Traitement de données en tables

Nous allons travailler sur la base de données des accidents corporels de la circulation routière en France pour l'année 2020

Les données sont collectées par les forces de l'ordre (Gendarmerie et Police Nationale) et mises à disposition sur le site du ministère de l'Intérieur

`https://www.data.gouv.fr/fr/datasets/bases-de-donnees-annuelles-des-accidents-corpo`

Spécifications de la base

La base Etalab de données des accidents corporels de la circulation d'une année donnée, est répartie en 4 rubriques sous la forme pour chacune d'elles d'un fichier au format csv.

1. La rubrique **CARACTERISTIQUES** qui décrit les circonstances générales de l'accident
2. La rubrique **LIEUX** qui décrit le lieu principal de l'accident même si celui-ci s'est déroulé à une intersection
3. La rubrique **VEHICULES** impliqués
4. La rubrique **USAGERS** impliqués

Chacune des variables contenues dans une rubrique doit pouvoir être reliée aux variables des autres rubriques. Le n° d'identifiant de l'accident (Cf. "Num_Acc") présent dans ces 4 rubriques permet d'établir un lien entre toutes les variables qui décrivent un accident. Quand un accident comporte plusieurs véhicules, il faut aussi pouvoir relier chaque véhicule à ses occupants. Ce lien est fait par la variable id_vehicule.

La plupart des variables contenues dans les quatre fichiers précédemment énumérés peuvent contenir des cellules vides ou un zéro ou un point. Il s'agit, dans ces trois cas, d'une cellule non renseignée par les forces de l'ordre ou sans objet.

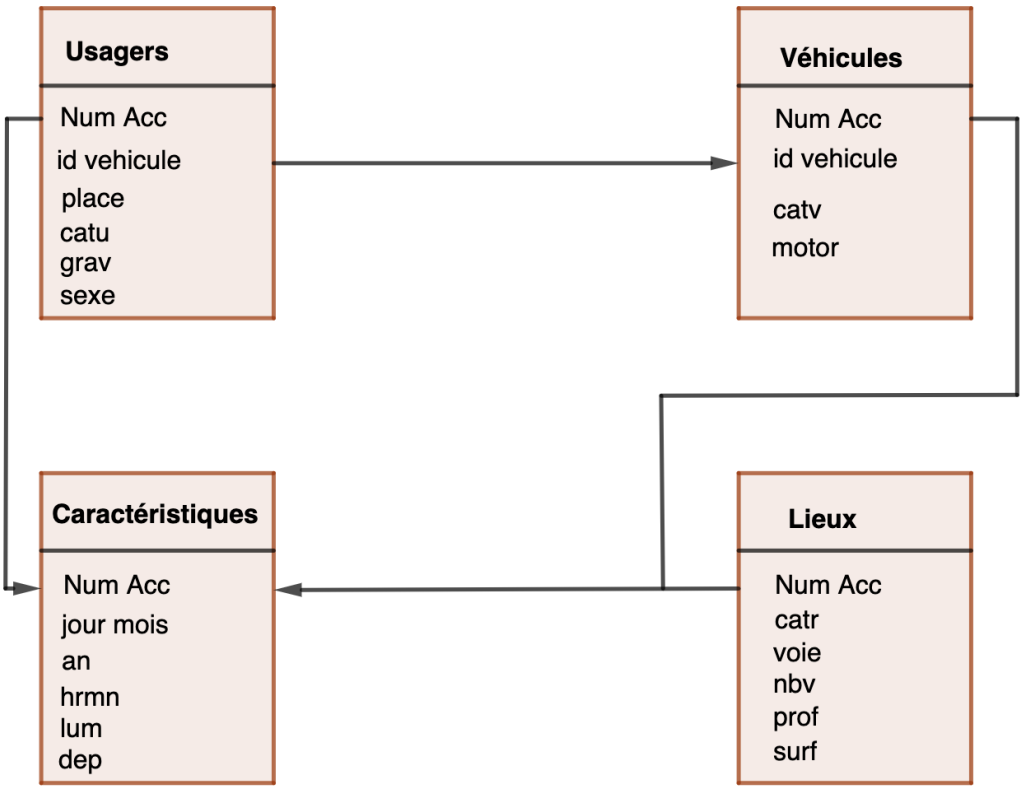
On peut faire une représentation "physique" de la base sous la forme de tables reliées entre elles par des flèches

Chaque accident est caractérisé par un identifiant unique `Num Acc` dans la table **Caractéristiques**

Toutes les autres tables **font référence à** cet identifiant

Chaque véhicule impliqué dans un accident est caractérisé par un identifiant unique `id vehicule`

La table **Usagers** fait référence à cet identifiant



1 Fichier csv et importation de table

Le format **csv** (pour comma separated values, valeurs séparées par des virgules) est un des formats de fichier utilisé pour échanger des données dans le Web

Regardons sur un exemple le fichier `usagers_2020.csv`.

Sur le site du ministère de l'Intérieur on peut visualiser le contenu du fichier et voici les premières lignes du fichier :

Num_Acc	id_vehicule	num_veh	place	catu	grav	sexe	an_nais	trajet
202000000001	154 742 274	B01	1	1	1	1	1983	5
202000000001	154 742 275	A01	1	1	3	1	1982	5
202000000002	154 742 273	A01	1	1	1	1	1997	1
202000000002	154 742 273	A01	10	3	4	1	1967	5
202000000003	154 742 271	B01	1	1	1	1	1985	0
202000000003	154 742 271	B01	4	2	3	1	2014	0

La première ligne est celle des **attributs** de la table : Num_Acc, id_vehicule, etc...

Pour bien comprendre le sens de chacun de ces attributs il convient de lire au préalable le document d'accompagnement `description-des-bases-de-donnees-onisr-annees-2005-a-20`

Ensuite chaque ligne est un usager impliqué dans un accident de la route

Comment traiter ces données ?

Nous allons ouvrir ces fichiers en lecture avec Python pour ensuite traiter ces données

Python a un module `csv` pour cela

Ensuite une fonction `cree_tab2D(nom_fichier)` ouvre un fichier .csv en lecture et transforme le contenu en une liste de liste

Pour pouvoir visualiser les premières lignes de la liste on a une fonction `affiche(nbLignes,tableau)`

```
from csv import reader

def cree_tab2D(nom_fichier:str)->list:
    """
    ouvre un fichier .csv en lecture
    renvoie les données sous la forme
    d'une liste de listes
    """
    with open(nom_fichier,"r") as fichier:
        return list(reader(fichier))

def affiche(nbLignes:int,tableau:list)->None:
    """
    affiche les nbLignes premières listes de tableau
    y compris la première contenant les attributs
    """
    for i in range(nbLignes):
        print(tableau[i])

usagers = []
usagers = creationTableau2D("usagers-2020.csv")
affiche(7,usagers)
```

On observe alors dans la console

```
['Num_Acc;"id_vehicule";"num_veh";"place";"catu";"grav";"sexe";"an_nais";"trajet";"secu1";"secu2";"secu
"actp";"etatp"']
['202000000001;"154\xa0742\xa0274";"B01";"1";"1";"1";"1";"1983";"5";"1";"0";" -1";" -1";" -1";" -1"']
['202000000001;"154\xa0742\xa0275";"A01";"1";"1";"3";"1";"1982";"5";"2";"6";" -1";" -1";" -1";" -1"']
['202000000002;"154\xa0742\xa0273";"A01";"1";"1";"1";"1";"1997";"1";"8";"0";" -1";" -1";" -1";" -1"']
['202000000002;"154\xa0742\xa0273";"A01";"10";"3";"4";"1";"1967";"5";"0";" -1";" -1";"3";"3";"1"']
['202000000003;"154\xa0742\xa0271";"B01";"1";"1";"1";"1";"1985";"0";"1";"0";" -1";" -1";" -1";" -1"']
['202000000003;"154\xa0742\xa0271";"B01";"4";"2";"3";"1";"2014";"0";"8";"0";" -1";" -1";" -1";" -1"']
```

1. Chaque liste ne contient qu'un seul élément, une seule chaîne de caractères entre ' '
2. Cette chaîne est une concaténation de chaînes avec le séparateur ;
3. Chaque chaîne est encadrée par des guillemets doubles "
4. La classe `str` a une méthode `split()` qui permet de "découper" une chaîne de caractères suivant un séparateur

   ```
   chaine = 'pomme;banane;orange'
   liste = chaine.split(';')
   ```

   Après exécution on observe dans la console

   ```
   >>> chaine
   ```

```
'pomme;banane;orange'

>>> liste
['pomme', 'banane', 'orange']
```

5. On pourrait traiter les données mais la documentation Python sur le module csv `https://docs.python.org/fr/3.6/library/csv.html#csv-contents` nous donne des outils pour que les données soient sous une forme plus exploitables
   On précise que

   (a) Le délimiteur est le symbole ";"

   (b) Par défaut l'attribut `quotechar` est '"', il sert à délimiter les caractères spéciaux comme le délimiteur
   Mettre l'attribut `doublequote` à False empêche l'acquisition de `quotechar`

```
def cree_tab2D_valide(nom_fichier:str)->list:
    """
    ouvre un fichier .csv en lecture
    renvoie les données sous la forme
    d'une liste de listes
    """
    with open(nomFichier,"r", newline='') as f:
        return list(reader(f, delimiter=';',
         doublequote=False))
```

Maintenant on observe dans la console

```
['Num_Acc', 'id_vehicule', 'num_veh', 'place', 'catu', 'grav', 'sexe', 'an_nais', 'trajet', 'secu1',
'secu2', 'secu3', 'locp', 'actp', 'etatp']
['202000000001', '154\xa0742\xa0274', 'B01', '1', '1', '1', '1', '1983', '5', '1', '0', ' -1', ' -1',
' -1', ' -1']
['202000000001', '154\xa0742\xa0275', 'A01', '1', '1', '3', '1', '1982', '5', '2', '6', ' -1', ' -1',
' -1', ' -1']
['202000000002', '154\xa0742\xa0273', 'A01', '1', '1', '1', '1', '1997', '1', '8', '0', ' -1', ' -1',
' -1', ' -1']
['202000000002', '154\xa0742\xa0273', 'A01', '10', '3', '4', '1', '1967', '5', '0', ' -1', ' -1', '3'
, '3', '1']
['202000000003', '154\xa0742\xa0271', 'B01', '1', '1', '1', '1', '1985', '0', '1', '0', ' -1', ' -1',
' -1', ' -1']
['202000000003', '154\xa0742\xa0271', 'B01', '4', '2', '3', '1', '2014', '0', '8', '0', ' -1', ' -1',
' -1', ' -1']
```

2 Recherche dans une table

Maintenant que nous avons la liste des données on peut vouloir rechercher un certain nombre d'informations

Par exemple on aimerait afficher tous les usagers ayant moins de 1 an ce qui en 2020 correspond à une date de naissance supérieure ou égale à 2019

```
ANNEE_REF = 2020

def afficher_age(seuilAge,tableau):

    for i in range(1,len(tableau)):
        if ANNEE_REF - int(tableau[i][7]) <= seuilAge:
            print(tableau[i])

afficher_age(1,usagers)
```

On pourrait affiner ensuite et rechercher les accidents mortels parmi les très jeunes enfants

```
def afficher_age_grav(seuilAge,grav,tableau):

    for i in range(1,len(tableau)):
        if ANNEE_REF - int(tableau[i][7]) <= seuilAge and\
        tableau[i][5] == grav:
            print(tableau[i])

afficher_age_grav(1,'2',usagers)
```

Ces opérations de recherche ne conservent qu'un certain nombre de lignes d'une table on parle alors de **sélection**

Au lieu de supprimer des lignes on peut vouloir ne garder que certaines colonnes on parle alors de **projection**

Par exemple on veut garder uniquement les colonnes de l'attribut `grav` et de l'attribut `an_nais`, les colonnes d'indice 5 et 7

```
def projection(tableau):
    nouvelle_table = []
    for i in range(1,len(tableau)):
        nouvelle_table.append(tableau[i][5:6] +  tableau[i][7:8])
return nouvelle_table
```

Dans cette nouvelle table il risque d'avoir des doublons. Si on veut éliminer les doublons on va trier dans l'ordre croissant un des attributs (voir plus loin)

Par exemple On peut aussi vouloir faire des **statistiques sur la table** :

Par exemple compter le nombre d'accidents mortels en 2020, pour cela on va définir une fonction plus générale qui compte le nombre de lignes dans une table vérifiant une certaine propriété

```
def somme(indice_champ,valeur,tableau):
    nb = 0
    for i in range(1,len(tableau)):
        if tableau[i][indice_champ] == valeur :
            nb += 1
    return nb
```

A l'exécution

```
print(somme(5,'2',usagers))
```

donne 2780 morts sur la route en 2020

```
print(somme(4,'3',usagers))
```

donne 8243 piétons accidentés sur la route en 2020

3 Tri dans une table

Une propriété fondamentale d'une base de données est :

"Dans une table il ne doit pas y avoir de doublons" d'où l'importance de l'identifiant

Pour éliminer les doublons on trie la table suivant un attribut dans l'ordre croissant, ce qui fera apparaître les doublons puis on ne garde qu'un exemplaire par doublon

Par exemple supposons que l'on veuille trier les accidents dans l'ordre croissant suivant la date de naissance

- Python dispose d'une fonction `sorted(liste)` qui tri une liste en renvoyant une nouvelle liste
- La classe `list` a une méthode `sort()` qui permettent de trier sur place la liste avec une complexité temporelle dans le pire des cas de l'ordre de $n\ln(n)$ où n est la taille de la liste

```
>>> liste = [1,4,1,4]
>>> sorted(liste)
[1, 1, 4, 4]
>>> liste
[1, 4, 1, 4]
>>> liste.sort()
>>> liste
[1, 1, 4, 4]
```

Si on veut trier dans l'ordre décroissant on utilise l'option `reverse`

```
>>> liste = [1,4,1,4]
>>> sorted(liste,reverse=True)
[4, 4, 1, 1]
>>> liste
[1, 4, 1, 4]
>>> liste.sort(reverse=True)
>>> liste
[4, 4,1,1]
```

C'est un tri basé sur des comparaisons. Par défaut c'est l'ordre sur les nombres et l'ordre lexicographique sur les chaînes de caractères

Cependant on peut choisir d'autres critères de comparaison, par exemple en utilisant une clé qui est une fonction qui prend en argument un élément de la liste ici une chaîne

de caractères renvoie une nouvelle chaîne de caractères en minuscule pour ensuite faire la comparaison avec l'ordre lexicographique

```
>>> liste = ['Banane','pomme','ananas','A']
>>> sorted(liste,key=str.lower)
['A', 'ananas', 'Banane', 'pomme']
```

Comment trier une liste de listes ?

Une liste de listes est une liste où chaque liste-élément peut être vue comme une chaîne de caractères pour pouvoir utiliser une clé comme précédemment cependant avec une nuance permettant de faire la comparaison

Ainsi la clé sera une fonction qui renvoie un des éléments de la liste qui suit soit l'ordre des nombres soit l'ordre lexicographique

Par exemple supposons que l'on ait une liste de tuples composé d'un nom et d'un numéro de téléphone de type str

On veut trier cette liste en fonction de l'ordre lexicographique sur les noms

```
def cle(x:tuple)->str:
    return x[0]

liste = [('Einstein','0607080910'),('Zweistein','0608070910'),
('Dreistein','06090807')]
nouvelle_liste = sorted(liste,key=cle)
print(nouvelle_liste)
```

A l'exécution on obtient

```
>>>  [('Dreistein','06090807'), ('Einstein','0607080910'),
 ('Zweistein','0608070910')]
```

Si on veut trier la table `usagers` suivant l'année de naissance

```
def annee_naissance(usager):
    return usager[7]
usagers = []
usagers = creationTableau2D("usagers-2020.csv")
tri_usagers_age = sorted(usagers[1:],key=annee_naissance)
```

La méthode `sort()` a les même options `key` et `reverse` que la fonction `sorted()`

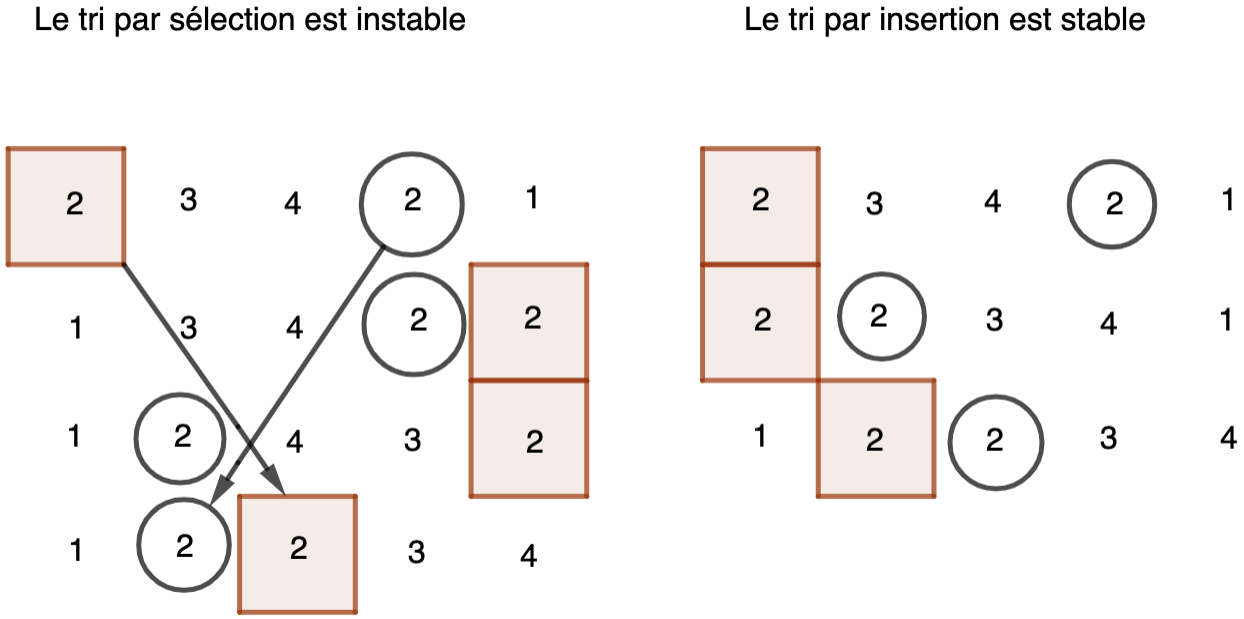
3.1 Stabilité d'un tri

On veut trier le tableau T = [$\boxed{2}$, 3, 4, ②, 1] par sélection et par insertion

Sur l'image suivante on constate :

1. Les positions relatives des valeurs 2 se sont inversées après le tri par sélection. On dit alors que le tri par sélection est **instable**
2. Les positions relatives des valeurs 2 ne se sont pas inversées après le tri par insertion. On dit alors que le tri par insertion est **stable**

Les fonctions `sorted()` *et* `sort()` *ont un algorithme de tri stable*



Exemple d'application : Imaginons une liste de tuples composés d'un nom (une chaîne de caractères) et d'un poids en kg un entier

Supposons que l'on trie d'abord cette liste suivant le poids, puis suivant le nom

Si le tri est stable alors on ne change pas les positions relatives des personnes de même poids lorsqu'on va effectuer le tri suivant l'ordre lexicographique sur les noms

4 Fusion de tables

Voici deux tables **T1** et **T2** ayant un attribut en commun B

On aimerait fusionner les deux tables suivant cet attribut en commun

A	B
1	2
4	5
2	1
3	1
1	1

B	C
1	2
2	5
2	1
3	1
1	3

1. Une première façon de faire est de parcourir la table T1, et pour chaque élément de T1 parcourir la table T2 et regarder s'il existe un élément de T2 ayant un attribut courant avec l'élément de T1, si c'est le cas créer une liste fusionnant les éléments de T1 et T2

   Dans le cas particulier où T1 est la liste `usagers` et T2 la liste `caracteristiques`, on va fusionner chaque usager caractérisé par le numéro d'accident avec les colonnes de 1 à 6 de caractéristiques ce qui donne en Python

```
def fusion(tableau1,tableau2 ):
    nouvelle_table = []
    for i in range(1,len(tableau1)):
        for j in range(1,len(tableau2)):
            if tableau1[i][0] == tableau2[j][0]:
                nouvelle_table.append(tableau1[i] +\
                 tableau2[i][1:7])
                break
    return nouvelle_table
```

La complexité en temps de cette approche est proportionnelle au produit des longueurs des tables T1 et T2 et puisque les longueurs des listes sont grandes il ne faut surtout pas exécuter cette fonction

2. La deuxième approche est de trier les deux tables suivant cet attribut en commun, à l'aide d'un tri stable (insertion par exemple)

A	B
2	1
3	1
1	1
1	2
4	5

B	C
1	2
1	3
2	5
2	1
3	1

Ensuite on va parcourir ensemble T1 et T2 pour obtenir

A	B	C
2	1	2
2	1	3
3	1	2
3	1	3
1	1	2
1	1	3
1	2	5
1	2	1

Dans le cadre de la fusion des tables `usagers` et `caracteristiques` l'algorithme de fusion est plus simple

```
def fusion2(tableau1,tableau2):
    nouvelle_table = []
    i1,i2 = 1,1
    while i1 < len(tableau1):
        if tableau1[i1][0] == tableau2[i2][0]:
            nouvelle_table.append(tableau1[i1] +\
                 tableau2[i2])
            i1 += 1
        else:
            i2 += 1
    return nouvelle_table
```

5 Exercices

Ex 1

Aller sur le site de MétéoFrance

`https://donneespubliques.meteofrance.fr/?fond=produit&id_produit=95&id_rubrique=32`

1. Télécharger le fichier des relevés du mois de Janvier 2022
2. Dans un premier temps visualiser les données avec `cree_tab2D(nom_fichier)` et `affiche(nbLignes,tableau)`
3. Dans un second temps visualiser les données avec `cree_tab2D_valide(nom_fichier)`

Ex 2

Parmi les 2780 morts sur la route quel est le nombre de piétons ? de passagers ?

Ex 3

Parmi les conducteurs impliqués dans des accidents mortels on aimerait avoir la distribution des fréquences selon les classes d'âges [18,30] ,[30,40], [40,50] etc...

Ex 4

Sur quelle balise en mer a-t-on mesuré la hauteur de vague maximale pour le mois de Janvier 2022 ?

Ex 5

Voici une table **T**

Noms	Quantité (kg)	Prix au Kg (€)
Pomme de terre	1,5	2,5
Courgette	0,5	3
Aubergine	0,7	3,5
Bananes	1	1,5

1. Sélectionner les lignes dont l'attribut Prix au kg est supérieur à 2 €
2. Faire une projection sur les colonnes Noms et Prix au kilo
3. Si on veut afficher les noms et produits dont le prix au kg est supérieur à 2 € peut on commuter les opérations de sélection et de projection ?

Ex 6

Voici une table **T**

A	B	C
1	2	3
4	5	1
2	1	4
3	1	2
1	2	5

1. Trier cette table selon l'attribut A par sélection, puis selon l'attribut C par sélection. Qu'observe-t-on ?
2. Trier cette table selon l'attribut A par insertion, puis selon l'attribut C par insertion. Qu'observe-t-on ?

Ex 7

Voici une table **T**

A	B	C
1	2	3
4	5	1
2	1	4
3	1	2
1	2	5

Faire une projection en ne conservant que les colonnes A et B. Obtient-on une table sans doublons ?

Proposer un algorithme pour éliminer les doublons s'il y en a

Ex 8

Voici deux tables **T1** et **T2** ayant un attribut en commun B

Fusionner les deux tables suivant cet attribut en commun

A	B
3	0
4	5
0	1
3	2
1	1

B	C
1	4
1	0
1	1

Ex 9

Fusionner les tables Usagers et Caractéristiques

1. Combien y-a-t-il d'accidents mortels en 2020 dans le département de L'Essonne ? En Yvelines ?
2. Combien d'accidents mortels en 2020 dans votre commune ? Où ?
3. Combien d'accidents mortels en 2020 de nuit ?

Ex 10

Fusionner les tables Usagers et Véhicules

1. Combien d'accidents mortels en 2020 en moto ? en vélo ?en poids lourds ?
2. La proportion des voitures "puissantes" dans les accidents mortels est-elle plus ou moins élevée que celle des voitures "moyennes" ?
3.