



THINKR

Le 'tidyeval'

ou comment programmer dans le {tidyverse}



Vincent Guyader

Data Scientist, expert logiciel R.

- <https://thinkr.fr>
- http://twitter.com/thinkr_fr
- <https://github.com/ThinkR-open>



Vincent Guyader
Codeur Fou, formateur et expert
logiciel R



Sébastien Rochette
Modélisateur, Formateur R, Joueur de
cartographies



Diane Beldame
Dompteuse de dragons données,
formatrice logiciel R



Colin Fay
Data Analyst, Formateur R, expert
Social Media

C'est quoi le {tidyverse} ?



Tidyverse

[Packages](#) [Articles](#) [Learn](#) [Help](#) [Contribute](#)



R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying philosophy and common APIs.

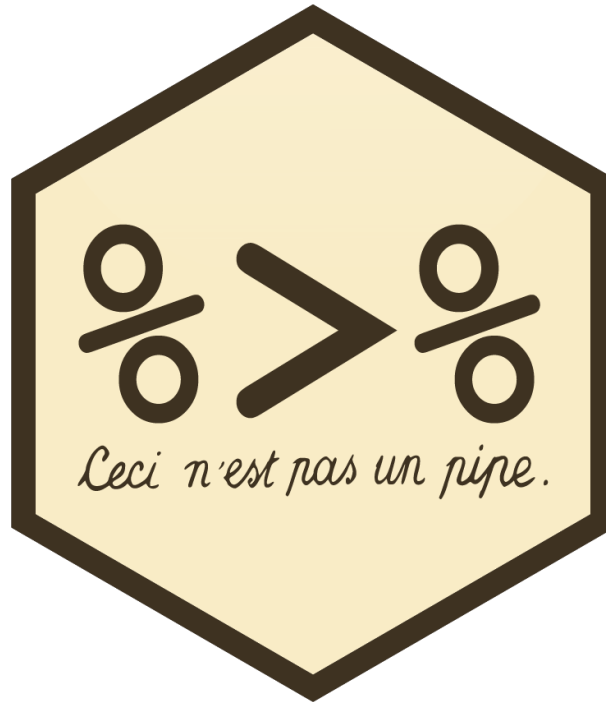
Install the complete tidyverse with:

```
install.packages("tidyverse")
```



Écrire comme on parle avec le %>%

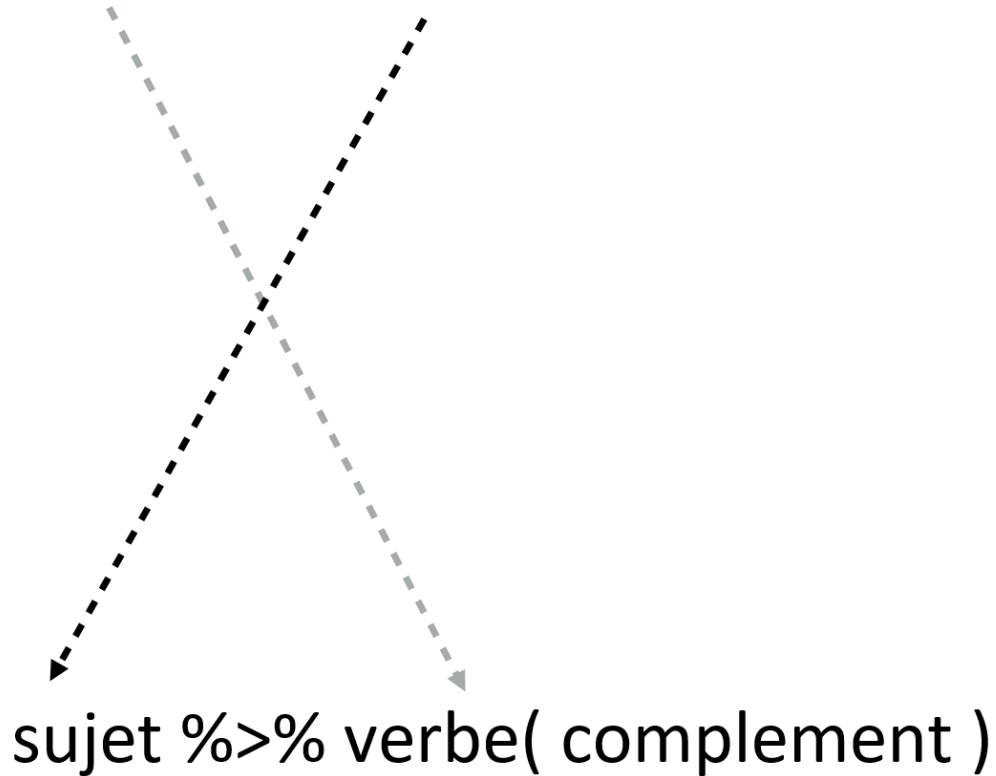
- à prononcer "païpe"
- raccourci clavier : "Ctrl + Shift + M"





Une question de grammaire

verbe(sujet, complement)





arrange() trie les observations...

A	C	D
Black	Black	Light Gray
Light Gray	Medium Gray	Light Gray
Medium Gray	Light Gray	Medium Gray
Dark Gray	Light Gray	Black
Light Gray	Light Gray	Light Gray



A	C	D
Light Gray	Medium Gray	Light Gray
Light Gray	Light Gray	Light Gray
Medium Gray	Light Gray	Medium Gray
Dark Gray	Light Gray	Black
Black	Black	Light Gray

```
iris %>% arrange( Sepal.Length )  
iris %>% arrange( desc(Sepal.Length), Petal.Length )
```



filter() réduit la hauteur du dataset

lignes	A	B	C
1			
2			
3			
4			



lignes	A	B	C
1			
3			

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



Réduire le jeu de données en largeur : `select()`

A	B	C	D	E



A	C	D

```
iris %>% select( Species, Sepal.Width, Petal.Width )
```




mutate () pour transformer ou créer des variables

mutate () modifie le jeu de données en largeur pour y adjoindre des colonnes, et/ou modifier des variables existantes.

A	C	D

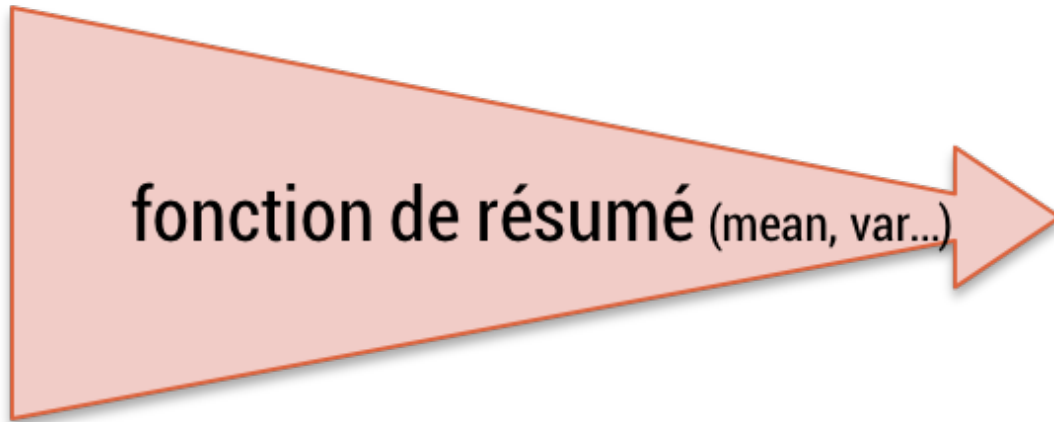


A	C	D	F

```
iris %>%  
  mutate( ratio_sepal = Sepal.Length / Sepal.Width)
```



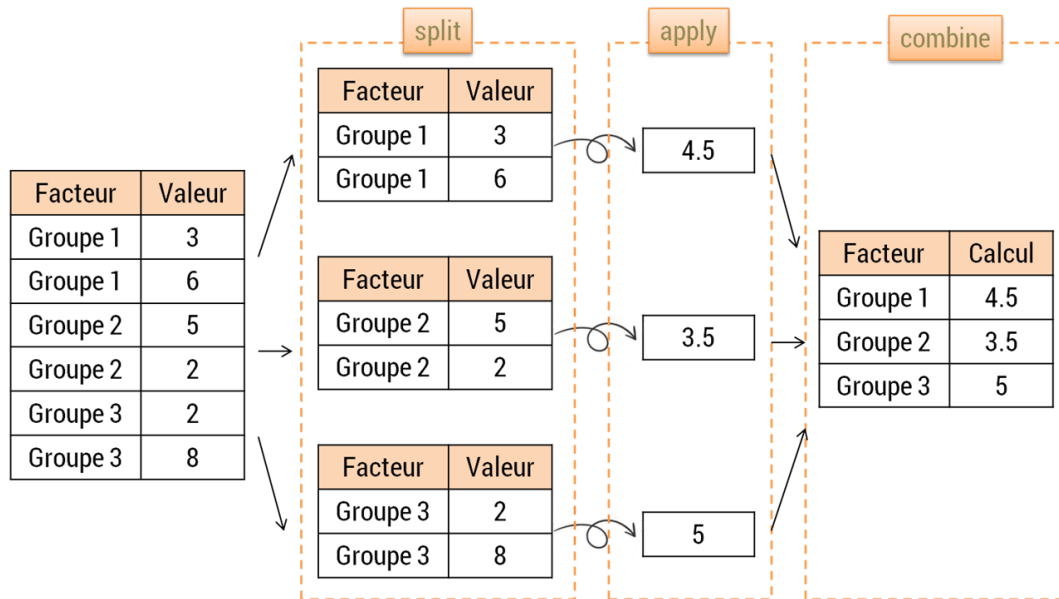
Résumer l'information avec : `summarise()`



```
iris %>%  
  summarise( moyenne = mean( Sepal.Length ) )
```



Résumer l'information **par groupe** avec : `group_by` & `summarise` ()



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



```
iris %>%  
  arrange( Sepal.Width ) %>%  
  filter( Sepal.Width >3 ) %>%  
  select( -Sepal.Width ) %>%  
  group_by (Species ) %>%  
  summarise( nombre = n() )
```

```
#> # A tibble: 3 x 2  
#>   Species     nombre  
#>   <fct>      <int>  
#> 1 setosa         42  
#> 2 versicolor    8  
#> 3 virginica     17
```

...C'est simple, (basique) et lisible !

Faisons une fonction du script précédent





Faisons une fonction du script précédent

```
filter_and_count <- function(dataset, variable){  
  
  dataset %>%  
  arrange( variable ) %>%  
  filter( variable > 3 ) %>%  
  select( -variable ) %>%  
  group_by( Species ) %>%  
  summarise( nombre = n() )  
  
}
```



Faisons une fonction du script précédent

```
filter_and_count <- function(dataset, variable){  
  
  dataset %>%  
  arrange( variable ) %>%  
  filter( variable > 3 ) %>%  
  select( -variable ) %>%  
  group_by( Species ) %>%  
  summarise( nombre = n() )  
  
}
```

```
iris %>% filter_and_count(Sepal.Length)
```

```
#> Error in arrange_impl(.data, dots): objet 'Sepal.Length' introuvable
```



Faisons une fonction du script précédent

```
filter_and_count <- function(dataset, variable){  
  
  dataset %>%  
  arrange( variable ) %>%  
  filter( variable > 3 ) %>%  
  select( -variable ) %>%  
  group_by( Species ) %>%  
  summarise( nombre = n() )  
  
}
```

```
iris %>% filter_and_count(Sepal.Length)
```

```
#> Error in arrange_impl(.data, dots): objet 'Sepal.Length' introuvable
```

```
iris %>% filter_and_count("Sepal.Length")
```

```
#> Error in arrange_impl(.data, dots): incorrect size (1) at position 1,  
expecting : 150
```




NSE de quoi ??

Quelle est la différence entre ces 2 instructions ?

```
library("dplyr")
```

et

```
library(dplyr)
```



NSE de quoi ??

Quelle est la différence entre ces 2 instructions ?

```
library("dplyr")
```

et

```
library(dplyr)
```

Ok, mais alors que fait le code suivant ?

```
dplyr <- "data.table"  
library(dplyr)
```



NSE de quoi ??

Quelle est la différence entre ces 2 instructions ?

```
library("dplyr")
```

et

```
library(dplyr)
```

Ok, mais alors que fait le code suivant ?

```
dplyr <- "data.table"  
library(dplyr)
```

Et celui ci ?

```
y <- "data.table"  
library(y)
```



NSE de quoi ??

Quelle est la différence entre ces 2 instructions ?

```
library("dplyr")
```

et

```
library(dplyr)
```

Ok, mais alors que fait le code suivant ?

```
dplyr <- "data.table"  
library(dplyr)
```

Et celui ci ?

```
y <- "data.table"  
library(y)
```

La fonction `library` utilise le paramètre `character.only`

```
dplyr <- "data.table"  
library(dplyr) #charge {dplyr}  
library(dplyr,character.only = TRUE) #charge {data.table}
```



NSE + {dplyr} dans ses propres fonctions ?

Dans {dplyr}, tous les paramètres (sauf `.data`) sont évalués de manière non standard



NSE + {dplyr} dans ses propres fonctions ?

Dans {dplyr}, tous les paramètres (sauf `.data`) sont évalués de manière non standard

Pour créer une fonction qui utilise et se comporte comme les fonctions du {tidyverse}, il faut utiliser conjointement 2 outils :



NSE + {dplyr} dans ses propres fonctions ?

Dans {dplyr}, tous les paramètres (sauf `.data`) sont évalués de manière non standard

Pour créer une fonction qui utilise et se comporte comme les fonctions du {tidyverse}, il faut utiliser conjointement 2 outils :

- La fonction **enquo()**



NSE + {dplyr} dans ses propres fonctions ?

Dans {dplyr}, tous les paramètres (sauf `.data`) sont évalués de manière non standard

Pour créer une fonction qui utilise et se comporte comme les fonctions du {tidyverse}, il faut utiliser conjointement 2 outils :

- La fonction **enquo()**
- Et l'opérateur **!!** (*à prononcer bang-bang*)

Construisons la fonction



```
filter_and_count <- function(dataset, variable){  
  
  dataset %>%  
  arrange( variable ) %>%  
  filter( variable > 3 ) %>%  
  select( -variable ) %>%  
  group_by( Species ) %>%  
  summarise( nombre = n() )  
  
}
```

Construisons la fonction



```
filter_and_count <- function(dataset, variable){  
  
  variable <- enquo(variable) # Pour 'quoter' la variable  
  
  dataset %>%  
  arrange( variable ) %>%  
  filter( variable > 3 ) %>%  
  select( -variable ) %>%  
  group_by( Species ) %>%  
  summarise( nombre = n() )  
  
}
```

Construisons la fonction



```
filter_and_count <- function(dataset, variable){  
  
  variable <- enquo(variable) # Pour 'quoter' la variable  
  
  dataset %>%  
  arrange( !! variable ) %>% # Pour 'unquoter' la variable  
  filter( !! variable > 3 ) %>%  
  select( - !! variable ) %>%  
  group_by( Species ) %>%  
  summarise( nombre = n() )  
  
}
```

Construisons la fonction



```
filter_and_count <- function(dataset, variable, seuil = 3){  
  
  variable <- enquo(variable) # Pour 'quoter' la variable  
  
  dataset %>%  
  arrange( !! variable ) %>% # Pour 'unquoter' la variable  
  filter( !! variable > seuil ) %>%  
  select( - !! variable ) %>%  
  group_by( Species ) %>%  
  summarise( nombre = n() )  
  
}
```

Ça marche plutôt bien



```
iris %>%  
  filter_and_count(Sepal.Width)
```

```
#> # A tibble: 3 x 2  
#>   Species     nombre  
#>   <fct>       <int>  
#> 1 setosa         42  
#> 2 versicolor    8  
#> 3 virginica     17
```

Ça marche plutôt bien



```
iris %>%  
filter_and_count(Sepal.Width)
```

```
#> # A tibble: 3 x 2  
#>   Species     nombre  
#>   <fct>      <int>  
#> 1 setosa         42  
#> 2 versicolor    8  
#> 3 virginica     17
```

```
iris %>%  
filter_and_count(Petal.Length)
```

```
#> # A tibble: 2 x 2  
#>   Species     nombre  
#>   <fct>      <int>  
#> 1 versicolor    49  
#> 2 virginica     50
```



Ca marche plutôt bien

```
iris %>%  
  filter_and_count(Sepal.Width)
```

```
#> # A tibble: 3 x 2  
#>   Species     nombre  
#>   <fct>      <int>  
#> 1 setosa         42  
#> 2 versicolor     8  
#> 3 virginica     17
```

```
iris %>%  
  filter_and_count(Sepal.Length,  
  seuil = 5.3)
```

```
#> # A tibble: 3 x 2  
#>   Species     nombre  
#>   <fct>      <int>  
#> 1 setosa         10  
#> 2 versicolor    45  
#> 3 virginica     49
```

```
iris %>%  
  filter_and_count(Petal.Length)
```

```
#> # A tibble: 2 x 2  
#>   Species     nombre  
#>   <fct>      <int>  
#> 1 versicolor    49  
#> 2 virginica     50
```



Rajoutons un paramètre...

...pour contrôler le nom de la colonne nombre

```
iris %>% filter_and_count(Sepal.Width, nom = comptage)
```

```
#> # A tibble: 3 x 2
#>   Species    comptage
#>   <fct>      <int>
#> 1 setosa         42
#> 2 versicolor    8
#> 3 virginica     17
```

Comment faire cela ?



Rajoutons un paramètre...

```
filter_and_count <- function(dataset, variable, seuil = 3, nom = nombre){  
  
  variable <- enquos(variable)  
  
  dataset %>%  
    arrange( !! variable ) %>%  
    filter( !! variable > seuil ) %>%  
    select( - !! variable ) %>%  
    group_by( Species ) %>%  
    summarise( nom = n() )  
}
```



Rajoutons un paramètre...

```
filter_and_count <- function(dataset, variable, seuil = 3, nom = nombre){  
  
  variable <- enquos(variable)  
  
  dataset %>%  
    arrange( !! variable ) %>%  
    filter( !! variable > seuil ) %>%  
    select( - !! variable ) %>%  
    group_by( Species ) %>%  
    summarise( nom = n() )  
}
```

```
iris %>% filter_and_count(Sepal.Width, nom = comptage)
```

```
#> # A tibble: 3 x 2  
#>   Species      nom  
#>   <fct>      <int>  
#> 1 setosa      42  
#> 2 versicolor  8  
#> 3 virginica   17
```



Rajoutons un paramètre...

```
filter_and_count <- function(dataset, variable, seuil = 3, nom = nombre){  
  
  variable <- enquos(variable)  
  nom <- enquos(nom)  
  dataset %>%  
    arrange( !! variable ) %>%  
    filter( !! variable > seuil ) %>%  
    select( - !! variable ) %>%  
    group_by( Species ) %>%  
    summarise( !! nom = n() )  
}
```

```
# Error: unexpected '=' in:  
# "    group_by( Species ) %>%  
#    summarise( !! nom ="  
# Error: unexpected '}' in "}"
```



Rajoutons un paramètre...

```
filter_and_count <- function(dataset, variable, seuil = 3, nom = nombre){  
  
  variable <- enquos(variable)  
  nom <- enquos(nom)  
  dataset %>%  
    arrange( !! variable ) %>%  
    filter( !! variable > seuil ) %>%  
    select( - !! variable ) %>%  
    group_by( Species ) %>%  
    summarise( !! nom := n() )  
}
```



Rajoutons un paramètre...

```
filter_and_count <- function(dataset, variable, seuil = 3, nom = nombre){  
  
  variable <- enquos(variable)  
  nom <- enquos(nom)  
  dataset %>%  
    arrange( !! variable ) %>%  
    filter( !! variable > seuil ) %>%  
    select( - !! variable ) %>%  
    group_by( Species ) %>%  
    summarise( !! nom := n() )  
}
```

```
iris %>% filter_and_count(Sepal.Width, nom = comptage)
```

```
#> Error: The LHS of `:=` must be a string or a symbol
```



Rajoutons un paramètre...

```
filter_and_count <- function(dataset, variable, seuil = 3, nom = nombre){  
  
  variable <- enquos(variable)  
  nom <- enquos(nom)  
  dataset %>%  
    arrange( !! variable ) %>%  
    filter( !! variable > seuil ) %>%  
    select( - !! variable ) %>%  
    group_by( Species ) %>%  
    summarise( !! quo_name(nom) := n() )  
}
```

```
iris %>% filter_and_count(Sepal.Width, nom = comptage)
```

```
#> # A tibble: 3 x 2  
#>   Species    comptage  
#>   <fct>      <int>  
#> 1 setosa         42  
#> 2 versicolor    8  
#> 3 virginica     17
```



Et Si ma fonction utilise les dots : . . . ?

Pour gérer le cas où une fonction doit être capable de prendre un nombre indéterminé de paramètres, il faut utiliser classiquement les . . . accompagnés de :

- la fonction **quos()**



Et Si ma fonction utilise les dots : . . . ?

Pour gérer le cas où une fonction doit être capable de prendre un nombre indéterminé de paramètres, il faut utiliser classiquement les . . . accompagnés de :

- la fonction **quos()**
- l'opérateur **!!!** (bang-bang-bang)

Exemple



```
mon_count <- function(dataset, ..., nom=nombre){  
  variables <- quos(...) # une liste de quosure  
  nom <- enquo(nom)  
  dataset %>%  
    group_by( !!! variables ) %>%  
    summarise( !! quo_name(nom) := n() )  
}
```



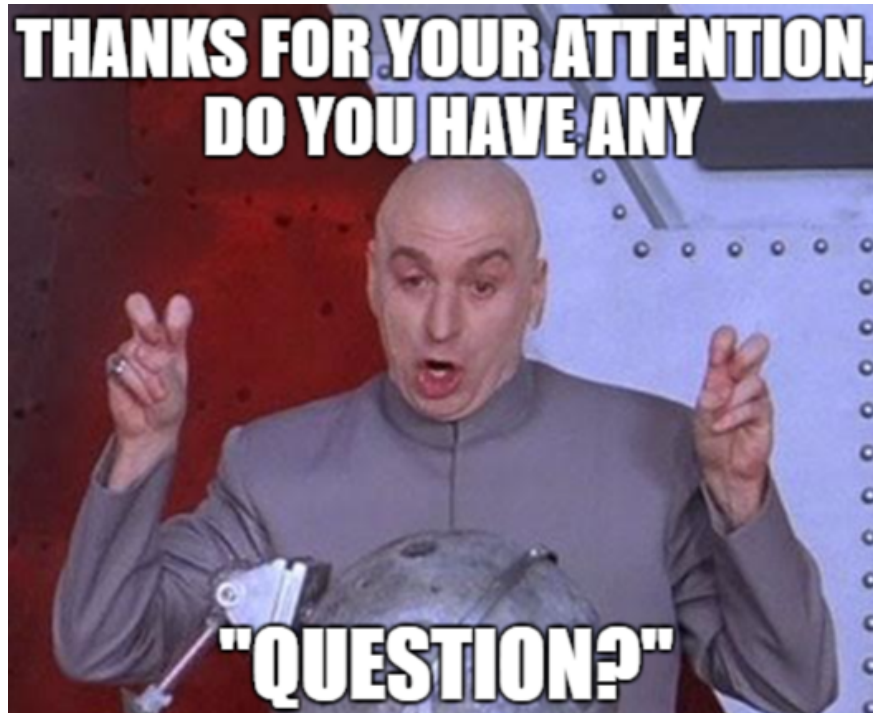
Exemple

```
mon_count <- function(dataset, ..., nom=nombre){
  variables <- quos(...) # une liste de quosure
  nom <- enquos(nom)
  dataset %>%
    group_by( !!! variables ) %>%
    summarise( !! quo_name(nom) := n() )
}
```

```
iris %>%
  mutate(couleur = sample(c("bleu", "rouge"), 150, replace = TRUE)) %>%
  mon_count(Species, couleur)
```

```
#> # A tibble: 6 x 3
#> # Groups:   Species [?]
#>   Species    couleur nombre
#>   <fct>      <chr>    <int>
#> 1 setosa     bleu        18
#> 2 setosa     rouge       32
#> 3 versicolor bleu        22
#> 4 versicolor rouge       28
```

Merci !



Me trouver sur le net:

- vincent@thinkr.fr
- <http://twitter.com/vincentguyader>
- http://twitter.com/thinkr_fr

Mais aussi:

- <https://thinkr.fr/>