Traitement des données en tables

qkzk

2020/07/31

Dans cette partie nous allons étudier les opérations de tri selon une colonne, la notion de domaine de valeur et la fusion de tables.

1. Tri d'une table selon une colonne

- Puisqu'une table est représentée par une liste, on peut la trier en utilisant la fonction de tri `sorted` qui dispose d'un argument `key` permettant de préciser selon quel critère une liste doit être triée (qui doit être une fonction de variables les objets à trier). Un troisième argument, `reverse` (un booléen), permet de préciser si l'on veut le résultat par ordre croissant (par défaut) ou décroissant (en précisant `reverse=True`).

  Examinez soigneusement les exemples suivants :

```
>>> ma_liste = [10, 3, 71, 96]
>>> sorted(ma_liste)
[3, 10, 71, 96]
>>> sorted(ma_liste, reverse=True)
[96, 71, 10, 3]
>>> mes_couples = [('a', 3), ('d', 2), ('c', 5), ('b', 1)]
>>> sorted(mes_couples)                        # tuples triés selon leur premier élément
[('a', 3), ('b', 1), ('c', 5), ('d', 2)]
>>> sorted(mes_couples, key=lambda x: x[1]) # tuples triés selon leur second élément
[('b', 1), ('d', 2), ('a', 3), ('c', 5)]
```

- On peut alors créer une fonction `tri` qui trie n'importe quel table en donnant l'attribut choisi pour le tri et en précisant si l'on veut obtenir le tri dans l'ordre décroissant.

```
def tri(table, attribut, decroit=False):
    def critere(ligne):
        return ligne[attribut]
    return sorted(table, key=critere, reverse=decroit)
```

  Exemple : Pour trier dans l'ordre décroissant la table `Table1` selon les notes de maths, on fera :

```
>>> tri(Table1, 'Maths', True)
```

  Qui donne :

	Nom	Maths	Info	Anglais
0	Zoé	19	17	15
1	Joe	16	18	17
2	Max	14	13	19

En bref : *Lorsque l'on traite de grandes quantités de données, celles-ci sont souvent réparties dans plusieurs tables. On est alors amené à regrouper des données dans une nouvelle table. Cette opération s'appelle jointure de tables*

2. Fusion de deux tables pour un même attribut

- On veut fusionner deux tables selon un attribut commun. On va sélectionner dans chaque table la ligne ayant la même valeur pour l'attribut choisi.
- Reprenons le tableau `Table1` des parties précédentes :

	Nom	Anglais	Info	Maths
0	Joe	17	18	16
1	Zoé	15	17	19
2	Max	19	13	14

Définissons une seconde table, `Table2` donnant l'âge et le courriel de certains élèves :

	Nom	Age	Courriel
0	Joe	16	joe@info.fr
1	Zoé	15	zoe@info.fr

- On voudrait regrouper les données des deux tables. Elles ont l'attribut `Nom` en commun. On veut obtenir la table suivante :

	Nom	Age	Courriel	Anglais	Info	Maths
0	Joe	16	joe@info.fr	17	18	16
1	Zoé	15	zoe@info.fr	15	17	19

On chosit d'exclure la ligne concernant `Max` car il n'est pas présent dans la seconde table.

On effectuera la jointure selon le nom avec la commande :

```
>>> jointure(Table1, Table2, 'Nom')
```

On utilise ici une fonction `jointure` définie pour l'occasion, comme celle fournie plus bas.

3. La fusion de deux tables pour des attributs différents

- Cependant dans certaines tables, l'attribut commun peut avoir une autre appellation. Par exemple la seconde table peut aussi exister sous la forme :

	Name	Age	Email	English	CS	Maths
0	Joe	16	joe@info.fr	17	18	16
1	Zoé	15	zoe@info.fr	15	17	19

- Cette fois, on précisera l'attribut de la seconde table :

```
>>> jointure(Table1, Table2, 'Nom', 'Name')
```

4. Exemple de fonction effectuant une jointure

- Voici une proposition de code

```
from copy import deepcopy
def jointure(table1, table2, cle1, cle2=None):
    if cle2 is None:
        cle2 = cle1
    new_table = []
```

```
6       for line1 in table1:
7           for line2 in table2:
8               if line1[cle1] == line2[cle2]:
9                   new_line = deepcopy(line1)
10                  for cle in line2:
11                      if cle != cle2:
12                          new_line[cle] = line2[cle]
13                  new_table.append(new_line)
14      return new_table
```

- Ligne 3 : par défaut, les clés de jointure portent le même nom
- Ligne 5 : la future table créée, vide au départ
- Ligne 8 : on ne considère que les lignes où les cellules de l'attribut choisi sont identiques.
- Ligne 9 : on copie entièrement la ligne de `table1`
- Ligne 10 : on copie la ligne de `table2` sans répéter la cellule de jointure.

À noter : *En terminale, vous découvrirez la gestion des bases de données relationnelles, notamment à l'aide du langage SQL. Dans ce langage, la jointure donnée en exemple s'écrira :*

```
SELECT nom
FROM Table1 JOIN Table2
ON Table1.Nom = Table2.Nom
```

5. Domaine de valeur

Un problème subsiste : que faire si les données ont des formats différents ?

On considère les deux tables `seconde` et `première` dont voici des extraits :

seconde

Nom	Prénom	moyenne	avis
Duchmol	Robert	3	nul
Lemeilleur	Franky	19	moyen
Poivre	Jacques	12	méchant

première

Nom	Prénom	DS1	DS2	moyenne
Durant	Martine	1	5	3
Philonard	Albert	16	12	14
Pommier	Fanny	13	14	12

domaine de valeur :

C'est l'ensemble des champs qui sont communs à plusieurs tables.

Ici : Nom, Prénom, moyenne

Admettons qu'on ait importé ces tables dans des variables `seconde` et `premiere`.

Avant de réaliser la fusion, il faut ajouter une étape, de formatage des valeurs, consistant à donner à chaque enregistrement le même format qu'aux autres.

Par exemple, pour formater un enregistrement :

```
def formater_eleve(eleve):
  return {"nom": eleve["nom"],
          "prenom": eleve["prenom"],
          "moyenne":eleve["moyenne"]}
```

On doit parcourir la liste et en créer une nouvelle, avant de fusionner :

```
premieres_formatees = [formater_eleve(eleve) for eleve in premiere]
secondes_formatees = [formater_eleve(eleve) for eleve in seconde]
```

Chaque série ayant un format particulier, il peut-être nécessaire de les traiter individuellement.

Enfin, on peut tout regrouper et obtenir l'ensemble avec un format cohérent :

```
eleves_formates = secondes_formatees + premieres_formatees
```

On obtient alors une seule table, disposant d'un format cohérent, avec tous les élèves.