UiO : Department of Biosciences
University of Oslo

MBV4410/9410 Fall 2016

Dec. 5 – Introduction to R

Outline

Monday

Before lunch:

- Transcriptomics (lectures)

After lunch:

- Basic R/RStudio (lecture)
- Installing/setting up R/RStudio
- Basic R (practical)

Tuesday

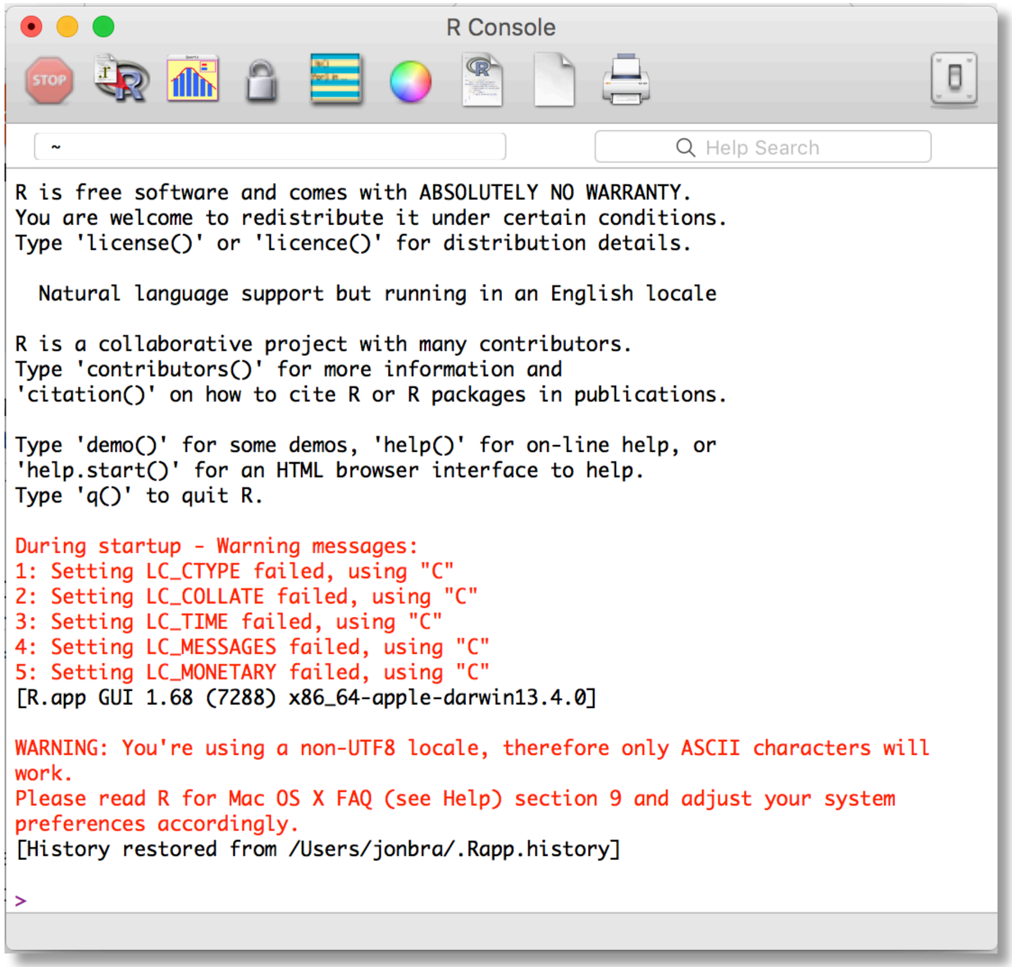
Before lunch:

- Transcriptomics (lectures)

After lunch:

- Bioconductor (lecture)
- Transcriptomics/DE-test (practical)

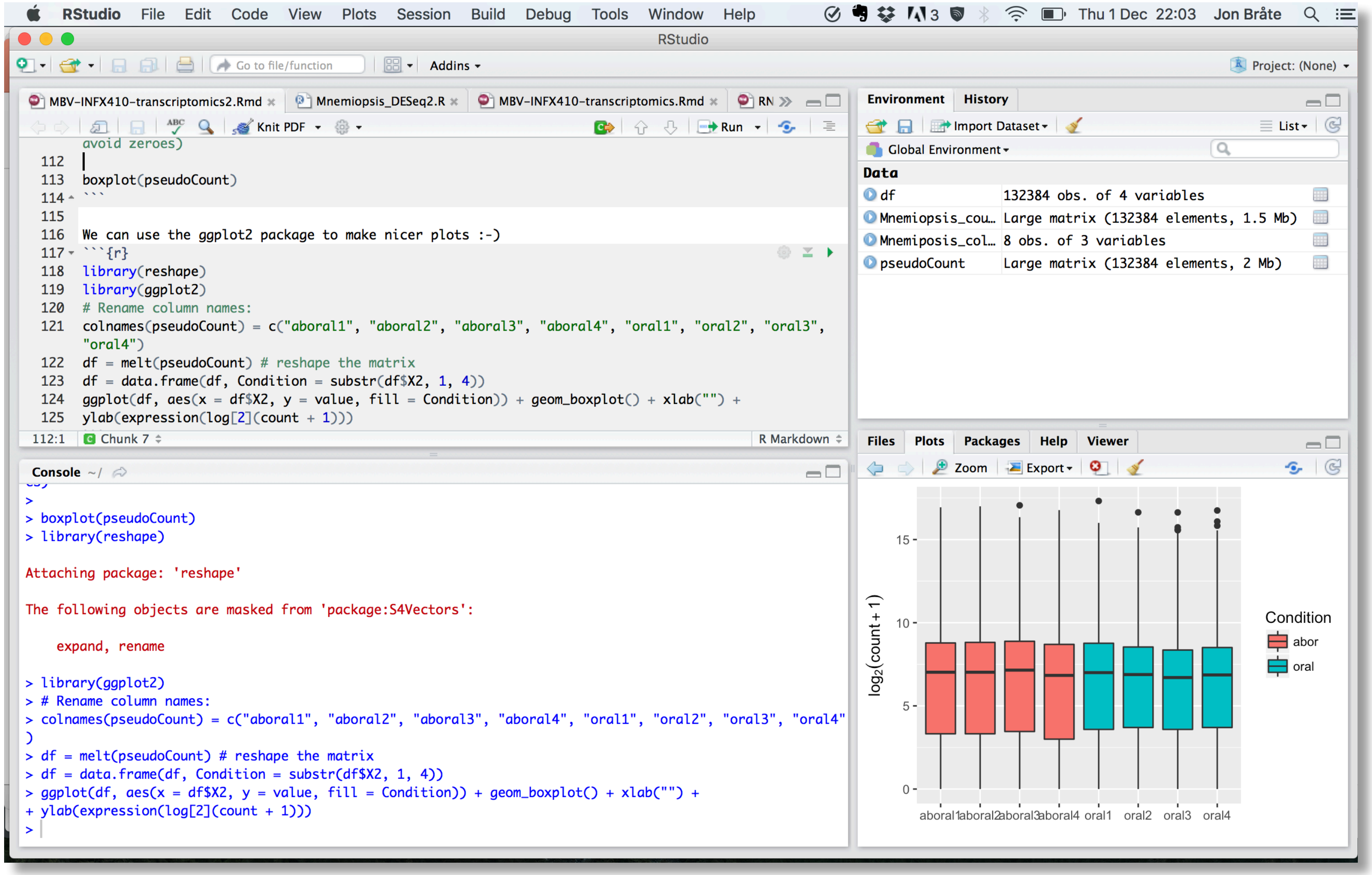
R



```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
[R.app GUI 1.68 (7288) x86_64-apple-darwin13.4.0]

WARNING: You're using a non-UTF8 locale, therefore only ASCII characters will
work.
Please read R for Mac OS X FAQ (see Help) section 9 and adjust your system
preferences accordingly.
[History restored from /Users/jonbra/.Rapp.history]

>
```

UiO : **Department of Biosciences**
University of Oslo

RStudio



```
avoid zeroes)

boxplot(pseudoCount)
```

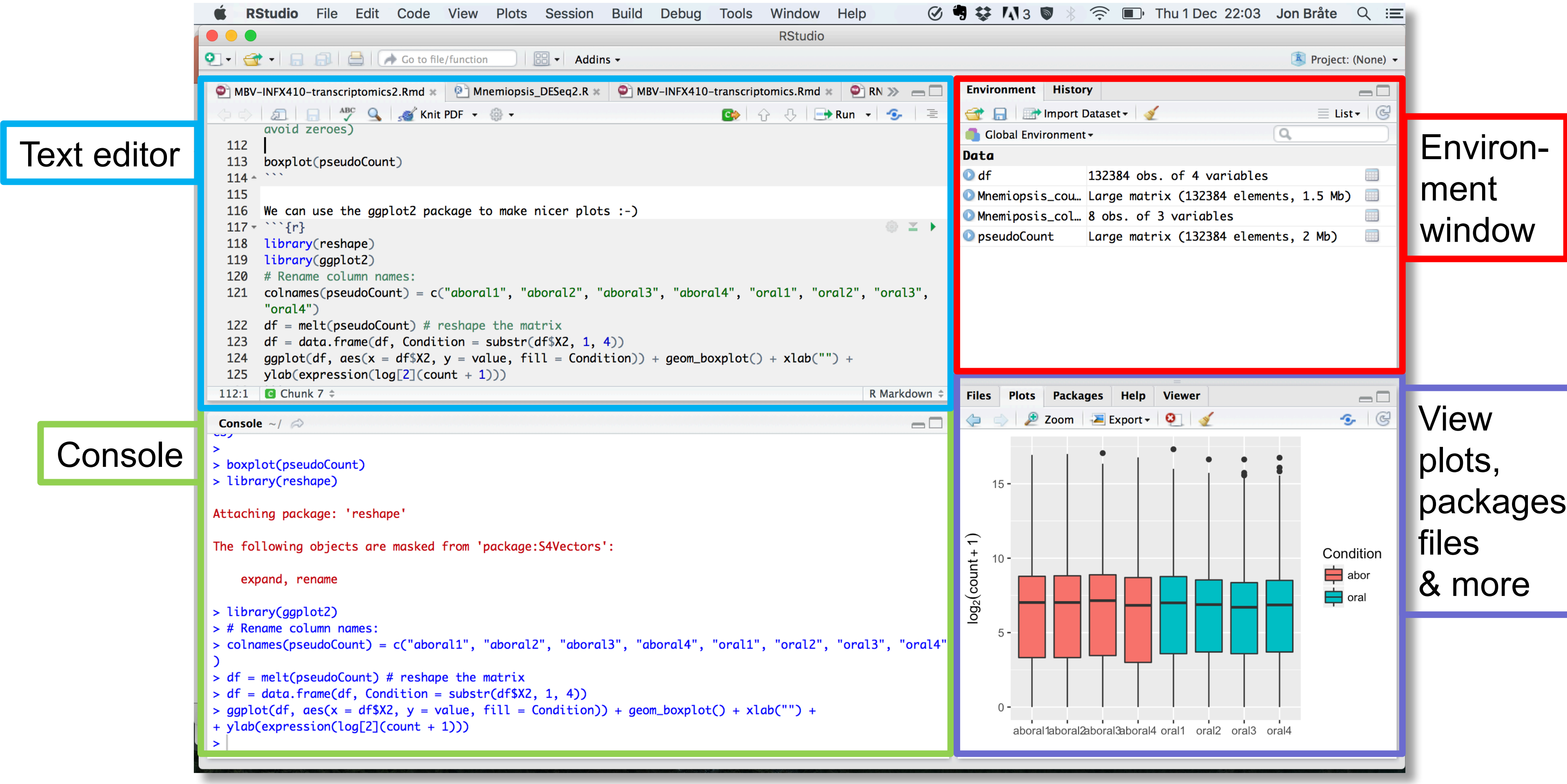
We can use the ggplot2 package to make nicer plots :-)
```{r}
library(reshape)
library(ggplot2)
# Rename column names:
colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3",
"oral4")
df = melt(pseudoCount) # reshape the matrix
df = data.frame(df, Condition = substr(df$X2, 1, 4))
ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
ylab(expression(log[2](count + 1)))
```

Data

df	132384 obs. of 4 variables
Mnemiopsis_cou…	Large matrix (132384 elements, 1.5 Mb)
Mnemiposis_col…	8 obs. of 3 variables
pseudoCount	Large matrix (132384 elements, 2 Mb)

```
>
> boxplot(pseudoCount)
> library(reshape)

Attaching package: 'reshape'

The following objects are masked from 'package:S4Vectors':

    expand, rename

> library(ggplot2)
> # Rename column names:
> colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3", "oral4"
)
> df = melt(pseudoCount) # reshape the matrix
> df = data.frame(df, Condition = substr(df$X2, 1, 4))
> ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
+ ylab(expression(log[2](count + 1)))
>
```

RStudio


Text editor
Console
Environ-ment window
View plots, packages files & more
RStudio File Edit Code View Plots Session Build Debug Tools Window Help
Thu 1 Dec 22:03 Jon Bråte
Go to file/function
Addins
Project: (None)
MBV-INFX410-transcriptomics2.Rmd
Mnemiopsis_DESeq2.R
MBV-INFX410-transcriptomics.Rmd
Knit PDF
Run
Environment
History
Import Dataset
List
Global Environment
Files
Plots
Packages
Help
Viewer
Zoom
Export
Condition
abor
oral
log2(count + 1)
aboral1 aboral2 aboral3 aboral4 oral1 oral2 oral3 oral4

```
avoid zeroes)

boxplot(pseudoCount)
```

We can use the ggplot2 package to make nicer plots :-)
```{r}
library(reshape)
library(ggplot2)
# Rename column names:
colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3",
"oral4")
df = melt(pseudoCount) # reshape the matrix
df = data.frame(df, Condition = substr(df$X2, 1, 4))
ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
ylab(expression(log[2](count + 1)))
```

112:1 Chunk 7 R Markdown

Console ~/

```
>
> boxplot(pseudoCount)
> library(reshape)

Attaching package: 'reshape'

The following objects are masked from 'package:S4Vectors':

    expand, rename

> library(ggplot2)
> # Rename column names:
> colnames(pseudoCount) = c("aboral1", "aboral2", "aboral3", "aboral4", "oral1", "oral2", "oral3", "oral4"
)
> df = melt(pseudoCount) # reshape the matrix
> df = data.frame(df, Condition = substr(df$X2, 1, 4))
> ggplot(df, aes(x = df$X2, y = value, fill = Condition)) + geom_boxplot() + xlab("") +
+ ylab(expression(log[2](count + 1)))
>
```

Data

df	132384 obs. of 4 variables
Mnemiopsis_cou…	Large matrix (132384 elements, 1.5 Mb)
Mnemiposis_col…	8 obs. of 3 variables
pseudoCount	Large matrix (132384 elements, 2 Mb)

Some basic R

Most of this is taken from http://www.r-tutor.com/r-introduction

Download R https://cran.r-project.org/mirrors.html
(Pick one of the nearest mirrors)

Download RStudio https://www.rstudio.com/

Variable assignment

We assign values to variables with the assignment operator "=” (can also use “<-”). Just typing the variable by itself at the prompt will print out the value.

```
> x = 1
> x
[1] 1
> x <- 1
> x
[1] 1
>
```

Functions

R functions are invoked by its name, then followed by the parenthesis, and zero or more arguments. The following apply the function c to combine three numeric values into a vector.

```
> c(1, 2, 3)
[1] 1 2 3
```

Comments

All text after the hash tag"#" within the same line is considered a comment.

```
> 1 + 1      # this is a comment
[1] 2
```

Getting help

R provides extensive documentation. For example, entering ?c or help(c) at the prompt gives documentation of the function c in R.

```
> help(c)
```

Basic data types

Numeric

```
> x = 10.5       # assign a decimal value
> x             # print the value of x
[1] 10.5
> class(x)       # print the class name of x
[1] "numeric"
```

Integer

```
> as.integer(3.14)    # coerce a numeric value
[1] 3
```

Basic data types

Logical
A logical value is often created via comparison between variables.

```
> x = 1; y = 2   # sample values
> z = x > y      # is x larger than y?
> z              # print the logical value
[1] FALSE
> class(z)       # print the class name of z
[1] "logical"
```

Basic data types

Logical
Standard logical operations are "&" (and), "|" (or), and "!" (negation).

```
> u = TRUE; v = FALSE
> u & v          # u AND v
[1] FALSE
> u | v          # u OR v
[1] TRUE
> !u             # negation of u
[1] FALSE
```

Basic data types

Character
A character object is used to represent string values in R. We convert objects into character values with the as.character() function:

```
x = as.character(3.14)
> x              # print the character string
[1] "3.14"
> class(x)       # print the class name of x
[1] "character"
```

Basic data types

Character
Two character values can be concatenated with the paste function.

```
> fname = "Joe"; lname ="Smith"
> paste(fname, lname)
[1] "Joe Smith"
```

To extract a substring, we apply the substr function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.

```
> substr("Mary has a little lamb.", start=3, stop=12)
[1] "ry has a l"
```

Basic data types

And to replace the first occurrence of the word "little" by another word "big" in the string, we apply the sub function.

```
> sub("little", "big", "Mary has a little lamb.")
[1] "Mary has a big lamb."
```

Vectors

A vector is a sequence of data elements of the same basic type. Members in a vector are officially called *components* (but usually *members*).

Here is a vector containing three numeric values 2, 3 and 5.

```
> c(2, 3, 5)
[1] 2 3 5
```

And here is a vector of logical values.

```
> c(TRUE, FALSE, TRUE, FALSE, FALSE)
[1]  TRUE FALSE  TRUE FALSE FALSE
```

Vectors

A vector can contain character strings.

```
> c("aa", "bb", "cc", "dd", "ee")
[1] "aa" "bb" "cc" "dd" "ee"
```

The number of members in a vector is given by the length function.

```
> length(c("aa", "bb", "cc", "dd", "ee"))
[1] 5
```

Combining vectors

Vectors can be combined via the function c. For example, the following two vectors n and s are combined into a new vector containing elements from both vectors.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> c(n, s)
[1] "2"  "3"  "5"  "aa" "bb" "cc" "dd" "ee"
```

The values are now strings

Part of a vector

```
> A <- c("aa", "AA", "bb", "BB", "cc", "CC")
> B <- c("aa", "bb", "cc", "dd", "ee", "ff")
> x <- A %in% B
> x
[1] TRUE FALSE TRUE FALSE TRUE FALSE
> A[x]
[1] "aa" "bb" "cc"
```

Vector index

We retrieve values in a vector by declaring an index inside a single square bracket "[]" operator. NB! R is one-based (not zero-based like Python)

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[3]
[1] "cc"
```

Negative index

If the index is negative, it removes the member whose position has the same absolute value as the negative index. For example, the following creates a vector slice with the third member removed.

```
> s[-3]
[1] "aa" "bb" "dd" "ee"
```

Numeric Index Vector

A new vector can be sliced from a given vector with a numeric index vector, which consists of member positions of the original vector to be retrieved.

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[c(2, 3)]
[1] "bb" "cc"
```

Or using a range index

```
> s[2:4]
[1] "bb" "cc" "dd"
```

Matrices

```
> A = matrix(
+   c(2, 4, 3, 1, 5, 7), # the data elements
+   nrow=2,              # number of rows
+   ncol=3,              # number of columns
+   byrow = TRUE)        # fill matrix by rows

> A                      # print the matrix
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Matrices

An element at the mth row, nth column of A can be accessed by the expression A[m, n].

```
> A[2, 3]      # element at 2nd row, 3rd column
[1] 7
```

The entire mth row A can be extracted as A[m,].

```
> A[2, ]       # the 2nd row
[1] 1 5 7
```

Similarly, the entire nth column A can be extracted as A[,n].

```
> A[ ,3]       # the 3rd column
[1] 3 7
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Matrices

We can also extract more than one rows or columns at a time.

```
> A[ ,c(1,3)]  # the 1st and 3rd columns
     [,1] [,2]
[1,]    2    3
[2,]    1    7
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Matrices

We can give names to the columns and rows using colnames() and rownames().

```
>  colnames(A) = c("col1", "col2", "col3")
> A
     col1 col2 col3
[1,]    2    4    3
[2,]    1    5    7
> rownames(A) = c("row1", "row2")
> A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Matrices

Or in one go using dimnames().

```
> dimnames(A) = list(
+   c("row1", "row2"),          # row names
+   c("col1", "col2", "col3")) # column names

> A                    # print A
     col1 col2 col3
row1    2    4    3
row2    1    5    7
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Matrices

If we assign names to the rows and columns of the matrix, than we can access the elements by names.

```
> A["row2", "col3"] # element at 2nd row, 3rd column
[1] 7
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Transpose

We construct the transpose of a matrix by interchanging its columns and rows with the function t.

```
> t(A)      # transpose the matrix A
     [,1] [,2]
[1,]    2    1
[2,]    4    5
[3,]    3    7
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Combining matrices

The columns of two matrices having the same number of rows can be combined into a larger matrix.

```
> B = matrix(c(7, 4, 2), nrow=1, ncol=3)
> B
     [,1]  [,2]  [,3]
[1,]    7     4     2
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Combining matrices

Then we can combine the rows of A and B with rbind (row bind).

```
> rbind(A, B)
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
[3,]    7    4    2
```

BTW; Columns can be combined with cbind.

Lists

A list is a generic vector containing other objects. For example, the following variable x is a list containing copies of three vectors n, s, b, and a numeric value 3.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd", "ee")
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)
> x = list(n, s, b, 3)   # x contains copies of n, s, b
> x
[[1]]
[1] 2 3 5
[[2]]
[1] "aa" "bb" "cc" "dd" "ee”
[[3]]
[1]  TRUE FALSE  TRUE FALSE FALSE
[[4]]
[1] 3
```

Data frames

A data frame is used for storing data tables. It is a list of vectors of equal length. For example, the following variable df is a data frame containing three vectors n, s, b.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)       # df is a data frame
> df
  n  s     b
1 2 aa  TRUE
2 3 bb FALSE
3 5 cc  TRUE
```

Data frames can be indexed, sliced and combined like matrices

Data frames

The individual columns can be accessed by the “$” sign. Very useful when you are plotting a single or more columns.

```
> df$n
[1] 2 3 5
> df$b
[1]  TRUE FALSE  TRUE
```

Data frames

There are many built-in data frames in R for testing purposes. A popular one is called **mtcars**.

```
> mtcars
                    mpg cyl disp  hp drat   wt ...
Mazda RX4          21.0   6  160 110 3.90 2.62 ...
Mazda RX4 Wag      21.0   6  160 110 3.90 2.88 ...
Datsun 710         22.8   4  108  93 3.85 2.32 ...
                   ............
```

The top line of the table, called the **header**, contains the column names. Each horizontal line afterward denotes a **data row**, which begins with the name of the row, and then followed by the actual data. Each data member of a row is called a **cell**.

Viewing

You can use head() and tail() to see the top and last lines of an object (like in Unix)

```
> head(mtcars, 3)           # view top 3 lines
               mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

```
100 a1 b1
200 a2 b2
300 a3 b3
400 a4 b4
```

Read/write files

Assume you have a text file called mydata.txt

```
> mydata = read.table("mydata.txt")
> mydata
            V1  V2 V3
1          100   a1 b1
2          200   a2 b2
3          300   a3 b3
4          400   a4 b4
```

```
c1      c2 c3
100,1 a1 b1
200,2 a2 b2
300,3 a3 b3
400,4 a4 b4
```

Read/write files

If the file is tab-separated (i.e. tabs (\t) between the columns, use sep = “\t”. If the first line specifies the names of the columns, use header = TRUE

```
> mydata = read.table("mydata.txt“, sep = “\t“, dec = “,“, header = TRUE)
> mydata
         c1  c2 c3
1       100  a1 b1
2       200  a2 b2
3       300  a3 b3
4       400  a4 b4
```

There are many more options in read.table(). See help(read.table). There are also other functions for reading files. E.g read.csv() and read.csv2() for reading comma separated files, and read.xls() for excel sheets.

Remove objects

```
> X = 1
> rm(x)
```

And autocomplete also works in R…

Press the tab-button to autocomplete existing objects (and most other things. Try it...)

Working directory

```
> getwd()   # prints the current working directory
> setwd(”file/path”)   # change the working directory
```

Note that the forward slash should be used as the path separator even on Windows platform.

```
> setwd("C:/MyDoc")
```

Packages

One of the main strengths of R are the many available packages which gives extra functionality in addition to the built-in functions. Most packages and information can be found at https://cran.r-project.org/

```
> install.packages(”package”)  # install the package from CRAN
> library(pacakge)    # load the package to make it active
```

- Now we will install the necessary packages for tomorrow
- And then you can do some basic exercises in R – and go home!