



Introduction Python pour la Data Science DataFrame

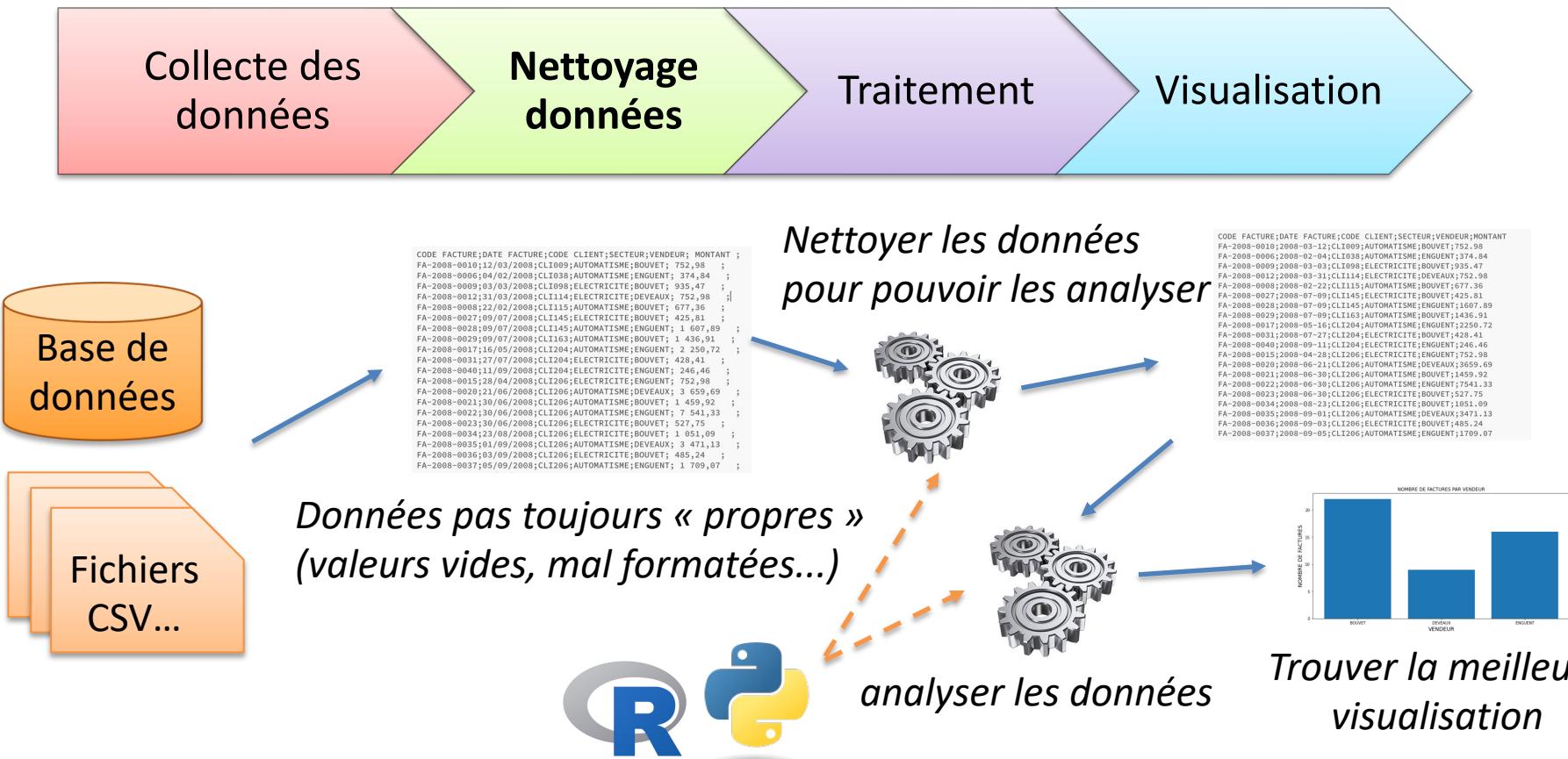


Fichiers sur

<https://github.com/mkirschpin/CoursPython>
<http://kirschpm.fr/cours/PythonDataScience/>

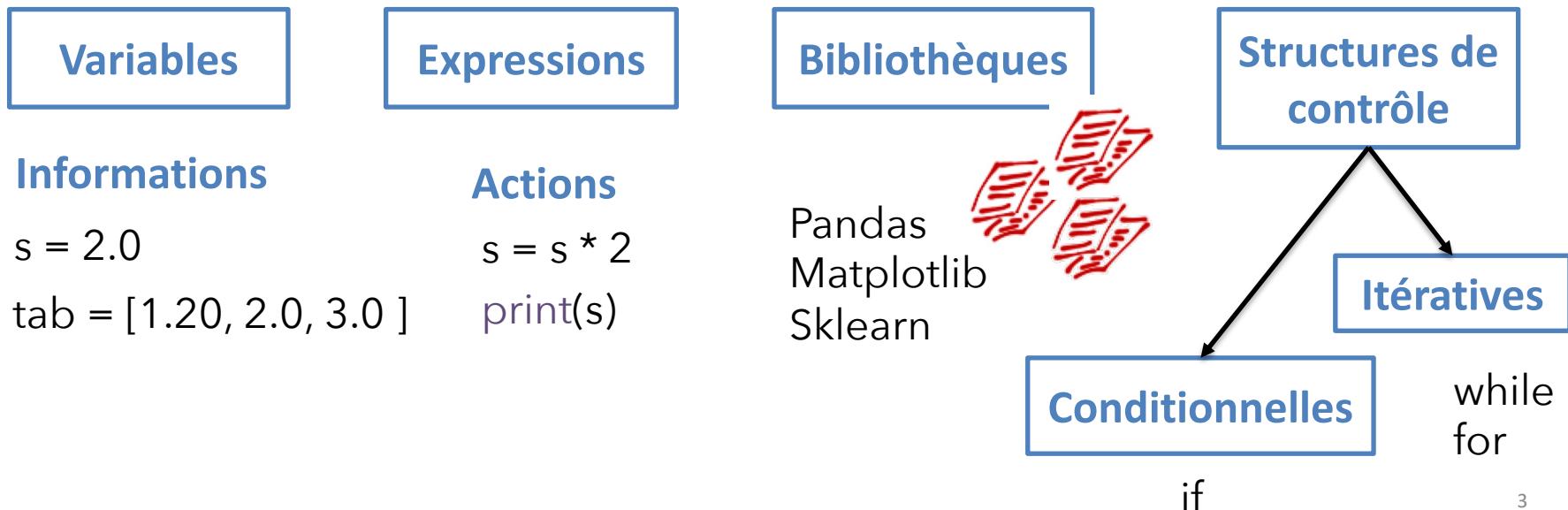
Étapes de traitement des données

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



Python

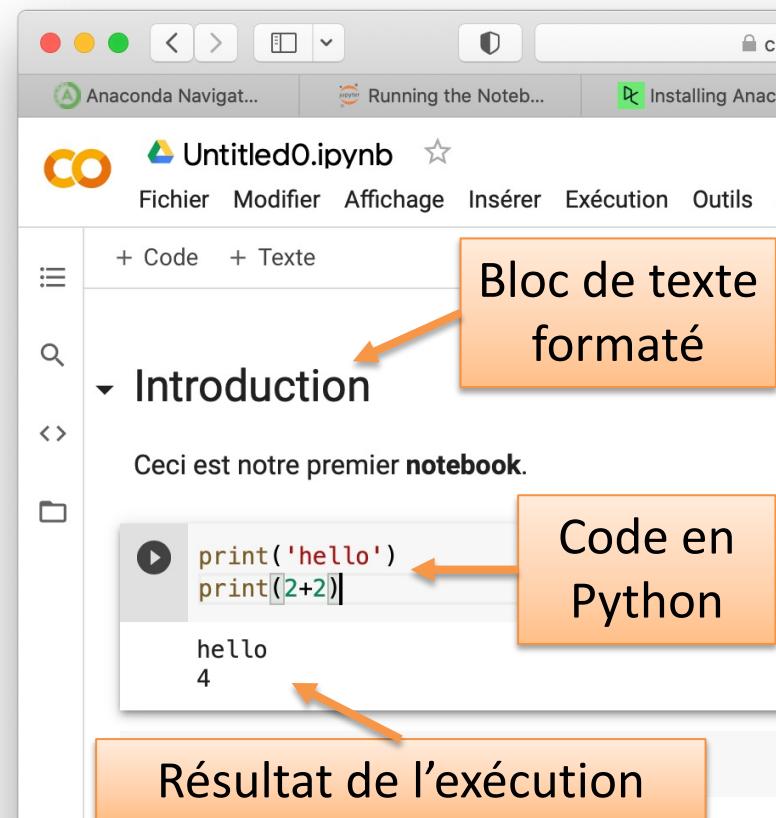
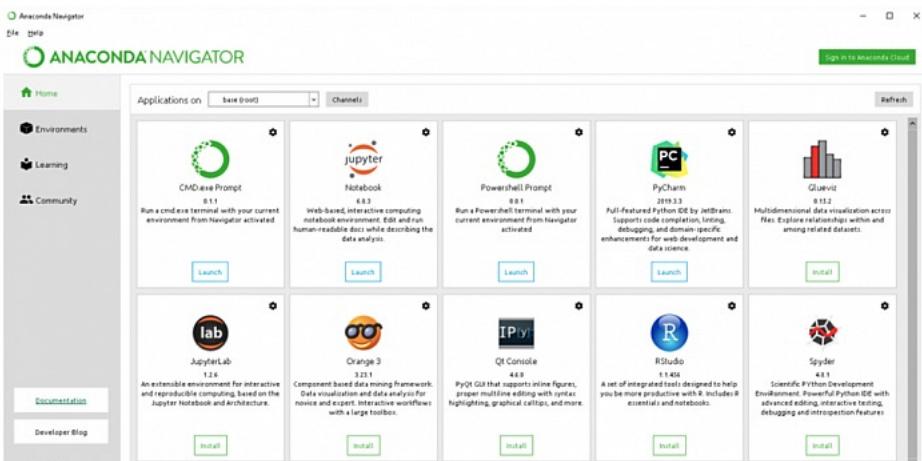
- Pourquoi Python ?
 - Langage de **programmation**
 - Nombreuses ressources pour la **data analyse**
 - Performance
- Éléments de base à maîtriser



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 Lab** (ou Jupyter Notebook)



Python : Variables

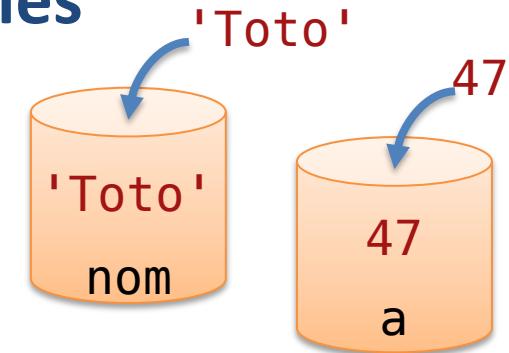
- Variables

- Permettent de stocker de l'information

Affectation :

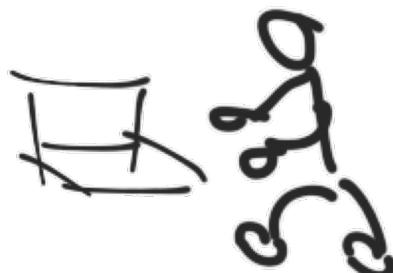
On attribue une valeur à la variable

```
a = 47
nom = 'Toto'
print('Hello ! Je suis', nom,
      "et j'ai", a , 'ans.')
```

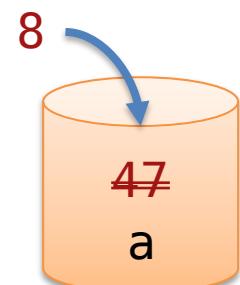


Hello ! Je suis Toto
et j'ai 47 ans.

double 8



```
a = len(nom) * 2
```



```
print(double', a)
```

`print (val , val , val ...)`

On affiche une valeur (une variable ou un message)

`len (collection)`

permet de connaitre la taille d'une collection

Opérations
(`print`, `len`, * ...)

Permettent la réalisation de certaines actions



UNIVERSITÉ PARIS 1

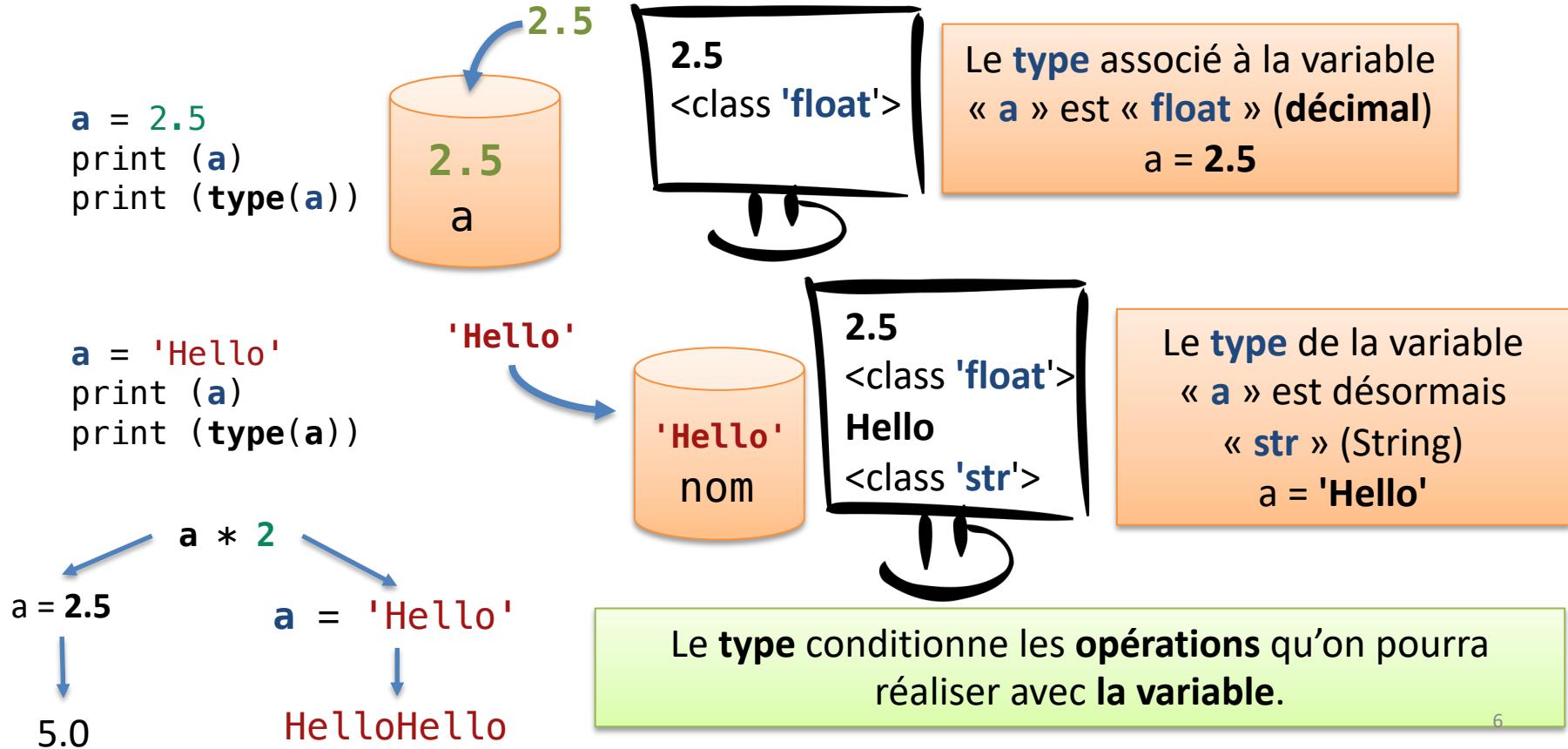
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

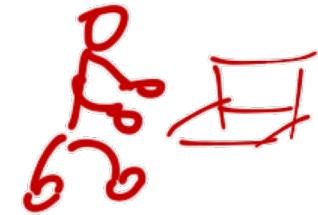
Python : Variables

• 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 !

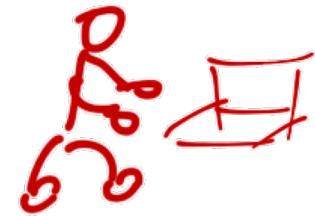


- Aller sur <https://colab.research.google.com/>
- (ou ouvrir Anaconda Navigator et lancer JupyterLab)
- Démarrer un nouveau Notebook Python

The collage consists of four screenshots:

- Google Colab:** A screenshot of a web browser window showing the Google Colab interface. The URL is colab.research.google.com. The page displays a notebook titled "Untitled5.ipynb - Colab". The interface includes a sidebar with "Sommaire" and "Exemples", a main workspace with a "Code" tab, and a message: "Commencez à coder ou à générer avec l'IA".
- JupyterLab:** A screenshot of a desktop application window titled "JupyterLab". It shows a file browser with a list of Python files in a directory named "Python / cours /". Below the file browser are two tabs: "Notebook" and "Console".
- Anaconda Navigator:** A screenshot of a desktop application window titled "ANACONDA NAVIGATOR". It lists various environments and tools, including "Jupyter Notebook", "PyCharm Community", "Qt Console", "Spyder", and "VS Code". Each entry has a "Launch" button.
- Colab in Navigator:** A screenshot of a desktop application window titled "colab.research.google.com". It shows the "Ouvrir le notebook" (Open notebook) dialog box from Google Colab, which lists examples like "Overview of Colab Features", "Markdown Guide", and "Charts in Colab".

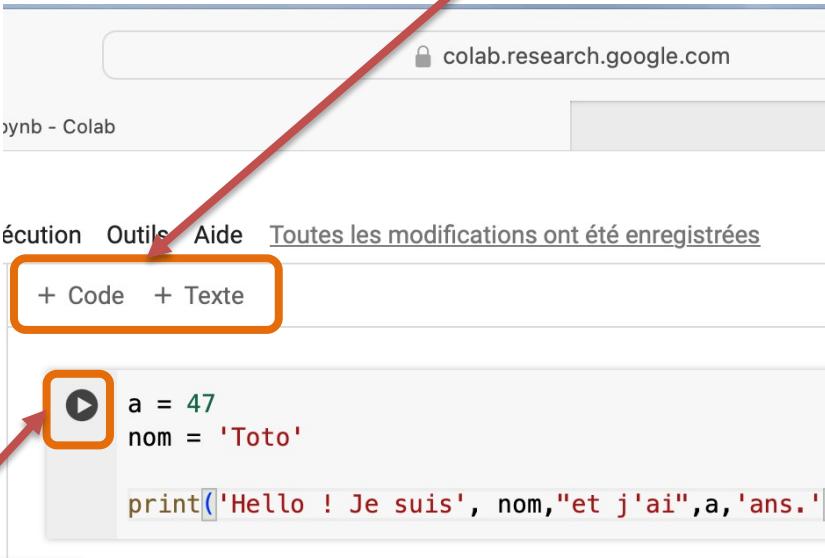
Hands On !



• Exercice : Prise en main Notebook

- Ajouter un bloc de **Markdown** et y ajouter un texte
- Ajouter un **bloc de code Python** :
 - Créer une variable nommée « **a** »
 - Lui affecter la valeur **2.5**
 - Faire **a * 5**
 - **Exécuter le bloc**
- Sur un nouveau bloc de code :
 - Affecter la valeur '**toto**' à la variable « **a** »
 - Faire **a * 5**
 - **Exécuter le bloc**

Choisir le type de bloc
(markdown ou code)



The screenshot shows a Google Colab notebook interface. At the top, there's a header bar with the URL 'colab.research.google.com'. Below it, a toolbar has buttons for 'Execution' (highlighted with an orange box), 'Outils', 'Aide', and a status message 'Toutes les modifications ont été enregistrées'. Underneath the toolbar, there are two buttons: '+ Code' (highlighted with an orange box) and '+ Texte'. The main workspace contains the following Python code:

```
a = 47
nom = 'Toto'

print('Hello ! Je suis', nom, "et j'ai", a, 'ans.')
```

A red arrow points from the text 'Choisir le type de bloc (markdown ou code)' to the '+ Code' button. Another red arrow points from the text 'Exécuter le bloc' to the play button icon in the code cell.



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**)

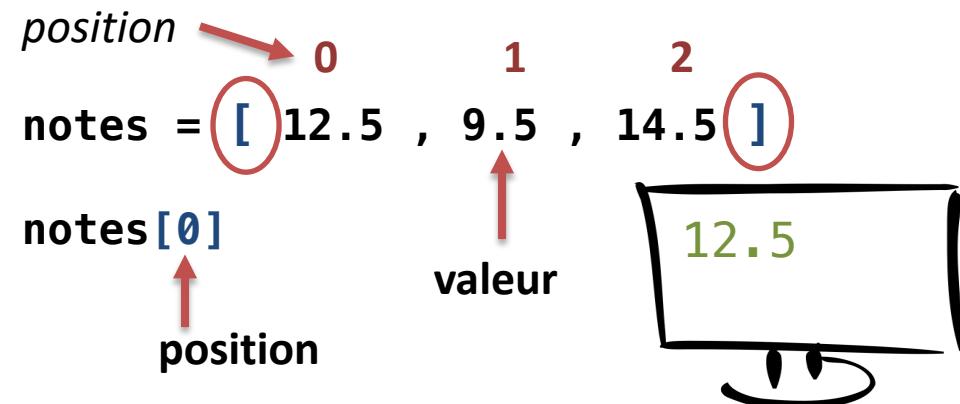
variable clé : valeur

```
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



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

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)
```

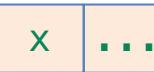
```
{'nom': 'Toto', 'matr': 100,  
'formation': 'marketing',  
'notes': [12.5, 9.5, 14.5]}
```

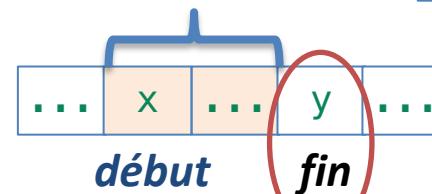
notes[0:2]

On récupère les deux premières valeurs

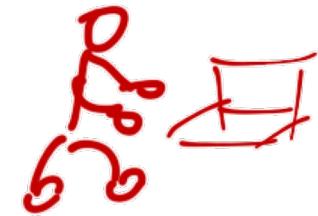
12.5	9.5	14.5
0	1	2

[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 la liste. Les listes sont utiles pour stocker plusieurs éléments qui doivent être manipulés ensemble.

[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]}
```



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Python : bibliothèques

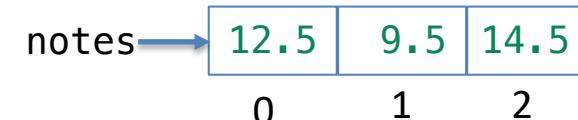
- Une bibliothèque est un **catalogue de fonctions**
 - Ensemble des **codes prêts** et disponibles pour **usage**
 - Disponibles sur des sujets très variés :
 - Data analyse (**pandas**), Machine Learning (**sklearn**)...
- Pour utiliser une bibliothèque, on doit d'abord l'importer

On importe toute la bibliothèque

```
import math as mt
```

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

*as → alias « mt »
(pour faire court)*



3.082207001484488

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



On ne réinvente pas la roue !



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Python : Pandas

- **Pandas**

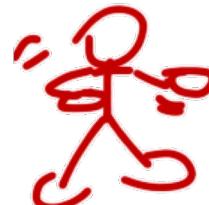
- Importante bibliothèque dédiée à l'analyse de données
- Deux structures de données majeures : **Series** et **DataFrame**

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']

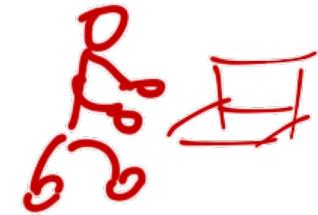
	index	colonnes
'Suisse'	8	'Geneve'
'France'	70	'Paris'
'USA'	320	'Washington'
'Chine'	1200	'Pequin'

```
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()
```

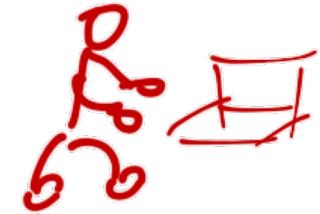
```
<class 'pandas.core.frame.DataFrame'>  
Index: 4 entries, Suisse to Chine  
Data columns (total 2 columns):  
 #   Column      Non-Null Count  Dtype    
 ---    
 0   Habitants    4 non-null      int64
```

```
[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>

<http://kirschpm.fr/cours/PythonDataScience/>

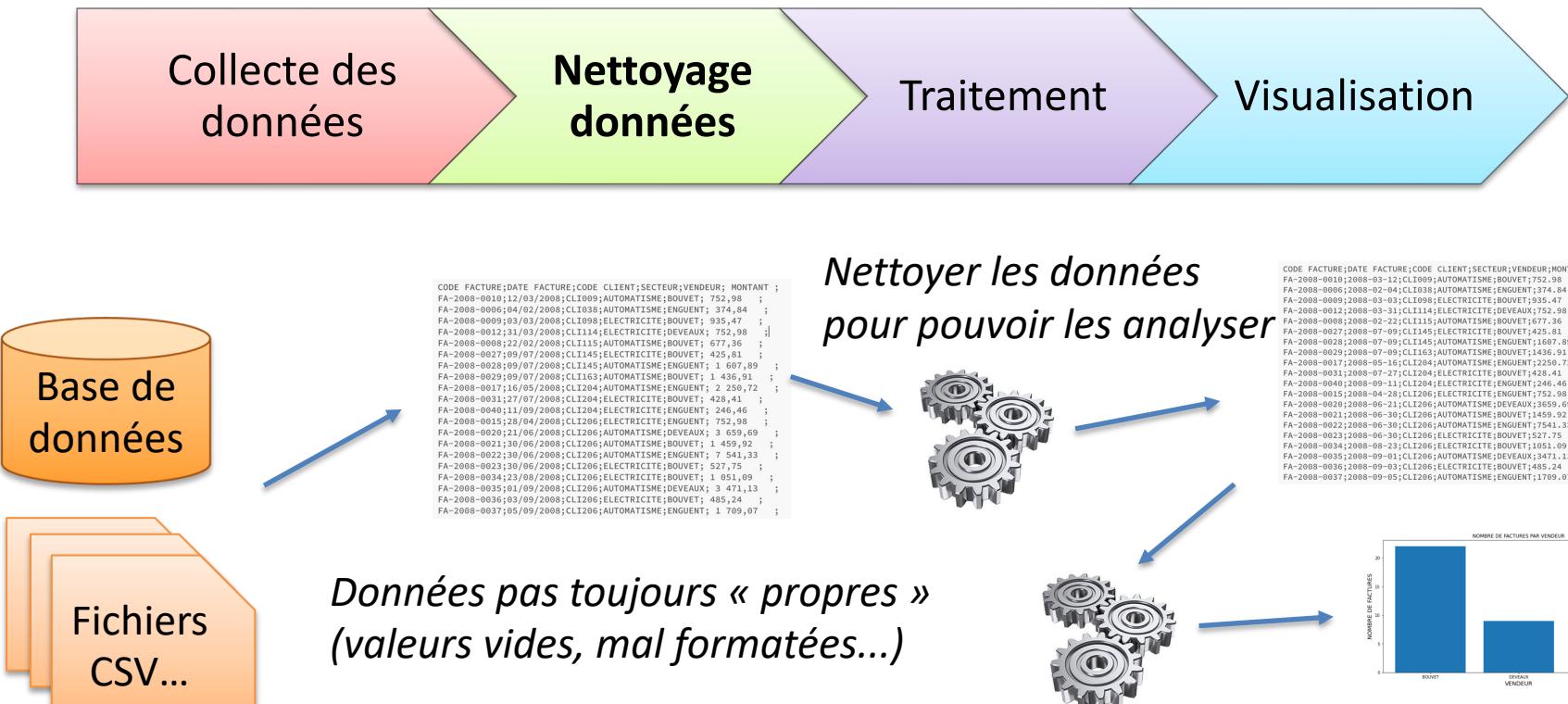


UNIVERSITÉ PARIS 1

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

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**)

En-tête (nom des colonnes)

On peut indiquer l'URL où se trouve le fichier :
`'http://www.kirschpm.fr/cours/PythonDataScience/files/VentesAgenceU.csv'`

```
import pandas as pnd  
ventes = pnd.read_csv('files/VentesAgenceU.csv',
```

Autres options possibles

Par ex. si index est une date :

`parse_dates=True`
`dayfirst=True`

Séparateur → `delimiter=';'`,
Ligne(s) d'en-tête → `header=[0]`,
`index_col=[0]` → 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]:

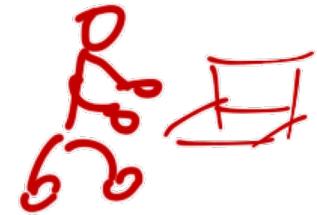
ventes.head()

CODE FACTURE	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT	Unnamed: 6
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
 - Lire le fichier « **VentesAgenceU.csv** » qui se trouve sur
<http://www.kirschpm.fr/cours/PythonDataScience/files/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



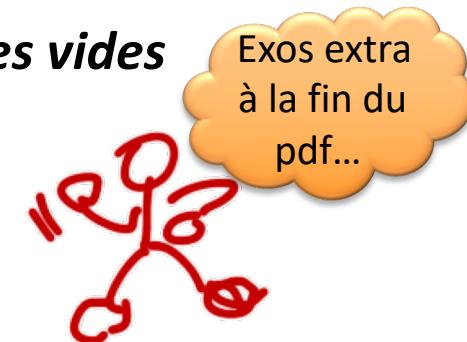
UNIVERSITÉ PARIS 1

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

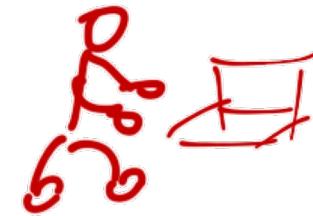
PANTHÉON SORBONNE

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	

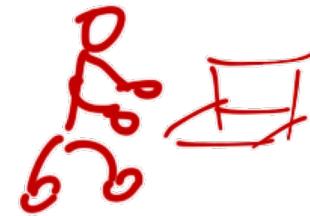
```
[5]: df2.dropna(axis='index')
```

```
[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*

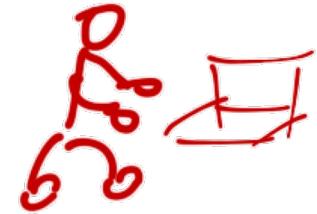
```
[9]: df2.dropna(inplace=True)
```

```
[10]: df2
```

```
[10]:
```

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

Hands On !



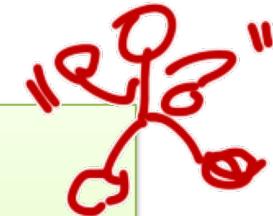
- **Exercice : nettoyer les valeurs vides**

- 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

Tester d'abord sans le
inplace=True

Différentes possibilités :

- **dropna(axis='index', how='all', inplace=True)**
- **dropna(axis='columns', how='all', inplace=True)**
ou
- **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)

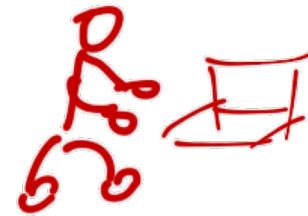
	DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT
CODE FACTURE					
FA-2008-0019	12/06/2008	CLI300	ELECTRICITE	BOUVET	1_039,05
FA-2008-0033	14/08/2008	CLI300	AUTOMATISME	DEVEAUX	535,90
...
FA-2008-0030	18/07/2008	CLI403	ELECTRICITE	DEVEAUX	436,86
FA-2008-0039	09/09/2008	CLI403	AUTOMATISME	BOUVET	3_521,80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

[12]: 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

[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.

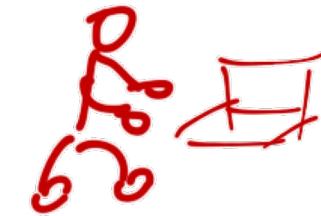
Colonne contenant les valeurs à convertir

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'] } )
```

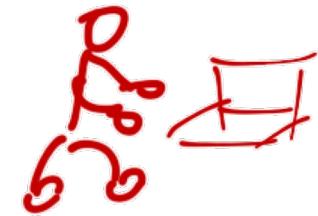
- Afficher les données
- Vérifier le DataFrame avec **info**

	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

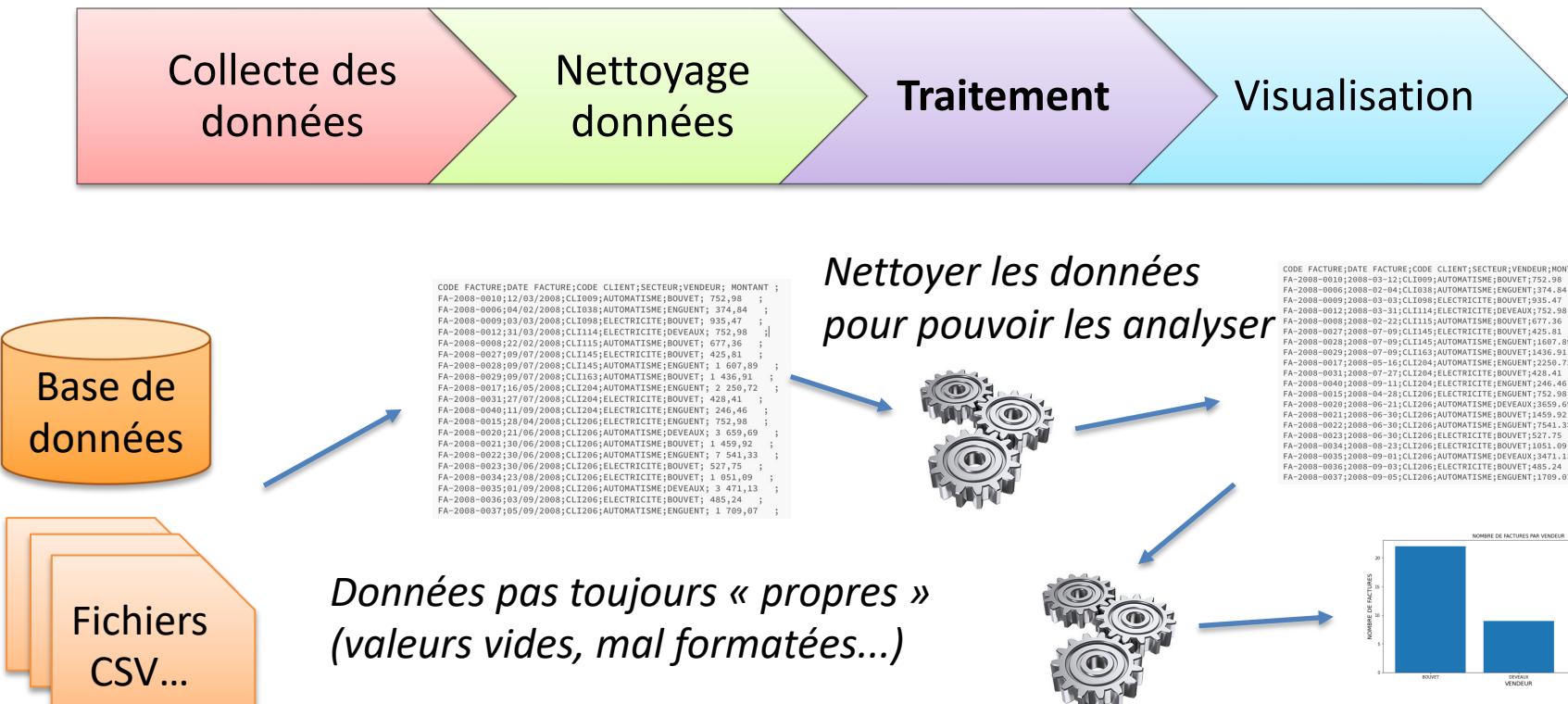


UNIVERSITÉ PARIS 1

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' ")
```

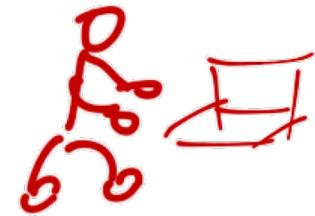
		DATE FACTURE	CODE CLIENT	SECTEUR	VENDEUR	MONTANT
	CODE FACTURE					
1	FA-2008-0012	2008-03-31	CLI114	ELECTRICITE	DEVEAUX	752.98
2	FA-2008-0020	2008-06-21	CLI206	AUTOMATISME	DEVEAUX	3659.69
3	FA-2008-0035	2008-09-01	CLI206	AUTOMATISME	DEVEAUX	3471.13
4	FA-2008-0045	2008-09-23	CLI206	AUTOMATISME	DEVEAUX	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 !



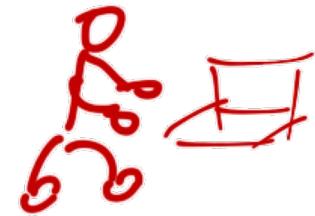
- **Exercice : Traitement des données**
 - Créer un **nouveau Notebook** pour cette activité
 - Lire le fichier « **VentesPropre.csv** » qui se trouve sur
<http://www.kirschpm.fr/cours/PythonDataScience/files/VentesPropre.csv>
sur un **DataFrame**
 - Convertir la colonne « **DATE FACTURE** » en date
 - 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 ?**

Ne pas oublier le
import pandas

*attention au format
" " de la date*

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

Hands On !

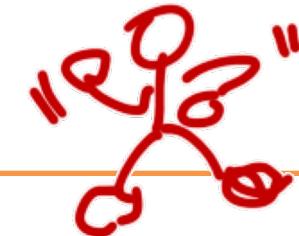


- **Exercice : Traitement des données**

- 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 **pnd** 
- **Nouveau notebook,**
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()**



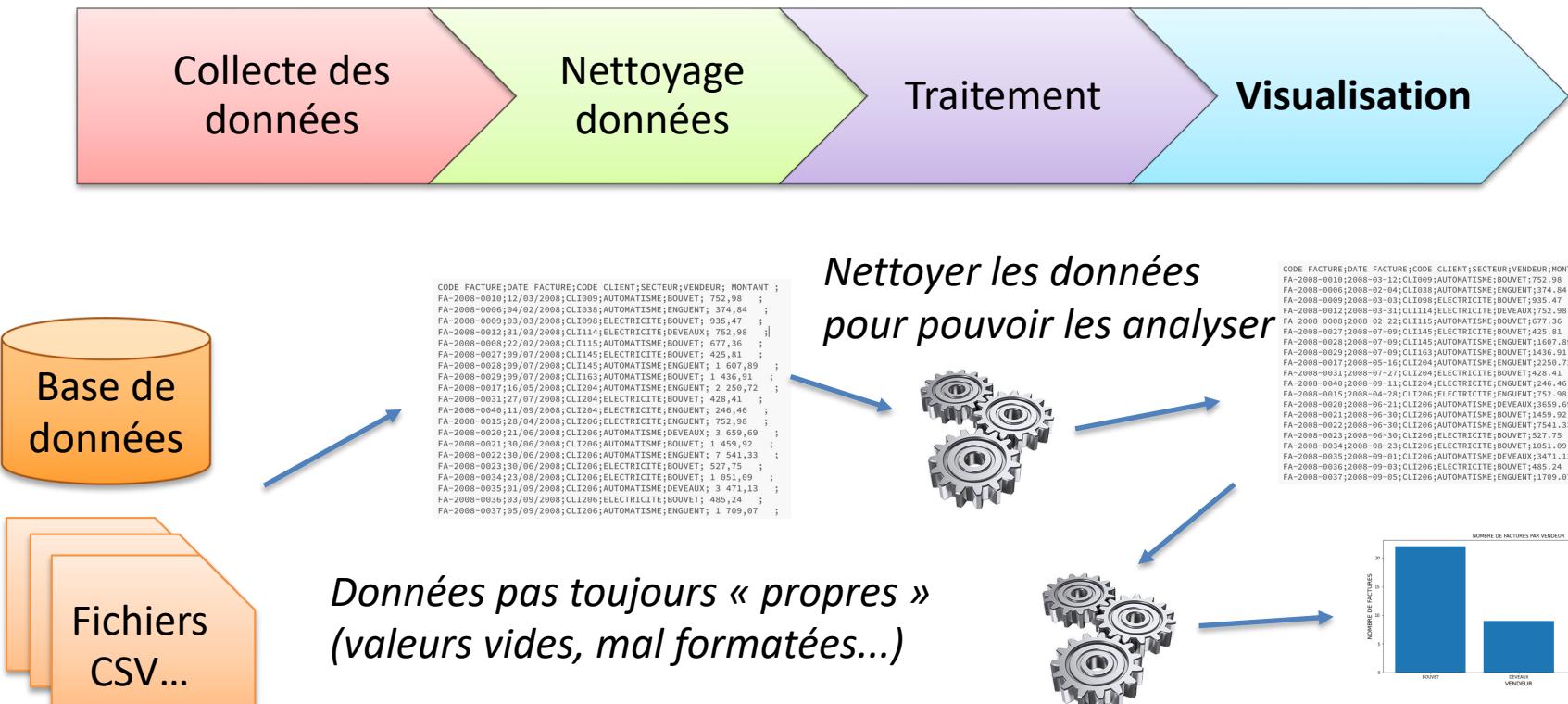


UNIVERSITÉ PARIS 1

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

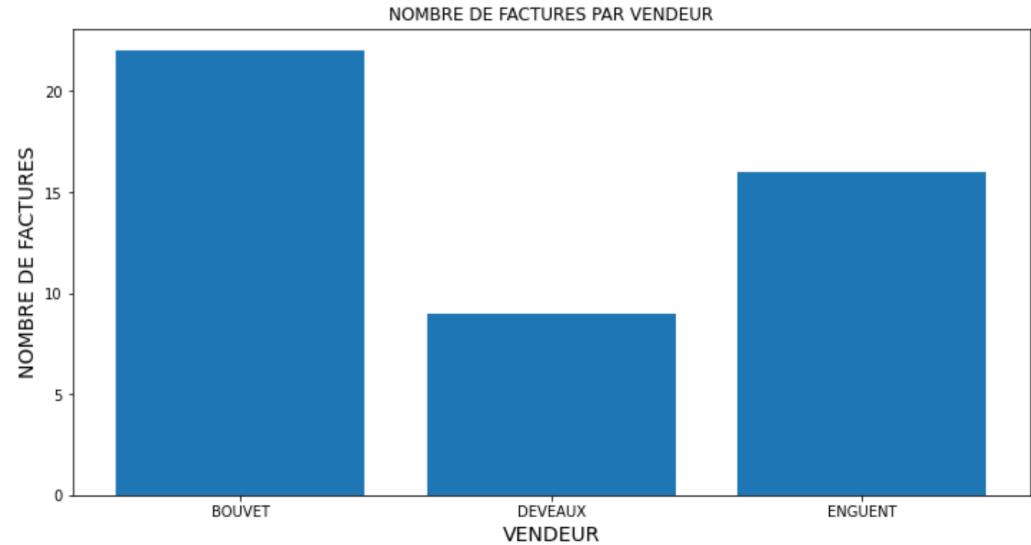
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

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
```





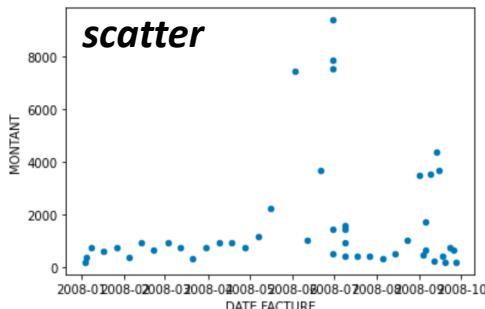
UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

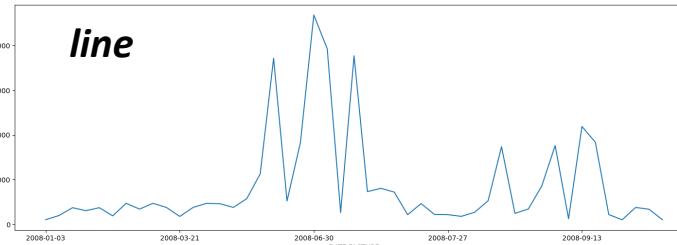
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

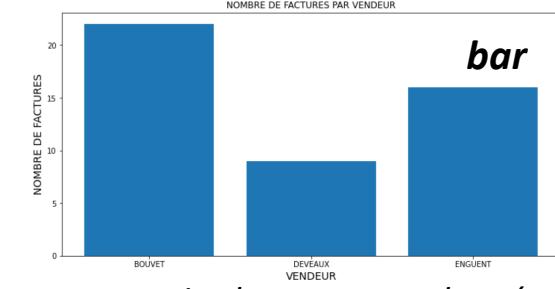
- Matplotlib est une bibliothèque de visualisation des données
- Possibilité de créer différents types de graphiques



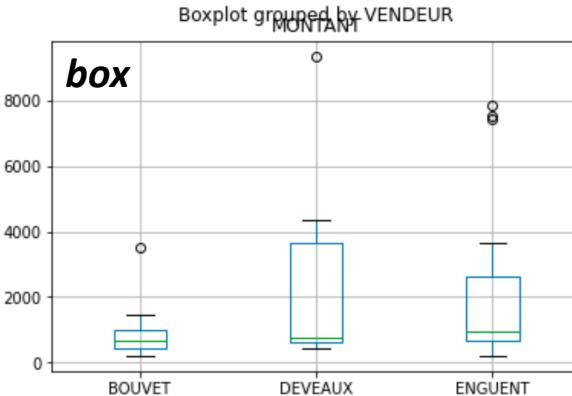
Relation entre 2 variables,
identifier clusters, outliers



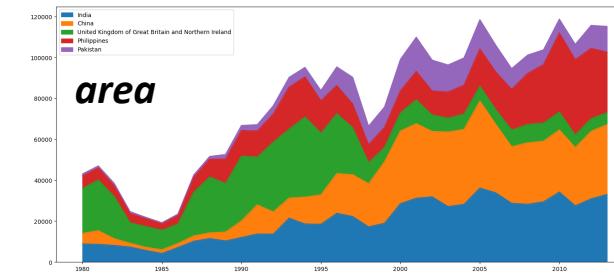
tendance, évolution données continues



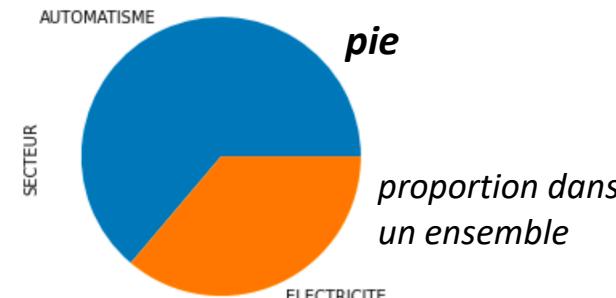
magnitude, comparer données discrètes



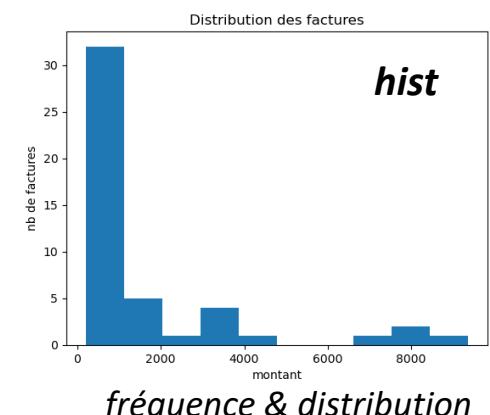
comparer &
visualiser distribution statistique



magnitude,
données cumulatives



proportion dans
un ensemble



fréquence & distribution



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
(pas sur une nouvelle fenêtre)

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

figsize= (largeur , hauteur)

Mode d'emploi :
1) Créer
2) Paramétriser
3) Afficher

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

On **affiche** le graphique

données axes x et y

On définit le **graphique** proprement dit



UNIVERSITÉ PARIS 1

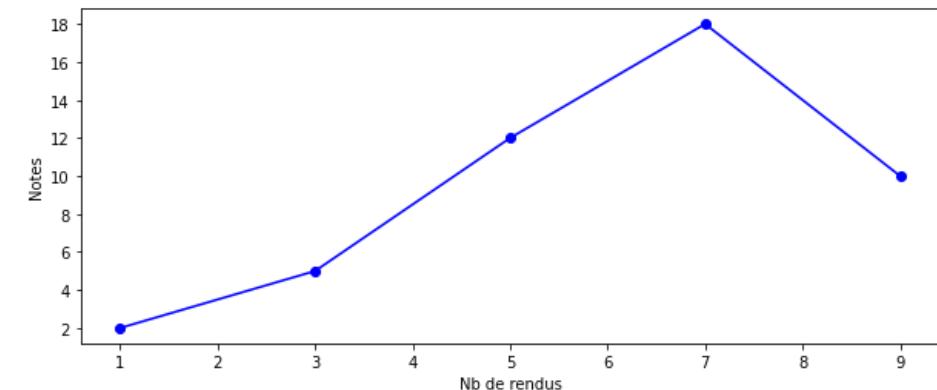
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

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

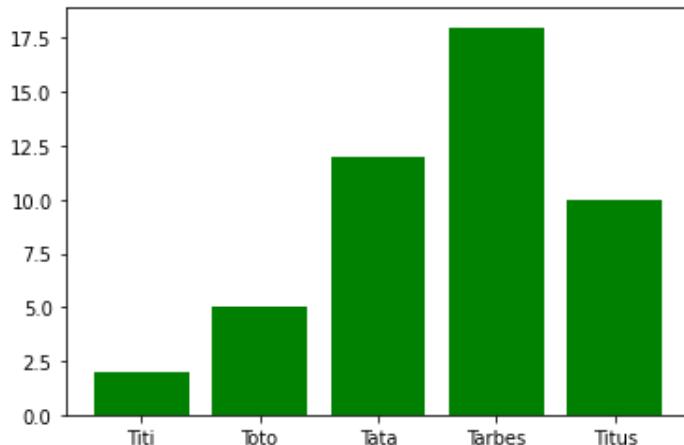
Possibilité de choisir le marqueur



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

x

y

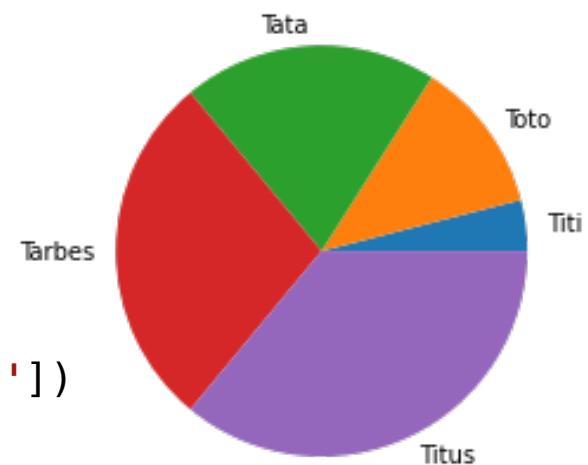


Possibilité de choisir la couleur

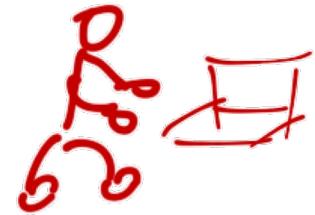
```
plt.pie(donnees['rendus'], labels=donnees['nom'])
```

valeurs

étiquettes



Hands On !



• 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']})
```

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

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

```
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'])
```

On n'oublie pas :

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



UNIVERSITÉ PARIS 1

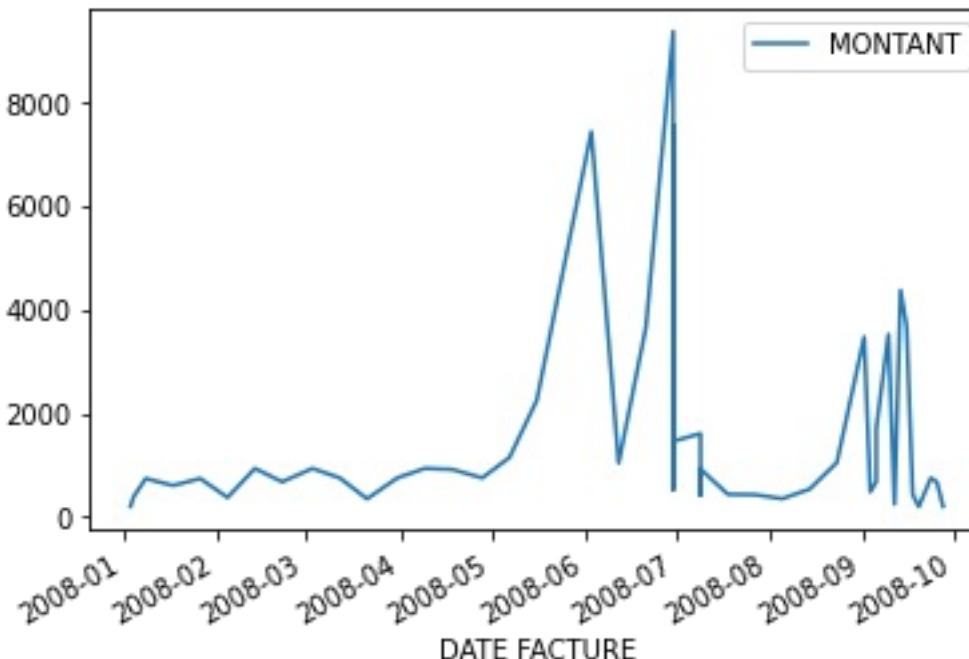
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

- On peut utiliser **Matplotlib** directement à partir d'un **DataFrame**
 - Opération **plot** ou par type de graphique (**hist**, **boxplot**...)

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





UNIVERSITÉ PARIS 1

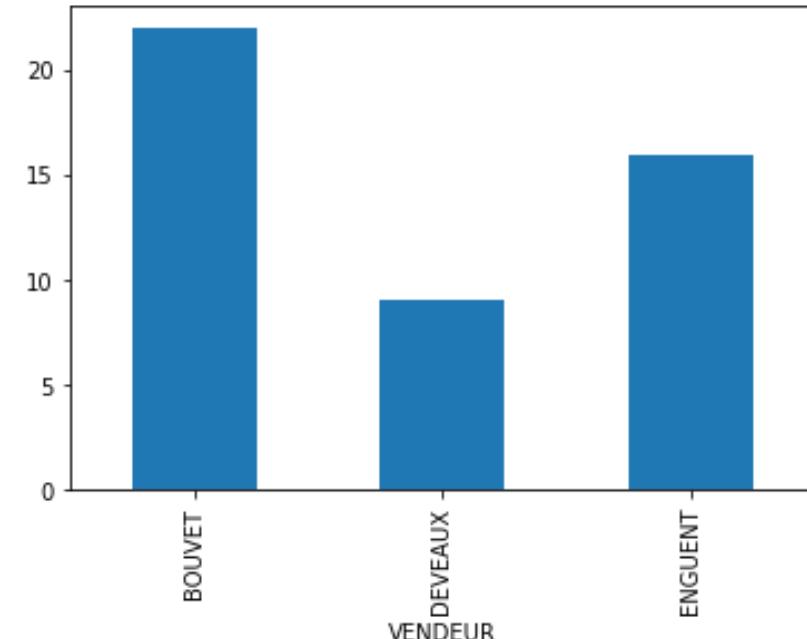
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

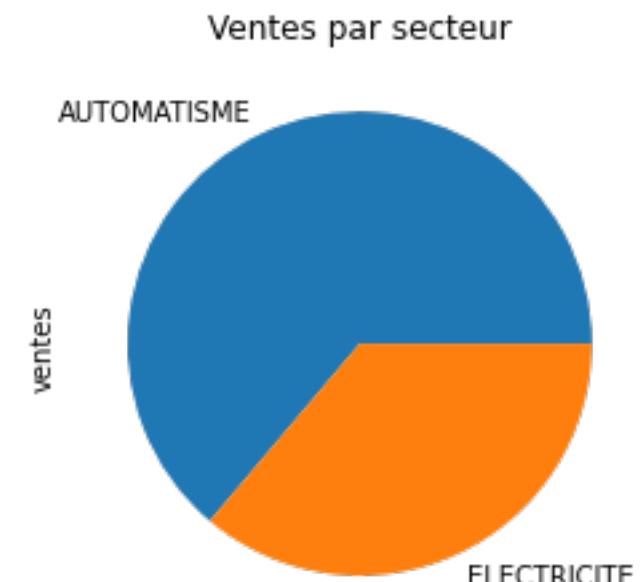
*Choix des données:
ventes **par vendeur***

`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*



UNIVERSITÉ PARIS 1

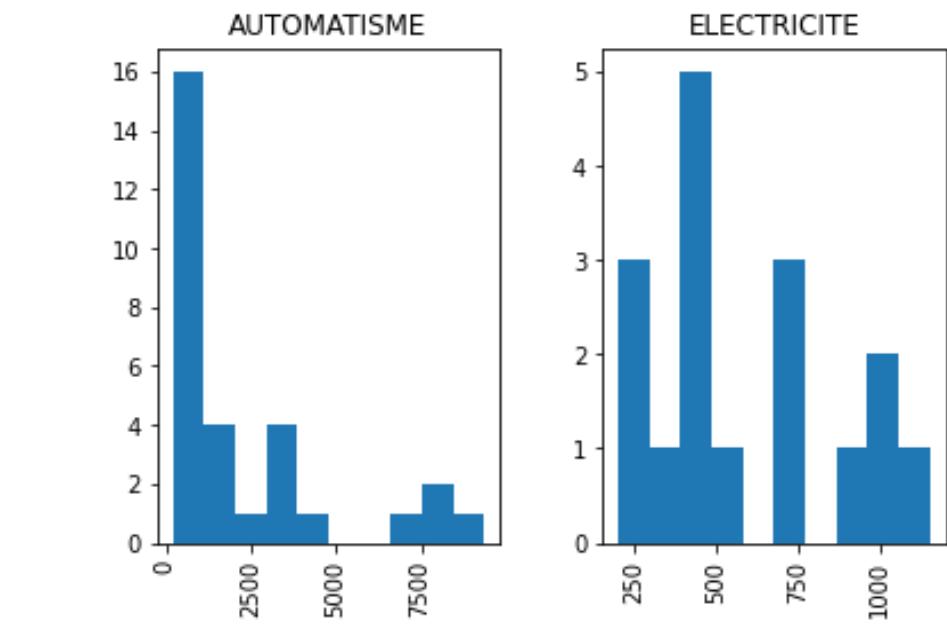
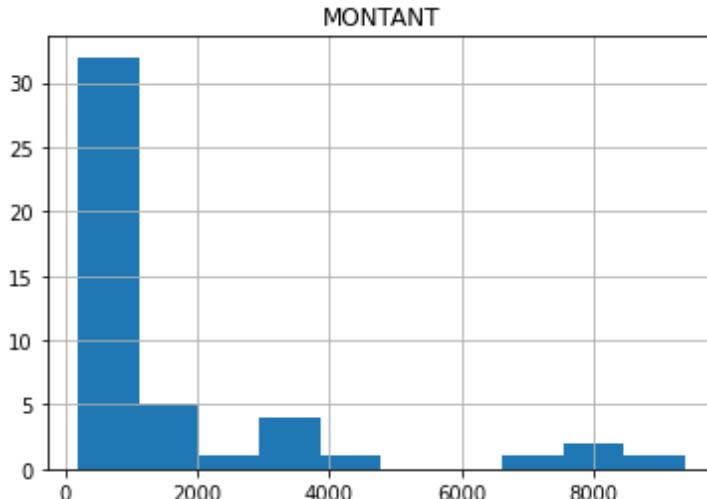
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Visualisation avec Matplotlib

- On peut utiliser **Matplotlib** directement à partir d'un **DataFrame**
 - Opération **plot** ou par type de graphique (**hist, boxplot...**)

ventes.hist(column='MONTANT')



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



UNIVERSITÉ PARIS 1

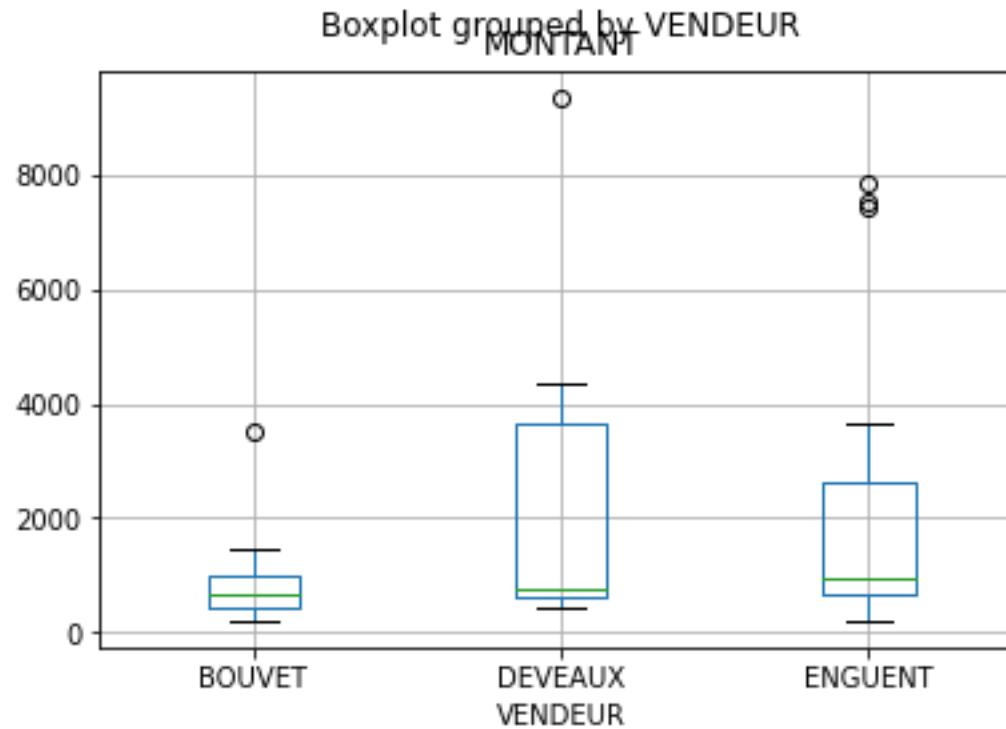
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

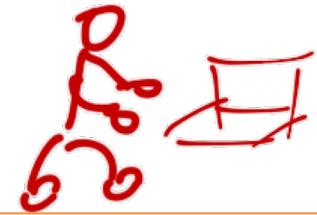
Visualisation avec Matplotlib

- On peut utiliser **Matplotlib** directement à partir d'un **DataFrame**
 - Opération **plot** ou par type de graphique (**hist**, **boxplot...**)

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



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')
```

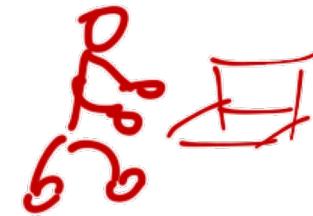
- Créer un **graphique camembert (pie)** du **nombre de factures par secteur**

```
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('...')  

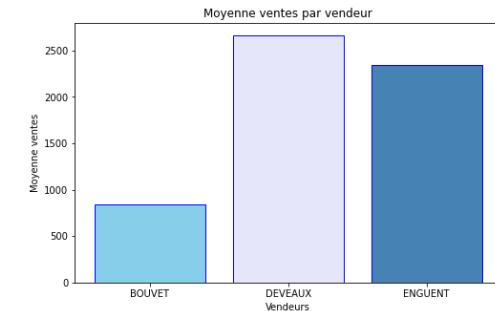
plt.ylabel('...')
```

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

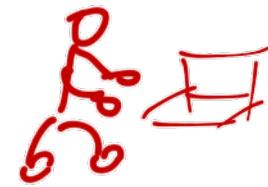
plt.bar(moyennes.index, moyennes['MONTANT'],  

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

        edgecolor='blue')
```



Hands On !



- **Exercice : visualisation des données**

- Trouver les valeurs **moyennes et médianes** des ventes **par vendeur**

```
df_moy_med = ventes.groupby(by='VENDEUR')[['MONTANT']].agg(['mean', 'median'])
df_moy_med
```

*agg permet de faire plusieurs
calculs sur le **groupby***

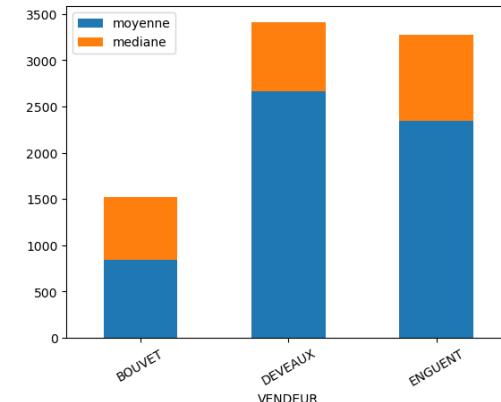
- Réaliser un **graphique en barres** avec ces informations **empilées**
- Paramétrier le graphique :
 - Axe X avec une légère **rotation** `plt.xticks(rotation=30)`
 - Légende **personnalisée** `plt.legend(["moyenne", "mediane"])`

```
df_moy_med.plot(kind='bar', stacked=True)
```

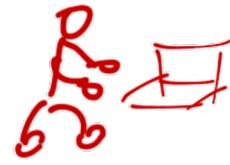
```
plt.xticks(rotation=30, horizontalalignment="center")
```

```
plt.legend(["moyenne", "mediane"])
```

	mean	median
VENDEUR		
BOUVET	840.522273	679.57
DEVEAUX	2661.862222	752.98
ENGUENT	2344.361875	929.72



Hands On !



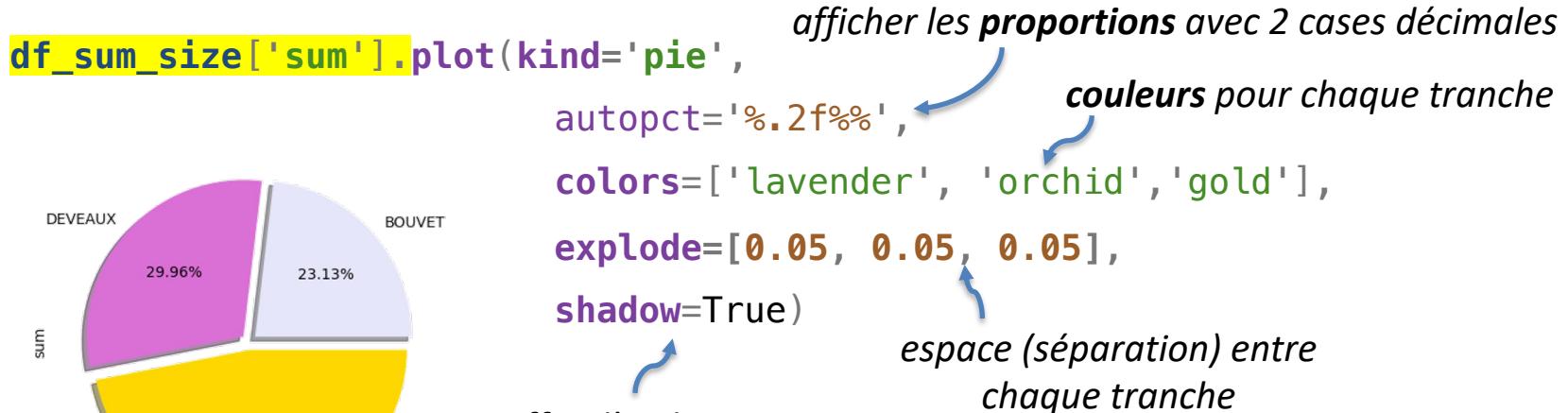
- Exercice : visualisation des données

- Trouver les total des ventes et le nombre de factures par vendeur

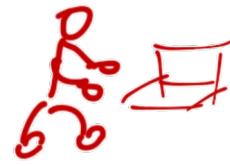
```
df_sum_size = ventes.groupby(by='VENDEUR')[['MONTANT']].agg(['sum', 'size'])
df_sum_size
```

*agg va nous créer 2 colonnes :
'sum' et 'size'*

- Réaliser un **graphique en camembert** des sommes ('sum')



Hands On !



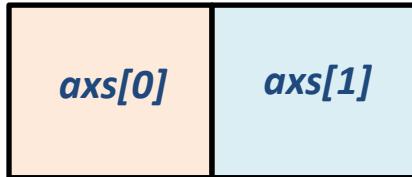
- Exercice : visualisation des données

- On essaie de faire une figure avec deux graphiques

axs : liste avec les zones pour les graphiques

```
fig, axs = plt.subplots(1, 2)
```

fig, axes = subplot (1, 2)



subplot : création de la figure avec 1 ligne et 2 colonnes

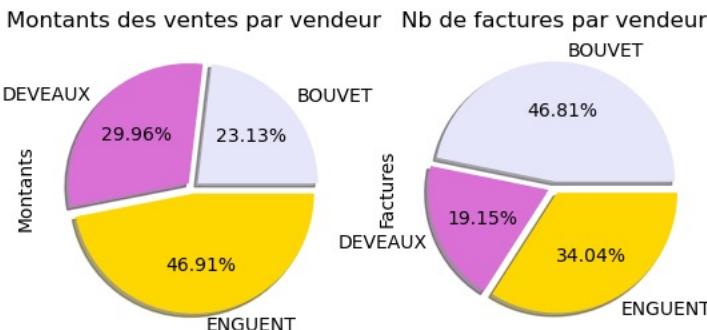
ax=axs[0] : on indique une zone précise pour le graphique

```
df_sum_size['sum'].plot(kind='pie', ax=axs[0],  
                        autopct='%.2f%%',  
                        colors=['lavender', 'orchid', 'gold'],  
                        explode=[0.05, 0.05, 0.05],  
                        shadow=True,  
                        title='Montants des ventes par vendeur',  
                        ylabel='Montants')
```

ax=axs[1] : on indique une zone précise pour le graphique

```
df_sum_size['size'].plot(kind='pie', ax=axs[1],  
                        autopct='%.2f%%',  
                        colors=['lavender', 'orchid', 'gold'],  
                        explode=[0.05, 0.05, 0.05],  
                        shadow=True,  
                        title='Nb de factures par vendeur',  
                        ylabel='Factures')
```

ylabel : label pour l'axe Y



VENDEUR	sum	size
BOUDET	18491.49	22
DEVEAUX	23956.76	9
ENGUENT	37509.79	16



Pour aller plus loin...

Nettoyage & Préparation des données

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- Changer le nom d'une colonne
- Eliminer un caractère indésirable
- Obtenir plusieurs informations à partir des dates (jour, mois, année, jour de la semaine, etc.)
- Transformation des données texte en donnée numérique avec des encodeurs
- Regrouper des données en catégorie (« chaud », « froid », « cher », « abordable », « pas cher »)
- Normalisation des données





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Changer le nom d'une colonne

- **Difficulté à résoudre :**

- Parfois, les noms des colonnes qu'on retrouve dans les données ne sont pas très représentatifs (ou comportent même des erreurs)

- **Solution possible :**

- On peut changer les noms des colonnes d'un DataFrame grâce à l'opération « **rename** »

DataFrame à modifier

```
df.rename ( columns={ ' MONTANT ' : 'MONTANT' } , inplace=True )
```

Dictionnaire avec colonnes à changer
{ ancien nom : nouveau nom }

inplace=True
Modifie le DataFrame

*On peut aussi se servir de l'attribut **columns** pour indiquer la colonne à supprimer par sa position (première colonne = position [0])*

Changer le nom d'une colonne



- Exemple : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »

- Sur le notebook « **ExoExtra1.ipynb** », on va lire le fichier « **VentesAgenceU.csv** »

```

import pandas as pd Ne pas oublier l'import
dfFactures = pd.read_csv
  ('http://kirschpm.fr/cours/PythonDataScience/files/VentesAgenceU.csv',
   delimiter=';', header=[0], index_col=[0])

```

dfFactures.info()

info

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

```

for col in dfFactures.columns :
  print('##{}##'.format(col))

```

Boucle pour afficher le nom de chaque colonne entre « ## » (et mieux voir les espaces)

```

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

```

Espace dans le nom de la colonne

Changer le nom d'une colonne



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - On modifie le nom de la colonne avec « **rename** »
 - Il faut indiquer le **nom à changer** et le **nouveau nom**

```
dfFactures.rename ( columns={' MONTANT ':'MONTANT'}, inplace=True)
```

Ancien nom Nouveau nom

On modifie le DataFrame

dfFactures.columns [4]

On peut aussi se servir de l'attribut « **columns** » qui contient les noms de toutes les colonnes, afin d'indiquer la colonne par sa position.

```
dfFactures.rename ( columns={dfFactures.columns[4] :'MONTANT'}, inplace=True)
```

Changer le nom d'une colonne



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - On modifie le nom de la colonne avec « **rename** »

```
dfFactures.rename ( columns={' MONTANT ':'MONTANT'}, inplace=True)
```

Ancien nom *Nouveau nom* *On modifie le DataFrame*

```
for col in dfFactures.columns :
    print('##{}##'.format(col))
```

```
dfFactures.info()
```

```
##DATE FACTURE##
##CODE CLIENT##
##SECTEUR##
##VENDEUR##
##MONTANT##
##Unnamed: 6##
```

Boucle pour afficher le nom de chaque colonne entre « ## » (et mieux voir les espaces)

Nouveau nom de la colonne

info

#	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

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



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Eliminer les caractères indésirables

- **Difficulté à résoudre :**
 - Parfois, dans nos données, on retrouve des **caractères indésirables** qu'il faut **supprimer** ou **remplacer** par d'autres
 - Par exemple, dans le fichier « **VentesAgenceU.csv** », colonne « **Montant** » :
 - On retrouve un « **_** » dans les chiffres : **1_123,5 → 1123.5**
 - Où une « **,** » à la place du « **.** » sur les chiffres : **123,5 → 123.5**
 - On doit « corriger » les données avant de pouvoir les **convertir** aux bons formats (**to_datetime**, **to_numeric**)
- **Solution possible :**
 - Appliquer l'opération « **str.replace** »



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Eliminer les caractères indésirables

Colonne où placer les données une fois nettoyées

```
df['MONTANT'] = df['MONTANT'].str.replace('_', '')
```

Considère le contenu de la colonne comme une chaîne de caractères (string)

Ça peut être la même colonne ou pas (nouvelle colonne)...

Caractère à remplacer

Nouveau caractère

Colonne avec les valeurs nettoyées

```
df['MONTANT'] = df['MONTANT'].apply(lambda x: str(x).replace('_', ''))
```

Pour chaque valeur x de la colonne 'Montant', on applique l'action indiquée par la fonction **lambda**

Colonne à modifier

Même chose, mais maintenant en utilisant l'opération **apply**

Caractère à remplacer

Nouveau caractère

On voit la valeur x comme une string

On remplace un caractère par un autre



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Eliminer les caractères indésirables



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - Toujours sur le notebook « **ExoExtra1.ipynb** »
 - Observer les données de la colonne « **MONTANT** » avec « **sample** »

```
dfFactures.sample(5)
```

- Remplacer le « `_` » par une chaîne vide
`''` pour les supprimer
- Vérifier le résultat avec « **sample** »

SECTEUR	VENDEUR	MONTANT
TOMATISME	ENGUENT	7_541,33
TOMATISME	BOUVET	1_459,92
ELECTRICITE	ENGUENT	246,46
TOMATISME	BOUVET	752,98
TOMATISME	BOUVET	198,85

```
dfFactures['MONTANT'] =  
    dfFactures['MONTANT'].str.replace('_', '')  
  
dfFactures['MONTANT'].sample(5)
```

CODE FACTURE	MONTANT
FA-2008-0014	917,38
FA-2008-0005	739,83
FA-2008-0034	1051,09
FA-2008-0020	3659,69
FA-2008-0047	204,01

Name: MONTANT, dtype: object

Eliminer les caractères indésirables



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - Remplacer le « , » par un « . »
 - Observer le résultat avec un « **sample** »

```
dfFactures['MONTANT'] =  
    dfFactures['MONTANT'].apply (lambda x: str(x).replace(',', '.',))
```

```
dfFactures['MONTANT'].sample(5)
```

CODE FACTURE	
FA-2008-0012	752.98
FA-2008-0036	485.24
FA-2008-0013	935.47
FA-2008-0024	9367.87
FA-2008-0037	<u>1709.07</u>

Name: MONTANT, dtype: object

Eliminer les caractères indésirables



- **Exemple** : colonne « **MONTANT** » fichier « **VentesAgenceU.csv** »
 - Une fois les données nettoyées, on peut les convertir en « **numérique** » avec l’opération « **to_numeric** » offerte par **Pandas**

```
dfFactures['MONTANT'] =  
    pd.to_numeric(dfFactures['MONTANT'], errors='coerce')  
  
dfFactures.info()
```

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	float64

L’option **errors = ‘coerce’** remplace par **NaN** les valeurs qu’il n’a pas su convertir. On n’a pas d’erreur dans ce cas, mais un **NaN** à la place.



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- **Obtenir plusieurs informations à partir des dates (jour, mois, année, jour de la semaine, etc.)**
- Transformation des données texte en donnée numérique avec des encodeurs
- Regrouper des données en catégorie (« chaud », « froid », « cher », « abordable », « pas cher »)
- Normalisation des données





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations à partir des dates

• Difficulté à résoudre :

- Parfois, on souhaite obtenir différentes informations à partir d'une date :
 - Jour du mois, année, semaine de l'année, jour de la semaine, etc.

• Solution possible :

- Une fois qu'on a convertit nos données en dates (à l'aide de « `to_datetime` »), on peut extraire des nombreuses informations

Colonne où garder le résultat	Colonne avec les dates	En tant que date (<code>DateTime</code> → <code>dt</code>), on lui demande des informations	Exemple
			2008-04-28
<code>df['JOUR FACTURE'] = df['DATE FACTURE'].dt.day</code>		<i>Jour uniquement</i>	28
<code>df['MOIS FACTURE'] = df['DATE FACTURE'].dt.month</code>		<i>Mois uniquement</i>	4
<code>df['ANNEE FACTURE'] = df['DATE FACTURE'].dt.year</code>		<i>Année uniquement</i>	2008
<code>df['JOUR SEMAINE'] = df['DATE FACTURE'].dt.weekday</code>		<i>Jour de la semaine</i>	0
<code>df['NOM JOUR'] = df['DATE FACTURE'].dt.day_name()</code>		<i>Jour de la sem. (texte)</i>	Monday
<code>df['NOM MOIS'] = df['DATE FACTURE'].dt.month_name()</code>		<i>Mois (texte)</i>	April
<code>df['SEMAINE ANNEE'] = df['DATE FACTURE'].dt.isocalendar().week</code>		<i>Sem. année</i>	18
	<i>Nouvelle version</i>		
	<i>Ancienne version : df['DATE FACTURE'].dt.weekofyear</i>		

Obtenir des informations des dates

ExoExtra1.ipynb



- **Exemple** : colonne « DATE FACTURE » fichier « VentesAgenceU.csv »
 - Toujours sur le notebook « ExoExtra1.ipynb », on va extraire plusieurs informations de la colonne « DATE FACTURE ».
 - Avant cela, on doit convertir la colonne « DATE FACTURE » en date avec l'opération « `to_datetime` »

Colonne avec les valeurs une fois converties

```
dfFactures['DATE FACTURE'] = pd.to_datetime(dfFactures['DATE FACTURE'], dayfirst=True)
```

Colonne à convertir

```
dfFactures['DATE FACTURE'].sample(5)
```

- Un « `sample` » nous permettra d'observer les résultats

CODE FACTURE	
FA-2008-0033	2008-08-14
FA-2008-0036	2008-03-09
FA-2008-0015	2008-04-28
FA-2008-0034	2008-08-23
FA-2008-0041	2008-09-13

Name: DATE FACTURE, dtype: datetime64[ns]

Obtenir des informations des dates



- **Exemple** : colonne « **DATE FACTURE** » fichier « **VentesAgenceU.csv** »
 - On peut désormais ajouter au *DataFrame* « **dfFactures** » des nouvelles colonnes avec les informations suivantes :
 - **Jour** → nouvelle colonne « **JOUR FACTURE** »
 - **Mois** → nouvelle colonne « **MOIS FACTURE** »
 - **Année** → nouvelle colonne « **ANNEE FACTURE** »
 - **Jour de la semaine** → nouvelle colonne « **JOUR SEMAINE** »
 - **Jour de la semaine (en texte)** → nouvelle colonne « **NOM JOUR** »
 - **Mois de l'année (en texte)** → nouvelle colonne « **NOM MOIS** »
 - **Semaine de l'année** → nouvelle colonne « **SEMAINE ANNEE** »
 - A nouveau, un « **sample** » nous permettra d'observer les résultats



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates

ExoExtra1.ipynb

Nouvelle colonne

Colonne avec la date

```
dfFactures['JOUR FACTURE'] = dfFactures['DATE FACTURE'].dt.day
```

```
dfFactures['MOIS FACTURE'] = dfFactures['DATE FACTURE'].dt.month
```

```
dfFactures['ANNEE FACTURE'] = dfFactures['DATE FACTURE'].dt.year
```

```
dfFactures['JOUR SEMAINE'] = dfFactures['DATE FACTURE'].dt.weekday
```

```
dfFactures['NOM JOUR'] = dfFactures['DATE FACTURE'].dt.day_name()
```

```
dfFactures['NOM MOIS'] = dfFactures['DATE FACTURE'].dt.month_name()
```

Information qu'on souhaite
dt.xxx

Attention aux parenthèses ()

```
dfFactures['SEMAINE ANNEE'] = dfFactures['DATE FACTURE'].dt.isocalendar().week
```

```
dfFactures.sample(5)
```

Pour info

Septembre 2008

	Di	Lu	Ma	Me	Je	Ve	Sa
	1	2	3	4	5	6	
	7	8	9	10	11	12	13
	14	15	16	17	18	19	20
	21	22	23	24	25	26	27
	28	29	30				

ANT

JOUR FACTURE	MOIS FACTURE	ANNEE FACTURE	JOUR SEMAINE	NOM JOUR	NOM MOIS	SEMAINE ANNEE
--------------	--------------	---------------	--------------	----------	----------	---------------

Nouvelles colonnes

FA-2008-0046	2008-09-25	CLI206	AUTOMATISME	ENGUENT	7432.82	1.04	25	9	2008	3	Thursday	September	39
FA-2008-0018	2008-03-06	CLI222	AUTOMATISME	ENGUENT	7432.82	6	3	2008	3	Thursday	March	10	
FA-2008-0022	2008-06-30					30	6	2008	0	Monday	June	27	
FA-2008-0036	2008-03-09					9	3	2008	6	Sunday	March	10	
FA-2008-0044	2008-09-19	CLI206	AUTOMATISME	BOUVET	198.85	19	9	2008	4	Friday	September	38	

Les jours de la semaine sont comptés à partir de 0 pour Lundi
("Monday")



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations à partir des dates

- On peut faire la même chose avec le **temps**, lorsqu'on a des informations temporelles (**date et heure**, ou juste **heure**).
- Par contre, les données doivent contenir, dès le départ, l'information sur l'heure.
 - Exemple : la colonne « DATE FACTURE » ne contient pas cette information, les opérations ci-dessous ne marcheront pas sur ces données

Colonne avec l'information

(date / heure)

2011-12-16 19:42:23+00:00

```
df['time'] = df['datetime'].dt.time           19:42:23
df['hour'] = df['datetime'].dt.hour          19
df['minute'] = df['datetime'].dt.minute       42
```

*Colonne où garder
le résultat*

*En tant que DateTime (dt), on
lui demande des informations*

*Information demandée
(heure:min:seconde , juste l'heure,
juste les minutes)*



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates



- **Exemple** : colonne « **pickup_datetime** » fichier « **mini_taxi.csv** »
 - Sur le notebook « **ExoExtra2.ipynb** », on va lire le fichier « **mini_taxi.csv** ».
 - Attention à ne pas oublier le « **import pandas as pnd** »
(nouveau notebook, donc à nouveau import)
 - A l'aide d'un « **info** », on remarque la colonne « **pickup_datetime** » est un « **object** ». On va donc devoir la convertir avec « **to_datetime** »

```
import pandas as pnd
```

```
dfTaxi = pnd.read_csv(  
    'http://kirschpm.fr/cours/PythonDataScience/files/mini_taxi.csv',  
    index_col=[0])
```

```
dfTaxi.info()
```

info ↗

#	Column	Non-Null Count	Dtype
0	fare_amount	5999 non-null	float64
1	pickup_datetime	5999 non-null	object
2	pickup_longitude	5999 non-null	float64
3	pickup_latitude	5999 non-null	float64
4	dropoff_longitude	5999 non-null	float64



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates



- Exemple : colonne « pickup_datetime » fichier « mini_taxi.csv »

```
dfTaxi['pickup_datetime'] = pd.to_datetime(dfTaxi['pickup_datetime'])
```

```
dfTaxi.info()
```

Index: 5999 entries, 2009-06-15 17:26:21.000000 to 2014-12-

Data columns (total 7 columns):

#	Column	Non-Null Count	Dtype
0	fare_amount	5999 non-null	float64
1	pickup_datetime	5999 non-null	datetime64[ns, UTC]
2	pickup_longitude	5999 non-null	float64
3	pickup_latitude	5999 non-null	float64
4	dropoff lonaitude	5999 non-null	float64

info

dfTaxi.head()

key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	40.712278	1
2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.979268	40.782004	1
2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.991242	40.750562	2
2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.991567	40.758092	1
2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.956655	40.783762	1



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates



- **Exemple** : colonne « `pickup_datetime` » fichier « `mini_taxi.csv` »
 - On va ajouter au *DataFrame* « `dfTaxi` » des nouvelles colonnes avec des informations qu'on pourra extraire des **heures de pickup** :
 - **Heure:minute** → nouvelle colonne « `time` »
 - **Juste Heure** → nouvelle colonne « `hour` »
 - **Juste Minutes** → nouvelle colonne « `time` »
 - Un « `sample` » nous permettra d'observer les résultats

```
dfTaxi['time'] = dfTaxi['pickup_datetime'].dt.time  
dfTaxi['hour'] = dfTaxi['pickup_datetime'].dt.hour  
dfTaxi['minute'] = dfTaxi['pickup_datetime'].dt.minute
```

fare_amount	pickup_datetime	time	hour	minute
key		...		
2010-01-20 20:39:00.0000009	9.7 2010-01-20 20:39:00+00:00	20:39:00	20	39
2013-08-09 21:57:46.0000003	7.0 2013-08-09 21:57:46+00:00	21:57:46	21	57
2011-09-01 19:26:46.0000003	3.3 2011-09-01 19:26:46+00:00	19:26:46	19	26
2010-05-22 23:29:57.0000003	25.3 2010-05-22 23:29:57+00:00	23:29:57	23	29



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Obtenir des informations des dates



- **Exemple** : colonne « `pickup_datetime` » fichier « `mini_taxi.csv` »
 - Avec l'information sur l'**heure**, on peut déduire si la course s'est déroulée la **nuit ou non** (ce qui, normalement, aurait une incidence sur le prix)
 - On va utiliser l'opération « `apply` » pour créer une nouvelle colonne « `night` » avec « `True` » si la course s'est déroulée **avant 7h** ou **après 19h**.

Colonne où garder
le résultat

```
dfTaxi['night'] = dfTaxi['hour'].apply(lambda x: (x<7 or x>19))
```

```
dfTaxi.sample(5)
```

Colonne avec l'heure

Pour **chaque valeur x de la colonne**,
on teste si **x<7 ou si x>19**

La colonne « `night` » contiendra le
résultat de ce test

fare_amount	pickup_datetime	...	time	hour	minute	night
key						
2009-09-08 18:32:47.0000004	9.3 2009-09-08 18:32:47+00:00		18:32:47	18	32	False
2012-04-20 15:01:33.0000004	5.7 2012-04-20 15:01:33+00:00	...	15:01:33	15	1	False
2011-03-05 01:13:04.0000002	9.3 2011-03-05 01:13:04+00:00		01:13:04	1	13	True



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- ✓ Obtenir plusieurs informations à partir des dates
(jour, mois, année, jour de la semaine, etc.)
- **Transformation des données texte en donnée numérique avec des encodeurs**
- Regrouper des données en catégorie (« chaud », « froid »,
« cher », « abordable », « pas cher »)
- Normalisation des données





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

- **Difficulté à résoudre :**

- Lorsque certaines **colonnes** contiennent des **textes** (des « **catégories** »), il faut les « **traduire** » en **valeurs numériques**
- Exemples :
 - Genre : « male » / « female »,
 - Navigateurs : « Safari » / « Chrome » / « Firefox »
- Les différents modèles de ML proposés sur **SKLearn ne savent pas** travailler avec de texte

- **Solution possible :**

- Utiliser des « **encodeurs** »
 - Les encodeurs **traduisent** des **catégories** exprimées en texte en format **numérique**
- Différents types de ces « **encodeurs** » existent
 - Proposés par la bibliothèque **Sklearn**, mais aussi par la bibliothèque **Pandas**



UNIVERSITÉ PARIS 1

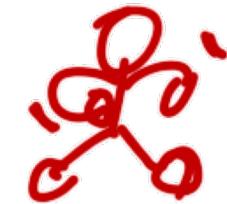
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

- Transformation des données en **Sklearn**
 - SKLearn ne manipule que des données numériques
 - On considère que le **texte** indique des **catégories**
 - *Pas un texte « libre »*

['approves' , 'disapproves']  [0 1]



• Encoders sur Sklearn

- Différents **encoders** disponibles sur **sklearn.preprocessing**
 - **LabelEncoder** : Transformation des **labels** (**target**) en valeurs entières (0 à n-1)
 - **OrdinalEncoder** : Transformation des **données** (**features**) en valeurs entières (de 0 à n-1)
 - **OneHotEncoder** : Transformation des **données** (**features**) en valeurs **binaires**
- Quelque soit l'encoder, on suit les mêmes **étapes** :
Création encodeur → entraînement (« fit ») → application (« transform »)



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	approves

Target (classes)
« Y_set »

Y = labEnc.inverse_transform(Yenc)

Transformation inverse
(des valeurs aux labels)

['approves' 'disapproves' 'approves' 'approves' 'disapproves' 'disapproves'

'approves' 'approves'] ← Y_set : valeurs originales

[0 1 0 0 1 1 0 0] ← Y_enc : valeurs encodées

LabelEncoder

Conversion des targets en valeurs numériques

```
from sklearn.preprocessing import LabelEncoder
```

```
labEnc = LabelEncoder()  
labEnc.fit( Y_set )
```

Création et
entraînement de
l'encoder

Yenc = labEnc.transform(Y_set)

Transformation
des valeurs

valeurs

labEnc.classes_ ← Classes retrouvées

['approves' 'disapproves']



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple :** transformation d'un « **target** » texte en numérique

- Sur le nouveau notebook « **ExoExtra3.ipynb** »
on va créer un DataFrame simple,
juste pour nos tests

```
import pandas as pd
```

```
dfCategories = pd.DataFrame (  
    [ ['male', 'from US', 'uses Safari', 'approves'],  
     ['female', 'from Europe', 'uses Firefox', 'disapproves'] ,  
     ['female', 'from US', 'uses Safari', 'approves'],  
     ['male', 'from Europe', 'uses Safari', 'approves'],  
     ['female', 'from US', 'uses Firefox', 'disapproves'] ,  
     ['male', 'from Europe', 'uses Chrome', 'disapproves'] ,  
     ['female', 'from Asia', 'uses Chrome', 'approves'],  
     ['male', 'from Asia', 'uses Chrome', 'approves'] ],  
    columns=['sex', 'region','browser', 'vote' ] )
```

```
dfCategories
```

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	approves



- **Exemple :** transformation d'un « **target** » texte en numérique
 - On va créer maintenant un **LabelEncoder**
 - On entraîne l'encoder avec la colonne « **vote** »

```
from sklearn.preprocessing import LabelEncoder
```

Ne pas oublier l'import

```
labEnc = LabelEncoder()
```

Création de l'objet encoder

```
labEnc.fit( dfCategories[ 'vote' ] )
```

*Entrainement de l'encoder avec la colonne
contenant les catégories qu'on souhaite
transformer.*

*On peut voir les **classes** qu'il a reconnu (après le **fit**)*

```
print (labEnc.classes_)
```

*La valeur « approves »
sera transformée en 0*

['approves' 'disapproves']

0

*« disapproves »
en 1*



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple :** transformation d'un « **target** » texte en numérique
 - Maintenant, on peut utiliser l'encoder pour transformer la colonne « **vote** »
 - On va créer une nouvelle colonne « **code_vote** » avec les valeurs transformées

Colonne qui va recevoir les valeurs transformées

Colonne à transformer

dfCategories['code_vote'] = labEnc.transform(dfCategories['vote'])

Transformation des données selon le modèle d'encoder entraîné

dfCategories

	sex	region	browser	vote	code_vote
0	male	from US	uses Safari	approves	0
1	female	from Europe	uses Firefox	disapproves	1
2	female	from US	uses Safari	approves	0
3	male	from Europe	uses Safari	approves	0
4	female	from US	uses Firefox	disapproves	1
5	male	from Europe	uses Chrome	disapproves	1
6	female	from Asia	uses Chrome	approves	0
7	male	from Asia	uses Chrome	approves	0

Nouvelle colonne avec les valeurs transformées



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



• Exemple : transformation d'un « target » texte en numérique

- On peut procéder à la **transformation inverse** d'un *array* de valeurs et obtenir les catégories correspondantes
- On va créer d'abord un *array* avec une séquence aléatoire de 0 et 1

```
import numpy as np
```

On va utiliser la bibliothèque Numpy pour générer l'array de n's aléatoires

```
aleatoire = np.random.randint(low=0, high=2, size=5)
```

```
print (aleatoire)
```

```
[0 1 0 1 0]
```

Randint produit un array de taille « size » (5) avec des chiffres entre « low » (0) et « high » (2), celui-ci exclut (donc de 0 et 1)

```
cate_aleatoire = labEnc.inverse_transform(aleatoire)
```

```
print (cate_aleatoire)
```

```
['approves' 'disapproves' 'approves' 'disapproves' 'approves']
```

On réalise la transformation inverse

On obtient les classes correspondantes



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	

Features (variables)
« X_set »

L'encoder attend
un **DataFrame**
avec un ensemble
de **features**

X_set : valeurs originales

```
['male' 'from Europe' 'uses Safari']
['female' 'from US' 'uses Firefox']
['male' 'from Europe' 'uses Chrome']
['female' 'from Asia' 'uses Chrome']
```

Xord : valeurs encodées

[1.	1.	2.]
[0.	2.	1.]
[1.	1.	0.]
[0.	0.	0.]

OrdinalEncoder

Conversion des **features** en valeurs **entiers**

```
from sklearn.preprocessing import OrdinalEncoder
```

```
ordEnc = OrdinalEncoder()
ordEnc.fit( X_set )
```

Création et
entraînement de
l'encoder

```
Xord = ordEnc.transform( X_set )
```

Transformation
des valeurs

valeurs

```
X = ordEnc.inverse_transform(Xord)
```

Transformation inverse
(des valeurs aux données)

ordEnc.categories_

Liste des
catégories

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique
 - On va désormais utiliser un **OrdinalEncoder**, qu'on va créer
 - On entraîne l'encoder avec un **DataFrame** ne contenant que colonne « **sex** »

```

from sklearn.preprocessing import OrdinalEncoder
Ne pas oublier l'import

ordEnc = OrdinalEncoder()
Création de l'objet encoder

ordEnc.fit( dfCategories[['sex']] )
Entrainement de l'encoder
Attention à l'usage des [[ ]], car les
encoders pour les features attendent tout
un DataFrame et pas une simple colonne
(comme le LabelEncoder)
    
```

On peut voir les **catégories** qu'il a reconnu (*après le fit*)

```
print (ordEnc.categories_)
```

La valeur « female » sera transformée en 0

0
1

« male » sera transformé en 1

```
[array(['female', 'male'], dtype=object)]
```



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : transformation d'une « **feature** » texte en numérique
 - On peut maintenant utiliser un **OrdinalEncoder** pour transformer les valeurs de la colonne « **sex** »
 - On va ajouter une nouvelle colonne « **code_sex** » avec les valeurs transformées

*Colonne qui va recevoir
les valeurs transformées*

```
dfCategories['code_sex'] = ordEnc.transform( dfCategories [['sex']] )
```

dfCategories

	sex	region	browser	vote	code_vote	code_sex
0	male	from US	uses Safari	approves	0	1.0
1	female	from Europe	uses Firefox	disapproves	1	0.0
2	female	from US	uses Safari	approves	0	0.0
3	male	from Europe	uses Safari	approves	0	1.0
4	female	from US	uses Firefox	disapproves	1	0.0
5	male	from Europe	uses Chrome	disapproves	1	1.0
6	female	from Asia	uses Chrome	approves	0	0.0
7	male	from Asia	uses Chrome	approves	0	1.0

DataFrame contenant les features à transformer.

Attention donc à l'usage des [[[]]] afin d'avoir un DataFrame avec les colonnes qu'on veut transformer

Nouvelle colonne avec les valeurs transformées



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique
 - Pour illustrer la **transformation inverse**, on va utiliser les valeurs de notre *array* de valeurs 0 et 1 aléatoires
 - Il faut d'abord créer un *DataFrame* à partir de cet *array*

```
dfAleatoire = pnd.DataFrame(aleatoire)
print (dfAleatoire)

ordEnc.inverse_transform( dfAleatoire )
```

```
print(dfAleatoire)
```

	0
0	1
1	0
2	1
3	0
4	1

Données dfAleatoire

```
array([['male'],
       ['female'],
       ['male'],
       ['female'],
       ['male']], dtype=object)
```

On va utiliser la variable « **aleatoire** » qu'on avait créé dans l'exercice précédent et construire un **DataFrame** avec (puisque **OrdinalEncoder** attend un **DataFrame** avec des valeurs)

Catégories obtenues avec la transformation inverse



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
-				



Chaque colonne sera « éclatée » en plusieurs, en fonction du nombre de catégories présentes.

sex	region	browser
[0. 1.]	[0. 0. 1.]	[0. 0. 1.]
[1. 0.]	[0. 1. 0.]	[0. 1. 0.]
[1. 0.]	[0. 0. 1.]	[0. 0. 1.]

```
print(Xoh.toarray())
```

OneHotEncoder

Conversion des **features** en valeurs **binaires**

```
from sklearn.preprocessing import OneHotEncoder
```

```
ohEnc = OneHotEncoder()
ohEnc.fit( X_set )
```

Création et entraînement de l'encoder

```
Xoh = ohEnc.transform( X_set )
```

Transformation des valeurs

DataFrame avec les valeurs

```
X = ohEnc.inverse_transform(Xoh)
```

Transformation inverse (des valeurs aux données)

```
ohEnc.get_feature_names()
```

Liste des catégories



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : transformation d'une « **feature** » texte en numérique
 - On va utiliser maintenant un **OneHotEncoder** pour encoder un **DataFrame** avec la colonne « **region** ».

```
from sklearn.preprocessing import OneHotEncoder
```

Ne pas oublier l'import

```
ohEnc = OneHotEncoder()
```

Création de l'objet encoder

```
ohEnc.fit( dfCategories[['region']] )
```

Entrainement de l'encoder
Attention à l'usage des [[]], car les encoders pour les features attendent tout un DataFrame et pas une simple colonne (comme le LabelEncoder)

On peut voir les catégories qu'il a reconnu (après le fit)

```
print (ohEnc.get_feature_names())
```

```
['x0_from Asia' 'x0_from Europe' 'x0_from US']
```



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique
 - Une fois entraîné, on peut transformer les valeurs présentes dans la colonne « **region** ».

```
Xoh = ohEnc.transform( dfCategories[['region']] )
```

```
print ( Xoh.toarray() )
```

On va créer un *nouveau DataFrame* avec ces valeurs afin de pouvoir les ajouter à notre DataFrame original

	x0_from Asia	x0_from Europe	x0_from US
0	0.0	0.0	1.0
1	0.0	1.0	0.0
2	0.0	0.0	1.0
3	0.0	1.0	0.0
4	0.0	0.0	1.0
5	0.0	1.0	0.0
6	1.0	0.0	0.0
7	1.0	0.0	0.0

DataFrame avec les valeurs à transformer

```
dfXoh = pd.DataFrame(Xoh.toarray(),  
columns=ohEnc.get_feature_names())
```

dfXoh

91



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple :** transformation d'une « **feature** » texte en numérique
 - Puis, on peut ajouter les nouvelles valeurs transformées à notre DataFrame « **dfCatégories** » (sur un nouveau DataFrame ou sur le même).

```
dfCategories_OneHot = pd.concat( [ dfCategories, dfXoh ] ,  
                                 axis='columns' )
```

Avec l'opération **concat** de Pandas

ou

Avec l'opération **join** propre aux **DataFrame**

Deux manières de combiner un DataFrame à un autre :

```
dfCategories_OneHot = dfCategories.join(dfXoh)
```

dfCategories_OneHot

	sex	region	browser	vote	code_vote	code_sex	x0_from Asia	x0_from Europe	x0_from US
0	male	from US	uses Safari	approves	0	1.0	0.0	0.0	1.0
1	female	from Europe	uses Firefox	disapproves	1	0.0	0.0	1.0	0.0
2	female	from US	uses Safari	approves	0	0.0	0.0	0.0	1.0
3	male	from Europe	uses Safari	approves	0	1.0	0.0	1.0	0.0
4	female	from US	uses Firefox	disapproves	1	0.0	0.0	0.0	1.0



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

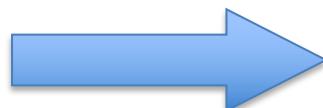
• *Encoding - Bibliothèque Pandas*

- La bibliothèque **Pandas** propose un *encoding* de type **One Hot** appelé « **get_dummies** »
- Pas besoin de phase d'entraînement, on l'utilise directement

df_weather.value_counts(df_weather['Description'])		Description	
		Normal	4992
		Warm	2507
		Cold	2501
			dtype: int64

On a 3 catégories (« Normal », « Warm » et « Cold »), chacune deviendra une nouvelle colonne.

Description		Description_Cold	Description_Normal	Description_Warm
Cold		1	0	0
Warm	pnd.get_dummies (df_weather)	0	0	1
Normal		0	1	0
Cold		1	0	0
Cold		1	0	0





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation des données texte en numérique

• Encoding - Bibliothèque Pandas

- On peut réduire le nombre de colonnes créées avec l'option « **drop_first = True** »

```
df_dummies = pd.get_dummies ( df_weather, drop_first=True )
```

Dans **OneHotEncoder**, on écrit
OneHotEncoder(drop='first')

On représente la 1^{ère} catégorie par la combinaison de 0 sur les autres

On obtient 1 colonne en moins

```
Data columns (total 9 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --  
 0   Temperature_c    10000 non-null   float64  
 1   Humidity          10000 non-null   float64  
 2   Wind_Speed_kmh   10000 non-null   float64  
 3   Wind_Bearing_degrees 10000 non-null int64  
 4   Visibility_km     10000 non-null   float64  
 5   Pressure_millibars 10000 non-null   float64  
 6   Rain              10000 non-null   int64  
 7   Description_Normal 10000 non-null   uint8  
 8   Description_Warm   10000 non-null   uint8
```

Nouvelles colonnes

Là où on avait « 1 » sur la colonne « **Description_cold** », on se retrouve juste avec « 0 » sur les autres colonnes.

	Description_Normal	Description_Warm
0	cold	0
1	0	1
2	1	0
3	0	0
4	0	0

df_dummies.head()



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : transformation d'une « **feature** » texte en numérique
 - Toujours sur le notebook « **ExoExtra3.ipynb** », on va utiliser l'encoder « **get_dummies** » pour transformer les colonnes « **region** » et « **browser** ».

Si on a bien fait

```
import pandas as pd  
au début de notre notebook
```

```
dummies = pd.get_dummies (dfCategories [['region', 'browser']],  
                           drop_first=True)
```

dummies

DataFrame avec les colonnes à transformer (notre « X set »)

*Option **drop_first** pour réduire le nombre de colonnes*

	region_from Europe	region_from US	browser_uses Firefox	browser_uses Safari
0	0	1	0	1
1	1	0	1	0
2	0	1	0	1
3	1	0	0	1
4	0	1	1	0
5	1	0	0	0
6	0	0	0	0
7	0	0	0	0



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : transformation d'une « **feature** » texte en numérique
 - Une fois les valeurs transformées, on peut les ajouter à notre DataFrame

```
dfCategories_dummies = pd.concat ( [ dfCategories, dummies ],  
                                 axis='columns' )
```

DataFrame original et DataFrame avec les données transformées

dfCategories_dummies  *DataFrame avec toutes les données (originelles et transformées)*

	sex	region	browser	vote	code_vote	code_sex	region_from Europe	region_from US	browser_uses Firefox	browser_uses Safari
0	male	from US	uses Safari	approves	0	1.0	0	1	0	1
1	female	from Europe	uses Firefox	disapproves	1	0.0	1	0	1	0
2	female	from US	uses Safari	approves	0	0.0	0	1	0	1
3	male	from Europe	uses Safari	approves	0	1.0	1	0	0	1
4	female	from US	uses Firefox	disapproves	1	0.0	0	1	1	0
5	male	from Europe	uses Chrome	disapproves	1	1.0	1	0	0	0
6	female	from Asia	uses Chrome	approves	0	0.0	0	0	0	0
7	male	from Asia	uses Chrome	approves	0	1.0	0	0	0	0

Transformation des données texte en numérique





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- ✓ Obtenir plusieurs informations à partir des dates
(jour, mois, année, jour de la semaine, etc.)
- ✓ Transformation des données texte en donnée numérique
avec des encodeurs
- **Regrouper des données en catégorie**
- Normalisation des données





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Regrouper des données en catégorie

- **Difficulté à résoudre :**

- Parfois, on souhaite regrouper nos données en catégories
 - Par exemple : au lieu d'avoir des valeurs d'âge (6, 20, 56...), avoir des classes d'âge (enfant, adulte, senior...)

- **Solution possible :**

- Transformer une variable quantitative (âge) en variable qualitative (classe d'âge)
- On appelle ça faire de la « **discrétisation** »
- On peut utiliser les opérations « **cut** » et « **qcut** » proposées par **Pandas** pour trouver des catégories à partir des données
 - Opération « **cut** » : organise les données en n catégories (on fournit n)
 - Opération « **qcut** » : organise les données en n catégories et s'assure que celles-ci sont **équilibrées** (avec une **répartition constante** d'individus)



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Regroupement des données en catégories

- Opérations « **cut** » et « **qcut** » : plusieurs possibilités

Colonne qui recevra les catégories

```
df['categorie'] = pnd.cut ( df['prix'], bins=4 )
```

Colonne avec les données à discréteriser

```
df['categorie'] = pnd.cut ( df['prix'], bins=4,
```

Nombre de catégories à utiliser

```
labels=['pas cher', 'normal',  
'cher', 'très cher'])
```

Étiquettes pour les catégories

```
df['categorie'] = pnd.cut ( df['prix'], bins=[ df['prix'].min(),
```

30, 50, 100,
df['prix'].max()])

Tranches pour les catégories

min-30, 30-50, 50-100, 100-max

Option **labels** aussi possible avec **qcut**

```
df['qcategorie'] = pnd.qcut( df['prix'] , q=4 )
```

Colonne qui recevra les catégories

Colonne avec les données

Nombre de catégories souhaitées



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Regrouper des données en catégorie



- **Exemple :** extraire des catégories de prix
 - Sur un nouveau notebook « **ExoExtra4.ipynb** », on va lire le fichier « **mini_taxi.csv** ».
 - Attention à ne pas oublier le « **import pandas as pd** » (**nouveau notebook → import**)
 - On va essayer d'extraire des catégories de prix à partir de la colonne « **fare_amount** »

```
import pandas as pd
```

```
dfTaxi =  
pd.read_csv('http://kirschpm.fr/cours/PythonDataScience/files/mini_taxi.csv',  
            index_col=[0])
```

```
dfTaxi.info()
```

```
Index: 5999 entries, 2009-06-15 17:26:21.000000  
Data columns (total 7 columns):
```

#	Column	Non-Null Count	Dtype
---	---	-----	-----
0	fare_amount	5999 non-null	float64
1	pickup_datetime	5999 non-null	object
2	pickup_longitude	5999 non-null	float64
3	pickup_latitude	5999 non-null	float64

Regrouper des données en catégorie



- **Exemple** : extraire des catégories de prix
 - Afin de mieux comprendre les valeurs qu'on retrouve dans la colonne « **fare_amount** », on va utiliser l'opération « **describe** »

```
dfTaxi['fare_amount'].describe()
```

count	5999.000000
mean	11.385616
std	9.805378
min	-2.900000
25%	6.000000
50%	8.500000
75%	12.900000
max	180.000000

Valeurs négatives

Name: fare_amount, dtype: float64

```
dfTaxi.loc[dfTaxi['fare_amount'] < 0]
```

key	fare_amount	pickup_datetime	pickup_lo
2010-03-09 23:37:10.0000005	-2.9	2010-03-09 23:37:10 UTC	-73.
2015-03-22 05:14:27.0000001	-2.5	2015-03-22 05:14:27 UTC	-74



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



• Exemple : extraire des catégories de prix

- On ne garde que les lignes avec un prix « correct » (c.a.d. prix ≥ 0)

On remplace `dfTaxi` par l'ensemble de valeurs qui vérifient la condition

```
dfTaxi = dfTaxi.loc[ dfTaxi['fare_amount'] >= 0 ]
```

```
dfTaxi['fare_amount'].describe()
```

```
dfTaxi.sample(5)
```

Moins de valeurs dans le DataFrame

	count	mean	std	min	25%	50%	75%	max	Name: fare_amount, dtype: float64
count	5997.000000	11.390313	9.803637	0.010000	6.000000	8.500000	12.900000	180.000000	Plus de valeur négative

key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
2009-02-14 08:13:00.000000022	8.1	2009-02-14 08:13:00 UTC	-73.984345	41.890200
2010-12-18 02:34:31.00000003	6.5	2010-12-18 02:34:31 UTC	-73.984948	41.890200
2010-01-11 15:42:00.0000000186	7.3	2010-01-11 15:42:00 UTC	-73.981355	41.890200
2015-02-03 08:02:33.00000003	4.5	2015-02-03 08:02:33 UTC	-73.988861	41.890200
2015-03-22 16:37:27.00000003	4.5	2015-03-22 16:37:27 UTC	-73.981056	41.890200



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : extraire des catégories de prix
 - Maintenant qu'on a des données propres, on peut organiser les prix (colonne « `fare_amount` ») en 4 catégories
 - On créer 2 nouvelles colonnes dans le *DataFrame* `dfTaxi`
 - Colonne « `cat_prix` » avec les catégories
 - Colonne « `cat_prix_label` » avec les catégories en label

```
dfTaxi['cat_prix']= pnd.cut (dfTaxi['fare_amount'], bins=4)
```

```
dfTaxi['cat_prix_label'] = pnd.cut (dfTaxi['fare_amount'], bins=4,  
labels=['pas cher', 'normal', 'cher', 'très cher'])
```

```
dfTaxi.sample(5)
```

	fare_amount	pickup_datetime		passenger_count	cat_prix	cat_prix_label
key			...			
2013-08-31 15:49:00.000000034	6.0	2013-08-31 15:49:00 UTC	...	1	(-0.17, 45.008]	pas cher
2015-02-27 07:50:56.00000003	10.5	2015-02-27 07:50:56 UTC	...	1	(-0.17, 45.008]	pas cher
2015-05-19 06:43:30.00000006	11.0	2015-05-19 06:43:30 UTC	...	2	(-0.17, 45.008]	pas cher



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : extraire des catégories de prix
 - A l'aide de l'opération « `value_counts` », on peut vérifier la distribution des individus dans les classes
 - On observe que cette distribution n'est pas homogène, vu que la première catégorie (jusqu'à \$45) regroupe une majorité d'individus

```
print (dfTaxi['cat_prix'].value_counts())
```

(-0.17, 45.008]	5876
(45.008, 90.005]	118
(135.002, 180.0]	2
(90.005, 135.002]	1

*Catégories très déséquilibrées,
ce qui peut poser problème pour
les modèles de ML*



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE
ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Transformation de texte en numérique



- **Exemple** : extraire des catégories de prix

- L'opération « **qcut** » permet alors d'obtenir des catégories équilibrées, c.a.d. avec une répartition uniforme des individus entre les catégories
- On va l'utiliser pour faire une nouvelle répartition des valeurs de la colonne « **fare_amount** »

```
dfTaxi['qcat_prix'] = pnd.qcut( dfTaxi['fare_amount'], q=4 )
```

```
print (dfTaxi['qcat_prix'].value_counts())
```

```
dfTaxi.sample(5)
```

(6.0, 8.5]	1594
(0.0090000000000001, 6.0]	1542
(12.9, 180.0]	1470
(8.5, 12.9]	1391

Name: qcat_prix, dtype: int64

	fare_amount	cat_prix	cat_prix_label	qcat_prix
key				
2012-08-11 17:31:14.0000004	8.5	(-0.17, 45.008]	pas cher	(6.0, 8.5]
2010-11-27 01:51:28.0000007	6.9	(-0.17, 45.008]	pas cher	(6.0, 8.5]
2014-06-10 14:18:29.0000001	9.5	(-0.17, 45.008]	pas cher	(8.5, 12.9]

Catégories équilibrées
Individus répartis de manière équilibrée entre les catégories



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Plan : Nettoyage & Préparation des données

Difficultés à résoudre :

- ✓ Changer le nom d'une colonne
- ✓ Eliminer un caractère indésirable
- ✓ Obtenir plusieurs informations à partir des dates
(jour, mois, année, jour de la semaine, etc.)
- ✓ Transformation des données texte en donnée numérique
avec des encodeurs
- ✓ Regrouper des données en catégorie
- **Normalisation des données**



Normalisation

ExempleScaler.ipynb



- **Normalisation**

- Certains algorithmes vont être particulièrement sensibles à **l'ordre de grandeur des variables**
 - Régression linéaire, K-Means...
- Lorsque celle-ci est trop différente, une **normalisation** peut s'avérer nécessaire
 - Ex : prix des maisons (500 000 €) et taux bancaires (0,05)
- Différent types de « **scalers** » existent
 - Paquet **sklearn.preprocessing**
 - **StandardScaler** : normalisation basée sur la moyenne
 - **MinMaxScaler** : normalisation à partir des valeurs de min et max
 - **RobustScaler** : normalisation plus « robuste » (moins sensible aux *outliers*), basée sur la moyenne

Normalisation

ExempleScaler.ipynb



• Normalisation

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
x_train_scal = scaler.fit_transform(x_train)
```

```
x_test_scal = scaler.transform(x_test)
```

On entraîne le Scaler avec les données d'entraînement

On transforme les données de test avant de les utiliser

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
```

```
x_train_scal = scaler.fit_transform(x_train)
```

```
x_test_scal = scaler.transform(x_test)
```

On peut indiquer au **MinMaxScaler** l'intervalle des données à utiliser pour la normalisation.

Normalisation

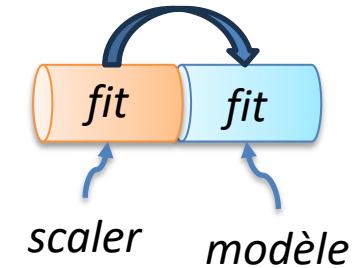
ExempleScaler.ipynb



• Pipeline

- Puisque les données devront être systématiquement transformées avant d'être utilisées, on peut définir un « **pipeline** » de transformation
- Dans le pipeline, les données sont transmises d'une opération à une autre, dans une chaîne :
 - La sortie du **fit** du **scaler** est transmise au **fit** du **modèle**
 - La sortie du **transform** du **scaler** est transmise au **predict** du **modèle**

```
scaler = MinMaxScaler()  
  
linear = LinearRegression()  
  
pipeline = Pipeline (  
    [ ('minmax', scaler),  
     ('linear', linear) ] )
```



Plus besoin de transformer les données avant, le pipeline s'en occupe

```
pipeline.fit(x_train, y_train)  
y_pred = pipeline.predict(x_test)
```