Projet transverse

Auteurs :
COMBEAU Malo
FROMENT Félix
LABAT Valentin
PONCET Camille

Encadrant :
DE COURCHELLE Inès

Sommaire

I - Réflexions sur le sujet

Les minutes après la présentation du sujet, nous avons organisé une réunion avec tous les membres de notre groupe, afin de réfléchir dessus, et d'organiser notre temps.

La réflexion principale portait sur le stockage des données. Etant donnée nos connaissances sur le sujet, et les facilités d'usage du langage, nous avons décidé d'utiliser les bases de données sql. De plus, afin d'offrir une meilleure expérience utilisateur, nous avons décidé d'implémenter une interface graphique.

En fonction de nos facultés et de nos préférences, nous nous sommes donc réparti les tâches comme suit :

- **Malo :** Histogrammes, carte de la France et diagramme de classe
- **Félix :** Squelette, fonctions de base et interface graphique
- **Valentin :** Base de données, liaison Java/BDD et datamining
- **Camille :** fonction cheminLePlusCourt() et datamining

Concernant la communication, nous nous sommes tenus au courant de nos avancements par la messagerie Facebook (qui a surtout servi à répondre aux questions). Nous avons aussi fait une réunion tous les trois jours pour se tenir au courant et voir comment fusionner les travaux finis. Le planning initial a plutôt été bien suivi.

Cependant, après les problèmes liés à l'accident de Félix, nous nous sommes répartis les tâches qui lui étaient désignées. Ainsi, Malo s'est vu attribué la suite de la construction du squelette Java, ainsi que le début des fonctions de base Java, et Valentin la fin des fonctions de base et l'interface graphique.

Choix effectués :

Tout au long de ce projet, nous avons dû effectuer des choix, notamment sur la façon de représenter l'épidémie et sa propagation. Nous les expliquerons ci-dessous, et nous indiquerons dans quelle manière et lesquels de ces paramètres pourront être modifiés par l'utilisateur, afin de moduler sa recherche selon ces choix. Notons tout d'abord que la majeure partie des choix étudiés ci-après sont implémentés dans la classe Scénario, qui sera décrite au chapitre concernant l'implémentation de notre problème en JAVA. Dans le cas contraire, cela sera précisé.

1) Détermination du confinement

Dans notre projet, le confinement sera représenté par un 0 ou un 1, à la manière d'une représentation booléenne, ou encore du langage binaire. Ainsi, afin d'initialiser les communes confinées et celles qui ne le sont pas lors du tour 0, nous avons choisi de

produire un nombre aléatoire en 0 et 1 pour chaque commune, puis d'arrondir celui-ci à l'entier le plus proche. Nous obtenons ainsi des 0 et des 1 aléatoirement.

2) Durée du confinement

Lorsqu'une commune devient confinée, il faut choisir une date de confinement. En nous inspirant de la réalité et en voyant que les durées de confinement choisies étaient différentes en fonction des pays, nous avons décidé de ne pas déterminer une durée fixe pour le confinement, mais plutôt de faire une durée aléatoire entre deux bornes. Pour la borne inférieure, étant donné que les symptômes du COVID-19 se déclarent au bout de deux semaines, nous avons décidé de fixer la durée minimale de confinement à **20 jours**. La durée maximale a été fixée à **60 jours**, du fait que la durée du premier confinement dans la plupart des pays ait été égale à ce nombre de jours. Ces paramètres ont été choisis en fonction de critère personnels et peuvent évidemment être modifiés par l'utilisateur, en modifiant les variables associées dans les fonctions déroulement et initialisation.

3) Précision du terme "sain"

Le virus n'étant pas encore totalement connu par les chercheurs et le personnel médical, on ne sait pas vraiment si le virus peut être attrapé à plusieurs reprises ou non. Nous avons donc choisi de définir qu'on ne peut attraper le virus qu'une seule fois. Ainsi, le nombre de sains considéré dans notre base de données ne comporte pas les guéris, et on a la relation telle que :

populationOriginelle = totalMorts + totalGueris + totalSains + totalContamines

4) Précision du terme "morts"

Notre étude, pour des raisons de facilité de traitement de données, considère seulement les morts liés au virus, et non les morts naturelles. On considère que, sur l'échantillon considéré, il n'y a pas de morts naturelles. Ainsi, seules les personnes qui sont infectées peuvent mourir.

5) Pourcentage de personnes contaminées

Lors de l'initialisation de l'étude, il était nécessaire de définir le nombre de personnes contaminées dans chaque commune. Notre idée a donc été que, si une commune est confinée, alors c'est qu'elle possède logiquement beaucoup de personnes contaminées. Ainsi, une commune confinée possède, à l'initialisation **entre 40 et 100%** de sa population contaminée, tandis qu'une commune non confinée en possède entre **0 et 39%**. Une nouvelle

fois, ces chiffres ont été choisis arbitrairement et pourront être modifiés par l'utilisateur en modifiant ces valeurs dans les fonctions initialisation.

6) Pas de dégâts lors de l'initialisation

Nous avons choisi de ne pas générer de morts ni de guéris lors de l'initialisation, en la considérant comme le jour 0, et donc le paramètre d'entrée de notre fonction, comme si l'étude commençait au moment de la première itération du déroulement, et donc au jour 1.

7) Augmentation du nombre de contaminés

Cette augmentation représente l'augmentation de nouveaux cas découverts chaque jours. Là encore, nous avons dû choisir des critères. Comme plus il y a de personnes contaminées, plus le virus se transmet, nous avons fait en sorte que le nombre de nouveaux contaminés entre deux jours consécutifs augmente de 19%. De plus, comme le confinement réduit le nombre de contacts entre individus, nous avons décidé que le nombre de nouveaux contaminés entre deux jours consécutifs diminue chaque jour de 10%. Ces chiffres peuvent être modifiés par l'utilisateur dans les fonctions déroulement.

8) Nombre de morts et de guérisons

L'indice de létalité d'une maladie ne dépendant pas du nombre d'infectés (un virus avec un indice de létalité de 10% tuera 10% d'une population de 100 personnes, mais également 10% d'une population de 200 personnes), nous avons décidé de fixer celui-là. Cependant, il est important de remarquer que le COVID-19 est plus souvent létal chez les personnes vulnérables, et très peu chez les personnes en pleine possession de leurs moyens. Cependant, ce paramètre, au même titre que celui de l'âge de la population n'apparaît pas dans notre étude. Donc, le virus , en réalité, n'a pas le même indice de létalité en fonction de la population concernée, ce qui n'est pas le cas ici. Notons tout de même que les chiffres utilisés ici ne sont pas aberrants, et ont été choisis en fonction de l'étude des résultats du virus que l'on peut retrouver sur le site du gouvernement français.
Pour ce qui est du nombre de guérisons, nous avons plus ou moins suivi le même raisonnement, en affectant un pourcentage de nouveaux guéris chaque jour en fonction de l'effectif contaminé. Ainsi, 15% des contaminés guérissent chaque jour dans notre cas. Ces deux pourcentages sont des paramètres changeables par l'utilisateur dans les fonctions déroulement.

9) Condition de confinement

Notre étude nous imposait de déterminer un moment à partir duquel une commune passait de non confinée à confinée. Nous avons déterminé cette condition telle que, si la commune possède moins de deux fois plus de sains que de contaminés, alors la commune passait comme commune confinée.

II - Réflexion sur les distances

La réflexion sur les distances a été un des éléments essentiels des premiers jours de notre projet. En effet, la donnée des villes nous a été fournie par les professeurs deux jours après le début du projet, mais une donnée manquait toujours. En effet, en ayant simplement la superficie et les coordonnées de chaque ville, on ne pouvait pas définir un plan viable du pays, et on ne pouvait ainsi pas prévoir si en allant d'une ville à une autre, on passait par une troisième ou pas, car on n'avait pas la forme de chaque commune, ou l'ensemble des communes adjacentes. Ainsi, cette question posée au corps enseignant a entraîné la réponse suivante : "Nous considérons une matrice symétrique de distances". Avec cette réponse, on pouvait donc considérer aller de Paris à Marseille sans passer par d'autres villes, ce qui nous semblait étrange et peu représentatif de la réalité. Et ce qui rendait également la fonction plusCourtChemin(), qui avait pour but de chercher la distance minimale entre deux communes non confinées (sans passer par de commune confinée), inutile ; la distance renvoyée aurait en effet toujours été la ligne droite.

C'est pourquoi, lors de l'une de nos communications avec le corps enseignant de Cergy, nous avons posé la même question. La réponse a été toute autre, et on nous a donc invité à chercher la liste des communes adjacentes sur internet. Grâce à ces données, nous avons pu établir la distance réelle minimale entre deux communes non confinées, en passant de communes adjacentes à d'autres.

1) Distance entre deux communes

Après avoir mis au clair la notion de communes adjacentes, nous devions calculer la distance entre de telles communes. Pour cela, nous avions les coordonnées géodésiques de chaque commune à disposition: latitude et longitude en degrés.

Plusieurs méthodes de calcul étaient alors disponibles; de la moins précise avec la méthode de Pythagore considérant la Terre comme plate à la plus précise avec la formule de Haversine calculant la distance du grand cercle entre deux points. Cette dernière permet d'avoir 4 chiffres après la virgule ce qui donne une précision optimale. Néanmoins, afin de garantir un temps de calcul raisonnable et des résultats précis à deux chiffres après la virgule, nous avons choisi d'utiliser la formule se basant sur la loi des sinus. Celle-ci est un juste milieu entre rapidité et précision. Elle considére que le rayon de courbure de la Terre est identique en tout point contrairement à la formule de Haversine qui le calcul localement.

2) Algorithme du plus court chemin

Une des missions de notre projet a été de trouver le plus court chemin entre deux communes données. Afin d'implémenter cette fonction, nous avons utilisé l'algorithme de Dijkstra. Celui-ci permet de calculer les plus courts chemins à partir d'une source vers tous les autres sommets dans un graphe pondéré pouvant être orienté. Dans notre cas, nous avons considéré chaque commune comme un noeud du graphe et chaque distance entre commune adjacentes comme le poids des arcs entre les noeuds. Un chemin entre deux communes adjacentes pouvant être emprunté dans les deux sens, nous avons un graphe non orienté, ce qui permet de réduire les contraintes du problème.

L'implémentation de l'algorithme consiste en la construction progressive d'un sous-graphe dans lequel sont classées les différentes communes par ordre croissant de leur distance minimale à la commune de départ. La distance à la source correspond à la somme des distances entre les communes empruntés. L'algorithme prend en entrée une liste de commune sous forme d'une table hachée (*hashSet*). Parcourir une table contenant toutes le communes de France étant trop coûteux, nous avons fait le choix de garder les communes se trouvant dans un département donné, ici l'Ain.

a) Initialisation de l'algorithme

Concernant l'algorithme en lui-même, nous avons commencé par considérer que les distances entre chaque commune par rapport à la source étaient infinies (MAX_VALUE) et que la distance à la source elle-même était nulle. Le sous-graphe de départ est donc vide.

b) Déroulement de l'algorithme

Au cours de chaque itération, on choisit en dehors du sous-graphe un sommet de distance minimale et on l'ajoute au sous-graphe. Ensuite, on met à jour les distances des sommets voisins de celui ajouté. La mise à jour s'opère comme suit : la nouvelle distance du sommet voisin est le minimum entre la distance existante et celle obtenue en ajoutant le poids de l'arc entre sommet voisin et sommet ajouté à la distance du sommet ajouté. On continue ainsi jusqu'à épuisement des sommets (ou jusqu'à sélection du sommet d'arrivée).

c) Spécificités

Dans le cadre du sujet, les communes confinées ne pouvaient pas être traversées. De ce fait, nous les avons retirées du graphe initial pour ne pas les considérer comme des points d'arrêts au cours de l'algorithme.

Par ailleurs, certaines communes n'avaient visiblement pas de communes adjacentes. Cela est dû aux correspondances entre les base de données des communes Françaises et celle des communes adjacentes qui sont inexactes pour certaines communes. Par exemple, les communes "Le Poizat" et "Lalleyriat" ont fusionné en 2016, alors qu'elles sont présentes en tant que deux communes distinctes dans l'une des deux bases de données. Ainsi, il n'existe pas de chemin pour se rendre dans ces communes puisqu'elles n'existent pas dans l'une

des bases de données. Cette problématique de correspondance concerne une vingtaine de communes sur les 419 communes de l'Ain étudiées. Nous avons décidé de les évincer.
L'algorithme fonctionne mal avec les DROM.

III - Partie JAVA

Dans cette partie seront détaillées les classes de notre Projet. Les informations basiques sont déjà présentes dans les commentaires Javadoc, mais ici, il y aura plus de détails et certains choix seront expliqués.

1) Classe Commune

La classe Commune d'écrit les communes. Elle est composée du **nom** de la commune, de sa **population**, de sa **surface**, de ses coordonnées (**longitude** et **latitude**), le **nombre de décès en 24h**, le **nombre de nouveaux contaminées**, le **nombre de guérisons** et la **liste des historiques** des communes.
Elle possède les fonctions d'ajout, de suppression, et de modification des communes. Ces fonctions permettent de faire ces actions directement sur la base de données.

2) Classe Adjacent

Cette classe est composé des attributs nom, numéro insee, nombre de voisins, numéros insee des voisins et noms des voisins. Elle permet d'accéder à un document .csv répertoriant les communes adjacentes de chaque commune. Elle possède simplement la procédure toString() dans sa structure.

3) Classe ComConf et ComNonConf

Ces deux classes décrivent respectivement les communes confinées et les communes non confinées. Elles étendent la classe Commune, et possèdent, en plus la **date de confinement** et la **durée prévisionnelle** du confinement pour la première et la date du relevé pour la seconde.

4) Classe Date

Cette classe permet de créer une date LocalDate à partir des dates que l'on utilise dans nos projets, qui sont sous forme de chaînes de caractère.

5) Classe Departement

Composée du **nom** et du **code** du département, ainsi que de la **liste des communes** qui le constitue cette classe possède les fonctions permettant d'ajouter, de supprimer et de modifier des départements dans la base de données. Cependant, un département possédant des communes, un suppresion d'un département nécessite tout d'abord la suppression des communes qui la compose. On effectue donc une suppression en cascade, qui supprime d'abord toutes les communes du département, avant de supprimer le département en lui-même.

6) Classe Dijkstra

Cette classe permet de mettre en oeuvre l'algorithme du plus court chemin expliqué précédemment. Elle contient deux attributs : les communes avec une distance connue de la source et celles avec une distance encore inconnue.
De plus, trois fonctions permettent de mettre en oeuvre l'algorithme de dijkstra. L'une d'elle renvoie la commune avec la distance à la source la plus basse des communes encore non visitées, tandis que l'autre compare la distance réelle à la distance calculée entre deux communes.

7) Classe Distance

Composée de deux communes (une source et une destination) et de la distance en km entre ces deux communes, cette classe nous permet de calculer les distances entre chaque commune. L'unique fonction de cette classe retourne les distances entre une commune et toutes ses communes adjacentes. A noter que la distance renvoyée entre deux communes est -1 si le trajet n'est pas possible.

8) Classe Graphe

Cette classe est composé d'un unique attribut qui est un *set* de communes. La procédure ***Ajouter une commune*** permet de compléter le graphe. Cette classe est utilisée pour initialiser l'algorithme de Dijkstra.

9) Classe Histogramme

Voir plus bas dans le chapitre Histogrammes.

10) Classe Historique

Cette classe comporte tous les relevés qui sont effectués toutes les 24 heures, ainsi que la donnée si la ville est confinée ou non.

11) Classe Region

Comme les classes Departement et Communes décrites plus haut, cette classe possède des fonctions permettant d'ajouter, modifier et supprimer des régions dans la base de données, avec, une nouvelle fois, une suppression en cascade.

12) Classe Scenario

La classe Scénario est une classe centrale. En effet, c'est dans celle-ci que se trouvent les deux fonctions qui permettent le déroulement de la simulation, grâce aux fonctions initialisation et déroulement. Ces deux fonctions mentionnées juste avant existent en deux exemplaires. Le premier couple permet d'effectuer la simulation sur un échantillon composé de nombreCommunes communes. Les communes possédant l'id compris entre 1 et ce nombre là feront parti de la simulation. Ainsi, chaque personne pourra moduler l'amplitude de son étude en fonction des capacités de son PC et du temps qu'il veut y consacrer. Le deuxième couple permet de choisir les régions impactées par l'étude, en fonction de leur id. Ainsi, l'utilisateur peut choisir les régions qu'il veut étudier, et faire une étude qui permettrait de déterminer des clusters et qui rendrait la fonction distance bien plus logique et réelle. C'est pour cette raison que c'est la deuxième solution que nous avons directement implémenté cette fonction dans notre programme, mais l'utilisateur pourra très bien modifier ce choix et considérer le premier.
Outre ces deux fonction, il y a aussi la fonction vider(), qui permet de vider les tables Historique et Total, et d'ainsi recommencer une étude sans avoir à recharger le fichier d'origine contenant toutes les structures des tables et le remplissage des données "classiques". On retrouve également la fonction dateSuivante, qui permet de renvoyer la date du lendemain sous forme de chaîne de caractère en fonction de la date du jour sous forme de chaîne de caractère elle aussi. Enfin, la fonction bissextile() vient compléter cette fonction décrite plus haut, afin de gérer le cas où une année est bissextile ou non.
De plus, la fonction simulation permet d'effectuer la simulation. C'est également la fonction qui est appelé par l'interface graphique. C'est donc la fonction de substitution à la fonction main qui se trouve dans la classe Test, et qui sert pour l'exécution sans l'interface.

13) Classe Test

La classe Test est celle qui possède la fonction main. C'est celle que l'on utilise comme classe principale lorsque l'on n'utilise pas l'interface graphique. C'est donc ici que l'on modifie la date de départ, le code des régions que l'on veut utiliser, ainsi que le nombre de jours de notre simulation. Notons que la fonction déroulement possède seulement trois régions en paramètre, pour une question d'esthétisme. Pour faire plus de régions, il faut donc dupliquer l'instruction de l'appel de la fonction déroulement, en changeant les codes en paramètres.

14) Interface graphique

L'interface graphique permet de modifier, ajouter et supprimer des départements, régions, et communes, ainsi que lancer une simulation et vider les tables *Historique* et *Total*, grâce aux fonctions définies précédemment. Elle est situé dans la classe myWindow et c'est sa fonction main qui est appelée lors de l'exécution de l'interface graphique.

IV - Partie base de données

Pour un traitement plus facile des données, nous avons donc décidé de créer une base de données. *Vous trouverez cette base de données dans le dossier BDD*. Pour cela, plusieurs actions ont dû être effectuées, et elles seront détaillées ci-dessous.

1) Organisation des fichiers

Etant donnée la multitude de données que nous avons été amené à traiter (notamment les données de la table Commune qui fait plus de 36000 lignes !), nous avons opté pour une organisation claire et simple.

Le squelette de la BDD se trouve donc dans le fichier **bdd.sql**. Ce fichier créé une nouvelle BDD du nom de Projet (**en la supprimant auparavant si elle existe déjà**). Ensuite, elle rentre dans la BDD créée, puis appelle chaque fichier dans le dossier *Tables*, afin de créer les tables du projet, puis chaque fichier dans le dossier *Données*, afin de remplir les tables du projet des données originelles. Chaque table qui sera décrite ci-après possède donc obligatoirement un fichier de création de structure dans le dossier *Fichier* et peut posséder un fichier d'ajout de donnés dans la base dans le dossier *Données*, si nous avons jugé nécessaire de le faire.

2) Récupération et modification de la table Commune

La table Commune, qui est la partie centrale de notre projet, ne pouvait être créée de toute pièces par nous-mêmes. C'est pourquoi nous avons récupéré des données officielles,

comportant toutes les villes françaises. Ainsi, nous avons récupéré les données nécessaires sur le site suivant :
https://sql.sh/736-base-donnees-villes-francaises?fbclid=IwAR2gaDfRA2zimtDbwZu4L8qGrNGwExO1S1K0zVWS8jvjmbTRK6W5_bjwZS4
Afin de nous faciliter au plus la tâche, nous avons ensuite pris les données directement sous formes sql, afin d'enlever l'étape de l'importation. Cependant, la table Commune en elle-même ne convenait pas. En effet, de nombreuses données étaient inutiles, et d'autres données manquaient.

Réflexion :

Premièrement, les noms utilisés dans la table ne convenaient pas. Nous avons donc tout d'abord modifié le nom de la table, qui était *villes_france_free* par *Commune*, afin d'avoir un nom cohérent avec l'application java.

Ensuite, nous avons remarqué que chaque nom de colonne commençait par "villes_". Grâce à l'éditeur de texte java et à son remplacement multiple, nous avons supprimé le préfixe dans toutes ces données. Par exemple, la colonne "villes_id" devenait ainsi "id". Ainsi, les données nous semblaient plus viables et simples d'utilisation.

Dans un troisième temps, nous avons supprimé l'ensemble des données inutiles. Ainsi, nous avons remarqué que plusieurs occurrences du nom de la commune existaient. Nous avons donc gardé la donné la plus simple, soit "**nom**", et les noms **slug**, **simple**, **réel**, **soundex** et **metaphone** ont été supprimés. D'autres données apparaissaient en plusieurs exemplaires. C'est le cas de la population et des coordonnées. Pour la population, nous avions le choix entre la population de **2010**, la population de **1999** et une prévision de la prévision de **2012**, arrondie à la centaine. Dans la logique, nous aurions choisi une population la plus proche possible de la date actuelle. Seulement, la prévision de date, avec son arrondi à la centaine, nous faisait perdre beaucoup de précision. C'est pourquoi nous avons gardé la population de 2010, et supprimé les deux autres. Pour les coordonnées, nous avons supprimé les altitudes, que nous avons décidé de négliger, et nous avons gardé les **latitudes et longitudes en degré**, les autres n'étaient que des répétitions dans des unités différentes, et la formule trouvée pour calculer les distances utilisaient des degrés. Enfin, des données n'auraient pas été utiles pour notre problème.Nous avons donc supprimé les colonnes **commune**, **code_commune**, **arrondissement**, **canton**, **amdi** et **densité**.

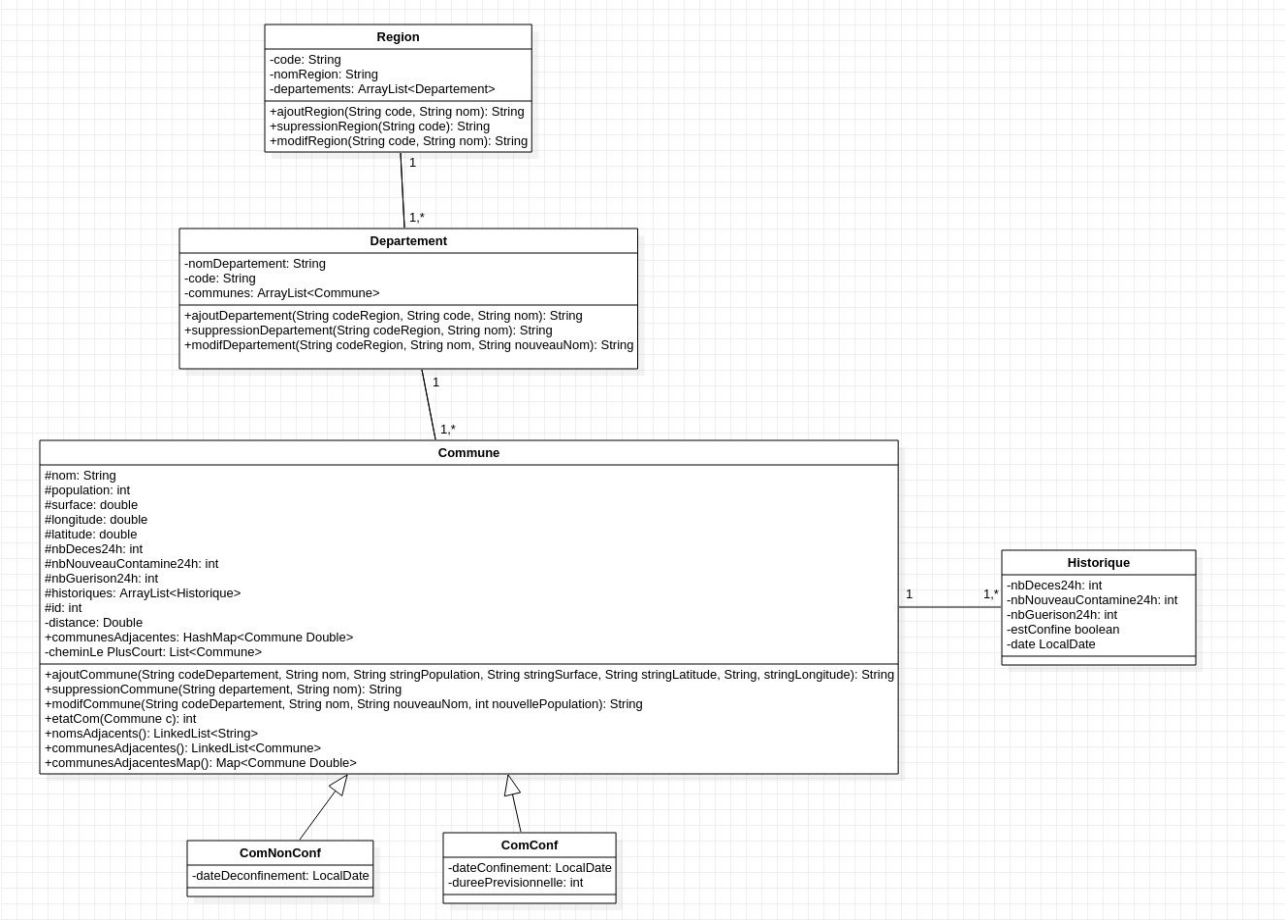
Enfin, nous avons ajouté certaines données qui manquaient dans cette table, tels que **id_confinee**, qui indique si la commune est confinée ou non, qui est une clé étrangère qui rappelle l'id de la table *Confinée*, table créée en amont, et détaillée ci-après. De plus, la donnée **date_dernier_confinement** a été ajoutée, afin de garder une trace du confinement le plus récent de la commune.

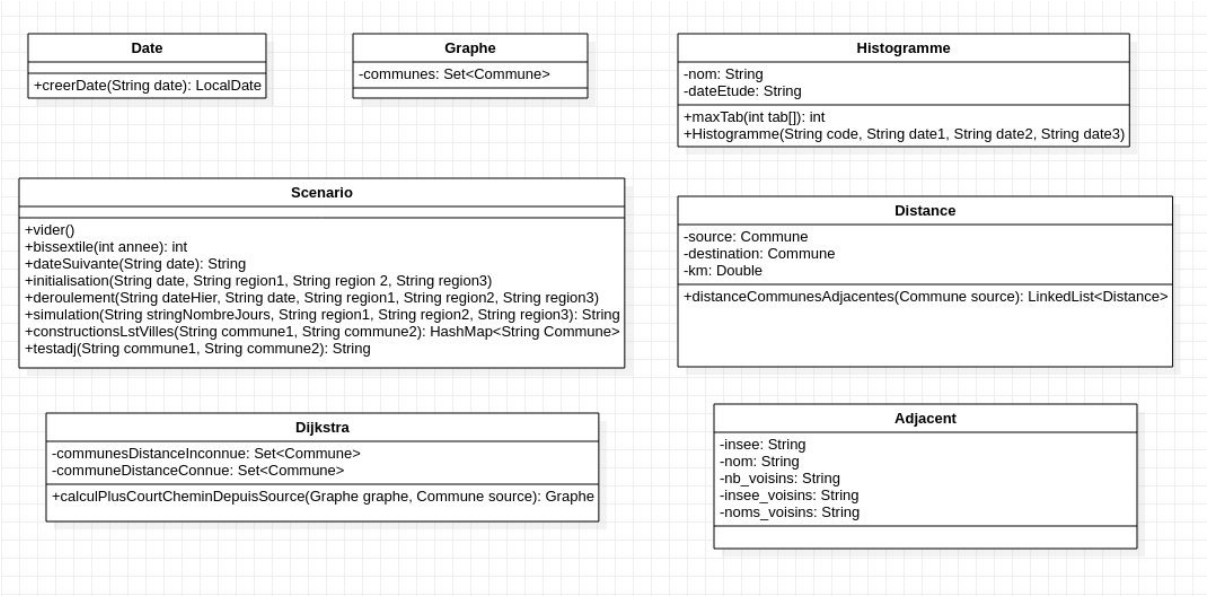
Implémentation :

Lors de l'implémentation des démarches décrites précédemment, deux solutions s'offraient à nous. Nous pouvions soit supprimer les données inutiles en supprimant les colonnes des tables, soit laisser les données telles qu'elles, pour supprimer les données inutiles ensuite, après que les données soient rentrées. Une nouvelle fois, bien que la première solution semble la plus logique, car nécessitant logiquement moins de traitement sql, nous avons choisi la seconde option, la première n'étant pas possible. En effet,

l'insertion des données dans la table étaient déjà présentes dans le fichier récupéré, et supprimer les colonnes signifiait supprimer les valeurs inutiles pour chaque ville, ce qui n'aurait pas été envisageable, car nécessitant une durée considérable. Le rapport temps/efficacité était bien meilleur en laissant les données telles qu'elles, puis en supprimant les colonnes inutiles, en utilisant la commande Sql **ALTER**, et en faisant de même pour l'ajout de données absentes. Etant donné la grande quantité de données, les suppressions de données prennent un peu de temps, mais ce temps, qui ne dépasse pas 8 secondes, est raisonnable. Alors, pourquoi ne pas laisser les données inutiles ? Tout simplement pour faciliter et accélérer les traitements postérieurs sur la table, et ne pas stocker des données d'aucune utilité.

Nous en sommes donc arrivé aux modélisations suivantes :





3) Table Confinée

Cette table, évoquée précédemment, nous permet de savoir si une commune est confinée ou non. Elle est composée d'un **id**, qui sert de clé étrangère dans la table *Commune*, et d'un **état**, une chaîne de caractères décrivant si la ville est confinée ou non.

4) Table Distance

Cette table permet, comme son nom l'indique, d'afficher la distance entre deux communes. Elle est constituée de deux **id**, référents aux id des deux *Communes* prises en compte. Ce couple officie comme clé primaire de la Table. La troisième et dernière donnée de cette table est la **distance** entre les deux villes.

5) Table Historique

L'historique est une donnée centrale dans notre étude. Nous avons ici choisi de décomposer en deux tables les évènements liés à la propagation de la maladie. Cette table *Historique* permet d'afficher les relevés quotidiens pour chaque ville en affichant, par ville, le nombre de **morts,** de **guéris,** et de **nouveaux contaminés,** ainsi que la **date du relevé**. Cette table possède de plus un **id_confinee**, clé étrangère vers la table *Confinée*, qui indique si la commune était confinée au moment de cette historique. Si oui, on retrouve la **date** où le confinement a commencé, ainsi que la **durée** restante avant la fin de celui-ci. Sinon, on retrouve la **date du déconfinement**. Les variables non concernées par chacun des deux cas valent alors NULL. Ainsi, si la Commune a été confinée puis déconfinée à deux reprises, on pourra retrouver les données pour chacun de ces confinements.

6) Table Total

Cette table compose la seconde partie de l'historique. Comme son nom l'indique, elle est composée du nombre total de **sains**, de **morts** et de **guéris** dans chaque ville, ainsi que la **date** à laquelle correspondent ces données.

7) Table Adjacent

Cette table contient les communes adjacentes à chaque commune. Elle possède comme clé primaire l'**id** de la commune, et a pour paramètre son **nom**, le **nombre de voisins** et leur **noms**, listés de la forme 'Voisin1|Voisin2|....|VoisinN' ainsi que **insee_voisins**, qui est organisée de la même manière que la liste des noms des voisins. Elle est tirée du fichier excel "Communes adjacentes au 1/1/2018' trouvable à l'adresse ci-dessous :
https://www.data.gouv.fr/fr/datasets/liste-des-adjacences-des-communes-francaises/?fbclid=IwAR1NH_mbTnplkOLNAcjoSQ-lkNlW9TeWHzq7MuLxGnHf2bgedFMKa05b130
Une fois ce fichier téléchargé, nous avons importé les données dans une table sql *Adjacent*, puis supprimé les données inutiles pour notre étude, soit la colonne **as$**.
Problèmes rencontrés : Lors de l'importation des données du fichier csv vers la base de données, nous avons d'abord eu un problème de Warnings. En effet, au début, il y en avait plusieurs milliers. Cependant, cette erreur a vite été résolue, en assignant une taille plus grande à la liste des noms et des codes des voisins. Cependant, il restait toujours 11 Warnings. Grâce à la commande **SHOW WARNINGS**, nous avons remarqué que le problème venait de 11 duplicatas de clé primaire. Etant donné le faible nombre de duplicatas, nous avons décidé de les traiter à la main. En se renseignant sur internet, nous avons vu que ces cas concernaient des villes qui avaient subies une fusion. Par exemple, Ornacieux et Ornacieux-Balbins possèdent le même id, car Ornacieux-Balbins réside de la fusion d'Ornacieux et de Balbins. Pour chacun de ces cas, nous avons donc gardé l'occurrence qui apparaissait dans la Table Commune, et supprimé l'autre, ainsi que toutes les apparitions de celle-ci dans le fichier csv.
Nous avons ainsi supprimé Ornacieux-Balbins, Abriès-Ristolas, Longeault-Pluvaut, Val-de-Virieu, Ancenis-Saint-Géréon, Belleville-en-Beaujolais, Deux-Grosnes, Porte-de-Savoie, Saint-Genix-les-Villages, Glières-Val-de-Borne et Vallières-sur-Fier;
Cette différence vient de la date de création de ces deux bases de données, et ainsi des fusions qui n'avaient pas encore eu lieu.

8) Tables Departement et Region

Ces tables sont tirées des tables téléchargées dans le fichier sql French-zip-code 3.0.0 à l'adresse suivante :

https://www.data.gouv.fr/fr/datasets/regions-departements-villes-et-villages-de-france-et-dout re-mer/

La table *Region* possède le **code** de la région et son **nom** ; les autres colonnes sont supprimées après l'ajout des données, pour la même raison que pour la table *Commune*. De la même manière, la table *Departement* est constituée du **code de la région**, qui est une clé étrangère vers la table *Region*, du **code du département**, et de son **nom**.

9) Autres modifications

Plusieurs modifications ont dû être effectuées, afin de rendre l'exploitation des données viable.

Premièrement, dans la base de données qui nous a permis de construire la table Commune, une erreur apparaissait dans l'intégration de données qui possédaient une apostrophe. Par exemple, en voulant rentrer Saint-Pierre-d'Irube, on insérait la valeur 'Saint-Pierre-d''Irube'. Une erreur survenait donc lors de l'utilisation de cette base de données, car Mysql croyait qu'on lui insérait deux chaînes de caractère pour une seule. Grâce une nouvelle fois à Atom et à son remplacement multiple, tous les caractères **' '** ont été remplacés par **\'**, ce qui a permit de résoudre le problème.

Deuxièmement, un erreur d'encodage apparaissait également lors de l'importation des données de la table citée précédemment. En cherchant la nature de l'erreur, nous avons pu remarquer que la version de l'encodage de notre table possédait des problèmes. En effet, selon nos recherches, l'UTF-8 utilisé anciennement dans sql ne comprenait pas l'ensemble des caractères. C'est pourquoi nous avons modifié l'encodage de notre table Commune vers **utf8_unicode_ci**, soit une version plus récente du codage UTF-8 en sql, ce qui nous a permit de supprimer cette erreur de codage.

Troisièmement, les tables importées avaient, à chaque fois, un surplus de guillemets. En effet, il y avait des guillemets autour du nom des tables, et autour du nom des données présentes dans les tables. Tous ces guillemet-là ont donc dû être supprimés manuellement.

V - Histogrammes

1) Sources

N'ayant jamais travaillé avec des histogrammes, nous avons dû faire des recherches sur les façons de faire et les packages à utiliser. Nous avons tout d'abord voulu essayer de faire selon ce lien :
https://jmdoudoux.developpez.com/cours/developpons/java/chap-bibliotheques-free.php
mais cela n'a jamais fonctionné, probablement pas disponible pour la version que nous avons. Nous nous sommes donc rabattu sur quelque chose de moins joli, mais plus simple à utiliser, avec le lien suivant :

https://sebastien-estienne.developpez.com/tutoriels/java/exercices-graphisme-java2d/?page=exo1
Il utilise la bibliothèque java.awt et javax que nous avons dû importer. Avoir un exemple ainsi que son code sous les yeux nous à permis de comprendre plus rapidement l'utilisation de ce package.

2) Variables globales

La classe Histogramme étant divisée en deux fonctions, nous avons dû utiliser des variables globales :

- des tableaux répertoriant les valeurs récupérées dans la base de données
- les maximums de chaque jeu de données
- les couleurs utilisées
- les dates de notre étude
- les valeurs concernants le calibration de nos barres d'histogrammes (marges, largeurs)
- deux incréments différents pour la hauteurs des barres en fonction des valeurs
- une fonction max qui renvoie simplement le max d'un tableau d'entiers

3) Fonction Histogramme

Cette fonction récupère les valeurs dans la base de données pour les stocker dans un tableau. Elle récupère en entrée le code de la région, ainsi que trois dates, qui correspondent aux trois barres des histogrammes pour voir l'évolution au cours du temps. On récupères huits informations :

- le nombre d'habitants de la région pour calculer le taux ensuite
- le nombre de communes de la région pour la même raison
- le nombre de communes confinées
- le nombre de communes non confinées
- le nombre de communes jamais confinées
- le nombre de nouvelles personnes contaminées dans cette région dans les 24h
- le nombre de nouvelles personnes décédées dans cette région dans les 24h
- le nombre de nouvelles personnes guéris dans cette région dans les 24h

On stocke ensuite les taux de chaque variables pour que ça soit plus visuel dans des tableaux, que nous utiliserons dans la fonction suivante. On détermine aussi les propriétés de la fenêtre qu'on ouvrira pour afficher les histogrammes.

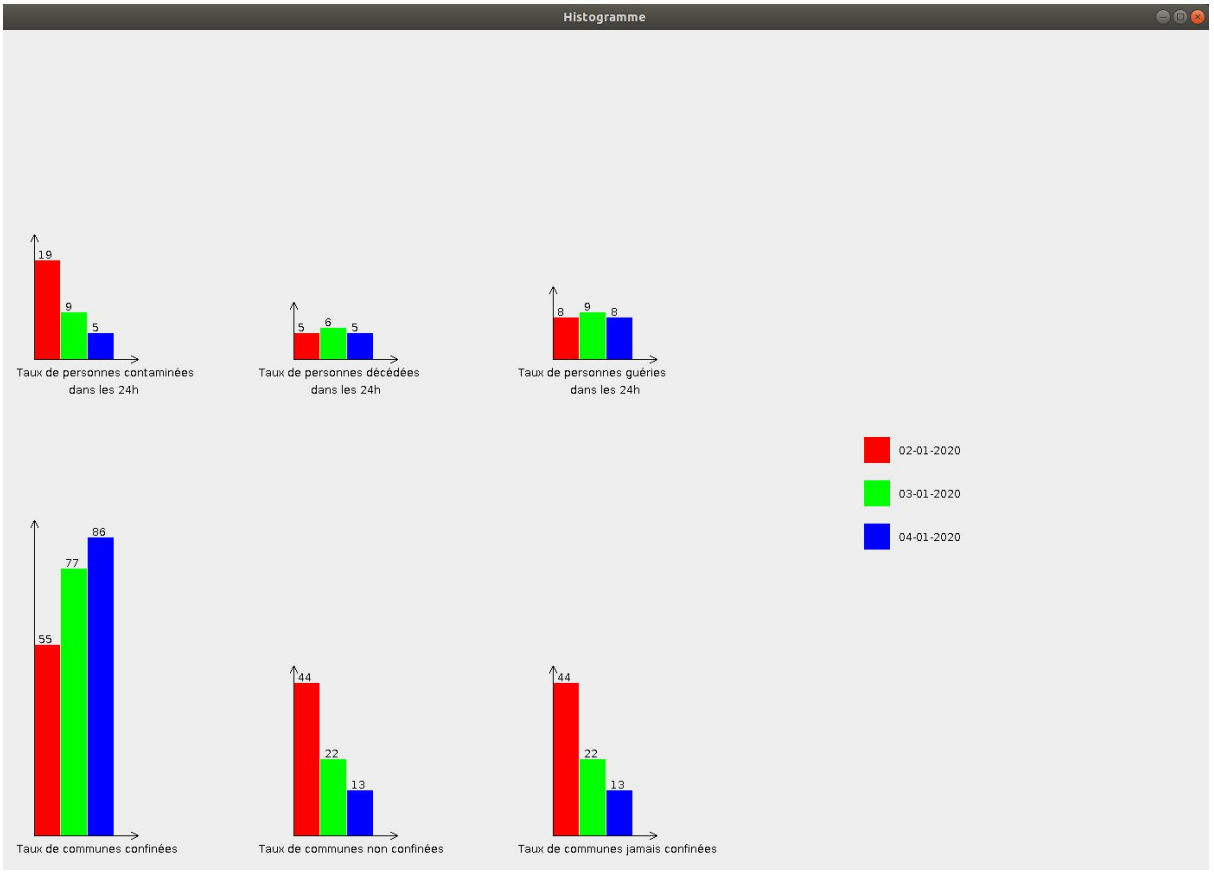
4) Redéfinition de la fonction paint

La deuxième fonction utilisée est la redéfinition de la fonction d'affichage paint. Elle prend en paramètre le graphique qu'on vient de définir ci-dessus. Pour chaque

histogramme, on lance une boucle qui nous affiche les trois barres de l'histogramme en fonction des tableaux de la fonction précédentes, les valeurs au dessus des barres, puis les axes X et Y ainsi que la légende. Cette partie est difficilement automatisable car on travaille avec des pixels : ça ne s'adapte pas en fonction de la taille de l'écran et c'est spécifique à chaque utilisation. On finit avec l'affichage de la légende puisqu'on a trouvé qu'afficher les dates aux dessus des barres faisait trop chargé.

5) Visuel

En exécutant cette classe, on se retrouve donc avec la fenêtre suivante :



On voit donc bien les six histogrammes sur la région donnée en départ : taux de personnes contaminés dans les 24h, taux de personnes décédées dans les 24h, taux de personnes guéries dans les 24h, taux de communes confinées, taux de communes non confinées, taux de communes jamais confinées. Les trois barres représentent, comme l'indique la légende à droite, les états en pourcentages aux dates données. Ces histogrammes nous permettent donc de suivre l'évolution.

VI - Data Mining

Après avoir réalisé la partie modélisation JAVA du projet, nous nous sommes penchés sur les données brutes liées au covid-19 afin de mettre en avant l'évolution de la pandémie, de son début à aujourd'hui. Notre étude s'est décomposée en deux grandes parties, commençant par la collecte de données pour finir par la prévision de l'évolution de la maladie, en passant par l'analyse des données brutes.

1) Collecte de données

Cette première partie peut sembler au premier abords rapide et peu coûteuse, mais elle nécessite un regard critique.

Le sujet nous donnant accès à quelques sources de données (sources gouvernementales et kaggle), nous avons commencer par étudier ces nombreuses données. Comme nous pouvions le prévoir, elles étaient très nombreuses, trop nombreuses pour réaliser une étude sur chacune d'entre elles. Nous avons donc fait un tri en conservant les données qui nous semblaient les plus exploitables dans la suite de notre étude. Par la suite, nous avons utilisé le logiciel R pour analyser ces données, notamment les nettoyer. Cela a consisté à éliminer les variables avec des données manquantes.

Cette première approche réalisée, nous sommes passés à l'exploitation des données.

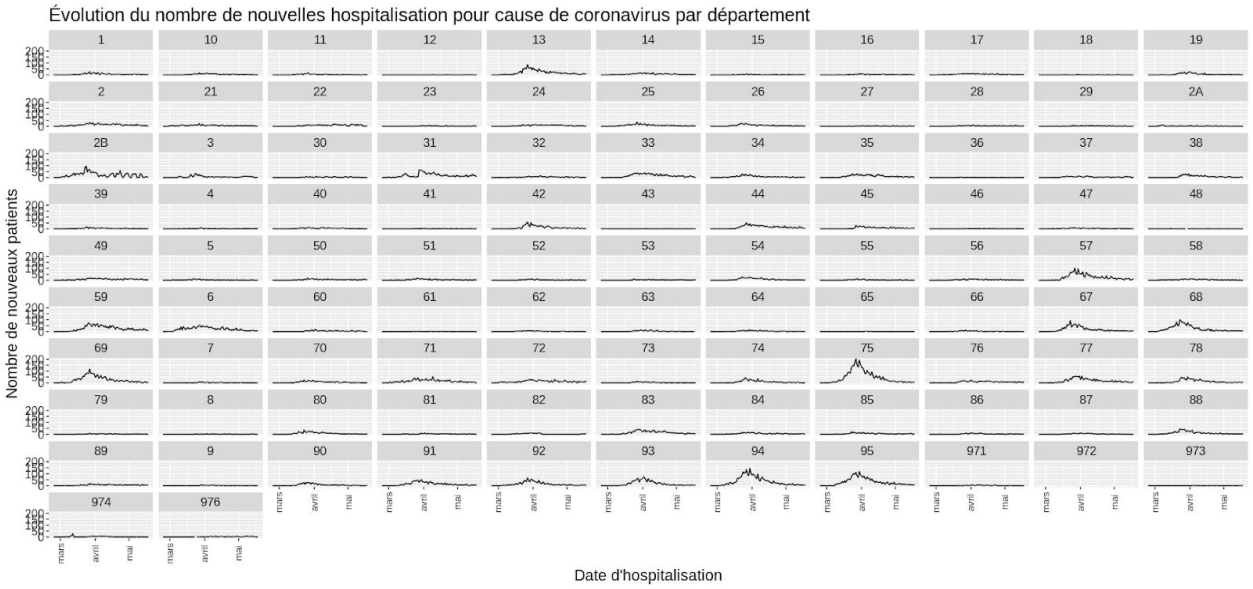
2) Statistique descriptive

Pour donner un ordre idées, un des jeux de données que nous possédions comportait 51 000 entrées avec une quinzaine de variables pour chaque entrée, soit un tableau de dimensions 51000*15 données. Bien qu'ayant tous les renseignements nécessaires dans ces jeux de données, ils ne sont absolument pas exploitables visuellement. Nous avons donc essayé de représenter ces données telle qu'elles soient interprétables au premier coup d'oeil.

1) Une évolution temporelle

Tout d'abord, nous nous sommes intéressés à l'évolution de la pandémie en France. Pour cela, un set de données représentant le nombre d'hospitalisation par département par jour nous a servi de base. Ainsi, nous avons tracé un graphique composé d'un courbe par département qui représente l'évolution du nombre d'hospitalisations pour cause de coronavirus allant du 24 février 2020 au 18 mai 2020.

Ci-dessous le graphique expliqué suivi du code l'ayant engendré.



(Voir ANNEXE 1 pour un agrandissement du diagramme)

Quels départements ont enregistré le plus d'hospitalisations ?

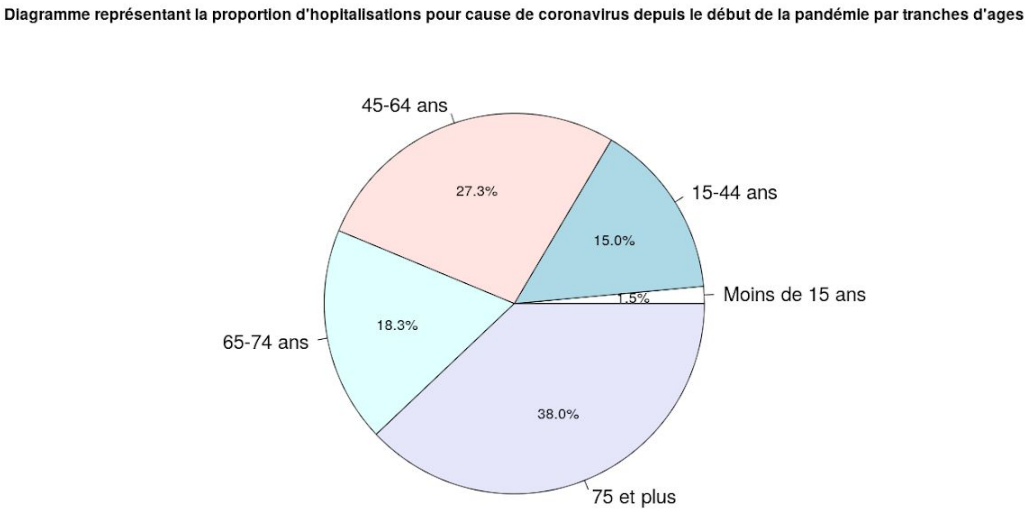
Ce graphique permet de mettre en avant les départements les plus touchés par l'épidémie. Parmi ceux-ci, il y la région Parisienne avec le 75, le 93, le 94, et le 95, le Rhône (69), la Moselle (57) et les Bouches-du-Rhônes (13).

Nous pouvons aussi apercevoir l'évolution de la pandémie dans ces départements : tandis que dans le 75, le pic a été franchi et l'épidémie semble avoir été rapidement freinée, la Haute-Corse (2B) subit des oscillations preuve d'une résurgence de la pandémie.

2) L'âge: un facteur de risque ?

Après avoir analysé l'évolution globale de la pandémie, nous nous sommes penchés sur le type de personne principalement touché par ce virus. Le premier critère étudié est l'âge des patients. Bien que nous ne disposions pas de l'âge exact de chaque patient, nous avions accès à la tranche d'âge à laquelle celui-ci appartenait. Ainsi, nous avons pu réaliser un diagramme "Camembert" représentant le pourcentage d'hospitalisations pour cause de coronavirus par tranche d'âge.

Ci-dessous le diagramme obtenu suivit du code lié.



Quelle tranche d'âge est la plus touchée par l'épidémie ?

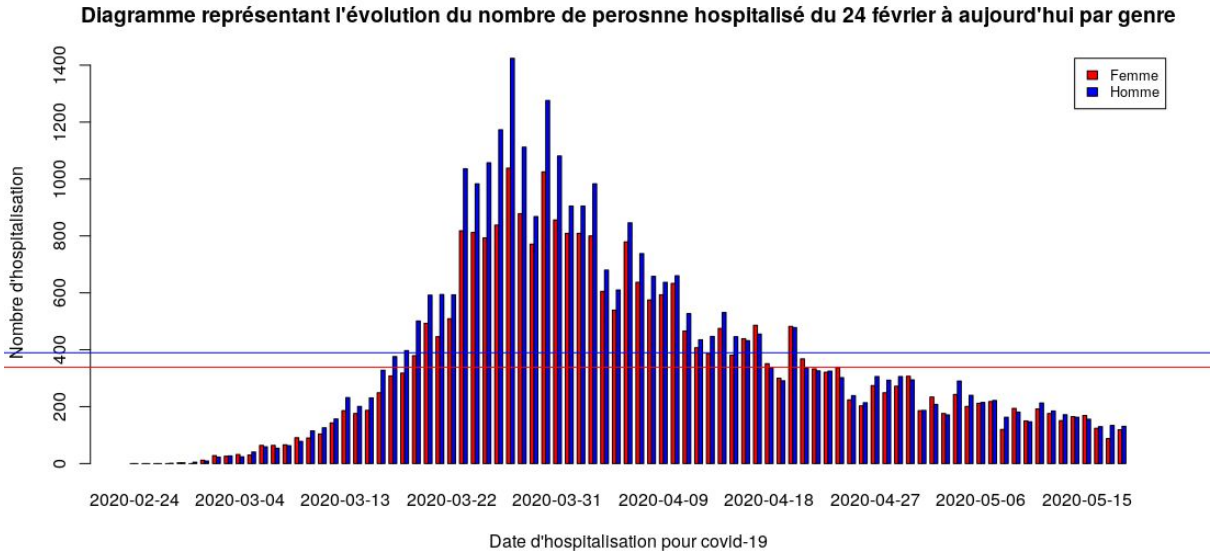
Ce graphique met en avant, de manière très claire, la proportion de personnes hospitalisées à cause du coronavirus. Nous pouvons observer que les moins de 15 ans représentent moins de 2 pourcents des patients tandis que les plus de 75 ans sont principalement touchés par le virus puisqu'ils représentent 38 pourcents des patients.

La vieillesse est donc un facteur de risque important dans cette épidémie.

3) Un genre plus vulnérable ?

Pour finir, nous avons comparé les proportions d'hospitalisations à cause du coronavirus selon le genre en France. Pour cela, nous avons tracé une diagramme en bâtons représentant le nombre d'hospitalisations pour chaque genre en fonction de la date; allant du 24 février 2020 au 18 mai 2020.

Ci-dessous la graphique et le code correspondant.



Est-ce que les hommes sont plus touchés par le coronavirus que les femmes ?

C'est à cette question que notre diagramme répond. Visuellement, la réponse est claire : les hommes ont en moyenne été plus touchés par le coronavirus que les femmes. Le pic de l'épidémie montre une différence flagrante entre les femmes et les hommes; 1424 hommes ont été hospitalisés le 27 mars 2020, alors que seulement 1038 femmes ont été concernées.

VII) Améliorations possibles

1) Précision du sujet

Il est à noter que certains points du sujets n'étaient pas clairement précisés. Nous pensons notamment à la notion de distance (comme expliqué en début de rapport), qui était cruciale afin de définir une base de données saine, et contenant les informations nécessaires à nos traitements. Les questions posées sur ce point ont, de plus, entraîné des réponses différentes en fonction des interlocuteurs. Quelques jours ont donc été perdus pour ce qui est de la réflexion, et de l'attente des réponses.

2) Histogrammes

Nos histogrammes n'ont que trois dates d'études par région , comme c'est traité pixel par pixel. Nous pourrions donc améliorer cela en rendu la chose un peu plus automatique, et en rendant le visuel un peu plus beau.

3) Carte

Comme la carte était présentée comme un bonus dans la vidéo de présentation du projet faite par les professeurs de Cergy, nous n'avons pas mis la priorité sur l'affichage de la carte de France. De plus, cette vidéo est apparue près de cinq jours après le début du projet et nous n'étions que trois personnes à pouvoir fournir un travail effectif à ce moment là. Nous avons donc essayé de travailler dessus, en nous documentant beaucoup notamment, lorsque les autres points étaient traités de façon satisfaisant, mais la complexité de la tâche et le peu de temps restant ne nous ont pas permi de l'implémenter.

4) Détails techniques

Il est certain que tous les aspects désirés n'ont pu être mis en place par manque de temps. En effet, un affichage dans l'interface graphique du contenant de la base de données aurait été appréciable, afin de ne pas obligatoirement intervertir entre l'interface et la fenêtre mysql.

Une fonctionnalité permettant de prévoir le temps que mettent les requêtes à s'effectuer aurait pu être un plus, notamment lorsque l'on travaille avec une base de données si grande.

Un paramètre important de l'expansion du virus est l'âge des individus et leur maladie. Ce paramètre aurait rendu l'étude plus juste.

Le temps d'importation et de traitement de données est très élevé étant donnée la quantité de données présentes. Cependant, certains paramètres comme le nombre de régions considérées permet de réduire ce temps.

Évolution du nombre de nouvelles hospitalisation pour cause de coronavirus par département
Nombre de nouveaux patients
Date d'hospitalisation
mars
avril
mai
1
2
2B
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
21
22
23
24
25
26
27
28
29
2A
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
971
972
973
974
976

ANNEXE1