

PYTHON – PANDAS - SQL

SOMMAIRE

Sommaire	1
PANDAS - SQL	3
Pandas et SQL, première approche – principes des tables.....	3
Présentation	3
Principes des tables de données SQL	4
L'« abstrait » et le « concret » dans la table	4
Nommez et se représenter les tables.....	5
Relation entre un individu et sa table : relation « être 1 »	6
Relation entre la table et ses attributs : relation « avoir 1 ». Jamais avoir plusieurs.	7
Pandas et SQL, première approche – chargement des données	8
Notebook des premiers usages SQL de Pandas	8
Chargement d'un tableau CSV.....	8
Fichier CSV	8
Variante seaborn.....	8
Pandas et SQL, première approche – mono-table.....	9
Les opérations du SQL mono-table	9
Projection : choix des colonnes	9
Restriction : choix des lignes.....	9
Tri	9
Statistiques de bases	9
Statistiques sur des regroupement : agrégations.....	9
Insertion de lignes.....	9
Modification de lignes	9
Suppression de lignes	9
Affichage d'un tableau CSV : équivalent d'un SELECT *	10
Bases	10
Variantes : limit.....	10
Forme ou format d'un tableau : nombre de lignes et de colonnes	10
Sélection de colonnes : équivalent d'un SELECT pclass, survived, sex, age	11
Bases	11
Variante 1 : sans le .loc[]	11
Variante 2 : .drop().....	11
Supprimer les doublons : uniques(), drop_duplicates(),	12
Bases	12
Méthode drop_duplicate	12
Variante avec une colonne	12
Renommer les colonnes : rename.....	13
Bases	13
Méthode rename	13
Variantes : toutes les colonnes :	13
Ajouter une colonne – Attributs calculés : insert pandas	14
Trier.....	15
Sélectionner des lignes	15
Slicing	15
Tableau de booléens.....	15
Pandas et SQL, première approche – statistiques	16
Statistiques élémentaires	16
Cas du count()	16

Moyenne avec un WHERE	16
GROUP BY de base	17
Documentation	17
Version simple : taux de survie par sexe	17
Principes.....	17
Version simplifiée	17
Version plus complexe : taux de survie et moyenne d'âge par sexe et classe	18
reset_index()	18
Autres usages :	18
GROUP BY avec toutes sortes de statistiques : agg()	19
HAVING	20
Documentation	20
Version simple : taux de survie par sexe	20
PIVOT	21
Documentation	21
Principes.....	21
Exemple : un tableau avec :	21
Variante.....	22
Documentation	22
Pandas et SQL, première approche – Visualisation des résultats	23
Matplotlib - visualisation des résultats	23
Notebook	23
Notebook	23
Pandas et SQL, première approche – DML.....	24
DML : update – update pandas	24
SQL	24
Pandas.....	24
DML : insert – append pandas.....	25
SQL	25
Pandas.....	25
Pandas et SQL, première approche – multi-tables	26
Les opérations du SQL multi-table	26
Usages minimum	26
Exemples de code :	26
UNION = concaténation : pd.concat([df, ...])	26
SQL : UNION ensembliste	26
Pandas : pd.concat()	26
Variantes du pd.concat : on concatène les colonnes	26
Jointure naturelle : clé étrangère = clé primaire : MERGE	27
Présentation.....	27
Données de départ	27
SQL : jointure naturelle : JOIN	27
Pandas : pd.merge().....	27
Left JOIN : clé étrangère = clé primaire : MERGE + how='left'	28
SQL : jointure naturelle : LEFT JOIN	28
Pandas : pd.merge().....	28
Jointure sur la clé primaire (jointure sur une clé étrangère-primaire : table d'espèce).....	29
Données de départ	29
Pandas : pd.merge().....	29

Edition : juin 2022

PANDAS - SQL

Pandas et SQL, première approche – principes des tables

https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_sql.html

Présentation

- Pandas permet de faire des opérations équivalentes à celles qu'on fait sur les tables des bases de données relationnelles en SQL.
- On va donc commencer par rappeler les principes des tables de données SQL.
- Puis rappeler les opérations qu'on peut faire en SQL.
- Ensuite, on verra comment faire ça avec Pandas

Principes des tables de données SQL

L'« abstrait » et le « concret » dans la table

- Dans une table, il faut distinguer entre :
 - ⇒ la première ligne qui définit les caractéristiques de chaque élément de la table (ici des articles)
 - ⇒ les lignes suivantes qui définissent les éléments un par un.
- La **première ligne** est une description **abstraite** d'un article.
- Les **lignes suivantes** sont les descriptions **concrètes** de chaque article, un par un.

Première ligne : abstraites

ARTICLES

id	nom	desc	Prix
a1	n1	d1	10
a2	n2	d2	15
a3	n3	d3	5
a4	n4	d4	10

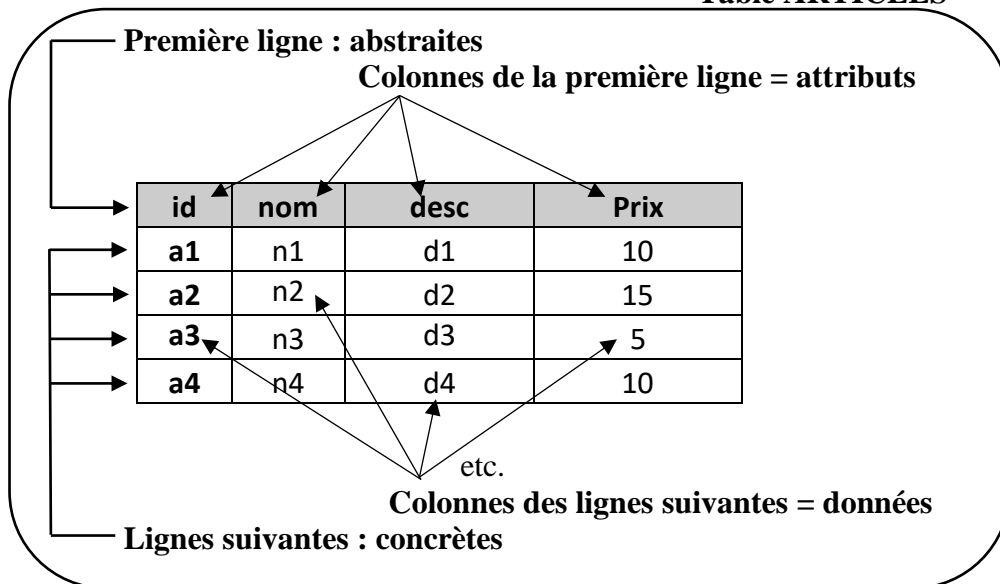
Lignes suivantes : concrètes

- La modélisation consiste à **trouver toutes les premières lignes de toutes les tables** qui vont permettre de ranger concrètement toutes les informations qu'on va manipuler.
- Pour ne pas se perdre dans l'abstraction, il faut **toujours imaginer des données concrètes à mettre dans nos tables**. C'est la **première clé d'une modélisation réussie**.

Nommez et se représenter les tables

- La table est le cœur de l'organisation des données dans les bases de données relationnelle.
 - ⇒ Une **table** est un **ensemble** regroupant des individus, sans ordre particulier.
 - ⇒ La **première ligne de la table** correspond à sa description générale, sa définition.
 - ⇒ Chaque **ligne suivante** dans la table est un **individu**.
 - ⇒ Chaque **colonne de la première ligne** est une **propriété** de la table qu'on appelle aussi **attribut**.
 - ⇒ Chaque **colonne des lignes suivantes** est une **donnée**.

Table ARTICLES = ensemble des Articles



a1
a2
a3
A4

Relation entre un individu et sa table : relation « être 1 »

- **La relation entre un individu et sa table est une relation « est 1 ».**
- Par exemple :
 - ⇒ L'individu identifié par la clé primaire a1 « est 1 » ARTICLE.
 - ⇒ L'individu identifié par la clé d'usage n2 « est 1 » ARTICLE.
- Pour chaque ligne de la table, on peut dire : cette ligne « est 1 » ARTICLE.

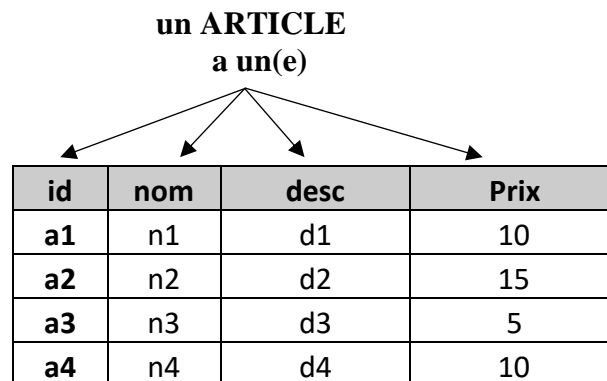
id	nom	desc	Prix
a1	n1	d1	10
a2	n2	d2	15
a3	n3	d3	5
a4	n4	d4	10

Relation entre la table et ses attributs : relation « avoir 1 ». Jamais avoir plusieurs.

- **La relation entre la table et un attribut est obligatoirement une relation « a 1 ».**

Par exemple :

- Un article « a 1 » id
 - Un article « a 1 » nom
 - Un article « a 1 » description
 - Un article « a 1 » prix
-
- C'est vrai aussi pour chaque individu concret :
 - L'article « a1 » : a un nom, a une description, a une adresse.
 - L'article identifié par son nom « n2 » : a un id, a une description, a un prix.
 - etc.



- **La relation d'un attribut a sa table n'est jamais « avoir plusieurs » :**

Par exemple :

- Si un article a plusieurs photos, photos ne pourra pas être un attribut de la table ARTICLE
⇒ Il faudra créer une table ARTICLE.

Notebook des premiers usages SQL de Pandas

- On mettra nos exemples dans des notebooks.
- Tous les codes de ce chapitre sont dans un notebook dont l'image HTML est ici : <http://bliaudet.free.fr/notebook/NoteBook-Pandas-0.html>

Chargement d'un tableau CSV

Fichier CSV

- On peut récupérer le fichier titanic sur github : <https://github.com/mwaskom/seaborn-data>
⇒ Le Titanic, le « hello world » de la data-analyse en python !
- Le fichier csv directement -> [ici](#). Il faut le copier-coller et l'enregistrer dans titanic.csv par défaut dans le dossier de démarrage du serveur jupyter.
- C'est un tableau d'environ 800 lignes.
⇒ On le charge avec la méthode read_csv()

```
import pandas as pd

df = pd.read_csv('titanic.csv') # df c'est la table excel
                                # dataframe pandas
```

Variante seaborn

- Avec seaborn et la fonction [load_dataset\(\)](#), on récupère directement des données de [github](#).
- Il faut faire une installation seaborn :

```
pip3 install seaborn
```

- Ici, on va récupérer le fichier titanic

```
import pandas as pd. # ça importe le numpy
import seaborn as sns

df_ti = sns.load_dataset('titanic') # csv de gitub
```

- On peut aussi filtrer les colonnes directement :

```
df_ti = sns.load_dataset('titanic') [['survived', 'sex',
'pclass']]
```


Les opérations du SQL mono-table

Projection : choix des colonnes

- 1 : Projection « primaire » : avec l'identifiant ou l'équivalent : SELECT
- 2 : Projection avec élimination de doublon : DISTINCT
- 3 : Renommer les attributs
- 4 : Fabriquer de nouveau attributs : les attributs calculés

Restriction : choix des lignes

- 5 : Restriction simple : WHERE
- 6 : Restriction avec imbrication de requêtes : IN et NOT IN

Tri

- 7 : ORDER BY

Statistiques de bases

- 8 : count, min, max, avg

Statistiques sur des regroupement : agrégations

- 9 : Agrégations de base : GROUP BY
- 10 : Restriction sur les agrégations : HAVING

Insertion de lignes

- 11 : INSERT

Modification de lignes

- 12 : UPDATE

Suppression de lignes

- 13 : DELETE

Il faudra pouvoir faire toutes ces opérations en pandas.

Affichage d'un tableau CSV : équivalent d'un SELECT *

Bases

- On veut tout projeter
- SELECT SQL :

```
SELECT *  
FROM df
```

- Code Pandas :

```
df
```

Variantes : limit

- SELECT SQL :

```
SELECT *  
FROM df  
LIMIT 5
```

- Code Pandas :

```
df.head()    # les 5 premières lignes  
df.head(10)  # les 10 premières lignes  
  
df.tail()    # les 5 dernières lignes  
df           # les 5 premières et dernières lignes
```

Forme ou format d'un tableau : nombre de lignes et de colonnes

```
df.shape    # (891, 15)
```

```
df.index     # le nom des index de ligne
```

Ca peut être un range (index numéroté)
Ou une séquence non mutable

```
df.columns   # le nom des index de colonne
```

C'est un index : une séquence non mutable

```
df.axes      # le nom des index de ligne et de colonnes
```

```
df.dtypes    # les types de toutes les colonnes
```

Sélection de colonnes : équivalent d'un SELECT pclass, survived, sex, age

Bases

- On veut projeter uniquement les colonnes pclass, survived, sex et age.
- SELECT SQL :

```
SELECT pclass, survived, sex, age  
FROM df
```

- Code Pandas :

```
df.loc[ : , ['pclass', 'survived', 'sex', 'age']]
```

- Pour créer un nouveau tableau avec des colonnes au choix :

```
df2=df[['pclass', 'survived', 'sex', 'age']]  
  
df2.columns
```

Variante 1 : sans le .loc[]

- A éviter : on utilisera toujours le « loc »

```
df[['pclass', 'survived', 'sex', 'age']]
```

⇒ Notez les doubles crochets.

Variante 2 : .drop()

- Pour créer un nouveau tableau avec des colonnes en moins :

```
df.drop(  
    [  
        'sibsp', 'parch', 'fare', 'embarked', 'class', 'who',  
        'adult_male', 'deck', 'embark_town', 'alive', 'alone'  
    ],  
    axis=1  
)  
  
df2.columns
```

⇒ On peut passer le code entre parenthèses ou crochet à la ligne : c'est plus lisible

Supprimer les doublons : `unique()`, `drop_duplicates()`,

Bases

- On veut projeter uniquement des colonnes sans doublons.
- SELECT SQL :

```
SELECT DISTINCT col1, col2  
FROM df
```

- Code Pandas :

```
res = df.loc[:, ['col1', 'col2']].drop_duplicates()
```

- ⇒ Ca donne une Series Pandas
- ⇒ On peut faire `list(res)`

Méthode `drop_duplicates`

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

Variante avec une colonne

```
df.loc[:, ['col1']].drop_duplicates()
```

```
df.col1.unique ()
```

- ⇒ Ca donne une ndarray numpy
- ⇒ On peut faire `list(res)`

Renommer les colonnes : rename

Bases

- On veut renommer une colonne
- SELECT SQL :

```
SELECT pclass AS classe, survived AS survivant  
FROM df
```

- Code pandas :

```
df.rename(columns={'pclass': 'classe', 'survived': 'survivant'})
```

Méthode rename

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rename.html>

Variantes : toutes les colonnes :

```
df3=df.loc[:, ['pclass', 'survived', 'age']]
```

```
df3.columns=['classe', 'survivant', 'age']
```

Ajouter une colonne – Attributs calculés : insert pandas

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.insert.html>

```
df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
df
```

```
col1 col2
```

```
0    1    3
```

```
1    2    4
```

```
df.insert(1, "newcol", [99, 99])
df
```

```
col1 newcol col2
```

```
0    1    99    3
```

```
1    2    99    4
```

A partir de là, on pourra créer des attributs calculés dans nos DF

Trier

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

```
df.age.sort_values()

df.age.sort_values(ascending=False)

df.loc[:, ['pclass', 'age']].sort_values(by=['pclass', 'age'])

df.pclass.drop_duplicates().sort_values()
```

Sélectionner des lignes

Slicing

```
df[4:10]
```

Tableau de booléens

```
print( df.pclass==3 )
print( df[df.pclass==3] )
print( df[df.age > 30] )
print( df[ ( df.pclass==3 ) & ( df.age > 30 ) ] ) # et : &      ou : |
```

Version loc, à privilégier ! C'est beaucoup plus clair !

```
# Version loc
print( df.loc[:, 'pclass'] == 3 )
print( df.loc[
    df.loc[:, 'pclass'] == 3
])
print( df.loc[:, 'age'] > 30 )
print( df.loc[
    ( df.loc[:, 'pclass'] == 3 ) & ( df.loc[:, 'age'] > 30 )
])
```

Pandas et SQL, première approche – statistiques

Statistiques élémentaires

```
df.describe()  
df.describe(include="all")
```

- SQL :

```
SELECT avg(age) age_moyen  
FROM df
```

- Pandas :

```
df.attribut.mean()  
# mean(), min(), max(), count()
```

Cas du count()

- Count() compte les NaN

⇒ On utilise dropna() pour supprimer les valeurs à NaN

```
df.attribut.dropna().count() # ne compte pas les NaN
```

Moyenne avec un WHERE

```
import seaborn as sns  
  
df_ti = sns.load_dataset('titanic')[['survived', 'sex',  
    'pclass']]  
df_ti  
  
df_ti.loc[df_ti.sex=='male', 'survived'].mean()  
  
df_ti.loc [df_ti.sex=='female', 'survived'].mean()
```

⇒ Un GROUP BY serait pratique !

GROUP BY de base

Documentation

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html>

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

Version simple : taux de survie par sexe

- SELECT SQL :

```
SELECT sex, avg(survived) as taux_survie
FROM df
GROUP BY sex
```

- Code pandas :

```
res= df_ti.loc[
    :,
    ['sex', 'survived']
].groupby(['sex']) \
    .mean()

res
```

⇒ Le \ permet de passer à la ligne : c'est plus lisible

- Résultats :

```
      survived
sex
female  0.742038
male    0.188908
```

Principes

Le 'sex', c'est l'index.

Version simplifiée

- Code pandas :

```
# df à 3 attributs : ['survived', 'sex', 'pclass']

res = df_ti.groupby(['sex']).mean()

res
```

⇒ Ici, le group by s'applique automatiquement à ce qui est possible

Version plus complexe : taux de survie et moyenne d'âge par sexe et classe

- On veut calculer le taux de survie des passagers par sexe et par classe
- SELECT SQL :

```
SELECT sex, pclass, avg(survived) as taux_survie, avg(age)
age_moyen
FROM df
GROUP BY sex, pclass
```

- Code pandas :

```
res=df.loc[
    :,
    ['sex', 'pclass', 'survived', 'age']
].groupby(['sex', 'pclass'])\
    .mean()
res
```

⇒ Le \ permet de passer le code à la ligne, on peut mettre des indentations : c'est plus lisible

- Résultats :

		survived	age
sex	pclass		
female	1	0.968085	34.611765
	2	0.921053	28.722973
	3	0.500000	21.750000
male	1	0.368852	21.750000
	2	0.157407	21.750000
	3	0.135447	26.507589

reset_index()

'sex' et 'pclass' : c'est l'index. C'est un index multiple : ce n'est pas pratique. On remet l'index en colonne avec reset_index

```
res=df.loc[
    :,
    ['sex', 'pclass', 'survived', 'age']
].groupby(['sex', 'pclass'])\
    .mean()\
    .reset_index()
res
```

- Résultats :

	sex	pclass	survived	age
0	female	1	0.968085	34.611765
1	female	2	0.921053	28.722973
2	female	3	0.500000	21.750000
3	male	1	0.368852	41.281386
4	male	2	0.157407	30.740707
5	male	3	0.135447	26.507589

Autres usages :

- Pour avoir la moyenne par sex et classe de tous les attributs numériques :

```
df.groupby(['sex', 'pclass']).mean().reset_index()
```

GROUP BY avec toutes sortes de statistiques : agg()

- SELECT SQL :

```
SELECT sex, pclass,
       avg(survived) as taux_survie,
       avg(age) age_moyen, min(age) age_min, max(age) age_max
FROM df
GROUP BY sex, pclass
```

- Code pandas :

```
df_ti = sns.load_dataset('titanic')
print(df_ti.shape)
res= df_ti.loc[
    :,
    ['sex', 'pclass', 'survived', 'age']
].groupby(['sex', 'pclass'])\
    .agg({
        'survived': ['mean'],
        'age': ['mean', 'min', 'max']
    })\
    .reset_index()

print(res.round(2)) # round(2) s'applique à tous les float
```

- ⇒ Au lieu d'appliquer la méthode de statistique après le groupby, on applique la méthode agg() qui va nous permettre de passer plusieurs méthodes statistiques.
- ⇒ On passe un dictionnaire dans la fonction agg()
- ⇒ Round(2) permet de formater les float
- ⇒ Le \ permet de passer à la ligne : c'est plus lisible

- Résultats :

	sex	pclass	survived	age		
			mean	mean	min	max
0	female	1	0.97	34.61	2.00	63.0
1	female	2	0.92	28.72	2.00	57.0
2	female	3	0.50	21.75	0.75	63.0
3	male	1	0.37	41.28	0.92	80.0
4	male	2	0.16	30.74	0.67	70.0
5	male	3	0.14	26.51	0.42	74.0
5	male	3	0.14	26.51	0.42	74.0

HAVING

Documentation

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html>

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

Version simple : taux de survie par sexe

- SELECT SQL :

```
SELECT sex, avg(survived) as taux_survie
FROM df
GROUP BY sex
HAVING taux_survie>0.5
```

- Code pandas :

```
res=(
    df.loc[:, ['sex', 'survived']]
    .groupby(['sex']).mean()
)
res.loc[res.survived > 0.5, :]
```

⇒ Les () autour de la formule permettent de passer le code à la ligne, on peut mettre des indentations : c'est plus lisible

- Résultats :

```
      survived
sex
female  0.742038
```

PIVOT

Documentation

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

Principes

- La fonction `pivot_table()` permet de produire un tableau croisé (qu'on appelle pivot).
- Le pivot n'est pas standard en SQL : on le trouve sous ORACLE et SQL_SERVER mais pas sur MySQL.
- C'est une opération très puissante. Il faudra creuser la documentation pour en voir d'autres aspects.

Exemple : un tableau avec :

- ⇒ en lignes les classes
- ⇒ en colonnes le sexe
- ⇒ en données les taux de survie.

	survived	
	female	male
pclass		
1	0.968085	0.368852
2	0.921053	0.157407
3	0.500000	0.135447

- ⇒ 96% des femmes de 1^{ère} classe ont survécu,
- ⇒ 13% des hommes de 3^{ème} classe ont survécu,
- ⇒ etc.

- Code pandas :

```
res=df.loc[
    :,
    ['pclass', 'survived', 'sex']
].pivot_table(index = 'pclass', columns = 'sex')
res
```

- ⇒ Le \ permet de passer le code à la ligne, on peut mettre des indentations : c'est plus lisible
C'est utile pour séparer les « .appelDeFonction() ».

Variante

- Ici on précise l'attribut de statistique (survived) et la fonction de statistique (mean).
- Code pandas :

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('titanic') # csv de gitub
df.columns
df.head()
res = df.pivot_table(
    'survived',
    aggfunc=np.mean,
    index='class',
    columns='sex'
)
res
```

⇒ On peut passer le code entre parenthèses ou crochets à la ligne : c'est plus lisible

⇒ Mean est le calcul par défaut : on peut ne pas le préciser :

```
titanic.pivot_table('survived', index='class', columns='sex')
```

- Résultats :

sex	female	male
class		
First	0.968085	0.368852
Second	0.921053	0.157407
Third	0.500000	0.135447

- Avec un sum, on obtient :

sex	female	male
class		
First	91	45
Second	70	17
Third	72	47

Documentation

Le pivot est un outil puissant.

Documentation officielle :

<http://pandas.pydata.org/pandas-docs/stable/reshaping.html>

Exemples :

<https://github.com/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/03.09-Pivot-Tables.ipynb>

Matplotlib - visualisation des résultats

Notebook

- On mettra nos exemples dans des notebooks.
- Tous les codes de ce chapitre sont dans un notebook dont l'image HTML est ici : <http://bliaudet.free.fr/notebook/NoteBook-Matplotlib-0.html>

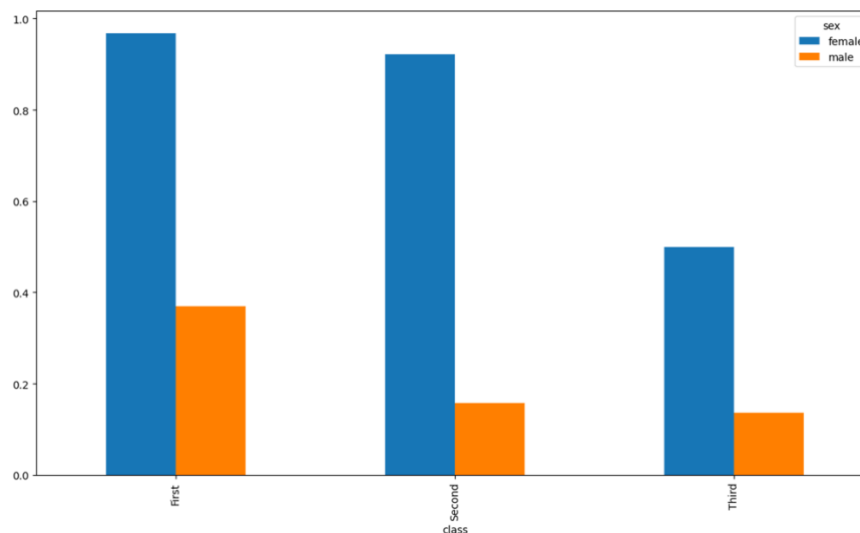
Notebook

- Pour avoir un graphique, on utilise la fonction « plot » de la bibliothèque « matplotlib ».
- Le code est le suivant, appliqué au tableau « res » précédemment calculé, par pivot ou par group by.
- Code :

```
import matplotlib.pyplot as plt

res.plot(kind='bar') # pour avoir des barres
plt.show()
```

- Résultat :



- Variante :
⇒ On peut sommer le nombre de survivant par sexe et par classe plutôt que calculer un pourcentage.

Pandas et SQL, première approche – DML

DML : update – update pandas

SQL

2 updates :

```
UPDATE df
SET B = d
WHERE id = 1
```

```
UPDATE df
SET B = e
WHERE id = 2
```

Pandas

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.update.html>

```
df = pd.DataFrame({'A': ['a', 'b', 'c'], 'B': ['x', 'y', 'z']})
new_df = pd.DataFrame({'B': ['d', 'e']}, index=[1, 2])
df.update(new_df)
```

```
Entrée [58]: df = pd.DataFrame({'A': ['a', 'b', 'c'],
                                'B': ['x', 'y', 'z']})
df
```

Out[58]:

	A	B
0	a	x
1	b	y
2	c	z

```
Entrée [59]: new_df = pd.DataFrame({'B': ['d', 'e']}, index=[1, 2])
new_df
```

Out[59]:

	B
1	d
2	e

```
Entrée [60]: df.update(new_df)
df
```

Out[60]:

	A	B
0	a	x
1	b	d
2	c	e

DML : insert – append pandas

SQL

```
INSERT INTO...
```

Pandas

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.append.html>

```
df = pd.DataFrame([[1, 2], [3, 4]],  
                  columns=list('AB'), index=['x', 'y'])  
  
df2 = pd.DataFrame([[5, 6], [7, 8]],  
                   columns=list('AB'), index=['x', 'y'])  
  
df.append(df2)
```

Pandas et SQL, première approche – multi-tables

Les opérations du SQL multi-table

Usages minimum

1. L'union de 2 tables (intersection et différence peuvent se traiter avec des requêtes imbriquées)
2. La jointure naturelle : jointure entre une clé étrangère et la clé primaire correspondante
3. Les requêtes imbriquées minimum : IN et not IN
4. Left join : pour rajouter, après jointure naturelles, les lignes dont la clé étrangère valait NULL.

Avec ces usages, on peut traiter toutes les requêtes SQL.

Ce sont des notions conceptuellement difficiles mais qui permettent de traiter des calculs complexes en un ou peu d lignes, et facilement quand on a l'habitude (ou avec des modèles).

Exemples de code :

UNION = concaténation : `pd.concat([df, ...])`

SQL : UNION ensembliste

Table df1(x, y, z)

Table df2(x, y, z)

```
df1
UNION
df2
```

Pandas : `pd.concat()`

Usage de base : on concatène les lignes

<https://pandas.pydata.org/docs/reference/api/pandas.concat.html>

```
df = pd.concat([df1, df2])
```

Variantes du `pd.concat` : on concatène les colonnes

En précisant `axis=1` : on concatène les colonnes. Ce n'est plus une UNION ensembliste

```
df = pd.concat([df1, df2], axis=1)
```

Jointure naturelle : clé étrangère = clé primaire : MERGE

Présentation

Le MERGE est l'équivalent du JOIN SQL. Le JOIN existe aussi en Pandas, mais est plus spécifique. C'est pourquoi on préfère utiliser tout le temps le MERGE.

Données de départ

```
personnel  date embauche  groupe
0         Bob          2015    R&D
1        Lisa          2018    R&D
2         Sue          2020   None
3         Bob          2020    RH
groupe      ville
0    R&D    Paris
1    RH   Nanterre
```

A noter qu'on n'a pas besoin de clé primaire dans la table principale pour faire le JOIN

On crée deux tables-dataframes :

```
df_personnel = pd.DataFrame({
    'personnel': ['Bob', 'Lisa', 'Sue', 'Bob'],
    'date embauche': [2015, 2018, 2020, 2020],
    'groupe': ['R&D', 'R&D', None, 'RH'],
})
df_groupe = pd.DataFrame({
    'groupe': ['R&D', 'RH'],
    'ville': ['Paris', 'Nanterre'],
})

print(df_personnel)
print(df_groupe)
```

SQL : jointure naturelle : JOIN

```
SELECT p.*, g.ville
FROM df_personnel p
JOIN df_groupe g ON g.groupe=p.groupe
```

Pandas : pd.merge()

<https://pandas.pydata.org/docs/reference/api/pandas.merge.html>

```
df = pd.merge(df_personnel, df_groupe) # pas de crochets
```

```
personnel  date embauche  groupe  ville
0         Bob          2015    R&D    Paris
1        Lisa          2018    R&D    Paris
2         Bob          2020    RH   Nanterre
```

Left JOIN : clé étrangère = clé primaire : MERGE + how='left'

SQL : jointure naturelle : LEFT JOIN

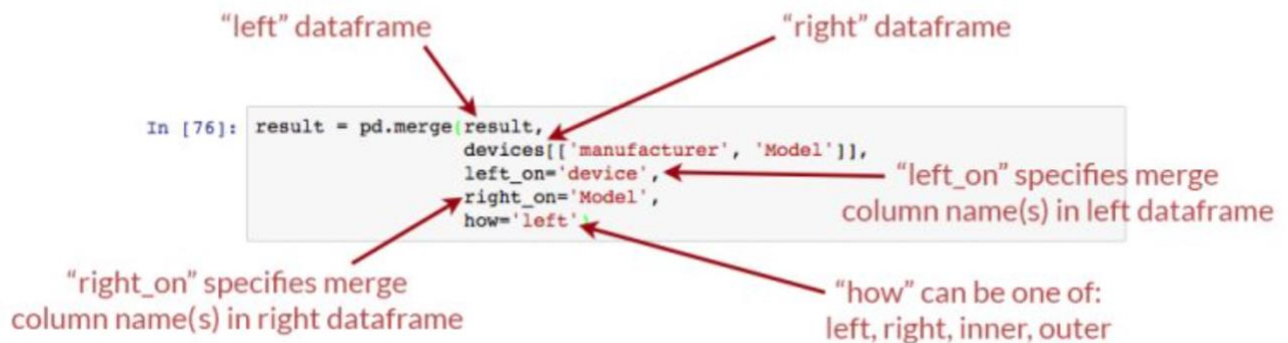
```
SELECT p.*, g.ville
FROM df_personnel p
LEFT JOIN df_groupe g ON g.groupe=p.groupe
```

Pandas : pd.merge()

```
pd.merge(df_personnel, df_groupe, how='left')
```

	personnel	date embauche	groupe	ville
0	Bob	2015	R&D	Paris
1	Lisa	2018	R&D	Paris
2	Sue	2020	None	NaN
3	Bob	2020	RH	Nanterre

How : left, right, inner, outer



Jointure sur la clé primaire (jointure sur une clé étrangère-primaire : table d'espèce)

Données de départ

```
personnel  groupe
0         Bob    SAF
1         Lisa   R&D
2         Sue    RH
personnel  date embauche
0         Lisa      2004
1         Bob      2008
2         Sue      2014
```

Pandas : `pd.merge()`

```
pd.merge(df1, df2)
```

```
personnel  groupe  date embauche
0         Bob    SAF      2008
1         Lisa   R&D      2004
2         Sue    RH      2014
```

Ca rajoute les colonnes proprement autour de personnel qui fait office d'id