TP OUMOBIO 4: Utiliser des outils avancés sous R

Mathieu Brevet

2024-09-16

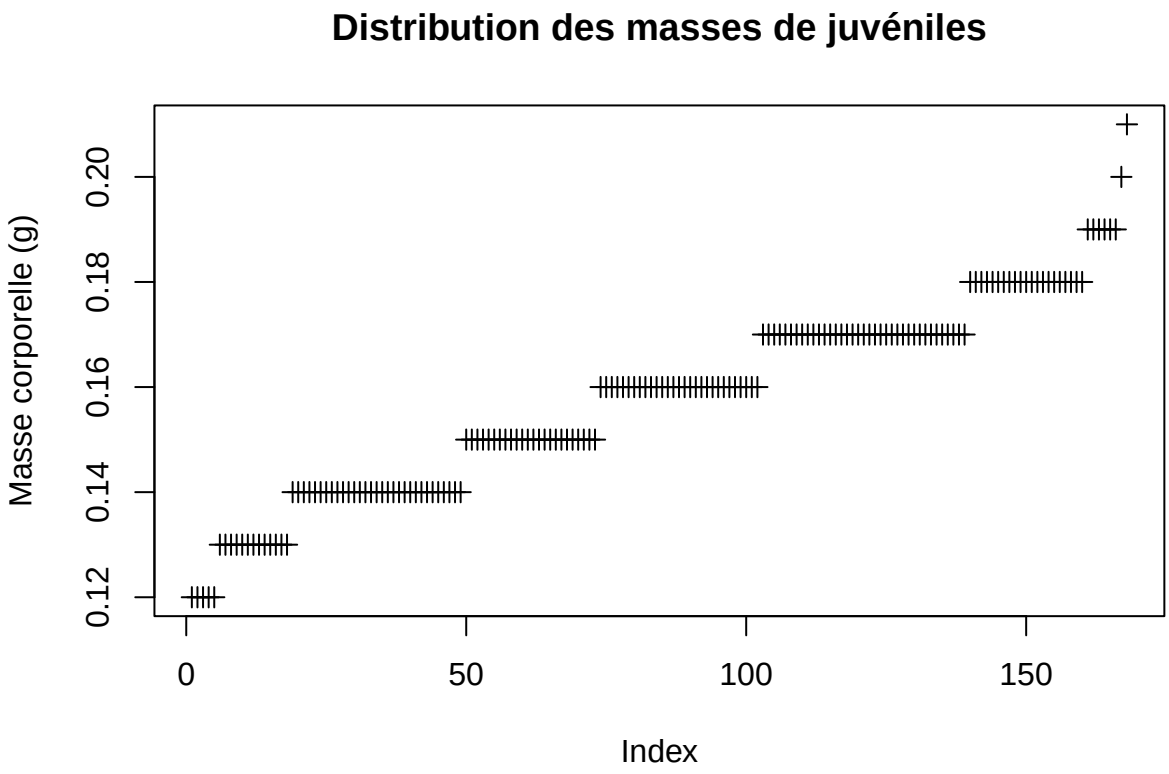
Bienvenue dans ce quatrième et dernier TP sur R ! Après avoir vu les principales fonctions et commandes de base sous R nous allons maintenant aborder les commandes et pratiques avancées sur R et Rstudio: l'utilisation d'éléments de programmation (conditions sur les commandes et boucles pour les répéter), les paramètres graphiques avancées, l'utilisation de commandes prédéfinies par d'autres utilisateurs (paquet de fonctions) et la gestion des erreurs.

```
setwd("~/ATER PAU 2024/Cours modifiés/OUMOBIOS")


data_lezard = read.table("Suivi_lezard_vivipare.csv", header = T, sep = "\t", dec = ",")
```

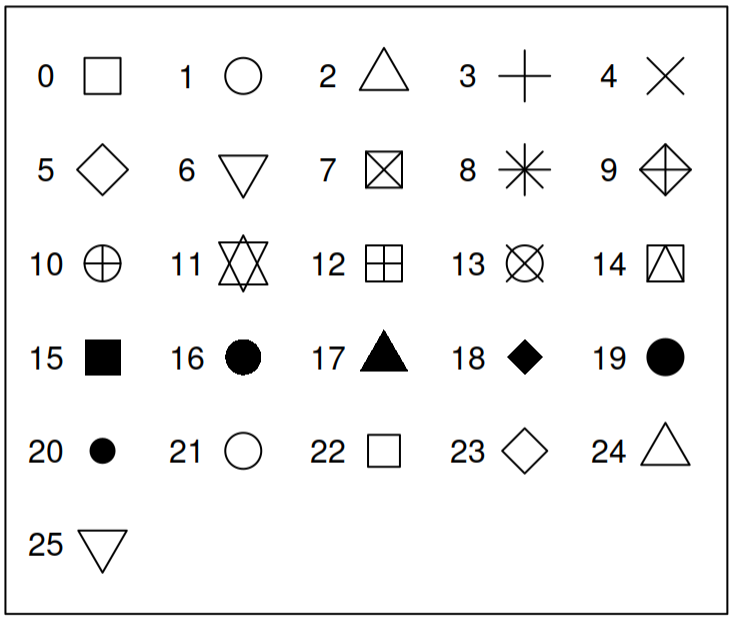
Paramètres graphiques avancés

Les graphiques que nous avons produits jusqu'à maintenant sont relativement simples, mais il existe de nombreuses options graphiques dans R permettant de **personnaliser et mettre en forme** ses graphiques. Ces options permettent notamment de gérer la couleurs, la taille et le style des figurés, d'ajouter du texte au graphique, de mettre en forme tout texte apparaissant dans le graphique, ou encore de modifier les axes et les marges du graphique. Voici quelques exemples des principales options existantes:

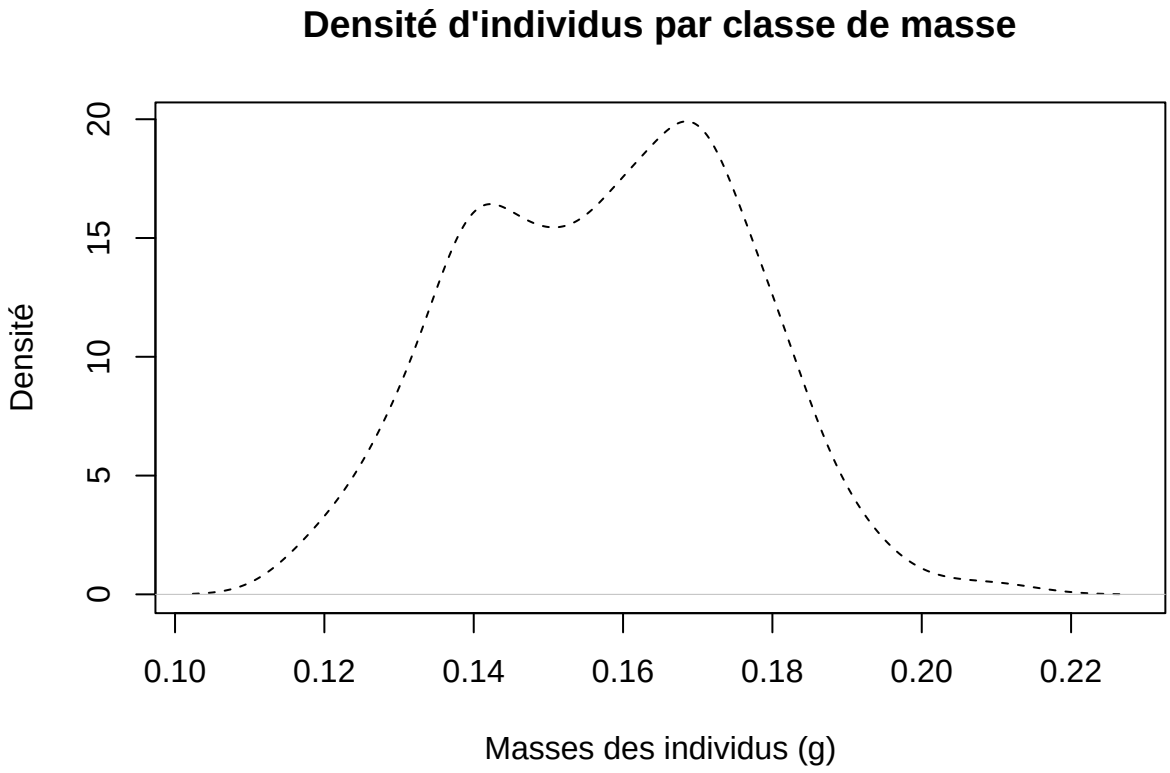
```
# modification des figurés:

## symbole:

plot(
sort(data_lezard$M_IND),
main = "Distribution des masses de juvéniles",
ylab = "Masse corporelle (g)",
pch = 3
)
```



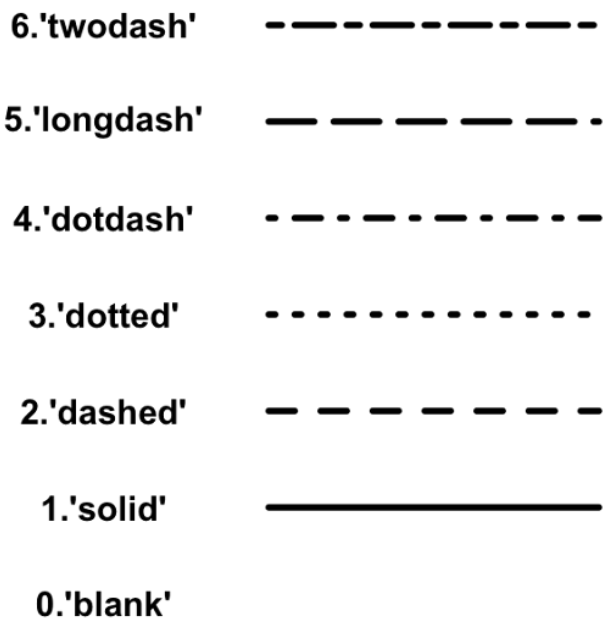

```
# pour les points l'argument "pch" permet de gérer le type de symbole,
# vous pouvez trouver ci-dessous les différents symboles utilisables:
```



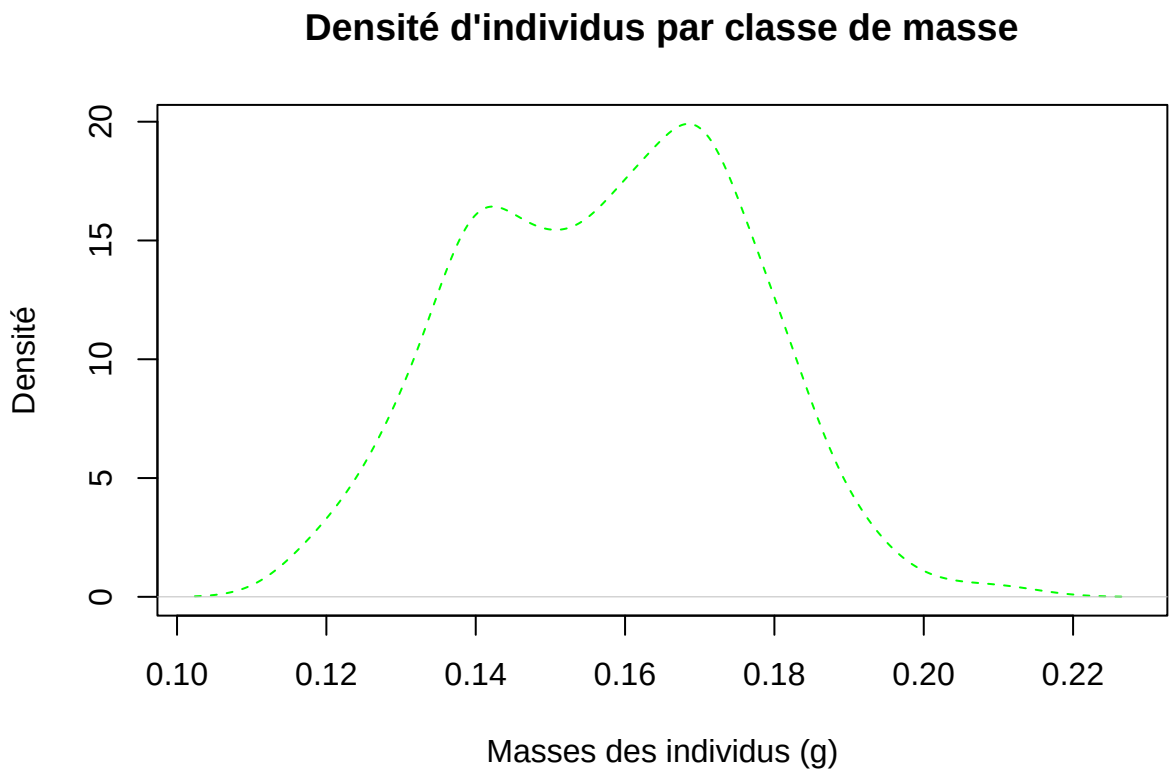

```
plot(
density(data_lezard$M_IND),
main = "Densité d'individus par classe de masse",
xlab = "Masses des individus (g)",
ylab = "Densité",
lty = "dashed"
)
```




```
# pour les lignes l'argument "lty" permet de gérer le type de symbole,
# vous pouvez trouver ci-dessous les différents symboles utilisables:
```



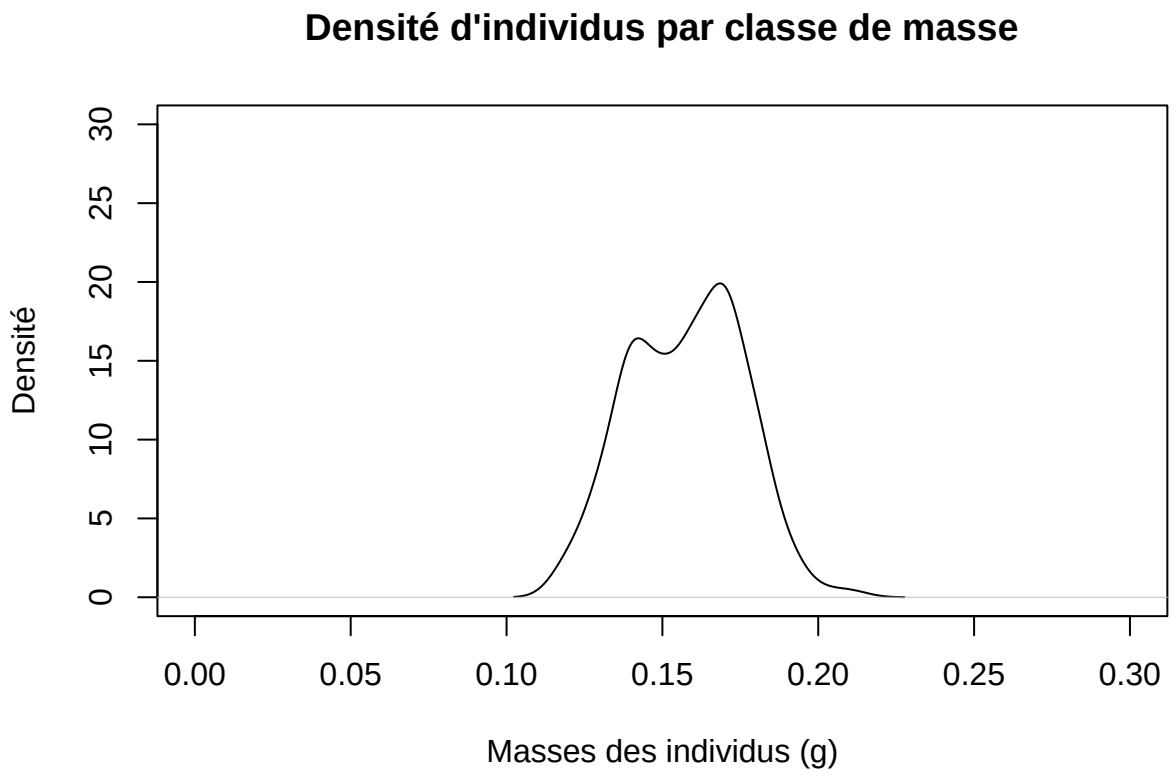
```r
# couleurs:

plot(
density(data_lezard$M_IND),
main = "Densité d'individus par classe de masse",
xlab = "Masses des individus (g)",
ylab = "Densité",
lty = "dashed",
col = "green"
)
```



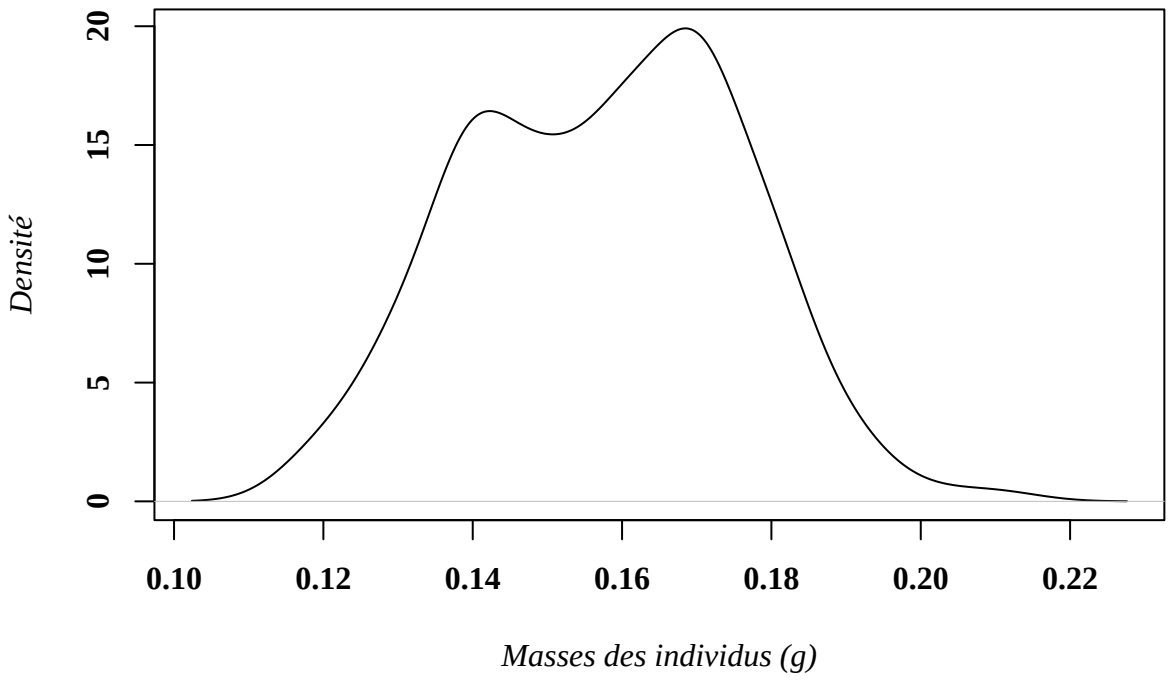

```r
# les couleurs sont gérées à l'aide l'argument "col",
# voici les couleurs pouvant être utilisés dans R:
```

white	coral4	deepskyblue	gray28	gray88	grey40	grey100	lightpink2	mistyrose2	plum	slategray2
aliceblue	cornflowerblue	deepskyblue1	gray29	gray89	grey41	honeydew	lightpink3	mistyrose3	plum1	slategray3
antiquewhite	cornsilk	deepskyblue2	gray30	gray90	grey42	honeydew1	lightpink4	mistyrose4	plum2	slategray4
antiquewhite1	cornsilk1	deepskyblue3	gray31	gray91	grey43	honeydew2	lightsalmon	moccasin	plum3	slategrey
antiquewhite2	cornsilk2	deepskyblue4	gray32	gray92	grey44	honeydew3	lightsalmon1	navajowhite	plum4	snow
antiquewhite3	cornsilk3	dimgray	gray33	gray93	grey45	honeydew4	lightsalmon2	navajowhite1	powderblue	snow1
antiquewhite4	cornsilk4	dimgrey	gray34	gray94	grey46	hotpink	lightsalmon3	navajowhite2	purple	snow2
aquamarine	cyan	dodgerblue	gray35	gray95	grey47	hotpink1	lightsalmon4	navajowhite3	purple1	snow3
aquamarine1	cyan1	dodgerblue1	gray36	gray96	grey48	hotpink2	lightseagreen	navajowhite4	purple2	snow4
aquamarine2	cyan2	dodgerblue2	gray37	gray97	grey49	hotpink3	lightskyblue	navy	purple3	springgreen
aquamarine3	cyan3	dodgerblue3	gray38	gray98	grey50	hotpink4	lightskyblue1	navyblue	purple4	springgreen1
aquamarine4	cyan4	dodgerblue4	gray39	gray99	grey51	indianred	lightskyblue2	oldlace	red	springgreen2
azure	darkblue	firebrick	gray40	gray100	grey52	indianred1	lightskyblue3	olivedrab	red1	springgreen3
azure1	darkcyan	firebrick1	gray41	green	grey53	indianred2	lightskyblue4	olivedrab1	red2	springgreen4
azure2	darkgoldenrod	firebrick2	gray42	green1	grey54	indianred3	lightslateblue	olivedrab2	red3	steelblue
azure3	darkgoldenrod1	firebrick3	gray43	green2	grey55	indianred4	lightslategray	olivedrab3	red4	steelblue1
azure4	darkgoldenrod2	firebrick4	gray44	green3	grey56	ivory	lightslategrey	olivedrab4	rosybrown	steelblue2
beige	darkgoldenrod3	floralwhite	gray45	green4	grey57	ivory1	lightsteelblue	orange	rosybrown1	steelblue3
bisque	darkgoldenrod4	forestgreen	gray46	greenyellow	grey58	ivory2	lightsteelblue1	orange1	rosybrown2	steelblue4
bisque1	darkgray	gainsboro	gray47	grey	grey59	ivory3	lightsteelblue2	orange2	rosybrown3	tan
bisque2	darkgreen	ghostwhite	gray48	grey0	grey60	ivory4	lightsteelblue3	orange3	rosybrown4	tan1
bisque3	darkgrey	gold	gray49	grey1	grey61	khaki	lightsteelblue4	orange4	royalblue	tan2
bisque4	darkkhaki	gold1	gray50	grey2	grey62	khaki1	lightyellow	orangered	royalblue1	tan3
black	darkmagenta	gold2	gray51	grey3	grey63	khaki2	lightyellow1	orangered1	royalblue2	tan4
blanchedalmon	darkolivegreen	gold3	gray52	grey4	grey64	khaki3	lightyellow2	orangered2	royalblue3	thistle
blue	darkolivegreen1	gold4	gray53	grey5	grey65	khaki4	lightyellow3	orangered3	royalblue4	thistle1
blue1	darkolivegreen2	goldenrod	gray54	grey6	grey66	lavender	lightyellow4	orangered4	saddlebrown	thistle2
blue2	darkolivegreen3	goldenrod1	gray55	grey7	grey67	lavenderblush	limegreen	orchid	salmon	thistle3
blue3	darkolivegreen4	goldenrod2	gray56	grey8	grey68	lavenderblush1	linen	orchid1	salmon1	thistle4
blue4	darkorange	goldenrod3	gray57	grey9	grey69	lavenderblush2	magenta	orchid2	salmon2	tomato
blueviolet	darkorange1	goldenrod4	gray58	grey10	grey70	lavenderblush3	magenta1	orchid3	salmon3	tomato1
brown	darkorange2	gray	gray59	grey11	grey71	lavenderblush4	magenta2	orchid4	salmon4	tomato2
brown1	darkorange3	gray0	gray60	grey12	grey72	lawngreen	magenta3	palegoldenrod	sandybrown	tomato3
brown2	darkorange4	gray1	gray61	grey13	grey73	lemonchiffon	magenta4	palegreen	seagreen	tomato4
brown3	darkorchid	gray2	gray62	grey14	grey74	lemonchiffon1	maroon	palegreen1	seagreen1	turquoise
brown4	darkorchid1	gray3	gray63	grey15	grey75	lemonchiffon2	maroon1	palegreen2	seagreen2	turquoise1
burlywood	darkorchid2	gray4	gray64	grey16	grey76	lemonchiffon3	maroon2	palegreen3	seagreen3	turquoise2
burlywood1	darkorchid3	gray5	gray65	grey17	grey77	lemonchiffon4	maroon3	palegreen4	seagreen4	turquoise3
burlywood2	darkorchid4	gray6	gray66	grey18	grey78	lightblue	maroon4	paleturquoise	seashell	turquoise4
burlywood3	darkred	gray7	gray67	grey19	grey79	lightblue1	mediumaquamari	paleturquoise1	seashell1	violet
burlywood4	darksalmon	gray8	gray68	grey20	grey80	lightblue2	mediumblue	paleturquoise2	seashell2	violetred
cadetblue	darkseagreen	gray9	gray69	grey21	grey81	lightblue3	mediumorchid	paleturquoise3	seashell3	violetred1
cadetblue1	darkseagreen1	gray10	gray70	grey22	grey82	lightblue4	mediumorchid1	paleturquoise4	seashell4	violetred2
cadetblue2	darkseagreen2	gray11	gray71	grey23	grey83	lightcoral	mediumorchid2	palevioletred	sienna	violetred3
cadetblue3	darkseagreen3	gray12	gray72	grey24	grey84	lightcyan	mediumorchid3	palevioletred1	sienna1	violetred4
cadetblue4	darkseagreen4	gray13	gray73	grey25	grey85	lightcyan1	mediumorchid4	palevioletred2	sienna2	wheat
chartreuse	darkslateblue	gray14	gray74	grey26	grey86	lightcyan2	mediumpurple	palevioletred3	sienna3	wheat1
chartreuse1	darkslategray	gray15	gray75	grey27	grey87	lightcyan3	mediumpurple1	palevioletred4	sienna4	wheat2
chartreuse2	darkslategray1	gray16	gray76	grey28	grey88	lightcyan4	mediumpurple2	papayawhip	skyblue	wheat3
chartreuse3	darkslategray2	gray17	gray77	grey29	grey89	lightgoldenrod	mediumpurple3	peachpuff	skyblue1	wheat4
chartreuse4	darkslategray3	gray18	gray78	grey30	grey90	lightgoldenrod	mediumpurple4	peachpuff1	skyblue2	whitesmoke
chocolate	darkslategray4	gray19	gray79	grey31	grey91	lightgoldenrod2	mediumseagree	peachpuff2	skyblue3	yellow
chocolate1	darkslategrey	gray20	gray80	grey32	grey92	lightgoldenrod3	mediumslateblu	peachpuff3	skyblue4	yellow1
chocolate2	darkturquoise	gray21	gray81	grey33	grey93	lightgoldenrod4	mediumspringgre	peachpuff4	slateblue	yellow2
chocolate3	darkviolet	gray22	gray82	grey34	grey94	lightgoldenrodyel	mediumturquois	peru	slateblue1	yellow3
chocolate4	deeppink	gray23	gray83	grey35	grey95	lightgray	mediumvioletred	pink	slateblue2	yellow4
coral	deeppink1	gray24	gray84	grey36	grey96	lightgreen	midnightblue	pink1	slateblue3	yellowgreen
coral1	deeppink2	gray25	gray85	grey37	grey97	lightgrey	mintcream	pink2	slateblue4	
coral2	deeppink3	gray26	gray86	grey38	grey98	lightpink	mistyrose	pink3	slategray	
coral3	deeppink4	gray27	gray87	grey39	grey99	lightpink1	mistyrose1	pink4	slategray1	

```
# modification des axes:


plot(
density(data_lezard$M_IND),
main = "Densité d'individus par classe de masse",
xlab = "Masses des individus (g)",
ylab = "Densité",
xlim = c(0, 0.3),
ylim = c(0, 30)
)
```

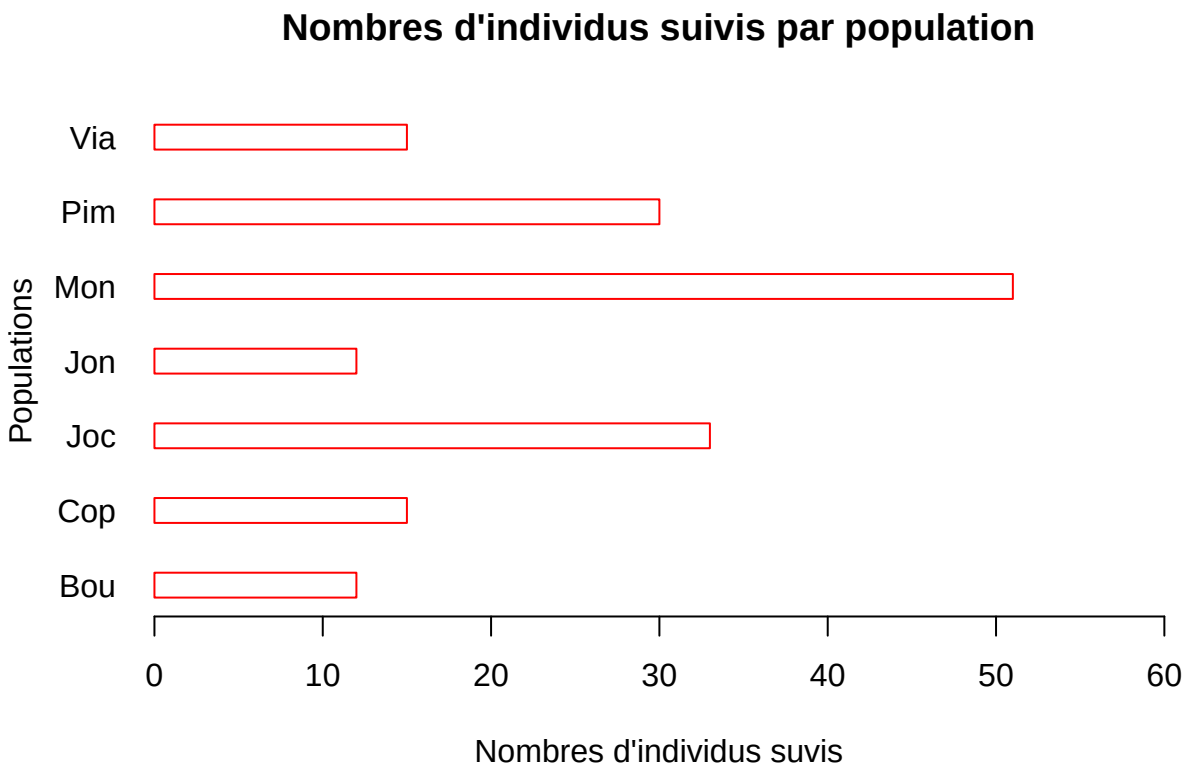



```r
# xlim et ylim définissent les intervalles des axes horizontaux et verticaux
# (peut aussi être géré avec l'argument "asp" qui gère le ratio d'apparence entre axes)



# gestion texte:


plot(
    density(data_lezard$M_IND),
    main = "Densité d'individus par classe de masse",
    xlab = "Masses des individus (g)",
    ylab = "Densité",
    cex.main = 1.5,
    font.lab = 3,
    font.axis = 2,
    family = "serif"
)
```




```
# les différents arguments présentés ici permettent de gérer l'aspect du texte contenu
# dans le graphique (soit pour tous les textes du graphique si on n'indique pas de point médian,
# soit sur le titre / sous-titre / légendes / axes en indiquant l'élément après le point):
# cex.main/sub/lab/axis: taille de la police
# font.main/sub/lab/axis: effet de police
# (1: normal, 2: gras, 3: italique, 4: gras et italique)
# family.main/sub/lab/axis: type de police









# gestion barplot / boxplot :

barplot(
  table(data_lezard$POP),
  main = "Nombres d'individus suivis par population",
  xlab = "Nombres d'individus suvis",
  ylab = "Populations",
  xlim = c(0,60),
  space = 2,
  horiz = T,
  border = "red",
  col = "white",
```

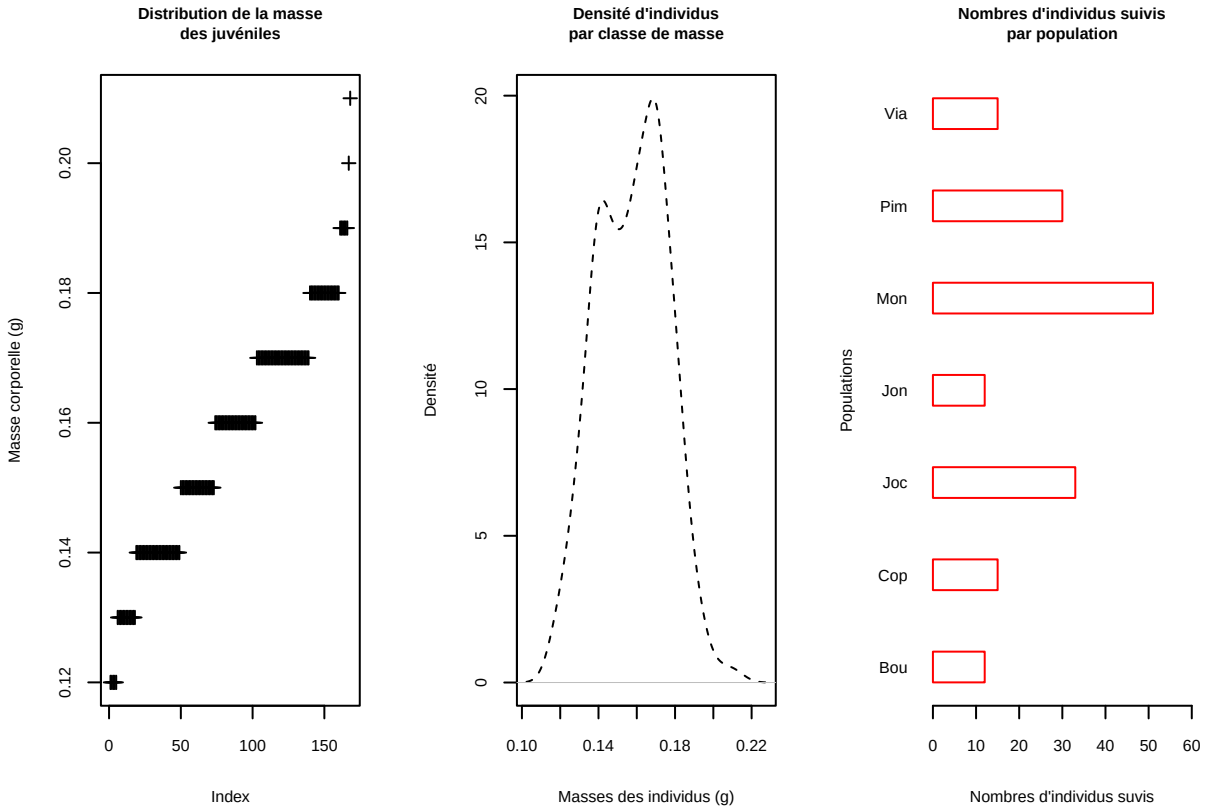
```
  names.arg = c("Bou", "Cop", "Joc", "Jon", "Mon", "Pim", "Via"),
  las = 1)
```



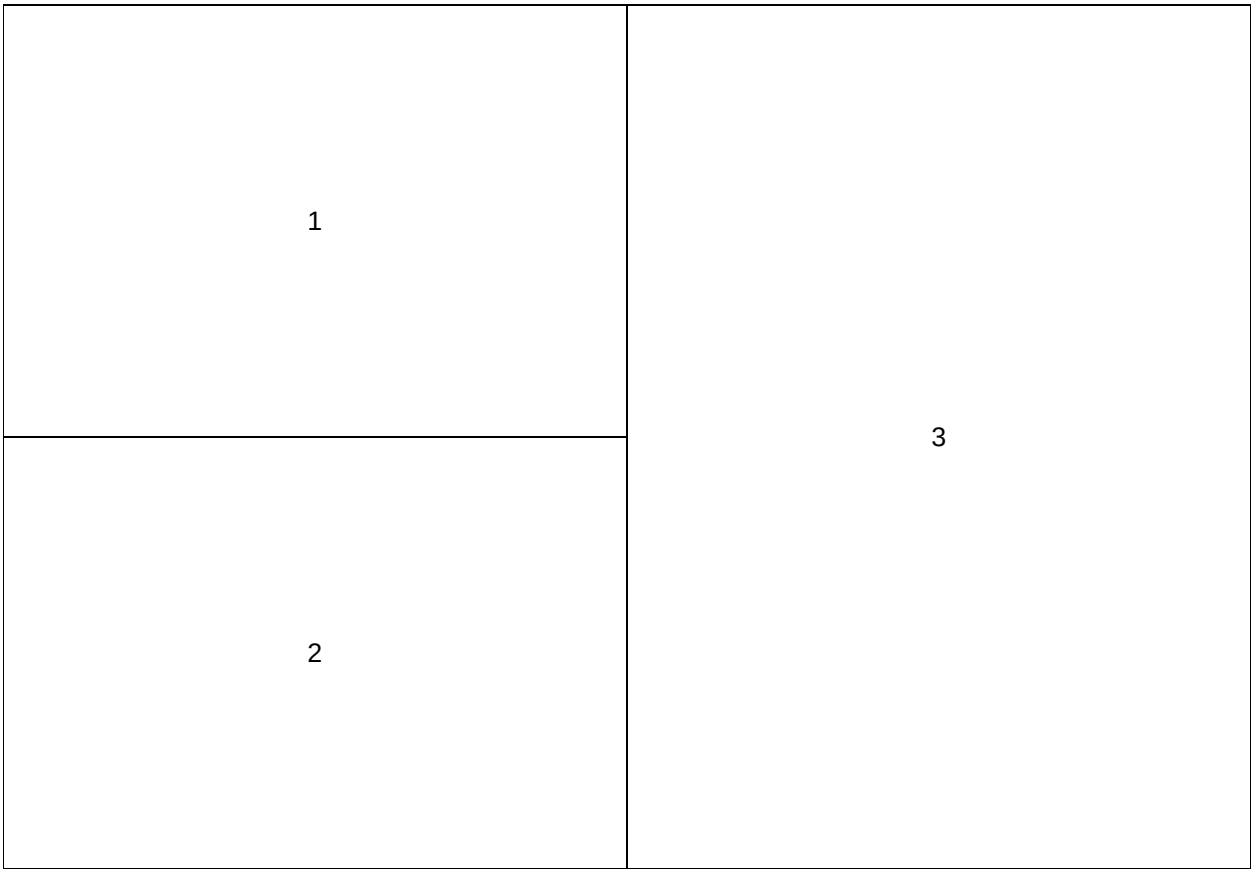

```
# "space" permet de gérer l'espacement entre les barres / "horiz" permet de faire
# apparaître le graphique à l'horizontale / "border" permet de gérer la couleur des
# lignes / "col" permet de gérer la couleur du remplissage / "names.arg" permet de
# gérer le nom des légendes de catégories
# NB: pour boxplot remplacer "horiz" par "horizontal", "names.arg" par "names" et
# utiliser "width" et non pas "space" pour gérer la largeur des boxplots
```

```
# panels de figures:

par(mfrow=c(1, 3))
# modification du nombre de figures par sortie graphique (nombre de ligne de figure,
# nombre de figure par ligne), ici on définit donc que toutes les prochaines sorties
# contiendront une ligne de trois graphiques

plot(
sort(data_lezard$M_IND),
main = "Distribution de la masse\ndes juvéniles",
ylab = "Masse corporelle (g)",
pch = 3,
cex.axis = 0.8,
cex.lab=0.8,
cex.main=0.8)
```

```r
plot(
density(data_lezard$M_IND),
main = "Densité d'individus\npar classe de masse",
xlab = "Masses des individus (g)",
ylab = "Densité",
lty = "dashed",
cex.axis = 0.8,
cex.lab=0.8,
cex.main=0.8)


barplot(
  table(data_lezard$POP),
  main = "Nombres d'individus suivis\npar population",
  xlab = "Nombres d'individus suvis",
  ylab = "Populations",
  xlim = c(0,60),
  space = 2,
  horiz = T,
  border = "red",
  col = "white",
  names.arg = c("Bou", "Cop", "Joc", "Jon", "Mon", "Pim", "Via"),
  las = 1,
  cex.names = 0.8,
  cex.axis = 0.8,
  cex.lab=0.8,
  cex.main=0.8)
```

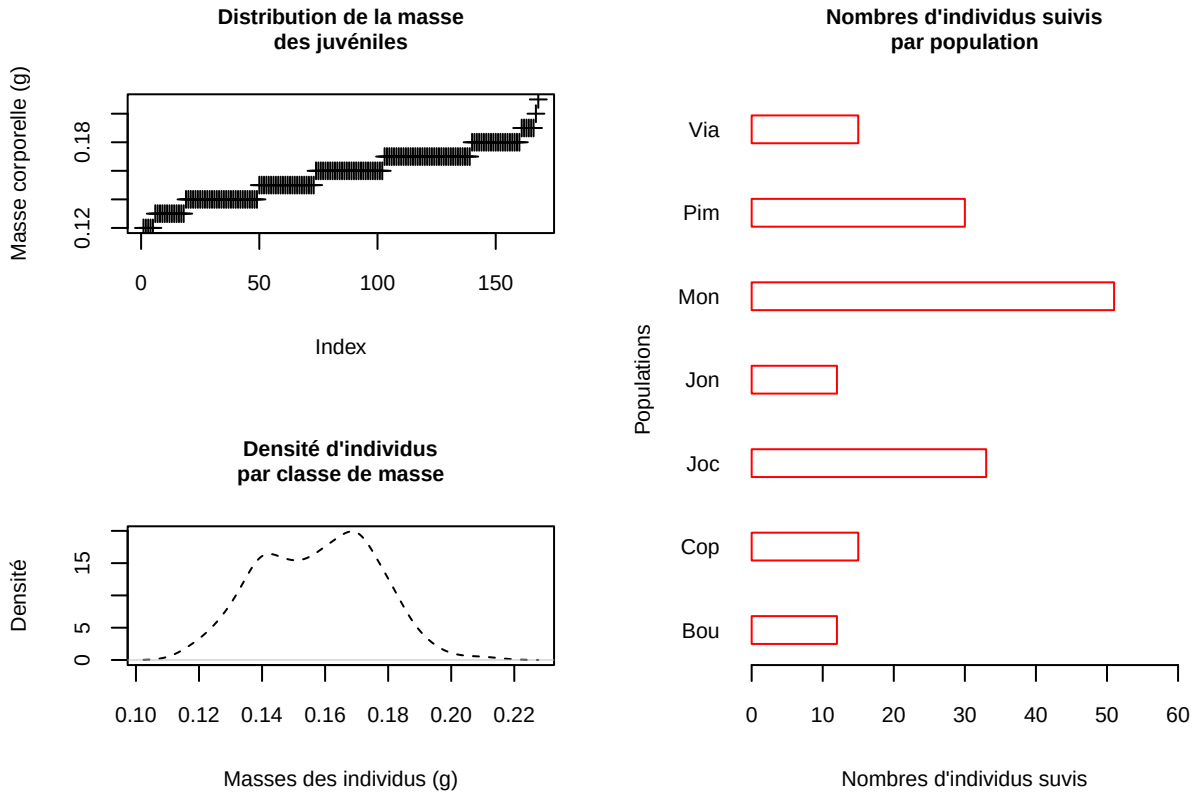



```
par(mfrow=c(1, 1))



# on peut aussi gérer finement l'agencement des figures et les marges externes
# du panel de figure


par(mfrow = c(2, 2))
# ici on indique qu'on va réaliser un panel prenant 2 lignes et 2 colonnes, puis on change
# les marges extérieures du panel (argument "oma=c(bas, gauche, haut, droite)",
# en nombre de ligne)

layout(mat = matrix(c(1, 2, 3, 3), 2, 2, byrow = F))
# ici on indique l'arrangement de nos trois figures, les figures 1 et 2 seront dans la
# première colonne et la figure 3 prendra toute la deuxième colonne (la matrice étant
# définie par colonne, avec 2 lignes et 2 colonnes)
layout.show(n=3)
```




```r
# teste si notre arrangement graphique est bien le bon, en indiquant en entrée le nombre
# de figures qu'on souhaite voir apparaître dans notre panel

plot(
sort(data_lezard$M_IND),
main = "Distribution de la masse\ndes juvéniles",
ylab = "Masse corporelle (g)",
pch = 3,
cex.axis = 0.8,
cex.lab=0.8,
cex.main=0.8)

plot(
density(data_lezard$M_IND),
main = "Densité d'individus\npar classe de masse",
xlab = "Masses des individus (g)",
ylab = "Densité",
lty = "dashed",
cex.axis = 0.8,
cex.lab=0.8,
cex.main=0.8)


barplot(
  table(data_lezard$POP),
  main = "Nombres d'individus suivis\npar population",
```

```
  xlab = "Nombres d'individus suvis",
  ylab = "Populations",
  xlim = c(0,60),
  space = 2,
  horiz = T,
  border = "red",
  col = "white",
  names.arg = c("Bou", "Cop", "Joc", "Jon", "Mon", "Pim", "Via"),
  las = 1,
  cex.names = 0.8,
  cex.axis = 0.8,
  cex.lab=0.8,
  cex.main=0.8)
```




```
par(mfrow = c(1, 1))
# ré-initialisation des paramètres graphiques
```

ASTUCES

Il existe plusieurs aides-mémoires et tutoriels pour gérer les paramètres graphiques sur R. En voici quelques exemples: https://nicolascasajus.fr/files/pdfs/graphr.pdf, http://publish.illinois.edu/johnrgallagher/files/2015/10/BaseGraphicsCheatsheet.pdf, http://www.sthda.com/english/wiki/graphical-parameters.

POUR ALLER PLUS LOIN

On peut aussi gérer les marges des graphiques en utilisant les commandes par(mar()) (pour les marges internes) ou par(oma()) (pour les marges externes), cela peut être particulièrement utile pour ajouter du texte autour d'un graphique (fonction "text()") ou un titre sur un panel de figure (fonction "title()").

MISE EN APPLICATION

Reprenez les graphiques réalisés au cours de la séance et apportez leur les améliorations graphiques que vous jugerez pertinentes.

Utiliser des paquets développés par d'autres utilisateurs

Qu'est-ce qu'un paquet / librairie ?

Comme nous l'avons abordé à la section précédente vous êtes loin d'être seul à travailler sur R. Il en résulte donc que lorsque vous devez créer une fonction donnée il y a une forte probabilité pour que quelqu'un l'ait déjà fait avant vous (sauf si vous abordez un sujet à la pointe des bio-statistiques). Dans de nombreux cas les utilisateurs de R ayant créé ces nouvelles fonctions peuvent les rendre accessible au reste de la communauté en créant un paquet (ou "package" en anglais), composé d'un ensemble de fonction thématique. Ce paquet peut être chargé dans R sous la forme d'une librairie ("library" en anglais) qui vous permettra d'avoir accès à toutes les fonctions contenues dans le paquet.

Prenons un exemple concret avec les fonctions calculant les coefficient d'asymétrie et d'aplatissement, que nous avions créés dans le second TP. Si on fait une recherche à propos de l'existence de telles fonctions sur R, on trouve rapidement l'existence d'un paquet nommé "EnvStats" qui définit ces fonctions. À travers cet exemple nous allons voir comment charger un paquet dans R et comment l'utiliser:

```
install.packages("EnvStats")
# installation du paquet 'EnvStats' (toujours indiquer le nom du paquet entre
# guillemet pour l'installation) Rq: cette commande est à utiliser une unique
# fois sur votre ordinateur, par la suite l'installation est conservé sur votre
# ordinateur (par contre la librairie doit être chargée à chaque fois que vous
# rouvrez une session R)
```

```
library(EnvStats)
```

```
##
## Attachement du package : 'EnvStats'

## Les objets suivants sont masqués depuis 'package:stats':
##
##     predict, predict.lm
```

```
# chargement du paquet sur votre session R
```

```r
# NB : il est fortement conseillé de regrouper le chargement des librairies au
# même endroit dans votre code, en préambule (au même endroit que là où vous
# définissez vos fonctions récurrentes), pour une meilleure lisibilité des
# dépendances associées à votre code




`?`(kurtosis)
```

```
## démarrage du serveur d'aide httpd ...
```

```
##  fini
```

```r
`?`(skewness)
# aide des fonctions calculant les coefficients de Fisher d'aplatissement et
# d'asymétrie



kurtosis(
    data_lezard[!duplicated(data_lezard$ID_MOTHERS),
        ]$M_MOTHERS
)
```

```
## [1] -0.5090337
```

```r
skewness(
    data_lezard[!duplicated(data_lezard$ID_MOTHERS),
        ]$M_MOTHERS
)
```

```
## [1] 0.1401522
```

```r
# légèrement différents que ce que nous obtenions car ici utilisation
# d'estimateurs ''biaisés' (voir aide)
```

L'exemple de ce paquet de fonction est intéressant car il illustre aussi le risque associé au chargement d'un paquet de fonction. Un message d'avertissement s'est affiché lorsque vous avez chargé la librairie "EnvStats" dans votre environnement R. Celui-ci indique que certaines fonctions de cette librairie ont le même nom que des fonctions de base contenues dans R. Ces fonctions sont alors masqués par celle que vous venez de charger en utilisant la librairie. Il faut être extrêmement prudent avec cela car on peut ainsi se retrouver à utiliser la mauvaise fonction !
Voici comment gérer ce type de cas:

```r
# utilisation de la dernière fonction définie avec le nom 'predict' (donc celle
# de la librairie):
predict(x)
```

```r
# utilisation de la fonction contenue dans l'environnement de base R:
stats::predict(x)
# on indique avant le nom de fonction le paquet duquel elle provient, et on
# sépare le nom du paquet et de la fonction par '::'

# utilisation de la fonction contenue dans le package:
EnvStats::predict(x)


# NB: quand il existe dans votre code un conflit sur un nom de fonction et que
# vous utilisez une de ces fonctions dans votre code, il faut TOUJOURS indiquer
# le paquet utilisé (comme réalisé sur les deux lignes de code précédentes)
```

POUR ALLER PLUS LOIN

Les paquets "officiels" produits sur R sont tous déposés sur l'archive CRAN (Comprehensive R Archive Network) de R. Un manuel de référence est alors associé à chaque package et peut facilement être retrouvé en ligne, avec parfois également des vignettes explicatives sur le fonctionnement du package (pas obligatoire et relativement rare). Il arrive toutefois que des packages soient produits en dehors des archives R (soit parce qu'ils sont en développement, soit parce qu'ils ne satisfont pas tous les prérequis nécessaire pour être enregistré dans l'archive officielle), notamment dans les dépôts tels que R-Forge, RForge ou directement sur le GitHub des développeurs.

Un exemple de paquet avancé: les graphiques ggplot

Certains paquets de fonction sont particulièrement utile et sont devenus très communément utilisé par les biostatisticiens. Nous allons aborder ici en particulier le package proposant un nouveau mode de production graphique "ggplot2". Ce mode de visualisation graphique est devenu quasiment incontournable du fait de la qualité des sorties graphiques qu'il propose et de la relative simplicité pour les produire. Vous pouvez trouver ci dessous un mémo sur les différentes commandes possibles d'utiliser grâce à ce paquet:

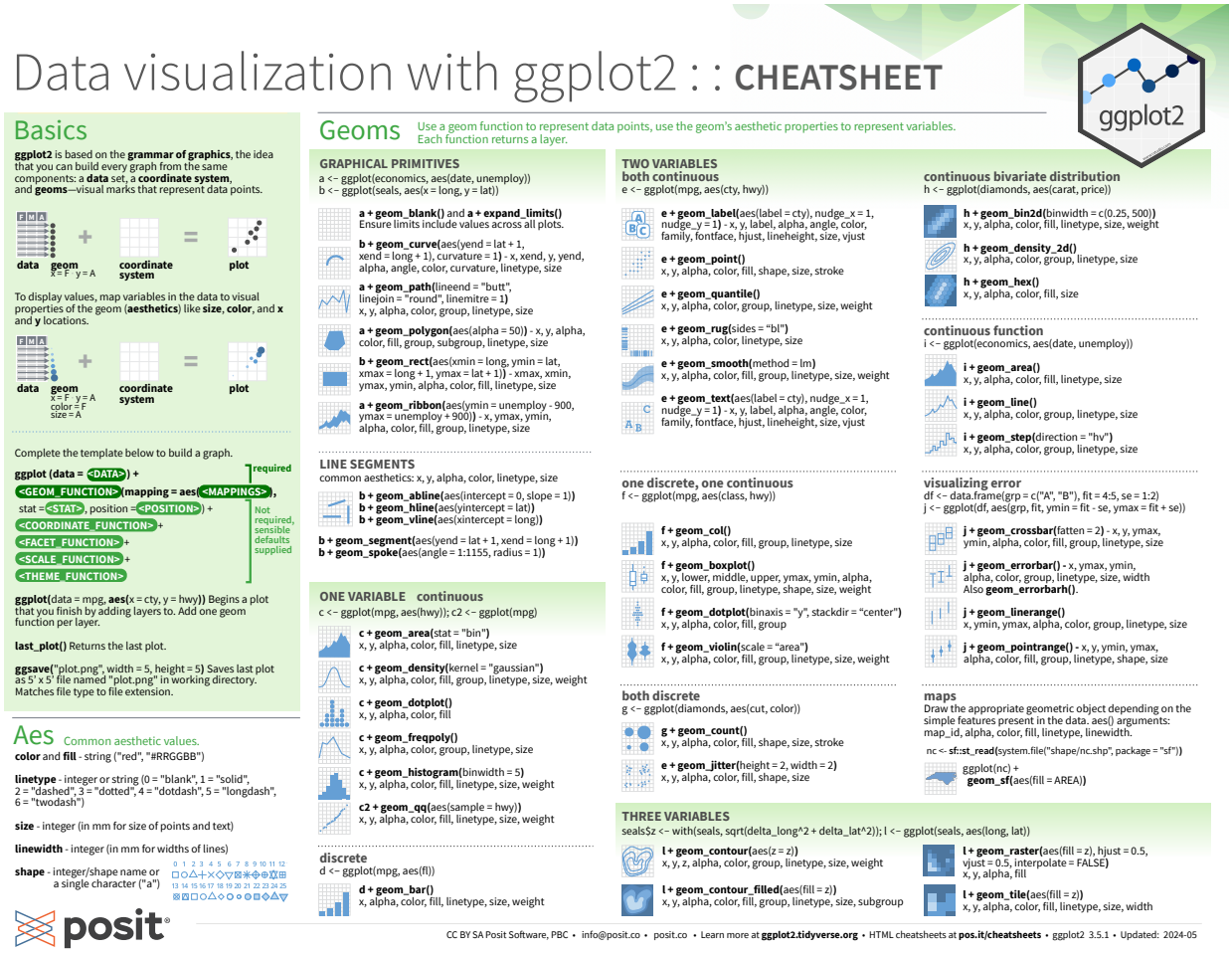
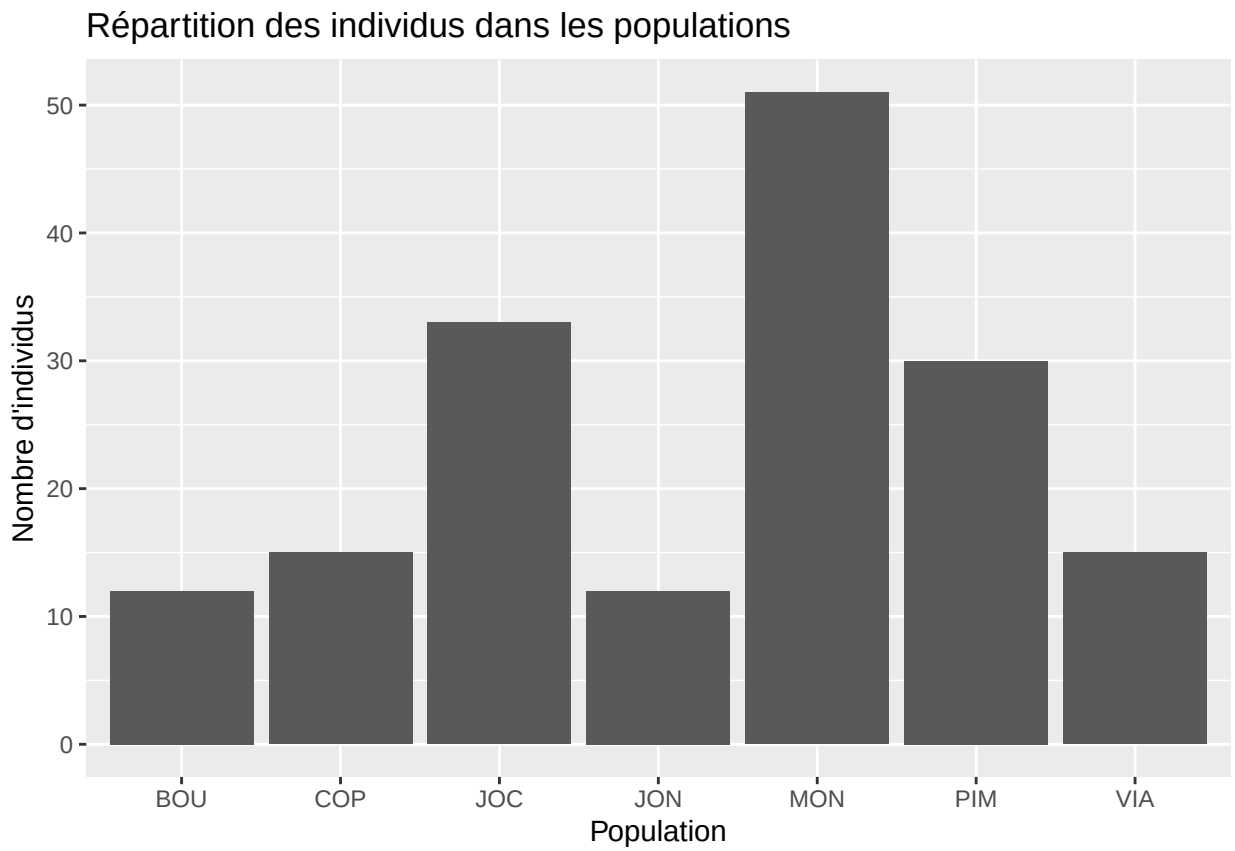


Figure 1: Ecriture fichier Mardown

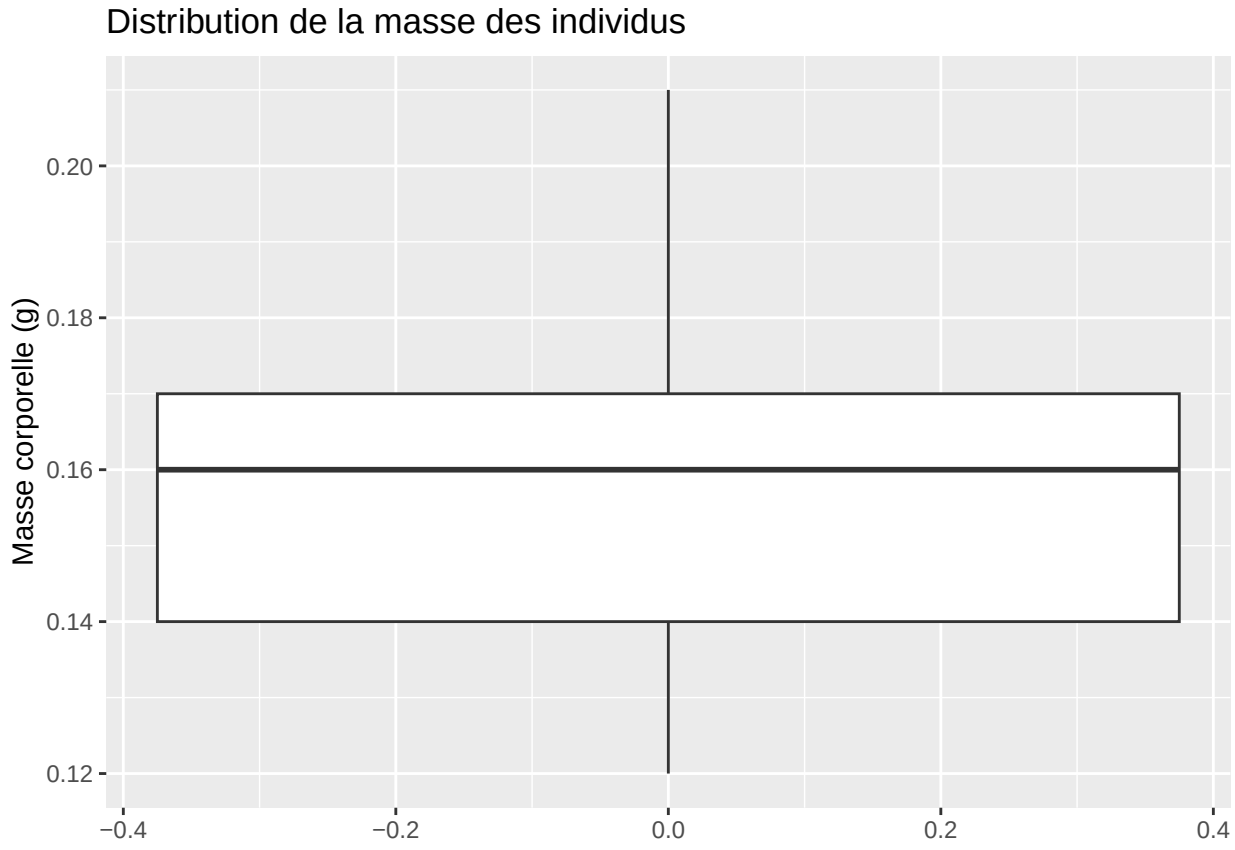
Prenons quelques exemples à partir des sorties graphiques les plus centrales que nous avons abordées au cours des derniers TP:

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

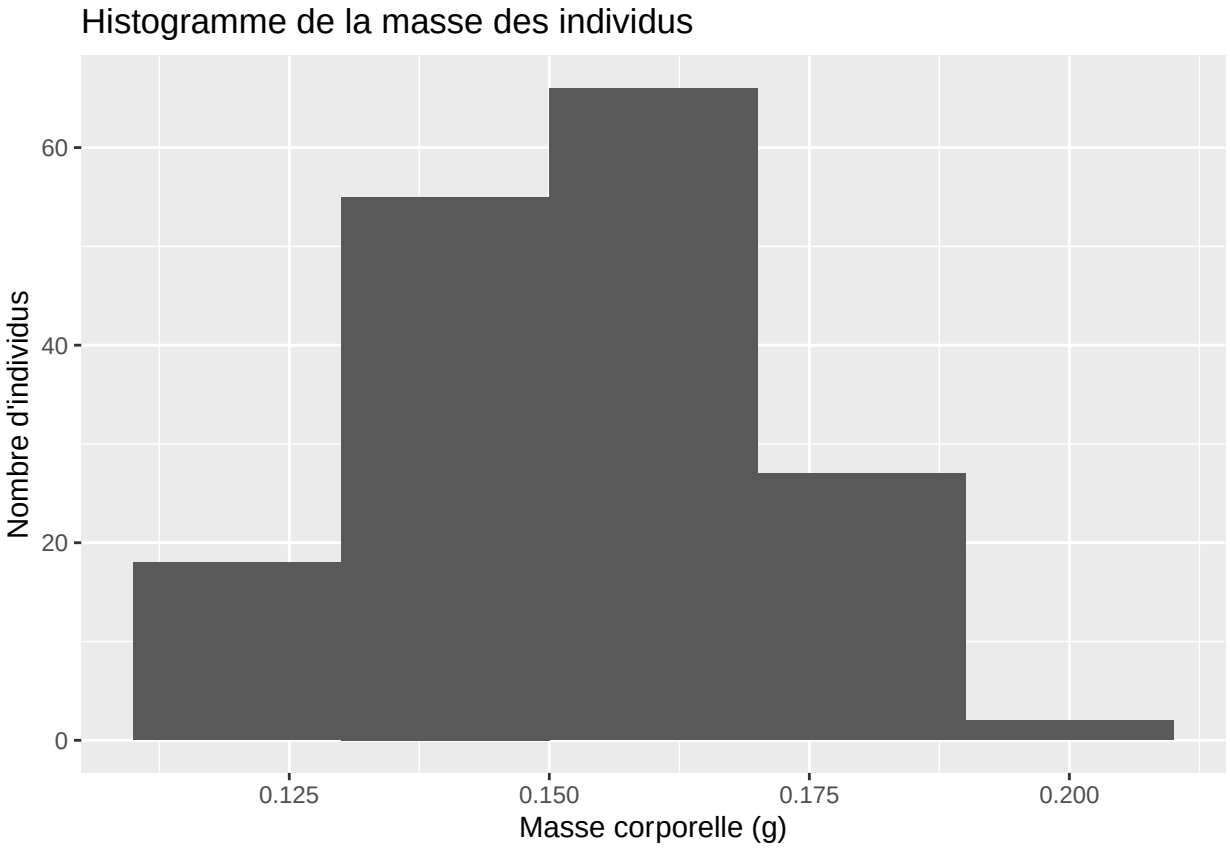
```
library(ggplot2)


ggplot(data = data_lezard, aes(POP)) + # indication du jeu de données d'étude,
  # puis de la variable d'intérêt
  geom_bar() +  # type de graphique
  labs(title = "Répartition des individus dans les populations",
       x = "Population", y = "Nombre d'individus") # légendes
```




```
ggplot(data = data_lezard, aes(y = M_IND)) + #positionnement de la variable d'intérêt sur
  # l'axe des ordonnées
  geom_boxplot() +
  labs(title = "Distribution de la masse des individus", y = "Masse corporelle (g)")
```




```
ggplot(data = data_lezard, aes(M_IND)) +
    geom_histogram(binwidth = 0.02) + # largeur des classes d'histogramme
    labs(title = "Histogramme de la masse des individus",
         x = "Masse corporelle (g)", y = "Nombre d'individus")
```



POUR ALLER PLUS LOIN

Il existe dans R une collection de packages visant à simplifier les analyses de données sous R: le tidyverse. Le tidyverse est une façon très particulière d'aborder R, il vise en effet à remplacer toutes les fonctions usuelles de R par de nouvelles et change complètement la syntaxe d'écriture du code R. L'idée est de fournir un mode d'écriture du code plus facilement lisible et plus intuitif pour des néophytes. Cela a des avantages pour un premier apprentissage mais aussi des inconvénients: il vous sera moins aisé de switcher vers un autre langage de programmation depuis la syntaxe (très) particulière du tidyverse, et ces changements se font au prix d'une faible perte d'efficacité (lorsque l'on compare par exemple les fonctions "purrr" à celles de la famille "apply", ou en général pour toutes les commandes du tidyverse par rapport à des syntaxes créées pour gagner en efficacité comme celle du package "data.table"). Une autre critique majeure est que l'accumulation des différentes syntaxes utilisées par le tidyverse (et potentiellement d'autres syntaxes) ne rend pas forcément le code plus simple à parcourir.

MISE EN APPLICATION

Reprenez les graphiques produits au cours des TPs en utilisant les outils ggplot.

Gestion des erreurs, avertissements et blocage sous R

Nous allons tout d'abord revenir sur les erreurs et messages d'avertissement dans R. Comme vous avez déjà pu probablement vous en apercevoir, R retourne des messages d'avertissement ou d'erreur en rouge dans la console lorsque vous essayez respectivement de réaliser une action potentiellement erronée ou interdite par la commande/fonction que vous utilisez. Voici des exemples ci-dessous:

```
mean(c("a", "b", "c"))
```

```
## Warning in mean.default(c("a", "b", "c")): l'argument n'est ni numérique, ni
## logique : renvoi de NA
```

```
## [1] NA
```

```
as.numeric(c("a", "b", "c"))
```

```
## Warning: NAs introduits lors de la conversion automatique
```

```
## [1] NA NA NA
```

```
# avertissement: création de valeurs manquantes car opérations nécessitant des
# valeurs numériques


log(-1)
```

```
## Warning in log(-1): Production de NaN
```

```
## [1] NaN
```

```
sqrt(-1)
```

```
## Warning in sqrt(-1): Production de NaN
```

```
## [1] NaN
```

```
# avertissement: création de valeurs manquantes car opérations interdites,
# logarithme et racine carré d'une valeur négative (en dehors de l'ensemble de
# définition de la fonction)







median(data_lezard)
```

```
## Error in median.default(data_lezard): données numériques nécessaires
```

```r
# Erreur: valeurs numériques nécessaires (moyenne sur un tableau de données:
# aucun sens)

median(data_lizard$SVL_MOTHERS)
```

```
## Error in eval(expr, envir, enclos): objet 'data_lizard' introuvable
```

```r
# Erreur: faute de frappe dans le nom de la variable

data_lezard[data$SVL_MOTHERS == 65]
```

```
## Error in data$SVL_MOTHERS: objet de type 'closure' non indiçable
```

```r
# Erreur: sélection sans indiquer sur quels éléments du tableau elle s'opère
# (ligne, colonne), il faudrait positionner une virgule après la condition pour
# sélectionner les lignes

data_lezard[, "SVL_MOTHERS"] + data_lezard[, "SEX"]
```

```
## Error in data_lezard[, "SVL_MOTHERS"] + data_lezard[, "SEX"]: argument non numérique pour un opérateu
```

```r
# Erreur: tentative d'additionner des valeurs numériques et chaînes de
# caractères
```

Dans le cas où vous n'arrivez pas à déterminer l'origine de l'erreur malgré le message fourni par R, commencez par utiliser la fonction help() pour essayer de voir si vous n'avez pas commis une erreur dans l'utilisation de la fonction. Si cela ne s'applique pas à votre cas (utilisation d'une suite de commande plutôt qu'une fonction prédéfinie) ou si vous n'arrivez pas à trouver l'origine de l'erreur à partir de l'aide, vous pouvez également vous appuyer sur les forums d'entraide (le plus actif étant stackoverflow, en anglais) ou sur certains sites spécialisés (ThinkR, STHDA).

Il existe toutefois des cas plus complexe dans lesquels l'erreur ce situe dans un bloc de code contenant de nombreuses fonctions et commandes. Dans ce cas il est très important d'agir séquentiellement, en découpant et testant chaque portion du code utilisé afin d'identifier la source de l'erreur. Voici un exemple concret:

```r
mode <- function (vect) { # argument de la fonction: vecteur
  names.POSIXlt(                # récupère le nom sur une variable avec identifiant
    which.max(                  # récupère la (ou les) positions de la valeur maximale
      table(                    # table des effectifs du vecteur
        vect)
      )
    )
}


mode(data_lezard$SEX)
```

```
## Error in x$year: $ operator is invalid for atomic vectors
```

```r
# d'où provient l'erreur ?

table(data_lezard$SEX)
```

```
##
##  f  m
## 80 88
```

```r
# ok

which.max(table(data_lezard$SEX))
```

```
## m
## 2
```

```r
# ok

names.POSIXlt(which.max(table(data_lezard$SEX)))
```

```
## Error in x$year: $ operator is invalid for atomic vectors
```

```r
# not ok: l'erreur provient de la fonction names.POSIXlt()

?names.POSIXlt
# il s'agit de la mauvaise fonction ! On a confondu avec la fonction "names()"
```

Il est aussi possible d'avoir recours à l'intelligence artificielle (générative), celle-ci est assez performante quand il s'agit d'interpréter ou d'écrire des lignes de codes tant que l'on reste dans un niveau de complexité raisonnable. Par exemple, reprenez l'exemple ci-dessus et copiez le code correspondant à la fonction (avec l'erreur) dans ChatGPT: l'IA détecte immédiatement l'erreur, vous propose une correction de code, donne une explication de chaque ligne de code et suggère même une voie d'amélioration (pour les vecteurs contenant plusieurs valeurs avec la même fréquence maximale). Dans un cas simple comme celui-ci l'outil se révèle particulièrement efficace !
Faisons un autre essai, en demandant cette fois-ci de réaliser un code pour une fonction précise, par exemple: "Comment coder sur R un fonction calculant le moment d'ordre de 3 d'une distribution numérique ?". Là encore l'outil fonctionne très bien: il nous donne un rappel de ce qu'est un moment d'ordre 3 et nous propose une fonction R pour calculer cela.

Il ne faut pas oublier toutefois de garder un esprit critique fort lorsqu'on utilise un outil comme ChatGPT. Si celui-ci est très efficace pour des commandes simples, il peut s'avérer beaucoup plus faillible sur des exemples complexes. Etant entraîné sur des données pré-existantes et étant d'autant plus efficace qu'il a été confronté fréquemment à un cas de figure, il sera forcément sensible à des problèmes nouveaux ou peu étudiés. Il peut également être difficile de l'utiliser sur des cas concrets, car cet outil n'est pas construit pour saisir toutes les spécificités de votre jeu de données et pourrait manquer certaines informations clés. Enfin, sur des problèmes complexes, l'outil a tendance à s'appuyer fortement sur les fonctions créés par d'autres utilisateurs (paquets) et a le défaut d'appeler parfois un grand nombre de dépendance pouvant générer des conflits mais aussi un moindre contrôle sur les actions effectués par le code.

Il convient donc d'utiliser ce genre d'outil avec certaines bonnes pratiques: il faut leur donner des consignes claires et précises, si possible génériques (en évitant d'utiliser des dénominations propre à notre cas particulier), toujours bien relire et vérifier ligne par ligne le code qui est donné, rester critique sur ses propositions en se demandant s'il n'y a pas des alternatives plus efficaces ou plus simple et lisible, et bien sûr ne pas

hésitez à pointer les erreurs que celui-ci réalise (en se basant sur d'autres ressources techniques comme l'aide incluse dans R ou les forums/sites spécialisés).

POUR ALLER PLUS LOIN

Il existe d'autres IA génératives pouvant aider aux analyses statistiques sous R comme Copilot développé par Github, R wizard pour de la suggestion de code sur R, voire même "Data Analyst" qui vous permet de gérer des analyses complexes en utilisant directement votre jeu de données. Ce dernier outil est à utiliser avec beaucoup de prudence, car si ChatGPT est très performant pour écrire du code, il l'est beaucoup moins pour construire un fil d'analyse statistique logique. Attention également à ne jamais fournir des données sensibles à ce genre d'outil gratuit car vos données seront très probablement récupérées. Ces outils ne sont toutefois pas, pour la plupart, en libre accès et s'utilise souvent via des licences payantes.

EXERCICE

- Vous pouvez trouver ci-dessous un portion de code contenant de nombreuses erreurs, identifiez ces erreurs et corrigez-les. Expliquez ce que réalise ce code.

cat_taille = as.character(SVL_IND)
table(cattaille)
barplot(cat_taille,
main = "Nombres d'individus par valeurs de taille",
xlab = "Tailles mesurées",
ylab = "Nombres d'individus"
)
rm(cat_taille)
summary(data_lezard[!dplicated(data_lezard$ID_MOTHERS), M_MOTHERS])
hist(
data_lezard[!duplicated(data_lezard$ID_MOTHERS)]$M_MOTHERS,
main = "Nombres de mères par classe de masse (histogramme)",
xlab = "Masse corporelle (g)",
ylab = "Effectifs
)
boxplot(
data_lezard[!duplicated(data_lezard$ID_MOTHERS),]$M_MOTHERS,
main == "Distribution des masses corporelles des mères (boxplot)"
ylab = "Masse corporelle (g)"
)

```r
cat_taille = as.character(data_lezard$SVL_IND)

table(cat_taille)

barplot(
    table(cat_taille),
    main = "Nombres d'individus par valeurs de taille", xlab = "Tailles mesurées",
    ylab = "Nombres d'individus"
)

rm(cat_taille)
```

```r
summary(
    data_lezard[!duplicated(data_lezard$ID_MOTHERS),
        "M_MOTHERS"]
)


hist(
    data_lezard[!duplicated(data_lezard$ID_MOTHERS),
        ]$M_MOTHERS, main = "Nombres de mères par classe de masse (histogramme)",
    xlab = "Masse corporelle (g)", ylab = "Effectifs"
)

boxplot(
    data_lezard[!duplicated(data_lezard$ID_MOTHERS),
        ]$M_MOTHERS, main = "Distribution des masses corporelles des mères (boxplot)",
    ylab = "Masse corporelle (g)"
)



# Ce code permet de décrire la distribution des classes de taille et de la
# masse des mères parmi les individus étudiés, numériquement (table des
# effectifs / résumé statistique de la distribution continue) et graphiquement
# (diagramme en barres / histogramme et boxplot)
```

Bilan

Les principaux paramètres avancés **graphiques** à retenir sont les suivants:

- **col**: couleurs des figurés
- **pch / lty**: types de figuré (points / lignes)
- **cex**: taille de police