

# PYTHON – PANDAS - DATE

## SOMMAIRE

|   |          |
|---|----------|
| <b>Sommaire .....</b>   | <b>1</b> |
| <b>PANDAS – DATES.....</b>  | <b>2</b> |
| <b>Dates et séries temporelle en Pandas .....</b>                             | <b>2</b> |
| Principes numpy .....   | 2        |
| 2 types : DateTime64 et TimeDelta64 .....                                     | 2        |
| Création d'un objet datetime64 .....  | 2        |
| Création d'un objet timedelta64 .....   | 2        |
| Principes Pandas .....  | 3        |
| 3 objets Pandas : timestamp .....   | 3        |
| La fonction pd.to_datetime .....  | 3        |
| La fonction pd.date_range() .....   | 3        |
| Créer une Série avec une date comme index .....                               | 4        |
| Rechercher des dates dans une série avec une date : pratique !.....           | 4        |
| Regroupement par la fonction resample() : pratique ! proche du group_by ..... | 4        |

**Edition : août 2022**

# PANDAS – DATES

## Dates et séries temporelle en Pandas

### Principes numpy

#### 2 types : DateTime64 et TimeDelta64

- Les 2 sur 64 bit
- On précise la résolution temporelle voulue : année, jour, seconde, milliseconde, etc : l'encodage gère ensuite pour coder le maximum de dates sur 64 bits.
- Exemple : avec une résolution de nano-secondes, on va de 1678 à 2262. En millisecondes, on couvre 600 millions d'années.

#### Création d'un objet datetime64

- On utilise un format ISO 86 01 pour la date (ce n'est pas important)

```
np.datetime64('2018-06-30')  
np.datetime64('2018-06-30 08:35:23')  
np.datetime64('2018-06-30 08:35:23', 'ns') # ns pour nanoseconde
```

#### Création d'un objet timedelta64

- Obtenu automatiquement quand on fait une différence entre 2 dates :

```
np.datetime64('2018-06-30 08:35:23') - np.datetime64('2018-06-20  
08:37:23')
```

## Principes Pandas

[https://pandas.pydata.org/pandas-docs/stable/user\\_guide/timeseries.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html)

### 3 objets Pandas : timestamp

- En Pandas, on n'utilise pas les objets numpy mais 3 objets Pandas :
  - 1) [Timestamp](#) : les dates
  - 2) [Period](#) : une date associée à une durée
  - 3) [Timedelta](#) : les intervalles

⇒ Le parsing est plus flexible que l'ISO 8601

⇒ On pourra créer des index : un objet Timestamp a un index associé : DatetimeIndex (une sorte de Série de Timestamp)

### La fonction pd.to\_datetime

- La fonction `to_datetime()` est très pratique (un peu magique !).
- Avec une date, elle parse à peu près n'importe quel format pour fournir un Timestamp
- Avec plusieurs dates, elle fournit un index automatiquement : un DatetimeIndex

```
pd.to_datetime('10 june 1973 8h30')
Timestamp('1973-06-10 08:30:00')
```

```
pd.to_datetime(['10 june 1973 8h30', '22-JUNE-1973'])
DatetimeIndex(['1973-06-10 08:30:00', '1973-06-22 00:00:00'],
              dtype='datetime64[ns]', freq=None)
```

### La fonction pd.date\_range()

La fonction `date_range()` permet de fournir des séries de dates :

```
index = pd.date_range('1 jan 2018', periods=1000, freq='D')
# 1000 jours à partir du 1 janvier 2018
index
df = pd.DataFrame({
    'dates': index,
})
```

```
index = pd.date_range('1 jan 2018', periods=1000,
                      freq='43h36min')
index = pd.date_range('1 jan 2018', periods=1000, freq='43h36t')
index
# t c'est comme min
```

### **Créer une Série avec une date comme index**

```
index = pd.date_range('1 jan 2018', periods=1000, freq='D')
s = pd.Series(np.random.randint(100, size=1000), index=index)
s
```

### **Rechercher des dates dans une série avec une date : pratique !**

```
s['2018']
```

```
s['dec2018']
```

```
s['dec 2018':'3 jan 2019'] # slicing !
```

### **Regroupement par la fonction resample() : pratique ! proche du group by**

```
s.resample('M').mean() # par mois
```

```
s.resample('W-WED').mean() # par semaine à partir du mercredi
```

```
s.resample? # documentation à regarder
```