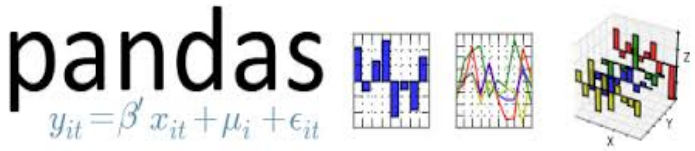
ENSAI - École nationale de la statistique et
de l'analyse de l'information, Rennes

Pandas en 10 minutes

Farah AIT SALAHT
farah.ait-salaht@ensai.fr

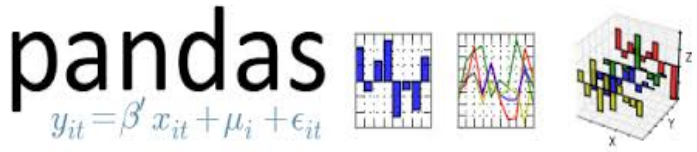
janvier 2017

Pandas



- Python Data Analysis Library, similar to:
  - R
  - MATLAB
  - SAS
- Combined with the IPython toolkit
- Built on top of NumPy, SciPy, to some extent matplotlib
- **Pa**nel **Da**ta **S**ystem
- Open source, BSD-licensed
- Key Components
  - Series
  - DataFrame

Pandas



La richesse des fonctionnalités de la librairie pandas est une des raisons, si ce n'est la principale, d'utiliser Python pour extraire, préparer, éventuellement analyser, des données.

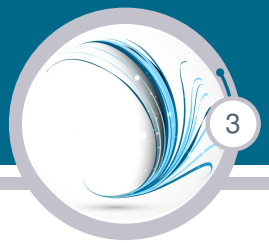
Objet : les classes Series et DataFrame ou table de données.

Lire, écrire : création et exportation de tables de données à partir de fichiers textes (séparateurs, .csv, format fixe, compressés), HTML, XML, SQL...

Gestion d'une table : sélection des lignes, colonnes, transformations, réorganisation par niveau d'un facteur, exclusion ou imputation élémentaire de données manquantes...

Statistiques élémentaires, bivariées, tri à plat (nombre de valeurs nulles, valeurs manquantes...), graphiques, statistiques par groupe...

Structures de données : Series



- La classe Series est l'association de deux tableaux unidimensionnels. Le premier est un ensemble de valeurs indexées par le 2ème qui est souvent une série temporelle.
- Plusieures façons de construire une série

```
In [ ]: import pandas as pd
In [ ]: s=pd.Series(list('abcdef'))
In [ ]: s
Out[ ]:
0    a
1    b
2    c
3    d
4    e
5    f

In [ ]: s=pd.Series([2,4,6,8])

In [ ]: s
Out[ ]:
0    2
1    4
2    6
3    8
```

Series - Travailler avec les indices

- Les index de série peuvent être spécifiés
- Les valeurs individuelles peuvent être sélectionnées par index
- Plusieurs valeurs peuvent être sélectionnées avec plusieurs index

```
In [ ]: s=pd.Series([2, 4, 6, 8],index=['f', 'a', 'c', 'e'])
In [ ]: s
Out[ ]:
f    2
a    4
c    6
e    8

In [ ]: s['a']
Out[ ]: 4

In [ ]: s[['a', 'c']]
Out[ ]:
a    4
c    6
```

Opérations

- Filtrage
- Opérations sur les données avec Numpy

```
In [ ]: s
Out[ ]:
f     2
a     4
c     6
e     8

In [ ]: s[s>4]
Out[ ]:
c     6
e     8

In [ ]: s>4
Out[ ]:
f    False
a    False
c     True
e     True

In [ ]: s*2
Out[ ]:
f     4
a     8
c    12
e    16
```

Séries - Données incomplètes

- Pandas s'accommode avec les données incomplètes

```
In [ ]: sdata={'b':100, 'c':150, 'd':200}
In [ ]: s=pd.Series(sdata)
In [ ]: s
Out[ ]:
b     100
c     150
d     200

In [ ]: s=pd.Series(sdata,list('abcd'))
In [ ]: s
Out[ ]:
a       NaN
b     100.0
c     150.0
d     200.0

In [ ]: s*2
Out[ ]:
a       NaN
b     200.0
c     300.0
d     400.0
```

Séries - alignement automatique

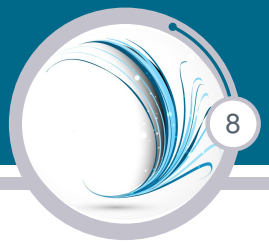
- Les données sont automatiquement alignées

```
In [ ]: s2=pd.Series([1,2,3],index=['c','b','a'])
Out[ ]: s2
c     1
b     2
a     3

In [ ]: s
Out[ ]:
a       NaN
b     100.0
c     150.0
d     200.0

In [ ]: s*s2
Out[ ]:
a       NaN
b     200.0
c     150.0
d       NaN
```

Structure de données : DataFrame



- Proche de celle du même nom dans le langage R
- Permet d'associer avec le même index de lignes des colonnes ou variables de types différents (entier, réel, booléen, caractère).
- `DataFrame` est un tableau bi-dimensionnel avec des index de lignes et de colonnes (peut également être vu comme une liste de Series partageant le même index).
- L'index de colonne (noms des variables) est un objet de type dict (dictionnaire).

DataFrame

Création avec dict de listes de longueur égale

```
In [ ]: data={'state': ['FL', 'FL', 'GA', 'GA', 'GA'],
        'year': [2010, 2011, 2008, 2010, 2011],
        'pop': [18.8, 19.1, 9.7, 9.7, 9.8]}
In [ ]: frame=pd.DataFrame(data)
In [ ]: frame
Out[ ]:
     pop  state   year
0   18.8     FL   2010
1   19.1     FL   2011
2    9.7     GA   2008
3    9.7     GA   2010
```

Création avec dict de dicts

```
In [ ]: pop_data={'FL':{2010:18.8, 2011:19.1},
          'GA':{2008: 9.7, 2010: 9.7, 2011:9.8}}
In [ ]: pop=   pd.DataFrame(pop_data)

In [ ]: pop
Out[ ]:
          FL    GA
2008     NaN   9.7
2010    18.8   9.7
2011    19.1   9.8
```

DataFrame

- Les colonnes peuvent être récupérées comme la série
  - notation dict
  - notation d'attribut
- Les lignes peuvent être récupérées par position ou par nom (En utilisant l'attribut `ix`)

```
In [ ]: frame['state']          # ou In [ ]: frame.state
Out [ ]:
0     FL
1     FL
2     GA
3     GA
4     GA
Name: state
In [ ]: frame.describe
Out[12]:
<bound method NDFrame.describe
of      pop state  year
0   18.8     FL  2010
1   19.1     FL  2011
2    9.7     GA  2008
3    9.7     GA  2010
4    9.8     GA  2011>
```

DataFrame


```
# ordre des colonnes
In []: pd.DataFrame(data,columns=["year","state","pop"])

Out[]:
   year state   pop
0  2010    FL  18.8
1  2011    FL  19.1
2  2008    GA   9.7
3  2010    GA   9.7
4  2011    GA   9.8

#liste des colonnes
In []: frame.columns

Out[43]: Index(['pop', 'state', 'year'], dtype='object')
```

DataFrame

- De nouvelles colonnes peuvent être ajoutées (par un calcul ou par une affectation directe). Les colonnes qui ne sont pas présentes dans la série de données sont `NaN`

```
In [ ]: frame['other'] = NaN
In [ ]: frame
Out[ ]:
    pop state  year  other
0  18.8    FL  2010    NaN
1  19.1    FL  2011    NaN
2   9.7    GA  2008    NaN
3   9.7    GA  2010    NaN
4   9.8    GA  2011    NaN

In [ ]:frame['calc']=frame['pop']*2
In [ ]: frame
Out[ ]:
    pop state  year  other  calc
0  18.8    FL  2010    NaN  37.6
1  19.1    FL  2011    NaN  38.2
2   9.7    GA  2008    NaN  19.4
3   9.7    GA  2010    NaN  19.4
4   9.8    GA  2011    NaN  19.6
```

DataFrame - réindexation

- Création d'un nouvel objet avec les données conformes à un nouvel index

```
In [ ]: obj=pd.Series(['blue','purple', 'red'],index=[0,2,4])
In [ ]: obj
Out[ ]:  0       blue
         2     purple
         4        red
In [ ]: obj.reindex(range(4))
Out[ ]:  0       blue
         1        NaN
         2     purple
         3        NaN
In [ ]: obj.reindex(range(5),fill_value='black')
Out[ ]:  0       blue
         1      black
         2     purple
         3      black
         4        red
In [ ]: obj.reindex(range(5),method='ffill')
Out[ ]:  0       blue
         1       blue
         2     purple
         3     purple
         4        red
```

Fonctionnalité

- Synthèse et Statistiques descriptives

```
In [ ]: pop
Out[ ]:
          FL    GA
2008     NaN   9.7
2010    18.8   9.7
2011    19.1   9.8

In [ ]: pop.sum()
Out[ ]:
FL     37.9
GA     29.2

In [ ]: pop.mean()
Out[ ]:
FL     18.950000
GA      9.733333
```

```
In [ ]: pop.describe()
Out[ ]:
                FL         GA
count     2.000000   3.000000
mean     18.950000   9.733333
std       0.212132   0.057735
min      18.800000   9.700000
25%            NaN   9.700000
50%            NaN   9.700000
75%            NaN   9.750000
max      19.100000   9.800000
```

Fonctionnalité

- indexation Boolean

```
In [ ]: pop
Out[ ]:
         FL    GA
2008    NaN   9.7
2010   18.8   9.7
2011   19.1   9.8

In [ ]: pop < 9.8
Out[ ]:
          FL      GA
2008   False    True
2010   False    True
2011   False   False

In [ ]: pop[pop < 9.8]=0
In [ ]: pop
Out[ ]:
         FL    GA
2008    NaN   0.0
2010   18.8   0.0
2011   19.1   9.8
```

Chargement des données

- pandas permet de gérer le chargement de données de différentes façons
- des données en fichier texte
  - read_csv
  - read_table
- Des données structurées (JSON, XML, HTML)
  - fonctionne bien avec les bibliothèques existantes
- Excel (dépend des packages) xlrd et openpyxl)
- Base de données
  - module `pandas.io.sql` (`read_frame`)

Chargement des données

Lire, écrire des tables de données.

Syntaxe

Exemple :

```
# importation
import pandas as pd
data=pd.read_csv("fichier.csv")
# ou encore de facon equivalente
data=pd.read_table("fichier.csv", sep=",")
# qui utilise la tabulation comme
# separateur par defaut
```

Liste des principales options : (lien à la liste complète ici)

path	chemin ou nom du fichier ou URL
sep	délimiter comme , ; | $\backslash t$ ou $\backslash s+$ pour un nombre variable d'espaces
header	défaut 0, la première ligne contient le nom des variables ; si `None` les noms sont générés.
index_col	noms ou numéros de colonnes définissant les index de lignes.

Chargement des données

Liste des principales options : (Suite)

names	si `header=None`, liste des noms des variables
nrows	utile pour tester et limiter le nombre de lignes à lire
usecols	sélectionne une liste des variables à lire pour éviter de lire des champs ou variables volumineuses et inutiles.
skip_blank_lines	à `True` pour sauter les lignes blanches
converters	appliquer une fonction à une colonne ou variable
chunksize	taille des morceaux à lire itérativement

Remarques :

- De nombreuses options de gestion des dates et séries ne sont pas citées ici
- `chunksize` provoque la lecture d'un gros fichier par morceaux de même taille (nombre de lignes). Des fonctions (comptage, dénombrement...) peuvent ensuite s'appliquer itérativement sur les morceaux

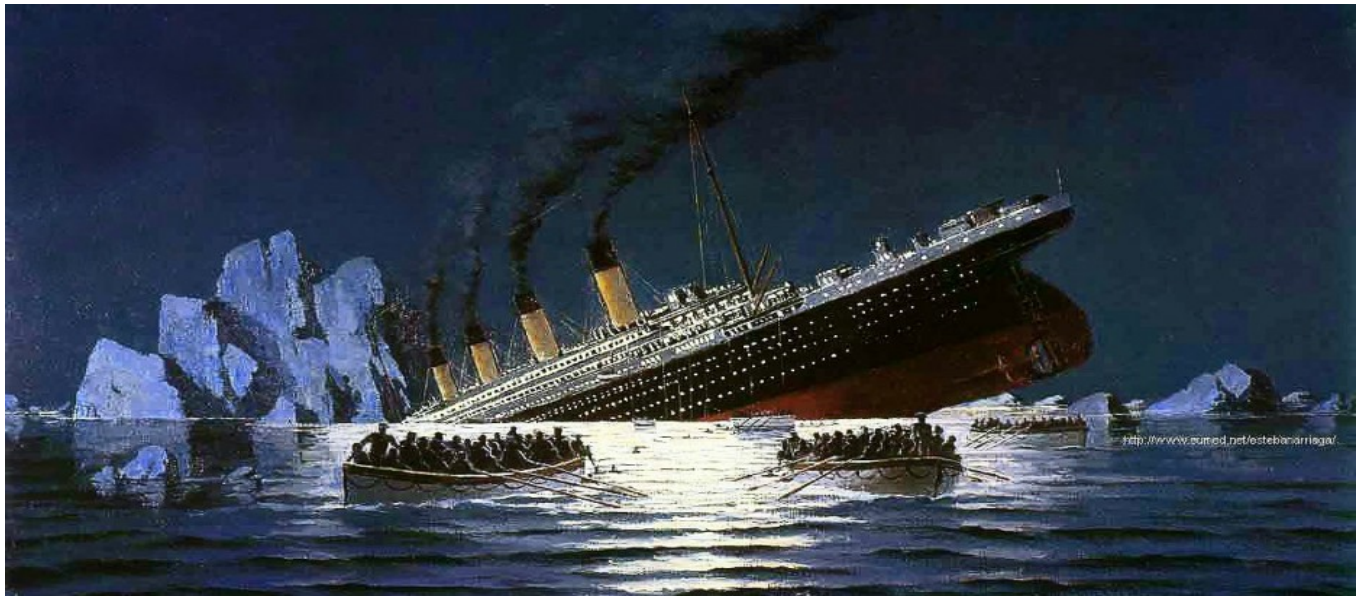
Exemple

Considérons à présent des données réelles

C’est plus parlant!!!

Exemple -Titanic-

- Les données choisies pour illustrer cet exemple sont issues d'une compétition du site Kaggle : Titanic:Machine learnic from Disaster. Le concours est terminé, mais les données sont toujours disponibles sur le site.
- Ces données vont illustrer l'utilisation de pandas. Elles sont directement lues à partir de leur URL. Ou sinon sur Moodle, voir le Cours "Python pour la programmation scientifique" pour les charger vers le répertoire de travail de Python.



Exemple -Titanic-

Récupérer les données

Charger en mémoire le jeu de données avec la librairie pandas :

```
# Importations

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# tester la lecture
df = pd.read_csv("titanic-train.csv",nrows=5)
print(df)
df.tail()

# tout lire
df = pd.read_csv("titanic-train.csv")
df.head()
type(df)
df.dtypes

# visualiser les 10 premieres lignes
df.head(10)
```

Exemple -Titanic-

Colonnes

Description des variables :

- **survival** : indique la mort ou la survie du passager pour (0 ou 1)
- **pclass** : classe des chambres du navire, à 3 niveaux (1 ; 2 ou 3)
- **name** : nom de la personne
- **sex** : sexe du passager
- **age** : âge du passager
- **sibsp** (Sibling and Spouse) : nombre de membres de la famille du passager de type frère, soeur, demi-frère, demi-soeur, époux, épouse
- **parch** (Parent and Child) : nombre de membres de la famille du passager du type père, mère, fils, fille, beau-fils, belle-fille
- **ticket** : numéro du ticket
- **fare** : prix du ticket
- **cabin** : numéro de cabine
- **embarked** : port d'embarquement du passager (C = Cherbourg; Q = Queenstown; S = Southampton)

Exemple -Titanic-

Résumé des données

Les commandes **dtype** et **count** vont nous donner des informations utiles pour disposer d'une vue d'ensemble du problème et définir les candidats pour notre premier modèle.

```
In[] : df.dtypes
Out[]:
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

```
In [] : df.count()
Out[]:
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Cabin          204
Embarked       889
dtype: int64
```

Exemple -Titanic-

Sélectionner et renommer les colonnes qu'on veut utiliser

```
# Selectionner et renommer les colonnes qu'on veut utiliser
df=pd.read_csv("titanic-train.csv",skiprows=1,
    header=None,usecols=[1,2,4,5,9,11], names=["Surv", "Classe",
    "Genre", "Age", "Prix", "Port"],dtype={"Surv":object,
"Classe":object, "Genre":object, "Port":object})

df.head()
   Surv  Classe    Genre    Age      Prix  Port
0     0       3     male   22.0    7.2500     S
1     1       1   female   38.0   71.2833     C
2     1       3   female   26.0    7.9250     S
3     1       1   female   35.0   53.1000     S
4     0       3     male   35.0    8.0500     S

df.dtypes
Surv         object
Classe       object
Genre        objects
Age         float64
Prix        float64
Port         object
dtype: object
```

Exemple -Titanic-

Statistiques descriptives élémentaires

```
# description univariee
df.describe()
df["Age"].hist()          # figure 1
plt.show()
df.boxplot("Age")         # figure 2
plt.show()

# qualitatif
In [ ]: df["Surv"].value_counts()
Out[ ]:
0    549
1    342
Name: Surv, dtype: int64

In [ ]: df["Classe"].value_counts()
Out[ ]:
3    491
1    216
2    184
Name: Classe, dtype: int64
In [ ]: df["Genre"].value_counts()

In [ ]: df["Port"].value_counts()
```

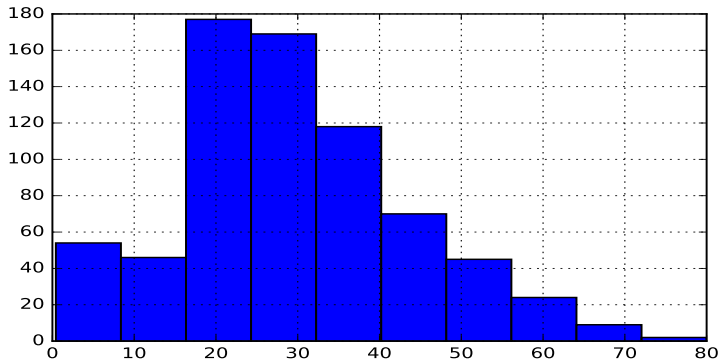


Figure: Figure 1

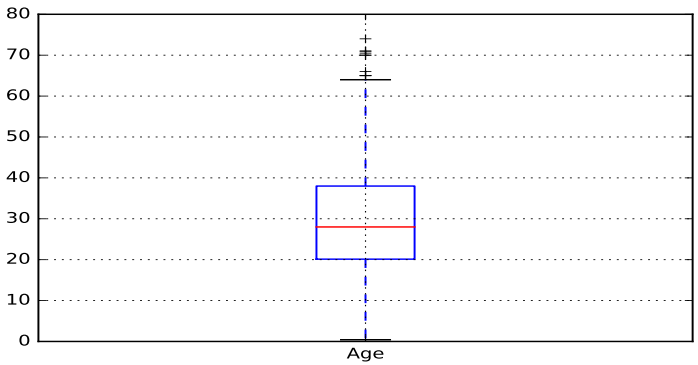


Figure: Figure 2

Exemple -Titanic-

Statistiques descriptives élémentaires

```
# description bivariee
# Correlationdf.corr()
# Nuage
df.plot(kind="scatter",x="Age",y="Prix")
plt.show() # figure 3
# afficher une selection
df[df["Age"]>60][["Genre","Classe",
                  "Age","Surv"]]
Out[]:
      Genre  Classe    Age  Surv
33     male       2   66.0     0
54     male       1   65.0     0
96     male       1   71.0     0
116    male       3   70.5     0
...
..
745    male       1   70.0     0
851    male       3   74.0     0

# parallele boxplots
df.boxplot(column="Age",by="Classe")
plt.show() # figure 4
df.boxplot(column="Prix",by="Surv")
plt.show()
```

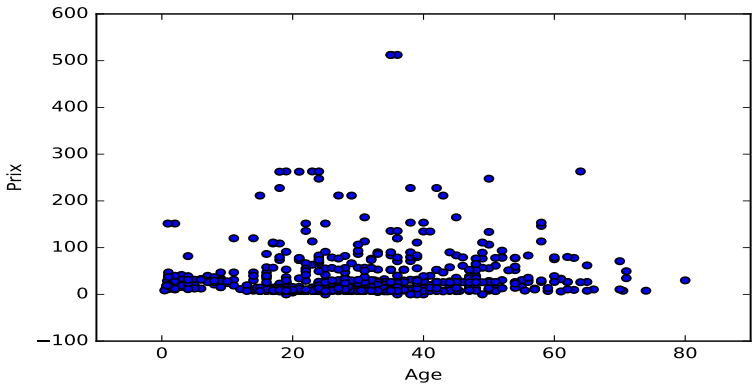


Figure: Figure 3

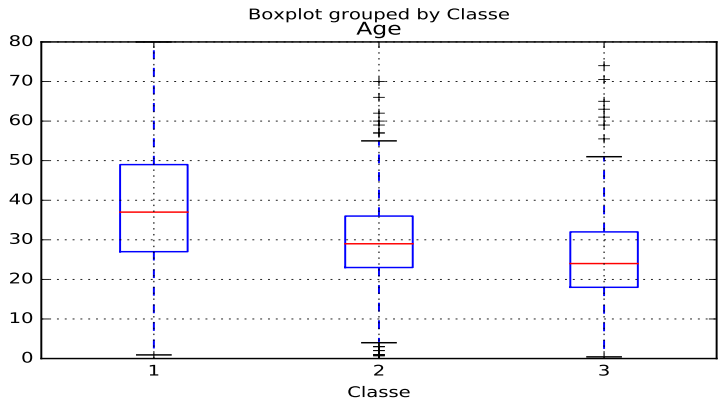


Figure: Figure 4

Exemple -Titanic-

Statistiques descriptives élémentaires

```
# table de contingence


table=pd.crosstab(df["Surv"],df["Classe"])

print(table)

# Mosaics plots

from statsmodels.graphics.mosaicplot
     import mosaic

mosaic(df,["Classe","Genre"]) # figure 5
plt.show()

mosaic(df,["Surv","Classe"])  # figure 6
plt.show()
```

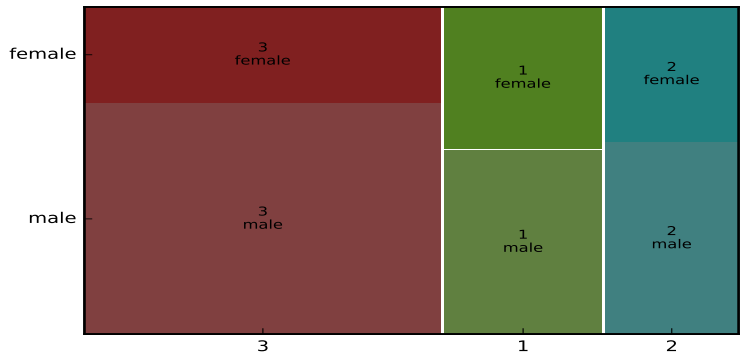


Figure: Figure 5

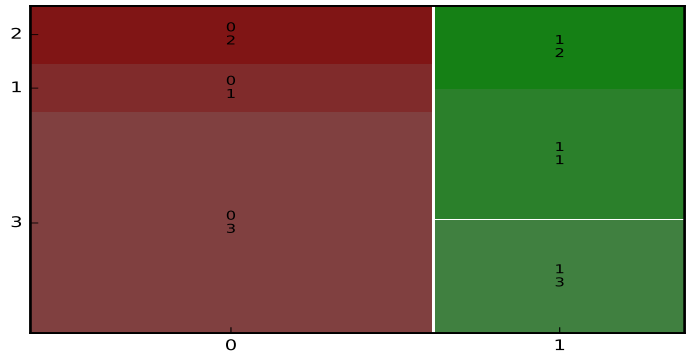


Figure: Figure 6

Exemple -Titanic-

Imputation de données manquantes : Point souvent délicat.

- **De nombreuses stratégies.** Nous décrivons ici que les plus élémentaires :
  - **Supprimer toutes les observations présentant des données manquantes**

```
In [ ]: df.count()
Out[ ]:
Surv       891
Classe     891
Genre      891
Age        714
Prix       891
Port       889
dtype: int64
```

```
# les individus ou lignes
In [ ]: df1 = df.dropna(axis=0)
Out[ ]:
Surv       712
Classe     712
Genre      712
Age        712
Prix       712
Port       712
dtype: int64

# les variables ou colonnes
In [ ]: df2=df.dropna(axis=1)
In [ ]: df2.count()
Out[ ]:
Surv       891
Classe     891
Genre      891
Prix       891
dtype: int64
```

Exemple -Titanic-

Imputation de données manquantes :

Autres stratégies :

- **Cas quantitatif : une valeur manquante est imputée par la moyenne ou la médiane.**

```
# Remplacement par la mediane d'une variable
# quantitative
df=df.fillna(df.median())
df.describe()
# par la modalite "mediane" de AgeQ
  #Discretisation d'une variable quantitative
df["AgeQ"]=pd.qcut(df.Age,3,labels=["Ag1", "Ag2", "Ag3"])
df.info()
df.AgeQ=df["AgeQ"].fillna("Ag2")
# par le port le plus frequent
df["Port"].value_counts()
df.Port=df["Port"].fillna("Ps")
df.info()
```

Exemple -Titanic-

Imputation de données manquantes :

Autres stratégies :

- **Cas qualitatif : modalité la plus fréquente ou répartition aléatoire selon les fréquences observées des modalités**

```
# par le port le plus frequent
df["Port"].value_counts()
S     644
C     168
Q      77
Name: Port, dtype: int64

df.Port=df["Port"].fillna("S")
df.info()
```

- Cas d'une série chronologique : imputation par la valeur précédente ou suivante ou par interpolation linéaire, polynomiale ou lissage spline.

Place au TP

À vous de jouer!!!