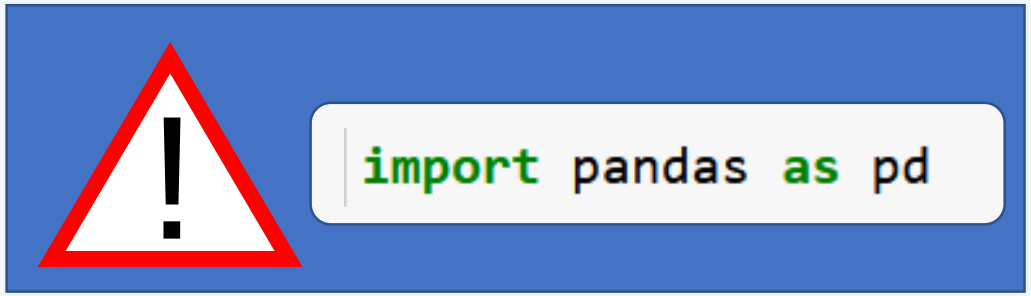
Introduction à pandas

Un tableur à la Python

Plan de la présentation

- Présentation et gestion des objets
  - pd.Series, pd.DataFrame
- Premiers traitements
- Création et gestion des figures
- Introduction aux séries temporelles



Présentation des objets

pd.Series et pd.DataFrames

pd.Series: présentation

- Séries de valeurs avec des indices


```
Elfe      200
Nain      120
Hobbit    110
dtype: int64
```

pd.Series: quels indices et quelles valeurs?

- Les variables peuvent être typées ou non

```
valeur 1     0
valeur 2     1
valeur 3     4
valeur 4     9
valeur 5    16
valeur 6    25
valeur 7    36
valeur 8    49
dtype: int64
```

```
valeur 1    1.000000
valeur 2    0.540302
valeur 3   -0.416147
valeur 4   -0.989992
valeur 5   -0.653644
valeur 6    0.283662
valeur 7    0.960170
valeur 8    0.753902
dtype: float64
```

```
valeur 1           Word
valeur 2          Excel
valeur 3    Open office
valeur 4           Word
valeur 5     Powerpoint
valeur 6      Photoshop
valeur 7         Dolphin
valeur 8    Virtual Box
dtype: object
```

```
valeur 1          Word
valeur 2           124
valeur 3       3.14159
valeur 4       Bonjour
valeur 5    [1, 2, 3]
valeur 6     Photoshop
valeur 7            x²
valeur 8          45.3
dtype: object
```

- Plusieurs types d'indices possibles

```
0     0
1     1
2     4
3     9
4    16
5    25
6    36
7    49
dtype: int64
```

```
valeur 1     0
valeur 2     1
valeur 3     4
valeur 4     9
valeur 5    16
valeur 6    25
valeur 7    36
valeur 8    49
dtype: int64
```

```
2012-01-03     0
2012-01-04     1
2012-01-05     4
2012-01-06     9
2012-01-07    16
2012-01-08    25
2012-01-09    36
2012-01-10    49
Freq: D, dtype: int64
```

pd.Series: Création d'un objet

- Source: une liste de valeur + une liste d'indices

```
personnages = ['Elfe', 'Nain', 'Hobbit']
taille = [200, 120, 110]
```

- Utiliser **pd.Series()**

```
series_taille = pd.Series(data=taille, index=personnages, name='taille')
```

```
Elfe      200
Nain      120
Hobbit    110
Name: taille, dtype: int64
```

Nécessaire

Facultatif, si non précisé: 0,1,2 ,…

Facultatif

pd.Series: Accéder aux valeurs

```
Elfe      200
Nain      120
Hobbit    110
Name: taille, dtype: int64
```

- Accès aux valeurs par l'indice (comme un dictionnaire)

```
taille_elfe = series_taille.loc['Elfe']
```

- Accès aux valeurs par la position (comme une liste)

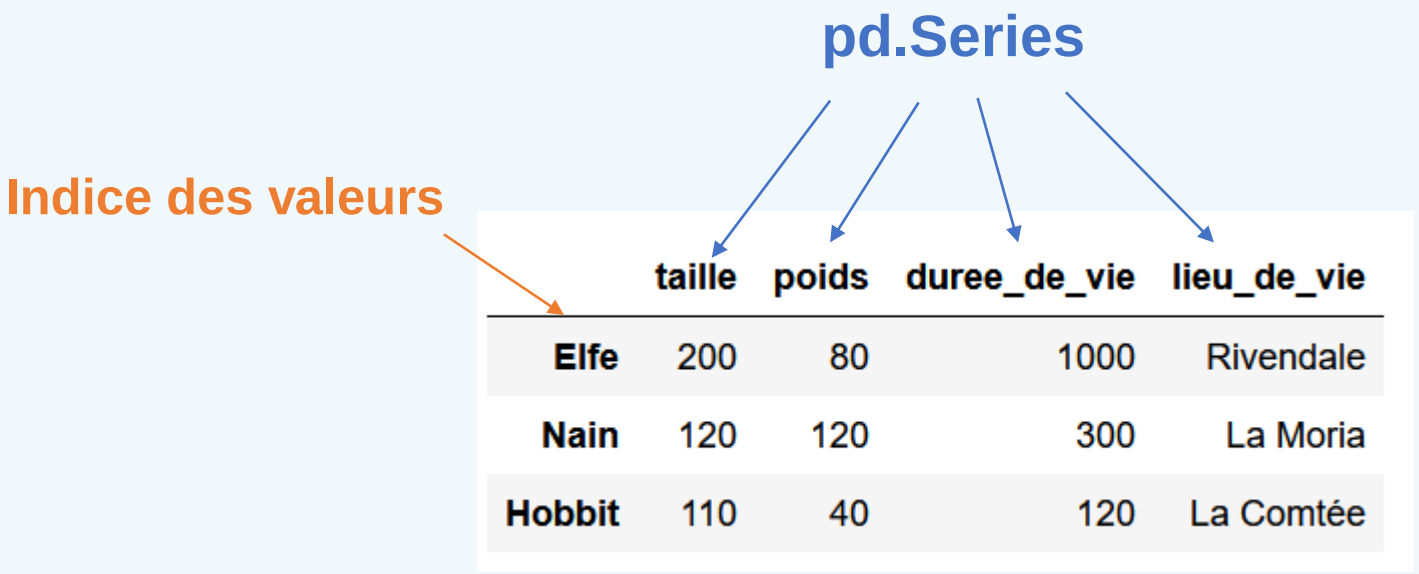
```
taille_elfe = series_taille.iloc[0]
```

- Accéder au tableau de valeur

```
series_taille.values
```

pd.DataFrame: présentation

- Une collection de Series sur les mêmes indices
- Les series ont des noms



	taille	poids	duree_de_vie	lieu_de_vie
Elfe	200	80	1000	Rivendale
Nain	120	120	300	La Moria
Hobbit	110	40	120	La Comtée

pd.DataFrame: Création d'un objet

- Sources:
  - une liste d'indices
  - Un ensemble de valeurs
    - Un ensemble de listes
    - Un tableau de valeurs

```
personnages = ['Elfe', 'Nain', 'Hobbit']
taille = [200, 120, 110]
poids = [80, 120, 40]
duree_de_vie = [1000, 300, 120]
lieux_de_vie = ['Rivendale', 'La Moria', 'La Comtée']
```

- pd.DataFrame()

```
personnages_lotr = pd.DataFrame(data = {"taille":taille,
                                        "poids":poids,
                                        "duree_de_vie":duree_de_vie,
                                        "lieu_de_vie":lieux_de_vie},
                                index=personnages)
```

```
personnages_lotr = pd.DataFrame(data = [[200,  80, 1000, 'Rivendale'],
                                        [120, 120, 300 , 'La Moria' ],
                                        [110, 110, 40  , 'La Comtée']],
                                columns = ['taille','poids','durée_de_vie', 'lieu_de_vie'],
                                index=personnages)
```

pd.DataFrame: Accès aux données

	taille	poids	duree_de_vie	lieu_de_vie
Elfe	200	80	1000	Rivendale
Nain	120	120	300	La Moria
Hobbit	110	40	120	La Comtée

- **Accéder aux valeurs d'un indice**
  - Par l'indice

```
infos_elfe = personnages_lotr.loc['Elfe']
```

  - Par la position

```
infos_elfe = personnages_lotr.iloc[0]
```

- Renvoie une pd.Series

```
taille                200
poids                  80
duree_de_vie         1000
lieu_de_vie     Rivendale
Name: Elfe, dtype: object
```

- **Accéder aux valeurs d'une pd.Series**
  - Par le nom de variable

```
poids = personnages_lotr['poids']
poids = personnages_lotr.poids
```

  - Par la position

```
poids = personnages_lotr.iloc[:,1]
```

- Renvoie une pd.Series

```
Elfe      80
Nain     120
Hobbit    40
Name: poids, dtype: int64
```

pd.DataFrame: Ajouter des données

- **Ajouter une colonne**
  - Il faut une autre liste de valeurs

```
exemple_de_personnage = ['Legolas', 'Gimli', 'Frodon']
```

  - Ajouter la colonne comme si on voulait accéder aux valeurs

```
personnages_lotr['exemple_de_personnage'] = exemple_de_personnage
```

	taille	poids	duree_de_vie	lieu_de_vie	exemple_de_personnage
Elfe	200	80	1000	Rivendale	Legolas
Nain	120	120	300	La Moria	Gimli
Hobbit	110	40	120	La Comtée	Frodon

- On peut aussi écraser une colonne de cette manière

pd.DataFrame: Ajouter des données

- **Ajouter une ligne**
  - Créer une dataframe similaire avec les valeurs à ajouter

```
dataframe_humain = pd.DataFrame({"taille":[175],
                                 "poids":[75],
                                 "duree_de_vie":[80],
                                 "lieu_de_vie":["Gondor"],
                                 "exemple_de_personnage":['Aragorn']},
                                index=['Humain'])
```

  - Concaténer les deux dataframes ensemble **ATTENTION: nouvel objet**

```
personnages_lotr = pd.concat([personnages_lotr, dataframe_humain])
```

	taille	poids	durée_de_vie	lieu_de_vie	exemple_de_personnage
Elfe	200	80	1000	Rivendale	Legolas
Nain	120	120	300	La Moria	Gimli
Hobbit	110	110	40	La Comtée	Frodon
Humain	175	75	80	Gondor	Aragorn

Résumé : les objets pandas

pd.Series : une seule variable

pd.DataFrame : plusieurs variables alignées

- Accès facile aux valeurs en fonction des indices
- Objets modifiables

Lire et écrire des fichiers

Format supportés

- Pandas supporte de nombreux formats en lecture et écriture
  - csv
  - excel (xls, xlsx) : *requiert openpyxl*
  - txt
  - json
  - …
- Tout ce qui ressemble à un tableau de valeurs
- Possibilité d’ouvrir directement depuis internet

Ouvrir un fichier

- Pour ouvrir un fichier on utilise pd.read_format()

```
penguins = pd.read_json('../data/penguins_dataset.json')

planets = pd.read_excel('../data/exoplanets_discoveries.xlsx')

boats = pd.read_csv('../data/fishing_boats.csv')
```

- Quelques arguments important
  - index_col → indiquer la colonne correspondant aux indices
  - skiprows → ne pas ouvrir les n premières lignes
  - nrows → ouvrir seulement n premières lignes

```
boats = pd.read_csv('../data/fishing_boats.csv', index_col=1, skiprows=5, nrows=10)
```

  - Plen d'autres possibilités

Sauvegarder un fichier

- Pour sauvegarder un fichier on utilise pd.to_format()

```
personnages_lotr.to_csv('../data/lotr_personnages.csv')

personnages_lotr.to_json('../data/lotr_personnages.json')

personnages_lotr.to_excel('../data/lotr_personnages.xlsx')
```

Présentation des exemples

Présentation des dataframes exemples

- Penguins: caractéristiques de manchots

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	None
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

- Souce: dataset exemple Seaborn

Présentation des dataframes exemples

- Planets: découverte d'exoplanètes

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

- Souce: dataset exemple Seaborn

Présentation des dataframes exemples

- Boats: bateaux de pêche norvégiens

	ID	peche_reussie	age_moteur	taille_bateau	puissance_moteur	port	poids_prises	valeur_prises	v
0	1993001257	1.0	10.0	10.50	367.0	RISØR	5.0	95.70	
1	1993005128	1.0	26.0	21.30	970.0	BÅTSFJORD	19.0	564.59	
2	1996007882	0.0	32.0	12.13	190.0	ØKSNES	0.0	0.00	
3	1999009281	0.0	8.0	27.45	1014.0	AVERØY	0.0	0.00	
4	1986007228	1.0	31.0	9.22	80.0	BRØNNØY	12.5	816.34	
...	...	...	...	...	...	...	...	...	
199995	1996009732	1.0	16.0	10.66	152.0	TRONDHEIM	40.0	204.44	
199996	2012057209	0.0	4.0	8.30	230.0	GISKE	0.0	0.00	
199997	1986006960	0.0	19.0	7.40	53.0	VEGA	0.0	0.00	
199998	1982010538	0.0	34.0	7.47	22.0	TORSKEN	0.0	0.00	
199999	2008040308	0.0	0.0	27.48	1000.0	HAREID	0.0	0.00	

200000 rows × 13 columns

- Source: https://www.kaggle.com/datasets/alexanderbader/fishing-data-north-atlantic

Premières analyses

Sélection de données

- Sélection par indice ou position
  - Par position:

```
boats.iloc[3,2]
```

  - Par indice/colonne

```
boats.valeur_prises.loc[1920009055]
boats['valeur_prises'].loc[1920009055]
```

  - Plusieurs indices/colonnes

```
boats[['taille_bateau','valeur_prises']].loc[[2014064112,1998003078]]
```

ID	taille_bateau	puissance_moteur	valeur_prises
204027474	7.90	86.0	0.000000
1920004082	18.84	235.0	0.000000
1920004897	14.15	156.0	1496.407500
1920009055	10.36	100.0	1077.495833
1923010500	13.78	150.0	1733.734286
...	...	...	...
2018103253	38.65	1050.0	0.000000
2018103353	6.62	22.0	1304.160000
2018103393	5.35	40.0	2844.820000
2018103413	10.95	400.0	0.000000
2018103493	6.20	50.0	463.050000

Sélection de données par condition

- Données avec taille du bateau > 10m?

- Extraction de données

```
boats.loc[boats.taille_bateau > 10]
```

- Masquage des données

```
boats.where(boats.taille_bateau > 10, np.nan)
```

- Multi conditions?
  - et
  - ou

```
boats.loc[(boats.taille_bateau > 10)&(boats.puissance_moteur>1000)]
boats.loc[(boats.taille_bateau > 10)|(boats.puissance_moteur>1000)]
```

ID	taille_bateau	puissance_moteur	valeur_prises
204027474	7.90	86.0	0.000000
1920004082	18.84	235.0	0.000000
1920004897	14.15	156.0	1496.407500
1920009055	10.36	100.0	1077.495833
1923010500	13.78	150.0	1733.734286
...	...	...	...
2018103253	38.65	1050.0	0.000000
2018103353	6.62	22.0	1304.160000
2018103393	5.35	40.0	2844.820000
2018103413	10.95	400.0	0.000000
2018103493	6.20	50.0	463.050000

Opérations à la numpy

- Calcul sous la forme Series.mean()

```
mean_mass_penguins = penguins.body_mass_g.mean()
```

- Opérations possibles:
  - mean()
  - sum()
  - min(), max()
  - median()
  - idxmax(), idxmin()
  - quantile([q1,q2, …])
  - count() → valeurs non nan

- Ignore les nans par defaut

- Applicable sur des dataframes

```
boats.max()
```

```
taille_bateau        9.429419e+01
puissance_moteur     1.100000e+04
valeur_prises        1.722900e+06
dtype: float64
```

Tri des données

- Trier les données en fonction des indices

```
boats.sort_index()
```

- Trier les données en fonction d'une colonne

```
boats.sort_values('puissance_moteur', ascending=False)
```

ID	taille_bateau	puissance_moteur	valeur_prises
2014064112	71.100000	11000.0	605459.368750
2015071414	75.228571	10200.0	505773.928571
2017098553	81.200000	9400.0	263008.740000
2004027185	74.200000	9000.0	399144.750000
1998003078	70.000000	8920.0	226050.106061

Sélection de groupes: groupby

- On peut créer des groupes en fonction d'une colonne puis on peut appliquer les fonctions précédentes

```
penguins.groupby('sex').mean()
```

sex	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
Female	42.096970	16.425455	197.363636	3862.272727
Male	45.854762	17.891071	204.505952	4545.684524

- Fonctionne aussi pour plusieurs critères

```
penguins.groupby(['sex','species']).median()
```

sex	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
Female	Adelie	37.00	17.60	188.0	3400.0
	Chinstrap	46.30	17.65	192.0	3550.0
	Gentoo	45.50	14.25	212.0	4700.0
Male	Adelie	40.60	18.90	193.0	4000.0
	Chinstrap	50.95	19.30	200.5	3950.0
	Gentoo	49.50	15.70	221.0	5500.0

Sélection de groupes: fonctions perso

- On peu utiliser data.groupby().apply(fonction) avec une fonction personnalisée

```
def std_perso(data):
    return np.sqrt(((data - data.mean())**2).mean())
```

```
penguins.groupby('sex').bill_depth_mm.apply(std_perso)
```

```
sex
Female    1.790231
Male      1.857797
Name: bill_depth_mm, dtype: float64
```

Combiner des dataframes

- Concaténer par les indices

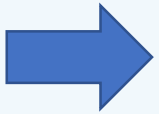
```
pd.concat([df1,df2])
```

df1

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011

df2

	method	number	orbital_period	mass	distance	year
3	Radial Velocity	1	326.03	19.4	110.62	2007
4	Radial Velocity	1	516.22	10.5	119.47	2009
5	Radial Velocity	1	185.84	4.8	76.39	2008



	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009
5	Radial Velocity	1	185.840	4.80	76.39	2008

Combiner des dataframes

- Concaténer par les colonnes

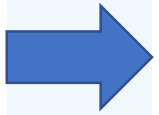
```
pd.concat([df1,df2], axis=1)
```

	method	number	orbital_period
0	Radial Velocity	1	269.300000
1	Radial Velocity	1	874.774000
2	Radial Velocity	1	763.000000
3	Radial Velocity	1	326.030000
4	Radial Velocity	1	516.220000

df1

	mass	distance	year
0	7.10	77.40	2006
1	2.21	56.95	2008
2	2.60	19.84	2011
3	19.40	110.62	2007
4	10.50	119.47	2009

df2



	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.10	77.40	2006
1	Radial Velocity	1	874.774000	2.21	56.95	2008
2	Radial Velocity	1	763.000000	2.60	19.84	2011
3	Radial Velocity	1	326.030000	19.40	110.62	2007
4	Radial Velocity	1	516.220000	10.50	119.47	2009

Une opération avancée

- Calculer le coefficient de corrélation pour chaque paire de variables

```
boats[['peche_reussie', 'age_moteur', 'puissance_moteur', 'taille_bateau','valeur_prises']].corr()
```

	peche_reussie	age_moteur	puissance_moteur	taille_bateau	valeur_prises
peche_reussie	1.000000	-0.003450	-0.001844	-0.000991	0.120867
age_moteur	-0.003450	1.000000	-0.244228	-0.179130	-0.077828
puissance_moteur	-0.001844	-0.244228	1.000000	0.915077	0.345274
taille_bateau	-0.000991	-0.179130	0.915077	1.000000	0.334019
valeur_prises	0.120867	-0.077828	0.345274	0.334019	1.000000

Résumé : premières analyses

Quelques fonctionnalités disponibles :

Sélection de données par indice/conditions	**df.loc[indice] / df.loc [condition]**
Opérations comme avec numpy	**df.mean( )**
Tri des données par colonnes/indices	**df.sort_values( ) / df.sort_index( )**
Opérations/sélections par groupe	**df.groupby(colonne).mean( )**
Combinaison de dataframes par alignement	**pd.concat([df1, df2])**
Calcul de corrélation	**df.corr()**

Et bien plus encore …

Bonus : les « One liners »

- Une commande renvoie un nouvel objet (Series, DataFrame)
- On peut donc enchainer les commandes sur une seule ligne

```
boats.loc[boats.taille_bateau < 10]\
     .groupby('ID').mean()\
     .sort_values('poids_prises', ascending=False)\
     .iloc[:10]\
     .corr()
```

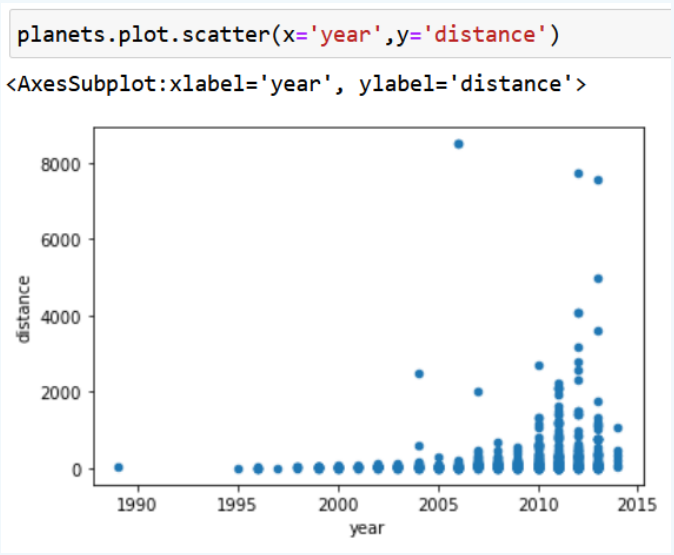
Premières visualisations

Interface de plots de pandas

- Intègre directement des librairies de plots avec une interface haut niveau

```
planets.plot.scatter(x='year',y='distance')
```

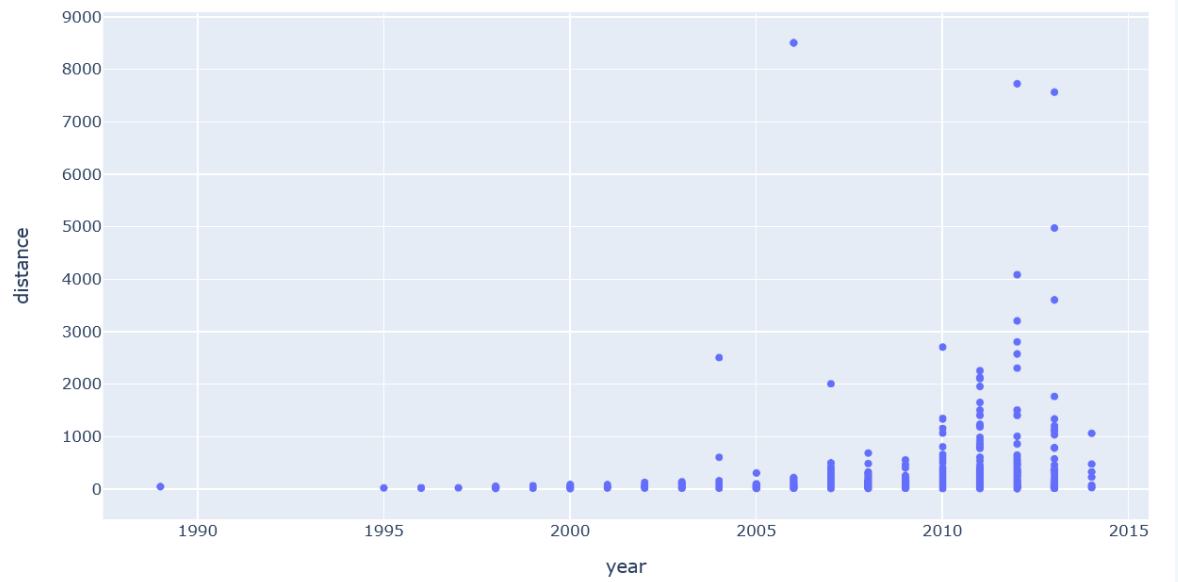
```
<AxesSubplot:xlabel='year', ylabel='distance'>
```



- Matplotlib par défaut
- Plotly/Bokeh/hvplot possible

```
pd.options.plotting.backend = "plotly"
```

```
planets.plot.scatter(x='year',y='distance')
```

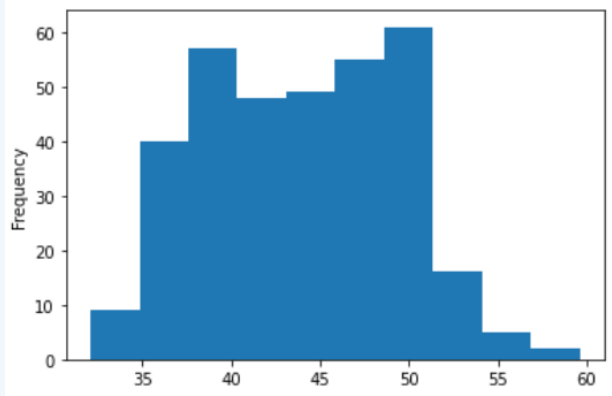


Visualisation statistiques pour les Series

- Code sous la forme Series.plot.[type de plot]()

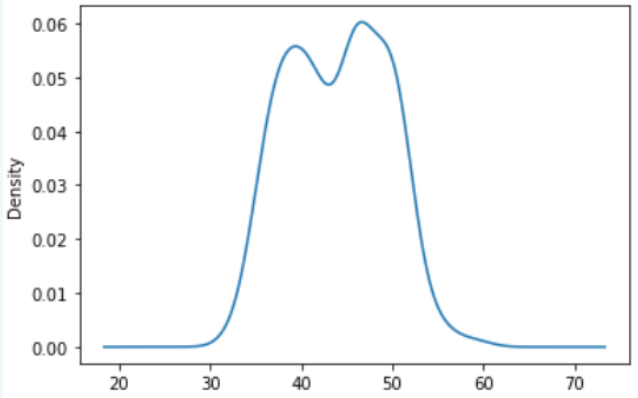
```
penguins.bill_length_mm.plot.hist()
```

```
<AxesSubplot:ylabel='Frequency'>
```



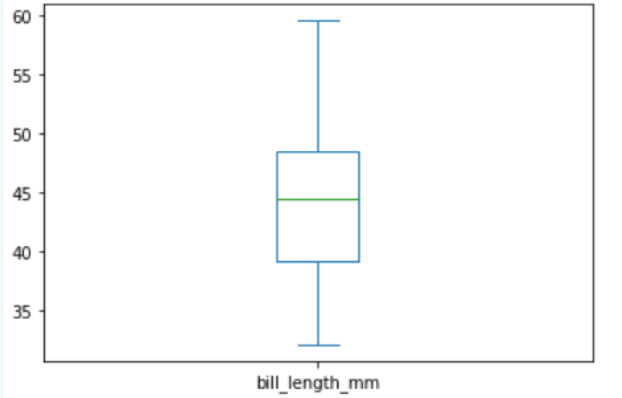

```
penguins.bill_length_mm.plot.kde()
```

```
<AxesSubplot:ylabel='Density'>
```




```
penguins.bill_length_mm.plot.box()
```

```
<AxesSubplot:>
```



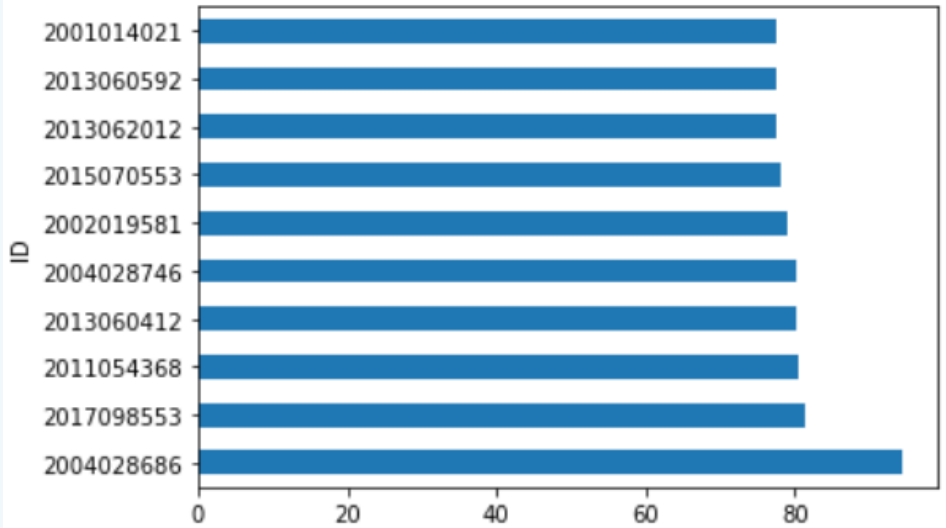
- Pandas appelle la fonction de matplotlib: mêmes arguments

Plots catégoriques: barplot

Pour un nombre de donnée limité

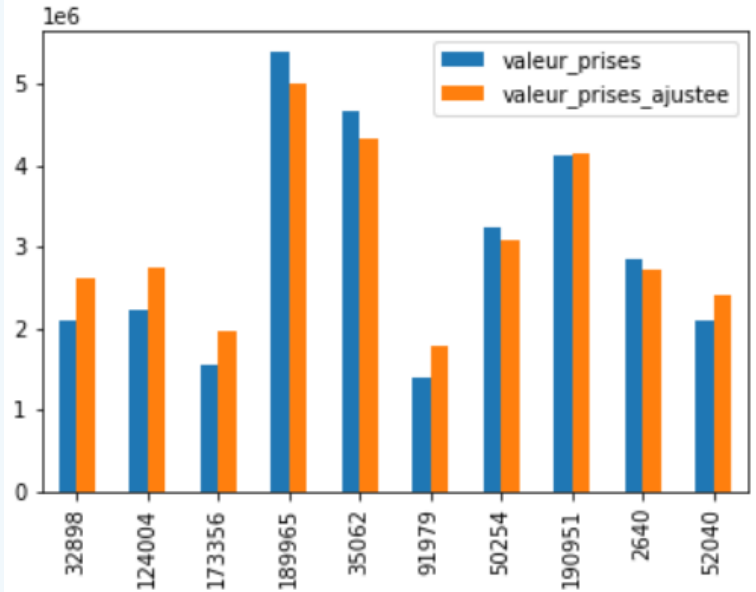
```
boats.sort_values('taille_bateau',
                  ascending=False).iloc[:10].taille_bateau.plot.barh()
```

```
<AxesSubplot:ylabel='ID'>
```




```
boats.sort_values('poids_prises', ascending=False)\
     .iloc[:10][['valeur_prises','valeur_prises_ajustee']]\
     .plot.bar()
```

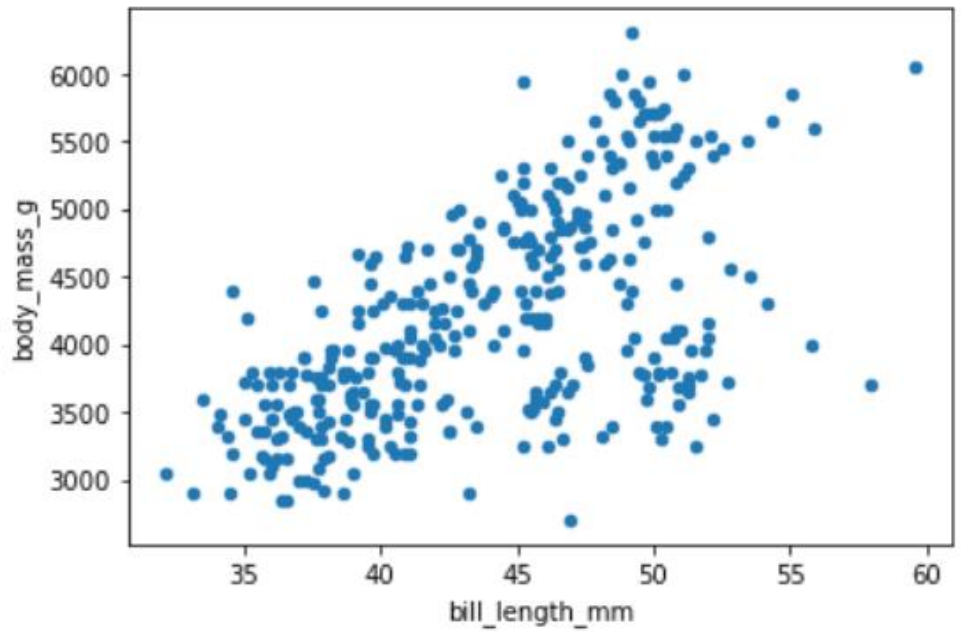
```
<AxesSubplot:>
```



Plots relationnels: scatterplot et hexbin

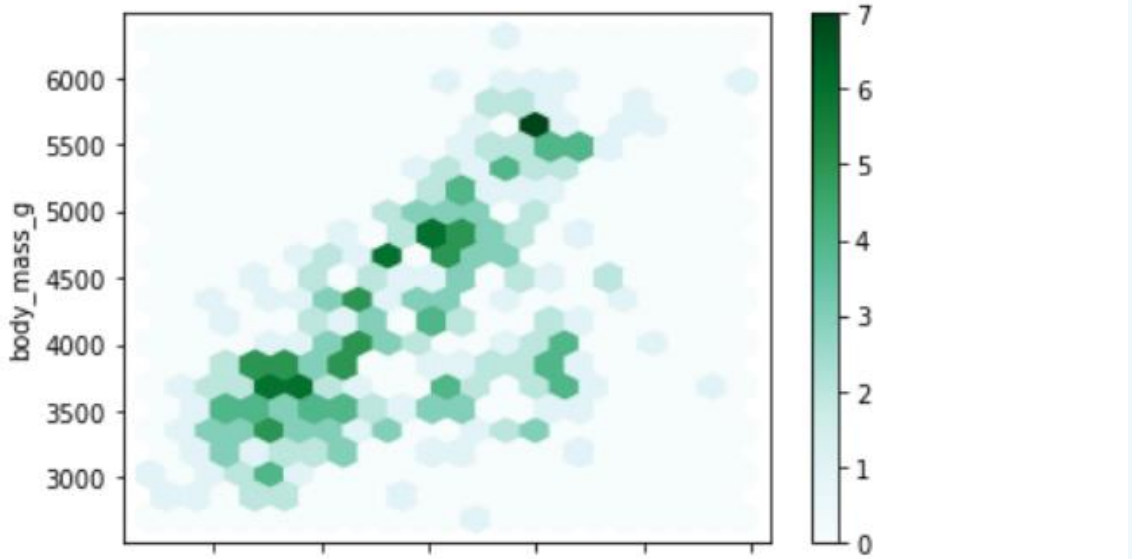
```
penguins.plot.scatter(x='bill_length_mm', y='body_mass_g')
```

```
<AxesSubplot:xlabel='bill_length_mm', ylabel='body_mass_g'>
```




```
penguins.plot.hexbin(x='bill_length_mm', y='body_mass_g',
                     gridsize=20)
```

```
<AxesSubplot:xlabel='bill_length_mm', ylabel='body_mass_g'>
```

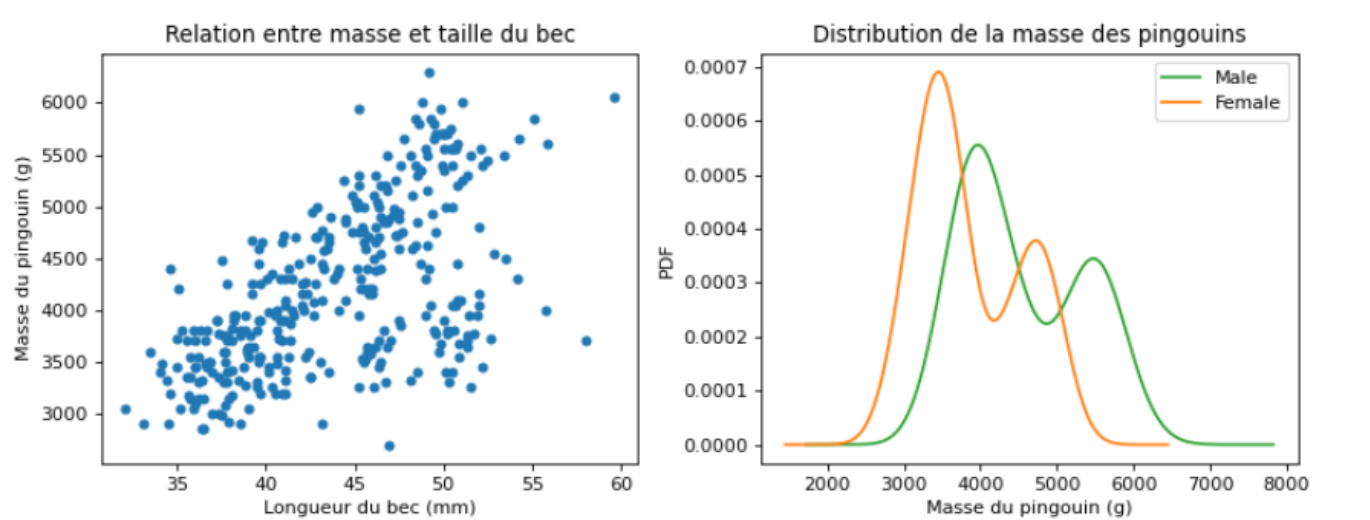


Intégration à matplotlib

```
import matplotlib.pyplot as plt
fig, axs = plt.subplots(1,2, figsize=(10,4),dpi=80)
penguins.plot.scatter(x='bill_length_mm', y='body_mass_g',ax=axs[0])
penguins.loc[penguins.sex=='Male'].body_mass_g.plot.kde(ax=axs[1], label='Male', color='tab:green')
penguins.loc[penguins.sex=='Female'].body_mass_g.plot.kde(ax=axs[1], label='Female', color='tab:orange')
axs[1].legend()

axs[0].set_title('Relation entre masse et taille du bec')
axs[1].set_title('Distribution de la masse des pingouins')
axs[0].set_ylabel('Masse du pingouin (g)')
axs[1].set_ylabel('PDF')
axs[0].set_xlabel('Longueur du bec (mm)')
axs[1].set_xlabel('Masse du pingouin (g)')

plt.tight_layout()
```



Résumé : premières figures

Visualisations statistiques : Series.plot.hist( ) Series.plot.kde() Series.plot.box( )

Visualisations catégoriques : df.plot.bar( )

Visualisations relationnelles : df.plot.scatter( ) df.plot.hexbin( ) df.plot.kde( )

Intégration à matplotlib / bokeh / plotly…

Bonus : intégration à seaborn

Librairie de visualisation
optimisée pour les dataframes.
Exemple :

```
In [3]: mpg
Out[3]:
      mpg  cylinders  displacement  horsepower  weight  acceleration  model_year origin
0    18.0          8         307.0       130.0    3504          12.0          70    usa
1    15.0          8         350.0       165.0    3693          11.5          70    usa
2    18.0          8         318.0       150.0    3436          11.0          70    usa
3    16.0          8         304.0       150.0    3433          12.0          70    usa
4    17.0          8         302.0       140.0    3449          10.5          70    usa
```

```
# Plot miles per gallon against horsepower with other semantics
sns.relplot(x="horsepower", y="mpg", hue="origin", size="weight",
            sizes=(40, 400), alpha=.5, palette="muted",
            height=6, data=mpg)
```

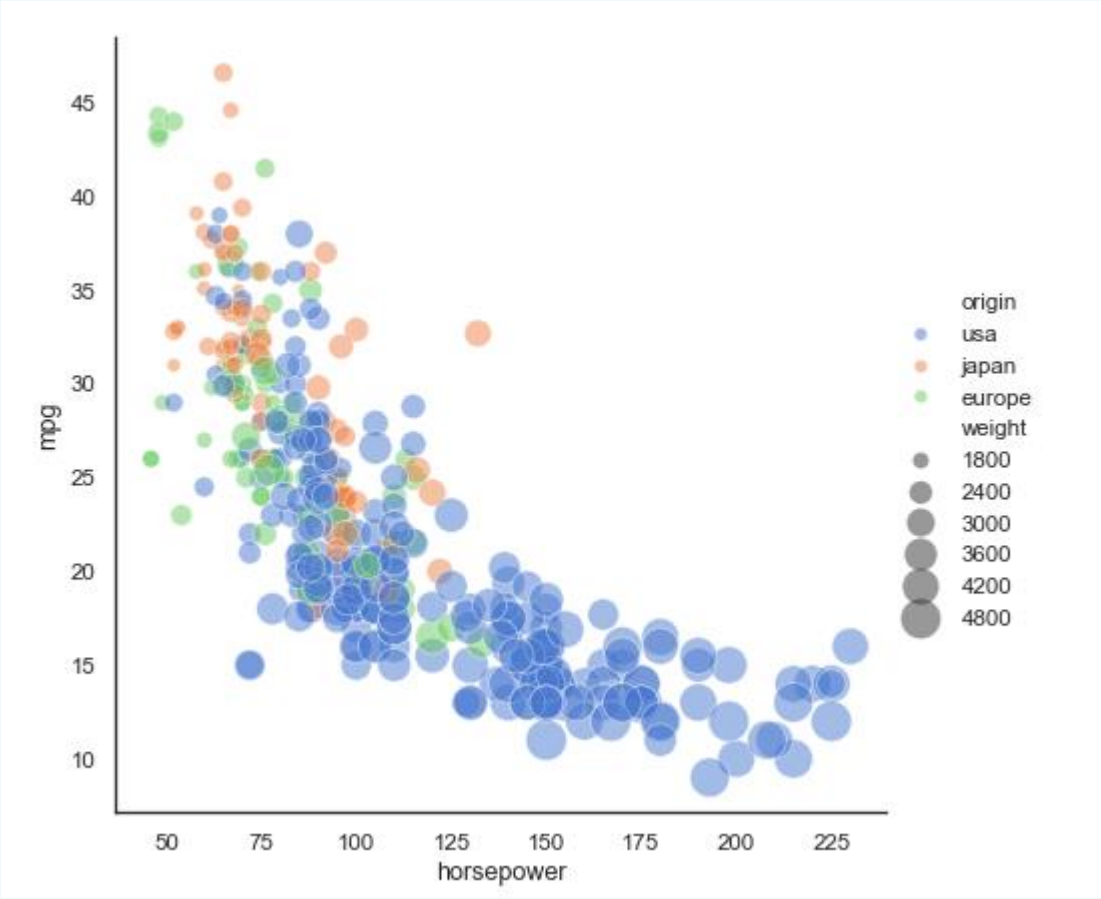


Figure de www.seaborn.pydata.org

Introduction aux séries temporelles

Exemple de données

- Series ou DataFrame avec un indice temporel
- Pas de temps constant ou non
- Exemple: capteur de température in situ dans l'océan

	Temperature
2019-04-13 14:00:00	28.4728
2019-04-13 14:00:10	28.4780
2019-04-13 14:00:20	28.4818
2019-04-13 14:00:30	28.5127
2019-04-13 14:00:40	28.5199
...	...
2020-10-02 03:19:09	28.6340
2020-10-02 03:19:19	28.6339
2020-10-02 03:19:29	28.6339
2020-10-02 03:19:39	28.6341
2020-10-02 03:19:49	28.6339

Datetime Index: utiliser pd.to_datetime()

- On utilise pd.to_datetime(string) pour créer un timestamp

```
pd.to_datetime('2012-01-25')
```

```
Timestamp('2012-01-25 00:00:00')
```

- Idem avec une list de string pour un DatetimeIndex:

```
pd.to_datetime(['2012','2013','2014'])
```

```
DatetimeIndex(['2012-01-01', '2013-01-01', '2014-01-01'], dtype='datetime64[ns]', freq=None)
```

Datetime Index: à partir de liste de données

Sample Number	Date	Time	Temperature
1	13-04-2019	14:00:00	28.4728
2	13-04-2019	14:00:10	28.4780
3	13-04-2019	14:00:20	28.4818
4	13-04-2019	14:00:30	28.5127
5	13-04-2019	14:00:40	28.5199

- Créer une liste avec les informations de dates

```
list_date = df.Date + ' ' + df.Time
```

```
Sample Number
1          13-04-2019 14:00:00
2          13-04-2019 14:00:10
3          13-04-2019 14:00:20
4          13-04-2019 14:00:30
5          13-04-2019 14:00:40
```

- Utiliser pd.to_datetime

```
pd.to_datetime(list_date, infer_datetime_format=True)
```

```
Sample Number
1          2019-04-13 14:00:00
2          2019-04-13 14:00:10
3          2019-04-13 14:00:20
4          2019-04-13 14:00:30
5          2019-04-13 14:00:40
```

- Changer l'indice

```
df.index =  pd.to_datetime(df.Date + ' ' + df.Time,
                           infer_datetime_format=True)
df
```

	Date	Time	Temperature
2019-04-13 14:00:00	13-04-2019	14:00:00	28.4728
2019-04-13 14:00:10	13-04-2019	14:00:10	28.4780
2019-04-13 14:00:20	13-04-2019	14:00:20	28.4818
2019-04-13 14:00:30	13-04-2019	14:00:30	28.5127
2019-04-13 14:00:40	13-04-2019	14:00:40	28.5199

Datetime Index: créer un indice

- On utilise pd.date_range()

```
pd.date_range(start='2012-01-01', end='2013-12-25', freq='m')
```

```
DatetimeIndex(['2012-01-31', '2012-02-29', '2012-03-31', '2012-04-30',
               '2012-05-31', '2012-06-30', '2012-07-31', '2012-08-31',
               '2012-09-30', '2012-10-31', '2012-11-30', '2012-12-31',
               '2013-01-31', '2013-02-28', '2013-03-31', '2013-04-30',
               '2013-05-31', '2013-06-30', '2013-07-31', '2013-08-31',
               '2013-09-30', '2013-10-31', '2013-11-30'],
              dtype='datetime64[ns]', freq='M')
```

```
pd.date_range(start='2012-01-01 12:00:00', periods=20, freq='10s')
```

```
DatetimeIndex(['2012-01-01 12:00:00', '2012-01-01 12:00:10',
               '2012-01-01 12:00:20', '2012-01-01 12:00:30',
               '2012-01-01 12:00:40', '2012-01-01 12:00:50',
               '2012-01-01 12:01:00', '2012-01-01 12:01:10',
               '2012-01-01 12:01:20', '2012-01-01 12:01:30',
               '2012-01-01 12:01:40', '2012-01-01 12:01:50',
               '2012-01-01 12:02:00', '2012-01-01 12:02:10',
               '2012-01-01 12:02:20', '2012-01-01 12:02:30',
               '2012-01-01 12:02:40', '2012-01-01 12:02:50',
               '2012-01-01 12:03:00', '2012-01-01 12:03:10'],
              dtype='datetime64[ns]', freq='10S')
```

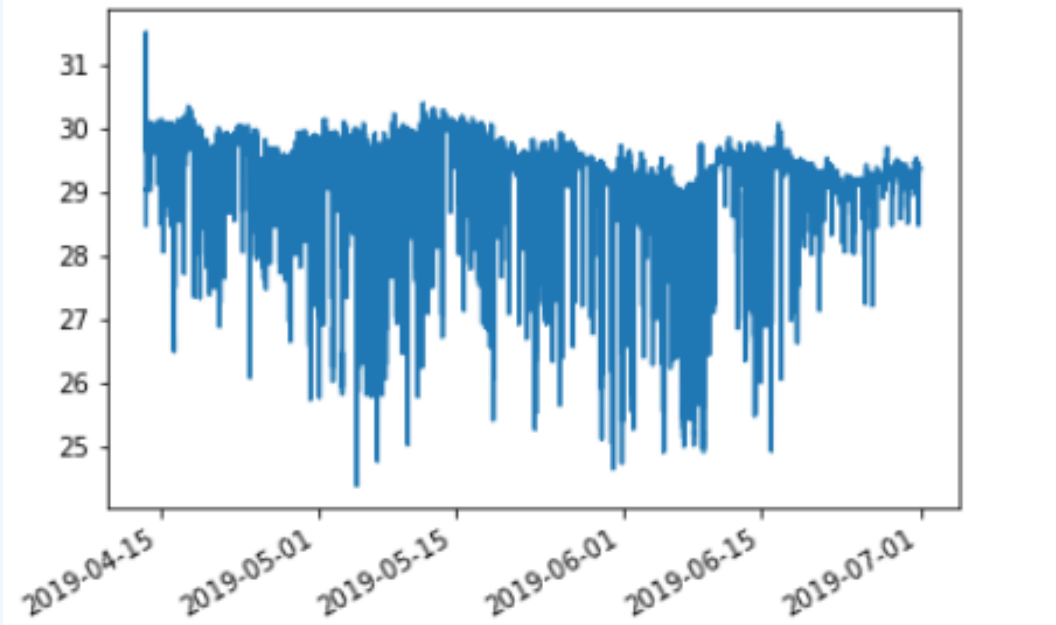
- Fréquences?
  - y, q, m, w, d, h, min, s, ms, us, ns
  - On peut ajouter un nombre: 10h

Afficher la série temporelle

- Visualisation rapide

```
df.Temperature.plot()
```

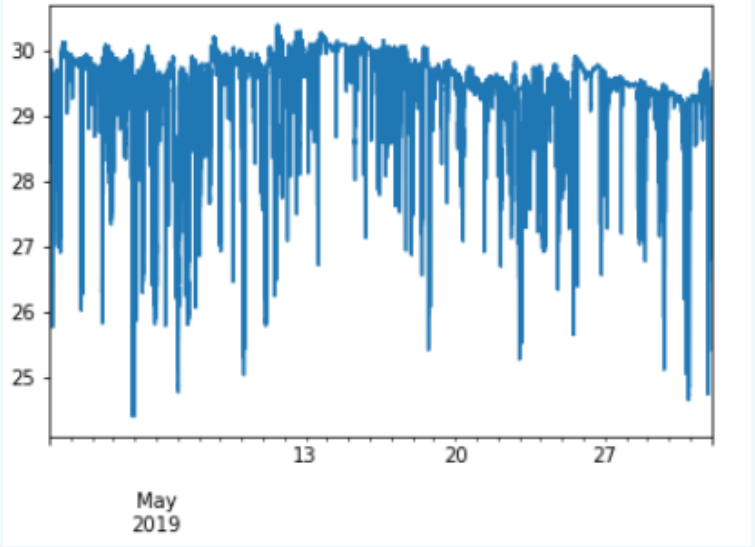
```
<AxesSubplot:>
```



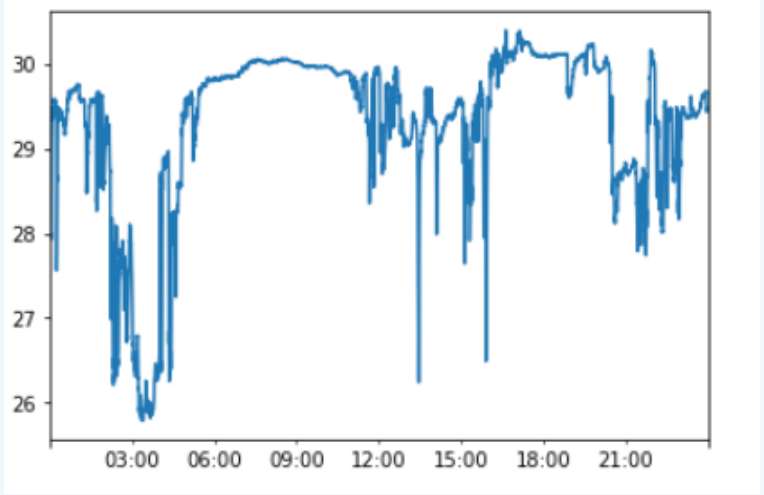
Sélection de données par le temps

- On utilise df.loc[time] pour un moment précis

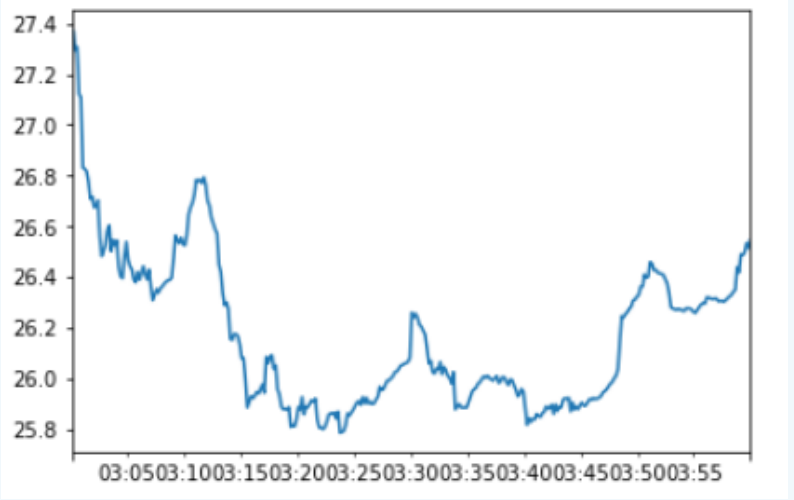
```
df.Temperature.loc['2019-05'].plot()
<AxesSubplot:>
```




```
df.Temperature.loc['2019-05-11'].plot()
<AxesSubplot:>
```




```
df.Temperature.loc['2019-05-11 3H'].plot()
<AxesSubplot:>
```

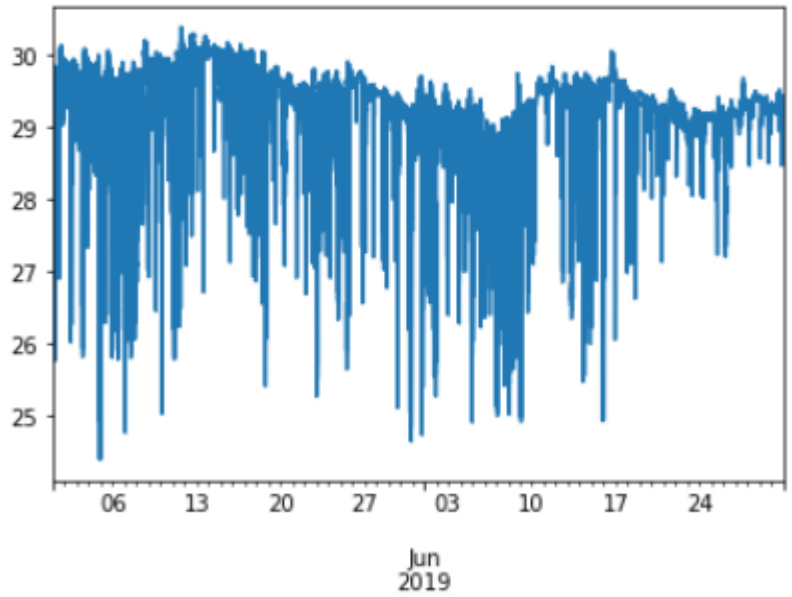


Sélection de données par le temps

- On utilise df.loc[time1 : time2] pour les données entre 1 et 2

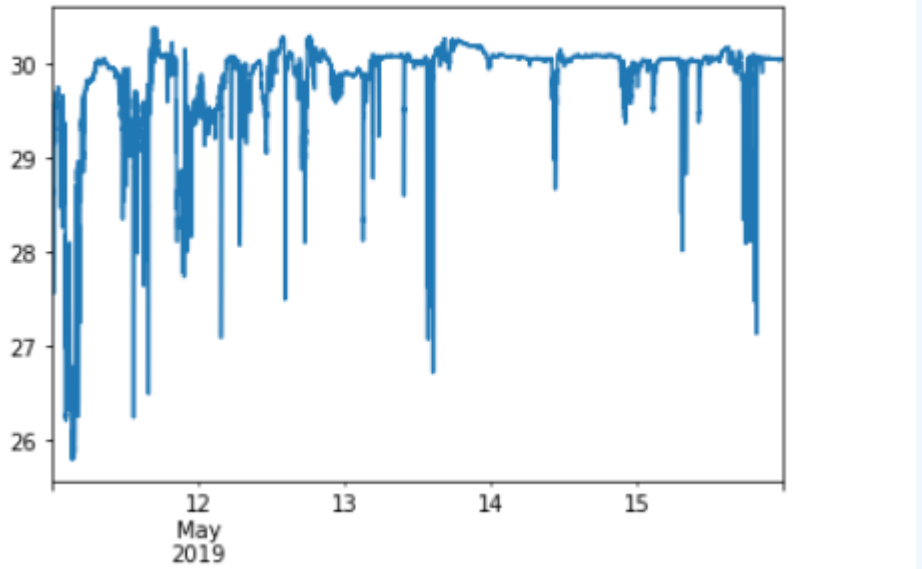
```
df.Temperature.loc['2019-05':'2019-09'].plot()
```

```
<AxesSubplot:>
```




```
df.Temperature.loc['2019-05-11':'2019-05-15'].plot()
```

```
<AxesSubplot:>
```



Rééchantillonage

- En utilisant les mêmes fréquences que pour date_range:

```
df.Temperature.resample('H')
```

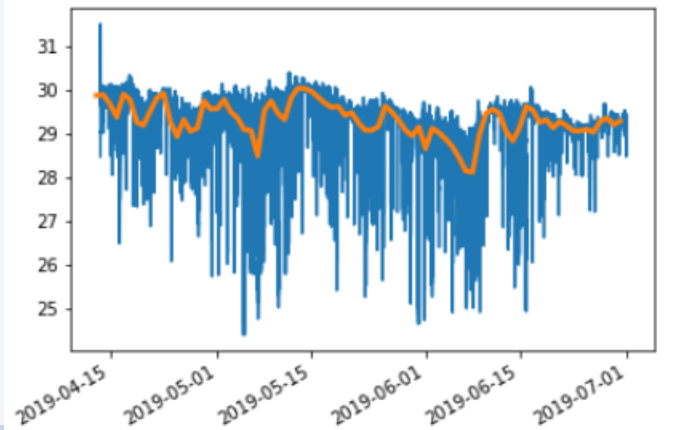
- Renvoie un objet comme groupby: attend une opération
  - Mean, max, min, median, quantile, count, … ou apply()

```
df.Temperature.resample('D').mean()
```

```
2019-04-13    29.869072
2019-04-14    29.883743
2019-04-15    29.671858
2019-04-16    29.371685
2019-04-17    29.896633
```

```
df.Temperature.plot()
df.Temperature.resample('D').mean().plot(lw=3)
```

```
<AxesSubplot:>
```



Opérations en fenêtres glissantes

- Intérêt: garde le même taux d'échantillonage, regarde ce qu'il se passe autour
- Renvoie un objet comme groupby: attend une opération
  - Mean, max, min, median, quantile, count, … ou apply()
- Dataframe.rolling(taille_fenetre)

```
df.Temperature.rolling(5, center=True).mean()
```

```
2019-04-13 14:00:00         NaN
2019-04-13 14:00:10         NaN
2019-04-13 14:00:20    28.49304
2019-04-13 14:00:30    28.50608
2019-04-13 14:00:40    28.52274
```

Positionne l'indice du résultat: fin ou centre

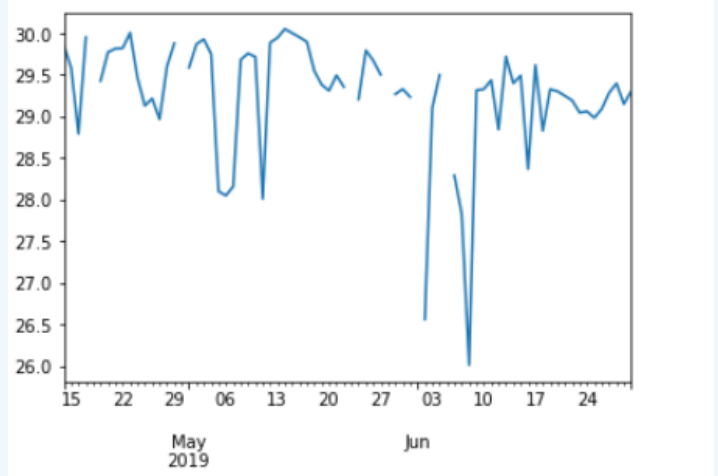
Gestion de données manquantes/irrégulières

	Temperature
2019-04-14 01:04:09	29.8508
2019-04-14 16:16:29	29.8557
2019-04-15 05:10:19	30.0335
2019-04-15 07:51:29	29.4874
2019-04-15 14:08:49	29.8838

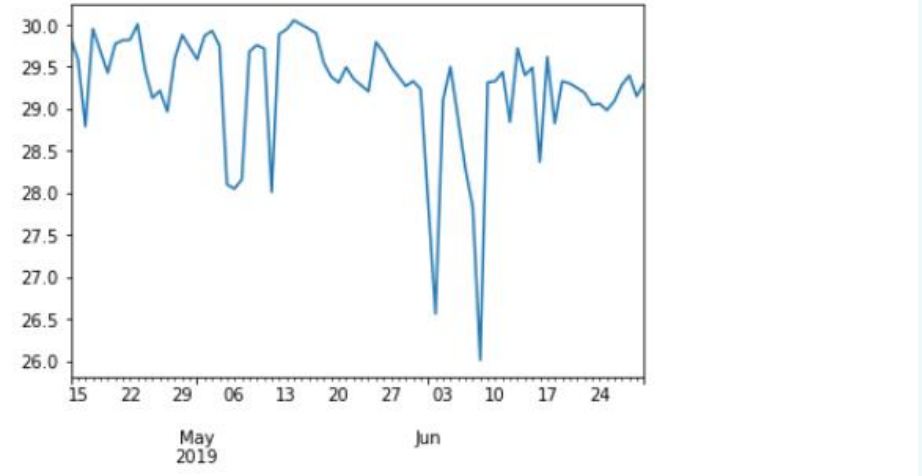
- Obtenir des données journalières?

series.interpolate( )

```
df_irregulier.Temperature.resample('D').mean().plot()
<AxesSubplot:>
```




```
df_irregulier.Temperature.resample('D').mean().interpolate().plot()
<AxesSubplot:>
```



Accéder aux information de dates

- De nombreux attributs disponibles sur un DatetimeIndex
  - Month, year, dayofweek, is_month_start, is_leap_year, ...

```
df.index.month
```

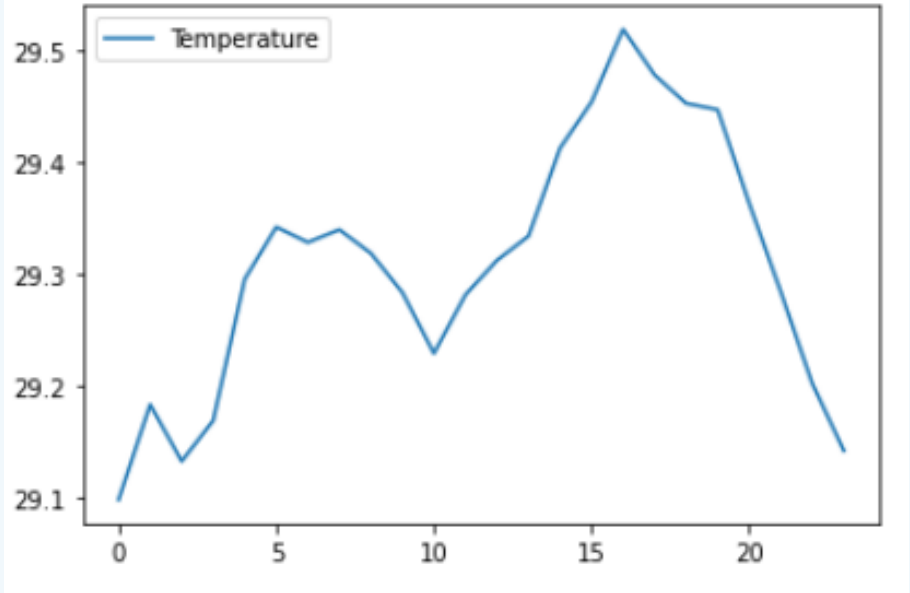
```
Int64Index([13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
            ...
            30, 30, 30, 30, 30, 30, 30, 30, 30, 30],
           dtype='int64', length=677521)
```

Groupby avec des données temporelles

- Cycle journalier de la température

```
df.groupby(df.index.hour).mean().plot()
```

```
<AxesSubplot:>
```



Résumé : séries temporelles

series/dataframe avec indice temporelle (numpy datetime/pandas date_range, …)

Quelques fonctionnalités disponibles :

Sélection temporelle :	df.loc["2012-10-04"]
Sélection d'une période :	df.loc["2008-07" : "2015-07"]
Rééchantillonage :	df.resample("D").mean()
Fenêtres glissantes :	df.rolling(5).mean()
Remplir les NaN :	df.interpolate( )
Informations temporelles :	df.index.hour / df.index.day_of_year / …

Des questions?