Cours 1 : Les fonctions

Exercices

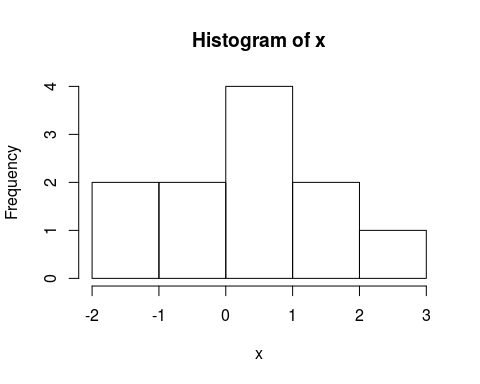
# Exemple d'utilisation

Ecrire une fonction qui, à partir d'une variable, effectue les étapes suivantes :

* Affiche le nombre de données manquante s'il y en a ;
* Impute les données manquantes par la moyenne ;
* Représente un histogramme des données après imputation ;
* Et retourne à l'utilisateur un summary() de la variable après imputation.

impute\_and\_describe <- function(x) {  
 if (any(is.na(x))) {  
 message(paste(  
 "Le nombre de données manquantes est",  
 sum(is.na(x))  
 ))  
 }  
 x[is.na(x)] <- mean(x, na.rm = TRUE)  
 hist(x)  
 summary(x)  
}  
impute\_and\_describe(c(rnorm(10), NA))

## Le nombre de données manquantes est 1



## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -1.8380 -0.5772 0.2887 0.2359 0.9381 2.4430

# Le lexical scoping

Ecrire une fonction spécifique à un jeu de données. Par exemple à partir du jeu de données infert, une fonction :

* Qui prend comme argument le nom d'une variable ;
* Et retourne la force d'association de cette variable avec la variable case (cas ou témoin) ;
* Sans prendre infert comme argument.

assoc\_case\_infert <- function(x) {  
 if (is.numeric(infert[, x])) {  
 t.test(  
 infert[infert$case == 0, x],  
 infert[infert$case == 1, x]  
 )  
 } else {  
 chisq.test(table(  
 infert$case,  
 infert[, x]  
 ))  
 }  
}  
assoc\_case\_infert("age")

##   
## Welch Two Sample t-test  
##   
## data: infert[infert$case == 0, x] and infert[infert$case == 1, x]  
## t = -0.055288, df = 163.77, p-value = 0.956  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -1.439600 1.361177  
## sample estimates:  
## mean of x mean of y   
## 31.49091 31.53012

assoc\_case\_infert("education")

## Warning in chisq.test(table(infert$case, infert[, x])): Chi-squared  
## approximation may be incorrect

##   
## Pearson's Chi-squared test  
##   
## data: table(infert$case, infert[, x])  
## X-squared = 0.0022896, df = 2, p-value = 0.9989

infert est recherché dans l'environnement où f() est définie. Si on appelle f() depuis une fonction où on défini un autre jeu de données infert on risque d'avoir des surprises.

Exemple : faire une fonction qui rééchantillonne infert, le modifie, et appelle à nouveau assoc\_case\_infert().

resample\_and\_test <- function(x) {  
 resamp <- sample(  
 seq\_len(nrow(infert)),  
 size = nrow(infert),  
 replace = TRUE  
 )  
 infert <- infert[resamp, ]  
 assoc\_case\_infert(x)  
}  
resample\_and\_test("age")

##   
## Welch Two Sample t-test  
##   
## data: infert[infert$case == 0, x] and infert[infert$case == 1, x]  
## t = -0.055288, df = 163.77, p-value = 0.956  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -1.439600 1.361177  
## sample estimates:  
## mean of x mean of y   
## 31.49091 31.53012

Ca ne fonctionne pas car le infert utilisé par la fonction assoc\_case\_infert() est toujours celui de son *enclosing environment*, l'environnement global, et pas l'infert de son *calling environment*, l'*execution environment* de resample\_and\_test().

# Manipulation des arguments

Ecrire une fonction qui a les proprités suivantes :

* Prend 2 arguments x et y ;
* Représente un nuage de points des arguments ;
* x a comme valeur par défault 10 valeurs aléatoires entre 0 et 1 ;
* Si y est manquant alors x est représenté en ordonnée, selon son index en abcisse.

myplot1 <- function(x = runif(10), y) {  
 if(missing(y)) {  
 y <- x  
 x <- seq\_along(y)  
 }  
 plot(x, y)  
}

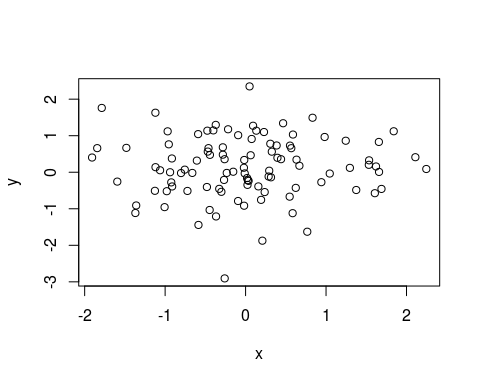
Autre exercice : pareil, mais si y est manquant il prend les valeurs des index de x.

myplot2 <- function(x = runif(10), y = seq\_along(x)) {  
 plot(x, y)  
}

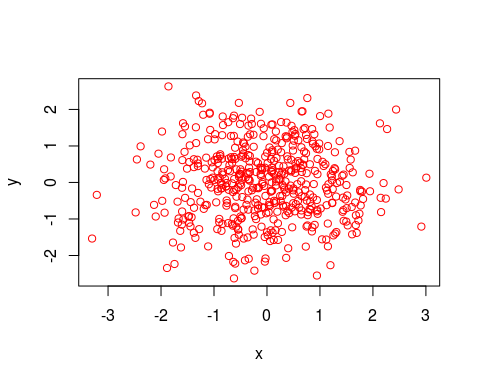
# Dot Dot Dot (...)

Faire un fonction qui prend comme argument un nombre de points et représente un nuage de points aléatoirement généré selon un distribution normale. Les paramètres de plot() doivent pouvoir être modifiés depuis la fonction.

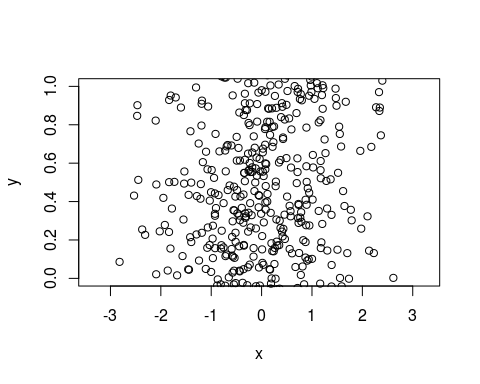
random\_scatter <- function(n, ...) {  
 x <- rnorm(n)  
 y <- rnorm(n)  
   
 plot(x, y, ...)  
}  
random\_scatter(100)



random\_scatter(500, col = "red")



random\_scatter(1000, ylim = c(0, 1))



Faire une fonction qui à partir d'un nombre arbitraire de t.test() retourne le plus significatif.[[1]](#footnote-29)

most\_signif\_ttest <- function(...) {  
 list\_ttest <- list(...)  
   
 extract\_p <- function(x)  
 x$p.value  
   
 pvals <- unlist(lapply(list\_ttest, extract\_p))  
   
 list\_ttest[[which(pvals == min(pvals))]]  
}  
most\_signif\_ttest(  
 t.test(rnorm(10), rnorm(10)),  
 t.test(rnorm(10), rnorm(10)),  
 t.test(rnorm(10), rnorm(10)),  
 t.test(rnorm(10), rnorm(10))  
)

##   
## Welch Two Sample t-test  
##   
## data: rnorm(10) and rnorm(10)  
## t = -0.97768, df = 15.13, p-value = 0.3436  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -1.2011506 0.4453481  
## sample estimates:  
## mean of x mean of y   
## -0.09934543 0.27855581

## Valeurs de retour complexes

Faire une fonction qui retourne pour une variable :

* Sa moyenne ;
* Son étendue ;
* La comparaison de sa moyenne à 0.

some\_stats <- function(x) {  
 list(  
 moyenne = mean(x),  
 etendue = range(x),  
 test0 = t.test(x)  
 )  
}  
some\_stats(rnorm(10))

## $moyenne  
## [1] 0.45912  
##   
## $etendue  
## [1] -0.975683 1.767514  
##   
## $test0  
##   
## One Sample t-test  
##   
## data: x  
## t = 1.6646, df = 9, p-value = 0.1304  
## alternative hypothesis: true mean is not equal to 0  
## 95 percent confidence interval:  
## -0.164817 1.083057  
## sample estimates:  
## mean of x   
## 0.45912

Faire une fonction qui impute les valeurs manquantes par la moyenne, retourne un vecteur, mais garde la trace de quelles valeurs sont imputées.

impute\_mean <- function(x) {  
 res <- x  
 res[is.na(x)] <- mean(x, na.rm = TRUE)  
   
 attr(res, "imputed") <- is.na(x)  
 res  
}  
  
(test <- impute\_mean(c(1, 2, NA, 3)))

## [1] 1 2 2 3  
## attr(,"imputed")  
## [1] FALSE FALSE TRUE FALSE

attr(test, "imputed")

## [1] FALSE FALSE TRUE FALSE

# Une fonction argument d'une fonction

Reprendre l'exemple précédent, mais utiliser une fonction arbitraire pour faire l'imputation. Du coup la fonction arbitraire prend peut-être d'autres arguments arbitraires ?

impute\_generalized <- function(x, f, ...) {  
 res <- x  
 res[is.na(x)] <- f(na.omit(x), ...)  
   
 attr(res, "imputed") <- is.na(x)  
 res  
}  
x <- c(1, 2, NA, 3)  
impute\_generalized(x, quantile, probs = .25)

## [1] 1.0 2.0 1.5 3.0  
## attr(,"imputed")  
## [1] FALSE FALSE TRUE FALSE

1. Plus avancé, utilise lapply() (prévu pour la leçon suivante). [↑](#footnote-ref-29)