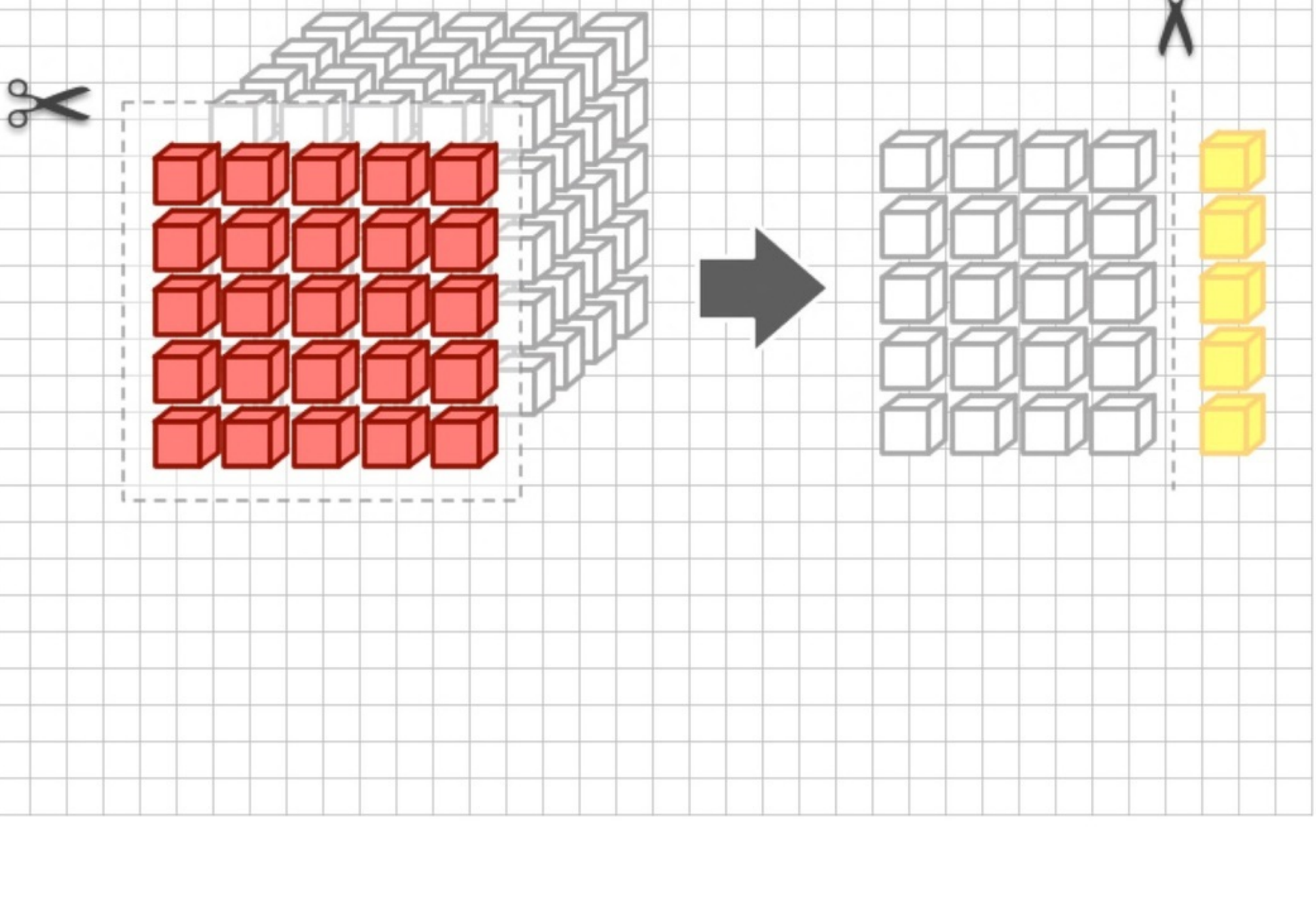
PrivateTeacher

Cours Privés de Science

Traitement des Données avec R



Introduction

Imaginons que l'on se promène en forêt durant une belle journée de printemps. La température est agréable, et le chant des oiseaux se fait toujours plus audible à mesure que l'on s'enfonce dans le bois

Soudain, un bruissement léger mais continu retient notre attention. En baissant les yeux on remarque alors des fourmis qui s'affairent. Un peu plus loin, construite avec des brindilles, on apperçois une fourmillière.

D'innombrables fourmis y travaille. Chacune semblant aller dans sa propre direction, toutes pourtant participant à l'élaboration d'un même édifice.

La nature regorge d'organisations semblables dont le fonctionnement reposent sur le comportement collectif d'un ensemble d'individus.

Lorsque l'on souhaite étudier ce genre de système, il est inutile de s'occuper de chaque individu séparément. A la place il faut se doter d'outils capable de saisir la fourmilière dans son ensemble.

La statistique est une discipline qui a été imaginée dans ce but.

Les statistiques sont faites pour traiter utiliser un grand nombre d'observations.

Le nerf de la guerre

En tant que logiciel de statistique, R n'aura donc besoin que d'une série de valeurs

Voici la manière la plus simple d'entrer une série de valeurs dans R

```
X = c(1,4,5,7)
```

Cette commande (on dit aussi cette instruction) crée une variable nommée "X" contenant les valeurs 1, 4, 5 et 7.

À partir de là, tout est possible. R n'a besoin de rien d'autre pour travailler.

On peut dès lors

- calculer la moyenne de toutes ces valeurs : `mean(x)`
- calculer l'écart-type : `sd(x)`
- ou encore calculer le carré de chaque valeur : `x**2`

Importer des valeurs

De plus souvent, les données sont déjà inscrites dans un fichier.

Si tel est le cas, pas besoin d'entrer les valeurs à la main comme nous l'avons fait, il suffit d'importer le fichier.

Trois scénarios sont possibles :

1) S'il s'agit d'un fichier CSV, on utilisera la commande :

```
data = read.csv("data.csv")
```

2) S'il s'agit d'un fichier Excel .xls on utilisera la commande :

```
data = read_xls("data.xls")
```
⊛

3) S'il s'agit d'un fichier RData enfin, on utilisera la commande :

```
load("data.RData")
```
⊛⊛

Chacune de ces commandes fait la même chose : Lire le contenu du fichier "data" et le placer dans la variable "data"

⊛ Pour fonctionner, la commande `read_xls()` nécessite la librairie "readxl" :

```
install.packages("readxl")
library("readxl")
```

⊛⊛ Le résultat de la commande `load(data.RData)` est de charger tout l'environnement de travail sauvegardé dans le fichier `data.RData`

Celui-ci contient donc toutes les informations au sujet des variables. En particulier il contient :

1) Le nom des variables
2) et leur contenu.

type de variable

Il important de connaître le type de variable avec lesquelles on travail car différents types de variables donnent la possibilité de faire différentes opérations.

On peut additionner deux variables numériques par exemple, mais cette opération est impossible avec une variable de type caractère

Le type d'une variable détermine les opérations que l'on peut faire avec.

Les variables que nous avons utilisé jusqu'ici contenaient des valeurs numériques.

Pour s'en assurer, il suffit d'utiliser la commande `class( )`

Prenons par exemple les deux variables suivantes

```
x = 4
y = c(1,2,3)
```

Chacune de ces deux variables contient des valeurs numérique c'est pourquoi la commande classe nous retourne le même résultat : numeric

```
class(x)
```
→ numeric

```
class(y)
```
→ numeric

Seule la commande length() permet ici de faire la différence. Il s'agit en effet d'une fonction qui retourne la longueure d'une variable. Dans le cas présent, on aura donc :

```
length(x)
```
→ 1

```
length(y)
```
→ 3

Dataframe

Il exist un type de variable qui permet de contenir plusieurs type de donnée différent. Du text et des chiffres p. exple.

Ce type de variable extrêmement flexible et pratique est utilisé très souvent

La structure des dataframe est la suivante

index	label colonne 1	label colonne 2	'''	label colonne 10
1	45	41.8		Matias
2	21	2.9		Hugo
3	37	10.1		Léa
'''	'''	'''		'''
10	7	0.01		Geneviève

Il s'agit simplement de donnée de différent type organisées en colonne.

La première colonne contient toujours le nombre de ligne, numéroté de 1 jusqu'au nombre de mesure.

Notre dataframe contient donc 10 lignes

Un exemple simple

Il s'agit des dataframe. Ce type de variable s'obtient grâce à la commande

```
data.frame( )
```

```
x = c( 4, 7, 3, 2 )
l = c( "A", "B", "C", "D" )

data = data.frame ( x, l )

class ( data )
```

→ data.frame

Affichons à présent le contenu de data :

```
data
```

→

	x	l
1	4	A
2	7	B
3	3	C
4	2	D

Par défaut le nom de la colonne est le même que celui de la variable qui à servi à la créer.

On peut très bien spécifier un autre nom lors de la création du data frame :

```
data = data.frame( col1 = x , col2 = l )
```

```
data
```

	col1	col2
1	4	A
2	7	B
3	3	C
4	2	D

Les dataframes sont donc un moyen puissant pour organiser des données de type différent au sein d'une même structure.

L'intérêt des dataframes cependant est bien plus grand. Ce type de variable en effet permet non seulement de réunir les observations, mais également de les organiser et de le trier

C'est ce que nous allons voir à présent.

Situer les valeurs

Commencons tout d'abord par créer une petite structure comme nous savons le faire :

```
x = c(1, 3, 5, 7, 9, 11)
y = c(2, 4, 6, 8, 10, 12)
z = c("A", "A", "B", "B", "C", "C")
```

Nous avons donc deux variables de type numérique et une variable de type caractère que nous allons mettre ensemble sous les nom de colonnes "data_x", "data_y" et "label" à l'aide de la commande suivante :

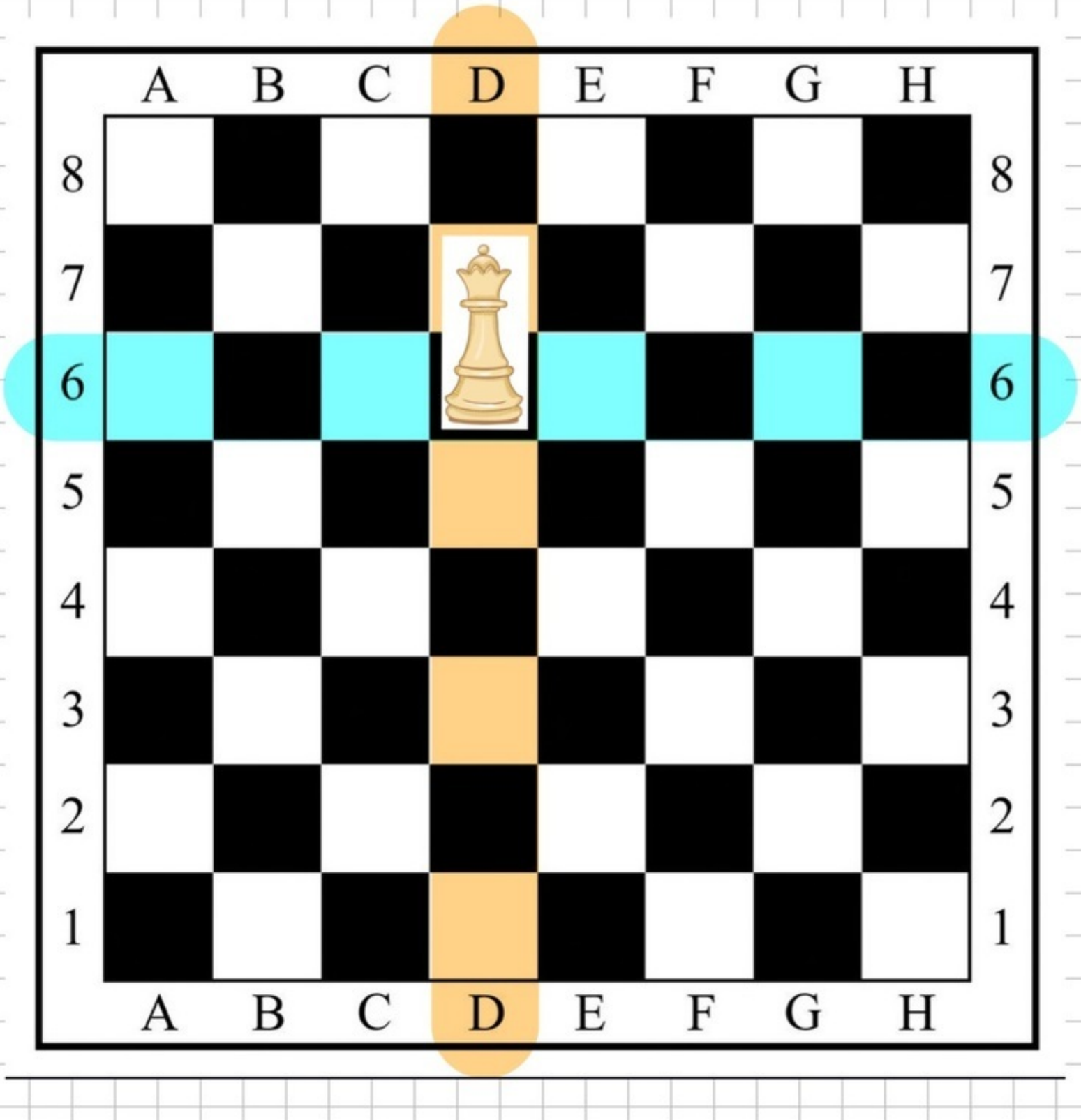
```
data = data.frame(
          data_x = x,
          data_y = y,
          label = z    )
```

Cette commande nous le savons, va regrouper les valeurs des variables "x", "y" et "z" sous forme de tableau.

Voici le résultat :

	data_x	data_y	label
1	1	2	A
2	3	4	A
3	5	6	B
4	7	8	B
5	9	10	C
6	11	12	C

Les tableaux sont un type de donnée
(le type " dataframe ")
dont les éléments peuvent être situé
de la même manière que l'on peut
situer un pion sur un jeux d'échec,
À l'aide de ses deux coordonnées :

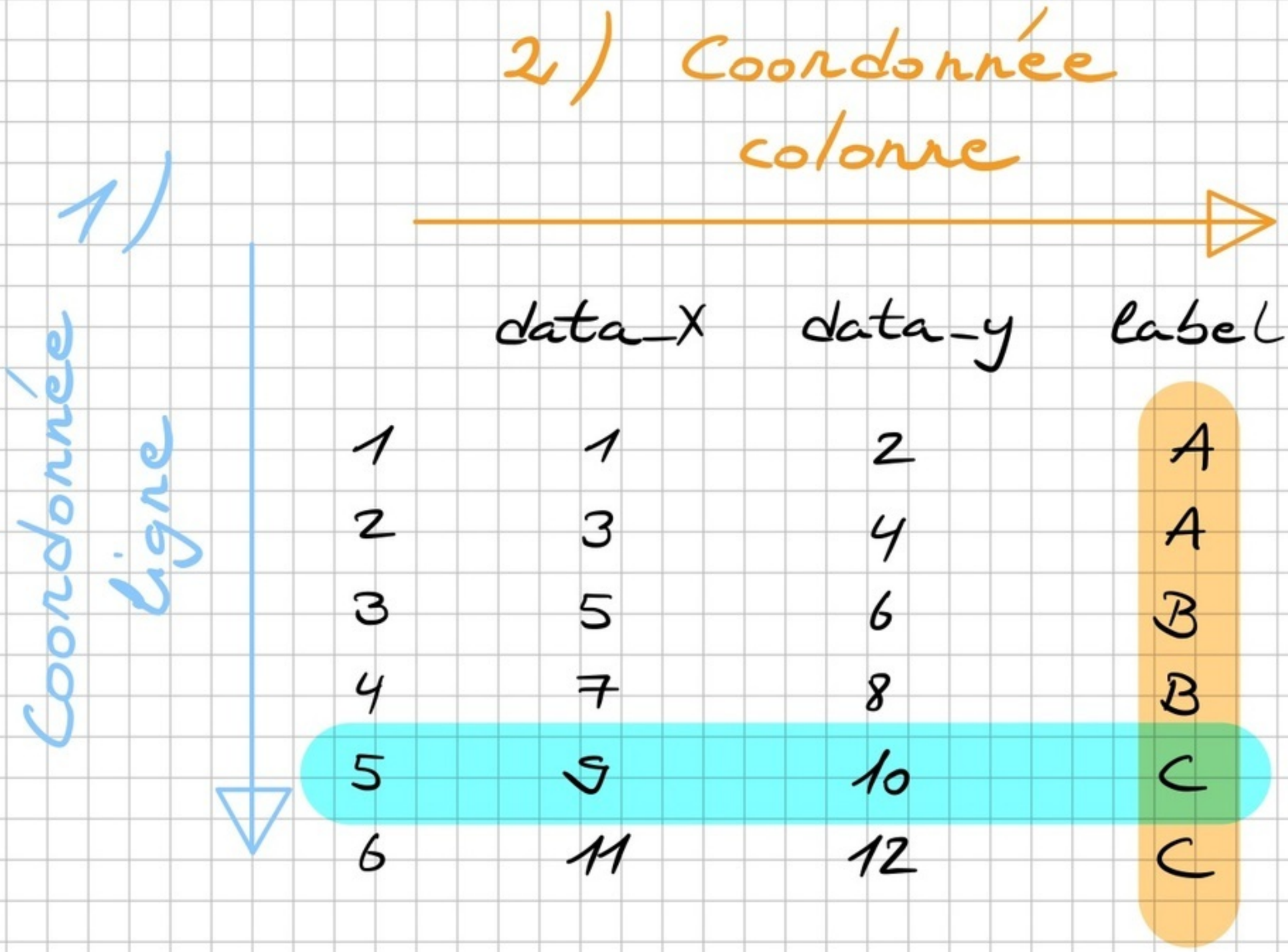


La reine  est en positon 6D

Un système de coordonnées

Les tableaux sont donc des espaces à deux dimentions dont les éléments sont accessibles à l'aide de deux coordonnées :

1) le numéro de la ligne
2) le numéro de la colonne



La valeur "C" se trouve à la 5e ligne, 3e colonne.

La syntaxe qui permet d'accéder à cet élément est la suivante : `data [5,3]`

Qui repose sur des conditions

Pour accéder aux éléments d'une structure de type dataframe, on utilise donc le numéro de la ligne et celui de la colonne que l'on met entre crochets

data [5,3]

S'il est utile de se représenter les deux chiffres entre crochet comme des coordonnées il est plus utile encore de réaliser qu'il s'agit en réalité de conditions

La première est une condition sur la ligne alors que la deuxième est une condition sur la colonne.

[Condition ligne , Condition colonne]

Ce détail peut sembler anodin, mais nous allons voir que sur lui repose toute la souplesse / possibilités dont les dataframes sont capables.

Voici le "mécanisme" qui entre en jeux lorsque l'on invoque la commande

data [5,3]

R va commencer par parcourir toutes les ligne du datframe

Tant que la condition sur la ligne n'est pas respectée (ligne = 5) R passe à la ligne suivante

Quand la condition sur la ligne est respectée, R s'arrête et commence à parcourir les colonnes.

Le même processus se reproduit maintenant avec les colonnes :

R parcourt les colonnes
Tant que la condition n'est pas respectée (colonne = 3) R passe a la suivante

Si la condition est respectée, R a suffisamment d'info pour situer l'élément

→ Afficher

Condition sur la ligne

Ceci étant dit, on peut dès lors comprendre comment la commande suivante affiche le contenu de toute une ligne, la 5e :

```
data[5,]
```

Cette commande dit : il y a une condition sur la ligne, (ligne = 5) mais il n'y à pas de condition sur la colonne (pas de 2e chiffre) Sans conditions sur la colonne, R va donc les afficher toute.

		data_x	data_y	label
	1	1	2	A
data[2,]	2	3	4	A
	3	5	6	B
	4	7	8	B
data[5,]	5	9	10	C
	6	11	12	C

```
data[2,]
     -->   2  3  4  A
data[5,]
     -->   5  9  10 C
```

Condition sur la colonne

Suivant la même logique, on peut tout aussi bien demander a R de n'afficher que la 3e colonne :

`data[,3]`

`data[,1]` → `data_x`

`data[,3]` → `label`

data_x	data_y	label
1	1	2	A
2	3	4	A
3	5	6	B
4	7	8	B
5	9	10	C
6	11	12	C

`data[,1]`

→ 1 3 5 7 9 11

`data[,3]`

→ A A B B C C

On comprend maintenant pourquoi la commande

```
data[]
```

affiche tout le contenu de la variable "data" ^_^

L'exemple précédent nous montre que l'on peut accéder au contenu d'une colonne en donnant simplement le numéro de la commande en question.

Il existe une deuxième manière de faire qui tire parti du label attribué à chaque colonne

Plutôt que de donner le numéro de colonne, on peut aussi donner son nom

Dans ce cas on écrira :

```
data[,"label"]
```

Ou plus simplement :

```
data$label
```

On peut donc accéder à une colonne soit par son numéro, soit par son label.

data_x	data_y	label
1	1	A
2	3	A
3	5	B
4	7	B
5	9	C
6	11	C

data[, "data_x"]

→ 1 3 5 7 9 11

data[, 1]

→ 1 3 5 7 9 11

Quelques Exemples

Voici quelques exemples de ce que l'on peut faire avec les conditions :

1) Afficher les lignes 1 à 4 :

```
data[1:4,]
```

2) Afficher la ligne 1 et 4

```
data[c(1,4),]
```

3) Afficher les colonnes 1 à 3

```
data[,1:3]
```

4) Afficher les colonnes 1 et 3

```
data[,c(1,3)]
```

5) Afficher la colonne nommée : "label"

```
data[,"label"]
```

On peut enfin combiner les conditions

6) Afficher les lignes 1 à 6 des colonnes 1 et 3 :

```
data[1:6,c(1,3)]
```

Utilisation des masques

Jusqu'à maintenant on s'est contenté de selectionner les lignes et les colonnes à l'aide de leur numéro et de leur label.

Le logiciel R permet bien plus de flexibilités que ca.

On peut par exemple poser une condition sur les lignes d'une colonne en particulier.

Exécutons par exemple la commande suivante

```
data$label == "A"
```

Cette commande donne à R l'instruction suivante :

> sous la colonne "label" du dataframe "data" selectionne les lignes qui contiennent "A"

Il s'agit donc d'une condition sur les lignes, mais au sein d'une colonne donnée.

Cette commande nous retourne :

→ T T F F F F

La lettre T signifie True (Vrai)
La lettre F signifie False (Faux)

La réponse de R suite a notre commande

data $ label == "A"

est donc la suivante :

La condition qui m'a été donnée est vraie sur la première ligne, vrai sur la deuxième et fausse partout ailleurs.

Ou, en d'autres termes plus concis :

T T F F F F

Notre dataframe contient 6 lignes, on se retrouve donc avec 6 résultats 1 pour chaque ligne du dataframe.

la colonne "label"

	data_X	data_y	label	
1	1	2	A	T
2	3	4	A	T
3	5	6	B	F
4	7	8	B	F
5	9	10	C	F
6	11	12	C	F

Cette suite de lettre constitue ce que l'on appel un masque.

Tous comme les masques dans la vraie vie, on s'en sert pour ne laisser voir que ce que l'on choisi de montrer

On peut donc s'en servir pour selectionner des lignes au sein du dataframe.

La commande :

```
data[data$label=="A",]
```

retourne donc :

	data_X	data_y	label
1	1	2	A
2	3	4	A

Simplifier l'écriture

On peut également écrire la commande de manière plus compacte en mettant le résultat de la condition dans une variable :

```
condition = data$label == "A"
```

la variable condition contient maintenant

```
T T F F F F
```

on peut dès lors s'en servire pour afficher selectivement les ligne au sein du dataframe.

```
data[condition,]
```

Cette commande est absolument identique à :

```
data[data$label == "A",]
```

et produit donc le même résultat qu'auparavent.

Combiner les conditions

Avec ce qu'on a appris jusqu'à maintenant on peut faire des dingueries !

Par exemple, utiliser la condition sur la colonne "label" mais n'afficher que le contenu de la colonne "data_y"

```
data[data$label == "A", "data_y"]
```

Condition sur la ligne

Condition sur la colonne

	data_x	data_y	label
1	1	2	A
2	3	4	A
3	5	6	B
4	7	8	B
5	9	10	C
6	11	12	C

```
data[data$label == "A", "data_y"]
```

→ 2 4

On peut aussi trier selon les valeurs numériques d'une colonne en particulier

```
data$data_X < 6
```

On exprime ici la condition que les valeurs de la colonnes "data_X" doivent être plus petites que 6.

data_X	data_y	label	
1	1	2	A	T
2	3	4	A	T
3	5	6	B	T
4	7	8	B	F
5	9	10	C	F
6	11	12	C	F

```
data [ data$data_X < 6 , ]
```

→

	data_X	data_y	label
1	1	2	A
2	3	4	A
3	5	6	B

Ou encore utiliser la condition sur la colonne "data_x" pour n'afficher que le contenu des colonnes 2 à 3

```
cond_ligne = data$data_X < 6
cond_colne = 2:3
data[cond_ligne, cond_colne]
```

	data_X	data_y	label
1	1	2	A
2	3	4	A
3	5	6	B
4	7	8	B
5	9	10	C
6	11	12	C

```
data[cond_ligne, cond_colne]
```

ce qui est équivalent à :

```
data[data$data_X < 6, 2:3]
```

→

	data_y	label
1	2	A
2	4	A
3	6	B

Conclusion

Si l'on avait par exemple un dataframe avec une colonne "genre" ayant pour valeurs "homme" et "femme", nous sommes capable maintenant de selectionner les observations faites sur les femmes spécifiquement.

```
data[data$genre == "femme", ]
```

Et c'est ainsi qu'il est possible de trier et selectionner les données au sein d'un dataframe selon les valeurs d'une colonne en particulier.