CODE R POUR SONOREZÉ

4 janvier 2022

Tristan Lorino

Table des matières

1 Préalable

On commence par créer un projet RStudio qui sera contenu dans un dossier (déjà existant ou à créer). Dans ce dossier, on créé un dossier nommé `R` dans lequel on va stocker trois scripts R, qui seront à **exécuter dans cet ordre l'un après l'autre** : `import_database.R`, `Statistiques-v2.R` et `Graphiques.R`.

Le fichier `import_database.R` se charge en premier lieu de télécharger et installer un package nommé `pacman`, qui va se charger d'importer si besoin, et de charger tous les autres packages nécessaires à l'exécution des scripts R.

```
install.packages("pacman")
library(pacman)
pacman::p_load(thematic,lubridate,dplyr,tidyr,stringr,ggplot2,forcats,rgdal,sp,sf,
               rgeos,tidyr,devtools,leaflet,leaflet.extras,jsonlite,RColorBrewer,
               viridis,forcats,mapview,scales)
```

2 Importation des données

Le fichier `import_database.R` permet d'importer les données NoiseCapture relatives au projet Sonorezé.

Il nécessite l'emploi de quatre package : `{here}` pour utiliser des chemins relatifs dans le projet R, `{lubridate}` pour une utilisation poussée des dates, `{dplyr}` pour une gestion avancée des commandes et `{stringr}` pour une gestion poussée des chaînes de caractères.

Il faut au préalable créer un dossier (ou répertoire) `metabase` dans le dossier du projet RStudio. L'exécution du script R va créer un dossier `data` dans `metabase`, puis un dossier `unzip` dans `data`. Tous les fichiers zippés des traces vont être stockés dans `data`, puis vont être dézippés dans `unzip` : chaque extraction d'archive donne lieu à la création d'un dossier. Finalement on obtient, pour chaque trace, un dossier contenant deux fichiers : `meta.properties` et `track.geojson`.

On extrait ensuite :

- du fichier `meta.properties` : les champs `uuid` (pour lequel on ne conserve que les quatre premiers caractères), `Gain_calibration` et `method_calibration` ;

```
uuid <- meta %>%
        filter(startsWith(V1, 'uuid=')) %>%
        str_sub(., 6, 9)

      gain_cal<- meta %>%
        filter(startsWith(V1, 'gain_calibration=')) %>%
        str_remove(., "gain_calibration=")

      method_cal<- meta %>%
        filter(startsWith(V1, 'method_calibration=')) %>%
        str_remove(., "method_calibration=")
```

- du fichier `track.geojson` : les champs `leq_utc` (exprimé au format epoch, il est converti en date UTC), ceux des mesures de bruit pour les différentes fréquences, le Leq moyen, les coordonnées GPS et la précision (*accuracy*).

```
data <- data[c("leq_100", "leq_125", ...,"leq_12500","leq_16000","leq_mean","x","y","leq_utc","accuracy")]
```

0n va ensuite repositionner en première place les colonnes `Id, Date, x` et `y`, puis assembler ces différentes informations pour n'avoir plus qu'un seul tableau de données nommés `noisecapture_data`, que l'on va stocker dans le répertoire courant sous le nom `noisecapture_data.Rda`.

```
temp <- data %>%
        mutate(Date = lubridate::as_datetime(.$leq_utc/1000, tz="UTC")) %>%
        select(!leq_utc) %>%
        mutate(Id=uuid) %>%
        mutate(gain_calibration=gain_cal) %>%
        mutate(method_calibration=method_cal) %>%
        relocate(Id,Date,x,y)

      noisecapture_data <- bind_rows(temp, noisecapture_data)
      noisecapture_data
      save(noisecapture_data, file ="noisecapture_data.Rda")
```

3 Statistiques

3.1 Introduction

Outre les packages précédemment cités, on charge en plus le package `{forcats}`, qui permet de recoder facilement les modalités d'un facteur.

En premier lieu, on regarde le nombre de mesures (enregistrements au pas d'une seconde) réalisées. Il y en a :

```
load(here("noisecapture_data.Rda"))
data_nc <-as.data.frame(noisecapture_data)
dim(data_nc)[1]

## [1] 0
```

Ensuite on détecte les valeurs manquantes pour les coordonnées GPS, ainsi que les éventuelles dates aberrantes. Il y en a :

```
databer <- data_nc[data_nc$Date < "2021-11-01 00:00:01" | data_nc$x == "NA" | data_nc$y =="NA",]
dim(databer)[1]

## [1] 0
```

On calcule ensuite le nombre de mesures avec une précision supérieure à 20. Il y en a :

```
data_accuracy <- data_nc[data_nc$accuracy >= 20,]
dim(data_accuracy)

## [1] 0 0
```

Si l'on retire les mesures avec données manquantes ou précision insuffisante, on retient :

```
data_nc <- data_nc %>%
  filter(x != "NA") %>%
  filter(accuracy<20)

## Error: Problem with `filter()` input `..1`.
## i Input `..1` is `x != "NA"`.
## x objet 'x' introuvable

dim(data_nc)[1]

## [1] 0
```

soit 97,3 % des mesures.

3.2 Statistiques relatives aux mesures

On va créer différents tableaux de données :

- un nommé `AnaMes` pour le nombre de mesures par participants, avec pourcentage sur l'ensemble :

```
AnaMes <- data_nc %>%
  group_by(Id) %>%
  summarise(n = n()) %>%
  mutate(Pourcentage=n/sum(n)*100) %>%
  arrange(desc(n))
```

- un nommé `data_mes`, qui correspond à `data_nc`, auquel on ajoute deux champs, `DateJour` et `DateHeure` :

```
data_mes <- data_nc %>%
  mutate(DateJour=as.Date(Date)) %>%
  mutate(DateHeure=hour(as.POSIXct(Date)))
data_mes
```

- l'évolution du nombre de mesures par jour :

```
data_mes_jour <- as.data.frame(data_mes) %>%
  group_by(DateJour) %>%
  summarise(NbMes = n()) %>%
  mutate(NbMesCum=cumsum(NbMes))
```

- l'évolution du nombre de participants par jour :

```
data_ind <- data_mes %>%
  group_by(DateJour) %>%
  summarise(NbId = n_distinct(Id))
```

3.3 Statistiques relatives aux traces

À partir du tableau de données `data_mes`, on créé un tableau de donnnées `data_trace` avec :

- un identifiant des traces (plage de mesures consécutives) :

```
data_trace <- as.data.frame(data_mes) %>%
  arrange(Id,Date) %>%
  mutate(IdTrace=cumsum(c(TRUE, as.integer(diff(as.POSIXct(Date)), units = "secs") >= 2L)))
```

- deux variables relatives au jour, `DateJour` le jour sous la forme de date entière et `DateJourSem` le jour sous forme de jour de la semaine :

```
  mutate(DateJour=as.Date(Date)) %>%
  mutate(DateJourSem=wday(Date)) %>%
```

- et on calcule la durée de chaque trace en secondes et minutes :

```
  group_by(IdTrace) %>%
  mutate(DureeSec=n()) %>%
  arrange(Id,desc(DureeSec)) %>%
  mutate(DureeMinutes=DureeSec/60)
```

Ensuite on calcule le nombre de traces par participants, avec pourcentage sur l'ensemble :

```
data_trace %>%
  distinct(IdTrace, .keep_all = TRUE) %>%
  group_by(Id) %>%
  summarise(n = n()) %>%
  mutate(Pourcentage=n/sum(n)*100) %>%
  arrange(desc(n))
```

On créé ensuite une variable factorielle `Classe` pour des plages de durée :

```
data_trace <- data_trace %>%
  mutate(Classe = case_when(DureeMinutes < 5 ~ "0-5",
                            DureeMinutes < 10 ~ "5-10",
                            DureeMinutes < 15 ~ "10-15",
                            DureeMinutes < 20 ~ "15-20",
                            DureeMinutes < 25 ~ "20-25",
                            DureeMinutes < 30 ~ "25-30",
                            DureeMinutes < 35 ~ "30-35",
                            DureeMinutes < 40 ~ "35-40",
                            DureeMinutes < 45 ~ "40-45",
                            DureeMinutes < 50 ~ "45-50",
                            DureeMinutes < 55 ~ "50-55",
                            DureeMinutes < 60 ~ "55-60",
                            TRUE ~ "> 60")) %>%
  mutate(Classe = factor(Classe))
```

Mais les modalités ne sont pas dans le bon ordre :

```
levels(data_trace$Classe)
[1] "0-5"   "> 60"  "20-25" "15-20" "5-10"
```

Pour les réordonner, on fait appel au package `{forcats}` :

```
data_trace <- data_trace %>%
  mutate(Classe = forcats::fct_relevel(Classe,c("0-5","5-10","15-20","20-25","> 60"))) %>%
  arrange(Classe)
levels(data_trace$Classe)

[1] "0-5"   "5-10"  "15-20" "20-25" "> 60"
```

On détermine ensuite l'évolution du nombre de traces par jour et en cumulé :

```
data_trace_jour <- data_trace %>%
  group_by(DateJour) %>%
  distinct(IdTrace, .keep_all = TRUE) %>%
  summarise(NbTraceParJour = n()) %>%
  mutate(NbTraceParJourCum=cumsum(NbTraceParJour))
```

Enfin on détermine l'évolution du nombre de traces par jour de la semaine :

```
data_trace %>%
  group_by(DateJourSem) %>%
  distinct(IdTrace, .keep_all = TRUE) %>%
  summarise(NbTraceParJourSem = n()) %>%
  mutate(NbTracParJourSemCum=cumsum(NbTraceParJourSem))
```

4 Graphiques

4.1 Généralités

Pour avoir une virgule comme marque de décimale, on écrit :

```
options(OutDec=",")
```

Pour avoir un délimiteur de millier, on écrit (par exemple ici pour l'axe des ordonnées) :

```
scale_y_continuous(labels=function(x) format(x, big.mark = " ", scientific = FALSE), name="Effectif")
```

Pour des formats particuliers, comme la date, on fait appel au package `{scales}` :

```
scale_x_date(breaks = data_mes_jour$DateJour, name="Date", date_labels = "%d/%m")
```

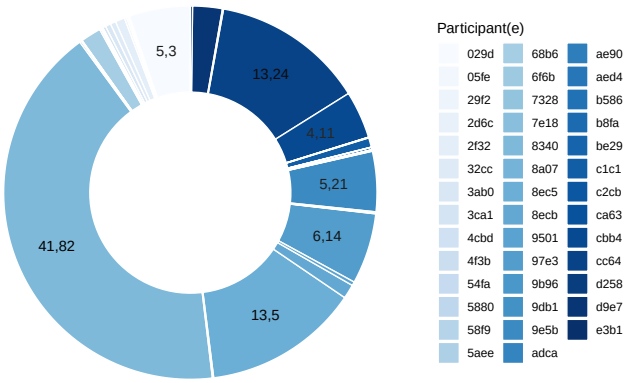
Enfin on sauvegarde les graphiques aux formats `png` et `pdf` dans un dossier spécifique qu'il aura fallu créer au préalable dans le dossier du projet (ce dossier spécifique s'appelle ici `Images`) :

```
p <- ggplot(...)
ggsave(device="pdf", here("Images","Pourcent_participant.pdf"),p)
ggsave(device="jpeg", here("Images","Pourcent_participant.jpeg"),p)
```

4.2 Graphiques relatifs aux individus

Le code suivant a été trouvé sur le net pour afficher un camembert creux (*donut*) :

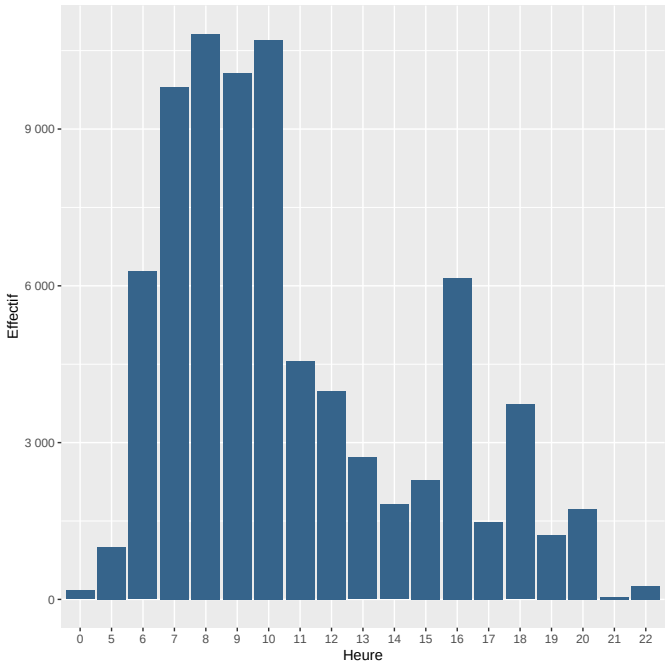
```
AnaMesDonut <- AnaMes %>%
  arrange(desc(Id)) %>%
  mutate(lab.ypos = cumsum(Pourcentage) - 0.5*Pourcentage) %>%
  arrange(desc(Pourcentage))

colourCount <- length(unique(AnaMesDonut$Id))
getPalette <- colorRampPalette(brewer.pal(9, "Blues"))
colgrey <- colorRampPalette(c("black", "white"))(colourCount)

p <- ggplot(AnaMesDonut, aes(x = 2, y = round(Pourcentage,2), fill = Id)) +
  geom_bar(stat = "identity", color = "white") +
  coord_polar(theta = "y", start = 0)+
  geom_text(aes(y = lab.ypos, label = ifelse(round(Pourcentage,2)>3,round(Pourcentage,2),"")), col = colgrey) +
  guides(colour = colgrey) +
  scale_fill_manual(name="Participant(e)", values=getPalette(colourCount)) +
  theme_void()+
  xlim(0.5, 2.5)
ggsave(device="pdf", here("Images","Pourcent_participant.pdf"),p)
ggsave(device="jpeg", here("Images","Pourcent_participant.jpeg"),p)
```



4.3 Graphiques relatifs aux mesures

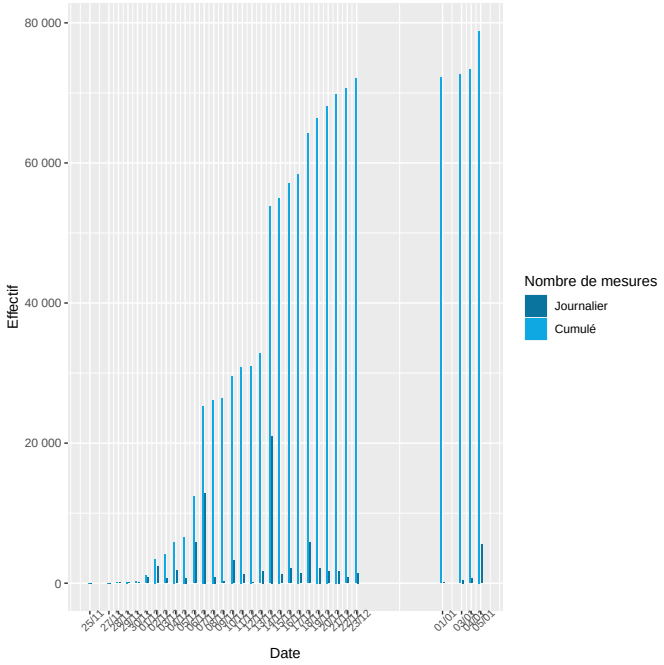
Pour la répartition des mesures en fonction de la période de la journée :

```
p <- ggplot(data_mes, aes(x = factor(DateHeure))) +
  geom_bar(fill="steelblue4") +
  scale_y_continuous(labels=function(x) format(x, big.mark = " ", scientific = FALSE), name="Effectif") +
  scale_x_discrete("Heure")
ggsave(device="pdf", here("Images","Distrib_heure.pdf"),p)
ggsave(device="jpeg", here("Images","Distrib_heure.jpeg"),p)
```



Pour la représentation graphique conjointe des nombre et nombre cumulé de mesures par jour, on a besoin de passer le tableau de données au format long (c.-à-d. de créer une colonne appelée `type` et comportant les valeurs `NbMes` et `NbMesCum`) :

```
p <-data_mes_jour %>%
  gather(., key="type",value="value",NbMes, NbMesCum) %>% #on passe au format long des données
  mutate(DateJour=as.Date(DateJour, format = "%d.%m.%Y")) %>%
  mutate(type = recode(type, NbMes = "Journalier", NbMesCum = "Cumulé")) %>%
  as.data.frame() %>%
  ggplot(aes(x=DateJour, y=value, fill=type)) +
  geom_col(width=0.4, position=position_dodge(width=0.5)) +
  scale_fill_manual(values=c("#10A8E3","#0A749F"), name="Nombre de mesures") +
  scale_y_continuous(labels=function(x) format(x, big.mark = " ", scientific = FALSE), name="Effectif") +
  scale_x_date(breaks = data_mes_jour$DateJour, name="Date", date_labels = "%d/%m") +
  theme(axis.text.x = element_text(size=8, angle=45)) +
  guides(fill = guide_legend(reverse=TRUE))
ggsave(device="pdf", here("Images","Mesures_distrib_jour.pdf"),p)
ggsave(device="jpeg", here("Images","Mesures_distrib_jour.jpeg"),p)
```

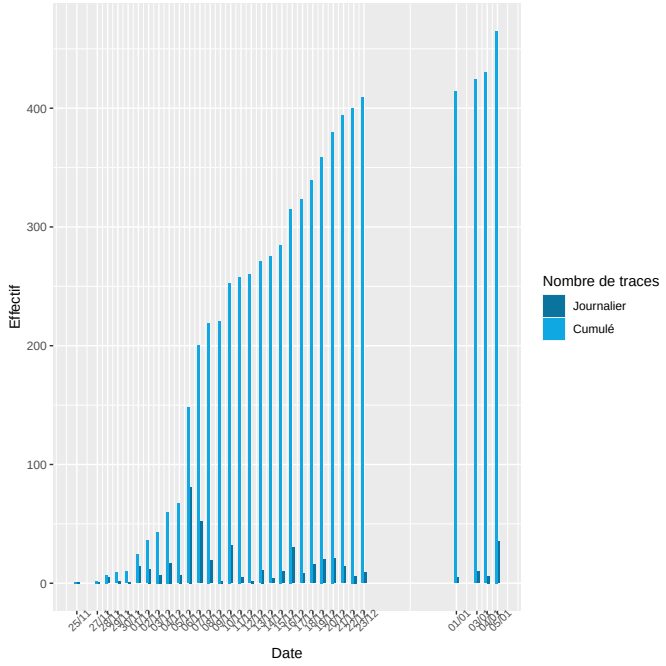


4.4 Graphiques relatifs aux traces

Pour la représentation graphique conjointe des nombre et nombre cumulé de traces par jour, il est encore nécessaire de passer au format long :

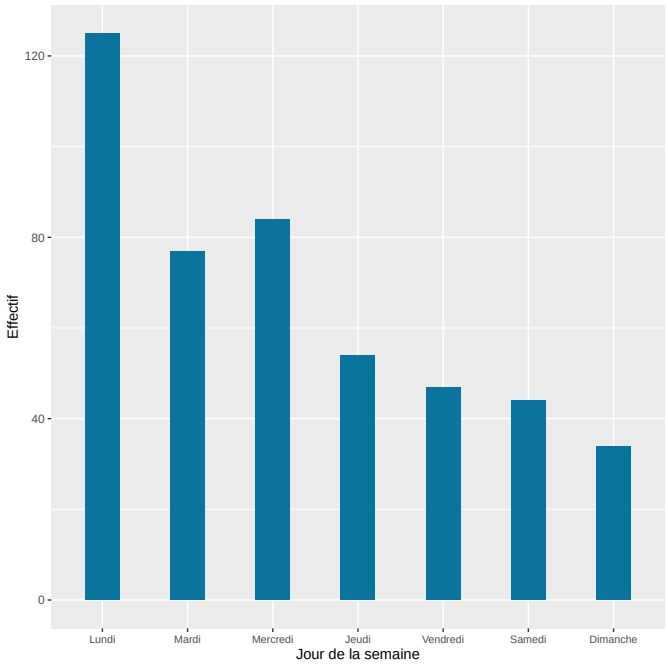
```
p <- data_trace_jour %>%
  gather(., key="type",value="value",NbTraceParJour, NbTraceParJourCum) %>% #on passe au format long des données
  mutate(DateJour=as.Date(DateJour, format = "%d.%m.%Y")) %>%
  mutate(type = recode(type, NbTraceParJour = "Journalier", NbTraceParJourCum = "Cumulé")) %>%
  as.data.frame() %>%
  ggplot(aes(x=DateJour, y=value, fill=type)) +
  geom_col(width=0.4, position=position_dodge(width=0.5)) +
  scale_fill_manual(values=c("#10A8E3","#0A749F"), name="Nombre de traces") +
  scale_y_continuous(name="Effectif") +
```

```
  scale_x_date(breaks = data_trace_jour$DateJour, name="Date", date_labels = "%d/%m") +
  theme(axis.text.x = element_text(size=8, angle=45)) +
  guides(fill = guide_legend(reverse=TRUE))
ggsave(device="pdf", here("Images","Traces_distrib_jour.pdf"),p)
ggsave(device="jpeg", here("Images","Traces_distrib_jour.jpeg"),p)
```



Pour la représentation graphique conjointe des nombre et nombre cumulé de traces par jour de la semaine :

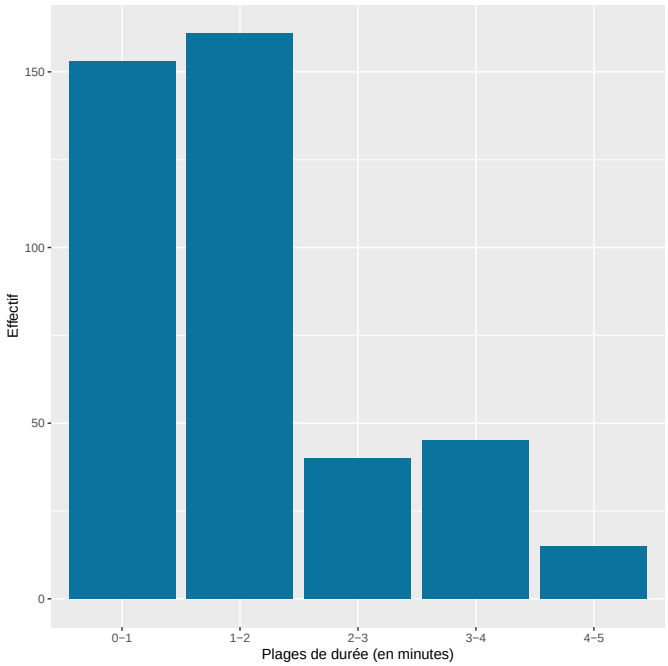
```
p <- data_trace_jour %>%
  mutate(JourJ=wday(as.POSIXct(DateJour), label = TRUE, abbr=FALSE)) %>%
  mutate(JourJ = factor(JourJ)) %>%
  mutate(JourJ=fct_relevel(JourJ,c("Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi","Dimanche"))) %>%
  arrange(JourJ) %>%
  as.data.frame() %>%
  ggplot(aes(x=JourJ,y=NbTraceParJour,fill=JourJ)) +
  geom_col(width=0.4, fill="#0A749F") +
  scale_y_continuous(name="Effectif") +
  scale_x_discrete(name="Jour de la semaine") +
  theme(axis.text.x = element_text(size=8)) +
  theme(legend.position = "none")
ggsave(device="pdf", here("Images","Traces_distrib_joursem.pdf"),p)
ggsave(device="jpeg", here("Images","Traces_distrib_joursem.jpeg"),p)
```



On va ensuite afficher la répartition des traces par plages de durées. D'emblée, pour avoir une échelle telle que les résultats soient lisibles, on va distinguer les traces dont la durée est inférieure à 5 min des autres.

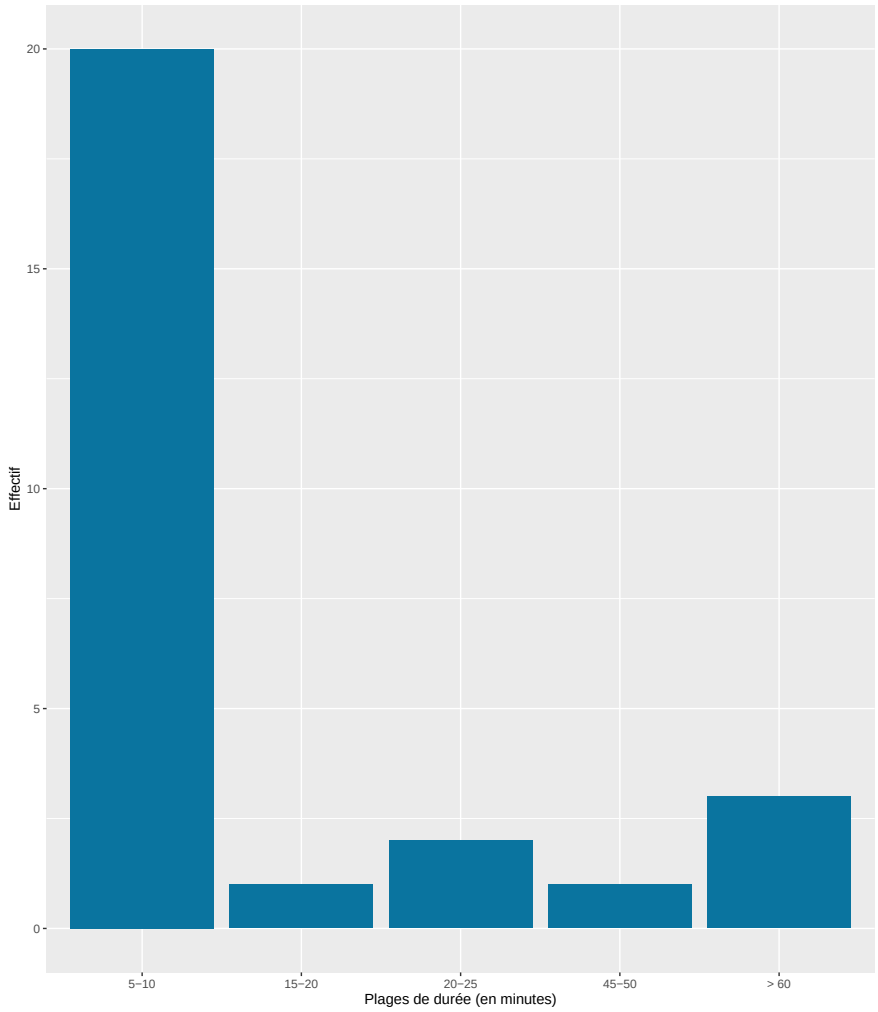
Pour les traces de courte durée :

```
trace_classe_inf5 <- data_trace %>%
  filter(DureeMinutes <= 5) %>%
  mutate(Classe = case_when(DureeMinutes < 1 ~ "0-1",
                            DureeMinutes < 2 ~ "1-2",
                            DureeMinutes < 3 ~ "2-3",
                            DureeMinutes < 4 ~ "3-4",
                            TRUE ~ "4-5")) %>%
  mutate(Classe = factor(Classe))
levels(trace_classe_inf5$Classe)

trace_classe_inf5 <- trace_classe_inf5 %>%
  distinct(IdTrace, .keep_all = TRUE) %>%
  mutate(Classe = forcats::fct_relevel(Classe,c("0-1","1-2","2-3","3-4","4-5"))) %>%
  arrange(Classe)
levels(trace_classe_inf5$Classe)

p <- ggplot(trace_classe_inf5,aes(x = Classe)) +
  geom_bar(fill="steelblue4") +
  scale_x_discrete(name="Plages de durée (en minutes)") +
  scale_y_continuous(name="Effectif", breaks= scales::pretty_breaks())
ggsave(device="pdf", here("Images","Traces_distrib_dureemin.pdf"),p)
ggsave(device="jpeg", here("Images","Traces_distrib_dureemin.jpeg"),p)
```



Pour les traces de plus longue durée :

```
trace_classe_sup5 <- trace_classe %>%
  distinct(IdTrace, .keep_all = TRUE) %>%
  filter(Classe != "0-5")

ggplot(trace_classe_sup5,aes(x = Classe)) +
  geom_bar(fill="steelblue4") +
  scale_x_discrete(name="Plages de durée (en minutes)") +
  scale_y_continuous(name="Effectif", breaks= scales::pretty_breaks())
ggsave(device="pdf", here("Images","Traces_distrib_dureemax.pdf"),p)
ggsave(device="jpeg", here("Images","Traces_distrib_dureemax.jpeg"),p)
```



4.5 Carto-graphiques

Merci à Pierre pour son initiation et le code ci-après ;-)

À noter qu'en théorie il est possible de sauvegarder directement les cartes dans des fichiers `png` grâce au package `{mapview}` (mais cela ne fonctionne pour l'instant pas pour moi) :

```
map <- leaflet(...)
mapshot(map, here("Images","Mesures_Polygones.png"),remove_controls = c("zoomControl"))
```

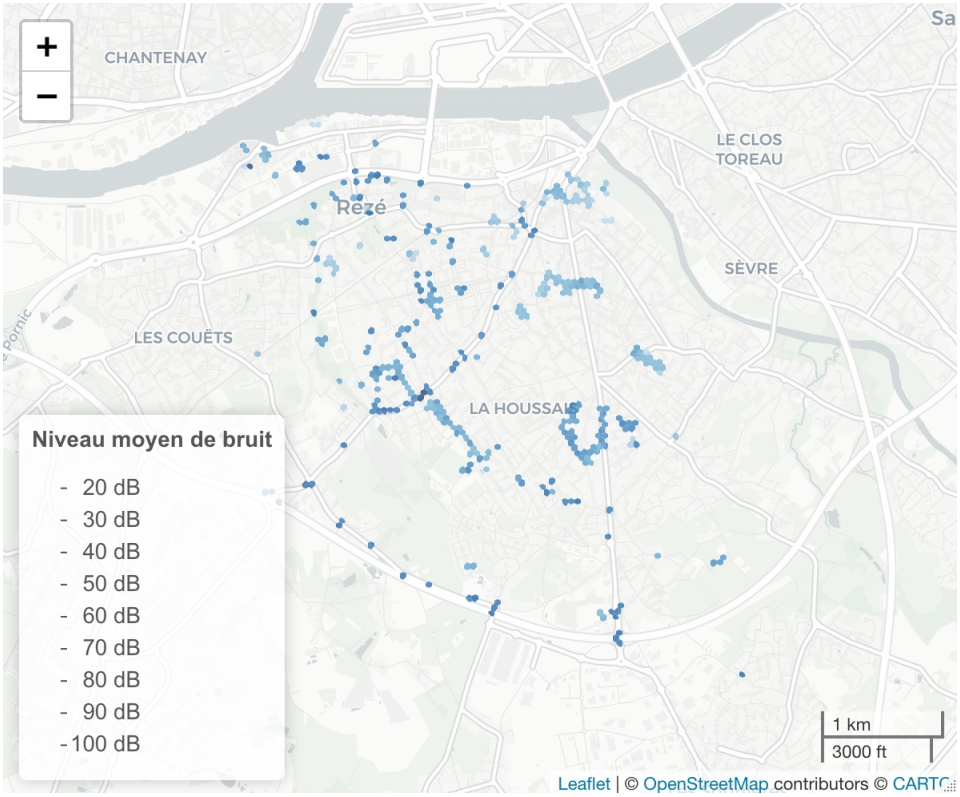
Il me reste un problème : la mise en page de la légende.
Pour évider un problème d'erreur survenant dans le calcul de la médiane ci-après, et qui exige des variables numériques, on ne retient que les variables d'intérêt :

```
Data_numerique <- data_nc[,c("x","y","leq_mean")]
```

Ensuite on représente les niveaux de bruits avec des polygones :

```
SPDF <- SpatialPointsDataFrame(coords=Data_numerique[,1:2], data=as.data.frame(Data_numerique))
SPDF_sf <- st_as_sf(SPDF, crs = 4326, agr = "constant", remove = F)
st_crs(SPDF_sf) <- 4326 #EPSG WGS84
```

```r
SPDF_sf_2154 <- st_transform(SPDF_sf, 2154) #LAMBERT 93

bbox <- st_bbox(SPDF_sf_2154)
grid <- sf::st_make_grid(st_as_sfc(bbox),cellsize = 50, square = FALSE)  #Grid of 50 meters
st_crs(grid) <- 2154

stations <- aggregate(x = SPDF_sf_2154, by = grid, FUN = median)         #Calculate median

stations <- stations %>%
  drop_na(geometry) %>%
  drop_na(leq_mean)

stations_sf <- st_as_sf(stations, crs = 2154, agr = "constant",
                        remove = F)

stations_sf_4326 <- stations_sf %>%
  st_transform(4326) # repasse en WGS84 (LATti LONgi )

pal2 <- colorNumeric(
  palette = "Blues", #"RdYlGn",
  #n = 9,
  domain = stations$leq_mean,
  #na.color = "transparent",
  #reverse = TRUE
  )

map <- leaflet(stations_sf_4326) %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  addScaleBar( position = c("bottomright"))%>%
  addPolygons(color = ~pal2(leq_mean), weight = 0, smoothFactor = 0.,
              opacity = 0.0, fillOpacity = 0.7,
              ) %>%
  addLegend("bottomleft",
            pal = pal2,
            #colors = ~pal2,
            values = stations$leq_mean,
            title = "Niveau moyen de bruit",
            #labels = pal2,
            labFormat = labelFormat(suffix = " dB",big.mark = " ", transform = identity),
            opacity = 1
            )
map
#
mapshot(map, here("Images","Mesures_Polygones.png"),remove_controls = c("zoomControl"))
```


CHANTENAY
LE CLOS TOREAU
Rezé
SÈVRE
LES COUËTS
LA HOUSSAIE
Niveau moyen de bruit
- 20 dB
- 30 dB
- 40 dB
- 50 dB
- 60 dB
- 70 dB
- 80 dB
- 90 dB
- 100 dB
1 km
3000 ft
Leaflet | © OpenStreetMap contributors © CARTO