# Notes de cours | Pivot avancé

January 2024

## Intro

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 parfois 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 comment 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 combien d'argent les patients ont dépensé pour le traitement de la tuberculose
- Données de biomarqueur d'une étude sur les entéropathogènes en Zambie
- Une enquête sur l'alimentation 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 artificiel de tailles et de poids pour les 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 annee1_poids
##   <chr>  <chr>         <chr>         <chr>
## 1 A      80cm          85cm          5kg
## 2 B      85cm          90cm          7kg
## 3 C      90cm          100cm         6kg
## # ℹ 1 more variable: annee2_poids <chr>
```

Si vous pivotez toutes les colonnes de mesure, 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 pas ce que vous voulez (généralement), car maintenant vous avez deux types de données différents dans la même colonne—poids et taille.

Pour obtenir la bonne forme, vous devrez 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 enfant-période, un format approprié en format long !

Ce que fait le code ci-dessus peut ne pas être clair, mais vous devriez déjà pouvoir répondre à la question de pratique ci-dessous en faisant correspondre les modèles avec notre exemple. Après la question de pratique, nous expliquerons l'argument `names_sep` et l'identifiant ".value" plus en détail.

**Pratique 1**

Considérez cet autre ensemble de données artificielles :

```
stats <-
e::tribble(
ulte,  ~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 annee2_VIH
##   <chr>       <dbl>      <dbl> <chr>      <chr>
## 1 A              25         30 Positive   Positive
## 2 B              34         28 Negative   Positive
## 3 C              19         17 Negative   Negative
```

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

| adulte | annee | IMC | VIH |
|---|---|---|---|

```
e_stats %>%
_longer(_________)
```

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 dans la sortie du jeu de données au format long dans notre exemple en utilisant la fonction `parse_number()` de readr :

```
stats_long <-
t_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 un peu plus l'appel `pivot_longer()` que nous avons vu ci-dessus :

```
stats
```

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

```
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 cadre de données `enfant_stats` d'origine (`annee1_taille`, `annee2_taille` etc.) sont composés de trois parties :

- la période référencée : par exemple "anneee1"
- 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 |

| nom_colonne | periode | separateur | ".value" |
|---|---|---|---|
| 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 (année et poids) enregistrées.

Si nous avons un séparateur différent, cet argument changerait. Par exemple, si le séparateur était un espace vide, " ", vous auriez `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. Considérons chacun à son tour :

**La chaîne "periode"** indique que nous voulons déplacer 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 en toute 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 faire une colonne séparée pour chaque valeur distincte qui apparaît après le séparateur. Dans notre exemple, ces valeurs distinctes 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
```

---

Pour reformuler le point, le placeholder “.value” indique à `pivot_longer()` que nous voulons séparer les valeurs “taille” et “poids” dans des colonnes séparées, car il y a les deux types de valeurs qu’on retrouve après le séparateur “_” dans les noms de colonnes.

Cela signifie que si vous aviez un jeu de données au format large avec trois types de valeurs, vous obtiendriez des 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 t1_imc t2_imc
##   <chr>  <chr>  <chr>  <chr>      <chr>       <dbl>  <dbl>
## 1 a      5 mois 8 mois 13%        15%            14     15
## 2 b      7 mois 9 mois 15%        17%            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 faire 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
```


### Pratique 2

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.

Le tableau de sortie ressemble au suivant :

```
ance_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


ance_stats
```


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

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

| enfant | annee | tete | cou | hanche |
|---|---|---|---|---|

```
ance_stats %>%
_longer(_________)
```

## 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 poids_annee1
##   <chr>  <chr>         <chr>         <chr>
## 1 A      80cm          85cm          5kg
## 2 B      85cm          90cm          7kg
## 3 C      90cm          100cm         6kg
## # ℹ 1 more variable: poids_annee2 <chr>
```

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

Comment notre commande `pivot_longer()` peut-elle s'adapter à cela ? 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 auriez `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 !

### Pratique 3

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


Ce jeu de données a les colonnes suivantes :

- LPS_1 et LPS_2 : niveaux de lipopolysaccharide, mesurés par Pyrochrome LAL, en EU/mL
- LBP_1 et LBP_2 : niveaux de protéine de liaison au LPS, en pg/mL
- IFABP_1 et IFAPB_2 : niveaux de protéine de liaison aux acides gras de type intestinal, en pg/mL

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

| ID | numero_echantillon | LPS | LBP | IFABP |
|---|---|---|---|---|

```
athogenes_zambie_large %>%
_longer(_____________)
```

### Un exemple non lié à 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 des données de séries temporelles. Voyons un exemple rapide non lié à 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 mere_poids
##   <chr>        <dbl>      <dbl>       <dbl>      <dbl>
## 1 a              180         80         160         70
## 2 b              185         90         150         76
## 3 c              182         93         143         78
```

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

Pour pivoter cela 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 exemple spécial peut se produire lorsque vous essayez de pivoter 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 en R, et parfois il doit être échappé

### Pratique 4

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 VIH.annee2
##   <chr>       <dbl>      <dbl> <chr>      <chr>
## 1 A              25         30 Positive   Positive
## 2 B              34         28 Negative   Positive
## 3 C              19         17 Negative   Negative
```


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

| adulte | annee | IMC | VIH |
|---|---|---|---|

```
e_stats_point_sep %>%
_longer(_________)
```

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

Parfois, vous n'avez pas de séparateur net.

Considérez ces données d'enquête de l'Inde qui ont examiné combien d'argent les patients dépensaient pour le traitement de la tuberculose :

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

es
```

```
## # A tibble: 5 × 7
##       id premiere_visite_emplacement premiere_visite_cout
##    <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
## # ℹ 4 more variables: deuxieme_visite_emplacement <chr>,
## #   deuxieme_visite_cout <dbl>, …
```

Il n'a pas de séparateur net entre les indicateurs de temps (premier, deuxième, troisième) et le type de valeur (cout, emplacement). C'est-à-dire, plutôt que quelque chose comme "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 pivot habituelle, vous obtiendrez une 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_emplacement premiere__visite_cout
##    <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
## # ℹ 4 more variables: deuxieme__visite_emplacement <chr>,
## #   deuxieme__visite_cout <dbl>, …
```

Maintenant, nous pouvons essayer la 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 visite_cout
##    <dbl> <chr>         <chr>                     <dbl>
## 1 100202 premiere      GH                            0
## 2 100202 deuxieme      <NA>                          0
## 3 100202 troisieme     <NA>                          0
## 4 100396 premiere      Pvt. docto                 1500
## 5 100396 deuxieme      Pvt. clini                 1000
```

Maintenant, nettoyons le jeu de données :

```
tes_long %>%
primer les observations inexistantes
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 visite_cout
##    <dbl>         <dbl> <chr>                     <dbl>
## 1 100202             1 GH                            0
## 2 100396             1 Pvt. docto                 1500
## 3 100396             2 Pvt. clini                 1000
## 4 100396             3 Pvt. hospi                 2500
## 5 100590             1 Pvt. docto                 2000
```

Au-dessus, nous supprimons d'abord les observations où nous n'avons pas l'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 de numero des visites, 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 utiliserons des données d'enquête sur l'alimentation au Vietnam. Des femmes à Hanoi ont été interrogées sur leurs achats alimentaires, et cela a été utilisé pour créer des profils nutritionnels pour chaque femme. Ici, nous utiliserons un sous-ensemble de ces données pour 61 ménages qui sont venus pour 2 visites, enregistrant :

- `enerc_kcal_s_1` : l'énergie consommée à partir de l'ingrédient/nourriture (Kcal) lors de la première visite (avec `_2` pour la deuxième visite)
- `sec_s_1` : la consommation sèche à partir de l'ingrédient/nourriture (g) lors de la première visite (avec `_2` pour la deuxième visite)
- `eau_s_1` : l'eau consommée à partir de l'ingrédient/nourriture (g) lors de la première visite (avec `_2` pour la deuxième visite)
- `graisse_s_1` : les Lipides consommés à partir de l'ingrédient/nourriture (g) lors de la première visite (avec `_2` pour la deuxième visite)

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

te_alimentaire_vietnam_large
```

**PRACTICE**

**(in RMD)**

```
## # A tibble: 5 × 10
##    ...1 menage_id enerc_kcal_s_1 enerc_kcal_s_2 sec_s_1
##   <dbl>     <dbl>          <dbl>          <dbl>   <dbl>
## 1     1       348          2268.          1386.    548.
## 2     2       354          2775.          1240.    600.
## 3     3        53          3104.          2075.    646.
## 4     4        18          2802.          2146.    620.
## 5     5       211          1298.          1191.    269.
## # ℹ 5 more variables: sec_s_2 <dbl>, eau_s_1 <dbl>,
## #   eau_s_2 <dbl>, graisse_s_1 <dbl>, graisse_s_2 <dbl>
```

Vous devriez d'abord distinguer si nous avons un opérateur précis ou non. Sur cette base, renommez vos colonnes si nécessaire. Ensuite, rassemblez les différents enregistrements de visite (1 et 2) dans une seule colonne pour l'énergie, le poids des lipides, le poids de l'eau et le 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 %>%
_longer(_________)
```

## Du format long au format large

Nous venons de voir comment effectuer certaines opérations complexes du format large au format long, que nous avons vues dans la leçon précédente, essentielles pour tracer et manipuler. Voyons la transformation inverse.

Il pourrait être utile de passer du format long au format large pour effectuer différentes transformations, filtres et traitement des valeurs manquantes (NA). Dans ce format, vos mesures / données collectées deviennent les colonnes du jeu de données.

Prenons les données sur les entéropathogènes de Zambie et cette fois, prenons l'original ! En effet, ce que vous manipuliez auparavant était un jeu de données **préparé pour vous**, en format large. **Le jeu de données original 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 IFABP_2
##   <dbl> <dbl> <dbl>  <dbl> <dbl>   <dbl>   <dbl>
## 1  1002  222.  390. 38414. 6840.  1294.     610.
## 2  1003  181.   NA  26888.   NA     22.5     NA
## 3  1004  257.  221. 49183. 5426.     0       0
```

```
## 4  1005   NA   369.    NA  1938.      0     1010.
## 5  1006  275.   NA  61758.   NA       0       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 différentes données de groupe : par exemple, `LPS_1` et `LPS_2`.

Nous utilisons ce “pivotement avancé” car nous pivotons plusieurs variables en même temps, mais comme vous pouvez le voir, la syntaxe est assez simple - les mêmes arguments sont utilisés comme nous l’avons fait avec les pivotements plus simples dans la leçon précédente - `names_from` et `values_from`.

---

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

Ici, nous utiliserons la variable `numero_visite` pour créer une nouvelle variable pour l’énergie, l’eau, la matière grasse et le contenu sec des aliments enregistrés lors de différentes 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 sec_s_1 sec_s_2
##       <dbl>          <dbl>          <dbl>   <dbl>   <dbl>
## 1       348          2268.          1386.    548.    281.
## 2       354          2775.          1240.    600.    284.
## 3        53          3104.          2075.    646.    451.
## 4        18          2802.          2146.    620.    807.
## 5       211          1298.          1191.    269.    288.
```

```
## # ℹ 4 more variables: eau_s_1 <dbl>, eau_s_2 <dbl>,
## #   graisse_s_1 <dbl>, 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`, `grasisse_s`, `eau_s`) pour créer les nouvelles colonnes représentant différentes données de groupe : 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 poids 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 sommer ensemble la consommation d'énergie 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 energie_totale_k…¹
##       <dbl>          <dbl>          <dbl>              <dbl>
## 1        14          1040.          1663.              2704.
## 2        17          2100.          1286.              3386.
## 3        18          2802.          2146.              4948.
## 4        22          3187.          1582.              4769.
## 5        24          2359.          2026.              4385.
## # ℹ abbreviated name: ¹energie_totale_kcal
```

Cependant, vous pourriez obtenir quelque chose de similaire dans le format long :

```
te_alimentaire_vietnam_long %>%
_by(menage_id) %>%
tize(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.
```


**Pratique 5**


Prenez le jeu de données `tb_visites_long` que nous avons manipulé ci-dessus et pivotez-le à nouveau en format large.

```
ites_long %>%
 wider(_________)
```

## Bilan !

Vos compétences en manipulation de données viennent d'être renforcées avec le pivotement avancé. Cette compétence s'avérera souvent essentielle lors de la manipulation de 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 pivotement vous sera utile non seulement pour votre manipulation de données, mais aussi pour vos tâches de 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

# Références

# Solutions des exercices pratiques

## *Solution exercice pratique 1*

```
stats %>%
vot_longer(cols = 2:5,
           names_sep = "_",
           names_to = c("annee", ".value"))
```

```
## # A tibble: 5 × 4
##   adulte annee    IMC VIH
##   <chr>  <chr>  <dbl> <chr>
## 1 A      annee1    25 Positive
## 2 A      annee2    30 Positive
## 3 B      annee1    34 Negative
## 4 B      annee2    28 Positive
## 5 C      annee1    19 Negative
```

## *Solution exercice pratique 2*

```
nce_stats %>%
vot_longer(cols = 2:7,
           names_to = c("annee", ".value"),
           names_sep = "_")
```

```
## # A tibble: 5 × 5
##   enfant annee  tete   cou hanche
##   <chr>  <chr> <dbl> <dbl>  <dbl>
## 1 a      ann1     45    23     51
## 2 a      ann2     48    24     52
## 3 b      ann1     48    24     52
## 4 b      ann2     50    26     52
## 5 c      ann1     50    24     53
```

## *Solution exercice pratique 3*

```
pathogenes_zambie_large %>%
_longer(!ID,
        names_to = c(".value", "numero_echantillon"),
        names_sep = "_")
```

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

### *Solution exercice pratique 4*

```
stats_point_sep %>%
pivot_longer(cols = 2:5,
             names_sep = "\\.",
             names_to = c(".value", "annee"))
```

```
## # A tibble: 5 × 4
##   adulte annee    IMC VIH
##   <chr>  <chr>  <dbl> <chr>
## 1 A      annee1    25 Positive
## 2 A      annee2    30 Positive
## 3 B      annee1    34 Negative
## 4 B      annee2    28 Positive
## 5 C      annee1    19 Negative
```

### *Solution exercice pratique 5*

```
ltes_long %>%
pivot_wider(names_from = numero_visite,
            values_from = c(visite_emplacement, visite_cout))
```

```
## # A tibble: 5 × 7
##       id visite_emplacement_premiere visite_emplacement_de…¹
##    <dbl> <chr>                       <chr>
## 1 100202 GH                          <NA>
## 2 100396 Pvt. docto                  Pvt. clini
## 3 100590 Pvt. docto                  Pvt. docto
## 4 100687 Pvt. hospi                  Pvt. hospi
## 5 100784 Pvt. docto                  GH
## # ℹ abbreviated name: ¹visite_emplacement_deuxieme
## # ℹ 4 more variables: visite_emplacement_troisieme <chr>, …
```