# Documentation technique pour le code R shiny « Velo’V »

Rédigé par Tidiane GRANDJEAN, Arnaud VANNAXAY et Cordelia GUY

Dans le cadre du BUT Science des Données

Une image contenant plein air, vélo, Roue de vélo, roue

Description générée automatiquement

Table des matières

[Documentation technique pour le code R shiny « Velo’V » 1](#_Toc149494274)

[Table des matières 2](#_Toc149494275)

[Introduction : 3](#_Toc149494276)

[Package utilisés : 3](#_Toc149494277)

[Analyse du code : 4](#_Toc149494278)

[Script Serveur : 4](#_Toc149494279)

[Lignes 48-51 : Import des données 4](#_Toc149494280)

[Lignes 71-75 : CSV des codes postaux 4](#_Toc149494281)

[Lignes 78-82 : calcul du nombre de vélos disponibles total 4](#_Toc149494282)

[Lignes 88-113 : Création de la carte 5](#_Toc149494283)

[Lignes 116- 128 : création des filtres par stations et codes postaux 5](#_Toc149494284)

[Lignes 134-146 : Nombre de vélos et places disponibles en fonction du code postal 6](#_Toc149494285)

[Lignes 151-159 : nombre moyen de vélo par station par codes postaux 6](#_Toc149494286)

[Ligne 164-172 : nombre de vélos disponibles en fonction de la station 6](#_Toc149494287)

[Lignes 177-185 : Nombre de places disponibles en fonction de la station 7](#_Toc149494288)

[Lignes 190-196 : bouton refresh 7](#_Toc149494289)

[Lignes 202-222 : Création du pie chart 7](#_Toc149494290)

[Lignes 228-238 : Création du df\_ratio 8](#_Toc149494291)

[Lignes 240-243 : tri du df\_ratio 8](#_Toc149494292)

[Lignes 244-257 : Top 3 des stations du df ratio 8](#_Toc149494293)

[Lignes 262-292 : Export du graphique en JPEG 9](#_Toc149494294)

[Script UI : 9](#_Toc149494295)

[Lignes 13 – 26 : Sidebar Menu 9](#_Toc149494296)

[Lignes 27 – 41 : Dashboard body 10](#_Toc149494297)

[Lignes 43 – 59 : Accueil 10](#_Toc149494298)

[Lignes 60 -79 : Stations 11](#_Toc149494299)

## Introduction :

Dans le cadre de notre formation BUT Science des données nous avons dû créer un code R shiny afin d’obtenir des informations et statistiques en temps réel sur les vélos de la métropole de Lyon. Pour se faire nous avons eu besoin d’une API afin de chercher les données en temps réel sur le site JCDecaux.

Ce travail c’est réalisé en groupe qui est composé de Tidiane GRANDJEAN, Arnaud VANNAXAY et Cordelia GUY.

Ce document va documenter le code R afin de comprendre sa technique, il est complété par la documents fonctionnel qui lui sera plus orienter niveau utilisateur.

Ce document sera découpé en deux partie avec en premier point les packages utilisés, puis une description du code R.

## Package utilisés :

"httr" : Utile lors de l’utilisation d’API

"jsonlite" : utile pour la conversion de données

"tidygeocoder" : Permet de géocoder les adresses de l’API afin d’obtenir les codes postaux

"dplyr" : Aide pour la manipulation de données (group\_by)

"ggplot2" : Augmente la capacité de création de graphique dans R

"shiny" : Permet de créer des applications RShiny

"markdown" : Utile pour formater du code

"leaflet" : Utile pour la génération de carte interactives

"DT" : Permet de créer des tableaux de données interactifs

"RMySQL" : Permet de rédiger des requêtes SQL (utile pour l’implantation dans une base de données)

"cli" : Utile pour l’interaction utilisateur

"plotly" : utile pour la génération de graphique

"RColorBrewer" : permet d’avoir un plus large choix de couleur

"shinydashboard" : Permet d’avoir des visuels de dashboard plus ergonomique

"rsconnect" : permet de déploiement de l’application sur le web

## Analyse du code :

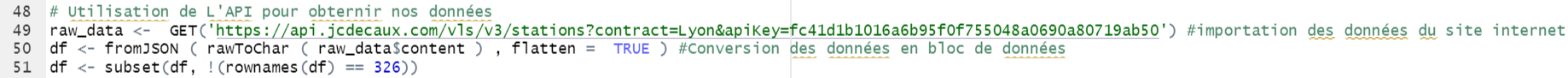
Le code est devisé en 3 scripts :

* Serveur
* UI
* App (lancement de l’app)

### Script Serveur :

Dans la partie serveur nous avons toutes les définitions de variables, graphiques, tables… La partie visuelle de l’application se gère dans la partie UI. Voici une explication des différents blocs de code dans la partie serveur :

#### Lignes 48-51 : Import des données



On importe les données de JCDecaux grâce à l’API dans la variable « raw\_data ». Puis, on converti les données de format JSON en chaine de caractère pour les exploiter. On met cette nouvelle base dans la variable « df »

#### Lignes 71-75 : CSV des codes postaux

Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement

Géocoding des adresse afin d’obtenir le code postal, ecriture et lecture du fichier csv « id\_cp.csv ». Puis mélange de df et id\_geocoded (lecture du fichier) avec comme attributs commun « number » dans la variable df

#### Lignes 78-82 : calcul du nombre de vélos disponibles total

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

Dans la variable « totvelodispo » on calcul la somme des vélos disponible et dans la variable « totplacedispo » on calcule la somme des places disponibles.

#### Lignes 88-113 : Création de la carte

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Initialisation de la carte « plot » en créant l’arrière-plan (tiles).

Ensuite on vérifie si l’utilisateur à crée un filtre « stationFilter ».

Si une station est filtré la carte sera zoomé sur les positions de «selected\_station » (lignes 98-99)

On créer une variable data on l’on met les positions de chaque stations (longitude, latitude, et nom 🡪 mag)

Puis on créer les étiquettes qui seront visibles sur la map avec le nom des stations, le nombre de places de vélos disponibles, le nombre de vélos disponibles ainsi que l’état de la station. Toutes ces données sont récupérées sur le data frame df.

#### Lignes 116- 128 : création des filtres par stations et codes postaux

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Création du filtre « selectedStations » :

L’on regarde si une station est sélectionné dans le filtre :

Si cela est le cas on retourne la station filtrée (ligne 119)

Sinon on renvoie toutes les stations de df (unique renvoie les stations qu’une seule fois) (ligne 121)

Création du filtre « selectedPostalCodes » :

L’on regarde si un code postal est sélectionné dans le filtre :

Si cela est le cas on retourne le code postal filtré (ligne 119)

Sinon on renvoie toutes les codes postaux de df (unique renvoie les stations qu’une seule fois) (ligne 121)

#### Lignes 134-146 : Nombre de vélos et places disponibles en fonction du code postal

Une image contenant texte, capture d’écran, Police, algèbre

Description générée automatiquement

Ce code créer la somme des vélos et places disponible en fonction du code postal filtré. En filtrant les bases de données avec le code postal puis en sommant les places et vélos disponibles.

#### Lignes 151-159 : nombre moyen de vélo par station par codes postaux

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Dans l’objet infoBoxVelosDispoMoy on filtre les stations par code postaux puis on somme les vélos disponibles qu’on divise par le nombre de vélo dispo « totvelodispo » qu’on met ensuite en pourcentage (x 100). Cette moyenne ira dans la variable « velos\_dispo\_moy ».

On note ensuite l’affichage (texte puis variable avec l’icône bicycle)

#### Ligne 164-172 : nombre de vélos disponibles en fonction de la station

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Dans l’objet infoBoxVelosDispo on filtre les stations par code postaux puis on somme les vélos disponibles. Cette somme ira dans la variable « velos\_dispo ».

On note ensuite l’affichage (texte puis variable avec l’icône bicycle)

#### Lignes 177-185 : Nombre de places disponibles en fonction de la station

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Dans l’objet infoBoxPlacesDispo on filtre les stations par station puis on somme les places disponibles. Cette somme ira dans la variable « places\_dispo ».

On note ensuite l’affichage (texte puis variable avec l’icône map-marker)

#### Lignes 190-196 : bouton refresh

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Création du bouton refresh :

Ligne 191 : regarde si le bouton refresh est activé, si c’est le cas :

On actualise les données dans la variable « raw\_data »

Ensuite on convertie les données de JSON en charactère dans la variable « new\_df ».

On mélange les données de « new\_df » et « id\_geocoded » avec comme attributs commun « number ».

Pour finir dans le cadre prévu à cet effet on note « Données rafraîchies 😊 »

#### Lignes 202-222 : Création du pie chart

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

On filtre la base de données « df\_filtred » sur le code postal sélectionné « selectedCodePostal ».

On calcul la variable « vélos\_disponibles », « vélos\_non\_disponible » et « total\_velos » selon la base de données filtrée.

On calcul ensuite le « pourcentage\_disponible » et « pourcentage\_non\_disponible » que l’on intègre au data frame « data »

On crée le graphique avec la base de données « data »

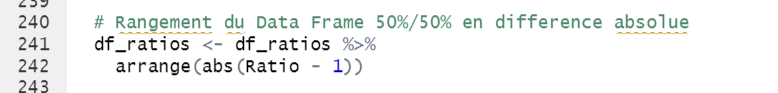
#### Lignes 228-238 : Création du df\_ratio

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement

Création de la variable «df\_ratio » qui contient la variable « Velo\_disponibles » (somme de tous les vélos disponibles), « Places\_disponible » (somme de tous les stands disponibles) et « Ratio » (Ve groupé par le code postal et le nom de la station. Les lignes sans ratio calculé sont supprimées.

#### Lignes 240-243 : tri du df\_ratio



Tri du data frame « df\_ratio » en priorisant les ratios le plus proche de 1.

#### Lignes 244-257 : Top 3 des stations du df ratio

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Si l’utilisateur filtre sur un code postal, on filtre le df\_ratio avec le code postal choisi et en ne gardant que les 3 meilleurs ratios que l’on stock dans la variable « top3\_stations\_filtered ».

On choisi ensuite d’afficher la variable sous forme de tableau.

#### Lignes 262-292 : Export du graphique en JPEG

Une image contenant texte, capture d’écran, Police, document

Description générée automatiquement

Export du graphique pie chart en format JPEG.

Code lignes 268 à 288 = 220 à 222

### Script UI :

Ce script permet à l’application de se mettre en forme avec le menu item, comment sont organisées les pages etc…

#### Lignes 13 – 26 : Sidebar Menu

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Définition du titre du tableau de bord : « Velo’v Dashboard » et du menu à gauche qui contient : Accueil, Station, le bouton refresh ainsi qu’un zone de texte intitulé « message de confirmation ».

#### Lignes 27 – 41 : Dashboard body

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Définition de la couleur et la taille des éléments dans le tableau de bord.

#### Lignes 43 – 59 : Accueil

Une image contenant texte, capture d’écran

Description générée automatiquement

Définition de la page « Accueil » :

Une première ligne avec :

* Une box :
  + Un titre
  + Une zone de filtre selectInput avec « stationFilter »
* Un infoBoxOutput avec « infoBoxVelosDispo »
* Un infoBoxOutput avec « infoBoxPlacesDispo »

Une deuxième ligne avec :

* Une box :
  + Un titre « Carte vélo’v de Lyon »
  + leafleOutput avec la carte « plot »

#### Lignes 60 -79 : Stations

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Définition de la page « Stations » :

Une première ligne avec :

* Une box
  + Un titre
  + Une zone de filtre selectInput avec « codePostalFilter »
* Un infoBoxOutput avec « infoBoxVelosDispoMoy »

Une deuxième ligne avec :

* Une box
  + Un downloadButton qui permet d’exporter le graphique avec «exportPieChart »
  + Un plotOutput avec le graphique « percentageChart »
  + Un titre
* Une box
  + Titre
  + Un tableOutput avec « top3StationsTable »

### Script Application :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

Permet d’exécuter l’application à l’aide de la fonction shinyApp.