TP02 - Découverte de R - Exercices

Justine Guégan - ICONICS

1 juin 2017

Ce polycopié rassemble différents exercices vous permettant d'apprendre à utiliser le langage de programmation R, en particulier, la lecture/écriture de fichiers, les graphiques, l'interprétation d'un code R existant. De manière générale, pensez à lire toutes les sous-questions car elles permettent de vous guider.

La notation entre <> n'est pas une notation R, c'est juste pour vous indiquer de personnaliser ce qu'il y a entre <>. Par exmeple, `brewer.pal(4, "<NomPaletteDeVotreChoix>"")` pourrait correspondre à `brewer.pal(4, "Greys")`.

Exercice 1 : KEGG pathways

Cet exercice a pour but de vous faire:

- **installer un package Bioconductor**
- **réutiliser un code R créé par une personne tierce**

1. Installer le package Pathview de Bioconductor (https://bioconductor.org/packages/release/bioc/html/pathview.html)

```
source('http://www.bioconductor.org/biocLite.R')
biocLite("pathview")
```

NB : Ne mettez pas à jour les packages si on vous le propose car cela peut être long
Redémarrez RStudio si nécessaire

2. Exécuter le code R `pathview.R` ligne à ligne et regarder le résultat.
3. Adapter ce code pour qu'il fonctionne sur le fichier d'entrée `myGenes.csv`, sur le pathway `hsa04012`. Pensez à :
   - Lire **correctement** le fichier `myGenes.csv`. (quel est le séparateur de colonne ?)
   - Imprimer le tableau lu, vérifier sa classe, ses dimensions ...

```
library(pathview)
genesTable = read.table("exo1/myGenes.csv", sep=";", header=TRUE)
print(genesTable)
class(genesTable)
dim(genesTable)
genes = as.vector(genesTable[,2])
pathwayID = "hsa04012"
geneIDTYPE = "SYMBOL"
pathview(genes, pathway.id = pathwayID, gene.idtype = geneIDTYPE,plot.col.key= FALSE)
```

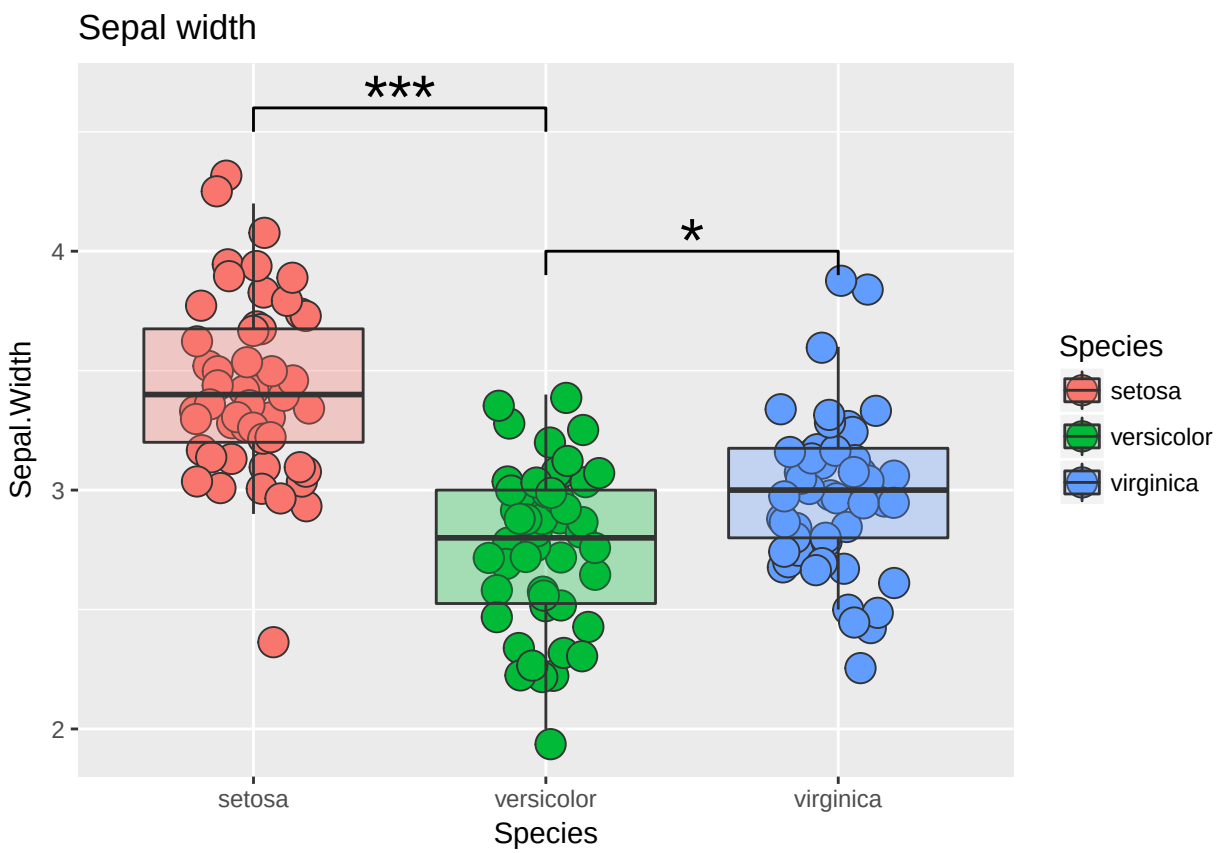
Exercice 2 : From iris data to expression data

Cet exercice a pour but de vous faire:

- **"décrypter"" du code R**
- **installer des packages**
- **apréhender le package graphique `ggplot2`**
- **apréhender un test statistique simple**

Vous avez trouvé sur internet une figure que vous trouvez très intéressante et que vous aimeriez reproduire pour vos propres données.

```
data(iris)
library(ggplot2)
p2 = ggplot(iris, aes(x=Species, y=Sepal.Width)) +
  geom_point(aes(fill=Species), size=5, shape=21, colour="grey20",
  position=position_jitter(width=0.2, height=0.1)) +
  geom_boxplot(outlier.colour=NA, aes(fill=Species), colour="grey20", alpha=0.3) +
  ggtitle("Sepal width")
df1 <- data.frame(a = c(1,1,2,2), b = c(4.5, 4.6,4.6,4.5))
df2 <- data.frame(a = c(2,2,3,3), b = c(3.9, 4,4,3.9))
p2 + geom_line(data = df1, aes(x = a, y = b)) + annotate("text", x = 1.5, y = 4.65, label = "***", size
  geom_line(data = df2, aes(x = a, y = b)) + annotate("text", x = 2.5, y = 4.05, label = "*", size = 8)
```



Coup de chance ! vous trouvez aussi le code R qui permet de générer cette figure ! `iris.R`

1. Essayez de comprendre ce que fait le code du script `iris.R`, juste en lisant les lignes de code, notamment les lignes suivantes (n'y passez pas trop de temps non plus, on voit tout en détail par la suite) :

- `library(ggplot2)`
- `res1 = t.test(iris$Sepal.Width[which(iris$Species == "setosa")], iris$Sepal.Width[which(iris$Species == "versicolor")])`
- `pdf("sepalWidth.pdf")`
  `print(p3)`
  `dev.off()`

2. Placez-vous à l'endroit où vous avez enregistré le script `iris.R` : Session –> Set working directory –> Choose directory.

Exécutez à présent la ligne de commande suivante :

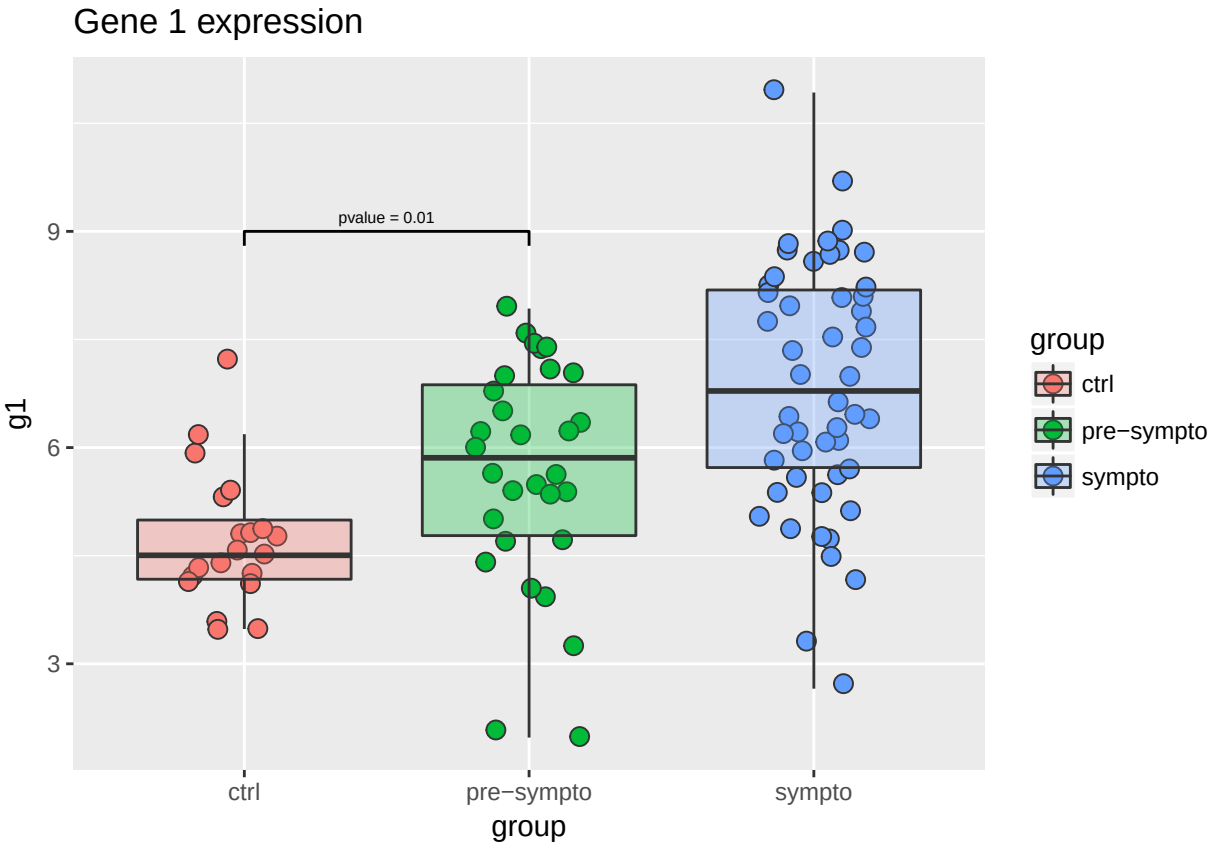
```
source("iris.R")
```

Que s'est-il passé ?

3. Vous trouvez la figure finale générée très intéressante et vous vous dites que vous pourriez vous en inspirer pour représenter vos données d'expression de gènes. Justement, vous avez screené votre géne fétiche appelé *g1* chez 100 individus classés en 3 groupes : contrôles, pré-symptomatiques et symptomatiques. De plus, le t-test réalisé pourrait apporter une valeur statistique à vote étude.

L'idée est donc de produire la figure suivante, en indiquant la pvalue issue du t-test entre les contrôles et les pré-symptomatiques.

```
library(ggplot2)
data = read.table("exo2/gene1.txt", sep="\t", header=TRUE)
p2 = ggplot(data, aes(x=group, y=g1)) +
geom_point(aes(fill=group), size=3, shape=21, colour="grey20",
position=position_jitter(width=0.2, height=0.1)) +
geom_boxplot(outlier.colour=NA, aes(fill=group), colour="grey20", alpha=0.3) +
ggtitle("Gene 1 expression")
res1 = t.test(data$g1[which(data$group == "ctrl")], data$g1[which(data$group == "pre-sympto")])
df1 <- data.frame(a = c(1,1,2,2), b = c(8.8, 9,9, 8.8))
p2 + geom_line(data = df1, aes(x = a, y = b)) +
annotate("text", x = 1.5, y = 9.2,
label = paste("pvalue = ", round(res1$p.value,2), sep=""), size = 2)
```



3.1 Lire le fichier `gene1.txt` (vous pouvez déjà le regarder sous excel ou notepad etc ... pour voir ce qu'il contient)

3.2 Comme pour le jeu de données `iris`, lancez quelques fonctions qui vous permettent d'avoir des infos sur le jeu de données (aperçu des données, stats de base, dimensions du tableau ...)

3.3 Adaptez les lignes de commandes de `iris.R` qui créent le premier graphique. Essayez de diminuer la taille des points, adaptez le titre ...

3.4 Lancez un t-test entre les contrôles et les pré-symptomatiques.

3.5 Adaptez les lignes de commandes de `iris.R` qui créent le graphique final.

Exercice 3 : Résumer et visualiser une liste de variants

Cet exercice a pour but de vous faire:

- **lire/écrire des fichiers**
- **filtrer des tableaux**
- **faire des représentations graphiques de base**

Le fichier `VariantsTable.xlsx` (ou `VariantsTable.csv`) est un tableau contenant une liste de variations génomiques (snps/indels) issues d'une capture de gènes faite par séquençage à haut débit, et ce, pour 4 patients : Sample1, Sample2, Sample3 et Sample4. Cette liste de variants est annotée avec différentes base de données représentant les différentes colonnes du fichier. Regardez ce fichier sous Excel pour commencer.

1. Lire le fichier `VariantsTable.csv`

2. Comme d'habitude, regarder la taille du tableau, sa classe etc ...

3. Combien y-at-il de variants par échantillon ? Faites une représentation en bâtons (`barplot()`) du nombre de variants par échantillon. Précisez un titre et colorez les barres en rouge et leur trait en bleu.

```
data = read.table("exo3/VariantsTable.csv", sep="\t", header=TRUE)
dim(data)
```

```
## [1] 5369   40
```

```
head(data)
```

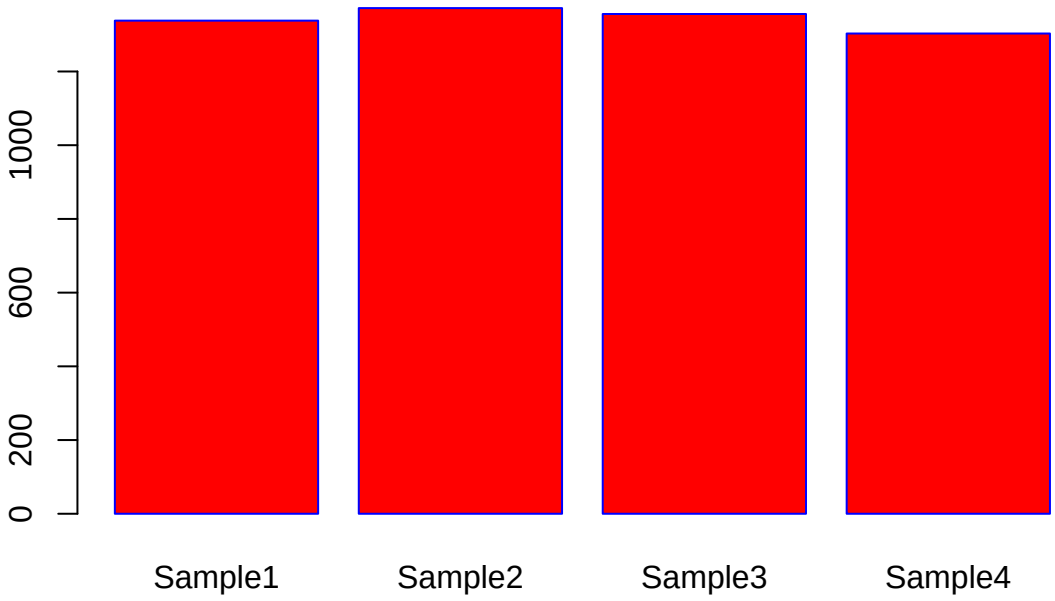
```
##   SampleID Chrom   Start     End Ref Alt Zygosity FILTER QualityScore   DP
## 1  Sample1  chr1  852875  852875   C   T      hom   PASS     55815.77 1791
## 2  Sample1  chr1  852964  852964   T   G      hom   PASS     42576.77 1359
## 3  Sample1  chr1 1602787 1602787   G   T      hom   PASS     46697.77 1390
## 4  Sample1  chr1 1886859 1886859   C   T      het   PASS     14953.77 1259
## 5  Sample1  chr1 1887019 1887019   A   G      hom   PASS     37481.77 1134
## 6  Sample1  chr1 2031342 2031342   T   C      hom   PASS     31792.77 1081
##   Depth.for.Ref Depth.for.Alt Frequency Func.refGene    Gene.refGene
## 1             0          1783       1.0 ncRNA_exonic    LOC100130417
## 2             5          1354       1.0 ncRNA_exonic    LOC100130417
## 3             0          1388       1.0     intronic CDK11B,SLC35E2B
## 4           640           613       0.5         UTR3          CFAP74
## 5             0          1129       1.0       exonic          CFAP74
## 6             1          1074       1.0     intronic           PRKCZ
##       GeneDetail.refGene ExonicFunc.refGene
## 1                   <NA>               <NA>
## 2                   <NA>               <NA>
## 3                   <NA>               <NA>
## 4 NM_001080484:c.*158G>A               <NA>
## 5                   <NA>           stoploss
## 6                   <NA>               <NA>
##                              AAChange.refGene     snp138 PopFreqMax
## 1                                        <NA> rs13303369       0.60
## 2                                        <NA>  rs4970461       0.88
## 3                                        <NA>  rs4639697       1.00
## 4                                        <NA>  rs3795290       0.40
## 5 CFAP74:NM_001080484:exon18:c.T2287C:p.X763Q rs28548017       0.86
```

```
## 6                                       <NA>  rs2376802       0.94
##   ExAC_ALL ESP6500siv2_ALL ESP6500siv2_AA ESP6500siv2_EA CG46 SIFT_score
## 1        .               .              .              . 0.46       <NA>
## 2        .               .              .              . 0.69       <NA>
## 3        .               .              .              .   1.       <NA>
## 4     0.33               .              .              . 0.15       <NA>
## 5     0.82            0.81           0.78           0.82 0.79          .
## 6        .               .              .              . 0.64       <NA>
##   SIFT_pred Polyphen2_HVAR_score Polyphen2_HVAR_pred LRT_score LRT_pred
## 1      <NA>                 <NA>                <NA>      <NA>     <NA>
## 2      <NA>                 <NA>                <NA>      <NA>     <NA>
## 3      <NA>                 <NA>                <NA>      <NA>     <NA>
## 4      <NA>                 <NA>                <NA>      <NA>     <NA>
## 5         .                    .                   .         .        .
## 6      <NA>                 <NA>                <NA>      <NA>     <NA>
##   MutationTaster_score MutationTaster_pred VEST3_score CADD_raw CADD_phred
## 1                   NA                <NA>        <NA>       NA         NA
## 2                   NA                <NA>        <NA>       NA         NA
## 3                   NA                <NA>        <NA>       NA         NA
## 4                   NA                <NA>        <NA>       NA         NA
## 5                    1                   P           .   -1.192      0.076
## 6                   NA                <NA>        <NA>       NA         NA
##   GERP..._RS phyloP46way_placental phyloP100way_vertebrate
## 1         NA                    NA                     NA
## 2         NA                    NA                     NA
## 3         NA                    NA                     NA
## 4         NA                    NA                     NA
## 5     -0.588                -0.644                 -0.638
## 6         NA                    NA                     NA
##   SiPhy_29way_logOdds
## 1                  NA
## 2                  NA
## 3                  NA
## 4                  NA
## 5                5.09
## 6                  NA
```

```
table(data$SampleID)
```

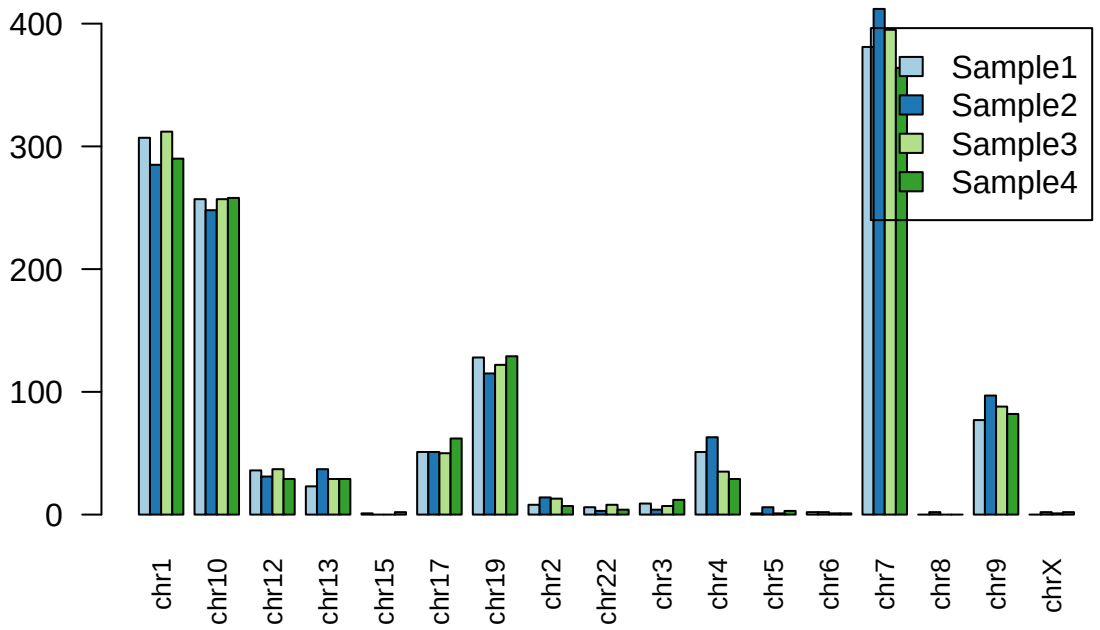
```
##
## Sample1 Sample2 Sample3 Sample4
##    1338    1372    1356    1303
```

```
barplot(table(data$SampleID), main="Nombre de variants par échantillon", col="red",border = "blue")
```


Nombre de variants par échantillon
0
200
600
1000
Sample1
Sample2
Sample3
Sample4

4. On souhaite à présent représenter en barplot le nombre de variants par chromosome, par échantillon, de telle sorte que le graphique généré ressemble à celui ci-dessous (les sous-questions vous guident pour générer la figure):

```
library(RColorBrewer)
data = read.table("exo3/VariantsTable.csv", sep="\t", header=TRUE)
tabChrSample = table(data$SampleID, data$Chrom)
barplot(tabChrSample, beside=TRUE, col=brewer.pal(4,"Paired"),legend.text = TRUE,cex.names = 0.8,las=2)
```



4.1 Pour cela, regardez l'aide de la fonction `table`, et inspirez-vous des lignes d'exemple. L'idée est d'obtenir un tableau de ce type :

```
print(tabChrSample[,1:6])
```

```
##
##           chr1 chr10 chr12 chr13 chr15 chr17
##   Sample1  307   257    36    23     1    51
##   Sample2  285   248    31    37     0    51
##   Sample3  312   257    37    29     0    50
##   Sample4  290   258    29    29     2    62
```

4.2 Puis créer le graphique toujours à l'aide de la fonction `barplot`. Regarder l'aide de cette fonction, notamment les arguments, afin :

- de mettre les barres les unes à côté des autres,
- d'ajouter une légende

4.3 Les noms de tous les chromosomes ne sont pas présents ? C'est normal car la police est trop grande : diminuer la taille de la police avec l'argument `cex.names` (par défait, `cex.names=1`). Si vous voulez tourner de 90° le noms des labels en x, utilisez l'argument `las=2`.

4.4 Vous voulez de la couleur plutôt que des nuances de gris ? Le package `RColorBrewer` est un package qui contient des palettes de couleurs. Pour l'utiliser, chargez le package, puis tapez `display.brewer.all()`.

Ainsi vous visualisez toutes les palettes de couleur du package. Choisissez le nom de l'une d'entre elles et, dans les arguments de `barplot`, ajoutez :
`col=brewer.pal(<nombreDeCouleursQueVousVoulez>,"<NomDeLaPalette>")`.

5. On veut à présent travailler sur l'impact fonctionnel des variants.

5.1 A partir de la colonne `ExonicFunc.refGene`, comptabilisez le nombre de variants par catégorie **pour l'échantillon Sample1**. *Indice : which() ; ==*

5.2 A partir du résultat de la question précédente, représentez cette fois ces données sous forme de camembert, encore appelé pie chart (`pie()`). Le rendu de base de cette fonction n'est pas trés joli ... Essayons de l'améliorer

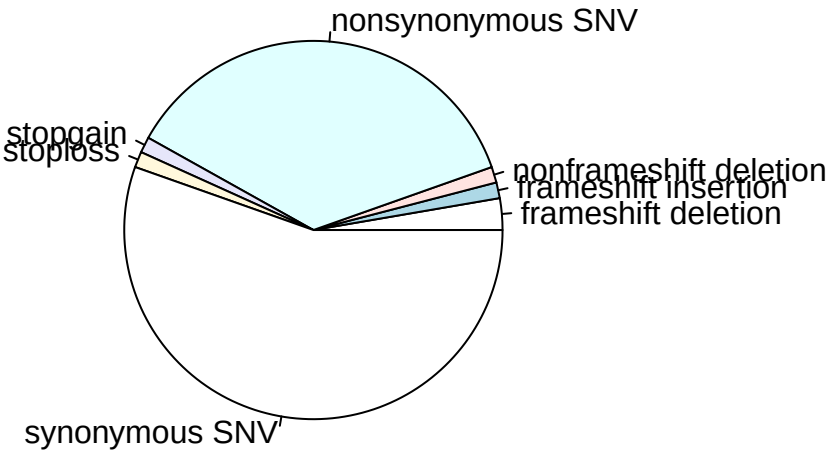
5.3

- triez le résultat de la question 5.1 (`sort()`)
- dans les arguments de `pie`, supprimez les bords du camembert, changez les couleurs, toujours à l'aide du package RColorBrewer,
- les tranches de certaines catégories sont si fines, que l'on a du mal à distinguer les noms. Il vaudrait peut-être mieux afficher une légende à côté de la figure. Pour cela, supprimer les labels dans la fonction `pie`, puis ajoutez une légende avec la commande suivante :

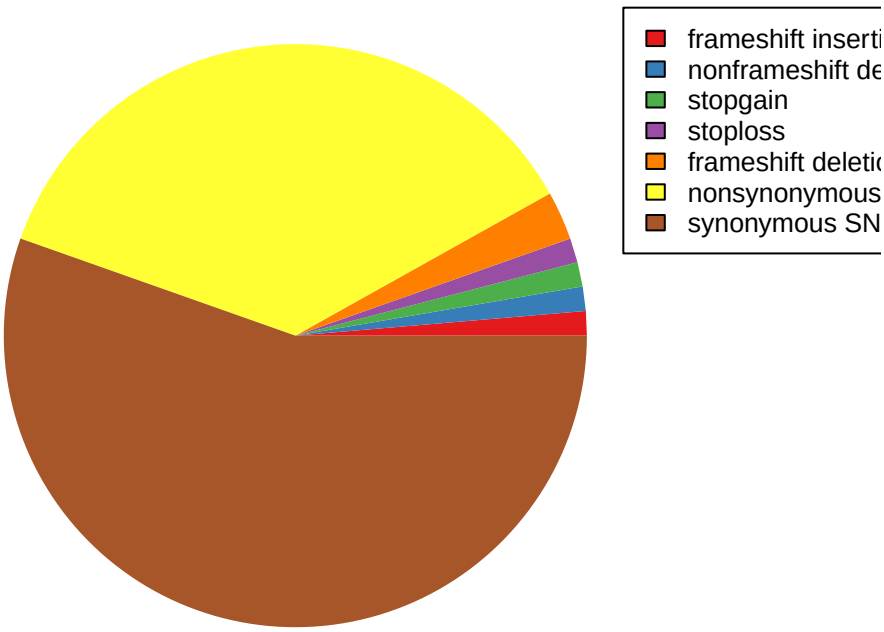
```
legend(0.9,0.9, legend=names(<NomObjet>), fill =brewer.pal(7, "<VotreCouleur>"), cex=0.8)
```

- le camembert parait trop petit comparé à la légende ? Nous pouvons diminuer les marges de la figure avec la commande suivante : `par(mar=c(1,1,1,1))`. Pour résumer, dans l'odre, diminuer les marges de la figure, plotter le pie chart, plotter la légende.

```
datared = data[which(data$SampleID == "Sample1"),]
exonic = table(datared$ExonicFunc.refGene)
pie(exonic)
```




```
exonicsort = sort(exonic)
par(mar=c(1,1,1,1))
pie(exonicsort, border=FALSE, col=brewer.pal(7, "Set1"), labels=NA)
legend(0.9,0.9, legend=names(exonicsort), fill =brewer.pal(7, "Set1"), cex=0.8)
```



6 On souhaite créer un tableau contenant uniquement des variants d'intérêt :

6.1 Créer un vecteur `vec1` contenant uniquement les indices des variants des catégories suivantes pour les 4 échantillons : nonsynonymous SNV, stoploss, stopgain. *Indice* ***`which()`***

6.2 Créer un vecteur `vec2` contenant uniquement les indices des variants dont le score CADD (`CADD_phred`) est supérieur à 12.37. *Indice* ***`which()`***

6.3 Créer un vecteur `vec3` contenant les indices communs entre la question 6.1 et 6.2 *Indice* ***`intersect()`***

6.4 Créer un nouveau tableau `tab` contenant uniquement les variants nonsynonymous SNV, stoploss, stopgain, dont le CADD_phred score est $>$ 12.37. Vous pouvez visualiser ce tableau dans RStudio avec `View(tab)`.

6.5 Ecrire ce nouveau tableau sur votre ordinateur grâce à la fonction `write.table`.

```
vec1 = which(data$ExonicFunc.refGene == "nonsynonymous SNV" | data$ExonicFunc.refGene == "stopgain" | da
vec2 = which(data$CADD_phred > 12.37)
vec3 = intersect(vec1, vec2)
tab = data[vec3,]
write.table(tab, file="exo3/outputTable.txt", sep="t", quote=FALSE, row.names=FALSE)
```

Exercice 4 : Intersecter des intervalles génomiques

Cet exercice a pour but de vous faire:

- **"décrypter"" du code R**
- **installer des packages**
- **découvrir le package GenomicRanges**
- **filtrer des matrices**
- **créer des graphiques de base**

Le package GenomicRanges permet de représenter et de manipuler des intervalles génomiques et des varaibles liées au génome. C'est un package Bioconductor, donc vous devez à présent savoir comment l'installer !

Dans cet exercice, on cherche à intersecter 2 fichiers contenant des positions génomiques : 1 fichier contenant une liste de transcrits, et 1 contenant une liste de snps. Inspirez-vous du script `findOverlaps.R` pour cet exercice.

1. Sourcez le script `findOverlaps.R`. Si des packages manquent, installez-les, soit via RStudio, soit via Bioconductor.
2. Essayer de comprendre les grandes lignes du script.
3. Créer un nouveau script pour répondre à cet exercice :

   3.1 Lire les fichiers `transcriptsTable.txt` et `snpsTable.txt`. Comme d'habitude, vérifier les dimensions, visualiser les données, les noms de colonnes etc ...

   3.2 En vous aidant du script `findOverlaps.R` et de l'aide du package GenomicRanges, créer 2 tableaux tab1 et tab2 contenant uniquement les snp dont la pvalue (*ie.* score) est inférieure à 0.01 et 0.005 respectivement ET qui overlappent un transcrit.

```
library("GenomicRanges")
library(plyr)
data = read.table("exo4/transcriptsTable.txt", sep="\t", header=FALSE)
snp = read.table("exo4/snpsTable.txt", sep="\t", header=TRUE)

snp1 = snp[which(snp$score < 0.01),]
snp2 = snp[which(snp$score < 0.005),]

data_grange <- with(data,
                    GRanges(V2, IRanges(V4,V5,names=V1),
                    V3
                    ))

snp1_grange <- with(snp1,
                   GRanges(chr, IRanges(start,end,names=id),
                   strand
                   ))
snp2_grange <- with(snp2,
                   GRanges(chr, IRanges(start,end,names=id),
                   strand
                   ))

counts1 = countOverlaps(data_grange, snp1_grange,ignore.strand=FALSE)
counts1 = data.frame(query=names(counts1), number=counts1)
```

```
overlaps1 <- findOverlaps(data_grange, snp1_grange,ignore.strand=FALSE)
match_hit1 <- data.frame(names(data_grange)[queryHits(overlaps1)],
                         names(snp1_grange)[subjectHits(overlaps1)],
                         stringsAsFactors=F)
names(match_hit1) <- c('query','subject')
match_hit1_uniq = ddply(match_hit1, .(query), summarize, subject = paste(subject, collapse = ';'))
tab1 = merge(match_hit1_uniq, counts1, by="query",all.y=TRUE)

counts2 = countOverlaps(data_grange, snp2_grange,ignore.strand=FALSE)
counts2 = data.frame(query=names(counts2), number=counts2)
overlaps2 <- findOverlaps(data_grange, snp2_grange,ignore.strand=FALSE)
match_hit2 <- data.frame(names(data_grange)[queryHits(overlaps2)],
                         names(snp2_grange)[subjectHits(overlaps2)],
                         stringsAsFactors=F)
names(match_hit2) <- c('query','subject')
match_hit2_uniq = ddply(match_hit2, .(query), summarize, subject = paste(subject, collapse = ';'))
tab2 = merge(match_hit2_uniq, counts2, by="query",all.y=TRUE)

TAB = merge(tab1, tab2, by="query")
TAB2 = TAB[which(TAB$number.x != 0 | TAB$number.y != 0),]
TAB2sort = TAB2[order(TAB2$number.x),]
par(mar=c(8,4,4,2))
plot(TAB2sort$number.x, ylim=c(0,3), pch=16, type="b", xaxt="n", xlab="", ylab="#", lab=c(1,4,4))
points(TAB2sort$number.y, col="red", pch=16, type="b")
axis(side=1, labels=TAB2sort$query, at=1:nrow(TAB2sort), las=2, cex=0.5)
```