

# TP Pandas – Première approche – Outils et méthode

On peut toujours s'aider de ces exemples de cours :

[http://bliaudet.free.fr/notebook/pandas\\_cours\\_1.html](http://bliaudet.free.fr/notebook/pandas_cours_1.html)

## Chargement – Analyse basique – Compréhension - Enregistrement – CSV et JSON

### Étape 1 : chargement et description

On charge un fichier .csv : le fichier data1.csv

On regarde le .csv dans VS Code, dans Excel

On le charge : on récupère un df : dataframe

On affiche la taille du df : .shape

On affiche les premières et les dernières lignes : df.info

Combien y a-t-il de colonnes d'après .shape ? Combien y a-t-il de colonnes d'après .info. Pourquoi ?

Tester d'autres outils : .head(), .head(10, .column, .info(), .describe())

Il faut bien comprendre le describe()

### Étape 2 : enregistrement de CSV à « ; » et de JSON

#### Pratique

On enregistre le df dans un fichier data1\_pv.csv en format CSV avec point-virgules :

```
df.to_csv('data1_pv.csv', sep=';', index=False)
```

On regarde le .csv dans VS Code, dans Excel

Dans excel, on efface le fare et le survived du premier individu.

On enregistre le df dans un fichier data1.json en format JSON :

```
to_json("data1.json", orient='records')
```

On regarde le .json dans VS Code et dans un navigateur : charger une extension « json formatter » pour améliorer l'affichage dans le navigateur.

### Étape 3 : nommer sa table, restructurer sa table

#### Théorie

L'objectif est de nommer sa table, son fichier, son df et de mettre les colonnes dans un ordre plus pratique.

Ce n'est pas nécessaire mais c'est pratique : l'important est seulement de bien savoir ce que veulent dire nos données.

Principes des tables Excel de BD ou pour pandas : <http://bliaudet.free.fr/IMG/pdf/02-Pandas-SQL.pdf#page=3>

#### Pratique - 1

Notre table est une table de : passagers\_du\_titanic.

Notre fichier peut s'appeler : passagers\_du\_titanic.csv

Notre df peut s'appeler : df\_passagers\_du\_titanic.

«Turja » est le nom d'un passager\_du\_titanic

On a avec les df, une colonne d'index de 0 à 1044. Cet index, c'est l'identifiant : l'équivalent de l'id (la clé primaire en SQL). Le name est la « clé d'usage » : l'attribut qui sert d'identifiant dans la vie quotidienne (même si il peut y avoir des doublons = des homonymes). On pourrait mettre le « name » en première colonne, après l'index : ce serait plus clair.

On peut écrire :

```
df_passagers_du_titanic = df[
    ['name', 'sex', 'age', 'pclass', 'fare', 'survived']
]
```

## Pratique - 2

L'attribut name contient 3 informations : on peut les séparer en 3 colonnes distinctes :

```
df['lastname'] =
df['name'].str.split(',').str[0].str.strip()
```

```
df['titre'] = df['name'].str.split(' ').str[1].str[:-1])
```

```
df['firstnames'] =
df['name'].str.split('.').str[1].str.strip()
```

On restructure notre df :

```
df_passagers_du_titanic = df [
    ['titre', 'lastname', 'firstnames',
    'sex', 'age', 'pclass', 'fare', 'survived']
]
```

L'ordre c'est :  
d'abord l'index,  
ensuite le nom (ici : titre, lastname, firstnames),  
ensuite les infos de la personne : sex et age ,  
ensuite les infos du voyage : pclass et fare,  
ensuite l'info « a-t-il survécu au voyage ».

## Étape 4 : trier les df pour mieux les comprendre

### Principes

Principe général :

```
df.sort_values(
    by=['colonne1', 'colonne2'],
    ascending=[True, False]
)
```

Au minimum :

```
df.sort_values(by="survived")
```

Enchainement de méthodes :

```
df.sort_values(by="fare").info
```

### Pratique

Testez des tris : ce n'est pas pratique. Ca rejoint une information qu'on a déjà avec `.describe()`

## Nettoyer ses données

### Principes

Pour faire des statistiques, il faut éviter qu'il y ait des valeurs non renseignées (on dit null) dans le df.

`.dropna()` permet de supprimer les lignes qui contiennent des valeurs non renseignées

`.dropna(axis=1)` permet de supprimer les colonnes qui contiennent des valeurs non renseignées

### Pratique

On regarde le fichier `data1_pv.csv` dans Excel

Dans excel, on efface le fare et le survived du premier individu et on enregistre.

On charge le fichier `data1.csv` : on récupère un df

On affiche la taille du df : `.shape`

On crée un `df_dropna_row` et un `df_dropna_col`

On affiche le shape de ces 2 df. Que constatez-vous ?

On charge le fichier `data1_pv.csv` : on récupère un `df_pv`

On affiche la taille du `df_pv` : `.shape`

On crée un `df_pv_dropna_row` et un `df_pv_dropna_col`

On affiche le shape de ces 2 df. Que constatez-vous ?

## Statistiques basiques de chaque attribut

### Principes

```
df[« fare »].min()
.min(), .max(), .mean(), .count()
```

On a déjà tout ça dans `describe()`

Ce sera utile si on ne sélectionne que certaines lignes.

## Sélection de colonnes

### Principes

```
df_subset = df[['age', 'survived']]
```

### Pratique

Sélectionnez les colonnes age, survived et fare puis faites un `describe()`

Quelle est la valeur moyenne de l'âge des passagers ?

## Sélection de lignes : df.loc à préférer sur df[ df[

### Principes

```
lignes_1_et_3 = df.loc[[1, 3]]  
lignes_1_a_3_inclus = df.loc[1:3]
```

```
lignes_age_sup_30 = df.loc[df['age'] > 30]  
lignes_age_sup_30 = df[ df['age'] > 30]
```

```
df_male = df[ df['sex'] == 'male']  
df_male = df.loc[df['sex'] == 'male']
```

## Sélection de lignes et de colonne : df.loc

### Principes

```
selection = df.loc[df['age'] > 30, ['name', 'pclass']]  
# on filtre et on ne garde que 2 colonnes
```

```
selection = df[ [df['age'] > 30, ['name', 'city']] => bug !
```

## Accès à une cellule : df.loc et df.iloc (index loc)

### Principes

```
df.loc[0, 'age'] # vaut l'age de l'individu d'index 0  
df.loc[10, 'age'] # vaut l'age de l'individu d'index 10  
df.iloc[0]['age'] # vaut l'age de l'individu d'index 0  
df.loc[10]['age'] # vaut l'age de l'individu d'index 10  
  
df.loc[0] # c'est l'individu d'index 0  
df.iloc[0] # c'est l'individu d'index 0
```

## Group by

### Principes

- Quand on veut la moyenne des âges par sexe, on fait un regroupement par sexe.
- On peut aussi vouloir la moyenne des âges par classe et par sexe : on fait un regroupement par classe et par sexe.
- Le critère de regroupement devient l'identifiant de la table de résultats.
- Les autres colonnes sont les statistiques associées : dans notre exemple la moyenne des âges.
- « A la main », sous Excel : on trie selon les colonnes de regroupement, puis on fait la moyenne des âges pour des valeurs identiques de regroupement (mâle ou mâle de 2<sup>ème</sup> classe, etc.)

### Écritures pandas :

#### Nombre total de passagers par sexe et par classe

```
sex_class_counts = df.groupby(
    ['sex', 'pclass']
).size()

print(sex_class_counts)
```

#### Moyenne des âges par sexe et par classe

```
sex_class_ages_mean = df.groupby(
    ['sex', 'pclass']
)['age'].mean()

print(sex_class_ages_mean)
```

#### Moyenne des survivants par sexe et par classe

Version 1 :

```
sex_class_surviveds_mean_v1 = df.groupby(
    ['sex', 'pclass']
)['survived'].mean()

print(sex_class_surviveds_mean_v1)
```

Version 2 :

```
sex_class_surviveds_mean_v2 = df.groupby(
    ['sex', 'pclass']
).agg(
    survived_mean=('survived', 'mean')
)

print(sex_class_surviveds_mean_v2)
```

Par sexe et par classe : nombre de survivants, de non survivants, taux de survivants et taux de non survivants :

```
resultats = df.groupby(
    ['sex', 'pclass']
).agg(
    nb_survivants= ('survived', 'sum'),
    nb_non_survivants= (
        'survived',
        lambda x: (x == 0).sum()
    ),
    taux_survivants= (
        'survived',
        lambda x: round(x.sum() / len(x) * 100, 2)
    ),
    taux_non_survivants=(
        'survived',
        lambda x: round((x == 0).sum() / len(x) * 100, 2)
    )
)
print(resultats)
```

**A noter :**

```
    taux_survivants= (
        'survived',
        lambda x: round(x.sum() / len(x) * 100, 2)
    ),
```

Veut dire qu'on crée une colonne pour un couple sex-class : la colonne taux\_survivant.

Elle est calculée avec survived

Pour tous les x de survived : on fait la somme (sum) qu'on divise par le nombre d'éléments (len), le tout \* 100 et arrondi à 1 chiffres après la virgule.

## Suite du TP

1)

Sélectionnez les colonnes age, survived et fare puis affichez les statistiques descriptives (mean, min, max) pour ces colonnes.

Quelle est la valeur moyenne de l'âge des passagers ?

Visualisez la valeur de survie selon l'âge et la classe :

```
plt.scatter(df['age'], df['pclass'], c=df['survived'], s=4)
plt.xlabel('âge')
plt.ylabel('classe')
plt.title('âge vs classe coloré par survie')
plt.show()
```

2)

Filtrez les passagers ayant plus de 30 ans et appartenant à la première classe, puis affichez leurs informations.

### 3) Comparaison entre hommes et femmes

Visualisez la valeur de survie selon l'âge et la classe pour les hommes.

Visualisez la valeur de survie selon l'âge et la classe pour les femmes.

Mettez les deux graphiques dans une même fenêtre

4) Calculez le nombre total de passagers pour chaque sexe et chaque classe.

5. Calculez la survie moyenne par sexe et classe

6) Parmi les femmes de la première classe, quel est le pourcentage de survie ?

### 7) Analyse des tarifs

- Sélectionnez les passagers ayant payé un tarif inférieur à la moyenne :
- Représentez par un nuage de points l'âge en fonction de la classe pour ces passagers.
- Affichez un histogramme des tarifs sur 10 barres : `plt.hist(df['fare'], bins=10)`
- Affichez la répartition des passagers ayant payé moins de 50 unités monétaires en fonction de leur classe.