# Nettoyer les données

## Semaine 5

### Ali Assi, PhD

Que signifie "nettoyer les données" ? En voici une liste non exhaustive :

- renommer les colonnes
- renommer l'index
- supprimer les colonnes non pertinentes
- diviser une colonne en deux
- combiner deux colonnes ou plus en une seule
- supprimer les lignes sans données
- supprimer les lignes répétées
- supprimer les lignes contenant des données manquantes (NaN)
- remplacer les données NaN par une valeur unique
- remplacer les données NaN par interpolation
- normaliser les chaînes de caractères
- corriger les fautes de frappe dans les chaînes de caractères
- supprimer les espaces dans les chaînes de caractères
- corriger les types utilisés pour les colonnes
- identifier et supprimer les outliers

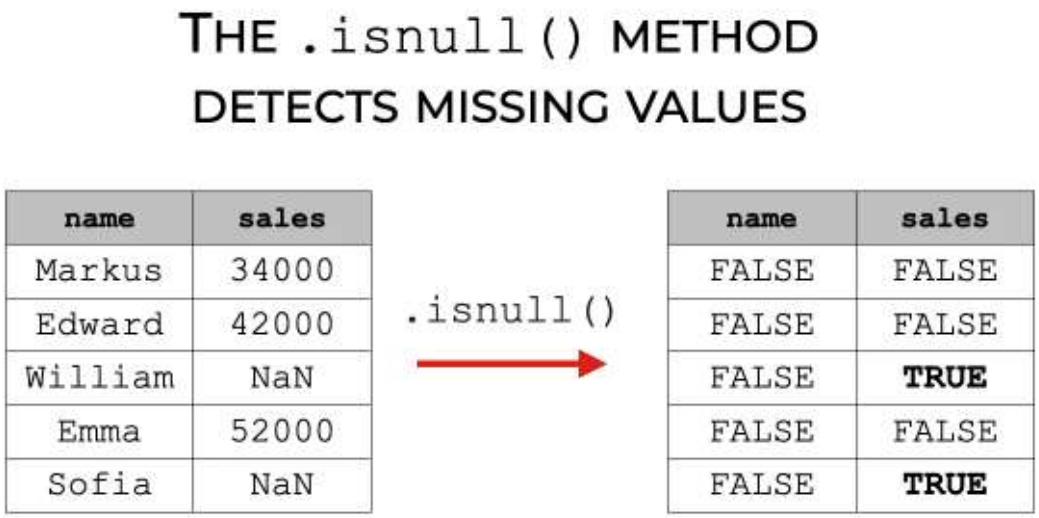
Nous avons déjà abordé certaines de ces techniques dans les chapitres précédents. Mais on souligne l'importance du nettoyage des données qui permet de produire une analyse aussi précise que possible.

Dans ce chapitre, nous allons donc étudier quelques techniques Pandas pour nettoyer nos données. Nous verrons quelques façons de traiter les valeurs `NaN` . Nous verrons comment préserver autant de données que possible, même lorsqu'elles sont assez bruitées. Nous verrons comment mieux comprendre nos données et leurs limitations. Enfin, nous étudierons quelques techniques plus avancées pour transformer nos données en une forme plus facile à analyser.

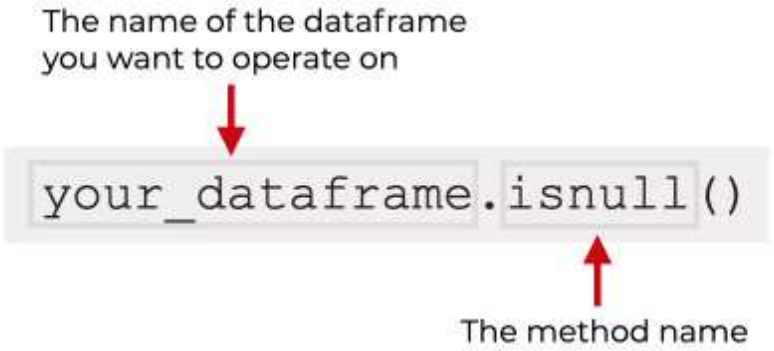
# Quelques références utiles

1. `len(df)` ou `len(df.index)` , Obtenir le nombre de lignes dans un dataframe.

2. `df.isnull()` , détecte les valeurs manquantes dans un dataframe.




ou




ou bien


In [1]:
```
import pandas as pd
import numpy as np

sales_data = pd.DataFrame({"name":["William","Emma","Sofia","Markus","Edward","Th
,"region":["East",np.nan,"East","South","West","West","South","West","West","East
,"sales":[50000,52000,90000,np.nan,42000,72000,49000,np.nan,67000,65000,67000]
,"expenses":[42000,43000,np.nan,44000,38000,39000,42000,np.nan,39000,44000,45000]
```

In [3]:
```
# identify the missing values in an entire dataframe
sales_data
```

Out[3]:

| | name | region | sales | expenses |
|---|---|---|---|---|
| 0 | William | East | 50000.0 | 42000.0 |
| 1 | Emma | NaN | 52000.0 | 43000.0 |
| 2 | Sofia | East | 90000.0 | NaN |
| 3 | Markus | South | NaN | 44000.0 |
| 4 | Edward | West | 42000.0 | 38000.0 |
| 5 | Thomas | West | 72000.0 | 39000.0 |
| 6 | Ethan | South | 49000.0 | 42000.0 |
| 7 | Olivia | West | NaN | NaN |
| 8 | Arun | West | 67000.0 | 39000.0 |
| 9 | Anika | East | 65000.0 | 44000.0 |
| 10 | Paulo | South | 67000.0 | 45000.0 |

In [4]:
```
sales_data.isnull()
```

Out[4]:

| | name | region | sales | expenses |
|---|---|---|---|---|
| 0 | False | False | False | False |
| 1 | False | True | False | False |
| 2 | False | False | False | True |
| 3 | False | False | True | False |
| 4 | False | False | False | False |
| 5 | False | False | False | False |
| 6 | False | False | False | False |
| 7 | False | False | True | True |
| 8 | False | False | False | False |
| 9 | False | False | False | False |
| 10 | False | False | False | False |

In [5]:
```
# find missing values in a pandas dataframe column
sales_data.sales.isnull()
```

Out[5]:
```
0     False
1     False
2     False
3      True
4     False
5     False
6     False
7      True
8     False
9     False
10    False
Name: sales, dtype: bool
```

3. `df.replace()` , Remplacer les valeurs d'une ou plusieurs colonnes par d'autres valeurs

In [2]:
```
df = pd.DataFrame({'A': [0, 1, 2, 3, 4],
                   'B': [5, 6, 7, 8, 9],
                   'C': ['a', 'b', 'c', 'd', 'e']})
df
```

Out[2]:

| | A | B | C |
|---|---|---|---|
| **0** | 0 | 5 | a |
| **1** | 1 | 6 | b |
| **2** | 2 | 7 | c |
| **3** | 3 | 8 | d |
| **4** | 4 | 9 | e |

In [4]:
```
df2 = df.replace(0, 5)
df2
```

Out[4]:

| | A | B | C |
|---|---|---|---|
| **0** | 0 | 5 | a |
| **1** | 1 | 6 | b |
| **2** | 2 | 7 | c |
| **3** | 3 | 8 | d |
| **4** | 4 | 9 | e |

In [8]:
```
df = pd.DataFrame({'A': [0, 1, 2, 3, 4],
                   'B': [5, 6, 7, 8, 9],
                   'C': ['a', 'b', 'c', 'd', 'e']})
df
```

Out[8]:

| | A | B | C |
|---|---|---|---|
| **0** | 0 | 5 | a |
| **1** | 1 | 6 | b |
| **2** | 2 | 7 | c |
| **3** | 3 | 8 | d |
| **4** | 4 | 9 | e |

In [9]:
```
df.replace(0, 5, inplace = True)
df
```

Out[9]:

| | A | B | C |
|---|---|---|---|
| **0** | 5 | 5 | a |
| **1** | 1 | 6 | b |
| **2** | 2 | 7 | c |
| **3** | 3 | 8 | d |
| **4** | 4 | 9 | e |

4. `df['colname'].str` , Travailler avec des données textuelles

In [5]:
```
s = pd.Series(["A", "B", "C", "Aaba", "Baca", "CABA", "dog", "cat"], dtype="strin
s
```

Out[5]:
```
0       A
1       B
2       C
3    Aaba
4    Baca
5    CABA
6     dog
7     cat
dtype: string
```

In [16]:
```
s2 = s.str.lower()
s2
```

Out[16]:
```
0       a
1       b
2       c
3    aaba
4    baca
5    caba
6     dog
7     cat
dtype: string
```

In [17]:
```
s2 = s.str.upper()
s2
```

Out[17]:
```
0       A
1       B
2       C
3    AABA
4    BACA
5    CABA
6     DOG
7     CAT
dtype: string
```

In [18]:
```
s2 = s.str.isdigit()
s2
```

Out[18]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6    False
7    False
dtype: boolean
```

5. `df.sort_index` , Réorganiser les lignes d'un dataframe en fonction des valeurs de son index, dans l'ordre croissant

In [7]:
```
df = pd.DataFrame([1, 2, 3, 4, 5], index = [100, 29, 234, 1, 150], columns = ['A'
df
```

Out[7]:

| | A |
|---|---|
| **100** | 1 |
| **29** | 2 |
| **234** | 3 |
| **1** | 4 |
| **150** | 5 |

In [8]:
```
df2 = df.sort_index()
df2
```

Out[8]:

| | A |
|---|---|
| **1** | 4 |
| **29** | 2 |
| **100** | 1 |
| **150** | 5 |
| **234** | 3 |

In [10]:
```
df.sort_index(inplace = True, ascending = False)
df
```

Out[10]:

| | A |
|---|---|
| **234** | 3 |
| **150** | 5 |
| **100** | 1 |
| **29** | 2 |
| **1** | 4 |

In [22]:
```
df.sort_index(inplace = True)
df
```

Out[22]:

| | A |
|---|---|
| **1** | 4 |
| **29** | 2 |
| **100** | 1 |
| **150** | 5 |
| **234** | 3 |

6. `unique()` , Renvoie une série contenant les valeurs uniques (c'est-à-dire distinctes) dans une série ou un dataframe, y compris `NaN` (s'il apparaît dans la série ou le dataframe)

In [11]:
```
data = {'name': ['Sheldon', 'Penny', 'Amy', 'Penny', 'Raj', 'Sheldon'],
            'year': [2012, 2012, 2013, 2014, 2014,2012 ],
            'episodes': [42, 24, 31, 29, 37, 40]}

df = pd.DataFrame(data, index = ['a', 'b', 'c', 'd', 'e','f'])

df
```

Out[11]:

| | name | year | episodes |
|---|---|---|---|
| **a** | Sheldon | 2012 | 42 |
| **b** | Penny | 2012 | 24 |
| **c** | Amy | 2013 | 31 |
| **d** | Penny | 2014 | 29 |
| **e** | Raj | 2014 | 37 |
| **f** | Sheldon | 2012 | 40 |

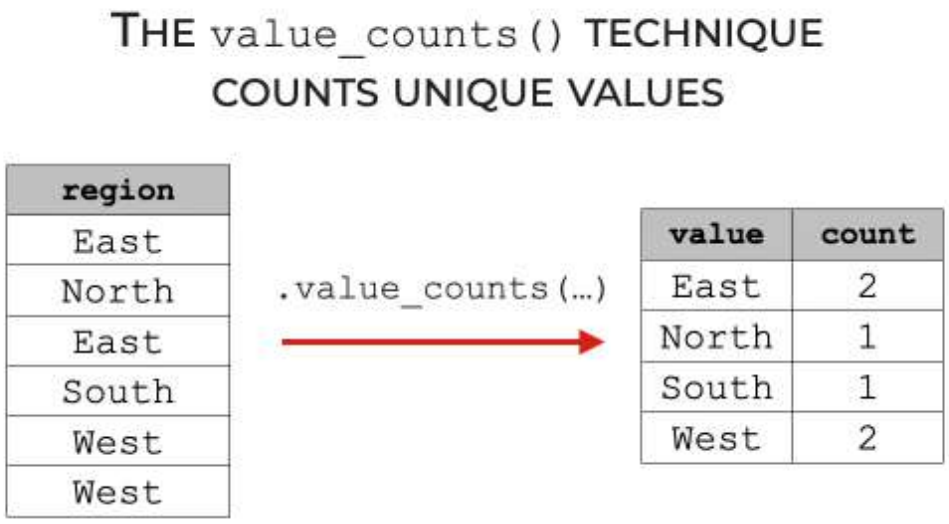
In [12]:
```
df.name.unique()
```

Out[12]:
```
array(['Sheldon', 'Penny', 'Amy', 'Raj'], dtype=object)
```

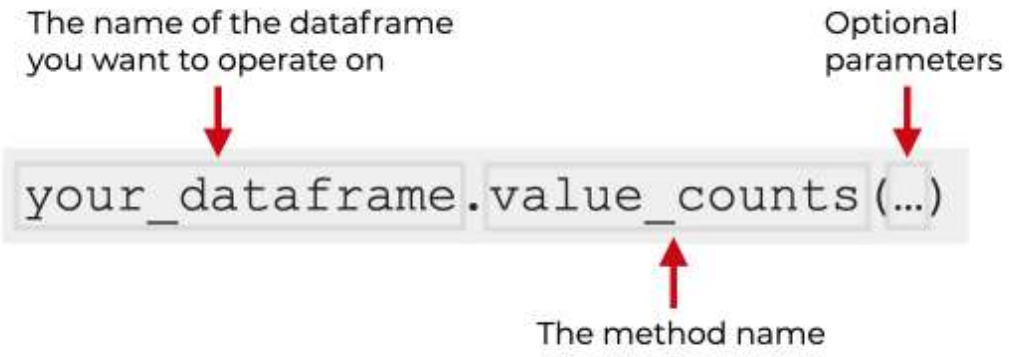
In [26]:
```
pd.unique(df['name'])
```

Out[26]:
```
array(['Sheldon', 'Penny', 'Amy', 'Raj'], dtype=object)
```

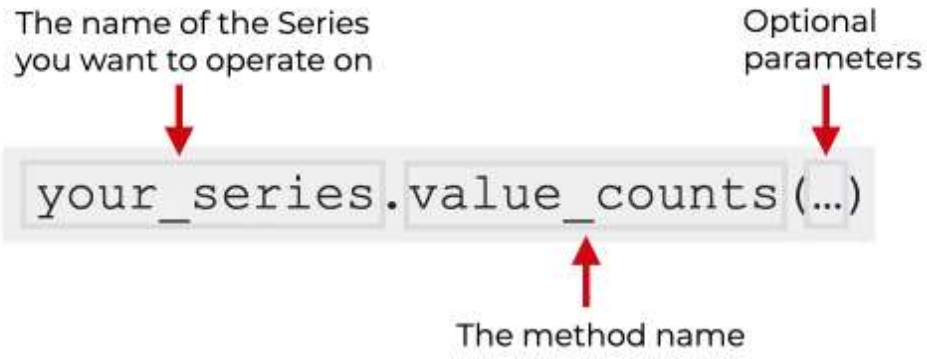
7. `s.value_counts` , renvoie une série triée (fréquence décroissante) comptant combien de fois chaque valeur apparaît dans la série `s`




ou




ou bien



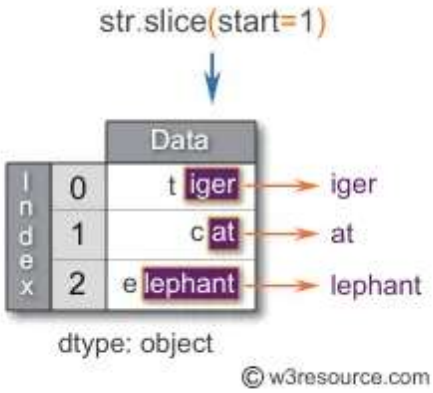

In [13]:
```
#define series
s = pd.Series(['Lahore','Murree','Islamabad','Karachi','Lahore','Faislabad','Isla

#print series
s
```

Out[13]:
```
0       Lahore
1       Murree
2    Islamabad
3      Karachi
4       Lahore
5    Faislabad
6    Islamabad
dtype: object
```

In [14]:
```
s.value_counts()
```

Out[14]:
```
Lahore       2
Islamabad    2
Murree       1
Karachi      1
Faislabad    1
dtype: int64
```

8. La méthode `str.slice()` de Pandas possède un certain nombre de variantes qui vous permettent de découper une colonne de chaînes de différentes manières.




Découpage en une plage de caractères : Vous pouvez découper une colonne de chaînes en une plage de caractères comprise entre les indices `start` et `stop` en utilisant la méthode `str.slice(start, stop)` . Par exemple :

In [19]:
```
import pandas as pd

# Create a sample DataFrame with a string column
data = {'Name': ['John Smith', 'Jane Doe', 'Mark Johnson'], 'Age': [30, 25, 40]}
df = pd.DataFrame(data)
df
```

Out[19]:

| | Name | Age |
|---|---|---|
| **0** | Johnaaaaaaaa Smith | 30 |
| **1** | Jane Doe | 25 |
| **2** | Mark Johnson | 40 |

In [20]:
```
# Slice the 'Name' column to the first 5 characters
df['Name'] = df['Name'].str.slice(0, 5)

# Display the updated DataFrame
df
```

Out[20]:

| | Name | Age |
|---|---|---|
| **0** | Johna | 30 |
| **1** | Jane | 25 |
| **2** | Mark | 40 |

Découpage à partir d'un point de départ spécifique : vous pouvez utiliser la méthode `str.slice(start)` pour découper une colonne de chaînes à partir d'un indice de départ spécifique. Par exemple

In [30]:
```
import pandas as pd

# Create a sample DataFrame with a string column
data = {'Name': ['John Smith', 'Jane Doe', 'Mark Johnson'], 'Age': [30, 25, 40]}
df = pd.DataFrame(data)

# Slice the 'Name' column starting from the 5th character
df['Name'] = df['Name'].str.slice(5)

# Display the updated DataFrame
df
```

Out[30]:

| | Name | Age |
|---|---|---|
| **0** | Smith | 30 |
| **1** | Doe | 25 |
| **2** | Johnson | 40 |

Découpe jusqu'à un point d'arrêt spécifique à partir de la fin de la chaîne : Vous pouvez utiliser la méthode `str.slice(stop)` pour découper une colonne de chaîne jusqu'à un indice d'arrêt spécifique à partir de la fin de la chaîne. Par exemple :

In [4]:
```
import pandas as pd

# Create a sample DataFrame with a string column
data = {'Name': ['John Smith', 'Jane Doe', 'Mark Johnson'], 'Age': [30, 25, 40]}
df = pd.DataFrame(data)

# Slice the 'Name' column up to the 5th character from the end of the string
df['Name'] = df['Name'].str.slice(-5)

# Display the updated DataFrame
df
```

Out[4]:

| | Name | Age |
|---|---|---|
| **0** | Smith | 30 |
| **1** | e Doe | 25 |
| **2** | hnson | 40 |

# Exercice

Précédemment, nous avons examiné les contraventions de stationnement données à New York au cours de l'année 2020. Nous avons pu analyser ces données et en tirer des conclusions intéressantes. Mais considérons que ces données sont saisies par un officier de police, un inspecteur du stationnement ou une autre personne, ce qui signifie qu'il y a de fortes chances qu'il y ait parfois des données manquantes ou incorrectes. Cela peut sembler un problème mineur, mais cela peut signifier beaucoup de choses :

1. des voitures qui reçoivent des contraventions de manière incorrecte,
2. des statistiques erronées dans le système,
3. des personnes qui qui sont exemptées d'amendes en raison d'informations incorrectes, etc.

Dans cet exercice, nous allons identifier les valeurs manquantes, l'un des problèmes les plus courants que vous rencontrerez. Nous allons voir quelle est la fréquence des valeurs manquantes et quel effet elles peuvent avoir. Notez que pour les besoins de cet exercice, je vais supposer qu'une contravention de stationnement pour laquelle il manque des données pourrait être rejetée.

**1. Créez un dataframe à partir du fichier nyc-parking-violations-2020.csv. Seule une partie de colonnes nous intéresse :**

- Plate ID
- Registration State
- Vehicle Make
- Vehicle Color
- Violation Time
- Street Name

In [21]:
```
filename = './data/nyc-parking-violations-2020.csv'

df = pd.read_csv(filename,
                 usecols = ['Plate ID',  'Registration State',
                        'Vehicle Make', 'Vehicle Color', 'Violation Time', 'Stree

df.head()
```

Out[21]:

| | Plate ID | Registration State | Vehicle Make | Violation Time | Street Name | Vehicle Color |
|---|---|---|---|---|---|---|
| **0** | J58JKX | NJ | HONDA | 0523P | 43 ST | BK |
| **1** | KRE6058 | PA | ME/BE | 0428P | UNION ST | BLK |
| **2** | 444326R | NJ | LEXUS | 0625A | CLERMONT AVENUE | BLACK |
| **3** | F728330 | OH | CHEVR | 1106A | DIVISION AVE | NaN |
| **4** | FMY9090 | NY | JEEP | 1253A | GRAND ST | GREY |

**2. Combien de lignes contient le dataframe lorsqu'il est stocké en mémoire ?**

**3. Supprimez les lignes contenant des données manquantes (c'est-à-dire une valeur `NaN`). Combien de lignes reste-t-il après cet ajustement ?**

**4. Combien de contraventions de stationnement comportent des informations manquantes ?**

**5. Si chaque ticket de parking rapporte 100 dollars à la ville et que les données manquantes signifient que le ticket peut être contesté avec succès, combien d'argent la ville de New York pourrait-elle perdre en raison de ces données manquantes ?**

**6. Supposons plutôt qu'une contravention ne peut être rejetée que si la plaque d'immatriculation, l'État, la marque du véhicule et/ou le nom de la rue sont manquants. Supprimez les lignes auxquelles il manque un ou plusieurs de ces éléments. Combien de lignes reste-t-il ?**

**7. Combien de lignes avons-nous supprimées ?**

**8. En supposant que la contravention coûte $100\$$, combien d'argent la ville perdrait-elle en raison de ces données manquantes ?**

**9. Supposons maintenant que les contraventions peuvent être annulées s'il manque la plaque d'immatriculation, l'État et/ou le nom de la rue - c'est-à-dire la même chose que pour la question précédente, mais sans exiger la marque de la voiture. Supprimez les lignes auxquelles il manque un ou plusieurs de ces éléments. Combien de lignes reste-t-il ?**

**10. Combien de lignes avons-nous supprimées ?**

**11. En supposant que le ticket soit de $100\$$, combien d'argent la ville perdrait-elle en raison de ces données manquantes ?**

# Exercice

Pour cet exercice, nous allons examiner un dataset, une liste de célébrités décédées en 2016 et dont le décès a été enregistré dans Wikipedia - y compris la date du décès, une courte biographie et la cause du décès. Le problème est que cet ensemble de données est désordonné, avec des données manquantes et des données erronées qui nous empêcheront de travailler facilement avec ces données comme nous le souhaiterions.

L'objectif de cet exercice est de trouver l'âge moyen des célébrités décédées entre février et juillet 2016. Pour y répondre, un certain nombre d'étapes sont nécessaires :

**1. Créez un dataframe à partir du fichier celebrity_deaths_2016.csv. Pour cet exercice, nous n'utiliserons que deux colonnes :**

- dateofdeath

In [ ]:

**2. Créer une nouvelle colonne `month` , contenant le mois de la colonne `dateofdeath` .**

In [ ]:

**3. Utiliser la colonne `month` comme index du dataframe**

In [ ]:

**4. Trier le dataframe par l'index**

In [ ]:

**5. Supprimer tous les nombres non entiers de la colonne de l'âge. Indication: utiliser les méthodes `dropna()` et `str.isdigit()` . La méthode `isdigit` permet de déterminer quelles lignes d'une colonne peuvent être transformées en nombres entiers.**

In [ ]:

**6. La fonction `str.isdigit()` renvoie `True` si une chaîne ne contient que des chiffres (et n'est pas vide). (Elle renverra `False` s'il y a un signe `-` ou un point décimal, elle n'est donc pas fiable de trouver des nombres, mais elle fonctionnera avec des `ages` ). Pandas propose une solution plus élégante, à savoir la fonction `pd.to_numeric` . Cette fonction tente de créer une nouvelle série avec des valeurs numériques. La fonction essaie de transformer les valeurs en entiers, mais si elle n'y parvient pas, elle renvoie des flottants. Aussi elle lèvera une exception s'il rencontre une chaîne de caractères qui ne peut pas être transformée en `int` ou en `float` . Mais si nous passons le mot-clé `errors='coerce'` , alors elle transformera toutes les valeurs qu'il ne peut pas convertir en `NaN` . Supprimer tous les nombres non entiers de la colonne de l'âge avec cette fonction.**

In [ ]:

**7. Transformer la colonne d'âge en une valeur entière. Si vous n'avez pas appliqué (6).**

In [ ]:

**8. Afficher une description de votre dataframe. Remarquer vous une erreur qui se glisse dans les données?**

In [ ]:

**9. Garder uniquement les personnes âgées de moins de 120 ans :**

In [ ]:

**10. Trouvez l'âge moyen des célébrités décédées entre février et juillet.**

In [ ]: