Manipulation de données avec R

2 - Manipulation de tableaux

Robin Cura & Lise Vaudor
d'après L. Vaudor : Formation startR (2018).

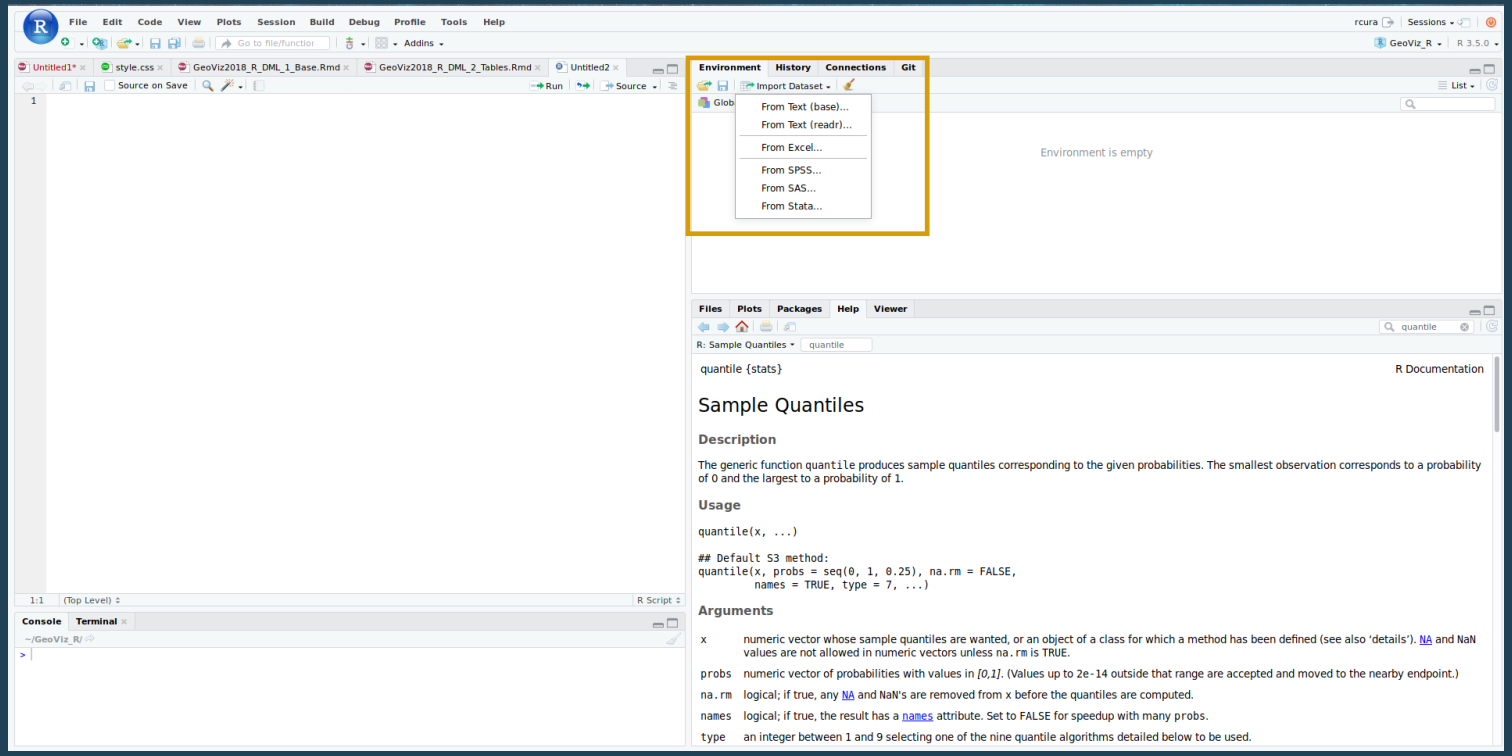
15/10/2018

École Thématique GeoViz 2018

Sommaire

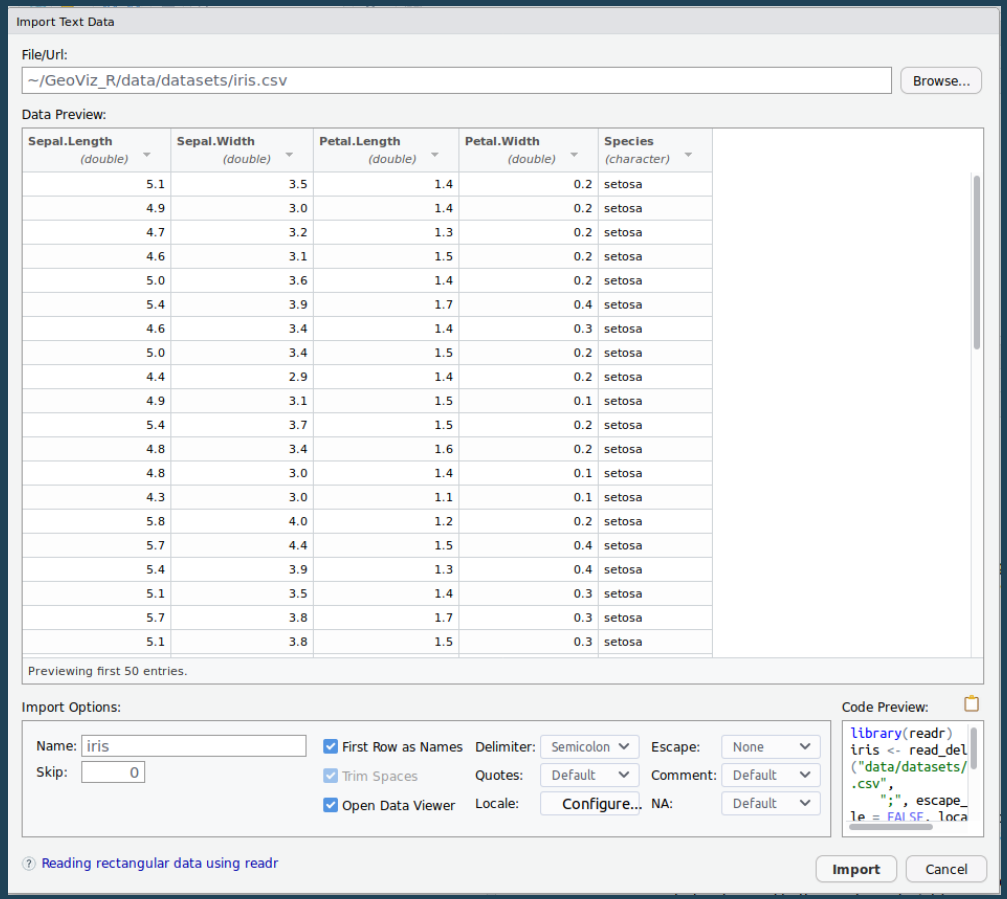
Lecture de tables : readr

- RStudio dispose d'un outil graphique pour importer des données :



Lecture de tables : readr

- RStudio dispose d'un outil graphique pour importer des données :



Lecture de tables : `readr`

- L'interface graphique génère un code que l'on peut copier/coller dans ses scripts, mais on peut aussi écrire le code correspondant directement :
    - On commence par charger le *package* `readr` :

```
library(readr)
```

    - On peut alors choisir entre les différentes fonctions du package :
        - `read_csv()` : lecture d'un csv séparé par des virgules (`,`), format numérique anglais (`.`)
        - `read_csv2()` : lecture d'un csv séparé par des points-virgules (`;`), format numérique français (`,`)
        - `read_delim()` : fonction plus générique et paramétrable, pour lire des fichiers délimités plus spécifiques (délimité par des tabulations, contenant des caractères spéciaux etc.)
            - Pour comprendre les arguments : `?read_delim`

Lecture de tables : readr

On va ouvrir et lire la table de données `dans_ma_rue.csv` :

```
library(readr)
df_dmr <- read_delim("data/dans-ma-rue.csv",
                     delim = ";",
                     escape_double = FALSE,
                     trim_ws = TRUE)
```

```
## Parsed with column specification:
## cols(
##   TYPE = col_character(),
##   SOUSTYPE = col_character(),
##   ADRESSE = col_character(),
##   CODE_POSTAL = col_integer(),
##   VILLE = col_character(),
##   ARRONDISSEMENT = col_double(),
##   DATEDECL = col_datetime(format = ""),
##   `ANNEE DECLARATION` = col_integer(),
##   `MOIS DECLARATION` = col_integer(),
##   NUMERO = col_double(),
##   PREFIXE = col_character(),
##   INTERVENANT = col_character(),
##   `CONSEIL DE QUARTIER` = col_character(),
##   OBJECTID = col_integer(),
##   geo_shape = col_character(),
##   geo_point_2d = col_character()
## )
```

- Un message indique les colonnes et les types qui leur ont été attribuées automatiquement.

Lecture de tables : readr

```
## Parsed with column specification:
## cols(
##   TYPE = col_character(),
##   SOUSTYPE = col_character(),
##   ADRESSE = col_character(),
##   CODE_POSTAL = col_integer(),
##   VILLE = col_character(),
##   ARRONDISSEMENT = col_double(),
##   DATEDECL = col_datetime(format = ""),
##   `ANNEE DECLARATION` = col_integer(),
##   `MOIS DECLARATION` = col_integer(),
##   NUMERO = col_double(),
##   PREFIXE = col_character(),
##   INTERVENANT = col_character(),
##   `CONSEIL DE QUARTIER` = col_character(),
##   OBJECTID = col_integer(),
##   geo_shape = col_character(),
##   geo_point_2d = col_character()
## )
```

- On remarque tout de suite plusieurs problèmes :
  - Le code postal a été interpreté comme un `integer`, ce qui peut être gênant
  - Certaines colonnes contiennent des espaces (`ANNEE DECLARATION`)
    - Dans ce cas, on encadre le nom de la colonne avec des "backticks", i.e. `` "`" `` (Alt Gr + 7)
  - L'information spatiale est contenue dans deux colonnes (`geo_shape` et `geo_point_2d`) peu explicites

Lecture de tables : readr

- Quand on lit une table avec `readr`, l'objet retourné est un `tibble` (`tbl`):

```
class(df_dmr)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

- L'affichage des `tibbles` est plus condensé et explicite, et ce format simplifie la gestion, en particulier pour la manipulation avec `dplyr` :

```
df_dmr
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Mobi… Mobilie… 16 rue…       75011 Pari…             11
##  2 Obje… Gravats… 15 rue…       75018 Pari…             18
##  3 Obje… Meubles… 16 rue…       75019 Pari…             19
##  4 Voir… Station… 15 rue…       75003 Pari…              3
##  5 Prop… Malprop… 24 rue…       75018 Pari…             18
##  6 Mobi… Mobilie… 9 vill…       75014 Pari…             14
##  7 Obje… Planche… 19 rue…       75011 Pari…             11
##  8 Obje… Objets … 3 aven…       75016 Pari…             16
##  9 Voir… Trottoi… 185b r…       75018 Pari…             18
## 10 Prop… Malprop… 138 ru…       75020 Pari…             20
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

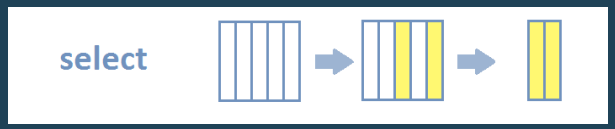
Manipuler des données avec `dplyr`

- Avant de vouloir modifier le jeu de données chargé, voyons déjà comment manipuler son contenu
- On utilise pour cela le *package* `dplyr`, et notamment 3 fonctions :
    - **`select()`** permet de sélectionner des variables (colonnes)
    - **`filter()`** permet de filtrer les individus (lignes)
    - **`arrange()`** permet de réarranger le tableau selon l'ordre des variables

On commence toujours par charger le package :

```
library(dplyr)
```

dplyr: select()



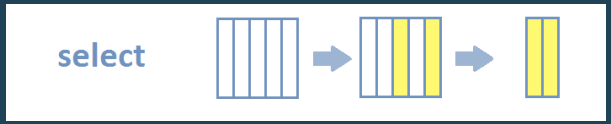
select() permet de sélectionner des variables

Par exemple, on peut sélectionner des variables de la table `df_dmr` :

```
print(df_dmr)
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Mobi… Mobilie… 16 rue…       75011 Pari…             11
##  2 Obje… Gravats… 15 rue…       75018 Pari…             18
##  3 Obje… Meubles… 16 rue…       75019 Pari…             19
##  4 Voir… Station… 15 rue…       75003 Pari…              3
##  5 Prop… Malprop… 24 rue…       75018 Pari…             18
##  6 Mobi… Mobilie… 9 vill…       75014 Pari…             14
##  7 Obje… Planche… 19 rue…       75011 Pari…             11
##  8 Obje… Objets … 3 aven…       75016 Pari…             16
##  9 Voir… Trottoi… 185b r…       75018 Pari…             18
## 10 Prop… Malprop… 138 ru…       75020 Pari…             20
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

dplyr: select()



select() permet de sélectionner des variables

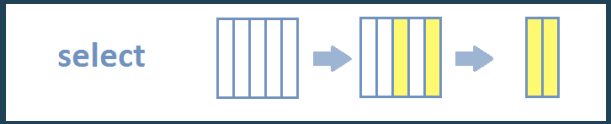
Par exemple, on peut sélectionner des variables de la table `df_dmr` :

- Par leur nom

```
select(df_dmr, TYPE, SOUSTYPE, CODE_POSTAL)
```

```
## # A tibble: 343,642 x 3
##    TYPE                          SOUSTYPE                      CODE_POSTAL
##    <chr>                         <chr>                               <int>
##  1 Mobiliers urbains dégradés (… Mobiliers de protection :Pot…       75011
##  2 Objets abandonnés             Gravats ou déchets de chanti…       75018
##  3 Objets abandonnés             Meubles et éléments de décor…       75019
##  4 Voirie et déplacements        Stationnement gênant de véhi…       75003
##  5 Propreté                      Malpropreté des mobiliers de…       75018
##  6 Mobiliers urbains dégradés (… Mobiliers de protection :Pot…       75014
##  7 Objets abandonnés             Planches et palettes                75011
##  8 Objets abandonnés             Objets entrant dans plusieur…       75016
##  9 Voirie et déplacements        Trottoirs:Revêtement manquant       75018
## 10 Propreté                      Malpropreté du sol:Déchets d…       75020
## # ... with 343,632 more rows
```

dplyr: select()



select() permet de sélectionner des variables

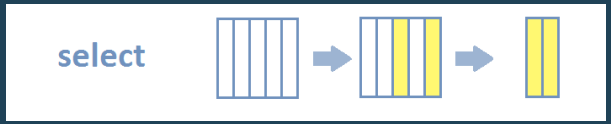
Par exemple, on peut sélectionner des variables de la table `df_dmr` :

- Par leur nom, par leur numéro

```
select(df_dmr, 3:7)
```

```
## # A tibble: 343,642 x 5
##    ADRESSE           CODE_POSTAL VILLE  ARRONDISSEMENT DATEDECL
##    <chr>                   <int> <chr>           <dbl> <dttm>
##  1 16 rue de l'oril…       75011 Paris…             11 2017-12-02 01:00:00
##  2 15 rue joseph di…       75018 Paris…             18 2017-11-30 01:00:00
##  3 16 rue archereau…       75019 Paris…             19 2017-12-03 01:00:00
##  4 15 rue réaumur, …       75003 Paris…              3 2017-12-04 01:00:00
##  5 24 rue custine, …       75018 Paris…             18 2017-12-03 01:00:00
##  6 9 villa virginie…       75014 Paris…             14 2017-10-15 02:00:00
##  7 19 rue de mont-l…       75011 Paris…             11 2017-12-02 01:00:00
##  8 3 avenue boudon,…       75016 Paris…             16 2017-12-05 01:00:00
##  9 185b rue ordener…       75018 Paris…             18 2017-07-17 02:00:00
## 10 138 rue pellepor…       75020 Paris…             20 2017-12-05 01:00:00
## # ... with 343,632 more rows
```

dplyr: select()



select() permet de sélectionner des variables

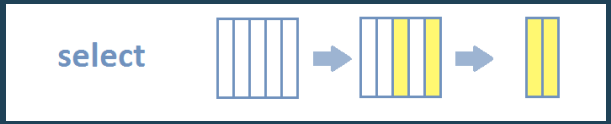
Par exemple, on peut sélectionner des variables de la table `df_dmr` :

- Par leur nom, par leur numéro, en excluant des variables

```
select(df_dmr, -(4:7), - SOUSTYPE)
```

```
## # A tibble: 343,642 x 11
##    TYPE  ADRESSE `ANNEE DECLARAT… `MOIS DECLARATI… NUMERO PREFIXE
##    <chr> <chr>              <int>            <int>  <dbl> <chr>
##  1 Mobi… 16 rue…             2017               12    576 S
##  2 Obje… 15 rue…             2017               11   9444 A
##  3 Obje… 16 rue…             2017               12    696 G
##  4 Voir… 15 rue…             2017               12   1259 A
##  5 Prop… 24 rue…             2017               12    818 S
##  6 Mobi… 9 vill…             2017               10   5376 G
##  7 Obje… 19 rue…             2017               12    525 G
##  8 Obje… 3 aven…             2017               12   1533 S
##  9 Voir… 185b r…             2017                7   4433 S
## 10 Prop… 138 ru…             2017               12   1653 S
## # ... with 343,632 more rows, and 5 more variables: INTERVENANT <chr>,
## #   `CONSEIL DE QUARTIER` <chr>, OBJECTID <int>, geo_shape <chr>,
## #   geo_point_2d <chr>
```

dplyr: select()



select() permet de sélectionner des variables

Par exemple, on peut sélectionner des variables de la table `df_dmr` :

- Par leur nom, par leur numéro, en excluant des variables, ou avec des fonctions de sélection :
  - `starts_with()` pour les variables qui commencent par un texte donné
  - `ends_with()` pour les variables qui terminent par un texte donné
  - `contains()` pour les variables qui contiennent un texte donné

dplyr: select()

```
colnames(df_dmr)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE DECLARATION"   "MOIS DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL DE QUARTIER" "OBJECTID"            "geo_shape"
## [16] "geo_point_2d"
```

```
select(df_dmr, starts_with("A"))
```

```
## # A tibble: 343,642 x 3
##    ADRESSE                          ARRONDISSEMENT `ANNEE DECLARATION`
##    <chr>                                     <dbl>               <int>
##  1 16 rue de l'orillon, 75011 PARIS             11                2017
##  2 15 rue joseph dijon, 75018 PARIS             18                2017
##  3 16 rue archereau, 75019 PARIS                19                2017
##  4 15 rue réaumur, 75003 PARIS                   3                2017
##  5 24 rue custine, 75018 PARIS                  18                2017
##  6 9 villa virginie, 75014 PARIS                14                2017
##  7 19 rue de mont-louis, 75011 PARIS            11                2017
##  8 3 avenue boudon, 75016 PARIS                 16                2017
##  9 185b rue ordener, 75018 PARIS                18                2017
## 10 138 rue pelleport, 75020 PARIS               20                2017
## # ... with 343,632 more rows
```

dplyr: select()

```
colnames(df_dmr)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE DECLARATION"   "MOIS DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL DE QUARTIER" "OBJECTID"            "geo_shape"
## [16] "geo_point_2d"
```

```
select(df_dmr, ends_with("DECLARATION"))
```

```
## # A tibble: 343,642 x 2
##    `ANNEE DECLARATION` `MOIS DECLARATION`
##                  <int>              <int>
##  1                2017                 12
##  2                2017                 11
##  3                2017                 12
##  4                2017                 12
##  5                2017                 12
##  6                2017                 10
##  7                2017                 12
##  8                2017                 12
##  9                2017                  7
## 10                2017                 12
## # ... with 343,632 more rows
```

dplyr: select()

```
colnames(df_dmr)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE DECLARATION"   "MOIS DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL DE QUARTIER" "OBJECTID"            "geo_shape"
## [16] "geo_point_2d"
```

```
select(df_dmr, contains("TYPE"))
```

```
## # A tibble: 343,642 x 2
##    TYPE                                SOUSTYPE
##    <chr>                               <chr>
##  1 Mobiliers urbains dégradés (arrach… Mobiliers de protection :Potelet, …
##  2 Objets abandonnés                   Gravats ou déchets de chantier
##  3 Objets abandonnés                   Meubles et éléments de décoration
##  4 Voirie et déplacements              Stationnement gênant de véhicule m…
##  5 Propreté                            Malpropreté des mobiliers de colle…
##  6 Mobiliers urbains dégradés (arrach… Mobiliers de protection :Potelet, …
##  7 Objets abandonnés                   Planches et palettes
##  8 Objets abandonnés                   Objets entrant dans plusieurs caté…
##  9 Voirie et déplacements              Trottoirs:Revêtement manquant
## 10 Propreté                            Malpropreté du sol:Déchets divers …
## # ... with 343,632 more rows
```

dplyr: filter()

filter() permet de filtrer les lignes du tableau en fonction des valeurs de telle ou telle variable.

```
nrow(df_dmr)
```

```
## [1] 343642
```

```
df_dmr_paris6 <- filter(df_dmr, CODE_POSTAL == 75006)
nrow(df_dmr_paris6)
```

```
## [1] 5892
```

```
head(df_dmr_paris6)
```

```
## # A tibble: 6 x 16
##   TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##   <chr> <chr>    <chr>         <int> <chr>          <dbl>
## 1 Graf… Graffit… 1 rue …       75006 Pari…              6
## 2 Prob… Chantie… 25 rue…       75006 Pari…              6
## 3 Prop… Malprop… 2 rue …       75006 Pari…              6
## 4 Voir… Trottoi… 6 plac…       75006 Pari…              6
## 5 Obje… Épave d… place …       75006 Pari…              6
## 6 Voir… Station… 15 rue…       75006 Pari…              6
## # ... with 10 more variables: DATEDECL <dttm>, `ANNEE DECLARATION` <int>,
## #   `MOIS DECLARATION` <int>, NUMERO <dbl>, PREFIXE <chr>,
## #   INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>, OBJECTID <int>,
```

dplyr: filter()

filter() permet de filtrer les lignes du tableau en fonction des valeurs de telle ou telle variable.

- On peut combiner plusieurs conditions logiques :
  - ET logique, par une virgule ou un &,
  - OU logique, par l'opérateur |,
  - NON logique par l'opérateur !).

dplyr: filter()

```
nrow(df_dmr)
```

```
## [1] 343642
```

```
filter(df_dmr, CODE_POSTAL == 75006 & `ANNEE DECLARATION` == 2018)
```

```
## # A tibble: 359 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Obje… Épave d… place …       75006 Pari…              6
##  2 Voir… Station… 15 rue…       75006 Pari…              6
##  3 Eau … Dysfonc… 62 rue…       75006 Pari…              6
##  4 Graf… Graffit… 8 rue …       75006 Pari…              6
##  5 Obje… Autre o… 12 rue…       75006 Pari…              6
##  6 Prob… Chantie… 9 rue …       75006 Pari…              6
##  7 Graf… Graffit… 19 rue…       75006 Pari…              6
##  8 Prop… Malprop… 9 rue …       75006 Pari…              6
##  9 Voir… Station… 16 rue…       75006 Pari…              6
## 10 Arbr… Problèm… 32 rue…       75006 Pari…              6
## # ... with 349 more rows, and 10 more variables: DATEDECL <dttm>, `ANNEE
## #   DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

```
nrow(filter(df_dmr, CODE_POSTAL == 75006, `ANNEE DECLARATION` == 2018))
```

```
## [1] 359
```

dplyr: filter()

```
nrow(df_dmr)
```

```
## [1] 343642
```

```
filter(df_dmr, CODE_POSTAL == 75006 | CODE_POSTAL == 75007)
```

```
## # A tibble: 13,668 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Prop… Malprop… 20 rue…       75007 Pari…              7
##  2 Graf… Graffit… 23 rue…       75007 Pari…              7
##  3 Graf… Graffit… 1 rue …       75006 Pari…              6
##  4 Mobi… Autres … 20 ave…       75007 Pari…              7
##  5 Mobi… Mobilie… avenue…       75007 Pari…              7
##  6 Obje… Planche… 8 rue …       75007 Pari…              7
##  7 Prob… Chantie… 25 rue…       75006 Pari…              6
##  8 Prop… Malprop… 46 ave…       75007 Pari…              7
##  9 Prop… Malprop… boulev…       75007 Pari…              7
## 10 Obje… Autre o… avenue…       75007 Pari…              7
## # ... with 13,658 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

dplyr: filter()

```
nrow(df_dmr)
```

```
## [1] 343642
```

```
filter(df_dmr, !(CODE_POSTAL == 75006))
```

```
## # A tibble: 337,750 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Mobi… Mobilie… 16 rue…       75011 Pari…             11
##  2 Obje… Gravats… 15 rue…       75018 Pari…             18
##  3 Obje… Meubles… 16 rue…       75019 Pari…             19
##  4 Voir… Station… 15 rue…       75003 Pari…              3
##  5 Prop… Malprop… 24 rue…       75018 Pari…             18
##  6 Mobi… Mobilie… 9 vill…       75014 Pari…             14
##  7 Obje… Planche… 19 rue…       75011 Pari…             11
##  8 Obje… Objets … 3 aven…       75016 Pari…             16
##  9 Voir… Trottoi… 185b r…       75018 Pari…             18
## 10 Prop… Malprop… 138 ru…       75020 Pari…             20
## # ... with 337,740 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

dplyr: filter()

```
nrow(df_dmr)
```

```
## [1] 343642
```

```
# L'opérateur %in% permet de chercher parmi les valeurs d'un vecteur
filter(df_dmr, CODE_POSTAL %in% c(75006, 75008))
```

```
## # A tibble: 10,768 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Prop… Malprop… 38 rue…       75008 Pari…              8
##  2 Graf… Graffit… 1 rue …       75006 Pari…              6
##  3 Prob… Chantie… 25 rue…       75006 Pari…              6
##  4 Arbr… Problèm… jardin…       75008 Pari…              8
##  5 Prop… Malprop… 2 rue …       75006 Pari…              6
##  6 Voir… Trottoi… 6 plac…       75006 Pari…              6
##  7 Prop… Malprop… 22 rue…       75008 Pari…              8
##  8 Voir… Gênes à… 19 rue…       75008 Pari…              8
##  9 Obje… Épave d… place …       75006 Pari…              6
## 10 Voir… Station… 15 rue…       75006 Pari…              6
## # ... with 10,758 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

dplyr: arrange()

Pour trier un tableau selon l'ordre d'une variable (réarranger les lignes d'un tableau), on peut faire appel à la fonction `arrange` :

```
df_dmr
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Mobi… Mobilie… 16 rue…       75011 Pari…             11
##  2 Obje… Gravats… 15 rue…       75018 Pari…             18
##  3 Obje… Meubles… 16 rue…       75019 Pari…             19
##  4 Voir… Station… 15 rue…       75003 Pari…              3
##  5 Prop… Malprop… 24 rue…       75018 Pari…             18
##  6 Mobi… Mobilie… 9 vill…       75014 Pari…             14
##  7 Obje… Planche… 19 rue…       75011 Pari…             11
##  8 Obje… Objets … 3 aven…       75016 Pari…             16
##  9 Voir… Trottoi… 185b r…       75018 Pari…             18
## 10 Prop… Malprop… 138 ru…       75020 Pari…             20
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

```
arrange(df_dmr, CODE_POSTAL)
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Obje… Objets … 12 rue…       75001 Pari…              1
##  2 Prop… Malprop… 23 Rue…       75001 Pari…              1
##  3 Graf… Affiche… 12 rue…       75001 Pari…              1
##  4 Graf… Graffit… 3 rue …       75001 Pari…              1
##  5 Graf… Graffit… 11 rue…       75001 Pari…              1
##  6 Graf… Graffit… place …       75001 Pari…              1
##  7 Du v… Pot ou … 296  r…       75001 Pari…              1
##  8 Obje… Autre o… 17 rue…       75001 Pari…              1
##  9 Écla… Fils él… 1 rue …       75001 Pari…              1
## 10 Graf… Graffit… 20 rue…       75001 Pari…              1
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

dplyr: arrange()

On peut aussi trier un `tibble` par plusieurs variables

```
arrange(df_dmr, CODE_POSTAL)
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Obje… Objets … 12 rue…       75001 Pari…              1
##  2 Prop… Malprop… 23 Rue…       75001 Pari…              1
##  3 Graf… Affiche… 12 rue…       75001 Pari…              1
##  4 Graf… Graffit… 3 rue …       75001 Pari…              1
##  5 Graf… Graffit… 11 rue…       75001 Pari…              1
##  6 Graf… Graffit… place …       75001 Pari…              1
##  7 Du v… Pot ou … 296  r…       75001 Pari…              1
##  8 Obje… Autre o… 17 rue…       75001 Pari…              1
##  9 Écla… Fils él… 1 rue …       75001 Pari…              1
## 10 Graf… Graffit… 20 rue…       75001 Pari…              1
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

```
arrange(df_dmr, CODE_POSTAL, TYPE)
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Arbr… Problèm… 9 rue …       75001 Pari…              1
##  2 Arbr… Problèm… 10 rue…       75001 Pari…              1
##  3 Arbr… Problèm… 4 plac…       75001 Pari…              1
##  4 Arbr… Problèm… rue be…       75001 Pari…              1
##  5 Arbr… Problèm… 67 rue…       75001 Pari…              1
##  6 Arbr… Problèm… rue de…       75001 Pari…              1
##  7 Arbr… Problèm… 38 rue…       75001 Pari…              1
##  8 Arbr… Problèm… 53 rue…       75001 Pari…              1
##  9 Arbr… Problèm… 22 rue…       75001 Pari…              1
## 10 Arbr… Problèm… place …       75001 Pari…              1
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

dplyr: arrange()

On peut aussi trier un `tibble` par plusieurs variables, et par ordre décroissant avec `desc()`

```
arrange(df_dmr, CODE_POSTAL, TYPE)
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Arbr… Problèm… 9 rue …       75001 Pari…              1
##  2 Arbr… Problèm… 10 rue…       75001 Pari…              1
##  3 Arbr… Problèm… 4 plac…       75001 Pari…              1
##  4 Arbr… Problèm… rue be…       75001 Pari…              1
##  5 Arbr… Problèm… 67 rue…       75001 Pari…              1
##  6 Arbr… Problèm… rue de…       75001 Pari…              1
##  7 Arbr… Problèm… 38 rue…       75001 Pari…              1
##  8 Arbr… Problèm… 53 rue…       75001 Pari…              1
##  9 Arbr… Problèm… 22 rue…       75001 Pari…              1
## 10 Arbr… Problèm… place …       75001 Pari…              1
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

```
arrange(df_dmr, desc(CODE_POSTAL), TYPE)
```

```
## # A tibble: 343,642 x 16
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL VILLE ARRONDISSEMENT
##    <chr> <chr>    <chr>         <int> <chr>          <dbl>
##  1 Arbr… Problèm… 174 ru…       75020 Pari…             20
##  2 Arbr… Problèm… 84 bou…       75020 Pari…             20
##  3 Arbr… Problèm… 353 ru…       75020 Pari…             20
##  4 Arbr… Problèm… 6  rue…       75020 Pari…             20
##  5 Arbr… Problèm… 5 rue …       75020 Pari…             20
##  6 Arbr… Problèm… 20 rue…       75020 Pari…             20
##  7 Arbr… Problèm… rue fr…       75020 Pari…             20
##  8 Arbr… Problèm… 120 av…       75020 Pari…             20
##  9 Arbr… Problèm… 50b  r…       75020 Pari…             20
## 10 Arbr… Problèm… 74  bo…       75020 Pari…             20
## # ... with 343,632 more rows, and 10 more variables: DATEDECL <dttm>,
## #   `ANNEE DECLARATION` <int>, `MOIS DECLARATION` <int>, NUMERO <dbl>,
## #   PREFIXE <chr>, INTERVENANT <chr>, `CONSEIL DE QUARTIER` <chr>,
## #   OBJECTID <int>, geo_shape <chr>, geo_point_2d <chr>
```

Enchaîner des opérations

Si on veut ré-organiser un tableau avec plusieurs opérations, on peut procéder de plusieurs manières :

- En mettant chaque étape dans une variable

```
tableau_reorganise <- select(df_dmr, TYPE, CODE_POSTAL, `ANNEE DECLARATION`)
tableau_reorganise <- filter(tableau_reorganise, CODE_POSTAL %in% c(75006, 75008))
tableau_reorganise <- arrange(tableau_reorganise, desc(`ANNEE DECLARATION`), TYPE)
tableau_reorganise
```

```
## # A tibble: 10,768 x 3
##    TYPE                        CODE_POSTAL `ANNEE DECLARATION`
##    <chr>                             <int>               <int>
##  1 Arbres, végétaux et animaux       75006                2018
##  2 Arbres, végétaux et animaux       75008                2018
##  3 Arbres, végétaux et animaux       75008                2018
##  4 Arbres, végétaux et animaux       75008                2018
##  5 Arbres, végétaux et animaux       75008                2018
##  6 Arbres, végétaux et animaux       75008                2018
##  7 Arbres, végétaux et animaux       75008                2018
##  8 Arbres, végétaux et animaux       75008                2018
##  9 Arbres, végétaux et animaux       75008                2018
## 10 Eau et assainissement             75006                2018
## # ... with 10,758 more rows
```

Enchaîner des opérations

Si on veut ré-organiser un tableau avec plusieurs opérations, on peut procéder de plusieurs manières :

- En mettant chaque étape dans une variable -> risque d'erreur important
- En imbriquant les opérations...

```
arrange(
filter(
select(df_dmr,
       TYPE, CODE_POSTAL, `ANNEE DECLARATION`),
CODE_POSTAL %in% c(75006, 75008)),
desc(`ANNEE DECLARATION`, TYPE)
)
```

```
## # A tibble: 10,768 x 3
##    TYPE                                    CODE_POSTAL `ANNEE DECLARATIO…
##    <chr>                                         <int>               <int>
##  1 Objets abandonnés                             75006                2018
##  2 Voirie et déplacements                        75006                2018
##  3 Voirie et déplacements                        75008                2018
##  4 Graffitis, tags, affiches et autocollan…      75008                2018
##  5 Voirie et déplacements                        75008                2018
##  6 Eau et assainissement                         75006                2018
##  7 Graffitis, tags, affiches et autocollan…      75008                2018
##  8 Propreté                                      75008                2018
##  9 Voirie et déplacements                        75008                2018
## 10 Objets abandonnés                             75008                2018
## # ... with 10,758 more rows
```

Enchaîner des opérations

Si on veut ré-organiser un tableau avec plusieurs opérations, on peut procéder de plusieurs manières :

- En mettant chaque étape dans une variable -> risque d'erreur important
- En imbriquant les opérations... -> c'est illisible

```
arrange(
filter(
select(df_dmr,
       TYPE, CODE_POSTAL, `ANNEE DECLARATION`),
CODE_POSTAL %in% c(75006, 75008)),
desc(`ANNEE DECLARATION`, TYPE)
)
```

```
## # A tibble: 10,768 x 3
##    TYPE                                    CODE_POSTAL `ANNEE DECLARATIO…
##    <chr>                                         <int>              <int>
##  1 Objets abandonnés                             75006               2018
##  2 Voirie et déplacements                        75006               2018
##  3 Voirie et déplacements                        75008               2018
##  4 Graffitis, tags, affiches et autocollan…      75008               2018
##  5 Voirie et déplacements                        75008               2018
##  6 Eau et assainissement                         75006               2018
##  7 Graffitis, tags, affiches et autocollan…      75008               2018
##  8 Propreté                                      75008               2018
##  9 Voirie et déplacements                        75008               2018
## 10 Objets abandonnés                             75008               2018
## # ... with 10,758 more rows
```

Enchaîner des opérations

Si on veut ré-organiser un tableau avec plusieurs opérations, on peut procéder de plusieurs manières :

- En mettant chaque étape dans une variable -> risque d'erreur important
- En imbriquant les opérations... -> c'est illisible
- En utilisant un opérateur de chaînage, le pipe : %>%

```
df_dmr$CODE_POSTAL %>% mean() %>% log()
```

```
## [1] 11.22542
```

```
log(mean(df_dmr$CODE_POSTAL))
```

```
## [1] 11.22542
```

Enchaîner des opérations

Si on veut ré-organiser un tableau avec plusieurs opérations, on peut procéder de plusieurs manières :

- En mettant chaque étape dans une variable -> risque d'erreur important
- En imbriquant les opérations... -> c'est illisible
- En utilisant un opérateur de chaînage, le pipe : %>%

```
tableau_reorganise <- select(df_dmr, TYPE, CODE_POSTAL, `ANNEE DECLARATION`)
tableau_reorganise <- filter(tableau_reorganise, CODE_POSTAL %in% c(75006, 75008))
tableau_reorganise <- arrange(tableau_reorganise, desc(`ANNEE DECLARATION`), TYPE)
```

peut être écrit :

```
tableau_reorganise <- df_dmr %>%
    select(TYPE, CODE_POSTAL, `ANNEE DECLARATION`) %>%
    filter(CODE_POSTAL %in% c(75006, 75008)) %>%
    arrange(desc(`ANNEE DECLARATION`), TYPE)
```

N.B : Attention, le premier argument des fonctions (le tableau) est ici implicitement communiqué par le pipe

Modifier un tableau de données avec `dplyr`

Ré-organiser des données ne suffit généralement pas pour les adapter à une représentation, il faut alors modifier le tableau de données en entrée :

- **`rename()`** permet de renommer des colonnes
- **`mutate()`** permet de créer et ajouter de nouvelles variables (colonnes)
- **`group_by()`** permet de réaliser des agrégations
- **`summarise()`** permet de calculer un résumé statistique, en particulier au cours de la création d'un tableau agrégé par **`group_by()`**
- **`left_join()`**, **`inner_join()`** etc. permettent de réaliser des jointures

Renommer des colonnes avec rename

Pour simplifier ou modifier des noms de colonnes :

```
colnames(df_dmr)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE DECLARATION"   "MOIS DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL DE QUARTIER" "OBJECTID"            "geo_shape"
## [16] "geo_point_2d"
```

```
df_dmr_renamed <- rename(df_dmr, ANNEE_DECLARATION = `ANNEE DECLARATION`)
colnames(df_dmr_renamed)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE_DECLARATION"   "MOIS DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL DE QUARTIER" "OBJECTID"            "geo_shape"
## [16] "geo_point_2d"
```

Renommer des colonnes avec `rename`

On peut aussi renommer plusieurs colonnes d'un coup :

```
df_dmr_renamed <- rename(df_dmr,
                        ANNEE_DECLARATION = `ANNEE DECLARATION`,
                        MOIS_DECLARATION = `MOIS DECLARATION`)
colnames(df_dmr_renamed)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE_DECLARATION"   "MOIS_DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL DE QUARTIER" "OBJECTID"            "geo_shape"
## [16] "geo_point_2d"
```

Et même utiliser des fonctions pour renommer automatiquement (plus complexe), par exemple en manipulant les chaînes de craactères avec le package `stringr` :

```
library(stringr)
df_dmr_renamed <- rename_at(.tbl = df_dmr,
                           .vars = vars(contains(" ")),
                           .funs = funs(str_replace_all(string = .,
                                                        pattern = " ",
                                                        replacement = "_")))
```

Renommer des colonnes avec `rename`

```
df_dmr_renamed <- rename_at(.tbl = df_dmr,
                           .vars = vars(contains(" ")),
                           .funs = funs(str_replace_all(string = .,
                                                        pattern = " ",
                                                        replacement = "_")))
colnames(df_dmr_renamed)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE_DECLARATION"   "MOIS_DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL_DE_QUARTIER" "OBJECTID"            "geo_shape"
## [16] "geo_point_2d"
```

-> On renomme toutes (`rename_at`) les colonnes qui contiennent un espace (`vars(contains(" "))`), par un appel de fonctions (`funs` ici) qui remplace tous (`str_replace_all()`) les espaces (`pattern = " "`) par des underscores (`replacement = "_"`).

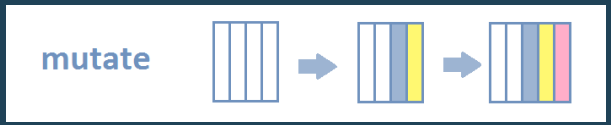
Renommer des colonnes avec `rename`

On peut à nouveau enchaîner les opérations pour disposer d'une chaîne de renommage claire et ré-utilisable :

```
df_dmr_renamed <- df_dmr %>%
  rename_at(.vars = vars(contains(" ")),
            .funs = funs(str_replace_all(string = ., pattern = " ", replacement = "_"))) %>%
  rename_all(funs(toupper(.)))
colnames(df_dmr_renamed)
```

```
##  [1] "TYPE"                "SOUSTYPE"            "ADRESSE"
##  [4] "CODE_POSTAL"         "VILLE"               "ARRONDISSEMENT"
##  [7] "DATEDECL"            "ANNEE_DECLARATION"   "MOIS_DECLARATION"
## [10] "NUMERO"              "PREFIXE"             "INTERVENANT"
## [13] "CONSEIL_DE_QUARTIER" "OBJECTID"            "GEO_SHAPE"
## [16] "GEO_POINT_2D"
```

Créer et modifier des colonnes avec `mutate`



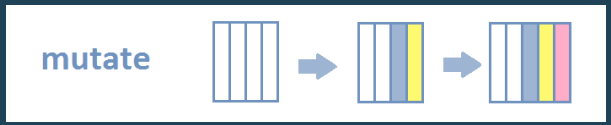
Pour créer de nouvelles variables et les ajouter au tableau de données on peut utiliser la fonction mutate().

Voici par exemple comment procéder si je souhaite créer une nouvelle variable, "DATE_DECLARATION", en combinant les variables ANNEE_DECLARATION et MOIS_DECLARATION à l'aide de la fonction paste() (concaténation)

```
df_dmr_renamed_muted <- df_dmr_renamed %>%
  mutate(DATE_DECLARATION = paste(ANNEE_DECLARATION, MOIS_DECLARATION, sep = "-"))
head(df_dmr_renamed_muted$DATE_DECLARATION)
```

```
## [1] "2017-12" "2017-11" "2017-12" "2017-12" "2017-12" "2017-10"
```

Créer et modifier des colonnes avec `mutate`



- On peut aussi modifier des variables existantes :

```
df_dmr_renamed_muted <- df_dmr_renamed %>%
  mutate(CODE_POSTAL = as.character(CODE_POSTAL))
```

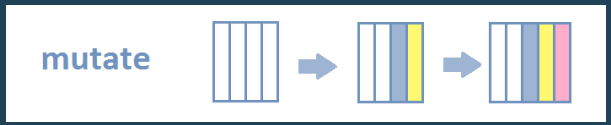
```
df_dmr_renamed %>% select(3:5)
```

```
## # A tibble: 343,642 x 3
##    ADRESSE                          CODE_POSTAL VILLE
##    <chr>                                  <int> <chr>
##  1 16 rue de l'orillon, 75011 PARIS       75011 Paris 11
##  2 15 rue joseph dijon, 75018 PARIS       75018 Paris 18
##  3 16 rue archereau, 75019 PARIS          75019 Paris 19
##  4 15 rue réaumur, 75003 PARIS            75003 Paris 3
##  5 24 rue custine, 75018 PARIS            75018 Paris 18
##  6 9 villa virginie, 75014 PARIS          75014 Paris 14
##  7 19 rue de mont-louis, 75011 PARIS      75011 Paris 11
##  8 3 avenue boudon, 75016 PARIS           75016 Paris 16
##  9 185b rue ordener, 75018 PARIS          75018 Paris 18
## 10 138 rue pelleport, 75020 PARIS         75020 Paris 20
## # ... with 343,632 more rows
```

```
df_dmr_renamed_muted %>% select(3:5)
```

```
## # A tibble: 343,642 x 3
##    ADRESSE                          CODE_POSTAL VILLE
##    <chr>                            <chr>       <chr>
##  1 16 rue de l'orillon, 75011 PARIS 75011       Paris 11
##  2 15 rue joseph dijon, 75018 PARIS 75018       Paris 18
##  3 16 rue archereau, 75019 PARIS    75019       Paris 19
##  4 15 rue réaumur, 75003 PARIS      75003       Paris 3
##  5 24 rue custine, 75018 PARIS      75018       Paris 18
##  6 9 villa virginie, 75014 PARIS    75014       Paris 14
##  7 19 rue de mont-louis, 75011 PARIS 75011      Paris 11
##  8 3 avenue boudon, 75016 PARIS     75016       Paris 16
##  9 185b rue ordener, 75018 PARIS    75018       Paris 18
## 10 138 rue pelleport, 75020 PARIS   75020       Paris 20
## # ... with 343,632 more rows
```

Créer et modifier des colonnes avec `mutate`

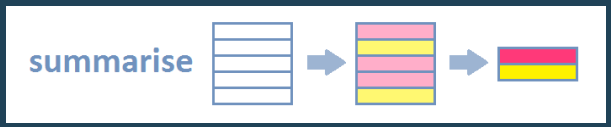


- Un exemple de discrétisation d'une variable continue, ici avec la fonction `cut()`

```
df_dmr_renamed_muted <- df_dmr_renamed_muted %>%
  mutate(TRIMESTRE_DECLARATION = cut(MOIS_DECLARATION,
                         breaks = c(0, 3, 6, 9, 12),
                         labels = c("Q1", "Q2", "Q3", "Q4")))
df_dmr_renamed_muted %>%
  select(MOIS_DECLARATION, TRIMESTRE_DECLARATION)
```

```
## # A tibble: 343,642 x 2
##    MOIS_DECLARATION TRIMESTRE_DECLARATION
##               <int> <fct>
##  1               12 Q4
##  2               11 Q4
##  3               12 Q4
##  4               12 Q4
##  5               12 Q4
##  6               10 Q4
##  7               12 Q4
##  8               12 Q4
##  9                7 Q3
## 10               12 Q4
## # ... with 343,632 more rows
```

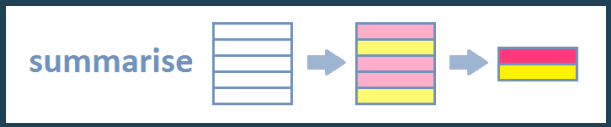
Résumer avec `group_by` et `summarise`



Comme dans les tableaux croisés dynamiques des tableurs, il peut être utile de résumer l'information contenue dans une colonne en fonction d'une agrégation spécifique.

- Avec `dplyr`, la syntaxe de groupage ressemble à celle du `SQL` : `GROUP BY`
- Une fois le groupage effectué, on peut réaliser l'opération d'agrégation : moyenne (`mean()`), somme (`sum()`), compte (`n()`) etc...

Résumer avec `group_by` et `summarise`

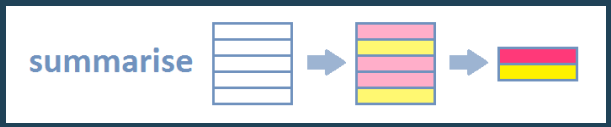


- Par exemple, on peut vouloir connaître le nombre de déclarations ayant été reportées chaque année :

```
df_dmr_renamed_muted %>%
group_by(ANNEE_DECLARATION) %>%
summarise(NbDeclarations = n()) %>%
arrange(ANNEE_DECLARATION)
```

```
## # A tibble: 7 x 2
##   ANNEE_DECLARATION NbDeclarations
##               <int>          <int>
## 1              2012          11931
## 2              2013          31503
## 3              2014          46873
## 4              2015          57767
## 5              2016          70167
## 6              2017          92229
## 7              2018          33172
```

Résumer avec group_by et summarise

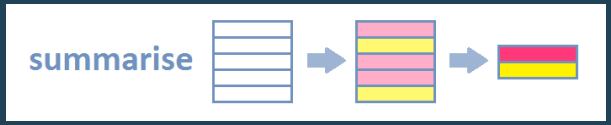


- On peut mener cette agrégation en fonction de plusieurs variables

```
df_dmr_renamed_muted %>%
  group_by(ANNEE_DECLARATION, TRIMESTRE_DECLARATION) %>%
  summarise(NbDeclarations = n()) %>%
  arrange(ANNEE_DECLARATION, TRIMESTRE_DECLARATION)
```

```
## # A tibble: 23 x 3
## # Groups:   ANNEE_DECLARATION [7]
##    ANNEE_DECLARATION TRIMESTRE_DECLARATION NbDeclarations
##                <int> <fct>                          <int>
##  1              2012 Q3                              1643
##  2              2012 Q4                             10288
##  3              2013 Q1                              9485
##  4              2013 Q2                              6261
##  5              2013 Q3                              8694
##  6              2013 Q4                              7063
##  7              2014 Q1                              8709
##  8              2014 Q2                             10577
##  9              2014 Q3                             14943
## 10              2014 Q4                             12644
## # ... with 13 more rows
```

Résumer avec `group_by` et `summarise`



- On peut aussi créer plusieurs résumés statistiques, et les enchaîner :

```
df_dmr_renamed_muted %>%
  group_by(ANNEE_DECLARATION, ARRONDISSEMENT) %>%
  summarise(NbDeclarations = n()) %>%
  group_by(ARRONDISSEMENT) %>%
  summarise(NbDeclarationTotal = sum(NbDeclarations),
            NbDeclarationAnnuel = mean(NbDeclarations)) %>%
  head()
```

```
## # A tibble: 6 x 3
##   ARRONDISSEMENT NbDeclarationTotal NbDeclarationAnnuel
##            <dbl>              <int>               <dbl>
## 1              1               3638                520.
## 2              2               5724                818.
## 3              3              10464               1495.
## 4              4               7871               1124.
## 5              5               8592               1227.
## 6              6               5882                840.
```

Les jointures

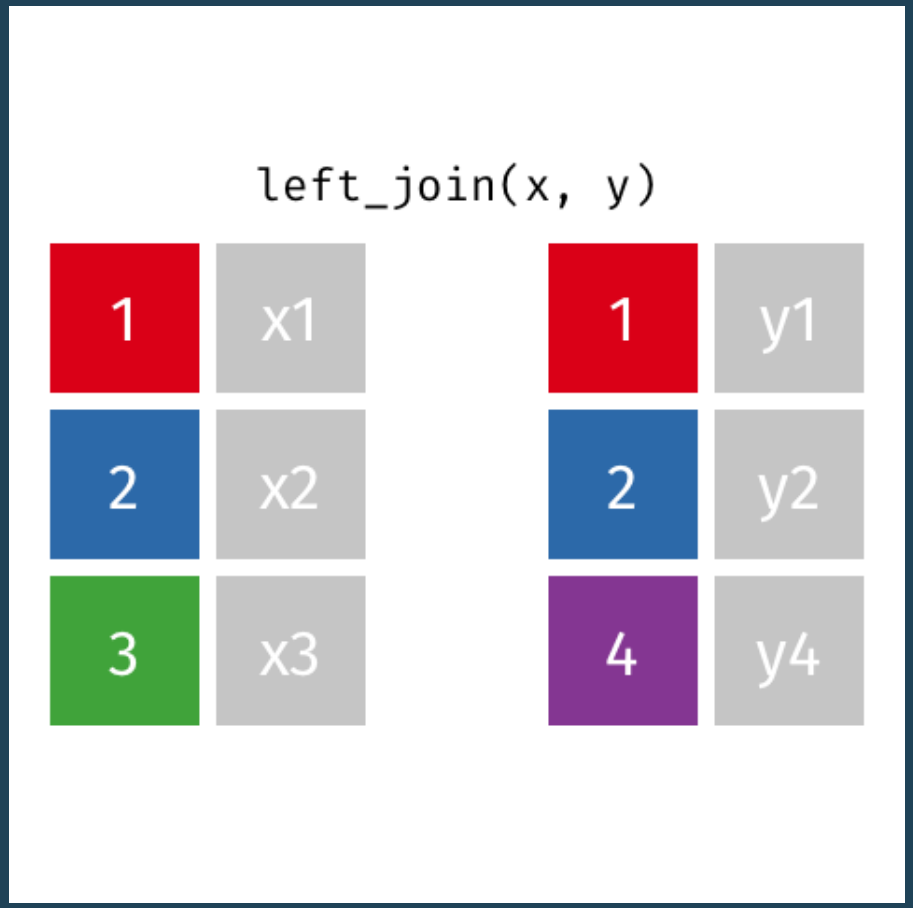
`dplyr` simplifie la réalisation de jointures entre des tableaux, en suivant une syntaxe proche du `SQL`. Les fonctions les plus utiles sont :

- `left_join()` : Jointure qui ajoute les colonnes de `y` à celles de `x`, et conservent toutes les lignes de `x`
- `inner_join()` : Jointure qui ajoutent les colonnes de `y` à celles de `x`, et ne conservent que les lignes de `x` et `y` qui sont présentes dans les deux tableaux
- `semi_join()` : Jointure qui ne conserve que les éléments de `x` présents dans `y`, et ne conserve que les colonnes de `x` : semblable à un filtre
- `union()` : Union des lignes uniques, avec suppression des doublons

- etc.

- Voir l'ensemble des opérations illustrées sur https://github.com/gadenbuie/tidy-animated-verbs

left_join()

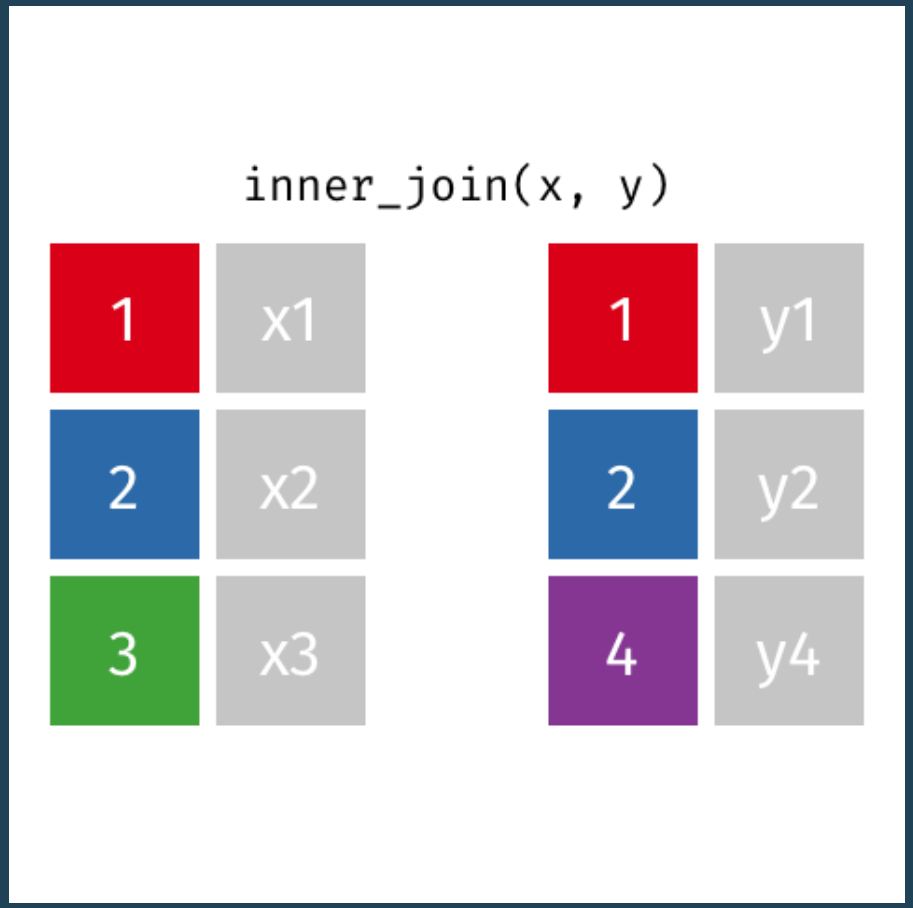


left_join()

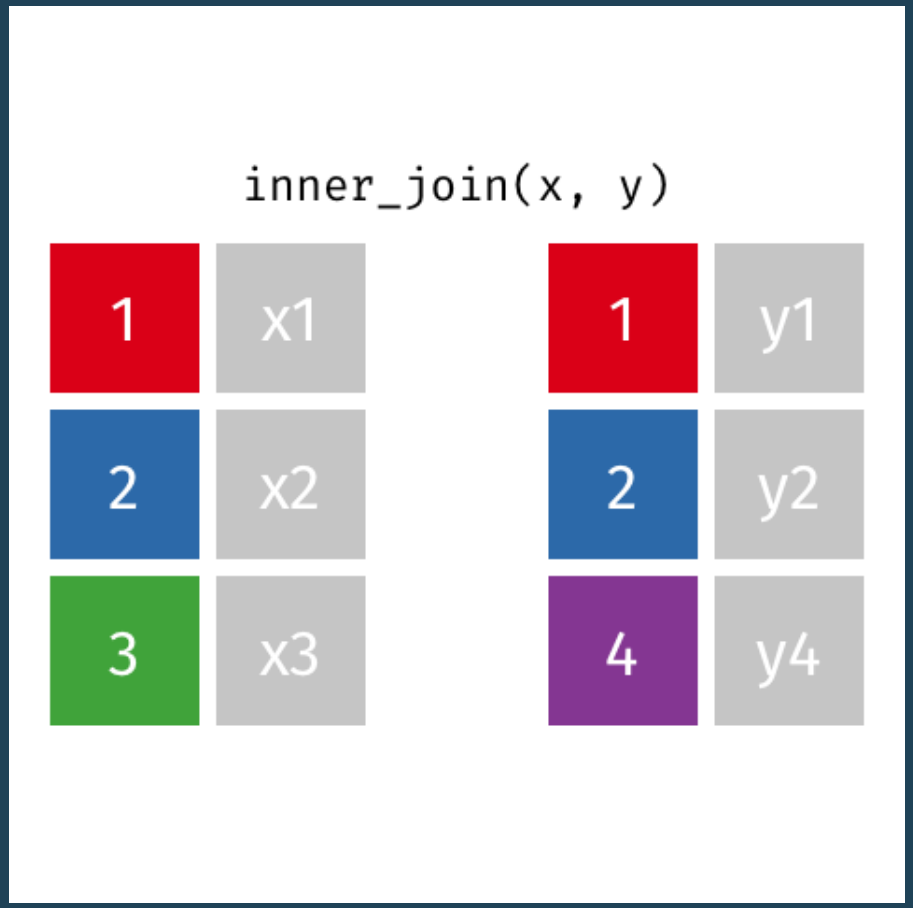
```
total_arrondissement <- df_dmr_renamed_muted %>%
  group_by(CODE_POSTAL) %>%
  summarise(NbArrondissement = n())

df_dmr_renamed_muted %>%
  group_by(ANNEE_DECLARATION, CODE_POSTAL) %>%
  summarise(NbDeclarations = n()) %>%
  left_join(y = total_arrondissement,
            by = "CODE_POSTAL") %>%
  mutate(Pct_Declaration_Annee =
           NbDeclarations / NbArrondissement * 100
         ) %>%
  select(CODE_POSTAL,
         ANNEE_DECLARATION,
         Pct_Declaration_Annee)
```

```
## # A tibble: 140 x 3
## # Groups:   ANNEE_DECLARATION [7]
##    CODE_POSTAL ANNEE_DECLARATION Pct_Declaration_Annee
##    <chr>                   <int>                 <dbl>
##  1 75001                    2012                 3.37
##  2 75002                    2012                 2.19
##  3 75003                    2012                 4.67
##  4 75004                    2012                 2.03
##  5 75005                    2012                 6.83
##  6 75006                    2012                 9.30
##  7 75007                    2012                 0.347
##  8 75008                    2012                 1.31
##  9 75009                    2012                 3.71
## 10 75010                    2012                 1.80
## # ... with 130 more rows
```

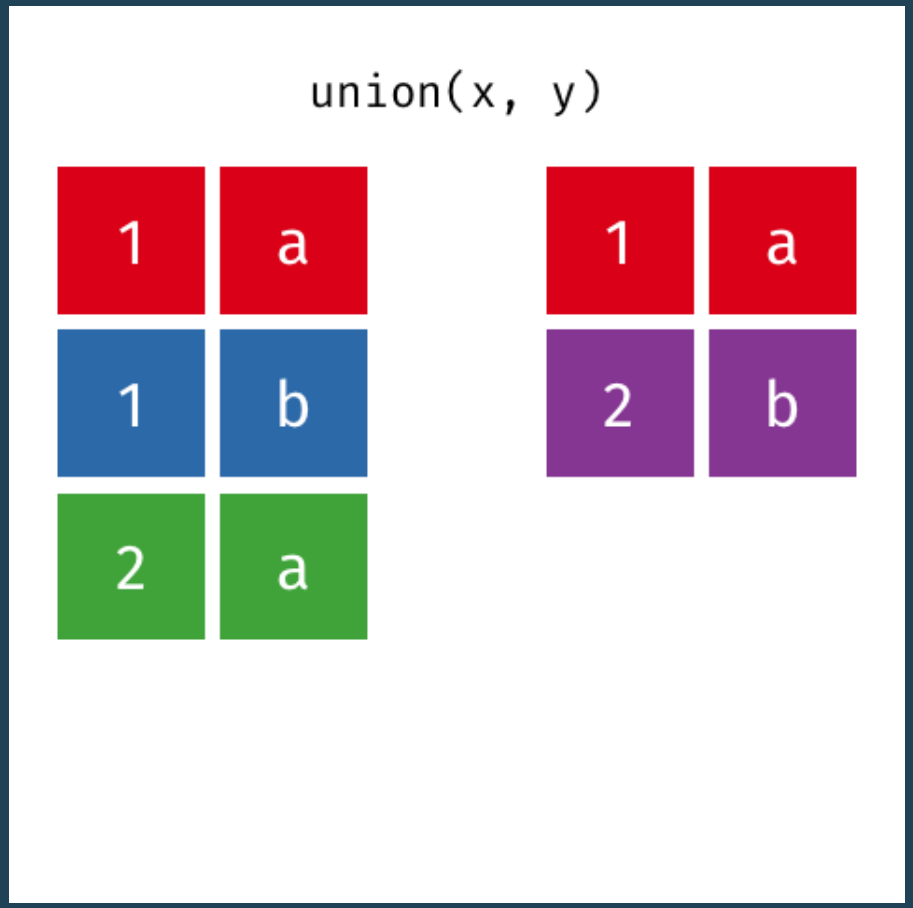
inner_join()



semi_join()

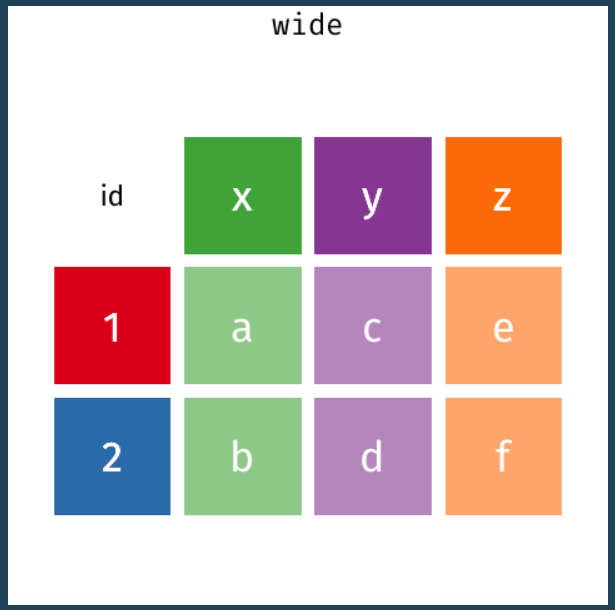


union()



Restructurer un tableau avec `tidyr`

Que ce soit pour des traitements ou pour restructurer les données en vue de leur (géo)visualisation, il est souvent utile de *passer de formats longs à des formats larges** et vice-versa



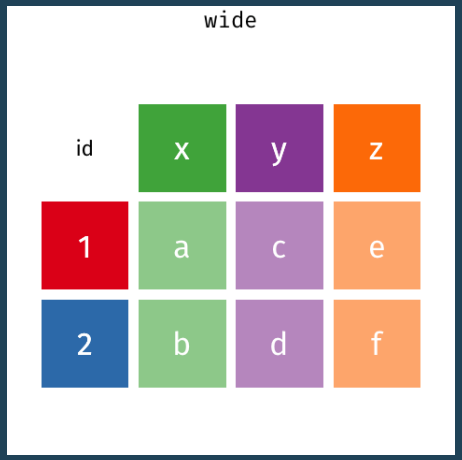
Tidy Animated Verbs - Garrick Aden-Buie - github.com/gadenbuie/tidy-animated-verbs

- On effectue ces opérations avec le *package* `tidyr`

```
library(tidyr)
```

Restructurer un tableau avec `tidyr`

Que ce soit pour des traitements ou pour restructurer les données en vue de leur (géo)visualisation, il est souvent utile de passer de formats longs à des formats larges et vice-versa



Tidy Animated Verbs - Garrick Aden-Buie - github.com/gadenbuie/tidy-animated-verbs

- De large à long : `gather()`
- De long à large : `spread()`

Restructurer un tableau avec `tidyr`

Un exemple sur les années : on a actuellement un tableau où les incidents sont déclarés les uns après les autres. On en a déjà fait un résumé par an et par arrondissement:

```
resume_annuel <- df_dmr_renamed_muted %>%
  group_by(ANNEE_DECLARATION, CODE_POSTAL) %>%
  summarise(NbDeclarations = n())
resume_annuel
```

```
## # A tibble: 140 x 3
## # Groups:   ANNEE_DECLARATION [?]
##    ANNEE_DECLARATION CODE_POSTAL NbDeclarations
##                <int> <chr>                <int>
##  1              2012 75001                  122
##  2              2012 75002                  124
##  3              2012 75003                  491
##  4              2012 75004                  160
##  5              2012 75005                  586
##  6              2012 75006                  548
##  7              2012 75007                   27
##  8              2012 75008                   64
##  9              2012 75009                  431
## 10              2012 75010                  521
## # ... with 130 more rows
```

- Le résultat est au format long
- => Pour le passer au format large : `spread()`

Restructurer un tableau avec tidyr

```
head(resume_annuel)
```

```
## # A tibble: 6 x 3
## # Groups:   ANNEE_DECLARATION [1]
##   ANNEE_DECLARATION CODE_POSTAL NbDeclarations
##               <int> <chr>                <int>
## 1              2012 75001                  122
## 2              2012 75002                  124
## 3              2012 75003                  491
## 4              2012 75004                  160
## 5              2012 75005                  586
## 6              2012 75006                  548
```

```
resume_annuel_large <- resume_annuel %>%
  spread(key = ANNEE_DECLARATION, value = NbDeclarations)
head(resume_annuel_large)
```

```
## # A tibble: 6 x 8
##   CODE_POSTAL `2012` `2013` `2014` `2015` `2016` `2017` `2018`
##   <chr>        <int>  <int>  <int>  <int>  <int>  <int>  <int>
## 1 75001          122    158    607    651    730    929    419
## 2 75002          124    286    997   1027   1192   1480    547
## 3 75003          491    566   2038   2286   1962   2465    699
## 4 75004          160    483   1473   1539   1494   1957    759
## 5 75005          586    591   1485   1985   1673   1730    534
## 6 75006          548    418    911   1250   1199   1207    359
```

-> On obtient un résultat plus lisible

Restructurer un tableau avec `tidyr`

Pour repasser au format long :

```
head(resume_annuel_large)
```

```
## # A tibble: 6 x 8
##   CODE_POSTAL `2012` `2013` `2014` `2015` `2016` `2017` `2018`
##   <chr>        <int>  <int>  <int>  <int>  <int>  <int>  <int>
## 1 75001          122    158    607    651    730    929    419
## 2 75002          124    286    997   1027   1192   1480    547
## 3 75003          491    566   2038   2286   1962   2465    699
## 4 75004          160    483   1473   1539   1494   1957    759
## 5 75005          586    591   1485   1985   1673   1730    534
## 6 75006          548    418    911   1250   1199   1207    359
```

```
resume_annuel_large %>% gather(key = ANNEE, NbIncidents, -CODE_POSTAL) %>% head()
```

```
## # A tibble: 6 x 3
##   CODE_POSTAL ANNEE NbIncidents
##   <chr>       <chr>       <int>
## 1 75001       2012          122
## 2 75002       2012          124
## 3 75003       2012          491
## 4 75004       2012          160
## 5 75005       2012          586
## 6 75006       2012          548
```

-> On regroupe toutes les colonnes sauf `CODE_POSTAL`, avec une nouvelle colonne `ANNEE` qui contiendra les anciens noms de colonnes, et une colonne `NbIncidents` qui contient les valeurs qui étaient renseignées dans chaque variable d'année.

Diviser des colonnes avec `tidyr`

- Le tableau de données contient une variable actuellement inexploitable : `GEO_POINT_2D`

```
df_dmr_renamed_muted %>% select(GEO_POINT_2D)
```

```
## # A tibble: 343,642 x 1
##    GEO_POINT_2D
##    <chr>
##  1 48.8698129958, 2.37522169904
##  2 48.8940050015, 2.34619070499
##  3 48.8897799964, 2.37295579974
##  4 48.8649799983, 2.35818549424
##  5 48.8886791451, 2.34735286463
##  6 48.8241899952, 2.32663080197
##  7 48.8584600029, 2.38904789509
##  8 48.84965983, 2.26992865126
##  9 48.8936878445, 2.3351792967
## 10 48.8712345434, 2.39900714303
## # ... with 343,632 more rows
```

- => Pour pouvoir utiliser cette information spatiale, il faut extraire les coordonnées de cette chaîne de caractères.

Diviser des colonnes avec `tidyr`

- On peut utiliser la fonction `separate()` du *package* `tidyr`, qui permet de diviser une colonne en plusieurs nouvelles variables, en fonction d'un séparateur à définir :

```
df_dmr_spatialise <- df_dmr_renamed_muted %>%
  separate(col = GEO_POINT_2D,
           into = c("Lat", "Long"), sep = ", ")

df_dmr_spatialise %>% select(Lat, Long) %>% head()
```

```
## # A tibble: 6 x 2
##   Lat           Long
##   <chr>         <chr>
## 1 48.8698129958 2.37522169904
## 2 48.8940050015 2.34619070499
## 3 48.8897799964 2.37295579974
## 4 48.8649799983 2.35818549424
## 5 48.8886791451 2.34735286463
## 6 48.8241899952 2.32663080197
```

- Et il ne reste plus qu'à convertir ces nouvelles colonnes en valeurs numériques

```
df_dmr_spatialise <- df_dmr_spatialise %>%
  mutate(Lat = as.numeric(Lat)) %>%
  mutate(Long = as.numeric(Long))

df_dmr_spatialise %>% select(Lat, Long) %>% head()
```

```
## # A tibble: 6 x 2
##     Lat  Long
##   <dbl> <dbl>
## 1  48.9  2.38
## 2  48.9  2.35
## 3  48.9  2.37
## 4  48.9  2.36
## 5  48.9  2.35
## 6  48.8  2.33
```

Enregistrer un tableau avec `readr`

Une fois les données manipulées, il faut bien sûr penser à les exporter/enregistrer.

- Comme pour la lecture, on utilise le *package* `readr` :
  - Au lieu d'utiliser `read_csv()`, `read_delim()` etc.
  - on fait appel aux fonctions `write_csv()`, `write_delim()` etc.

```
library(readr)
write_csv(df_dmr_spatialise,
          path = "data/dans_ma_rue_clean.csv")
```

N.B. On peut aussi enregistrer une variable dans un format spécifique à R, qui conservera le jeu de données "en l'état", au format `RDS` :

```
saveRDS(object = df_dmr_spatialise,
        file = "dans_ma_rue_clean.RDS")
```

- On pourra re-charger le jeu de données avec la fonction `ReadRDS`:

```
df_dmr_spatialise <- readRDS(file = "dans_ma_rue_clean.RDS")
```

`tibble`, `readr`, `dplyr`, `tidyr` etc. : le `tidyverse`

- On jongle avec de nombreux *packages* ici :
  - `readr` pour la lecture/écriture de tableaux, qui s'appuie sur
  - `tibble` pour le format de données ;
  - `dplyr` pour la manipulation de tableaux ;
  - `tidyr` pour la restructuration de tableaux ;
  - `stringr` pour la manipulation de chaînes de caractères ;
  - `magrittr`, sur lequel tous ces *packages* s'appuient pour gérer les pipes (`%>%`)
- Plutôt que d'avoir à charger les bons *packages* à chaque fois, on peut se contenter de charger un unique *package* qui les rassemble tous : `tidyverse` => plus besoin de réflechir à quel *package* mobiliser !

```
library(tidyverse)
tidyverse_packages()
```

```
##  [1] "broom"       "cli"         "crayon"      "dplyr"       "dbplyr"
##  [6] "forcats"     "ggplot2"     "haven"       "hms"         "httr"
## [11] "jsonlite"    "lubridate"   "magrittr"    "modelr"      "purrr"
## [16] "readr"       "readxl\n(>=" "reprex"      "rlang"       "rstudioapi"
## [21] "rvest"       "stringr"     "tibble"      "tidyr"       "xml2"
## [26] "tidyverse"
```

Récapitulatif

On souhaite reprendre, depuis le début, l'ensemble des étapes permettant d'analyser, trimestre par trimestre, pour chaque année, l'évolution relative (au total) du nombre de déclarations de chaque arrondissement

Préparation de la session

```
# On commence par supprimer l'ensemble des données créées jusque là :
rm(list = ls())
# On charge le package tidyverse
library(tidyverse)
```

Chargement des données

```
# Lecture du jeu de données
df_dmr <- read_delim(file = "data/dans-ma-rue.csv", delim = ";")
```

```
## Parsed with column specification:
## cols(
##   TYPE = col_character(),
##   SOUSTYPE = col_character(),
##   ADRESSE = col_character(),
##   CODE_POSTAL = col_integer(),
##   VILLE = col_character(),
##   ARRONDISSEMENT = col_double(),
##   DATEDECL = col_datetime(format = ""),
##   `ANNEE DECLARATION` = col_integer(),
##   `MOIS DECLARATION` = col_integer(),
##   NUMERO = col_double(),
##   PREFIXE = col_character(),
##   INTERVENANT = col_character(),
##   `CONSEIL DE QUARTIER` = col_character(),
##   OBJECTID = col_integer(),
##   geo_shape = col_character(),
##   geo_point_2d = col_character()
## )
```

Reformatage

```
df_dmr_clean <- df_dmr %>%
  rename_all(.funs = funs(str_replace_all(
    string = ., pattern = " ", replacement = "_"
    ))) %>%
  rename_all(.funs = funs(toupper)) %>%
  mutate(CODE_POSTAL = as.character(CODE_POSTAL)) %>%
  select(-VILLE, -DATEDECL, -(NUMERO:GEO_SHAPE)) %>%
  mutate(TRIMESTRE = cut(MOIS_DECLARATION,
                         breaks = seq(from = 0, to = 12, by = 3),
                         labels = paste("Q", 1:4, sep=""))) %>%
  mutate(TRIMESTRE = as.character(TRIMESTRE)) %>%
  mutate(TRIMESTRE_DECLARATION =paste(ANNEE_DECLARATION,
                                      TRIMESTRE,
                                      sep="-")) %>%
  arrange(TRIMESTRE_DECLARATION, CODE_POSTAL)

df_dmr_clean
```

```
## # A tibble: 343,642 x 10
##    TYPE  SOUSTYPE ADRESSE CODE_POSTAL ARRONDISSEMENT ANNEE_DECLARATI…
##    <chr> <chr>    <chr>   <chr>                <dbl>            <int>
##  1 Voir… Panneau… 210 ru… 75001                    1             2012
##  2 Obje… Planche… 227 Ru… 75001                    1             2012
##  3 Voir… Marquag… 220 Ru… 75001                    1             2012
##  4 Prop… Malprop… 210 ru… 75001                    1             2012
##  5 Voir… Marquag… 210 ru… 75001                    1             2012
##  6 Voir… Marquag… 210 ru… 75001                    1             2012
##  7 Prop… Malprop… 19-21 … 75004                    4             2012
##  8 Obje… Équipem… 1 rue … 75004                    4             2012
##  9 Voir… Marquag… 3 boul… 75004                    4             2012
## 10 Prop… Malprop… 19-21 … 75004                    4             2012
## # ... with 343,632 more rows, and 4 more variables:
## #   MOIS_DECLARATION <int>, GEO_POINT_2D <chr>, TRIMESTRE <chr>,
## #   TRIMESTRE_DECLARATION <chr>
```

Calcul des totaux

```
total_declaration <- df_dmr_clean %>%
  group_by(CODE_POSTAL) %>%
  summarise(NbTotal = n())

total_declaration
```

```
## # A tibble: 20 x 2
##    CODE_POSTAL NbTotal
##    <chr>         <int>
##  1 75001          3616
##  2 75002          5653
##  3 75003         10507
##  4 75004          7865
##  5 75005          8584
##  6 75006          5892
##  7 75007          7776
##  8 75008          4876
##  9 75009         11629
## 10 75010         28932
## 11 75011         42432
## 12 75012         18911
## 13 75013         14543
## 14 75014         10040
## 15 75015         25948
## 16 75016         17053
## 17 75017         26170
## 18 75018         43038
## 19 75019         23008
## 20 75020         27169
```

Préparation du jeu de données final

```
donnees_resumees <- df_dmr_clean %>%
  group_by(TRIMESTRE_DECLARATION, CODE_POSTAL) %>%
  summarise(NbLocal = n())

donnees_resumees
```

```
## # A tibble: 457 x 3
## # Groups:   TRIMESTRE_DECLARATION [?]
##    TRIMESTRE_DECLARATION CODE_POSTAL NbLocal
##    <chr>                 <chr>         <int>
##  1 2012-Q3               75001             6
##  2 2012-Q3               75004             7
##  3 2012-Q3               75005            13
##  4 2012-Q3               75007             1
##  5 2012-Q3               75008            14
##  6 2012-Q3               75009             4
##  7 2012-Q3               75010             6
##  8 2012-Q3               75011            41
##  9 2012-Q3               75012            74
## 10 2012-Q3               75013           730
## # ... with 447 more rows
```

```
# Le tableau fait mention de "Groups" restant suite au summarise :
# On enlève cet artefact avec la fonction ungroup()

donnees_resumees <- donnees_resumees %>%
  ungroup()
```

Jointure des deux jeux de données

```
donnees_finales <- donnees_resumees %>%
  full_join(y = total_declaration, by = "CODE_POSTAL") %>%
  mutate(TAUX_DECLARATION = NbLocal / NbTotal) %>%
  arrange(TRIMESTRE_DECLARATION, CODE_POSTAL)

donnees_finales
```

```
## # A tibble: 457 x 5
##    TRIMESTRE_DECLARATION CODE_POSTAL NbLocal NbTotal TAUX_DECLARATION
##    <chr>                 <chr>         <int>   <int>            <dbl>
##  1 2012-Q3               75001             6    3616         0.00166
##  2 2012-Q3               75004             7    7865         0.000890
##  3 2012-Q3               75005            13    8584         0.00151
##  4 2012-Q3               75007             1    7776         0.000129
##  5 2012-Q3               75008            14    4876         0.00287
##  6 2012-Q3               75009             4   11629         0.000344
##  7 2012-Q3               75010             6   28932         0.000207
##  8 2012-Q3               75011            41   42432         0.000966
##  9 2012-Q3               75012            74   18911         0.00391
## 10 2012-Q3               75013           730   14543         0.0502
## # ... with 447 more rows
```

Données manquantes

- Pendant certains trimestres, il n'y a pas eu d'incidents déclarés, et donc pas de lignes correspondantes.
- Pour que le jeu de données soit complet, on va ajouter ces éléments possibles, et les remplir par les taux correspondants (0%), avec la fonction `complete()` : on veut que toutes les combinaisons TRIMESTRE/CODE_POSTAL soient remplies :

```
donnees_finales_completes <- donnees_finales %>%
  complete(TRIMESTRE_DECLARATION, CODE_POSTAL)

donnees_finales_completes
```

```
## # A tibble: 460 x 5
##    TRIMESTRE_DECLARATION CODE_POSTAL NbLocal NbTotal TAUX_DECLARATION
##    <chr>                 <chr>         <int>   <int>            <dbl>
##  1 2012-Q3               75001             6    3616         0.00166
##  2 2012-Q3               75002            NA      NA        NA
##  3 2012-Q3               75003            NA      NA        NA
##  4 2012-Q3               75004             7    7865         0.000890
##  5 2012-Q3               75005            13    8584         0.00151
##  6 2012-Q3               75006            NA      NA        NA
##  7 2012-Q3               75007             1    7776         0.000129
##  8 2012-Q3               75008            14    4876         0.00287
##  9 2012-Q3               75009             4   11629         0.000344
## 10 2012-Q3               75010             6   28932         0.000207
## # ... with 450 more rows
```

Remplacement conditionnel

- Les données manquantes sont ajoutées, et remplies avec des `NA`
- On va remplacer ces NA, dans la colonnes `TAUX_DECLARATION`, par la valeur `0`, avec l'opérateur `if_else()` :

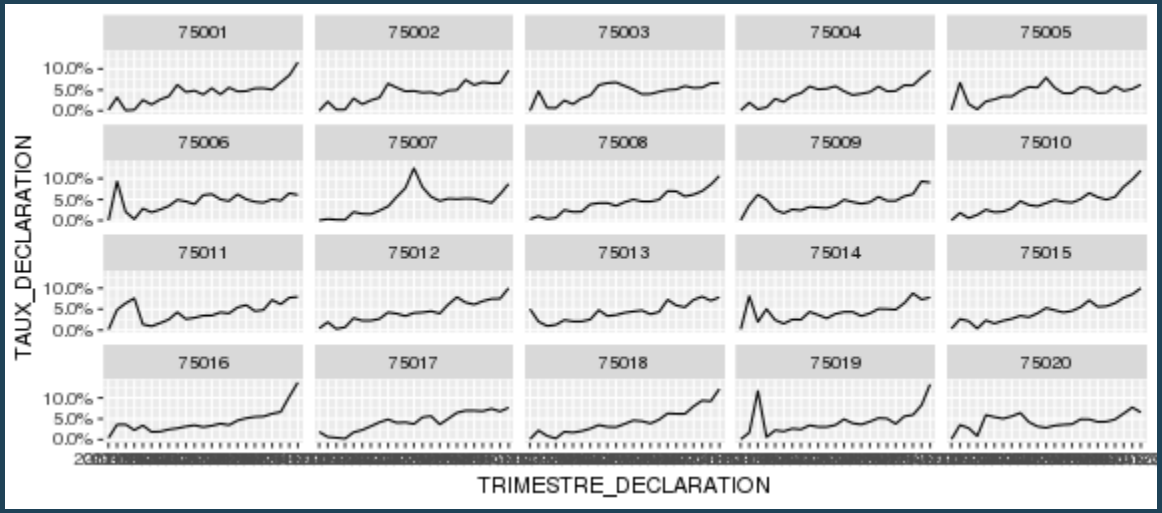
```
donnees_finales_completes <- donnees_finales_completes %>%
  mutate(TAUX_DECLARATION = if_else(condition = is.na(TAUX_DECLARATION),
                                    true = 0,
                                    false = TAUX_DECLARATION))

donnees_finales_completes
```

```
## # A tibble: 460 x 5
##    TRIMESTRE_DECLARATION CODE_POSTAL NbLocal NbTotal TAUX_DECLARATION
##    <chr>                 <chr>         <int>   <int>           <dbl>
##  1 2012-Q3               75001             6    3616         0.00166
##  2 2012-Q3               75002            NA      NA         0
##  3 2012-Q3               75003            NA      NA         0
##  4 2012-Q3               75004             7    7865         0.000890
##  5 2012-Q3               75005            13    8584         0.00151
##  6 2012-Q3               75006            NA      NA         0
##  7 2012-Q3               75007             1    7776         0.000129
##  8 2012-Q3               75008            14    4876         0.00287
##  9 2012-Q3               75009             4   11629         0.000344
## 10 2012-Q3               75010             6   28932         0.000207
## # ... with 450 more rows
```

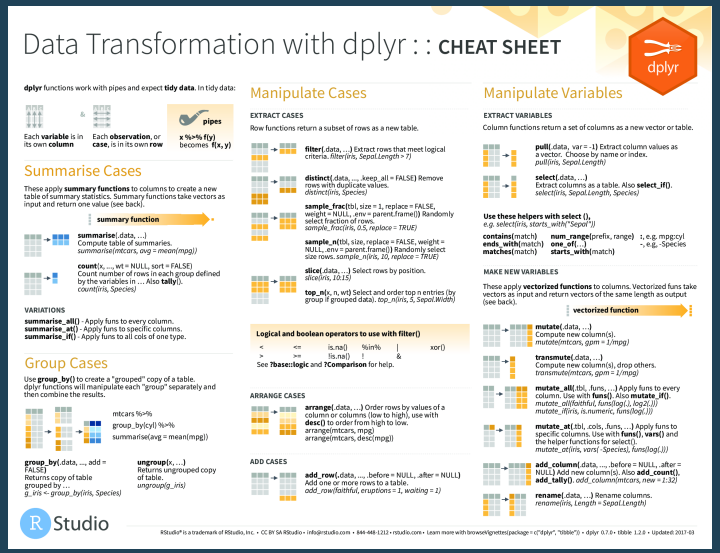
Réalisation d'un graphique synthétique

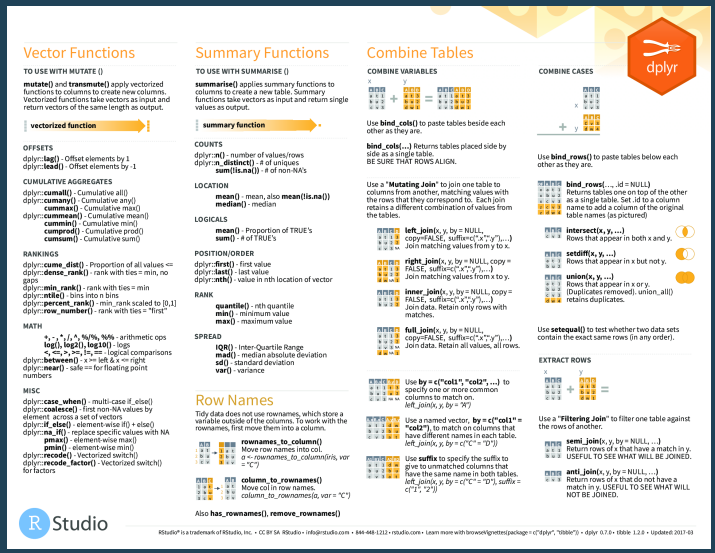
Explications au prochain cours

```
ggplot(donnees_finales_completes) +
  geom_line(mapping = aes(x = TRIMESTRE_DECLARATION,
                          y = TAUX_DECLARATION,
                          group = CODE_POSTAL)) +
  facet_wrap(~CODE_POSTAL, nrow = 4) +
  scale_y_continuous(labels = scales::percent)
```



Cheatsets





www.rstudio.com/resources/cheatsheets/#dplyr