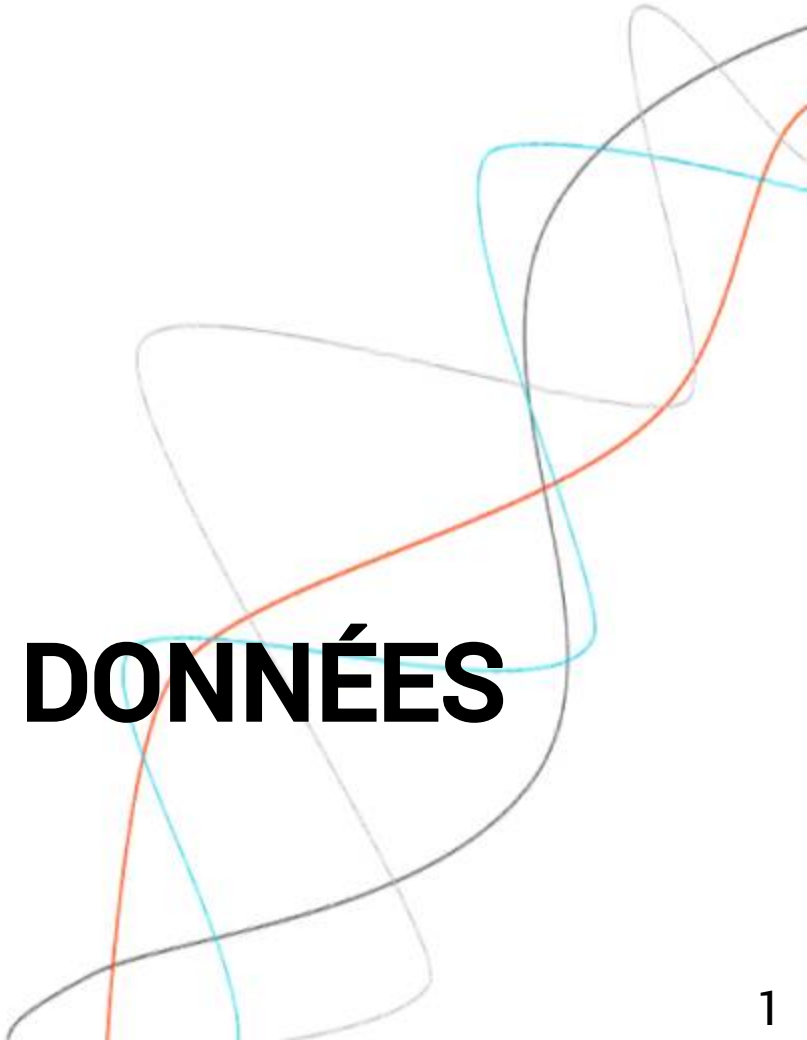
MANIPULATION DE DONNÉES AVEC {dplyr}



Qu’est-ce que {dplyr} ?

- Un package par Hadley Wickham (encore lui !) accéléré par Romain François

- Une « grammaire » de manipulation de données performante…

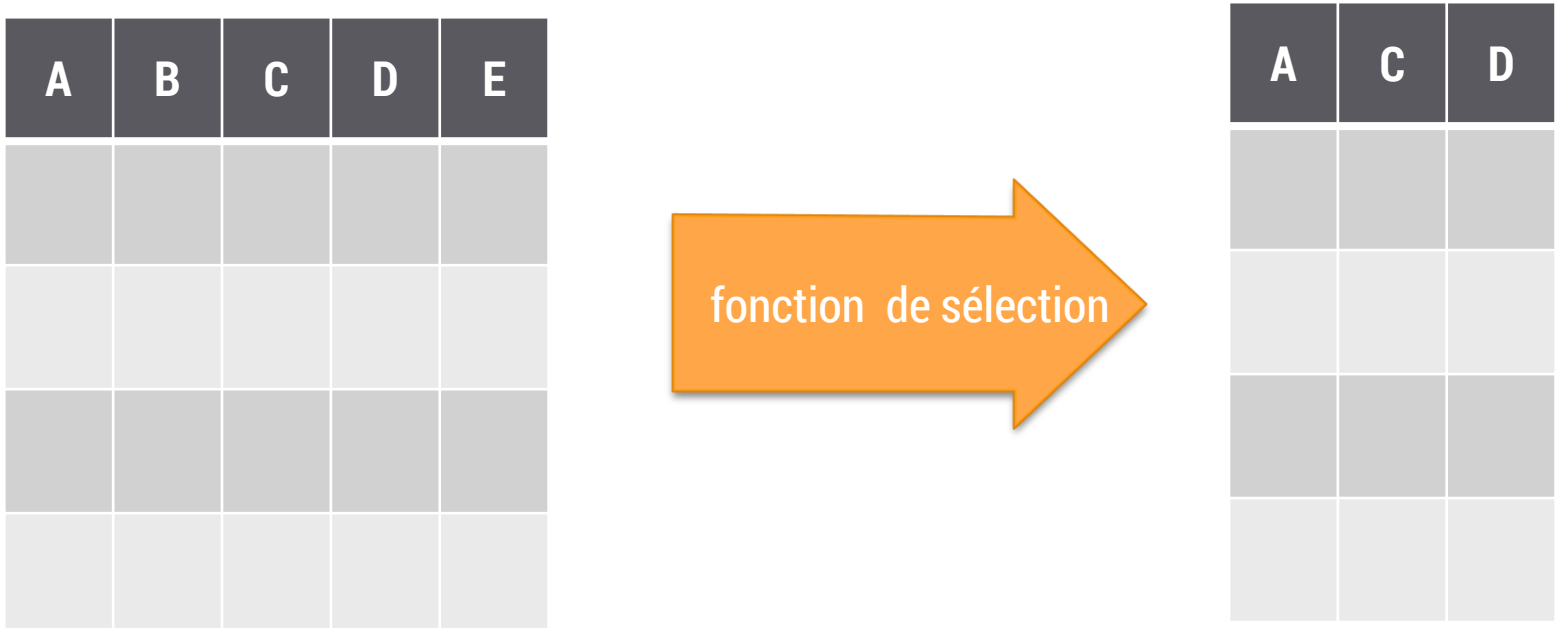
- … potentialisée par l’utilisation du « pipe » (%>%)…

- …pour écrire du code lisible, explicite.

La grammaire {dplyr}

Elle est organisé autour de « verbes » qui facilitent les opérations courantes sur les lignes et les colonnes :

- select() : sélectionne des colonnes, réduit le jeu de données en largeur



La grammaire {dplyr}

Elle est organisé autour de « verbes » qui facilitent les opérations courantes sur les lignes et les colonnes :

- filter() : filtre sur les lignes, réduit le jeu de données en hauteur

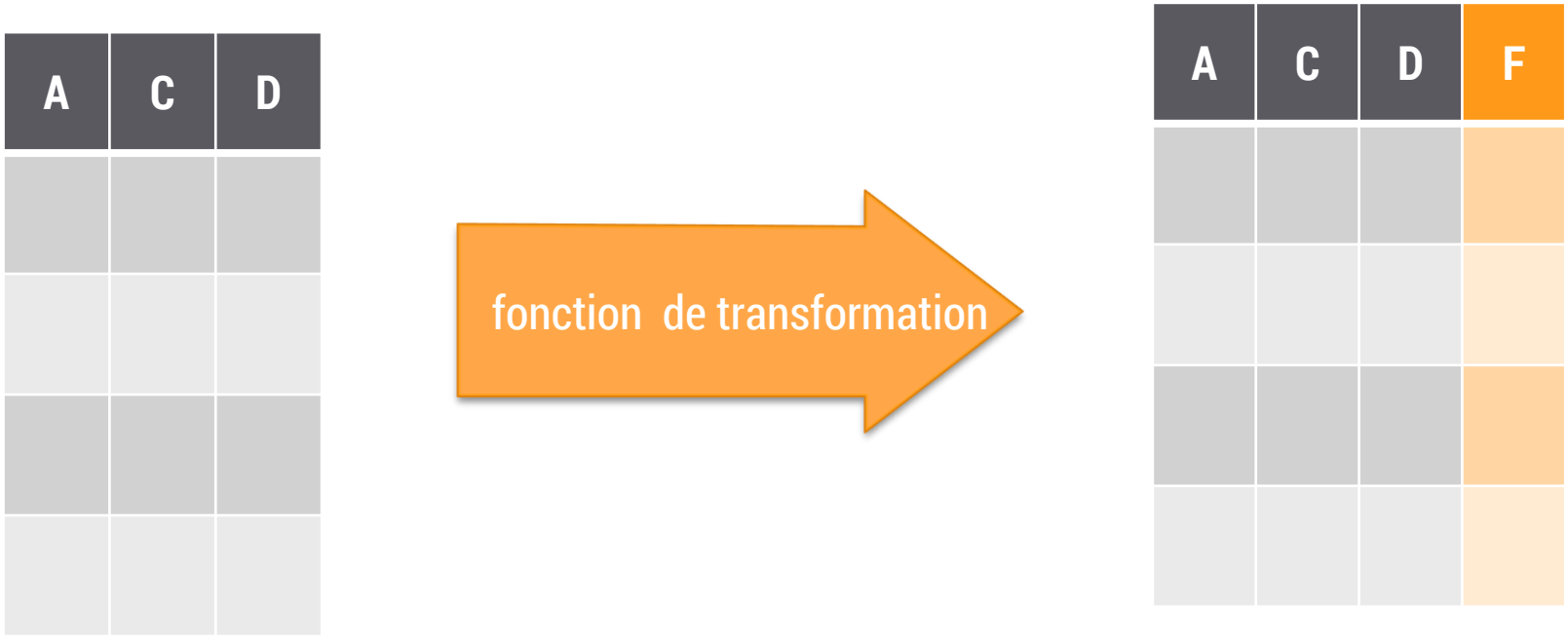
lignes	A	B	C
1			
2			
3			
4			

lignes	A	B	C
1			
3			

La grammaire {dplyr}

Elle est organisé autour de « verbes » qui facilitent les opérations courantes sur les lignes et les colonnes :

- mutate() : crée une variable, élargit le jeu de données



La grammaire {dplyr}

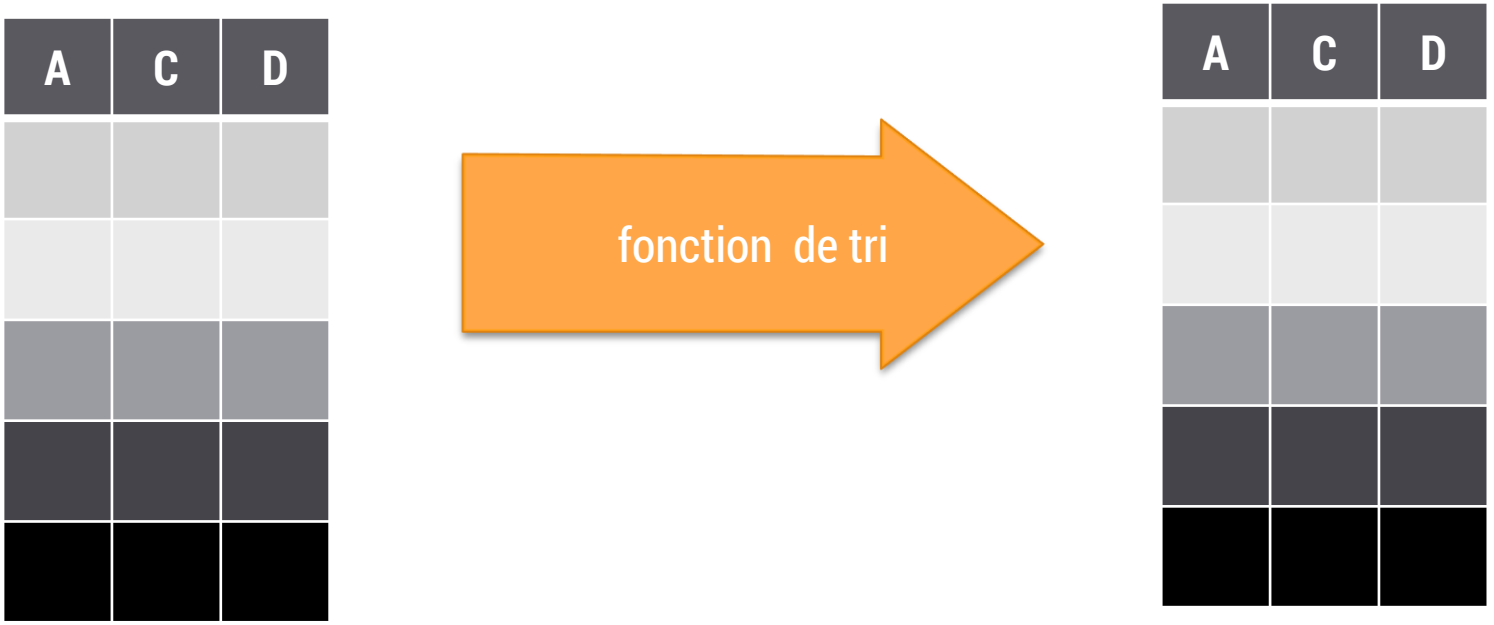
Elle est organisé autour de « verbes » qui facilitent les opérations courantes sur les lignes et les colonnes :

- summarise() : résume le jeu de données en un tableau plus petit

La grammaire {dplyr}

Elle est organisé autour de « verbes » qui facilitent les opérations courantes sur les lignes et les colonnes :

- arrange() : réordonne le jeu de données



> Live code session

IMPORT DES DONNÉES

A propos de la class « tibble »

Les dataframe qui sont renvoyés par une des fonctions d'import du « hadleyverse » (read_excel, read_csv) se voient appliquer une classe supplémentaire, ils deviennent des `tbl`

```
iris_tbl<-as.tbl(iris)
class(iris)# [1] "data.frame"
class(iris_tbl)  #[1] "tbl_df" "tbl" "data.frame "
```

L'affichage de ces bases de données est améliorée, seule les premières lignes sont affichées, un nombre optimal de colonne est présenté et le type de chaque variable est indiqués.

Dans les tbl , dans le cas d'une sélection d'une unique colonne c'est toujours un tbl qui est renvoyé, et non plus un vecteur. Plus besoin d'utiliser drop=FALSE.

```
class(iris[,1]) # "numeric"
class(iris[,1,drop=FALSE]) # "data.frame"
class(iris_tbl[,1])# "tbl_df" "data.frame"
```

Cependant toutes les fonction de R ne savent pas encore les prendre en charge, il est possible de supprimer la class tbl grace a

```
as.data.frame(iris_tbl)
```

DESCRIPTION DE SELECT(), FILTER() ET ARRANGE()

La grammaire {dplyr} – select()

La syntaxe de la fonction select() est de la forme :

```
select( données, fonctions_de_selection, … )
```

```
library(dplyr)
head(select(iris, Petal.Length, Petal.Width))
head(select(iris, -Petal.Length, -Petal.Width))
head(select(iris, -starts_with("Petal")))
head(select(iris, starts_with("Petal")))
head(select(iris, -ends_with("Width")))
head(select(iris, -contains("etal")))
head(select(iris, -matches(".t.")))
head(select(iris,Sepal.Width:Petal.Width))
```

La grammaire {dplyr} – filter()

La syntaxe de la fonction filter() est de la forme :

```
filter( données, fonctions_de_filtre, … )
```

```
filter(iris,Species=="virginica" , Petal.Width<3)
filter(iris,Species=="virginica" & Petal.Width<3)
filter(iris,Species=="virginica" | Petal.Width>1.3)
```

La grammaire {dplyr} – arrange()

La syntaxe de la fonction filter() est de la forme :

```
arrange( données, fonctions_de_tri, … )
```

```
arrange(iris, Petal.Length)
arrange(iris, desc( Petal.Length))
```

> Live code session

TRAVAUX PRATIQUES !

DESCRIPTION DE MUTATE() ET SUMMARISE()

La grammaire {dplyr} – mutate()

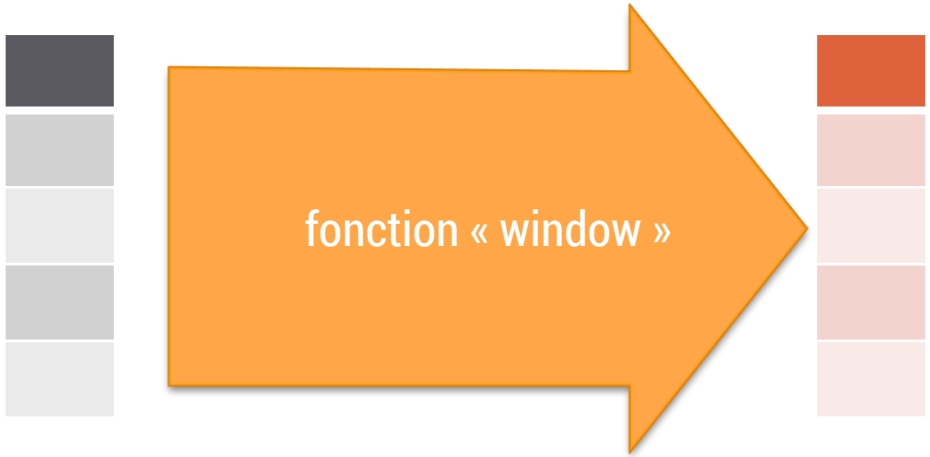
La syntaxe de la fonction mutate() est de la forme :

```
mutate( données, nouvelle_variable = operations(), …)
```

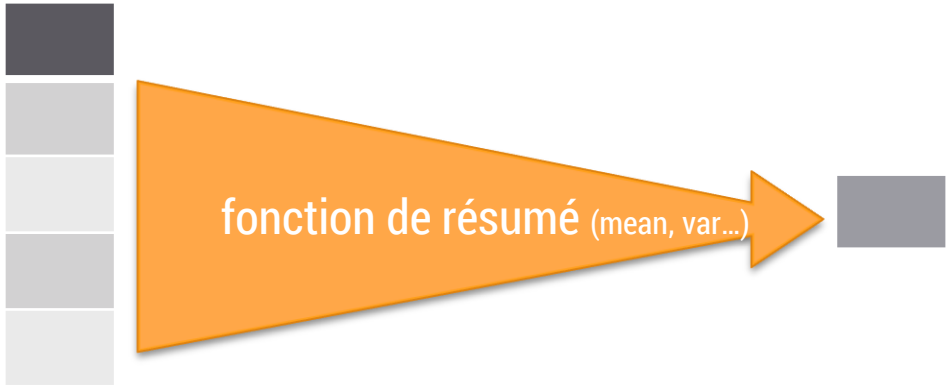
```
mutate(iris,
ratio_Sepal = Sepal.Length/Sepal.Width,
ratio_Petal = Petal.Length/Petal.Width)
```

Fonctions « windows » vs fonctions de résumé

- mutate() est une fonction dite «window » : elle prends en entrée un vecteur et rend un vecteur



- summarise() est une fonction de résumé : elle prends en entrée un vecteur et ne renvoie qu'une seule information, un résumé du vecteur



Exemple de summarise()

```
summarise(iris, moyenne = mean(Sepal.Length))
```

```
  moyenne
1 5.843333
```

Exemple de summarise()

```
summarise(iris, moyenne = mean(Sepal.Length))
```

```
  moyenne
1 5.843333
```

```
summarise(iris, moyenne = mean(Sepal.Length), variance
= var(Sepal.Length),nombre=n())
```

```
  moyenne variance   nombre
1 5.843333 0.6856935  150
```

> Live code session

TRAVAUX PRATIQUES !

QUAND IL Y A TROP DE VARIABLES A TRANSFORMER OU RÉSUMER (ET QU'ON A LA FLEMME)

Summarise_each() & mutate_each()

Summarise_each et mutate_each permettent de condenser l'écriture et d'appliquer une ou plusieurs transformations à un groupe de variables

```
summarise_each(iris,funs(mean),Sepal.Length:Petal.Width)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1 5.843333     3.057333    3.758        1.199333
```

```
summarise_each(iris,funs(var),starts_with("Sepal"))
```

```
  Sepal.Length Sepal.Width
 1 0.6856935    0.1899794
```

```
summarise_each(iris,funs(var,mean),starts_with("Sepal"))
```

```
Sepal.Length_var Sepal.Width_var Sepal.Length_mean
Sepal.Width_mean
1 0.6856935      0.1899794       5.843333          3.057333
```

Summarise_each() & mutate_each()

Lorsque l'on est amené à utiliser la fonction funs() il est parfois nécessaire de passer des arguments aux fonctions appelées. Pour cela il faut utiliser la notation suivante :

funs(mean(.,na.rm=TRUE)) au lieu de funs(mean)

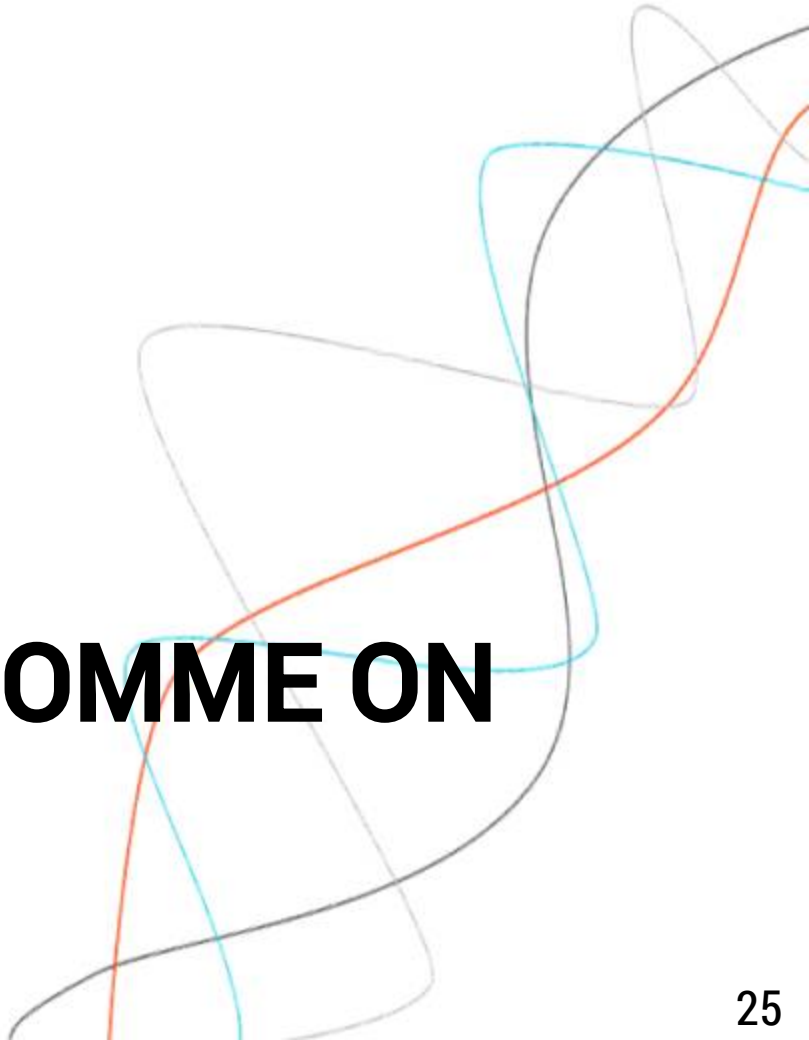
```
iris_avec_NA <- iris
iris_avec_NA[1,"Sepal.Length"]<-NA ; head(iris_avec_NA)

summarise_each(iris_avec_NA,funs(mean),Sepal.Length:Petal.Width)

  Sepal.Length Sepal.Width Petal.Length Petal.Width
1           NA    3.057333        3.758    1.199333

summarise_each(iris_avec_NA,funs(mean(.,na.rm=TRUE)),Sepal.Lengt
h:Petal.Width)

  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.843333    3.057333        3.758    1.199333
```

> Live code session

TRAVAUX PRATIQUES !

Ecrire du code élégamment

CODER PRESQUE COMME ON PARLE



Coder (presque) comme on parle avec magrittr (1)

Problème : lorsqu'on veut enchaîner les opérations, on les imbrique et le code devient rapidement long, illisible, sujet à erreurs de parenthèses

Solution : le « pipe » magrittr ou %>% (Ctrl+Shift+M dans RStudio)

Comment ça marche ?
Le terme à **gauche** de %>% devient le premier argument de la fonction à sa **droite**

Ex :
filter(iris, Petal.Length > 3)
iris%>%filter(Petal.Length > 3)

Coder (presque) comme on parle avec magrittr (2)

verbe(sujet , complément, ...)

sujet %>% verbe(complément, ...)

Coder (presque) comme on parle avec magrittr (3)

Dans iris, filtrer les individus de type setosa, ne conserver que les variables quantitatives et construire une variable surface qui additionne la surface de pétales et la surface de sépales :

Version imbriquée :

```
mutate(select(filter(iris, Species=="setosa"), -Species), surface_totale =
Petal.Length*Petal.Width + Sepal.Length*Sepal.Width)
```

Version magrittr :

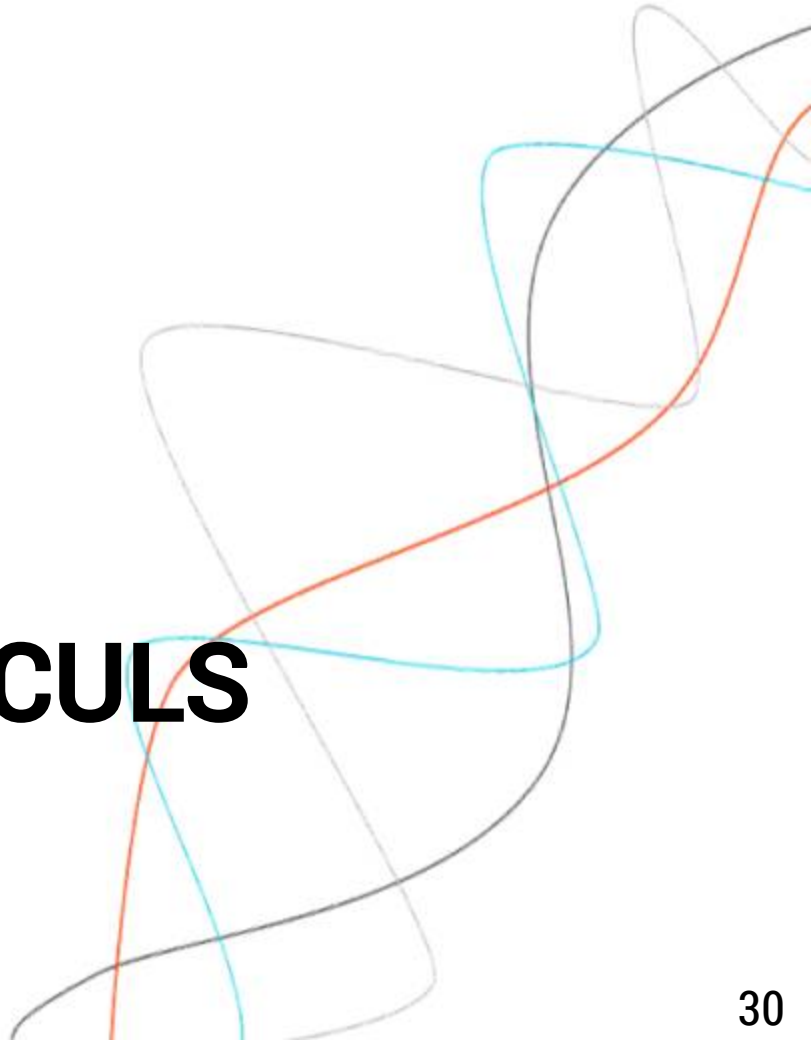
```
iris%>% filter(Species=="setosa")%>%
        select(-Species)%>%
        mutate(surface_totale = Petal.Length*Petal.Width +
        Sepal.Length*Sepal.Width)
```

> Live code session

TRAVAUX PRATIQUES !

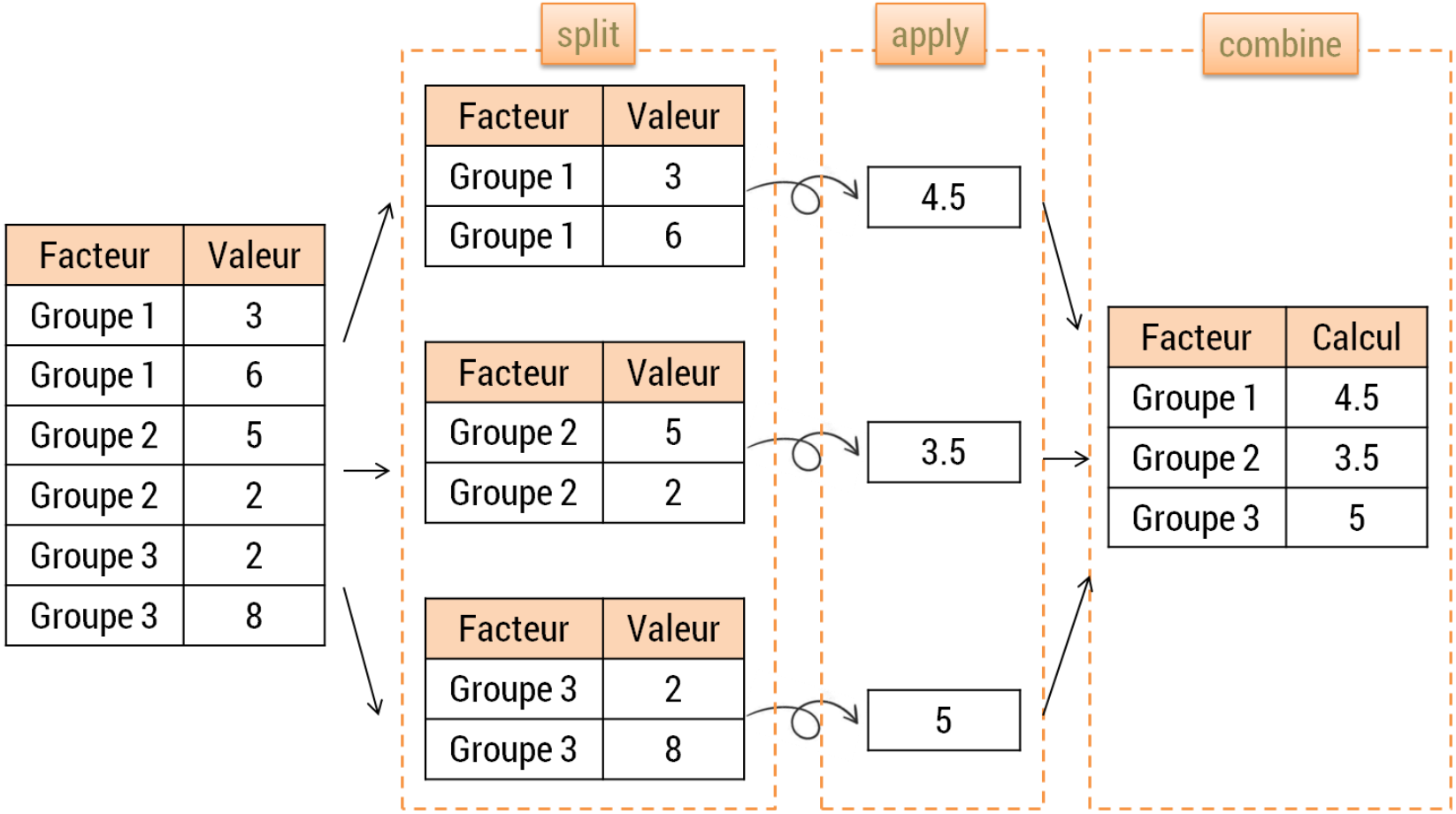
Appliquer la stratégie du split-apply-combine

GROUPER LES CALCULS



La stratégie du « split – apply – combine » (1)

Split - Séparer le jeu de données en sous-éléments,…
Apply - …appliquer une fonction sur chacun et…
Combine - …recombiner les résultats



La stratégie du « split – apply – combine » (2)

En grammaire dplyr, c'est la fonction `group_by()` qui permet de réaliser l'opération de « split » et c'est `summarise()` qui se charge de réaliser les fonctions de résumé à appliquer aux sous-ensembles

Exemple :

```
iris %>% group_by(Species) %>%
        summarise(moyenne_Sepal.Length=mean(Sepal.Length),
        variance_Sepal.Length=var(Sepal.Length))
```

```
Source: local data frame [3 x 3]
Species moyenne_Sepal.Length variance_Sepal.Length
(fctr)        (dbl)   (dbl)
1 setosa     5.006  0.1242490
2 versicolor 5.936  0.2664327
3 virginica  6.588  0.4043429
```

> Live code session

TRAVAUX PRATIQUES !