Refresher on the R software

This document introduces basic R command to:

- perform data management
- create graphical display
- create and work with tables (2 by 2 or 2 by 2 by k)

We refer to `http://r.sund.ku.dk/` for a general introduction to the R software and recommend RStudio as a user interface. The document focuses on operations that will be used in the practicals.

⚠ copy pasting code from a pdf containing the symbol ∼ will typically lead to an error. Simply re-type the ∼ symbol should solve the (encoding) problem.

There are many ways to perform a given operation in R (e.g. subset a dataset) and for the purpose of the course it generally does not matter how you do. This document mainly refers to core R functions instead of using specialized packages (e.g. tidyverse, data.table, . . .). The one exception is graphical displays where we recommend the ggplot2 package. You are welcome to use other packages or syntax as soon as:

- you are able to do the operations listed in this document in a reasonnable amount of time.
- you have some understanding of what is going on and are able to adapt the instructions to a new problem.

We will need the following packages:

```
library(ggplot2)
library(Epi)
library(survival)
```

and for illustration, we will use the dataset `BrCa` which originates from a study about survival after breast cancer. The dataset contains information about the age and grade of the tumor, survival time after surgery, and outcome (alive or death) at end of follow-up.

```
data(BrCa, package = "Epi")
```

Contents

1 Data management

1.1 Overview

Only display the first lines of the dataset:

```
head(BrCa)
## head(BrCa, 3) ## display less
## head(BrCa, 10) ## display more
```

```
   pid year age meno      size grade nodes   pr    pr.tr  er ...
1 1264 1986  54 post   <=20 mm     2     0 1360 7.215975 149 ...
2 1150 1990  55 post >20-50 mm     2     0  763 6.638568 763 ...
3  838 1988  34  pre   <=20 mm     2     0  113 4.736198 109 ...
4 1214 1990  42 post   <=20 mm     2     0  465 6.144186  79 ...
5 1130 1989  35  pre   <=20 mm     2     0   82 4.418841  25 ...
6 1118 1987  50 post   <=20 mm     2     0   75 4.330733  10 ...
```

For concisness only the first 10 columns were shown (17 columns should be displayed in your R console). To only display the last lines of the dataset:

```
tail(BrCa) ## can also use tail(BrCa, 3) or tail(BrCa, 10)
```

```
      pid year age meno      size grade nodes  pr    pr.tr  er ...
2977 2587 1990  42  pre >20-50 mm     3     4   7 2.079442   1 ...
2978 1832 1993  52  pre    >50 mm     3    15  20 3.044522  52 ...
2979 2362 1987  49  pre    >50 mm     3    15 103 4.644391  85 ...
2980 1907 1986  66 post    >50 mm     3    10 153 5.036953 183 ...
2981 1755 1989  47  pre    >50 mm     3    15 109 4.700480  42 ...
2982 1482 1987  79 post >20-50 mm     3    15  12 2.564949   6 ...
```

Dimensions (number of rows and columns)

```
dim(BrCa)
```

```
[1] 2982   17
```

Extract column names:

```
names(BrCa)
```

```
 [1] "pid"    "year"   "age"    "meno"   "size"   "grade"  "nodes"  "pr"
 [9] "pr.tr"  "er"     "hormon" "chemo"  "tor"    "tom"    "tod"    "tox"
[17] "xst"
```

Extract row names:

```
[1] "1"   "2"   "3"   "4"   "5"   "6"   "..."
```

Type of variables stored in each column with examples:

```
str(BrCa) ## summary(BrCa) is an alternative
```

```
’data.frame’:        2982 obs. of  17 variables:
 $ pid   : int  1264 1150 838 1214 1130 1118 386 1417 927 489 ...
 $ year  : int  1986 1990 1988 1990 1989 1987 1989 1993 1984 1989 ...
 $ age   : int  54 55 34 42 35 50 46 40 36 42 ...
 $ meno  : Factor w/ 2 levels "pre","post": 2 2 1 2 1 2 2 1 1 1 ...
 $ size  : Factor w/ 3 levels "<=20 mm",">20-50 mm",..: 1 2 1 1 1 1 1 1 1 1 ...
 $ grade : Factor w/ 2 levels "2","3": 1 1 1 1 1 1 1 1 1 1 ...
 $ nodes : int  0 0 0 0 0 0 0 0 0 0 ...
 $ pr    : int  1360 763 113 465 82 75 174 0 43 462 ...
 $ pr.tr : num  7.22 6.64 4.74 6.14 4.42 ...
 $ er    : int  149 763 109 79 25 10 56 2 23 75 ...
 $ hormon: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ chemo : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ tor   : num  NA NA NA NA NA ...
 $ tom   : num  NA NA NA NA NA NA NA NA NA NA ...
 $ tod   : num  NA NA NA NA NA ...
 $ tox   : num  12.97 8.78 9.41 10.47 10.35 ...
 $ xst   : Factor w/ 2 levels "Alive","Dead": 1 1 1 1 1 2 1 1 1 1 ...
```

Type of object

```
class(BrCa)
```

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

1.2 Operation on columns

Subset by column when knowing the column names:

```
BrCaR <- BrCa[,c("pid","age","grade","tox","xst")]
BrCaR
```

```
     pid age grade         tox   xst
1   1264  54     2 12.97193654 Alive
2   1150  55     2  8.78302511 Alive
3    838  34     2  9.41273117 Alive
4   1214  42     2 10.47227923 Alive
5   1130  35     2 10.35181363 Alive
6   1118  50     2 10.91854858  Dead
...  ... ...   ...         ...   ...
```

When extracting a single column, R will simplify the data format and return a vector instead of a data.frame:

```
BrCa[,"age"]
```

```
[1] 54  55  34  42  35  50  ...
```

To keep the data format regardless to the number of columns request, one should add the argument `drop=FALSE`:

```
BrCa[,"age",drop=FALSE]
```

```
    age
1    54
2    55
3    34
4    42
5    35
6    50
... ...
```

Renaming columns

```
names(BrCaR) <- c("id","age","grade","time","status")
BrCaR
```

```
      id age grade        time status
1   1264  54     2 12.97193654  Alive
2   1150  55     2  8.78302511  Alive
3    838  34     2  9.41273117  Alive
4   1214  42     2 10.47227923  Alive
5   1130  35     2 10.35181363  Alive
6   1118  50     2 10.91854858   Dead
...  ... ...   ...         ...    ...
```

It is also possible use the position of the columns to subset a data.frame, e.g.:

⚠ this is however more error prone, make the code harder to read, and is thus not recommended

```
head(BrCa[,1:3]) ## same as BrCa[,c(1,2,3)]
```

```
     pid year age
1   1264 1986  54
2   1150 1990  55
3    838 1988  34
4   1214 1990  42
5   1130 1989  35
6   1118 1987  50
...  ...  ... ...
```

1.3 Operation on rows

1.3.1 Basic subset

Re-order the dataset by increasing column values:

```
BrCaR[order(BrCaR$time),]
```

```
       id age grade       time status
1905  407  54     2 0.09856263  Alive
2945 3004  75     3 0.12320329   Dead
2334 2962  66     3 0.17522246   Dead
2949 2956  87     3 0.20260096   Dead
2815 2979  75     3 0.26557153   Dead
1956  537  58     3 0.27652293  Alive
...   ... ...   ...        ...    ...
```

Re-order the dataset by decreasing column values:

```
BrCaR[order(BrCaR$time, decreasing  = TRUE),]
```

```
       id age grade        time status
292   767  49     2 19.28268305  Alive
1822   19  51     3 19.23887761  Alive
1134 1386  45     3 18.85284042  Alive
204   576  37     3 18.42299779  Alive
39   2720  59     2 17.60438029  Alive
326  2776  70     2 17.34428533  Alive
...   ... ...   ...         ...    ...
```

Select row(s) corresponding to a specific column value:

```
BrCaR[BrCaR$id == 1150,]
```

```
    id age grade     time status
2 1150   55     2 8.783025  Alive
```

```
BrCaR[BrCaR$status == "Alive",]
```

```
      id age grade         time status
1    1264  54     2 12.97193654  Alive
2    1150  55     2  8.78302511  Alive
3     838  34     2  9.41273117  Alive
4    1214  42     2 10.47227923  Alive
5    1130  35     2 10.35181363  Alive
7     386  46     2 10.20396996  Alive
...  ... ...   ...         ...    ...
```

Select rows corresponding to not have a specific column value:

```
BrCaR[BrCaR$status != "Alive",]
```

```
      id age grade         time status
6   1118  50      2 10.91854858   Dead
42  2765  26      2  4.20807679   Dead
45  2544  51      3 10.11362076   Dead
65  2037  50      2  8.07118416   Dead
70  1233  69      2 10.51882299   Dead
86  1816  54      2 11.09103394   Dead
...  ... ...   ...         ...    ...
```

Select rows corresponding to multiple column values:

```
BrCaR[BrCaR$id %in% c(100,101,150),]
```

```
      id age grade       time status
709  101  47      2  8.473648  Alive
1457 100  44      3 15.307323  Alive
1551 150  47      3  4.539357  Alive
```

```
BrCaR[BrCaR$id %in% c(-1,100,101,150),] ## -1 was not found
```

```
      id age grade       time status
709  101  47      2  8.473648  Alive
1457 100  44      3 15.307323  Alive
1551 150  47      3  4.539357  Alive
```

Select rows whose column values differs from a set of values:

```
BrCaR[BrCaR$grade %in% 1:2 == FALSE,]
```

```
      id age grade         time status
11    481  54      3  6.84462674  Alive
12    477  29      3 10.08624204  Alive
14   1320  42      3 10.00136884  Alive
15     24  52      3 12.77207438  Alive
27    224  46      3 10.21492132  Alive
28    522  64      3  6.89390818  Alive
...  ... ...   ...         ...    ...
```

1.3.2 Complex subset

When the subset of rows to take follows a complex criteria, it can be easier to do it in several step. Consider extracting 5 patients of for each combination of grade 2/3 and alive/dead. First compute the index of line corresponding to each combination separately:

```
BrCaR.grad2Dead <- BrCaR[BrCaR$grade == 2 & BrCaR$status == "Dead",]
BrCaR.grad3Dead <- BrCaR[BrCaR$grade == 3 & BrCaR$status == "Dead",]
BrCaR.grad2Alive <- BrCaR[BrCaR$grade == 2 & BrCaR$status == "Alive",]
BrCaR.grad3Alive <- BrCaR[BrCaR$grade == 3 & BrCaR$status == "Alive",]
```

The `&` symbol extract the row when both criteria are true whereas the `|` would extract if any of the two criteria are true:

```
c("&" = TRUE & FALSE, "|" = TRUE | FALSE)
```

```
    &     |
FALSE  TRUE
```

The subset is obtain by combining the first five lines of each combination-specific dataset:

```
BrCaR.subset <- rbind(BrCaR.grad2Dead[1:5,],
                      BrCaR.grad3Dead[1:5,],
                      BrCaR.grad2Alive[1:5,],
                      BrCaR.grad3Alive[1:5,])
```

More concise syntax can be obtain with dedicated functions, often at the cost of readability, e.g.:

```
BrCaR.subset2 <- do.call(rbind, by(BrCaR,
                    INDICES = interaction(BrCaR[,c("grade","status")]),
                    FUN = head, n = 5))
```

This is the same dataset as the previous one up to formating and re-ordering:

```
all(sort(BrCaR.subset2$id)==sort(BrCaR.subset$id))
```

```
[1] TRUE
```

1.4 Recasting

Convert numerical values to factor:

```
factor(c(1,2,3,2,3), levels = 1:3, labels = c("g1","g2","g3"))
```

```
[1] g1 g2 g3 g2 g3
Levels: g1 g2 g3
```

Convert categorical value to indicator function:

```
as.numeric(BrCaR$status=="Dead")
```

```
[1] 0   0   0   0   0   1   ...
```

1.5 Summary statistics on subgroups

Either do it ’manually’ by subsetting the dataset and evaluating the statistic:

```
mean(BrCaR[BrCaR$grad==2,"age"]) ## mean age of grade 2 patients
mean(BrCaR[BrCaR$grad==2,"status"]=="Dead") ## proportion of deaths
```

```
[1] 54.38161
[1] 0.3299748
```

Otherwise use dedicated functions, e.g.:

```
tapply(BrCaR$age, INDEX = BrCaR$grade, FUN = mean)
```

```
       2        3
54.38161 55.30393
```

or for computing the same statistic on multiple columns:

```
aggregate(cbind(death=status=="Dead",age) ~ grade, data = BrCaR,
          FUN = mean)
```

```
  grade     death      age
1     2 0.3299748 54.38161
2     3 0.4616088 55.30393
```

Here the syntax `status=="Dead"` indicates how to convert the categorical variable status to numeric (`"Dead"` is 1, `"Alive"` is 0). The surprising syntax `death=status=="Dead"` is to give a name to results and `cbind` is to combine the two statistics (death and age) into two different columns.

It is also possible use a function to obtain a specific statistic, e.g. the smallest and largest event time:

```
aggregate(time ~ grade, data = BrCaR,
          FUN = function(t){c(min = min(t), max = max(t))})
```

```
  grade   time.min    time.max
1     2 0.09856263 19.28268305
2     3 0.12320329 19.23887761
```

Here we define a function which transform its input, the timepoints of individuals with a specific grade (arbitrarily named `t`), into a vector with two elements: the first being the smallest value the input and the second the largest value.

1.6 Creating objects

1.6.1 Vector

Different numbers can be combined into a vector using `c`:

```
vec <- c(1,2,5,10)
vec
```

```
[1]  1  2  5 10
```

A convenient shortcut for consecutive integers is:

```
vec2 <- 1:5
vec2
```

```
[1] 1 2 3 4 5
```

Vectors can also be combined into a longer vector:

```
c(vec,vec2)
```

```
[1]  1  2  5 10  1  2  3  4  5
```

⚠ Any mixture of numeric and character values will recast automatically the vector into a character

```
c("a",1:5)
```

```
[1] "a" "1" "2" "3" "4" "5"
```

1.6.2 Matrices

Vectors of same size can be combined into a matrix using `rbind` or `cbind`:

```
vec2 <- c(2,5,10,2)
rbind(vec,vec2) ## row-binding
```

```
     [,1] [,2] [,3] [,4]
vec     1    2    5   10
vec2    2    5   10    2
```

```
M <- cbind(vec,vec2, newcolumn = 1) ## column-binding
M
```

```
     vec vec2 newcolumn
[1,]   1    2         1
[2,]   2    5         1
[3,]   5   10         1
[4,]  10    2         1
```

A single value will be duplicated to match the dimension of the rest of the vectors.

```
class(M)
```

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

⚠ Any mixture of numeric and character values will recast automatically the matrix into a character

1.6.3 Data frame

Data frames are useful to combine information of different types: typically parameter names (i.e. character) with their value (numeric). Consider the following simplistic example where we estimate the mean baseline age depending on the severity of the disease:

```
e.lm <- lm(age ~ 0 + grade, data = BrCaR)
```

```
  grade2   grade3 
54.38161 55.30393 
```

We can extract the coefficient values and names using:

```
coef(e.lm)
```

```
  grade2   grade3
54.38161 55.30393
```

To store it in a data.frame format we can do

```
df.lm <- data.frame(names(coef(e.lm)), coef(e.lm))
df.lm
```

```
       names.coef.e.lm.. coef.e.lm.
grade2            grade2   54.38161
grade3            grade3   55.30393
```

And add confidence intervals:

```
df.lm <- cbind(df.lm, confint(e.lm))
df.lm
```

```
       names.coef.e.lm.. coef.e.lm.    2.5 %   97.5 %
grade2            grade2   54.38161 53.48058 55.28265
grade3            grade3   55.30393 54.76114 55.84672
```

1.7 Updating names, row names, column names

The `names` method can be use to name elements of a vector and columns of a data.frame, e.g.:

```
names(vec) <- c("baby","child","adult","senior")
vec
```

```
baby  child  adult senior
   1      2      5     10
```

```
names(df.lm) <- c("name","estimate","lower","upper")
df.lm
```

```
         name estimate    lower    upper
grade2 grade2 54.38161 53.48058 55.28265
grade3 grade3 55.30393 54.76114 55.84672
```

If is possible to name 'on the fly' a vector using `setNames`:

```
setNames(c(1,2,5,10), c("baby","child","adult","senior"))
```

```
baby  child  adult senior
   1      2      5     10
```

To remove names use:

```
unname(vec) ## vector
```

```
[1]  1  2  5 10
```

```
rownames(df.lm) <- NULL ## data.frame
```

For matrices one should use `colnames` instead of `names` to rename the columns

```
colnames(M) <- c("a","b","c")
M
```

```
      a  b c
[1,]  1  2 1
[2,]  2  5 1
[3,]  5 10 1
[4,] 10  2 1
```

2 Data visualization

2.1 Individual trajectories

A display of the individual trajectories (here on a subset of the data) can be obtained first displaying the time at risk using a segment and then points to indicate the type of event:

```
ggTraj <- ggplot(BrCaR.subset)
ggTraj <- ggTraj + geom_segment(aes(x = age, xend = age + time,
                                    y = id, yend = id, color = grade))
ggTraj <- ggTraj + geom_point(aes(x = age + time, y = id, shape = status),
                              size = 2)
ggTraj ## see left panel next page for a graphical display
```

2.2 Forest plot

Results from different analyses can be displayed in a single graph using a forest plot. For instance we can dsiplay the mean baseline age previously computed:

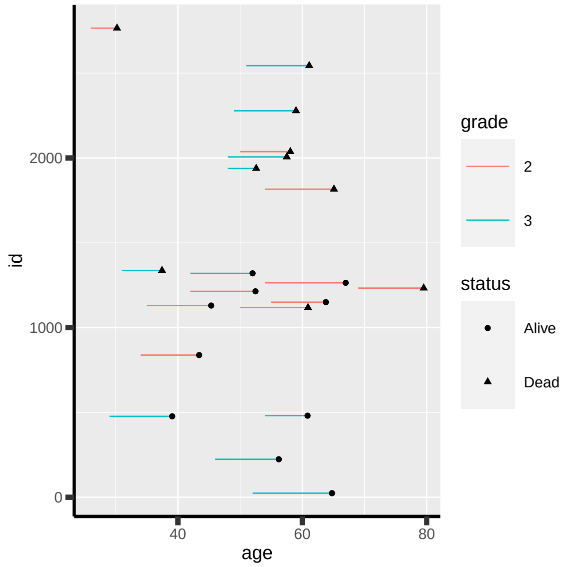
```
df.lm
```

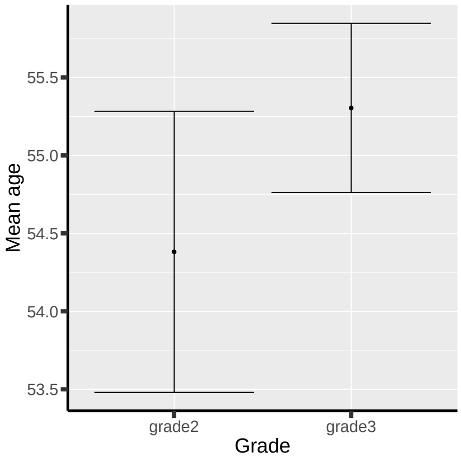
```
    name estimate    lower    upper
1 grade2 54.38161 53.48058 55.28265
2 grade3 55.30393 54.76114 55.84672
```

per severity group using:

```
ggForest <- ggplot(df.lm, aes(x = name, y = estimate,
                              ymin = lower, ymax = upper))
ggForest <- ggForest + geom_point() + geom_errorbar()
ggForest <- ggForest + labs(y = "Mean age", x = "Grade")
ggForest ## see right panel next page for a graphical display
```

⚠ one should make sure that the results are indeed comparable. This is typically not the case when comparing regression coefficient from non-linear models based on different covariate sets.


id
2000
1000
0
40
60
80
age
grade
2
3
status
Alive
Dead


Mean age
55.5
55.0
54.5
54.0
53.5
grade2
grade3
Grade

3 Contingency tables

3.1 Categorical variables

The function `table` creates a 2 by 2 table (or more generally p by q tables), counting the number of occurences of the possible combinations between two variables. The function `rowSums` and `colSums` can be used to obtain, respectively, the total by row and by column:

```
t22 <- table(BrCaR$grade,
             outcome = BrCaR$status)
t22
```

```
  outcome
   Alive Dead
2    532  262
3   1178 1010
```

```
rowSums(t22)
```

```
  2    3
794 2188
```

```
colSums(t22)
```

```
Alive  Dead
 1710  1272
```

As in a data.frame one can use row and column names to extract objects:

```
t22["2","Alive"]
```

```
[1] 532
```

The function `addmargin` is useful to add the total per row or column:

```
addmargins(t22, margin = 2)
```

```
 outcome
  Alive Dead  Sum
2   532  262  794
3  1178 1010 2188
```

Similarly the function `prop.table` can be use to evaluate the proportion of events:

```
prop.table(t22, margin = 1)
```

```
 outcome
      Alive      Dead
2 0.6700252 0.3299748
3 0.5383912 0.4616088
```

3.2 Continuous variables

`xtabs` can be used to sum values of variables per group:

```
BrCaR$status.bin <- BrCaR$status=="Dead"
t23.end <- xtabs(cbind(n=1, death = status.bin, person.year=time) ~ grade,
                 data = BrCaR)
t23.end
```

```
grade         n     death person.year
    2   794.000   262.000    6323.439
    3  2188.000  1010.000   14947.300
```

This can be thought as a shortcut to:

(i) split the dataset per group,

(ii) add a column of 1 and select the columns `status.bin` and `time`

(iii) sum the values within each column:

```
BrCaR.2 <- BrCaR[BrCaR$grade == "2",] ## (i)
data.tempo <- cbind(n=1, death = BrCaR.2$status.bin,
                    person.year=BrCaR.2$time) ## (ii)
colSums(data.tempo) ## (iii)
```

```
      n       death person.year
794.000     262.000    6323.439
```

To restrict to 10 years follow-up, we should only count deaths happening within the first 10 years and limit to 10 the number of person.year for a given person:

```
t23.10 <- xtabs(cbind(n=1,
                      death = (time<=10)*status.bin,
                      person.year=pmin(time,10)) ~ grade,
                data = BrCaR)
t23.10
```

```
grade         n     death person.year
    2   794.000   231.000    5852.509
    3  2188.000   940.000   14149.946
```

3.3 3-dimensional tables

Here is an example of 3-dimensional table (exposure, outcome, covariate):

```
Titanic.adult <- aperm(Titanic[,,"Adult",],c(2,3,1))
Titanic.adult
```

```
   , , Class = 1st                       , , Class = 3rd

           Survived                              Survived
   Sex       No Yes                      Sex       No Yes
     Male   118  57                        Male   387  75
     Female   4 140                        Female  89  76

   , , Class = 2nd                       , , Class = Crew

           Survived                              Survived
   Sex       No Yes                      Sex       No Yes
     Male   154  14                        Male   670 192
     Female  13  80                        Female   3  20
```

The `ftable` function is convenient to obtain a condensed representation:

```
ftable(Titanic.adult)
```

```
                Class 1st 2nd 3rd Crew
Sex    Survived                       
Male   No             118 154 387  670
       Yes             57  14  75  192
Female No               4  13  89    3
       Yes            140  80  76   20
```

One can subset, e.g. the crew, using the usual bracket with one more comma for the 3rd dimension:

```
Titanic.adult[,,"Crew"] ## back to 2 by 2 table
```

```
        Survived
Sex       No Yes
  Male   670 192
  Female   3  20
```