APPENDIX 1

The most important R objects and basic operations with them

If you need to know which class of R objects `x` belongs to, you can use the function `is(x)`. Alternatively, you can type `class(x)`, which will return only one value:

```
> x <- 1:3
> x
[1] 1 2 3
> is(x)
[1] "integer"       "numeric"       "vector"
[4] "data.frameRowLabels"
```

1 VECTORS

A vector is a sequence of numeric values, characters or logical (Boolean) values TRUE or FALSE.

1.1 Numeric vectors

– How to create a vector:

```
> v <- c(1:5, 10, 5) # creates a vector with all numbers from 1
to 5 inclusively, 10 and 5.
> v
[1] 1 2 3 4 5 10 5
> v1 <- seq(0, 100, 10) # creates a vector with numbers from 0
(inclusively) to 100 at an interval of 10
> v1
[1] 0 10 20 30 40 50 60 70 80 90 100
> v2 <- rep(1:5, 3) # creates a vector with the sequence of num-
bers from 1 to 5 repeated three times
> v2
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

– How to see the summary statistics of a vector:

```
> summary(v) # returns the minimum, 1st quartile, median, mean,
3rd quartile and maximum
Min. 1st   Qu.    Median   Mean    3rd Qu.   Max.
1.000      2.500  4.000    4.286   5.000     10.000
```

– How to sort a vector:

```
> sort(v) # sorts in increasing order
[1] 1 2 3 4 5 5 10
> sort(v, decreasing = TRUE) # sorts in decreasing order
[1] 10 5 5 4 3 2 1
```

– How to check how many elements a vector consists of:

```
> length(v)
[1] 7
```

– How to select elements of a vector:

```
> v[c(3, 6)] # returns only the 3rd and the 6th elements
[1] 3 10
> v[-1] # returns all elements except the first one
[1] 2 3 4 5 10 5
> v[v > 2] # returns the elements of vector v that are greater
than 2
[1] 3 4 5 10 5
> v[v != 5] # returns the elements of vector v that are not equal
to 5
[1] 1 2 3 4 10
> v[v < 3|v==10] # returns the elements of vector v that are
smaller than 3 OR are equal to 10
[1] 1 2 10
> v[v > 4 & v < 10] # returns the elements of vector v that are
greater than 4 AND smaller than 10
[1] 5 5
```

– How to obtain the indices of vector elements that meet some conditions:

```
> which(v == 5) # returns the indices of the elements of vector
v that are equal to 5
[1] 5 7
```

– How to sum all values of a vector:

```
> sum(v)
[1] 30
```

– How to combine two or more vectors into one:

```
> w <- c(20, 30, 40)
> y <- c(v, w)
> y
[1] 1 2 3 4 5 10 5 20 30 40
```

– How to perform basic arithmetic operations on all elements of a vector:

```
> v + 3 # adds 3 to every element of vector v
[1] 4 5 6 7 8 13 8
> v *2 # multiply every element of vector v by 2
[1] 2 4 6 8 10 20 10
> v^2 # raises every element of vector v to the power of 2
[1] 1 4 9 16 25 100 25
> log(v) # returns the natural logarithm of every element of
vector v
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 2.3025851
1.6094379
```

– How to round the elements of a vector:

```
> round(log(v), 2) # rounds the logarithmic values of vector v,
displaying only two digits after the decimal separator
[1] 0.00 0.69 1.10 1.39 1.61 2.30 1.61
```

1.2 Character vectors

A character vector is a special type of vectors that contain characters and/or character strings.

– How to create a character vector:

```
> charv <- c("do", "re", "mi", "do")
> charv
[1] "do" "re" "mi" "do"
```

– How to obtain the indices of the elements of a character vector that have a specific value:

```
> which(charv == "do")
[1] 1 4
> grep("d", charv) # returns the indices of all elements of
character vector charv that contain 'd'
[1] 1 4
```

– How to replace characters in a character vector:

```
> gsub("m", "t", charv) # replace every 'm' in the elements of
the character vector charv with 't'
[1] "do" "re" "ti" "do"
> paste(charv, "!", sep = "") # add '! ' to every element of
character vector charv, without any graphical separators (the
default separator is a space)
[1] "do!" "re!" "mi!" "do!"
```

– How to turn a character vector into a factor:

```
> f <- as.factor(charv)
> f
[1] do re mi do
Levels: do mi re
```

1.3 Logical vectors

– How to create a logical vector:

```
> boolv <- c(TRUE, FALSE, FALSE, TRUE)
> boolv
[1] TRUE FALSE FALSE TRUE
```

– How to obtain the indices of the elements of a character vector that have a specific value (TRUE or FALSE):

```
> which(boolv == TRUE)
[1] 1 4
```

2 FACTORS

A factor is a sequence of values of a categorical variable. The values are called levels.

– How to create a factor:

```
> f <- factor(c("do", "re", "mi", "do"))
> f
[1] do re mi do
Levels: do mi re
```

See also how to transform a character vector into a factor.

– How to see the levels of a factor:

```
> levels(f)
[1] "do" "mi" "re"
```

– How to change the order of the levels (the default order is alphabetic):

```
> f1 <- relevel(f, ref = "mi") # make mi the first (reference)
level
> levels(f1)
[1] "mi" "do" "re"
```

– How to see how many times each level of a factor occurs:

```
> table(f)
f
do  mi  re
2   1   1
```

Alternatively:

```
> summary(f)
do  mi  re
2   1   1
```

– How to turn a factor into a numeric vector:

```
> v3 <- as.numeric(f)
> v3
[1] 1 3 2 1 # the numbers correspond to the order of factor
levels
```

– How to turn a factor into a character vector:

```
> charv1 <- as.character(f)
> charv1
[1] "do" "re" "mi" "do"
```

– How to make an ordered factor:

```
> ord <- c("XS", "S", "M", "L", "XL")
> ord
[1] "XS" "S" "M" "L" "XL"
> ord <- ordered(ord, levels = c("XS", "S", "M", "L", "XL"))
> ord
[1] XS S M L XL
Levels: XS < S < M < L < XL
```

– How to cross-tabulate two factors:

```
> gender <- factor(c("M", "F", "M", "F", "M", "F"))
> gender
[1] M F M F M F
Levels: F M
```

```
> group <- factor(c("A", "A", "B", "A", "B", "B"))
> group
[1] A A B A B B
Levels: A B

> table(group, gender)
gender
group   F   M
A       2   1
B       1   2
```

Alternatively:

```
> xtabs(~ group + gender)
gender
group   F   M
A       2   1
B       1   2
```

3 MATRICES

A matrix is a two-dimensional array. Arrays can have any number of dimensions. Matrices can contain numbers, characters or Boolean values.

– How to create a matrix from scratch:

```
> m <- matrix(c(2, 7, 5, 3, 8, 11, 2, 4, 5), nrow = 3)
> m
      [,1]  [,2]   [,3]
[1,]  2     3      2
[2,]  7     8      4
[3,]  5     11     5
```

– How to learn the number of rows and columns in a matrix:

```
> dim(m)
[1] 3 3
```

– How to select columns, rows and elements:

```
> m[, 1] # selects column 1
[1] 2 7 5
> m[2, ] # selects row 2
[1] 7 8 4
> m[3, 3] # selects the element in row 3 and column 3
[1] 5
```

```
> m[c(1, 3),] # selects rows 1 and 3
       [,1]   [,2]   [,3]
[1,]   2      3      2
[2,]   5      11     5
> m[, -2] # selects all columns except the second one
       [,1]   [,2]
[1,]   2      2
[2,]   7      4
[3,]   5      5
```

– How to add the names of rows and columns:

```
> rownames(m) <- c("A", "B", "C")
> colnames(m) <- c("X", "Y", "Z")
> m
   X   Y   Z
A  2   3   2
B  7   8   4
C  5   11  5
```

– How to obtain a vector with row or column sums:

```
# add na.rm = TRUE if the matrix contains missing values
> rowSums(m)
A  B   C
7  19  21
> colSums(m)
X    Y   Z
14   22  11
```

– How to obtain a vector with row or column means:

```
> rowMeans(m)
A         B         C
2.333333  6.333333  7.000000
> colMeans(m)
X         Y         Z
4.666667  7.333333  3.666667
```

– How to apply a function to rows or columns:

```
> apply(m, 1, sd) # applies the function sd() to every row
A       B        C
0.5773503        2.0816660    3.4641016
```

```
> apply(m, 2, var) # applies the function var() to every column
X          Y          Z
6.333333   16.333333   2.333333
```

– How to transpose a matrix, i.e. swap the rows and columns:

```
> t(m)
   A  B  C
X  2  7  5
Y  3  8  11
Z  2  4  5
```

– How to turn counts in a table into proportions:

```
> prop.table(m) # the sum of all cells is 1
   X            Y            Z
A  0.04255319   0.06382979   0.04255319
B  0.14893617   0.17021277   0.08510638
C  0.10638298   0.23404255   0.10638298
> prop.table(m, 1) # the sum of each row is 1
X  Y            Z
A  0.2857143    0.4285714    0.2857143
B  0.3684211    0.4210526    0.2105263
C  0.2380952    0.5238095    0.2380952
> prop.table(m, 2) # the sum of each column is 1
   X            Y            Z
A  0.1428571    0.1363636    0.1818182
B  0.5000000    0.3636364    0.3636364
C  0.3571429    0.5000000    0.4545455
```

4 Data frames

A data frame is an object that contains any kind of variable: numeric, character and logic vectors, as well as ordered and unordered factors. The rows are usually observations, and the columns represent variables.

– How to create a data frame:

```
> group <- factor(c("A", "A", "B", "A", "B", "B")) # one can also
create a simple character vector. The latter will be transformed
into a factor automatically by data.frame().
> gender <- factor(c("M", "F", "M", "F", "M", "F"))
> grade <- c(2.3, 1.7, 5, 3, 1.3, 5)
```

```
> df <- data.frame(group, gender, grade)
> df
   group  gender  grade
1  A      M       2.3
2  A      F       1.7
3  B      M       5.0
4  A      F       3.0
5  B      M       1.3
6  B      F       5.0
```

– How to see the structure of a data frame:

```
> str(df)
'data.frame':    6 obs. of 3 variables:
$ group: Factor w/ 2 levels "A","B": 1 1 2 1 2 2
$ gender: Factor w/ 2 levels "F","M": 2 1 2 1 2 1
$ grade: num 2.3 1.7 5 3 1.3 5
> summary(df)
group   gender   grade
A:3     F:3      Min.    :1.30
B:3     M:3      1st Qu. :1.85
                 Median  :2.65
                 Mean    :3.05
                 3rd Qu. :4.50
                 Max.    :5.00
```

– How to add new variables:

```
> df$pass <- c(TRUE, TRUE, FALSE, TRUE, TRUE, FALSE)
> df
   group   gender   grade   pass
1  A       M        2.3     TRUE
2  A       F        1.7     TRUE
3  B       M        5.0     FALSE
4  A       F        3.0     TRUE
5  B       M        1.3     TRUE
6  B       F        5.0     FALSE
```

– How to select columns and/or rows of a data frame:

```
> df$group # selects the column group only
[1] A A B A B B
Levels: A B
> df[, 2:3] # selects columns 2 and 3
```

```
   gender  grade
1  M       2.3
2  F       1.7
3  M       5.0
4  F       3.0
5  M       1.3
6  F       5.0
> df[c("grade","pass")] # selects columns grade and pass by
their names
   grade   pass
1  2.3     TRUE
2  1.7     TRUE
3  5.0     FALSE
4  3.0     TRUE
5  1.3     TRUE
6  5.0     FALSE
> df[c(2, 4, 6), ] # selects observations (i.e. rows) 2, 4 and 6
   group   gender   grade   pass
2  A       F        1.7     TRUE
4  A       F        3.0     TRUE
6  B       F        5.0     FALSE
> df[2:5, -1] # selects observations from 2 to 5 and all columns
except the first one
   gender   grade   pass
2  F        1.7     TRUE
3  M        5.0     FALSE
4  F        3.0     TRUE
5  M        1.3     TRUE
```

– How to select only observations that meet specific criteria:

```
> df[df$gender == "F", ] # returns observations with gender F
   group   gender   grade   pass
2  A       F        1.7     TRUE
4  A       F        3.0     TRUE
6  B       F        5.0     FALSE
> df[df$gender == "F" & df$grade < 5, ] # returns observations
with gender F AND grade smaller than 5
   group   gender   grade   pass
2  A       F        1.7     TRUE
4  A       F        3.0     TRUE
```

```
> df[df$gender == "F" | df$group == "A", ] # returns observa-
tions with gender F OR group A
group      gender   grade   pass
1  A       M        2.3     TRUE
2  A       F        1.7     TRUE
4  A       F        3.0     TRUE
6  B       F        5.0     FALSE
```

– How to obtain the indices (row names) of observations that meet certain criteria:

```
> which(df$gender=="F" & df$grade < 5)
[1] 2 4
```

– How to sort a data frame by values in one or more columns:

```
> df[order(df$grade), ] # sorts the data frame by grade in
ascending order
   group   gender   grade   pass
5  B       M        1.3     TRUE
2  A       F        1.7     TRUE
1  A       M        2.3     TRUE
4  A       F        3.0     TRUE
3  B       M        5.0     FALSE
6  B       F        5.0     FALSE
> df[order(-df$grade), ] # sorts the data frame by grade in
descending order
   group   gender   grade   pass
3  B       M        5.0     FALSE
6  B       F        5.0     FALSE
4  A       F        3.0     TRUE
1  A       M        2.3     TRUE
2  A       F        1.7     TRUE
5  B       M        1.3     TRUE
> df[order(df[,1], -df[,3]), ] # sorts the data frame first by
group in alphabetic order and then by grade in descending order
   group  gender   grade   pass
4  A      F        3.0     TRUE
1  A      M        2.3     TRUE
2  A      F        1.7     TRUE
3  B      M        5.0     FALSE
6  B      F        5.0     FALSE
5  B      M        1.3     TRUE
```

– How to read a data frame from a file:

```
> df <- read.table("C/yourDirectory/yourFile", header = TRUE)
# reads a file with tab-separated columns (e.g. a copy of an
Excel spreadsheet) and treats the first line as column names. By
default, header = FALSE. You can specify the column separator by
adding, for example, sep = "\t" (a tab).
> df <- read.csv("C/yourDirectory/yourFile") # reads a file with
comma-separated columns. The first line in the file is treated as
column names.
```

– How to export a data frame to a file:

```
> write.table(df, file = 'C/yourDirectory/yourFile', sep = "\t",
quote = FALSE) # the separator is a tab (by default, the separa-
tor is a space). The values in character vectors or factors are
not surrounded by double quotes (by default, they are). The row
and column names are saved by default.
```

– How to edit a data frame with the help of an editing interface:
Select *Edit* > *Data editor* in the main menu of the R GUI and enter the name of the data frame that you want to edit. A spreadsheet will open, where you can edit the data.

Alternatively:

```
> fix(df)
```

After you have finished editing, simply close the window with the spreadsheet. The changes will be saved.