

# Patience: Base package graphics

advanced plots in the base package

# Graphical Functions

- Two types of plotting function in the **base package**:
  - ✓ high-level, e.g., `plot()`, `boxplot()`, `image()`
  - ✓ low-level, e.g., `title()`, `points()`, `abline()`
- In general, **modularity is good**, meaning use low level function to decorate plots.
- Let's start with something familiar - the “mtcars” dataset.

# An Old Favourite

**Do it yourself:** Get it into the memory:

```
> attach(mtcars)
```

Refresh yours!

```
> mtcars ↵
```

```
> ?mtcars↵
```

```
> View(mtcars) ↵
```

```
> str(mtcars) ↵
```

```
> plot(mtcars) ↵
```

Notice how insane the plot() function can be = all against all. Let's make things behave properly.

# Bar Charts and Aggregating Counts

We can use `barplot` for plotting counts (in which both the x and y variable are **categorical**).

**Do it yourself:** Construct a `barplot` for number of gears in `mtcars` data set:

```
> gear ←
```

```
> counts <- table(gear) ←
```

Yes, `'table()'` tabulates count data = cool...

```
> counts ←
```

```
> barplot(counts) ←
```

That looks alright, but a bit simple...

# Elaborating Plots

**Do it yourself:** plot the density of the counts:

```
> barplot(counts/length(gear),  
names.arg=c("three", "four", "five")) ←
```

and now label our plot:

```
> title(main="Car Distribution",  
xlab="Number of Gears", ylab="Relative  
Frequency") ←
```

We wrote decorative text using a low level function.

# Bar Charts with Point Estimates

We can use `barplot` for point estimates of a **continuous** y variable; we need an `*apply` function.

**Do it yourself:** calculate the mean mpg rating for cars with different numbers of gears:

```
> heights <- tapply(mpg, gear, mean) ␣
```

and now plot and label these:

```
> barplot(heights) ␣
```

```
> title(main="Mean Efficiency",  
xlab="Number of Gears", ylab="Miles  
per Gallon")
```

# Confidence Intervals

We can also plot error bars. For this we need an extra library\*.

**Do it yourself:** install and then load the package gplots:

```
> install.packages('gplots')  
> library(gplots)
```

This package contains an enhanced version of the `barplot()` function called `barplot2()`.

\* it is also possible to plot error bars with the base package + the low-level function `arrows()`.



# Confidence Intervals

**Do it yourself:** here we exploit summary stats produced by the `t.test()` function\*:

```
> lower <- tapply(mpg, gear, function(i)
t.test(i)$conf.int[1]) ␣
> upper <- tapply(mpg, gear, function(i)
t.test(i)$conf.int[2]) ␣
```

**Plot the graph and decorate:**

```
> barplot2(heights, plot.ci=TRUE, ci.l=lower,
ci.u=upper, ci.width=0.2) ␣
> title(main="Car Efficiency", xlab="Number of
Gears", ylab="Miles per Gallon") ␣
```



# Multiple Predictors

**Do it yourself:** Construct a `barplot` for number of gears in `mtcars` data set grouped by `vs`:

```
> counts <- table(vs, gear) ␣  
> barplot(counts,  
col=c("lightblue", "mistyrose"), legend=T,  
args.legend=list(title="vs")) ␣  
> title(main="Car Distribution by Gears  
and VS", xlab="Number of Gears") ␣
```

Try adding `beside=TRUE` as an argument to the `barplot()` call. **What happens?**

# Distribution Plots: Boxplot

You can use `boxplot()` to get an idea of the distribution of  $y$  ~ (as affected by)  $x$  with an intuitive model formula.

## Do it yourself

```
> boxplot(mpg ~ am,  
names=c("Automatic", "Manual")) ␣  
> title(main="Boxplot Showing  
Distribution", xlab="Transmission",  
ylab="Miles per US Gallon") ␣
```

Something like `tapply()` is happening behind the scenes.

## Warning!!!



**Do it yourself** This is what happens when you pass variables without the model notation:

```
> boxplot(am, mpg, names=c("all am  
value", "all mpg values"))←
```

So there's nothing wrong with passing vectors to `boxplot()`, but don't expect `tapply()` behaviour unless you use `~` notation.

# Distribution Plots: Violin Plots

- A nice alternative is `vioplot()`:

## **Do it yourself**

Start by installing and loading the required library:

```
> install.packages('vioplot')  
> library(vioplot)
```

# Subsetting Data

- The ``subset()`` function helps to split a variable/vector using a factor:

**Do it yourself:** Get subsets of data:

```
> auto <- subset(mpg, am==0,  
data=mtcars) ↵
```

```
> man <- subset(mpg, am==1,  
data=mtcars) ↵
```

See my notes for the tidyverse approach to this (which I prefer).

# Violin Plots

- Now we can try a violin plot:

**Do it yourself:** Now we can plot:

```
> vioplot(auto, man,  
names=c("Automatic", "Manual")) ␣  
> title(main="Violin Plot",  
xlab="Transmission", ylab="Miles per  
US Gallon") ␣
```

- But ***be careful*** – this does not support the formula notation hence the need for vector splitting...

# Distribution Plots: Histograms

**Do it yourself:** Let's compare distributions:

```
> hist(auto, col=rgb(1,0,0,0.5),  
breaks=seq(10,36,2), xlim=c(10,35),  
ylim=c(0,5), main="", xlab="") ␣  
> hist(man, col=rgb(0,0,1,0.5),  
breaks=seq(10,36,2), add=T) ␣
```

**Decorate:**

```
> title(main="Double Histogram", xlab="Miles  
per Gallon") ␣  
> legend("topright", c("automatic", "manual"),  
fill=c(rgb(1,0,0,0.5), rgb(0,0,1,0.5))) ␣
```



# Advanced Boxplots (base)

The formula notation in `boxplot` supports interrogation of multiple predictors.

**Do it yourself:** We can plot by two categorical predictors using `boxplot` too.

```
> boxplot(mpg~vs*am, data=mtcars,  
col=(c("mistyrose", "lightblue")))  
  
> title(main="Car Engines", xlab="Config *  
Transmission", ylab="Miles per gallon") ←
```

Try adding a `notch=TRUE` argument to the plot call.

I'll leave further decoration to you...

# Cleaning Up

**Do it yourself** Detach from the dataset

```
> detach(mtcars) ↵
```

You can also clean up variables:

```
> rm(auto, man) ↵
```

```
> rm(list=ls()) ↵
```

Using `rm()` is playing with fire!

# Controlling Graphical Parameters

## A microarray with minimal data...

**Do it yourself:** Here we load some data from a file:

```
> setwd('~/Desktop/render-master') ←  
> array <- read.csv('heatmaps_in_r.csv',  
header = TRUE, comment.char="#", row.names =  
1) ←
```

We reset some graphical parameters and make our heatmap:

```
> op <- par(cex.axis=0.6, las=2) ←  
> image(x=1:10, y=1:4, z=data.matrix(array),  
col=rev(heat.colors(10)), xlab='', ylab='') ←  
> hist(rnorm(100)) ←
```

# Controlling Graphics

**Do it yourself:** Reset graphics back to original:

```
> par(op) ↵
```

```
> hist(rnorm(100)) ↵
```

Now you can nerd out with graphical parameters:

```
> ?par ↵
```

```
> ?points ↵
```

```
> ?colours ↵
```

```
> ?image ↵
```

Take home: parameters can be set and reset within each device. Meaning...

# Writing to a Device

**Do it yourself:** Now let's get serious with graphics parameters and plotting devices:

```
> pdf("nanoarray.pdf", 12, 6) ␣  
> par(cex.axis=0.6, las=2) ␣  
> image(x=1:10, y=1:4, z=data.matrix(array),  
col=rev(heat.colors(10)), xlab='', ylab='',  
axes=F) ␣  
> axis(1, at=1:10, labels=rownames(array)) ␣  
> axis(2, at=1:4, labels=colnames(array)) ␣  
> title(xlab="sample", ylab="probe",  
cex.main=2) ␣  
> dev.off() ␣
```

# Using source() to run all lines in script

- There is a script called makearray.R in the render-master folder:

**Do it yourself:** run this script:

```
> source('makearray.R') ↵
```

- What has this script done?
- Hints:
  - Look in the folder again for new files
  - Open the script using RStudio or a text editor
  - Can you see any new objects in memory?



# In RStudio's Environment

The screenshot displays the RStudio interface with three main panes. The top-left pane shows an R script named 'makearray.R' with the following code:

```
1 # This is a simple R script. The hash indicates that a line should be ignored
2 # by the R interpreter.
3
4 # Let's start by defining a function:
5 rescale <- function(vec){
6   floor = min(vec)
7   extent = max(vec) - floor
8   return( as.vector( scale(vec, center=floor, scale=extent) ) )
9 }
10
11
12 # note this is built using the scale() base function - take a look (using ?scale)
13
14 # Fetching the data:
15 array <- read.csv('heatmaps_in_r.csv', header = TRUE, comment.char="#", row.names = 1)
16
17 # Let's make our new array:
18 array_norm_byprobe <- apply(array, 2, rescale)
19
20 # And then we can make our image:
21 pdf("nanoarray_byprobe.pdf", 12, 6)
22 par(cex.axis=0.6, las=2)
23 image(x=1:10, y=1:4, z=data.matrix(array_norm_byprobe), col=rev(heat.colors(10)), xlab='', ylab='', axes=F)
24 axis(1, at=1:10, labels=row.names(array_norm_byprobe))
25 axis(2, at=1:4, labels=colnames(array_norm_byprobe))
26 title(xlab="sample", ylab="probe", cex.main=2)
27 dev.off()
28
```

The top-right pane shows the Environment pane with the following data and functions:

Global Environment	
<b>Data</b>	
array	10 obs. of 4 variables
array_norm_byprobe	num [1:10, 1:4] 0.449 0.461 0.508 0.384 0.489 ...
<b>Functions</b>	
rescale	function (vec)

The bottom-left pane shows the Console with the following output:

```
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> setwd("~/Github/render/")
> source("~/Github/render/makearray.R")
>
```

The bottom-right pane shows the Files pane with the following files and folders:

Name	Size	Modified
..		
.Rhistory	67 B	Sep 11, 2017, 12:33 AM
ANOVA.txt	223 B	May 12, 2017, 6:19 PM
heatmap		
heatmaps_in_r.csv	473 B	May 12, 2017, 6:19 PM
makearray.R	924 B	Sep 11, 2017, 12:33 AM
nanoarray_byprobe.pdf	4.6 KB	Sep 11, 2017, 12:34 AM
prescription.xlsx	46.3 KB	May 12, 2017, 6:19 PM
README.md	5.8 KB	May 14, 2017, 12:14 AM
simple.txt	118 B	May 12, 2017, 6:19 PM
smoking.csv	175 B	May 12, 2017, 6:19 PM
time_series		
Training Workshop 13 May 2017.pdf	947.5 KB	May 12, 2017, 6:19 PM