

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

Using R

- ▶ 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

Using R (cont'd)

- ▶ 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>

Using R (cont'd)

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

Using R (cont'd)

- ▶ Can I use R without Rstudio?
- ▶ Can I use Rstudio without R?

Practical Examples

► 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	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

Practical Examples (cont'd)

- ▶ **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>

Practical Examples (cont'd)

- What is a **scalar**/vector/matrix

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

Practical Examples (cont'd)

- What is a scalar/**vector**/matrix

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

Practical Examples (cont'd)

- What is a scalar/**vector**/matrix

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

Practical Examples (cont'd)

- What is a scalar/vector/matrix

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

Practical Examples (cont'd)

- What is a scalar/vector/**matrix**

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

Practical Examples (cont'd)

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

Basics in 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

Basics in R (cont'd)

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
```

```
x
```

```
[1] 103473
```

Basics in R (cont'd)

- ▶ Basic arithmetics

`+, -, *, /, ^`

- ▶ Complicated arithmetics

Basics in R (cont'd)

- ▶ 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 **=**

Basics in R (cont'd)

- ▶ 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()**
- ▶ Not a number
 - ▶ is coded as **NaN** (Not a Number). Example:

```
0/0
```

```
[1] NaN
```

Basics in R (cont'd)

Demos

► Basic R [R](#) [html](#)

Common R Objects

- 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	0	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

Common R Objects (cont'd)

- ▶ in R Everything (data, results, ...) is an object
- ▶ In order to list the created variables use `objects()` or `ls()`

```
objects()
```

- ▶ In order to remove everything:

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

Common R Objects (cont'd)

- 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 ...
```

Common R Objects (cont'd)

The simplest data structures are:

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

To find out what type of vector you have you can use the mode function

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

```
[1] "numeric"
```

Common R Objects (cont'd)

- ▶ Every object in R has a class
 - ▶ **Vectors** have the same type of elements
 - ▶ **Matrices** have the same type of elements
 - ▶ **Data.frames** and lists do not need to have the same type of elements
 - ▶ **Lists** may have elements of different length and type

Common R Objects (cont'd)

- Differences between **Vector**, Matrices, Data.frames and Lists

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

```
[1] 58.76523 56.44627 70.07255 54.74059 38.10541 66.25873
```

Common R Objects (cont'd)

- Differences between Vector, **Matrices**, Data.frames and Lists

```
pbmc[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

Common R Objects (cont'd)

- Differences between Vector, Matrices, **Data.frames** and Lists

```
pbpc[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

Common R Objects (cont'd)

- Differences between Vector, Matrices, Data.frames and **Lists**

```
list(pbc[1:6, c("sex")], pbc[1:2, c("sex", "bili")], pbc$age[1:4])
```

```
[[1]]
```

```
[1] 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
```

Common R Objects (cont'd)

Create a vector

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

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

```
vec <- c(1:5)  
vec
```

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

Common R Objects (cont'd)

Create a matrix

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

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

```
vec <- c(1, 2, 3, 4, 5, 6)
mat <- matrix(vec, 3, 3,
              byrow = TRUE)
mat
```

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

Common R Objects (cont'd)

Create a data frame

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

	pbcs.sex	pbcs.age
1	f	58.76523
2	f	56.44627
3	m	70.07255

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

	Gender	Age
1	f	58.76523
2	f	56.44627
3	m	70.07255

Common R Objects (cont'd)

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 (cont'd)

Demos

▶ Common R Objects [R](#) [html](#)

Importing Data and Saving your Work

- ▶ 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 statistical packages
 - ▶ package: **foreign**, function: **read.spss()** , **read.dta()**
 - ▶ package: **Hmisc**, function: **sas.get()**
 - ▶ package: **readxl**, function: **read_excel()**
 - ▶ package: **haven**, functio: **read_spss()**
 - ▶ etc

Importing Data and Saving your Work (cont'd)

Tips:

- ▶ Short names are preferred 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)

Importing Data and Saving your Work (cont'd)

- ▶ **ls()** lists R objects in a session
- ▶ **save()**
 - ▶ can be used to save a list of R objects
 - ▶ a binary file with all the objects available in your last R session
 - ▶ platform independent
- ▶ 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 importing data from other statistical packages
 - ▶ function: **read.table()**, **read.csv()**
 - ▶ package: **foreign**, function: **write.spss()**, **write.dta()**
 - ▶ etc

Importing Data and Saving your Work (cont'd)

- Create a csv, xls or text file from an R object

```
write.csv(mydata, "c:/mydata.txt")  
write.table(mydata, "c:/mydata.txt", sep="\t")  
library(xlsx)  
write.xlsx(mydata, "c:/mydata.xlsx")
```

Importing Data and Saving your Work (cont'd)

Demos

- ▶ Importing and Saving **R** **html**

Data Transformation

- ▶ Round continuous variables
- ▶ **numeric** variables to **factors**
- ▶ Compute new variables
 - ▶ transform variables
- ▶ Matrices of wide \iff long format
- ▶ Missing values

Data Transformation (cont'd)

Round continuous variables

```
DF <- format(pbc[,c("id", "time", "status", "trt", "age",  
                    "sex", "bili", "chol")], digits = 2)  
head(DF)
```

	id	time	status	trt	age	sex	bili	chol
1	1	400	2	1	59	f	14.5	261
2	2	4500	0	1	56	f	1.1	302
3	3	1012	2	1	70	m	1.4	176
4	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

Data Transformation (cont'd)

numeric variables to **factors**

```
DF <- pbc[,c("id", "time", "status", "trt", "age",  
            "sex", "bili", "chol")]  
DF$trt <- factor(DF$trt, levels = c(1, 2),  
                labels = c("D-penicillmain", "placebo"))  
head(DF)
```

	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

Data Transformation (cont'd)

- Compute new variables
 - transform variables

```
DF <- pbc[,c("id", "time", "status", "trt", "age",  
            "sex", "bili", "chol")]  
DF$time <- DF$time/30  
DF$time_years <- DF$time/12  
head(DF)
```

	id	time	status	trt	age	sex	bili	chol	time_years
1	1	13.33333	2	1	58.76523	f	14.5	261	1.111111
2	2	150.00000	0	1	56.44627	f	1.1	302	12.500000
3	3	33.73333	2	1	70.07255	m	1.4	176	2.811111
4	4	64.16667	2	1	54.74059	f	1.8	244	5.347222
5	5	50.13333	1	2	38.10541	f	3.4	279	4.177778
6	6	83.43333	2	2	66.25873	f	0.8	248	6.952778

Data Transformation (cont'd)

- ▶ Matrices of **wide** \iff long format

```
head(pbc[,c("id", "time", "status", "trt", "age",  
            "sex", "bili", "chol")])
```

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

Data Transformation (cont'd)

- ▶ Matrices of wide \iff **long** format

```
head(pbcseq[, c("id", "futime", "status", "trt", "age", "day",  
               "sex", "bili", "chol")])
```

	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

Data Transformation (cont'd)

► Missing values

```
head(pbc[is.na(pbc$chol) == TRUE, c("id", "time", "status",  
                                     "trt", "age", "sex",  
                                     "bili", "chol")])
```

	id	time	status	trt	age	sex	bili	chol
14	14	1217	2	2	56.22177	m	0.8	NA
40	40	4467	0	1	46.66940	f	1.3	NA
41	41	1350	2	1	33.63450	f	6.8	NA
42	42	4453	0	2	33.69473	f	2.1	NA
45	45	4025	0	2	41.79329	f	0.6	NA
49	49	708	2	2	61.15264	f	0.8	NA

Data Transformation (cont'd)

Demos

▶ Data Transformation [R](#) [html](#)

Data Exploration

- ▶ Common questions
 - ▶ What is the mean and sd for age?
 - ▶ What is the mean and sd 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 sd for age in males?
 - ▶ What is the mean and sd for baseline serum bilirubin?

Data Exploration (cont'd)

- ▶ Common questions: **Hints**
- ▶ Check functions: **mean(...), sd(...), percent(...), median(...), IQR(...)**

Data Exploration (cont'd)

Demos

► Data Exploration **R** **html**

Data Visualization

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

Data Visualization (cont'd)

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

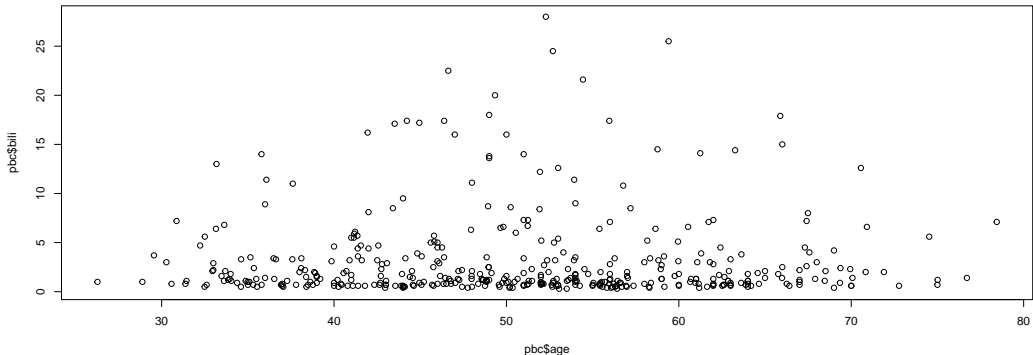
Data Visualization (cont'd)

- ▶ 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**

Data Visualization (cont'd)

Continuous variables

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



Data Visualization (cont'd)

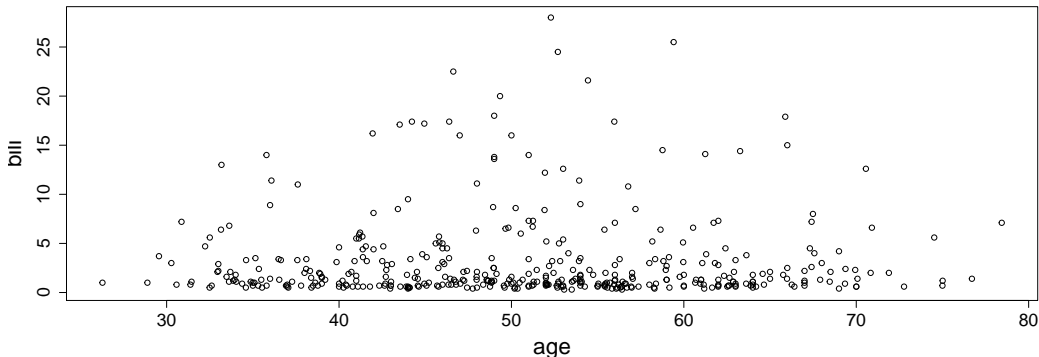
Important plotting functions

- ▶ **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

Data Visualization (cont'd)

Continuous variables

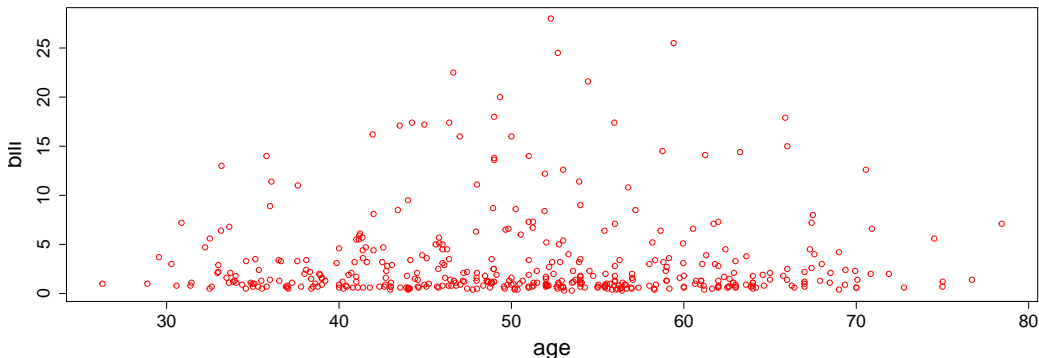
```
plot(pbc$age, pbc$bili, xlab = "age", ylab = "bili",  
     cex.lab = 1.9, cex.axis = 1.5)
```



Data Visualization (cont'd)

Continuous variables

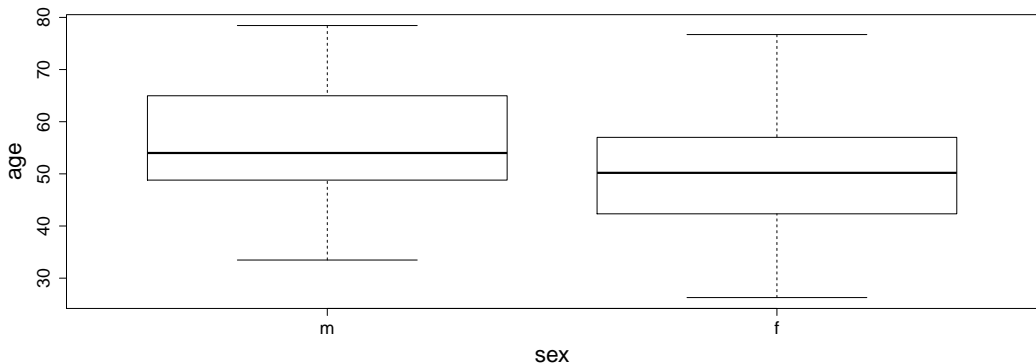
```
plot(pbc$age, pbc$bili, xlab = "age", ylab = "bili",  
     cex.lab = 1.9, cex.axis = 1.5, col = "red")
```



Data Visualization (cont'd)

Continuous variables per group

```
boxplot(pbc$age ~ pbc$sex, xlab = "sex", ylab = "age",  
        cex.lab = 1.9, cex.axis = 1.5)
```



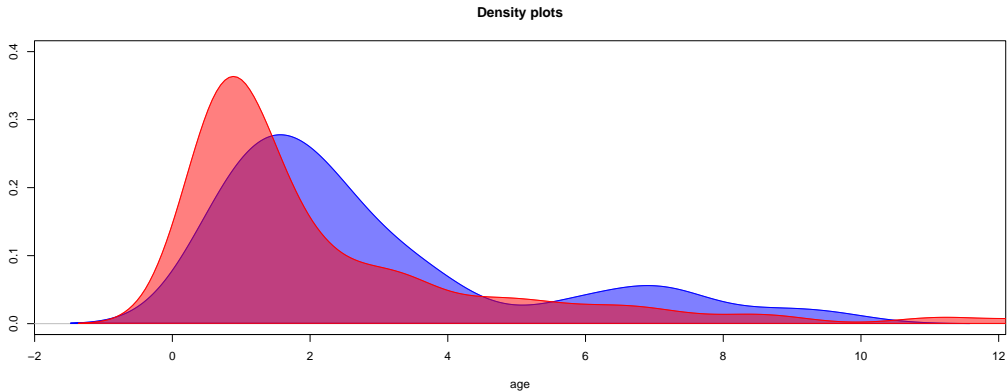
Data Visualization (cont'd)

Continuous variables per group

```
pbm_male_bili <- pbm$bili[pbm$sex == "m"]
pbm_female_bili <- pbm$bili[pbm$sex == "f"]
plot(density(pbm_male_bili), col = rgb(0,0,1,0.5),
     main = "Density plots", xlab = "age", ylab = "")
polygon(density(pbm_male_bili), col = rgb(0,0,1,0.5), border = "blue")
lines(density(pbm_female_bili), col = rgb(1,0,0,0.5))
polygon(density(pbm_female_bili), col = rgb(1,0,0,0.5), border = "red")
legend(20,0.03, c("male", "female"),
      col = c(rgb(0,0,1,0.5), rgb(1,0,0,0.5)), lty = 1)
```

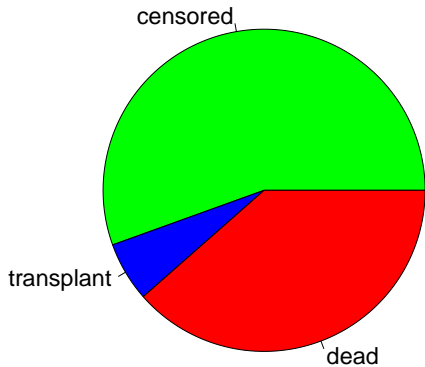
Data Visualization (cont'd)

Continuous variables per group



Data Visualization (cont'd)

Categorical variables



Data Visualization (cont'd)

Demos

► Data Visualization [R](#) [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()`
- ▶ `par()`
- ▶ `xyplot()`
- ▶ `ggplot()`

Import/Save/Present

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

Other

- ▶ `install.packages()`
- ▶ `library()`
- ▶ `ls(), getwd()`

Indexing/Subsetting

- ▶ 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

Indexing/Subsetting (cont'd)

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

Indexing/Subsetting (cont'd)

Position indexing with vectors:

- 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/Subsetting (cont'd)

Position indexing with vectors:

- Use duplicated 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)]
```

```
[1] 7 7 7
```

Indexing/Subsetting (cont'd)

Position indexing with vectors:

- 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]
```

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

- **Positive and negative indices cannot be combined**

Indexing/Subsetting (cont'd)

Logical indexing with vectors:

- 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, FALSE)
x[y]
```

```
[1] 6
```

Indexing/Subsetting (cont'd)

Logical indexing with vectors:

- Logical indexing is often used 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]
```

```
[1] 8 9 10
```

Indexing/Subsetting (cont'd)

Name/character indexing with vectors:

- ▶ When a vector has been named the element names can be used as indices

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

```
foo one
  5   7
```

- ▶ Use the function `names` to obtain the names

Indexing/Subsetting (cont'd)

Indexing with Matrices:

- ▶ Indexing matrices as similar to indexing vectors but with double index
 - ▶ The first position denotes the rows
 - ▶ The first position denotes the columns

```
mat <- matrix(1:6, 3, 3)
```

```
mat
```

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

```
mat[2, 2]
```

```
[1] 5
```

Indexing/Subsetting (cont'd)

Indexing with Matrices:

- ▶ When we leave a position blank all elements are selected

```
mat <- matrix(1:6, 3, 3)
mat
```

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

```
mat[2, ]
```

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


Indexing/Subsetting (cont'd)

Indexing with Data Frames:

- ▶ When you use 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

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

DF[2]

	y
1	35
2	23
3	14
4	45

Indexing/Subsetting (cont'd)

Indexing with Data Frames:

- ▶ When you using a double index, indexing works like a matrix

```
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

```
DF[4, 2]
```

```
[1] 45
```

- ▶ When you using a double index, indexing works like a matrix

```
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

```
DF[DF$y < 30, ]
```

	x	y
2	2	23

Indexing/Subsetting (cont'd)

Indexing with Data Frames:

- Logical indexing in data frame

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

	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/Subsetting (cont'd)

Indexing with Data Frames:

- Logical indexing in data frame

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

	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	transplant	2	38.10541	f	0
6	6	2503	dead	2	66.25873	f	0

Indexing/Subsetting (cont'd)

Indexing with Data Frames:

- Logical indexing in data frame

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

	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/Subsetting (cont'd)

Indexing with Lists:

- Lists can be subsetting in the same way as vectors using - Note that the output is a list

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

```
$y  
[1] 14 45
```

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

```
mylist[2]
```

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

Indexing/Subsetting (cont'd)

Indexing with Lists:

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

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

```
$y  
[1] 14 45
```

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

```
mylist[[2]]
```

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

Indexing/Subsetting (cont'd)

Indexing with Lists:

- ▶ `$` provides a convenient notation to extract an element by name - Note that the output is a vector

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

```
$y  
[1] 14 45
```

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

```
mylist$y
```

```
[1] 14 45
```


Useful Summary: Indexing/Subsetting

Vectors

- ▶ `[]`
- ▶ `[""]`

Matrices

- ▶ `[,]`

Data frames

- ▶ `[,]`
- ▶ `$`
- ▶ `[[]]`

Lists

- ▶ `[]`
- ▶ `[[]]`
- ▶ `$`

Useful Summary: Indexing/Subsetting (cont'd)

Demos

- ▶ Indexing Subsetting [R](#) [html](#)

Practicals

- ▶ Basic Use R [html](#)