# BST02: Using R for Statistics in Medical Research

Part B: Basic use of R

Eleni-Rosalina Andrinopoulou

Department of Biostatistics, Erasmus Medical Center

■ e.andrinopoulou@erasmusmc.nl

24 - 28 February 2020



## In this Section

- Using R
- ► In practice examples
- ► Basics in R
- ► Common R objects
- Importing data and saving your work
- Data transformation
- Data exploration
- Data visualization
- ► Indexing/Subsetting
- ► Lots of practising

- ► R is a command-based procedural language
  - write and execute commands
  - use and define functions
- You may write the commands in the R console (Windows) or in a shell (Linux)

You will become more familiar with the syntax as you use it

- Strongly advisable to use a suitable text editor Some available options:
  - RWinEdt (for Windows; you also need WinEdt installed)
  - ► Tinn-R (for Windows; http://sciviews.org/Tinn-R/)
  - Rkward (for Linux)
  - Emacs (w. ESS, all platforms)
  - Visual Studio (for Windows)
  - Rstudio (all major platforms; http://www.rstudio.org/)
  - ▶ for more check https://r-dir.com/blog/2013/01/list-of-r-editors.html

- ► For this course: Rstudio (http://www.rstudio.org/)
  - ▶ free
  - works fine in Windows, MacOS and Linux
  - helpful with errors

► Can I use R without Rstudio?

► Can I use Rstudio without R?

## ► Package survival - pbc data set

id	time	status	trt	age	sex	bili	chol
1	400	2	1	58.76523	f	14.5	261
2	4500	Ο	1	56.44627	f	1.1	302
3	1012	2	1	70.07255	m	1.4	176
4	1925	2	1	54.74059	f	1.8	244
5	1504	1	2	38.10541	f	3.4	279

- ▶ id: case number
- ▶ time: number of days between registration and the earlier of death, transplantation, or study analysis in July, 1986
- ▶ status: status at endpoint, 0/1/2 for censored, transplant, dead
- ▶ trt: 1/2/NA for D-penicillmain, placebo, not randomised
- **age**: in years
- ► sex: m/f
- bili: serum bilirunbin (mg/dl)
- ► chol: serum cholesterol (mg/dl)

#### More details:

https://stat.ethz.ch/R-manual/R-devel/library/survival/html/pbc.html

id	time	status	trt	age	sex	bili	chol
1	400	2	1	58.76523	f	14.5	261
2	4500	Ο	1	56.44627	f	1.1	302
3	1012	2	1	70.07255	m	1.4	176
4	1925	2	1	54.74059	f	1.8	244
5	1504	1	2	38.10541	f	3.4	279

id	time	status	trt	age	sex	bili	chol
1	400	2	1	58.76523	f	14.5	261
2	4500	0	1	56.44627	f	1.1	302
3	1012	2	1	70.07255	m	1.4	176
4	1925	2	1	54.74059	f	1.8	244
5	1504	1	2	38.10541	f	3.4	279

id	time	status	trt	age	sex	bili	chol
1	400	2	1	58.76523	f	14.5	261
2	4500	Ο	1	56.44627	f	1.1	302
3	1012	2	1	70.07255	m	1.4	176
4	1925	2	1	54.74059	f	1.8	244
5	1504	1	2	38.10541	f	3.4	279

id	time	status	trt	age	sex	bili	chol
1	400	2	1	58.76523	f	14.5	261
2	4500	0	1	56.44627	f	1.1	302
3	1012	2	1	70.07255	m	1.4	176
4	1925	2	1	54.74059	f	1.8	244
5	1504	1	2	38.10541	f	3.4	279

id	time	status	trt	age	sex	bili	chol
1	400	2	1	58.76523	f	14.5	261
2	4500	0	1	56.44627	f	1.1	302
3	1012	2	1	70.07255	m	1.4	176
4	1925	2	1	54.74059	f	1.8	244
5	1504	1	2	38.10541	f	3.4	279

- Common questions
  - What is the average age?
  - What is the average serum bilirubin?
  - ▶ What is the average serum cholesterol?
  - ▶ What is the percentage of females?
  - ▶ How many missing values do we have for serum cholesterol?

#### All these questions can be answered using R!

- ► Elementary commands: **expressions** and **assignments**
- ► An expression given as command is evaluated printed and discarded
- ► An **assignment** evaluates an expression and passes the value to a variable the result is not automatically printed

Expression is given as a command,

103473

[1] 103473

▶ However, it cannot be viewed again unless the command is rerun.

Expression is given as a command,

103473

[1] 103473

▶ However, it cannot be viewed again unless the command is rerun.

In order to store information, the expression should assign the command

```
x <- 103473
```

Х

[1] 103473

#### You can use R as a calculator!

► Basic arithmetics

```
+, -, *, /, ^
```

[1] 203473

Complicated arithmetics

#### Tips:

- ► R is case sensitive, e.g.,
  - ▶ "sex" is different than "Sex"
- ► Commands are separated by a semi-colon or by a newline
- ► Comments can be put anywhere, starting with a hashmark #: everything to the end of the line is a comment
- Assign a value to an object by <- or =</p>

- Missing values
  - ▶ are coded as NA (i.e., not available) is.na()
- ► Infinity
  - ▶ is coded as Inf (plus infinity) or -Inf (minus infinity) is.finite()
- ► The Null objects
  - are coded as NULL
- Not a number
  - ▶ is coded as NaN (Not a Number). Example:

0/0

[1] NaN

#### **Demos**

► Basic R R html

► There are different kinds of variables

id	time	status	trt	age	sex	bili	chol
1	400	2	1	58.77	f	14.5	261
2	4500	Ο	1	56.45	f	1.1	302
3	1012	2	1	70.07	m	1.4	176
4	1925	2	1	54.74	f	1.8	244
5	1504	1	2	38.11	f	3.4	279

- ▶ in R Everything (data, results, ...) is an object
- ▶ In order to list the created objects use the following functions

```
objects()
ls()
```

- ▶ in R Everything (data, results, ...) is an object
- ▶ In order to list the created objects use the following functions

```
objects()
ls()
```

► In order to remove objects

```
rm(...)
rm(list=ls(all=TRUE))
```

► To investigate a specific object (e.g. pbc)

```
str(pbc[,c("id", "time", "status", "trt", "age", "sex", "bili", "chol")])
'data.frame': 418 obs. of 8 variables:
 $ id : int 1 2 3 4 5 6 7 8 9 10 ...
 $ time : int 400 4500 1012 1925 1504 2503 1832 2466 2400 51 ...
 $ status: int 2 0 2 2 1 2 0 2 2 2 ...
 $ trt : int 1 1 1 1 2 2 2 2 1 2 ...
 $ age : num
               58.8 56.4 70.1 54.7 38.1 ...
 $ sex : Factor w/ 2 levels "m", "f": 2 2 1 2 2 2 2 2 2 2 ...
 $ bili : num
               14.5 1.1 1.4 1.8 3.4 0.8 1 0.3 3.2 12.6 ...
 $ chol : int
               261 302 176 244 279 248 322 280 562 200 ...
```

The simplest data structures are:

- ▶ **numeric** : quantitative data
- character: qualitative data
- ► integer : whole numbers
- ► logical : TRUE or FALSE
- ► factors : qualitative data (levels)

To find out what type of object you have, you can use the following function

```
mode(pbc$age)
```

[1] "numeric"

- Data structure:
  - Vectors have the same type of elements
  - ▶ Matrices have the same type of elements with the same length
  - ▶ Data.frames have elements of different type with the same length
  - ▶ Lists have elements of different type and length

▶ Differences between **vector**, matrix, data.frame and list

```
pbc[1:6, c("age")]
```

[1] 58.76523 56.44627 70.07255 54.74059 38.10541 66.25873

▶ Differences between vector, **matrix**, data.frame and list

```
pbc[1:6, c("age", "bili", "chol")]
      age bili chol
1 58.76523 14.5 261
2 56.44627 1.1 302
3 70.07255 1.4 176
4 54 74059 1.8 244
5 38.10541 3.4 279
6 66.25873 0.8 248
```

▶ Differences between vector, matrix, data.frame and list

```
pbc[1:6, c("id", "sex", "bili", "chol")]
```

```
id sex bili chol
1 1 f 14.5 261
2 2 f 1.1 302
3 3 m 1.4 176
4 4 f 1.8 244
5 5 f 3.4 279
6 6 f 0.8 248
```

Differences between vector, matrix, data.frame and list

```
list(pbc[1:6, c("sex")], pbc[1:2, c("sex", "bili")], pbc$age[1:4])
[[1]]
\lceil 1 \rceil f f m f f f
Levels: m f
[[2]]
  sex bili
1 f 14.5
2 f 1.1
[[3]]
[1] 58.76523 56.44627 70.07255 54.74059
```

#### **Create a vector**

```
vec <- c(1, 2, 3, 4, 5)
vec
```

[1] 1 2 3 4 5

vec <- c(1:5)
vec</pre>

[1] 1 2 3 4 5

#### Create a matrix

```
vec <- c(1, 2, 3, 4, 5, 6)
mat <- matrix(vec, 3, 3)
mat</pre>
```

```
[,1] [,2] [,3]
[1,] 1 4 1
[2,] 2 5 2
[3,] 3 6 3
```

#### Create a matrix

3

[2,] 4 5 6

[3,] 1 2

#### Create a data frame

```
dtf <- data.frame(pbc$sex, pbc$age)
dtf[1:3,]

  pbc.sex  pbc.age
1     f 58.76523
2     f 56.44627
3     m 70.07255</pre>
```

#### Create a data frame

```
dtf <- data.frame(pbc$sex, pbc$age)
dtf[1:3,]

  pbc.sex  pbc.age
1     f 58.76523
2     f 56.44627
3     m 70.07255</pre>
```

	Gender	Age
1	f	58.76523
2	f	56.44627
3	m	70.07255

# **Common R Objects**

#### Create a list

```
list1 <- list(vec = c(1:5), mat = pbc[1:2, c("age", "sex")])
list1
$vec
[1] 1 2 3 4 5
$mat
      age sex
1 58.76523 f
2 56.44627 f
```

# **Common R Objects**

#### **Demos**

► Common R Objects R html

#### **Practicals**

► Common R Objects html

- function: read.table(), read.csv() and its variants
  - note: use forward slashes or double backward slashes in the file names, e.g.,

```
"C:/Documents and Settings/User/Data/file.txt" Or "C:\\Documents and Settings\\User\\Data\\file.txt"
```

- Specialized functions for importing data from other programs
  - package: foreign, function: read.spss(), read.dta()
  - package: Hmisc, function: sas.get()
  - package: openxlsx, function: read.xlsx()
  - package: readxl, function: read\_excel()
  - package: haven, function: read\_spss()
  - etc

### Tips:

- Short names are prefered over longer names
- Try to avoid using names that contain symbols
- Avoid spaces in names
- ► Remove any comments in your data set
- Make sure that any missing values in your data set are indicated with the same value (or no value)

- save()
  - can be used to save a list of R objects
  - a binary file with all the objects available in your last R session
- You can load your saved R objects using load()
  - be careful about overwriting
- Using saveRDS and readRDS you can save and read a single R object
  - The result has to be assigned to a variable

- Specialized functions for exporting data to other programs
  - function: write.table(), write.csv()
  - package: foreign, function: write.spss(), write.dta()
  - package: openxlsx, function: write.xlsx()
  - etc

#### **Demos**

► Importing and Saving R html

#### **Practicals**

► Importing and Saving html

### You will never receive the perfect data set!

- Round continuous variables
- Convert numeric variables to factors
- Compute new variables
  - transform variables
- ▶ Data sets of wide ⇔ long format

Round continuous variables

```
id time status trt age sex bili chol
1 400 2 1 59 f 14.5 261
2 2 4500 0 1 56 f 1.1 302
3 1012 2 1 70 m 1.4 176
4 1925 2 1 55 f 1.8 244
5 5 1504 1 2 38 f 3.4 279
6 6 2503 2 2 66 f 0.8 248
```

Convert numeric variables to factors

```
id time status trt age sex bili chol
1 1 400 2 D-penicillmain 58.76523 f 14.5 261
2 2 4500 0 D-penicillmain 56.44627 f 1.1 302
3 3 1012 2 D-penicillmain 70.07255 m 1.4 176
4 4 1925 2 D-penicillmain 54.74059 f 1.8 244
5 5 1504 1 placebo 38.10541 f 3.4 279
6 6 2503 2 placebo 66.25873 f 0.8 248
```

- Compute new variables
  - transform variables

```
id
      time status trt
                         age sex bili chol time years
   13.33333
               2 1 58.76523
                              f 14.5 261
                                           1.111111
2 150 00000
               0 1 56.44627 f 1.1 302
                                          12.500000
   33.73333
               2 1 70.07255 m 1.4 176 2.811111
   64.16667
               2 1 54.74059 f 1.8 244 5.347222
                1 2 38.10541 f 3.4 279 4.177778
5
   50.13333
                   2 66.25873 f 0.8 248
6
   83.43333
                                           6.952778
```

▶ Data sets of wide ←⇒ long format

▶ Data sets of wide ⇔ long format

```
id futime status trt age day sex bili chol
1 1 400 2 1 58.76523 0 f 14.5 261
2 1 400 2 1 58.76523 192 f 21.3 NA
3 2 5169 0 1 56.44627 0 f 1.1 302
4 2 5169 0 1 56.44627 182 f 0.8 NA
5 2 5169 0 1 56.44627 365 f 1.0 NA
6 2 5169 0 1 56.44627 768 f 1.9 NA
```

#### **Demos**

► Data Transformation R html

- Common questions
  - What is the mean and standard deviation for age?
  - What is the mean and standard deviation for time?
  - What is the median and interquartile range for age?
  - What is the percentage of placebo patients?
  - ► What is the percentage of females?
  - What is the mean and standard deviation for age in males?
  - ▶ What is the mean and standard deviation for baseline serum bilirubin?
  - ▶ What is the percentage of missings in serum bilirubin?

### All these questions can be answered using R!

**▶** Hints

► Check functions: mean(...), sd(...), percent(...), median(...), IQR(...)

**▶** Hints

► Check functions: mean(...), sd(...), percent(...), median(...), IQR(...)

What is the mean value for age?

mean(pbc\$age)

[1] 50.74155

#### **Demos**

► Data Exploration R html

- ▶ It is important to investigate each variable in our data set using plots
  - Descriptive statistics for continuous and categorical variables
  - Distributions of variables
  - Distributions of variables per group
  - Extreme values
  - ► Linear/nonlinear evolutions

- ► R has very powerful graphics capabilities
- ► Some good references are
  - ▶ Murrel, P. (2005) R Graphics. Boca Raton: Chapman & Hall/CRC.
  - Sarkar, D. (2008) Lattice Multivariate Data Visualization with R. New York: Springer-Verlag.

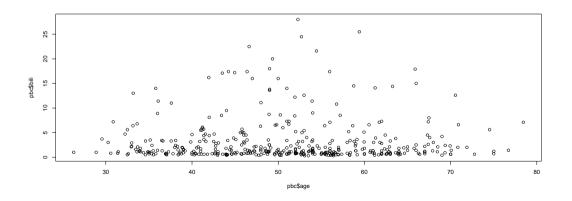
- ► Traditional graphics system
  - package graphics
- ► Trellis graphics system
  - package lattice (which is based on package grid)
- ► Grammar of Graphics implementation (i.e., Wilkinson, L. (1999) The Grammar of Graphics. New York: Springer-Verlag)
  - packages ggplot & ggplot2

### Important plotting basic functions

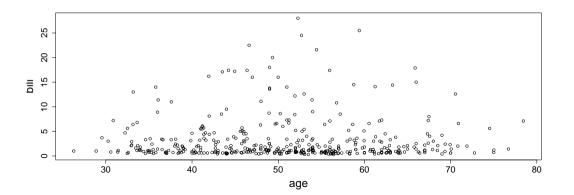
- plot(): scatter plot (and others)
- **▶ barplot()**: bar plots
- **boxplot()**: box-and-whisker plots
- ► dotchart(): dot plots
- ▶ hist(): histograms
- ▶ pie(): pie charts
- qqnorm(), qqline(), qqplot(): distribution plots
- ▶ pairs(): for multivariate data

Continuous variables

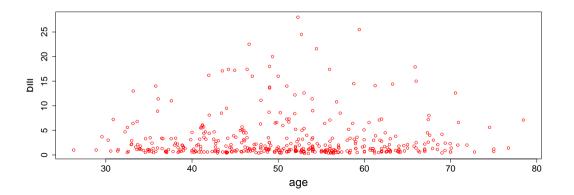
plot(pbc\$age, pbc\$bili)



#### Continuous variables

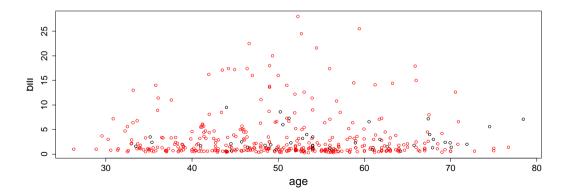


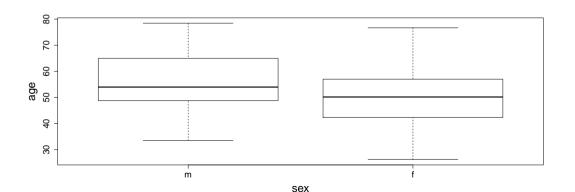
#### Continuous variables



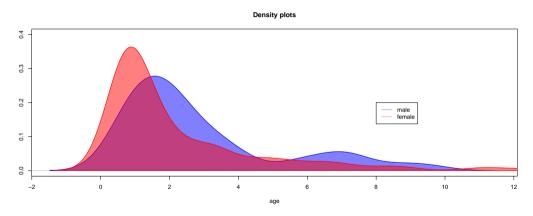
► For more options check

?plot

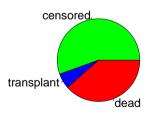




```
pbc male bili <- pbc$bili[pbc$sex == "m"]</pre>
pbc female bili <- pbc$bili[pbc$sex == "f"]</pre>
plot(density(pbc male bili), col = rgb(0.0.1.0.5),
     main = "Density plots", xlab = "age", vlab = "")
polygon(density(pbc_male_bili), col = rgb(0,0,1,0.5),
        border = "blue")
lines(density(pbc_female_bili), col = rgb(1,0,0,0.5))
polygon(density(pbc female bili), col = rgb(1,0,0,0.5),
        border = "red")
legend(8,0.2, c("male", "female"),
       col = c(rgb(0,0,1,0.5), rgb(1,0,0,0.5)), ltv = 1)
```



### Categorical variables



#### **Demos**

► Data Visualization R html

### **Practicals**

► Data Manipulation html

# **Useful Summary: Data manipulation**

### **Common R objects**

- ► c()
- ▶ matrix()
- data.frame()
- ▶ list()
- cbind(), rbind()

### **Transformation**

- ► factor()
- reshape()
- order()
- ► complete.cases()

#### **Exploration**

- ▶ is.na()
- ▶ dim()
- ▶ mean(), sd()
- ▶ median().IQR()
- percent()

### Visualization

- ▶ plot(), legend()
  - ▶ hist()
  - barchart()
- boxplot()
- xyplot(), ggplot()
- ▶ par()

# Import/Save/Present

- head()
- save(), saveRDS()
- load(), readRDS()
- read.csv(), write.csv(),
  read.xlsx(),
  write.xlsx(),
  - read.table(),
    write.table()

### Other

- install.packages(),
  library()
- ► ls(), objects(), getwd()

- When transforming and analyzing data we often need to select specific observations or variables
  - Examples: Select ...
    - ▶ the 3rd element for vector age
    - the 3rd column of the pbc data set
    - ▶ the sex of the 10th patient
    - the baseline details of the 5th patient
    - the serum cholesterol for all males
    - the age for male patients or patients that have serum bilirubin more than
       3
    - ▶ the first measurement per patient

- ▶ This can be done using square bracket ([]) notation and indices.
- ► Three basic types
  - position indexing
  - ► logical indexing
  - name indexing

### Indexing with vector

For position indexing, use a **positive** value to select an element

```
x <- c(6:17)
x
[1] 6 7 8 9 10 11 12 13 14 15 16 17
x[2]
```

[1] 7

Use multiple positive values to select multiple elements

```
x[c(2,3,4)]
```

[1] 7 8 9

### Indexing with vector

 $\lceil 1 \rceil \ 7 \ 7 \ 7$ 

► For position indexing, use dublicated **positive** values to select the same elements

```
x <- c(6:17)
x

[1] 6 7 8 9 10 11 12 13 14 15 16 17

x[c(2,2,2)]
```

66

### Indexing with vector

► For position indexing, use a **negative** value to remove an element

```
x <- c(6:17)
x

[1] 6 7 8 9 10 11 12 13 14 15 16 17

x[-5]
```

Positive and negative indices cannot be combined

#### Indexing with vector

► Use logical index of the same length to select elements where the value is **TRUE** 

```
x <- c(6:10)
y <- c(TRUE, FALSE, FALSE, FALSE)
x[y]</pre>
```

Γ17 6

### Indexing with vector

[1] 8 9 10

Use logical indexing in combination with conditions

```
x < -c(6:10)
x[x > 7]
[1] 8 9 10
x[x > 7 & x > 9]
[1] 10
x[x > 7 | x > 9]
```

#### Indexing with vector

► For name/character indexing, use the name of the element

```
x <- c(foo=5, bar=4, one=7, two=12, three=2)
x[c('foo', 'one')]

foo one
   5   7</pre>
```

▶ Use the function names to obtain the names

### Indexing with matrix

- ► Indexing matrices are similar to indexing vectors but with double index
  - ▶ The first position denotes the rows ["index", ]
  - ► The first position denotes the columns [, "index"]

#### Indexing with matrix

- ► Indexing matrices are similar to indexing vectors but with double index
  - ▶ The first position denotes the rows ["index",]
  - ► The first position denotes the columns [, "index"]

```
mat <- matrix(1:4, 2, 2)
mat

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

Use position indexing as:

```
mat <- matrix(1:4, 2, 2)
mat[2, 2]
```

[1] 4

### Indexing with matrix

When we leave a position blank all elements are selected

```
mat <- matrix(1:4, 2, 2)
mat

[,1] [,2]
[1,] 1 3
[2,] 2 4

mat[2,]</pre>
```

[1] 2 4

#### Indexing with data.frame

- Works with single and double index
- ▶ When using a single index the data frame acts like a list of variables

4 4 45

#### Indexing with data.frame

- Works with single and double index
- ▶ When using a single index the data frame acts like a list of variables

```
DF <- data.frame(x = 1:4,
y = c(35, 23, 14, 45))
DF

x y
1 1 35
2 2 23
3 3 14
4 4 45
```

Use position indexing as:

DF[2]

#### Indexing with data.frame

\$ provides a convinient notation to extract an element by name

head(pbc\$time)

[1] 400 4500 1012 1925 1504 2503

### Indexing with data.frame

 When using a double index, indexing works like a matrix

```
DF <- data.frame(x = 1:3,
y = c(35, 23, 14))
DF
```

```
x
```

- 1 1 35
- 2 2 23
- 3 3 14

### Indexing with data.frame

When using a double index, indexing works like a matrix

DF

x y 1 1 35 2 2 23

3 3 14

Use position indexing

```
DF[2,]
```

x y 2 2 2 3

Use logical indexing

DF[DF\$y < 30, ]

x y 2 2 23 3 3 14

#### Indexing with data.frame

Combine logical and position indexing in data frame

```
pbcDF <- data.frame(pbc)
head(pbc[pbc$sex == "m", 1:7])</pre>
```

```
    id
    time
    status
    trt
    age
    sex
    ascites

    3
    3
    1012
    dead
    1
    70.07255
    m
    0

    14
    14
    1217
    dead
    2
    56.22177
    m
    1

    21
    21
    3445
    censored
    2
    64.18891
    m
    0

    24
    24
    4079
    dead
    1
    44.52019
    m
    0

    48
    48
    4427
    censored
    2
    49.13621
    m
    0

    52
    52
    2386
    dead
    1
    50.54073
    m
    0
```

### Indexing with data.frame

Combine logical and position indexing in data frame

```
pbcDF <- data.frame(pbc)
head(pbc[pbc$age > 30 | pbc$sex == "f", 1:7])
```

	${\tt id}$	time	status	trt	age	sex	ascites
1	1	400	dead	1	58.76523	f	1
2	2	4500	censored	1	56.44627	f	0
3	3	1012	dead	1	70.07255	m	0
4	4	1925	dead	1	54.74059	f	0
5	5	1504	${\tt transplant}$	2	38.10541	f	0
6	6	2503	dead	2	66.25873	f	0

### Indexing with data.frame

Combine logical and position indexing in data frame

```
pbcDF <- data.frame(pbc)
head(pbc[pbc$age > 30 & pbc$sex == "f", 1:7])
```

	${\tt id}$	time	status	trt	age	sex	ascites
1	1	400	dead	1	58.76523	f	1
2	2	4500	censored	1	56.44627	f	0
4	4	1925	dead	1	54.74059	f	0
5	5	1504	transplant	2	38.10541	f	0
6	6	2503	dead	2	66.25873	f	0
7	7	1832	censored	2	55.53457	f	0

#### Indexing with list

► Lists can be subsetted in the same way as vectors using single brackets - Note that the output is a list

Use position indexing

```
mylist <- list(y = c(14, 45), z = c("m", "f", "f"))
mylist[2]
```

```
$z
[1] "m" "f" "f"
```

#### Indexing with list

► Double square brackets can be also used - Note that the output is a vector

Use position indexing

```
mylist <- list(y = c(14, 45), z = c("m", "f", "f"))
mylist[[2]]
```

```
[1] "m" "f" "f"
```

#### Indexing with list

[1] 14 45

\$ provides a convinient notation to extract an element by name -Note that the output is a vector

```
mylist \leftarrow list(y = c(14, 45), z = c("m", "f", "f"))
mylist
$y
[1] 14 45
$z
[1] "m" "f" "f"
mylist$y
```

8

 Use the following webpage to further investigate indexing and subsetting https://emcbiostatistics.shinyapps.io/indexing/

#### **Demos**

Shiny app indexing subsetting R

In order to run the app you will need to install the packages:

- survival
- shiny

# **Useful Summary: Indexing/Subsetting**

#### **Demos**

► Indexing/Subsetting R html

#### **Practicals**

► Indexing/Subsetting html

# **Useful Summary: Indexing/Subsetting**

#### **Vectors**

- **>** []
- ► [""] for categorical variables

#### **Matrices**

**(**,]

#### **Data frames**

- **(**,]
- **▶** [[]], []
- **\$**

#### Lists

- **(**]
- **▶** [[]]
- ▶ \$