TP 01

Objectifs :
— import de données : gestion des formats et des sources hétérogènes...
— manipulation de données : filtrage, croisement, modification...
— visualisation basique

Exercice 1 : Une première utilisation de la librairie `Numpy`

La librairie `numpy` est une librairie d'outils mathématiques optimisés qui sert de bases à de très nombreux modules Python, dont ceux que nous manipulerons par la suite. Il convient donc de savoir manipuler les types propres à `numpy` avant d'aller plus loin.

Q1. Créez une celulle uniquement dédiée à l'importation des bibliothèques, et importez-y `numpy`.

Q2. Travaillons d'abord sur une structure d'une seule dimension ! Nous disposons d'une liste contenant les revenus mensuels de vos clients et leurs noms dans une autre. Il faut comprendre que *Ava gagne 1800€* alors que *Bob* gagne 1500€ mensuels. Notez que pratiquement toutes les questions du TPs sont à faire sans écrire la moindre boucle (sauf mention explicite) !

```
liste_revenus = [1800, 1500, 2200, 3000, 2172, 5000, 1400, 1200, 1100, 1300]
liste_noms = ['Ava', 'Bob', 'Cao', 'Dan', 'Eli', 'Fay', 'Guy', 'Hue', 'Ian', 'Joy']
```

Q2.1. Créez un *array* nommé `revenus` à partir de `liste_revenus` et affichez son contenu. Faites de même pour les noms.

Q2.2. Créez un *array* `hauts_revenus` contenant uniquement les revenus supérieurs à 2000€, via une indexation booléenne ou via la fonction `where(...)` de `numpy`. Affichez son contenu.

Q2.3. Affichez les noms des personnes gagnant plus de 2000€ mensuels, à partir de la fonction `where(...)`.

Q2.4. Calculez la moyenne des revenus de tous vos clients, via la méthode `mean()`. Affichez en arrondissant grâce à une *f-string*.

Q2.5. Calculez et affichez en arrondissant (en K€) la somme des revenus **annuels** de tous vos clients.

Q2.6. Le client gagnant 1100€ a reçu une augmentation de 150€. De même, *Ava* a été augmenté de 100€. Répercutez ces hausses dans votre structure. Veillez à n'utiliser que les informations à votre disposition pour effectuer chacune des opérations : à partir de la valeur elle-même et l'indexation booléenne d'un côté, à partir du nom à associer avec la fonction `where(...)` de l'autre.

Q2.7. Affichez les noms et revenus de chacun des clients à raison d'un client par ligne, au moyen d'une boucle.

Q3. Travaillons désormais en deux dimensions. Nous disposons toujours d'une liste de clients qui incluent leur revenu, leur âge et leur nombre d'enfants à charge. À chaque ligne correspond un client : `liste_clients` contient ses informations tandis que `liste_noms` contient son nom à l'indice correspondant.

```
liste_clients=np.array([
    [1800, 21, 0],
    [1500, 54, 2],
    [2200, 28, 3],
    [3000, 37, 1],
    [2172, 37, 2],
    [5000, 32, 0],
    [1400, 23, 0],
    [1200, 25, 1],
    [1100, 19, 0],
    [1300, 31, 2]
])
liste_noms = ['Ava', 'Bob', 'Cao', 'Dan', 'Eli', 'Fay', 'Guy', 'Hue', 'Ian', 'Joy']
```

Q3.1. Cao souhaiterait contracter un prêt immobilier. Affichez les informations sur Cao, via la fonction `index(...)`. Calculez ses mensualités maximales possibles, en sachant que le taux d'endettement maximum est de 35% (il ne pourra donc pas rembourser plus de 35% de son revenu par mois).

Q3.2. Kim est une nouvelle cliente qui vient d'arriver : `kim = [1900, 31, 1]`. Ajoutez ces informations à vos données via la fonction `append(...)` ou `vstack(...)`. Affichez à nouveau les informations (et vérifiez les dimensions du tableau obtenu).

Q3.3. Stockez enfin l'ensemble des informations sur les âges de notre clientèle, correspondant à la colonne d'indice 1, dans une variable `clients_ages` et affichez-la. Les données sont-elles encore sous forme de colonne ?

Exercice 2 : Génération, manipulation et extraction

`numpy` contient beaucoup de fonctions et d'opérateurs dédiés à des opérations spécifiques, qui vous permettront de faire face à un grand nombre de situation.

Q1. Intéressons-nous dans un premier temps à la génération de données et leur visualisation.

Q1.1. Générez un premier vecteur `v1` contenant la séquence des 10 premiers nombres impairs via la fonction `arange(...)`.

Q1.2. Générez un second vecteur `v2` de 100 nombres entiers générés aléatoirement dont les éléments sont compris entre 25 et 70, via le générateur `integers(...)`.

On rappelle qu'il convient d'instancier un générateur aléatoire au préalable, sur lequel on invoque les distributions à utiliser :

```
# définition du générateur aléatoire
gen = np.random.default_rng()
```

Q1.3. On souhaite visualiser les nombres qu'on vient de générer et connaître leur répartition. En utilisant le module `matplotlib`, tracez un histogramme via la fonction `hist(...)` et l'option `bins=10` pour connaître la répartition par dizaine. Ajoutez un titre à votre graphique.

Q1.4. Augmentez la valeur de l'option `bins` pour constater son effet.

Q2. Manipulons et transformons ces vecteurs.

Q2.1. Le vecteur `v2` est trop long. Ne conservez que ses 10 premiers éléments.

Q2.2. Fusionnez les vecteurs `v1` et `v2` grâce à la fonction `concatenate(...)`. Affichez le contenu et la taille de ce nouveau vecteur.

Q2.3. Mélangez le contenu du vecteur `v1v2` via la méthode `shuffle(...)` qui s'invoque sur un générateur aléatoire. Affichez ensuite son contenu.

Q2.4. Déterminez les positions des éléments minimum et maximum du vecteur `v1v2` via les fonctions `argmin(...)` et `argmax(...)`.

Notez que ces fonctions bénéficient d'une option `axis` afin d'être appliquées directement sur des tableaux à plusieurs dimensions.

Q2.5. On souhaite transformer ce vecteur en une matrice de 4 éléments par ligne, via la fonction `reshape(...)`. Notez qu'il faut un tuple décrivant les dimensions de découpage, et qu'une dimension de -1 est recalculée en fonction des autres dimensions fournies.

Q2.6. Créez une matrice `v1v_row`, composée de deux lignes correspondant respectivement aux vecteurs `v1` et `v2`, via la fonction `stack(...)`. C'est une fusion de type *row-wise*.

Q2.7. Créez une autre matrice `v1v2_column`, composée de deux colonnes correspondant respectivement aux vecteurs `v1` et `v2` : soit par transposition de `v1v2_row` et l'opérateur `T`, soit à partir de la fonction `stack(...)` en spécifiant l'option `axis`. C'est une fusion de type *column-wise*.

Q2.8. Ajoutez à `v1v2_column` un index : une nouvelle première colonne dont les valeurs correspondant aux numéros d'ordre des lignes, grâce à la fonction `column_stack(...)`.

Q3. Intéressons-nous à la matrice suivante :

```
A = np.array([
    [1, 2, 3],
    [1, 2, 3],
    [4, 5, 6],
    [1, 1, 1],
    [7, 8, 9],
    [1, 2, 3],
    [10, 10, 10],
    [1, 1, 1],
    [14, 15, 16],
    [1, 2, 3]
])
```

Q3.1. Écrivez le code nécessaire à l'extraction des lignes différentes de la première, grâce à la fonction `all(...)`, son option `axis` et l'opérateur de négation $\approx$.

Q3.2. Extrayez toutes les lignes uniques (sans doublons), via la fonction `unique(...) et son option axis`.

Q3.3. Générez une matrice B binaire de même dimension que A, en invoquant la méthode `integers(...)` sur le générateur aléatoire, en sachant que le paramètre correspondant à la taille peut être un tuple. Croisez ces deux matrices pour obtenir une matrice C contenant A[i,j] si B[i,j]>0, -A[i,j] sinon, en utilisant la fonction `where(...)`.

Q3.4. Échangez la première et la dernière ligne de la matrice C.

Q3.5. Triez la matrice C selon la 2ème colonne grâce à la fonction `argsort(...)`.

Q3.6. Soustrayez la moyenne de chaque ligne à ces mêmes lignes grâce à la fonction `mean(...)` et ses options `axis` et `keepdims`.

Exercice 3 : Rappels de statistiques S3

On rappelle que la fonction de densité d'une loi normale d'espérance μ et d'écart-type σ, est notée $\mathcal{N}(\mu, \sigma)$, s'exprime par :

$$f : x \mapsto \frac{1}{\sigma\sqrt{2\,\pi}}\, e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Graphiquement, la probabilité d'un évènement X, notée $P[a < X < b]$, telle que $-\infty \leq a \leq b \leq +\infty$ correspond à l'aire de la zone comprise :

— en abscisses, entre a et b (autrement dit : $a < x < b$)

— en ordonnées, entre la l'axe des abscisses et la courbe représentative de la densité (autrement dit : $0 < y < f(x)$)

Q1. Créez une cellule d'import des bibliothèques nécessaires, à savoir `numpy` et `matplotlib.pyplot`.

Q2. Commençons par visualiser une loi normale.

Q2.1. Définissez une fonction `densité`, prenant en paramètre l'espérance μ et l'écart-type σ, qui calcule la valeur associée à une variable `x` prédéfinie.

```
def densité(mu, sigma): ...
```

Q2.2. Générez un vecteur x de 1000 valeurs entre 0 et 80, régulièrement espacées, via la fonction `linspace(...)`. Ces valeurs serviront à calculer `densité(x)`, et tracer la loi normale correspondante.

Q2.3. Tracez la courbe de la fonction de la densité de la loi normale, ici $\mathcal{N}(35, 10)$, grâce à la fonction `plot(...)`. Veillez à ajouter un titre et une légende.

Q3. Comprenons maintenant une loi normale.

Q3.1. Tracez $\mathcal{N}(35, 10)$, $\mathcal{N}(35, 20)$ et $\mathcal{N}(35, 5)$ sur le même graphe. Ajoutez une légende en haut à droite et un titre à votre courbe. Constatez l'impact de l'écart-type sur la loi normale.

Q3.2. De même, tracez $\mathcal{N}(35, 10)$, $\mathcal{N}(45, 10)$ et $\mathcal{N}(55, 10)$ sur le même graphe. Constatez l'impact de l'espérance sur la loi normale.

Q3.3. À partir du tracé de la densité, estimez la valeur des probabilités suivantes, où X est une variable aléatoire de loi $\mathcal{N}(35, 10)$:

— $P[X < 35]$
— $P[X < 45]$
— $P[35 < X < 45]$
— $P[X > 45]$

Q4. Nous allons maintenant simuler et représenter des échantillons pour la loi normale $\mathcal{N}(35, 10)$ pour comprendre l'impact des différents paramètres.

Q4.1. Générez un vecteur nommé `ages100` contenant 100 éléments, qui vérifie cette loi statistique. Il convient d'invoquer la distribution `.normal(...)` sur un générateur aléatoire.

Q4.2. Tracez un histogramme (via `hist(..., density=True, alpha=0.25)` représentant l'échantillon. Vérifiez sa forme en superposant le graphe de la densité de la loi qu'il est censé suivre (toujours sur 1000 points entre 0 et 80).

Q4.3. Afin de constater l'impact de la taille de l'échantillon, générez des vecteurs nommés `ages10`, `ages10K` et `ages1M` contenant respectivement 10, 100000 et 1e6 éléments, qui vérifient cette loi statistique.

Q4.4. Tracez les différents histogrammes sur le même graphe ainsi que la loi de densité pour observer l'impact de la taille de l'échantillon.

Q4.5. Reste à évaluer l'importance de finesse de l'histogramme : le paramètre `bins`. Pour l'échantillon `ages10K`, tracez les histogrammes pour la loi normale $\mathcal{N}(35, 10)$ obtenus pour des valeurs de `bins` de 10, 100 et 1000. Constatez l'effet ainsi obtenu.

Exercice 4 : Création d'un *mock*

`numpy` propose un ensemble de méthodes permettant la génération aléatoire de données respectant certaines spécifications données. Ces méthodes peuvent être en particulier utile pour la création de *mock*, c'est-à-dire d'ensemble de données vérifiant certaines caractéristiques.

Q1. Intéressons-nous à la génération de choix (valeurs prédéfinies qualitatives ou quantitatives) respectant des proportions données.

Q1.1. Grâce à la méthode `choice(...)` d'un générateur aléatoire, générez un échantillon de 100 données représentant des hommes (valeur '1'), des femmes (valeur '2') et des personnes masquant cette information (valeur '3'). Les proportions à respecter sont respectivement 77% d'hommes, 20% de femmes et 3% de valeurs masquées.

Q1.2. Dénombrez les occurences de chacune des valeurs, via la fonction `count_nonzero(...)`.

Q1.3. Dessinez un diagramme à barres (`bar(...)`) permettant de vérifier le bon respect des proportions indiquées lors de la génération. Veillez à ajouter un titre et à adapter les graduations des abscisses aux labels '`M`', '`F`' et '`?`' grâce à la fonction `xticks(...)`.

Q1.4. Personnalisons un peu plus votre graphique. Ajoutez les nombres exacts au sommet des colonnes correspondantes, en utilisant `plt.text(...)`.

Q2. On souhaite désormais compléter nos données de genre par des âges (afin d'obtenir une matrice dont la première colonne représente les genres et la seconde les âges).

Q2.1. Générez un vecteur nommé `ages` de 100 éléments, qui simule une loi statistique normale $\mathcal{N}(35, 10)$.

Q2.2. Tracez un histogramme représentant la nouvelle donnée et vérifiez sa forme. Ajoutez-y un titre comme précédemment.

Q2.3. Ajoutez à votre histogramme le tracé de la loi normale qu'il est censé suivre.

Q2.4. Ajoutez à vos données de genre cette nouvelle distribution sur les âges, via la fonction `column_stack(...)`.

Q3. On souhaite ajouter une nouvelle information, à savoir le nombre d'amendes qu'ils ont déjà reçu. Par défaut, nous considérons qu'ils commenceront tous comme "vierge" de toute amende.

Q3.1. Ajoutez une nouvelle colonne à vos données, où tous les éléments valent 0. Utilisez la fonction `zeros(...)` pour générer les données.

Q3.2. Plutôt qu'une génération aléatoire suivant une distribution de probabilité, on souhaite que exactement 10 aient reçus une amende, 20 autres deux amendes et un seul dernier 5 amendes. Effectuez les instructions nécessaires et mélangez les données une fois l'opération réalisée.

Q3.3. Générez le graphique approprié illustrant votre bonne génération des données.

Q4. On souhaite finalement ajouter des données sur les salaires dans une nouvelle et dernière colonne. On sait que les salaires vérifient une loi statistique de *Laplace* : $X \sim Laplace(\mu, \lambda)$, qui est caractérisée par une moyenne μ et un paramètre de diversité λ. Elle s'exprime à l'aide de la densité suivante :

$$f(x|\mu, \lambda) = \frac{1}{2\lambda} exp\left(-\frac{|x-\mu|}{\lambda}\right)$$

Q4.1. Générez un tableau pour compléter vos données selon une distribution de *Laplace*, $Laplace(1500, 100)$, via la fonction de même nom, invocable sur un générateur aléatoire.

Q4.2. Générez le graphique approprié illustrant votre bonne génération des données. Ajoutez-y la distribution que vos données sont sensées suivre.

Q4.3. N'oubliez pas d'ajouter ces données à votre tableau `data`. Vous devriez maintenant avoir 4 colonnes : une pour le genre, une pour l'âge, une pour le nombre d'amendes et une dernière pour le salaire.