# TP 24
# Statistiques

Ce TP est à lire en parallèle du chapitre de mathématiques sur les statistiques. Ses buts sont multiples :

- Savoir écrire des fonctions Python qui calculent les différents indicateurs statistiques étudiés (moyenne, médiane, etc), en se remémorant au passage leur définitions et quelques-unes de leurs propriétés.
- Appliquer ces notions sur des exemples concrets de données statistiques, qu'il est bien plus aisé de manipuler en séance informatique qu'en TD.
- Introduire la bibliothèque Pandas, pour traiter les données mais aussi pour obtenir facilement des représentations graphiques.

Tout du long nous utiliserons le fichier joint `tips.csv` qui contient des statistiques sur des repas au restautant et les pourboires laissés (en anglais *tips*), en fonction du montant total de la facture, du jour et du moment de la journée, et des clients. L'origine de ce fichier est bien ancienne et les données semblent être issues réellement d'un serveur au restaurant. Les prix sont en Dollar. La question générale qu'on peut se poser est : quels facteurs influencent la facture et le pourboire laissés par les clients ?

## I Aperçu de la bibliothèque Pandas

La bibliothèque Pandas a pour but de manipuler des données statistiques en Python. Elle repose en fait largement sur Numpy (pour enregistrer les données) et sur Matplotlib (pour les représenter graphiquement). On va donc charger tout cela avec les commandes

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

puis on charge d'un seul coup tout le fichier de statistiques avec

```
tips = pd.read_csv("tips.csv")
```

Observez le résultat :

```
>>> tips
     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
```

La variable `tips` est de type « *Pandas DataFrame* », et représente d'un coup tout ce tableau de données. C'est une table, comme nous en avons déjà vues, en lignes les différentes entrée (une entrée pour chaque repas au restaurant, numérotés), en colonne les attributs (ou ici : caractères) étudiés.

**Exercice 1** *Mise en jambe*
Quels sont les caractères édudiés ici ? Lesquels sont qualitatifs et lesquels sont quantitatifs ?

On peut accéder à chacune des colonnes avec la même syntaxe que pour les listes et les dictionnaires, chaque colonne est de type « *Pandas Series* » (série statistique à une variable) :

```
>>> tips["total_bill"]
0      16.99
1      10.34
2      21.01
3      23.68
4      24.59
       ...
239    29.03
240    27.18
241    22.67
242    17.82
243    18.78
Name: total_bill, Length: 244, dtype: float64
```

Remarquez le tout dernier mot en bas à droite : chaque série a un type des données, hérités de Numpy, ici `float64` indique bien que ce sont des nombres à virgule flottante. Autre exemple, qualitatif :

```
>>> tips["smoker"]
0       No
1       No
2       No
3       No
4       No
      ...
239     No
240    Yes
241    Yes
242     No
243     No
Name: smoker, Length: 244, dtype: object
```

On peut obtenir le résumé de ces informations pour toutes les colonnes d'un coup :

```
>>> tips.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    object
 3   smoker      244 non-null    object
 4   day         244 non-null    object
 5   time        244 non-null    object
 6   size        244 non-null    int64
dtypes: float64(2), int64(1), object(4)
memory usage: 13.5+ KB
```

Les mots `non-null` qui apparaissent signifient que Pandas donne la possibilité d'avoir des données manquantes. Comme en SQL, une moyenne ou un compte total ne sont pas les mêmes si on ignore les données manquantes ou si on les complète par zéro.

Enfin, on peut accéder individuellement à la ligne numérotée $i$ à travers la méthode `loc[i]` : sur toute la table

```
>>> tips.loc[0]
total_bill     16.99
tip             1.01
sex           Female
smoker            No
day              Sun
time          Dinner
size               2
```

ou bien sur une seule colonne

```
>>> tips["total_bill"].loc[0]
16.99
```

ou sélectionner seulement une partie de la table qui nous intéresse, par exemple garder seulement les repas du samedi (condition `tips["day"] == "Sat"`) :

```
>>> tips[tips["day"] == "Sat"]
     total_bill   tip     sex smoker  day    time  size
19        20.65  3.35    Male     No  Sat  Dinner     3
20        17.92  4.08    Male     No  Sat  Dinner     2
21        20.29  2.75  Female     No  Sat  Dinner     2
22        15.77  2.23  Female     No  Sat  Dinner     2
23        39.42  7.58    Male     No  Sat  Dinner     4
..          ...   ...     ...    ...  ...     ...   ...
238       35.83  4.67  Female     No  Sat  Dinner     3
239       29.03  5.92    Male     No  Sat  Dinner     3
240       27.18  2.00  Female    Yes  Sat  Dinner     2
241       22.67  2.00    Male    Yes  Sat  Dinner     2
242       17.82  1.75    Male     No  Sat  Dinner     2

[87 rows x 7 columns]
```

En résumé les objets *Pandas DataFrame* peuvent être facilement découpés (par lignes ou par colonnes), transformés, fusionnés, et on peut même retrouver des concepts bien connus du SQL (requêtes `SELECT`, avec `WHERE`, `ORDER BY`, `GROUP BY`) mais tout en Python.

Cependant **notre but n'est pas d'approfondir sur la bibliothèque Pandas**. Pour nous l'intérêt principal est d'avoir un unique objet qui contient d'un coup toute la table, dont on peut extraire de diverses façons les données qu'on souhaite. À tout moment, les colonnes peuvent être converties en banales listes Python, pour pouvoir appliquer dessus toutes les méthodes que nous connaissons déjà :

```
>>> tips["total_bill"].to_list()
[16.99, 10.34, 21.01, 23.68, 24.59, ...]
```

Ainsi les exercices se feront uniquement en travaillant avec des listes. Les tests sont dans le fichier joint, ainsi que toutes les syntaxes pouvant être un petit peu subtiles : bien observer les exemples et les tests, et faire le lien avec notre problème de pourboires.

# II Indicateurs statistiques à une variable

Commençons par les plus classiques et aussi les plus faciles à programmer. On rappelle qu'on note en mathématiques $x = (x_i)_{1 \leqslant i \leqslant N}$ une **série statistique à une variable**, $N$ est l'**effectif**, et la **moyenne** $\overline{x}$ est

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

**Exercice 2** *Classique classique*
On travaille avec une liste Python habituelle, supposée non vide, qu'on considère comme une série statistique à une variable. Écrire les fonctions suivantes :

1. `moyenne(L)` : la moyenne de la liste `L`,
2. `maximum(L)` : le maximum de la liste,
3. `minimum(L)` : le minimum.

La **variance** $s_x^2$, elle, est définie comme la moyenne des carrés des écarts à la moyenne :

$$s_x^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2$$

et l'**écart-type** $s_x$ est la racine carrée de la variance :

$$s_x = \sqrt{s_x^2}$$

Pour calculer la variance, on pourrait penser à utiliser la fonction `moyenne` déjà écrite. Mais il est plus efficace d'utiliser la formule de König-Huygens :

**Théorème**

Formule de König-Huygens :

$$s_x^2 = \overline{x^2} - \overline{x}^2$$

(la variance est égale à la moyenne des carrés, moins le carré de la moyenne).

**Exercice 3** *Mathématiques*
Démontrer cette formule.

Ainsi la méthode la plus pratique consiste à calculer, *simultanément avec une seule boucle*, la somme de la série statistique ainsi que la somme des carrés (par exemple en utilisant deux variables accumulatrices, `S` pour la somme et `SS` pour la somme des carrés) ; puis à appliquer la formule de König-Huygens ci-dessus.

**Exercice 4**

1. Écrire la fonction `variance(L)` qui calcule la variance de la liste `L`, par la méthode décrite ci-dessus.
2. En déduire une fonction `ecarttype(L)` qui calcule l'écart-type. On pourra utiliser la fonction `sqrt` du module Numpy pour la racine carrée.

# III La médiane et le problème du regroupement en classes

La médiane d'une série statistique est facile à calculer quand la liste est déjà *triée* : alors c'est simplement la valeur du milieu… En général ce n'est pas aussi simple. Un cas intéressant à traiter est celui où les données sont déjà regroupées par classe, et c'est ce que nous allons traiter maintenant.

Les caractères qualitatifs sont souvent donnés en les regroupant entre eux, et en donnant l'effectif de chaque classe (par exemple : combien de repas chaque jour de la semaine). Pandas sait très bien le faire :

```
>>> tips.groupby("day").size()
day
Fri     19
Sat     87
Sun     76
Thur    62
```

Les données quantitatives, telles qu'ici le montant de la facture, peuvent aussi être regroupées par classes. On se fixe un certain nombre d'intervalles donnés par leur bornes $a_0 < a_1 < \cdots < a_p$ et on donne le nombre de valeurs

de la série qui sont dans chacun des intervalles $[a_0, a_1[$, $[a_1, a_2[$, ..., $[a_{p-1}, a_p[$. Ici pour le montant total de la facture, on va prendre pour bornes, pour commencer, les nombres entiers.

**Exercice 5**

1. Écrire une fonction `regroupe(L)` qui, étant donnée une liste `L` représentant une série statistique à une variable (on suppose que ce sont uniquement des valeurs positives, mais pas nécessairement entières), renvoie une liste `C` telle que `C[j]` est le nombre de valeurs de `L` comprises dans l'intervalle $[j, j+1[$.

   On pourra pour cela utiliser la fonction `maximum` pour déterminer la taille de `C` ; et la fonction `int(x)` pour obtenir la partie entière d'un nombre à virgule flottante `x`.

2. Bonus : écrire une fonction `regroupe_dixième(L)` qui renvoie une liste `C` comme ci-dessus telle que `C[j]` est le nombre de valeurs de `L` qui sont dans l'intervalle $\left[\frac{j}{10}, \frac{j+1}{10}\right[$.

Venons-en maintenant au calcul de la médiane, pour des données déjà regroupées par classe. Il y a plusieurs variantes possibles de la définition de la médiane ; nous adoptons la définition suivante, pour laquelle la médiane est définie uniquement et qui est la plus pratique à programmer :

**Définition**

La **médiane** d'une série statistique $x$ est la *plus petite valeur de la série* telle que *au moins* la moitié de l'effectif soit inférieur ou égal à la médiane.

Ainsi par exemple la médiane de la liste $(1, 4, 8, 9, 11)$ est bien égale à 8 (pour une liste de longueur impaire, il y a bien une valeur au milieu), la médiane de $(1, 3, 4, 8, 9, 11)$ est égale à 4 (la valeur juste avant le milieu : il y a 3 valeurs qui lui sont inférieures ou égales, et 3 autres), et la médiane de $(1, 2, 4, 4, 4, 8)$ est aussi égale à 4 (c'est bien la plus petite valeur de la série pour laquelle il y a au moins 3 valeurs de la série qui lui sont inférieures, ici il y a même 5 valeurs qui lui sont inférieures ou égales).

**Exercice 6**

On se donne une liste `C` provenant de la fonction `regroupe` de l'exercice précédent : `C` représente une série statistiques regroupée par classes, où `C[j]` donne le nombre de valeurs égales à `j`.

1. Écrire une fonction `cumuls(C)` qui renvoie la liste des effectifs cumulées de `C` : une liste `D` de même longueur que `C` où `D[j]`donne le nombre de valeurs qui sont inférieures ou égales à `j`.
2. Écrire une fonction `mediane(C)` qui renvoie la médiane de la série statistique représentée par `C`.

On définit aussi le **premier quartile** $Q_1$ (resp. **troisième quartile** $Q_3$) comme la plus petite valeur de la série telle qu'au moins le quart (resp. les trois quarts) de l'effectif lui soit inférieur ; et l'**écart inter-quartile** est $Q_3 - Q_1$. La médiane se note aussi $Q_2$, car c'est le deuxième quartile.

Pour ne pas ré-écrire plusieurs fois la même fonction, on définit pour tout nombre $p \in [0, 1]$ (souvent exprimé en pourcentage) le **quantile d'ordre *p*** comme la plus petite valeur de la série telle qu'au moins la proportion $p$ de l'effectif lui soit inférieur. Alors $Q_1$ est bien le quantile d'ordre $\frac{1}{4}$ et $Q_3$ est celui d'ordre $\frac{3}{4}$.

**Exercice 7**

1. Écrire la fonction `quantile(C, p)`.
2. En déduire les fonctions `quartile1(C)` (premier quartile), `quartile3(C)` (troisième quartile) et `interquartile(C)` (écart inter-quartiles).

# IV Représentations graphiques

**Aucune connaissance spécifique à la bibliothèque Pandas** n'est exigible, et toutes les fonctions présentées ici existent dans Matplotlib (Pandas agit ici comme une sur-couche de Matplotlib : il présente des fonctions et des types d'objets différents, mais à la fin, il les passe à Matplotlib).

Pour tracer nos graphiques, nous avons parfois besoin de choisir la colonne qui est concernée ; de grouper les données qualitatives (section précédente) ; ou bien de fabriquer des petits tableaux croisés d'effectifs. Ceux-ci peuvent être réalisés à l'aide de la fonction Pandas `crosstab(X, Y)` :

```
>>> pd.crosstab(tips["day"], tips["sex"])
sex   Female  Male
day
Fri        9    10
Sat       28    59
Sun       18    58
Thur      32    30
```

Dans le fichier joint, **observez les programmes et les représentations graphiques**, notamment observez bien la table qui est produite avant de tracer.

**Diagramme en barres** En anglais c'est un *bar plot*, la fonction s'appelle `plot.bar()`. En abscisses on a des donnés **qualitatives** (des « étiquettes »), en ordonnée leur valeur correspondante. Éventuellement, si on a plusieurs séries de données partageant les mêmes abscisses (résultant d'un tableau croisé d'effectif comme ci-dessus), on peut tracer plusieurs barres côte à côte, ou bien empilées.

**Diagramme en camembert** En anglais *pie plot* (une tarte, plutôt qu'un camembert), la fonction est `plot.pie()`. Le contexte est en fait **le même que pour un diagramme en barres** : étant données des données qualitativés, et leur valeur correspondante, les données sont représentées comme des parts sur un camembert, dont l'angle au centre est proportionnel à la valeur. Éventuellement, si on a plusieurs séries de valeurs partageant les mêmes étiquettes, on peut placer plusieurs camemberts côte à côte.

**Boite à moustaches** En anglais *box plot*, fonction `plot.box()` ou `boxplot()`. C'est une représentation d'une **série statistique à une variable** (elle s'applique donc directement sur `tips["total_bill"]`, par exemple). La boite indique les premiers et troisièmes quartiles, et la médiane avec une barre verticale. Les moustaches s'étendent hors de la boite, diverses conventions existent : celle par défaut de Pandas et de Matplotlib est que leur longueur est au maximum 1,5 fois l'écart inter-quartile, en s'arrêtant sur la valeur de la série la plus éloignée possible. Les éventuelles valeurs dépassant cette limite sont indiquées par un point individuel. Une autre convention est que les moustaches s'étendent du premier au neuvième décile (ainsi il y a précisement 10 % de l'effectif hors des moustaches d'un côté et 10 % de l'autre). Les boites sont par défaut tracées verticalement, mais on peut aussi les tracer horizontalement. Là encore, on peut grouper une série statistique selon un critère pour placer plusieurs boites côte à côte.

**Histogramme** Cela ressemble à première vu au diagramme en barres ; mais en abscisse, ce sont des données **quantitatives regroupées en classes**, et en ordonnée l'effectif correspondant. La fonction est `plot.hist()` ou bien `hist()`. Un argument optionnel `bins` (casiers, corbeilles) permet de choisir le nombre de classes, plus il y en a plus la courbe va être sembler fine (mais s'il y en a trop par rapport au nombre de données, les irrégularités vont se voir). On peut aussi l'utiliser pour visualiser facilement les effectifs cumulés.

**Nuage de points** C'est une représentation d'une série statistique à deux variables. En anglais *scatter plot*, fonction `plot.scatter()`, à laquelle on passe les données en abscisses et les données en ordonnées.

**À vous !**

**Exercice 8**
Observer le fichier joint et les différents graphiques, et tracer quelques graphiques supplémentaires (en groupant selon un critère, ou bien en filtrant) pour déterminer quels facteurs influencent la facture et le pourboire.

# V Statistiques à deux variables

Une série statistiques à deux variables est donnée par une liste de couples $(x_i, y_i)_{1\leqslant i\leqslant N}$ : c'est la mesure de deux caractères $x_i$ et $y_i$ pour un même individu $i$. On la représente en Python par deux listes `X` et `Y` supposées de la même taille, ainsi les mesures de l'individu $i$ sont `X[i]` et `Y[i]`. La représentation graphique la plus naturelle est un nuage de points, et le points de coordonnées $(\overline{x}, \overline{y})$ s'appelle le **point moyen**. On rappelle la définition de la **covariance** $s_{x,y}$ :

$$s_{x,y} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})$$

(si $x = y$, on retrouve la définition de la variance), ainsi que du **coefficient de corrélation linéaire** $r^2_{x,y}$ et $r_{x,y}$

$$r_{x,y}^2 = \frac{s_{x,y}^2}{s_x^2 s_y^2} \qquad r_{x,y} = \frac{s_{x,y}}{s_x s_y}$$

(les notations cachent le fait que pour calculer $r_{x,y}$, il y a des racines carrées au dénominateur).

**Exercice 9** *Mathématiques*
Démontrer l'inégalité $|r_{x,y}| \leqslant 1$, avec égalité si et seulement si il existe une relation affine entre $x$ et $y$ (c'est-à-dire des nombres $a$ et $b$ tels que $\forall 1 \leqslant i \leqslant N$, $y_i = ax_i + b$).

La covariance se calcule plus facilement avec la formule de König-Huygens :

$$s_{x,y} = \overline{x \times y} - \overline{x} \times \overline{y}$$

(là encore, pour $x = y$ on retrouve la version précédente de König-Huygens). Ainsi la méthode la plus pratique pour calculer la covariance est de calculer, avec une seule boucle, à la fois la somme de la série $x$, la somme de la série $y$, et la somme des $x \times y$ ; puis d'appliquer la formule.

**Exercice 10**
1. Écrire la fonction `covariance(X, Y)`.
2. En déduire les fonctions `r2(X, Y)` et `r(X, Y)`.

L'intérêt de la covariance, c'est de pouvoir étudier la droite de régression affine de la série statistique $y$ par rapport à $x$. Il s'agit de l'unique droite qui passe par le point moyen $(\overline{x}, \overline{y})$ et dont le coefficient directeur est

$$a = \frac{s_{x,y}}{s_x^2}$$

(dans le cas où il existe une relation affine $y = ax + b$, alors par les propriétés formelles de la covariance, on trouve $s_{x,y} = s_{ax+b,x} = as_{x,x} = as_x^2$ donc la valeur de $a$ est bien $\frac{s_{x,y}}{s_x^2}$) ; c'est donc la droite d'équation $Y = a(X - \overline{x}) + \overline{y}$.

**Exercice 11** *Révisions Matplotlib*
1. Tracer sur un même graphique le nuage de points des pourboires en fonction du montant total de la facture, avec le point moyen (bien marqué d'une autre couleur) et la droite de régression affine. Quelle interprétation donner ici du coeffecient $a$ ?
2. Tracer plusieurs autres nuages de points, en filtrant selon certains critères (uniquement les hommes ou les femmes ; ou bien les fumeurs ou les non-fumeurs), en déterminant à chaque fois le coefficient $a$.