Chapitre IV Visualisation des données

Cours Data Science M1 IA

Ilyas Bambrik

2024

Table des matières

Introduction

Dans ce chapitre, vous apprendrais à utiliser *seaborn*[8*], un outil de visualisation de données puissant mais facile à utiliser. Cette bibliothèque s'appuie sur *matplotlib.pyplot*[9*] pour dessiner et visualiser les graphiques.

Graphique linéaire

I

Le type de graphique le plus simple est le graphique linéaire. Ce type de graphique est souvent utilisé pour visualiser comment la valeur d'une colonne change en fonction du temps ainsi que les valeurs aberrantes de celle-ci. En outre, ce type de graphe peut être utilisé pour visualiser la relation entre deux variables/colonnes ou à fin de comparer plusieurs séries de valeurs. Pour commencer, nous allons importer le Dataset décrivant le nombre de visites aux musées de Los Angeles[10*] pour chaque mois entre 2014 et 2018.

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

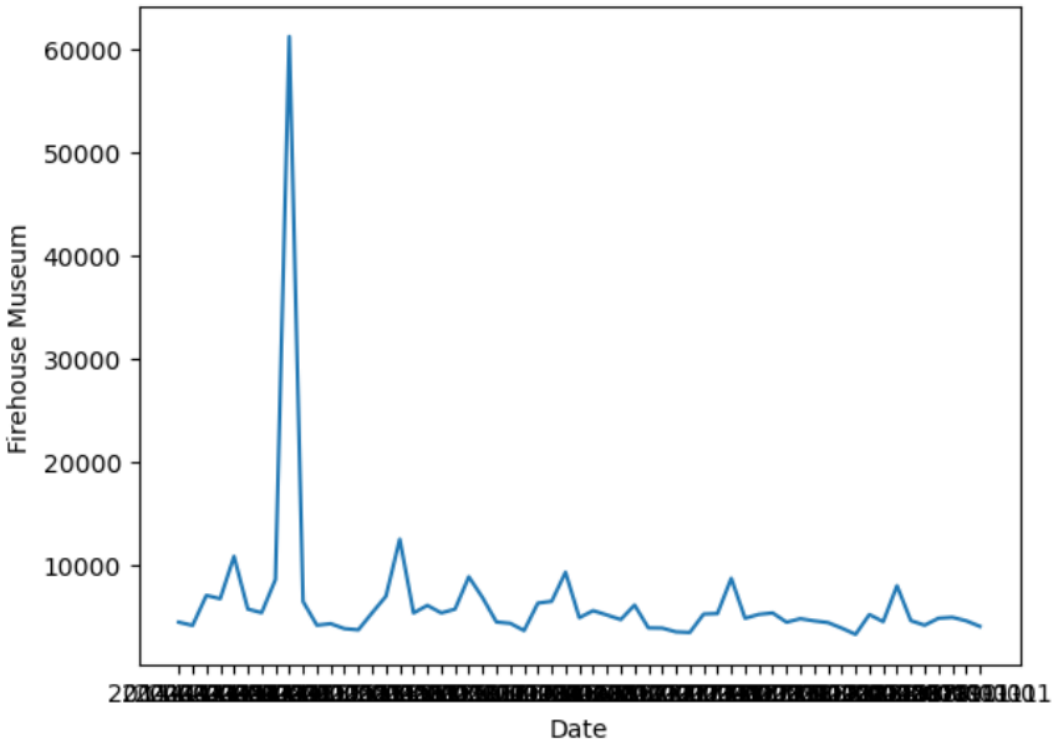
	Date	Avila Adobe	Firehouse Museum	Chinese American Museum	America Tropical Interpretive Center
0	2014-01-01	24778	4486	1581	6602
1	2014-02-01	18976	4172	1785	5029
2	2014-03-01	25231	7082	3229	8129
3	2014-04-01	26989	6756	2129	2824
4	2014-05-01	36883	10858	3676	10694

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

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

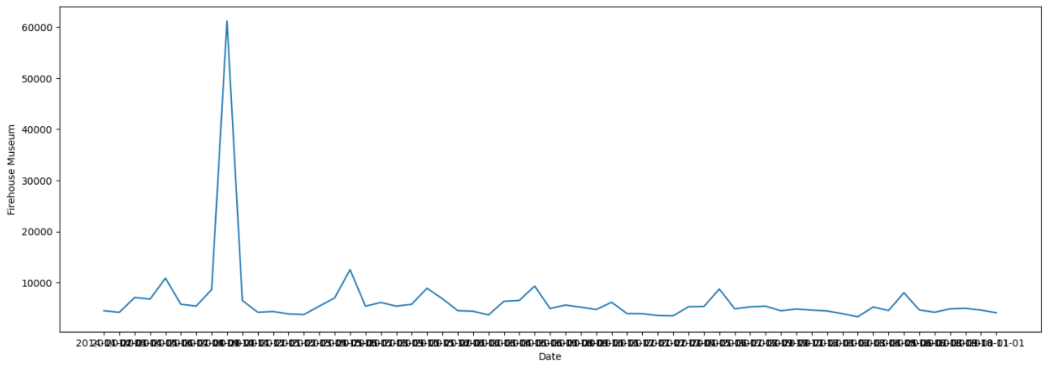
Supposons que nous souhaitons afficher le nombre des visiteurs du musée 'Firehouse Museum' (3em colonne) en fonction des mois. Pour faire ceci, nous devons faire appel à *sns.lineplot*. Les arguments optionnels *x* et *y* prennent respectivement les colonnes / Series à tracer :

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



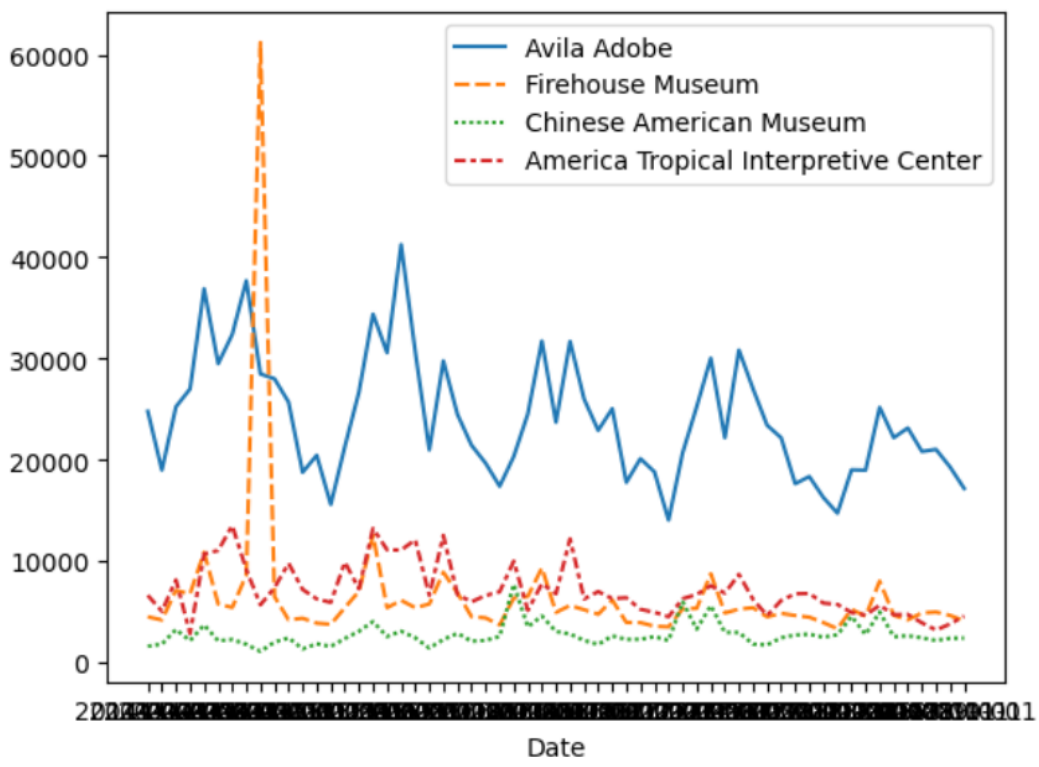
Comme annoncé initialement, *seaborn* dépend de *matplotlib.pyplot*. Par exemple, pour redimensionné la taille de la figure affichée, la méthode *pyplot.figure*() est utilisée avec l'argument optionnel *figsize* qui précise la largeur et la hauteur de la figure en pouce (1 pouce =100 pixels).

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



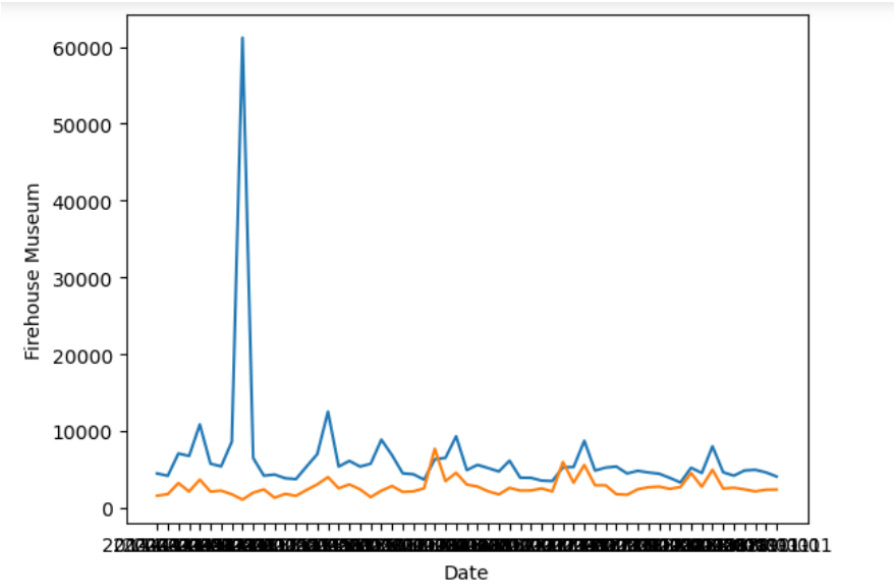
Si nous souhaitons tracer plusieurs colonnes en fonction du temps, il existe plusieurs façons de réaliser ceci. La plus simple est de mettre la colonne représentant l'axe des x comme index d'un DataFrame contenant les colonnes à tracer. Par la suite, au lieu d'utiliser les arguments *x* et *y*, il suffit 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)
```

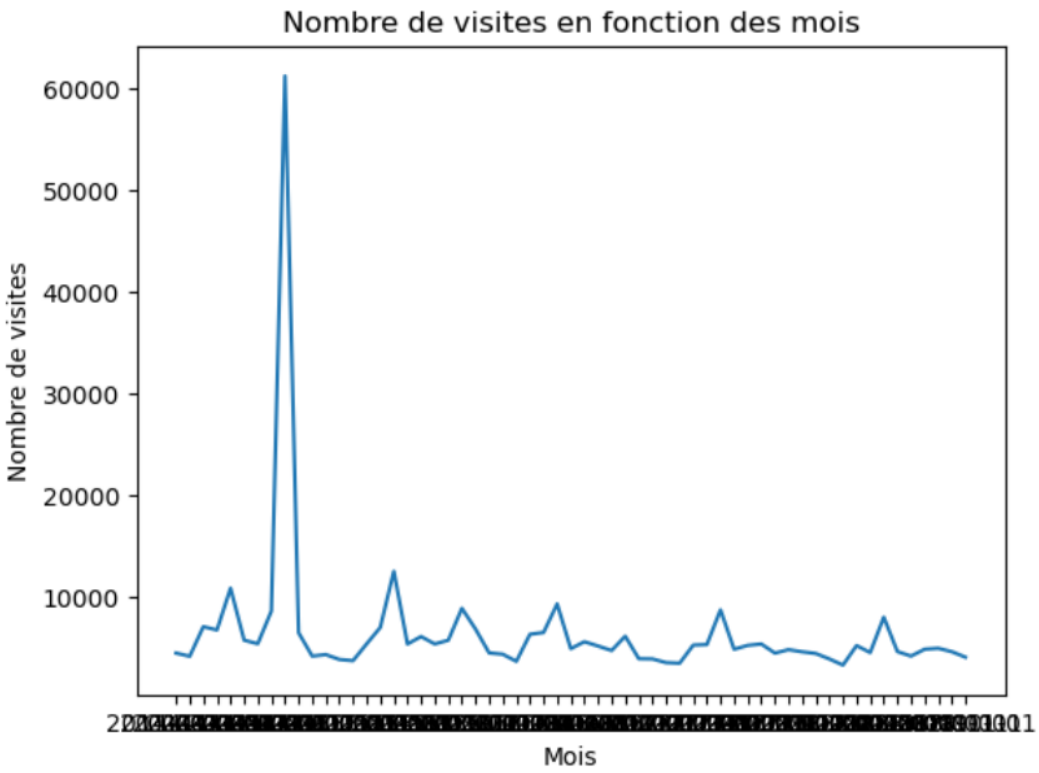


L'objet *matplotlib.axes.Axes.plot* possède une méthode *.get_figure()* qui retourne un objet *matplotlib.figure.Figure* . Ce dernier permet de sauvegarder une image avec la méthode *.savefig()*. L'exemple suivant enregistre la figure dans le fichier *resultat.png* :

```
plt.figure(figsize=(18,6))
axe=sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
fig=axe.get_figure()
print(type(fig))
fig.savefig("resultat.png")
```

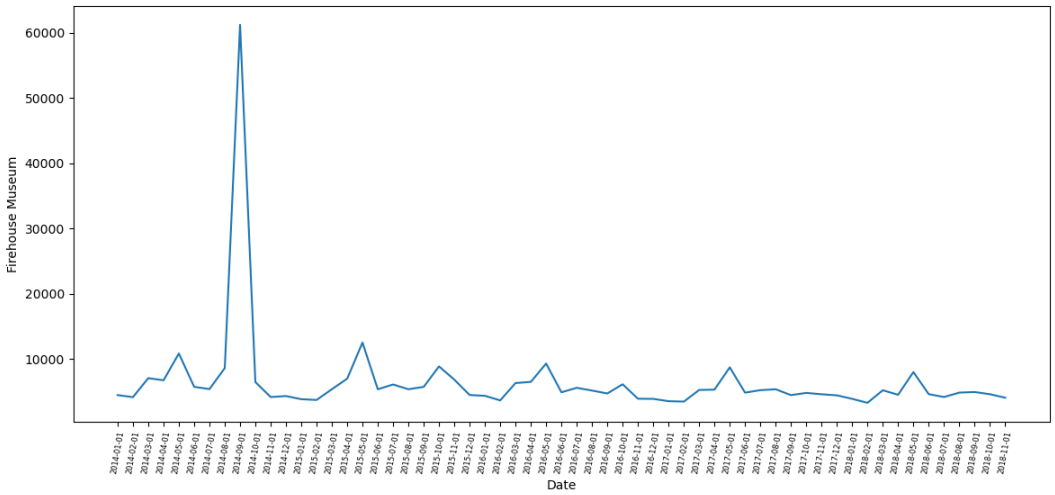
matplotlib.pyplot permet de modifier toutes les propriétés de la figure tel que le titre et les étiquettes des axes. Par défaut, les étiquettes des axes prennent le nom de la colonne / Series. En outre, il est possible de modifier les étiquettes des axes avec la méthode *pyplot.xlabel* et *pyplot.ylable*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



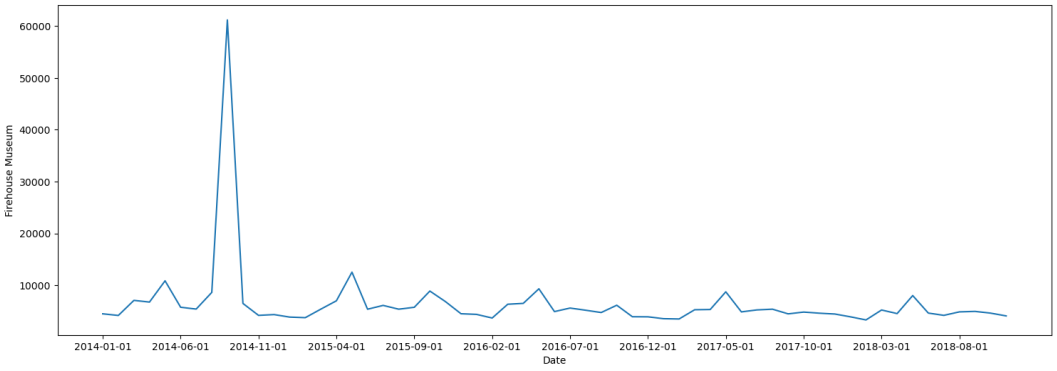
Il est apparent que les dates dans l'axe des x sont chevauchées. Pour résoudre ce problème il existe plusieurs solutions offertes par *matplotlib.pyplot*. Par exemple, augmenter la largeur de la figure et/ou diminuer le taille du police de l'axe des x peuvent présenter une solution. Une meilleur solution 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 *axis.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.

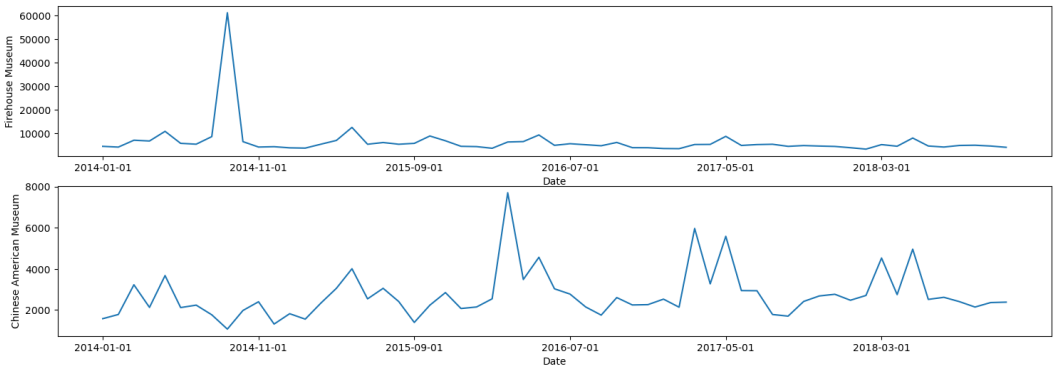
```
sns.lineplot(x=df.Date,y=df['Firehouse Museum'])
plt.show()
```

Subplots et FacetGrid

II

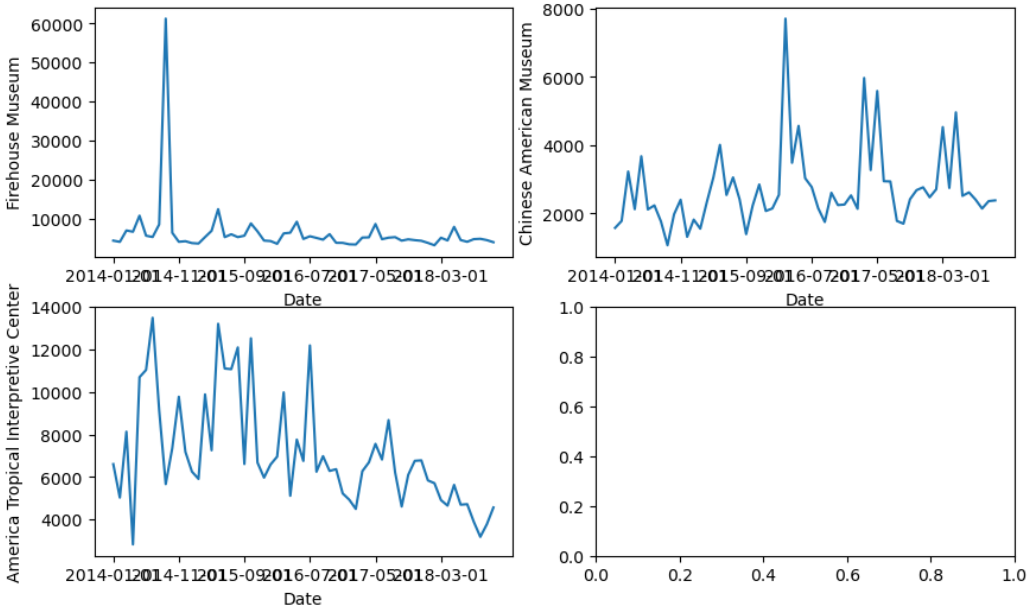
Alors que l'objet *matplotlib.axes.Axes.plot* représente le plan pour un seul graphe, il est possible de créer une grille contenant plusieurs plans avec la méthode *pyplot.subplots*. Cette méthode prend en paramètre le nombre de ligne et de colonne dans la grille. L'exemple suivant crée une grille d'une 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()
```



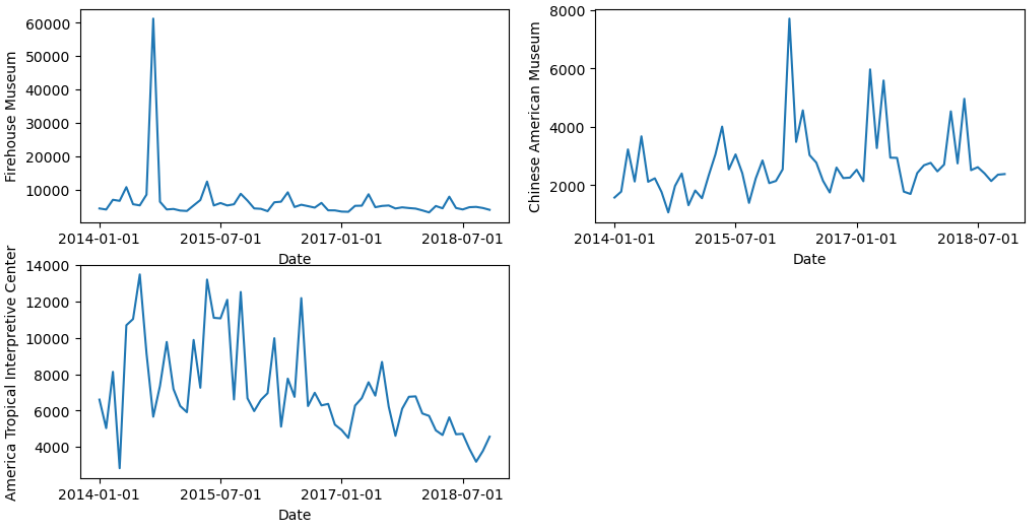
L'exemple suivant, crée une grille 2 x 2. Ainsi, contrairement à l'objet axes du premier exemple qui avais une seule dimension, l'objet axes retourné par *subplots* dans l'exemple suivant est une matrice de deux dimensions 2 x 2 :

```
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 utilisé 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
```

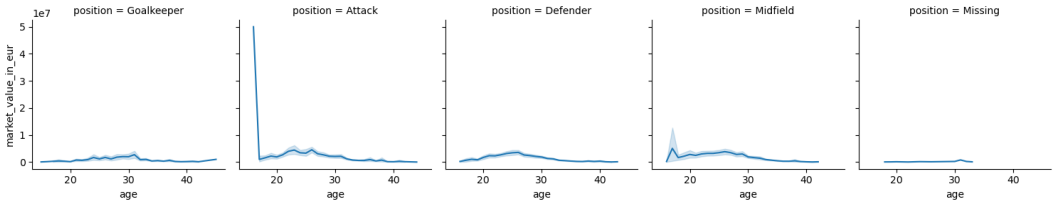


Parfois, il est utile de diviser les données en sous-ensemble selon une ou plusieurs colonnes catégoriques et tracer chaque graphique dans le même plan. Possible de faire ceci avec *pyplot.subplots* mais il y a un alternatif prédéfini avec *seaborn.FacetGrid*. L'utilité de cet alternative est mise en 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
```

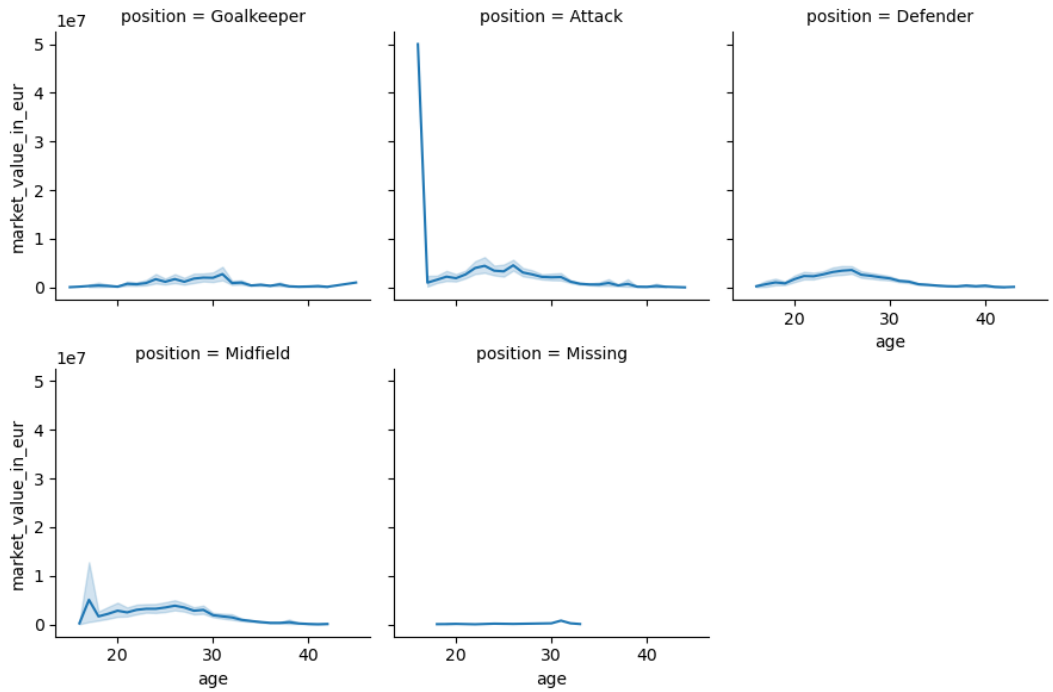
Le constructeur de la classe *seaborn.FacetGrid* prend en paramètre le *DataFrame* et le(s) nom(s) de la colonne pour diviser le Dataset avec les arguments *col* (comme dans l'exemple suivant "*position*" du Dataset *players.csv*) et/ou *row*. L'objet *FacetGrid* créé propose la méthode *map* qui prend 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")
```



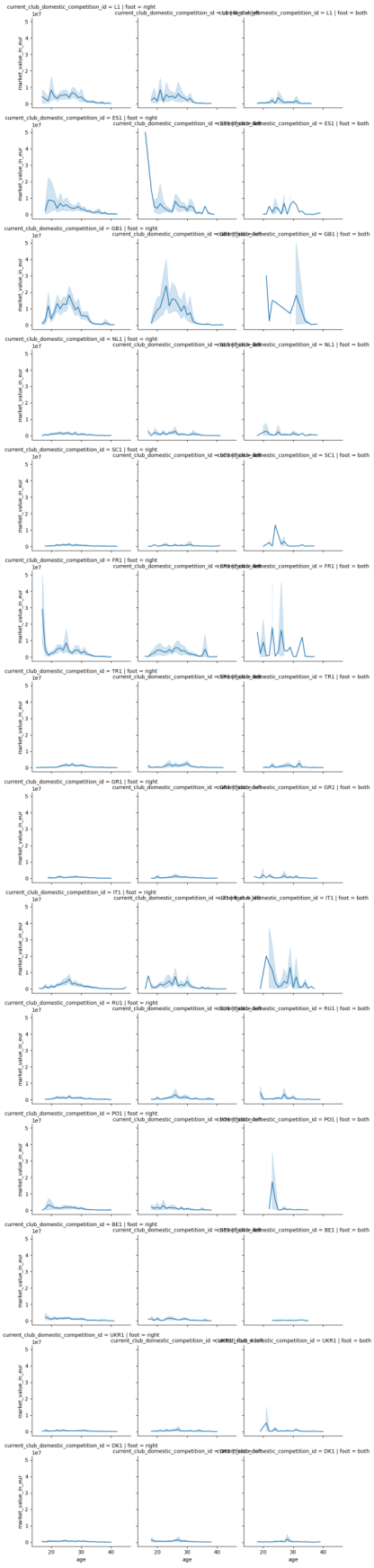
L'argument optionnel *col_wrap* du constructeur *FacetGrid* permet de spécifier le nombre maximal de figures par ligne (dans cet exemple *col_wrap*=3) :

```
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* [11*] 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.

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

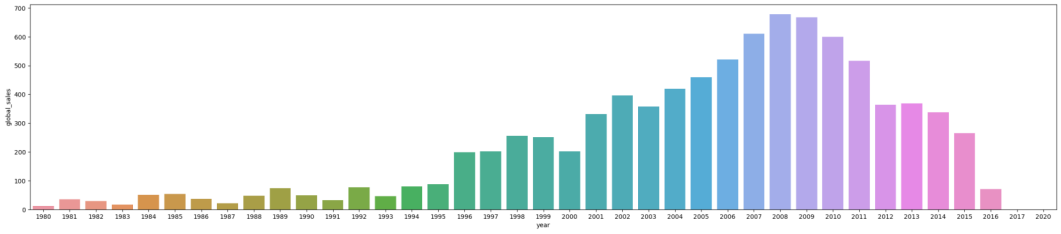
	rank	name	platform	year	genre	publisher	na_sales	eu_sales	jp_sales	other_sales	global_sales
0	1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	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 catégoriques, 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*.

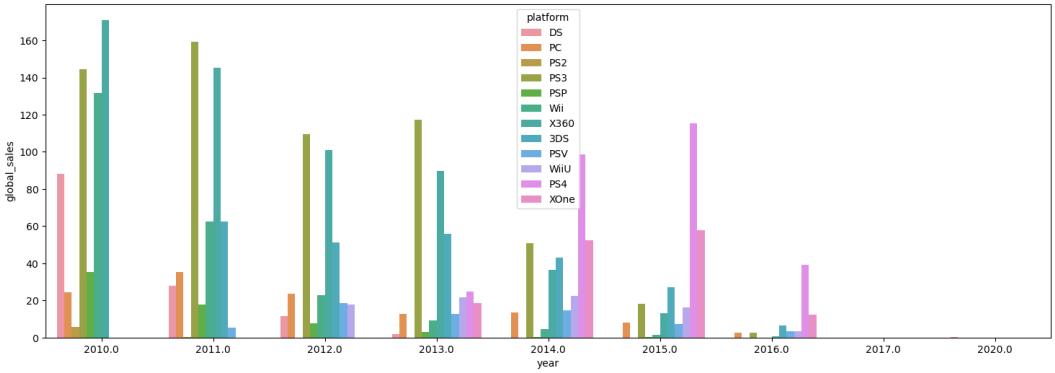
Supposant que nous souhaitons tracer la quantité des copies vendues chaque année. Ici bien-sur, nous avons une variable discrète (*global_sales*) tracée en fonction d'une variable discrète (*year*). La fonction *seaborn.barplot* permet de faire ce-ci :

```
1 plt.figure(figsize=(30,6))
2 ventes_par_platform=df.groupby('year').global_sales.sum()
3 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.

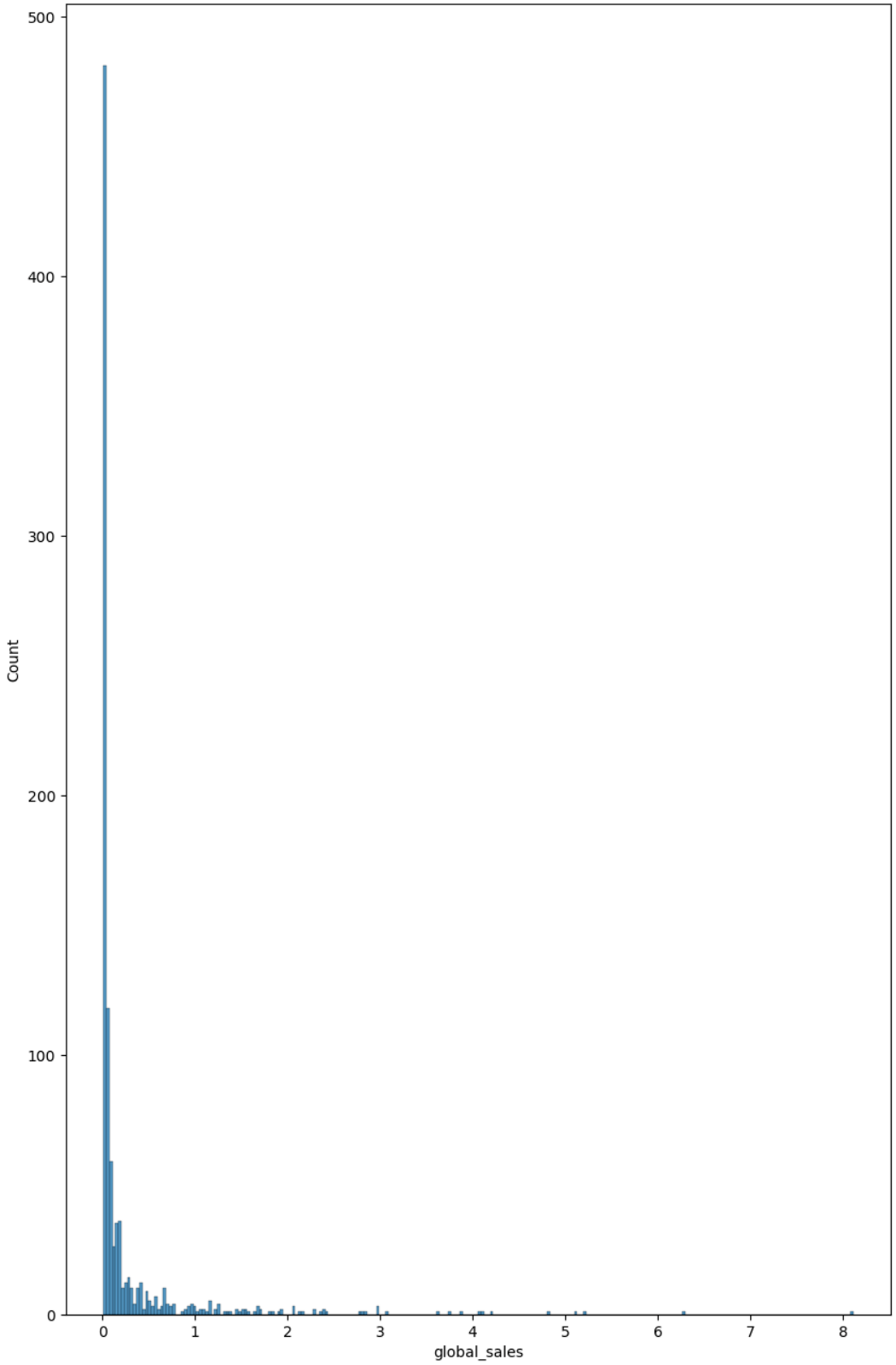
```
1 plt.figure(figsize=(18,6))
2 df10=df[df.year>=2010]
3 ventes_par_platform=df10.groupby(['year','platform']).global_sales.sum().
  reset_index(['platform',"year"])
4 sns.barplot(x="year",y="global_sales",hue="platform",data=ventes_par_platform)
```



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).

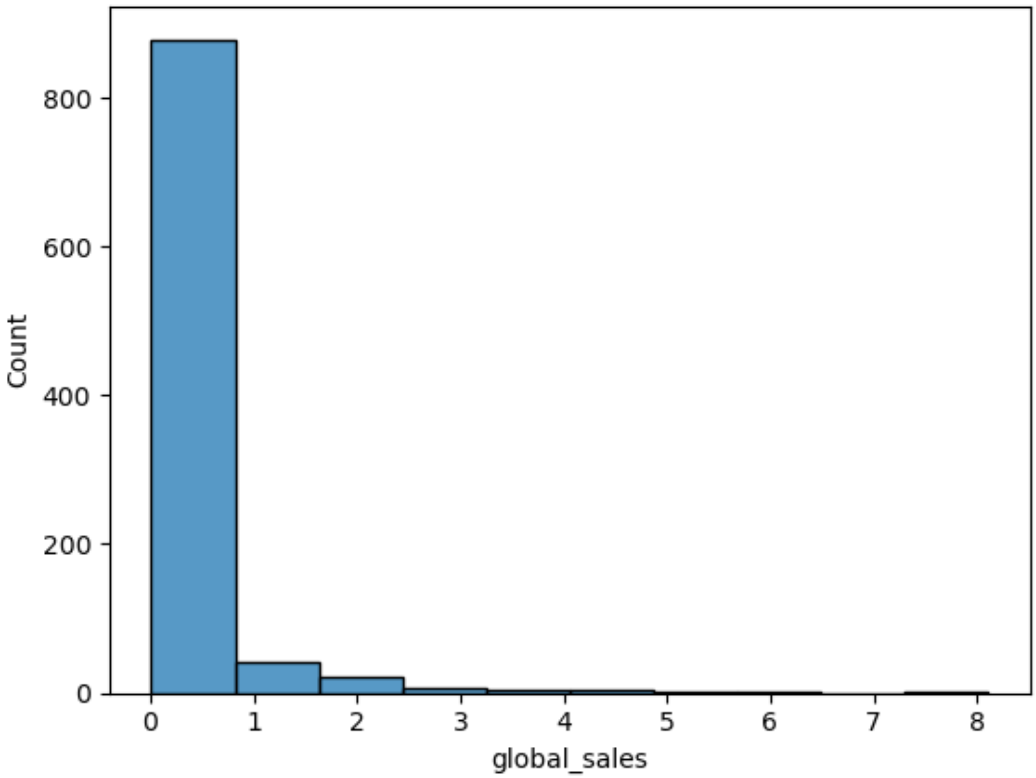
Par exemple, traçons la distribution des ventes des jeux sur PC avec la méthode *seaborn.histplot*: celle-ci divise la variable discrète en intervalles de la même taille et trace le nombre d'observation appartenant à chaque intervalle.

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



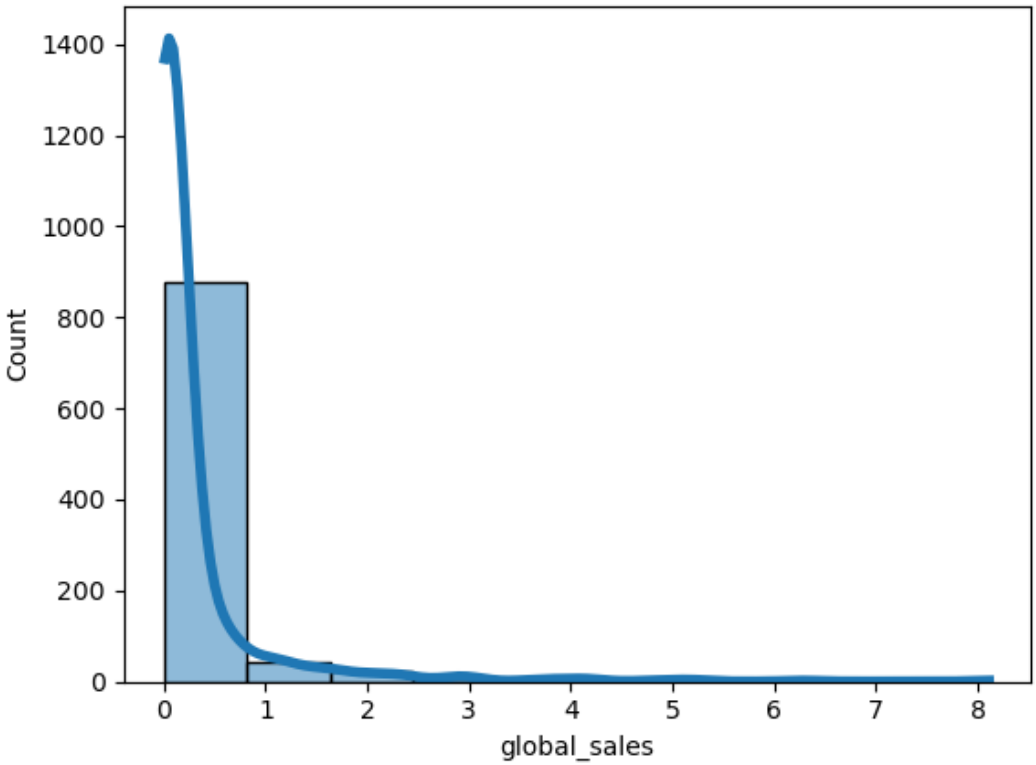
Comme dans *cut*, nous pouvons utiliser le paramètre *bins* pour spécifier le nombre de intervalles ou les intervalles spécifiques à tracer.

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



Pour afficher la fonction de densité estimée 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

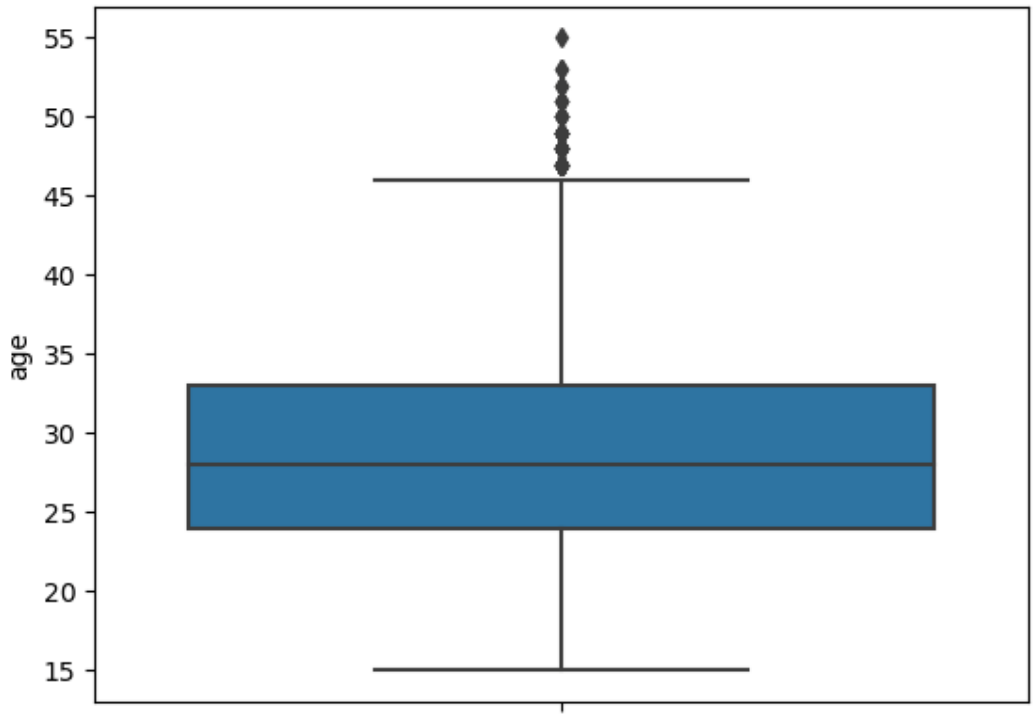
Boxplot

IV

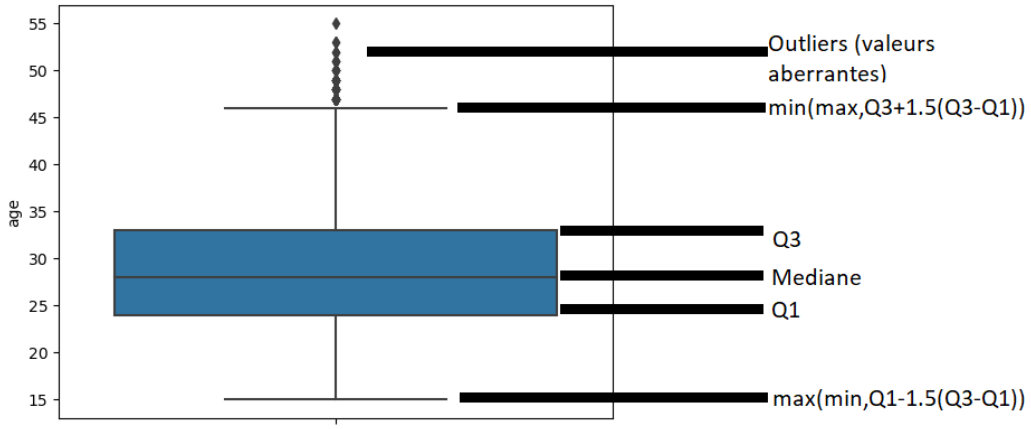
Un diagramme *Boxplot* (*whisker plot*) montre la distribution des données quantitatives de manière à faciliter les comparaisons entre les variables ou entre les niveaux d'une variable catégorielle. Le cadre de ce graphique montre les quartiles de l'ensemble de données tandis que les moustaches s'étendent 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*):

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

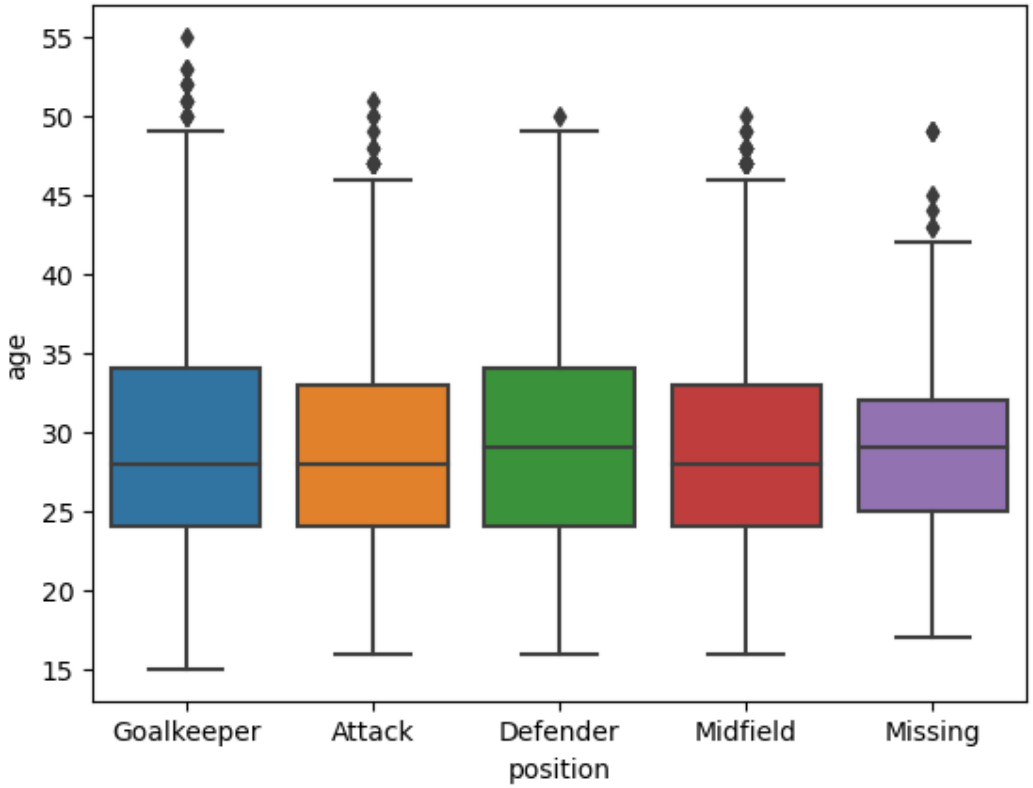


Pour rappel, la *médiane* c'est l'élément placé au *milieu des valeurs après le trie*. *Q1* c'est l'élément placé au milieu de la partie inférieure (entre médiane et le minimum) *après le trie* et *Q3* c'est l'élément placé au milieu de la partie supérieure (entre médiane et le maximum) *après le trie*. 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.

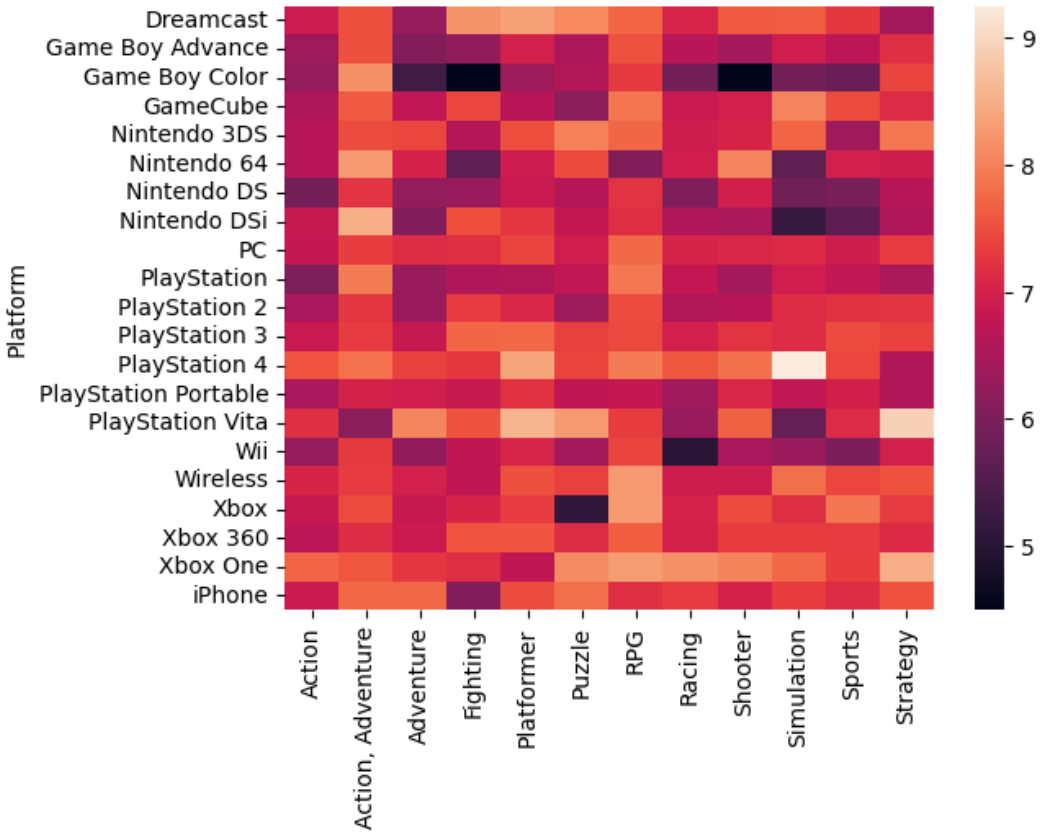
Prenant le DataSet *ign_scores.csv*[12*], qui montre pour chaque catégorie de jeux, le score moyen par plate-forme :

```
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

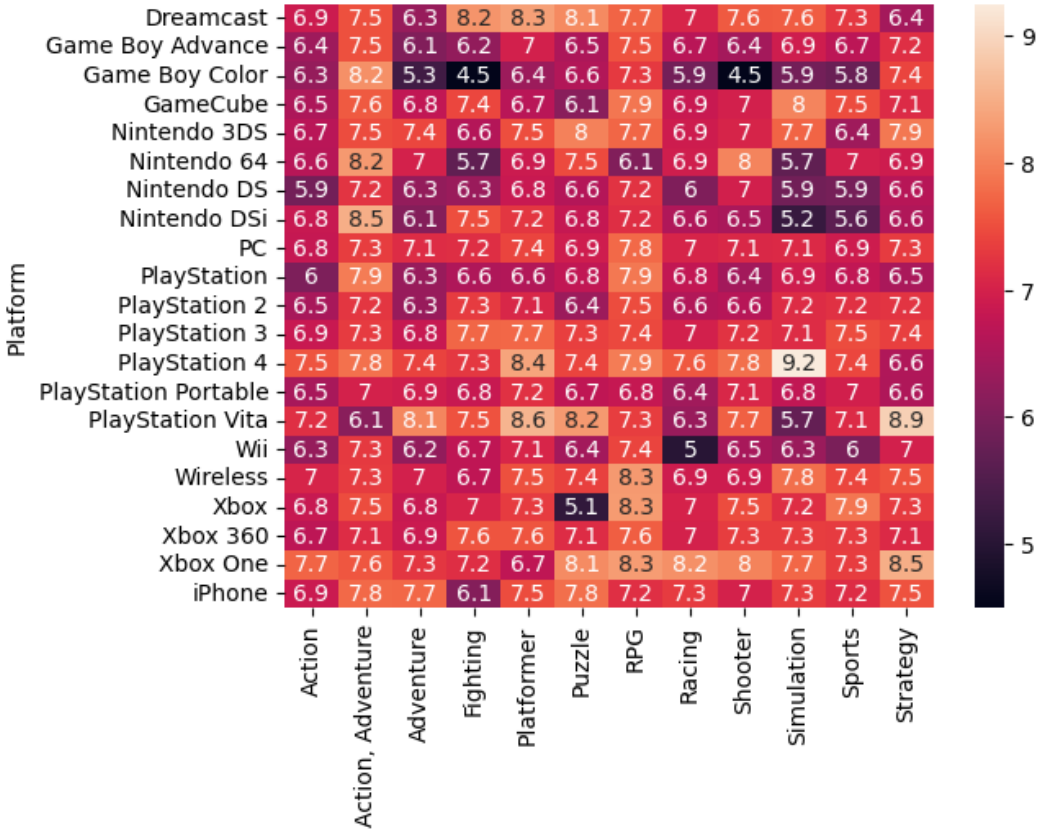
Pour tracer le graphe *Heatmap* des scores selon le genre (*colonne*) et la plate-forme (*index*)

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



Les valeurs du Data Set sont encodées avec une 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

Nuage de points (scatterplot)

VI

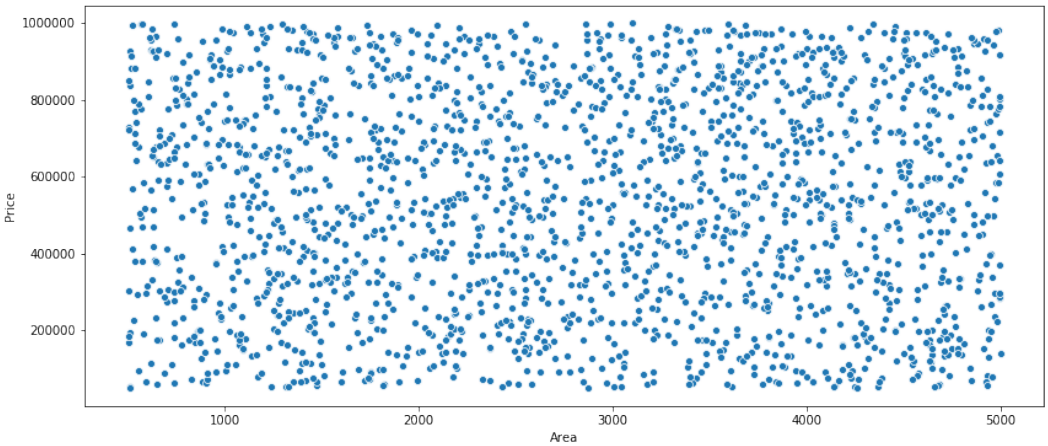
Un graphe très similaire au *lineplot* est le *scatterplot*. Ce graphe est utile pour identifier la relation entre deux variables / colonnes. Prenant le DataSet *house_prices.csv*[12*] :

```
df=pd.read_csv('house_prices.csv')
df.head()
```

	Id	Area	Bedrooms	Bathrooms	Floors	YearBuilt	Location	Condition	Garage	Price
0	1	1360	5	4	3	1970	Downtown	Excellent	No	149919
1	2	4272	5	4	3	1958	Downtown	Excellent	No	424998
2	3	3592	2	2	3	1938	Downtown	Good	No	266746
3	4	966	4	2	2	1902	Suburban	Fair	Yes	244020
4	5	4926	1	4	2	1975	Downtown	Fair	Yes	636056

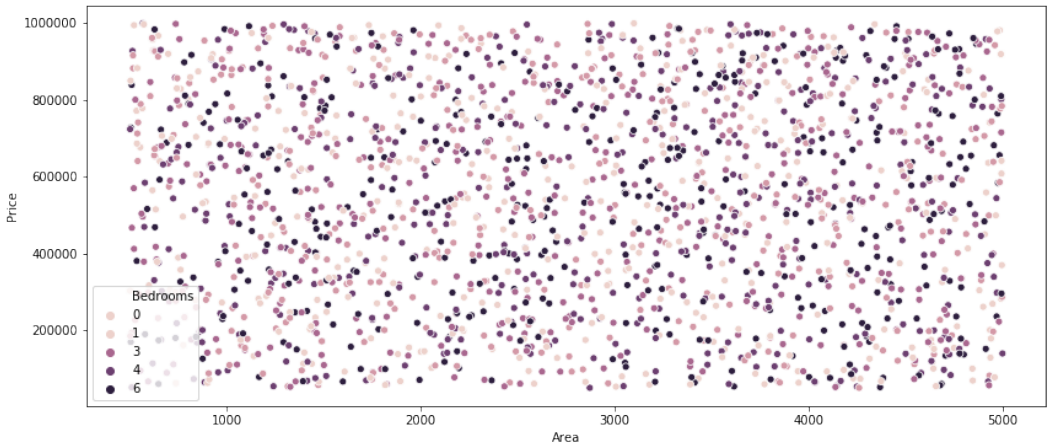
Le code suivant trace un *scatterplot* illustrant la relation entre la colonne *Area* et *Price* :

```
plt.figure(figsize=(14,6))
sns.scatterplot(x="Area",y="Price",data=df)
```



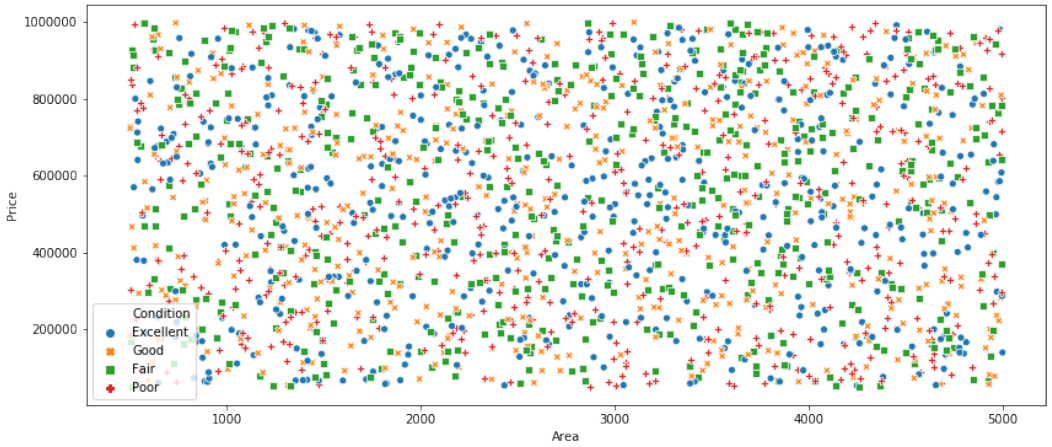
Pour diviser les observations selon la colonne *Bedrooms* (nombre de chambres), on utilise le paramètre *hue* comme d'autre graphiques vus précédemment:

```
plt.figure(figsize=(14,6))
sns.scatterplot(x="Area",y="Price",hue="Bedrooms",data=df)
```



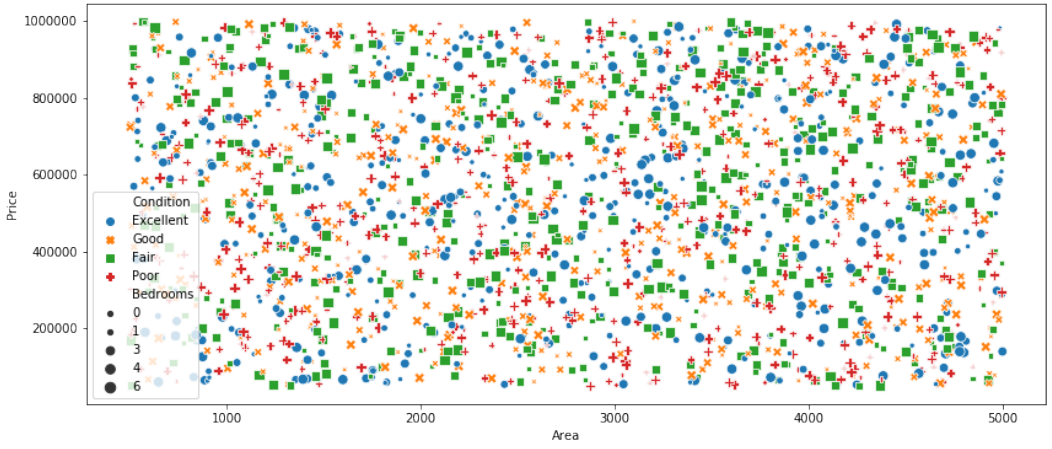
Pour définir le marqueur de point selon une colonne, le paramètre optionnel *style* est utilisé. Dans le code suivant, le marqueur des points est définie selon la colonne *Condition*:

```
plt.figure(figsize=(14,6))
sns.scatterplot(x="Area",y="Price",style="Condition",hue="Condition",data=df)
```



Il est aussi possible de définir la taille des marqueurs selon une colonne avec le paramètre *size*. L'exemple suivant illustre comment définir les tailles des marqueurs selon la colonne *Bedrooms* :

```
plt.figure(figsize=(14,6))
sns.scatterplot(x="Area",y="Price",style="Condition",hue="Condition",size=
 "Bedrooms",data=df)
```



Pour terminer, le paramètre optionnel *sizes* permet de définir l'échelle des tailles des marqueurs avec un tuple :

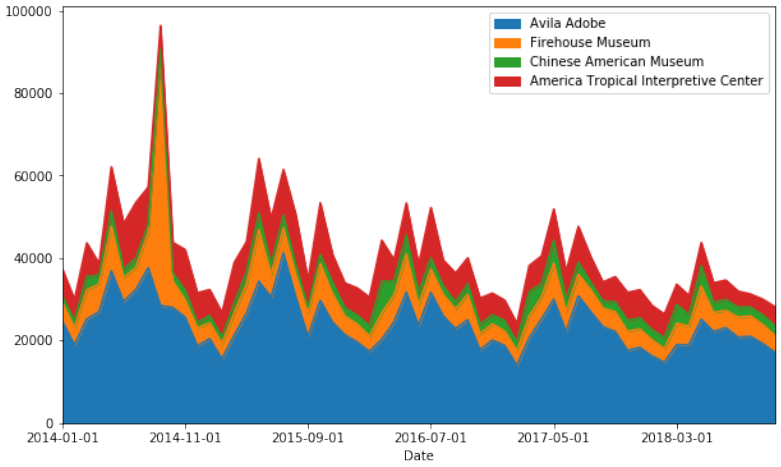
```
1 plt.figure(figsize=(14,6))
2 sns.scatterplot(x="Area",y="Price",style="Condition",hue="Condition",size=
  "Bedrooms",sizes=(10,150),data=df)
```

Area plot

VII

Le graphe de type Area plot est utilisé pour visualiser la différence entre l'évolution de deux/plusieurs variables au fils du temps. L'amplitude d'une variable est représentée par le surface entre la courbe de la variable et celle de la variable en dessous. Prenant l'évolution des visites des musés dans *museum_visitors.csv*. La façon la plus simple d'afficher un area plot est la méthode *.plot()* du DataFrame avec le paramètre *kind="area"*.

```
df.reset_index().set_index('Date',inplace=True)
df.plot(kind="area",figsize=(10,6))
```



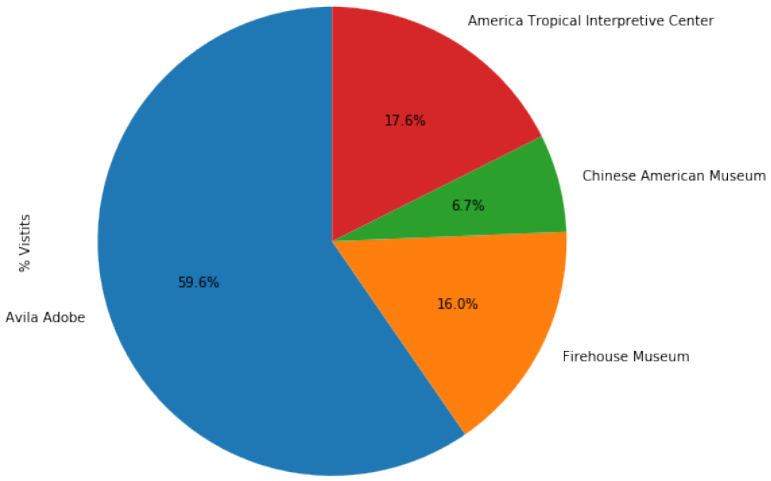
Pie plot et Donut plot

VIII

Pie plot est l'un des graphes les plus utilisé servant à illustrer les proportion numériques / répartition en pourcentages. Le code suivant trace les pourcentages des visites cumulées dans le DataSet *museum_visitors.csv*.

- *kind*="pie" pour tracer un Pie plot ;
- *autopct* pour afficher les pourcentages ;
- *startangle* permet de définir l'angle de début de la première proportion ;

```
proportion_des_visites=df.iloc[:,1:].sum() # prendre seulement les colonnes nombre de visites par musée
proportion_des_visites.name="% Vistits"
proportion_des_visites.plot(kind="pie",autopct='%1.1f%%',figsize=(8,8),startangle=90)
```

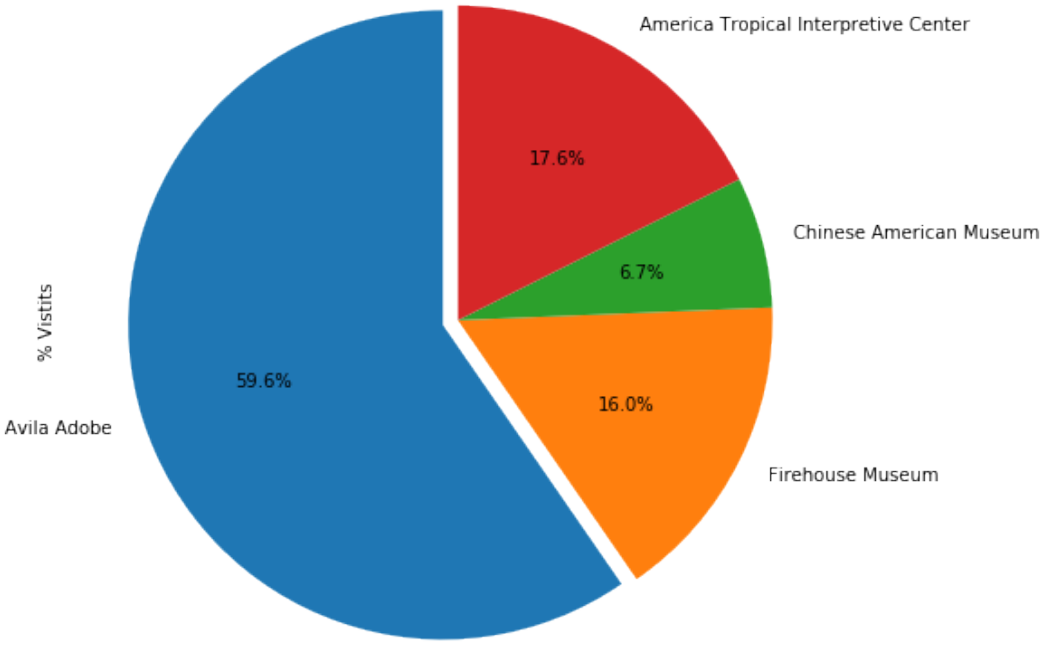


D'autres paramètres peuvent être utilisés pour personnaliser l'affichage comme :

- *explode* : qui permet de couper un ou plusieurs proportions du graphe;
- *pctdistance* : distance entre le pourcentage et le centre ;
- *labeldistance* : distance entre le label et le centre ;
- *colors :* tableau contenant les couleurs des proportions ;
- *wedgeprops* : offre la possibilité de styler le Pie plot. La clé *edgecolor de* ce dictionnaire définie la couleur des frontières entre les proportions ;

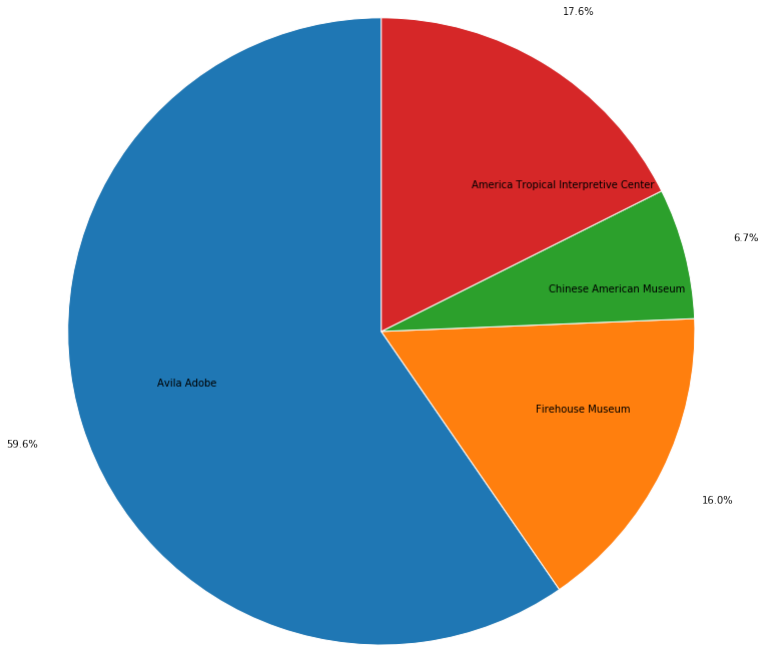
Le code suivant affiche un Pie plot avec les label à l'intérieur des surfaces, les pourcentages à l'extérieur. Les frontières entre les proportions sont colorées en blanc et la deuxième proportion est découpée grâce à *explode*.

```
proportion_des_visites.plot(kind="pie",autopct='%1.1f%%',figsize=(8,8),startangle=
90, explode= [.05]+[0]*(proportion_des_visites.shape[0]-1))
```



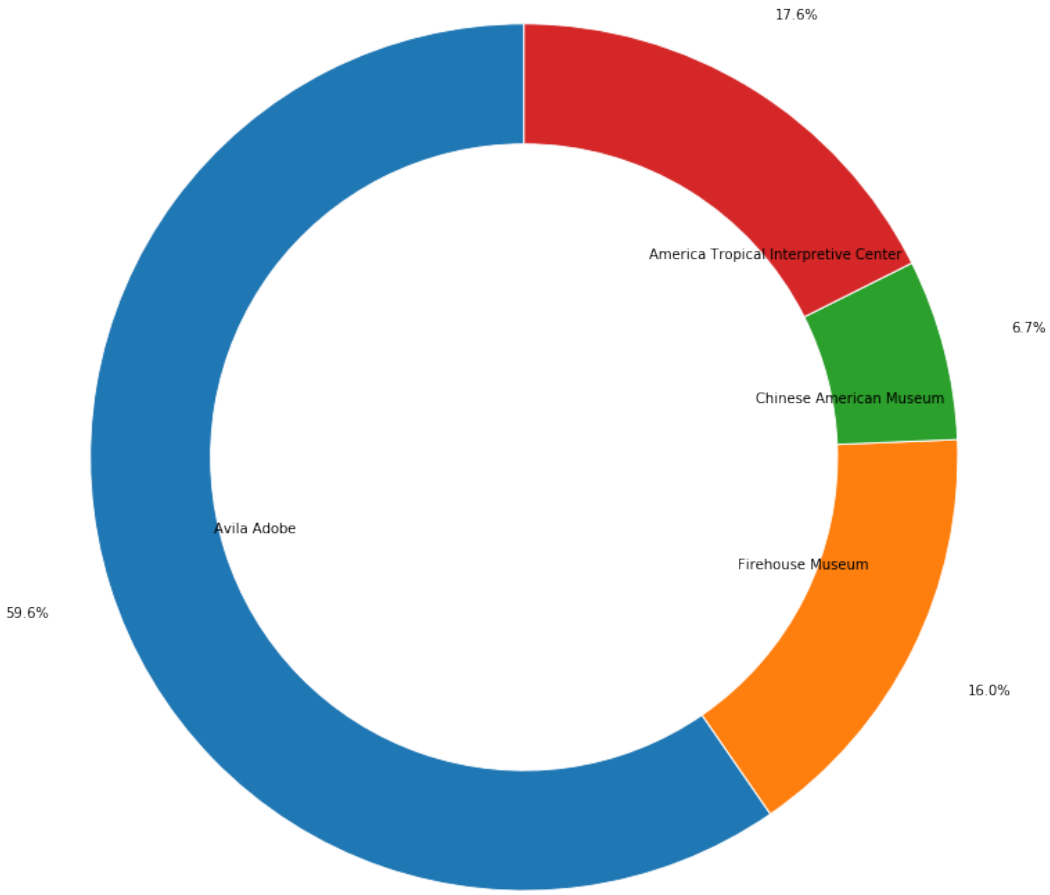
L'exemple utilise *axes.pie* au lieu de *DataFrame.plot(kind="pie")*. Par contre, les deux méthodes admettent les mêmes paramètre (sauf *figsize*) et produisent le même résultat :

```
fig, ax = plt.subplots(figsize=(8,8))
ax.pie(proportion_des_visites,autopct='%1.1f%%',startangle=90,pctdistance=1.2,
labeldistance=0.55,labels=proportion_des_visites.index, radius=1.8,wedgeprops={
"edgecolor":'white'})
```



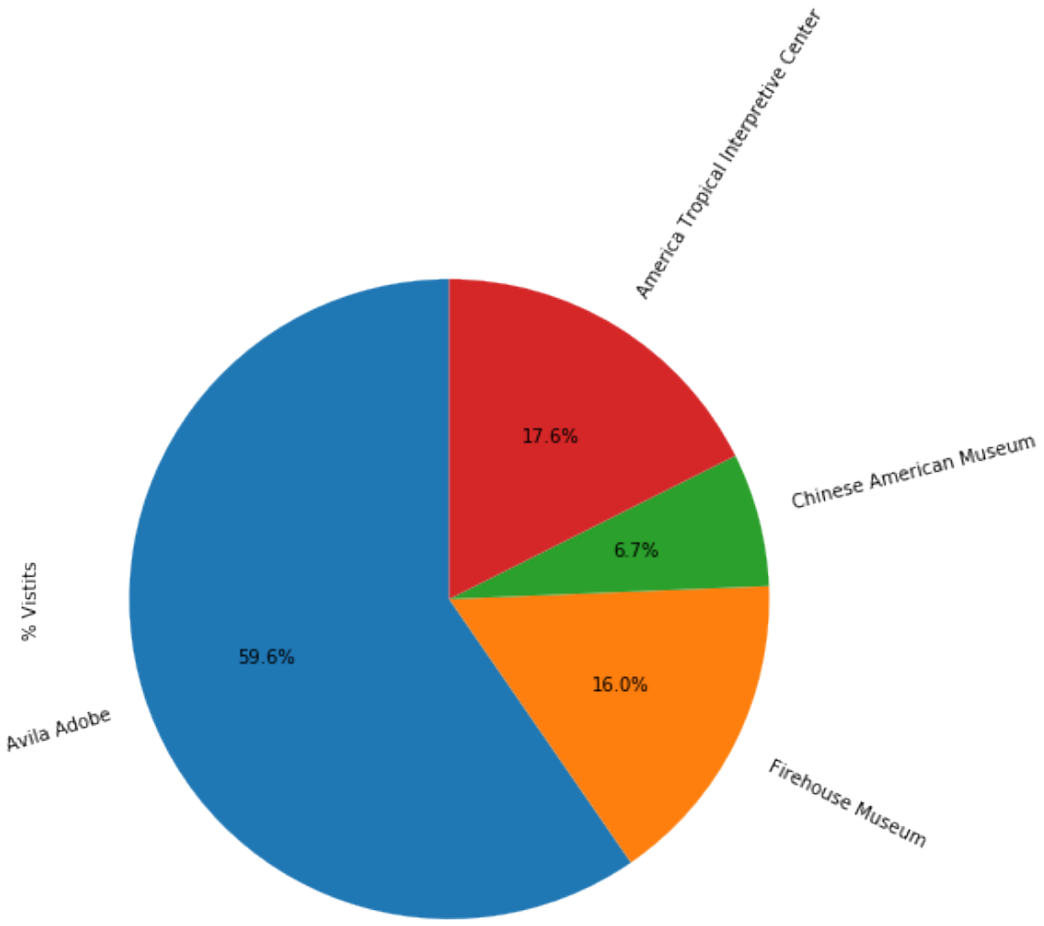
Le Donut plot est un sous type du Pie plot avec un cercle au milieu pour laisser de l'espace vide à d'autres information ou un autre graphe imbriqué. Pour transformer le graphe précédant en Donut plot, il suffit d'ajouter la clé *width* au *wedgeprops* avec le diamètre du cercle au milieu.

```
1 fig, ax = plt.subplots(figsize=(8,8))
2 ax.pie(proportion_des_visites,autopct='%1.1f%%',startangle=90,pctdistance=1.2,
  labeldistance=0.55,labels=proportion_des_visites.index, radius=1.8,wedgeprops={
  "width":0.5,"edgecolor":'w'})
```



Que ça soit pour le Donut plot ou Pie plot, le paramètre optionnel *rotatelabels* mis à *True* permet d'afficher le label de chaque proportion selon l'angle correspondant :

```
1 proportion_des_visites.plot(kind="pie",rotatelabels=True,autopct='%1.1f%%',figsize
  =(8,8),startangle=90)
```


America Tropical Interpretive Center
17.6%
Chinese American Museum
6.7%
% Vistits
59.6%
16.0%
Avila Adobe
Firehouse Museum

Word Cloud

IX

Le word cloud est un graphe permettant d'afficher des données textuelles. Dans ce graphe, le police d'un terme affiché dans le graphe dépend du nombre d'occurrence de celui-ci dans le texte. Ainsi, les termes les plus fréquents / importants sont affichés avec un police plus large.

Pour installer le module *wordcloud*[14*] vous devez d'abord installer la version *9.5* de *pillow*[15*] car c'est la dernière version compatible avec ce module. Les versions plus récentes de *pillow* ne sont pas compatibles avec ce module :

```
pip3 install Pillow==9.5.0
```

Par la suite, il est nécessaire d'installer le module wordcloud :

```
pip install wordcloud
```

Commençons par la lecture du texte d'un fichier. La fonction open permet de lire un fichier dans un mode spécifié. Par défaut, la fonction *open* ouvre le fichier en mode lecture. La méthode *read()* de l'objet retourné permet de lire et retourner tout le contenu du fichier. Le paramètre optionnel *encoding* l'encodage du fichier lu :

```
text=open("tcpip.txt", encoding='utf-8').read()
```

Les deux éléments nécessaires pour imprimer un wordcloud minimal sont la classe *WordCloud* et l'objet *matplotlib.pyplot* :

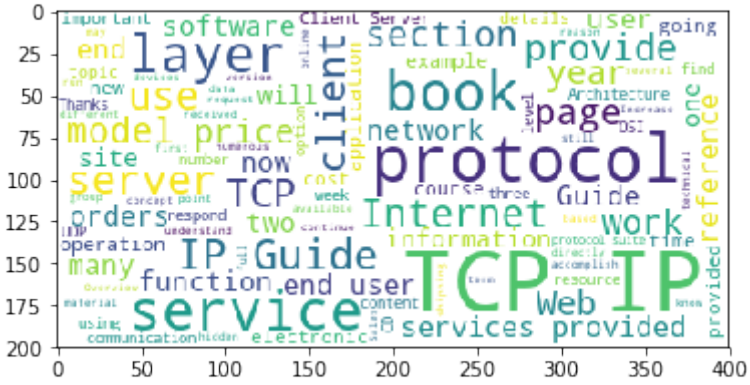
```
from wordcloud import WordCloud
from matplotlib import pyplot as plt
```

Le constructeur *WordCloud* peut prendre plusieurs paramètres optionnels. Dans cet exemple :

- *max_font_size* représente la taille du police maximale lors de l'affichage ;
- *max_words* représente le nombre maximale de mots affichés dans le graphe ;
- *background* représente la couleur de l'arrière plan du graphe ;

La méthode *.generate(texte)* de l'objet créé par le constructeur *WordCloud* génère et renvoie le *wordcloud*. Le résultat retourné est une image sous forme matricielle qui peut être affichée avec *plt.imshow()* :

```
word_cloud=WordCloud(max_font_size=50, max_words=100, background_color="white").
generate(text)
plt.imshow(word_cloud)
```



Le module wordcloud propose aussi une liste de mots vides qui doivent être ignorés lors du traitement du texte. L'ensemble des mots vides dépend du langage (français, anglais, arabe) :

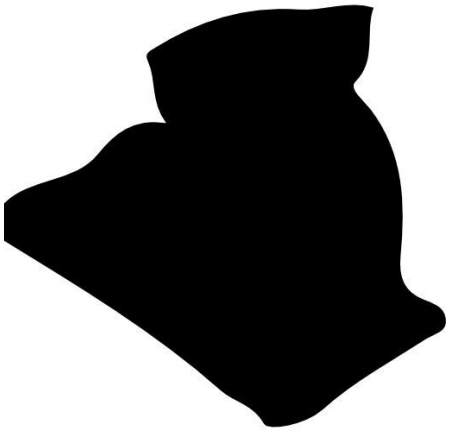
- Par exemple, en français, les mots : le ,la, est, que alors, etc sont considérés comme des mots vides même s'ils se répètent très fréquament ;
- En anglais, les mots : the, he, are, you, then, etc sont considérés comme des mots vides pour les mêmes raisons ;

Les mots vides du langage anglais sont inclus dans la collection *wordcloud.STOPWORDS*. Le code suivant importe la liste des mots vides, transforme celle-ci en *hashset* avec la fonction *set*, et puis utilise cette collection dans le paramètre optionnel *stopwords* du constructeur *WordCloud*.

plt.axis("off") permet de supprimer les axes x et y de la figure :

```
1 from wordcloud import STOPWORDS
2 mots_vides=set(STOPWORDS)
3 word_cloud=WordCloud(stopwords=mots_vides,max_font_size=50, max_words=100,
  background_color="white").generate(text)
4 plt.imshow(word_cloud)
5 plt.axis("off")
```

Le module *wordcloud* permet aussi d'afficher le graphe avec un masque qui peut être lu à partir d'une image. Ce dernier doit définir les zones d'affichage du texte. Une zone blanche n'est pas utilisée dans l'affichage (valeur du pixel =255). En revanche, toute zone avec une couleur différente de la couleur blanche (!=255) peut contenir le texte du word cloud. Prenons la carte de l'Algérie comme masque :



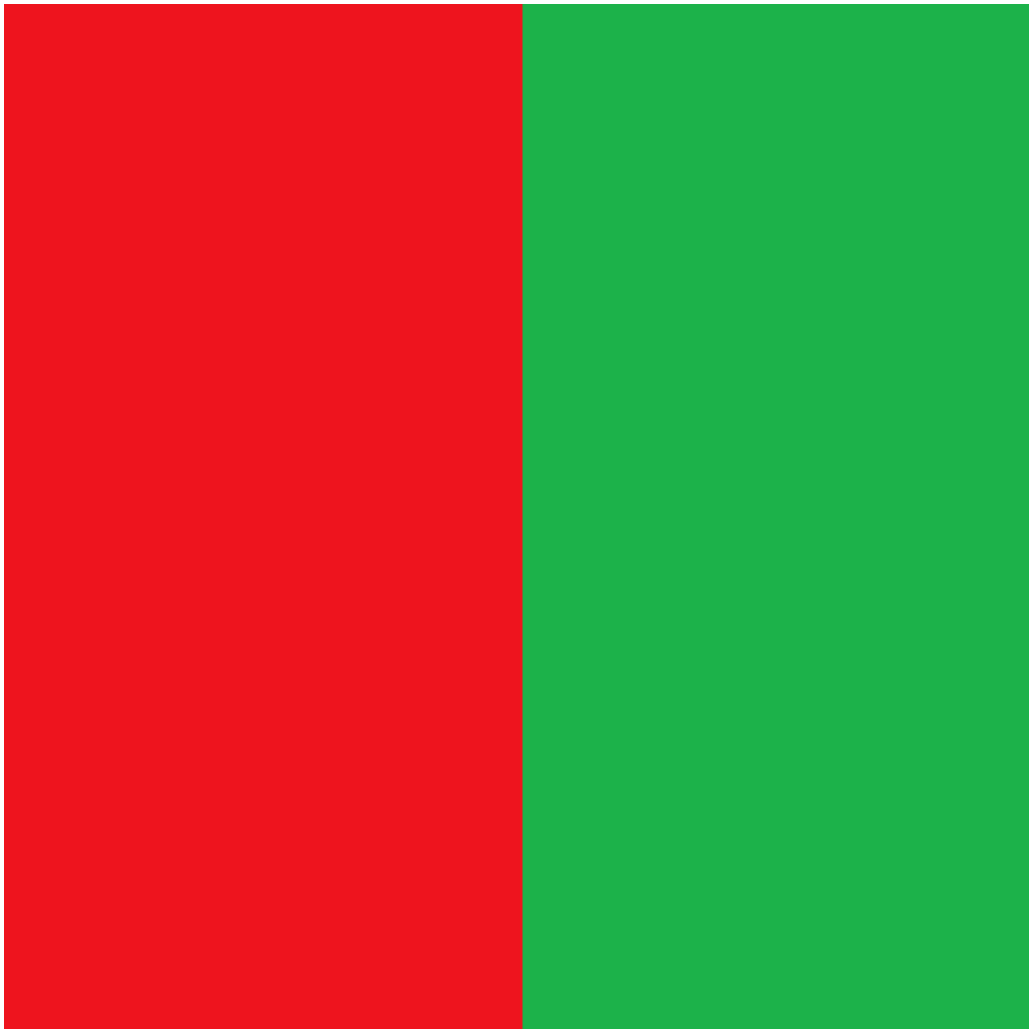
Pour ouvrir, on utilise *numpy* et *PIL.Image*. Le résultat retourné est une matrice représentant l'image "*masque algerie.jpg"* avec les valeurs de couleurs pour chaque pixel :

```
1 from PIL import Image
2 import numpy as np
3 masque = np.array(Image.open("masque algerie.jpg"))
```

Le code est identique au word cloud précédent sauf que l'image est utilisée comme masque avec le paramètre optionnel *mask dans* le constructeur *WordCloud*:

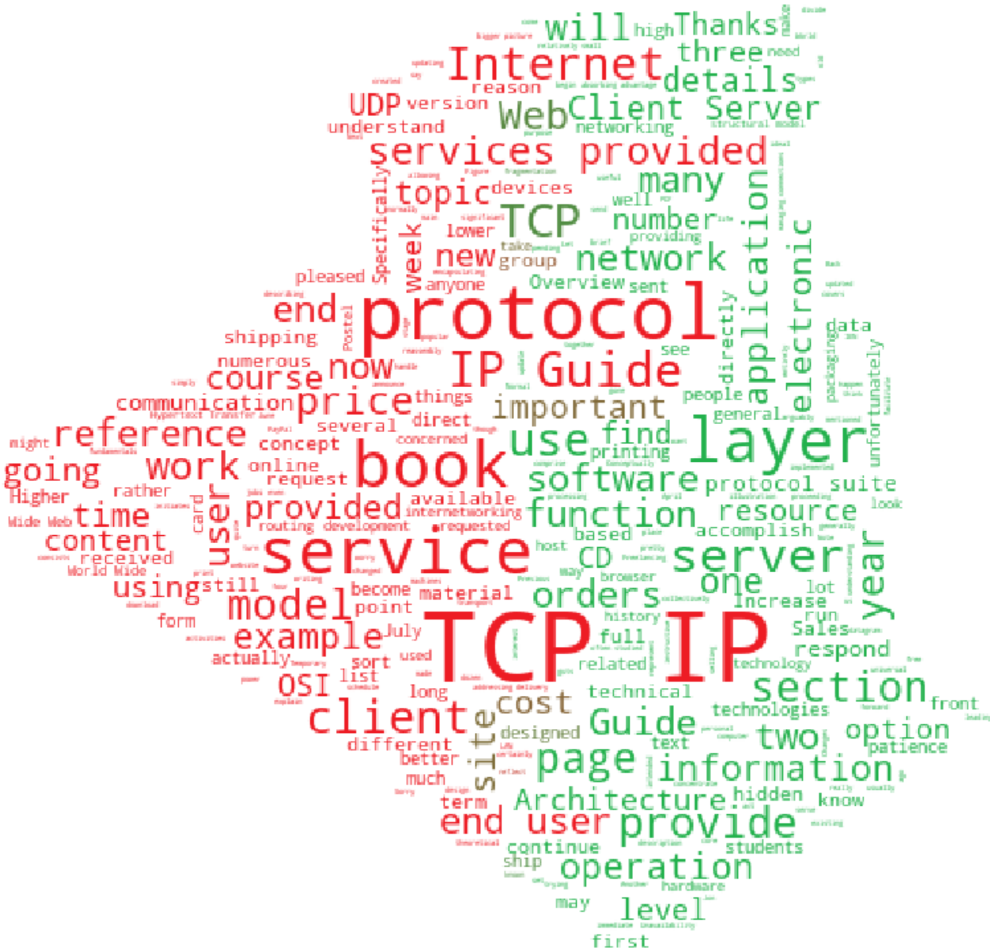
```
1 plt.figure(figsize=(14,14))
2 word_cloud=WordCloud(max_font_size=50, max_words=1000, mask=masque,
  background_color="white").generate(text)
3 plt.imshow(word_cloud)
4 plt.axis("off")
```

Le module *wordcloud* fournie aussi une classe *ImageColorGenerator* qui permet de définir le schéma de coloriage. Ci dessous une image, *shcema_couleur.png*,de la même taille que le masque qui sera utilisée pour le constructeur *ImageColorGenerator :*

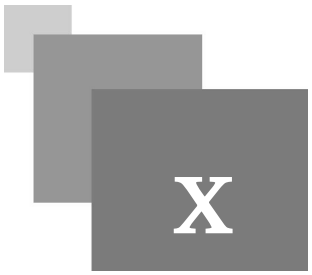


Le code suivant crée le schéma de coloriage selon l'image *shcema_couleur.png* et renvoie le résultat. Pour définir l'image comme schéma de coloriage du word cloud, la méthode *recolor()* du wordcloud avec le paramètre *color_func* sont utilisés. Le paramètre *color_func* prend le schéma de coloriage créé initialement avec *ImageColorGenerator* :

```
1 from wordcloud import ImageColorGenerator
2 schema_couleurs = ImageColorGenerator(np.array(Image.open("shcema_couleur.png")))
3 plt.figure(figsize=(14,14))
4 word_cloud=WordCloud(max_font_size=50, max_words=1000, mask=masque,
  background_color="white").generate(text)
5 plt.imshow(word_cloud.recolor(color_func=schema_couleurs))
6 plt.axis("off")
```


protocol
TCP IP
service
book
layer
Internet
IP Guide
client
services provided
server
reference
model
example
price
course
work
going
time
content
using
provided
user
end
now
topic
TCP
Web
Client Server
details
will
Thanks
three
network
application
electronic
use
software
function
resource
orders
section
information
provide
operation
Architecture
page
Guide
level
one
year
two
option
important
many
number
UDP
OSI
cost
site
end user

Plotly



Les librairies *matplotlib* et *seaborn* sont suffisantes pour la visualisation des données. Cependant, pour créer des visualisations interactives et les inclure dans des applications destinées aux décideurs, des modules plus sophistiqués sont nécessaires. Dans ce chapitre, nous allons introduire *Plotly*[16*], un module permettant de créer des graphes interactives ainsi que *Dash* qui permet de servir ces graphes dans des pages web sous forme d'un Dashboard.

Plotly est un module très similaire à *seaborn* mais avec des types de graphes supplémentaires et un affichage interactive. La commande suivante permet d'installer ce module :

```
conda install plotly::plotly
```

Prenant le DataSet *canada.csv*[17*] contenant le nombre d'immigrés vers Canada entre 2004 et 2013 :

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

	Unnamed: 0	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	...	2004	2005	2006	2007	2008	2009	2010	2011	20
0	0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	...	2978	3436	3009	2652	2111	1746	1758	2203	26
1	1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	...	1450	1223	856	702	560	716	561	539	6
2	2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	...	3616	3626	4807	3623	4005	5393	4752	4325	37
3	3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	...	0	0	1	0	0	0	0	0	
4	4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	...	0	0	1	1	0	0	0	0	

Particulièrement, prenant le nombre d'immigrés depuis l'Algérie vers Canada :

```
1 data_alg=df[df.OdName=="Algeria"].loc[:,"2004":].T.rename(columns={2:"Algeria"})
2 data_alg.head()
```

	Algeria
2004	3616
2005	3626
2006	4807
2007	3623
2008	4005

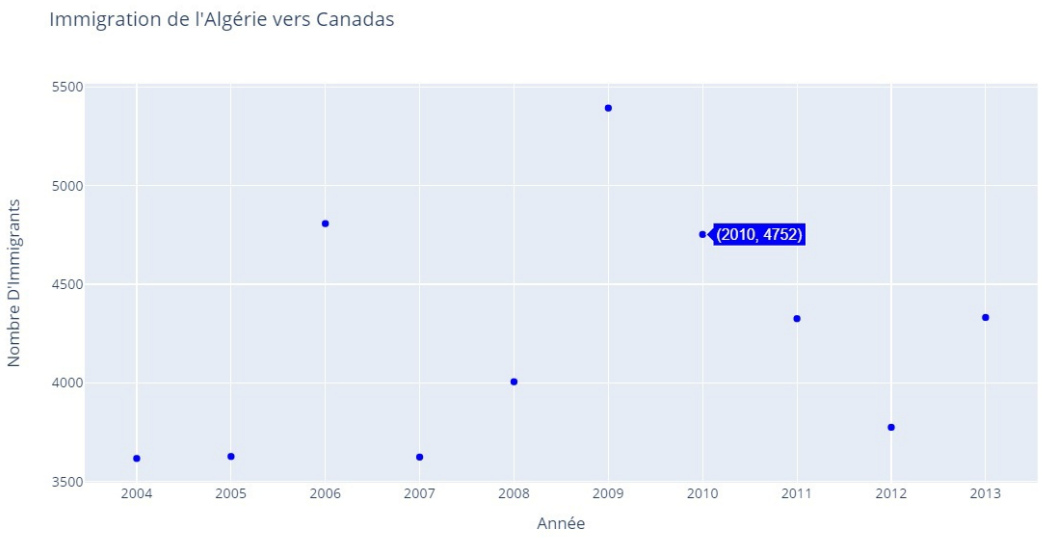
Pour tracer avec Plotly, deux objets sont fournies : *graph_objects* et *express*. Dans ce premier exemple, nous allons démontrer comment utiliser *graph_objects*.

- La méthode *Figure*() de l'objet *graph_objects* permet de créer un plan de traçage ;
- La méthode *add_trace()* permet d'ajouter le graphe passé en paramètre au plan ;

- La fonction *go.Scatter()* permet de créer un graphe d'une manière très similaire à *seaborn*. Le paramètre optionnel *mode="markers"* permet de spécifier que le graphe à créer est un scatter plot (sans ce paramètre, le graphe tracé sera un lineplot) ;

Contrairement au graphes produits par *seaborn* et *matplotlib*, celui créé avec Plotly est interactif (il est possible d'interagir avec les échantillons, zoomer, etc)

```
import plotly.graph_objects as go
fig=go.Figure()
fig.add_trace(go.Scatter(x=data_alg.index, y=data_alg.Algeria, mode='markers',
marker={"color":'blue'}))
fig.update_layout(title='Immigration de l\'Algérie vers Canadas' , xaxis_title=
'Année', yaxis_title='Nombre D\'Immigrants')
fig.show()
```



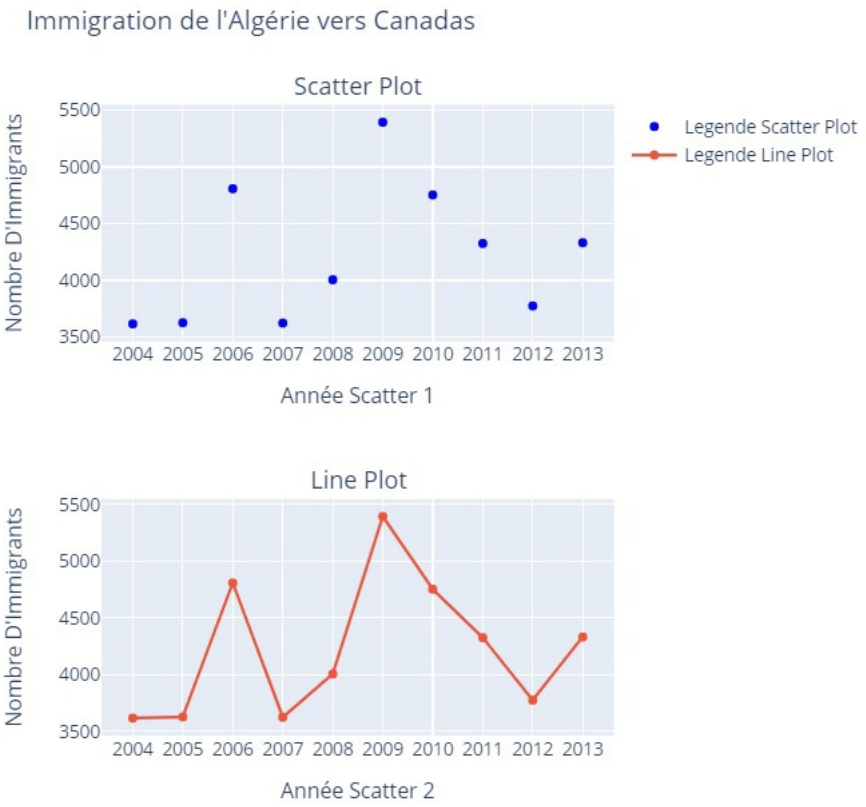
- Pour créer plusieurs sous-graphes dans la même figure, la méthode *make_subplots* est utilisée avec comm paramètres, le nombre de lignes et colonnes dans la grille (*rows/cols*) ainsi que la liste des titres des sous graphes (*subplot_titles*);
- Lors de l'invocation de *fig.add_trace()*, la position du graphe à ajouter est spécifiée avec les paramètres optionnels *row* et *col* ;
- Lors de la création du graphe, dans cet exemple go.Scatter() , il est possible de spécifier la légende attribuée au graphe avec le paramètre *name*. Par défaut, le graphe créé par go.Scatter est un lineplot (*mode="lines"*) ;
- Pour donner un titre à l'axe des x et y les méthodes *fig.update_xaxes* et *fig.update_yaxes* sont utilisées ave comme paramètres la position du sous-graphe désigné (row/col) et le titre de l'axe (*title_text*) ;
- Les paramètres optionnels *height* et *width* de *fig.update_layout()* permettent de spécifier la taille du plan en pixels;

```
from plotly.subplots import make_subplots
fig = make_subplots(rows=2, cols=1,subplot_titles=["Scatter Plot", "Line Plot"])

fig.add_trace(go.Scatter(x=data_alg.index, y=data_alg.Algeria, mode='markers',
marker={"color":'blue'},name='Legende Scatter Plot')
              ,row=1,col=1)

fig.add_trace(go.Scatter(x=data_alg.index, y=data_alg.Algeria,name='Legende Line
Plot')
               ,row=2,col=1)
```

```
9
10 fig.update_xaxes(title_text="Année Scatter 1", row=1, col=1)
11 fig.update_xaxes(title_text="Année Scatter 2", row=2, col=1)
12
13 fig.update_yaxes(title_text="Nombre D\'Immigrants", row=1, col=1)
14 fig.update_yaxes(title_text="Nombre D\'Immigrants", row=2, col=1)
15
16
17 fig.update_layout(height=600, width=600,title='Immigration de l\'Algérie vers
   Canadas' )
18 fig.show()
19
```



Prenant, le nombre d'immigrés par continent pour l'année 2013 :

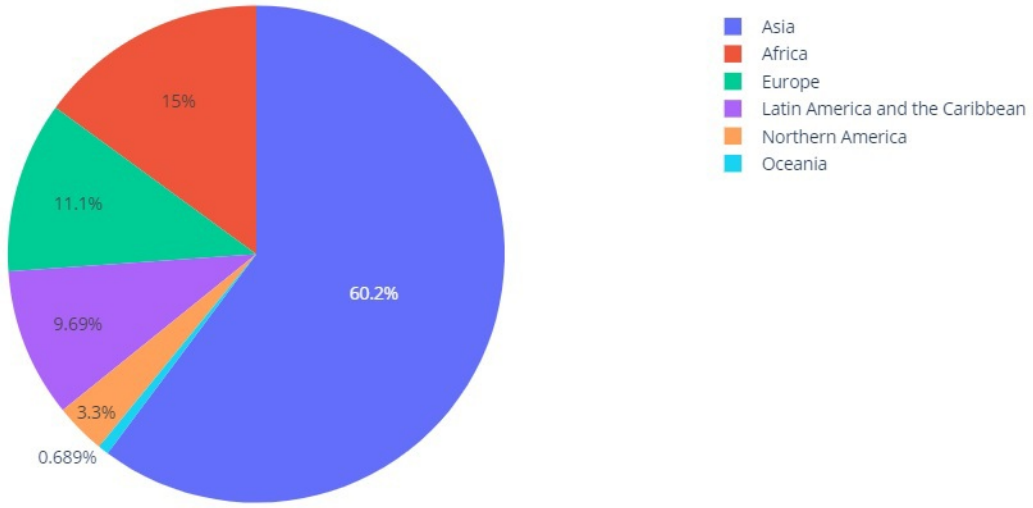
```
1 img_2013=df.groupby(["AreaName"])["2013"].sum()
2 img_2013
```

```
1 AreaName
2 Africa                                38543
3 Asia                                 155075
4 Europe                                28691
5 Latin America and the Caribbean       24950
6 Northern America                       8503
7 Oceania                                1775
8 Name: 2013, dtype: int64
```

Pour tracer un Pie plot, il suffit d'invoquer *graph_objects.Pie()* avec les deux paramètres :

- *labels* :contenant les étiquettes des proportion à tracer ;
- *values* :contenant les proportions pour chaque continent;

```
1 fig=go.Figure()
2 fig.add_trace(go.Pie(labels=img_2013.index,values=img_2013))
3 fig.show()
```



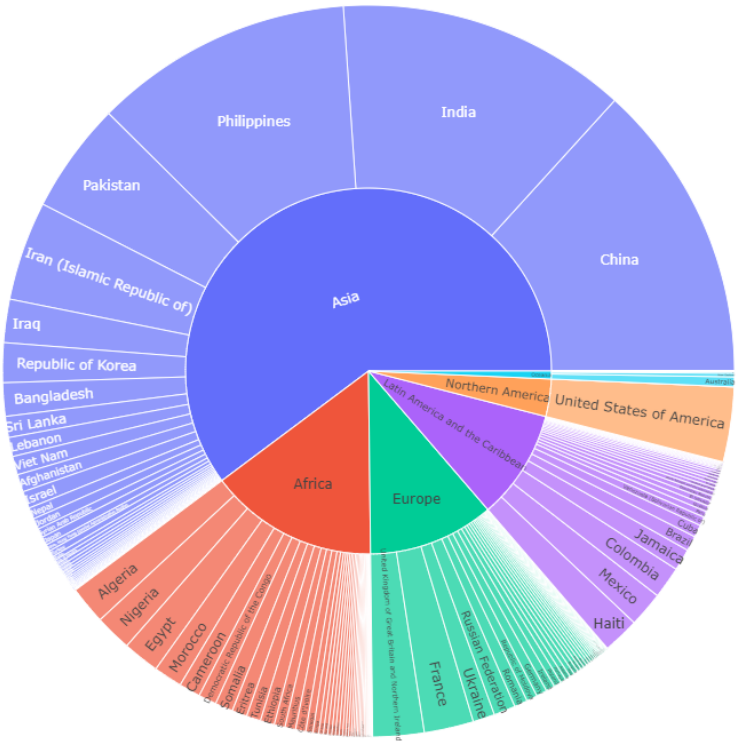
Plotly *express* (*px*) est un objet qui offre plusieurs des fonctionnalités de *graph_objects* mais avec une syntaxe plus simple. Le code suivant illustre comment créer un Pie plot avec Plotly express.

- Pour commencer, il est nécessaire d'importer *plotly.express* (px);
- *AreaName* est retirer de l'index afin d'utiliser le nom de la colonne lors du traçage (*Ligne 2*) ;
- *px.pie()* prend en paramètre le DataFrame avec la colonne des proportions spécifiée par le paramètre *values* et les libellés des proportions spécifiés par le paramètre *names* ;

```
1 import plotly.express as px
2 img_2013=img_2013.reset_index()
3 fig = px.pie(img_2013, values='2013', names='AreaName')
4 fig.show()
```

L'un des graphes les plus intéressants offerts par Plotly est le Sunburst plot. Cette représentation est un Pie plot avec les proportions réparties hiérarchiquement. L'argument optionnel *path* permet de définir comment les proportions sont structurées. Dans cet exemple, les proportions sont divisées par *AreaName* en premier temps et puis par *OdName* :

```
1 img_agg=df.groupby(["AreaName","OdName"])["2013"].sum().reset_index()
2 fig = px.sunburst(img_agg, path=['AreaName', 'OdName'], values='2013',width=800,
  height=800)
3 fig.show()
```


Asia
Africa
Europe
Northern America
Latin America and the Caribbean
Oceania
China
India
Philippines
Pakistan
Iran (Islamic Republic of)
Iraq
Republic of Korea
Bangladesh
Sri Lanka
Lebanon
Viet Nam
Afghanistan
Israel
Algeria
Nigeria
Egypt
Morocco
Cameroon
Somalia
Eritrea
Kenya
Ethiopia
France
Ukraine
Russian Federation
United States of America
Haiti
Mexico
Colombia
Jamaica
Cuba
Brazil

Dash

XI

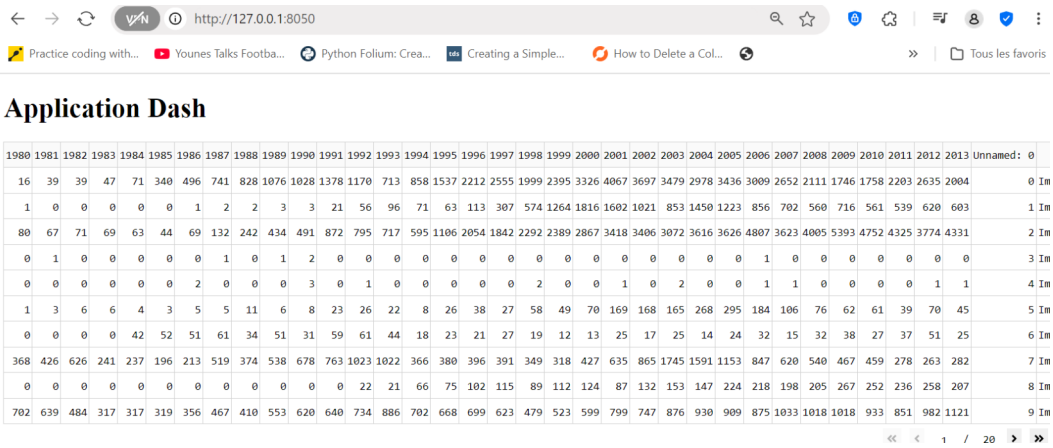
Dash[18*] est un framework qui permet de créer des applications web servant des données et visualisations comme Plotly. La commande suivante permet d'installer ce module :

```
conda install conda-forge::dash
```

L'exemple ci-dessous permet de créer une application qui affiche le DataFrame *canada.csv* dans une page HTML:

- *Dash()* permet de créer une application ;
- *app.layout* permet de définir la composition de la page à servir avec des éléments HTML (Div, Input, Radio, H1, etc) ;
- Le module *html* permet de créer des éléments HTML et de les composer comme dans une page web ;
- *dash_table.DataTable()* permet de créer un tableau HTML à partir d'un DataFrame converti sous forme de dictionnaire (*to_dict*). Le paramètre optionnel *page_size* indique le nombre maximal de lignes par page ;
- *app.run()* permet de lancer le serveur applicatif créé ;

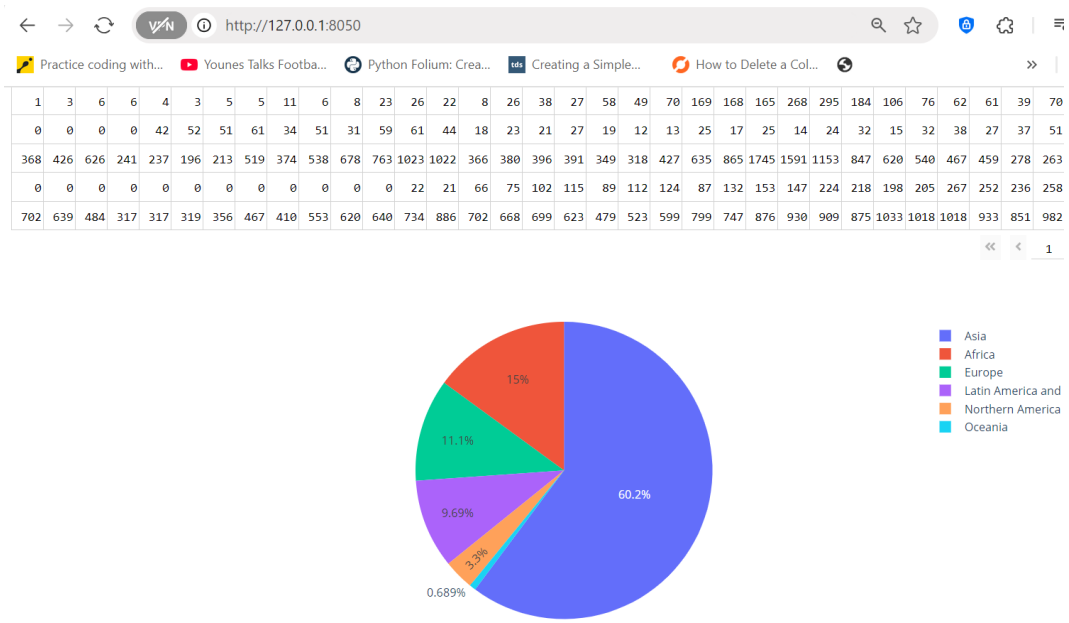
```
1  from dash import Dash, html, dash_table
2  import pandas as pd
3
4  df = pd.read_csv('canada.csv')
5
6  app = Dash()
7
8  app.layout = html.Div([
9      html.H1(children='Application Dash'),
10     dash_table.DataTable(data=df.to_dict("records"), page_size=10)
11     ])
12
13 app.run(debug=True)
14
```



Pour afficher un graphe Plotly dans l'application Dash, il est nécessaire d'importer *dcc* (Dash Core Components). Ce dernier propose des composants préfabriqués comme les Sliders et Dropdowns. Particulièrement, *dcc.Graph* permet de déclarer un graphe avec le paramètre *figure* qui est sensé être affecté à un graphe Plotly.

L'exemple suivant déclare un graphe Plotly *express pie* (*Ligne 8*) et inclut celui-ci dans le composant *dcc.Graph* (*Ligne 15*) avec le paramètre optionnel *figure*. A noter que le paramètre style (*Ligne 13*) permet de styler les composants HTML avec un dictionnaire. Les attributs utilisés pour styler les éléments de la page sont les mêmes attributs de style Javascript avec des nominations Pascal Case (*fontSize* au lieu de *font-size*, *fontFamily* au lieu de *font-family*,*backgroundColor* au lieu de *background-color*).

```
1 from dash import Dash, html, dcc, dash_table
2 import pandas as pd
3 import plotly.express as px
4
5 df = pd.read_csv('canada.csv')
6 img_2013=df.groupby(["AreaName"])["2013"].sum().reset_index()
7
8 fig = px.pie(img_2013, values='2013', names='AreaName')
9
10 app = Dash()
11
12 app.layout = html.Div([
13     html.H1(children='Application Dash',style={"fontSize": "48px", "color": "blue"
  },),
14     dash_table.DataTable(data=df.to_dict("records"), page_size=10),
15     dcc.Graph(figure=fig)
16     ])
17
18 app.run(debug=True)
19
```



L'objet *Dropdown* (*Ligne 24*) est un élément permettant de sélectionner une valeur comme l'élément select en HTML. Celui-ci est définie par une liste de valeurs/libellés sélectionnables à travers le paramètre *options* (*Ligne 26*). Le paramètre *value* (Ligne 27) défini la valeur sélectionnée par défaut.

Pour rendre l'application interactif, il est nécessaire de donner des identifiants aux éléments d'entrés (dans cet exemple *dcc.Dropdown* avec *id="year-filter"*) et sorties (*dcc.Graph* avec *id="plotly-fig"*).

La fonction nommée *callback* et importée au début du code est appelée un *décorateur*. Celle-ci prend une fonction en paramètre et retourne une nouvelle fonction qui étend le fonctionnement de la première. Cette fonction permet de déclarer les composants entrées (*Input Ligne 14*) et sorties (*Output Ligne 13*) dans la page HTML. Les entrées/sorties sont identifiées par leurs identifiants (*component_id*) et la valeur contrôlée (*component_property*).

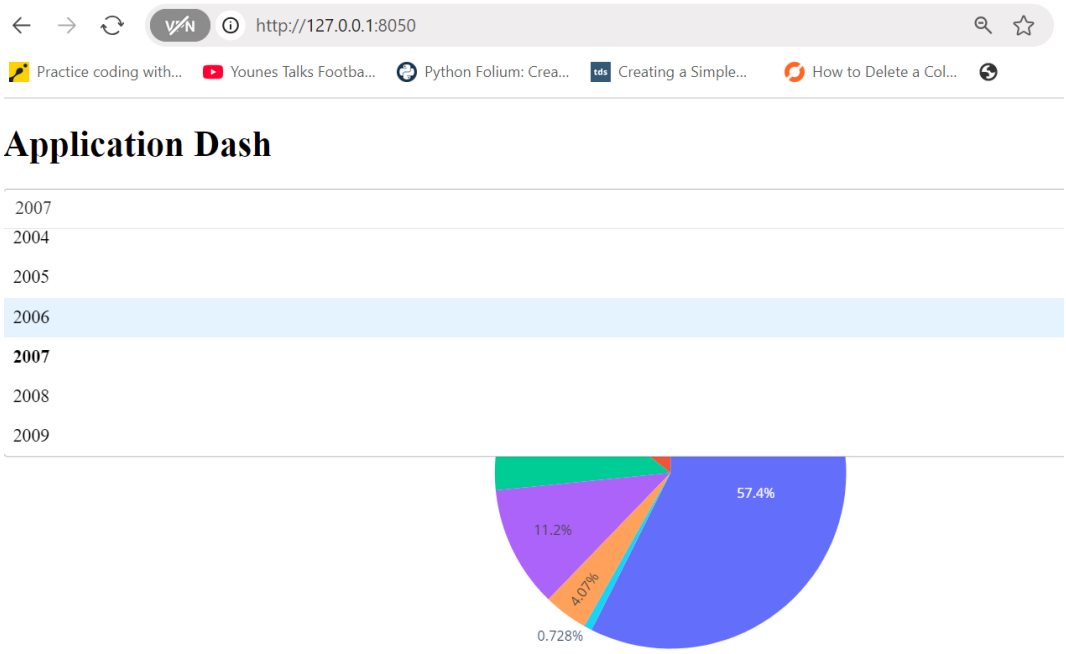
Dans cet exemple, l'entrée est le composant *dcc.Dropdown* désigné par l'identifiant *year-filter (Ligne 14)*. L'attribut pris comme entrée de ce composant est l'attribut *value*. En outre, la sortie est le composant *dcc.Graph* désigné par l'identifiant *plotly-fig (Ligne 13)*. L'attribut pris comme sortie de ce composant est l'attribut *figure*.

La syntaxe *@callback* signifie que chaque fois la fonction déclarée tout juste après (*update_graph(selected_year) Ligne 16*), celle-ci est décorée avec la fonction *callback*. La fonction *update_graph(selected_year)* est appelée de manière asynchrone chaque fois que la valeur du Input déclaré (*Dropdown id=year-filter*) change (comme . *addEventListener() Javascript*). Cette fonction sélectionne la colonne désignée par l'année en paramètre *selected_year* et recalcule le Dataframe utilisé pour créer le Pie plot. Lorsque le composant *Dropdown* change de valeur, *update_graph(selected_year)* est invoquée avec *selected_year* qui prend la nouvelle valeur du composant.

Pour terminer, la figure créée avec *px.pie()* est retournée par cette fonction. La valeur retournée par cette fonction sera placée dans l'attribut contrôle du Output (*figure*).

```
1 from dash import Dash, html, dcc,callback, Output, Input
2 import pandas as pd
3 import plotly.express as px
4
5 df = pd.read_csv('canada.csv')
6 img_2013=df.groupby(["AreaName"])["2013"].sum().reset_index()
7
8 years=[*map(str,range(2004,2014))]
9
10 fig = px.pie(img_2013, values='2013', names='AreaName')
11
12 @callback(
13     Output(component_id='plotly-fig', component_property='figure'),
14     Input(component_id='year-filter', component_property='value')
15 )
16 def update_graph(selected_year):
17     img_year=df.groupby(["AreaName"])[selected_year].sum().reset_index()
18     return px.pie(img_year, values=selected_year, names='AreaName')
19
20 app = Dash()
21
22 app.layout = html.Div([
23     html.H1(children='Application Dash'),
24     dcc.Dropdown(
25     id="year-filter",
26     options=[{"label": year, "value": year} for year in years],
27     value="2013"
28     ),
29     dcc.Graph(figure=fig, id="plotly-fig")
30     ])
31
32 app.run(debug=True)
```

33


http://127.0.0.1:8050
Practice coding with...
Younes Talks Footba...
Python Folium: Crea...
Creating a Simple...
How to Delete a Col...
Application Dash
2007
2004
2005
2006
2007
2008
2009
57.4%
11.2%
4.07%
0.728%

Références

[08] seaborn 2023, https://seaborn.pydata.org/

[09] matplotlib 2023, https://matplotlib.org/

[10] Los Angeles Museum Visitors 2023, https://www.kaggle.com/datasets/cityofLA/los-angeles-museum-visitors

[11] Video Games Sales 2023, https://www.kaggle.com/datasets/ulrikthygepedersen/video-games-sales

[12] IGN Video Games Scores
2023, https://drive.google.com/file/d/1TW-_fV8Wv_xLo5mW_XXgy-Vlg05zGAF7/view?usp=drive_link

[13] House Prices
2024, https://drive.google.com/file/d/1S6pvHwqn6MQS0JFY4rD8CwmKUtW9vRE8/view?usp=drive_link

[14] Wordcloud 2024, https://pypi.org/project/wordcloud/

[15] Pillow 2024, https://pillow.readthedocs.io/en/stable/

[16] Plotly 2024, https://plotly.com/

[17] Canada Immigration
2024, https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-DV0101EN-SkillsNetwork/Data%20Files/Canada.xlsx

[18] Dash 2024, https://dash.plotly.com/