# SY09 Printemps 2022
# TD/TP 01 — Manipulation de données

## 1 Travaux pratiques

### 1.1 Chargement d'un jeu de données

Les jeux de données sont communément stockés dans des fichiers textes au format dit « csv » (*comma separated value*). Il s'agit d'un format décrivant un tableau individus–variables : une ligne liste les caractéristiques d'un individu, séparées par une virgule ; et une colonne liste les valeurs d'une variable pour tous les individus. Dans certains fichiers, la première ligne est parfois une ligne d'en-tête (ou `header`) spécifiant le nom de chacun des prédicteurs. Parfois, la première colonne n'est pas un prédicteur mais un identifiant ou un nom d'individu qui n'est pas un prédicteur. Les fichiers « csv » ont plusieurs variantes, le séparateur (la virgule pour le fichier « csv ») peut changer. La plupart du temps, le séparateur est une virgule, une espace, un point virgule ou une tabulation.

Pour charger des données représentant un tableau individus–variables, on utilise la bibliothèque `pandas`. On la charge avec l'instruction suivante

```
In  [1]: import pandas as pd
```

Pour charger un fichier `csv`, on utilise la fonction `pd.read_csv` en spécifiant le chemin du fichier `csv` à charger.

1 Charger le fichier `data/sy02-p2016.csv` dans la variable `X`.

```
In  [2]: X = pd.read_csv("data/sy02-p2016.csv")
```

Pour contrôler le bon chargement des données, on peut vérifier le nombres de caractéristiques ainsi que le nombre d'individus avec l'attribut `shape`, le type des caractéristiques avec la méthode `info`.

2 Vérifier qu'il y a 296 individus et 11 caractéristiques.

```
In  [3]: X.shape
Out [3]: (296, 11)
In  [4]: X.info()
```

```
Out [4]: <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 296 entries, 0 to 295
         Data columns (total 11 columns):
          #   Column                  Non-Null Count  Dtype
         ---  ------                  --------------  -----
          0   nom                     296 non-null    object
          1   specialite              296 non-null    object
          2   niveau                  296 non-null    int64
          3   statut                  296 non-null    object
          4   dernier diplome obtenu  290 non-null    object
          5   note median             293 non-null    float64
          6   correcteur median       293 non-null    object
          7   note final              284 non-null    float64
          8   correcteur final        284 non-null    object
          9   note totale             284 non-null    float64
          10  resultat                296 non-null    object
         dtypes: float64(3), int64(1), object(7)
         memory usage: 25.6+ KB
```

3 En utilisant les options de chargement `sep`, `index_col` et `header`, charger les fichiers suivants :
- `data/sy02-p2016-2.csv`
- `data/sy02-p2016-3.csv`
- `data/sy02-p2016-4.csv`
- `data/sy02-p2016-5.csv`

Vérifier qu'ils contiennent les mêmes informations que le premier jeu de données.

```
In  [5]: X2 = pd.read_csv("data/sy02-p2016-2.csv", sep="&")
         X3 = pd.read_csv("data/sy02-p2016-3.csv", sep="\t")
         X4 = pd.read_csv("data/sy02-p2016-4.csv", sep=";")
         X5 = pd.read_csv("data/sy02-p2016-5.csv", sep=" ", index_col=0)
         X2.shape; X3.shape; X4.shape; X5.shape
Out [5]: (296, 11)
         (296, 11)
         (296, 11)
         (296, 11)
```

## 1.2 Conversion de types

Lors du chargement d'un fichier texte, si le type de la colonne n'est pas spécifié avec l'argument `dtype`, `Pandas` essaie de deviner le type de chaque prédicteur. Les types les plus utilisés sont les suivants
- `np.float64` : Correspond à une variable quantitative continue
- `np.int64` : Correspond à une variable quantitative discrète (les entiers naturels)
- `bool` : Correspond à une variable binaire
- `object` : Lorsqu'aucune des classes ci-dessus ne convient, le type générique `object` est utilisé
- `category` : Correspond à une variable qualitative à plusieurs modalités. `Pandas` ne convertit jamais automatiquement vers ce type, il faut le faire *a posteriori*.

```
In  [6]: from io import StringIO

         pd.read_csv(StringIO("0\n1.4"), header=None).dtypes
Out [6]: 0    float64
         dtype: object
In  [7]: pd.read_csv(StringIO("0\n1"), header=None).dtypes
```

```
Out [7]: 0    int64
         dtype: object
In  [8]: pd.read_csv(StringIO("T\nF"), header=None).dtypes
Out [8]: 0    object
         dtype: object
In  [9]: pd.read_csv(StringIO("True\nFalse"), header=None).dtypes
Out [9]: 0    bool
         dtype: object
In  [10]: pd.read_csv(StringIO("Vrai\nFaux"), header=None).dtypes
Out [10]: 0    object
          dtype: object
```

Lorsque le type n'est pas correctement détecté, on peut le corriger manuellement en faisant appel à la méthode `astype(<type>)`.

Pour les variables catégorielles, le type n'est pas encore défini. Il faut donc d'abord le définir

```
ects_type = pd.CategoricalDtype(categories=["R", "G", "B"])
```

et l'utiliser ensuite avec `astype(<type>)`

```
X.col = X.col.astype(ects_type)
```

Si le type n'est pas réutilisé pour d'autres prédicteurs, on peut directement le créer en même temps que la colonne.

```
X.col = pd.Categorical(X.col, categories=["R", "G", "B"])
```

Si les modalités sont ordonnées, on peut le spécifier avec l'argument `ordered`.

4 Corriger le type de chaque prédicteur présent dans le fichier `data/sy02-p2016.csv`.

```
In  [11]: X.specialite = pd.Categorical(X.specialite)
          X.statut = pd.Categorical(X.statut)
          X["dernier diplome obtenu"] = pd.Categorical(X["dernier diplome
           ↪  obtenu"])
          correcteur_type = pd.CategoricalDtype(
              pd.concat([X["correcteur median"], X["correcteur
               ↪  final"]]).dropna().unique()
          )
          X["correcteur median"] = X["correcteur median"].astype(correcteur_type)
          X["correcteur final"] = X["correcteur final"].astype(correcteur_type)
          X.resultat = pd.Categorical(
              X.resultat, categories=["ABS", "F", "Fx", "E", "D", "C", "B", "A"],
               ↪  ordered=True
          )
          X.info()
```

```
Out [11]: <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 296 entries, 0 to 295
          Data columns (total 11 columns):
           #   Column                  Non-Null Count  Dtype
          ---  ------                  --------------  -----
           0   nom                     296 non-null    object
           1   specialite              296 non-null    category
           2   niveau                  296 non-null    int64
           3   statut                  296 non-null    category
           4   dernier diplome obtenu  290 non-null    category
           5   note median             293 non-null    float64
           6   correcteur median       293 non-null    category
           7   note final              284 non-null    float64
           8   correcteur final        284 non-null    category
           9   note totale             284 non-null    float64
           10  resultat                296 non-null    category
          dtypes: category(6), float64(3), int64(1), object(1)
          memory usage: 15.4+ KB
```

## 1.3 Transformation

Même lorsque le jeu de données est nettoyé et qu'il ne présente plus d'erreurs manifestes, il est souvent nécessaire de transformer certains prédicteurs voire la structure elle-même du jeu de données.

Lorsque la donnée sous-jacente est de type chaine de caractères, `Pandas` fournit un nombre important de fonctions pour extraire l'information utile. On peut par exemple utiliser les *slices* :

```
In  [12]: X = pd.read_csv("data/sy02-p2016.csv")
          X.nom
Out [12]: 0        Etu1
          1        Etu2
          2        Etu3
          3        Etu4
          4        Etu5
                   ...
          291    Etu292
          292    Etu293
          293    Etu294
          294    Etu295
          295    Etu296
          Name: nom, Length: 296, dtype: object
In  [13]: X.nom.str[3:]
Out [13]: 0        1
          1        2
          2        3
          3        4
          4        5
                  ...
          291    292
          292    293
          293    294
          294    295
          295    296
          Name: nom, Length: 296, dtype: object
```

Il faut utiliser la méthode `str` pour avoir accès à toutes ces fonctions d'extraction. Pour lister ces

fonctions, on pourra exécuter l'instruction

```
dir(X.nom.str)
```

5 Le prédicteur `Semestre` du jeu de données présent dans le fichier `data/effectifs.csv` contient des données de la forme `SemestreXXXXX`. En utilisant les *slices* extraire la donnée `XXXXX`.

```
In  [14]: X = pd.read_csv("data/effectifs.csv")
          X = X.assign(Semestre=X.Semestre.str[8:])
          X
Out [14]:   Semestre  SY02   SY09  SY19
          0    P2019   220   75.0   NaN
          1    A2019   180    NaN  82.0
          2    A2018   200    NaN  78.0
          3    P2018   210   76.0   NaN
          4    A2017   189    NaN  69.0
          5    P2017   230  102.0   NaN
          6    A2016   213    NaN  52.0
          7    P2016   242   93.0   NaN
```

La donnée est maintenant de la forme « `SDDDD` » avec `S` le semestre (« A » ou « P ») et `DDDD` l'année. Cependant, cette donnée n'est toujours pas exploitable.

6 Créer deux autres colonnes contenant respectivement le semestre et l'année. On pourra utiliser la fonction `assign`.

```
In  [15]: X = X.assign(
            Saison=X.Semestre.str[0],
            Annee=X.Semestre.str[1:]
          )
          X.drop(columns="Semestre", inplace=True)
          X
Out [15]:    SY02   SY09  SY19 Saison Annee
          0   220   75.0   NaN      P  2019
          1   180    NaN  82.0      A  2019
          2   200    NaN  78.0      A  2018
          3   210   76.0   NaN      P  2018
          4   189    NaN  69.0      A  2017
          5   230  102.0   NaN      P  2017
          6   213    NaN  52.0      A  2016
          7   242   93.0   NaN      P  2016
```

Il est souvent souhaitable de factoriser plusieurs colonnes stockant des données ayant la même signification en deux colonnes seulement : une colonne stocke le nom de la colonne et l'autre la valeur correspondante. Un exemple classique est présent dans la table 1.

Pour réaliser cette opération avec `Pandas`, on utilise la fonction `melt`.

```
In  [16]: X1 = pd.DataFrame(
              dict(
                  Person=["Bob", "Alice", "Steve"],
                  Age=[32, 24, 64],
                  Weight=[128, 86, 95],
                  Height=[180, 175, 165],
              )
          )
          X1.melt(id_vars=["Person"])
```

Table 1 – Représentation « wide » et « long »

(a) Format « wide »

| Person | Age | Weight | Height |
|---|---|---|---|
| Bob | 32 | 128 | 180 |
| Alice | 24 | 86 | 175 |
| Steve | 64 | 95 | 165 |

(b) Format « long »

| Person | Variable | Value |
|---|---|---|
| Bob | Age | 32 |
| Bob | Weight | 128 |
| Bob | Height | 180 |
| Alice | Age | 24 |
| Alice | Weight | 86 |
| Alice | Height | 175 |
| Steve | Age | 64 |
| Steve | Weight | 95 |
| Steve | Height | 165 |

```
Out [16]:   Person variable  value
          0    Bob      Age     32
          1  Alice      Age     24
          2  Steve      Age     64
          3    Bob   Weight    128
          4  Alice   Weight     86
          5  Steve   Weight     95
          6    Bob   Height    180
          7  Alice   Height    175
          8  Steve   Height    165
```

On peut renommer les colonnes `variable` et `value` en utilisant les arguments `var_name` et `value_name`.

7 Convertir le jeu de données précédent au format « long », enlever les effectifs inexistants et convertir en nombre entier.

```
In  [17]: X = X.melt(id_vars=["Saison", "Annee"], value_name="effectif",
          ↪  var_name="UV")
          X = X.loc[~pd.isna(X.effectif)]
          X = X.assign(effectif=X.effectif.astype(int))
          X
Out [17]:    Saison Annee    UV  effectif
          0       P  2019  SY02       220
          1       A  2019  SY02       180
          2       A  2018  SY02       200
          3       P  2018  SY02       210
          4       A  2017  SY02       189
          5       P  2017  SY02       230
          6       A  2016  SY02       213
          7       P  2016  SY02       242
          8       P  2019  SY09        75
          11      P  2018  SY09        76
          13      P  2017  SY09       102
          15      P  2016  SY09        93
          17      A  2019  SY19        82
          18      A  2018  SY19        78
          20      A  2017  SY19        69
          22      A  2016  SY19        52
```

8 Convertir le jeu de données iris en format « long ». On pourra charger le jeu de donnée iris avec les instructions suivantes.

```
import seaborn as sns
iris = sns.load_dataset("iris")
```

```
In [18]: import seaborn as sns
         iris = sns.load_dataset("iris")
         iris = iris.melt(id_vars=["species"])
```

9 Scinder la colonne des longueurs/largeurs des sépales/pétales en deux colonnes.

```
In [19]: iris = iris.assign(
             type=iris.variable.str[:5],
             dim=iris.variable.str[6:]
         )
         iris = iris.drop(columns=["variable"])
         iris
Out [19]:        species  value   type     dim
         0       setosa    5.1  sepal  length
         1       setosa    4.9  sepal  length
         2       setosa    4.7  sepal  length
         3       setosa    4.6  sepal  length
         4       setosa    5.0  sepal  length
         ..         ...    ...    ...     ...
         595  virginica    2.3  petal   width
         596  virginica    1.9  petal   width
         597  virginica    2.0  petal   width
         598  virginica    2.3  petal   width
         599  virginica    1.8  petal   width

         [600 rows x 4 columns]
```

On peut plus généralement utiliser la fonction `str.split` avec l'argument `expand=True`.

## 1.4 Jeu de données babies

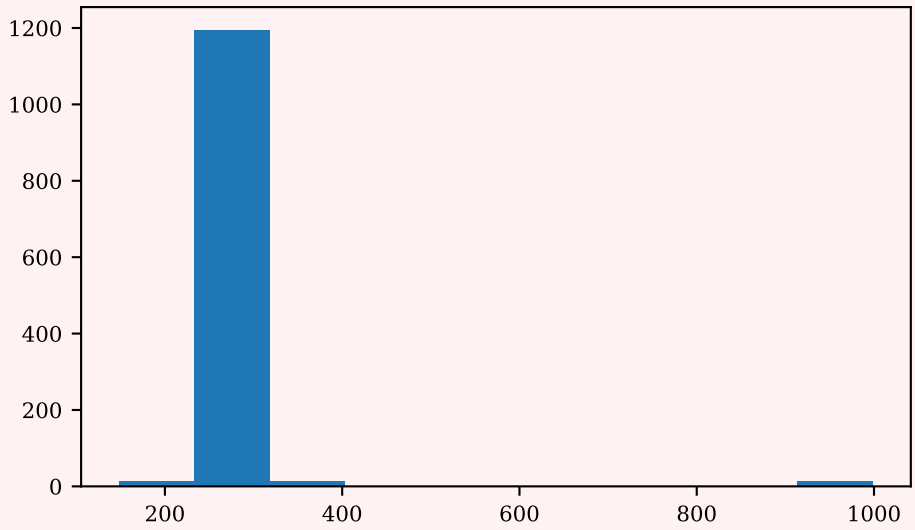
Le jeu de données contenu dans le fichier babies23.data est constitué de 1236 bébés décrits par 23 variables.

10 Charger le jeu de données et sélectionner les colonnes `wt`, `gestation`, `parity`, `age`, `ht`, `wt.1`, `smoke`, `ed` que l'on renommera en `bwt`, `gestation`, `parity`, `age`, `height`, `weight`, `smoke`, `education`. Lors du chargement, on pourra utiliser le séparateur `"\s +"` qui correspond un ou plusieurs espaces.

```
In [20]: babies = pd.read_csv("data/babies23.data", delimiter="\s+")
         babies = babies[["wt", "gestation", "parity", "age", "ht", "wt.1",
         ↪  "smoke", "ed"]]
         babies.columns = [
             "bwt",
             "gestation",
             "parity",
             "age",
             "height",
             "weight",
             "smoke",
             "education",
         ]
```

11 Faites l'histogramme des durées de gestation en jours. Que remarquez-vous ?

```
In [21]: plt.hist(babies.gestation)
         plt.show()
```




Les valeurs voisines de 1000 sont des valeurs absurdes qui correspondent vraisemblalement à des valeurs manquantes.

D'une manière générale dans ce jeu de données, lorsque la valeur de certains prédicteurs est inconnue une valeur prédéfinie est utilisée :

- Pour la colonne `bwt`, on utilise 999
- Pour la colonne `gestation`, on utilise 999
- Pour la colonne `age`, on utilise 99
- Pour la colonne `height`, on utilise 99
- Pour la colonne `weight`, on utilise 999
- Pour la colonne `smoke`, on utilise 9
- Pour la colonne `education`, on utilise 9

12 Remplacer toutes ces valeurs prédéfinies par `np.nan`.

```
In [22]: babies.loc[babies.bwt == 999, "bwt"] = np.nan
         babies.loc[babies.gestation == 999, "gestation"] = np.nan
         babies.loc[babies.age == 99, "age"] = np.nan
         babies.loc[babies.height == 99, "height"] = np.nan
         babies.loc[babies.weight == 999, "weight"] = np.nan
         babies.loc[babies.smoke == 9, "smoke"] = np.nan
         babies.loc[babies.education == 9, "education"] = np.nan
         babies.info()
```

```
Out [22]: <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 1236 entries, 0 to 1235
          Data columns (total 8 columns):
           #   Column      Non-Null Count  Dtype
          ---  ------      --------------  -----
           0   bwt         1236 non-null   float64
           1   gestation   1223 non-null   float64
           2   parity      1236 non-null   int64
           3   age         1234 non-null   float64
           4   height      1214 non-null   float64
           5   weight      1200 non-null   float64
           6   smoke       1226 non-null   float64
           7   education   1235 non-null   float64
          dtypes: float64(7), int64(1)
          memory usage: 77.4 KB
```

13 Pour la variable `smoke`, la documentation du jeu de données dit

```
smoke: does mother smoke?
0=never,
1=smokes now,
2=until current pregnancy,
3=once did, not now,
9=unknown
```

Recoder la variable `smoke` de manière à ce que la modalité « 1 » soit recodée en `Smoking` et les autres modalités en `NonSmoking`.

```
In  [23]: mask = babies.smoke == 1
          babies.loc[mask, "smoke"] = "Smoking"
          babies.loc[~mask, "smoke"] = "NonSmoking"
          babies.smoke = babies.smoke.astype("category")
          babies.smoke
Out [23]: 0       NonSmoking
          1       NonSmoking
          2          Smoking
          3       NonSmoking
          4          Smoking
                     ...
          1231    NonSmoking
          1232    NonSmoking
          1233       Smoking
          1234    NonSmoking
          1235    NonSmoking
          Name: smoke, Length: 1236, dtype: category
          Categories (2, object): ['NonSmoking', 'Smoking']
```

## 1.5 Dissimilarité et distance

On admet qu'une dissimilarité $d$ est une distance si et seulement si

$$S_{ijk} = 2d_{ij}^2 d_{ik}^2 + 2d_{ij}^2 d_{jk}^2 + 2d_{ik}^2 d_{jk}^2 - d_{jk}^4 - d_{ik}^4 - d_{ij}^4 \geq 0,$$

pour tout triplet $(i, j, k)$ d'éléments distincts appartenant à $\{1, \ldots, n\}$.

Ainsi, on peut tester si une dissimilarité est une distance en vérifiant le signe de la quantité

$$S_{\min} = \min_{i,\, j,\, k \text{ distincts}} S_{ijk}.$$

14 Écrire une fonction calculant la quantité $S_{ijk}$ puis une fonction calculant la quantité $S_{\min}$.

```
In [24]: def Sijk(d, i, j, k):
             return (
                 2 * d[i, j] ** 2 * d[i, k] ** 2
                 + 2 * d[i, j] ** 2 * d[j, k] ** 2
                 + 2 * d[i, k] ** 2 * d[j, k] ** 2
                 - d[i, j] ** 4
                 - d[i, k] ** 4
                 - d[j, k] ** 4
             )

         def S_min(d):
             N = d.shape[0]
             return min(
                 Sijk(d, i, j, k)
                 for i in range(N)
                 for j in range(N) if j > i
                 for k in range(N) if k > j
             )
```

On crée une dissimilarité quelconque avec le code suivant :

```
from numpy.random import default_rng
rng = default_rng()

N = 5
d = rng.exponential(scale=1, size=(N, N))
d = (d + d.T) / 2                                    # Symétrisation
d[range(N), range(N)] = 0                            # Mise à zéro de la diagonale
```

À partir d'une dissimilarité quelconque $d$, on définit une autre dissimilarité $d^\gamma$ comme suit

$$d^\gamma_{ij} = \begin{cases} d_{ij} & \text{si } i = j, \\ d_{ij} + \gamma & \text{sinon,} \end{cases}$$

avec $\gamma \geq -\min_{i \neq j} d_{ij}$.

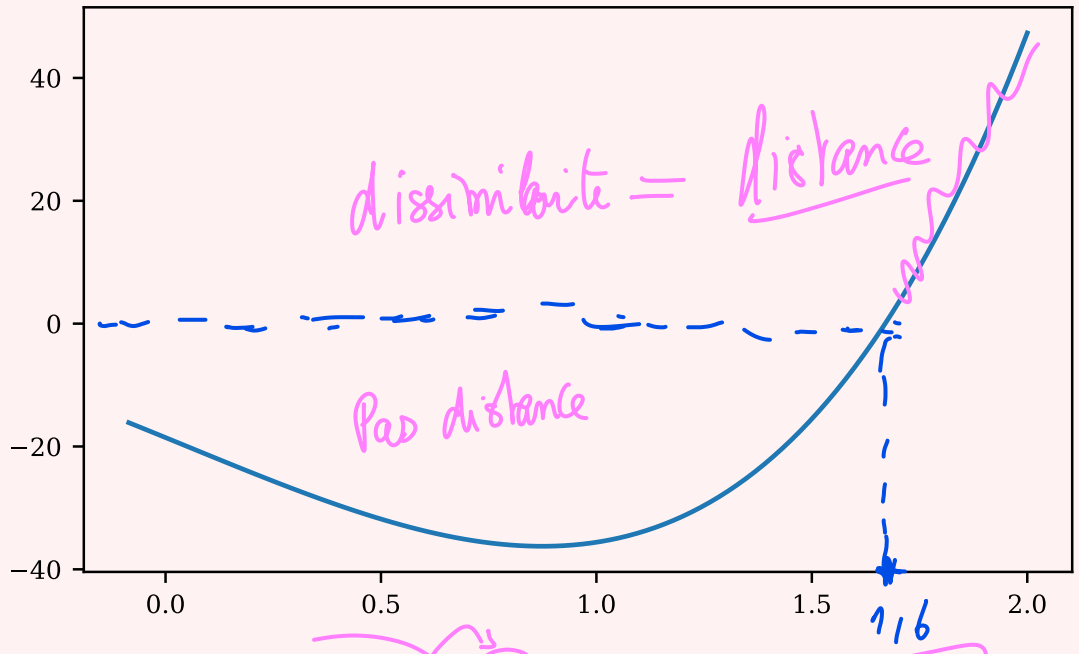
15 Montrer expérimentalement qu'il existe un seuil $\gamma_0$ tel que
- $d^\gamma$ est une distance pour $\gamma \geq \gamma_0$
- $d^\gamma$ est une dissimilarité pour $\gamma < \gamma_0$

```
In [25]: def ndiagadd(d, e):
             "Ajoute la quantité `e` hors diagonale"
             d = d + e
             d[range(N), range(N)] = 0
             return d

         N = 5

         from numpy.random import default_rng
         rng = default_rng(42)
         d = rng.exponential(scale=1, size=(N, N))
         d = (d + d.T) / 2
         d[range(N), range(N)] = 0

         val_min = -d[d > 0].min()
         gammas = np.linspace(val_min, 2, 100)
         S_mins = [S_min(ndiagadd(d, e)) for e in gammas]
         plt.plot(gammas, S_mins)
         plt.show()
```




On recherche ici le changement de signe de $S_{\min}$ qui a lieu aux alentours de $\gamma_0 \approx 1.6$.

16 Montrer expérimentalement que $\gamma_0 = \max_{i,j,k} d_{ij} - d_{ik} - d_{jk}$.

```
In [26]: max(
             d[i, j] - d[i, k] - d[k, j]
             for i in range(N)
             for j in range(N)
             for k in range(N)
         )
Out [26]: 1.670187046638726
```

On retrouve l'antécédent de 0 de la courbe précédente.

17 Démontrer que lorsque la distance est en plus euclidienne, on a $S_{ijk} = 4A^2$ avec $A$ l'aire du

triangle de longueur $d_{ij}$, $d_{ik}$, $d_{jk}$. On pourra utiliser la formule de Héron pour calculer l'aire d'un triangle avec la longueur de ses trois arêtes.

En notant les longueurs des trois arêtes, $a$, $b$ et $c$, l'aire $A$ vaut

$$A = \sqrt{p(p-a)(p-b)(p-c)} \qquad \text{(formule de Héron)}$$

avec $p$ le demi-périmètre. Ce qui donne

$$\begin{aligned} 4A^2 &= (a+b+c)(a+b-c)(a+c-b)(b+c-a) \\ &= 2a^2b^2 + 2b^2c^2 + 2a^2c^2 - a^4 - b^4 - c^4 \\ &= S_{ijk} \end{aligned}$$

On retrouve donc le fait que $S_{ijk}$ est une quantité positive dans le cas d'une distance euclidienne (ce qu'on a admis pour une simple distance).

18 En utilisant la quantité $S_{\min}$, montrer expérimentalement les résultats suivants :

Si $d$ est une distance alors les dissimilarités suivantes sont aussi des distances :

1. $d_{ij}^{(1/r)}$ avec $r \geq 1$,
2. $d_{ij}/(d_{ij} + c)$ avec $c > 0$.

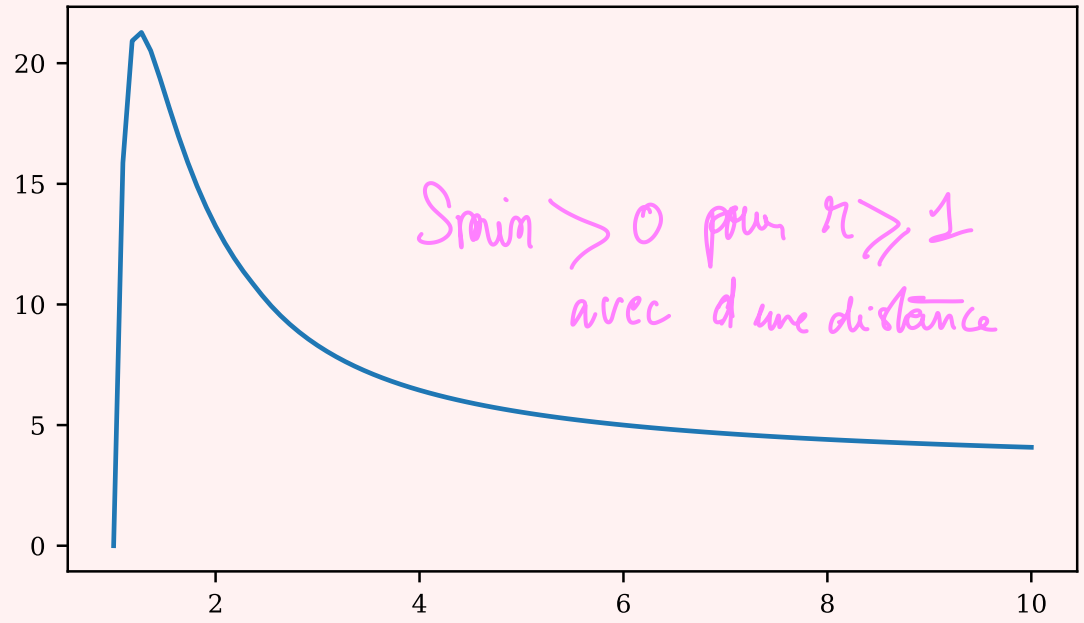
```
In [27]: N = 5

         from numpy.random import default_rng
         rng = default_rng(42)
         d = rng.exponential(scale=1, size=(N, N))
         d = (d + d.T) / 2
         d[range(N), range(N)] = 0

         # Génération d'une distance avec γ0
         gamma0 = max(
             d[i, j] - d[i, k] - d[k, j]
             for i in range(N)
             for j in range(N)
             for k in range(N)
         )
         d = ndiagadd(d, gamma0)

         # Calcul des S_min
         rs = np.linspace(1, 10, 100)
         Smins = [S_min(d**(1/r)) for r in rs]

         plt.plot(rs, Smins)
         plt.show()
```

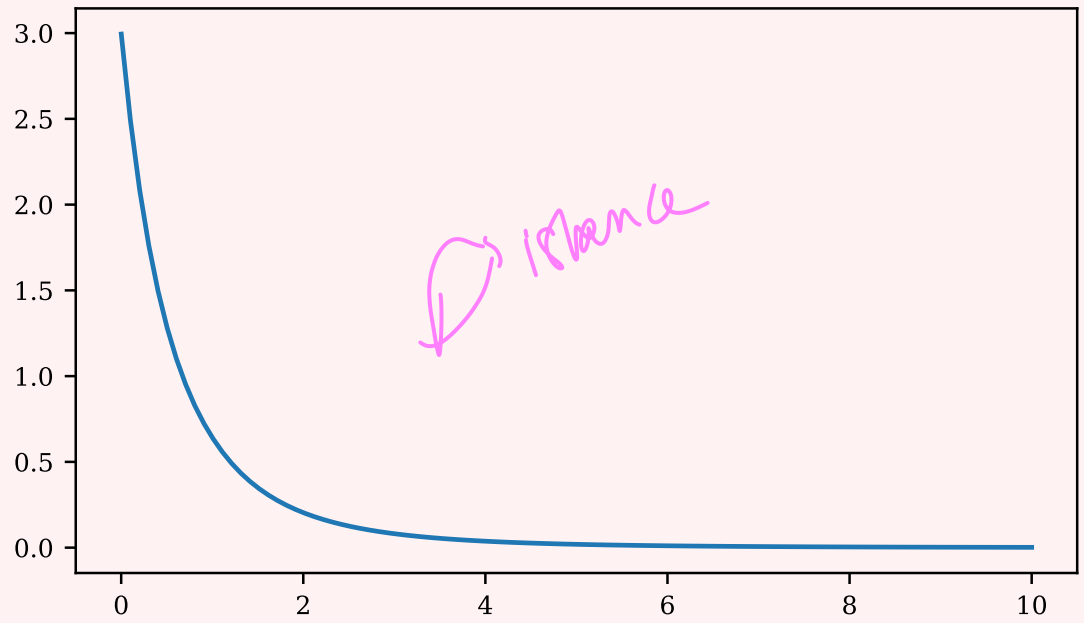



$S_{min} > 0$ pour $r \geq 1$ avec $d$ une distance

```
In [28]: cs = np.linspace(0.001, 10, 100)
         Smins = [S_min(d / (d + c)) for c in cs]

         plt.plot(cs, Smins)
         plt.show()
```

$c > 0$




Distance

Dans les deux cas, $S_{\min} \geq 0$ pour $r \geq 1$ et $c > 1$ ce qui montre que les dissimilarités correspondantes sont des distances.

# 2 Exercices

## 2.1 Proximités

19 On considère les matrices suivantes :

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 0 & -1 \\ 1 & 2 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 3 & 2 \\ 3 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix}, \quad C = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 0 & 2 & 3 \\ 2 & 0 & 1 \\ 4 & 1 & 0 \end{pmatrix};$$
$$E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad F = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad G = \begin{pmatrix} 10 & 1 & 3 \\ 6 & 9 & 2 \\ 5 & 0 & 8 \end{pmatrix}; \quad H = \begin{pmatrix} 10 & 4 & -1 \\ 4 & 10 & 5 \\ -1 & 5 & 10 \end{pmatrix}.$$

Lesquelles sont des matrices de proximité, et de quel type de proximité s'agit-il ?

A n'est pas une proximité car une entrée est négative.
B n'est ni une similarité ni une dissimilarité car la diagonale n'est pas dominante ni égale à zéro.
C est une matrice symétrique à diagonale nulle, c'est donc une dissimilarité. C'est en fait aussi une distance, une ultramétrique et une distance euclidienne.
D est une matrice à diagonale nulle mais pas symétrique.
E est une matrice de similarité car positive, symétrique à diagonale constante et dominante.
F est une matrice positive, symétrique, à diagonale constante et dominante : c'est donc une similarité.
G a une diagonale non constante, ce n'est donc pas une similarité.
H n'est pas une proximité car une entrée est négative.

## 2.2 Indice de Rand

On suppose que $X_1, \ldots, X_n$ sont $n$ caractéristiques binaires d'une population $\Omega$. On note $x_i$ la $i$-ième caractéristique de l'individu $x$, et on considère la similarité suivante entre deux individus $x$ et $y$ :

$$s(x, y) = \frac{a + d}{a + d + b + c},$$

où

$$a = \text{card}\{i, x_i = 1, y_i = 1\}, \quad d = \text{card}\{i, x_i = 0, y_i = 0\},$$
$$b = \text{card}\{i, x_i = 0, y_i = 1\}, \quad c = \text{card}\{i, x_i = 1, y_i = 0\}.$$

On pose $d = 1 - s$.

20 Montrer que

$$d(x, y) = \frac{b + c}{n}$$

Il suffit de remarquer que $a + b + c + d = n$.

21 Montrer que $d$ vérifie les propriétés de séparation et de symétrie.

L'expression $b + c$ est symétrique en $x$ et $y$ d'où la symétrie de $d$. Pour la séparation,

$$d(x, y) = 0 \iff b = c = 0$$
$$\iff x = y$$

22 On note

$$A = \text{card}\{i, x_i = 0, y_i = 0, z_i = 0\}, \quad B = \text{card}\{i, x_i = 0, y_i = 0, z_i = 1\},$$
$$C = \text{card}\{i, x_i = 0, y_i = 1, z_i = 0\} \quad D = \text{card}\{i, x_i = 0, y_i = 1, z_i = 1\},$$
$$E = \text{card}\{i, x_i = 1, y_i = 0, z_i = 0\}, \quad F = \text{card}\{i, x_i = 1, y_i = 0, z_i = 1\},$$
$$G = \text{card}\{i, x_i = 1, y_i = 1, z_i = 0\}, \quad H = \text{card}\{i, x_i = 1, y_i = 1, z_i = 1\}.$$

22 a Exprimer $d(x, y)$, $d(y, z)$ et $d(x, z)$ en fonction de $A$, $B$, $C$, $D$, $E$, $F$, $G$ et $H$.

On trouve

$$d(x, y) = \frac{C + D + E + F}{n}, \quad d(x, z) = \frac{B + D + E + G}{n}, \quad d(y, z) = \frac{B + C + F + G}{n}.$$

22 b En déduire que $d$ est une distance.

Des trois inégalités précédentes, on déduit

$$d(x, y) + d(y, z) - d(x, z) = 2\frac{D + E}{n} \geq 0,$$

d'où l'inégalité triangulaire. La proximité $d$ vérifie les propriétés de symétrie, de séparation et l'inégalité triangulaire, c'est donc une distance.

## 2.3 Ultramétrique

23 Montrer que la distance qui vaut tout le temps 1 sauf pour deux éléments identiques où elle vaut 0 est une distance ultramétrique.

La distance $d$ est donc définie par

$$d(x, y) = \begin{cases} 1 & \text{si } x \neq y, \\ 0 & \text{sinon.} \end{cases}$$

*Démonstration.* La distance est ultramétrique si elle vérifie les propriétés de symétrie, de séparation, et l'inégalité ultramétrique. Bien que l'énoncé suppose implicitement que $d$ est une distance (et vérifie donc les deux premières propriétés), nous les démontrerons à nouveau.

1. On a bien évidemment $d(x, y) = d(y, x)$ dès que $x = y$. Si $x \neq y$ on a alors $d(x, y) = 1 = d(y, x)$. La distance $d$ est donc symétrique.
2. Par définition, on a
$$d(x, y) = 0 \quad \Longleftrightarrow \quad x = y,$$
ce qui est exactement la propriété de séparation.
3. Soit $x$, $y$ et $z$ trois éléments de $\Omega$. Dès que les éléments ne sont pas tous distincts, l'inégalité ultramétrique est trivialement vérifiée. Dans le cas contraire, si $x$, $y$ et $z$ sont distincts, on a $d(x, y) = d(y, z) = d(x, z) = 1$ d'après la définition. L'inégalité ultramétrique est encore vérifiée.

En conclusion, la distance $d$ est ultramétrique. □

## 2.4 Ultramétrique et géométrie

24 Soit $\Omega$ un ensemble muni d'une ultramétrique $d$. Montrer que tout triangle dont les sommets sont des points de $\Omega$ est soit équilatéral, soit isocèle avec une petite base.

Soit $x, y, z$ trois points de $\Omega$. On pose $a = d(x, y)$, $b = d(y, z)$ et $c = d(x, z)$. Sans perte de généralité on peut supposer que $a \leq b \leq c$. En appliquant l'inégalité ultramétrique à deux reprises :

$$d(x, z) \leq \max(d(x, y), d(y, z)) \iff c \leq \max(a, b) = b$$
$$d(y, z) \leq \max(d(y, x), d(x, z)) \iff b \leq \max(a, c) = c$$

On a donc $b = c$. Si $a < b$, le triangle est isocèle avec une petite base. Si $a = b$, le triangle est équilatéral.