Chapitre IV Visualisation des données

M1 - IA / GL

Ilyas Bambrik

2024

Table des matières

Introduction

Dans ce chapitre, vous apprendrez à faire passer vos visualisations de données au niveau supérieur avec *seaborn*, un outil de visualisation de données puissant mais facile à utiliser. Cette bibliothèque s'appuie sur *matplotlib.pyplot* pour dessiner et visualiser les figures.

Graphique linéaire

I

Le type de graphique le plus simple est le graphique linéaire. Ce type de graphique est souvent utilisé pour visualiser la relation entre deux variables / colonnes. Pour commencer, nous allons importer le Dataset décrivant le nombre de visites aux musées de Los Angeles pour chaque mois entre 2014 et 2018.

```
1 import pandas as pd
2 df=pd.read_csv('museum_visitors.csv')
3 df.head()
```

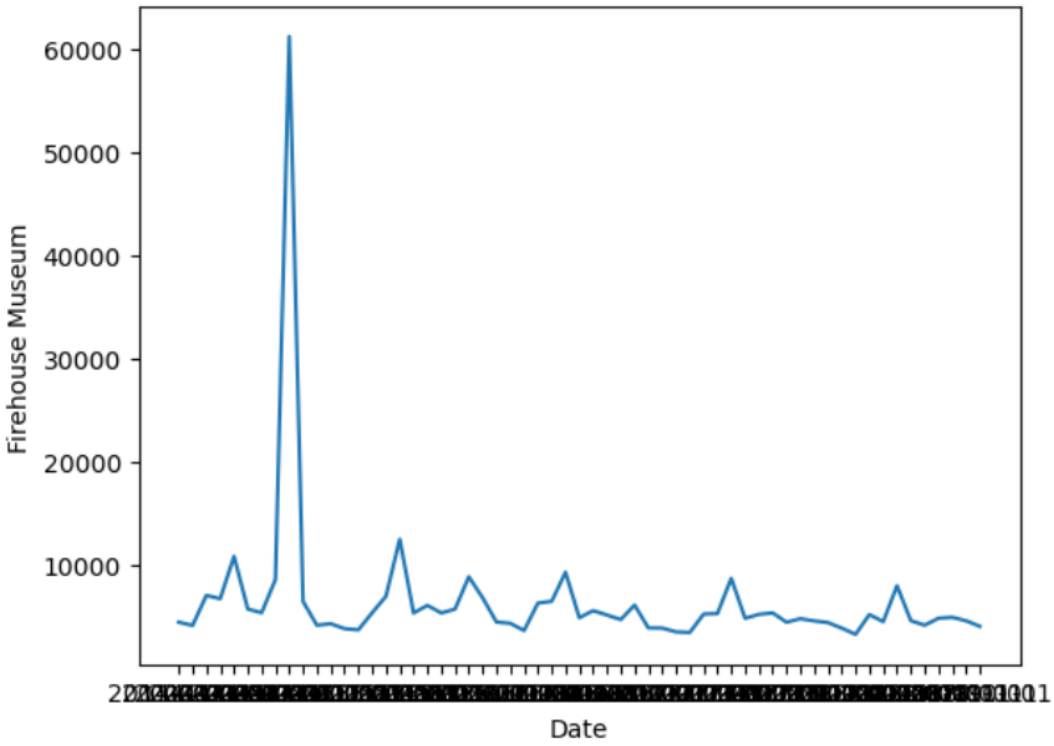
	Date	Avila Adobe	Firehouse Museum	Chinese American Museum	America Tropical Interpretive Center
0	2014-01-01	24778	4486	1581	6602
1	2014-02-01	18976	4172	1785	5029
2	2014-03-01	25231	7082	3229	8129
3	2014-04-01	26989	6756	2129	2824
4	2014-05-01	36883	10858	3676	10694

Pour utiliser *seaborn*, nous devons importer ce module ainsi que *matplotlib.pyplot*.

```
1 import seaborn as sns, matplotlib.pyplot as plt
```

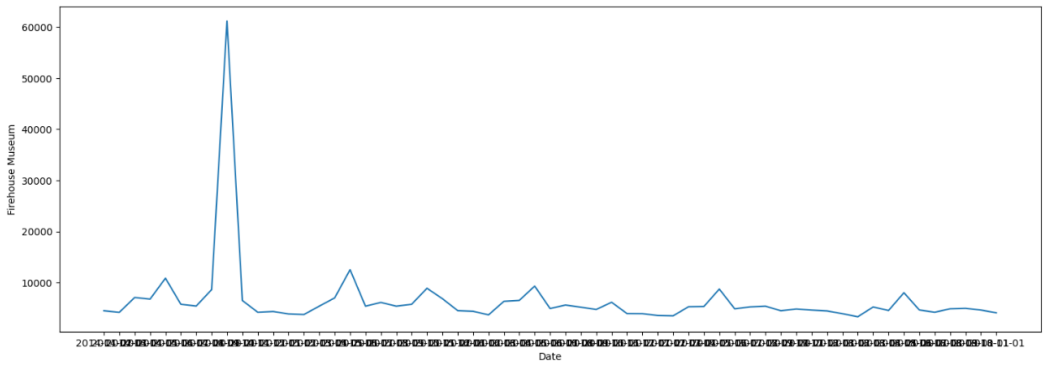
Supposons que nous souhaitons afficher le la nombre des visiteurs du musée 'Firehouse Museum' (3em colonne) en fonction des mois. Pour faire ceci, nous devons faire appel à *sns.lineplot*. Les arguments optionnels *x* et *y* prennent respectivement les colonnes / Series à tracer :

```
1 sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
```



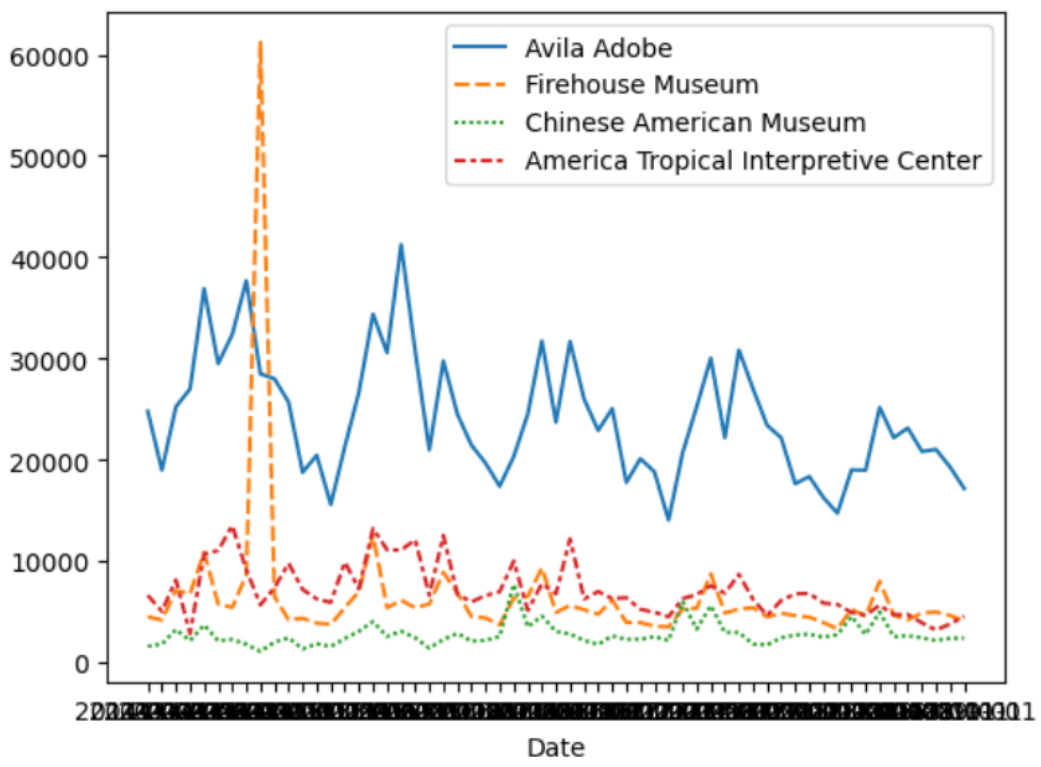
Comme annoncé initialement, *seaborn* dépend de *matplotlib.pyplot*. Par exemple, pour redimensionné la taille de la figure affichée, la méthode *pyplot.figure*() est utilisée avec l'argument optionnel *figsize* qui précise la largeur et la hauteur de la figure en pouce (1 pouce =100 pixels).

```
1 plt.figure(figsize=(18,6))
2 sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
```



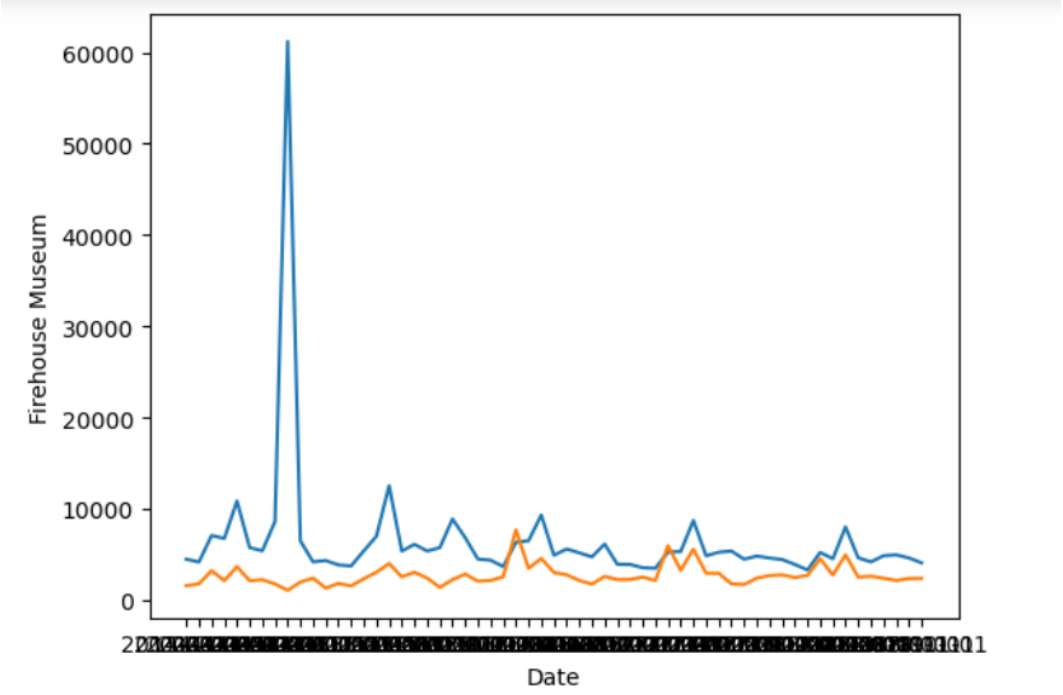
Si nous souhaitons tracer plusieurs colonnes en fonction de la même variable (dans cette exemple la colonne Date), il existe plusieurs façons de réaliser ceci. La plus simple est de mettre la colonne représentant l'axe des x comme index d'un DataFrame contenant les colonnes à tracer. Par la suite, au lieu d'utiliser les arguments *x* et *y*, il suffit d'utiliser l'argument *data* qui prend le DataFrame à tracer:

```
1 sns.lineplot(data=df.set_index('Date'))
```



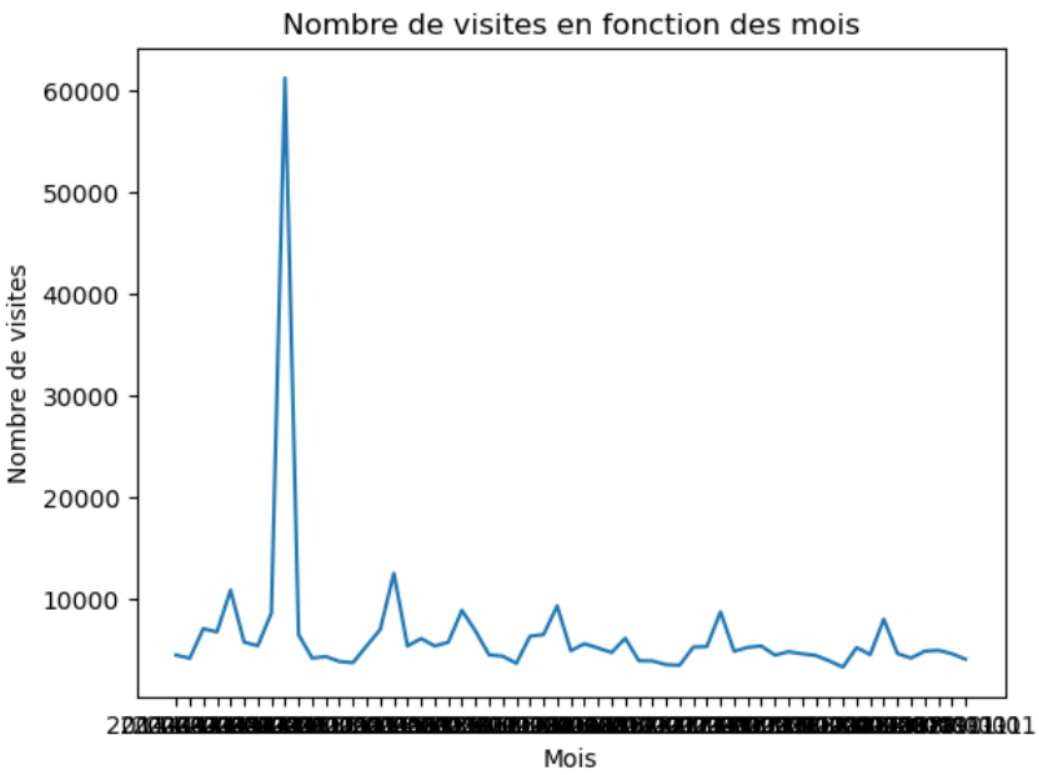
L'autre méthode est un peux plus complexe et utilise l'objet retournée par *sns.lineplot* qui est de type *matplotlib.axes.Axes.plot*. Cet objet peut être passé comme argument à une autre appel de la fonction *sns.lineplot* pour tracer plusieurs variables dans le même plan. Dans le code suivant, le résultat du premier *sns.lineplot* est placé dans la variable axe, puis celle-ci est passée au paramètre optionnel *ax* du deuxième appel *sns.lineplot*.

```
1 axe=sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
2 sns.lineplot(x=df.Date,y=df['Chinese American Museum'],ax=axe)
3
```



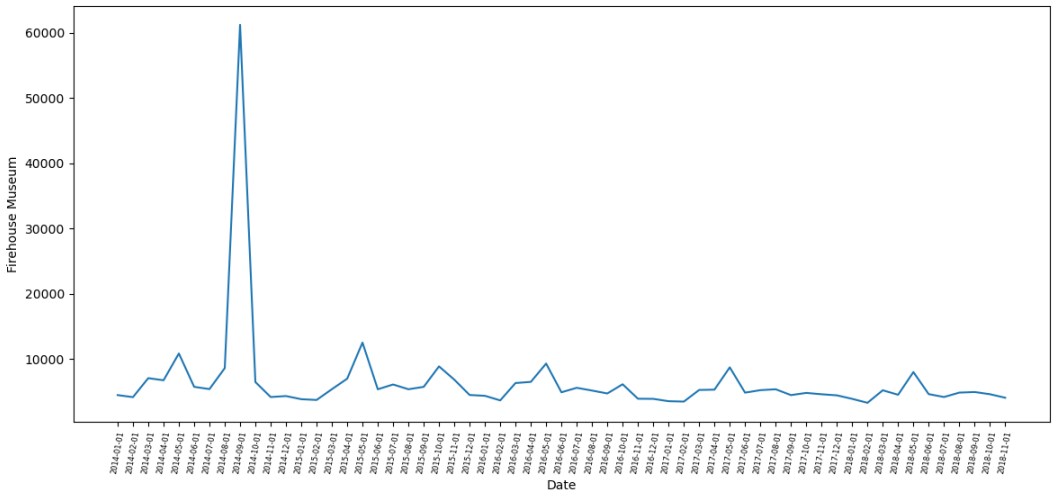
matplotlib.pyplot permet de modifier toutes les propriétés de la figure tel que le titre et les étiquettes des axes. Par défaut, les étiquettes des axes prennent le nom de la colonne / Series. En outre, il est possible de modifier les étiquettes des axes avec la méthode *pyplot.xlabel* et *pyplot.ylable* ainsi que le titre avec *pyplot.title*.

```
sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
plt.xlabel("Mois")
plt.ylabel("Nombre de visites")
plt.title("Nombre de visites en fonction des mois")
```



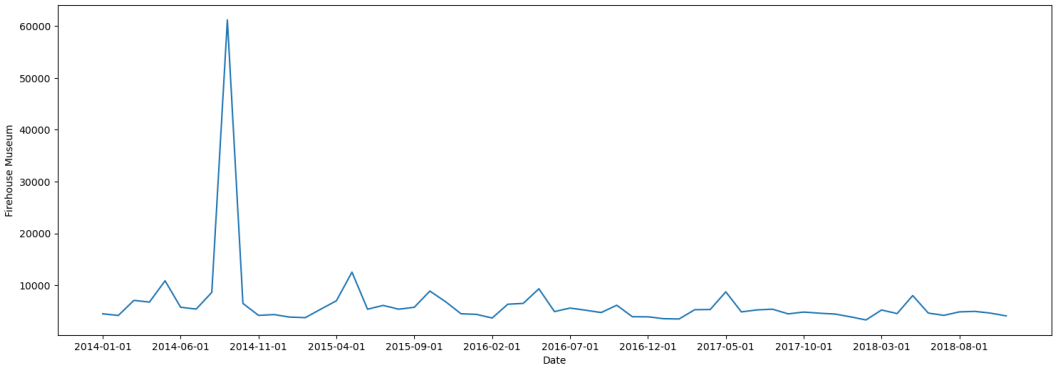
Il est apparent que les dates dans l'axe des x sont chevauchées. Pour résoudre ce problème il existe plusieurs solutions offertes par *matplotlib.pyplot*. Par exemple, augmenter la largeur de la figure et/ou diminuer le taille du police de l'axe des x peuvent présenter une solution. Une meilleur solution consiste à faire une rotation des étiquettes de l'axe des x avec la méthode *ax.set_xticklabels*. Cette méthode prend en paramètre la liste des étiquettes d'un objet *matplotlib.axes.Axes.plot* retournée par *sns.lineplot* et permet de spécifier un angle de rotation (dans cet exemple 80°). Cette méthode permet aussi de modifier la taille du police avec l'argument *fontsize*.

```
plt.figure(figsize=(14,6))
ax=sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
ax.set_xticklabels(ax.get_xticklabels(), rotation=80, fontsize=6)
plt.show()
```



Une autre manière de résoudre ce problème est d'afficher seulement le premier étiquette pour chaque séquence de cinq étiquettes. La méthode *xaxis.get_major_ticks()* de l'objet *matplotlib.axes.Axes.plot* retourne la liste des étiquettes que nous pouvons parcourir et manipuler. La méthode *set_visible* d'un étiquette permet de modifier la visibilité d'un étiquette.

```
plt.figure(figsize=(18,6))
axe=sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
etq=axe.xaxis.get_major_ticks()
for i in range(len(etq)):
    if i%5!=0:
        etq[i].set_visible(False)
plt.show()
```



Remarque : mathplotlib.pyplot

Dans *jupyter*, l'affichage des graphiques se fait automatiquement mais si vous utilisez *seaborn* dans un programme interactif, vous devez manuellement faire appel à *matplotlib.pyplot.show()* pour faire apparaître la figure.

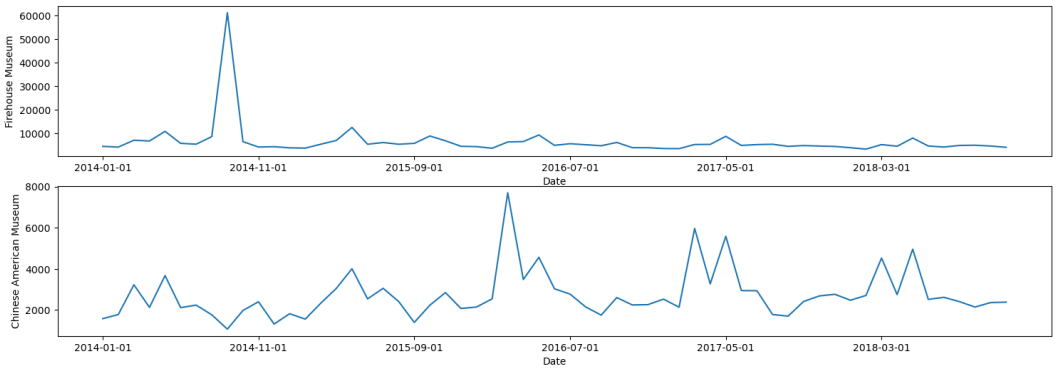
```
sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
plt.show()
```

Subplots et FacetGrid

II

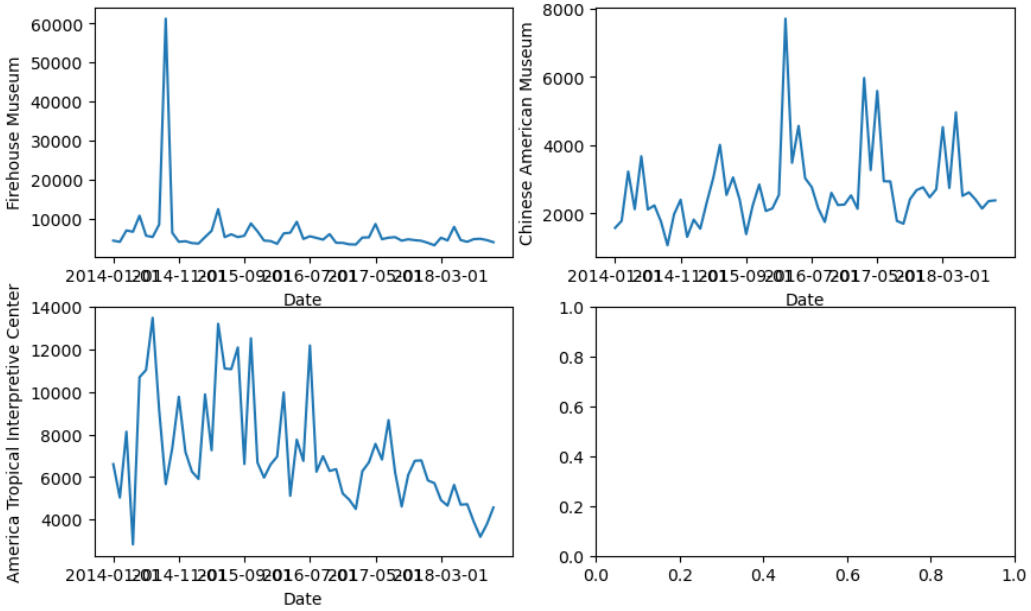
Alors que l'objet *matplotlib.axes.Axes.plot* représente le plan pour un seul graphe, il est possible de créer une grille contenant plusieurs plans avec la méthode *pyplot.subplots*. Cette méthode prend en paramètre le nombre de ligne et de colonne dans la grille. L'exemple suivant crée une grille d'une seule colonne et deux lignes. Par la suite, l'argument *ax* de la fonction *linechart* est utilisé pour spécifier la position de la figure sur la grille :

```
fig, axes = plt.subplots(2, 1,figsize=(18,6))
sns.lineplot(x=df.Date,y=df['Firehouse Museum'],ax=axes[0])
sns.lineplot(x=df.Date,y=df["Chinese American Museum"],ax=axes[1])
etq1=axes[0].xaxis.get_major_ticks()
etq2=axes[1].xaxis.get_major_ticks()
for i in range(len(etq1)):
    if i%10!=0:
        etq1[i].set_visible(False)
        etq2[i].set_visible(False)
plt.show()
```



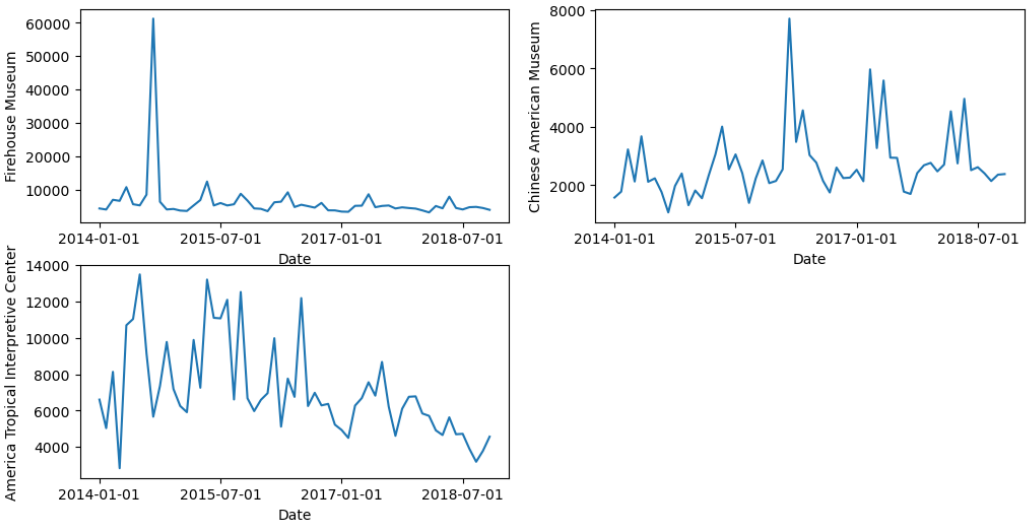
L'exemple suivant, crée une grille 2 x 2. Ainsi, contrairement à l'objet axes du premier exemple qui avais une seule dimension, l'objet axes retourné par *subplots* dans l'exemple suivant est une matrice de deux dimensions 2 x 2 :

```
fig, axes = plt.subplots(2, 2,figsize=(10,6))
sns.lineplot(x=df.Date,y=df['Firehouse Museum'],ax=axes[0,0])
sns.lineplot(x=df.Date,y=df["Chinese American Museum"],ax=axes[0,1])
sns.lineplot(x=df.Date,y=df['America Tropical Interpretive Center'],ax=axes[1,0])
etq1=axes[0][0].xaxis.get_major_ticks()
etq2=axes[0][1].xaxis.get_major_ticks()
etq3=axes[1][0].xaxis.get_major_ticks()
for i in range(len(etq1)):
    if i%10!=0:
        etq1[i].set_visible(False)
        etq2[i].set_visible(False)
        etq3[i].set_visible(False)
plt.show()
```



Cependant, nous avons utiliser seulement trois figures de notre grille. Pour cacher les axes de la dernière figure, il suffit d'utiliser la méthode *set_visible* de la case 1,1 de l'objet axes (*ligne 8*) :

```
1 fig, axes = plt.subplots(2, 2,figsize=(12,6))
2 sns.lineplot(x=df.Date,y=df['Firehouse Museum'],ax=axes[0,0])
3 sns.lineplot(x=df.Date,y=df["Chinese American Museum"],ax=axes[0,1])
4 sns.lineplot(x=df.Date,y=df['America Tropical Interpretive Center'],ax=axes[1,0])
5 etq1=axes[0][0].xaxis.get_major_ticks()
6 etq2=axes[0][1].xaxis.get_major_ticks()
7 etq3=axes[1][0].xaxis.get_major_ticks()
8 axes[1][1].set_visible(False) # cache la figure à la position 1,1 de la grille
9 for i in range(len(etq1)):
10     if i%18!=0:
11         etq1[i].set_visible(False)
12         etq2[i].set_visible(False)
13         etq3[i].set_visible(False)
14 plt.show()
15
```

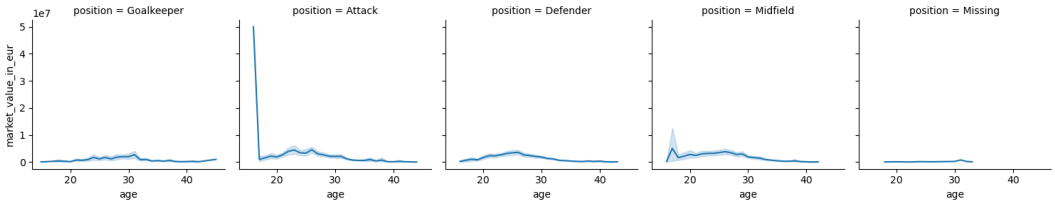


Parfois, il est utile de diviser les données en sous-ensemble selon une ou plusieurs colonnes catégoriques et tracer chaque graphique dans le même plan. Possible de faire ceci avec *pyplot.subplots* mais il y a un alternatif prédéfini avec *seaborn.FacetGrid*. L'utilité de cet alternative est mise en évidence lorsque le nombre de valeurs uniques dans la colonne catégorique est élevé. Prenant le Dataset *players.csv* et commençant par la création d'une colonne contenant l'age du joueur :

```
from datetime import date
df=pd.read_csv('players.csv')
df['age']=date.today().year-pd.to_datetime(df.date_of_birth).dt.year
```

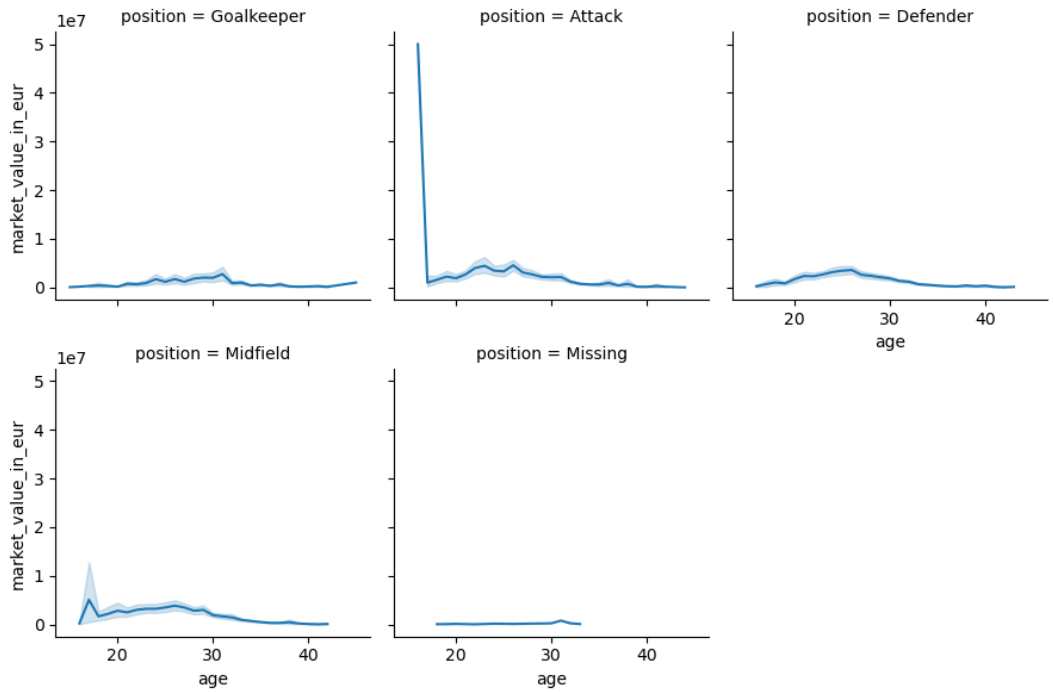
Le constructeur de la classe *seaborn.FacetGrid* prend en paramètre le *DataFrame* et le(s) nom(s) de la colonne pour diviser le Dataset avec les argument *col* (comme dans l'exemple suivant "*position*" du Dataset *players.csv*) et /ou *row*. L'objet *FacetGrid* créé propose la méthode *map* qui prend les noms des colonnes à tracer (*"age"* et *"market_value_in_eur"*) et la fonction *seaborn* de traçage (dans l'exemple suivant *lineplot*) :

```
fg=sns.FacetGrid(df, col="position")
fg.map(sns.lineplot,"age","market_value_in_eur")
```



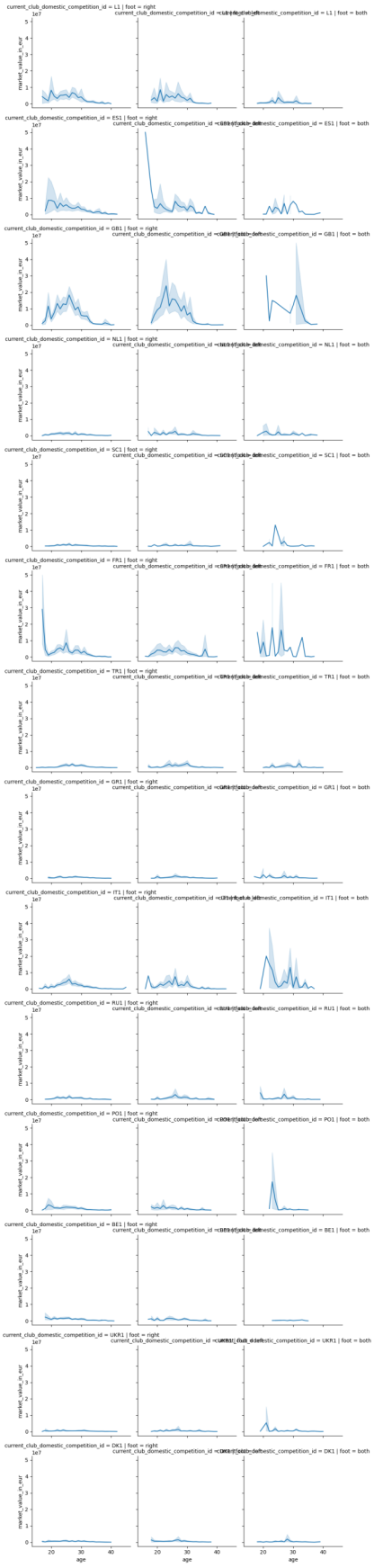
L'argument optionnel *col_wrap* du constructeur *FacetGrid* permet de spécifier le nombre maximal de figures par ligne (dans cet exemple *col_wrap*=3) :

```
fg=sns.FacetGrid(df, col="position",col_wrap=3)
fg.map(sns.lineplot,"age","market_value_in_eur")
```



Dans l'exemple précédant, nous avant diviser le DataSet par la colonne *position* avec l'argument *col*. Il est aussi possible de diviser par une autre variable catégorique avec l'argument optionnel *row* (dans l'exemple suivant row = *"current_club_domestic_competition_id"*) :

```
1 fg=sns.FacetGrid(df, col="foot", row="current_club_domestic_competition_id")
2 fg.map(sns.lineplot,"age","market_value_in_eur")
```


current_club_domestic_competition_id = L1 | foot = right
current_club_domestic_competition_id = L1 | foot = left
current_club_domestic_competition_id = L1 | foot = both
current_club_domestic_competition_id = ES1 | foot = right
current_club_domestic_competition_id = ES1 | foot = left
current_club_domestic_competition_id = ES1 | foot = both
current_club_domestic_competition_id = GB1 | foot = right
current_club_domestic_competition_id = GB1 | foot = left
current_club_domestic_competition_id = GB1 | foot = both
current_club_domestic_competition_id = NL1 | foot = right
current_club_domestic_competition_id = NL1 | foot = left
current_club_domestic_competition_id = NL1 | foot = both
current_club_domestic_competition_id = SC1 | foot = right
current_club_domestic_competition_id = SC1 | foot = left
current_club_domestic_competition_id = SC1 | foot = both
current_club_domestic_competition_id = FR1 | foot = right
current_club_domestic_competition_id = FR1 | foot = left
current_club_domestic_competition_id = FR1 | foot = both
current_club_domestic_competition_id = TR1 | foot = right
current_club_domestic_competition_id = TR1 | foot = left
current_club_domestic_competition_id = TR1 | foot = both
current_club_domestic_competition_id = GR1 | foot = right
current_club_domestic_competition_id = GR1 | foot = left
current_club_domestic_competition_id = GR1 | foot = both
current_club_domestic_competition_id = IT1 | foot = right
current_club_domestic_competition_id = IT1 | foot = left
current_club_domestic_competition_id = IT1 | foot = both
current_club_domestic_competition_id = RU1 | foot = right
current_club_domestic_competition_id = RU1 | foot = left
current_club_domestic_competition_id = RU1 | foot = both
current_club_domestic_competition_id = PO1 | foot = right
current_club_domestic_competition_id = PO1 | foot = left
current_club_domestic_competition_id = PO1 | foot = both
current_club_domestic_competition_id = BE1 | foot = right
current_club_domestic_competition_id = BE1 | foot = left
current_club_domestic_competition_id = BE1 | foot = both
current_club_domestic_competition_id = UKR1 | foot = right
current_club_domestic_competition_id = UKR1 | foot = left
current_club_domestic_competition_id = UKR1 | foot = both
current_club_domestic_competition_id = DK1 | foot = right
current_club_domestic_competition_id = DK1 | foot = left
current_club_domestic_competition_id = DK1 | foot = both
1e7
market_value_in_eur
age

Barplots et Histogrames

Prenant le Dataset *video_games_sales.csv* qui fournit des informations sur les ventes de jeux vidéo, s'étalant sur plusieurs années et sur différentes régions et plates-formes. A noter que les colonnes *na_sales, eu_sales, jp_sales, other_sales, global_sales* représentent le nombre de copies vendues en millions.

```
import pandas as pd,seaborn as sns, matplotlib.pyplot as plt
df=pd.read_csv('video_games_sales.csv')
df.head()
```

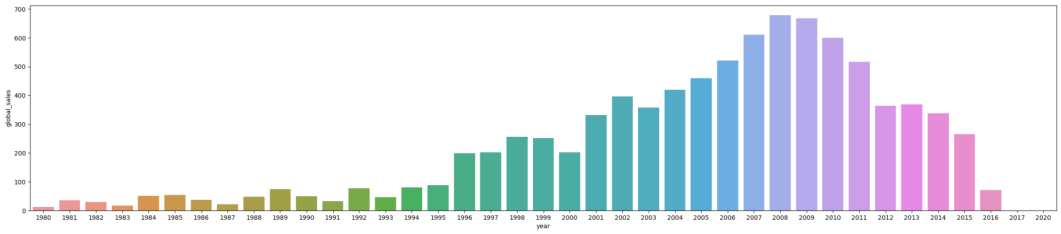
	rank	name	platform	year	genre	publisher	na_sales	eu_sales	jp_sales	other_sales	global_sales
0	1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
3	4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00
4	5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37

Un *Barplot* (graphique à barres) est un graphique qui peut être horizontal ou vertical. Le point important à noter concernant les graphiques à barres est la longueur de chaque barre: plus leur longueur est grande, plus leur valeur est élevée. Les graphiques à barres sont l'une des nombreuses techniques utilisées pour présenter les données sous une forme visuelle afin que le lecteur puisse facilement reconnaître des modèles ou des tendances.

Les *Barplot*s présentent généralement des variables catégorielles, des variables discrètes ou des variables continues regroupées en intervalles de classes. Ils sont constitués d'un axe et d'une série de barres horizontales ou verticales étiquetées et *séparées par des espaces*.

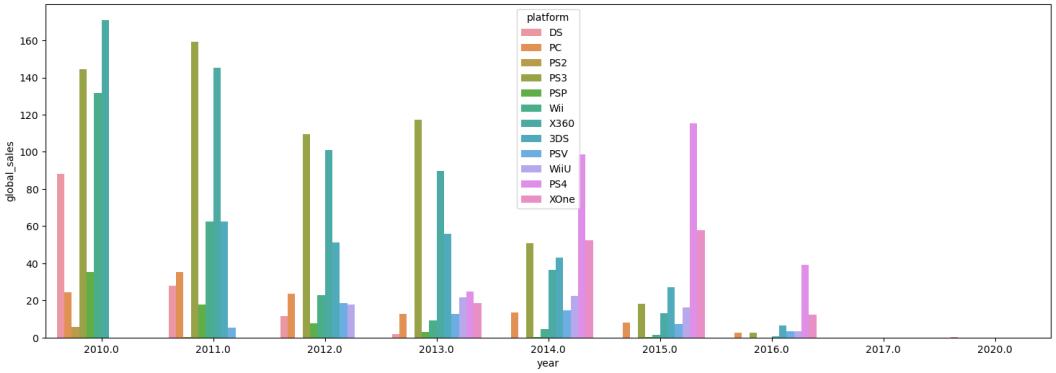
Supposant que nous souhaitons tracer la quantité des copies vendues chaque année. Ici bien-sur, nous avons une variable continue (*global_sales*) tracée en fonction d'une variable discrète (*year*). La fonction *seaborn.barplot* permet de faire ce-ci :

```
plt.figure(figsize=(30,6))
ventes_par_platform=df.groupby('year').global_sales.sum()
sns.barplot(x=ventes_par_platform.index.astype(int),y=ventes_par_platform)
```



Supposons que nous souhaitons toujours présenter le même graphe mais pour chaque année, il faudra diviser encore par la variable catégorique *platforme*. Ainsi pour chaque année, nous traçons la quantité vendue par plate-forme. Ceci, est fait par l'argument optionnel *hue* qui prend le nom de la variable catégorique qui sert à diviser les observations à tracer. Ce paramètre est utilisable dans la plupart des graphes *seaborn* de la même façon. Pour simplifier l'affichage, nous allons prendre seulement les ventes enregistrées depuis 2010.

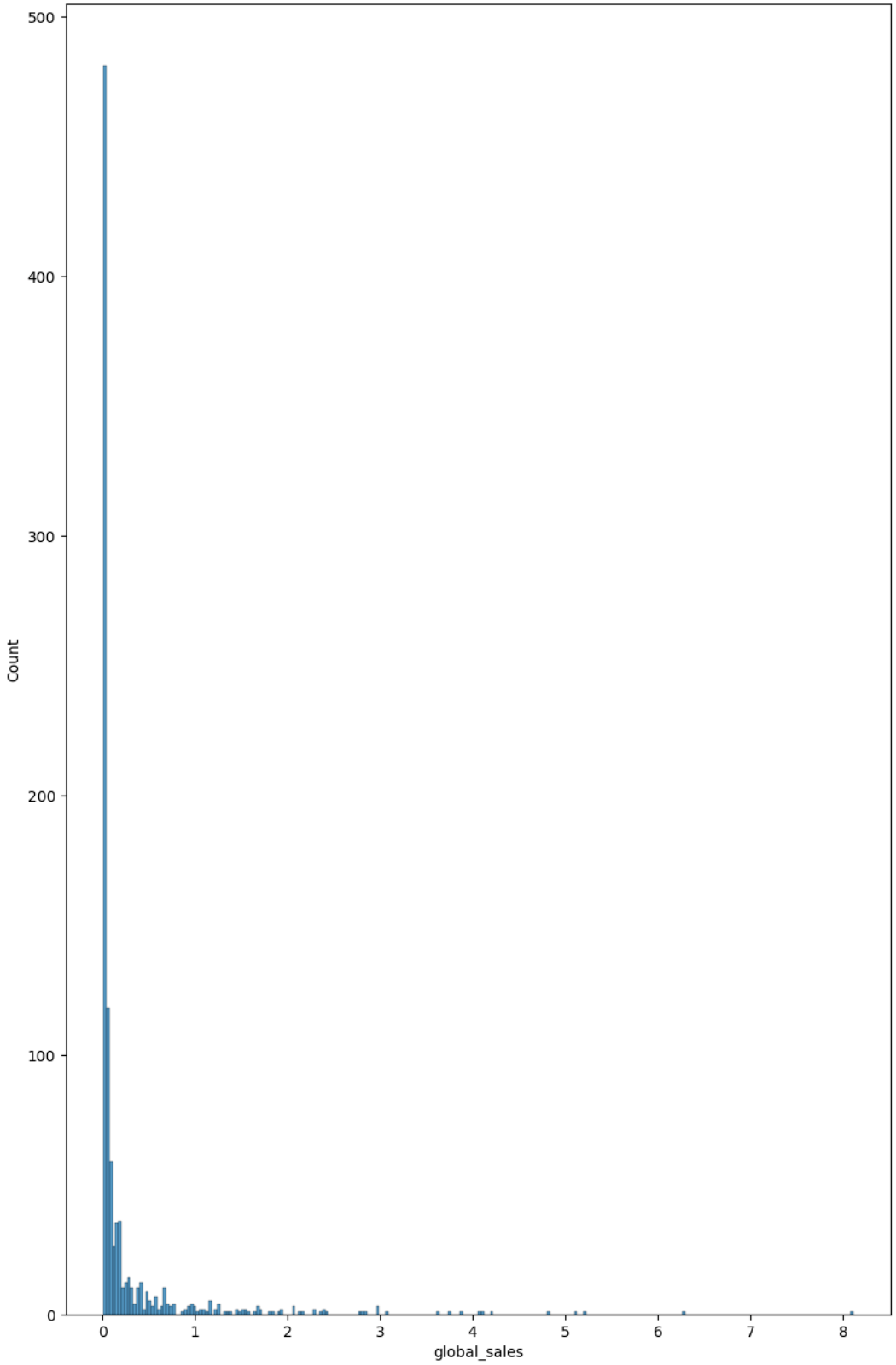
```
1 plt.figure(figsize=(18,6))
2 df10=df[df.year>=2010]
3 ventes_par_platform=df10.groupby(['year','platform']).global_sales.sum().
  reset_index(['platform',"year"])
4 sns.barplot(x="year",y="global_sales",hue="platform",data=ventes_par_platform)
```



Un graphe populaire similaire au *Barplot* est l'histogramme (*Histplot*). Mais contrairement au précédant, il est utilisé pour résumer des données discrètes ou *continues* mesurées sur une échelle d'intervalle (*les barplots ne peuvent pas être utilisés sur une variable continue*). Ce type de représentation est souvent utilisée pour illustrer les principales caractéristiques de la distribution des données et peut également aider à détecter des observations inhabituelles (valeurs aberrantes).

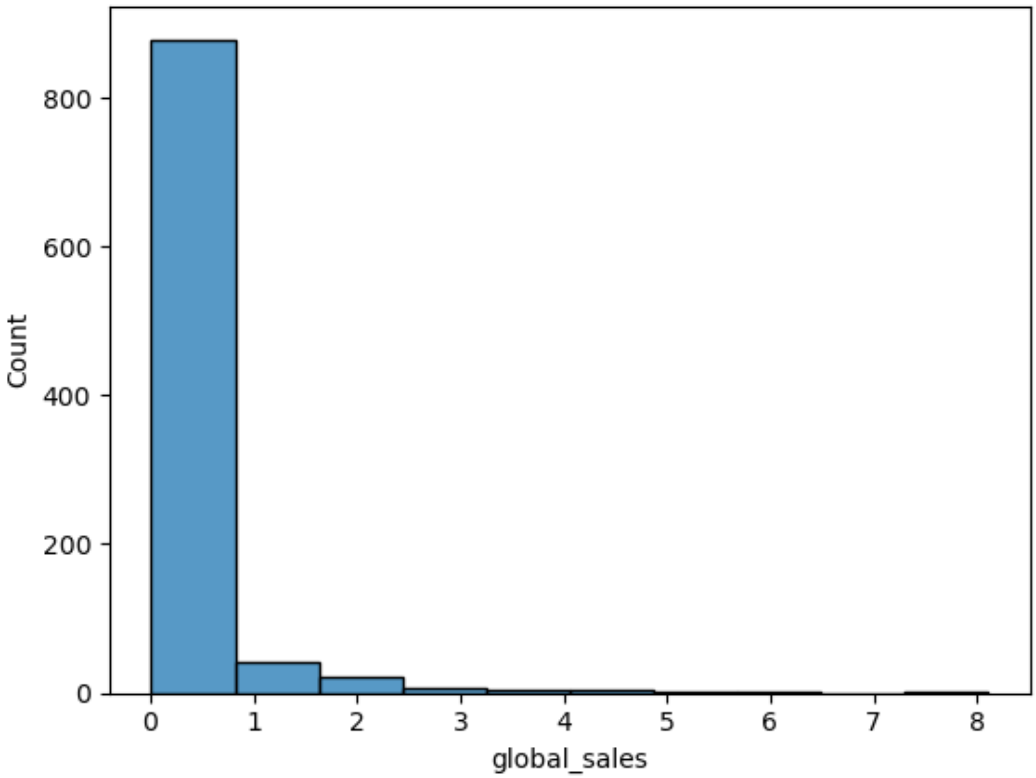
Par exemple, traçons la distribution des ventes des jeux sur PC avec la méthode *seaborn.histplot*: celle-ci divise la lvariable continue en intervalles de la même taille et trace le nombre d'observation appartenant à chaque intervalle.

```
1 plt.figure(figsize=(10,16))
2 sns.histplot(x=df[(df.platform=='PC') ].global_sales)
```



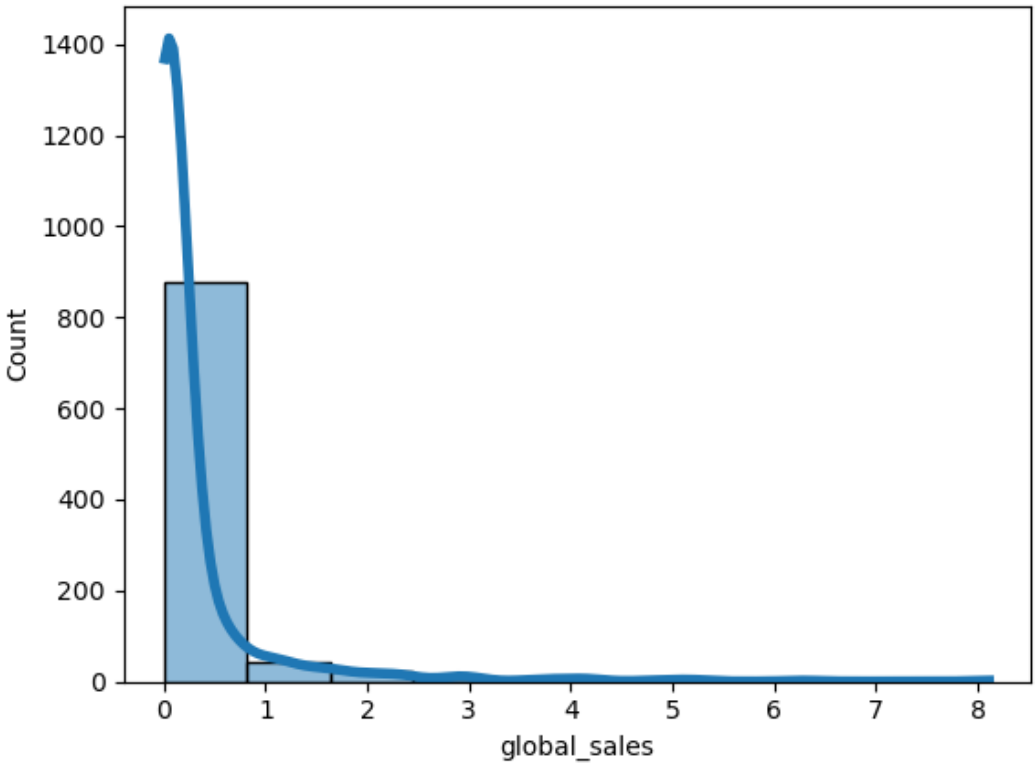
Comme dans *cut*, nous pouvons utiliser le paramètre *bins* pour spécifier le nombre de intervalles ou les intervalles spécifiques à tracer.

```
1 sns.histplot(x=df[(df.platform=='PC') ].global_sales,bins=10)
```



Pour afficher la fonction de densité estimé par la méthode Kernel Density Estimation (KDE), il suffit d'utiliser l'argument *kde=True*. Le graphe illustre que les ventes pour la plus part des jeux vidéos sur PC ne dépassent pas 1 million de copies. L'argument optionnel *line_kws* prend un dictionnaire qui permet de modifier les propriétés du graphe de la fonction de densité (ici, l'épaisseur de la ligne est modifier avec *lw*). L'affichage montre que selon la fonction de distribution estimée, les ventes sur pc suit une *loi exponentielle*.

```
sns.histplot(x=df[(df.platform=='PC') ].global_sales,bins=10, kde=True,line_kws={
 'lw': 4})
```


1400
1200
1000
800
600
400
200
0
Count
0
1
2
3
4
5
6
7
8
global_sales

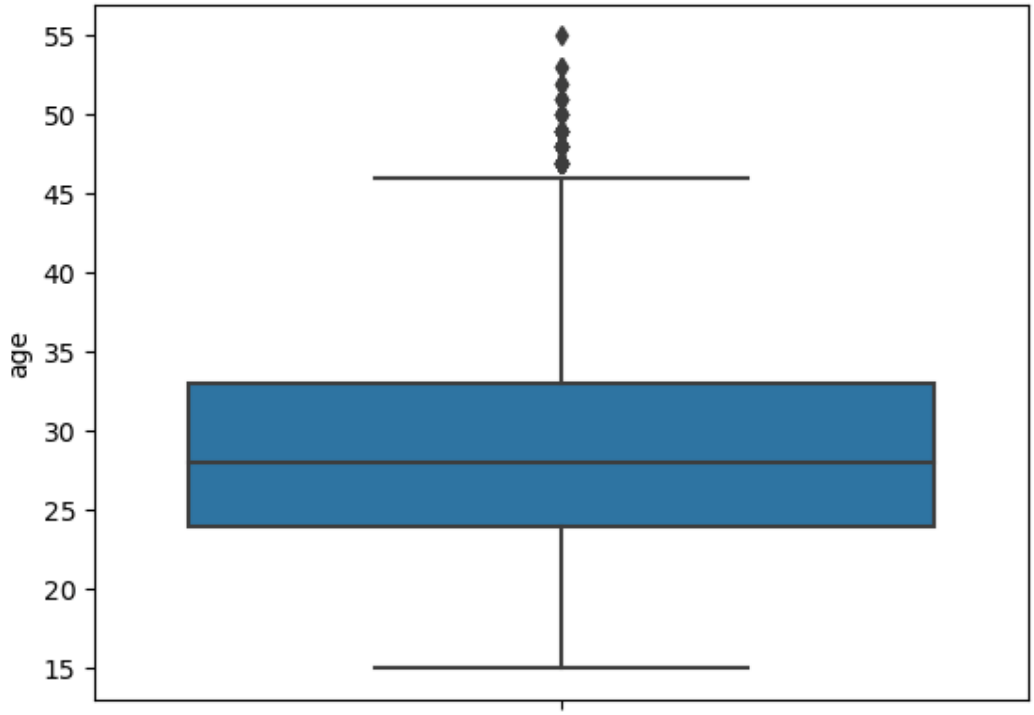
Boxplot

IV

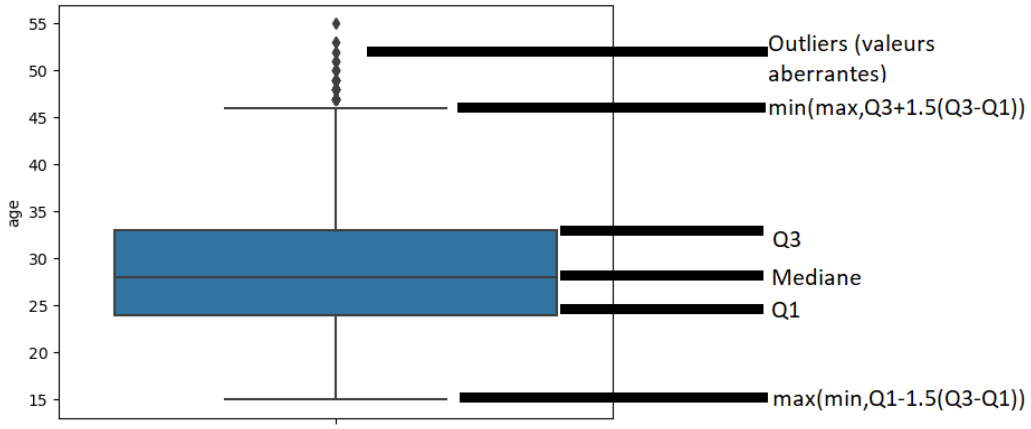
Un diagramme *Boxplot* (*whisker plot*) montre la distribution des données quantitatives de manière à faciliter les comparaisons entre les variables ou entre les niveaux d'une variable catégorielle. Le cadre de ce graphique montre les quartiles de l'ensemble de données tandis que les moustaches s'étendent pour montrer le reste de la distribution, à l'exception des points déterminés comme étant « aberrants » à l'aide d'une méthode de l'intervalle inter-quartile.

Prenant le Dataset *players.csv*. Le code suivant affiche le *boxplot* de l'age des joueurs (cette colonne est calculée à partir de *date_of_birth*):

```
1
2 players["age"]=date.today().year-pd.to_datetime(players.date_of_birth).dt.year
3 sns.boxplot(y="age",data=players)
```

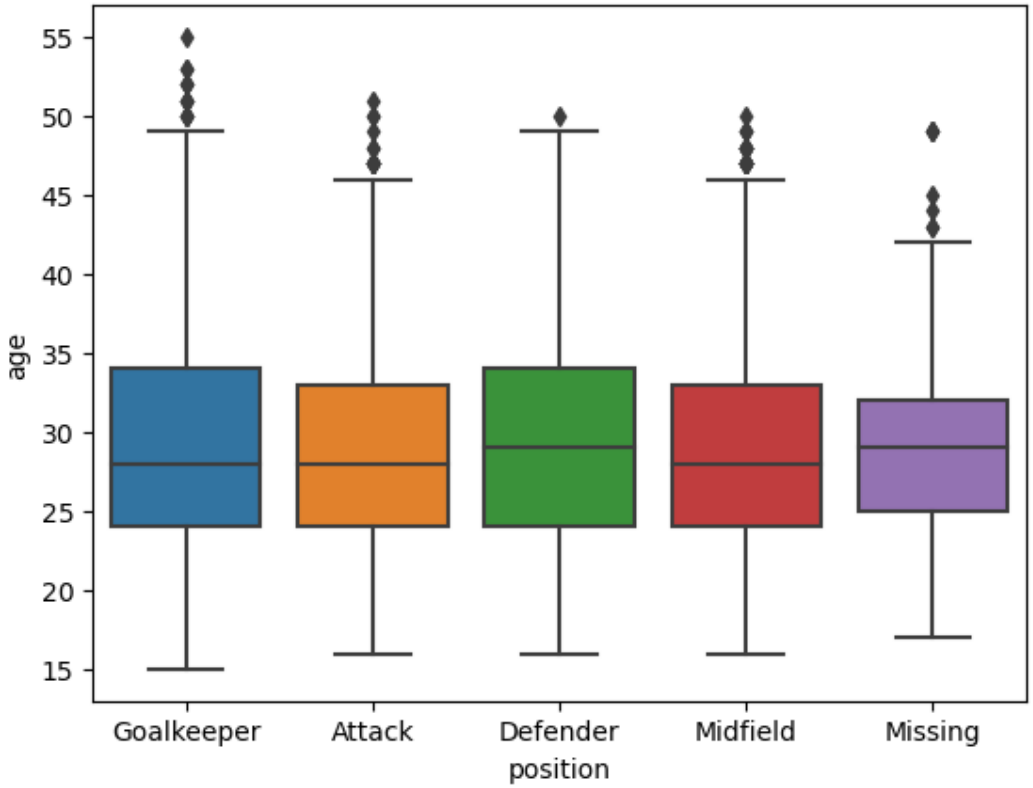


Pour rappel, la *médiane* c'est l'élément placé au *milieu des valeurs après le trie*. *Q1* c'est l'élément placé au milieu de la partie inférieure (entre médiane et le minimum) *après le trie* et *Q3* c'est l'élément placé au milieu de la partie supérieure (entre médiane et le maximum) *après le trie*. Toute valeur supérieure à Q3+1.5(Q3-Q1) ou inférieure à Q1-1.5(Q3-Q1) est considérée comme une valeur aberrante :



Il est possible de tracer le box plot d'une variable pour chaque valeur d'une autre variable catégorique ou discrète. Par exemple, le code suivant trace le *boxplot* de l'*age* selon la *position*. Ceci est fait simplement en spécifiant la deuxième variable avec l'argument *x*.

```
sns.boxplot(y="age",x="position",data=players )
```



Heatmap

V

Un *Heatmap* (carte thermique) est un moyen de représenter les données sous une forme bidimensionnelle. Les valeurs des données sont représentées sous forme de couleurs dans le graphique. L'objectif de la carte thermique est de fournir un résumé visuel coloré des informations. Cette représentation est utilisée souvent pour afficher une matrice de confusion de deux variables catégoriques/discrètes ainsi que la matrice de corrélation.

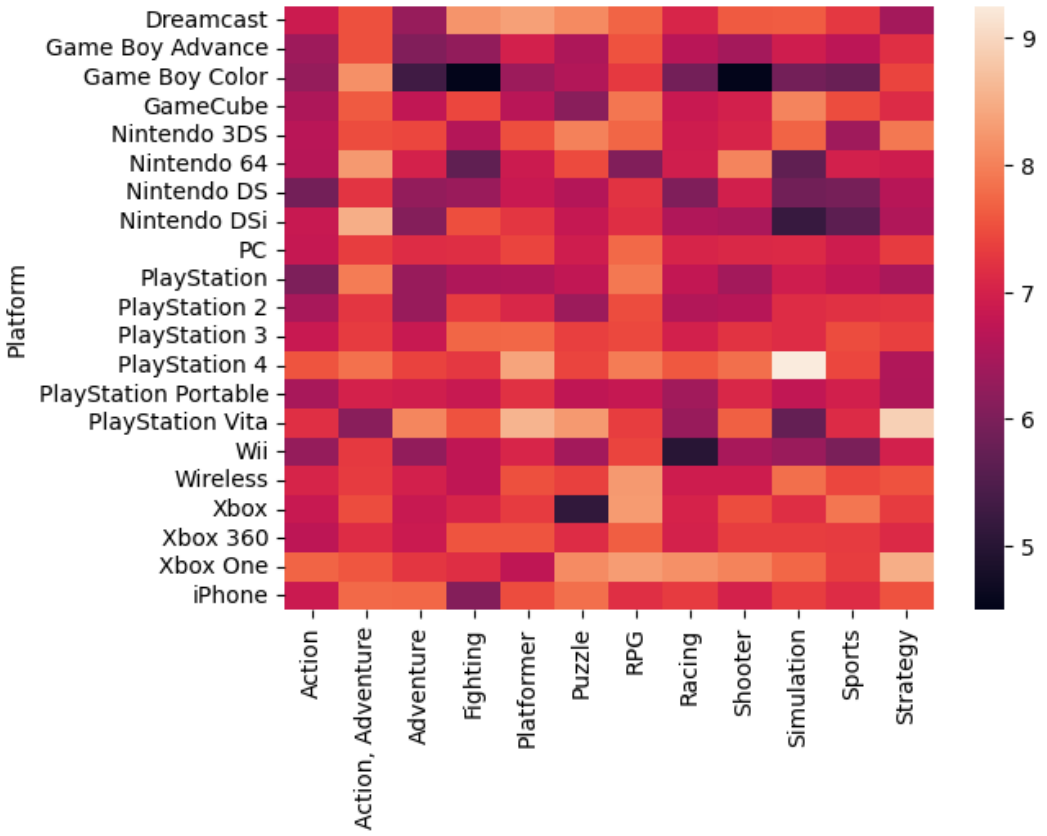
Prenant le DataSet *ign_scores.csv*, qui montre pour chaque catégorie de jeux, le score moyen par plate-forme :

```
scores=pd.read_csv('ign_scores.csv',index_col=0)
scores.head()
```

Platform	Action	Action, Adventure	Adventure	Fighting	Platformer	Puzzle	RPG	Racing	Shooter	Simulation	Sports	Strategy
Dreamcast	6.882857	7.511111	6.281818	8.200000	8.340000	8.088889	7.700000	7.042500	7.616667	7.628571	7.272222	6.433333
Game Boy Advance	6.373077	7.507692	6.057143	6.226316	6.970588	6.532143	7.542857	6.657143	6.444444	6.928571	6.694444	7.175000
Game Boy Color	6.272727	8.166667	5.307692	4.500000	6.352941	6.583333	7.285714	5.897436	4.500000	5.900000	5.790698	7.400000
GameCube	6.532584	7.608333	6.753846	7.422222	6.665714	6.133333	7.890909	6.852632	6.981818	8.028571	7.481319	7.116667
Nintendo 3DS	6.670833	7.481818	7.414286	6.614286	7.503448	8.000000	7.719231	6.900000	7.033333	7.700000	6.388889	7.900000

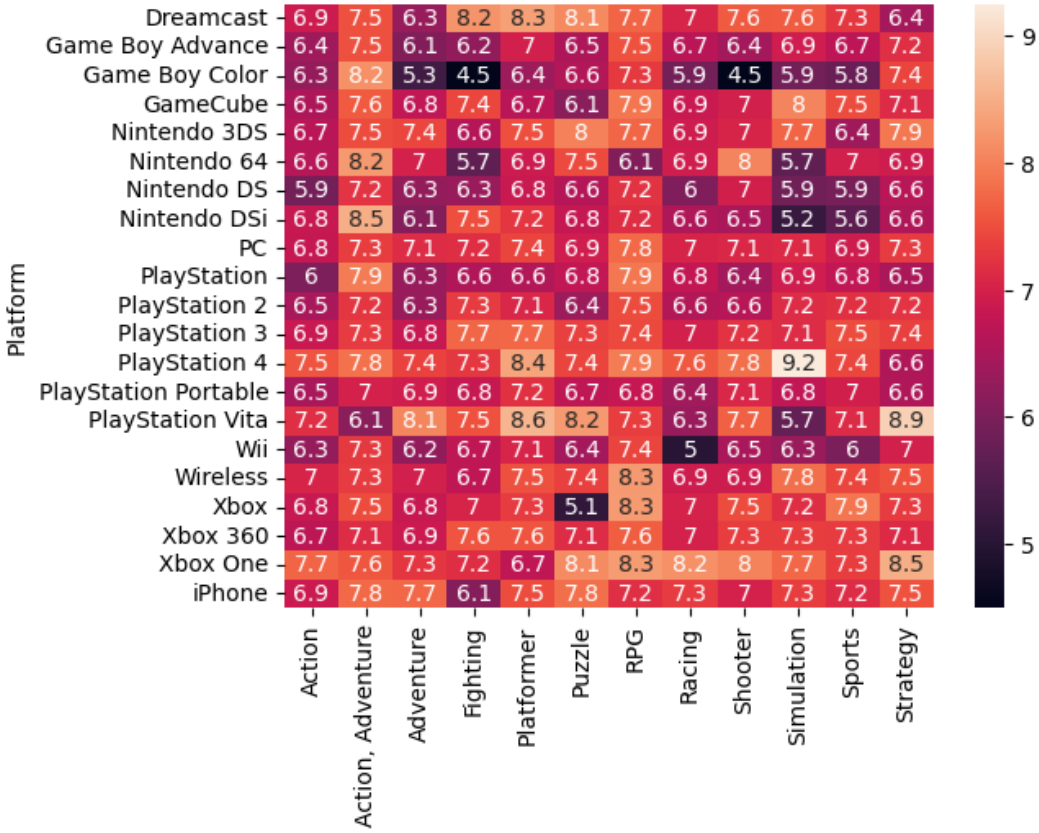
Pour tracer le graphe *Heatmap* des scores selon le genre (*colonne*) et la plate-forme (*index*)

```
sns.heatmap(scores)
```



Les valeurs du Data Set sont encodées avec un couleur claire pour les valeurs les plus élevées et une couleur sombre pour les valeurs proches de 0. Possible d'afficher les valeurs correspondantes sur le graphe avec l'argument *annot=True*.

```
1 sns.heatmap(scores,annot=True)
```


Platform
Dreamcast 6.9 7.5 6.3 8.2 8.3 8.1 7.7 7 7.6 7.6 7.3 6.4
Game Boy Advance 6.4 7.5 6.1 6.2 7 6.5 7.5 6.7 6.4 6.9 6.7 7.2
Game Boy Color 6.3 8.2 5.3 4.5 6.4 6.6 7.3 5.9 4.5 5.9 5.8 7.4
GameCube 6.5 7.6 6.8 7.4 6.7 6.1 7.9 6.9 7 8 7.5 7.1
Nintendo 3DS 6.7 7.5 7.4 6.6 7.5 8 7.7 6.9 7 7.7 6.4 7.9
Nintendo 64 6.6 8.2 7 5.7 6.9 7.5 6.1 6.9 8 5.7 7 6.9
Nintendo DS 5.9 7.2 6.3 6.3 6.8 6.6 7.2 6 7 5.9 5.9 6.6
Nintendo DSi 6.8 8.5 6.1 7.5 7.2 6.8 7.2 6.6 6.5 5.2 5.6 6.6
PC 6.8 7.3 7.1 7.2 7.4 6.9 7.8 7 7.1 7.1 6.9 7.3
PlayStation 6 7.9 6.3 6.6 6.6 6.8 7.9 6.8 6.4 6.9 6.8 6.5
PlayStation 2 6.5 7.2 6.3 7.3 7.1 6.4 7.5 6.6 6.6 7.2 7.2 7.2
PlayStation 3 6.9 7.3 6.8 7.7 7.7 7.3 7.4 7 7.2 7.1 7.5 7.4
PlayStation 4 7.5 7.8 7.4 7.3 8.4 7.4 7.9 7.6 7.8 9.2 7.4 6.6
PlayStation Portable 6.5 7 6.9 6.8 7.2 6.7 6.8 6.4 7.1 6.8 7 6.6
PlayStation Vita 7.2 6.1 8.1 7.5 8.6 8.2 7.3 6.3 7.7 5.7 7.1 8.9
Wii 6.3 7.3 6.2 6.7 7.1 6.4 7.4 5 6.5 6.3 6 7
Wireless 7 7.3 7 6.7 7.5 7.4 8.3 6.9 6.9 7.8 7.4 7.5
Xbox 6.8 7.5 6.8 7 7.3 5.1 8.3 7 7.5 7.2 7.9 7.3
Xbox 360 6.7 7.1 6.9 7.6 7.6 7.1 7.6 7 7.3 7.3 7.3 7.1
Xbox One 7.7 7.6 7.3 7.2 6.7 8.1 8.3 8.2 8 7.7 7.3 8.5
iPhone 6.9 7.8 7.7 6.1 7.5 7.8 7.2 7.3 7 7.3 7.2 7.5
Action
Action, Adventure
Adventure
Fighting
Platformer
Puzzle
RPG
Racing
Shooter
Simulation
Sports
Strategy
9
8
7
6
5