Missions

Première mission :

En migrant les données d'un client d’une base de données à l’autre, il a été constaté que le nombre de répondants pouvait ne pas être le même. Il s'agissait alors d'identifier les clients avec ces problèmes d'incohérence, puis si tel était le cas, mettre en saillance les identifiants des répondants qui posaient problème.

1. Chargement des données

Dans un premier temps j'ai chargé les données et gardé uniquement les clients sur Sharing-Data et Feedback-Bime.

```{r}

rm(list =ls())

# Library

library(data.table)
library(uptasticsearch)
library(lubridate)
library(tidyverse)
library(stringr)
library(knitr)
library(kableExtra)
library(mailR)

setwd("C:/Users/travail02/Documents/Valérie")

# Lister l'ensemble des index

index_list = fread("http:...")

qbodySD <- '{
"query": {
"range": {9
"respondentDateEnd": {
"gte": "now-2M",
"lte": "now",
"format": "dd/MM/yyyy||yyyy"
}
}
}
}'
```

```
# Dans la table index_list, suppression des colonnes filter, routing.index et routing.search
index_list$filter = NULL
index_list$routing.index = NULL
index_list$routing.search = NULL

# Initialisation du vecteur type_base avec une chaîne de caractères vide
type_base <- rep("", length(index_list$alias))

# Création d'une nouvelle colonne (type_base) au dataframe index_list
index_list <- data.frame(index_list, type_base=type_base, stringsAsFactors = FALSE)

# Dans la colonne type_base, pour les clients (dans la colonne alias) commençant par
# client- mettre sharing-data et pour ceux par feedback-bime- feedback-bime
index_list$type_base[grepl(pattern = "client-", x = index_list$alias)] = "sharing-data"
index_list$type_base[grepl(pattern = "feedback-bime-", x = index_list$alias)] = "feedback-
bime"

# Dans la colonne type_base, pour les clients (dans la colonne alias) commençant par
# feedback-preprod- et reviews- mettre autre
index_list$type_base[grepl(pattern = "feedback-preprod-", x = index_list$alias)] = "autre"
index_list$type_base[grepl(pattern = "reviews-", x = index_list$alias)] = "autre"

# Enlever les lignes autre dans le dataframe index_list
index_list <- index_list[!index_list$type_base=="autre", ]
```

2. Créer une fonction qui permet d'interroger Elastic Search avec trois inputs : le type de base, le client et le nombre de répondants sur les deux derniers mois

Cette fonction permet d'obtenir le nombre de répondants selon le client en appelant son index. Néanmoins dans Sharing-Data la colonne "respondentDateEnd" correspond à "experience.feedback.date" dans Feedback-Bime. Ainsi il fallait le préciser dans la fonction.

```
```{r}

connect_client <- function(index_call,type_base){

  if (type_base == 'feedback-bime') {
    qbodySD = gsub(pattern = "respondentDateEnd", replacement =
"experience.feedback.date", qbodySD)
  }

  client <- es_search(
    es_host = 'http:…'
    , es_index = index_call
    , size = 5000
    , query_body = qbodySD
    , n_cores = 3
```
```

```
  )
  return(client)
}
```

3. Boucle pour enregistrer chaque client

Cette boucle permet d'enregistrer dans une variable la connexion de chaque client via son index en prenant en compte son type de base. Mais également il s'agit de remplacer le "-" par "_".

```{r}

for (i in 1:length(index_list$alias)) {
     v_client <- assign(gsub(pattern = "-",replacement = "_", index_list$alias[i]),
                        connect_client(index_list$alias[i], index_list$type_base[i]))
}
```

4. Ajout de la colonne nb_repondants

Il s'agissait de rajouter une colonne nb_repondants au dataframe servant à indiquer le nombre de répondants pour chaque client.

```{r}

# Création de la colonne nombre de répondants initialisée à NaN
nb_repondants <- rep(NaN, length(index_list$alias))

# Ajout de cette colonne au dataframe
index_list <- cbind(index_list, nb_repondants)

# Création d'une boucle pour remplir la colonne nouvellement créée
for (i in 1:length(index_list$alias)) {
# Appel à la fonction connect_client() et sauvegarde des données dans la variable v_client
v_client <- get(gsub(pattern = "-",replacement = "_", index_list$alias[i]))
# Sauvegarde du nombre de répondants dans le dataframe. On enregistre le nombre de
# lignes avec la fonction dim()
# avec la fonction ifelse, on remplace les NULL par des zéros sinon la boucle ne démarre
# pas
index_list$nb_repondants[i] <- ifelse(is.null(dim(v_client)), 0, dim(v_client))
```

5. Ajout de la colonne client

J'ai rajouté une colonne donnant les noms des clients.

```{r}

# Création de la colonne client initialisée à NaN
client <- rep(NaN, length(index_list$alias))

# Ajout de cette colonne au dataframe
index_list <- cbind(index_list, client)

# Dans la colonne client rajouter le contenu de la colonne alias
# en y enlevant les préfixes client- et feedback-bime-
for (i in 1:length(index_list$client)){
  if (index_list$type_base[i] == 'sharing-data') {
    index_list$client[i] <- gsub("client-", "",index_list$alias[i])
  }
}

for (i in 1:length(index_list$client)) {
  if (index_list$type_base[i] == 'feedback-bime') {
    index_list$client[i] <- gsub("feedback-bime-", "",index_list$alias[i])
  }
}

# Conversion de la classe colonne client du type numeric au type character
index_list$client <- as.character(index_list$client)
```

6. Conversion de erdf vers enedis

J'ai appliqué cette conversion car Enedis n'avait pas le même nom sur Sharing-Data et Feedback-Bime

```{r}

index_list$client = gsub("erdf","enedis", x=index_list$client)
index_list$alias = gsub("erdf","enedis", x=index_list$alias)
index_list$index = gsub("erdf","enedis", x=index_list$index)
```

7. Sélection des clients uniquement à la fois sur Sharing-Data et Feedback-Bime et suppression des autres

```{r}

# Sélection des clients qui sont à la fois sur Sharing-Data et Feedback-Bime
resultat_intersect <- intersect(index_list$client[index_list$type_base=="sharing-data"],
                                index_list$client[index_list$type_base=="feedback-bime"])
```

```
# On crée une nouvelle colonne "a_supprimer" que l'on initialise à TRUE
# puis on l'ajoute au dataframe
a_supprimer <- rep(TRUE, times=length(index_list$client))
index_list <- cbind(index_list, a_supprimer)

# Boucle qui parcourt chaque client. Si le client existe dans l'intersection, cela signifie
# qu'il est présent dans les 2 types de base, donc il ne faut pas le supprimer, c'est pourquoi
# on met la valeur de la colonne "a_supprimer" à FALSE pour ce client
  for (i in 1:length(index_list$client)) {
    if (index_list$client[i] %in% resultat_intersect) {
      index_list$a_supprimer[i] = FALSE
    }
  }

# Suppression des clients qui ne sont pas dans les deux types de base : Sharing-Data et
Feedback-Bime
index_list <- index_list[(index_list$a_supprimer == FALSE),]
```

8. Ajout d'une colonne egal

Cette colonne indique OK quand le nombre de répondants sur Sharing-Data et Feedback-Bime est le même, NOK dans le cas contraire.

```
```{r}

# La fonction count() permet de compter le nombre de fois qu'un client apparaît dans le
# dataframe, ce qui donne le nombre de répondants pour chaque client que l'on stocke dans
# un nouveau dataframe nommé « egalite ». Client est la colonne qui contient tous les clients
# uniques et la colonne nb_repondants le nombre de répondants.
egalite <- plyr::count(index_list, vars=c("client","nb_repondants"))

# On joint les 2 dataframes sur les colonnes "client" et "nb_repondants"
# copy = FALSE pour joindre le dataframe de gauche (on ne crée pas de nouveau
# dataframe)
# donc on joint les données de la colonne "egalite" sur notre dataframe principal index_list
index_list <- dplyr::left_join(index_list, egalite, by=c("client","nb_repondants"),
copy=FALSE)

# Clients côte à côte dans le tableau pour faciliter la lecture
index_list<-index_list[order(index_list$client),]

# Renommer la colonne freq egal
names(index_list)[6] <- "egal"

# Remplacer dans la colonne egal, 1 par NOK
# Remplacer dans la colonne egal, 1 par OK
index_list$egal <- gsub("1", "NOK", x = index_list$egal)
```
```

```
index_list$egal <- gsub("2", "OK", x = index_list$egal)
```

9. Création d'un tableau HTML

Quand c'est OK, c'est vert et quand c'est rouge c'est NOK.

```{r}

# Faire un tableau faisant ressortir les infos qu'on veut
# En vert, OK et en rouge, NOK
table_html <- index_list %>%
  mutate(
    egal = cell_spec(egal, format="html", color=ifelse(egal!="OK", "red", "lightgreen"))
    #, background=ifelse(index_list$egal!="OK", "#ffe6e6", ""))
  ) %>%
  kable(format="html", escape=F) %>%
  kable_styling(
    bootstrap_options=c("striped","condensed","responsive"),
    full_width=F,
    fixed_thead=T
  )

# Ecriture de la variable table_html (contenant notre tableau formaté) dans un fichier html
chemin_complet_et_nom_fichier <- "C:\\Users\\travail02\\Documents\\Valérie\\table.html"
filecon <- file(chemin_complet_et_nom_fichier) # connexion
writeLines(table_html, filecon)
close(filecon)
```

Voici ci-dessous un exemple de tableau HTML avec des données différentes de celles de l'entreprise :

alias	type_base	nb_repondants	index	client	egal
feedback-bime-auchan	feedback-bime	5	feedback-bime-auchan	auchan	OK
client-auchan	sharing data	5	client-auchan	auchan	OK
client-enedis	sharing data	10	client-enedis	enedis	OK
feedback-bime-enedis	feedback-bime	10	feedback-bime-enedis	enedis	OK
client-speedy	sharing data	2	client-speedy	speedy	NOK
feedback-bime-speedy	feedback-bime	7	feedback-bime-speedy	speedy	NOK

10. Envoi par mail de la table

```{r}
```

```
send.mail(from = "Vgrimault@satisfactory.fr",
        to = c("Vgrimault@satisfactory.fr"),
        subject = "Nb_repondants Sharing-Data et Feedback",
        html = T,
        inline = T,
        body = "message_mail",
        smtp = list(host.name = "outlook.office365.com", port = 587, user.name =
"Vgrimault@satisfactory.fr", passwd = "Passwordd92999", tls = T),
        authenticate = T,
        send = T,
        encoding = "utf-8",attach.files = "table.html")
```

11. Repérer les identifiants des répondants qui posent problème

```{r}

# On calcule la longueur maximum des clients sur les deux types de base
longueur_max <- length(unique(index_list$client))

# Recodage ERDF vers Enedis
client_enedis = client_erdf
rm(client_erdf)

# On fait une boucle pour identifier les répondants qui n'apparaissent pas sur Feedback-
# Bime
for (i in 1:longueur_max) {
# on récupère le dataframe correspondant à chaque client selon le type de base (Sharing-
# Data ou Feedback-Bime)
index_list_alias_sd <-get(gsub("-","_",index_list$alias[index_list$client ==
unique(index_list$client)[i]
                                        & index_list$type_base == "sharing-data"]))
index_list_alias_fb <- get(gsub("-","_",index_list$alias[index_list$client ==
unique(index_list$client)[i]
                                         & index_list$type_base == "feedback-bime"]))

# Dans la colonne `_type` pour Sharing-Data, on enlève le préfixe questionnaire_
index_list_alias_sd$`_type`=gsub("^questionnaire_", "", x=index_list_alias_sd$`_type`)

# On concatène les colonnes type et contactId pour que les identifiants sur Sharing-Data
# et Feedback-Bime soient les mêmes afin de pouvoir les comparer
index_list_alias_sd$`_id`<- paste(index_list_alias_sd$`_type`,
index_list_alias_sd$contactId, sep="_")

# On cherche à savoir quels _id de Sharing-Data ne sont pas présents dans Feedback-Bime
assign(paste(unique(index_list$client)[i],"ERROR",sep="_"),
     index_list_alias_sd[!(index_list_alias_sd$'_id'%in% index_list_alias_fb$'_id'),]$`_id`)
```

```
#index_list_fb$alias[i] <- index_list_alias_fb[!(index_list_alias_fb$'_id'%in%
#index_list_alias_sd$'_id'),]

  }

print(index_list)
```

J'ai tout d'abord notamment sélectionné les clients à la fois sur Feedback-Bime et Sharing-Data. Une fois sélectionnés, il s'agissait de comparer le nombre de répondants sur ces deux bases. Les résultats ont été générés dans un tableau avec une colonne nommée " égal " où il est écrit en vert OK si le nombre est égal et NOK s'il ne l'est pas. Ces tableaux sont ensuite automatiquement envoyés par mail. Dans le cas d'une non-égalité, une fonction permet de repérer les identifiants des répondants qui sont sur la base Sharing-Data, mais ne sont pas sur la base Feedback-Bime. Il suffit alors de mettre le nom du client puis ajouter _ERROR pour obtenir cette liste.

Deuxième mission :

Les réponses aux questionnaires sont retranscrites dans des colonnes dont l'entête est sous la forme questionIdn°.lineIdn°, ayant uniformisé le nom de ces entêtes sur Sharing-Data et Feedback-Bime. A chaque réponse est appliqué un code. Il s'agissait dans ces colonnes, pour chacun des questionnaires de chaque client, pour la base Sharing-Data, de remplacer ce code par un autre code indiqué dans le tableau de correspondance. Dans ce dernier, il fallait dans la colonne "reponse_ID" trouver son équivalence dans la colonne "Colonne_EnTete", après avoir bien enlevé les balises HTML dans la colonne "Colonne_EnTete". Le but étant que le code de Feedback-Bime soit aussi appliqué dans Sharing-Data. En effet, je n’ai pas eu le temps pour finir cette mission, mais il était de question de mettre les données de Sharing-Data et Feedback-Bime ensemble, pour pouvoir comparer chaque entreprise à la moyenne des autres entreprises du même secteur. Ce qui en l’état n’est pas possible.

1. Chargement des données pour Sharing-Data

Dans un premier temps j'ai chargé les données et gardé uniquement les clients sur Sharing-Data. Egalement j'ai renommé la colonne alias client et le dataframe index_list df.

```{r}

rm(list =ls())

# Library

library(data.table)
library(uptasticsearch)
library(lubridate)
library(tidyverse)
library(stringr)
library(knitr)
```

```
library(kableExtra)
library(mailR)

setwd("C:/Users/travail02/Documents/Valérie")

# Lister l'ensemble des index

index_list = fread("http:...")

qbodySD <- '{
"query": {
"range": {
"respondentDateEnd": {
"gte": "now-2M",
"lte": "now",
"format": "dd/MM/yyyy||yyyy"
}
}
}
}'

# Dans la table index_list, suppression des colonnes filter, routing.index et routing.search
index_list$filter = NULL
index_list$routing.index = NULL
index_list$routing.search = NULL

# Initialisation du vecteur type_base avec une chaîne de caractères vide
type_base <- rep("", length(index_list$alias))

# Création d'une nouvelle colonne (type_base) au dataframe index_list
index_list <- data.frame(index_list, type_base=type_base, stringsAsFactors = FALSE)

# Dans la colonne type_base, pour les clients (dans la colonne alias) commençant par
# client- mettre sharing-data et pour ceux par feedback-bime- mettre feedback-bime
index_list$type_base[grepl(pattern = "client-",x = index_list$alias)] = "sharing-data"
index_list$type_base[grepl(pattern = "feedback-bime-",x = index_list$alias)] = "feedback-
bime"

# Dans la colonne type_base nouvellement crée, pour les clients (dans la colonne alias)
# commençant par feedback-preprod- et reviews- mettre autre
index_list$type_base[grepl(pattern = "feedback-preprod-",x = index_list$alias)] = "autre"
index_list$type_base[grepl(pattern = "reviews-",x = index_list$alias)] = "autre"

# Enlever les lignes autre dans le dataframe index_list
index_list <- index_list[!index_list$type_base=="autre", ]

# Ne garder que les clients dans la base sharing data
index_list <- index_list[index_list$type_base=="sharing data",]
```

```
# Renommer la colonne alias client
colnames(index_list)[colnames(index_list)=="alias"] <- "client"

# Renommer index_list par df
df <- index_list
```

2. Fonction pour convertir les colonnes de type "list" en type "vector"

```{r}

# Création de la fonction list_to_vector
list_to_vector <- function(df) {
  # Fonction list_to_vector qui convertit une colonne de type "list" en colonne de type
  # "vecteur"
  # @param df: le dataframe sur lequel on veut convertir les colonnes de type "list" en
  # "vecteur"
  # @return le daframe dont les colonnes de type "list" ont été converties en "vecteur"

  # On récupère toutes les classes des colonnes du dataframe
classe_colonnes <- sapply(df,  FUN=class)

  # On boucle sur chaque colonne du dataframe
  for (i in 1:length(classe_colonnes) {

    # Si la colonne en cours est de type "list", on convertit les NULL en NA
    # et on convertit la colonne en vecteur
    if(classe_colonnes[i] == "list") {
      df[df[, i] == "NULL", i] = NA
      df[, i] = unlist(df[, i])
    }
  }

  return(df)
} # //FIN fonction list to-vector()
```

3. Fonction appelant le client via son index

```{r}

# Créer une fonction appelant le client via son index
connect_client <- function(index_call){

  client <- es_search(
    es_host = 'http://…'
    , es_index = index_call
```

```
    , size = 5000
    , query_body = qbodySD
    , n_cores = 3
  )
  return(client)

# Boucle appelant la fonction connect_client pour récupérer données des clients
for (i in 1:length(df$client)) {
      v_client <- assign(gsub(pattern = "-",replacement = "_", df$client[i]),
                         connect_client(df$client[i]))
}
```

4. Chargement des données pour le tableau de correspondance

```{r}

# Créer la fonction get_SQL qui récupère les données de la base SQL pour alimenter la
# variable map
get_SQL <- function(connexion_à_bdd, requête) {
  # <code de la fonction ici>
  # A faire... je n'avais pas les identifiants pour me connecter à la base SQL
  # renvoie le dataframe
  return(df)
}

# Appel la fonction "get_SQL" pour récupérer les données SQL dans la variable map
requete <- "la requete SELECT à  mettre ici"
map <- get_SQL(connexion_a_bdd, requete)

map$line <- paste0("lineId", map$Ligne_Num)
map$nom_champ <- paste(map$question, map$line, sep=".")
map$nom_champ <- gsub("Q","q", x=map$nom_champ)
# nom de colonne avec un "q" minuscule dans Sharing-Data pour les colonnes
# "questionIdnn.lineIdn"
map$question=NULL
map$line=NULL

# Enlever les balises HTML de la colonne "Colonne_EnTete" (<p> ou </p>)
map$Colonne_EnTete <- gsub("<p>|</p>", "", x=map$Colonne_EnTete)
```

5. Fonction qui recode pour chaque questionnaire d'un client

```{r}

  # Créer la fonction "recode_data"
```

```
recode_data <- function(df_SD, questionnaire_id, map_csv) {
# de la colonne "Colonne_EnTete" du tableau de correspondance
# @param df_SD : le dataframe Sharing-Data
# @param questionnaire_id : l'identifiant du questionnaire identique au tableau de
# correspondance
# @param : map_csv : contenu du fichier CSV dans un dataframe

# on enregistre le df_SD pour le questionnaire passé en paramètre de la fonction
df_SD_questionnaire <- df_SD[df_SD$questionnaireId==questionnaire_id,]

# Création du vecteur contenant toutes les questions uniques du tableau de correspondance
# passé en paramètre de la function
questId_lineId_uniq <- c(unique(map_csv$nom_champ))

# Récupère l'indice de chaque question pour le questionnaire passé en paramètre de la
# function dans le dataframe Sharing-Data
indices <- match(questId_lineId_uniq, colnames(df_SD))
# On ne garde que les indices qui ont été trouvés
indices <- indices[!is.na(indices)] # on enlève les NA

# Boucle sur les lignes de df_SD
for (lig in 1:length(df_SD_questionnaire$questionnaireId)) {

  # Boucle sur les colonnes de Sharing-Data en parcourant toutes les questions du
  # questionnaire
  for (col in 1:length(indices)) {

    # On ne cherche la correspondance dans le tableau de correspondance que si la donnée
    # du dataframe Sharing-Data n'est pas NULL (sans char et avec)
    if(!is.null(df_SD_questionnaire[lig, indices[col]]) & df_SD_questionnaire[lig,
indices[col]] != "NULL") {

      # On regarde si l'identifiant du questionnaire de Sharing-Data a une correspondance
      # dans le tableau adequate
      if(df_SD_questionnaire[lig, indices[col]] %in% map_csv$Reponse_ID) {

        # On récupère l'indice correspondant dans la colonne Reponse_ID du tableau de
        # correspondence
        index_reponse_id <- match(df_SD_questionnaire[lig, indices[col]],
map_csv$Reponse_ID)
        # On remplace la donnée dans Sharing-Data par la donnée de Colonne_EnTete du
        # tableau de correspondence
        df_SD_questionnaire[lig, indices[col]] =
map_csv$Colonne_EnTete[index_reponse_id]
      }
    }
  }
}
```

```
return(df_SD_questionnaire)
} # //FIN de recode_data()
```

6. Boucle qui recode pour les clients dans son ensemble

```{r}

# Boucle sur les clients
for (i in 1:length(df$client)) {

  # On boucle sur les questionnaires du client concerné
  df_client <- get(gsub(pattern = "-", replacement = "_", df$client[i]))
  quest_id_uniq <- unique(df_client$questionnaireId)
  for(j in 1:length(quest_id_uniq)) {
    # Pour chaque questionnaire du client concerné, on appelle la fonction "recode_data"
    # pour recoder les colonnes "questionIdnn.lineIdn" avec les données correspondantes de
    # la colonne "Colonne_EnTete"
    df_client[df_client$questionnaireId==quest_id_uniq[j],] <- recode_data(df_client, quest_id_uniq[j], map[map$questionnaire_ID==quest_id_uniq[j],])
  }
}
```

J'ai tout d'abord notamment sélectionné les clients uniquement dans la base Sharing-Data. Comme il m'a été demandé également j'ai créé une fonction "list_to_vector", le but étant que dès que la boucle rencontre une colonne de type "list", elle la convertit en vecteur. Après préparation des données, j'ai créé successivement plusieurs fonctions pour remplacer le code dans les colonnes sous la forme questionIdn°.lineIdn° par son équivalence dans le tableau de correspondance. Avec Elastic Search, j'ai eu en premier lieu à créer une fonction "connect_client" permettant d'appeler le client via son index. Puis pour le tableau de correspondance il fallait récupérer les données sur la base SQL (procédure à finir). Une fois les données chargées, j'ai créé une fonction permettant de remplacer le nombre concerné pour chaque questionnaire d'un client. L'index (ou la position) des colonnes étant différent selon les questionnaires, cette fonction recherche automatiquement les index des colonnes voulues pour y apporter les modifications nécessaires. Puis j'ai créé une boucle qui prend en compte les clients dans leur ensemble pour que toute la base Sharing-Data soit nouvellement codifiée.