SY09 Printemps 2020
TP 01 — Manipulation de données

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/data.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):
         nom                       296 non-null object
         specialite                296 non-null object
         niveau                    296 non-null int64
         statut                    296 non-null object
         dernier diplome obtenu    290 non-null object
         note median               293 non-null float64
         correcteur median         293 non-null object
         note final                284 non-null float64
         correcteur final          284 non-null object
         note totale               284 non-null float64
         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)
```

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/data.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):
          nom                        296 non-null object
          specialite                 296 non-null category
          niveau                     296 non-null int64
          statut                     296 non-null category
          dernier diplome obtenu     290 non-null category
          note median                293 non-null float64
          correcteur median          293 non-null category
          note final                 284 non-null float64
          correcteur final           284 non-null category
          note totale                284 non-null float64
          resultat                   296 non-null category
          dtypes: category(6), float64(3), int64(1), object(1)
          memory usage: 15.4+ KB
```

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 « B » 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"])

          df1 = iris.variable.str.split("_", expand=True)
          df2 = iris.drop(columns=["variable"])
          pd.concat((df1, df2), axis=1)
Out [18]:          0       1    species  value
          0    sepal  length     setosa    5.1
          1    sepal  length     setosa    4.9
          2    sepal  length     setosa    4.7
          3    sepal  length     setosa    4.6
          4    sepal  length     setosa    5.0
          ..     ...     ...        ...    ...
          595  petal   width  virginica    2.3
          596  petal   width  virginica    1.9
          597  petal   width  virginica    2.0
          598  petal   width  virginica    2.3
          599  petal   width  virginica    1.8

          [600 rows x 4 columns]
```

(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]
```

4 Jeu de données babies

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

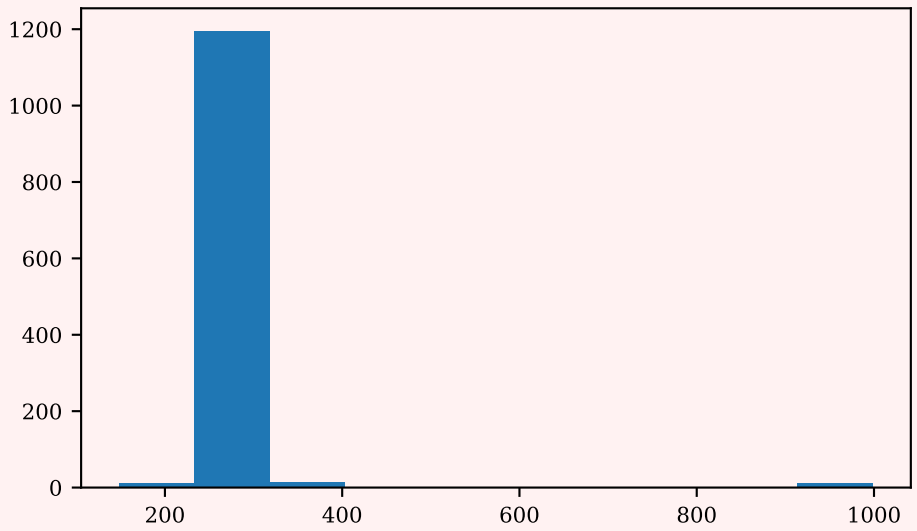
(10) Charger le jeu de données et sélectionner les colonnes `wt`, `gestation`, `parity`, `age`, `ht`, `wt.1`,

`smoke`, `ed` que l'on renommera en `bwt`, `gestation`, `parity`, `age`, `height`, `weight`, `smoke`, `education`.

```
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):
          bwt          1236 non-null float64
          gestation    1223 non-null float64
          parity       1236 non-null int64
          age          1234 non-null float64
          height       1214 non-null float64
          weight       1200 non-null float64
          smoke        1226 non-null float64
          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]
```