Aix*Marseille université
Socialement engagée

Pandas

DataFrames, analyse descriptive, données temporelles, données textuelles…

Patrice Bellot
patrice.bellot@univ-amu.fr

février 2025

Introduction

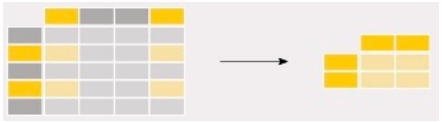
— Pandas a pour objectif de fournir les structures de données et les méthodes pour le traitement des données, leur extraction, leur manipulation

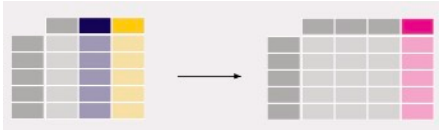
— Les données peuvent provenir de différentes sources et formats : CSV, JSON, XML, Excel, SQL… = **des données structurées**

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

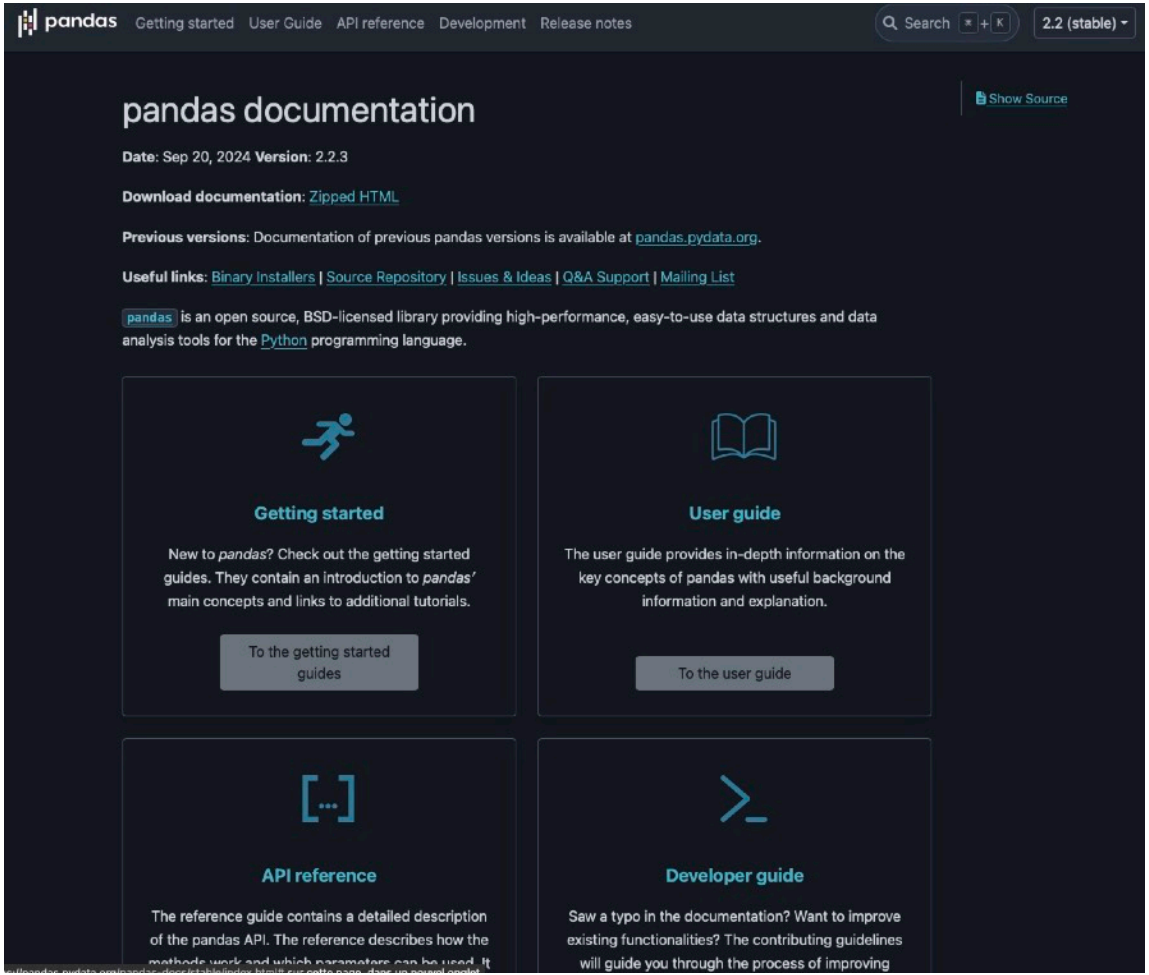
— NumPy est en quelque sorte la base de Pandas (beaucoup d'opérations sur les *arrays* NumPy s'appliquent aux *Series* Pandas, les *dataframes* peuvent souvent être utilisées comme des matrices)

pandas





séries temporelles
textes



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

pandas.DataFrame.drop

DataFrame.drop(*labels=None, *, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise'*) [source]

Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding axis, or by directly specifying index or column names. When using a multi-index, labels on different levels can be removed by specifying the level. See the user guide for more information about the now unused levels.

Parameters:

labels : *single label or list-like*

Index or column labels to drop. A tuple will be used as a single label and not treated as a list-like.

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

Whether to drop labels from the index (0 or 'index') or columns (1 or 'columns').

index : *single label or list-like*

Alternative to specifying axis (`labels, axis=0` is equivalent to `index=labels`).

columns : *single label or list-like*

Alternative to specifying axis (`labels, axis=1` is equivalent to `columns=labels`).

level : *int or level name, optional*

For MultiIndex, level from which the labels will be removed.

inplace : *bool, default False*

If False, return a copy. Otherwise, do operation in place and return None.

errors : *{'ignore', 'raise'}, default 'raise'*

If 'ignore', suppress error and only existing labels are dropped.

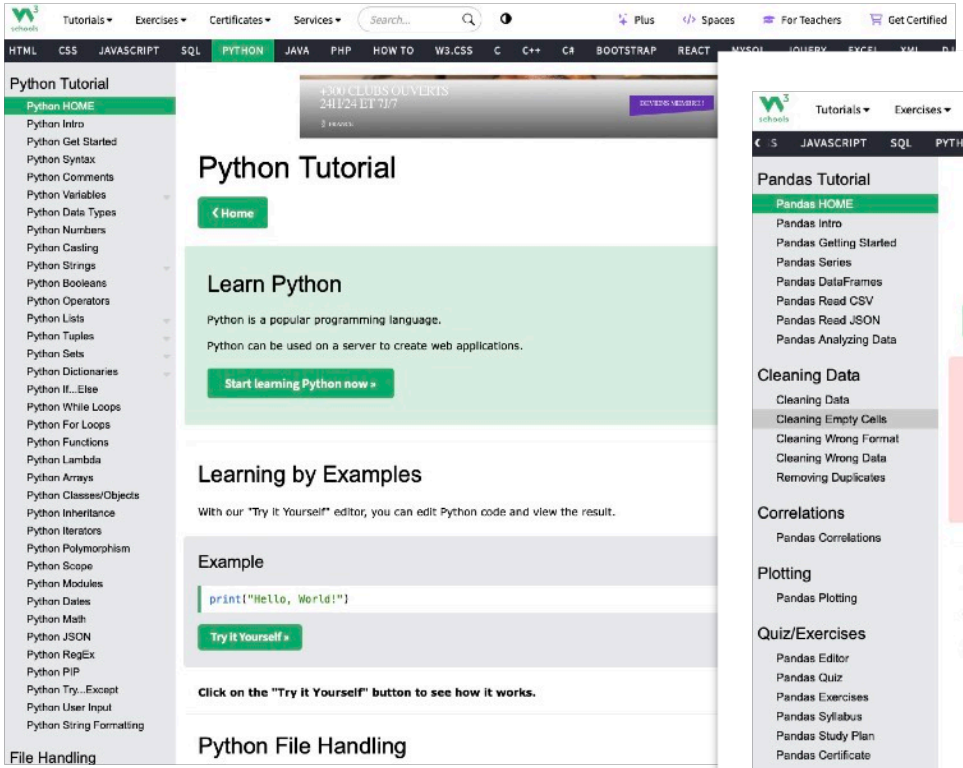
après l'* tous les arguments doivent être appelés avec leur nom

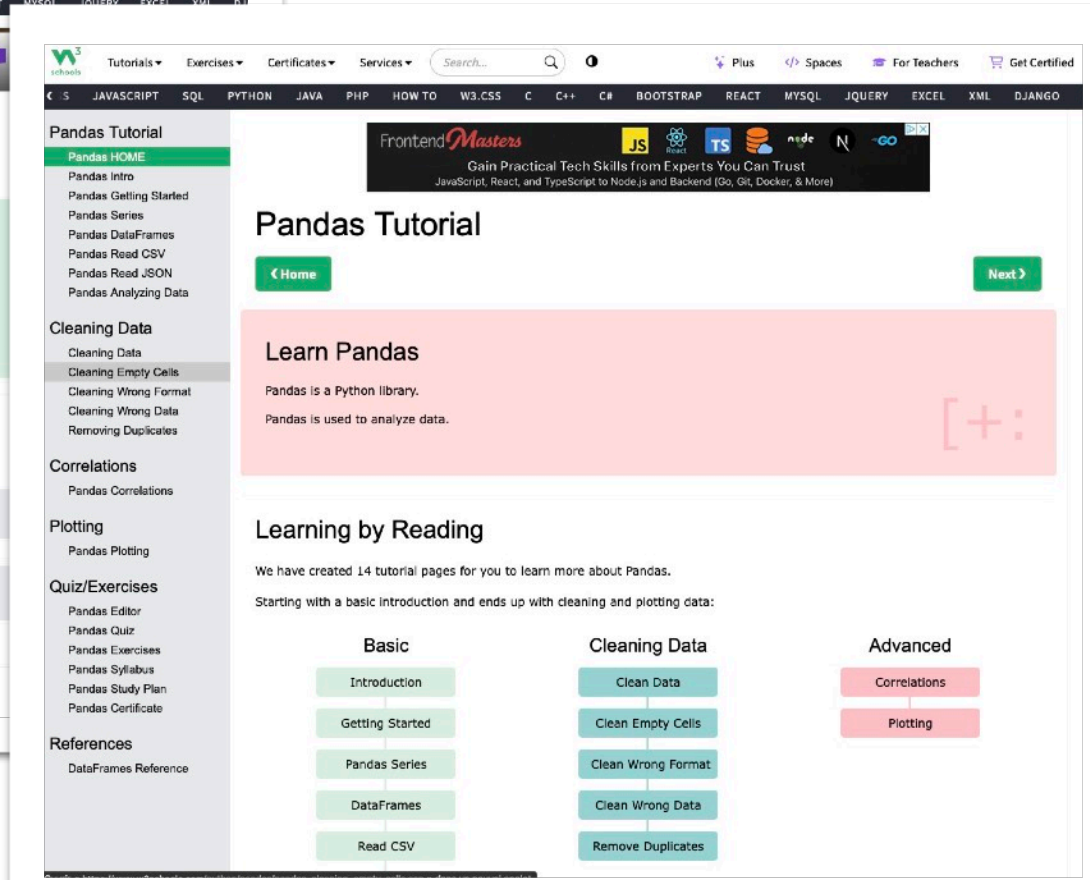
```
df.drop('A', axis=1)
```

équivalent de :

```
df.drop(labels='A', axis=1)
```

https://www.w3schools.com/python





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

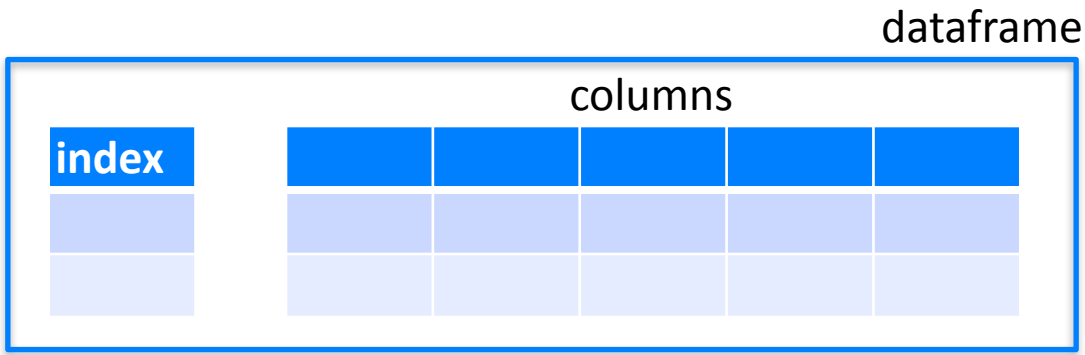
Data Frames et *Series*
accès aux données

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])
```

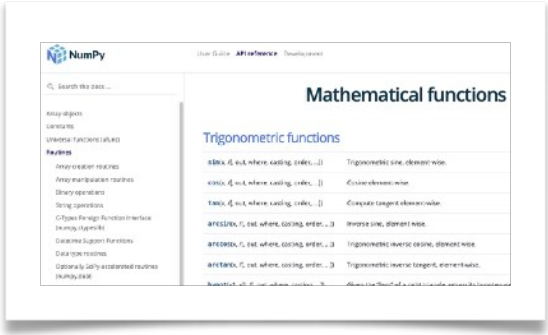
```
33 s = pd.Series([4,1,10,15])
   print(s[s>1])
   print(s[s==4])

   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
```

Exemple dans VsCode avec « Data Wrangler »

```
import pandas as pd
import numpy as np
```
[12] ✓ 0.0s

```
data = { 'col1': [1, 2, 3, 4, 7, 11], 'col2': ['a','b',np.nan,'d', 'e', 140], 'col3': [7, 5, 8, 12, 1,11] }

df = pd.DataFrame(data)
```
[14] ✓ 0.0s

```
df
```
[15] ✓ 0.0s Ouvrir « df » dans Data Wrangler

	col1	col2	col3
0	1	a	7
1	2	b	5
2	3	NaN	8
3	4	d	12
4	7	e	1
5	11	140	11

Untitled-1.ipynb ● df(2) [DW] ×

Exporter en tant que fichier | Actualiser des données | Signaler un problème | 6 lignes x

	# col1	col2	# col3
	Manquant: 0 (0%) Distinct: 6 (100%)	Manquant: 1 (17%) Distinct: 5 (83%)	Manquant: 0 (0%) Distinct: 6 (100%)
	Min 1 … Max 11	a 17% b 17% d 17% Autre 33%	Min 1 … Max 12
0	1	a	7
1	2	b	5
2	3	*Valeur manquante*	8
3	4	d	12
4	7	e	1
5	11	140	11

Dataframes : noms de lignes et de colonnes

```
{'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
```

Exemple matrice NumPy vers DataFrame

```
a = np.asmatrix('1, 2, 4; 3, 4, 5')
```

✓ 0.0s

```
a
```

✓ 0.0s

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

```
df = pd.DataFrame(a, columns=['col1','col2','col3'], index=['item1','item2'])
df
```

✓ 0.0s Ouvrir « df » dans Data Wrangler

	col1	col2	col3
item1	1	2	4
item2	3	4	5

```
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

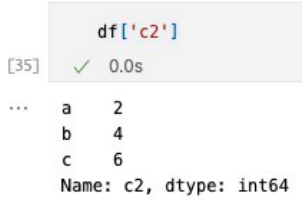
```
86 df.columns
86 Index(['c1', 'c2'], dtype='object')

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
```

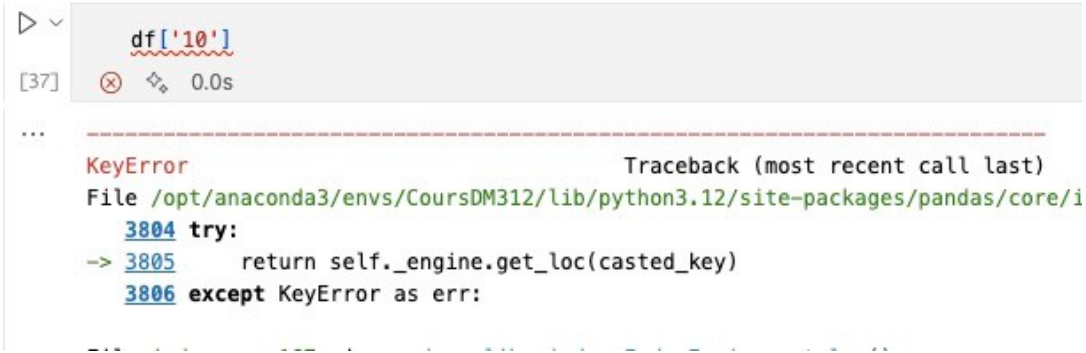
Comment accéder aux colonnes ?

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

	10	c2
a	1	2
b	3	4
c	5	6

`df['c2']`





`df[10]`

`df[[10,'c2']]`

Comment accéder aux lignes ?

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

	10	c2
a	1	2
b	3	4
c	5	6

```
df.loc['a']
```

```
In 192  1  s=df.loc['a']
        2  type(s)

Out 192    pandas.core.series.Series
```

```
10    1
c2    2
Name: a, dtype: int64
```

```
df.loc[['a','c']]
```

	10	c2
a	1	2
c	5	6

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

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

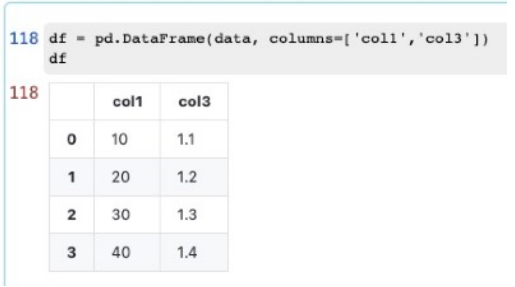
Colab

```
df.loc['b','c2']
[39]  ✓ 0.0s
...   np.int64(4)

print(df['c2']['b'])
[41]  ✓ 0.0s
...   4
```

Vs Code

```
data= {'col1': [1, 2, 3, 4], 'col2': ['a','b','c','d'], 'col3': [1.1, 1.2, 1.3, 1.4]}

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

```
df
```



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

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

121 `df.loc[1]`

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

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

le rang 0 et le rang 2

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

123

	col1	col3
0	10	1.1
2	30	1.3

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

du rang 0 au rang 2 inclus

128 `df.loc[0:2]`

128

	col1	col3
0	10	1.1
1	20	1.2
2	30	1.3

!

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

du rang 0 au rang 2 NON inclus

129 `df[0:2]`

129

	col1	col3
0	10	1.1
1	20	1.2

```
df = pd.DataFrame({0:[0,1,2,3], 1:[-1,0,1,0], 2:[-2,-1,0,1], 3:[4,5,6,7]}, index=['a','b','w','d'])
```

```
df
```
[81] ✓ 0.0s Ouvrir « df »

	0	1	2	3
a	0	-1	-2	4
b	1	0	-1	5
w	2	1	0	6
d	3	0	1	7

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

```
df.loc['b':'d']
```
[82] ✓ 0.0s

	0	1	2	3
b	1	0	-1	5
w	2	1	0	6
d	3	0	1	7

`df['b':'d']`

```
df['b':'d']
```
[83] ✓ 0.0s

	0	1	2	3
b	1	0	-1	5
w	2	1	0	6
d	3	0	1	7

```
df = pd.DataFrame(data={0:[0,1,2,3], 1:[-1,0,1,0], 2:[-2,-1,0,1], 3:[4,5,6,7]})
```

	0	1	2	3
0	0	-1	-2	4
1	1	0	-1	5
2	2	1	0	6
3	3	0	1	7

`df.loc[2]`

```
[93] df.loc[2]
     ✓ 0.0s
0    2
1    1
2    0
3    6
Name: 2, dtype: int64
```

`df[2]`

```
[94] df[2]
     ✓ 0.0s
0   -2
1   -1
2    0
3    1
Name: 2, dtype: int64
```

`df.loc[:,1:3]`

```
[95] df.loc[:,1:3]
     ✓ 0.0s
```

	1	2	3
0	-1	-2	4
1	0	-1	5
2	1	0	6
3	0	1	7

`df.loc[1:2,2:3]`

```
[97] df.loc[1:2,2:3]
     ✓ 0.0s
```

	2	3
1	-1	5
2	0	6

Attention aux types…

```
df = pd.DataFrame(data={0:[0,1,2,3], 1:[-1,0,1,0], '2':[-2,-1,0,1],'3':[4,5,6,7]})
```

	0	1	2	3
0	0	-1	-2	4
1	1	0	-1	5
2	2	1	0	6
3	3	0	1	7

Accéder à la colonne *3* ?

```
df[3]
```

```
Traceback…
KeyError: 3
```

```
df['3']
```

```
0    4
1    5
2    6
3    7
Name: 3, dtype: int64
```

```
df.loc[3]
```

```
0    3
1    0
2    1
3    7
Name: 3, dtype: int64
```

Accès par saut

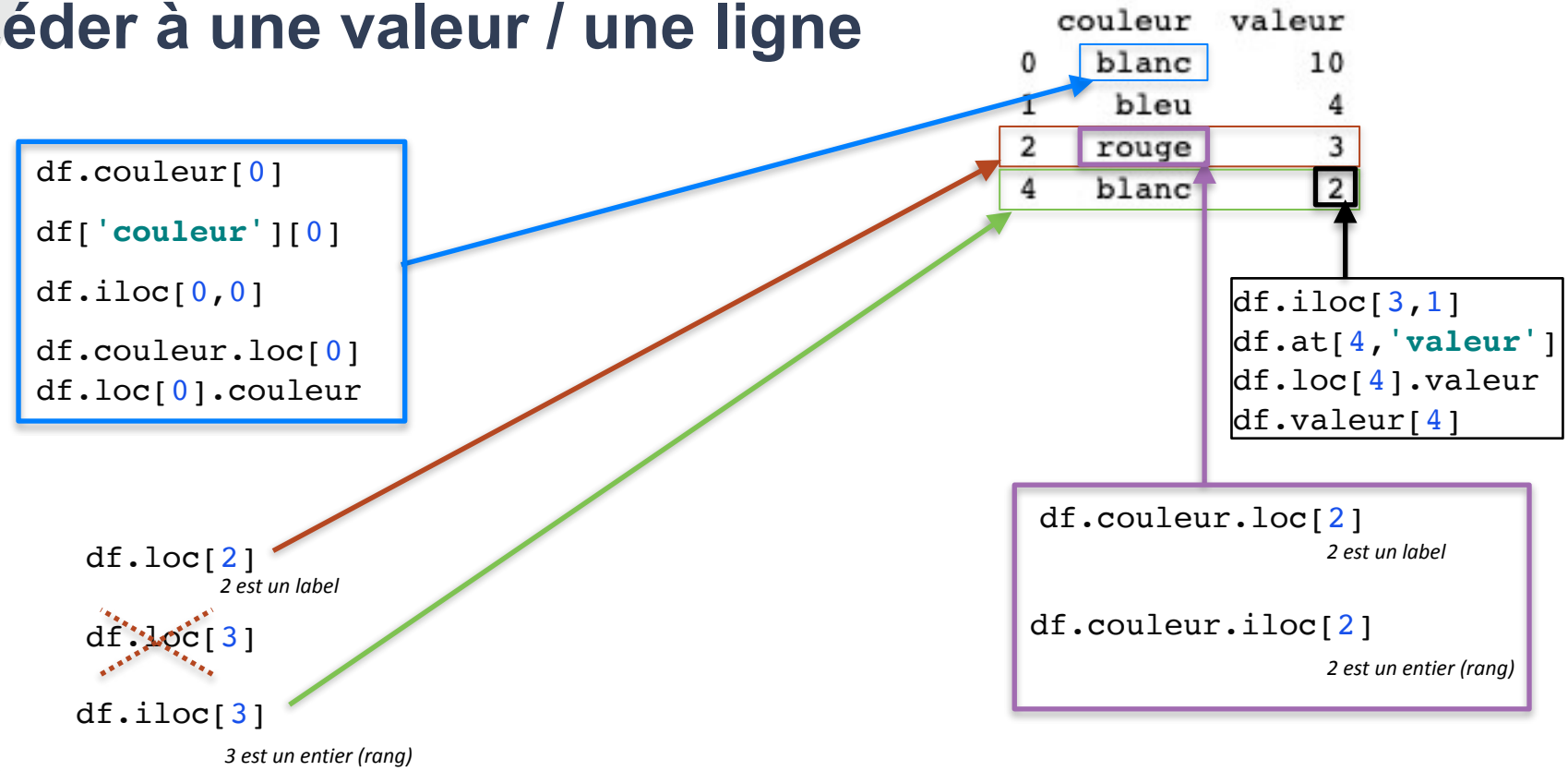
Une ligne sur 2 en partant de 0 (*i.e. les lignes paires*) :

```
df.iloc[::2]
```

Une ligne sur 2 en partant de 1 (*i.e. les lignes impaires*) :

```
df.iloc[1::2]
```

Accéder à une valeur / une ligne



Aix*Marseille université
Socialement engagée

Transformation et sélection

Les axes

	A	B
0	-0.235474	-0.150743

itérations selon les lignes
(pour chaque colonne)

```
df.mean()
df.mean(axis=0)
```

```
A   -0.235474
B   -0.150743
```

la moyenne de chaque colonne

itérations selon les colonnes
(pour chaque ligne)

```
df.mean(axis=1)
```

```
0   -0.193108
```

la moyenne de chaque ligne

Sélectionner les cellules selon une valeur

```
df = pd.DataFrame(np.arange(90,100).reshape(-1,2), columns=['A','B'])
```

	A	B
0	90	91
1	92	93
2	94	95
3	96	97
4	98	99

Quand la valeur est égale à 90 :

```
df[df==90]
```

	A	B
0	90.0	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN

Quand la valeur est comprise entre 90 et 94:

```
df[(df>90) & (df<94)]
```

	A	B
0	NaN	91.0
1	92.0	93.0
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN

Quand la valeur est paire :

```
df[df%2==0]
```

	A	B
0	90	NaN
1	92	NaN
2	94	NaN
3	96	NaN
4	98	NaN

```
df[df%2==0].A
```

Quand la valeur col. A est impaire :

```
df[df.A%2==1]
```

A	B

Supprimer ligne, colonne

```
df
[98] ✓ 0.0s  Ouvrir « df »
```

	0	1	2	3
0	0	-1	-2	4
1	1	0	-1	5
2	2	1	0	6
3	3	0	1	7

Comment supprimer une ligne ?

```
df.drop(1)

df.drop(index=[2,3], axis=0)
```
(par défaut)

```
df.drop(1)
[99] ✓ 0.0s
```

	0	1	2	3
0	0	-1	-2	4
2	2	1	0	6
3	3	0	1	7

```
df = df.drop(1)

df.drop(1, inplace=True)
```
(plus efficace en terme de mémoire)

Comment supprimer une colonne ?

`df.drop(columns=[1,2])` équivaut à : `df.drop([1,2], axis=1)` (crée une copie si valeur par défaut de inplace)

`del df.col1` (plus rapide)

`colonne_supprimee = df.pop(1)` (supprime la colonne et retourne la colonne supprimée)

```
df.pop(1)
[120] ✓ 0.0s
0   -1
1    0
2    1
3    0
Name: 1, dtype: int64
```

Comment supprimer en même temps une colonne et une ligne ?

```
df.drop(index=2, columns=1)
```

Supprimer sous condition

```
df = pd.DataFrame(np.arange(90,100).reshape(-1,2), columns=['A','B'])
```

	A	B
0	90	91
1	92	93
2	94	95
3	96	97
4	98	99

Supprimer les lignes dont au moins une valeur est supérieure à 94 :

```
index_a_supprimer = df[(df['A'] > 94) | (df['B'] > 94)].index
df.drop(index_a_supprimer)
```

	A	B
0	90	91
1	92	93

Explication :

```
df[(df['A'] > 94) | (df['B'] > 94)]
✓ 0.0s
```

	A	B
2	94	95
3	96	97
4	98	99

```
df[(df['A'] > 94) | (df['B'] > 94)].index
✓ 0.0s
Index([2, 3, 4], dtype='int64')
```

ou avec *query* : `df.query('A <= 94 and B <= 94')`

Sélectionner des lignes : *query*

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

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

La ligne pour laquelle la valeur de col2 est la chaîne de caractères *b* :

```
df.query('col2 == "b"')
```

	col1	col2	col3
1	20	b	1.2

La ligne pour laquelle la valeur de col2 est celle d'une variable *val* : `val = 1.3`

```
df.query('col3 == @val')
```

	col1	col2	col3
2	30	c	1.3

Sélectionner des lignes : *query*

```
data = { 'col avec espace' : [10,20,30,40],
         'col2' : ['a','b','c','d'],
         'col3' : [1.1,1.2,1.3,1.4]
         }
df = pd.DataFrame(data)
```

	col avec espace	col2	col3
0	10	a	1.1
1	20	b	1.2
2	30	c	1.3
3	40	d	1.4

Les lignes pour lesquelles la valeur de ‘col avec espace’ est > 10 :

```
df.query('`col avec espace` > 10')
```

	col avec espace	col2	col3
1	20	b	1.2
2	30	c	1.3
3	40	d	1.4

La ligne pour laquelle la valeur de ‘col avec espace’ est égale à la valeur retournée par une fonction f() :

```
def f():
    return 30
```

```
df.query('`col avec espace` == @f()')
```

	col avec espace	col2	col3
2	30	c	1.3

Sélectionner des lignes : *query*

```
df = pd.DataFrame(data={'a':[0,1,2],
'b':[-1,0,1], 'c':[-2,-1,0]})
```

	a	b	c
0	0	-1	-2
1	1	0	-1
2	2	1	0

Python pur vs. fonction *query()*

Opérateurs binaires	Opérateurs binaires Python
AND	&
OR	|
XOR	^
NOT	~

`df.query('a >= 0')` équivaut à : `df[df['a'] >= 0]` (légèrement plus rapide)

`df.query('(a >= 0) and (b < 0)')` équivaut à : `df[(df['a'] >= 0) & (df['b'] < 0)]`

~~`df[(df['a'] >= 0) and (df['b'] < 0)]`~~

`df.query('index == 1')` équivaut à : `df.loc[1]`

`df.query('index <= c+a')` équivaut à : `df[df.index <= df['c'] + df['a']]`

`df.query('(a + b + c) > 0')` équivaut à : `df.loc[df.sum(axis=1)>0]`

les lignes dont la somme des éléments est positive

```
df.sum()
✓ 0.0s
a    3
b    0
c   -3
```

```
df.sum(axis=1)
✓ 0.0s
0   -3
1    0
2    3
```

Sélectionner des colonnes

```
df = pd.DataFrame(data={'a':[0,1,2],
'b':[-1,0,1], 'c':[-2,-1,0]})
```

	a	b	c
0	0	-1	-2
1	1	0	-1
2	2	1	0

Les colonnes dont la somme des éléments est positive :

```
df.loc[:,df.sum()>0]
```

Les colonnes dont la somme des éléments est positive et les lignes dont la somme des éléments est supérieure à 1 :

```
df.loc[df.sum(axis=1)>1, df.sum()>0]
```

Les colonnes qui contiennent au moins une valeur positive :

	a	b
0	0	-1
1	1	0
2	2	1

```
df.loc[:, (df > 0).any()]
```

```
df>0
```
✓ 0.0s

	a	b	c
0	False	False	False
1	True	False	False
2	True	True	False

Les colonnes qui contiennent au moins deux valeurs négatives :

```
df.loc[:, (df < 0).sum() >= 2]
```

	c
0	-2
1	-1
2	0

```
(df>0).sum()
```
✓ 0.0s

a	2
b	1
c	0

nd d'éléments > 0 par colonne

Utiliser des masques booléens

```
df = pd.DataFrame(data={'a':[0,1,2], 'b':[-1,0,1], 'c':[-2,-1,0]})
```

	a	b	c
0	0	-1	-2
1	1	0	-1
2	2	1	0

```
df>=0
```
✓ 0.0s

	a	b	c
0	True	False	False
1	True	True	False
2	True	True	True

```
df['a']>=0
```
✓ 0.0s

```
0    True
1    True
2    True
```

```
df['b']<0
```
✓ 0.0s

```
0     True
1    False
2    False
```

```
(df['a']>=0) & (df['b']<0)
```
✓ 0.0s

```
0     True
1    False
2    False
```

```
masque = (df['a']>=0) & (df['b']<0)
```

```
df[masque]
```

	a	b	c
0	0	-1	-2

équivaut à : `df.query('a >= 0 and b < 0')`

```
masque = (df>=0)   df[masque]
```

	a	b	c
0	0	NaN	NaN
1	1	0.0	NaN
2	2	1.0	0.0

```
masque.any(axis=1)
```

```
0    True
1    True
2    True
```

```
df[masque.any(axis=1)]
```

	a	b	c
0	0	-1	-2
1	1	0	-1
2	2	1	0

```
masque.all(axis=1)
```

```
0    False
1    False
2     True
```

	a	b	c
2	2	1	0

Filtrer les lignes selon des valeurs

```
df = pd.DataFrame(data={'a':[0,1,2], 'b':[-1,0,1], 'c':[-2,-1,0]})
```

	a	b	c
0	0	-1	-2
1	1	0	-1
2	2	1	0

```
liste = [1, 2]
```

```
df[df['a'].isin(liste)]
```

	a	b	c
1	1	0	-1
2	2	1	0

```
print(df['a'].isin(liste))
```

```
0    False
1     True
2     True
```

```
df[df.isin(liste)]
```

	a	b	c
0	NaN	NaN	NaN
1	1.0	NaN	NaN
2	2.0	1.0	NaN

```
df.isin(liste)
```

	a	b	c
0	False	False	False
1	True	False	False
2	True	True	False

```
df[df['a'].isin(liste)][['a','b']]
```

	a	b
1	1	0
2	2	1

Transformer une série en un *dataframe*

N'avoir que la 1ère ligne de la sélection dans un dataframe

	a	b
1	1	0

```
df[df['a'].isin(liste)][['a','b']].loc[1]
```

```
a    1
b    0
Name: 1, dtype: int64
```

```
type(df[df['a'].isin(liste)][['a','b']].loc[1])
```

pandas.core.series.Series

```
pd.DataFrame(df[df['a'].isin(liste)][['a','b']].loc[1])
```

	1
a	1
b	0

```
DataFrame.transpose(pd.DataFrame(df[df['a'].isin(liste)][['a','b']].loc[1]))
```

	a	b
1	1	0

Utiliser *filter*

```
df = pd.DataFrame({
    'A_1': [1, 2, 3],
    'A_2': [4, 5, 6],
    'B': [7, 8, 9],
    'C_9': [10, 11, 12]
})
```

	A_1	A_2	B	C_9
0	1	4	7	10
1	2	5	8	11
2	3	6	9	12

Les colonnes dont le nom contient *A_* :

```
df.filter(like='A_')
```

	A_1	A_2
0	1	4
1	2	5
2	3	6

équivaut à : `df.loc[:, df.columns.str.startswith('A')]`

Les colonnes dont le nom contient un chiffre :

```
df.filter(regex='_[0-9]')
```

équivaut à :

```
df.filter(regex='_\\d')
df.filter(regex=r'_\d')
```

Les colonnes dont le nom commence par *A* ou *C* :

```
df.filter(regex='^[AC]')
```

	A_1	A_2	C_9
0	1	4	10
1	2	5	11
2	3	6	12

Utiliser *filter*

```
df = pd.DataFrame({
    1 : [1, 2, 3],
    'A_2': [4, 5, 6],
    'B': [7, 8, 9],
    'C_9': [10, 11, 12]
})
```

	1	A_2	B	C_9
0	1	4	7	10
1	2	5	8	11
2	3	6	9	12

pandas.DataFrame.filter

DataFrame.filter(*items=None, like=None, regex=None, axis=None*) [source]

Subset the dataframe rows or columns according to the specified index labels.

Note that this routine does not filter a dataframe on its contents. The filter is applied to the labels of the index.

Parameters:

items : *list-like*
Keep labels from axis which are in items.

like : *str*
Keep labels from axis for which "like in label == True".

regex : *str (regular expression)*
Keep labels from axis for which re.search(regex, label) == True.

axis : *{0 or 'index', 1 or 'columns', None}, default None*
The axis to filter on, expressed either as an index (int) or axis name (str). By default this is the info axis, 'columns' for DataFrame. For *Series* this parameter is unused and defaults to *None*.

```
df.filter(like='1')
```

	1
0	1
1	2
2	3

```
df.filter(like='1', axis=0)
```

	1	A_2	B	C_9
1	2	5	8	11

```
df = pd.DataFrame({
    1 : [1, 2, 3],
    'A_2': [4, 5, 6],
    'B': [7, 8, 9],
    'C_9': [10, 11, 12]
})
```

	1	A_2	B	C_9
0	1	4	7	10
1	2	5	8	11
2	3	6	9	12

Les colonnes dont le nom ne contient pas *A_* :

```
df.loc[:, ~df.columns.str.startswith('A')]
```

```
df = pd.DataFrame({
    1 : [1, 2, 3],
    'A_2': [4, 5, 6],
    'B': [7, 8, 9],
    'C_9': [10, 11, 12]
}, index=['a', 'b', 'c'])
```

	1	A_2	B	C_9
a	1	4	7	10
b	2	5	8	11
c	3	6	9	12

Les lignes dont le nom ne contient pas *a* :

```
df.loc[~df.index.str.startswith('a'),:]
```

	1	A_2	B	C_9
b	2	5	8	11
c	3	6	9	12

Le module *RegEx*

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

RegEx Functions

The `re` module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Example

Print the part of the string where there was a match.

The regular expression looks for any words that starts with an upper case "S":

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

Example

Replace every white-space character with the number 9:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

Example

Search the string to see if it starts with "The" and ends with "Spain":

```
import re

txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

Example

Print a list of all matches:

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Special Sequences

A special sequence is a `\` followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Sets

A set is a set of characters inside a pair of square brackets `[]` with a special meaning:

Set	Description
[arn]	Returns a match where one of the specified characters (`a` , `r` , or `n`) are present
[a-n]	Returns a match for any lower case character, alphabetically between `a` and `n`
[^arn]	Returns a match for any character EXCEPT `a` , `r` , and `n`
[0123]	Returns a match where any of the specified digits (`0` , `1` , `2` , or `3`) are present
[0-9]	Returns a match for any digit between `0` and `9`
[0-5][0-9]	Returns a match for any two-digit numbers from `00` and `59`
[a-zA-Z]	Returns a match for any character alphabetically between `a` and `z` , lower case OR upper case
[+]	In sets, `+` , `*` , `.` , `|` , `()` , `$` , `{}` has no special meaning, so `[+]` means: return a match for any `+` character in the string

Remplacer les valeurs selon une condition: *where* et *mask*

```
df = pd.DataFrame(np.arange(90,100).reshape(-1,2), columns=['A','B'])
```

	A	B
0	90	91
1	92	93
2	94	95
3	96	97
4	98	99

```
masque = (df % 2 ==0)
```

```
print(masque)
```

	A	B
0	True	False
1	True	False
2	True	False
3	True	False
4	True	False

— *where :* quand la condition est vraie, valeur conservée, quand la condition est fausse : la valeur est remplacée par le deuxième argument

```
df.where(masque,0)
```

	A	B
0	90	0
1	92	0
2	94	0
3	96	0
4	98	0

```
df.where(masque, df+1)
```

	A	B
0	90	92
1	92	94
2	94	96
3	96	98
4	98	100

— *mask* fait l'inverse de *where*

```
df.mask(masque, df*2)
```

	A	B
0	180	91
1	184	93
2	188	95
3	192	97
4	196	99

	A	B
0	90	91
1	92	93
2	94	95
3	96	97
4	98	99

Remplacer les valeurs comprises entre 92 et 96 par 90 :

avec *where* : `df.where((df < 92) | (df > 96), 90)`

copie

avec *mask* : `df.mask((df >= 92) & (df <= 96), 90)`

copie

sans *where* ni *mask* : `df[(df >= 92) & (df <= 96)] = 90`

modification directe

	A	B
0	90	91
1	90	90
2	90	90
3	90	97
4	98	99

Remplacer des valeurs par correspondances : *map* et *replace*

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

```
df['col2'] = df['col2'].map({'a':'abc','b':'bcd'})
```

	col1	col2	col3
0	10	abc	1.1
1	20	bcd	1.2
2	30	NaN	1.3
3	40	NaN	1.4

NaN pour les correspondances non définie

```
df['col2'] = df['col2'].replace({'a':'abc','b':'bcd'})
```

	col1	col2	col3
0	10	abc	1.1
1	20	bcd	1.2
2	30	c	1.3
3	40	d	1.4

valeur initiale conservée pour les correspondances non définies

Remplacer des valeurs selon une fonction : *apply*

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

```
def fonction_carre (x):
    return x**2

df['col3']=df['col3'].apply(fonction_carre)
```

	col1	col2	col3
0	10	abc	1.21
1	20	bcd	1.44
2	30	c	1.69
3	40	d	1.96

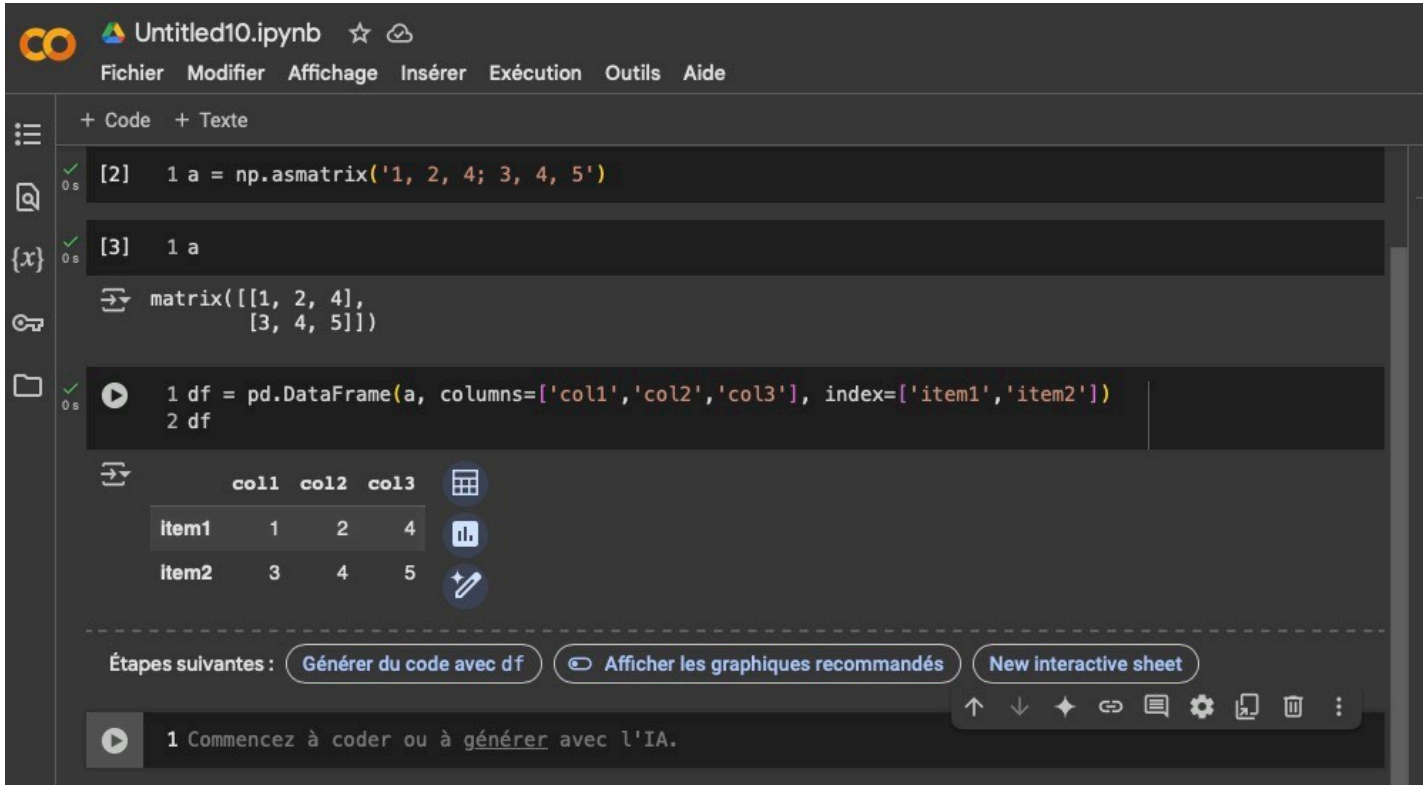
```
def fonction_puissance (x,n):
    return x**n

df['col3']=df['col3'].apply(fonction_puissance,n=100)
```

	col1	col2	col3
0	10	a	1.378061e+04
1	20	b	8.281797e+07
2	30	c	2.479335e+11
3	40	d	4.100186e+14

Aix*Marseille université
Socialement engagée

Edition en direct Colab / VsCode

Dans Google Colab



```
1 df = pd.DataFrame(a, columns=['col1','col2','col3'], index=['item1','item2'])
2 df
```

1 to 2 of 2 entries Filter

Copy table
CSV JSON Markdown
Copy

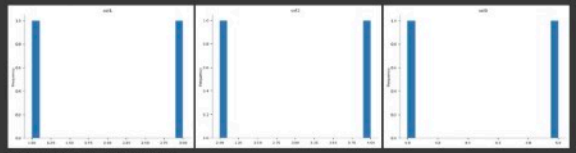
```
index,col1,col2,col3
item1,1,2,4
item2,3,4,5
```

index	col1 ▲	col2	col3
item1	1	2	4
item2	3	4	5

Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Distributions



Categorical distributions

```
1 df = pd.DataFrame(a, columns=['col1','col2','col3'], index=['item1','item2'])
2 df
```

1 to 2 of 2 entries Filter

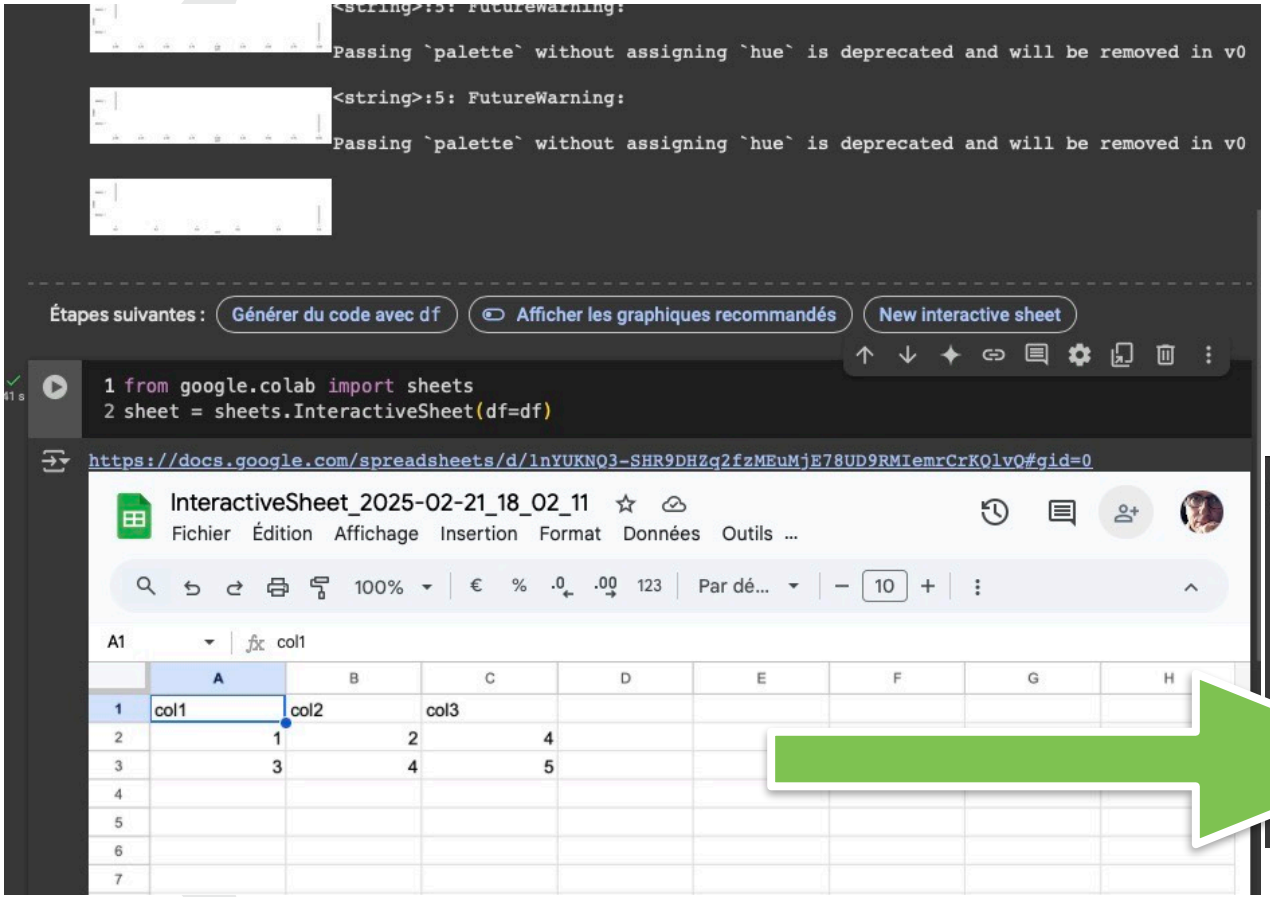
index: col1: to

col2: to col3: to

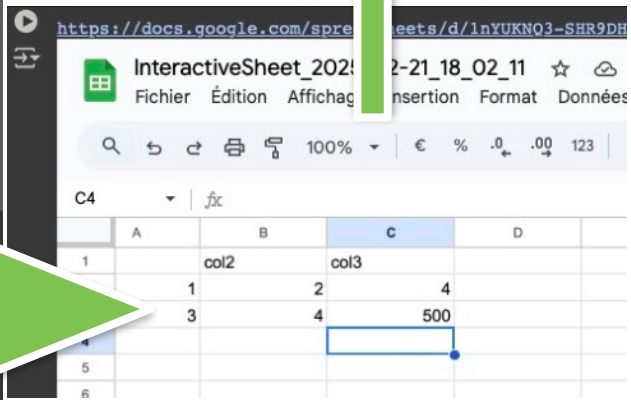
Search by all fields:

index	col1 ▲	col2	col3
item1	1	2	4
item2	3	4	5

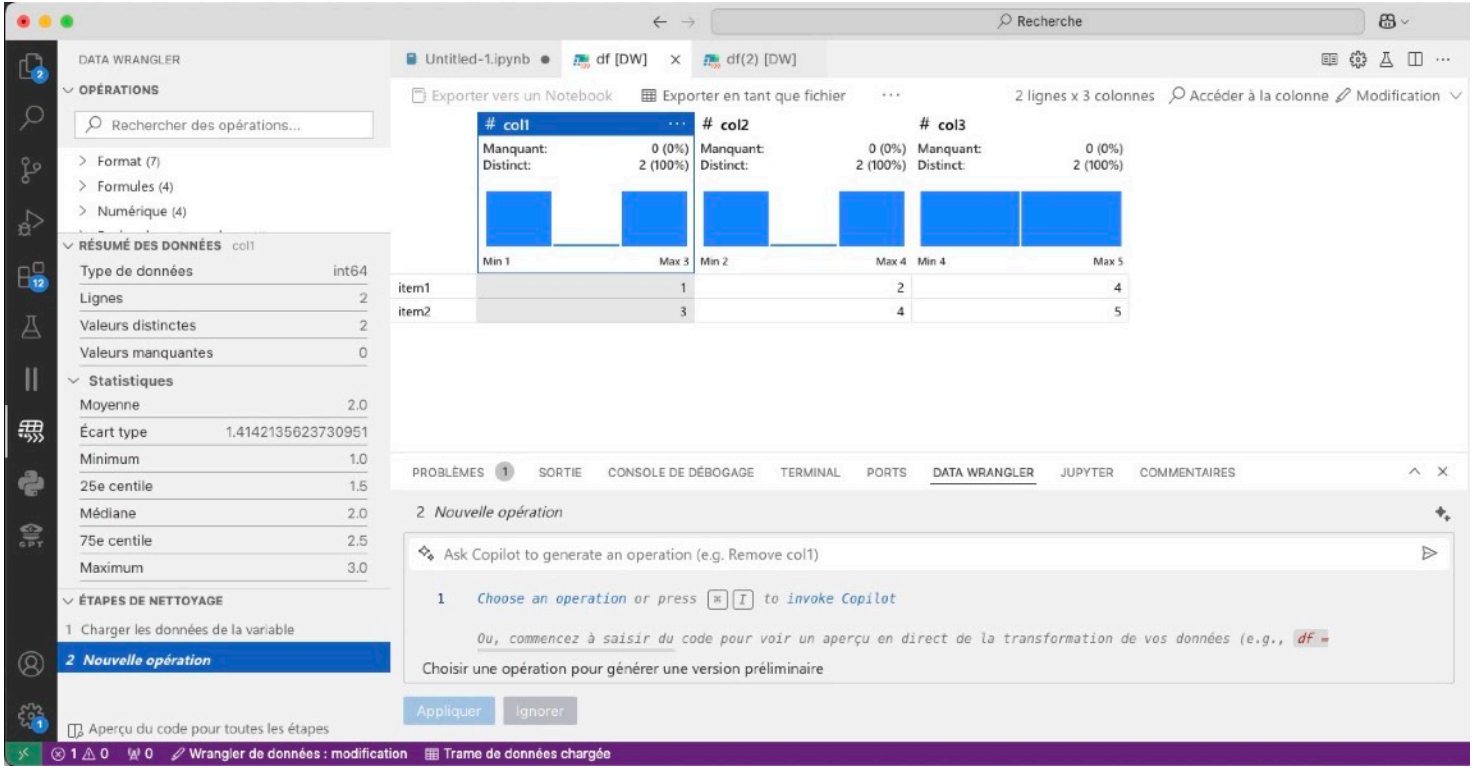
Show 25 per page

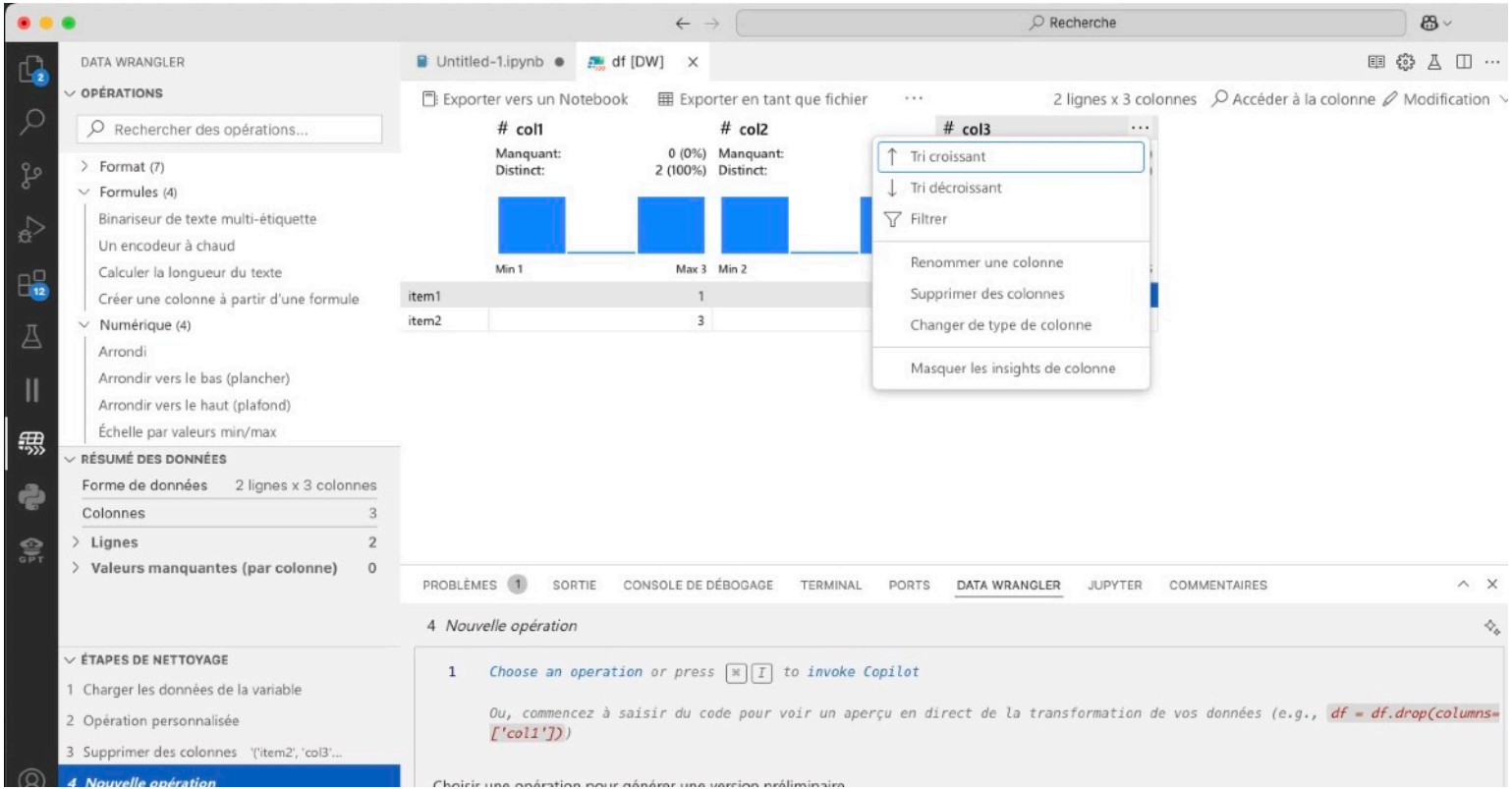

<string>:5: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0
<string>:5: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0
Étapes suivantes :
Générer du code avec df
Afficher les graphiques recommandés
New interactive sheet
1 from google.colab import sheets
2 sheet = sheets.InteractiveSheet(df=df)
https://docs.google.com/spreadsheets/d/1nYUKNQ3-SHR9DHZq2fzMEuMjE78UD9RMIemrCrKQlvQ#gid=0
InteractiveSheet_2025-02-21_18_02_11
Fichier Édition Affichage Insertion Format Données Outils ...
A1
col1
col2
col3


1 sheet.as_df()
col1 col2 col3
0 1 2 4
1 3 4 500

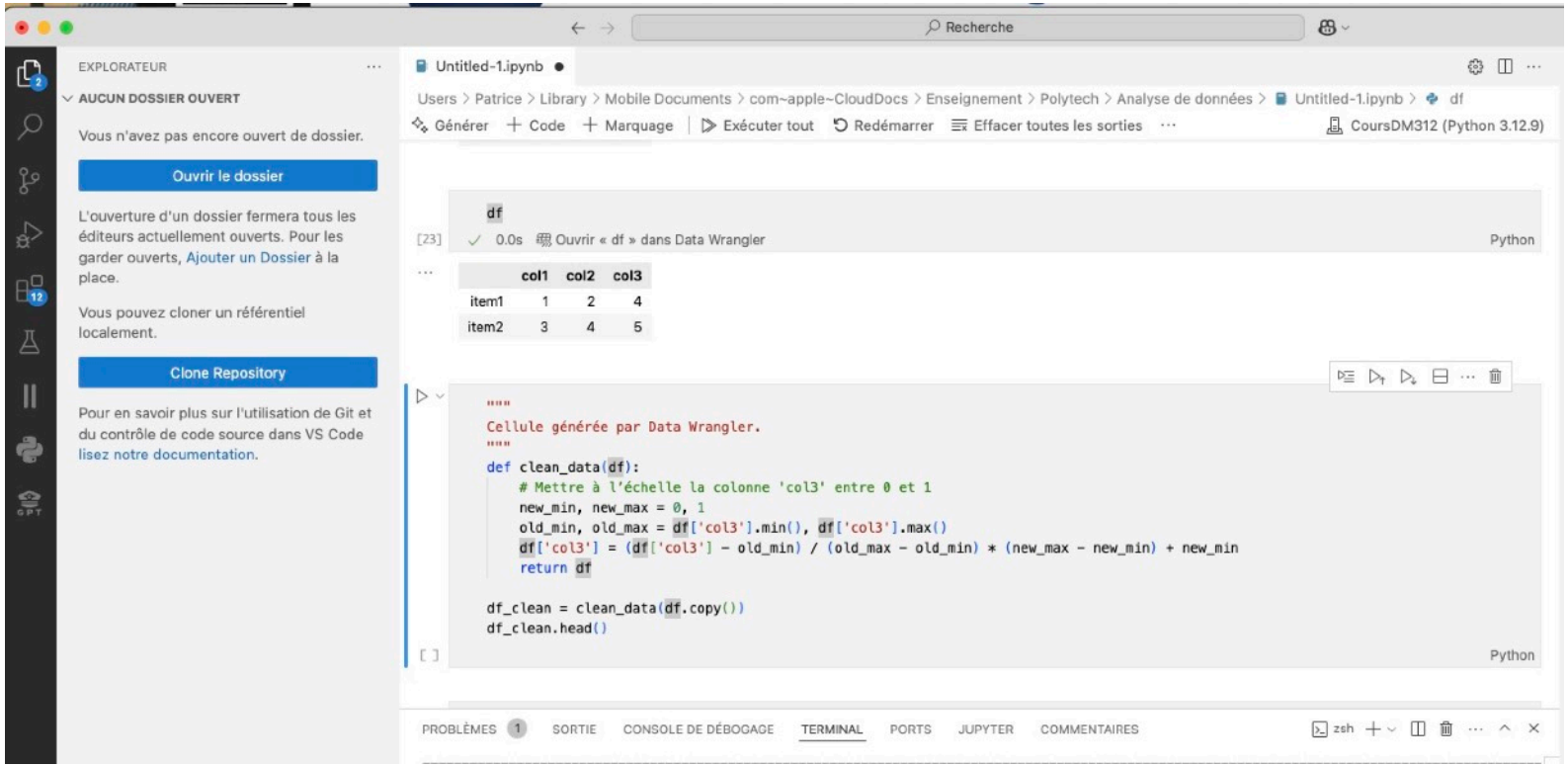

InteractiveSheet_2025-02-21_18_02_11
Fichier Édition Affichage Insertion Format Données
C4
col2
col3
3
4
500

Exemple dans VsCode




DATA WRANGLER
OPÉRATIONS
Rechercher des opérations...
Format (7)
Formules (4)
Binariseur de texte multi-étiquette
Un encodeur à chaud
Calculer la longueur du texte
Créer une colonne à partir d'une formule
Numérique (4)
Arrondi
Arrondir vers le bas (plancher)
Arrondir vers le haut (plafond)
Échelle par valeurs min/max
RÉSUMÉ DES DONNÉES
Forme de données 2 lignes x 3 colonnes
Colonnes 3
Lignes 2
Valeurs manquantes (par colonne) 0
ÉTAPES DE NETTOYAGE
1 Charger les données de la variable
2 Opération personnalisée
3 Supprimer des colonnes
4 Nouvelle opération
Untitled-1.ipynb
df [DW]
Exporter vers un Notebook
Exporter en tant que fichier
2 lignes x 3 colonnes
Accéder à la colonne
Modification
col1
Manquant: 0 (0%)
Distinct: 2 (100%)
col2
Manquant:
Distinct:
col3
Tri croissant
Tri décroissant
Filtrer
Renommer une colonne
Supprimer des colonnes
Changer de type de colonne
Masquer les insights de colonne
Min 1
Max 3
Min 2
item1 1
item2 3
PROBLÈMES 1
SORTIE
CONSOLE DE DÉBOGAGE
TERMINAL
PORTS
DATA WRANGLER
JUPYTER
COMMENTAIRES
4 Nouvelle opération
1 Choose an operation or press ⌘ I to invoke Copilot
Ou, commencez à saisir du code pour voir un aperçu en direct de la transformation de vos données (e.g., df = df.drop(columns=['col1']))
Choisir une opération pour générer une version préliminaire

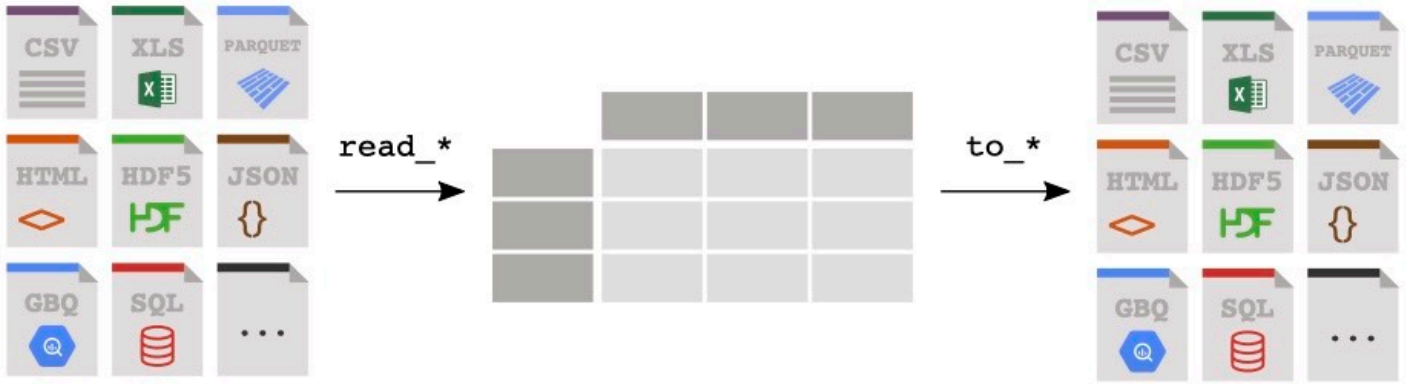
https://learn.microsoft.com/fr-fr/shows/visual-studio-code/mastering-your-data-with-data-wrangler-in-vs-code



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.

encoding : *str, optional*

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

thousands : *str, optional*

Thousands separator.

decimal : *str, default '.'*

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

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 (Colab)

A la création
d'un notebook Colab :

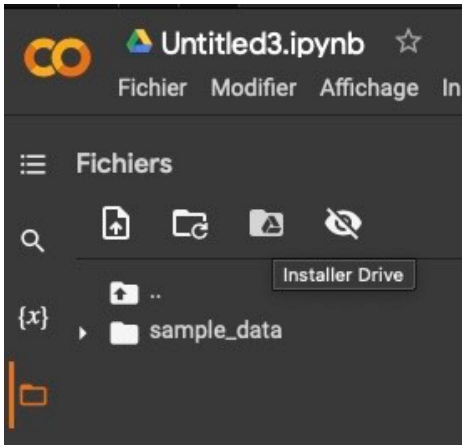
```
1 import os
2 os.getcwd()
```

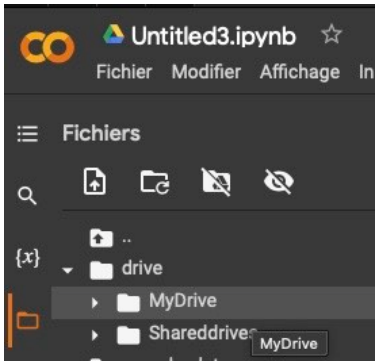
```
'/content'
```

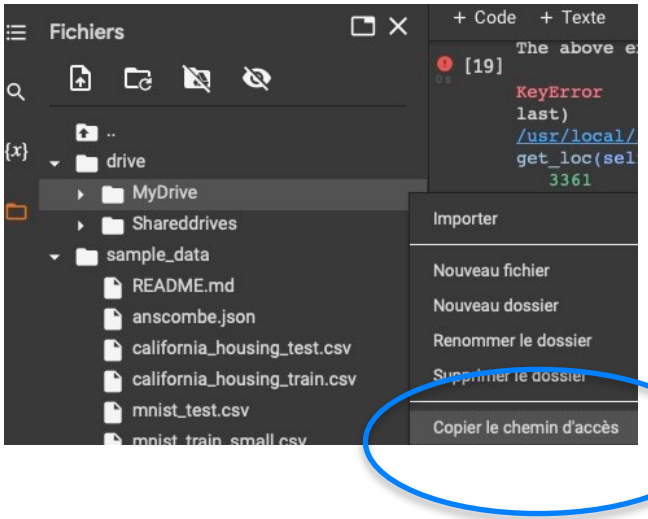
Positionnement
dans Google Drive :

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

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







Les chemins d'accès selon l'OS :

Pour Windows, plusieurs manières de les noter :

```
chemin = "C:\\Users\\nom\\Documents\\fichier.txt"
chemin = "C:/Users/nom/Documents/fichier.txt"
chemin = r"C:\Users\nom\Documents\fichier.txt"
```

Pour MacOS et Linux :

```
chemin = "/Volume/Users/nom/Documents/fichier.txt"
```

Utilisation de Path :

```
from pathlib import Path
chemin = Path("Users", "nom", "Documents", "fichier.txt")
print(chemin.parent)  # dossier/sous_dossier
print(chemin.name)    # fichier.txt
print(chemin.suffix)  # .txt
```

✓ 0.0s

```
Users/nom/Documents
fichier.txt
.txt
```

Exemple : les données *Titanic*

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

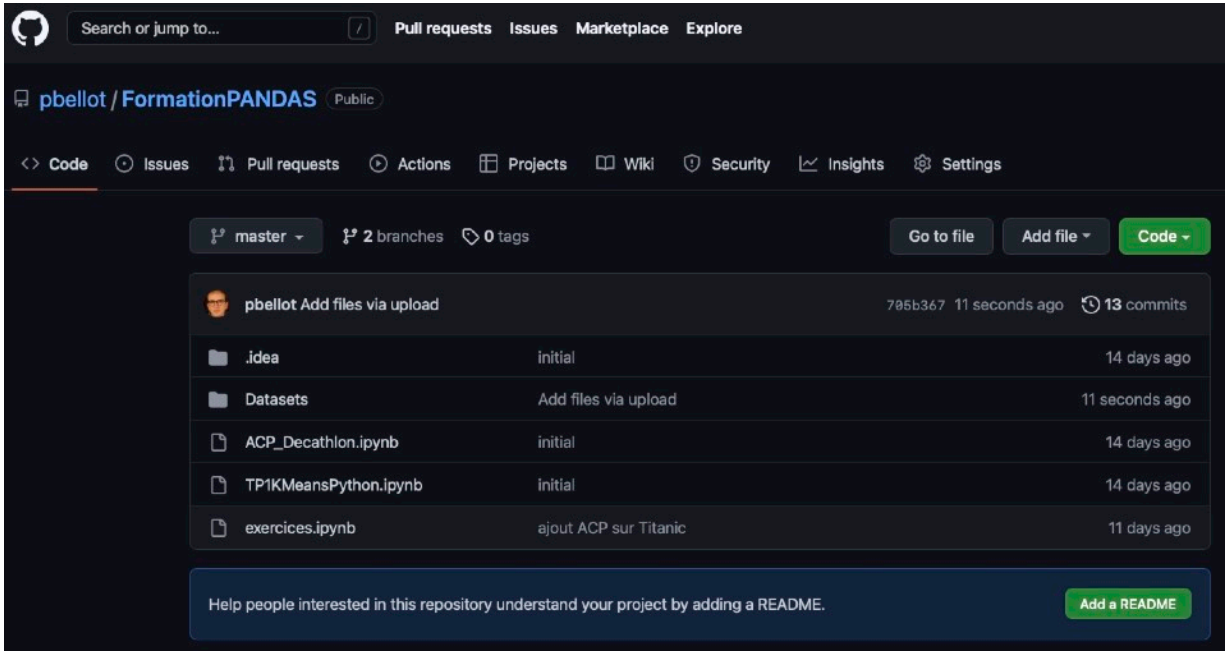
```
titanic.head(10)
```

	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(

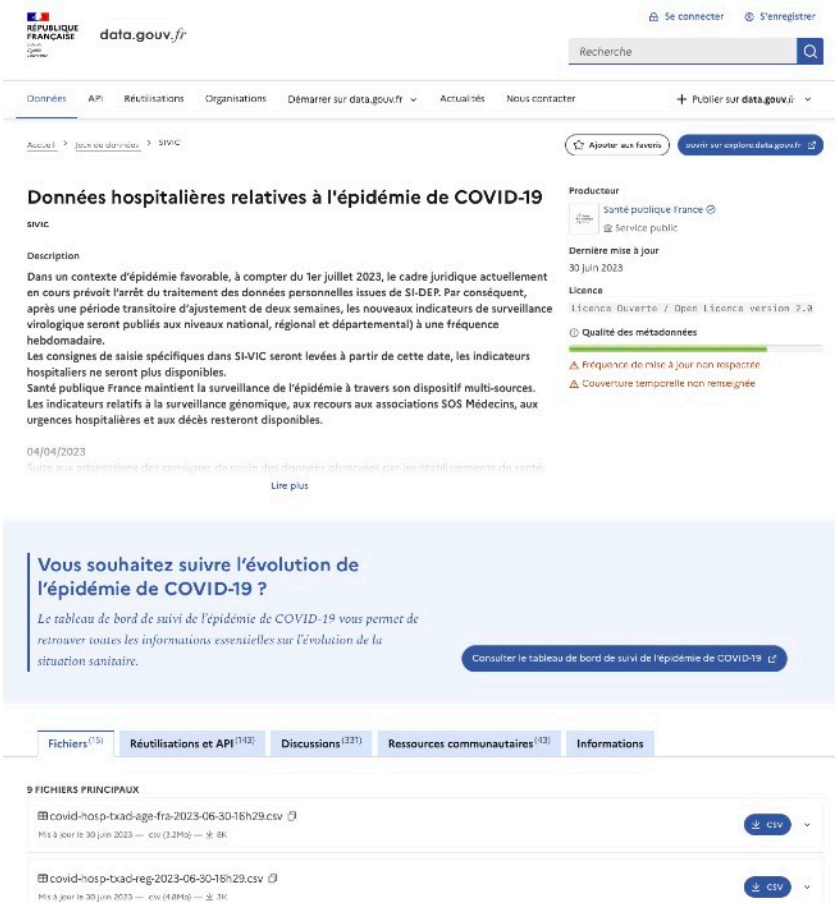
- PassengerId: Id of every passenger.
- Survived: This feature have value 0 and 1. 0 for not survived and 1 for survived.
- Pclass: There are 3 classes: Class 1, Class 2 and Class 3.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger.
- SibSp: Indication that passenger have 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.

Données à récupérer ici :

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
```

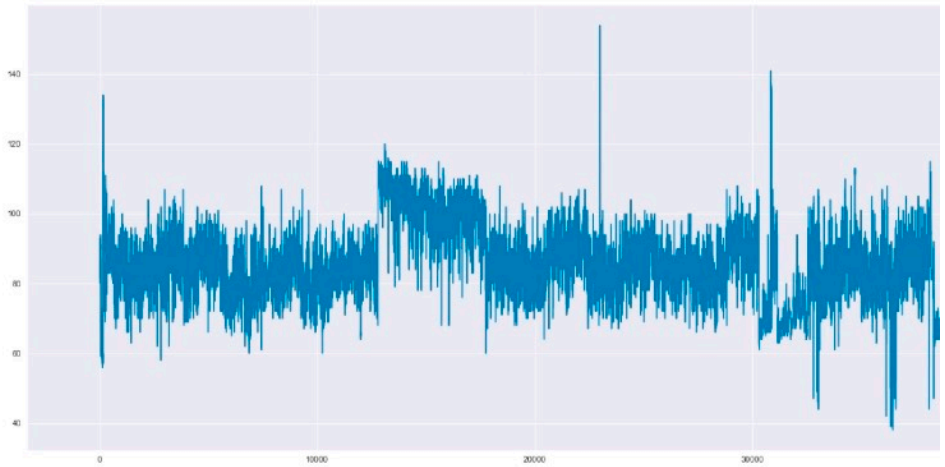
```
rythme.head(10)
```

	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

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

Lecture de tables HTML

```
<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		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* (suite)

(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 :

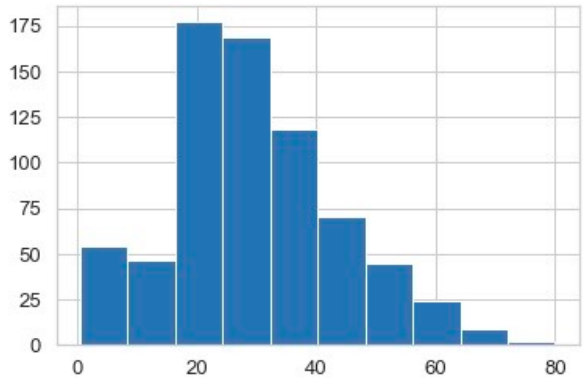
```
titanic.dropna(axis=1).info()
```

```
<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

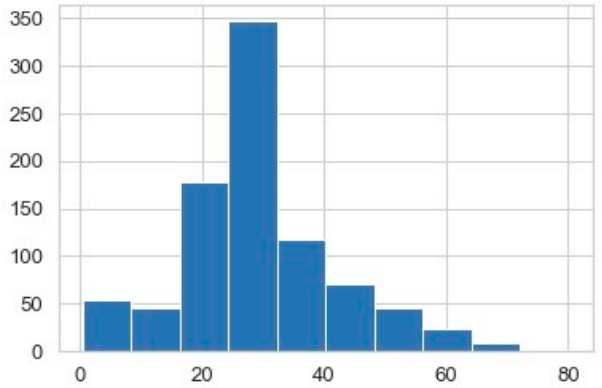
```
titanic.Age.hist()
```

```
<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
```

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
```

Serie

`dg['c1','Gap']`

`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

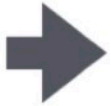
Pivot

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

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



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)
```

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

	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

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 : *groupie*

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']).count()
```

Animal	Max Speed
Falcon	2
Parrot	2

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

Animal	Max Speed
Falcon	375.0
Parrot	25.0

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

```
>>> 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
```

Subdiviser un dataframe en plusieurs selon les valeurs d'une colonne

```
data = {'Category': ['A', 'A', 'B', 'B'],
'Value': [1, 2, 3, 4]}
df = pd.DataFrame(data)
```

	Category	Value
0	A	1
1	A	2
2	B	3
3	B	4

En créant de nouveaux dataframes :

```
subsets =  {}
for category in df['Category'].unique():
    subsets[category] = df[df['Category'] == category]

for subset in subsets:
    display(subsets[subset])
```

	Category	Value
0	A	1
1	A	2

	Category	Value
2	B	3
3	B	4

Sans créer de nouveaux dataframes :

```
grouped = df.groupby('Category')
for name, group in grouped:
    display(group)
```

En créant une liste :

```
subsets = [group for name, group in df.groupby('Category')]
for subset in subsets:
    display(subset)
```

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* de tailles différentes

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

f1

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

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

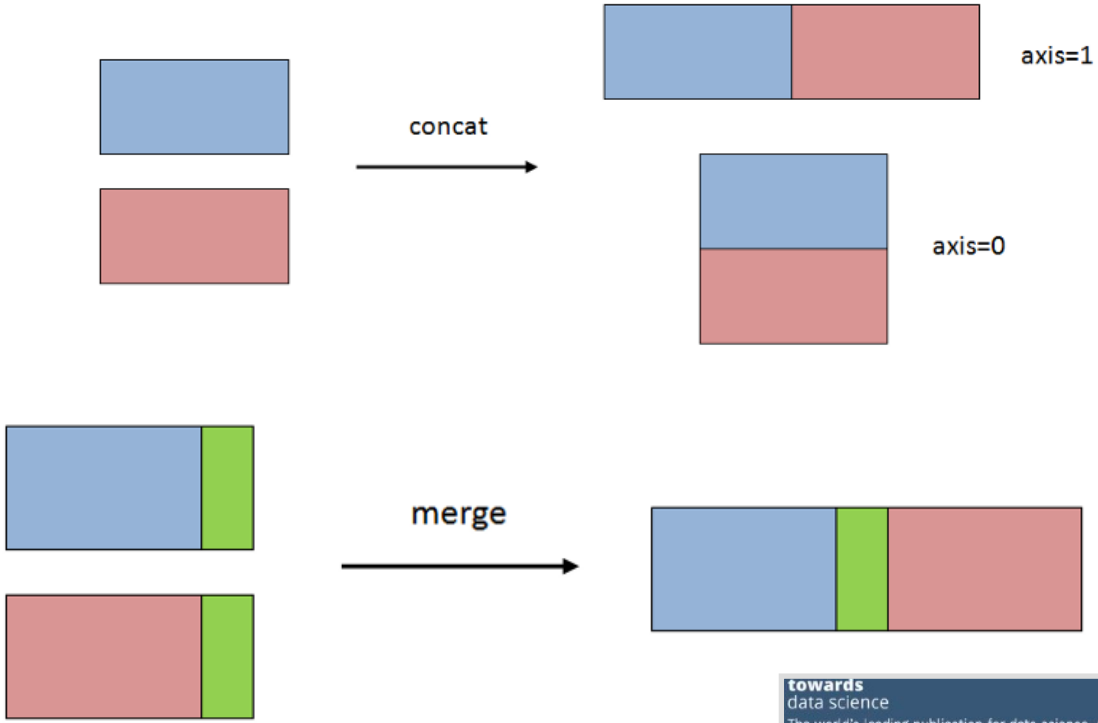
f2

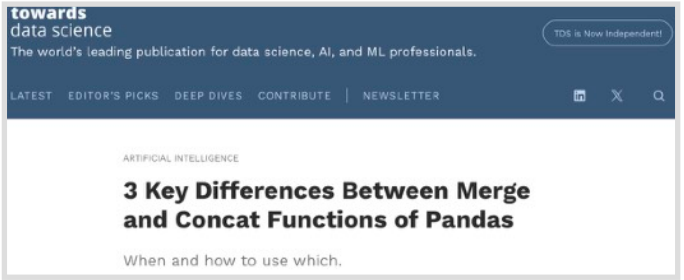
	c4	c2	c5	c6
a	0	1	2	3
d	4	5	6	7

```
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





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

Combiner 2 data frames : *concat*

```
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"])
```

```
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
```

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

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

```
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

Aix*Marseille université
Socialement engagée

Statistique descriptive et visualisation

```
titanic.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
titanic["Age"].mean()
```

```
titanic["Age"].mean()
```

```
29.69911764705882
```

```
titanic[["Age", "Fare"]].median()
```

```
titanic[["Age", "Fare"]].median()
```

```
Age     28.0000
Fare    14.4542
dtype: float64
```

```
titanic.groupby("Sex").mean()
```

```
titanic.groupby("Sex").mean()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
Sex							
female	431.028662	0.742038	2.159236	27.915709	0.694268	0.649682	44.479818
male	454.147314	0.188908	2.389948	30.726645	0.429809	0.235702	25.523893

```
titanic[["Sex", "Age"]].groupby("Sex").mean()
titanic.groupby("Sex")[["Sex", "Age"]].mean()
```

	Age
Sex	
female	27.915709
male	30.726645

```
titanic[["Survived","Pclass"]].groupby("Pclass").describe()
```

	Survived							
	count	mean	std	min	25%	50%	75%	max
Pclass								
1	216.0	0.629630	0.484026	0.0	0.0	1.0	1.0	1.0
2	184.0	0.472826	0.500623	0.0	0.0	0.0	1.0	1.0
3	491.0	0.242363	0.428949	0.0	0.0	0.0	0.0	1.0

```
titanic.groupby("Pclass").sum()
```

	PassengerId	Survived	Age	SibSp	Parch	Fare
Pclass						
1	99705	136	7111.42	90	77	18177.4125
2	82056	87	5168.83	74	70	3801.8417
3	215625	119	8924.92	302	193	6714.6951

```
titanic.groupby("Sex").sum()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
Sex							
female	135343	233	678	7286.00	218	204	13966.6628
male	262043	109	1379	13919.17	248	136	14727.2865

What is the mean ticket fare price for each of the sex and cabin class combinations?

```
titanic.groupby(["Sex", "Pclass"])["Fare"].mean()
```

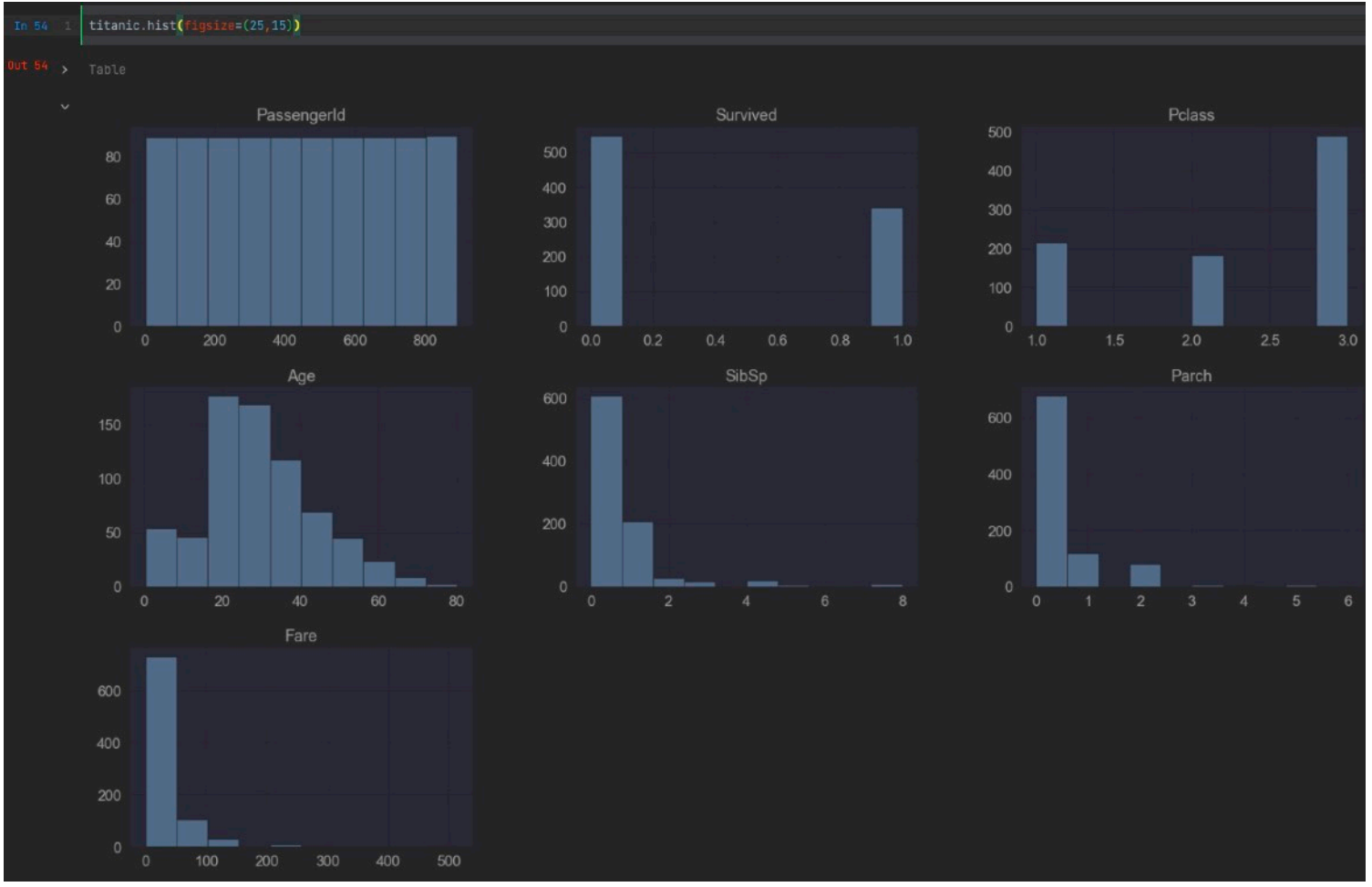
```
Sex     Pclass
female  1         106.125798
        2          21.970121
        3          16.118810
male    1          67.226127
        2          19.741782
        3          12.661633
```

```
titanic.groupby(["Pclass","Sex"])["Fare"].mean()
```

```
Pclass  Sex
1       female    106.125798
        male       67.226127
2       female     21.970121
        male       19.741782
3       female     16.118810
        male       12.661633
```

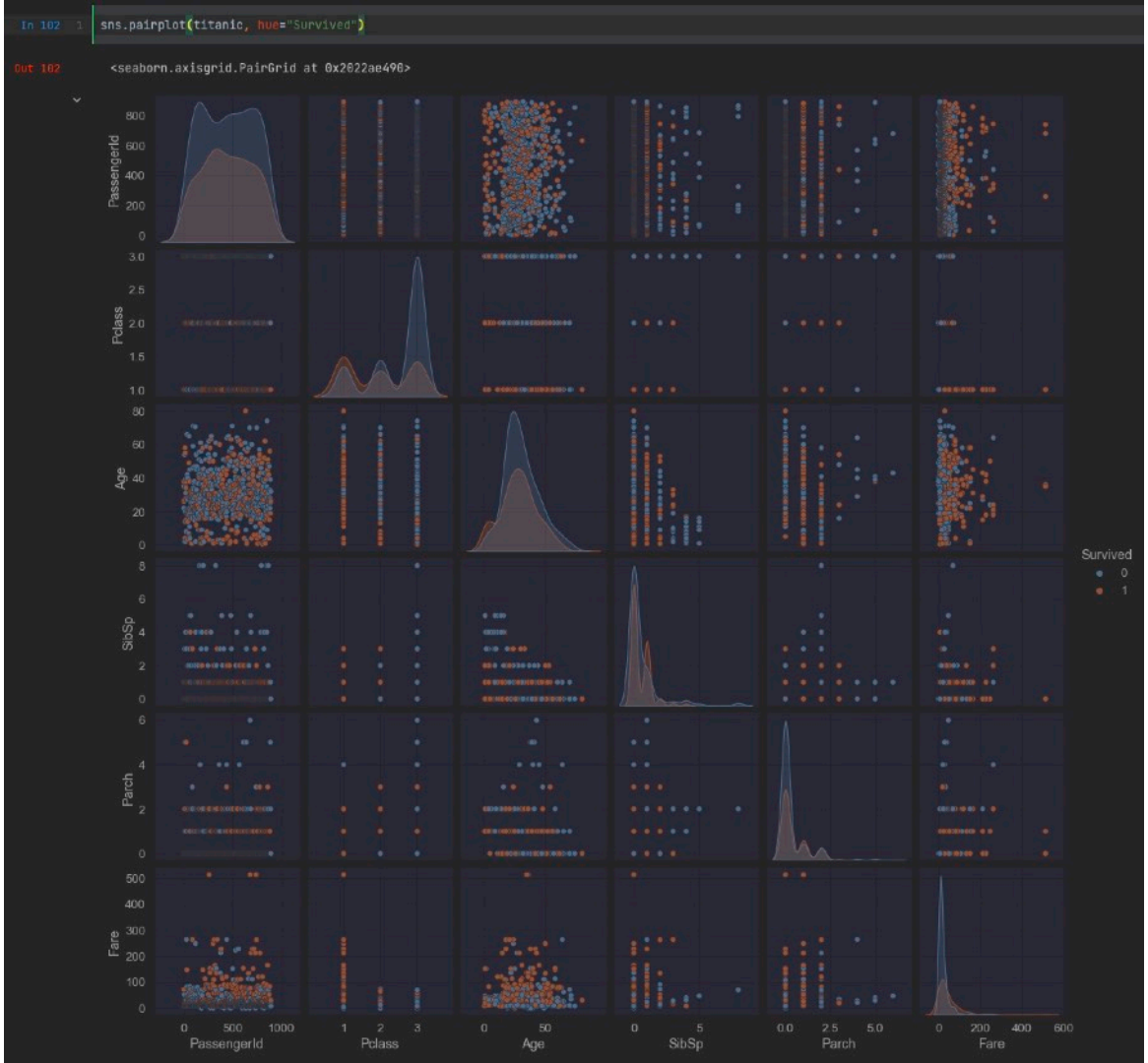
```
In 54  1  titanic.hist(figsize=(25,15))
```

Out 54 › Table



```
In 102  1  sns.pairplot(titanic, hue="Survived")
```

```
Out 102  <seaborn.axisgrid.PairGrid at 0x2022ae496>
```

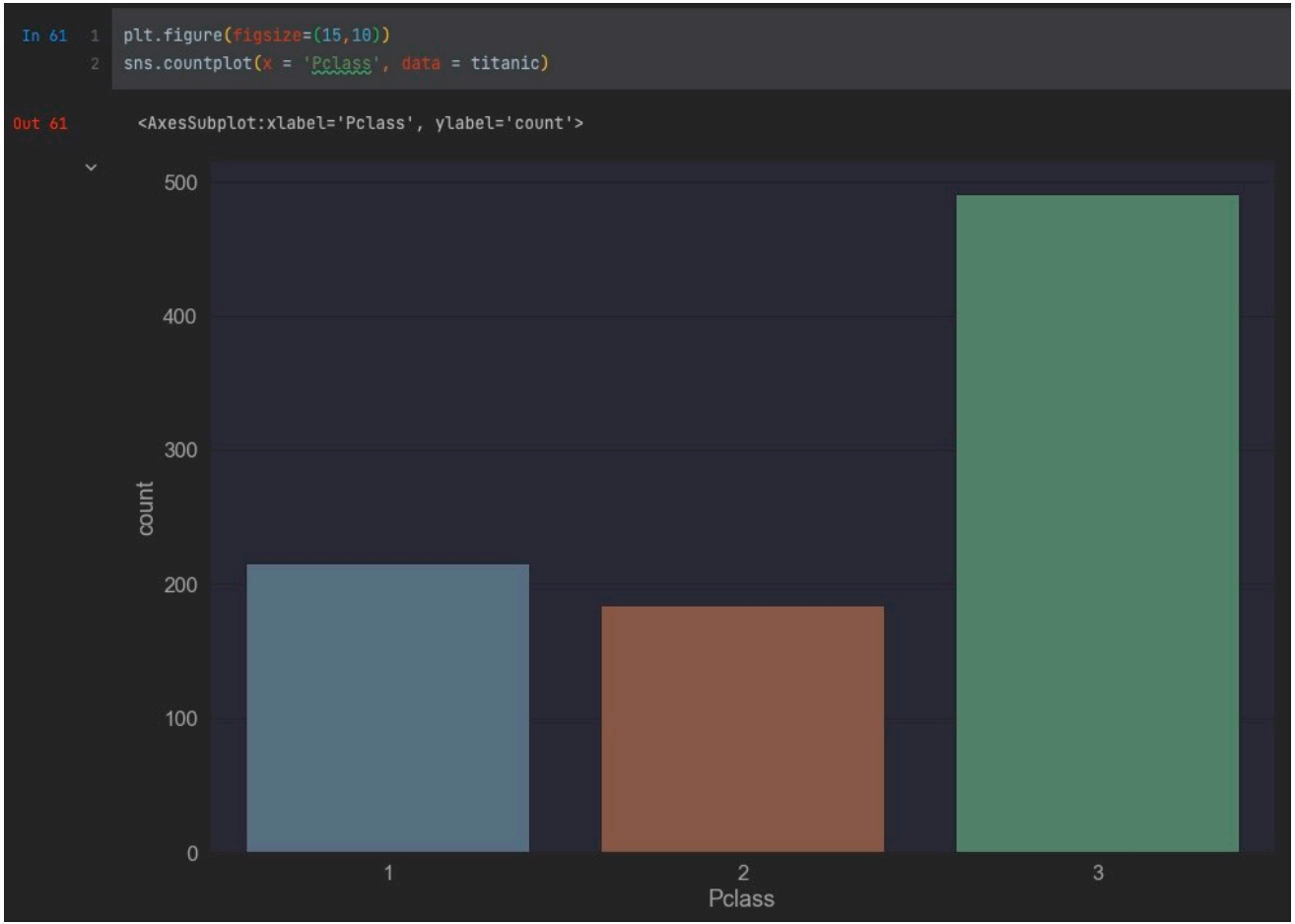


```
import matplotlib.pyplot as plt
import seaborn as sns
```

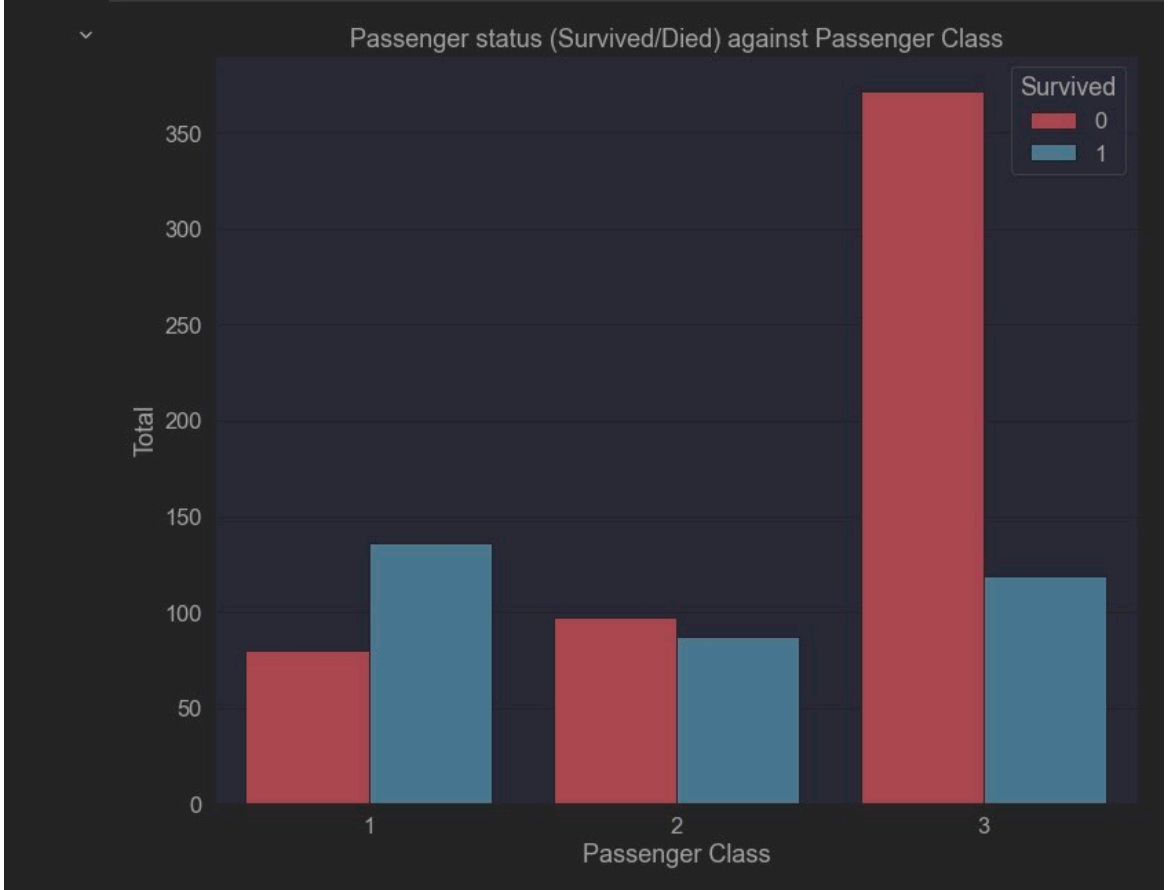


```
In 61   1  plt.figure(figsize=(15,10))
        2  sns.countplot(x = 'Pclass', data = titanic)
```

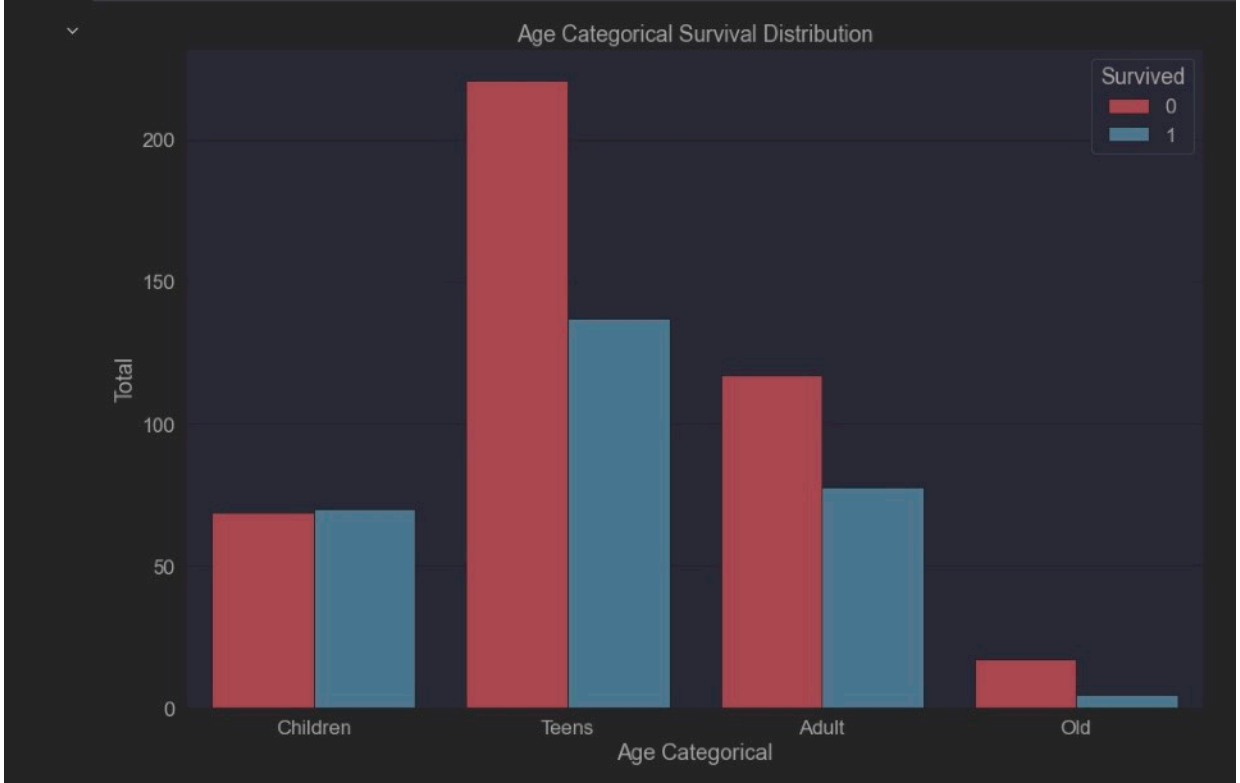
```
Out 61  <AxesSubplot:xlabel='Pclass', ylabel='count'>
```

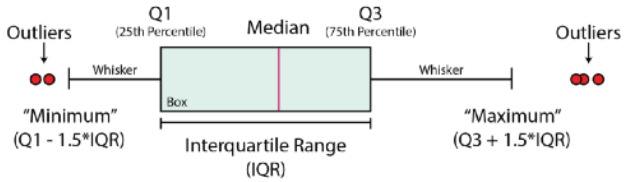


In 57
```
fig, ax = plt.subplots(1,1, figsize = (12,10))
ax = sns.countplot(x = 'Pclass', hue = 'Survived', palette = 'Set1', data = titanic)
ax.set(title = 'Passenger status (Survived/Died) against Passenger Class',
       xlabel = 'Passenger Class', ylabel = 'Total')
plt.show()
```



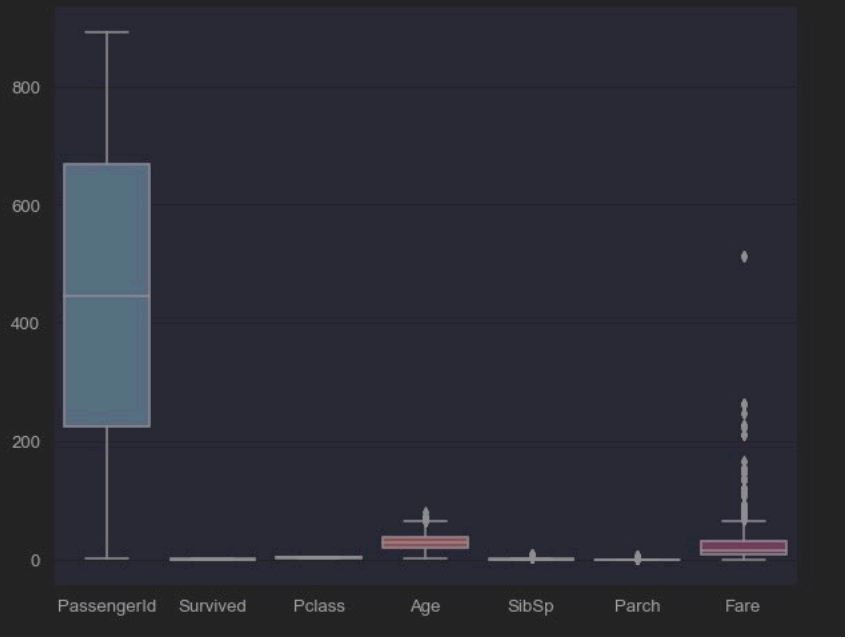
In 66
```
# We look at Age column and set Intevals on the ages and the map them to their categories as
# (Children, Teen, Adult, Old)
plt.figure(figsize=(15,10))
interval = (0,18,35,60,120)
categories = ['Children','Teens','Adult', 'Old']
titanic['Age_cats'] = pd.cut(titanic.Age, interval, labels = categories)
ax = sns.countplot(x = 'Age_cats', data = titanic, hue = 'Survived', palette = 'Set1')
ax.set(xlabel='Age Categorical', ylabel='Total',
       title="Age Categorical Survival Distribution")
plt.show()
```



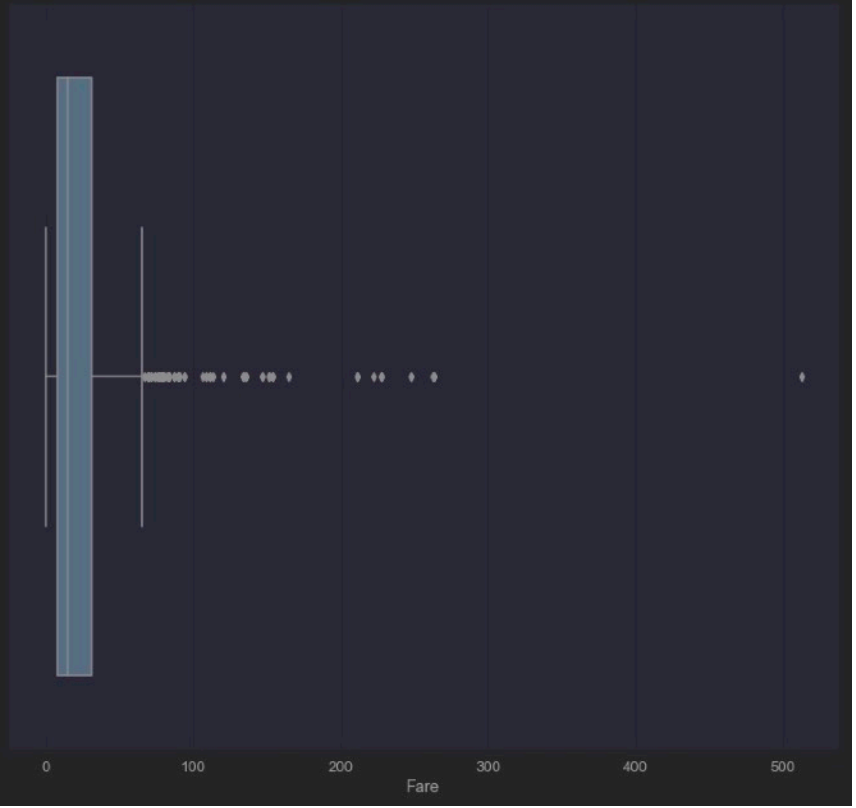


https://towardsdatascience.com/creating-boxplots-of-well-log-data-using-matplotlib-in-python-34c3816e73f4

```
In 69  1  sns.set(font_scale = 1.2)
       2  plt.figure(figsize=(10,8))
       3  ax = sns.boxplot(data=titanic, orient="v")
```

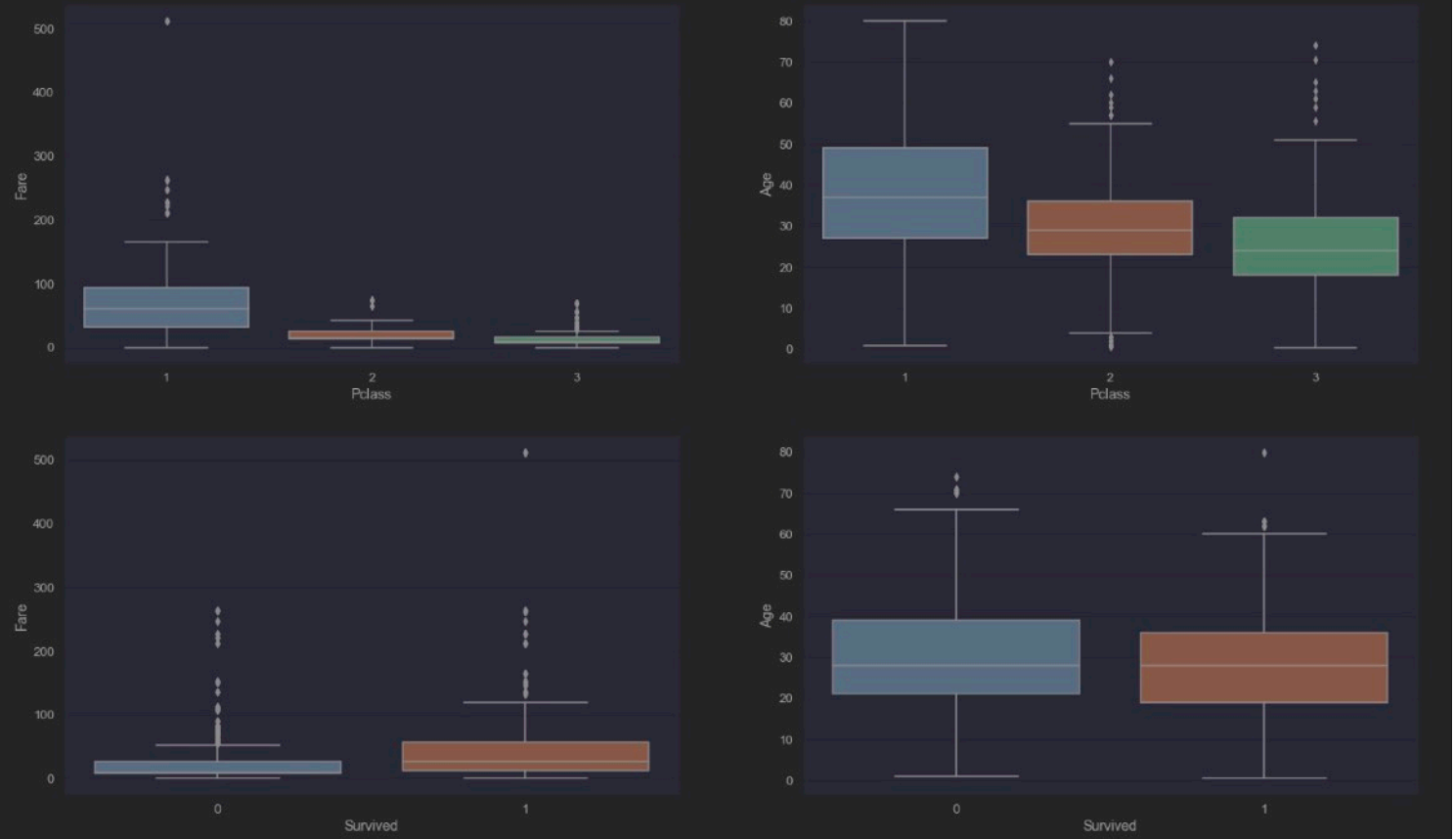


```
In 91  1  plt.figure(figsize=(13,12))
       2  sns.boxplot(data=titanic, x='Fare')

Out 91    <AxesSubplot:xlabel='Fare'>
```



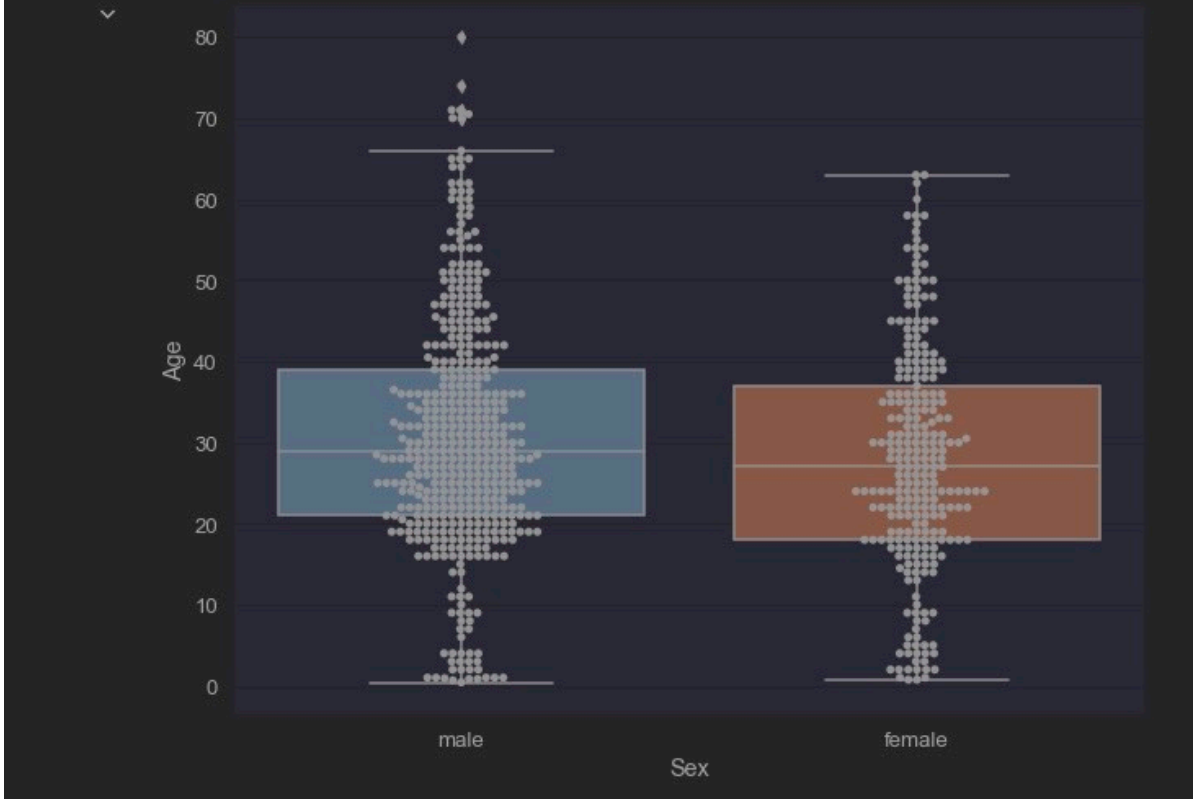
In 88
```
fig, axes = plt.subplots(2, 2, figsize=(25, 15))
sns.boxplot(ax=axes[0, 0], data=titanic, x='Pclass', y='Fare')
sns.boxplot(ax=axes[0, 1], data=titanic, x='Pclass', y='Age')
sns.boxplot(ax=axes[1, 0], data=titanic, x='Survived', y='Fare')
sns.boxplot(ax=axes[1, 1], data=titanic, x='Survived', y='Age')
```

Out 88
```
<AxesSubplot:xlabel='Survived', ylabel='Age'>
```



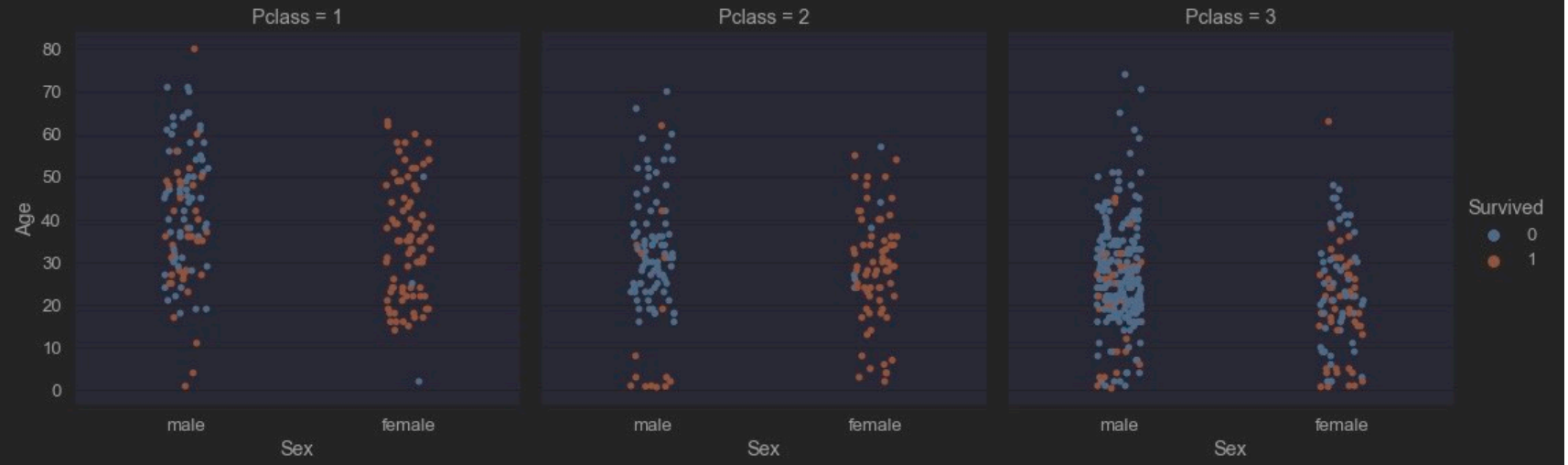
In 71

```
1 sns.set(font_scale = 1.2)
2 plt.figure(figsize=(10,8))
3 ax = sns.boxplot(x='Sex', y="Age",data=titanic, orient="v")
4 ax = sns.swarmplot(x='Sex', y="Age", data=titanic, color=".25")
```



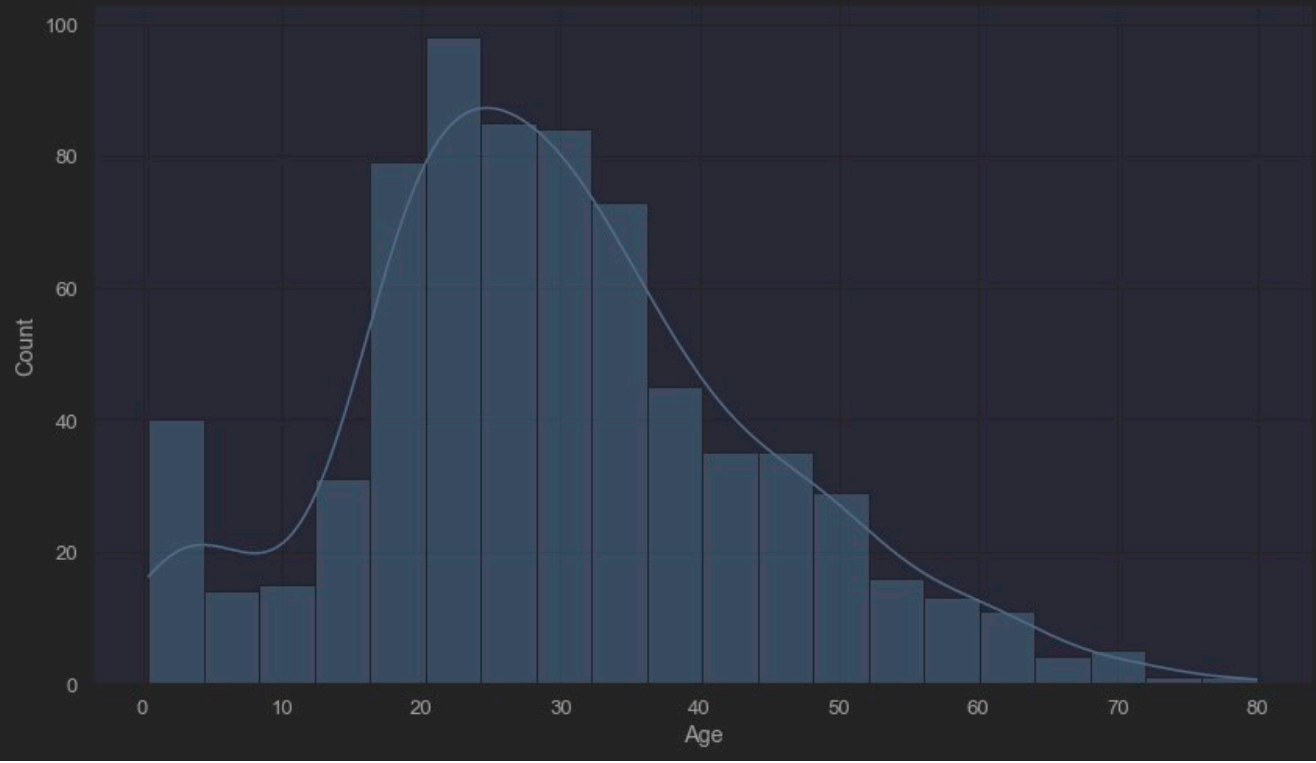
In 74
```
1 sns.set(font_scale = 1.2)
2 plt.figure(figsize=(13,12))
3 ax = sns.catplot(x='Sex', y="Age", hue="Survived", col="Pclass", data=titanic, orient="v")
```

```
<Figure size 936x864 with 0 Axes>
```



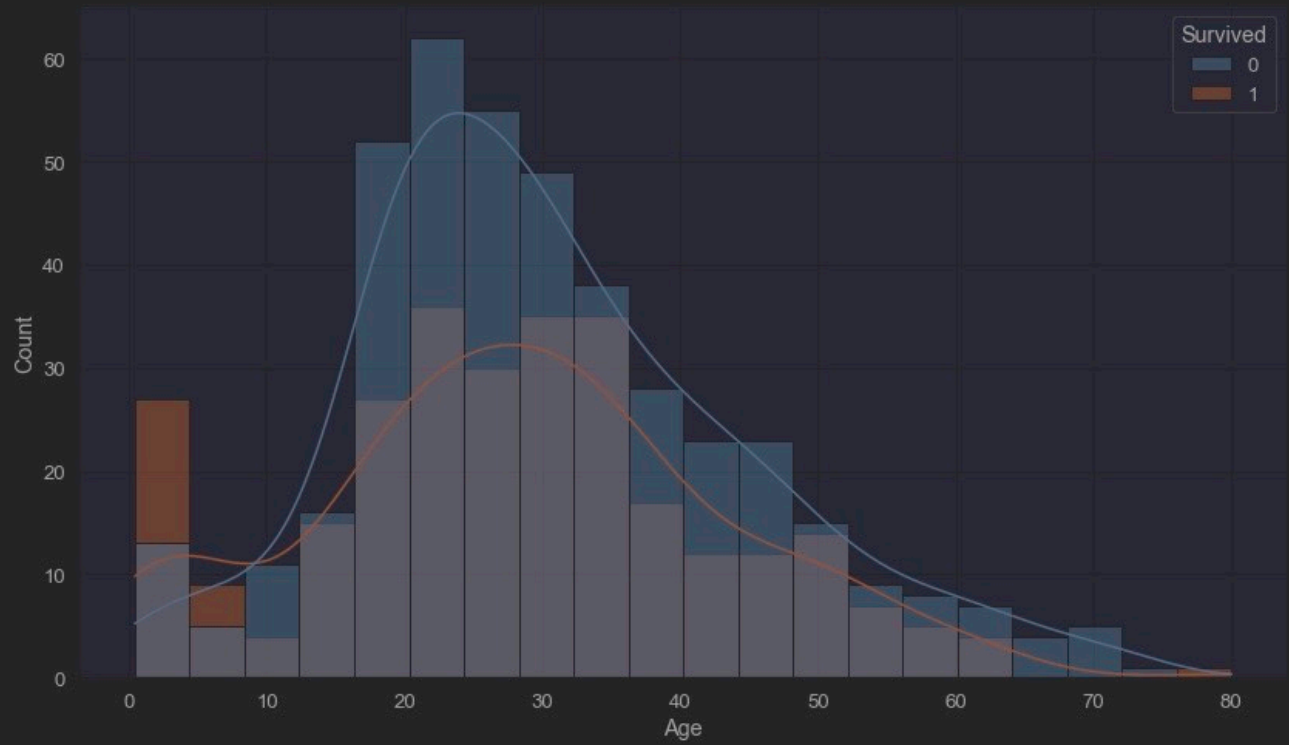
```
In 96   1  plt.figure(figsize=(14,8))
        2  sns.histplot(data = titanic, x="Age", kde=True )
```

```
Out 96  <AxesSubplot:xlabel='Age', ylabel='Count'>
```



```
In 97   1  plt.figure(figsize=(14,8))
        2  sns.histplot(data = titanic, hue="Survived", x="Age", kde=True )
```

```
Out 97  <AxesSubplot:xlabel='Age', ylabel='Count'>
```



Camemberts

```
In 177  1  titanic.Pclass.value_counts()
```

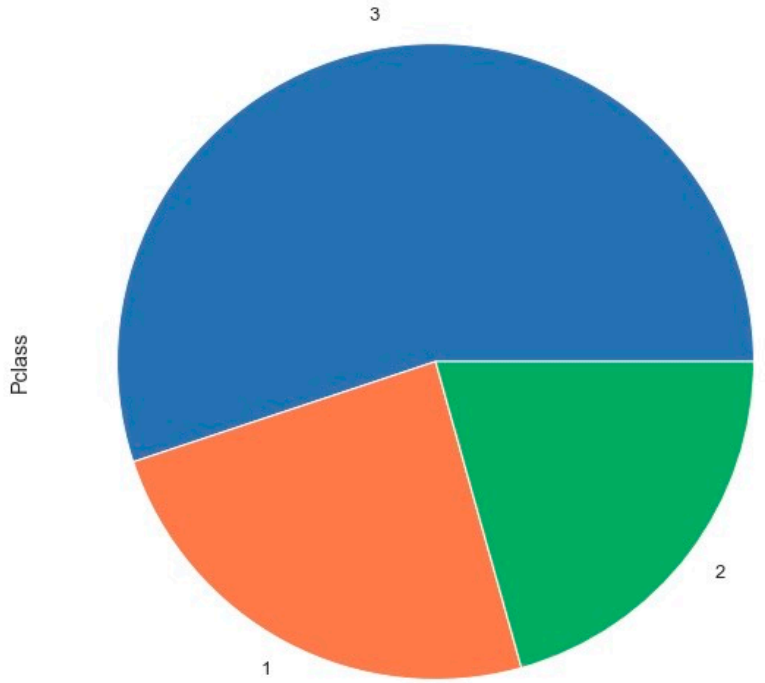
Out 177

	Pclass
3	491
1	216
2	184

Length: 3, dtype: int64 Open in new tab

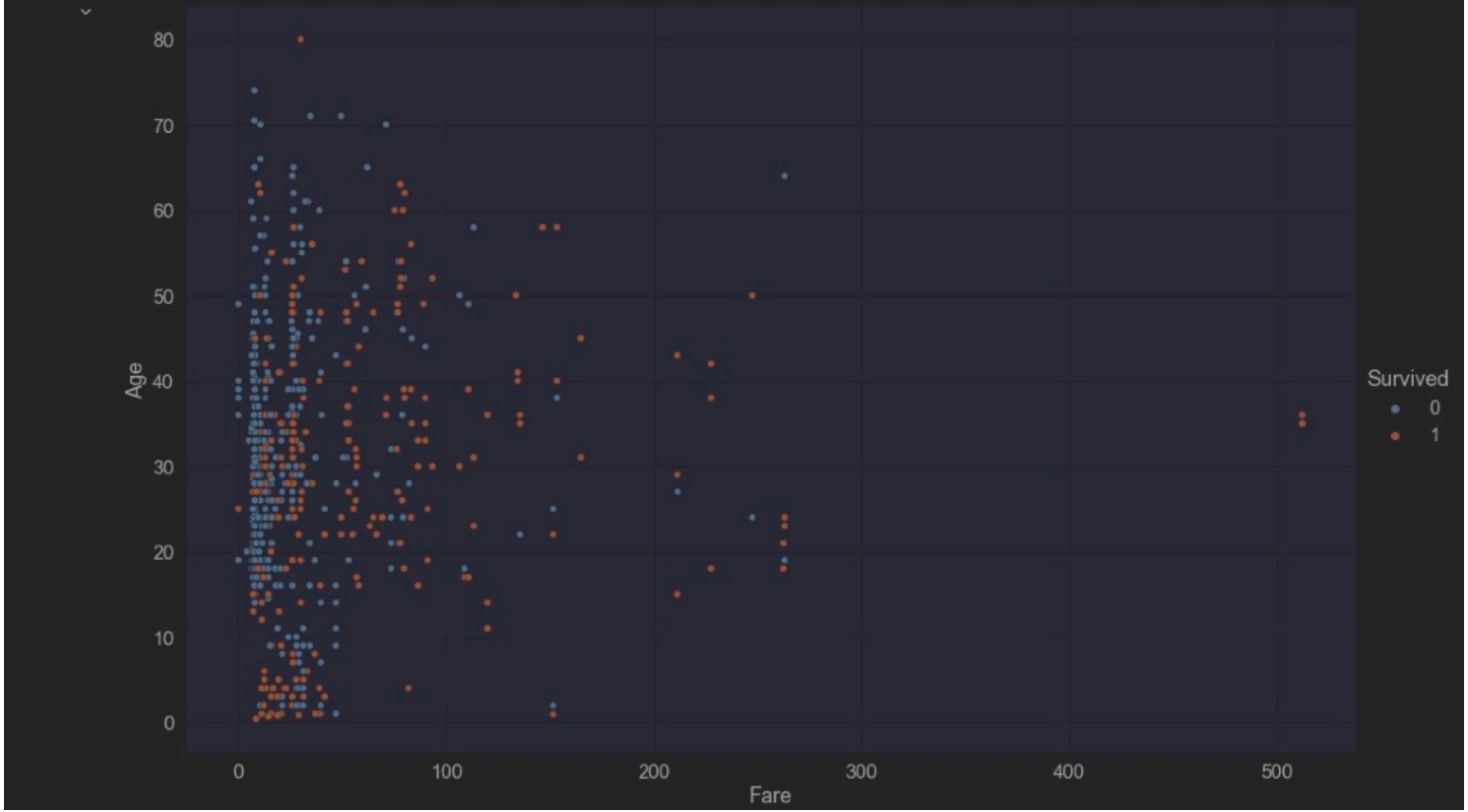
```
plt.figure(figsize=(15,10))
titanic.Pclass.value_counts().plot(kind='pie')
```

```
<AxesSubplot:ylabel='Pclass'>
```



```
In 25  1  sns.set(font_scale = 1.5)
       2  sns.relplot(x='Fare', y='Age', data=titanic, kind='scatter', hue='Survived', height=10, aspect=1.5)
```

```
Out 25  <seaborn.axisgrid.FacetGrid at 0x1f3a5d310>
```



Corrélation entre variables

```
corr = titanic.corr()
corr
```

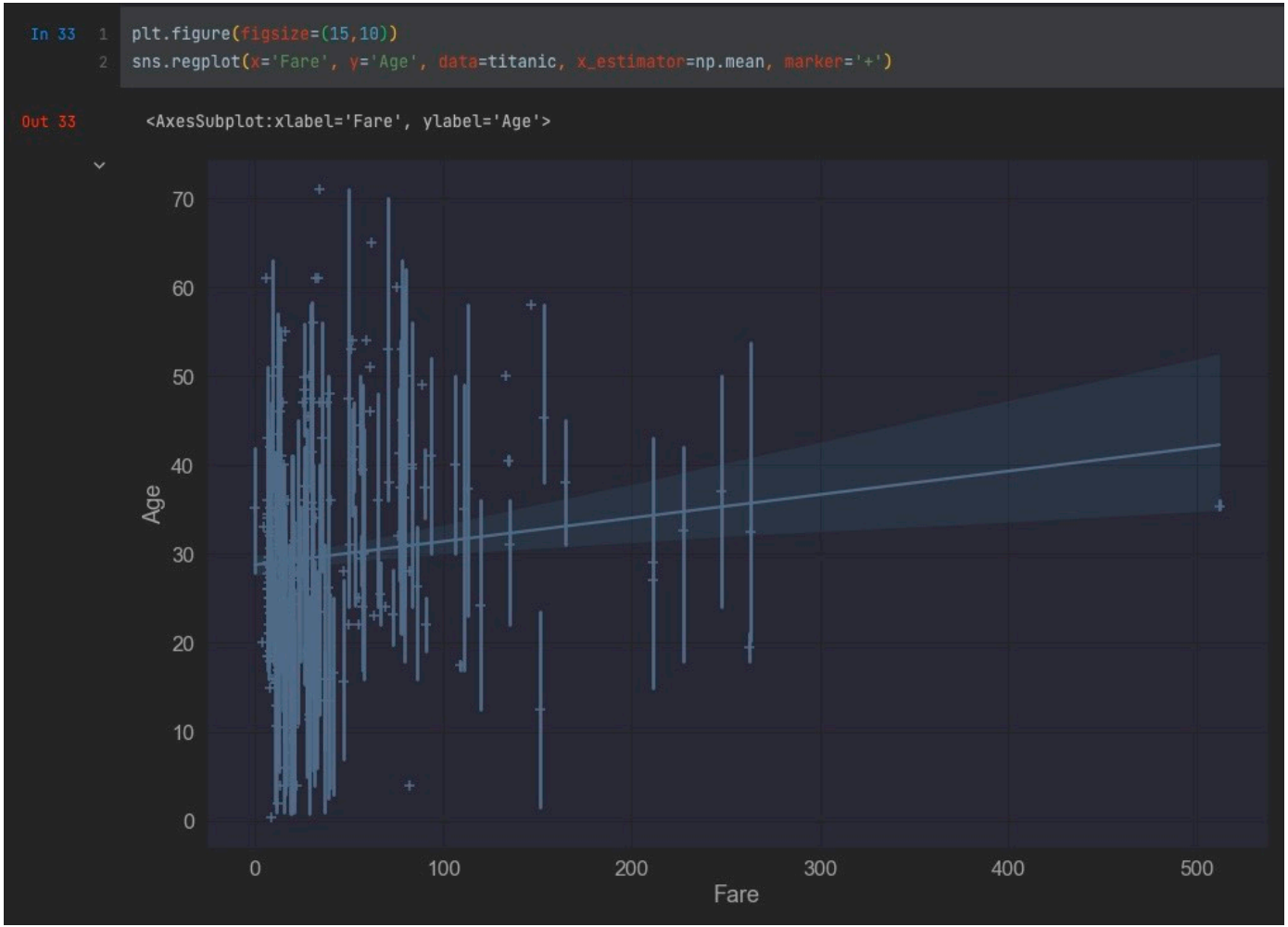
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Fare_round
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658	0.012671
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	0.257322
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	-0.549542
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	0.096076
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	0.159736
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	0.216290
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	1.000000
Fare_round	0.012671	0.257322	-0.549542	0.096076	0.159736	0.216290	1.000000	1.000000

```
plt.xticks(rotation=45)
sns.heatmap(corr,cmap="YlGnBu",annot = True)
```

```
<AxesSubplot:>
```

In 33
```
1 plt.figure(figsize=(15,10))
2 sns.regplot(x='Fare', y='Age', data=titanic, x_estimator=np.mean, marker='+')
```

Out 33
```
<AxesSubplot:xlabel='Fare', ylabel='Age'>
```

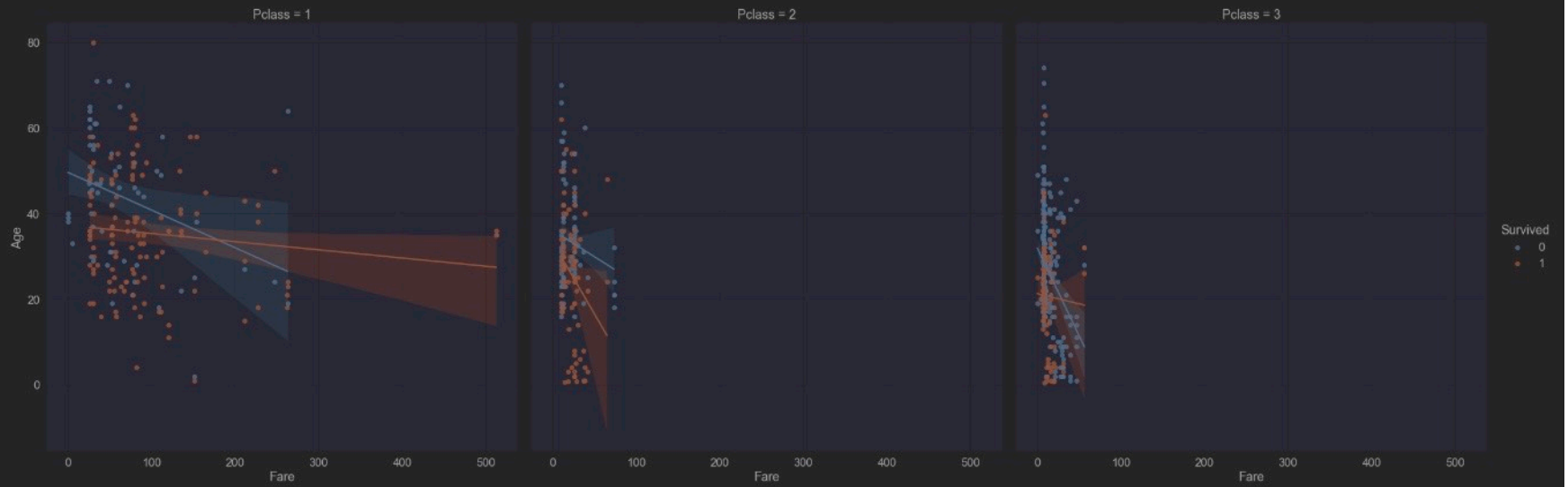


```
In 45  1  sns.lmplot(x='Fare', y='Age', hue="Survived", data=titanic, height=10)

Out 45    <seaborn.axisgrid.FacetGrid at 0x1f4a8ec40>
```

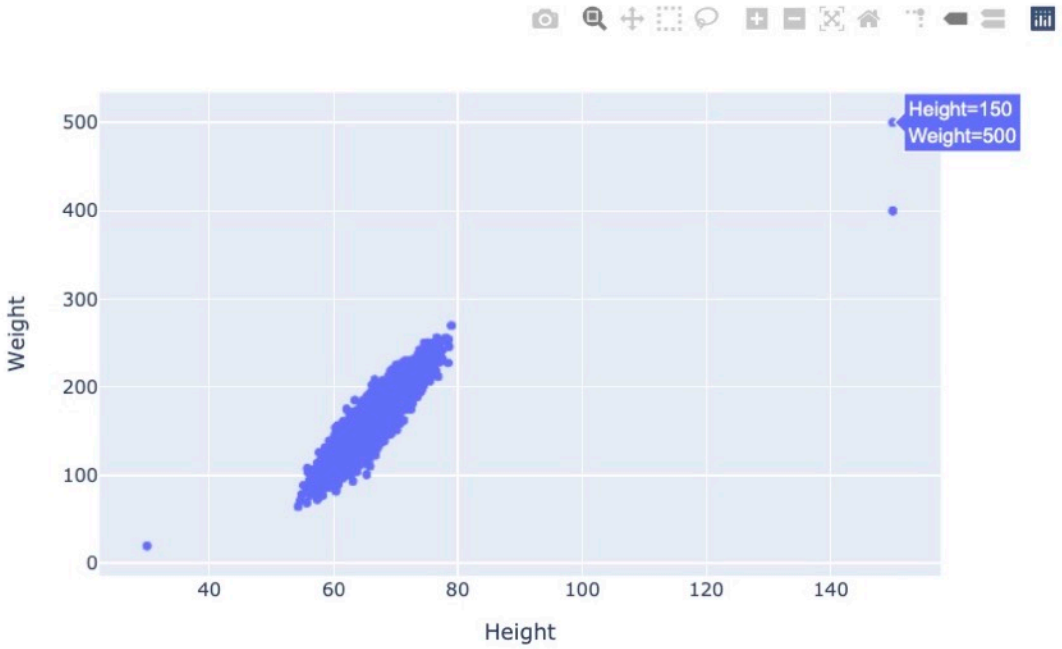
avec facettes

```
In 48  1  sns.set(font_scale = 1.5)
       2  sns.lmplot(x='Fare', y='Age', hue="Survived", col='Pclass', data=titanic, height=10)
```

```
Out 48  <seaborn.axisgrid.FacetGrid at 0x1f46e7580>
```



```
import plotly.express as px
fig = px.scatter(wh, x='Height',y='Weight')
fig.show()
```



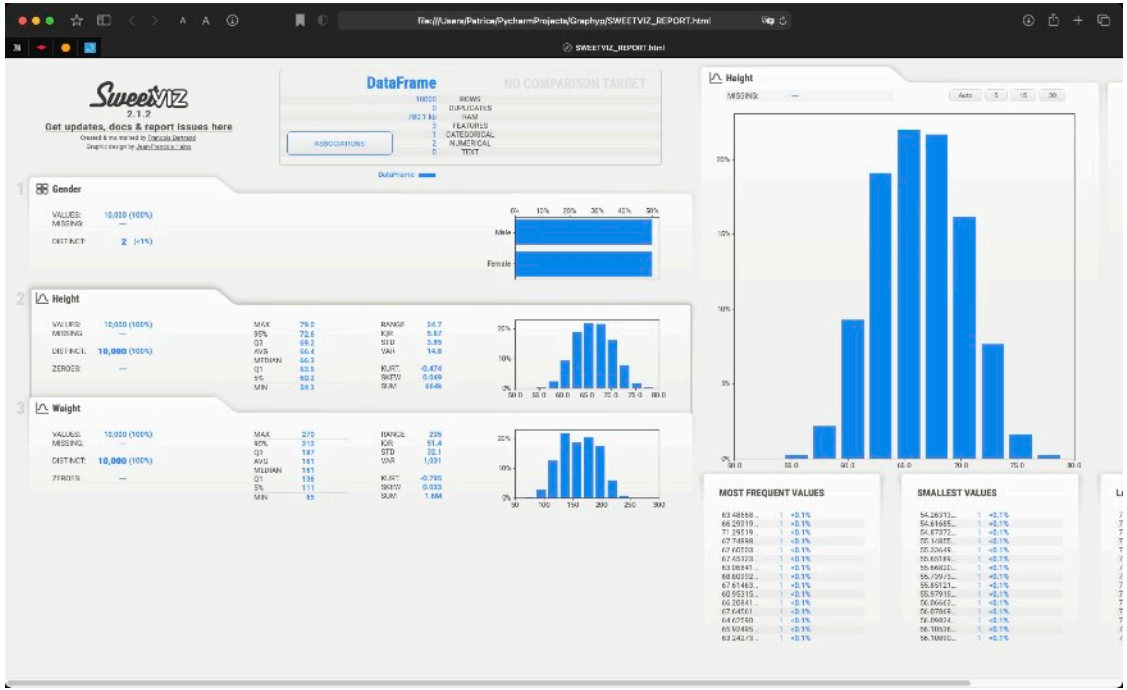
https://medium.com/spatial-data-science/4-tools-to-speed-up-exploratory-data-analysis-eda-in-python-e240ebcd18de

Des modules « tout prêts »

https://pypi.org/project/sweetviz/

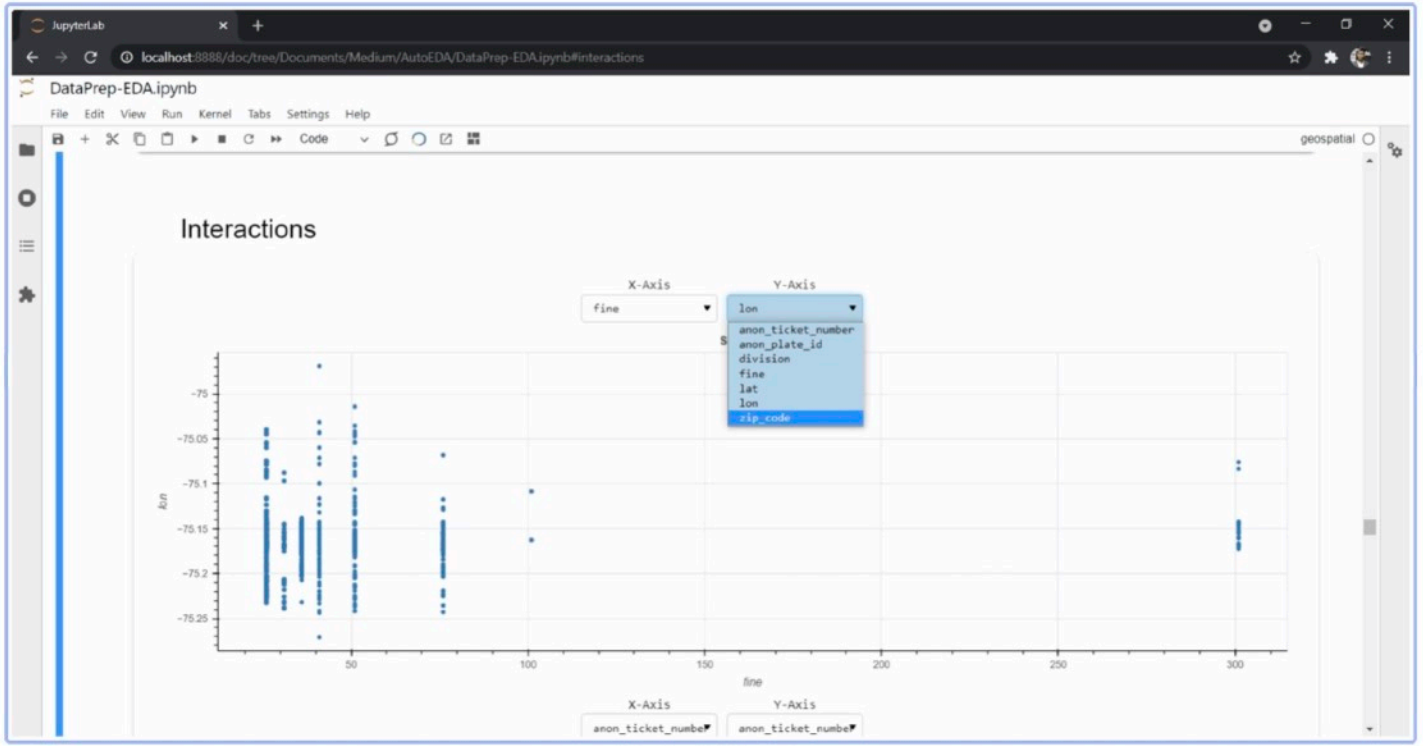
SweetViz

```
import sweetviz as sv

my_report = sv.analyze(df)
my_report.show_html()
```



dataprep

```
from dataprep.eda import create_report

create_report(df)
```

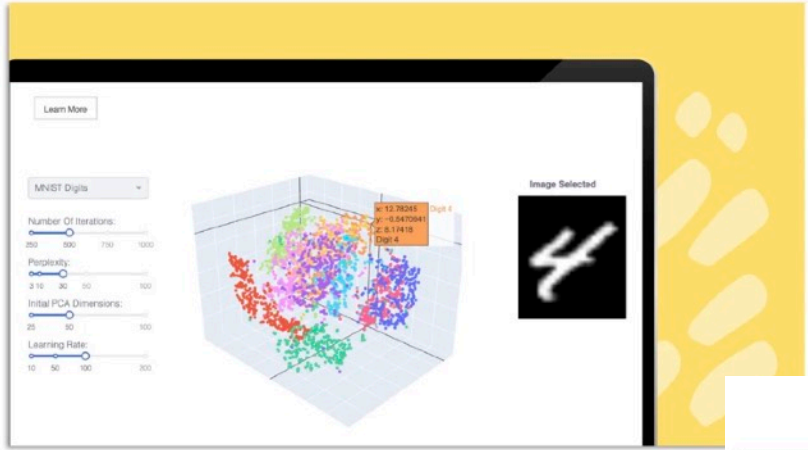


DataPrep EDS report.

Aix*Marseille université
Socialement engagée

https://plotly.com

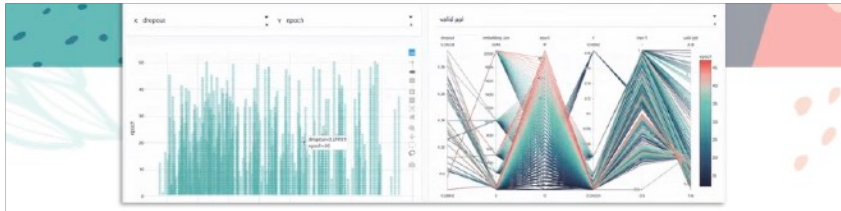
ML & AI apps made with Dash



Clustering

Dash is the fastest way to dep
apps for dimensionality redu

This Dash app demos TSNE
lines of Python code



Dash Enterprise is the low-code platform for ML & data science apps.

With Dash Enterprise, full-stack AI applications that used to require a team of front-end, back-end, and DevOps engineers can now be built, deployed, and hyperscaled by a single data scientist within hours.

plotly

Plotly Open Source Graphing Libraries

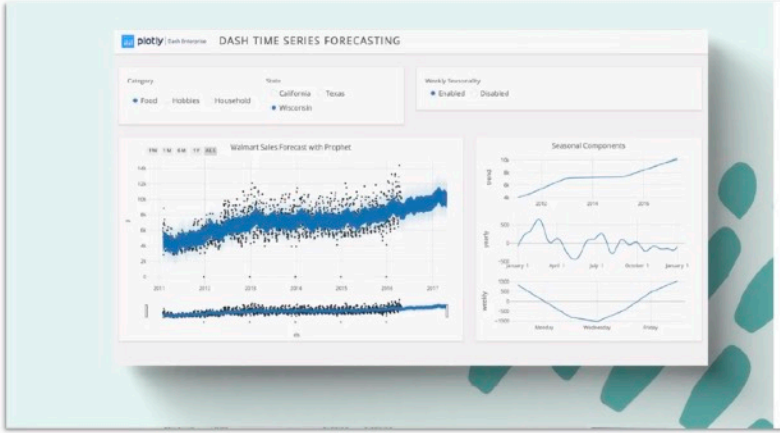
Interactive charts and maps for Python, R, and JavaScript.

Plotly Python Open Source Graphing Library

Plotly R Open Source Graphing Library

Plotly JavaScript Open Source Graphing Library

ML & AI apps made with Dash



Predictive Analytics

Dash is the fastest way to deploy Python-based apps for predictive analytics and forecasting.

Use predictive analytics + Dash to put neural networks, nonlinear regressions, decision trees, SVMs, and other forecasting methods in the hands of business users.

This Dash app demos Facebook's Prophet library in ~200 lines of Python code

Aix*Marseille université
Socialement engagée

Données temporelles

Le module de base *datetime*

```
import datetime

x = datetime.datetime.now()
print(x)
```

```
2021-06-02 11:03:29.693092
```

```
print(x.year)
print(x.month)
print(x.day)
print(x.strftime("%B"))
```

```
2021
6
2
June
```

```
x = datetime.datetime(2020, 5, 17)
```

Directive	Description
%a	Weekday, short version
%A	Weekday, full version
%w	Weekday as a number 0-6, 0 is Sunday
%d	Day of month 01-31
%b	Month name, short version
%B	Month name, full version
%m	Month as a number 01-12
%y	Year, short version, without century
%Y	Year, full version
%H	Hour 00-23
%I	Hour 00-12
%p	AM/PM
%M	Minute 00-59
%S	Second 00-59
%f	Microsecond 000000-999999
%z	UTC offset
%Z	Timezone
%j	Day number of year 001-366
%U	Week number of year, Sunday as the first day of week, 00-53
%W	Week number of year, Monday as the first day of week, 00-53

%c	Local version of date and time
%x	Local version of date
%X	Local version of time
%%	A % character
%G	ISO 8601 year
%u	ISO 8601 weekday (1-7)
%V	ISO 8601 weeknumber (01-53)

Les modules NumPy *datetime64* et *timedelta*

```
import numpy as np
np.datetime64('2005-02-25')
```

Example

All the dates for one month:

```
>>> np.arange('2005-02', '2005-03', dtype='datetime64[D]')
array(['2005-02-01', '2005-02-02', '2005-02-03', '2005-02-04',
       '2005-02-05', '2005-02-06', '2005-02-07', '2005-02-08',
       '2005-02-09', '2005-02-10', '2005-02-11', '2005-02-12',
       '2005-02-13', '2005-02-14', '2005-02-15', '2005-02-16',
       '2005-02-17', '2005-02-18', '2005-02-19', '2005-02-20',
       '2005-02-21', '2005-02-22', '2005-02-23', '2005-02-24',
       '2005-02-25', '2005-02-26', '2005-02-27', '2005-02-28'],
      dtype='datetime64[D]')
```

Example

```
>>> np.datetime64('2009-01-01') - np.datetime64('2008-01-01')
numpy.timedelta64(366,'D')
```

```
>>> np.datetime64('2009') + np.timedelta64(20, 'D')
numpy.datetime64('2009-01-21')
```

```
>>> np.datetime64('2011-06-15T00:00') + np.timedelta64(12, 'h')
numpy.datetime64('2011-06-15T12:00')
```

Pandas et *datetime*

```
dti = pd.to_datetime(["1/1/2018", np.datetime64("2018-01-01"),
datetime.datetime(2018, 1, 1)])
```

pandas.to_datetime

pandas.to_datetime(*arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True*) [source]

Convert argument to datetime.

Parameters: **arg** : *int, float, str, datetime, list, tuple, 1-d array, Series, DataFrame/dict-like*

The object to convert to a datetime.

errors : *{'ignore', 'raise', 'coerce'}, default 'raise'*

- If 'raise', then invalid parsing will raise an exception.
- If 'coerce', then invalid parsing will be set as NaT.
- If 'ignore', then invalid parsing will return the input.

dayfirst : *bool, default False*

Specify a date parse order if *arg* is str or its list-likes. If True, parses dates with the day first, eg 10/11/12 is parsed as 2012-11-10. Warning: dayfirst=True is not strict, but will prefer to parse with day first (this is a known bug, based on dateutil behavior).

yearfirst : *bool, default False*

Specify a date parse order if *arg* is str or its list-likes.

- If True parses dates with the year first, eg 10/11/12 is parsed as 2010-11-12.
- If both dayfirst and yearfirst are True, yearfirst is preceded (same as dateutil).

Warning: yearfirst=True is not strict, but will prefer to parse with year first (this is a known bug, based on dateutil behavior).

utc : *bool, default None*

Return UTC DatetimeIndex if True (converting any tz-aware datetime.datetime objects as well).

format : *str, default None*

The strftime to parse time, eg "%d/%m/%Y", note that "%f" will parse all the way up to nanoseconds. See strftime documentation for more information on choices: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior.

exact : *bool, True by default*

Behaves as: - If True, require an exact format match. - If False, allow the format to match anywhere in the target string.

unit : *str, default 'ns'*

The unit of the arg (D,s,ms,us,ns) denote the unit, which is an integer or float number. This will be based off the origin. Example, with unit='ms' and origin='unix' (the default), this would calculate the number of milliseconds to the unix epoch start.

infer_datetime_format : *bool, default False*

If True and no *format* is given, attempt to infer the format of the datetime strings based on the first non-NaN element, 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.

origin : *scalar, default 'unix'*

Define the reference date. The numeric values would be parsed as number of units (defined by *unit*) since this reference date.

- If 'unix' (or POSIX) time; origin is set to 1970-01-01.
- If 'julian', unit must be 'D', and origin is set to beginning of Julian Calendar. Julian day number 0 is assigned to the day starting at noon on January 1, 4713 BC.
- If Timestamp convertible, origin is set to Timestamp identified by origin.

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

Lorsque la date est sur plusieurs colonnes

	year	month	day
0	2015	2	4
1	2016	3	5

```
pd.to_datetime(df)
```

```
0   2015-02-04
1   2016-03-05
```

	year	month	day	temp
0	2015	2	4	15
1	2016	3	5	20

```
pd.to_datetime(df[['year','month','day']])
```

```
df['date'] =
pd.to_datetime(df[['year', 'month', 'day']])
```

	year	month	day	temp	date
0	2015	2	4	15	2015-02-04
1	2016	3	5	20	2016-03-05

```
df.set_index('date')
```

```
df.set_index('date', inplace=True)
```

	year	month	day	temp
date				
2015-02-04	2015	2	4	15
2016-03-05	2016	3	5	20

```
df.drop(['year', 'month', 'day'],axis=1, inplace=True)
```

	temp
date	
2015-02-04	15
2016-03-05	20

Manipulating and converting date times with timezone information

```
In [6]: dti = dti.tz_localize("UTC")

In [7]: dti
Out[7]:
DatetimeIndex(['2018-01-01 00:00:00+00:00', '2018-01-01 01:00:00+00:00',
               '2018-01-01 02:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='H')

In [8]: dti.tz_convert("US/Pacific")
Out[8]:
DatetimeIndex(['2017-12-31 16:00:00-08:00', '2017-12-31 17:00:00-08:00',
               '2017-12-31 18:00:00-08:00'],
              dtype='datetime64[ns, US/Pacific]', freq='H')
```

Resampling or converting a time series to a particular frequency

```
In [9]: idx = pd.date_range("2018-01-01", periods=5, freq="H")

In [10]: ts = pd.Series(range(len(idx)), index=idx)

In [11]: ts
Out[11]:
2018-01-01 00:00:00    0
2018-01-01 01:00:00    1
2018-01-01 02:00:00    2
2018-01-01 03:00:00    3
2018-01-01 04:00:00    4
Freq: H, dtype: int64

In [12]: ts.resample("2H").mean()
Out[12]:
2018-01-01 00:00:00    0.5
2018-01-01 02:00:00    2.5
2018-01-01 04:00:00    4.0
Freq: 2H, dtype: float64
```

If you use dates which start with the day first (i.e. European style), you can pass the `dayfirst` flag:

```
In [45]: pd.to_datetime(["04-01-2012 10:00"], dayfirst=True)
Out[45]: DatetimeIndex(['2012-01-04 10:00:00'], dtype='datetime64[ns]', freq=None)

In [46]: pd.to_datetime(["14-01-2012", "01-14-2012"], dayfirst=True)
Out[46]: DatetimeIndex(['2012-01-14', '2012-01-14'], dtype='datetime64[ns]', freq=None)
```

Warning

You see in the above example that `dayfirst` isn't strict, so if a date can't be parsed with the day being first it will be parsed as if `dayfirst` were False.

Invalid data

The default behavior, `errors='raise'`, is to raise when unparsable:

```
In [2]: pd.to_datetime(['2009/07/31', 'asd'], errors='raise')
ValueError: Unknown string format
```

Pass `errors='ignore'` to return the original input when unparsable:

```
In [56]: pd.to_datetime(["2009/07/31", "asd"], errors="ignore")
Out[56]: Index(['2009/07/31', 'asd'], dtype='object')
```

Pass `errors='coerce'` to convert unparsable data to `NaT` (not a time):

```
In [57]: pd.to_datetime(["2009/07/31", "asd"], errors="coerce")
Out[57]: DatetimeIndex(['2009-07-31', 'NaT'], dtype='datetime64[ns]', freq=None)
```

Des formats de dates spécifiques

```
df = pd.DataFrame({'date': ['2016-6-10 20:30:0',
                            '2016-7-1 19:45:30',
                            '2013-10-12 4:5:1'],
                   'value': [2, 3, 4]})

df['date'] = pd.to_datetime(df['date'], format="%Y-%d-%m %H:%M:%S")
df
```

	date	value
0	2016-10-06 20:30:00	2
1	2016-01-07 19:45:30	3
2	2013-12-10 04:05:01	4

Ajouter une colonne « âge »

5. Get the age from the date of birth

The simplest solution to get age is by subtracting year:

```
today = pd.to_datetime('today')
df['age'] = today.year - df['DoB'].dt.year

df
```

	name	DoB	year	month	day	week_of_year	day_of_week	is_leap_year	day_of_week_name	age
0	Tom	1997-08-05	1997	8	5	32	1	False	Tuesday	23
1	Andy	1996-04-28	1996	4	28	17	6	True	Sunday	24
2	Lucas	1995-12-16	1995	12	16	50	5	False	Saturday	25

https://towardsdatascience.com/working-with-datetime-in-pandas-dataframe-663f7af6c587

However, this is not accurate as people might haven't had their birthday this year. A more accurate solution would be to consider the birthday

```
# Year difference
today = pd.to_datetime('today')
diff_y = today.year - df['DoB'].dt.year
# Haven't had birthday
b_md = df['DoB'].apply(lambda x: (x.month,x.day) )
no_birthday = b_md > (today.month,today.day)

df['age'] = diff_y - no_birthday
df
```

	name	DoB	year	month	day	week_of_year	day_of_week	is_leap_year	day_of_week_name	age
0	Tom	1997-08-05	1997	8	5	32	1	False	Tuesday	23
1	Andy	1996-04-28	1996	4	28	17	6	True	Sunday	24
2	Lucas	1995-12-16	1995	12	16	50	5	False	Saturday	24

https://towardsdatascience.com/working-with-datetime-in-pandas-dataframe-663f7af6c587

Sélectionner selon l'année

```
df.loc['2018']
```

date	num	city
2018-01-01 09:00:00	2	London
2018-01-01 09:01:00	1	London
2018-01-01 09:02:00	3	London
2018-01-01 09:03:00	3	London
2018-01-01 09:04:00	3	London
...	...	...
2018-12-31 15:56:00	4	Cambridge
2018-12-31 15:57:00	2	Cambridge
2018-12-31 15:58:00	3	Cambridge
2018-12-31 15:59:00	3	Cambridge
2018-12-31 16:00:00	2	Cambridge

439524 rows × 2 columns

Get the total num in 2018

```
df.loc['2018','num'].sum()

1231190
```

Get the total num for each city in 2018

```
df['2018'].groupby('city').sum()
```

city	num
Cambridge	308428
Durham	307965
London	307431
Oxford	307366

Sélectionner une période

Select data between 2016 and 2018

```
df.loc['2016' : '2018']
```

Select data between 10 and 11 o'clock on the 2nd May 2018

```
df.loc['2018-5-2 10' : '2018-5-2 11' ]
```

Select data between 10:30 and 10:45 on the 2nd May 2018

```
df.loc['2018-5-2 10:30' : '2018-5-2 10:45' ]
```

And to select data between time, we should use `between_time()`, for example, 10:30 and 10:45

```
df.between_time('10:30','10:45')
```

https://towardsdatascience.com/working-with-datetime-in-pandas-dataframe-663f7af6c587

Aix*Marseille université
Socialement engagée

Séries temporelles

DateOffsets

Alias	Description
B	business day frequency
C	custom business day frequency
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
SMS	semi-month start frequency (1st and 15th)
BMS	business month start frequency
BQS	business quarter start frequency
A, Y	year end frequency
BA, BY	business year end frequency
AS, YS	year start frequency
BAS, BYS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds

https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases

```
In [238]: pd.date_range(start, periods=5, freq="B")
Out[238]:
DatetimeIndex(['2011-01-03', '2011-01-04', '2011-01-05', '2011-01-06',
               '2011-01-07'],
              dtype='datetime64[ns]', freq='B')

In [239]: pd.date_range(start, periods=5, freq=pd.offsets.BDay())
Out[239]:
DatetimeIndex(['2011-01-03', '2011-01-04', '2011-01-05', '2011-01-06',
               '2011-01-07'],
              dtype='datetime64[ns]', freq='B')
```

You can combine together day and intraday offsets:

```
In [240]: pd.date_range(start, periods=10, freq="2h20min")
Out[240]:
DatetimeIndex(['2011-01-01 00:00:00', '2011-01-01 02:20:00',
               '2011-01-01 04:40:00', '2011-01-01 07:00:00',
               '2011-01-01 09:20:00', '2011-01-01 11:40:00',
               '2011-01-01 14:00:00', '2011-01-01 16:20:00',
               '2011-01-01 18:40:00', '2011-01-01 21:00:00'],
              dtype='datetime64[ns]', freq='140T')

In [241]: pd.date_range(start, periods=10, freq="1D10U")
Out[241]:
DatetimeIndex([       '2011-01-01 00:00:00', '2011-01-02 00:00:00.000010',
               '2011-01-03 00:00:00.000020', '2011-01-04 00:00:00.000030',
               '2011-01-05 00:00:00.000040', '2011-01-06 00:00:00.000050',
               '2011-01-07 00:00:00.000060', '2011-01-08 00:00:00.000070',
               '2011-01-09 00:00:00.000080', '2011-01-10 00:00:00.000090'],
              dtype='datetime64[ns]', freq='86400000010U')
```

Sélectionner les lignes pour un jour (n°) donné

dataframe[‘colonne’].day n’est pas possible

AttributeError: 'Series' object has no attribute 'days'

pandas.Series.dt

`Series.dt()`

Accessor object for datetimelike properties of the Series values.

```
rythme[rythme['creationDate'].dt.day==27].head()
```

```
rythme.query('creationDate.dt.day==27').head()
```

	creationDate	startDate	value
24016	2020-05-27 17:59:38+01:00	2020-05-27 17:56:03+01:00	98
24017	2020-05-27 18:05:26+01:00	2020-05-27 18:02:36+01:00	85
24018	2020-05-27 18:07:07+01:00	2020-05-27 18:07:06+01:00	85
24019	2020-05-27 18:07:07+01:00	2020-05-27 18:07:07+01:00	83
24020	2020-05-27 18:07:09+01:00	2020-05-27 18:07:08+01:00	83

Sélectionner les lignes pour un jour précis

```
mask = rythme['creationDate'].between('2022-02-27 0:00','2022-02-27 23:59')
rythme.loc[mask].head()
```

	creationDate	startDate	value
65957	2022-02-27 15:49:40+01:00	2021-04-21 15:11:51+01:00	87
65958	2022-02-27 15:52:32+01:00	2022-02-27 15:52:31+01:00	91
65959	2022-02-27 15:52:32+01:00	2022-02-27 15:52:32+01:00	90
65960	2022-02-27 15:52:34+01:00	2022-02-27 15:52:33+01:00	88
65961	2022-02-27 15:52:34+01:00	2022-02-27 15:52:34+01:00	88

mieux :

```
mask = rythme['creationDate'].between('2022-02-27','2022-02-28', inclusive='left')
```

et si date dans index :

```
df_data_minute[df_data_minute.index.get_loc("2022-02-27")]
```

```
df_data_minute.loc["2022-02-27"]
```

Aix*Marseille université
Socialement engagée

In 42
```
rythme.tail()
```

Out 42

	creationDate	startD	value
66863	2022-02-27 16:11:35+01:00	2022-...	86
66864	2022-02-27 16:11:36+01:00	2022-...	85
66865	2022-02-27 16:11:36+01:00	2022-...	85
66866	2022-02-27 16:11:38+01:00	2022-...	85
66867	2022-02-27 16:11:39+01:00	2022-...	85

5 rows × 3 columns Open in new tab

In 54
```
df_data_minute = rythme.groupby(pd.Grouper(key='creationDate', axis=0, freq='S')).mean()
df_rollmean = df_data_minute.resample(rule='T').mean()
df_rollstd = df_data_minute.resample(rule='T').std()
fig, axs = plt.subplots(1, 1, figsize=(25, 10))
axs.plot(df_data_minute['value'], color='blue', label = "Original")
axs.plot(df_rollmean['value'], color='red', label = "Mean")
axs.plot(df_rollmean['value'], color='green', label = "std")
plt.legend()
plt.show()
```

Aix*Marseille université
Socialement engagée

Ré-échantillonnage

In 1
```
1 import pandas as pd
2 from datetime import datetime
3 import numpy as np
```

In 2
```
1 # création d'un intervalle temporel
2 intervalle = pd.date_range(start='2022-05-01',end='2022-05-10',freq='H')
```

In 5
```
1 intervalle
```

Out 5
```
DatetimeIndex(['2022-05-01 00:00:00', '2022-05-01 01:00:00',
               '2022-05-01 02:00:00', '2022-05-01 03:00:00',
               '2022-05-01 04:00:00', '2022-05-01 05:00:00',
               '2022-05-01 06:00:00', '2022-05-01 07:00:00',
               '2022-05-01 08:00:00', '2022-05-01 09:00:00',
               ...
               '2022-05-09 15:00:00', '2022-05-09 16:00:00',
               '2022-05-09 17:00:00', '2022-05-09 18:00:00',
               '2022-05-09 19:00:00', '2022-05-09 20:00:00',
               '2022-05-09 21:00:00', '2022-05-09 22:00:00',
               '2022-05-09 23:00:00', '2022-05-10 00:00:00'],
              dtype='datetime64[ns]', length=217, freq='H')
```

In 6
```
1 type(intervalle[0])
```

Out 6
```
pandas._libs.tslibs.timestamps.Timestamp
```

In 9
```
#création d'un dataframe avec des valeurs aléatoires
df = pd.DataFrame(np.random.randint(0,100,size=(len(intervalle))), index = intervalle, columns=["valeur"])
```

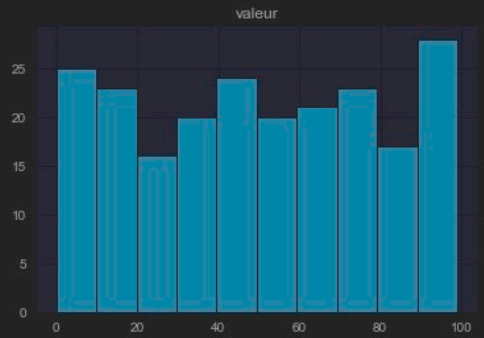
In 12
```
display(df)
```

	valeur
2022-05-01 00:00:00	7
2022-05-01 01:00:00	90
2022-05-01 02:00:00	54
2022-05-01 03:00:00	75
2022-05-01 04:00:00	22
...	...
2022-05-09 20:00:00	50
2022-05-09 21:00:00	47
2022-05-09 22:00:00	7
2022-05-09 23:00:00	7
2022-05-10 00:00:00	90

217 rows × 1 columns Open in new tab

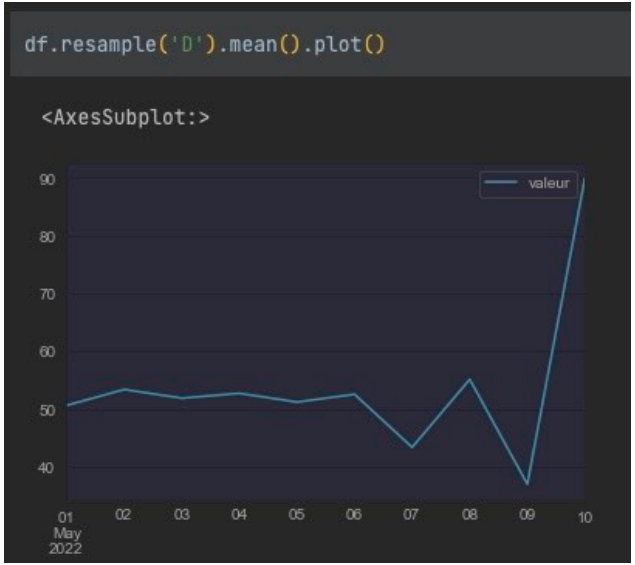
In 13
```
df.hist()
```

Out 13
```
array([[<AxesSubplot:title={'center':'valeur'}>]], dtype=object)
```



```
df.resample('D').mean().plot()
```

```
<AxesSubplot:>
```




```
In 25   1  df.resample('D').mean()
```

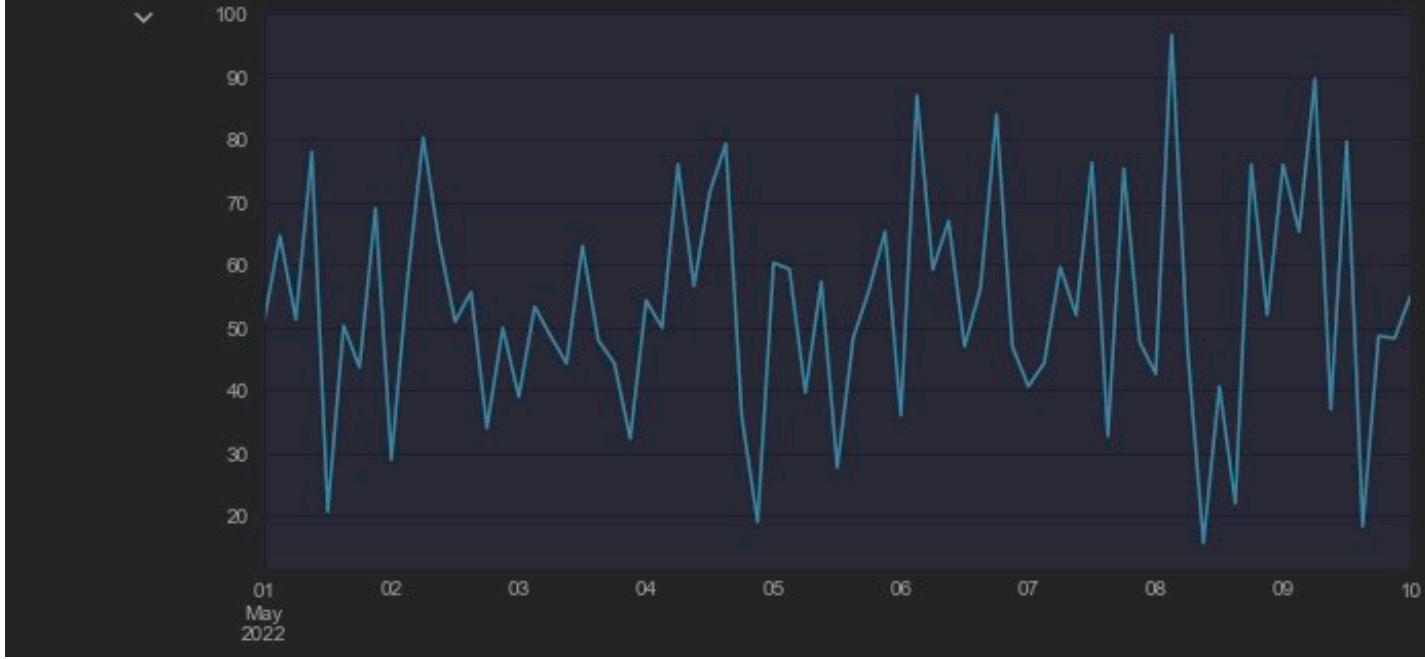
Out 25

	valeur
2022-05-01	50.625000
2022-05-02	53.375000
2022-05-03	51.875000
2022-05-04	52.708333
2022-05-05	51.208333
2022-05-06	52.541667
2022-05-07	43.375000
2022-05-08	55.125000
2022-05-09	37.000000
2022-05-10	90.000000

10 rows × 1 columns Open in new tab

```
In 52  1  df['valeur'].resample('3H').mean().plot(figsize=(10,5))
```

```
Out 52    <AxesSubplot:>
```



Somme par jour : groupby sur index

2022-05-09 20:00:00	49	146.0
2022-05-09 21:00:00	1	60.0
2022-05-09 22:00:00	77	127.0
2022-05-09 23:00:00	67	145.0
2022-05-10 00:00:00	55	199.0

```
In 48   1  df.groupby(df.index.day)['valeur'].sum()
```

Out 48

	valeur
1	1286
2	1263
3	1119
4	1329
5	1242
6	1451
7	1286
8	1177
9	1389
10	55

Synthèse par fenêtre glissante : rolling

```
df.head()
```

	valeur
2022-05-01 00:00:00	77
2022-05-01 01:00:00	60
2022-05-01 02:00:00	16
2022-05-01 03:00:00	88
2022-05-01 04:00:00	82

```
In 44  1  df['Somme par...'] = df.valeur.rolling(3).sum()
```

```
In 45  1  df
```

Out 45

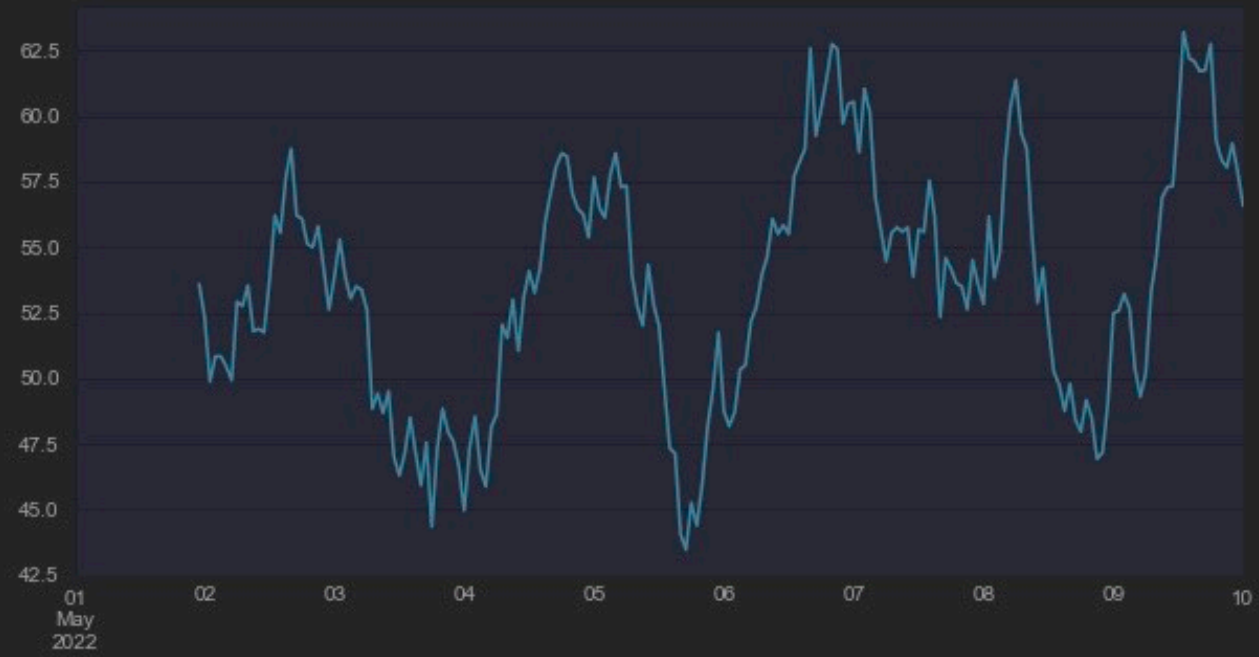
	valeur	Somme par...
2022-05-01 00:00:00	77	NaN
2022-05-01 01:00:00	60	NaN
2022-05-01 02:00:00	16	153.0
2022-05-01 03:00:00	88	164.0
2022-05-01 04:00:00	82	186.0
2022-05-01 05:00:00	24	194.0
2022-05-01 06:00:00	1	107.0
2022-05-01 07:00:00	99	124.0
2022-05-01 08:00:00	54	154.0

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html

Moyenne glissante sur 24 heures

```
In 57   1  df.valeur.rolling(24).mean().plot(figsize=(10,5))
```

```
Out 57     <AxesSubplot:>
```



Exemple de série avec tendance, cycles et saisonnalité

In 59

```
df = pd.read_csv("https://raw.githubusercontent.com/AileenNielsen/TimeSeriesAnalysisWithPython/master/data/AirPassengers.csv")
df['Month'] = pd.to_datetime(df['Month'],infer_datetime_format=True)
df = df.set_index(['Month'])
df.head(10)
```

Out 59

Month	#Passe
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121
1949-06-01	135
1949-07-01	148
1949-08-01	148
1949-09-01	136
1949-10-01	119

```
In 62  1  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 144 entries, 1949-01-01 to 1960-12-01
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   #Passengers  144 non-null    int64
dtypes: int64(1)
memory usage: 2.2 KB
```

```
In 63  1  df.describe()
```

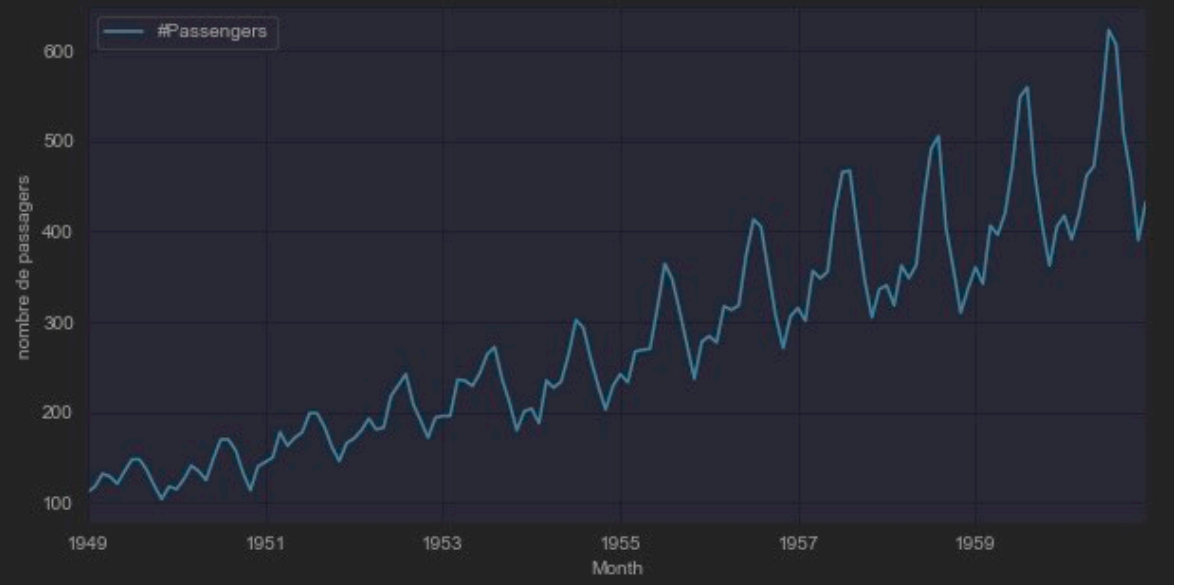
Out 63

	#Passe
count	144.0...
mean	280.2...
std	119.9...
min	104.0...
25%	180.0...
50%	265.5...
75%	360.5...
max	622.0...

8 rows × 1 columns Open in new tab

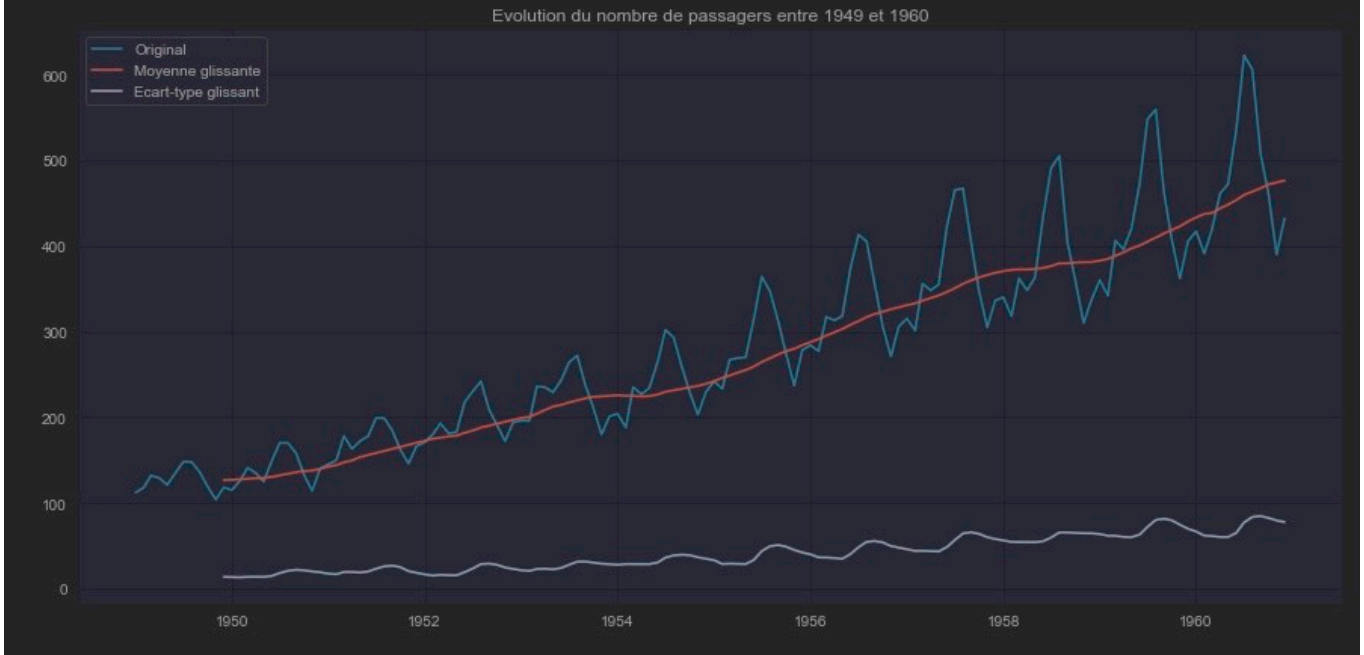
```
df.plot(figsize=(10,5), ylabel='nombre de passagers')
```

```
<AxesSubplot:xlabel='Month', ylabel='nombre de passagers'>
```



```
df["Moyenne glissante"] = df['#Passengers'].rolling(window=12).mean()
df["Ecart-type glissant"] = df['#Passengers'].rolling(window=12).std()
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,7))
plt.plot(df["#Passengers"], color='#379BDB', label='Original')
plt.plot(df["Moyenne glissante"], color='#D22A0D', label='Moyenne glissante')
plt.plot(df["Ecart-type glissant"], color='#142039', label='Ecart-type glissant')
plt.legend(loc='best')
plt.title('Evolution du nombre de passagers entre 1949 et 1960')
plt.show(block=False)
```



Stationnarité ? Non !

Augmented Dickey-Fuller test

```
from statsmodels.tsa.stattools import adfuller
import pmdarima as pm
#Augmented Dickey-Fuller test:
print('Results of Dickey Fuller Test:')
dftest = adfuller(df['#Passengers'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)
```

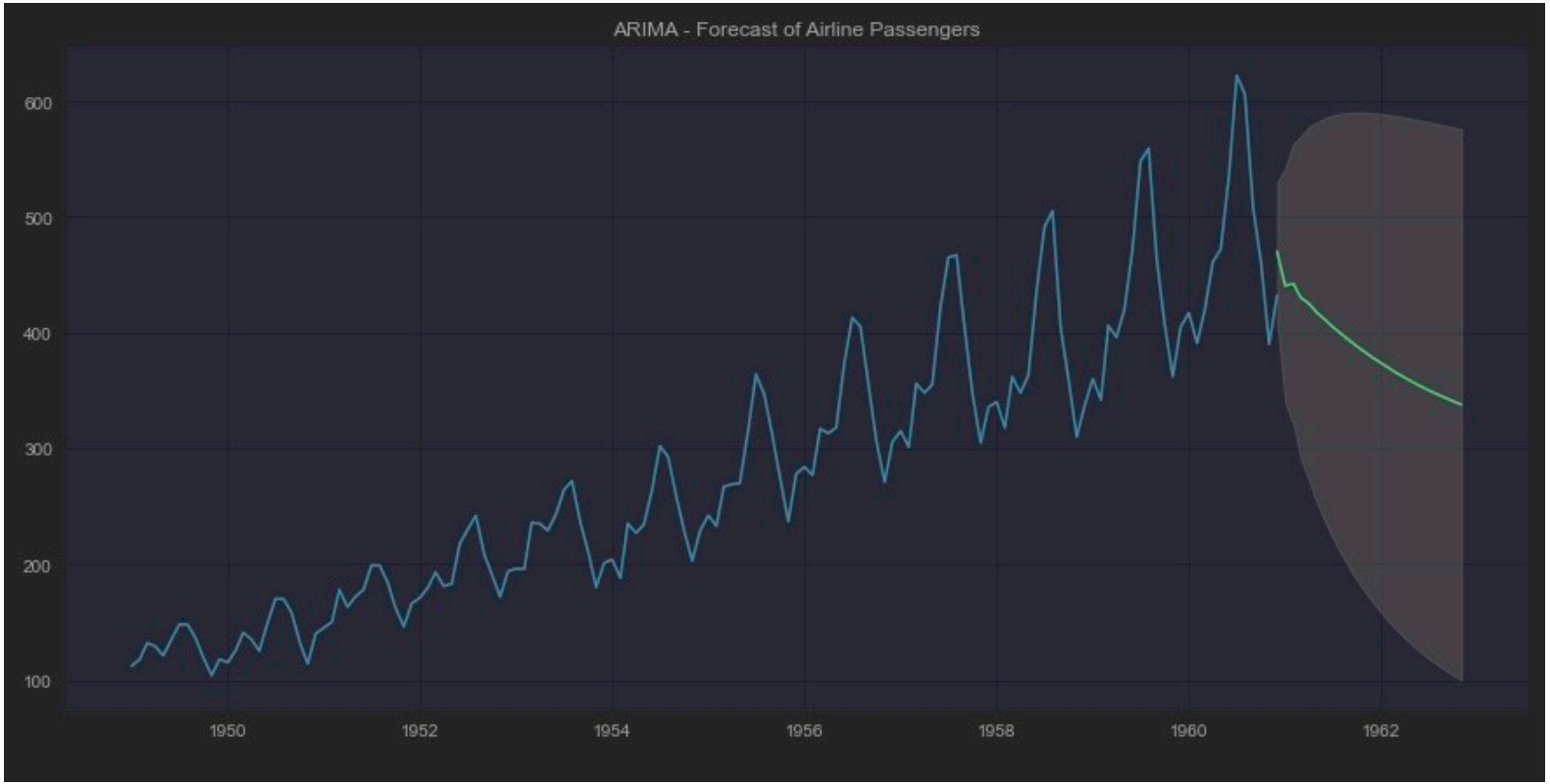
```
Results of Dickey Fuller Test:
Test Statistic                  0.815369
p-value                         0.991880
#Lags Used                     13.000000
Number of Observations Used    130.000000
```

Risque de se tromper en affirmant qu'elle est stationnaire > 99,1 %

Essai de modélisation avec ARIMA

Prise en compte des valeurs antérieures (auto-régressif)
+ des moyennes mobiles (lissage) avec suppression de la tendance générale

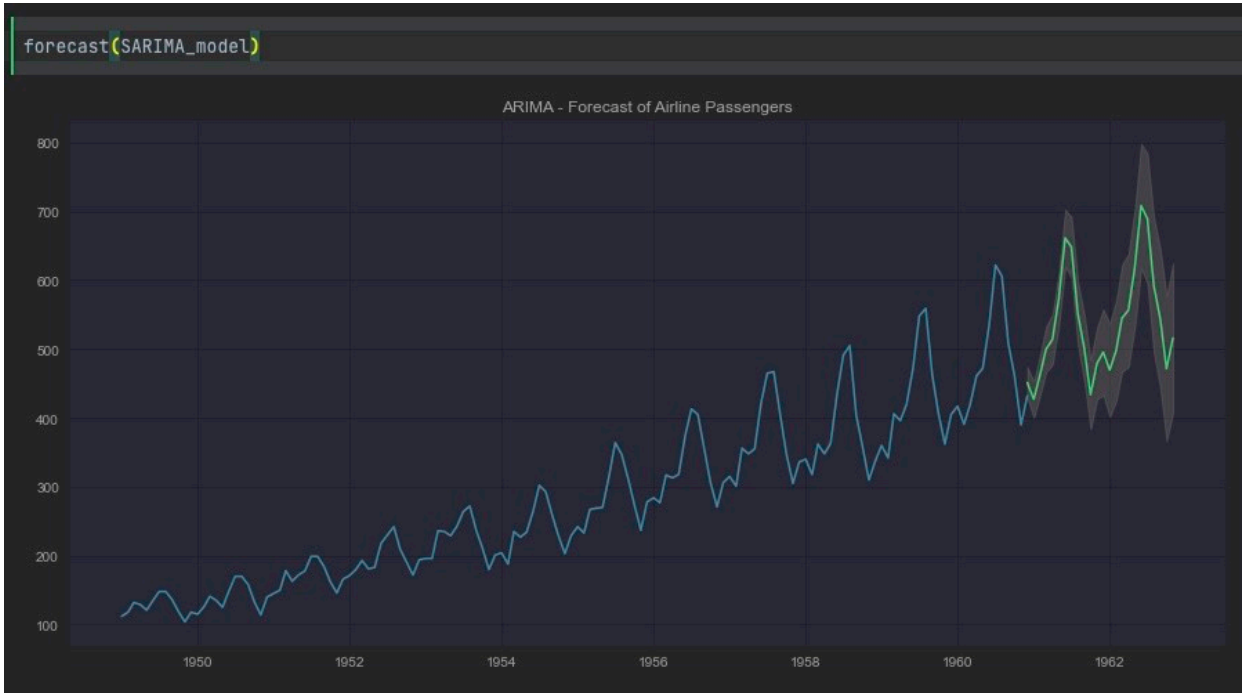
```
def forecast(ARIMA_model, periods=24):
    # Forecast
    n_periods = periods
    fitted, confint = ARIMA_model.predict(n_periods=n_periods, return_conf_int=True)
    index_of_fc = pd.date_range(df.index[-1], periods = n_periods, freq='MS')

    # make series for plotting purpose
    fitted_series = pd.Series(fitted, index=index_of_fc)
    lower_series = pd.Series(confint[:, 0], index=index_of_fc)
    upper_series = pd.Series(confint[:, 1], index=index_of_fc)

    # Plot
    plt.figure(figsize=(15,7))
    plt.plot(df["#Passengers"], color='#1f76b4')
    plt.plot(fitted_series, color='darkgreen')
    plt.fill_between(lower_series.index,
                     lower_series,
                     upper_series,
                     color='k', alpha=.15)

    plt.title("ARIMA - Forecast of Airline Passengers")
    plt.show()

forecast(ARIMA_model)
```


ARIMA - Forecast of Airline Passengers
600
500
400
300
200
100
1950
1952
1954
1956
1958
1960
1962

Essai avec SARIMA

Ajout d'une composante saisonnière

```
# Seasonal - fit stepwise auto-ARIMA
SARIMA_model = pm.auto_arima(df["#Passengers"], start_p=1, start_q=1,
                             test='adf',
                             max_p=3, max_q=3,
                             m=12, #12 is the frequncy of the cycle
                             start_P=0,
                             seasonal=True, #set to seasonal
                             d=None,
                             D=1, #order of the seasonal differencing
                             trace=False,
                             error_action='ignore',
                             suppress_warnings=True,
                             stepwise=True)
```



Aix*Marseille université
Socialement engagée

Les données textuelles

Pandas, Représentations vectorielles, NLTK…

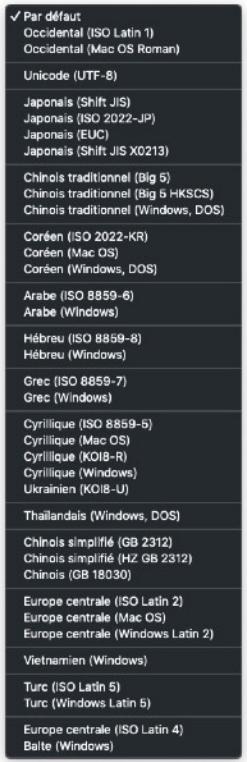
Qu’est-ce qu’un texte ?

Des caractères, des mots, des phrases, des concepts, des documents
Des références, des structures (syntaxe, discours), des rôles

Analyse lexicale, analyse syntaxique, analyse sémantique, pragmatique

Des arborescences, des sacs de mots, des représentations distribuées, des formes

L'encodage des caractères



Dec	Hex	Char	Dec	Hex	Char
64	40	@	96	60	`
65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q

7	00A8	00A9	00AA	00AB	00AC	00A
7	00B8	00B9	00BA	00BB	00BC	00B
7	00C8	00C9	00CA	00CB	00CC	00C
7	00D8	00D9	00DA	00DB	00DC	00D
7	00E8	00E9	00EA	00EB	00EC	00E
7	00F8	00F9	00FA	00FB	00FC	00F

ISO-Latin 1 sur 8 bits
(ISO/CEI 8859)

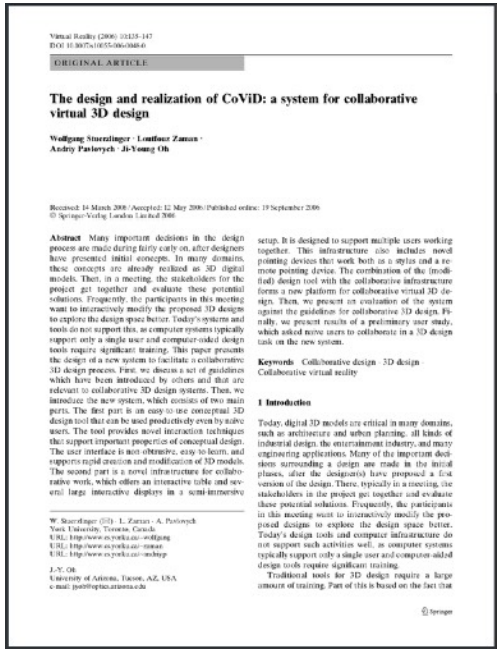
face-smiling

	№	Code	Browser	Appl	Goog	FB
940	ά	έ	ή	ί	ΰ	α	β	γ	δ	ε	😀	😀	😀
950	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο			
960	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω	😃	😃	😃
970	ϊ	ϋ	ό	ύ	ώ	ϗ	ϐ	ϑ	ϒ	ϓ			
980	ϔ	φ	ϖ	ϗ	Ϙ	ϙ	Ϛ	ϛ	Ϝ	ϝ	😄	😄	😄
990	ϟ	Ϡ	ϡ	Ϣ	ϣ	Ϥ	ϥ	Ϧ	ϧ	ϩ			
1000	Ϫ	ϫ	Ϭ	ϭ	Ϯ	ϯ	ϰ	ϱ	ϲ	ϳ			
1010	ϴ	ϵ	϶	Ϸ	ϸ	Ϲ	Ϻ	ϻ	ϼ	Ͻ	😁	😁	😁
1020	Ͼ	Ͽ	Ѐ	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї			
1030	Ј	Љ	Њ	Ћ	Ќ	Ѝ	Ў	Џ	А	Б			
1040	В	Г	Д	Е	Ж	З	И	Й	К	Л			
1050	М	Н	О	П	Р	С	Т	У	Ф	Х			
1060	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я			
1070	а	б	в	г	д	е	ж	з	и	й			
1080	к	л	м	н	о	п	р	с	т	у			
1090	ф	х	ц	ч	ш	щ	ъ	ы	ь	э			
1100	ю	я	ѐ	ё	ђ	ѓ	є	ѕ	і	ї			
1110	ј	љ	њ	ћ	ќ	ѝ	ў	џ	Ѡ	ѡ			
1120	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ					

Unicode (1988) dont UTF-8
plus de 100 000 caractères
(dont > 3 000 émojis)
de 1 jusqu'à 6 octets

https://unicode.org/emoji/

L'encodage des documents



Virtual Reality (2006) 10:135–147
DOI 10.1007/s10055-006-0048-0

ORIGINAL ARTICLE

The design and realization of CoViD: a system for collaborative virtual 3D design

Wolfgang Stuerzlinger · Loutfouz Zaman ·
Andriy Pavlovych · Ji-Young Oh

Received: 14 March 2006 / Accepted: 12 May 2006 / Published online: 19 September 2006

Abstract Many important decisions in the design process are made during fairly early on, after designers have presented initial concepts. In many domains, these concepts are already realized as 3D digital models. Then, in a meeting, the stakeholders for the project get together and evaluate these potential solutions. Frequently, the participants in this meeting want to interactively modify the proposed 3D designs to explore the design space better. Today's systems and tools do not support this, as computer systems typically support only a single user and computer-aided design tools require significant training. This paper presents the design of a new system to facilitate a collaborative 3D design process. First, we discuss a set of guidelines which have been introduced by others and that are relevant to collaborative 3D design systems. Then, we introduce the new system, which consists of two main parts. The first part is an easy-to-use conceptual 3D design tool that can be used productively even by naive users. The tool provides novel interaction techniques that support important properties of conceptual design. The user interface is non-obtrusive, easy-to-learn, and supports rapid creation and modification of 3D models. The second part is a novel infrastructure for collaborative work, which offers an interactive table and several large interactive displays in a semi-immersive setup. It is designed to support multiple users working together. This infrastructure also includes novel pointing devices that work both as a stylus and a remote pointing device. The combination of the (modified) design tool with the collaborative infrastructure forms a new platform for collaborative virtual 3D design. Then, we present an evaluation of the system against the guidelines for collaborative 3D design. Finally, we present results of a preliminary user study, which asked naive users to collaborate in a 3D design task on the new system.

Keywords Collaborative design · 3D design · Collaborative virtual reality

1 Introduction

Today, digital 3D models are critical in many domains, such as architecture and urban planning, all kinds of industrial design, the entertainment industry, and many engineering applications. Many of the important decisions surrounding a design are made in the initial phases, after the designer(s) have proposed a first version of the design. There, typically in a meeting, the stakeholders in the project get together and evaluate these potential solutions. Frequently, the participants in this meeting want to interactively modify the proposed designs to explore the design space better. Today's design tools and computer infrastructure do not support such activities well, as computer systems typically support only a single user and computer-aided design tools require significant training.

Traditional tools for 3D design require a large amount of training. Part of this is based on the fact that

W. Stuerzlinger (✉) · L. Zaman · A. Pavlovych
York University, Toronto, Canada
URL: http://www.cs.yorku.ca/~wolfgang
URL: http://www.cs.yorku.ca/~zaman
URL: http://www.cs.yorku.ca/~andriyp

J.-Y. Oh
University of Arizona, Tucson, AZ, USA
e-mail: jyoh@optics.arizona.edu

Springer

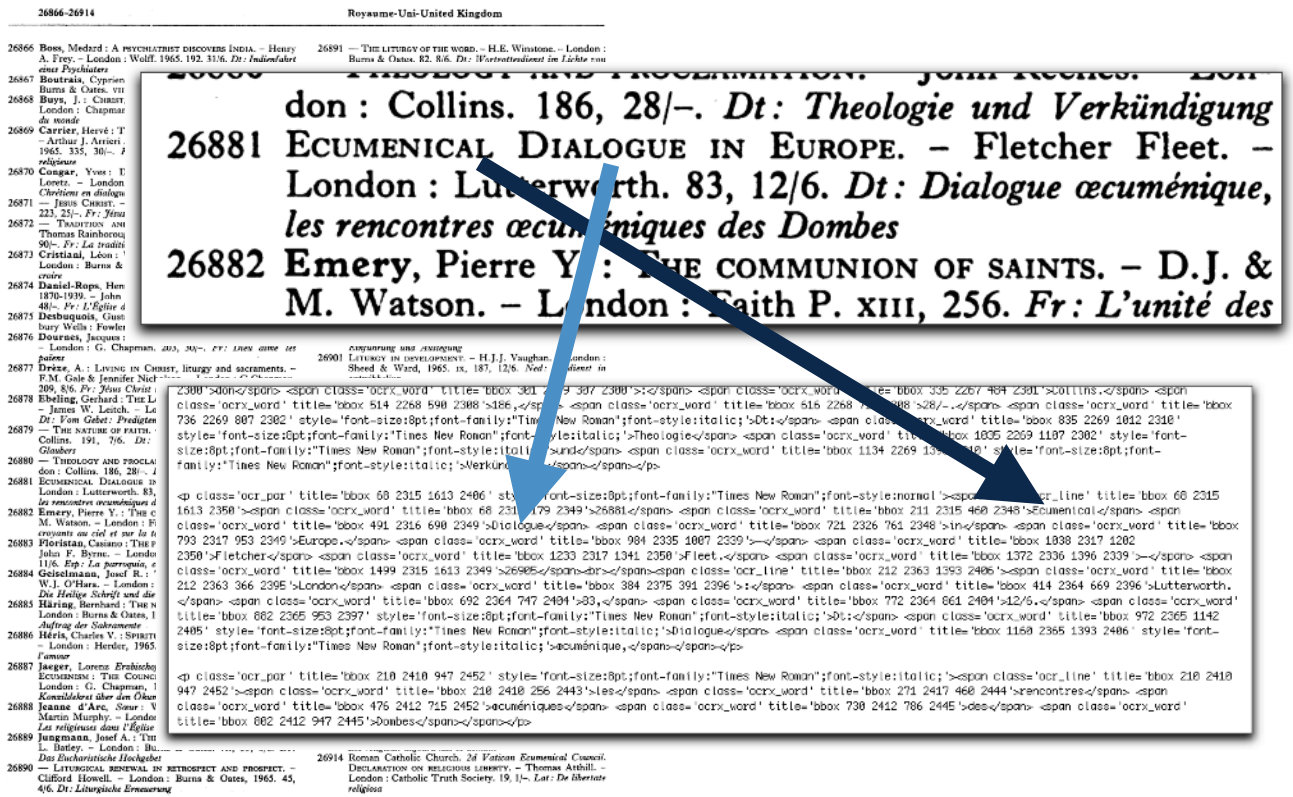
Evitons les PDF…

PDF to text

```
(base) [ Patrice Mac-Pro-de-Patrice ~ ] more /Users/Patrice/Downloads/ark__67375_V
"/Users/Patrice/Downloads/ark__67375_VQC-MZJ5GK10-Q/PDF/10055_2006_Article_48.pdf"
%PDF-1.3^M%<80><84><88><8C><90><94><98><9C><A0><A4><A8><AC><B0><B4><B8><BC><C0>
<C4><C8><CC><D0><D4><D8><DC><E0><E4><E8><EC><F0><F4><F8><FC>^M 1 0 obj^M<< /CropBo
notsAM[ 2 0 R 3 0 R 4 0 R 5 0 R 6 0 R 7 0 R 8 0 R]^M] /Rotate 0 /MediaBox^M[ 0 0 5
 0 R /Contents 10 0 R /Type /Page^M>>^Mendobj^M2 0 obj^M<< /Rect^M[ 367.483032 41
ype /Link /Border^M[ 0 0 0^M] /Type /Annot^M>>^Mendobj^M3 0 obj^M<< /Rect^M[ 435.6
R18) /Subtype /Link /Border
^M[ 0 0 0^M] /Type /Annot^M>>^Mendobj^M4 0 obj^M<< /Rect^M[ 306.142029 255.628006
k /Border^M[ 0 0 0^M] /Type /Annot^M>>^Mendobj^M5 0 obj^M<< /Rect^M[ 430.356018 15
ype /Link /Border^M[ 0 0 0^M] /Type /Annot^M>>^Mendobj^M6 0 obj^M<< /Rect^M[ 454.
CR9) /Subtype /Link /Border^M[ 0 0 0^M] /Type /Annot^M>>^Mendobj^M7 0 obj^M<< /Rec
 /Dest (LCR10) /Subtype /Link /Border^M[ 0 0 0^M] /Type /Annot^M>>^Mendobj^M8 0 ob
.997009^M] /Dest (LCR6) /Subtype /Link /Border^M[ 0 0 0^M] /Type /Annot^M>>^Mendob
 185 0 R^M>> /ProcSet^M[ /PDF /Text^M] /ExtGState^M<< /GS1 163 0 R^M>> /ColorSpace
0 obj^M<< /Filter /FlateDecode /Length 6236^M>>^Mstream
H=89>tWH<8F><E3><E8>^Q<BC><F7><8F>^X67U-D0><ED><91>^<EA>_<87>]^V<9E>_<97>^D^C^@
<EC>^A<E8>6wd<D1>^Q$^<9D>^?<B1>T<C9><DF>N<BD>^<CA>vQ^TA<D0>C^<FC>^V<BD><F7><AA>s
10055_2006_Article_48.pdf
%PDF-1.3^M%<80><84><88><8C><90><94><98><9C><A0><A4><A8><AC><B0><B4><B8><BC><C0>
<C4><C8><CC><D0><D4><D8><DC><E0><E4><E8><EC><F0><F4><F8><FC>^M 1 0 obj^M<< /CropBo
notsAM[ 2 0 R 3 0
 0 R /Contents 10
ype /Link /Border^
R18) /Subtype /Li
 /Dest (LCR5) /Su
.997009^M] /Dest
5029 165.997009^M
L2 490.109985 165
9 156.019012 530.
 /F8 79 0 R /F2 1
 183 0 R^M>>
AM>>^Mendobj^M10
H=89>tWH<8F><E3><
<EC>^A<E8>6wd<D1>
<EA>_<9F><FF>^<
....skipping...
%PDF-1.3^M%<80><8
8><FC>^M1 0 obj^M
Rotate 0 /MediaBo
```

The design and realization of CoViD: a system for collaborative virtual 3D design ISTEX Springer-Verlag London Springer-Verlag L
The design and realization of CoViD: a system for collaborative virtual 3D design Wolfgang Stuerzlinger York University Toronto C
CANADA Andriy Pavlovych York University Toronto CANADA Ji-Young Oh jyoh@optics.arizona.edu University of Arizona Tucs
D569A7062DC3227A0BF16EF149FBDE7980229167 arlc:67375/VQC-MZJ5GK10-Q s10055-006-0048-0 10.1007/s10055-006-0048
4338 1434-9957 Springer-Verlag London 2006 10 2 135 147 pub2TEI-ISTEX A set of style sheets for converting XML documents e
common TEI format. We use TEI Abstract Many important decisions in the design process are made during fairly early on, after desi
domains, these concepts are already realized as 3D digital models. Then, in a meeting, the stakeholders for the project get together an
participants in this meeting want to interactively modify the proposed 3D designs to explore the design space better. Today's systems
typically support only a single user and computer-aided design tools require significant training. This paper presents the design of a n
process. First, we discuss a set of guidelines which have been introduced by others and that are relevant to collaborative 3D design sy
consists of two main parts. The first part is an easy-to-use conceptual 3D design tool that can be used productively even by naive user
support important properties of conceptual design. The user interface is non-obtrusive, easy-to-learn, and supports rapid creation and
infrastructure for collaborative work, which offers an interactive table and several large interactive displays in a semi-immersive setu
together. This infrastructure also includes novel pointing devices that work both as a stylus and a remote pointing device. The combin
collaborative infrastructure forms a new platform for collaborative virtual 3D design. Then, we present an evaluation of the system ag
Finally, we present results of a preliminary user study, which asked naive users to collaborate in a 3D design task on the new system.
Vision Artificial Intelligence (incl. Robotics) Computing Methodologies Computer Graphics Collaborative design 3D design Collabo
Accepted ePublished Virtual Reality (2006) 10:135–147 DOI 10.1007/s10055-006-0048-0 O R I G I N A L A R T I C L E The design
Virtual 3D design Wolfgang Stuerzlinger Æ Loutfouz Zaman Æ Andriy Pavlovych Æ Ji-Young Oh Received: 14 March 2006 / Accep
2006 Springer-Verlag London Limited 2006 Abstract Many important decisions in the design process are made during fairly early on,
many domains, these concepts are already realized as 3D digital models. Then, in a meeting, the stakeholders for the project get togeth
the participants in this meeting want to interactively modify the proposed 3D designs to explore the design space better. Today's syste
systems typically support only a single user and computer-aided design tools require significant training. This paper presents the desig
design process. First, we discuss a set of guidelines which have been introduced by others and that are relevant to collaborative 3D de
which consists of two main parts. The first part is an easy-to-use conceptual 3D design tool that can be used productively even by nai
techniques that support important properties of conceptual design. The user interface is non-obtrusive, easy-to-learn, and supports rap
second part is a novel infrastructure for collaborative work, which offers an interactive table and several large interactive displays in a
A. Pavlovych York University, Toronto, Canada URL: http://www.cs.yorku.ca/~wolfgang URL: http://www.cs.yorku.ca/~zaman URL
University of Arizona, Tucson, AZ, USA e-mail: jyoh@optics.arizona.edu setup. It is designed to support multiple users working tog
devices that work both as a stylus and a remote pointing device. The combination of the (modified) design tool with the collaborative
virtual 3D design. Then, we present an evaluation of the system against the guidelines for collaborative 3D design. Finally, we presen
native users to collaborate in a 3D design task on the new system. Keywords Collaborative design 3D design Collaborative virtual rea
critical in many domains, such as architecture and urban planning, all kinds of industrial design, the entertainment industry, and many
decisions surrounding a design are made in the initial phases, after the designer(s) have proposed a first version of the design. There,
get together and evaluate these potential solutions. Frequently, the participants in this meeting want to interactively modify the propos

Des documents « images »



L'encodage des documents : formats ouverts et méta-données

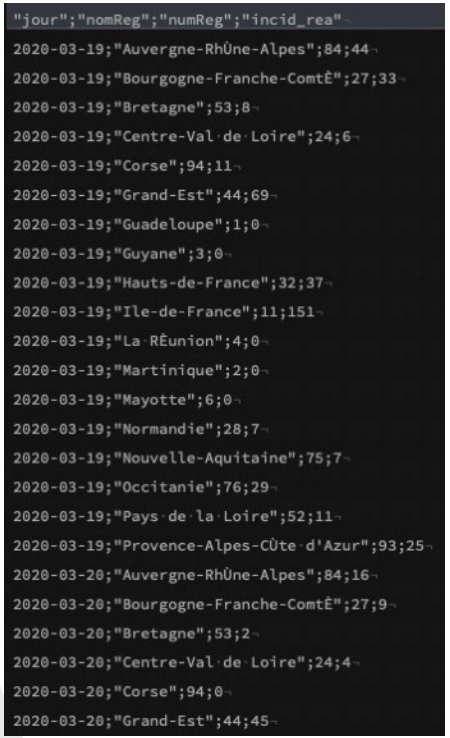
```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0"
     xmlns:ns1="http://standoff.proposal"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="https://xml-schema.delivery.istex.fr/formats/tei-istex.xsd">
   <teiHeader>
      <fileDesc>
         <titleStmt>
            <title level="a" type="main" xml:lang="en">The design and realization of CoViD: a system for
                design</title>
         </titleStmt>
         <publicationStmt>
            <authority>ISTEX</authority>
            <publisher ref="https://scientific-publisher.data.istex.fr/ark:/67375/H02-SWLMH5L1-1">Spring
            <pubPlace>London</pubPlace>
            <availability>
               <licence>Springer-Verlag London Limited</licence>
               <p scheme="https://loaded-corpus.data.istex.fr/ark:/67375/XBH-3XSW68JL-F">springer</p>
            </availability>
            <date type="published" when="2006">2006</date>
         </publicationStmt>
         <notesStmt>
            <note type="content-type"
                  subtype="research-article"
                  source="OriginalPaper"
                  scheme="https://content-type.data.istex.fr/ark:/67375/XTP-1JC4F85T-7">research-article
            <note type="publication-type"
                  subtype="journal"
                  scheme="https://publication-type.data.istex.fr/ark:/67375/JMC-5WTPMB5N-F">journal</not
         </notesStmt>
         <sourceDesc>
            <biblStruct>
               <analytic>
                  <title level="a" type="main" xml:lang="en">The design and realization of CoViD: a syst
                      design</title>
                  <author role="corresp">
                     <persName>
                        <forename type="first">Wolfgang</forename>
                        <surname>Stuerzlinger</surname>
                     </persName>
                     <affiliation>
                        <orgName type="institution">York University</orgName>
                        <address>
                           <settlement>Toronto</settlement>
                           <country key="CA" xml:lang="en">CANADA</country>
                        </address>
                     </affiliation>
```

XML


```
Abstract Many important decisions in the design
process are made during fairly early on, after designers
have presented initial concepts. In many domains,
these concepts are already realized as 3D digital
models. Then, in a meeting, the stakeholders for the
project get together and evaluate these potential
solutions. Frequently, the participants in this meeting
want to interactively modify the proposed 3D designs
to explore the design space better. Today's systems and
tools do not support this, as computer systems typically
support only a single user and computer-aided design
tools require significant training. This paper presents
the design of a new system to facilitate a collaborative
3D design process. First, we discuss a set of guidelines
which have been introduced by others and that are
relevant to collaborative 3D design systems. Then, we
introduce the new system, which consists of two main
parts. The first part is an easy-to-use conceptual 3D
design tool that can be used productively even by naïve
users. The tool provides novel interaction techniques
that support important properties of conceptual design.
The user interface is non-obtrusive, easy-to-learn, and
supports rapid creation and modification of 3D models.
The second part is a novel infrastructure for collaborative work, which of
    a semi-immersive

W. Stuerzlinger (&amp;amp;)  L. Zaman  A. Pavlovych
York University, Toronto, Canada
URL: http://www.cs.yorku.ca/~wolfgang
URL: http://www.cs.yorku.ca/~zaman
URL: http://www.cs.yorku.ca/~andriyp
J.-Y. Oh
University of Arizona, Tucson, AZ, USA
e-mail: jyoh@optics.arizona.edu

setup. It is designed to support multiple users working
together. This infrastructure also includes novel
pointing devices that work both as a stylus and a remote pointing device.
    collaborative infrastructure
forms a new platform for collaborative virtual 3D design. Then, we present
against the guidelines for collaborative 3D design. Finally, we present re
which asked naive users to collaborate in a 3D design
task on the new system.
Keywords Collaborative design  3D design
Collaborative virtual reality
```

https://tei-c.org/release/doc/tei-p5-doc/en/html/SG.html

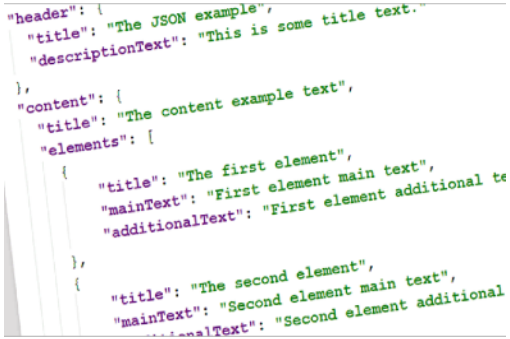
D'autres formats ouverts



```
"jour";"nomReg";"numReg";"incid_rea"
2020-03-19;"Auvergne-RhÙne-Alpes";84;44
2020-03-19;"Bourgogne-Franche-ComtÈ";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 RÉunion";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-CÙte d'Azur";93;25
2020-03-20;"Auvergne-RhÙne-Alpes";84;16
2020-03-20;"Bourgogne-Franche-ComtÈ";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
```

CSV

https://en.wikipedia.org/wiki/Comma-separated_values



```
"header": {
  "title": "The JSON example",
  "descriptionText": "This is some title text."
},
"content": {
  "title": "The content example text",
  "elements": [
    {
      "title": "The first element",
      "mainText": "First element main text",
      "additionalText": "First element additional te
    },
    {
      "title": "The second element",
      "mainText": "Second element main text",
      "additionalText": "Second element additional
```

JSON

https://en.wikipedia.org/wiki/JSON

```
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     133.7
      quantity:  1

bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
    city:   East Centerville
    state:  KS

ship-to:  *id001

specialDelivery:  >
    Follow the Yellow Brick
```

YAML

https://en.wikipedia.org/wiki/YAML

Balises orientées affichage vs. annotations sémantiques

BACHELARD, Gaston

1975 *La formation de l'esprit scientifique* (Paris, Librairie philosophique J. Vrin) [1re éd. 1938].

BARTHES, Roland

1957 *Mythologies* (Paris, Le Seuil).

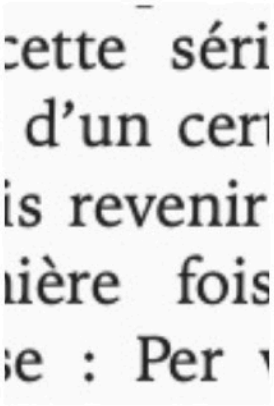
BENVENISTE, Émile

1966 De la subjectivité dans le langage, *in* É. Benveniste (dir.), *Problèmes de linguistique générale*, I (Paris, Gallimard) : 258-266.

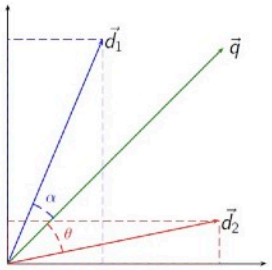
1974 La forme et le sens dans le langage, *in* É. Benveniste (dir.), *Problèmes de linguistique générale*, II (Paris, Gallimard) : 215-240.

```
<bibl><hi rend="bold">C</hi><hi rend="small-caps bold">oupry</hi><hi rend="bold">, François<lb/></hi>1999 <hi r
<bibl><hi rend="bold">C</hi><hi rend="small-caps bold">ruz</hi><hi rend="bold"> S</hi><hi rend="small-caps bold
<bibl><hi rend="bold">D</hi><hi rend="small-caps bold">escola</hi><hi rend="bold">, Philippe<lb/></hi>1994 Rétr
<bibl><hi rend="bold">D</hi><hi rend="small-caps bold">ubois</hi><hi rend="bold">, Jacques<lb/></hi>1992 <hi re
<bibl>
  <hi xml:lang="en" rend="italic bold">Etnoloska sticisca</hi>
  <hi xml:lang="en" rend="italic bold">
    <lb/>
  </hi>
  <hi xml:lang="en">1997 </hi>
  <hi xml:lang="en" rend="italic">Prejudices and stereotypes in the social sciences and humanities</hi>
  <hi xml:lang="en">, 5 et 7 (Ljubljana, Jezernik, Bozidzr Ed.).</hi>
</bibl>
<bibl><hi rend="bold">F</hi><hi rend="small-caps bold">erney</hi><hi rend="bold">, Alice<lb/></hi>1997 <hi rend
<bibl><hi rend="bold">G</hi><hi rend="small-caps bold">eertz</hi><hi rend="bold">, Clifford<lb/></hi>1986 Comme
<bibl>
  <hi xml:lang="en" rend="bold">G</hi>
  <hi xml:lang="en" rend="small-caps bold">ilman</hi>
  <hi xml:lang="en" rend="bold">, Sander L.<lb/></hi>
  <hi xml:lang="en">1985 </hi>
  <hi xml:lang="en" rend="italic">Difference and pathology, stereotypes of sexuality, race and madness</hi>
  <hi xml:lang="en"> (Ithaca, Cornell University Press).</hi>
</bibl>
```

Des formes, des distributions et des vecteurs



Des formes rassemblées,
des sacs (de lettres, de mots)



Des vecteurs

Mot	Probabilité

Des modèles de langues

$$P(Classe|(w_1, w_2, ..., w_{T-1}, w_T))$$

$$\text{Similarité}(d_1, d_2) \approx \vec{d_1}.\vec{d_2}$$

$$\text{Similarité}(d_1, d_2) \approx \cos(\vec{d_1}, \vec{d_2})$$

$$P(w_1, w_2, ..., w_{T-1}, w_T) = \prod_{t=1}^{T} P(w_t|w_{t-1}, w_{t-2}, ..., w_1)$$

the	cat	sat	on	the	mat	$P(w_1)$
the	cat	sat	on	the	mat	$P(w_2\|w_1)$
the	cat	sat	on	the	mat	$P(w_3\|w_2, w_1)$
the	cat	sat	on	the	mat	$P(w_4\|w_3, w_2, w_1)$
the	cat	sat	on	the	mat	$P(w_5\|w_4, w_3, w_2, w_1)$
the	cat	sat	on	the	mat	$P(w_6\|w_5, w_4, w_3, w_2, w_1)$

Slide Credit: Piotr Mirowski

Exemple : suppression des balises

suppression des balises (tags) :

```
df.Abstract.replace(to_replace=r"<.*?>", value="", inplace=True, regex=True)
```

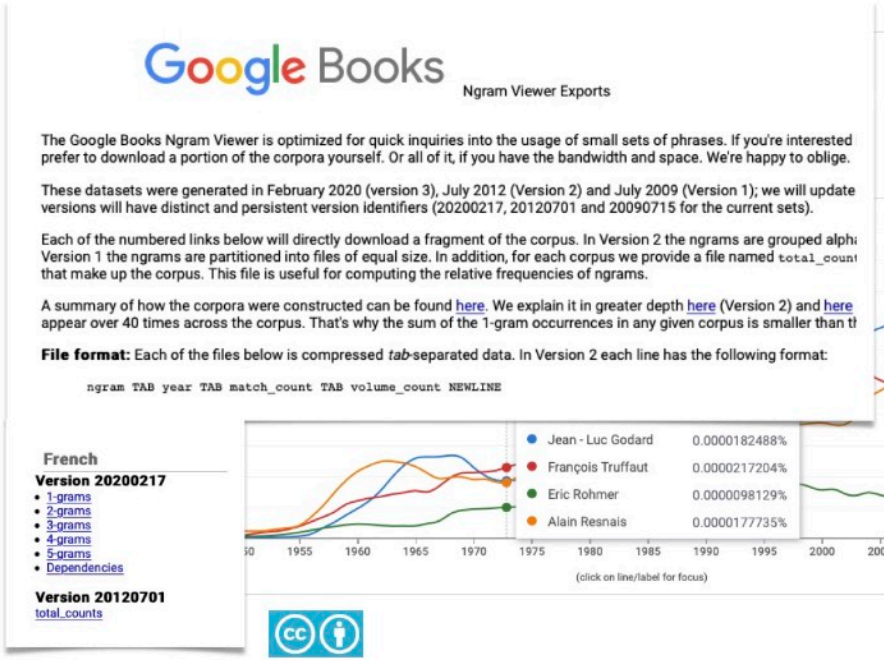
Expressions régulières (*Regular Expressions = regex*)

https://docs.python.org/3/library/re.html

suppression des parenthèses :

```
df.Abstract.replace(to_replace="[()]", value="", inplace=True, regex=True)
```

Un 1er exemple de modèle de langue (2009)



Google Books

Ngram Viewer Exports

The Google Books Ngram Viewer is optimized for quick inquiries into the usage of small sets of phrases. If you're interested prefer to download a portion of the corpora yourself. Or all of it, if you have the bandwidth and space. We're happy to oblige.

These datasets were generated in February 2020 (version 3), July 2012 (Version 2) and July 2009 (Version 1); we will update versions will have distinct and persistent version identifiers (20200217, 20120701 and 20090715 for the current sets).

Each of the numbered links below will directly download a fragment of the corpus. In Version 2 the ngrams are grouped alph Version 1 the ngrams are partitioned into files of equal size. In addition, for each corpus we provide a file named `total_count` that make up the corpus. This file is useful for computing the relative frequencies of ngrams.

A summary of how the corpora were constructed can be found here. We explain it in greater depth here (Version 2) and here appear over 40 times across the corpus. That's why the sum of the 1-gram occurrences in any given corpus is smaller than t

File format: Each of the files below is compressed *tab*-separated data. In Version 2 each line has the following format:

```
ngram TAB year TAB match_count TAB volume_count NEWLINE
```

French

Version 20200217

- 1-grams
- 2-grams
- 3-grams
- 4-grams
- 5-grams
- Dependencies

Version 20120701

total_counts

Jean - Luc Godard 0.0000182488%
François Truffaut 0.0000217204%
Eric Rohmer 0.0000098129%
Alain Resnais 0.0000177735%

1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

(click on line/label for focus)

Jean - Luc Godard
François Truffaut
Alain Resnais
Eric Rohmer

Language	#Volumes	#Tokens
English	4,541,627	468,491,999,592
Spanish	854,649	83,967,471,303
French	792,118	102,174,681,393
German	657,991	64,784,628,286
Russian	591,310	67,137,666,353
Italian	305,763	40,288,810,817
Chinese	302,652	26,859,461,025
Hebrew	70,636	8,172,543,728

Table 1: Number of volumes and tokens for each language in our corpus. The total collection contains more than 6% of all books ever published.

https://books.google.com/ngrams

Documents, lexique et *dataframes :* des représentations

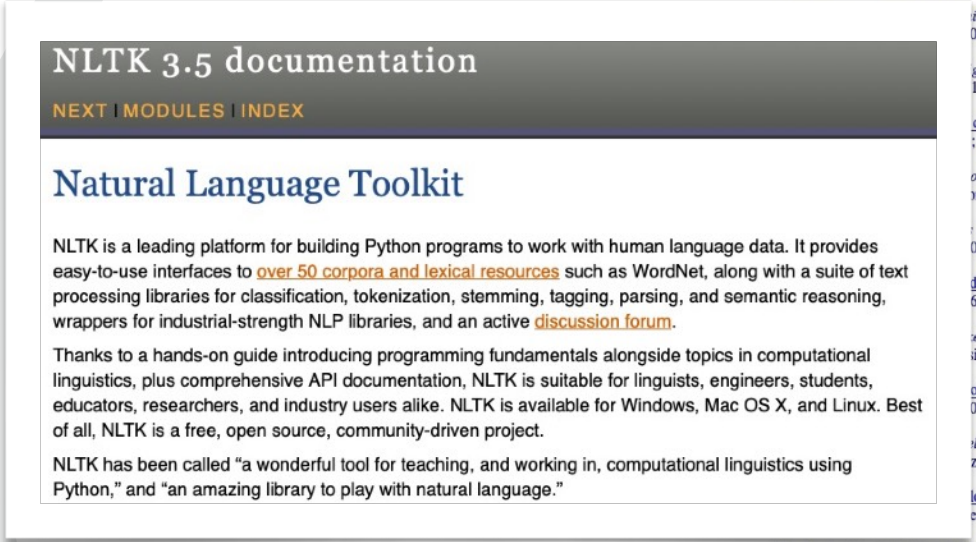
	méta-donnée 1	méta-donnée 2	…
document 1			
document 2			
document 3			
…			

	mot 1	mot 2	…
document 1			
document 2			
document 3			
…			

sacs de mots

	dim1	dim 2	…
mot 1			
mot 2			
mot 3			
…			

espaces de représentation réduits



http://www.nltk.org

NLTK Corpora

NLTK has built-in support for dozens of corpora and trained models, as listed below. To use these within NLTK we recommend that

Please consult the README file included with each corpus for further information.

icode Version 7.0.0 character properties in Perl [download | source]
00266; author: ; copyright: ; license: ;

gner (Sultan et al. 2015) subset of the Paraphrase Database. [download | source]
11; author: ; copyright: ; license: Creative Commons Attribution 3.0 Unported (CC-BY);

download | source]
; author: Jan Strunk; copyright: ; license: ;

or de Sufixos da Lingua Portuguesa) [download | source]
r: Viviane Moreira Orengo (vmorengo@inf.ufrgs.br) and Christian Huyck; copyright: ; license: ;

[download | source]
0510; author: ; copyright: ; license: ;

d | source]
6785405; author: ; copyright: ; license: ;

er (Maximum entropy) [download | source]
ize: 13404747; author: ; copyright: ; license: ;

ownload | source]
0961490; author: ; copyright: ; license: ;

l [download | source]
ze: 24516205; author: ; copyright: ; license: ;

load | source]
e: 49396025; author: ; copyright: ; license: ;

11. *Evaluation data from WMT15* [download | source]
id: `wmt15_eval`; size: 383096; author: ; copyright: ; license: ;

12. *Grammars for Spanish* [download | source]
id: `spanish_grammars`; size: 4047; author: Kepa Sarasola; copyright: ; license: ;

13. *Sample Grammars* [download | source]
id: `sample_grammars`; size: 20293; author: ; copyright: ; license: ;

14. *Large context-free and feature-based grammars for parser comparison* [download | source]
id: `large_grammars`; size: 283747; author: ; copyright: ; license: See the individual grammar files;

15. *Grammars from NLTK Book* [download | source]
id: `book_grammars`; size: 9103; author: Ewan Klein; copyright: ; license: ;

16. *Grammars for Basque* [download | source]
id: `basque_grammars`; size: 4704; author: Kepa Sarasola; copyright: ; license: ;

http://www.nltk.org

Some simple things you can do with NLTK

Tokenize and tag some text:

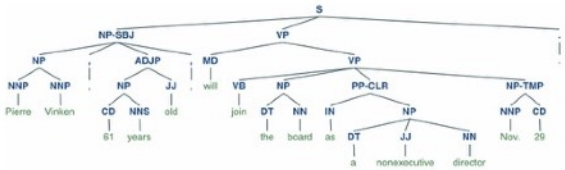
```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

Identify named entities:

```
>>> entities = nltk.chunk.ne_chunk(tagged)
>>> entities
Tree('S', [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'),
          ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'),
      Tree('PERSON', [('Arthur', 'NNP')]),
          ('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'),
          ('very', 'RB'), ('good', 'JJ'), ('.', '.')])
```

Display a parse tree:

```
>>> from nltk.corpus import treebank
>>> t = treebank.parsed_sents('wsj_0001.mrg')[0]
>>> t.draw()
```



Exemple : élimination des mots outils (NLTK)

```
{"weren't", 'this', 'well', 'aurez', 'étiez', 'hadn', 'not', 'by', 'wouldn', 'se', 'soit', 'use', 'possible',
 'can', 'techniques', 'now', 'd', 'il', 'étés', 'aurai', 'too', 'conference', 'étaient', 'his', 'y', 'de',
'or', 'because', 'myself', 'sur', 'eurent', 'j', 'to', 'her', 'own', 'eux', 'tu', 'ait', 'êtes', 'à', 'ta',
"don't", 'mustn', 'au', 'avais', 'whom', 'our', 'seras', 'des', 'some', "won't", 'auraient', 'above',
"you'll", 'improving', "aren't", 'weren', 'mais', 't', 'yours', 'and', 'them', 'eûtes', 'do', 'couldn',
'wasn', 'est', "doesn't", "hadn't", 'c', 'aurions', 'aurons', 'avait', 'until', 'between', 'ou', 'him',
'into', 'ayants', 'towards', "you'd", 'any', 'doing', 'aient', 'allows', 'several', 'global', "she's",
'étantes', 'what', 'avec', "shan't", 'es', 'fusses', 'eus', 'my', 'd'', 'fûmes', 'tes', 'serions', 'of',
'fut', "needn't", 'best', 'furent', 'from', "wasn't", 'yourselves', 'ayantes', 'paper', 'each', 'with',
```

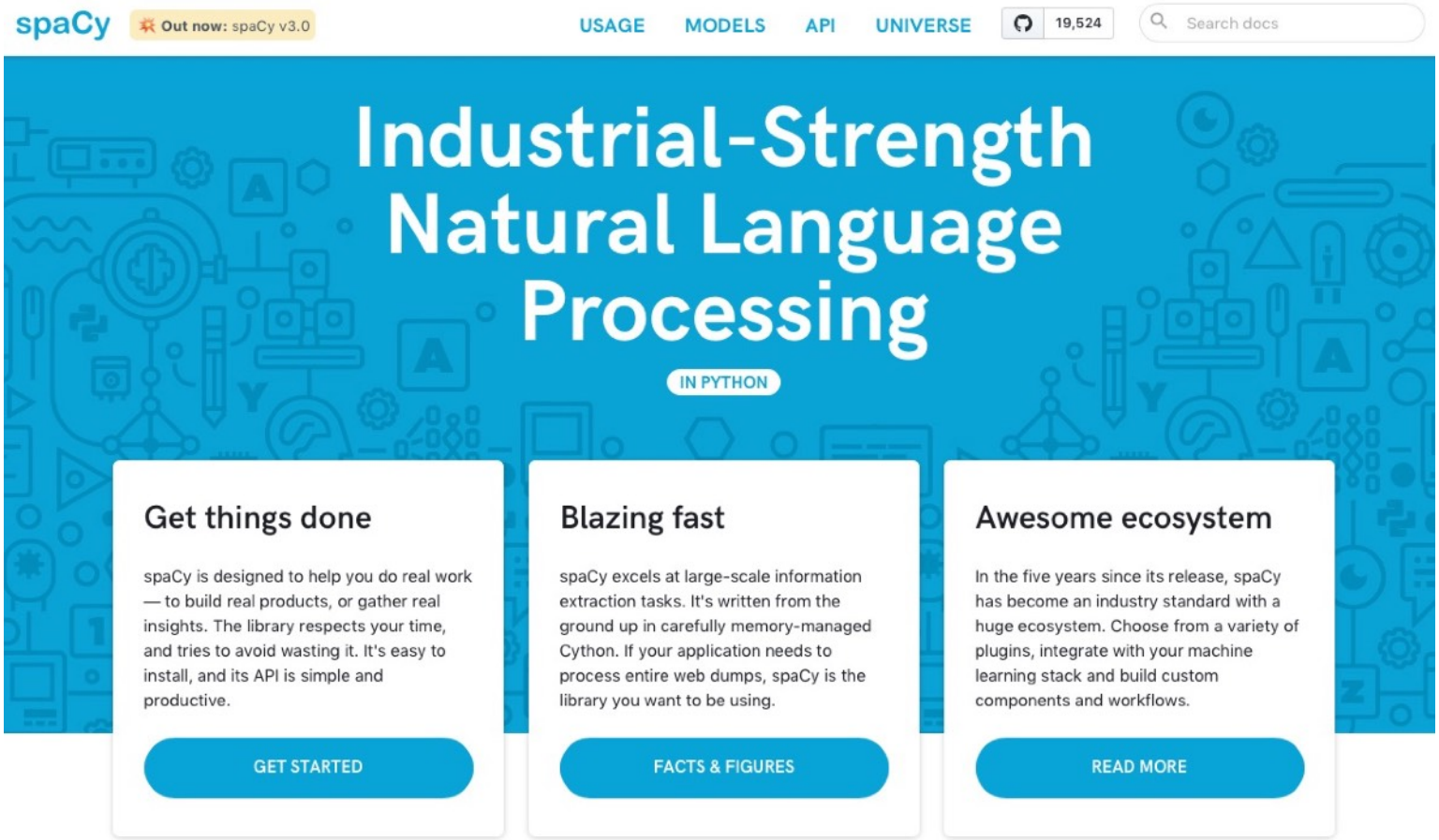
les mots outils de l'anglais et du français

```
from nltk.corpus import stopwords
    stop_words = set(stopwords.words('english'))
    more_stop_words = {"problem", "results", "show", "based", "paper", "using", "two", "one",
"used", "approach", "present", "first", "may", "use", "given", "many", "known", "efficient",
"study", "proposed", "case", "different", "source", "analysis", "method", "problems", "proces
"several", "multiple", "consider", "prove", "possible", "novel", "work", "shown", "provide",
"including", "paper", "high", "des", "obtained", "well", "allows", "simple", "due", "via",
"towards", "pour", "improving", "approaches", "global", "output", "input", "dans", "techniqu
"tool", "research", "conference", "scientific", "proceedings", "exploiting", "international"
"applications", "d'", "In", "le", "thesis", "projet"}
    stop_words.update(more_stop_words)
    mots_outils = set(stopwords.words('french'))
    stop_words.update(mots_outils)
```

concatène des chaînes d'une liste

```
df.Title = df.Title.apply(lambda x: ' '.join([word for word in
x.split() if word.lower() not in (stop_words)]))
```

https://spacy.io/



Edit the code & try spaCy spaCy v3.0 · Python 3 · via Binder

```
# pip install -U spacy
# python -m spacy download en_core_web_sm
import spacy

# Load English tokenizer, tagger, parser and NER
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. “I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn’t "
        "worth talking to,” said Thrun, in an interview with Recode earlier "
        "this week.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)
```

RUN

```
Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'th
e company', 'him', 'I', 'you', 'very senior CEOs', 'major American car companies',
'my hand', 'I', 'Thrun', 'an interview', 'Recode']
Verbs: ['start', 'work', 'drive', 'take', 'tell', 'shake', 'turn', 'be', 'talk', '
say']
Sebastian Thrun PERSON
2007 DATE
American NORP
Thrun PERSON
Recode PERSON
earlier this week DATE
```

Features

- Support for **69+ languages**
- **58 trained pipelines** for 18 languages
- Multi-task learning with pretrained **transformers** like BERT
- Pretrained **word vectors**
- State-of-the-art speed
- Production-ready **training system**
- Linguistically-motivated **tokenization**
- Components for **named entity** recognition, part-of-speech tagging, dependency parsing, sentence segmentation, **text classification**, lemmatization, morphological analysis, entity linking and more
- Easily extensible with **custom components** and attributes
- Support for custom models in **PyTorch**, **TensorFlow** and other frameworks
- Built in **visualizers** for syntax and NER
- Easy **model packaging**, deployment and workflow management
- Robust, rigorously evaluated accuracy

https://spacy.io/

Features

In the documentation, you'll come across mentions of spaCy's features and capabilities. Some of them refer to linguistic concepts, while others are related to more general machine learning functionality.

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) **Tagging**	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of "was" is "be", and the lemma of "rats" is "rat".
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named "real-world" objects, like persons, companies or locations.
Entity Linking (EL)	Disambiguating textual entities to unique identifiers in a knowledge base.
Similarity	Comparing words, text spans and documents and how similar they are to each other.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.
Training	Updating and improving a statistical model's predictions.
Serialization	Saving objects to files or byte strings.

Exemple : élimination des mots outils (spaCy)

```
# Étape 1: Installez et importez les bibliothèques nécessaires.
import pandas as pd
import spacy

# Installez spaCy et le modèle de langue (par exemple pour l'anglais)
# !pip install spacy
# !python -m spacy download en_core_web_sm

# Étape 2: Chargez le modèle de langue de spaCy.
nlp = spacy.load("en_core_web_sm")

# Étape 3: Créez une fonction pour filtrer les stop words.
def remove_stopwords(text):
    doc = nlp(text)
    filtered_sentence = " ".join([token.text for token in doc if not token.is_stop])
    return filtered_sentence

# Exemple de DataFrame
data = {
    'text': ["This is a sample sentence.", "Another example with different words."]
}
df = pd.DataFrame(data)

# Étape 4: Appliquez cette fonction à la colonne du DataFrame.
df['filtered_text'] = df['text'].apply(remove_stopwords)
print(df)
```

Aix*Marseille université
Socialement engagée

Annexes : pense-bêtes

Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

`>>> x+2` `7`	Sum of two variables
`>>> x-2` `3`	Subtraction of two variables
`>>> x*2` `10`	Multiplication of two variables
`>>> x**2` `25`	Exponentiation of a variable
`>>> x%2` `1`	Remainder of a variable
`>>> x/float(2)` `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis
learn — Machine learning
NumPy — Scientific computing
matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python
spyder — Free IDE that is included with Anaconda
jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset	
`>>> my_array[1]` `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False,  True], dtype=bool)
>>> my_array * 2
  array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

DataCamp
Learn Python for Data Science Interactively

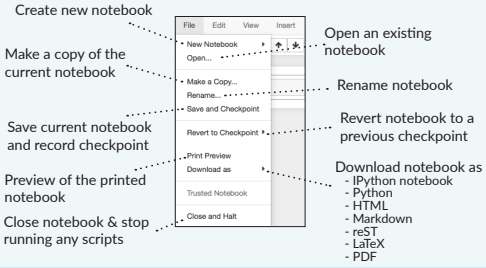
Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

Jupyter Notebook

Learn More Python for Data Science Interactively at www.DataCamp.com

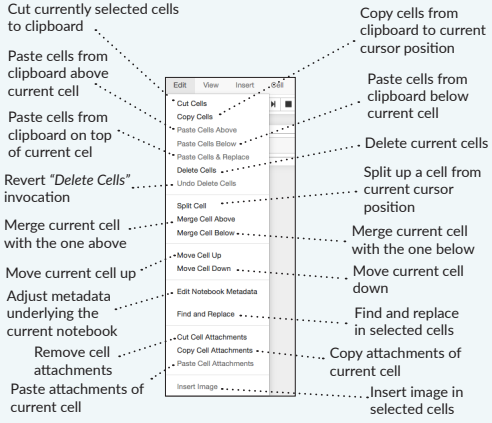
Saving/Loading Notebooks



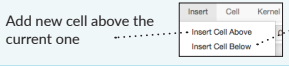
Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Edit Cells



Insert Cells

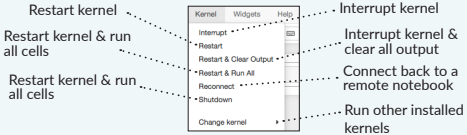


Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython — IRkernel — IJulia

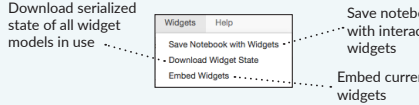
Installing Jupyter Notebook will automatically install the IPython kernel.



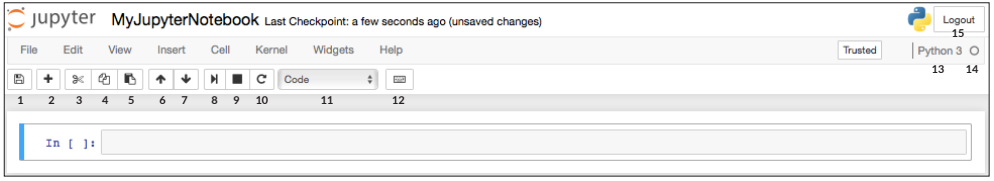
Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.



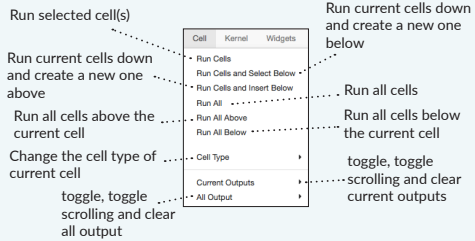
Command Mode:



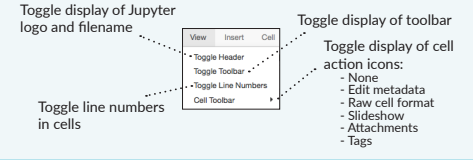
Edit Mode:

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

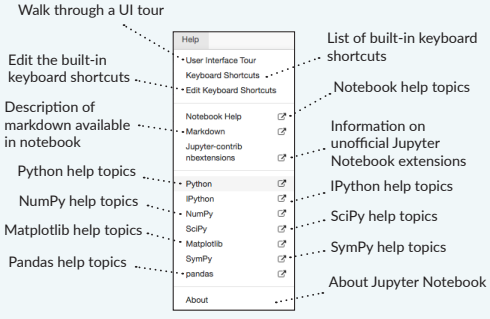
Executing Cells



View Cells



Asking For Help



Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

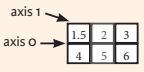
Use the following import convention:

```
>>> import numpy as np
```

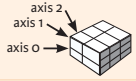
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()          Array-wise sum
>>> a.min()          Array-wise minimum value
>>> b.max(axis=0)    Maximum value of an array row
>>> b.cumsum(axis=1) Cumulative sum of the elements
>>> a.mean()         Mean
>>> b.median()       Median
>>> a.corrcoef()     Correlation coefficient
>>> np.std(b)        Standard deviation
```

Copying Arrays

```
>>> h = a.view()     Create a view of the array with the same data
>>> np.copy(a)       Create a copy of the array
>>> h = a.copy()     Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()         Sort an array
>>> c.sort(axis=0)   Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                              Select the element at the 2nd index
  3
>>> b[1,2]                            Select the element at row 0 column 2
  6.0                                  (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                            Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                          Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                             Select all items at row 0
  array([[1.5, 2., 3.]])               (equivalent to b[0:1, :])
>>> c[1,...]                          Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                         Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                            Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]     Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]     Select a subset of the matrix's rows
  array([[ 4. ,  5. ,  6. ,  4. ],     and columns
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)              Permute array dimensions
>>> i.T                              Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                        Flatten the array
>>> g.reshape(3,-2)                  Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                  Return a new array with shape (2,6)
>>> np.append(h,g)                   Append items to an array
>>> np.insert(a, 1, 5)               Insert items in an array
>>> np.delete(a,[1])                 Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)     Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                 Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                       Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                 Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))           Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                       Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                   Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]   index
>>> np.vsplit(c,2)                   Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3. ,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]]])]
```

Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]
>>> np.ogrid[0:2,0:2]
>>> np.r_[[3,[0]*5,-1:1:10j]
>>> np.c_[b,c]
```

Create a dense meshgrid
Create an open meshgrid
Stack arrays vertically (row-wise)
Create stacked column-wise arrays

Shape Manipulation

```
>>> np.transpose(b)
>>> b.flatten()
>>> np.hstack((b,c))
>>> np.vstack((a,b))
>>> np.hsplit(c,2)
>>> np.vpslit(d,2)
```

Permute array dimensions
Flatten the array
Stack arrays horizontally (column-wise)
Stack arrays vertically (row-wise)
Split the array horizontally at the 2nd index
Split the array vertically at the 2nd index

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5])
```

Create a polynomial object

Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)
```

Vectorize functions

Type Handling

```
>>> np.real(c)
>>> np.imag(c)
>>> np.real_if_close(c,tol=1000)
>>> np.cast['f'](np.pi)
```

Return the real part of the array elements
Return the imaginary part of the array elements
Return a real array if complex parts close to 0
Cast object to a data type

Other Useful Functions

```
>>> np.angle(b,deg=True)
>>> g = np.linspace(0,np.pi,num=5)
>>> g [3:] += np.pi
>>> np.unwrap(g)
>>> np.logspace(0,10,3)
>>> np.select([c<4],[c*2])
>>> misc.factorial(a)
>>> misc.comb(10,3,exact=True)
>>> misc.central_diff_weights(3)
>>> misc.derivative(myfunc,1.0)
```

Return the angle of the complex argument
Create an array of evenly spaced values (number of samples)
Unwrap
Create an array of evenly spaced values (log scale)
Return values from a list of arrays depending on conditions
Factorial
Combine N things taken at k time
Weights for Np-point central derivative
Find the n-th derivative of a function at a point

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I
>>> linalg.inv(A)
```

Inverse
Inverse

```
>>> A.T
```

Tranpose matrix

```
>>> A.H
```

Conjugate transposition

```
>>> np.trace(A)
```

Trace

Norm

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

Frobenius norm
L1 norm (max column sum)
L inf norm (max row sum)

Rank

```
>>> np.linalg.matrix_rank(C)
```

Matrix rank

Determinant

```
>>> linalg.det(A)
```

Determinant

Solving linear problems

```
>>> linalg.solve(A,b)
>>> E = np.mat(a).T
>>> linalg.lstsq(D,E)
```

Solver for dense matrices
Solver for dense matrices
Least-squares solution to linear matrix equation

Generalized inverse

```
>>> linalg.pinv(C)
```

Compute the pseudo-inverse of a matrix (least-squares solver)

```
>>> linalg.pinv2(C)
```

Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)
>>> G = np.mat(np.identity(2))
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)
>>> I = sparse.csc_matrix(D)
>>> J = sparse.dok_matrix(A)
>>> E.todense()
>>> sparse.isspmatrix_csc(A)
```

Create a 2X2 identity matrix
Create a 2x2 identity matrix
Compressed Sparse Row matrix
Compressed Sparse Column matrix
Dictionary Of Keys matrix
Sparse matrix to full matrix
Identify sparse matrix

Sparse Matrix Routines

Inverse

```
>>> sparse.linalg.inv(I)
```

Inverse

Norm

```
>>> sparse.linalg.norm(I)
```

Norm

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)
```

Solver for sparse matrices

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)
```

Sparse matrix exponential

Matrix Functions

Addition

```
>>> np.add(A,D)
```

Addition

Subtraction

```
>>> np.subtract(A,D)
```

Subtraction

Division

```
>>> np.divide(A,D)
```

Division

Multiplication

```
>>> np.multiply(D,A)
>>> np.dot(A,D)
>>> np.vdot(A,D)
>>> np.inner(A,D)
>>> np.outer(A,D)
>>> np.tensordot(A,D)
>>> np.kron(A,D)
```

Multiplication
Dot product
Vector dot product
Inner product
Outer product
Tensor dot product
Kronecker product

Exponential Functions

```
>>> linalg.expm(A)
>>> linalg.expm2(A)
>>> linalg.expm3(D)
```

Matrix exponential
Matrix exponential (Taylor Series)
Matrix exponential (eigenvalue decomposition)

Logarithm Function

```
>>> linalg.logm(A)
```

Matrix logarithm

Trigonometric Functions

```
>>> linalg.sinm(D)
>>> linalg.cosm(D)
>>> linalg.tanm(A)
```

Matrix sine
Matrix cosine
Matrix tangent

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
>>> linalg.coshm(D)
>>> linalg.tanhm(A)
```

Hyperbolic matrix sine
Hyperbolic matrix cosine
Hyperbolic matrix tangent

Matrix Sign Function

```
>>> np.sigm(A)
```

Matrix sign function

Matrix Square Root

```
>>> linalg.sqrtm(A)
```

Matrix square root

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)
```

Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
>>> l1, l2 = la
>>> v[:,0]
>>> v[:,1]
>>> linalg.eigvals(A)
```

Solve ordinary or generalized eigenvalue problem for square matrix
Unpack eigenvalues
First eigenvector
Second eigenvector
Unpack eigenvalues

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)
```

Singular Value Decomposition (SVD)
Construct sigma matrix in SVD

LU Decomposition

```
>>> P,L,U = linalg.lu(C)
```

LU Decomposition

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)
>>> sparse.linalg.svds(H, 2)
```

Eigenvalues and eigenvectors
SVD

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

DataCamp
Learn Python for Data Science Interactively

Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science Interactively at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

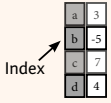
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

Columns

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

Index

A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5

>>> df[1:]
   Country     Capital  Population
1   India  New Delhi  1303171035
2  Brazil   Brasília   207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
 Country      Brazil
 Capital    Brasília
 Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
 0     Brussels
 1    New Delhi
 2     Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
 'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
```

Series `s` where value is not >1 `s` where value is <-1 or >2

```
>>> df[df['Population']>1200000000]
```

Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a     10.0
  b      NaN
  c      5.0
  d      7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c     5.0
  d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

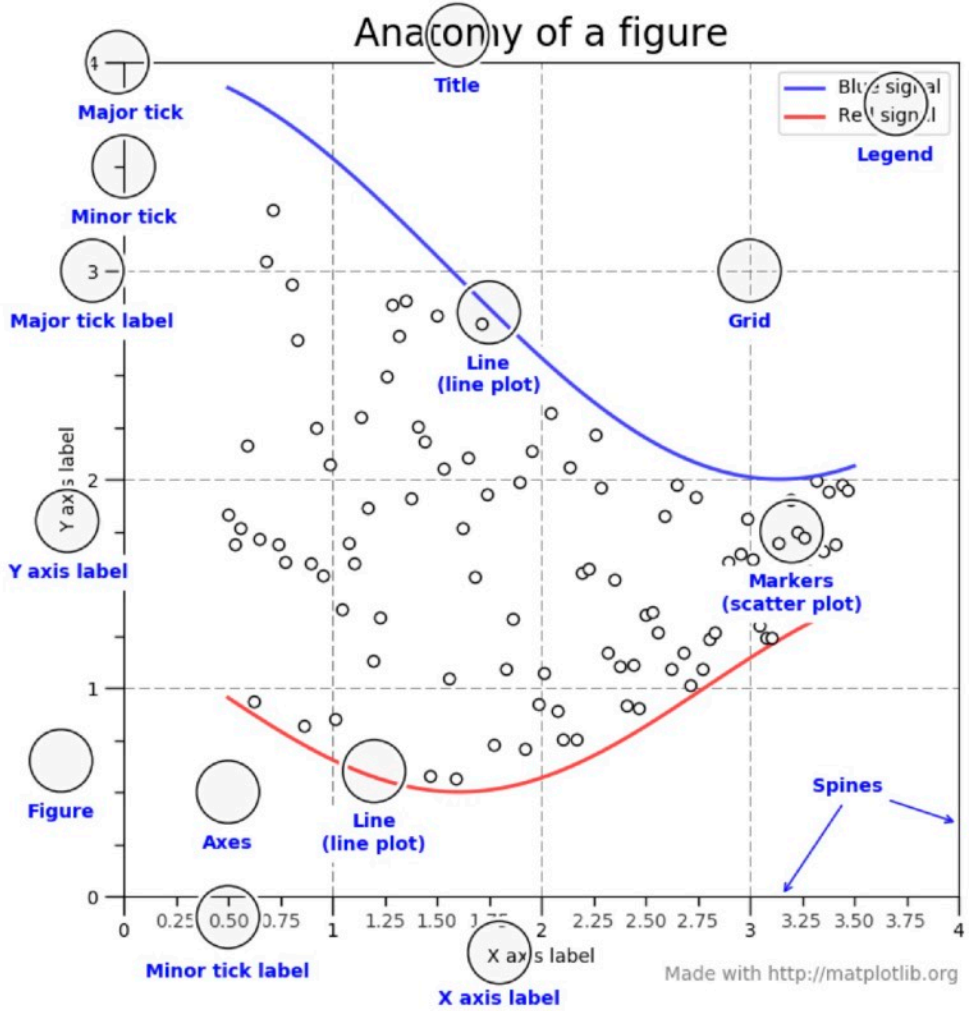
Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

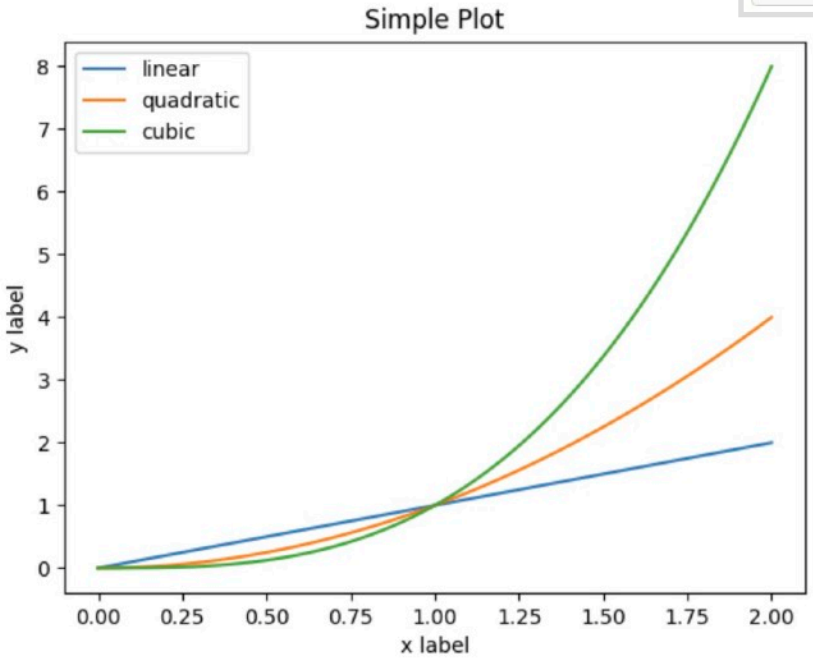

Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line
(line plot)
Y axis label
Markers
(scatter plot)
Figure
Axes
Line
(line plot)
Spines
Minor tick label
X axis label
Made with http://matplotlib.org

```python
x = np.linspace(0, 2, 100)

# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
```

=

or (pyplot-style)

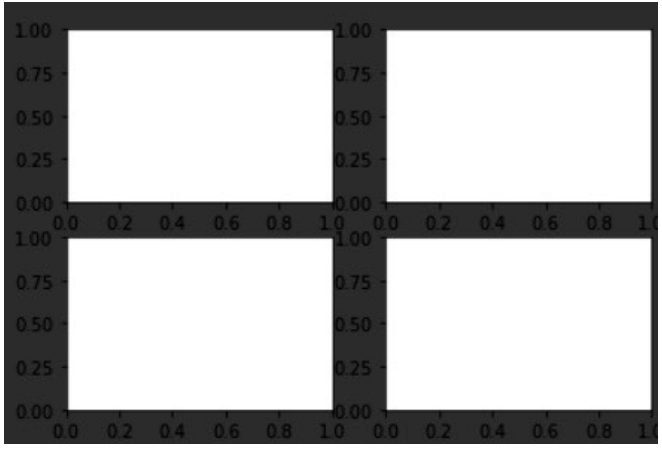
```python
x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
```



```
fig = plt.figure()  # an empty figure with no Axes
```

```
fig, ax = plt.subplots()  # a figure with a single Axes
```

```
fig, axs = plt.subplots(2, 2)  # a figure with a 2x2 grid of Axes
```



Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

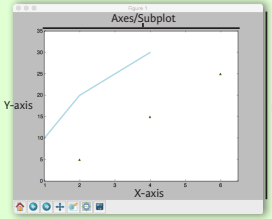
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Legends

```
>>> ax.set(title='An Example Axes',
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')
```

Set a title and x-and y-axis labels
No overlapping plot elements

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```

Manually set x-ticks
Make y-ticks longer and go in and out

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()
```

Adjust the spacing between subplots
Fit subplot(s) in to the figure area

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Make the top axis line for a plot invisible
Move the bottom axis line outward

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them
Draw unconnected points, scaled or colored
Plot vertical rectangles (constant width)
Plot horiontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and o

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes
Plot a 2D field of arrows
Plot a 2D field of arrows

Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

Plot a histogram
Make a box and whisker plot
Make a violin plot

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

Colormapped or RGB arrays

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)
```

Pseudocolor plot of 2D array
Pseudocolor plot of 2D array
Plot contours
Plot filled contours
Label a contour plot

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

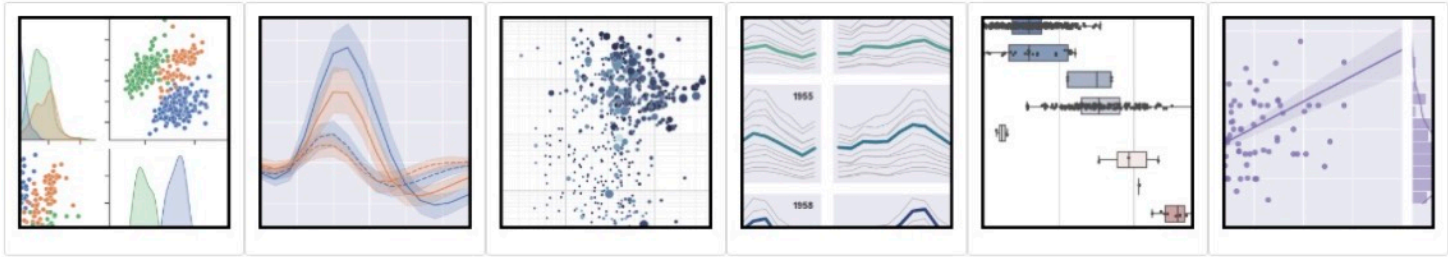
Clear an axis
Clear the entire figure
Close a window

DataCamp
Learn Python for Data Science Interactively

https://seaborn.pydata.org

seaborn 0.11.1 Gallery Tutorial API Site Page Search

seaborn: statistical data visualization



Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

For a brief introduction to the ideas behind the library, you can read the introductory notes. Visit the installation page to see how you can download the package and get started with it. You can browse the example gallery to see what you can do with seaborn, and then check out the tutorial and API reference to find out how.

To see the code or report a bug, please visit the GitHub repository. General support questions are most at home on stackoverflow or discourse, which have dedicated channels for seaborn.

Contents

Features

- Relational: API | Tutorial
- Distribution: API | Tutorial
- Categorical: API | Tutorial
- Regression: API | Tutorial
- Multiples: API | Tutorial
- Style: API | Tutorial
- Color: API | Tutorial

Back to top

Aix*Marseille université
Socialement engagée

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6))
```
Create a figure and one subplot

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```
(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters

Return a dict of params or use with `with` to temporarily set the style

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                    font_scale=1.5,
                    rc={"lines.linewidth":2.5})
```
Set context to `"talk"`
Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
>>> sns.color_palette("husl")
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui)
```
Define the color palette
Use with `with` to temporarily set palette
Set your own color palette

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
```
Subplot grid for plotting conditional relationships

```
>>> sns.factorplot(x="pclass",
                   y="survived",
                   hue="sex",
                   data=titanic)
```
Draw a categorical plot onto a Facetgrid

```
>>> sns.lmplot(x="sepal_width",
               y="sepal_length",
               hue="species",
               data=iris)
```
Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
```
Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions

```
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
```
Grid for bivariate plot with marginal univariate plots

```
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```
Plot bivariate distribution

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species",
                  y="petal_length",
                  data=iris)
```
Scatterplot with one categorical variable

```
>>> sns.swarmplot(x="species",
                  y="petal_length",
                  data=iris)
```
Categorical scatterplot with non-overlapping points

Bar Chart

```
>>> sns.barplot(x="sex",
                y="survived",
                hue="class",
                data=titanic)
```
Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

```
>>> sns.countplot(x="deck",
                  data=titanic,
                  palette="Greens_d")
```
Show count of observations

Point Plot

```
>>> sns.pointplot(x="class",
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                           "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```
Show point estimates and confidence intervals as rectangular bars

Boxplot

```
>>> sns.boxplot(x="alive",
                y="age",
                hue="adult_male",
                data=titanic)
```
Boxplot

```
>>> sns.boxplot(data=iris,orient="h")
```
Boxplot with wide-form data

Violinplot

```
>>> sns.violinplot(x="age",
                   y="sex",
                   hue="survived",
                   data=titanic)
```
Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",
                y="sepal_length",
                data=iris,
                ax=ax)
```
Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
                        kde=False,
                        color="b")
```
Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```
Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                      "Sex")
>>> h.set(xlim=(0,5),
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```
Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax,yticks=[0,5])
>>> plt.tight_layout()
```
Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

```
>>> plt.show()
>>> plt.savefig("foo.png")
>>> plt.savefig("foo.png",
                transparent=True)
```
Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

Also see Matplotlib

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```
Clear an axis
Clear an entire figure
Close a window

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

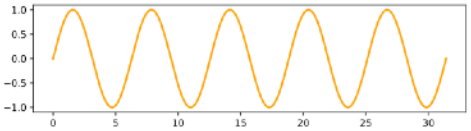
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

4 Observe



Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```

```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```

```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```

```
X = np.arange(5)
Y = np.random.uniform(0,1,5)
ax.errorbar(X, Y, Y/4)
```

```
Z = np.random.normal(0,1,(100,3))

ax.boxplot(Z)
```

Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```

```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```

```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```

```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```

Organize

You can plot several data on the the same figure but you can also split a figure in several subplots (named *Axes*):

```
X = np.linspace(0,10,100)
Y1, Y1 = np.sin(X), np.cos(X)
ax.plot(X, Y1, Y2)
```

```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```


```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```

Explore

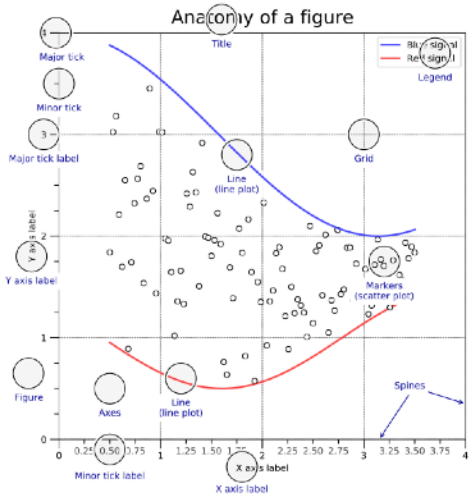
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Matplotlib for intermediate users

A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.



Figure, axes & spines

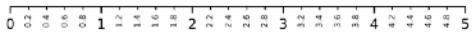
```
fig, axs = plt.subplots((3,3))
axs[0,0].set_facecolor("#ddddff")
axs[2,2].set_facecolor("#ffffdd")
```

```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#ddddff")
```

```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```

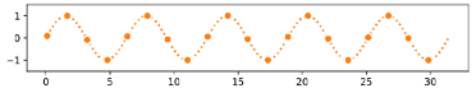
Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x',which='minor',rotation=90)
```



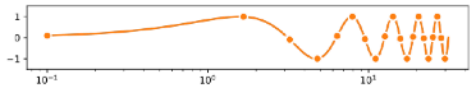
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=25, mec="1.0")
```



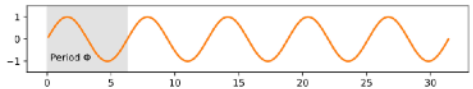
Scales & Projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=25, mec="1.0")
```



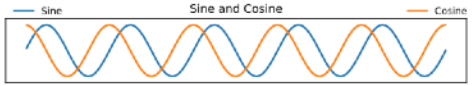
Text & Ornaments

```
ax.fill_betweenx([-1,1],[0],[2*np.pi])
ax.text(0, -1, r" Period $\Phi$")
```



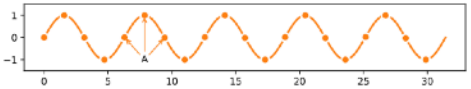
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0,1,1,.1),ncol=2,
          mode="expand",  loc="lower left")
```



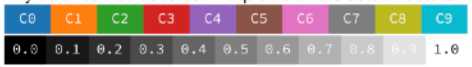
Annotation

```
ax.annotate("A", (X[250],Y[250]),(X[250],-1),
  ha="center", va="center",arrowprops =
  {"arrowstyle" : "->", "color": "C1"})
```



Colors

Any color can be used but Matplotlib offers sets of colors:



Size & DPI

Consider a square figure to be included in a two-columns A4 paper with 2cm margins on each side and a column separation of 1cm. The width of a figure is (21 - 2*2 - 1)/2 = 8cm. One inch being 2.54cm, figure size should be 3.15×3.15 in.

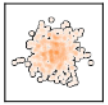
```
fig = plt.figure(figsize=(3.15,3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Matplotlib tips & tricks

Transparency

Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density and multiple scatter plots can be used to delineate a frontier.

```
X = np.random.normal(-1,1,500)
Y = np.random.normal(-1,1,500)
ax.scatter(X, Y, 50, "0.0", lw=2) # optional
ax.scatter(X, Y, 50, "1.0", lw=0) # optional
ax.scatter(X, Y, 40, "C1",  lw=0, alpha=0.1)
```



Rasterization

If your figure is made of a lot graphical elements such as a huge scatter, you can rasterize them to save memory and keep other elements in vector format.

```
X = np.random.normal(-1, 1, 10_000)
Y = np.random.normal(-1, 1, 10_000)
ax.scatter(X, Y, rasterized=True)
fig.savefig("rasterized-figure.pdf", dpi=600)
```

Offline rendering

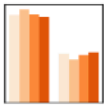
Use the Agg backend to render a figure directly in an array.

```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure())
... # draw som stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

Range of continuous colors

You can use colormap to pick a range of continuous colors.

```
X = np.random.randn(1000, 4)
cmap = plt.get_cmap("Blues")
colors = [cmap(i) for in in [.2,.4,.6,.8]]

ax.hist(X, 2, histtype='bar', color=colors)
```



Text outline

Use text outline to make text more visible.

```
import matplotlib.patheffects as fx
text = ax.text(0.5, 0.1, "Label")
text.set_path_effects([
  fx.Stroke(linewidth=3, foreground='1.0'),
  fx.Normal()])
```

Multiline plot

You can plot several lines at once using None as separator.

```
X,Y = [], []
for x in np.linspace(0, 10*np.pi, 100):
  X.extend([x, x, None]), Y.extend([0, sin(x), None])
ax.plot(X, Y, "black")
```

Dotted lines

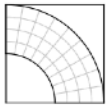
To have rounded dotted lines, use a custom `linestyle` and modify `dash_capstyle`.

```
ax.plot([0,1], [0,0], "C1",
        linestyle = (0, (0.01, 1)), dash_capstyle="round")
ax.plot([0,1], [1,1], "C1",
        linestyle = (0, (0.01, 2)), dash_capstyle="round")
```

Combining axes

You can use overlaid axes with different projections.

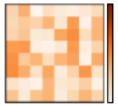
```
ax1 = fig.add_axes([0,0,1,1],
                   label="cartesian")
ax2 = fig.add_axes([0,0,1,1],
                   label="polar",
                   projection="polar")
```



Colorbar adjustment

You can adjust colorbar aspect when adding it.

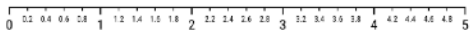
```
im = ax.imshow(Z)

cb = plt.colorbar(im,
        fraction=0.046, pad=0.04)
cb.set_ticks([])
```



Taking advantage of typography

You can use a condensed face such as Roboto Condensed to save space on tick labels.

```
for tick in ax.get_xticklabels(which='both'):
    tick.set_fontname("Roboto Condensed")
```



Getting rid of margins

Once your figure is finished, you can call `tight_layout()` to remove white margins. If there are remaining margins, you can use the `pdfcrop` utility (comes with TeX live).

Hatching

You can achieve nice visual effect with thick hatch patterns.

```
cmap = plt.get_cmap("Oranges")
plt.rcParams['hatch.color'] = cmap(0.2)
plt.rcParams['hatch.linewidth'] = 8
ax.bar(X, Y, color=cmap(0.6), hatch="/" )
```

Read the documentation

Matplotlib comes with an extensive documenation explaining every detals of each command and is generally accompanied by examples with. Together with the huge online gallery, this documentation is a gold-mine.

matplotlib

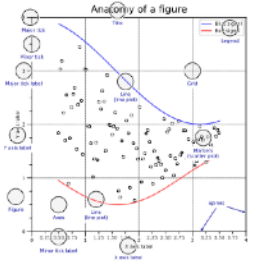
Cheat sheet

Version 3.2

Quick start

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X,Y,color='C1')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](cols,rows,…)
fig, axs = plt.subplots(3,3)

G = gridspec(cols,rows,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax=d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/matplotlib
- gitter.im/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X],Y,[fmt],…)
X, Y, fmt, color, marker, linestyle

scatter(X,Y,…)
X, Y, [s]izes, [c]olors, markers, cmap

bar[h](x,height,…)
x, height, width, bottom, align, color

imshow(Z,[cmap],…)
Z, cmap, interpolation, extent, origin

contour[f]([X],[Y],Z,n…)
X, Y, Z, levels, colors, extent, origin

quiver([X],[Y],U,V,…)
X, Y, U, V, C, units, angles

pie(X,[explode],…)
Z, explode, labels, colors, radius

text(x,y,text,…)
x, y, text, va, ha, size, weight, transform

fill[_between][x]( … )
X, Y1, Y2, color, where
```

Advanced plots

```
step(X,Y,[fmt],…)
X, Y, fmt, color, marker, where

boxplot(X,…)
X, notch, sym, bootstrap, widths

errorbar(X,Y,xerr,yerr,…)
X, Y, xerr, yerr, fmt

hist(X, bins, …)
X, bins, range, density, weights

violinplot(D,…)
D, positions, widths, vert

barbs([X],[Y], U, V, …)
X, Y, U, V, C, length, pivot, sizes

eventplot(positions,…)
positions, orientation, lineoffsets

hexbin(X,Y,C,…)
X, Y, C, gridsize, bins

xcorr(X,Y,…)
X, Y, normed, detrend
```

Scales

```
ax.set_[xy]scale(scale,…)
```

linear — any values
log — values > 0
symlog — any values
logit — 0 < values < 1

Projections

```
subplot(…,projection=p)
```

p='polar'
p='3d'

```
p=Orthographic()
from cartopy.crs import Cartographic
```

Lines

```
linestyle or ls
capstyle or dash_capstyle
```

"butt" "round" "projecting"

Markers

```
markevery
```

10 [0, -1] |25, 5| [0, 25, -1]

Colors

```
'Cn'
'x'
'name'
(R,G,B[,A])
'#RRGGBB[AA]'
'x,y'
```

Colormaps

```
plt.get_cmap(name)
```

Uniform: viridis, magma, plasma
Sequential: Greys, YlOrBr, Wistia
Diverging: Spectral, coolwarm, RdGy
Qualitative: tab10, tab20
Cyclic: twilight

Tick locators

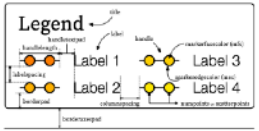
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
ticker.NullFormatter()
ticker.FixedFormatter(['', '2', '1', ''])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

```
ax.legend(…)
handles, labels, loc, title, frameon
```




```
ax.colorbar(…)
mappable, ax, cax, orientation
```

```
ax.annotate(…)
text, xy, xytext, xycoords, textcoords, arrowprops
```

Event handling

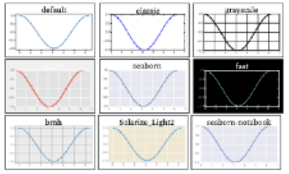
```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0,2*np.pi,100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
  line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
  plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.patch.set_alpha(0)
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(list)
ax.set_[xy]ticklabels(list)
ax.set_[sup]title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

ax.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
fig.patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

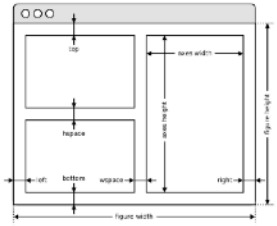
ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

Ten Simple Rules

1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure
4. Captions Are Not Optional
5. Do Not Trust the Defaults
6. Use Color Effectively
7. Do Not Mislead the Reader
8. Avoid "Chartjunk"
9. Message Trumps Beauty
10. Get the Right Tool

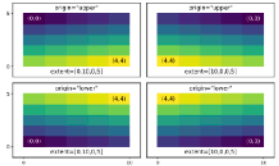
Axes adjustements

plt.subplot_adjust(...)



Extent & origin

ax.imshow(extent=..., origin=...)



Text alignments

ax.text(..., ha=... , va=..., ...)

Text parameters

ax.text(..., family=... , size=..., weight = ...)
ax.text(..., fontproperties = ...)

The quick brown fox — xx-large (1.72)
The quick brown fox — x-large (1.44)
The quick brown fox — large (1.20)
The quick brown fox — medium (1.00)
The quick brown fox — small (0.83)
The quick brown fox — x-small (0.69)
The quick brown fox — xx-small (0.58)

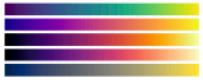
The quick brown fox jumps over the lazy dog — black (900)
The quick brown fox jumps over the lazy dog — bold (700)
The quick brown fox jumps over the lazy dog — semibold (600)
The quick brown fox jumps over the lazy dog — normal (400)
The quick brown fox jumps over the lazy dog — ultralight (100)

The quick brown fox jumps over the lazy dog — monospace
The quick brown fox jumps over the lazy dog — serif
The quick brown fox jumps over the lazy dog — sans
The quick brown fox jumps over the lazy dog — cursive

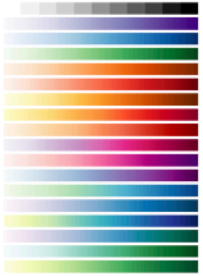
The quick brown fox jumps over the lazy dog — italic
The quick brown fox jumps over the lazy dog — normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG — small-caps
The quick brown fox jumps over the lazy dog — normal

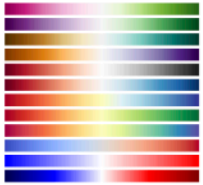
Uniform colormaps



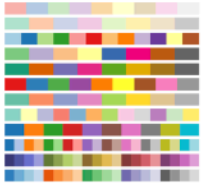
Sequential colormaps



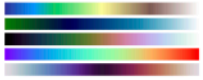
Diverging colormaps



Qualitative colormaps

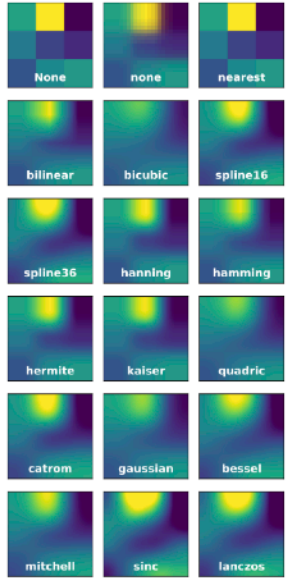


Miscellaneous colormaps

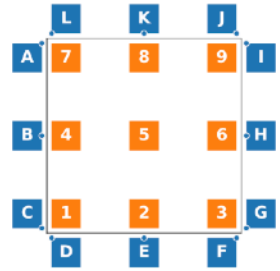


Color names

Image interpolation



Legend placement

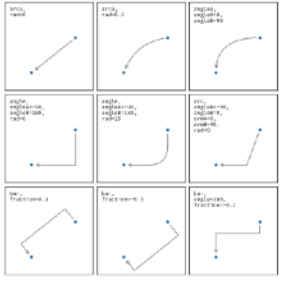


ax.legend(loc="string", bbox_to_anchor=(x,y))

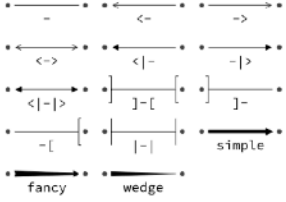
1: lower left — 2: lower center — 3: lower right
4: left — 5: center — 6: right
7: upper left — 8: upper center — 9: upper right

A: upper right / (-1, 9) — B: right / (-1,.5)
C: lower right / (-1,.1) — D: upper left / (.1,-.1)
E: upper center / (.5,.1) — F: upper right / (.9,.1)
G: lower left / (1.1,.1) — H: left / (1.1,.5)
I: upper left / (1.1,.9) — J: lower right / (.9,1.1)
K: lower center / (.5,1.1) — L: lower left / (.1,1.1)

Annotation connection styles



Annotation arrow styles



How do I ...

... resize a figure?
→ fig.set_size_inches(w,h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure?
→ ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_xticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.set_[xy]ticks(rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0),1,1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R+0.5870*G+0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                          slow
plot(X, Y, marker="o", ls="")          fast

for i in range(n): plot(X[i])          slow
plot(sum([x+[None] for x in X],[]))    fast

cla(), imshow(...), canvas.draw()      slow
im.set_data(...), canvas.draw()        fast
```

Beyond Matplotlib

Seaborn: Statistical Data Visualization
Cartopy: Geospatial Data Processing
yt: Volumetric data Visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A Grammar of Graphics for Python

NUMFOCUS
OPEN CODE = BETTER SCIENCE