https://wustl.box.com/v/EtzelBaseRGraphicsSlides
https://wustl.box.com/v/EtzelBaseRGraphicsCode

base R graphics

Joset A. Etzel, PhD
jetzel@wustl.edu | mvpa.blogspot.com | @JosetAEtzel
Cognitive Control and Psychopathology Lab
Washington University in St. Louis

“base” R graphics? Aren’t there just “R graphics”?

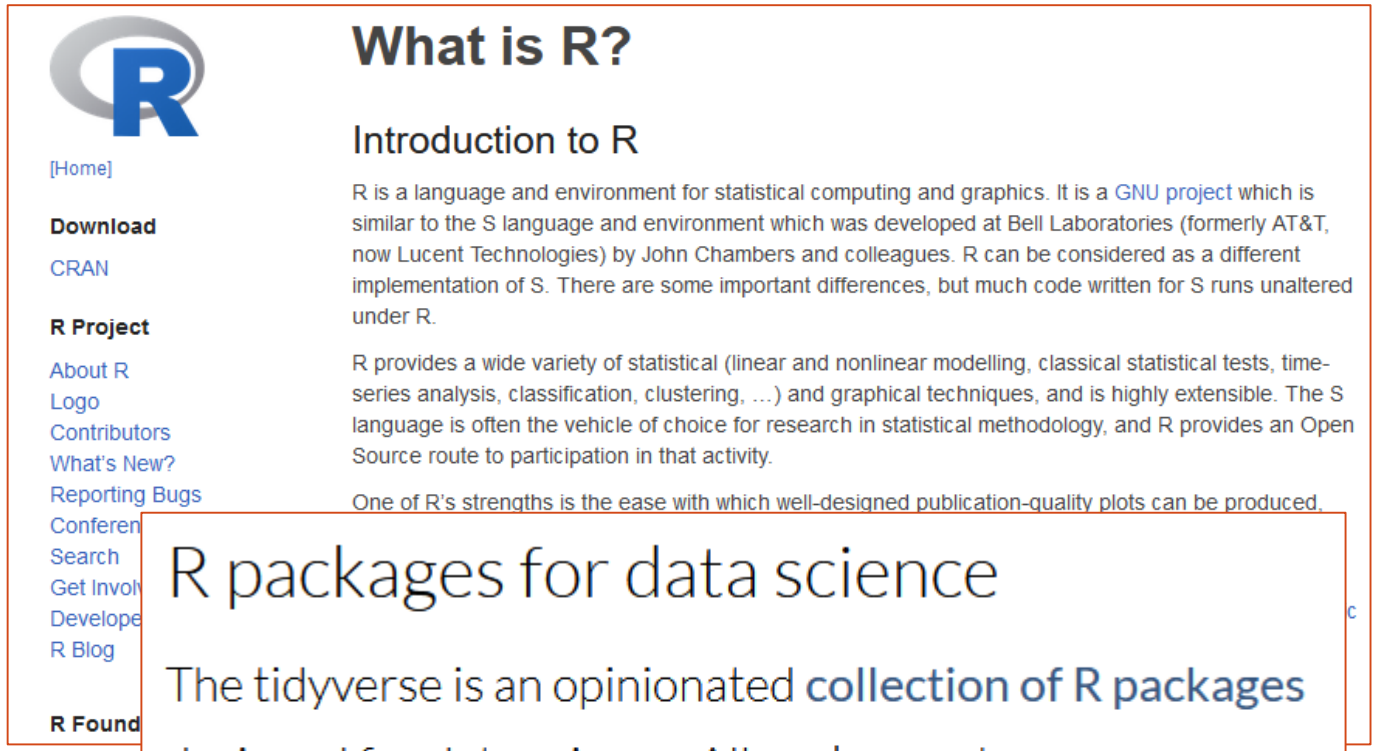
https://www.r-project.org/

No. R is open source (started in 1993, a variant of S-PLUS). Many people have contributed code over the years, so there are often multiple ways to do things (and variable syntax!).

Two (currently) common ways to make graphs with R are **ggplot2** (part of the tidyverse) and **base R graphics** (the first version; included in standard R installations).

I won’t be using any tidy packages today, partly to be “pure” base R, and partly because I very rarely use them.

But it is absolutely possible to mix the two: base R graphs from tidy-built data frames, base R layout and saving of ggplot2 graphics, etc.



R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

www.tidyverse.org

One minute of personal history

When I started there was only base R and lattice graphics.

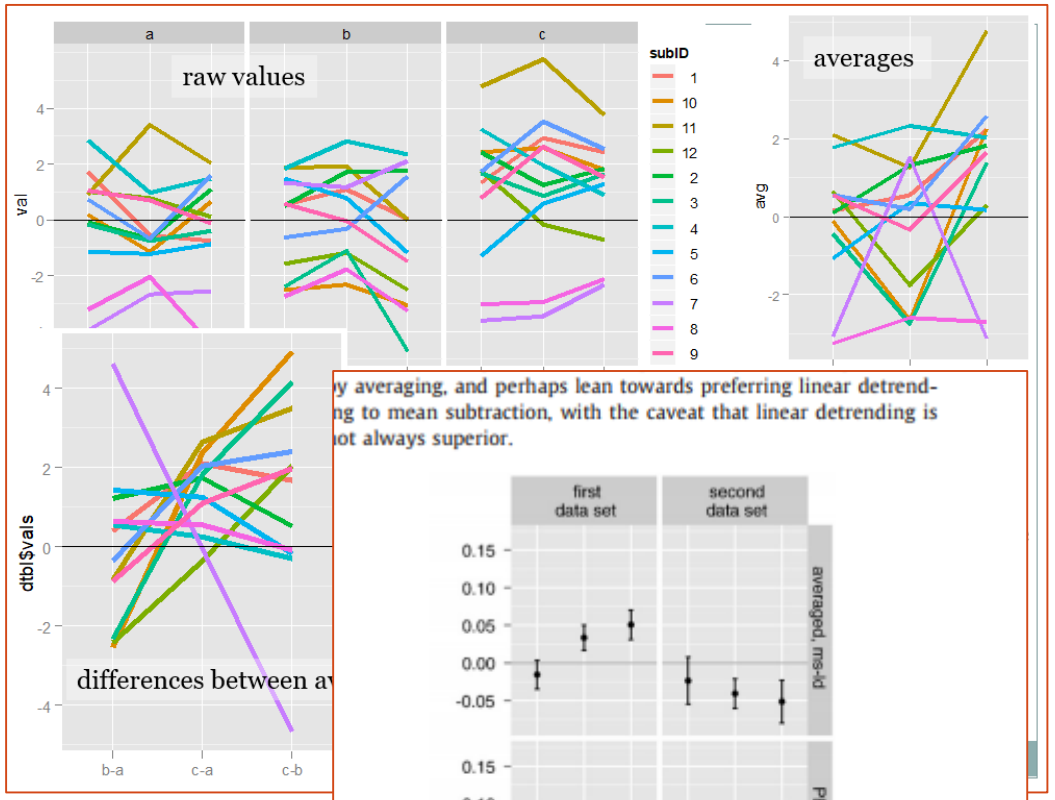
I started using ggplot2 and reshape/plyr functions when they came out (2007), using ggplot2 pretty exclusively for a few years (even for publications!).

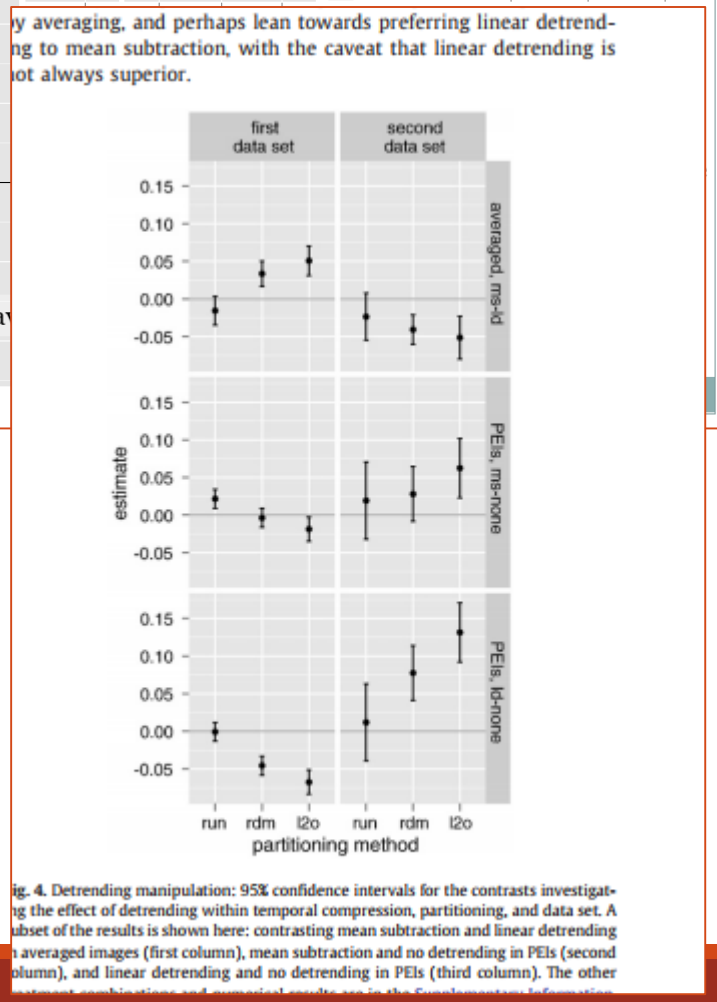
Around 2012 I started shifting back to base R graphics (and loops) and have stayed there since.

Why? I felt **too limited** by ggplot2; I was frustrated by not being able to make graphs look exactly like I wanted and needing to arrange large datasets into the required formats.

Perspective: I'm a staff scientist, and long-term code stability and clarity is very important: can a person very new to R/programming read this code? Will someone be able to understand and run it in 15 years?

Style warning: I tend to use many more semicolons than required.

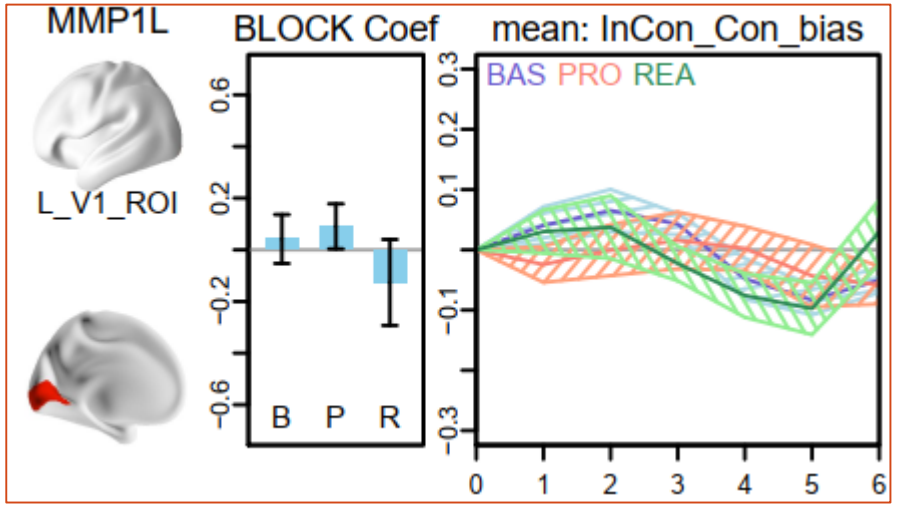


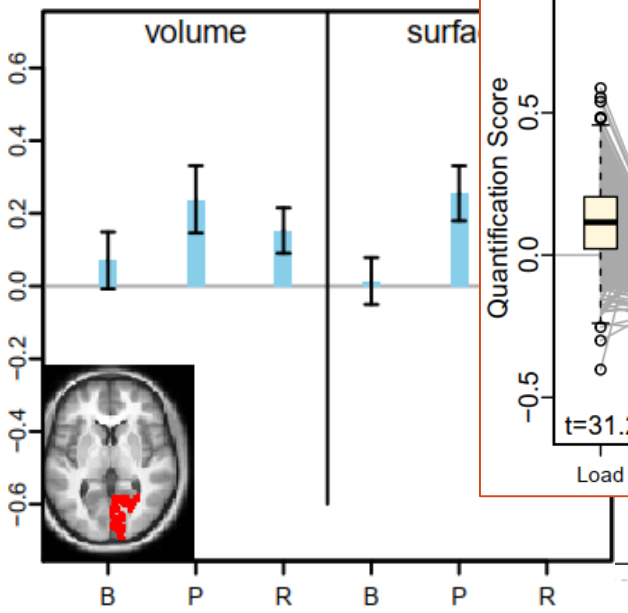


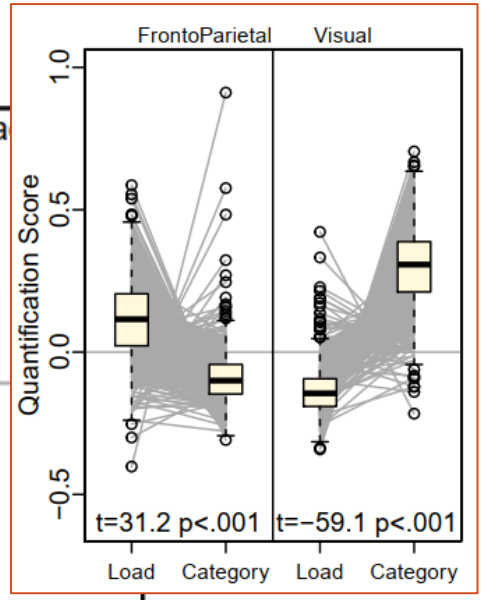
… preview of my base R graphing style.

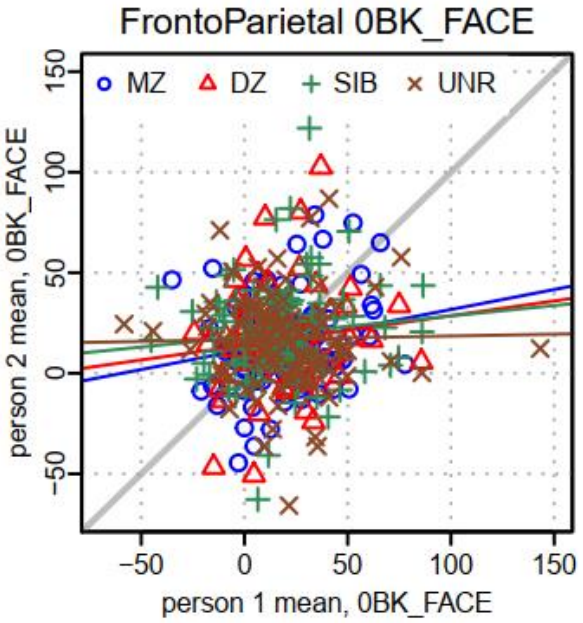
Some of my base R graphics

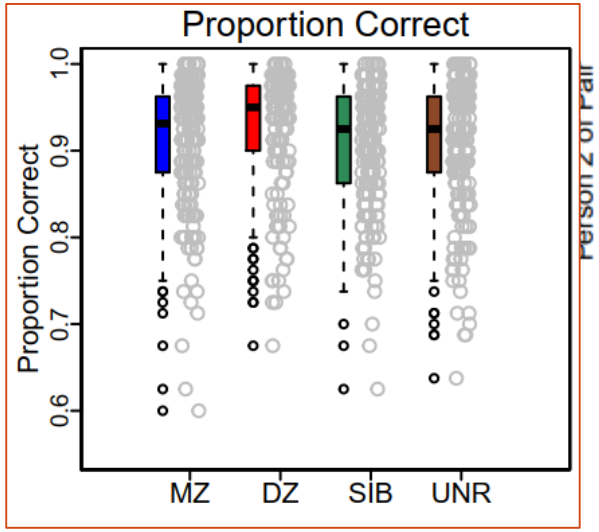
https://osf.io/p6msu/

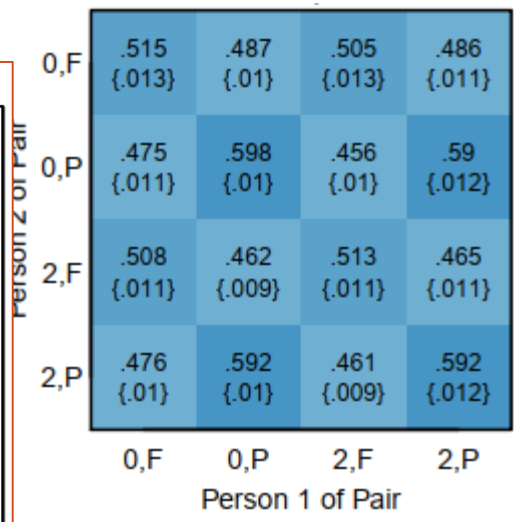


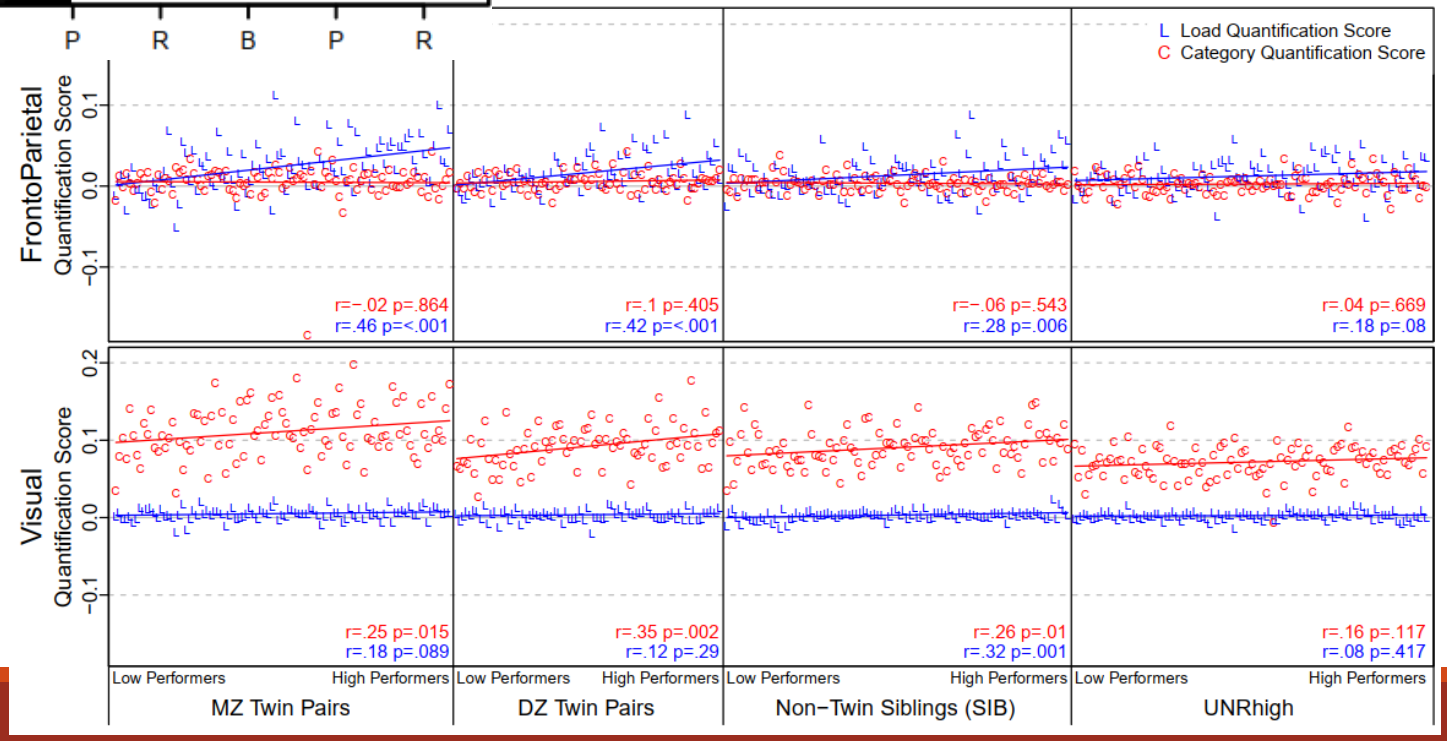






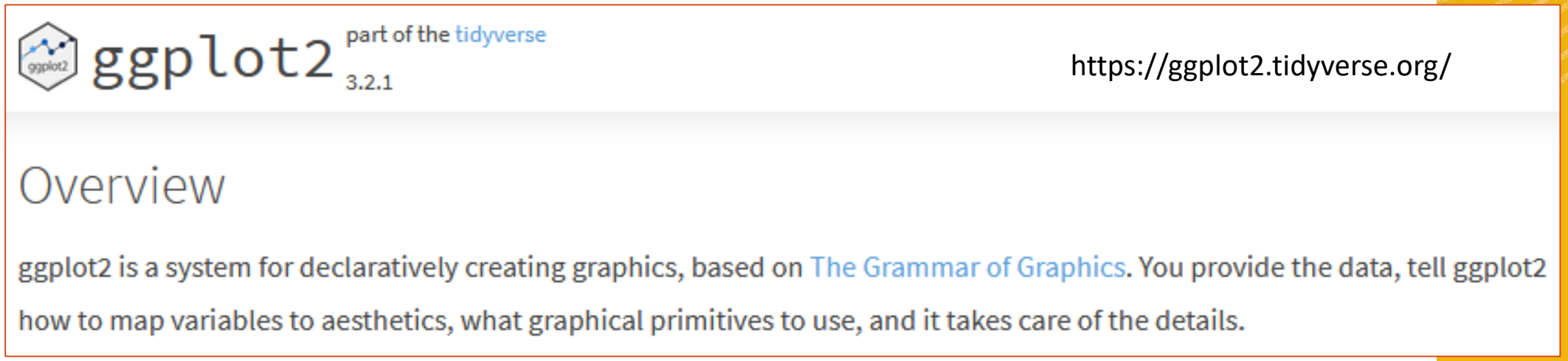




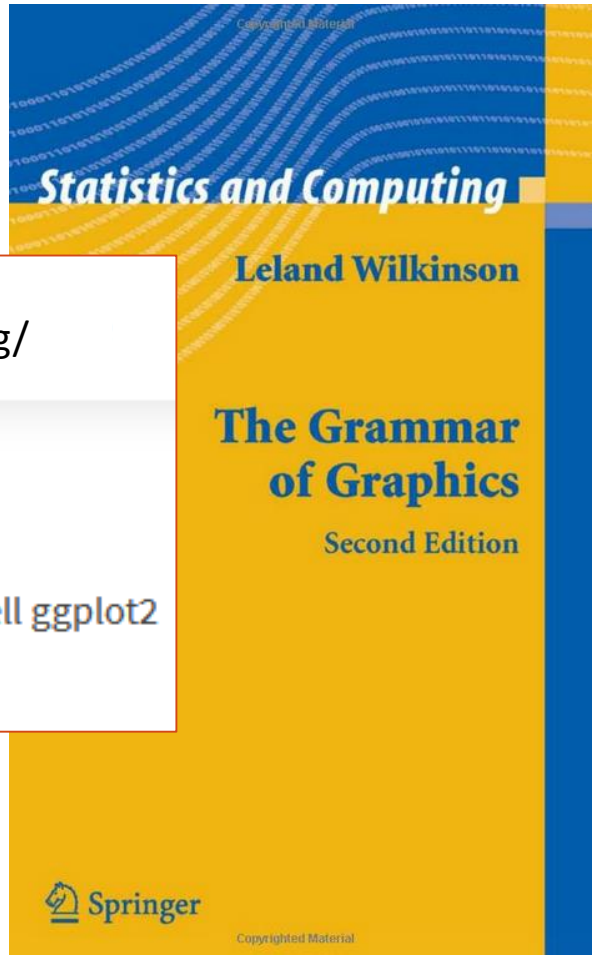


Aim of this talk: convey my recommended strategy and a starting toolkit

Base R graphics involves a lot of esoteric commands and options (usually decently documented; google is your friend). How to combine the commands is often unclear, particularly if you're used to the tidyverse or another design-heavy environment.



https://ggplot2.tidyverse.org/



With base R graphics it's **all details**: there's no need to collect the data into a logical structure first, because you build up the plot from independent pieces.

… total control, for better or worse.

First: Plan your plot

Think about your data and how you want to plot it (what do you want to know?); **literally sketch it out**.

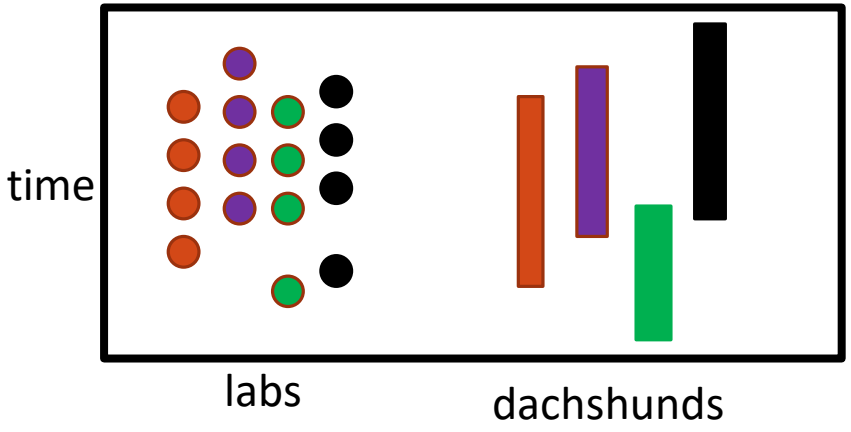
We need a toy dataset …

Measurements of **how long** (in minutes) 30 dogs of **two breeds** (Labrador retrievers and dachshunds) played with **toys of four different types** (rope, frisbee, bone, plush).

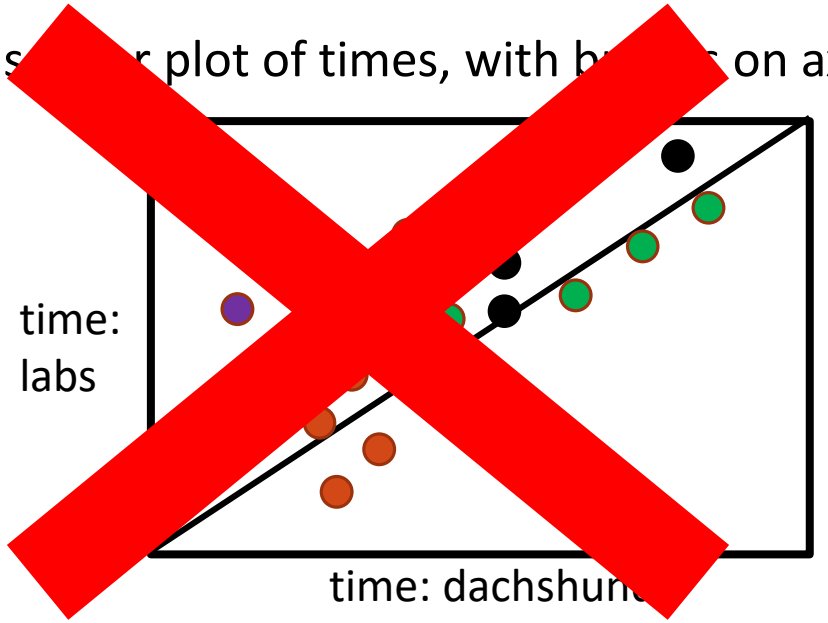
```
> head(data.tbl)
  dog.id breed.id  toy.id play.min
1      1 labrador frisbee 31.42174
2      2 labrador frisbee 25.24188
3      3 labrador frisbee 45.50001
4      4 labrador frisbee 55.93779
5      5 labrador frisbee 48.02602
6      6 labrador frisbee 22.75976
>
```

Question: Did the two types of dogs vary in how long they played with the different types of toys?

Idea: show distribution of times for the dogs, separated by breed and toy.



Idea: s r plot of times, with b s on axes.



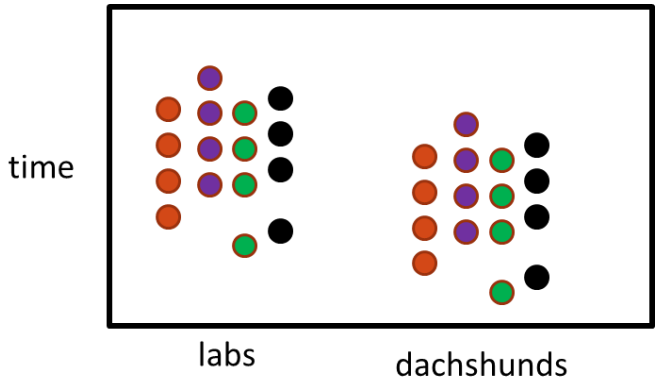
No! dachshund 1 != Labrador 1!

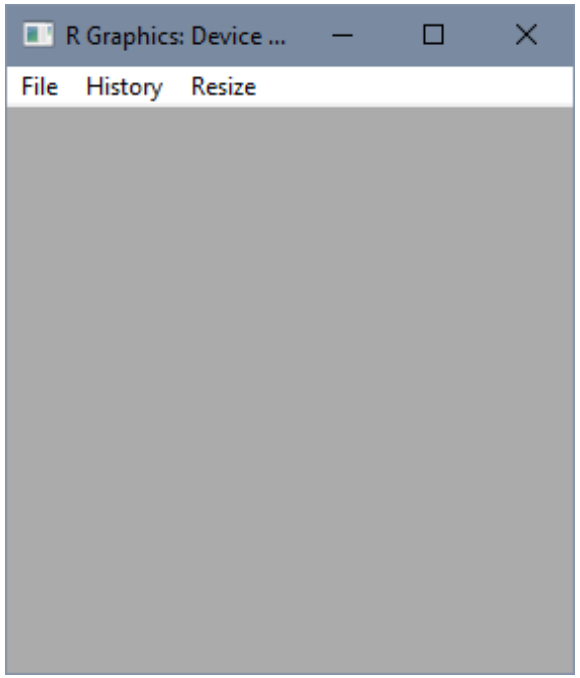
(just because you can make a plot doesn't mean it's sensible)

Start with a blank plot (of a defined size)

Always start by defining the plot window size, layout, and margins. All plot features (titles, points …) vary with window size, so specify it first.

… once you have a good combination, you can use it on screen in knitr/markdown, printing to file, etc., and it will look exactly the same.






```
windows(width=3, height=3);  # specify plot window size, try quartz() on macOS
# layout(matrix()) commands here if want more than one plot in the window
par(mar=c(2, 2.5, 1.5, 0.75), mgp=c(1.1, 0.2, 0), tcl=-0.2);    # specify margins
```

Next, calculate/specify the x- and y-axis limits and other general parameters.

… base R graphics calculates these for you if you use the all-in-one command plotting functions (plot(); boxplot(); etc.), but not for plots made piece-by-piece like in this demo.

We want spots for each breed on the x-axis:

```
x.lim <- c(0.7, (length(breed.ids)+0.3));   # x-axis limits; one for each breed plus padding
```

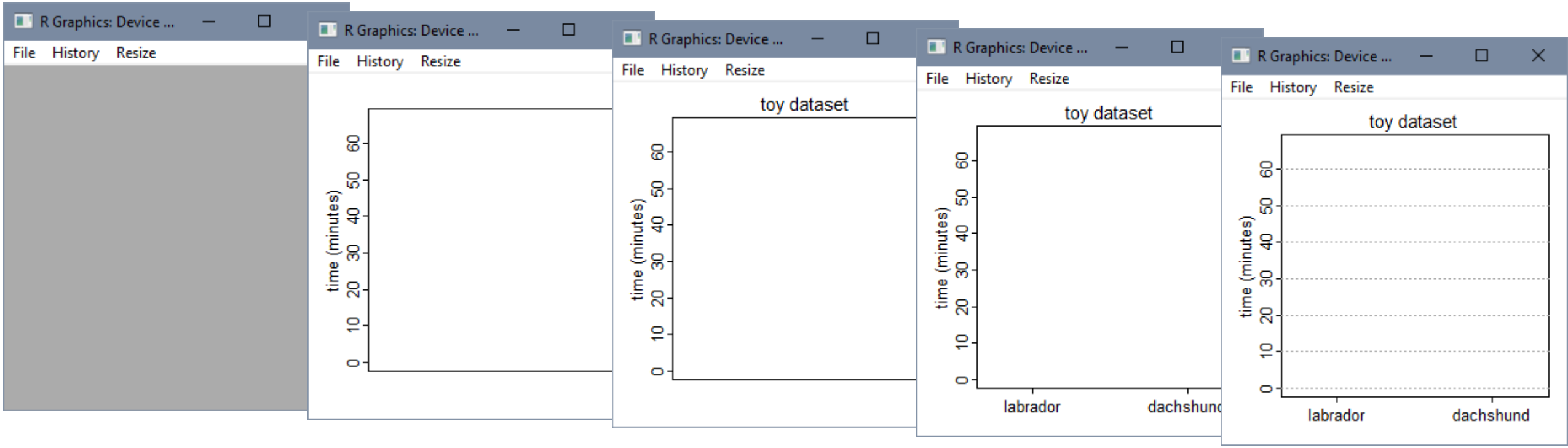
And room to show all the data on the y-axis:

```
y.lim <- c(min(data.tbl$play.min, na.rm=TRUE), max(data.tbl$play.min, na.rm=TRUE)+7);  # y-axis limits. +7 at top for legend

toy.cols <- c('firebrick', 'darkmagenta', 'forestgreen', 'grey40');   # color to plot each of the toys (toy.ids order)
shifts <- c(-0.15, -0.05, 0.05, 0.15);     # x offsets (where to put the points, relative to 1 and 2 (since x.lim <- c(0.7, 2.3);)
```

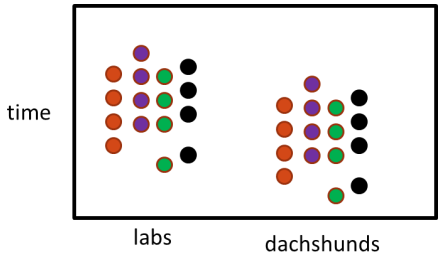
Start with a blank plot (of a defined size)

Add a blank plot to the window, with just the basics (titles, grid lines, axes, etc.) – no data (yet).

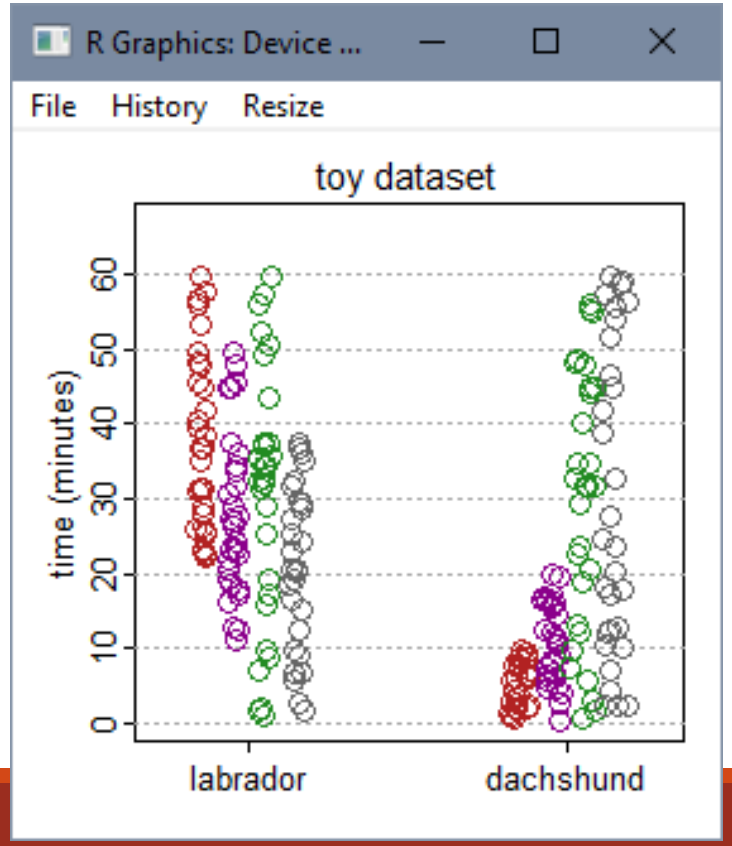
```
plot(x=0, y=0, xlim=x.lim, ylim=y.lim, xaxt='n', col='white', xlab="", ylab="time (minutes)", main="", cex.lab=0.8, cex.axis=0.8);
# blank plot with calculated axis limits, only y-axis shown

mtext(side=3, text="toy dataset", line=0.15, cex=0.9);    # add plot title (on top)

axis(side=1, at=1:length(breed.ids), labels=breed.ids, cex.axis=0.8);   # add the x-axis breed labels

grid(nx=NA, ny=NULL, col='darkgrey');    # add horizontal grid lines
```



Add the data: points, lines, bars, images, boxplots, ….




```
for (tid in 1:length(toy.ids)) {
   for (bid in 1:length(breed.ids)) {     # tid <- 3;  bid <- 1;
      vals <- data.tbl$play.min[which(data.tbl$breed.id == breed.ids[bid] & data.tbl$toy.id== toy.ids[tid])];  # get the data.
      if (length(vals) != num.dogs) { stop("length(vals) != num.dogs"); }     # bit of error-checking (more in real dataset)
      points(x=jitter(rep(bid+shifts[tid], length(vals))), y=vals, col=toy.cols[tid], cex=1.2);   # points w/jitter (for less overplotting)
   }
}
```



A few strategy/logic notes:

I prefer loops for this type of code; find the clarity worth the verboseness.

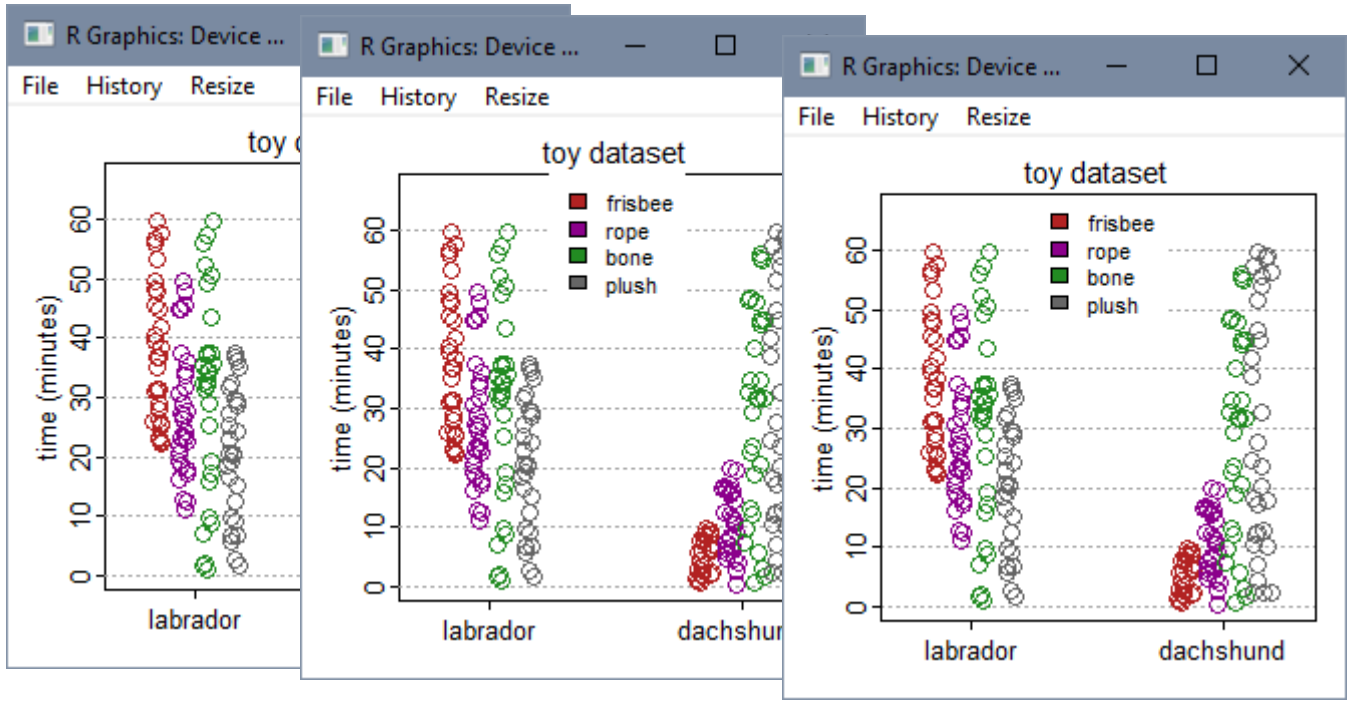
The **shifts** variable defines the spacing of the columns of points at each breed.

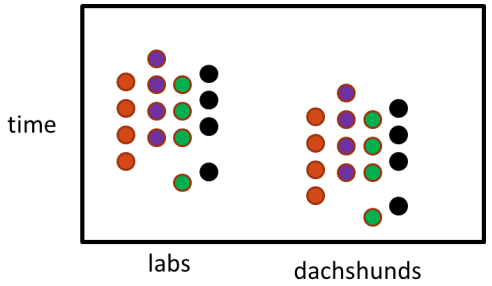
```
shifts <- c(-0.15, -0.05, 0.05, 0.15);     # x offsets
```

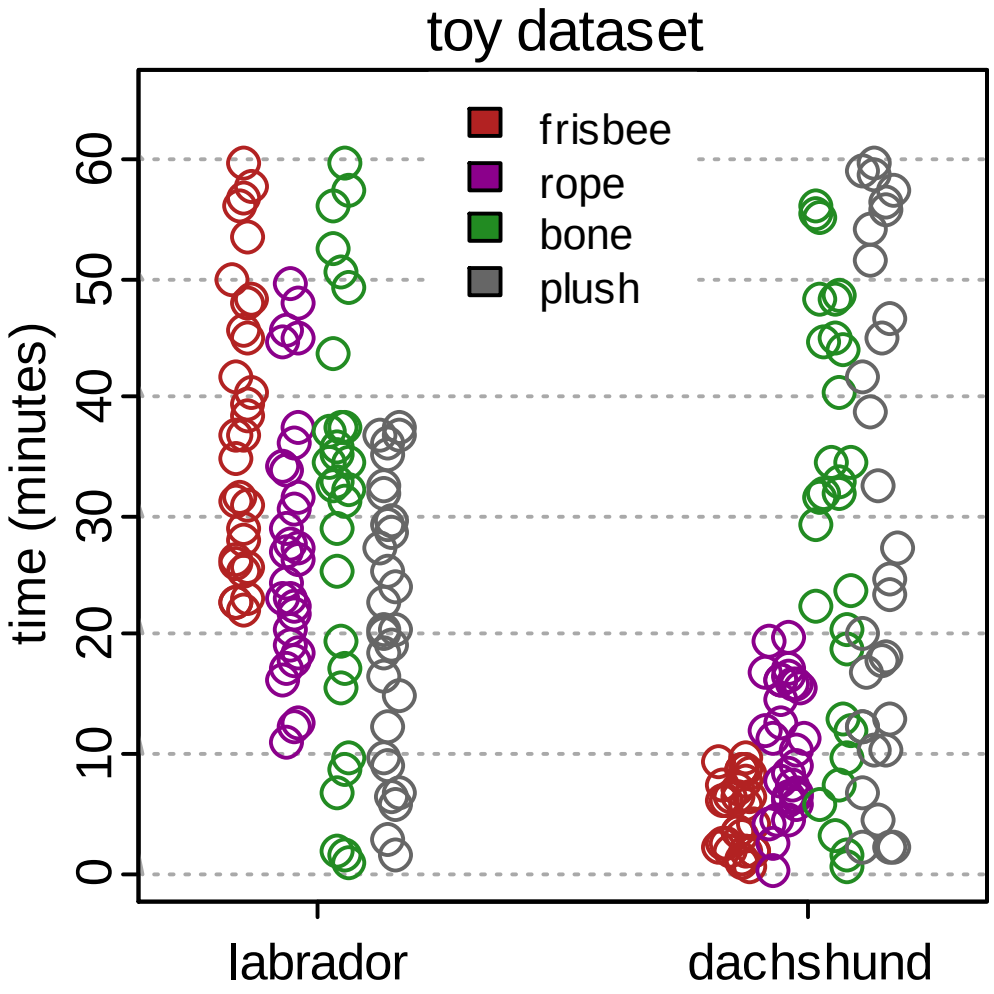
This is the only part of the plotting code where having the data is absolutely required. A data.frame containing all the data is not necessary; each toy and breed could be stored in separate files or calculated in this loop, for example.

Finish the plot with a legend and a box()

```
legend(x='top', legend=toy.ids, fill=toy.cols, horiz=FALSE, cex=0.7, bg='white', box.col='white');
box();  # redraw the box around the outside
```







I put the step-by-step graphs in as screen captures; vector formats nicer.

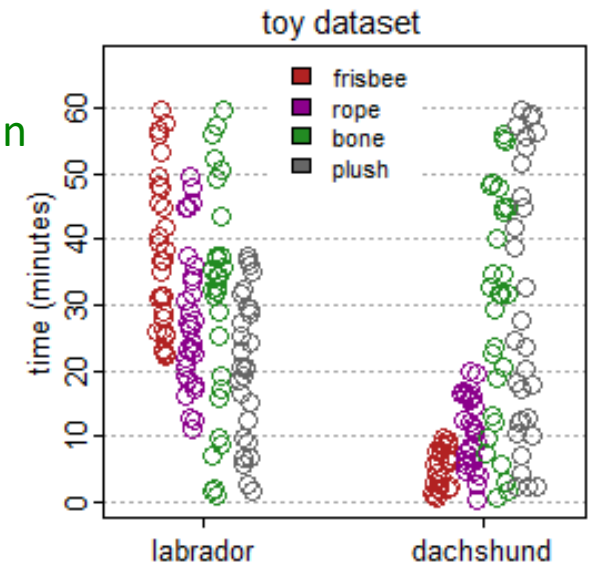
Can also print to disk: replace windows() with tiff(), jpeg(), etc.

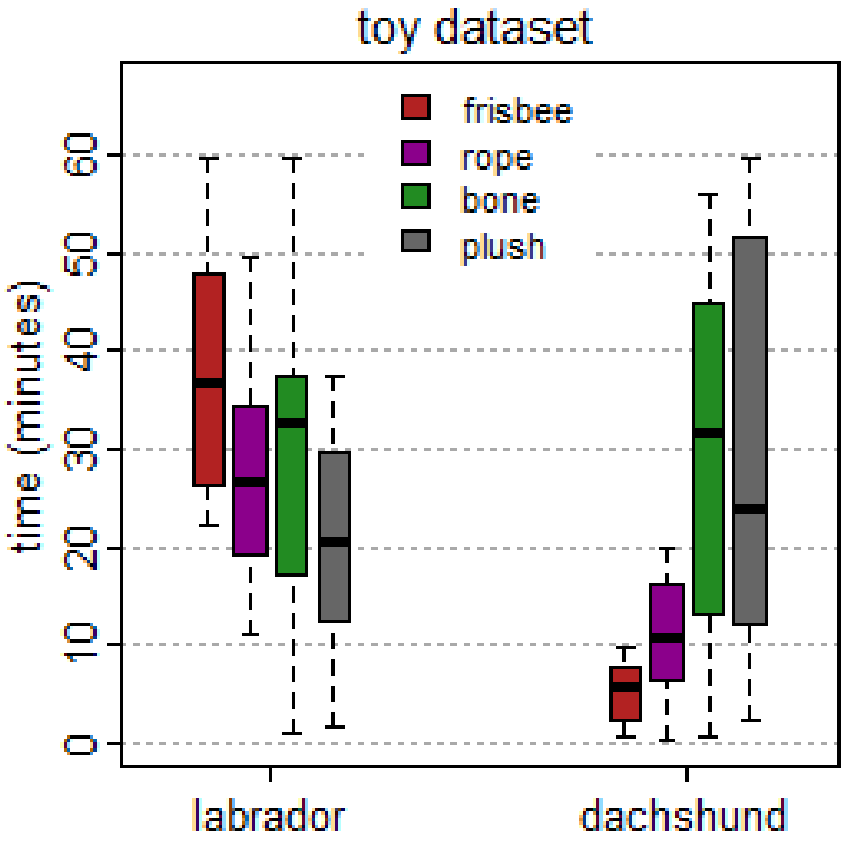
```
# windows(8, 3);  # original plot window command
tiff("d:/temp/figure.tif", width=8, height=3, units="in", res=600);
##### plotting code #####  (ggplot2, base R, etc.)
dev.off();     # release file
```

Switching things up: boxplots instead of points, but same labels, colors, etc.

How much code needs to change? Just one line: **boxplot()** instead of **points()**.

```
points(x=jitter(rep(bid+shifts[tid], length(vals))), y=vals, col=toy.cols[tid], cex=1.2);   # points version
boxplot(vals, at=bid+shifts[tid], col=toy.cols[tid], add=TRUE, xaxt='n', yaxt='n', bty='n',
        boxwex=0.15,  cex=0.7);      # boxplot version
```





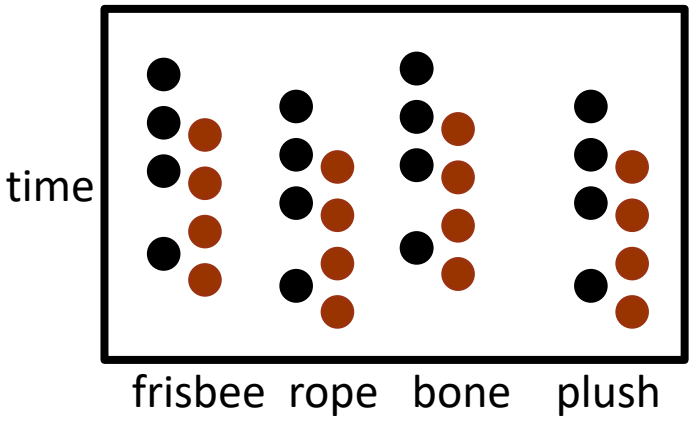
(The boxplot command is long; it draws new plots by default, so we need to specify many options to add properly.)

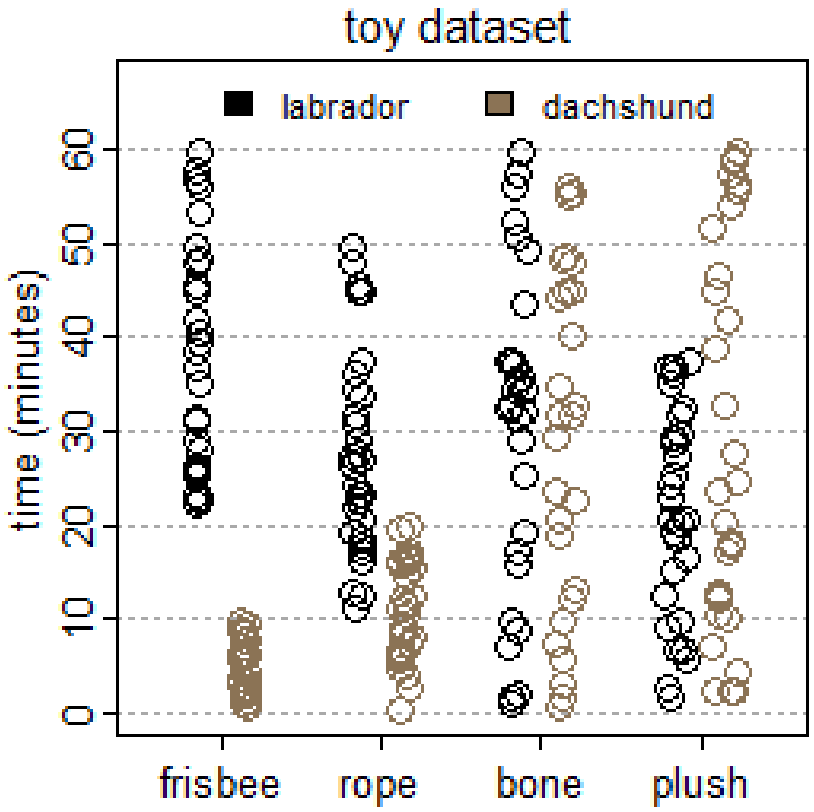
Switching things up: dots again, but toys along the x-axis

How much code needs to change?

x-axis range and labels; color and spacing of columns; legend. Code structure and logic is unchanged.

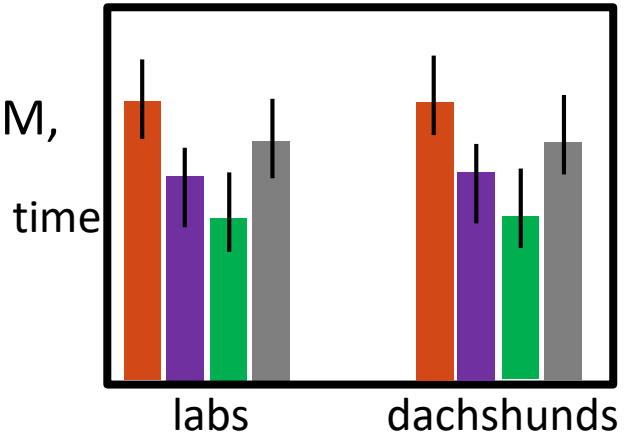
```
shifts <- c(-0.15, -0.05, 0.05, 0.15);        # x offsets for the four toys
toy.cols <- c('firebrick', 'darkmagenta', 'forestgreen', 'grey40');
x.lim <- c(0.7, (length(breed.ids)+0.3));   # x-axis limits

b.shifts <- c(-0.15, 0.15);  # x offsets for the two breeds
b.cols <- c("black", "burlywood4");
x.lim <- c(0.5, (length(toy.ids)+0.5));   # x-axis limits

axis(side=1, at=1:length(breed.ids), labels=breed.ids, cex.axis=0.8);
axis(side=1, at=1:length(toy.ids), labels=toy.ids, cex.axis=0.8);

points(x=jitter(rep(bid+shifts[tid], length(vals))), y=vals, col=toy.cols[tid]);
points(x=jitter(rep(tid+b.shifts[bid], length(vals))), y=vals, col=b.cols[bid]);
```



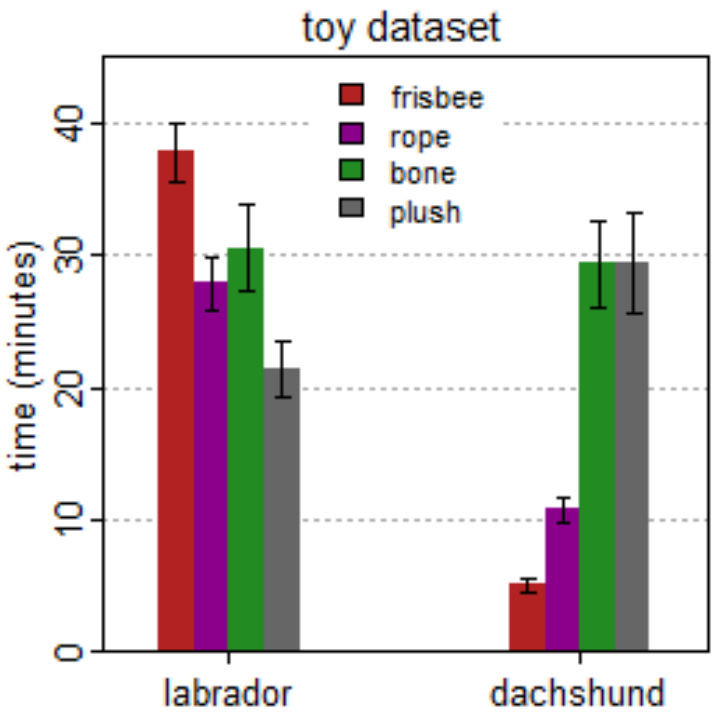


Switching things up: bars with means and SEMs

Still replace **points()**, but need to add multiple lines of code: calculate the mean and SEM, then draw the bars (as rectangles) and add the SEM lines (**full manual method**).




```
mean.val <- mean(vals, na.rm=TRUE);    # calculate the mean
rect(xleft=bid+shifts[tid]-bar.half, xright=bid+shifts[tid]+bar.half, ybottom=y.lim[1],
      ytop=mean.val, border=NA, col=toy.cols[tid]);   # rectangle for bar

sem.val <- sd(vals, na.rm=TRUE)/sqrt(length(vals));   # calculate the SEM
arrows(x0=bid+shifts[tid], x1=bid+shifts[tid],
            y0=mean.val, y1=mean.val+sem.val, angle=90, length=0.025);  # up bar
arrows(x0=bid+shifts[tid], x1=bid+shifts[tid],
            y0=mean.val, y1=mean.val-sem.val, angle=90, length=0.025);  # down bar
```



Necessary to **draw rectangles** to make bar charts?!??

Base R graphics includes a lot of plotting functions

Googling “base r graphics” will bring up zillions of hits, which point you to the names of the built-in plotting commands (along with other useful things).

base r graphics

All | Images | Shopping | News | Videos | More

About 684,000,000 results (0.37 seconds)

R Base Graphs - Easy Guides - Wiki - STHDA
www.sthda.com › english › wiki › r-base-graphs
This chapter contains articles describing how to visualize data using **R ba**
Creating **graphs**; Saving **graphs**; File formats for exporting plots.
Graphical parameters · Scatter Plot Matrices - R Base ... · Generic plot typ

R Base Graphics - RPubs
rpubs.com › SusanEJohnston
Aug 30, 2013 - **R Base Graphics**: An Idiot's Guide. One of the most powe
it's ability to produce a wide range of **graphics** to quickly and ...

Base R Graphics Cheat Sheet - David Gerard
https://dcgerard.github.io › stat234 › base_r_cheatsheet
Aug 8, 2017 - Abstract: I reproduce some of the plots from Rstudio's ggplo
Base R graphics. I didn't try to pretty up these plots, but you ...

The R Graphics Package - R
https://stat.ethz.ch › R-manual › R-devel › library › graphics › html
Documentation for package '**graphics**' version 4.0.0. DESCRIPTION file

- Creati
- Generi
- Scatte
- Scatte
- Box Pl
- Strip Charts: 1-D scatter Plots
- Bar Plots
- Line Plots
- Pie
- Hist
- QQ
- Dot
- Plot
- R b

Here, we'll describe how to create **bar plots** in R. The function **barplot**() can be used to create a **bar plot** with vertical or horizontal bars.

The second site’s example tries to make a barplot with error bars … but has great difficulty, eventually using a gplots library function.

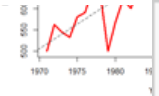
R Base Graphics: An Idiot's Guide

One of the most powerful functions of R is it's ability to produce a wide range of graphics to quickly and easily visu

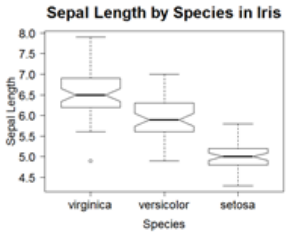
Making the leap from chiefly graphical programmes, such as Excel and Sigmaplot. may seem tricky. However, with

Last year, I presented an informal course on the basics of R Graphics University of Turku. In this blog post, I am pr
in R, including:

1. Basic Histogram



2. Line Graph w



4. Boxplot with reordered/ formatted axes

5. Barplot with error bars using summary data

Ugh. I warn you - this will not be pretty.Let's create a new data frame with information on three populations of

```
barplot(dragons$TalonLength, names = dragons$Population,
        ylim=c(0,70),xlim=c(0,4),yaxs='i', xaxs='i',
        main="Dragon Talon Length in the UK",
        ylab="Mean Talon Length",
        xlab="Country")
par(new=T)
plotCI (dragons$TalonLength,
        uiw = dragons$SE, liw = dragons$SE,
        gap=0,sfrac=0.01,pch="",
        ylim=c(0,70),
        xlim=c(0.4,3.7),
        yaxs='i', xaxs='i',axes=F,ylab="",xlab="")
```

Using barplot()

The examples and help let us know that barplot() can take a data.table with precalculated means.

Looping version!

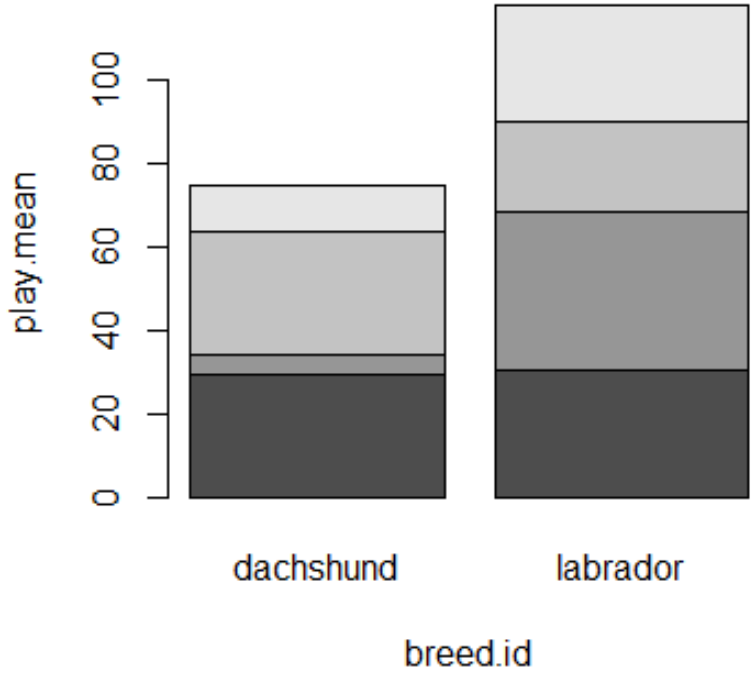
```
mean.tbl <- data.frame(array(NA, c(length(toy.ids)*length(breed.ids),4)));
colnames(mean.tbl) <- c("breed.id", "toy.id", "play.mean", "play.sem");
ctr <- 1;  # row counter
for (tid in 1:length(toy.ids)) {
   for (bid in 1:length(breed.ids)) {   # tid <- 1;  bid <- 2;
     vals <- data.tbl$play.min[which(data.tbl$breed.id == breed.ids[bid] & data.tbl$toy.id== toy.ids[tid])];
     if (length(vals) != num.dogs) { stop("length(vals) != num.dogs"); }  # bit of error-checking

     mean.tbl$breed.id[ctr] <- breed.ids[bid];
     mean.tbl$toy.id[ctr] <- toy.ids[tid];
     mean.tbl$play.mean[ctr] <- mean(vals, na.rm=TRUE);    # calculate and store the mean
     mean.tbl$play.sem[ctr] <- sd(vals, na.rm=TRUE)/sqrt(length(vals));   # calculate and store the SEM
     ctr <- ctr + 1;
   }
}
mean.tbl    # print the new data.frame
```

```
mean.tbl
 breed.id  toy.id play.mean  play.sem
 labrador frisbee 37.790252 2.1748316
dachshund frisbee  5.004731 0.5306263
 labrador    rope 27.838501 1.9948391
dachshund    rope 10.735797 0.9908724
 labrador    bone 30.535096 3.1654374
dachshund    bone 29.350348 3.1920232
 labrador   plush 21.335055 2.0393984
dachshund   plush 29.397078 3.7526007
```

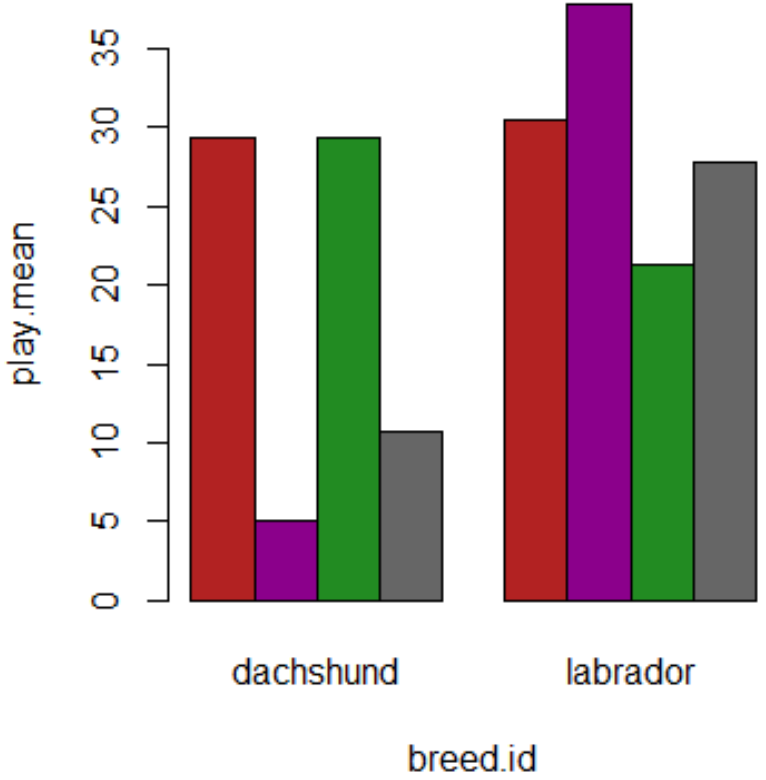
Using barplot() … continued

The minimum code is very short … but with an ugly result:
barplot(play.mean~toy.id+breed.id, data=mean.tbl);

```
mean.tbl
 breed.id  toy.id play.mean  play.sem
 labrador frisbee 37.790252 2.1748316
dachshund frisbee  5.004731 0.5306263
 labrador    rope 27.838501 1.9948391
```

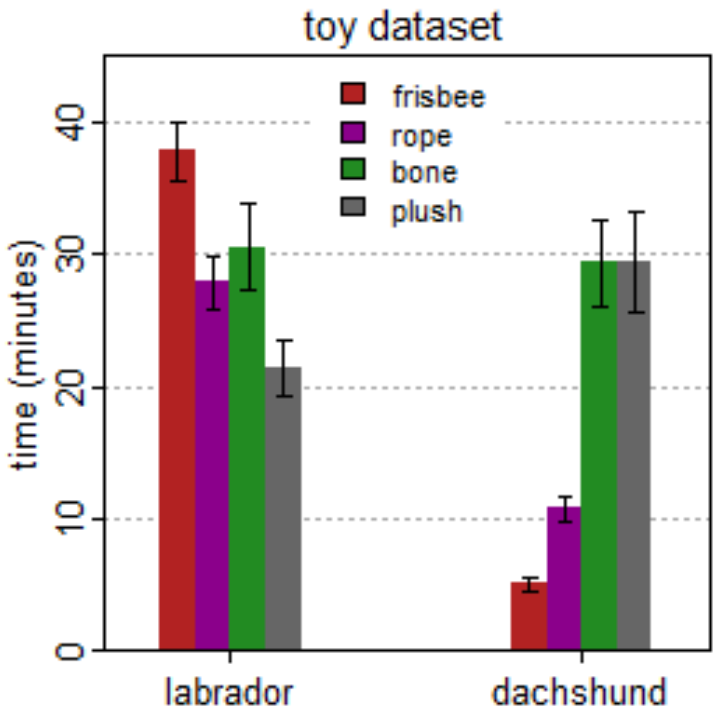


A few additions fix the stacking and colors:
barplot(play.mean~toy.id+breed.id, data=mean.tbl,
beside=TRUE, col=toy.cols);



!???!!?

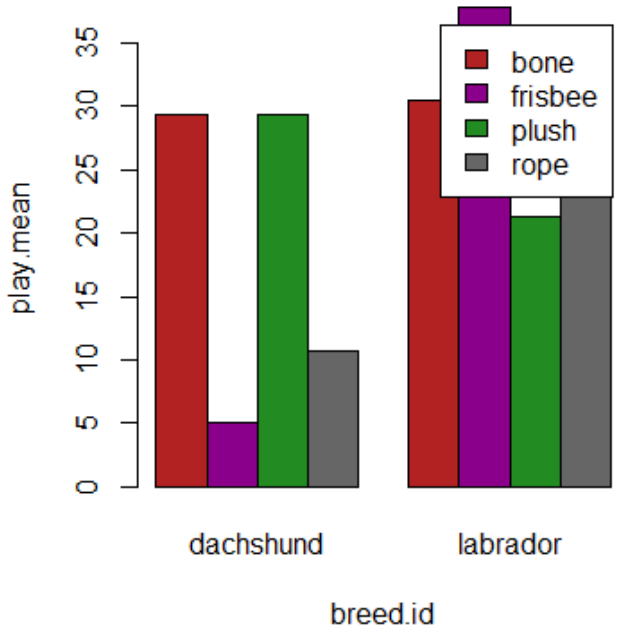
why so different?

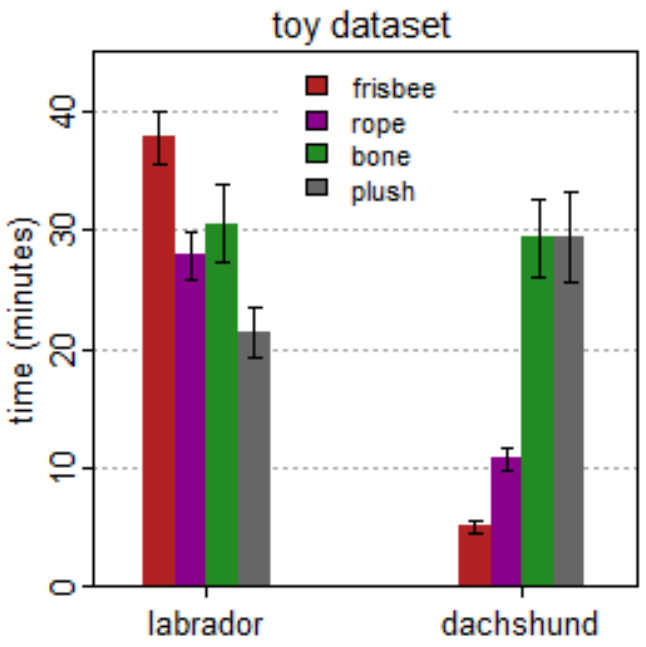


Using barplot() … continued

We can make the ordering match: change the breed.id and toy.id columns in mean.tbl to factors and specify the level order.

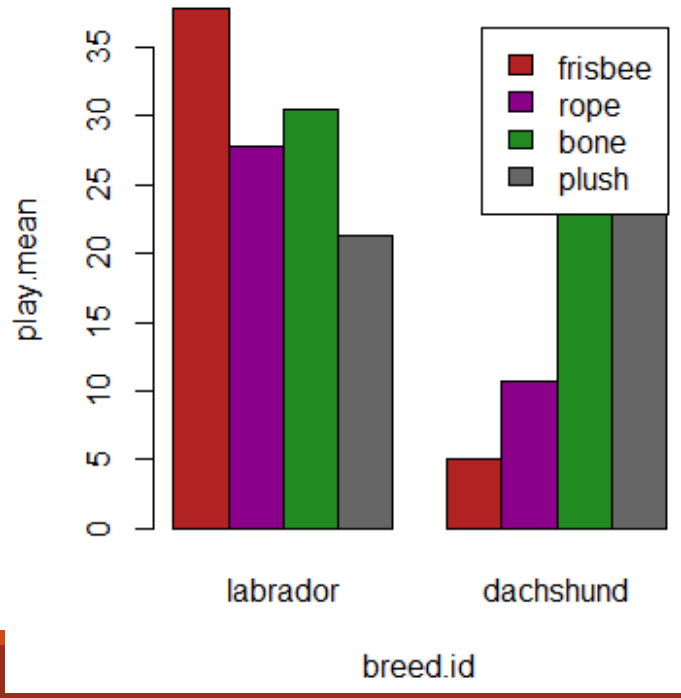
```
mean.tbl$breed.id <- factor(mean.tbl$breed.id,
      levels=c("labrador", "dachshund“, ordered=TRUE);
mean.tbl$toy.id <- factor(mean.tbl$toy.id,
      levels=toy.ids, ordered=TRUE);

barplot(play.mean~toy.id+breed.id, data=mean.tbl,
      beside=TRUE, col=toy.cols, legend.text=TRUE);
```

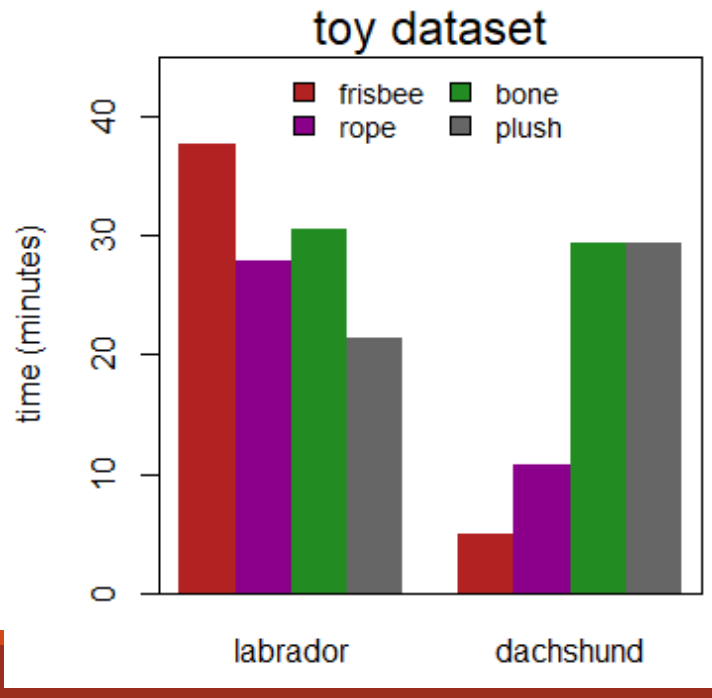




Still need to fix the axis labels and titles, make room for and move the legend ….

```
barplot(play.mean~toy.id+breed.id, data=mean.tbl,
      beside=TRUE, col=toy.cols, border=NA, ylim=c(0, 45),
      xlab="", ylab="time (minutes)", main="",
      legend.text=TRUE, args.legend=list(x='top', ncol=2,
      cex=0.9, bty='n'));
mtext(side=3, text="toy dataset", line=0.2, cex=1.5);
box();
```

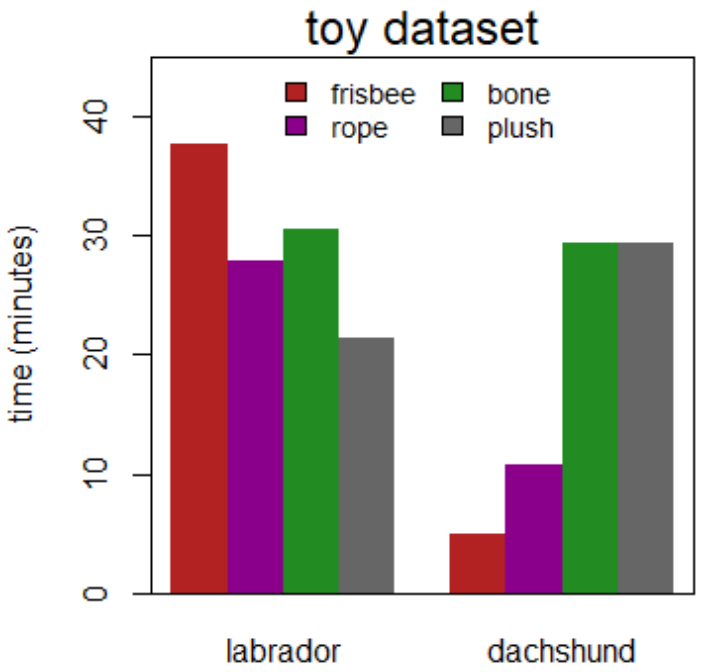


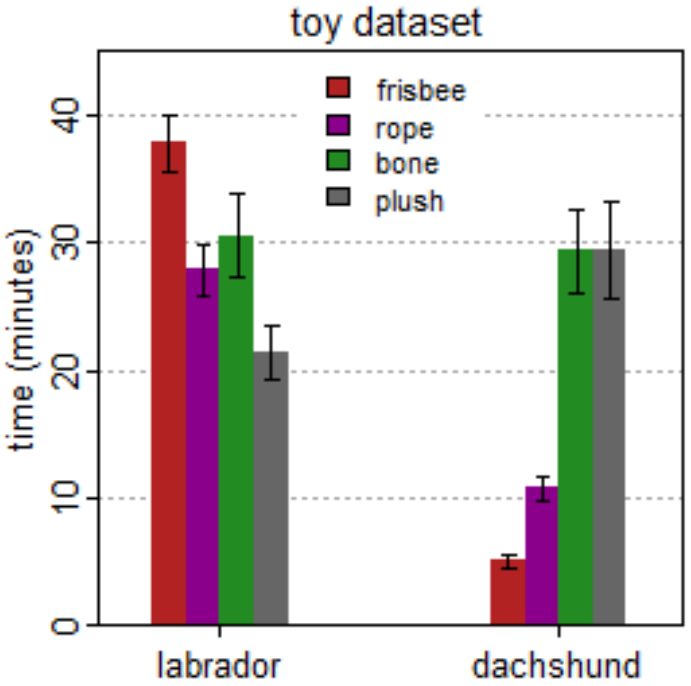


Using barplot() … continued

Better; could improve still further … but we're moving towards the other code: separate commands, not just barplot().

And some plotting options (bar ordering, which color goes with which toy, etc.) are not set directly when plotting, but by changing data.frame properties, which can be confusing, and problematic in complex cases.

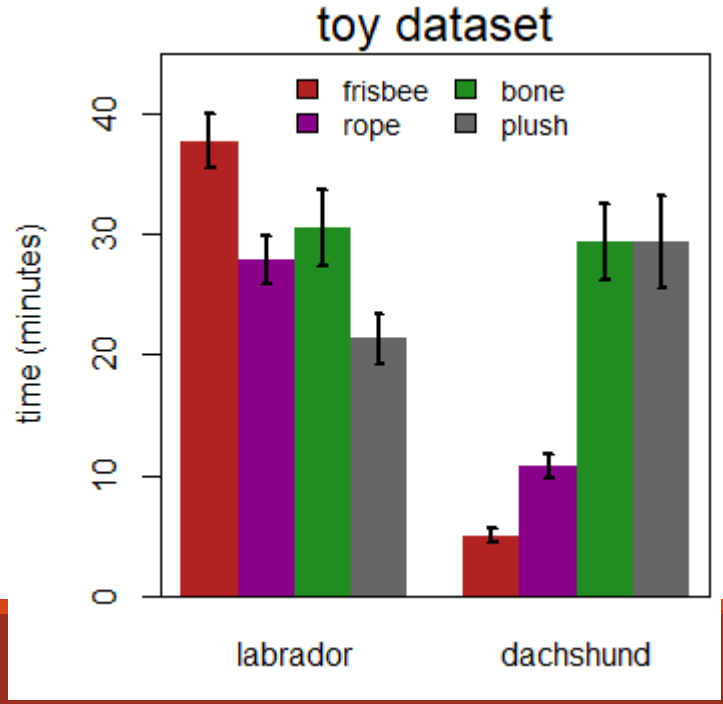




The SEM bars? The grid lines?

For SEM we can use a separate arrows() call again … where are the bar centers?

```
str(boxplot(CODE));  # num [1:4, 1:2] 1.5 2.5 3.5 4.5 6.5 7.5 8.5 9.5

centers <- c(1.5,6.5, 2.5,7.5, 3.5,8.5, 4.5,9.5);  # hard-coded in mean.tbl row order

arrows(x0=centers, x1=centers, y0=mean.tbl$play.mean, y1=mean.tbl$play.mean+
mean.tbl$play.sem, angle=90, length=0.025, lwd=2);  # up error bars

arrows(x0=centers, x1=centers, y0=mean.tbl$play.mean, y1=mean.tbl$play.mean-
mean.tbl$play.sem, angle=90, length=0.025, lwd=2);  # down error bars
```



Using barplot() … final

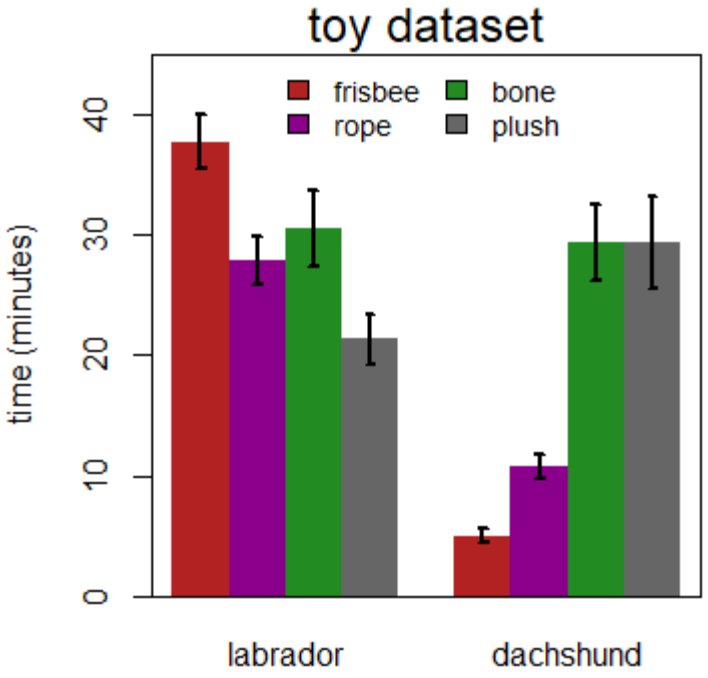
Gridlines?

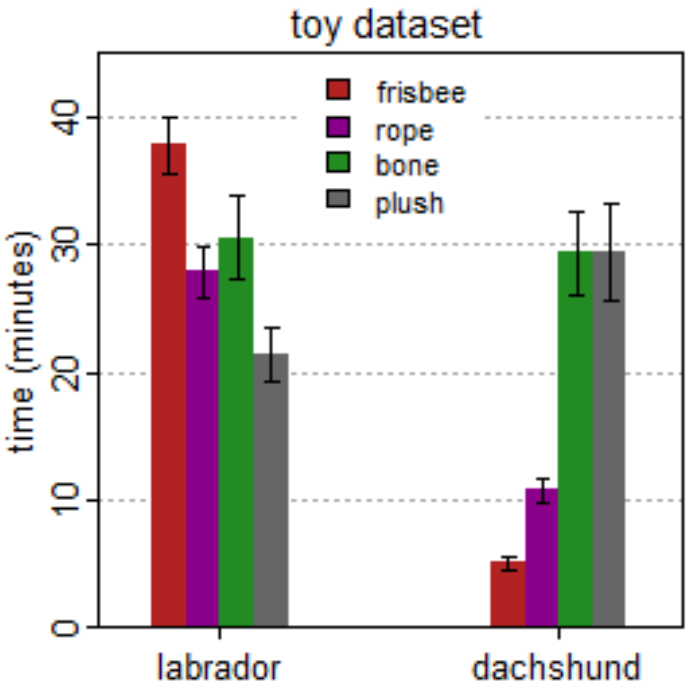
I think would need to go back to drawing a blank plot, then adding the grid, then using barplot() with add=TRUE.

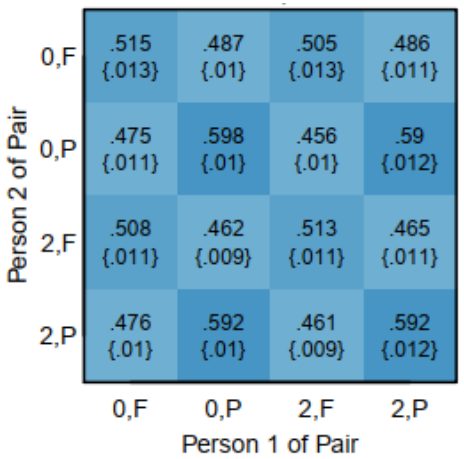
Is barplot() easier than rect()? The built-in functions useful in general?

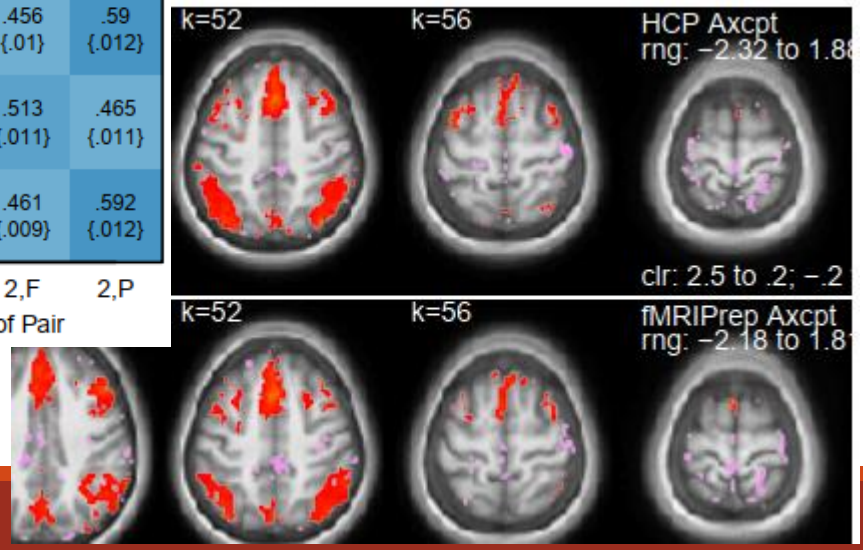
Not really, in my opinion; I generally add points(), lines(), rect(), and text() to blank plots.

Exceptions for me are specialized plotting functions: boxplot() and image(), and occasionally mosaicplot() and pairs() (scatterplot matrices).









made with image()

Layout: arranging more than one plot

I suggest layout(matrix()) for arranging groups of plots, not the others (e.g., par(mfrow))

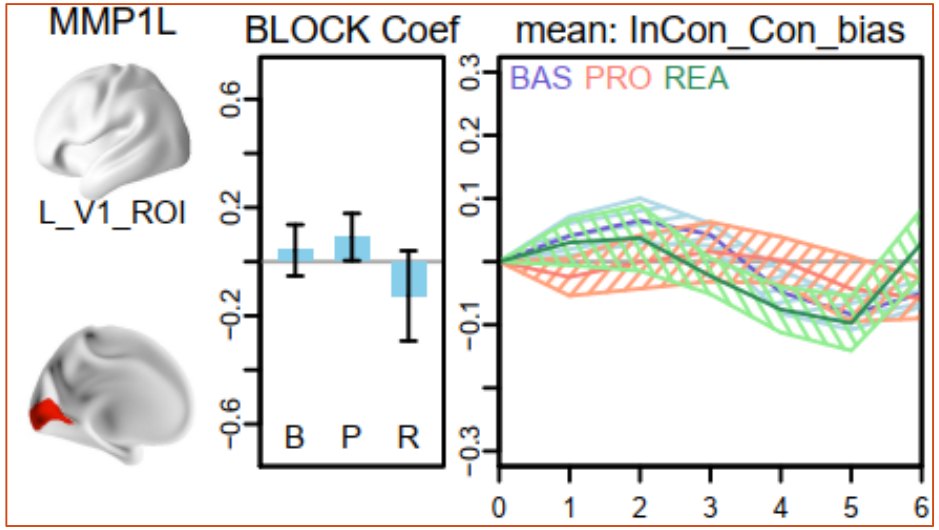
Put the layout code after the window size and before the par() for margins. The following plots will be drawn in the next window (in order).

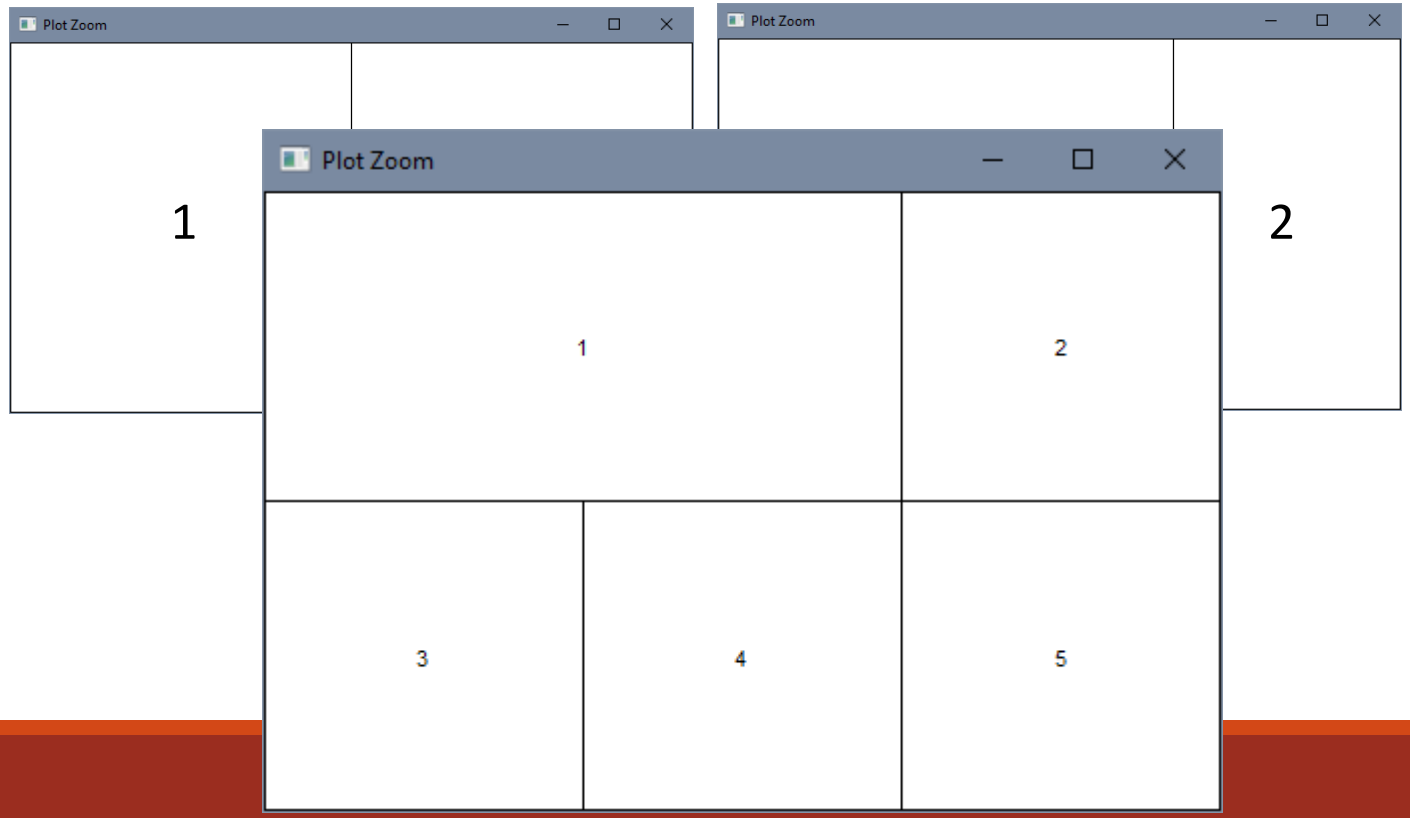
layout(matrix(c(1,2), c(1,2))); # two plots side-by-side, equal sizes

layout(matrix(c(1,1,2), c(1,3))); # two plots side-by-side, first twice as wide as the second

To test, use **layout.show(2);**, where the number is the number of graph spots you expect to see.

These can get complex, such as:
layout(matrix(c(1,1, 2, 3, 4, 5), nrow=2, byrow=TRUE));
layout.show(5);





Setting parameters (font size, margin width, title spacing, etc.)

R will guess settings based on the plot window size; resizing plots will change the sizes (usually). R generally guesses badly, and resizing by mouse can make disasters; much less frustrating to set window size, then adjust fonts, etc. to match (as in these demos).

cex=1 is R's size guess for most base R graphics. Bigger numbers are bigger (cex=2 is twice R's guess); smaller, smaller.

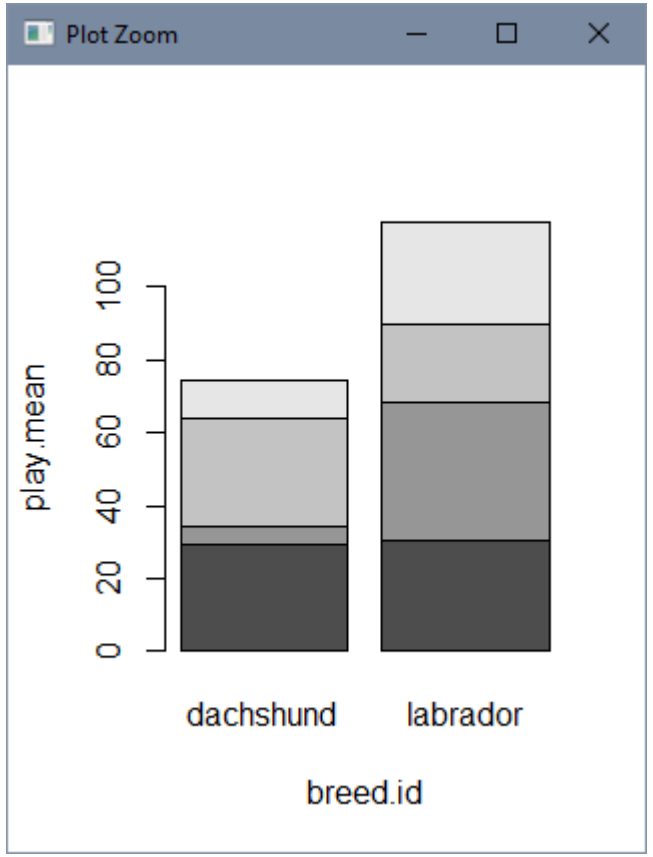
If there's only one size to change, the functions usually have one cex:

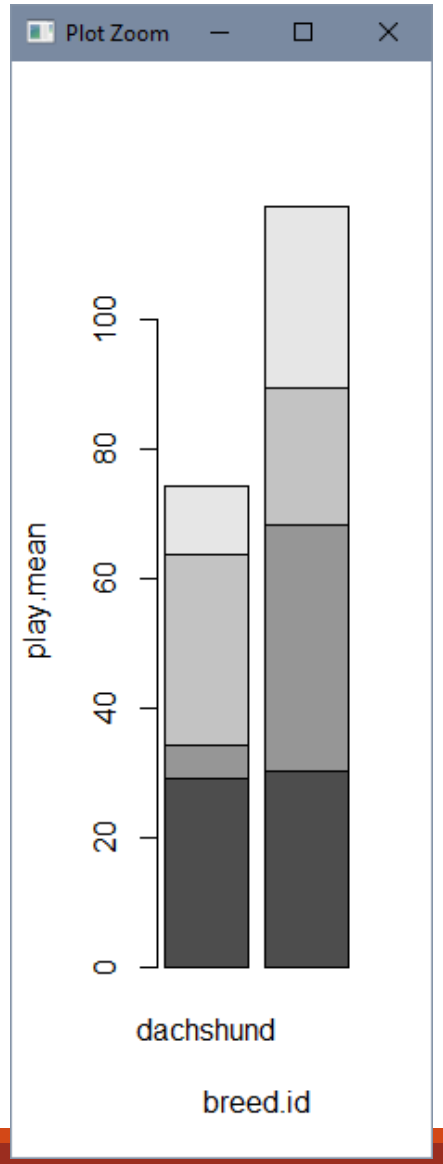
- points(): cex changes character size
- legend(): changes entire legend size, box, text, etc. at once
- boxplot(): cex changes outlier point size

But if there are multiple things, there are variants, such as for plot():

- cex.lab changes axis label size
- cex.axis changes tick text size
- cex.main changes top title size

Consult the help and cheat sheets for other parameters, such as plotting characters (pch), line types (lty), etc.





Parting advice

I suggest using words instead of numbers for clarity when possible (e.g., **lty='dashed'** instead of **lty=2**), but some things don't have word options.

For example: plotting characters (points() and plot()) are set by **pch**.

No one can remember all of these esoteric settings. There are many "cheat sheets", online posts, and books to help; Google is your friend.

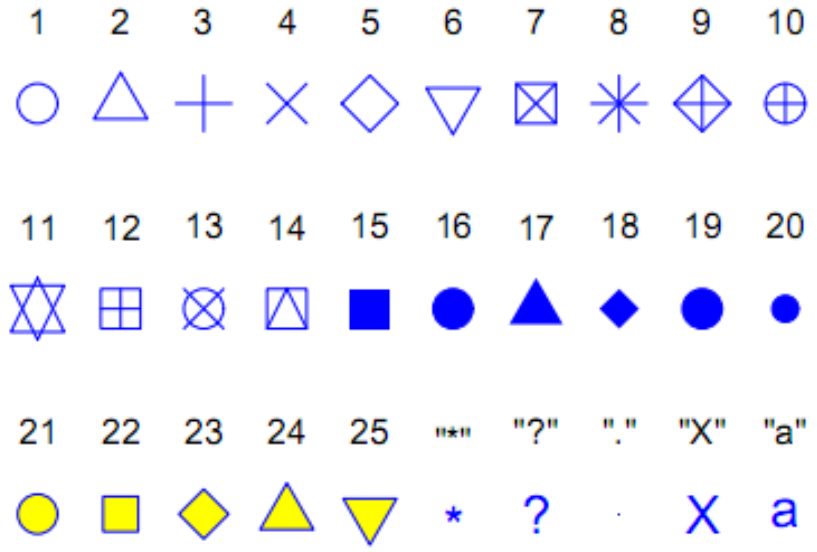


Figure 2: The plotting symbols in R (`pch=1:25`). The colours were obtained with the options `col="blue"`, `bg="yellow"`, the second option has an effect only for the symbols 21–25. Any character can be used (`pch="*"`, `"?"`, `"."`, ...).

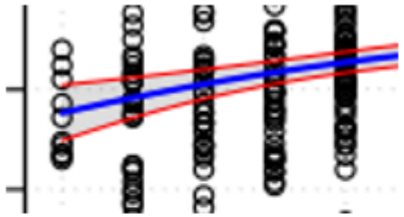
(I lost this source.)

https://www.r-graph-gallery.com/
www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
http://www.gastonsanchez.com/visually-enforced/resources/2015/09/22/R-cheat-sheet-graphical-parameters/
http://www.joyce-robbins.com/blog/2016/04/20/r-base-graphics-cheatsheet/
https://rstudio.com/resources/cheatsheets/

A bit more parting advice

I strongly suggest you start your own “cheat sheet” or collection of demo code. I save snippets of difficult plots in **OneNote** with brief descriptions of how to do it or a link to the file with the relevant code.

> … semi transparency, two-row axis labels, adding little thumbnail images to existing plots, plotting with different y-axis scaling on the same window, etc., etc., etc.

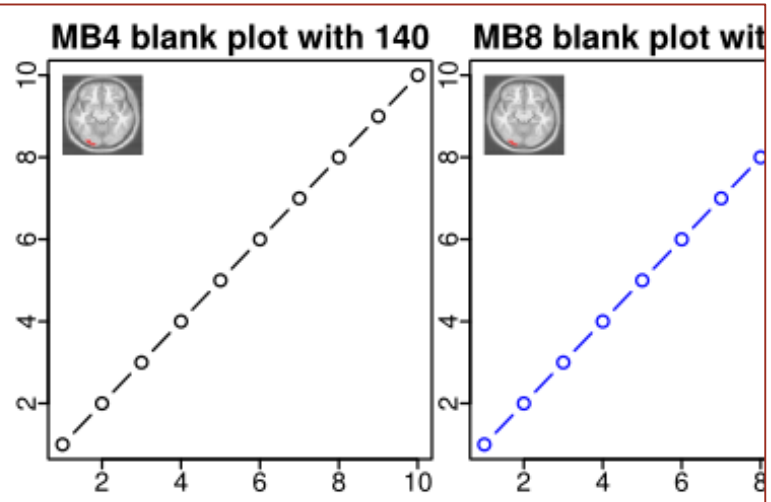


semi-transparent colors:

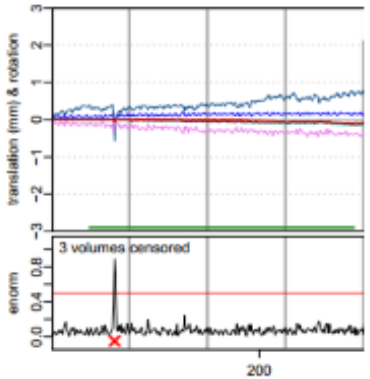
```
get.color <- function(clr) {    # clr <- "grey";
  # adapted from http://research.stowers.org/mcm/efg/R/Color/Chart/
  # add 22 onto the end for transparency, see # http://r.789695.n4.nabble.com/How-to-draw-a-transparent-polygon-td4690978.html
  c <- col2rgb(clr)
  return(paste0(sprintf("#%02X%02X%02X", c[1],c[2],c[3], c[1], c[2], c[3]), "22"));
}
```

shaded curves:

```
# shade the confidence interval; adapted from http://www.alisonsinclair.ca/2011/03/shading-between-curves-in-r/
  polygon(c(xval$x,rev(xval$x)),c(pr.out$fit[,"upr"],rev(pr.out$fit[,"lwr"])), col=get.color("black"), border=NA);
```



"D:\gitFiles_ccplabwustl\R01\Jo\knitr\knitrBrainInset\knitrBrainInset.rnw"
10/3/2016 11:59 AM - Screen Clipping
see also D:\svnFiles\demoCode\knitRdemo\brainInsetBarCharts



"D:\gitFiles_ccplabwustl\R01\Jo\for800msecTR\knitr\fMRI_movementSumma
\template_fMRI_movementSummary.rnw"
10/27/2016 4:59 PM - Screen Clipping

Two plots close together, using
par(fig=c(0,1,0.4,1), mar=c(0.1, 1.75, 1, 0.75))
plot()

Jo Etzel is supported by NIH R37MH066078 to Todd Braver.

base R graphics

Joset A. Etzel, PhD
jetzel@wustl.edu | mvpa.blogspot.com | @JosetAEtzel
Cognitive Control and Psychopathology Lab
Washington University in St. Louis

Suggested Exercises

- Change the plotting symbols for each toy to from dots to xs. Make the symbols larger and change their color.
- Capitalize the labels and change their size.
- Make the margins larger or smaller. Move the title farther from the plot.
- Move the bars and/or columns of points farther apart.
- Show the mean for each toy and breed on the same plot as the dots for the full dataset.
- Draw two plots side-by-side or up-and-down with layout.
- Add words to the plot with the text() command and eliminate the legend.
- Draw boxplots with points alongside (to show the entire distribution, rug-style).
- Connect the times for each individual dog with lines (e.g., to see if some dogs play with all four toys longer than other dogs).