Aix*Marseille université
Socialement engagée

Pandas

Patrice Bellot
patrice.bellot@univ-amu.fr

juin 2022

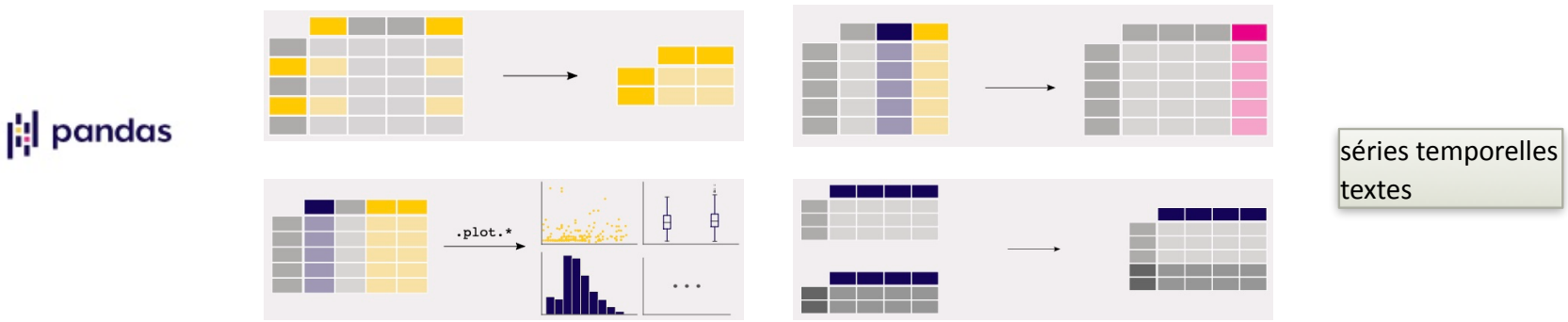
Introduction

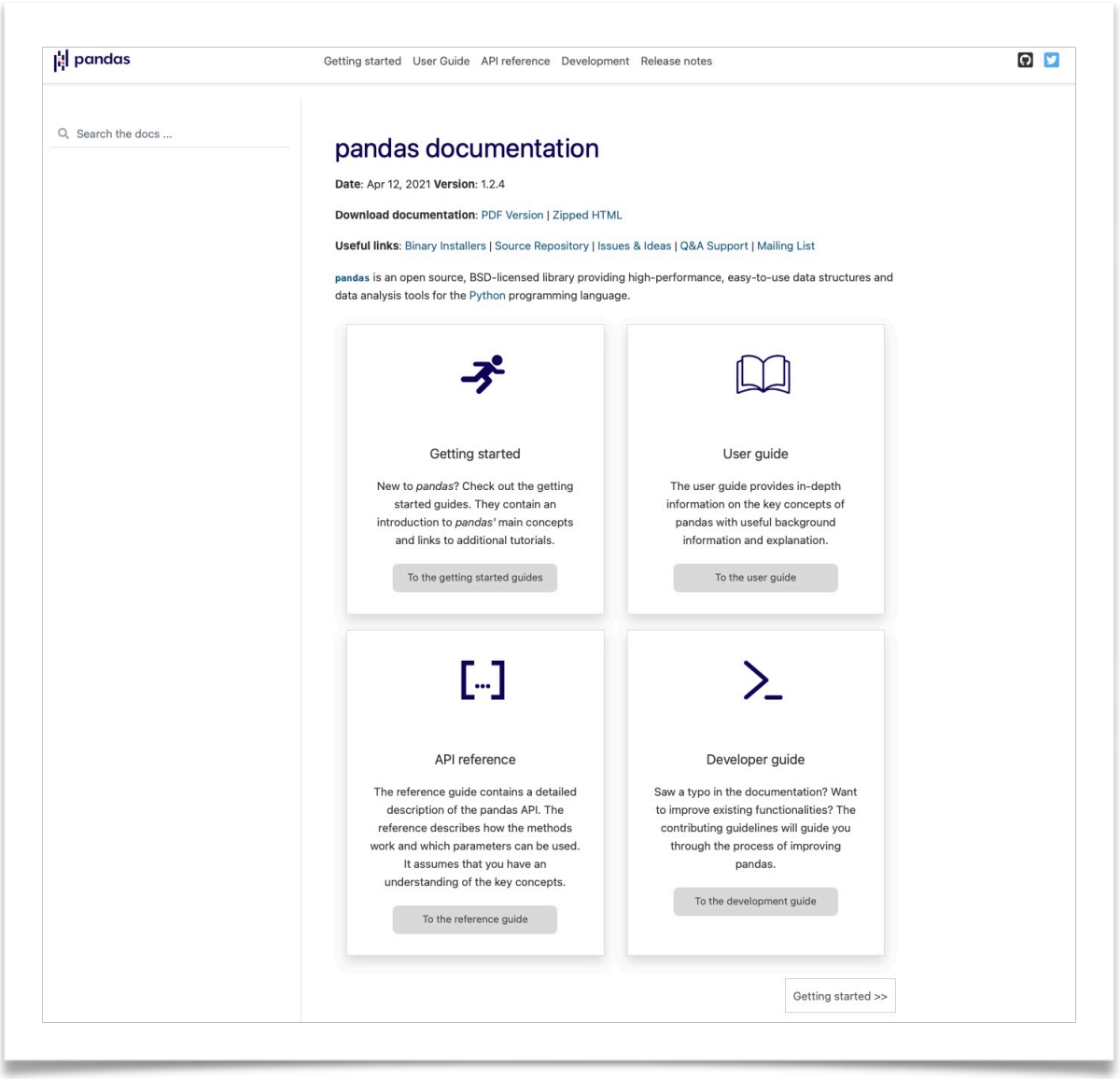
— Pandas a pour objectif de fournir les structures de données et les méthodes pour le traitement des données, leur extraction, leur manipulation

— Les données peuvent provenir de différentes sources et formats : CSV, Excel, SQL… = des **données tabulaires**

— Pandas s'intègre bien avec les autres bibliothèques de fouille de données, de calcul ou d'apprentissage automatique : NumPy, SciKitLearn, Keras…

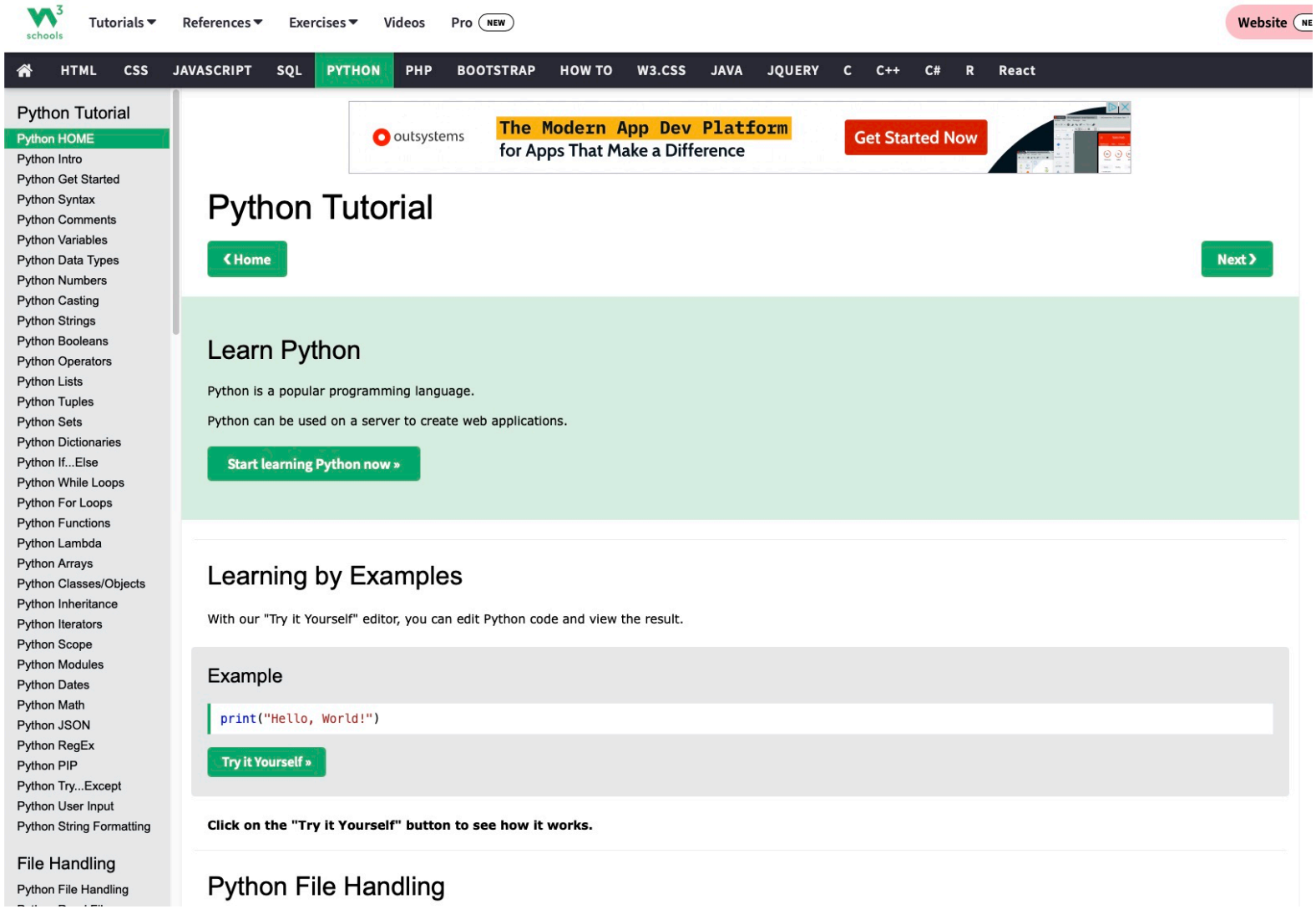
— NumPy est en quelque sorte la base de Pandas (beaucoup d'opérations sur les *arrays* NumPy s'appliquent aux *Series* Pandas, les *dataframes* peuvent souvent être utilisées comme des matrices)





https://pandas.pydata.org/pandas-docs/stable/index.html

https://www.w3schools.com/python/default.asp



Plan

— Manipulations de base sur les *Series* et les *Dataframes*
— Importation de données (.csv notamment)
— Opérations de transformation des *Dataframes*
— Statistique descriptive et graphiques
— Les séries temporelles
— Le cas des données textuelles avec 2 exemples

Aix*Marseille université
Socialement engagée

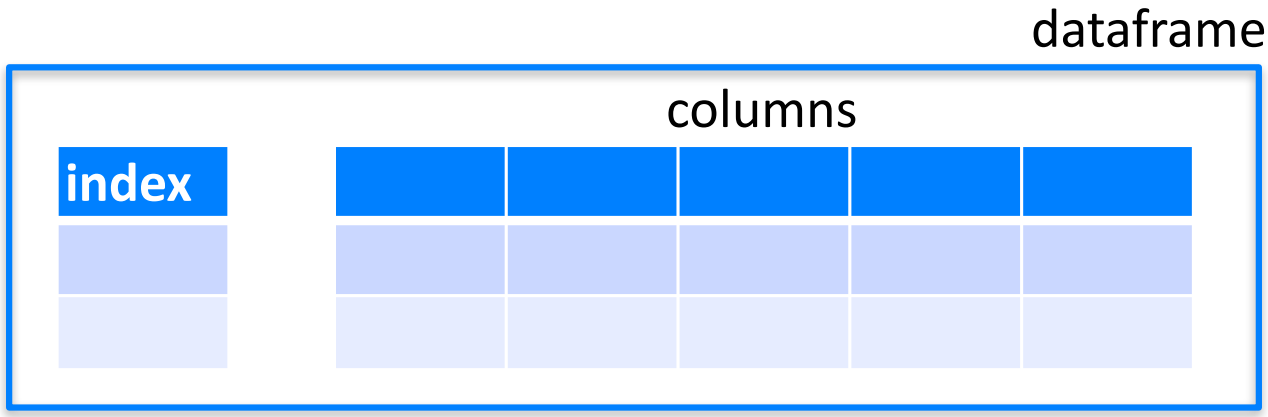
Dataframes et *Series*
accès aux données

Les structures de données : *Series* et *Dataframes*

— *Series*: des données à une dimension (value) repérées par des labels (index)

index	value

— *Dataframes*: les données ont plusieurs dimensions, chacune dans une colonne



Series

```
import pandas

s = pandas.Series()
```

```
import pandas as pd

s = pd.Series()
```

```
import pandas as pd

s = pd.Series([12,5])
print(s)
```

```
s = pd.Series(['a',"hjhkhhj"])
s = pd.Series([4,'rtg'])
```

```
18 s= pd.Series([4,'rtg',5,1000,0.5,"abcd"])
   print(s)

   0        4
   1      rtg
   2        5
   3     1000
   4      0.5
   5     abcd
   dtype: object
```

index

```
print(s[1])
```

```
s= pd.Series([4,'rtg',5,1000,0.5,"abcd"], index=["x1","x2","x3","x4","x5","x6"])
```

```
x1       4
x2     rtg
x3       5
x4    1000
x5     0.5
x6    abcd
```

```
print(s[1])
print(s["x2"])
print(s[0:3])
print(s["x2":"x5"])
print(s[["x2","x5"]])
print(s[[1,4]])
```

```
s = pd.Series([4,1,10,15])
print(s[s>1])
print(s[s==4])
```

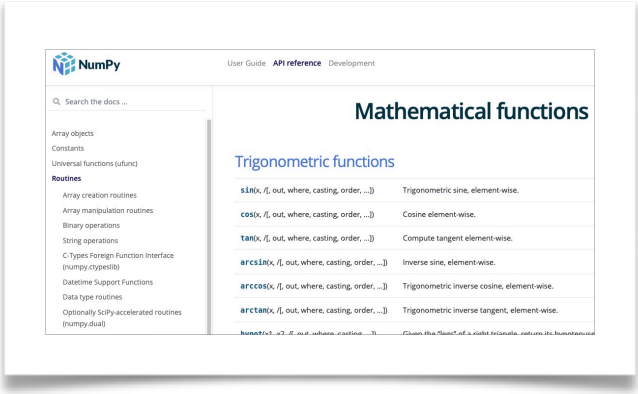
```
33 s = pd.Series([4,1,10,15])
   print(s[s>1])
   print(s[s==4])

   0     4
   2    10
   3    15
   dtype: int64
   0    4
   dtype: int64
```

```
t= s[s>1]
```

```
52 t*2

52 0     8
   2    20
   3    30
   dtype: int64
```

```
t*2

np.log2(t)
```

```
53 np.log2(t)

53 0    2.000000
   2    3.321928
   3    3.906891
   dtype: float64
```



https://numpy.org/doc/stable/reference/routines.math.html

```
s = pd.Series([10,10,5,4,2,1,2], index=['a','b','c','c','d','e','e'])
```

```
a    10
b    10
c     5
c     4
d     2
e     1
e     2
```

`s.unique()`

```
58  s.unique()
58  array([10,  5,  4,  2,  1])
```

`s.value_counts()`

```
59  s.value_counts()
59  10    2
    2     2
    1     1
    4     1
    5     1
    dtype: int64
```

`s.isin([10,1])`

```
62  s.isin([10,1])
62  a     True
    b     True
    c    False
    c    False
    d    False
    e     True
    e    False
    dtype: bool
```

`s[s.isin([10,1])]`

```
a    10
b    10
c     5
c     4
d     2
e     1
e     2
```

```
a     True
b     True
c    False
c    False
d    False
e     True
e    False
```

```
s[s.isin([10,1])]
```

```
65  s[s.isin([10,1])]

65  a    10
    b    10
    e     1
```

```
s[[0,1,5]]
```

```
s1 = pd.Series([1,1,1,1], index=['a','b','c','d'])
s2 = pd.Series([2,20,200], index=['b','e','c'])

s1+s2
```

```
70  s1 = pd.Series([1,1,1,1], index=['a','b','c','d'])
    s2 = pd.Series([2,20,200], index=['b','e','c'])
    s1+s2
```

```
70  a      NaN
    b      3.0
    c    201.0
    d      NaN
    e      NaN
    dtype: float64
```

Dataframes

index	nom colonne 1	nom colonne 2	nom colonne 3	nom colonne 4	nom colonne 5
0					
1					

```
data = { 'col1' : [10,20,30,40],
    'col2' : ['a','b','c','d'],
        'col3' : [1.1,1.2,1.3,1.4]
}
```

```
df = pd.DataFrame(data)
print(df)
```

```
   col1 col2  col3
0    10    a   1.1
1    20    b   1.2
2    30    c   1.3
3    40    d   1.4
```

Dataframes

```
{'col1': [10, 20, 30, 40], 'col2': ['a', 'b', 'c', 'd'], 'col3': [1.1, 1.2, 1.3, 1.4]}
```

```
df = pd.DataFrame(data, columns=['col1','col3'], index=['item1','item2','item3','item4'])
```

```
78 df = pd.DataFrame(data, columns=['col1','col3'], index=['item1','item2','item3','item4'])
   print(df)
```

```
       col1  col3
item1    10   1.1
item2    20   1.2
item3    30   1.3
item4    40   1.4
```

Comment créer un *data frame* depuis un *array* Numpy ?

```
df = pd.DataFrame(np.array([1,2,3,4,5,6]).reshape((3,2)),
       index=['a','b','c'], columns=['c1','c2'])
```

```
83 df = pd.DataFrame(np.array([1,2,3,4,5,6]).reshape((3,2)), index=['a','b','c'], columns=['c1','c2'])
   print(df)

   c1  c2
a   1   2
b   3   4
c   5   6
```

```
[1]   1 import numpy as np

[2]   1 a=np.mat('[1 2 4 ; 3 4 5.]')

[3]   1 a
```

```
matrix([[1., 2., 4.],
        [3., 4., 5.]])
```

```
[5]   1 import pandas as pd
      2 df = pd.DataFrame(a)
      3 df
```

	0	1	2
0	1.0	2.0	4.0
1	3.0	4.0	5.0

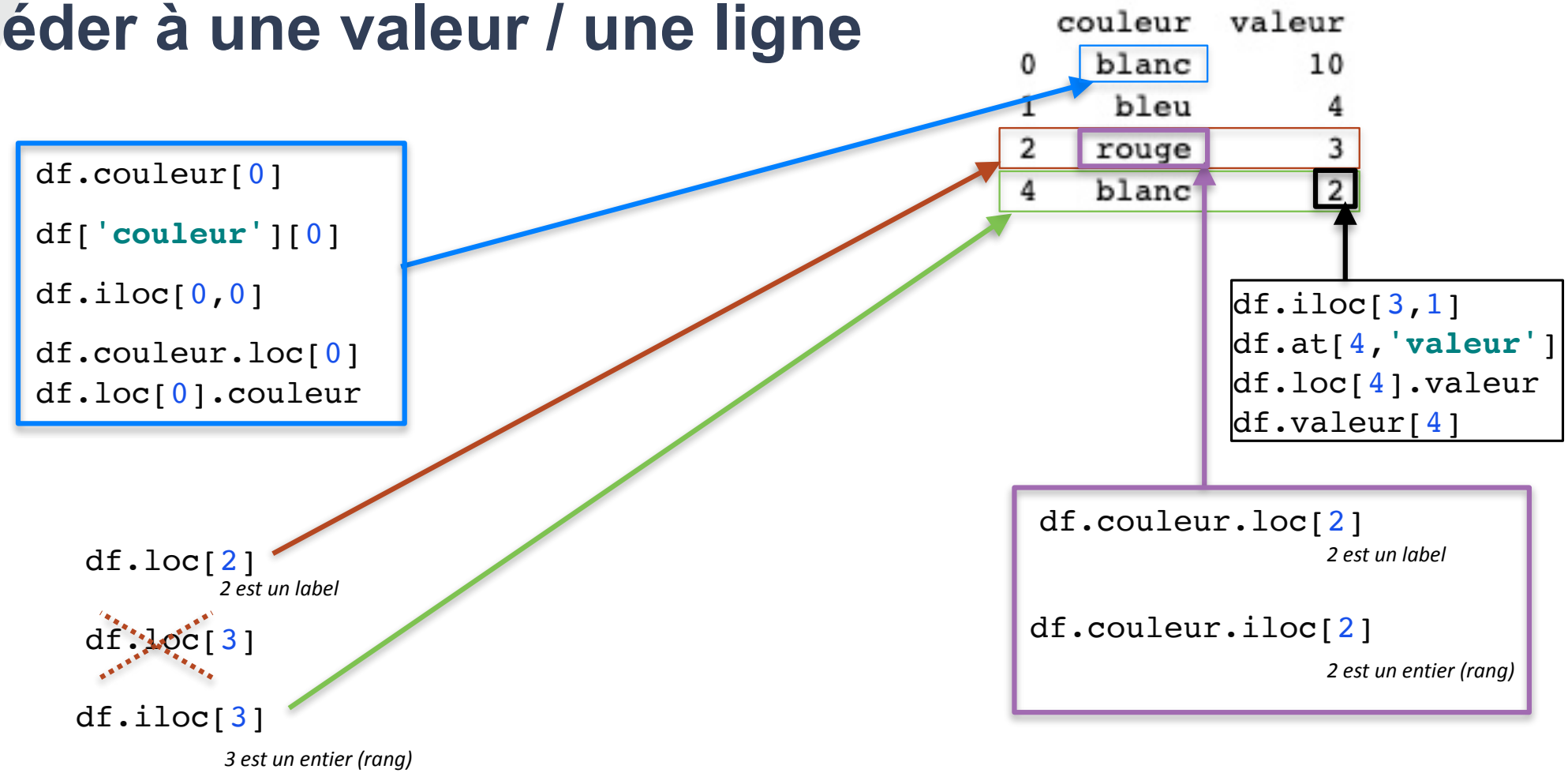
Data table display (Google Colab)

```
df.columns

df.columns[0]

df.index

df.index[1]

df.values

df['c2']

df.c2
```

	c1	c2
a	1	2
b	3	4
c	5	6

```
86 df.columns
86 Index(['c1', 'c2'], dtype='object')

88 df.columns[0]
88 'c1'

90 df.index
90 Index(['a', 'b', 'c'], dtype='object')

92 df.index[1]
92 'b'

94 df.values
94 array([[1, 2],
          [3, 4],
          [5, 6]])

96 df['c2']
96 a    2
   b    4
   c    6
   Name: c2, dtype: int64

97 df.c2
97 a    2
   b    4
   c    6
   Name: c2, dtype: int64
```

Accéder à une valeur / une ligne



Comment accéder aux lignes ?

```
df[0]
```

mais… attention !

```
df = pd.DataFrame(np.array([1,2,3,4,5,6]).reshape((3,2)),
index=['a','b','c'], columns=[10,'c2'])
```

```
df[10]
```

```
df[10][2]
```

	10	c2
a	1	2
b	3	4
c	5	6

```
a    1
b    3
c    5
```

```
103 df[10][2]
103 5
```

```
109 df
```

	10	c2
a	1	2
b	3	4
c	5	6

```
df['c2'][1]
```

```
[18]  1 df['c2'][1]
      4
```

```
df.loc['b']
```

```
112 df.loc['b']
112 10    3
    c2    4
    Name: b, dtype: int64
```

```
df.loc['b','c2']
```

```
1 df.loc['b','c2']
4
```

```
1 df.loc['b']['c2']
4
```

```
df[1:2]
```

```
114 df[1:2]
```

	10	c2
b	3	4

```
df = pd.DataFrame(data, columns=['col1','col3'])
```

```
df
```

```
df.loc[1]
```

```
df.loc[[0,2]]
```

```
df.loc[0:2]
```

du rang 0 au rang 2 inclus

!

```
df[0:2]
```

du rang 0 au rang 2 NON inclus

```
124 data
```

```
124 {'col1': [10, 20, 30, 40],
     'col2': ['a', 'b', 'c', 'd'],
     'col3': [1.1, 1.2, 1.3, 1.4]}
```

```
118 df = pd.DataFrame(data, columns=['col1','col3'])
    df
```

118

	col1	col3
0	10	1.1
1	20	1.2
2	30	1.3
3	40	1.4

```
121 df.loc[1]
```

```
121 col1    20.0
    col3     1.2
    Name: 1, dtype: float64
```

```
123 df.loc[[0,2]]
```

123

	col1	col3
0	10	1.1
2	30	1.3

```
128 df.loc[0:2]
```

128

	col1	col3
0	10	1.1
1	20	1.2
2	30	1.3

```
129 df[0:2]
```

129

	col1	col3
0	10	1.1
1	20	1.2

Comment supprimer une ligne ?

```
df.drop(1)
```

```
208 df.drop(1)
```

208

	col1	col3	colXX	colYY
0	10	1.1	10	0
2	30	1.3	10	2
3	40	1.4	10	3

Comment supprimer une colonne ?

```
df = df.drop("col1", axis=1)

del df.col1

df.pop("col1")
```

Sélectionner des valeurs

Python pur vs. méthode *query()*

	a	b	c
0	0.438921	0.118680	0.863670
1	0.138138	0.577363	0.686602
2	0.595307	0.564592	0.520630
3	0.913052	0.926075	0.616184
4	0.078718	0.854477	0.898725
5	0.076404	0.523211	0.591538
6	0.792342	0.216974	0.564056
7	0.397890	0.454131	0.915716
8	0.074315	0.437913	0.019794
9	0.559209	0.502065	0.026437

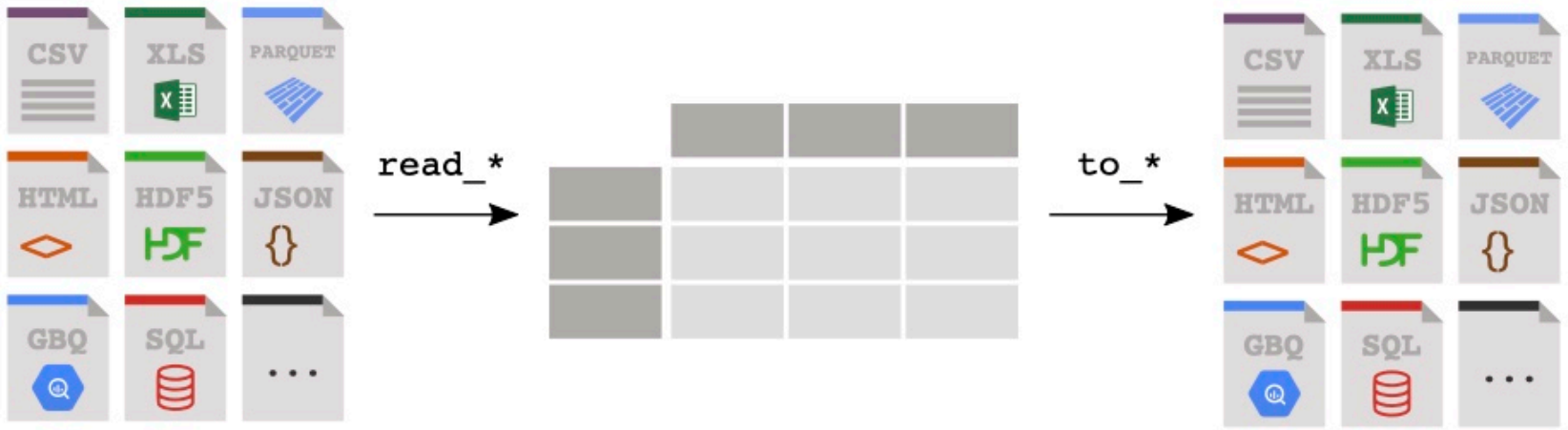
```
# pure python
In [218]: df[(df['a'] < df['b']) & (df['b'] < df['c'])]
Out[218]:
          a         b         c
1  0.138138  0.577363  0.686602
4  0.078718  0.854477  0.898725
5  0.076404  0.523211  0.591538
7  0.397890  0.454131  0.915716

# query
In [219]: df.query('(a < b) & (b < c)')
Out[219]:
          a         b         c
1  0.138138  0.577363  0.686602
4  0.078718  0.854477  0.898725
5  0.076404  0.523211  0.591538
7  0.397890  0.454131  0.915716
```

```
df.query('index < b < c')
```

Aix*Marseille université
Socialement engagée

Importation de données

Importation de données dans un data frame



https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/02_read_write.html#min-tut-02-read-write

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

Depuis un CSV

pandas.read_csv

pandas.read_csv(*filepath_or_buffer, sep=<object object>, delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None*) ¶ [source]

Read a comma-separated values (csv) file into DataFrame.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

sep : *str, default ','*

Delimiter to use. If sep is None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator by Python's builtin sniffer tool, `csv.Sniffer`. In addition, separators longer than 1 character and different from `'\s+'` will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: `'\r\t'`.

index_col : *int, str, sequence of int / str, or False, default* `None`

Column(s) to use as the row labels of the `DataFrame`, either given as string name or column index. If a sequence of int / str is given, a MultiIndex is used.
Note: `index_col=False` can be used to force pandas to *not* use the first column as the index, e.g. when you have a malformed file with delimiters at the end of each line.

`\d`	Digit
`\D`	Non-digit character
`\s`	Whitespace character
`\S`	Non-whitespace character
`\n`	New line character
`\t`	Tab character
`\uxxxx`	Unicode character specified by the hexadecimal number `xxxx`

quotechar : *str (length 1), optional*

The character used to denote the start and end of a quoted item. Quoted items can include the delimiter and it will be ignored.

thousands : *str, optional*

Thousands separator.

decimal : *str, default '.'*

Character to recognize as decimal point (e.g. use ',' for European data).

encoding : *str, optional*

Encoding to use for UTF when reading/writing (ex. 'utf-8'). List of Python standard encodings . .. versionchanged:: 1.2

latin_1 utf_8

parse_dates : ***bool or list of int or names or list of lists or dict, default False***

The behavior is as follows:

- boolean. If True -> try parsing the index.
- list of int or names. e.g. If [1, 2, 3] -> try parsing columns 1, 2, 3 each as a separate date column.
- list of lists. e.g. If [[1, 3]] -> combine columns 1 and 3 and parse as a single date column.
- dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call result 'foo'

If a column or index cannot be represented as an array of datetimes, say because of an unparsable value or a mixture of timezones, the column or index will be returned unaltered as an object data type. For non-standard datetime parsing, use `pd.to_datetime` after `pd.read_csv`. To parse an index or column with a mixture of timezones, specify `date_parser` to be a partially-applied **`pandas.to_datetime()`** with `utc=True`. See Parsing a CSV with mixed timezones for more.

Note: A fast-path exists for iso8601-formatted dates.

infer_datetime_format : ***bool, default False***

If True and *parse_dates* is enabled, pandas will attempt to infer the format of the datetime strings in the columns, and if it can be inferred, switch to a faster method of parsing them. In some cases this can increase the parsing speed by 5-10x.

dayfirst : ***bool, default False***

DD/MM format dates, international and European format.

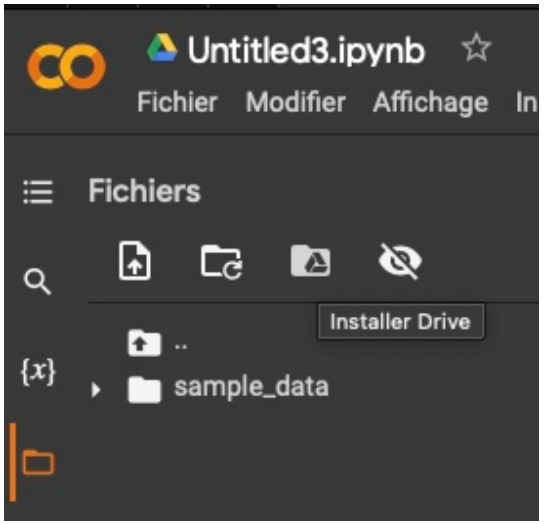
rappel sur les chemins d'accès fichiers

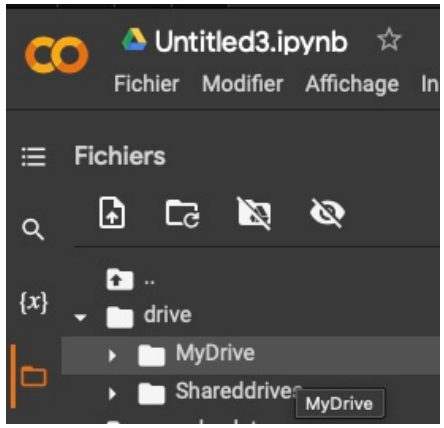
```
1 import os
2 os.getcwd()
```

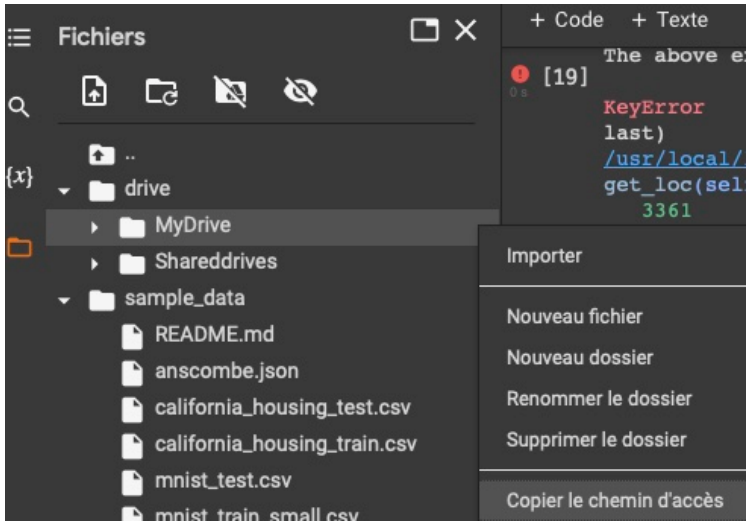
```
'/content'
```

```
1 os.chdir("/content/drive/MyDrive")
2 os.getcwd()
```

```
'/content/drive/MyDrive'
```







Exemple : les données *Titanic*

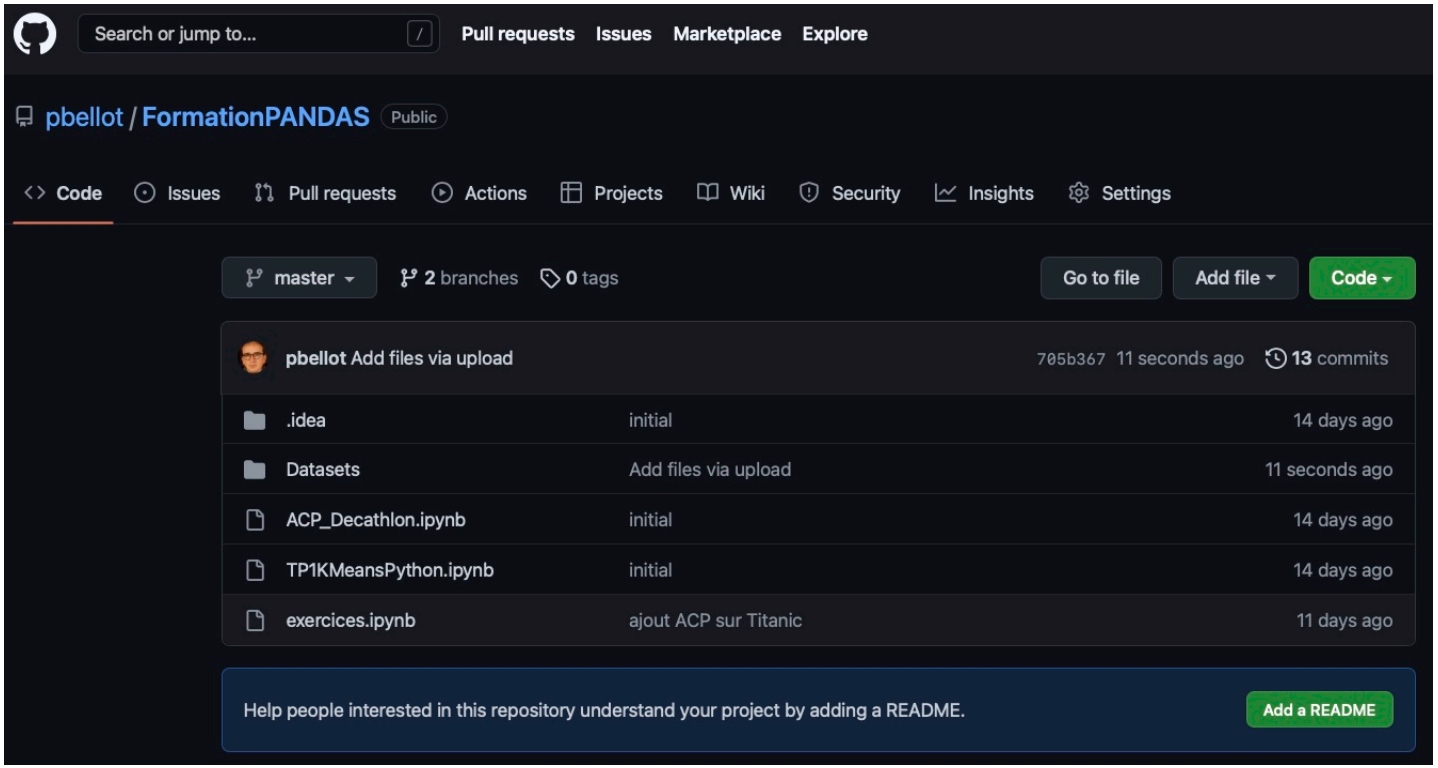
```
titanic = pd.read_csv("data/titanic.csv")
```

```
titanic.head(10)
```

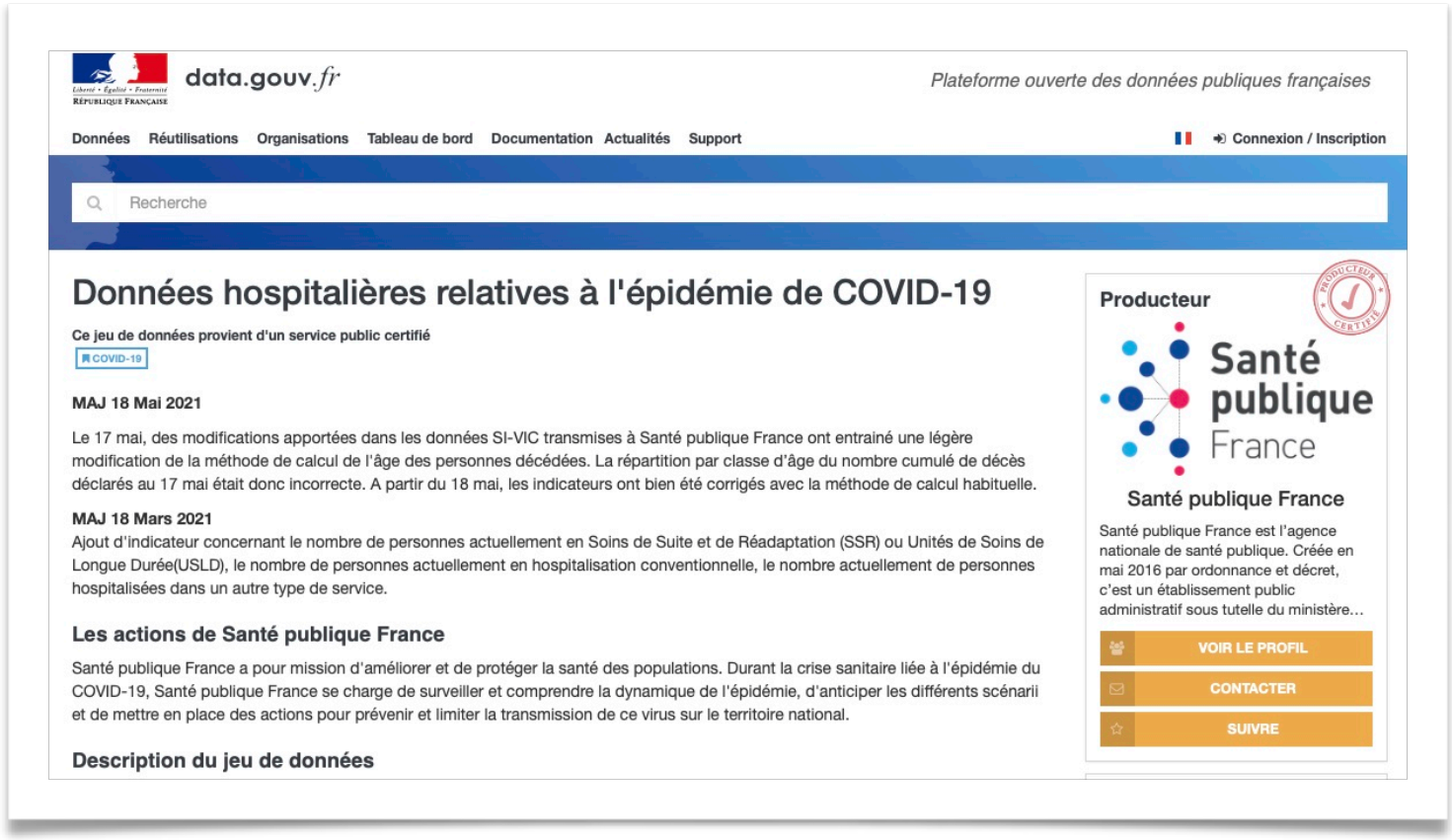
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William	male	35.0	0	0	373450	8.050(

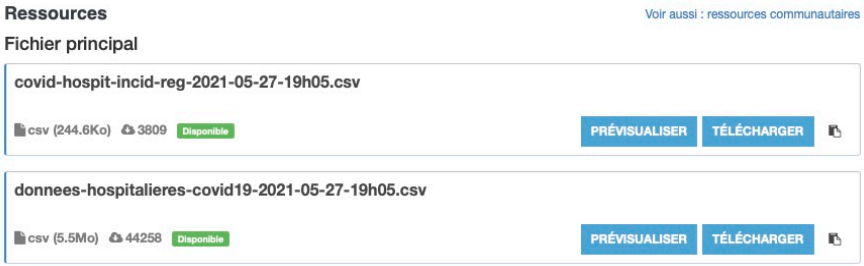
- PassengerId: Id of every passenger.
- Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
- Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger.
- SibSp: Indication that passenger have siblings and spouse.
- Parch: Whether a passenger is alone or have family.
- Ticket: Ticket number of passenger.
- Fare: Indicating the fare.
- Cabin: The cabin of passenger.
- Embarked: The embarked category.

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/06_calculate_statistics.html#min-tut-06-stats



https://github.com/pbellot/FormationPANDAS





https://www.data.gouv.fr/fr/datasets/donnees-hospitalieres-relatives-a-lepidemie-de-covid-19/#_

covid-hospit-incid-reg-2021-02-08-19h20.csv

```
"jour";"nomReg";"numReg";"incid_rea"
2020-03-19;"Auvergne-Rh<F4>ne-Alpes";84;44
2020-03-19;"Bourgogne-Franche-Comt<E9>";27;33
2020-03-19;"Bretagne";53;8
2020-03-19;"Centre-Val de Loire";24;6
2020-03-19;"Corse";94;11
2020-03-19;"Grand-Est";44;69
2020-03-19;"Guadeloupe";1;0
2020-03-19;"Guyane";3;0
2020-03-19;"Hauts-de-France";32;37
2020-03-19;"Ile-de-France";11;151
2020-03-19;"La R<E9>union";4;0
2020-03-19;"Martinique";2;0
2020-03-19;"Mayotte";6;0
2020-03-19;"Normandie";28;7
2020-03-19;"Nouvelle-Aquitaine";75;7
2020-03-19;"Occitanie";76;29
2020-03-19;"Pays de la Loire";52;11
2020-03-19;"Provence-Alpes-C<F4>te d'Azur";93;25
2020-03-20;"Auvergne-Rh<F4>ne-Alpes";84;16
2020-03-20;"Bourgogne-Franche-Comt<E9>";27;9
2020-03-20;"Bretagne";53;2
2020-03-20;"Centre-Val de Loire";24;4
2020-03-20;"Corse";94;0
2020-03-20;"Grand-Est";44;45
2020-03-20;"Guadeloupe";1;0
2020-03-20;"Guyane";3;0
2020-03-20;"Hauts-de-France";32;35
2020-03-20;"Ile-de-France";11;89
2020-03-20;"La R<E9>union";4;0
```

Colonne	Type	Description_FR
jour	string($date)	Date de notification
nomReg	string	Nom de la région
numReg	integer	Numéro de la région
incid_rea	integer	Nombre de nouveaux patients admis en réanimation dans les 24 dernières heures

https://www.data.gouv.fr/fr/datasets/donnees-hospitalieres-relatives-a-lepidemie-de-covid-19/#_

```
incidence = pd.read_csv('covid-hospit-incid-reg-2021-02-08-19h20.csv',
                        sep=';', encoding="latin_1")
incidence.head()
```

	jour	nomReg	numReg	incid_rea
0	2020-03-19	Auvergne-Rhône-Alpes	84	44
1	2020-03-19	Bourgogne-Franche-Comté	27	33
2	2020-03-19	Bretagne	53	8
3	2020-03-19	Centre-Val de Loire	24	6
4	2020-03-19	Corse	94	11

Skip row between header and data

```
In [200]: data = """;;;;
   .....:  ;;;;
   .....:  ;;;;
   .....:  ;;;;
   .....:  ;;;;
   .....:  ;;;;
   .....: ;;;;
   .....:  ;;;;
   .....:  ;;;;
   .....: ;;;;
   .....: date;Param1;Param2;Param4;Param5
   .....:     ;m²;°C;m²;m
   .....: ;;;;
   .....: 01.01.1990 00:00;1;1;2;3
   .....: 01.01.1990 01:00;5;3;4;5
   .....: 01.01.1990 02:00;9;5;6;7
   .....: 01.01.1990 03:00;13;7;8;9
   .....: 01.01.1990 04:00;17;9;10;11
   .....: 01.01.1990 05:00;21;11;12;13
   .....: """
   .....:
```

Option 1: pass rows explicitly to skip rows

```
In [201]: from io import StringIO

In [202]: pd.read_csv(
   .....:     StringIO(data),
   .....:     sep=";",
   .....:     skiprows=[11, 12],
   .....:     index_col=0,
   .....:     parse_dates=True,
   .....:     header=10,
   .....: )
   .....:
Out[202]:
                     Param1  Param2  Param4  Param5
date
1990-01-01 00:00:00       1       1       2       3
1990-01-01 01:00:00       5       3       4       5
1990-01-01 02:00:00       9       5       6       7
1990-01-01 03:00:00      13       7       8       9
1990-01-01 04:00:00      17       9      10      11
1990-01-01 05:00:00      21      11      12      13
```

Lire plusieurs fichiers .csv

```
rootDir = "/Users/Patrice/DataspellProjects/Physio/Data/La_planete_au_tresor/ecg/"
ecgDir = "electrocardiograms"
files = sorted(os.listdir(rootDir+ecgDir))
csvFiles = [f for f in files if f.endswith('.csv')]
```

```
print(csvFiles)
```

```
['ecg_2022-02-27_00.csv', 'ecg_2022-02-27_01.csv', 'ecg_2022-02-27_02.csv', 'ecg_2022-0
'ecg_2022-02-27_09.csv', 'ecg_2022-02-27_10.csv', 'ecg_2022-02-27_11.csv', 'ecg_2022-02
'ecg_2022-02-27_18.csv', 'ecg_2022-02-27_19.csv', 'ecg_2022-02-27_20.csv', 'ecg_2022-02
'ecg_2022-02-27_27.csv', 'ecg_2022-02-27_28.csv', 'ecg_2022-02-27_29.csv', 'ecg_2022-02
```

```
for f in csvFiles:
```

Fusionner plusieurs fichiers .csv en 1 df

```
In [190]: files = ["file_0.csv", "file_1.csv", "file_2.csv"]

In [191]: result = pd.concat([pd.read_csv(f) for f in files], ignore_index=True)
```

You can use the same approach to read all files matching a pattern. Here is an example using `glob`:

```
In [192]: import glob

In [193]: import os

In [194]: files = glob.glob("file_*.csv")

In [195]: result = pd.concat([pd.read_csv(f) for f in files], ignore_index=True)
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/cookbook.html#cookbook-csv

Lecture de fichiers XML

en-tête fichier export.xml généré par Apple Watch (rythme cardiaque)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HealthData [
<!-- HealthKit Export Version: 11 -->
<!ELEMENT HealthData (ExportDate,Me,(Record|Correlation|Workout|ActivitySummary|ClinicalRecord)*)>
<!ATTLIST HealthData
  locale CDATA #REQUIRED
>
<!ELEMENT ExportDate EMPTY>
<!ATTLIST ExportDate
  value CDATA #REQUIRED
>
<!ELEMENT Me EMPTY>
<!ATTLIST Me
  HKCharacteristicTypeIdentifierDateOfBirth         CDATA #REQUIRED
  HKCharacteristicTypeIdentifierBiologicalSex       CDATA #REQUIRED
  HKCharacteristicTypeIdentifierBloodType           CDATA #REQUIRED
  HKCharacteristicTypeIdentifierFitzpatrickSkinType CDATA #REQUIRED
>
<!ELEMENT Record ((MetadataEntry|HeartRateVariabilityMetadataList)*)>
<!ATTLIST Record
  type          CDATA #REQUIRED
  unit          CDATA #IMPLIED
  value         CDATA #IMPLIED
  sourceName    CDATA #REQUIRED
  sourceVersion CDATA #IMPLIED
  device        CDATA #IMPLIED
  creationDate  CDATA #IMPLIED
  startDate     CDATA #REQUIRED
  endDate       CDATA #REQUIRED
>
<!-- Note: Any Records that appear as children of a correlation also appear as top-level records in th
<!ELEMENT Correlation ((MetadataEntry|Record)*)>
<!ATTLIST Correlation
  type          CDATA #REQUIRED
  sourceName    CDATA #REQUIRED
  sourceVersion CDATA #IMPLIED
  device        CDATA #IMPLIED
  creationDate  CDATA #IMPLIED
  startDate     CDATA #REQUIRED
  endDate       CDATA #REQUIRED
>
```

```
<HealthData locale="fr_FR">
 <ExportDate value="2022-02-27 16:13:13 +0100"/>
 <Me HKCharacteristicTypeIdentifierDateOfBirth="" HKCharacteristicTypeIdentifierBiologicalSex="HKBiological
 <Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Apple Watch" sourceVersion="6.1.3" device="&l
09:29:35 +0100" startDate="2020-03-17 09:24:38 +0100" endDate="2020-03-17 09:24:38 +0100" value="92">
  <MetadataEntry key="HKMetadataKeyHeartRateMotionContext" value="1"/>
 </Record>
 <Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Apple Watch" sourceVersion="6.1.3" device="&l
11:41:52 +0100" startDate="2020-03-17 09:27:54 +0100" endDate="2020-03-17 09:27:54 +0100" value="93">
  <MetadataEntry key="HKMetadataKeyHeartRateMotionContext" value="0"/>
 </Record>
 <Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Apple Watch" sourceVersion="6.1.3" device="&l
11:06:59 +0100" startDate="2020-03-18 11:06:58 +0100" endDate="2020-03-18 11:06:58 +0100" value="82">
  <MetadataEntry key="HKMetadataKeyHeartRateMotionContext" value="0"/>
 </Record>
 <Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Apple Watch" sourceVersion="6.1.3" device="&l
11:07:04 +0100" startDate="2020-03-18 11:07:00 +0100" endDate="2020-03-18 11:07:00 +0100" value="82">
  <MetadataEntry key="HKMetadataKeyHeartRateMotionContext" value="0"/>
 </Record>
 <Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Apple Watch" sourceVersion="6.1.3" device="&l
11:07:13 +0100" startDate="2020-03-18 11:07:12 +0100" endDate="2020-03-18 11:07:12 +0100" value="91">
  <MetadataEntry key="HKMetadataKeyHeartRateMotionContext" value="0"/>
 </Record>
 <Record type="HKQuantityTypeIdentifierHeartRate" sourceName="Apple Watch" sourceVersion="6.1.3" device="&l
11:07:13 +0100" startDate="2020-03-18 11:07:13 +0100" endDate="2020-03-18 11:07:13 +0100" value="89">
  <MetadataEntry key="HKMetadataKeyHeartRateMotionContext" value="0"/>
 </Record>
```

```
rythme = pd.read_xml("export.xml", xpath=".//Record[@type='HKQuantityTypeIdentifierHeartRate']")
```

```
rythme.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 66868 entries, 0 to 66867
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   type            66868 non-null  object 
 1   sourceName      66868 non-null  object 
 2   sourceVersion   66868 non-null  object 
 3   device          66868 non-null  object 
 4   unit            66868 non-null  object 
 5   creationDate    66868 non-null  object 
 6   startDate       66868 non-null  object 
 7   endDate         66868 non-null  object 
 8   value           66868 non-null  float64
 9   MetadataEntry   0 non-null      float64
```

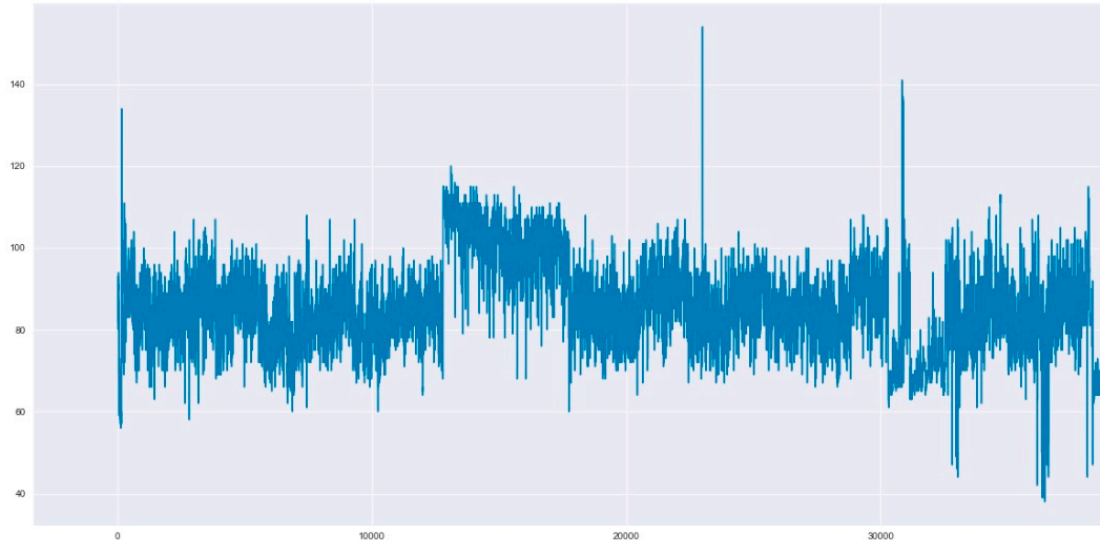
```
rythme.head(10)
```

	type	sourceName	sourceVersion	device	unit	creationDate	startDate	endDate	value
0	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x281669ae0>, name:Apple Watch, ma...	count/min	2020-03-17 09:29:35 +0100	2020-03-17 09:24:38 +0100	2020-03-17 09:24:38 +0100	92.0
1	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x28166bac0>, name:Apple Watch, ma...	count/min	2020-03-17 11:41:52 +0100	2020-03-17 09:27:54 +0100	2020-03-17 09:27:54 +0100	93.0
2	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x2816680a0>, name:Apple Watch, ma...	count/min	2020-03-18 11:06:59 +0100	2020-03-18 11:06:58 +0100	2020-03-18 11:06:58 +0100	82.0
3	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x28166bc00>, name:Apple Watch, ma...	count/min	2020-03-18 11:07:04 +0100	2020-03-18 11:07:00 +0100	2020-03-18 11:07:00 +0100	82.0
4	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x28166bb60>, name:Apple Watch, ma...	count/min	2020-03-18 11:07:13 +0100	2020-03-18 11:07:12 +0100	2020-03-18 11:07:12 +0100	91.0
5	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x281669b30>, name:Apple Watch, ma...	count/min	2020-03-18 11:07:13 +0100	2020-03-18 11:07:13 +0100	2020-03-18 11:07:13 +0100	89.0
6	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x281668b40>, name:Apple Watch, ma...	count/min	2020-03-18 11:07:15 +0100	2020-03-18 11:07:14 +0100	2020-03-18 11:07:14 +0100	89.0
7	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x28166aa80>, name:Apple Watch, ma...	count/min	2020-03-18 11:07:16 +0100	2020-03-18 11:07:15 +0100	2020-03-18 11:07:15 +0100	88.0
8	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x28166a7b0>, name:Apple Watch, ma...	count/min	2020-03-18 11:07:16 +0100	2020-03-18 11:07:16 +0100	2020-03-18 11:07:16 +0100	84.0
9	HKQuantityTypeIdentifierHeartRate	Apple Watch	6.1.3	<<HKDevice: 0x28166a0d0>, name:Apple Watch, ma...	count/min	2020-03-18 11:07:18 +0100	2020-03-18 11:07:17 +0100	2020-03-18 11:07:17 +0100	90.0

```
rythme = rythme[['creationDate','startDate','value']]
```

```
rythme['value'].astype(float).plot(figsize=(35,10))
```

```
<AxesSubplot:>
```



Lecture de fichiers Excel

```
# Returns a DataFrame
pd.read_excel("path_to_file.xls", sheet_name="Sheet1")
```

```
xlsx = pd.ExcelFile(« path_to_file.xls")
with pd.ExcelFile("path_to_file.xls") as xls:
    data["Sheet1"] = pd.read_excel(xls, "Sheet1", index_col=None, na_values=["NA"])
    data["Sheet2"] = pd.read_excel(xls, "Sheet2", index_col=1)
```

```
pd.read_excel("path_to_file.xls", "Sheet1", usecols="A,C:E")
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-excel

Lecture de bases SQL

— Nécessiter d'utiliser des pilotes spécifiques : SQLite (déjà inclus), psycopg2 pour PostgreSQL, pymysql for MySQL, cx_Oracle…

The key functions are:

`read_sql_table`(table_name, con[, schema, ...])	Read SQL database table into a DataFrame.
`read_sql_query`(sql, con[, index_col, ...])	Read SQL query into a DataFrame.
`read_sql`(sql, con[, index_col, ...])	Read SQL query or database table into a DataFrame.
`DataFrame.to_sql`(name, con[, schema, ...])	Write records stored in a DataFrame to a SQL database.

```
from sqlalchemy import create_engine
import pandas as pd
engine = create_engine('dialect://user:pass@host:port/schema', echo=False)
f = pd.read_sql_query('SELECT * FROM mytable', engine, index_col = 'ID')
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-sql

https://stackoverflow.com/questions/10065051/python-pandas-and-databases-like-mysql

Lecture de tables HTML

html5lib

https://pypi.org/project/html5lib/

```
<th style="width:130px;line-height:1.2em;text-align:center">
      <input type="text" name="Q3483" style="width:30px;text-align:right;border-style:inset;
      <br>

      oz 1 NLEA serving
      <br>56g
      <!--
      -->
      </th>

            </thead>
            <tbody>

            <tr class="even" >
            <td style="font-weight:bold" colspan="6" bgcolor="#dddddd" >Proximates</td>
            </tr>


                        <tr class="odd">
                        <td >Water

                        </td>

                        <td style="text-align:center;">g</td>
                        <td style="text-align:right;">51.70</td>


                              <td style="text-align:right;">28.95</td>
```

```
In [295]: url = (
              "https://raw.githubusercontent.com/pandas-dev/pandas/master/"
              "pandas/tests/io/data/html/spam.html"
          )

          dfs = pd.read_html(url)

          dfs
```

	Nutrient	Unit	Value per 100.0g	oz 1 NLEA serving	56g	U
	Proximates	Proximates	Proximates	Proximates	Proximates	P
	Water	g	51.70	28.95		
	Energy	kcal	315	176		
	Protein	g	13.40	7.50		
	Total lipid (fat)	g	26.60	14.90		
	...	...	...	...		
y acids,	total monounsaturated	g	13.505	7.563		
y acids,	total polyunsaturated	g	2.019	1.131		
	Cholesterol	mg	71	40		
	Other	Other	Other	Other		
	Caffeine	mg	0	0		

[37 rows x 6 columns]]

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-read-html

Specify an index column:

```
dfs = pd.read_html(url, index_col=0)
```

Specify a number of rows to skip:

```
dfs = pd.read_html(url, skiprows=0)
```

Specify a number of rows to skip using a list (`range` works as well):

```
dfs = pd.read_html(url, skiprows=range(2))
```

Specify an HTML attribute:

```
dfs1 = pd.read_html(url, attrs={"id": "table"})
dfs2 = pd.read_html(url, attrs={"class": "sortable"})
print(np.array_equal(dfs1[0], dfs2[0]))  # Should be True
```

Specify values that should be converted to NaN:

```
dfs = pd.read_html(url, na_values=["No Acquirer"])
```

Specify whether to keep the default set of NaN values:

```
dfs = pd.read_html(url, keep_default_na=False)
```

Lecture de fichiers JSON

```
In [229]: pd.read_json("test.json")
Out[229]:
                   A         B       date  ints  bools
2013-01-01 -1.294524  0.413738 2013-01-01     0   True
2013-01-02  0.276662 -0.472035 2013-01-01     1   True
2013-01-03 -0.013960 -0.362543 2013-01-01     2   True
2013-01-04 -0.006154 -0.923061 2013-01-01     3   True
2013-01-05  0.895717  0.805244 2013-01-01     4   True
```

```
In [231]: pd.read_json("test.json", dtype={"A": "float32", "bools": "int8"}).dtypes
Out[231]:
A               float32
B               float64
date     datetime64[ns]
ints              int64
bools              int8
dtype: object
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#reading-json

```
In [257]: data = [
   .....:     {"id": 1, "name": {"first": "Coleen", "last": "Volk"}},
   .....:     {"name": {"given": "Mose", "family": "Regner"}},
   .....:     {"id": 2, "name": "Faye Raker"},
   .....: ]
   .....:

In [258]: pd.json_normalize(data)
Out[258]:
    id name.first name.last name.given name.family        name
0  1.0     Coleen      Volk        NaN         NaN         NaN
1  NaN        NaN       NaN       Mose      Regner         NaN
2  2.0        NaN       NaN        NaN         NaN  Faye Raker
```

```
In [259]: data = [
   .....:     {
   .....:         "state": "Florida",
   .....:         "shortname": "FL",
   .....:         "info": {"governor": "Rick Scott"},
   .....:         "county": [
   .....:             {"name": "Dade", "population": 12345},
   .....:             {"name": "Broward", "population": 40000},
   .....:             {"name": "Palm Beach", "population": 60000},
   .....:         ],
   .....:     },
   .....:     {
   .....:         "state": "Ohio",
   .....:         "shortname": "OH",
   .....:         "info": {"governor": "John Kasich"},
   .....:         "county": [
   .....:             {"name": "Summit", "population": 1234},
   .....:             {"name": "Cuyahoga", "population": 1337},
   .....:         ],
   .....:     },
   .....: ]
   .....:

In [260]: pd.json_normalize(data, "county", ["state", "shortname", ["info", "governor"]])
Out[260]:
         name  population    state shortname info.governor
0        Dade       12345  Florida        FL    Rick Scott
1     Broward       40000  Florida        FL    Rick Scott
2  Palm Beach       60000  Florida        FL    Rick Scott
3      Summit        1234     Ohio        OH   John Kasich
4    Cuyahoga        1337     Ohio        OH   John Kasich
```

Aix*Marseille université
Socialement engagée

Transformation des *dataframes*

(Préparation des données)

titanic.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Redéfinir l'index

```
In 187  1  titanic_dummies_sansNA.set_index('PassengerId',inplace=True)
        2  titanic_dummies_sansNA
```

Out 187

PassengerId	Survived	Age	SibSp	Parch	Fare	P
1	0	22.0	1	0	7.2500	
2	1	38.0	1	0	71.2833	
3	1	26.0	0	0	7.9250	
4	1	35.0	1	0	53.1000	
5	0	35.0	0	0	8.0500	
7	0	54.0	0	0	51.8625	
8	0	2.0	3	1	21.0750	
9	1	27.0	0	2	11.1333	
10	1	14.0	1	0	30.0708	
11	1	4.0	1	1	16.7000	
12	1	58.0	0	0	26.5500	
13	0	20.0	0	0	8.0500	
14	0	39.0	1	5	31.2750	
15	0	14.0	0	0	7.8542	
16	1	55.0	0	0	16.0000	
17	0	2.0	4	1	29.1250	
19	0	31.0	1	0	18.0000	
21	0	35.0	0	0	26.0000	
22	1	34.0	0	0	13.0000	
23	1	15.0	0	0	8.0292	

pandas.DataFrame.set_index

`DataFrame.set_index(keys, drop=True, append=False, inplace=False, verify_integrity=False)` [source]

Set the DataFrame index using existing columns.

Set the DataFrame index (row labels) using one or more existing columns or arrays (of the correct length). The index can replace the existing index or expand on it.

Parameters:

keys : *label or array-like or list of labels/arrays*
This parameter can be either a single column key, a single array of the same length as the calling DataFrame, or a list containing an arbitrary combination of column keys and arrays. Here, "array" encompasses `Series`, `Index`, `np.ndarray`, and instances of `Iterator`.

drop : *bool, default True*
Delete columns to be used as the new index.

append : *bool, default False*
Whether to append columns to existing index.

inplace : *bool, default False*
If True, modifies the DataFrame in place (do not create a new object).

verify_integrity : *bool, default False*
Check the new index for duplicates. Otherwise defer the check until necessary. Setting to False will improve the performance of this method.

Returns:

DataFrame or None
Changed row labels or None if `inplace=True`.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  183 non-null    int64
 1   Survived     183 non-null    int64
 2   Pclass       183 non-null    int64
 3   Name         183 non-null    object
 4   Sex          183 non-null    object
 5   Age          183 non-null    float64
 6   SibSp        183 non-null    int64
 7   Parch        183 non-null    int64
 8   Ticket       183 non-null    object
 9   Fare         183 non-null    float64
 10  Cabin        183 non-null    object
 11  Embarked     183 non-null    object
```

```
titanic.dropna(inplace=True)
```

```
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  714 non-null    int64
 1   Survived     714 non-null    int64
 2   Pclass       714 non-null    int64
 3   Name         714 non-null    object
 4   Sex          714 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        714 non-null    int64
 7   Parch        714 non-null    int64
 8   Ticket       714 non-null    object
 9   Fare         714 non-null    float64
 10  Cabin        185 non-null    object
 11  Embarked     712 non-null    object
```

```
titanic.dropna(subset=['Age'],inplace=True)
```

What is the number of passengers in each of the cabin classes?

```
In [12]: titanic["Pclass"].value_counts()
Out[12]:
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

The `value_counts()` method counts the number of records for each category in a column.

The function is a shortcut, as it is actually a groupby operation in combination with counting of the number of records within each group:

```
In [13]: titanic.groupby("Pclass")["Pclass"].count()
Out[13]:
Pclass
1    216
2    184
3    491
Name: Pclass, dtype: int64
```

Suppression des colonnes inutiles

```
In 135  1  titanic.drop(axis=1,columns=['Name','Cabin','Ticket'],inplace=True)
```

```
In 136  1  titanic.head()
```

Out 136

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

5 rows × 10 columns Open in new tab

Arrondir les valeurs décimales

```
In 175  1  titanic['Fare_round'] = titanic['Fare'].round(1)
```

```
In 176  1  titanic.head()
```

Out 176

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Age_cats	Age_quart	Fare_round
0	1	0	3	male	22.0	1	0	7.2500	S	Teens	(19.0, 25.0]	7.2
1	2	1	1	female	38.0	1	0	71.2833	C	Adult	(32.0, 41.0]	71.3
2	3	1	3	female	26.0	0	0	7.9250	S	Teens	(25.0, 32.0]	7.9
3	4	1	1	female	35.0	1	0	53.1000	S	Teens	(32.0, 41.0]	53.1
4	5	0	3	male	35.0	0	0	8.0500	S	Teens	(32.0, 41.0]	8.0

Discrétisation par intervalles donnés (valeurs)

In 137
```
interval = (0,18,35,60,120)
categories = ['Children','Teens','Adult', 'Old']
titanic['Age_cats'] = pd.cut(titanic.Age, interval, labels = categories)
```

In 138
```
titanic.head()
```

Out 138

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Age_cats
0	1	0	3	male	22.0	1	0	7.2500	S	Teens
1	2	1	1	female	38.0	1	0	71.2833	C	Adult
2	3	1	3	female	26.0	0	0	7.9250	S	Teens
3	4	1	1	female	35.0	1	0	53.1000	S	Teens
4	5	0	3	male	35.0	0	0	8.0500	S	Teens

5 rows × 10 columns Open in new tab

Discrétisation équilibrée

```
titanic['Age_quart'] = pd.qcut(titanic.Age, 5, precision=0)
```

```
titanic.head(15)
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Age_cats	Age_quart
0	1	0	3	male	22.0	1	0	7.2500	S	Teens	(19.0, 25.0]
1	2	1	1	female	38.0	1	0	71.2833	C	Adult	(32.0, 41.0]
2	3	1	3	female	26.0	0	0	7.9250	S	Teens	(25.0, 32.0]
3	4	1	1	female	35.0	1	0	53.1000	S	Teens	(32.0, 41.0]
4	5	0	3	male	35.0	0	0	8.0500	S	Teens	(32.0, 41.0]
5	6	0	3	male	NaN	0	0	8.4583	Q	NaN	NaN
6	7	0	1	male	54.0	0	0	51.8625	S	Adult	(41.0, 80.0]
7	8	0	3	male	2.0	3	1	21.0750	S	Children	(-1.0, 19.0]
8	9	1	3	female	27.0	0	2	11.1333	S	Teens	(25.0, 32.0]
9	10	1	2	female	14.0	1	0	30.0708	C	Children	(-1.0, 19.0]
10	11	1	3	female	4.0	1	1	16.7000	S	Children	(-1.0, 19.0]
11	12	1	1	female	58.0	0	0	26.5500	S	Adult	(41.0, 80.0]
12	13	0	3	male	20.0	0	0	8.0500	S	Teens	(19.0, 25.0]
13	14	0	3	male	39.0	1	5	31.2750	S	Adult	(32.0, 41.0]
14	15	0	3	female	14.0	0	0	7.8542	S	Children	(-1.0, 19.0]

```
titanic['Age_quart'].value_counts()
```

	Age_quart
(-1.0, 19.0]	164
(32.0, 41.0]	144
(41.0, 80.0]	142
(19.0, 25.0]	137
(25.0, 32.0]	127

Filtrer les valeurs absentes (dropna)

Les lignes pour lesquelles il manque au moins une valeur OU qui ne contiennent aucune valeur : df.dropna(how='any') vs df.dropna(how='all')

```
titanic.dropna(how="any").info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 183 entries, 1 to 889
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  183 non-null    int64
 1   Survived     183 non-null    int64
 2   Pclass       183 non-null    int64
 3   Name         183 non-null    object
 4   Sex          183 non-null    object
 5   Age          183 non-null    float64
 6   SibSp        183 non-null    int64
 7   Parch        183 non-null    int64
 8   Ticket       183 non-null    object
 9   Fare         183 non-null    float64
 10  Cabin        183 non-null    object
 11  Embarked     183 non-null    object
 12  Age_cats     183 non-null    category
```

```
titanic.dropna(how="all").info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
 12  Age_cats     714 non-null    category
```

Filtrer les valeurs absentes (dropna)

- Les lignes qui contiennent une valeur absente pour une colonne précise

```
titanicSansNA = titanic.dropna(subset=['Pclass'])
titanicSansNA.groupby(titanicSansNA.Pclass).count()
```

Pclass	PassengerId	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_cats
1	216	216	216	216	186	216	216	216	216	176	214	186
2	184	184	184	184	173	184	184	184	184	16	184	173
3	491	491	491	491	355	491	491	491	491	12	491	355

Filtrer les valeurs absentes (dropna)

Les lignes qui ne contiennent pas de valeur pour au moins « une colonne parmi » :

```
titanic.dropna(subset=['Age','Cabin'],how="any").info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 185 entries, 1 to 889
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  185 non-null    int64
 1   Survived     185 non-null    int64
 2   Pclass       185 non-null    int64
 3   Name         185 non-null    object
 4   Sex          185 non-null    object
 5   Age          185 non-null    float64
 6   SibSp        185 non-null    int64
 7   Parch        185 non-null    int64
 8   Ticket       185 non-null    object
 9   Fare         185 non-null    float64
 10  Cabin        185 non-null    object
 11  Embarked     183 non-null    object
 12  Age_cats     185 non-null    category
```

pour « aucune colonne parmi » :

```
titanic.dropna(subset=['Age','Cabin'],how="all").info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 733 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  733 non-null    int64
 1   Survived     733 non-null    int64
 2   Pclass       733 non-null    int64
 3   Name         733 non-null    object
 4   Sex          733 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        733 non-null    int64
 7   Parch        733 non-null    int64
 8   Ticket       733 non-null    object
 9   Fare         733 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     731 non-null    object
 12  Age_cats     714 non-null    category
```

Filtrer les valeurs absentes (dropna)

Les colonnes qui ne sont pas complètes :

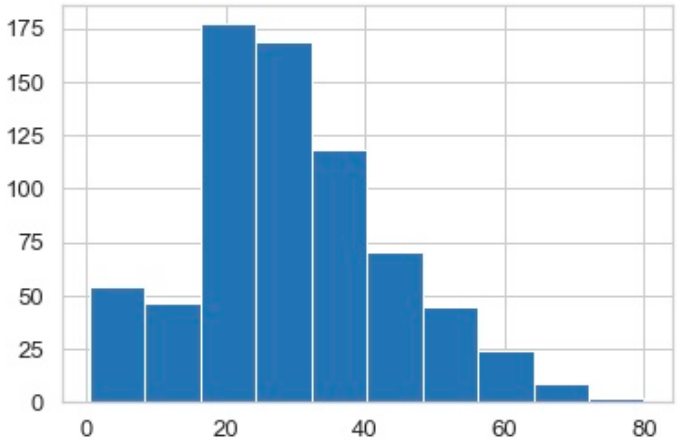
```
titanic.dropna(axis=1).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Ticket       891 non-null    object
 8   Fare         891 non-null    float64
```

Remplacer les valeurs absentes : fillna

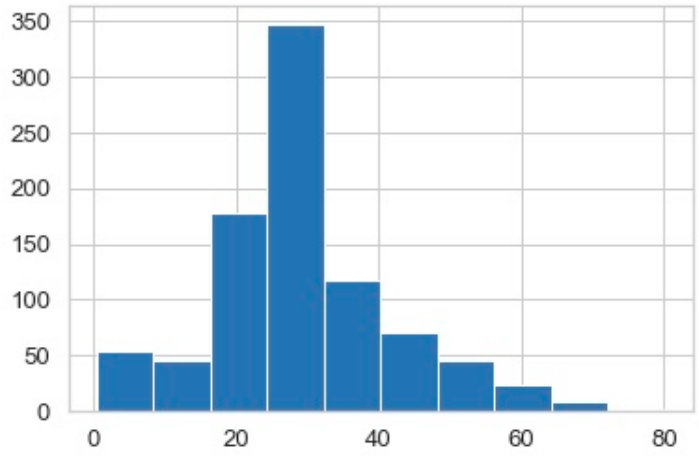
```
titanic.Age.hist()
```

```
<AxesSubplot:>
```




```
titanic.Age.fillna(titanic.Age.mean()).hist()
```

```
<AxesSubplot:>
```



```
>>> df = pd.DataFrame([[np.nan, 2, np.nan, 0],
...                    [3, 4, np.nan, 1],
...                    [np.nan, np.nan, np.nan, np.nan],
...                    [np.nan, 3, np.nan, 4]],
...                   columns=list("ABCD"))
>>> df
     A    B   C    D
0  NaN  2.0 NaN  0.0
1  3.0  4.0 NaN  1.0
2  NaN  NaN NaN  NaN
3  NaN  3.0 NaN  4.0
```

We can also propagate non-null values forward or backward.

```
>>> df.fillna(method="ffill")
     A    B   C    D
0  NaN  2.0 NaN  0.0
1  3.0  4.0 NaN  1.0
2  3.0  4.0 NaN  1.0
3  3.0  3.0 NaN  4.0
```

Replace all NaN elements in column 'A', 'B', 'C', and 'D', with 0, 1, 2, and 3 respectively.

```
>>> values = {"A": 0, "B": 1, "C": 2, "D": 3}
>>> df.fillna(value=values)
     A    B    C    D
0  0.0  2.0  2.0  0.0
1  3.0  4.0  2.0  1.0
2  0.0  1.0  2.0  3.0
3  0.0  3.0  2.0  4.0
```

Remplacer les valeurs d'une colonne

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
1	0	3	male	22.0	1	0	7.250
2	1	1	female	38.0	1	0	71.283
3	1	3	female	26.0	0	0	7.925
4	1	1	female	35.0	1	0	53.100
5	0	3	male	35.0	0	0	8.050
7	0	1	male	54.0	0	0	51.862

```
titanicSansNA['Sex'] = titanicSansNA['Sex'].map({'male':0,'female':1})
```

PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	E
1	0	3	0	22.0	1	0	7.2500	
2	1	1	1	38.0	1	0	71.2833	
3	1	3	1	26.0	0	0	7.9250	
4	1	1	1	35.0	1	0	53.1000	
5	0	3	0	35.0	0	0	8.0500	
7	0	1	0	54.0	0	0	51.8625	
8	0	3	0	2.0	3	1	21.0750	
9	1	3	1	27.0	0	2	11.1333	

Transformer les catégories en nouvelles variables

```
titanic.dtypes
```

	data
PassengerId	int64
Survived	int64
Pclass	int64
Sex	object
Age	float64
SibSp	int64
Parch	int64
Fare	float64
Embarked	object
Age_cats	category

```
In 154  1  titanic_dummies = titanic.copy()
        2  titanic_dummies = titanic_dummies.astype({'Pclass':'category'})
```

```
In 156  1  titanic_dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    category
 3   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
 8   Embarked     889 non-null    object
 9   Age_cats     714 non-null    category
dtypes: category(2), float64(2), int64(4), object(2)
memory usage: 57.9+ KB
```

In 159
```
titanic_dummies = pd.get_dummies(titanic_dummies)
titanic_dummies.head(10)
```

Out 159

	PassengerId	Survived	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Em
0	1	0	22.0	1	0	7.2500	0	0	1	0	1	0	0	
1	2	1	38.0	1	0	71.2833	1	0	0	1	0	1	0	
2	3	1	26.0	0	0	7.9250	0	0	1	1	0	0	0	
3	4	1	35.0	1	0	53.1000	1	0	0	1	0	0	0	
4	5	0	35.0	0	0	8.0500	0	0	1	0	1	0	0	
5	6	0	NaN	0	0	8.4583	0	0	1	0	1	0	1	
6	7	0	54.0	0	0	51.8625	1	0	0	0	1	0	0	
7	8	0	2.0	3	1	21.0750	0	0	1	0	1	0	0	
8	9	1	27.0	0	2	11.1333	0	0	1	1	0	0	0	
9	10	1	14.0	1	0	30.0708	0	1	0	1	0	1	0	

10 rows × 18 columns Open in new tab

In 162
```
titanic_dummies.sum()
```

Out 162

	data
PassengerId	397386.0000
Survived	342.0000
Age	21205.1700
SibSp	466.0000
Parch	340.0000
Fare	28693.9493
Pclass_1	216.0000
Pclass_2	184.0000
Pclass_3	491.0000
Sex_female	314.0000
Sex_male	577.0000
Embarked_C	168.0000
Embarked_Q	77.0000
Embarked_S	644.0000
Age_cats_Children	139.0000
Age_cats_Teens	358.0000
Age_cats_Adult	195.0000
Age_cats_Old	22.0000

Transformer les catégories en nouvelles variables

```
pd.get_dummies(titanic)
```

	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Age_cats_Children	Age_cats_Teens	Age_cats_Adult
	3	22.0	1	0	7.2500	0	1	0	0	1	0	1	0
	1	38.0	1	0	71.2833	1	0	1	0	0	0	0	1
	3	26.0	0	0	7.9250	1	0	0	0	1	0	1	0
	1	35.0	1	0	53.1000	1	0	0	0	1	0	1	0
	3	35.0	0	0	8.0500	0	1	0	0	1	0	1	0
	3	NaN	0	0	8.4583	0	1	0	1	0	0	0	0
	1	54.0	0	0	51.8625	0	1	0	0	1	0	0	1
	3	2.0	3	1	21.0750	0	1	0	0	1	1	0	0
	3	27.0	0	2	11.1333	1	0	0	0	1	0	1	0
	2	14.0	1	0	30.0708	1	0	1	0	0	1	0	0
	3	4.0	1	1	16.7000	1	0	0	0	1	1	0	0
	1	58.0	0	0	26.5500	1	0	0	0	1	0	0	1
	3	20.0	0	0	8.0500	0	1	0	0	1	0	1	0
	3	39.0	1	5	31.2750	0	1	0	0	1	0	0	1
	3	14.0	0	0	7.8542	1	0	0	0	1	1	0	0

Filtrer les valeurs absentes (dropna)

- Les lignes qui contiennent au moins une valeur absente : df.dropna()

Exemple : *quel impact sur les données Titanic si on les groupe par classe ?*

```
1 titanic.dropna().groupby(titanic.Pclass).count()
```

Pclass	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_cats
1	158	158	158	158	158	158	158	158	158	158	158	158	158
2	15	15	15	15	15	15	15	15	15	15	15	15	15
3	10	10	10	10	10	10	10	10	10	10	10	10	10

3 rows × 13 columns Open in new tab

```
1 titanic.groupby(titanic.Pclass).count()
```

Pclass	PassengerId	Survived	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Age_cats
1	216	216	216	216	186	216	216	216	216	176	214	186
2	184	184	184	184	173	184	184	184	184	16	184	173
3	491	491	491	491	355	491	491	491	491	12	491	355

Suppression des lignes dupliquées

```
df = pd.DataFrame( {'couleur':
['blanc','bleu','rouge','rouge','blanc'],
'valeur':[10,4,3,3,2]})
```

	couleur	valeur
0	blanc	10
1	bleu	4
2	rouge	3
3	rouge	3
4	blanc	2

df.duplicated()

```
df.duplicated()
```

```
0    False
1    False
2    False
3     True
4    False
```

df[df.duplicated()]

```
df[df.duplicated()]
```

	couleur	valeur
3	rouge	3

df.drop_duplicates()

```
df.drop_duplicates()
df
```

	couleur	valeur
0	blanc	10
1	bleu	4
2	rouge	3
3	rouge	3
4	blanc	2

df.drop_duplicates(inplace=True)

	couleur	valeur
0	blanc	10
1	bleu	4
2	rouge	3
4	blanc	2

Consider dataset containing ramen rating.

```
>>> df = pd.DataFrame({
...     'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
...     'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
...     'rating': [4, 4, 3.5, 15, 5]
... })
>>> df
    brand style  rating
0  Yum Yum   cup     4.0
1  Yum Yum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

By default, it removes duplicate rows based on all columns.

```
>>> df.drop_duplicates()
    brand style  rating
0  Yum Yum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

To remove duplicates on specific column(s), use `subset`.

```
>>> df.drop_duplicates(subset=['brand'])
    brand style  rating
0  Yum Yum   cup     4.0
2  Indomie   cup     3.5
```

To remove duplicates and keep last occurrences, use `keep`.

```
>>> df.drop_duplicates(subset=['brand', 'style'], keep='last')
    brand style  rating
1  Yum Yum   cup     4.0
2  Indomie   cup     3.5
4  Indomie  pack     5.0
```

Homogénéiser des valeurs (mapping)

```
>>> frame = pd.DataFrame({ 'item':['ball','mug','pen','pencil','ashtray'],
...                        'color':['white','rosso','verde','black','yellow'],
                           'price':[5.56,4.20,1.30,0.56,2.75]})
>>> frame
    color     item     price
0   white     ball      5.56
1   rosso      mug      4.20
2   verde      pen      1.30
3   black   pencil      0.56
4  yellow  ashtray      2.75
```

```
>>> newcolors = {
...     'rosso': 'red',
...     'verde': 'green'
... }
```

```
>>> frame.replace(newcolors)
     color      item  price
0    white      ball   5.56
1      red       mug   4.20
2    green       pen   1.30
3    black   pencil   0.56
4   yellow  ashtray   2.75
```

https://learning.oreilly.com/library/view/python-data-analytics/9781484239131/html/336498_2_En_6_Chapter.xhtml

Réorganisation et multi-index

	c1	c2	c3
Marseille	0	1	2
Aix	3	4	5
Gap	6	7	8

```
df = pd.DataFrame(np.arange(9).reshape(3,3), index =
['Marseille', 'Aix', 'Gap'], columns = ['c1','c2','c3'])
```

df['c3'].loc['Gap'] ou df['c3']['Gap'] ou df.loc['Gap','c3']

mais pas df['Gap']['c3'] et pas df.loc['c3','Gap']

```
dh = df.stack()
```

```
Marseille  c1    0
           c2    1
           c3    2
Aix        c1    3
           c2    4
           c3    5
Gap        c1    6
           c2    7
           c3    8
```

Serie

```
dg = df.unstack()
```

```
c1  Marseille    0
    Aix          3
    Gap          6
c2  Marseille    1
    Aix          4
    Gap          7
c3  Marseille    2
    Aix          5
    Gap          8
```

dg['c1','Gap']

Serie

```
dg['c1']
```

```
Marseille    0
Aix          3
Gap          6
```

```
dg.index
```

```
MultiIndex([('c1', 'Marseille'),
            ('c1',       'Aix'),
            ('c1',       'Gap'),
            ('c2', 'Marseille'),
            ('c2',       'Aix'),
            ('c2',       'Gap'),
            ('c3', 'Marseille'),
            ('c3',       'Aix'),
            ('c3',       'Gap')],
           )
```

Pivoter lignes et colonnes selon un « pivot »

Reshaping by pivoting DataFrame objects

Pivot

df

clé primaire *clé 2è* *valeurs*

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

```
df.pivot(index='foo',
         columns='bar',
         values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

clé 2è

valeurs

clé primaire

```
Première date :  2020-03-19 00:00:00
Fin :  2021-02-08 00:00:00
```

	nomReg	numReg	incid_rea
jour			
2020-03-19	Auvergne-Rhône-Alpes	84	44
2020-03-19	Bourgogne-Franche-Comté	27	33
2020-03-19	Bretagne	53	8
2020-03-19	Centre-Val de Loire	24	6
2020-03-19	Corse	94	11

```
newDF = incidence.pivot(index='nomReg', columns='jour', values='incid_rea')
```

jour	2020-03-19	2020-03-20	2020-03-21	2020-03-22	2020-03-23	2020-03-24	2020-03-25	2020-03-26	2020-03-27	2020-03-28	...
nomReg											
Auvergne-Rhône-Alpes	44	16	15	25	45	47	85	64	77	108	...
Bourgogne-Franche-Comté	33	9	11	14	12	19	30	19	24	21	...
Bretagne	8	2	9	8	6	8	5	10	9	19	...
Centre-Val de Loire	6	4	3	7	17	7	19	15	18	13	...
Corse	11	0	0	2	2	0	0	4	1	6	...

Permutation aléatoire des lignes

```
df = pd.DataFrame(np.arange(25).reshape(5,5))
```

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
4	20	21	22	23	24

```
ordre = np.random.permutation(5)
dg = df.take(ordre)
```

	0	1	2	3	4
3	15	16	17	18	19
1	5	6	7	8	9
4	20	21	22	23	24
0	0	1	2	3	4
2	10	11	12	13	14

	c1	c2	c3
Aix	3	4	5
Marseille	0	1	2
Gap	6	7	8

Echantillonnage aléatoire

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
4	20	21	22	23	24

numpy.random.randint

numpy.random.randint(*low*, *high=None*, *size=None*)

Return random integers from *low* (inclusive) to *high* (exclusive).

```
echantillon = np.random.randint(0,len(df),size=2)
df.take(echantillon)
```

5

	0	1	2	3	4
3	15	16	17	18	19
4	20	21	22	23	24

Agrégation de données

pandas.DataFrame.groupby

DataFrame.**groupby**(*by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=<object object>, observed=False, dropna=True*) [source]

Group DataFrame using a mapper or by a Series of columns.

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon', 'Parrot',
'Parrot'], 'Max Speed': [380., 370., 24., 26.]})
```

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

```
df.groupby(['Animal']).mean()
```

	Max Speed
Animal	
Falcon	375.0
Parrot	25.0

```
df.groupby(['Animal']).mean().loc['Falcon']
```

```
df = pd.DataFrame({'Animal': ['Falcon',
'Falcon', 'Parrot', 'Parrot'], 'Max Speed':
[380., 370., 24., 26.], 'Weight' : [10, 40, 3,
200]})
```

	Animal	Max Speed	Weight
0	Falcon	380.0	10
1	Falcon	370.0	40
2	Parrot	24.0	3
3	Parrot	26.0	200

```
dg = df.groupby(['Animal']).mean()
```

	Max Speed	Weight
Animal		
Falcon	375.0	25.0
Parrot	25.0	101.5

```
dg = df['Weight'].groupby(df['Animal']).mean()
```

```
Animal
Falcon    25.0
Parrot    101.5
```

```
>>> arrays = [['Falcon', 'Falcon', 'Parrot', 'Parrot'],
...           ['Captive', 'Wild', 'Captive', 'Wild']]
>>> index = pd.MultiIndex.from_arrays(arrays, names=('Animal', 'Type'))
>>> df = pd.DataFrame({'Max Speed': [390., 350., 30., 20.]},
...                   index=index)
>>> df
                Max Speed
Animal Type
Falcon Captive      390.0
       Wild         350.0
Parrot Captive       30.0
       Wild          20.0
>>> df.groupby(level=0).mean()
        Max Speed
Animal
Falcon      370.0
Parrot       25.0
>>> df.groupby(level="Type").mean()
         Max Speed
Type
Captive      210.0
Wild         185.0
```

Aix*Marseille université
Socialement engagée

Dataframes : tri et fusion

Trier (ordonner) les valeurs

pandas.DataFrame.sort_values

`DataFrame.sort_values`(*by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last', ignore_index=False, key=None*) [source]

Sort by the values along either axis.

Parameters: **by** : *str or list of str*

Name or list of names to sort by.

- if *axis* is 0 or *'index'* then *by* may contain index levels and/or column labels.
- if *axis* is 1 or *'columns'* then *by* may contain column levels and/or index labels.

axis : *{0 or 'index', 1 or 'columns'}, default 0*

Axis to be sorted.

ascending : *bool or list of bool, default True*

Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.

inplace : *bool, default False*

If True, perform operation in-place.

inplace : *bool, default False*

If True, perform operation in-place.

kind : *{'quicksort', 'mergesort', 'heapsort'}, default 'quicksort'*

Choice of sorting algorithm. See also ndarray.np.sort for more information. *mergesort* is the only stable algorithm. For DataFrames, this option is only applied when sorting on a single column or label.

na_position : *{'first', 'last'}, default 'last'*

Puts NaNs at the beginning if *first*; *last* puts NaNs at the end.

ignore_index : *bool, default False*

If True, the resulting axis will be labeled 0, 1, …, n - 1.
New in version 1.0.0.

key : *callable, optional*

Apply the key function to the values before sorting. This is similar to the *key* argument in the builtin `sorted()` function, with the notable difference that this *key* function should be *vectorized*. It should expect a `Series` and return a Series with the same shape as the input. It will be applied to each column in *by* independently.
New in version 1.1.0.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html

	col1	col2	col3	col4
0	A	2	0	a
1	A	1	1	B
2	B	9	9	c
3	NaN	8	4	D
4	D	7	2	e
5	C	4	3	F

Sort by col1

```
>>> df.sort_values(by=['col1'])
  col1  col2  col3 col4
0    A     2     0    a
1    A     1     1    B
2    B     9     9    c
5    C     4     3    F
4    D     7     2    e
3  NaN     8     4    D
```

Sort by multiple columns

```
>>> df.sort_values(by=['col1', 'col2'])
  col1  col2  col3 col4
1    A     1     1    B
0    A     2     0    a
2    B     9     9    c
5    C     4     3    F
4    D     7     2    e
3  NaN     8     4    D
```

Sort Descending

```
>>> df.sort_values(by='col1', ascending=False)
  col1  col2  col3 col4
4    D     7     2    e
5    C     4     3    F
2    B     9     9    c
0    A     2     0    a
1    A     1     1    B
3  NaN     8     4    D
```

Putting NAs first

```
>>> df.sort_values(by='col1', ascending=False, na_position='first')
  col1  col2  col3 col4
3  NaN     8     4    D
4    D     7     2    e
5    C     4     3    F
2    B     9     9    c
0    A     2     0    a
1    A     1     1    B
```

Sorting with a key function

```
>>> df.sort_values(by='col4', key=lambda col: col.str.lower())
   col1  col2  col3 col4
0    A     2     0    a
1    A     1     1    B
2    B     9     9    c
3  NaN     8     4    D
4    D     7     2    e
5    C     4     3    F
```

sort_index

```
>>> df = pd.DataFrame([1, 2, 3, 4, 5], index=[100, 29, 234, 1, 150],
...                   columns=['A'])
>>> df.sort_index()
     A
1    4
29   2
100  1
150  5
234  3
```

By default, it sorts in ascending order, to sort in descending order, use `ascending=False`

```
>>> df.sort_index(ascending=False)
     A
234  3
150  5
100  1
29   2
1    4
```

Addition de (valeurs de) 2 data frames

```
f1 = pd.DataFrame(np.arange(9).reshape((3,3)),
index=['a','b','c'], columns=['c1','c2','c3'])

f2 = pd.DataFrame(np.arange(8).reshape((2,4)),
index=['a','d'], columns=['c4','c2','c5','c6'])
```

f1

	c1	c2	c3
a	0	1	2
b	3	4	5
c	6	7	8

f2

	c4	c2	c5	c6
a	0	1	2	3
d	4	5	6	7

```
f1+f2

f1.add(f2)
```

f1+f2

	c1	c2	c3	c4	c5	c6
a	NaN	2.0	NaN	NaN	NaN	NaN
b	NaN	NaN	NaN	NaN	NaN	NaN
c	NaN	NaN	NaN	NaN	NaN	NaN
d	NaN	NaN	NaN	NaN	NaN	NaN

Combiner 2 data frames

```
pd.concat([f1,f2])
pd.concat([f1,f2], axis=0)
f1.append(f2)
```

```
pd.concat(
    objs,
    axis=0,
    join="outer",
    ignore_index=False,
    keys=None,
    levels=None,
    names=None,
    verify_integrity=False,
    copy=True,
)
```

f1

	c1	c2	c3
a	0	1	2
b	3	4	5
c	6	7	8

f2

	c4	c2	c5	c6
a	0	1	2	3
d	4	5	6	7

```
pd.concat([f1,f2], axis=1)
```

pd.concat([f1,f2])

	c1	c2	c3	c4	c5	c6
a	0.0	1	2.0	NaN	NaN	NaN
b	3.0	4	5.0	NaN	NaN	NaN
c	6.0	7	8.0	NaN	NaN	NaN
a	NaN	1	NaN	0.0	2.0	3.0
d	NaN	5	NaN	4.0	6.0	7.0

pd.concat([f1,f2], axis=1)

	c1	c2	c3	c4	c2	c5	c6
a	0.0	1.0	2.0	0.0	1.0	2.0	3.0
b	3.0	4.0	5.0	NaN	NaN	NaN	NaN
c	6.0	7.0	8.0	NaN	NaN	NaN	NaN
d	NaN	NaN	NaN	4.0	5.0	6.0	7.0

https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

```
pd.concat([f1,f2], axis=0, join='inner')
```

fusion par ligne avec intersection des colonnes

```
pd.concat([f1,f2], axis=0, join='inner')
```

	c2
a	1
b	4
c	7
a	1
d	5

```
pd.concat([f1,f2], axis=1, join='inner')
```

fusion par colonne avec intersection des lignes

```
pd.concat([f1,f2], axis=1, join='inner')
```

	c1	c2	c3	c4	c2	c5	c6
a	0	1	2	0	1	2	3

```
In [1]: df1 = pd.DataFrame(
   ...:     {
   ...:         "A": ["A0", "A1", "A2", "A3"],
   ...:         "B": ["B0", "B1", "B2", "B3"],
   ...:         "C": ["C0", "C1", "C2", "C3"],
   ...:         "D": ["D0", "D1", "D2", "D3"],
   ...:     },
   ...:     index=[0, 1, 2, 3],
   ...: )
   ...:

In [2]: df2 = pd.DataFrame(
   ...:     {
   ...:         "A": ["A4", "A5", "A6", "A7"],
   ...:         "B": ["B4", "B5", "B6", "B7"],
   ...:         "C": ["C4", "C5", "C6", "C7"],
   ...:         "D": ["D4", "D5", "D6", "D7"],
   ...:     },
   ...:     index=[4, 5, 6, 7],
   ...: )
   ...:

In [3]: df3 = pd.DataFrame(
   ...:     {
   ...:         "A": ["A8", "A9", "A10", "A11"],
   ...:         "B": ["B8", "B9", "B10", "B11"],
   ...:         "C": ["C8", "C9", "C10", "C11"],
   ...:         "D": ["D8", "D9", "D10", "D11"],
   ...:     },
   ...:     index=[8, 9, 10, 11],
   ...: )
   ...:

In [4]: frames = [df1, df2, df3]

In [5]: result = pd.concat(frames)
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
result = pd.concat(frames, keys=["x", "y", "z"])
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

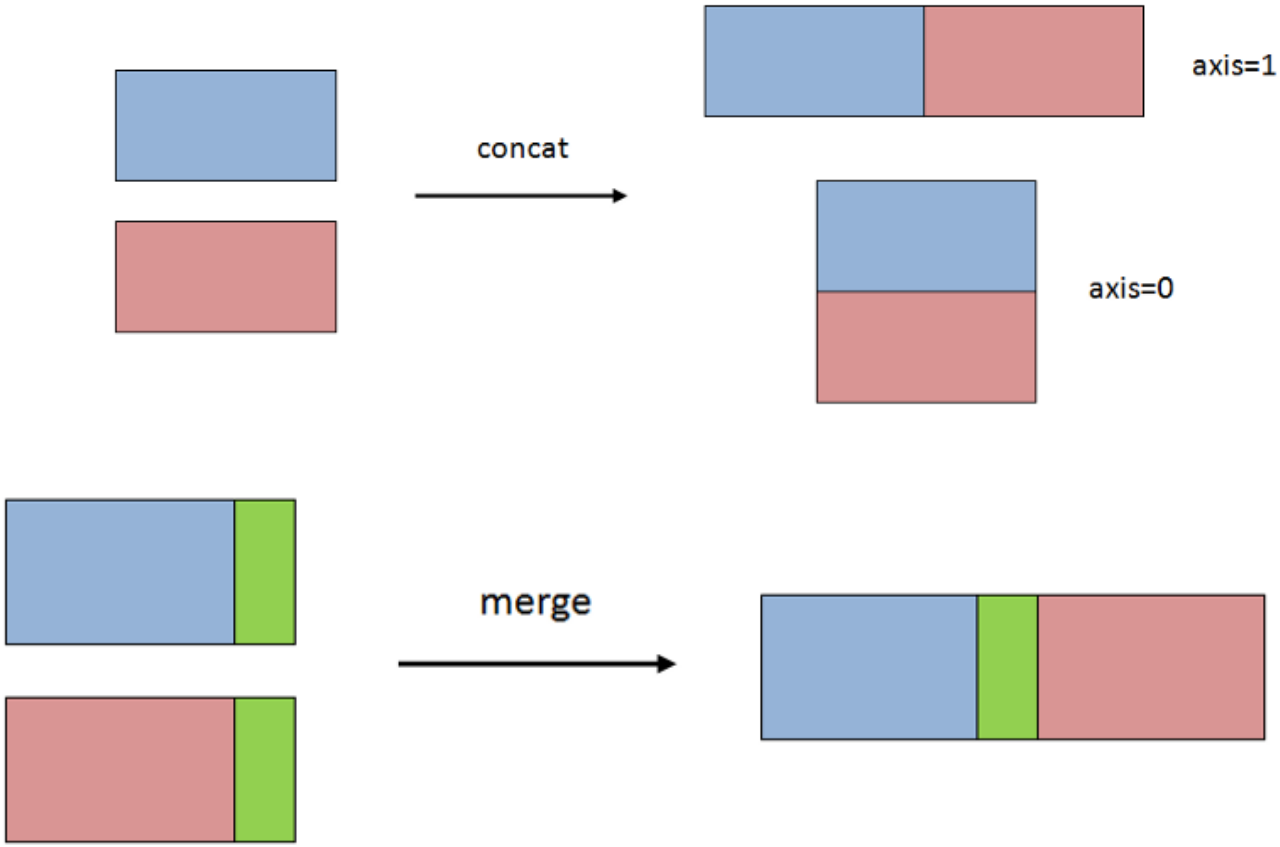
	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

		A	B	C	D
x	0	A0	B0	C0	D0
x	1	A1	B1	C1	D1
x	2	A2	B2	C2	D2
x	3	A3	B3	C3	D3
y	4	A4	B4	C4	D4
y	5	A5	B5	C5	D5
y	6	A6	B6	C6	D6
y	7	A7	B7	C7	D7
z	8	A8	B8	C8	D8
z	9	A9	B9	C9	D9
z	10	A10	B10	C10	D10
z	11	A11	B11	C11	D11

```
In [7]: result.loc["y"]
Out[7]:
    A   B   C   D
4  A4  B4  C4  D4
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7
```

```
frames = [ process_your_file(f) for f in files ]
result = pd.concat(frames)
```



https://towardsdatascience.com/3-key-differences-between-merge-and-concat-functions-of-pandas-ab2bab224b59

```
In [84]: left = pd.DataFrame(
   ....:     {"A": ["A0", "A1", "A2"], "B": ["B0", "B1", "B2"]}, index=["K0", "K1", "K2"]
   ....: )
   ....:

In [85]: right = pd.DataFrame(
   ....:     {"C": ["C0", "C2", "C3"], "D": ["D0", "D2", "D3"]}, index=["K0", "K2", "K3"]
   ....: )
   ....:

In [86]: result = left.join(right)
```

left

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

right

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

Result

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

```
result = left.join(right)
```

Result

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

```
result = left.join(right, how="outer")
```

Result

	A	B	C	D
K0	A0	B0	C0	D0
K2	A2	B2	C2	D2

```
result = left.join(right, how="inner")
```

voir https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

Rajouter une ou des lignes

```
>>> df = pd.DataFrame([[1, 2], [3, 4]], columns=list('AB'))
>>> df
   A  B
0  1  2
1  3  4
>>> df2 = pd.DataFrame([[5, 6], [7, 8]], columns=list('AB'))
>>> df.append(df2)
   A  B
0  1  2
1  3  4
0  5  6
1  7  8
```

With *ignore_index* set to True:

```
>>> df.append(df2, ignore_index=True)
   A  B
0  1  2
1  3  4
2  5  6
3  7  8
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.append.html

Appliquer une fonction sur une colonne (*apply*)

```
def nom_fonction (element):
        …
        return …
```

```
df[‘colonne_nouvelle’] = df[‘colonne’].apply(nom_fonction)
```