Rapport du Projet traitement de données

Covid-19 en France

Etudiants:

Placet Guillaume
Hassini Safae
Tabbai Aymane

Tutrice:

Yepnga Natacha

Table des matières

Résumé

Nous présentons dans ce rapport une application codée en langage Python permettant à un utilisateur quelconque de traiter les informations sur les hospitalisations dues au Covid-19. En effet notre base de données est constituée de 5 fichiers sur les hospitalisations journalières en France dues au Covid-19 en 2021, accompagnés de leur métadonnées. Ces fichiers sont de source gouvernementale. L'application va pouvoir importer les données, les lire, les extraires, les analyser par de la statistique de base et du clustering grâce à l'algorithme des kmeans.

Introduction

Notre travail portera sur la confection d'une application ayant pour but d'analyser par des méthodes statistiques les hospitalisations dues au Covid-19. Nous avons pour cela créé plusieurs classes d'objets en langage Python qui permettent d'importer, de modifier, de produire des statistiques basiques et plus avancées comme la méthode de clustering k means et enfin d'exporter les données en fichier csv.

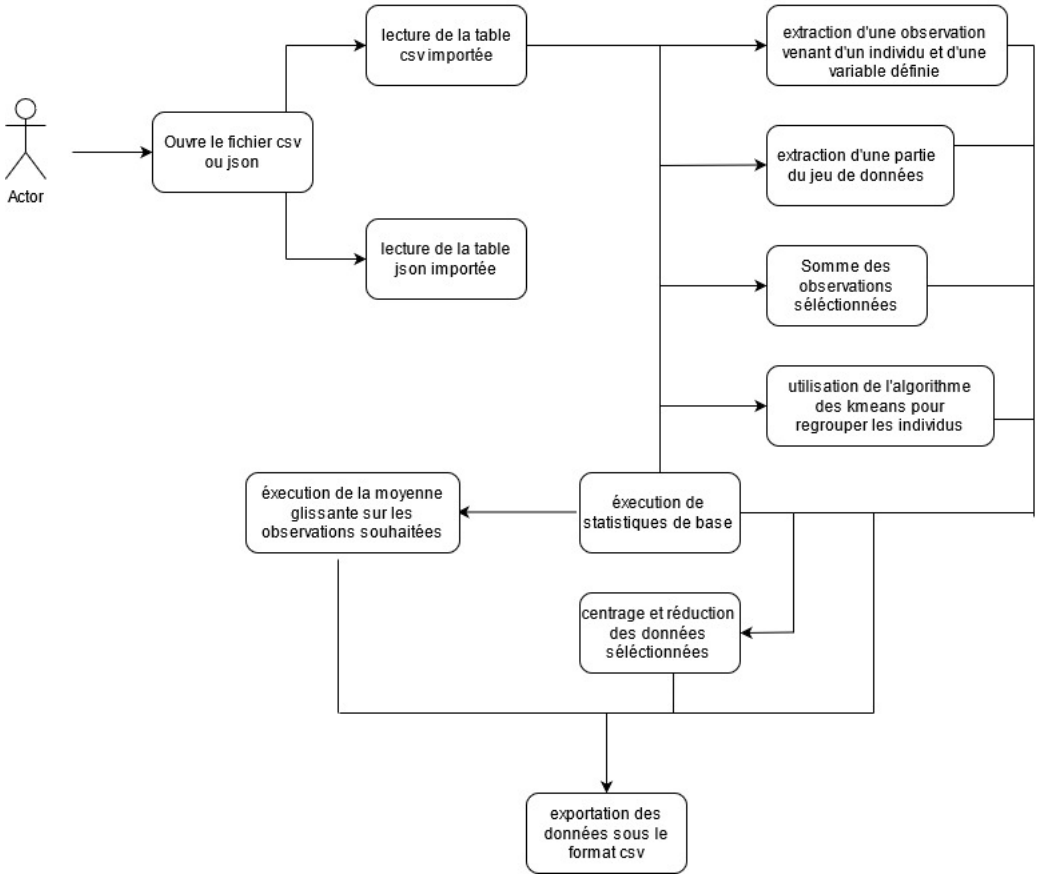
Dans les classes créés nous retrouvons : une classe qui se chargera de la sélection du nom des variables et de la sélection des observations. Une classe qui se charge de la sélection des données entre deux dates, une autre qui s'occupe de la sélection plus spécifiée, une autre qui prends en charge la somme des données d'une variable et une dernière qui exporte les données en csv.

Il y a aussi les classes de production statistique comme la classe pour les statistiques de base (moyenne, variance, écart-type),la classe pour la moyenne glissante et la classe pour le k means. Dans ces classes nous avons créé des méthodes (ou fonctions) comme la moyenne glissante ou le calcul de variance. L'architecture globale du code permet à l'utilisateur d'user des méthodes pour modifier traiter et analyser l'information d'un point de vue statistique.

Pour ce qui est de la base de données. Elle est composée d'un fichier contenant les vacances scolaires en France dans un format json et d'un dossier de données et métadonnées concernant les hospitalisations dues au Covid en France. Les métadonnées sont des informations que nous avons sur les données. En ce sens les données médicales étant sensibles nous pouvons supposer qu'elles permettent d'identifier la source des données elles mêmes.Ces données traitent de l'âge des patients, le nombre d'établissement pour les patients Covid, l'incidence en réanimation. Certaines données sont mesurées en Régions alors que d'autres le sont mais en Départements. Enfin certaines variables sont communes à plusieurs tables.

1 Description générale de l'application

Notre application est divisée en 11 classes contenant chacune des méthodes . 3 classes sont dédiées à l'importation et au bon formatage des données (**Data**, **ReadFile** et **Vacances**. 3 autres classes sont dédiées à la lecture et l'extraction des données(**Df_where**, **Df_window** et **Df_sumby**. 3 pour la statisique de base et avancée (**Statistic**, **Moyenne_gliss** et
Data_centrer_reduire, une classe pour le fonctionnement du kmeans et une dernière pour l'exportation des données. Avec cet interface l'utilisateur pourra faire intervenir dans l'ordre les méthodes qu'il veut pour importer lire et analyser ses données. Par exemple extraire les dix premières observations et en faire la moyenne.



Maquette de l'application

La maquette de l'application est différente de celle introduite dans le cahier de charges car dans cette période on était un peu perdus et on avait pas une vision exacte de ce qu'on avait à faire.

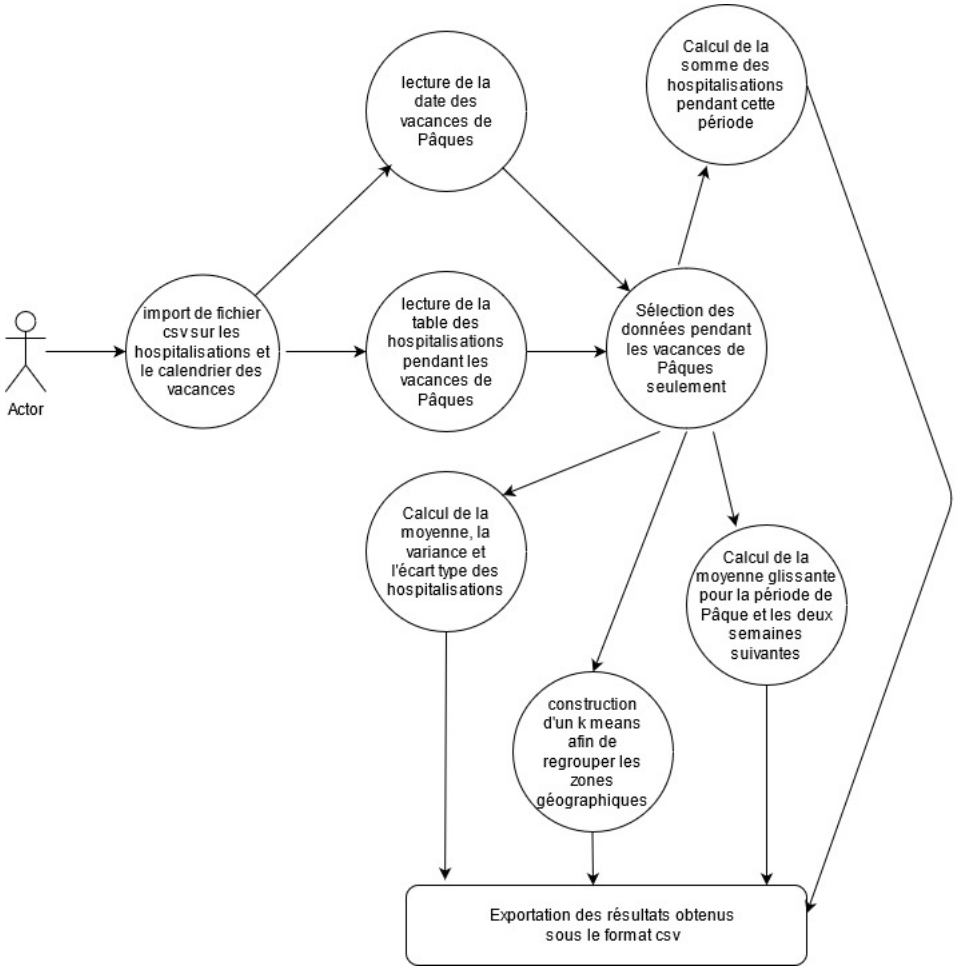


Diagramme de cas d'utilisation

2 Gestion des données :

2.1 Présentation des données :

Dans ce projet deux types de jeux données sont proposés ,il y'a des données au format CSV et un jeux de données au format json, les données au format csv sont disponibles sur le site data.gouv.fr, alors que le jeu de données au format json est disponible sur gitlab.com.

2.2 Importation des données :

Pour importer ces données dans Python, nous avons créé une classe ReadFile qui prend en paramètre le nom du fichier quand désire importer, et grâce à ces deux méthode **open_csv()** et **open_json()**, on importe nos données qu'il soient de format **csv** ou **json** Les données en format csv sont importées sous forme de liste de listes, les valeurs numériques sont converties en **str** , cette conversion a été prise en compte dans toutes les fonctionnalités de nos codes. Le jeu de données en format **json** sont importées sous forme de dictionnaire.

2.3 Transformation des données :

Pour manipuler les données en Format **csv**, on a créé le module Transformations qui contient les classes suivantes :

La classe **Data** : le constructeur de cette classe prend une liste de données comme paramètre, cette classe possède trois méthodes
get_items : Elle retourne une liste qui contient le nom des variables de notre jeu de données
get_value_in_items elle prend en paramètre le nom de la variable dont on veut retourner les observations, elle retourne une liste, telle que le premier élément représente le nom de la variable et ensuite les observations de cette variable
get : une méthode abstraite qui s'adapte aux classes suivantes

La classe **Df_Where** qui grâce à ces méthodes permet de sélectionner uniquement les données au sein desquelles une variable est égale à une certaine valeur, variable et valeur sont initiées dans le constructeur .

La classe **Df_Window** qui permet de sélectionner uniquement les données entre deux dates initiées au constructeur, les deux dates sont initiées dans le constructeur

La classe **Df_sumby** qui renvoie la somme d'une variable initiée dans le constructeur dans son jeu de données, le nom de la variable est initié dans le constructeur.

Pour manipuler le jeu de données en format json une classe **Vacances** a été créé, cette classe permet de gérer notre dictionnaire vacances, les méthodes de cette classe permettent de renvoyer la date des vacances, incrémentée dans une certaine zone.

2.4 Exportation des données :

Pour exporter nos données, nous avons créé une classe **Export** qui grâce à ces méthode crée un fichier **csv** et introduit dedans nos résultats., cette classe n'a pas été mise dans le diagramme de classe vu qu'elle n'a pas une relation directe avec notre classe, en effet nous avons créé un module test , qui teste notre programme et au sein de ce module Export a été importé afin de mettre les résultats de nos tests dans un fichier **csv** .

3 Présentation des classes :

3.1 Diagramme de classes :

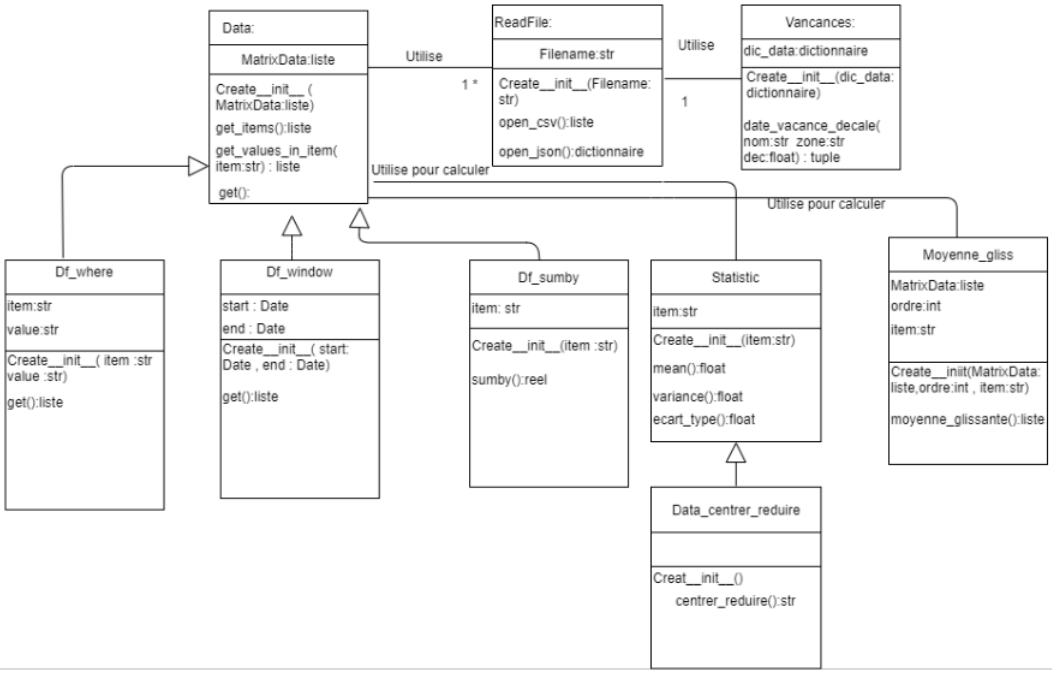


FIGURE 1 – Diagramme de classes

Les classes chargées de la transformation et la gestion des données ont été définis précédemment, reste à définir les classes statistiques : Pour la manipulation statistique de nos données nous avons créé les classes suivantes :

La classe **Statistic** :Le constructeur de la classe **Statistic** est initiée par deux attributs : notre jeu de données et le nom de la variable dont on veut calculer soit la moyenne , soit la variance ou l'écart-type, en effet nous avons définis trois méthodes dans cette classe :

La méthode **mean()** :qui renvoie la moyenne de la variable **item**
La méthode **variance()** : qui renvoie la variance de la variable item

La méthode **ecart_type()** : qui renvoie l'écart-type de la variable item
La classe **Moyenne_gliss** :Le constructeur de cette classe est initiée par notre jeu de données , l'ordre il s'agit de la taille et de la plage de données dont on veut calculer la moyenne, et le nom de la variable dont on veut calculer la moyenne glissante.

Cette classe comporte une seule méthode qui est **moyenne glissante()**,cette méthode renvoie une liste de moyennes qui prend en considération l'ordre définis dans le constructeur.
Et il y'a la classe **data_centrer_reduire** : le constructeur de cette classe prend en paramètre notre jeu de données et le nom de notre variables et grâce à ces méthodes et les méthodes héritées de la classe statistic elle nous renvoie une nouvelle liste des observations centrées réduites de la variable item initialisée au constructeur.

4 Kmeans

Nous avons créé une classe kmeans pour pouvoir utiliser l'algorithme des k means afin de faire un clustering (regroupement) des individus qui se ressemblent le plus. Notons que

cette classe n'est pas représentée dans le diagramme de classe car elle ne possède pas un lien direct avec les autres classes et à seulement été importé pour la phase de test. Dans ce code, l'utilisateur doit choisir le nombre de groupes k avec précaution pour pouvoir tirer du sens de ces calculs. Cette classe est composée de plusieurs fonctions dont les distances euclidiennes, la création de groupes (clusters) et de centroides puis la création d'une boucle principale. Ces fonctions sont utilisées par la méthode predict qui est la fonction principale de la classe. Cette classe possède 5 attributs :
K, le nombre de groupes à générer ; **max_iteration**, le nombre maximal d'iterations (100 par défaut) ; **graph** qui affiche ou non les graphiques (False par défaut) ; **clusters**, la liste des clusters ; **centroids**, la liste des centres de chaque cluster.

Elle possède 7 méthodes :

la méthode **predict()**, prends en entrée la liste de données, créé une liste de centroides aléatoire puis effectue les assignations afin de créer les groupes souhaités qui les retourne en sortie ;

get_cluster_labels(), qui prends en entrée la liste des clusters ou groupes, permet de générer les labels des clusters puis retourne ces labels ;

create_clusters() qui a pour entrée la liste des centroides, créé les groupes à partir des centroides labelisés pius renvoie les broupes ou clusters ;

closest_centroid(), ayant pour entrée les points et les centroides, utilise la fonction des distances euclidiennes pour déterminer le centroide le plus proche des points puis nous donne le nom du centroide le plus proche en sortie ;

get_centroids(), a pour entrée la liste des groupes ou clusters et permet de calculer le centroide d'un groupe en ayant comme sortie la liste des centroides nouvellement créée ;

is_converged(), prends en entrée la liste des anciens et nouveaux clusterspour savoir si les deux types de centroides sont convergents entre-eux et retourne false ou true selon la convergence ;

plot, ne prends pas de paramètres en entrée et affiche les résultats de manière graphique en sortie.

Finalement ce méthodes s'articulent pour que l'utilisateur puisse effectuer le k means souhaité.

5 Test et Résultats

Nous allons nous appuyer sur 5 questions pour pouvoir tester l'ensemble des fonctionnalités de notre programme :
Pour calculer le nombre total d'hospitalisations dues au Covid-19, nous allons utiliser les fonction get() de la classe **Df_window()** et **sumby()** de la classe **Df_sumby**

```
'La première question'
' Le nombre total d'hospitalisations dues au Covid-19'
covid_sans_sexe=t.Df_where(l.covid_sexe,'sexe','0').get()
nombre_total_hosp=t.Df_sumby(covid_sans_sexe,'hosp').sumby()
'On trouve que le nombre total d hospitalisations dues au Covid19 est'
'6178301'
export_resultat1=e.Export("nombre totale d'hospitalisations ",nombre_total_hosp)
export_resultat1.exporter("stockage5.csv")
```

FIGURE 2 – le nombre d'hospitalisations due au Covid-19 est 6178301

Pour répondre à la question "Combien de nouvelles hospitalisations ont eu lieu ces 7 derniers jours dans chaque département ?" Nous avons testé 5 fonctions issues de différentes classes. Les fonctions **Df_window()** et **Df_where** de la classe transformation pour extraire les données selon la variable **incid_hos** et la période donnée, puis en utilisant la fonction **Df_sumby** nous avons pu calculer la somme de cette variable sur chaque département et finalement nous avons utilisé la méthode **exporter()** de la classe Export pour stocker le résultat dans un fichier csv nommé stockage 5.csv

```
#Combien de nouvelles hospitalisations ont eu lieu ces 7 derniers jours dans chaque département?
b=t.Df_window(l.covid_dep,"2021-02-24","2021-03-03")
test=b.get()
nbr_hospitalisation_departemental={}
for k in range(1,102):

    test_dep=t.Df_where(test,"dep", test[k][0]).get()
    a=t.Df_sumby(test_dep,"incid_hosp").sumby()
    nbr_hospitalisation_departemental[test[k][0]]=a
    d=e.Export("nouvelle hospitalisations dans le departement "+ test[k][0],a)
    d.exporter("stockage5.csv")
```

FIGURE 3 – implementation du code pour répondre à la question

1	nouvelle hospitalisations dans le departement 01	68
2	nouvelle hospitalisations dans le departement 02	145
3	nouvelle hospitalisations dans le departement 03	39
4	nouvelle hospitalisations dans le departement 04	54
5	nouvelle hospitalisations dans le departement 05	41
6	nouvelle hospitalisations dans le departement 06	375
7	nouvelle hospitalisations dans le departement 07	61
8	nouvelle hospitalisations dans le departement 08	22
9	nouvelle hospitalisations dans le departement 09	9
10	nouvelle hospitalisations dans le departement 10	60
11	nouvelle hospitalisations dans le departement 11	34
12	nouvelle hospitalisations dans le departement 12	27
13	nouvelle hospitalisations dans le departement 13	642
14	nouvelle hospitalisations dans le departement 14	90
15	nouvelle hospitalisations dans le departement 15	13
16	nouvelle hospitalisations dans le departement 16	30

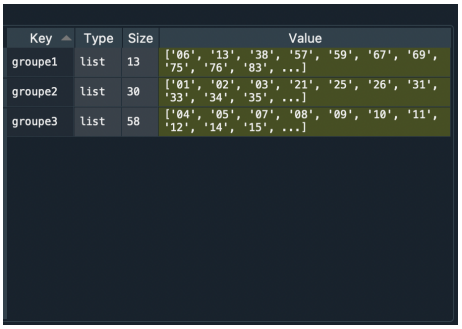
FIGURE 4 – Capture d'écran figurant les premières lignes du fichier stockage.csv

Come pour le test précédent nous avons utilisé les méthodes **Df_window()** et **Df_where** de la classe transformation pour extraire les données, après nous avons une matrice dont les lignes sont les observations des departements donnés par les 4 variables sur 31 jours du mois février, pour ce qui concerne le lissage nous nous sommes servis de la méthode

moyenne_glissante() de la classe Moyenne_gliss. Dans un dernier temps, après la transformation du jeu de donnée en array nous avons réalisé un kmeans avec k=3 en utilisant la méthode **predict()** de la classe KMeans.

```
'''on commence par extraire le jeux donnees sure la date 2020-12-29","2021-02-03
pour pouvoir calculer la moyenne glissante sur les le premier jour et le dernier jour
'''

b=t.Df_window(l.covid_dep,"2020-12-29","2021-02-03")
b1=b.get()


donnes=[]
for i in range(1,102) : #boucle sur les departement pour extraire les donnees sur chaque dep
    test_dep=t.Df_where(b1,"dep", test[i][0])
    test_dep1=test_dep.get()
    K=[]
    for j in range(2,6) :          #boucle sur les variables
        moyglis=s.moyenne_gliss(test_dep1,7,test[0][j]).moyenne_glissante()
        #moyenne glissante de lma i-eme dep sur la j-eme variable
        K=K+moyglis #stocker les les moyennes glissantes pourtout les variables de chaque dep
    donnes.append(K) #stocker les moyennes glissantes des depart



covid2=np.array(donnes) #transformation en array
k=KMeans(K=3,max_iteration=100,graph=False)# creer l'objet de la classe kmeans

t_pred=k.predict(covid2)#predire le regroupement avec la fonction predict
departement=t.Data(b1).get_values_in_item("dep")[:101]#list des dep
#presentation des resultats sous forme de dictionnaire
d={}
d["groupe1"]=[]
d["groupe2"]=[]
d["groupe3"]=[]
for i in range (0,len(departement)) :
    if t_pred[i]==0 :
        d["groupe1"].append(departement[i])
    elif t_pred[i]==1 :
        d["groupe2"].append(departement[i])
    else :
        d["groupe3"].append(departement[i])
```

FIGURE 5 – Implementation du test de la méthode Kmeans



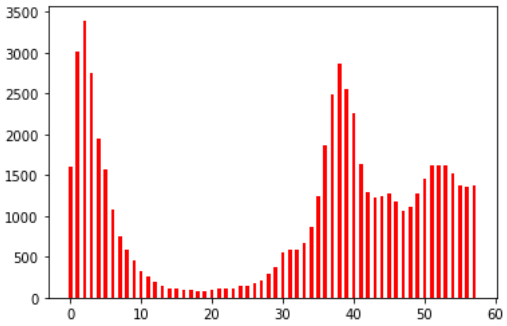
Key	Type	Size	Value
groupe1	list	13	['06', '13', '38', '57', '59', '67', '69', '75', '76', '83', ...]
groupe2	list	30	['01', '02', '03', '21', '25', '26', '31', '33', '34', '35', ...]
groupe3	list	58	['04', '05', '07', '08', '09', '10', '11', '12', '14', '15', ...]

résultat stocké dans un dictionnaire

102	le departement 06	groupe1
103	le departement 13	groupe1
104	le departement 38	groupe1
105	le departement 57	groupe1
106	le departement 59	groupe1
107	le departement 67	groupe1
108	le departement 69	groupe1
109	le departement 75	groupe1
110	le departement 76	groupe1
111	le departement 83	groupe1
112	le departement 92	groupe1
113	le departement 93	groupe1
114	le departement 94	groupe1
115	le departement 01	groupe2
116	le departement 02	groupe2
117	le departement 03	groupe2
118	le departement 21	groupe2
119	le departement 25	groupe2

résultat exporté vers un fichier csv

Pour faire évoluer la moyenne des nouvelles hospitalisations journalières de chaque semaine par rapport à une autre semaine nous avons dessiné un diagramme en barres où chaque barre représente la moyenne des nouvelles hospitalisations journalières dans une semaine. Les semaines sont dans l'ordre renseigné dans la liste **_dates_de_moyennes**. Dans cette partie, plusieurs fonctionnalitées de notre programme ont été testés, puisqu'on devait d'abord sélectionner toutes les dates de nos données ,et ensuite calculer la moyenne



Finalement pour répondre à la question posée, nous avons utilisé la méthode **date_vacance_decalé** de la classe Vacances pour retourner la période issu d'une décalage de 7 jours avec les vacances de la Toussaint, puis nous faisons l'extraction sur cette période avec la fonction get() de la classe **D_window()**. Nous calculons par la suite la somme par la methode sumby() de la classe **Df_sumby** et nous exportons la résultats dans le fichiers **stockage5.csv**. Cela permet d'ajouter une ligne contenant le nom du résultat et la valeur 3521

```
#question 5
#fonction qui calcule la somme selon la variable choisie durant une periode lié au vacance:

vac=v.Vacances(l.vacances)
(x,y)=vac.date_vacance_decale("Vacances de la Toussaint","2020-2021","Zone A",7)
periode_tous=b=t.Df_window(l.covid_dep,x,y)
rea=periode_tous.get()
reanimationvac=t.Df_sumby(rea,"incid_rea").sumby()
r=e.Export("nombre totale  de réanimation peadant la semaine suivant les Vacances de la Toussaint ",reanimationvac)
r.exporter("stockage5.csv")
```

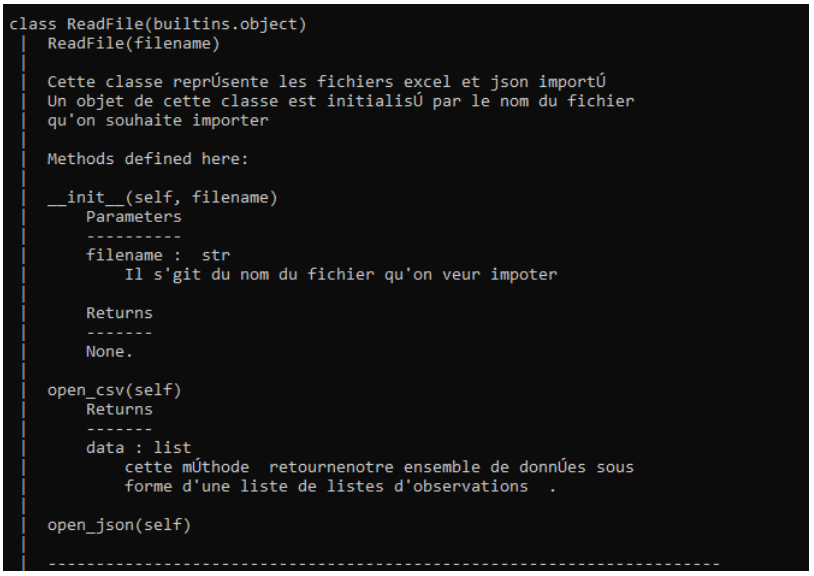
FIGURE 6 – implementation pour tester la focntion date_vacance_decale()

6 Conclusion

Pour conclure, nous avions pensé dans le cahier des charges à une architecture que nous avons modifié par la suite. En effet le découpage des classes est maintenant plus fin en séparant par exemple la moyenne glissante des statistiques de base. Nous avons ensuite créé le code. Ce dernier permet à tout utilisateur d'extraire les informations qu'il souhaite et les sommer dans la base des hospitalisations afin de mieux comprendre les dynamiques de l'épidémie de Covid-19 actuelle avec la possibilité de faire les statistiques de bases ainsi qu'un clustering kmeans. Nous aurions pu envisager une interface graphique afin de mieux visualiser le traitement des donnnées. Toutefois nous n'avons pas eu le temps.

7 Annexe :

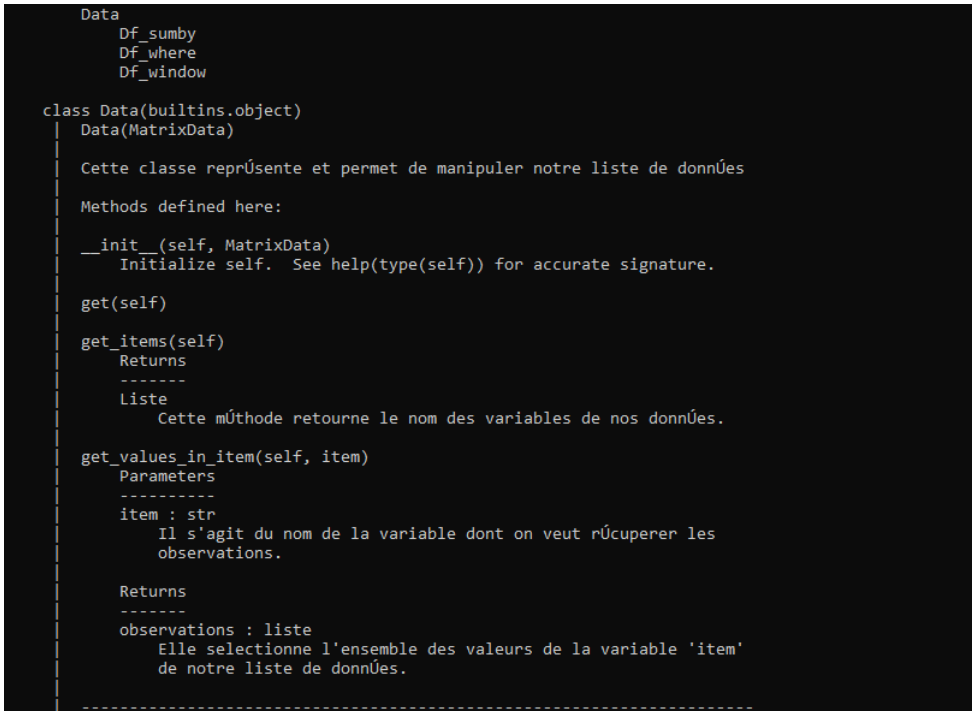
7.1 La classe ReadFile :



```
class ReadFile(builtins.object)
 |  ReadFile(filename)
 |
 |  Cette classe reprÚsente les fichiers excel et json importÚ
 |  Un objet de cette classe est initialisÚ par le nom du fichier
 |  qu'on souhaite importer
 |
 |  Methods defined here:
 |
 |  __init__(self, filename)
 |      Parameters
 |      ----------
 |      filename :  str
 |          Il s'git du nom du fichier qu'on veur impoter
 |
 |      Returns
 |      -------
 |      None.
 |
 |  open_csv(self)
 |      Returns
 |      -------
 |      data : list
 |          cette mÚthode  retournenotre ensemble de donnÚes sous
 |          forme d'une liste de listes d'observations  .
 |
 |  open_json(self)
 |
 |  ----------------------------------------------------------------------
```

FIGURE 7 – Documentation de la classe **ReadFile**

7.2 La classe Data :



```
    Data
        Df_sumby
        Df_where
        Df_window

class Data(builtins.object)
 |  Data(MatrixData)
 |
 |  Cette classe reprÚsente et permet de manipuler notre liste de donnÚes
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  get(self)
 |
 |  get_items(self)
 |      Returns
 |      -------
 |      Liste
 |          Cette mÚthode retourne le nom des variables de nos donnÚes.
 |
 |  get_values_in_item(self, item)
 |      Parameters
 |      ----------
 |      item : str
 |          Il s'agit du nom de la variable dont on veut rÚcuperer les
 |          observations.
 |
 |      Returns
 |      -------
 |      observations : liste
 |          Elle selectionne l'ensemble des valeurs de la variable 'item'
 |          de notre liste de donnÚes.
 |
 |  ----------------------------------------------------------------------
```

FIGURE 8 – Documentation de la classe Data

7.3 La classe Df_window :

```
class Df_window(Data)
 |  Df_window(MatrixData, start, end)
 |
 |  Method resolution order:
 |      Df_window
 |      Data
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData, start, end)
 |      Parameters
 |      ----------
 |      MatrixData : TYPE
 |          DESCRIPTION.
 |      start : Date
 |          Il s'agit de la date Ó partir de la quelle on souhaite sÚlectionner
 |          nos donnÚes..
 |      end : Date
 |          Il s'agit de la date de fin des donnÚes qu'on veut sÚlectionner..
 |
 |      Returns
 |      -------
 |      None.
 |
 |  get(self)
 |      Cette mÚthode permet de  sÚlectionner les donnÚs o¨ la date est
 |      entre start et end
 |
 |
 |      Returns
 |      -------
 |      resultat : Liste
 |          Il s'agit d'une liste de listes telles que les listes reprÚsentent
 |          les observations entre les dates start et end,les dates start et end sont
 |          incluses.
 |
 |  ----------------------------------------------------------------------
```

FIGURE 9 – Documentation de la classe **Df_window**

7.4 La classe Df_where :

```
class Df_where(Data)
 |  Df_where(MatrixData, item, value)
 |
 |  Method resolution order:
 |      Df_where
 |      Data
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData, item, value)
 |      Parameters
 |      ----------
 |      MatrixData : TYPE
 |          DESCRIPTION.
 |      item :str
 |      value : str
 |
 |      Returns
 |      -------
 |      None.
 |
 |  get(self)
 |      Cette mÚthode permet de  sÚlctionner les donnÚs o¨ item est
 |      Úgale Ó value
 |
 |      Returns
 |      -------
 |      resultat : liste
 |          liste des donnÚes dont la varible item Úgale value.
 |
 |  ----------------------------------------------------------------------
```

FIGURE 10 – Documentation de la classe **Df_where**

7.5 La classe Df_sumby :

```
class Df_sumby(Data)
 |  Df_sumby(MatrixData, item)
 |
 |  cette classe permet de manipuler nos donnÚes
 |
 |  Method resolution order:
 |      Df_sumby
 |      Data
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData, item)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  sumby(self)
 |          Cette mÚthode perment de retourner la somme des valeurs de
 |          la variable item
 |
 |      Returns
 |      -------
 |      somme:float.
 |          Il s'agit de la sommes des observations de la variable item'
 |
 |  ----------------------------------------------------------------------
```

FIGURE 11 – Documentation de la classe **Df_sumby**

7.6 Les méthodes héritées de la classe Data :

Les classes **Df_where**, **Df_window** et **Df_sumby** ont héritées les méthodes présentées dans la figure suivante :

```
----------------------------------------------------------------------
Methods inherited from Data:

get_items(self)
    Returns
    -------
    Liste
        Cette mÚthode retourne le nom des variables de nos donnÚes.

get_values_in_item(self, item)
    Parameters
    ----------
    item : str
        Il s'agit du nom de la variable dont on veut rÚcuperer les
        observations.

    Returns
    -------
    observations : liste
        Elle selectionne l'ensemble des valeurs de la variable 'item'
        de notre liste de donnÚes.

----------------------------------------------------------------------
Data descriptors inherited from Data:

__dict__
    dictionary for instance variables (if defined)

__weakref__
    list of weak references to the object (if defined)
```

FIGURE 12 – méthodes héritées

7.7 Documentation de la classe Vacances

```
class Vacances(builtins.object)
 |  Vacances(dic_data)
 |
 |  Methods defined here:
 |
 |  __init__(self, dic_data)
 |      cette classe nous permet de manipuler le fichier json
 |
 |      Parameters
 |      ----------
 |      dic_data : dictionnaire
 |          dictionnaire qui contient la calendrier des vacances scolaires.
 |
 |      Returns
 |      -------
 |      None.
 |
 |  date_vacance_decale(self, nom, annees, zone, dec=0)
 |      cette methode nous permet de retourner une periode decale apres ou avant date des vacances
 |
 |      Parameters
 |      ----------
 |      nom : str
 |          nom des vacances.
 |      annees : str
 |          annee scolaire.
 |      zone : str
 |          zone du departement.
 |      dec : float
 |          nombre de jours de decalage. par defut 0.
 |
 |      Returns
 |      -------
 |      str
 |          date de debut de la perode.
 |      TYPE
 |          date de fin de la periode.
 |
 |  ----------------------------------------------------------------------
```

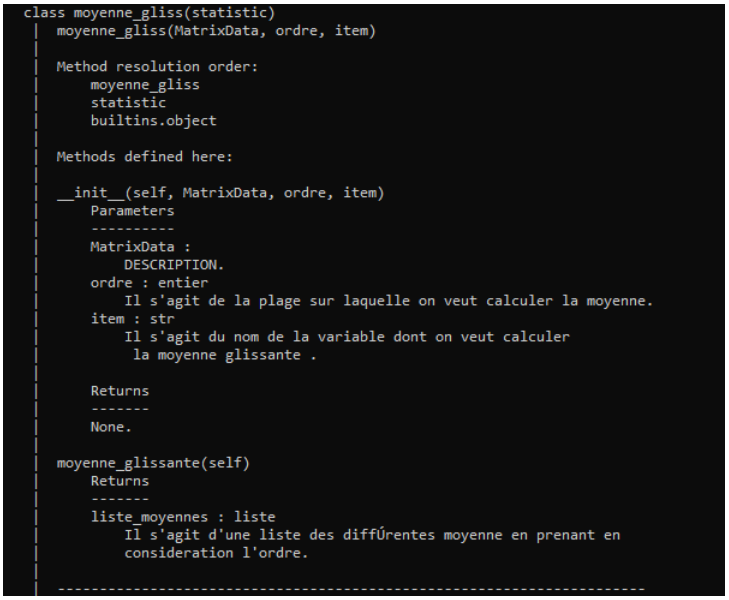
FIGURE 13 – Documentation de la classe Vacances

7.8 Documentation de la classe statistic :

```
class statistic(builtins.object)
 |  statistic(MatrixData, item)
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData, item)
 |
 |       Parameters
 |       ----------
 |       MatrixData : Liste
 |           Il s'agit d'une liste de listes qui illustre nos donnees.
 |      item : str
 |            Le nom de la variable dont on veut calculer les statistiques
 |            simples.
 |
 |       Returns
 |       -------
 |       None.
 |
 |  ecart_type(self)
 |       Cette mÜthode prend en paramÞtre une liste de donnÜes et renvoie
 |       l'Ücart type de la variable item
 |
 |
 |
 |      Returns
 |      -------
 |      reel
 |          l'Ücart-type de la variable item.
 |
 |  mean(self)
 |      Cette mÜthode prend en paramÞtre une liste de donnÜes et renvoie
 |      la moyenne de la variable item .
 |
 |
 |
 |      Returns
 |      -------
 |      reel
 |          La moyenne de la variable item .
 |
```

FIGURE 14 – Documentation de la classe statistic

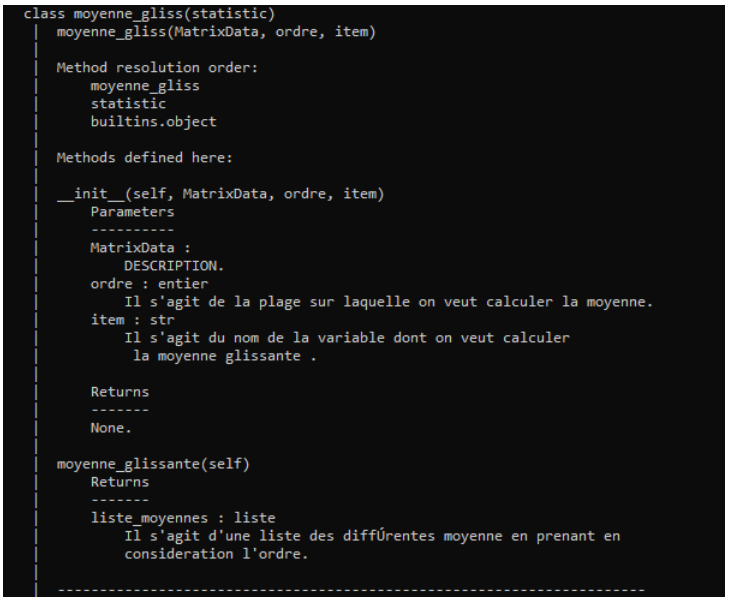
7.9 Documentation de la classe moyenne_gliss :



```
class moyenne_gliss(statistic)
 |  moyenne_gliss(MatrixData, ordre, item)
 |
 |  Method resolution order:
 |      moyenne_gliss
 |      statistic
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData, ordre, item)
 |      Parameters
 |      ----------
 |      MatrixData :
 |          DESCRIPTION.
 |      ordre : entier
 |          Il s'agit de la plage sur laquelle on veut calculer la moyenne.
 |      item : str
 |          Il s'agit du nom de la variable dont on veut calculer
 |           la moyenne glissante .
 |
 |      Returns
 |      -------
 |      None.
 |
 |  moyenne_glissante(self)
 |      Returns
 |      -------
 |      liste_moyennes : liste
 |          Il s'agit d'une liste des diffÚrentes moyenne en prenant en
 |          consideration l'ordre.
 |
 |  ----------------------------------------------------------------------
```

FIGURE 15 – Documentation de la classe moyenne_gliss :

7.10 Documentation de la classe moyenne_gliss :



```
class moyenne_gliss(statistic)
 |  moyenne_gliss(MatrixData, ordre, item)
 |
 |  Method resolution order:
 |      moyenne_gliss
 |      statistic
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData, ordre, item)
 |      Parameters
 |      ----------
 |      MatrixData :
 |          DESCRIPTION.
 |      ordre : entier
 |          Il s'agit de la plage sur laquelle on veut calculer la moyenne.
 |      item : str
 |          Il s'agit du nom de la variable dont on veut calculer
 |           la moyenne glissante .
 |
 |      Returns
 |      -------
 |      None.
 |
 |  moyenne_glissante(self)
 |      Returns
 |      -------
 |      liste_moyennes : liste
 |          Il s'agit d'une liste des diffÚrentes moyenne en prenant en
 |          consideration l'ordre.
 |
 |  ----------------------------------------------------------------------
```

FIGURE 16 – Documentation de la classe moyenne_gliss :

7.11 Documentation de la classe data_centrer_reduire :

```
class data_centrer_reduire(statistic)
 |  data_centrer_reduire(MatrixData, item)
 |
 |  Method resolution order:
 |      data_centrer_reduire
 |      statistic
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, MatrixData, item)
 |      Parameters
 |      ----------
 |      MatrixData : liste
 |          Il s'agit d'une liste de listes qui illustre nos donnees..
 |
 |      Returns
 |      -------
 |      None.
 |
 |  centrer_reduire(self)
 |      Cette mÚthode prend en paramÞtre le nom de la variable quand veut
 |      centrer et reduire, cette variable existe dans le jeu de donnÚes
 |      initialise dans le constructeur
 |      Parameters
 |      ----------
 |      item : str
 |          Il s'agit du nom de la varible quand veut centrer et reduire.
 |
 |      Returns
 |      -------
 |      liste
 |           Elle represente une liste des observations items centrees reduites
 |
 |  ----------------------------------------------------------------------
```

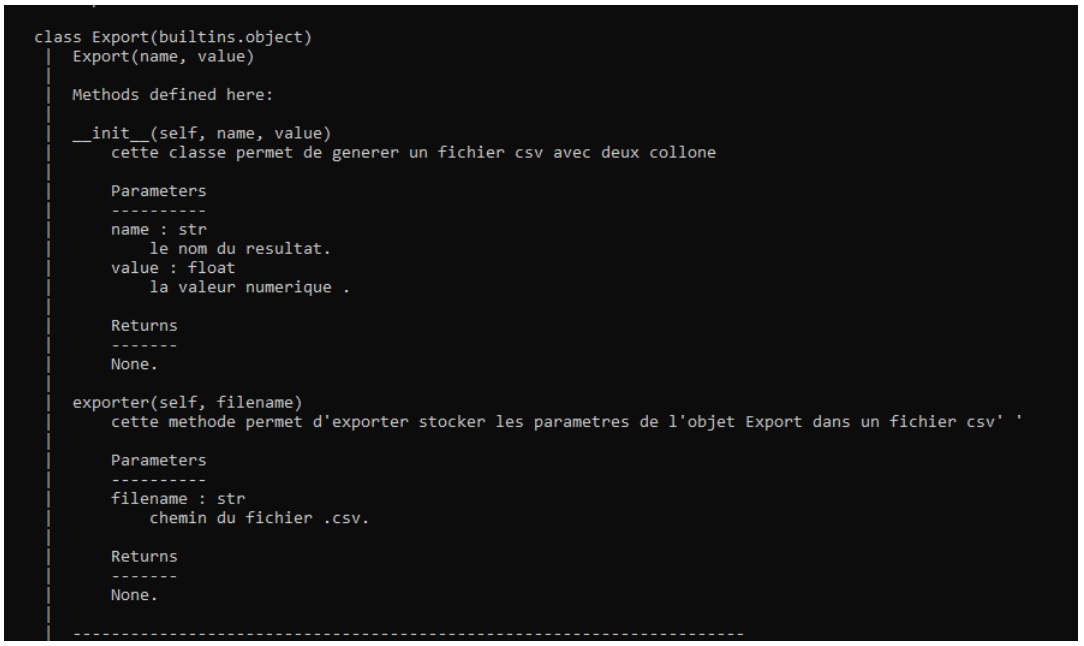
FIGURE 17 – Documentation de la classe data_centre_reduire :

7.12 Documentation de la classe Kmeans

```
class KMeans(builtins.object)
 |  KMeans(K, max_iteration=100, graph=False)
 |
 |   Cette classe est dediee a la methode de clustering des k means.
 |
 |  Attributes
 |  ----------
 |  K: int
 |      nombre de groupes Ó generer
 |  max_iteration: int
 |      nombre maximal d'iterations (100 par defaut)
 |  graph: booleen
 |      affiche ou non les graphiques (False par defaut)
 |  clusters:list
 |      liste des clusters
 |  centroids:list
 |      liste des centres de chaque cluster
 |
 |  Methods
 |  -------
 |  predict(X)
 |      cree une liste de centroides aleatoires
 |      puis effectue les assignations afin de creer les groupes souhaites
 |  get_cluster_labels(clusters)
 |      Permet de gÚnÚrer les labels des clusters
 |  create_clusters(centroids)
 |      cree les groupes a partir des centroides labelises
 |  closest_centroid(sample,centroids)
 |      utilise la fonction des distances euclidiennes pour determiner
 |      le centroide le plus proche des points
 |  get_centroids(clusters)
 |      permet de calculer le centroide d'un groupe
 |  is_converged(centroids_old,centroids)
 |      pour savoir si les deux centroides sont convergents entre eux
 |  plot()
 |      fonction pour afficher les resultats de maniÞre graphique
 |
```

FIGURE 18 – Documentation de la classe Kmeans

7.13 Documentation de la classe Export



```
class Export(builtins.object)
 |  Export(name, value)
 |
 |  Methods defined here:
 |
 |  __init__(self, name, value)
 |      cette classe permet de generer un fichier csv avec deux collone
 |
 |      Parameters
 |      ----------
 |      name : str
 |          le nom du resultat.
 |      value : float
 |          la valeur numerique .
 |
 |      Returns
 |      -------
 |      None.
 |
 |  exporter(self, filename)
 |      cette methode permet d'exporter stocker les parametres de l'objet Export dans un fichier csv' '
 |
 |      Parameters
 |      ----------
 |      filename : str
 |          chemin du fichier .csv.
 |
 |      Returns
 |      -------
 |      None.
 |
 |  ----------------------------------------------------------------------
```

FIGURE 19 – Documentation de la classe Export