Aides pratiques aux biostatistiques pour R.studio
License 2 sciences de la vie AMU

Pour tout feedback: igor.cano@etu.univ-amu.fr

Sommaire :

I - Syntaxe :

Précisions :

- Boîte de dialogue = fenêtre en bas à droite, endroit où l'on voit les résultats.
- Fenêtre graphique en bas à droite, on y retrouve les graphiques.
- En haut à droite, on voit toutes les variables et fichiers importés.
- En haut à gauche se situent nos lignes de codes et les différents tableaux que l'on crée.

On peut soit tout écrire à la suite, soit sauter des lignes, soit même espacer la syntaxe.
Exemple :

```
plot(fichier$col1,
fichier$col2[fichier$col2=="info1"],xlab="AHH",ylab="truc",main="prout",
col=c("blue","green","red"))
OU
plot(fichier$col1,
     fichier$col2[fichier1$col2=="info1"],
     xlab="AHH",
     ylab="truc",
     main="prout",
     col=c("blue","green","red"))
OU
plot(fichier$col1,
     fichier$col2[fichier1$col2=="info1"],
     xlab="AHH",
     ylab="truc",
     main="prout",
     col=c("blue",
           "green",
           "red"
           )
     )
```

Attention, dans les cas 2 et 3, il faut strictement respecter les tabulations:

```
plot(truc,
.....machin=c("a", # espacement de 5 caractères
............."b" # espacement de 13 caractères
.............)
.....)
```

Pour ajouter des notes, toujours utiliser ceci : #
Vérifier à la syntaxe des fonctions: majuscules, minuscules,ponctuation, syntaxe dans la fonction.

Utiliser une partie spécifique d'un fichier :
Permet de prendre certaines données d'une partie précise d'un fichier.

```
fichier$colonne_recherché
fichier$colonne_recherché[fichier$colonne_recherché=="info de cette
colonne"]
# un fichier fleur avec 1 colonne sur la taille de la fleur et une
colonne sur l'engrais utilisé,
# vous voulez voir que les tailles, vous faites :
fleur$taille
# par contre vous voulez voir la taille des fleur pour l'engrais purin,
# vous faites :
fleur$taille[fleur$engrais=="purin"]
```

Gardez en tête que l'on peut écrire son code de plusieurs façon :

```
plot(fichier$colonne1 ~fichier$colonne2)
# OU
plot(colonne1 ~colonne2, data=fichier)
```

$ signifie que l'on va regarder dans le fichier. ~ Signifie que l'on va faire notre fonction selon 2 données qui se suivent :

- Je regarde la taille en fonction du sexe → `taille ~sexe`
- Je regarde la taille des hommes dans le fichier taille → `taille$hommes`

Les différents types de variables et syntaxe particulière :
Pour retourner une commande (l'afficher dans la boîte de dialogue), utiliser `View()`, `res()` ou `return()`.

```
NA # valeur manquante
TRUE FALSE # logique ou booléen
numeric # réel ou entier (1,2 - 2)
complex # π, e, i...
character # caractères
\\ # backslash : " \ "
\n # saut de ligne
\t # tabulation
# # commentaires
```

Concaténer plusieurs variables :
Si vous voulez par exemple colorer votre graphique avec 3 couleurs, il ne suffit pas de simplement marquer les couleurs à la suite, car la fonction va mal comprendre. A la place utiliser `c()`.

```
plot(truc, machin; color="green", "red", blue") # -> FAUX
# faire
plot(truc, machin, color= c("green","red","blue") )
```

`c()` permet de rassembler plusieurs informations en 1 seule. Peut être appliquer à des fichiers, du texte, et à peu près tout du temps que c'est bien écrit.

Assigner des variables :
Dans python, vous faites `variable=fonction`. Dans R.studio c'est un peu plus compliqué. Vous avez plusieurs choix:

```
variable <- fonction
variable = fonction
variable -> fonction
```

II - Importer un fichier :

Il y a 2 façons d'importer des données. La façon compliqué, à savoir écrire la direction COMPLÈTE du fichier comme ci-après :

```
nom <- read.type1("C:/[...]/fichier.type2", header, sep, dec)
type 1 = type de fichier: tableaux -> table
                  texte, FASTA... -> csv
type 2 = format du fichier # .txt généralement
header = nom de colonne, "TRUE ou "FALSE"
sep = séparateur entre 2 colonnes, "." ou " " ou "\t"  (tabulation)
```

Ou alors vous faites la version simple :

- Allez dans la fenêtre en haut à droite ;
- `Import Dataset` ;
- `From base to text` ;
- Choisissez les différents paramètres (séparateur, décimale, titre…) ;
- Invoquez le fichier dans les lignes de codes comme n'importe quelle autre fonction.

Pour arrêter R, utilisez `q()`.
Pour interrompre un processus en cours: `INTERRUPT`.
Attention; écrivez le dans la boîte de dialogue.

III - Arithmétiques, vecteurs, matrices :

Arithmétique :
Les plus simples sont l'addition et la soustraction. La division est assez simple aussi.
Pour le reste :

```
x^ y # x puissance y
x%/% y # division euclidienne, renvoie la partie entière
x%% y # division euclidienne, renvoie le reste
sqrt(x) # racine carré
sum(x) # somme ∑, possibilité d'ajouter une condition ( sum(x>12) )
```

Vecteurs :

```
c(x, y, ...) # création d'un vecteur
seq(début, fin, by=pas) # séquences de chiffres d'un chiffre à un autre
avec un pas, par exemple les chiffres 2 à 2.
```

Les fonctions et conditions peuvent être applicable au sein du vecteur
Exemple : afficher les valeurs supérieur à X du vecteur.

```
x[x>X]
```

Matrices :

```
matrix(valeurs, nombre de lignes,nombre de colonnes)
# Exemple : matrice de 1 à 5 répartie sur 2 lignes et 5 colonnes
matrix(1:5,2,5)
```

IV - Opérations sur les données :

Boucles :
R.studio à une syntaxe particulière pour faire des boucles if, for et while:

```
for(i in variable) {commandes1,
                    commande2, ...
                   }
while (condition) {commandes}
repeat {commandes} # répète en boucle infini les commandes
if (condition) {
    commandes
} else {
 commandes si la condition if n'est pas respecté (else) }
```

Créer un tableau :

```
data.frame( "titre colonne" = colonne, ... )
# Attention, dans data.frame, il faut préciser toutes les informations,
même les titres.
# Exemple :
data.frame("sexe" = c("homme", "femme"),  # colonne 1
           "taille" = taille$hommes,   # colonne 2
           "taille" = taille$femmes) # colonne 3
```

Faire un subset :
Permet de couper notre fichier principal en sous-tableaux qui peut contenir que certaines colonnes ou parties de colonnes

```
nom <- subset( fichier, plage de données) # indiquer le fichier à
regarder ainsi que la colonne ou plage de données
# exemple :
iris_Ve <- subset(iris, Species=="versicolor")
```

Faire un agrégat :
Fonctionne comme `subset()`, mais donne directement le résultat de notre ligne de code. Contrairement à subset, aggregate peut séparer des données d'un fichier mais aussi y appliquer des fonctions de statistiques simples.

```
aggregate(colonne1 ~colonne2, data=fichier, fonction_appliquer)
# Exemple : voir la moyennes des 4 conditions d'un fichier :
aggregate(colonne1 ~colonne2, data=fichier, mean)
# Donnera le tableau avec les moyenne de chaque conditions du fichier
```

V - Visualiser des données :

Visualiser les données d'un fichier :

```
View(fichier)  # visualiser le fichier entier dans une autre fenêtre
summary(fichier) # quelques infos dans la boîte de dialogue
table(fichier) # donne un tableau dans la fenêtre de dialogue
```

Visualisation personnalisé des données :
La fonction `summary()` donne plusieurs informations, cependant on peut faire un sommaire personnalisé de nos données.

```
variable <- do.call(data.frame,
                    aggregate(colonne1 ~colonne2,
                    data=fichier,
                    FUN=fonctions(x) c(fonction1(x),
                                       fonction2(x),
                                       fonction3(x)
                                      )
                             )
                   )
names(variable) <- c("nom_données",
                     "nom_fonction1",
                     "nom_fonction2",
                     "nom_fonction3")
print(sommaire)
```

Attention, il est important de préciser le nom de chaque colonne !

Donner le nombre de lignes et de colonnes d'un tableau:

```
nrow(plage de données à regarder)
ncol(plage de données à regarder)
```

Compter des données:
Permet de calculer les effectifs pour certaines conditions:

```
length(fichier$colonne=="variable")
# Exemple: je veux connaitre le nombre d'oiseaux bleus dans un groupe
length(oiseaux$couleur=="bleu")
```

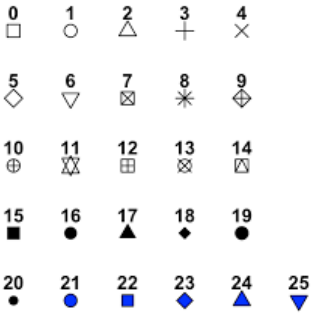
Visualisation de graphiques :

```
hist(données, paramètres) # histogramme
plot(données, paramètres) # histogramme, nuage de point, courbe...
boxplot(données, paramètres) # boite à moustache
stripchart(données, paramètres) # nuage de point à une dimension
barplot(données, paramètres) # histogramme de données qualitatives
pie(données, paamètres) # camenbert
```

Les paramètres sont variées :

- `xlab`, `ylab` et `main` pour les noms.
- `xlim`, `ylim` pour les paramètres de la forme du graphiques.
- `col`, `height`, `width`, `pch` pour les couleurs et formes des données.

`pch` est la forme des points:



`type=" "` donne la forme des données: `"p"` = points,`"l"` = lignes `"b"`= les deux.

VI - Statistiques simples :

```
mean() # moyenne
median() # médiane, si données bilatéral, moyenne=médiane=mode
var() # variance
sd() # écart type
IQR() # écart interquartile (50% des données [25%;75%]
quantile(fichier,%) # calculer Q1,Q2,Q3 et l'écart interquartile
```

VII - Statistiques descriptives :

Loi binomiales :

Trouver la probabilité d'avoir un résultat donné :
`dbinom(x,n,p)` avec x le résultat recherché, n la population et p la probabilité. Exemple :: 10 lancer d'une pièce, on veux 8 piles avec 50% de tomber sur pile:

```
dbinom(8,10,0.5)
```

Trouver la probabilité entre 2 valeurs :
Permet de trouver la probabilité de trouver une valeur dans un intervalle donné. P(x<X<y)=?

```
pbinom(valeur haute,p,lower.tail=TRUE)-pbinom(valeur
basse,p,lower.tail=FALSE)
```

A l'inverse, si vous voulez la probabilité que la valeur ne soit pas dans un certain intervalle:

```
pbinom(valeur basse,p,lower.tail=TRUE)+pbinom(valeur
haute,p,lower.tail=FALSE)
```

Pour les précisions concernant la syntaxe, voir ci-dessous.

Trouvé la probabilité d'avoir au dessous ou en dessous d'un résultat donné :

```
pbinom(x,n,p,lower.tail=?) # x le résultat recherché, n la population,
# p la probabilité et lower.tail le sens du test.
# Faire au moins 3 gains avec 100 tickets de loterie avec 10% de chance:
# P(X > 2 ) = ?
pbinom(2,100,0.10,lower.tail=FALSE)
# lower.tail -> FALSE pour > et TRUE pour <
```

L'inverse de d et pbinom :
Trouvé le résultat pour tel probabilité P(X=?)=x

```
qbinom(x,n,p)
```

Autres fonctions :

```
rbinom(x,n,p) # r = random, donne des valeurs aléatoires suivant la loi binomiale
```

Loi normale :

Exactement les mêmes principes que pour binomiales mais avec autre chose que pile ou face comme choix (quantitatif au lieu de qualitatif).

```
dnorm(x,n,p)
pnorm(x,n,p) # calcul de proportion, fonctionne comme sum() ou length() mais adapté à la loi normale.
qnorm(x,n,p)
rnorm(x,n,p) # valeurs aléatoires suivants la loi normale
```

Trouver un intervalle centré d'un pourcentage donné :
Exemple, on veut savoir l'intervalle qui contient 90% des valeurs d'IMC d'une population :

```
c( qnorm(0.05,n,p), qnorm(0.95,n,p))
```

Attention, le calcul d'un intervalle doit se faire de tel sorte à ce qui est laissé de côté soit équitablement répartie de part et d'autre de la moyenne. 90% c'est l'intervalle de 5% à 95%, 50% c'est l'intervalle entre 25 et 75%...

tests statistiques :

Interprétation des résultats avec les tests statistiques :

```
       Type de test
data:  plages de données
number of successes = x, number of trials = n, p-value = 0.6082
alternative hypothesis: true probability of success is not equal to μ
Nα percent confidence interval:
 inférieur supérieur
sample estimates:
probability of success
             σ
```

Avec x et n comme vu plus haut.
p-value: indique le résultat du test: validé si il est supérieur au taux d'erreur (1 - Nα)

μ est la valeur donnée dans H0
Nα est le niveaux de confiance (généralement 95%)
σ la probabilité de succès de μ si H0 est vraie.

Comparaison valeur théorique avec valeur observé:

```
binom.test(x=valeur observé,n= effectif , p=valeur théorique)
```

Voire la distribution d'un groupe de données par rapport à une valeur seuil :

```
t.test(plage de données, mu=valeur H0)
```

Comparer deux groupes de données :

```
t.test(plage de données 1, plage de données 2, mu= valeur H0,
alternative= H1)
# Si μ1 de la plage de données 1 est de 12 et qu'on veut savoir si la
plage de données est au dessus, on fait mu=12, alternative="lower" car
H0 = μ1>μ2 et H1 = μ1<μ2 .
# Le paramètre alternative est soit plus grand ("upper"), plus
petit("lower") ou les 2 ("two.sided").
```

Trouver un intervalle de confiance :

```
t.test(plage de données, data=fichier)
# On regarde ensuite l'intervalle qu'on donne. On peut préciser sinon
l'intervalle pour une confiance précise.
# Si ça ne suit pas une loi normale ou que ce n'est pas sur, on utilise:
wilcox.test(plage de données, data=fichier)
```

L'intervalle de confiance permet de trouver la significativité du test (erreur de 1ere ou 2eme erreur).

Trouver une variation des données :
Si vous voulez connaitre de quel valeur à quel valeur le fichier varie en moyenne (différent de la variance ou de l'écart interquartile)

```
t.test(plage de donnée, data=fichier)
```

VIII - “Décorations” et autres fonctions :

Nommer un graphique :
Permet de donner un nom aux axes, ainsi qu’un titre.

```
plot(fichier$colonne1 ~fichier$colonne2,
     xlab="titre axe des X",
     ylab="titre axe des Y",
     main="titre du graphique")
```

Ajouter une légende à un graphique :

```
legend("topleft", #lieu de la légende
       legend = c("Setosa", "Versicolor", "Virginica"), # données
       col = c("red", "green", "blue"), #couleurs de chaques données
       lty = 1, # type de ligne de la légende (ligne, pointillé...)
       lwd = 1, # taille de ligne de la légende (1 cm d'épaisseur)
       cex = 0.75,
       title = "Espèces", # titre de la légende
       text.font = 3, # taille du texte
       box.lty = 0)
```

Pour plus d’informations sur la fonction `legend()`, taper `help(legend)` dans la boîte de dialogue.

colorer un graphique :
Obligation d’avoir autant de couleurs que de variables.

```
plot(variable1 ~variable2, col=c("couleur1", "couleur2))
```

Attention: mettre les couleurs en anglais !

Placer plusieurs graphiques dans une même fenêtre :
Permet de placer plusieurs graphiques en même temps, pratiques lorsqu’il faut faire plusieurs graphiques d'affilée.

```
par(mfrow=c(nombre de colonnes, nombre de lignes))
# attention, remettre ensuite par(mfrow=c(1,1)) pour éviter de continuer
à avoir 25 graphiques par fenêtre.
```

Ajouter un titre principal au groupe de graphiques (voir `par(mfrow)`) :

```
mtext("") # Titre général
```

Ajouter des points sur un graphique :

```
points(x=coordonnée_abcisse, y=coordonnée_ordonée,
autres_infos(couleurs,forme...))
```

Ajouter des lignes sur un graphiques :
`abline()` permet de faire des lignes sous plusieurs formes.

```
abline(h=ligne_horizontal)
abline(v=ligne_vertical)
abline(a=12,b=2) # sous la forme
abline(coef)
abline(reg)
```

Pour plus d'informations, tapez `help(abline)` dans la boîte de dialogue en bas à gauche.

Droite de régression :
Permet de voir la tendance dans un nuage de points avec une ligne droite. C'est plus joli et ça aide si le boxplot est peu clair.

```
nom <- lm(fichier$colonne1 ~fichier$colonne2, data)
abline(nom, col = "couleur de la ligne") # couleur en anglais !
```

Fais sur google document avec les modules complémentaires Code blocks et Code syntax.
Auteur du document: CANO Igor (igor.cano@etu.univ-amu.fr)
Version 2 du 28/04/2022.
Encore plus de fonction pour R.studio (par Hubert Raymondaud sur http://revue.sesamath.net/).