Introduction to R

Department of Epidemiology and Data Science

9 - 13 March 2026

Part II

Day 3 and 4

Outline

Workspace management

- R uses **working directories**
- Working directory is a file path setting the location of any files you will use
- You can only have one active working directory at the time
- `getwd()` shows what your working directory is
- `setwd()` allows you to define your working directory

```
> setwd("C:\\Users\\P073787\\Documents\\")
> getwd()
```

Projects in RStudio

- RStudio allows you to create a R **project**
- It is a work directory designated with a `.Rproj` file
- Opening the project (File -> Open Project) will automatically set the working directory to the project working directory
- Recommendation: create for every research project a separate R project

R workspace

- The R **workspace** (or **working environment**) are all your objects and functions you have defined in the current session or loaded from a previous session=
- Objects can also be removed from the workspace with `rm()`

```
> rm(titanic3) #removes titanic3

> rm(list=ls()) #removes all objects
```

Importing Data: txt files

- Data frames in text format can be imported via `read.table()`:
- Many arguments, but the most important ones are:
    - `file`: the name of the file which the data are to be read from.
    - `header`: a logical value indicating whether the file contains the names of the variables as its first line, default=`TRUE`
    - `sep`: values on each line of the file are separated by this character, default:`""`
    - `dec`: the character used in the file for decimal points, default: `"."`
    - `row.names`: vector of row names or number giving the column with row names.
- `read.csv()` and `read.delim()` are identical to `read.table()` apart from other defaults: they are intended for comma-separated and tab-delimited files, respectively.

Importing Data: txt files

- **Common problems** when reading in tabular data are (especially when you use "Save as - tab-delimited file" from Excel):
  - Additional tabs: between columns or at the end of a row
  - Extra empty returns at the end of the file
  - Unusual characters such as the # symbol (see option `comment.char`) and " quotes (see option quote)
  - Presence of blank fields
  - Regional settings problems: decimal separator
  - Invisible spaces
- Use `dim()`, `head()`, `...` to compare the imported data with the original data file
- Be careful when using Excel as an intermediate in manipulating files: https://www.bbc.com/news/technology-54423988

Importing Data: other formats

- Data from SPSS, Stata and SAS: package **haven**: `read_dta()`, `read_sav()`, and `read_sas()`:

```
> library(haven)
> # STATA
> titanic3 <- read_dta("titanic3.dta")
```

- xls and xlsx: package **readxl** (`read_excel()` or `read_xlsx()` and `read_xls()`)
- Note: other packages exist.

Exporting Data

- Export to text file: `write.table()`
- Export to SPSS, Stata and SAS: package **haven** (`write_sav()`, `write_dta()`, `write_sas()`)
- Export to xlsx: package **writexl** (`write_xlsx()`)

Help with Importing/Exporting Data

- See R Data Import/Export Manual under **Help** or **Help - R Help (RStudio)**
- In RStudio via the menu **Import Dataset**. See https://support.posit.co/hc/en-us/articles/218611977-Importing-Data-with-RStudio
- See http://r4stats.com/examples/data-import/

EXERCISES

Let's do Questions 7, 8, 9

Outline

Sorting data (I)

Three functions to help you sort data:

- `sort()`: sorts a vector in an ascending order (default) or descending order (set `decreasing = TRUE`)
- `rank()`: gives the respective rank of the numbers present in the vector, the smallest number receiving the rank 1.
- `order()`: returns the indices of the vector in a sorted order (ascending or descending)

```
> x <- c(3, 1, 2, 5, 4)
> sort(x)

[1] 1 2 3 4 5

> rank(x)

[1] 3 1 2 5 4

> order(x)
```

Sorting data (II)

To sort the rows of a data frame according to the values of a specific column, use order()

```
> # convert titanic3 to a data frame
> titanic4 <- as.data.frame(titanic3)
> class(titanic4)

[1] "data.frame"

> head(titanic4[, c(3, 5, 9)])

                                name     age     fare
1    Allen, Miss. Elisabeth Walton 29.0000 211.3375
2   Allison, Master. Hudson Trevor  0.9167 151.5500
3     Allison, Miss. Helen Loraine  2.0000 151.5500
4 Allison, Mr. Hudson Joshua Crei 30.0000 151.5500
5 Allison, Mrs. Hudson J C (Bessi 25.0000 151.5500
6             Anderson, Mr. Harry 48.0000  26.5500
```

Sorting data (II)

To sort the rows of a data frame according to the values of a specific column, use order()

```
> titanic_sorted <- titanic4[order(titanic4$age), ]
> head(titanic_sorted[, c(3, 5, 9)])
```

```
                                  name    age    fare
764  Dean, Miss. Elizabeth Gladys \\"M 0.1667 20.5750
748    Danbom, Master. Gilbert Sigvard 0.3333 14.4000
1241   Thomas, Master. Assad Alexander 0.4167  8.5167
428         Hamalainen, Master. Viljo 0.6667 14.5000
658            Baclini, Miss. Eugenie 0.7500 19.2583
659      Baclini, Miss. Helene Barbara 0.7500 19.2583
```

Sorting data (III)

To sort the rows of a data frame according to the values of two (or more) specific columns, use `order()`. If values of first column are equal, the second column is used for sorting.

```
> titanic_sorted <- titanic4[order(
+   titanic4$age,
+   titanic4$fare
+ ), ]
> head(titanic_sorted[, c(3, 5, 9)])

                                       name    age    fare
764  Dean, Miss. Elizabeth Gladys \\"M 0.1667 20.5750
748    Danbom, Master. Gilbert Sigvard 0.3333 14.4000
1241   Thomas, Master. Assad Alexander 0.4167  8.5167
428          Hamalainen, Master. Viljo 0.6667 14.5000
1112    Peacock, Master. Alfred Edward 0.7500 13.7750
658             Baclini, Miss. Eugenie 0.7500 19.2583
```

Merging data

To combine two data frames by common column or row names (aka **merging**), use `merge()`

```
> authors <- data.frame(
+   name = c("Tukey", "Venables", "Tierney", "Ripley", "McN
+   nationality = c("US", "Australia", "US", "UK", "Austral
+   deceased = c("yes", rep("no", 4))
+ )
```

Merging data

To combine two data frames by common column or row names (aka **merging**), use `merge()`

```
> books <- data.frame(
+   name = c(
+     "Tukey", "Venables", "Tierney",
+     "Ripley", "Ripley", "McNeil", "R Core"
+   ),
+   title = c(
+     "Exploratory Data Analysis",
+     "Modern Applied Statistics ...",
+     "LISP-STAT",
+     "Spatial Statistics", "Stochastic Simulation",
+     "Interactive Data Analysis",
+     "An Introduction to R"
+   ),
+   other.author = c(
+     NA, "Ripley", NA, NA, NA, NA,
```

Merging data

To combine two data frames by common column or row names (aka **merging**), use merge()

```
> m0 <- merge(authors,books)
> m0
```

```
      name nationality deceased                          tit
1  McNeil   Australia       no     Interactive Data Analys
2   Ripley          UK       no            Spatial Statisti
3   Ripley          UK       no         Stochastic Simulati
4  Tierney          US       no                        LISP-ST
5    Tukey          US      yes     Exploratory Data Analys
6 Venables   Australia       no Modern Applied Statistics 
```

Reshaping data

For certain functions, the data needs to be in a specific format:

- *long*: repeated measurements in separate rows
- *wide*: repeated measurements in separate columns of the same row

The `reshape()` function of the **stats** package can reshape a data frame.

dplyr

- **dplyr** is an extensive R package that allows you to manipulate your data in a fast and easy way.
- It is not part of the standard R installation, and needs to be installed with `install.packages()`
- For a short introduction check: https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
- For a comparison with the base R functions: https://cran.r-project.org/web/packages/dplyr/vignettes/base.html

Summarizing data

- Data summary: `summary()`, `rowMeans()`, `colMeans()`
- Contingency tables: `table()`, `xtabs()`, `CrossTable` from **descr** package
- Summary by subgroups:
  - `aggregate()` and `tapply()`, `sapply()`, or `lapply()`
  - **doBy**, **Hmisc**, **compareGroups**, **dplyr**
- Graphical summary of data frames: `dfSummary()` in **summarytools**

EXERCISES

Let's do questions 10-16

Outline

If - else (I)

- R has a conditional construct: `if(){} else(){}`
- Depending on the outcome of a test, execute one or another statement

```
if(logical statement){

do this

} else {

do that

}
```

- Think carefully when making this construct. Using too many will make your code slow!
- Multiple `if - else` constructs can be nested within each other

If - else (II)

```
> x <- 10
> z <- if (x < 2) 4 else 3
> z

[1] 3
```

For - loop (I)

- A `for` loop is a very simple construct that lets you repeat a certain piece of code several times

```
for(loopIndex in loopVector){

repeat these functions

}
```

- *loopIndex*: this will change for each iteration of the loop
- *loopVector*: a vector specifying all values that the loopIndex will have.
- Multiple loops can be nested in each other

For - loop (II)

```
> for (i in 1:5) {
+ print(i)
+ }
```

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

For - loop (III)

```
> results <- rep(0, 2)
> results

[1] 0 0

> for (i in 1:length(results)) {
+ results[i] <- mean(A[i, ])
+ }
> results

[1] 5 6
```

Apply (I)

Functions from the `apply` family are convenient shorthands for `for` loops

```
apply(X, MARGIN, FUN, ...)
```

```
Arguments

X     an array, including a matrix

MARGIN   for a matrix 1 indicates rows, 2 indicates
columns

FUN the function to be applied
```

Apply (II)

```
> apply(A, 1, mean)

gene1 gene2
    5     6
```

Apply (III)

- `lapply()`: apply a function over a list or vector
- `sapply()`: similar to apply but more user-friendly if output can be coerced into a vector
- `tapply()`: can be used to split a vector in subgroups and apply a function for each subgroup
- `replicate()`: simpler version of `sapply` for the repeated evaluation of an expression
- `aggregate()`: extension of `tapply` for data frames

Apply (IV)

We will calculate for the titanic dataset the mean fare per passenger class

```
> with(titanic3, tapply(fare, pclass, mean, na.rm = TRUE))
       1        2        3
87.50899 21.17920 13.30289
```

EXERCISES

Let's do Questions 17, 18, and 19

Outline

Graphics

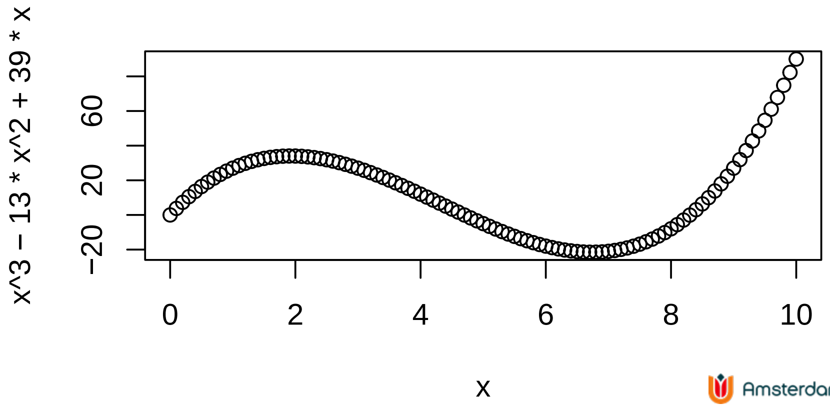
R has versatile tools for graphics. There are typically three steps to produce useful graphics:

- Creating the basic plot
- Enhancing the plot with labels, legends, colors, etc.
- Exporting the plot from R to use elsewhere

Basic plot (I)

It is straightforward to make a simple plot using functions from the **graphics** package (loaded by default):

```
> x <- (0:100) / 10
> plot(x, x^3 - 13 * x^2 + 39 * x)
```



Basic plot (II)

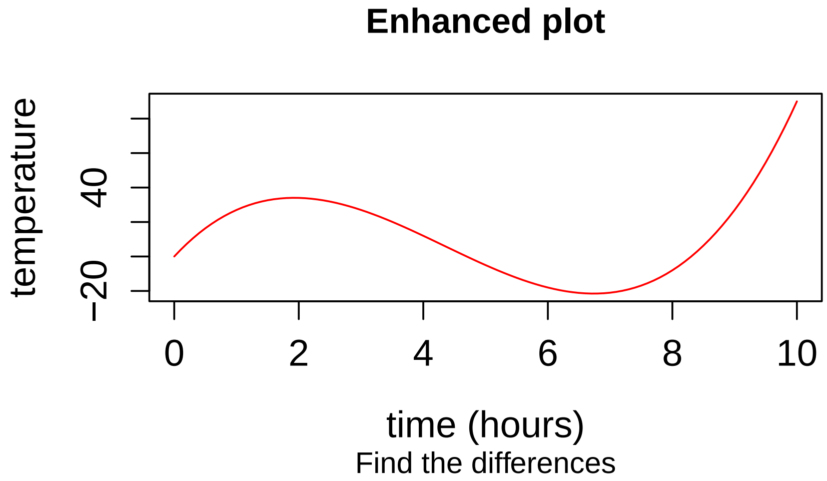
You can enhance/change the basic plot in multiple ways:

- `type`: specify the type of plot
- `main`: add an overall title for the plot
- `sub`: add a subtitle for the plot
- `xlab` and `ylab`: add a title for the axes
- `pch`: specify the symbol in case of points
- `col`: specify the color
- `cex.axis` and `cex.lab`: specify the size of the symbols and labels of the axes
- `par`: many more options for customization

Basic plot (III)

```
> plot(x, x^3 - 13 * x^2 + 39 * x,
+ type = "l",
+ col = "red",
+ xlab = "time (hours)",
+ ylab = "temperature",
+ main = "Enhanced plot",
+ sub = "Find the differences",
+ cex.axis = 1.25,
+ cex.lab = 1.25
+ )
```

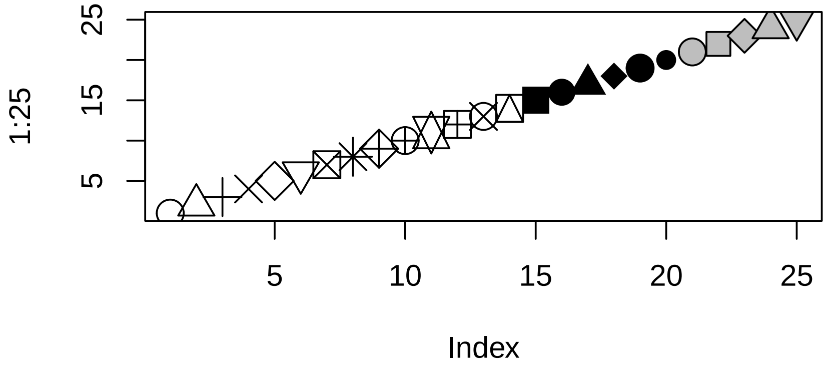
Basic plot (IV)



Basic plot (V)

There are 25 different plot symbols, see `?points`

```
> plot(1:25, pch = 1:25, cex = 2, bg = "grey")
```



Basic plot (VI)

The most commonly used options of `par`:

- `lwd` sets the line width
- `mfrow` and `mfcol` enable multiple plots in one figure
- `las` rotates axis symbols
- `mar` changes the margins of the figure
- `bg` changes the background colour

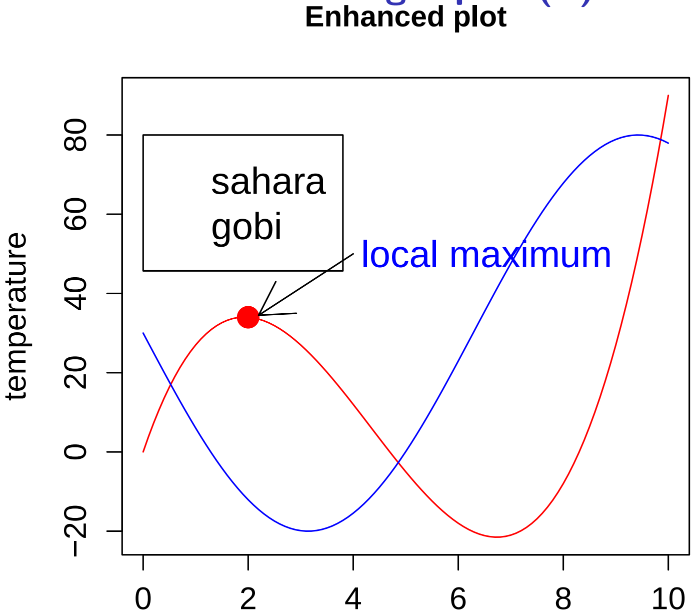
For more options see `?par`

Enhancing a plot (I)

With these functions, you can add extra elements to your plots:

- `points()`: add points
- `lines()`: add line
- `abline()`: add horizontal or vertical lines
- `arrows()`: add arrows
- `curve()`: add a curved line
- `rect()`: add a rectangle
- `text()`: add text
- `legend()`: add a legend
- `axis()`: add an axis

Enhancing a plot (II)



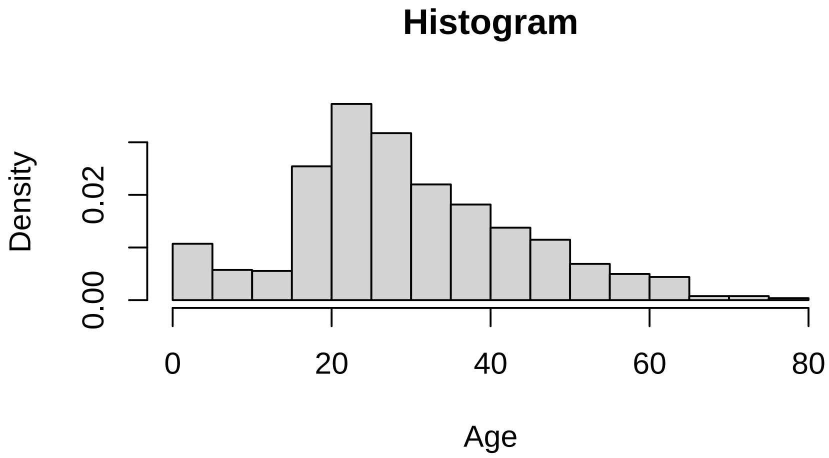
Histogram (I)

Use `hist()` for plotting histograms.

As always, see `?hist` for the many arguments of this function

```
> hist(titanic3$age,
+ breaks = 15, freq = FALSE,
+ main = "Histogram",
+ xlab = "Age"
+ )
```

Histogram (II)



Boxplot (I)

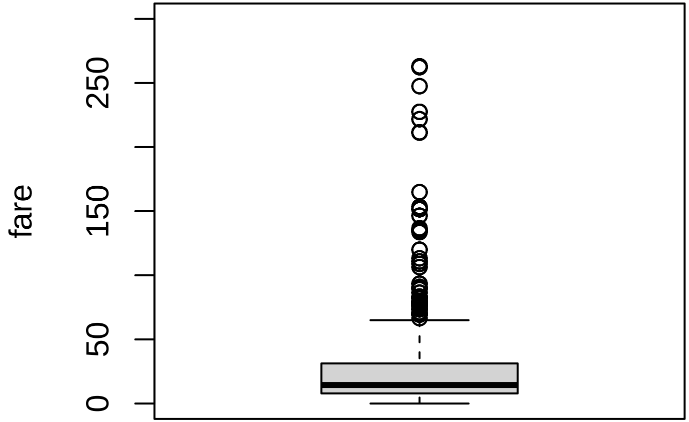
- Use `boxplot()` for plotting boxplots.
- For one group (a single vector):

```
> boxplot(titanic3$fare,
+ ylim = c(0, 300), ylab = "fare"
+ )
```

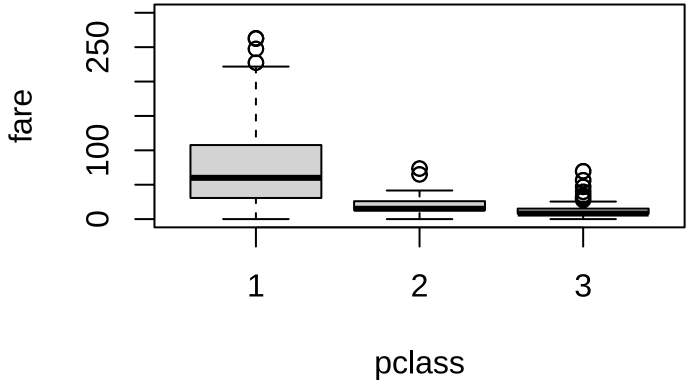
- For multiple groups:

```
> boxplot(fare ~ pclass,
+ data = titanic3, ylim = c(0, 300), ylab = "fare"
+ )
```

Boxplot (II)



Boxplot (III)



Barplot (I)

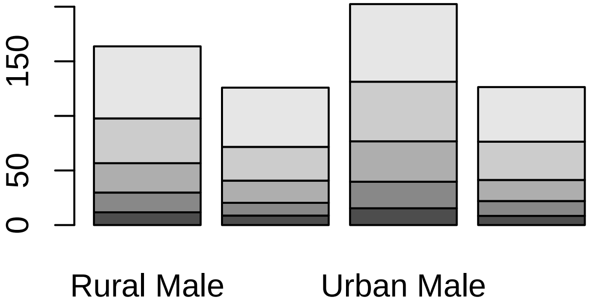
- Use `barplot()` for plotting barplots
- See `?barplot` for the many arguments of this function

```
> head(VADeaths)
      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
```

Barplot (II)

- Use `barplot()` for plotting barplots
- See `?barplot` for the many arguments of this function

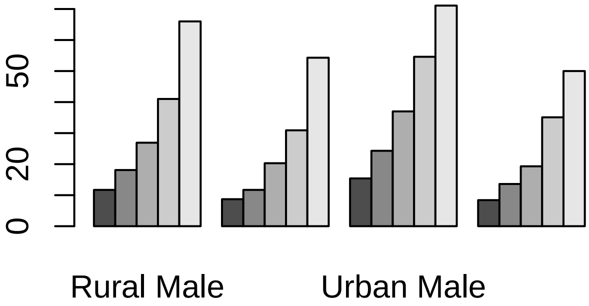
```
> barplot(VADeaths)
```



Barplot (III)

- Use `barplot()` for plotting barplots
- See `?barplot` for the many arguments of this function

```
> barplot(VADeaths, beside = TRUE)
```



Advanced R graphics

- Chapter 12 of“An Introduction to R” gives an introduction to **base** graphics
- **lattice**: very powerful for multipanel conditioning needs to be loaded first; `xyplot()` is the main function
- **ggplot2**: based on “the grammar of graphics” (see https://rstudio.github.io/cheatsheets/html/data-visualization.html)
- **ggvis**, **plotly**, **rCharts**, **Shiny**: interactive visualizations
- and in many more packages (**gplots**, **plotrix**, . . .)

Exporting figures

Two types of figure formats:

- Vector format (*pdf*, *eps*, *wmf*, *emf*)
  - digital image consisting of independent geometric objects (segments, polygons, curves, etc.)
  - can be enlarged without losing resolution
- Raster (*png*, *jpeg*, *tiff*)
  - rectangular grid of pixels, possibly with color
  - resolution impaired if image is enlarged

Graphics can be saved via the menu in the graphics/plots window, or a specific graphics file type can be created directly (`pdf()`, `win.metafile()`, `png()`) followed by `dev.off()`

Outline

Finding information

- `help()`
- `help.search()`: search the help system for documentation matching your search term
- `RSiteSearch()`: search CRAN (online) for documentation matching your search term
- package **sos**
- packages have sometimes *vignettes*, which introduce the package and their functions
- Google / Claude / . . .
- https://bioinformaticslaboratory.eu/gs-computing-in-r: Section Information on R

EXERCISES

Let's do Questions 20 - 24