DS n° 02 – DS02

- Faire tous les exercices dans un même fichier NomPrenom.py à sauvegarder,
- mettre en commentaire l'exercice et la question traités (ex : # Exercice 1),
- ne pas oublier pas de commenter ce qui est fait dans votre code (ex : # Je créé une fonction pour calculer la racine d'un nombre),
- il est possible de demander un déblocage pour une question, mais celle-ci sera notée 0,
- il faut vérifier avant de partir que le code peut s'exécuter et qu'il affiche les résultats que vous attendez.

Étude d'un jeu de données de l'épidémie de covid-19

L'objectif de cette partie est d'analyser les données issues des test virologiques (PCR) utilisés pour déterminer la propagation de l'épidémie de Covid-19.

Deux jeux de données vont être utilisés :
- Données relatives aux résultats des tests virologiques COVID-19 (données brutes),
- Indicateurs de suivi de l'épidémie de COVID-19 (calculés à partir des données brutes).

Grâce à cette analyse, il faudra tenter de retrouver le nombre de test effectués à partie des indicateurs de suivi et de confronter ce résultats aux données brutes.

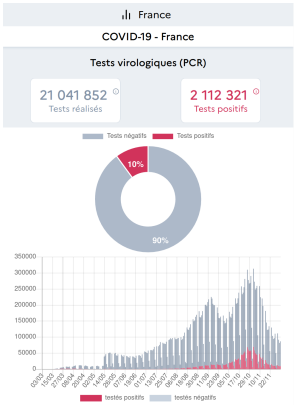

FIGURE 1 – Tableau de bord des données Covid-19

Liens vers les bases de données :
- https://www.data.gouv.fr/fr/datasets/indicateurs-de-suivi-de-lepidemie-de-covid-19/
- https://www.data.gouv.fr/fr/datasets/donnees-relatives-aux-resultats-des-tests-virologiques-...

Les données ont été téléchargées et formatées pour faciliter le travail. Elles sont présentes dans deux fichiers `source1.csv` et `source2.csv`. Ces fichiers sont présents dans le dossier de travail.

On rappelle les éléments suivants :
- Le **taux d'incidence** correspond au nombre de tests virologiques positifs pour 100.000 habitants sur une semaine. Il est calculé de la manière suivante :
  `taux_incidence=(100000*nb_cas_positif)/population`,
- Le **taux de positivité** correspond au nombre de personnes testées positives sur une semaine. Il est calculé de la manière suivante :
  `taux_positivite=(100*nb_cas_positifs)/nb_tests`,
- La population française est d'environ 67 000 000 habitants,
- Mettre en en-tête du fichier les éléments suivants :
  - `import matplotlib.pyplot as plt`
  - `from datetime import datetime, timedelta`

Lecture et analyse de la première base de données

1. Écrire un script python permettant de lire le contenu du fichier `source1.csv`, et de stocker son contenu dans une liste dont chaque élément serait une ligne du contenu du fichier. Afficher les deux premières lignes.

   Solution 1.

```
fichier=open('source1.csv','r')
contenu=fichier.read()
fichier.close()
```

```
lignes=contenu.split('\n')

print(lignes[0:2])
```

L'exécution du script affiche le résultat suivant :

```
['"extract_date","tx_incid","R","taux_occupation_sae","tx_pos"',
 '"2020-06-01",5.07875325643739,0.84,25.1,1.5664523487587']
```

On constate que les colonnes du fichier `csv` sont séparées par des ",".
Après avoir découpé la deuxième ligne selon ce séparateur et stocké son contenu dans une liste appelée `data`. Stocker ses éléments comme suit (attention de bien recopier les commandes) :
— la date du test : `jour_test = datetime.strptime(data[0], '"%Y-%m-%d"')`,
— le taux d'incidence : `taux_incidence=float(data[1])`,
— le taux de positivité : `taux_positivite=float(data[4])`.

2. Compléter le script en ajoutant les lignes permettant d'obtenir les trois données précédentes. Afficher ces valeurs pour la première semaine.

Solution 2.

```
data=lignes[1].split(",")
jour_test = datetime.strptime(data[0], '"%Y-%m-%d"')
taux_incidence=float(data[1])
taux_positivite=float(data[4])
print(jour_test,taux_incidence,taux_positivite)
```

L'exécution du script affiche le résultat suivant :
`2020-06-01 00:00:00 5.07875325643739 1.5664523487587`

3. Déterminer le nombre de tests effectués durant la première semaine à partir des données précédentes et grâce aux équations présentées en introduction. Nous ne garderons que la valeur entière de ce résultat.

Solution 3.
$taux_{incidence} = \frac{100000 * nb_{cas\ positif}}{population}$
$taux_{positivite} = \frac{100 * nb_{cas\ positif}}{nb_{tests}}$,
$nb_{cas\ positif} = \frac{taux_{incidence} * population}{100000}$
$nb_{tests} = \frac{100 * nb_{cas\ positifs}}{taux_{positivite}}$,
Donc,
$nb_{tests} = \frac{100 * \frac{taux_{incidence} * population}{100000}}{taux_{positivite}}$.

```
population=67000000
nb_tests=int((100*taux_incidence*population/100000)/taux_positivite)
print(nb_tests)
```

L'exécution du script affiche le résultat suivant :
217227

4. Créer deux listes `dates_tests1` et `nb_tests1` et y stocker les dates et le nombre de tests pour chaque semaine.

Solution 4.

```
dates_tests1=[]
nb_tests1=[]
for ligne in lignes[1:-1]:
    data=ligne.split(",")
    jour_test = datetime.strptime(data[0], '"%Y-%m-%d"')
    dates_tests1.append(jour_test)
    taux_incidence=float(data[1])
    taux_positivite=float(data[4])
    nb_tests=(100*taux_incidence*population/100000)/taux_positivite
    nb_tests1.append(int(nb_tests))
```

5. Tracer la courbe du nombre de tests en fonction de la date. (Une date telle que nous l'avons codée plus haut peut être utilisée comme un abscisse classique.)

Solution 5.

```
plt.plot(dates_tests1,nb_tests1)
```

Lecture et analyse de la deuxième base de données

6. Lire le contenu du fichier `source2.csv`, et stocker son contenu dans une liste dont chaque élément serait une ligne du contenu du fichier. Afficher les deux premières lignes.

Solution 6.

```
fichier_test=open('source2.csv','r')
contenu=fichier_test.read()
fichier_test.close()

lignes=contenu.split('\n')

print(lignes[0:2])
```

L'exécution du script affiche le résultat suivant :
`['fra;week;P_f;P_h;P;T_f;T_h;T;cl_age90', 'FR;2020-S21;64;56;125;5263;6192;11683;09']`
La dernière colonne, ayant ici la valeur `09` indique la tranche d'âge de la personne testée (ici entre 0 et 9 ans). Il y a donc les catégories suivantes :
— `09` : âge entre 0 et 9 ans,
— ...
— `89` : âge entre 80 et 89 ans,
— `90` : âge de 90 ans et plus,
— `0` : tous les âges.
Afin de comparer les résultats avec la base précédente, il faut prendre en compte toutes les tranches d'âge. C'est ce qui sera affiché lorsque cette colonne aura la valeur `0`.
ATTENTION, on constate que les colonnes du fichier csv sont cette fois-ci séparées par des ";".
Après avoir découpé la deuxième ligne selon ce séparateur et stocké son contenu dans une liste appelée `data` (SEULEMENT SI la valeur `data[8]=0`). Stocker ses éléments comme suit (attention de bien recopier les commandes) :
— numéro de la semaine à laquelle a eu lieu le test : `sem=data[1][-2:]`,
— jour du test (on ajoute le nombre de semaines passées au 1er janvier 2020) :
`jour_test = datetime(2020, 1, 1)+timedelta(weeks=int(sem))`
— le nombre de tests : `nb_tests(int(data[7]))`

7. Créer deux listes `dates_tests2` et `nb_tests2` et y stocker les dates et le nombre de tests pour chaque semaine.

Solution 7.

```
dates_tests2=[]
nb_tests2=[]
for ligne in lignes[1:-1]:
    data=ligne.split(";")
    if data[8]=='0':
        sem=data[1][-2:]
        jour_test = datetime(2020, 1, 1)+timedelta(weeks=int(sem))
        dates_tests2.append(jour_test)
        nb_tests=int(data[7])
        nb_tests2.append(nb_tests)
```

8. Tracer la courbe du nombre de tests en fonction de la date. (Une date telle que nous l'avons codée plus haut peut être utilisée comme un abscisse classique.)

Solution 8.

```
plt.figure(1)
plt.plot(dates_tests2,nb_tests2)
plt.show()
```

Le 1er janvier 2020, n'étant pas un dimanche, nous obtenons un léger décalage entre les deux courbes.

9. En sachant que le dimanche précédent le 1er janvier 2020 était le 29 décembre 2019, modifier le code précédent pour prendre en compte ce décalage et tracer à nouveau la courbe.

Solution 9.

```
dates_tests2=[]
nb_tests2=[]
for ligne in lignes[1:-1]:
    data=ligne.split(";")
    if data[8]=='0':
        sem=data[1][-2:]
        jour_test = datetime(2019, 12, 29)+timedelta(weeks=int(sem))
        dates_tests2.append(jour_test)
        nb_tests=int(data[7])
        nb_tests2.append(nb_tests)

plt.figure(1)
plt.plot(dates_tests2,nb_tests2)
plt.show()
```

Comparaison des deux courbes

L'objectif de cette partie est de déterminer l'erreur relative entre les données calculées. Avant de comparer les deux bases, on constate que leur remplissage n'a pas démarré en même temps, la première date stockée n'étant pas la même.

10. Créer une liste `dates_ecarts`, contenant les dates des tests et une liste `ecarts` contenant la différence entre le nombre de tests déterminé à partir de `nb_tests1` et celui déterminé à partir de `nb_tests2`. Il ne faudra remplir la liste que si les deux valeurs du nombre de test sont disponibles à cette date.

Solution 10.

```
dates_ecarts=[]
ecarts=[]
for i,date in enumerate(dates_tests2):
     if date in dates_tests1:
         dates_ecarts.append(date)
```

11. Tracer cet écart en fonction de la date (il faudra commenter, dans le script, l'affichage des courbes précédentes afin de ne pas écraser celle-ci.)

Solution 11.

```
plt.figure(2)
plt.plot(dates_ecarts,ecarts)
plt.show()
```

12. Proposer une fonction `get_max(liste)` qui permet de déterminer le maximum, en valeur absolue, de la liste `liste`.

Solution 12.

```
def get_max(liste):
    maximum=abs(liste[0])
    for valeur in liste:
        if abs(valeur)>maximum:
            maximum=abs(valeur)
    return maximum
```

13. Déterminer, à l'aide de cette fonction, le maximum, en valeur absolue, de `ecarts` et de `nb_tests1` et en déduire l'erreur relative maximale en % entre `nb_tests1` et `nb_tests2`.

Solution 13.

```
print('Nb écarts',get_max(ecarts))
print('Nb tests',get_max(nb_tests1))

print('Erreur relative pourcent',100*get_max(ecarts)/get_max(nb_tests1))
```