```
# File: AllBoxes
# This document has the content of all boxes from "WorkshopCropDiversity.pdf"
# the idea is that you can "copy" the content, "paste" in your R command
# window to follow the calculations.

# Note: Here the R prompt ">" as well as the R output are NOT presented!

# This is an example of a box for R input/output
seq(from=0, to=1, by=0.2)

##### Box 1
dir()
demo() # Which "demos" are there?
2^5; log2(64); sin(0); pi # Note ";" can be used as carriage return
x # Is there an object call "x" in the "environment"?
x <- 55 # Note the symbols "<-" are the assignment operator
x # Now we have the value of 5 "stored" in an object called "x"
X<-20
x == X # R knows about logical comparisons
x<=X #Is x less or equal than X
x>X #Is x larger than X?
x <- c(1,2,5,7,15,20) # A vector with 6 numbers
y
y^x # Will give some large numbers

## Assume that you want to make a Tukey test...
Tukey # Just try your look...
?? Tukey # Asking to see if the word "Tukey" appears somewhere

# R knows also about matrices...
? matrix
z <- matrix(c(1:4), nrow=2, ncol=2) # A 2 x 2 matrix...

# What the heck means "c(1:4)"?
? c
c(1:4)
c(1.1:2.2)
c(1.1:5.2)

# A more powerful way to create numeric sequences
? seq
seq(from=0, to=10, by=2)
# You do not need to write the parameters names (respect the order)
seq(0,10,2)

# There is also a function to "repeat" (replicate) elements
? rep
rep(0, 5)
rep(c(1:2), each=3)

# Characters can also been used
rep(c("a", "b", "c"), each=2)
rep(c("a", "b", "c"), 2)
letters # A predefined
LETTERS # Another

# There are ways to access the elements within a structure; the "[ ]" operator
temp <- seq(0, 10, 2) # Or seq(from=0, to=10, by=2)
temp

length(temp) # The length (number of elements) of temp
temp[3] # The third element
temp[3:5] # Elements 3 to 5
temp[7] # An element that does NOT exist

# The special value "NA" means "Not Available" or "missing" (important)
```

```
# Let’s see the operator "[ ]" for matrices
temp2 <- matrix(c(1:9), nrow=3, ncol=3, byrow=TRUE)
temp2
temp2[2,3]
temp2[,3] # Values in the third column
temp2[1,] # Values in the first row

# R has many pre-loaded data sets; let’s see
? data
data() # List the ones available
data(CO2) # Loads the "CO2" dataset
head(CO2, n=5) # Shows the first 5 rows
class(CO2) # Which class(es)?
nrow(CO2) # Number of rows
summary(CO2) # Let's see
CO2$Plant # All values of column Plant
CO2$Plant[1:5] # Values 1 to 5 of column Plant
class(CO2$Plant) # What kind of variable
class(CO2$Type) # What kind of variable
CO2$Type[1:5]
CO2[2:4,3:5] # Elements in rows 2 to 4 and columns 3 to 5

# Data Frames are a class of objects that can have columns with numbers or factors.
# Let's create a data.frame
? data.frame
temp.df <- data.frame(Treatment=rep(c("A", "B"), each=10),
   Result=c(rnorm(10, 10), rnorm(10, 11)))
summary(temp.df) # Note that your results for "Result" will be different!
tapply(temp.df$Result, temp.df$Treatment, summary)
summary(aov(Result ~ Treatment, data=temp.df)) # Summary of ANOVA
boxplot(Result ~ Treatment, data=temp.df) # Result not shown

# A powerful kind of structure within R are lists; let's see:
temp.l <- list("This list contains my temp objects", temp, temp2,
   temp.df, summary(aov(Result ~ Treatment, data=temp.df)))

temp.l # Show all elements of the list (not shown here)
temp.l[[1]] # The first element of the list
class(temp.l[[1]])
temp.l[[3]] # The third element of the list
temp.l[[3]][,2] # The second column of the third element of the list

# in R you can "program"; i.e., create functions
temp.f <- function(x=5){paste("my input was =", x)} # A silly function
class(temp.f)
temp.f # Shows the function
temp.f() # Execute the function with the default
temp.f(3.141592654) # Execute the function with other value
temp.f("Hello World!") # Execute the function with other value

# LOOPS
# As any programing language R can perform loops; see for example:
for(i in 1:5){print(i)}

for(i in 1:5){cat(i, "! = ", factorial(i), "\n", sep="")}
# Be creative and EXPLORE R possibilities!!!
q() # TO quit the R session; you can save (or not) these results.


##### Box 2
# Run R and go to your 'CabanaR' directory
# Note that IN YOUR computer the next line may point to a different place!
load("DummyExample.RData") # Load some stuff

# You can list the objects currently in your environment with
```

```
ls()

# The object that we will be abalysing:
my.dummy
class(my.dummy)
length(my.dummy)
my.dummy[1]
my.dummy[1] == my.dummy[2] # Are those IDENTICAL?

##### Box 3
# Let's split each one of the 4 sequences in my.dummy into characters:
temp <- strsplit(x=my.dummy, split="") # Let's try
class(temp) # Which class of object is "temp"
temp # Let's see it in full [ OUTPUT not shown here! ]
length(temp) # Number of sequences in the list
length(temp[[1]]) # Number of "bases" in the first sequence

# Now, having each one of the sequences as vectors in temp[[1]], temp[[2]], ...,
temp[[4]]
# we could compare them "base to base", for example:
temp[[1]] == temp[[2]] # Compare sequences 1 and 2 "base to base"

# Note how R convert logical values TRUE to 1 and FALSE to 0 when needed:
1 == 1 # That is a question
1*(1 == 1) # That is a numerical value
"A" == "A" # That is a question
1*("A" == "A") # That is a numerical value
"A" == "T" # That is a question
1*("A" == "T") # That is a numerical value

# We can use that fact to detect how many bases are equal
# (or different) between two sequences
sum(temp[[1]] == temp[[2]])
# Because we "sum" logical are converted to numbers
sum(temp[[1]] != temp[[2]]) # "!=" is short for different
46+4 # The full length of the sequences

##### Box 4
info.sites # My function to isolate informative sites:
info.sites(x=my.dummy) # Run with our 4 sequences
my.dumy.info <- info.sites(x=my.dummy) # Keep the result in an object
class(my.dumy.info)
ncol(my.dumy.info) # Number of informative bases
nrow(my.dumy.info) # Number of sequences
info.sites(x=my.dummy, out.as.matrix=FALSE) # Alternative output

##### Box 5
# The function that I programmed
SNP.dist
## Look what the function "as.dist" does
? as.dist
# Run the function...
SNP.dist(my.dumy.info) # Running with our matrix of informative sites

##### Box 6
my.dummy.dis # To remember the content of that object.
? hclust # See the function to obtain the clustering from a distance matrix
my.dummy.clu.1 <- hclust(my.dummy.dis, method = "complete")
plot(my.dummy.clu.1) # Plot shown as Figure 1.

##### Box 7
# List the functions that you have in your current environment:
lsf.str() # See the help for that function!
# You also have a vector with the names of objects important for the example:
dummy.stuff
dummy # What do we have here?
```

```
# Which were the sequences that we previously analyzed? (in my.dummy; 4 sequences)
my.dummy[1]
my.dummy[1]==dummy$Genome[4] # This is lin1.1
my.dummy[2]==dummy$Genome[5] # This is lin1.2
my.dummy[3]==dummy$Genome[7] # This is lin2.1
my.dummy[4]==dummy$Genome[8] # This is lin2.2

# Let's see the functions "randomDNA" and "mutateDNA"
randomDNA
mutateDNA

# Let's see two examples (NOTE your output will be different from the one shown, why?)
temp <- randomDNA(15)
temp
temp2 <- mutateDNA(temp, how.many=5)
temp2
temp == temp2
sum(temp == temp2)

# Here is the history to create the dataset 'dummy' (output not shown)

##### Box 8
# First, isolate the set of informative sites in the 8 sequences
dummy.info <- info.sites(dummy$Genome) # There are 8 sequences there
dummy.info
attributes(dummy)$row.names # The names of the sequences
attributes(dummy.info)$row.names <- attributes(dummy)$row.names
dummy.info
dummy.dis <- SNP.dist(dummy.info)
dummy.dis
attributes(dummy.dis) # See the attributes of that object
# ... Other attributes
attributes(dummy.dis)$Labels <- attributes(dummy.info)$row.names
dummy.dis
ncol(dummy.info) # Number of informative sites
17*dummy.dis # Row number of SNPs (mutations between ndividuals)
? hclust # To remember the methods:
# "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (=
WPGMA),
#  "median" (= WPGMC) or "centroid"

# I will multiply by 17 the distance to measure height directly in "SNPs".
plot(hclust(17*dummy.dis, "ward.D")) # Figure 3
plot(hclust(17*dummy.dis, "ward.D2")) # Not shown
plot(hclust(17*dummy.dis, "single")) # Figure 2
plot(hclust(17*dummy.dis, "complete")) # Not shown
plot(hclust(17*dummy.dis, "average")) # Figure 4
plot(hclust(17*dummy.dis, "mcquitty")) # Not shown
plot(hclust(17*dummy.dis, "median")) # Figure 5
plot(hclust(17*dummy.dis, "centroid")) # Not shown

### NOTE: Last part of this box are Notes, thus, not shown!!!

##### Box 9
# Structure of our distance matrix
17*dummy.dis
dummy.clu.med <- hclust(17*dummy.dis, method="median") # A cluster object
dummy.clu.med # If we ask to see (print) the object...
class(dummy.clu.med) # Which kind of class is it?
# However, the "primitive" form of this object is a "list", see
length(dummy.clu.med) # How many components?
names(dummy.clu.med) # Which names?

## See the description of the output with
? hclust # See section "Value"
# and analyze each component
```

```
dummy.clu.med$merge # Or also dummy.clu.med[[1]]

dummy.clu.med$height # The second component, dummy.clu.med[[2]]

dummy.clu.med$order # Also dummy.clu.med[[3]]

dummy.clu.med$labels # Also dummy.clu.med[[4]]

dummy.clu.med$call # Also dummy.clu.med[[5]]

##### Box 10
# Let's play with some of the parameters:
plot(dummy.clu.med, hang=0) # Result not shown as figure
plot(dummy.clu.med, hang=0.1) # Result not shown as figure

# Let's see the methods that exist for function "plot"
methods(plot)
? methods

# Ask help SPECIFICALLY for the plot.dendrogram method:
? plot.dendrogram
# Let's convert our object "dummy.clu.med" to a dendrogram:
dummy.med.den <- as.dendrogram(dummy.clu.med)

# Now in "plot(dummy.med.den)" we can use
# different options present in "? plot.dendrogram".
# Different options:
plot(dummy.med.den, type="triangle")
plot(dummy.med.den, center=T)
plot(dummy.med.den, edge.root=T)
plot(dummy.med.den, horiz=T)

# Modifying the edges (lines) of our dendrogram
my.edgePar <- vector("list", 3) # Define a list with three components
names(my.edgePar) <- c("col", "lty", "lwd") # Color, line type and line width
my.edgePar$col <- "red"
my.edgePar$lty <- 2
my.edgePar$lwd <- 2
plot(dummy.med.den, edgePar=my.edgePar)

my.edgePar$col <- c("red", "brown")
my.edgePar$lty <- 1
my.edgePar$lwd <- 2
plot(dummy.med.den, edgePar=my.edgePar)
plot(dummy.med.den, edgePar=my.edgePar, type="triangle")

# Modifying the nodes
my.nodePar <- vector("list", 3)
names(my.nodePar) <- c("pch", "cex", "col")
my.nodePar$pch <- c(1,2)
my.nodePar$cex <- c(1,2)
my.nodePar$col <- c("red", "brown")
plot(dummy.med.den, nodePar=my.nodePar)
plot(dummy.med.den, edgePar=my.edgePar, nodePar=my.nodePar)

##### Box 11
# Using dummy.clu.med
plot(dummy.med.den, edgePar=my.edgePar, nodePar=my.nodePar)
rect.hclust(dummy.clu.med, h=3, which=c(2,4))
rect.hclust(dummy.clu.med, h=3, which=c(2,4), border="darkviolet")
rect.hclust(dummy.clu.med, h=3, which=c(2,4), border=c("red","blue"))

cutree(dummy.clu.med, k=2)

##### Box 12
```

```
# Function to convert a matrix of informative sites into a matrix
# that could be used by the "dist" function.
# You can input the function using
source("info.sites2num.txt") # That file exist in your directory
info.sites2num # See the function
dummy.dat <- info.sites2num(x = dummy.info) # Obtains the new form of the data
dummy.dat[, 1:4] # All rows and the first 4 columns of the new matrix
dummy.info[,1] # This values were converted into the data above.
# Also for example:
dummy.info[,4:5]
dummy.dat[,13:20]
# Etc.

? dist # To see the options of the "dist" function.
# method can be "euclidean", "maximum", "manhattan", "canberra", "binary" or
"minkowski".

## Now, see that:
# Our distance given in mutations
# (obtained with my.dummy.dis <- SNP.dist(my.dumy.info))
17*dummy.dis

# Is equal than
(dist(dummy.dat, method="euclidean")^2) / 2

# Now you can try some alternative distances and clustering methods!
# Compare this with Figure 2
plot(hclust(dist(dummy.dat, method="euclidean"), "single"))
# Other possibilities...
plot(hclust(dist(dummy.dat, method="euclidean")^2, "median"))
plot(hclust(dist(dummy.dat, method="euclidean")^2, "average"))
plot(hclust(dist(dummy.dat, method="euclidean"), "average"))
# Changing distances
plot(hclust(dist(dummy.dat, method="manhattan"), "average"))
plot(hclust(dist(dummy.dat, method="binary"), "average"))
plot(hclust(dist(dummy.dat, method="binary"), "complete"))
## You have 6 distances and 8 clustering methods: 48 possible combinations!

##### Box 13
# You need to have installed the package "pvclust" into your computer!
library(pvclust) # Loads the package for its use.
# See the help for the package
? pvclust
# Or even better, visit web site
# http://stat.sys.i.kyoto-u.ac.jp/prog/pvclust/
system.time(
dummy.pv.1 <- pvclust(data=t(dummy.dat), method.hclust="average",
                  method.dist="euclidean", nboot=1000, iseed=1959)
)

dummy.pv.1
plot(dummy.pv.1) # Result presented as Figure 6.

### HERE I save all stuff from Boxes 2 to 13
# You can also do that (if you want)
# save.image("Boxes2to13.RData")

##### Box 14
# NOTE: Here I begin with a "clean" (new) environment
# However you can continue within your preview environment.
# Remember, you MUST be in the working directory
# ending with "CabanaR" !
load("incidence.RData") # This has the "incidence" matrix
class(incidence)
nrow(incidence)
ncol(incidence)
```

```
incidence[1:5, 1:5] # See some rows and columns of the matrix

# We have a matrix with results for 1338 accessions (rows) and
# 333 combinations Marker / Allele (columns).
# Let's keep the names of rows and columns in two vectors
acc.names <- attributes(incidence)$dimnames[[1]] # Names of accessions (columns)
mar.ale.names <- attributes(incidence)$dimnames[[2]] # Names of marker/alleles (rows)
head(acc.names)
head(mar.ale.names)

# We have 1338*333 = 445,554 data points in this large matrix
# A first summary can be given by tabulating the data
inc.table <- table(as.vector(incidence))
inc.table
sum(inc.table)
round(100*inc.table/sum(inc.table), 2) # In percentages

##### Box 15
# The information that we need is already in "mar.ale.names".
# Example of segregation of the names using "strsplit" (only the first two components)
strsplit(head(mar.ale.names, 2), split=".", fixed=T)
temp <- strsplit(mar.ale.names, split=".", fixed=T)
length(temp) # Same than the number of columns of incidence
# Make a data.frame to order the data
mar.allele <- data.frame(marker=rep("", 333), allele=rep("", 333), stringsAsFactors=F)
# And fill that data.frame
for(i in 1:333){
mar.allele[i, ] <- c(temp[[i]][1], temp[[i]][2])
}

head(mar.allele)
tail(mar.allele)

# Now we can obtain a table with the information that we want
length(unique(mar.allele$marker)) # How many different markers?
temp <- unique(mar.allele$marker) # Put them in a vector
temp
aleles.per.marker <- data.frame(Marker=temp, N.alleles=rep(NA, 14),
stringsAsFactors=F)
for(i in 1:14){
aleles.per.marker$N.alleles[i] <- nrow(mar.allele[mar.allele$marker ==
     aleles.per.marker$Marker[i], ])
}

aleles.per.marker # See our table
aleles.per.marker[order(aleles.per.marker$N.alleles),] # Orderd by N.alleles
table(aleles.per.marker$N.alleles) # Number of cases
summary(aleles.per.marker$N.alleles)
sd(aleles.per.marker$N.alleles)
## As an EXERCISE try to reproduce Table 3 with your data.

##### Box 16
# Obtain a vector to distinguish the accession group, remember
head(acc.names)
# Using "substring" we can get the accession group (acc.group)
substring(head(acc.names), 1, 2)
acc.group <- substring(acc.names, 1, 2)
table(acc.group) # Compare with Table 3.
acc.group

# We are going to need some extra function which are in file "ClustStructFun.txt"
source("ClustStructFun.txt") # Load such functions
lsf.str() # List functions in our current environment:

# Obtain the Euclidean distance from the incidence matrix
incidence.dist <- dist(incidence, method="euclidean")
```

```
length(incidence.dist)
summary(incidence.dist)
incidence.dist.df <- dist2data(incidence.dist)
class(incidence.dist.df)
nrow(incidence.dist.df)
head(incidence.dist.df)
tail(incidence.dist.df)
# Now we can add variables for the state, simply by taking the first two
# characters of the names
incidence.dist.df <- data.frame(incidence.dist.df,
                  row.st=substring(incidence.dist.df$row,1,2),
                  col.st=substring(incidence.dist.df$col,1,2),
                  stringsAsFactors=F)
head(incidence.dist.df)

# Now is the distance "between" or "within" groups?
incidence.dist.df <- data.frame(incidence.dist.df,
                  within=incidence.dist.df$row.st == incidence.dist.df$col.st)
head(incidence.dist.df)
head(incidence.dist.df[!incidence.dist.df$within,])

# Now some statistics for the value by the "within" variable
tapply(incidence.dist.df$value, incidence.dist.df$within, length)
tapply(incidence.dist.df$value, incidence.dist.df$within, summary)
tapply(incidence.dist.df$value, incidence.dist.df$within, sd)
t.test(value ~ within, data=incidence.dist.df)

? wilcox.test # A non-parametric test
wilcox.test(value ~ within, data=incidence.dist.df)

# Boxplot (Figure 7)
boxplot(value ~ within, data=incidence.dist.df,
                  xlab="Is distance within groups?",
                  ylab="Euclidean distance",
                  main="Distances in maize groups.",
                  sub="(see Box 16).")

##### Box 17
incidence.seg # See the function (not help available)
# Obtain the segregation of the data (remember what we have in "acc.group")
incidence.seg <- segregate.dis(incidence, classes=acc.group)
class(incidence.seg)
names(incidence.seg)
length(incidence.seg$main)
class(incidence.seg$main)
head(names(incidence.seg$main))
class(incidence.seg$classes.table)
length(incidence.seg$classes.table)
incidence.seg$classes.table # We have 16 groups

# Thus, there will be 16*15/2 = 120 comparisons "between" groups
# plus 16 within groups; a total of 120+16=136 comparisons.
incidence.seg.sum <- segregate.summary(incidence.seg)
nrow(incidence.seg.sum)
head(incidence.seg.sum)
tail(incidence.seg.sum)

# Note: Last row gives a summary of ALL 894453 distances between accessions.

# Now we are going to use another function to convert "incidence.seg" into
# a matrix. See function "segregated2matrix" (no help available)

maiz.group.dis <- segregated2matrix(incidence.seg)
round(maiz.group.dis)

# Note: Elements in the main diagonal are distances within group
```

```
# while other elements are distances between groups
# (of course, this matrix is symmetric)

# Finally we can construct a dendrogram to show the relation between groups.
maiz.group.t.dis <- as.dist(maiz.group.dis)
hclu.maiz.group <- hclust(maiz.group.t.dis, method="average")
# The plot presented as  Figure 8
plot(hclu.maiz.group, main="Cluster of groups of maize accessions\n
                  (using mean distance between groups)")

##### Box 18
# Let's obtain a matrix with only the "DG" or "NL" accessions

# How to select which rows of incidence correspond to "DG" or "NL"?
# Example
head(substring(attributes(incidence)$dimnames[[1]],1,2))
head(incidence[substring(attributes(incidence)$dimnames[[1]],1,2)=="DG", 1:5])
inc.DG.NL <- incidence[(substring(attributes(incidence)$dimnames[[1]],1,2)=="DG")|
                  (substring(attributes(incidence)$dimnames[[1]],1,2)=="NL"),]
nrow(inc.DG.NL)
head(inc.DG.NL[,1:5])
tail(inc.DG.NL[,1:5])

# Let's obtain labels for each accession
groupDG.NL <- substring(attributes(inc.DG.NL)$dimnames[[1]],1,2)
DG.NL.dis <- dist(inc.DG.NL) # Euclidean distance (default)
col.DG.NL <- groupDG.NL
col.DG.NL[col.DG.NL=="DG"] <- "red" # Assign red to DG
col.DG.NL[col.DG.NL=="NL"] <- "blue" # Assign blue to NL
# Now use "myColorDendrogram" (not help available)
# Presented as Figure 9
myColorDendrogram(hclust(DG.NL.dis, "average"), y=col.DG.NL, branchlength=1.6)
title(main="Dendrogram of accessions from DG and NL")
legend("topright", bty="n", legend=c("DG", "NL"), lty=1, col=c("red", "blue"))

# Statistics for distances within and between DG and NL
round(incidence.seg.sum[attributes(incidence.seg.sum)$dimnames[[1]]=="DG.DG",])
round(incidence.seg.sum[attributes(incidence.seg.sum)$dimnames[[1]]=="NL.NL",])
round(incidence.seg.sum[attributes(incidence.seg.sum)$dimnames[[1]]=="NL.DG",])

##### Box 19
# TESTING DG vs NL
seg.DG.NL <- segregate.dis(inc.DG.NL, classes=groupDG.NL)
segregate.summary(seg.DG.NL)
segregate.plot(seg.DG.NL)

# Trying to apply "segregate.test" blindly...
segregate.test(seg.DG.NL)

# The contrast of interest:
segregate.test(seg.DG.NL, cont="NL.DG")

##### Box 20
# Analyzing TE (teosintes) versus PA (palomero)
inc.TE.PA <- incidence[(substring(attributes(incidence)$dimnames[[1]],1,2)=="TE")|
                  (substring(attributes(incidence)$dimnames[[1]],1,2)=="PA"),]
nrow(inc.TE.PA)
groupTE.PA <- substring(attributes(inc.TE.PA)$dimnames[[1]],1,2)
table(groupTE.PA)
TE.PA.dis <- dist(inc.TE.PA) # Euclidean distance (default)
col.TE.PA <- groupTE.PA
col.TE.PA[col.TE.PA=="TE"] <- "red" # Assign red to TE
col.TE.PA[col.TE.PA=="PA"] <- "blue" # Assign red to PA
myColorDendrogram(hclust(TE.PA.dis, "average"), y=col.TE.PA, branchlength=1.6)
title(main="Dendrogram of accessions from TE and PL")
legend("topright", bty="n", legend=c("TE", "PL"), lty=1, col=c("red", "blue"))
```

```
seg.TE.PA <- segregate.dis(inc.TE.PA, classes=groupTE.PA)
segregate.summary(seg.TE.PA)

segregate.test(seg.TE.PA, cont="TE.PA")

# For Figure 10
col.TE.PA <- groupTE.PA
col.TE.PA[col.TE.PA=="TE"] <- "red" # Assign red to TE
col.TE.PA[col.TE.PA=="PA"] <- "blue" # Assign blue to PA

myColorDendrogram(hclust(TE.PA.dis, "average"), y=col.TE.PA, branchlength=1.6)
title(main="Dendrogram of accessions from TE and PA")
legend("topright", bty="n", legend=c("TE", "PA"), lty=1, col=c("red", "blue"))

##### Box 21
# NOTE: You need to work to obtain the corresponding results!

##### Box 22
inc.AMA <- AMA(incidence)
class(inc.AMA)
names(inc.AMA)
inc.AMA$Set.names
inc.AMA$Richness
table(substring(inc.AMA$Set.names,1,2))

# If you want to save your work with the maize data
# use for example:
# save.image("Boxes14to22.RData")

##### Box 22 (page 58; my mistake to repeat box numbers!)
### I will begin with a "fresh" (empty) R environment.
### Remember you mis be in directory "CabanaR".
# In MY case
# setwd("Documents/Cursos/2019/6_JuneCABANA/CABANA/CabanaR")
# Load the file "ChiliStuff.RData":
load("ChiliStuff.RData")
chili.stuff # A vector with the names of the objects that we loaded
# Let's see what we have
ac.data
ac.type

# Now, the "chili" object is large:
class(chili)
nrow(chili)
head(chili)

# id - Numerical identifier of the gene
# How many different genes do we have?
length(unique(chili$id))
length(unique(chili$ac)) # Number of accessions
unique(chili$ac) # Accessions

# Important: Not all genes are expressed in all accessions!, see
tapply(chili$id, chili$ac, length)
chili[chili$id==3,] # All data for the gene with id==3

# It will be more convenient to add the type (D, W or C) to the "ac"
for(i in 1:8){
chili$ac[chili$ac==ac.type$ac[i]] <- ac.type$key[i]
}

# Make a little experiment
# (obtain dendrograms based in expression of only one gene)
temp <- chili[chili$id==3,]
temp$ac
```

```
temp2 <- temp$ac
temp <- as.matrix(temp[,3:9])
attributes(temp)$dimnames[[1]] <- temp2
round(dist(temp),2)

# Plot the dendrograms with different methods
plot(hclust(dist(temp), method="complete")) # Figure 13 A
plot(hclust(dist(temp), method="average")) # Figure 13 B
plot(hclust(dist(temp), method="single")) # Figure 13 C
plot(hclust(dist(temp), method="ward.D")) # Figure 13 D

##### Box 23
# Commands to plot gene expression for gene with id=3
# (data are in the "temp" object)
plot(seq(0,60,10), temp[1,], col=rainbow(8)[1], type="l",
ylim=c(min(as.vector(temp)), max(as.vector(temp))),
lwd=2, xlab="Time (DAA)",
ylab="Normalized gene expression",
main="Gene expression of gene with id=3")
for(i in 2:8){
points(seq(0,60,10), temp[i,], type="l", col=rainbow(8)[i], lwd=2, lty=i)
}
legend(25,2.5, bty="n", legend=attributes(temp)$dimnames[[1]], lwd=2,
                  lty=c(1:8), col=rainbow(8))
# Presented as Figure 14

##### Box 24
# Finding the set of genes expressed in all accessions
ac.type$key
# Let's defines the sets of genes expressed in each accession
# using assign()
for(i in 1:8){
assign(paste("in", ac.type$key[i], sep="."), chili$id[chili$ac==ac.type$key[i]])
}
ls(patt="in") # The objects that were created by assign
length(in.CM.D) # Number of ids (expressed genes) in CM.D


# We need the intersection of ALL 8 sets; let's do it by parts
in.all <- intersect(in.CM.D, in.CO.W) # The first two
in.all <- intersect(in.all, in.CQ.C) # The third... (and so on)
in.all <- intersect(in.all, in.CW.D)
in.all <- intersect(in.all, in.QC.C)
in.all <- intersect(in.all, in.QU.W)
in.all <- intersect(in.all, in.ST.D)
in.all <- intersect(in.all, in.ZU.D)
length(in.all) # Number of genes expressed in all 8 accessions

head(in.all)
tail(in.all)

# Now we can define the set of genes that are not expressed in all accessions
not.in.all <- setdiff(unique(chili$id), in.all)
length(not.in.all) # Number of genes not expressed in some accessions
head(not.in.all)

# Now, define the logical variable "in.all" in "chili"
chili <- data.frame(chili, in.all=rep(TRUE, nrow(chili)), stringsAsFactors=F)
# And update to FALSE for all cases in "not.in.all"
for(i in 1:length(not.in.all)){
chili$in.all[chili$id == not.in.all[i]] <- FALSE
}
table(chili$in.all) # Number of cases
table(chili$in.all, chili$ac) # Number of cases

##### Box 25
```

```
# Considering all expressed genes per accession
# Note that there is interesting variation on times...
round(apply(chili[chili$in.all, 3:9], 2, mean), 2) # For all accessions
round(apply(chili[(chili$in.all)&(chili$ac=="CM.D"), 3:9], 2, mean),2) # Only one
ac.type$key[1]

# Let's put the means per accession in a matrix for each time
chili.all.mean <- matrix(NA, nrow=8, ncol=7, dimnames=list(ac.type$key, names(chili)
[3:9]))
# Now fill that matrix
for(i in 1:8){
chili.all.mean[i, ] <- apply(chili[(chili$in.all)&(chili$ac==ac.type$key[i]), 3:9], 2,
mean)
}

round(chili.all.mean,2) # To see our results

# Now we will do the same for the median
chili.all.median <- matrix(NA, nrow=8, ncol=7, dimnames=list(ac.type$key, names(chili)
[3:9]))
# Now fill that matrix
for(i in 1:8){
chili.all.median[i, ] <- apply(chili[(chili$in.all)&(chili$ac==ac.type$key[i]), 3:9],
2, median)
}

round(chili.all.median,2) # To see our results

# Let's plot the means (Figure 15)
plot(seq(0,60,10), chili.all.mean[1,], col=rainbow(8)[1], type="l",
ylim=c(min(as.vector(chili.all.mean)), max(as.vector(chili.all.mean))),
lwd=2, xlab="Time (DAA)",
ylab="Mean of normalized gene expression",
main="Average of gene expression of 25,645 genes\nper time and accession")
for(i in 2:8){
points(seq(0,60,10), chili.all.mean[i,], type="l", col=rainbow(8)[i], lwd=2, lty=i)
}
legend("bottomleft", bty="n", legend=attributes(chili.all.mean)$dimnames[[1]],
                  lwd=2, lty=c(1:8), col=rainbow(8))

# And obtain the dendrogram (Figure 16).
library(pvclust) # To evaluate "robustness"
? pvclust # Remember the use
# Without robustness measures
plot(hclust(dist(chili.all.mean), "average"))


system.time(
# Because ic can take some time
chili.all.mean.pv <- pvclust(t(chili.all.mean), method.hclust="average",
                  method.dist="euclidean", nboot=1000)
)

chili.all.mean.pv
# Figure 16
plot(chili.all.mean.pv)

# With medians
chili.all.median.pv <- pvclust(t(chili.all.median), method.hclust="average",
                  method.dist="euclidean", nboot=1000)

# Figure 17
plot(seq(0,60,10), chili.all.median[1,], col=rainbow(8)[1], type="l",
ylim=c(min(as.vector(chili.all.median)), max(as.vector(chili.all.median))),
lwd=2, xlab="Time (DAA)",
ylab="Median of normalized gene expression",
```

```
main="Median of gene expression of 25,645 genes\nper time and accession")
for(i in 2:8){
points(seq(0,60,10), chili.all.median[i,], type="l", col=rainbow(8)[i], lwd=2, lty=i)
}
legend("topright", bty="n", legend=attributes(chili.all.median)$dimnames[[1]], lwd=2,
                  lty=c(1:8), col=rainbow(8), cex=0.75)
# Figure 18
plot(chili.all.median.pv)

##### Box 26
ac.type$key
# We can isolate the data only for expressed genes
names(chili) # Remember the column names

temp <- chili[chili$in.all, 1:9]
nrow(temp)
nrow(temp)/8 # Number of genes expressed in all accessions

# We need to be sure that the order of the identifiers
# (id) is the same for all 8 accessions
# All values must be 1 (8*7/2=28 comparisons)
for(i in 1:7){
for(j in (i+1):8){
print(prod(temp$id[temp$ac==ac.type$key[i]] ==  temp$id[temp$ac==ac.type$key[j]]))
}}

# Now we can form the matrices, one for each time
e.t0 <- matrix(NA, nrow=8, ncol=25645, dimnames=list(ac.type$key,
                  temp$id[temp$ac==ac.type$key[1]]))
e.t0[1:3,1:5]

# The other empty matrices
e.t10<-e.t0;e.t20<-e.t0;e.t30<-e.t0;e.t40<-e.t0;e.t50<-e.t0;e.t60<-e.t0

# Fill the matrices
for(i in 1:8){
e.t0[i,] <- as.vector(temp$zT0[temp$ac==ac.type$key[i]])
e.t10[i,] <- as.vector(temp$zT10[temp$ac==ac.type$key[i]])
e.t20[i,] <- as.vector(temp$zT20[temp$ac==ac.type$key[i]])
e.t30[i,] <- as.vector(temp$zT30[temp$ac==ac.type$key[i]])
e.t40[i,] <- as.vector(temp$zT40[temp$ac==ac.type$key[i]])
e.t50[i,] <- as.vector(temp$zT50[temp$ac==ac.type$key[i]])
e.t60[i,] <- as.vector(temp$zT60[temp$ac==ac.type$key[i]])
}

# Now we can construct the dendrograms
# Note that the distances will be very large
# (they are distance in an space of 25645 dimensions!)
# Example
dist(e.t0)

# Constructing the dendrograms
d.t0 <- hclust(dist(e.t0), "average")
d.t10 <- hclust(dist(e.t10), "average")
d.t20 <- hclust(dist(e.t20), "average")
d.t30 <- hclust(dist(e.t30), "average")
d.t40 <- hclust(dist(e.t40), "average")
d.t50 <- hclust(dist(e.t50), "average")
d.t60 <- hclust(dist(e.t60), "average")

# Plotting the dendrogram
# (figures 19 and 20)

# Figure 19
plot(d.t0, main="Cluster at time 0 DAA")
plot(d.t10, main="Cluster at time 10 DAA")
```

```
plot(d.t20, main="Cluster at time 20 DAA")
plot(d.t30, main="Cluster at time 30 DAA")

# Figure 20
plot(d.t40, main="Cluster at time 40 DAA")
plot(d.t50, main="Cluster at time 50 DAA")
plot(d.t60, main="Cluster at time 60 DAA")

##### Box 27
# Being in your CabanaR/ directory load the functions to compare dendrograms
source("LikenessFun.txt") # Load the text file that contains the functions.
likeness # A vector with the names of the functions

# Before proceeding, remember the names of the objects that contain the
# dendrograms of figures 19 and 20:

# Let's see and briefly analyze the function "hclust2sets"
ls(patt="d.t") # Output not shown

# How an hclust object is composed?
class(d.t0)
names(d.t0)
d.t0$merge # The matrix of "merges"
d.t0$labels # The vector of labels (units in the dendrogram)

# Let's now see the function "hclust2sets"
hclust2sets

# Let's try that function on two of our dendrograms
hclust2sets(d.t0) # Dendrogram of Figure 19 A seen as sets:

# Check this result visually with Figure 19 A.
hclust2sets(d.t10) # Dendrogram of Figure 19 b seen as sets:

# Can you say which sets (clusters) are shared?

# Now, let's see function "compare.hclust"
compare.hclust

# And let's try that function (with alternative parameters)
compare.hclust(x=d.t0, y=d.t10, give.matrix = FALSE, do.cat = FALSE)
compare.hclust(x=d.t0, y=d.t10, give.matrix = FALSE, do.cat = TRUE)
compare.hclust(x=d.t0, y=d.t10, give.matrix = TRUE, do.cat = TRUE)

##### Box 28
# Dendrograms in figures 19 and 20 are 7 we can make all 7*6/2 = 21 comparisons
# Let's make a data frame to keep the results
comp.dend <- data.frame(d1=rep("",21), d2=rep("",21), lik=rep(NA,21),
stringsAsFactors=F)

temp.list <- list(d.t0, d.t10, d.t20, d.t30, d.t40, d.t50, d.t60)
names(temp.list) <- c("d.t0", "d.t10", "d.t20", "d.t30", "d.t40", "d.t50", "d.t60")

k <- 0
for(i in 1:6){
for(j in (i+1):7){
k <- k+1
comp.dend$d1[k] <- names(temp.list)[i]
comp.dend$d2[k] <- names(temp.list)[j]
comp.dend$lik[k] <- compare.hclust(temp.list[[i]], temp.list[[j]])
}
}

head(comp.dend)
summary(comp.dend$lik)
comp.dend[comp.dend$lik>0,]
```

```
# The "more alike" dendrograms share 50% of their clusters
1*compare.hclust(d.t10, d.t50, give.matrix=T)
# (here we convert FALSE to 0 and TRUE to 1)

hclust2sets(d.t10)
hclust2sets(d.t50)

##### Box 29
likeness.boot <-
                  function(de=d.t0, da=e.t0, B=100){
                  # likeness.boot
                  # Obtains a vector with likeness between an original dendrogram: de
                  # (de must be a dendrogram produced by "hclust" from matrix "da")
                  # and B bootstrap replicates of such dendrogram.

                  # Recover the method and distance method.
                  me <- de$method
                  dis.me <- de$dist.method
                  res <- rep(NA, B) # An empty vector to contain the results.

                  n.c <- ncol(da)

                  # Loop to obtain each value of likeness using function
                  # compare.hclust(x = d.t0, y = d.t10, give.matrix = FALSE, do.cat =
FALSE)
                  for(i in 1:B){
                                    sam <- sample(c(1:n.c), size=n.c, replace=TRUE)
                                    db <- hclust(dist(da[,sam], method=dis.me),
method=me)
                                    res[i] <- compare.hclust(x = de, y = db,
give.matrix = FALSE, do.cat = FALSE)
                  }
                  res
}
### NOTE: The function is also in file "likeness_boot.txt"
# You can use source("likeness_boot.txt") to load it in your environment.

### You can save the work done using, for example
## save.image("Boxes22to29.RData")
```