Notes de cours | Pivot avancé

February 2024

Introduction

Vous connaissez les opérations de pivot de base des jeux de données du format long au format large et vice versa. Cependant, comme c'est souvent le cas, les manipulations de base ne sont pas suffisantes pour le traitement des données que vous devez faire. Voyons maintenant le niveau suivant. Allons-y !

Objectifs d'apprentissage

1. Maîtriser le pivot complexe du format large au format long et du format long au format large

2. Savoir utiliser les séparateurs comme outil de pivot

Packages

```
er les packages
uire(pacman)) install.packages("pacman")
p_load(tidyverse, outbreaks, janitor, rio, here, knitr)
```

Jeux de données

Nous présenterons ces jeux de données au fur et à mesure, mais voici un aperçu :

- Données d'enquête d'une étude menée en Inde sur les dépenses des patients pour le traitement de la tuberculose
- Données d'une étude sur les biomarqueurs des entéropathogènes en Zambie
- Une enquête alimentaire au Vietnam

Du format large au format long

Parfois, vous avez plusieurs types de données au format large dans le même jeu de données. Considérez cet exemple factice de la taille et du poids des enfants sur deux ans :

```
stats <-
e::tribble(
fant, ~annee1_taille, ~annee2_taille, ~annee1_poids, ~annee2_poids,
"A",          "80cm",          "85cm",          "5kg",          "10kg",
"B",          "85cm",          "90cm",          "7kg",          "12kg",
"C",          "90cm",         "100cm",          "6kg",          "14kg"


stats
```

```
## # A tibble: 3 × 5
##   enfant annee1_taille annee2_taille
##   <chr>  <chr>         <chr>
## 1 A      80cm          85cm
## 2 B      85cm          90cm
## 3 C      90cm          100cm
##   annee1_poids annee2_poids
##   <chr>        <chr>
## 1 5kg          10kg
## 2 7kg          12kg
## 3 6kg          14kg
```

Si vous pivotez toutes les colonnes des mesures, vous obtiendrez des données trop longues :

```
stats %>%
_longer(2:5)
```

```
## # A tibble: 5 × 3
##   enfant name          value
##   <chr>  <chr>         <chr>
## 1 A      annee1_taille 80cm
## 2 A      annee2_taille 85cm
## 3 A      annee1_poids  5kg
```

```
## 4 A       annee2_poids  10kg
## 5 B       annee1_taille 85cm
```

Ce n'est (généralement) pas ce que nous recherchons, car maintenant vous avez deux données différentes dans la même colonne- le poids et la taille.

Pour obtenir le bon format, vous devez utiliser l'argument `names_sep` et l'identifiant ".value" :

```
stats %>%
_longer(2:5,
        names_sep = "_",
        names_to = c("periode", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>  <chr>   <chr>  <chr>
## 1 A      annee1  80cm   5kg
## 2 A      annee2  85cm   10kg
## 3 B      annee1  85cm   7kg
## 4 B      annee2  90cm   12kg
## 5 C      annee1  90cm   6kg
```

Maintenant, nous avons une ligne pour chaque combinaison enfant-période, un format long correct !

Ce que fait le code ci-dessus peut ne pas être clair, mais vous devriez déjà pouvoir répondre à l'exercice ci-dessous en reproduisant la syntaxe de l'exemple précédant. Après cet exercice , nous expliquerons l'argument `names_sep` et l'identifiant ".value" plus en détail.

Considérez cet autre ensemble de données factice :

```
stats <-
e::tribble(
ılte,  ~annee1_IMC,  ~annee2_IMC,  ~annee1_VIH,  ~annee2_VIH,
'A",          25,          30,  "Positive",  "Positive",
'B",          34,          28,  "Negative",  "Positive",
'C",          19,          17,  "Negative",  "Negative"



stats
```

```
## # A tibble: 3 × 5
##   adulte annee1_IMC annee2_IMC annee1_VIH
```

```
## 2 B              34         28 Negative
## 3 C              19         17 Negative
##   annee2_VIH
##   <chr>
## 1 Positive
## 2 Positive
## 3 Negative
```

Pivotez les données en un format long pour obtenir la structure suivante :

adulte	annee	IMC	VIH

```
e_long <-
e_stats %>%
_longer(_________)
```

```
iez votre réponse
Q_adulte_long()
```

PRACTICE (in RMD)

```
## Vous n'avez pas encore défini l'objet de réponse,
`Q_adulte_long`.
##  ▷1◁  2   3   4   5   6
```

```
_adulte_long()
```

```
##
## INDICE.
##
##   Utilisez la fonction `pivot_longer` du package
`tidyverse`.
##   - Spécifiez `cols = 2:5` pour sélectionner les colonnes à
pivoter.
##   - Utilisez `names_sep = "_"` et `names_to = c("year",
".value")` pour formater les noms.
##
```

```
obtenir la solution, exécutez la ligne ci-dessous !
ON_Q_adulte_long()
```

```
##
## SOLUTION
##
##   adulte_stats %>%
##       pivot_longer(cols = 2:5,
```

```
##                     names_sep = "_",
##                     names_to = c("annee", ".value"))
```

PRACTICE

(in RMD)

```
e question a une fonction solution similaire à celle-ci.
INT est remplacé par SOLUTION dans le nom de la fonction.)
vous devrez taper le nom de la fonction par vous-même.
 vise à vous dissuader de regarder la solution avant de répondre
 à la question.)
```

L'exemple ci-dessus `enfant_stats` a des nombres stockés en tant que caractères [...]

Comme vous l'avez vu dans la leçon précédente, vous pouvez facilement extraire les nombres à partir du jeux de données de sortie au format long en utilisant la fonction `parse_number()` de readr :

```
stats_long <-
_stats %>%
_longer(2:5,
        names_sep = "_",
        names_to = c("periode", ".value"))

stats_long
```

SIDE NOTE

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>  <chr>   <chr>  <chr>
## 1 A      annee1  80cm   5kg
## 2 A      annee2  85cm   10kg
## 3 B      annee1  85cm   7kg
## 4 B      annee2  90cm   12kg
## 5 C      annee1  90cm   6kg
```

```
stats_long %>%
e(taille = parse_number(taille),
  poids = parse_number(poids))
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>  <chr>    <dbl> <dbl>
## 1 A      annee1      80     5
## 2 A      annee2      85    10
## 3 B      annee1      85     7
```

SIDE NOTE

```
## 4 B       annee2       90     12
## 5 C       annee1       90      6
```

Comprendre `names_sep` et ".value"

Maintenant, décomposons l'appel `pivot_longer()` que nous avons vu ci-dessus :

```
stats
```

```
## # A tibble: 3 × 5
##   enfant annee1_taille annee2_taille
##   <chr>  <chr>         <chr>
## 1 A      80cm          85cm
## 2 B      85cm          90cm
## 3 C      90cm          100cm
##   annee1_poids annee2_poids
##   <chr>        <chr>
## 1 5kg          10kg
## 2 7kg          12kg
## 3 6kg          14kg
```

```
stats %>%
_longer(2:5,
        names_sep = "_",
        names_to = c("periode", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>  <chr>   <chr>  <chr>
## 1 A      annee1  80cm   5kg
## 2 A      annee2  85cm   10kg
## 3 B      annee1  85cm   7kg
## 4 B      annee2  90cm   12kg
## 5 C      annee1  90cm   6kg
```

Remarquez que les noms de colonnes dans le dataframe `enfant_stats` d'origine (`annee1_taille`, `annee2_taille` etc.) sont composés de trois parties :

- la période référencée : par exemple "annee1"
- un séparateur de soulignement, "_";
- et le type de valeur enregistrée "taille" ou "poids"

Nous pouvons faire un tableau avec ces parties :

nom_colonne	periode	separateur	".value"
annee1_taille	annee1	_	taille
annee2_taille	annee2	_	taille
annee1_poids	annee1	_	poids
annee2_poids	annee2	_	poids

Sur la base de ce tableau, il devrait maintenant être plus facile de comprendre les arguments `names_sep` et `names_to` que nous avons fournis à `pivot_longer()` :

`names_sep = "_"` :

C'est le séparateur entre l'indicateur de période (année) et les valeurs (taille et poids) enregistrées.

Si nous utilisons un séparateur différent, l'argument va aussi changer. Par exemple, si le séparateur est un espace vide, " ", vous aurez `names_sep = " "`, comme on le voit dans l'exemple ci-dessous :

```
stats_espace_sep <-
e::tribble(
fant, ~`ann1 taille`, ~`ann2 taille`, ~`ann1 poids`, ~`ann2 poids`,
"A",         "80cm",         "85cm",          "5kg",         "10kg",
"B",         "85cm",         "90cm",          "7kg",         "12kg",
"C",         "90cm",        "100cm",          "6kg",         "14kg"


stats_espace_sep %>%
_longer(2:5,
        names_sep = " ",
        names_to = c("periode", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>  <chr>   <chr>  <chr>
## 1 A      ann1    80cm   5kg
## 2 A      ann2    85cm   10kg
## 3 B      ann1    85cm   7kg
## 4 B      ann2    90cm   12kg
## 5 C      ann1    90cm   6kg
```

`names_to = c("periode", ".value")`

Ensuite, l'argument `names_to` indique comment les données doivent être restructurées. Nous avons passé un vecteur de deux chaînes de caractères, "periode" et ".value" à cet argument. Voyons le rôle de chaque élément :

La chaîne "periode" indique que nous voulons placer les données de chaque année (ou période) dans une ligne séparée. Notez qu'il n'y a rien de spécial dans le

mot “periode” utilisé ici ; nous pourrions changer cela par n’importe quelle autre chaîne. Donc, au lieu de “periode”, vous auriez pu écrire “temps” ou “annee_de_mesure” ou autre chose :

```
stats %>%
_longer(2:5,
        names_sep = "_",
        names_to = c("annee_de_mesure", ".value"))
```

```
## # A tibble: 5 × 4
##   enfant annee_de_mesure taille poids
##   <chr>  <chr>           <chr>  <chr>
## 1 A      annee1          80cm   5kg
## 2 A      annee2          85cm   10kg
## 3 B      annee1          85cm   7kg
## 4 B      annee2          90cm   12kg
## 5 C      annee1          90cm   6kg
```

Maintenant, **le placeholder “.value”** est un indicateur spécial, qui indique à `pivot_longer()` de créer une colonne séparée pour chaque valeur distincte qui apparaît après le séparateur. Dans notre exemple, ces valeurs sont “taille” et “poids”.

La chaîne “.value” ne peut pas être remplacée arbitrairement. Par exemple, ceci ne fonctionnera pas :

```
stats %>%
_longer(2:5,
        names_sep = "_",
        names_to = c("periode", "valeurs"))
```

```
## # A tibble: 5 × 4
##   enfant periode valeurs value
##   <chr>  <chr>   <chr>   <chr>
## 1 A      annee1  taille  80cm
## 2 A      annee2  taille  85cm
## 3 A      annee1  poids   5kg
## 4 A      annee2  poids   10kg
## 5 B      annee1  taille  85cm
```

Autrement dit, le placeholder “.value” indique à `pivot_longer()` que nous voulons séparer les valeurs “taille” et “poids” dans deux colonnes séparées, car nous avons deux types de valeurs après le séparateur “_” dans les noms de colonnes.

Cela signifie que si vous avez un jeu de données au format large avec trois types de valeurs, vous obtiendrez trois colonnes séparées, une pour chaque type de valeur. Par exemple, considérez le jeu de données fictif ci-dessous qui montre les enregistrements d’enfants, à deux moments, pour les variables suivantes :

- âge en mois,
- pourcentage de graisse corporelle
- IMC

```
stats_trois_valeurs <-
e::tribble(
nt,   ~t1_age,   ~t2_age, ~t1_graisse, ~t2_graisse, ~t1_imc, ~t2_imc,
',   "5 mois",   "8 mois",       "13%",       "15%",      14,      15,
',   "7 mois",   "9 mois",       "15%",       "17%",      16,      18

stats_trois_valeurs
```

```
## # A tibble: 2 × 7
##   enfant t1_age t2_age t1_graisse t2_graisse
##   <chr>  <chr>  <chr>  <chr>      <chr>
## 1 a      5 mois 8 mois 13%        15%
## 2 b      7 mois 9 mois 15%        17%
##   t1_imc t2_imc
##    <dbl>  <dbl>
## 1     14     15
## 2     16     18
```

Ici, dans les noms de colonnes, il y a trois types de valeurs qui apparaissent après le séparateur “_” : `age`, `graisse` et `imc`; la chaîne “.value” indique à `pivot_longer()` de créer une nouvelle colonne pour chaque type de valeur :

```
stats_trois_valeurs %>%
_longer(2:7,
        names_sep = "_",
        names_to = c("temps", ".value")
        )
```

```
## # A tibble: 4 × 5
##   enfant temps age    graisse   imc
##   <chr>  <chr> <chr>  <chr>   <dbl>
## 1 a      t1    5 mois 13%        14
## 2 a      t2    8 mois 15%        15
## 3 b      t1    7 mois 15%        16
## 4 b      t2    9 mois 17%        18
```

Un pédiatre enregistre les informations suivantes pour un ensemble d'enfants sur deux ans :

- périmètre cranien ;
- circonférence du cou ; et
- tour de hanches

le tout en centimètres.

Voici le tableau de sortie :


```
ıce_stats <-
e::tribble(
fant,~ann1_tete,~ann2_tete,~ann1_cou,~ann2_cou,~ann1_hanche,~ann2_
'a",          45,         48,        23,        24,           51,
 52,
'b",          48,         50,        24,        26,           52,
 52,
'c",          50,         52,        24,        27,           53,
 54


ıce_stats
```

```
## # A tibble: 3 × 7
##   enfant ann1_tete ann2_tete ann1_cou ann2_cou
##   <chr>      <dbl>     <dbl>    <dbl>    <dbl>
## 1 a             45        48       23       24
## 2 b             48        50       24       26
## 3 c             50        52       24       27
##   ann1_hanche ann2_hanche
##         <dbl>       <dbl>
## 1          51          52
## 2          52          52
## 3          53          54
```

Pivotez les données en un format long pour obtenir la structure suivante :

enfant	annee	tete	cou	hanche

```
ance_stats_long <-
ance_stats %>%
_longer(_________)
```

```
ez votre réponse
Q_croissance_stats_long()
```

```
## Vous n'avez pas encore défini l'objet de réponse,
`Q_croissance_stats_long`.
##   1  ▷2◁  3   4   5   6
```


```
_croissance_stats_long()
```

```
##
## INDICE.
##
##   Utilisez la fonction `pivot_longer` du package
`tidyverse`.
##   - Spécifiez `cols = 2:7` pour sélectionner les colonnes à
pivoter.
##   - Utilisez `names_to = c("year", ".value")` et `names_sep
= "_"` pour formater les noms.
##
```

Type de valeur *avant* le séparateur

Dans tous les exemples que nous avons utilisés jusqu'à présent, les noms de colonnes étaient construits de telle sorte que le type de valeur venait après le séparateur. Rappelez-vous notre tableau :

nom_colonne	periode	separateur	".value"
annee1_taille	annee1	_	taille
annee2_taille	annee2	_	taille
annee1_poids	annee1	_	poids
annee2_poids	annee2	_	poids

Mais bien sûr, les noms de colonnes pourraient être construits différemment, avec les types de valeurs venant avant le séparateur, comme dans cet exemple :

```
stats2 <-
e::tribble(
fant, ~taille_annee1, ~taille_annee2, ~poids_annee1, ~poids_annee2,
"A",         "80cm",          "85cm",         "5kg",         "10kg",
"B",         "85cm",          "90cm",         "7kg",         "12kg",
"C",         "90cm",         "100cm",         "6kg",         "14kg"


stats2
```

```
## # A tibble: 3 × 5
##   enfant taille_annee1 taille_annee2
##   <chr>  <chr>         <chr>
## 1 A      80cm          85cm
## 2 B      85cm          90cm
## 3 C      90cm          100cm
##   poids_annee1 poids_annee2
##   <chr>        <chr>
## 1 5kg          10kg
## 2 7kg          12kg
## 3 6kg          14kg
```

Ici, les types de valeurs (taille et poids) viennent avant le "_" séparateur.

Comment notre commande `pivot_longer()` peut-elle s'adapter à cela ? C'est simple ! Il suffit d'inverser l'ordre du vecteur donné à l'argument `names_to` :

Donc, au lieu de `names_to = c("temps", ".value")`, vous aurez `names_to = c(".value", "temps")` :

```
stats2 %>%
_longer(2:5,
        names_sep = "_",
        names_to = c(".value", "temps"))
```

```
## # A tibble: 5 × 4
##   enfant temps  taille poids
##   <chr>  <chr>  <chr>  <chr>
## 1 A      annee1 80cm   5kg
## 2 A      annee2 85cm   10kg
## 3 B      annee1 85cm   7kg
## 4 B      annee2 90cm   12kg
## 5 C      annee1 90cm   6kg
```

Et voilà !

Considérez le jeu de données suivant de la Zambie sur les entéropathogènes et leurs biomarqueurs.

```
athogenes_zambie_large<-
 read_csv(here("data/fr_enteropathogenes_zambie_large.csv"))

athogenes_zambie_large
```

```
## # A tibble: 5 × 7
##      ID LPS_1 LPS_2  LBP_1 LBP_2 IFABP_1
```

```
##   <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl>
## 1  1002  222.  390. 38414. 6840.  1294.
## 2  1003  181.   NA  26888.   NA     22.5
## 3  1004  257.  221. 49183. 5426.     0
## 4  1005   NA   369.    NA  1938.     0
## 5  1006  275.   NA  61758.   NA      0
##   IFABP_2
##     <dbl>
## 1    610.
## 2     NA
## 3      0
## 4   1010.
## 5     NA
```

Ce jeu de données se compose des colonnes suivantes :

- LPS_1 et LPS_2 : niveau des lipopolysaccharides, mesuré par Pyrochrome LAL, en EU/mL
- LBP_1 et LBP_2 : niveau des protéines de liaison au LPS, en pg/mL
- IFABP_1 et IFAPB_2 : niveau des protéines de liaison aux acides gras de type intestinal, en pg/mL

Pivotez le jeu de données pour qu'il ressemble à la structure suivante :

ID	numero_echantillon	LPS	LBP	IFABP

```
›pathogenes_zambie_large <-
›pathogenes_zambie_large %>%
_longer(_____________)
```

```
ez votre réponse
)_enteropathogenes_zambie_large()
```

```
## Vous n'avez pas encore défini l'objet de réponse,
`Q_enteropathogenes_zambie_large`.
##   1   2  ▷3◁  4   5   6
```

```
_enteropathogenes_zambie_large()
```


```
##
## INDICE.
##
##   Utilisez la fonction `pivot_longer` du package
`tidyverse` pour réaliser la pivotisation.
##   - Spécifiez les arguments `names_to = c(".value",
"numero_echantillon")` et `names_sep = "_"`.
##
```

Un exemple qui n'est pas une série temporelle

Jusqu'à présent, nous avons utilisé des ensembles de données personne-période (séries temporelles) pour illustrer l'idée de pivots complexes avec plusieurs types de valeurs.

Mais comme nous l'avons mentionné, tous les jeux de données nécessitant une restructuration ne sont pas forcément des données de séries temporelles. Voyons un exemple rapide qui n'est pas une série temporelle.

Vous pourriez mesurer la taille (cm) et le poids (kg) d'une série de couples parentaux dans un tableau comme celui-ci :

```
_stats <-
e::tribble(
_e, ~pere_taille, ~pere_poids, ~mere_taille, ~mere_poids,
a",          180,          80,          160,          70,
b",          185,          90,          150,          76,
c",          182,          93,          143,          78

_stats
```

```
## # A tibble: 3 × 5
##   couple pere_taille pere_poids mere_taille
##   <chr>        <dbl>      <dbl>       <dbl>
## 1 a              180         80         160
## 2 b              185         90         150
## 3 c              182         93         143
##   mere_poids
##        <dbl>
## 1         70
## 2         76
## 3         78
```

Ici, nous avons deux types de valeurs différents (poids et taille) pour chaque personne du couple.

Pour pivoter à une ligne par personne, nous aurons encore besoin des arguments `names_sep` et `names_to` :

```
_stats %>%
_longer(2:5,
        names_sep  = "_",
        names_to = c("personne", ".value"))
```

```
## # A tibble: 5 × 4
##   couple personne taille poids
##   <chr>  <chr>     <dbl> <dbl>
## 1 a      pere        180    80
## 2 a      mere        160    70
## 3 b      pere        185    90
## 4 b      mere        150    76
## 5 c      pere        182    93
```

Le séparateur est un trait de soulignement, “_”, donc nous avons utilisé `names_sep = "_"` et comme les types de valeurs viennent après le séparateur, l’identifiant “.value” a été placé en deuxième dans l’argument `names_to`.

Echapper le séparateur de point

Un cas spécial que vous pourriez rencontrer est un ensemble de données où le séparateur est un point.

```
stats_point_sep <-
e::tribble(
ant, ~annee1.taille, ~annee2.taille, ~annee1.poids, ~annee2.poids,
'A",          "80cm",          "85cm",          "5kg",         "10kg",
'B",          "85cm",          "90cm",          "7kg",         "12kg",
'C",          "90cm",         "100cm",          "6kg",         "14kg"


stats_point_sep %>%
_longer(2:5,
        names_to = c("periode", ".value"),
        names_sep = "\\.")
```

```
## # A tibble: 5 × 4
##   enfant periode taille poids
##   <chr>  <chr>   <chr>  <chr>
## 1 A      annee1  80cm   5kg
## 2 A      annee2  85cm   10kg
## 3 B      annee1  85cm   7kg
## 4 B      annee2  90cm   12kg
## 5 C      annee1  90cm   6kg
```

Ici, nous avons utilisé la chaîne “\.” pour indiquer un point “.” parce que le “.” est un caractère spécial dans R qui dans certains cas doit être échappé.

Considérez à nouveau les données adulte_stats que vous avez vues ci-dessus. Maintenant, les noms des colonnes ont été légèrement modifiés.

```
stats_point_sep <-
e::tribble(
ulte,  ~`IMC.annee1`,  ~`IMC.annee2`,  ~`VIH.annee1`,
 ~`VIH.annee2`,
'A",            25,            30,    "Positive",   "Positive",
'B",            34,            28,    "Negative",   "Positive",
'C",            19,            17,    "Negative",   "Negative"



stats_point_sep
```

```
## # A tibble: 3 × 5
##   adulte IMC.annee1 IMC.annee2 VIH.annee1
##   <chr>       <dbl>      <dbl> <chr>
## 1 A              25         30 Positive
## 2 B              34         28 Negative
## 3 C              19         17 Negative
##   VIH.annee2
##   <chr>
## 1 Positive
## 2 Positive
## 3 Negative
```

PRACTICE (in RMD)

Encore une fois, pivotez les données en un format long pour obtenir la structure suivante :

adulte	annee	IMC	VIH

```
e2_long <-
e_stats_point_sep %>%
_longer(_________)
```

```
ez votre réponse
)_adulte2_long()
```

```
## Vous n'avez pas encore défini l'objet de réponse,
`Q_adulte2_long`.
```

```
##   1   2   3  ▷4◁  5   6
```

```
adulte2_long()
```


```
##
## INDICE.
##
##   Utilisez la fonction `pivot_longer` du package
`tidyverse`.
##   - Spécifiez `cols = 2:5` pour sélectionner les colonnes à
pivoter.
##   - N'oubliez pas d'échapper correctement le caractère
point en utilisant `names_sep = "\."`
##   et utilisez `names_to = c(".value", "year")` pour
formater les noms.
##
```

Que faire quand vous n'avez pas un séparateur net ?

Parfois, vous n'avez pas de séparateur net.

Considérez ces données d'une enquête menée en Inde qui examine les dépenses des patients pour le traitement de la tuberculose :

```
tes <- read_csv(here("data/fr_india_tb_pathways_and_costs_data.csv")) %>%
_names() %>%
t(id, premiere_visite_emplacement, premiere_visite_cout,
 deuxieme_visite_emplacement, deuxieme_visite_cout,
 troisieme_visite_emplacement, troisieme_visite_cout)

tes
```

```
## # A tibble: 5 × 7
##       id premiere_visite_e…¹ premiere_visite…²
##    <dbl> <chr>                           <dbl>
## 1 100202 GH                                  0
## 2 100396 Pvt. docto                       1500
## 3 100590 Pvt. docto                       2000
## 4 100687 Pvt. hospi                      20000
## 5 100784 Pvt. docto                       1000
##   deuxieme_visite_emplacem…³ deuxieme_visite…⁴
##   <chr>                                 <dbl>
## 1 <NA>                                      0
## 2 Pvt. clini                             1000
## 3 Pvt. docto                             3000
## 4 Pvt. hospi                             1500
## 5 GH                                        0
```

```
## # ℹ abbreviated names: ¹premiere_visite_emplacement,
## #   ²premiere_visite_cout, ³deuxieme_visite_emplacement, …
```

Il n'y a pas de séparateur net entre les indicateurs de temps (premier, deuxième, troisième) et le type de valeur (cout, emplacement). C'est-à-dire, au lieu de "premierevisite_emplacement", nous avons plutôt "premiere_visite_emplacement", donc le trait de soulignement est utilisé pour deux buts. Pour cette raison, si vous essayez notre stratégie de pivot habituelle, vous obtiendrez un message d'erreur :

```
tes %>%
_longer(2:7,
        names_to = c("numero_visite", ".value"),
        names_sep = "_")
```

```
Error in `pivot_longer()`:
! Can't combine `premiere_visite_emplacement` <character> and
`premiere_visite_cout` <double>.
Run `rlang::last_trace()` to see where the error occurred.
```

La façon la plus directe de restructurer ce jeu de données avec succès serait d'utiliser un "regex" spécial (manipulation de chaînes de caractères), mais il est probable que vous n'ayez pas encore appris cela !

Alors pour l'instant, la solution que nous recommandons est de renommer manuellement vos colonnes pour insérer un séparateur clair, "__" :

```
tes_renomme <-
sites %>%
e(premiere__visite_emplacement = premiere_visite_emplacement,
  premiere__visite_cout = premiere_visite_cout,
  deuxieme__visite_emplacement = deuxieme_visite_emplacement,
  deuxieme__visite_cout = deuxieme_visite_cout,
  troisieme__visite_emplacement = troisieme_visite_emplacement,
  troisieme__visite_cout = troisieme_visite_cout)

tes_renomme
```

```
## # A tibble: 5 × 7
##       id premiere__visite_…¹ premiere__visit…²
##    <dbl> <chr>                           <dbl>
## 1 100202 GH                                  0
## 2 100396 Pvt. docto                       1500
## 3 100590 Pvt. docto                       2000
## 4 100687 Pvt. hospi                      20000
## 5 100784 Pvt. docto                       1000
##   deuxieme__visite_emplace…³ deuxieme__visit…⁴
##   <chr>                                 <dbl>
## 1 <NA>                                      0
## 2 Pvt. clini                             1000
## 3 Pvt. docto                             3000
```

```
## 4 Pvt. hospi                                  1500
## 5 GH                                             0
## # ℹ abbreviated names: ¹premiere__visite_emplacement,
## #   ²premiere__visite_cout, …
```

Maintenant, nous pouvons essayer le pivot :

```
tes_long <-
sites_renomme %>%
_longer(2:7,
        names_to = c("numero_visite", ".value"),
        names_sep = "__")
tes_long
```

```
## # A tibble: 5 × 4
##       id numero_visite visite_emplacement
##    <dbl> <chr>         <chr>
## 1 100202 premiere      GH
## 2 100202 deuxieme      <NA>
## 3 100202 troisieme     <NA>
## 4 100396 premiere      Pvt. docto
## 5 100396 deuxieme      Pvt. clini
##   visite_cout
##         <dbl>
## 1           0
## 2           0
## 3           0
## 4        1500
## 5        1000
```

Maintenant, nettoyons le jeu de données :

```
tes_long %>%
primer les observations manquantes
r(!visite_emplacement == "") %>%
her un nom significatif aux valeurs de numero_visite
e(numero_visite = case_when(numero_visite == "premiere" ~ 1,
                            numero_visite == "deuxieme" ~ 2,
                            numero_visite == "troisieme" ~ 3)) %>%
ssurer que visite_cout est numérique
e(visite_cout = as.numeric(visite_cout))
```

```
## # A tibble: 5 × 4
##       id numero_visite visite_emplacement
##    <dbl>         <dbl> <chr>
## 1 100202             1 GH
## 2 100396             1 Pvt. docto
## 3 100396             2 Pvt. clini
## 4 100396             3 Pvt. hospi
## 5 100590             1 Pvt. docto
##   visite_cout
```

```
##          <dbl>
## 1            0
## 2         1500
## 3         1000
## 4         2500
## 5         2000
```

Ici, nous avons d'abord supprimé les observations où nous n'avons pas d'information sur l'emplacement de la visite (c'est-à-dire que nous filtrons les lignes où la variable d'emplacement de la visite est définie à `""`). Nous convertissons ensuite en valeurs numériques la variable du numero de la visite, où les chaînes `"premiere"` à `"troisieme"` sont converties en valeurs numériques `1` à `3`. Enfin, nous nous assurons que la variable du coût de la visite est numérique en utilisant `mutate()` et la fonction d'aide `as.numeric()`.

Nous allons utiliser les données d'une enquête alimentaire au Vietnam. Des femmes de Hanoi ont été interrogées sur leurs achats alimentaires, et les données collectées ont servi à créer un profil nutritionnel de chaque femme. Ici, nous utiliserons un sous-ensemble de ces données de 61 ménages qui sont venus pour 2 visites, enregistrant :

- `enerc_kcal_s_1` : l'apport énergétique de l'ingrédient/nourriture (Kcal) lors de la première visite (`_2` pour la deuxième visite)
- `sec_s_1` : l'apport sec de l'ingrédient/nourriture (g) lors de la première visite (`_2` pour la deuxième visite)
- `eau_s_1` : l'apport en eau de l'ingrédient/nourriture (g) lors de la première visite (`_2` pour la deuxième visite)
- `graisse_s_1` : l'apport en lipides de l'ingrédient/nourriture (g) lors de la première visite (`_2` pour la deuxième visite)

```
e_alimentaire_vietnam_large <-
 read_csv(here("data/fr_diet_diversity_vietnam_wide.csv"))

e_alimentaire_vietnam_large
```

```
## # A tibble: 5 × 10
##    ...1 menage_id enerc_kcal_s_1
##   <dbl>     <dbl>          <dbl>
## 1     1         1            348          2268.
## 2     2         2            354          2775.
```

```
## 5     5       211          1298.
##   enerc_kcal_s_2 sec_s_1 sec_s_2 eau_s_1
##            <dbl>   <dbl>   <dbl>   <dbl>
## 1          1386.    548.    281.   4219.
## 2          1240.    600.    284.   2376.
## 3          2075.    646.    451.   2808.
## 4          2146.    620.    807.   3457.
## 5          1191.    269.    288.   2584.
## # ℹ 3 more variables: eau_s_2 <dbl>, graisse_s_1 <dbl>,
## #   graisse_s_2 <dbl>
```

Vous devrez d'abord vérifier si vous avez un opérateur net et renommer vos colonnes si nécessaire. Ensuite, rassemblez les données enregistrées sur les deux visite dans une colonne par type d'apport (énergétique, lipides, eau et poids sec). En d'autres termes, pivotez le jeu de données en un format long de cette forme :

menage_id	visite	enerc_kcal_s	sec_s	eau_s	graisse_s


```
site_alimentaire_vietnam_large <-
site_alimentaire_vietnam_large %>%
_longer(_________)
```

```
ez votre réponse
Q_diversite_alimentaire_vietnam_large()
```

```
## Vous n'avez pas encore défini l'objet de réponse,
`Q_diversite_alimentaire_vietnam_large`.
##   1   2   3   4  ▷5◁  6
```

```
_diversite_alimentaire_vietnam_large()
```

```
##
## INDICE.
##
##   Commencez par renommer les colonnes en utilisant la
fonction `rename` du package `tidyverse` afin qu'elles aient
un séparateur double underscore.
##   - Ensuite, utilisez `pivot_longer` et spécifiez la plage
de colonnes `2:9`.
##   - Utilisez `names_sep = "__"` et `names_to = c(".value",
"visit")` pour formater les noms.
##
```

Du format long au format large

Nous venons de voir comment effectuer certaines opérations complexes du format large au format long, qui, comme nous l'avons vu dans la leçon précédente, sont essentielles pour tracer et manipuler les données. Passons maintenant à la transformation inverse.

Il peut être utile de passer du format long au format large pour transformer et filtrer les données ou encore pour traiter des valeurs manquantes (NA). Dans ce format, vos mesures / données collectées deviennent les colonnes du jeu de données.

Cette fois-ci, nous allons utiliser le jeux de données originel sur les entéropathogènes en Zambie. En effet, ce que vous manipuliez jusqu'à présent était un jeu de données **préparé pour vous**, en format large. **Le jeu de données originel est au format long** et nous allons maintenant voir la préparation des données que j'ai faite au préalable, en coulisses. Vous êtes presque en train de devenir l'enseignant de cette leçon ;)

```
athogenes_zambie_long <-
 read_csv(here("data/fr_enteropathogenes_zambie_long.csv"))
athogenes_zambie_long
```

```
## # A tibble: 5 × 5
##      ID group   LPS    LBP  IFABP
##   <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1  1002     1  222. 38414. 1294.
## 2  1002     2  390.  6840.  610.
## 3  1003     1  181. 26888.   22.5
## 4  1004     2  221.  5426.    0
## 5  1004     1  257. 49183.    0
```

Voici comment nous le convertissons du format long au format large :

```
athogenes_zambie_large <-
opathogenes_zambie_long %>%
 wider(
es_from = group,
ues_from = c(LPS, LBP, IFABP)


athogenes_zambie_large
```

```
## # A tibble: 5 × 7
##      ID LPS_1 LPS_2  LBP_1 LBP_2 IFABP_1
##   <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl>
## 1  1002  222.  390. 38414. 6840.  1294.
## 2  1003  181.   NA  26888.   NA     22.5
## 3  1004  257.  221. 49183. 5426.     0
```

```
## 4  1005   NA   369.    NA  1938.      0
## 5  1006  275.   NA  61758.   NA       0
##   IFABP_2
##     <dbl>
## 1    610.
## 2     NA
## 3      0
## 4   1010.
## 5     NA
```

Vous pouvez voir que les valeurs de la variable `group` (1 ou 2) sont ajoutées aux noms des valeurs (LPS, LBP, IFABP) pour créer les nouvelles colonnes représentant différents groupes de données : par exemple, `LPS_1` et `LPS_2`.

Nous considérons que c'est une option "avancée" du pivot car nous pivotons plusieurs variables en même temps, mais comme vous pouvez le voir, la syntaxe est assez simple. Nous utilisons les mêmes arguments `names_from` et `values_from` qu'avec les pivots plus simples que nous avons vus dans la leçon précédente.

Voyons un autre exemple, en utilisant les données de l'enquête alimentaire du Vietnam que vous avez manipulées précédemment :

```
te_alimentaire_vietnam_long <-
 read_csv(here("data/fr_diet_diversity_vietnam_long.csv"))
te_alimentaire_vietnam_long
```

```
## # A tibble: 5 × 6
##   numero_visite menage_id enerc_kcal_s sec_s
##           <dbl>     <dbl>        <dbl> <dbl>
## 1             1       348        2268.  548.
## 2             1       354        2775.  600.
## 3             1        53        3104.  646.
## 4             1        18        2802.  620.
## 5             1       211        1298.  269.
##   eau_s graisse_s
##   <dbl>     <dbl>
## 1 4219.      78.4
## 2 2376.     115.
## 3 2808.     127.
## 4 3457.      87.4
## 5 2584.      47.8
```

Ici, nous allons utiliser la variable `numero_visite` pour créer une nouvelle variable pour les différents apports enregistrés lors des deux visites :

```
te_alimentaire_vietnam_large <-
site_alimentaire_vietnam_long %>%
 wider(
es_from = numero_visite,
ues_from = c(enerc_kcal_s, sec_s, eau_s, graisse_s)


te_alimentaire_vietnam_large
```

```
## # A tibble: 5 × 9
##   menage_id enerc_kcal_s_1 enerc_kcal_s_2
##       <dbl>          <dbl>          <dbl>
## 1       348          2268.          1386.
## 2       354          2775.          1240.
## 3        53          3104.          2075.
## 4        18          2802.          2146.
## 5       211          1298.          1191.
##   sec_s_1 sec_s_2 eau_s_1 eau_s_2 graisse_s_1
##     <dbl>   <dbl>   <dbl>   <dbl>       <dbl>
## 1    548.    281.   4219.   1997.        78.4
## 2    600.    284.   2376.   3145.       115.
## 3    646.    451.   2808.   2305.       127.
## 4    620.    807.   3457.   1903.        87.4
## 5    269.    288.   2584.   2269.        47.8
## # ℹ 1 more variable: graisse_s_2 <dbl>
```

Vous pouvez voir que les valeurs de la variable `numero_visite` (1 ou 2) sont ajoutées aux noms des valeurs (`enerc_kcal_s`, `sec_s`, `graisse_s`, `eau_s`) pour créer les nouvelles colonnes représentant différents groupes de données : par exemple, `eau_s_1` et `eau_s_2`. Nous avons pivoté en format large toutes ces variables en même temps. Maintenant, chaque mesure de l'apport par visite est représentée comme une seule variable (c'est-à-dire une colonne) dans le jeu de données.

Avec ce format, il est facile de faire la somme de l'apport énergétique par ménage par exemple :

```
te_alimentaire_vietnam_large %>%
t(menage_id, enerc_kcal_s_1, enerc_kcal_s_2) %>%
e(energie_totale_kcal = enerc_kcal_s_1 + enerc_kcal_s_2) %>%
ge(menage_id)
```

```
## # A tibble: 5 × 4
##   menage_id enerc_kcal_s_1 enerc_kcal_s_2
##       <dbl>          <dbl>          <dbl>
## 1        14          1040.          1663.
## 2        17          2100.          1286.
## 3        18          2802.          2146.
## 4        22          3187.          1582.
## 5        24          2359.          2026.
##   energie_totale_kcal
##                 <dbl>
```

```
## 1                2704.
## 2                3386.
## 3                4948.
## 4                4769.
## 5                4385.
```

Cependant, vous pourriez obtenir un résultat similaire avec le format long :

```
e_alimentaire_vietnam_long %>%
by(menage_id) %>%
ize(energie_totale = sum(enerc_kcal_s))
```

```
## # A tibble: 5 × 2
##   menage_id energie_totale
##       <dbl>          <dbl>
## 1        14          2704.
## 2        17          3386.
## 3        18          4948.
## 4        22          4769.
## 5        24          4385.
```

Prenez le jeu de données `tb_visites_long` que nous avons manipulé plus haut et pivotez-le à nouveau au format large.

```
ites_large <-
ites_long %>%
vot_wider(names_from = numero_visite,
          values_from = c(visite_emplacement, visite_cout))
```

```
z votre réponse
_tb_visites_large()
```

PRACTICE (in RMD)

```
## Correct !
##   1   2   3   4   5  ▷6◁
```

```
tb_visites_large()
```

```
##
## INDICE.
##
##   Commencez par utiliser la fonction `pivot_wider` du
package `tidyverse` sur le jeu de données `tb_visites_long`.
##   Vous voudrez spécifier quelles colonnes fourniront les
nouveaux noms de colonnes (`names_from`) et de quelles
colonnes obtenir les valeurs (`values_from`).
```


```
##   Ensuite, utilisez la fonction `rename` pour changer les
noms de colonnes comme requis.
##
##   tb_visites_long %>%
##     pivot_wider(_________)
##
```

Bilan !

Vos compétences en manipulation de données viennent d'être renforcées avec le pivot avancé. Cette compétence s'avérera souvent essentielle lors de la manipulation des données du monde réel. Je ne doute pas que vous la mettrez bientôt en pratique. Elle est également essentielle, comme nous l'avons vu, pour la conception des graphiques. J'espère donc que le pivot vous sera utile non seulement pour votre manipulation de données, mais aussi pour pour la conception des graphiques.

Contributeurs

Les membres suivants de l'équipe ont contribué à cette leçon :

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network
A firm believer in science for good, striving to ally programming, health and education

CAMILLE BEATRICE VALERA

Project Manager and Scientific Collaborator, The GRAPH Network

IMANE BENSOUDA KORACHI

R Developer and Instructor, the GRAPH Network

Références