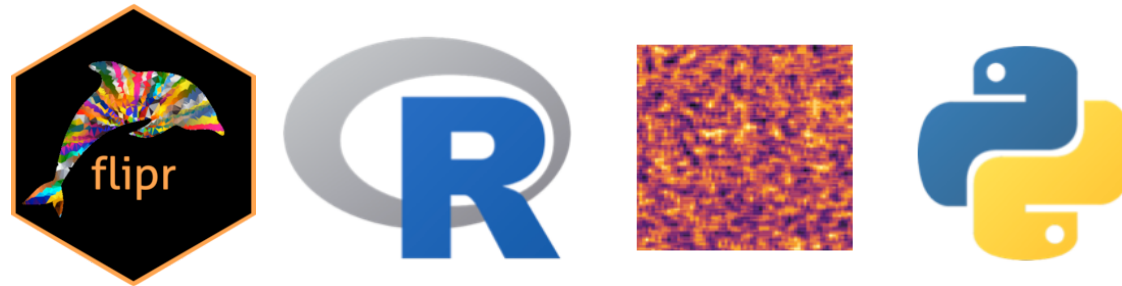


# Stage de Master 1

## Ingénierie statistique

Bonnes pratiques pour le développement collaboratif de logiciel

Application au développement de packages R autour de l'inférence statistique par permutation avec implémentation de tests unitaires



**Juliette Chiapello**

Encadrant de stage : **Aymeric Stamm**

Juin - Juillet 2021

# Sommaire

Création d'un  
package en R  
et mise en  
ligne sur  
Github

Création d'une  
application  
avec R :  
Shiny App

Les  
procédures  
pour tester  
le code

Les tests  
unitaires  
appliqués au  
package  
*flipr*

I

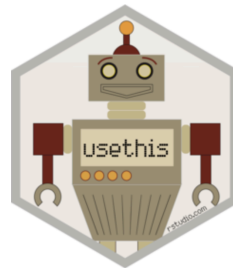
II

III

IV

## 1. Motivation

- Un projet est déjà organisé d'une manière proche de celle d'un package
- Les fonctions peuvent être appelées en chargeant la librairie
- Le package peut être partagé à d'autres utilisateurs
- C'est simple !



## 2.1 . En pratique - Création du package

### Libraries

```
library(devtools)
library(testthat)
```

OR

For personal start-up configuration with devtools and testthat, write in your .Rprofile startup file

```
if (interactive()) {
  suppressMessages(require(devtools))
}
```

Add also testthat

For details see : <https://r-pkgs.org/>

### FIRST SETUP (Do it once)

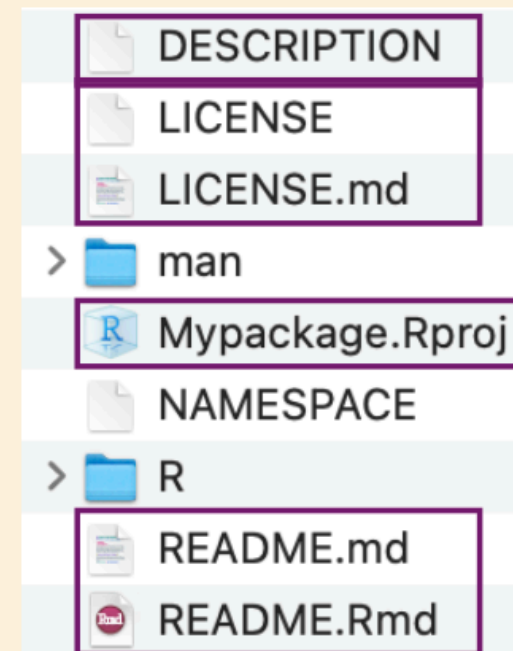
- Create your package :  
`create_package("path/Mypackage")`
- `Mypackage.Rproj` session is opened 1
- Edit the `DESCRIPTION` file : 2
  - Make yourself the author
  - Write some descriptive text in the TITLE and DESCRIPTION fields
- Choose a license (see <https://r-pkgs.org/license.html> for details) :

Ex : `use_mit_licence("Jane Doe")` (Write your name and surname) 3

- Initialize a `README.md` (and fill the fields)

(Optional if your package is only for you without using GitHub)

```
use_readme_rmd()
build_readme()
```

4


## 2.2 . En pratique - Développement du package

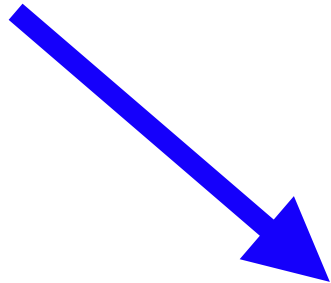
Add a function

```
use_r("Afunction")
```

The file `Afunction.R`  
is created

## 2.2 . En pratique - Développement du package

Add a function  
`use_r("Afunction")`  
The file `Afunction.R`  
is created






Write the function in the created  
file `Afunction.R`

If you need to use an fonction from another  
package to write your actual function, run :

```
use_package("package_name")  
package_name::thefunctionyouneed()
```

```
load_all()
```

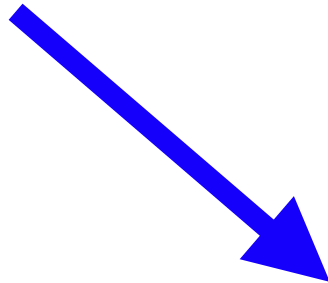
Shortcuts : CTRL + SHIFT + L    
CMD + SHIFT + L 

## 2.2 . En pratique - Développement du package

Add a function

```
use_r("Afunction")
```

The file **Afunction.R**  
is created






Write the function in the created  
file **Afunction.R**

If you need to use an fonction from another  
package to write your actual function, run :

```
use_package("package_name")  
package_name::thefunctionyouneed()
```

```
load_all()
```

Shortcuts : CTRL + SHIFT + L    
CMD + SHIFT + L 




Write the documentation :

Place the cursor somewhere in the  
function and then click on :

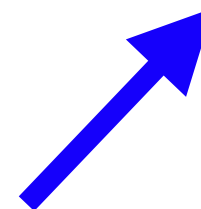
**Code > Insert Roxygen Skeleton**

Fill the fields. Then run :

```
document()
```

Shortcuts : CTRL + SHIFT + D    
CMD + SHIFT + D 

```
load_all()
```

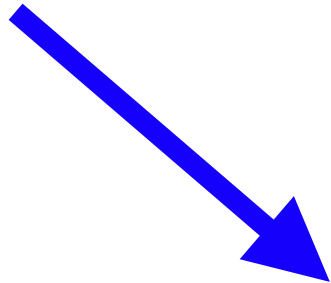


## 2.2 . En pratique - Développement du package

Add a function

```
use_r("Afunction")
```

The file **Afunction.R**  
is created







Write the function in the created  
file **Afunction.R**

If you need to use an fonction from another  
package to write your actual function, run :

```
use_package("package_name")  
package_name::thefunctionyouneed()
```

```
load_all()
```

Shortcuts : CTRL + SHIFT + L    
CMD + SHIFT + L  





Write the documentation :

Place the cursor somewhere in the  
function and then click on :

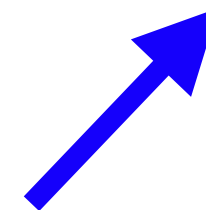
**Code > Insert Roxygen Skeleton**

Fill the fields. Then run :

```
document()
```

Shortcuts : CTRL + SHIFT + D    
CMD + SHIFT + D  

```
load_all()
```



Test your code








## 2.2 . En pratique - Développement du package

### `check()`

- ➡ It checks that the hole package works !
- ➡ You must correct errors/warnings/notes if needed

Shortcuts : CTRL + SHIFT + E    
CMD + SHIFT + E 

Run `install()` if you want :

- ➡ it installs your package on your computer and you can run `library(Mypackage)` to access your package whenever you want !

## 3.1 . Git et Github : utilisation générale

### Git

Version control system  
Local

#### Terminal

```
git config --global user.name "John Doe"  
git config --global user.email johndoe@example.com  
  
cd path/Monpackage  
git init  
git add .  
git commit -m «commentaire»
```

### Github

Sauvegarde en ligne  
Collaboration

#### Sur GitHub

Repository → New  
Copier les lignes du deuxième encart  
dans le terminal

`git push`  
Pour mettre en ligne

## 3.2 . Github : ajout de badges



```
use_github_action_check_standard()  
use_github_actions_badge(name = "R-CMD-check")
```



```
use_coverage()  
use_github_action("test-coverage")  
use_github_actions_badge(name = « test-coverage" )
```

Go on [codecov.io](https://codecov.io)

—> create an account

—> Go in Repos —> Not yet setup

```
covr::codecov(token = 'PASTE CLIPBOARD')
```



## 1. Principe d'une Shiny App

<https://mastering-shiny.org>

```
library(shiny)

ui <- fluidPage(
  # front end interface
)

server <- function(input, output, session) {
  # back end logic
}

shinyApp(ui, server)
```

### Exemple

```
ui <- fluidPage(
  textInput("name", "What's your name?"),
  textOutput("greeting")
)

server <- function(input, output, session) {
  output$greeting <- renderText({
    paste0("Hello ", input$name, "!")
  })
}
```

## 2. La réactivité

- Pour mieux maîtriser ce qui est réactualisé
- Pour accélérer les applications

(Sans la réactivité, tout est constamment réactualisé)



## 1. Classification

Functional testing

Unit tests,

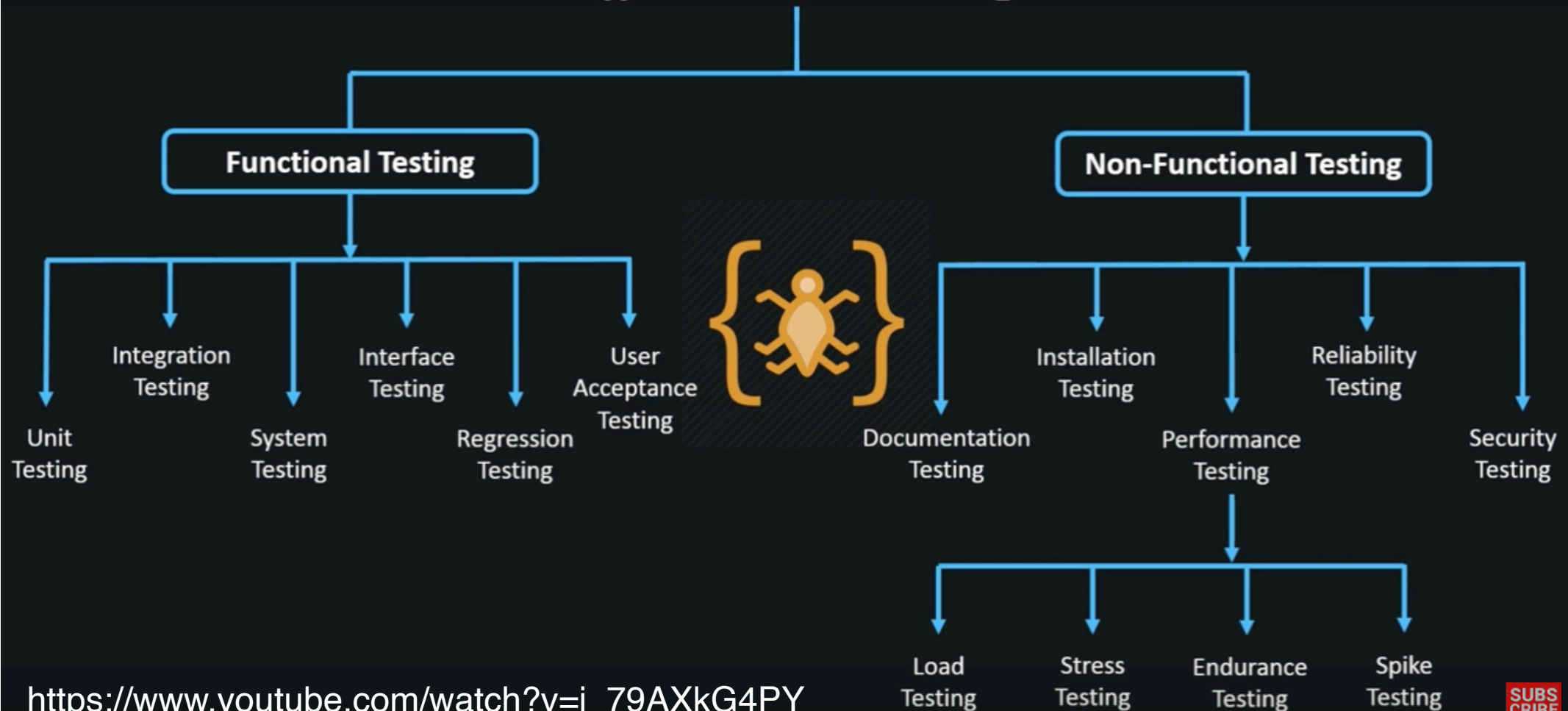
...

Non-functional testing

Functional Testing vs Non-Functional Testing | Software Testing Training | Edureka



## Types of Software Testing



[https://www.youtube.com/watch?v=j\\_79AXkG4PY](https://www.youtube.com/watch?v=j_79AXkG4PY)



## 2.1 . Les tests unitaires : motivation

- Vérification du bon fonctionnement du code. cf fonction direction —> le code est plus robuste (cas (2,2))
- Pouvoir remanier le code (refactoring)
- Coder d'un point de vue utilisateur
- Les tests correspondent à la documentation (il est facile de comprendre le code avec les tests)
- Gagner du temps



## 2.2 . Les tests unitaires : en pratique



```
test_that("abs function works for a negative number", {  
  #ARRANGE  
  negative ← -5  
  #ACT  
  actual ← abs(negative)  
  #ASSERT  
  expected ← 5  
  expect_equal(actual, expected)  
})
```

+ Snapshots et anomaly tests

## 2.2 . Les tests unitaires : en pratique



Create a unit tests file



```
use_testthat()  
use_test("Afunction")
```

Write unit tests in the created file

```
test-Afunction.R
```

Check all unit tests with `test()`

Or use a shortcut :

CRTL + SHIFT + T  

CMD + SHIFT + T 

```
i Loading randomfields  
i Testing randomfields  
✓ | OK F W S | Context  
✓ | 8         | direction  
✓ | 3         | iswholenumber [0.2 s]  
✓ | 6         | long_shaped_matrix  
✓ | 17        | moving_average [0.1 s]  
✓ | 2         | plot_matrix  
✓ | 9         | variance
```

== Results ==

Duration: 0.6 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 45 ]

## 2.3 . Les tests unitaires : propriétés

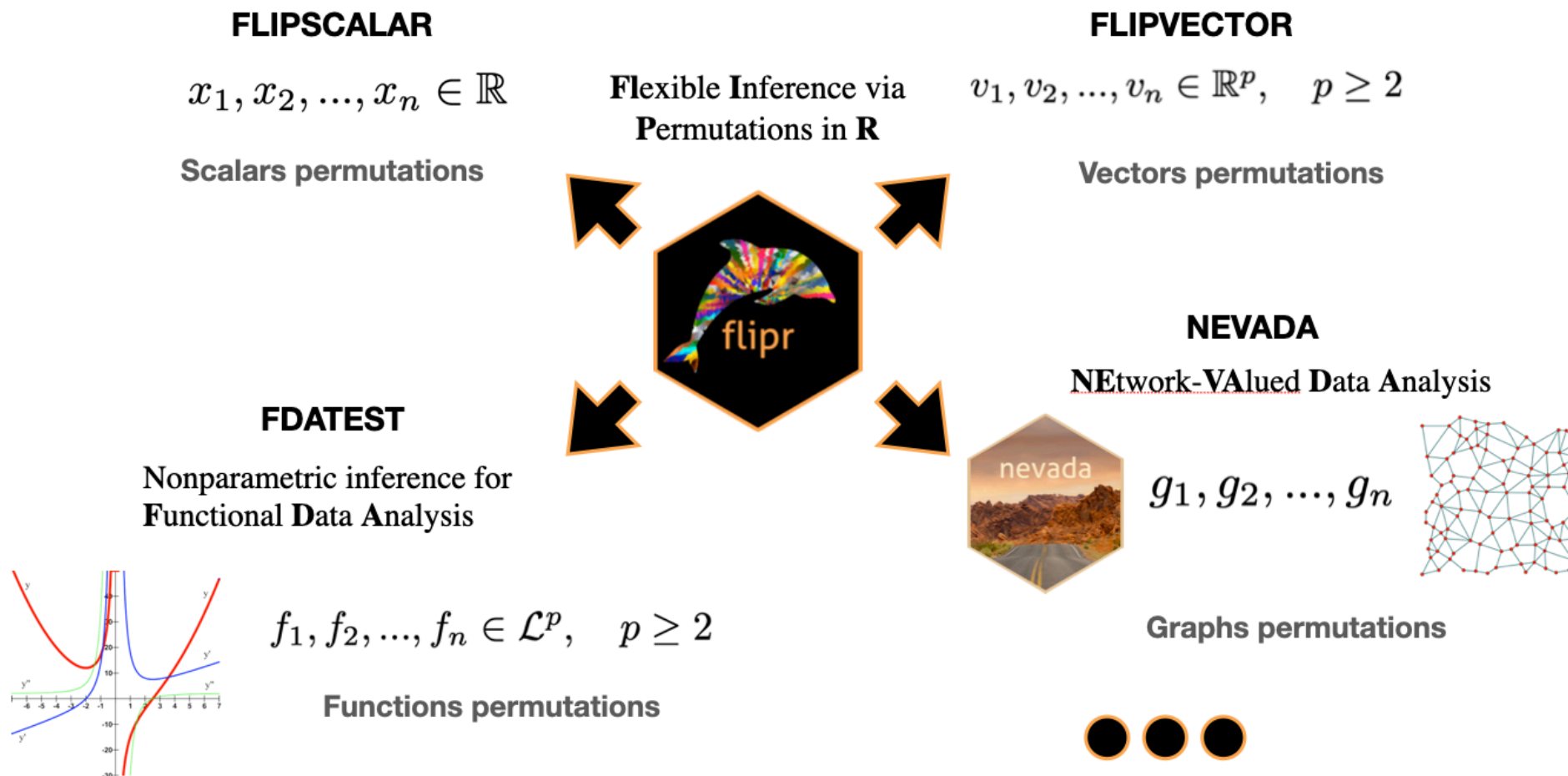
- Isolés les uns des autres (sinon problème de Flakiness)
- Pas de base de temps variable. Pas d'aléatoire (fixer un noyau)
- Rapides (problématique spécifique pour les datas sciences)
- Sensibles (il doivent échouer s'il y a un problème)
- Avoir les setup en interne (cf flipr)
- Spécifiques (remarque : ggplot2)

## 2.4 . Les tests unitaires : évaluer la qualité des tests

- Avec le pourcentage de couverture
- Avec les tests de mutation

## 2.5 . Les tests unitaires : le Test Driven Development

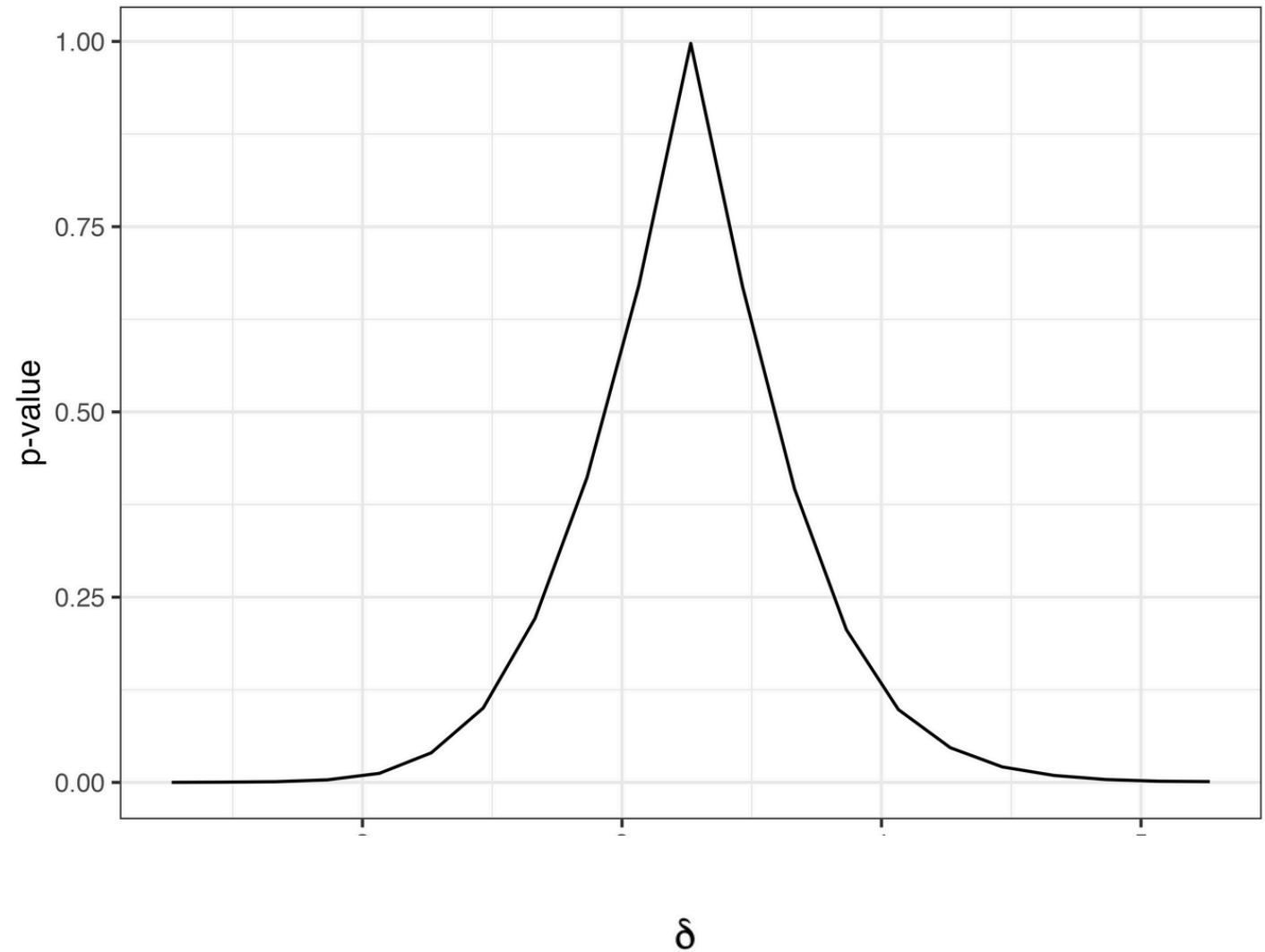
- Ecrire le test d'abord
- Le faire échouer
- Ecrire le minimum de code pour qu'il n'échoue plus
- Refactor

1. Le package *flipr*

## 1. Le package *flipr*

```
n <- 15  
x1 <- rnorm(n = n, mean = 0, sd = 1)  
x2 <- rnorm(n = n, mean = 1, sd = 1)
```

$$H_0 : \delta = \mu_2 - \mu_1 = 1$$



## 1. Le package *flipr*

```
n <- 15  
x1 <- rnorm(n = n, mean = 0, sd = 1)  
x2 <- rnorm(n = n, mean = 1, sd = 4)
```

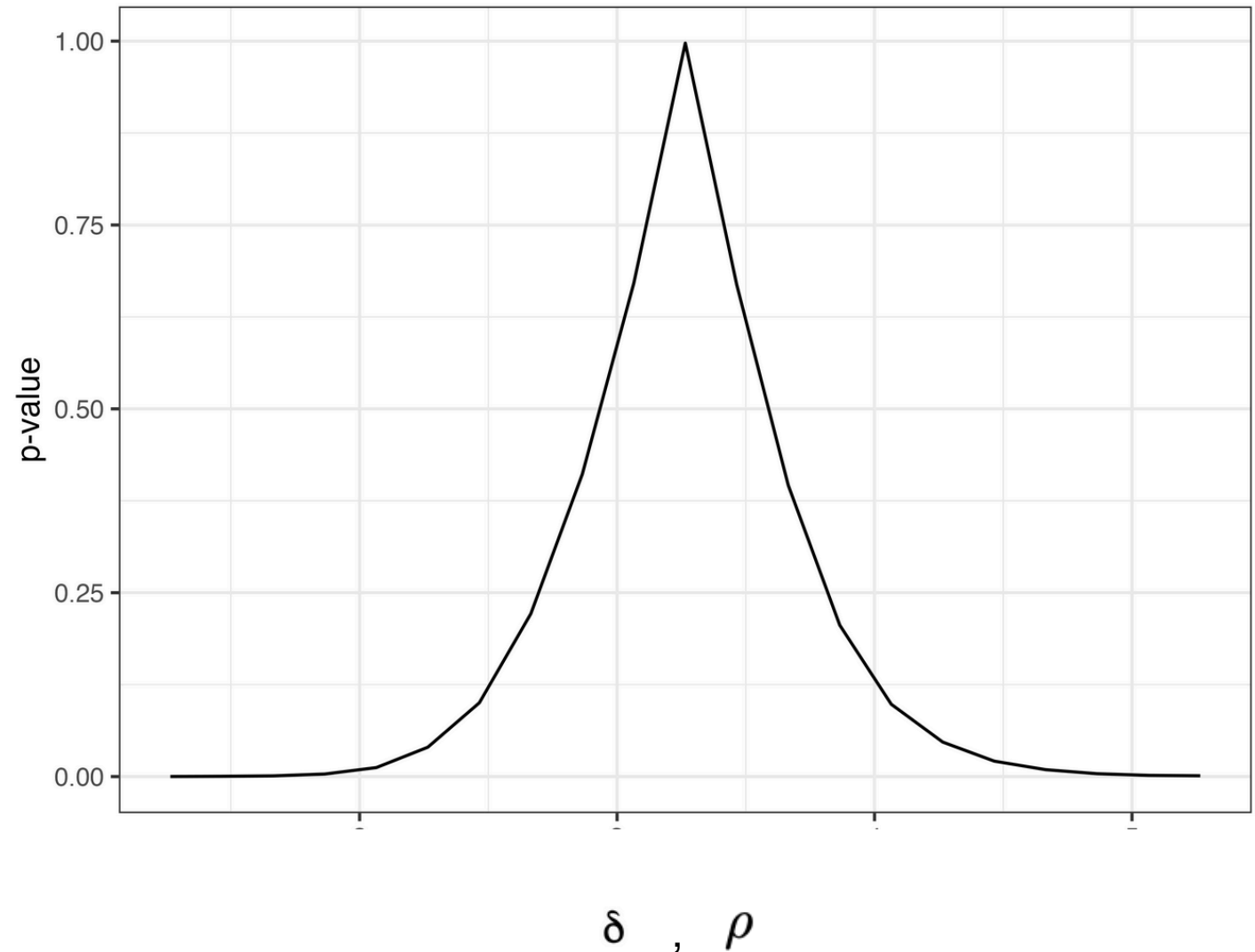
$$X_2 = \delta + \rho X_1$$

$$\mathbb{E}[X_2] = \delta + \rho \mathbb{E}[X_1]$$

$$\mathbb{V}[X_2] = \rho^2 \mathbb{V}[X_1]$$

$$H_0 : \quad \delta = \mu_2 - \frac{\sigma_2}{\sigma_1} \mu_1 = 1,$$

$$\rho = \frac{\sigma_2}{\sigma_1} = 4$$





## 2 . Tester *flipr* !

```
i Loading flipr
i Testing flipr
✓ | OK F W S | Context
✓ | 5         | plausibility-class [1.6 s]

== Results ==
=====
Duration: 1.6 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 5 ]
```

Remarque : t-stat ne fait pas de statistiques sur une série constante

**THANK YOU!**

Bonus

## FLIPSCALAR

$$x_1, x_2, \dots, x_n \in \mathbb{R}$$

Scalars permutations

Flexible Inference via  
Permutations in **R**

## FLIPVECTOR

$$v_1, v_2, \dots, v_n \in \mathbb{R}^p, \quad p \geq 2$$

Vectors permutations

## FDATEST

Nonparametric inference for  
Functional Data Analysis

$$f_1, f_2, \dots, f_n \in \mathcal{L}^p, \quad p \geq 2$$

Functions permutations

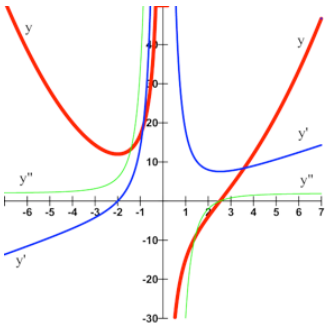
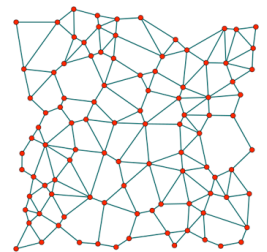


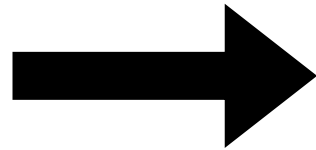
## NEVADA

NEtwork-VALued Data Analysis

$$g_1, g_2, \dots, g_n$$

Graphs permutations





*flipr s'étend  
avec flippy !*



**FLIPSCALAR**  
**FLIPVECTOR**  
**FDATEST**  
**NEVADA**  
...

**GUDHI**  
**GEOMSTATS**  
...

Flexible Inference via  
Permutations in **R**

**FLIPSCALAR**

**FLIPVECTOR**



**FDATEST**

**NEVADA**

