Chapitre II Transformation des données

Cours Data Science M1 IA

Ilyas Bambrik

Table des matières

Introduction 3

I - **Fonctions sommaire** 4

II - **Transformation de colonne** 8

III - **Fonctions d'aggregation** 12

IV - **Trie** 15

V - **Renommer les colonnes** 18

VI - **Combinaison de données** 20

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()* :

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

```
count    1799.000000
mean       46.880378
std        22.582401
min        14.100000
25%        27.900000
50%        42.100000
75%        62.100000
max        99.700000
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 qui représentent une variable continue. Par contre, pour les données sous forme de chaîne de caractères (variable discrète), voici ce que nous obtenons :

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

```
count              2047
unique              116
top       United States
freq                173
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 obtenir la moyenne de la colonne *International Outlook Score*, nous pouvons utiliser la fonction *mean*() :

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

```
46.88037798777098
```

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

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

```
array(['United Kingdom', 'United States', 'Switzerland', nan, 'Canada',
       'Australia', 'Singapore', 'Japan', 'France', 'Sweden', 'China',
       'South Korea', 'Netherlands', 'Germany', 'Belgium', 'Finland',
       'Denmark', 'Austria', 'Norway', 'New Zealand', 'Spain', 'Italy',
```

```
       'Saudi Arabia', 'Luxembourg', 'Qatar', 'Brazil', 'Israel',
       'Ireland', 'Taiwan', 'India', 'United Arab Emirates',
       'Brunei Darussalam', 'Iceland', 'Lebanon', 'Philippines',
       'Portugal', 'Iran', 'Malaysia', 'Poland', 'Egypt', 'Turkey',
       'Greece', 'Vietnam', 'Algeria', 'Nigeria', 'Tanzania', 'Chile',
       'Pakistan', 'Ukraine', 'Romania', 'Czech Republic', 'South Africa',
       'Northern Cyprus', 'Hong Kong', 'Ethiopia', 'Jordan', 'Serbia',
       'Sri Lanka', 'Jamaica', 'Zambia', 'Iraq', 'Costa Rica', 'Cyprus',
       'Bangladesh', 'Mozambique', 'Colombia', 'Kenya', 'Namibia', 'Peru',
       'Latvia', 'Oman', 'Thailand', 'Lithuania', 'Slovenia', 'Uganda',
       'Malta', 'Nepal', 'Kazakhstan', 'Mexico', 'Botswana', 'Slovakia',
       'Ghana', 'Morocco', 'Georgia', 'Tunisia', 'Mauritius', 'Hungary',
       'Puerto Rico', 'Ecuador', 'Fiji', 'Croatia', 'Estonia', 'Zimbabwe',
       'Indonesia', 'Argentina', 'Bulgaria', 'Venezuela', 'Azerbaijan',
       'Cuba', 'Montenegro', 'Uzbekistan', 'Palestine', 'Kuwait',
       'Somalia', 'Libya', 'Moldova', 'Kyrgyzstan', 'Malawi', 'Paraguay',
       'Mongolia', 'Armenia', 'Sudan', 'Turkmenistan', 'Uruguay',
       '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()* :

```
rankings.Location.value_counts()
```

```
United States     173
Japan             150
United Kingdom    149
India              91
China              82
                 ...
Puerto Rico         1
Mozambique          1
Mauritius           1
Namibia             1
Kosovo              1
Name: Location, Length: 116, dtype: int64
```

Le résultat de la méthode *value_counts()*, est une série avec avec comme index les valeurs uniques de la colonne désignée.

Pour transposer un DataFrame, la méthode *.transpose()* ou l'attribut *.T* sont utilisés. Ces deux derniers renvoie une copie du DataFrame transposé :

```
df.set_index("Name of University",inplace=True)
df.T.head() # .T est équivalent à df.transpose()
```

Name of University	University of Oxford	Harvard University	University of Cambridge	Stanford University	Massachusetts Institute of Technology	California Institute of Technology	Princeton University	University of California, Berkeley	Yale University	Imperial College London	...	Ulster University	Umm Al-Qura University	U
University Rank	1	2	3	3	5	6	7	8	9	10	...	-	-	
Location	United Kingdom	United States	United Kingdom	United States	United States	United States	United States	United States	United States	United Kingdom	...	NaN	NaN	
No of student	20,965	21,887	20,185	16,164	11,415	2,237	8,279	40,921	13,482	18,545	...	NaN	NaN	
No of student per staff	10.6	9.6	11.3	7.1	8.2	6.2	8	18.4	5.9	11.2	...	NaN	NaN	
International Student	42%	25%	39%	24%	33%	34%	23%	24%	21%	61%	...	NaN	NaN	

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 :

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

Pour extraire le minimum et le maximum d'une colonne/série, les méthodes *.min()* et *.max()* sont utilisées. En outre, pour trouver l'index du minimum et maximum les méthodes *.idxmin()* et *.idxmax()* sont appropriées. Le code suivant affiche le minimum/maximum de la colonne *International Outlook* Score ainsi que l'index du minimum/maximum.

```
df.set_index("Name of University",inplace=True)
print(df["International Outlook Score"].min()," , ",df["International Outlook Score"].max())
print(df["International Outlook Score"].idxmin()," , ",df["International Outlook Score"].idxmax())
```

```
14.1  ,  99.7
R V College of Engineering  ,  Macau University of Science and Technology
```

La méthode *.mode()* renvoie la valeur la plus fréquente dans la colonne et la méthode *.median()* renvoie l'élément au milieu après le classement des valeurs. L'exemple suivant affiche la médiane de la colonne *International Outlook Score* et la valeur la plus fréquente de la colonne *Location*.

```
print(df['International Outlook Score'].median())
print(df.Location.mode())
```

```
42.1
0    United States
dtype: object
```

Pour obtenir l'écart type, la méthode *.std()* est utilisée et la corrélation entre deux colonnes est accessible avec la méthode *.corr()*. Le code suivant affiche l'écart type de *Teaching Score* et la corrélation entre la colonne *No of student per staff* et *Teaching Score*.

```
print(df['Teaching Score'].std())
print(df['No of student per staff'].corr(df['Teaching Score']))
```

```
13.282242791347766
-0.13056850512148
```

Pour une *colonne/série* contenant des valeurs numériques, la méthode *.sum()* retourne la somme. Pour un *DataFrame*, .sum() retourne la somme de chaque colonne dans une série indexée par les noms des colonnes. Prenant les colonnes, de *OverAll Score* jusqu'à *International Outlook Score*, du DataFrame *wr2023.csv*. Le code suivant affiche la somme de la colonne *Teaching Score* (*Ligne 2*) ainsi que la somme de toutes les colonnes de df (*Ligne 4*):

```
df=df.loc[:,"OverAll Score":]
print(df['Teaching Score'].sum())
print()
print(df.sum())
```

```
48605.4

Teaching Score                 48605.4
Research Score                 41407.4
Citations Score                87244.1
Industry Income Score          84741.1
International Outlook Score    84337.8
dtype: float64
```

Lors de l'invocation de *.sum()* avec un DataFrame, cette méthode est appelée avec l'argument optionnel *axis=0* par défaut. Avec *axis=0*, la somme est effectuée verticalement sur chaque colonne. Par contre avec *axis=1*, la somme est effectuée sur chaque ligne :

```
df.sum(axis=1).head()
```

```
Name of University
University of Oxford                     462.1
Harvard University                       423.1
University of Cambridge                  437.4
Stanford University                      435.5
Massachusetts Institute of Technology    464.3
dtype: float64
```

De même, *DataFrame.min()/DataFrame.max()* donne l' minimum/maximum de chaque colonne. Ces-deux dernières méthodes sont appelées avec l'argument axis=0 par défaut. Par ailleurs, *DataFrame.min(axis=1) /DataFrame.max(axis=1)* donne le minimum/maximum pour chaque ligne :

```
df.min()
```

```
Teaching Score                 11.6
Research Score                  7.4
Citations Score                 0.8
Industry Income Score          36.9
International Outlook Score    14.1
dtype: float64
```

```
df.min(axis=1).head()
```

```
Name of University
University of Oxford                     74.9
Harvard University                       49.5
University of Cambridge                  54.2
Stanford University                      65.0
Massachusetts Institute of Technology    89.3
dtype: float64
```

Comme *min()* et *max()*, *median(axis=1)* donne la médiane de chaque ligne :

```
df.median(axis=1).head()
```

```
University of Oxford                     96.2
Harvard University                       94.8
University of Cambridge                  95.8
Stanford University                      94.2
Massachusetts Institute of Technology    90.9
dtype: float64
```

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* proposées par les classes DataFrame et Series 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 celle-ci 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 :

year	A	B	C	D
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 le code suivant, le résultat retourné par la méthode apply est un DataFrame avec les colonnes A et B centrées :

```
moyenne_colA=df.A.mean()
moyenne_colB=df.B.mean()

def centrer(ligne):
    ligne.A=ligne.A-moyenne_colA
    ligne.B=ligne.B-moyenne_colB
    return ligne

df.apply(f,axis=1)
```

year	A	B	C	D
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.

```
def moyenne(colonne):
```

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

```
A    256.6
B    653.0
C     91.6
D      0.6
dtype: float64
```

Remarque : apply

- 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 la surcharge des opérateurs arithmétiques (+, - / ,etc). Par exemple, voici un moyen plus rapide de centrer notre colonne A:

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

```
year
2012   -59.6
2011   -49.6
2010    45.4
2009    46.4
2008    17.4
Name: A, dtype: float64
```

Dans ce code, nous effectuons une opération entre une liste 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 voulons soustraire cette valeur moyenne de chaque valeur de la série.

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

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

```
year
2012    197 - 577
2011    207 - 614
2010    302 - 649
2009    303 - 754
2008    274 - 671
dtype: object
```

Remarque : astype

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

La méthode *.cumsum()* donne une série contenant la somme cumulative à chaque position de la série.

```
S=pd.Series([10,20,30,40,50,60,70,80])
```

```
S.cumsum()
```

```
0     10
1     30
2     60
3    100
4    150
5    210
6    280
7    360
dtype: int64
```

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() :*

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

```
location
Algeria      86
Angola        1
Argentina     7
Armenia       4
Australia    37
             ..
Venezuela     3
Vietnam       7
Yemen         4
Zambia        2
Zimbabwe      2
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* pour compter le nombre de valeurs non nulles avec la méthode *count()*. *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 :

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

```
Location
Albania       NaN
Algeria      30.4
Argentina    18.3
Armenia       NaN
Australia    67.1
             ...
Uzbekistan    NaN
Venezuela    15.9
Vietnam      18.7
Zambia       15.7
Zimbabwe     14.8
Name: Teaching Score, Length: 116, dtype: float64
```

Vous pouvez considérer chaque groupe que nous générons avec *groupby()* comme étant une tranche de notre DataFrame contenant uniquement les lignes qui correspondent à une valeur unique. Ces tranches du DataFrame nous sont accessibles directement à l'aide de la méthode *apply()*, et nous pouvons ensuite manipuler les données regroupées dedans. Par exemple, voici une façon de sélectionner la meilleur université dans chaque payé:

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

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
Albania	Reporter	Polytechnic University of Tirana	Albania	8,591	16.2	2%	NaN	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 Martín	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* de chaque tranche.

```
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])
```

Location	avg	min	max	len
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:

```
1 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	NaN
1950	Reporter	University of Laghouat	Algeria	34,345	35.2	0%	51 : 49	NaN	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	NaN
1837	Reporter	École Nationale Polytechnique	Algeria	1,489	8.7	0%	46 : 54	NaN	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*:

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

	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
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 colonnes, il suffit de passer une liste des noms des colonnes 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	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+	Blida 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 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
1883	Reporter	Hassiba Benbouali	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.

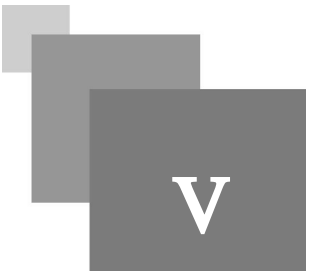
```
1 rankings.set_index('Name of University',inplace=True)
2 rankings.sort_index().head()
```

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

Comme mentionner, pour plusieurs méthode comme *set_index(), reset_index(), sort_values() et sort_index()*, celles-ci renvoient une copie modifiée du DataFrame. L'argument *inplace* est optionnel pour les méthodes précédentes et permet d'appliquer les changements sur l'objet DataFrame directement.

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 :

```
rankings.rename(columns={'Name of University':'University'},inplace=True)
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. Dans le code suivant, l'index avec la valeur 3 est remplacé par la chaîne de caractères *"trosieme"* :

```
rankings.rename(index={3:'trosieme'},inplace=True)
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	C
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	
trosieme	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	

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) :

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

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

Classement	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
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 :

```
rankings.rename_axis("Description",axis="columns",inplace=True)
#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 des 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 (équivalent à UNION dans SQL) :

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

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

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

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

```
df1 = pd.DataFrame([['a', 1], ['b', 2]],
                   columns=['colonne_commune', 'attribut_1'],index=[0,1])
df2 = pd.DataFrame([['c', 3], ['d', 4]],
                   columns=['colonne_commune', 'attribut_2'],index=[0,3])
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 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 dans *df1*:

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