

Introduction to R for flow cytometry data analysis Day 2

João Lourenço, Tania Wyss & Nadine Fournier

Translational Data Science – Facility

SIB Swiss Institute of Bioinformatics

With slides from Diana Marek, Thomas Junier, Wandrille Duchemin, Leonore Wigger

Outline

Day 2

06

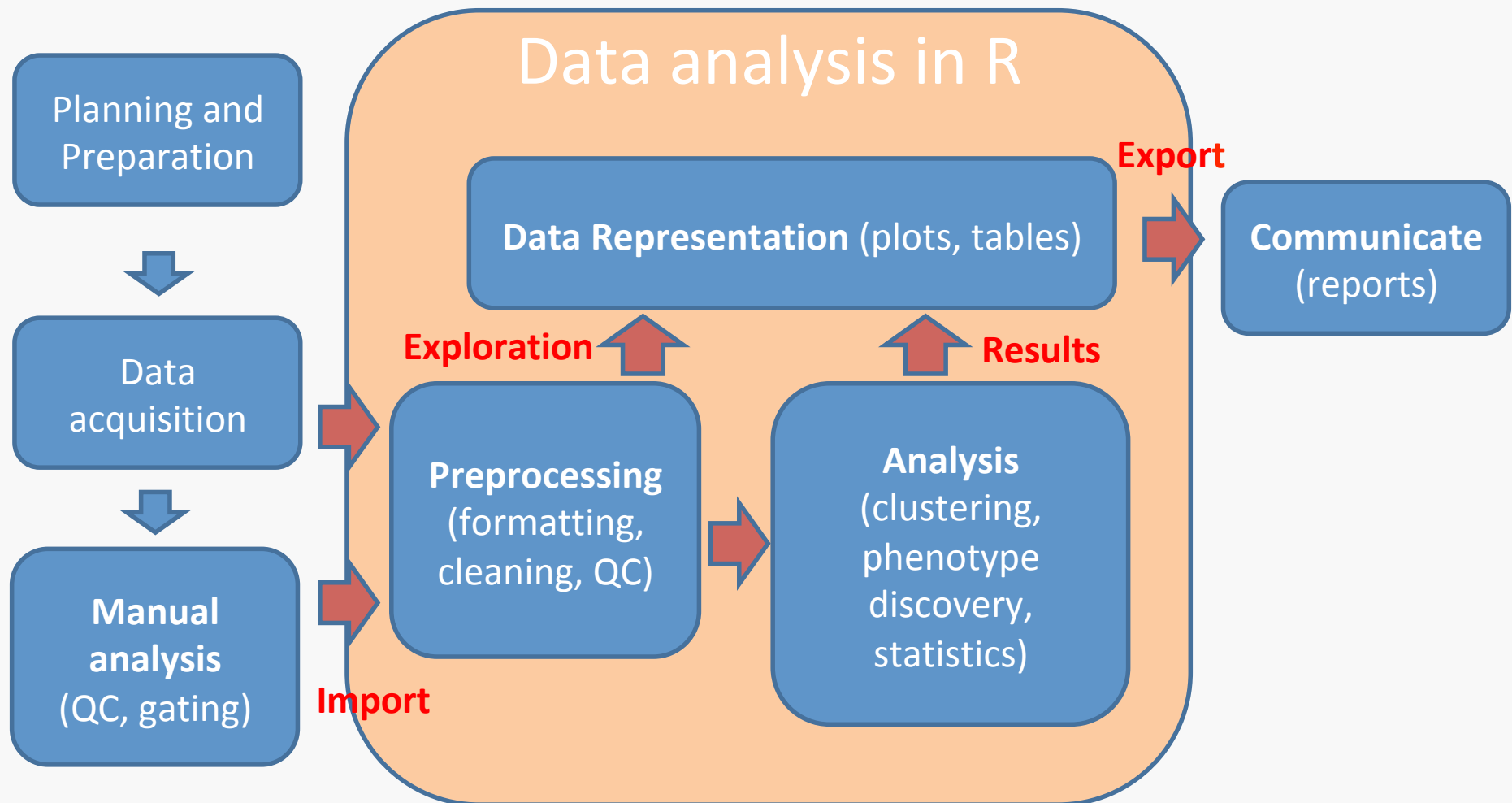
Building graphics in R

07

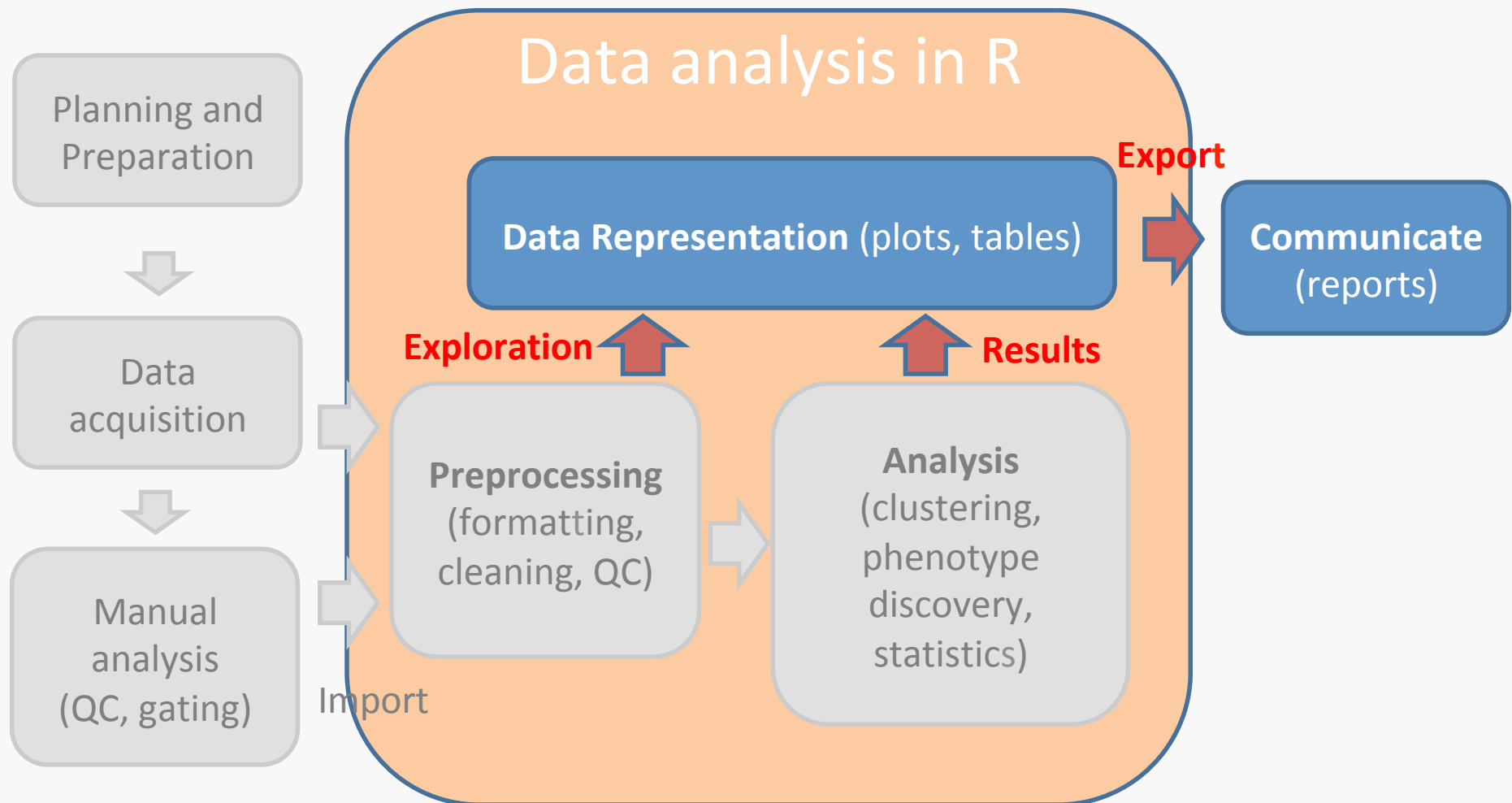
Starting with flow cytometry data in R (import and explore the data)

Examples and exercises are integrated in the chapters

Taking Advantage of R For Your Work



Taking Advantage of R For Your Work



06

Building graphics in R

R graphics

R is powerful for **plotting** graphs and figures. It provides several plotting systems:

- base widely used, comes with basic R installation
- ggplot2 widely used, based on the *Grammar of Graphics*

<http://vita.had.co.nz/papers/layered-grammar.pdf>

lattice mainly used for specialized needs, e.g. 3D plots

They have very different syntaxes, **cannot be mixed**, and need to be learned separately. This course introduces the **R base plotting system**.

R base plotting system

Plots are built up step by step with multiple function calls.

- **High-level graphics functions:**
 - Draw a new plot. Tailor its appearance with optional arguments.
- **Low-level graphics functions:**
 - Add graphical elements to an existing plot, piece by piece.

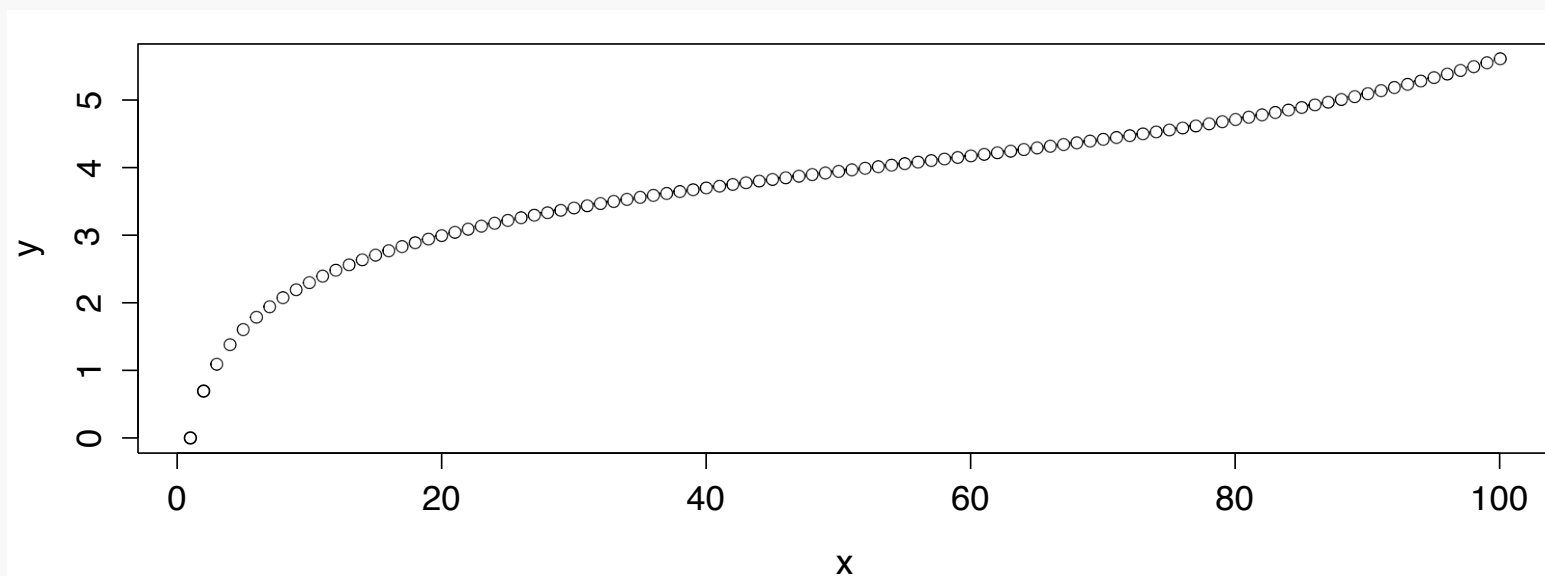
Plotting - the basics

- The generic function is **plot()**, which plots a variable y against a variable x .
- Takes the argument **type** to indicate the **type of plot** ("l" for lines, "p" for points, "b" for both, etc.). The default is **points**.

```
> x <- 1:100
```

```
> y <- log(x) + (x/100)^5
```

```
> plot(x,y) # equivalent to plot(x, y, type="p")
```



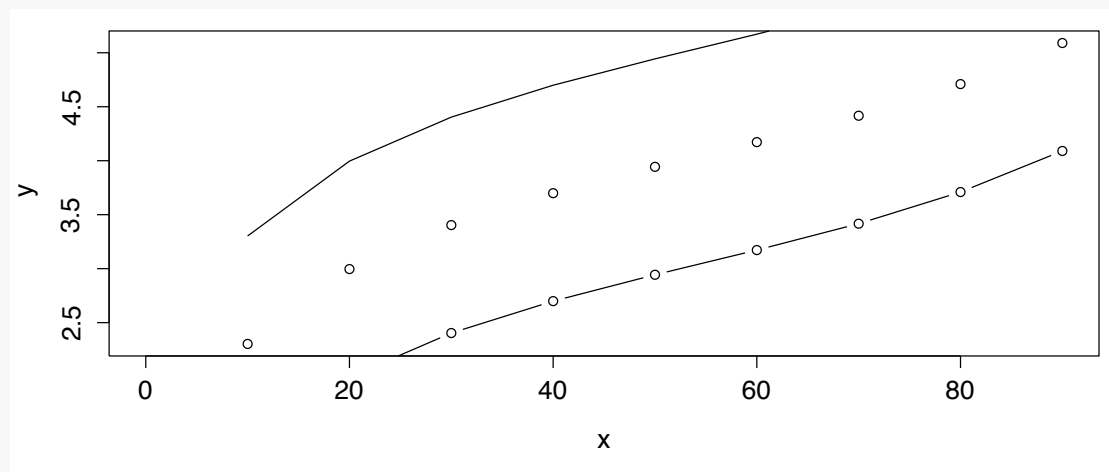
Adding elements to a plot

- Every time the `plot()` function is called, a new plot is created.
- In order to add more graphical elements to an already existing plot, low-level plotting commands can be used, such as:
 - `points()` to add points to an existing plot
 - `lines()` to add a line to an existing plot

The **type** argument can also be provided to those functions (e.g., "l" for lines, "p" for points and "b" for both). **Default for points(): "p", default for lines(): "l".**

```
>x <- seq(0,100, by=10)
>y <- log(x) + (x/100)^5
```

```
>plot(x,y)
>lines(x,y+1)
>points(x,y-1, type="b")
```



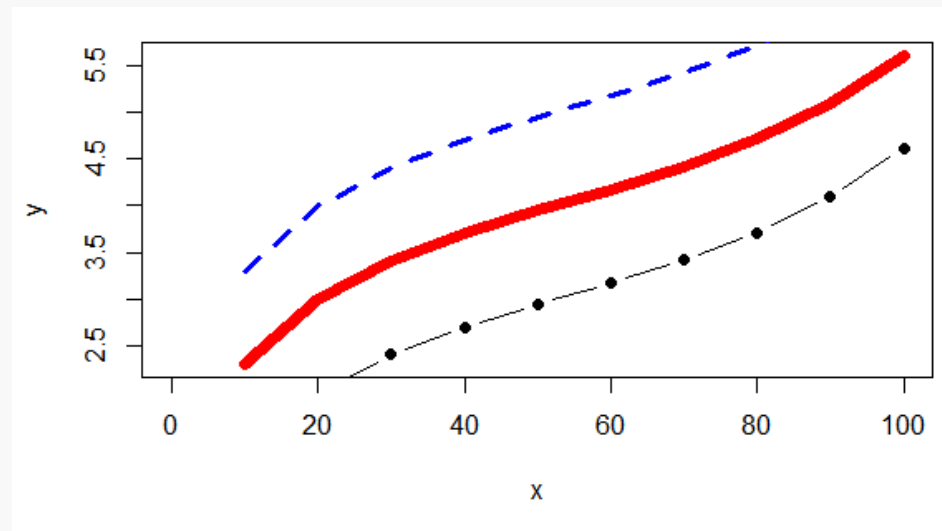
Customizing plots – Part 1

- **plot()**, **points()** and **lines()** all take customizing arguments, including:
 - **col** indicating the colour
 - **lwd** indicating the line width
 - **lty** indicating the line type
 - **pch** indicating the plotting character (symbol)







```
>plot(x, y, type="l", col="red",  
      lwd=7)
```

```
>lines(x, y+1, col="blue",  
       lty="dashed")
```

```
>points(x, y-1, type="b",  
        pch=19)
```





























R line types, to use with lty

	lty=1 or 'solid'
	lty=2 or 'dashed'
	lty=3 or 'dotted'
	lty=4 or 'dotdash'
	lty=5 or 'longdash'
	lty=6 or 'twodash'

(Do `help(par)` and search for “lty”)

R plotting characters, to use with pch

0 	1 	2 	3 	4 	
5 	6 	7 	8 	9 	
10 	11 	12 	13 	14 	
15 	16 	17 	18 	19 	
20 	21 	22 	23 	24 	25 

(Do [help\(points\)](#) and scroll 2-3 screens)

R color names

- R has 657 built-in color names
- Can be used in plotting functions
- Chart shows a subset
- `colors()` will output a list of all color names

white	aliceblue	antiquewhite	antiquewhite1	antiquewhite2
antiquewhite3	antiquewhite4	aquamarine	aquamarine1	aquamarine2
aquamarine3	aquamarine4	azure	azure1	azure2
azure3	azure4	beige	bisque	bisque1
bisque2	bisque3	bisque4		blanchedalmond
blue	blue1	blue2	blue3	blue4
blueviolet	brown	brown1	brown2	brown3
brown4	burlywood	burlywood1	burlywood2	burlywood3
burlywood4	cadetblue	cadetblue1	cadetblue2	cadetblue3
cadetblue4	chartreuse	chartreuse1	chartreuse2	chartreuse3
chartreuse4	chocolate	chocolate1	chocolate2	chocolate3
chocolate4	coral	coral1	coral2	coral3
coral4	cornflowerblue	cornsilk	cornsilk1	cornsilk2
cornsilk3	cornsilk4	cyan	cyan1	cyan2
cyan3	cyan4	darkblue	darkcyan	darkgoldenrod
darkgoldenrod1	darkgoldenrod2	darkgoldenrod3	darkgoldenrod4	darkgray
darkgreen	darkgrey	darkkhaki	darkmagenta	darkolivegreen
darkolivegreen1	darkolivegreen2	darkolivegreen3	darkolivegreen4	darkorange
darkorange1	darkorange2	darkorange3	darkorange4	darkorchid
darkorchid1	darkorchid2	darkorchid3	darkorchid4	darkred
darksalmon	darkseagreen	darkseagreen1	darkseagreen2	darkseagreen3
darkseagreen4	darkslateblue	darkslategray	darkslategray1	darkslategray2
darkslategray3	darkslategray4	darkslategrey	darkturquoise	darkviolet
deeppink	deeppink1	deeppink2	deeppink3	deeppink4
deepskyblue	deepskyblue1	deepskyblue2	deepskyblue3	deepskyblue4

See R color cheat sheet for the full color chart and other ways to define colors

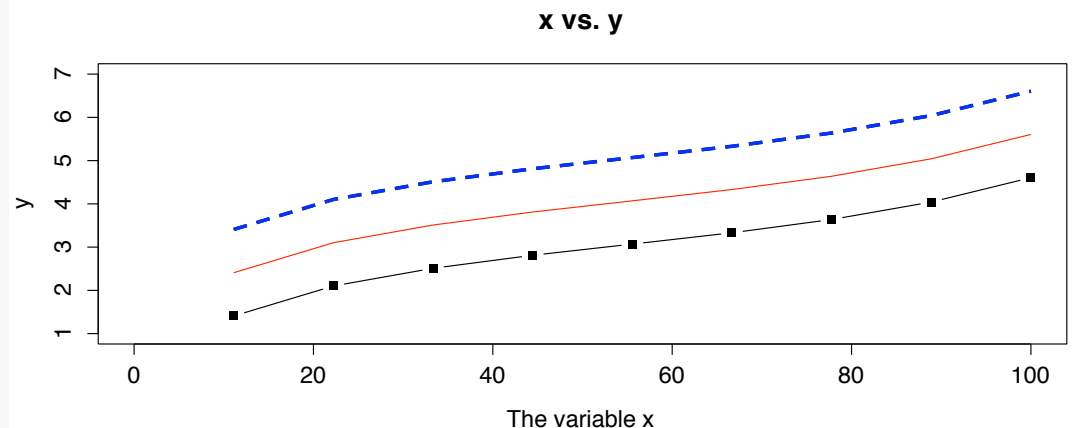
<https://r-charts.com/colors/>

www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf

Customizing plots – Part 2

- The **plot()** command takes further arguments to customize the plotting area:
 - **xlim** and **ylim** to set the limits on the x- and y-axis, respectively
 - **xlab** and **ylab** to set the labels for the x- and y-axis, respectively
 - **main** to set a title

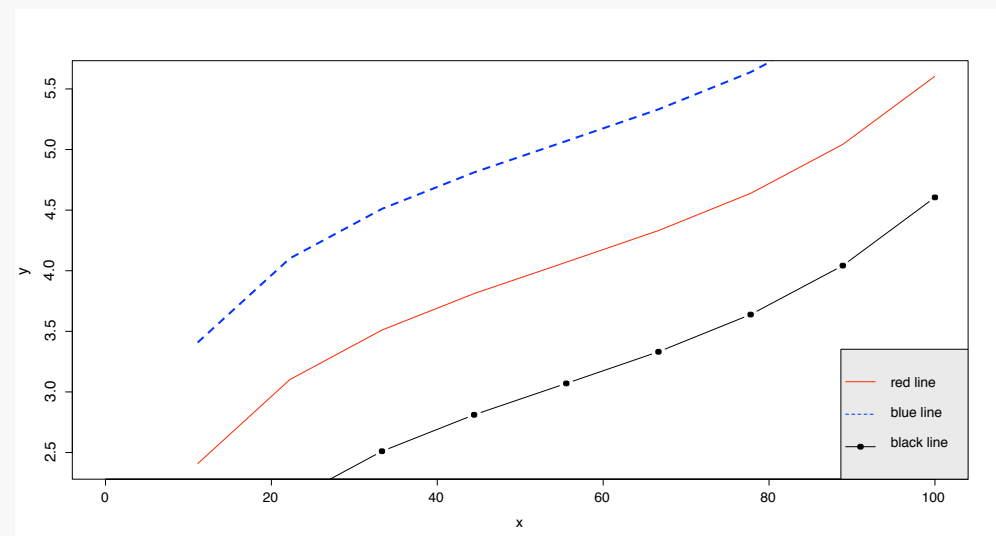
```
> x <- seq(0, 100, length.out=10)
> y <- log(x) + (x/100)^5
> plot(x,y, type="l", col="red", ylim=c(1,7),
  xlab="The variable x", main="x vs. y" )
> lines(x, y+1, lwd=3,lty="dashed", col="blue")
> points(x, y-1, type="b", pch=15)
```



Customizing plots – Part 3

- The **legend()** command can be used to **add legends** to plots:
 - **x** , **y** to set the numeric coordinates for positioning the legend.
 - x can be used by itself with a keyword for legend position: "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "center"
 - **legend** to set the text to appear in the legend
 - **col** to set the colours of points or lines
 - **lty** and **lwd** to set the line types and widths for lines appearing in the legend
 - **pch** to set the plotting symbols appearing in the legend
 - **bty** for box type around the legend ("o" for box, "n" for no box)
 - **bg** for background color

```
> legend(x="bottomright",  
legend=c("red line",  
"blue line", "black line"),  
lty=c(1,2,1), pch=c(NA,NA,19),  
col=c("red", "blue", "black"),  
bg="gray90")
```



How to get data for practicing and playing – Part 1

R contains many practice data sets (data frames), great for trying out plotting functions.

Display names of available data sets

```
>data() #data sets in standard packages  
>data(package = .packages(all.available = TRUE)) #data sets  
in all installed packages
```

Load and use a data set

```
>data(iris) #load the iris data (overwrite existing variable)  
>?iris      #get information about the iris data  
>head(iris) #display top few lines of the iris data frame
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

How to get data for practicing and playing – Part 2

R can easily simulate data drawn from a given distribution. The function `rnorm()` generates normally distributed data.

Example:

```
>rnorm(10) #numeric vector with 10 random
           values #drawn from normal
           distribution, #mean=0, sd=1
           (function defaults)
```

```
[1] 1.1053564 0.7937635 0.2743762 0.3574477 -0.7677099
[2] 0.5838973 0.6616164 0.1203090 -0.4060265 0.2778585
```

How to get data for practicing and playing – Part 2

R can easily simulate data drawn from a given distribution. The function `rnorm()` generates normally distributed data.

Example:

```
>rnorm(10) #numeric vector with 10 values
           #drawn from normal distribution,
           #mean=0, sd=1 (function defaults)
[1]  1.1053564  0.7937635  0.2743762  0.3574477 -0.7677099
[2]  0.5838973  0.6616164  0.1203090 -0.4060265  0.2778585
```

```
>rnorm(10, mean=10, sd=2) #customized mean and sd
[1]  6.253392  9.527140  9.398857 11.932284 11.472909
[2] 10.714245  7.656026 11.302829  9.332930 10.264157
```

If you want data from other distributions than normal:

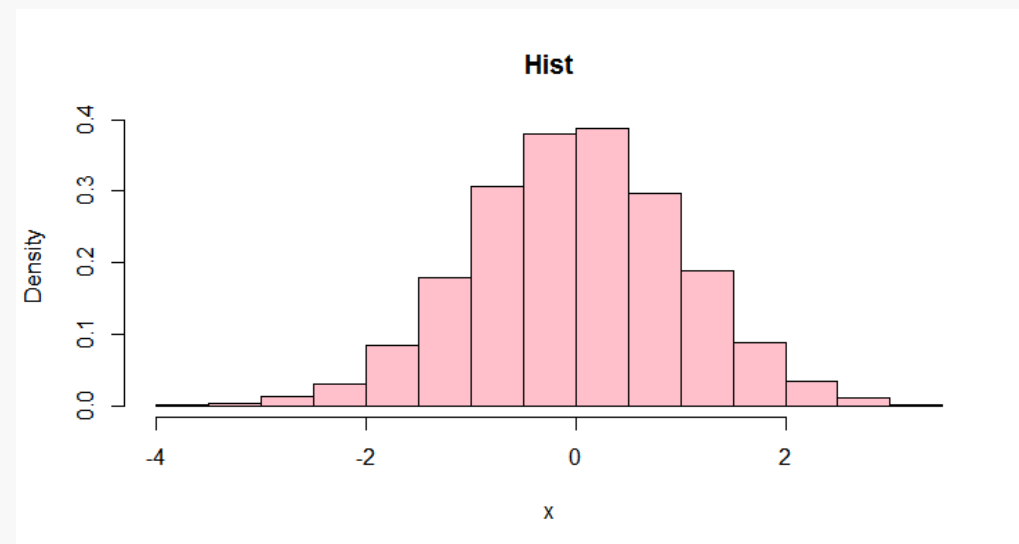
`rpois()` for poisson, `rbinom()` for binomial (see R help)

sd: standard deviation

The hist() function

- The function **hist()** produces a histogram, which counts the number of observations that fall into different ranges (bins)
- Rough visual representation of the distribution of the data.
 - **x** vector of data values for which the histogram will be constructed
 - **breaks** either a vector indicating breakpoints between histogram bins, or a single number for the number of bins (used as suggestion)
 - **freq** logical. If TRUE, cell height represents counts per bin. If FALSE, cell height is the fraction of values that fall into each bin (probability mass).

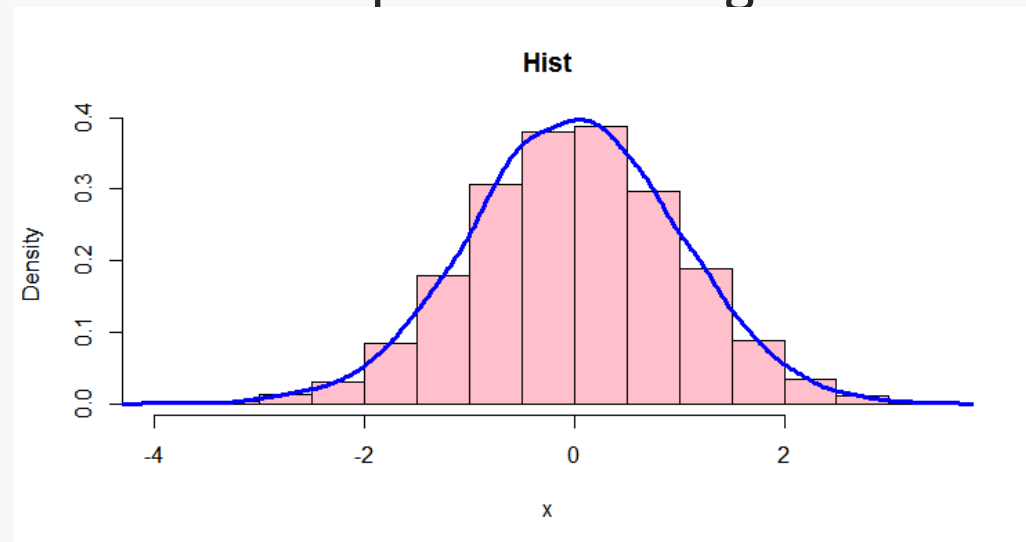
```
> x <- rnorm(10000)
> hist(x, breaks=20,
      freq=FALSE,
      main="Hist",
      col="pink")
```



The hist() and density() functions

- To add a smooth line to a histogram, use `density()`, which computes estimates of the probability density (kernel density estimates).
- This works as a complementary representation of the histogram **only** when `freq = FALSE`
- The line produced by `density()` often reflects the distribution better than a histogram.
- Use `lines()` to plot the result as a line on top of the histogram.

```
> x <- rnorm(10000)
> hist(x, freq=FALSE,
      main="Hist",
      col="pink")
> lines(density(x),
      col="blue", lwd=3)
```



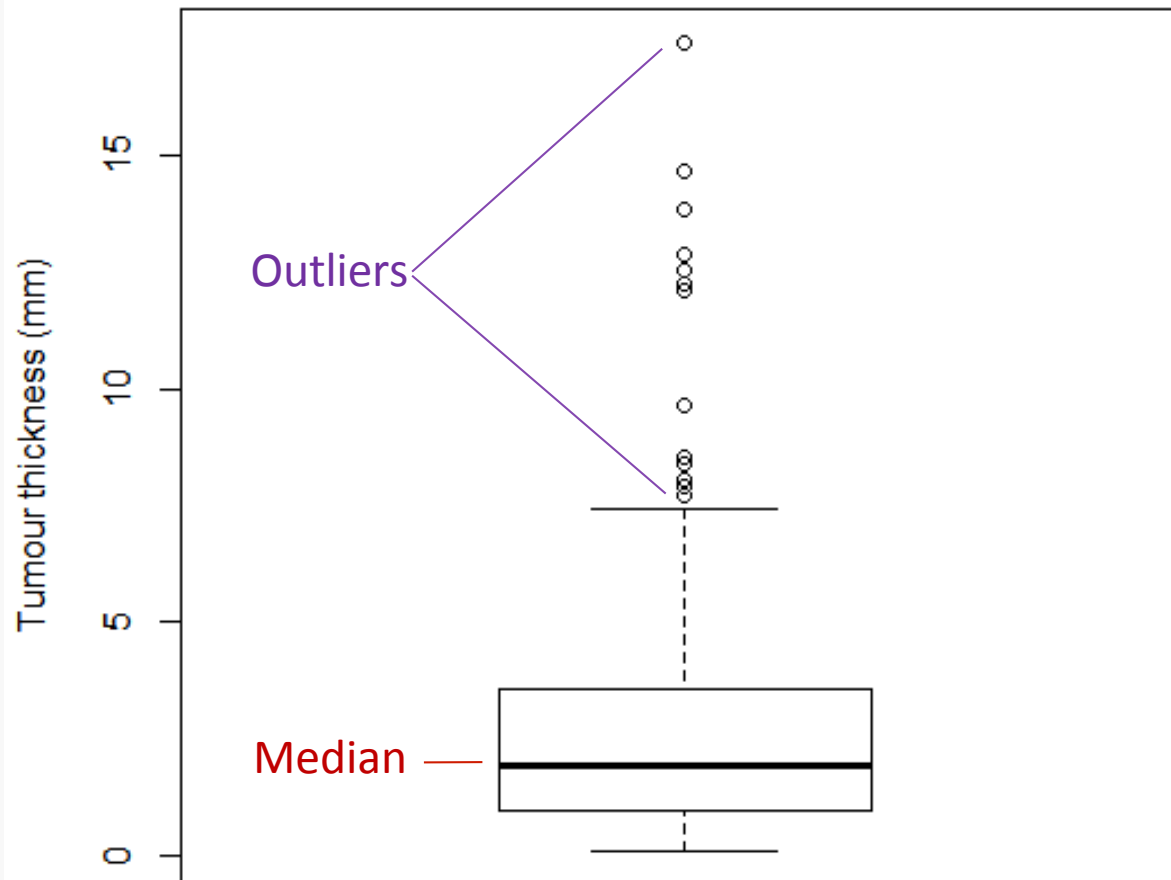
The boxplot() function

Convenient way of depicting the **spread of numerical data**

Box: Interquartile range (IQR), contains 50% of points

Whiskers: Extend from box, indicate variability outside upper and lower quartiles

Outliers: May be plotted as individual points



Example:

Melanoma thickness (mm)
in 205 patients

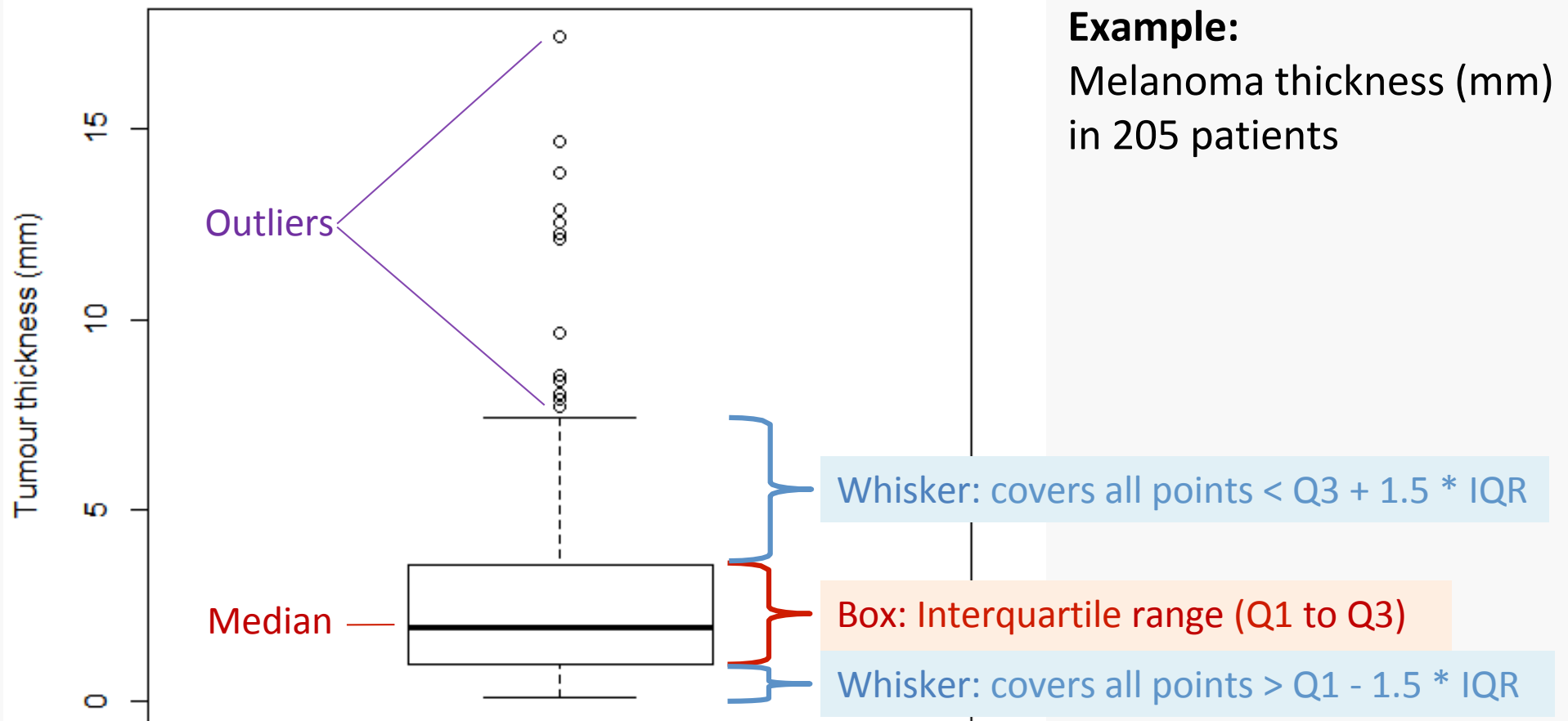
The boxplot() function

Convenient way of depicting the spread of numerical data

Box: Interquartile range (IQR), contains 50% of points

Whiskers: Extend from box, indicate variability outside upper and lower quartiles

Outliers: May be plotted as individual points



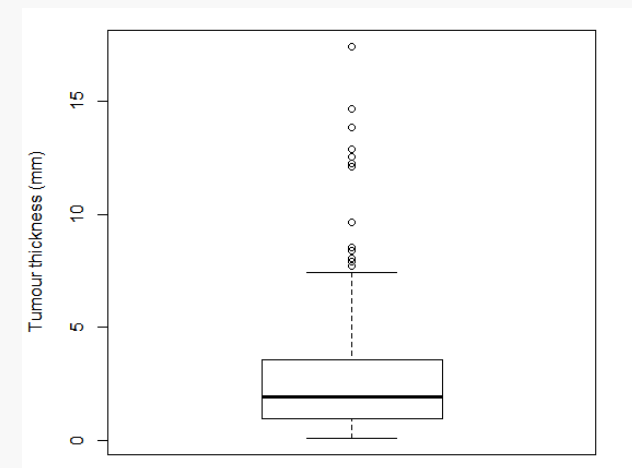
Boxplot: data and plotting code

```
>library(MASS)
>data(Melanoma) #Data from MASS package. 205 patients in
Denmark with melanoma
```

```
>head(Melanoma) #look inside the data set
  time status sex age year thickness ulcer
```

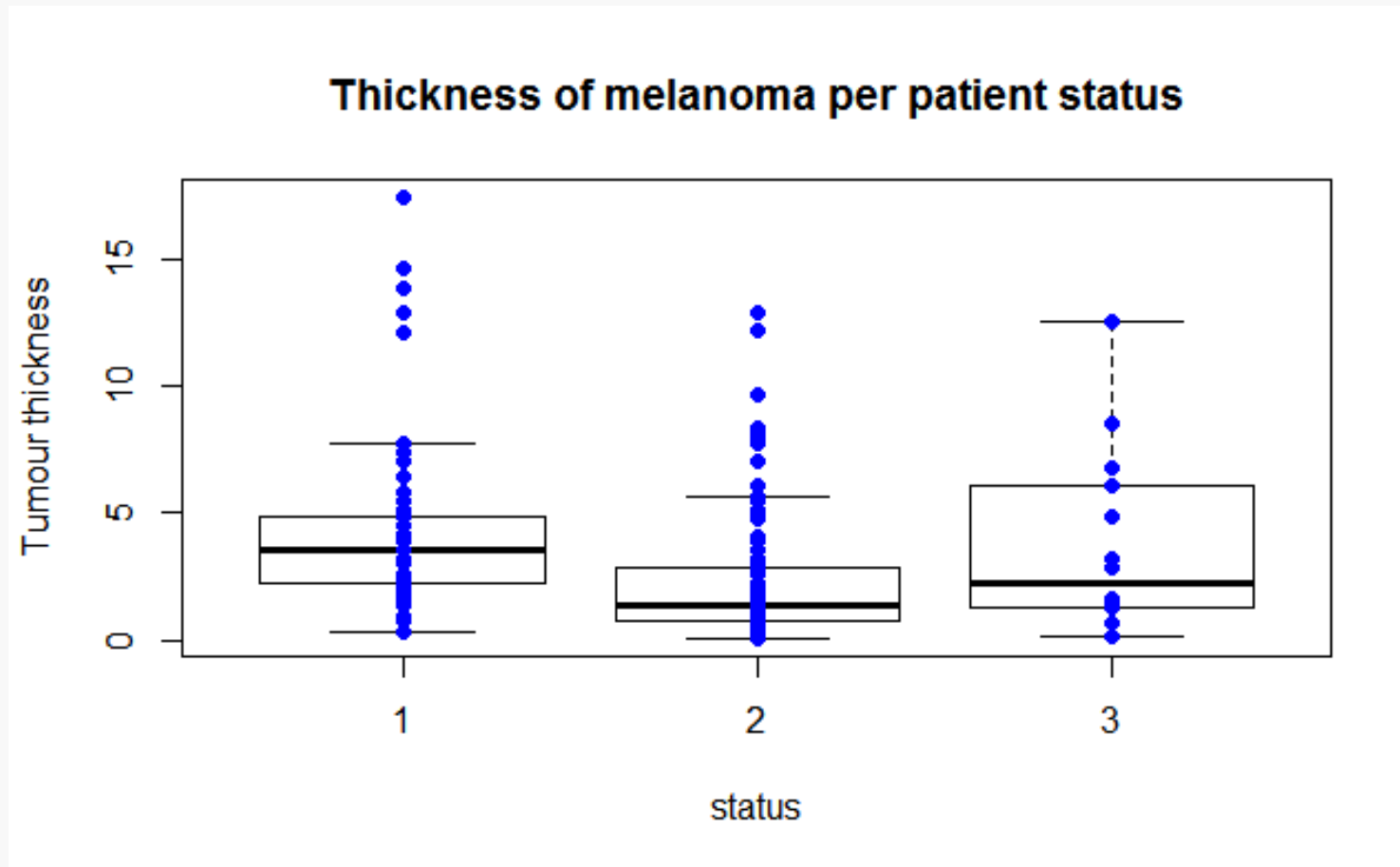
1	10	3	1	76	1972	6.76	1
2	30	3	1	56	1968	0.65	0
3	35	2	1	41	1977	1.34	0
4	99	3	0	71	1968	2.90	0
5	185	1	1	52	1965	12.08	1
6	204	1	1	28	1971	4.84	1

```
>boxplot(Melanoma$thickness,
          ylab="Tumour thickness (mm)",
          col="white")
```



More boxplots

- Make separate boxplots for **subgroups** of data
- Plot **individual data points** as an overlay of the boxplots.



status: 1 died from melanoma, 2 alive, 3 dead from other causes

More boxplots: data preparation

```
#check if the grouping variable is a factor (it is not!)
```

```
>str(Melanoma)
```

```
'data.frame': 205 obs. of 10 variables:  
 $ time      : int  10 30 35 99 185 204 210 232 232 279 ...  
 $ status    : int  3 3 2 3 1 1 1 3 1 1 ...  
 $ sex       : int  1 1 1 0 1 1 1 0 1 0 ...  
 $ age       : int  76 56 41 71 52 28 77 60 49 68 ...  
 $ year      : int  1972 1968 1977 1968 1965 1971 1972 1974  
 $ thickness: num  6.76 0.65 1.34 2.9 12.08 ...  
 $ ulcer     : int  1 0 0 0 1 1 1 1 1 1
```

```
#coerce the grouping variable to factor
```

```
>Melanoma$status <- factor(Melanoma$status)
```

More boxplots: plotting code

Method 1: Data subsets

```
>boxplot(Melanoma$thickness[Melanoma$status=="1"],  
         Melanoma$thickness[Melanoma$status=="2"],  
         Melanoma$thickness[Melanoma$status=="3"],  
         main="Thickness of melanoma per patient status",  
         xlab="status", ylab="Tumour thickness",  
         names=c("1", "2", "3"))
```

```
>points(Melanoma$status, Melanoma$thickness,  
        col="blue", pch=19) #adds the actual data points to the plot
```

Method 2: Formulas

```
>boxplot(thickness ~ status, data=Melanoma,  
         main="Thickness of melanoma per patient status",  
         xlab="status", ylab="Tumour thickness")
```

```
>points(thickness ~ status, data=Melanoma,  
        col="blue", pch=19) #adds the actual data points to the plot
```

The `abline()` function

`abline()` adds one or more straight lines through the current plot – vertical, horizontal or sloped.

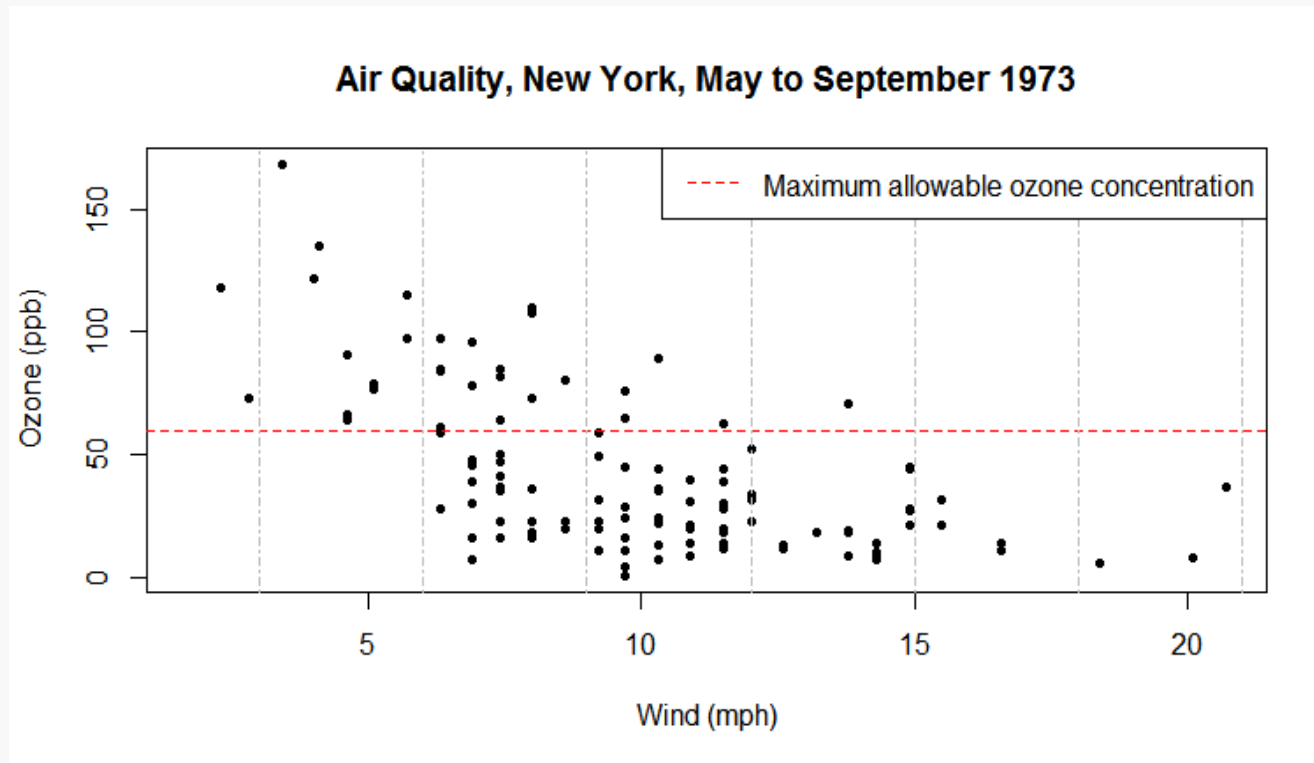
Useful for

- showing boundaries and cutoffs
- fitting straight trend lines through the data (cf. `lm()`)

Arguments:

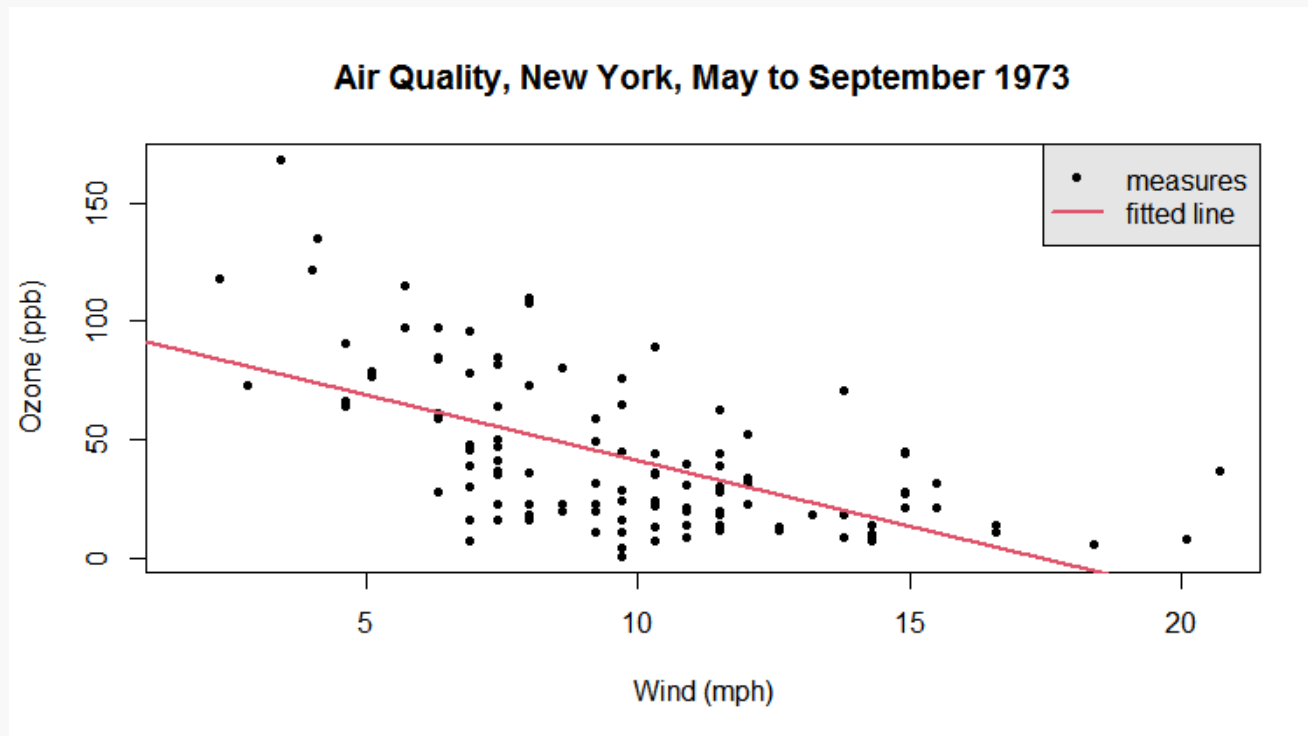
- `abline(v=c(...))`: add vertical line(s) at the given x value(s)
- `abline(h=c(...))`: add horizontal line(s) at the given y value(s)
- `abline(a= ,b=)`: add an affine line with intercept a and slope b
- `abline(reg=lm(...))`: add a trend line from a linear regression equivalent to `abline(lm(...))`

Example 1: Horizontal and vertical lines



- > `data(airquality) # Daily measurements, New York, May-Sept. 1973`
- > `plot(airquality$wind, airquality$Ozone, pch=20, xlab="wind (mph)", ylab="Ozone (ppb)")`
- > `abline(h=60, col="red", lty="dashed")`
- > `abline(v=seq(3,21,3), col="grey", lty="dotdash")`
- > `legend("topright", "Maximum allowable ozone concentration", col="red", lty="dashed")`

Example 2: Fitting a trend line



- > `plot(airquality$wind, airquality$Ozone, pch=20, xlab= "wind (mph)", ylab="Ozone (ppb)")`

- > `abline(lm(airquality$Ozone ~ airquality$wind), col=2, lwd=2)`

- > `legend("topright", legend= c("measures","fitted line"), pch= c(20, NA), lty = c(0, 1), lwd=c(NA, 2), col = c(1, 2), bg = "gray90")`

Draftsman's or Pairs Scatter Plots

- If x is a matrix or a data frame, **pairs()** draws all possible bivariate plots between the columns of x.

```
> data(iris)
> pairs(iris[,1:4], main="Edgar Anderson's Iris Data",
       pch=21, bg=c("red", "green3", "blue")[iris$Species])
```

bg:

color fill of circles with black outline ●

Colors:

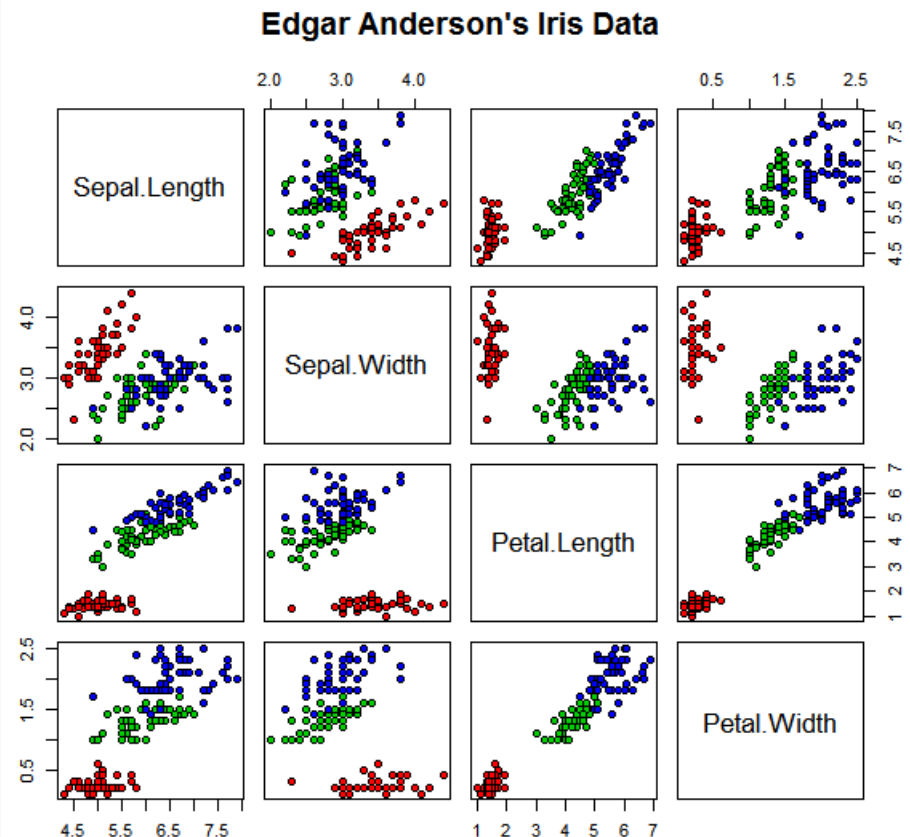
setosa in red

versicolor in green

virginica in blue

Why?

Each of the 3 listed colors gets attributed to one of the factors following the order of the levels



Let's practice - 7

Import the clinical data from the file `clinical_data_mod.csv`. This file contains the same data as `clinical_data.csv` and in addition, one more column.

- 1) Run `str()` to check your data frame: did it load correctly?
- 2) Convert `gender` and `response_to_treatment` to factor variables.
- 3) Plot a **histogram** of patient weight and customize it with colours, labels, title.
- 4) Make a **scatter plot** of height against patient weights using the function `plot()`.

Function arguments:

- use solid circles as plotting symbol
- add a title
- customize the axis labels (“Weight [kg]”, “Height [m]”)
- color the points **by gender**.

Add a **legend** for the gender. Fit a **trend line** using the function `abline()`.

- 5) Create a new column called “BMI” and compute the BMI of patients from their weight and height
- 6) Make **boxplots** of BMI from patients with different responses to treatment. Customize with title, labels, colors. Add points to the boxplots to show the individual values.
- 7) *Optional*: Repeat 6 with `stage`, instead of response to treatment. **Hint**: what kind of variable is `stage`?

Permanent Graphic Changes (I)

- The function `par()` allows to change the default values of many plotting parameters. All future calls to graphics functions will be affected.

- Example 1: set plotting colors and symbols

```
>par(col="red", pch=15)
```

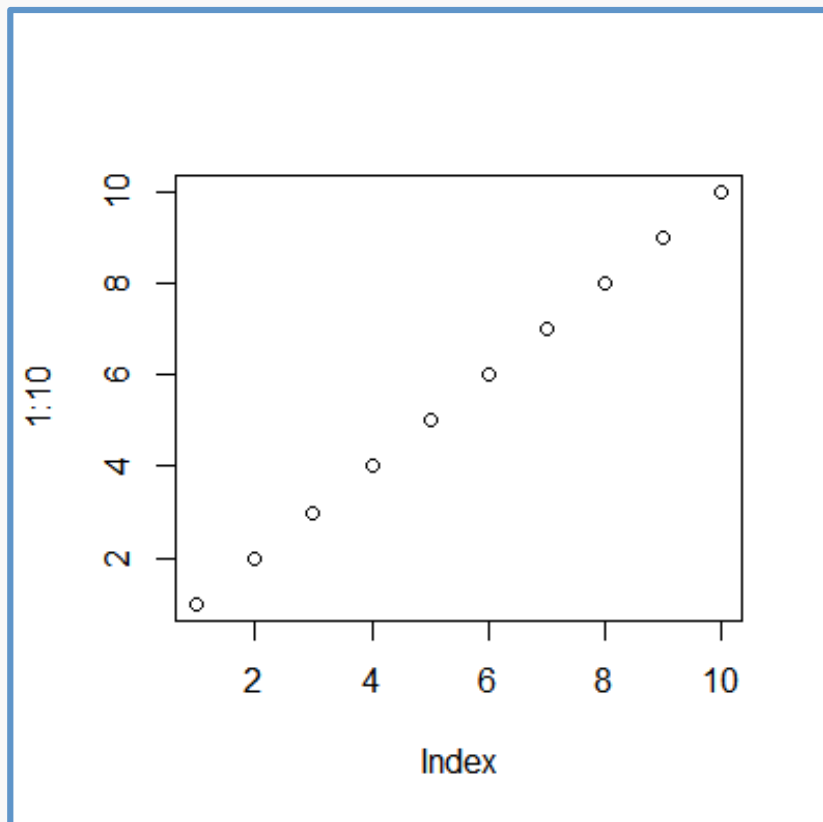
- Example 2: set margin widths for subsequent plots

- `mar` sets plot margins in number of lines
- use vectors of 4 values (`c(0,1,1,2)`) for the bottom, left, top, and right margins

```
>par(mar=c(5.1,4.1,4.1,2.1))      #set margins
```

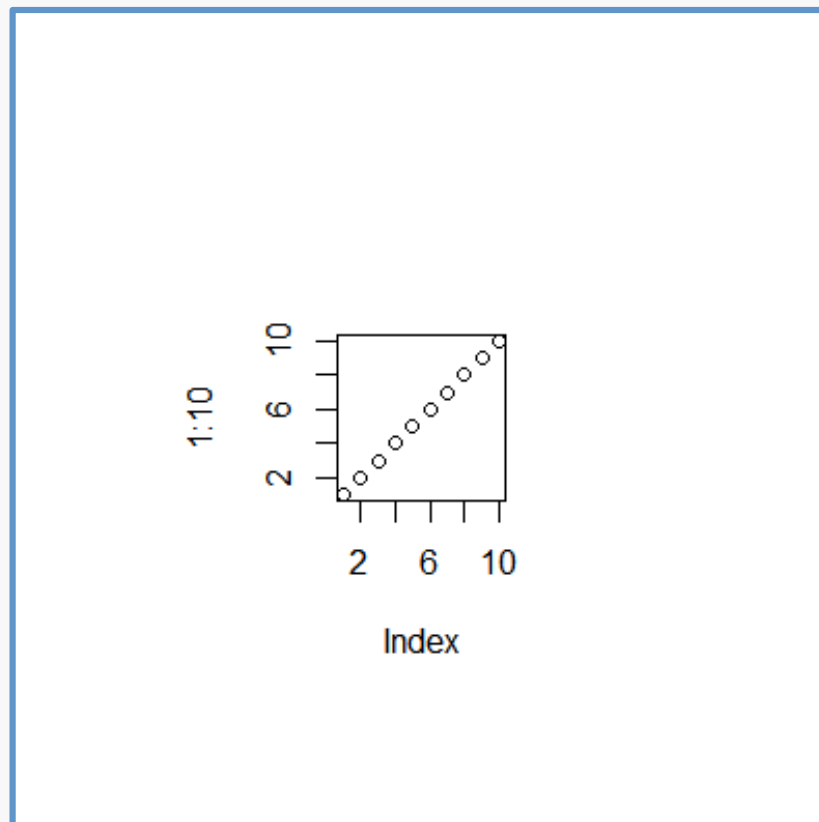
Normal Margins (bottom, left, top right):

```
>par(mar=c(5.1,4.1,4.1,2.1))  
>plot(1:10)
```



Wide Margins (bottom, left, top, right):

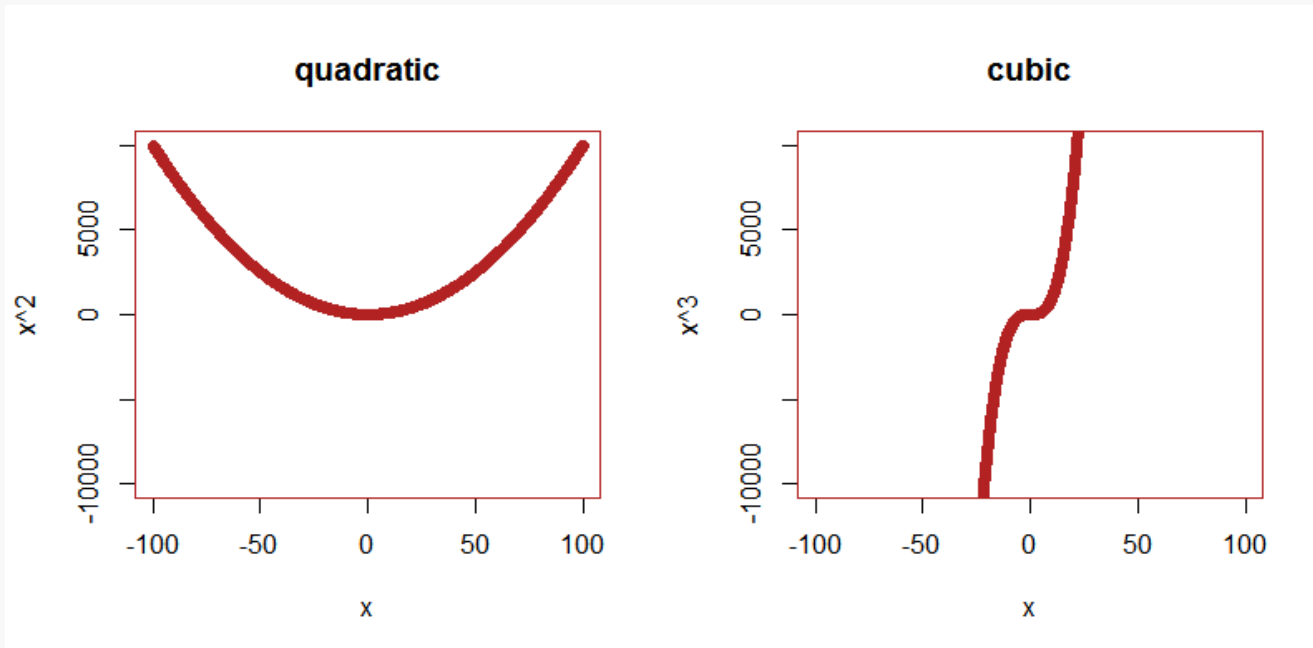
```
>par(mar=c(8.1,8.1,8.1,8.1))  
>plot(1:10)
```



Permanent Graphic Changes (II)

- Example 3: Generate multi-panel figures using `par()`
- **mfrow (or mfcoll)**: A vector of the form `c(nr, nc)`. Subsequent figures will be drawn in an `nr-by-nc` array by `rows` (or `columns`, respectively).

```
> par(mfrow=c(1,2),col="firebrick", pch=19) #1x2 plot array
> x <- seq(-100, 100, 0.1)
> plot(x, y=x^2, ylim = c(-10000,10000), main="quadratic")
> plot(x, y=x^3, ylim = c(-10000,10000), main="cubic")
```

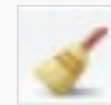


Current settings of par()

- Calling `par()` without parameters displays current settings
- If you changed nothing, all parameters are at default values

Resetting par()

- `par()` is automatically reset to defaults when you:
 - **Restart R** or close/switch **Rstudio projects**
 - Run `dev.off()`, which closes the most recent plot/plotting device
 - Run `graphics.off()`, which closes plots/plotting devices
 - **In RStudio**, **clear** all plots using the broom icon



Saving figures to files

- By default, R plots all graphics to the screen (i.e Plots window).
- R offers functions to export graphics to many formats (pdf, postscript, bmp, jpeg, png, tiff). The basic concept is to redirect the graphics output to a different “device”.
- Use `pdf()` to start redirection to a .pdf file, `png()` for a .png file, etc.
- Use `dev.off()` to close the redirection.

```
pdf(file="quadratic_cubic.pdf", width=7, height=4,
    paper="a4")
par(mfrow=c(1,2), col="firebrick", pch=19)
x <- seq(-100, 100, 0.1)
plot(x, y=x^2, ylim=c(-10000,10000), main="quadratic")
plot(x, y=x^3, ylim=c(-10000,10000), main="cubic")
dev.off()
```

- Alternatively you can use the RStudio interface:
 - Plots > *Export > Save as Image* (PNG, JPEG, TIFF, BMP, ...)
 - Plots > *Save as PDF*.

Arguments to graphics export functions

- Use correct file extension:
 - `postscript(file="a_name.ps", ...)`
 - `pdf(file="...pdf", ...)`
 - `jpeg(file="...jpg", ...)`
 - `png(file="...png", ...)`
- Each graphics device has a specific set of arguments that dictate characteristics of the output file : `height=`, `width=`, `horizontal=`, `res=`, `paper=`, `pointsize=`
- For `png`, `jpeg`, `tiff` (raster formats), the width and height of the graphics are given in pixels.
- For `pdf` and `postscript` (vector formats), the width and height of the graphics region are given in inches. Default values are 7. (Tip: A4 = 8.3" x 11.7"; set the width and height a little smaller for printing to A4 size).
- Only `pdf()` and `postscript` have an argument "paper". This can be set to common paper formats (`paper="a4"` for A4 in portrait orientation, `paper="a4r"` for A4 in landscape orientation).

Choosing an image file format

Raster graphics (png, tiff, jpeg):

- file sizes depend on the image size (number of pixels)
- once created, stretching the image leads to poor quality

Vector graphics (pdf, ps, eps, svg):

- file sizes depend on the number of drawing actions (e.g. number of points, lines,...)
- all elements can be scaled as desired

Embedding image files in MS Office documents (Word, PowerPoint):

- In Windows, png and tiff work best, pdf can get blurry.
- In macOS, pdf works well.
- Can also export plot from RStudio to clipboard, then paste.

Publication-quality figures:

- Vector graphics (pdf, eps) tend to be easier to adapt as they can be resized

File size tip: when a large number of points is plotted, pdfs can become large in file size and slow to display. When this is an issue, consider png.

Let's practice – 8

- 1) Make a multi-panel figure with the **four graphics (3, 4, 6 and 7 from previous exercise) on one page**, exporting the figure to a **pdf** file with paper size A4. Set width and height arguments in the call to pdf() to make it look nice.
- 2) **Optional:** Export the histogram (3 from previous exercise) to a **png** file. Set width and height arguments in the call to png() to make it look nice.

ggplot2

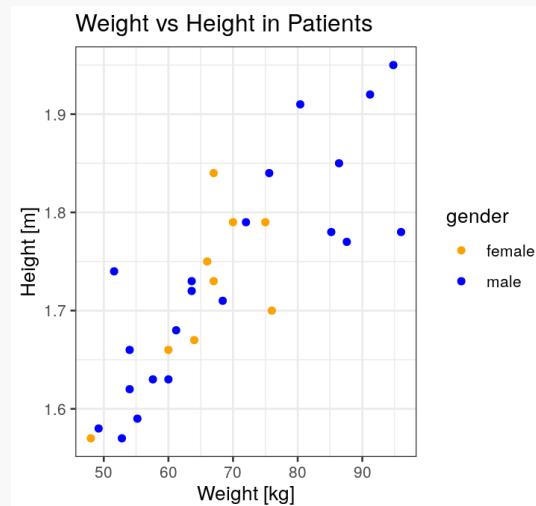
- Multi-panel figures: save the ggplot to an object, then display

```
p1 <- ggplot(dataset, aes(x, y, color=fact)) +  
  geom_type() +  
  additional_layers()
```

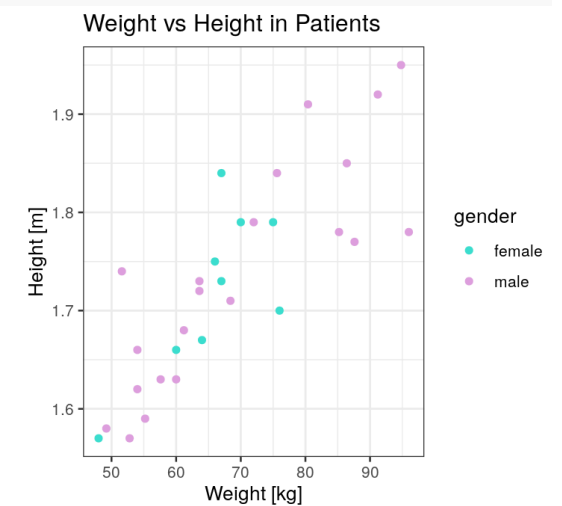
```
p2 <- ggplot(dataset, aes(x, y, color=fact)) +  
  geom_type() +  
  additional_layers()
```

```
# install.packages("cowplot")  
library(cowplot)  
plot_grid(p1, p2, nrow=1)
```

p1



p2

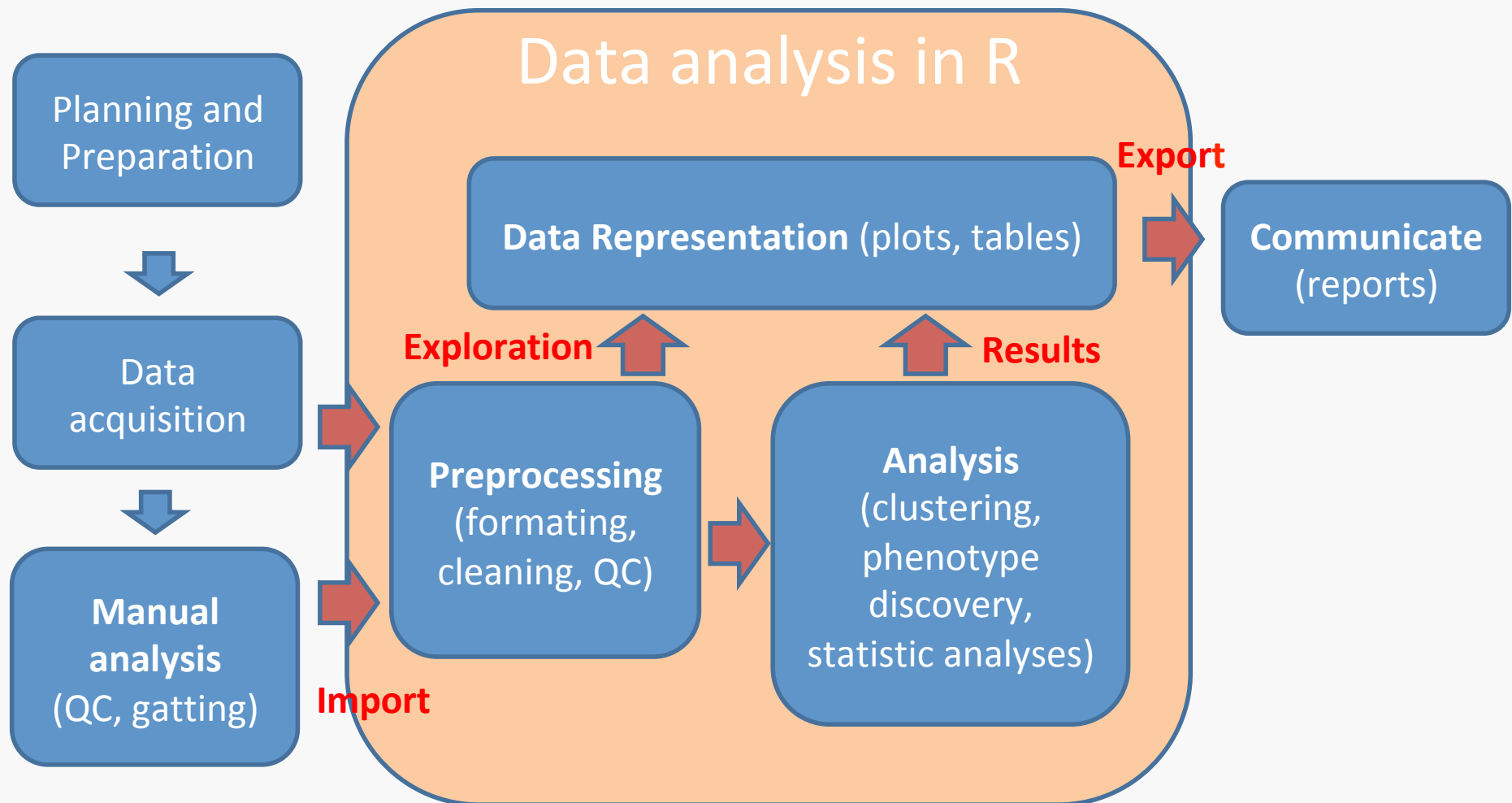


In a nutshell

- Introduction to **high-level** and **low-level plotting functions** in R
 - `plot()`, `lines()`, `points()`, `hist()`, `barplot()`, `boxplot()` ...
- **Customization** of plotting functions
 - Colours, line types, line widths, plotting characters...
 - Titles, labels, legend...
- **Permanent graphic changes**
- **Exporting graphics** in different formats

A nice resource you may want to use as inspiration and reference for plotting:
<https://r-graph-gallery.com/index.html>

Taking Advantage of R For Your Work



Taking Advantage of R For Your Work

