session_7

Julian Maitra

Session 7

Sujet : Manipuler les données (partie 1)

Le premier but de cette session est d'apprendre à importer des données dans RStudio à partir de **fichier CSV** et de les stocker en forme de data frame (format tabulaire).

Le deuxième but est de se familiariser avec `dplyr` (prononcé "dee-ply-er" en anglais), un paquet du `tidyverse` conçu pour manipuler les données en forme de data frame.

Contrairement aux jeux de données intégrés à R (tels que `diamonds`, etc.), la plupart des jeux de données du monde réel nécessitent des **transformations** avant de pouvoir être analysés et visualisés. À cette fin, nous examinerons également certaines transformations de types de variables et un nouvel opérateur, le `%>%` (angl. "pipe"; trad. franç. "tuyau"), qui est très utile pour les manipulations de données.

Lire des données dans RStudio

Par la suite, nous alons appliquer ces verbes avec un jeu de données concret, un fichier CSV nommé `igposts.csv`, qui contient des posts Instagram de célébritées qui ont reçu beaucoup d'interactions d'utilisateurs. Pour y arriver, il faut charger ce jeu de données dans notre session RStudio en cours.

> **i** À noter
>
> Un fichier de type **CSV** (angl. comma-separated values) est un format textuel simple pour sauvegarder des données en forme de tableau, c'est à dire avec des valeurs organisé par lignes et colonnes et séparées par virgules.
> Le format CSV présente l'avantage d'être assez léger et compatible avec tous les systèmes d'exploitation. En outre, il est utile pour stocker des données quantitatives en sciences sociales, qui se présentent souvent sous forme de tableaux.

Voici un exemple de format CSV simple :
nom, prénom, âge
Leclerc, Antoine, 53
Gomez, Hugo, 25
Nguyen, Laetitia, 19

Que nous pouvons afficher comme tableau (à noter que la première ligne désigne les noms des colonnes) :

nom	prénom	âge
Leclerc	Antoine	53
Gomez	Hugo	25
Nguyen	Laetitia	19

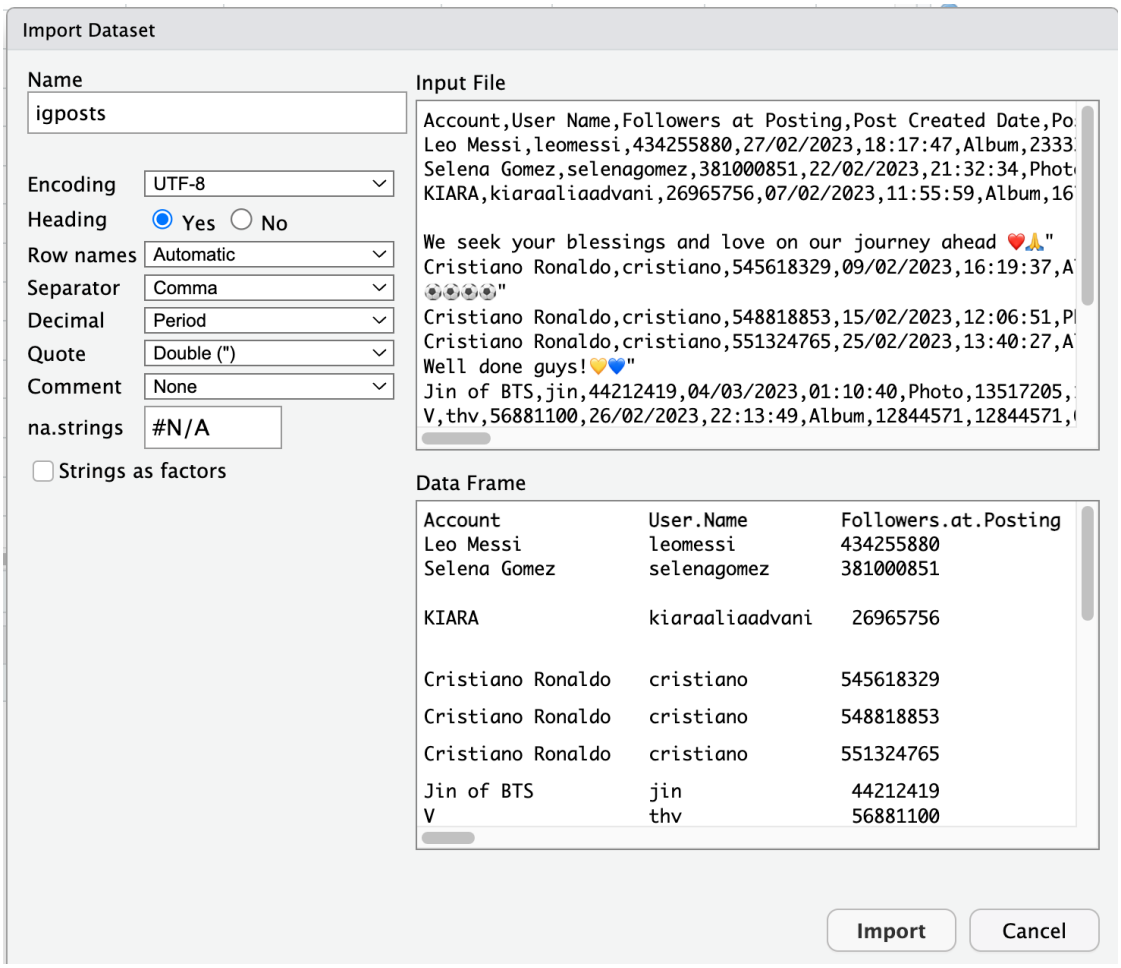
Avant l'importation de fichier CSV dans RStudio (et comme déjà vu pleins de fois dans le cours), il faut d'abord charger les paquets nécessaire avec `library()` et définir son répértoire de travail (vous devez adapter les paramètres de `setwd()` pour qu'il marche sur votre ordinateur) :

```
library(tidyverse)

setwd("/Users/domus_julian/Documents/GitHub/intro-a-R/code")
```

Pour charger le jeu de données `igposts.csv` dans RStudio, vous pouvez suivre le processus suivant :

1. Vous téléchargez le fichier `igposts.csv` depuis le site Moodle du cours (voir matériel de la 7e session).
2. Stockez le fichier `igposts.csv` dans le dossier que vous avez définit comme répértoire de travail.
3. Ensuite, vous utilisez le menu intéractif dans RStudio : File -> Import Dataset -> From Text (base)... et choisissez `igposts.csv`.
4. Une fenêtre avec des paramètre apparait, que vous pouvez remplir de la manière suivante :



- Explications des paramètres :
  - **Encoding = UTF-8** : Indique à RStudio d'utiliser l'encodage de caractères UTF-8 (qui est un standard très répandu), ce qui est important pour lire correctement toutes les lettres (notamment les caractères avec des accents : é, ä, ê, etc.).
  - **Heading = YES** : Indique à RStudio que la première ligne du fichier CSV contient les noms des colonnes.
  - **Separator = Comma** : Indique à RStudio que les valeurs du tableau sont séparées par des virgules (attention, certains documents francophones utilisent le point-virgule (;) à la place de la virgule).
  - **Decimal = Period** : Indique à RStudio que le point (.) est utilisé pour les décimales (et non la virgule comme souvent dans les documents en français).

- **na.strings = #N/A** : Indique à RStudio que les cellules vides dans le tableau sont désignées par #N/A (cela peut être différent dans d'autres fichiers CSV).

5. Finalement, vous pouvez copier et coller le code qui apparaît dans la console dans votre script R pour facilement recharger le fichier CSV ultérieurement.

```
# Par exemple, pour l'ordinateur de l'enseignant, ce code est le suivant :

igposts <- read.csv("/Users/domus_julian/Documents/GitHub/intro-a-R/code/igposts.csv", encodi

# Si tout a bien marché, l'objet igposts devrait apparaître dans votre environnement
# Nous pouvons afficher la structure de igposts pour une première impression :

str(igposts)
```

```
'data.frame':   100 obs. of  12 variables:
 $ Account             : chr  "Leo Messi" "Selena Gomez" "KIARA" "Cristiano Ronaldo" ...
 $ User.Name           : chr  "leomessi" "selenagomez" "kiaraaliaadvani" "cristiano" ...
 $ Followers.at.Posting: int  434255880 381000851 26965756 545618329 548818853 551324765 442
 $ Post.Created.Date   : chr  "27/02/2023" "22/02/2023" "07/02/2023" "09/02/2023" ...
 $ Post.Created.Time   : chr  "18:17:47" "21:32:34" "11:55:59" "16:19:37" ...
 $ Type                : chr  "Album" "Photo" "Album" "Album" ...
 $ Total.Interactions  : int  23333227 20020533 16776082 16501407 14683868 14619124 13517205
 $ Likes               : int  23075849 19760862 16554530 16317762 14603158 14489506 13517204
 $ Comments            : int  257378 259671 221552 183645 80710 129618 1 0 72141 80043 ...
 $ Views               : int  0 0 0 0 0 0 0 10998354 5648190 0 ...
 $ URL                 : chr  "https://www.instagram.com/p/CpLxnFlNJZn/" "https://www.instagr
 $ Description         : chr  "Gracias a todos los que hicieron posible que ganara este premi
```

Cela nous montre qu'il s'agit d'un data frame avec 100 observations et 12 variables. Chaque ligne représente un post Instagram d'une célébrité (unité statistique) et chaque colonnes une variable différente associé à ce post.

Notez que les variables sont soit de type textuel (chr = "character"; p. ex. `Account`= le nom du compte Insta qui a publié le post), soit numérique (int = "integer"; p. ex. `Total.Interactions` = la somme des `Likes`et `Comments` qu'un post a reçu).

Vous pouvez aussi cliquer sur l'objet et explorer le tableau manuellement pour le comprendre d'avantage (comme nous l'avons fait avec `mtcars`, etc.).

Attention

- Vous ne devez pas ouvrir les fichiers CSV avec des autres logiciels avant de les importer dans RStudio (par ex. avec Excel), car cela a souvent pour conséquence de modifier automatiquement le format du fichier CSV, ce qui entraîne ensuite des erreurs.
- Vous devriez enregistrer le fichier CSV directement dans votre répertoire de travail sans l'ouvrir, puis l'importer dans RStudio à l'aide du menu interactif.
- Vous devriez également éviter d'ouvrir le fichier CSV avec des applications ou autres (p. ex. application Moodle), ce qui peut également entraîner des erreurs.
- Il est préférable de télécharger les fichiers CSV avec des navigateurs standard tels que Chrome ou Safari.

Avant de continuer avec `dplyr`, regardons encore deux éléments cruciales pour la manipulation de données avec R :

- la transformation du types de variable (p. ex. de variables textuelles en facteurs)
- L'opérateur `%>%` (angl. "pipe"; trad. française "tuyau")

La transformation du type de variable

Lorsque l'on analyse des jeux de données, on constate parfois que toutes les variables **ne sont pas enregistrées dans un format utile pour l'analyse**. Parfois, par exemple, les valeurs numériques sont enregistrées sous forme de texte. Il est alors utile de les transformer dans un format numérique (par exemple avec la fonction `as.numeric()`.

Mais il y a aussi le cas suivant : les variables sous forme de texte ne représentent pas simplement des textes, mais certaines **catégories**. Dans ce cas, il est judicieux de les transformer en une variable catégorielle, c'est-à-dire un facteur, à l'aide de la fonction `as.factor()`.

Dans notre exemple de jeu de données `igposts`, qui contient des métriques sur les posts Instagram, nous pouvons justement constater cela. Certes, certaines variables textuelles, telles que `URL` (le lien vers le post) et `Description` (le message textuel associé au post Instagram), sont tout à fait pertinentes. D'autres, en revanche, sont des catégories. La plus évidente est la variable `Type`, qui indique si un post Instagram est une photo, une vidéo ou un album. Factorisons ci-dessous quelques-unes de ces variables catégorielles à l'aide de code :

```
# Transformons les colonnes "Account", "User.Name" et "Type" en facteurs :

igposts$Account <- as.factor(igposts$Account)
```

```r
igposts$User.Name <- as.factor(igposts$User.Name)

igposts$Type <- as.factor(igposts$Type)

# Vérifier, si ça a marché :

str(igposts)
```

```
'data.frame':   100 obs. of  12 variables:
 $ Account             : Factor w/ 38 levels "Anastasia Karanikolaou",..: 24 31 21 7 7 7 16 3
 $ User.Name           : Factor w/ 38 levels "agustd","arianagrande",..: 24 30 21 7 7 7 16 35
 $ Followers.at.Posting: int  434255880 381000851 26965756 545618329 548818853 551324765 442
 $ Post.Created.Date   : chr  "27/02/2023" "22/02/2023" "07/02/2023" "09/02/2023" ...
 $ Post.Created.Time   : chr  "18:17:47" "21:32:34" "11:55:59" "16:19:37" ...
 $ Type                : Factor w/ 2 levels "Album","Photo": 1 2 1 1 2 1 2 1 1 2 ...
 $ Total.Interactions  : int  23333227 20020533 16776082 16501407 14683868 14619124 13517205
 $ Likes               : int  23075849 19760862 16554530 16317762 14603158 14489506 13517204
 $ Comments            : int  257378 259671 221552 183645 80710 129618 1 0 72141 80043 ...
 $ Views               : int  0 0 0 0 0 0 0 10998354 5648190 0 ...
 $ URL                 : chr  "https://www.instagram.com/p/CpLxnFlNJZn/" "https://www.instagr
 $ Description         : chr  "Gracias a todos los que hicieron posible que ganara este premi
```

Nous allons voir par la suite, pourquoi ces factorisations sont utiles.

L'opérateur %>%

La fonction principale du pipe **%>%** : construire **une séquence de fonctions ("un tuyau")** qui est plus facile à comprendre et a lire dans le code.

```r
# Par exemple, regardez la fonction suivante : quel est le problème ?

log(sqrt(mean(c(1:100))))
```

```
[1] 1.960987
```

```r
# Le problème est que cette fonction est imbriquée et, à cause de cela, difficile à lire

# Avec le pipe, nous pouvons reécrire cette fonction comme ça et obtenir le même résultat ap

c(1:100) %>% mean() %>% sqrt() %>% log()
```

```
[1] 1.960987
```

> **Astuce**
>
> Vous pouvez créer le pipe `%>%` avec les raccourcis clavier suivants :
>
> - Sur **Windows** : Control + Shift + M
> - Sur **MacOS** : Command + Shift + M
>
> De plus, vous pouvez aussi utiliser `|>` comme opérateur pipe altérnatif (équivalent à `%>%`).

Vous pouvez déjà constater que travailler avec des jeux de données du monde réel, tels que le jeu de données Instagram `igposts`, nécessite un travail préparatoire substantiel. Mais rassurez-vous et ne vous découragez pas, plus vous pratiquez, plus cela devient facile et, bientôt, vous serez en mesure de charger facilement des ensembles de données dans RStudio ! Par ailleurs : il est normal de rencontrer des messages d'erreur. C'est une partie essentielle du processus d'apprentissage que de les surmonter en bricolant et en jouant avec vos données jusqu'à ce que vous puissiez les faire fonctionner.

Après avoir chargé notre fichier CSV dans RStudio, appliqué des transformations et notre connaissance de l'opérateur pipe, nous pouvons maintenant manipuler les données avec le package `dplyr`.

Manipuler les données avec `dplyr`

Le paquet `dplyr` est basé sur une logique de *verbes* pour les différents types de manipulation de données, notamment :

- Les verbes pour manipuler les données au niveau de **colonnes** :
  - `select()` : pour extraire une ou plusieurs colonnes d'un data frame.
  - `mutate()` : pour créer une ou plusieurs nouvelles colonnes dans un data frame.
- Les verbes pour manipuler les données au niveau de **lignes** :
  - `filter()` : pour filtrer les lignes d'un data frame selon un ou plusieurs critères.
  - `arrange()` : pour trier les lignes d'un data frame selon un ou plusieurs critères.
- Les verbes pour **résumer, compter et regrouper** les données :
  - `summarize()` : pour réduire une ou plusieurs colonnes d'un data frame en une seule ligne, notamment pour résumer des données.

- `count()` : pour compter le nombre d'observations d'une ou plusieurs catégories dans un data frame.
- `group_by()` : pour regrouper des données d'un data frame selon une ou plusieurs catégories.

Essayons maintenant d'appliquer tout cela avec du code et les données `igposts` pour comprendre ce que ça veut dire en pratique !

Manipulation de colonnes : `select()`et `mutate()`

```
# select() : fonction pour extraire une ou plusieurs colonnes d'un data frame

# Par exemple, nous pouvons extraire le quatres colonnes suivantes d'igposts (voir comment le

igposts %>% select(Account, Type, Followers.at.Posting, Total.Interactions)

# Avec les deux points dans la fonction select(), vous pouvez sélectionner plusieurs colonnes

igposts %>%
  select(Account:Views) %>%
  head()

# Vous pouvez également créer un nouvel objet, df1, qui est essentiellement un data frame plu

df1 <- igposts %>%
          select(Account,
                 Type,
                 Followers.at.Posting,
                 Total.Interactions,
                 URL)

str(df1)
```

```
# mutate() : fonction pour créer une ou plusieurs nouvelles colonnes dans un data frame

# Par exemple, nous pouvons créer une nouvelle colonne qui compte le nombre d'interactions pa

df1 %>% mutate(TI_par_Follower = Total.Interactions/Followers.at.Posting)

# Attention, sans opérateur <-, nous ne changeons pas l'objet df1
```

```r
# Pour ce faire et sauvegarder notre nouvelle colonne, nous pouvons l'écraser :

df1 <- df1 %>%
        mutate(TI_par_Follower = Total.Interactions/Followers.at.Posting)

str(df1)
```

Manipulation de lignes avec `filter()` et `arrange()`

```r
# filter() : fonction pour filtrer les lignes selon un ou plusieurs critères

# Pour filtrer pour une valeur spécifique d'une variable, p.ex. les posts de Cristiano Ronalo

df1 %>% filter(Account == "Cristiano Ronaldo")

# Pour plusieurs valeurs spécifique de la même variable, p. ex. les posts de Ronaldo et Messi

df1 %>% filter(Account %in% c("Cristiano Ronaldo", "Leo Messi"))

# Pour plusieurs valeurs spécifiques de variables différentes, p. ex. les posts de type Photo

df1 %>%
  filter(Account == "Cristiano Ronaldo",
         Type == "Photo")

# Filtrer à partir de valeurs numériques, p. ex. les posts avec au moins 10 millions d'intéra

df1 %>%
  filter(Total.Interactions >= 10000000)

# Filtrer pour les posts de Selena Gomez avec au moins 10 millions d'intéractions :

df1 %>%
  filter(Account == "Selena Gomez",
         Total.Interactions >= 10000000)
```

```r
# arrange() : fonction pour trier les lignes selon un ou plusieurs critères

# Trier des valeurs/facteurs textuelles par ordre alphabétique :
```

```r
df1 %>% arrange(Account) %>% head(10)

# Trier par le nombre d'intéractions (ordre croissant)

df1 %>%
  arrange(Total.Interactions) %>%
  head(10)

# Trier par le nombre d'intéractions par follower (ordre décroissant)

df1 %>%
  arrange(desc(TI_par_Follower)) %>%
  head(10)
```

Résumer, compter et regrouper avec `summarize()`, `count()` et `group_by()`

```r
# summarize() : fonction qui réduit une ou plusieurs colonnes en une seule ligne,
# notamment pour résumer des données

# Nous pouvons, par exemple, résumer la somme totale de Total.Interactions dans notre ensembl

df1 %>% summarize(sum_TI = sum(Total.Interactions))

# Nous pouvons aussi calculer les moyennes des intéractions et de followers de tout les compt

df1 %>%
  summarize(mean_TI = mean(Total.Interactions),
                  mean_Followers = mean(Followers.at.Posting))
```

```r
# count() : fonction qui compte le nombre d'observations d'une ou plusieurs catégories

# Compter l'occurence des types de posts Instagram (Photo et Album, dans ce cas) dans notre j

df1 %>%
  count(Type)

# Compter l'occurence des différents comptes dans notre jeu de données :

df1 %>%
  count(Account)
```

```r
# Compter l'occurence des différents comptes dans notre jeu de données en les triant par cett

df1 %>%
  count(Account) %>%
  arrange(-n) %>%
  head(10)
```

```r
# group_by() : fonction qui regroupe des données selon une ou plusieurs catégories

# Regroupement par type de post et comparaison du nombre moyen d'interactions par type de pos

df1 %>% group_by(Type) %>% summarize(mean_TI = mean(Total.Interactions))

# En utilisant le paramètre n=n() dans la fonction summarize(), nous pouvons également compte

df1 %>%
  group_by(Type) %>%
  summarize(n = n(),
            mean_TI = mean(Total.Interactions))

# Utilisons plusieurs verbes dplyr pour trouver les 10 comptes qui apparaissent le plus fréqu

df1 %>%
  group_by(Account) %>%
  summarize(n = n(),
            mean_Followers = mean(Followers.at.Posting),
            mean_TI = mean(Total.Interactions)) %>%
  arrange(desc(n)) %>%
  head(10)

# Utilisons plusieurs verbes dplyr pour comparer les Likes et Comments des posts de Ronaldo e

igposts %>%
  group_by(Account) %>%
  filter(Account %in% c("Cristiano Ronaldo", "Leo Messi")) %>%
  summarize(n = n(), mean_Followers = mean(Followers.at.Posting),
            mean_Likes = mean(Likes),
            mean_Comments = mean(Comments))
```

Devoir pour Session 8 et infos questionnaire/Problem Set 3

- Attention, **la session 8 du 17 et 18 avril 2024 sera une session en auto-apprentissage !** Il n'y a donc pas d'enseignement en classe à Fribourg.
- Le matériel de session 8, la deuxième partie de la manipulation de données, sera accessible sur ce site à partir du 17 avril.
- Assurez-vous d'avoir **rempli le questionnaire avant la date limite** (**19 avril 2024 à 23h59**). Tou(te)s les étudiant(e)s qui auront rempli le questionnaire à temps recevront 2 points de bonus pour le problème set 4 (10% des points).
- Si vous avez des difficultés à accéder au questionnaire via le lien personnalisé qui vous a été envoyé par e-mail : Essayez à nouveau avec différents navigateurs, sur votre téléphone portable et désactivez les éventuels ad-blockers. Si nécessaire, veuillez contacter l'enseignant afin de recevoir un nouveau lien par e-mail.
- Dans deux semaines, les 24 et 25 avril, aura lieu **Problem Set 3**. Il portera principalement sur les contenus des sessions 7 et 8 (manipulation de données). Comblez vous-même les éventuelles lacunes de vos connaissances (p. ex. configurer correctement le répertoire de travail, utiliser l'opérateur d'assignation `<-`, etc.) Assurez-vous de pouvoir lire correctement les fichiers CSV dans RStudio, car cela sera nécessaire pour résoudre l'ensemble de problèmes 3.