R for Data Analysis in Agrobiodiversity
3

Data visualization

Recap

- We discussed about the structure of functions
- We saw how to install and use R packages
- We presented the workspace and the working directory
- We saw how to load and write tables

Graphics

- Plots are a mean to effectively visualize your data
- Very useful for exploratory data analysis
- Choosing the right plot is VERY important for outputting your research

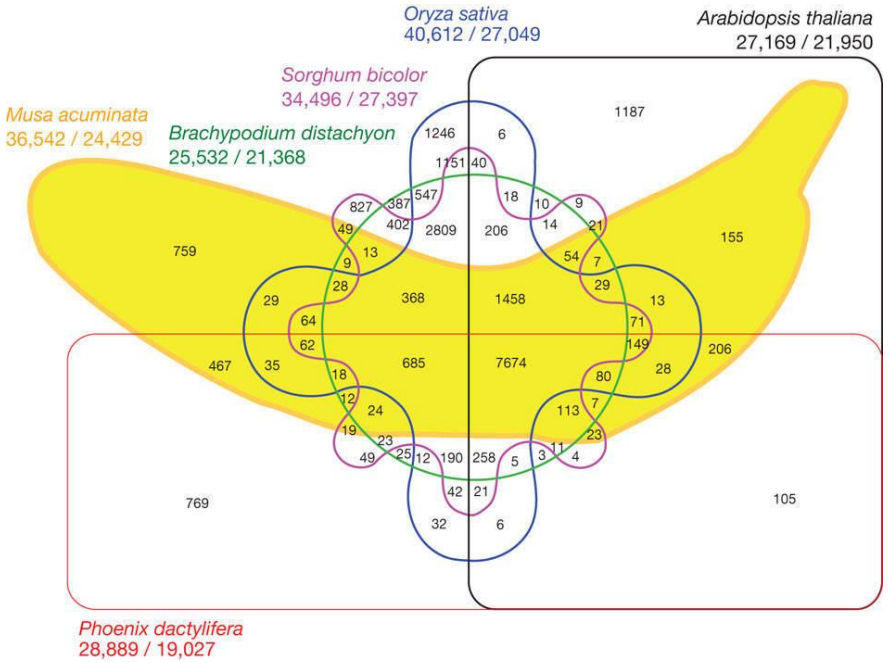


Chart Suggestions—A Thought-Starter

What would you like to show?

Comparison

- Among Items
  - One Variable per Item
    - Many Categories
      - Few Items
      - Many Items
    - Two Variables per Item
      - Variable Width Column Chart
      - Table or Table with Embedded Charts
      - Bar Chart
      - Column Chart
  - Over Time
    - Many Periods
      - Cyclical Data
        - Circular Area Chart
      - Non-Cyclical Data
        - Line Chart
    - Few Periods
      - Single or Few Categories
        - Column Chart
      - Many Categories
        - Line Chart

Relationship

- Two Variables
  - Scatter Chart
  - Bubble Chart
- Three Variables
  - Bubble Chart

Distribution

- Single Variable
  - Few Data Points
    - Column Histogram
  - Many Data Points
    - Line Histogram
- Two Variables
  - Scatter Chart
- Three Variables
  - 3D Area Chart

Composition

- Changing Over Time
  - Few Periods
    - Only Relative Differences Matter
      - Stacked 100% Column Chart
    - Relative and Absolute Differences Matter
      - Stacked Column Chart
  - Many Periods
    - Only Relative Differences Matter
      - Stacked 100% Area Chart
    - Relative and Absolute Differences Matter
      - Stacked Area Chart
- Static
  - Simple Share of Total
    - Pie Chart
  - Accumulation or Subtraction to Total
    - Waterfall Chart
  - Components of Components
    - Stacked 100% Column Chart with Subcomponents

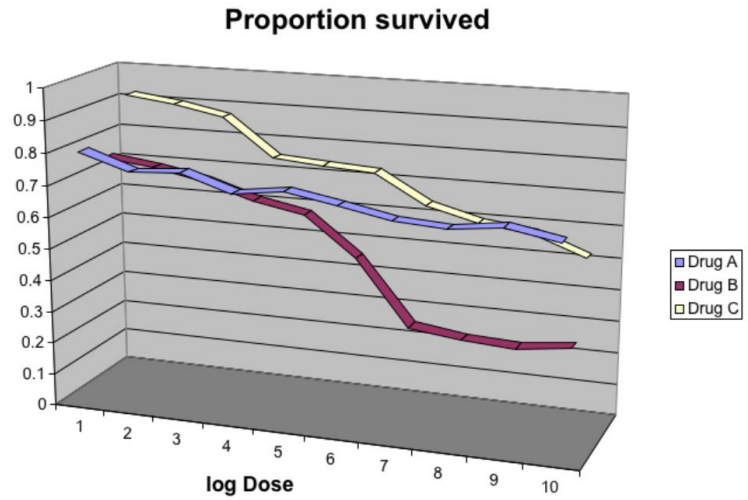
Some guidelines for displaying data

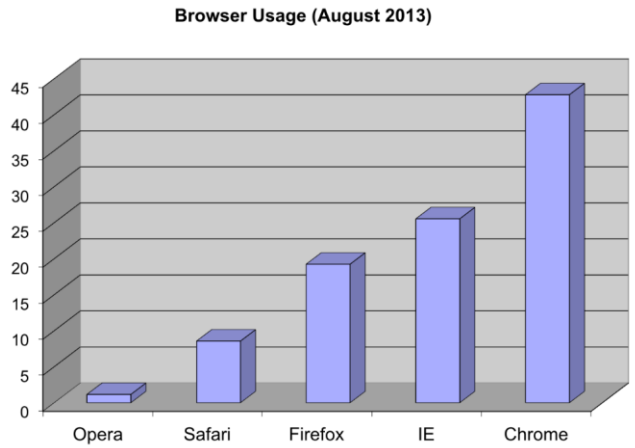
- Think twice before selecting a plot type
- Plots are means, not ends
- Be accurate and clear
- Let the data speak
- Show as much information as possible, taking care not to obscure the message
- Science not sales – Avoid unnecessary frills (esp. flashy colors, gratuitous 3d)
- In tables, every digit should be meaningful

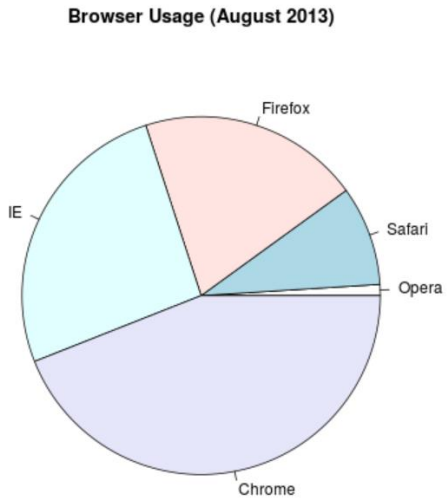
see Karl W. Broman, “How to Display Data Badly”

Coloring/3d

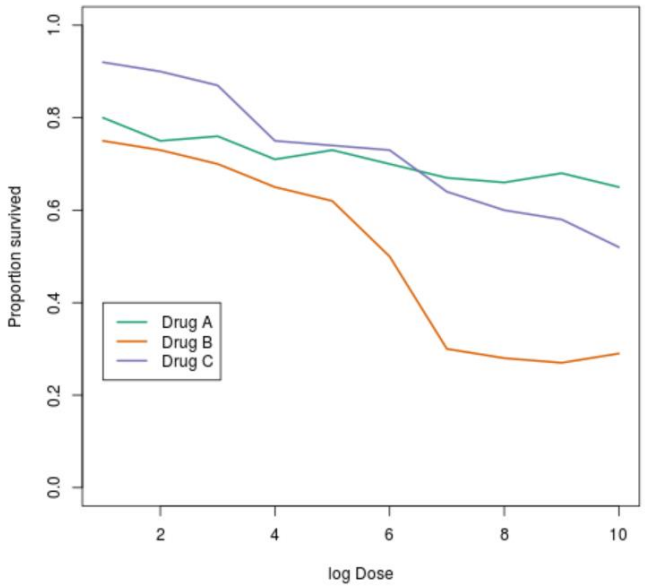
Bad

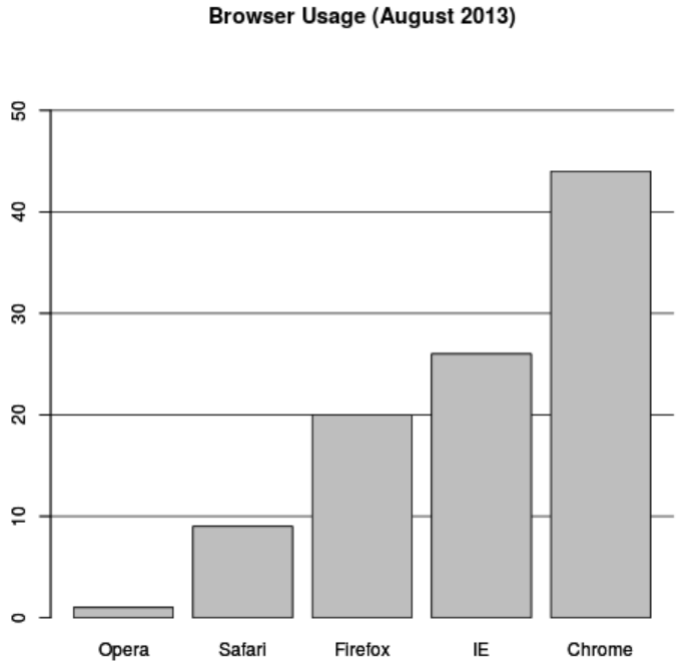






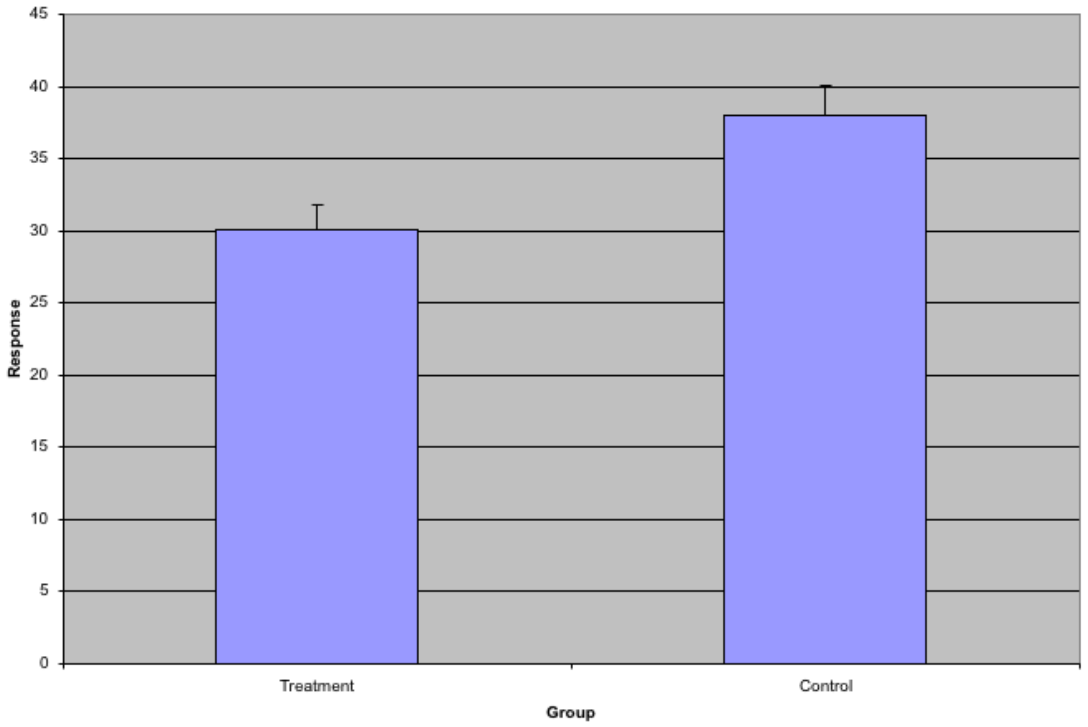
Good



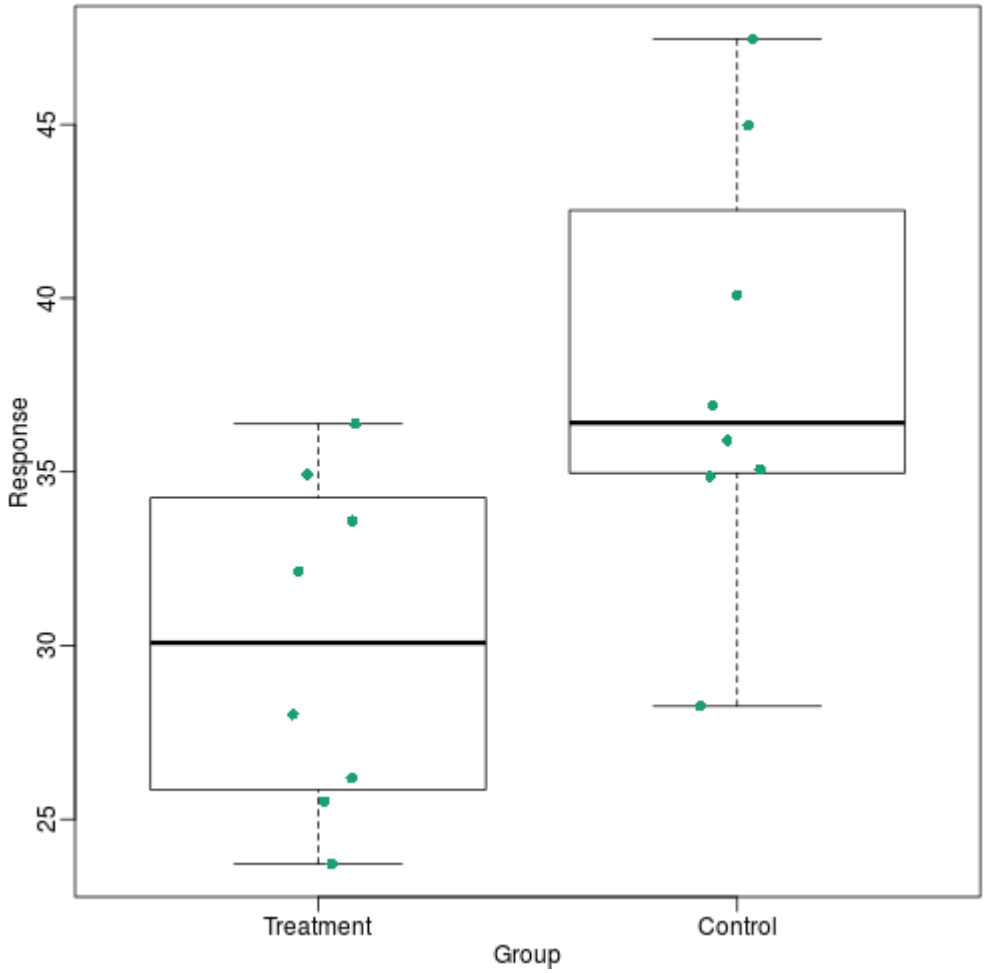


Treatment/control

Bad

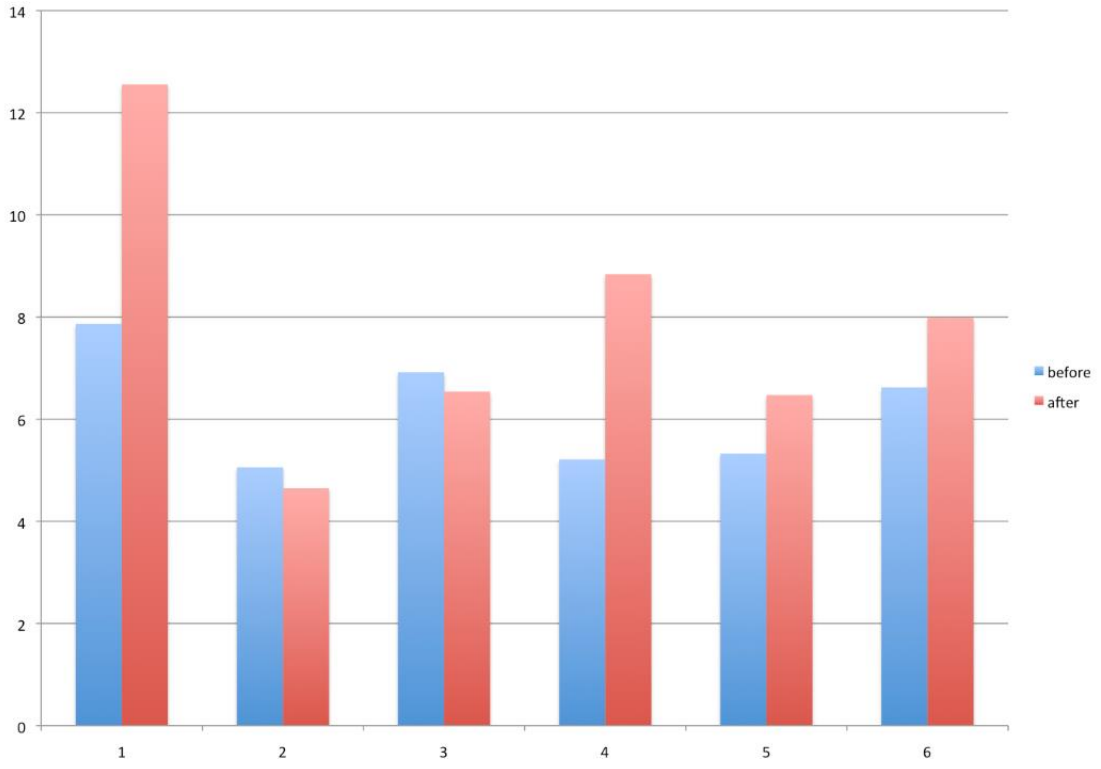


Good

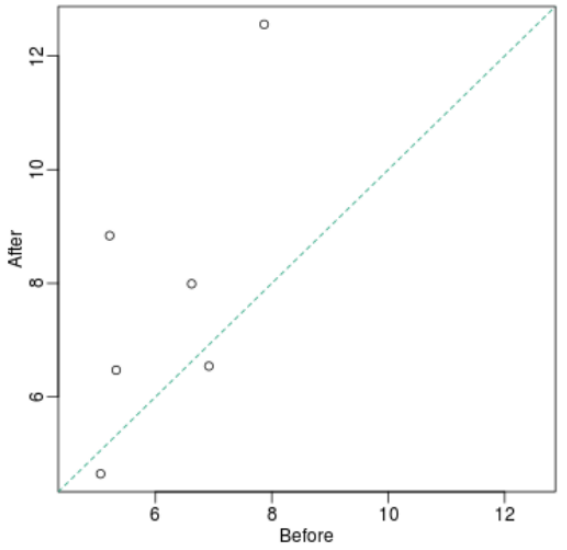


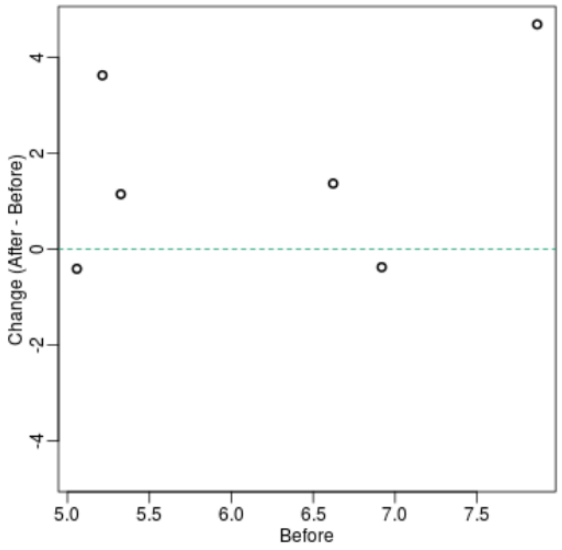
Before/after

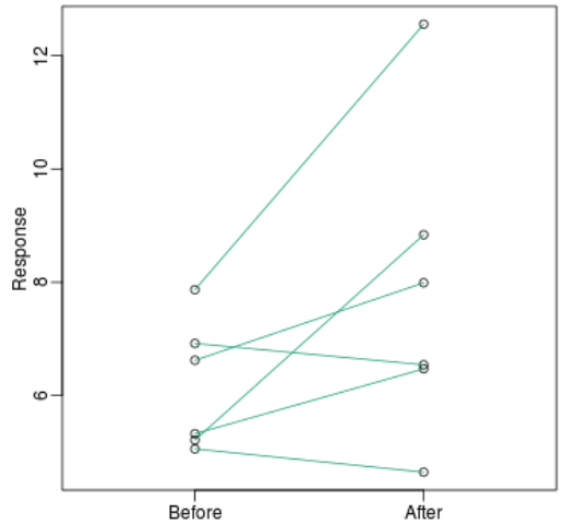
Bad

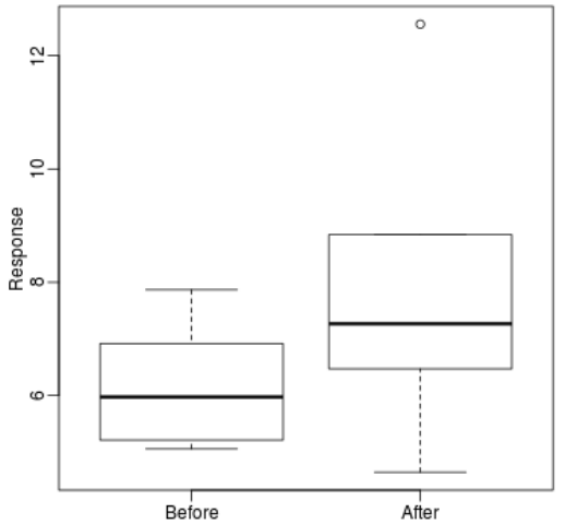


Good

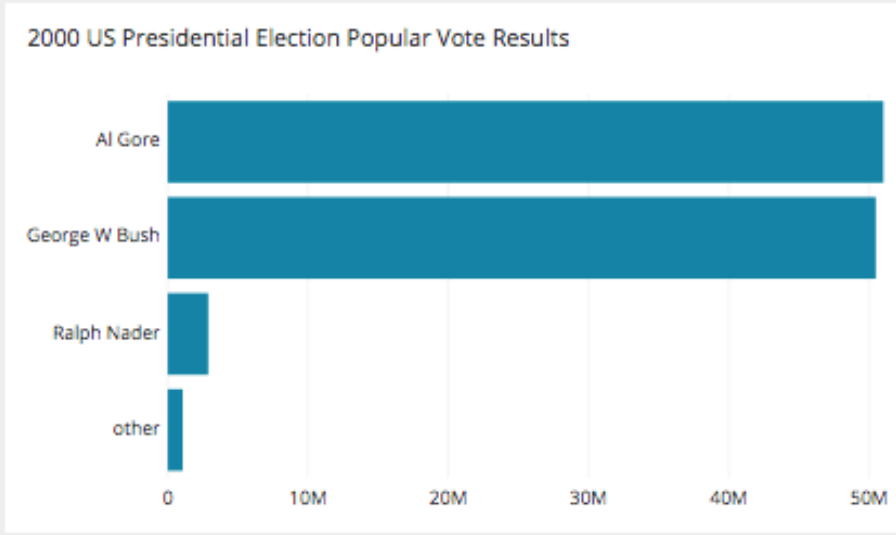


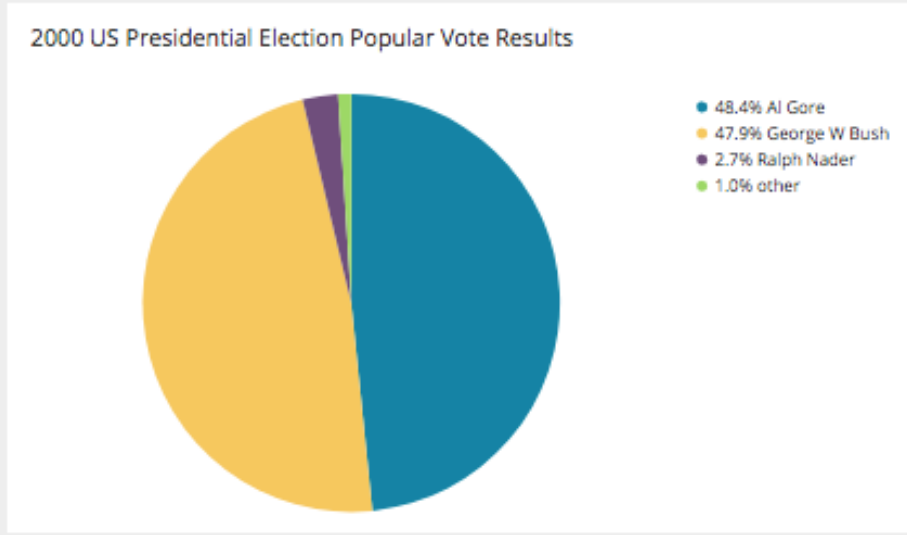


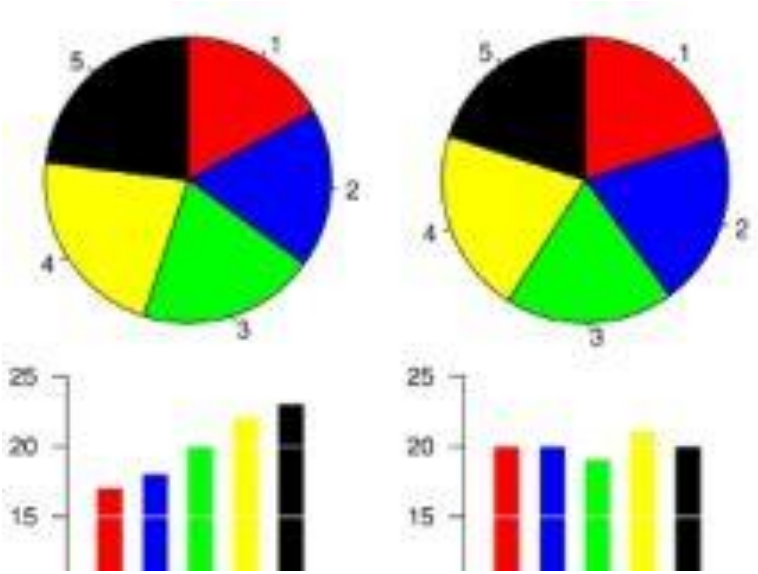




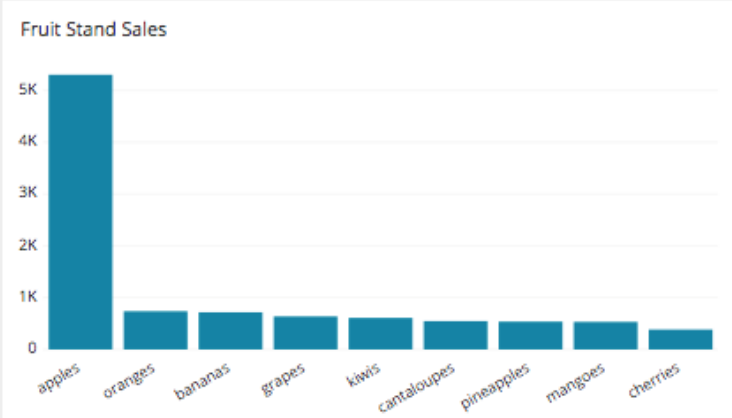
Sorting







Partitioning



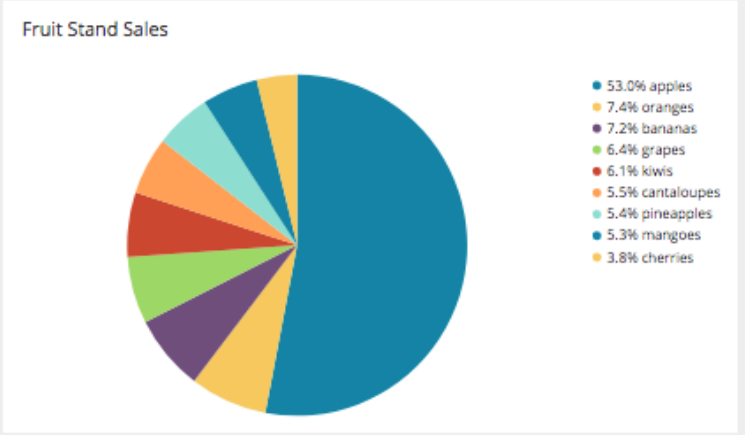


Table data (heights here in inches)

Bad table

team 1 76.39843 76.21026 81.68291 75.32815 77.18792
team 2 74.14399 71.10380 80.29749 81.58405 73.01144
team 3 71.51120 69.02173 85.80092 80.08623 72.80317
team 4 78.71579 72.80641 81.33673 76.30461 82.93404
team 5 73.42427 73.27942 79.20283 79.71137 80.30497
team 6 72.93721 71.81364 77.35770 81.69410 80.39703
team 7 68.37715 73.01345 79.10755 71.24982 77.19851
team 8 73.77538 75.59278 82.99395 75.57702 87.68162

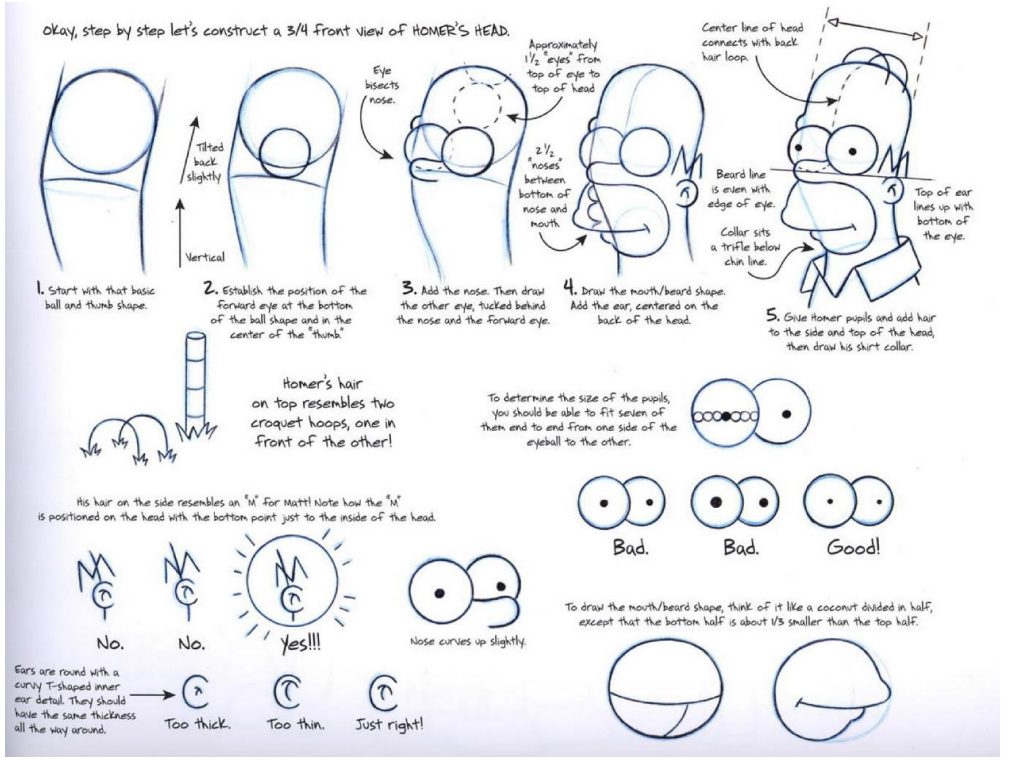
Good table

team 1 76.4 76.2 81.7 75.3 77.2
team 2 74.1 71.1 80.3 81.6 73.0
team 3 71.5 69.0 85.8 80.1 72.8
team 4 78.7 72.8 81.3 76.3 82.9
team 5 73.4 73.3 79.2 79.7 80.3
team 6 72.9 71.8 77.4 81.7 80.4
team 7 68.4 73.0 79.1 71.2 77.2
team 8 73.8 75.6 83.0 75.6 87.7

Plotting with R

Plotting in R is somewhat similar to what you would do with pen and paper

1. Choose the *graphical device*. This is the support of for what you are going to draw
2. Choose the area in which you do the plot
3. Decide the type of plot you want
4. Add features
5. Finalize the plot



The support of your plot is a **graphical device**

- By default, the graphical device is the computer screen. It is also called «device 0» this means that your plot will be printed on screen, and erased as soon as another plot is called
- You can specify a different device, most notably a file. In this case the plot will be saved in the specified location on your hard disk
- Every time you open a device, you have to close it to let R know that you are done drawing
- Devices can be called in series and be contained one in another, and will assume sequentila numbers from «device 1» to «device n»

Graphical devices specify which format you want your plot to be saved in

pdf(«filename.pdf»)

png(«filename.png»)

tiff(«filename.tif»)

Arguments include file name (mandatory), definition, aspect ratio, height, width

Once the plot is done, you need to close the graphic device using *dev.off()*:

```
> pdf("newplot.pdf", height=8, width=5)
> plot(x,y)
> dev.off()
null device
          1
```

When you are plotting on a device other than screen:

- You will not be able to see the plot until you close the device
- You will add to the plot until you close the device
- You will overwrite any file having the same name of the device
- If you do fail to close the device, you will open devices one in another and jeopardize your plotting effort

- A plot produced on screen may be resized and saved using the Rstudio GUI
- png(), bmp(), tiff(), jpeg() are raster/bitmap devices.
  - Good for images, good for sharing if compressed.
  - Highest quality (but bigger file size) is obtained with tiff format.
  - Many arguments to define size and definition (dpi)
- svg(), pdf() are vector devices.
  - Good for representing graphs
  - Based on geometry, not on pixels
  - Resource intensive for exceedingly complex images
  - Good to produce high definition plots (not dependent on scale)

Choice of device depends on the scope of plotting: for internal use, low definition is ok. Most journals require high definition images for publications (600 dpi and up)

The area in which to plot may be determined by graphical parameters («par»)

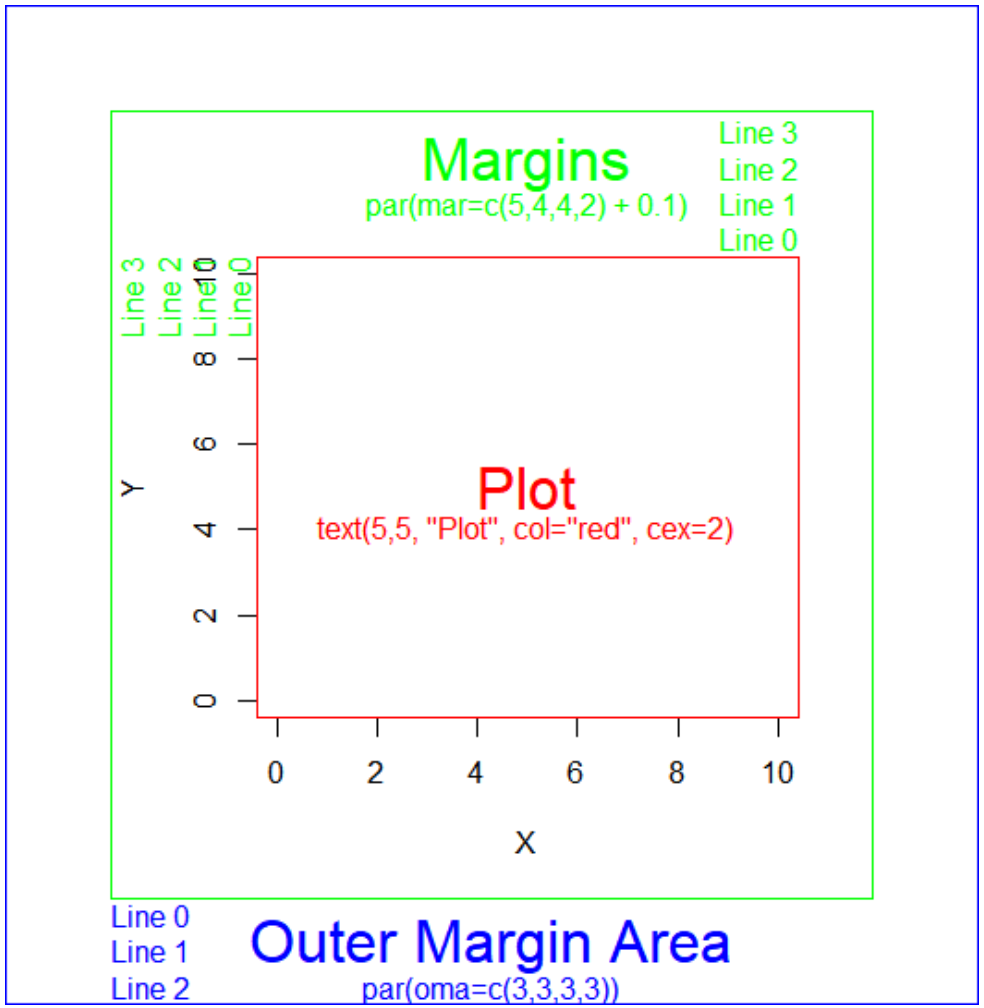
Graphical parameters may be indicated by the *par()* function before calling *plot()*

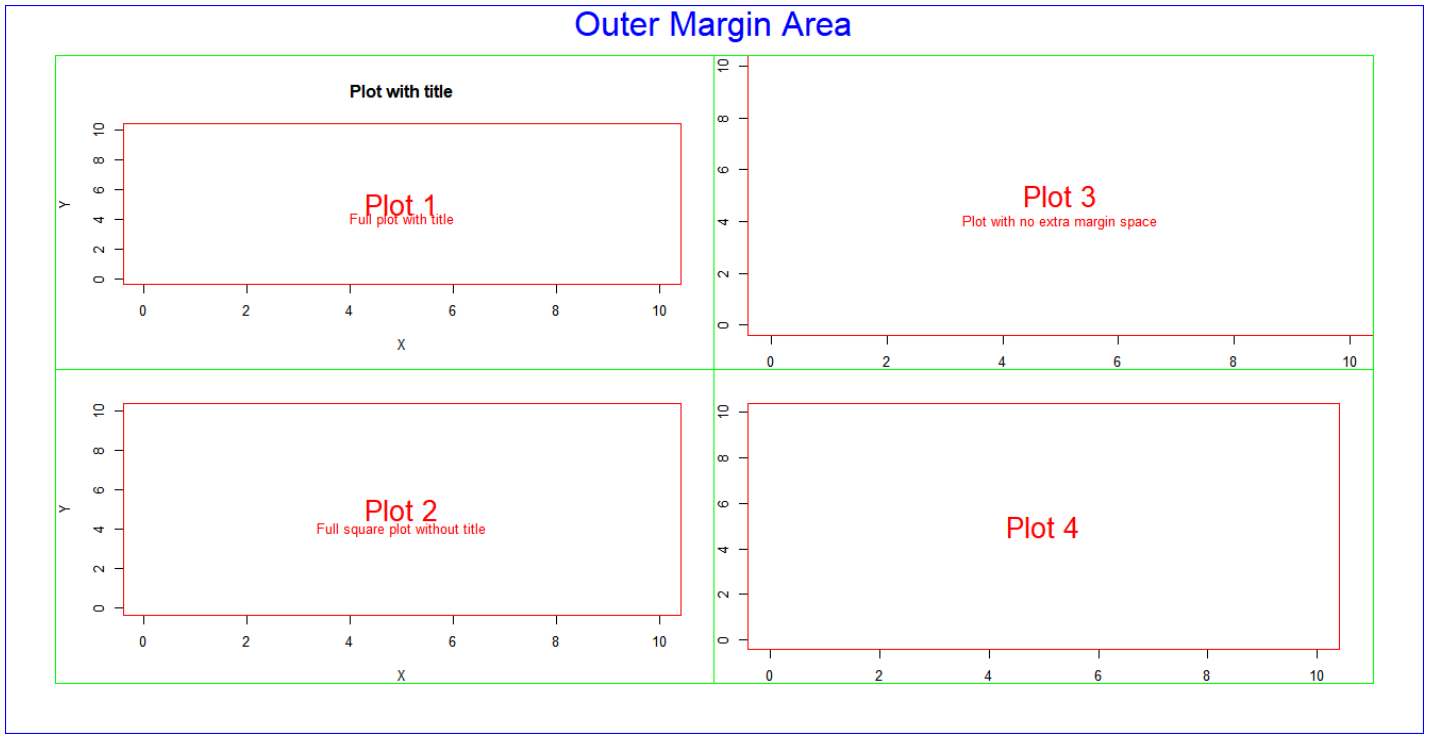
par(mai=c(2, 2, 1, 1), pin=c(5, 4), mfrow=c(1,2))

Margin size in inches

Plot size in inches

Plot arrangement (if multiple)





Source
http://rgraphics.limnology.wisc.edu

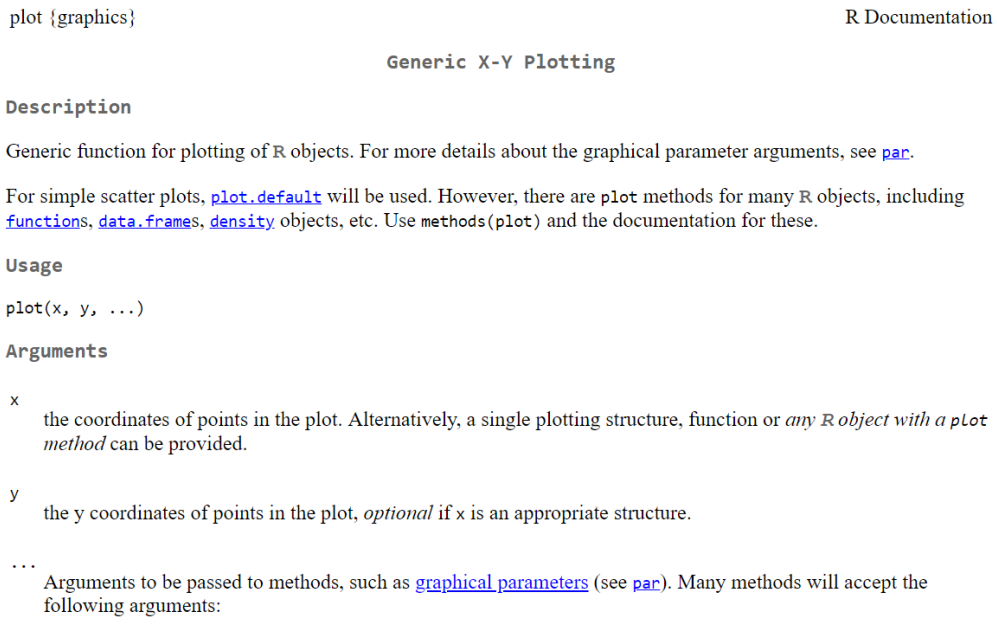
Note that R will use defaults in case parameters are not set (most cases for basic plotting)

Once you have a device and a plot area (both may be default), it is **time to plot**!

The pivotal function for X Y plotting is (not very surprisingly)

plot()

Very flexible function with a lot of customization



plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

Usage

```
plot(x, y, ...)
```

Arguments

x
the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any R object with a* plot *method* can be provided.

y
the y coordinates of points in the plot, *optional* if x is an appropriate structure.

...
Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

Most arguments are default in the function; you may change them at will

- «x» and «y» are not default, and need to be specified
- «main» defines the title
- «xlim» and «ylim» define the axis ranges (in data units)
- «type» defines the feature in the plot (points, lines, …)
- «ylab» and «xlab» define the axis lables
- «las» controls lables rotation
- «col» defines the color(s) to be given to features in the plot
- «pch» defines the symbols to be used
- «cex» defines the size of the symbols

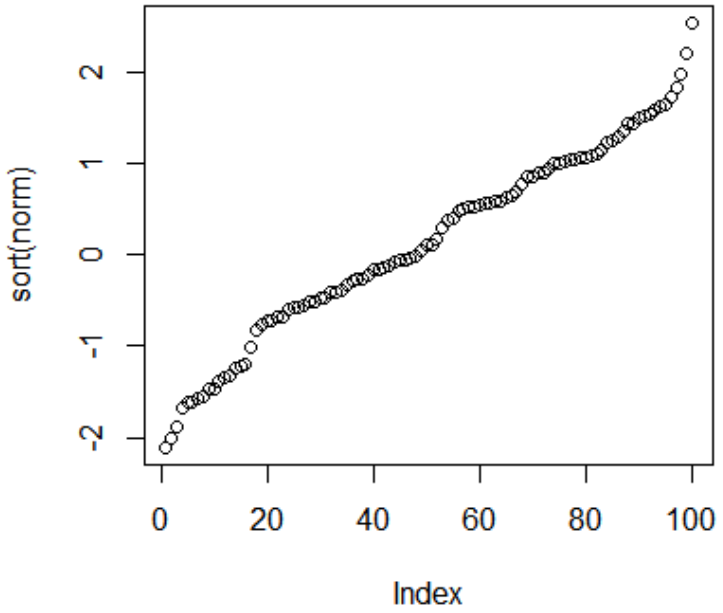

pch values and corresponding displays
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260
ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ
221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð
201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü
181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À Á Â Ã Ä Å Æ Ç È
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´
141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
Ž ‘ ’ “ ” • – — ˜ ™ š › œ ž Ÿ
121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
y z { | } ~ € ‚ ƒ „ … † ‡ ˆ ‰ Š ‹ Œ
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
e f g h i j k l m n o p q r s t u v w x
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Q R S T U V W X Y Z [\ ] ^ _ ` a b c d
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
= > ? @ A B C D E F G H I J K L M N O P
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; <
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
! " # $ % & ' (
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

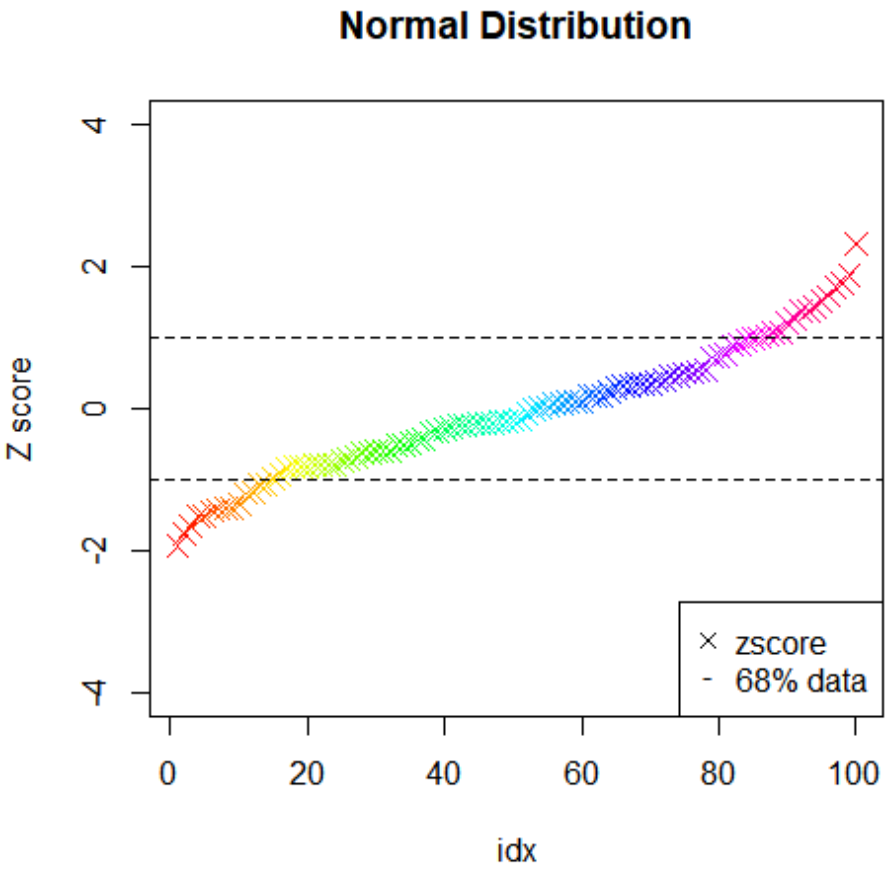
Adding features

Once a plot is done, calling again the plot function will overwrite it. You may add specific features on the same plot using these functions:

- axis() to add axes
- points(), lines(), arrows(), polygons(), to add points, lines, …
- abline() to add straight lines with specific intercept and slope
- text() to add text in specific coordinates
- legend(), **always to be used**

```
norm<- rnorm(100)
plot(sort(norm))
```






```
plot(sort(norm), col=rainbow(length(norm)),  xlab="idx", ylab="Z score",
      main="Normal Distribution", ylim=c(-4,4) , pch=4, cex=1.5)
abline(h=c(-1,1), lty=2)
legend("bottomright", legend=c("zscore","68% data"), pch=c(4, 45))
```

Some useful plot types

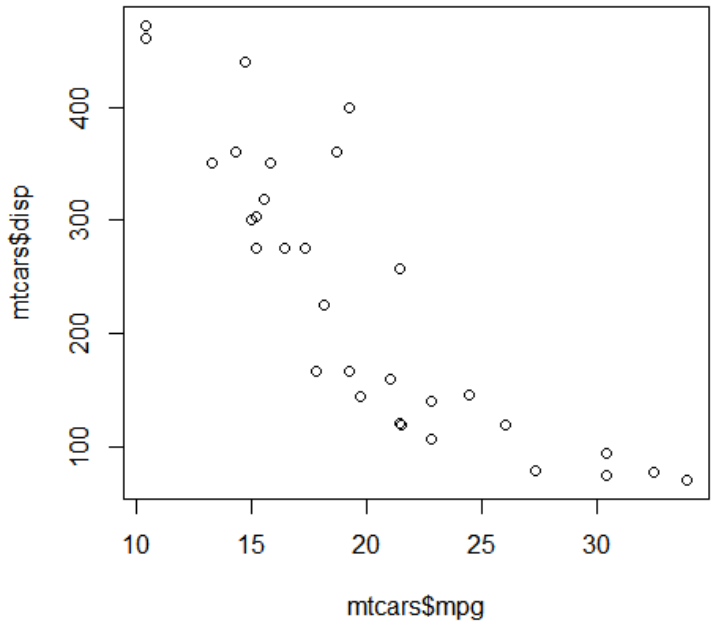
- XY plot
- Histograms
- Barplots
- Boxplots
- Heatmaps

```
data(mtcars)
```

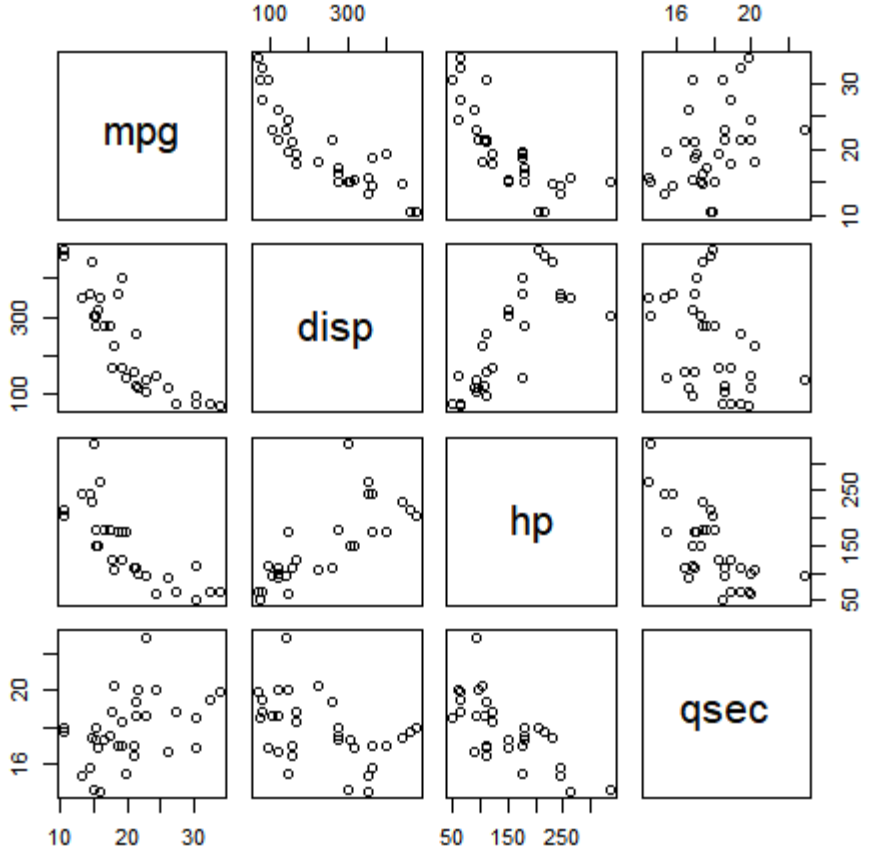
mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21	6	160	110	3.9	2.62	16.46	0	1	4	4
21	6	160	110	3.9	2.875	17.02	0	1	4	4
22.8	4	108	93	3.85	2.32	18.61	1	1	4	1
21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
18.1	6	225	105	2.76	3.46	20.22	1	0	3	1

xyplot

plot(mtcars$mpg, mtcars$disp)

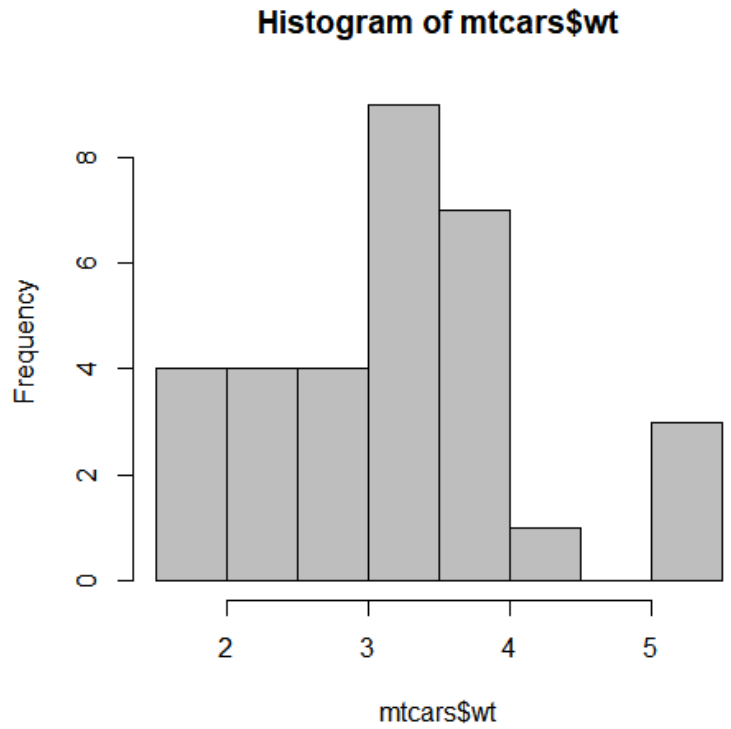


pairs(mtcars[,c(1,3,4,7)])

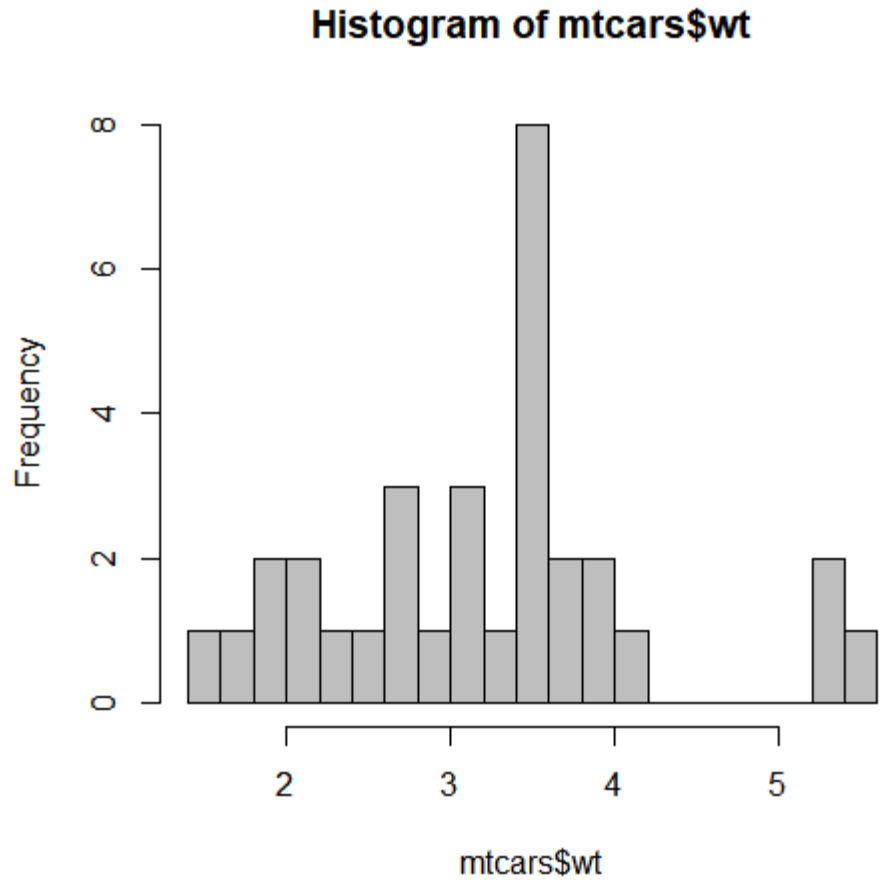


Histogram

```
hist(mtcars$wt, col="gray")
```

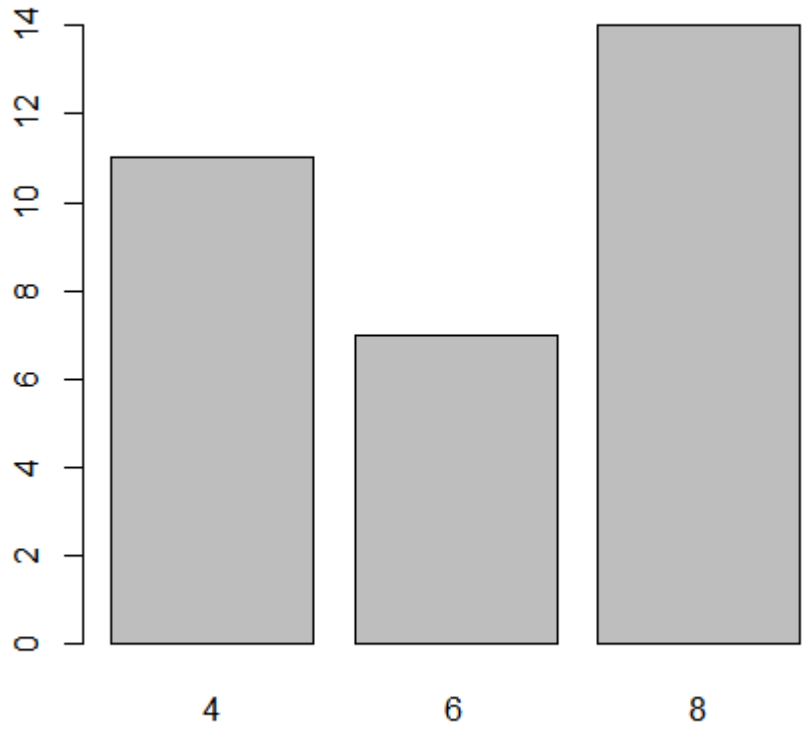



```
hist(mtcars$wt, breaks=20, col="gray")
```

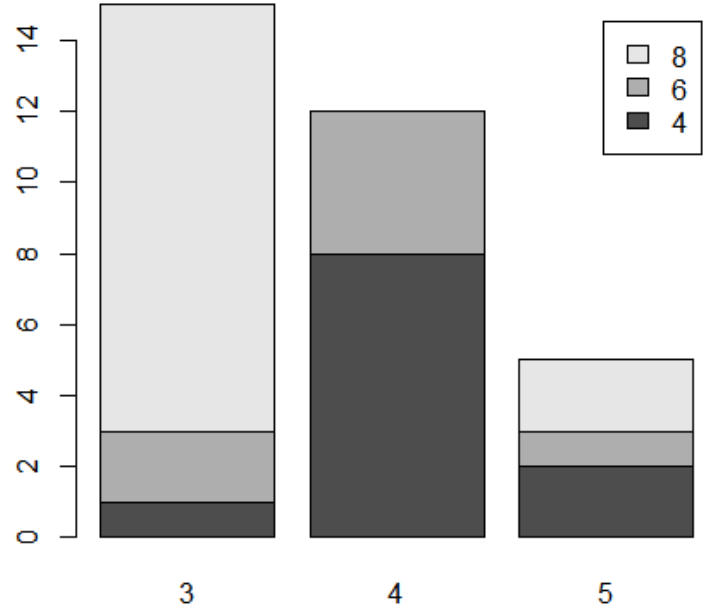


Barplot

```
counts <- table(mtcars$cyl)
barplot(counts)
```



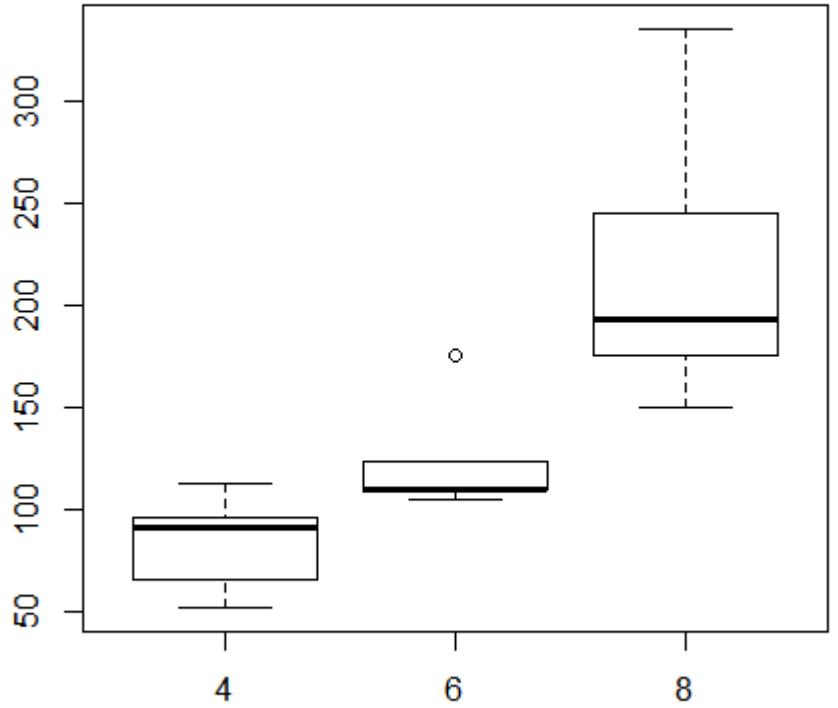

```
counts <- table(mtcars$cyl, mtcars$gear)
barplot(counts, legend = rownames(counts))
```



Boxplots

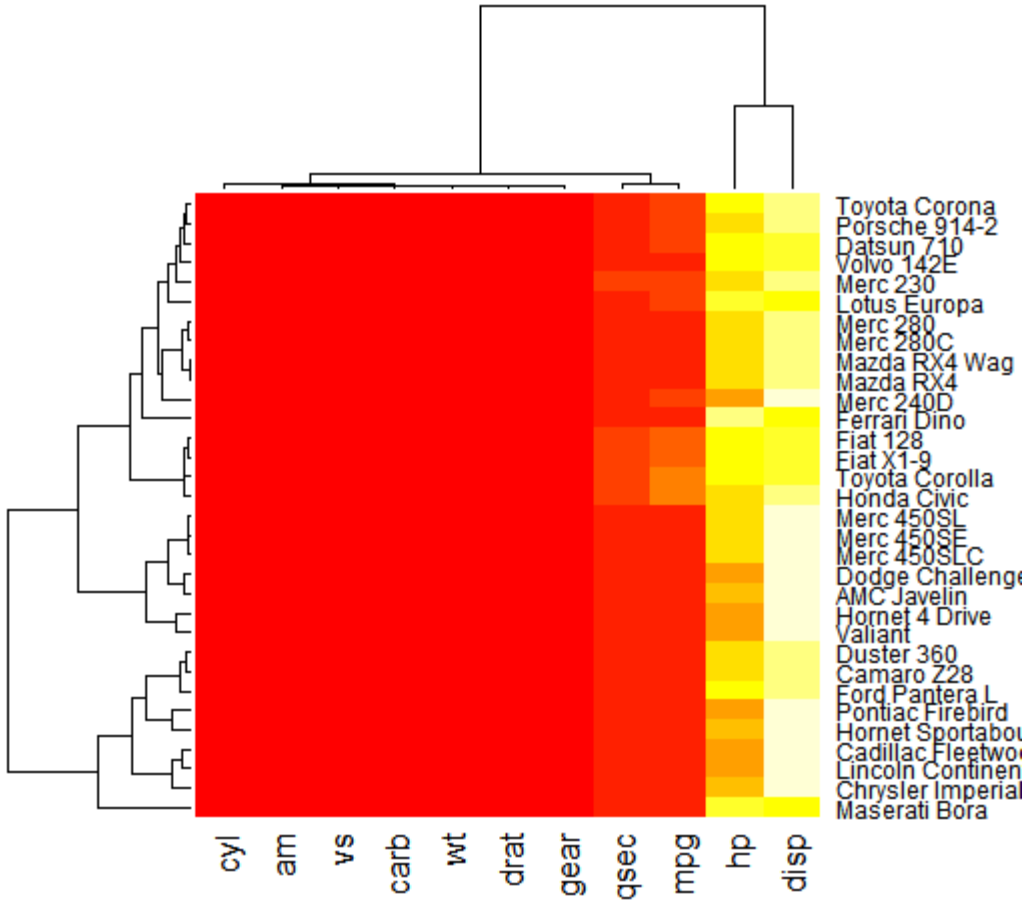
- Watch out for data types! Grouping variables are expected to be factors

```
boxplot(mtcars$hp ~ as.factor(mtcars$cyl))
```

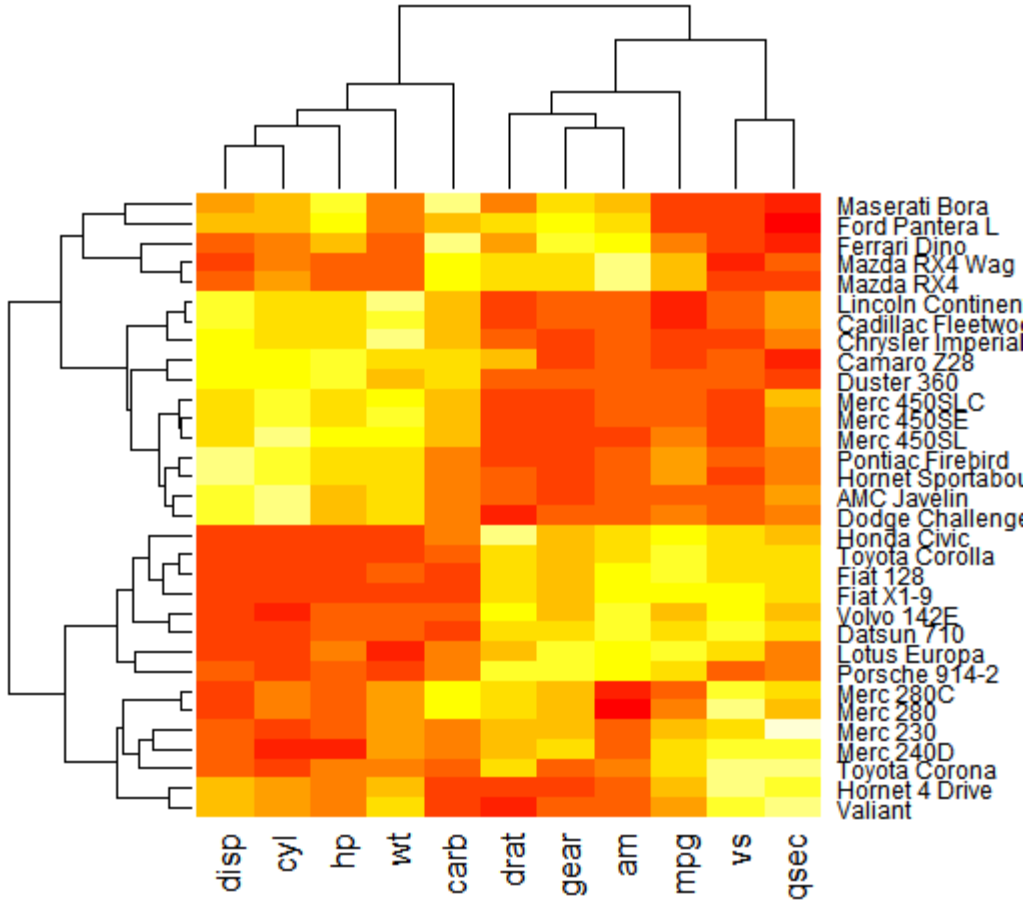


Heatmaps

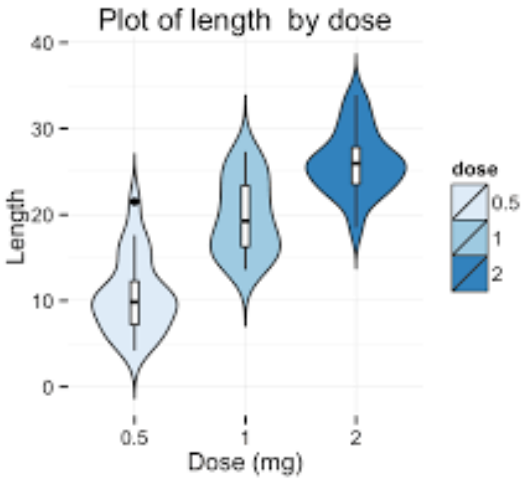
heatmap(as.matrix(mtcars))

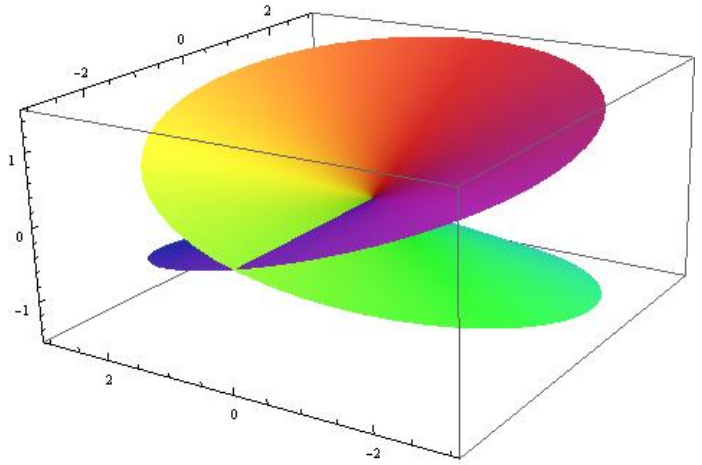


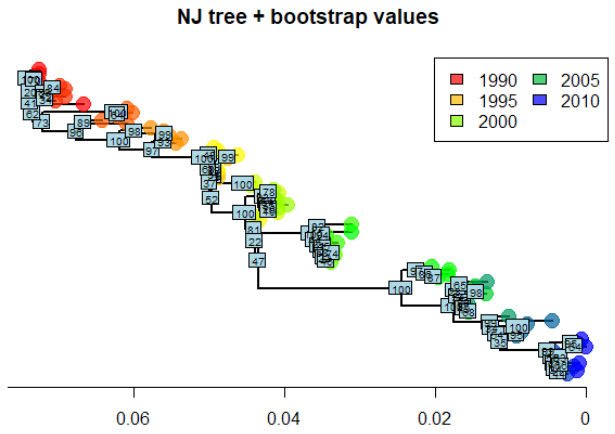
heatmap(scale(as.matrix(mtcars)))

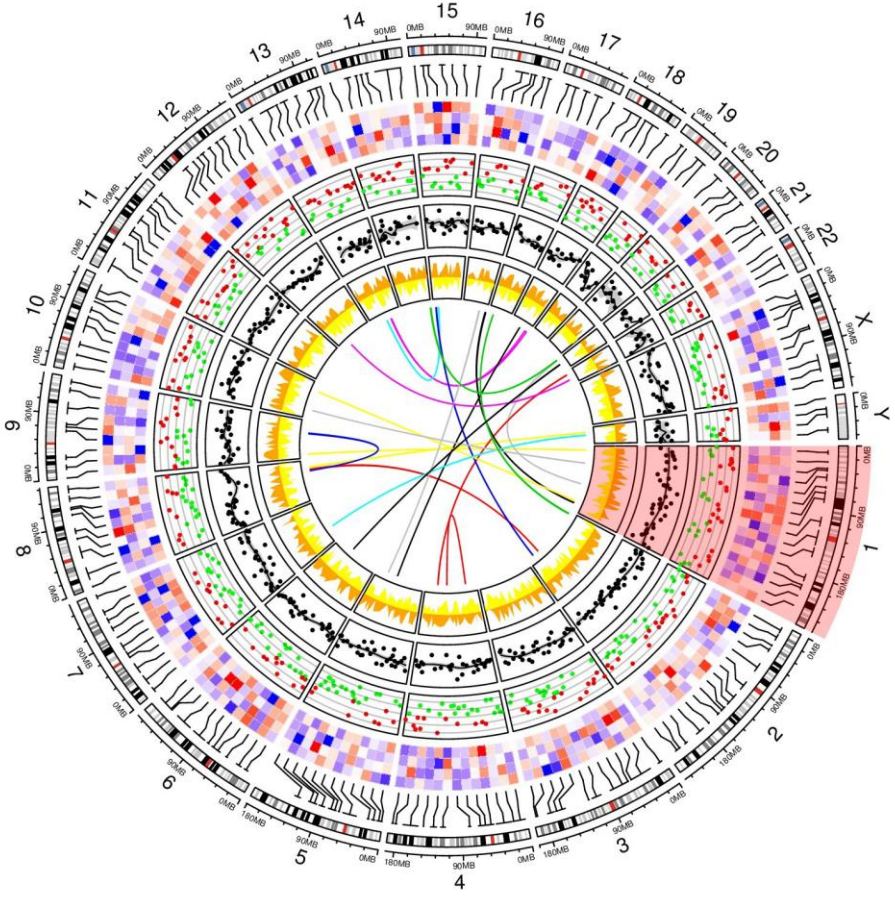


Many others (lots of packages)



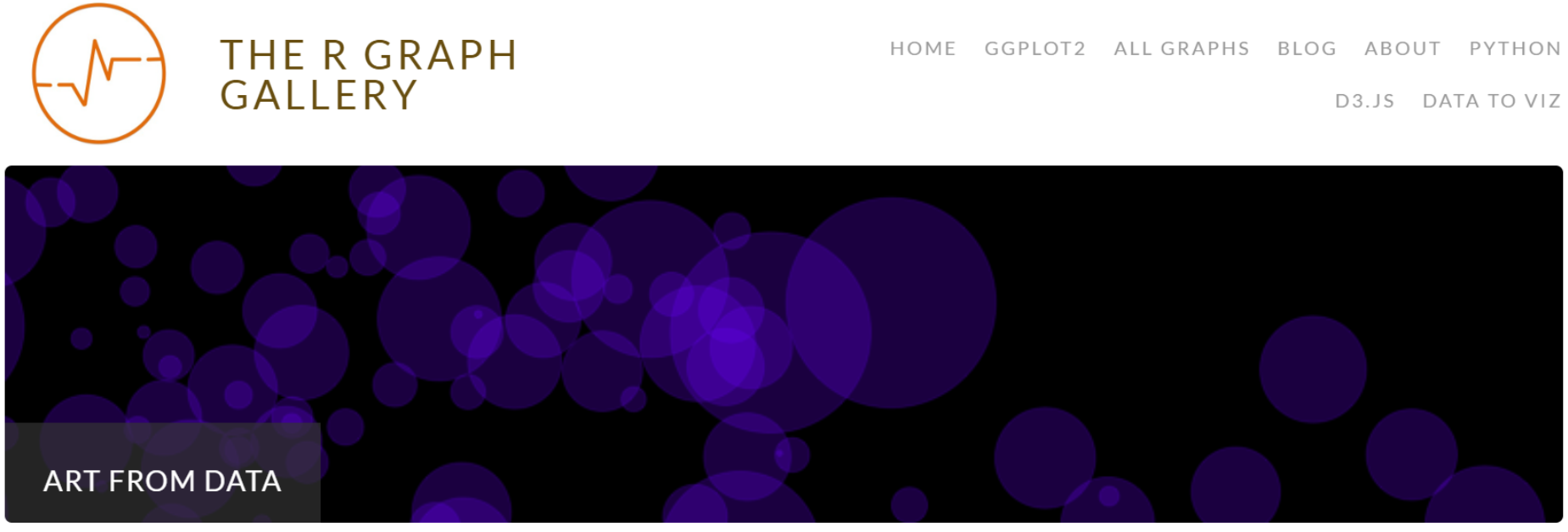




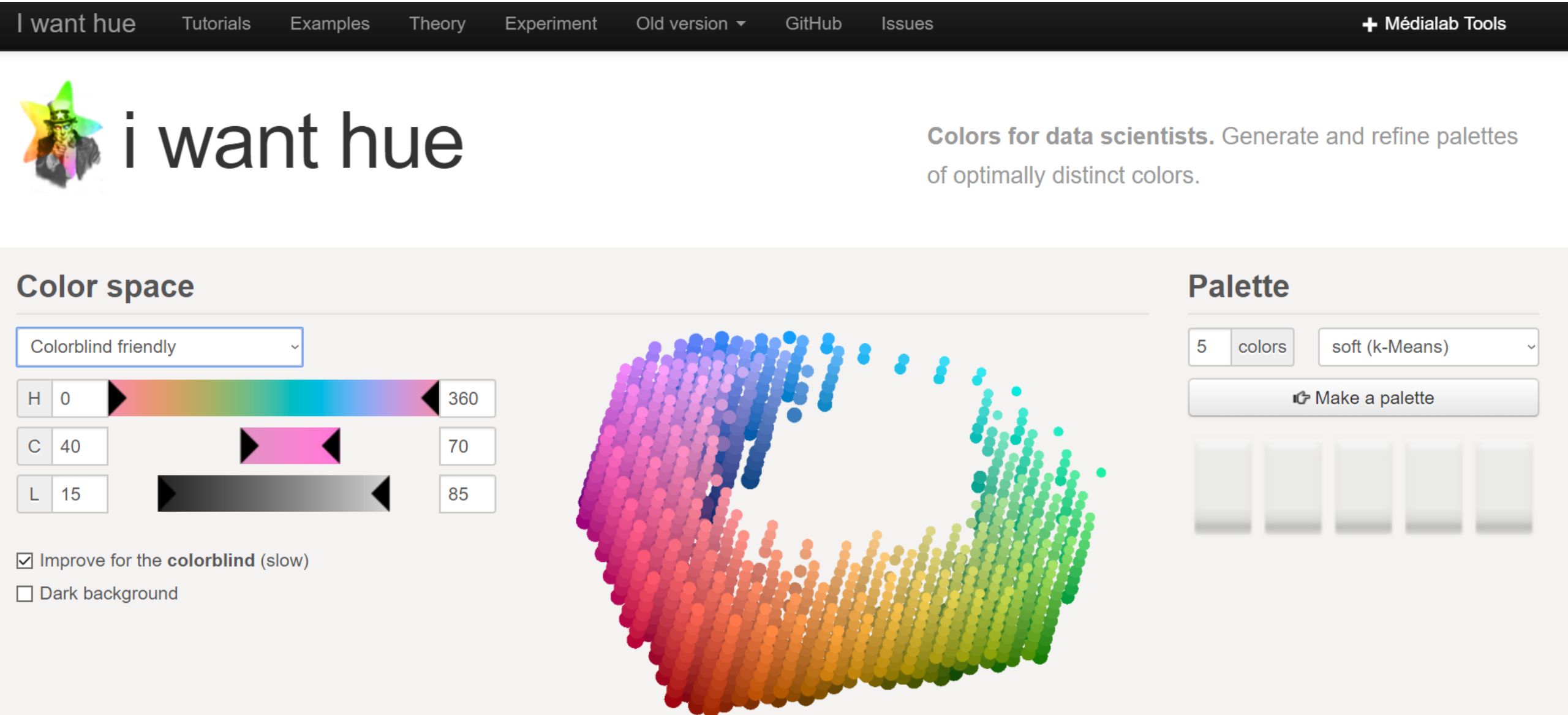


In case you need inspiration

https://www.r-graph-gallery.com



http://tools.medialab.sciences-po.fr/iwanthue/



A different (better) approach: ggplot

- Implementation of the grammar of graphics (gg)
- Package to be installyed, part of the tidyverse (tidyverse.org)
- By Hadley Wickham (http://hadley.nz)

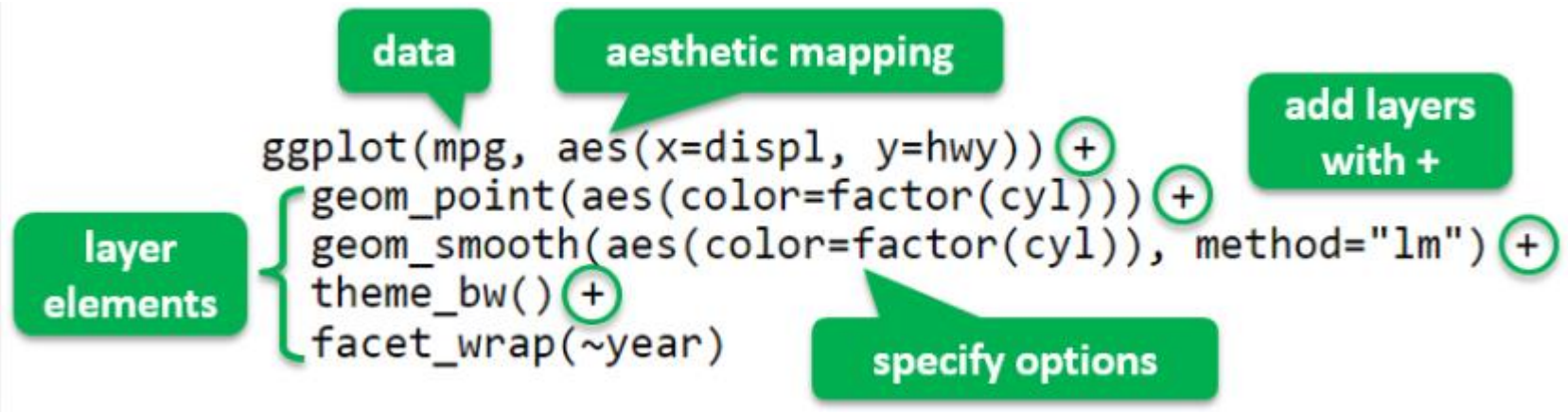
“In brief, the grammar tells us that a statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also contain statistical transformations of the data and is drawn on a specific coordinate system. Facetting can be used to generate the same plot for different subsets of the dataset. It is the combination of these independent components that make up a graphic.”

Components of the grammar of graphics

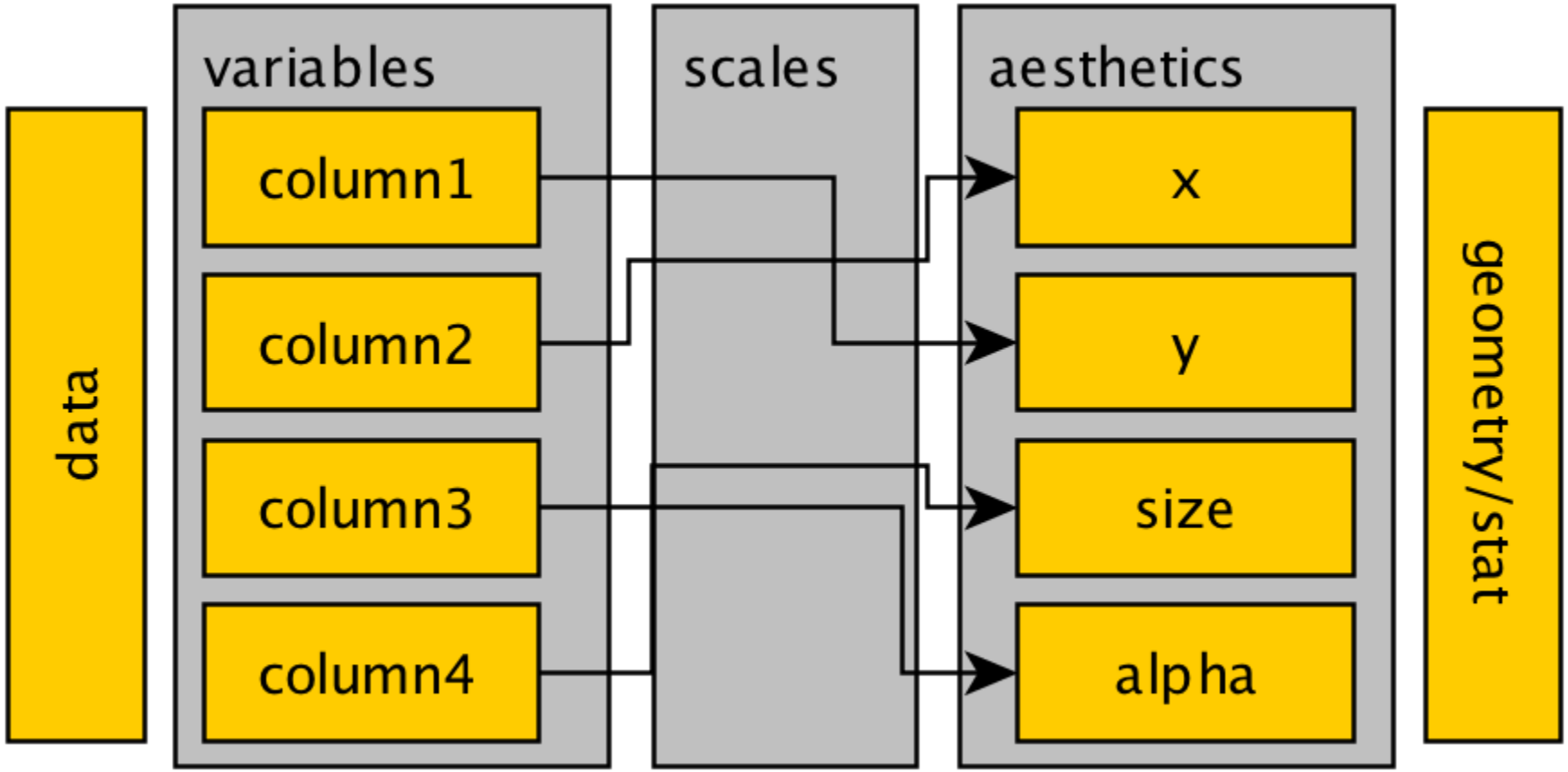
- Plots are made of layers
- Layers are made by mapping *values* in dataframe columns to *aesthetics*
- *Aesthetics* are properties that can visually express differences between values, for example:
  - *x*-position
  - *y*-position
  - shape
  - color
  - size
  - …
- *Values* are portrayed drawing a *geometric object* whose appearance and placement in space is dictated by the mapping of values to aesthetics.

Components of a ggplot

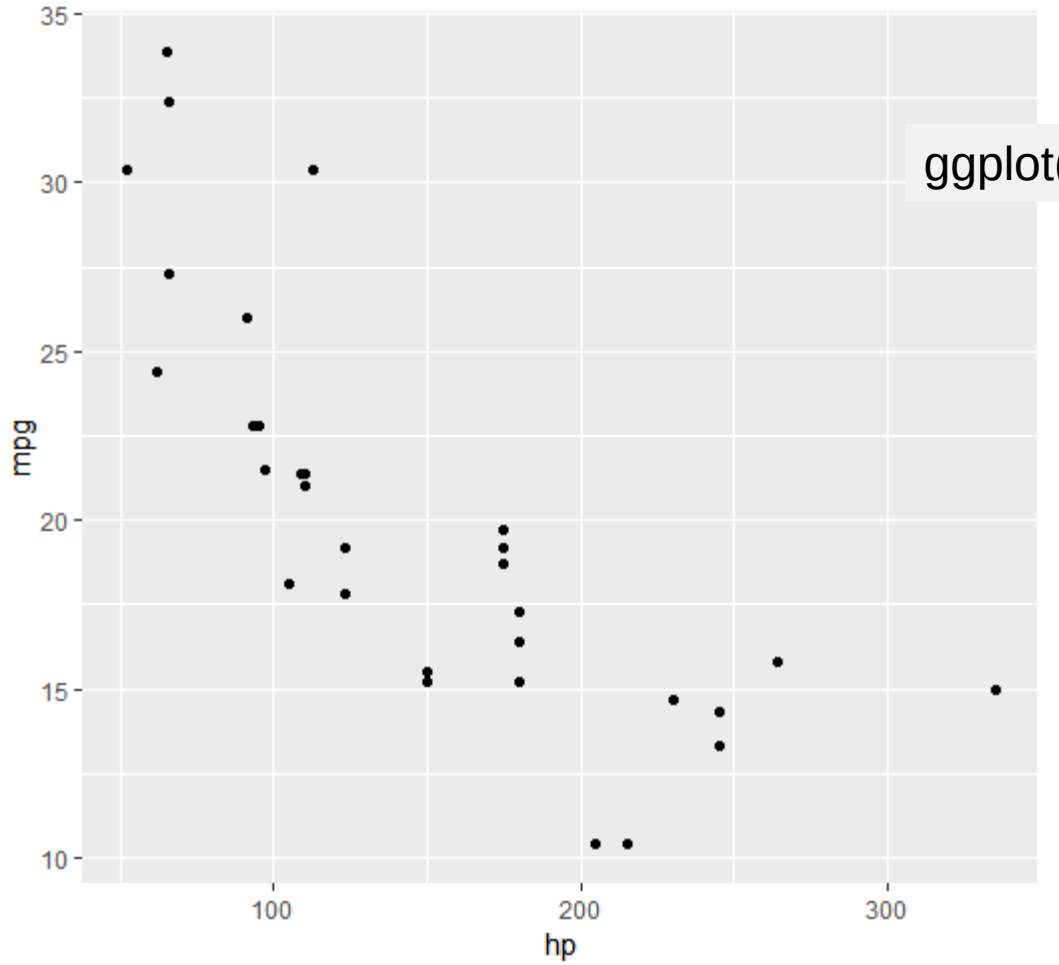
1. Data: a data.frame to visualize
2. Aesthetics: mapping varialbes of the data to aesthetic attributes (position, size, shape, color, fill, transparency, …)
3. Scales: mapping values of the data to visual values for each aesthetic (e.g. position, color, fill and shape scales)
4. Geometric objects: point, line, polygon, histogram, quantile, bar, …
5. Statistical transformations: bin, boxplot, density, contour, function, …
6. Coordinate system: Cartesian, polar, map projection, …
7. Facet: display split data in multi-panels (aka conditioning)
8. Theme: control non-data visual elements (title, axes, tick, …)



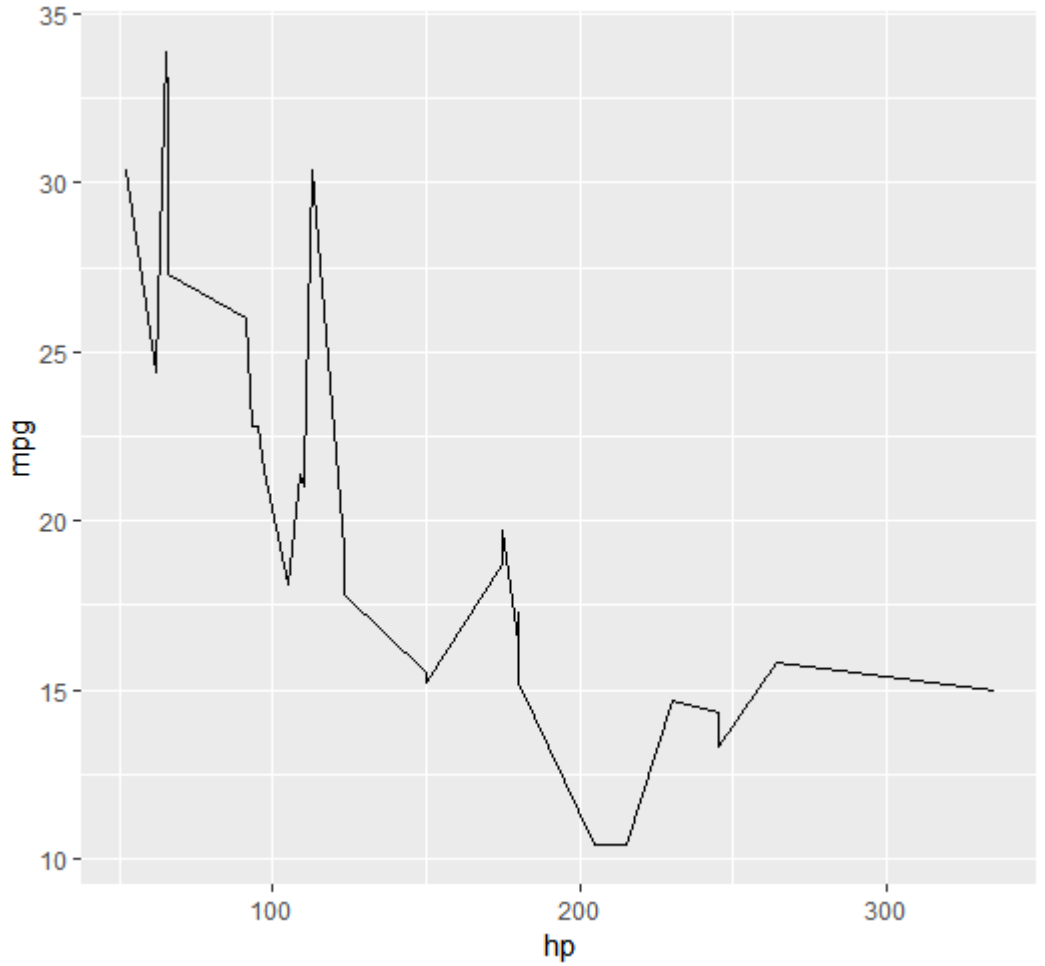
The minimum requirement to make a ggplot are data and a geometry



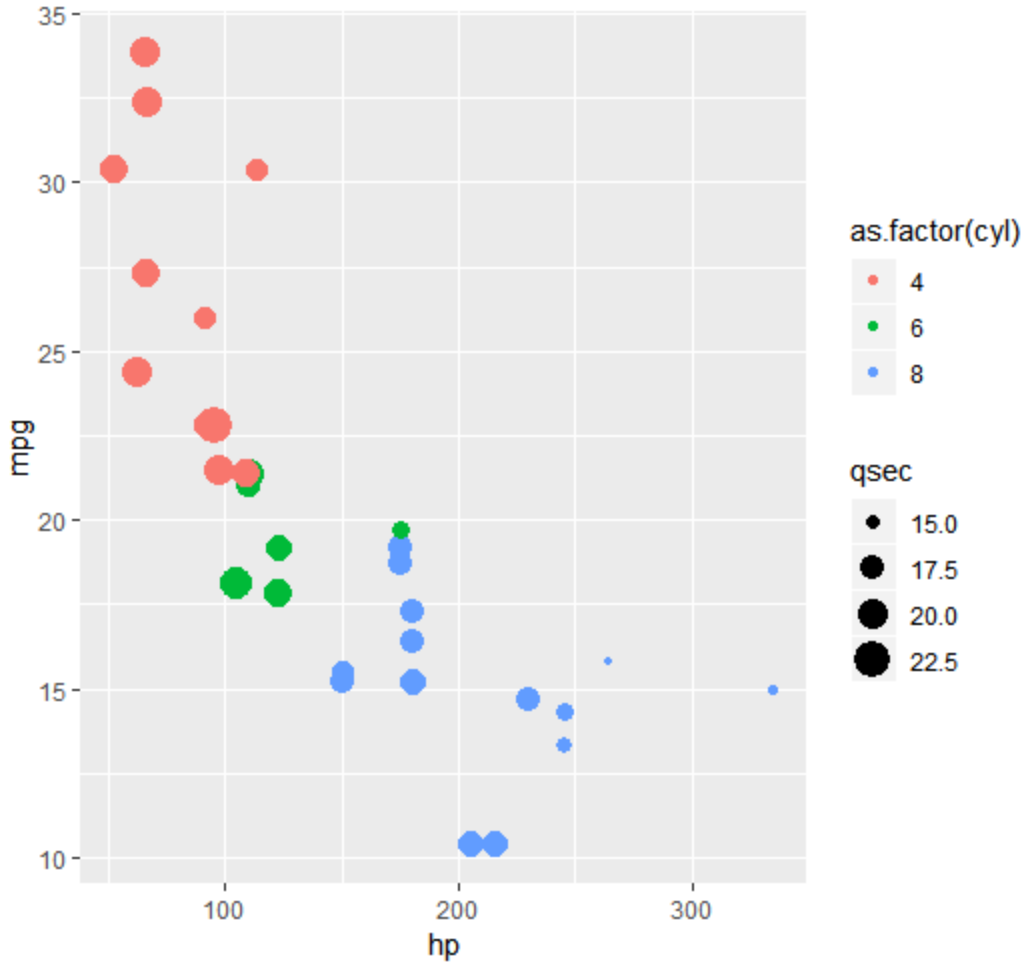
```
ggplot(data = mtcars, mapping = aes(x = hp, y = mpg)) + geom_point()
```



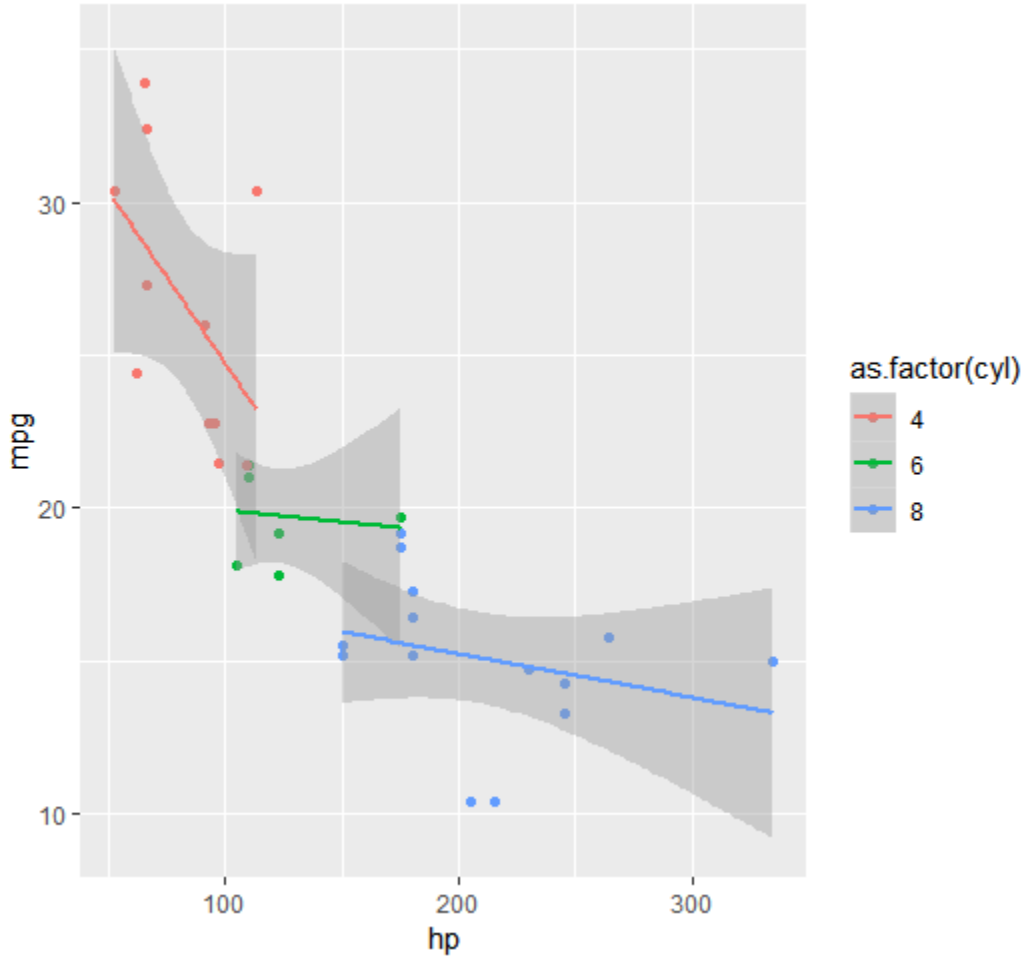

```
ggplot(data = mtcars, mapping = aes(x = hp, y = mpg)) + geom_line()
```



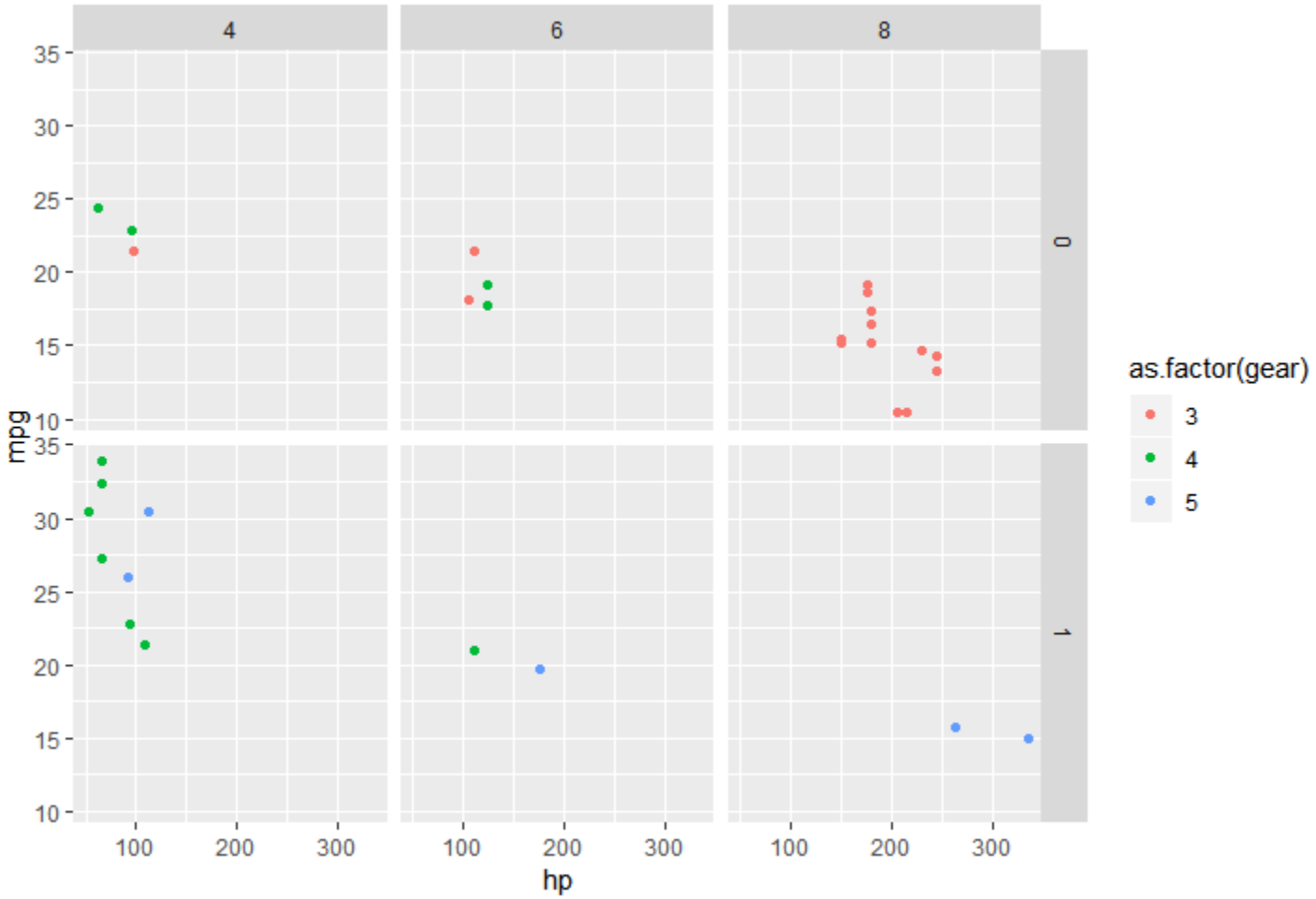
```
ggplot(data = mtcars, mapping = aes(x = hp, y = mpg,
size = qsec, color=as.factor(cyl))) + geom_point()
```




```
ggplot(data = mtcars, mapping = aes(x = hp, y = mpg,
color = as.factor(cyl))) + geom_point()+
stat_smooth(method = 'lm')
```



```
ggplot(data = mtcars, mapping = aes(x = hp, y = mpg, color = as.factor(gear))) +
geom_point()+ facet_grid(am ~ cyl)
```



Layer: geoms

A layer combines data, aesthetic mapping, a geom (geometric object), a stat (statistical transformation), and a position adjustment. Typically, you will create layers using a `geom_` function, overriding the default position and stat if needed.

`geom_abline()` `geom_hline()` `geom_vline()`

Reference lines: horizontal, vertical, and diagonal

`geom_bar()` `geom_col()` `stat_count()`

Bar charts

`geom_bin2d()` `stat_bin_2d()`

Heatmap of 2d bin counts

`geom_blank()`

Draw nothing

`geom_boxplot()` `stat_boxplot()`

A box and whiskers plot (in the style of Tukey)

`geom_contour()` `stat_contour()`

2d contours of a 3d surface

`geom_count()` `stat_sum()`

Count overlapping points

Layer: stats

A handful of layers are more easily specified with a `stat_` function, drawing attention to the statistical transformation rather than the visual appearance.

`stat_ecdf()`	Compute empirical cumulative distribution
`stat_ellipse()`	Compute normal confidence ellipses
`stat_function()`	Compute function for each x value
`stat_identity()`	Leave data as is
`stat_summary_2d()` `stat_summary_hex()`	Bin and summarise in 2d (rectangle & hexagons)
`stat_summary_bin()` `stat_summary()`	Summarise y values at unique/binned x
`stat_unique()`	Remove duplicates

Layer: position adjustment

All layers have a position adjustment that resolves overlapping geoms. Override the default by using the `position` argument to the `geom_` or `stat_` function.

	`position_dodge()` `position_dodge2()`	Dodge overlapping objects side-to-side
	`position_identity()`	Don't adjust position
	`position_jitter()`	Jitter points to avoid overplotting
	`position_jitterdodge()`	Simultaneously dodge and jitter
	`position_nudge()`	Nudge points a fixed distance
	`position_stack()` `position_fill()`	Stack overlapping objects on top of each another

Layer: annotations

Annotations are a special type of layer that don't inherit global settings from the plot. They are used to add fixed reference data to plots.

`geom_abline()` `geom_hline()` `geom_vline()`	Reference lines: horizontal, vertical, and diagonal
`annotate()`	Create an annotation layer
`annotation_custom()`	Annotation: Custom grob
`annotation_logticks()`	Annotation: log tick marks
`annotation_map()`	Annotation: a maps
`annotation_raster()`	Annotation: high-performance rectangular tiling
`borders()`	Create a layer of map borders

ggplot2

part of the tidyverse

3.1.0.9000

Aesthetics

The following help topics give a broad overview of some of the ways you can use each aesthetic.

`aes_colour_fill_alpha`	Colour related aesthetics: colour, fill and alpha
`aes_group_order`	Aesthetics: grouping
`aes_linetype_size_shape`	Differentiation related aesthetics: linetype, size, shape
`aes_position`	Position related aesthetics: x, y, xmin, xmax, ymin, ymax, xend, yend

Scales

Scales control the details of how data values are translated to visual properties. Override the default scales to tweak details like the axis labels or legend keys, or to use a completely different translation from data to aesthetic. `labs()` and `lims()` are convenient helpers for the most common adjustments to the labels and limits.

`labs()` `xlab()` `ylab()` `ggtitle()`	Modify axis, legend, and plot labels
`lims()` `xlim()` `ylim()`	Set scale limits
`expand_limits()`	Expand the plot limits, using data
`scale_alpha()` `scale_alpha_continuous()` `scale_alpha_discrete()` `scale_alpha_ordinal()`	Alpha transparency scales
`scale_colour_brewer()` `scale_fill_brewer()` `scale_colour_distiller()`	Sequential, diverging and qualitative colour scales from colorbrewer.org

Plotting exercise

Load the table «EtNAM.data.toy.txt»

Rookies:

- Create an histogram of plant height (PH_cm). Are there outliers? Which family and RIL?
- Create a xyplot of days to booting (DB), heading (DH), flowering (DF), and maturity (DF)
- Make a boxplot of TGW_gm by Et_family. Save it in a pdf named «TGW_family.pdf»

Pro:

- Use ggplot to plot the relation between male PVS and female PVS, sizing dots by grain yield. Fit a linear model, and divide the plot by Et_family

Rmarkdown

- A simplified version of markup language
- More attention on writing than on formatting
- Easily converted in HTML, pdf, docx, ppt, …

- Straightforward integration with R code
- Creation of documents with live R scripts
- Inclusion of script results in the document obtained
- Handy integration with Rstudio

Let’s see how Rmarkdown works using the script «Rmarkdown_intro.Rmd»