R_Journey

Conseil Important

Pour obtenir de l'aide sur un packages, utiliser : *library(help=**package**)*

Data Pipelines(Pointless Programmation)

```
library(magrittr)
```

Exercices

Exercice 1 Mean of Positive Values

```
rnorm(100) %>% ifelse(.>0,.,NA) %>% mean(., na.rm = TRUE)
```

```
## [1] 0.7588048
```

Exercice 2 Root Mean Square Error

```
d<-data.frame(t=0,y=rnorm(10))
```

```
d %$% c(sum((t-y)**2),1/nrow(.)) %>% (function(c){sqrt(c[1]*c[2])})
```

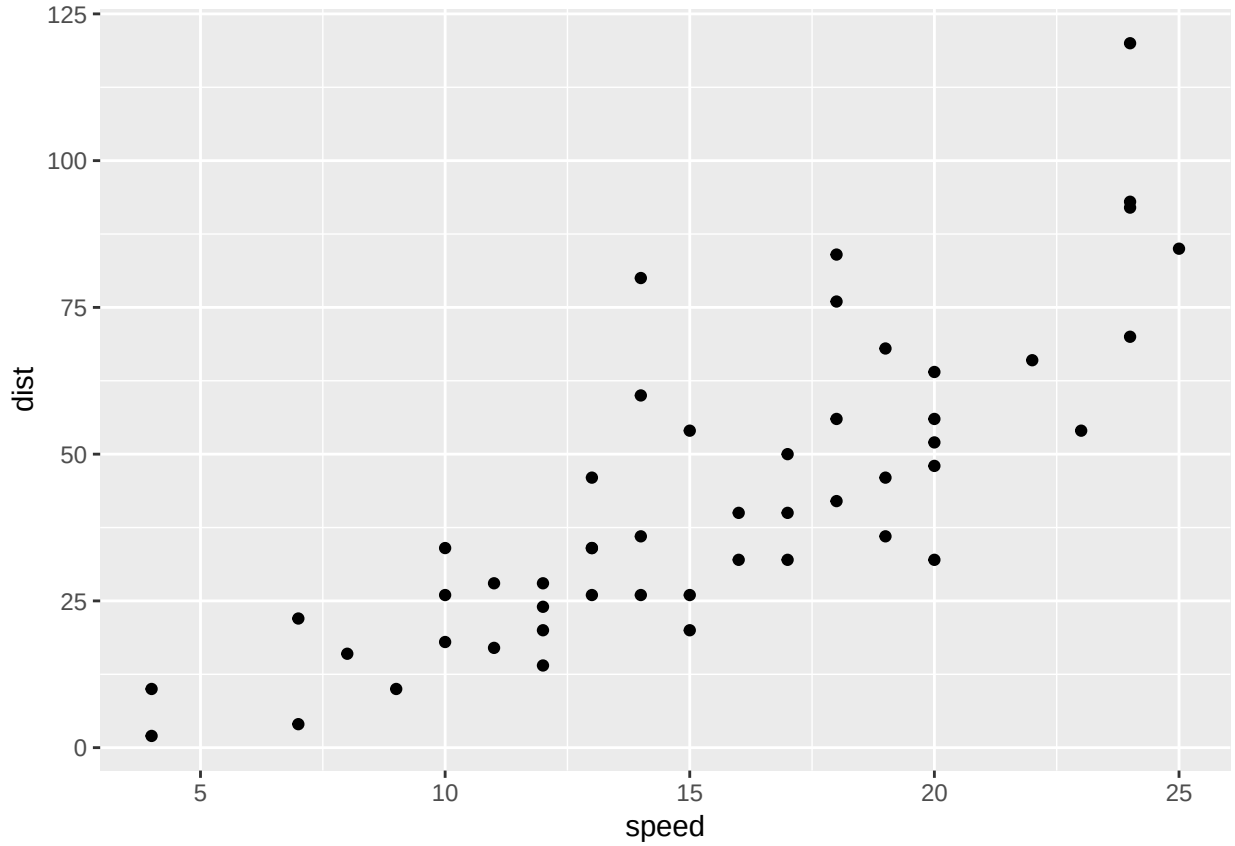
```
## [1] 1.057869
```

Data Manipulation

```
library(datasets)
library(help = "datasets")
library(ggplot2)
data(cars)
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```
cars %>% qplot(speed, dist, data = .)
```



Pour charger les packages

```
library(mlbench)
```

Pour obtenir la description d'un package

```
library(help = "mlbench")
```

```
cars %>% head(3)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
```

```
cars %>% tail(3)
```

```
##    speed dist
## 48    24   93
## 49    24  120
## 50    25   85
```

```
cars %>% summary
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
```

```
##  Max.   :25.0   Max.   :120.00
```

```
data(iris)
iris %>% summary
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Reading and Formatting Data

En général, le code associé à ces tâches est mis dans un chunk avec l'option ***cache=TRUE*** afin de ne pas devoir à chaque fois recharger le code sauf en cas de modifications.

Autre façon, on peut sauvegarder les data sur R, sous forme de fichier **.rda** pour RData.
Ex : ***formatted_breast_cancer %>% save(file = "data/formatted-breast-cancer.rda")***
On peut ensuite les charger à l'aide de : ***load("data/formatted-breast-cancer.rda")*** les données auront alors pour nom *formatted_breast_cancer*

Privilégiez la méthode avec les chunk et le cache ### Examples of Reading and Formatting Datasets : Breast Cancer

```
library(mlbench)
data(BreastCancer)
BreastCancer %>% head(3)
```

```
##        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 1 1000025            5         1          1             1            2
## 2 1002945            5         4          4             5            7
## 3 1015425            3         1          1             1            2
##   Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses  Class
## 1           1           3               1       1 benign
## 2          10           3               2       1 benign
## 3           2           3               1       1 benign
```

Pour lire des données à partir d'une URL

```
data_url="http://tinyurl.com/kw4xtts"
lines <- readLines(data_url)
lines[1:5]
```

```
## [1] "1000025,5,1,1,1,2,1,3,1,1,2"  "1002945,5,4,4,5,7,10,3,2,1,2"
## [3] "1015425,3,1,1,1,2,2,3,1,1,2"  "1016277,6,8,8,1,3,4,3,7,1,2"
## [5] "1017023,4,1,1,3,2,1,3,1,1,2"
```

Pour écrire les données d'une data.frame sur un fichier csv

```
setwd("~/Documents/R_trainning")
writeLines(lines, con = "data/raw-breast-cancer.csv")
```

```
raw_breast_cancer <- read.csv("data/raw-breast-cancer.csv")
raw_breast_cancer %>% head(3)
```

```
##   X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 1  1002945  5  4    4    5  7   10  3    2    1    2
## 2  1015425  3  1    1    1  2    2  3    1    1    2
## 3  1016277  6  8    8    1  3    4  3    7    1    2
```

```
raw_breast_cancer <- read.csv(data_url)
raw_breast_cancer %>% head(3)
```

```
##   X1000025 X5 X1 X1.1 X1.2 X2 X1.3 X3 X1.4 X1.5 X2.1
## 1  1002945  5  4    4    5  7   10  3    2    1    2
## 2  1015425  3  1    1    1  2    2  3    1    1    2
## 3  1016277  6  8    8    1  3    4  3    7    1    2
```

```
raw_breast_cancer <- read.csv(data_url, header = FALSE)
raw_breast_cancer %>% head(3)
```

```
##        V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 1 1000025  5  1  1  1  2  1  3  1   1   2
## 2 1002945  5  4  4  5  7 10  3  2   1   2
## 3 1015425  3  1  1  1  2  2  3  1   1   2
```

```
names(raw_breast_cancer) <- names(BreastCancer)
raw_breast_cancer %>% head(3)
```

```
##        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 1 1000025            5         1          1             1            2
## 2 1002945            5         4          4             5            7
## 3 1015425            3         1          1             1            2
##   Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses Class
## 1           1           3               1       1     2
## 2          10           3               2       1     2
## 3           2           3               1       1     2
```

```
raw_breast_cancer <- read.csv(data_url, header = FALSE, col.names = names(BreastCancer))
raw_breast_cancer %>% head(3)
```

```
##        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
## 1 1000025            5         1          1             1            2
## 2 1002945            5         4          4             5            7
## 3 1015425            3         1          1             1            2
##   Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses Class
## 1           1           3               1       1     2
## 2          10           3               2       1     2
## 3           2           3               1       1     2
```

```
formatted_breast_cancer <- raw_breast_cancer
```

Formatter les données de la variable Class

```
map_class <- function(x) {
    ifelse(x == 2, "bening",
```

```
    ifelse(x == 4, "malignant",
           NA))
}
mapped <- formatted_breast_cancer$Class %>% map_class
mapped %>% table
```

```
## .
##    bening malignant
##       458       241
```

```
# Autre façon:
map_class <- function(x) {
    ifelse(x == 2, "bening", "malignant")
}
mapped <- formatted_breast_cancer$Class %>% map_class
mapped %>% table
```

```
## .
##    bening malignant
##       458       241
```

```
formatted_breast_cancer$Class %>% unique
```

```
## [1] 2 4
```

L'utilisation du ***ifelse*** est assez contraignante, il ne doit y avoir uniquement que deux modalités (ex: maligne et bénine).
Une alternative à cela est l'utilisation d'un dictionnaire, pour créer un dictionnaire en R, on utilise les vecteurs

```
dict <- c("2" = "benign", "4" = "malignant")
map_class <- function(x) dict[as.character(x)]
mapped <- formatted_breast_cancer$Class %>% map_class
mapped %>% table
```

```
## .
##    benign malignant
##       458       241
```

```
mapped[1:5]
```

```
##        2        2        2        2        2
## "benign" "benign" "benign" "benign" "benign"
```

Les nombres ont éte mis entre "" afin que l'on est pas de problème lorsque l'on fait ***dict[2]***.
Comme on peut le voir les étiquettes on pour nom un nombre, pour enlever les noms, on utilise ***uname()***

```
mapped %<>% unname
mapped[1:5]
```

```
## [1] "benign" "benign" "benign" "benign" "benign"
```

Résumé

La lecture et le formattage entier des données peut se faire de cette forme

```
read.csv(data_url, header = FALSE,
         col.names = names(BreastCancer)) ->
  raw_breast_cancer ->
```

```
  formatted_breast_cancer
dict <- c("2" = "benign", "4" = "malignant")
map_class <- function(x) dict[as.character(x)]
formatted_breast_cancer$Class <-formatted_breast_cancer$Class %>%map_class %>%unname %>%
  factor(levels = c("benign", "malignant"))
```

Si l'on ne veut pas utiliser une fonction qu'on a créé pour mapper(formatter) les données pour ne pas s'encombrer, on peut utiliser les lamda expressions.

```
raw_breast_cancer$Class %>%
  { dict <- c("2" = "benign", "4" = "malignant")
    dict[as.character(.)]
  } %>%
  unname %>%
  factor(levels = c("benign", "malignant")) %>%
  table
```

```
## .
##    benign malignant
##       458       241
```

Examples of Reading and Formatting Datasets : Boston Housing

```
library(mlbench)
data(BostonHousing)
str(BostonHousing)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : num  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ b      : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
data_url = "http://tinyurl.com/zq2u8vx"
boston_housing <- read.table(data_url)
str(boston_housing)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ V1 : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ V2 : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ V3 : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ V4 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ V5 : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ V6 : num  6.58 6.42 7.18 7 7.15 ...
```

```
##  $ V7 : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ V8 : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ V9 : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ V10: num  296 242 242 222 222 222 311 311 311 311 ...
##  $ V11: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ V12: num  397 397 393 395 397 ...
##  $ V13: num  4.98 9.14 4.03 2.94 5.33 ...
##  $ V14: num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

Si on observe les données importées de l'URL, on remarque que la variable chas devrait être un facteur et que toutes les autres variables devraient être numériques. On doit aussi renommer les colonnes.
Pour y remédier nous faisons:

```
col_classes <- rep("numeric", length(BostonHousing))
col_classes[which("chas" == names(BostonHousing))] <- "factor"
boston_housing <- read.table(data_url,
                             col.names = names(BostonHousing),
                             colClasses = col_classes)
str(boston_housing)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : num  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ b      : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

The package readr

Le package readr contient des fonctions plus rapides que les fonctions fournis avec R pour la lecture de jeux de données.

```
library(readr)
setwd("~/Documents/R_trainning")
raw_breast_cancer <- read_csv("data/raw-breast-cancer.csv")
```

```
## Warning: Duplicated column names deduplicated: '1' => '1_1' [4], '1' =>
## '1_2' [5], '1' => '1_3' [7], '1' => '1_4' [9], '1' => '1_5' [10], '2' =>
## '2_1' [11]
```

```
## Parsed with column specification:
## cols(
##   `1000025` = col_integer(),
##   `5` = col_integer(),
##   `1` = col_integer(),
##   `1_1` = col_integer(),
##   `1_2` = col_integer(),
```

```
##   `2` = col_integer(),
##   `1_3` = col_character(),
##   `3` = col_integer(),
##   `1_4` = col_integer(),
##   `1_5` = col_integer(),
##   `2_1` = col_integer()
## )
```

```
raw_breast_cancer %>% head(3)
```

```
## # A tibble: 3 x 11
##   `1000025`   `5`   `1` `1_1` `1_2`   `2` `1_3`   `3` `1_4` `1_5` `2_1`
##       <int> <int> <int> <int> <int> <int> <chr> <int> <int> <int> <int>
## 1   1002945     5     4     4     5     7 10        3     2     1     2
## 2   1015425     3     1     1     1     2 2         3     1     1     2
## 3   1016277     6     8     8     1     3 4         3     7     1     2
```

Pour avoir le nom des colonnes, on peut faire:

```
raw_breast_cancer <- read_csv("data/raw-breast-cancer.csv",
                              col_names = names(BreastCancer))
```

```
## Parsed with column specification:
## cols(
##   Id = col_integer(),
##   Cl.thickness = col_integer(),
##   Cell.size = col_integer(),
##   Cell.shape = col_integer(),
##   Marg.adhesion = col_integer(),
##   Epith.c.size = col_integer(),
##   Bare.nuclei = col_character(),
##   Bl.cromatin = col_integer(),
##   Normal.nucleoli = col_integer(),
##   Mitoses = col_integer(),
##   Class = col_integer()
## )
```

```
raw_breast_cancer %>% head(3)
```

```
## # A tibble: 3 x 11
##       Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
##    <int>        <int>     <int>      <int>         <int>        <int>
## 1 1.00e6            5         1          1             1            2
## 2 1.00e6            5         4          4             5            7
## 3 1.02e6            3         1          1             1            2
## # ... with 5 more variables: Bare.nuclei <chr>, Bl.cromatin <int>,
## #   Normal.nucleoli <int>, Mitoses <int>, Class <int>
```

Manipulating Data with dplyr package

Le package ***dplyr*** fournit un certain nombre de fonctions très utiles pour manipuler les dataframes, en utilisant notamment les pipes. avec l'opérateur %>%, il est néanmoins conseiller d'importer aussi le package ***magrittr*** afin d'avoir une meilleure utilisation des pipes

```
library(dplyr)
```

```
##
```

```
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(magrittr)
library(help = dplyr)
iris %>% tbl_df # tbl_df est dépreciée, utilisée à la place tibble::as_tibble
```

```
## # 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
```

select(): Pick Selected Columns and Get Rid of the Rest

La fonction ***select()*** sélectionne les colonnes d'une dataframe(jeu de données). C'est équivalent à indexer les colonnes dans les données.
On peut l'utiliser pour sélectionner une colonne uniquement:

```
iris %>% as_tibble %>% select(Petal.Width) %>% head(3)
```

```
## # A tibble: 3 x 1
##   Petal.Width
##         <dbl>
## 1         0.2
## 2         0.2
## 3         0.2
```

Ou sélectionner plusieurs colonnes :

```
iris %>% as_tibble %>%  select(Sepal.Width, Petal.Length) %>% head(3)
```

```
## # A tibble: 3 x 2
##   Sepal.Width Petal.Length
##         <dbl>        <dbl>
## 1         3.5          1.4
## 2         3            1.4
## 3         3.2          1.3
```

sélectionner une gamme de colonnes:

```
iris %>% as_tibble %>%  select(Sepal.Length:Petal.Length) %>% head(3)
```

```
## # A tibble: 3 x 3
##   Sepal.Length Sepal.Width Petal.Length
##          <dbl>       <dbl>        <dbl>
## 1          5.1         3.5          1.4
## 2          4.9         3            1.4
## 3          4.7         3.2          1.3
```

La réelle utilité de la fonction ***select()*** est qu'elle ressemble au ***SELECT*** des requêtes ***SQL***.
Exemple : Pour sélectionner les colonnes qui commencent par le mot *Petal* ou qui se terminent par *Width* ...

```
iris %>% as_tibble %>%  select(starts_with("Petal")) %>% head(3)
```

```
## # A tibble: 3 x 2
##   Petal.Length Petal.Width
##          <dbl>       <dbl>
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
```

```
iris %>% as_tibble %>%  select(ends_with("Width")) %>% head(3)
```

```
## # A tibble: 3 x 2
##   Sepal.Width Petal.Width
##         <dbl>       <dbl>
## 1         3.5         0.2
## 2         3           0.2
## 3         3.2         0.2
```

```
iris %>% as_tibble %>%  select(contains("etal")) %>% head(3)
```

```
## # A tibble: 3 x 2
##   Petal.Length Petal.Width
##          <dbl>       <dbl>
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
```

On peut utiliser la recherche de motif sur le noms des colonnes, en utilisant les expressions régulières :

```
iris %>% as_tibble %>%  select(matches(".t.")) %>% head(3)
```

```
## # A tibble: 3 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##          <dbl>       <dbl>        <dbl>       <dbl>
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3            1.4         0.2
## 3          4.7         3.2          1.3         0.2
```

On peut aussi effacer des colonnes:

```
iris %>% as_tibble %>%  select(-starts_with("Petal")) %>% head(3)
```

```
## # A tibble: 3 x 3
##   Sepal.Length Sepal.Width Species
##          <dbl>       <dbl> <fct>
## 1          5.1         3.5 setosa
## 2          4.9         3   setosa
```

```
## 3          4.7         3.2 setosa
```

mutate():Add Computed Values to Your Data Frame

La fonction ***mutate()*** permet d'ajouter des colonnes au jeu de données en spécifiant l'expression de comment sont créer ces colonnes, comme par exemple en sommant deux colonnes.
Voici un exemple d'ajout d'une colonne:

```
iris %>% as_tibble %>%
  mutate(Petal.Width.plus.Length = Petal.Width + Petal.Length) %>%
  select(Species, Petal.Width.plus.Length) %>%  head(3)
```

```
## # A tibble: 3 x 2
##   Species Petal.Width.plus.Length
##   <fct>                     <dbl>
## 1 setosa                     1.60
## 2 setosa                     1.60
## 3 setosa                     1.5
```

On peut aussi ajouter plusieurs colonnes en une seule fois :

```
iris %>% as_tibble %>%
  mutate(Petal.Width.plus.Length = Petal.Width + Petal.Length,
         Sepal.Width.plus.Length = Sepal.Width + Sepal.Length) %>%
  select(Petal.Width.plus.Length, Sepal.Width.plus.Length) %>%  head(3)
```

```
## # A tibble: 3 x 2
##   Petal.Width.plus.Length Sepal.Width.plus.Length
##                     <dbl>                   <dbl>
## 1                    1.60                     8.6
## 2                    1.60                     7.9
## 3                    1.5                      7.9
```

Ou encore appeler plusieurs fois la fonction ***mutate()***.

transmute(): Add Computed Values to Your Data Frame and Get Rid of All Other Columns

transmute() fonctionne de façon similaire à la fonction ***mutate()*** combiné à ***select()***, d'où le résultat est de cette fonction est une dataframe qui contient uniquement les nouvelles colonnes créées.

```
my_data<-iris %>% as_tibble %>%
  transmute(Petal.Width.plus.Length = Petal.Width + Petal.Length) %>% head(3)
class(my_data)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

arrange(): Reorder Your Data Frame by Sorting Columns

arrange() permet de classer la dataframe en classant les colonnes selon ce dont l'on spécifie.

```
iris %>% as_tibble %>% arrange(Sepal.Length) %>% head(3)
```

```
## # A tibble: 3 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          4.3         3            1.1         0.1 setosa
```

```
## 2          4.4         2.9          1.4         0.2 setosa
## 3          4.4         3            1.3         0.2 setosa
```

Par défaut, pour les valeurs numériques le classement se fait par ordre croissant. Pour changer l'ordre de classement, on utilise ***desc*** :

```
iris %>% as_tibble %>%   arrange(desc(Sepal.Length)) %>%  head(3)
```

```
## # A tibble: 3 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          7.9         3.8          6.4         2   virginica
## 2          7.7         3.8          6.7         2.2 virginica
## 3          7.7         2.6          6.9         2.3 virginica
```

filter(): Pick Selected Rows and Get Rid of the Rest

arrange() permet de filtrer les lignes selon une condition logique. On donne en paramètre de la fonction un prédicat précisant quelles lignes doivent être choisies.

```
iris %>% as_tibble %>%
  filter(Sepal.Length > 5) %>% head(3)
```

```
## # A tibble: 3 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          5.4         3.9          1.7         0.4 setosa
## 3          5.4         3.7          1.5         0.2 setosa
```

```
# Autre exemple
iris %>% as_tibble() %>%
  filter(Sepal.Length > 5 & Species == "virginica") %>%
  select(Species, Sepal.Length) %>% head(3)
```

```
## # A tibble: 3 x 2
##   Species   Sepal.Length
##   <fct>            <dbl>
## 1 virginica          6.3
## 2 virginica          5.8
## 3 virginica          7.1
```

group_by(): Split Your Data Into Subtables Based on Column Values

group_by() permet de spécifier à **dplyr** que l'on veut travailler sur les données séparées en différents sous-ensemble. En elle-même la fonction n'est pas utile, elle permet juste de spécifier à **dplyr** que pour les futures opérations, il doit considérer différents sous-ensembles de données comme des jeux de données séparés. L'intérêt de cette fonction vient lorsque l'on l'utilise avec la fonction ***summarize()*** pour obtenir des statistiques synthétiques. On peut grouper par une ou plusieurs variables, il faut juste spécifier les colonnes par lesquelles on veut grouper comme des arguments séparés. Le groupement est efficace si les données sont autres que des nombres réels.

```
iris %>% as_tibble %>% group_by(Species) %>% head(3)
```

```
## # A tibble: 3 x 5
## # Groups:   Species [1]
```

```
##   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
```

Nous avons restructuré la data frame tel qu'ils y a des groupement, mais en attendant que l'on effectue de nouvelles opérations sur les données, il n'y a rien à constater de plus. Le réel intérêt est lorsque la fonction est associée à la fonction ***summarize()***.

summarise/summarize(): Calculate Summary Statistics

summarise est utilisée pour générer des statistiques synthétiques à partir de la data frane. Elle permet d'obtenir différentes statistiques en choisissant la variable que l'on veut(même principe que les proc en SAS). Exemple:

```
iris %>% summarise(Mean.Petal.Length = mean(Petal.Length),
              Mean.Sepal.Length = mean(Sepal.Length))
```

```
##   Mean.Petal.Length Mean.Sepal.Length
## 1             3.758          5.843333
```

```
# Exemple avec group by
iris %>%  group_by(Species) %>%
  summarise(Mean.Petal.Length = mean(Petal.Length))
```

```
## # A tibble: 3 x 2
##   Species    Mean.Petal.Length
##   <fct>                  <dbl>
## 1 setosa                  1.46
## 2 versicolor              4.26
## 3 virginica               5.55
```

Voici une liste de fonctions utiles à utiliser avec ***summarise*** :
* Center: ***mean()***, ***median()***,
* Spread: ***sd()*** , ***IQR()***, ***mad()***,
* Range: ***min()***, ***max()***, ***quantile()***,
* Position: ***first()***, ***last()***, ***nth()***,
* Count: ***n()***, ***n_distinct()***,
* Logical: ***any()***, ***all()***

```
iris %>% summarise(Observations = n())
```

```
##   Observations
## 1          150
```

```
# Exemple avec group by
iris %>% group_by(Species) %>% summarise(Number.Of.Species = n())
```

```
## # A tibble: 3 x 2
##   Species    Number.Of.Species
##   <fct>                  <int>
## 1 setosa                    50
## 2 versicolor                50
## 3 virginica                 50
```

Enfin on peut combiner plusieurs statistiques comme ceci :

```r
iris %>%  group_by(Species) %>%
  summarise(Number.Of.Samples = n(),
            Mean.Petal.Length = mean(Petal.Length))
```

```
## # A tibble: 3 x 3
##   Species    Number.Of.Samples Mean.Petal.Length
##   <fct>                  <int>             <dbl>
## 1 setosa                    50              1.46
## 2 versicolor                50              4.26
## 3 virginica                 50              5.55
```

Exemple Breast Cancer Data Manipulation

```r
formatted_breast_cancer <-
  raw_breast_cancer %>%
  mutate(
    Class = Class %>% {c("2" = "benign", "4" = "malignant")[as.character(.)]} %>%
    unname %>% factor(levels = c("benign", "malignant"))
  )

format_class <- . %>% {
  dict <- c("2" = "benign", "4" = "malignant")
  dict[as.character(.)]
} %>% unname %>% factor(levels = c("benign", "malignant"))

formatted_breast_cancer <-
  raw_breast_cancer %>% mutate(Class = format_class(Class))

formatted_breast_cancer %>%
  group_by(Class) %>%
  summarise(mean.thickness = mean(Cl.thickness))
```

```
## # A tibble: 2 x 2
##   Class     mean.thickness
##   <fct>              <dbl>
## 1 benign              2.96
## 2 malignant           7.20
```

```r
formatted_breast_cancer %>%
  group_by(Class) %>%
  summarise(mean.size = mean(Cell.size))
```

```
## # A tibble: 2 x 2
##   Class     mean.size
##   <fct>         <dbl>
## 1 benign         1.33
## 2 malignant      6.57
```

```r
formatted_breast_cancer %>%
  arrange(Cell.size) %>%
  group_by(Cell.size, Class) %>%
  summarise(ClassCount = n())
```

```
## # A tibble: 18 x 3
## # Groups:   Cell.size [?]
```

```
##    Cell.size Class     ClassCount
##        <int> <fct>          <int>
##  1         1 benign           380
##  2         1 malignant          4
##  3         2 benign            37
##  4         2 malignant          8
##  5         3 benign            27
##  6         3 malignant         25
##  7         4 benign             9
##  8         4 malignant         31
##  9         5 malignant         30
## 10         6 benign             2
## 11         6 malignant         25
## 12         7 benign             1
## 13         7 malignant         18
## 14         8 benign             1
## 15         8 malignant         28
## 16         9 benign             1
## 17         9 malignant          5
## 18        10 malignant         67
```

```
formatted_breast_cancer %>%
  group_by(Class, as.factor(Cell.size)) %>%
  summarise(mean.thickness = mean(Cl.thickness))
```

```
## # A tibble: 18 x 3
## # Groups:   Class [?]
##    Class     `as.factor(Cell.size)` mean.thickness
##    <fct>     <fct>                           <dbl>
##  1 benign    1                                2.76
##  2 benign    2                                3.49
##  3 benign    3                                3.81
##  4 benign    4                                5.11
##  5 benign    6                                5
##  6 benign    7                                5
##  7 benign    8                                6
##  8 benign    9                                6
##  9 malignant 1                                7.25
## 10 malignant 2                                6.75
## 11 malignant 3                                6.44
## 12 malignant 4                                7.71
## 13 malignant 5                                6.87
## 14 malignant 6                                6.88
## 15 malignant 7                                6.89
## 16 malignant 8                                7.18
## 17 malignant 9                                8.8
## 18 malignant 10                               7.52
```

Tidying Data with tidyr

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns, and tables are matched up with observations, variables, and types.

Peut se comprenrdre comme une façon efficace de résumer avec des statistiques synthétiques et des graphes

les données.

Pour cela on va se concentrer sur des colonnes en particulier dans notre dataframe

```
library(tidyr)
```

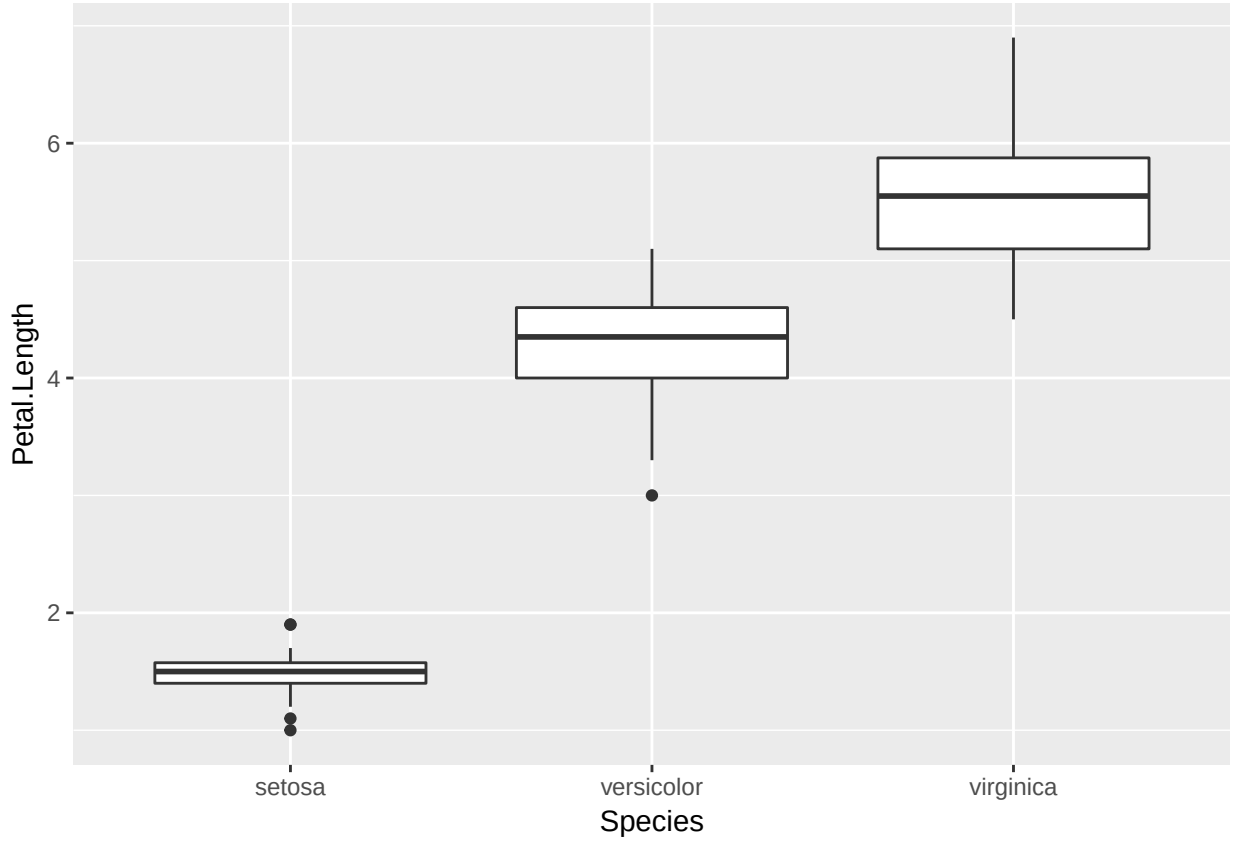
```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:magrittr':
##
##     extract
```

```
iris %>% select(Species, Petal.Length) %>% head(3)
```

```
##   Species Petal.Length
## 1  setosa          1.4
## 2  setosa          1.4
## 3  setosa          1.3
```

```
iris %>% select(Species, Petal.Length) %>%
  qplot(Species, Petal.Length, geom = "boxplot", data = .)
```



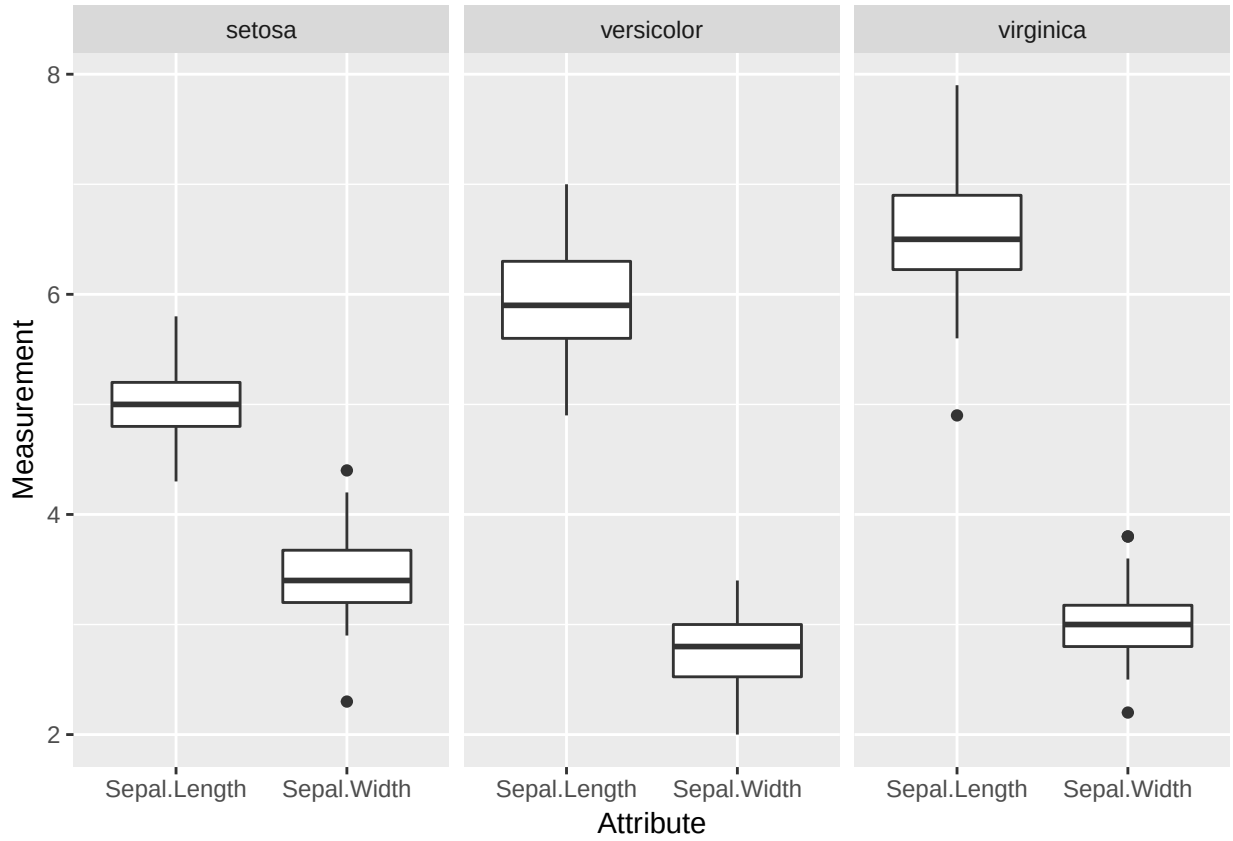
gather()

La fonction ***gather*** permet de fusionner des colonnes en lignes(voir aide). Dans l'exemple qui suit *Sepal.Length* et *Sepal.Width* sont devenus une seule et même ligne avec en début de ligne *Sepal.Length* et en fin de ligne *Sepal.Width*, cela ressemble à ***rbind***

```
g<-iris %>%
  gather(key = Attribute, value = Measurement,
         Sepal.Length, Sepal.Width) %>%
  select(Species, Attribute, Measurement)
g %>% head(3)
```

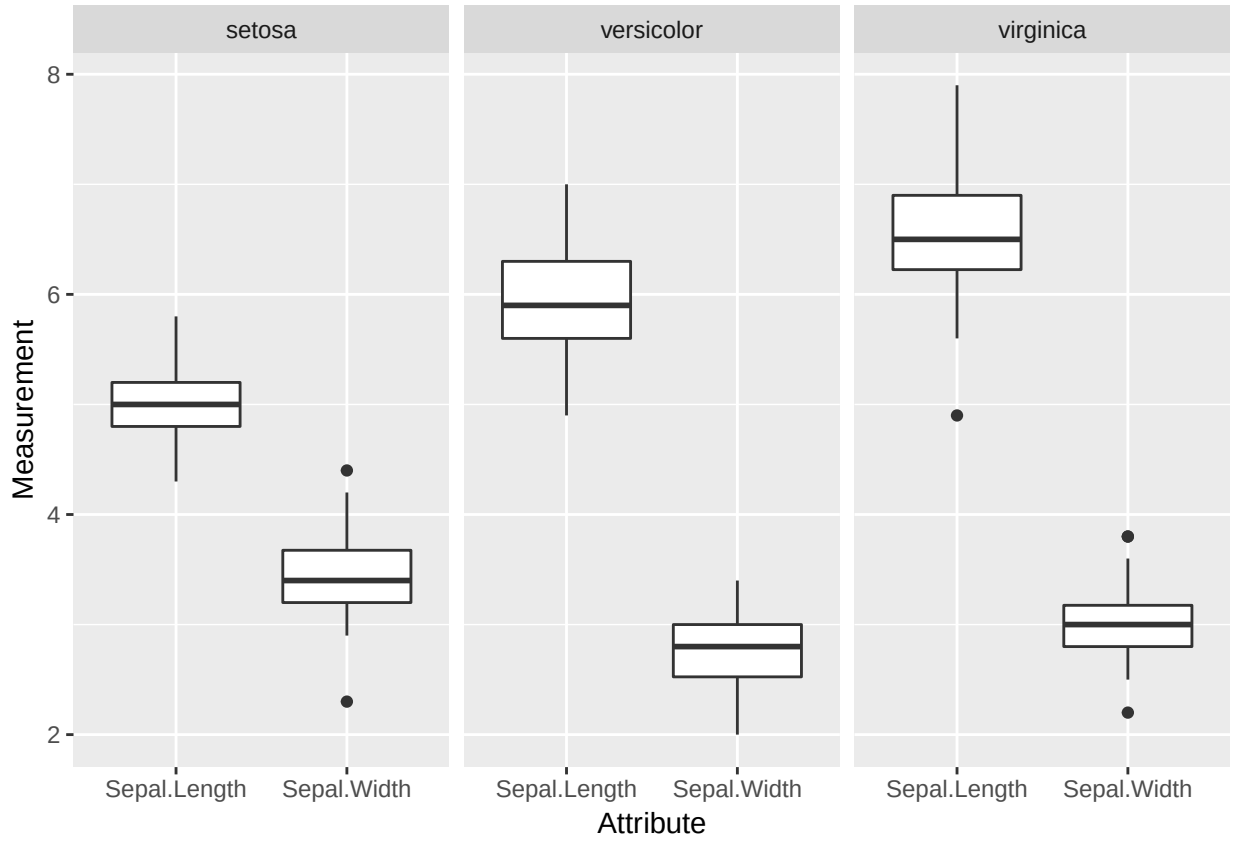
```
##   Species    Attribute Measurement
## 1  setosa Sepal.Length         5.1
## 2  setosa Sepal.Length         4.9
## 3  setosa Sepal.Length         4.7
```

```
iris %>%
  gather(key = Attribute, value = Measurement,
         Sepal.Length, Sepal.Width) %>%
  select(Species, Attribute, Measurement) %>%
  qplot(Attribute, Measurement,
          geom = "boxplot",
          facets = . ~ Species, data = .)
```



This code tells gather() to make a column called Attributes that contains the names of columns from the input data frame and another called Measurement that will contain the values of the key columns. From the resulting data frame, you can see that the Attribute column contains the Sepal.Length and Sepal. Width names (well, you can see it if you don't run it through head(); in the output here you only see Sepal. Length), and another column that shows the Measurements. This transforms the data into a form where we can plot the attributes against measurements (see Figure 3-3 for the result).

```
iris %>%
  gather(key = Attribute, value = Measurement,
         Sepal.Length, Sepal.Width) %>%
  select(Species, Attribute, Measurement) %>%
  qplot(Attribute, Measurement,
          geom = "boxplot",
          facets = . ~ Species, data = .)
```



Exercices

```
my_data <- read_csv("data/Loan_payments.csv")
```

```
## Parsed with column specification:
## cols(
##   Loan_ID = col_character(),
##   loan_status = col_character(),
##   Principal = col_integer(),
##   terms = col_integer(),
##   effective_date = col_character(),
##   due_date = col_character(),
##   paid_off_time = col_character(),
##   past_due_days = col_integer(),
##   age = col_integer(),
##   education = col_character(),
```

```
##   Gender = col_character()
## )
```

```
my_data %>% as_tibble %>% head(3)
```

```
## # A tibble: 3 x 11
##   Loan_ID loan_status Principal terms effective_date due_date paid_off_time
##   <chr>   <chr>           <int> <int> <chr>          <chr>    <chr>
## 1 xqd201~ PAIDOFF          1000    30 9/8/2016       10/7/20~ 9/14/2016 19~
## 2 xqd201~ PAIDOFF          1000    30 9/8/2016       10/7/20~ 10/7/2016 9:~
## 3 xqd201~ PAIDOFF          1000    30 9/8/2016       10/7/20~ 9/25/2016 16~
## # ... with 4 more variables: past_due_days <int>, age <int>,
## #   education <chr>, Gender <chr>
```

```
my_data %>% as_tibble %>% select(Principal,age) %>% head(3)
```

```
## # A tibble: 3 x 2
##   Principal   age
##       <int> <int>
## 1      1000    45
## 2      1000    50
## 3      1000    33
```

```
my_data %>% as_tibble %>% arrange(age) %>% head(3)
```

```
## # A tibble: 3 x 11
##   Loan_ID loan_status Principal terms effective_date due_date paid_off_time
##   <chr>   <chr>           <int> <int> <chr>          <chr>    <chr>
## 1 xqd201~ COLLECTION       1000    30 9/11/2016      10/10/2~ <NA>
## 2 xqd201~ PAIDOFF          1000    30 9/13/2016      10/12/2~ 10/12/2016 2~
## 3 xqd201~ PAIDOFF          1000    30 9/13/2016      10/12/2~ 10/12/2016 9~
## # ... with 4 more variables: past_due_days <int>, age <int>,
## #   education <chr>, Gender <chr>
```

```
my_data %>% as_tibble %>% arrange(desc(age)) %>% head(3)
```

```
## # A tibble: 3 x 11
##   Loan_ID loan_status Principal terms effective_date due_date paid_off_time
##   <chr>   <chr>           <int> <int> <chr>          <chr>    <chr>
## 1 xqd201~ PAIDOFF           800    15 9/13/2016      9/27/20~ 9/26/2016 7:~
## 2 xqd201~ PAIDOFF          1000    30 9/8/2016       10/7/20~ 10/7/2016 9:~
## 3 xqd201~ PAIDOFF           800    15 9/11/2016      9/25/20~ 9/25/2016 19~
## # ... with 4 more variables: past_due_days <int>, age <int>,
## #   education <chr>, Gender <chr>
```

```
my_data %>% as_tibble %>% filter(Principal>900 & age>40) %>% select(age, Principal) %>% head(3)
```

```
## # A tibble: 3 x 2
##     age Principal
##   <int>     <int>
## 1    45      1000
## 2    50      1000
## 3    43      1000
```

```
my_data %>% group_by(Principal) %>% summarise(mean.age = mean(age))
```

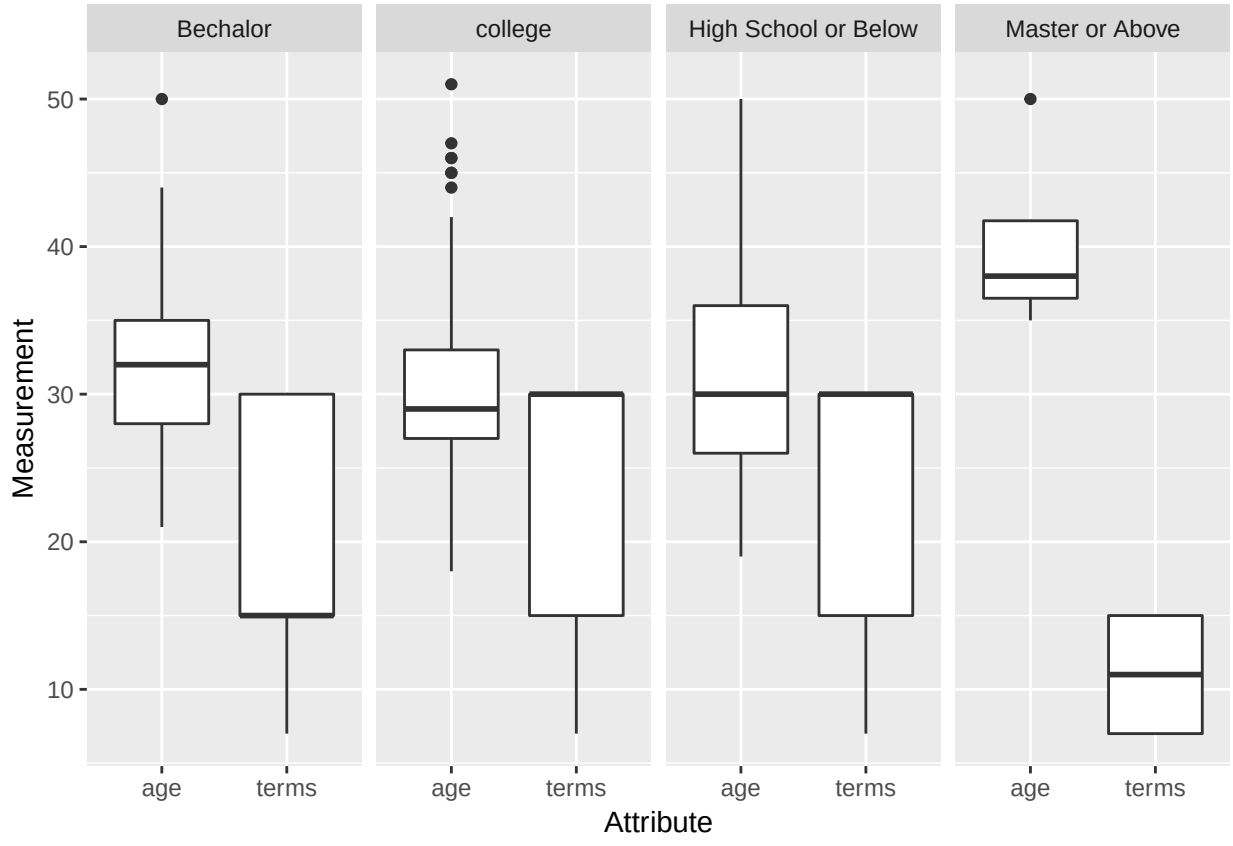
```
## # A tibble: 6 x 2
##   Principal mean.age
```

```
##       <int>    <dbl>
## 1       300     31.3
## 2       500     27.3
## 3       700     33
## 4       800     32.8
## 5       900     27
## 6      1000     30.7
```

```
my_data %>% group_by(Principal) %>% summarise(count.age = n_distinct(age))
```

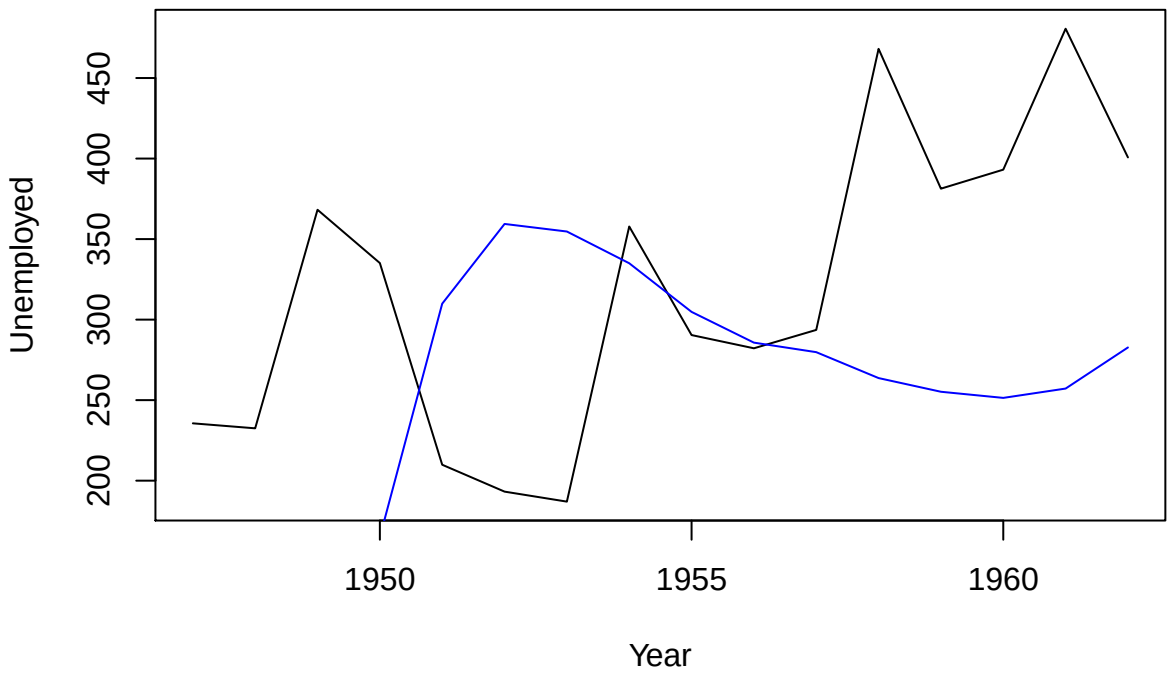
```
## # A tibble: 6 x 2
##   Principal count.age
##       <int>     <int>
## 1       300         4
## 2       500         3
## 3       700         1
## 4       800        28
## 5       900         2
## 6      1000        32
```

```
my_data %>% gather(key = Attribute, value = Measurement,terms, age) %>%
  select(education, Attribute, Measurement) %>%
  qplot(Attribute, Measurement,geom = "boxplot",facets = .~education, data = .)
```

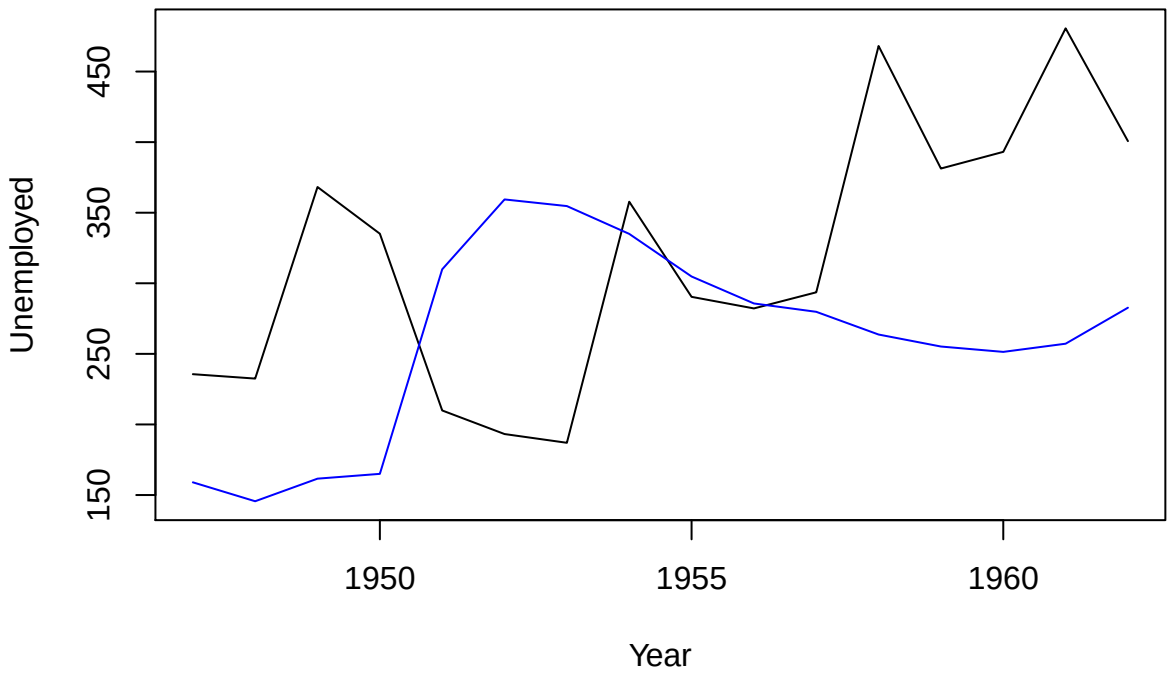


Visualizing Data

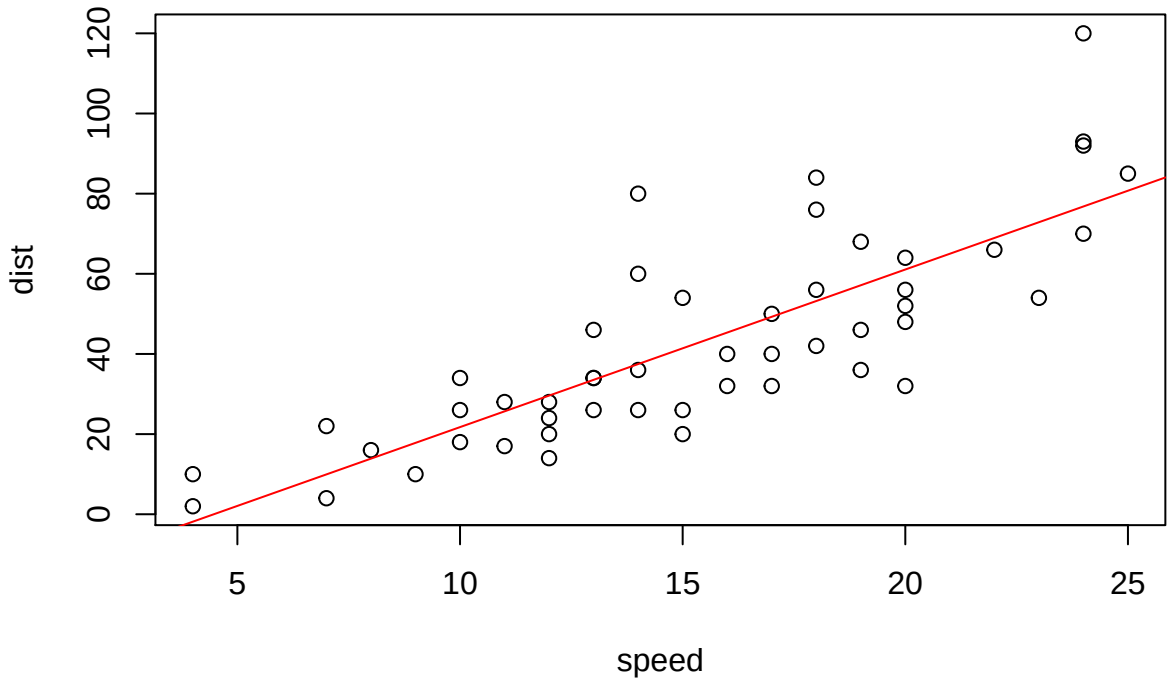
```
longley %>% plot(Unemployed ~ Year, data = ., type = 'l')
longley %>% lines(Armed.Forces ~ Year, data = ., col = "blue")
```



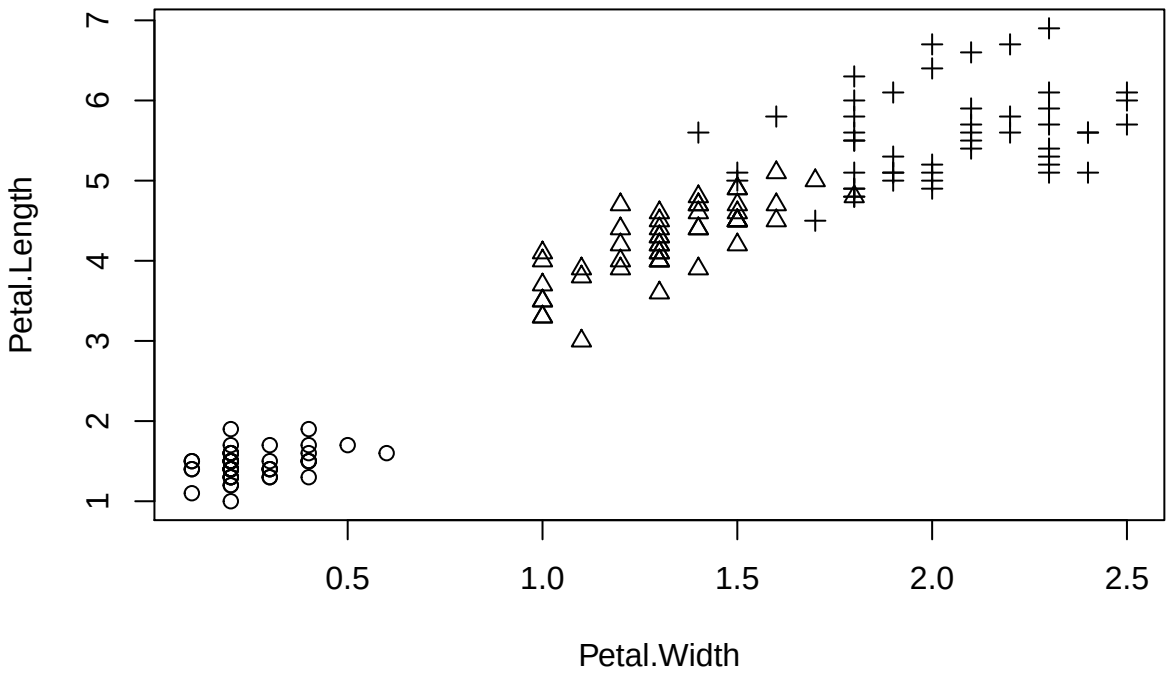

```
longley %$% plot(Unemployed ~ Year, type = 'l',
                 ylim = range(c(Unemployed, Armed.Forces)))
longley %>% lines(Armed.Forces ~ Year, data = ., col = "blue")
```




```
cars %>% plot(dist ~ speed, data = .)
cars %>% lm(dist ~ speed, data = .) %>% abline(col = "red")
```




```
shape_map <- c("setosa" = 1, "versicolor" = 2, "virginica" = 3)
iris %$% plot(Petal.Length ~ Petal.Width, pch = shape_map[Species])
```



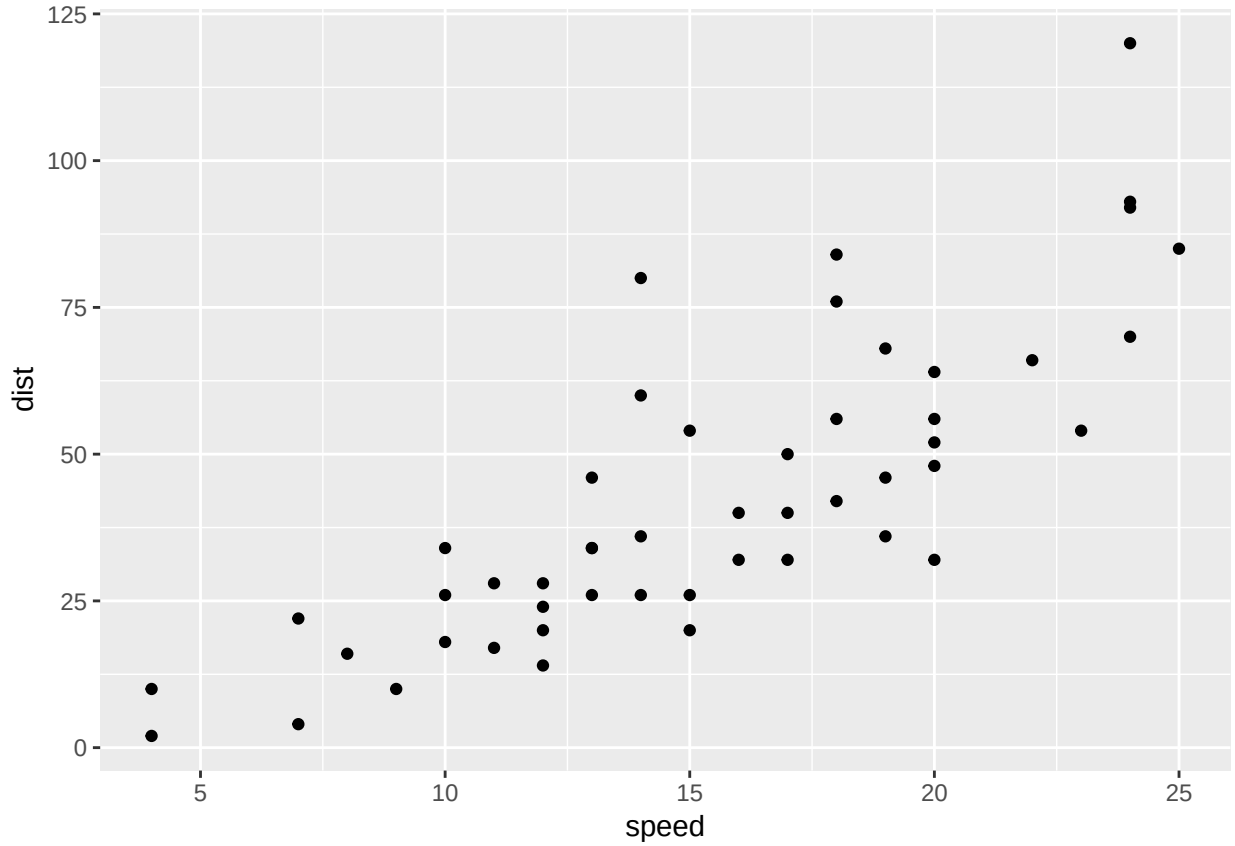
The Grammar of Graphics and the ggplot2 Package

```
library(help = "ggplot2")
```

Using qplot

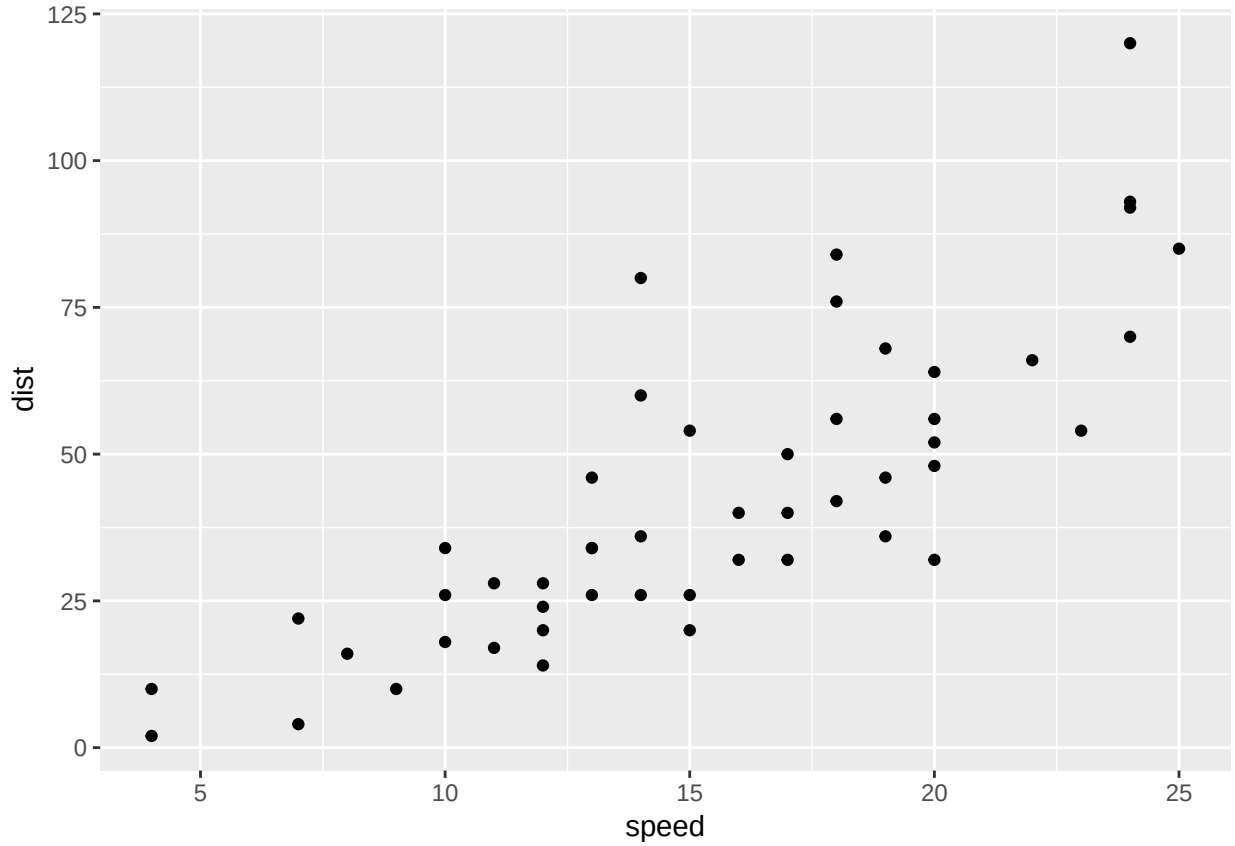
Cette fonction peut être utilisé pour faire de simples graphiques

```
cars %>% qplot(speed, dist, data = .)
```



L'avantage du package ggplot2 est que les graphique deviennent des objets que l'on peut modifier pour infine afficher :

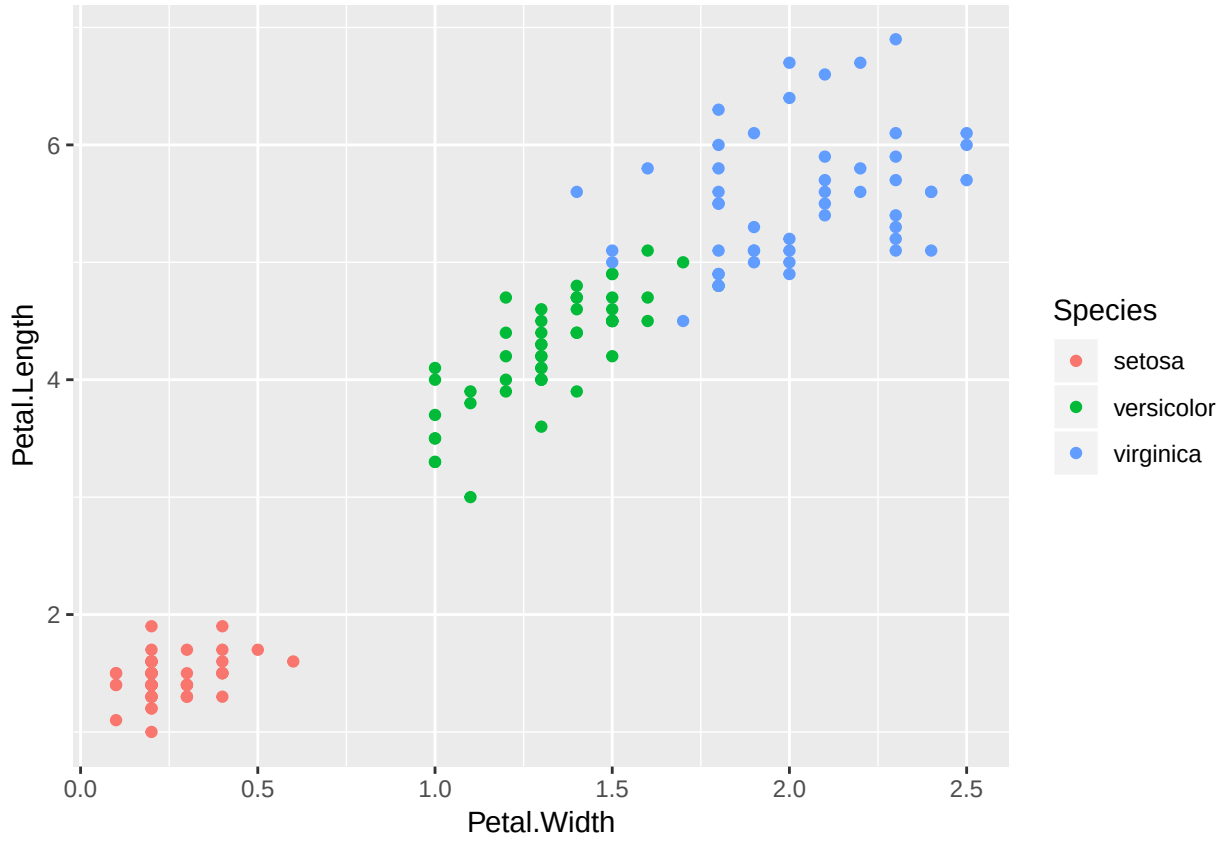
```
p <- cars %>% qplot(speed, dist, data = .)
p
```




```
# est équivalent à
# p <- cars %>% qplot(speed, dist, data = .)
# print(p)
```

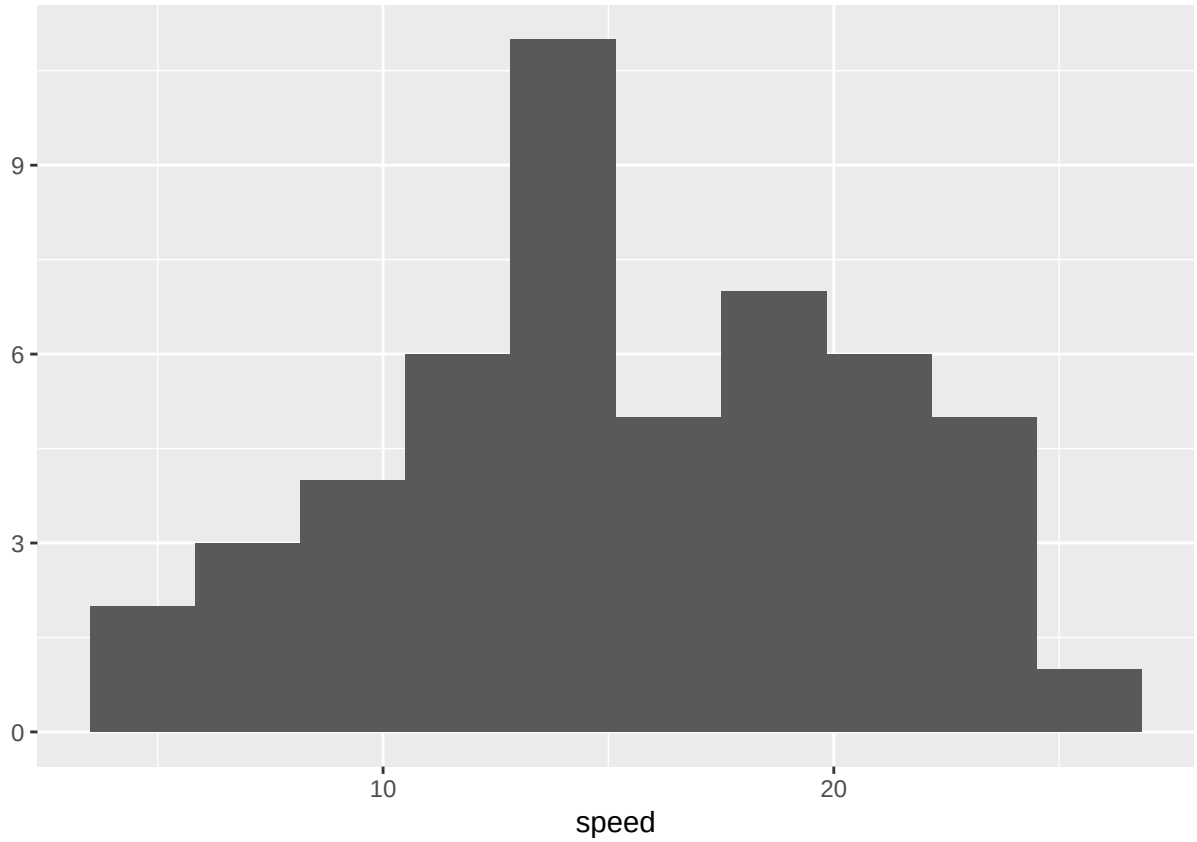
Avec ***qplot*** on peut obtenir un meilleur affichage plus simplement qu'avec la fonction ***plot***. Pour colorer avec la fonction ***plot***, il faut créer une fonction qui créer un vecteur de nombres correspondant à chaque étiquette d'espèce.
Avec ***qplot***, on a juste besoin de spécifier que les couleurs dépendent de la colonne de la variable *Species*

```
iris %>% qplot(Petal.Width, Petal.Length ,
               color = Species, data = .)
```



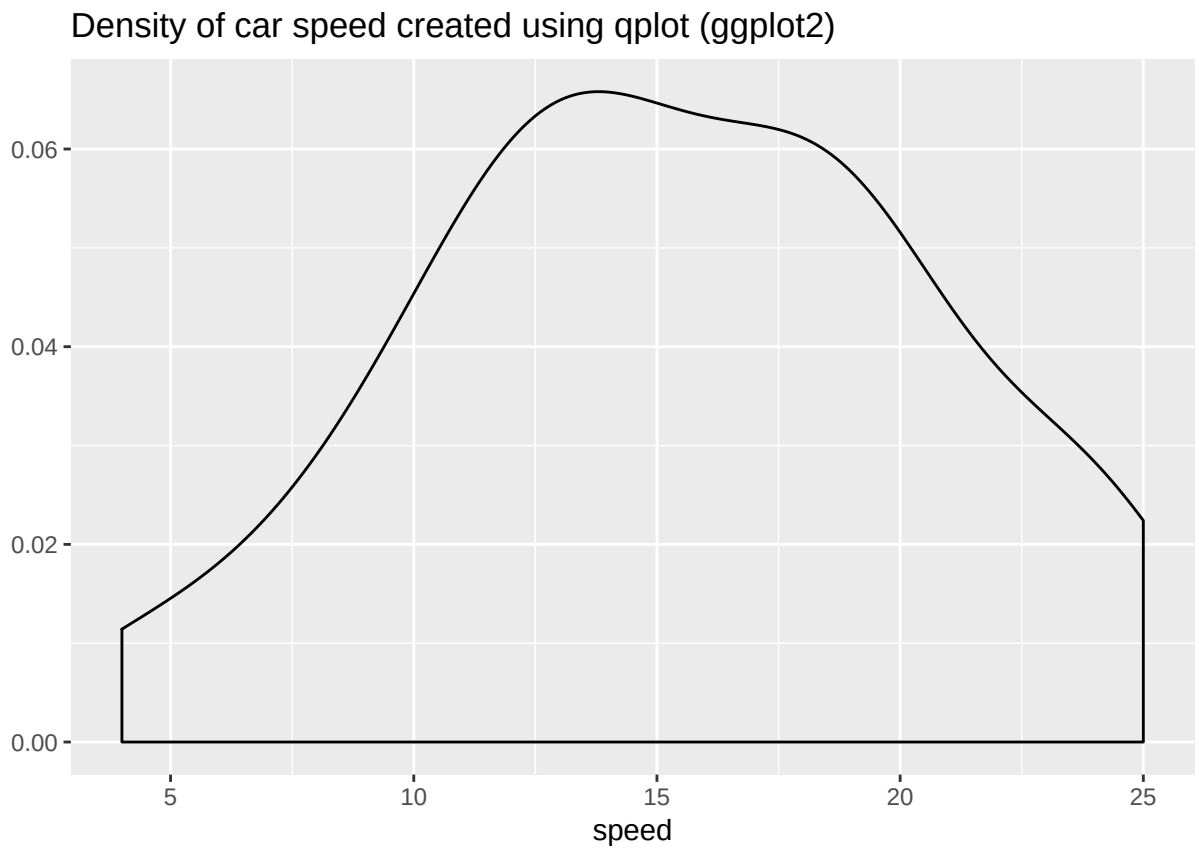
On peut utiliser ***qplot*** pour tracer d'autres graphiques que des nuages de points. Par exemple si l'on donne en paramètre une seule variable, ***qplot*** va alors comprendre qu'il faut tracer un histogramme:

```
cars %>% qplot(speed, data = ., bins = 10)
```



Si on veut afficher la densité, il n'y a qu'à préciser que ***geom = density***:

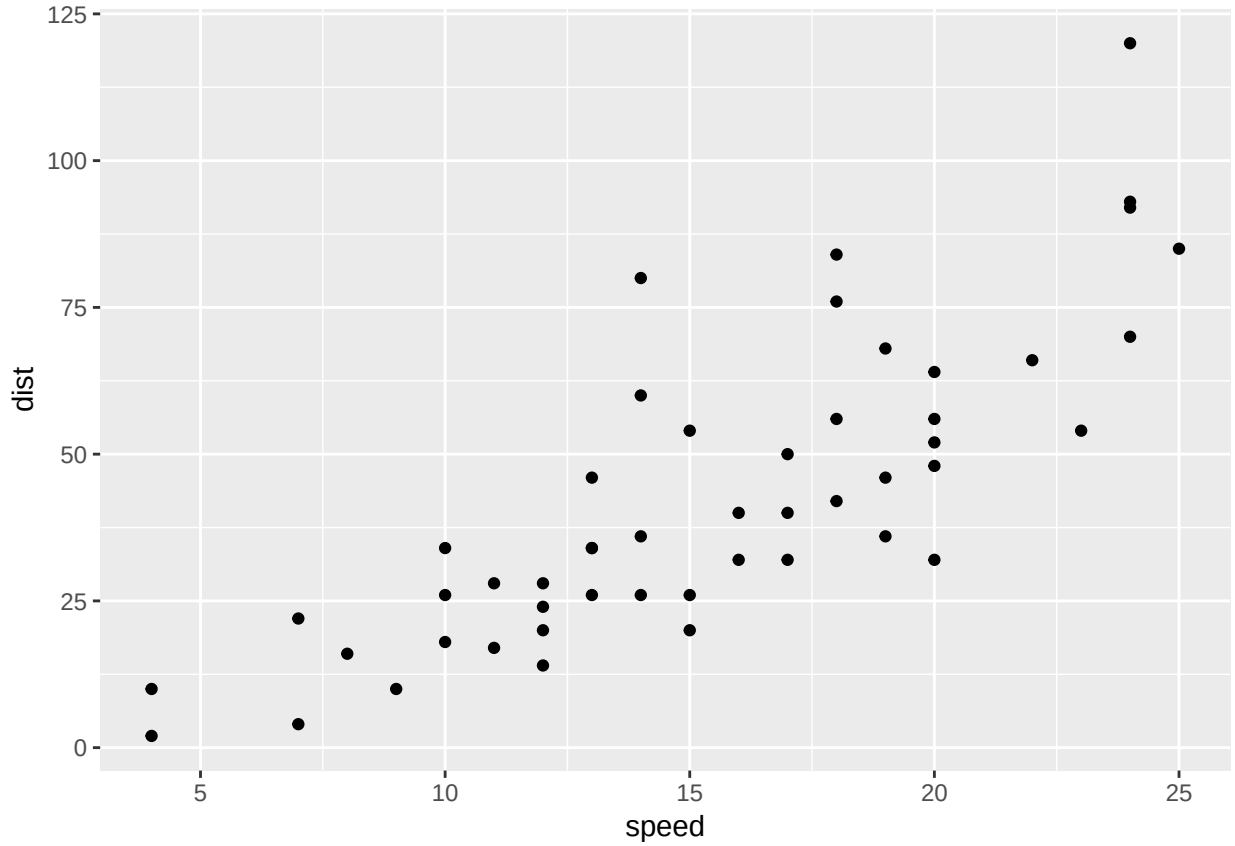
```
cars %>% qplot(speed, data=., geom = "density") + ggtitle("Density of car speed created using qplot (ggp
```



Using Geometries

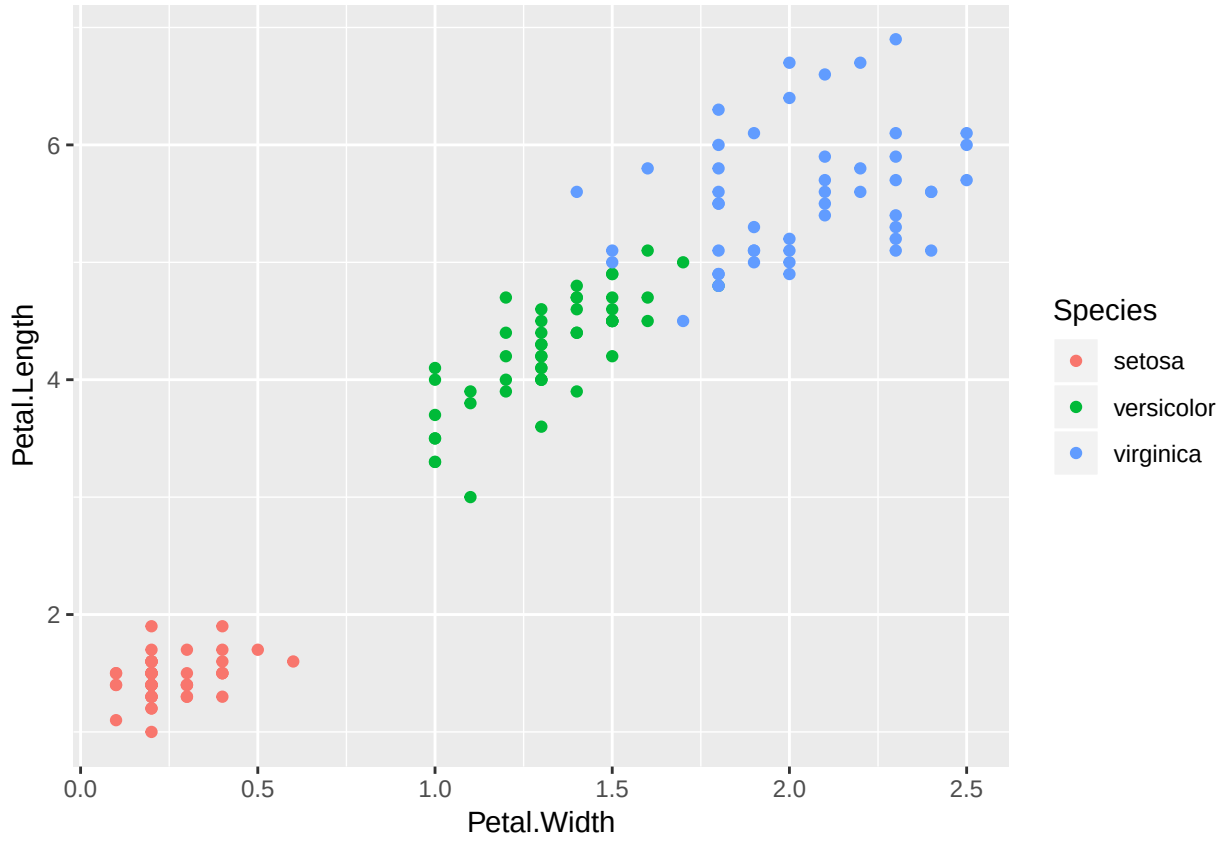
Les géomètries permettent de préciser quels sont les types de graphiques souhaités, elles permettent aussi de superposer plusieurs graphes.
Pour créer un graphe avec des géomètries, on doit créer un objet ggplot, on doit assigner le paramètre *Speed* de la dataframe à l'axe x et le paramètre *dist* à l'axe des y, pour ensuite tracer les données comme des points.

```
ggplot(cars) + geom_point(aes(x = speed, y = dist))
```

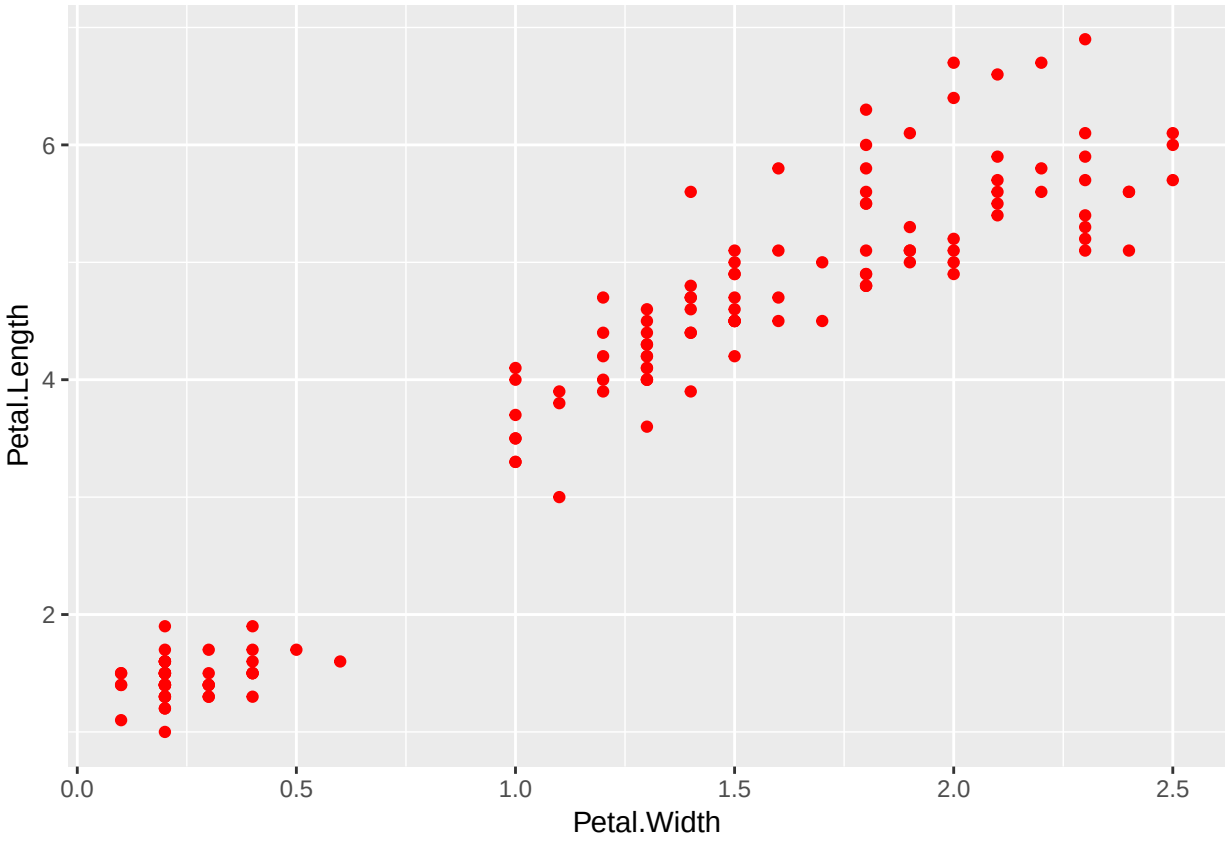


Avec les données iris, on a l'équivalent du graphe effectué avec le ***qplot***, en utilisant ***ggplot***.

```
iris %>% ggplot + geom_point(aes(x = Petal.Width, y = Petal.Length, color = Species))
```

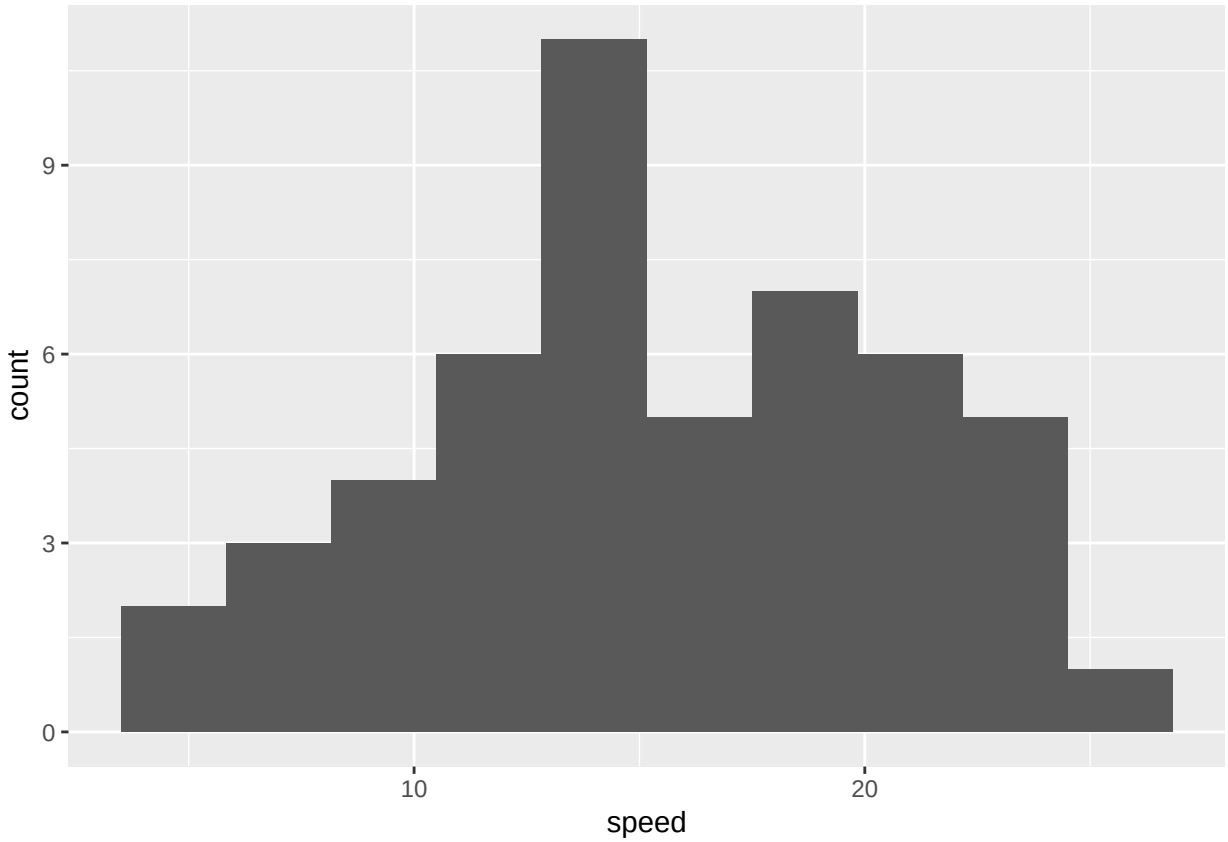



```
iris %>% ggplot +
  geom_point(aes(x = Petal.Width, y = Petal.Length), color = "red")
```

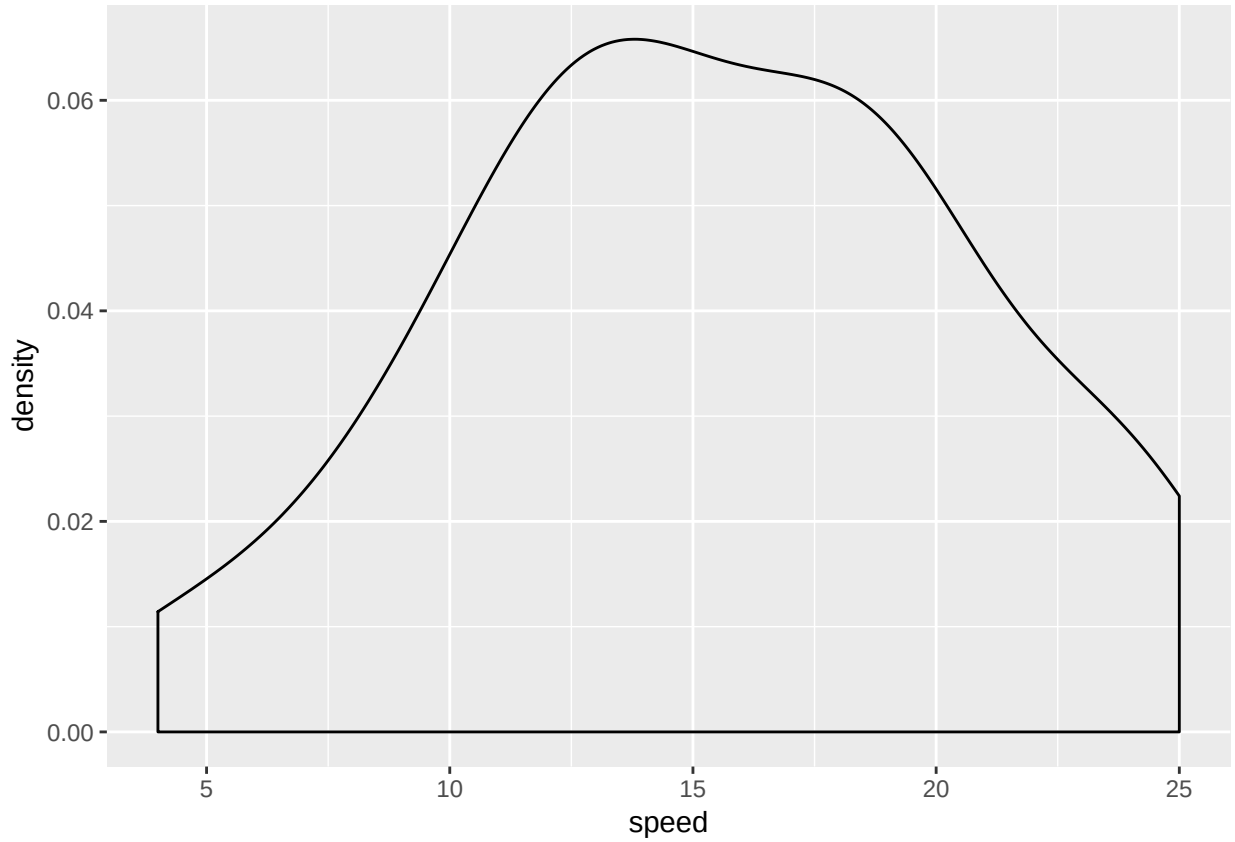


On peut construire les graphiques de densité et histogrammes aussi en utilisant ggplot2.

```
cars %>% ggplot + geom_histogram(aes(x = speed), bins = 10)
```

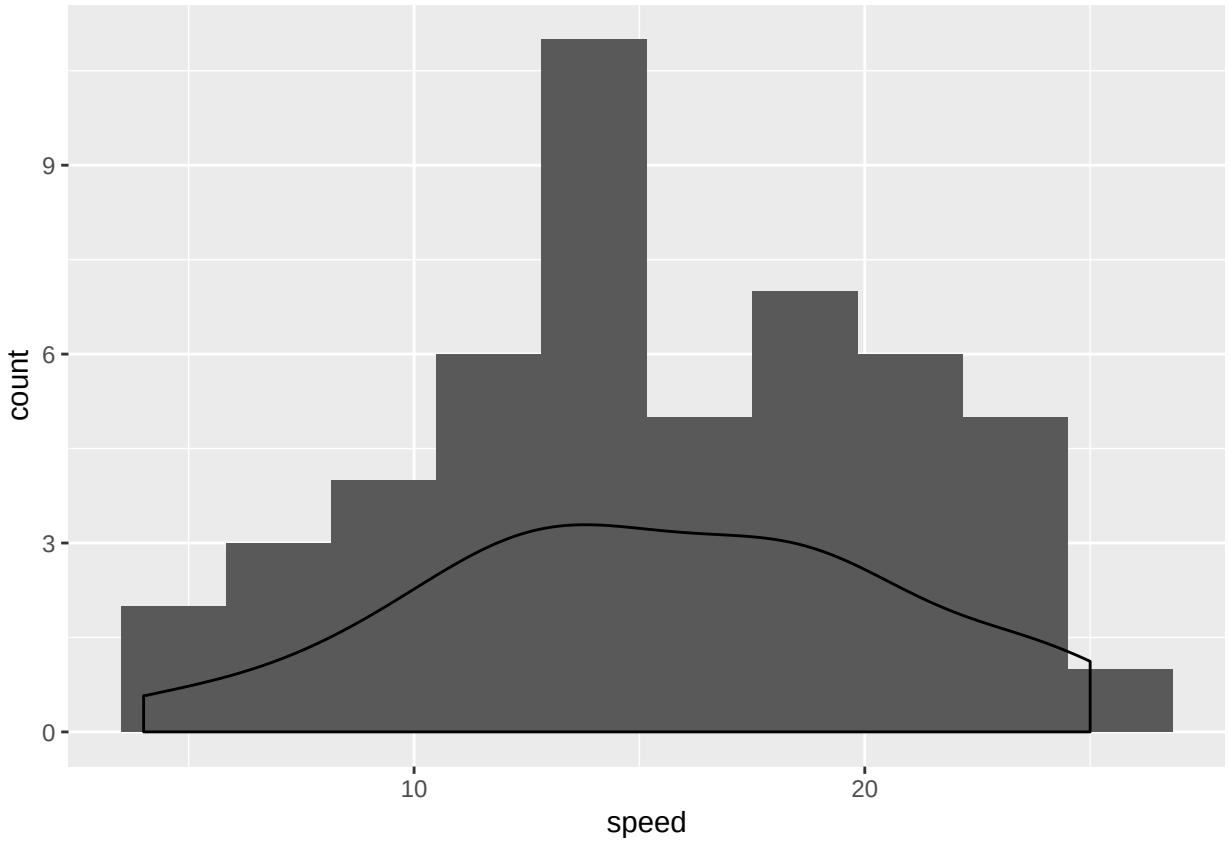



```
cars %>% ggplot + geom_density(aes(x = speed))
```



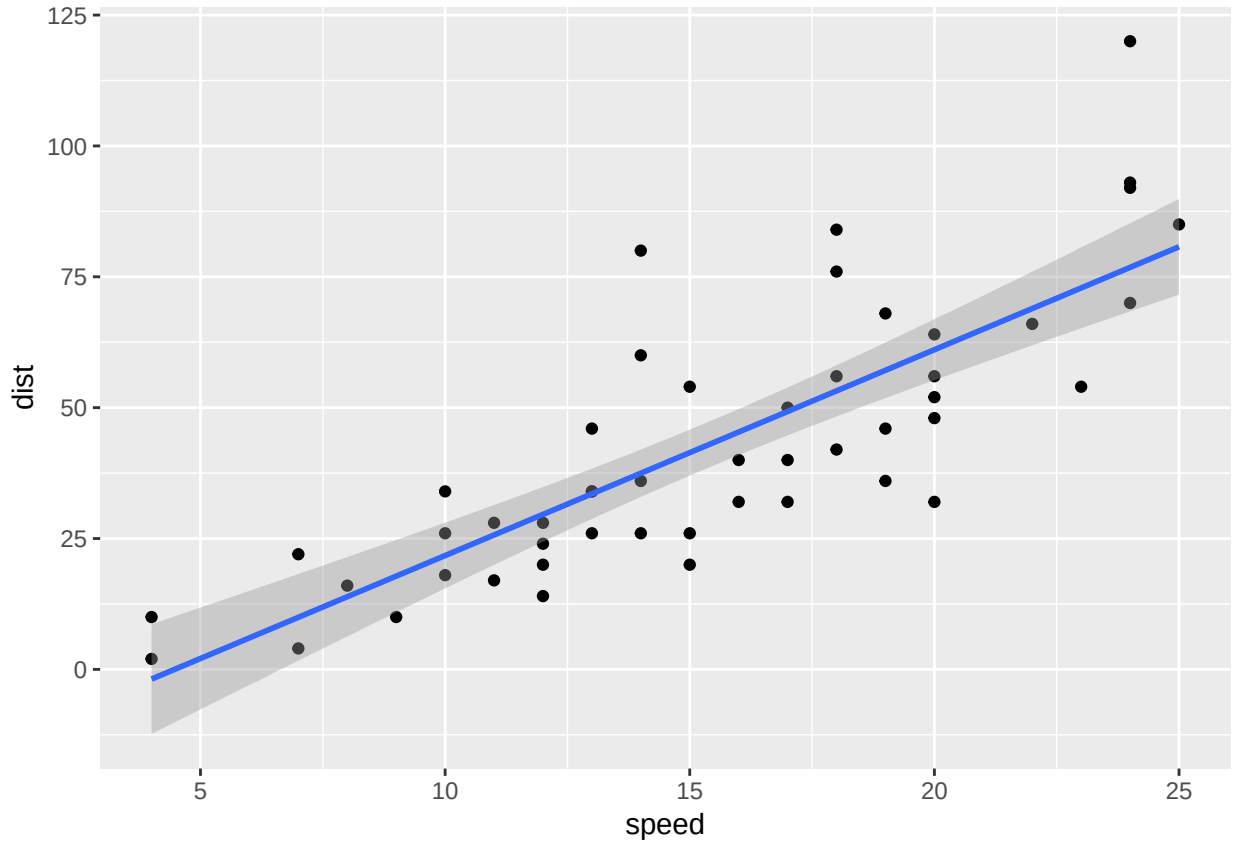
On peut aussi combiner les deux graphes en un seul graphe, en les superposant:

```
cars %>% ggplot(aes(x = speed, y = ..count..)) +  geom_histogram(bins = 10) +   geom_density()
```



Autre exemple de superposition de graphique : nuage de points et droite linéaire, pour cela on précise que ***geo_smooth = lm***, ce qui permet d'utiliser la méthode de régression linéaire:

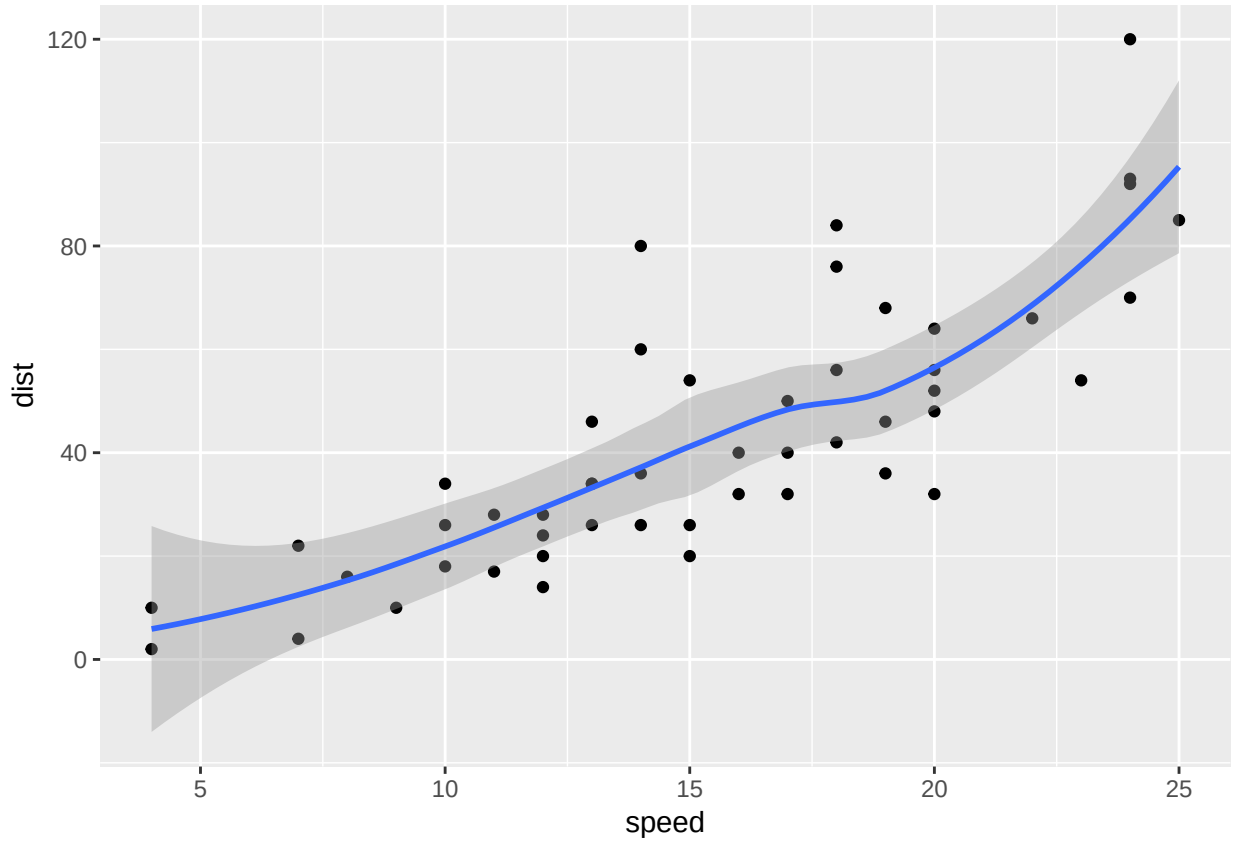
```
cars %>% ggplot(aes(x = speed, y = dist)) +  geom_point() + geom_smooth(method = "lm")
```



Si on ne précise pas ***geo_smooth = lm***, on n'utilisera pas la régression linéaire, on aura à place ***geo_smooth = loess***.
LOESS(Locally weighted Scatterplot Smoother) est une méthode de régression non paramétrique fortement connexe qui combine plusieurs modèles de régression multiple au sein d'un méta-modèle qui repose sur la méthode des k plus proches voisins:

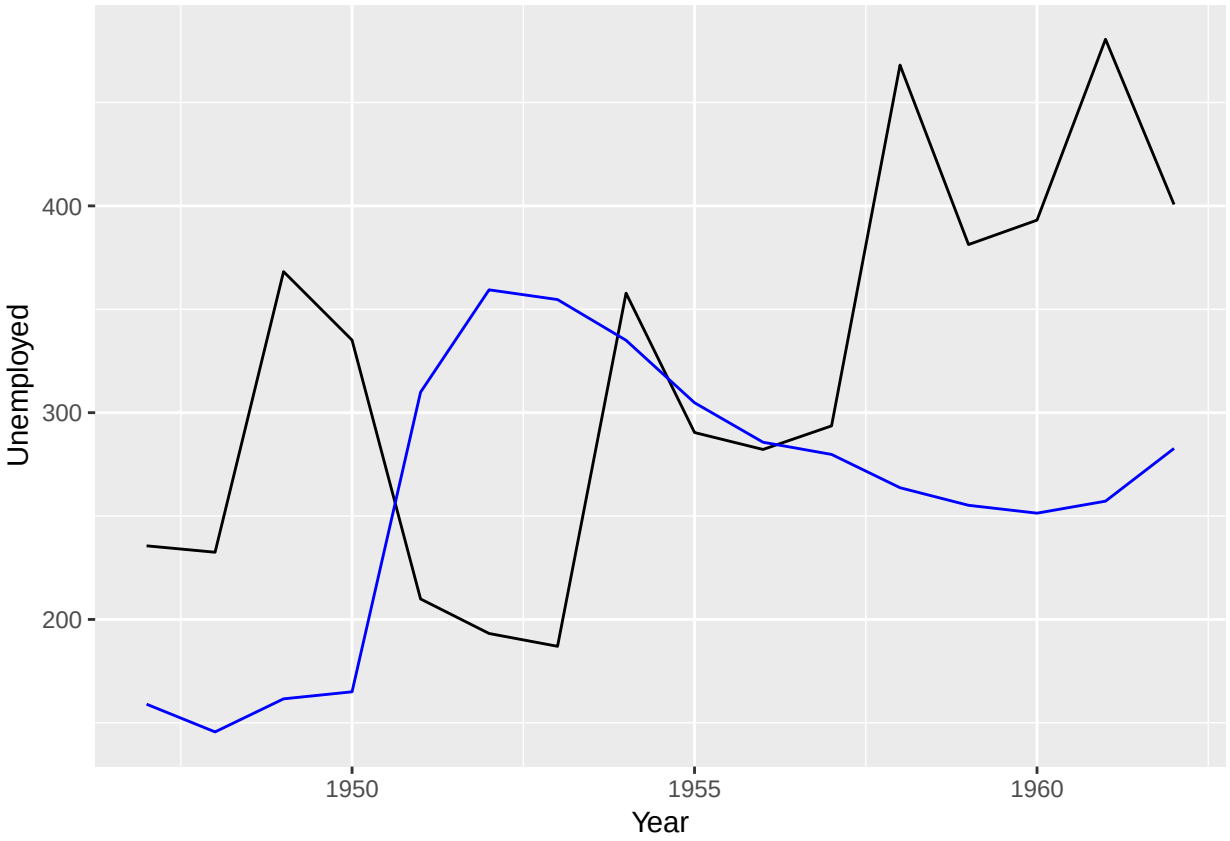
```
cars %>% ggplot(aes(x = speed, y = dist)) + geom_point() + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



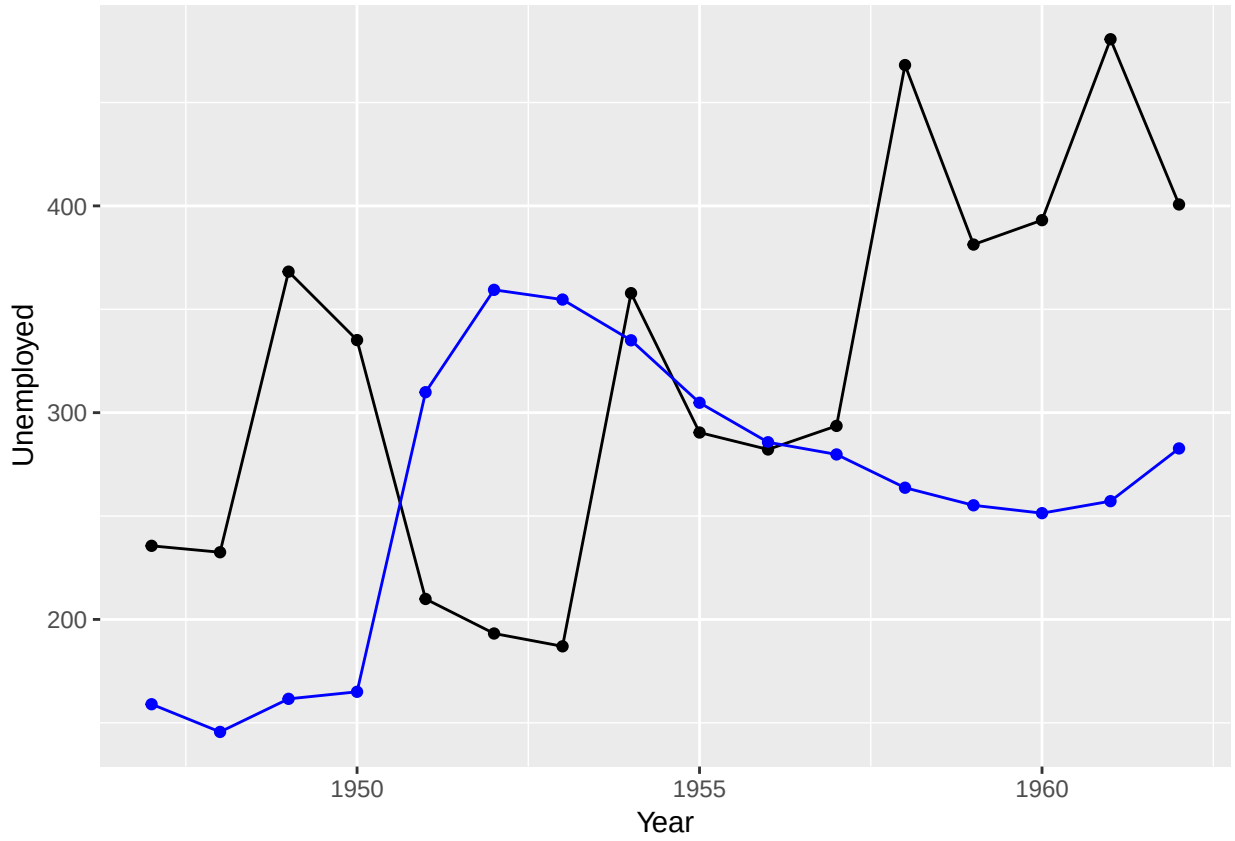
On peut aussi utiliser d'autres fonctions comme _**geo_line*_

```
longley %>% ggplot(aes(x = Year)) +   geom_line(aes(y = Unemployed)) +   geom_line(aes(y = Armed.Forces)
```

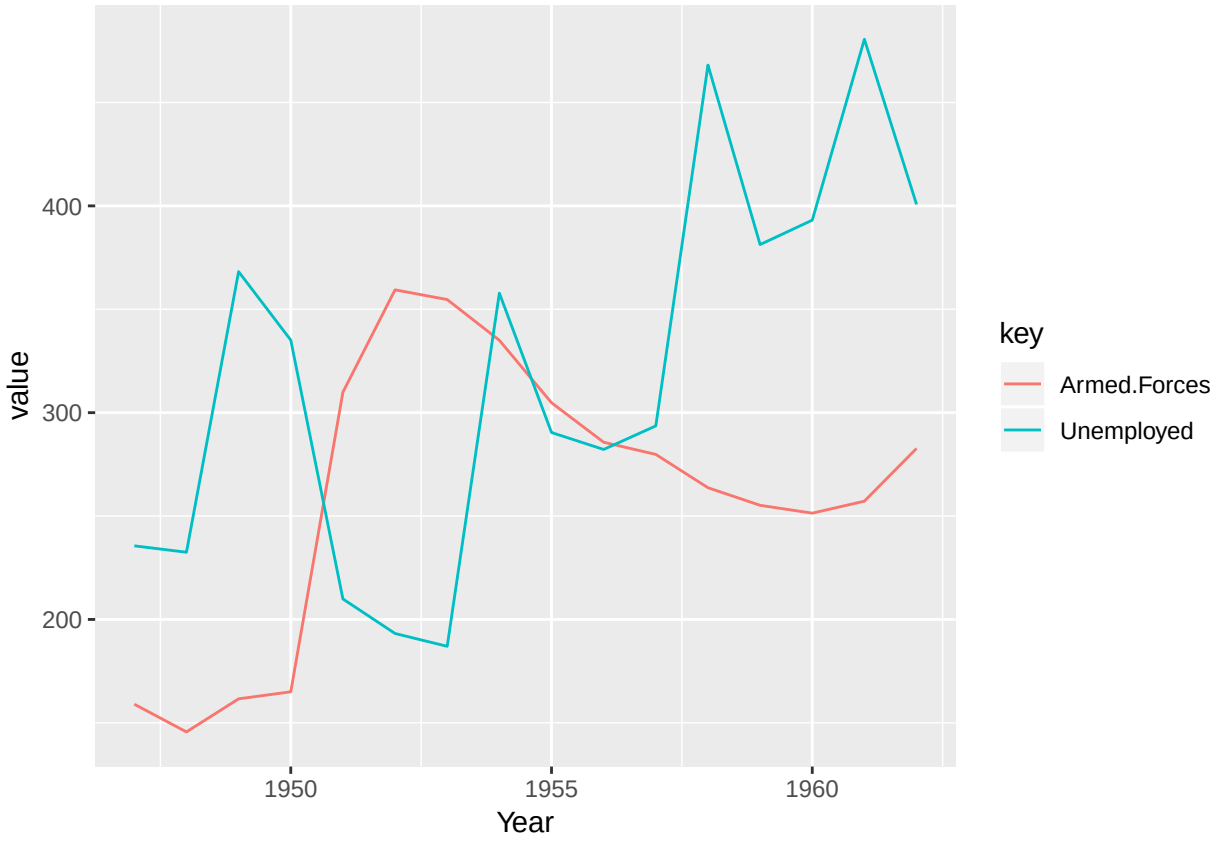


On peut aussi combiner un graphe de point avec un graphe de droite, jouer sur la couleur ou avoir des graphes séparés:

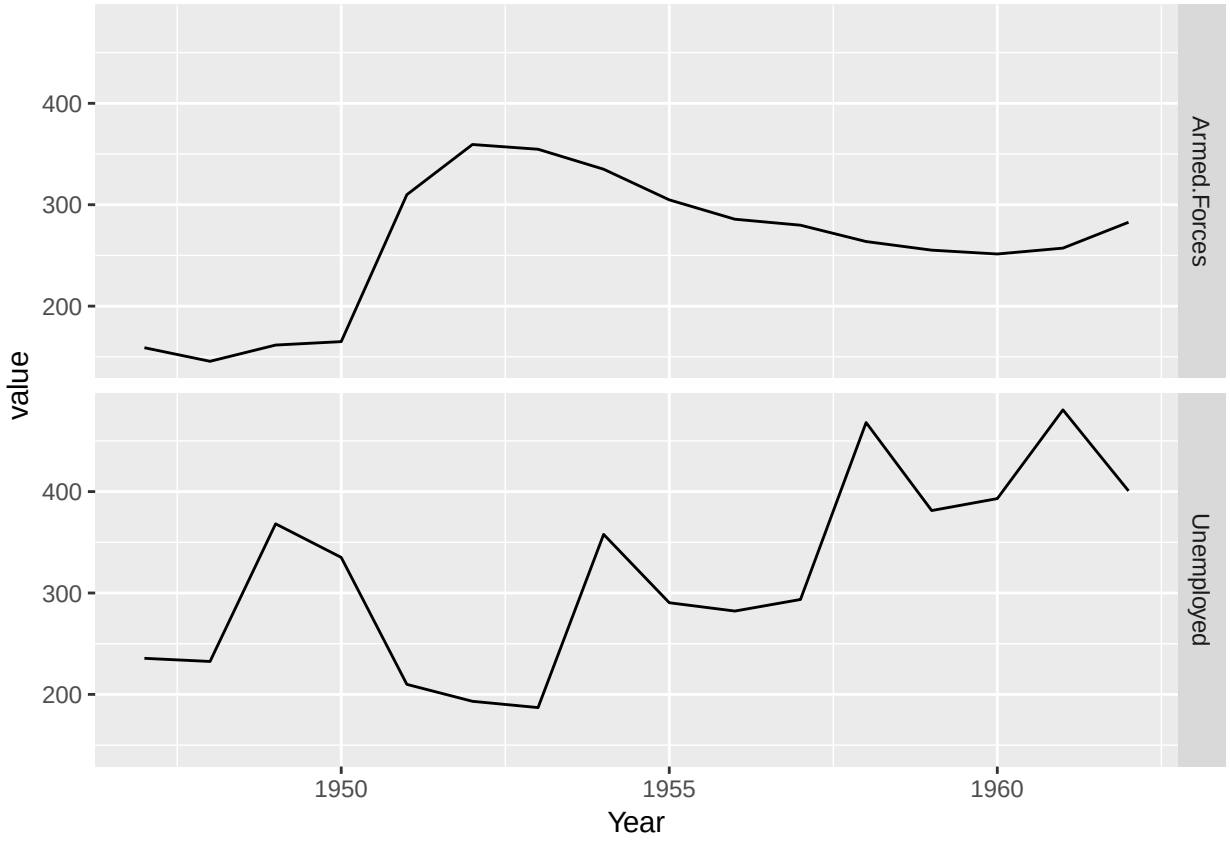
```
longley %>% ggplot(aes(x = Year)) +
  geom_point(aes(y = Unemployed)) +
  geom_point(aes(y = Armed.Forces), color = "blue") +
  geom_line(aes(y = Unemployed)) +
  geom_line(aes(y = Armed.Forces), color = "blue")
```




```
longley %>% gather(key, value, Unemployed, Armed.Forces) %>%
  ggplot(aes(x = Year, y = value, color = key)) + geom_line()
```



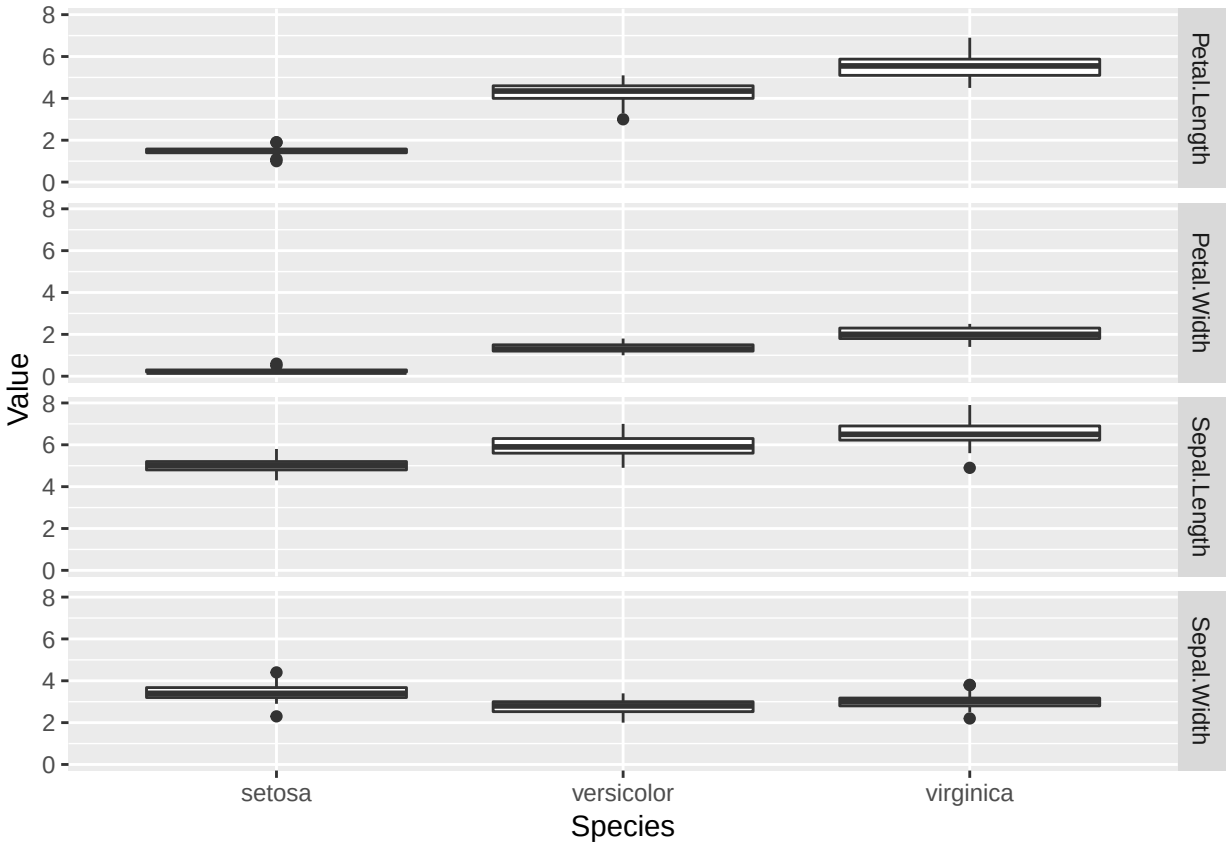

```
longley %>% gather(key, value, Unemployed, Armed.Forces) %>%
  ggplot(aes(x = Year, y = value)) + geom_line() +
  facet_grid(key ~ .)
```



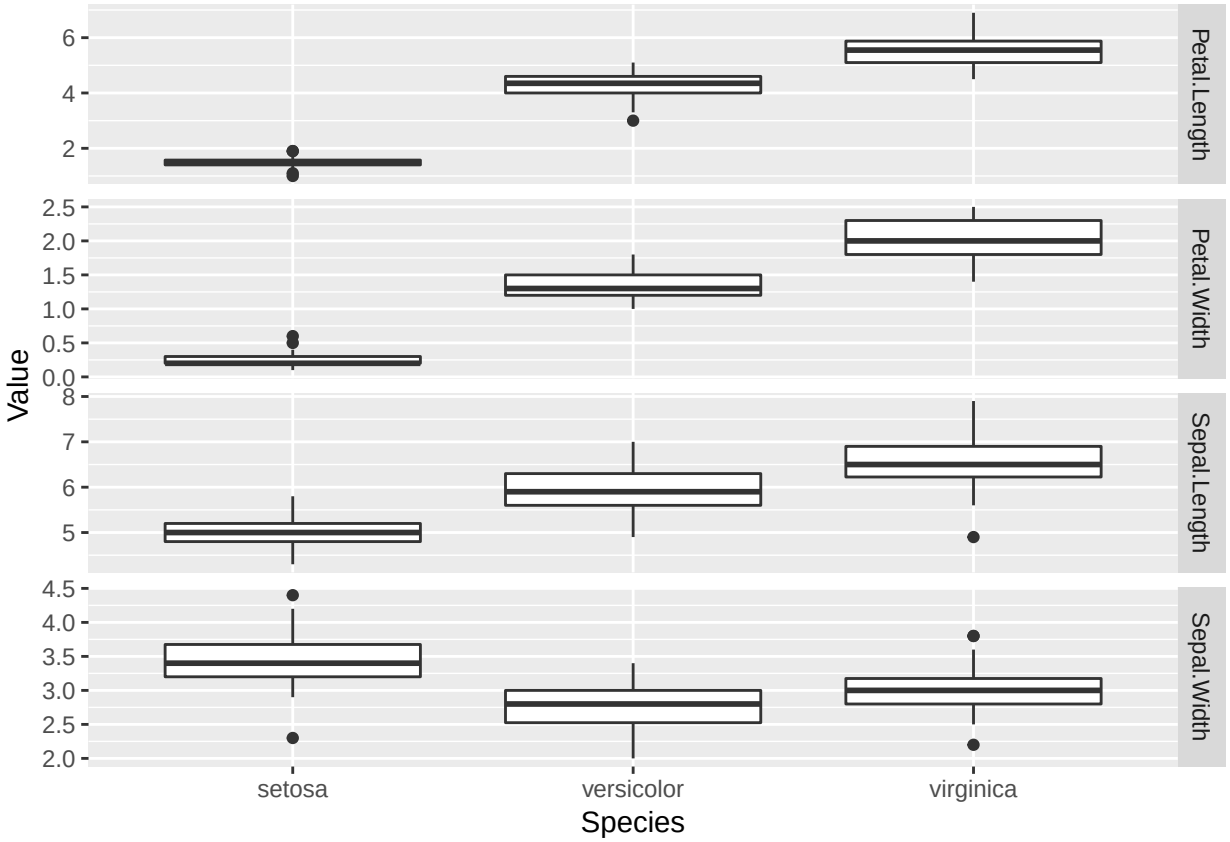
Facets

Les Facets sont des sous-graphes pour mettre en évidence des sous-ensembles de données:

```
iris %>% gather(Measurement, Value, -Species) %>%  ggplot(aes(x = Species, y = Value)) +  geom_boxplot()
  facet_grid(Measurement ~ .)
```

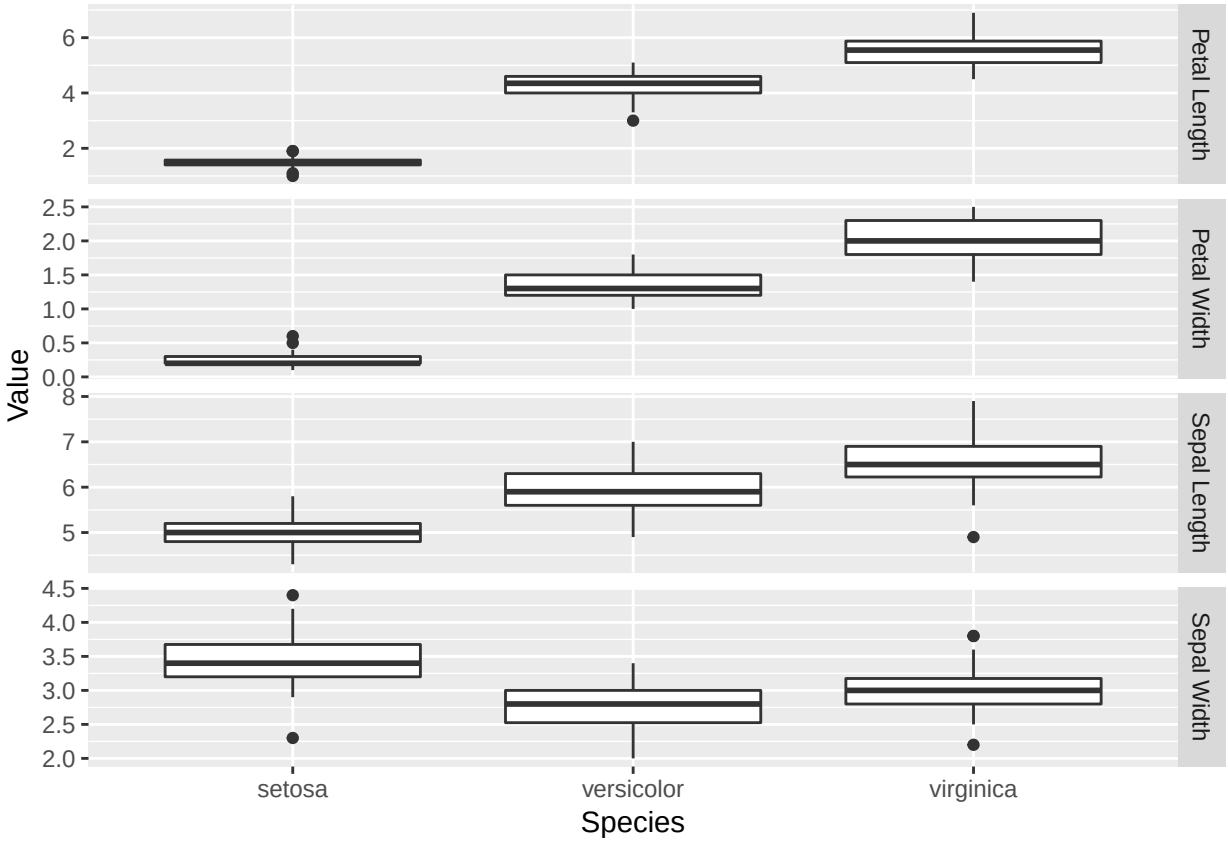



```
iris %>% gather(Measurement, Value, -Species) %>%  ggplot(aes(x = Species, y = Value)) +  geom_boxplot()
  facet_grid(Measurement ~ ., scale = "free_y")
```



On peut formatter le nom des étiquettes des sous-graphes, en utilisant la fonction ***labeller***, exemple avec le jeu de données ***Iris***, on veut effacer les points dans le nom des variables ***Petal.length***, ***Petal.Width***:

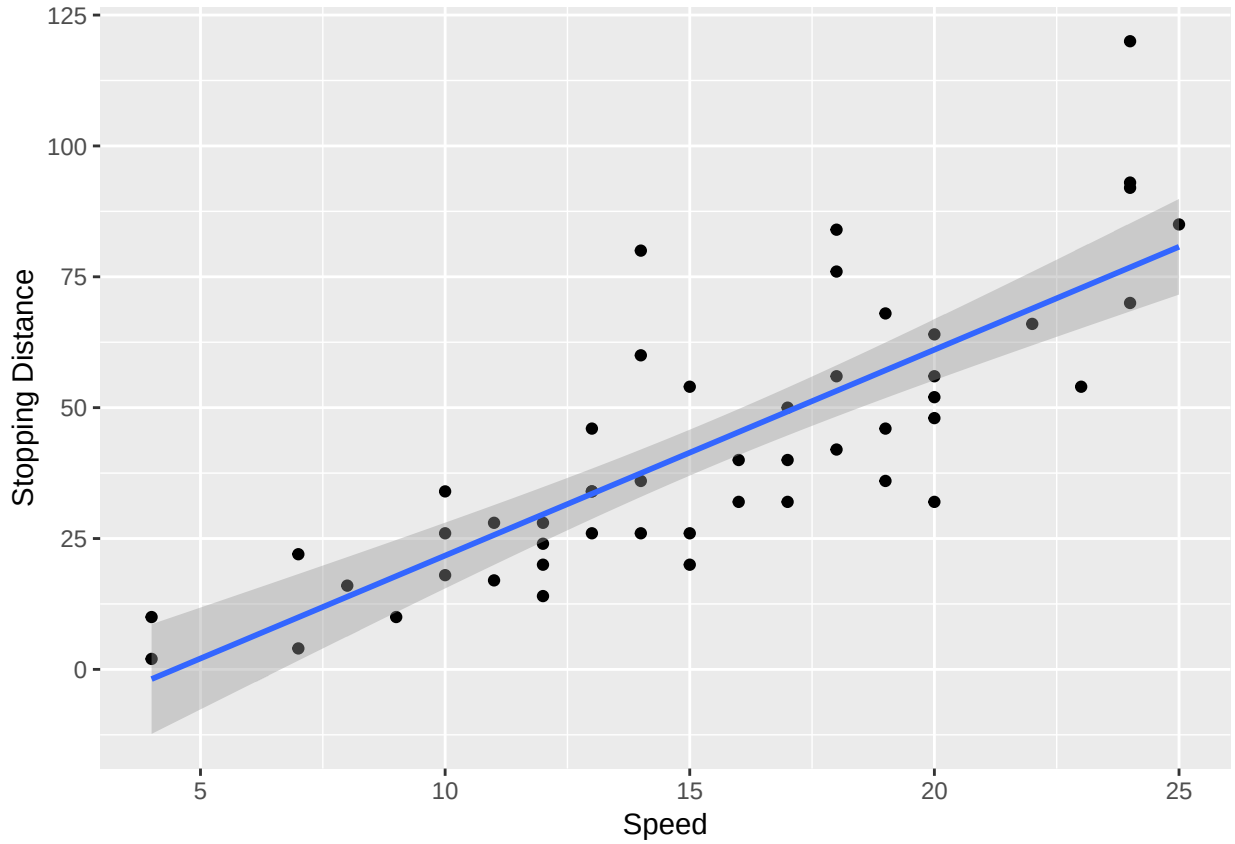
```
label_map <- c(Petal.Width = "Petal Width",
               Petal.Length = "Petal Length",
               Sepal.Width = "Sepal Width",
               Sepal.Length = "Sepal Length")
iris %>% gather(Measurement, Value, -Species) %>%
  ggplot(aes(x = Species, y = Value)) +
  geom_boxplot() +
  facet_grid(Measurement ~ ., scale = "free_y",
             labeller = labeller(Measurement = label_map))
```



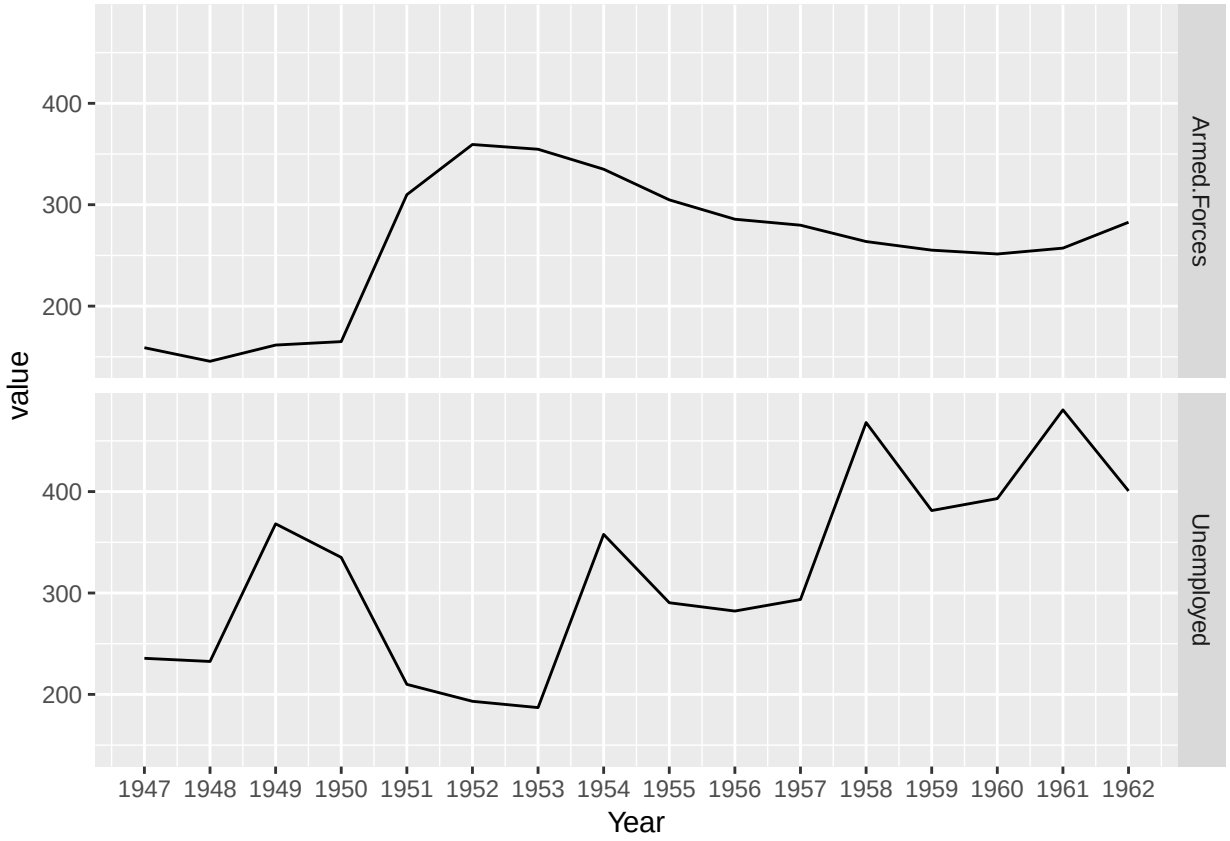
Scaling(dimensionnement)

Pour spécifier les titres des axes, on utilise les fonctions ***scale_x_continuous*** et ***scale_y_continuous***, on aurait pu utiliser aussi ***xlab*** et ***ylab***

```
cars %>% ggplot(aes(x = speed, y = dist)) +
  geom_point() + geom_smooth(method = "lm") +
  scale_x_continuous("Speed") +
  scale_y_continuous("Stopping Distance")
```

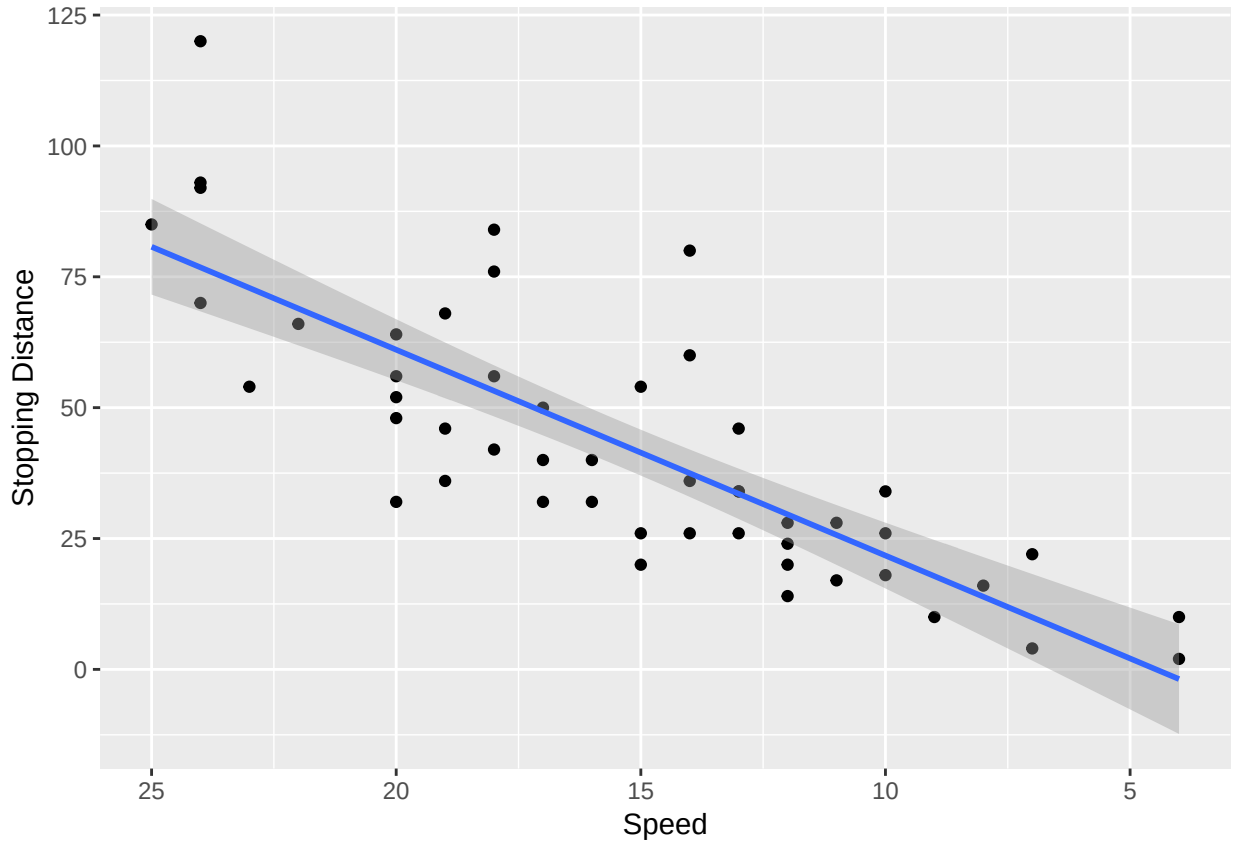



```
longley %>% gather(key, value, Unemployed, Armed.Forces) %>%
  ggplot(aes(x = Year, y = value)) + geom_line() +
  scale_x_continuous(breaks = 1947:1962) +
  facet_grid(key ~ .)
```



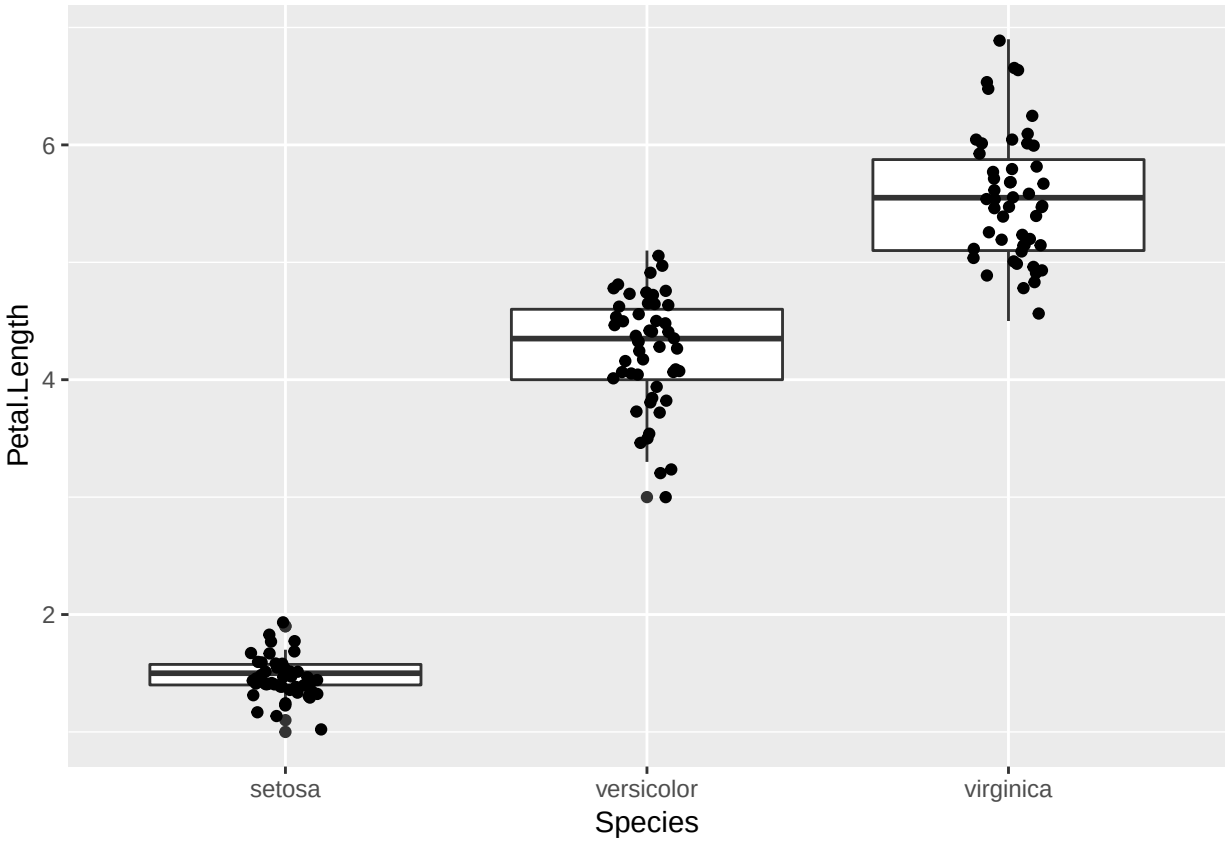
On peut aussi afficher le graphe de façon inverser, par ordre décroissant de graduation:

```
cars %>% ggplot(aes(x = speed, y = dist)) +
  geom_point() + geom_smooth(method = "lm") +
  scale_x_reverse("Speed") +
  scale_y_continuous("Stopping Distance")
```



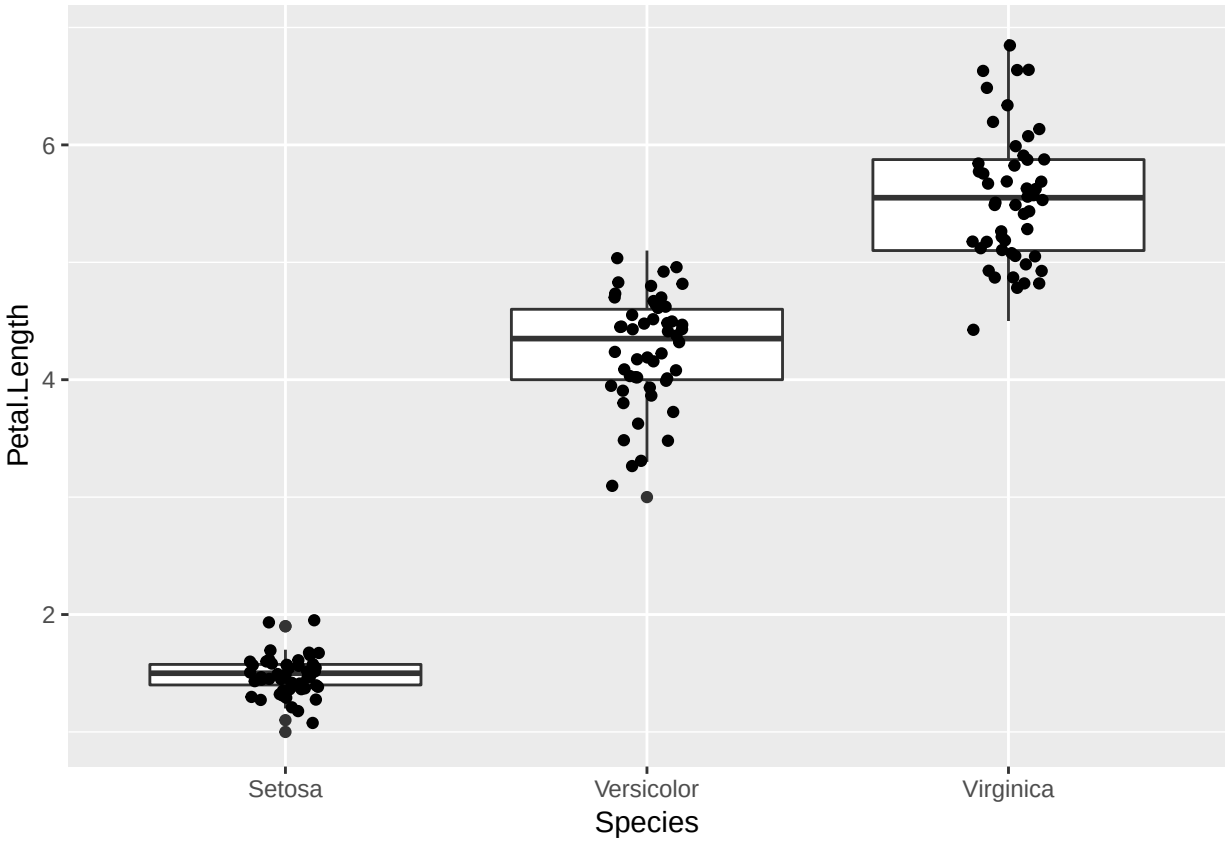
On peut aussi ajouter des nuages de points aux boxplot comme suit:

```
iris %>% ggplot(aes(x = Species, y = Petal.Length)) +
  geom_boxplot() + geom_jitter(width = 0.1, height = 0.1)
```



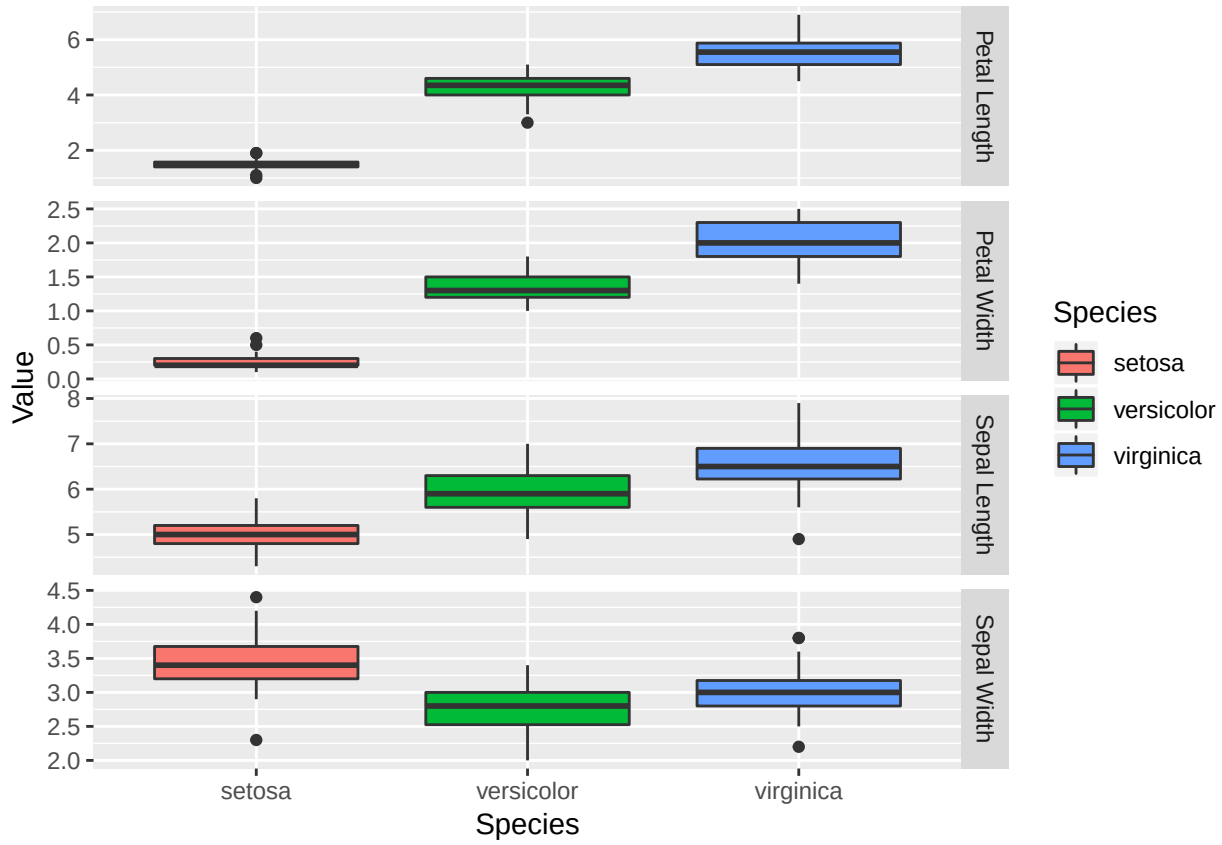
On peut modifier les labels sur les axes x:

```
iris %>% ggplot(aes(x = Species, y = Petal.Length)) +
  geom_boxplot() + geom_jitter(width = 0.1, height = 0.1) +
  scale_x_discrete(labels = c("setosa" = "Setosa",
                              "versicolor" = "Versicolor",
                              "virginica" = "Virginica"))
```



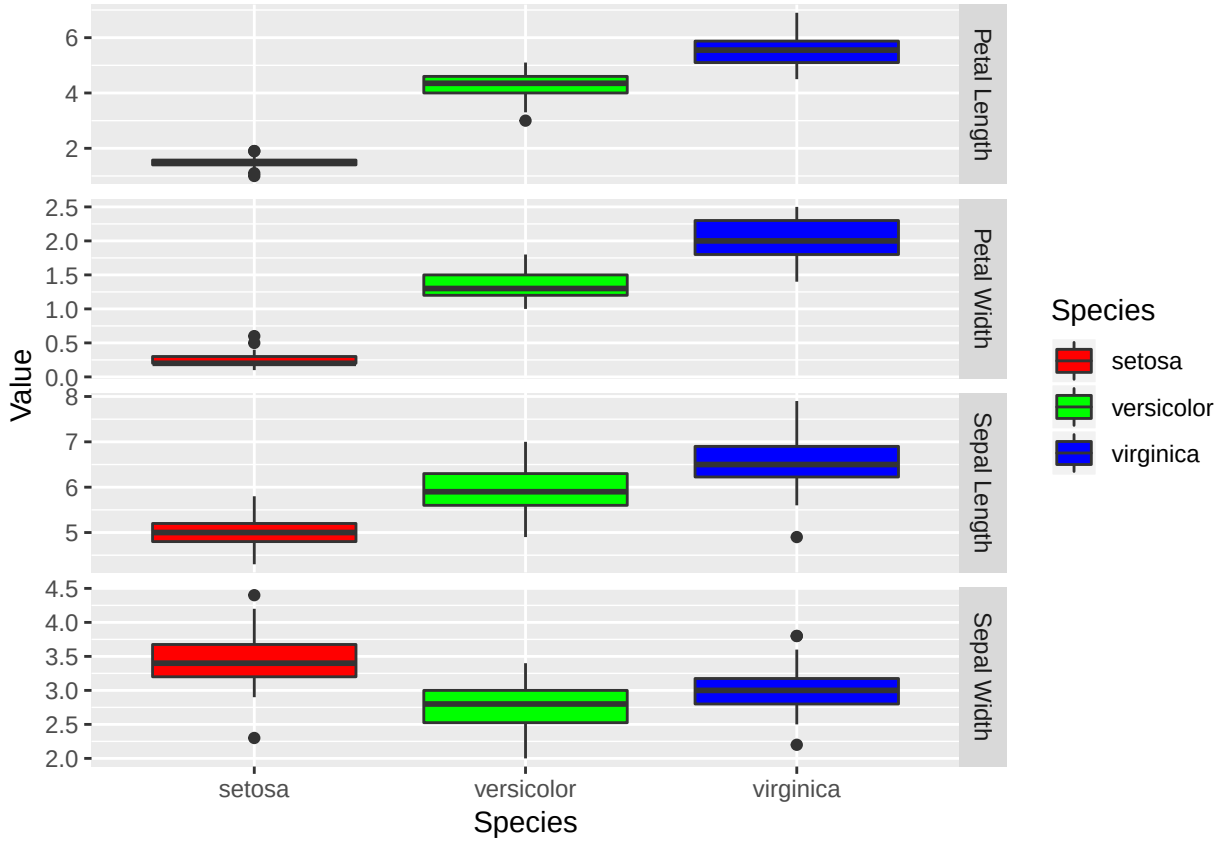
On peut aussi colorier les diagrammes à moustache en fonction des étiquettes de la variable ***Species*** :

```
label_map <- c(Petal.Width = "Petal Width",
               Petal.Length = "Petal Length",
               Sepal.Width = "Sepal Width",
               Sepal.Length = "Sepal Length")
iris %>% gather(Measurement, Value, -Species) %>%
  ggplot(aes(x = Species, y = Value, fill = Species)) +
  geom_boxplot() +
  facet_grid(Measurement ~ ., scale = "free_y",
               labeller = labeller(Measurement = label_map))
```



Ou jouer sur le constraste du graphe en fonction des étiquettes de la variable ***Species*** :

```
iris %>% gather(Measurement, Value, -Species) %>%
  ggplot(aes(x = Species, y = Value, fill = Species)) +
  geom_boxplot() +
  scale_fill_manual(values = c("red", "green", "blue")) +
  facet_grid(Measurement ~ ., scale = "free_y",
             labeller = labeller(Measurement = label_map))
```

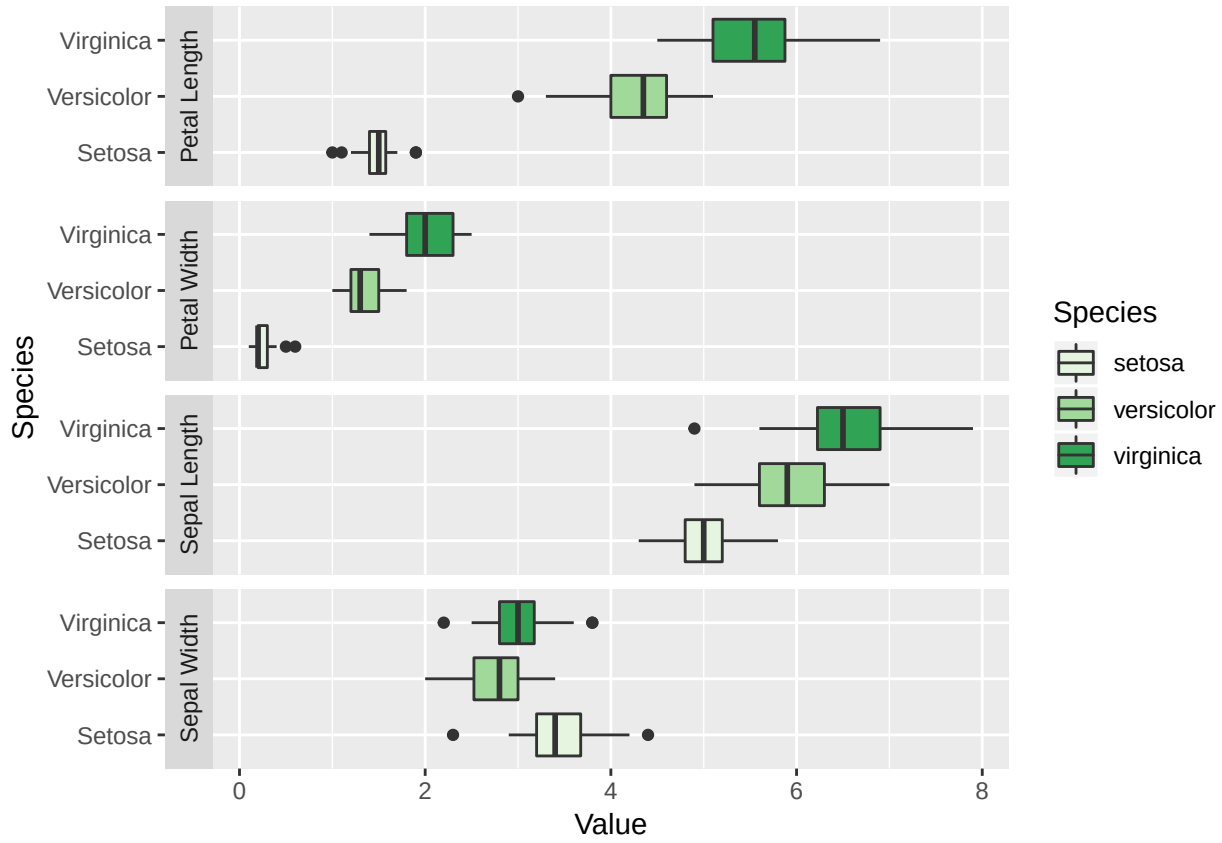


Themes and Other Graphics Transformations

Dans cette section, nous allons voir comment changer l'affichage des graphiques

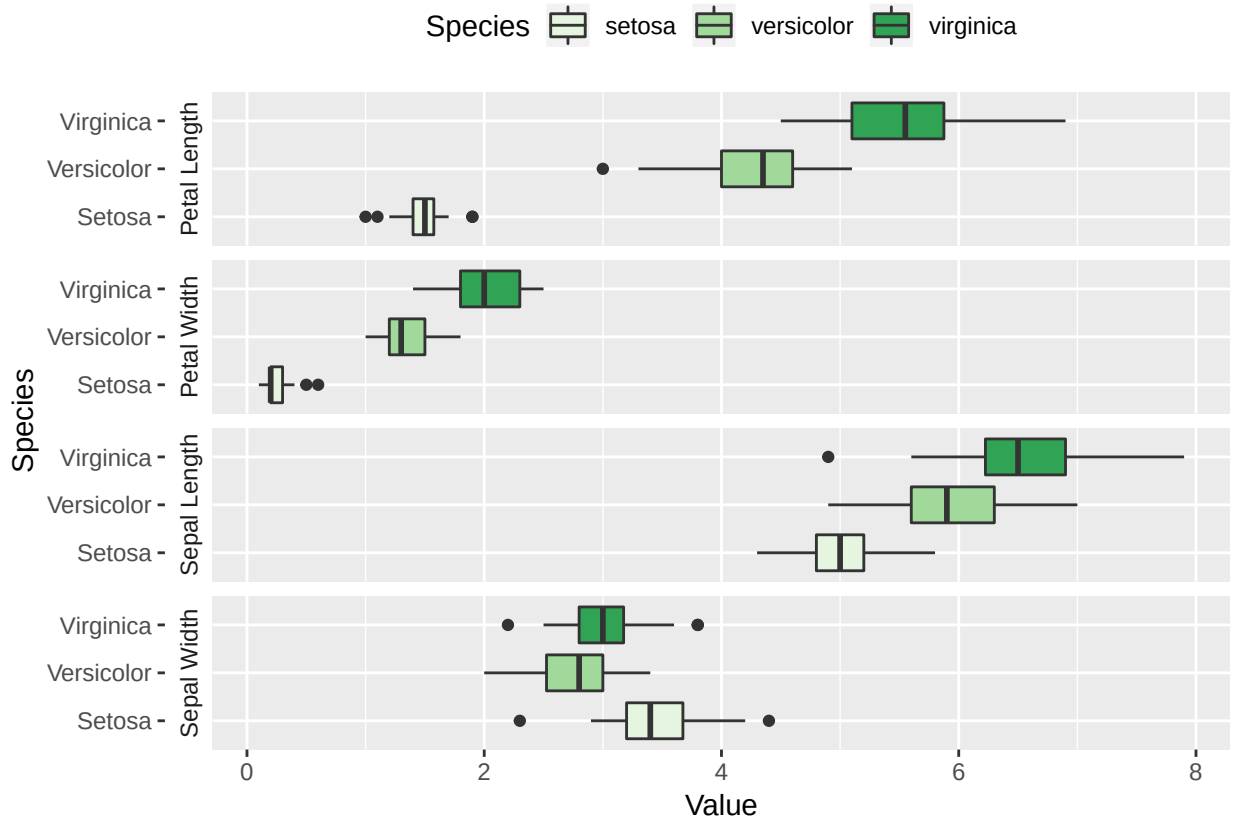
Voici l'affichage sans utiliser de thème:

```
iris %>% gather(Measurement, Value, -Species) %>%
  ggplot(aes(x = Species, y = Value, fill = Species)) +
  geom_boxplot() +
  scale_x_discrete(labels = c("setosa" = "Setosa",
                              "versicolor" = "Versicolor",
                              "virginica" = "Virginica")) +
  scale_fill_brewer(palette = "Greens") +
  facet_grid(Measurement ~ ., switch = "y",
             labeller = labeller(Measurement = label_map)) +
  coord_flip()
```

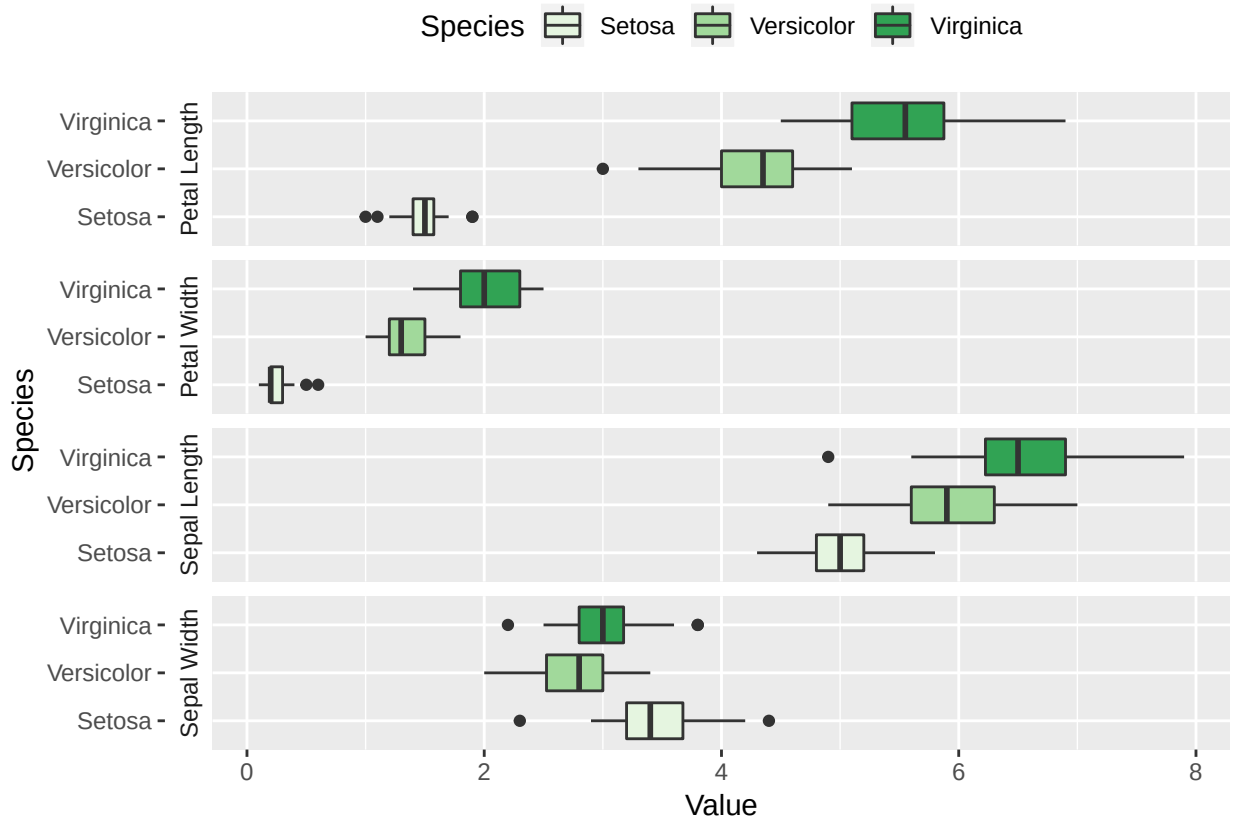


Maintenant on affiche le même graphe en utilisant des thèmes, ce qui permet de positionner la légende, de choisir le fond

```
iris %>% gather(Measurement, Value, -Species) %>%
  ggplot(aes(x = Species, y = Value, fill = Species)) +
  geom_boxplot() +
  scale_x_discrete(labels = c("setosa" = "Setosa",
                              "versicolor" = "Versicolor",
                              "virginica" = "Virginica")) +
  scale_fill_brewer(palette = "Greens") +
  facet_grid(Measurement ~ ., switch = "y",
             labeller = labeller(Measurement = label_map)) +
  coord_flip() +
  theme(strip.background = element_blank()) +
  theme(legend.position="top")
```




```
label_map <- c(Petal.Width = "Petal Width",
               Petal.Length = "Petal Length",
               Sepal.Width = "Sepal Width",
               Sepal.Length = "Sepal Length")
species_map <- c(setosa = "Setosa",
                 versicolor = "Versicolor",
                 virginica = "Virginica")
iris %>% gather(Measurement, Value, -Species) %>%
  ggplot(aes(x = Species, y = Value, fill = Species)) +
  geom_boxplot() +
  scale_x_discrete(labels = species_map) +
  scale_fill_brewer(palette = "Greens", labels = species_map) +
  facet_grid(Measurement ~ ., switch = "y",
             labeller = labeller(Measurement = label_map)) +
  coord_flip() +
  theme(strip.background = element_blank()) +
  theme(legend.position="top")
```



Figures with Multiple Plots

On peut afficher deux graphes en 1 seul graphe en utilisant le package ***gridExtra***, avec la fonction ***grid.arrange()*** :

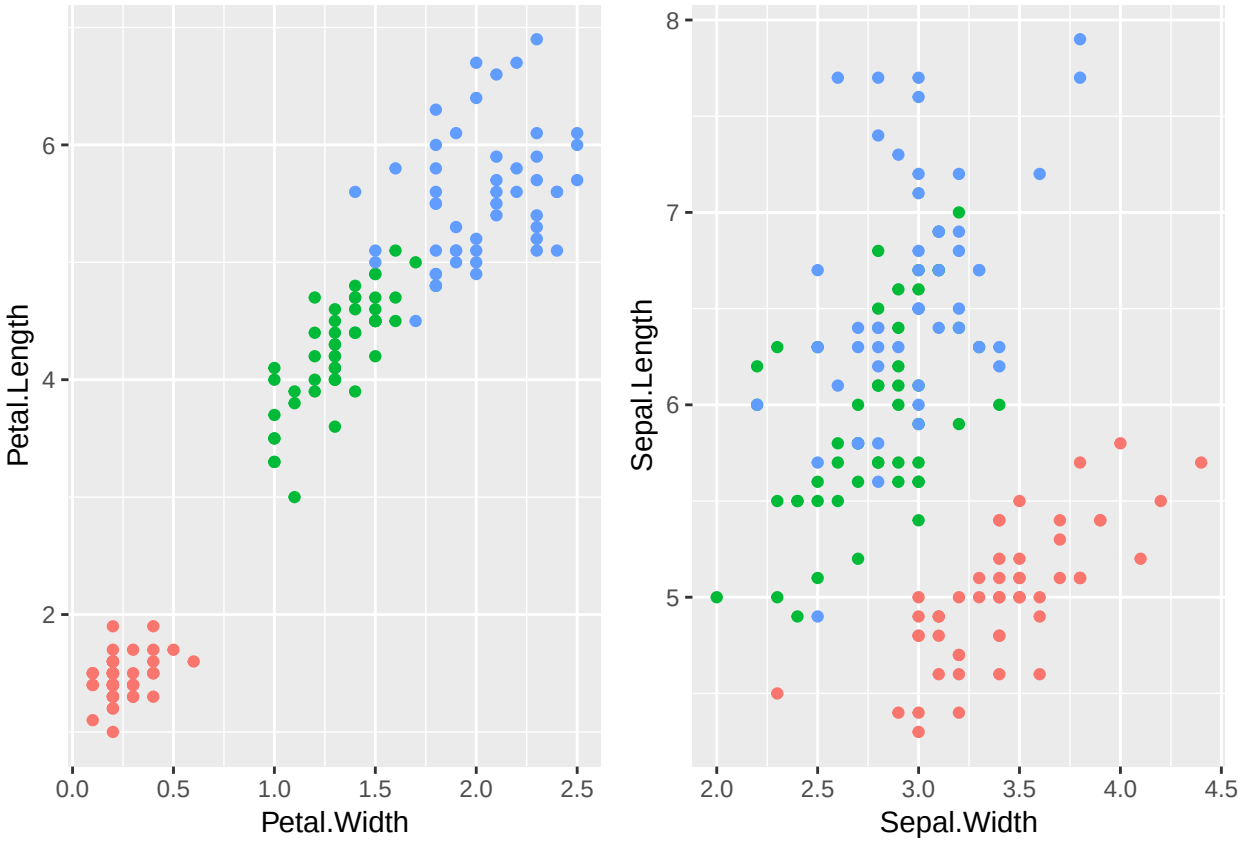
```
petal <- iris %>% ggplot() +
  geom_point(aes(x = Petal.Width, y = Petal.Length,
                 color = Species)) +
  theme(legend.position="none")
sepal <- iris %>% ggplot() +
  geom_point(aes(x = Sepal.Width, y = Sepal.Length,
                 color = Species)) +
  theme(legend.position="none")

library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
grid.arrange(petal, sepal, ncol = 2)
```



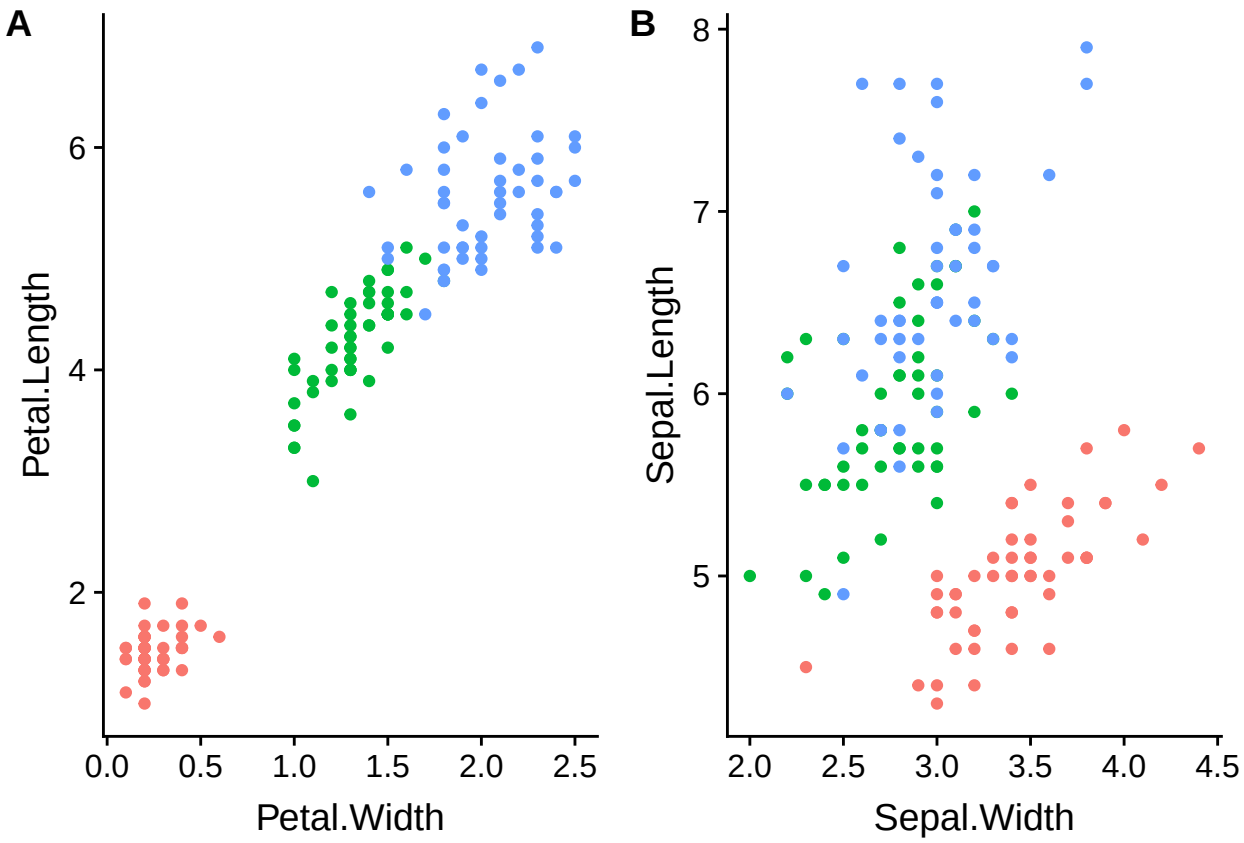
Si on ne veut pas du thème, en utilisant le package ***cowplot*** on peut spécifier le thème :

```
library(cowplot)
```

```
##
## Attaching package: 'cowplot'
```

```
## The following object is masked from 'package:ggplot2':
##
##     ggsave
```

```
plot_grid(petal, sepal, labels = c("A", "B"))
```



Working with Large Datasets

Pour extraire des échantillons d'un jeu de données, on a le package ***dplyr***.
Pour sélectionner n ligne d'un jeu de données afin de construire un échantillon, on peut faire comme ci-dessous :

```
library(dplyr)
iris %>% sample_n(size = 5)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 109          6.7         2.5          5.8         1.8  virginica
## 69           6.2         2.2          4.5         1.5 versicolor
## 41           5.0         3.5          1.3         0.3     setosa
## 107          4.9         2.5          4.5         1.7  virginica
## 137          6.3         3.4          5.6         2.4  virginica
```

Pour sélectionner un sous jeu de données, on peut utiliser la fonction ***sample_frac***, l'attribut size permet de spécifier la fraction que les lignes doivent vérifier, exemple si $size = 0.2$ et que le jeu de données contient 150 lignes, alors seront sélectionnées les lignes telles que $n/150 = 0.2$

```
iris %>% sample_frac(size = 0.01)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 57          6.3         3.3          4.7         1.6 versicolor
## 64          6.1         2.9          4.7         1.4 versicolor
```

On peut mesurer aussi la consommation de RAM, avec le package ***pryr*** :

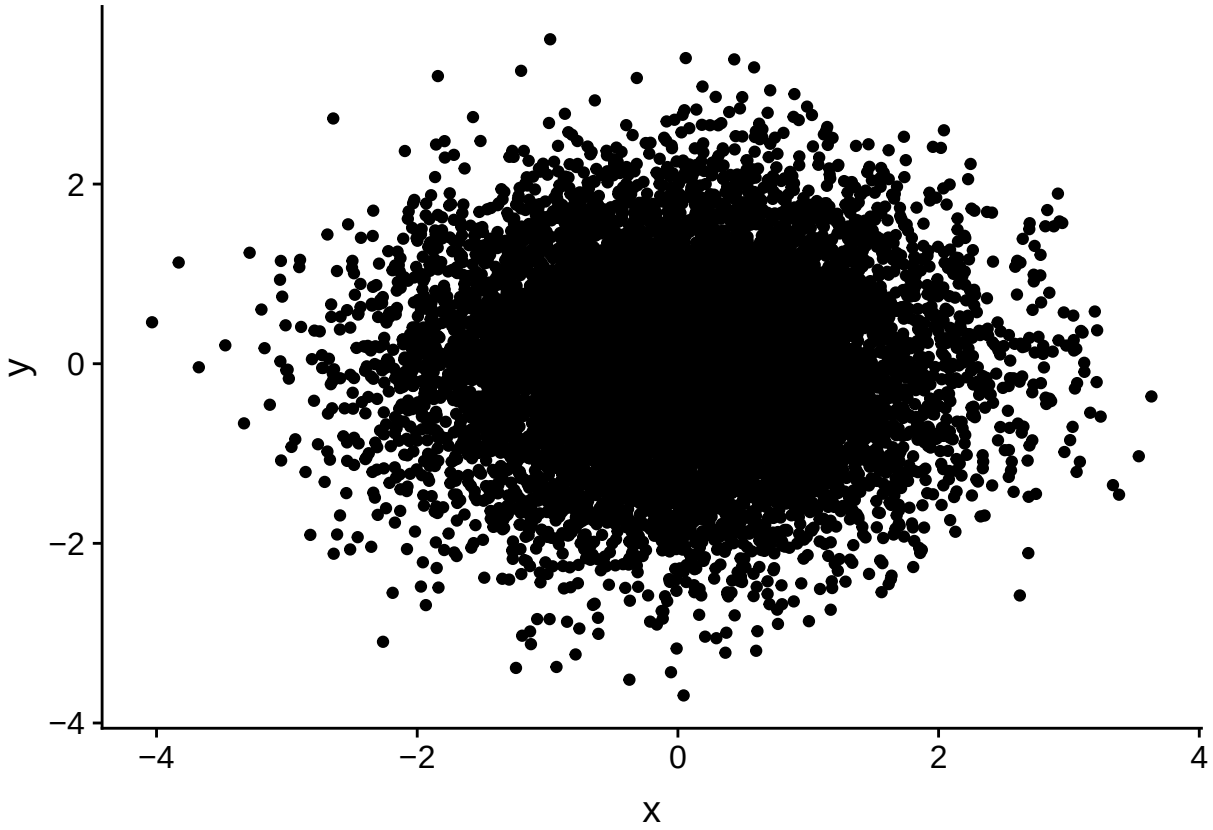
```
library(pryr)
mem_change(x <- rnorm(10000))
```

```
## 78.9 kB
```

Too Large to Plot

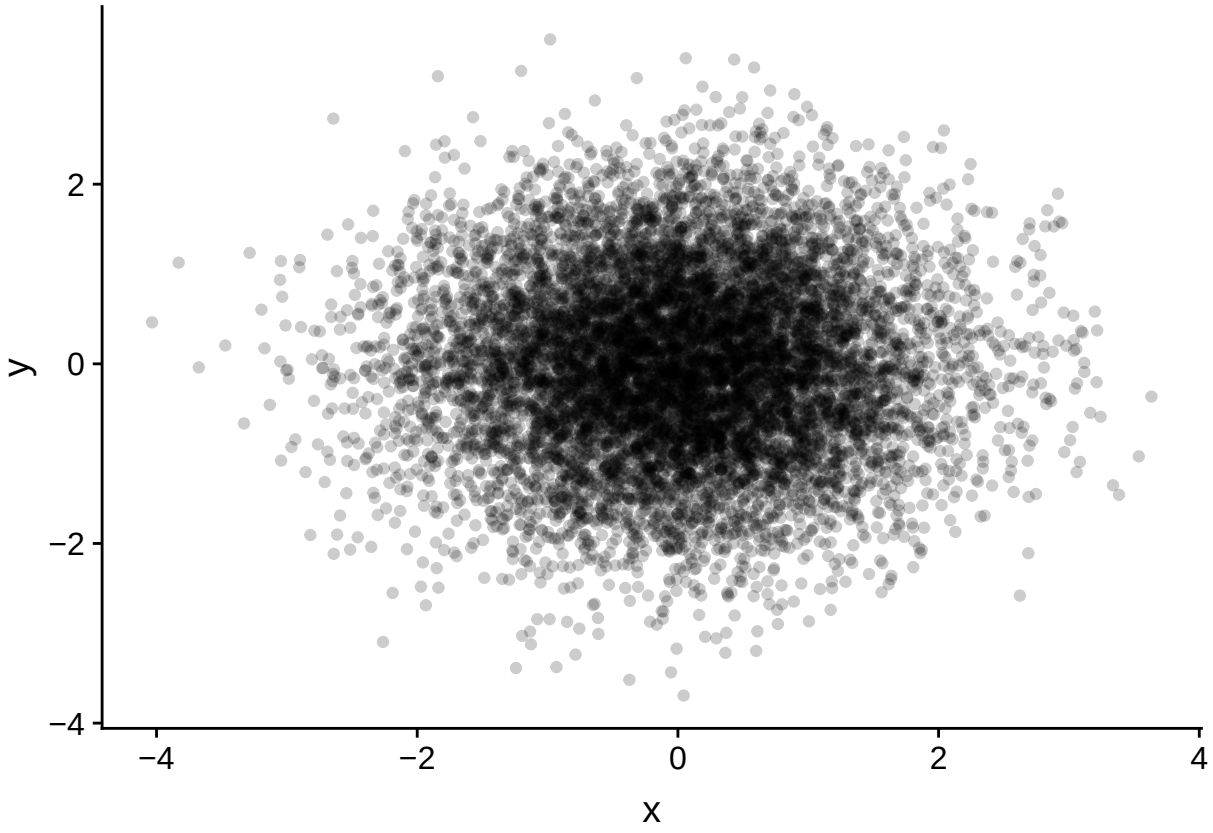
En utilisant l'affichage standard, on obtient ceci:

```
d <- data.frame(x = rnorm(10000), y = rnorm(10000))
d %>% ggplot(aes(x = x, y = y)) +
  geom_point()
```



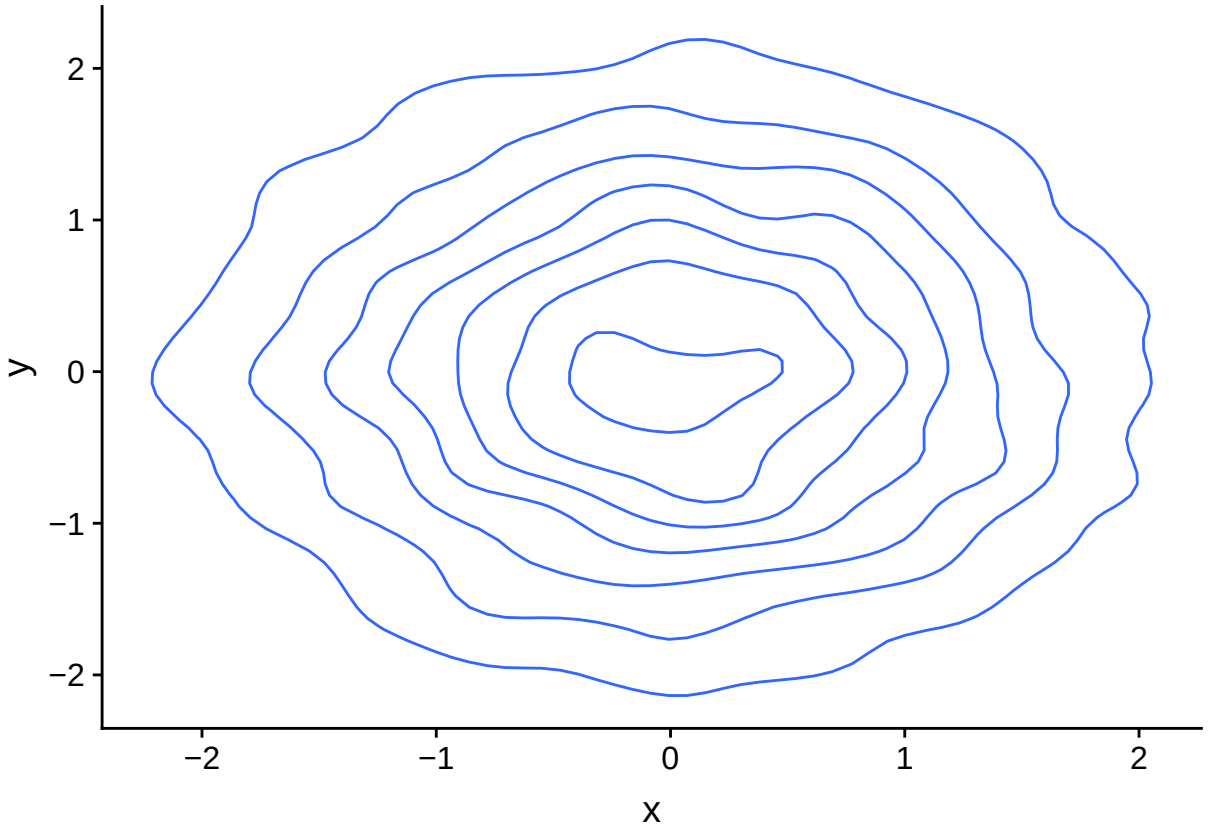
On peut utiliser le contraste pour foncer ou éclaircir certains points:

```
d %>% ggplot(aes(x = x, y = y)) +
  geom_point(alpha = 0.2)
```



On peut aussi se contenter d'afficher la densité

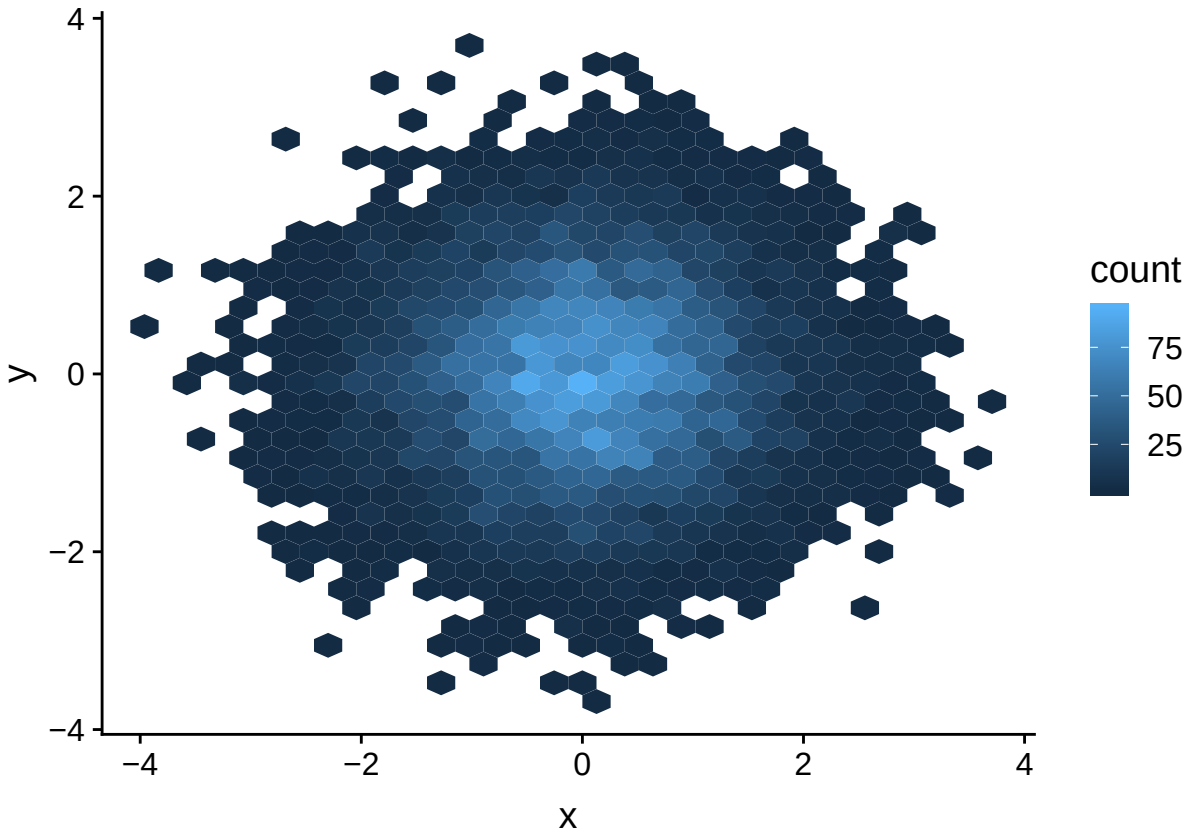
```
d %>% ggplot(aes(x = x, y = y)) +
  geom_density_2d()
```



Hex Figure

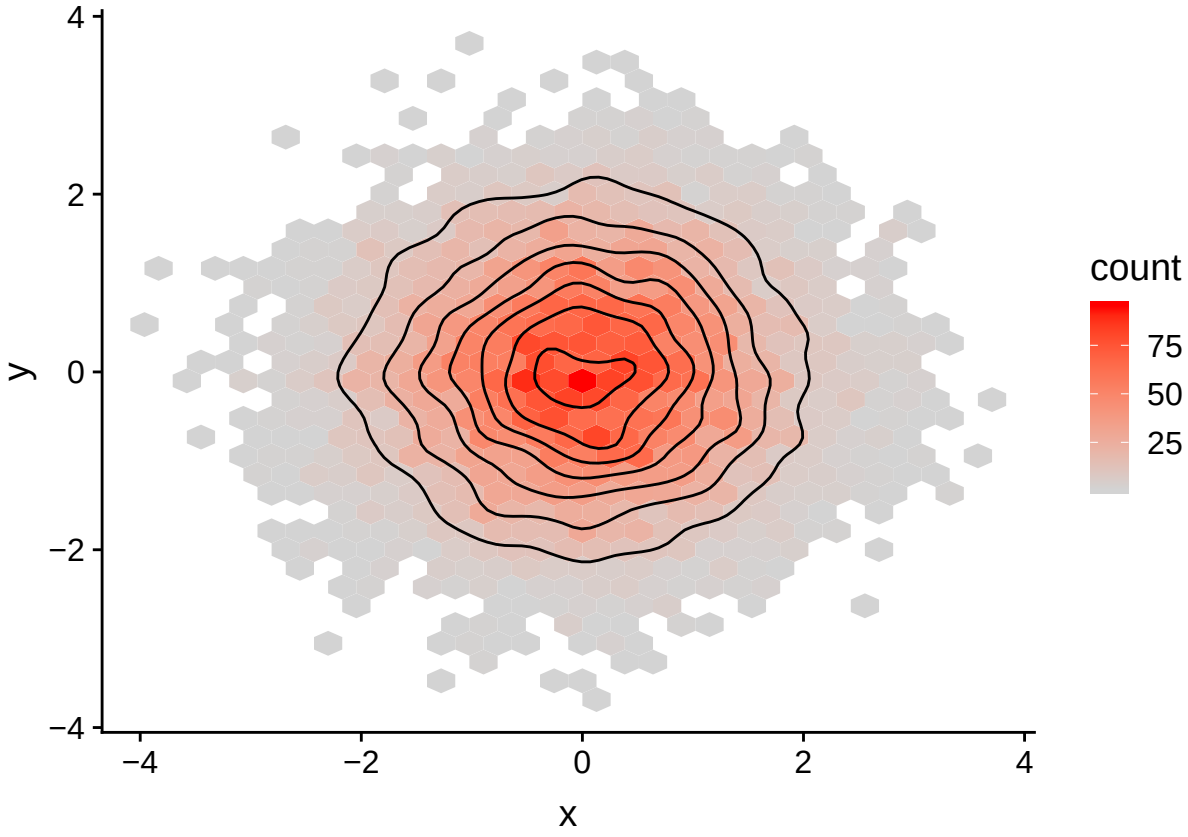
On peut afficher la densité en utilisant des figures hexagonal pour tracer les histogrammes au lieu des carrés ou barres classiques

```
d %>% ggplot(aes(x = x, y = y)) +
  geom_hex()
```



On peut aussi combiner les deux densité sous forme de courbe et sous forme d'histogramme dans un même graphe qui est plus intéressant que les deux graphes seuls:

```
d %>% ggplot(aes(x = x, y = y)) + geom_hex() +  scale_fill_gradient(low = "lightgray", high = "red") +
  geom_density2d(color = "black")
```


4
2
0
−2
−4
y
−4
−2
0
2
4
x
count
75
50
25