TD 1
Bases Python et Statistiques

Ce TD a pour objectif de vous faire développer un petit module de statistiques *from scratch*. Vous démarrerez depuis l'écriture de quelques lignes de code directement dans l'interpréteur, pour atteindre plusieurs fichiers dont un importable et une classe. Durant tout l'exercice, vous utiliserez au fur et à mesure chaque notion clé du langage Python. Ces notions vous permettront d'utiliser plus tard les extensions et modules Python dédiés ou optimisés pour l'apprentissage automatique, le calcul parallèle, ou bien même pour d'autres usages dans la sécurité ou le scripting générique.

1 Prise en main de Python

Ouvrez un terminal disposant d'un shell, puis lancez python en tapant simplement la commande :

```
python
```

Assurez-vous d'avoir au minimum une version 3.8.

Afin de prendre en main l'interpréteur et le langage, nous allons démarrer en faisant des statistiques simples.
Commencez par importer le module **`random`** afin de disposer d'un générateur de nombres aléatoires, ainsi que le module **`math`**.

```
import math
import random
```

Le module **`random`** contient une fonction **`random()`** affichant un nombre entre 0 et 1 (exclu).
Commencez par afficher un nombre aléatoire en tapant :

```
random.random()
```

1) Vérifiez le type de la valeur retournée.
2) Multipliez le nombre par 100, puis transformez-le en entier, afin d'obtenir un pourcentage.
3) Générez maintenant 10 nombres aléatoires avec une boucle **`for`**. Ceci correspondra à une distribution de 10 nombres.

Afin de vous faire gagner du temps dans la correction syntaxique, et surtout pour travailler correctement, vous devrez continuer l'exercice en tapant votre code dans des fichiers **`.py`**. Vous pourrez ainsi sauvegarder vos travaux et nous les rendre.
Néanmoins, vous pouvez utilisez le shell python pour tester certaines opérations rapidement.

2 Premier script

Avant de continuer, créez un fichier **`simple_stats.py`** dans lequel vous écrirez votre code.
N'oubliez pas de mettre sur la première ligne du fichier :

```
#! /usr/bin/python
```

Recopiez ensuite la boucle **`for`** de la question précédente afin de générer 10 nombres aléatoires.

4) Testez votre script avec la commande suivante :

```
python simple_stats.py
```

Étant donné que vous n'avez pas demandé à écrire sur la sortie standard les résultats, le script ne produit rien.

5) Utilisez la fonction **`print`** pour écrire les nombres générés, puis la quantité de nombres générés.

$$\text{Nombre de Valeurs} = \text{cardinal de la distribution} = card(L) = |L|$$

Continuons le code précédent afin de calculer manuellement quelques statistiques sur notre distribution aléatoire.

6) Assignez le plus petit nombre de la distribution à une variable nommée **`DistrMin`**, et faites de même pour le plus grand nombre dans la variable **`DistrMax`**. Affichez ensuite les deux variables sur la sortie standard.

7) Calculez et affichez l'*étendue* de la distribution en faisant la différence entre la valeur la plus élevée de la distribution à la valeur la plus petite.

$$\text{étendue} = E = max(L) - min(L)$$

8) Calculez et affichez la *moyenne* de la distribution en faisant la somme de toutes les valeurs, puis en divisant par le nombre de valeurs.

$$\text{moyenne} = \mu = \bar{L} = \frac{\sum L_i}{|L|}$$

3 Listes

Afin de permettre à l'utilisateur de choisir le nombre de valeurs dans la distribution, utilisez le module **`sys`** pour pouvoir lire les arguments donnés par le shell. Pour rappel, il suffit de lire la séquence **`argv`** fournie par le module **`sys`**, sans oublier de vérifier sa taille.

9) Votre script doit maintenant vérifier si un paramètre est fourni : si oui, le premier paramètre correspondra au nombre de valeurs que la distribution doit comporter, si non, la distribution sera composée de 10 valeurs. Si le nombre donné en paramètre est inférieur ou égal à 0, la distribution sera composée de 1 nombre. Le nombre de valeurs de la distribution sera conservé dans la variable nommée **`DistrNbValues`**.

Pour mieux manipuler votre distribution, nous allons créer une liste dans une variable nommée **`MyDistribution`**.

10) Créez une liste vide **`MyDistribution`**, puis remplissez-la au fur-et-à-mesure avec la méthode **`append()`**.

11) Calculez et affichez la *médiane* de la distribution en choisissant la valeur au centre de la distribution **triée**. Utilisez la valeur par défaut (**`floor`** du module **`math`**) en cas de quantité impaire de valeurs dans la distribution (les tableaux démarrant à 0, on y applique donc déjà le +1).

$$\text{médiane} = L_{\frac{\lfloor L \rfloor}{2}}$$

4 Fonctions

Votre code devenant de plus en plus complexe, mais de plus en plus structuré, vous allez maintenant écrire des fonctions. Étant donné que votre code contient déjà ces fonctions de façon très/trop intégrée, cette étape correspond au processus de *refactoring* dans l'industrie : le code est démêlé et réécrit afin de pouvoir être mieux maintenu par de nouvelles équipes n'ayant pas toujours participé à l'écriture initiale (donc n'ayant pas vécues les mêmes contextes, contraintes, et exigences que la première équipe).

12) Implémentez une fonction **`GetDistrNbValues`** qui ne prend aucun paramètre, mais renvoie le nombre de valeurs demandé en argument dans la ligne de commande par l'utilisateur.
13) Implémentez une fonction **`Mean`** calculant la moyenne à partir de la liste contenant votre distribution.
14) Implémentez une fonction **`Median`** calculant la médiane à partir de la liste contenant votre distribution.

Pour profiter des retours multiples de fonctions de Python, vous allez également écrire une fonction renvoyant simultanément le minimum, le maximum, et la distribution.

15) Implémentez une fonction **`GenerateDistribution`** prenant en paramètre le nombre souhaité de valeurs dans la distribution, et renvoyant dans cet ordre précis : la valeur minimum, la valeur maximum, et la liste contenant la distribution.
16) Implémentez une fonction **`main`** récupérant dans la variable **`DistrNbValues`** le nombre souhaité de valeurs dans la distribution, puis créant la distribution **`MyDistribution`** et stockant le minimum et le maximum dans les variables **`DistrMin`** et **`DistrMax`**. Cette fonction sera la seule appelée dans le script : aucune autre instruction ne sera présente hors des fonctions.

Parmi les mesures statistiques utiles se trouve l'*écart interquartile* basé sur la différence entre des *quartiles*. L'idée des quartiles est de diviser la distribution en 4 parties, chaque quartile étant un séparateur, et d'observer l'écart entre certains séparateurs pour mesurer la dispersion des valeurs.

— le quartile 0 (Q_0) est la valeur la plus petite de la distribution
— le 1^{er} quartile (Q_1) sépare les 25% premières valeurs de la distribution des autres
— le $2^{ème}$ quartile (Q_2) sépare les 50% premières valeurs de la distribution des autres
— le $3^{ème}$ quartile (Q_3) sépare les 75% premières valeurs de la distribution des autres
— le quartile 4 (Q_4) est la valeur la plus élevée de la distribution

Plus visuellement, la distribution triée est séparée ainsi :

$$Q_0 \text{ ... (partie 1) ... } Q_1 \text{ ... (partie 2) ... } Q_2 \text{ ... (partie 3) ... } Q_3 \text{ ... (partie 4) ... } Q_4$$

Dans le cas dit *discret* (c'est-à-dire non continu : on peut dénombrer les valeurs que l'on étudie), il suffit de calculer le *rang* des valeurs, c'est-à-dire leur position dans la distribution triée par ordre croissant. Un quartile dans le cas discret est donc simplement une valeur à une position précise dans la distribution triée.

Pour une distribution contenant n valeurs, on calcule les rangs ainsi :

— le quartile 0 (Q_0) est donc au rang 1
— le 1^{er} quartile (Q_1) est au rang $(n+3)/4$
— le $2^{ème}$ quartile (Q_2) est au rang $(n+1)/2$ (il s'agit de la médiane)
— le $3^{ème}$ quartile (Q_3) est au rang $(3n+1)/4$
— le quartile 4 (Q_4) est donc au rang n

Testons sur la distribution : 1 2 3 4 5 6 7 8 9

— $Q_0 = 1$
— $Q_1 = 3$
— $Q_2 = 5$
— $Q_3 = 7$
— $Q_4 = 9$

La première partie contient donc les valeurs de 1 2 3, la deuxième partie les valeurs de 3 4 5, la troisième partie les valeurs 5 6 7, et la quatrième partie les valeurs 7 8 9.

Si on ne tombe pas sur un rang précis, il est normalement nécessaire de calculer un poids entre la valeur au dessus et la valeur au dessous. Mais pour simplifier notre cas, utilisons le rang inférieur (avec **`math.floor`**).

17) Créez cinq fonctions nommées **`QuartileZero`**, **`QuartileOne`**, **`QuartileTwo`**, **`QuartileThree`**, **`QuartileFour`** calculant la valeur de chaque quartile pour une distribution donnée.

Revenons à la métrique qui nous intéressait : l'écart interquartile. Celui-ci se calcule simplement ainsi :

$$\text{écart interquartile} = EI = Q_3 - Q_1$$

18) Implémentez le calcul de l'*écart interquartile* dans une fonction nommée **`InterQuartileRange`** qui prendra une distribution en paramètre.

19) Implémentez maintenant le calcul de la *variance* de la distribution dans une fonction nommée **`Variance`**. Cette fonction prendra en paramètre la distribution, et retournera la variance.

La variance permet de mesurer la dispersion des valeurs de la distribution par rapport à la moyenne. L'interprétation est simple : une variance élevée indique que les nombres de la distribution sont éloignés de la moyenne (1 et 99 par rapport à 50), une variance faible indique que les nombres de la distribution sont proches de la moyenne (42 et 58 par rapport à 50).

$$\text{variance} = \sigma^2 = Var(L) = \frac{1}{|L|}\sum(L_i - \bar{L})^2$$

Ce calcul peut paraître rebutant, mais il est très simple lorsque l'on regarde de plus près chaque partie de l'expression :

- $(L_i - \bar{L})$ correspond à la différence entre chaque élément de la distribution et la moyenne de la distribution.
- $(L_i - \bar{L})^2$ correspond au carré de la différence entre chaque élément de la distribution et la moyenne de la distribution.
- $\sum(L_i - \bar{L})^2$ correspond à la somme des carrés précédemment décrits. Il s'agit donc de faire une boucle effectuant les traitements précédemment décrits, pour ajouter le résultat à une variable initialisée à 0 juste avant la boucle.
- $\frac{1}{|L|}\sum(L_i - \bar{L})^2$ correspond à la division par le nombre d'éléments de la distribution du total de la somme précédente.

En développant le calcul, on peut arriver à une formule beaucoup plus adaptée au développement, surtout avec des distributions extrêmement grandes (c'est-à-dire des distributions sur lesquelles il coûterait très/trop cher de passer plusieurs fois pour calculer tout d'abord la *moyenne*, et seulement ensuite la différence de chaque élément avec la moyenne), ou lorsque les valeurs arrivent au fur et à mesure sans savoir précisément quand la distribution s'arrêtera (et que l'on souhaite un aperçu des statistiques en temps réel) :

$$\text{variance} = \sigma^2 = Var(L) = \frac{1}{|L|}\sum(L_i - \bar{L})^2 = \left(\frac{1}{|L|}\sum L_i^2\right) - \bar{L}^2$$

Analysons cette deuxième formule pour comprendre son intérêt en tant que développeur :

- L_i^2 correspond à la mise au carré de chaque élément de la distribution.
- $\sum L_i^2$ correspond à la somme de tous les éléments de la distribution mis au carré individuellement, c'est-à-dire d'itérer sur chaque élément, en faire son propre carré, puis d'ajouter ce résultat à une variable mise à 0 juste avant la boucle.
- $\frac{1}{|L|}\sum L_i^2$ correspond à la division de la somme précédente par le nombre d'éléments de la distribution.
- $(\frac{1}{|L|}\sum L_i^2) - \bar{L}^2$ correspond à la différence entre la division précédente, et la moyenne de la distribution mise au carré.

Cette deuxième formule est en réalité bien plus efficace dans un contexte séquentiel (c'est-à-dire lorsque chaque opération est effectuée l'une après l'autre), car nous pouvons faire une somme de tous les éléments qui arrivent et une somme de chacun de leur carré, et seulement lorsque tous les nombres ont été analysés, nous pouvons en déduire la moyenne et la soustraire. Chaque élément n'aura été lu qu'une seule et unique fois lorsqu'il a été généré, et nous avons fait évolué deux variables distinctes à côté : une somme simple (pour produire la moyenne), et une somme des carrés.

20) Implémentez maintenant le calcul de l'*écart type* de la distribution dans une fonction nommée **`StandardDeviation`**. Cette fonction prendra en paramètre la distribution, et retournera l'écart type.

L'écart type permet de mesurer la dispersion d'une distribution. Plus la mesure est élevée, plus la distribution contient des valeurs éloignées (3 et 90), et plus la mesure est faible, plus la distribution contient des valeurs proches (42 et 46). On peut comparer les écarts types de distributions issues de mêmes choses (par exemple, les écarts types de plusieurs classes composées d'étudiants).

$$\text{écart type} = \sigma = \sqrt{Var(L)}$$

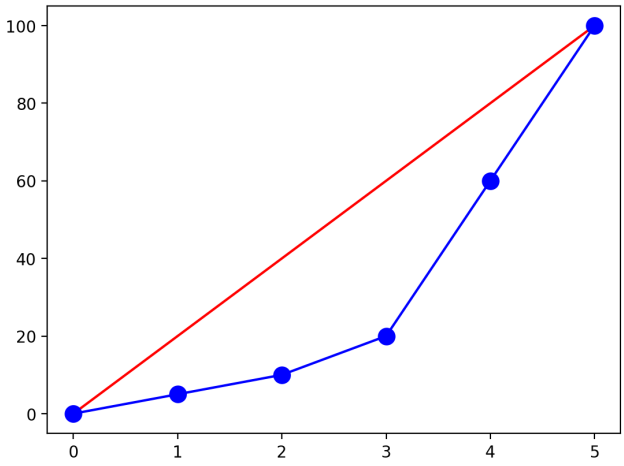
21) Implémentez maintenant le calcul du *coefficient de variation* (ou *écart type relatif*) de la distribution dans une fonction nommée **`RelativeStandardDeviation`**. Cette fonction prendra en paramètre la distribution, et retournera le coefficient de variation.

L'écart type relatif permet de mesurer la dispersion d'une distribution autour de la moyenne en générant un pourcentage indépendant des objets étudiés par la distribution. Plus le pourcentage est faible, plus la distribution est concentrée autour de sa moyenne, plus le pourcentage est élevé, plus la distribution est étalée loin de sa moyenne. Cependant, l'écart type relatif ne fonctionne pas lorsque la moyenne est proche de 0 : celui-ci va tendre vers l'infini (donc dépasser les 100%) et sera très sensible aux variations.

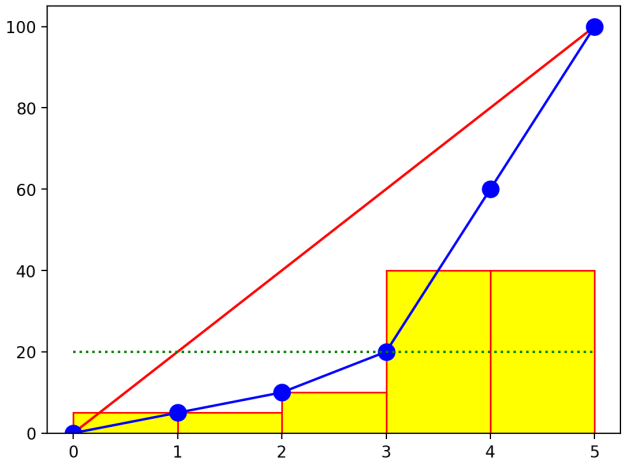
$$\text{coefficient de variation} = c_v = \frac{\sigma}{\mu} = \frac{\sqrt{Var(L)}}{\bar{L}}$$

Un autre indice intéressant issu particulièrement du monde économique et social se nomme le *coefficient de Gini*. Ce coefficient a été constitué pour mesurer les inégalités de revenus au sein d'une population, mais il permet plus généralement de mesurer l'inégalité de répartition d'une variable (donc son interprétation s'approche des mesures de dispersion). Le coefficient de Gini calcule une valeur entre 0 (égalité parfaite entre toutes les valeurs de la distribution) et 1 (inégalité totale/une seule valeur est positive, et toutes les autres sont nulles).
Le calcul théorique de ce coefficient peut paraitre très rebutant à cause des termes employés, et pourtant, il est très facile à comprendre. De plus, le calcul pratique dans le cas discret (pour rappel : non continu/on peut dénombrer les valeurs de la distribution), n'est pas complexe.

Le coefficient de Gini est défini comme le double de l'aire entre la courbe de Lorenz d'une distribution parfaitement égalitaire, et la courbe de Lorenz de la distribution étudiée.
Qu'est-ce qu'une courbe de Lorenz ? Tout simplement la courbe formée par la somme des éléments triés par ordre croissant d'une distribution. Plus visuellement, le tracé bleu correspond à la courbe de Lorenz de la distribution 5 10 40 5 40, et le tracé rouge correspond à la courbe de Lorenz d'une distribution parfaitement égalitaire.



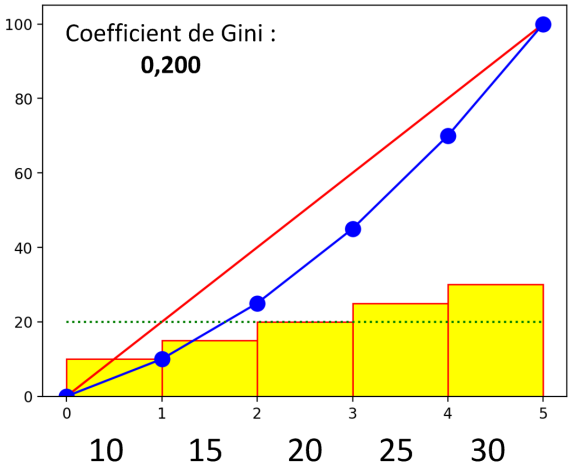
Encore plus visuellement, une fois la distribution ordonnée, on obtient donc 5 5 10 40 40. L'histogramme suivant montre visuellement chaque valeur ajoutée entre les points de la courbe bleue. Donc, en effectuant la somme des valeurs au fur et à mesure, chaque point de la courbe bleue correspond à 5 10 20 60 100.

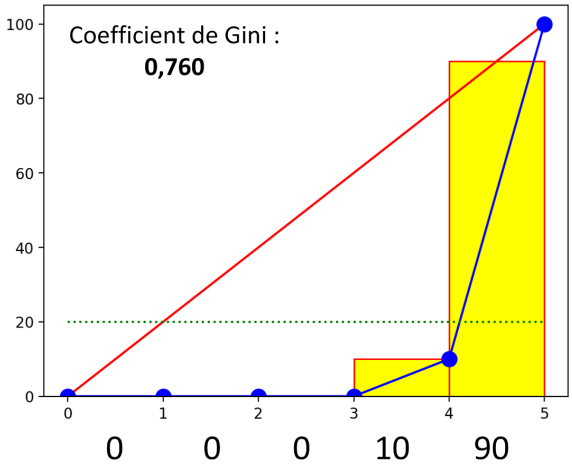


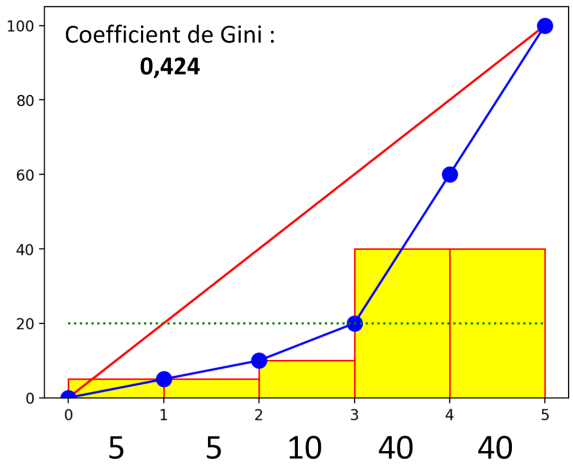
Pour rappel, le coefficient de Gini correspond au double de l'aire entre la courbe bleue et la courbe rouge. Pour bien apprécier visuellement l'écart de cette aire, prenons maintenant deux autres distributions, et comparons l'ensemble des cas :

- 0 0 0 10 90 (une distribution très inégale) : Coeff Gini $= 0,760$
- 5 5 10 40 40 (une distribution plutôt inégale) : Coeff Gini $= 0,424$
- 10 15 20 25 30 (une distribution peu inégale) : Coeff Gini $= 0,200$

Pour bien visualiser ces cas, la moyenne des distributions (20) a été tracée en pointillés verts.







Comme vous pouvez le constater, le cas peu inégal montre une courbe bleue proche de la courbe rouge, et inversement, le cas très inégal montre une courbe bleue très éloignée de la courbe rouge. Pour rappel, la moyenne n'est pas un indicateur de dispersion, mais bien un indicateur de tendance centrale ou de position. Les nombreux indicateurs que vous venez d'implémenter vous permettront donc de mieux interpréter les données dont vous disposerez dans votre carrière, et éventuellement de voir des phénomènes jusque-là peu visibles.

Reprenons l'implémentation du coefficient de Gini : vous venez de comprendre comment celui-ci est théoriquement construit et interprété. Mais comment l'implémenter facilement ?
Deux formules nous intéressent particulièrement, la première est l'équation de Kendal et Stuart. Dans celle-ci, n correspond au nombre d'éléments dans la distribution, et $L_1, L_2, ..., L_n$ correspondent à chaque élément de la distribution.

$$G = \frac{(1/2n^2)\sum_{i=1}^{n}\sum_{j=1}^{n}|L_i - L_j|}{(1/n)\sum_{i=1}^{n} L_i}$$

En observant cette formule, vous devriez remarquer que la double somme teste littéralement tous les éléments de la distribution deux à deux, ce qui est particulièrement long. Et qu'il est par la suite nécessaire de diviser le tout par la somme des éléments.

Si vous implémentez chacune des parties indépendamment, le temps de calcul sera particulièrement long. Et si vous implémenter toute la formule dans une fonction, vous aurez beaucoup de variables à conserver, et le code risque d'être difficile à comprendre.

Heureusement, vous devriez remarquer que le diviseur est constitué du nombre d'éléments de la distribution, et de leur somme... ce qui ressemble à la moyenne. Et effectivement, Mookherjee et Shorrocks ont simplifié cette équation en introduisant la moyenne (μ) :

$$G = \frac{1}{2\mu n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} |L_i - L_j|$$

Bien que cette formule semble encore complexe, décomposons-là :

- $L_i - L_j$ correspond à la différence entre chaque élément de la distribution.
- $|L_i - L_j|$ correspond à la valeur absolue de la différence entre chaque élément de la distribution.
- $\sum_{i=1}^{n} \sum_{j=1}^{n} |L_i - L_j|$ correspond à la somme des différences absolues des éléments de la distribution.
  Il s'agit donc de faire une double boucle effectuant la différence entre chaque élément de la distribution, pour ajouter le résultat à une variable initialisée à 0 juste avant la boucle. La double boucle implique simplement d'itérer comme sur un tableau à deux dimensions ($|L_1 - L_1| + |L_1 - L_2| + ... + |L_2 - L_1| + |L_2 - L_2| + ... + |L_n - L_n|$).
- $2\mu n^2$ correspond simplement à 2 multiplié par la moyenne de la distribution, multiplié par la quantité d'éléments (le cardinal) au carré.

Pour conclure, pensez tout de même que le deux formules peuvent être optimisées lors de leur implémentation (on peut calculer la moyenne au fur et à mesure des boucles, ou dans d'autres cas, voire, on peut accumuler la somme des éléments dans une variable).

5 Modules

Une fois toutes les fonctions écrites, nous allons créer un second fichier **`MyOwnStats.py`** qui servira de module.

Ce module va contenir toutes les fonctions que vous avez précédemment écrites, exceptée la fonction **`main()`**.

22) Copiez les fonctions, sauf **`main()`**, dans le nouveau fichier.

Une fois les fonctions copiées dans le nouveau fichier, enregistrez votre travail, et supprimez les fonctions du fichier **`simple_stats.py`**. Faites maintenant un ajout de module dans votre ancien script, afin d'importer les fonctions, et mettez à jour le contenu de votre fonction **`main()`**.

23) Supprimez les fonctions du précédent script, importez le module que vous venez de créer, et mettez à jour la fonction **`main()`** pour appeler correctement votre module et vos fonctions.

6 Classes et Objets

Une fois le module écrit, vous allez copier le fichier **`MyOwnStats.py`** vers un troisième fichier nommé **`MyStats.py`** dans lequel vous allez créer une classe **`MyStats`**. Cette classe va contenir toutes les fonctions que vous avez écrit précédemment (sans **`main()`**), ainsi que la distribution étudiée comme attribut.

Le constructeur de cette classe prendra un paramètre optionnel nommé **`NbValues`** qui représentera le nombre de valeurs à générer. Si aucune valeur n'est indiquée, on assignera 10 à **`NbValues`**. Vous l'aurez compris, cette classe permettra d'étudier les statistiques d'une distribution générée aléatoirement.

24) Écrivez une classe **`MyStats`** (dans le fichier **`MyStats.py`**) qui contiendra une liste de nombres en attribut, et les différentes fonctions de génération de distribution et calculs de mesures statistiques. Le constructeur prendra un paramètre optionnel **`NbValues`** qui sera mis à 10 s'il n'est pas précisé par l'utilisateur.

Un deuxième paramètre optionnel sera ajouté au constructeur et prendra une liste de nombres en paramètre. Le constructeur copiera chacun des nombres de cette liste vers celle en attribut de la classe.

Si vous souhaitez optimiser les calculs des différentes mesures précédemment décrites, vous pouvez tout à fait effectuer l'ensemble des calculs de moyenne, variance, et autres lors de la création de chaque objet. Ainsi, l'appel aux méthodes censées calculer et renvoyer les valeurs ne feront que renvoyer les résultats stockés dans l'objet.

25) Créez un nouveau script nommé **`simple_stats_obj.py`** qui instanciera deux fois la classe **`MyStats`** avec le même nombre de valeurs, et comparez chacune des mesures précédemment codées.