame(data, index=['AAPL', 'TSLA', 'AMZN'])
print(df)
```

Résultat :

```
      Entreprise  Prix
AAPL       Apple   150
TSLA       Tesla   750
AMZN      Amazon  3500
```

Les étiquettes d'index sont maintenant `AAPL, TSLA, AMZN`.

Accéder à plusieurs lignes : Pour accéder à plusieurs lignes, on peut spécifier une plage d'index avec `iloc` ou `loc` :

```
# Accéder aux lignes d'index 1 à 3 (exclus 3)
lignes = df.iloc[1:3]
print(lignes)
```

Notes

- `iloc` est basé sur les positions (index numériques).
- `loc` est basé sur les étiquettes définies dans l'index.
- Si l'index n'est pas numérique, vous devrez utiliser `loc`.

La fonction `head()`

La méthode `head()` est utilisée pour afficher les premières lignes d'un `DataFrame` ou d'une `Série`.

Syntaxe :

```
DataFrame.head(n=5)
```

- **n** : (optionnel) Le nombre de lignes à afficher.
- **Par défaut** : Affiche les 5 premières lignes si aucun argument n'est spécifié.
- **Personnalisation** : Vous pouvez passer un entier pour afficher un autre nombre de lignes, par exemple `head(10)`.
- **Si moins de lignes disponibles** : Si le `DataFrame` contient moins de `n` lignes, toutes les lignes disponibles seront affichées.

Exemple :

```
import pandas as pd

data = {
    'Entreprise': ['Apple', 'Tesla', 'Amazon', 'Microsoft', 'Google'],
    'Prix': [150, 750, 3500, 300, 2800]
}
df = pd.DataFrame(data)

# Par défaut (5 premières lignes)
print(df.head())

# Personnalisation (afficher les 3 premières lignes)
print(df.head(3))
```

La fonction `info()` :

La méthode `info()` est utilisée pour obtenir un résumé concis de l'état d'un `DataFrame` en affichant des informations sur les colonnes, le nombre de valeurs non nulles, les types de données et la mémoire utilisée.

Syntaxe :

```
DataFrame.info(verbose=None, buf=None, max_cols=None, memory_usage=None, null_counts=None
```

Points clés :

- Affiche des informations sur les colonnes du `DataFrame` : nom, type de données et nombre de valeurs non nulles.
- Permet d'obtenir une vue rapide sur les types de données (par exemple, `int64`, `float64`, `object`, etc.).
- Affiche le nombre de valeurs manquantes dans chaque colonne.
- Pratique pour comprendre la structure du `DataFrame` sans afficher toutes les données.

Exemple :

```
import pandas as pd

data = {
    'Entreprise': ['Apple', 'Tesla', 'Amazon', 'Microsoft', 'Google'],
    'Prix': [150, 750, 3500, 300, 2800],
    'Volume': [1000000, 900000, 1300000, 1200000, 1500000]
}
df = pd.DataFrame(data)

# Afficher les informations du DataFrame
df.info()
```

Exemple de sortie :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Entreprise  5 non-null      object
 1   Prix        5 non-null      int64
 2   Volume      5 non-null      int64
dtypes: int64(2), object(1)
memory usage: 203.0+ bytes
```

Remarque 6. Les **valeurs manquantes** dans un `DataFrame` ou une `Série` représentent des données absentes ou non définies dans un ensemble de données. Ces valeurs peuvent être causées par différents facteurs, tels que des erreurs de collecte de données, des données omises, ou des informations qui n'étaient pas disponibles au moment de l'enregistrement des données.
Dans Pandas, les valeurs manquantes sont souvent représentées par les valeurs suivantes :

- `NaN` (Not a Number) pour les données numériques.
- `None` pour les objets (par exemple, des chaînes de caractères).

Identifier les valeurs manquantes

- `isna()` ou `isnull()` : Ces méthodes retournent un `DataFrame` de la même taille avec des valeurs booléennes (`True` si la valeur est manquante, `False` sinon).
- `notna()` ou `notnull()` : L'inverse de `isna()` ou `isnull()`. Elles retournent `True` pour les valeurs non manquantes.

La visualisation graphique

La visualisation graphique en Python est un processus qui permet de représenter des données sous forme de graphiques pour mieux les comprendre et les analyser. Voici les étapes de base pour réaliser une visualisation graphique avec la bibliothèque populaire `Matplotlib`.

Étapes de la visualisation graphique :

1. Importer les bibliothèques nécessaires :

```
import matplotlib.pyplot as plt  # Pour des graphiques basiques
import seaborn as sns            # Pour des graphiques statistiques
import pandas as pd              # Pour la manipulation des données
```

2. Préparer les données :

```
data = {
    'Date': pd.date_range(start='2024-01-01', periods=5, freq='D'),
    'Prix': [100, 102, 104, 107, 110]
}
df = pd.DataFrame(data)
```

3. Choisir le type de graphique :
   - **Graphique linéaire (line plot)** : Pour observer une évolution dans le temps.
   - **Histogramme (histogram)** : Pour visualiser la distribution d'une variable.
   - **Diagramme à barres (bar chart)** : Pour comparer différentes catégories.
   - **Nuage de points (scatter plot)** : Pour analyser la relation entre deux variables continues.

- **Diagramme en boîte (box plot)** : Pour visualiser la distribution et les valeurs extrêmes. Exemple : Graphique linéaire pour l'évolution du prix.

```
plt.plot(df['Date'], df['Prix'])
```

- Personnaliser le graphique : Une fois le type de graphique choisi, vous pouvez personnaliser plusieurs aspects comme le titre, les labels des axes, la couleur des lignes et ajouter des légendes pour améliorer la lisibilité et l'esthétique du graphique.

```
plt.plot(df['Date'], df['Prix'], marker='o', linestyle='-', color='b')
plt.title("Évolution du prix de l'action")
plt.xlabel("Date")
plt.ylabel("Prix ($)")
```

- Afficher le graphique : La dernière étape consiste à afficher le graphique créé en utilisant la fonction `show()` de `Matplotlib`.

```
plt.show()
```

Exemple complet de visualisation

Voici un exemple complet de visualisation de l'évolution du prix d'une action sur 5 jours.

```
import pandas as pd
import matplotlib.pyplot as plt

# Étape 2 : Préparer les données
data = {
    'Date': pd.date_range(start='2024-01-01', periods=5, freq='D'),
    'Prix': [100, 102, 104, 107, 110]
}
df = pd.DataFrame(data)

# Étape 3 : Choisir le type de graphique
plt.plot(df['Date'], df['Prix'], marker='o', linestyle='-', color='b')

```

```
14 # Étape 4 : Personnaliser le graphique
15 plt.title("Évolution du prix de l'action")
16 plt.xlabel("Date")
17 plt.ylabel("Prix ($)")
18
19 # Étape 5 : Afficher le graphique
20 plt.show()
```

Résumé des Types de Graphiques

Type de Graphique	Appel en Python
Graphique Linéaire	`plt.plot(x, y)`
Histogramme	`plt.hist(data)`
Diagramme à Barres	`plt.bar(categories, values)`
Diagramme en Boîte	`plt.boxplot(data)`
Nuage de Points	`plt.scatter(x, y)`
Diagramme Circulaire	`plt.pie(sizes, labels)`