5

La bibliothèque tidyverse

Le type tibble

La bibliothèque « **tidyverse** » contient la bibliothèque contient la bibliothèque « **tibble** ». Les types « **tibble** » de la bibliothèque du même nom sont des cadres de données qui modifient certains comportements pour faciliter le travail.

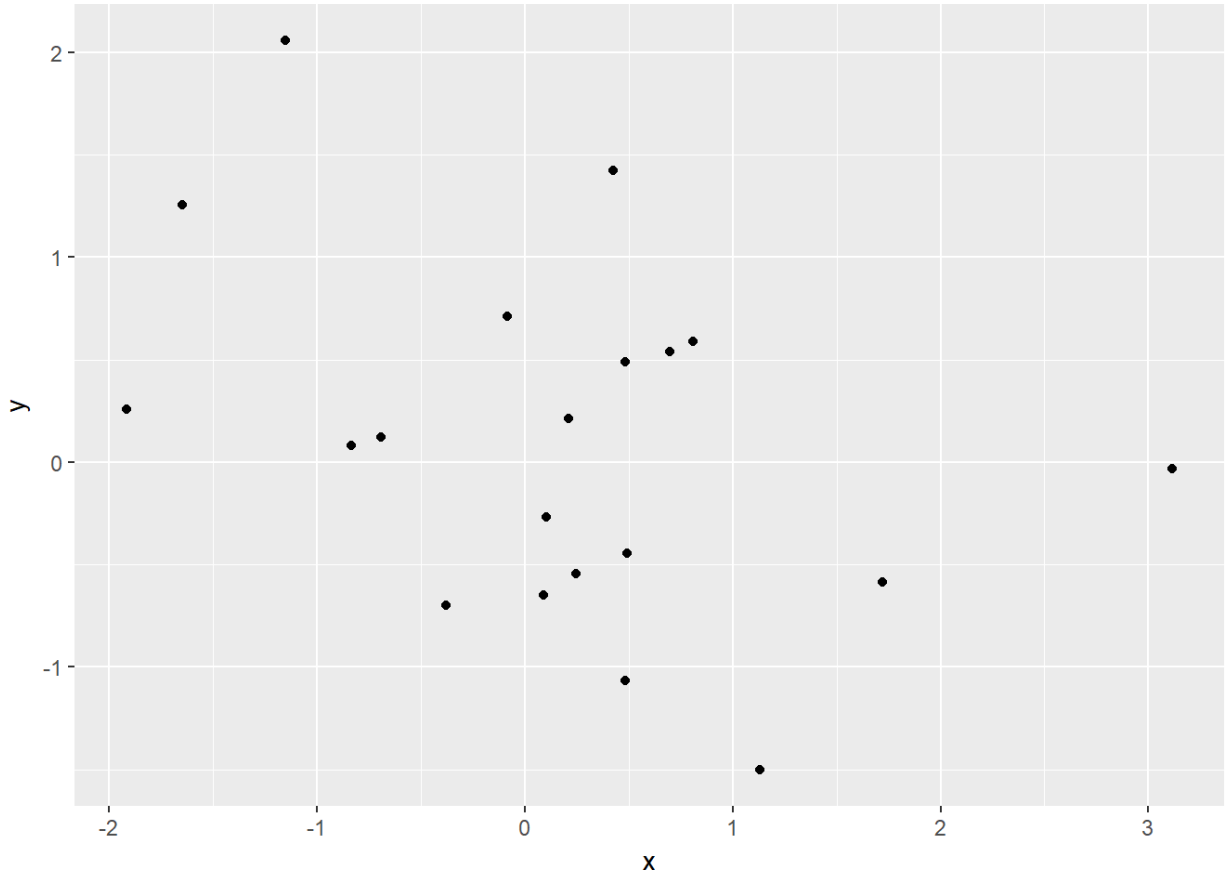
```
> as_tibble(iris)
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
 1          5.1         3.5          1.4         0.2  setosa
 2          4.9         3.0          1.4         0.2  setosa
 3          4.7         3.2          1.3         0.2  setosa
...
# ... with 140 more rows
> tibble( x = 1:5, y = 1, z = x ^ 2 + y)
# A tibble: 5 x 3
      x     y     z
  <int> <dbl> <dbl>
1     1     1     2
2     2     1     5
3     3     1    10
4     4     1    17
5     5     1    26
> tibble( `:)` = "smile", ` ` = "space", `2000` = "number" )
# A tibble: 1 x 3
   `:)`   ` ` `2000`
  <chr> <chr>  <chr>
1 smile space number
> tbd <- tribble(
+     ~x, ~y, ~z,
+     #--|--|----
+     "a", 2, 3.6,
+     "b", 1, 8.5)
> class(tbd)
[1] "tbl_df"     "tbl"        "data.frame"
> tbd
# A tibble: 2 x 3
      x     y     z
  <chr> <dbl> <dbl>
1     a     2   3.6
2     b     1   8.5
```

La méthode d'impression montre que les 10 premières lignes et toutes les colonnes qui correspondent à l'écran. Cela facilite le travail avec de grandes données. En plus de son nom, chaque colonne indique son type, une caractéristique intéressante empruntée à « **str** ». La conversion en « **data.frame** » est toujours possible à l'aide de la fontion « **as.data.frame** ».

%>%

La fonction pipe « **%>%** » de la bibliothèque « **magrittr** » est un outil puissant pour exprimer clairement une séquence d'opérations multiples.

```
> rnorm(40) %>%
+     matrix(ncol = 2,dimnames =
+                  list(c(letters[1:20]),c("x", "y"))) %>%
+     as.data.frame %>%
+     as_tibble %>%
+     ggplot() +
+          geom_point(mapping = aes(x = x, y = y))
```

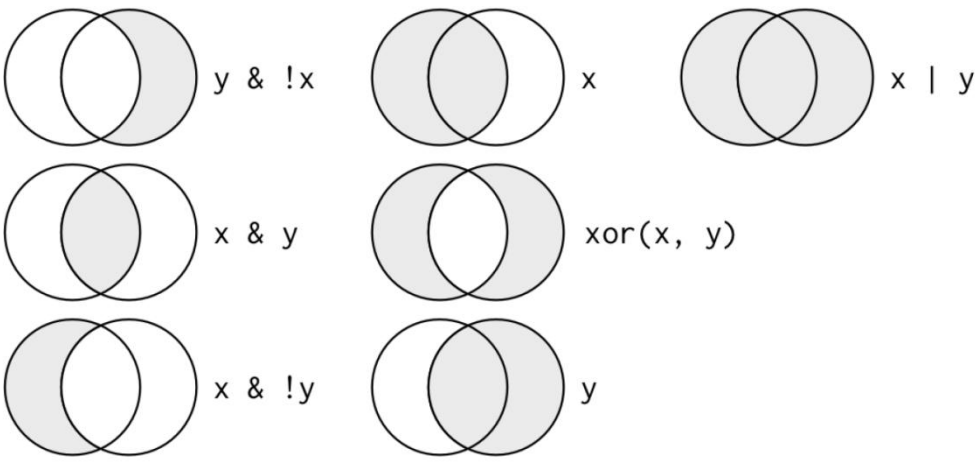


La fonction filter

La fonction « **filter** » permet de créer des sous-ensembles d'observations en fonction de leurs valeurs. Le premier argument est le nom du jeu de données et les suivants sont les expressions qui filtrent le jeu de données.

```
> library(tidyverse)
> (dtib <- filter(villes,Altitude > 300))
# A tibble: 8 x 4
  Latitude Longitude Altitude               Nom
     <dbl>     <dbl>    <int>             <chr>
1 48.58100  5.959833      336        Nancy-Ochey
2 45.86117  1.175000      402 Limoges-Bellegarde
3 45.78683  3.149333      331        Clermont-Fd
4 45.07450  3.764000      833      Le Puy-Loudes
5 44.11850  3.019500      712             Millau
6 44.56567  6.502333      871             Embrun
7 43.18800  0.000000      360       Tarbes-Ossun
8 43.00533  1.106833      414          St Girons
> (dtib <- filter(villes,Altitude > 300, Latitude > 44.7))
# A tibble: 4 x 4
  Latitude Longitude Altitude               Nom
     <dbl>     <dbl>    <int>             <chr>
1 48.58100  5.959833      336        Nancy-Ochey
2 45.86117  1.175000      402 Limoges-Bellegarde
3 45.78683  3.149333      331        Clermont-Fd
4 45.07450  3.764000      833      Le Puy-Loudes
> (dtib <- filter(villes,Altitude>300,Latitude>44.7,Longitude>5))
# A tibble: 1 x 4
  Latitude Longitude Altitude         Nom
     <dbl>     <dbl>    <int>       <chr>
1   48.581  5.959833      336 Nancy-Ochey
```

Vous pouvez utiliser tous les opérateurs de comparaison **R** fournit la suite standard : **>**, **>=**, **<**, **<=**, **!=** et **==**. Les arguments multiples pour la fonction « **filtrer** » sont combinés avec « **&** », ainsi chaque expression doit être vraie pour qu'une ligne soit incluse dans la sortie. Mais vous pouvez utiliser n'importe quel opérateur logique la figure suivante montre l'ensemble complet des opérations booléennes.



La fonction arrange

La fonction permet de réorganiser les enregistrements en effectuant un ordonnancement cotissant ou décroissant sur l'ensemble des colonnes passent en argument.

```
> (dtib <- arrange(villes,Nom))
# A tibble: 42 x 4
   Latitude Longitude Altitude                Nom
      <dbl>     <dbl>    <int>              <chr>
 1 50.13600  1.834000       69          Abbeville
 2 41.91800  8.792667        5            Ajaccio
 3 48.44550  0.110167      143            Alencon
 4 47.61433  7.510000      263      Bale-Mulhouse
 5 42.54067  9.485167       10             Bastia
 6 47.29433 -3.218333       34 Belle Ile-Le Talut
 7 44.83067 -0.691333       47  Bordeaux-Merignac
 8 47.05917  2.359833      161            Bourges
 9 48.44417 -4.412000       94     Brest-Guipavas
10 49.18000 -0.456167       67     Caen-Carpiquet
# ... with 32 more rows
> villes$Altitude <- villes %>% .$Altitude %>% round(-2)
> (dtib <- arrange(villes,desc(Altitude), Nom))
# A tibble: 42 x 4
   Latitude Longitude Altitude                Nom
      <dbl>     <dbl>    <dbl>              <chr>
 1 44.56567  6.502333      900             Embrun
 2 45.07450  3.764000      800      Le Puy-Loudes
 3 44.11850  3.019500      700             Millau
 4 45.86117  1.175000      400 Limoges-Bellegarde
 5 43.00533  1.106833      400          St Girons
 6 43.18800  0.000000      400       Tarbes-Ossun
 7 47.61433  7.510000      300      Bale-Mulhouse
 8 45.78683  3.149333      300        Clermont-Fd
 9 44.74500  1.396667      300            Gourdon
10 48.58100  5.959833      300        Nancy-Ochey
# ... with 32 more rows
```

La fonction select

Il n'est pas rare d'avoir des jeux de données avec des centaines voire des milliers de variables. Dans ce cas, le premier défi est de choisir les variables par lesquelles vous êtes réellement intéressé. La fonction « select » vous permet de faire ce choix et de créer un sous-ensemble utile en utilisant des opérations basées sur le nom des variables.

```
> load(file = "tab.coise.all.meteo.station.mois.RData")
> meteo$Ville <- row.names(meteo)
> row.names(meteo) <- NULL
> (dtib <- as_tibble(select(meteo,Ville, Latitude:Altitude,
+                           Temperature_1,Humidite_1)))
# A tibble: 42 x 6
              Ville Latitude Longitude Altitude Temperature_1 Humidite_1
              <chr>    <dbl>     <dbl>    <int>         <dbl>      <dbl>
 1         Abbeville 50.13600  1.834000       69      277.4198   76.51099
 2     Lille-Lesquin 50.57000  3.097500       47      276.9457   73.70795
 3   Pte De La Hague 49.72517 -1.939833        6      281.0448   72.55805
 4    Caen-Carpiquet 49.18000 -0.456167       67      278.5307   75.37260
 5        Rouen-Boos 49.38300  1.181667      151      277.1021   75.27007
 6      Reims-Prunay 49.20967  4.155333       95      276.4751   75.00737
 7    Brest-Guipavas 48.44417 -4.412000       94      280.2434   74.40114
 8       Ploumanac'h 48.82583 -3.473167       55      280.9442   73.57597
 9 Rennes-St Jacques 48.06883 -1.734000       36      279.0961   75.25467
10           Alencon 48.44550  0.110167      143      277.5210   75.25919
# ... with 32 more rows
> (dtib <- as_tibble(select(meteo,-(1:84))))
# A tibble: 42 x 4
   Latitude Longitude Altitude             Ville
      <dbl>     <dbl>    <int>             <chr>
 1 50.13600  1.834000       69         Abbeville
 2 50.57000  3.097500       47     Lille-Lesquin
 3 49.72517 -1.939833        6   Pte De La Hague
 4 49.18000 -0.456167       67    Caen-Carpiquet
 5 49.38300  1.181667      151        Rouen-Boos
 6 49.20967  4.155333       95      Reims-Prunay
 7 48.44417 -4.412000       94    Brest-Guipavas
 8 48.82583 -3.473167       55       Ploumanac'h
 9 48.06883 -1.734000       36 Rennes-St Jacques
10 48.44550  0.110167      143           Alencon
# ... with 32 more rows
> meteo %>%
+   select(-(1:84)) %>%
+   select(Ville,Altitude,Latitude,Longitude) %>%
+   as_tibble()
# A tibble: 42 x 4
              Ville Altitude Latitude Longitude
              <chr>    <int>    <dbl>     <dbl>
 1         Abbeville       69 50.13600  1.834000
 2     Lille-Lesquin       47 50.57000  3.097500
 3   Pte De La Hague        6 49.72517 -1.939833
 4    Caen-Carpiquet       67 49.18000 -0.456167
 5        Rouen-Boos      151 49.38300  1.181667
```

```
 6      Reims-Prunay       95 49.20967  4.155333
 7    Brest-Guipavas       94 48.44417 -4.412000
 8       Ploumanac'h       55 48.82583 -3.473167
 9 Rennes-St Jacques       36 48.06883 -1.734000
10           Alencon      143 48.44550  0.110167
# ... with 32 more rows
```

Il existe plusieurs fonctions permettent d'augmenter la puissance de la sélection des colonnes :

- **starts_with ("abc")** : sélectionne les colonnes qui commencent par **"abc"**.
- **ends_with ("xyz")** : sélectionne les colonnes qui se terminent par **"xyz"**.
- **contains("ijk")** : sélectionne les colonnes contenant **"ijk"**.
- **matches("(.)\\1")** : sélectionne les colonnes qui correspondent à une expression régulière.
- **num_range ("x", 1: 3)** : sélectionne les colonnes correspondant à **x1**, **x2** et **x3**.
- **one_of("nom",...) :** sélectionne les colonnes dans la liste.
- **everything() :** toutes les colonnes.

```
> meteo %>%
+  select(Ville,ends_with("_1")) %>%
+  select(Ville,contains("Vent")) %>%
+  as_tibble()
# A tibble: 42 x 3
               Ville DirectionVent_1 VitesseVent_1
               <chr>           <dbl>         <dbl>
 1         Abbeville        17.10827     117.76696
 2     Lille-Lesquin        18.54223     115.76911
 3   Pte De La Hague        18.94434     109.47090
 4    Caen-Carpiquet        17.17797     116.59325
 5        Rouen-Boos        17.64855     117.43879
 6      Reims-Prunay        16.10138      89.39263
 7    Brest-Guipavas        18.96726     113.17549
 8       Ploumanac'h        18.58362     123.55899
 9 Rennes-St Jacques        17.03901     103.85014
10           Alencon        18.77879      99.76567
# ... with 32 more rows
```

slice

Il est possible de sélectionner les lignes selon leur position à l'aide de la fonction « **slice** ».

```
> meteo %>% ungroup() %>% select(Nom,T.Janv:T.Mai)
# A tibble: 41 x 6
                 Nom T.Janv T.Fevr T.Mars T.Avr T.Mai
               <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
 1          Abbeville   5.69   5.52   6.13  8.70 13.38
 2            Ajaccio  10.43  10.79  11.19 14.69 16.90
 3            Alencon   5.23   5.67   6.09  8.31 13.34
```

```
 4      Bale-Mulhouse   3.71   5.15   5.42  9.60 14.15
 5             Bastia  10.41  11.15  11.61 14.66 17.43
 6 Belle Ile-Le Talut  10.11   8.95   8.65 10.61 14.09
 7  Bordeaux-Merignac   8.77   8.48   9.38 11.71 15.92
 8            Bourges   5.21   5.72   6.44  9.31 13.51
 9     Brest-Guipavas   8.45   7.36   7.32  8.90 12.89
10     Caen-Carpiquet   6.20   6.33   6.40  8.39 13.15
# ... with 31 more rows
> meteo %>% ungroup() %>% select(Nom,T.Janv:T.Mai) %>% slice(3:8)
# A tibble: 6 x 6
                Nom T.Janv T.Fevr T.Mars T.Avr T.Mai
              <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
1           Alencon   5.23   5.67   6.09  8.31 13.34
2      Bale-Mulhouse   3.71   5.15   5.42  9.60 14.15
3             Bastia  10.41  11.15  11.61 14.66 17.43
4 Belle Ile-Le Talut  10.11   8.95   8.65 10.61 14.09
5  Bordeaux-Merignac   8.77   8.48   9.38 11.71 15.92
6            Bourges   5.21   5.72   6.44  9.31 13.51
```

top_n

Il est possible de sélectionner uniquement les premiers n observations ou les n premiers groups si les données sont groupées.

```
> meteo %>% ungroup() %>% select(Nom,T.Janv:T.Mai) %>% top_n(3,Nom)
# A tibble: 3 x 6
              Nom T.Janv T.Fevr T.Mars T.Avr T.Mai
            <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
1 Toulouse-Blagnac   8.37   8.06   9.22 12.52 15.47
2           Tours   6.06   6.38   7.07  9.50 13.89
3  Troyes-Barberey   4.77   5.65   5.80  9.46 13.71
> meteo %>% ungroup() %>% select(Nom,T.Janv:T.Mai) %>%
+     top_n(3,T.Janv)
# A tibble: 3 x 6
                Nom T.Janv T.Fevr T.Mars T.Avr T.Mai
              <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
1           Ajaccio  10.43  10.79  11.19 14.69 16.90
2            Bastia  10.41  11.15  11.61 14.66 17.43
3 Belle Ile-Le Talut  10.11   8.95   8.65 10.61 14.09
> meteo %>% ungroup() %>% select(Nom,T.Janv:T.Mai) %>%
+     top_n(3)
Selecting by T.Mai
# A tibble: 3 x 6
       Nom T.Janv T.Fevr T.Mars T.Avr T.Mai
     <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
1    Bastia  10.41  11.15  11.61 14.66 17.43
2 Marignane   9.14   9.78  10.60 14.66 17.73
3      Nice   9.71  10.67  11.67 15.40 17.64
```

La fonction distinct

La fonction « **distinct** » permet d'extraire uniquement les enregistrements uniques du jeu de données.

```
> meteo %>% select (Nom, Annee, Mois, Jour)
# A tibble: 121,730 x 4
                Nom Annee  Mois  Jour
 *            <chr> <int> <int> <int>
 1         Abbeville  2016     1     1
 2     Lille-Lesquin  2016     1     1
 3   Pte De La Hague  2016     1     1
 4        Rouen-Boos  2016     1     1
 5      Reims-Prunay  2016     1     1
 6    Brest-Guipavas  2016     1     1
 7       Ploumanac'h  2016     1     1
 8 Rennes-St Jacques  2016     1     1
 9             Orly  2016     1     1
10   Troyes-Barberey  2016     1     1
# ... with 121,720 more rows
> meteo %>% select (Nom, Annee, Mois, Jour) %>% distinct()
# A tibble: 15,304 x 4
                Nom Annee  Mois  Jour
              <chr> <int> <int> <int>
 1         Abbeville  2016     1     1
 2     Lille-Lesquin  2016     1     1
 3   Pte De La Hague  2016     1     1
 4        Rouen-Boos  2016     1     1
 5      Reims-Prunay  2016     1     1
 6    Brest-Guipavas  2016     1     1
 7       Ploumanac'h  2016     1     1
 8 Rennes-St Jacques  2016     1     1
 9             Orly  2016     1     1
10   Troyes-Barberey  2016     1     1
# ... with 15,294 more rows
> meteo %>% select (Nom) %>% distinct()
# A tibble: 42 x 1
                Nom
              <chr>
 1         Abbeville
 2     Lille-Lesquin
 3   Pte De La Hague
 4        Rouen-Boos
 5      Reims-Prunay
 6    Brest-Guipavas
 7       Ploumanac'h
 8 Rennes-St Jacques
 9             Orly
10   Troyes-Barberey
# ... with 32 more rows
```

La fonction mutate

En plus de sélectionner des ensembles de colonnes existantes, il est souvent utile d'ajouter une ou plusieurs nouvelles colonnes qui sont calculées à partir des colonnes existantes. La fonction **« mutate »** ajoute des nouvelles colonnes à la fin de votre jeu de données. Rappelez-vous que lorsque vous êtes dans **RStudio**, la façon la plus simple de voir toutes les colonnes est d'utiliser la fonction **« View ».**

```
> meteo %>%
+   select(Ville,num_range("Temperature_",c(1:2,12))) %>%
+   select(Ville,DecembreK=Temperature_12,
+                JanvierK=Temperature_1,
+                FevrierK=Temperature_2) %>%
+   mutate(HiverMK   = round((DecembreK + JanvierK + FevrierK)/3,2),
+          DecembreC = round(DecembreK - 273.15,2),
+          ReductionHiver  = round((HiverMK - mean(HiverMK))
+                                  / sd(HiverMK),2))  %>%
+  as_tibble()
# A tibble: 42 x 7
              Ville DecembreK JanvierK FevrierK HiverMK DecembreC ReductionHiver
              <chr>     <dbl>    <dbl>    <dbl>   <dbl>     <dbl>          <dbl>
 1         Abbeville  278.0680 277.4198 278.1130  277.87      4.92          -0.47
 2     Lille-Lesquin  277.6721 276.9457 277.8775  277.50      4.52          -0.64
 3   Pte De La Hague  282.1112 281.0448 280.8068  281.32      8.96           1.10
 4    Caen-Carpiquet  279.1076 278.5307 279.0905  278.91      5.96           0.00
 5        Rouen-Boos  277.6144 277.1021 277.8138  277.51      4.46          -0.64
 6      Reims-Prunay  278.3470 276.4751 277.6049  277.48      5.20          -0.65
 7    Brest-Guipavas  280.7902 280.2434 280.3027  280.45      7.64           0.71
 8       Ploumanac'h  281.6388 280.9442 280.9305  281.17      8.49           1.04
 9 Rennes-St Jacques  279.4583 279.0961 279.6072  279.39      6.31           0.22
10           Alencon  277.9457 277.5210 278.1968  277.89      4.80          -0.47
# ... with 32 more rows
```

transmute

Si vous souhaitez seulement conserver les nouvelles variables vous devez utiliser la fonction « **transmute** ».

```
> meteo %>%
+     select(Ville,num_range("Temperature_",c(1:2,12))) %>%
+     select(Ville,JanvierK=Temperature_1) %>%
+     transmute(JanvierC = round(JanvierK - 273.15,2))  %>%
+     as_tibble()
# A tibble: 42 x 1
   JanvierC
      <dbl>
 1     4.27
 2     3.80
 3     7.89
 4     5.38
 5     3.95
 6     3.33
 7     7.09
 8     7.79
 9     5.95
10     4.37
# ... with 32 more rows
```

Les fonctions utiles de création

lead et lag

Les fonctions « **lead** » et « **lag** » vous permettent de se référencer les valeurs de lignes suivantes et respectivement précédentes. Cela vous permet de calculer les différences ou de trouver lorsque les valeurs qui changent.

```
> library(lubridate)
> airQuality %>%
+   mutate(DateHeure= paste(Date,Time),
+          DateHeure= as.POSIXct(strptime(DateHeure,
+                                '%d/%m/%Y %H.%M.%S')))  %>%
+   select(DateHeure, AH) %>%
+   filter(year(DateHeure) == 2004 & month(DateHeure) == 3)%>%
+   mutate(flagAH  = lag(AH),
+          fleadAH = lead(AH),
+          fminAH  = min(AH),
+          fmaxAH  = max(AH),
+          fsdAH   = sd(AH))%>%
+ as_tibble()
# A tibble: 509 x 7
             DateHeure     AH flagAH fleadAH fminAH fmaxAH     fsdAH
                <dttm>  <dbl>  <dbl>   <dbl>  <dbl>  <dbl>     <dbl>
 1 2004-03-10 18:00:00 0.7578     NA  0.7255 0.4023 1.0945 0.1400445
 2 2004-03-10 19:00:00 0.7255 0.7578  0.7502 0.4023 1.0945 0.1400445
 3 2004-03-10 20:00:00 0.7502 0.7255  0.7867 0.4023 1.0945 0.1400445
 4 2004-03-10 21:00:00 0.7867 0.7502  0.7888 0.4023 1.0945 0.1400445
 5 2004-03-10 22:00:00 0.7888 0.7867  0.7848 0.4023 1.0945 0.1400445
 6 2004-03-10 23:00:00 0.7848 0.7888  0.7603 0.4023 1.0945 0.1400445
 7 2004-03-11 00:00:00 0.7603 0.7848  0.7702 0.4023 1.0945 0.1400445
 8 2004-03-11 01:00:00 0.7702 0.7603  0.7648 0.4023 1.0945 0.1400445
 9 2004-03-11 02:00:00 0.7648 0.7702  0.7517 0.4023 1.0945 0.1400445
10 2004-03-11 03:00:00 0.7517 0.7648  0.7465 0.4023 1.0945 0.1400445
# ... with 499 more rows
```

Agrégats cumulatifs et glissants

Les fonctions « **cumsum** », « **cumprod** », « **cummin** », « **cummax** » de la bibliothèque « **dplyr** » permettent d'effectuer des calculs dans une fenêtre qui commence à la première ligne et suit la ligne courante.

```
> airQuality %>%
+   mutate(DateHeure= paste(Date,Time),
+          DateHeure= as.POSIXct(
+                       strptime(DateHeure,'%d/%m/%Y %H.%M.%S')))  %>%
+   select(DateHeure, AH) %>%
+   filter(year(DateHeure) == 2004 & month(DateHeure) == 3)%>%
+   mutate(fsum    = cumsum(AH),
+          fprod   = cumprod(AH),
+          fmin    = cummin(AH),
+          fmax    = cummax(AH),
```

```
+           fmean   = cummean(AH))%>%
+ as_tibble()
# A tibble: 509 x 7
             DateHeure     AH   fsum      fprod   fmin   fmax     fmean
                <dttm>  <dbl>  <dbl>      <dbl>  <dbl>  <dbl>     <dbl>
 1 2004-03-10 18:00:00 0.7578 0.7578 0.75780000 0.7578 0.7578 0.7578000
 2 2004-03-10 19:00:00 0.7255 1.4833 0.54978390 0.7255 0.7578 0.7416500
 3 2004-03-10 20:00:00 0.7502 2.2335 0.41244788 0.7255 0.7578 0.7445000
 4 2004-03-10 21:00:00 0.7867 3.0202 0.32447275 0.7255 0.7867 0.7550500
 5 2004-03-10 22:00:00 0.7888 3.8090 0.25594410 0.7255 0.7888 0.7618000
 6 2004-03-10 23:00:00 0.7848 4.5938 0.20086493 0.7255 0.7888 0.7656333
 7 2004-03-11 00:00:00 0.7603 5.3541 0.15271761 0.7255 0.7888 0.7648714
 8 2004-03-11 01:00:00 0.7702 6.1243 0.11762310 0.7255 0.7888 0.7655375
 9 2004-03-11 02:00:00 0.7648 6.8891 0.08995815 0.7255 0.7888 0.7654556
10 2004-03-11 03:00:00 0.7517 7.6408 0.06762154 0.7255 0.7888 0.7640800
# ... with 499 more rows
```

Classements

Les fonctions « **row_number** », « **ntile** », « **min_rank** », « **dense_rank** », « **percent_rank** » et « **cume_dist** » permettent d'effectuer des palmarès et classifications de vos données.

```
> airQuality %>%
+   mutate(DateHeure= paste(Date,Time),
+          DateHeure= as.POSIXct(
+                     strptime(DateHeure,'%d/%m/%Y %H.%M.%S')),
+          JourAnnee = yday(DateHeure))  %>%
+   filter(year(DateHeure) == 2004 & month(DateHeure) == 3)%>%
+   arrange(DateHeure)%>%
+   select(JourAnnee, AH) %>%
+   mutate( frn       = row_number(JourAnnee),
+           fminR     = min_rank(JourAnnee),
+           fdenseR   = dense_rank(JourAnnee),
+           fpercentR = percent_rank(JourAnnee),
+           fcumedist = cume_dist(JourAnnee),
+           fntile    = ntile(JourAnnee,170))%>%
+ as_tibble()
# A tibble: 509 x 8
   JourAnnee     AH   frn fminR fdenseR  fpercentR  fcumedist fntile
       <dbl>  <dbl> <int> <int>   <int>      <dbl>      <dbl>  <int>
 1        70 0.7578     1     1       1 0.00000000 0.01178782      1
 2        70 0.7255     2     1       1 0.00000000 0.01178782      1
 3        70 0.7502     3     1       1 0.00000000 0.01178782      1
 4        70 0.7867     4     1       1 0.00000000 0.01178782      2
 5        70 0.7888     5     1       1 0.00000000 0.01178782      2
 6        70 0.7848     6     1       1 0.00000000 0.01178782      2
 7        71 0.7603     7     7       2 0.01181102 0.05893910      3
 8        71 0.7702     8     7       2 0.01181102 0.05893910      3
 9        71 0.7648     9     7       2 0.01181102 0.05893910      3
10        71 0.7517    10     7       2 0.01181102 0.05893910      4
# ... with 499 more rows
```

La fonction summarize

Cette fonction permet de faire de calculs récapitulatifs du jeu de données. Attention la fonction « **summarise** » utilisée toute seule n'est pas très intéressante elle devient très utile en l'association avec la fonction « **group_by** ».

```
> airQuality %>%
+   mutate( Annee = as.integer(substring(Date,7,10)),
+           Mois  = as.integer(substring(Date,4,5)))  %>%
+   select(Annee, Mois, AH) %>%
+   summarize( MoyAH = mean(AH),
+              SumAH = sum(AH))
# A tibble: 1 x 2
      MoyAH     SumAH
      <dbl>     <dbl>
1 -6.837604 -63979.46
>
> mean(airQuality$AH)
[1] -6.837604
> sum(airQuality$AH)
[1] -63979.46
```

Le langage R dispose de plusieurs fonctions de calcul très utiles :

- **sd(x)**, **IQR(x)**, **mad(x)**
- **min(x)**, **quantile(x,0.25)**, **max(x)**
- **first(x)**, **nth(x,2)**, **last(x)**
- **n()**, **n_distinct(x)**

```
> airQuality %>%
+   mutate( Annee = as.integer(substring(Date,7,10)),
+           Mois  = as.integer(substring(Date,4,5)),
+           AH    = abs(AH))  %>%
+   select(Annee, Mois, AH) %>%
+   group_by(Annee, Mois)%>%
+   summarize( SumAH = round(sum(AH),2)) %>%
+   filter(rank(desc(SumAH)) < 4) %>%
+   as_tibble()
Source: local data frame [6 x 3]
Groups: Annee [2]

# A tibble: 6 x 3
  Annee  Mois    SumAH
  <int> <int>    <dbl>
1  2004     6  8469.81
2  2004     8 10925.51
3  2004    12 16518.28
4  2005     1 12633.77
5  2005     2 15508.41
6  2005     3   780.83
> airQuality %>%
+   mutate( Annee = as.integer(substring(Date,7,10)),
+           Mois  = as.integer(substring(Date,4,5)),
```

```
+           AH    = abs(AH),
+           MedAH = median(AH))  %>%
+   select(Annee, Mois, MedAH, AH) %>%
+   group_by(Annee, Mois)%>%
+   summarize( SumAHT  = round(sum(AH),2),
+              SumAHSM = round(sum(AH >= MedAH),2)) %>%
+   as_tibble()
Source: local data frame [14 x 4]
Groups: Annee [?]

# A tibble: 14 x 4
   Annee  Mois   SumAHT SumAHSM
   <int> <int>    <dbl>   <dbl>
 1  2004     3   402.60      26
 2  2004     4  6038.10     217
 3  2004     5  3498.85     279
 4  2004     6  8469.81     656
 5  2004     7  1123.33     601
 6  2004     8 10925.51     731
 7  2004     9  4502.37     569
 8  2004    10  1287.82     694
 9  2004    11   676.79     311
10  2004    12 16518.28     264
11  2005     1 12633.77      84
12  2005     2 15508.41      80
13  2005     3   780.83     169
14  2005     4    54.08       0
```

Les regroupements

La fonction group_by

Une fois que vous avez utilisée la fonction « **group_by** » tous les calculs sont effectués par groupes d'individus.

```
> (parJour<- airQuality %>%
+   mutate( Annee = as.integer(substring(Date,7,10)),
+           Mois  = as.integer(substring(Date,4,5)),
+           Jour  = as.integer(substring(Date,1,2)))  %>%
+   select(Annee, Mois, Jour, AH) %>%
+   group_by(Annee, Mois, Jour)%>%
+   summarize( NbLignes = n(),
+              MoyAH    = round(mean(AH),2),
+              SumAH    = round(sum(AH),2))%>%
+ as_tibble())
Source: local data frame [391 x 6]
Groups: Annee, Mois [?]

# A tibble: 391 x 6
   Annee  Mois  Jour NbLignes MoyAH SumAH
   <int> <int> <int>    <int> <dbl> <dbl>
 1  2004     3    10        6  0.77  4.59
 2  2004     3    11       24  0.78 18.62
 3  2004     3    12       24  0.66 15.91
 4  2004     3    13       24  0.73 17.58
 5  2004     3    14       24  0.85 20.39
 6  2004     3    15       24  0.94 22.66
 7  2004     3    16       24  0.87 20.90
 8  2004     3    17       24  0.80 19.32
 9  2004     3    18       24  0.83 19.82
10  2004     3    19       24  0.92 22.18
# ... with 381 more rows
> (parMois <- parJour %>%
+   group_by(Annee, Mois)%>%
+   summarize( NbLignes = n(),
+              SumAH    = round(sum(SumAH),2))%>%
+ as_tibble())
Source: local data frame [14 x 4]
Groups: Annee [?]

# A tibble: 14 x 4
   Annee  Mois NbLignes     SumAH
   <int> <int>    <int>     <dbl>
 1  2004     3       22    402.60
 2  2004     4       30  -4761.89
 3  2004     5       31  -2101.15
 4  2004     6       30  -6730.21
 5  2004     7       31    723.33
 6  2004     8       31  -8674.50
```

```
 7  2004     9       30  -2697.65
 8  2004    10       31    887.82
 9  2004    11       30    676.79
10  2004    12       31 -15481.72
11  2005     1       31 -11766.23
12  2005     2       28 -14891.63
13  2005     3       31    380.86
14  2005     4        4     54.08
> (parAnnee <- parJour %>%
+   group_by(Annee)%>%
+   summarize( NbLignes = n(),
+              SumAH    = round(sum(SumAH),2))%>%
+ as_tibble())
# A tibble: 2 x 3
  Annee NbLignes     SumAH
  <int>    <int>     <dbl>
1  2004      297 -37756.58
2  2005       94 -26222.92
```

La fonction ungroup

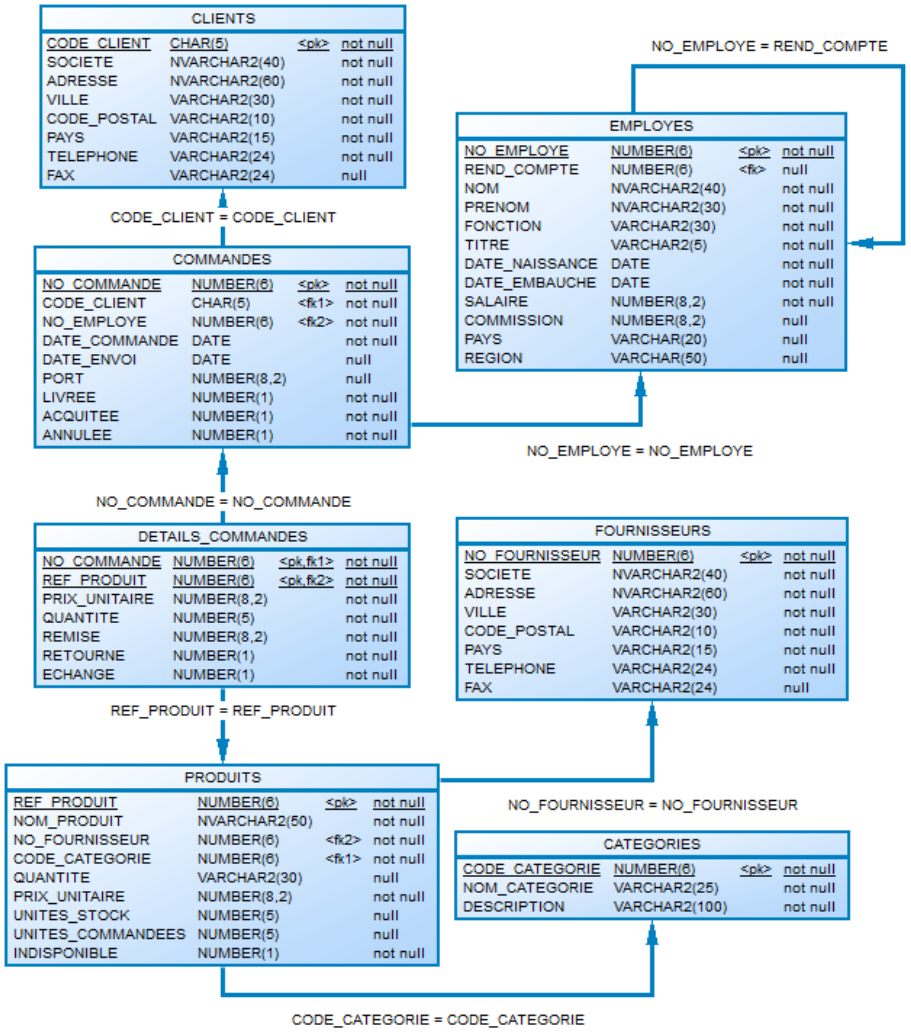
Il est possible de supprimer le regroupement et effectuer des opérations sur les données non groupées en utilisant la fonction « **ungroup** ».

```
> (airQuality <- airQuality %>%
+         mutate( Annee = as.integer(substring(Date,7,10)),
+                 Mois  = as.integer(substring(Date,4,5)),
+                 AH    = abs(AH))  %>%
+         select(Annee, Mois, AH) %>%
+         as_tibble())
# A tibble: 9,357 x 3
   Annee  Mois     AH
   <int> <int>  <dbl>
 1  2004     3 0.7578
 2  2004     3 0.7255
 3  2004     3 0.7502
 4  2004     3 0.7867
 5  2004     3 0.7888
 6  2004     3 0.7848
 7  2004     3 0.7603
 8  2004     3 0.7702
 9  2004     3 0.7648
10  2004     3 0.7517
# ... with 9,347 more rows
> (sumAM<- airQuality %>% group_by(Annee, Mois))
# A tibble: 9,357 x 3
# Groups:   Annee, Mois [14]
   Annee  Mois     AH
   <int> <int>  <dbl>
 1  2004     3 0.7578
 2  2004     3 0.7255
 3  2004     3 0.7502
 4  2004     3 0.7867
 5  2004     3 0.7888
```

```
 6  2004     3 0.7848
 7  2004     3 0.7603
 8  2004     3 0.7702
 9  2004     3 0.7648
10  2004     3 0.7517
# ... with 9,347 more rows
> (s <- sumAM %>% summarise(n = n()))
# A tibble: 14 x 3
# Groups:   Annee [?]
   Annee  Mois     n
   <int> <int> <int>
 1  2004     3   510
 2  2004     4   720
 3  2004     5   744
 4  2004     6   720
 5  2004     7   744
 6  2004     8   744
 7  2004     9   720
 8  2004    10   744
 9  2004    11   720
10  2004    12   744
11  2005     1   744
12  2005     2   672
13  2005     3   744
14  2005     4    87
> (s %>% summarise(n = sum(n)))
# A tibble: 2 x 2
  Annee     n
  <int> <int>
1  2004  7110
2  2005  2247
> (s %>% ungroup() %>% summarise(n = sum(n)))
# A tibble: 1 x 1
      n
  <int>
1  9357
```

Les données relationnelles

Il est rare qu'une analyse de données ne comporte qu'une seule table de données. Typiquement, vous avez plusieurs tables de données et vous devez les combiner pour répondre aux questions qui vous intéressent. Collectivement, plusieurs tables de données sont appelées données relationnelles car ce sont les relations, et pas seulement les jeux de données individuels, qui sont important. Une façon de montrer les relations entre les différentes tables est avec un dessin :




```
> library(tidyverse)
> filenames <- list.files("c:/Solutions/donnees/stagiaire",
+                         pattern="*.csv", full.names=TRUE)
> stagiaire<- sapply(filenames,FUN=read_delim,delim=";",
+                 trim_ws = TRUE,locale = locale(encoding = "UTF-8"))
> names(stagiaire)<-sub(".csv","",
+            sub("c:/Solutions/donnees/stagiaire/",
+                "", names(stagiaire)))
> stagiaire$employes$Date.Naissance <- as.Date(
+                    stagiaire$employes$Date.Naissance , "%Y-%m-%d")
> stagiaire$employes$Date.Embauche  <- as.Date(
+                    stagiaire$employes$Date.Embauche , "%Y-%m-%d")
> stagiaire$commandes$Date.Commande <- as.Date(
```

```
+                     stagiaire$commandes$Date.Commande , "%Y-%m-%d")
> stagiaire$commandes$Date.Envoi    <- as.Date(
+                     stagiaire$commandes$Date.Envoi, "%Y-%m-%d")
> sapply(stagiaire,print)
# A tibble: 10 x 3
    Code                Categorie                              Description
   <int>                    <chr>                                    <chr>
 1     1                  Boissons            Boissons, cafés, thés, bières
 2     2                Condiments         Sauces, assaisonnements et épices
 3     3                  Desserts                    Desserts et friandises
 4     4         Produits laitiers                                  Fromages
 5     5         Pâtes et céréales        Pains, biscuits, pâtes et céréales
 6     6                   Viandes                          Viandes préparées
 7     7             Produits secs              Fruits secs, raisins, autres
 8     8 Poissons et fruits de mer        Poissons, fruits de mer, escargots
 9     9                Conserves Fruits, légumes en conserve et confitures
10    10        Viande en conserve                         Viande en conserve
...
```

Les clés

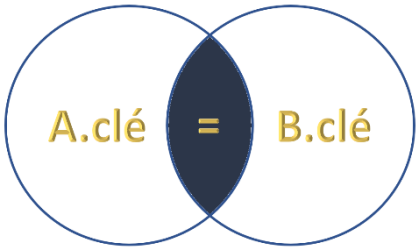
Les variables utilisées pour connecter chaque paire de tables sont appelées des clés. Une clé est une variable (ou un ensemble de variables) qui identifie de manière unique une observation.

```
> stagiaire$details_commandes %>%
+     count( Commande,Ref.Produit) %>%
+     filter(n > 1)
# A tibble: 0 x 3
# ... with 3 variables: Commande <int>, Ref.Produit <int>, n <int>
> stagiaire$details_commandes %>%
+     count( Ref.Produit) %>%
+     filter(n > 1)
# A tibble: 120 x 2
   Ref.Produit     n
         <int> <int>
 1           1  3789
 2           2  4103
 3           3  3873
 4           4  4165
 5           5  3686
 6           6  3936
 7           7  3916
 8           8  4096
 9           9  4327
10          10  4254
# ... with 110 more rows
```

Les jointures classiques

Une jointure vous permet de combiner les variables à partir de deux tables. Il correspond tout d'abord aux observations par leurs clés, puis copie les variables d'une table à l'autre. De la même manière que la fonction « **mutate** », les fonctions de jointure ajoutent des variables à droite, donc si vous avez déjà beaucoup de variables, les nouvelles variables ne seront pas imprimées.

Les jointures internes INNER_JOIN



Le type le plus simple de jointure est la « **inner_join** » qui correspond à des paires d'observations chaque fois que leurs clés sont égales.

```
> stagiaire$employes %>%
+     select(No.Employe,Nom,Fonction) %>%
+     inner_join(stagiaire$commandes, by = "No.Employe") %>%
+     select(Nom,Fonction,Code.Client,Port,Date.Commande,Livree)
# A tibble: 13,462 x 6
     Nom        Fonction Code.Client  Port Date.Commande Livree
   <chr>           <chr>       <chr> <dbl>        <date>  <int>
 1 Besse Représentant(e)       BSBEV  57.6    2010-02-02      1
 2 Besse Représentant(e)       ISLAT  82.5    2010-02-14      1
 3 Besse Représentant(e)       CONSH  99.8    2010-01-11      1
 4 Besse Représentant(e)       ISLAT  59.6    2010-01-08      1
 5 Besse Représentant(e)       SEVES  56.6    2010-02-17      1
 6 Besse Représentant(e)       EASTC  85.2    2010-01-26      1
 7 Besse Représentant(e)       ISLAT  81.8    2010-02-24      1
 8 Besse Représentant(e)       ISLAT  72.7    2010-02-05      1
 9 Besse Représentant(e)       NORTS  82.4    2010-02-23      1
10 Besse Représentant(e)       EASTC  73.3    2010-01-22      1
# ... with 13,452 more rows
> stagiaire$clients %>%
+     select(Client,Ville) %>%
+     inner_join(stagiaire$fournisseurs, by = "Ville") %>%
+     select(Client, Fournisseur, Ville) %>%
+     arrange(Ville,Client)
# A tibble: 15 x 3
                  Client                      Fournisseur          Ville
                   <chr>                            <chr>          <chr>
 1    Alfreds Futterkiste       Heli Süßwaren GmbH  Co. KG         Berlin
 2    Lehmanns Marktstand Plutzer Lebensmittelgroßmärkte AG Frankfurt a.M.
 3        Around the Horn                   Exotic Liquids         London
 4          B's Beverages                   Exotic Liquids         London
 5  Consolidated Holdings                   Exotic Liquids         London
 6     Eastern Connection                   Exotic Liquids         London
```

7	North/South	Exotic Liquids	London
8	Seven Seas Imports	Exotic Liquids	London
9	Mère Paillarde	Ma Maison	Montréal
10	Paris spécialités	Aux joyeux ecclésiastiques	Paris
11	Spécialités du monde	Aux joyeux ecclésiastiques	Paris
12	Comércio Mineiro	Refrescos Americanas LTDA	São Paulo
13	Familia Arquibaldo	Refrescos Americanas LTDA	São Paulo
14	Queen Cozinha	Refrescos Americanas LTDA	São Paulo
15	Tradição Hipermercados	Refrescos Americanas LTDA	São Paulo

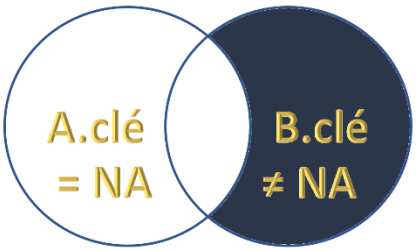
Les jointures externes

Une équi-jointure « **inner_join** » conserve les observations qui apparaissent dans les deux tableaux. Une jointure externe conserve les observations qui apparaissent dans au moins une des tables.

dplyr	merge
`inner_join(x, y)`	`merge(x, y)`
`left_join(x, y)`	`merge(x, y, all.x = TRUE)`
`right_join(x, y)`	`merge(x, y, all.y = TRUE),`
`full_join(x, y)`	`merge(x, y, all.x = TRUE, all.y = TRUE)`

dplyr	SQL
`inner_join(x, y, by = "z")`	`SELECT * FROM x INNER JOIN y USING(z)`
`left_join(x, y, by = "z")`	`SELECT * FROM x LEFT OUTER JOIN y USING (z)`
`right_join(x, y, by = "z")`	`SELECT * FROM x RIGHT OUTER JOIN y USING (z)`
`full_join(x, y, by = "z")`	`SELECT * FROM x FULL OUTER JOIN y USING (z)`

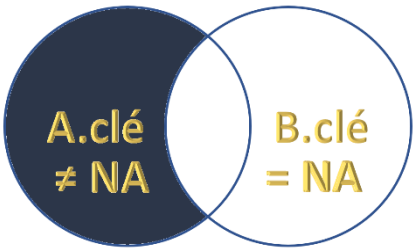
La jointure à gauche left_join




```
> stagiaire$clients %>%
+     select(Client,Ville) %>%
+     left_join(stagiaire$fournisseurs, by = "Ville") %>%
+     select(Ville, Client, Fournisseur) %>%
+     arrange(Ville,Client)
# A tibble: 91 x 3
          Ville                       Client                Fournisseur
          <chr>                        <chr>                      <chr>
 1       Aachen     Drachenblut Delikatessen                       <NA>
 2  Albuquerque   Rattlesnake Canyon Grocery                       <NA>
 3    Anchorage       Old World Delicatessen                       <NA>
 4        Århus                   Vaffeljernet                       <NA>
 5    Barcelona       Galería del gastrónomo                       <NA>
 6 Barquisimeto            LILA-Supermercado                       <NA>
 7      Bergamo Magazzini Alimentari Riuniti                       <NA>
 8       Berlin          Alfreds Futterkiste Heli Süßwaren GmbH  Co. KG
 9         Bern            Chop-suey Chinese                       <NA>
10        Boise           Save-a-lot Markets                       <NA>
# ... with 81 more rows
```

L'exemple précèdent présente une jointure à gauche qui conserve toutes les observations qui apparaissent dans l'ensemble « **clients** ».

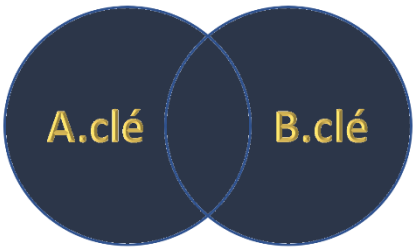
La jointure à droite right_join




```
> stagiaire$clients %>%
+     select(Client,Ville) %>%
+     right_join(stagiaire$fournisseurs, by = "Ville") %>%
+     select(Ville, Client, Fournisseur) %>%
+     arrange(Ville,Client)
# A tibble: 38 x 3
           Ville              Client                           Fournisseur
           <chr>               <chr>                                 <chr>
 1      Ann Arbor                <NA>              Grandma Kelly's Homestead
 2         Annecy                <NA>                           Gai pâturage
 3           Bend                <NA>                     Bigfoot Breweries
 4         Berlin Alfreds Futterkiste            Heli Süßwaren GmbH  Co. KG
 5         Boston                <NA>           New England Seafood Cannery
 6       Cuxhaven                <NA> Nord-Ost-Fisch Handelsgesellschaft mbH
 7 Frankfurt a.M. Lehmanns Marktstand      Plutzer Lebensmittelgroßmärkte AG
 8       Göteborg                <NA>                         PB Knäckebröd AB
 9   Lappeenranta                <NA>                                 Karkki Oy
10         London     Around the Horn                         Exotic Liquids
# ... with 28 more rows
```

L'exemple précèdent présente une jointure à droite qui conserve toutes les observations qui apparaissent dans l'ensemble « **fournisseurs** ».

La jointure complète full_join




```
> stagiaire$clients %>%
+     select(Client,Ville) %>%
+     full_join(stagiaire$fournisseurs, by = "Ville") %>%
+     select(Ville, Client, Fournisseur) %>%
+     arrange(Ville,Client)
# A tibble: 114 x 3
          Ville                      Client                Fournisseur
          <chr>                       <chr>                      <chr>
 1       Aachen     Drachenblut Delikatessen                       <NA>
 2  Albuquerque   Rattlesnake Canyon Grocery                       <NA>
```

```
 3    Anchorage       Old World Delicatessen                      <NA>
 4    Ann Arbor                        <NA> Grandma Kelly's Homestead
 5       Annecy                        <NA>              Gai pâturage
 6        Århus                Vaffeljernet                      <NA>
 7    Barcelona       Galería del gastrónomo                      <NA>
 8 Barquisimeto            LILA-Supermercado                      <NA>
 9         Bend                        <NA>         Bigfoot Breweries
10      Bergamo Magazzini Alimentari Riuniti                      <NA>
# ... with 104 more rows
```

L'exemple précèdent présente une jointure à droite qui conserve toutes les observations qui apparaissent dans l'ensemble « **clients** » et dans l'ensemble « **fournisseurs** ».

La définition des clés

Jusqu'à présent, les paires de tables ont toujours été jointes par une variable unique, et cette variable a le même nom dans les deux tableaux. Vous pouvez utiliser d'autres valeurs pour l'argument « **by** » par relier les tables d'une autre manière.

La valeur par défaut pour l'argument est « **by = NULL** », et utilise toutes les variables du même nom qui apparaissent dans les deux tableaux.

```
> stagiaire$employes %>%
+     left_join(stagiaire$commandes)
Joining, by = "No.Employe"
# A tibble: 13,481 x 20
   No.Employe Manager       Nom      Prenom        Fonction Titre
        <int>   <chr>     <chr>       <chr>           <chr> <chr>
 1         37    null    Giroux Jean-Claude       Président    M.
 2         14      37    Fuller      Andrew  Vice-Président    M.
...
> stagiaire$clients %>%
+     left_join(stagiaire$fournisseurs)
Joining, by = c("Adresse", "Ville", "Code.Postal", "Pays", "Telephone",
"Fax")
# A tibble: 91 x 10
    Code                                Client                     Adresse
   <chr>                                 <chr>                       <chr>
 1 CONSH               Consolidated Holdings  Berkeley Gardens12  Brewery
 2 DRACD             Drachenblut Delikatessen                Walserweg 21
 3 DUMON                     Du monde entier 67, rue des Cinquante Otages
...
```

Vous pouvez également utiliser pour l'argument « **by** » un vecteur de caractère nommé « **by = c ("a" = "b")** ». Ainsi la variable « **a** » dans la première table doit correspondre à la variable « **b** » dans la deuxième table.

```
> stagiaire$categories %>%
+     left_join(stagiaire$produits, by = c("Code"="Categorie")) %>%
+     select(Categorie,Produit) %>%
+     arrange(Produit)
# A tibble: 120 x 2
                    Categorie            Produit
                        <chr>              <chr>
 1                    Viandes       Alice Mutton
 2              Produits secs            Amandes
 3                 Condiments      Aniseed Syrup
 4                   Boissons               Beer
 5 Poissons et fruits de mer   Boston Crab Meat
 6                  Conserves Boysenberry Spread
 7                   Desserts        Brownie Mix
 8                 Condiments    Cajun Seasoning
 9                   Desserts           Cake Mix
10          Produits laitiers  Camembert Pierrot
# ... with 110 more rows
```

Le filtrage à l'aide d'une jointure

Il est possible d'utiliser le mécanisme de jointure pour filtrer les observations de la première table mais à l'aide de cette syntaxe vous retrouvez uniquement les variables de la première table.

semi_join

Le filtre conserve uniquement les observations de la première table qui ont une correspondance dans la deuxième.

```
> stagiaire$commandes %>%
+     count(No.Employe)
# A tibble: 92 x 2
   No.Employe     n
        <int> <int>
 1          1   220
 2          2    92
 3          3   159
 4          4   221
 5          5    80
 6          6    89
 7          7   417
 8          8    20
 9          9    90
10         10   125
# ... with 82 more rows
> stagiaire$employes
# A tibble: 111 x 12
   No.Employe Manager       Nom      Prenom        Fonction Titre
        <int>   <chr>     <chr>       <chr>           <chr> <chr>
 1         37    null    Giroux Jean-Claude       Président    M.
 2         14      37    Fuller      Andrew  Vice-Président    M.
 3         18      37  Brasseur       Hervé  Vice-Président    M.
 4         24      14  Buchanan      Steven Chef des ventes    M.
 5         95      18     Leger      Pierre Chef des ventes    M.
 6         11      18     Belin     Chantal Chef des ventes   Mme
 7         33      18  Chambaud      Axelle Chef des ventes   Mme
 8         86      18     Ragon       André Chef des ventes    M.
 9         23      14 Splingart       Lydia Chef des ventes   Mme
10          1      86     Besse        José Représentant(e)    M.
# ... with 101 more rows, and 6 more variables: Date.Naissance <date>,
#   Date.Embauche <date>, Salaire <int>, Commission <chr>, Pays <chr>,
#   Region <chr>
> stagiaire$employes %>%
+               semi_join(stagiaire$commandes)
Joining, by = "No.Employe"
# A tibble: 92 x 12
   No.Employe Manager        Nom   Prenom        Fonction Titre Date.Naissance
        <int>   <chr>      <chr>    <chr>           <chr> <chr>          <date>
 1         84      24     Coutou   Myriam Représentant(e)   Mme      1985-08-31
 2         78      24     Rollet Philippe Représentant(e)    M.      1985-03-09
 3         72      24      Herve   Didier Représentant(e)    M.      1979-09-19
 4        111      86   Teixeira  Claudia Représentant(e)   Mme      1977-12-24
 5         39      23      Jenny   Michel Représentant(e)    M.      1966-06-16
```

```
 6         88      23 Mauroussset    James Représentant(e)    M.     1978-09-26
 7         51      33    Alvarez   Marcel Représentant(e)    M.     1967-07-09
 8          3      33   Letertre   Sylvie Représentant(e)   Mme     1979-01-09
 9         45      86   Gregoire    Renée Représentant(e)   Mme     1982-01-25
10         65      33     Griner Florence Représentant(e)   Mme     1986-09-23
# ... with 82 more rows, and 5 more variables: Date.Embauche <date>,
#   Salaire <int>, Commission <chr>, Pays <chr>, Region <chr>
```

anti_join

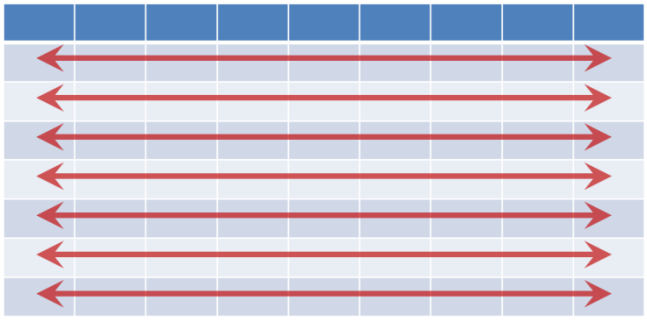
Le filtre conserve uniquement les observations de la première table qui n'ont pas de correspondance dans la deuxième.

```
> stagiaire$employes %>%
+     anti_join(stagiaire$commandes)
Joining, by = "No.Employe"
# A tibble: 19 x 12
   No.Employe Manager        Nom         Prenom                Fonction Titre
        <int>   <chr>      <chr>          <chr>                   <chr> <chr>
 1         11      18      Belin        Chantal        Chef des ventes   Mme
 2         14      37     Fuller         Andrew         Vice-Président    M.
 3         18      37   Brasseur          Hervé         Vice-Président    M.
 4         21    null    Poupard      Claudette Assistante commerciale   Mme
 5         23      14  Splingart          Lydia        Chef des ventes   Mme
 6         24      14   Buchanan         Steven        Chef des ventes    M.
 7         27    null     Maurer      Véronique Assistante commerciale   Mme
 8         30    null   Callahan          Laura Assistante commerciale  Mlle
 9         33      18   Chambaud         Axelle        Chef des ventes   Mme
10         37    null     Giroux    Jean-Claude              Président    M.
11         44    null    Etienne       Brigitte Assistante commerciale   Mme
12         57    null Grangirard       Patricia Assistante commerciale   Mme
13         64    null    Guerdon       Béatrice Assistante commerciale   Mme
14         75    null      Devie        Thérèse Assistante commerciale   Mme
15         86      18      Ragon          André        Chef des ventes    M.
16         89    null    Pouetre Camille-Hélène Assistante commerciale   Mme
17         95      18      Leger          Pierre        Chef des ventes    M.
18        104    null     Ziliox      Francoise Assistante commerciale   Mme
19        109    null     Lampis      Gabrielle Assistante commerciale   Mme
# ... with 6 more variables: Date.Naissance <date>, Date.Embauche <date>,
#   Salaire <int>, Commission <chr>, Pays <chr>, Region <chr>
```

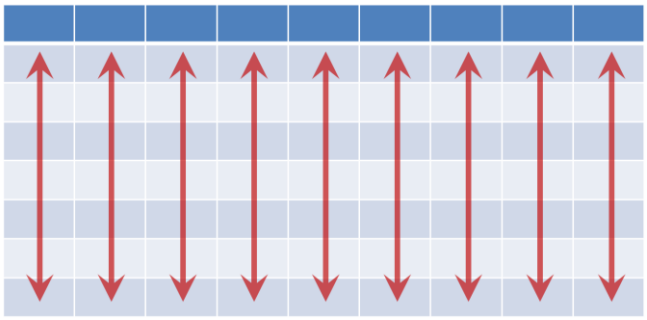
La réorganisation des données

Dans la plupart des analyses réelles vous devrez faire un traitement de préparation de vos jeux de données. La première étape consiste toujours à déterminer quelles sont les variables et quelles sont les observations.

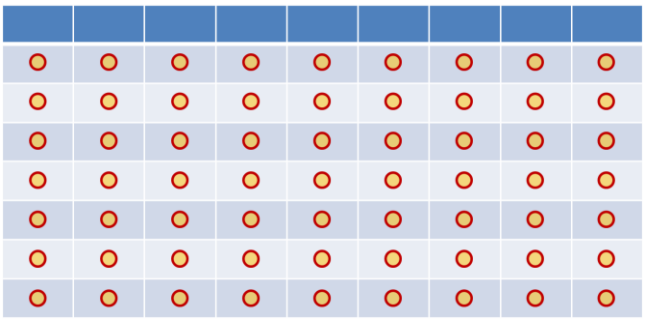
Chaque observation doit avoir sa propre ligne.



Chaque variable doit avoir sa propre colonne.



Chaque valeur doit avoir sa propre cellule.



Ces trois règles sont interdépendantes car il est impossible de conserver seulement deux des trois.

Vous pouvez avoir dans certains jeux de données une variable répartie sur plusieurs colonnes. Il y a également le cas où une observation pourrait être dispersée sur plusieurs lignes.

```
> library(tidyverse)
> library(tools)
> rm(list = ls())
> filenames <- list.files("donnees/meteo", pattern="synop.*.csv",
+                full.names=TRUE)
> lmeteo <-lapply(filenames,read_delim,";", escape_double = FALSE,
```

```
+   na = "mq", col_types = cols(.default = "i",
+   date = col_datetime(format = "%Y%m%d%H%M%S"),
+   ff = col_double(), t = col_double()),
+   locale = locale(encoding = "UTF-8"), trim_ws = TRUE)
> postesMeteo <-read_delim(file="donnees/meteo/postesSynop.csv",
+   delim= ";", col_types = cols(ID = col_integer()),
+   locale = locale(encoding = "UTF-8"), trim_ws = TRUE)
> postesMeteo<- postesMeteo %>%
+                mutate( Nom = toTitleCase(tolower(Nom)))  %>%
+                filter(ID < 8000) #france métropolitaine
> meteo <-lmeteo[[1]]
> for(n in 2:length(lmeteo))  meteo <-bind_rows(meteo,lmeteo[[n]])
> (meteo <- meteo %>%
+     select(numer_sta,date,dd,ff,t,u,vv,n,pres,rr1) %>%
+     transmute(Station                = numer_sta,
+               Date                   = date     ,
+               VitesseVent            = ff       ,
+               Temperature            = round(t - 273.15,2) ,
+               Humidite               = u        ,
+               Visibilite             = vv       ,
+               Pression               = pres     ) %>%
+     inner_join(postesMeteo,by = c("Station" = "ID")) %>%
+     select ( Nom,Date,VitesseVent,
+              Temperature,Humidite,Visibilite,Pression) %>%
+     separate(Date,
+          c("Annee","Mois","Jour","Heure","Minutes","Secondes"))%>%
+     select (Nom,Annee,Mois,Jour,Heure,
+             VitesseVent,Temperature,Humidite,Pression) %>%
+     group_by(Nom,Mois)%>%
+     summarize(VitesseVent = round(mean(VitesseVent,na.rm = T),2),
+               Temperature = round(mean(Temperature,na.rm = T),2),
+               Humidite    = round(mean(Humidite,na.rm    = T),2),
+               Pression    = round(mean(Pression,na.rm    = T),2)))
# A tibble: 503 x 6
# Groups:   Nom [?]
         Nom  Mois VitesseVent Temperature Humidite Pression
       <chr> <chr>       <dbl>       <dbl>    <dbl>    <dbl>
 1 Abbeville    01        4.84        5.69    82.58 100288.3
 2 Abbeville    02        5.20        5.52    82.47 100311.4
 3 Abbeville    03        4.46        6.13    77.85 100618.6
 4 Abbeville    04        3.94        8.70    75.20 100456.7
 5 Abbeville    05        3.46       13.38    77.44 100570.0
 6 Abbeville    06        3.46       15.50    85.17 100662.9
 7 Abbeville    07        3.48       17.64    77.35 101026.2
 8 Abbeville    08        3.47       18.35    76.22 101106.3
 9 Abbeville    09        2.85       17.47    78.67 100991.1
10 Abbeville    10        3.11       10.81    83.69 101258.8
# ... with 493 more rows
```

La fonction spread

Dans le cas ou une observation est dispersée sur plusieurs lignes il est nécessaire de transformer en variables certaines valeurs des observations (créer un tableau croisé).

Abbeville	Janvier	5.69
Abbeville	Février	5.52
Abbeville	Mars	6.13
Abbeville	Avril	8.70
Abbeville	Mai	13.38
Abbeville	Juin	15.50
Abbeville	Juillet	17.64
Abbeville	Août	18.35
Abbeville	Septembre	17.47
Abbeville	Octobre	10.81
Abbeville	Novembre	7.06
Abbeville	Décembre	4.62

	Janvier	Février	Mars	Avril	Mai	Juin	Juillet	Août	Septembre	Octobre	Novembre	Décembre
Abbeville	5.69	5.52	6.13	8.70	13.38	15.50	17.64	18.35	17.47	10.81	7.06	4.62

La transformation à l'aide de la fonction « **spread** » nécessite deux paramètres :

- La colonne qui contient des noms de variables, la colonne de « **clé** ».
- La colonne qui contient des valeurs des variables

```
> meteo
# A tibble: 503 x 6
# Groups:   Nom [42]
         Nom  Mois VitesseVent Temperature Humidite Pression
       <chr> <int>       <dbl>       <dbl>    <dbl>    <dbl>
 1 Abbeville     1        4.84        5.69    82.58 100288.3
 2 Abbeville     2        5.20        5.52    82.47 100311.4
 3 Abbeville     3        4.46        6.13    77.85 100618.6
 4 Abbeville     4        3.94        8.70    75.20 100456.7
 5 Abbeville     5        3.46       13.38    77.44 100570.0
 6 Abbeville     6        3.46       15.50    85.17 100662.9
 7 Abbeville     7        3.48       17.64    77.35 101026.2
 8 Abbeville     8        3.47       18.35    76.22 101106.3
 9 Abbeville     9        2.85       17.47    78.67 100991.1
10 Abbeville    10        3.11       10.81    83.69 101258.8
# ... with 493 more rows
> Temperature <- meteo %>%
+     select (Nom,Mois,Temperature) %>%
+     spread(key = "Mois",value="Temperature")
> names(Temperature) <- c('Nom',paste('T',c('janv',
+                      'fevr','mars','avr','mai','juin','juil'
+                     ,'aout','sept','oct','nov','dec'),sep='.'))
> Temperature
# A tibble: 42 x 13
# Groups:   Nom [42]
                  Nom T.janv T.fevr T.mars T.avr T.mai T.juin T.juil
 *              <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>  <dbl>  <dbl>
```

```
 1          Abbeville   5.69   5.52   6.13  8.70 13.38  15.50  17.64
 2            Ajaccio  10.43  10.79  11.19 14.69 16.90  20.58  23.84
 3            Alencon   5.23   5.67   6.09  8.31 13.34  16.14  18.65
 4      Bale-Mulhouse   3.71   5.15   5.42  9.60 14.15  18.02  20.66
 5             Bastia  10.41  11.15  11.61 14.66 17.43  21.83  25.05
 6 Belle Ile-Le Talut  10.11   8.95   8.65 10.61 14.09  16.35  18.33
 7  Bordeaux-Merignac   8.77   8.48   9.38 11.71 15.92  18.78  21.26
 8            Bourges   5.21   5.72   6.44  9.31 13.51  16.84  20.35
 9     Brest-Guipavas   8.45   7.36   7.32  8.90 12.89  15.33  17.11
10     Caen-Carpiquet   6.20   6.33   6.40  8.39 13.15  15.54  17.76
# ... with 32 more rows, and 5 more variables: T.aout <dbl>,
#   T.sept <dbl>, T.oct <dbl>, T.nov <dbl>, T.dec <dbl>
```

Il est possible de changer les noms des variables en même temps que on effectue l'opération.

```
> (VitesseVent <- meteo %>%
+     select (Nom,Mois,VitesseVent) %>%
+     spread(key = "Mois",value="VitesseVent") %>%
+     rename( V.Janv = `1`,V.Fevr = `2`,
+             V.Mars = `3`,V.Avr  = `4`,
+             V.Mai  = `5`,V.Juin = `6`,
+             V.Juil = `7`,V.Aout = `8`,
+             V.Sept = `9`,V.Oct  = `10`,
+             V.Nov  = `11`,V.Dec  = `12`))
# A tibble: 42 x 13
# Groups:   Nom [42]
                  Nom V.Janv V.Fevr V.Mars V.Avr V.Mai V.Juin V.Juil
 *              <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>  <dbl>  <dbl>
 1          Abbeville   4.84   5.20   4.46  3.94  3.46   3.46   3.48
 2            Ajaccio   2.69   2.71   2.84  2.84  3.22   3.00   3.12
 3            Alencon   3.52   4.42   4.12  3.14  3.00   3.04   2.75
 4      Bale-Mulhouse   3.06   3.62   2.91  2.56  2.39   2.28   2.14
 5             Bastia   2.97   3.01   3.45  2.29  3.07   2.46   2.48
 6 Belle Ile-Le Talut   8.81   8.13   7.36  5.88  4.83   4.98   4.80
 7  Bordeaux-Merignac   3.89   4.16   3.85  3.40  3.76   3.06   2.94
 8            Bourges   3.75   4.29   3.65  2.71  3.03   2.75   2.57
 9     Brest-Guipavas   6.04   5.56   5.65  4.45  3.92   3.85   3.50
10     Caen-Carpiquet   5.74   5.59   4.57  4.08  3.52   3.40   3.45
# ... with 32 more rows, and 5 more variables: V.Aout <dbl>,
#   V.Sept <dbl>, V.Oct <dbl>, V.Nov <dbl>, V.Dec <dbl>
```

Il est également possible de faire ces transformations et en même temps de faire la jointure entre tous les jeux des données.

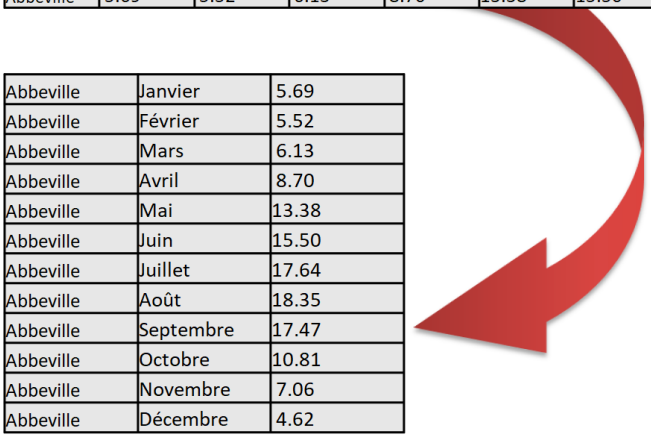
```
> (meteo <-
+     (meteo %>%
+          select (Nom,Mois,Temperature) %>%
+          spread(key = "Mois",value="Temperature") %>%
+          rename( T.Janv = `1`,T.Fevr = `2`,
+                  T.Mars = `3`,T.Avr  = `4`,
+                  T.Mai  = `5`,T.Juin = `6`,
+                  T.Juil = `7`,T.Aout = `8`,
+                  T.Sept = `9`,T.Oct  = `10`,
+                  T.Nov  = `11`,T.Dec  = `12`)) %>%
+     inner_join(
+         (meteo %>%
```

```
+              select (Nom,Mois,VitesseVent) %>%
+              spread(key = "Mois",value="VitesseVent") %>%
+              rename( V.Janv = `1`,V.Fevr = `2`,
+                      V.Mars = `3`,V.Avr  = `4`,
+                      V.Mai  = `5`,V.Juin = `6`,
+                      V.Juil = `7`,V.Aout = `8`,
+                      V.Sept = `9`,V.Oct  = `10`,
+                      V.Nov  = `11`,V.Dec  = `12`)),by="Nom")  %>%
+     inner_join(
+         (meteo %>%
+              select (Nom,Mois,Humidite) %>%
+              spread(key = "Mois",value="Humidite") %>%
+              rename( H.Janv = `1`,H.Fevr = `2`,
+                      H.Mars = `3`,H.Avr  = `4`,
+                      H.Mai  = `5`,H.Juin = `6`,
+                      H.Juil = `7`,H.Aout = `8`,
+                      H.Sept = `9`,H.Oct  = `10`,
+                      H.Nov  = `11`,H.Dec  = `12`)),by="Nom")  %>%
+     inner_join(
+         (meteo %>%
+              select (Nom,Mois,Pression) %>%
+              spread(key = "Mois",value="Pression") %>%
+              rename( P.Janv = `1`,P.Fevr = `2`,
+                      P.Mars = `3`,P.Avr  = `4`,
+                      P.Mai  = `5`,P.Juin = `6`,
+                      P.Juil = `7`,P.Aout = `8`,
+                      P.Sept = `9`,P.Oct  = `10`,
+                      P.Nov  = `11`,P.Dec  = `12`)),by="Nom")  %>%
+     inner_join(postesMeteo,by="Nom")  %>%
+     select(-ID) %>%
+     drop_na() %>%
+     column_to_rownames(var = "Nom"))
# A tibble: 41 x 51
# Groups:   Nom [41]
   T.Janv T.Fevr T.Mars T.Avr T.Mai T.Juin T.Juil T.Aout T.Sept T.Oct T.Nov
 *  <dbl>  <dbl>  <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl>
 1   5.69   5.52   6.13  8.70 13.38  15.50  17.64  18.35  17.47 10.81  7.06
 2  10.43  10.79  11.19 14.69 16.90  20.58  23.84  23.41  21.50 17.42 13.55
 3   5.23   5.67   6.09  8.31 13.34  16.14  18.65  19.24  17.23 10.20  7.34
 4   3.71   5.15   5.42  9.60 14.15  18.02  20.66  20.14  17.34  9.33  5.72
 5  10.41  11.15  11.61 14.66 17.43  21.83  25.05  24.51  22.42 16.98 13.38
 6  10.11   8.95   8.65 10.61 14.09  16.35  18.33  19.29  18.04 13.34 10.65
 7   8.77   8.48   9.38 11.71 15.92  18.78  21.26  22.33  20.16 13.34  9.95
 8   5.21   5.72   6.44  9.31 13.51  16.84  20.35  21.08  18.26 10.72  7.34
 9   8.45   7.36   7.32  8.90 12.89  15.33  17.11  17.37  16.38 11.12  8.98
10   6.20   6.33   6.40  8.39 13.15  15.54  17.76  18.33  17.38 10.89  8.03
# ... with 31 more rows, and 40 more variables: T.Dec <dbl>, V.Janv <dbl>,
#   V.Fevr <dbl>, V.Mars <dbl>, V.Avr <dbl>, V.Mai <dbl>, V.Juin <dbl>,
#   V.Juil <dbl>, V.Aout <dbl>, V.Sept <dbl>, V.Oct <dbl>, V.Nov <dbl>,
#   V.Dec <dbl>, H.Janv <dbl>, H.Fevr <dbl>, H.Mars <dbl>, H.Avr <dbl>,
#   H.Mai <dbl>, H.Juin <dbl>, H.Juil <dbl>, H.Aout <dbl>, H.Sept <dbl>,
#   H.Oct <dbl>, H.Nov <dbl>, H.Dec <dbl>, P.Janv <dbl>, P.Fevr <dbl>,
#   P.Mars <dbl>, P.Avr <dbl>, P.Mai <dbl>, P.Juin <dbl>, P.Juil <dbl>,
#   P.Aout <dbl>, P.Sept <dbl>, P.Oct <dbl>, P.Nov <dbl>, P.Dec <dbl>,
#   Latitude <dbl>, Longitude <dbl>, Altitude <int>
```

La fonction gather

La fonction « **gather** » est l'opposé de la commande « **spread** » permettent de rassembler plusieurs colonnes dans une nouvelle variable.

	Janvier	Février	Mars	Avril	Mai	Juin	Juillet	Août	Septembre	Octobre	Novembre	Décembre
Abbeville	5.69	5.52	6.13	8.70	13.38	15.50	17.64	18.35	17.47	10.81	7.06	4.62



Abbeville	Janvier	5.69
Abbeville	Février	5.52
Abbeville	Mars	6.13
Abbeville	Avril	8.70
Abbeville	Mai	13.38
Abbeville	Juin	15.50
Abbeville	Juillet	17.64
Abbeville	Août	18.35
Abbeville	Septembre	17.47
Abbeville	Octobre	10.81
Abbeville	Novembre	7.06
Abbeville	Décembre	4.62

La transformation à l'aide de la commande « **gather** » nécessite trois paramètres :

- Le nom de l'ensemble de colonnes qui représentent les valeurs.
- Le nom de la variable dont les valeurs forment les noms des colonnes.
- Le nom de la variable dont les valeurs sont réparties sur les cellules.

```
> Temperature
# A tibble: 42 x 13
# Groups:   Nom [42]
                 Nom T.Janv T.Fevr T.Mars T.Avr T.Mai T.Juin T.Juil T.Aout
 *             <chr>  <dbl>  <dbl>  <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>
 1          Abbeville   5.69   5.52   6.13  8.70 13.38  15.50  17.64  18.35
 2            Ajaccio  10.43  10.79  11.19 14.69 16.90  20.58  23.84  23.41
 3            Alencon   5.23   5.67   6.09  8.31 13.34  16.14  18.65  19.24
 4      Bale-Mulhouse   3.71   5.15   5.42  9.60 14.15  18.02  20.66  20.14
 5             Bastia  10.41  11.15  11.61 14.66 17.43  21.83  25.05  24.51
 6 Belle Ile-Le Talut  10.11   8.95   8.65 10.61 14.09  16.35  18.33  19.29
 7  Bordeaux-Merignac   8.77   8.48   9.38 11.71 15.92  18.78  21.26  22.33
 8            Bourges   5.21   5.72   6.44  9.31 13.51  16.84  20.35  21.08
 9     Brest-Guipavas   8.45   7.36   7.32  8.90 12.89  15.33  17.11  17.37
10     Caen-Carpiquet   6.20   6.33   6.40  8.39 13.15  15.54  17.76  18.33
# ... with 32 more rows, and 4 more variables: T.Sept <dbl>, T.Oct <dbl>,
#   T.Nov <dbl>, T.Dec <dbl>
> Temperature %>%
+     gather(key = Mois, value = Temperature, -Nom) %>%
+     mutate(Mois = sub('T.','',Mois))
# A tibble: 504 x 3
# Groups:   Nom [42]
                  Nom  Mois Temperature
                <chr> <chr>       <dbl>
 1          Abbeville  Janv        5.69
 2            Ajaccio  Janv       10.43
 3            Alencon  Janv        5.23
```

```
 4      Bale-Mulhouse  Janv        3.71
 5             Bastia  Janv       10.41
 6 Belle Ile-Le Talut  Janv       10.11
 7  Bordeaux-Merignac  Janv        8.77
 8            Bourges  Janv        5.21
 9     Brest-Guipavas  Janv        8.45
10     Caen-Carpiquet  Janv        6.20
# ... with 494 more rows

> names(Temperature) <- c('Nom',1:12)
> Temperature %>%
+     gather(key = Mois, value = Temperature, -Nom) %>%
+     mutate(Mois = as.integer(Mois))
# A tibble: 504 x 3
                 Nom  Mois Temperature
               <chr> <int>       <dbl>
 1          Abbeville     1        5.69
 2            Ajaccio     1       10.43
 3            Alencon     1        5.23
 4      Bale-Mulhouse     1        3.71
 5             Bastia     1       10.41
 6 Belle Ile-Le Talut     1       10.11
 7  Bordeaux-Merignac     1        8.77
 8            Bourges     1        5.21
 9     Brest-Guipavas     1        8.45
10     Caen-Carpiquet     1        6.20
# ... with 494 more rows
```

La division ou la concaténation

Dans les jeux de données il est possible qu'une colonne contienne plusieurs variables délimités par un ou plusieurs caractères de séparation.

separate

```
separate(data, col, into,
      sep = "[^[:alnum:]]+", remove = TRUE,
      convert = FALSE,
      extra = "warn", fill = "warn", ...)
```

```
> meteo %>%
+     select(numer_sta,date) %>%
+     mutate(date = as.character(date))
# A tibble: 164,326 x 2
   numer_sta               date
       <int>              <chr>
 1      7005 2016-01-01 00:00:00
 2      7015 2016-01-01 00:00:00
 3      7020 2016-01-01 00:00:00
 4      7037 2016-01-01 00:00:00
 5      7072 2016-01-01 00:00:00
 6      7110 2016-01-01 00:00:00
 7      7117 2016-01-01 00:00:00
 8      7130 2016-01-01 00:00:00
 9      7149 2016-01-01 00:00:00
10      7168 2016-01-01 00:00:00
# ... with 164,316 more rows
> meteo %>%
+     select(numer_sta,date) %>%
+     separate(date,c("Annee","Mois","Jour",
+                     "Heure","Minutes","Secondes"),convert = T)
# A tibble: 164,326 x 7
   numer_sta Annee  Mois  Jour Heure Minutes Secondes
 *     <int> <int> <int> <int> <int>   <int>    <int>
 1      7005  2016     1     1     0       0        0
 2      7015  2016     1     1     0       0        0
 3      7020  2016     1     1     0       0        0
 4      7037  2016     1     1     0       0        0
 5      7072  2016     1     1     0       0        0
 6      7110  2016     1     1     0       0        0
 7      7117  2016     1     1     0       0        0
 8      7130  2016     1     1     0       0        0
 9      7149  2016     1     1     0       0        0
10      7168  2016     1     1     0       0        0
# ... with 164,316 more rows
```

Vous pouvez également souhaiter d'effectuer la séparation par la position des caractères dans la chaine de caractères. Les valeurs positives commencent à **1** à gauche de la chaîne et les valeurs négatives commencent à **-1** à droite de la chaîne.

```
> meteo %>%
```

```
+     select(numer_sta,date) %>%
+     separate(date,c("Annee","-","Mois",
+                     "-","Jour","Autre","Secondes"),
+              sep = c(4,5,7,8,10,-3),convert = T)
# A tibble: 164,326 x 8
   numer_sta Annee   `-`  Mois   `-`  Jour   Autre Secondes
 *     <int> <int> <chr> <int> <chr> <int>   <chr>    <int>
 1      7005  2016     -     1     -     1  00:00:        0
 2      7015  2016     -     1     -     1  00:00:        0
 3      7020  2016     -     1     -     1  00:00:        0
 4      7037  2016     -     1     -     1  00:00:        0
 5      7072  2016     -     1     -     1  00:00:        0
 6      7110  2016     -     1     -     1  00:00:        0
 7      7117  2016     -     1     -     1  00:00:        0
 8      7130  2016     -     1     -     1  00:00:        0
 9      7149  2016     -     1     -     1  00:00:        0
10      7168  2016     -     1     -     1  00:00:        0
# ... with 164,316 more rows
```

unite

La fonction « **unite** » est moins souvent nécessaire que la fonction « **separate** » mais c'est toujours un outil intéressant pour manipuler transformer vos jeux de données.

```
> meteo %>%
+     select(numer_sta,date) %>%
+     separate(date,c("Annee","Mois","Jour",
+                     "Heure","Minutes","Secondes"))
# A tibble: 164,326 x 7
   numer_sta Annee  Mois  Jour Heure Minutes Secondes
 *     <int> <chr> <chr> <chr> <chr>   <chr>    <chr>
 1      7005  2016    01    01    00      00       00
 2      7015  2016    01    01    00      00       00
 3      7020  2016    01    01    00      00       00
# ... with 164,316 more rows
> meteo %>%
+     select(numer_sta,date) %>%
+     separate(date,c("Annee","Mois","Jour",
+                     "Heure","Minutes","Secondes")) %>%
+     unite("Annee_Mois", "Annee", "Mois")
# A tibble: 164,326 x 6
   numer_sta Annee_Mois  Jour Heure Minutes Secondes
 *     <int>      <chr> <chr> <chr>   <chr>    <chr>
 1      7005    2016_01    01    00      00       00
 2      7015    2016_01    01    00      00       00
 3      7020    2016_01    01    00      00       00
 4      7037    2016_01    01    00      00       00
 5      7072    2016_01    01    00      00       00
 6      7110    2016_01    01    00      00       00
 7      7117    2016_01    01    00      00       00
 8      7130    2016_01    01    00      00       00
 9      7149    2016_01    01    00      00       00
# ... with 164,316 more rows
```