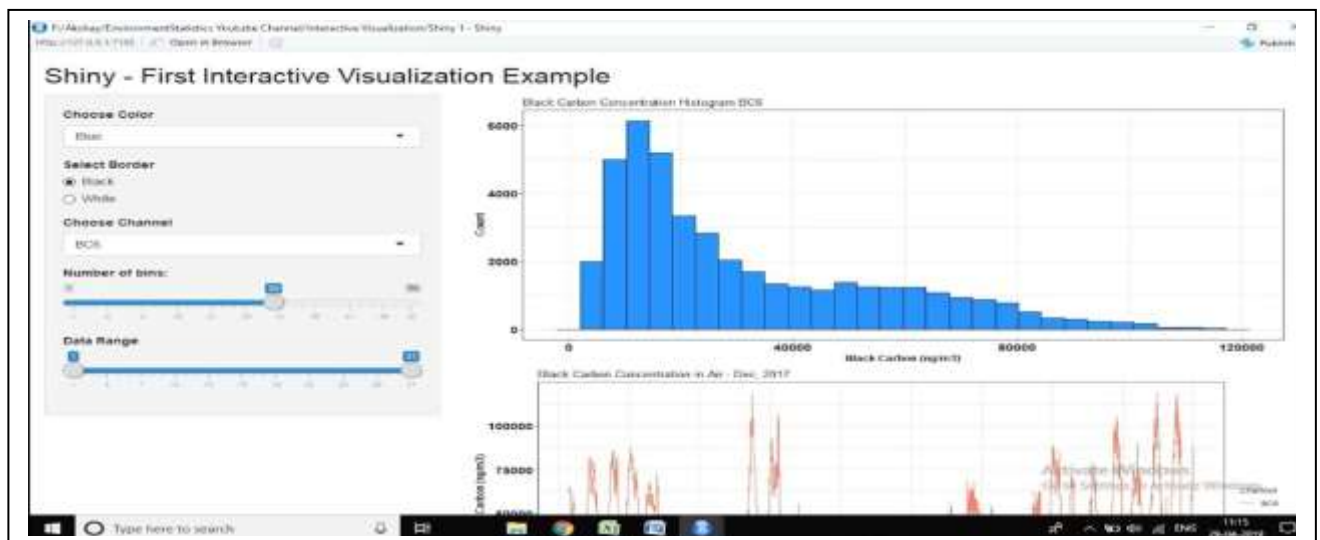


07/12/2021

Déploiement d'une application DataViz sur R



Réalisé par :

- HAMZA Assoumani Chissi
- SIKALIE Vanelle-natacha

Table des matières

Introduction	2
Problématique	2
I. Dès l'acquisition de données à la construction de l'entrepôt de données.....	3
A) Sources de données	3
B) Modèle de l'entrepôt de données	3
Présentation de données.....	4
C) Traitement et intégration des données	5
II. APPLICATION.....	9
A. Architecture	9
Description des fichiers	10
B. Prise en main de l'application	14
Les constituants des fenêtres.....	15
Conclusion.....	25
Liens utiles	26

Introduction

Partout dans le monde actuellement, on parle toujours de la pandémie COVID-19. Ce dernier, qui est peut-être aujourd'hui le virus qui marque l'histoire depuis le XXe siècle, ne cesse pas de prendre de l'ampleur à l'échelle mondiale. Des investissements colossaux dans les grandes nations pour la recherche d'une solution, mais malheureusement, nous ne sommes pas à l'abri du virus. Ce qui prouve d'ailleurs que nous portons toujours des masques depuis plus de deux ans.

Cependant, la plupart des gens n'acceptent pas la réalité. Cela peut être par manque de confiance avec les personnes compétentes, par l'ignorance ou parfois un manque de transmission de l'information aux autres. Ce monde est divers et la plupart ne sont pas à l'aise avec les chiffres. Par ailleurs, d'autres solutions existent pour transmettre un message sans avoir à parler forcément des chiffres. Des visuelles par exemple.

Cette histoire résume l'intérêt que porte les tableaux de bord qui sont souvent présentés dans les journaux pour informer la population sur les tendances de la pandémie.

Dans ce présent rapport, nous allons présenter une solution sur R permettant de visualiser l'évolution de la pandémie à l'échelle national française.

Dans un premier temps, nous expliquerons les détails depuis l'acquisition de données jusqu'à la construction de l'entrepôt de données. Puis nous présenterons également l'application DATAVIZ réalisée sur Rshiny.

Problématique

Depuis le 24 janvier 2020, date des premiers cas de Covid 19 en France jusqu'à nos jours, la pandémie covid 19 a mis fin à la vie de plusieurs citoyens français. Notre application R shiny a pour but de mesurer l'impact de la covid sur le nombre de décès en France, mais aussi l'impact de la vaccination sur ce dernier.

I. Dès l'acquisition de données à la construction de l'entrepôt de données

A) Sources de données

Nous nous sommes référés à deux sites fournisseurs de données open source (data.gouv.fr et insee.fr).

➤ Dans data.gouv.fr

Pour l'obtention des données ayant trait à la covid, nous avons utilisé le jeu de données comprenant l'essentiel des indicateurs de synthèse permettant le suivi de l'épidémie de COVID-19 en France disponible [ici](#).

Nous avons utilisé les données issues du système d'information Vaccin Covid permettant de dénombrer en temps quasi-réel (J-1), le nombre de personnes ayant reçu une injection de vaccin anti-covid en tenant compte du nombre de doses reçues, du vaccin, de l'âge, du sexe ainsi que du niveau géographique (national, régional et départemental). Ces données sont disponibles [ici](#).

Vu que notre analyse devrait être faite en fonction des départements et des régions nous avons cherché également les données sur les [régions](#) et [départements](#) de France.

➤ Dans insee.fr

Nous nous sommes référés au site insee.fr pour l'obtention des données sur le nombre de décès en France avant et après la COVID 19 disponible [ici](#).

B) Modèle de l'entrepôt de données

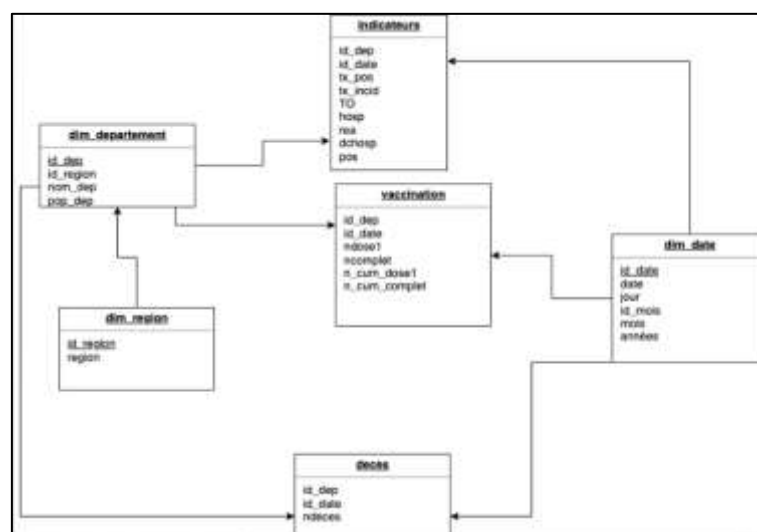


Figure 1 : modèle de l'entrepôt

Notre modèle relationnel pour la construction de notre entrepôt de données est constitué est six tables.

Présentation de données

➤ Table de faits “indicateurs”

Cette table contient les différents champs suivants :

hosp = Nombre de patients actuellement hospitalisés pour COVID-19.

rea = Nombre de patients actuellement en réanimation ou en soins intensifs.

dchosp = Décès à l’hôpital

pos = Nombre de personnes déclarées positives (J-3 date de prélèvement)

tx_pos = Taux de positivité des tests virologiques

tx_incid = Taux d'incidence (activité épidémique : Le taux d'incidence Correspond au nombre de personnes testées positives (RT-PCR et test antigénique) pour la première fois depuis plus de 60 jours rapporté à la taille de la population. Il est exprimé pour 100 000 habitants)

➤ Table de faits “ vaccination”

ndose1 = nombre de personnes ayant reçu une première dose du vaccin

ncomplet = nombre de personnes ayant reçu toutes les dose du vaccin

n_cum_dose_1 = cumuler du nombre de personnes ayant reçu la première dose

n_cum_complet = cumuler du nombre de personnes ayant reçu la première dose

➤ Table de faits “décès”

La table de faits “décès” contient le nombre de décès “**ndeces**” par département depuis l’année 2019

➤ Table de dimension “dim_date”

id_date = clé primaire permettant d’identifier une date

date = sous un format Date , elle indique la date du jour

jour = integer permettant d’identifier le jour d’un mois précis

id_mois = clé primaire permettant d’identifier le mois

mois = string donnant le nom d’un mois

annee = attribut révélant l’année d’une date précise

➤ **Table de dimension “dim_département”**

id_dep : clé primaire permettant d’identifier un département

id_region : clé primaire permettant d’identifier une région

nom_dep : nom du département

➤ **Table de dimension “ dim_region”**

id_region : clé primaire permettant d’identifier une région

nom_region : nom de la région

C) Traitement et intégration des données

En vue de combiner nos données provenant de plusieurs sources, nous avons utilisé l’outil d’intégration Talend. Nos fichiers finaux csv de notre entrepôt de données sont stockés sur Amazon Simple Storage Service (Amazon S3) qui est un service de stockage d'objets dans le cloud. S3 offre une capacité de mise à l'échelle, une disponibilité des données, une sécurité et des performances de pointe, ce qui nous permet de stocker et d’extraire n’importe quel volume de données de n’importe quel emplacement.

Nous avons récupéré les fichiers bruts au format csv sur la plupart de nos données. Mais nous avons également utilisé des API pour construire les métadonnées de certains fichiers.

Une métadonnée est une définition de la structure du format d’un fichier qu’on exporte localement. Dans Talend, il existe le composant **tREST** qui permet de lire de données à partir d’un API.

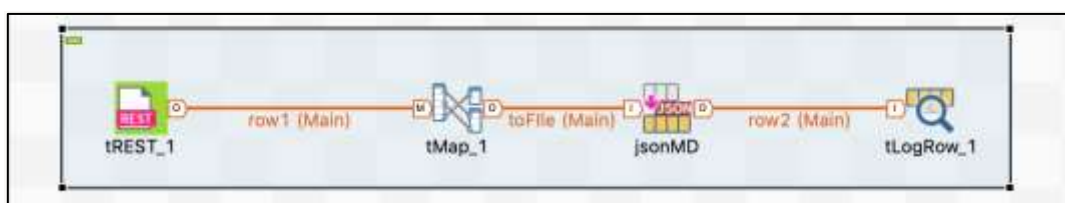


Figure 2: Composant tREST pour l’extraction de données externe via un API

➤ **Table de Dimension dim_date**

Pour l’alimentation de notre table de dimension dim_date, nous avons utilisé le composant tCalendar et à l’aide du composant tMap nous avons extrait les attributs qui nous semblaient les plus utiles. Avant l’exécution de notre job Talend nous avons établi une connexion sur notre espace de stockage s3 à l’aide du composant tS3Connection et après le job Talend nous avons stocker le fichiers csv généré sur Amazon S3 à l’aide du composant tS3put.

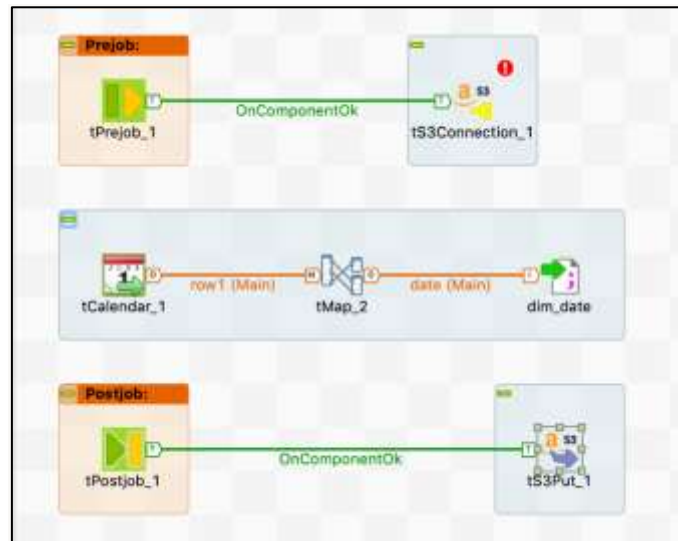


Figure 3 : Construction et exportation de la table Date

➤ Table de faits indicateur

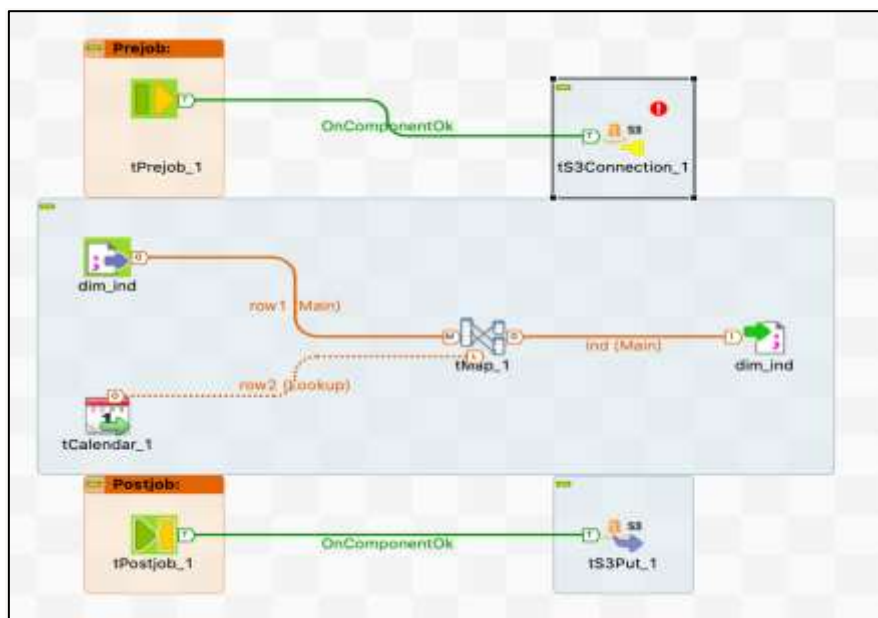


Figure 4 : Construction et exportation de la table indicateur

Pour l'alimentation de notre table de faits indicateur, nous avons fusionné le fichier brut contenant nos indicateurs avec le composant tCalendar pour obtenir les id_date correspondante. En outre, nous avons sélectionné les attributs liés à notre problématique et supprimé ceux qui nous semblaient inutiles. Il faut noter que le fichier contenant les infos sur

les indicateurs contenaient déjà les informations sur les départements et les régions. Nous avons stocké le fichiers csv final de notre table de fait indicateur sur Amazon S3 à l'aide du composant ts3put.

➤ Table de faits décès

Concernant la table de faits décès, les données issues de INSEE présentait plusieurs erreurs et sa structure n'était pas conforme à nos attentes. Pour pallier à cette situation, nous avons utilisé d'une part un code VBA et d'autre part des programmes sur python. Cette étape génère au final trois fichiers correspondant aux données de décès des années 2019, 2020 et 2021.

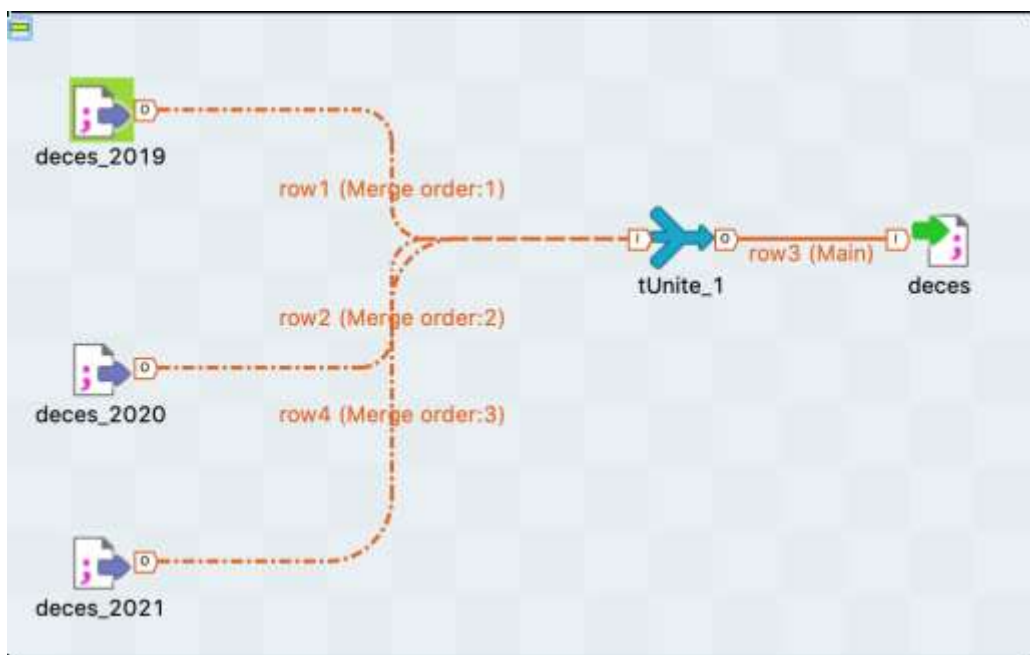


Figure 5 : reconstruction du fichier décès

Nous avons également utilisé talend pour concaténer nos 3 fichiers sur un seul fichier sortie.

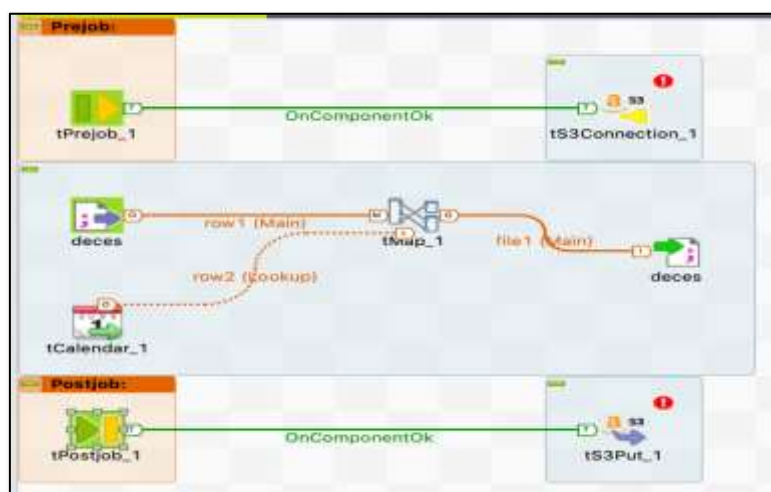


Figure 6 : exportation du fichier deces vers le Cloud.

Nos données étant toutes sur le même fichier, nous avons stoker le fichiers csv de notre table de fait décès sur Amazon S3.

➤ **Table de faits vaccination**

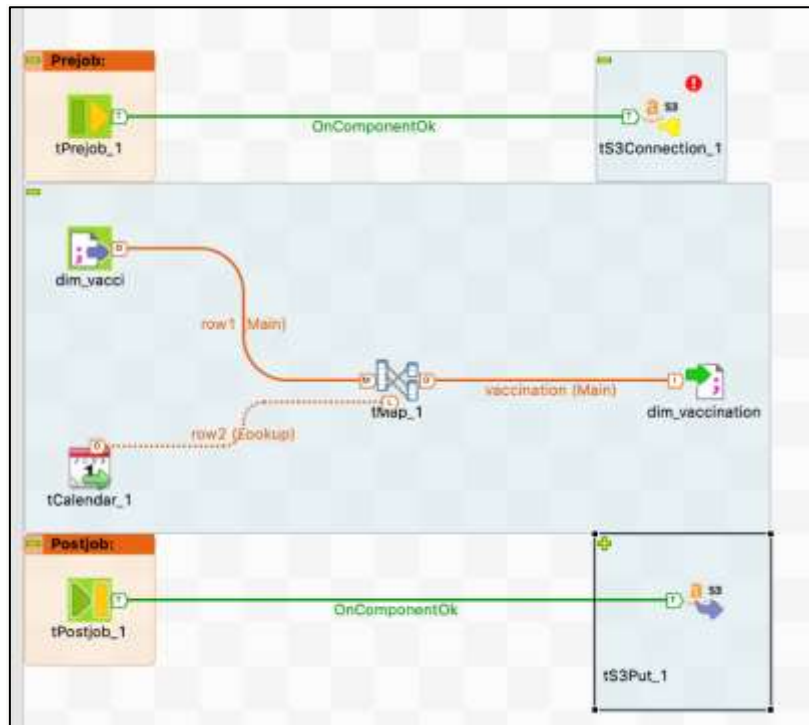


Figure 7 : construction et exportation de la table vaccination

Même principe pour la construction de la table de faits vaccination comme présenté sur la figure 7.

II. APPLICATION

Cette deuxième partie consiste à appréhender le fonctionnement et le mode d'utilisation de l'application. Nous parlerons en première lieu des différents modules qui composent l'architecture de notre application, puis d'un guide utilisateur détaillé.

A. Architecture

L'application a été développée sous R, en utilisant la librairie Shiny. Shiny est un package R qui facilite la construction d'applications web interactive depuis R. L'utilisateur peut simplement manipuler l'application pour exécuter et afficher des résultats fournis par du code R sur le navigateur.

Shiny présente plusieurs avantages, parmi eux :

- Les résultats fournis sont réactifs, c'est-à-dire que quand l'utilisateur fournit une nouvelle valeur en entrée (input) via un widget les codes R qui dépendent de cet input sont réexécutés et leurs sorties affichées (output).
- On peut également introduire du HTML.

Notre application **Shiny App** est construite en deux parties :

1. Un côté **UI** (user interface) qui regroupe tous les éléments de mise en forme et d'affichage de l'interface utilisateur elle-même (affichage des inputs et des outputs).
2. Un côté serveur(server) où sont exécutés les codes R qui servent à produire les outputs (graphiques, tables, traitements, etc.) et à les mettre à jour en cas de changement dans les valeurs d'inputs.

La façon la plus simple pour lancer une application shiny sur R est d'aller sur *File -> New file -> Shiny Web App -> Multiple File*.

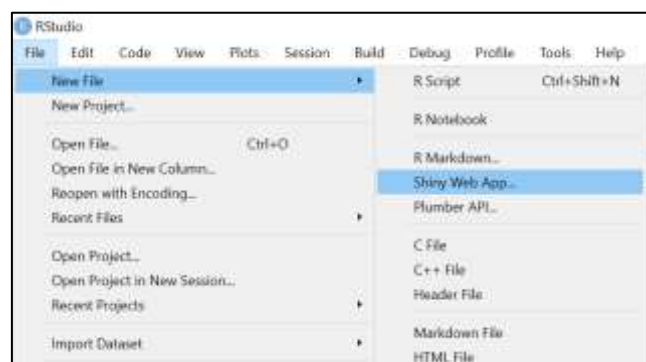


Figure 8 : a- lancement d'une application Shiny.

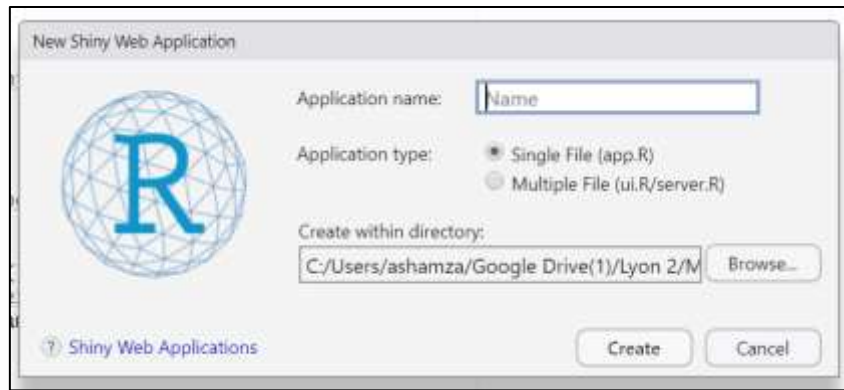


Figure 9: b- lancement d'une application Shiny.

Deux structures de base sont possibles pour les apps :

- Soit avoir tout réuni dans un même script (app.R),
- Soit séparer la partie **ui** et la partie **server** dans deux fichiers (ui.R et server.R). C'est cette deuxième solution que nous avons privilégiée pour avoir une meilleure visibilité du programme.

Nous avons également ajouté un fichier **global** (global.R). L'avantage est que les commandes du fichier global.R sont exécutées dans un environnement "global". C'est-à-dire que les packages qu'on y charge ou les objets qu'on y crée seront disponibles à la fois pour les parties UI et Server. Les commandes de global sont donc exécutées une fois pour toutes (c'est-à-dire, une fois par session, et avant toute autre chose) au lancement de l'application. Donc il n'est pas possible de déclarer des variables réactives dans ce fichier global.

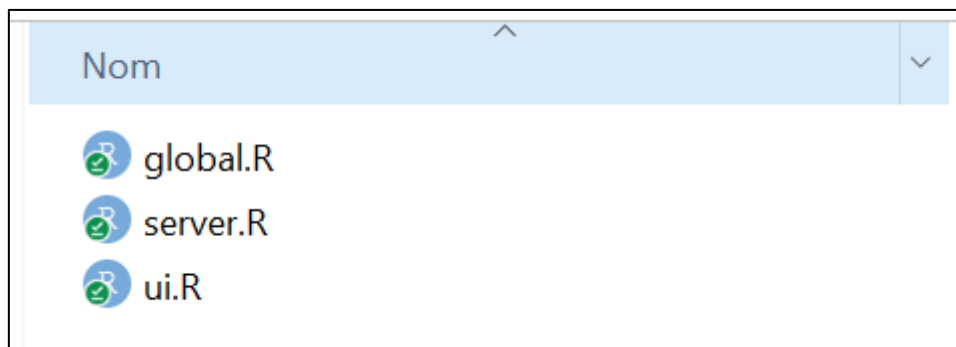


Figure 10: arborescente de l'application.

Il est strictement conseillé de respecter le nommage de ces trois fichiers sinon l'application ne fonctionnera pas (Oui, j'ai déjà testé en amont.).

Description des fichiers

1. Le fichier global (global.R)

Comme il a été mentionné précédemment, c'est le fichier qui contient les variables à partager entre le serveur et l'interface. Dans notre cas, nous avons essentiellement chargé nos jeux de

données directement du cloud, les librairies nécessaires, ainsi que la définition de nos fonctions.

```
#librairies
library(plotly)
library(sqldf)
library(rnaturalearth)
library(magrittr)
library(tmap)
library(shiny)
#library(shinydashboard)

#chargement des données

#région
regions=read.csv("https://datafileapp.s3.eu-west-3.amazonaws.com/region.csv",header = T,sep = ";", encoding
= "UTF-8")
colnames(regions) <- c("id_region","region")
#département
dep=read.csv("https://datafileapp.s3.eu-west-3.amazonaws.com/departement.csv",header = T,sep=";", encoding
= "UTF-8")
#table date
ddate=read.csv("https://datafileapp.s3.eu-west-3.amazonaws.com/dim_date.csv",header = T,sep=";",
encoding="UTF-8")
#deces
deces=read.csv("https://datafileapp.s3.eu-west-3.amazonaws.com/dim_deces.csv",header = T,sep = ";",
encoding="UTF-8")
#indicateurs
indi=read.csv("https://datafileapp.s3.eu-west-3.amazonaws.com/dim_indicateurs.csv",header = T,sep = ";",
encoding = "UTF-8")
#indi$dep=as.character(indi$dep)
#vaccination
vaccin=read.csv("https://datafileapp.s3.eu-west-3.amazonaws.com/dim_vaccination.csv", header = T, sep =
";", encoding = "UTF-8")

#name_region
name_reg=ordered(regions$region)
#name dep
name_dep=ordered(dep$nom_dep)

#get id dep
getId_dep<-function(nom_dep){
  idl=dep[dep$nom_dep==nom_dep,"id_dep"]
  return(idl)
}

#get id reg
getId_reg<-function(nom_reg){
  idl=regions[regions$region==nom_reg,"id_region"]
  return(idl)
}

#get id date
getId_date <- function(iddate){
  madate=ddate[ddate$date==iddate,"id_date"]
  return(madate)
}
```

Code source du fichier global.R

Six fichiers qui forment notre modèle ont été chargés ici à savoir :

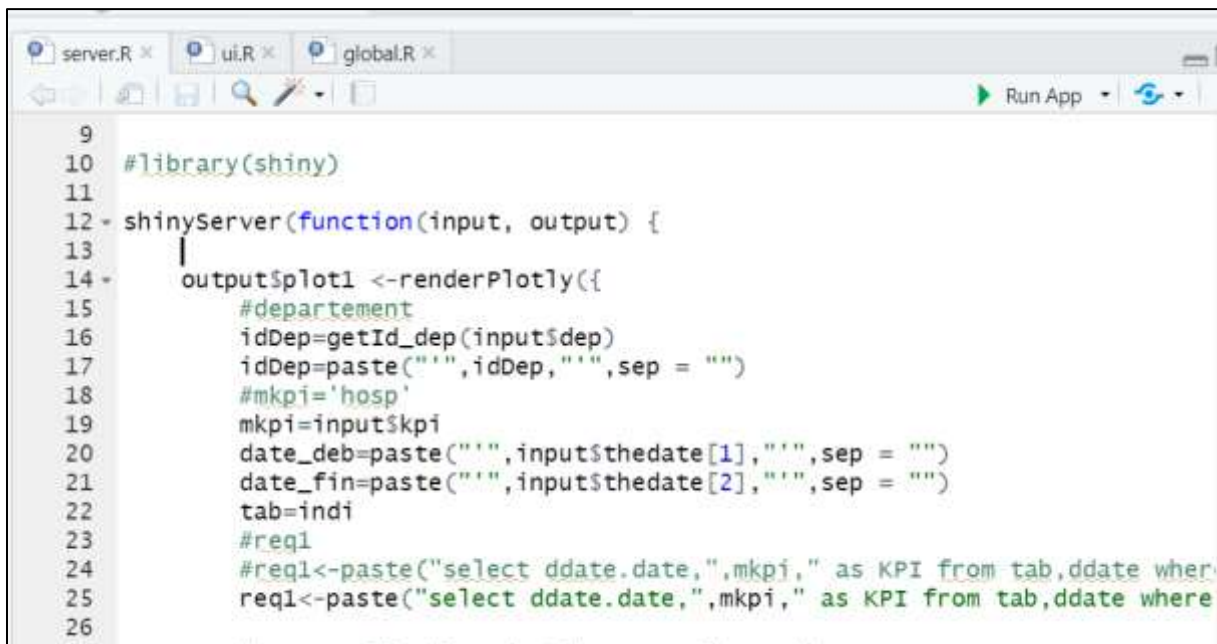
- La table **region** qui contient les régions et les id correspondantes
- La table **departement** qui contient les départements et les id correspondantes
- La table **dim_date** qui contient les dates depuis 01/01/2019 au 31/12/2021. On y trouve les id date, jour, mois, année, etc.

- La table **dim_deces** qui contient le total de décès aux quotidiennes.
- La table **dim_indicateurs** qui contient les indicateurs (KPI) liés à l'épidémie COVID 19 à savoir le taux d'incidence (tx_incid), les nombres de tests positifs (pos), le taux de positivité (tx_pos), le nombre d'hospitalisation (hosp), le nombre de décès à l'hôpital (dchosp), le taux d'occupation des lits (TO) et le nombre de réanimation(rea).
- La table **dim_vaccination** qui contient les indicateurs liés à la vaccination (1^{ère} et 2^{ème} dose).

On y trouve également quelques fonctions prédéfinies permettant par exemple de récupérer l'id département en fonction du nom de département (**getId_dep**), l'id région avec **getId_reg** ou l'id date avec **getId_date**.

2. Le fichier serveur (server.R)

Le fichier server.R contient l'ensemble du code R qui doit être exécuté par l'application pour fournir les sorties. On encapsule le code utilisé pour créer l'output dans une fonction de type **render*()**. Exemple **renderText()** si la sortie souhaitée est de type texte. Les outputs sont les composants de l'interface graphique qui permettent d'afficher des éléments résultant d'un traitement dans R (graphiques, tables, textes...).



```

9
10 #library(shiny)
11
12 shinyServer(function(input, output) {
13   |
14   output$plot1 <- renderPlotly({
15     #departement
16     idDep=getId_dep(input$dep)
17     idDep=paste("'",idDep,"'",sep = "'")
18     #mkpi='hosp'
19     mkpi=input$skpi
20     date_deb=paste("'",input$thedata[1],"'",sep = "'")
21     date_fin=paste("'",input$thedata[2],"'",sep = "'")
22     tab=indi
23     #req1
24     #req1<-paste("select ddate.date,"mkpi," as KPI from tab,ddate wher
25     req1<-paste("select ddate.date,"mkpi," as KPI from tab,ddate where
26

```

Figure 11: Exemple d'un bout de code sur la partie serveur.

Tous les objets créés sur l'interface UI sont identifiés, et peuvent être récupérés directement au côté serveur en faisant référence. Par exemple la ligne 16 de la figure précédente, nous avons utilisé **input\$dep** pour récupérer la valeur du département choisi, puis la fonction **getId_dep()** pour récupérer son numéro de département associé. Le code source du fichier serveur sera expliqué au fur et à mesure.

Une particularité à retenir sur Shiny au côté serveur est qu'on définit une fonction de type **shinyServer**, avec input et output comme arguments. Le corps de cette fonction s'écrit donc comme une suite de lignes de commandes. Les commandes sont séparées par des retours à la ligne. C'est vraiment des instructions de R, donc pas de limitation d'une instruction par exemple le « ; ou , ».

3. Le fichier interface utilisateur (ui.R)

Le fichier ui.R contient les instructions de construction et mise en forme de l'interface utilisateur. Il contient principalement les inputs et les outputs. Les inputs sont les composants (widgets) de l'interface graphique qui permettent aux utilisateurs de fournir des valeurs aux paramètres d'entrée. Par exemple les checkbox, bouton d'action, zone de texte, etc.

```
10 # Define UI for application that draws a histogram
11 shinyUI(fluidPage(
12   # Application title
13   titlePanel("Informations Covid-19"),
14
15   # Sidebar with a slider input for number of bins
16   sidebarLayout(
17     sidebarPanel(
18       selectInput("dep", label = h4("Département"),
19                 #choices = list(textOutput("reg"), "Choice 2" = 2, "C
20                 choices= levels(name_dep),
21                 multiple = T,
22                 selected = NULL),
23
24       #gestion date
25       dateRangeInput("thedata", label=h4("Plage temps"),
26                     start = "2020-03-19",
27                     end = "2021-03-30",
28                     format = "yyyy-mm-dd", startview = "month", separ
29
30     #choix de l'indicateur
```

Figure 12: Exemple d'un bout de code sur la partie UI.

Il faut noter que les éléments définis sur ui (user interface) sont des objets de type UI issu de l'appel à une fonction, ici **fluidPage()**. Les différents éléments passés à fluidPage() sont donc des arguments : ils sont séparés par des virgules.

On ajoute des éléments d'entrée à l'interface avec les fonctions de type ***Input()**, et des éléments de sortie à l'interface avec les fonctions de type ***Output()**.

Dans la prochaine partie, nous expliquerons en détail le code source du fichier ui.R.

B. Prise en main de l'application

On peut lancer l'application directement sur le fichier ui.R ou server.R en cliquant sur le bouton **Run App**.

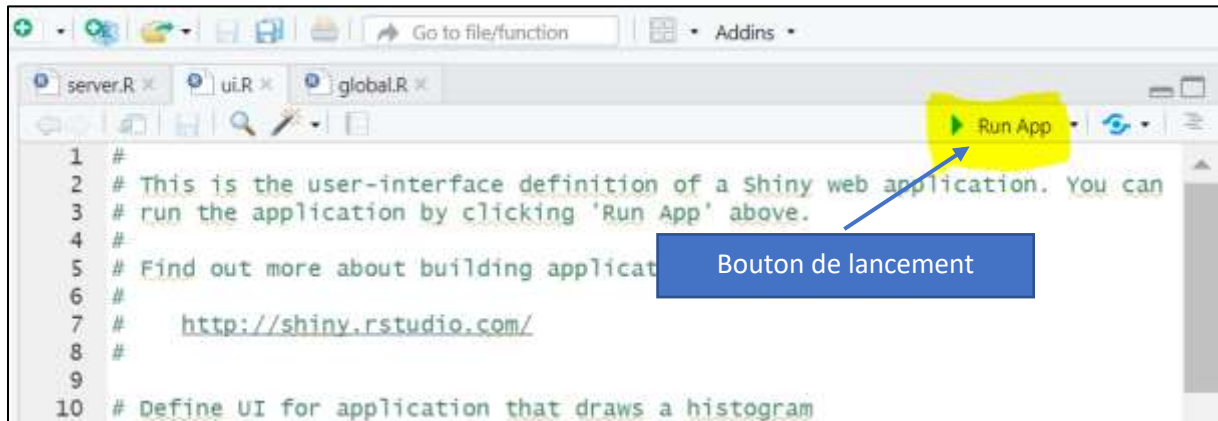


Figure 13: lancement de l'application.

Après quelques secondes pour le chargement de données, l'interface de l'application s'affiche comme le montre la figure suivante.



Figure 14: interface d'accueil

Nous avons divisé la fenêtre en deux éléments :

- Un élément latéral plus étroit qui contient généralement des inputs permettant d'affiner la recherche.
- Un élément principal qui contient généralement des outputs (les graphes, tableaux, etc.).

Cela n'est pas une règle absolue. On peut tout de même arranger nos éléments comme on le souhaite. Mais nous avons essayé de suivre une logique en inspirant sur l'exemple du [site du gouvernement](#).

Les constituants des fenêtres

1. Fenêtre latérale

La fenêtre latérale est constituée de quatre entrées.



Département(s)

Plage temps

2020-03-19 00 2021-03-30

L'indicateur

- ☒ Taux d'incidence
- ☐ Taux de positivité
- ☐ Nombre d'hospitalisation
- ☐ Nombre de réanimation
- ☐ Nombre de décès COVID-19
- ☐ Tests positifs

Épidémie en fonction :

- ☒ Vaccin
- ☐ Décès

Figure 14: fenêtre latérale

- **Département** : Ici, nous choisissons le(s) département(s) qui nous souhaitons visualiser selon les différents graphes. Dans certains cas, nous devons choisir deux départements pour faire une comparaison.
- **Plage temps** : c'est l'intervalle de temps qu'on souhaite analyser.
- **L'indicateur** : nous avons le choix entre les différents KPI que nous souhaitons voir sur l'analyse.
- **Critère de comparaison** : le critère de comparaison permet entre autres de situer l'évolution de l'épidémie en fonction du suivi de la vaccination ou des décès.

Tous ces éléments forment des entrées, ils sont donc définis du côté *user interface*.

```
# Define UI for application
shinyUI(fluidPage(
  #selection du thème
  #shinythemes::themeSelector(),
  theme = shinytheme("sandstone"),
  # Application title
  titlePanel("Informations Covid-19"),

  # subdivison de l'interface
  sidebarLayout(
    sidebarPanel(
      #choix département
      selectInput("dep", label = h4("Département(s)"),
        choices= levels(name_dep),
        multiple = T,
        selected = NULL),

      #gestion date
      dateRangeInput("thedata", label=h4("Plage temps"),
        start = "2020-03-19",
        end = "2021-03-30",
        format = "yyyy-mm-dd", startview = "month", separator = " to "),

      #choix de l'indicateur
      radioButtons("kpi", label = h4("L'indicateur"),
        choices = list("Taux d'incidence" = "tx_incid",
          "Taux de positivité" = "tx_pos",
          # "Taux d'occupation" = "TO",
          "Nombre d'hospitalisation" = "hosp",
          "Nombre de réanimation" = "rea",
          "Nombre de décès COVID19" = "dchosp",
          "Tests positifs" = "pos"
        ),
        selected = "tx_incid"),

      #terme de comparaison
      radioButtons("choix", label = h5("L'épidémie en fonction :"),
        choices = list("Vaccin"="1", "Décès"="2"),
        selected = "1")

    ),

    # Show the plots
    mainPanel(
      .....
    )
  )
))
```

Code source côté user interface

Définition des attributs :

- **sidebarPanel()** : il permet de regrouper les éléments sur le menu latéral.
- **selectInput()** : qui est un encapsulé sur le sidebarPanel, permet définir un *select box*. Il est identifié par son premier argument qui doit être unique afin qu'on puisse récupérer sa valeur au coté serveur.
- **dateRangeInput()** : ce widget permet de définir une entrée de type date sous forme d'intervalle.
- **radionButtons()** : Comme dans d'autres langages, HTML par exemple, il est possible de définir des boutons radio. Son objectif est de permettre à l'utilisateur de choisir une et une seule option parmi les options proposées.
- **mainPanel()** : qui est un attribut direct de **sidebarLayout()**, il permet de gérer la fenêtre principale. On y place généralement les sorties.

2. Fenêtre principale

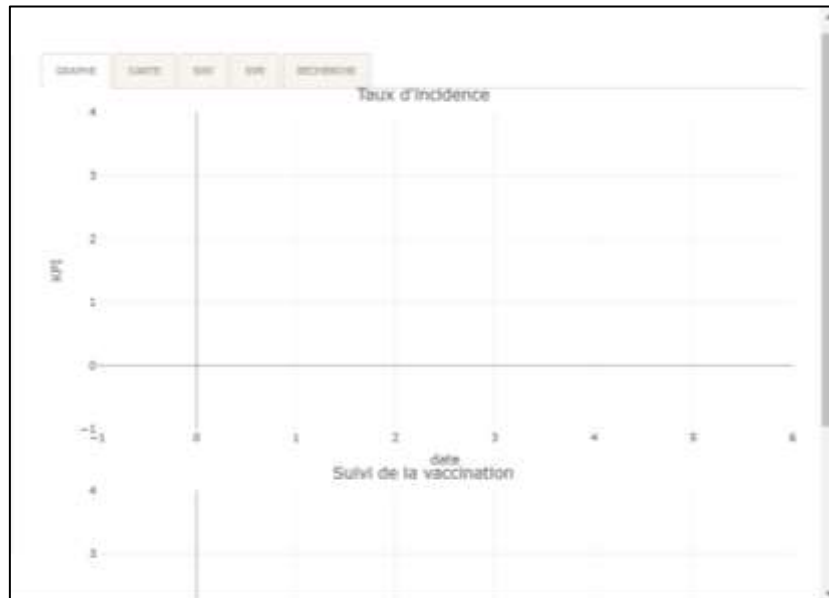


Figure 15: fenêtre principale

La fenêtre principale a été subdivisée en plusieurs onglets.

- a) **GRAPHE** : cette fenêtre nous permet selon un département et le KPI choisi, d'analyser la situation en fonction de la vaccination ou du nombre de décès.



Figure 16: exemple d'analyse.

Par exemple présenté ici (figure), nous avons essayé d'analyser l'évolution du Taux d'incidence du département **Allier** en fonction de la vaccination, mais l'outil nous offre la possibilité de faire une comparaison avec le nombre de décès. Cela nous permettra

de conclure réellement si la COVID a un impact majeur sur les décès par rapport aux années précédentes.



Figure 17: Comparaison en fonction de la vaccination

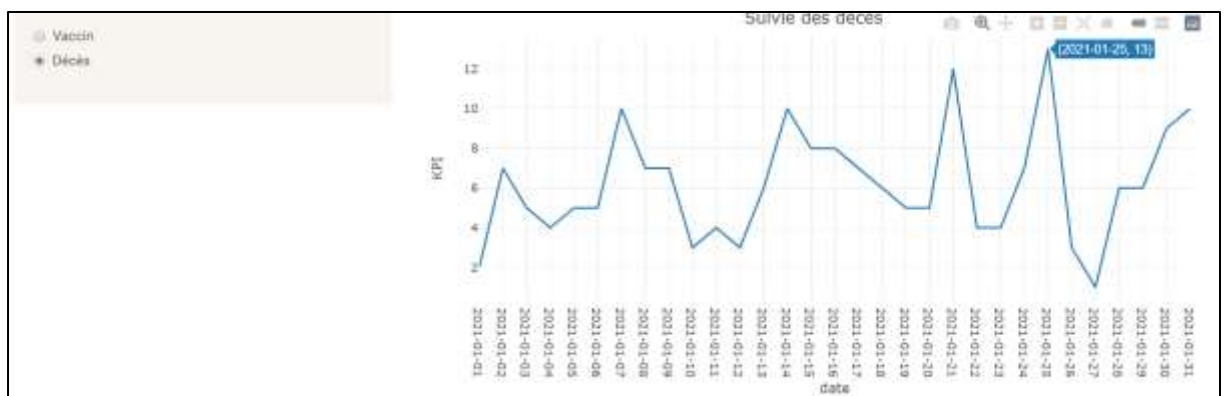


Figure 18: Comparaison en fonction des décès

Sur le fichier ui.R, l'input pour le choix de département est identifié par "dep". Le *name_dep* est une variable globale qui contient la liste des départements triés. Nous avons utilisé l'option *multiple=T* pour donner la possibilité de faire un choix multiple et *selected=NULL* permet de n'est pas cocher une valeur par défaut.

```
#choix département
selectInput("dep", label = h4("Département(s)"),
  choices= levels(name_dep),
  multiple = T,
  selected = NULL),
```

Code source sur le fichier ui.R pour le choix de département

La valeur de l'input a été récupérée au niveau serveur pour personnaliser la sortie en fonction du département choisi.

```

11
12 - shinyServer(function(input, output) {
13
14 -   output$plot1 <-renderPlotly({
15     #departement
16     idDep=getId_dep(input$dep)
17     idDep=paste("'",idDep,"'",sep = "")
18     #mkpi='hosp'
19     mkpi=input$skpi
20     date_deb=paste("'",input$thedata[1],"'",sep = "")
21     date_fin=paste("'",input$thedata[2],"'",sep = "")
22     tab=indi
23     #req1
24     #req1<-paste("select ddate.date,",mkpi," as KPI from tab,ddate wher
25     req1<-paste("select ddate.date,",mkpi," as KPI from tab,ddate where
26
27     #personnalisation du titre pour le graphe|
28 -     if(input$skpi=="tx_incid"){
29       titre="Taux d'incidence"
30 -     }
31 -     if(input$skpi=="tx_pos"){
32       titre="Taux de positivité"
33 -     }
34 -     if(input$skpi=="hosp"){
35       titre="Nombre de personnes actuellement hospitalisées"
36 -     }
37 -     if(input$skpi=="rea"){
38       titre="Nombre de patients actuellement en réanimation ou en soi
39 -     }
40

```

Figure 19: code source coté serveur

Dans le fichier serveur, l'**output\$plot1** a été défini pour une sortie de type **renderPlotly**. Ce qui signifie que la sortie entendue doit être un graphe de la librairie plotly. Plusieurs inputs ont été utilisés ici. Par exemple on a récupéré le département à la ligne 16 (input\$dep), l'indicateur qu'on souhaite analyser à la ligne 19 (input\$skpi), mais également l'intervalle de temps sur les lignes 20 et 21 (input\$thedata[]). Puis nous avons créé notre requête sql à partir de ces données.

- b) **CARTE** : cet onglet permet de visualiser sur une carte géographique les différentes mesures pour une date donnée. La carte est divisée en fonction des départements.

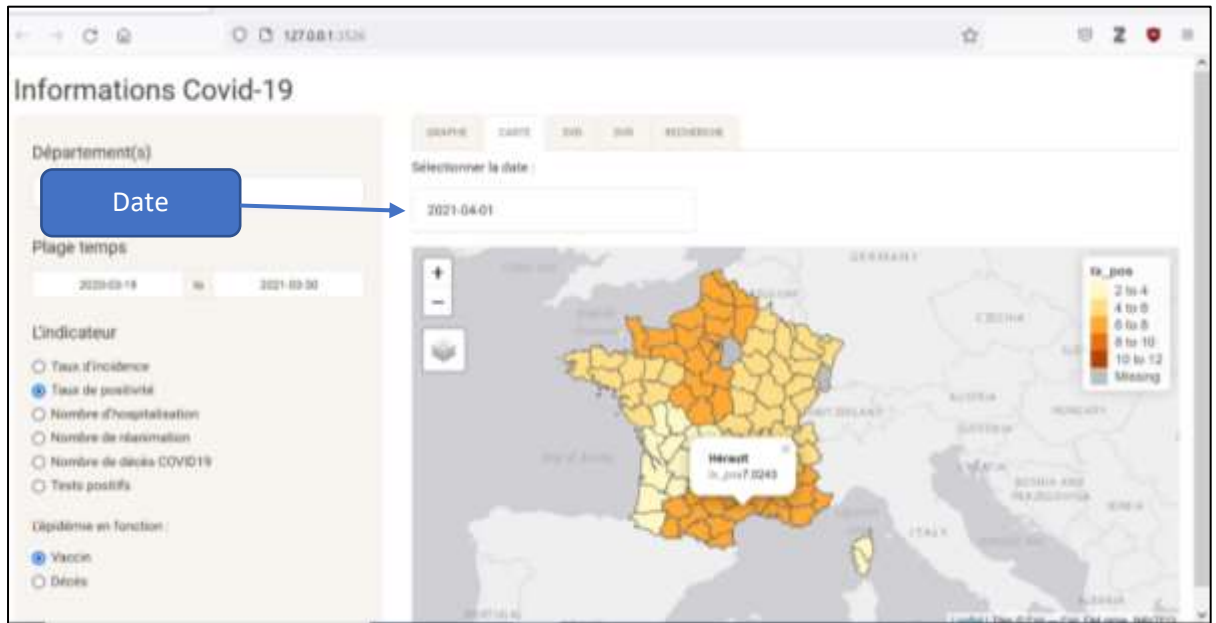


Figure 20: Vue géographique

Dans notre exemple ici, l'analyse porte sur la date "2021-04-01" et l'indicateur *Taux de positivité*. On peut également zoomer sur les départements et faire un click sur le nom d'un département pour faire apparaître l'information.

Dans le fichier serveur, nous avons récupéré deux inputs dont la date (`input$dater`) et l'indicateur de mesure (`input$kpi`). On a construit notre dataframe avec ces données puis l'utilisation de la librairie *tmap* pour générer la sortie. Ce qui est bien ici est que Shiny permet de gérer directement les sorties de type Tmap avec la fonction *renderTmap*.

```

150 - output$mytmap <- renderTmap({
151   # Récupérer la France Métropolitaine de {rnaturlaearth}
152   france <- ne_states(country = "France", returnclass = "sf")
153
154   tempDf <- merge(dep,regions,by='id_region',all.x = TRUE,sort=FALSE)[c
155   colnames(tempDf) <- c('id_dep','name','region','id_region')
156
157   dffrance <- merge(france,tempDf,by='name',all.x = TRUE,sort=FALSE)
158
159   madate=getId_date(input$dater)
160
161   id1=indi[indi$id_date==madate,]
162
163   colnames(id1) <- c('id_dep','id_date','id_region','tx_pos','tx_incid')
164
165   Dfinal <- merge(dffrance,id1,by='id_region',all.x = TRUE,sort=FALSE)
166
167   Dfinal$id_region <- NULL
168   #####
169 -
170
171   tmap_mode(mode = "view")
172   tmap_options(bg.color = "white", legend.text.color = "white")
173   tm_shape(Dfinal) + tm_polygons(col = input$kpi)
174 - })
175

```

Figure 21: Code source pour générer la carte.

- c) **SVD (Suivi par département)** : cet onglet permet de faire une comparaison entre deux départements dans un graphique sur un intervalle de temps en fonction d'un KPI.



Figure 22: Comparaison graphique entre deux départements

Il faut tout de même noter qu'une erreur s'affiche si vous choisissez qu'un département. Ce qui est normal puisque le programme n'arrive pas construire le dataframe.

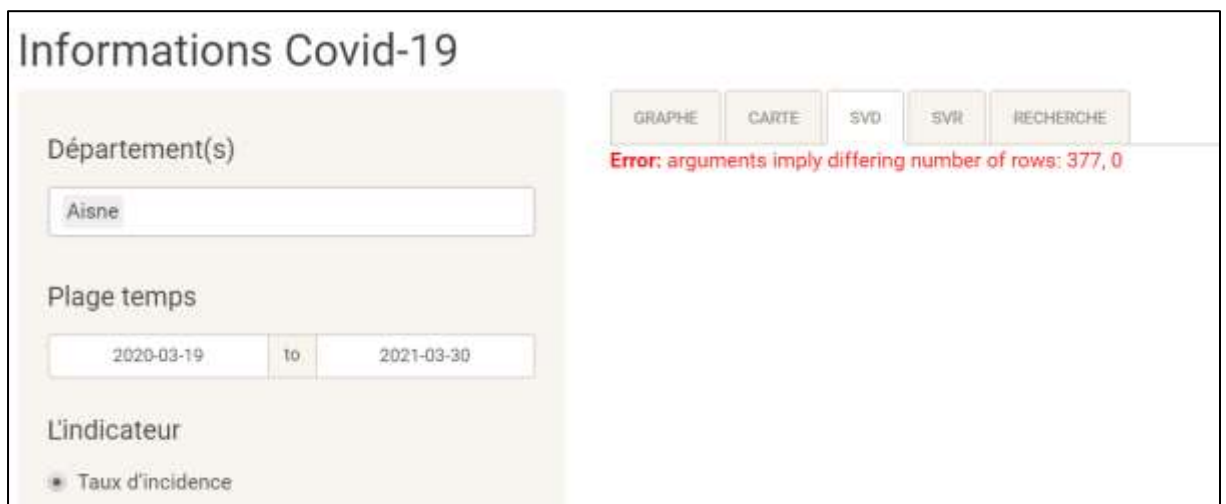


Figure 23: Message d'erreur

Pour le code source coté serveur, nous avons récupéré les inputs : `input$dep[1]` et `input$dep[2]` pour les départements, l'indicateur d'analyse (`input$kpi`) et l'axe de temps (`input$thedata[i]`, $i=1,2$). La fonction de sortie est de type `renderPlotly`. Il est aussi important de préciser que toutes les variables d'entrées réactives devront être placées à l'intérieur d'une fonction de type `render*()` sur le serveur, sinon le serveur shiny générera une erreur.

Il existe d'autre particularité permettant d'isoler des variables ou codes qui dépendent des inputs réactives pour que le serveur ne recalcule pas la sortie à chaque modification de l'input. Mais nous n'avons pas eu à l'utiliser dans notre cas.

```

55 - output$plot2 <-renderPlotly({
56   #departement
57   idDep1=getId_dep(input$dep[1])
58   idDep1=paste("",idDep1,"",sep = "")
59   idDep2=getId_dep(input$dep[2])
60   idDep2=paste("",idDep2,"",sep = "")
61   #choix indicateur
62   mkpi=input$skpi
63   #range date
64   date_deb=paste("",input$thedate[1],"",sep = "")
65   date_fin=paste("",input$thedate[2],"",sep = "")
66
67   tab=indi
68
69   #req1
70   req1<-paste("select ddate.date,"mkpi," as KPI1 from tab,ddate where t
71
72   #req2
73   req2<-paste("select ddate.date,"mkpi," as KPI2 from tab,ddate where
74
75   dfq1 <- sqldf(req1)
76   dfq2 <- sqldf(req2)
77   dt=cbind(dfq1,dfq2$KPI2)
78

```

Figure 24 : code source server : onglet SVD

- d) **SVD (Suivi par région)** : cet onglet nous permet de faire une comparaison entre régions pour l'indicateur choisi à d'une date donnée.



Figure 25 : graphique comparaison entre région

Nous avons la possibilité de filtré une région en faisant un simple clic sur son nom dans la palette à droite.

Le code source sur le côté serveur construit une sortie de type renderPlotly. Il utilise les inputs pour le KPI (input\$kpi) et la date (input\$dtr) comme variables réactives.

```

135 - output$plot6 <- renderPlotly({
136   tempDf <- merge(dep,regions,by='id_region',all.x = TRUE,sort=FALSE)[c(
137     colnames(tempDf) <- c('dep','nom_dep','region')
138     tempDf <- merge(tempDf,indi,by="dep")
139     tempDf <- merge(tempDf,ddate,by="id_date")[c('date','nom_dep','region')
140     tempDf <- tempDf[tempDf$date==input$dtr,] #filtrage selon la date
141
142     #récupérer la colonne désigner par le kpi avec input$kpi
143     tempy <- tempDf[,input$kpi]
144
145     #construire le graphe
146     fig <- plot_ly(tempDf, type='pie', labels = ~ region, values = tempy
147     fig <- fig %>% layout(uniformtext=list(minsize=12, mode='hide'))
148     fig
149 - })
150

```

Figure 26 : code source server : onglet SVR

- e) **RECHERCHE** : cet onglet permet de faire une recherche selon des critères. Par exemple filtré selon les départements, par région ou par mot-clé.

Informations Covid-19

Département(s)

Plage temps

L'indicateur

L'épidémie en fonction :

Recherche

DATE	DEPART	REGION	TX POS	TX INCID	TX OCPL	NBR HOSPI	NBR REA	DC HOSP	TEST POS
2020-03-18	Val-d'Oise	Ile-de-France			0.2563208	90	29	2	
2020-03-18	Val-de-Marne	Ile-de-France			0.2563208	122	32	5	
2020-03-18	Hauts-de-Seine	Ile-de-France			0.2563208	149	48	4	
2020-03-18	Yvelines	Ile-de-France			0.2563208	69	18	5	
2020-03-18	Seine-Saint-Denis	Ile-de-France			0.2563208	92	33	5	
2020-03-18	Essonne	Ile-de-France			0.2563208	51	16	1	
2020-03-18	Seine-et-Marne	Ile-de-France			0.2563208	25	13	0	

Figure 28 : table de recherche selon les critères

Le résultat est représenté sous forme d'une table qui regroupe toutes les informations nécessaires.


```

-   output$matable <- renderDataTable({
-       tempDf <- merge(dep,regions,by='id_region',all.x = TRUE,sort=FALSE)[c
-       colnames(tempDf) <- c('dep','nom_dep','region')
-       tempDf <- merge(tempDf,indi,by="dep")
-       tempDf <- merge(tempDf,ddate,by="id_date")[c('date','nom_dep','region')
-       colnames(tempDf) <- c('DATE','DEPART','REGION','TX POS','TX INCID','TX
-       tempDf
-   })

```

Figure 29 : code source server : onglet recherche

Pour le code source rien n'est magique ici, il s'agit d'une fusion de données pour les tables région, département, date et celle des indicateurs. L'astuce ici est qu'il fallait éviter d'utiliser une requête sql, car elle consommait beaucoup des ressources mémoires.

Conclusion

Nous avons pu montrer la puissance du logiciel R avec sa librairie Shiny qu'il est possible de créer des applications plus performantes au niveau de la visualisation des données d'une part, même si la solution que nous avons proposée ici n'est plus comparable aux solutions qui existent déjà dans le marché.

D'autre part, ce projet nous a permis de travailler sur les outils BI comme TALEND, mais également des plateformes Cloud comme Amazon Web Service. Ce qui était pour nous un projet riche et complet en connaissance selon notre formation.

Liens utiles :

- [Code source de l'application](#)
- [Vidéo de démonstration](#)
- [Exécution de l'application en ligne](#)

Contact :

- HAMZA : assoumani-chissi.hamza@univ-lyon2.fr
- SIKALIE : vn.sikalie@univ-lyon2.fr