Python

Statistiques

12 septembre 2023

Version 3

Fabrice BOISSIER <fabrice.boissier@epita.fr>

Copyright

Table des matières

1 Consignes Générales

Les informations suivantes sont très importantes :

Le non-respect d'une des consignes suivantes entraînera des sanctions pouvant aller jusqu'à la multiplication de la note finale par 0.

Ces consignes sont claires, non-ambiguës, et ont un objectif précis. En outre, elles ne sont pas négociables.

N'hésitez pas à demander si vous ne comprenez pas une des règles.

Consigne Générale 0 : Vous devez lire le sujet.
Consigne Générale 1 : Vous devez respecter les consignes.
Consigne Générale 2 : Vous devez rendre le travail dans les délais prévus.

Consigne Générale 3 : Le travail doit être rendu dans le format décrit à la section Format de Rendu.

Consigne Générale 4 : Le travail rendu ne doit pas contenir de fichiers binaires, temporaires, ou d'erreurs (`*~`, `*.o`, `*.a`, `*.so`, `*#*`, `*core`, `*.log`, `*.exe`, binaires, ...).

Consigne Générale 5 : Dans l'ensemble de ce document, la casse (caractères majuscules et minuscules) est très importante. Vous devez strictement respecter les majuscules et minuscules imposées dans les messages et noms de fichiers du sujet.

Consigne Générale 6 : Dans l'ensemble de ce document, **`login`** correspond à votre login.

Consigne Générale 7 : Dans l'ensemble de ce document, **`nom1-nom2`** correspond à la combinaison des deux noms de votre binôme (par exemple pour Fabrice BOISSIER et Mark ANGOUSTURES, cela donnera **`boissier-angoustures`**).

Consigne Générale 8 : Dans l'ensemble de ce document, le caractère ␣ correspond à une espace (s'il vous est demandé d'afficher ␣␣␣, vous devez afficher trois espaces consécutives).

Consigne Générale 9 : Tout retard, même d'une seconde, entraîne la note non négociable de 0.

Consigne Générale 10 : La triche (échange de code, copie de code ou de texte, ...) entraîne **au mieux** la note non négociable de 0.

Consigne Générale 11 : En cas de problème avec le projet, vous devez contacter le plus tôt possible les responsables du sujet aux adresses mail indiquées.

Conseil : N'attendez pas la dernière minute pour commencer à travailler sur le sujet.

2 Format de Rendu

Responsable(s) du projet :	**Fabrice BOISSIER** <fabrice.boissier@epita.fr> **Lisa MONPIERRE** <lisa.monpierre@epita.fr>
Balise(s) du projet :	**[PYTHON][TP1]**
Nombre d'étudiant(s) par rendu :	1
Procédure de rendu :	Devoir/Assignment sur Teams
Nom du répertoire :	`login-Python-Stats`
Nom de l'archive :	`login-Python-Stats.tar.bz2`
Date maximale de rendu :	29/10/2022 23h42
Durée du projet :	1,5 mois
Architecture/OS :	Linux - Ubuntu (x86_64)
Langage(s) :	Python
Compilateur/Interpréteur :	**/usr/bin/python3.10**
Options du compilateur/interpréteur :	

Les fichiers suivants sont requis :

`AUTHORS`	contient le(s) nom(s) et prénom(s) de(s) auteur(s).
`README`	contient la description du projet et des exercices, ainsi que la façon d'utiliser le projet.

Votre code sera testé automatiquement, vous devez donc scrupuleusement respecter les spécifications pour pouvoir obtenir des points en validant les exercices. Votre code sera testé en appelant chaque script avec l'interpréteur python (et éventuellement un ou des arguments) :

```
python3.10 script.py arg1 arg2 [...]
```

L'arborescence attendue pour le projet est la suivante :

```
login-Python-Stats/
login-Python-Stats/AUTHORS
login-Python-Stats/README
login-Python-Stats/src/
login-Python-Stats/src/class/
login-Python-Stats/src/class/MyOwnStats.py
login-Python-Stats/src/module/
login-Python-Stats/src/module/MyOwnStats.py
```

3 Aide Mémoire

Le travail doit être rendu au format ***.tar.bz2***, c'est-à-dire une archive **bz2** compressée avec un outil adapté (voir **`man 1 tar`** et **`man 1 bz2`**).
Tout autre format d'archive (zip, rar, 7zip, gz, gzip, ...) ne sera pas pris en compte, et votre travail ne sera pas corrigé (entraînant la note de 0).

Pour générer une archive *tar* en y mettant les dossiers *folder1* et *folder2*, vous devez taper :

```
tar cvf MyTarball.tar folder1 folder2
```

Pour générer une archive *tar* et la compresser avec GZip, vous devez taper :

```
tar cvzf MyTarball.tar.gz folder1 folder2
```

Pour générer une archive *tar* et la compresser avec BZip2, vous devez taper :

```
tar cvjf MyTarball.tar.bz2 folder1 folder2
```

Pour lister le contenu d'une archive *tar*, vous devez taper :

```
tar tf MyTarball.tar.bz2
```

Pour extraire le contenu d'une archive *tar*, vous devez taper :

```
tar xvf MyTarball.tar.bz2
```

Dans ce sujet précis, vous ferez du code en Python, qui affichera les résultats dans le terminal (donc des flux de sortie qui pourront être redirigés vers un fichier texte).

4 Exercice 1 - Module statistiques

Nom du(es) fichier(s) :	**`MyOwnStats.py`**
Répertoire :	**`login-Python-Stats/src/module/`**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions externes autorisées :	**`math.sqrt`**, **`random.random`**, **`random.randint`**

Objectif : Le but de l'exercice est d'écrire un module python contenant plusieurs fonctions permettant de calculer des statistiques sur une distribution.

Les fonctions statistiques seront implémentées dans leur version évaluant une population complète et non pas un échantillon, elles renverront des flottants pour la plupart (sauf celles traitant des rangs ou des valeurs directement).

Vous devez implémenter les fonctions décrites plus bas, si nécessaire en déclarant vos propres fonctions. Ce code sera testé en étant lui-même appelé par un autre script, vous devez donc absolument respecter les prototypes décrits ici.

Vous ne devez pas implémenter de classe dans cet exercice, mais bel et bien un module. Vous ne devez pas non plus importer la classe de l'exercice 2.

Pour construire ce module, vous avez le droit d'importer exclusivement 3 fonctions de modules externes :

- **`math.sqrt`** du module `math`
- **`random.random`** du module `random`
- **`random.randint`** du module `random`

Cependant, vous pouvez parfaitement utiliser les fonctions, types, et classes internes à Python (tels que **`print()`**, **`int()`**, **`str()`**, **`range()`**, **`list()`**, et autres).

Titre

Notez bien qu'aucune fonction de ce module ne doit modifier la distribution donnée en paramètre ! Pas même les fonctions de tri, d'ajout, de suppression, ou de fusion, ni les autres fonctions ayant besoin que la distribution soit triée.

Vous devez implémenter les fonctions suivantes :

```
GenerateDistribution(DistrNbValues)
SortDistribution(Distribution, [asc=True])
PrintDistribution(Distribution)

AddValueDistribution(Distribution, Value)
AddValuePositionDistribution(Distribution, Value, Position)
DeleteValueDistribution(Distribution, Value)
DeletePositionDistribution(Distribution, Position)
MergeDistribution(Distribution, NewDistribution)

Min(Distribution)
Max(Distribution)
Cardinality(Distribution)
Range(Distribution)

Sum(Distribution)
Mean(Distribution)
Median(Distribution)
GetQuartile(Distribution, Quartile)

InterQuartileRange(Distribution)
Variance(Distribution)
StandardDeviation(Distribution)
RelativeStandardDeviation(Distribution)
GiniCoefficient(Distribution)
```

Liste des fonctions pour le module de statistiques

`GenerateDistribution(DistrNbValues)`

Cette fonction crée et renvoie un tuple d'entiers aléatoires représentants une distribution, ainsi que les valeurs minimale et maximale. Les valeurs doivent être renvoyées exactement dans cet ordre : la valeur minimum, la valeur maximum, et le tuple contenant la distribution. La distribution créée peut contenir plusieurs fois la même valeur, et doit être composée de nombres entiers entre 0 et 1000. Le paramètre *DistrNbValues* est un entier indiquant le nombre d'entiers que la distribution doit contenir. Si *DistrNbValues* est nul, alors vous renverrez un tuple vide, avec le minimum et le maximum à 0. Si *DistrNbValues* est négatif, alors vous renverrez une distribution contenant 10 entiers.

`SortDistribution(Distribution, [asc=True])`

Cette fonction renvoie la version triée de la distribution donnée en paramètre. Le paramètre *Distribution* est la distribution à trier (il doit s'agir d'un tuple d'entiers). Le paramètre *asc* est un booléen indiquant si le tuple doit être trié par ordre ascendant ou descendant. Par défaut, le tri doit être ascendant (à la position 0 doit se trouver l'entier

le plus petit de la distribution, et à la dernière position on doit trouver l'entier le plus grand).

Attention : cette fonction ne doit pas modifier le paramètre donné, mais bien renvoyer une copie triée !

`PrintDistribution(Distribution)`

Cette fonction écrit le contenu de la distribution sur la sortie standard. Le format attendu est simple : un seul élément et sa position doivent être affichés par ligne.

`[position] : valeur`

Ce qui donnerait pour la distribution $[10, 5, 42]$:

```
$ python3.10 example.py
[0] : 10
[1] : 5
[2] : 42
$
```

Si la distribution est vide, vous n'afficherez rien.

`AddValueDistribution(Distribution, Value)`

Cette fonction renvoie une distribution contenant la valeur ajoutée à la distribution donnée en paramètre.
Le paramètre *Value* est la valeur à ajouter. La valeur supplémentaire s'ajoute en fin de tuple. On peut ajouter une valeur déjà existante.
Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message.

Ainsi, pour la distribution $[10, 5, 42]$, lorsque l'on ajoute la valeur 6, on doit obtenir la distribution suivante : $[10, 5, 42, 6]$.

`AddValuePositionDistribution(Distribution, Value, Position)`

Cette fonction renvoie une distribution contenant la valeur ajoutée à la position précise dans la distribution donnée en paramètre.
Le paramètre *Value* est la valeur à ajouter. On peut ajouter une valeur déjà existante.
Le paramètre *Position* indique la position dans le tuple où ajouter l'élément (l'ancien élément présent à cette position sera décalé à la position suivante : si on ajoute en position 0, l'élément qui y était sera poussé en position 1). Si la position est négative, on ajoute l'élément en position 0. Si la position est supérieure à la dernière position du tuple, on ajoute l'élément en fin de tuple/après la dernière position.
Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message.

`DeleteValueDistribution(Distribution, Value)`

Cette fonction renvoie une copie de la distribution donnée en paramètre dont on a supprimé une occurrence de la valeur indiquée (la première occurrence trouvée depuis la position 0). Le paramètre *Value* est la valeur à supprimer.
Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message. Si la valeur n'est pas trouvée dans le tuple, vous déclencherez une exception **`ValueNotFoundInDistribution`** sans afficher de message. Si la distribution ne contient plus aucune valeur au moment de l'appel à cette fonction, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message. L'exception **`ValueNotInteger`** prime sur l'exception **`DistributionEmpty`** qui elle-même prime sur l'exception **`ValueNotFoundInDistribution`**.

`DeletePositionDistribution(Distribution, Position)`

Cette fonction renvoie une copie de la distribution donnée en paramètre dont on a supprimé la valeur située à la position donnée en paramètre. Le paramètre *Position* est la position de la valeur à supprimer. Si la position est négative, vous supprimerez l'élément en position 0. Si la position est supérieure à la dernière position du tuple, vous supprimerez le dernier élément du tuple. Si la distribution donnée en paramètre est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`MergeDistribution(Distribution, NewDistribution)`

Cette fonction renvoie un tuple issu de la fusion des deux distributions données en paramètre. Le paramètre *NewDistribution* est la distribution à ajouter après le tuple en premier paramètre.

`Min(Distribution)`

Cette fonction renvoie la valeur minimale de la distribution. Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`Max(Distribution)`

Cette fonction renvoie la valeur maximale de la distribution. Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`Cardinality(Distribution)`

Cette fonction calcule et renvoie le *cardinal* de la distribution.

`Range(Distribution)`

Cette fonction calcule et renvoie l'*étendue* de la distribution. Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`Sum(Distribution)`

Cette fonction calcule et renvoie la *somme* des éléments de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`Mean(Distribution)`

Cette fonction calcule et renvoie la *moyenne* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`Median(Distribution)`

Cette fonction calcule et renvoie la *médiane* de la distribution (en float). Si la distribution a un nombre impair d'éléments, vous renverrez l'élément à la position par défaut/obtenue par la partie entière inférieure (par exemple, pour une distribution composée de 3 éléments, vous renverrez l'élément en position 1 et non pas en 2). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

Rappel : cette fonction ne doit pas modifier la distribution donnée en paramètre !

`GetQuartile(Distribution, Quartile)`

Cette fonction calcule et renvoie le *quartile* demandé de la distribution (en float). Le paramètre **`Quartile`** correspond à un entier entre 0 et 4 (inclus). Les quartiles 1, 2, et 3 correspondent aux quartiles séparant les différentes parties de la distribution. Le quartile 0 correspond à la plus petite valeur de la distribution, et le quartile 4 correspond à la plus grande valeur de la distribution.
Si le paramètre **`Quartile`** est inférieur à 0 ou supérieur à 4, vous déclencherez une exception **`IncorrectQuartileParameter`** sans afficher de message. En cas de position non entière, vous suivrez les indications du cours à la fin du document. Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher. L'exception **`IncorrectQuartileParameter`** prime sur l'exception **`DistributionEmpty`**.

Rappel : cette fonction ne doit pas modifier la distribution donnée en paramètre !

`InterQuartileRange(Distribution)`

Cette fonction calcule et renvoie l'*écart interquartile* de la distribution (en float). C'est-à-dire la différence entre le quartile 3 et le quartile 1. Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`Variance(Distribution)`

Cette fonction calcule et renvoie la *variance* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`StandardDeviation(Distribution)`

Cette fonction calcule et renvoie l'*écart type* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`RelativeStandardDeviation(Distribution)`

Cette fonction calcule et renvoie le *coefficient de variation* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`GiniCoefficient(Distribution)`

Cette fonction calcule et renvoie le *coefficient de Gini* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

5 Exercice 2 - Classe statistiques

Nom du(es) fichier(s) :	**`MyOwnStats.py`**
Répertoire :	**`login-Python-Stats/src/class/`**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions externes autorisées :	**`math.sqrt`**, **`random.random`**, **`random.randint`**

Objectif : Le but de l'exercice est d'écrire une classe python contenant une distribution et plusieurs méthodes permettant de calculer des statistiques dessus (comme le module de l'exercice 1, mais cette fois dans le paradigme objet).

Les fonctions statistiques seront implémentées dans leur version évaluant une population complète et non pas un échantillon, elles renverront des flottants pour la plupart (sauf celles traitant des rangs ou des valeurs directement).

Vous devez implémenter une classe **`MyDistribution`** contenant les méthodes décrites plus bas, si nécessaire en déclarant vos propres méthodes et attributs. Ce code sera testé en étant lui-même appelé par un autre script, vous devez donc absolument respecter les prototypes décrits ici.

Vous ne devez pas implémenter les fonctions d'un module dans cet exercice, mais bel et bien une classe et ses méthodes. Vous ne devez pas non plus importer le module de l'exercice 1, pour la bonne raison que vous pourrez cette fois optimiser votre code (par exemple en déclenchant automatiquement le re-calcul des indicateurs lors de l'ajout de nouvelles valeurs, ou en stockant dans un attribut dédié la somme des valeurs de la distribution courante, voire d'autres sommes ou calculs communs à plusieurs indicateurs). Vous pouvez donc ajouter des méthodes ou attributs supplémentaires utiles. Cependant, vous ne devez toujours pas trier la distribution sans qu'un appel à la méthode de tri ait été réalisé par l'utilisateur de votre classe.

Pour construire cette classe, vous avez le droit d'importer exclusivement 3 fonctions de modules externes :

- **`math.sqrt`** du module `math`
- **`random.random`** du module `random`
- **`random.randint`** du module `random`

Cependant, vous pouvez parfaitement utiliser les fonctions, types, et classes internes à Python (tels que **`print()`**, **`int()`**, **`str()`**, **`range()`**, **`list()`**, et autres).

Titre

Notez bien qu'aucune méthode de ce module ne doit modifier la distribution donnée en paramètre, exceptée la méthode de tri ! Pas même les méthodes d'ajout, de suppression, de fusion, ou les autres méthodes ayant besoin que la distribution soit triée.

Vous devez implémenter les méthodes suivantes :

```
__init__(self, DistrNbValues)
GetDistribution(self)
SortDistribution(self, [asc=True])
PrintDistribution(self)

AddValueDistribution(self, Value)
AddValuePositionDistribution(self, Value, Position)
DeleteValueDistribution(self, Value)
DeletePositionDistribution(self, Position)
MergeDistribution(self, NewDistribution)

Sum(self)
Min(self)
Max(self)
Cardinality(self)
Range(self)

Mean(self)
Median(self)
GetQuartile(self, Quartile)

InterQuartileRange(self)
Variance(self)
StandardDeviation(self)
RelativeStandardDeviation(self)
GiniCoefficient(self)
```

Liste des méthodes pour la classe de statistiques

`__init__(self, DistrNbValues)`

Le constructeur de la classe crée une liste d'entiers aléatoires représentants une distribution, et la conserve dans un attribut. La distribution créée peut contenir plusieurs fois la même valeur, et doit être composée de nombres entiers entre 0 et 1000. Le paramètre *DistrNbValues* est un entier indiquant le nombre d'entiers que la distribution doit contenir. Si *DistrNbValues* est nul, alors vous allouerez une liste vide. Si *DistrNbValues* est négatif, alors vous créerez une distribution contenant 10 entiers.

`GetDistribution(self)`

Cette méthode renvoie une copie de liste représentant la distribution.

`SortDistribution(self, [asc=True])`

Cette méthode trie la distribution de l'objet. Le paramètre *asc* est un booléen indiquant si la liste doit être triée par ordre ascendant ou descendant. Par défaut, le tri doit être

ascendant (à la position 0 doit se trouver l'entier le plus petit de la distribution, et à la dernière position on doit trouver l'entier le plus grand).

`PrintDistribution(self)`

Cette méthode écrit le contenu de la distribution sur la sortie standard. Le format attendu est simple : un seul élément et sa position doivent être affichés par ligne.

`[position] : valeur`

Ce qui donnerait pour la distribution $[10, 5, 42]$:

```
$ python3.10 example.py
[0] : 10
[1] : 5
[2] : 42
$
```

Si la distribution est vide, vous n'afficherez rien.

`AddValueDistribution(self, Value)`

Cette méthode ajoute la valeur donnée en paramètre à la distribution. Le paramètre *Value* est la valeur à ajouter. La valeur supplémentaire s'ajoute en fin de liste. On peut ajouter une valeur déjà existante. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message.

Ainsi, pour la distribution $[10, 5, 42]$, lorsque l'on ajoute la valeur 6, on doit obtenir la distribution suivante : $[10, 5, 42, 6]$.

`AddValuePositionDistribution(self, Value, Position)`

Cette méthode ajoute une valeur à la position précise dans la distribution. Le paramètre *Value* est la valeur à ajouter. On peut ajouter une valeur déjà existante. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message. Le paramètre *Position* indique la position dans la liste où ajouter l'élément (l'ancien élément présent à cette position sera décalé à la position suivante : si on ajoute en position 0, l'élément qui y était sera poussé en position 1). Si la position est négative, on ajoute l'élément en position 0. Si la position est supérieure à la dernière position de la liste, on ajoute l'élément en fin de liste/après la dernière position.

`DeleteValueDistribution(self, Value)`

Cette méthode supprime une occurrence de la valeur indiquée (la première occurrence trouvée depuis la position 0). Le paramètre *Value* est la valeur à supprimer.
Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message. Si la valeur n'est pas trouvée dans la liste, vous déclencherez une exception **`ValueNotFoundInDistribution`** sans afficher de message. Si la distribution ne contient plus aucune valeur au moment de l'appel à cette méthode, vous

déclencherez une exception **DistributionEmpty** sans afficher de message. L'exception **ValueNotInteger** prime sur l'exception **DistributionEmpty** qui elle-même prime sur l'exception **ValueNotFoundInDistribution**.

DeletePositionDistribution(self, Position)

Cette méthode supprime la valeur située à la position donnée en paramètre. Le paramètre *Position* est la position de la valeur à supprimer. Si la position est négative, vous supprimerez l'élément en position 0. Si la position est supérieure à la dernière position de la liste, vous supprimerez le dernier élément de la liste. Si la distribution ne contient plus aucune valeur au moment de l'appel à cette méthode, vous déclencherez une exception **DistributionEmpty** sans afficher de message.

MergeDistribution(self, NewDistribution)

Cette méthode fusionne la distribution donnée en paramètre à celle gérée par la classe. Le paramètre *NewDistribution* est la distribution à ajouter à la fin de la distribution déjà contenue dans l'objet.

Min(self)

Cette méthode renvoie la valeur minimale de la distribution. Si la distribution est vide, vous déclencherez une exception **DistributionEmpty** sans rien afficher.

Max(self)

Cette méthode renvoie la valeur maximale de la distribution. Si la distribution est vide, vous déclencherez une exception **DistributionEmpty** sans rien afficher.

Cardinality(self)

Cette méthode calcule et renvoie le *cardinal* de la distribution.

Range(self)

Cette méthode calcule et renvoie l'*étendue* de la distribution. Si la distribution est vide, vous déclencherez une exception **DistributionEmpty** sans rien afficher.

Sum(self)

Cette méthode calcule et renvoie la *somme* des éléments de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **DistributionEmpty** sans rien afficher.

Mean(self)

Cette méthode calcule et renvoie la *moyenne* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **DistributionEmpty** sans rien afficher.

`Median(self)`

Cette méthode calcule et renvoie la *médiane* de la distribution (en float). Si la distribution a un nombre impair d'éléments, vous renverrez l'élément à la position par défaut/obtenue par la partie entière inférieure (par exemple, pour une distribution composée de 3 éléments, vous renverrez l'élément en position 1 et non pas en 2). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

Rappel : cette fonction ne doit pas modifier la distribution contenue dans l'objet !

`GetQuartile(self, Quartile)`

Cette méthode calcule et renvoie le *quartile* demandé de la distribution (en float). Le paramètre **`Quartile`** correspond à un entier entre 0 et 4 (inclus). Les quartiles 1, 2, et 3 correspondent aux quartiles séparant les différentes parties de la distribution. Le quartile 0 correspond à la plus petite valeur de la distribution, et le quartile 4 correspond à la plus grande valeur de la distribution.
Si le paramètre **`Quartile`** est inférieur à 0 ou supérieur à 4, vous déclencherez une exception **`IncorrectQuartileParameter`** sans afficher de message. En cas de position non entière, vous suivrez les indications du cours à la fin du document. Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher. L'exception **`IncorrectQuartileParameter`** prime sur l'exception **`DistributionEmpty`**.

Rappel : cette fonction ne doit pas modifier la distribution contenue dans l'objet !

`InterQuartileRange(self)`

Cette méthode calcule et renvoie l'*écart interquartile* de la distribution (en float). C'est-à-dire la différence entre le quartile 3 et le quartile 1. Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`Variance(self)`

Cette méthode calcule et renvoie la *variance* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`StandardDeviation(self)`

Cette méthode calcule et renvoie l'*écart type* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

`RelativeStandardDeviation(self)`

Cette méthode calcule et renvoie le *coefficient de variation* de la distribution (en float). Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`GiniCoefficient(self)`

Cette méthode calcule et renvoie le *coefficient de Gini* de la distribution (en float). Si la distribution est vide, vous déclencherez une exception **`DistributionEmpty`** sans rien afficher.

Optionnel : si vous souhaitez augmenter la difficulté de l'exercice, n'hésitez pas à optimiser votre code de différentes manières. Vous pouvez typiquement recalculer certaines valeurs intermédiaires utiles lors de la mise à jour de la distribution (ajouter ou supprimer chaque valeur au fur et à mesure), et faire les calculs les plus lourds uniquement lorsqu'un développeur tiers demande explicitement une métrique. Vous pouvez également implémenter plusieurs fois cette classe et comparer dans quels scénarios d'usage chaque version sera la plus rapide : de nombreux ajouts et suppressions d'une valeur à chaque fois formant une grande liste avec quelques demandes de métriques dans l'ensemble du scénario, ou quelques fusions de grandes listes avec de nombreux accès à toutes les métriques possibles entre chaque fusion.

6 Formules statistiques

6.1 Cardinal

Le *cardinal* est le nombre de valeurs contenues dans une distribution.

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

6.2 Étendue

L'*étendue* est la différence entre la plus grande valeur de la distribution et la plus petite valeur de la distribution.

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

6.3 Moyenne

La *moyenne arithmétique*, communément appelée *moyenne*, est la somme de tous les éléments de la distribution divisée par le cardinal.

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

6.4 Médiane

La *médiane* est la valeur centrale d'une distribution triée.

- En cas de quantité impaire de valeurs, on prend la valeur à la position $\frac{n+1}{2}$.
- En cas de quantité paire de valeurs, on prend la moyenne des deux valeurs centrales.

$$\text{médiane} = L_{\frac{|L|}{2}}$$

Par exemple pour la distribution suivante : 4 5 6
La médiane sera de 5, car $\frac{3+1}{2} = 2$ et 5 et le deuxième nombre de la distribution.

Par exemple pour la distribution suivante : 1 2 3 4
La médiane sera de $2,5$, car 2 et 3 sont les valeurs centrales et $\mu(2;3) = 2,5$.

Attention : n'oubliez pas que Python compte depuis 0 pour la position dans les listes.

6.5 Quartiles et Écart interquartiles

6.5.1 Quartiles

L'*écart interquartile* est 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 n'existe pas une seule méthode universellement acceptée... Vous implémenterez donc celle de Wikipédia (en français au 20 août 2023) décrite par la suite : 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 : 10 20 30 40 50 60 70 80 90
- $Q_0 = L_1 = 10$
- $Q_1 = L_3 = 30$
- $Q_2 = L_5 = 50$
- $Q_3 = L_7 = 70$
- $Q_4 = L_9 = 90$

La première partie contient donc les valeurs de 10 20 30, la deuxième partie les valeurs de 30 40 50, la troisième partie les valeurs 50 60 70, et la quatrième partie les valeurs 70 80 90.

Testons maintenant sur la distribution : 10 20 30 40 50 60 70 80
- $Q_0 = L_1 = 10$
- $Q_1 = L_{2,75} =$??
- $Q_2 = L_{4,5} =$??
- $Q_3 = L_{6,25} =$??
- $Q_4 = L_8 = 80$

Si on ne tombe pas sur un entier pour le rang, on effectue la moyenne entre le rang inférieur et le rang supérieur en affectant des poids :

— si le reste après la virgule est à $0,25$: on affecte le poids 3 au rang inférieur, et le poids 1 au rang supérieur

$$Q_X = \frac{3 * L_{n-1} \ + \ 1 * L_{n+1}}{3 \ + \ 1}$$

— si le reste après la virgule est à $0,5$: on affecte le poids 1 au rang inférieur, et le poids 1 au rang supérieur (cela revient à la moyenne des deux)

$$Q_X = \frac{1 * L_{n-1} \ + \ 1 * L_{n+1}}{1 \ + \ 1}$$

— si le reste après la virgule est à $0,75$: on affecte le poids 1 au rang inférieur, et le poids 3 au rang supérieur

$$Q_X = \frac{1 * L_{n-1} \ + \ 3 * L_{n+1}}{1 \ + \ 3}$$

Reprenons la distribution : 10 20 30 40 50 60 70 80

$$\begin{aligned}
Q_0 &= L_1 = 10 \\
Q_1 &= L_{2,75} = \tfrac{1*L_2 \ + \ 3*L_3}{1 \ + \ 3} = \tfrac{1*20+3*30}{1+3} = \tfrac{110}{4} = 27,5 \\
Q_2 &= L_{4,5} = \tfrac{1*L_4 \ + \ 1*L_5}{1 \ + \ 1} = \tfrac{1*40+1*50}{1+1} = \tfrac{90}{2} = 45 \\
Q_3 &= L_{6,25} = \tfrac{3*L_6 \ + \ 1*L_7}{3 \ + \ 1} = \tfrac{3*60+1*70}{3+1} = \tfrac{250}{4} = 62,5 \\
Q_4 &= L_8 = 80
\end{aligned}$$

6.5.2 Écart interquartile

L'*écart interquartile* ou *étendue interquartile* (ou *interquartile range* en anglais et son acronyme *IQR*) est simplement la différence entre le quartile 3 et le quartile 1.

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

6.6 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). Ici, nous implémenterons la variance d'une population complète, et non pas juste d'un échantillon.

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

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.

6.7 Écart type

L'*écart type* (ou *standard deviation* en anglais) 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). Ici, nous implémenterons l'écart type d'une population complète, et non pas juste d'un échantillon.

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

6.8 Coefficient de variation/Écart type relatif

Le *coefficient de variation* ou *écart type relatif* (ou *relative standard deviation* en anglais et son acronyme *RSD*) 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. Ici, nous implémenterons l'écart type relatif d'une population complète, et non pas juste d'un échantillon.

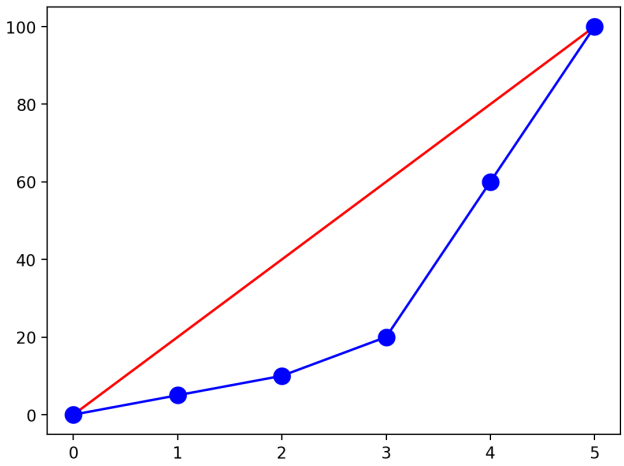
$$\text{coefficient de variation} = c_v = \frac{\sigma}{\mu} = \frac{\sqrt{Var(L)}}{\bar{L}}$$

6.9 Coefficient de Gini

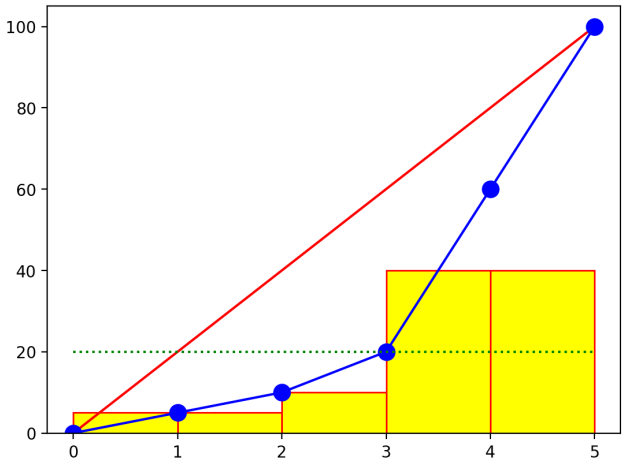
Le *coefficient de Gini* est un indice particulièrement utilisé dans le monde économique et social. Ce coefficient a été constitué pour mesurer les inégalités de revenus au sein d'une population, mais il permet plus généralement de mesurer l'inégalité de répartition d'une variable (donc son interprétation 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.



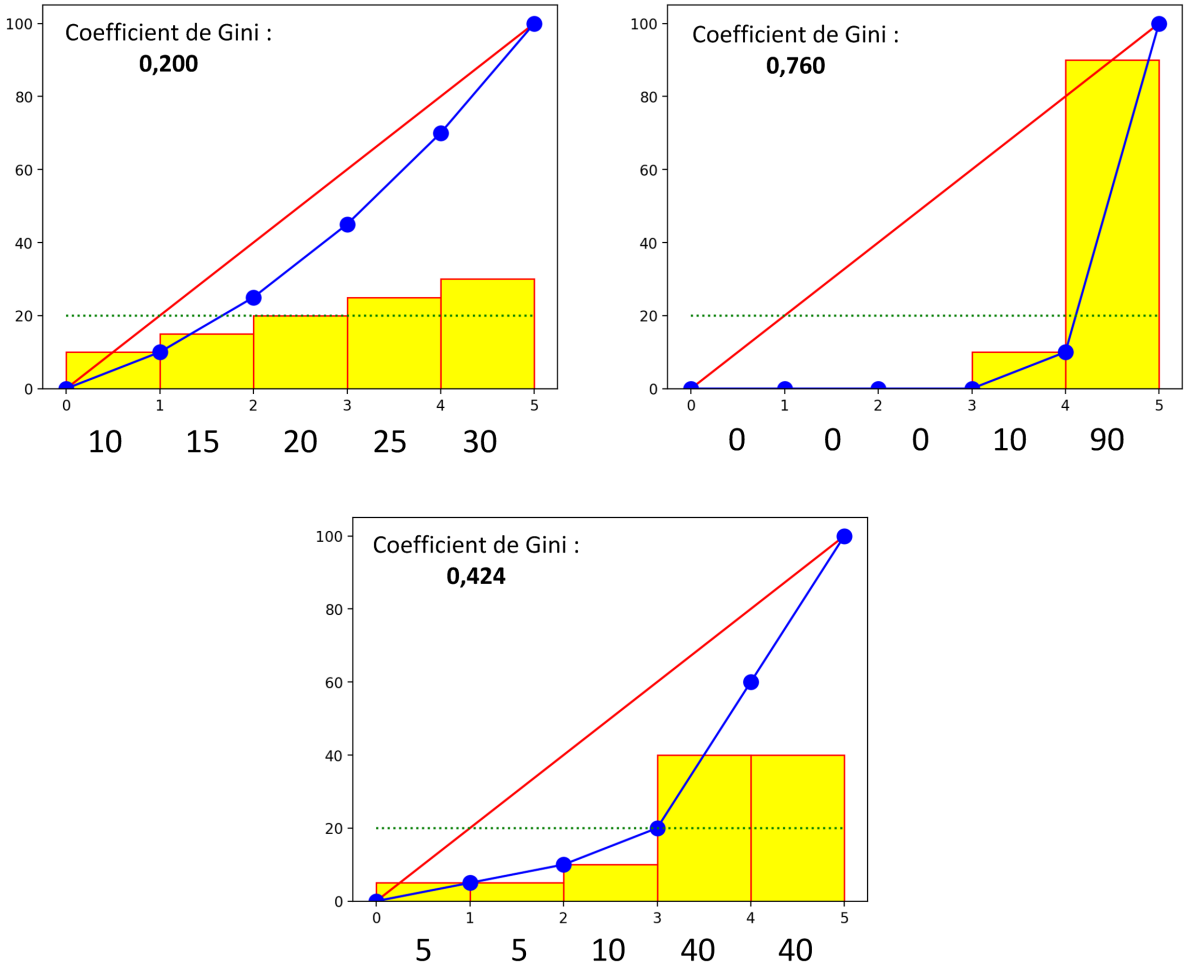
Encore plus visuellement, une fois la distribution ordonnée, on obtient donc 5 5 10 40 40. L'histogramme suivant montre visuellement chaque valeur ajoutée entre les points de la courbe bleue. Donc, en effectuant la somme des valeurs au fur et à mesure, chaque point de la courbe bleue correspond à 5 10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