Vincent Laloux

11205586

Résumé

Dans le cadre du cours de logiciels statistiques de la session d’hiver 2021, j’ai travaillé sur un mini-projet de programmation sur le logiciel R. L’objectif est d’utiliser plusieurs moyens d’agrégation des données d’une table. Je me suis ensuite penché sur les visualisations cartographiques.

Travail de session – R

Logiciels Statistiques

# BLOC #1

Introduction :

Avant de débuter ce projet, je pense important de mentionner que j’ai effectué un premier nettoyage succin pour obtenir toutes les observations liées au niveau géographique des CLSC. Pour rappel la base de données originale contient plusieurs régions administratives, qui comprennes plusieurs régions : En ordre décroissant de « granularité » nous avons les observations liées au Québec, RSS, RTS, RLS, CLSC. Pour chacune de ces régions, nous avons les observations de la population masculine, féminine, total, ainsi que par année. Nous avons 46 années dans notre base de données : de 1996 à 2041.

1. La fonction aggregate()

La fonction aggregate, bien que puissante, est relativement difficile d’utilisation. On verra ici que sa syntaxe peut varier et ses sorties également.

Partons des apprentissages réalisés dans le cadre du cours. Afin d’avoir le portrait de la population moyenne entre 1996 et 2041 par CLSC nous écrivons :

clsc\_base\_H=subset(clsc\_base,clsc\_base$Sexe=="Total")

test=aggregate(clsc\_base\_H$TousLesAges~clsc\_base\_H$Territoire,

FUN=mean,

na.rm = TRUE) # On perd le titre des colonnes

Nous obtenons alors la table suivante :

Remarques :

1. Nos colonnes sont mal étiquetées, ce qui nous force à les réetiqueter avec une nouvelle ligne de code comme :

colnames(test)=c("Territoire","Population")

1. Nous avons du entrer une ligne de code au préalable pour utiliser les observations qui utilisent le Total (Féminin+Masculin)
2. Nous ne sommes pas en mesure d’arrondir le résultat de notre agrégation. Pour pallier ce problème, nous devons utiliser la fonction liste pour définir notre variables référence (ici Territoire) pour l’agrégation des observations de la variable numérique (ici TousLesAge). Nous sommes en mesure de créer une focntion imbriqué qui pourra calculer l’arrondi de notre moyenne. Ainsi :

Test=aggregate(clsc\_base\_H$TousLesAges,list(clsc\_base\_H$Territoire),function(x) round(mean(x,0))) #utiliser list() nous permets d'executer une fonction qui prend cette fois la moyenne et son arrondie

Nous obtenons la table suivante :



Une bonne chose de faite ! Cependant, le nom de colonnes à cette fois changé : on remarque qu’en utilisant les listes, la fonction aggregate() donne automatiquement le nom Group.1 pour la variable de référence et x pour la variable à agréger.

Nous allons essayer maintenant d’alléger notre code pour ne pas avoir à préciser avant l’agrégation la table filtrée. Autrement dit nous allons filtré notre base de données directement dans la fonction aggregate(). Nous verrons également qu’il est facile d’ajouter des niveaux de granularité en ajoutant des arguments à la fonction de liste et que nous pouvons nommer le titre de nos colonnes pour obtenir en sortie les bons titres. C’est un autre avantage de la fonction list nous offre ici :

test<-aggregate(clsc\_base$TousLesAges,list(Territoire=clsc\_base$Territoire, Sexe=clsc\_base$Sexe=="Total"),function(x) round(mean(x,0))) #utiliser list() nous permets de selectionner les observations de notre colonne SEXE

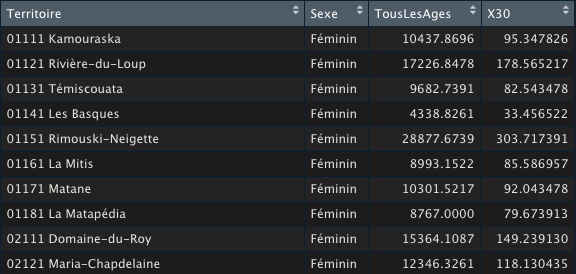
Une image contenant texte, capture d’écran, tableau de points, écran

Description générée automatiquement

On remarque que notre nouvelle colonne Group.2 retourne un booléen pour nous indiquer lorsque la condition de Sexe=="Total" est remplie. Pas idéal, mais sur beaucoup de données, il sera facile de faire une requête pour obtenir notre table finale.

Par ailleurs, on remarque que la dernière colonne n’a pas été nommé. Pour résoudre ce problème nous pouvons utiliser l’argument data dans la fonction aggregate() qui nous permettra de ne pas avoir à spécifier le jeu de données suivi d’un $, mais directement le nom de la variable que nous voulons utiliser. Notons que ce petit argument rend le code beaucoup plus lisible. En utilisant cbind() on est en mesure de spécifier plusieurs colonnes sur lesquelles appliquer l’agrégation (déjà vu en cours)

aggregate(cbind(TousLesAges,X30)~Territoire+Sexe, data=clsc\_base, FUN=mean, na.rm = TRUE) # On guarde le titre des colonnes



En revanche, cette syntaxe nous fait perdre l’utilité de la fonction liste décrite précédemment.

Au-delà de cet exemple, nous pouvons également utiliser différentes fonctions dans l’argument FUN. Ici ou va produire deux tables qui nous donnerons les résulates de quantiles les plus faible et les plus élevé. On note ici que l’on a utilisé la fonction substr pour n’obtenir les résultats que sur les différents RSS de la région (composé de 2 digit d’identification).

aggregate(clsc\_base\_T$TousLesAges, by=list(RSS=substr(clsc\_base\_T$Territoire,1,2)), FUN=quantile, probs=c(0.05,0.95))

Finalemement la fonction Lenght nous renseigne sur le nombre de ligne contenu par observation agréger.

aggregate(clsc\_base\_T$TousLesAges, by=list(RSS=substr(clsc\_base\_T$Territoire,1,2), Annee=clsc\_base\_T$Annee), FUN=length)

Une image contenant texte, calculatrice, noir

Description générée automatiquement

Ici nous pouvons par conséquent déduire que la région « 03 » compte 13 CLSC en 1996.

On remarque que cette syntaxe est relativement complexe à maitriser, et arriver au bon résultat demande beaucoup d’essaies erreur au début.

1. Librairie sqldf

Cette librairie nous permet d’utiliser la syntaxe de requêtes SQL pour produire des tableaux. Grâce à cette librairie, nous nous appliquerons à développer la hiérarchie des régions administratives du ministère de la santé.

Tout d’abord essayons de reproduire nos exemples ci-dessus :

sqldf("select RSS\_code, avg(TousLesAges) from clsc\_base group by RSS\_code")

Une image contenant table

Description générée automatiquement

Nous constatons que cette function nous permet d’executer une requête sql même si elle est comprise entre guillement (d’habitude réservée au charactères)

test=sqldf("select distinct RSS\_code, count(distinct RTS\_code) as RTS\_code, count(distinct RLS\_code) as RLS\_code, count(distinct CLSC\_code) as CLSC\_code from clsc\_base group by RSS\_code, Annee")

Une image contenant table

Description générée automatiquement

Nous constatons que notre requête SQL fonctionne bel et bien : nous obtenons le nombre de régions en fonction des région RSS !

1. Librairie dplyr

La librairie dplyr n’est plus à présenter, mais il s’agit d’un incontournable dans le traitement de données et notamment dans leur agrégation. Nous examinerons par conséquent diverses options d’agrégation de la fonction summarise.

Ce premier pipe de données nous permets de directement choisir les données que l’on souhaite dans utiliser (select()) et nous pouvons également les filtre (filer()). Comme nous l’avons vu en cours, pour agréger des données nous avons besoin de deux étapes dans le pipe ; group\_by et summarise. Group\_by nous renseigne sur la classe avec laquelle nous allons agréger nos données. Summarise contient les variables où nous agrégerons les données ainsi que la fonction d’agrégation que nous utiliserons. Notons ce premier pipe :

Quebec= clsc\_base %>%

select(CLSC\_code,RLS\_code,RTS\_code,RSS\_code, Annee, Sexe, TousLesAges, c(9:99)) %>%

filter(Sexe=="Total") %>%

group\_by(Annee) %>%

summarise(test=range(TousLesAges))

Avec dyplr nous sommes en mesure d’utiliser un nombre considérable de fonction : ici, nous obtenons pour chaque année la plus petite valeur observées (correspond à un clsc) ainsi que la plus grande :

Une image contenant texte, tableau de points

Description générée automatiquement

Un des avantages également et que nous n’avons pas pu expérimenter lors des précédents exemples, c’est de créer plusieurs fonctions d’agrégations sur une même ligne, pour différentes variables de notre dataframe de départ.

Quebec= clsc\_base %>%

select(CLSC\_code,RLS\_code,RTS\_code,RSS\_code, Annee, Sexe, TousLesAges, c(9:99)) %>%

filter(Sexe=="Total") %>%

group\_by(Annee) %>%

  summarise(tibble(min=min(TousLesAges),max=max(TousLesAges),moyenne=mean(TousLesAges),total=sum(TousLesAges)))

Une image contenant texte, tableau de points

Description générée automatiquement

Tout comme nous avons pu le faire dans la fonction aggregate nous pouvons également inclure des fonctions prédéfinies :

#function

quibble2 <- function(x, q = c(0.25, 0.5, 0.75)) {

tibble("{{ x }}" := quantile(x, q), "{{ x }}\_q" := q)

}

Quebec= clsc\_base %>%

select(CLSC\_code,RLS\_code,RTS\_code,RSS\_code, Annee, Sexe, TousLesAges, c(9:99)) %>%

filter(Sexe=="Total") %>%

group\_by(Annee) %>%

summarise(quibble2(TousLesAges, c(0.25, 0.5, 0.75)))

Une image contenant texte, tableau de points

Description générée automatiquement

Nous avons vu que dplyr nous donne plus de flexibilité, plus facilement que la fonction de base de R. Cependant il y a également autre chose que nous n’avons pas pu réussir, c’est d’agréger les 101 âges. A la main cela prendrai trop de temps. Heureusement dplyr nous permet de le faire selon deux manières :

Quebec= clsc\_base %>%

select(CLSC\_code,RLS\_code,RTS\_code,RSS\_code, Annee, Sexe, TousLesAges, c(9:99)) %>%

filter(Sexe=="Total") %>%

group\_by(Annee) %>%

summarise(across(everything(), list(mean)))

Une image contenant texte, moniteur, écran, noir

Description générée automatiquement

Ici la fonction across nous permet de définir les variables ainsi que la fonction qui doit leur être appliqués. Notons que les variables catégorielles sont désormais absentes (NA)

Quebec= clsc\_base %>%

select(CLSC\_code,RLS\_code,RTS\_code,RSS\_code, Annee, Sexe, TousLesAges, c(9:99)) %>%

filter(Sexe=="Total") %>%

group\_by(Annee) %>%

summarise\_all(.funs = c(mean="mean"))

Une image contenant texte, moniteur, écran, noir

Description générée automatiquement

# BLOC #2

Introduction :

Pour représenter graphiquement les différentes zones administratives du québec, j’ai dû au préalable charger plusieurs fichiers de données de type « shape files ». Ces fichiers de données doivent être lu au préalable grâce à la fonction st\_read() de la librairie sf.

J’ai pu importer deux je de données contenant des données de type geometry qui contiennent les formes des régions des RSS. Il est important de noter que lorsque l’on joint les données, nous devons spécifier en premier le fichier contenant les formes géométriques.

Un premier fichier contiendra la population du Québec sur les 18 région à l’année 2010.

Pour charger cet ensemble de données dans R, nous appelons la fonction st\_read de sf (toutes les fonctions du paquet sf commencent par le préfixe st\_, qui signifie spatiotemporel) et fournissons le chemin d'accès au fichier .shp.

1. La librairie GGPLOT2

La puissance de ggplot2 s’illustre par sa diversité d’application. J’ai en effet pu l’utiliser pour représenter mes données avec lesquelles nous avons travaillés au bloc précédent.

Cependant, en matière de carte, ggplot2 fut particulièrement long à exécuter. La taille et la complexité des formes géographiques en sont probablement la raison.

La première étape fut d’obtenir une carte représentant les régions RSS du québec grâce à la dimension geometry de notre dataframe. Lorsque l'on trace un ensemble de données vectorielles à partir d'un objet sf, on utilise la couche geom\_sf pour afficher les caractéristiques spatiales sur une carte. Il n'est pas nécessaire de spécifier les mappings x et y dans aes, puisque ceux-ci sont définis par l'objet sf lui-même. Les lignes du graticule sont également dessinées automatiquement.

ggplot(data=rssshp1) +

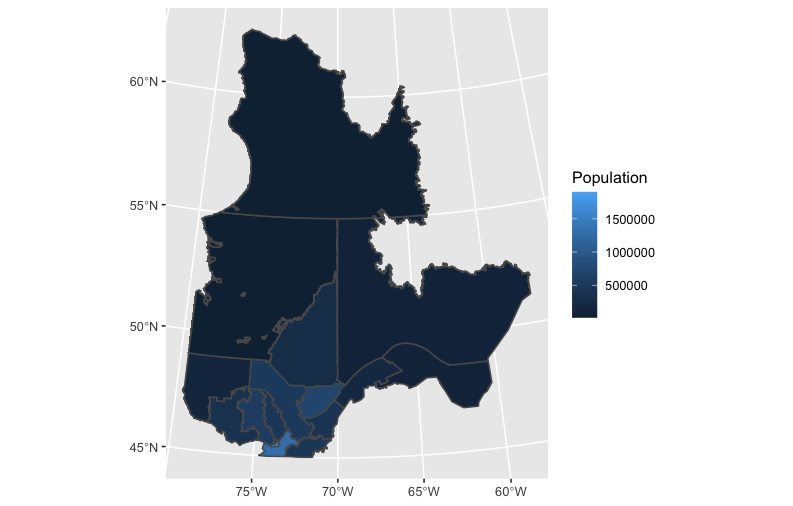
geom\_sf(aes(fill=RSS\_code))

Une image contenant carte

Description générée automatiquement

On voit que cette carte n’est pour l’instant pas très informative. Comme observé pendant le cours, ggplot2 fonctionne un peu à la manière dplyr, en ajoutant des étapes subséquentes à chaque ligne grâce au + (et non avec >%>).

Nous décidons alors d’utiliser notre variables population pour obtenir une carte de la répartition de la population sur le territoire du Québec.



On remarque que les couleurs sont relativement mal ajustés, on aimera pouvoir les inverser pour avoir les régions les plus peuplé en foncé et les moins peuplé en clair :

ggplot(data=rssshp1) +

geom\_sf(aes(fill=Population)) +

scale\_fill\_gradient (low= "#E69F00", high= "#56B4E9")

Une image contenant carte

Description générée automatiquement

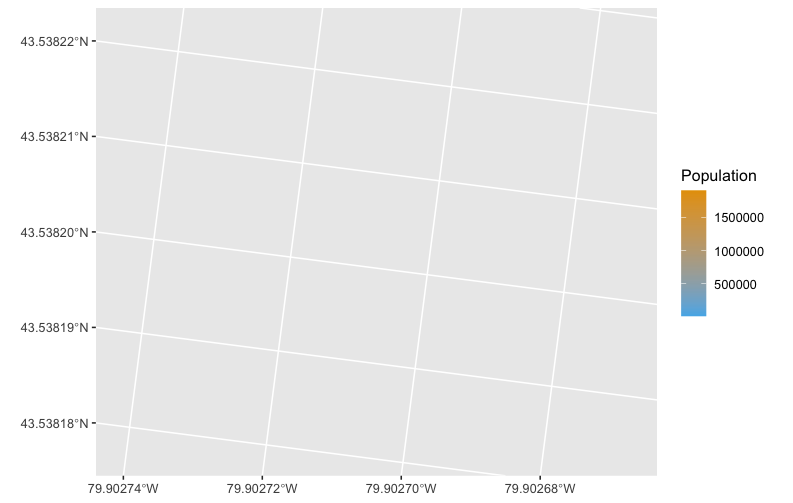
On observe que la région de montréal semble la plus peuplé, mais on a du mal à la distinguer. On aimera zoomer sur cette région. Cependant ggplot2 ne nous donne pas l’occasion d’interagir avec les graphes, il nous faudra donc préciser une nouvelle ligne de code qui nous permettra de préciser les coordonnées de notre fenêtre :

ggplot(data=rssshp1) +

geom\_sf(aes(fill=Population)) +

scale\_fill\_gradient (low= "#56B4E9", high= "#E69F00") +

coord\_sf(xlim = c(-76, -70), ylim = c(45, 50))



On peut voir que notre fenêtre est vide. En cause : le coordinate reference system (CRS) de notre jeu de données n’est pas bien référencé. Essayons-nous à adapter le CRS de notre dataframe :

mrc\_proj <- st\_transform(rssshp1, crs = 4269)

on peut vérifier le crs de notre nouveau dataframe avec la fonction suivante

st\_crs(mrc\_proj)

Nous pouvons réesayer de notre zoom précédent avec la fonction coord\_sf

ggplot(data = mrc\_proj) +

geom\_sf(aes(fill = Population)) +

scale\_fill\_gradient (low= "#56B4E9", high= "#E69F00") +

geom\_sf\_label(aes(label = Etiquette),label.size = 0.1) +

coord\_sf(xlim = c(-75, -70), ylim = c(45, 47)) +

annotate("point", x = -73.5, y = 45.5, colour = "red", size = 2)

Une image contenant carte

Description générée automatiquement

1. La librairie Tmap

Cette librairie c’est révélé plus agréable d’utilisation : en effet, au lieu d’afficher un graph, la visualisation peut se faire au format html dans la fenêtre viewer.

Mais tout d’abord, j’ai voulu reproduire nos étapes réalisées avec ggplot2. Dans un premier temps on peut observer la carte du Québec sans avoir précisé quelles variables seront utilisé pour la caractériser :

tm\_shape(rssshp1)+

tm\_polygons(id="RSS\_code")

Nous observons une carte étant presque en tous points similaires à notre première carte, au détail près que nous n’observons pas les lignes de longitude et latitude en arrière-plan.

Une image contenant carte

Description générée automatiquement

Avant de continuer, on va désormais utiliser le viewer de R. Pour cela il suffit de spécifier à tmap d’utiliser le viewer :

tmap\_mode("view")

Désormais nos cartes apparaîtrons dans le viewer. Nous essayons maintenant de reproduire notre carte avec la population totale.

tmap\_mode("view")

tm\_shape(rssshp1) +

tm\_polygons("Population", id="RSS\_code", palette="Oranges")

Nous observons un visuel, plus agréable qu’obtenue avec ggplot2. L’interface est beaucoup plus interactive et on peut interagir en consultant les informations de caques régions socio sanitaire (RSS). On peut également changer la couche de la carte.

Une image contenant carte

Description générée automatiquementUne image contenant carte

Description générée automatiquement