

Chapitre II

Transformation des données

MI - IA / GL

Ilyas Bambrik

Table des matières



Introduction	3
I - Fonctions sommaire	4
II - Transformation de colonne	6
III - Fonctions d'aggregation	9
IV - Trie	12
V - Renommer les colonnes	15
VI - Combinaison de données	17

Introduction



Dans la section précédente, nous avons appris à sélectionner les données pertinentes dans un DataFrame ou une série. Extraire les bonnes données de notre source d'information est essentiel pour accomplir le travail, comme nous l'avons démontré dans les exercices.

Cependant, les données ne sortent pas toujours présentées dans le format souhaité dès le départ. Parfois, nous devons faire plus de travail nous-mêmes pour transformer celles-ci en fonction de la tâche à accomplir. Ce chapitre couvrira différentes transformations que nous pouvons appliquer à nos données.



Fonctions sommaire



Pandas fournit de nombreuses fonctions prédéfinies simples qui transforment les données de manière utile. Par exemple, considérons la méthode `describe()` :

```
1 rankings["International Outlook Score"].describe()
```

```
1 count      1799.000000
2 mean        46.880378
3 std         22.582401
4 min         14.100000
5 25%         27.900000
6 50%         42.100000
7 75%         62.100000
8 max         99.700000
9 Name: International Outlook Score, dtype: float64
```

Cette méthode génère un résumé de haut niveau de la colonne donnée *"International Outlook Score"*. En outre, cette méthode est sensible au type de la colonne, ce qui signifie que sa sortie change en fonction du type de données de l'entrée. Le résultat ci-dessus n'a de sens que pour les données numériques ; pour les données sous forme de chaîne de caractères, voici ce que nous obtenons :

```
1 rankings["Location"].describe()
```

```
1 count          2047
2 unique          116
3 top      United States
4 freq          173
5 Name: Location, dtype: object
```

Si vous souhaitez obtenir des statistiques récapitulatives simples sur une colonne (moyenne, écart type, etc) d'un DataFrame ou Series, il existe généralement une fonction pandas utile qui permet d'y parvenir.

Par exemple, pour voir la moyenne de la colonne *International Outlook Score*, nous pouvons utiliser la fonction `mean()` :

```
1 rankings["International Outlook Score"].mean()
```

```
1 46.88037798777098
```

Pour voir la liste de valeurs uniques d'une colonne, nous pouvons utiliser la fonction `unique()` :

```
1 rankings.Location.unique()
```

```
1 array(['United Kingdom', 'United States', 'Switzerland', nan, 'Canada',
2       'Australia', 'Singapore', 'Japan', 'France', 'Sweden', 'China',
3       'South Korea', 'Netherlands', 'Germany', 'Belgium', 'Finland',
4       'Denmark', 'Austria', 'Norway', 'New Zealand', 'Spain', 'Italy',
5       'Saudi Arabia', 'Luxembourg', 'Qatar', 'Brazil', 'Israel',
```

```

6      'Ireland', 'Taiwan', 'India', 'United Arab Emirates',
7      'Brunei Darussalam', 'Iceland', 'Lebanon', 'Philippines',
8      'Portugal', 'Iran', 'Malaysia', 'Poland', 'Egypt', 'Turkey',
9      'Greece', 'Vietnam', 'Algeria', 'Nigeria', 'Tanzania', 'Chile',
10     'Pakistan', 'Ukraine', 'Romania', 'Czech Republic', 'South Africa',
11     'Northern Cyprus', 'Hong Kong', 'Ethiopia', 'Jordan', 'Serbia',
12     'Sri Lanka', 'Jamaica', 'Zambia', 'Iraq', 'Costa Rica', 'Cyprus',
13     'Bangladesh', 'Mozambique', 'Colombia', 'Kenya', 'Namibia', 'Peru',
14     'Latvia', 'Oman', 'Thailand', 'Lithuania', 'Slovenia', 'Uganda',
15     'Malta', 'Nepal', 'Kazakhstan', 'Mexico', 'Botswana', 'Slovakia',
16     'Ghana', 'Morocco', 'Georgia', 'Tunisia', 'Mauritius', 'Hungary',
17     'Puerto Rico', 'Ecuador', 'Fiji', 'Croatia', 'Estonia', 'Zimbabwe',
18     'Indonesia', 'Argentina', 'Bulgaria', 'Venezuela', 'Azerbaijan',
19     'Cuba', 'Montenegro', 'Uzbekistan', 'Palestine', 'Kuwait',
20     'Somalia', 'Libya', 'Moldova', 'Kyrgyzstan', 'Malawi', 'Paraguay',
21     'Mongolia', 'Armenia', 'Sudan', 'Turkmenistan', 'Uruguay',
22     'Albania', 'Cambodia', 'Kosovo'], dtype=object)

```

Pour voir une liste de valeurs uniques et leurs fréquences d'apparition dans l'ensemble de données, nous pouvons utiliser la méthode `value_counts()` :

```

1 rankings.Location.value_counts()

```

1	United States	173
2	Japan	150
3	United Kingdom	149
4	India	91
5	China	82
6	...	
7	Puerto Rico	1
8	Mozambique	1
9	Mauritius	1
10	Namibia	1
11	Kosovo	1
12	Name: Location, Length: 116, dtype: int64	

Remarque : Enregistrement d'un DataFrame

Pour enregistrer un DataFrame sous forme d'un fichier csv, la méthode `to_csv` de l'objet `DataFrame` est utilisée avec le nom de fichier comme paramètre :

```

1 df=pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'],
2                  'Sue': ['Pretty good.', 'Bland.']}),
3                  index=['Product A', 'Product B'])
4 df.to_csv('fichier_df.csv')

```

Transformation de colonne

II

Une fonction *map* est un terme emprunté des mathématiques pour désigner une fonction qui prend un ensemble de valeurs et les « mapper » sur un autre ensemble de valeurs. En science des données, nous avons souvent besoin de créer de nouvelles représentations à partir de données existantes, ou de transformer les données du format dans lequel elles se trouvent actuellement vers le format dans lequel nous souhaitons qu'elles soient plus tard. Les fonctions *map* proposés par les classes *DataFrame* et *Series* qui assurent ce travail.

Il existe deux méthodes de cette catégorie que vous utiliserez souvent.

Par exemple, supposons que nous voulions centrer les valeurs de la colonne *International Outlook Score*. Nous pouvons procéder comme suit :

```
1 rankings_mean=rankings["International Outlook Score"].mean()
2 rankings["International Outlook Score"].map(lambda value:value-rankings_mean)
```

```
1 0      49.319622
2 1      33.619622
3 2      48.919622
4 3      32.919622
5 4      42.419622
6      ...
7 2336   29.919622
8 2337   41.819622
9 2338   25.119622
10 2339    0.719622
11 2340   25.519622
12 Name: International Outlook Score, Length: 2341, dtype: float64
```

Remarque

La fonction que vous passer en paramètre à *map* doit prendre une seule valeur de la série (une valeur de la colonne *International Outlook Score*, dans l'exemple précédent) et renvoie une version transformée de cette valeur. *map* renvoie une nouvelle série où toutes les valeurs ont été transformées par votre fonction sans changer la colonne originale.

La fonction *apply* est la méthode équivalente si l'on souhaite transformer un *DataFrame* entier en appelant une méthode personnalisée sur chaque ligne. Cette fonction agit comme la fonction *map()*. Elle prend une fonction comme paramètre et applique cette fonction à chaque ligne du *DataFrame*. Il est possible de spécifier un axe sur lequel vous souhaitez que votre fonction agisse (0 pour les colonnes et 1 pour les lignes). Tout comme la fonction *map()*, la méthode *apply()* peut également être utilisée avec des fonctions anonymes ou des fonctions *lambda*.

Par exemple, prenons le DataFrame suivant :

	A	B	C	D
year				
2012	697	577	90	0
2011	707	614	91	1
2010	802	649	96	1
2009	803	754	84	0
2008	774	671	97	1

En exécutant l'instruction suivante, le résultat retourné par la méthode `apply` est un DataFrame avec les colonnes A et B centrées :

```

1 moyenne_colA=df.A.mean()
2 moyenne_colB=df.B.mean()
3
4 def centrer(ligne):
5     ligne.A=ligne.A-moyenne_colA
6     ligne.B=ligne.B-moyenne_colB
7     return ligne
8
9 df.apply(f,axis=1)

```

	A	B	C	D
year				
2012	-59.6	-76.0	90.0	0.0
2011	-49.6	-39.0	91.0	1.0
2010	45.4	-4.0	96.0	1.0
2009	46.4	101.0	84.0	0.0
2008	17.4	18.0	97.0	1.0

L'argument `axis=1` de la méthode `apply` spécifie que la fonction `centrer` s'applique sur chaque ligne. Si `axis=0`, la fonction en paramètre est appelé sur chaque colonne.

```

1 def moyenne(colonne):

```

```
2     return sum(colonne)/len(colonne)
3 df.apply(moyenne,axis=0)
```

```
1 A      256.6
2 B      653.0
3 C       91.6
4 D        0.6
5 dtype: float64
```

Remarque

- La méthode *apply* ne modifie pas le DataFrame original comme la fonction *map*.
- *apply* retourne la liste des valeurs retournées par la fonction paramètre (dans l'exemple précédant la fonction centrer retourne un objet *ligne* modifié)

Pandas fournit de nombreuses opérations de sommaire utiles en tant que fonctionnalités intégrées. Par exemple, voici un moyen plus rapide de centrer notre colonne A:

```
1 df.A-df.A.mean()
```

```
1 year
2 2012   -59.6
3 2011   -49.6
4 2010    45.4
5 2009    46.4
6 2008    17.4
7 Name: A, dtype: float64
```

Dans ce code, nous effectuons une opération entre un grand nombre de valeurs du côté gauche (tout dans la série) et une seule valeur du côté droit (la valeur moyenne). Pandas examine cette expression et comprend que nous devons vouloir soustraire cette valeur moyenne de chaque valeur de l'ensemble de données.

Pandas comprend également quoi faire si nous effectuons ces opérations entre des séries. Par exemple, un moyen simple de combiner les informations sur les colonne A et B sous forme de chaîne de caractère. L'instruction convertit les colonne A et B en chaîne de caractère (avec la méthode *astype* de *Series*) et puis chaque valeur de la colonne A est concaténée avec "-" et la valeur de la colonne B.

```
1 df.A.astype(str)+" - "+df.B.astype(str)
```

```
1 year
2 2012   197 - 577
3 2011   207 - 614
4 2010   302 - 649
5 2009   303 - 754
6 2008   274 - 671
7 dtype: object
```

Remarque : *astype*

La méthode *astype* d'une série (comme dans le code précédant) convertit le type de la colonne selon le type passé en paramètre.

Fonctions d'aggregation



Comme dans SQL, un DataFrame offre la fonction *groupby* permettant de regrouper les lignes selon les valeurs d'une ou plusieurs colonnes. Par exemple, une fonction que nous avons déjà utilisé est la fonction *value_counts()*. Nous pouvons reproduire ce que fait *value_counts()* :

```
1 # equivlante à rankings.Location.value_counts()
2 rankings.groupby('location').location.count()
```

```
1 location
2 Algeria      86
3 Angola       1
4 Argentina    7
5 Armenia      4
6 Australia    37
7              ..
8 Venezuela    3
9 Vietnam      7
10 Yemen       4
11 Zambia      2
12 Zimbabwe    2
13 Name: location, Length: 127, dtype: int64
```

groupby() a créé des groupes d'entrées correspondants aux valeurs de la colonne *Location*. Ensuite, pour chacun de ces groupes, nous avons récupéré la colonne *Location* et compté combien de fois elle est apparue. *value_counts()* n'est qu'un raccourci de cette opération *groupby()*.

Nous pouvons utiliser n'importe laquelle des fonctions de synthèse avec ces données (comme min,max, mean). Par exemple, pour obtenir le meilleur score d'enseignement (Teaching Score) pour chaque payé, nous pouvons procéder comme suit :

```
1 rankings.groupby('Location')['Teaching Score'].max()
```

```
1 Location
2 Albania      NaN
3 Algeria      30.4
4 Argentina    18.3
5 Armenia      NaN
6 Australia    67.1
7              ...
8 Uzbekistan   NaN
9 Venezuela    15.9
10 Vietnam     18.7
11 Zambia      15.7
12 Zimbabwe    14.8
13 Name: Teaching Score, Length: 116, dtype: float64
```

Vous pouvez considérer chaque groupe que nous générons comme étant une tranche de notre *DataFrame* contenant uniquement des données avec des valeurs qui correspondent. Ce *DataFrame* nous est accessible directement à l'aide de la méthode `apply()`, et nous pouvons ensuite manipuler les données. Par exemple, voici une façon de sélectionner la meilleure université dans chaque pays :


```
1 rankings.groupby('location').apply(lambda x:x.iloc[0])
```

University Rank	Name of University	Location	No of student	student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score	Citations Score	Industry Income Score	International Outlook Score	
Location													
Albania	Reporter	Polytechnic University of Tirana	Albania	8,591	16.2	2%	NaN	NaN	NaN	NaN	NaN	NaN	
Algeria	401-500	Ferhat Abbas Sétif University 1	Algeria	34,637	22.7	1%	63 : 37	42.1-44.9	18.2	19.8	94.7	36.9	40.0
Argentina	1201-1500	National University of San Martin	Argentina	16,276	16.1	5%	60 : 40	18.4-24.3	18.2	10.4	28.6	37.6	40.8
Armenia	Reporter	National Polytechnic University of Armenia	Armenia	5,742	8.7	2%	20 : 80	NaN	NaN	NaN	NaN	NaN	NaN
Australia	34	University of Melbourne	Australia	49,588	23.8	47%	58 : 42	77.6	67.1	75.9	85.8	78.1	93.6
...
Uzbekistan	Reporter	Tashkent Metropolitan University	Uzbekistan	317	22.6	5%	51 : 49	NaN	NaN	NaN	NaN	NaN	NaN
Venezuela	1501+	University of the Andes, Venezuela	Venezuela	24,372	11.7	2%	61 : 39	10.4-18.3	15.9	7.8	8.9	36.9	48.1
Vietnam	401-500	Duy Tan University	Vietnam	21,670	21.9	0%	62 : 38	42.1-44.9	14.4	12.6	100.0	37.7	48.6
Zambia	501-600	University of Zambia	Zambia	16,903	21.2	1%	35 : 65	39.3-42.0	15.7	9.2	96.8	38.4	49.9
Zimbabwe	1001-1200	University of Zimbabwe	Zimbabwe	20,598	23.9	0%	51 : 49	24.4-29.7	14.8	23.3	35.6	43.1	50.0

Une autre méthode importante de `groupby()` est `agg()`, qui vous permet d'exécuter simultanément plusieurs fonctions différentes sur une colonne du *DataFrame* (comme `apply` et `map` mais celle-ci applique une liste de méthodes simultanément). Par exemple, nous pouvons générer un simple résumé statistique de la colonne *Teaching Score*. Premièrement les lignes avec une valeur *Teaching Score* nulle sont supprimés, les lignes sont ensuite groupées par la colonne *Location* et à la fin, les fonctions `avg`, `min`, `max`, `len` sont appliquées sur la colonne *Teaching Score*.

```
1 def avg(valeurs) :
2     return sum(valeurs)/len(valeurs)
3 rankings[rankings["Teaching Score"].notnull()].groupby('Location')['Teaching Score'].agg([avg,min,max,len])
```

	avg	min	max	len
Location				
Algeria	19.538462	13.6	30.4	13
Argentina	16.666667	13.5	18.3	3
Australia	30.272000	17.8	67.1	25
Austria	32.763636	23.5	49.4	11
Azerbaijan	16.500000	16.5	16.5	1
...
United States	41.901754	17.1	94.8	171
Venezuela	15.900000	15.9	15.9	1
Vietnam	15.360000	13.7	18.7	5
Zambia	15.700000	15.7	15.7	1
Zimbabwe	14.800000	14.8	14.8	1

 *Fondamental : Ordre post groupby*

En regardant à nouveau le résultat du *groupby*('Location'), nous pouvons voir que le résultat est indexé par la colonne *Location*.

Trie

IV

Pour obtenir les données dans l'ordre souhaité, nous pouvons les trier. La méthode `sort_values()` est pratique pour cela. L'instruction suivante trie les entrées du DataFrame selon les valeurs de la colonne `Location` d'une manière ascendante:

```
l rankings.sort_values('Location')
```

University Rank	Name of University	Location	No of student	student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score	Citations Score	Industry Income Score	International Outlook Score	
2059	Reporter	Polytechnic University of Tirana	Albania	8,591	16.2	2%	NaN	NaN	NaN	NaN	NaN	NaN	
1950	Reporter	University of Laghouat	Algeria	34,345	35.2	0%	51 : 49	NaN	NaN	NaN	NaN	NaN	
1838	Reporter	École Polytechnique d'Architecture et d'Urbanisme	Algeria	1,371	10.1	0%	69 : 31	NaN	NaN	NaN	NaN	NaN	
1837	Reporter	École Nationale Polytechnique	Algeria	1,489	8.7	0%	46 : 54	NaN	NaN	NaN	NaN	NaN	
1159	1201–1500	Université 8 Mai 1945 Guelma	Algeria	17,530	20.1	1%	67 : 33	18.4–24.3	16.2	8.6	41.3	37.0	36.5
...	
2336	-	University of the West of Scotland	NaN	NaN	NaN	NaN	NaN	34.0–39.2	24.1	15.5	61.5	37.9	76.8
2337	-	University of Windsor	NaN	NaN	NaN	NaN	NaN	34.0–39.2	35.1	29.4	34.5	44.2	88.7
2338	-	University of Wolverhampton	NaN	NaN	NaN	NaN	NaN	34.0–39.2	18.2	14.3	68.8	37.3	72.0
2339	-	University of Wuppertal	NaN	NaN	NaN	NaN	NaN	34.0–39.2	26.4	26.7	52.8	52.1	47.6
2340	-	Xi'an Jiaotong-Liverpool University	NaN	NaN	NaN	NaN	NaN	34.0–39.2	17.8	14.8	68.2	38.2	72.4

Cependant, parfois, nous souhaitons un tri décroissant. Cela est faisable avec le paramètre optionnel `ascending=False`:

```
l rankings.sort_values('Location', ascending=False)
```

University Rank	Name of University	Location	No of student	No of student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score	Citations Score	Industry Income Score	International Outlook Score	
1158	1001–1200	University of Zimbabwe	Zimbabwe	20,598	23.9	0%	51 : 49	24.4–29.7	14.8	23.3	35.6	43.1	50.0
591	501–600	University of Zambia	Zambia	16,903	21.2	1%	35 : 65	39.3–42.0	15.7	9.2	96.8	38.4	49.9
1999	Reporter	Mulungushi University	Zambia	6,988	54.6	0%	41 : 59	NaN	NaN	NaN	NaN	NaN	NaN
1539	1501+	Hanoi University of Science and Technology	Vietnam	32,758	31.1	0%	22 : 78	10.4–18.3	14.9	12.3	20.2	43.5	39.4
1686	1501+	Vietnam National University (Ho Chi Minh City)	Vietnam	75,704	21.7	1%	46 : 54	10.4–18.3	15.1	10.1	20.7	41.5	35.6
...
2336	-	University of the West of Scotland	NaN	NaN	NaN	NaN	NaN	34.0–39.2	24.1	15.5	61.5	37.9	76.8
2337	-	University of Windsor	NaN	NaN	NaN	NaN	NaN	34.0–39.2	35.1	29.4	34.5	44.2	88.7
2338	-	University of Wolverhampton	NaN	NaN	NaN	NaN	NaN	34.0–39.2	18.2	14.3	68.8	37.3	72.0
2339	-	University of Wuppertal	NaN	NaN	NaN	NaN	NaN	34.0–39.2	26.4	26.7	52.8	52.1	47.6
2340	-	Xi'an Jiaotong-Liverpool University	NaN	NaN	NaN	NaN	NaN	34.0–39.2	17.8	14.8	68.2	38.2	72.4

Afin de trier selon plusieurs colonne, il suffit de passer une liste des noms de colonne à la méthode `sort_values`. Le code suivant trie les entrées selon la colonne `Location`, et pour les entrées ayant la même valeur de la colonne `Location`, ces derniers sont triés par la colonne `Name of University` (tri ascendant pour les deux colonnes).

```
1 rankings.sort_values(['Location', 'Name of University'])
```

University Rank	Name of University	Location	No of student	No of student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score	Citations Score	Industry Income Score	International Outlook Score	
2059	Reporter	Polytechnic University of Tirana	Albania	8,591	16.2	2%	NaN	NaN	NaN	NaN	NaN	NaN	
1462	1501+	Badji Mokhtar University – Annaba	Algeria	41,841	14.2	1%	63 : 37	10.4–18.3	20.0	8.1	11.3	37.1	32.7
1472	1501+	Bida 1 University	Algeria	30,076	20.4	1%	68 : 32	10.4–18.3	18.3	8.3	18.2	37.1	35.9
422	401–500	Ferhat Abbas Setif University 1	Algeria	34,637	22.7	1%	63 : 37	42.1–44.9	18.2	19.8	94.7	36.9	40.0
1883	Reporter	Hassiba Benboual University of	Algeria	30,121	22.3	1%	71 : 29	NaN	NaN	NaN	NaN	NaN	NaN

Pour trier par les valeurs d'index, la méthode `sort_index()` est utilisée.

```
1 rankings.set_index('Name of University', inplace=True)
2 rankings.sort_index().head()
```

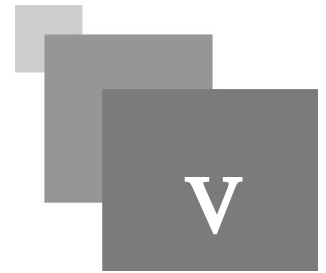
Name of University	University Rank	Location	No of student	No of student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score	Citations Score	Industry Income Score	International Outlook Score
AECC University College	Reporter	United Kingdom	615	15.4	43%	47 : 53	NaN	NaN	NaN	NaN	NaN	NaN
AGH University of Krakow	1201–1500	Poland	20,240	12.7	3%	35 : 65	18.4–24.3	20.6	18.6	18.7	56.8	25.5
Aalborg University	251–300	Denmark	16,818	14.7	14%	NaN	48.9–51.1	29.5	43.3	73.2	51.5	76.4
Aalto University	201–250	Finland	13,306	19.7	21%	38 : 62	51.2–54.3	38.3	40.4	67.5	53.2	81.9
Aarhus University	117	Denmark	26,475	13.5	10%	55 : 45	61.2	40.6	60.0	77.6	74.1	77.9



Remarque : Argument inplace

Afin d'appliquer le changement d'index sans réaffecter le résultat, le paramètre optionnel *inplace=True* est utilisé.

Renommer les colonnes



La première fonction que nous présenterons ici est `rename()`, qui permet de modifier les noms d'index et/ou les noms de colonnes. Par exemple, pour modifier la colonne nommée *Name of University* de notre DataFrame de données en *University*, nous ferions :

```
1 rankings.rename(columns={'Name of University': 'University'}, inplace=True)
2 rankings.head()
```

	University Rank	University	Location	No of student	No of student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score	Citations Score
0	1	University of Oxford	United Kingdom	20,965	10.6	42%	48 : 52	96.4	92.3	99.7	99.0
1	2	Harvard University	United States	21,887	9.6	25%	50 : 50	95.2	94.8	99.0	99.3
2	3	University of Cambridge	United Kingdom	20,185	11.3	39%	47 : 53	94.8	90.9	99.5	97.0
3	3	Stanford University	United States	16,164	7.1	24%	46 : 54	94.8	94.2	96.7	99.8
4	5	Massachusetts Institute of Technology	United States	11,415	8.2	33%	40 : 60	94.2	90.7	93.6	99.8

D'ailleurs, il est possible de renommer les valeurs de l'index selon leurs positions. Dans le code suivant, l'index avec la valeur 3 est remplacé par la chaîne de caractères troisieme :

```
1 rankings.rename(index={3: 'troisieme'}, inplace=True)
2 rankings.head()
```

	University Rank	Name of University	Location	No of student	No of student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score	Citations Score
0	1	University of Oxford	United Kingdom	20,965	10.6	42%	48 : 52	96.4	92.3	99.7	99.0
1	2	Harvard University	United States	21,887	9.6	25%	50 : 50	95.2	94.8	99.0	99.3
2	3	University of Cambridge	United Kingdom	20,185	11.3	39%	47 : 53	94.8	90.9	99.5	97.0
troisieme	3	Stanford University	United States	16,164	7.1	24%	46 : 54	94.8	94.2	96.7	99.8
4	5	Massachusetts Institute of Technology	United States	11,415	8.2	33%	40 : 60	94.2	90.7	93.6	99.8

Il est aussi possible de renommer les axes. Par exemple, pour renommer l'axe index à *Classement*, il suffit d'écrire (`axis="index"` est équivalent à `axis=0`) :

```
1 rankings.rename_axis("Classement", axis="index", inplace=True)
```

```
2 #rankings.rename_axis("Classement",axis=0,inplace=True)
```

	University Rank	Name of University	Location	No of student	No of student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score
Classement										
0	1	University of Oxford	United Kingdom	20,965	10.6	42%	48 : 52	96.4	92.3	99.7
1	2	Harvard University	United States	21,887	9.6	25%	50 : 50	95.2	94.8	99.0
2	3	University of Cambridge	United Kingdom	20,185	11.3	39%	47 : 53	94.8	90.9	99.5
3	3	Stanford University	United States	16,164	7.1	24%	46 : 54	94.8	94.2	96.7
4	5	Massachusetts Institute of Technology	United States	11,415	8.2	33%	40 : 60	94.2	90.7	93.6

Pour renommer l'axe des colonnes :

```
1 rankings.rename_axis("Description",axis="columns",inplace=True)
2 #rankings.rename_axis("Description",axis=1,inplace=True)
```

Description	University Rank	Name of University	Location	No of student	No of student per staff	International Student	Female:Male Ratio	OverAll Score	Teaching Score	Research Score
Classement										
0	1	University of Oxford	United Kingdom	20,965	10.6	42%	48 : 52	96.4	92.3	99.7
1	2	Harvard University	United States	21,887	9.6	25%	50 : 50	95.2	94.8	99.0
2	3	University of Cambridge	United Kingdom	20,185	11.3	39%	47 : 53	94.8	90.9	99.5
3	3	Stanford University	United States	16,164	7.1	24%	46 : 54	94.8	94.2	96.7
4	5	Massachusetts Institute of Technology	United States	11,415	8.2	33%	40 : 60	94.2	90.7	93.6

Combinaison de données

VI

Lors de l'exécution d'opérations sur un DataFrame, nous aurons parfois besoin de combiner différents DataFrames et/ou Series de manière non triviale. Pandas dispose de deux méthodes principales pour accomplir ce travail. Par ordre de complexité croissante, ce sont `concat()` et `join()`.

Pour concaténer deux DataFrames verticalement :

```
1 df1 = pd.DataFrame([[ 'a', 1], [ 'b', 2]],
2                     columns=[ 'letter', 'number'], index=[0,1])
3 df2 = pd.DataFrame([[ 'c', 3], [ 'd', 4]],
4                     columns=[ 'letter', 'number'], index=[2,3])
5 resultat=pd.concat ([df1,df2])
6 #resultat=pd.concat ([df1,df2], axis=0)
7 resultat
```

	letter	number
0	a	1
1	b	2
2	c	3
3	d	4

Pour concaténer horizontalement deux DataFrames, il est possible d'utiliser l'option `axis=1`. Les lignes possédant les mêmes valeurs d'index sont concaténées dans la même ligne :

```
1 df1 = pd.DataFrame([[ 'a', 1], [ 'b', 2]],
2                     columns=[ 'letter', 'number'], index=[0,1])
3 df2 = pd.DataFrame([[ 'c', 3], [ 'd', 4]],
4                     columns=[ 'letter', 'number'], index=[0,1])
5 resultat=pd.concat ([df1,df2], axis=1)
6 resultat
```

	letter	number	letter	number
0	a	1	c	3
1	b	2	d	4

Lors de la concaténation horizontale, si la valeur d'index pour une ligne ne correspond pas entre les deux DataFrame, les valeurs des colonnes manquantes seront vides. Dans l'exemple suivant, df1 et df2 ne comportent pas une ligne correspondant à l'indice 3 et 1 respectivement :

```

1 df1 = pd.DataFrame([[ 'a', 1], [ 'b', 2]],
2                     columns=[ 'letter', 'number'],index=[0,1])
3 df2 = pd.DataFrame([[ 'c', 3], [ 'd', 4]],
4                     columns=[ 'letter', 'number'],index=[0,3])
5 resultat=pd.concat([df1,df2],axis=1)
6 resultat

```

	letter	number	letter	number
0	a	1.0	c	3.0
1	b	2.0	NaN	NaN
3	NaN	NaN	d	4.0

La méthode *join()* d'un DataFrame permet de combiner celui-ci avec un autre objet DataFrame selon la valeur d'index (comme un *left join* en SQL). Dans l'exemple suivant, les lignes de df1 sont combinées avec les lignes de df2 selon la valeur de l'index :

```

1 df1 = pd.DataFrame([[ 'a', 1], [ 'b', 2]],
2                     columns=[ 'attribut_0', 'attribut_1'],index=[0,1])
3 df2 = pd.DataFrame([[ 'c', 3], [ 'd', 4]],
4                     columns=[ 'attribut_3', 'attribut_2'],index=[0,3])
5 df1.join(df2)

```

	attribut_0	attribut_1	attribut_3	attribut_2
0	a	1	c	3.0
1	b	2	NaN	NaN

S'il existe des colonnes ayant les mêmes noms entre les DataFrames combinés, il faudra spécifier un suffixe qui va être concaténé avec les noms de colonnes en commun. Dans cet exemple, df1 et df2 possèdent une colonne commune nommée *colonne_commune*. L'argument *lsuffix* et *rsuffix*, spécifient les suffixes ajoutés respectivement à la colonne de df1 et df2 :

```

1 df1 = pd.DataFrame([[ 'a', 1], [ 'b', 2]],
2                     columns=[ 'colonne_commune', 'attribut_1'],index=[0,1])
3 df2 = pd.DataFrame([[ 'c', 3], [ 'd', 4]],
4                     columns=[ 'colonne_commune', 'attribut_2'],index=[0,3])
5 df1.join(df2,lsuffix='df1', rsuffix='df2')

```

	colonne_communedf1	attribut_1	colonne_communedf2	attribut_2
0	a	1	c	3.0
1	b	2	NaN	NaN

La méthode *join* utilise toujours l'index de la colonne du DataFrame pour la combinaison, mais il est possible de spécifier le nom de la colonne utilisée pour la jointure. Dans l'exemple suivant, la colonne *attribut_0* est définie comme l'index *df2* et puis lors de la jointure, l'option *on* spécifie la colonne de la jointure (la colonne *attribut_0* de *df1*) :

```

1 df1 = pd.DataFrame([[ 'a', 1], [ 'b', 2]],
2                     columns=[ 'attribut_0', 'attribut_1'], index=[0,1])
3 df2 = pd.DataFrame([[ 'c', 3], [ 'b', 4]],
4                     columns=[ 'attribut_0', 'attribut_2'], index=[0,3])
5 df1.join(df2.set_index("attribut_0"), on="attribut_0" )

```

	attribut_0	attribut_1	attribut_2
0	a	1	NaN
1	b	2	4.0