Class 5 - Programming in R

1. Conditional expressions

When building specific functions or a script, some conditions may be explicitly stated to allow specific operations.

1.1. "if" statement

Generically, this expression is written this form:

```
if (test_expression) {
  statement
}
```

```r
#example

x <- 5
if(x > 0){
print("Positive number")
}
```

```
## [1] "Positive number"
```

```r
# Check if a numeric object is even.
# [The remainder of an even number divided by 2 is 0]
# [operator %% outputs the remainder of a division]

y <- 6

if(y %% 2 == 0){
print("This number is even")
}
```

```
## [1] "This number is even"
```

```r
if(x %% 2 == 0){
print("This number is even")
}
```

1.2. "if/else" statement

The presence of the "if", conditions the script to a specific path. Somtimes there's also a possibility to add one or multiple conditional options.

```
if (test_expression) {
  statement1
} else {
  statement2
}

- "if this condition is met, go this way, else got the other way"
```

```r
#if...else statement
```

```r
x <- -5
if(x > 0){
  print("Non-negative number")
  } else {
  print("Negative number")
}
```

```
## [1] "Negative number"
```

There is also the possibility of adding multiple alternative pathways

```
if ( test_expression1) {
  statement1
} else if ( test_expression2) {
  statement2
} else if ( test_expression3) {
  statement3
} else {
  statement4
}
```

Yet, in this case all these conditions need to be mutually exclusive since only one statement will get executed depending upon the test_expressions.

```r
x <- 0

if (x < 0) {
  print("Negative number")
} else if (x > 0) {
  print("Positive number")
} else
  print("Zero")
```

```
## [1] "Zero"
```

Activity 5.1

- a.1) Complete the function "sum_eval()", that takes a numeric (integer) vector of any length, sums all the elements and prints "Sum is even" or "Sum is odd".

```r
# sum_eval <- function(arg){
#   sum.res <- sum(arg)
#   if ( _________ ) { __________ }  else { ________ }
# }

#### test the function
# vector_a <- c( 34, 56, 25,64,51, 55, 89)
# vector_b <- c( 78, 43, 90, 64, 3, 34, 89)

# sum_eval(vector_a)
```

- a.2) Complete the function "itqb_search()", that takes a vector of words, of any length, and prints "itqb is present" if "itqb" is one of the words present, or "no hit" if not present. *hint:function tolower converts all character srings to lower case*

```r
# itqb_search <- function( ___ ){
#   if ( _____ %in% tolower( arg ) ) { desision <- _____ }  else { decision <- ____ }
# return(decision)
# }

## test the function
# vector_c <- c("Champalimaud", "IGC", "IMM", "IBMC", "CIBIO")
# vector_d <- c("ITQB", "open", "day")
# itqb_search(vector_c)
```

- a.3) Write a function that takes two arguments, a numeric p-value and a significance value, and evaluates if H0 should be rejected, using alpha of 0.05.

2. for loops

Loops are used in programming to repeat a specific block of code, a specific amount of times

A for loop is usually used to iterate over a vector.

```
for i in vector {
... statement ...
}
```

if the vector has 5 elements, a cycle will be created and the statement will be run 5 times. At every cycle, i will iterate over the elements of the vector

```r
random_vector <- c(45,3333333,345,1,0)

for (i in random_vector) {
  print(i)
  print("end of cycle")
}
```

```
## [1] 45
## [1] "end of cycle"
## [1] 3333333
## [1] "end of cycle"
## [1] 345
## [1] "end of cycle"
## [1] 1
## [1] "end of cycle"
## [1] 0
## [1] "end of cycle"
```

```r
print("new vector")
```

```
## [1] "new vector"
```

```r
name_vector <- c("Ana","Joao","Miguel")

for (i in name_vector) {
  print(i)
  print("end of cycle")
}
```

```
## [1] "Ana"
## [1] "end of cycle"
```

```
## [1] "Joao"
## [1] "end of cycle"
## [1] "Miguel"
## [1] "end of cycle"
```

We can set a counter inside the loop to count how may cycles were run (length of the vector)

```
n = 0
for (i in random_vector) {
  n = n + 1
  print("couting")
}
```

```
## [1] "couting"
## [1] "couting"
## [1] "couting"
## [1] "couting"
## [1] "couting"
```

```
n
```

```
## [1] 5
```

If statements and for loops can be used in the same operations

```
# screen a vector and count the number of numbers higher than 5
random_vector <-c(3.4, 5.5, 4.9, 5.6, 6.6, 7.8, 1.3, 6.7)
count <- 0

for (i in random_vector) {
  if (i > 4) {
    count <- count + 1
  }
}
count
```

```
## [1] 6
```

Activity 5.2

a.4) complete the following function that takes a numeric vector a counts the number of even numbers inside.

```
# myEven <- function( v ) {
#   count = ____
#   for (number in _____ ) {
#     if ( _____ ) { count = count +1 }
#   }
#   return(count)
# }

# vector_a <- c(2,3,6,5,4,56,67,86)
#myEven(vector_a)
```

a.5) write a function that takes a vector of numbers higher and lower than 0 and returns another vector only with positive values.

hint: you need to create an empty vector before starting the loop *hint: to include 34 in a vector: vector <- c(vector, 34)*

```
myPositive <- function(v){
  positive <- c()
  for (element in v) {
    if (element > 0){
      positive <- c(positive, element)
    }
  }
return(positive)
}

test_vector <- c(-1,2, -3, 5, 6, -15, 56)
myPositive(test_vector)
```

```
## [1]  2  5  6 56
```

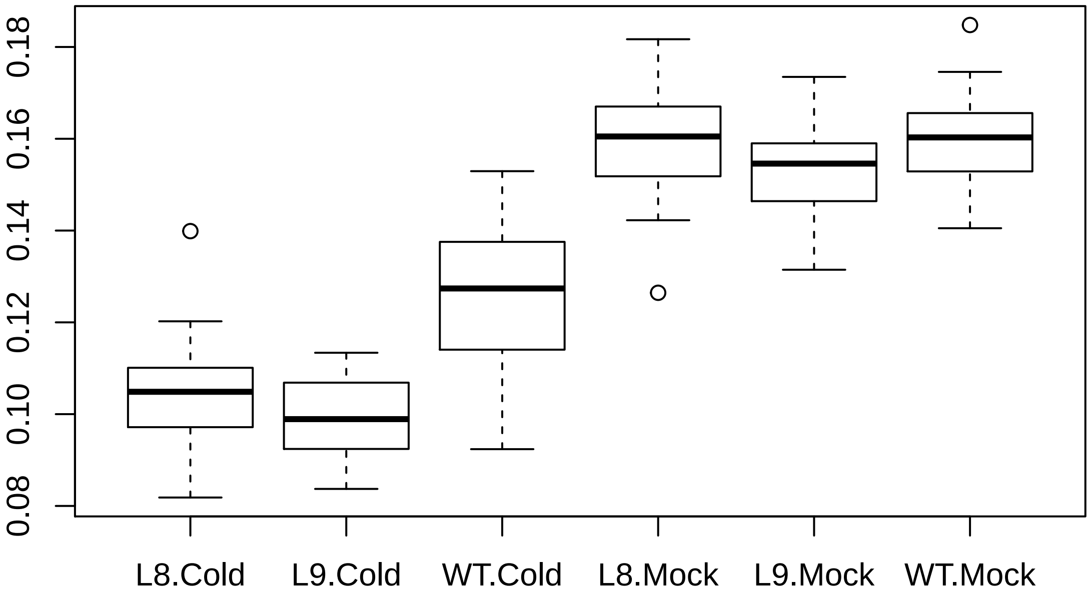
3. Automated analysis

Things to think before starting build a script to analysi multiple data: - files to analyse must be structured - names of files should be also structured (if they contain information in the ID of the samples is a plus) - define the output : in this case, iteratively import csv files to extract the only the first column, sample name and condition, and add this information in a final table that merges the information from all files. The final table should contain three columns:

using list.file() we can list the files of interest (that match a pattern) and save these names in an object

```
files <- list.files(path = "./", pattern = "201808")

# Now we will create a function to iterate over all files and collect the informaton

makeTable <- function(files) {

  #first we open an empty table to host the values of interest
  table <- data.frame(sample = factor(),
                      condition = factor(), ETR = numeric())
  # then we start the loop
  for (file in files){
    # open a file
    df1 <- read.csv( file, row.names = 1 )

  # decompose the name of the file to remove IDs
  # strsplit() separates the name of the file by "_"
  # unlist convert the result of strsplit() into a vector to facilitate indexing
    file_data <- unlist(strsplit(file, "_"))
    # Now we would like to change Control to Mock
    if (file_data[1] == "Control") { file_data[1] <- "Mock" }
  # collect all ETR measurements, they are always in column 1
    ETR_m <- df1[,1]
  # create a vector with the factor Control/Cold contained in file,
  # with the same length as the ETR measurements
    condition <- rep(file_data[1], length(ETR_m))
  # do the same with line
    sample <- rep(file_data[2], length(ETR_m))
  # create a temporary dataframe by binding all three vectors by column
```

```
    temp_df <- cbind.data.frame(sample , condition, ETR_m)
  # bind temporary dataframe to the master_df
  # we will bind by rows
    table <- rbind(table, temp_df)
  }
  return(table)
}

ETR_table<- makeTable(files)

#create a new column with a new factor resulting from the combination of both factors

ETR_table$samplecondition <- interaction(ETR_table$sample, ETR_table$condition)

boxplot(formula = ETR_table$ETR_m ~ ETR_table$samplecondition)
```



In this case we have two factors (condition, sample) for one continuous variable (ETR_m), so we can compute a Two-way ANOVA. For the sake of brevity, let's assume that samples follow a normal distribution and have equal variances.

rule of thumb: ANOVA is robust to heterogeneity of variance so long as the largest variance is not more than 4 times the smallest variance

```
res.aov <- aov(ETR_m ~ condition + sample, data = ETR_table)
summary(res.aov)
```

```
##                Df  Sum Sq Mean Sq F value   Pr(>F)
## condition       1 0.06783 0.06783  371.60  < 2e-16 ***
## sample          2 0.00582 0.00291   15.93 7.72e-07 ***
## Residuals     116 0.02117 0.00018
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
TukeyHSD(res.aov)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
```

```
##
## Fit: aov(formula = ETR_m ~ condition + sample, data = ETR_table)
##
## $condition
##                 diff        lwr       upr p adj
## Mock-Cold 0.04754964 0.04266407 0.0524352     0
##
## $sample
##               diff          lwr          upr     p adj
## L9-L8 -0.006247672 -0.013420168 0.0009248243 0.1010317
## WT-L8  0.010616178  0.003443682 0.0177886745 0.0018117
## WT-L9  0.016863850  0.009691354 0.0240363466 0.0000005
```

```r
res.aov <- aov(ETR_m ~ condition + sample + condition * sample, data = ETR_table)
summary(res.aov)
```

```
##                   Df  Sum Sq Mean Sq F value   Pr(>F)
## condition          1 0.06783 0.06783 418.201  < 2e-16 ***
## sample             2 0.00582 0.00291  17.926  1.7e-07 ***
## condition:sample   2 0.00268 0.00134   8.274 0.000441 ***
## Residuals        114 0.01849 0.00016
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

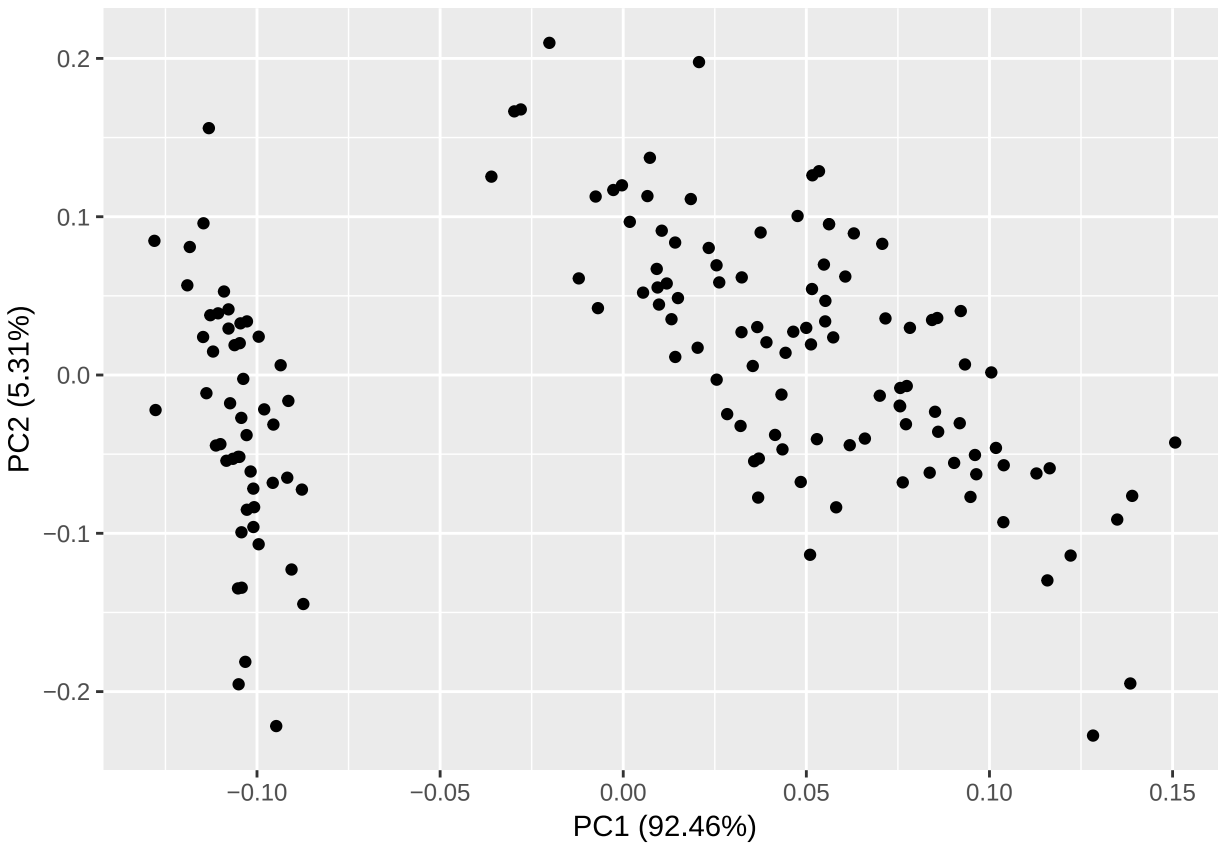
```r
TukeyHSD(res.aov)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = ETR_m ~ condition + sample + condition * sample, data = ETR_table)
##
## $condition
##                 diff        lwr        upr p adj
## Mock-Cold 0.04754964 0.04294349 0.05215578     0
##
## $sample
##               diff          lwr          upr     p adj
## L9-L8 -0.006247672 -0.013010240 0.0005148954 0.0765216
## WT-L8  0.010616178  0.003853611 0.0173787457 0.0008766
## WT-L9  0.016863850  0.010101283 0.0236264178 0.0000001
##
## $`condition:sample`
##                          diff          lwr          upr     p adj
## Mock:L8-Cold:L8  5.548453e-02  0.043810259  0.067158806 0.0000000
## Cold:L9-Cold:L8 -4.959737e-03 -0.016634010  0.006714537 0.8205056
## Mock:L9-Cold:L8  4.794892e-02  0.036274652  0.059623198 0.0000000
## Cold:WT-Cold:L8  2.123059e-02  0.009556314  0.032904861 0.0000095
## Mock:WT-Cold:L8  5.548630e-02  0.043812028  0.067160574 0.0000000
## Cold:L9-Mock:L8 -6.044427e-02 -0.072118542 -0.048769996 0.0000000
## Mock:L9-Mock:L8 -7.535608e-03 -0.019209881  0.004138666 0.4251938
## Cold:WT-Mock:L8 -3.425394e-02 -0.045928218 -0.022579672 0.0000000
## Mock:WT-Mock:L8  1.768782e-06 -0.011672504  0.011676042 1.0000000
## Mock:L9-Cold:L9  5.290866e-02  0.041234388  0.064582934 0.0000000
## Cold:WT-Cold:L9  2.619032e-02  0.014516051  0.037864597 0.0000000
## Mock:WT-Cold:L9  6.044604e-02  0.048771765  0.072120311 0.0000000
```

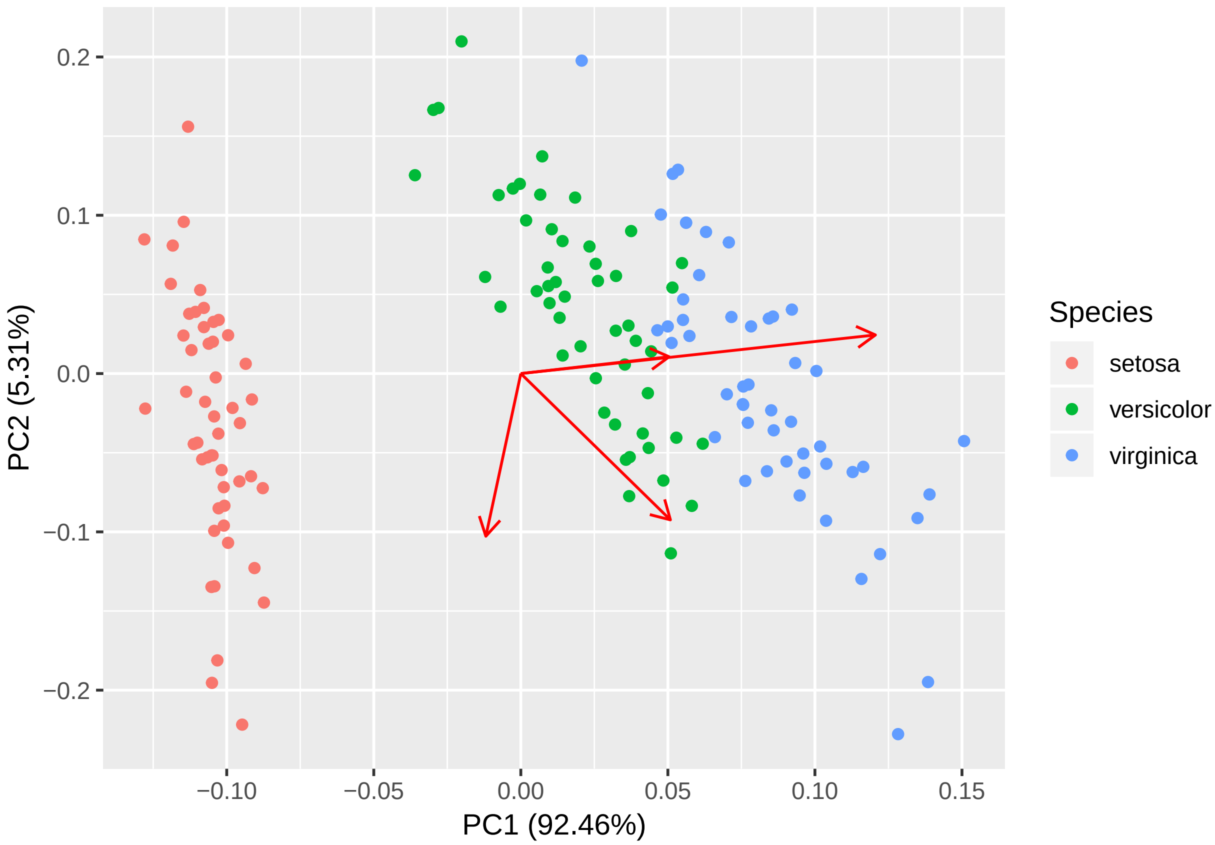
```
## Cold:WT-Mock:L9 -2.671834e-02 -0.038392611 -0.015044064 0.0000000
## Mock:WT-Mock:L9  7.537376e-03 -0.004136897  0.019211650 0.4249226
## Mock:WT-Cold:WT  3.425571e-02  0.022581441  0.045929987 0.0000000
```

3. EXTRA: Building PCA for multivariate analysys

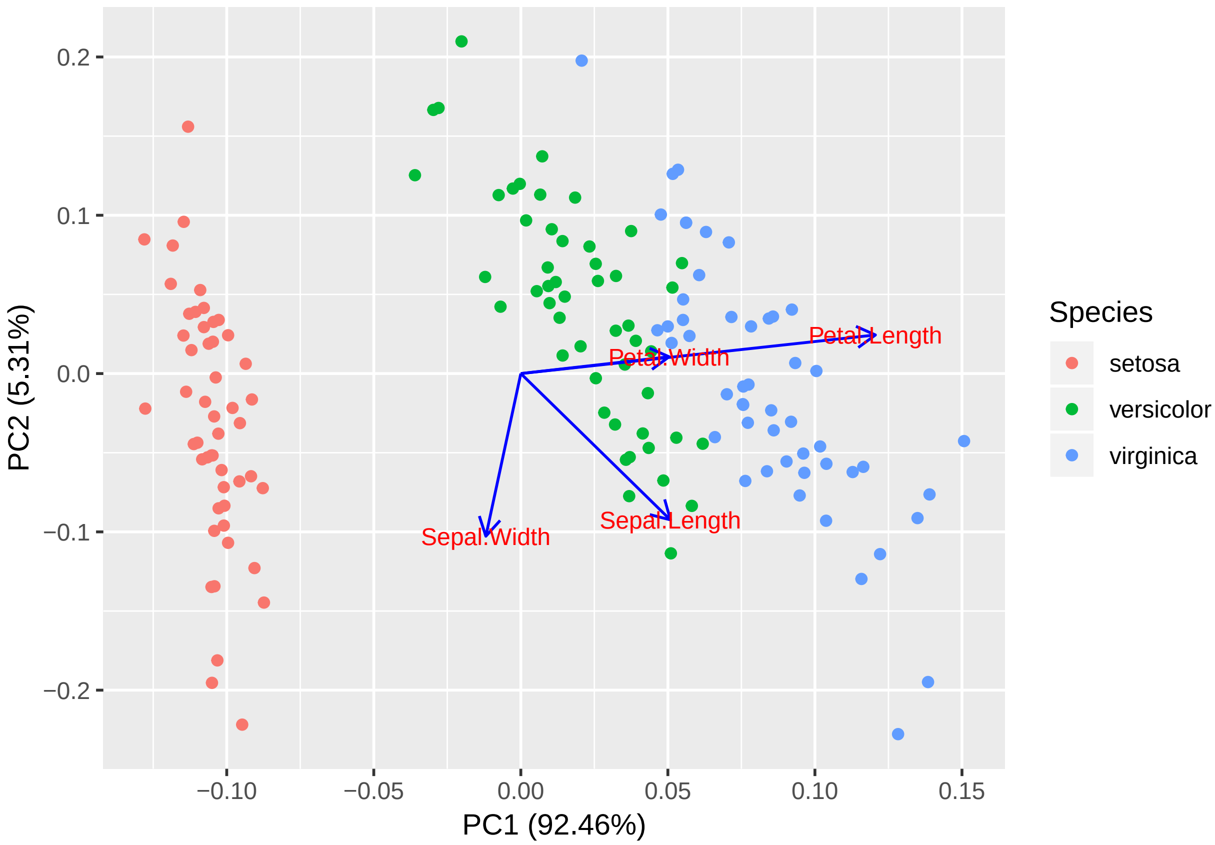
```r
#install.packages("ggplot2")
#install.packages("ggfortify")

library(ggplot2)
library(ggfortify)

df <- iris[c(1, 2, 3, 4)]

autoplot(prcomp(df))
```




```r
autoplot(prcomp(df), data = iris, colour = 'Species',
         loadings = TRUE)
```




```r
autoplot(prcomp(df), data = iris, colour = 'Species',
         loadings = TRUE, loadings.colour = 'blue',
         loadings.label = TRUE, loadings.label.size = 3)
```


0.2
0.1
0.0
−0.1
−0.2
PC2 (5.31%)
−0.10
−0.05
0.00
0.05
0.10
0.15
PC1 (92.46%)
Petal.Length
Petal.Width
Sepal.Width
Sepal.Length
Species
setosa
versicolor
virginica