# Lecture/écriture/manipulation/analyse de données

# Pandas

# Objectifs :

Manipulation de grands volumes de données structurées (tabulées, multi-dimensionnelles, voire hétérogènes) dans le but de leur exploitation.

La bibliothèque contient donc un grand nombre d'outils pour **récupérer, agréger, combiner, transformer, trier, découper, déplacer** et **exporter** les données, gérer les valeurs manquantes, **calculer** des statistiques et **produire** des graphiques.

Liens :

- Site officiel
- Python Simple

# Structures de données de pandas :

- ***Series*** **(1 dimension)**

**Création :**

```
s = pandas.Series([1, 2, 5, 7])
```

- ***DataFrame*** **(2 dimensions)**

**Création :**

```
ar = numpy.array([[1.1, 2, 3.3, 4], [2.7, 10, 5.4, 7], [5.3, 9, 1.5, 15]])
```

```
df = pandas.DataFrame(ar, index = ['a1', 'a2', 'a3'], columns = ['A', 'B', 'C', 'D'])
```

# Fonctions de bases

- `df.dtypes` : les types des différentes colonnes
- df.info() : des infos sur le dataframe
- df.head(2) : renvoie un dataframe avec les 2 premières lignes.
- df.tail(2) pour les deux dernières.
- df.columns : les noms des colonnes / df.columns.values : le nom des colonnes sous forme d'array numpy.
- df.index : les noms des lignes (individus) /df.index.values : le nom des lignes sous forme d'array numpy.
- df.values : pour récupérer le dataframe sous forme d'array numpy 2d.
- df.describe() : renvoie un dataframe donnant des statistiques sur les valeurs uniquement sur les colonnes numériques (faire df.describe(include = 'all') pour avoir toutes les colonnes)
- df.shape : renvoie la dimension du dataframe sous forme (nombre de lignes, nombre de colonnes)
- df.memory_usage() : donne une série avec la place occupée par chaque colonne
- df.shape : renvoie la dimension du dataframe sous forme (nombre de lignes, nombre de colonnes)
- df.memory_usage() : donne une série avec la place occupeée par chaque colonne (sum(df.memory_usage()) donne la mémoire totale occupée).

# Indexation/Sélection

- **Accès à une partie des données :**
  - avec le nom de la variable/index de la ligne :df['A']/df.loc['a2'] ou les 2 ; df.loc[['a2', 'a3'], ['A', 'C']]
  - avec les numéros des lignes et colonnes df.iloc[1:3,[0, 2]]
- **Affectation** df.iat[1, 2] = 7.
- **Accés avec condition** : df[df['A'] > 2] : renvoie un dataframe avec seulement les lignes où la condition est vérifiée : f[(df['A'] > 2) & - (df['B'] < 6)]
- **Tester les valeurs nulles d'une colonne** : pd.isnull(df['A']) ou aussi df['A'].isnull()

# Modifications de Dataframes

- **Changer une cellule** : df.loc[df.index[3], 'A']
- **Renommer l'ensemble des colonnes** : df.columns = ['a', 'B']

  renommer une colonne en particulier : myList = list(df.columns); myList[0] = 'a'; df.columns = myList
- **Réordonner des colonnes** :df.reindex(columns = ['B', 'C', 'A'])
- **Ajout d'une colonne :**

  df['E'] = pd.Series([1, 0, 1], index = ['a1', 'a2', 'a3'])

  df.assign(E = df['A'] + df['B'], F = 2 * df['A'])
- **Destruction de colonnes** : del df['A'] /df.drop(['A', 'C'], axis = 1)
- **Valeurs non définies**

  df.dropna(how = 'any') : renvoie un dataframe avec les lignes contenant au moins une valeur NaN supprimée (how = 'all' : supprime les lignes où toutes les valeurs sont NaN).

  df.dropna(axis = 1, how = 'any') : supprime les colonnes ayant au moins un NaN plutôt que les lignes (le défaut est axis = 0).

  df.fillna(0) : renvoie un dataframe avec toutes les valeurs NaN remplacées par 0.

  df.isnull() : renvoie un dataframe de booléens, avec True dans toutes les cellules non définies.
- **Copie** :df2 = df.copy() : df2 est alors un dataframe indépendant.
- **Lignes redondantes enlevées** en n'en conservant qu'une seule df.drop_duplicates()
- **Transposition** : df.T

# Opérations sur les dataframes :

**Opérations élémentaires :**

- df1 + df2,2 * df + 3, 1 / df , df ** 2
- pour des dataframes booléens -df : not, df1 & df2 : et, df1 | df2 : ou., df1 ^ df2 : ou exclusif.
- opérations de comparaison : df1.eq(df2), df1.ne(df2), df1.lt(df2), df1.le(df2), df1.lt(df2), df1.le(df2) : égalité, non
- réductions booléennes :
  a. (df > 0).all() : renvoie une série avec un élément par colonne qui est True si toutes les valeurs sont > 0
  b. (df > 0).any() : renvoie une série avec un élément par colonne qui est True si une des valeurs est > 0

**Ajout colonne :**

- df.add(df['A'], axis = 'rows') (ou df.add(df['A'], axis = 'rows')).
- idem avec sub(), mul(), div() pour les autres opérations.
- pour l'ajout d'une ligne à toutes les autres : df.add(df.iloc[0], axis = 'columns') ou df.add(df.iloc[0], axis = 1)

**Tri**

- df.sort_index(axis = 0, ascending = False) : renvoie un dataframe avec les lignes triées par ordre décroissant des labels (le défaut est ascendant)
- df.sort_values(by = 'C') : renvoie un dataframe avec les lignes triées de telle sorte que la colonne 'C' soit dans l'ordre croissant :

**Statistiques**

- df.mean() : renvoie une Series des moyennes de chaque colonne (en ignorant les NaN) :
- df.mean(axis = 1) : calcule les moyennes par ligne plutôt que par colonne.
- fonctions stat : min, max, median, std,var ,

# Fonction

- **enlever à chaque ligne la moyenne de la ligne** : df.sub(df.mean(axis = 1), axis = 0)
- **enlever à chaque colonne la moyenne de la colonne** : df.sub(df.mean(axis = 0), axis = 1) (mais df.sub(df.mean()) suffit).
- **Index du maximum** : df.idxmin() : renvoie une Série qui donne pour chaque colone l'index où la valeur est minimale.
- **Quantile en ligne** : df.quantile([0.25, 0.5, 0.75]))
- **Somme cumulée** :df2 = df.cumsum() :
- **Application d'une fonction** :df.apply(lambda x: f(x))
- **Concaténer** en ligne: pd.concat([df1, df2])
- **Concaténer** en colonnes : pd.concat([df1, df2], axis = 1)
- **jointure simple** (inner) pd.merge(df1, df2)
- **jointure externe** : pd.merge(df1, df2, how = 'outer')

**Réorganisation des dataframes :**

- **Pivotage** df.pivot(index = 'A', columns = 'T', values = 'V') /pivot_table
- **Stacking** : consiste à empiler les différentes colonnes : df.stack()
- **Agrégats** df.groupby('A'): groupe avec les valeurs de A
- **Lecture et écriture de fichiers**
  - df = pandas.read_csv('myFile.csv')
  - df.to_csv('myFile.csv', sep = '\t')

- **Panel** : dictionnaire de dataframes de même dimension, donc une array en 3D