

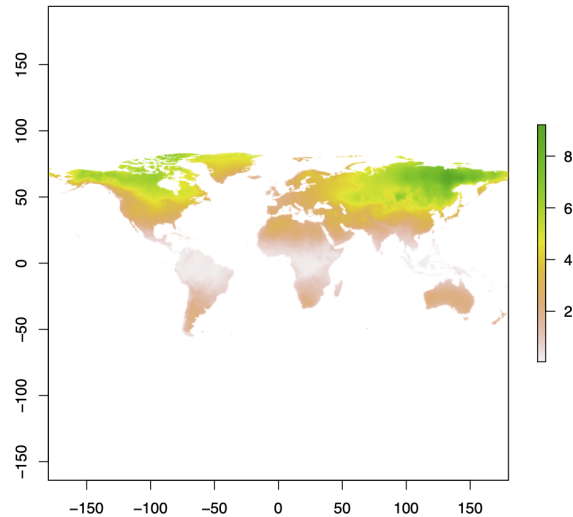


## Base R graphics

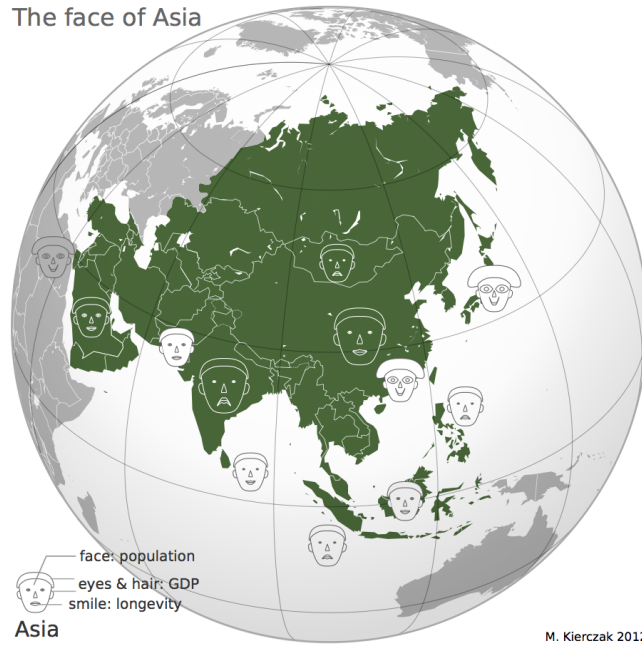
R Foundations for Life Scientists

Marcin Kierczak

# Example graphics

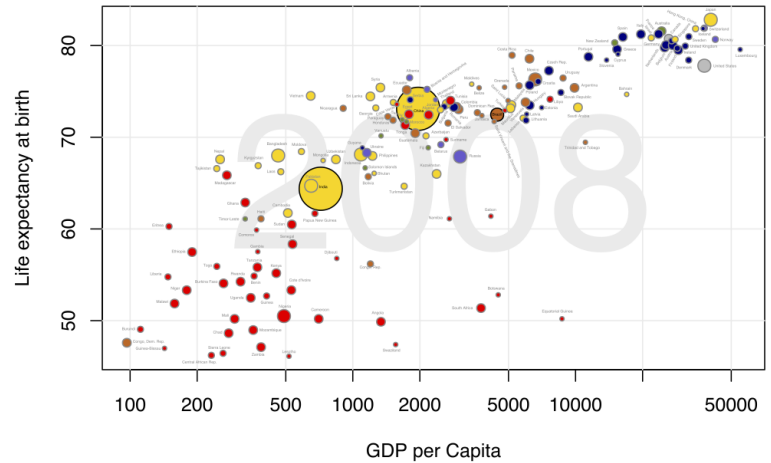


The face of Asia



Stability of the climate (courtesy of Dr. Mats Pettersson).

# Example graphics



# Graphical devices

The concept of a **graphical device** is crucial for understanding R graphics.

A device can be a screen (default) or a file. Some R packages introduce their own devices, e.g. Cairo device. Creating a plot entails:

- opening a graphical device (not necessary for plotting on screen),
- plotting to the graphical device,
- closing the graphical device (very important!)

## The most commonly used graphical devices are:

- screen,
- bitmap/raster devices: `png()`, `bmp()`, `tiff()`, `jpeg()`
- vector devices: `svg()`, `pdf()`,
- `Cairo` versions of the above devices – for Windows users they offer higher quality graphics,  
For more information visit [this link](#).

# Working with graphical devices

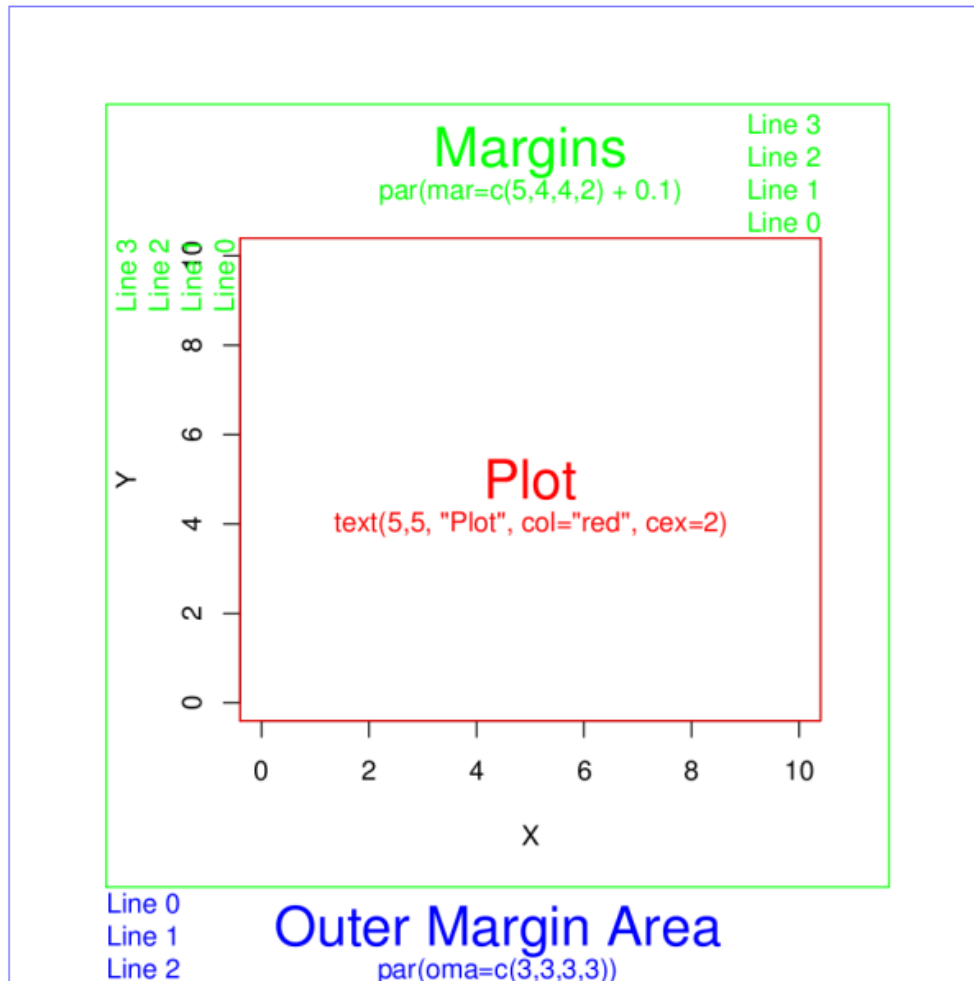
```
png(filename = 'myplot.png', width = 320, height = 320, antialias = T)
plot(x=c(1,2,7), y=c(2,3,5))
dev.off()
```

What we did was, in sequence:

- open a graphical device, here `png()` with some parameters,
- do the actual plotting to the device using `plot()` and
- close the graphical device using `dev.off()`.

It is of paramount importance to remember to close graphical devices. Otherwise, our plots may end up in some random weird files or on screens.

# Standard graphical device viewport



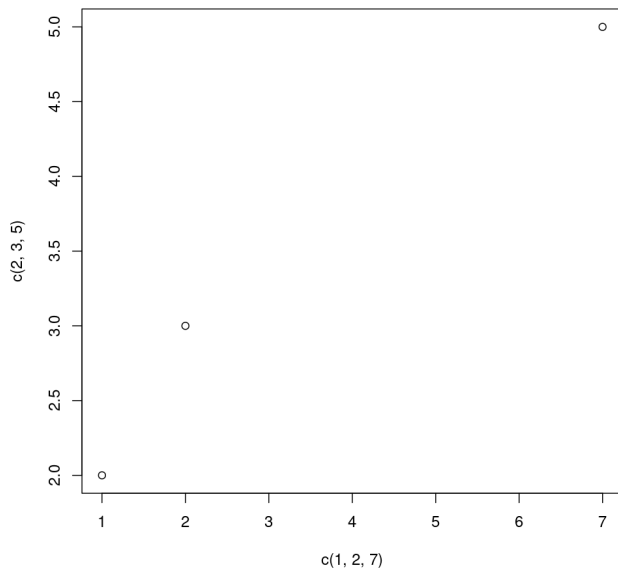
source: [rgraphics.limnology.wisc.edu](http://rgraphics.limnology.wisc.edu)

## `base::plot()` basics

`base::plot()` is a basic command that lets you visualize your data and results. It is very powerful yet takes some effort to fully master it. Let's begin with plotting three points:

- `A(1,2);`
- `B(2,3);`
- `C(7,5);`

```
plot(x=c(1,2,7), y=c(2,3,5))
```

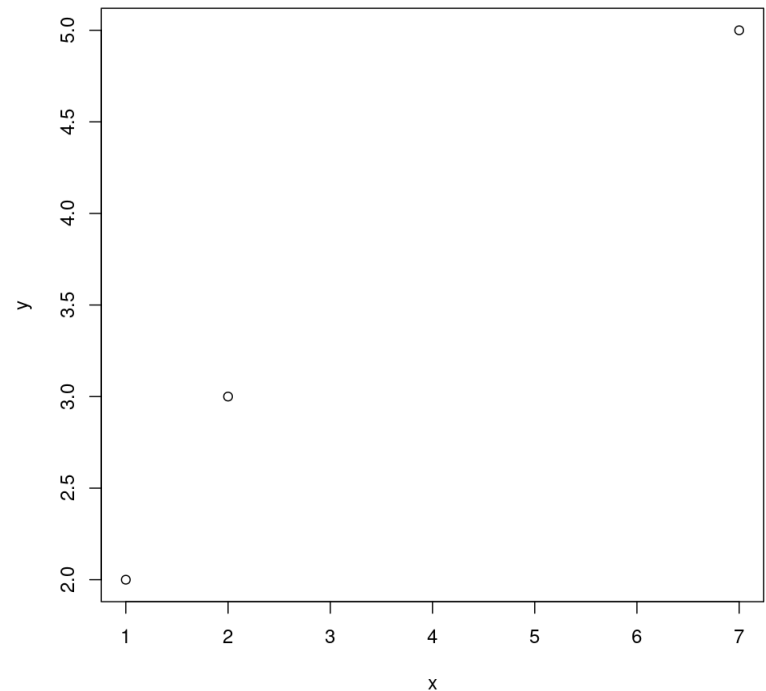


# Anatomy of a plot — episode 1

For convenience, we will create a data frame with our points:

```
df <- data.frame(x = c(1, 2, 7), y = c(2, 3, 5),  
df  
plot(df)
```

```
##   x y  
## A 1 2  
## B 2 3  
## C 7 5
```





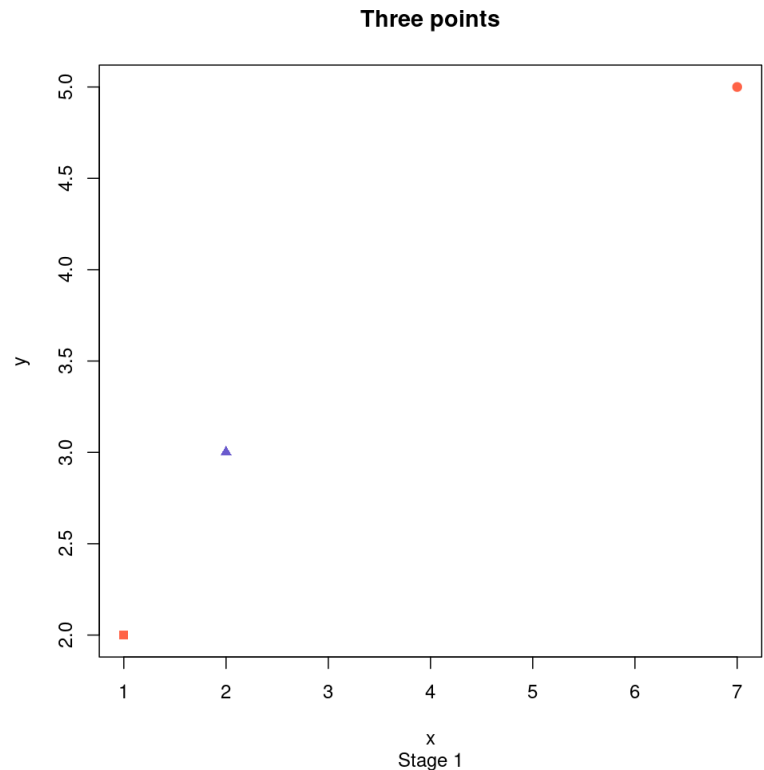
# Anatomy of a plot — episode 2

There is many parameters one can set in `plot()`. Let's have a closer look at some of them:

- `pch` – type of the plotting symbol
- `col` – color of the points
- `cex` – scale for points
- `main` – main title of the plot
- `sub` – subtitle of the plot
- `xlab` – X-axis label
- `ylab` – Y-axis label
- `las` – axis labels orientation
- `cex.axis` – axis labels scale

Let's make our plot a bit fancier...

```
# recycling rule in action!  
plot(df, pch = c(15, 17, 19), col = c("tomato", "blue", "red"),  
      sub = "Stage 1")
```



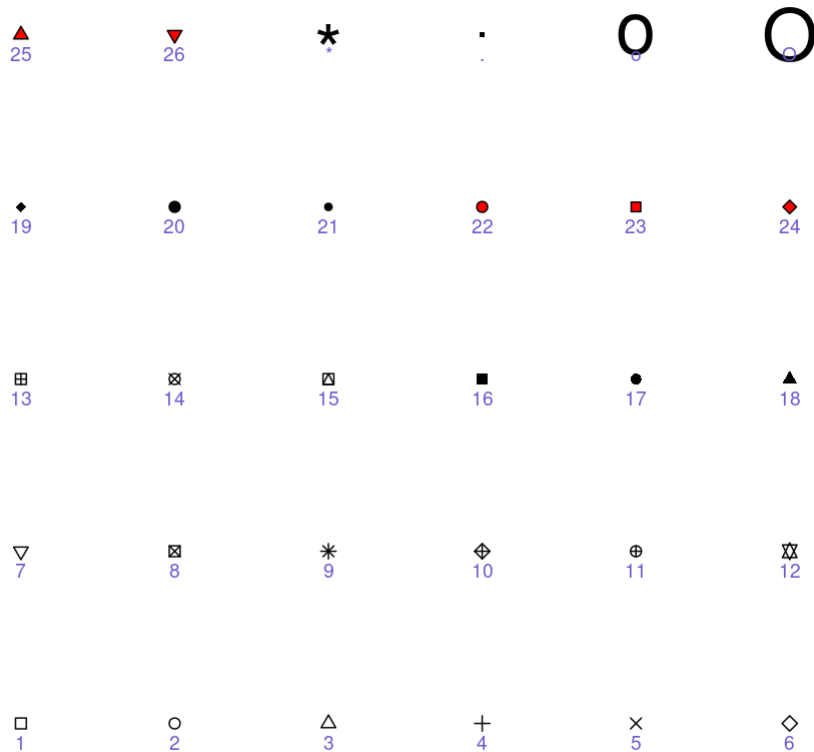
# Graphical parameters

Graphical parameters can be set in two different ways:

- as plotting function arguments, e.g. `plot(dat, cex = 0.5)`
- using `par()` to set parameters globally

```
# read current graphical parameters
par()
# first, save the current parameters so that you
# can set them back if needed
old_par <- par() # should work in theory, practise varies :-)
# now, modify what you want
par(cex.axis = 0.8, bg='grey')
# do your plotting
plot(.....)
# restore the old parameters if you want
par(old_par)
```

# The `pch` parameter



# How to make the `pch` cheatsheet

```
# create a grid of coordinates
coords <- expand.grid(1:6,1:6)
# make a vector of numerical pch symbols
pch.num <- c(0:25)
# and a vector of character pch symbols
pch.symb <- c('*', '.', 'o', 'O', '0', '-', '+', '|', '%', '#')
# plot numerical pch
plot(coords[1:26,1], coords[1:26,2], pch=pch.num,
      bty='n', xaxt='n', yaxt='n', bg='red',
      xlab='', ylab='')
# and character pch's
points(coords[27:36,1], coords[27:36,2], pch=pch.symb)
# label them
text(coords[,1], coords[,2], c(1:26, pch.symb), pos = 1,
      col='slateblue', cex=.8)
```

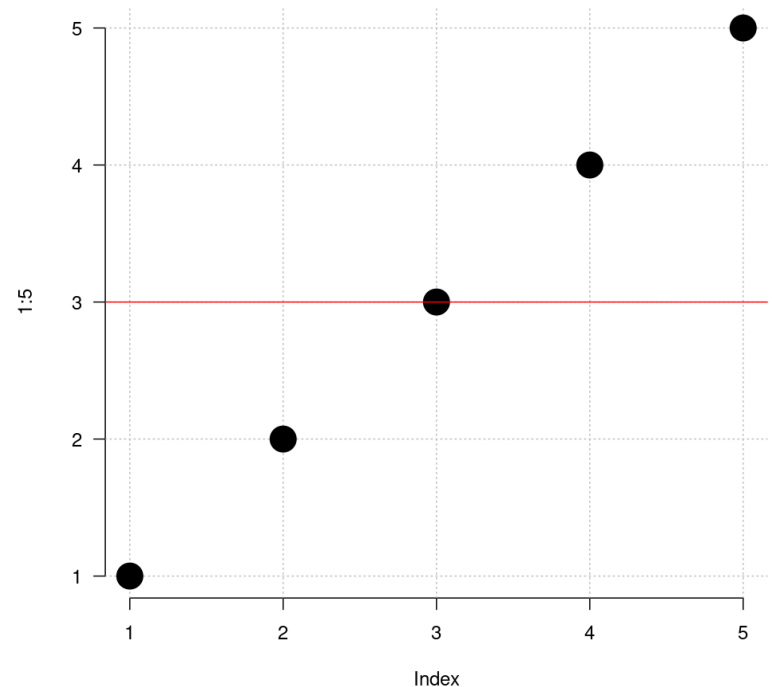
Now, make your own cheatsheet for the `lty` parameter!

# Layers

Elements are added to a plot in the same order you plot them. It is like layers in a graphical program. Think about it! For instance the auxiliary grid lines should be plotted before the actual data points.

```
# make an empty plot  
plot(1:5, type = "n", las = 1, bty = "n")  
grid(col = "grey", lty = 3)  
points(1:5, pch = 19, cex = 3)  
abline(h = 3, col = "red")
```

The line overlaps one data point. It is better to plot it before plotting `points()`



# Some thoughts about plotting

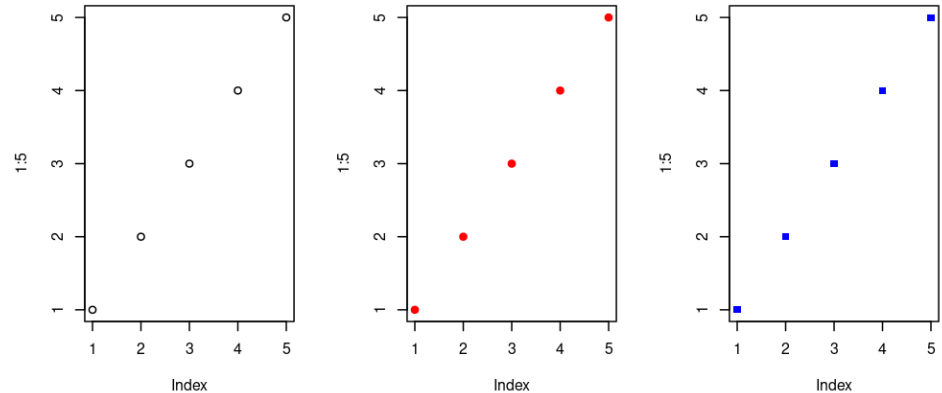
There is a few points you should have in mind when working with plots:

- raster or vector graphics,
- colors, e.g. color-blind people, warm vs. cool colors and optical illusions,
- avoid complications, e.g. 3D plots, pie charts etc.,
- use black and shades of grey for things you do not need to emphasize, i.e. basically everything except your main result,
- avoid 3 axes on one figure.

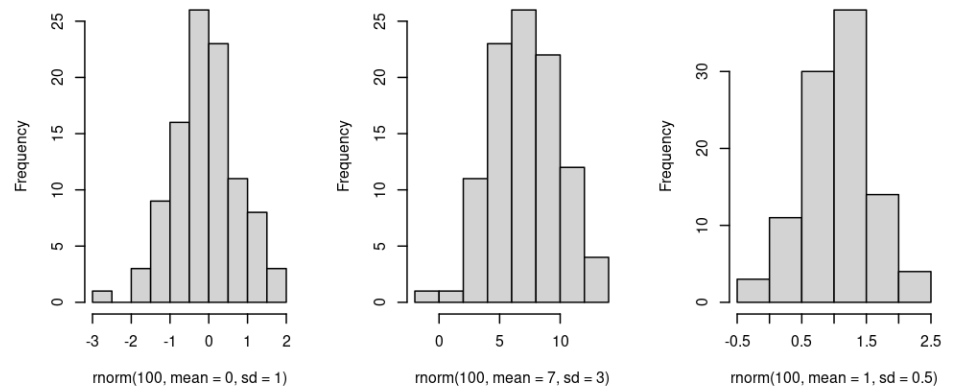
# Many plots on one figure

```
par(mfrow=c(2,3))
plot(1:5)
plot(1:5, pch=19, col='red')
plot(1:5, pch=15, col='blue')
hist(rnorm(100, mean = 0, sd=1))
hist(rnorm(100, mean = 7, sd=3))
hist(rnorm(100, mean = 1, sd=0.5))
par(mfrow=c(1,1))
```

Alternative: use `par(mfcol=c(3,2))`.



Histogram of `rnorm(100, mean = 0, sd=1)`, histogram of `rnorm(100, mean = 7, sd=3)`, histogram of `rnorm(100, mean = 1, sd=0.5)`



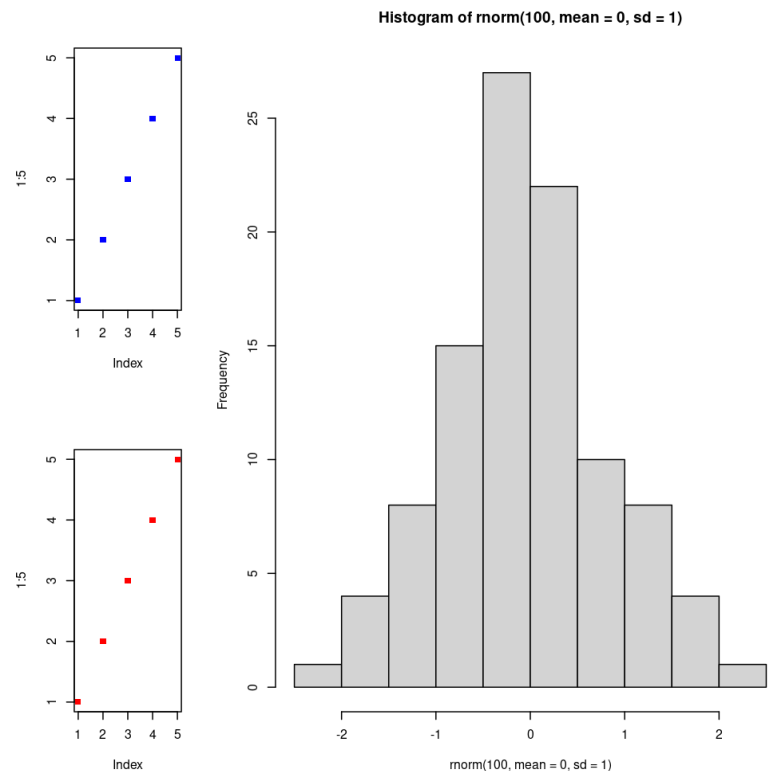
# Using `graphics::layout()`

```
M <- matrix(c(1,2,2,2,3,2,2,2), nrow = 2, ncol = 5)
layout(mat = M)
plot(1:5, pch=15, col='blue')
hist(rnorm(100, mean = 0, sd=1))
plot(1:5, pch=15, col='red')
```

```
M <- matrix(c(1,2,2,2,3,2,2,2), nrow = 2, ncol = 5)
layout(mat = M)
plot(1:5, pch=15, col='blue')
hist(rnorm(100, mean = 0, sd=1))
plot(1:5, pch=15, col='red')
```

M

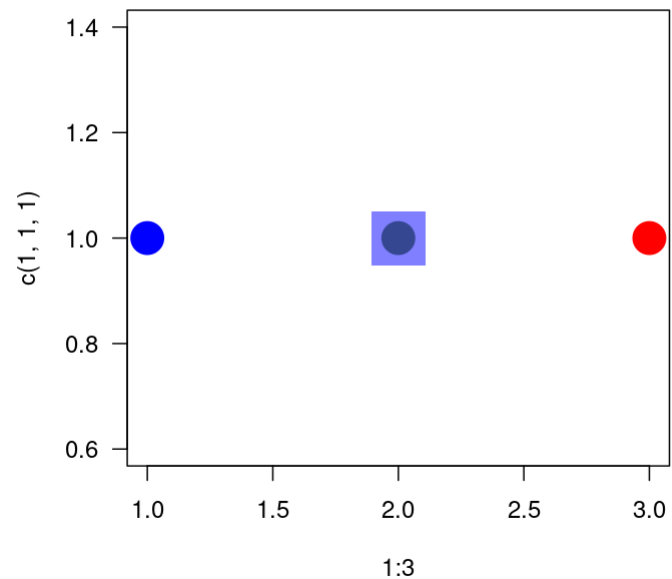
```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    2    2
## [2,]    3    2    2    2
```





# Defining colors

```
mycol <- c(rgb(0, 0, 1), "olivedrab", "#FF0000")  
plot(1:3, c(1, 1, 1), col = mycol, pch = 19, cex = 3)  
points(2, 1, col = rgb(0, 0, 1, 0.5), pch = 15, cex = 5)
```

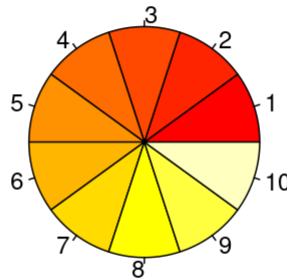


# Color palettes

There some built-in palettes: default, hsv, gray, rainbow, terrain.colors, topo.colors, cm.colors, heat.colors

```
mypal <- heat.colors(10)
mypal
pie(x = rep(1, times = 10), col = mypal)
```

```
## [1] "#FF0000" "#FF2400" "#FF4900" "#FF6D00" "#FF9200" "#FFB600" "#FFDB00"
## [8] "#FFFF00" "#FFFF40" "#FFFFBF"
```

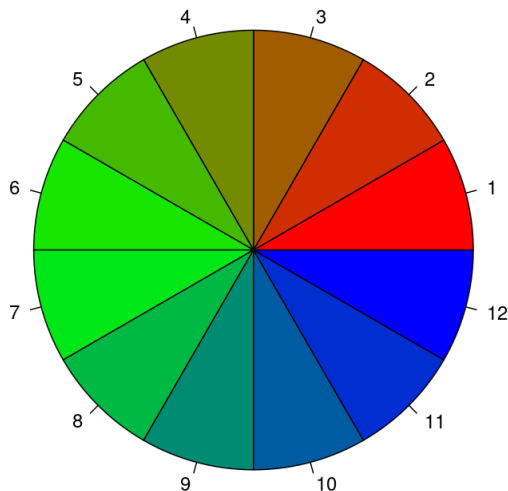


# Custom color palettes

You can easily create custom palettes:

```
mypal <- colorRampPalette(c("red", "green",  
  pie(x = rep(1, times = 12), col = mypal(12))  
  class(mypal)
```

```
## [1] "function"
```



**Note:** `grDevices::colorRampPalette()`

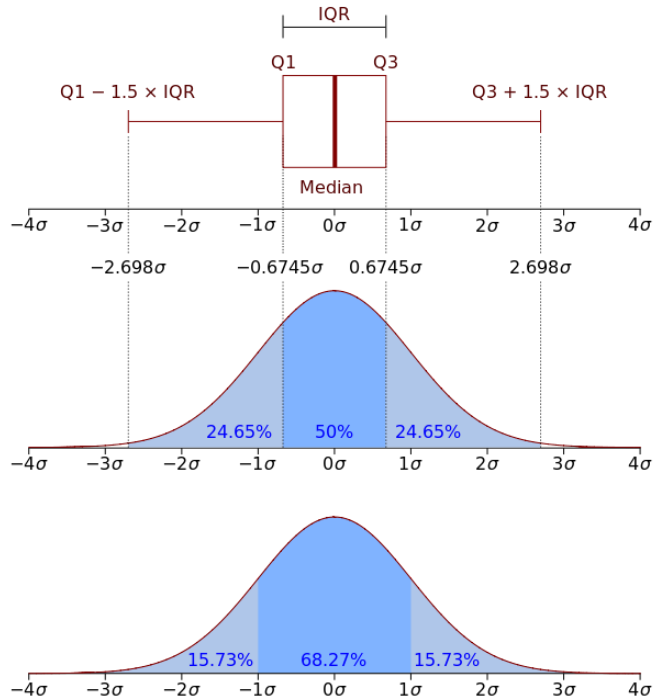
returns a function for generating colors based on the defined custom palette!

There is an excellent package `RColorBrewer` that offers a number of pre-made palettes, e.g. color-blind safe palette.

Package `wesanderson` offers palettes based on Wes Anderson's movies :-)

# Box-and-whiskers plot — theory

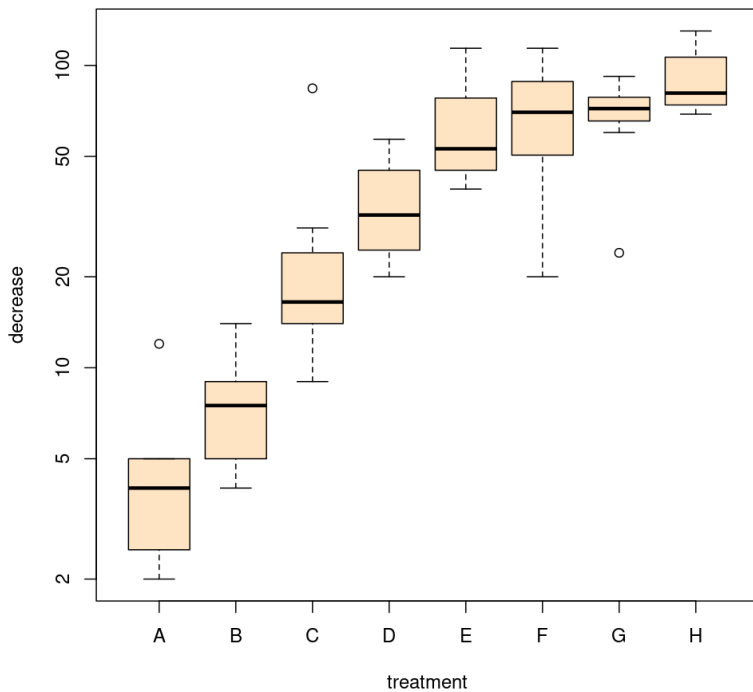
There are also more specialized R functions used for creating specific types of plots. One commonly used is `graphics::boxplot()`



**Rule of thumb.** If median of one boxplot is outside the box of another, the median difference is likely to be significant  $\alpha = 5\%$ .

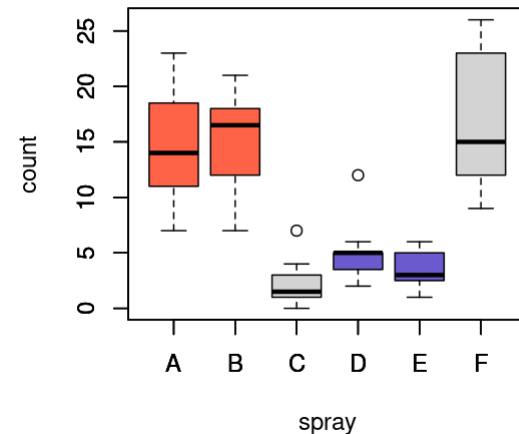
# Box-and-whiskers plot

```
boxplot(decrease ~ treatment,  
data = OrchardSprays,  
log = "y", col = "bisque",  
varwidth=TRUE)
```



The `add=TRUE` parameter in plots allows you to compose plots using previously plotted stuff!

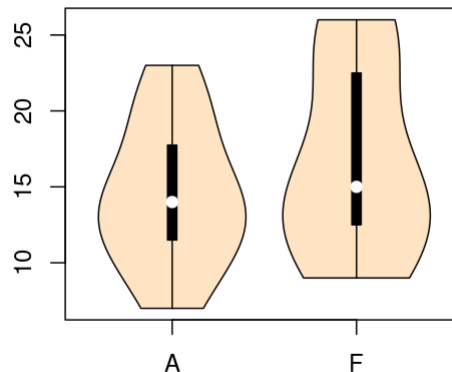
```
attach(InsectSprays)  
boxplot(count~spray, data=InsectSprays[spray  
boxplot(count~spray, data=InsectSprays[spray  
boxplot(count~spray, data=InsectSprays[spray  
detach(InsectSprays)
```



# Violin plots

Package `vioplot` empowers your graphics repertoire with a variation of box-and-whiskers plot called *violin plot*.

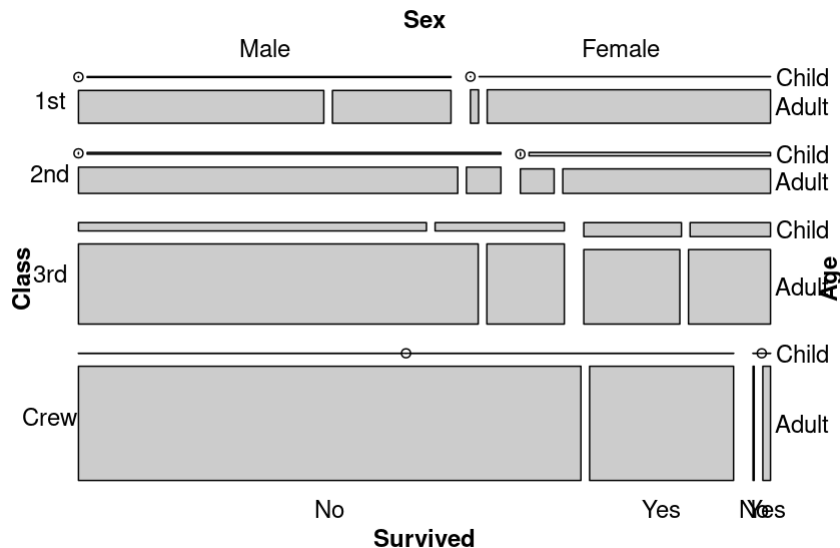
```
attach(InsectSprays)  
vioplot::vioplot(count[spray=="A"],  
count[spray=="F"],  
col="bisque",  
names=c("A", "F"))  
detach(InsectSprays)
```



A violin plot is very similar to a boxplot, but it also visualizes distribution of your datapoints.

# Plotting categorical data

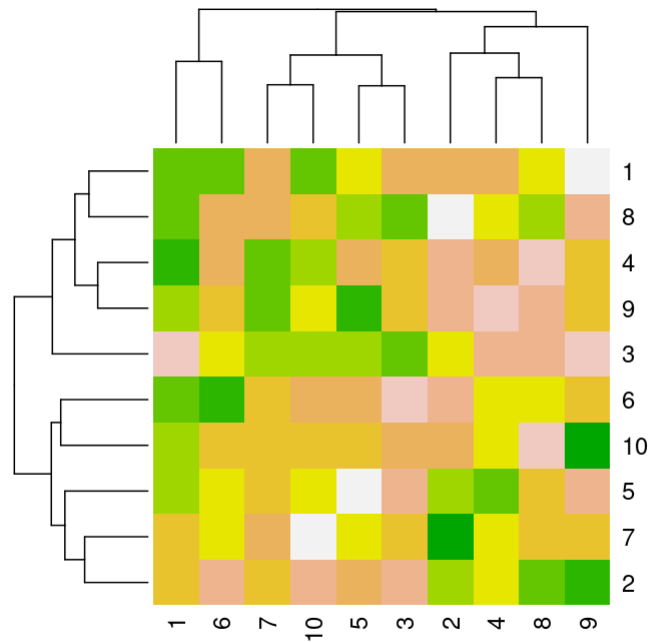
```
data(Titanic) # Load the data
vcd::mosaic(Titanic,
            labeling = vcd::labeling_border(rot_labels = c(0,0,0,0)))
```



Package **vcd** provides a lot of ways of visualizing categorical data.

# Plotting simple heatmaps

```
heatmap(matrix(rnorm(100, mean = 0, sd = 1), nrow = 10),  
col=terrain.colors(10))
```





Thank you! Questions?

Graphics from  freepik.com

Created: 14-Aug-2023 • Roy Francis • SciLifeLab • NBIS

