notes3

2022-06-11

Subsetting matrices

```
m <- matrix(1:12, nrow=3)
m
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
m_sub <- m[2:3, 1:2] #selecting for rows 2 to 3, and for columns 1 to 2
m_sub
```

```
##      [,1] [,2]
## [1,]    2    5
## [2,]    3    6
```

```
m_sub1 <- m[c(1,3), c(2,4)] #selecting for rows 1 and 3, and for columns 2 and 4
m_sub1
```

```
##      [,1] [,2]
## [1,]    4   10
## [2,]    6   12
```

If we are only interested either in certain rows or columns, but not both, let's do:

```
m_sub2 <- m[c(1,3), ] #all columns are kept
m_sub3 <- m[ , c(2,4)] #all rows are kept
m_sub2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    3    6    9   12
```

```
m_sub3
```

```
##      [,1] [,2]
## [1,]    4   10
## [2,]    5   11
## [3,]    6   12
```

CAREFUL NOW, as we are selecting either only 1 row or 1 column:

```
m_sub4 <- m[2, ] #only row 2 is selected
m_sub5 <- m[ , 1] #only column 1 is selected
m_sub4
```

```
## [1]  2  5  8 11
```

```
m_sub5
```

```
## [1] 1 2 3
```

```
class(m_sub4)
```

```
## [1] "integer"
```

```
dim(m_sub4)
```

```
## NULL
```

So basically when only 1 row or 1 column is selected, we no longer deal with a matrix but with an integer vector. If instead we wish to preserve the matrix as class, we need to subset in this way:

```
m_sub6 <- m[2, , drop=FALSE]
m_sub6
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    5    8   11
```

```
class(m_sub6)
```

```
## [1] "matrix" "array"
```

```
dim(m_sub6)
```

```
## [1] 1 4
```

Rows and columns' names

As for vectors, also the elements within a matrix (rows and columns) can have names:

```
mat <- matrix(1:12, nrow=4)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```r
rownames(mat) <- c("bpm", "pO2", "Hb", "age")
colnames(mat) <- c("John", "Hannah", "Serena")
mat
```

```
##     John Hannah Serena
## bpm    1      5      9
## pO2    2      6     10
## Hb     3      7     11
## age    4      8     12
```

So then you can call (subset) the rows/columns by their names:

```r
mat["bpm", "Serena"] #to select row 1 and column 3
```

```
## [1] 9
```

```r
mat[c("pO2", "age"), c("Hannah", "Serena")]
```

```
##     Hannah Serena
## pO2      6     10
## age      8     12
```

Data frames

Main way to represent datasets. Collections of observations: the same variables are observed for many individuals.

```r
class(iris)
```

```
## [1] "data.frame"
```

```r
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Each row = 1 observation (a flower), each column = 1 variable. **Data frames are different from matrices because matrices are vectors**: data frames instead can host elements of different type (we can say they are lists represented in a matrix form). In the 'iris' dataset we have: - 4 numerical, continuous variables (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width) - 1 categorical variable (Species)

Different data frame:

```
load("C:/Users/seren/Desktop/Provero/clinical.RData")
head(clinical)
```

```
##        patient gender              histological_type StGallen2013 rfs_time
## 1 TCGA-3C-AAAU FEMALE INFILTRATING LOBULAR CARCINOMA         <NA>     1808
## 2 TCGA-3C-AALI FEMALE  INFILTRATING DUCTAL CARCINOMA    LumB_HER2     4005
## 3 TCGA-3C-AALJ FEMALE  INFILTRATING DUCTAL CARCINOMA    LumB_HER2     1474
## 4 TCGA-3C-AALK FEMALE  INFILTRATING DUCTAL CARCINOMA    LumB_HER2     1448
## 5 TCGA-4H-AAAK FEMALE INFILTRATING LOBULAR CARCINOMA         LumA      348
## 6 TCGA-5L-AATO FEMALE INFILTRATING LOBULAR CARCINOMA         <NA>     1477
```

where for each patient there are:

- 1 character variable (patient code)
- 3 categorical variables (gender, histological_type, StGallen2013 molecular classification)
- 1 numerical variable (rfs_time, which is recurrence-free survival time)

To generate a data frame:

```
x <- 1:7
y <- c("One", "Two", "Three", "Four", "Five", "Six", "Seven")
z <- c(T, F, T, F, T, F, T)

df <- data.frame(number=x, character=y, logical=z)
df
```

```
##   number character logical
## 1      1       One    TRUE
## 2      2       Two   FALSE
## 3      3     Three    TRUE
## 4      4      Four   FALSE
## 5      5      Five    TRUE
## 6      6       Six   FALSE
## 7      7     Seven    TRUE
```

```
class(df)
```

```
## [1] "data.frame"
```

```
dim(df)
```

```
## [1] 7 3
```

As with lists, you can extract elements from a data frame with:

```
df$number #or again using dm[[1]]
```

```
## [1] 1 2 3 4 5 6 7
```

```
df$character #or by calling dm[[2]]
```

```
## [1] "One"   "Two"   "Three" "Four"  "Five"  "Six"   "Seven"
```

```
df1 <- df[ , "number"] #you are subsetting the dataframe, selecting all rows and just the column 'numbe
df2 <- df[1 , "character"] #subsetting the dataframe selecting row 1 and column 'character'
df1
```

```
## [1] 1 2 3 4 5 6 7
```

```
df2
```

```
## [1] "One"
```

Both ‘df2’ and ‘df3’ are vectors. How to keep them a data frame?

```
df4 <- df[ , "number", drop=FALSE]
df4
```

```
##   number
## 1      1
## 2      2
## 3      3
## 4      4
## 5      5
## 6      6
## 7      7
```

Note that **if you select only 1 row instead, you still get a data frame as output** (because rows contain elements of different types, while columns don’t):

```
df5 <- df[1, ]
df5
```

```
##   number character logical
## 1      1       One    TRUE
```

```
class(df5)
```

```
## [1] "data.frame"
```

```
dim(df5)
```

```
## [1] 1 3
```

Factors

Class used to describe *categorical variables* (i.e. they represent one of a finite number of possible states). Let’s recall:

```r
x <- 1:7
y <- c("One", "Two", "Three", "Four", "Five", "Six", "Seven")
z <- c(T, F, T, F, T, F, T)
df <- data.frame(number=x, character=y, logical=z)
```

```r
class(x)
```

```
## [1] "integer"
```

```r
class(y)
```

```
## [1] "character"
```

```r
class(z)
```

```
## [1] "logical"
```

```r
class(df)
```

```
## [1] "data.frame"
```

When a character vector (y) is included in a data frame, it's transformed into a factor when the `stringsAsFactors` parameter is set TRUE (by default it is FALSE):

```r
class(df$character)
```

```
## [1] "character"
```

```r
df1 <- data.frame(number=x, character=y, logical=z, stringsAsFactors=TRUE)
class(df1$character)
```

```
## [1] "factor"
```

Example: differentially expressed genes

```r
load("C:/Users/seren/Desktop/Provero/expr.RData")
class(expr)
```

```
## [1] "matrix" "array"
```

it's okay to have a matrix, no need for a data frame now.

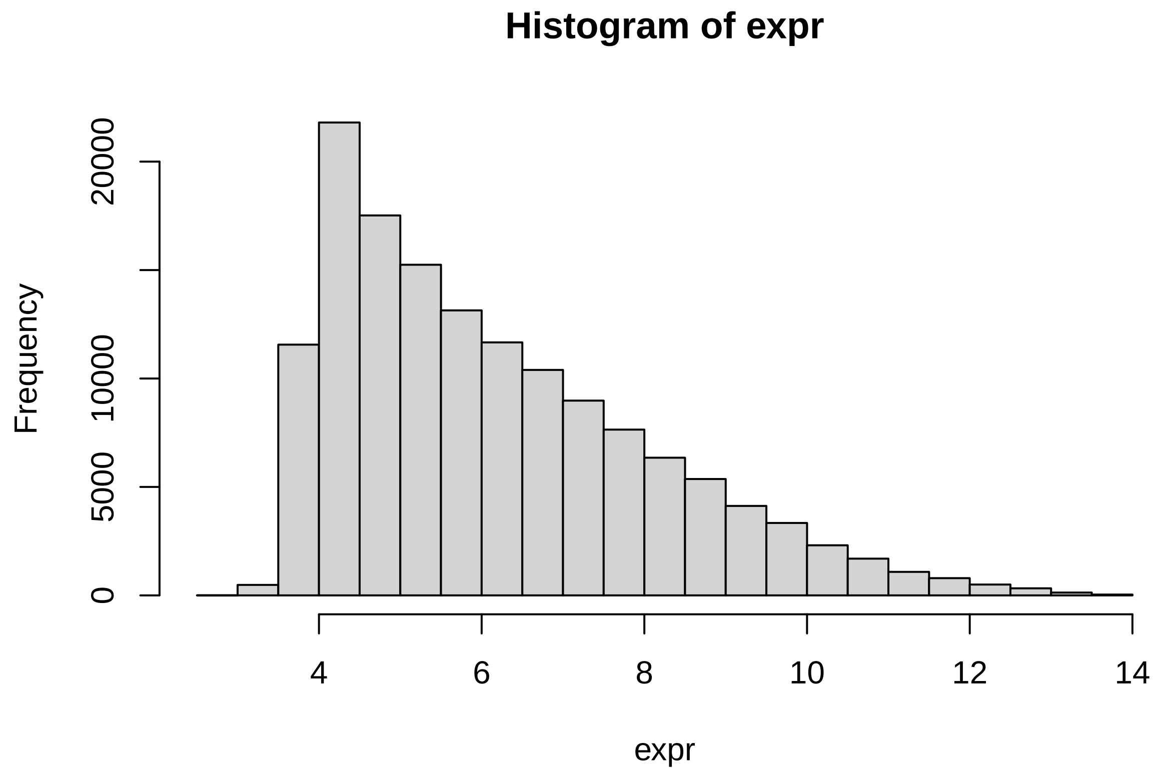
```r
dim(expr)
```

```
## [1] 12042    12
```

```r
head(expr)
```

```
##            mes1     mes2     mes3     mes4     mes5     mes6     neu1     neu2
## RNF14  8.102109 6.431024 7.233297 5.536356 7.980071 8.120170 6.807137 7.481830
## UBE2Q1 9.535052 9.531678 6.666299 6.934365 9.379369 9.046204 9.466937 9.369915
## RNF17  4.149430 4.374377 5.037466 5.135278 4.172273 4.529581 4.609084 4.456974
## RNF10  7.142845 6.882857 5.524656 6.236647 7.379514 6.893025 6.624718 7.057433
## RNF11  9.563294 8.782727 8.522499 6.927014 8.938691 8.707519 8.289256 9.274853
## RNF13  9.552312 9.237758 9.556759 7.951833 9.873957 9.457725 9.294424 9.016314
##             neu3      neu4     neu5      neu6
## RNF14   9.232656  8.474975 9.032406  8.901127
## UBE2Q1  9.171364  9.165563 9.742695  9.236809
## RNF17   4.161883  4.132093 4.293262  4.159367
## RNF10   6.897234  7.229054 6.788821  6.940517
## RNF11   9.317841  9.505812 9.941937  9.478019
## RNF13  10.465560 10.228730 9.599728 10.633883
```

```r
hist(expr)
```



We need to compare the former 6 columns (mes) to the latter 6 (neu), and the two are the two different samples:

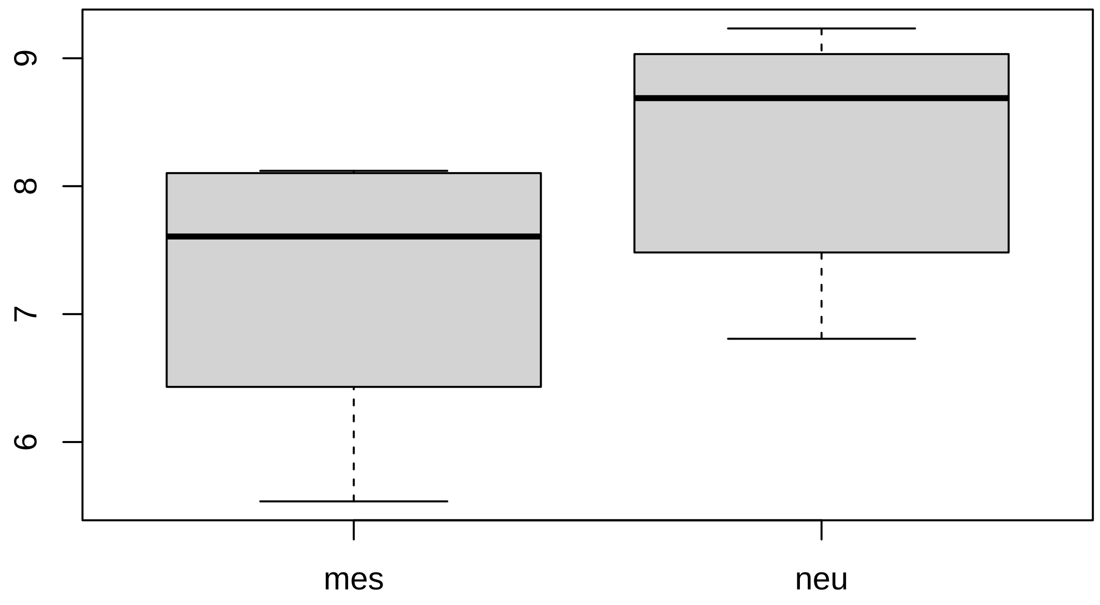
```r
colnames(expr)
```

```
##  [1] "mes1" "mes2" "mes3" "mes4" "mes5" "mes6" "neu1" "neu2" "neu3" "neu4"
## [11] "neu5" "neu6"
```

Let's compare the two samples looking at the expression of 1 gene only (called `g1`):

```
g1 <- expr[1, ] #since the first gene occupies the first row, we select only that row

boxplot(g1[1:6], g1[7:12], names=c("mes", "neu")) #same as doing 'boxplot(expr[1, 1:6], expr[1, 7:12],
```



Is this difference really signficant or is it due to random fluctuations? We'll look for the **P-value**.

```
t <- t.test(g1[1:6], g1[7:12])
t
```

```
##
##  Welch Two Sample t-test
##
## data:  g1[1:6] and g1[7:12]
## t = -1.8552, df = 9.9196, p-value = 0.09349
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.3958560  0.2201548
## sample estimates:
## mean of x mean of y
##  7.233838  8.321689
```

```r
#now we extract the P-value and we put in a variable called 'p':
p <- t$p.value
p
```

```
## [1] 0.093495
```

```r
#however it's better to write a function allowing to extract the P-value for any gene we want, more gen
get_p <- function(x) {
t.test(x[1:6], x[7:12])$p.value
}

get_p(g1)
```

```
## [1] 0.093495
```

The probability of g1 to be more expressed in "mes" sample owing to random fluctuations is ~10%.

To retrieve (extract) the P-values of all genes, the `get_p` should be systematically applied to all rows in this way:

```r
p <- apply(X=expr, MARGIN=1, FUN=get_p)
head(p)
```

```
##     RNF14    UBE2Q1     RNF17     RNF10     RNF11     RNF13
## 0.0934950 0.1866016 0.2102106 0.4300444 0.1228958 0.1515633
```

None of these 6 genes seem to be actually differentially expressed (high P-values).

Now we put these P-values in a data frame:

```r
diffexpr <- data.frame(P=p)
head(diffexpr)
```

```
##                P
## RNF14  0.0934950
## UBE2Q1 0.1866016
## RNF17  0.2102106
## RNF10  0.4300444
## RNF11  0.1228958
## RNF13  0.1515633
```

Other than P-values we need the **fold change** for each gene to know which one are up-regulated and which ones down-regulated:

```r
get_fc <- function(x){
  mean(x[1:6])-mean(x[7:12])
}

# 'x' as function input is again for the row (gene) we're interested in

fc <- apply(X=expr, MARGIN=1, FUN=get_fc) # to systematically apply the function to all rows
head(fc)
```

```
##      RNF14     UBE2Q1      RNF17      RNF10      RNF11      RNF13
## -1.0878506 -0.8433860  0.2642901 -0.2463724 -0.7276621 -0.6013825
```

We used the logFC because data were in log scale (i.e., after logarithmic transformation). Now we add this piece of info (fold change) to the data frame we created with P-values:

```
diffexpr$logFC <- fc
head(diffexpr)
```

```
##                P      logFC
## RNF14  0.0934950 -1.0878506
## UBE2Q1 0.1866016 -0.8433860
## RNF17  0.2102106  0.2642901
## RNF10  0.4300444 -0.2463724
## RNF11  0.1228958 -0.7276621
## RNF13  0.1515633 -0.6013825
```

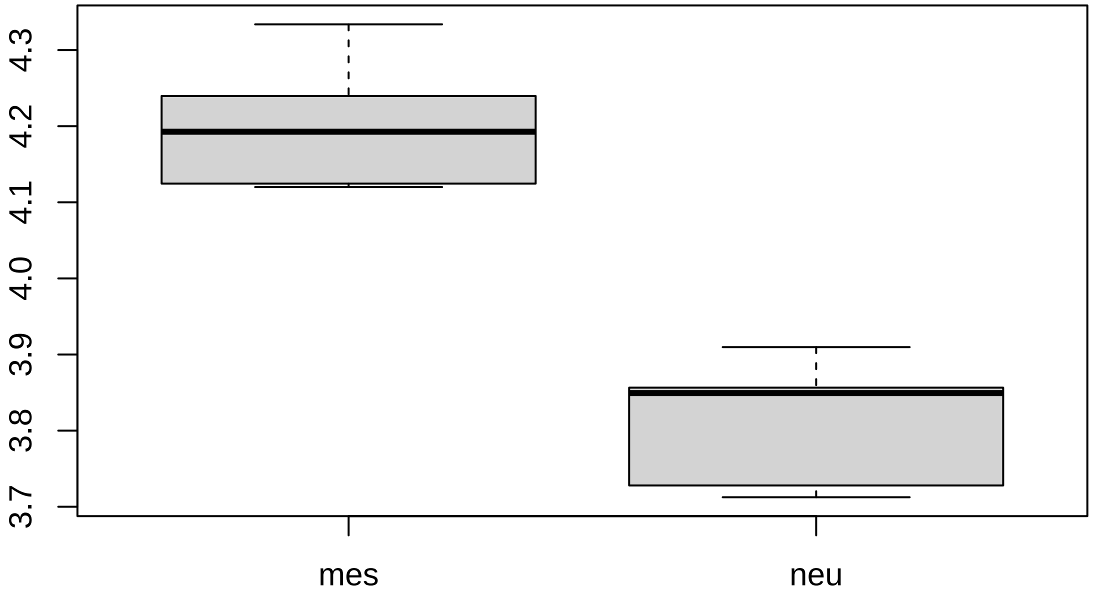
Now let's order the data by decreasing P-value (from the smaller to the grater):

```
diffexpr <- diffexpr[order(diffexpr$P), ] #the space in the end stands for the fact that columns are le
head(diffexpr)
```

```
##                     P      logFC
## SERPINC1 1.085867e-05  0.3830255
## NETO2    4.284471e-05 -3.0850230
## ITCH     5.185839e-05 -1.1894569
## CLDN10   7.505848e-05 -3.5143170
## ERF      8.560227e-05 -0.9486367
## PAK2     1.124295e-04 -1.0847683
```

SERPINC1 is the gene that is most certainly differentially expressed between "mes" and "neu" (smaller P). Let's visualize graphically this:

```
boxplot(expr["SERPINC1", 1:6], expr["SERPINC1", 7:12], names=c("mes", "neu"))
```



Let's evaluate the FC (not the logFC!) for this gene. Remember that $log_2 FC = diffexpr["SERPINC1", "logFC"]$, so that $FC = 2^{diffexpr["SERPINC1", "logFC"]}$:

```
FC <- 2^diffexpr["SERPINC1", "logFC"] #so we have 2 to the power of the value contained in row "SERPINC
FC
```

```
## [1] 1.304074
```