Renaudpack2 - tuto

Contents

1 Introduction **2**
- 1.1 Pour l'installer . . . 2
- 1.2 Pour le mettre à jour . . . 2
- 1.3 Pour le charger . . . 2

2 Les fonctions graphiques pour les analyses univariées **3**
- 2.1 Un facteur et des barres d'erreur avec la fonction `barres.plot` . . . 3
- 2.2 Pour rajouter les lettres des post-hoc avec la fonction `anovLetters` . . . 5
- 2.3 Deux facteurs et des barres d'erreur avec la fonction `barres.plot.beside` . . . 6
- 2.4 Faire un tableau récapitulatif avec `Tableau_recap` . . . 9
- 2.5 Des séries temporelles avec `Time.factor.plot` . . . 10
- 2.6 Des jolies couleurs avec les fonctions `Couleur_continue` et `modif_coul` . . . 11
- 2.7 Des diagrammes en barres mais avec des points avec `point.plot` . . . 14

3 Fonctions graphiques pour les analyses multivarées **16**

4 Pour de la manipulation de données **16**
- 4.1 La fonction `BBtransf` . . . 16
- 4.2 La fonction `Classes_def` . . . 16
- 4.3 La fonction `combin.tab` . . . 16
- 4.4 La fonction `combin.tabV0` . . . 16

5 Pour des tâches très spécifiques : **16**
- 5.1 Autour de l'indice de l'integrité de la structure des communautés . . . 16
- 5.2 La fonction `ComStructIndices` . . . 16
- 5.3 La fonction `structure.plot` . . . 16
- 5.4 La fonction `structure.plotV2` . . . 16
- 5.5 La fonction " . . . 16

Dernière mise à jour : 20-03-2018

1 Introduction

Il s'agit d'un tuto pour mes fonctions perso, réunnies dans le package au doux sobriquet de `Renaudpack2`

Depuis peu l'installation est possible via Github, c'est ultra simple en utilisant une fonction du package `devtools`, qu'il faut donc potentiellement installer si ce n'est déjà fait :

1.1 Pour l'installer

```
install.packages("devtools")
#à ne faire que si vous n'avez jamais installé devtools

library(devtools)
devtools::install_github("RenaudJau/Renaudpack2")
```

1.2 Pour le mettre à jour

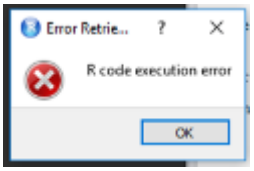
```
install.packages("devtools")
#à ne faire que si vous n'avez jamais installé devtools

library(devtools)
devtools::update_packages("Renaudpack2")
```

1.3 Pour le charger

Comme n'importe quel package, avec `library` :

```
library(Renaudpack2)
```

Si jamais il y a un message d'erreur du style de l'impression écran ci-après :



Je ne sais pas encore d'où ça vient, mais en redémarrant R `Session > Restart R` ou `Ctrl+Shift+F10`, et en refaisant `library(Renaudpack2)` ça semble fonctionner.

2 Les fonctions graphiques pour les analyses univariées

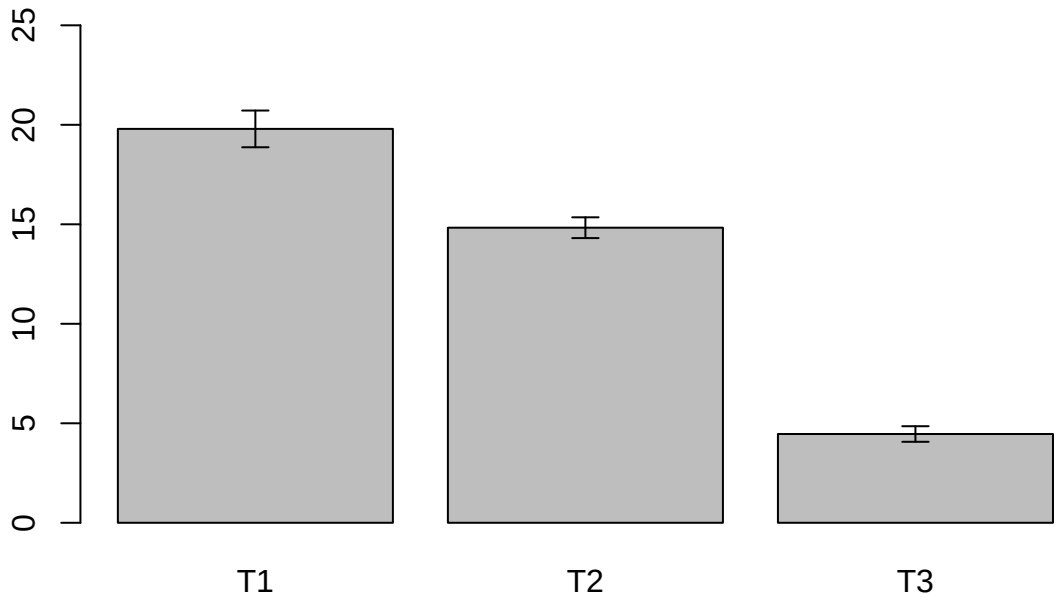
Pour les besoins de ce tuto, on créé des fausses données :

```
biomasse<-c(rnorm(35,20,5),rnorm(35,15,3),rnorm(35,5,3))
traitement<-factor(rep(c("T1","T2","T3"),each=35))
```

2.1 Un facteur et des barres d'erreur avec la fonction `barres.plot`

Le graphique de base :

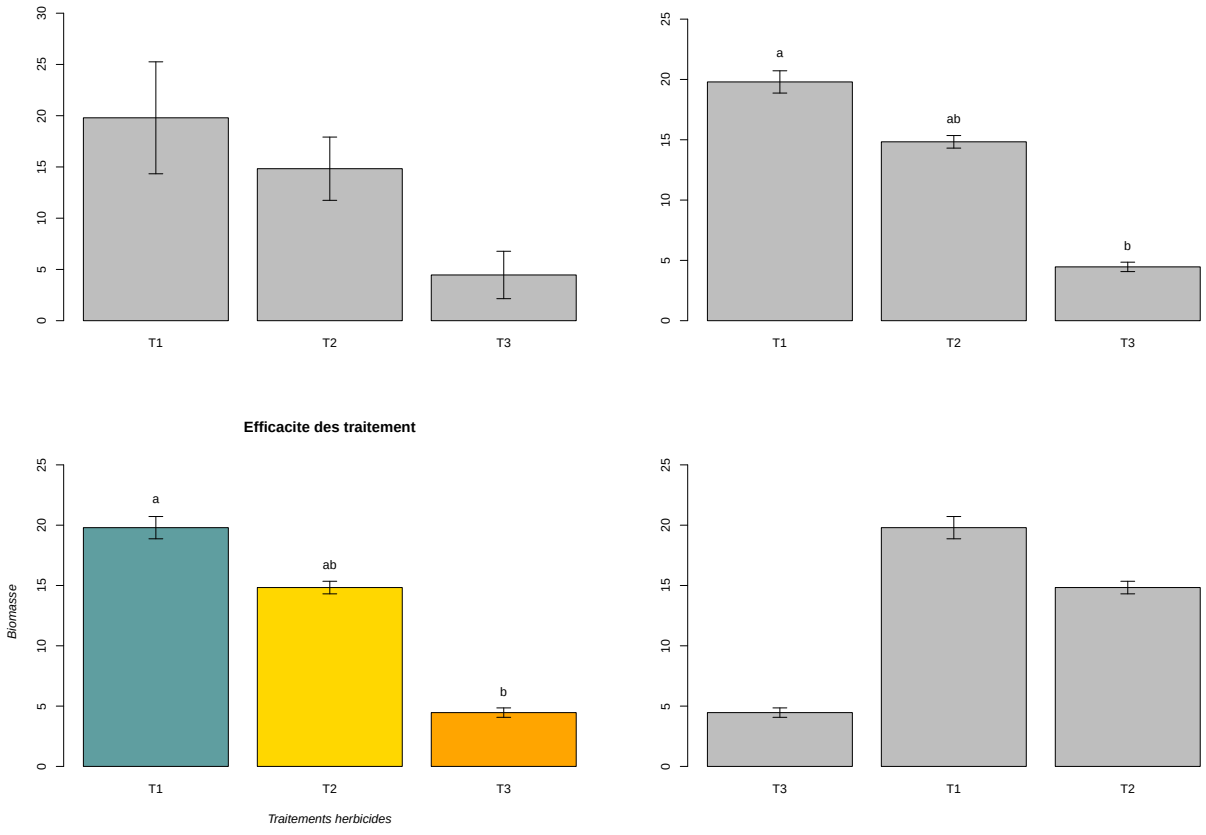
```
barres.plot(biomasse,traitement)
```



Il y a plusieurs choses que l'on peut modifier pour personaliser son graphique :

- changer le type d'erreur représentée
- ajouter des lettres d'un test post-hoc
- changer des paramètres classiques de la fonction `barplot`
- changer l'ordre des modalités (ça ça n'est pas spécifique à la fonction `barres.plot`)

```
par(mfrow=c(2,2)) #pour avoir 4 graphiques sur une même fenêtre
barres.plot(biomasse,traitement,ecart=sd)
#erreur standard par défaut, ici écart-type
barres.plot(biomasse,traitement,lettres=c("a","ab","b"))
barres.plot(biomasse,traitement,lettres=c("a","ab","b"),
col=c("cadetblue","gold","orange"),
xlab="Traitements herbicides",ylab="Biomasse",font.lab=3,
main="Efficacite des traitement")
traitement2<-factor(traitement,levels=c("T3","T1","T2"))
barres.plot(biomasse,traitement2)
```




```
par(mfrow=c(1,1)) #Pour restaurer les paramètres graphiques
```

2.2 Pour rajouter les lettres des post-hoc avec la fonction `anovLetters`

Si vous ne voulez pas vous embêter à calculer vous même les lettres des tests post-hoc, et si et uniquement si vous êtes dans les conditions d'utilisation d'une ANOVA à un facteur, alors on peut faire ça :

```
anov<-aov(biomasse~traitement)
summary(anov)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## traitement    2   4285  2142.3   143.8 <2e-16 ***
## Residuals   102   1520    14.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

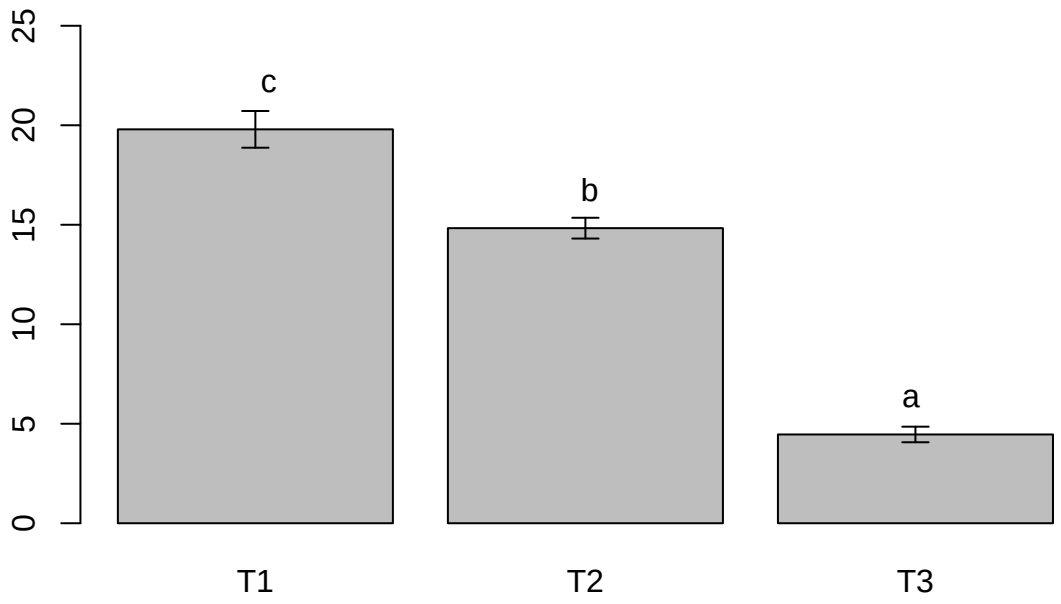
OK, il y a un effet significatif, du coup `anovLetters` renvoie la liste des lettres...

```
lettresPH<-anovLetters(VAR = biomasse,FAC = traitement)
lettresPH
```

```
## [1] "   c" "  b " " a  "
```

...que l'on peut utiliser directement sur `barres.plot` :

```
barres.plot(biomasse,traitement,lettres = lettresPH)
```



*Note : possibilité de changer le seuil alpha en utilisant l'argument `ALPHA`.

2.3 Deux facteurs et des barres d'erreur avec la fonction `barres.plot.beside`

Pour les besoins de ce tuto, on créé des fausses données, avec 2 facteurs, l'âge et le sexe, et une variable, la taille :

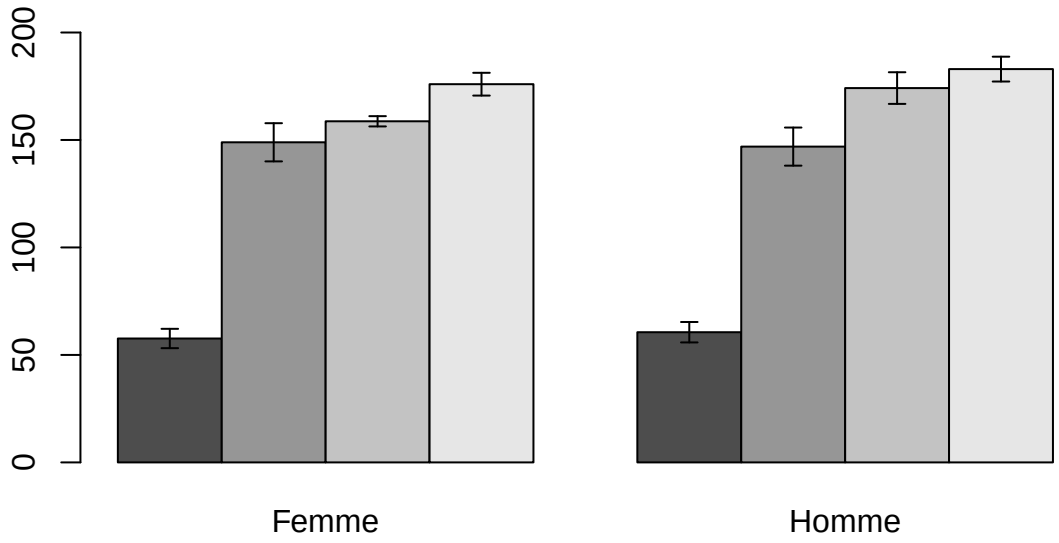
```
Age=factor(rep(c(2,10,20,30),each=10))
Sexe=factor(rep(rep(c("Homme","Femme"),5),4))
Taille=c(rnorm(5,60,7),rnorm(5,55,7),rnorm(5,145,15),rnorm(5,129,15),rnorm(5,175,15),
         rnorm(5,165,15),rnorm(5,175,15),rnorm(5,165,15))
```

En gros ça donne ça :

Age	Sexe	Taille
2	Homme	61.62692
2	Femme	69.19582
2	Homme	53.72726
2	Femme	67.46225
2	Homme	78.68693
2	Femme	46.63852

Le graphique de base :

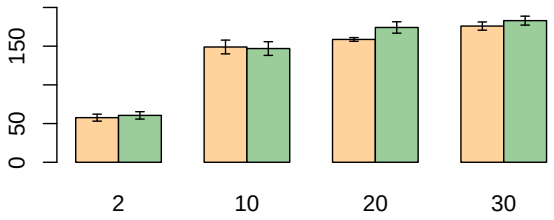
```
barres.plot.beside(Taille,Sexe,Age)
```



On peut aussi modifier quelques petits trucs :

- si on inverse les facteurs, avec des couleurs

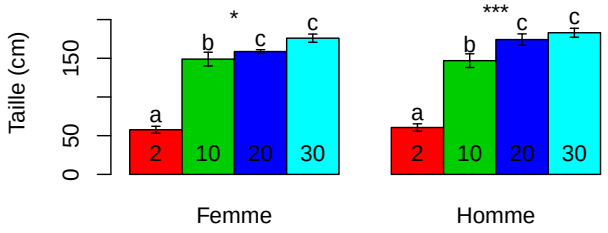
```
barres.plot.beside(Taille,Age,Sexe,col=c("burlywood1","darkseagreen3"))
```



- avec les annotations qui vont bien

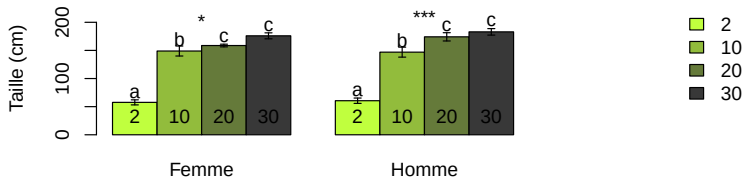
Attention, les lettres et étoiles sont à définir soit même, c-à-d après tests statistiques, ici pour l'exemple, c'est de l'aléatoire...), et avec des couleurs simples (mais moches...)

```
barres.plot.beside(Taille,Sexe,Age,POSI="bottom",
lettres=c("a","b","c","c","a","b","c","c"),
                   etoiles=c("*","***"),ylab="Taille (cm)",col=2:5)
```



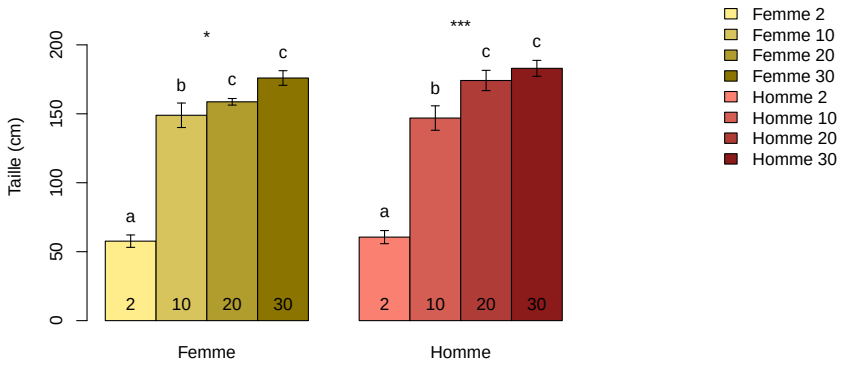
- avec des couleurs choisies avec la fonction `colorRampPalette` et une légende

```
par(mfrow=c(1,2))
coul<-c(colorRampPalette(colors=c("olivedrab1", "grey22"))(4))
barres.plot.beside(Taille,Sexe,Age,POSI="bottom",
                   lettres=c("a","b","c","c","a","b","c","c"),
                   etoiles=c("*","***"),col=coul,ylab="Taille (cm)")
plot(1,type="n",axes = F,ann = F)
legend("topleft",levels(Age),fill=coul,bty="n")
```



- avec des couleurs pour chaque combinaison de modalités

```
par(mfrow=c(1,2))
coul2<-c(colorRampPalette(colors=c("lightgoldenrod1", "gold4"))(4),
         colorRampPalette(colors=c("salmon", "firebrick4"))(4))
barres.plot.beside(Taille,Sexe,Age,POSI="bottom",
                   lettres=c("a","b","c","c","a","b","c","c"),
                   etoiles=c("*","***"),col=coul2,ylab="Taille (cm)")
plot(1,type="n",axes = F,ann = F)
legend("topleft",paste(rep(levels(Sexe),each=length(levels(Age))),
                       rep(levels(Age),2),sep=" "),fill=coul2,bty="n")
```



2.4 Faire un tableau récapitulatif avec `Tableau_recap`

Cette fonction n'est pas graphique, elle donne les informations de `barres.plot` sous forme d'un tableau. Elle a peu d'utilité en dehors d'un document Rmarkdown.

```
Tableau_recap(VAR = biomasse,FAC = traitement,ROUND = 2)
```

```
##    Modalites Nombre Moyennes Erreur
## T1        T1     35    19.79   0.92
## T2        T2     35    14.83   0.52
## T3        T3     35     4.46   0.39
```

Pour que ce soit plus joli, il est cosneillé d'utiliser la fonction `kable`du package `knitr` (pour l'installer : `install.packages("knitr")`).

```
kable(Tableau_recap(VAR = biomasse,FAC = traitement,ROUND = 2))
```

	Modalites	Nombre	Moyennes	Erreur
T1	T1	35	19.79	0.92
T2	T2	35	14.83	0.52
T3	T3	35	4.46	0.39

Si on fait un test posthoc, on peut rajouter les lettres dans le tableau :

```
anov<-aov(biomasse~traitement)
summary(anov)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## traitement    2   4285  2142.3   143.8 <2e-16 ***
## Residuals   102   1520    14.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
TukeyHSD(anov)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = biomasse ~ traitement)
##
## $traitement
##             diff        lwr        upr   p adj
## T2-T1  -4.964996  -7.159451  -2.770542 1.4e-06
## T3-T1 -15.333038 -17.527493 -13.138584 0.0e+00
## T3-T2 -10.368042 -12.562497  -8.173588 0.0e+00
```

```
#les lettres sont donc a, b et c :
kable(Tableau_recap(VAR = biomasse,FAC = traitement,ROUND = 2,
              LETTRES = c("a","b","c")))
```

	Modalites	Nombre	Moyennes	Erreur	Lettres
T1	T1	35	19.79	0.92	a
T2	T2	35	14.83	0.52	b
T3	T3	35	4.46	0.39	c

2.5 Des séries temporelles avec `Time.factor.plot`

Pour les besoins de ce tuto, on créé des fausses données :

```
tim<-rep(c(1:4),each=6)
fac<-factor(rep(rep(c("A","B"),each=3),4))
vari<-c(rnorm(3,5,2),rnorm(3,3,2),rnorm(3,3,2),rnorm(3,8,2),
        rnorm(3,8,2),rnorm(3,10,2),rnorm(3,16,2),rnorm(3,12,2))
```

En gros ça donne ça :

tim	fac	vari
1	A	5.838619
1	A	3.849189
1	A	5.554638
1	B	2.826740
1	B	6.716423
1	B	3.203413

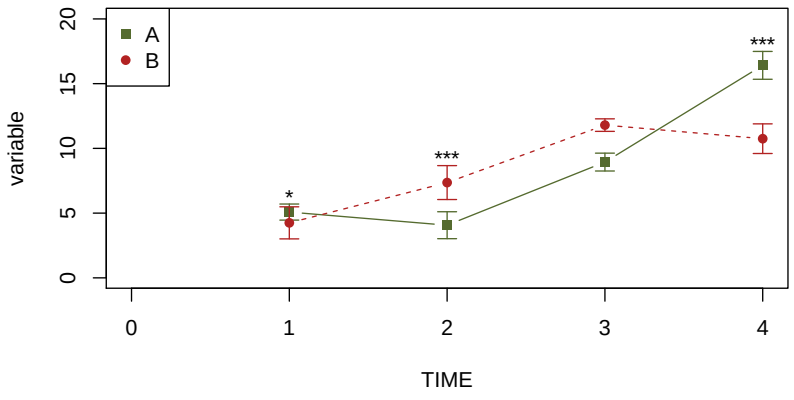
Utilisation de la fonction : *Pourquoi ce message d'erreur? Bonne question.. à résoudre..*

```
Time.factor.plot(tim,fac,vari,etoiles=c("*","***","","***"),
                 pch = c(15,16),couleur = c("darkolivegreen","firebrick"),
                 lty = c(1,2))
```

```
## Warning in if (couleur == c(0)) rep(1, length(levels(factor(FACTOR)))) else
## couleur: la condition a une longueur > 1 et seul le premier élément est
## utilisé
```

```
## Warning in if (pch == c(0)) rep(1, length(levels(factor(FACTOR)))) else
## pch: la condition a une longueur > 1 et seul le premier élément est utilisé
```

```
## Warning in if (lty == c(0)) rep(1, length(levels(factor(FACTOR)))) else
## lty: la condition a une longueur > 1 et seul le premier élément est utilisé
```



2.6 Des jolies couleurs avec les fonctions `Couleur_continue` et `modif_coul`

`Couleur_continue` permet de transformer une variable en palette de couleur, utile pour ajouter une 'dimension' à un plot en 2D

Par exemple, si on a des données (fausses ici..) avec la couverture végétale en fonction de l'altitude, et une information sur la taille des plantes :

taille	recouvrement	altitude
89.2	99	467
18.0	98	1257
40.1	99	1955
90.6	96	2287
86.0	96	1920
83.9	96	1397

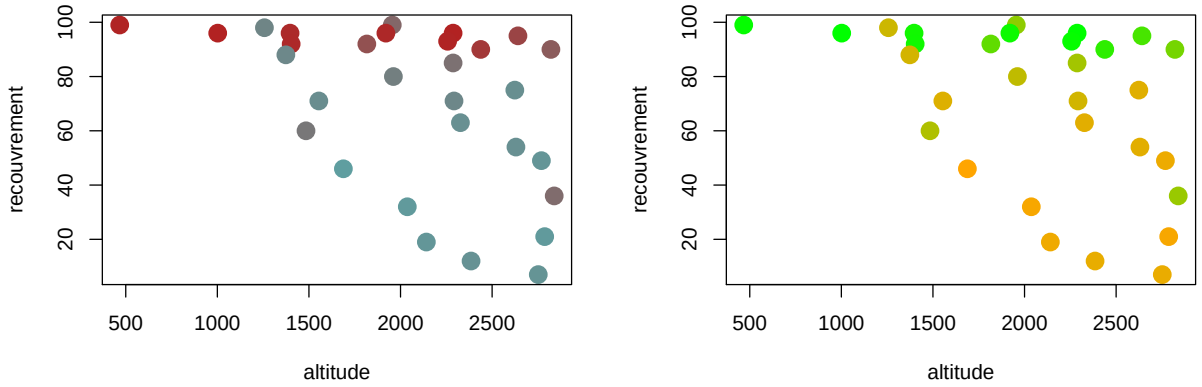
On peut créer une couleur pour chaque valeur de taille (ici avec des couleurs plus froides pour les tailles les plus petites) :

```
coul_taille<-Couleur_continue(VAR = taille,COLORS = c("cadetblue","firebrick"))
coul_taille
```

```
##  [1] "#B02424" "#6F8586" "#836768" "#B22222" "#AD2828" "#AC2A2A" "#B22222"
##  [8] "#AD2929" "#925151" "#B02424" "#A33838" "#9A4445" "#8B5B5B" "#7C7172"
## [15] "#737F80" "#6D8789" "#688F90" "#689092" "#689092" "#649597" "#806C6D"
## [22] "#619A9C" "#659495" "#669294" "#639698" "#619A9C" "#5F9EA0" "#787778"
## [29] "#698D8F" "#6D888A"
```

Ces couleurs (en code héxadécimal) peuvent être utilisées dans un graphique, avec la possibilité de changer les couleurs comme on veut :

```
par(mfrow=c(1,2))
plot(x = altitude, y = recouvrement, pch=16, cex=2, col=coul_taille)
plot(x = altitude, y = recouvrement, pch=16, cex=2,
     col=Couleur_continue(taille,COLORS=c("Orange","Green")))
```

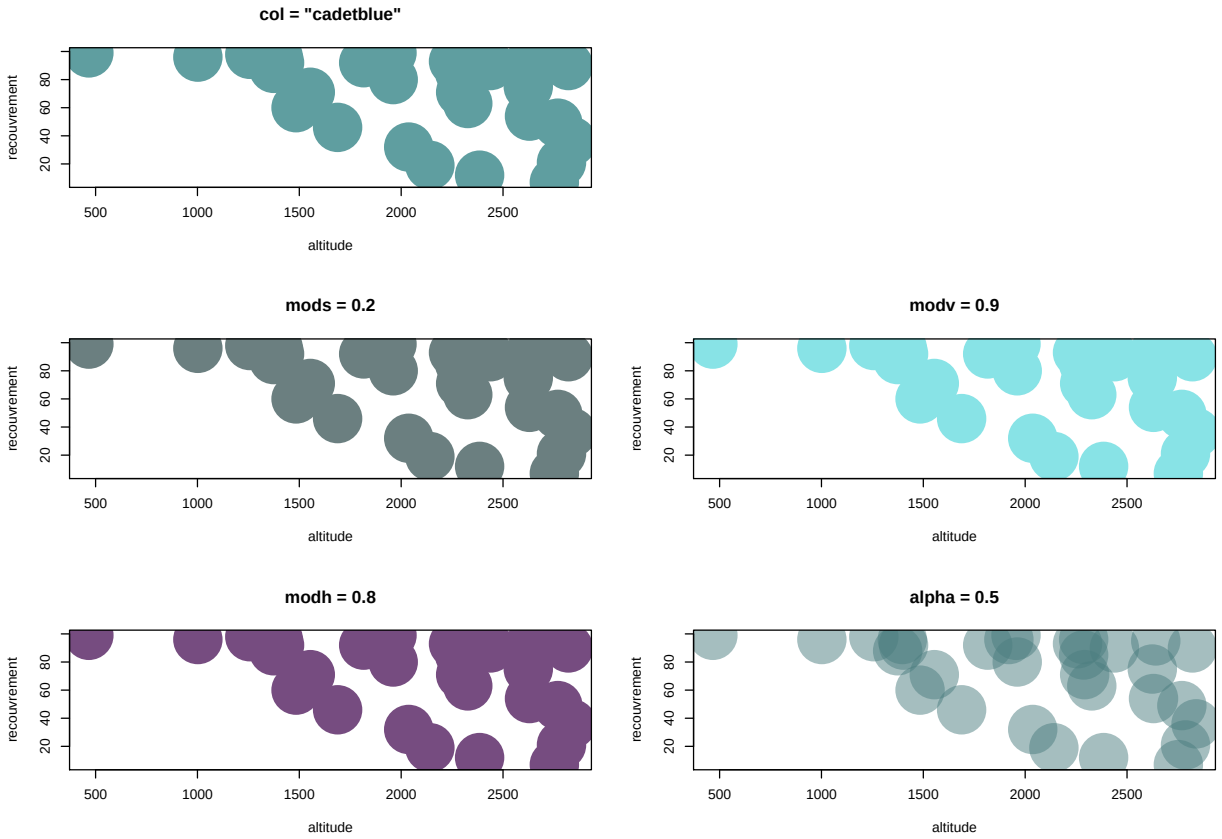


La fonction `modif_coul` permet de modifier une couleur existante :

Par exemple, en partant de la couleur `cadetblue` (qui est un bleu pastel), on peut :

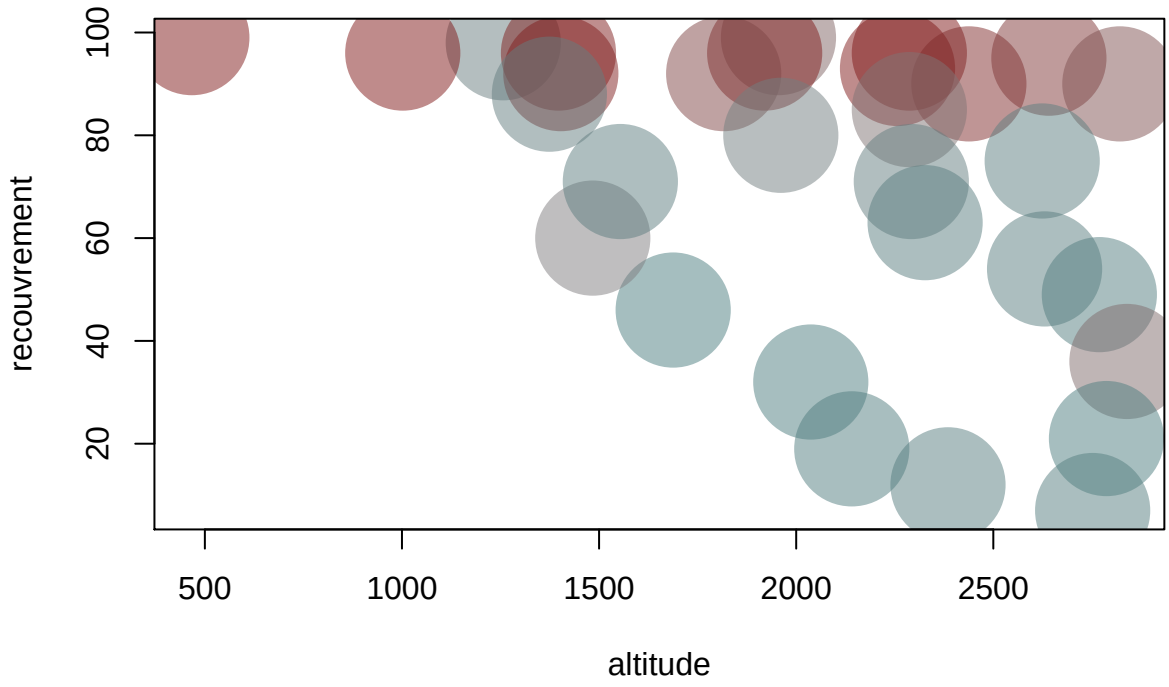
- modifier la saturation, sous 0.5 ça désature, au dessus, ça sature
- modifier la brillance, sous 0.5 ça fonce, au dessus, ça éclaircit
- modifier la teinte, ça tourne en rond..
- modifier la transparence sous 1, ça devient transparent

```
par(mfrow=c(3,2))
plot(x = altitude, y = recouvrement, pch = 16, cex = 8, col = "cadetblue",
     main = 'col = "cadetblue"')
plot(1,type="n",axes = F,ann = F) #juste pour un graphique vide..
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",mods = 0.2), main = "mods = 0.2")
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",modv = 0.9), main = "modv = 0.9")
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",modh = 0.8), main = "modh = 0.8")
plot(x = altitude, y = recouvrement, pch = 16, cex = 8,
     col = modif_coul(COULEUR = "cadetblue",alpha = 0.5), main = "alpha = 0.5")
```



On peut aussi combiner les deux fonctions :

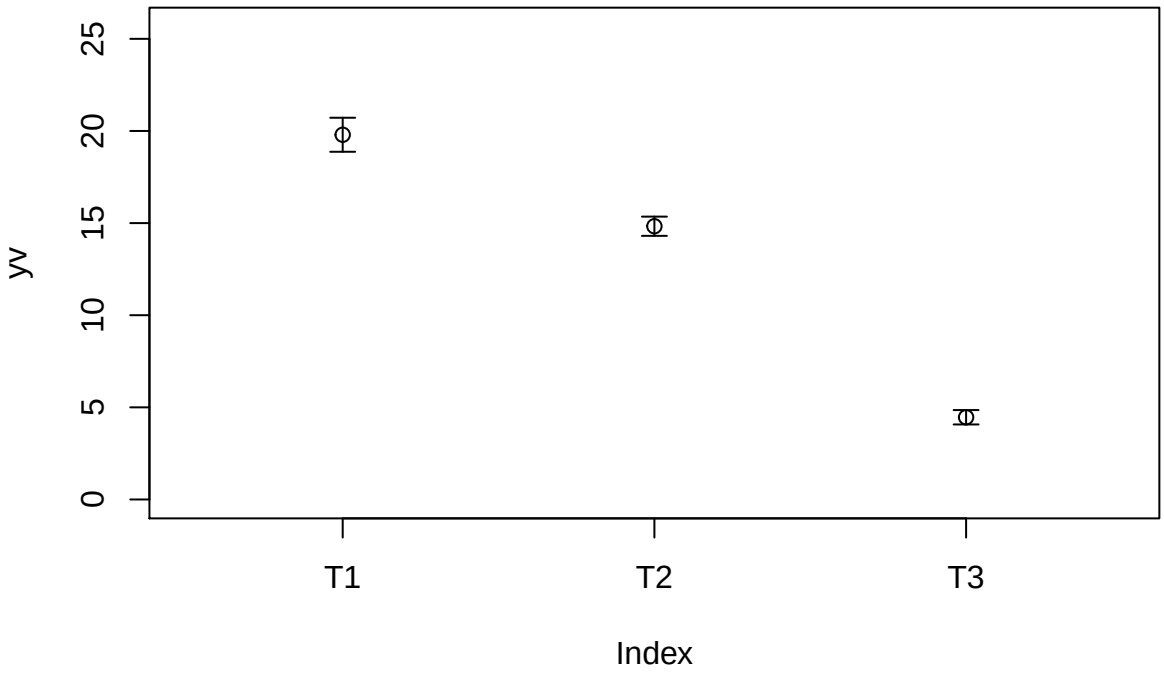
```
coul_taille_transp <- sapply(coul_taille, function(x) modif_coul(x, alpha = 0.5))
plot(x = altitude, y = recouvrement, pch=16, cex=8, col=coul_taille_transp)
```



2.7 Des diagrammes en barres mais avec des points avec `point.plot`

Quasi identique à l'utilisation de `barres.plot`.. Le graphique de base :

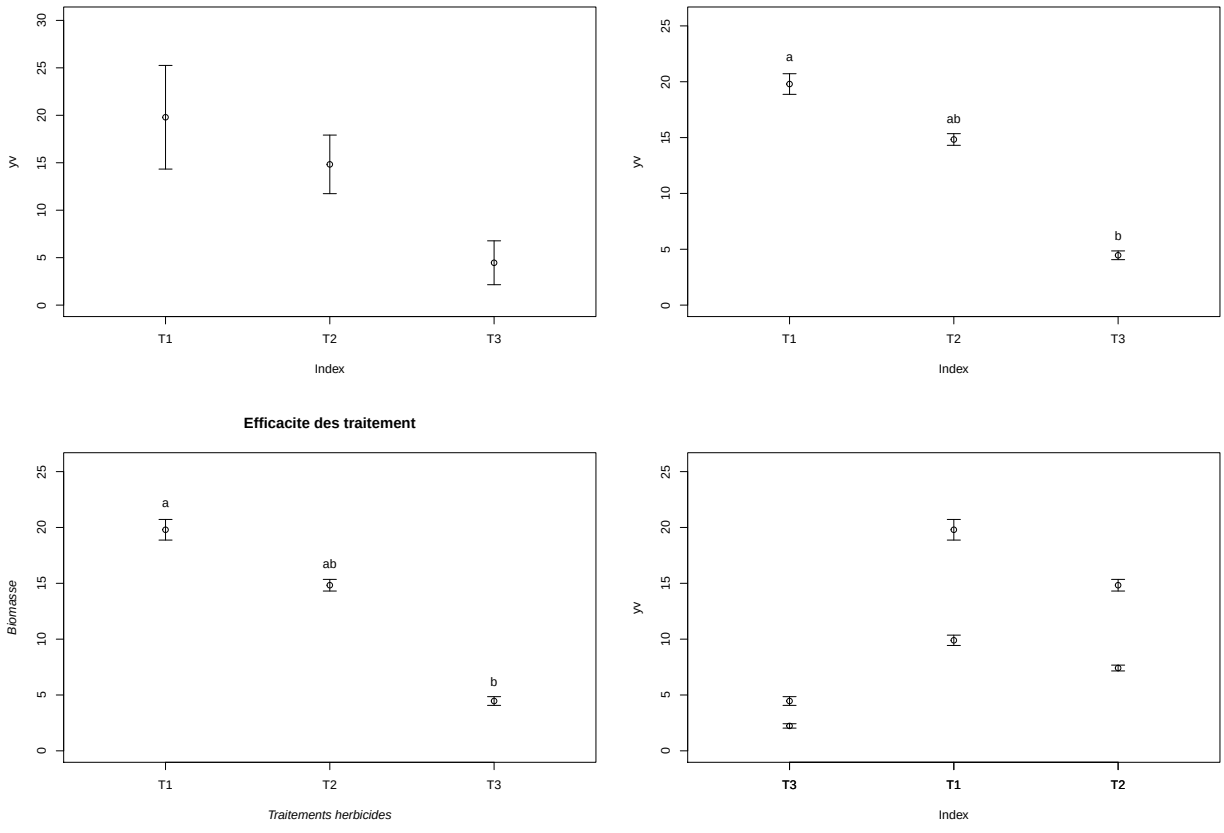
```
point.plot(biomasse,traitement)
```



Il y a plusieurs choses que l'on peut modifier pour personaliser son graphique :

- changer le type d'erreur représentée
- ajouter des lettres d'un test post-hoc
- changer des paramètres classiques de la fonction `barplot`
- changer l'ordre des modalités (ça ça n'est pas spécifique à la fonction `point.plot`) et surajouter sur un graphique

```
par(mfrow=c(2,2)) #pour avoir 4 graphiques sur une même fenêtre
point.plot(biomasse,traitement,ecart=sd)
#erreur standard par défaut, ici écart-type
point.plot(biomasse,traitement,lettres=c("a","ab","b"))
point.plot(biomasse,traitement,lettres=c("a","ab","b"),
xlab="Traitements herbicides",ylab="Biomasse",font.lab=3,
main="Efficacite des traitement")
traitement2<-factor(traitement,levels=c("T3","T1","T2"))
point.plot(biomasse,traitement2)
point.plot(biomasse/2,traitement2, add = TRUE)
```



3 Fonctions graphiques pour les analyses multivarées

4 Pour de la manipulation de données

4.1 La fonction `BBtransf`

4.2 La fonction `Classes_def`

4.3 La fonction `combin.tab`

4.4 La fonction `combin.tabV0`

5 Pour des tâches très spécifiques :

5.1 Autour de l'indice de l'integrité de la structure des communautés

5.2 La fonction `ComStructIndices`

5.3 La fonction `structure.plot`

5.4 La fonction `structure.plotV2`

5.5 La fonction “

“

“

“

“

“

“