

# Visualisation avec

Dernière mise à jour : 09 sept. 2024

Sandrine LYSER

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Pour bien commencer</b>	<b>3</b>
2.1	Quelques conseils pour un travail reproductible	3
2.1.1	La reproductibilité	3
2.1.2	Conventions de nommage	4
2.1.3	Structurer son code	4
2.2	Utiliser R avec RStudio	5
2.2.1	Rappels sur le fonctionnement des packages	5
2.2.2	L'encodage de caractères	6
2.2.3	Travailler avec les projets RStudio	6
2.2.4	Le <b>tidyverse</b>	8
2.2.5	Importer divers types de données	9
2.2.6	Exporter les données	10
<b>3</b>	<b>Exploration des données</b>	<b>10</b>
3.1	Ouverture d'un jeu de données dans Excel	10
3.2	Exploration des données avec le package <code>[reactable]</code>	11
3.3	Visualisation des données manquantes	11
3.4	Exploration et manipulation des données avec le package <code>[dplyr]</code>	12
3.4.1	Le package <code>[dplyr]</code>	12
3.4.2	Sélection de <b>lignes</b> avec la fonction <code>filter()</code>	12
3.4.3	Sélection <b>d'une colonne</b> avec les fonctions <code>pull()</code> ou <code>select()</code>	13
3.4.4	Sélection de <b>plusieurs colonnes</b> avec <code>select()</code>	14
3.4.5	Résumé d'une ou plusieurs variables avec <code>summarise()</code>	14
3.4.6	Regroupement de données et calculs sur des groupes d'observations avec <code>group_by()</code>	15
3.4.7	Création de nouvelles variables avec <code>mutate()</code>	15
3.4.8	Changement de nom pour les colonnes avec <code>rename()</code>	16
3.5	Exploration automatique	16
3.5.1	Sous forme de tableau	16
3.5.2	Sous forme de rapport	17
<b>4</b>	<b>Visualisation des données</b>	<b>17</b>
4.1	Les types de visualisations graphiques	17
4.1.1	Représentations graphiques	17
4.1.2	Quelques rappels pour la réalisation des graphiques	18
4.1.3	Plusieurs systèmes graphiques sous R	18
4.2	Visualisation graphique avec <code>[ggplot2]</code>	18
4.2.1	Le package <code>[ggplot2]</code>	18
4.2.2	La grammaire <code>ggplot</code>	18
4.2.3	Éléments de personnalisation des graphiques	28
4.2.4	Sauvegarde des graphiques <code>ggplot</code>	29
4.2.5	Exemple de mise en forme d'un graphique	29
4.2.6	Des aides pour faciliter la réalisation des graphiques <code>[ggplot2]</code>	31
4.3	Pour aller plus loin	32
4.3.1	Graphiques animés	32

4.3.2	Autres productions . . . . .	33
<b>5</b>	<b>Pour conclure</b>	<b>34</b>
<b>6</b>	<b>Bibliographie</b>	<b>35</b>
<b>7</b>	<b>Annexes. Rappels sur la visualisation graphique R standard</b>	<b>37</b>
7.1	Création d'un graphique . . . . .	37
7.1.1	Fonction <code>plot()</code> . . . . .	39
7.1.2	Fonctions spécifiques . . . . .	40
7.2	Exemple de mise en forme d'un graphique . . . . .	42
7.3	Rappel sur les marges . . . . .	45

**Document diffusé sous licence  
CC BY-NC-SA 4.0**



# 1 Introduction

## Dans ce cours...

- vous apprendrez à utiliser R avec RStudio et à créer un projet RStudio
- vous apprendrez à travailler de manière **reproductible**, en organisant votre travail et en structurant vos scripts
- vous découvrirez l'univers du **tidyverse**
- vous apprendrez à visualiser les données
  - sous forme de tableaux
  - sous forme de graphiques, en utilisant [ggplot2](#)

**i** La plus grande partie des séances s'articulera autour de 2 grands volets : un volet présentation et un volet application.

Les exemples du cours s'appuient sur le jeu de données `iris`, qui se compose de 150 observations décrites par 4 variables

- **Sepal.Length** : longueur des sépales de la fleur d'iris
- **Sepal.Width** : largeur des sépales de la fleur d'iris
- **Petal.Length** : longueur des pétales de la fleur d'iris
- **Petal.Width** : largeur des pétales de la fleur d'iris
- **Species** : espèce de la fleur d'iris

## ! Évaluation des connaissances

### 1. Une évaluation "continue"

Le **script** des exercices d'application réalisés en séance devra être **rendu lors de la dernière séance**

### 2. Une évaluation lors de la dernière séance

Un **QCM** d'évaluation des connaissances + un **exercice** supplémentaire

# 2 Pour bien commencer

## 2.1 Quelques conseils pour un travail reproductible

### 2.1.1 La reproductibilité

= **Processus qui peut être reproduit par une autre personne et/ou à partir des mêmes données mises à jour et qui permet d'expliquer et partager son travail**

S'appuie sur 3 grands principes, interagissant entre eux et non séquentiels (Orozco et al., 2020)

1. Organiser son travail
  - Utiliser les scripts pour conserver l'ensemble du code
  - Structurer en répertoires pour séparer données brutes, données créées, scripts, documentation, résultats, etc.
  - Définir une stratégie pour différencier fichiers mis à jour/anciens fichiers
  - Utiliser une convention de dénomination des fichiers et leur donner des noms explicites

**💡** Pour les **fichiers**, on peut les nommer avec la structure `numero_nom_millesime.extension`, sans symboles spéciaux dans le nom

Exemple : `01_import_donnees_20230606.R`

**💡** **Caractères spéciaux** : `' - . , ; : \ / $ ^` caractères accentués, etc.

## 2. Coder pour les autres

- Donner des noms explicites aux objets et fonctions et utiliser les conventions de nommage
- Distinguer variables d'origine et variables créées en adoptant une règle de convention (par ex. utiliser `recod` en préfixe ou suffixe du nom des variables)
- Utiliser des chemins relatifs dans les programmes pour exécuter sans problème le code sur un autre ordinateur
- Coder de façon compréhensible avec des commentaires



- Pour les **variables**, utiliser un **nom** ex : `temperature_max`
- Pour les **fonctions**, utiliser un **verbe** ex : `create.map` serait une fonction qui permettrait de créer une carte

## 3. Automatiser le plus possible

- Éviter le copier-coller
- Faciliter le processus de création de chaque tableau, figure, etc.
- Identifier l'ordre d'exécution des programmes pour ré-exécuter facilement tous les programmes



Aide sur le processus d'analyse et de communication des données reproductibles



R Workflow

### 2.1.2 Conventions de nommage

- **alllowercase** : tout en minuscule, sans séparateur
- **period.separated** : tout en minuscule, mots séparés par des points
- **underscore\_separated** : tout en minuscule, mots séparés par un *underscore* (`_`)
- **lowerCamelCase** : première lettre des mots en majuscule, à l'exception du premier mot ; et si nom simple, tout en minuscule
- **UpperCamelCase** : première lettre des mots en majuscule, y compris le premier et même lorsque le nom est composé d'un seul mot



**Possibilité de mixer les conventions mais garder une cohérence afin de faciliter la compréhension des fichiers et du code**



Mise en garde

**R est sensible à la casse** ce qui signifie que `variable` et `Variable` sont deux objets différents

### 2.1.3 Structurer son code

1. Commencer l'écriture du script par les métadonnées (titre, auteur, date, version de R, encodage, etc.)

```
1 # ...
2 # ... Script: ...
3 # ... Author: ...
4 # ... Créé le : ...
5 # ... Dernière modification : ...
6 # ... Text default encoding : UTF-8 ...
7 # ... R version 4.2.1 (2022-06-23 ucrt) --- "Funny-Looking Kid" ...
8 # ...
```

2. Ajouter la liste des packages utilisés et leur version

```
# "à la main"
library(tidyverse)    # v1.3.2
library(dplyr)        # v1.1.0
```

OU

```
# automatique
packages_utilises <- sessionInfo()
```

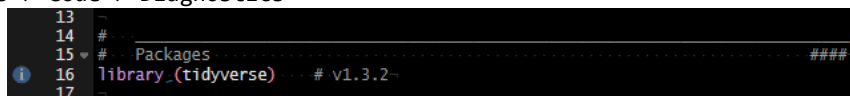
3. Structurer en parties, sous-parties, grâce à l'*addin strcode*

```
#
# Titre 1 #####
## .....
## Titre 2 #####
```

💡 Aide pour écrire les scripts

🌐 [Tidyverse Style Guide](#)

⚙️ Activer les options de diagnostic pour signaler les erreurs dans les scripts sous RStudio : **Tools > Global Options > Code > Diagnostics**



```
13
14 #
15 # Packages .....
16 library(tidyverse) ... # v1.3.2
17
```

- package [lstyler](#) : mettre en forme le code selon le *Tidyverse Style Guide* (ou selon son propre style)
- package [lintr](#) : vérifier que le code est conforme à un guide de style donné

## 2.2 Utiliser R avec RStudio

### 2.2.1 Rappels sur le fonctionnement des packages

1. Installer, **dans cet ordre**, les logiciels R à partir du [CRAN](#) (*Comprehensive R Archive Network*) & RStudio via le site de [Posit](#)
2. **Installer** un package

```
# Installer Le package ggplot2
install.packages("ggplot2")
```

3. **Charger** un package

```
# Charger Le package ggplot2
library(ggplot2)
```

### 💡 Le préfixage

La notation

```
dplyr::filter(iris, Species == "setosa")
```

peut remplacer le code

```
library(dplyr)
filter(iris, Species == "setosa")
```

#### Avantages

- Évite les conflits si 2 packages ont une fonction avec le même nom (mais des paramètres différents)
- Identifie le package d'une fonction
- Ne charge pas la totalité d'un package pour n'utiliser qu'une seule de ses fonctions

**Ne pas oublier de citer dans les rapports, la version de R ainsi que les packages et les fonctions utilisés**

```
# Pour citer R
citation()
```

```
# Pour citer un package (ici ggplot2)
citation("ggplot2")
```

### 2.2.2 L'encodage de caractères

- Plusieurs encodages
  - ASCII
  - ISO 8859-15
  - UTF-16
  - UTF-8
  - et bien d'autres encore
- différents selon le système d'exploitation
  - sous Windows : Windows-1252
  - sous Linux : UTF-8

**Utiliser l'encodage UTF-8 pour les scripts et les données**, le plus universel à l'heure actuelle

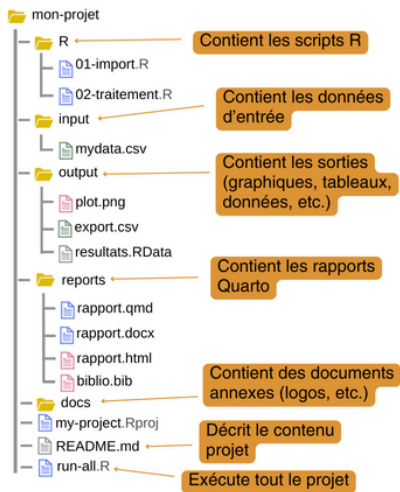
### 💡 Comment paramétrer sous RStudio ?

⚙️ Aller dans Tools > Global Option > Code > Saving > Default text encoding et sélectionner UTF-8

### 2.2.3 Travailler avec les projets RStudio

Possibilité de travailler sur plusieurs projets simultanément ⇒ meilleure organisation du travail

Répertoire par défaut du projet = dossier où est enregistré le projet ⇒ améliore la portabilité



```
name_of_project
|--data
|  |--2017report.csv
|  |--2016report.pdf
|  |--summary2016_2017.csv
|--docs
|  |--01-analysis.Rmd
|  |--01-analysis.html
|--scripts
|  |--exploratory_analysis.R
|  |--pdf_scraper.R
|--name_of_project.Rproj
|--run_all.R
```

```
name_of_project
|--raw_data
|  |--WhateverData.xlsx
|  |--2017report.csv
|  |--2016report.pdf
|--output_data
|  |--summary2016_2017.csv
|--rmd
|  |--01-analysis.Rmd
|--docs
|  |--01-analysis.html
|  |--01-analysis.pdf
|  |--02-deeper.html
|  |--02-deeper.pdf
|--scripts
|  |--exploratory_analysis.R
|  |--pdf_scraper.R
|--name_of_project.Rproj
|--run_all.R
```

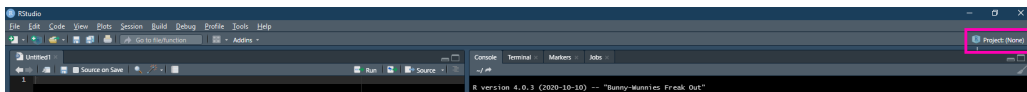
Source (figures 2 & 3) : <https://learn.r-journalism.com/en/publishing/workflow/r-projects/>



- **Rassembler tous les éléments liés au projet dans un seul dossier**, avec une organisation la plus adaptée à ses usages
- Le plus important, ce sont les **données initiales** et les **scripts de traitement**

### Comment faire ?

À partir de l'icône dédiée en haut à droite de RStudio



Cliquer ensuite sur New Directory > New Project et compléter les informations requises

New Project Wizard

Back Create New Project

Directory name:

Create project as subdirectory of:  Browse...

☐ Create a git repository

☐ Use renv with this project


☐ Open in new session

Create Project Cancel

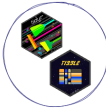
- Ensemble de packages qui reprend des opérations courantes de R mais de façon unifiée avec une syntaxe commune

## LES DIFFÉRENTES ÉTAPES DE LA DATA SCIENCE

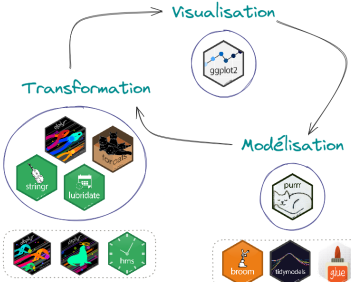
### Les packages du Tidyverse & leurs principaux objectifs




#### RANGEMENT



#### EXPLORATION



#### COMMUNICATION



verse, chargés automatiquement avec `library(tidyverse)`

tidyverse, à charger explicitement

fournit le pipe de tidyverse

pour créer un exemple reproductible

- `{dplyr}` : manipulation des données
- `{forcats}` : traitement des variables qualitatives
- `{ggplot2}` : visualisation des données
- `{lubridate}` : manipulation des dates et heures
- `{purrr}` : programmation
- `{readr}` : import de données
- `{stringr}` : manipulation des chaînes de caractères
- `{tibble}` : version modernisée des *dataframes*, plus pratiques à utiliser que les *dataframes* "classiques"
- `{tidyr}` : nettoyage, remise en forme des données

**</>** Le tidyverse fournit dans le package `magrittr` un *pipe*, noté `>%>`, qui permet d'enchaîner les opérations de traitement.

```
data %>%
  fonction1() %>%
  fonction2() %>%
  fonction3()
# équivaut à
fonction3(fonction2(fonction1(data)))
```

```
data |>
  fonction1() |>
  fonction2() |>
  fonction3()
```



## 2.2.5 Importer divers types de données

- Données dans des formats "plats" ou délimités (.csv, .txt)
- Données issues d'Excel ou d'autres logiciels de statistique (Stata, SPSS, etc.)
- Données stockées dans des bases de données relationnelles

💡 Bien étudier son jeu de données à importer, en se posant les questions suivantes

1. La première ligne contient-elle le nom des variables ?
2. Quel est le séparateur des colonnes ? (, , ;, espace(s), \t)
3. Quel est le caractère utilisé pour indiquer les décimales ? Le point (à l'anglo-saxonne) ou la virgule (à la française) ?
4. Les valeurs textuelles sont-elles encadrées par des guillemets ? Si oui, s'agit-il de guillemets simples (') ou de guillemets doubles (") ?
5. Y a-t-il des valeurs manquantes ? Si oui, comment sont-elles indiquées ?

### 2.2.5.1 En pratique

Type de fichier	Séparateur de colonnes	Séparateur décimal	Base R	Tidyverse
Délimité	,	.	read.csv()	readr::read_csv()
	;	,	read.csv2()	readr::read_csv2()
	un espace (ou plus)	.	read.table()	readr::read_table()
	\t	.	read.delim()	readr::read_delim()
	\t	,	read.delim2()	readr::read_delim2()
Excel			xlsx::read.xlsx()	readxl::read_excel()
SPSS			foreign::read.spss()	haven::read_spss()
				haven::read_sav()
Stata			foreign::read.dta()	haven::read_stata()
SAS			foreign::read.ssd()	haven::read_sas()
dBase			foreign::read.dbf()	

### 2.2.5.2 Exemple pour un fichier .csv

```
# En Langage R de base
read.csv(file.csv, header = TRUE, sep = ";", dec = ".")
```

```
# En "tidyverse"
readr::read_csv(file.csv, col_names = TRUE)
```

**i** Les fonctions `read.csv2()` et `readr::read_csv2()` considèrent par défaut que le séparateur est le point-virgule et que la virgule sert de séparateur décimal

## 2.2.6 Exporter les données

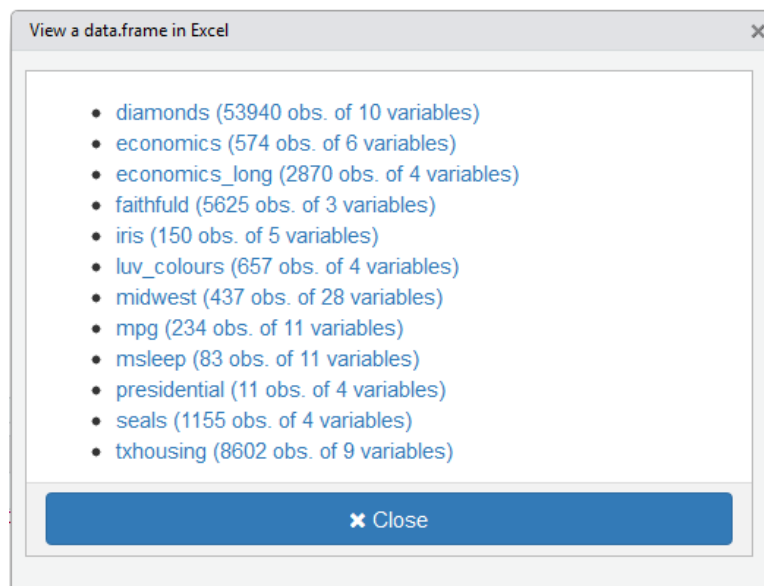
Type de fichier	Base R	Tidyverse
.txt	<code>write.table()</code>	<code>readr::write_delim(delim = "\t")</code>
.csv	<code>write.csv()</code>	<code>readr::write_csv()</code>
.xlsx	<code>openxlsx2::write_xlsx()</code> OU <code>writexl::write_xlsx()</code>	
.dbf	<code>foreign::write.dbf()</code>	
.sav (SPSS)	<code>foreign::write.foreign(package = "SPSS")</code>	<code>haven::write_sav()</code>
.dta (Stata)	<code>foreign::write.dta()</code>	<code>haven::write_dta()</code>

## 3 Exploration des données

### 3.1 Ouverture d'un jeu de données dans Excel

L'*addin viewxl* permet d'exporter interactivement des *dataframe* de l'environnement global vers Excel

- avec la fonction `view_in_xl`



- dans la console

```
viewxl::view_in_xl(iris)
```

### 3.2 Exploration des données avec le package `reactable`

Visualisation plus élaborée qu'avec le tableur de R

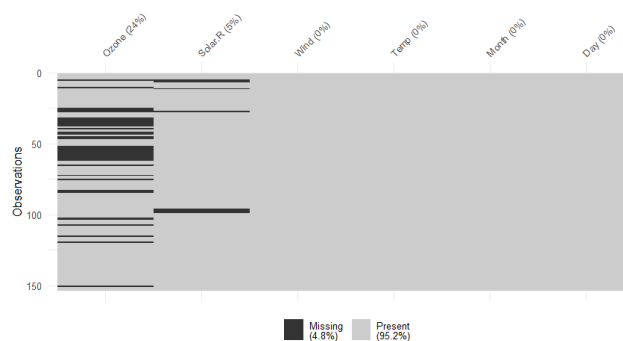
```
reactable::reactable(iris,
  filterable = TRUE, # pour filtrer les données
  searchable = TRUE, # pour faire une recherche dans les données
  defaultPageSize = 4) # pour paramétrer le nombre de lignes affichées
```

<div>Search</div>				
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
1-4 of 150 rows		Previous	1	2 3 4 5 ... 38 Next

### 3.3 Visualisation des données manquantes

La fonction `vis_miss()` du package `visdat` permet de visualiser les données manquantes d'un jeu de données

```
airquality |>
  visdat::vis_miss()
```



### 3.4 Exploration et manipulation des données avec le package `dplyr`

#### 3.4.1 Le package `dplyr`

Ce package, dédié à la manipulation des données, peut être chargé via le `tidyverse` ou individuellement

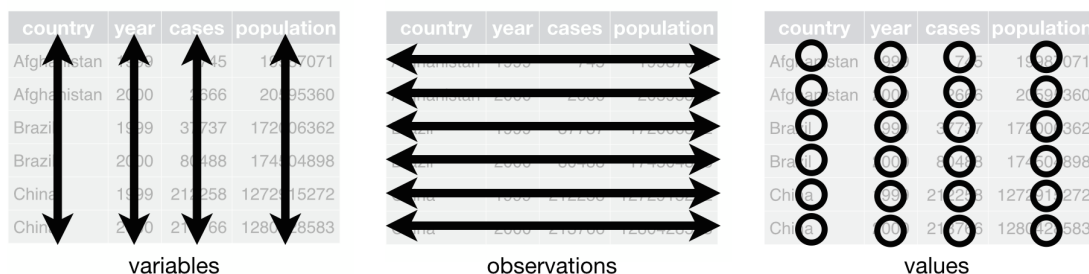
```
# Chargement via le tidyverse
library(tidyverse)
```

OU

```
# Chargement du package seul
library(dplyr)
```

S'utilise sur des jeux de données "*tidy*"

- chaque variable est dans une colonne
- chaque observation est dans une ligne
- chaque valeur est dans une cellule



Source : Wickham & Golemund (2018)

#### Principales fonctions à connaître

- `filter()` : sélection de lignes
- `select()` : sélection de variables
- `summarise()` : résumé des données
- `group_by()` : regroupement de données
- `mutate()` : création des variables
- `rename()` : renommage des variables

 [Cheatsheet dplyr](#) (aide-mémoire des principales fonctions)

#### 3.4.2 Sélection de lignes avec la fonction `filter()`

- Selon un seul critère

```
# Sepal.Length est supérieure à 7.5
iris |>
  dplyr::filter(Sepal.Length > 7.5)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	7.6	3.0	6.6	2.1	virginica
2	7.7	3.8	6.7	2.2	virginica
3	7.7	2.6	6.9	2.3	virginica
4	7.7	2.8	6.7	2.0	virginica
5	7.9	3.8	6.4	2.0	virginica
6	7.7	3.0	6.1	2.3	virginica

- **Selon plusieurs critères**, grâce aux opérateurs & (ET) et | (OU)

```
# Sepal.Length supérieure à 5 ET Petal.Length inférieure à 1.4
iris |>
  dplyr::filter(Sepal.Length > 5 & Petal.Length < 1.4)
```

```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.8         4.0         1.2         0.2  setosa
2          5.4         3.9         1.3         0.4  setosa
3          5.5         3.5         1.3         0.2  setosa
```

#### **i** Opérateurs logiques les plus courants

Opérateur	Description
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
==	Exactement égal à
!=	Différent de
	Ou
&	Et
%in%	Appartient à
is.na()	Est une donnée manquante

### 3.4.3 Sélection d'une colonne avec les fonctions pull() ou select()

- **Fonction pull()**

```
# Sélection de la variable Species
iris |>
  dplyr::pull(Species)
```

```
[1] setosa setosa setosa setosa setosa setosa
[7] setosa setosa setosa setosa setosa setosa
[13] setosa setosa setosa setosa setosa setosa
[19] setosa setosa setosa setosa setosa setosa
...
```

- **Fonction select()**

```
# Sélection de la variable Species
iris |>
  dplyr::select(Species)
```

```
Species
1  setosa
2  setosa
3  setosa
...
```

#### **💡** Quelle différence entre ces 2 fonctions ?

La fonction `pull()` renvoie une la colonne sous forme de vecteur, tandis que la fonction `select()` renvoie la colonne sous forme de *dataframe*

### 3.4.4 Sélection de plusieurs colonnes avec select()

```
# Sélection des variables sauf Species
# = on indique la variable à supprimer
iris |>
  dplyr::select(-Species)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1           5.1          3.5          1.4          0.2
2           4.9          3.0          1.4          0.2
3           4.7          3.2          1.3          0.2
...
```

```
# Sélection de Petal.Length et Sepal.Length
# = on sélectionne les variables à conserver
iris |>
  dplyr::select(c("Petal.Length",
                  "Sepal.Length"))
```

```
  Petal.Length Sepal.Length
1           1.4           5.1
2           1.4           4.9
3           1.3           4.7
...
```



On peut faire les tests sur les noms de variables avec les fonctions `contains()`, `starts_with()`, `ends_with()` ou encore `matches()`

```
# Sélection des variables dont le nom contient "Length"
# = on sélectionne en fonction du nom des variables
iris %>%
  dplyr::select(contains("Length"))
```

Voir la page d'aide de la fonction `select()` du package [dplyr](#) pour plus d'informations sur ces fonctions et les autres fonctions disponibles

### 3.4.5 Résumé d'une ou plusieurs variables avec summarise()

#### • Une variable

```
# Valeur la plus faible de Sepal.Length
iris |>
  summarise(min(Sepal.Length))
```

```
min(Sepal.Length)
1           4.3
```

#### • Plusieurs variables

```
# Moyenne de chaque variable dont le nom contient "Width"
iris |>
  summarise(across(contains("Width"), mean))
```

```
  Sepal.Width Petal.Width
1    3.057333    1.199333
```

```
# Moyenne de chaque variable numérique du tableau
iris |>
  summarise(across(where(is.numeric),
                    ~ mean(.x, na.rm = TRUE)))
```

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

💡 Indicateurs les plus couramment utilisés dans la fonction `summarise()`

- `min()` & `max()`
- `mean()` & `sd()`
- `median()`, `quantile()` & `IQR()`
- `n()`

### 3.4.6 Regroupement de données et calculs sur des groupes d'observations avec `group_by()`

```
# Calcul de la moyenne de Sepal.Length en fonction de la variable Species
iris |>
  group_by(Species) |>                                # Groupes selon les modalités de 'Species'
  summarise(Moyenne = mean(Sepal.Length,              # Moyenne pour chaque groupe
                        na.rm = TRUE))
```

```
# A tibble: 3 x 2
  Species Moyenne
  <fct>    <dbl>
1 setosa    5.01
2 versicolor 5.94
3 virginica  6.59
```

### 3.4.7 Création de nouvelles variables avec `mutate()`

Exemple de création de deux variables : *Sepal\_Area* (superficie des sépales) et *Petal\_Area* (superficie des pétales)

```
iris |>
  dplyr::mutate(Sepal_Area = Sepal.Length * Sepal.Width,
                Petal_Area = Petal.Length * Petal.Width)
```

```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal_Area Petal_Area
1           5.1         3.5          1.4         0.2   setosa     17.85      0.28
2           4.9         3.0          1.4         0.2   setosa     14.70      0.28
3           4.7         3.2          1.3         0.2   setosa     15.04      0.26
...
```

#### 🔥 Mise en garde

Dans cet exemple, les nouvelles variables sont calculées (et affichées dans la console) mais ne sont pas ajoutées au *dataframe*.

Pour ajouter la(les) nouvelle(s) variable(s) créée(s) avec `mutate()` au jeu de données, ne pas oublier de les affecter au *dataframe* :

```
iris |>
  dplyr::mutate(Sepal_Area = Sepal.Length * Sepal.Width,
                Petal_Area = Petal.Length * Petal.Width) -> iris
```

### 3.4.8 Changement de nom pour les colonnes avec `rename()`

```
# On renomme la colonne 'Species' en 'Especes'
iris |>
  dplyr::rename(Especes = Species)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Especes	Sepal_Area	Petal_Area
1	5.1	3.5	1.4	0.2	setosa	17.85	0.28
2	4.9	3.0	1.4	0.2	setosa	14.70	0.28
3	4.7	3.2	1.3	0.2	setosa	15.04	0.26
...							

#### Mise en garde

Comme précédemment, dans cet exemple, la variable renommée n'est pas modifiée dans le *dataframe* `iris`.

Pour modifier de façon durable le nom de la variable dans le *dataframe*, il ne faut pas oublier d'affecter la modification au *dataframe* :

```
iris |>
  dplyr::rename(Especes = Species) -> iris
```

## 3.5 Exploration automatique

- Plusieurs packages fournissent des fonctions permettant d'avoir un aperçu **automatique** du jeu de données...
  - nombre d'observations
  - nombre de variables
  - type des variables
  - résumé statistique
  - etc.
- ... avec un rendu différent
  - uniquement sous forme de tableaux
  - avec représentations graphiques
  - compilé dans un rapport

### 3.5.1 Sous forme de tableau

#### Aperçu de la structure du jeu de données, des variables

- Fonction `str()`

```
str(iris)
```

- Fonction `glimpse()` du package `[dplyr]`

```
dplyr::glimpse(iris)
```

- Fonction `describe()` du package `[questionr]`

```
questionr::describe(iris)
```



## Statistiques descriptives, adaptées à la nature des variables (QN ou QL)

- Fonction `summary()`

```
summary(iris)
```

- Fonction `tbl_summary()` du package `gtsummary`

```
gtsummary::tbl_summary(iris)
```

Les résultats sont disponibles dans l'onglet 'Viewer'

- Fonction `dfSummary()` du package `summarytools`

```
summarytools::dfSummary(iris)
```

- Fonction `skim()` du package `skimr`

```
skimr::skim(iris)
```

### 3.5.2 Sous forme de rapport

Le package `DataExplorer` permet d'explorer le jeu de données "en un clic"

- Les fonctions `plot_histogram()`, `plot_bar()`, `plot_density()` fournissent des sorties graphiques adaptées automatiquement au type des variables

```
iris |>  
DataExplorer::plot_histogram()
```

- La fonction `create_report()` permet de créer un rapport automatique, au format HTML, qui donne un état des lieux du jeu de données (type des variables, données manquantes) et diverses visualisations graphiques (distributions, analyses de corrélations, ACP)

```
iris |>  
DataExplorer::create_report()
```



**Toutes ces commandes ont une réelle utilité pour une première exploration des données, mais elles ne remplacent pas une analyse approfondie et adaptée à la problématique de l'étude**

## 4 Visualisation des données

### 4.1 Les types de visualisations graphiques

#### 4.1.1 Représentations graphiques

Différentes formes de représentations graphiques en fonction

- des principales variables utilisées (variables continues, discrètes ou un mix des deux)
- et de leur nombre (une ou plusieurs)

Voir le site [from Data to Viz](#)

### 4.1.2 Quelques rappels pour la réalisation des graphiques

- "Le moins est le mieux"  $\Rightarrow$  épurer le graphique et retirer la 3D pour le rendre plus compréhensible
- Les objets proches sont perçus comme appartenant à un même groupe  $\Rightarrow$  positionner côte à côte les éléments que l'on souhaite voir analysés ensemble, comparés l'un à l'autre
- Certains éléments peuvent être omis sans réduire la compréhension (axe des abscisses selon les cas)
- Pour alléger les visualisations, ne pas "fermer" les graphiques avec un cadre
- Dans un diagramme en barre, l'axe des ordonnées commence à 0

### 4.1.3 Plusieurs systèmes graphiques sous R

- Un système de base, avec le package `{graphics}`
- Un système plus complet, le système *grid*, sur lequel se basent les packages
  - `{lattice}` qui fournit des visualisations de données, puissantes et élégantes, inspirées des graphiques treillis
  - `{ggplot2}` qui définit des graphiques en utilisant une grammaire particulière (*grammar of graphics*)

## 4.2 Visualisation graphique avec `{ggplot2}`

### 4.2.1 Le package `{ggplot2}`

- Installation

```
install.packages("ggplot2")
```

- Chargement

```
library(ggplot2)
```

- Création d'un graphique

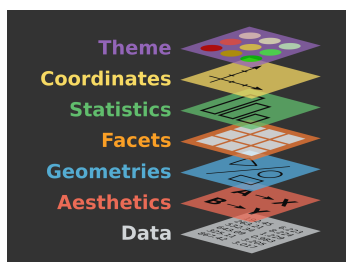
Le principe consiste à initialiser un graphique avec la commande `ggplot()` puis à ajouter des couches permettant de représenter les données et mettre en forme le graphique

```
ggplot(data = dataframe, aes = (x, y)) + ...
```

- Le système de création des graphiques est basé sur *The Grammar of Graphics* publié initialement par Wilkinson (1999)
- L'idée est de décomposer la construction d'un graphique en plusieurs éléments, que l'on appelle *layers* (couches, en français)

### 4.2.2 La grammaire ggplot

Une structure en 7 *layers* superposables



Source : [ThinkR \(2016\)](#)

**! Important**

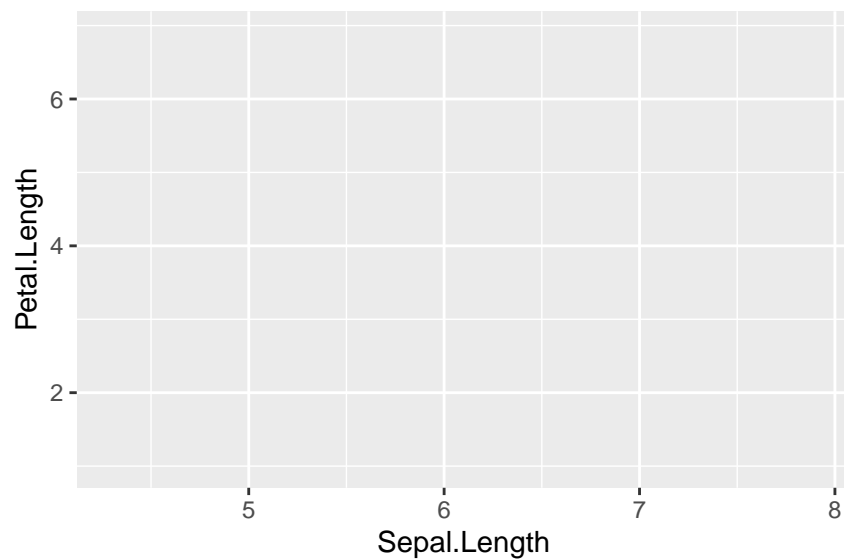
Les trois premiers *layers* (*data*, *aesthetics* et *geometries*) sont indispensables pour créer un graphique  
Les quatre autres *layers* permettent de paramétrer plus finement le graphique

#### 4.2.2.1 Couches *data* & *aes*

Suivent l'initialisation du graphique faite avec `ggplot()`

- **data** : jeu de données (*dataframe* ou *tibble*)
- **aes** (aesthetics) : variables à représenter

```
ggplot(data = iris) +  
  aes(x = Sepal.Length, y = Petal.Length)
```



À ce stade, on visualise le canevas du graphique mais les données ne sont pas encore représentées

- Dans `aes()` on précise les éléments visuels du graphique

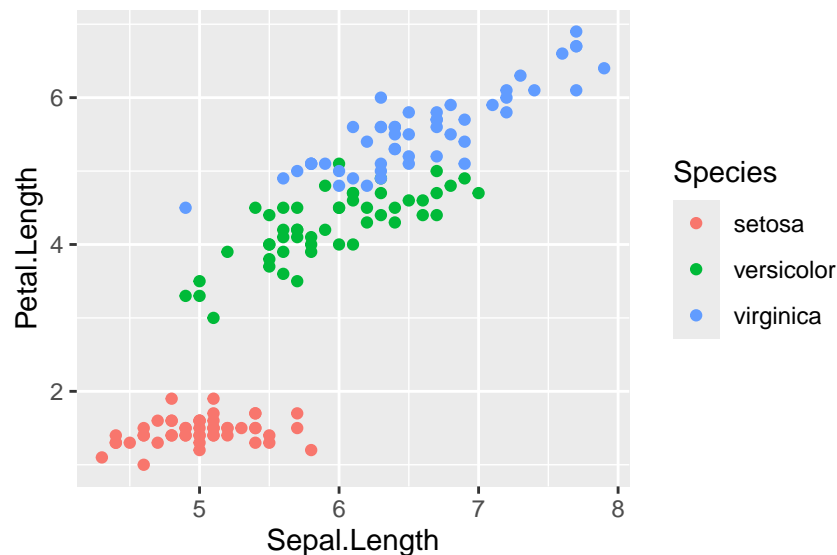
<i>Aesthetic</i>	Description
x	variable en abscisse
y	variable en ordonnée
colour	couleur du contour (des points, lignes, etc.)
fill	couleur de remplissage (des points, formes, etc.)
size	taille des points, épaisseur des lignes
alpha	transparence
linetype	motif des tirets des lignes
shape	type de formes

```
# Points colorés selon les modalités de Species  
ggplot(data = iris) +  
  aes(x = Sepal.Length, y = Petal.Length,  
      colour = Species)
```

#### 4.2.2.2 Couche *geom*

Détermine le type de représentation souhaité

```
# Nuage de points avec geom_point
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Petal.Length,
      colour = Species) +
  geom_point()
```



Avec *geom\_*, les données sont représentées sur le graphique

##### • Principales géométries

Geometry	Description	Aspects esthétiques
<code>geom_point()</code>	Nuage de points ( $\Leftrightarrow$ <code>plot()</code> )	<b>x</b> , <b>y</b> , colour, fill, group, shape, size, alpha
<code>geom_bar()</code>	Diagramme en barres ( $\Leftrightarrow$ <code>barplot()</code> )	<b>x</b> , <b>y</b> , colour, fill, group, linetype, linewidth, alpha
<code>geom_histogram()</code>	Histogramme ( $\Leftrightarrow$ <code>hist()</code> )	<i>identiques à <code>geom_bar()</code></i>
<code>geom_boxplot()</code>	Boîte à moustaches	<b>x</b> OU <b>y</b> , <b>ymin</b> OU <b>xmin</b> , <b>ymax</b> OU <b>xmax</b> , colour, fill, group, linetype, linewidth, shape, size, alpha
<code>geom_density()</code>	Densité	<b>x</b> , <b>y</b> , colour, fill, group, linetype, linewidth, alpha
<code>geom_text()</code>	Texte	<b>x</b> , <b>y</b> , label, colour, angle, group, hjust, lineheight, size, vjust, alpha
<code>geom_label()</code>	Texte	<i>identiques à <code>geom_text()</code> + fill</i>

Les aspects esthétiques obligatoires sont indiqués en gras

Les fonctions *geom\_* s'ajoutent avec l'opérateur +

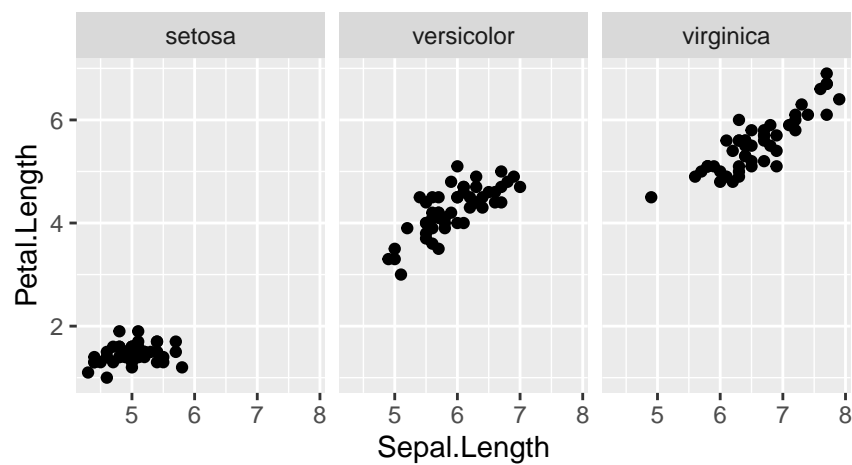
 L'ensemble des fonctions *geom\_* est détaillé dans la *cheatsheet [ggplot2]* 

#### 4.2.2.3 Couche *facet*

Sépare la fenêtre graphique pour faire le même graphique en fonction des modalités d'une ou plusieurs variables qualitatives

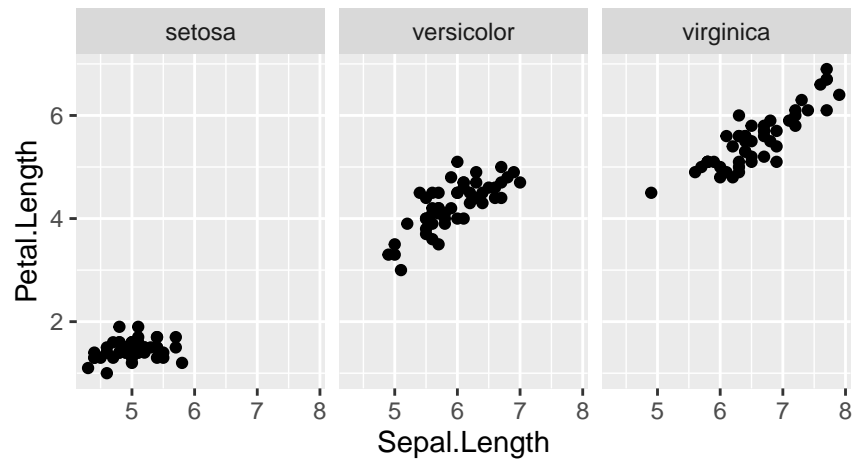
- `facet_wrap()` : graphiques disposés **les uns à côté des autres**, avec une répartition automatique dans la page

```
# Graphiques en fonction de "Species"  
ggplot(data = iris) +  
  aes(x = Sepal.Length,  
      y = Petal.Length) +  
  geom_point() +  
  facet_wrap(vars(Species))
```



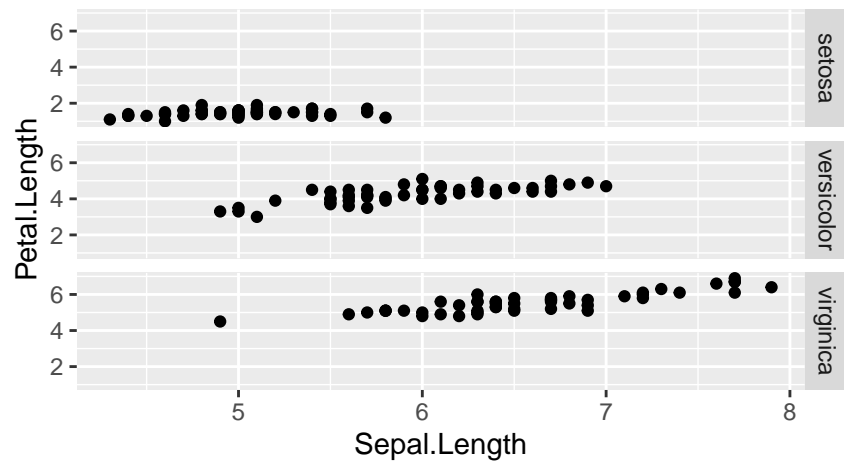
- `facet_grid()` : graphiques disposés selon une grille, avec une **répartition en colonne**

```
# Graphiques en fonction de "Species"
ggplot(data = iris) +
  aes(x = Sepal.Length,
      y = Petal.Length) +
  geom_point() +
  facet_grid(cols = vars(Species))
```



- `facet_grid()` : graphiques disposés selon une grille, avec une **répartition en ligne**

```
# Graphiques en fonction de "Species"
ggplot(data = iris) +
  aes(x = Sepal.Length,
      y = Petal.Length) +
  geom_point() +
  facet_grid(rows = vars(Species))
```

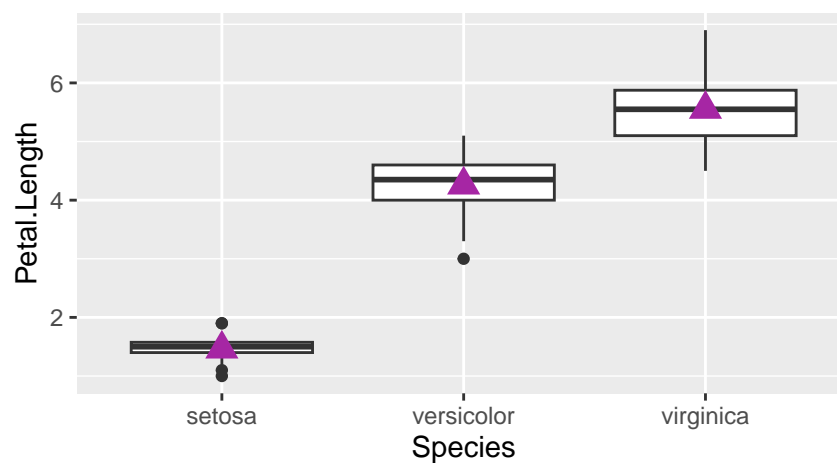


#### 4.2.2.4 Couche *stat*

Effectue un calcul sur les données avant qu'elles ne soient affichées

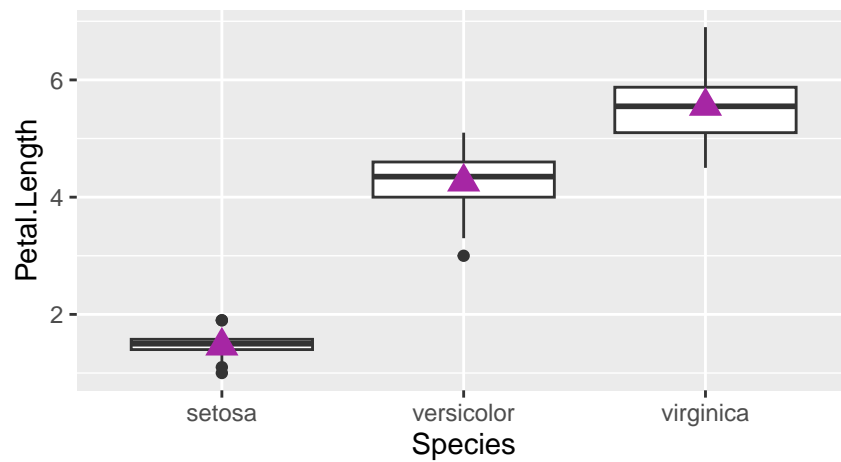
- avec une fonction `stat_()`

```
# Moyenne de Petal.Length
# pour chaque modalité de Species
# et affichage sur les boxplots
ggplot(data = iris) +
  aes(x = Species,
      y = Petal.Length) +
  geom_boxplot() +
  stat_summary(geom = "point",
              fun = "mean",
              colour = "#A626A4",
              shape = 17,
              size = 4)
```



- avec une fonction `geom_()`

```
# Moyenne de Petal.Length
# pour chaque modalité de Species
# et affichage sur les boxplots
ggplot(data = iris) +
  aes(x = Species,
      y = Petal.Length) +
  geom_boxplot() +
  geom_point(stat = "summary",
            fun = "mean",
            colour = "#A626A4",
            shape = 17,
            size = 4)
```

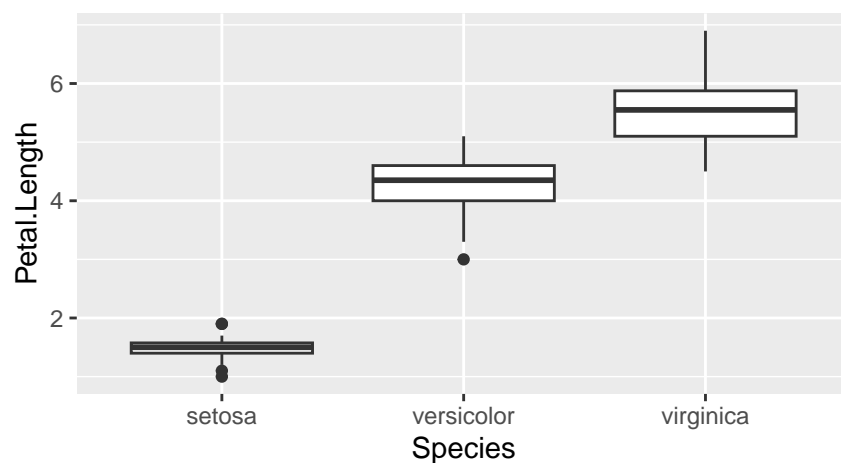


### Quelques équivalences "stats/geoms"

<i>Statistics</i>	<i>Geometry</i>
stat_count()	geom_bar()
stat_bin()	geom_histogram()
stat_boxplot()	geom_boxplot()
stat_density()	geom_density()
stat_quantile()	geom_quantile()
stat_sum()	geom_count()
stat_bindot()	geom_dotplot()

- avec geom\_boxplot()

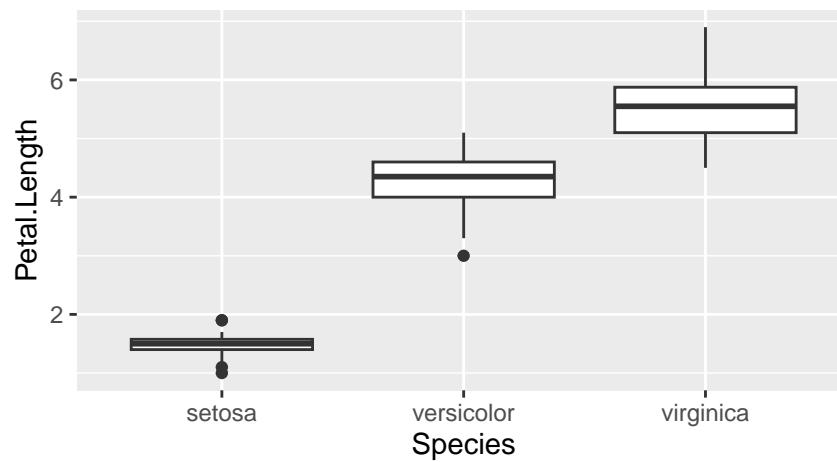
```
ggplot(data = iris) +
  aes(x = Species,
      y = Petal.Length) +
  geom_boxplot()
```



- avec stat\_boxplot()

```
ggplot(data = iris) +
  aes(x = Species,
      y = Petal.Length) +
  stat_boxplot()
```

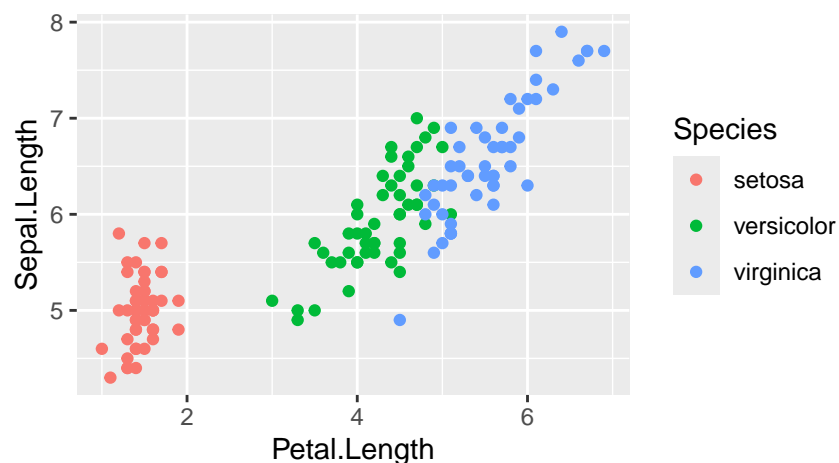




#### 4.2.2.5 Couche coord

- Permet de choisir le système de coordonnées dans lequel sont projetées les données
- Par défaut, il s'agit du système cartésien, mais on trouve d'autres systèmes :
  - `coord_equal()` : pour avoir la même échelle pour l'axe des abscisses et des ordonnées (cartésien)
  - `coord_flip()` : pour inverser l'axe des abscisses et des ordonnées
  - `coord_cartesian()` : pour fixer des limites
  - `coord_radial()` : pour les graphiques circulaires
  - `coord_map()` : pour différentes projections cartographiques

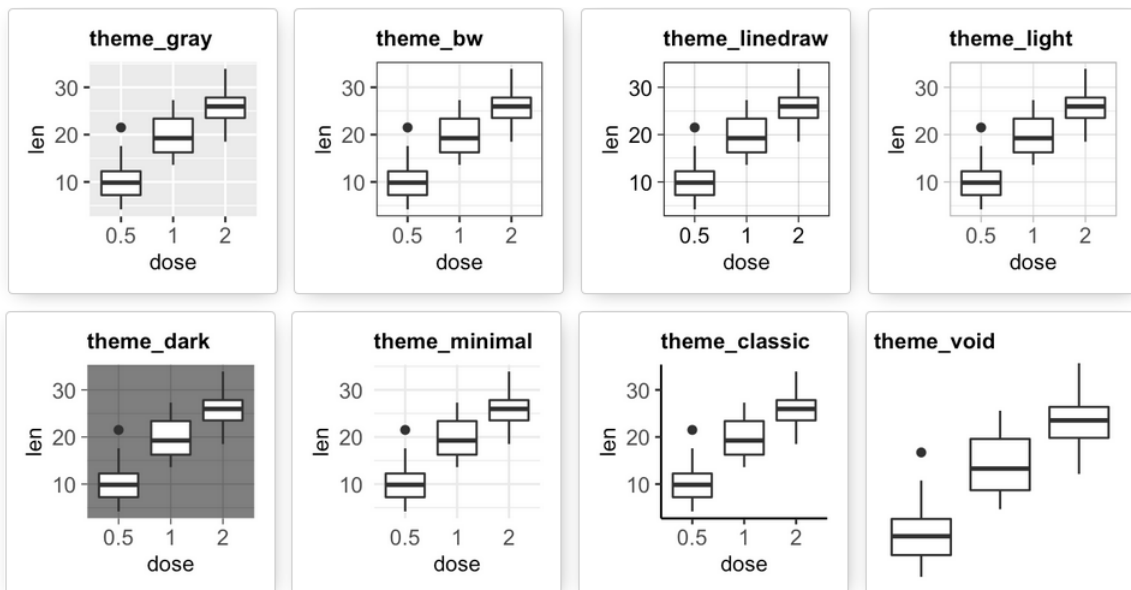
```
# Inversion axe des abscisses / axe des ordonnées
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Petal.Length,
      colour = Species) +
  geom_point() +
  coord_flip()
```



#### 4.2.2.6 Couche *theme*

Personnalise les composantes hors données du graphique : titres, étiquettes, polices, arrière-plan, grilles et légendes

- **Utiliser** un thème prédéfini



```
# Graphique avec Le theme Dark
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Petal.Length) +
  geom_point() +
  theme_dark()
```

On peut définir le thème pour toute la durée de la session avec `theme_set()`

```
# Theme Dark pour tous les plots
theme_set(theme_dark)

ggplot() + ...
```

- **Éditer** un thème, à l'aide de l'interface graphique du package [lggThemeAssist](#)

💡 Comment ça marche ?

1. Sélectionner le code du graphique
  2. Lancer la commande `ggThemeAssistAddin()`
  3. Effectuer les modifications souhaitées sur les différents éléments
- Après validation dans l'interface, le code s'actualise dans le script

**i** On peut également modifier un thème existant avec `theme_update()`

- **Créer** et **utiliser** son propre thème, pour personnaliser l'apparence des graphiques, en ajustant les divers éléments

```
# Création du thème
theme <- theme(element.name = element_function(),
               element.name = element_function())

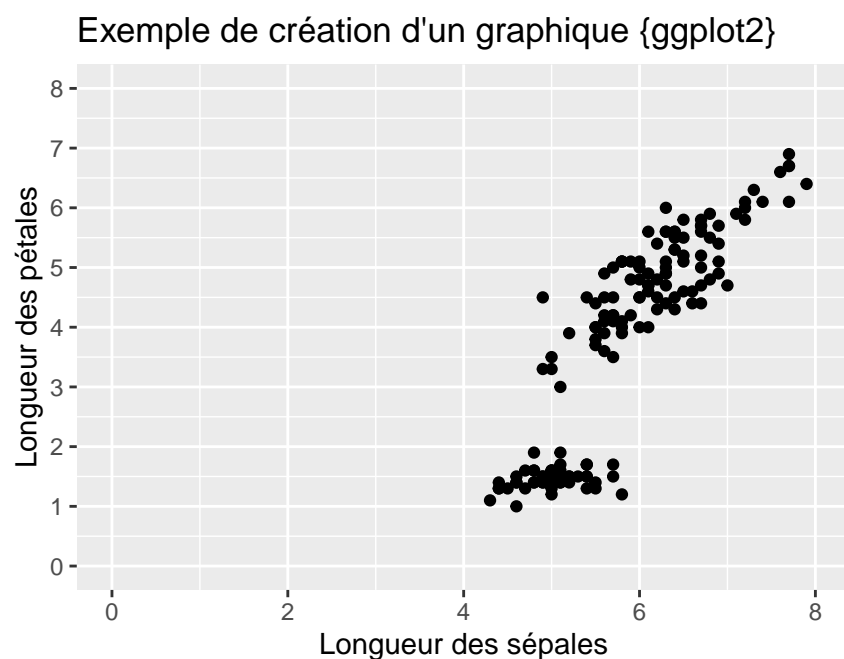
# Utilisation du thème
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Petal.Length) +
  geom_point() +
  theme
```

 Se reporter à la section *theme* de Wickham (2016a) pour plus de détails

#### 4.2.2.7 Couches additionnelles

- `scale_...()` : modifier les attributs liés
  - aux axes (échelles, limites, titre, format des chiffres par exemple)
  - aux données représentées (couleurs, etc.)
- `ggtitle()` : titre du graphique
- `xlab()` : intitulé de l'axe des abscisses
- `ylab()` : intitulé de l'axe des ordonnées

```
# On modifie les limites des axes (abscisses et ordonnées)
# et les titres
ggplot(data = iris) +
  aes(x = Sepal.Length, y = Petal.Length) +
  geom_point() +
  scale_x_continuous(limits = c(0, 8)) +
  scale_y_continuous(limits = c(0, 8),
                    breaks = seq(from = 0, to = 8, by = 1)) +
  ggtitle(label = "Exemple de création d'un graphique {ggplot2}") +
  xlab(label = "Longueur des sépales") +
  ylab(label = "Longueur des pétales")
```



### 4.2.3 Éléments de personnalisation des graphiques

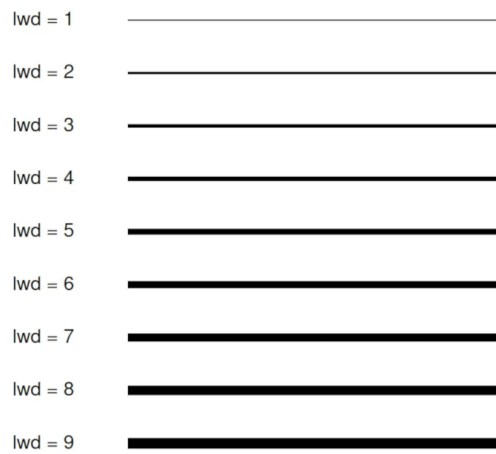


Figure 1: Les épaisseurs de lignes

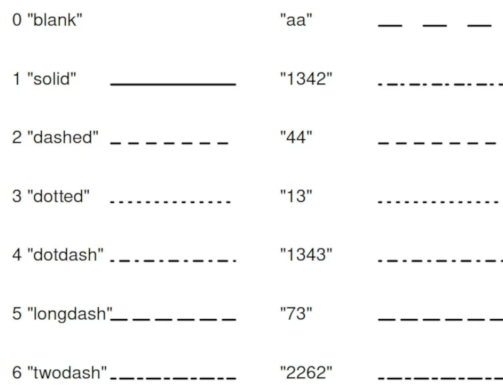


Figure 2: Les types de lignes

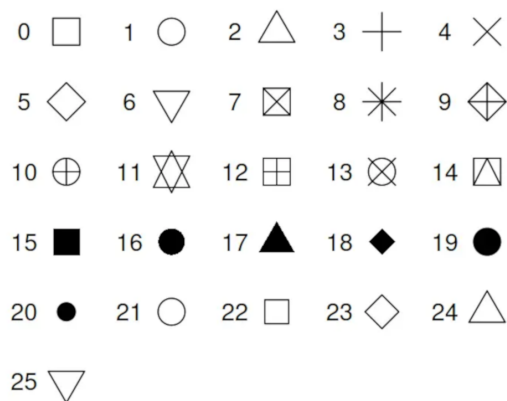


Figure 3: Les symboles de points

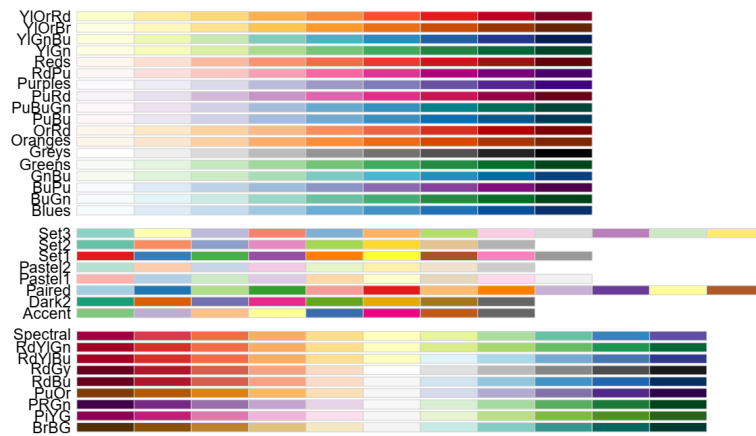


Figure 4: Les couleurs (package [RColorBrewer](#))

#### 💡 Aide pour le choix des couleurs

##### • Packages

- [paletteer](#) : recense des milliers de palettes (2759) issues de packages différents (75)
- [cols4all](#) : une application Shiny pour explorer les palettes, accessibles pour tous, y compris les personnes souffrant d'un déficit de vision des couleurs

```
cols4all::c4a_gui()
```

##### • Sites

- 🌐 informations sur une couleur : <https://chir.ag/projects/name-that-color/>
- 🌐 un générateur de palettes de couleurs : <https://coolers.co/>
- 🌐 palettes de couleurs prédéfinies : <https://flatuicolors.com/>
- 🌐 palettes de couleurs pour la cartographie : <http://colorbrewer2.org>

#### 4.2.4 Sauvegarde des graphiques ggplot

- À partir de l'onglet 'Plots' de RStudio, où s'affichent généralement les graphiques, cliquer sur le bouton 'Export' qui donne accès à trois options :
  - save as image
  - save as PDF
  - copy to clipboard
- Directement dans le script, en utilisant les fonctions `jpeg()`, `png()`, `bmp()`, `tiff()` ou `pdf()` du package `[grDevices]` qui permettent d'enregistrer les graphiques au format 'image' ou 'pdf'

```
png(filename = "fichier.png")
plot(...)
dev.off()
```

- On préférera utiliser la fonction `ggsave()` du package [ggplot2](#) à la fin de la création du graphique

```
ggplot(...) +
  ...
ggsave(filename = name.extension)
```

#### 4.2.5 Exemple de mise en forme d'un graphique

Voici un exemple de graphique personnalisé, représentant deux variables quantitatives et une qualitative, avec explication succincte des différentes fonctions utilisées et de leurs paramètres (non exhaustifs)

```

# Création d'un data frame avec des statistiques sur les 2 variables étudiées
iris |>
  dplyr::summarise(mean.SL = mean(Sepal.Length),
                   mean.PL = mean(Petal.Length),
                   max.SL = max(Sepal.Length),
                   max.PL = max(Petal.Length)) -> irisstats

# --- GRAPHIQUE
# Représentation des données
ggplot(data = iris) +
  aes(x = Sepal.Length,
       y = Petal.Length,
       color = Species) + # couleur des points différente selon les valeurs de Species
  geom_point(shape = 1, # type de symbole
             size = 1.5, # taille des symboles
             stroke = 1.5) + # épaisseur de la bordure des symboles

# Personnalisation des AXES
## Possibilité 1.
  scale_x_continuous(name = "axe des abscisses", # titre pour l'axe des x
                    limits = c(0, 10), # limites de l'axe des x
                    labels = scales::label_number(accuracy = 1)) + # pas de décimales
                                                                    # (0.1 pour 1 décimale)

## Possibilité 2.
  scale_y_continuous(limits = c(0, 10), # limites de l'axe des y
                    labels = scales::label_number(accuracy = 1)) + # pas de décimales
  ylab("axe des ordonnées") + # titre pour l'axe des y
  theme(axis.text.y = element_text(angle = 90)) + # rotation des étiquettes

# Ajout de TITRE
ggtitle(label = "Titre du graphique") + # ajout d'un titre

# Personnalisation du titre
  theme(plot.title = element_text(size = 15, # taille de police
                                  face = "bold", # type de police
                                  colour = "#004949", # couleur du titre
                                  hjust = 0.5)) + # alignement 0=gauche, 0.5=centré, 1=droite

# Ajout de LÉGENDE
  scale_color_manual(name = "Iris", # titre de la légende
                    labels = levels(iris$Species), # valeurs de la légende
                    values = hcl.colors(3, "viridis")) + # couleur des symboles de la légende

# Personnalisation de la légende
  theme(legend.position = "right", # position ("left", "top", "right", "bottom")
        legend.text = element_text(size = 10)) + # taille de la légende

# Ajout de LIGNE
## horizontale
  geom_hline(yintercept = max(iris$Petal.Length), # coupe l'axe des y à cette valeur
            linetype = "dashed", # type de ligne
            linewidth = 0.5, # épaisseur de ligne
            color = "grey50") + # couleur de ligne

## verticale
  geom_vline(xintercept = max(iris$Sepal.Length), # coupe l'axe des x à cette valeur
            linetype = "dashed", # type de ligne
            linewidth = 0.5, # épaisseur de ligne
            color = "grey50") + # couleur de ligne

# Ajout de POINT
  geom_point(data = irisstats,
            aes(x = mean.SL, # coordonnée x du point
               y = mean.PL, # coordonnée y du point
               colour = "#FF6DB6", # couleur du symbole
               shape = 15, # type de symbole
               size = 3) + # taille du symbole

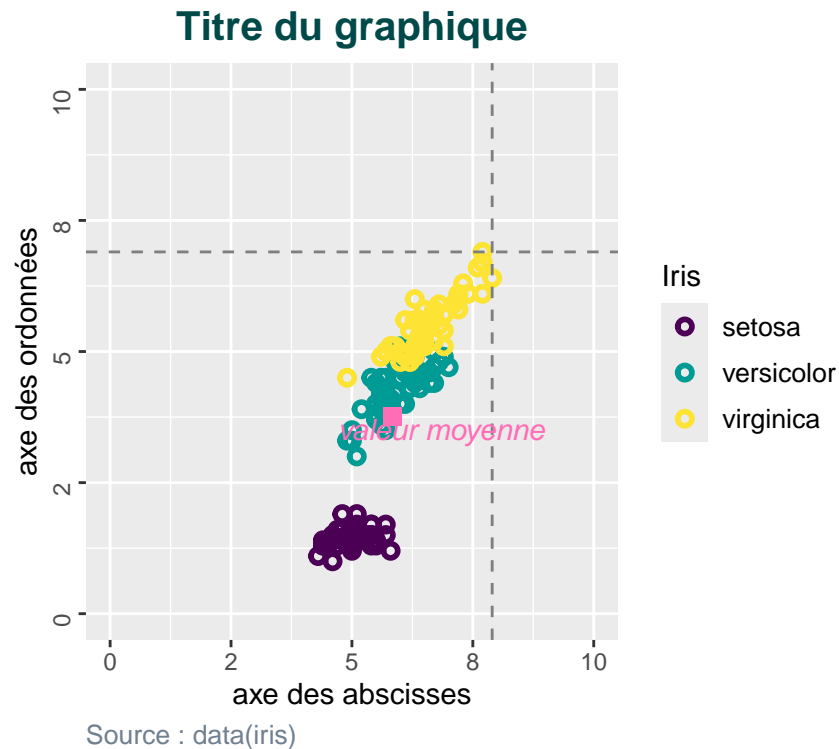
# Ajout de TEXTE

```

```

annotate("text",
  label = "valeur moyenne",      # texte à afficher
  x = irisstats$mean.SL + 1,      # emplacement horizontal du texte (centre)
  y = irisstats$mean.PL - 0.25,  # emplacement vertical du texte (centre)
  colour = "#FF6DB6",           # couleur du texte
  fontface = 'italic') +        # style du texte
# Ajout de SOURCE
labs(caption = "Source : data(iris)") + # texte à afficher
# Personnalisation de la source
theme(plot.caption = element_text(hjust = 0,           # alignement 0=gauche, 0.5=centré, 1=droite
  size = 10,      # taille du texte
  colour = "slategrey")) # couleur du texte

```



#### 4.2.6 Des aides pour faciliter la réalisation des graphiques **[ggplot2]**

- [lesquissel](#), une interface "clic bouton" pour créer des graphiques, avec la possibilité de récupérer les lignes de code
- [ggrepel](#) pour placer les étiquettes dans les graphiques sans qu'elles ne se superposent
- Des ressources en ligne
  - [ggplot2: Elegant Graphics for Data Analysis](#)
  - [R Graphics Cookbook, 2nd edition](#)
  - [Data Visualization & Information Design](#)
  - [ggplot2extensions](#) qui recense les extensions développées par les utilisateurs de R
  - [r-charts](#) pour trouver des exemples de code pour les graphiques (y compris graphiques en R de base) et les couleurs

## 4.3 Pour aller plus loin

### 4.3.1 Graphiques animés

- Les graphiques réalisés en base R ou avec le package [ggplot2](#) sont statiques
- Il est possible d'animer ces graphiques, pour des présentations ou des applications web
- Plusieurs packages pour réaliser des graphiques animés

#### 4.3.1.1 Package [plotly](#)

```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Petal.Length, color = Species) +  
  geom_point() -> p  
# Infobulle pour afficher Les valeurs de  
# Species, Sepal.Length et Petal.Length au survol d'un point  
plotly::ggplotly(p)
```

#### 4.3.1.2 Package [ggiraph](#)

```
ggplot(iris) +  
  ggiraph::geom_point_interactive(aes(x = Sepal.Length,  
                                       y = Petal.Length,  
                                       tooltip = Species,  
                                       data_id = row.names(iris))) -> p  
# Infobulle pour afficher La valeur de Species au clic sur un point  
ggiraph::girafe(ggobj = p)
```

#### 4.3.1.3 Package [gganimate](#)

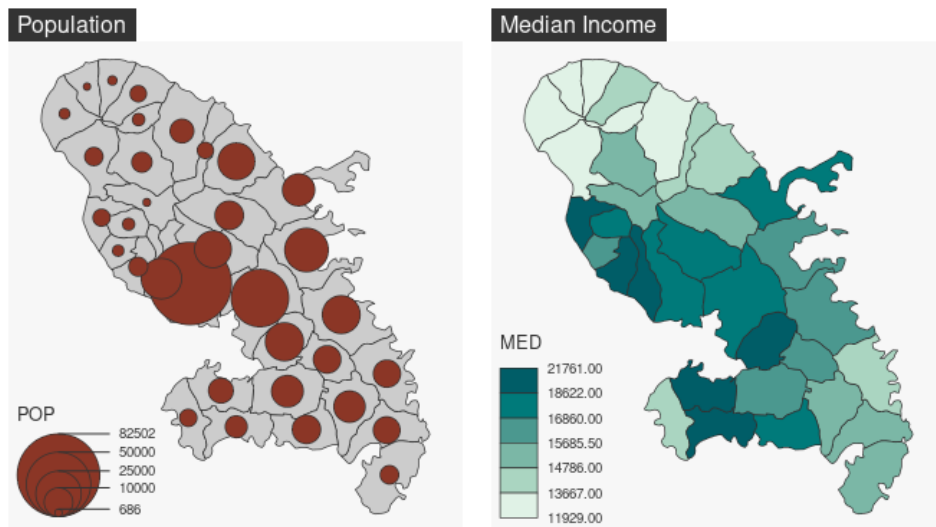
```
ggplot(iris) +  
  aes(x = Sepal.Length, y = Petal.Length) +  
  geom_point() +  
  # Affichage des points animé selon Les modalités de Species  
  gganimate::transition_states(states = Species, # état  
                               transition_length = 2, # durée de transition  
                               state_length = 3, # durée de pause  
                               wrap = TRUE) + # transition dernier - premier  
  labs(title = "{closest_state}") # affichage de La valeur de La modalité dans Le titre
```



### 4.3.2 Autres productions

- **Cartes**

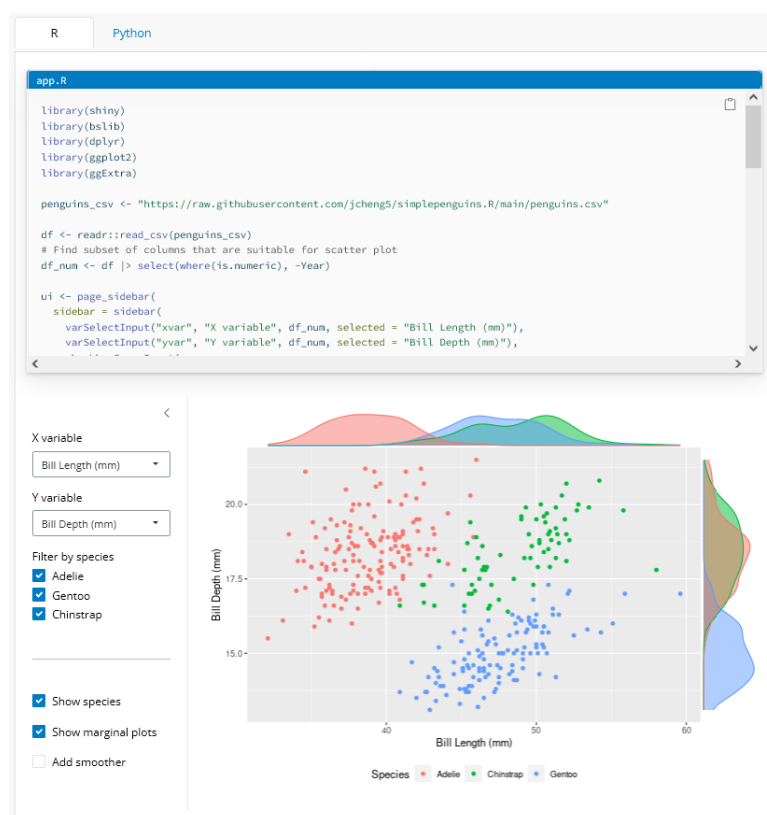
- package [ggmap](#)
- package [leaflet](#)
- packages [sf](#) & [maps](#) pour les traitements spatiaux et la cartographie "comme un pro"



Source : [mapsf](#)

- **Applications web**

- package [shiny](#) pour développer des applications interactives (d'analyse et de visualisation des données), dans une syntaxe R, pouvant être complétée par du HTML/CSS/JS



Source : [Shiny](#)









## 5 Pour conclure

### De nombreuses ressources pour faciliter l'usage de R

- Des packages et/ou [addins](#) (extensions)

- [\[addinslist\]](#) : parcourir et installer les *addins* de RStudio
- [\[questionr\]](#) : discrétiser une variable ; réordonner ou recoder un facteur
- [\[ReplaceInFiles\]](#) : rechercher et remplacer une valeur dans plusieurs fichiers
- [\[strcode\]](#) : structurer le code (sauts de section avec titre en en-tête)

- Des ressources en ligne

-  Sur le CRAN, des [manuels](#) et d'[autres contributions](#)
-  [R project](#)
-  [Rseek](#) : moteur de recherche dédié à R
-  Des [cheatsheets](#) (fiches synthétiques) pour certains packages (accessibles également via la page d'accueil de la fenêtre d'aide de RStudio)
-  [frrrenchies](#) : liste de ressources francophones
-  [StackOverflow](#)
-  [R journal](#)
-  [Journal of Statistical Software](#)

## 6 Bibliographie

### Références citées

- Orozco, V., Bontemps, C., Maigné, E., Piguët, V., Hofstetter, A., Lacroix, A., Levert, F., & Rousselle, J.-M. (2020). How To Make A Pie: Reproducible Research for Empirical Economics and Econometrics. *Journal of Economic Surveys*, 34(5), 11341169. <https://doi.org/10.1111/joes.12389>
- Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>
- Wickham, H., & Grommond, G. (2018). *R pour les data sciences : importer, classer, transformer, visualiser et modéliser les données*. Eyrolles. <https://r4ds.had.co.nz/>
- Wilkinson, L. (1999). *The Grammar of Graphics* (1st éd.). Springer New York, NY. <https://doi.org/10.1007/978-1-4757-3100-2>

### Packages

- Attali, D. (2023). *addinslist: Discover and Install Useful RStudio Addins*. <https://github.com/daattali/addinslist>
- Bache, S. M., & Wickham, H. (2022). *magrittr: A Forward-Pipe Operator for R*. <https://magrittr.tidyverse.org>
- Barbone, J. M., & Garbuszus, J. M. (2024). *openxlsx2: Read, Write and Edit xlsx Files*. <https://janmarvin.github.io/openxlsx2/>
- Barnier, J., Briatte, F., & Larmarange, J. (2023). *questionr: Functions to Make Surveys Processing Easier*. <https://juba.github.io/questionr/>
- Chang, W., Cheng, J., Allaire, J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., & Borges, B. (2024). *shiny: Web Application Framework for R*. <https://shiny.posit.co/>
- Cheng, J., Schloerke, B., Karambelkar, B., & Xie, Y. (2024). *leaflet: Create Interactive Web Maps with the JavaScript Leaflet Library*. <https://rstudio.github.io/leaflet/>
- Comtois, D. (2022). *summarytools: Tools to Quickly and Neatly Summarize Data*. <https://github.com/dcomtois/summarytools>
- Cui, B. (2024). *DataExplorer: Automate Data Exploration and Treatment*. <http://boxuancui.github.io/DataExplorer/>
- Dragulescu, A., & Arendt, C. (2020). *xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files*. <https://github.com/colearendt/xlsx>
- file., S. A. (2024). *paletteer: Comprehensive Collection of Color Palettes*. <https://github.com/EmilHvitfeldt/paletteer>
- Giraud, T. (2024). *mapsf: Thematic Cartography*. <https://riatelab.github.io/mapsf/>
- Gohel, D., & Skintzos, P. (2024). *ggiraph: Make ggplot2 Graphics Interactive*. <https://davidgohel.github.io/ggiraph/>
- Grommond, G., & Wickham, H. (2011). Dates and Times Made Easy with lubridate. *Journal of Statistical Software*, 40(3), 125. <https://www.jstatsoft.org/v40/i03/>
- Gross, C., & Ottolinger, P. (2016). *ggThemeAssist: Add-in to Customize ggplot2 Themes*. <https://github.com/calligross/ggthemeassist>
- Hester, J., Angly, F., Hyde, R., Chirico, M., Ren, K., Rosenstock, A., & Patil, I. (2024). *lintr: A Linter for R Code*. <https://github.com/r-lib/lintr>
- Kahle, D., & Wickham, H. (2013). ggmap: Spatial Visualization with ggplot2. *The R Journal*, 5(1), 144161. <https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf>
- Kahle, D., Wickham, H., & Jackson, S. (2023). *ggmap: Spatial Visualization with ggplot2*. <https://github.com/dkahle/ggmap>
- Kranz, S. (2024). *ReplaceInFiles: RStudio Addin to Find and Replace in Multiple Files*. <https://github.com/skranz/ReplaceInFiles>
- Lin, G. (2023). *reactable: Interactive Data Tables for R*. <https://glin.github.io/reactable/>
- Meyer, F., & Perrier, V. (2024a). *esquisse: Explore and Visualize Your Data Interactively*. <https://dreamrs.github.io/esquisse/>
- Meyer, F., & Perrier, V. (2024b). *viewxl: View data.frames in Excel*. <https://github.com/dreamRs/viewxl>
- Müller, K., & Walthert, L. (2024). *styler: Non-Invasive Pretty Printing of R Code*. <https://github.com/r-lib/styler>
- Müller, K., & Wickham, H. (2023). *tibble: Simple Data Frames*. <https://tibble.tidyverse.org/>
- Neuwirth, E. (2022). *RColorBrewer: ColorBrewer Palettes*. <https://CRAN.R-project.org/package=RColorBrewer>
- Ooms, J. (2024). *writexl: Export Data Frames to Excel xlsx Format*. <https://docs.ropensci.org/writexl/>
- Pebesma, E. (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1), 439446. <https://doi.org/10.32614/RJ-2018-009>

Pebesma, E. (2024). *sf: Simple Features for R*. <https://r-spatial.github.io/sf/>

Pebesma, E., & Bivand, R. (2023). *Spatial Data Science: With applications in R*. Chapman and Hall/CRC. <https://doi.org/10.1201/9780429459016>

Pedersen, T. L., & Robinson, D. (2024). *gganimate: A Grammar of Animated Graphics*. <https://gganimate.com>

R Core Team. (2024). *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, ...* <https://svn.r-project.org/R-packages/trunk/foreign/>

Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. Springer. <http://lmdvr.r-forge.r-project.org>

Sarkar, D. (2024). *lattice: Trellis Graphics for R*. <https://lattice.r-forge.r-project.org/>

Sievert, C. (2020). *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman; Hall/CRC. <https://plotly-r.com>

Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., & Despouy, P. (2024). *plotly: Create Interactive Web Graphics via plotly.js*. <https://plotly-r.com>

Sjoberg, D. D., Larmarange, J., Curry, M., Lavery, J., Whiting, K., & Zabor, E. C. (2023). *gtsummary: Presentation-Ready Data Summary and Analytic Result Tables*. <https://github.com/ddsjoberg/gtsummary>

Sjoberg, D. D., Whiting, K., Curry, M., Lavery, J. A., & Larmarange, J. (2021). Reproducible Summary Tables with the gtsummary Package. *The R Journal*, 13, 570580. <https://doi.org/10.32614/RJ-2021-053>

Slowikowski, K. (2024). *ggrepel: Automatically Position Non-Overlapping Text Labels with ggplot2*. <https://ggrepel.slowkow.com/>

Spinu, V., Grolemond, G., & Wickham, H. (2023). *lubridate: Make Dealing with Dates a Little Easier*. <https://lubridate.tidyverse.org>

Tennekes, M. (2024). *cols4all: Colors for all*. <https://mtennekes.github.io/cols4all/>

Tierney, N. (2017). visdat: Visualising Whole Data Frames. *JOSS*, 2(16), 355. <https://doi.org/10.21105/joss.00355>

Tierney, N. (2023). *visdat: Preliminary Visualisation of Data*. <https://docs.ropensci.org/visdat/>

Walthert, L. (2024). *strcode: Structure and abstract your code*. <https://github.com/lorenzwaltbert/strcode>

Waring, E., Quinn, M., McNamara, A., Arino de la Rubia, E., Zhu, H., & Ellis, S. (2022). *skimr: Compact and Flexible Summaries of Data*. <https://docs.ropensci.org/skimr/>

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>

Wickham, H. (2023a). *forcats: Tools for Working with Categorical Variables (Factors)*. <https://forcats.tidyverse.org/>

Wickham, H. (2023b). *stringr: Simple, Consistent Wrappers for Common String Operations*. <https://stringr.tidyverse.org>

Wickham, H., & Bryan, J. (2023). *readxl: Read Excel Files*. <https://readxl.tidyverse.org>

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., Dunnington, D., & van den Brand, T. (2024). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. <https://ggplot2.tidyverse.org>

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *dplyr: A Grammar of Data Manipulation*. <https://dplyr.tidyverse.org>

Wickham, H., & Henry, L. (2023). *purrr: Functional Programming Tools*. <https://purrr.tidyverse.org/>

Wickham, H., Hester, J., & Bryan, J. (2024). *readr: Read Rectangular Text Data*. <https://readr.tidyverse.org>

Wickham, H., Miller, E., & Smith, D. (2023). *haven: Import and Export SPSS, Stata and SAS Files*. <https://haven.tidyverse.org>

Wickham, H., Vaughan, D., & Girlich, M. (2024). *tidyr: Tidy Messy Data*. <https://tidyr.tidyverse.org>

## 7 Annexes. Rappels sur la visualisation graphique R standard

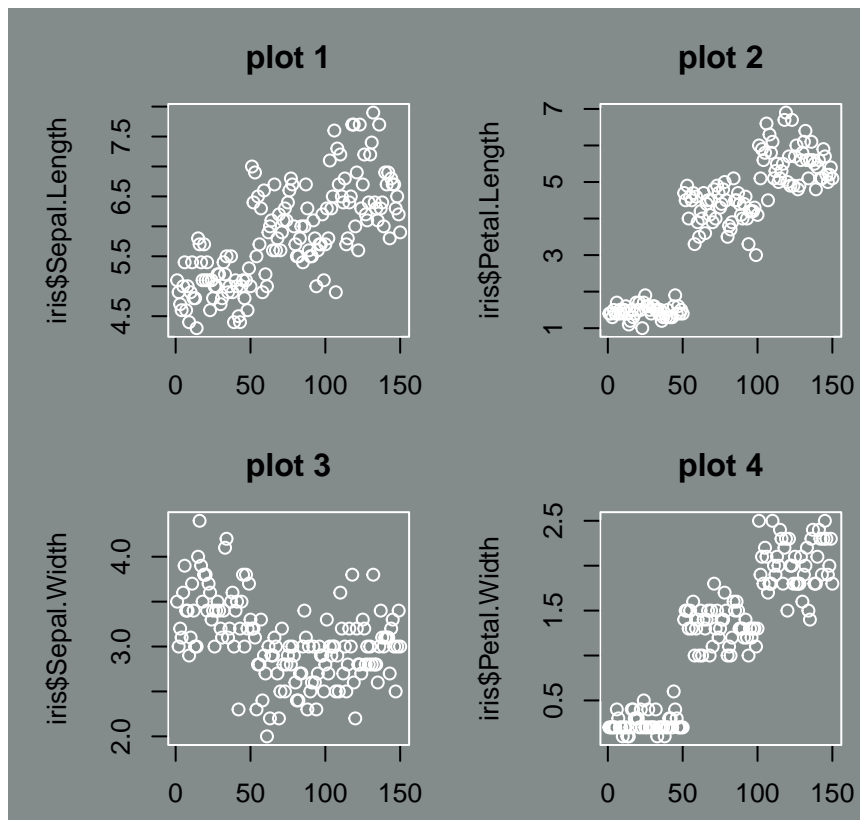
### 7.1 Création d'un graphique

En 3 étapes, à l'aide des packages `[graphics]` et `[grDevices]`

#### 1. Configuration des paramètres graphiques (optionnelle)

- avec `par()` : pour définir ou interroger les paramètres de la fenêtre graphique (marges, couleurs, découpage de la fenêtre, etc.)
- avec `layout()` : pour personnaliser la disposition des graphiques
- Fonction `par()`

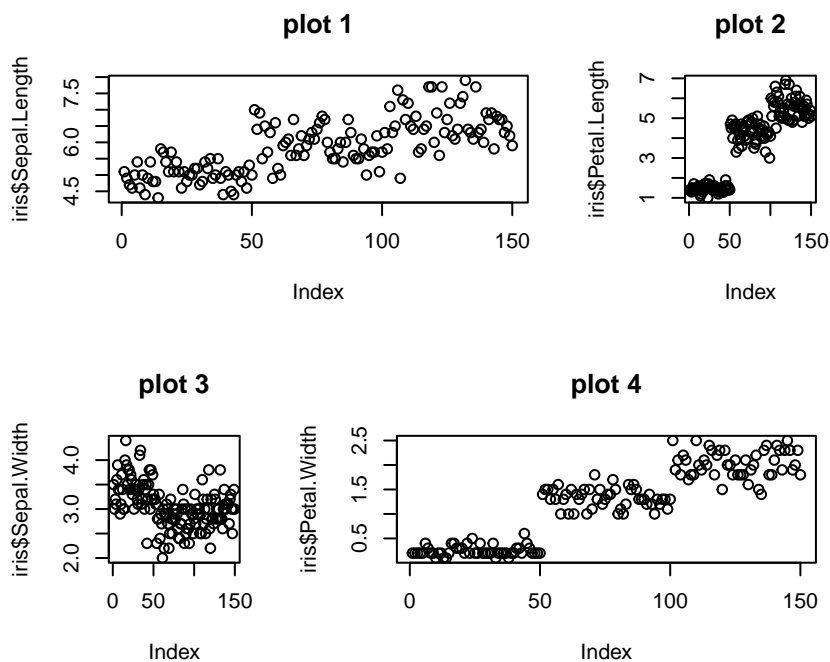
```
# Exemple pour 4 graphiques dans une fenêtre
par(mfrow = c(2, 2),          # Fenêtre découpée en 2 lignes et 2 colonnes
    bg = "azure4",            # Couleur de fond
    col = "white",            # Couleur du graphique
    mar = c(b = 2.5, l = 5,   # Marges : bottom, left
            t = 3, r = 1))    # top, right
plot(iris$Sepal.Length, main = "plot 1")
plot(iris$Petal.Length, main = "plot 2")
plot(iris$Sepal.Width, main = "plot 3")
plot(iris$Petal.Width, main = "plot 4")
```



- Fonction `layout()`

```
# Découpage de la fenêtre graphique
layout(matrix(nrow = 4,          # Fenêtre découpée en 4 lignes
             ncol = 3,          # et 3 colonnes
             data = c(1, 1, 2,   # places occupées par chaque graphique
                     1, 1, 2,
                     3, 4, 4,   # par ex., plot 3 sur col. 1 et ligne 3
                     3, 4, 4),  # et sur col. 1 et ligne 4
             byrow = TRUE))      # matrice remplie par ligne

plot(iris$Sepal.Length, main = "plot 1")
plot(iris$Petal.Length, main = "plot 2")
plot(iris$Sepal.Width, main = "plot 3")
plot(iris$Petal.Width, main = "plot 4")
```



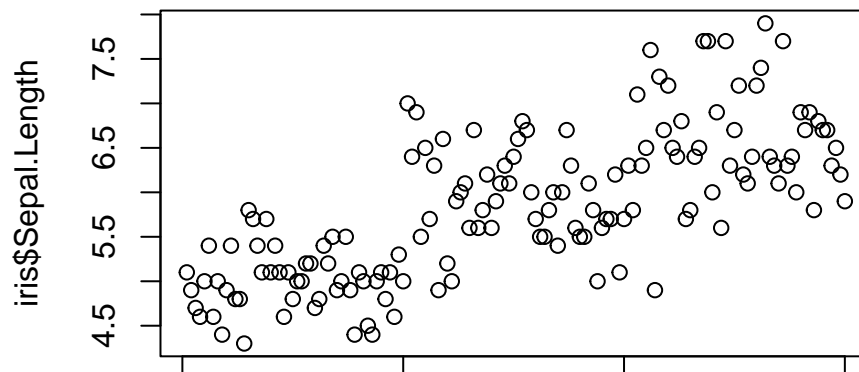
## 2. Initialisation du graphique

- avec la fonction de base `plot(x = VarX, y = VarY)` ou sous forme de formule `plot(varY ~ varX)`
- avec les fonctions pour un type spécifique
  - `hist()` : histogramme des fréquences d'une variable numérique
  - `boxplot()` : boîte à moustaches pour la distribution d'une variable continue
  - `barplot()` : diagramme en barres pour la fréquence des valeurs d'une variable catégorielle
  - `pie()` : diagramme circulaire/graphique en secteurs/camembert pour la fréquence des valeurs d'une variable catégorielle **non ordonnée**

### 7.1.1 Fonction plot()

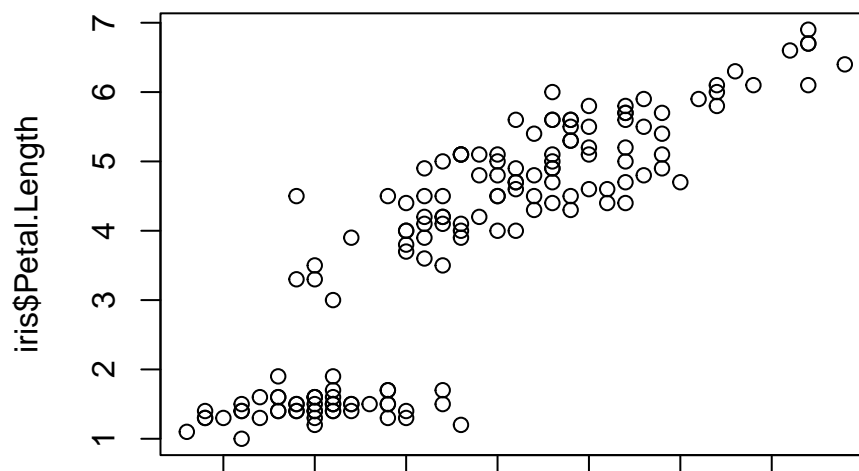
1 variable QN - Nuage de points ordonnés

```
plot(iris$Sepal.Length)
```



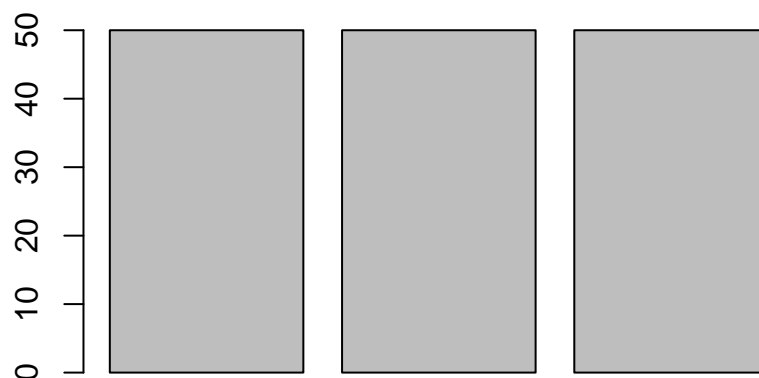
2 variables QN - Nuage de points

```
plot(x = iris$Sepal.Length,  
     y = iris$Petal.Length)
```



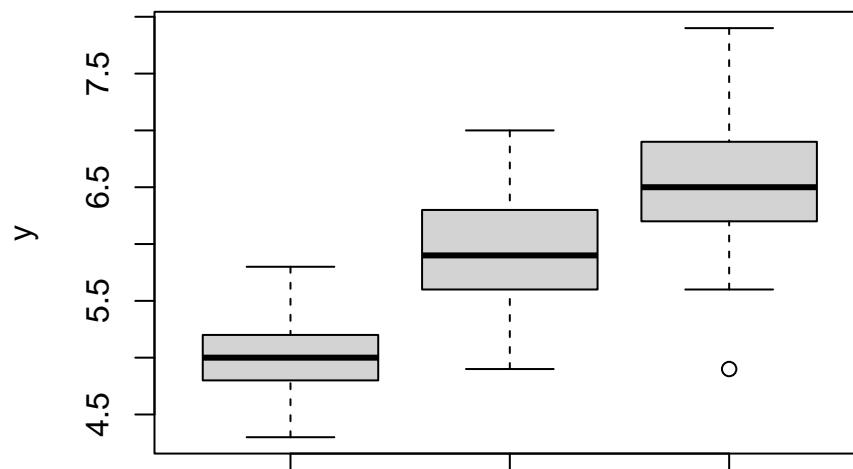
1 variable QL - Diagramme en barres

```
plot(iris$Species)
```



### 1 variable QN & 1 variable QL - **Boxplot**

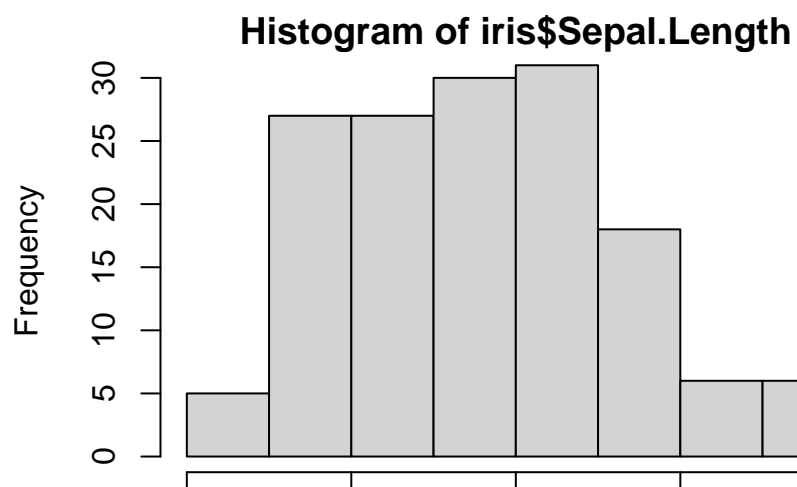
```
plot(x = iris$Species,  
     y = iris$Sepal.Length)
```



### 7.1.2 Fonctions spécifiques

#### Histogramme de distribution

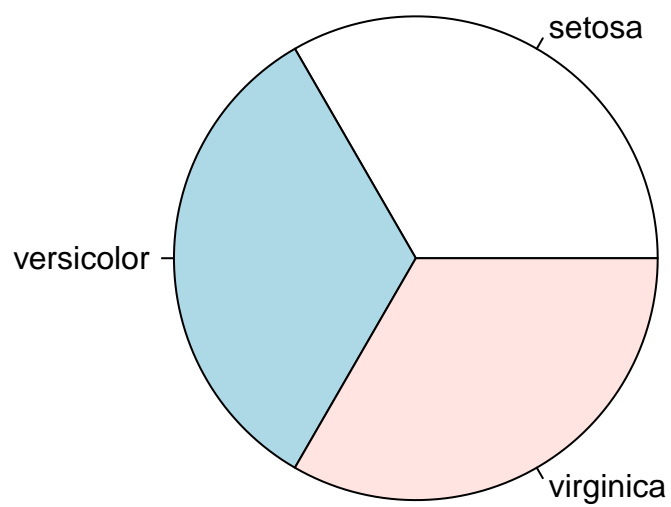
```
hist(iris$Sepal.Length)
```





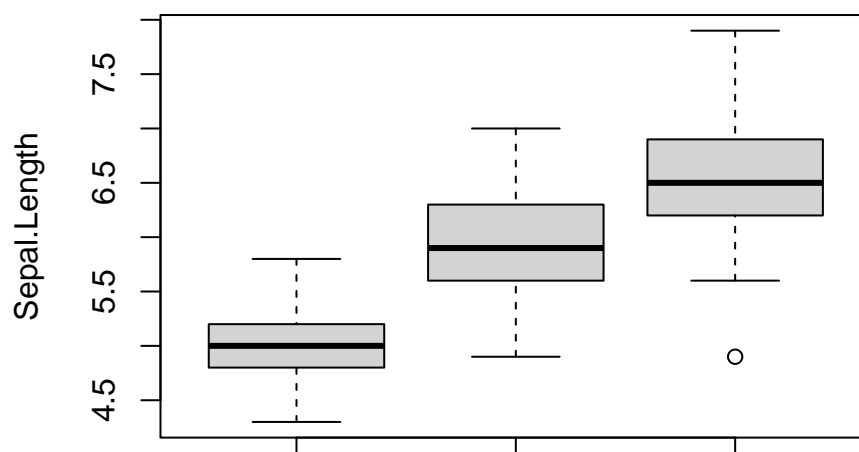
## Camembert

```
pie(table(iris$Species))
```



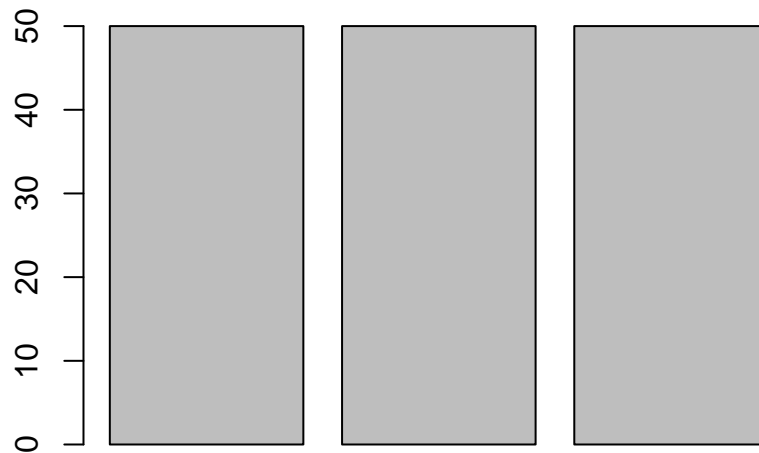
## Boxplot

```
boxplot(Sepal.Length ~ Species, data = iris)
```



## Diagramme en barres

```
barplot(table(iris$Species))
```



### 3. Ajout d'éléments au graphique initialisé à l'étape précédente

- `points()` : des points
- `lines()` : des points reliés par une ligne
- `abline()` : une ligne droite horizontale qui traverse la zone graphique
- `legend()` : une légende
- `axis()` : un axe
- `text()` : du texte dans la zone graphique
- `mtext()` : du texte dans les marges de la figure ou de la fenêtre graphique
- `title()` : du texte dans le titre, le sous-titre, les légendes des axes
- etc.

## 7.2 Exemple de mise en forme d'un graphique

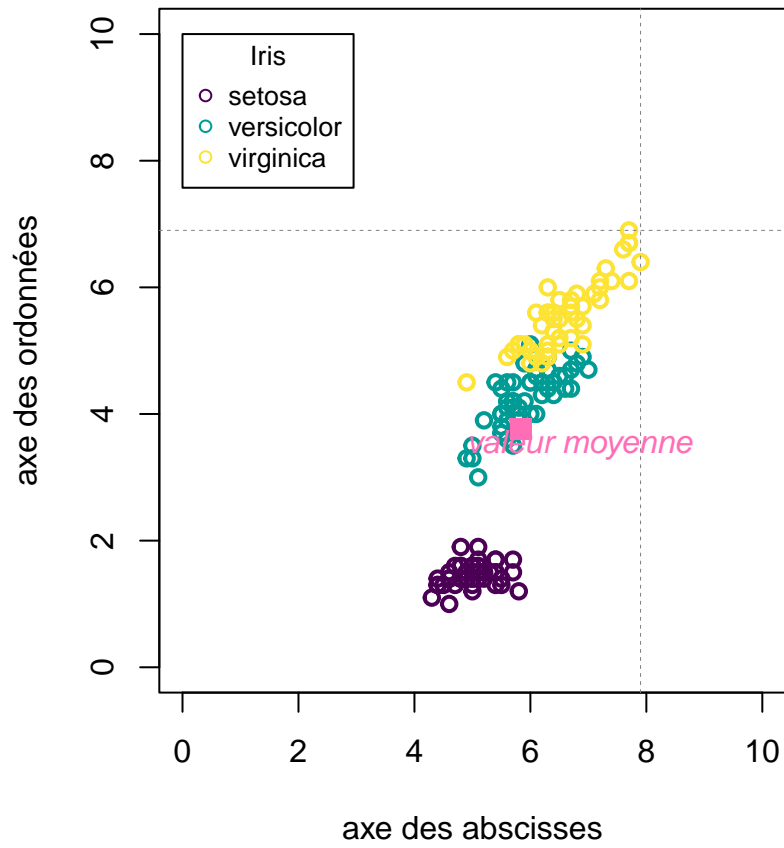
Refaisons le graphique personnalisé de la section [Section 4.2.5](#), avec les commandes de base de R

```

# Paramétrage des marges (c(bas, gauche, haut, droite))
# par défaut par(mar = c(5, 4, 4, 2) + 0.1)
par(mar = c(8, 4, 4, 2) + 0.1) # valeurs en lignes de la marge
# Choix de la palette couleurs
palette(hcl.colors(3, "viridis"))
# --- GRAPHIQUE
# Représentation des données
plot(x = iris$Sepal.Length,
     y = iris$Petal.Length,
     col = iris$Species,      # symboles colorés selon les modalités de la variable Species
     # col = "#490092",      # OU couleur unique des symboles
     type = "p",             # "p" (points), "l" (lignes), "b" ou "o" (les 2)
     pch = 1,                # type de symboles (valeur entre 0 et 25)
     cex = 1,                # taille des symboles
     lwd = 2,                # épaisseur de la bordure des symboles (valeur entre 1 et 9)
     xlim = c(0, 10),        # limites de l'axe des x
     ylim = c(0, 10),        # limites de l'axe des y
     xlab = "axe des abscisses", # titre pour l'axe des x
     ylab = "axe des ordonnées") # titre pour l'axe des y
# Ajout de TITRE
title(main = "Titre du graphique",
      cex.main = 1.2,        # taille de la police
      font.main = 2,          # 1=plain, 2=bold, 3=italic, 4=bold it., 5=symbol
      col.main = "#004949")   # couleur
# Ajout de LÉGENDE
legend(x = 0,                # emplacement de la légende (coordonnée x)
       y = 10,               # emplacement de la légende (coordonnée y)
       title = "Iris",        # titre de la légende
       legend = levels(iris$Species), # valeurs de la légende
       col = hcl.colors(3, "viridis"), # couleur des symboles de la légende
       cex = 0.8,            # taille de la légende
       pch = 1)              # symbole de la légende
# Ajout de LIGNE
abline(v = max(iris$Sepal.Length), # coupe l'axe des x à cette valeur
       h = max(iris$Petal.Length), # coupe l'axe des y à cette valeur
       lty = 2,                  # type de ligne
       lwd = 0.5,                # épaisseur de ligne
       col = "grey50")           # couleur de ligne
# Ajout de POINT
points(x = mean(iris$Sepal.Length), # coordonnée x du point
       y = mean(iris$Petal.Length), # coordonnée y du point
       col = "#FF6DB6",             # couleur du symbole
       pch = 15,                    # type de symbole
       cex = 1.5)                   # taille du symbole
# Ajout de TEXTE
text("valeur moyenne",            # texte à afficher
     x = mean(iris$Sepal.Length) + 1, # emplacement horizontal du texte (centre)
     y = mean(iris$Petal.Length) - 0.25, # emplacement vertical du texte (centre)
     col = "#FF6DB6",              # couleur du texte
     font = 3)                     # style du texte
# Ajout de SOURCE
mtext(text = "Source : data(iris)", # texte à afficher
      side = 1,                    # emplacement : 1=bottom, 2=left, 3=top, 4=right
      line = 5,                    # positionnement du texte
      adj = 0,                     # alignement : 0=gauche, 0.5=centré, 1=droite
      cex = 0.9,                   # taille de police
      col = "slategrey")           # couleur de police

```

## Titre du graphique



Source : data(iris)

7.3 Rappel sur les marges

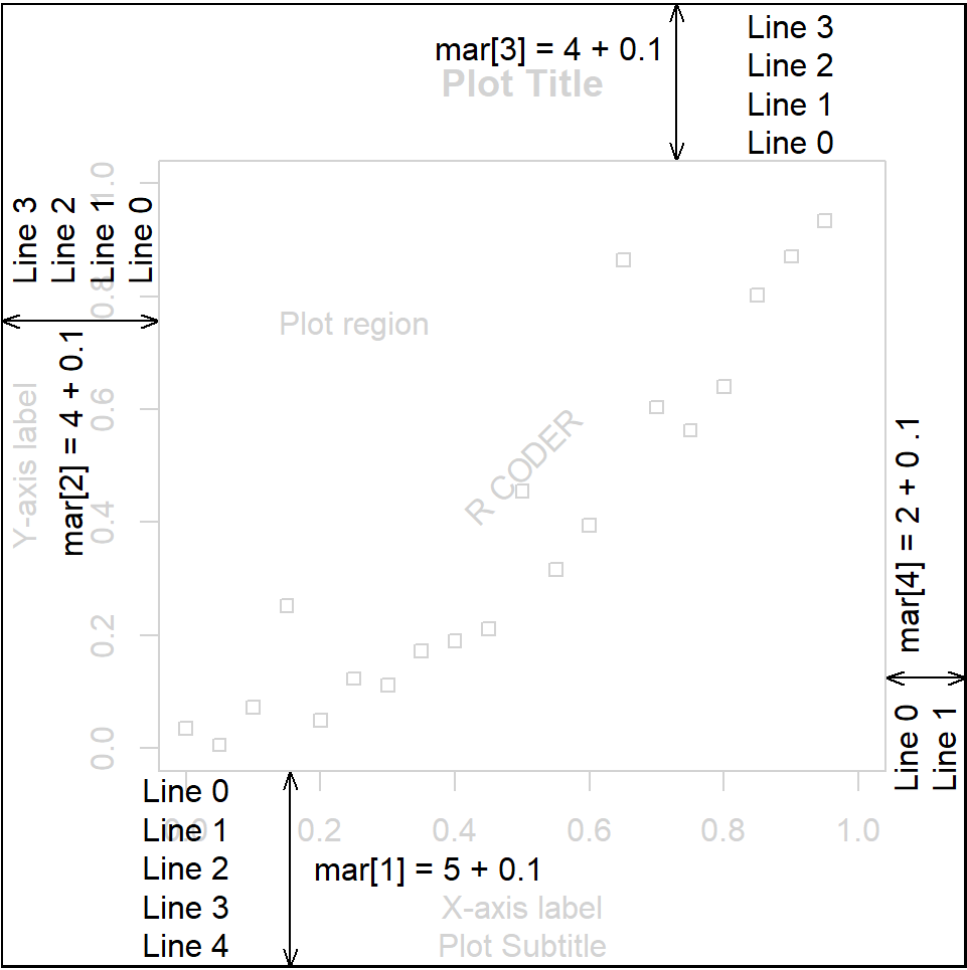


Figure 5: Marges des graphiques

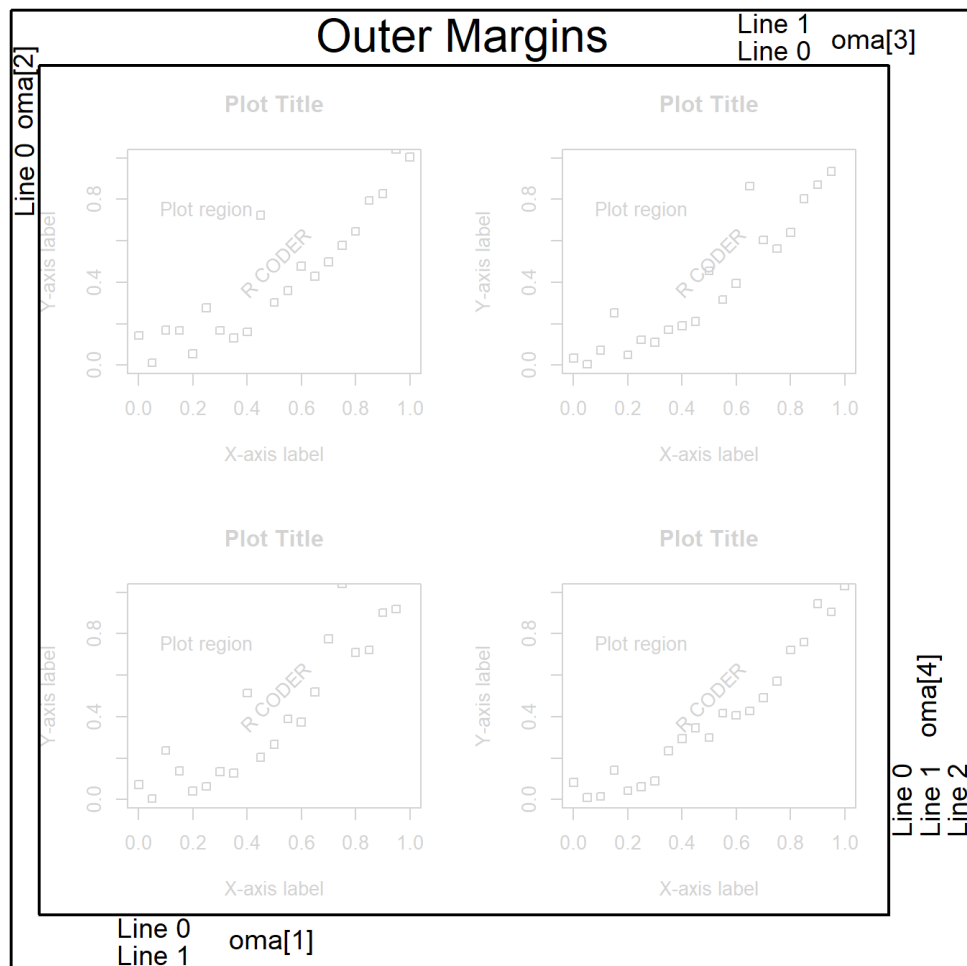


Figure 6: Marges extérieures

Source : <https://r-charts.com/base-r/margins/>