ESILV
LÉONARD DE VINCI

ÉCOLE D'INGÉNIEURS
PARIS-LA DÉFENSE

Python For Data Analysis

Imen Ouled Dlala

imen.ouled_dlala@devinci.fr
Bureau: L405

General Plan

- The basics of python for data science
- Data Science modules
  - Pandas, Numpy, Scipy
- Data analysis and visualization
  - Seaborn, Matplotlib, Bokeh
- Webscrapping
- Machine learning and Datasets
  - Sklearn, tensorflow
- API Django / Flask

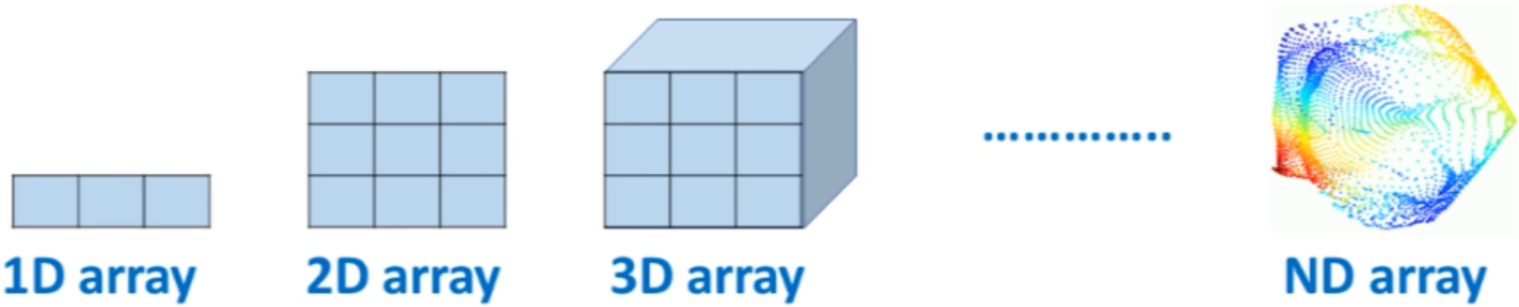
Scientific modules : Plan

1. Numpy
2. Pandas

Numpy

Numpy

Numerical Python Library It is a library that provides a high-performance multidimensional array object, and tools for working with these arrays.
ND Array



Numpy

1Darray

List =

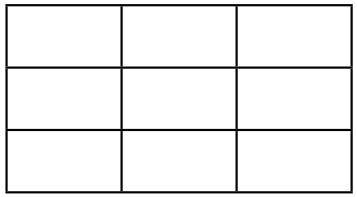
index	0	1	2	3

ndarray =

index	0	1	2	3

ndarray is much faster and better for scientific calculation.

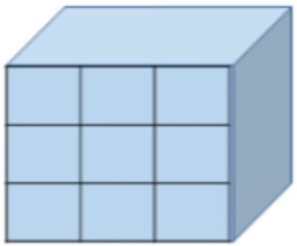
2Darray



	A	B	C	D
1	Segment	Country	Product	Discount Band
3	Government	Canada	Carretera	None
4	Government	Germany	Carretera	None
5	Midmarket	France	Carretera	None
6	Midmarket	Germany	Carretera	None
7	Midmarket	Mexico	Carretera	None
8	Government	Germany	Carretera	None
9	Midmarket	Germany	Montana	None
10	Channel Partners	Canada	Montana	None
11	Government	France	Montana	None
12	Channel Partners	Germany	Montana	None
13	Midmarket	Mexico	Montana	None
14	Enterprise	Canada	Montana	None

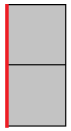
Numpy

3Darray

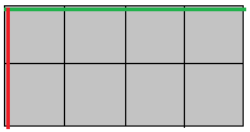


Numpy

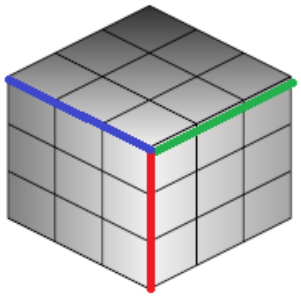
ndarray.shape



ndim = 1
shape=(2,)



ndim =2
shape=(2, 4)



ndim= 3
shape=(3, 3, 3)

shape is a tuple !

shape[0]=2
shape[1]=4

Numpy

```
import numpy as np
```

```
# np.array(object)
# Create a rank 1 array
A = np.array([1,2,3])
```

```
array([1, 2, 3])
```

```
A.ndim
A.shape
A.size
```

```
1
(3,)
3
```

```
# Create a rank 2 array
B = np.array([[1,2,3], [4,5,6]])
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
B.ndim
B.shape
B.size
```

```
2
(2,3)
6
```

Numpy

```
#np.zeros(shape)
C = np.zeros((3,4))
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
#np.ones(shape)
D =np.ones((3,4))
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

```
#np.full(shape, value)
E =np.full((3,3),7)
```

```
array([[7, 7, 7],
       [7, 7, 7],
       [7, 7, 7]])
```

```
#np.linspace(start, end, quantity)
np.linspace(0,5,3)
```

```
array([0. , 2.5, 5. ])
```

```
#np.arange(start, end, step)
np.arange(0,3,0.5)
```

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5])
```

Numpy

numpy.random.randn(d0, d1, ..., dn)

Return a sample (or samples) from the "standard normal" distribution

```
np.random.randn(3, 4)
```

```
array([[ 1.17966391, -0.5956569 ,  0.65984529, -1.19343449],
       [ 0.455953  ,  1.26902337, -1.15777501, -0.36264622],
       [-1.29907221, -0.15721062, -0.33505458, -0.31872119]])
```

```
np.random.seed(0)
np.random.randn(3, 4)
```

```
array([[ 1.76405235,  0.40015721,  0.97873798,  2.2408932 ],
       [ 1.86755799, -0.97727788,  0.95008842, -0.15135721],
       [-0.10321885,  0.4105985 ,  0.14404357,  1.45427351]]
```

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

Return random integers from the "discrete uniform" distribution of the specified dtype in the "half-open" interval [low, high).

```
np.random.randint(0,10,[5,5])
```

```
array([[8, 1, 5, 9, 8],
       [9, 4, 3, 0, 3],
       [5, 0, 2, 3, 8],
       [1, 3, 3, 3, 7],
       [0, 1, 9, 9, 0]])
```

Numpy : Manipulation

```
#F= np.hstack((array1, array2))
#shape[0] equal
F= np.hstack((C,E))
```

```
array([[0., 0., 0., 0., 7., 7., 7.],
       [0., 0., 0., 0., 7., 7., 7.],
       [0., 0., 0., 0., 7., 7., 7.]])
```

```
#G = np.vstack((array1, array2))
#shape[1] equal
G = np.vstack((C,D))
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

```
#H = np.concatenate((array1, array2), axis=0)
H = np.concatenate((C, D), axis=0)
```

Numpy

```
I = H.reshape((4,6))
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1.]])
```

```
J = np.array([1,2,3,4])
J.shape
```

(4,)

```
J= J.reshape((J.shape[0]), 1)
J.shape
```

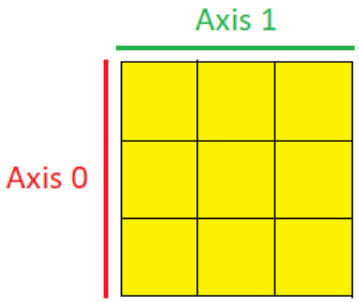
(4,1)

```
J= J.squeeze()
J.shape
```

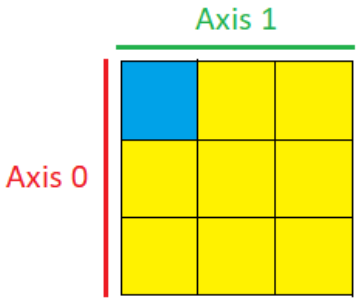
(4,)

```
Flatten
i =i.ravel()
i
```

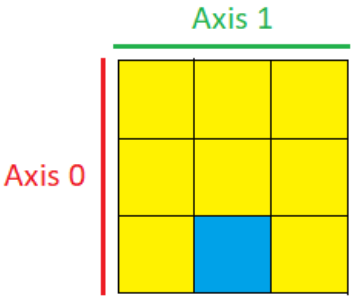
Numpy : Indexing



A[line,column]

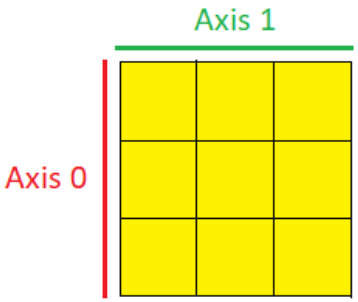


A[0,0]

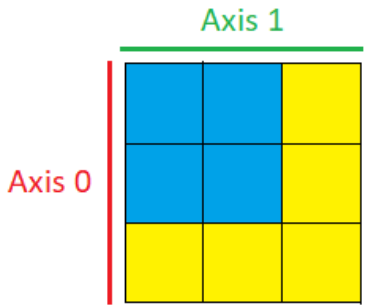


A[2,1]

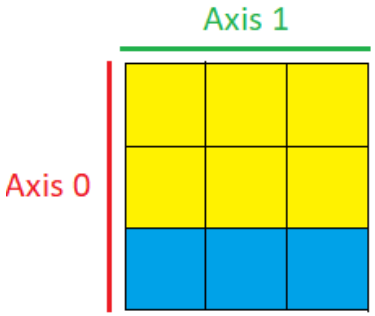
Numpy :Slicing



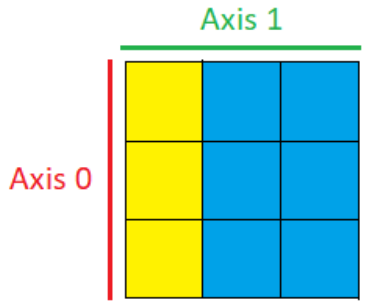
A[start :end,start :end]



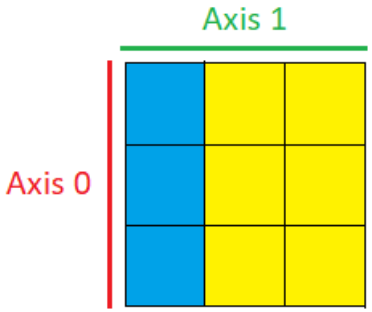
A[0 :2,0 :2]



A[2, :]
A[2]

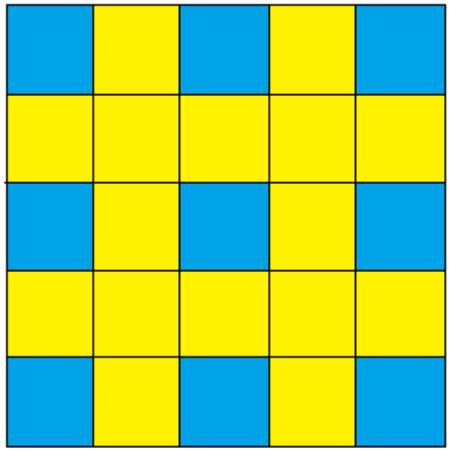


A[:,1 :]
A[:,-2 :]



A[:,0]

Numpy :Slicing



A[start :end :step,start :end :step]
A[: :2, : :2]

Numpy : Boolean Indexing

$A =$

5	4	5
4	4	4
5	4	5

$A < 5 \rightarrow$

F	T	F
T	T	T
F	T	F

$A[A < 5] = 10 \rightarrow$

5	10	5
10	10	10
5	10	5

Numpy

Statistics :

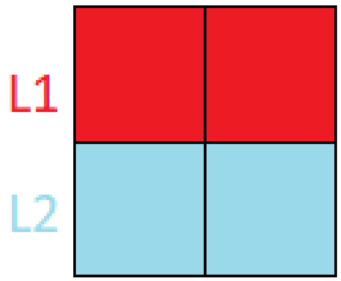
https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.statistics.html

Mathematical functions :

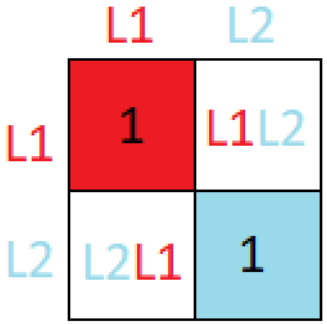
https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html

Numpy :Statistics

A =



np.corrcoef(A)




```
np.random.seed(0)
A= np.random.randint(0,10,[2,3])
A
```

```
array([[5, 0, 3],
       [3, 7, 9]])
```

```
np.corrcoef(A)
```

```
array([[ 1.        , -0.56362148],
       [-0.56362148,  1.        ]])
```

```
np.corrcoef(A)[0,1]
```

```
-0.5636214801906779
```

Numpy :Statistics

A=

5	0	3
3	7	9

np.unique(A,return_counts=True)

Elements=

0	3	5	7	9

Repetition=

1	2	1	1	1

np.unique(A,return_counts=True)

```
(array([0, 3, 5, 7, 9]), array([1, 2, 1, 1, 1], dtype=int64))
```

Numpy : Statistics

A.argsort() : Returns the indices that would sort this array, without modifying the array A.

```
values, counts = np.unique(A, return_counts = True)
```

```
counts.argsort()
```

```
array([0, 2, 3, 4, 1], dtype=int64)
```

```
for i, j in zip (values[counts.argsort()], counts[counts.argsort()]) :
    print(f'.value {i} appears {j}')
```

```
.value 0 appears 1
.value 5 appears 1
.value 7 appears 1
.value 9 appears 1
.value 3 appears 2
```

Numpy : NaN Corrections

NaN : Not a Number

```
A= np.random.randn(5,5)
A[0,1]=np.nan
A[2,3]=np.nan
```

```
array([[-0.7827755 ,         nan, -0.34518616, -0.88180055, -0.44265324],
       [-0.5409163 , -1.32322737, -0.11279892,  0.90734594,  0.81526991],
       [ 0.22909795, -1.02617878,  0.47752547,         nan, -0.73145824],
       [-1.60540226,  0.98947618,  0.11081461, -0.38093141,  0.11495917],
       [ 0.34531264, -1.73495876,  1.65835111,  2.29977152, -0.47113526]])
```

```
np.isnan(A)
```

```
array([[False,  True, False, False, False],
       [False, False, False, False, False],
       [False, False, False,  True, False],
       [False, False, False, False, False],
       [False, False, False, False, False]])
```

```
np.isnan(A).sum() #Output : 2
```

Numpy

np.nanmean() : Compute the arithmetic mean along the specified axis, ignoring NaNs

np.mean(A)

nan

np.nanmean(A)

-0.10571731518539351

A [np.isnan(A)] = 0

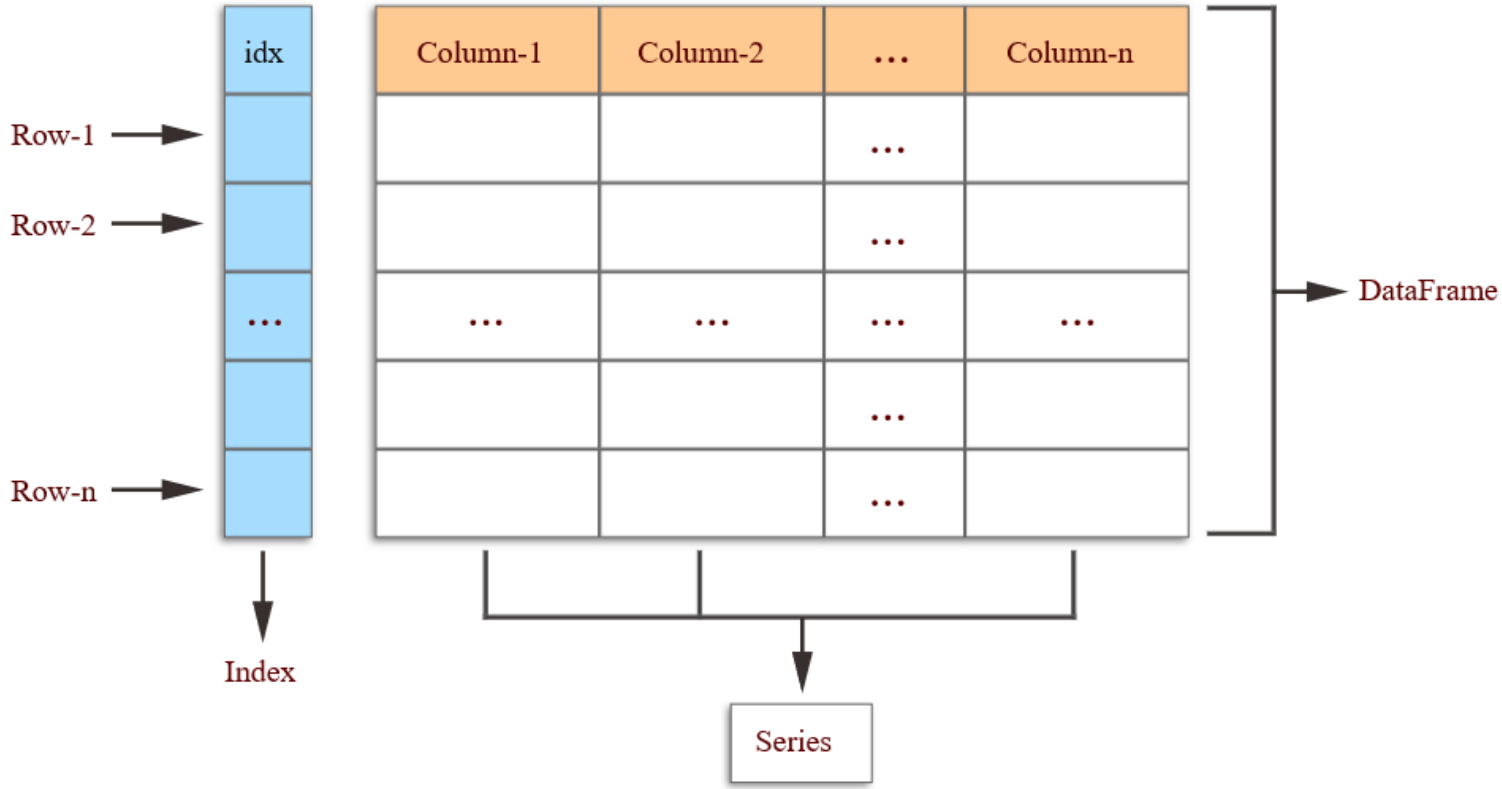
```
array([[-0.7827755 ,  0.        , -0.34518616, -0.88180055, -0.44265324],
       [-0.5409163 , -1.32322737, -0.11279892,  0.90734594,  0.81526991],
       [ 0.22909795, -1.02617878,  0.47752547,  0.        , -0.73145824],
       [-1.60540226,  0.98947618,  0.11081461, -0.38093141,  0.11495917],
       [ 0.34531264, -1.73495876,  1.65835111,  2.29977152, -0.47113526]])
```

Pandas

Pandas

Pandas Panel Datas

pandas is a Python package providing fast, flexible, and expressive data structures.



Pandas

Pandas Basic commands :

```
import pandas as pd
```

Pandas version :

```
print(pd.__version__)
```

Create a DataFrame from a dictionary

```
import pandas as pd
job = [ "journaliste", "capitaine"]
names = ["tintin", "hadock" ]
personage={"job" : job, "names" : names}

df= pd.DataFrame(personage)
df
```

	job	names
0	journaliste	tintin
1	capitaine	hadock

Pandas

Selection

df[col]	Returns column with label col as Series
df[[col1, col2]]	Returns columns as a new DataFrame
s.iloc[0]	Selection by position
s.loc['index_one']	Selection by index
df.iloc[0, :]	First row
df.iloc[0,0]	First element of first column

Viewing \Inspecting Data

df.head(n)	First n rows of the DataFrame
df.tail(n)	Last n rows of the DataFrame
df.shape	Number of rows and columns
df.info()	Index, Datatype and Memory information
df.describe()	Summary statistics for numerical columns
s.value_counts(dropna=False)	View unique values and counts
df.apply(pd.Series.value_counts)	Unique values and counts for all columns

Pandas

Data Cleaning

df.columns = ['a','b','c']	Rename columns
pd.isnull()	Checks for null Values, Returns Boolean Arrray
pd.notnull()	Opposite of pd.isnull()
df.dropna()	Drop all rows that contain null values
df.dropna(axis=1)	Drop all columns that contain null values
df.dropna(axis=1,thresh=n)	Drop all rows that have less than n non null values
df.fillna(x)	Replace all null values with x
df.rename(columns=lambda x : x + 1)	Mass renaming of columns
df.set_index('column_one')	Change the index
df.rename(index=lambda x : x + 1)	Mass renaming of index

Pandas

Viewing data

```
print ("shape          =", df.shape)
print ("Columns        =", df.columns)
print ("job            =", df["job"])
print ("Type column =", type(df["job"]))
```

```
shape        = (2, 2)
Columns      = Index(['job', 'names'], dtype='object')
job          = 0    journaliste
1      capitaine
Name: job, dtype: object
Type column = <class 'pandas.core.series.Series'>
```

Adding a column

```
age=[25, 50]
df['age']= age
df
```

	job	names	age
0	journaliste	tintin	25
1	capitaine	hadock	50

Pandas

Manipulation of index

```
df.index=['heros', 'secondaire']
df.index.name='type'
df
```

type	job	names	age
heros	journaliste	tintin	25
secondaire	capitaine	hadock	50

creation of a time index

```
dates = pd.date_range(start = '01/01/1931' , end = '01/02/1931')
df.index = dates
df
```

	job	names	age
1931-01-01	journaliste	tintin	25
1931-01-02	capitaine	hadock	50

Pandas

```
pds= pd.DataFrame({'title' :['Au congo','Cigares pharaon'],
                   'origin' :['France','France']})
# Affectation of the same time index
pds.index= dates
pds
```

	title	origin
1931-01-01	Au congo	France
1931-01-02	Cigares pharaon	France

```
# Use of the index to add a column
df['title']=pds['title']
df
```

	job	names	age	title
1931-01-01	journaliste	tintin	25	Au congo
1931-01-02	capitaine	hadock	50	Cigares pharaon

```
# Adding lines one after other
new_df_2 =df.append(pds)
new_df_2
```

	job	names	age	title	origin
1931-01-01	journaliste	tintin	25.0	Au congo	NaN
1931-01-02	capitaine	hadock	50.0	Cigares pharaon	NaN
1931-01-01	NaN	NaN	NaN	Au congo	France
1931-01-02	NaN	NaN	NaN	Cigares pharaon	France

```
new_df_1=df.merge(pds)
new_df_1
```

	job	names	age	title	origin
0	journaliste	tintin	25	Au congo	France
1	capitaine	hadock	50	Cigares pharaon	France

Pandas

```
new_df_3=pd.concat([df,pds],axis=1)
new_df_3
```

	job	names	age	title	title	origin
1931-01-01	journaliste	tintin	25	Au congo	Au congo	France
1931-01-02	capitaine	hadock	50	Cigares pharaon	Cigares pharaon	France

Pandas

Datasets : `https://github.com/absabry/python_dataset`

Import librairies

```
import pandas as pd
import os
```

Import the Data

```
path='C :\\Users\io201817 \Desktop \Data \cyclistes.csv'
```

Check that the path exists

```
assert(os.path.isfile(path))
```

Create the dataframe

```
df=pd.read_csv(path)
```

Pandas

df.head()

	id	sexe	sportivite	age	sur_velo
0	101	F	-0.285718	66	True
1	102	H	2.219441	37	True
2	103	F	2.637251	49	True
3	104	F	1.413551	33	True
4	105	H	-1.331255	36	True

Counting adults vs minors

```
adults = df.age>18
```

```
minors =df.age<18
```

```
nb_adults=df[adults]["age"].count()
nb_minors=df[minors]["age"].count()
```

```
nb_adults, nb_minors
```

```
(949, 36)
```

Pandas

Add a column in the df by assigning a column name to a list of values

```
cyclistes=df
```

```
cyclistes["is_adult"]=adults
```

```
df.head()
```

	id	sexe	sportivite	age	sur_velo	is_adult
0	101	F	-0.285718	66	True	True
1	102	H	2.219441	37	True	True
2	103	F	2.637251	49	True	True
3	104	F	1.413551	33	True	True
4	105	H	-1.331255	36	True	True

Count the average number of "sportive" by sex

```
cyclistes[minors]["age"].mean()
```

```
16.055555555555557
```

```
cyclistes[adults]["age"].mean()
```

```
50.04531085353003
```

```
for majority in [minors,adults] :
    print("majority %s"
    %(cycliste[majority]["age"].mean()))
```

```
majority 16.055555555555557
majority 50.04531085353003
```

```
cyclistes.groupby(["is_adult"])["age"].mean()
```

```
is_adult
False    16.627451
True     50.045311
Name: age, dtype: float64
```

Pandas

```
pandas.pivot_table(data, values=None, index=None, columns=None,
aggfunc='mean', fill_value=None, margins=False, dropna=True,
margins_name='All', observed=False)
```

data		The dataframe to pivot
values		The column to aggregate (if blank, will aggregate all numerical values)
index		The column to group data by
columns		The key to group by
aggfunc	'mean'	The function to use to aggregate
fill_value		Value to replace missing values with
margins	False True	Add a row and column for totals
dropna		To choose to not include columns where all entries are NaN
margins_name	'All'	Name of total row / column
observed	False	Only for categorical data – if True will only show observed values for categorical groups

End

Good Lecture !

This course is inspired from Romain Jouin that I thank for sharing