Une introduction à Tidyverse

Christophe Ambroise (d'après Hadley Wickham)

25/05/2018

Préambule

Avertissement

De nombreux extraits de cette séquence de transparents (surtout les codes) sont plagiés, inspirés ou traduits du livre ''R for data science: import, tidy, transform, visualize, and model data" (Wickham & Grolemund, 2016)[1]

[1] Les sources de ce livre sont disponibles sur github

Prérequis

Une petite expérience de la programmation, que l'on peut acquérir par exemple avec le livre Hands on Programming with R[2]

R, RStudio, une collection de packages appelés **tidyverse**, et quelques autres packages.

[2] Ce livre est disponible gratuitement sur github

Tidyverse

Pour installer l'ensemble des packages contenus dans **tidyverse**

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

Pour utiliser les packages, il suffit d'utiliser la fonction `library()`

```
library(magrittr)
library(tidyverse)
```

3

[3] tidyverse charge les paquets ggplot2, tibble, tidyr, readr, purrr et dplyr. Ceux-ci sont considérés comme le **core** du tidyverse

les packages au cœur de `Tidyverse`

- `ggplot2`, pour les graphiques
- `dplyr`, pour transformer et résumer le contenu des données
- `tidyr`, pour transformer la structure des données de type tableau
- `purrr`, une mise à niveau des outils de programmation fonctionnelle de R base
- `tibble`, un tableau de données amélioré

La philosophie du `tidyverse`

D'après Hadley Wickham (directeur scientifique de RStudio)

> *permettre à l'analyste de se concentrer sur les questions de fond plutôt que sur les aspects techniques de l'analyse des données.*

Quelques packages de données

```
install.packages(c("nycflights13", "gapminder", "Lahman"))
```

Le pipe "%>%

Le pipe

Objectif

Le but du pipe est de vous aider à écrire le code d'une manière plus facile à lire et à comprendre.

Principe

- `x %>% f()` est équivalent à `f(x)`
- `x %>% f(y)` est équivalent à `f(x, y)`
- `x %>% f(y,.)` est équivalent à `f(y,x)`

```
x<-3
log(x)
```

```
## [1] 1.098612
```

```
x %>% log()
```

```
## [1] 1.098612
```

(Ré-)Affectation

Pour l'affectation, magrittr fournit l'opérateur %<>% qui vous permet de remplacer le code comme:

```
mtcars <- mtcars%>% transform(cyl = cyl * 2)
```

par

```
mtcars %<>% transform(cyl = cyl * 2)
```

T-pipe %T>%

Fonction à effet de bords et fin prématurée de pipe

- Peut-être que vous voulez imprimer un objet, ou le tracez
- ces fonctions ne renvoient rien et terminent effectivement le pipe

Solution

- Pour contourner ce problème, vous pouvez utiliser le “tee” pipe. % T>%
- fonctionne comme %>% sauf qu’il retourne le côté gauche au lieu
- “tee” parce que c’est comme un tuyau en forme de T

T-pipe %T>% I

Sans T

```
rnorm (100) %>%
  matrix(ncol = 2) %>%
  plot() %>%
  str()
```

T-pipe %T>% II

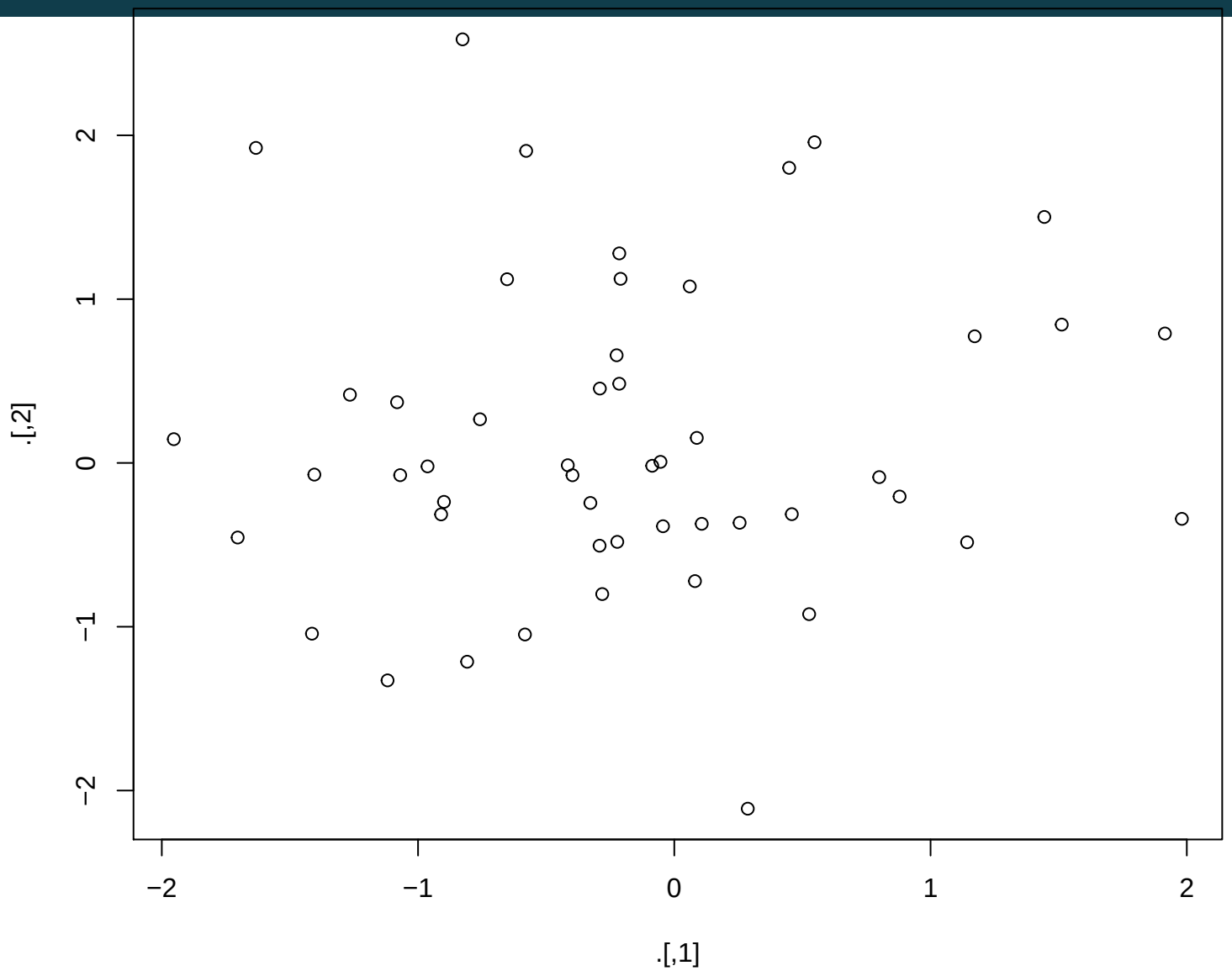


Figure 1: Graphique bivarié d'un échantillon gaussien

T-pipe %T>% I

Avec T

```
rnorm (100) %>%
  matrix(ncol = 2) %T>%
  plot() %>%
  str()
```

T-pipe %T>% II

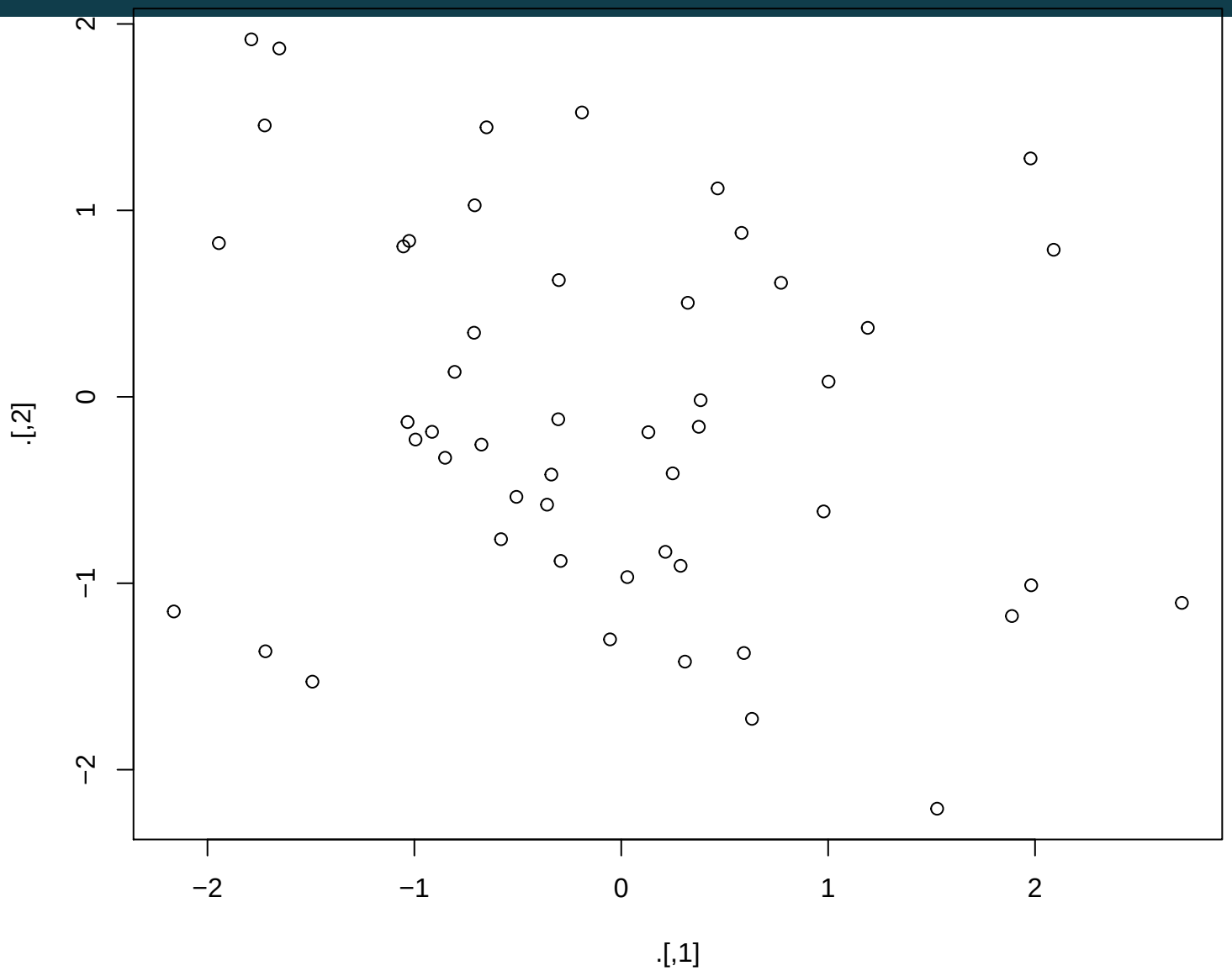


Figure 2: Graphique bivarié d'un échantillon gaussien

Les structures de données `Tibble`

La `Tibble` philosophie I

`tibble` versus `data.frame`

Les `Tibbles` sont des tableau de données, mais ils modifient certains comportements plus anciens pour rendre la vie un peu plus facile.
Il est difficile de changer la base R sans casser le code existant, donc la plupart des innovations se produisent dans les paquets.

Conversion d'un `data.frame`

```
as_tibble(iris)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

Création d'un `tibble`

```
tibble(
  x = 1:5,
  y = 1,
  z = x ^ 2 + y
)
```

```
## # A tibble: 5 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     1     2
## 2     2     1     5
## 3     3     1    10
## 4     4     1    17
## 5     5     1    26
```

Noms de colonnes d'un `tibble` I

Les noms peuvent ne pas commencer par une lettre ou contenir des caractères inhabituels comme un espace. Pour faire référence à ces variables, vous devez les entourer de guillemets

```
tb <- tibble(
  `:)` = "smile",
  ` ` = "space",
  `2000` = "number"
)
tb
```

```
## # A tibble: 1 x 3
##   `:)`  ` `   `2000`
##   <chr> <chr> <chr>
## 1 smile space number
```

Vous aurez également besoin des backticks lorsque vous travaillez avec ces variables dans d'autres packages, tels que ggplot2, dplyr et tidyr.

Noms des lignes

Différence par rapport au `data.frame`

Les lignes n'ont pas de noms

Solution

- Pour utiliser un nom il suffit de créer une colonne supplémentaire
- `rownames_to_column ()` peut vous aider

Noms des lignes, exemple

```
swiss %>% as_tibble(rownames = "Province")
```

```
## # A tibble: 47 x 7
##    Province     Fertility Agriculture Examination Education Catholic
##    <chr>            <dbl>       <dbl>       <int>     <int>    <dbl>
##  1 Courtelary        80.2        17             15        12     9.96
##  2 Delemont          83.1        45.1            6         9    84.8
##  3 Franches-Mnt      92.5        39.7            5         5    93.4
##  4 Moutier           85.8        36.5           12         7    33.8
##  5 Neuveville        76.9        43.5           17        15     5.16
##  6 Porrentruy        76.1        35.3            9         7    90.6
##  7 Broye             83.8        70.2           16         7    92.8
##  8 Glane             92.4        67.8           14         8    97.2
##  9 Gruyere           82.4        53.3           12         7    97.7
## 10 Sarine            82.9        45.2           16        13    91.4
## # ... with 37 more rows, and 1 more variable: Infant.Mortality <dbl>
```

Un type `list` pour une colonne de `tibble`

Mettre tout (et pas n'importe quoi) dans une case

- les `tibble` permettent les cellules contenant des listes
- les `tibble` permettent les cellules contenant des data.frame

Création (voir les primitives de `tidyR`)

- `tidyr::nest()` pour convertir un `data.frame` groupé en `data.frame` imbriqué
- `mutate()` en utilisant des fonctions vectorisées qui renvoient une liste
- `summarize()` qui renvoient plusieurs résultats

Exemple: Un type `list` pour une colonne de `tibble`

```
iris %>%
  group_by(Species) %>%
  nest(.key = Data) %>%
  mutate(Model = purrr::map(Data,
                            ~ lm(data = .,
                                 Sepal.Length ~ Petal.Length))) %>%
  mutate(Summary = purrr::map(Model, summary)) %>%
  mutate(`R squared` = purrr::map_dbl(Summary, ~ .$r.squared))
```

```
## # A tibble: 3 x 5
##   Species    Data              Model    Summary          `R squared`
##   <fct>      <list>            <list>   <list>                 <dbl>
## 1 setosa     <tibble [50 x 4]> <S3: lm> <S3: summary.lm>       0.0714
## 2 versicolor <tibble [50 x 4]> <S3: lm> <S3: summary.lm>       0.569
## 3 virginica  <tibble [50 x 4]> <S3: lm> <S3: summary.lm>       0.747
```

Les manupilations faciles avec `dplyr`

En bref

`dplyr` est une grammaire de la manipulation des données, fournissant un ensemble cohérent de verbes qui vous aident à résoudre les défis de manipulation de données les plus courants:

- `select()` sélectionne les variables en fonction de leur nom.
- `filter ()` sélectionne les observations en fonction de leurs valeurs.
- `arrange()` modifie l'ordre des lignes.
- `mutate()` ajoute de nouvelles variables qui sont des fonctions de variables existantes
- `summarize()` réduit plusieurs valeurs à un seul résumé.

Sélection des lignes avec `filter()` I

```
data("mtcars")
mtcars %>% head(10)
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

Sélection des lignes avec `filter()` II

```
mtcars %>% filter(cyl == 4, mpg>30) %>%
           head(10)
```

```
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
## 2 30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
## 3 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
## 4 30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

`filter ()` entrées

1. données
2. expression de filtrage

`filter ()` sortie

- génère un `tibble`
- **Sans modifier** les données d'origines

Réordonner les lignes avec `arrange()`

```
mtcars %>%
  arrange(desc(carb), mpg) %>%
  head(5)
```

```
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
## 2 19.7   6  145 175 3.62 2.770 15.50  0  1    5    6
## 3 10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## 4 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## 5 13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
```

Principe de `arrange()`

Fonctionne comme `filter()` mais réordonne les lignes suivant une série de conditions

Sélection de colonnes avec `select()` I

Sélection par noms de colonnes

```
mtcars %>%
  select(mpg, wt, cyl) %>%
  head(10)
```

```
##                    mpg    wt cyl
## Mazda RX4         21.0 2.620   6
## Mazda RX4 Wag     21.0 2.875   6
## Datsun 710        22.8 2.320   4
## Hornet 4 Drive    21.4 3.215   6
## Hornet Sportabout 18.7 3.440   8
## Valiant           18.1 3.460   6
## Duster 360        14.3 3.570   8
## Merc 240D         24.4 3.190   4
## Merc 230          22.8 3.150   4
## Merc 280          19.2 3.440   6
```

Sélection de colonnes avec `select()` II

Sélection par indices de colonnes

```
mtcars %>%
  select(1,2,5:7) %>%
  head(10)
```

```
##                    mpg cyl drat    wt  qsec
## Mazda RX4         21.0   6 3.90 2.620 16.46
## Mazda RX4 Wag     21.0   6 3.90 2.875 17.02
## Datsun 710        22.8   4 3.85 2.320 18.61
## Hornet 4 Drive    21.4   6 3.08 3.215 19.44
## Hornet Sportabout 18.7   8 3.15 3.440 17.02
## Valiant           18.1   6 2.76 3.460 20.22
## Duster 360        14.3   8 3.21 3.570 15.84
## Merc 240D         24.4   4 3.69 3.190 20.00
## Merc 230          22.8   4 3.92 3.150 22.90
## Merc 280          19.2   6 3.92 3.440 18.30
```

mutate() nouvelles variables fonctions des variables existantes I

```
mtcars %<>%
  mutate(cyl2 = 2 * cyl,
         cyl4 = 2 * cyl2,
         disp = disp * 0.0163871,
         drat = NULL)
knitr::kable(mtcars[1:6, 1:6], caption = 'Sous tableau de mtcars avec de nouv
```

Table 1: Sous tableau de mtcars avec de nouvelles variable

mpg	cyl	disp	hp	wt	qsec
21.0	6	2.621936	110	2.620	16.46
21.0	6	2.621936	110	2.875	17.02
22.8	4	1.769807	93	2.320	18.61
21.4	6	4.227872	110	3.215	19.44
18.7	8	5.899356	175	3.440	17.02

`mutate()` nouvelles variables fonctions des variables existantes II

18.1	6	3.687098	105	3.460	20.22

`mutate()` nouvelles variables fonctions des variables existantes III

Remarque

Tout comme les autres primitives de `dyplr` `mutate()` ne modifie pas le tableau d'origine

Combiner `summarise()` et `group_by()` pour calculer des résumés par groupe de lignes I

```
mtcars %>%
  summarise(Mean_mpg = mean(mpg),
            Var_disp = var(disp)) %>%
  knitr::kable()
```

Mean_mpg	Var_disp
20.09062	4.124944

Combiner `summarise()` et `group_by()` pour calculer des résumés par groupe de lignes II

`summarise()`

Calcul des résumé numériques sur les lignes

```
mtcars %>%
  group_by(cyl,am) %>%
  summarise(Mean_mpg = mean(mpg), Var_disp = var(disp)) %>% knitr::kable(
```

cyl	am	Mean_mpg	Var_disp
4	0	22.90000	0.0524032
4	1	28.07500	0.1125972
6	0	19.12500	0.5375852
6	1	20.56667	0.0201403
8	0	15.05000	1.3852790
8	1	15.40000	0.3356713

L'objet tableau de données groupées `grouped_df`

`group_by()` ne fait pas grand chose en dehors de la création d'un objet "grouped_df". La magie se produit vraiment quand vous lancez un `summarise()` sur cet objet.
[a]

[a] Tous les groupes possibles sont formés à partir des combinaisons de variables `cyl`, `am`

Les données propres avec `tidyR`

Les primitives

Un jeu de données élève centré

```
grades <- tibble(
  Name = c("Tommy", "Mary", "Gary", "Cathy"),
  Sexage = c("m.15", "f.15", "m.16", "f.14"),
  Math = c(10, 15, 16, 14),
  Philo = c(11, 13, 10, 12),
  English = c(12, 13, 17, 10)
  )
knitr::kable(grades,caption = "Notes par élève")
```

Table 4: Notes par élève

Name	Sexage	Math	Philo	English
Tommy	m.15	10	11	12
Mary	f.15	15	13	13
Gary	m.16	16	10	17
Cathy	f.14	14	12	10

separate()

```
grades <-
  grades %>%
  separate(Sexage, into = c("Sex", "Age"))
knitr::kable(grades,caption = "Notes par élève avec variables sexe et age")
```

Table 5: Notes par élève avec variables sexe et age

Name	Sex	Age	Math	Philo	English
Tommy	m	15	10	11	12
Mary	f	15	15	13	13
Gary	m	16	16	10	17
Cathy	f	14	14	12	10

Remarque

L'instruction inverse de `separate()` est `unite()`

gather()

```
grades <-
  grades %>%
  gather(Math,Philo,English, key = Topic, value = Grade)
knitr::kable(grades,caption="Tableau `tidy")
```

Table 6: Tableau ‘tidy

Name	Sex	Age	Topic	Grade
Tommy	m	15	Math	10
Mary	f	15	Math	15
Gary	m	16	Math	16
Cathy	f	14	Math	14
Tommy	m	15	Philo	11
Mary	f	15	Philo	13
Gary	m	16	Philo	10
Cathy	f	14	Philo	12
Tommy	m	15	English	12
Mary	f	15	English	13
Gary	m	16	English	17
Cathy	f	14	English	10

Avantage d'un tableau `tidy`

Dans un tableau `tidy` les observations sont basiques, chaque colonne est de nature différente et permet de nombreux regroupements d'observations

Moyenne par élève

```
grades %>%
  group_by(Name) %>%
  summarise(Mean = mean(Grade)) %>%
  knitr::kable(caption="Moyenne par élève")
```

Table 7: Moyenne par élève

Name	Mean
Cathy	12.00000
Gary	14.33333
Mary	13.66667
Tommy	11.00000

Moyenne par matière

```
grades %>%
  group_by(Topic) %>%
  summarise(Mean = mean(Grade)) %>%
  knitr::kable(caption="Moyenne par matière")
```

Table 8: Moyenne par matière

Topic	Mean
English	13.00
Math	13.75
Philo	11.50

Les tableaux `tidy` s'accordent avec `ggplot2`

Hardley Wickham est le programmeur de

- `tidyR`
- `ggplot2`

La philosophie de présentation des données est la même dans les deux package, et permet d'obtenir des analyses complexes en très peu de lignes.

Les graphiques avec ggplot2

Premiers pas

Une question

- Est-que les voitures à gros moteurs consomment plus que les voitures à petits moteurs ?
- À quoi ressemble la relation entre la taille du moteur et l'efficacité énergétique?
- Est-ce positif? Négatif? Linéaire? Non linéaire?

Le data frame `mpg`

`mpg` contient des observations recueillies par l'agence américaine de protection de l'environnement sur 38 modèles de voiture.

```
mpg
```

```
## # A tibble: 234 x 11
##    manufacturer model    displ  year   cyl trans   drv     cty   hwy fl
##    <chr>        <chr>    <dbl> <int> <int> <chr>   <chr> <int> <int> <ch
##  1 audi         a4         1.8  1999     4 auto(l~ f        18    29 p
##  2 audi         a4         1.8  1999     4 manual~ f        21    29 p
##  3 audi         a4         2    2008     4 manual~ f        20    31 p
##  4 audi         a4         2    2008     4 auto(a~ f        21    30 p
##  5 audi         a4         2.8  1999     6 auto(l~ f        16    26 p
##  6 audi         a4         2.8  1999     6 manual~ f        18    26 p
##  7 audi         a4         3.1  2008     6 auto(a~ f        18    27 p
##  8 audi         a4 quat~   1.8  1999     4 manual~ 4        18    26 p
##  9 audi         a4 quat~   1.8  1999     4 auto(l~ 4        16    25 p
## 10 audi         a4 quat~   2    2008     4 manual~ 4        20    28 p
## # ... with 224 more rows, and 1 more variable: class <chr>
```

Parmi les variables dans `mpg`:

1. `displ`, la taille du moteur d'une voiture, en litres.
2. `hwy`, l'efficacité énergétique d'une voiture sur l'autoroute, en miles par gallon (mpg).

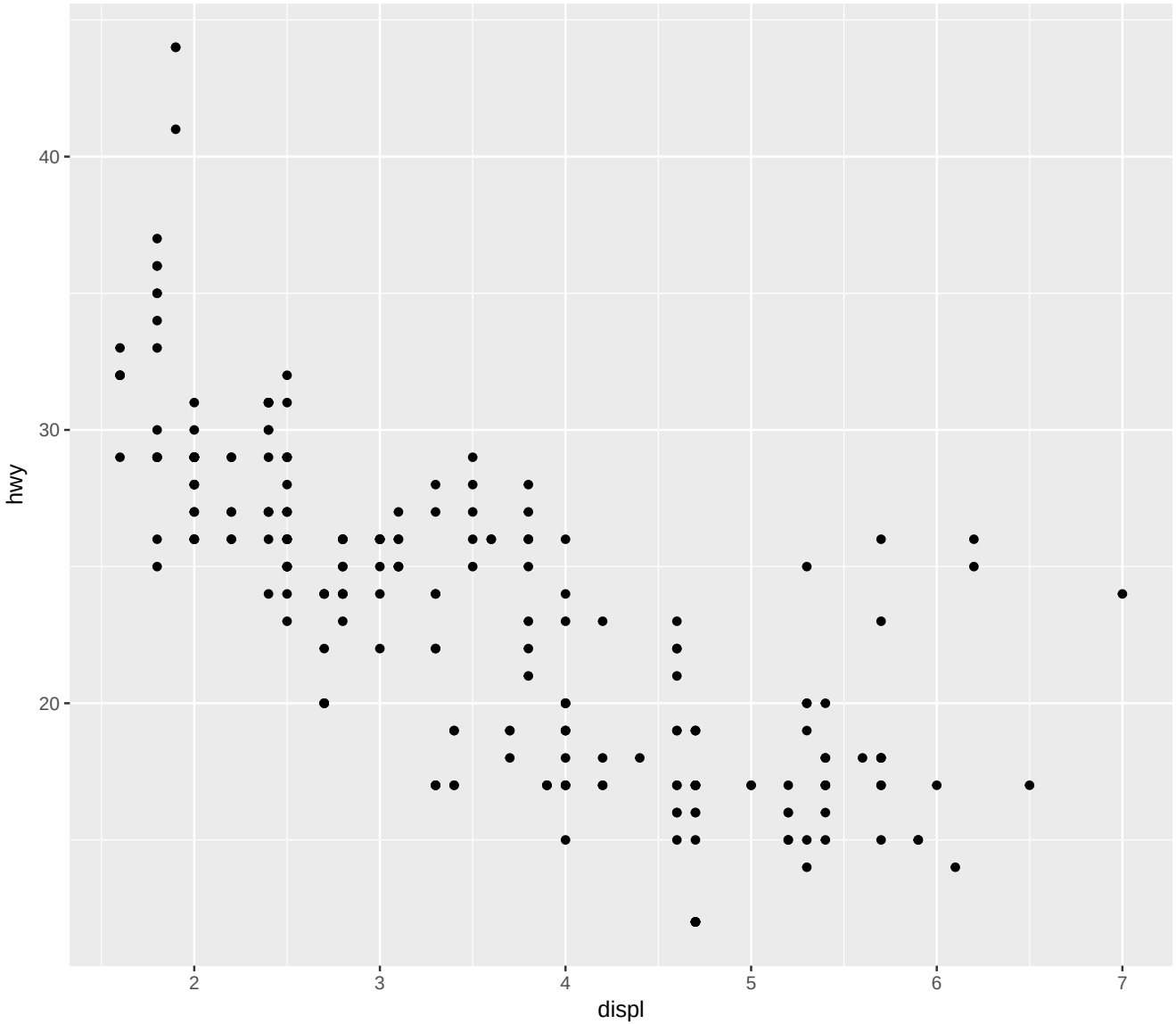
Une voiture à faible consommation de carburant consomme plus de carburant qu'une voiture à haut rendement l'efficacité énergétique lorsqu'ils voyagent à la même distance.

Pour en savoir plus sur `mpg`, ouvrez sa page d'aide en exécutant`?mpg`.

Un graphique avec ggplot I

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

Un graphique avec `ggplot` II



Un graphique avec ggplot III

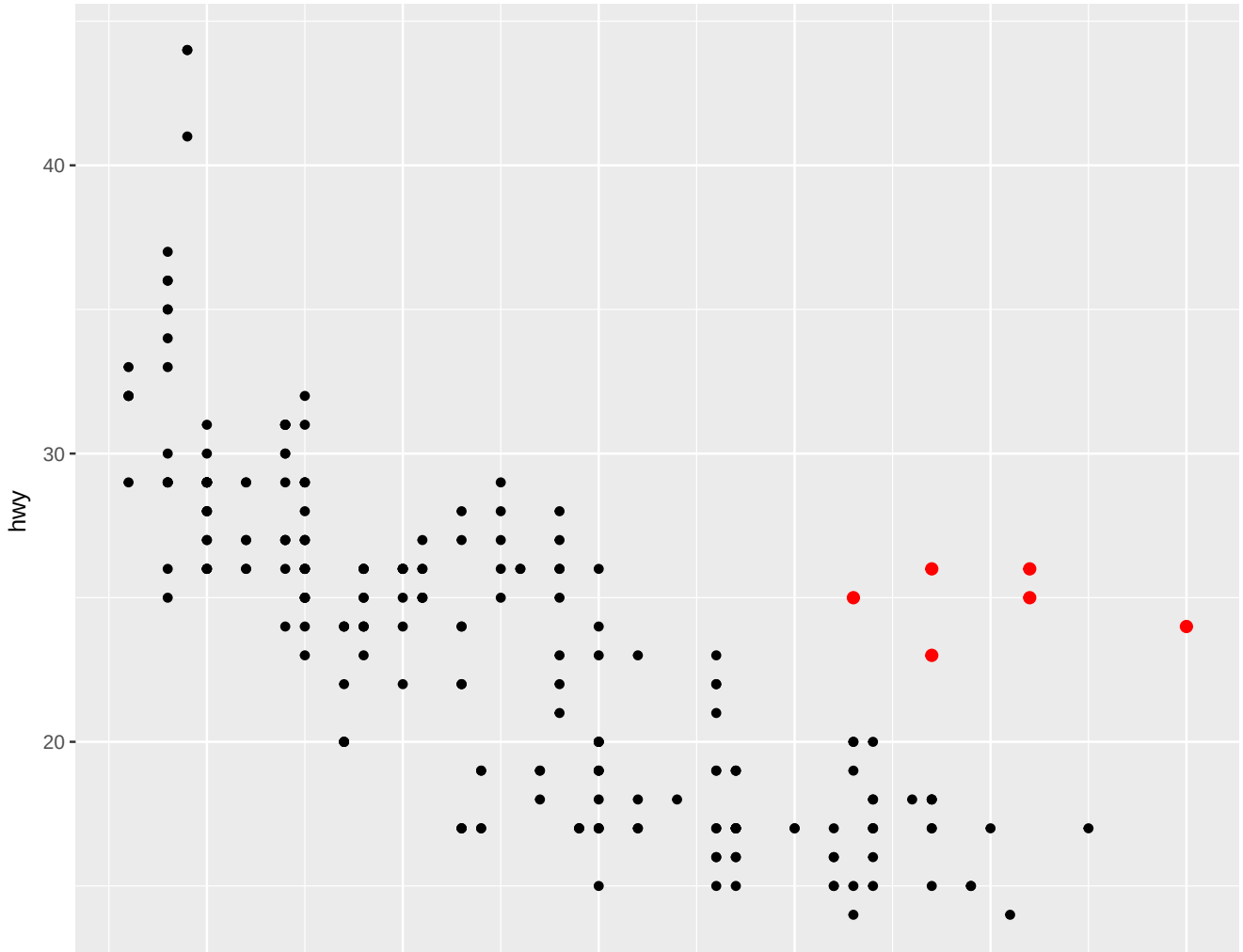
- ggplot (). ggplot () crée un système de coordonnées auquel vous pouvez ajouter des calques.
- ggplot (data = mpg) crée un graphe vide
- geom_point () ajoute une couche de points à votre tracé
- Chaque fonction geom dans ggplot2 prend un argument mapping qui définit comment les variables de votre ensemble de données sont mappées aux propriétés visuelles.
- mapping est toujours associé àaes (),
- x et y deaes ()spécifient les variables à mapper avec les axes x et y.
- ggplot2 recherche la variable mappée dans l'argument data, dans ce cas,mpg.

En résumé

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

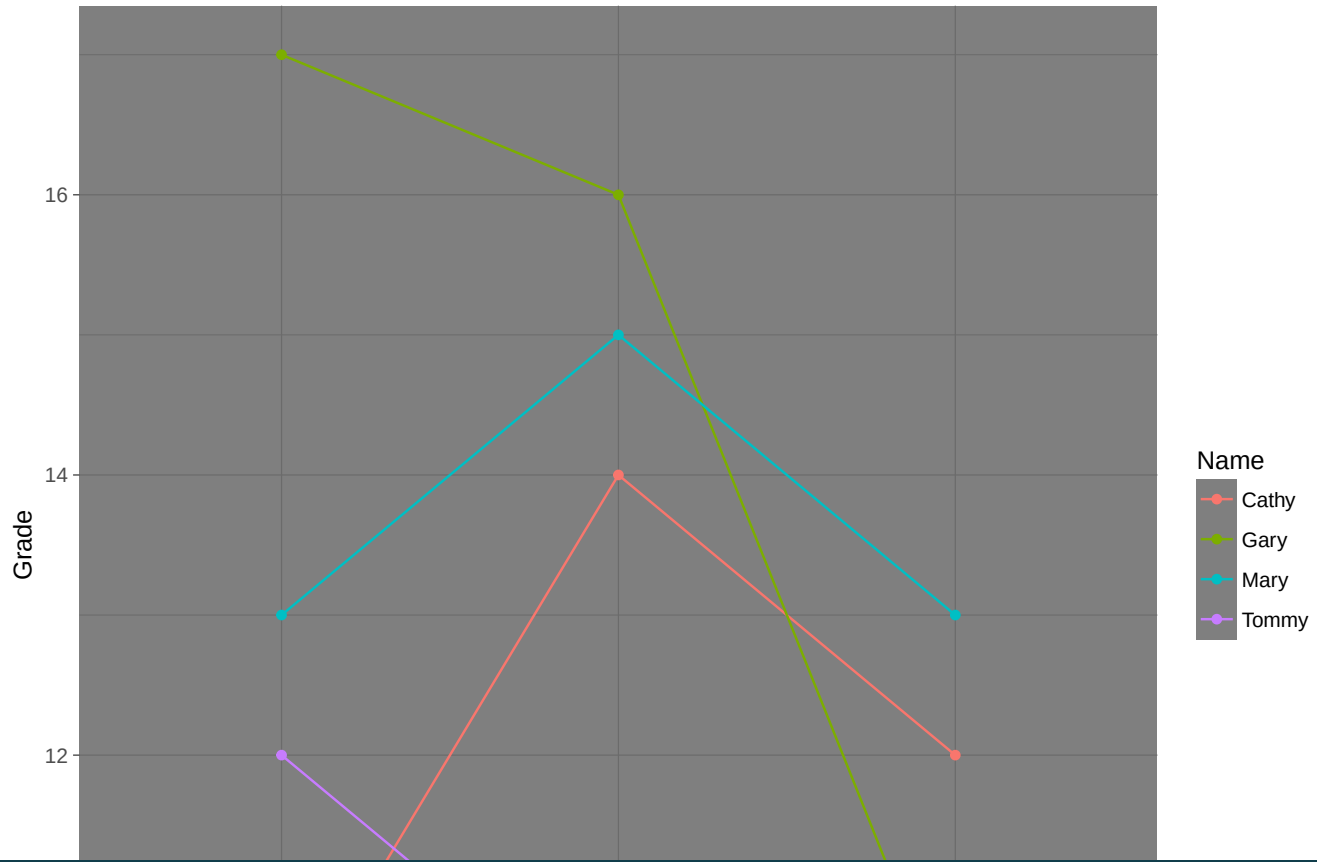
Superpositions de couches

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_point(data = filter(mpg, displ > 5, hwy > 20), colour = "red", size
```



Retour sur les notes

```
grades %>%
  ggplot(aes(Topic, Grade, color = Name)) +
  geom_point() +
  geom_line(aes(group = Name)) +
  theme_dark()
```



Exercices

1. Exécutez `ggplot (data = mpg)`. Que voyez-vous ?
2. Combien de lignes sont dans `mpg`? Combien de colonnes?
3. Que décrit la variable `drv`? Lisez l'aide pour `?mpg` pour trouver en dehors.
4. Faites un nuage de points de `hwy` `vscyl`.
5. Que se passe-t-il si vous faites un nuage de points de `class` `vsdrv`? Pourquoi est-ce n'est ce pas utile?

Réferences