

Introduction à R - partie 4

Aurélie Labbe, Léo Belzile

Nous allons voir ici des notions plus avancées de **R**. Avant de commencer, assurez-vous que vous avez bien téléchargé tous les fichiers nécessaires pour la séance. Vous devez avoir assimilé les notions de la première, deuxième et troisième parties de l'introduction à **R** avant de commencer cette séance.

Dans ce document, nous allons couvrir les thèmes suivants:

- Le concept de tuyau
- La librairie `dplyr` pour manipuler des données
- Autres outils de manipulation de données en **R**

Données

Nous allons continuer à utiliser ici les données présentes dans le fichier `salesmen.txt` qui contient les ventes réalisées par les employés d'une compagnie. Chaque ligne du fichier représente un employé et les variables sont les suivantes:

- `EmpID`: ID de l'employé
- `Month`: mois de l'année (de 1 à 12)
- `nSales`: nombre de ventes totales
- `TotSales`: total des ventes (en \$)

Le concept de tuyau

L'opérateur de piping `|>`, natif à **R** depuis la version 4.1, permet d'éviter d'utiliser les parenthèses emboîtées lors de l'appel de fonctions. Voici des exemples de commandes, avec ou sans tuyau, qui sont équivalentes:

```
donnees <- read.table("Data/salesmen.txt", header = TRUE)

# l'argument de gauche sert de premier argument à la fonction de droite
nrow(donnees)
```

```
[1] 1800
```

```
donnees |> nrow()
```

```
[1] 1800
```

```
# Autre exemple:  
x <- c(1, 2, 5, 6)  
round(exp(diff(log(x))), 1)
```

```
[1] 2.0 2.5 1.2
```

```
# L'ordre des opérations est plus lisible avec le tuyau  
x |>  
  log() |>  
  diff() |>  
  exp() |>  
  round(1) # ici, 1 est fourni au 2e argument (digits)
```

```
[1] 2.0 2.5 1.2
```

L'opérateur `|>` prend donc tout ce qui se trouve à sa gauche et l'insère, comme premier argument, dans la fonction à sa droite. Le code se lit donc de la gauche vers la droite. Il est recommandé de changer de ligne après chaque instance de l'opérateur `|>` afin d'améliorer la lisibilité du code.

Paquet dplyr

La collection de paquets **tidyverse** contient une collection d'outils pour la science des données. Parmi ceux-ci, le paquet **dplyr** contient des fonctions qui peuvent être très utiles pour préparer des données, et notamment pour réaliser des tâches impliquant le **filtrage** de données (par ex., sélectionner certaines lignes ou colonnes remplissant certaines conditions), la **fusion** de tableaux sur des lignes ou colonnes ou **l'agrégation** de données (par ex., le calcul de résumés statistique sur des sous-ensembles de données). Nous allons voir en détail ces trois aspects dans les sections suivantes.

Filtrage de données

La fonction `filter` du paquet `dplyr` permet de sélectionner les lignes qui vérifient une condition logique. Si on a plusieurs conditions, on les met séparément avec des virgules. Chaque énoncé doit être vrai (intersection) pour que la ligne soit sélectionnée.

```
library(dplyr)
# Extrait les obs pour lesquelles EmpID > 1005
donnees |> filter(EmpID > 1005)
```

```
# A tibble: 1,080 x 4
  EmpID Month nSales TotSales
  <int> <int>   <int>    <dbl>
1  1006     1      20    18879.
2  1006     2      13    12167.
3  1006     3      21    19674.
4  1006     4      20    19160.
5  1006     5      16    16508.
6  1006     6      14    13779.
7  1006     7      19    17183.
8  1006     8      17    15768.
9  1006     9      17    15315.
10 1006    10      21    20167.
# i 1,070 more rows
```

```
# Extrait les obs pour lesquelles nSales <= 10 ET EmpID > 1005
# Tous les énoncés séparés par des virgules doivent être vrais
donnees |> filter(EmpID > 1005,
                  nSales <= 10)
```

```
# A tibble: 55 x 4
  EmpID Month nSales TotSales
  <int> <int>   <int>    <dbl>
1  1100     4       8     8828.
2  1102     3       6     5953.
3  1102     4       9     8444.
4  1103     1      10     9800.
5  1200     1       8     9304.
6  1201     4      10    10317.
7  1202     3       9     9975.
8  1202     6      10    10141.
```

```

  9 1203      2      5    5567.
10 1203      8     10    9684.
# i 45 more rows

```

```

# Extrait les obs pour lesquelles EmpID > 1005 ou nSales <= 10
donnees |> filter(EmpID %in% c(1000, 1005))

```

```

# A tibble: 24 x 4
  EmpID Month nSales TotSales
  <int> <int>   <int>    <dbl>
1  1000     1      11    12607.
2  1000     2       9     9211.
3  1000     3      14    14999.
4  1000     4      13    13421.
5  1000     5      14    14186.
6  1000     6      10    10012.
7  1000     7      25    25031.
8  1000     8      11    11537.
9  1000     9      19    18125.
10 1000    10      15    14390.
# i 14 more rows

```

```

# Erreurs fréquentes:
# 1. Mettre un seul signe d'égalité plutôt que ==
# 2. Mettre des énoncés mutuellement exclusifs plutôt qu'utiliser %in%
donnees |> filter(
  EmpID == 1000,
  EmpID == 1005)

```

```

# A tibble: 0 x 4
# i 4 variables: EmpID <int>, Month <int>, nSales <int>, TotSales <dbl>

```

```

# Retourne un tableau vide, puisque les deux énoncés ne peuvent être simultanément vrais

```

Sélection de variables

La fonction `select` de `dplyr` permet de sélectionner des colonnes par nom ou par leur position.

```
library(dplyr)
# Créer un tableau qui contient les colonnes EmpID et Month
donnees |>
  select(EmpID,
         Month)
```

```
# A tibble: 1,800 x 2
```

```
  EmpID Month
  <int> <int>
1    100     1
2    100     2
3    100     3
4    100     4
5    100     5
6    100     6
7    100     7
8    100     8
9    100     9
10   100    10
```

```
# i 1,790 more rows
```

```
# Créer un tableau sans les colonnes nSales et TotSales de donnees
donnees |>
  select(-c(nSales, TotSales))
```

```
# A tibble: 1,800 x 2
```

```
  EmpID Month
  <int> <int>
1    100     1
2    100     2
3    100     3
4    100     4
5    100     5
6    100     6
7    100     7
8    100     8
9    100     9
10   100    10
```

```
# i 1,790 more rows
```

```
# Créer un tableau qui contient toutes les colonnes entre EmpID et nSales
donnees |>
  select(EmpID:nSales)
```

```
# A tibble: 1,800 x 3
  EmpID Month nSales
  <int> <int>   <int>
1    100     1     16
2    100     2     21
3    100     3     11
4    100     4     18
5    100     5     13
6    100     6     13
7    100     7     17
8    100     8     21
9    100     9     20
10   100    10     13
# i 1,790 more rows
```

Il existe d'autres options très utiles de la fonction `select` telles que (par exemple):

- `start_with`: sélectionne les colonnes dont le nom commence par
- `ends_with`: sélectionne les colonnes dont le nom se termine par
- `contains`: sélectionne les colonnes dont le nom contient la chaîne de caractères

```
donnees |>
  select(contains("Sales"))
```

```
# A tibble: 1,800 x 2
  nSales TotSales
  <int>   <dbl>
1     16 15687.
2     21 21497.
3     11 12016.
4     18 18537.
5     13 13441.
6     13 12749.
7     17 16932.
8     21 21865.
9     20 20147.
10     13 12928.
# i 1,790 more rows
```

```
donnees |>
  select(starts_with("Emp"))
```

```
# A tibble: 1,800 x 1
```

```
  EmpID
  <int>
```

```
1    100
2    100
3    100
4    100
5    100
6    100
7    100
8    100
9    100
10   100
```

```
# i 1,790 more rows
```

Modification et création de colonnes

On aura souvent besoin de créer de nouvelles colonnes à partir de données existantes. La fonction `mutate` permet de faire des modifications à partir de colonnes existantes et d'opérations arithmétiques courantes.

```
donnees <- donnees |>
  mutate(ventesMoy = TotSales / nSales,
         EmpID = factor(EmpID))
```

Combiner des bases de données

Il existe plusieurs fonctions de la librairie `dplyr` qui permettent de combiner ensemble des tables de données même lorsque le nombre de lignes et colonnes n'est pas le même:

- `bind_rows(x, y)`: combine les tables `x` et `y` par lignes même si elles n'ont pas le même nombre de colonnes (les variables manquantes sont traitées comme des valeurs manquantes)
- `inner_join(x, y, by =)`: combine les observations de `x` et `y` par rapport à la variable `by`
- `left_join(x, y, by =)`: combine les observations par rapport à la variable `by` dans la table `x`, même si elles n'apparaissent pas dans la table `y`

- `left_join(x, y, by =)`: combine les observations par rapport à la variable `by` dans la table `y`, même si elles n'apparaissent pas dans la table `x`
- `full_join(x, y, by =)`: combine les observations par rapport à la variable `by` dans `x` ou dans `y`.

Pour illustrer l'utilisation de ces fonctions, nous allons utiliser trois fichiers de données qui contiennent chacun des parties des données `salesmen.txt`:

```
partie1 <- read.table("Data/sales_part1.txt")
colnames(partie1)
```

```
[1] "EmpID" "Month" "nSales" "Team"
```

```
unique(partie1$EmpID)
```

```
[1] 100 101 102 103 104 105 200 201 202 203 204 300 301 302 400 401 402 403 404
[20] 405 406 500 501 502 503 504 505 506 600 601 602 603 604 700 701 702 703 704
[39] 705 706 800 801 802 803 804 900 901 902 903 904 905 906 907 908
```

```
partie2 <- read.table("Data/sales_part2.txt")
partie3 <- read.table("Data/sales_part3.txt")
```

Voici donc des exemples de code pour combiner les données:

```
# Combiner ensemble les données partie1 et part 2 par ligne
sales_combine <- bind_rows(partie1, partie2)
```

Exercise 0.1. Que font les commandes ci-dessous?

```
sales_full <- sales_combine |>
  full_join(y = partie3,
            by = c("EmpID", "Month"))
head(sales_full)
```

	EmpID	Month	nSales	Team	TotSales
1	100	1	16	A	15687.46
2	100	2	21	A	21496.56
3	100	3	11	A	12015.78
4	100	4	18	A	18537.36
5	100	5	13	A	13440.72
6	100	6	13	A	12748.57


```
tail(sales_full)
```

	EmpID	Month	nSales	Team	TotSales
1795	2504	7	30	<NA>	30474.46
1796	2504	8	20	<NA>	19388.83
1797	2504	9	22	<NA>	23814.61
1798	2504	10	17	<NA>	15438.43
1799	2504	11	21	<NA>	20388.20
1800	2504	12	27	<NA>	27398.28

```
sales_jtgauche <- partie1 |>
  left_join(y = partie3,
            by = c("EmpID", "Month"))
head(sales_jtgauche)
```

	EmpID	Month	nSales	Team	TotSales
1	100	1	16	A	15687.46
2	100	2	21	A	21496.56
3	100	3	11	A	12015.78
4	100	4	18	A	18537.36
5	100	5	13	A	13440.72
6	100	6	13	A	12748.57

```
tail(sales_jtgauche)
```

	EmpID	Month	nSales	Team	TotSales
643	908	7	26	A	23665.94
644	908	8	25	A	22634.97
645	908	9	27	A	24878.33
646	908	10	25	A	24022.05
647	908	11	27	A	26595.12
648	908	12	34	A	32262.33

```
sales_jtdroite <- partie1 |>
  right_join(y = partie3,
             by = c("EmpID", "Month"))
head(sales_jtdroite)
```

	EmpID	Month	nSales	Team	TotSales
1	100	1	16	A	15687.46
2	100	2	21	A	21496.56
3	100	3	11	A	12015.78
4	100	4	18	A	18537.36
5	100	5	13	A	13440.72
6	100	6	13	A	12748.57

```
tail(sales_jtdroite)
```

	EmpID	Month	nSales	Team	TotSales
1795	2504	7	NA	<NA>	30474.46
1796	2504	8	NA	<NA>	19388.83
1797	2504	9	NA	<NA>	23814.61
1798	2504	10	NA	<NA>	15438.43
1799	2504	11	NA	<NA>	20388.20
1800	2504	12	NA	<NA>	27398.28

Calculer des statistiques descriptives

Il peut être utile de savoir comment calculer une statistique (`mean`, `min`, `max`, etc.) sur un sous-groupe de données, lequel est défini comme facteur. Pour ce faire avec `dplyr`, on regroupe d'abord les données avec `group_by`, puis on utilise la fonction `summarize` pour calculer

```
# Combinaison group_by + summarize dans `dplyr`
donnees |>
  group_by(EmpID) |> # nom de facteur(s)
  summarize(somme_nSales = sum(nSales, na.rm = TRUE),
            somme_TotSales = sum(TotSales, na.rm = TRUE))
```

```
# A tibble: 150 x 3
  EmpID somme_nSales somme_TotSales
  <fct>      <int>      <dbl>
1 100         203      204039.
2 101         189      184920.
3 102         209      207600.
4 103         211      207766.
5 104         257      255765.
6 105         276      264471.
7 200         193      198301.
8 201         208      211428.
```

```

  9 202          211      211426.
10 203          238      237408.
# i 140 more rows

```

```

# Calculer la moyenne de toutes les colonnes numériques
donnees |>
  group_by(EmpID) |>
  summarize(across(where(is.numeric), ~ mean(.x, na.rm = TRUE)))

```

```

# A tibble: 150 x 5
  EmpID Month nSales TotSales ventesMoy
  <fct> <dbl> <dbl>    <dbl>    <dbl>
1 100     6.5  16.9   17003.    1008.
2 101     6.5  15.8   15410.     988.
3 102     6.5   19    18873.     997.
4 103     6.5  17.6   17314.     993.
5 104     6.5  21.4   21314.     989.
6 105     6.5  23    22039.     960.
7 200     6.5  16.1   16525.    1030.
8 201     6.5  18.9   19221.    1026.
9 202     6.5  17.6   17619.     993.
10 203     6.5  21.6   21583.    1002.
# i 140 more rows

```

Fonctions de base

Si le paquet `dplyr` offre des fonctionnalités et une syntaxe conviviale, il y a plusieurs fonctions de base de **R** qui peuvent être très utiles.

Sélection de variables et d'observations

La fonction `subset` permet à la fois de sélectionner certaines observations remplissant une condition logique, mais aussi de sélectionner certaines colonnes. Voici deux exemples:

```
subset(donnees, EmpID == 1000 & Month <= 5)
```

```

# A tibble: 5 x 5
  EmpID Month nSales TotSales ventesMoy
  <fct> <int> <int>    <dbl>    <dbl>
1 1000     1    11   12607.    1146.

```

2	1000	2	9	9211.	1023.
3	1000	3	14	14999.	1071.
4	1000	4	13	13421.	1032.
5	1000	5	14	14186.	1013.

```
subset(donnees, EmpID == 1000 & Month <= 5,
       select = c(Month, nSales, TotSales))
```

```
# A tibble: 5 x 3
  Month nSales TotSales
  <int> <int>    <dbl>
1     1     11  12607.
2     2      9   9211.
3     3     14  14999.
4     4     13  13421.
5     5     14  14186.
```

Combiner des tables de données

Il existe deux fonctions de base qui permettent de combiner des tables de données par lignes (fonction `rbind`, auquel cas le nombre de colonnes des deux tableaux est supposé être le même) ou bien par colonne (fonction `cbind`, auquel cas le nombre de lignes est supposé être le même). Voici un petit exemple d'utilisation de ces fonctions:

```
# Créer
mat1 <- matrix(1:25, ncol = 5, byrow = TRUE)
mat2 <- matrix(1:15, ncol = 5, byrow = TRUE)

# Combiner des lignes
rbind(mat1, mat2)
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	2	3	4	5
[2,]	6	7	8	9	10
[3,]	11	12	13	14	15
[4,]	16	17	18	19	20
[5,]	21	22	23	24	25
[6,]	1	2	3	4	5
[7,]	6	7	8	9	10
[8,]	11	12	13	14	15

```
# Combiner des colonnes
mat3 <- matrix(1:15, nrow = 5, byrow = TRUE)
cbind(mat1, mat3)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]     1     2     3     4     5     1     2     3
[2,]     6     7     8     9    10     4     5     6
[3,]    11    12    13    14    15     7     8     9
[4,]    16    17    18    19    20    10    11    12
[5,]    21    22    23    24    25    13    14    15
```

Finalement, vous pouvez utiliser aussi la fonction `merge` qui permet elle aussi de combiner des tables de données selon des colonnes spécifiques. Voici un exemple:

```
emp_comb <- rbind(partie1, data.frame(partie2, Team="B"))
sales_merge2 <- merge(x = emp_comb,
                      y = partie3,
                      by = c("EmpID", "Month"),
                      all = TRUE)
head(sales_merge2)
```

```
      EmpID Month nSales Team TotSales
1      100     1      16    A 15687.46
2      100    10      13    A 12928.39
3      100    11      19    A 19046.06
4      100    12      21    A 19194.89
5      100     2      21    A 21496.56
6      100     3      11    A 12015.78
```

```
tail(sales_merge2)
```

```
      EmpID Month nSales Team TotSales
1795    908     4      21    A 19258.47
1796    908     5      29    A 25121.55
1797    908     6      24    A 20892.45
1798    908     7      26    A 23665.94
1799    908     8      25    A 22634.97
1800    908     9      27    A 24878.33
```

Agrégation de données

La fonction `aggregate` permet de faire cela, et voici un exemple de code:

```
# Fonction `aggregate` (R de base)
with(donnees,
      aggregate(cbind(nSales, TotSales) ~ EmpID,
                 FUN = sum,
                 na.rm = TRUE))
```

	EmpID	nSales	TotSales
1	100	203	204039.5
2	101	189	184920.4
3	102	209	207600.4
4	103	211	207765.6
5	104	257	255764.7
6	105	276	264470.8
7	200	193	198300.9
8	201	208	211427.8
9	202	211	211426.0
10	203	238	237408.0
11	204	264	262125.0
12	300	199	197977.2
13	301	210	206689.4
14	302	205	208765.4
15	400	203	206193.2
16	401	207	202586.7
17	402	246	242076.3
18	403	233	234518.1
19	404	281	269880.0
20	405	283	274410.8
21	406	295	282261.0
22	500	223	228445.0
23	501	233	235884.4
24	502	234	228115.0
25	503	250	242167.9
26	504	260	255789.6
27	505	288	283639.3
28	506	292	280439.4
29	600	144	145777.8
30	601	192	191112.3
31	602	196	198036.0

32	603	201	197128.6
33	604	211	206561.7
34	700	163	160855.4
35	701	167	168458.6
36	702	180	176954.6
37	703	205	202607.5
38	704	221	213603.5
39	705	251	238456.5
40	706	215	198628.4
41	800	198	200301.5
42	801	223	223230.2
43	802	209	204910.3
44	803	221	218682.0
45	804	246	240987.6
46	900	172	174186.4
47	901	213	212516.1
48	902	198	194776.5
49	903	173	167886.8
50	904	204	194490.4
51	905	258	246696.0
52	906	254	245263.2
53	907	252	239204.5
54	908	278	256114.5
55	1000	178	179792.6
56	1001	213	209506.3
57	1002	208	206029.9
58	1003	234	228536.9
59	1004	234	226237.2
60	1005	229	223012.3
61	1006	228	215096.5
62	1007	270	259014.4
63	1100	214	219088.6
64	1101	247	246387.0
65	1102	225	225190.9
66	1103	228	226709.7
67	1104	242	237797.5
68	1200	202	203578.1
69	1201	207	199460.7
70	1202	203	200125.3
71	1203	193	191619.9
72	1300	180	176988.1
73	1301	198	197062.6
74	1302	186	186461.2

75	1400	217	218684.9
76	1401	209	217729.1
77	1402	215	210744.6
78	1403	256	252864.4
79	1404	272	270552.0
80	1405	248	238706.6
81	1406	299	283162.5
82	1407	314	303596.0
83	1500	180	185997.6
84	1501	262	266798.6
85	1502	217	218464.0
86	1503	258	253930.3
87	1504	257	254646.0
88	1600	198	201915.1
89	1601	199	204561.2
90	1602	207	211120.7
91	1603	248	242953.1
92	1604	295	287122.2
93	1605	267	254836.3
94	1606	311	296701.5
95	1700	204	208463.4
96	1701	209	212288.7
97	1702	251	251236.9
98	1703	251	245920.6
99	1704	266	256055.9
100	1800	163	163323.0
101	1801	177	178632.4
102	1802	173	171918.4
103	1803	204	201003.5
104	1804	190	183505.8
105	1805	231	222153.7
106	1806	244	226039.6
107	1807	233	219732.1
108	1900	217	224160.9
109	1901	254	256521.1
110	1902	260	261009.6
111	1903	236	232955.6
112	1904	303	292933.2
113	1905	282	275781.6
114	1906	268	259256.6
115	1907	290	276045.6
116	1908	307	284890.2
117	2000	201	205382.6

118	2001	235	237587.7
119	2002	218	219976.2
120	2003	267	266339.8
121	2004	278	268989.1
122	2100	190	191494.0
123	2101	198	205558.6
124	2102	222	222570.7
125	2103	211	209293.6
126	2104	246	240079.0
127	2200	227	232220.1
128	2201	263	256421.4
129	2202	237	240998.4
130	2203	245	247267.8
131	2204	259	254283.7
132	2205	300	293434.4
133	2300	205	206322.0
134	2301	228	231874.4
135	2302	240	239919.2
136	2303	284	278275.7
137	2400	199	202468.9
138	2401	211	209210.2
139	2402	232	235596.9
140	2403	174	172906.4
141	2404	227	225247.0
142	2405	264	257150.6
143	2406	292	281411.2
144	2407	268	250351.9
145	2408	305	279961.9
146	2500	154	153564.8
147	2501	142	141856.0
148	2502	169	172312.8
149	2503	182	180321.1
150	2504	175	172642.0