Cours-TP1

DataFrame, Centrer-réduire

Statistique pour la SAé 2.04

Tiphaine Jézéquel

2023-2024

Listes *vs* Numpy.Array

Numpy.Array : une structure supplémentaire sur certaines listes

Structure qui s'applique sur

- des listes d'objets de même type

```
In [2]: liste1=[2,3,5]

In [3]: liste1
Out[3]: [2, 3, 5]

In [4]: array1=np.array(liste1)

In [5]: array1
Out[5]: array([2, 3, 5])
```

- ou des listes de listes de même longueur

```
In [9]: liste3=[[2,1],[3,5]]

In [10]: liste3
Out[10]: [[2, 1], [3, 5]]

In [11]: np.array(liste3)
Out[11]:
array([[2, 1],
       [3, 5]])
```

↪ ne s'applique pas sur une liste de listes de longueurs différentes :

```
In [6]: liste2=[[2],[3,5]]

In [7]: liste2
Out[7]: [[2], [3, 5]]

In [8]: array2=np.array(liste2)
C:\Users\tiphaine\AppData\Local\Temp/ipykernel_
VisibleDeprecationWarning: Creating an ndarray
```

Retour simple au type liste

```
In [17]: array1.tolist()
Out[17]: [2, 3, 5]
```

```
In [18]: array3.tolist()
Out[18]: [[2, 1], [3, 5]]
```


Point commun : slicing, mutabilité

Accéder à un élement

- liste ou np.array à 1 dimension :

```
In [22]: liste1[0]
Out[22]: 2

In [23]: array1[0]
Out[23]: 2
```

- liste de listes ou np.array à 2 dimensions :

```
In [24]: liste3[0][1]
Out[24]: 1

In [25]: array3[0][1]
Out[25]: 1

In [26]: array3[0,1]
Out[26]: 1
```

Modifier un élement

- liste ou np.array à 1 dimension :

```
In [27]: liste1
Out[27]: [2, 3, 5]

In [28]: liste1[0]=10

In [29]: liste1
Out[29]: [10, 3, 5]
```

- liste de listes ou np.array à 2 dimensions :

```
In [31]: array3[0,1]=10

In [32]: array3
Out[32]:
array([[ 2, 10],
       [ 3,  5]])
```

Différence : les opérations + et * sur les listes et les np.array

Opérations +, * sur les listes

Pour les listes, + et * se réfèrent à la concaténation :

```
In [34]: liste1
Out[34]: [10, 3, 5]

In [35]: liste1+liste1
Out[35]: [10, 3, 5, 10, 3, 5]
```

```
In [36]: liste1*2
Out[36]: [10, 3, 5, 10, 3, 5]

In [37]: liste1*3
Out[37]: [10, 3, 5, 10, 3, 5, 10, 3, 5]
```

Opérations +, * sur les np.array

Pour les np.array, + et * sont les opérations numériques sur les éléments :

```
In [38]: array1
Out[38]: array([2, 3, 5])

In [39]: array1+array1
Out[39]: array([ 4,  6, 10])
```

```
In [40]: array1*2
Out[40]: array([ 4,  6, 10])

In [41]: array1*3
Out[41]: array([ 6,  9, 15])
```


Quelques conséquences de ces différences

Création d'une liste / d'un numpy.array

```
In [54]: liste4=[]
    ...: for i in range(4):
    ...:     liste4 += [i]

In [55]: liste4
Out[55]: [0, 1, 2, 3]
```

```
In [56]: array4=np.zeros(4)

In [57]: array4
Out[57]: array([0., 0., 0., 0.])

In [58]: for i in range(4):
    ...:     array4[i] = i

In [59]: array4
Out[59]: array([0., 1., 2., 3.])
```

- Les opérations numériques, donc les fonctions statistiques, s'utilisent de manière + sûre sur des np.array.
- La concaténation de np.array est + compliquée, il faut utiliser la fonction `np.concatenate`.

La structure Pandas.DataFrame

C'est une structure qui se rajoute sur un np.array pour créer un tableau avec des noms de lignes et de colonnes :

	Maths	Prog	Com
Albert	1	2	3
Maria	12	13	10
Zoe	15	12	13

C'est un objet qui se trouve dans la librairie Python Pandas :

```
import pandas as pd
```

Créer un DataFrame à partir d'un fichier .csv

Pour un fichier `MonFichier.csv` se trouvant dans le même dossier que le fichier Python utilisé :

```
MonDataFrame=pd.read_csv("MonFichier.csv")
```


• De np.array à DataFrame et inversement

Pour créer un DataFrame à partir d'un np.array, il faut créer la liste des noms des lignes, et la liste des noms des colonnes, puis les combiner ainsi :

```
In [71]: array5
Out[71]:
array([[ 1,  2,  3],
       [12, 13, 10],
       [15, 12, 13]])

In [72]: noms_lignes=['Albert','Maria','Zoe']

In [73]: noms_colonnes=['Maths','Prog','Com']

In [74]: df5=pd.DataFrame(data=array5,index=noms_lignes,columns=noms_colonnes)

In [75]: df5
Out[75]:
        Maths  Prog  Com
Albert      1     2    3
Maria      12    13   10
Zoe        15    12   13
```

Pour revenir à la structure np.array à partir d'un DataFrame, c'est simple :

```
In [76]: array5=df5.to_numpy()

In [77]: array5
Out[77]:
array([[ 1,  2,  3],
       [12, 13, 10],
       [15, 12, 13]])
```

Statistique pour la SAé 2.04 - TP1
DataFrame - Centrer-réduire

Les données Sangliers.csv

Dans ce TP et le suivant, on va travailler sur les données récoltées en 2019 par les étudiants de DUT Jean-Baptiste Le Chanu, Gwendal Houssaye et Théo Leveque dans le cadre de leur projet de Statistique.

Leur problématique était la suivante :

Est-ce que l'évolution des dégâts causés par les sangliers à une raison ?

Et pour cela ils avaient trouvé, pour les années 2000 à 2015, les données suivantes :

1.1) Nombre de sanglier prélevé en France par année
1.2) Indemnisation en euro des dégâts causés par les sangliers en France
1.3) Consommation de viande de porc dans l'OCDE
1.4) Nombre de chasseur en France

1. Télécharger sur Moodle le fichier Sangliers.csv.

Création d'un DataFrame à partir d'un fichier .csv

Dans les TP de Statistiques pour la SAé, on utilisera la librairie Pandas pour importer nos fichiers .csv et manipuler des DataFrames. Commencer par l'importer, ainsi que Numpy :

```
import numpy as np
import pandas as pd
```

2. Importer le fichier `Sangliers.csv` dans Python sous la forme d'un DataFrame `SangliersDF`.
3. Dans la console, tester les commandes suivantes. Que fait chacune d'elles ?
   ```
   SangliersDF.columns
   SangliersDF.index
   SangliersDF.shape
   SangliersDF['Annees']
   ```
4. Quels sont les noms donnés aux lignes de ce DataFrame ?

Création d'un DataFrame de toutes pièces

On a déjà créé un premier DataFrame `SangliersDF` en important notre fichier .csv avec Pandas. On a vu dans la partie cours qu'on peut créer un DataFrame à partir d'un tableau numpy.array.

5. Créer un DataFrame `MonDataFrame` similaire à celui du cours, en remplaçant les noms des 3 lignes par le vôtre et ceux de vos voisin.es les plus proches dans cette salle, et en remplissant les colonnes par l'Age, l'Année de Naissance et l'Année d'obtention du Brevet des collèges.

DataFrame et Numpy.Array

Comme on a travaillé avec des tableau np.array en Python jusque là, il sera intéressant de savoir passer du format Data Frame au format np.array et inversement.
Dans les questions suivantes, on va s'entrainer à passer d'un DataFrame à des np.array et de np.array à DataFrame. L'objectif de ces questions est de créer un DataFrame `Sangliers0DF` dans lequel le nom des lignes sera les années.

6. A partir du DataFrame `SangliersDF`, créer un np.array `SangliersAr` contenant toutes les valeurs numériques du DataFrame `SangliersDF`.

7. En utilisant des commandes de slicing, créer un np.array `AnneesAr` contenant uniquement les années, puis un autre np.array `Sangliers0Ar` contenant toutes les données sauf les années.

   Indication : on pourra s'aider de l'Aide-Mémoire Python présent dans la section "Tutos Python" de cette ressource sur Moodle.

8. A l'aide d'une des commandes vues dans la question 3 et de la commande `.to_numpy()`, créer un np.array `ColAr` contenant les noms des colonnes de `SangliersDF`.

9. En utilisant `Sangliers0Ar`, `AnneesAr` et `ColAr`, créer un DataFrame `Sanglier0DF` contenant les données sangliers sauf les années, et ayant les années comme noms de lignes.

Fonctions Numpy de statistiques, Centrer-réduire

Dans les TP de la Ressource de Statistiques R2.08, on a créé des fonctions pour calculer la moyenne, la variance, le coefficient de corrélation etc.

La librairie Numpy contient en fait des fonctions toutes faites pour faire ces calculs. Vous trouverez une liste de ces fonctions sur le site *numpy.org* en cherchant la page *Statistics* dans la partie Documentation.

10. A l'aide des fonctions de statistiques de Numpy, calculer la moyenne et l'écart-type de chaque colonne du tableau `Sangliers0Ar`.

    Possible en une seule commande : regardez la doc des fonctions numpy !

11. Créer une fonction `Centreduire`, qui prend en paramètre un tableau np.array `T` et renvoie un nouveau tableau `Res` contenant les colonnes de `T` centrées-réduites.

    Centrer-réduire les colonnes d'un tableau est expliqué dans la partie 3 du Cours d'Introduction à la SAé.

12. Appliquer cette fonction à `Sangliers0Ar`, et recréer un DataFrame `Sangliers0DF_CR` à partir des données centrées-réduites.

Pour les plus rapides

Numpy.cov et Matrice de Covariance

Dans cette partie, on va apprendre à utiliser la fonction `np.cov`, en lien avec la covariance. Et on va apprendre ce qu'est une *Matrice de Covariance* !

13. Reprendre votre fonction `Covar` du TP3 de Statistiques. L'utiliser pour calculer la covariance entre le Nombre de Sangliers Prélevés et le Montant des Indemnisations, avec les chiffres avant d'avoir centré-réduit.

14. Appliquer la fonction Numpy `np.cov` aux 2 mêmes séries de données :

    - passer en paramètre les 2 séries sous forme d'un seul tableau à 2 colonnes,
    - indiquer l'option `rowvar=False`, pour préciser que les variables ne sont pas en lignes mais en colonnes,
    - indiquer l'option `bias=True`, pour que la fonction ne fasse pas de corrections.

    Comparer le résultat renvoyé avec votre résultat de la question précédente, ainsi qu'avec la variance du Nombre de Sangliers Prélevés et la variance du Montant des Indemnisations. Devinez-vous ce que contient la matrice renvoyée par `np.cov` ?

15. Que se passe-t-il si on applique `np.cov` aux données centrées-réduite ? Pourquoi ?

16. La fonction `np.cov` fonctionne aussi si on lui donne un tableau contenant plus de 2 séries de variables (plus de 2 colonnes). La tester avec 3 séries et essayer de deviner ce qu'elle renvoie.

    Quand vous aurez compris, vous aurez appris ce qu'est la *Matrice de Covariance* pour plusieurs séries de variables !