Swiss Institute of Bioinformatics

First Steps with R in Life Sciences: Graphics

With slides from: Diana Marek, Geoffrey Fucile, Alex Smith, Linda Dib, Leonore Wigger, Wandrille Duchemin

elixir SWITZERLAND

Exam – 0.5 ECTS

Take-home exam: data analysis tasks, available on course page.

Exam is graded as "pass" or "fail".

- Submit analysis to your teacher **in one week (ask for their e-mail).**
- If you pass, you will receive a **certificate of achievement** from the SIB Training Team, which you can submit to your educational institution.
- If you don't take the exam, you will receive a **certificate of attendance**.

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, implements the *Grammar of Graphics* (Wilkinson, Springer (2005)))
- 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.

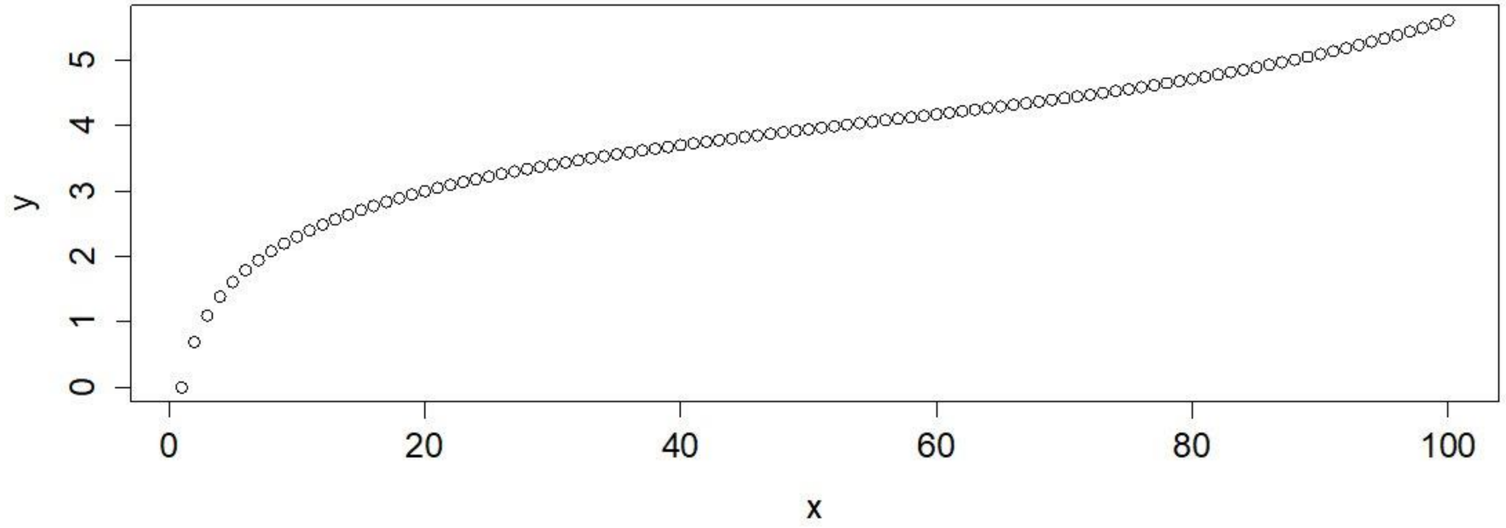
Low-level graphics functions: add graphical elements to an existing plot, piece by piece.

Plotting – the basics

Generic function **plot():**

- plots a variable *y* against a variable *x*.
- argument **type** : 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

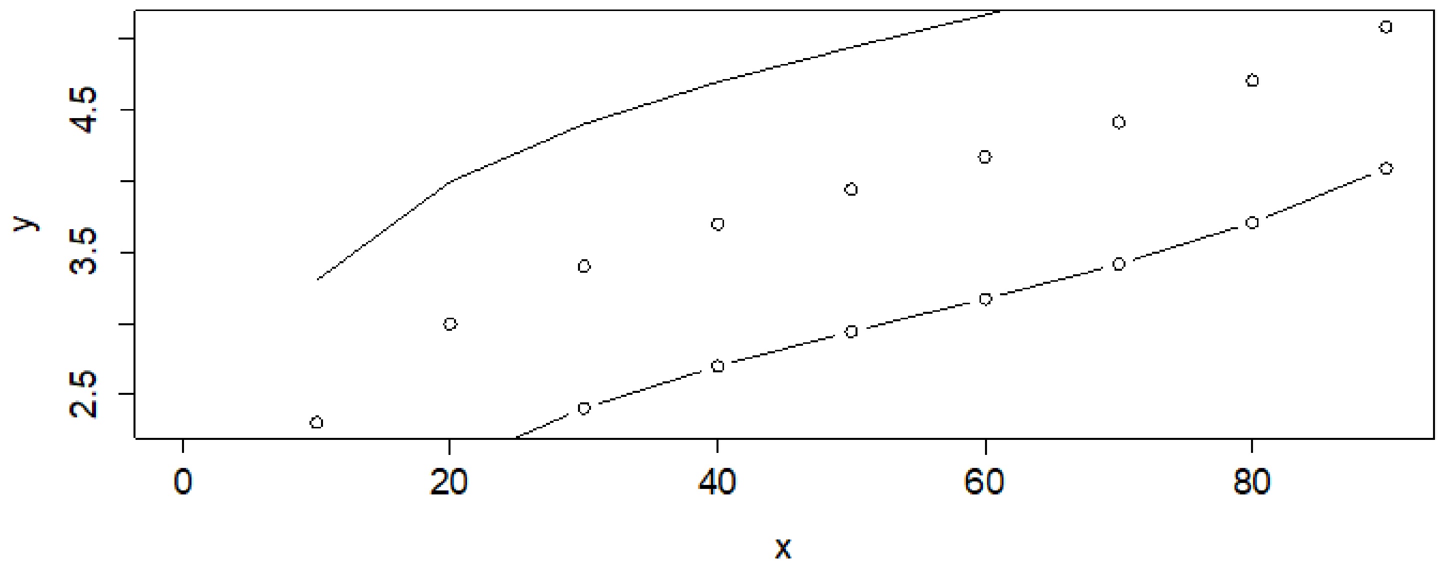
a new plot is created every time the plot() function is called .

low-level plotting commands: add graphical elements to an existing plot

- **points()** to add points to an existing plot
- **lines()** to add a line to an existing plot
- These function also have the **type** argument (e.g., "**l**" for lines, "**p**" for points and "**b**" for both).

```
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")
```

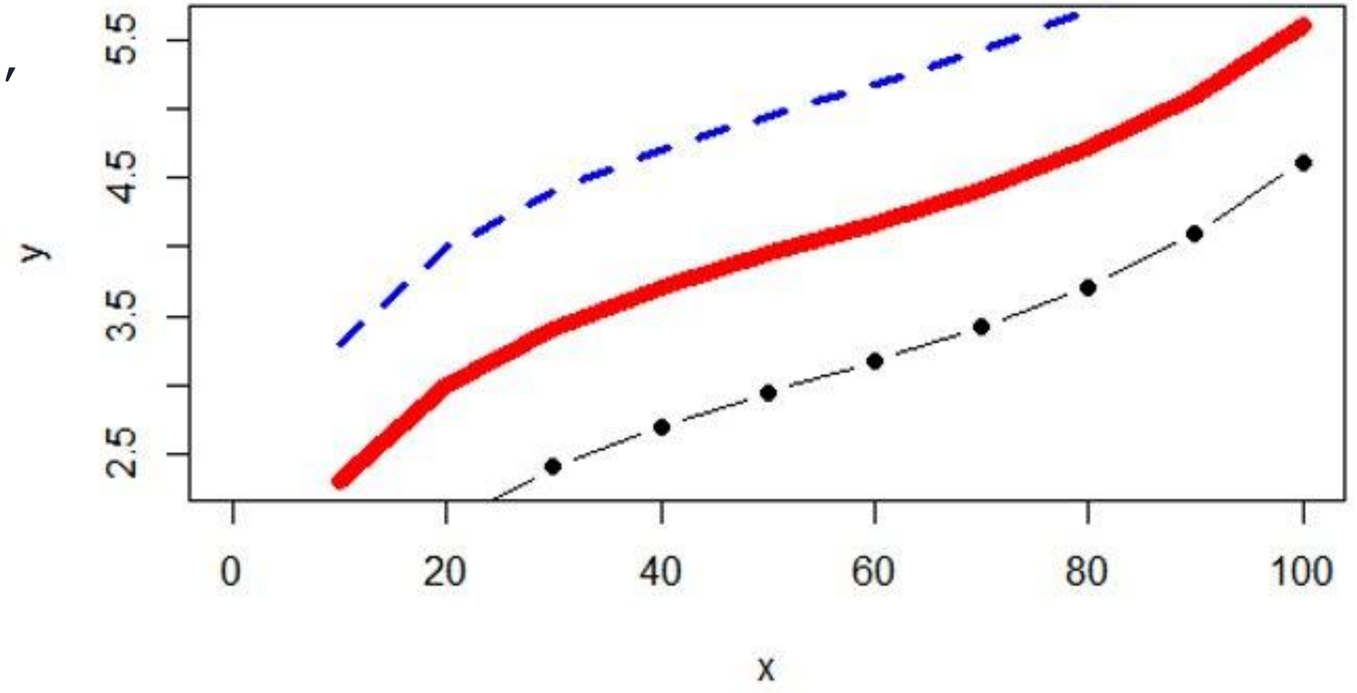


Customizing plots (I)

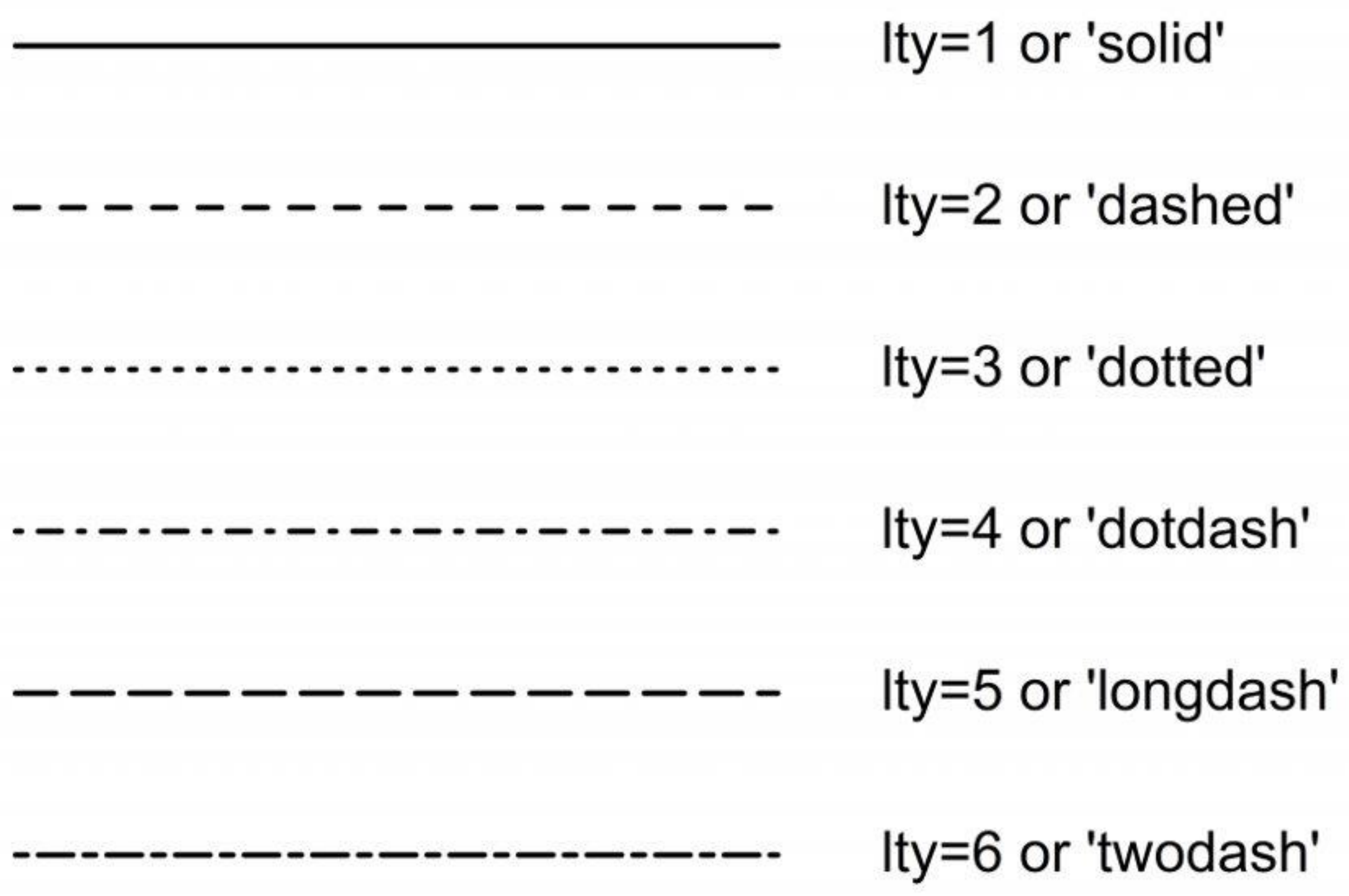
plot(), **points()** and **lines()** all take customizing arguments, including:

- **col** indicating the colour
- **lwd** indicating the line width
- **lty** indicating the line type
- **pch** indicating the plotting character (symbol)

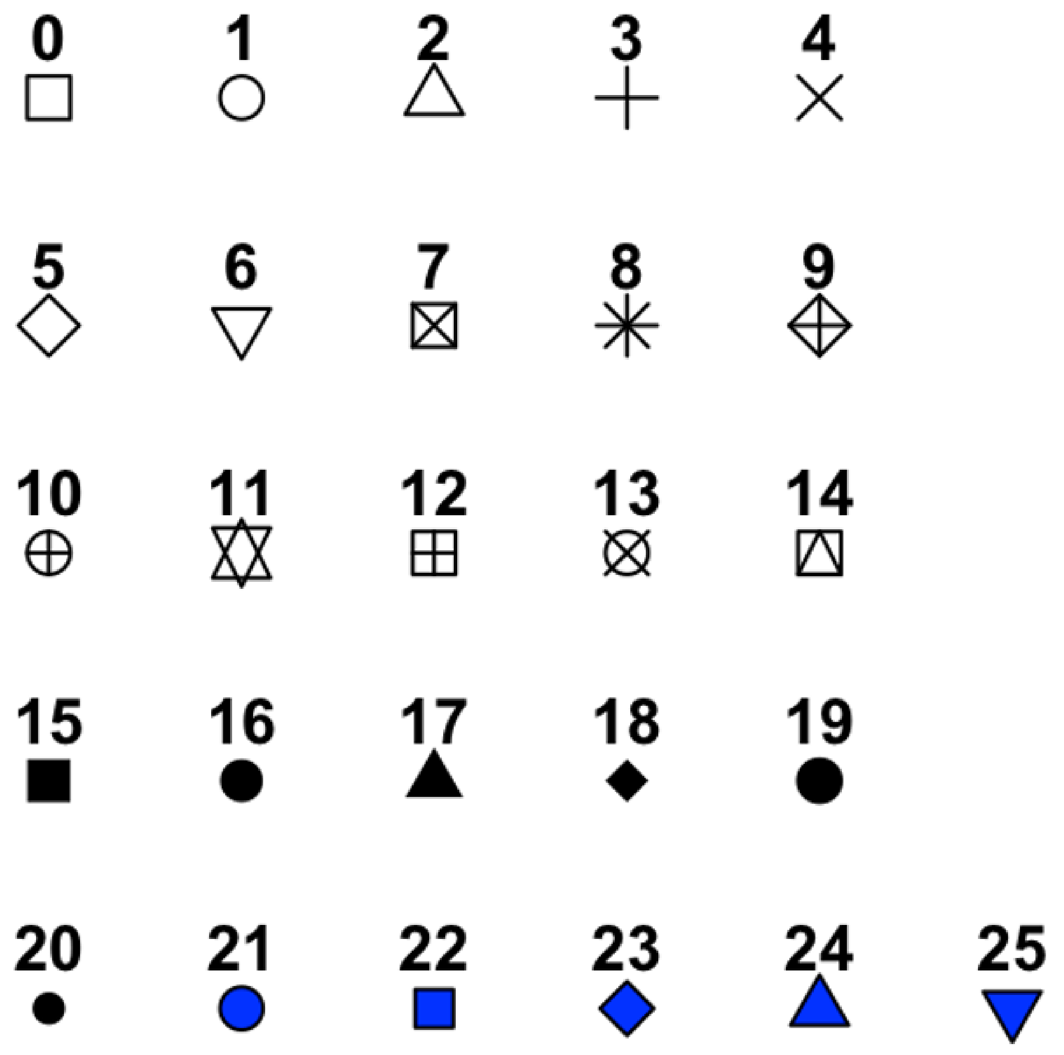
```
plot(x, y, type="l", col="red",
      lwd=7)
lines(x, y+1, col="blue",
       lty="dashed")
points(x, y-1, type="b",
        pch=19)
```



R line types, to use with lty



R plotting characters, to use with pch

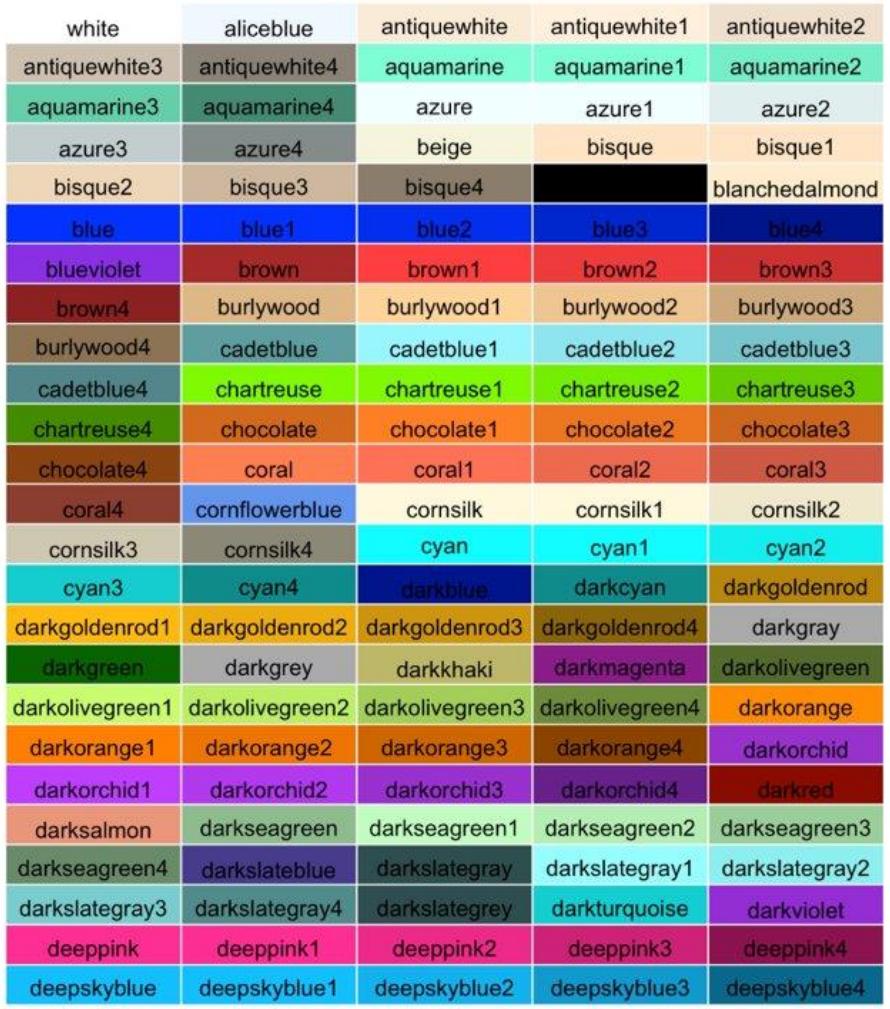


R color names

657 built-in color names

`colors()` to get a full list

Here is a subset -->



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

also possible to define colors:

- Hex codes
- RGB numbers
- Numbers 1 to 8

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

Customizing plots (II)

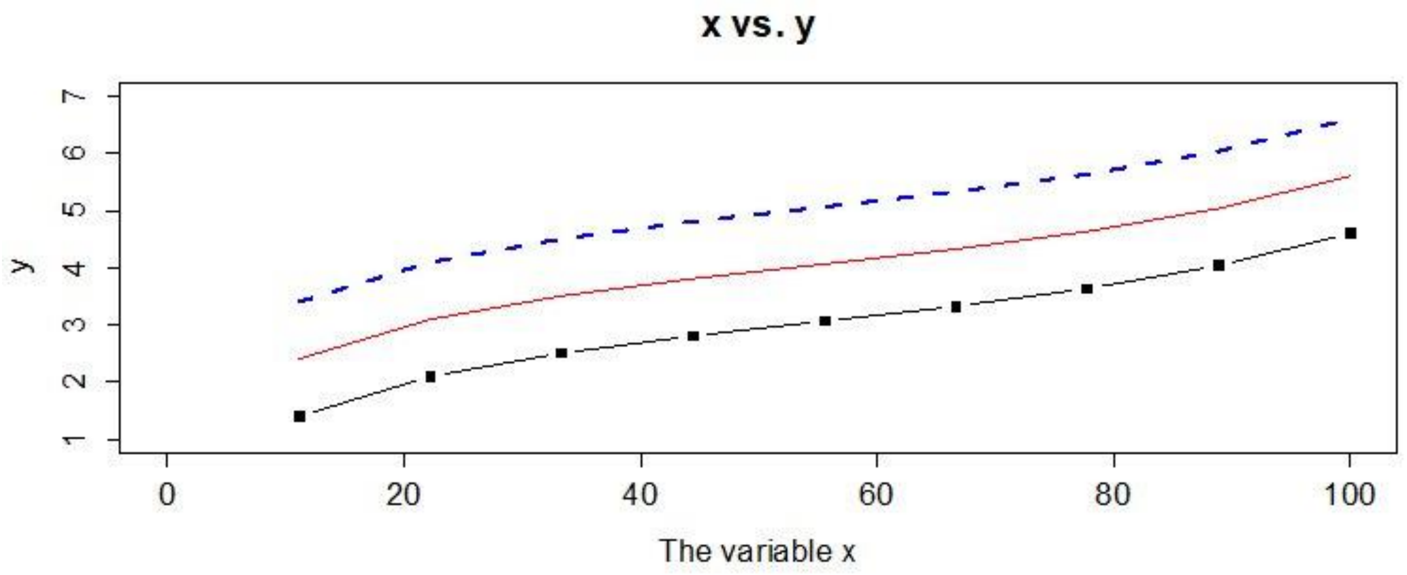
The **plot()** command takes further arguments to customize the plotting area:

- **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)
```

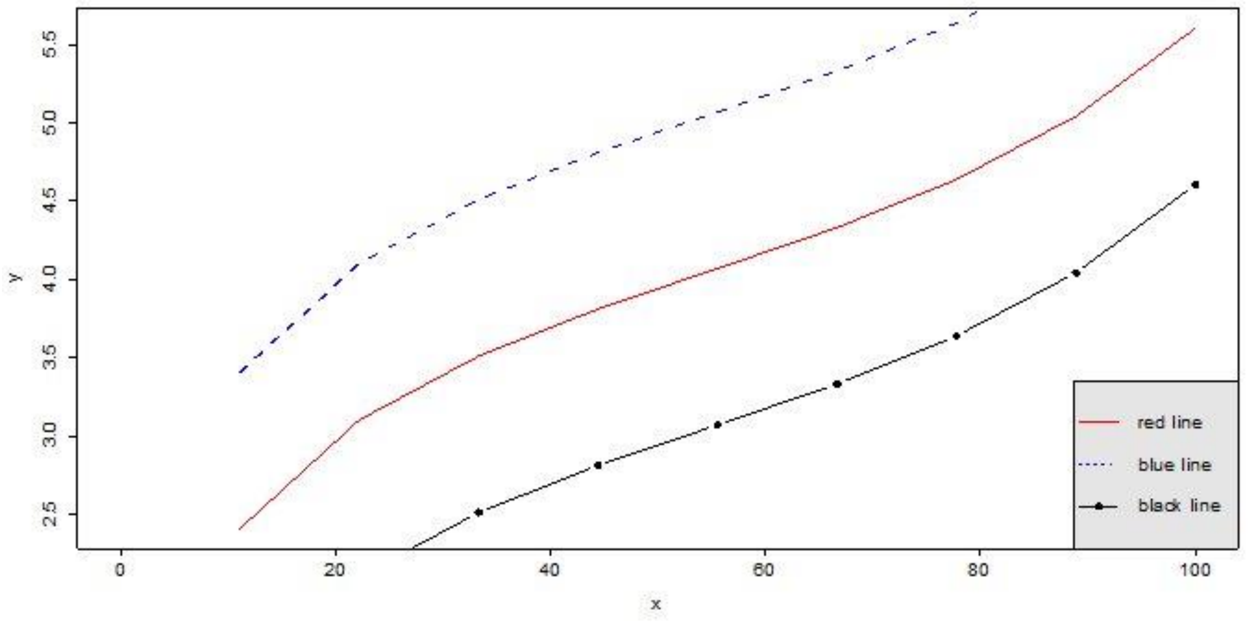


Customizing plots (III)

The **legend()** command adds a legend to plots:

- **x** , **y** to set the numeric coordinates for positioning the legend.
  - OR 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

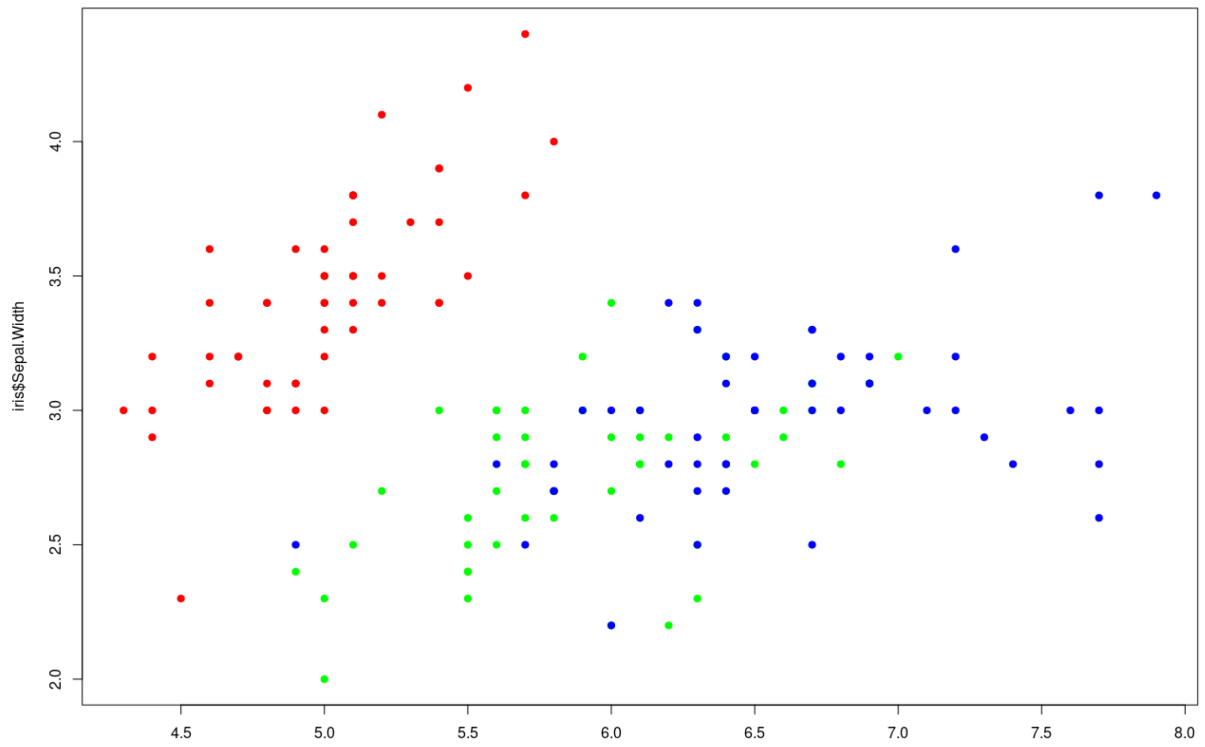
```
legend(x="bottomright",
       legend=c("red line",
           "blue line", "black line"),
       lty=c(1,2,1),
       pch=c(NA,NA,19),
       col=c("red", "blue", "black"),
       bg="gray90")
```



Customizing plots (IV)

When doing a scatterplot, it is common to color the points according to a categorical variable

```
data(iris)
plot(iris$Sepal.Length, iris$Sepal.Width ,
     col = c('red','green','blue')[ iris$Species ],
     pch=19)
```



Digression: why does the previous trick work?

```
iris$Species  # this is a factor
[1] setosa     versicolor   virginica     setosa     ...
Levels: setosa versicolor virginica

as.numeric(iris$Species) # factor coerced to numeric
[1] 1 2 3 1 ...

# when selecting elements in a vector,
# nothing prevents you from repeating an index
c('red','green','blue')[ c(1,1,1,2,2,2,3,3,3) ]
[1] "red" "red" "red" "green" "green" "green" ...

# factor are auto-coerced to numeric in this case:
c('red','green','blue')[iris$Species]
[1] "red" "green" "blue" "red" ...
```

abline() (I)

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:

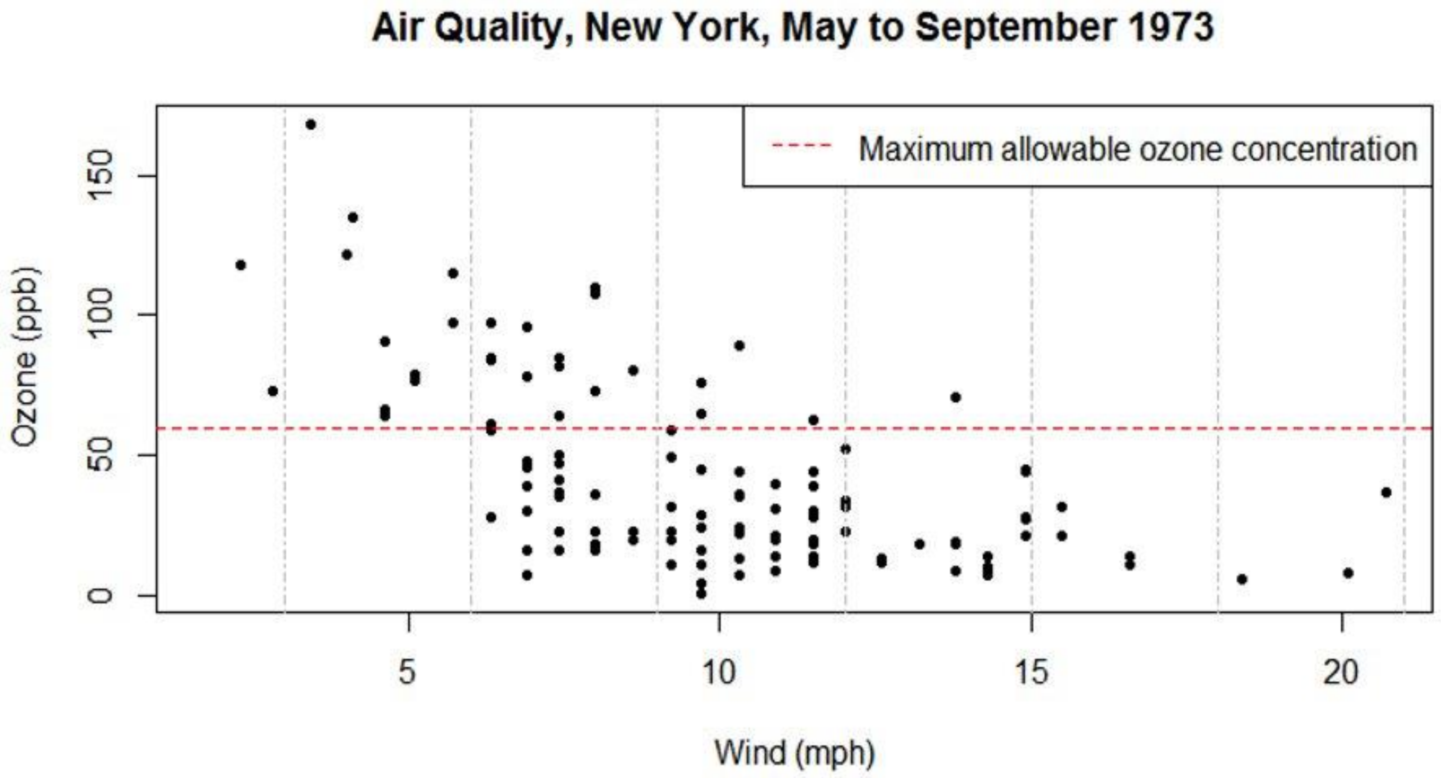
• abline(**v**=c(...)):	add vertical line(s) at the given x value(s)
• abline(**h**=c(...)):	add horizontal line(s) at the given y value(s)
• abline(**a**= ,**b**=):	add an affine line with intercept a and slope b
• abline(**reg**=**lm**(...)):	add a trend line from a linear regression equivalent to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

abline() (II) - 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")
```

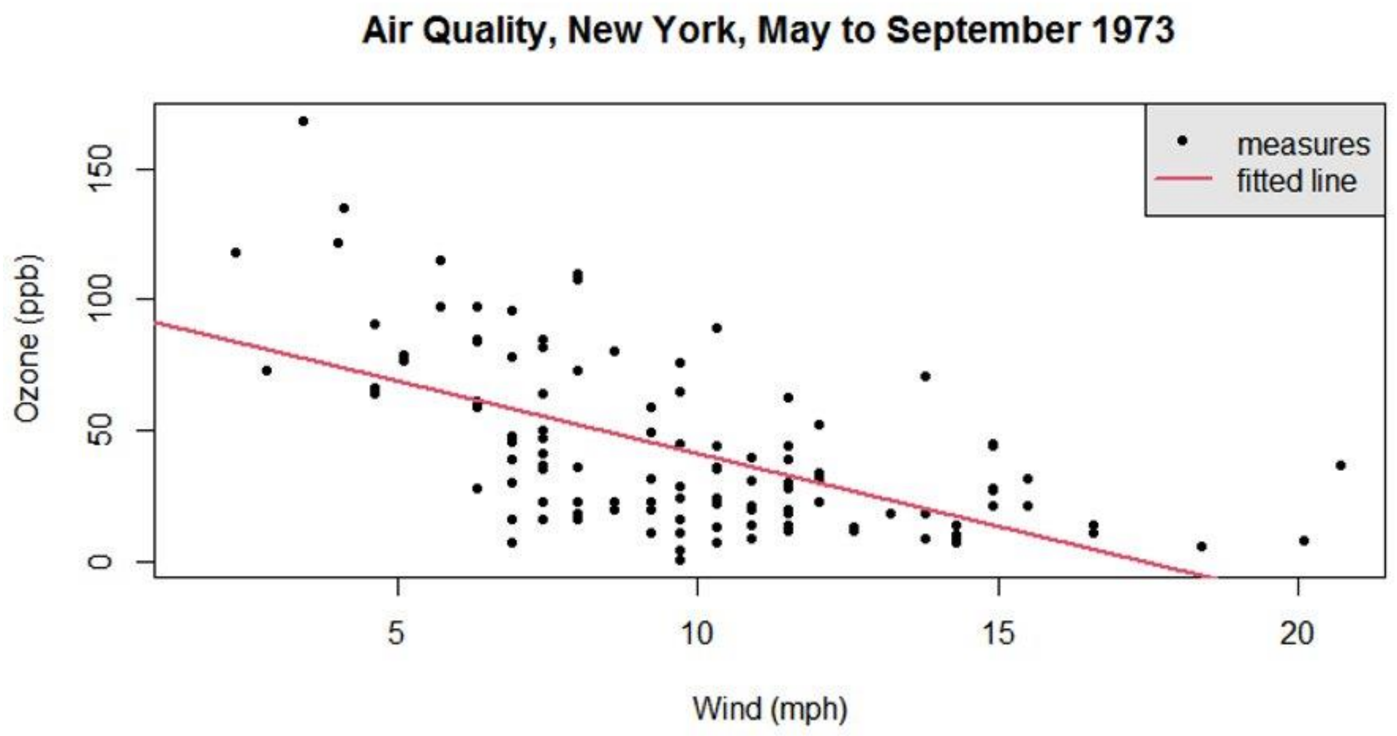


abline() (III) - trend line

```
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")
```



Let's practice - 7

Import the mouse data from the file course_dataset/mice_data_mod.csv.
This file contains the same data as mice_data.csv and in addition, two more columns.

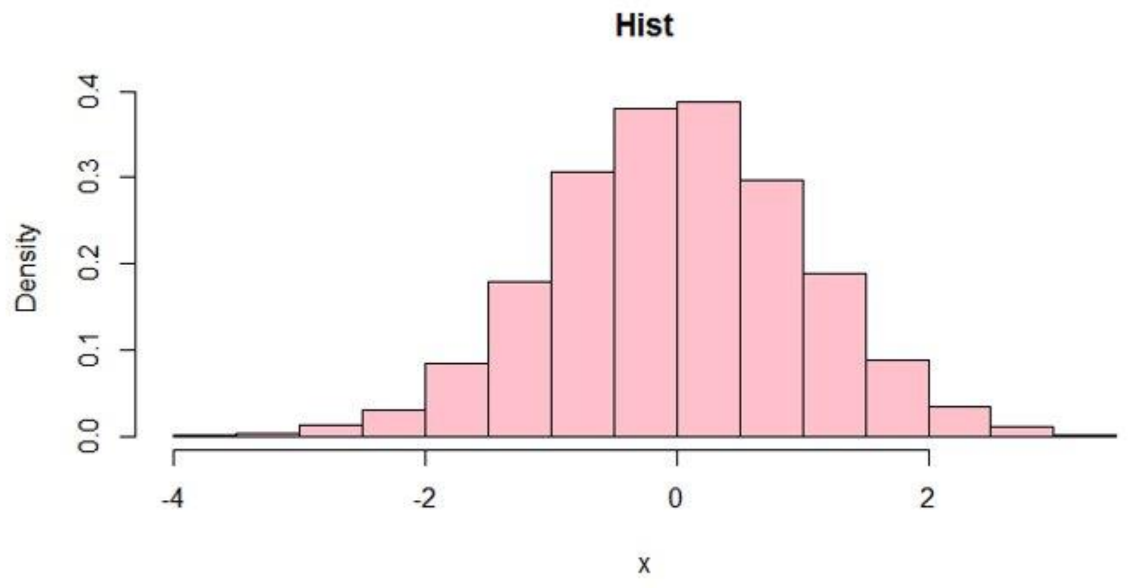
1. Run **str()** to check your data frame: did it load correctly?
2. Convert genotype and diet to factor variables.
3. Make a **scatter plot** of respiratory rate against mouse weights using the function plot().
   - use solid circles as plotting symbol
   - add a title
   - customize the axis labels (“Weight [g]”, “Respiratory Rate [bpm]”)
   - color the points **by genotype.**
4. Fit a trend line using the function **abline()**
5. Add a legend for the genotype

Histograms

hist() creates a histogram in a new plot (like plot). Main parameters:

- **breaks**: guides the number of bins. Number or vector of breakpoints
- **freq**:
    - TRUE : y-axis represents counts per bin
    - FALSE: y-axis represents density

```
x<- rnorm(10000) # 10000 random draw in a normal distribution
hist(x, freq=FALSE,
        main="Hist",
        col="pink")
```



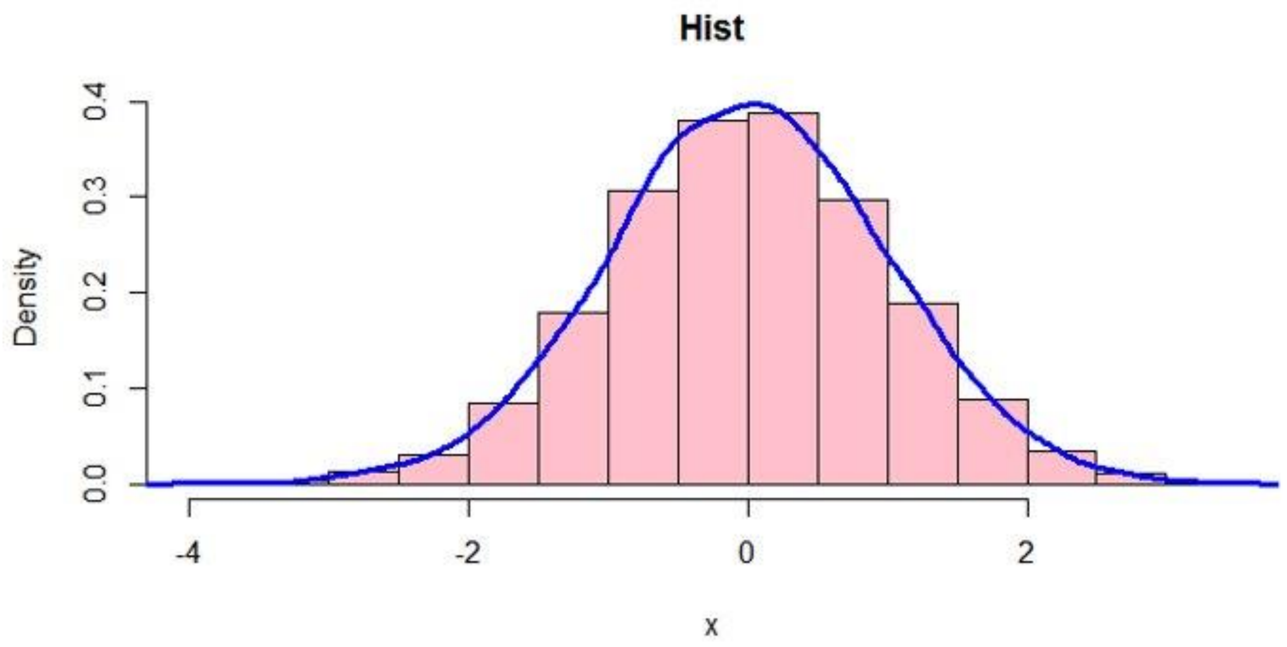
Histograms and density line

density() estimates the probability density of the data (kernel density estimates).

The output of **density()** can be given to the **lines()** function.

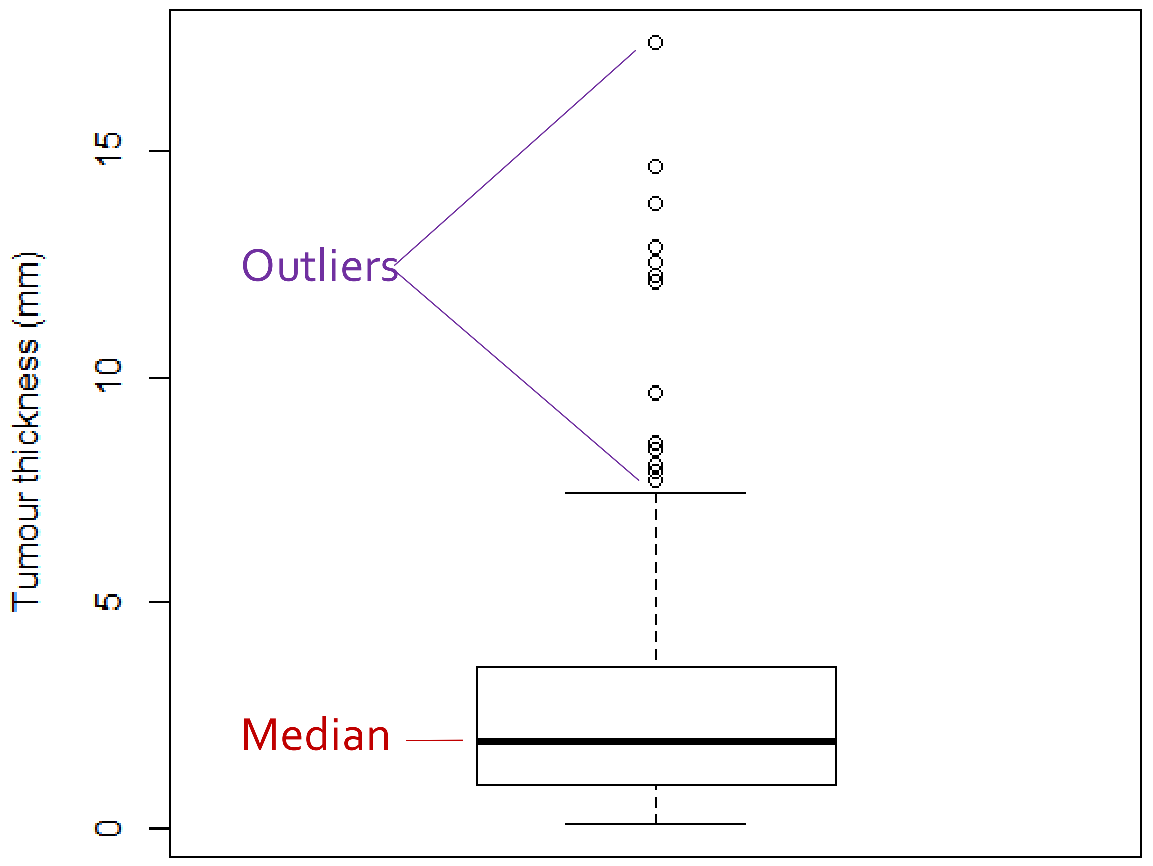
```
x<- rnorm(10000) # 10000 random draw in a normal distribution
hist(x, freq=FALSE,
        main="Hist",
        col="pink")

lines(density(x),
      Col="blue",
      lwd=3)
```



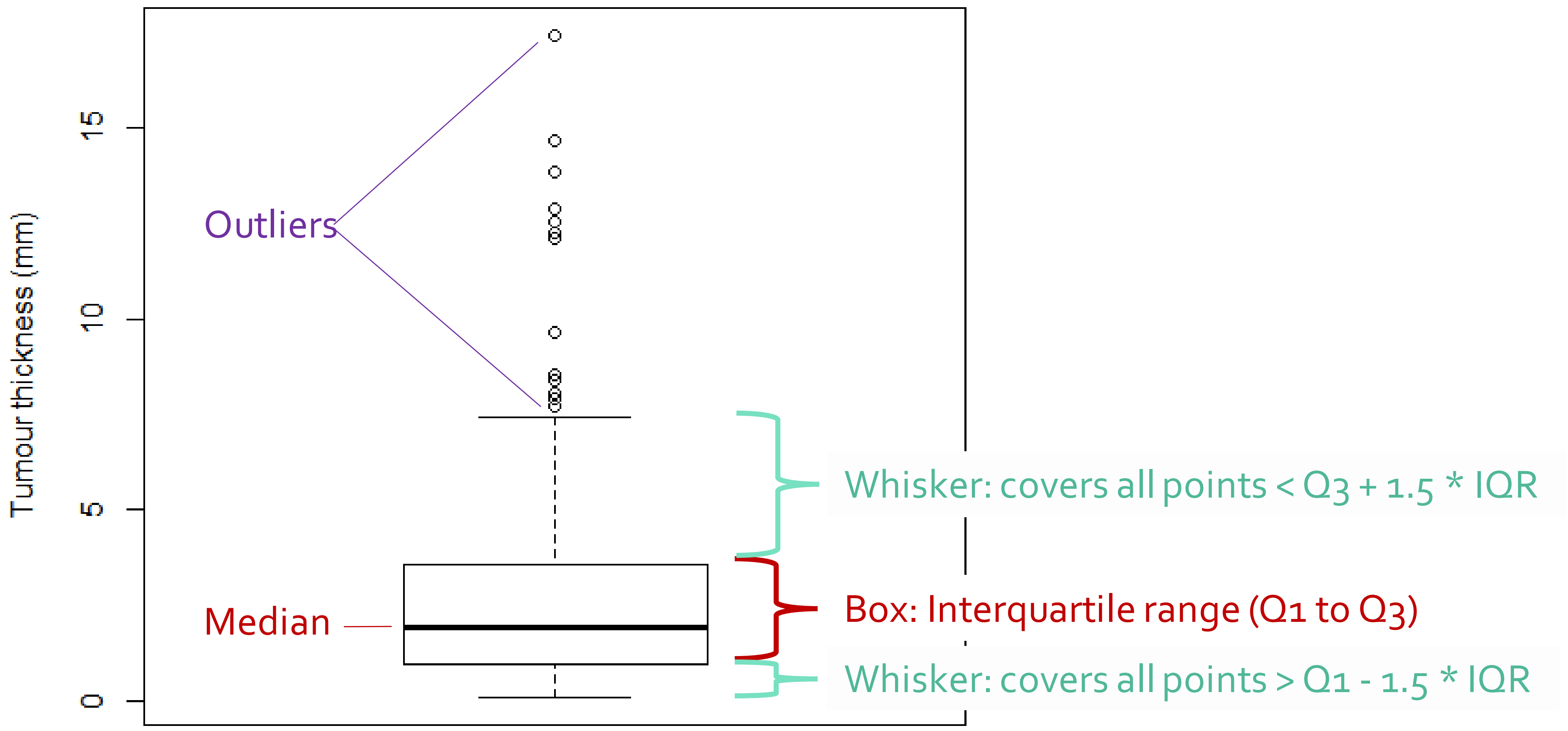
Boxplots (I) - anatomy

Example: melanoma thickness in 205 patients



Boxplots (II) - anatomy

Example: melanoma thickness in 205 patients



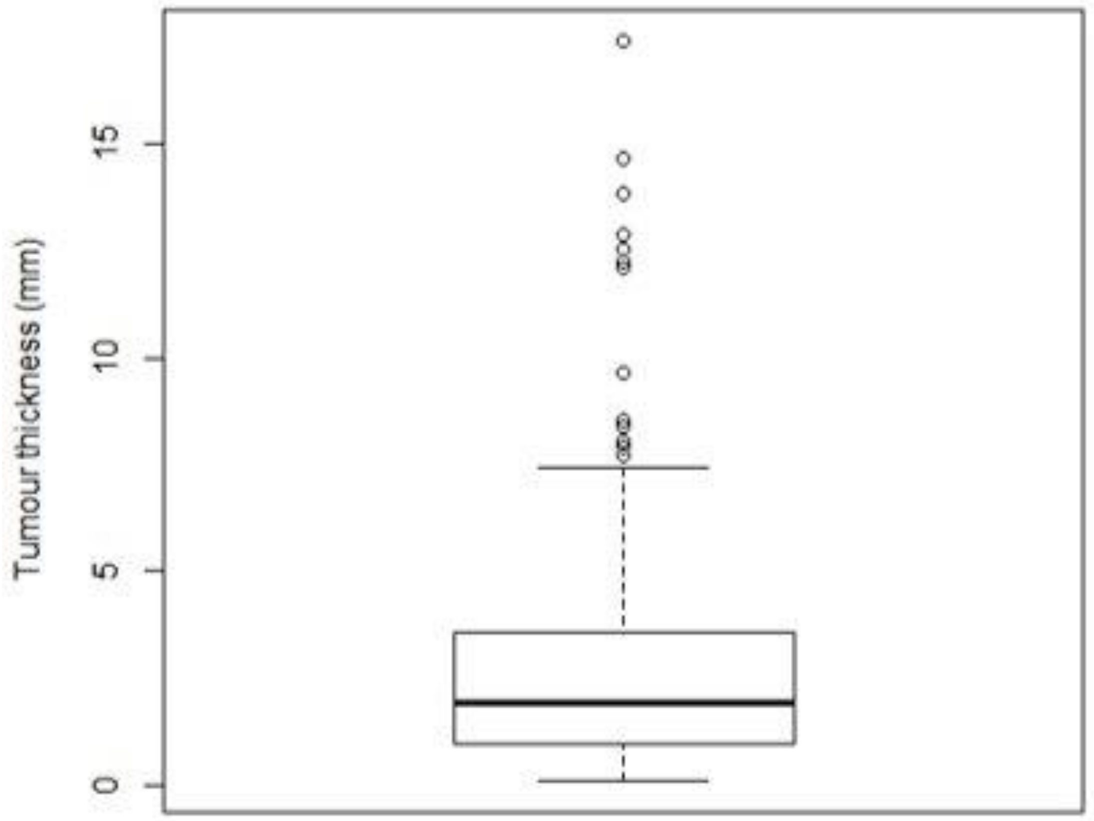
Boxplot (III) - plotting code

```
library(MASS)

data(Melanoma) # 205 patients with malignant melanoma

head(Melanoma)
 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')
```



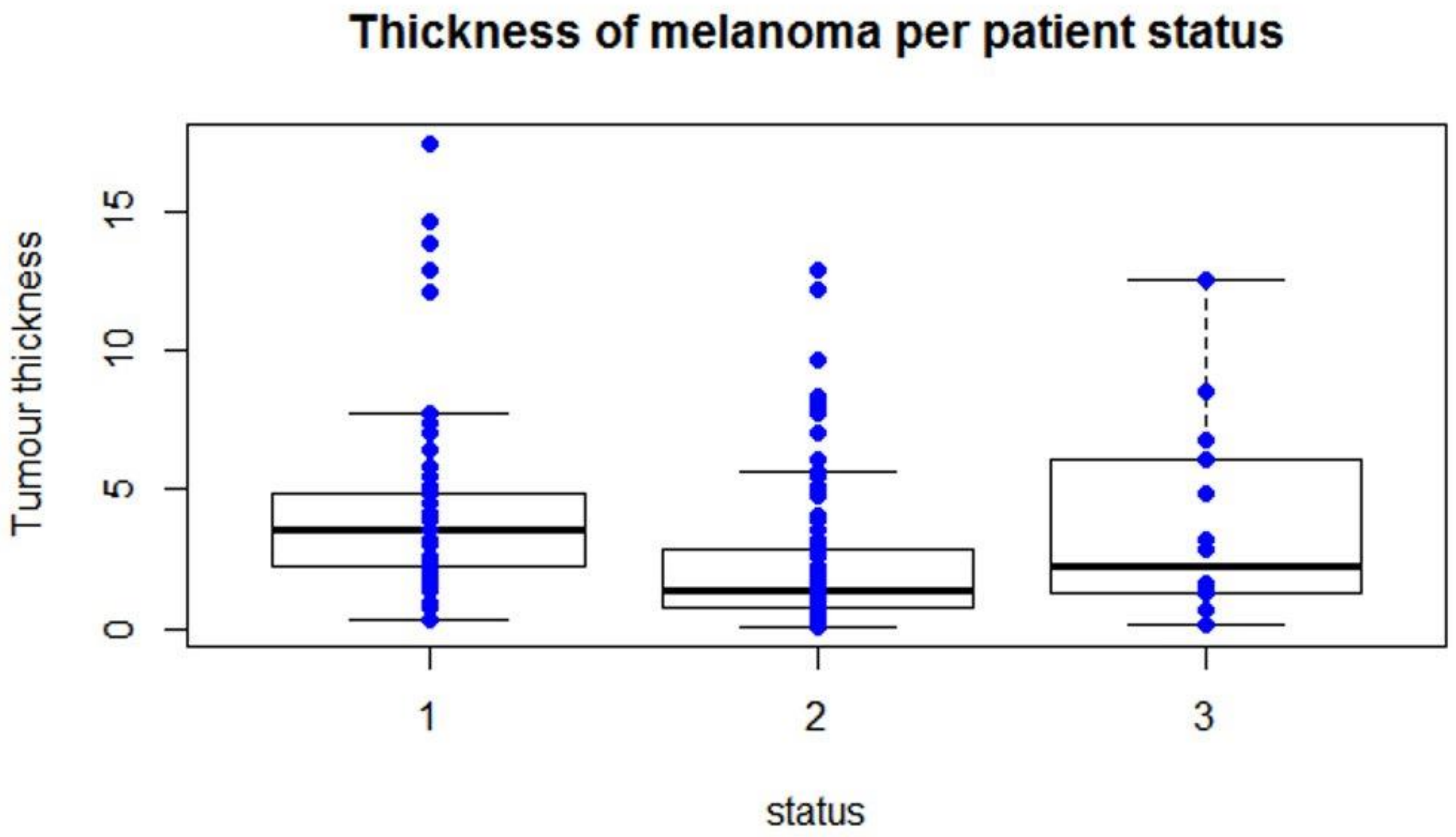
Boxplot (IV) - more boxplots

We want

1. separate boxplots for subgroups
2. individual points overlaid

status:

1. died from melanoma
2. alive
3. dead from other causes



Boxplot (V) - 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)
```

Boxplot (VI) - 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
```

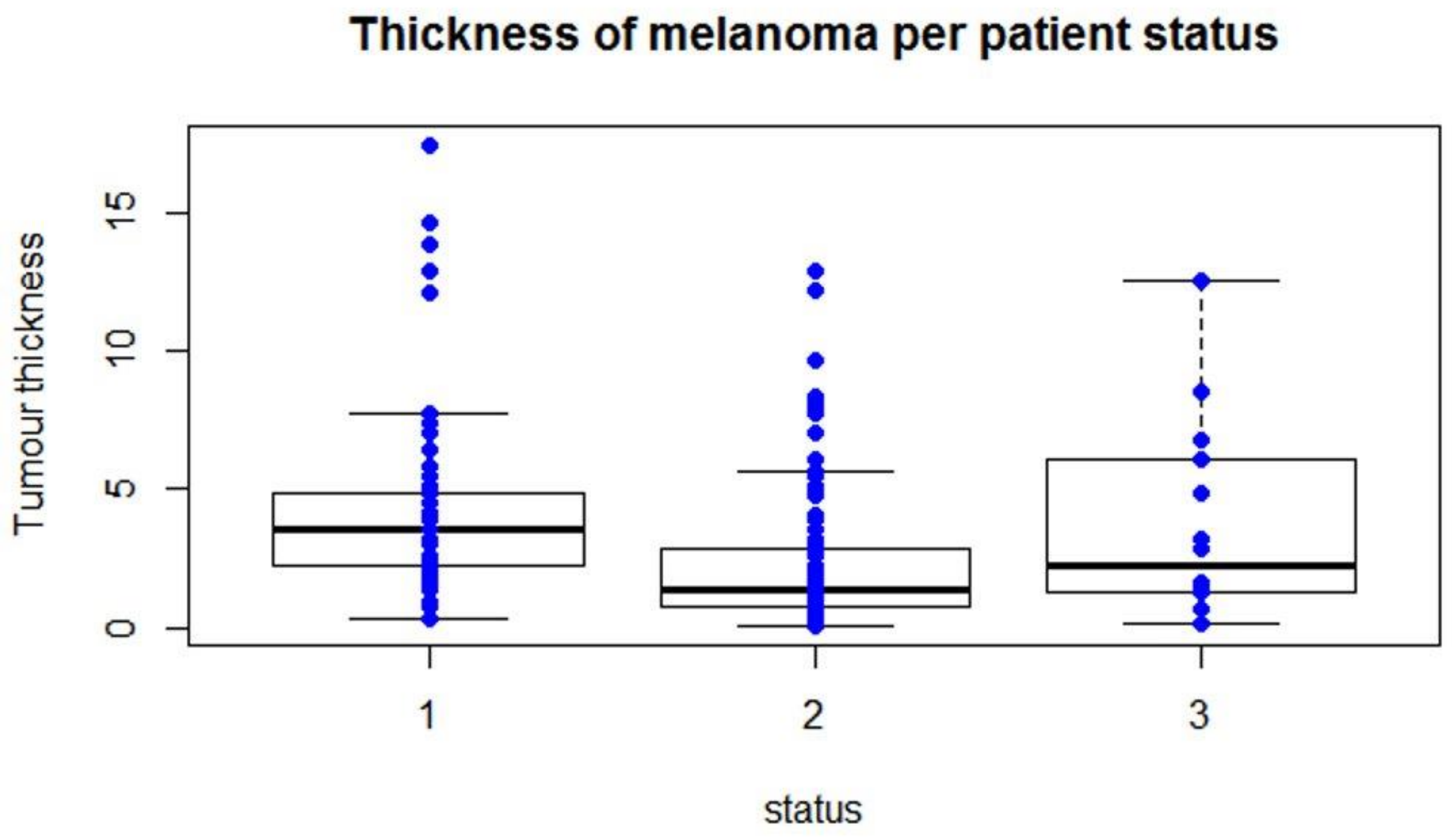
Method 2: Formulas

```
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
```

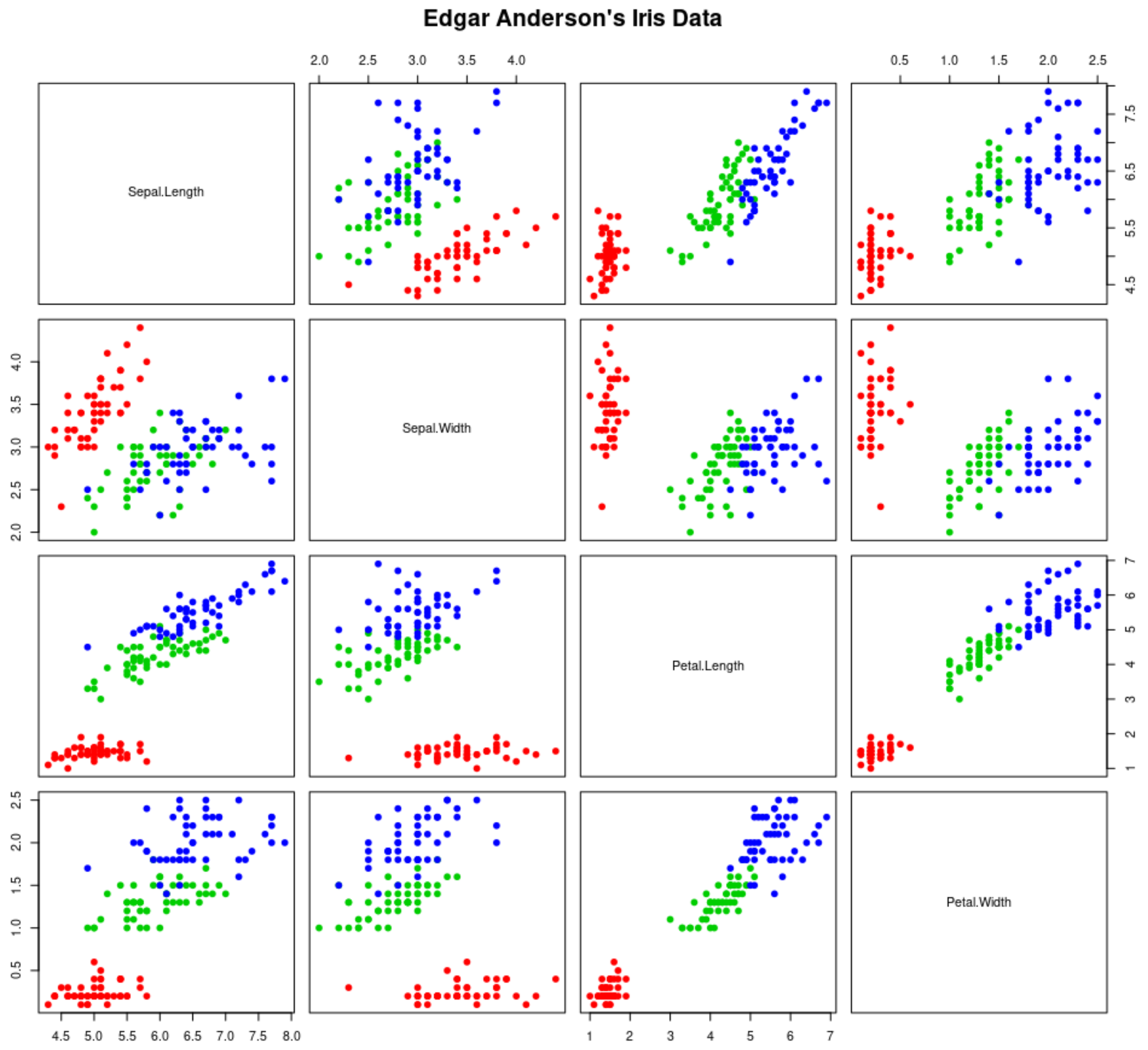
Boxplot (VII) - result

Both methods give the same result:



Pairs scatter plot - pairs()

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], pch=19,
      col=c("red", "green3", "blue")[iris$Species])
```

Let's practice - 8

This is a continuation of the previous practice: we will continue to plot the mice data

1. Plot a **histogram** of mouse weight and customize it with title, labels, colors. Represent the density line on top.
2. Make **boxplots** of weights from WT and KO mice. Customize with title, labels, colors.
3. *Optional:* Repeat 2 with diet instead of genotype.

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

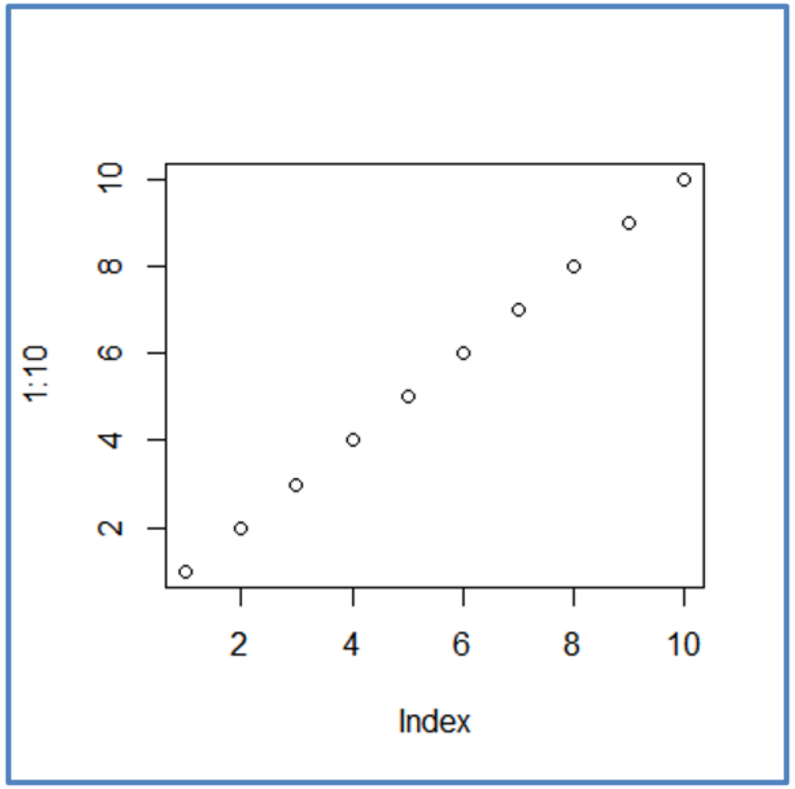
- **mar** sets plot margins in number of lines
- **mai** sets plot margins in inches
- use vectors of 4 values (c(0,1,1,2)) for the bottom, left, top, and right margins

```
par(mar=c(5.1,4.1,4.1,2.1))      #set margins in lines
par(mai=c(1.02,0.82,0.82,0.42)) #set margins in inches
```

Permanent graphic changes (II)

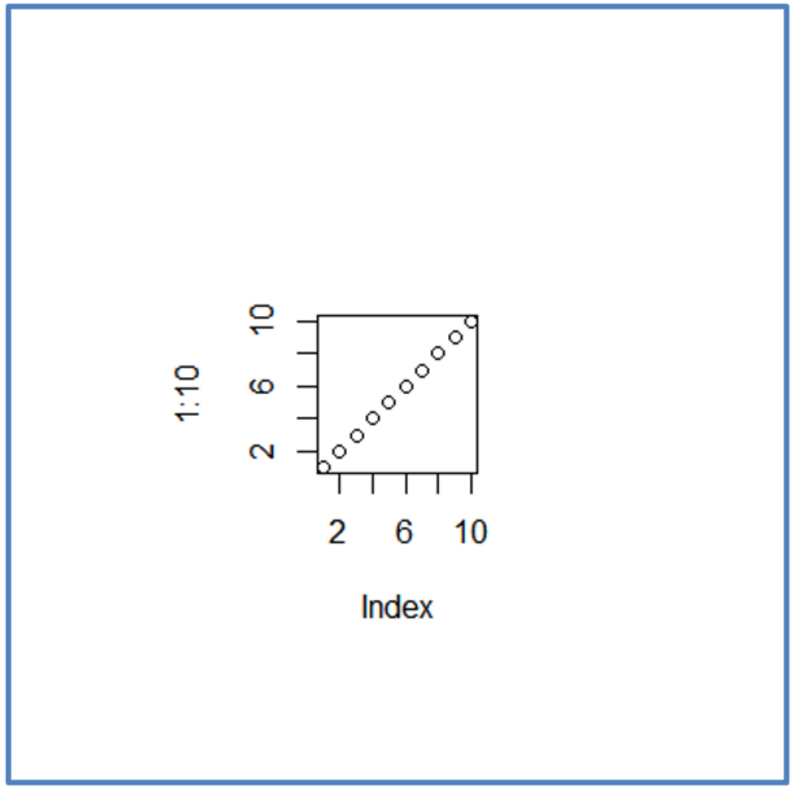
Normal margins

```
# 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)
```



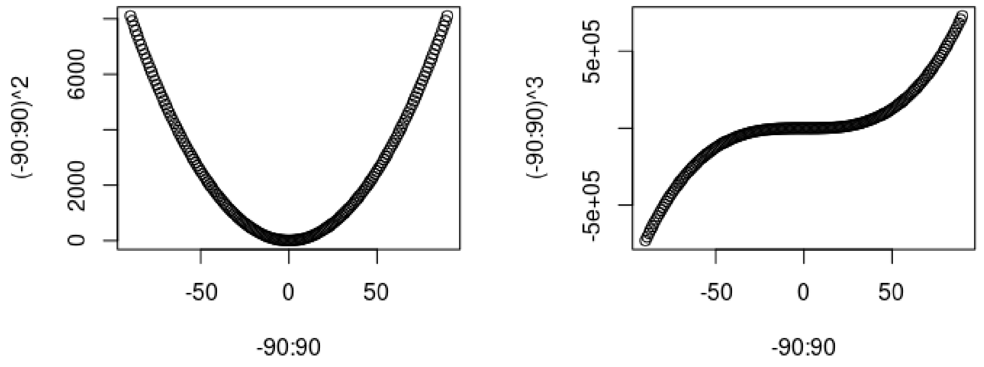
Permanent graphic changes (III)

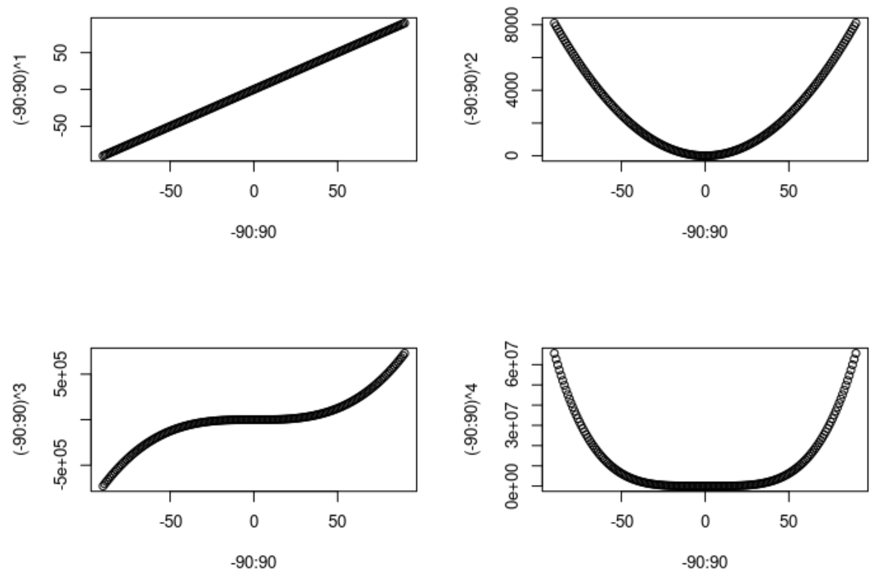
par() can be used to generate a multi-panel figures with the argument

- mfrow: a vector with two elements (number of rows, number of columns)

```
par(mfrow=c(1,2))
plot(-90:90 , (-90:90)**2 )
plot(-90:90 , (-90:90)**3 )
```

```
par(mfrow=c(2,2))
plot(-90:90 , (-90:90)**1 )
plot(-90:90 , (-90:90)**2 )
plot(-90:90 , (-90:90)**3 )
plot(-90:90 , (-90:90)**4 )
```



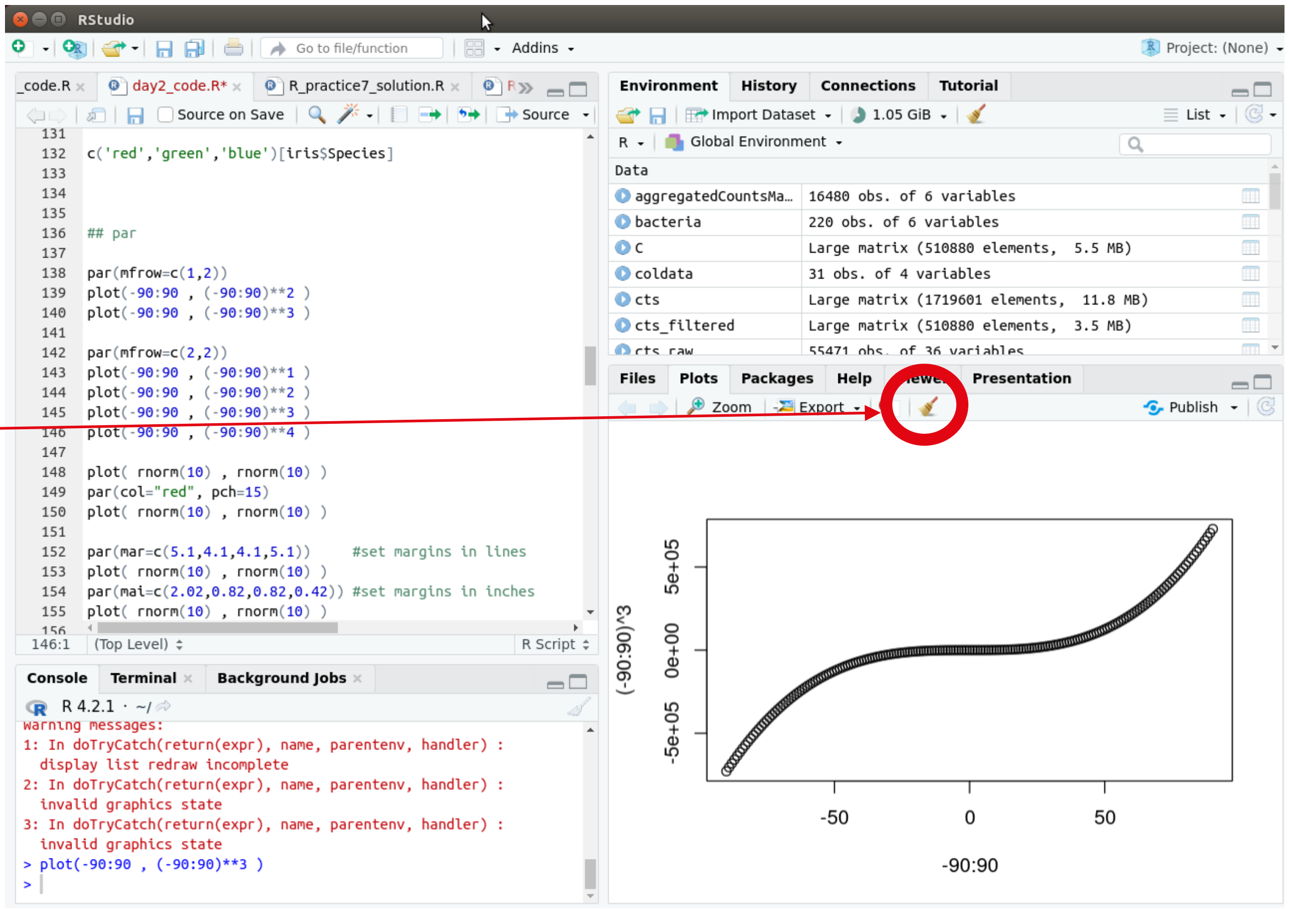


Permanent graphic changes (IV)

Current settings of par(): call **par()** without arguments

Resetting par()

- **Restart R** or switch **Rstudio projects**
- **dev.off()** : closes the most recent plot
- **graphics.off()** : closes all plots
- **Rstudio** broom icon



```
c('red','green','blue')[iris$Species]



## par

par(mfrow=c(1,2))
plot(-90:90 , (-90:90)**2 )
plot(-90:90 , (-90:90)**3 )

par(mfrow=c(2,2))
plot(-90:90 , (-90:90)**1 )
plot(-90:90 , (-90:90)**2 )
plot(-90:90 , (-90:90)**3 )
plot(-90:90 , (-90:90)**4 )

plot( rnorm(10) , rnorm(10) )
par(col="red", pch=15)
plot( rnorm(10) , rnorm(10) )

par(mar=c(5.1,4.1,4.1,5.1))      #set margins in lines
plot( rnorm(10) , rnorm(10) )
par(mai=c(2.02,0.82,0.82,0.42)) #set margins in inches
plot( rnorm(10) , rnorm(10) )
```

146:1 (Top Level) R Script

Data	
aggregatedCountsMa…	16480 obs. of 6 variables
bacteria	220 obs. of 6 variables
C	Large matrix (510880 elements, 5.5 MB)
coldata	31 obs. of 4 variables
cts	Large matrix (1719601 elements, 11.8 MB)
cts_filtered	Large matrix (510880 elements, 3.5 MB)
cts_raw	55471 obs. of 36 variables

Console Terminal Background Jobs

R 4.2.1 · ~/

```
Warning messages:
1: In doTryCatch(return(expr), name, parentenv, handler) :
  display list redraw incomplete
2: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
3: In doTryCatch(return(expr), name, parentenv, handler) :
  invalid graphics state
> plot(-90:90 , (-90:90)**3 )
> 
```

Saving figures to files

Programmatic method: surround the plotting code with

- Before format function: pdf(), png(), jpeg(),…
- After: dev.off()

```
pdf(file="quadratic_cubic.pdf", width=7, height=4, paper="a4")
plot(-90:90,(-90:90)**2)
dev.off()
```

Alternatively, you may use Rstudio interface:

- Plots > Export > Save as Image (PNG,JPEG,TIGG,BMP)
- Plots > Save as PDF

Graphic export functions

Use correct file extension:

- postscript(file="a_name.ps", ...)
- pdf(file="...pdf", ...)
- jpeg(file=" ...jpg", ...)
- png(file=" ....png", ...)

- Different devices have different set of arguments: height=, width=, res=, paper=, pointsize=, ...
- png, jpeg, tiff (raster formats): width and height are in pixels.
- pdf and postscript (vector formats): width and height 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

- pdf and postscript have an argument "paper".

  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 many points are plotted, pdfs can become large in file size and slow to display. When this is an issue, consider png.

Let's practice - 9

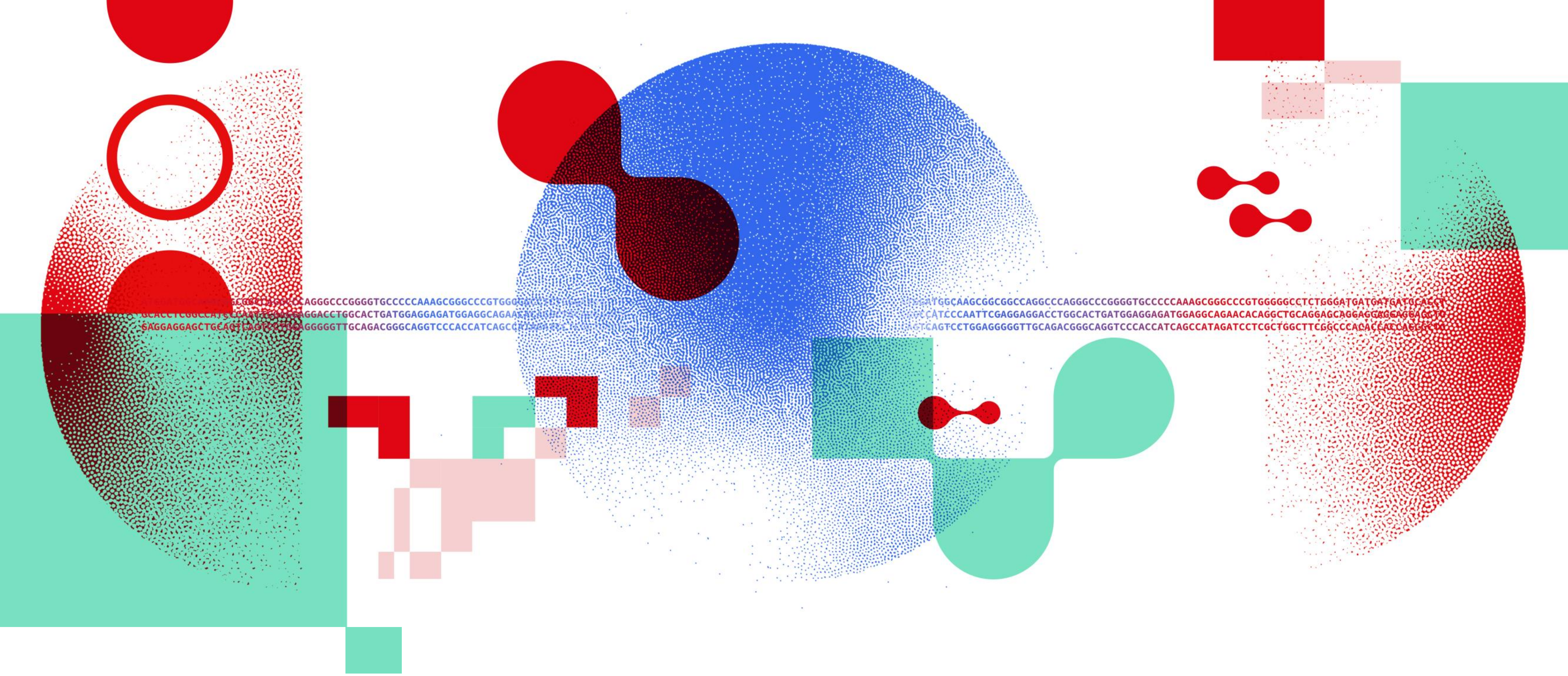
This is a continuation of the previous practice: we will continue to plot the mice data

1. Make a multi-panel figure with the **four graphics (from the previous exercises) on one page**
2. Change the code to export 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.
3. **Optional:** Export an histogram (from previous exercise) to a **png** file**.** Set width and height arguments in the call to png() to make it look nice.

Plots are initiated with: plot(), hist(), boxplot(), …

Further modification with: points(), lines(), abline(), …

examples: https://r-graph-gallery.com/index.html



Thank you

sib.swiss