SY09 Printemps 2022
TD/TP 02 — Visualisation de données

La visualisation est une part importante de l'analyse de données que ce soit lors de travaux préliminaires d'exploration pour mettre en évidence des informations cachées ou pour la présentation de résultats.

1 Travaux pratiques

Il existe de nombreux outils permettant de faire de la visualisation. Parmi les solutions existantes en Python, nous utiliserons la bibliothèque `seaborn` qui a l'avantage d'être assez exhaustive tout en proposant une interface consistante. Elle s'appuie sur la bibliothèque `matplotlib` que nous utiliserons également pour adapter certaines figures.

Un alias couramment utilisé est le suivant :

```
import seaborn as sns
```

Nous utiliserons les jeux de données `titanic` et `iris`, présents dans `seaborn`, qu'on peut charger avec les instructions suivantes :

```
titanic = sns.load_dataset("titanic")
iris = sns.load_dataset("iris")
```

1.1 Visualisation univariée

La visualisation univariée consiste à étudier un seul prédicteur à la fois sans tenir compte des liens avec les autres prédicteurs.

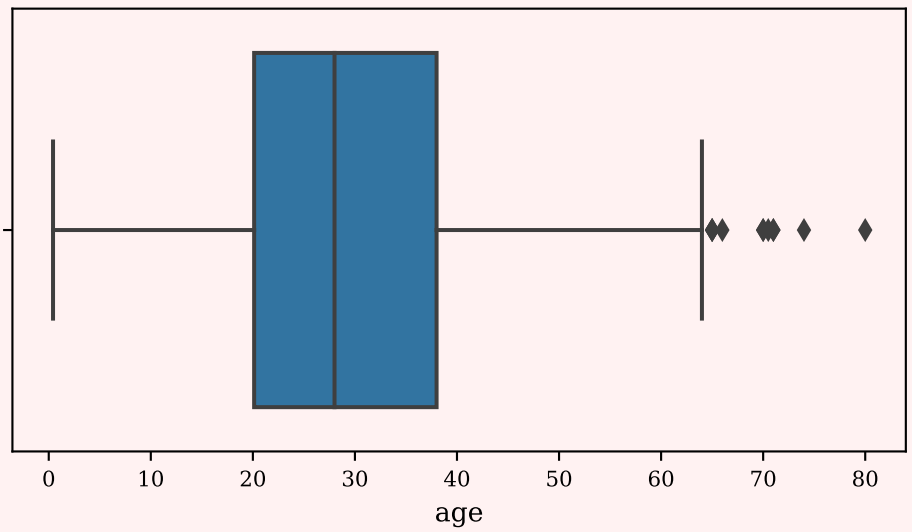
Prédicteur quantitatif

Pour les prédicteurs quantitatifs, on peut utiliser les boites à moustaches. Avec `seaborn`, on a la syntaxe suivante :

```
sns.boxplot(x=<Série Pandas>)
```

1 Tracer la boite à moustaches des âges des passagers du Titanic. Quel est l'âge médian ?

```
In [1]: sns.boxplot(x=titanic.age)
        plt.show()
```



L'âge médian est autour de 28 ans.

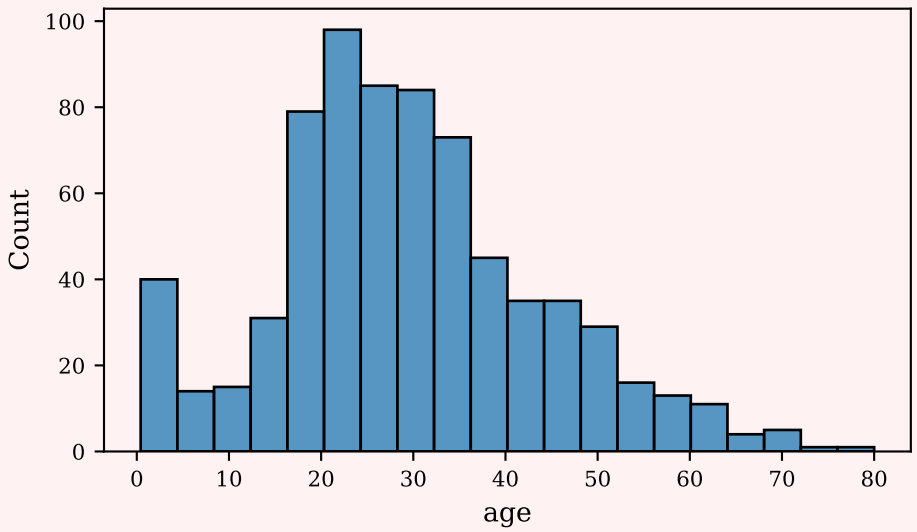
L'autre visualisation classique d'une variable quantitative est l'histogramme qu'on peut obtenir avec l'instruction suivante :

```
sns.histplot(<Série Pandas>)
```

2 Tracer l'histogramme des âges des passagers. Que font les arguments optionnels suivants ?

1. `bins`
2. `kde`
3. `rug`

```
In [2]: sns.histplot(titanic.age)
        plt.show()
```



On peut modifier l'apparence de l'histogramme avec les arguments optionnels suivants :

1. `bins` : permet de régler le nombre de boîtes.
2. `kde` : ajoute une courbe de densité

3. `rug` : ajoute une distribution unidimensionnelle des données

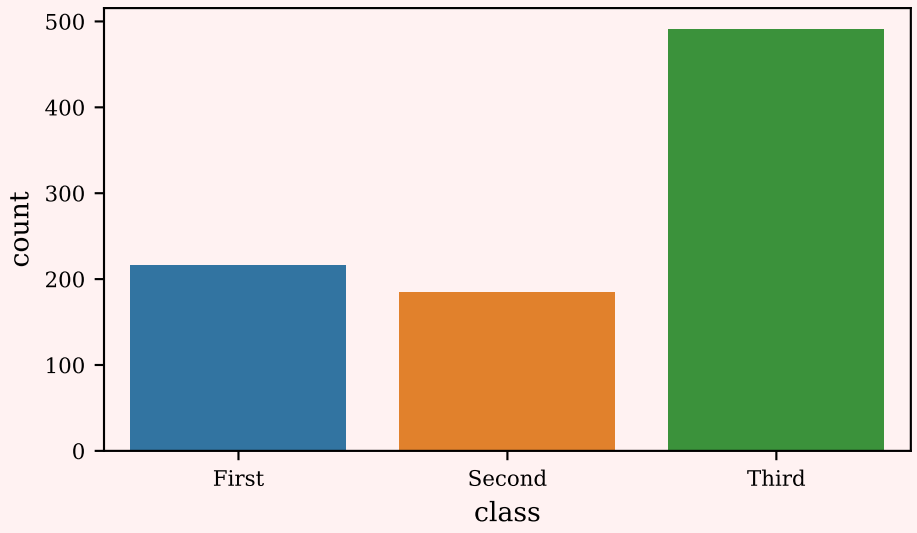
Prédicteur qualitatif

Pour les prédicteurs qualitatifs, on utilise un diagramme en barres avec la syntaxe suivante :

```
sns.countplot(x=<Colonne>, data=<DataFrame Pandas>)
```

3 Représenter la donnée des classes des passagers.

```
In [3]: sns.countplot(x="class", data=titanic)
        plt.show()
```



1.2 Visualisation multivariée

La visualisation multivariée consiste à étudier les relations pouvant exister entre plusieurs prédicteurs. Les techniques utilisées varient principalement en fonction de la nature des prédicteurs : qualitatifs ou quantitatifs.

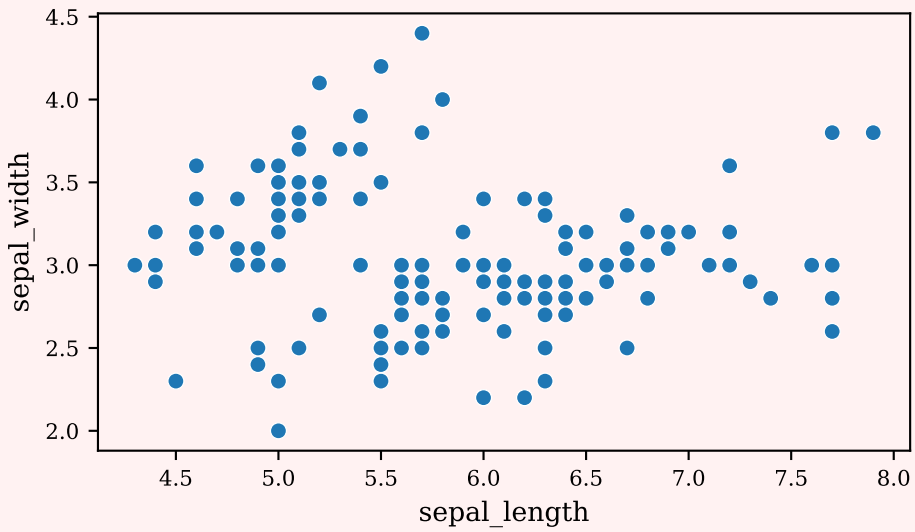
Quantitatif *vs* quantitatif

Diagramme de dispersion Le diagramme de dispersion permet de visualiser les relations existantes entre deux variables qualitatives. On trace un diagramme de dispersion avec l'instruction suivante :

```
sns.scatterplot(
  x=<Colonne des abscisses>,
  y=<Colonne des ordonnées>,
  data=<DataFrame Pandas>
)
```

4 Tracer le diagramme de dispersion de la largeur du sépale en fonction de sa longueur.

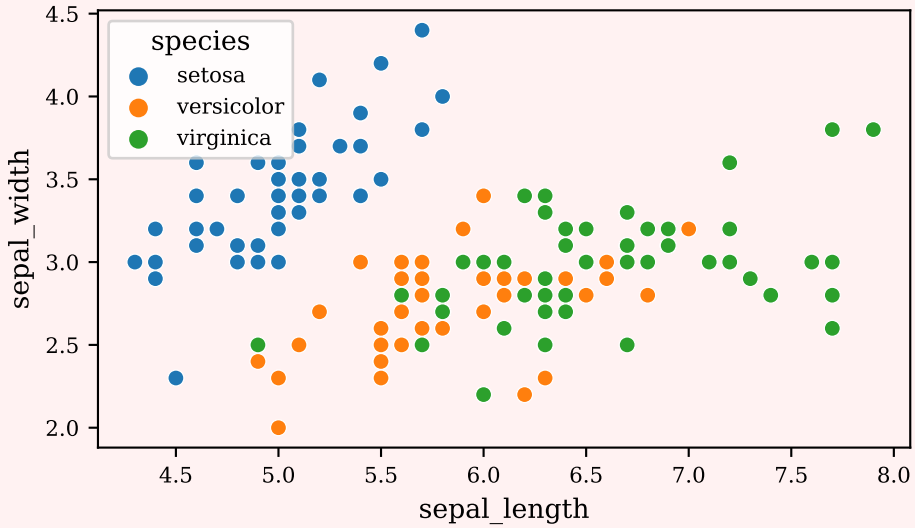
```
In [4]: sns.scatterplot(x="sepal_length", y="sepal_width", data=iris)
        plt.show()
```



Il est possible de visualiser d'autres variables en jouant sur la couleur, la forme ou la taille des points avec les arguments `hue`, `style` et `size`.

5 Rajouter la donnée de classe avec la couleur.

```
In [5]: sns.scatterplot(x="sepal_length", y="sepal_width", hue="species",
        ↪ data=iris)
        plt.show()
```



6 En plus de la donnée de classe, ajouter la longueur du pétale avec la taille du point.

```
In [6]: sns.scatterplot(x="sepal_length", y="sepal_width", hue="species",
        ↪ size="petal_length", data=iris)
        plt.show()
```

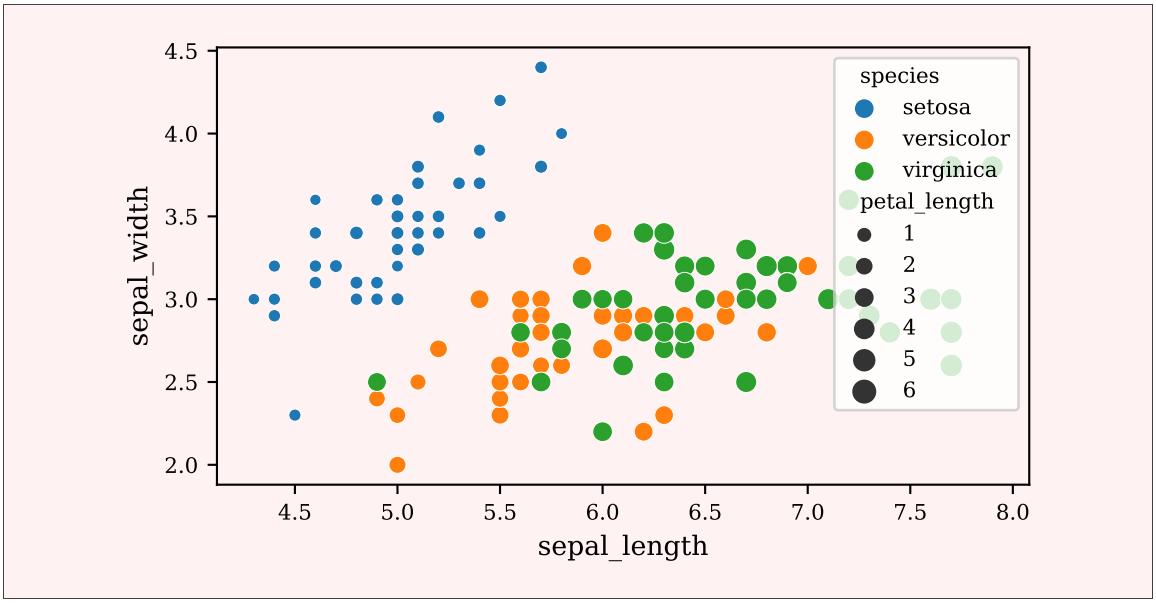
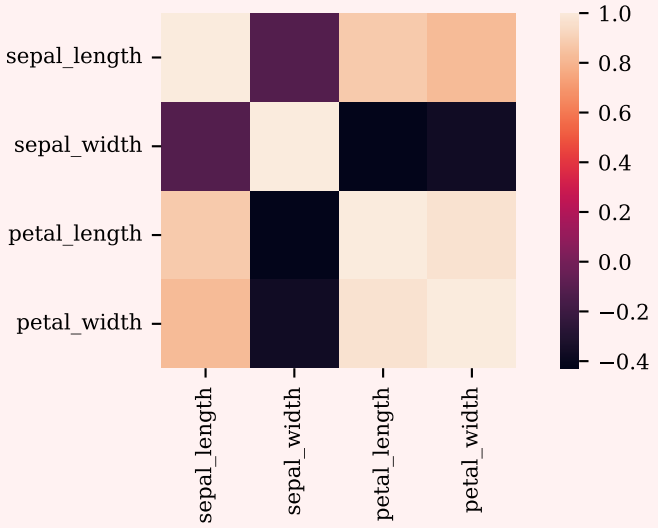


Diagramme de corrélation Le diagramme de corrélation est utile pour avoir une idée des liens de type linéaire entre des variables quantitatives. Le diagramme de corrélation n'est pas directement supporté par `seaborn` : il faut d'abord créer ce tableau à l'aide de la fonction `Pandas corr` et utiliser ensuite la visualisation `heatmap`.

```
corr = <DataFrame Pandas>.corr()
sns.heatmap(corr) # nooutput
```

7 Étudier les corrélations linéaires des variables quantitatives du jeu de données iris. Peut-on dire que les largeurs du sépale et du pétale sont corrélés négativement ? Étudier l'influence de l'espèce.

```
In [7]: corr = iris.corr()
        sns.heatmap(corr, square=True)
        plt.show()
```



La corrélation globale est négative. Conditionnellement à la classe, la corrélation est positive (paradoxe de Simpson).

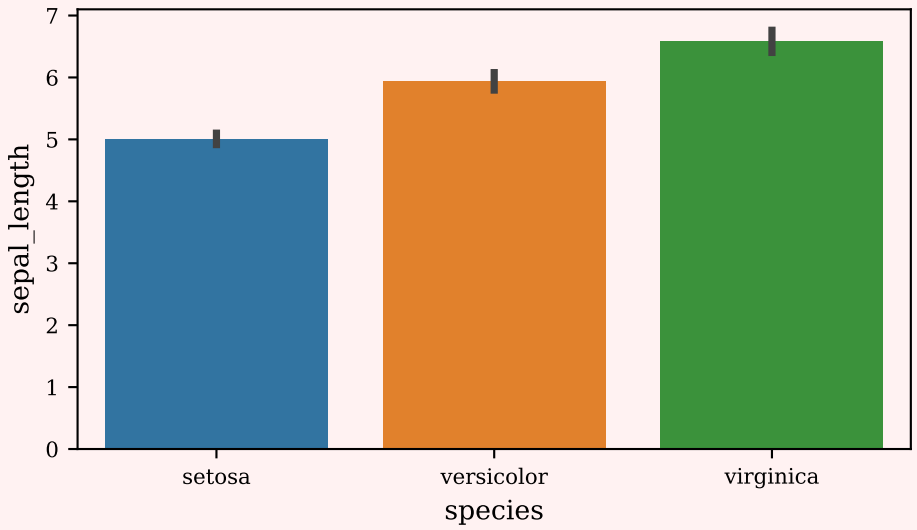
Qualitatif *vs* quantitatif

Diagramme en barres avec dispersion La fonction `barplot` permet de visualiser une variable quantitative avec une indication sur la dispersion des données sous forme d'un intervalle de confiance en fonction d'une colonne qualitative. La syntaxe est la suivante :

```
sns.barplot(
  x=<Colonne qualitative>,
  y=<Colonne quantitative>,
  data=<DataFrame Pandas>
)
```

8 Avec le jeu de données `iris`, visualiser la longueur des sépales en fonction de l'espèce.

```
In [8]: sns.barplot(x="species", y="sepal_length", data=iris)
        plt.show()
```

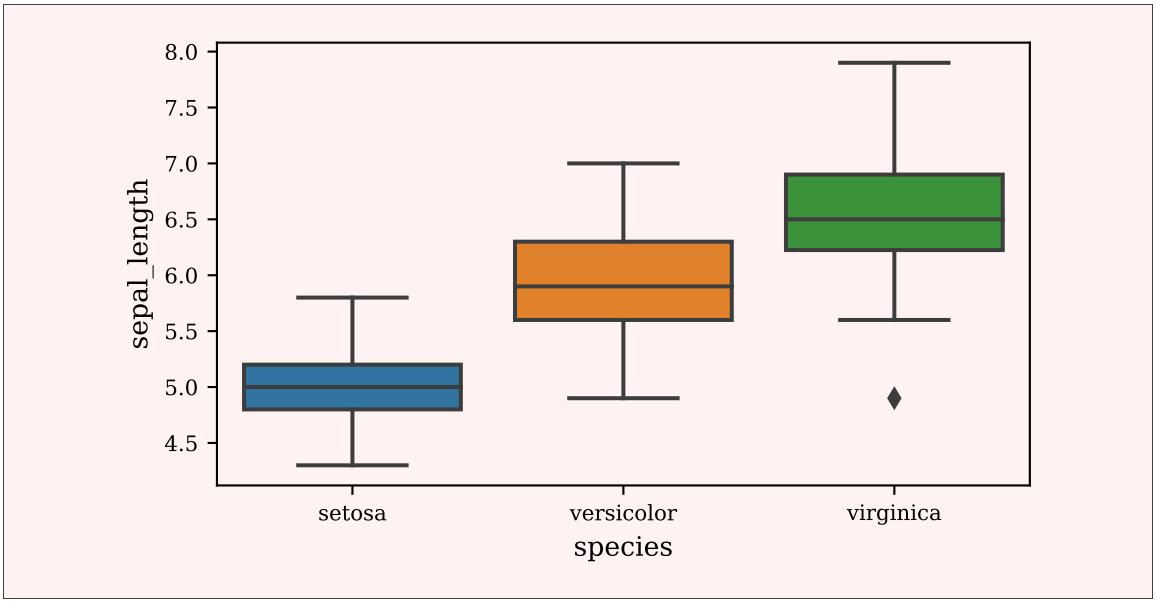


Boites à moustaches multiples Au lieu d'afficher un diagramme en bâtons, on peut afficher des boites à moustaches. La syntaxe est similaire.

```
sns.boxplot(
  x=<Colonne qualitative>,
  y=<Colonne quantitative>,
  data=<DataFrame Pandas>
)
```

9 Avec le jeu de données `iris`, visualiser les boites à moustaches des longueur des sépales en fonction de l'espèce.

```
In [9]: sns.boxplot(x="species", y="sepal_length", data=iris)
        plt.show()
```



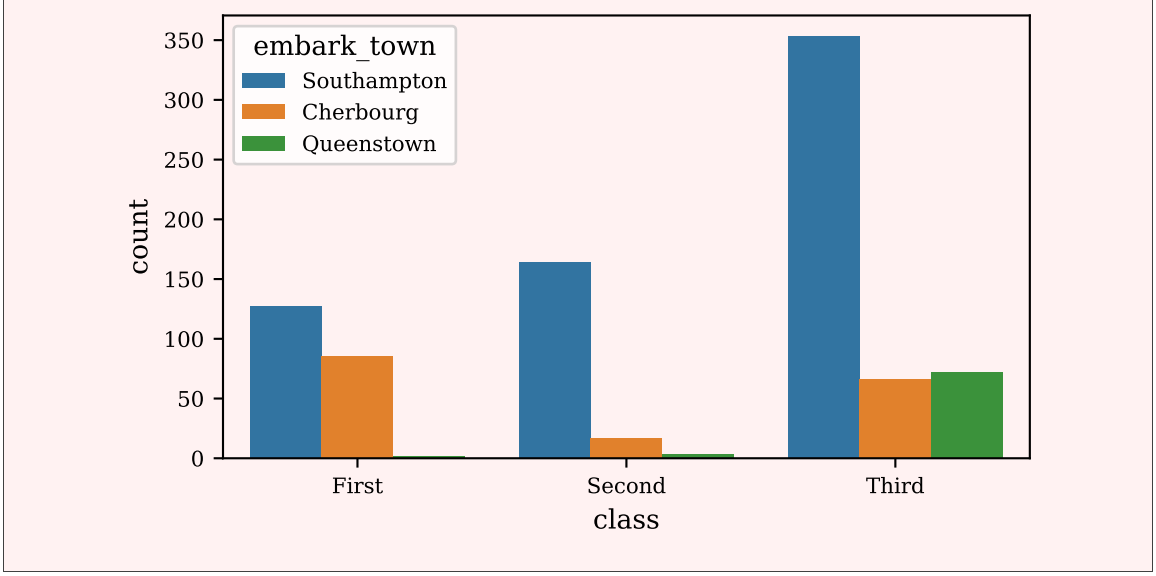
Qualitatif *vs* qualitatif

Lorsque les deux variables sont qualitatives, les représentations possibles sont centrées autour du tableau de contingence. On peut utiliser la fonction `countplot` en fournissant la deuxième variable qualitative en argument.

```
sns.countplot(
  x=<Colonne qualitative>,
  hue=<Colonne qualitative>,
  data=<DataFrame Pandas>
)
```

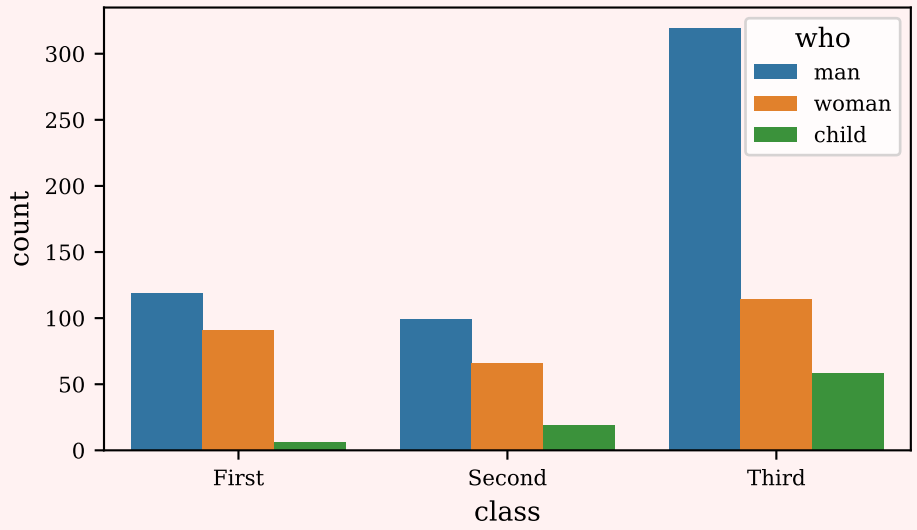
10 Tracer les lieux d'embarquation en fonction de la classe.

```
In [10]: sns.countplot(x="class", hue="embark_town", data=titanic)
         plt.show()
```



11 Visualiser le descripteur « who » en fonction de la classe.

```
In [11]: sns.countplot(x="class", hue="who", data=titanic)
         plt.show()
```



Représentation multiple

Dans certains cas, il est utile de pouvoir appliquer une même visualisation en parallèle sur des sous-jeux de données identifiés par les modalités d'un ou plusieurs descripteurs. Chacune des représentations est appelée une facette.

Pour réaliser cela avec `seaborn`, suivant le type de la facette, il faut utiliser les fonctions suivantes :
- `catplot` pour des facettes de type *qualitatif* vs *quantitatif* ou *qualitatif*
- `relplot` pour des facettes de type *quantitatif* vs *quantitatif*
- `displot` pour des facettes de type histogramme

Pour chacune des fonctions, on spécifie comment partitionner le jeu de données avec les arguments `row` et `col` qui sont des noms de colonnes dans le jeu de données.

```
sns.relplot(
  x=<Colonne des abscisses>,
  y=<Colonne des ordonnées>,
  kind=<Type de facette>,
  row=<Colonne des lignes>,
  col=<Colonne des colonnes>,
  data=<DataFrame Pandas>
)
```

12 Visualiser « age » en fonction de « class » avec trois histogrammes.

```
In [12]: sns.displot(x="age", col="class", kind="hist", data=titanic)
```

13 Visualiser « age » en fonction de « class » et « who » avec des histogrammes.

```
In [13]: sns.displot(x="age", col="class", row="who", kind="hist", data=titanic)
```

1.3 Visualisation du jeu de données `sy02-p2019.csv`

14 Charger le jeu de données contenu dans le fichier `data/sy02-p2019.csv`. Quel est le type de la colonne « Note médian » ? Pourquoi ? On pourra utiliser l'argument `na_values`.

```
In [14]: X = pd.read_csv("data/sy02-p2019.csv")
         X["Note médian"].dtype
Out [14]: dtype('O')
```

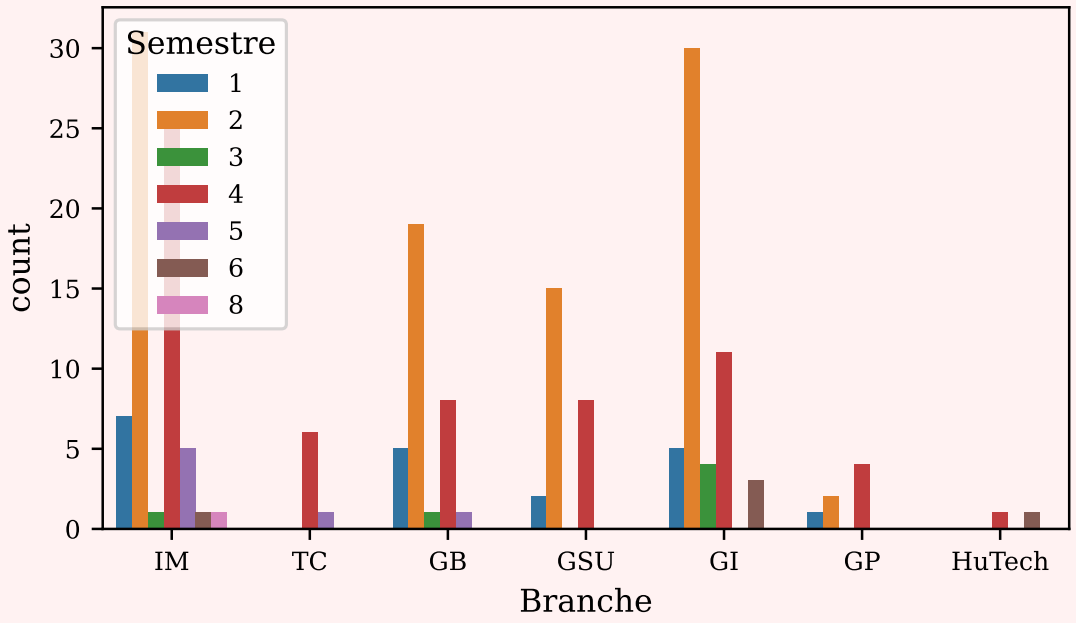
Les données ne sont pas correctement chargées. La colonne correspondant à la note de médian n'est pas reconnue comme quantitative à cause du « ABS ».

```
In [15]: X = pd.read_csv("data/sy02-p2019.csv", na_values="ABS")
         X["Note médian"].dtype
Out [15]: dtype('float64')
```

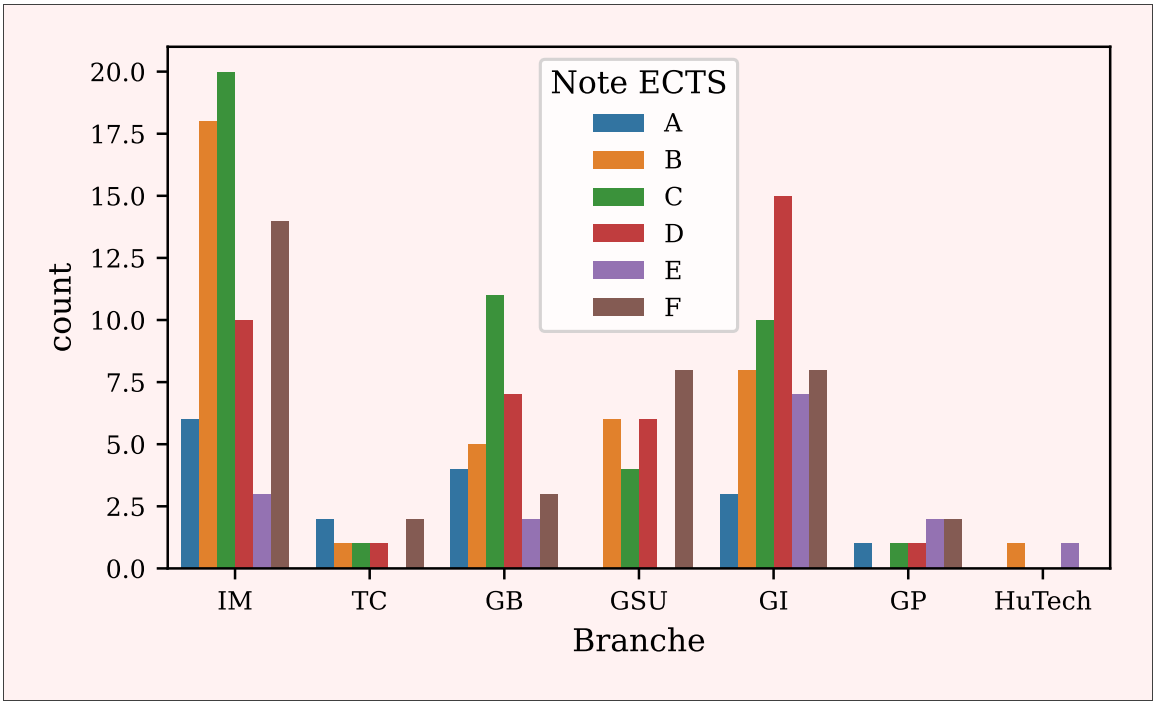
15 Visualiser les effectifs des différentes branches. Rajouter

1. l'information de « niveau »,
2. la note ECTS. Les modalités sont-elles dans l'ordre ?

```
In [16]: sns.countplot(x="Branche", hue="Semestre", data=X)
         plt.show()
```



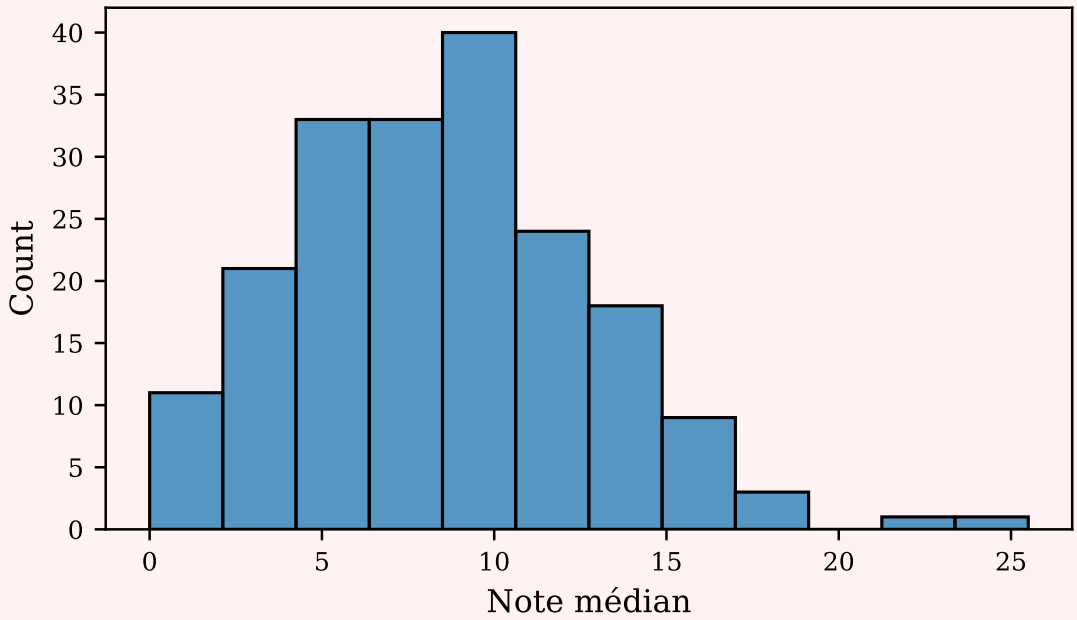

```
In [17]: X["Note ECTS"] = pd.Categorical(X["Note ECTS"], ordered=True)
         sns.countplot(x="Branche", hue="Note ECTS", data=X)
         plt.show()
```



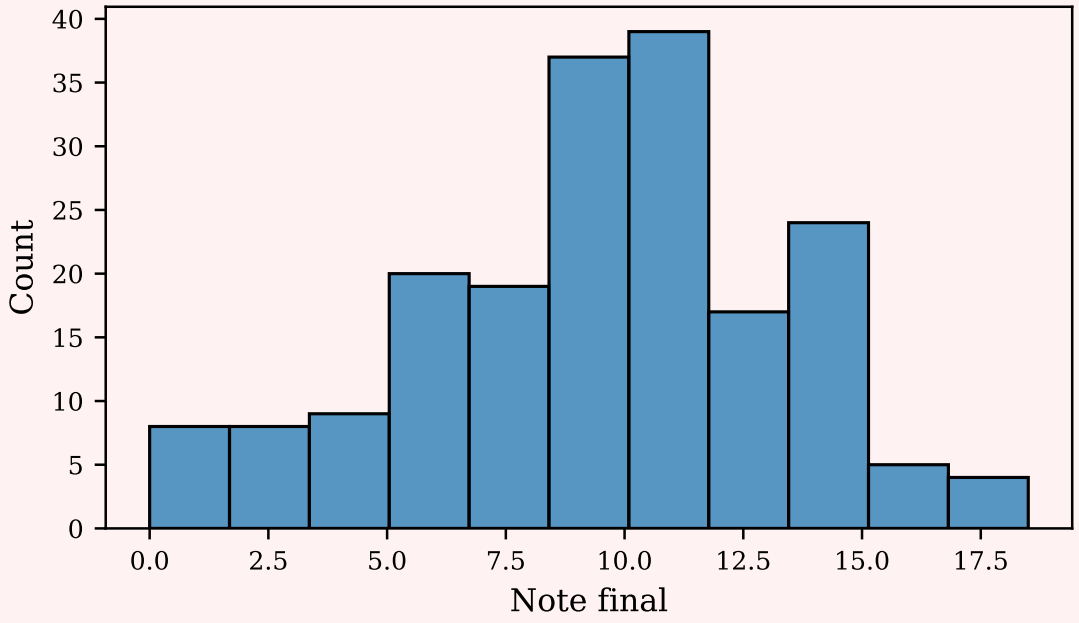
16 Visualiser les distributions des notes de médian et des notes de final. Afficher le diagramme de dispersion.

Synthétiser les trois visualisations précédentes en utilisant `sns.jointplot`.

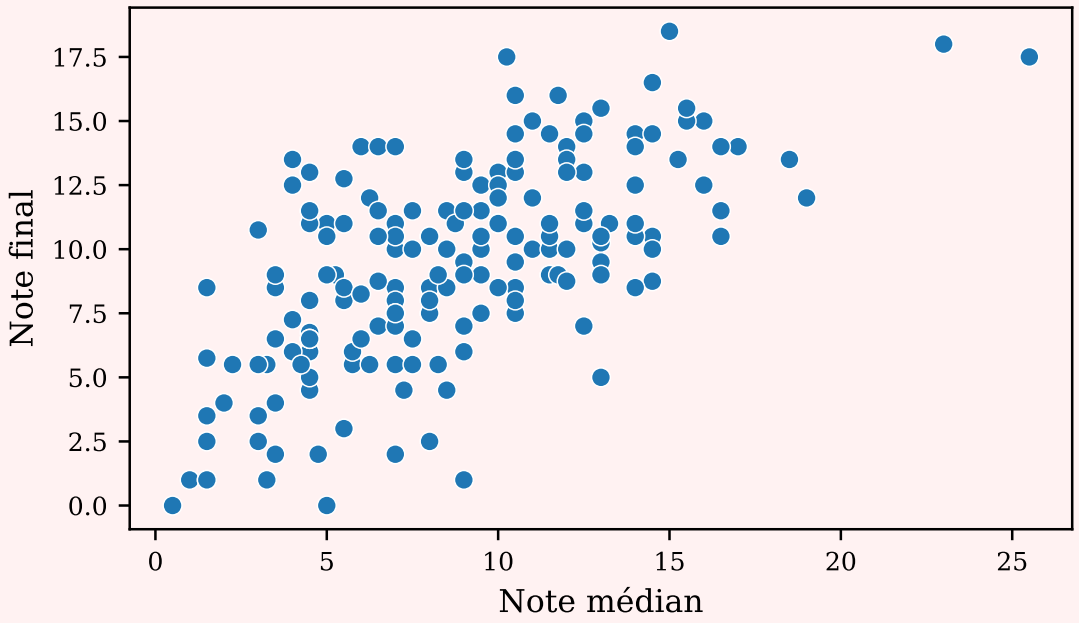
```
In [18]: sns.histplot(X["Note médian"])
         plt.show()
```



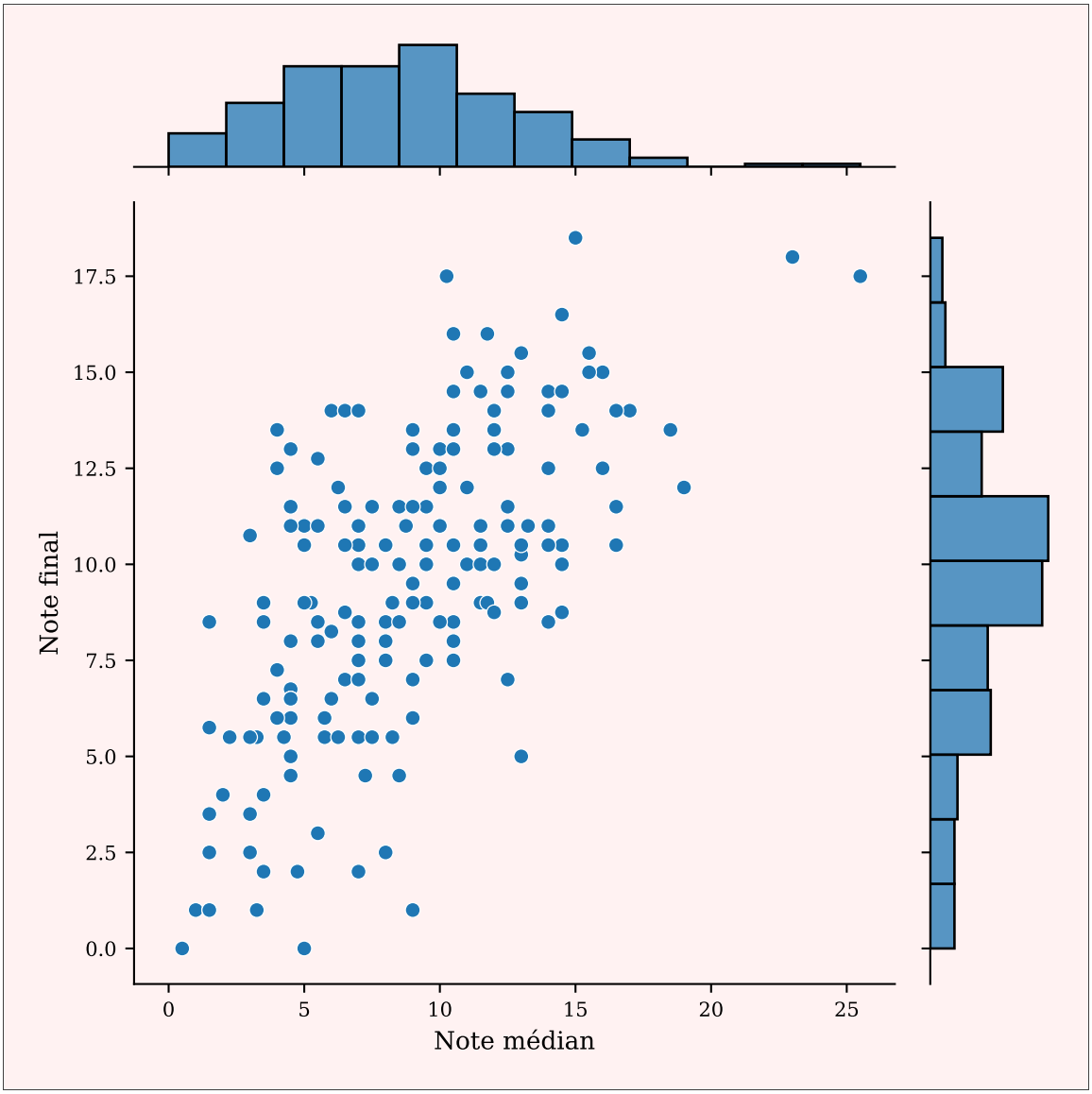

```
In [19]: sns.histplot(X["Note final"])
         plt.show()
```



In [20]:
```
sns.scatterplot(x="Note médian", y="Note final", data=X)
plt.show()
```

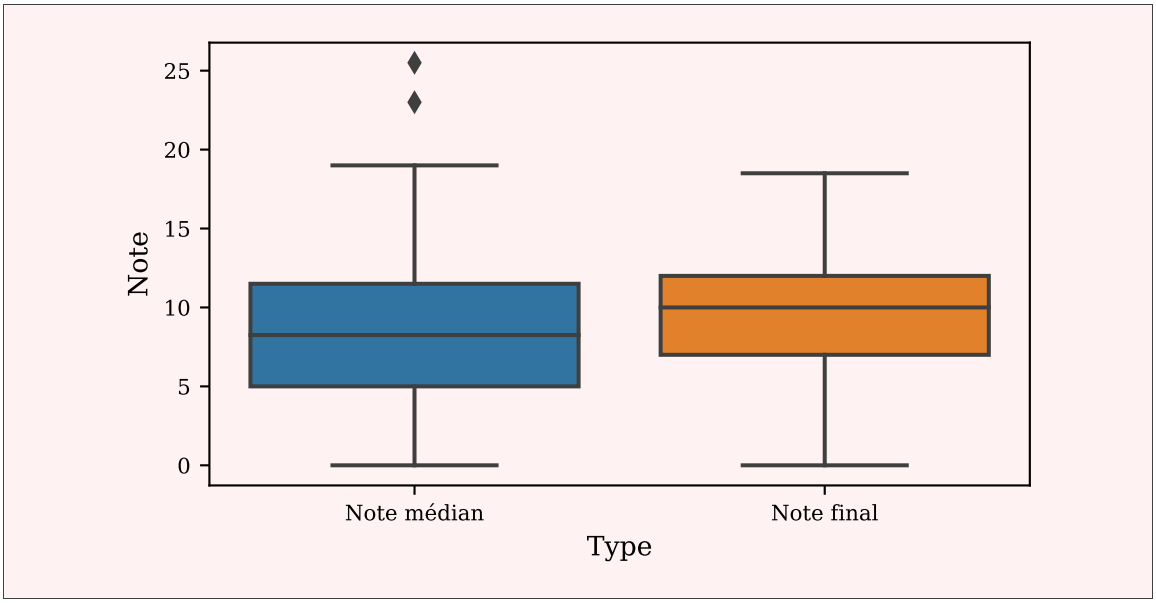


In [21]:
```
sns.jointplot(x="Note médian", y="Note final", data=X)
plt.show()
```



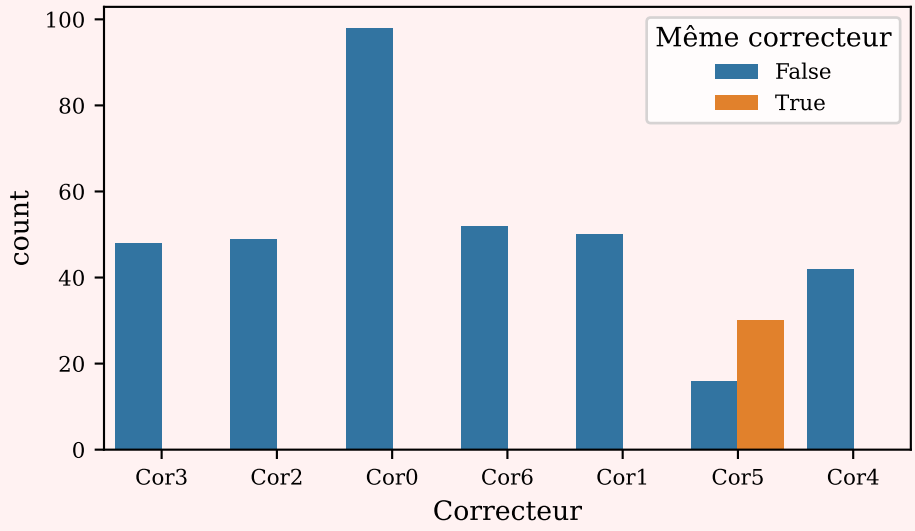
17 Tracer en parallèle les boites à moustaches des notes de médian et de final. On pourra utiliser la fonction `melt`.

```
In [22]: X1 = X.melt(
             value_vars=["Note médian", "Note final"], var_name="Type",
             ↪ value_name="Note"
         )
         sns.boxplot(x="Type", y="Note", data=X1)
         plt.show()
```

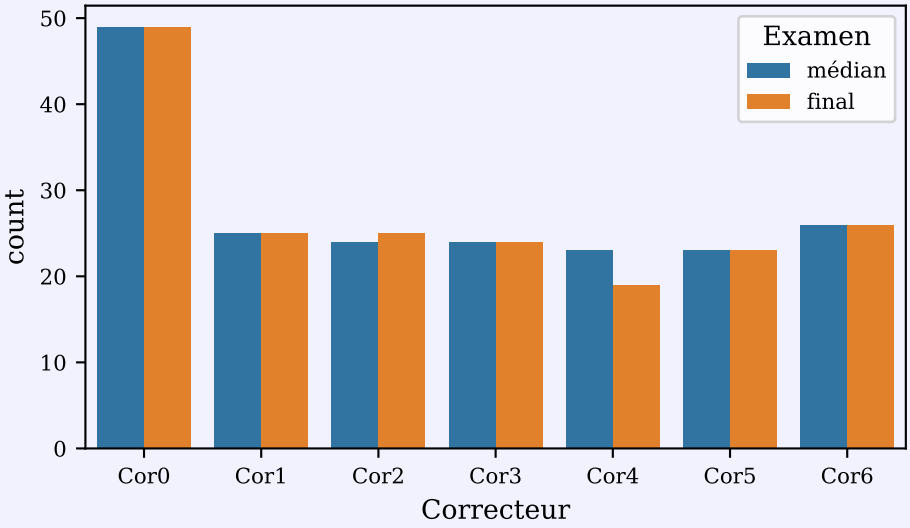


18 Trouver une visualisation du nombre de copies corrigées sur tout le semestre (médian et final confondu) par correcteur en indiquant pour chaque correcteur la part des copies correspondant à un même étudiant (médian et final corrigés par ce même correcteur) et la part des copies qui ne sont pas corrigées par le même correcteur.

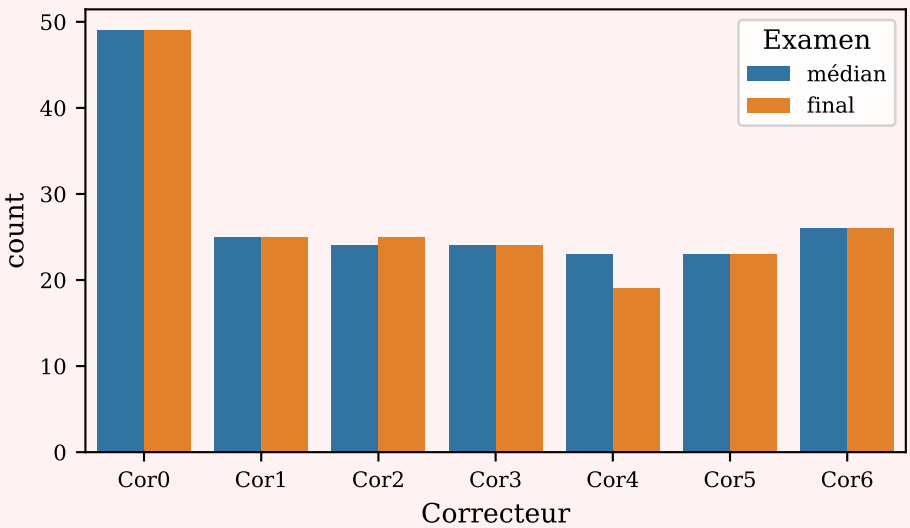
```
In [23]: X["Même correcteur"] = X["Correcteur médian"] == X["Correcteur final"]
         X1 = X.melt(
             id_vars=["Même correcteur"],
             value_vars=["Correcteur final", "Correcteur médian"],
             var_name="Examen",
             value_name="Correcteur",
         )

         sns.countplot(x="Correcteur", hue="Même correcteur", data=X1)
         plt.show()
```



19 Comment obtenir la figure suivante ?



In [24]:
```
X1 = X.rename(columns={"Correcteur médian": "médian", "Correcteur final": "final"})

X1 = X1.melt(
    id_vars=["Note médian", "Note final"],
    value_vars=["médian", "final"],
    var_name="Examen",
    value_name="Correcteur",
)

X1["Note"] = np.where(X1["Examen"] == "médian", X1["Note médian"], X1["Note final"])

X1 = X1.drop(columns=["Note médian", "Note final"])

sns.countplot(x="Correcteur", hue="Examen", data=X1)
plt.show()
```



2 Exercices

2.1 Histogramme et estimation de densité

On considère une population $\mathcal{P}$ d'individus, sur lesquels on observe un caractère $X \in \mathbb{R}$. On souhaite montrer formellement qu'un histogramme est un estimateur de la densité de X.

20 Montrer que [1]

$$f(x) = \lim_{h \to 0} \frac{1}{h} \mathbb{P}\left(x \leq X \leq x + h\right). \tag{1}$$

On rappellera tout d'abord que

$$\mathbb{P}\left(x \leq X \leq x + h\right) = F_X(x + h) - F_X(x),$$

où F_X est la fonction de répartition de la variable X. On rappellera ensuite que la densité est la dérivée de cette fonction de répartition ; soit, par définition, en tout $x \in \mathbb{R}$:

$$f(x) = \lim_{h \to 0} \frac{F_X(x + h) - F_X(x)}{h} = \lim_{h \to 0} \frac{1}{h} \mathbb{P}\left(x \leq X \leq x + h\right).$$

21 On considère maintenant un histogramme construit à partir d'un échantillon de réalisations $x_1, x_2, \ldots, x_n$ de X. Justifier l'emploi de l'histogramme comme estimation de la densité de X telle que définie par l'équation (1).

Avec la définition ci-dessus, on rappelle qu'un histogramme estime la densité de X au point x par [a]

$$\widehat{f}(x) = \frac{1}{h} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{V(x)}(x_i);$$

dans cette expression, le voisinage $V(x)$ du point x est défini comme l'intervalle $[y; y + h[$ dans lequel le point x se trouve (la densité estimée est donc la même pour tous les points $x \in [y; y + h[$). Cette expression coïncide donc avec l'équation (1) lorsque

$$\underbrace{\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{V(x)}(x_i)}_{\widehat{p}(V(x))} = \mathbb{P}\left(X \in V(x)\right); \tag{2}$$

cette égalité est évidemment généralement fausse. Remarquons toutefois que le terme de gauche de l'équation (2) estime celui de droite : en effet, introduisons une variable $Z(V(x))$ définie par

$$Z(V(x)) = \begin{cases} 1 & \text{si } X \in V(x), \\ 0 & \text{sinon;} \end{cases}$$

par définition, $Z(V(x)) \sim \mathcal{B}(p(V(x)))$, où $p(V(x)) = \mathbb{P}(X \in V(x))$ est classiquement estimée à partir d'un échantillon par $\widehat{p}(V(x))$ (proportion de points situés dans l'intervalle $V(x)$).

Cette estimation, obtenue par la méthode des moments et la méthode du maximum de vraisemblance, est donc la réalisation d'un estimateur convergent. En conclusion, l'estimation $\widehat{p}(V(x))$ est d'autant plus proche de $\mathbb{P}\left(X \in V(x)\right)$, pour tout $x \in \mathbb{R}$, que l'échantillon est de grande taille.

Lorsque l'échantillon est de taille limitée, il sera nécessaire que la taille h des intervalles à partir desquels l'histogramme est construit soit suffisamment grande pour que les estimations $\widehat{p}(V(x))$ soient raisonnablement proches de $\mathbb{P}\left(X \in V(x)\right)$, pour tout $x \in \mathbb{R}$.

a. au prix d'une approximation sur la semi-ouverture de l'intervalle, sans grande conséquence sur le résultat final

1. On remarquera que la propriété (1) n'a pas tout-à-fait la même forme que dans le polycopié de cours.

2.2 Méthode des noyaux et apprentissage

On reprend l'estimateur de densité par la méthode des noyaux, en utilisant le noyau gaussien :

$$\widehat{f}(x;h) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right), \quad K(u) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}u^2\right). \tag{3}$$

On veut apprendre le paramètre $h^\star$ optimal par la méthode du maximum de vraisemblance. Notons que cette stratégie n'est pas la seule possible, et qu'elle n'est pas la plus utilisée en pratique.

On considère pour cela un échantillon de valeurs y_j $(j = 1, \dots, m)$, *qui peut être distinct de l'échantillon de valeurs* x_i $(i = 1, \dots, n)$[2] : l'échantillon des x_i est utilisé pour estimer la densité en tout point x via l'équation (3), et l'échantillon des y_j est utilisé pour choisir le paramètre h. En pratique, si on dispose d'un ensemble de données de taille suffisante, on peut sélectionner au hasard une partie des données pour constituer l'échantillon $x_1, \dots, x_n$, et le reste pour l'échantillon $y_1, \dots, y_m$.

22 Donner l'expression développée de la densité estimée en un point x quelconque. En déduire l'expression de la vraisemblance $L(h; y_1, \dots, y_m)$.

En combinant les deux relations données dans l'équation (3), on obtient trivialement

$$\widehat{f}(x;h) = \frac{1}{n}\sum_{i=1}^{n} (2\pi)^{-1/2} h^{-1} \exp\left(-\frac{1}{2}\frac{(x-x_i)^2}{h^2}\right).$$

Par définition, la vraisemblance $L(h; y_1, \dots, y_m)$ est la densité jointe de $y_1, \dots, y_m$ obtenue avec h :

$$L(h; y_1, \dots, y_m) = \widehat{f}(y_1, \dots, y_m; h) = \prod_{j=1}^{m} \widehat{f}(y_j; h),$$

en supposant l'indépendance des variables aléatoires deux à deux : $Y_j \perp\!\!\!\perp Y_{j'}$, pour tout $j \neq j'$.

23 Est-il raisonnable d'utiliser l'échantillon $x_1, \dots, x_n$ pour estimer le paramètre h par maximum de vraisemblance (c'est-à-dire de le substituer à l'échantillon $y_1, \dots, y_m$) ? Pourquoi ?

Sans précautions supplémentaires, cette stratégie donne une solution triviale (et non satisfaisante) :

$$\lim_{h\to 0} L(h; x_1, \dots, x_n) = +\infty.$$

Une telle valeur de h n'aboutit évidemment pas à une estimation convenable de la densité.

24 Calculer la log-vraisemblance, puis sa dérivée première.

On a

$$\ln L(h;\cdots) = \sum_{j=1}^{m} \ln \widehat{f}(y_j), \qquad \frac{\partial \ln L(h;\cdots)}{\partial h} = \sum_{j=1}^{m} \frac{\frac{\partial}{\partial h}\widehat{f}(y_j)}{\widehat{f}(y_j)}.$$

Calculons tout d'abord la dérivée de la densité :

$$\begin{aligned}\frac{\partial \widehat{f}(y_j)}{\partial h} &= \frac{1}{n}\sum_{i=1}^{n} (2\pi)^{-1/2}\left(-h^{-2}\exp\left(-\frac{(y_j-x_i)^2}{2h^2}\right) + h^{-4}(y_j-x_i)^2\exp\left(-\frac{(y_j-x_i)^2}{2h^2}\right)\right),\\ &= \frac{1}{n}\sum_{i=1}^{n} (2\pi)^{-1/2} h^{-1}\exp\left(-\frac{(y_j-x_i)^2}{2h^2}\right)\left(h^{-3}(y_j-x_i)^2 - h^{-1}\right);\end{aligned}$$

2. Les valeurs x_i et y_j vivent dans le même espace, mais on utilise volontairement une notation spécifique pour les différencier les unes des autres, étant donné les rôles bien distincts des deux échantillons.

on obtient donc

$$\frac{\partial \widehat{f}(y_j)/\partial h}{\widehat{f}(y_j)} = \frac{\sum_{i=1}^n \exp\left(-(y_j - x_i)^2/(2h^2)\right)\left(h^{-3}(y_j - x_i)^2 - h^{-1}\right)}{\sum_{i=1}^n \exp\left(-(y_j - x_i)^2/(2h^2)\right)}.$$

25 Montrer qu'annuler la dérivée première revient à exprimer le paramètre h comme

$$h = \left(\frac{1}{m}\sum_{j=1}^m \frac{\sum_{i=1}^n w_{ji}(h)(y_j - x_i)^2}{\sum_{i=1}^n w_{ji}(h)}\right)^{1/2}, \tag{4}$$

en précisant l'expression des poids $w_{ji}(h)$ qui dépendent de h.

L'annulation de la dérivée première donne

$$\frac{\partial \ln L(h;\cdots)}{\partial h} = 0 \quad \Leftrightarrow \quad \sum_{j=1}^m \frac{\sum_{i=1}^n \exp\left(-(y_j - x_i)^2/(2h^2)\right)h^{-3}(y_j - x_i)^2}{\sum_{i=1}^n \exp\left(-(y_j - x_i)^2/(2h^2)\right)} = mh^{-1},$$

$$\Leftrightarrow \quad h^2 = \frac{1}{m}\sum_{j=1}^m \frac{\sum_{i=1}^n \exp\left(-(y_j - x_i)^2/(2h^2)\right)(y_j - x_i)^2}{\sum_{i=1}^n \exp\left(-(y_j - x_i)^2/(2h^2)\right)};$$

en utilisant le fait que h est un paramètre de largeur de bande (donc positif), on retrouve bien l'expression désirée, avec $w_{ji}(h) = \exp\left(-(y_j - x_i)^2/(2h^2)\right)$.

26 L'annulation de la log-vraisemblance ne permet pas de trouver une forme analytique d'un EMV. On propose donc de calculer une estimation $h^\star$ qui maximise $\ln L(h;\cdots)$ de manière numérique.

On remarquera que l'équation (4) est du type $h = g(h)$: tout point fixe $h^\star$ de g est donc un point critique de $\ln L(h;\cdots)$. On propose donc de choisir une valeur $h_{(0)}$, puis de calculer itérativement

$$h_{(q+1)} = \left(\frac{1}{m}\sum_{j=1}^m \frac{\sum_{i=1}^n w_{ji}(h_{(q)})(y_j - x_i)^2}{\sum_{i=1}^n w_{ji}(h_{(q)})}\right)^{1/2},$$

jusqu'à ce que $|h_{(q+1)} - h_{(q)}| < \varepsilon$ (avec ε une valeur choisie). On admettra que les conditions d'application du théorème du point fixe sont satisfaites, et que la procédure ci-dessus mène bien à un maximum de la log-vraisemblance.

On pourra utiliser la fonction `pairwise_distances` de la librairie `scikit-learn` pour calculer les distances euclidiennes (au carré) entre les points de `X` et `Y` :

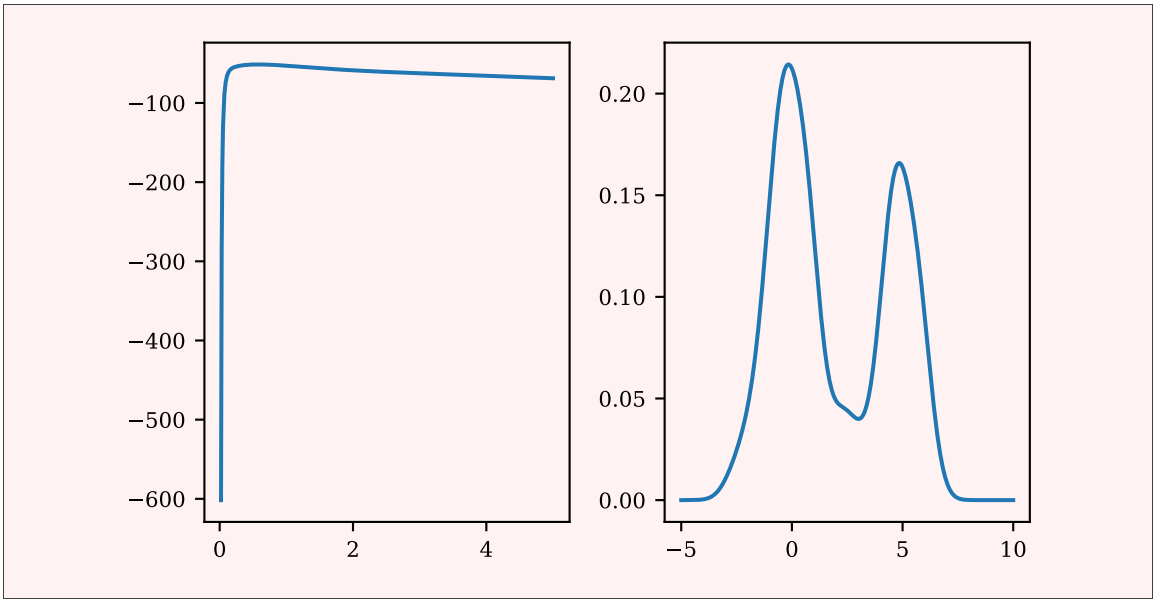
```
from sklearn.metrics import pairwise_distances as pedist
```

On pourra s'appuyer sur les fonctions suivantes, qui calculent les distances euclidiennes entre exemples, les poids $w_{ji}(h)$, la densité pour l'échantillon $Y = (y_1, \ldots, y_m)$, et la log-vraisemblance :

```
In [25]: def wgcomp(distXY, h):
             wgts = np.exp(-distXY**2/(2*h**2))
             return wgts

         def kdens(X, Y, h):
             distXY = pedist(X, Y)
             weights = wgcomp(distXY, h)
             dens = np.mean(weights/np.sqrt(2*np.pi)/h,axis=0)
             return dens

         def loglike(h, X, Y):
             dens = kdens(X, Y, h)
             logl = np.sum(np.log(dens))
             return logl
```

On peut alors calculer le point fixe avec le code suivant :

```
In [26]: # imports
         import numpy as np
         import scipy.stats as spst
         #from functions import pedist, wgcomp, kdens, loglike, pedistloo,
         ↪  kdensloo, loglikeloo
         import matplotlib.pyplot as plt

         X = np.append(spst.norm.rvs(loc=0,scale=1,
         ↪  size=30),spst.norm.rvs(loc=5,scale=0.85, size=20))
         X.shape = (X.shape[0],1)
         Y = np.append(spst.norm.rvs(loc=0,scale=1,
         ↪  size=15),spst.norm.rvs(loc=5,scale=0.85, size=10))
         Y.shape = (Y.shape[0],1)

         distXY = pedist(X, Y)

         hcur = spst.uniform.rvs()*5
         hprv = np.inf
         hini = hcur

         while np.abs(hcur-hprv)>1e-10:
             hprv = hcur
             wgts = wgcomp(distXY, hcur)
             hcur = np.sqrt(np.mean(np.sum(wgts*distXY**2,axis=0) /
              ↪  np.sum(wgts,axis=0)))

         # affichage
         hplt = np.linspace(start=0.01, stop=5, num=500)
         Z = np.linspace(start=-5, stop=10, num=1501)

         fig, axs = plt.subplots(1, 2)
         axs[0].plot(hplt, [loglike(h, X, Y) for h in hplt])
         axs[1].plot(Z, kdens(X, Z.reshape((Z.shape[0], 1)), hcur))
```



27 En pratique, on ne dispose souvent pas d'un échantillon $y_1, \ldots, y_m$ distinct pour calculer $h^\star$ (par exemple parce qu'on a trop peu de données disponibles). On « ré-utilise » donc l'échantillon $x_1, \ldots, x_n$ à la place de $y_1, \ldots, y_m$, mais *on prend soin d'écarter la valeur x_i de l'échantillon X lorsqu'on calcule la densité en x_i* : on obtient ainsi une *version leave-one-out* de la log-vraisemblance :

$$\ln L_{loo}(h; x_1, \ldots; x_m) = \sum_{i=1}^{n} \ln \widehat{f}_{loo}(x_i; h), \quad \widehat{f}_{loo}(x_i; h) = \frac{1}{n-1} \sum_{j \neq i} (2\pi)^{-1/2} h^{-1} \exp\left(-\frac{1}{2} \frac{(x_i - x_j)^2}{h^2}\right).$$

Modifier le code précédent pour implémenter la stratégie d'estimation par maximisation de la log-vraisemblance leave-one-out.

Il suffit en fait de modifier le calcul des distances, de manière à supprimer les éléments diagonaux de la matrice `distXY` (en décalant les éléments à droite de la diagonale vers la gauche). On peut modifier les fonctions existantes :

```
In [27]: def pedistloo(X):
             distXX = pedist(X, X)
             distXXloo = distXX[~np.eye(len(distXX),
             ↪  dtype=bool)].reshape(len(distXX), -1)
             return distXXloo

         def kdensloo(X, h):
             distXXloo = pedistloo(X)
             weights = wgcomp(distXXloo, h)
             dens = np.mean(weights/np.sqrt(2*np.pi)/h,axis=0)
             return dens

         def loglikeloo(h, X):
             dens = kdensloo(X, h)
             logl = np.sum(np.log(dens))
             return logl
```

On pourra alors calculer l'estimation $h^\star_{loo}$ et afficher le résultat de la manière suivante :

In [28]:
```
distXXloo = pedistloo(X)

hcur_ = hini
hprv = np.inf

while np.abs(hcur_-hprv)>1e-10:
    hprv = hcur_
    wgts = wgcomp(distXXloo, hcur_)
    hcur_ = np.sqrt(np.mean(np.sum(wgts*distXXloo**2,axis=0) /
    ↪  np.sum(wgts,axis=0)))

hplt = np.linspace(start=0.02, stop=5, num=250)
Z = np.linspace(start=-5, stop=10, num=1501)

fig, axs = plt.subplots(2, 2)
axs[0,0].plot(hplt, [loglike(h, X, Y) for h in hplt])
axs[0,1].plot(Z, kdens(X, Z.reshape((Z.shape[0],1)), hcur))
axs[1,0].plot(hplt, [loglikeloo(h, X) for h in hplt])
axs[1,1].plot(Z, kdens(X, Z.reshape((Z.shape[0],1)), hcur_))
```