Améliorer la courbe d'apprentissage de R au delà de `Tidyverse` ?

Les packages `chart` et `flow`

Philippe Grosjean & Guyliann Engels

Université de Mons, Belgique
Laboratoire d'Écologie numérique des Milieux aquatiques
<Philippe.Grosjean@umons.ac.be><https://github.com/SciViews>

UMONS

Qui sommes-nous?

- Biologistes marins (coraux, plancton) à l'Université de Mons en Belgique.

UMONS

Qui sommes-nous?

- Développeurs en R (mainteneurs de 17 packages sur CRAN dont `tcltk2`, `mlearning`, `pastecs`, `zooimage`, `SciViews`, `svDialogs`, etc.)
- Recherche sur l'évolution et la maintenance logicielles.

An Empirical Study of Identical Function Clones in CRAN

Maëlick Claes, Tom Mens, Narjisse Tabout, Philippe Grosjean
Software Engineering Lab & Ecologie numérique des Milieux aquatiques Lab
COMPLEXYS Research Institute, University of Mons
Email: firstname.lastname@umons.ac.be

***Abstract*—Code clone analysis is a very active subject of study, and research on inter-project code clones is starting to emerge. In the context of software package repositories specifically, developers are confronted with the choice between depending on code implemented in other packages, or cloning this code in their own package. This article presents an empirical study of identical function clones in the CRAN package archive network, in order to understand the extent of this practice in the R community. Depending on too many packages may hamper maintainability as unexpected conflicts may arise during package updates. Duplicating functions from other packages may reduce maintainability since bug fixes or code changes are not propagated automatically to its clones. We study how the characteristics of cloned functions in CRAN snapshots evolve over time, and classify these clones**

In particular, we want to understand to which extent functions are Type-1 clones across packages, why R package developers clone functions, and if clones could be avoided by the introduction of explicit dependencies. Thus our longitudinal empirical study of inter-package function clones in CRAN focuses on the following research questions:

1) How prevalent are clones in CRAN, and how does this evolve over time?
2) How and why did clones appear?
3) Is it possible to remove clones and how?

ONS

Qui sommes-nous?

- Enseignants, y compris biostatistiques et science des données (voir http://biodatascience-course.sciviews.org)

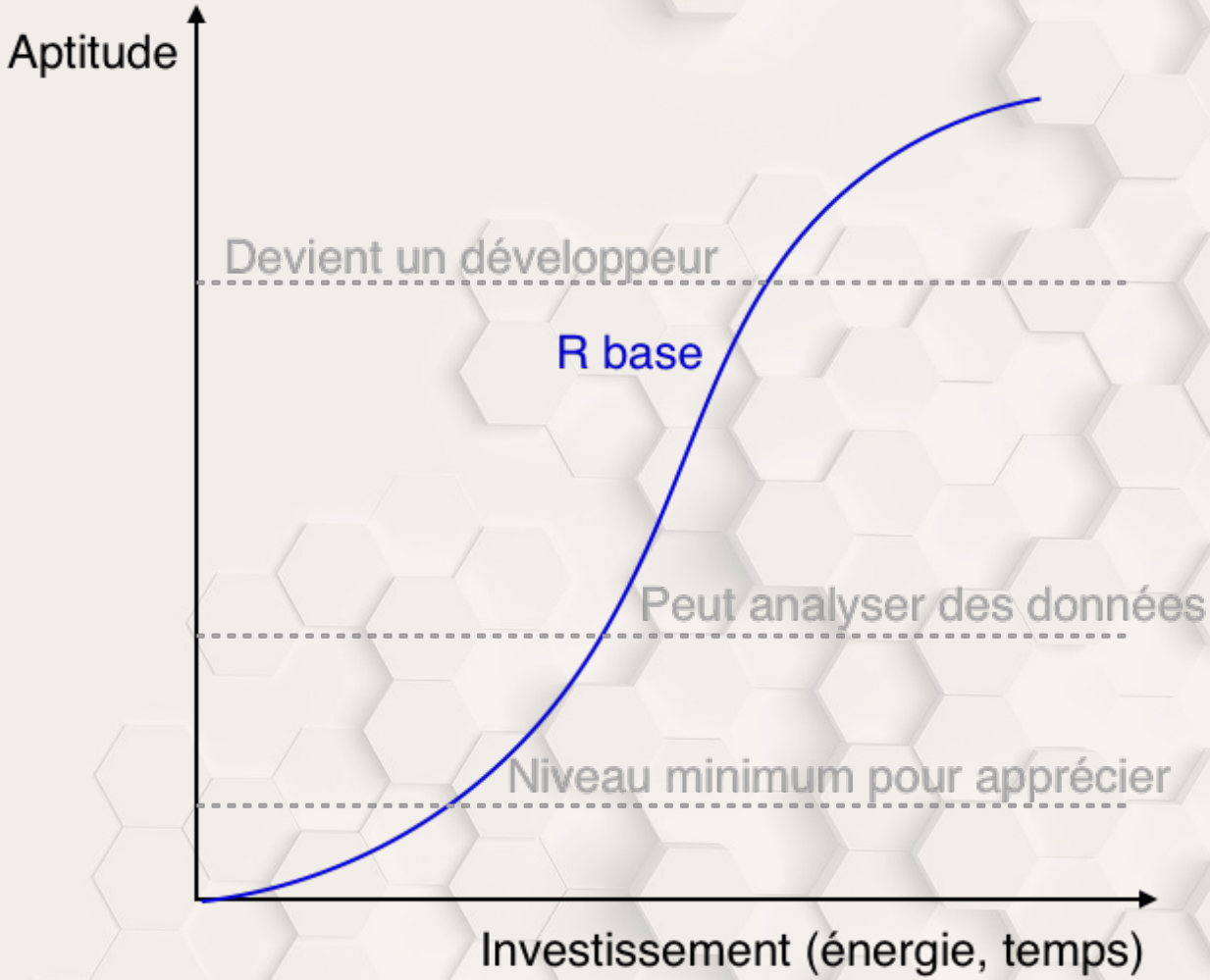
UMONS

En observant nos étudiants, nous en déduisons les aspects les plus difficiles dans l'apprentissage de R, et nous réfléchissons ensuite à la façon de les simplifier.

Deux exemples seront détaillés ici.

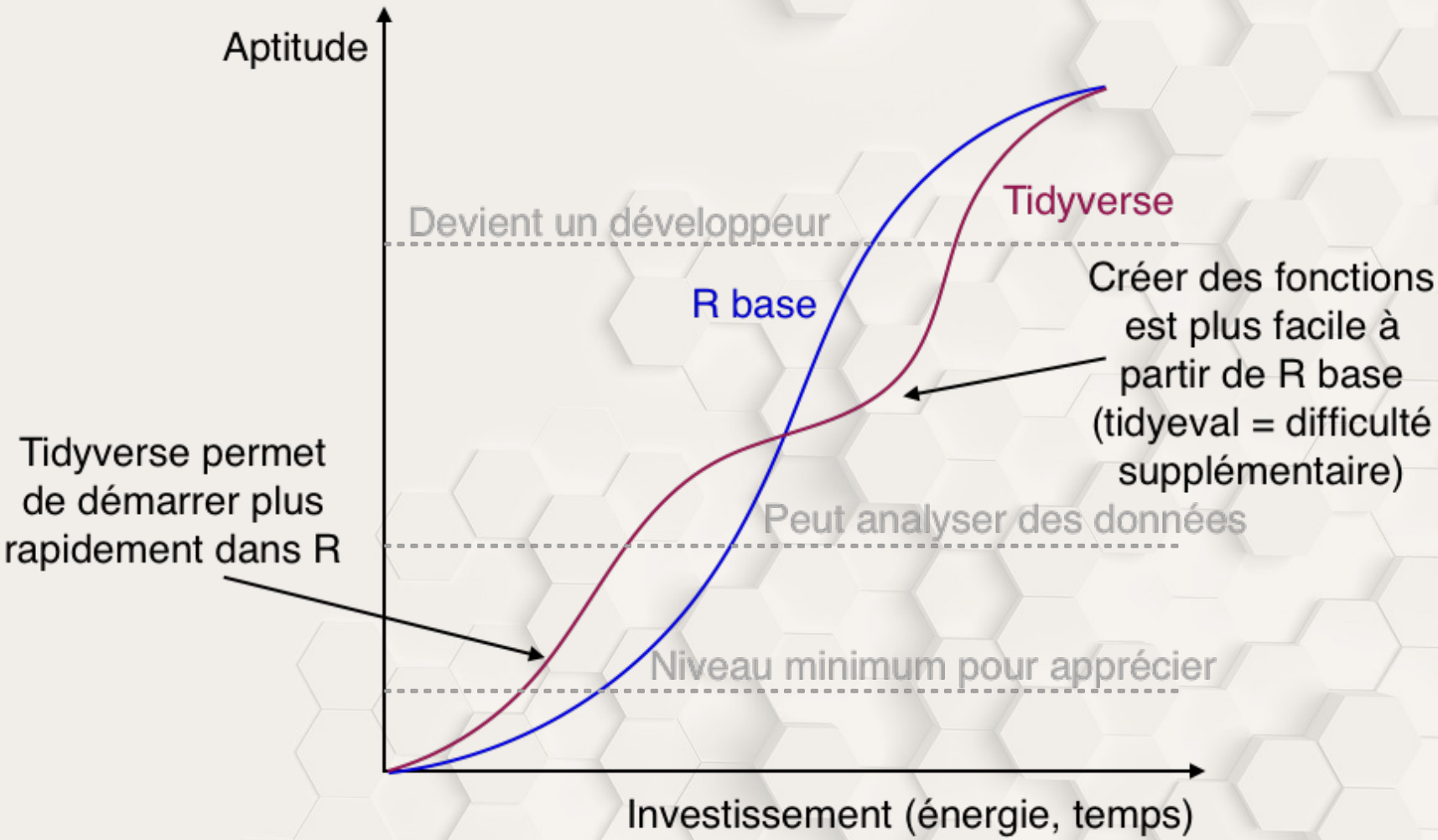
UMONS

Courbe d'apprentissage de R



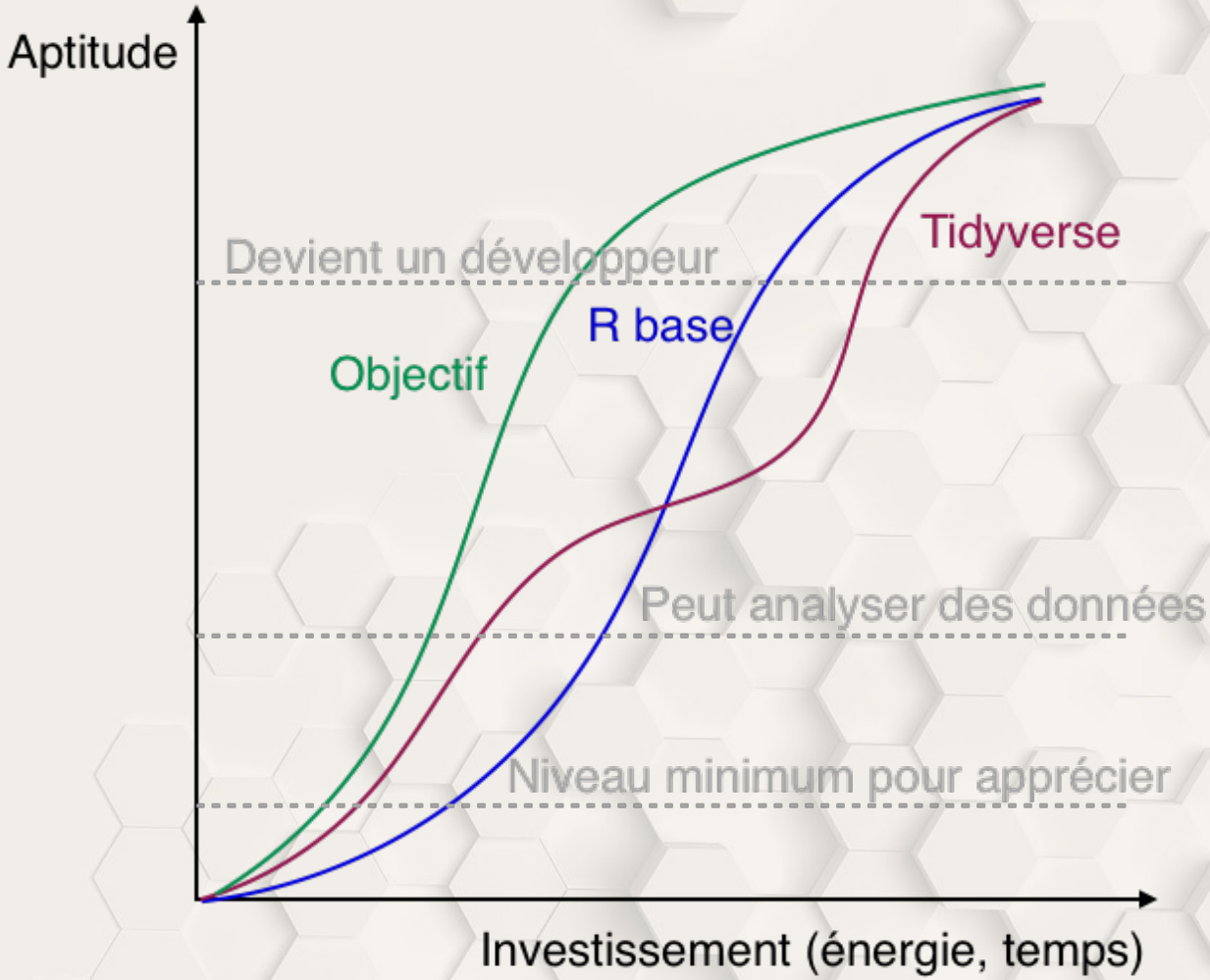
UMONS

Courbe d'apprentissage de R avec `Tidyverse`



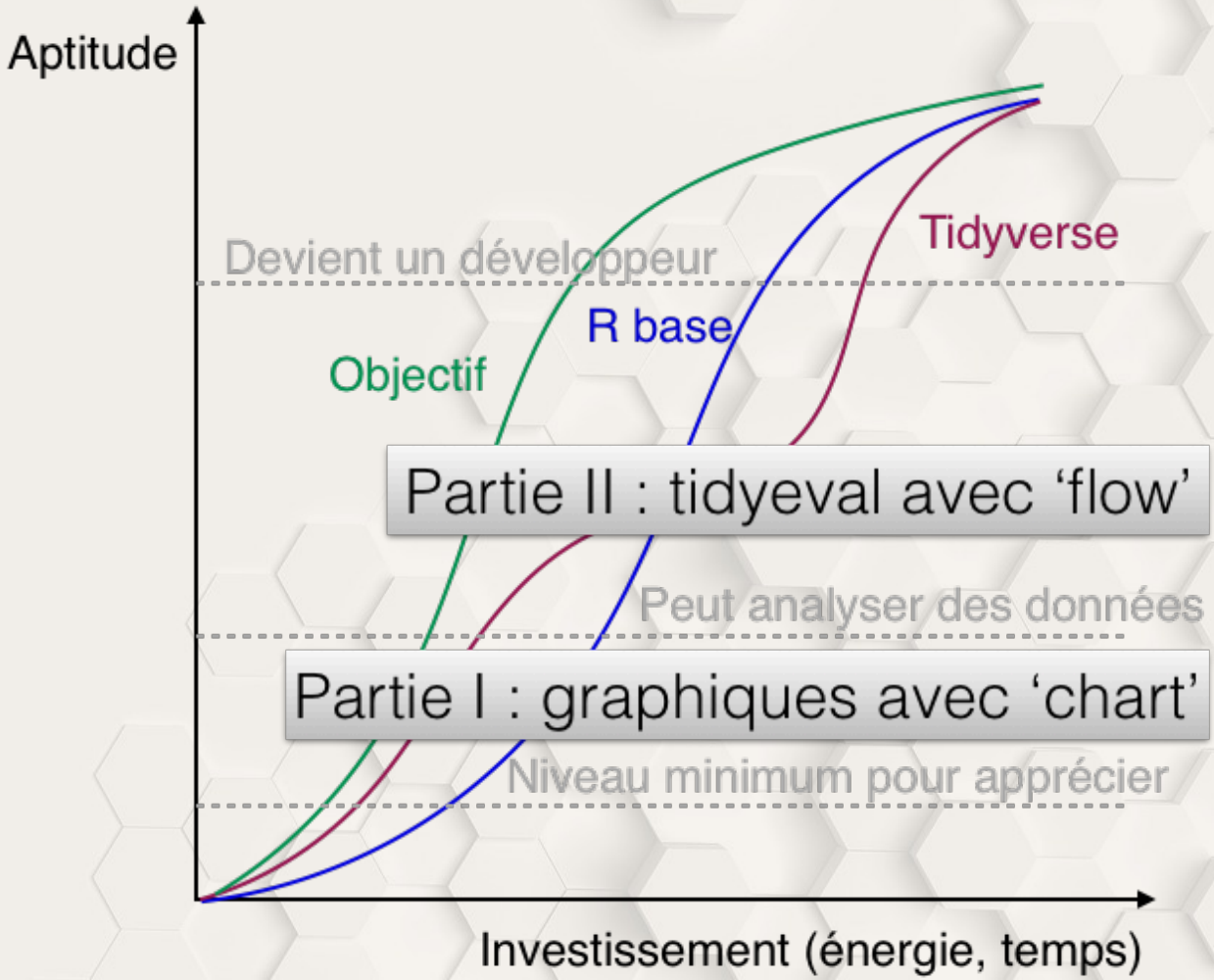
UMONS

Courbe d'apprentissage idéale



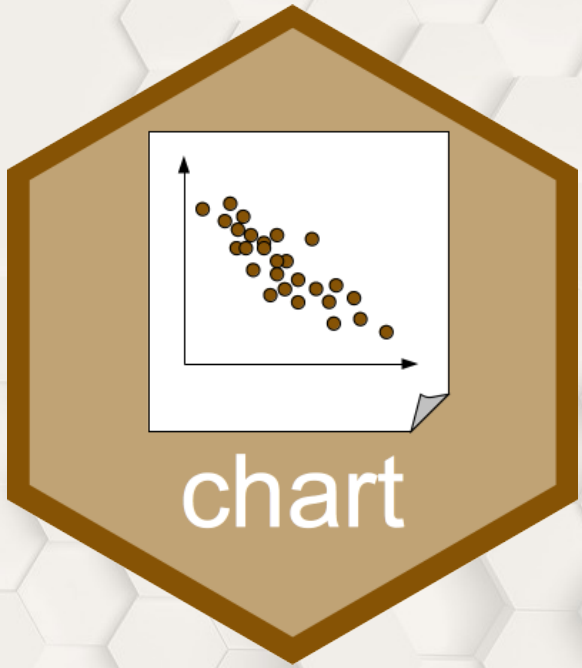
UMONS

Courbe d'apprentissage idéale



UMONS

Partie I : faciliter les graphiques R pour les débutants (package chart)



UMONS

L'écosystème R propose un vaste choix de fonctions

Trois philosophies différentes principales (cf. Amélia McNamara) : **dollar** & **formule** (R base) *versus* **Tidyverse**

R Syntax Comparison : : CHEAT SHEET

Dollar sign syntax

`goal(data$x, data$y)`

SUMMARY STATISTICS:

one continuous variable:
```
mean(mtcars$mpg)
```

one categorical variable:
```
table(mtcars$cyl)
```

two categorical variables:
```
table(mtcars$cyl, mtcars$am)
```

one continuous, one categorical:
```
mean(mtcars$mpg[mtcars$cyl==4])
mean(mtcars$mpg[mtcars$cyl==6])
mean(mtcars$mpg[mtcars$cyl==8])
```

PLOTTING:

one continuous variable:
```
hist(mtcars$disp)

boxplot(mtcars$disp)
```

one categorical variable:
```
barplot(table(mtcars$cyl))
```

two continuous variables:
```
plot(mtcars$disp, mtcars$mpg)
```

two categorical variables:
```
mosaicplot(table(mtcars$am, mtcars$cyl))
```

one continuous, one categorical:
```
histogram(mtcars$disp[mtcars$cyl==4])
histogram(mtcars$disp[mtcars$cyl==6])
histogram(mtcars$disp[mtcars$cyl==8])

boxplot(mtcars$disp[mtcars$cyl==4])
boxplot(mtcars$disp[mtcars$cyl==6])
boxplot(mtcars$disp[mtcars$cyl==8])
```

WRANGLING:

subsetting:
```
mtcars[mtcars$mpg>30, ]
```

making a new variable:
```
mtcars$efficient[mtcars$mpg>30] <- TRUE
mtcars$efficient[mtcars$mpg<30] <- FALSE
```

Formula syntax

`goal(y~x|z, data=data, group=w)`

SUMMARY STATISTICS:

one continuous variable:
```
mosaic::mean(~mpg, data=mtcars)
```

one categorical variable:
```
mosaic::tally(~cyl, data=mtcars)
```

two categorical variables:
```
mosaic::tally(cyl~am, data=mtcars)
```

one continuous, one categorical:
```
mosaic::mean(mpg~cyl, data=mtcars)
```

tilde

PLOTTING:

one continuous variable:
```
lattice::histogram(~disp, data=mtcars)

lattice::bwplot(~disp, data=mtcars)
```

one categorical variable:
```
mosaic::bargraph(~cyl, data=mtcars)
```

two continuous variables:
```
lattice::xyplot(mpg~disp, data=mtcars)
```

two categorical variables:
```
mosaic::bargraph(~am, data=mtcars, group=cyl)
```

one continuous, one categorical:
```
lattice::histogram(~disp|cyl, data=mtcars)

lattice::bwplot(cyl~disp, data=mtcars)
```

The variety of R syntaxes give you many ways to "say" the same thing

read **across** the cheatsheet to see how different syntaxes approach the same problem

Tidyverse syntax

`data %>% goal(x)`

SUMMARY STATISTICS:

one continuous variable:
```
mtcars %>% dplyr::summarize(mean(mpg))
```

one categorical variable:
```
mtcars %>% dplyr::group_by(cyl) %>%
  dplyr::summarize(n())
```

the pipe

two categorical variables:
```
mtcars %>% dplyr::group_by(cyl, am) %>%
  dplyr::summarize(n())
```

one continuous, one categorical:
```
mtcars %>%  dplyr::group_by(cyl) %>%
   dplyr::summarize(mean(mpg))
```

PLOTTING:

one continuous variable:
```
ggplot2::qplot(x=mpg, data=mtcars, geom = "histogram")

ggplot2::qplot(y=disp, x=1, data=mtcars, geom="boxplot")
```

one categorical variable:
```
ggplot2::qplot(x=cyl, data=mtcars, geom="bar")
```

two continuous variables:
```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

two categorical variables:
```
ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") +
       facet_grid(.~am)
```

one continuous, one categorical:
```
ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") +
       facet_grid(.~cyl)

ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars,
               geom="boxplot")
```

WRANGLING:

subsetting:
```
mtcars %>% dplyr::filter(mpg>30)
```

making a new variable:
```
mtcars <- mtcars %>%
dplyr::mutate(efficient = if_else(mpg>30,  TRUE, FALSE))
```

SMITH COLLEGE

UMONS

Graphiques sous R

- Trois moteurs graphiques principaux **base R (“dollar” et formule)**, **lattice (formule)** & **ggplot2 (tidyverse)**
- Rendu différent, syntaxe différente, incompatibilités
- **ggplot2** comme première approche (cf. David Robinson)

David Robinson

Chief Data Scientist at DataCamp,

Don't teach built-in plotting to beginners (teach ggplot2)

I have some experience teaching R programming (see, for instance, my Introduction to the Tidyverse course). One of the atypical choices I make is to start by teaching Hadley Wickham's ggplot2 package, rather than the built-in R plotting.

Many times that I mention this choice to

Graphiques sous R - difficultés évitables pour les débutants

- Aucun des 3 moteurs graphiques principaux de R n'est simple (reflet de leurs nombreuses possibilités), mais...

1. Est-il possible de limiter les différences visuelles (thèmes homogènes) ?
2. Est-il possible de rendre leurs interfaces respectives un peu plus cohérentes ?
3. Est-il possible de les assembler en figures composites ?

Voyons ensemble avec un exemple très simple d'une analyse par régression linéaire et le package `chart` quelques pistes d'amélioration des 3 points précédents.

UMONS

Graphiques sous R - difficultés évitables pour les débutants

- Aucun des 3 moteurs graphiques principaux de R n'est simple (reflet de leurs nombreuses possibilités), mais...

1. Est-il possible de limiter les différences visuelles (thèmes homogènes) ?
2. Est-il possible de rendre leurs interfaces respectives un peu plus cohérentes ?
3. Est-il possible de les assembler en figures composites ?

Voyons ensemble avec un exemple très simple d'une analyse par régression linéaire et le package `chart` quelques pistes d'amélioration des 3 points précédents.

UMONS

Graphiques sous R - difficultés évitables pour les débutants

- Aucun des 3 moteurs graphiques principaux de R n'est simple (reflet de leurs nombreuses possibilités), mais...

1. Est-il possible de limiter les différences visuelles (thèmes homogènes) ?
2. Est-il possible de rendre leurs interfaces respectives un peu plus cohérentes ?
3. Est-il possible de les assembler en figures composites ?

Voyons ensemble avec un exemple très simple d'une analyse par régression linéaire et le package `chart` quelques pistes d'amélioration des 3 points précédents.

UMONS

Graphiques sous R - difficultés évitables pour les débutants

- Aucun des 3 moteurs graphiques principaux de R n'est simple (reflet de leurs nombreuses possibilités), mais...

1. Est-il possible de limiter les différences visuelles (thèmes homogènes) ?
2. Est-il possible de rendre leurs interfaces respectives un peu plus cohérentes ?
3. Est-il possible de les assembler en figures composites ?

Voyons ensemble avec un exemple très simple d'une analyse par régression linéaire et le package `chart` quelques pistes d'amélioration des 3 points précédents.

UMONS

Graphiques sous R - difficultés évitables pour les débutants

- Aucun des 3 moteurs graphiques principaux de R n'est simple (reflet de leurs nombreuses possibilités), mais...

1. Est-il possible de limiter les différences visuelles (thèmes homogènes) ?
2. Est-il possible de rendre leurs interfaces respectives un peu plus cohérentes ?
3. Est-il possible de les assembler en figures composites ?

Voyons ensemble avec un exemple très simple d'une analyse par régression linéaire et le package `chart` *quelques pistes d'amélioration des 3 points précédents.*

UMONS

Analyse de la masse de squelette d'oursins

Jeu de données `urchin_bio` dans le package `data` qui propose aussi une fonction `read()` pour charger et **"enrichir"** les jeux de données (label et unité des variables dans différentes langues) :

```
#install.packages("devtools")
#devtools::install_github("SciViews/data")
urchin <- data::read("urchin_bio", package = "data", lang = "FR")
```

Nous aurons besoin aussi de `tidyverse`, `chart` et de leurs dépendances :

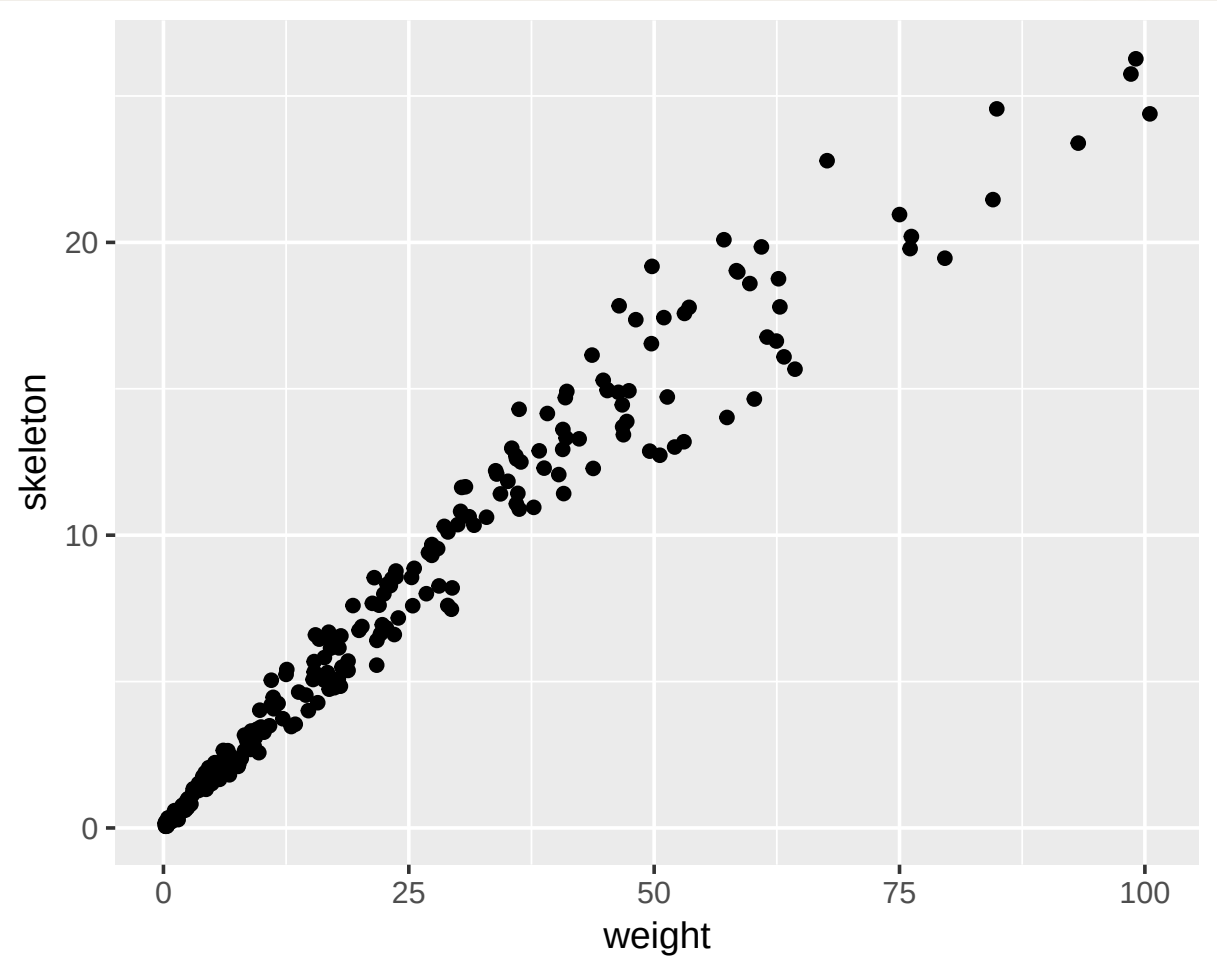
```
#install.packages(c("tidyverse", "latticeExtra", "cowplot",
#  "pryr", "ggpubr", "ggplotify"))
#devtools::install_github("SciViews/chart")
library(tidyverse)
library(chart)
```

Mettons-nous maintenant dans la peau d'un débutant qui découvre les outils nécessaires pour analyser ces données…

UMONS

Premier graphique avec ggplot2

```
ggplot(data = urchin, aes(weight, skeleton)) +
  geom_point()
```



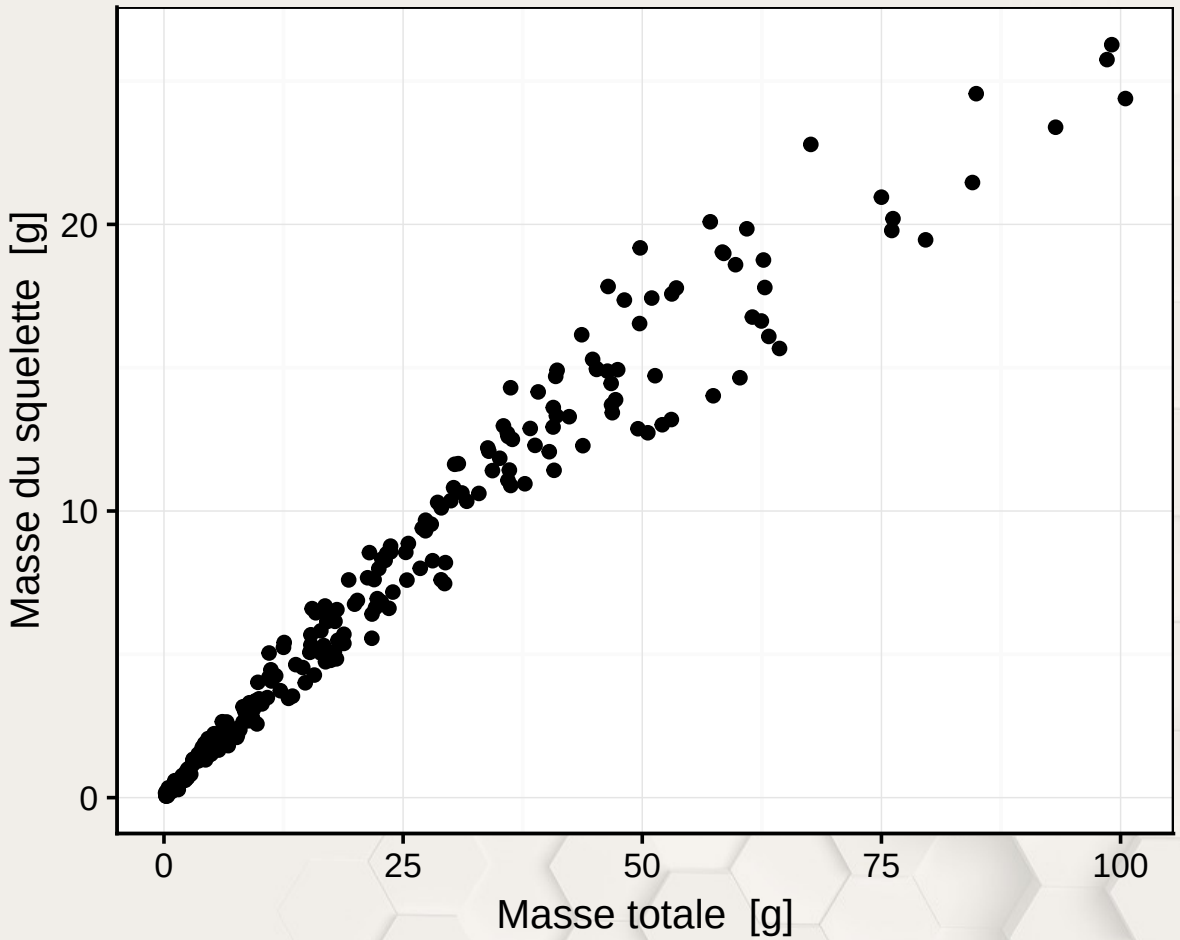
Code facile à comprendre et résultat très plaisant, mais...

- Libellés des axes par défaut sub-optimaux (unités manquantes)
- Thème gris particulier (distingue `ggplot2` des 2 autres)

UMONS

Premier graphique, version `chart`

```
chart(data = urchin, aes(weight, skeleton)) +
  geom_point()
```



Règle `chart` #1 : `chart()` peut simplement remplacer `ggplot()`.

- Substitution facile à retenir
- Libellés des axes et unités automatiques (si renseignés dans le jeu de données)
- Thème plus proche du "publication-ready"

UMONS

Suite logique de l'analyse : régression linéaire

```
(lmod <- lm(data = urchin, skeleton ~ weight))
```

```
Call:
lm(formula = skeleton ~ weight, data = urchin)

Coefficients:
(Intercept)       weight
     0.6882       0.2828
```

Pattern non compatible avec celui du graphique.

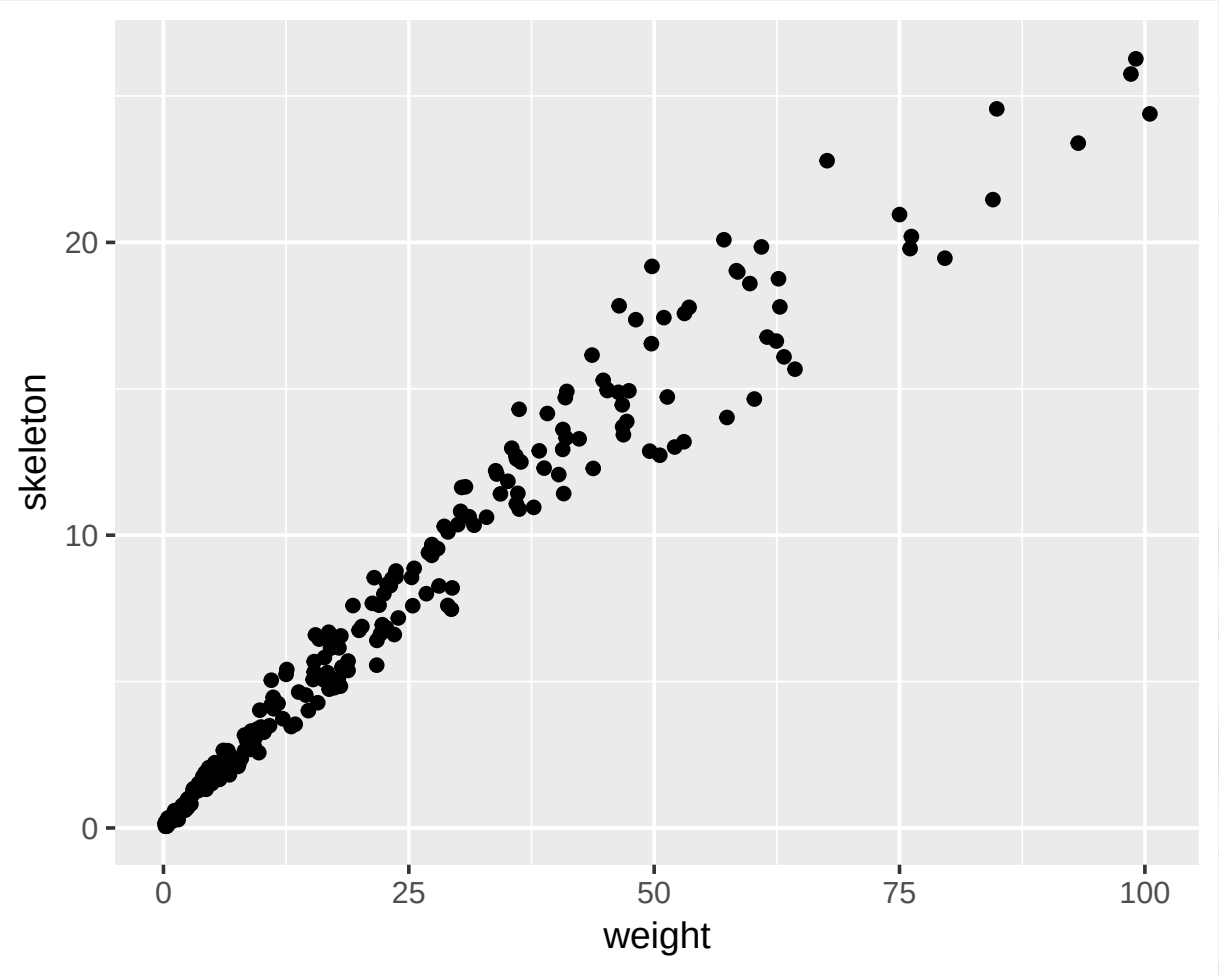
- `aes(<x>, <y>)` *versus* `<y> ~ <x>`
- Approche Tidyverse *versus* formula
- Inversion de la position des variables

Comment simplifier vers un pattern unique?

UMONS

Utilisation de formules avec ggplot2

```
ggplot(data = urchin, f_aes(skeleton ~ weight)) +
  geom_point()
```



Règle chart #2: formule `f_aes()` se substitue à `aes()`.

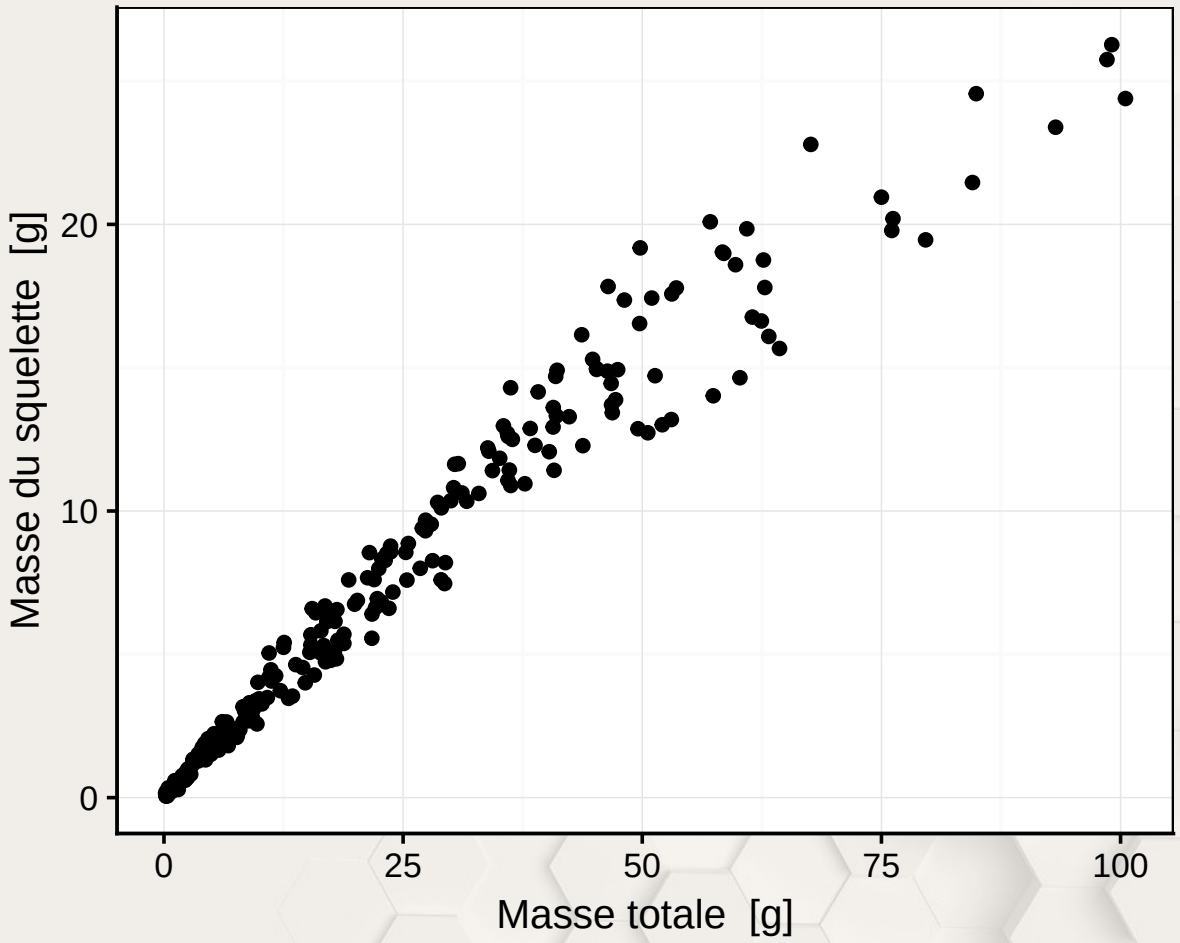
- Convergence vers un pattern identique graphe/modèle dans les cas simples :

```
<fun>(data = <df>,
<formula>)
```

UMONS

Utilisation conjointe de formules et de `chart()`

```
chart(data = urchin, skeleton ~ weight) +
  geom_point()
```



`f_aes()` est implicite!

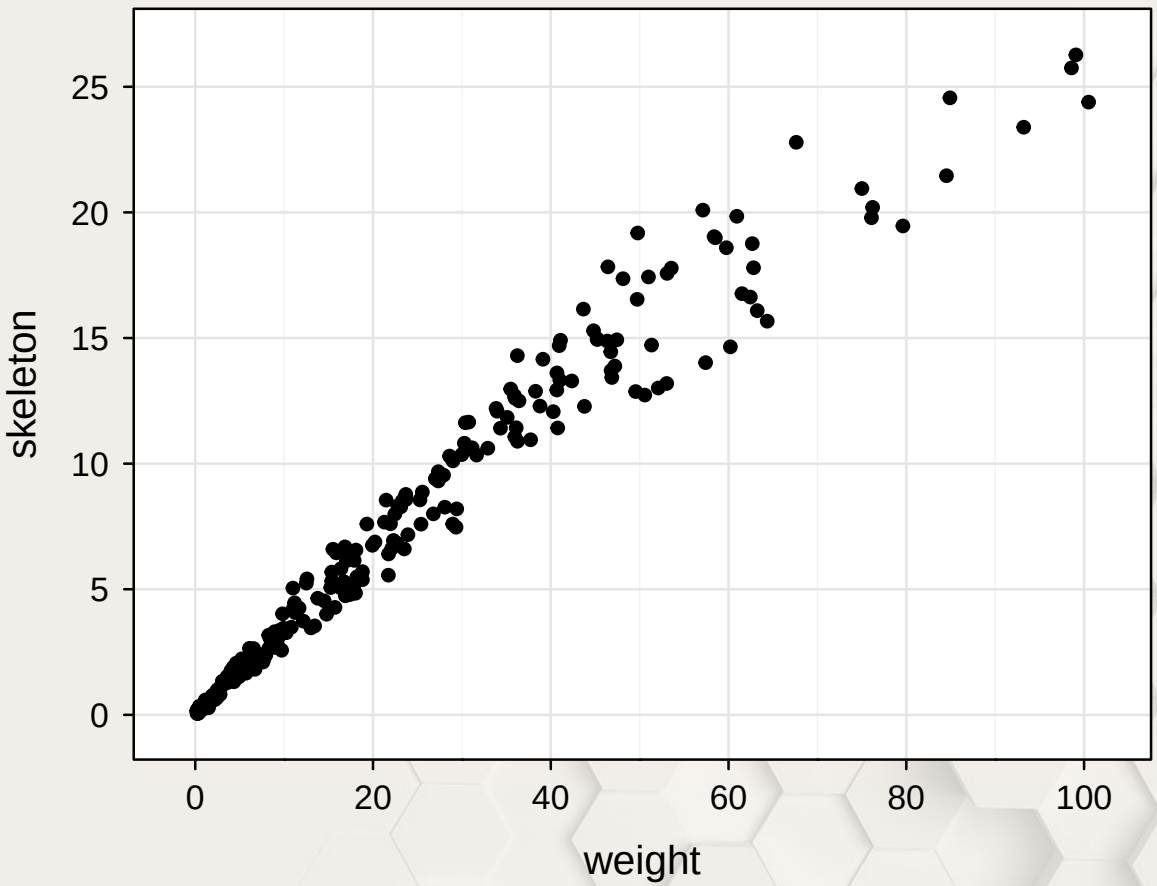
- Pattern réellement identique entre `lm()` et `chart()` plus facile pour le débutant :

```
<fun>(data = <df>,
<formula>)
```

UMONS

chart$<fun>() compatible avec lattice et base plots

```
chart$xyplot(data = urchin, skeleton ~ weight)
```



Règle chart #3: chart$<fun>() permet de varier le type de graphique, y compris base ou lattice !

- Le pattern reste très semblable :

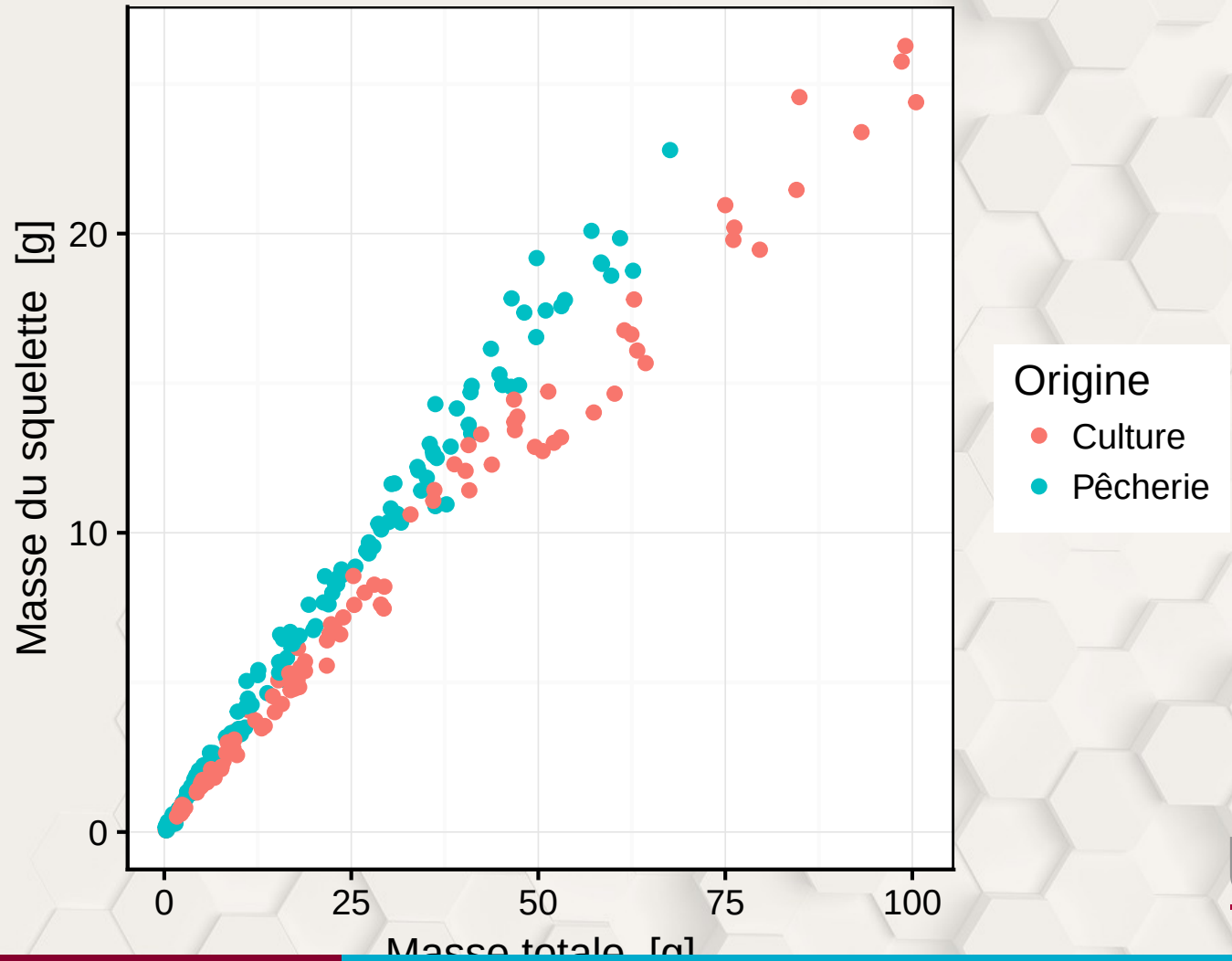
```
<fun>$<type>(data = <df>, <formula>)
```

- Thèmes `ggplot2` / `lattice` / `base` homogènes

UMONS

Arguments supplémentaires `aes` intégrables à la formule avec `%<par>=%`

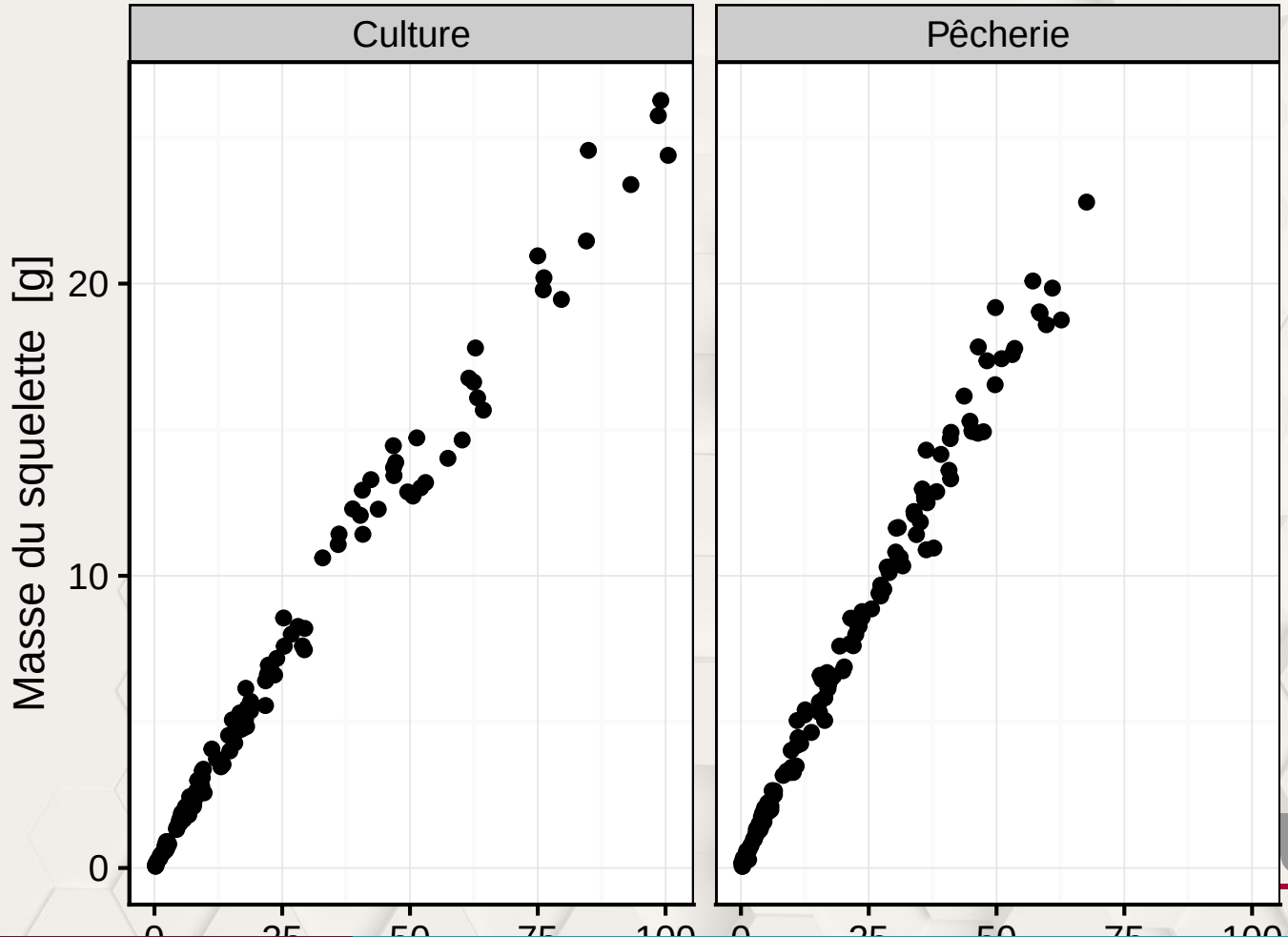
```
#chart$geom_point(data = urchin, skeleton ~ weight, col = origin)
chart$geom_point(data = urchin, skeleton ~ weight %col=% origin)
```



UMONS

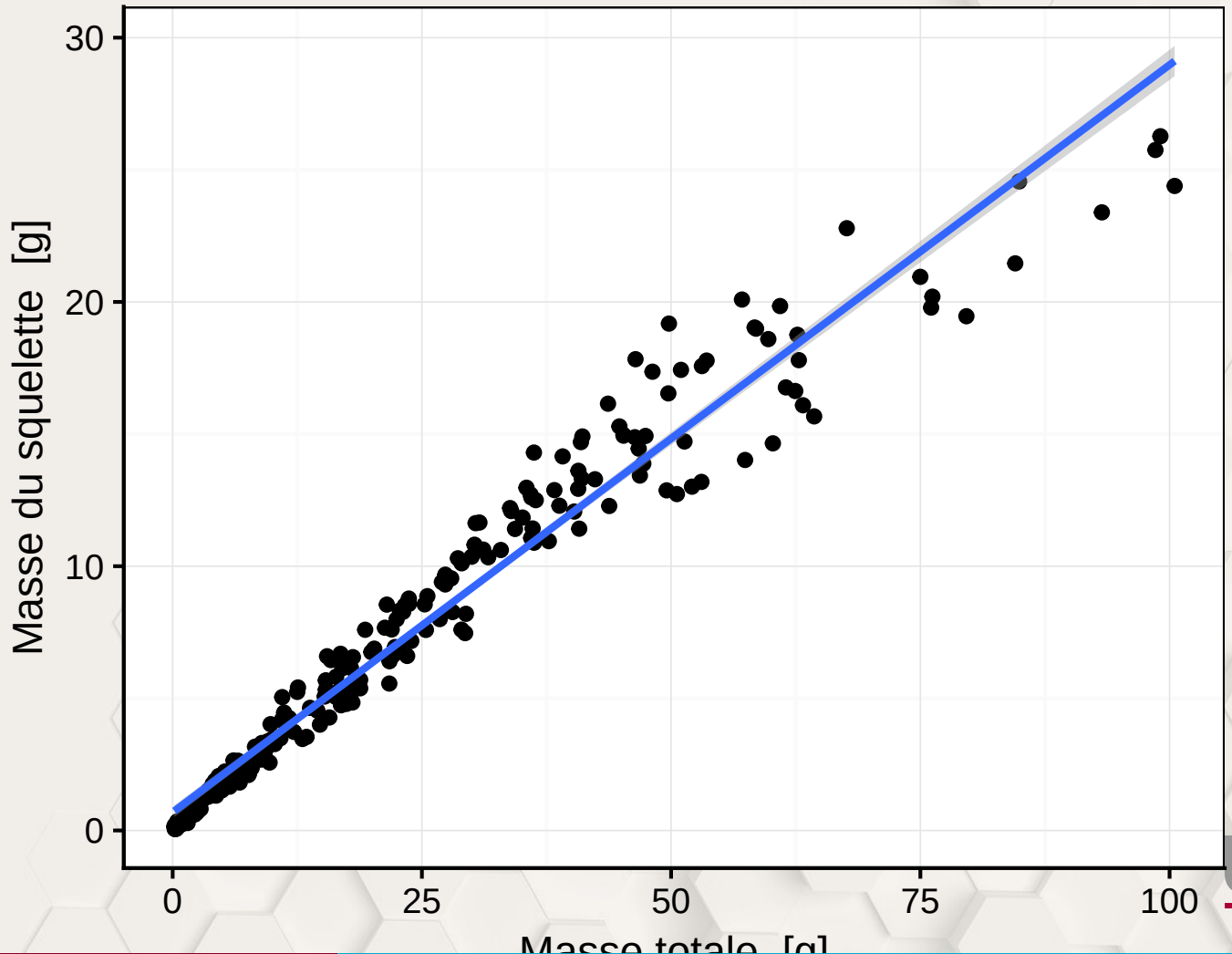
Facettes intégrables dans la formule avec | (= `lattice`)

```
#chart$geom_point(data = urchin, skeleton ~ weight) +
#  facet_wrap(~origin)
chart$geom_point(data = urchin, skeleton ~ weight | origin)
```



Ajout de la droite de régression

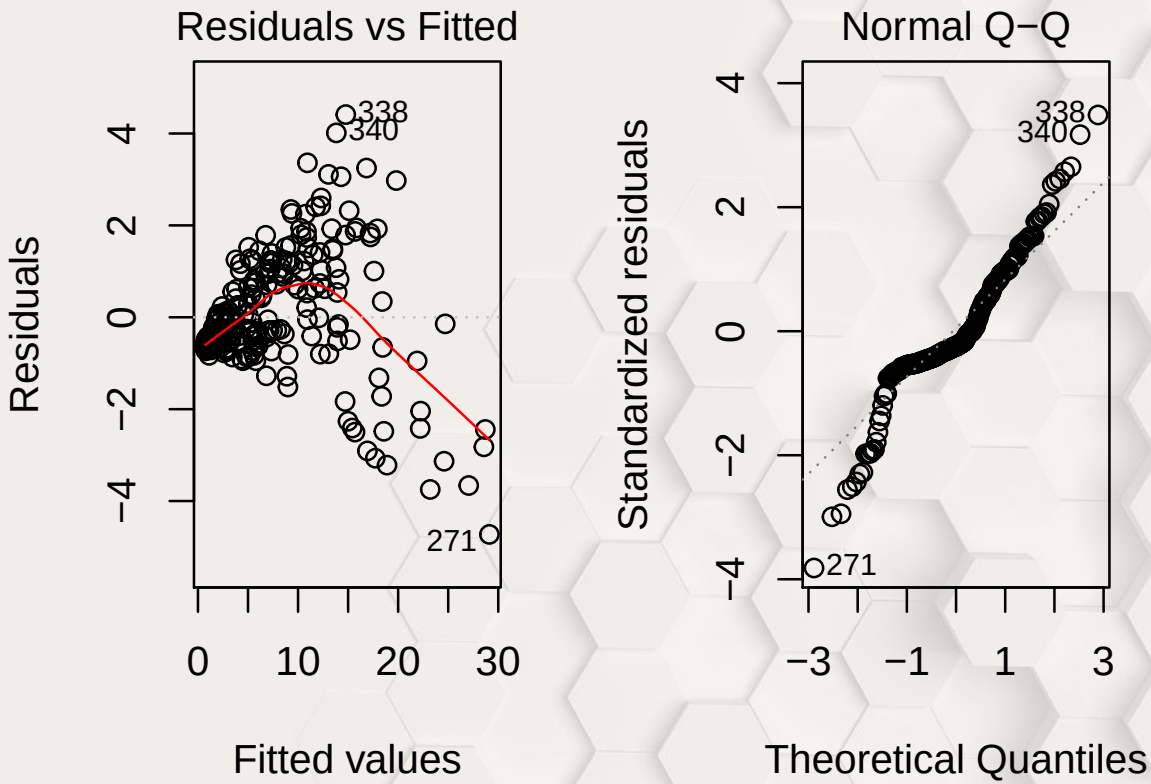
```
chart$geom_point(data = urchin, skeleton ~ weight) +
  geom_smooth(method = "lm")
```



UMONS

Suite de l'analyse : graphe des résidus

```
par(mfrow = c(1L, 2L))
plot(lmod, which = 1L)
plot(lmod, which = 2L)
```



Analyse des résidus de `lm()`
=> graphiques de base
Comment combiner avec le graphe `ggplot2` *précédent dans une figure composite ?*

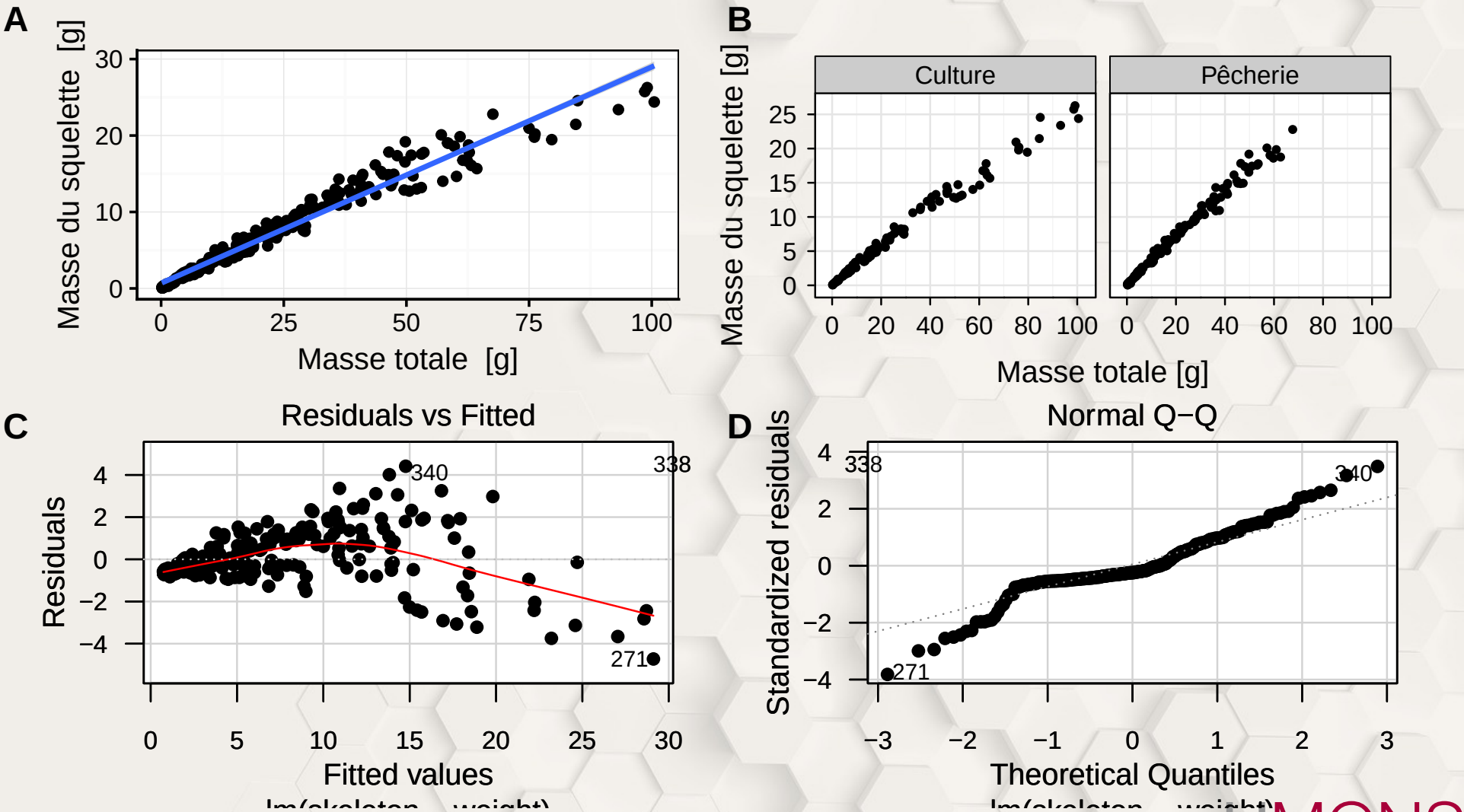
UMONS

Compatibilité des graphiques `chart` base/lattice/ggplot2 entre eux

```
# ggplot2
c1 <- chart$geom_point(data = urchin, skeleton ~ weight) +
  geom_smooth(method = "lm")
# Lattice plot
c2 <- chart$xyplot(data = urchin, skeleton ~ weight | origin)
# Base plots
c3 <- chart$plot(lmod, which = 1L)
c4 <- chart$plot(lmod, which = 2L)
```

UMONS

```
ggarrange(c1, c2, c3, c4, labels = "AUTO")
```



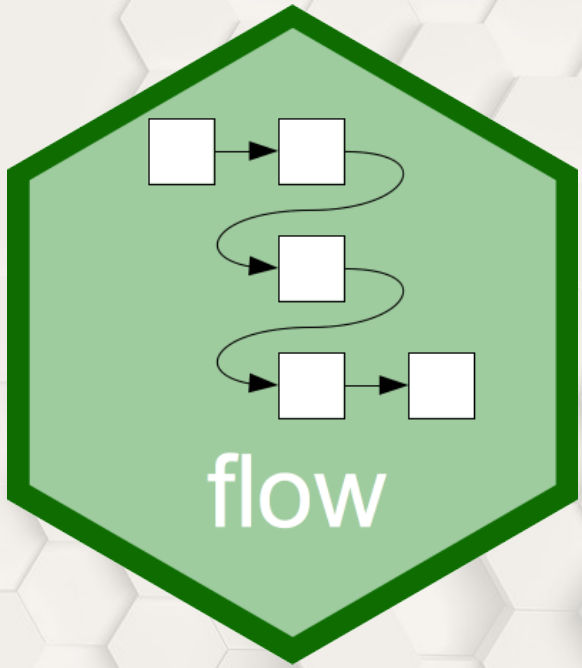
A retenir...

- Les graphiques `chart` peuvent tous être assemblés en une figure composite, qu'ils soient `ggplot2`, graphes de base ou `lattice` *(fini)*
- Les formules "étendues" sont aussi utilisables avec `ggplot2` *(fini)*
- Les thèmes des 3 moteurs graphiques sont homogénéisés le plus possible avec `chart` *(encore perfectible)*
- Si des attributs `label` et `units` existent, ils sont utilisés pour de meilleurs labels des axes *(reste à implémenter pour `lattice` et graphes de base)*

Travail en cours... d'autres idées d'améliorations sont les bienvenues, pull request sur https://github.com/SciViews/chart, s'il-vous-plait !

UMONS

Partie II : faciliter la réutilisation de pipelines & tidy évaluation (package flow)



UMONS

Pipeline dans Tidyverse

Voici un exemple de pipeline simple avec `%>%` :

```
urchin %>%
  mutate(lgsk = log(skeleton)) %>%
  summarise(mean = mean(lgsk,
    na.rm = TRUE))
```

mean
1.308811

Comment transformer ce pipeline en fonction réutilisable ?

- Nécessité de maîtriser le mécanisme “tidyeval” de tidyverse => barrière technologique qu’il serait souhaitable de limiter
- Approche par le package `flow` :

```
#devtools::install_github("SciViews/flow")
library(flow)
```

Pipeline dans Tidyverse

Voici un exemple de pipeline simple avec `%>%` :

```
urchin %>%
  mutate(lgsk = log(skeleton)) %>%
  summarise(mean = mean(lgsk,
    na.rm = TRUE))
```

mean
1.308811

Comment transformer ce pipeline en fonction réutilisable ?

- Nécessité de maîtriser le mécanisme “tidyeval” de tidyverse => barrière technologique qu’il serait souhaitable de limiter
- Approche par le package `flow` :

```
#devtools::install_github(”SciViews/flow”)
library(flow)
```

Création explicite d'un objet `flow` : permet d'inclure d'autres variables

Tidyverse

```
#
na_rm <- TRUE
urchin %>%
  mutate(lgsk = log(skeleton)) %>%
  summarise(mean = mean(lgsk,
    na.rm = !!na_rm))
```

flow

```
flow(urchin,
  na_rm = TRUE
) %>_%
  mutate(., lgsk = log(skeleton)) %>_%
  summarise(., mean = mean(lgsk,
    na.rm = na_rm_)) %>_% .
```

La variable `na_rm` est incluse dans l'objet `flow`

- Elle ne **"pollue" pas l'environnement** où le pipeline est exécuté, contrairement à ce qui se passe à gauche dans la forme classique
- Opérateur préfixé `!!` (tidyeval) de **tidyverse** est remplacé par l'"opérateur" `_` suffixé dans `flow`

UMONS

Création explicite d'un objet `flow` : permet d'inclure d'autres variables

Tidyverse

```
#
na_rm <- TRUE
urchin %>%
  mutate(lgsk = log(skeleton)) %>%
  summarise(mean = mean(lgsk,
    na.rm = !!na_rm))
```

flow

```
flow(urchin,
  na_rm = TRUE
) %>_%
  mutate(., lgsk = log(skeleton)) %>_%
  summarise(., mean = mean(lgsk,
    na.rm = na_rm_)) %>_% .
```

La variable `na_rm` est incluse dans l'objet `flow`

- Elle ne **"pollue" pas l'environnement** où le pipeline est exécuté, contrairement à ce qui se passe à gauche dans la forme classique
- Opérateur préfixé `!!` (tidyeval) de **tidyverse** est remplacé par l'"opérateur" `_` suffixé dans `flow`

UMONS

Utilisation plus transparente des quosures dans flow

Tidyverse

```
#
x <- quo(skeleton)
na_rm <- TRUE
urchin %>%
  mutate(lgsk = log(!!x)) %>%
  summarise(mean = mean(lgsk,
    na.rm = !!na_rm))
```

flow

```
flow(urchin,
  x_ = skeleton,
  na_rm = TRUE
) %>_%
  mutate(., lgsk = log(x_)) %>_%
  summarise(., mean = mean(lgsk,
    na.rm = na_rm_)) %>_% .
```

Une variable `flow` suffixée de `_` crée **automatiquement** une `quosure`

- Le passage d'expressions via les `quosures` devient transparent avec l'opérateur suffixé `_` dans `flow`
- la spécification de l'expression est largement simplifiée (pas besoin de `quo()` ou `enquo()`, objectif premier de l'utilisation de l'évaluation non standard !)

UMONS

Utilisation plus transparente des `quosures` dans `flow`

Tidyverse

```
#
x <- quo(skeleton)
na_rm <- TRUE
urchin %>%
  mutate(lgsk = log(!!x)) %>%
  summarise(mean = mean(lgsk,
    na.rm = !!na_rm))
```

flow

```
flow(urchin,
  x_ = skeleton,
  na_rm = TRUE
) %>_%
  mutate(., lgsk = log(x_)) %>_%
  summarise(., mean = mean(lgsk,
    na.rm = na_rm_)) %>_% .
```

Une variable `flow` suffixée de _ crée **automatiquement** une `quosure`

- Le passage d'expressions via les `quosures` devient transparent avec l'opérateur suffixé _ dans `flow`
- la spécification de l'expression est largement simplifiée (pas besoin de `quo()` ou `enquo()`, objectif premier de l'utilisation de l'évaluation non standard !)

UMONS

Version finale avec trois variables

Tidyverse

```
#
x <- quo(skeleton)
y <- ”lgsk”
y_quo <- as.quosure(as.name(y))
na_rm <- TRUE
urchin %>%
  mutate(!!y := log(!!x)) %>%
  summarise(mean = mean(!!y_quo,
    na.rm = !!na_rm))
```

flow

```
flow(urchin,
  x_ = skeleton,
  y_ = lgsk,

  na_rm = TRUE
) %>_%
  mutate(., y_ = log(x_)) %>_%
  summarise(., mean = mean(y_,
    na.rm = na_rm_)) %>_% .
```

La définition d’un nom de variable et son utilisation ensuite dans un pipeline est beaucoup plus simple avec `flow`

- Une seule variable (`y_`) au lieu de deux (`y` (`character`) et `y_quo` (`quosure`))!
- Pas d’obligation de remplacer `=` par `:=` pour conserver une syntaxe R correcte

UMONS

Version finale avec trois variables

Tidyverse

```
#
x <- quo(skeleton)
y <- "lgsk"
y_quo <- as.quosure(as.name(y))
na_rm <- TRUE
urchin %>%
  mutate(!!y := log(!!x)) %>%
  summarise(mean = mean(!!y_quo,
    na.rm = !!na_rm))
```

flow

```
flow(urchin,
  x_ = skeleton,
  y_ = lgsk,

  na_rm = TRUE
) %>_%
  mutate(., y_ = log(x_)) %>_%
  summarise(., mean = mean(y_,
    na.rm = na_rm_)) %>_% .
```

La définition d'un nom de variable et son utilisation ensuite dans un pipeline est beaucoup plus simple avec `flow`

- Une seule variable (`y_`) au lieu de deux (`y` (`character`) et `y_quo` (`quosure`))!
- Pas d'obligation de remplacer `=` par `:=` pour conserver une syntaxe R correcte

UMONS

Fonction réutilisable depuis un pipeline (“séquence fonctionnelle”)

Travail en cours...

Tidyverse

```
#
x <- quo(skeleton)
y <- ”lgsk”
y_quo <- as.quosure(as.name(y))
na_rm <- TRUE
foo <- . %>%
  mutate(!!y := log(!!x)) %>%
  summarise(mean = mean(!!y_quo,
    na.rm = !!na_rm))
# Utilisation
foo(urchin)
```

flow

```
foo <- flow_function(urchin,
  x_ = skeleton,
  y_ = lgsk,

  na_rm = TRUE
) %>_%
  mutate(., y_ = log(x_)) %>_%
  summarise(., mean = mean(y_,
    na.rm = na_rm_)) %>_% .
# Utilisation, autre variable
foo(urchin, x_ = weight)
```

Seul `flow` permet d’inclure d’autres variables dans la séquence fonctionnelle `foo`

- `flow()` est juste remplacé par `flow_function()`
- Passage à une fonction véritable **beaucoup plus facile et intuitive** à partir de `flow_function()` (conversion automatisée même possible)!

Fonction réutilisable depuis un pipeline (“séquence fonctionnelle”)

Travail en cours...

Tidyverse

```
#
x <- quo(skeleton)
y <- ”lgsk”
y_quo <- as.quosure(as.name(y))
na_rm <- TRUE
foo <- . %>%
  mutate(!!y := log(!!x)) %>%
  summarise(mean = mean(!!y_quo,
    na.rm = !!na_rm))
# Utilisation
foo(urchin)
```

flow

```
foo <- flow_function(urchin,
  x_ = skeleton,
  y_ = lgsk,

  na_rm = TRUE
) %>_%
  mutate(., y_ = log(x_)) %>_%
  summarise(., mean = mean(y_,
    na.rm = na_rm_)) %>_% .
# Utilisation, autre variable
foo(urchin, x_ = weight)
```

Seul `flow` permet d’inclure d’autres variables dans la séquence fonctionnelle `foo`

- `flow()` est juste remplacé par `flow_function()`
- Passage à une fonction véritable **beaucoup plus facile et intuitive** à partir de `flow_function()` (conversion automatisée même possible)!

A retenir...

- Les objets `flow` contiennent tout ce qui est nécessaire au pipeline, y compris des variables satellites éventuelles
- Le mécanisme “tidyeval” de tidyverse est beaucoup plus facile à implémenter et quasi-totalement transparent avec la convention `<var>_` de `flow`
- Le passage d'un pipeline `flow()` avec variables satellites à une fonction réutilisable est immédiat en remplaçant par `flow_function()`
- La transition pipeline tidyverse à usage unique vers la fonction réutilisable est graduelle et bien plus facile avec `flow`

Un useR devient un developeR en douceuR !

En cours de finalisation et soumission à CRAN... les contributions sont les bienvenues, pull request sur https://github.com/SciViews/flow, s'il-vous-plait !

UMONS

Merci

Avez-vous des questions ?

Présentation et version plus détaillée sous forme de **R Notebook** disponible à https://github.com/SciViews/RencontresRRennes2018

UMONS