Chapitre # (NUM) 5 Statistiques

Résumé & Plan

L'objectif de ce TP est d'appliquer les scripts Python vus en cours dans des exercices, il servira également de TD associé au cours de Statistiques. On termine par une analyse statistique d'un grand jeu de données, en commençant par la phase d'importation (déjà réalisée dans le Chapitre (ALGO) 5 – Arbres de la ville de Bordeaux).

Cadre

On suppose dans tout le TP qu'une série statistique est représentée par une liste comportant l'ensemble des observations.

Fichiers externes et mise en place

1. Avant de traiter les parties informatiques de ce TP, commencez par récupérer le fichier `Fonctions_TPStats.py` sur le site de la classe qui contient toutes les fonctions utiles, ainsi que le fichier `TPStats.py`.
2. Placez ces fichiers dans le même dossier. Au début de `TPStats.py` vous trouverez une ligne qui importe toutes les fonctions de `Fonctions_TPStats.py`, exécutez une fois ce code **en tant que script**.
3. Vous travaillerez ensuite directement dans le fichier `TPStats.py`.

1. STATISTIQUES UNIVARIÉES

1.1. Rappel des principales fonctions

Pour avoir la liste des modalités, il suffit de créer une nouvelle liste sans doublon.

(Modalités)

```
def sans_doublon(L):
    """
    renvoie la liste des éléments de L, chaque élément \
    ↪ apparaissant une unique fois
    """
    M = []
    for x in L:
        if x not in M:
            M.append(x)
    return M
```

On peut également transformer la série statistique de départ en dictionnaire de clefs les modalités, et valeurs l'effectif associé à chaque modalité. C'est la fonction `dico_occur` déjà rencontrée.

(Dictionnaire des effectifs)

```
def dico_occur(L):
    D = {}
    for x in L:
        if x not in D:
            D[x] = 1
        else :
            D[x] += 1
    return D
```

On peut également revenir à une liste d'observations si on le souhaite.

(Dictionnaire des effectifs vers liste)

```
def dico_occur_vers_liste(D):
    L = []
    for x in D:
        # x est une modalité, que l'on duplique autant de \
        ↪ fois que nécessaire
        eff_x = D[x]
        for _ in range(eff_x):
            L.append(x)
    return L
```

La fonction de calcul de moyenne s'appuie notamment sur celle qui calcule la somme.

(Moyenne)

```
def moyenne(L):
    """
    Renvoie la moyenne des éléments d'une liste
    """
    S = 0
    for x in L:
        S += x
    return S/len(L)
```

Si l'on préfère, on peut aussi calculer directement la moyenne à l'aide du dictionnaire des effectifs : dans ce cas, on pondère par l'effectif associé.

(Moyenne avec effectifs)

```
def moyenne_avec_eff(D):
    """
    Renvoie la moyenne d'une série associée au dictionnaire \
    ↪ des effectifs
    D
    """
    S = 0
    N = 0 # nombre d'éléments de la série
    for x in D:
        eff_x = D[x]
        S += x*eff_x
        N += eff_x
    return S/N
```

Pour la variance, on utilise généralement la version König-Huygens de la formule : $\mathbb{V}_x = \overline{x^2} - \overline{x}^2$ si x désigne une série statistique.

(Variance)

```
def variance(L):
    """
    Renvoie la variance, version KH
    """
    S2 = 0
    for x in L:
        S2 += x**2
    return S2/len(L) - moyenne(L)**2
```

Voyons quelques exemples d'exécutions.

Exemple 1 On code la série

19	26	23	20	22	24	20	24
22	20	21	19	21	22	19	20
21	21	22	21	23	22	21	24
25	23	22	19	20	26	24	25
23	26	25	25	21	22	25	24
23	22	24	24	25	23	25	22

avec la liste :

```
>>> L = [19, 26, 23, 20, 22, 24, 20, 24, 22, 20, 21, 19, 21, \
↪ 22, 19, 20, 21, 21, 22, 21, 23, 22, 21, 24, 25, 23, 22, 19, \
↪ 20, 26, 24, 25, 23, 26, 25, 25, 21, 22, 25, 24, 23, 22, 24, \
↪ 24, 25, 23, 25, 22]
>>> modalites(L)
[19, 26, 23, 20, 22, 24, 21, 25]
>>> D = dico_occur(L)
>>> D
{19: 4, 26: 3, 23: 6, 20: 5, 22: 9, 24: 7, 21: 7, 25: 7}
>>> dico_occur_vers_liste(D)
```

```
[19, 19, 19, 19, 26, 26, 26, 23, 23, 23, 23, 23, 23, 20, 20, \
↪ 20, 20, 20, 22, 22, 22, 22, 22, 22, 22, 22, 22, 24, 24, 24, \
↪ 24, 24, 24, 24, 21, 21, 21, 21, 21, 21, 21, 25, 25, 25, 25, \
↪ 25, 25, 25]
>>> moyenne(L)
22.5
>>> variance(L)
4.083333333333314
>>> moyenne_avec_eff(D) # on retrouve bien le même résultat
22.5
```

On peut également, après recherche du minimum et du maximum, renvoyer l'étendue de la série.

(Étendue d'une série)

```
def etendue(L):
    """
    Renvoie l'étendue de la série statistique des éléments de L
    """
    mini = L[0]
    maxi = L[0]
    for x in L[1:]:
        if x < mini:
            mini = x
        elif x > maxi:
            maxi = x
    return maxi - mini
```

Pour calculer la médiane, il faut au préalable trier la liste.

(Médiane)

```
def mediane(L):
    """
    Cherche la médiane d'une liste, après tri rapide des \
    ↪ observations
    """
    L_tri = tri_rapide_rec(L)
    n = len(L)
    if n % 2 == 1:
        # Nombre impair d'observations
        return L_tri[n//2]
    else:
        # Nombre pair d'observations
        return (L_tri[n//2-1] + L_tri[n//2])/2
```

On peut ensuite tester si la quantité retournée est bien une médiane, en contrôlant la définition : au moins la moitié des observations sont supérieures ou égales à la médiane, et au moins la moitié des observations sont inférieures ou égales à la médiane.

```
def mediane_verif(L, m):
    """
    Renvoie True si m est bien une médiane de L
    """
    nb_inf = 0
    nb_sup = 0
    for x in L:
        if x >= m:
            nb_sup += 1
        if x <= m:
            nb_inf += 1
    return nb_sup >= len(L)/2 and nb_inf >= len(L)/2
```

Exemple 2 **(Diamètres de pièces)**

```
>>> etendue(L)
7
>>> m = mediane(L)
>>> mediane_verif(L, m)
True
```

Plus généralement, voici comment calculer les 3 quartiles.

(Quartiles)

```
def quartiles(L):
    """
    renvoie Q1, Q2, Q3, après tri rapide des observations
    """
    L_tri = tri_rapide_rec(L)
    n = len(L)
    if n % 2 != 0:
        # Nombre impair d'observations
        Q2 = L_tri[n//2]
```

```
    else:
        # Nombre pair d'observations
        Q2 = (L_tri[n//2-1] + L_tri[n//2])/2
    if n % 4 != 0:
        # Nombre non multiple de 4 d'observations
        Q1 = L_tri[n//4]
        Q3 = L_tri[(3*n)//4]
    else:
        # Nombre multiple de 4 d'observations
        Q1 = L_tri[n//4-1]
        Q3 = L_tri[(3*n)//4-1]
    return Q1, Q2, Q3
```

1.2. Exercices

(Histogramme et diagramme en bâtons) Les deux commandes principales (rappelées en cas de besoin) sont les suivantes :

- `plt.bar(X, H)` pour les diagrammes en bâtons : `X` désigne la liste des modalités, et `H` la liste des hauteurs associée (effectifs ou modalités).
- `plt.hist(L, n)` pour les histogrammes : `L` désigne la liste des observations, et `n` le nombre de classes souhaitées. Le regroupement en classes des données est alors fait tout seul par la commande.

Exercice 1 | Effet d'un médicament Solution Un médecin effectue des recherches sur l'efficacité d'un nouveau béta-bloquant. Cette famille de médicaments est destinée à diminuer le rythme cardiaque des malades atteints de tachycardie (pouls supérieur à 60 battements par minute). Il a donc séparé les malades en 2 groupes : le groupe A reçoit le traitement d'un nouveau médicament, et le groupe B reçoit un placebo. Voici les résultats :

- A : 74 – 91 – 91 – 84 – 95 – 93 – 95 – 102 – 81 – 116 – 88 – 95
- B : 94 – 95 – 113 – 95 – 104 – 113 – 94 – 144 – 105 – 153

1. Calculer à la main la médiane des deux séries.

2. Coder ces deux séries à l'aide de deux listes. En déduire pour chaque série : l'étendue, la médiane, les valeurs extrêmes, le premier quartile et le troisième quartile. Complétez le tableau ci-dessous.

Étendue	Médiane	Min	Max	Q_1	Q_3

3. Construire le diagramme de Tuckey de ces deux séries. L'effet du médicament semble-t-il satisfaisant ?

4. Construire l'histogramme de chaque médicament, par exemple pour `n = 10` classes.

Exercice 2 | Solution Une entreprise souhaite étudier la consommation de cigarettes chez ses salariés qui fument.

Nb cigarettes par jour	5	10	15	20	25	30	35	40
Effectifs	8	17	12	8	5	3	2	1

1. Coder ce tableau à l'aide d'un dictionnaire des effectifs.
2. En déduire la médiane, la moyenne, l'étendue. *On pourra donc, si nécessaire, revenir à une liste complète d'observations plutôt qu'un dictionnaire au moyen de* `dico_occur_vers_liste`.

Médiane	Moyenne	Étendue

3. On souhaite ici calculer le mode de la série, *i.e.* la modalité d'effectif maximal.
   - **3.1)** Écrire une fonction `mode(L)` qui contient une série statistique sous forme de liste et qui retourne son mode. *On pourra utiliser le dictionnaire des effectifs.*
   - **3.2)** En déduire le mode de la série étudiée.

2. STATISTIQUES BIVARIÉES

2.1. Rappel des principales fonctions

En utilisant directement les définitions de chaque quantité, on en déduit les fonctions associées ci-dessous.

(Covariance)

```
def covariance(L, M):
    """
    Renvoie la covariance des deux séries
    """
    Prod = [L[i]*M[i] for i in range(len(M))]
    return moyenne(Prod) - moyenne(L)*moyenne(M)
```

(Coefficient de corrélation)

```
def coeff_cor(X, Y):
    """
    Renvoie le coefficient de corrélation des deux séries
    """
    return covariance(X, \
    ↪ Y)/(ma.sqrt(variance(X))*ma.sqrt(variance(Y)))
```

(Coefficients de régression)

```
def regression_lin(X, Y):
    """
    renvoie les coefficients a, b de régression  linéaire \
    ↪ associée au
    nuage de points (X, Y)
    """
    a = covariance(X, Y)/variance(X)
    b = moyenne(Y)-a*moyenne(X)
    return a, b
```

Exemple 3 (Relation entre le poids et la taille) Regardons ce que donne cette fonction sur les séries statistiques ci-après, la série X correspondant à des relevés de poids, et Y de taille.

```
>>> X = [150, 170, 162, 164, 172, 164, 167, 175, 176, 181]
>>> Y = [56, 56, 50, 62, 71, 54, 52, 62, 65, 65]
>>> a, b = regression_lin(X, Y)
>>> a
0.4485537487408167
>>> b # on retrouve bien les bonnes valeurs
-16.10188516333129
>>> coeff_cor(X, Y)**2
0.3442851600160366
```

Le coefficient de détermination est très inférieur à 0.9, la régression est donc très mauvaise (ce que l'on pouvait constater graphiquement).

2.2. Exercices

Exercice 3 | Distance et freinage Solution Le tableau suivant donne la distance de freinage d'un véhicule roulant sur route sèche en fonction de sa vitesse.

Vitesse en km/h	40	50	60	70	80	90	100	110
Distance en m	8	12	18	24	32	40	48	58

1. Calculer le coefficient de corrélation de cette série double.
2. Déterminer l'équation de la droite d'ajustement affine de la distance en fonction de la vitesse.

3. Estimer la distance de freinage d'un véhicule roulant à 120 km/h.

Exercice 4 | Solution Dans un laboratoire de biologie, on mesure la taille d'un échantillon de poissons d'âges différents soumis à l'administration d'une substance chimique (A) susceptible d'agir sur leur croissance. Les temps (en mois) sont représentés par la série x les longueurs (en mm) sont représentées par la série y. Les résultats obtenus sont les suivants :

$$\sum_{i=1}^{n} x_i = 125, \quad \sum_{i=1}^{n} x_i^2 = 295, \quad \sum_{i=1}^{n} y_i = 1160, \quad \sum_{i=1}^{n} x_i y_i = 2730, \quad n = 70.$$

Cet exercice est à traiter sans utiliser les fonctions Python associées et entièrement sur feuille. On peut néanmoins se servir de la console en tant que calculatrice.

1. Donner la meilleure estimation possible de la taille de ces poissons à 4 mois sachant que la croissance continue de manière linéaire.

2. Une autre substance chimique (B) a été administrée à une série de poissons de même race. L'équation de la droite de régression est alors : $y' = 0,403 + 7,331x$. Quelle substance semble le plus favoriser la croissance ?

Dans la suite, nous aurons parfois besoin de tracer un nuage de points et la droite de régression associée, on poursuit donc avec un exercice permettant d'effectuer ce tracé.

Exercice 5 | Tracer un nuage et la droite de régression Solution >_ Écrire une fonction `nuage` prenant en argument deux séries statistiques `L1` et `L2` de même taille et qui affiche le nuage de points pour lequel les abscisses sont les valeurs de la liste `L1` et les ordonnées correspondantes les valeurs de la liste `L2`. Le graphique devra également faire figurer le point moyen (en une autre couleur). *Pour le nuage de points, on utilisera l'option* `"bo", markersize=1`, *et pour le point moyen* `"ro", markersize=10`.

Dans la suite, on pourra utiliser la fonction ci-après, présente dans le fichier `Fonctions_TPStats.py`, qui en plus du nuage de points trace la droite de régression associée, et renvoie les coefficients de la droite, ainsi que le coefficient de détermination.

```
def nuage_reg(L1, L2):
    nuage(L1, L2)
    # droite de régression :
    a, b = regression_lin(L1, L2)
    r = coeff_cor(L1, L2)**2
    plt.plot(L1, [a*x+b for x in L1], "g", label="r =
"+str(round(r,2)))
    plt.legend()
    plt.show()
    return a, b, coeff_cor(L1, L2)**2
```

Exercice 6 | Pression et altitude Solution Le tableau ci-dessous indique la pression (en hPa) en fonction de l'altitude (en km) :

Altitude x	0	1	2	3	4	5	6	7	8
Pression y	1010	896	792	699	614	538	470	409	355
Altitude x	9	10	12	15	20	25	30	35	40
Pression y	306	264	194	121	55	26	12	6	3

1. >_ Le nuage de points associé à (x, y) suggère-t-il qu'un ajustement affine est adapté ?
2. >_ On pose $z = \ln(y)$. Visualiser le nuage de points associé à la série double (x, z). Un ajustement affine semble-t-il adapté ?
3. >_ Obtenir la droite de régression de z en x. En déduire une approximation de la pression à une altitude de 50 km.

Exercice 7 | Loi SPAR Solution Plus une région est vaste, plus le nombre d'espèces y vivant est grand. Pour modéliser mathématiquement ce phénomène (et mesurer la biodiversité), les scientifiques utilisent régulièrement la loi SPAR (« species-area relationship »). Elle stipule que si S représente la surface de la région étudiée et n le nombre d'espèces présentes dans cette région, alors $n = Cs^Z$, où C et Z sont des constantes à ajuster selon la région étudiée.

On demande à des élèves d'une classe de BCPST de vérifier cette loi pour les plantes présentes dans une prairie. Les données récoltées sont résumées dans le tableau suivant, exprimant le nombre d'espèces différentes présentes n en fonction de la surface s de la prairie étudiée :

Surface s en m^2	1	2	4	8	16	32	64
Nombre d'espèces n	6	7	8	10	10	13	14

Afin de mettre en valeur la relation donnée par la loi SPAR, on décide de procéder à une régression linéaire de $\ln n$ sur $\ln s$.

1. Justifier le choix de cette régression linéaire.

2. >_ Tracer le nuage de points correspondant aux variables $\ln s$ (en abscisses) et $\ln n$, ainsi que la droite de régression linéaire.
3. >_ Combien vaut le coefficient de corrélation linéaire ? Que peut-on en déduire ?
4. >_ Donner alors une approximation de C et Z.

Exercice 8 | Modèle de MICHAELIS-MENTEN Solution On mesure la vitesse d'une réaction biologique en fonction de la concentration d'une enzyme :

Concentration C (en mol/L)	0,0052	0,0104	0,0208	0,0416	0,0833	0,1617	0,3330
Vitesse v (en µmol/L/min))	0,866	1,466	2,114	2,666	3,236	3,636	3,636

1. >_ Visualiser le nuage de points associé à la série double (C, v). Un ajustement affine semble-t-il adapté ?

2. La courbe laisse penser à une relation du type $v = \dfrac{k_1 \times C}{k_2 + C}$: c'est la relation de MICHAELIS-MENTEN pour décrire la cinétique d'une réaction catalysée par une enzyme agissant sur un substrat.

2.1) Exprimer avec cette relation $\dfrac{1}{v}$ en fonction de $\dfrac{1}{C}$. Pour tester cette relation, quelle nouvelle série double est-il pertinent d'introduire ?

2.2) >_ Représenter le nuage de points de la nouvelle série double et calculer leur coefficient de corrélation. Qu'en déduire ?

2.3) >_ Estimer les réels k_1 et k_2 de l'équation de MICHAELIS-MENTEN.

3. ANALYSE STATISTIQUE D'UNE BASE DE DONNÉES

On souhaite dans cette partie mettre en application les différents scripts afin de traiter statistiquement un grand jeu de données, que nous avions déjà croisé dans un précédent chapitre (les arbres plantés à Bordeaux) : il s'agit d'un fichier CSV (**C**omma-**S**eparated **V**alues) que nous allons déjà devoir importer dans Python.

3.1. Rappels sur l'importation d'un csv

Le module `csv` contient des outils permettant de lire et manipuler des fichiers dont le format est `CSV`. On l'importe de la manière classique suivante.

```
>>> import csv
```

Télécharger ensuite le fichier `bor_arbres.csv` depuis le site de la classe. Ensuite, déposer le fichier de données dans le même répertoire que le fichier Python principal, puis exécuter au moins une fois **en tant que script**. On écrit ensuite dans l'éditeur :

```
>>> fichier = open('bor_arbres.csv',"r")
>>> data = csv.reader(fichier, delimiter=";")
```

On constate que `data` possède un type particulier.

```
>>> type(data)
<class '_csv.reader'>
```

Mais c'est un type itérable, au sens où l'on peut parcourir chacune des lignes à l'aide d'une simple boucle `for`. On peut donc stocker l'ensemble des données dans une grosse liste `DATA`, que l'on utilisera alors dans toute la suite. On prend garde à garder uniquement les arbres où les champs qui nous intéressent sont renseignés (colonnes 9, 12, 13 dans la suite).

```
DATA = []
for arbre in data:
    if len(arbre[9]) != 0 and len(arbre[12]) != 0 and \
    ↪ len(arbre[13]) != 0:
        # on ne garde que les arbres où les champs 9/12/13 sont \
        ↪ renseignés
        DATA.append(arbre)
```

On peut par exemple en afficher une petite partie.

```
>>> DATA[0:3]
```

```
[['Geo Point', 'Geo Shape', 'gid', 'geom_o', 'geom_err', 'localisa
tion', 'tranche_age', 'nom', 'circonference', 'hauteur', 'typo_esp
ace', 'statut', 'diametre', 'famille', 'genre', 'variete', 'geogra
phie', 'date_plantation', 'cdate', 'mdate', 'age_tranche_basse
'], ['44.8451436,-0.5719386', '{"type": "Point", "coordinates": [-
0.5719386, 44.8451436]}', '3034', '0', '', 'des Quinconces (place)
', '49 - 74', 'Platanus x hispanica', '147', '20', 'ESPACE_PUBLIC_
VEGETALISE', 'VIVANT', '47', 'Platanaceae', 'Platanus sp.', '', 'H
ybride Platanus orientalis x P. occidentalis', '', '2019-04-02T00:
00:00+02:00', '2019-04-02T00:00:00+02:00', '49'], ['44.8451677,-0.
5716563', '{"type": "Point", "coordinates": [-0.5716563, 44.845167
7]}', '3036', '0', '', 'des Quinconces (place)', '83 - 124', 'Plat
anus x hispanica', '248', '21', 'ESPACE_PUBLIC_VEGETALISE', 'VIVAN
T', '79', 'Platanaceae', 'Platanus sp.', '', 'Hybride Platanus ori
entalis x P. occidentalis', '', '2021-05-20T00:00:00+02:00', '2021
-05-20T00:00:00+02:00', '83']]
```

On enlève ensuite la ligne d'en-tête.

```
>>> del DATA[0] # suppression de la ligne d'en-tête
```

Une fois nos données récupérées, on ferme le fichier.

```
>>> fichier.close()
```

> **Remarque 1** En cas de besoin, il est possible d'ouvrir le fichier avec un tableur afin de mieux visualiser les données.

Dans toute la suite, les fonctions du début du TP pourront être utilisées librement pour répondre aux questions suivantes. La fonction médiane utilise un tri, par exemple le tri rapide. Vous vous rendrez compte, puisque les tailles des listes sont très importantes ici, que on l'on obtient une erreur de type « nombre maximal d'appels récursifs dépassé » lors d'une exécution du tri rapide. Afin d'éviter cette erreur, on indiquera en début d'éditeur :

```
>>> import sys
>>> sys.setrecursionlimit(10**5) # modification de la taille de \
↪ la pile d'appels récursifs
```

3.2. Statistiques univariées

Exercice 9 | Caractéristiques de la hauteur et du diamètre Solution

1. Que crée cette instruction ?
   ```
   mystere = [arbre[9] for arbre in DATA]
   ```
2. Déterminer le minimum et le maximum des hauteurs puis des diamètres des arbres recensés dans la ville de Bordeaux. On reportera les valeurs dans le tableau ci-après.

	Hauteurs des arbres	Diamètres des arbres
Minimum		
Maximum		

3. Même question avec la moyenne, la médiane et l'écart-type.

	Hauteurs des arbres	Diamètres des arbres
Moyenne		
Médiane		
Écart-type		

Exercice 10 | Visualisation par histogramme Solution >_

1. Tracer l'histogramme représentant les hauteurs des arbres de la ville de Bordeaux réparties en 50 classes.
2. Tracer l'histogramme représentant les diamètres des arbres de la ville de Bordeaux réparties en 100 classes.

3.3. Statistiques bivariées

Exercice 11 | Régression entre diamètre et hauteur Solution >_ Afficher le nuage de points correspondant aux hauteurs des arbres en abscisses, et diamètres en ordonnées, ainsi que la droite de régression. Un ajustement affine vous semble-t-il pertinent ?

Exercice 12 | Étude de familles d'arbres Solution >_ La colonne n°13 de la base contient les familles d'arbres contenues dans la base.

1. On souhaite dans un premier temps savoir quelles sont les familles les plus représentatives de la base de données, *i.e.* celles contenant au moins 10000 arbres.
   - **1.1)** Créer une liste `Lfam` la liste des familles présentes dans la base.
   - **1.2)** Écrire une fonction `selection_famille(N)` qui retourne la liste des familles de `Lfam` apparaissant au moins N fois. *On pourra utiliser le dictionnaire des occurrences de `Lfam`*
   - **1.3)** Quelles sont les familles d'arbres comportant plus de 3000 spécimens dans l'agglomération bordelaise? *On stockera le résultat dans une liste appelée `Lfam_frequent`*
2. Plutôt que d'effectuer une régression linéaire sur la base totale d'arbres, nous allons nous restreindre aux familles significatives trouvées précédemment (celles stockées dans `Lfam_frequent`). Écrire une fonction d'en-tête `meilleur_nuage(Lfam_frequent)` qui calcule le meilleur coefficient de détermination (pour la régression diamètre en ordonnée en fonction de la hauteur) sur toutes les familles d'arbres de `Lfam_frequent`, puis affiche la droite de régression ainsi que le point moyen et le nuage, et retourne le nom de la famille qui a été tracé. *On utilisera bien entendu la fonction nuage.*

4. SOLUTIONS DES EXERCICES

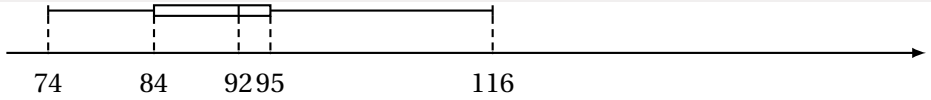
Solution (exercice 1) Énoncé Expliquons tout d'abord l'aspect mathématique, on commence par ordonner les valeurs :

- A : 74 – 81 – 84 – 88 – 91 – 91 – 93 – 95 – 95 – 95 – 102 – 116
- B : 94 – 94 – 95 – 95 – 104 – 105 – 113 – 113 – 144 – 153

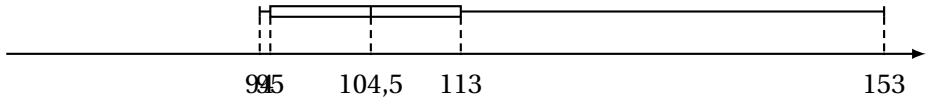
L'étendue de la série A est de 116 – 74, soit $\boxed{42}$. Sa médiane est la moyenne entre les deux valeurs du milieu, 91 et 93, soit $\boxed{92}$. L'étendue de la série B est de 153 – 94, soit $\boxed{59}$. Sa médiane est la moyenne entre les deux valeurs du milieu, 104 et 105, soit $\boxed{104.5}$. Pour la série A, on trouve $Q_1 = 84$ et $Q_3 = 95$.

1.
```
A = [74, 81, 84, 88, 91, 91, 93, 95, 95, 95, 102, 116]
B = [94, 94, 95, 95, 104, 105, 113, 113, 144, 153]
```
```
>>> moyenne(A)
92.08333333333333
>>> moyenne(B)
111.0
>>> quartiles(A)
(84, 92.0, 95)
>>> quartiles(B)
(95, 104.5, 113)
>>> etendue(A)
42
>>> etendue(B)
59
>>> maximum(A)
116
>>> minimum(A)
74
>>> maximum(B)
153
>>> minimum(B)
94
```

2.

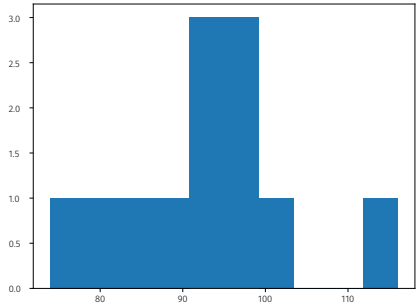


Pour la série B, on trouve $Q_1 = 95$ et $Q_3 = 113$. Il reste à construire les deux diagrammes.

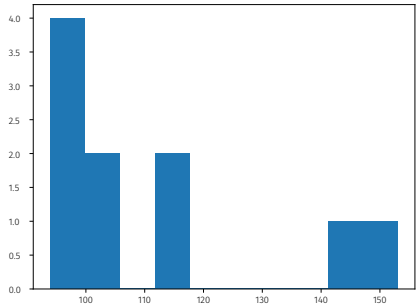


Les valeurs de la série A sont sensiblement plus faibles que celles de la série B : le médicament semble efficace. Pour conclure de manière plus qualitative, il faudrait cependant faire plutôt un test d'adéquation (voir programme de 2ème année).

3.
```
plt.hist(A, 10)
```



```
plt.hist(B, 10)
```



Solution (exercice 2) Énoncé

1.
```
D = {5:8, 10:17, 15:12, 20:8, 25:5, 30:3, 35:2, 40:1}
L = dico_occur_vers_liste(D)
```
```
>>> mediane(L)
15.0
>>> moyenne(L)
15.625
>>> etendue(L)
35
```

2. 2.1) On fait une recherche de valeur maximale du dictionnaire, on retourne ensuite la clef associé. Cette fonction est analogue à la fonction

`maximum` déjà rencontrée pour les listes. On initialise par exemple la valeur à zéro, puisque les valeurs du dictionnaire sont positives (effectifs). La petite difficulté ici est d'avoir l'une des modalités, c'est pour cette unique raison qu'on indique `L` en argument de fonction plutôt que le dictionnaire des effectifs.

```
def mode(L):
    eff_max = 0
    mode = L[0]
    D = dico_occur(L)
    for modalite in D:
        if D[modalite] > eff_max:
            eff_max = D[modalite]
            mode = modalite
    return mode
```

2.2) On l'exécute sur la série précédente.

```
>>> mode(L)
10
```

Solution (exercice 3) Énoncé

1. On note x la série des vitesses, et y la série des distances. On calcule la moyenne des deux séries : on obtient $\overline{x} = 75$ et $\overline{y} = 30$. On calcule également la moyenne de la série produit, et on obtient $\overline{xy} = 2627.5$. Grâce à la formule de KÖNIG-HUYGENS, on a alors :

$$\mathbb{C}_{x,y} = \overline{xy} - \overline{x} \times \overline{y} = 377.5.$$

Pour en déduire le coefficient de corrélation, on doit calculer les écart-types de x et y. Grâce à la formule de KÖNIG-HUYGENS on a :

$$\mathbb{V}_x = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - \overline{x}^2 = 525, \quad \mathbb{V}_y = \frac{1}{N}\sum_{i=1}^{N} y_i^2 - \overline{y}^2 = 275.$$

On obtient ainsi $\rho_{x,y} = \dfrac{\mathbb{C}_{x,y}}{\sigma_x \sigma_y} = \dfrac{\mathbb{C}_{x,y}}{\sqrt{\mathbb{V}_x \mathbb{V}_y}}$, soit $\boxed{\rho_{x,y} \simeq 0.99}$. On remarque que le coefficient de corrélation est très proche de 1 : les données sont quasiment alignées sur une droite.

2. D'après le cours, on sait que la droite d'ajustement affine a pour équation $y = ax + b$, avec $a = \dfrac{\mathbb{C}_{x,y}}{\mathbb{V}_x}$ et $b = \overline{y} - a\overline{x}$. On en déduit alors $a \simeq 0.72$ et $b \simeq -23.9$. L'équation de la droite d'ajustement affine est donc de $\boxed{y = 0.72x - 23.9}$.

3. D'après la régression linéaire, on a, pour une vitesse de 120 km/h, une distance de $0.72 \times 120 - 23.9$, soit environ $\boxed{62.4\text{m}}$.

Solution (exercice 4) Énoncé

1. Ici, on ne donne pas la série de manière exhaustive. On doit dégager les quantités nécessaires au calcul du coefficient directeur et de l'ordonnée à l'origine de la droite de régression linéaire. Définissions quelques variables dans Python afin d'effectuer les calculs.

```
>>> n = 70
>>> x_bar = 125/n
>>> y_bar = 1160/n
>>> xy_bar = 2730/n
>>> x_2bar = 295
>>> cov_xy = xy_bar - x_bar*y_bar
>>> var_x = x_2bar - x_bar**2
>>> a, b = cov_xy / var_x, y_bar - a*x_bar
>>> a, b
(0.03224058047032082, 15.7704397343914)
```

Maintenant que nous avons nos coefficients de la droite de régression, on peut répondre à la première question.

```
>>> a*4+b
15.899402056272685
```

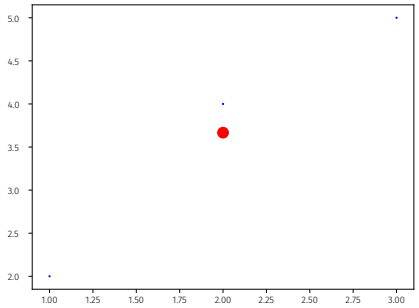
2. Il s'agit ici de comparer les coefficients directeurs.

```
>>> a
0.03224058047032082
```

La croissance est donc beaucoup plus importante pour la deuxième famille de poissons.

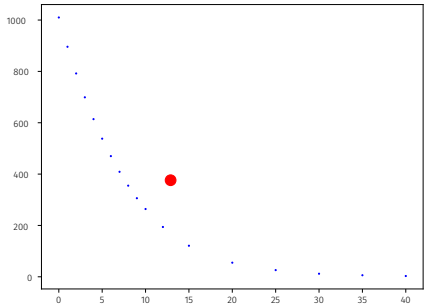
Solution (exercice 5) Énoncé

```
def nuage(L1, L2):
    plt.plot(L1, L2, "bo", markersize=1) # o pour des points \
    ↪ non reliés
    # point moyen
    x, y = moyenne(L1), moyenne(L2)
    plt.plot(x, y, "ro",  markersize=10)
nuage([1, 2, 3], [2, 4, 5])
plt.show()
```



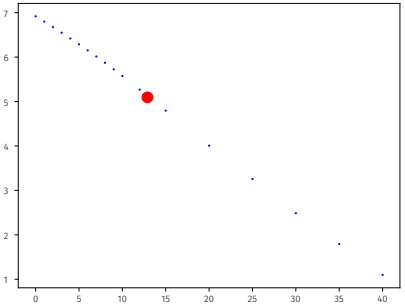
Solution (exercice 6) Énoncé

1.
```
L_alt = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 25, \
↪ 30, 35, 40]
L_press = [1010, 896, 792, 699, 614, 538, 470, 409, 355, \
↪ 306, 264, 194, 121, 55, 26, 12, 6, 3]
nuage(L_alt, L_press)
```



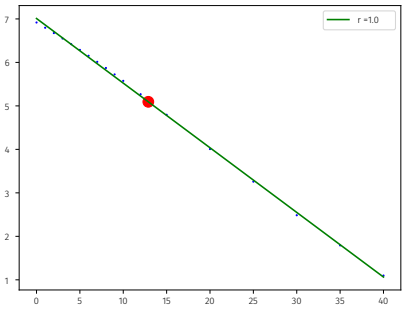
D'après la figure, le modèle linéaire ne semble donc pas très pertinent.

2.
```
L_lnpress = [ma.log(p) for p in L_press]
nuage(L_alt, L_lnpress)
```



Un ajustement affine semble donc adapté à cette nouvelle série bivariée (x,z).

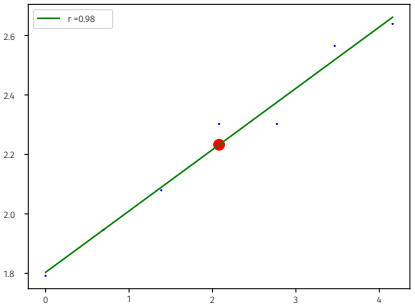
3.
```
>>> a, b, _ = nuage_reg(L_alt, L_lnpress)
>>> ma.exp(a*50+b) # pression estimée à 50 km, sans \
↪ l'exponentielle c'est son log
0.6554755510356131
```



Solution (exercice 7) Énoncé

1. Si la relation est vérifiée, alors en passant au logarithme on obtient : $\ln n = \ln C + Z \ln s$. Ainsi, la série $(\ln s, \ln n)$ suit un modèle affine.

2.
```
L_s = [1, 2, 4, 8, 16, 32, 64]
L_n = [6, 7, 8, 10, 10, 13, 14]
L_lns = [ma.log(x) for x in L_s]
L_lnn = [ma.log(x) for x in L_n]
a, b, rho = nuage_reg(L_lns, L_lnn)
```

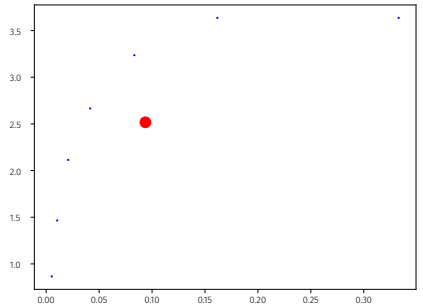


3. Le coefficient de corrélation vaut 0.98 d'après le question précédente. On suspecte donc une relation linéaire entre $\ln N$ et $\ln S$.

4. En reprenant la première question, on observe que $\ln C$ est l'ordonnée à l'origine de la droite de régression, et z son coefficient directeur.
```
>>> a # valeur approchée de z
0.20625981968084586
>>> ma.exp(b) # valeur approchée de C
6.070382539605234
```

Solution (exercice 8) Énoncé

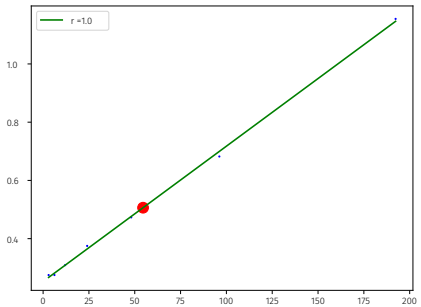
1.
```
L_C = [0.0052, 0.0104, 0.0208, 0.0416, 0.0833, 0.1617, \
↪ 0.3330]
L_v = [0.866, 1.466, 2.114, 2.666, 3.236, 3.636, 3.636]
nuage(L_C, L_v)
```



2. 2.1) On a :

$$v = \frac{k_1 \times C}{k_2 + C} \iff \frac{1}{v} = \frac{k_2 + C}{k_1 \times C} = \frac{1}{k_1}\left(\frac{k_2}{C} + 1\right) = \frac{k_2}{k_1}\frac{1}{C} + \frac{1}{k_1}.$$

2.2)
```
L_Cinv = [1/x for x in L_C]
L_vinv = [1/x for x in L_v]
a, b, rho = nuage_reg(L_Cinv, L_vinv)
```



2.3) Le réel k_1 correspond alors à l'inverse de l'ordonnée à l'origine de la régression, et k_2 à k_1 fois le coefficient directeur.

```
>>> k_1 = 1/b
>>> k_2 = a*k_1
>>> k_1
3.954226739657634
>>> k_2
0.018372531986217057
```

Solution (exercice 9) Énoncé

1. La commande crée par compréhension la liste des hauteurs sous forme de chaînes.

```
mystere = [arbre[9] for arbre in DATA]
>>> mystere[0:5] # les 5 premières
['20', '21', '35', '19', '12']
```

2. Pour cela on doit :

- créer une liste qui contient tous les diamètres,
- créer une liste qui contient toutes les hauteurs (déjà fait, il faut juste convertir chaque chaîne en flottant),
- puis calculer le minimum et maximum associés.

```
L_haut = [float(arbre[9]) for arbre in DATA]
L_diam = [float(arbre[12]) for arbre in DATA]
min_haut = minimum(L_haut)
min_diam = minimum(L_diam)
max_haut = maximum(L_haut)
max_diam = maximum(L_diam)
>>> min_haut
1.0
>>> min_diam
0.0
>>> max_haut
90.0
>>> max_diam
281.0
```

On en déduit alors le tableau complété.

	Hauteurs des arbres	Diamètres des arbres
Minimum	1.0	0.0
Maximum	90.0	281.0

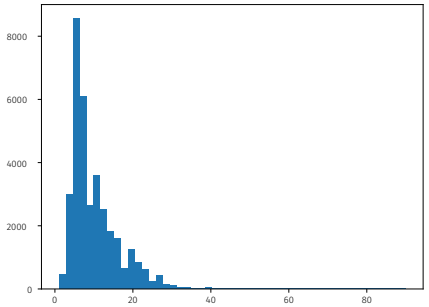
```
moy_haut = moyenne(L_haut)
moy_diam = moyenne(L_diam)
med_haut = mediane(L_haut)
med_diam = mediane(L_diam)
et_haut = variance(L_haut)**(0.5)
et_diam = variance(L_diam)**(0.5)
```

3.

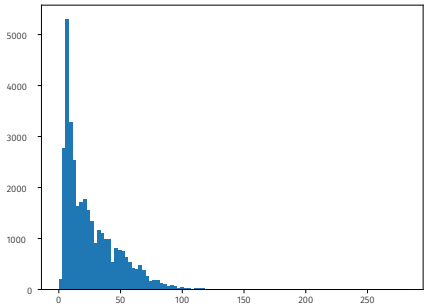
	Hauteurs des arbres	Diamètres des arbres
Moyenne	10.08	26.52
Médiane	8.0	19.0
Écart-type	6.06	22.64

Solution (exercice 10) Énoncé

1. `plt.hist(L_haut, 50)`

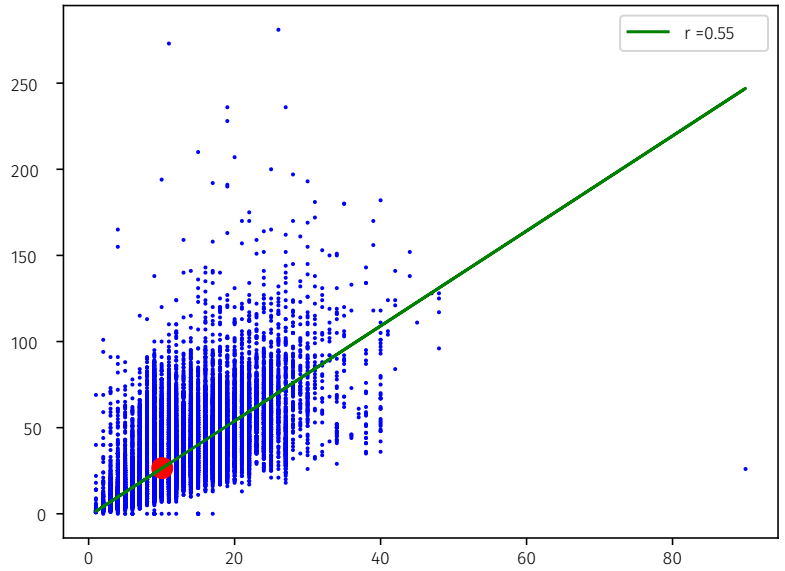


2. `plt.hist(L_diam, 100)`



Solution (exercice 11) Énoncé

```
nuage_reg(L_haut, L_diam)
```



Le coefficient de détermination est ici catastrophique, la régression n'est donc pas du tout pertinente.

Solution (exercice 12) Énoncé

1. **1.1)**

```
Lfam = []
for arbre in DATA:
    Lfam.append(arbre[13])
```

```
>>> Lfam[:10]
['Platanaceae', 'Platanaceae', 'Platanaceae', 'Platanac
eae', 'Platanaceae', 'Platanaceae', 'Platanaceae', 'Sap
indaceae', 'Sapindaceae', 'Platanaceae']
>>> len(Lfam) # nombre de familles d'arbres
34702
```

1.2) On parcourt les familles de `Lfam` et on regarde si elle apparaît au moins 10000 fois dans `DATA`.

```
def selection_famille(N):
    Lfam_frequent = []
    D = dico_occur(Lfam)
    for famille in D:
        if D[famille] > N:
            Lfam_frequent.append(famille)
    return Lfam_frequent
```

1.3)

```
>>> Lfam_frequent = selection_famille(3000)
```

```
>>> Lfam_frequent
['Platanaceae', 'Sapindaceae', 'Fagaceae', 'Malvaceae']
```

2. On doit donc parcourir les familles et chercher d'abord le meilleur coefficient de détermination (recherche de max).

```
def meilleur_nuage(Lfam_frequent):
    r_max = 0 # initialisation du max
    famille_max = ''
    for famille in Lfam_frequent:
        L_haut = [float(arbre[9]) for arbre in DATA if \
        ↪ arbre[13] == famille]
        L_diam = [float(arbre[9]) for arbre in DATA if \
        ↪ arbre[13] == famille]
        r = coeff_cor(L_haut, L_diam)**2
        if r > r_max:
            r_max = r
            famille_max = famille
    # on a trouvé la meilleure famille, on effectue la \
    ↪ régression
    L_haut = [float(arbre[9]) for arbre in DATA if \
    ↪ arbre[13] == famille_max]
    L_diam = [float(arbre[9]) for arbre in DATA if \
    ↪ arbre[13] == famille_max]
    nuage(L_haut, L_diam)
    return famille_max
```

```
>>> meilleur_nuage(Lfam_frequent)
'Platanaceae'
```