



Introduction Python DataFrame



Python



- C'est quoi ??
 - Langage de **programmation**
 - Très populaire pour la **data analyse**
- Éléments de base à maîtriser

Variables

Informations

```
s = 2.0
```

```
tab = [1.20, 2.0, 3.0 ]
```

Expressions

Actions

```
s = s * 2
```

```
print(s)
```

Bibliothèques

Pandas
Matplotlib
Sklearn

Structures de
contrôle

Itératives

Conditionnelles

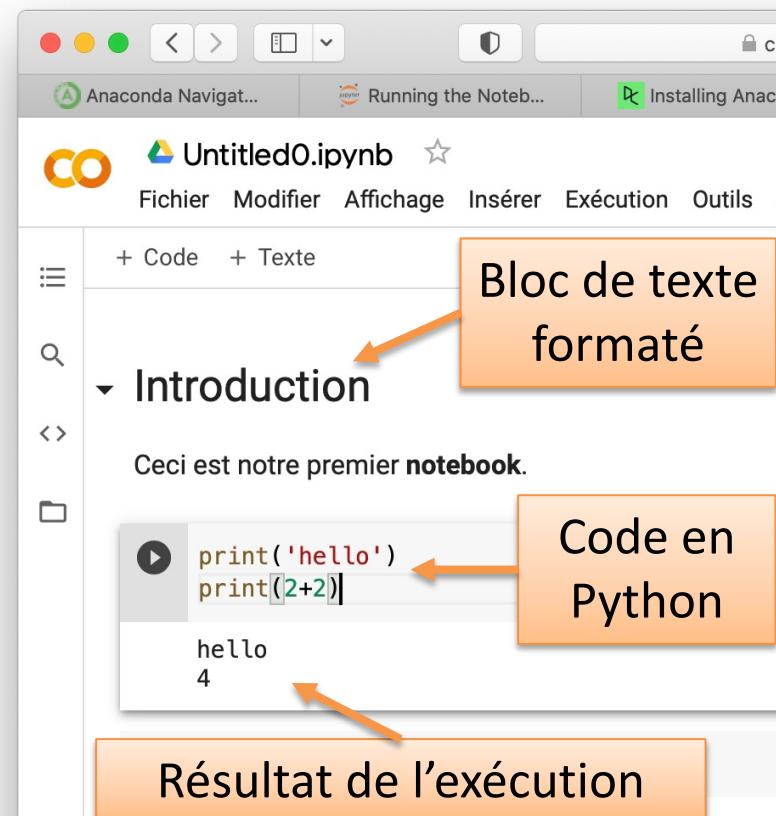
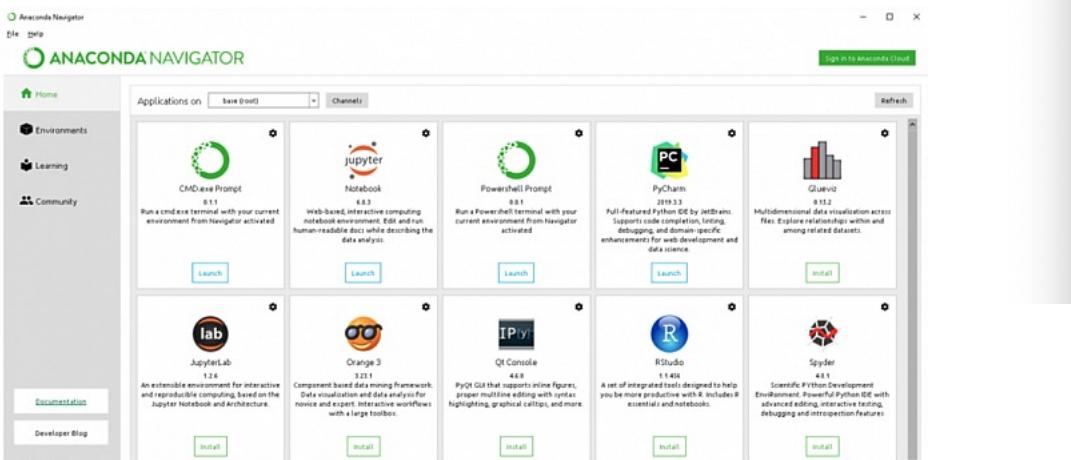
while
for

if

Python

- Comment on fait ? **Notebooks**

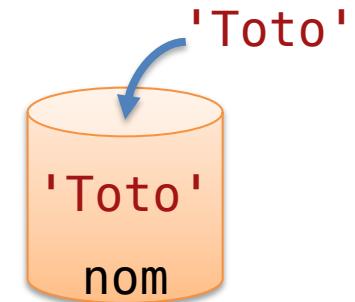
- Documents « actifs » : **blocs de texte + blocs de code**
- Expérimenter et documenter**
 - Texte** en format « **markdown** »
 - Code Python** exécuté en mode « **itératif** »
- Sur le Web (**navigateur**)
- Anaconda Navigator → Jupyter Notebook (ou Jupyter Lab)



Python : Variables

- **Variables**

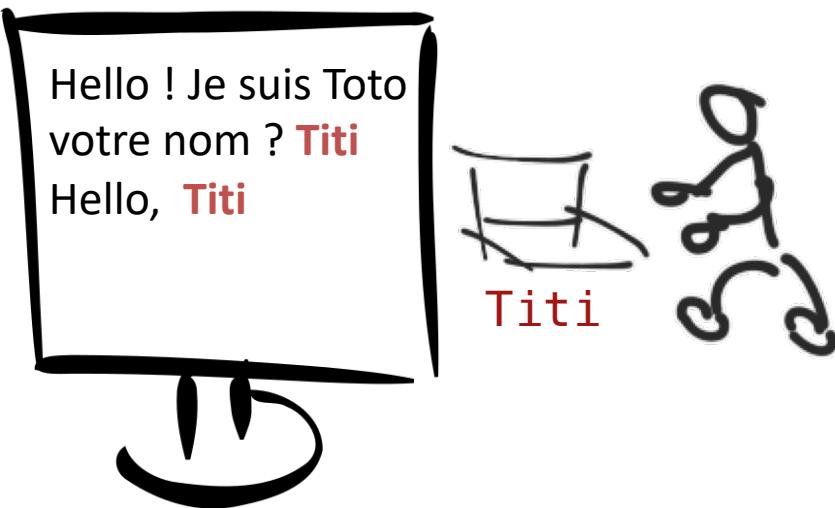
- Permettent de stocker de l'information



Affectation :

On attribue une valeur à la variable

```
nom = 'Toto'  
print('Hello ! Je suis', nom)
```



print (val , val , val ...)

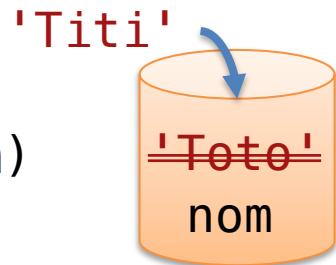
On affiche une valeur (une variable ou un message)

input (message)

permet de lire un texte à partir du clavier

```
nom = input('votre nom ? ')
```

```
print('Hello, ', nom)
```



Opérations
(*print, input...*)

Permettent la réalisation de certaines actions



UNIVERSITÉ PARIS 1

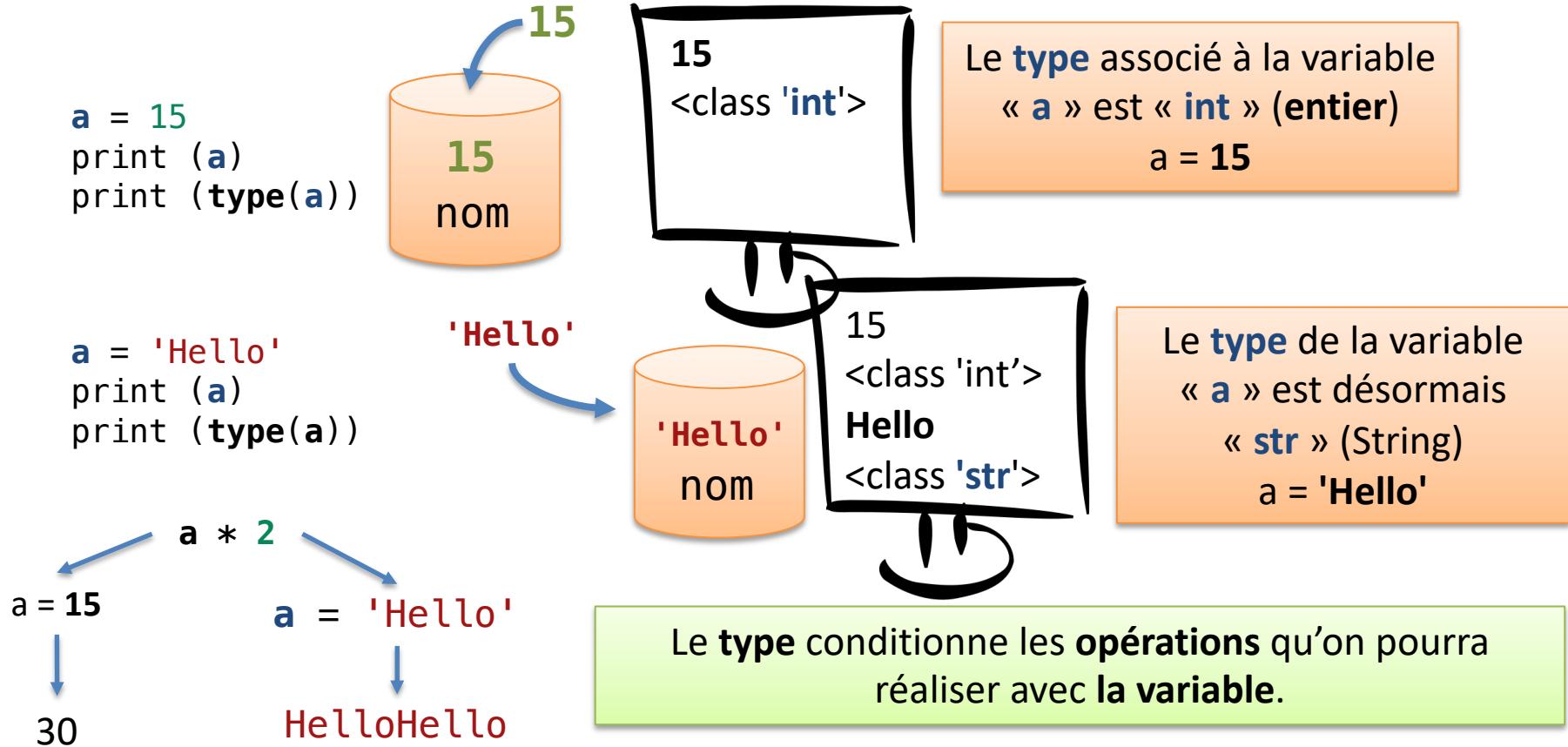
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

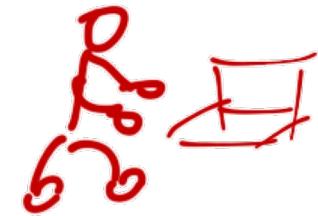
Python : Variables

• Variables

- Les variables abritent des données d'un certain **type** (entier, chaîne caractères...)
- Le « **type** » de la variable varie en fonction de son contenu



Hands On !



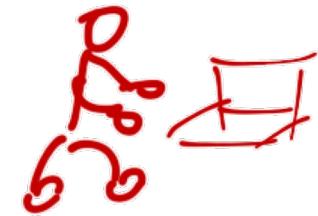
- Ouvrir Anaconda Navigator
- Lancer JupyterLab
- Démarrer un nouveau Notebook Python

The image shows a Mac desktop with the following windows open:

- Anaconda Navigator**: A grid of application icons including Datalore, IBM Watson Studio Cloud, JupyterLab, Notebook, PyCharm Community, Qt Console, Spyder, and VS Code. Each icon has a "Launch" button.
- JupyterLab launcher**: A window showing a list of files in the directory `Documents/Dropbox/Codigos/Python/cours/`. The list includes:

Name	Last Modified
files	4 hours ago
df_apply.py	2 months ago
draftsExos...	3 hours ago
exceptions...	2 months ago
filtre_lamb...	2 months ago
hello.py	7 months ago
intro_dataf...	5 days ago
note.ipynb	5 days ago
notesSerie...	13 days ago
- JupyterLab browser interface**: A window titled "localhost" showing the JupyterLab interface with a "Notebook" and "Console" tab. The "Notebook" tab is selected, showing a list of files in the same directory as the launcher.
- File browser**: A standard OS X file browser window showing the same directory structure as the JupyterLab launcher.

Hands On !



• Exercice : Prise en main Notebook

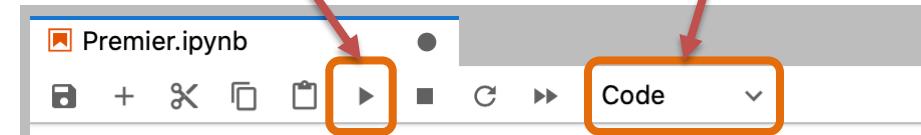
– Ajouter un bloc de **Markdown** et y ajouter un texte

Choisir le type de bloc
(markdown ou code)

– Ajouter un **bloc de code Python** :

- Créer une variable nommée « **a** »
- Lui affecter la valeur **2.5**
- Faire **a * 5**
- **Exécuter le bloc**

Exécuter le bloc



– Sur un nouveau bloc de code :

- Affecter la valeur '**toto**' à la variable « **a** »
- Faire **a * 5**
- **Exécuter le bloc**

Premiers pas

Mon premier exercice avec les Notebooks Python...

```
[1]: a = 2.5  
      a * 5
```

```
[1]: 12.5
```

```
[ ]:
```



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Python : Variables

• Listes et dictionnaires

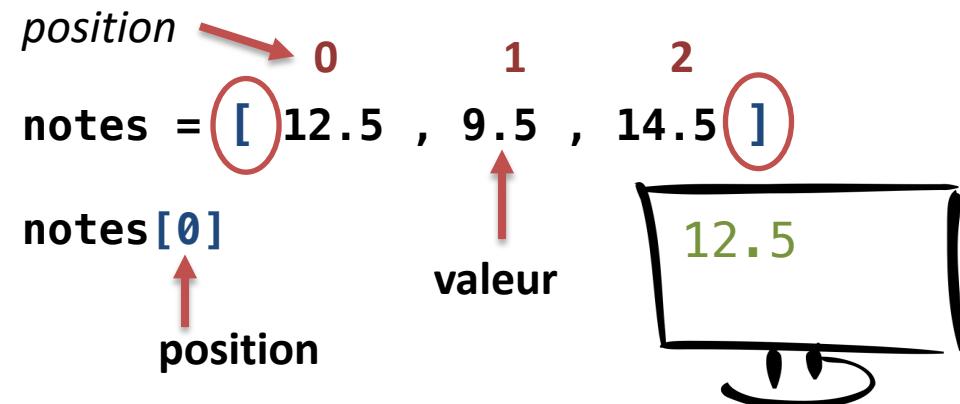
- Deux structures des données très utilisées
- **Dictionary** : collection de type « clé: valeur » (**indexée**)
- **List** : collection **ordonnée** et **modifiable** de valeurs (tableau)

variable clé : valeur

```
eleve = { 'nom': 'Toto',  
          'matr': 100,  
          'formation': 'marketing' }
```

`eleve['nom']`

Un dictionnaire est une collection
de paires clé - valeur



Une liste est une collection
ordonnée (par position)
de valeurs

Python : Variables

• Listes et dictionnaires

- **Dictionnaires** : on peut **récupérer** (ou **insérer**) des valeurs par leur **clé**
 - `dico [clé]`
- **Listes** : on peut **récupérer** des valeurs par leur **position**
 - `liste [position]`
 - Opérateur « slice » : `liste [début : fin]`

On ajoute un nouvelle paire clé : valeur

```
eleve ['notes'] = notes  
print(eleve)
```

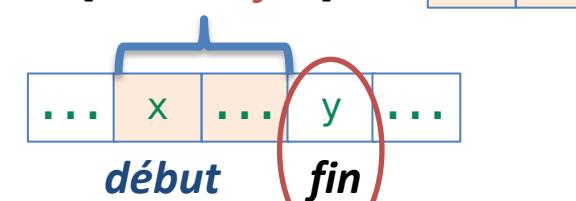
```
{'nom': 'Toto', 'matr': 100,  
'formation': 'marketing',  
'notes': [12.5, 9.5, 14.5]}
```

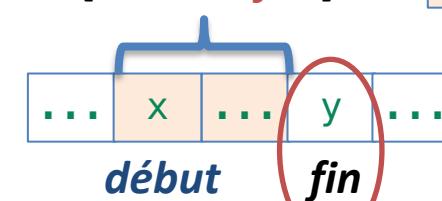
notes[0:2]

On récupère les deux premières valeurs

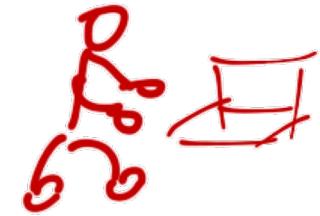
12.5	9.5	14.5
0	1	2

[12.5, 9.5]

liste [*début* : *fin*] → 



Hands On !



- **Exercice : Premiers pas avec Python**
 - Sur le notebook **Jupyter** créé précédemment...
 - Créer un **dictionnaire** nommé « **eleve** » avec les informations suivantes
 - Nom : Toto
 - Matr : 100
 - Formation : ‘marketing’
 - Afficher le nom de l’élève
 - Créer une **liste** nommée « **notes** » avec les valeurs suivantes
 - 12.5 , 9.5 et 14.5
 - Afficher la première note
 - Insérer la liste de notes dans le dictionnaire élève
 - Afficher le dictionnaire

```
Premier.ipynb
[7]: eleve = { 'nom':'Toto', 'matr': 100, 'formation': 'marketing'}
       print (eleve)
{'nom': 'Toto', 'matr': 100, 'formation': 'marketing'}

[9]: eleve['nom']
[9]: 'Toto'

Une liste est une collection ordonnée de valeurs. Chaque valeur occupe une place dans l'ordre.

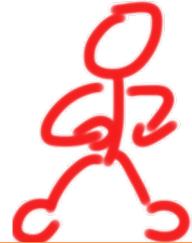
[10]: notes = [ 12.5 , 9.5 , 14.5 ]
       print (notes)
[12.5, 9.5, 14.5]

[12]: notes[0]
[12]: 12.5

[13]: eleve['notes'] = notes
       print(eleve)
{'nom': 'Toto', 'matr': 100, 'formation': 'marketing', 'notes': [12.5, 9.5, 14.5]}
```

Python : bibliothèques

- Une bibliothèque est un **catalogue de fonctions**
 - Ensemble des **codes prêts** et disponibles pour **usage**
 - Disponibles sur des sujets très variés :
 - Data analyse (**pandas**), Machine Learning (**sklearn**)...
- Pour utiliser une bibliothèque, on doit d'abord l'importer



On ne réinvente pas la roue !

On importe toute la bibliothèque

```
import math as mt
```

```
mt.sqrt(notes[1])
```

as → alias « mt » (pour faire court)

notes →

12.5	9.5	14.5
0	1	2

3.082207001484488

On utilise la fonction « sqrt » de la bibliothèque « math »



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Python : Pandas

- **Pandas**

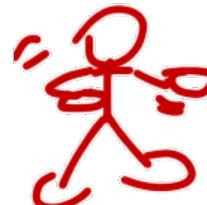
- Importante bibliothèque dédiée à l'analyse de données
- Deux structures de données majeures : **Series** et **DataFrame**

Series

séries de valeurs indexées
(c.a.d. séquence *<key, value>*)
Très utiles pour les **séries temporelles**

index	value
'Suisse'	8
'France'	70
'USA'	320
'Chine'	1200

Structure la plus utilisée pour la data analyse



DataFrame : structure tabulaire, où les données sont organisées en **colonnes** et **indexées**

index	colonnes	
	'Habitants'	'Capital'
'Suisse'	8	'Geneve'
'France'	70	'Paris'
'USA'	320	'Washington'
'Chine'	1200	'Pequin'

données



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Python : DataFrame

```
import pandas as pd
```

```
monDF = pd.DataFrame ( { 'Habitants' : [ 8, 70, 320, 1200 ],  
                         'Capital' : ['Geneve', 'Paris', 'Washington', 'Pequin'] },  
                         index=['Suisse', 'France', 'USA', 'Chine'])
```

listes avec les valeurs d'index

```
monDF['Capital']
```

'Capital'
'Geneve'
'Paris'
'Washington'
'Pequin'

On peut récupérer les données d'une colonne
monDF ['nom colonne']

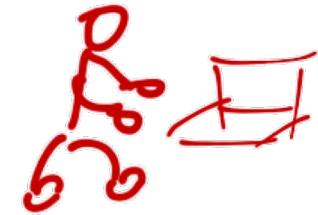
	index	colonnes
'Suisse'	8	'Geneve'
'France'	70	'Paris'
'USA'	320	'Washington'
'Chine'	1200	'Pequin'

```
monDF.loc['France']
```

Ou les données d'une ligne avec l'opération loc
monDF.loc ['valeur index']

'Habitants'	'Capital'
70	'Paris'

Hands On !



• Exercice : Prise en main d'un DataFrame

- Sur son Notebook Jupyter...
- Créer une variable « monDF » avec le DataFrame suivant

```
DataFrame ( { 'Habitants' : [ 8, 70, 320, 1200 ],  
            'Capital': ['Geneve', 'Paris', 'Washington', 'Pequin'] },  
            index=['Suisse', 'France', 'USA', 'Chine'])
```

- Utiliser les opérations « **info** » et « **describe** » pour obtenir des informations sur le DataFrame

- monDF.info()
- monDF.describe()

```
[3]: monDF.info()
```

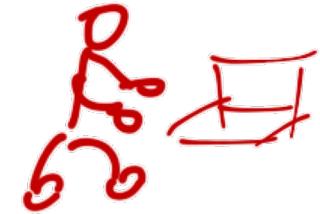
```
<class 'pandas.core.frame.DataFrame'>  
Index: 4 entries, Suisse to Chine  
Data columns (total 2 columns):  
 #   Column      Non-Null Count  Dtype    
 ---    
 0   Habitants    4 non-null      int64
```

```
[4]: monDF.describe()
```

```
[4]:
```

Habitants	
count	4.000000
mean	399.500000
std	550.443155
min	8.000000
25%	54.500000

Hands On !



- Exercice : trouver une information dans un DF avec « loc »
 - Avec le DataFrame « monDF » qu'on vient de créer
 - Trouver la capitale de la France
 - `monDF.loc ['France'] ['Capital']`
index colonne
 - Trouver les pays qui ont plus de 100 mil. habitants
 - `monDF.loc [monDF ['Habitants'] > 100]`
condition sur une colonne
 - Trouver les pays qui ont + de 100 mil. ET - de 1000 mil. habitants
 - `monDF.loc [(monDF ['Habitants'] > 100) & (monDF ['Habitants'] < 1000)]`
(condition sur une colonne) ET (condition sur une colonne)



Chaine de traitement avec les DataFrames

Fichiers sur

<https://github.com/mkirschpin/CoursPython>

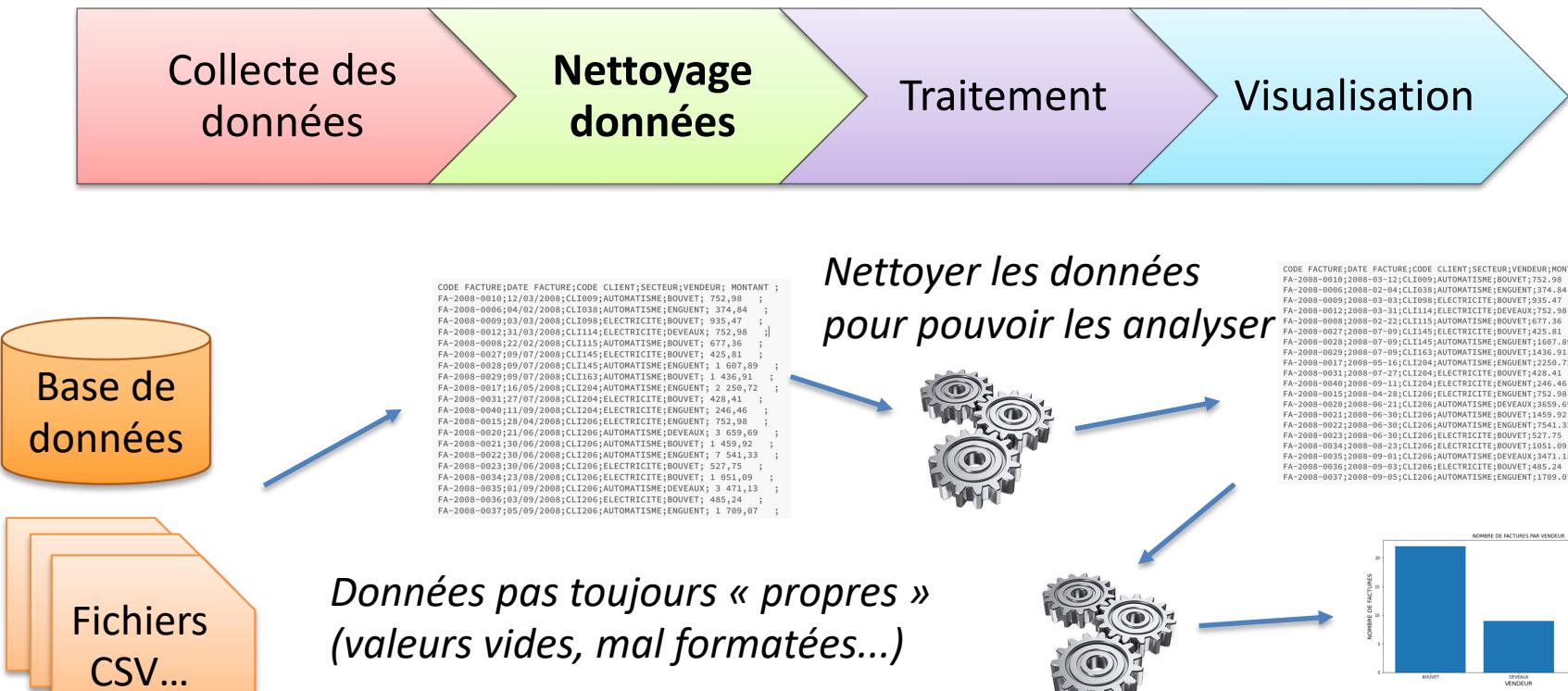


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Étapes de traitement des données

- Les projets d'analyse de données suivent un ensemble d'étapes bien précises



analyser les données

Trouver la meilleure visualisation



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Lecture d'un fichier

- Pandas permet de lire le **contenu d'un fichier** vers un **DataFrame**
- Différents **formats** : **CSV**, Excel, JSON, SQL...
- Opérations **read_xxx** : **read_csv**, **read_excel**, **read_sql**...

Fichier CSV

CODE FACTURE;DATE FACTURE;CODE CLIENT; MONTANT

FA-2008-0010;2008-03-12;CLI009; 752.98

FA-2008-0006;2008-02-04;CLI038; 374.84

Données séparées par ;
(séparateur)

```
import pandas as pd  
ventes = pd.read_csv('files/VentesAgenceU.csv',  
                     delimiter=';',  
                     header=[0],  
                     index_col=[0])
```

Autres options possibles

Par ex. si index est une date :

```
parse_dates=True  
dayfirst=True
```

Fichier à lire

Séparateur

Ligne(s) d'entête

Index



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT

ventes.info()

```
<class 'pandas.core.frame.DataFrame'>
Index: 50 entries, FA-2008-0010 to nan
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   DATE FACTURE    47 non-null    object  
 1   CODE CLIENT     47 non-null    object  
 2   SECTEUR         47 non-null    object  
 3   VENDEUR         47 non-null    object  
 4   MONTANT         47 non-null    object  
 5   Unnamed: 6       0 non-null    float64
dtypes: float64(1), object(5)
memory usage: 2.7+ KB
```

Lecture d'un fichier

Une fois le fichier lu, il faut vérifier les informations :

- monDF.info ()
- monDF.describe ()
- monDF.head (n)
- monDF.tail (n)

info : informations sur le DF

[]:

head : premières lignes du DF

[2]:

ventes.head()

[2]:

	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT	Unnamed: 6
--	--------------	-------------	---------	---------	---------	------------

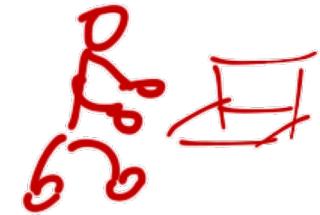
CODE FACTURE						
--------------	--	--	--	--	--	--

FA-2008-0010	12/03/2008	CLI009	AUTOMATISME	BOUVET	752,98	NaN
FA-2008-0006	04/02/2008	CLI038	AUTOMATISME	ENGUENT	374,84	NaN
FA-2008-0009	03/03/2008	CLI098	ELECTRICITE	BOUVET	935,47	NaN
FA-2008-0012	31/03/2008	CLI114	ELECTRICITE	DEVEAUX	752,98	NaN
FA-2008-0008	22/02/2008	CLI115	AUTOMATISME	BOUVET	677,36	NaN

Signes de problèmes :

- Mauvais types de données
- NaN / NaT

Hands On !



- **Exercice : Lecture et vérification des données**
 - Créer un nouveau Notebook pour cette activité
 - Télécharger le fichier « ***files/VentesAgenceU.csv*** » qui se trouve sur <http://www.kirschpm.fr/cours/PythonDataScience/>
 - Réaliser la lecture du « ***VentesAgenceU.csv*** » sur un **DataFrame**
 - Utiliser ensuite **info** pour afficher les informations de ce **DataFrame**
 - Utiliser **head** et **tail** pour afficher les 10 premières/dernières lignes de données
 - Utiliser **describe** pour une analyse rapide de ces données

Les données sont-elles « propres » ?



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Problèmes constatés

[4]: `ventes.head(10)`

	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT	Unnamed: 6
CODE FACTURE						
FA-2008-0010	12/03/2008	CLI009	AUTOMATISME	BOUVET	752,98	NaN
FA-2008-0006	04/02/2008	CLI038	AUTOMATISME	ENGUENT	374,84	NaN
...		NaN
FA-2008-0029	09/07/2008	CLI163	AUTOMATISME	BOUVET	1_436,91	NaN
FA-2008-0017	16/05/2008	CLI204	AUTOMATISME	ENGUENT	2_250,72	NaN
FA-2008-0031	27/07/2008	CLI204	ELECTRICITE	BOUVET	428,41	NaN

*Colonne vide**Chiffres mal formatés*[3]: `ventes.describe()`[3]: **Unnamed: 6**

count	0.0
mean	NaN
std	NaN
min	NaN
25%	NaN
50%	NaN
75%	NaN
max	NaN

	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT	Unnamed: 6
CODE FACTURE						
FA-2008-0014	18/04/2008	CLI312	AUTOMATISME	ENGUENT	917,38	NaN
...		NaN
FA-2008-0005	26/01/2008	CLI520	ELECTRICITE	BOUVET	739,83	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN

Lignes vides



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Comment nettoyer les données ?

- Différentes opérations peuvent être nécessaires pour nettoyer les données
 - Supprimer les lignes/colonnes contenant des cellules vides***
 - Remplacer les cellules vides
 - Corriger les types des données***
 - Remplacer les « , » par des « . » dans les nombres
 - Supprimer les espaces vides des chiffres et des noms des colonnes
 - ...



Les opérations nécessaires varient en fonction de la qualité du Dataset et de l'analyse



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Traiter les cases vides

- Les cases vides sont indiquées par des valeurs spéciales : **NaN** et **NaT**

- On peut soit les remplacer par une autre valeur : **fillna**

– `df2.fillna (value = { 'A' : 0 } , inplace = True)`

valeurs à remplacer

Dictionnaire { colonne : valeur }

*modifie le DataFrame (True)
ou non (False)*

- Soit les supprimer avec **dropna**

– Différentes options : **inplace=True**, **axis='index' / 'columns'** , **how = all**

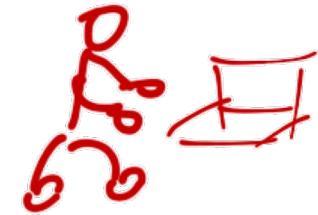
*Supprime les lignes ou les colonnes
contenant les valeurs vides*

*Supprime si toutes
les valeurs sont vides*

– `monDF.dropna(axis='index', inplace = True)`

– `monDF.dropna(axis='columns' , how ='all', inplace = True)`

Hands On !



- **Exercice : Prise en main opération dropna**
 - Sur notre **premier** notebook
 - Créer un nouveau **DataFrame** avec les données suivantes

```
[3]: df2 = pd.DataFrame({'A':[0, pd.NA, 1, -2],  
                      'B':['Titi', 'Toto', 'Tata', 'Tutu'],  
                      'C':['2020-01-01', '2021-02-02', pd.NaT, '2019-12-30']})  
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 4 entries, 0 to 3  
Data columns (total 3 columns):  
 #   Column  Non-Null Count  Dtype     
---  --  -----  -----  
 0   A      3 non-null    object    
 1   B      4 non-null    object    
 2   C      3 non-null    datetime64[ns]  
dtypes: datetime64[ns](1), object(2)  
memory usage: 224.0+ bytes
```

Afficher le **DataFrame** originel pour vérifier son état **entre les opérations**

- Vérifier la création du DataFrame avec l'opération **info**
- Observer les effets des opérations suivantes :
 - **df2.dropna(axis='columns')**
 - **df2.dropna(axis='index')**
 - **df2.dropna(inplace=True)**



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

```
[4]: df2.dropna(axis='columns')
```

```
[4]:
```

	B	Supprime les colonnes avec des valeurs vides
0	Titi	
1	Toto	
2	Tata	
3	Tutu	

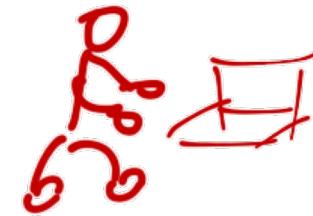
```
[5]: df2.dropna(axis='index')
```

```
[5]:
```

	A	B	C
0	0	Titi	2020-01-01
3	-2	Tutu	2019-12-30

Supprime les
lignes avec des
valeurs vides

Hands On !



```
[7]: df2
```

```
[7]:
```

	A	B	C
0	0	Titi	2020-01-01
1	<NA>	Toto	2021-02-02
2	1	Tata	NaT
3	-2	Tutu	2019-12-30

DataFrame
original

Supprime les lignes avec vides
On modifie le DataFrame

```
[9]: df2.dropna(inplace=True)
```

```
[10]: df2
```

```
[10]:
```

	A	B	C
0	0	Titi	2020-01-01
3	-2	Tutu	2019-12-30

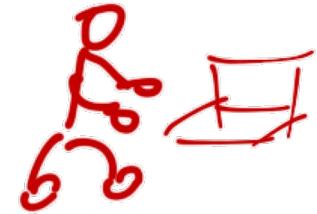


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Hands On !



- **Exercice : nettoyer les valeurs vides**

- Retour sur le DataFrame « ventes » (fichier « *VentesAgenceU.csv* »)
- Supprimer la colonne vide (« **Unnamed: 6** »)
- Supprimer les lignes vides à la fin du fichier
- Utiliser **info** et **tail** pour vérifier vos actions

Différentes possibilités :

- `dropna(axis='columns', how='all', inplace=True)`
- `dropna(axis='index', how='all', inplace=True)`
- `drop(columns=[ventes.columns[5]], inplace=True)`



[12]: ventes.tail(10)

	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT	Unnamed: 6
--	--------------	-------------	---------	---------	---------	------------

CODE FACTURE

FA-2008-0014	18/04/2008	CLI312	AUTOMATISME	ENGUENT	917,38	NaN
FA-2008-0011	21/03/2008	CLI334	ELECTRICITE	BOUVET	348,55	NaN
• • •	• • •	• • •	• • •	• • •	-	• • •
FA-2008-0007	13/02/2008	CLI512	AUTOMATISME	ENGUENT	935,47	NaN
FA-2008-0005	26/01/2008	CLI520	ELECTRICITE	BOUVET	739,83	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN



[19]: ventes.dropna(axis='index', how='all', inplace=True)

[21]: ventes.tail(10)

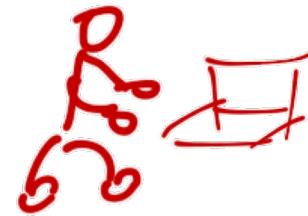
	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT
--	--------------	-------------	---------	---------	---------

CODE FACTURE

FA-2008-0019	12/06/2008	CLI300	ELECTRICITE	BOUVET	1_039,05
FA-2008-0033	14/08/2008	CLI300	AUTOMATISME	DEVEAUX	535,90
• • •	• • •	• • •	• • •	• • •	• • •
FA-2008-0030	18/07/2008	CLI403	ELECTRICITE	DEVEAUX	436,86
FA-2008-0039	09/09/2008	CLI403	AUTOMATISME	BOUVET	3_521,80
FA-2008-0007	13/02/2008	CLI512	AUTOMATISME	ENGUENT	935,47
FA-2008-0005	26/01/2008	CLI520	ELECTRICITE	BOUVET	739,83

[22]: ventes.info()

<class 'pandas.core.frame.DataFrame'>
Index: 47 entries, FA-2008-0010 to FA-2008-0005
Data columns (total 5 columns):
 # Column Non-Null Count Dtype
 --- --
 0 DATE FACTURE 47 non-null object
 1 CODE CLIENT 47 non-null object
 2 SECTEUR 47 non-null object
 3 VENDEUR 47 non-null object
 4 MONTANT 47 non-null object
 dtypes: object(5)
 memory usage: 2.2+ KB





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Traiter les types de données

- Les types de données reconnus à la lecture du fichier ne correspondent pas toujours à la réalité
- Il faut parfois convertir les données vers les **bons formats**
- Plusieurs opérations Panda peuvent le faire
 - `pnd.to_datetime`
 - `pnd.to_numeric`
 - `pnd.to_timedelta`





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Convertir du texte vers des dates

- Convertir un texte (type « **object** ») en date (« **daytime** »)

```
dfDates['Debut'] = pd.to_datetime(dfDates['Debut'], yearfirst=True)
```

to_datetime ne modifie pas les données.
Il faut donc les réaffecter au DataFrame.

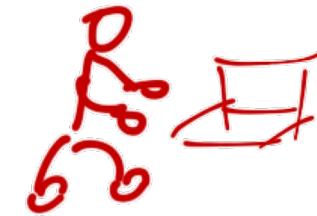
Colonne contenant les valeurs à convertir

Indication *format*
yearfirst : yyyy-mm-dd
dayfirst : dd-mm-yyyy

```
pnd.to_datetime ( dfDates['Fin'], dayfirst=True )  
pnd.to_datetime ( dfDates['Debut'],  
                  yearfirst=True )  
pnd.to_datetime ( dfDates['Examens'],  
                  format='%m%d%Y' )
```

	Debut	Fin	Examens	Semestre
0	2021-09-13	17/12/2021	01042022	S1
1	2022-01-24	23/04/2022	05042022	S2

Hands On !



- **Exercice : Prise en main opération to_datetime**

- Sur notre **premier** notebook
- Créer un nouveau **DataFrame** avec les données suivantes

```
dfDates = pd.DataFrame ({ 'Debut' : ['2021-09-13', '2022-01-24'],
                           'Fin' : ['17/12/2021', '23/04/2022'],
                           'Examens': ['01042022', '05042022'],
                           'Semestre' : ['S1', 'S2'] } )
```

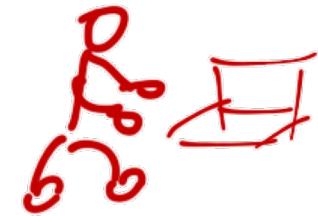
- Afficher les données
- Vérifier le DataFrame avec **info**

	Debut	Fin	Examens	Semestre
0	2021-09-13	17/12/2021	01042022	S1
1	2022-01-24	23/04/2022	05042022	S2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   Debut       2 non-null     object 
 1   Fin         2 non-null     object 
 2   Examens    2 non-null     object 
 3   Semestre   2 non-null     object 
dtypes: object(4)
memory usage: 192.0+ bytes
```

- Observer les effets des opérations suivantes :
 - `pnd.to_datetime(dfDates['Debut'], yearfirst=True)`
 - `pnd.to_datetime(dfDates['Fin'], dayfirst=True)`
 - `pnd.to_datetime(dfDates['Examens'], format='%m%d%Y')`

Hands On !



- **Exercice : convertir les dates**

- Retour sur le DataFrame « ventes » (fichier « **VentesAgenceU.csv** »)
- Utiliser **to_datetime** pour convertir en dates les données de la colonne « **DATE FACTURE** »
- Modifier les données sur le DataFrame « **ventes** »
- Utiliser **info** et **tail** pour vérifier vos actions

CODE FACTURE	DATE FACTURE
FA-2008-0019	12/06/2008
FA-2008-0033	14/08/2008
FA-2008-0042	15/09/2008
FA-2008-0014	18/04/2008
FA-2008-0011	21/03/2008



CODE FACTURE	DATE FACTURE
FA-2008-0019	2008-06-12
FA-2008-0033	2008-08-14
FA-2008-0042	2008-09-15
FA-2008-0014	2008-04-18
FA-2008-0011	2008-03-21

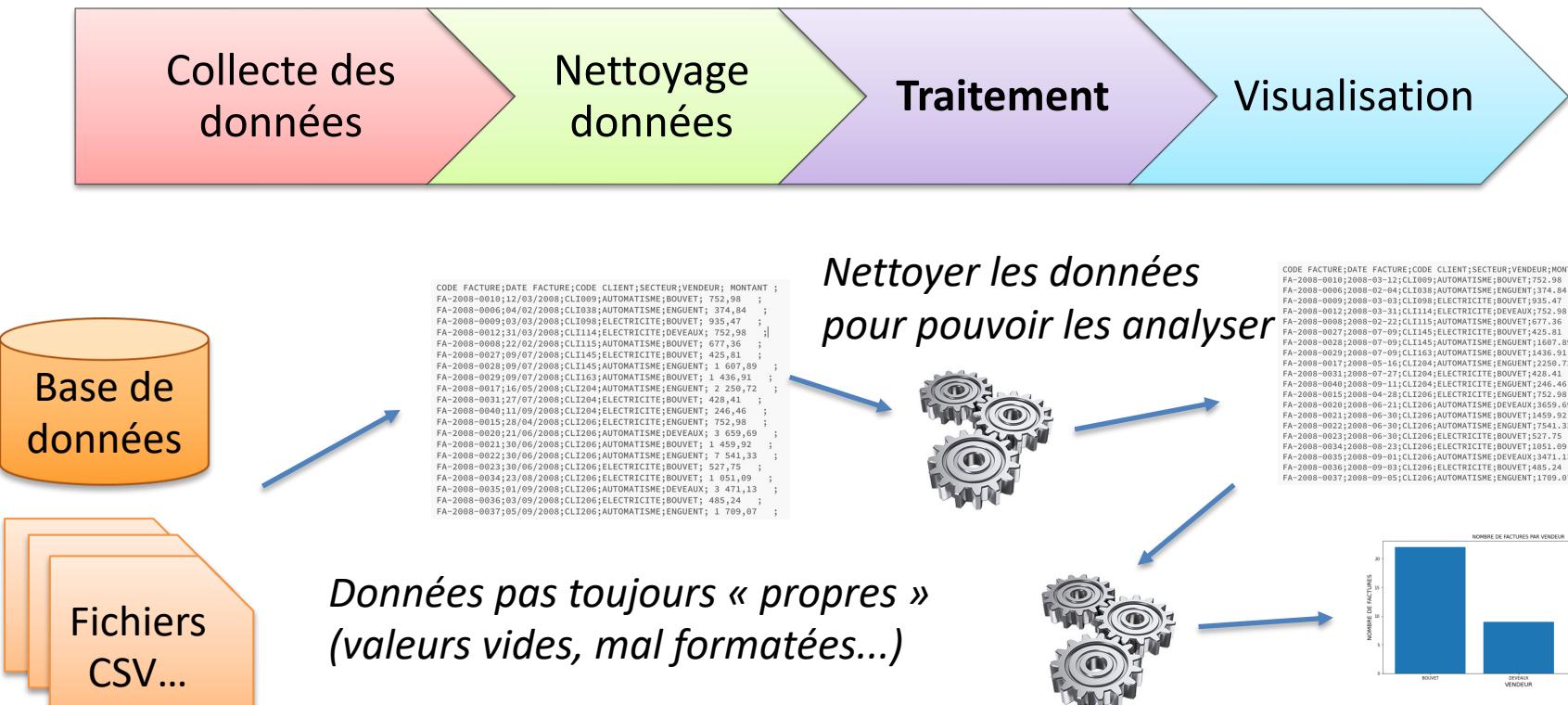


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Étapes de traitement des données

- Les projets d'analyse de données suivent un ensemble d'étapes bien précises





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Traitement

- Le traitement des données est directement lié à l'objectif recherché
 - Que veut-on comprendre / apprendre des données ?
- Exemple : **Business Intelligence**
 - Comprendre ce qui s'est passé
 - *Quel vendeur a vendu le plus ?*
 - *Quel secteur se développe le plus ?*



Traitement simples : Min, Max, Sum, Mean, Median, Count...

- Pandas offre différentes opérations d'analyse simples
 - Somme (**sum**), moyenne (**mean**), médiane (**median**), écart type (**std**), variance (**var**), **min**, **max**, nombre d'éléments (**count**)...

```
ventes.sum( numeric_only=True ) → MONTANT 79958.04
```

dtype: float64

```
ventes.mean( numeric_only=True ) → MONTANT 1701.234894
```

dtype: float64

On ne prend en compte que les valeurs numériques

```
ventes.count() →
```

DATE	FACTURE	47
CODE	CLIENT	47
SECTEUR		47
VENDEUR		47
MONTANT		47

dtype: int64

count :

compte le nombre de lignes ou le nombre de colonnes (axis='columns')

Traitements : Trouver les données

Query, Group By

- Pour réaliser le traitement que l'on souhaite, il faut trouver les **bonnes données**

groupby

Regrouper des données en fonction de(s) colonne(s) choisie(s)

query

Récupérer des données

Alternative à **loc**

Syntaxe semblable à SQL

Requête semblable à SQL :

Colonne Opération Valeur

Opérateurs logiques

and (&)

or (|)

Attention aux `` et aux ''

'nom colonne avec espace'

'valeur string ou date'

ventes.query("MONTANT > 2000 and `DATE FACTURE` >= '2008-09-01'")

	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT
CODE FACTURE					
FA-2008-0035	2008-09-01	CLI206	AUTOMATISME	DEVEAUX	3471.13
FA-2008-0041	2008-09-13	CLI222	AUTOMATISME	DEVEAUX	4374.79
FA-2008-0042	2008-09-15	CLI300	AUTOMATISME	ENGUENT	3667.68
FA-2008-0039	2008-09-09	CLI403	AUTOMATISME	BOUVET	3521.80

Traitements : Trouver les données

Query, Group By

- Pour réaliser le traitement que l'on souhaite, il faut trouver les **bonnes données**

groupby

Regrouper des données en fonction de(s) colonne(s) choisie(s)

query

Récupérer des données

Alternative à **loc**

Syntaxe semblable à SQL

```
ventes.groupby(by='VENDEUR').count()
```

On regroupe les données par vendeur
(**by** = 'colonne')

Puis on compte le nombre de lignes (**count**)

VENDEUR	DATE FACTURE	CODE CLIENT	SECTEUR	MONTANT
BOUVET	22	22	22	22
DEVEAUX	9	9	9	9
ENGUENT	16	16	16	16

Traitements : Trouver les données

Query, Group By

```
ventes.query("VENDEUR == 'DEVEAUX' ")
```

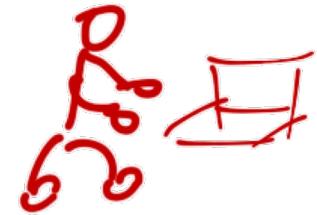
		DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT
	CODE FACTURE					
1	FA-2008-0012	2008-03-31	CLI114	ELECTRICITE	DEVEAUX	752.98
2	FA-2008-0020	2008-06-21	CLI206	AUTOMATISME	DEVEAUX	3659.69
3	FA-2008-0035	2008-09-01	CLI206	AUTOMATISME	DEVEAUX	3471.13
4	FA-2008-0045	2008-09-23	CLI206	AUTOMATISME	DEVEAUX	750.88
5	FA-2008-0024	2008-06-30	CLI209	AUTOMATISME	DEVEAUX	9367.87
6	FA-2008-0041	2008-09-13	CLI222	AUTOMATISME	DEVEAUX	4374.79
7	FA-2008-0004	2008-01-17	CLI235	AUTOMATISME	DEVEAUX	606.66
8	FA-2008-0033	2008-08-14	CLI300	AUTOMATISME	DEVEAUX	535.90
9	FA-2008-0030	2008-07-18	CLI403	ELECTRICITE	DEVEAUX	436.86

```
ventes.groupby(by='VENDEUR').count()
```

	DATE FACTURE	CODE CLIENT	SECTEUR	MONTANT
VENDEUR				
BOUVET	22	22	22	22
<u>DEVEAUX</u>	9	9	9	9
ENGUENT	16	16	16	16

Nombre de valeurs retrouvées dans chaque colonne (count) PAR Vendeur (groupby)

Hands On !



• Exercice : Traitement des données

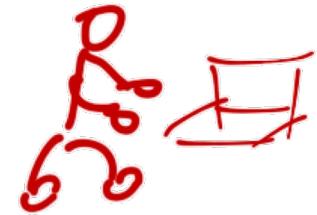
- Créer un **nouveau Notebook** pour cette activité
- Télécharger le fichier « ***files/VentesPropre.csv*** » qui se trouve sur <http://www.kirschpm.fr/cours/PythonDataScience/>
- Réaliser la lecture du « ***VentesPropre.csv*** » sur un **DataFrame**
- Convertir la colonne « **DATE FACTURE** » en date
- Trouver les factures de **plus de 2000 €** datant d'après le **1/9/2008**
- Trouver les factures du **vendeur DEVAUX**
- **Comparer** la sortie de ces deux instructions : **que font-elles ?**

```
ventes.query(" `VENDEUR` == 'DEVEAUX' ") .mean(numeric_only=True)
```

```
ventes.groupby(by='VENDEUR') .mean()
```

Ne pas oublier le
import pandas

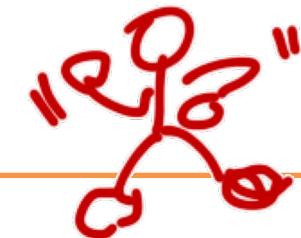
Hands On !



- **Exercice : Traitement des données**
 - Trouver la valeur **moyenne** des factures **par vendeur**
 - Trouver le **nombre** de factures **par vendeur**
 - Trouver la **somme** des factures **par secteur**

Suggestions :

- `import pandas as pdn` *Nouveau fichier,
Nouveau import*
- `pnd.read_csv('VentesPropre.csv', delimiter=';', header=[0], index_col=[0])`
- `pnd.to_datetime(ventes['DATE FACTURE'], dayfirst=True)`
- `ventes.query("MONTANT > 2000 and `DATE FACTURE` >= '2008-09-01' «)`
- `ventes.query(`VENDEUR` == 'DEVEAUX').mean(numeric_only=True)`
- `ventes.groupby(by='VENDEUR').mean()`
- `ventes.groupby(by='VENDEUR').size()`
- `ventes.groupby(by='SECTEUR').sum()`



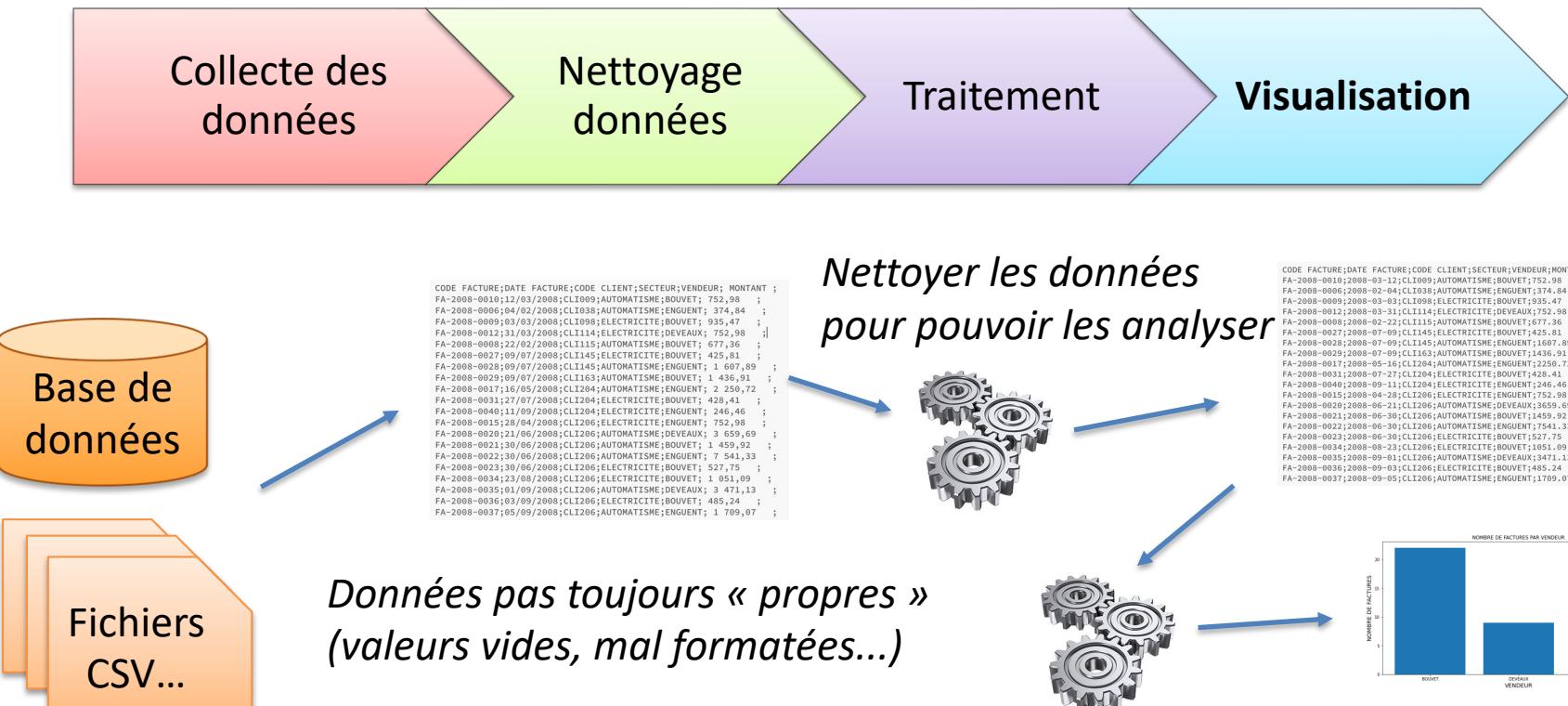


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Étapes de traitement des données

- Les projets d'analyse de données suivent un ensemble d'étapes bien précises





UNIVERSITÉ PARIS 1

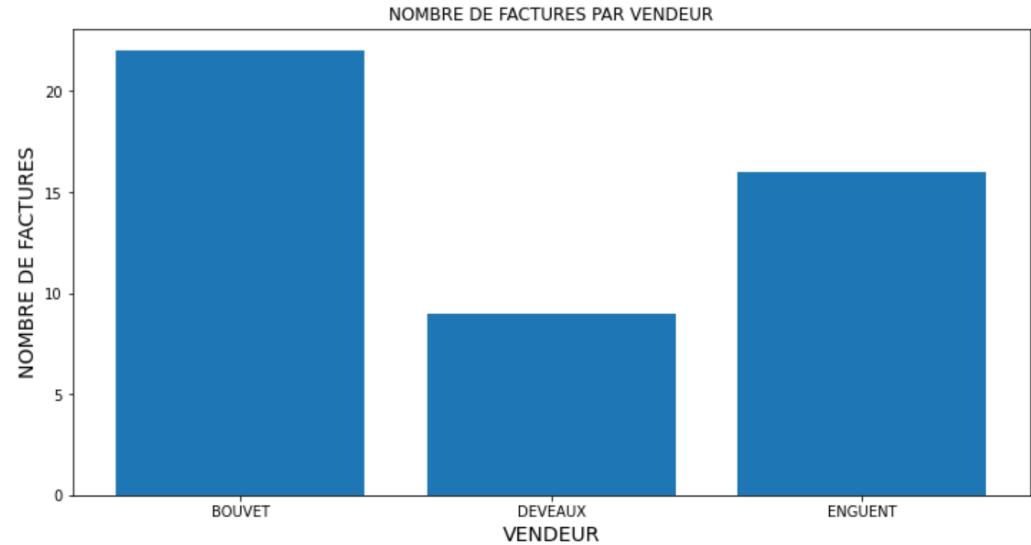
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation

- La visualisation est une partie essentielle de l'analyse de données
- Elle est essentielle pour :
 - **Mieux comprendre**
 - **Mieux communiquer**

```
VENDEUR
BOUVET      22
DEVEAUX      9
ENGUENT     16
dtype: int64
```





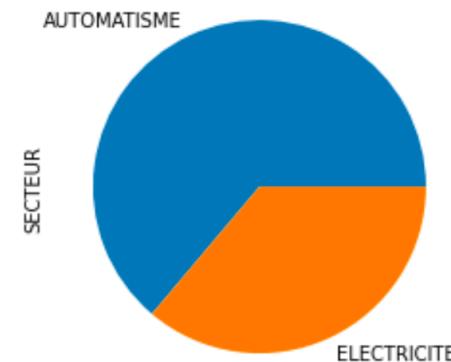
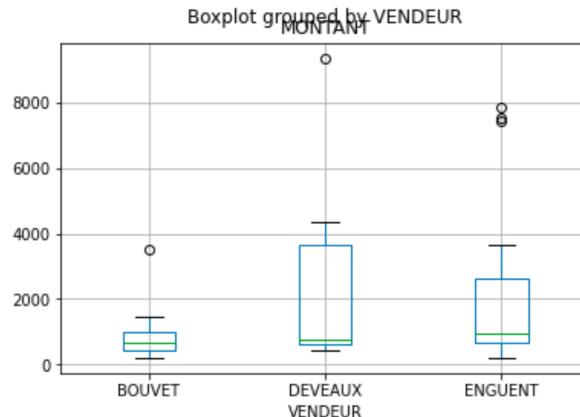
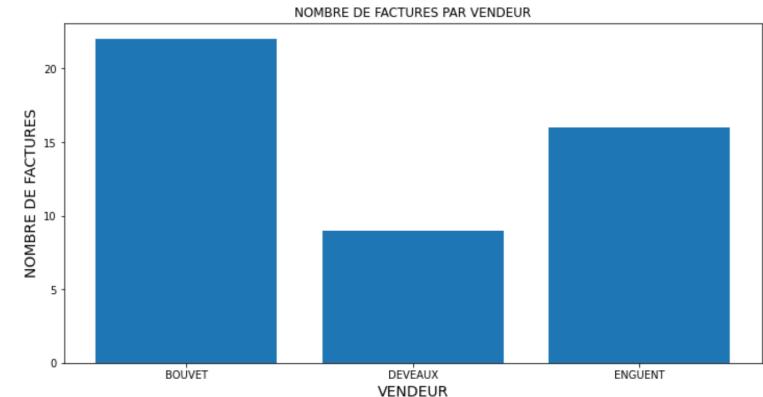
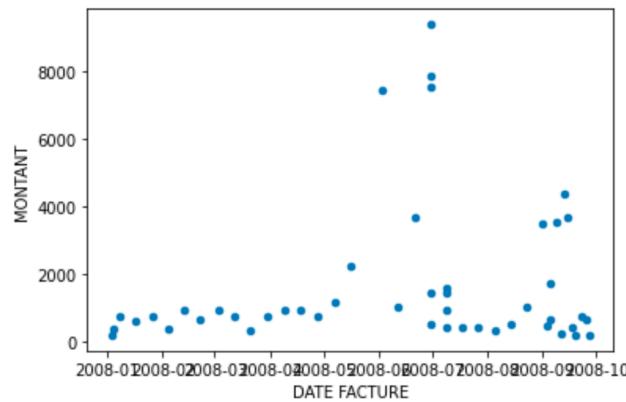
UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

- **Matplotlib** est une bibliothèque de **visualisation des données**
- Possibilité de créer différents types de **graphiques**





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

```
import matplotlib.pyplot as plt  
%matplotlib inline
```

Important : ne pas oublier le **pyplot**

Nécessaire pour afficher les graphiques sur le notebook
plt.show() en mode « batch »

Données : listes, array (Numpy), Series ou **DataFrames**

```
donnees = pd.DataFrame({'note': [2, 5, 12, 18, 10],  
                        'rendus':[1, 3, 5, 7, 9 ],  
                        'nom':['Titi','Toto','Tata','Tarbes','Titus']})
```

On va créer une **figure** qui contiendra le graphique

```
plt.figure(figsize=(10, 4))
```

```
plt.title("Rendus par notes")  
plt.xlabel("Nb de rendus")  
plt.ylabel("Notes")
```

figsize= (largeur , hauteur)

On définit les **paramètres** titre, labels x et y

```
plt.plot(donnees['rendus'], donnees['note'])
```

données axes x et y

On définit le **graphique** proprement dit



UNIVERSITÉ PARIS 1

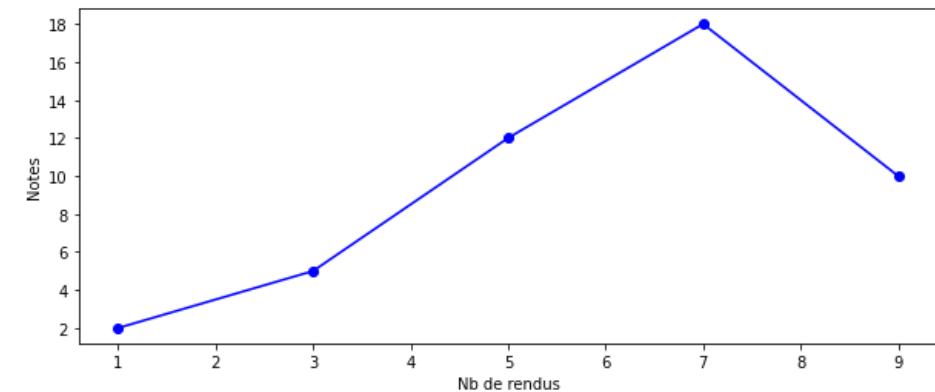
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

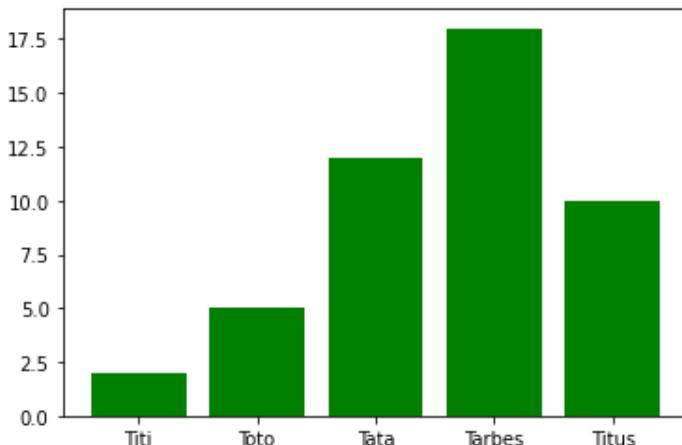
Visualisation avec Matplotlib

```
plt.plot(donnees['rendus'],  
         donnees['note'],  
         color='blue', marker='o' )
```

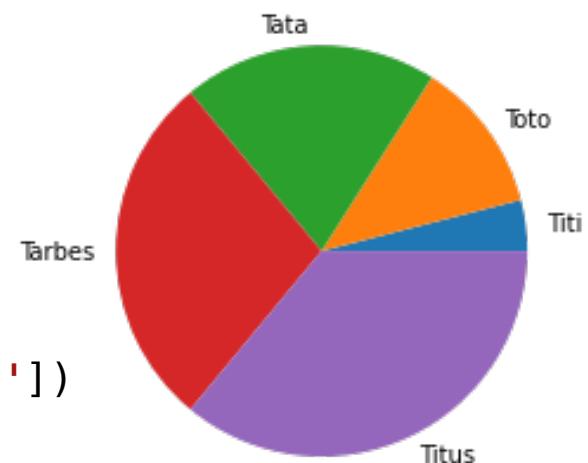
Possibilité de choisir le marqueur



```
plt.bar(donnees['nom'], donnees['note'], color='green')
```

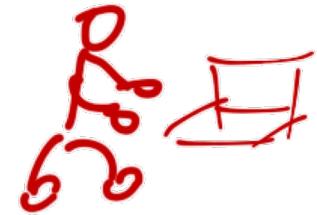


Possibilité de choisir la couleur



```
plt.pie(donnees['rendus'], labels=donnees['nom'])
```

Hands On !



On n'oublie pas :
`import matplotlib.pyplot as plt
%matplotlib inline`

• Exercice : prise en main Matplotlib

- Sur notre **premier** notebook
- Créer un nouveau **DataFrame** avec les données suivantes

```
donnees = pd.DataFrame({'note': [2, 5, 12, 18, 10],  
                        'rendus':[1, 3, 5, 7, 9 ],  
                        'nom':['Titi','Toto','Tata','Tarbes','Titus']})
```

```
plt.figure(figsize=(10, 4))
```

- Créer un **graphique en ligne** pour afficher les **notes** en fonction des **rendus**

```
plt.plot(donnees['rendus'], donnees['note'])
```

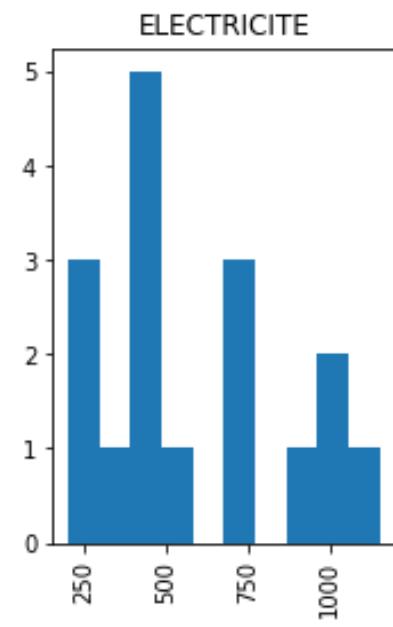
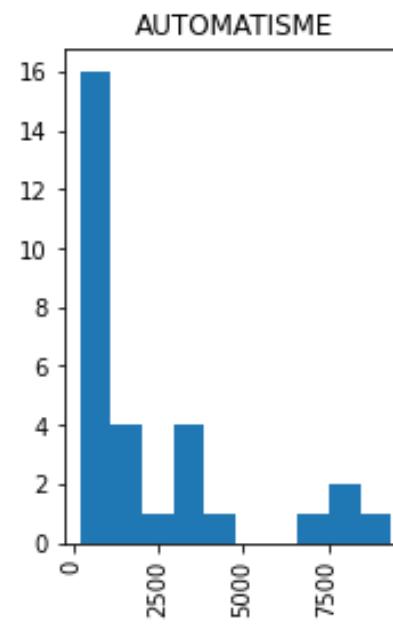
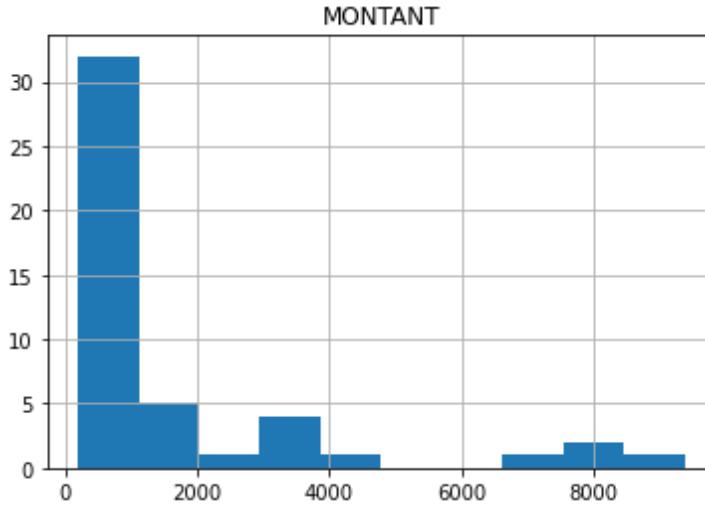
- Créer un **graphique en barres** pour afficher les **notes** associées à chaque **nom**

```
plt.bar(donnees['nom'], donnees['note'])
```

Visualisation avec Matplotlib

- On peut utiliser **Matplotlib** directement à partir d'un **DataFrame**
 - Opérations par type de graphique : **hist**, **boxplot** et **plot** (cas général)

```
ventes.hist(column='MONTANT')
```



```
ventes.hist(column='MONTANT', by='SECTEUR')
```



UNIVERSITÉ PARIS 1

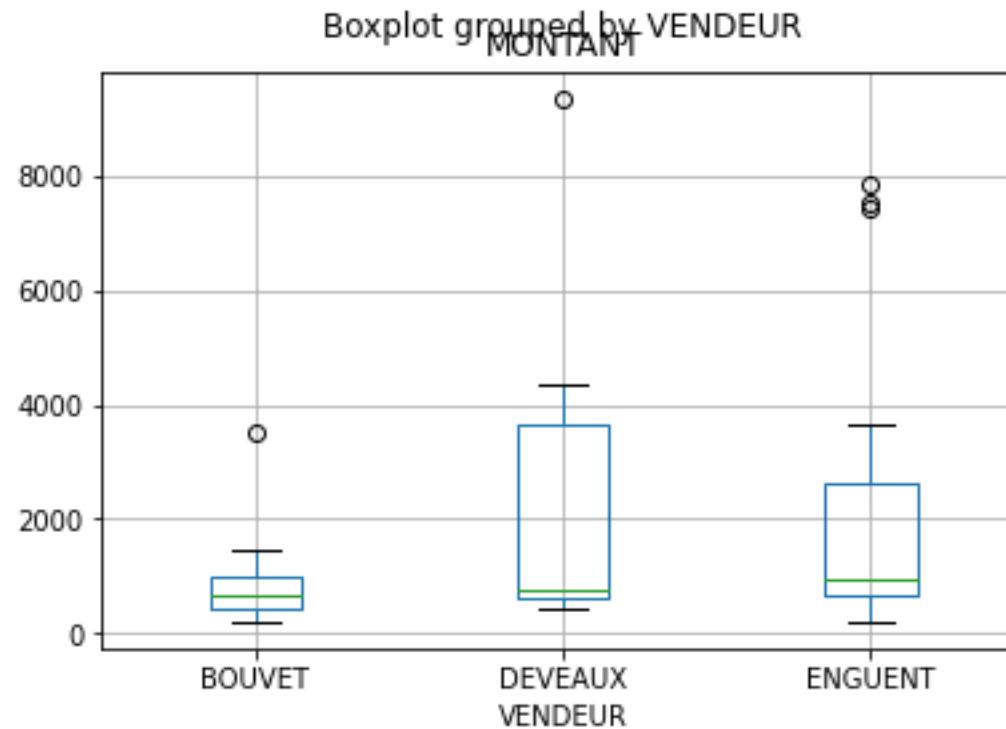
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

- On peut utiliser **Matplotlib** directement à partir d'un **DataFrame**
 - Opérations par type de graphique : **hist**, **boxplot** et **plot** (cas générale)

```
ventes.boxplot(column='MONTANT', by='VENDEUR')
```





UNIVERSITÉ PARIS 1

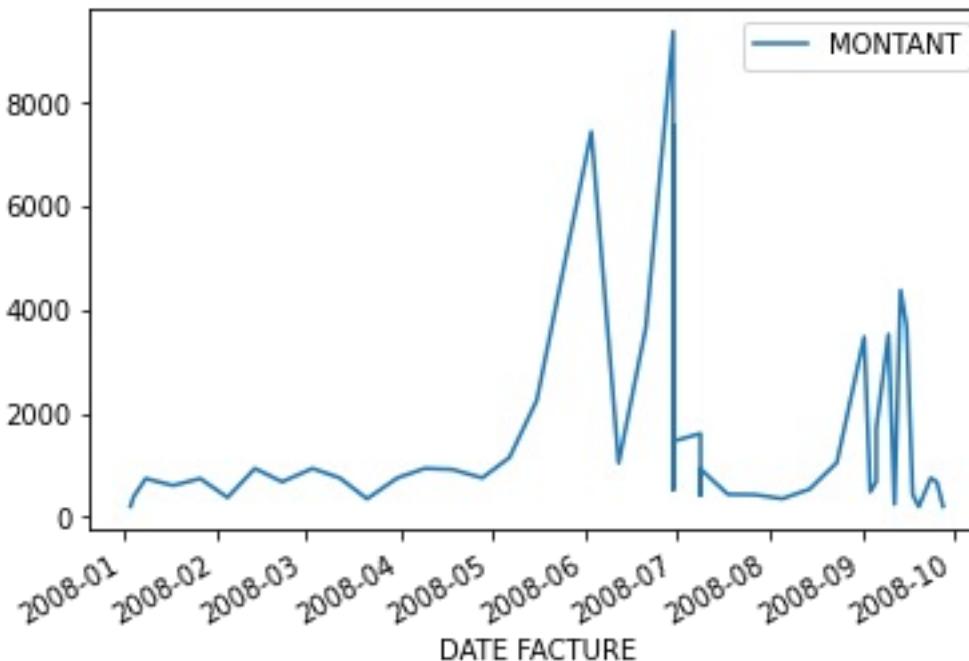
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

- On peut utiliser **Matplotlib** directement à partir d'un **DataFrame**
 - Opérations par type de graphique : **hist**, **boxplot** et **plot** (cas générale)

```
ventes.plot(x='DATE FACTURE', y='MONTANT')
```





UNIVERSITÉ PARIS 1

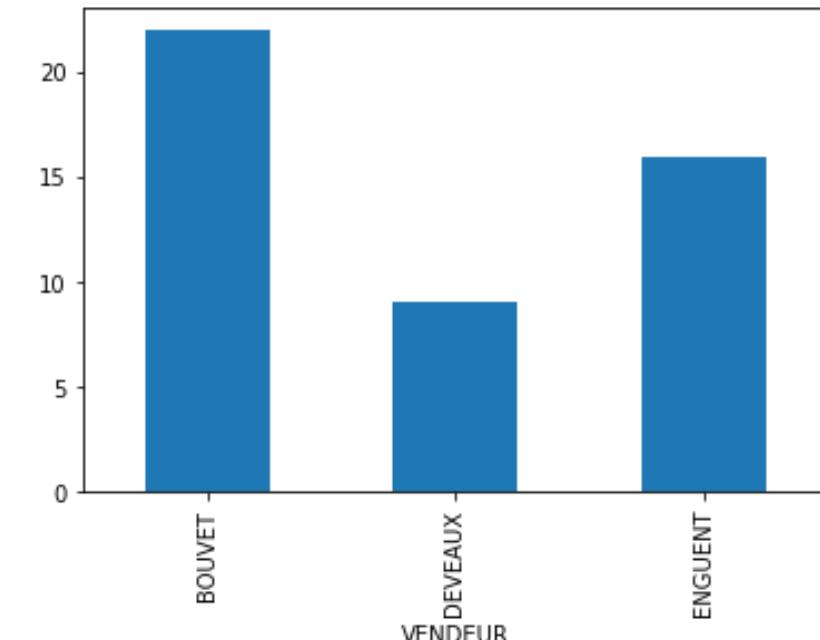
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

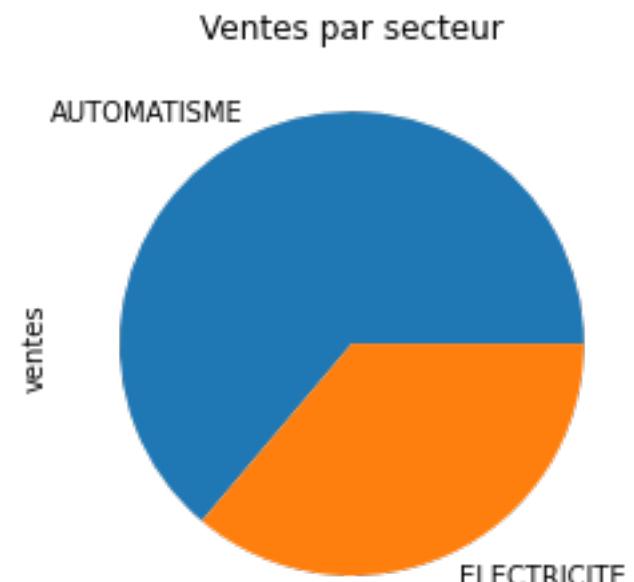
*Choix des données:
ventes **par vendeur***

```
ventes.groupby(by='VENDEUR').size().plot(kind='bar')
```



*On plot dans un graphique
type **barres***

```
ventes.groupby(by='SECTEUR').size().plot(kind='pie',  
title='Ventes par secteur',  
ylabel='ventes')
```



*Choix des paramètres :
titre, label axes x et y*

```
ventes.groupby(by='SECTEUR').size().plot(kind='pie',  
title='Ventes par secteur',  
ylabel='ventes')
```

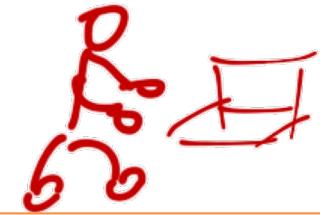


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Hands On !



On n'oublie pas :

```
import matplotlib.pyplot as plt  
%matplotlib inline
```

- **Exercice : visualisation des données**

- Retour sur le DataFrame « *VentesPropre.csv* »
- Créer un **graphique en barres** du **nombre de factures par vendeur**

```
ventes.groupby(by='VENDEUR').size().plot(kind='bar')
```

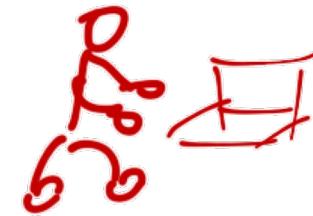
- Créer un **graphique camembert (pie)** du **nombre de factures par secteur**

```
ventes.groupby(by='SECTEUR').size().plot(kind='pie')
```

- Créer un **graphique boîte à moustaches (boxplot)** du **montant des factures par vendeur**

```
ventes.boxplot(column='MONTANT', by='VENDEUR')
```

Hands On !



- **Exercice : visualisation des données**

 - Toujours sur le DataFrame « *VentesPropre.csv* »

 - Observer la sortie des instructions suivantes :

```
moyennes = ventes.groupby(by='VENDEUR').mean()
print(moyennes)
```

 - Réaliser un **graphique en barres** avec ces informations

Suggestions :

Choisir ses paramètres :

```
plt.title('...')  

plt.xlabel('...')  

plt.ylabel('...')
```

```
plt.figure(figsize=(10, 4))  

plt.bar(moyennes.index, moyennes['MONTANT'],  

        color=['skyblue', 'lavender', 'steelblue'],  

        edgecolor='blue')
```

