# Principes, procédures et tips pour traiter les données dans R

PHilippe Gay Nicolas Bressoud

19/11/2022

# Table des matières

**Préface** **4**

**Comment lire ce livre** **5**

**1 Environnement de travail** **6**
1.1 GitHub et structure des dossiers . . . . . 6
1.2 Rédaction du code et grammaire . . . . . 7
1.3 RStudio et packages . . . . . 8
1.4 Sublime Text et Packages . . . . . 9
1.5 Scrivener 3 + Zotero + BetterBibText - Rédaction d'articles longs . . . . . 9

**2 Préparation des données** **10**
2.1 Bonnes pratiques dans le système de récolte des données (Qualtrics, ...) . . . . . 10
2.2 Premières étapes dans R (en principe, les noms des variables sont sains) . . . . 11
2.2.1 ajout de la variable de temps dans chaque df (cas simple) . . . . . 11
2.2.2 catégorisation de la variable de temps (cas avec horodatage préalable) . 12
2.2.3 création d'un data frame unique . . . . . 12
2.2.4 gestion des items à inverser . . . . . 13
2.2.5 calcul du score de chaque questionnaire par observation . . . . . 14
2.2.6 filtrage des observations . . . . . 14
2.2.7 gestion des données manquantes . . . . . 16
2.2.8 gestion des données extrêmes . . . . . 16

**3 Description des données** **17**
3.1 Obtenir une synthèse de données . . . . . 17
3.2 Principes . . . . . 18
3.3 Procédures . . . . . 18
3.4 Le cas des boucles . . . . . 18
3.5 Tableaux . . . . . 19
3.6 Plots . . . . . 19

**4 Rapport** **20**
4.1 Principes . . . . . 20
4.2 Procédures clés . . . . . 20
4.3 Un mot sur les normes APA . . . . . 20
4.4 Un mot sur Bookdown et GitHub . . . . . 20

**5 Etudes de cas** **21**
5.1 Data masking, Walrus opérator et sting interpolation . . . . . 21

**Références** **22**

# Préface

Pourquoi ce livre ? Contexte.

# Comment lire ce livre

philosophie, utilisation

# 1 Environnement de travail

Notre configuration de travail implique:

- *GitHub* qui agit comme lieu principal de **stockage**, de **communication** de codes ou/et résultats, de **versioning**.
- *RStudio* dont le fonctionnement avec `bookdown` est particulièrement appréciable
- *Sublime Text* qui doit être mieux connue mais permet de travailler sur des bouts de code avec une syntaxe couleurs et facilite ainsi le *copier/coller* d'anciens projets vers de nouveaux.

## 1.1 GitHub et structure des dossiers

On dispose d'un compte chez *GitHub*. Le compte est lié à 3 machines (2 macs et un pc). Le *commit* et le *push* se réalisent depuis RStudio directement. Sur chaque machine, on s'offre tout de même une solution *user friendly* avec *GitHub Desktop*.

Sur chaque machine, on dispose d'une structure des dossiers de type *Documents* > *GitHub* > *r-projects* (attention à la casse).

Le dossier *r-projects* contient autant de sous-dossiers que de projets à analyser.

Chaque sous-dossier a un nom structuré ainsi (attention à la casse) : *contexteannée_initialesauteurprincipal* (par exemple : *dupp2019_cr* ou *crips2019_lv*).

Dans chaque sous-dossier:

- un fichier *Rproj* est disponible et il porte le nom du sous-dossier
- un fichier présentant les données brutes (raw) est disponible (en lecture seule de préférence) et il porte un titre de la forme *nomsousdossier_raw*. Il peut être en format *.csv* ou *.xlsx*. Si les données brutes sont sur plusieurs fichiers (dans le cas de plusieurs temps de mesure, par exemple, on les distingue avec l'ajout *_t1*, …)
- un fichier de script *.R* intitulé *nomsousdossier_script*
- un fichier Rmarkdown *.Rmd* intitulé *nomsousdossier_rapport* et rédigé en parallèle du script. Ce rapport général appelle le script pour réaliser les sorties.

On ne s'est pas encore déterminé sur les bonnes pratiques pour la constitution du rapport en RMarkdown : Est-ce OK et satisfaisant de "simplement" rappeler le script et uniquement "afficher" les objets ?

- les sorties de type *HTML*, *PDF*, ou image (*png*, *svg*, ...) n'obéissent pas à des règles précises.
- plusieurs rapports peuvent coexister en fonction des destinataires ; ils sont tous une adaptation du rapport "master" décrit plus haut.

L'intérêt est d'avoir à tout moment sur GitHub une vision claire des modifications réalisées à travers les différentes étapes de mise à jour des fichier (traçabilité de la démarche). De plus, une attention particulière est accordée à l'écriture d'un code avec une grammaire (le plus possible) conventionnelle qui est lisible et commenté, que ce soit dans le script ou dans le rapport en RMarkdown.

Le script et le rapport se rédigent en parallèle.

## 1.2 Rédaction du code et grammaire

Le code suivant est, à notre sens, un exemple de bonne pratique car :

- des titres mis en évidence structurent le code
- les commentaires sont présents; ils se veulent précis et concis
- des espaces facilitent la lecture
- le code n'est pas - *à notre connaissance* - inutilement répétitif

```
###############
#visualisation#
###############

#score échelle HBSC

vis_hbs <- d_long_paired %>%
  ggplot() +
  aes(x = temps, color = group, y = hbs_sco) +
  geom_boxplot(alpha = .5) +
  geom_jitter(size = 5, alpha = .5, position = position_jitterdodge(dodge.width=.7, jitter
  stat_summary(fun = mean, geom = "point", size = 3, shape = 4) +
  stat_summary(fun = mean, aes(group = group), geom = "line") +
  labs(title = "Mesure des CPS", y = "Score au HBSC") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_brewer("Groupe", palette = "Set1")
```

Un autre exemple illustre cette grammaire. On rajoute quelques espaces pour faciliter la lecture et repérer les structures - *parfois nécessairement* - répétitives :

> On aimerait toutefois savoir comment éviter ces répétitions. Notamment quand on génère 10 graphiques qui ne varient, dans le code, qu'au niveau de l'axe Y et le titre, par exemple.

```
#Création des moyennes de chaque questionnaire pour chaque observation

d_long <- d_long %>%
  mutate(hbs_sco = rowMeans(select(.,starts_with("hbs")),na.rm =T),
         pec_sco = rowMeans(select(.,starts_with("pec")),na.rm =T),
         be_sco  = rowMeans(select(.,starts_with("be")) ,na.rm =T),
         est_sco = rowMeans(select(.,starts_with("est")),na.rm =T),
         cli_sco = rowMeans(select(.,starts_with("cli")),na.rm =T),
         sou_sco = rowMeans(select(.,starts_with("sou")),na.rm =T),
         mot_sco = rowMeans(select(.,starts_with("hbs")),na.rm =T))
```

**Une source pour la grammaire du codage peut être consultée à l'adresse :** https://www.inwt-statistics.com/read-blog/inwts-guidelines-for-r-code.html

## 1.3 RStudio et packages

On dispose en l'état de la version 4.0.2 de R ainsi que de la version 1.3.959 de RStudio.

Sur les macs, différents ajouts comme Xquartz ou des modules liés à LaTeX sont également installés.

> Mais honnêtement, on a perdu de vue leur rôle plus ou moins nécessaire sachant que LaTeX est certes nécessaire pour les sorties PDF sans que toutes les extensions liées à LaTex le soient… Ce sera à clarifier au prochain clean install.

Pour la version PC, on a installé tinyTEX 'tinytex::install_tinytex()' directement 'TinyTeX to C:/Users/nbr/AppData/Roaming/TinyTeX' depuis la console. On rappelle que `LaTeX` est nécessaire à la génération de *PDF*.

Dans R, les packages suivants sont installés:

- `tidyverse`, suite de packages pour travailler de manière *tidy* (bien rangé) : `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr`, `forcats`.
- `readxl`, permet de lire et importer les fichiers .xlsx.
- `bookdown`, permet de réaliser à peu de frais le présent livre

En principe, pour des raisons d'élégance du code, on cherche à limiter la sur-installation de nouveaux packages. Il s'agit d'explorer ce que les packages installés ont à offrir avant de courir sur d'autres fonctions vues sur le web.

## 1.4 Sublime Text et Packages

L'utilisation de Sublime Tex est ergonomique. Les packages du logiciel permettent de travailler comme R Studio, avec toutefois un environnement plus aéré et propice à la rédaction avec des codes couleurs agréables. Il est complémentaire à RStudio mais peut carrément le remplacer pour certaines courtes étapes de rédaction.

Les packages installés sont:

- sur PC : R-Box, R-IDE, LSP
- sur MAC : R-Box, SendCode

  On doit encore clarifier comment lier sur PC et MAC GitHub. Mais ce n'est pas prioritaire. GitHub sur Sublime Text ? Emmet, git, sublime github. https://gist.github.com/KedrikG/f7b955dc371b1204ec76ce862e2dcd2e

## 1.5 Scrivener 3 + Zotero + BetterBibText - Rédaction d'articles longs

On doit déterminer comment travailler en `RMarkdown` pour des articles longs via `Scrivener 3`. Ce logiciel a l'avantage de facilement découper l'article en plusieurs zones de travail et *fusionner* le tout à l'export. De plus, on peut lier dynamiquement la rédaction à Zotero via un fichier `Bibtex` ce qui est très intéressant.

On pense aussi à rassembler les éléments rédigés dans Scrivener et compiler le tout avec `Bookdown` sous R.

Un hypothétique *workflow" serait : script dans R > ébauche de rapport dans R (RMarkdown) > copier/coller de l'ébauche de rapport dans Scrivener > rédaction dans Scrivener en compatibilité avec `Bookdown` > export de Scrivener dans R > préparation de la sortie avec `Bookdown`.

Mais ce n'est pas optimal. Notre souhait est de pouvoir, p.ex., modifier une donnée dans la source des données (le fichier brut) et cliquer sur un (voire deux) boutons pour mettre à jour l'article final ! On n'y est pas encore !!

L'organisation de la rédaction est un gros chantier qui n'est pas du tout structuré en l'état.

# 2 Préparation des données

Au sein de ce chapitre, on cherche à atteindre le double objectif de/d' :

- expliciter une manière que l'on espère *élégante* et *efficace - Le minimum d'effort pour le maximum d'efficacité)* - de préparer les données
- se conformer aux apprentissages et pratiques en traitement des données (et progresser)

On cherche dans la mesure du possible à toujours travailler avec des *tidy datas* et à utiliser en priorité les fonctions de la série de packages de `tidyverse`.

## 2.1 Bonnes pratiques dans le système de récolte des données (Qualtrics, …)

Chaque participant · e reçoit un *id* partiellement anonymisé. Il est fourni par l'équipe de recherche (ce n'est pas construit par le ou la participant · e). L'*id* ne contient que des chiffres pour simplifier le problème des majuscules/minuscules à la saisie et éviter les confusions entre le *zéro* et la lettre *o* (oui oui, c'est du vécu). Ceci signifie que l'*id* dispose d'une structure qui permet de/d' :

- évaluer son authenticité
- faciliter la catégorisation des données
- repérer efficacement les *id* et leur catégorisation dans les traitement ultérieurs

  Exemple : un *id* comme *12.47.694* est structuré comme suit *groupe12.constantearbitraire47.nombrealéatoire*

Chaque *id* est **vérifié** dans Qualtrics, ce qui signifie qu'un · e participant · e doit confirmer l'*id* pour que le système passe à la suite. On relève aussi l'intérêt de demander à l'utilisateur · trice de **confirmer son groupe d'appartenance** en cochant un facteur dans une **liste imposée** dans Qualtrics. Ceci a l'avantage de repérer les saisies fantasques (une personne met un *id* bidon / un · e participant · e s'inscrit avec le bon *id* mais dans le mauvais groupe, …). Dans la préparation des données, il semble que l'on gagne ainsi un temps important pour trier les données valables. La confiance dans les données s'en trouve augmentée. On relève toutefois une faiblesse : le code n'est pas 100% anonyme et on peut toujours remonter à un groupe de participant · es. Il faut alors en amont s'engager à détruire le document qui a permis la génération des *id*.

Chaque questionnaire est identifié par 3 premières lettre significatives, le nombre d'items et un `_`:

> Exemple : Questionnaire sur la motivation scolaire en 13 items est identifié par `mot13`.

Ensuite, chacun des items est proprement nommé par ordre d'apparition, ce qui est fondamental, notamment pour le traitement des **items inversés** :

> Exemple : `mot13_1`, `mot13_2`, ..., `mot13_13` (il semble que Qualtrics ajoute par défaut le *underscore_* )

## 2.2 Premières étapes dans R (en principe, les noms des variables sont sains)

On commence par le chargement des packages et l'importation du ou des fichiers de données. Dans cet exemple, on prend le cas (plus complexe) où nous devons gérer et associer deux `data frames`.

```
library(tidyverse)
library(readxl)

# Importation des données Qualtrics à disposition ----
d_t1_raw <- read_excel("crips2019_lv_raw_t1.xlsx")
d_t2_raw <- read_excel("crips2019_lv_raw_t2.xlsx")
```

### 2.2.1 ajout de la variable de temps dans chaque df (cas simple)

Chaque fichier représente un temps de mesure. Il nous suffit d'ajouter une variable indiquant cette information.

Dans cet exemple, on règle en même temps le problème des *id* qui contiennent éventuellement des majuscules.

```
#Ajout de la variable temps sur chaque df et normalisation des id en minuscule.

d_t1 <- d_t1_raw %>%
  mutate(temps="temps 1",
         id=tolower(id))

d_t2 <- d_t2_raw %>%
```

```
  mutate(temps="temps 2",
         id=tolower(id))
```

### 2.2.2 catégorisation de la variable de temps (cas avec horodatage préalable)

Quand on possède un unique fichier regroupant toutes les observations, on peut catégoriser les données à partir de l'horodateur.

```
#modification de la variable "RecordedDate" en "date" en temps 1 et temps 2 tout en filtra
#mais on commence par la renommer.

d <- d %>%
  rename(date = RecordedDate) %>%  #dans cet ordre.
  mutate(id = tolower(id)) #gestion de la casse.

d <- d %>%
  mutate(
    date = case_when(date >= as.POSIXct("01.08.2019", format="%d.%m.%Y", tz="utc") & date
                     date >= as.POSIXct("01.06.2020", format="%d.%m.%Y", tz="utc") & date
                     TRUE ~ "autre temps")) #on privilégie case_when car on a 3 conditions

#Au passage, R a modifié le type de variable "date". On le laisser respirer... et on filtr

d <- d %>% filter(date =="temps 1" | date == "temps 2")
```

On est pas encore confiant sur la qualité du code ci-dessus.

### 2.2.3 création d'un data frame unique

A ce stade, à partir du cas simple, il ne semble pas y avoir de raison au maintien de 2 df différents. On peut créer un df unique à l'aide, simplement, de `bind_rows`. Cela nous évite de doubler les manipulations au temps 1 et au temps 2.

Au préalable, on s'est assuré que les noms des variables correspondaient à notre nomenclature, et que l'ordre des questions dans chaque questionnaire était le même au *temps 1* et au *temps 2*.

Les éventuelles nouvelles variables du temps 2 sont bien traitées grâce à l'ajout de `NA` pour les observations du temps 1.

```r
#Mise en formation long sur un df

d_long <- bind_rows(d_t1,d_t2)
```

On choisit la mise en format *long* (au lieu de *wide*) pour correspondre au attentes du traitement *tidy* des données. Cela signifie que chaque ligne est une observation et chaque colonne est une variable. Ceci implique que chaque participant · e se retrouve dans deux lignes : une concernant la modalité (facteur) **temps 1** et l'autre selon la modalité **temps 2**. C'était déroutant au début mais on a adopté cette manière de faire.

### 2.2.4 gestion des items à inverser

Les items à inverser sont traités en étant strictement remplacés via la fonction `mutate()` dans le cadre de l'enregistrement d'un nouveau `data frame`. R permet de remonter les changements donc cet écrasement n'empêche pas la vérification des processus, étape par étape.

```r
#Recondage des variables au score inversé

d_long <- d_long %>%
  mutate(hbs20__7 = 6 - hbs20__7,
         pec5__5 = 6 - pec5__5,
         be8__4 = 8 - be8__4,
         be8__5 = 8 - be8__5,
         be8__8 = 8 - be8__8,
         est10__3 = 5 - est10__3,
         est10__5 = 5 - est10__5,
         est10__7 = 5 - est10__7,
         est10__10 = 5 - est10__10,
         sou13__6 = 6 - sou13__6,
         sou13__9 = 6 - sou13__9,
         mot16__4 = 8 - mot16__4,
         mot16__8 = 8 - mot16__8,
         mot16__12 = 8 - mot16__12,
         mot16__15 = 8 - mot16__15,
         mot16__16 = 8 - mot16__16)
```

Le code paraît fastidieux et potentiellement source d'erreurs. De plus, ce recodage s'accompagne d'une feuille annexe sur laquelle on a noté avec prudence les items en cause et la formule pour inverser les scores… On ne sait mieux faire…

Ce procédé d'écrasement facilite l'analyse de la cohérence interne ou le calcul des scores par la suite (cf. chapitre *description*).

### 2.2.5 calcul du score de chaque questionnaire par observation

La variable qui contient le score global de chaque questionnaire par participant · e porte l'extension *_sco* :

> Exemple : `mot13_sco` est la variable de score total de `mot13_1` à `mot13_13` avec les variables qui ont été inversées (sans conservation dans le df de l'item non-inversé).

```
#Recondage des variables au score inversé

#Création des moyennes de chaque questionnaire pour chaque observation

d_long <- d_long %>%
  mutate(hbs_sco = rowMeans(select(.,starts_with("hbs")),na.rm =T),
         pec_sco = rowMeans(select(.,starts_with("pec")),na.rm =T),
         be_sco  = rowMeans(select(.,starts_with("be")) ,na.rm =T),
         est_sco = rowMeans(select(.,starts_with("est")),na.rm =T),
         cli_sco = rowMeans(select(.,starts_with("cli")),na.rm =T),
         sou_sco = rowMeans(select(.,starts_with("sou")),na.rm =T),
         mot_sco = rowMeans(select(.,starts_with("hbs")),na.rm =T))
```

> De nouveau, on ne crée pas d'objet intermédiaire sachant que R sait très bien nous permettre de vérifier le processus, étape par étape.

Ce code paraît très efficace. Il se base sur notre nomenclature sans risque d'erreur, indépendamment du nombre de variable ou de la position des colonnes.

> On apprécie, même si on ne comprend pas encore à satisfaction la fonction ´select´et son fonctionnement.

### 2.2.6 filtrage des observations

Ici, on a essayé de développer une technique pour ne garder que les *id* qui se retrouvent strictement au temps 1 et au temps 2.

Les enjeux sont cruciaux :

- on doit s'assurer qu'un même *id* ne se retrouve pas plusieurs fois dans le même temps (un · e participant · e peut avoir rempli 2 fois le questionnaire du temps, p.ex.)

- on doit aussi s'assurer que nous ferons nos comparaisons temps 1 / temps 2 sur des données complètes

C'est notre présomption de base. Il est dès lors important de filtrer les observations de manière stricte.

Pour cela, nous avons :

- créé un df de comparaison pour pouvoir dire à R quoi sélectionner. Ce df va, en deux étapes, ne garder que (1) les *id* uniques dans chaque temps puis (2) constituant une paire (t1, t2).

```
d_comp <- d_long %>%
  drop_na(id) %>% #par sécurité, on supprime les lignes dont l'id est vide
  arrange(id) %>% #visuel uniquement
  group_by(id, temps) %>%
  count(id) %>%
  filter(n==1) %>% #On s'est assuré que chaque id est unique dans chaque modalité de temps
  ungroup() %>%
  group_by(id) %>%
  count(id) %>%
  filter(n==2) %>% #on ne garde que les paires de id qui se retrouvent dans t1 et t2. C'es
  ungroup()
```

> On devrait vérifier la validité de cette méthode, sur le plan des bonnes pratiques stat, et aussi sur le plan du codage dans R...et aussi s'assurer qu'elle fonctionne bien comme on le pense en la mettant à l'épreuve.

- créé un nouveau df qui ne contient plus que les paires souhaitées

```
#Notre df de comparaison est prêt. On peut procéder à l'élagage de d_long.

d_long_paired <- d_long %>%
  filter(id %in% d_comp$id) %>%
  mutate(group=ifelse(group=="con", "contrôle","expérimental"))
```

> On a été un peu sauvé avec le logique `%in%` mais sans bien savoir pourquoi `==` ne convient pas. Dans l'example, on en a profité pour modifier les facteurs de la variable `group`. Les forums ne sont pas clairs sur la différence entre `ifelse`et `if_else`ou encore `case_when`. On doit clarifier cela.

### 2.2.7 gestion des données manquantes

pas encore de bonnes pratiques.

### 2.2.8 gestion des données extrêmes

pas encore de bonnes pratiques.

# 3 Description des données

## 3.1 Obtenir une synthèse de données

Voici deux exemples pour obtenir des petits résumés.

```
d_long_paired_sum <- d_long_paired %>%
   mutate(sex=ifelse(sex=="1", "garçons","filles")) %>%
   group_by(temps, group, sex) %>%
  summarise(n=n())

d_long_paired_sum2 <- d_long_paired %>%
  group_by(temps, group) %>%
  summarise(n=n(),
            mean_hbs=mean(hbs_sco),
            mean_pec=mean(pec_sco),
            mean_be=mean(be_sco),
            mean_est=mean(est_sco),
            mean_cli=mean(cli_sco),
            mean_sou=mean(sou_sco),
            mean_mot=mean(mot_sco),
            mean_sho1=mean(sho_1),
            mean_sho2=mean(sho_2),
            mean_sho3=mean(sho_3),
            mean_sho4=mean(sho_4))
```

## 3.2 Principes

## 3.3 Procédures

## 3.4 Le cas des boucles

La fonction `for` semble très utile pour faire faire des boucles. Quand je dois réaliser 10 plots de 10 VI, je peux juste préparer mon plot et l'intégrer dans une boucle. La contrainte semble être liée à `aes` et peut-être aussi au titre du plot dont on aimerait aussi "automatiser" la génération.

Infos à comprendre et tester ici : https://statistique-et-logiciel-r.com/comment-utiliser-ggplot-dans-une-boucle-ou-dans-une-fonction/

intégrer et expliquer le cas d'école réussi avec les données IBE:

```
#######################
#visualisation en loop#
#######################
# Liste des noms des variables que l'on veut (a à p uniquement).
var_list = names(d_paired)[6:21]

# création de la liste pour accueillir les 16 (21-5) plots
plot_list = list()

for (i in 1:16) {
  p <- d_paired %>%
    ggplot() +
    aes(x = dat) +
    aes_string(y = var_list[i]) +
    geom_jitter(size = 5, alpha = .5, width = 0.3) +
    stat_summary(fun = mean, geom = "point", size = 3, shape = 4, color = "red") +
    stat_summary(fun = mean, geom = "line", aes(group = 1), color = "red") +
    labs(title = paste("Mesure item",i), y = "Score") +
    theme(plot.title = element_text(hjust = 0.5))
  plot_list[[i]] = p
}
dev.off()

# enregistrement des plots en png par fichier séparé avec un nom correspondant au nom de l
for (i in 1:16) {
```

```
  temp_plot = plot_list[[i]]
  ggsave(temp_plot, file=paste0("plot_", var_list[[i]],".png"), width = 14, height = 10, u
}
```

## 3.5 Tableaux

## 3.6 Plots

intégrer mes trouvailles de crips2019 et ce qu'un plot doit montrer.

Travail avec Zoe. Notes à intégrer :

Réaliser des boxplots avec barres d'erreurs

[ ] –> plutôt utiliser select des variables ou filter des participants gather –> 1er argument nom et 2ème argument valeur

set.sid dans Rmarkdown pour figer les aléatoires, p.ex. le jitter pour pas que ça change à chaque lancement.

installer des kits de couleurs au besoin.

shape : les numéros correspondent à des formes de points.

geom_line : 3 arguments minimum. avec group. –> pourquoi "group" et pourquoi "1". Alt + N = ~

jitter –> position et positionjitterdodge pour comparer G et F

Afficher toutes les catégries, même les vides :

```
    scale_x_discrete(drop = FALSE) # Forcer l'affichage des catégories vides
# A voir si la variable doit être un facteur
d <- d %>%
  mutate(
         q1 = factor(q1, levels = c("1","2","3","4"), labels = c("pas du tout", "plutôt no
         q2 = factor(q2, levels = c("1","2","3","4"), labels = c("pas du tout", "plutôt no
         q3 = factor(q3, levels = c("1","2","3","4"), labels = c("pas du tout", "plutôt no
         q4 = factor(q4, levels = c("1","2","3","4"), labels = c("pas du tout", "plutôt no
  )
```

# 4 Rapport

## 4.1 Principes

Avec ressources (fameux document PDF de CS durant le pre-doc) Ce qui est attendu dans un document de type rapport. (à l'aide du rapport de ZL et de celui en production pour LV)

Kable et KableExtra

bon affichage d'un tableau

insérer image externe

## 4.2 Procédures clés

## 4.3 Un mot sur les normes APA

## 4.4 Un mot sur Bookdown et GitHub

cf. chapitre 1.

# 5 Etudes de cas

## 5.1 Data masking, Walrus opérator et sting interpolation

Exemple de Mutate dans une fonction ou même au sein d'un loop. voir sandbox.

```
# avec tunnel. pas essayé encore.
fun <- function(df, var) {
  df <- df %>%
    mutate("{var}" := key - {{var}})
}
# avec pull(). OK.

fun <- function(df, var) {
  df <- df %>%
    mutate("{var}" := key - pull(.,var))
}
```

# Références