3. More Advanced Graphics

Thomas Lumley
Ken Rice

Universities of Washington and Auckland

Lausanne, September 2014

Outline

- Colour and pre-attentive perception: facts about graphics

- Too many variables: parallel coordinates, transparency

- Too many dimensions: hexagonal binning, transparency

Colour coding

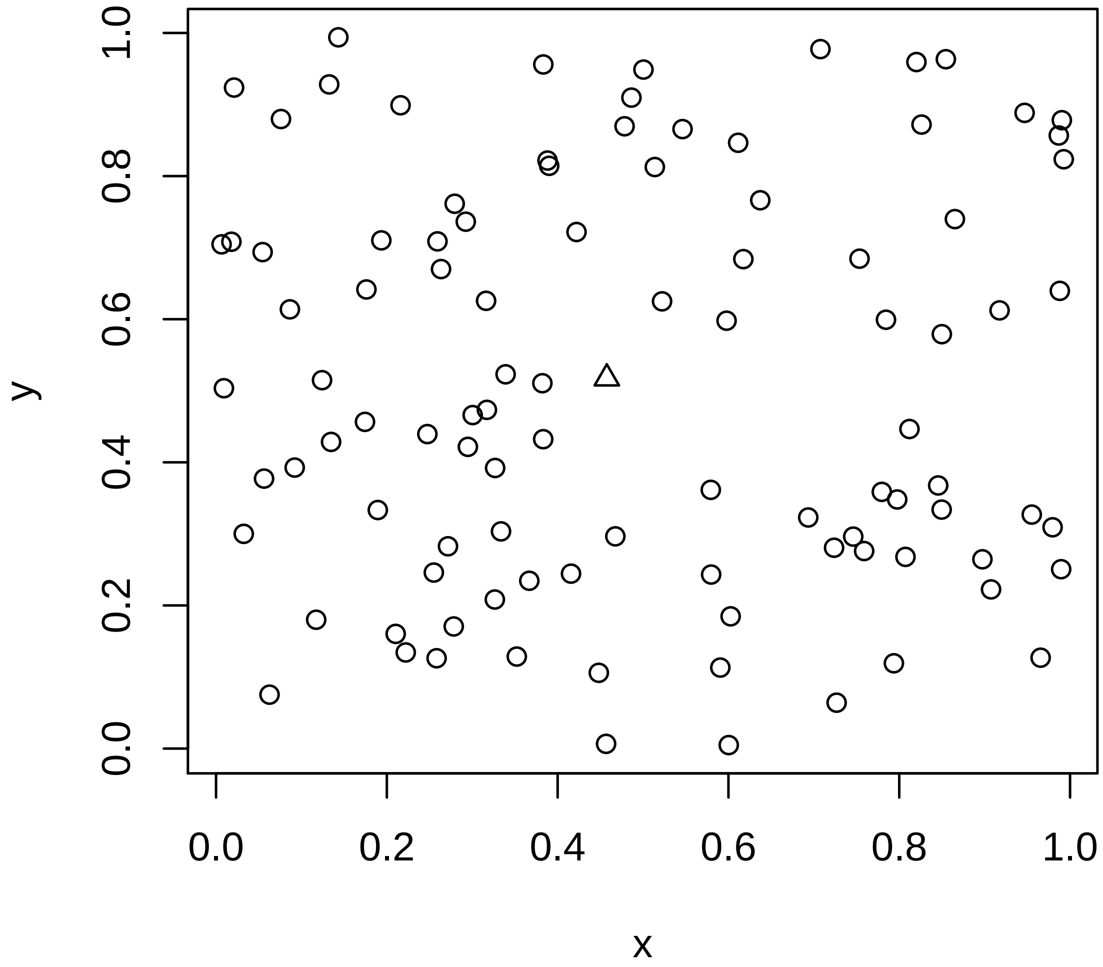
‘Simple’ plots involve two-dimensional data, which we measure on the x and y axes.

For higher-dimensions, some traditional approaches are;

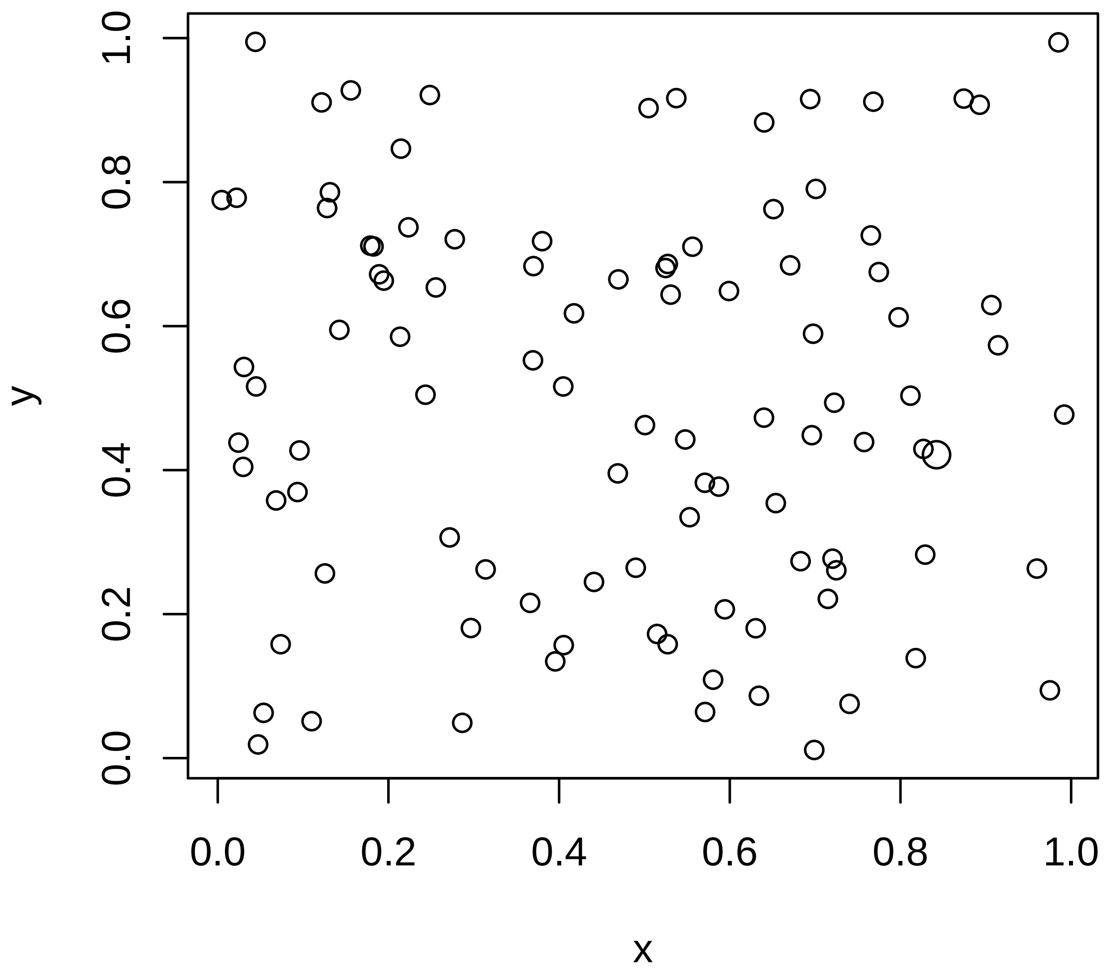
- Different colors for e.g. men, women (`col`)
- Different-shaped symbols (`pch`), or different sizes (`cex`)

For $\leq$ 100’s of data points, modest use of these is fine. But your eye is not good at concentrating e.g. just on the purple points, in a fully Technicolor plot;

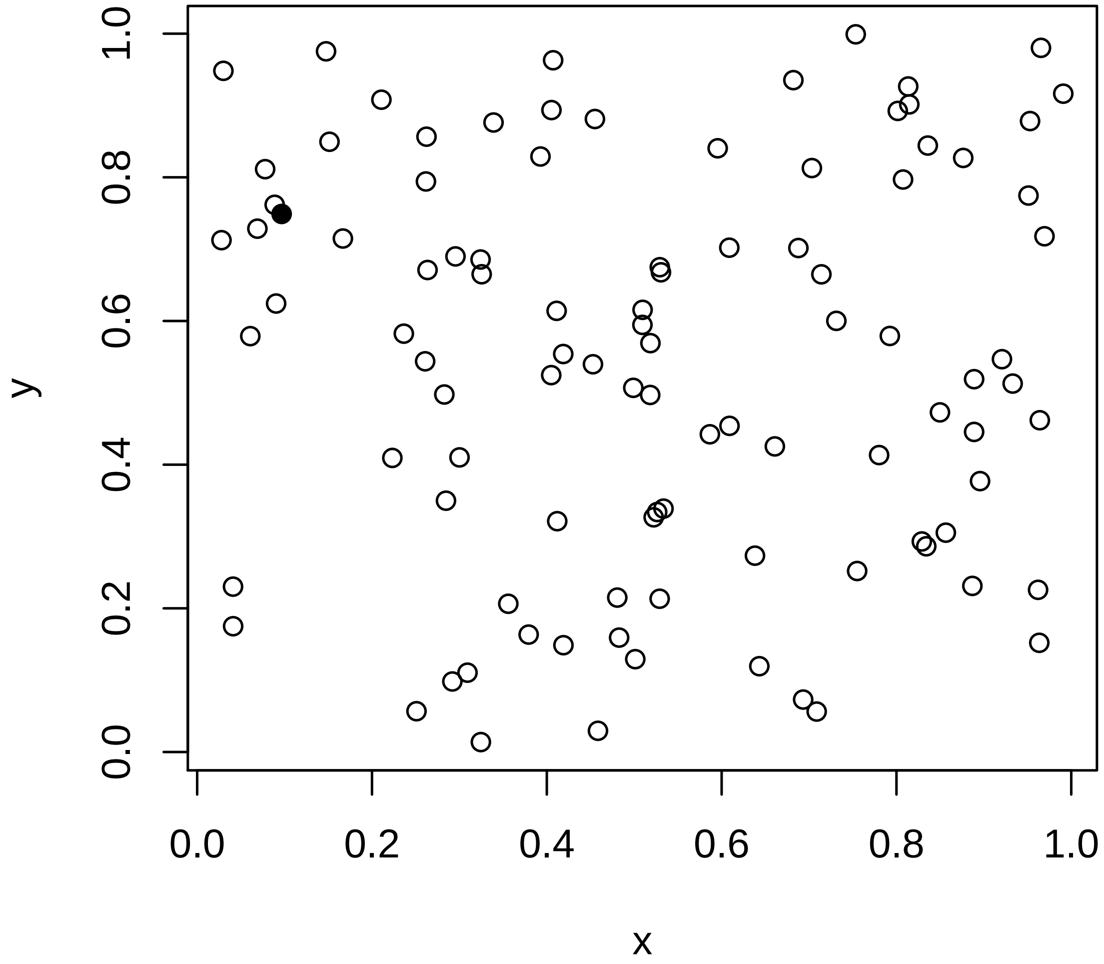
Which point is different?



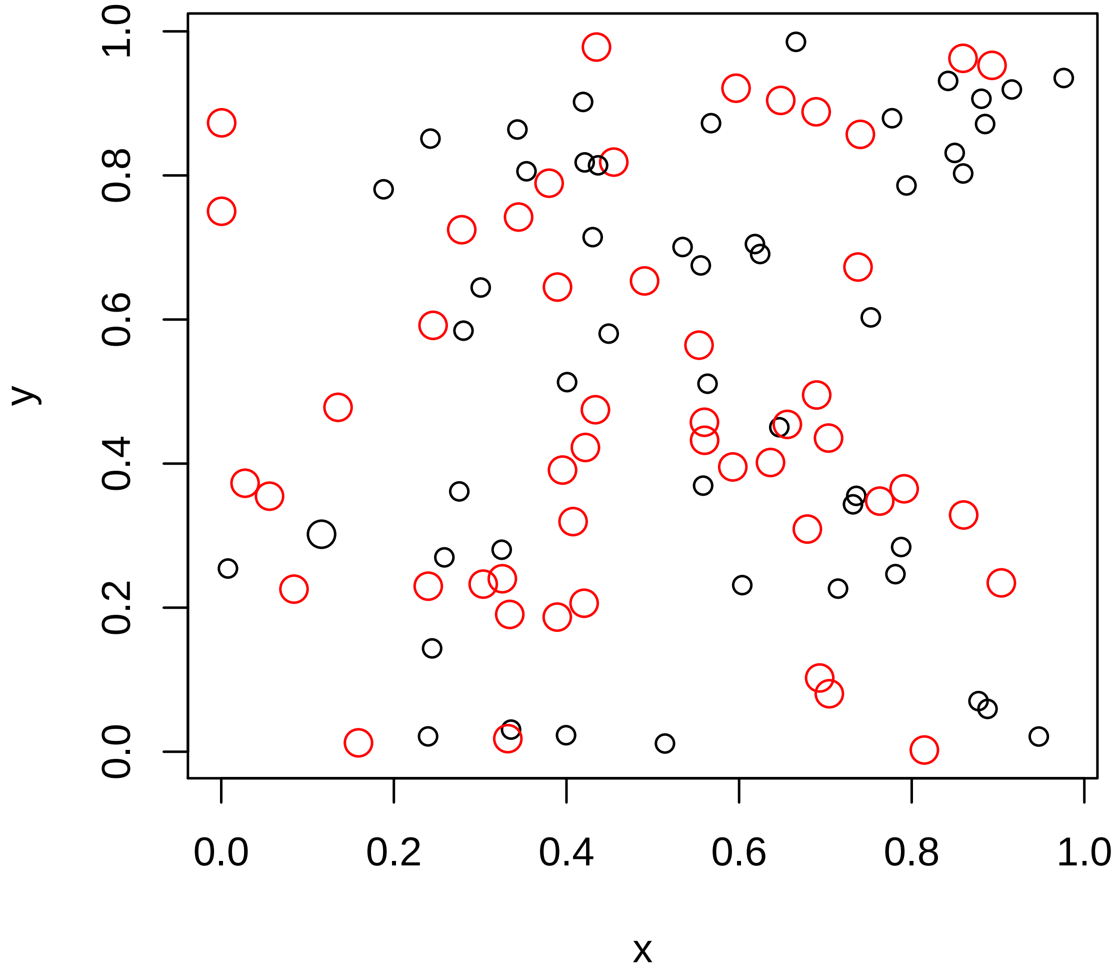
Which point is different?



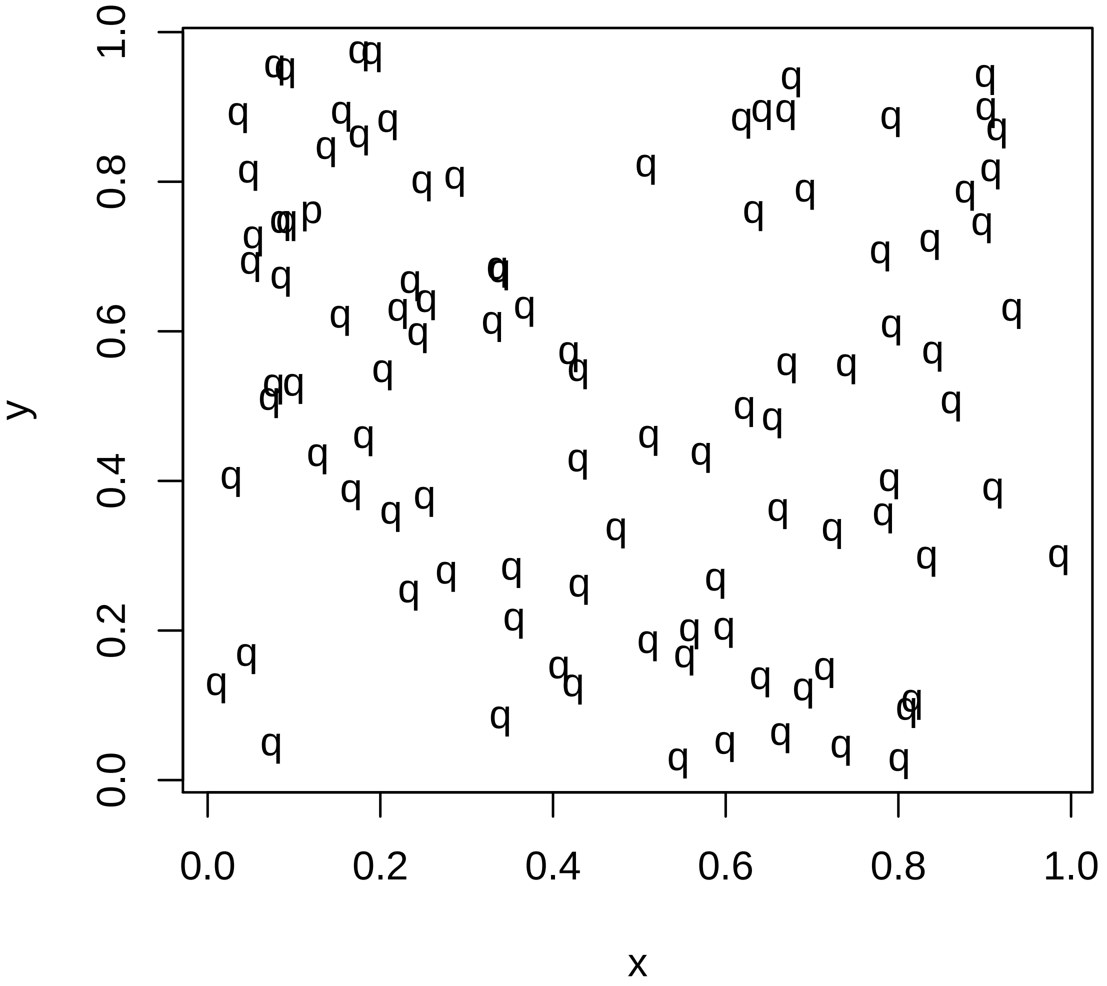
Which point is different?



Which point is different?



Which point is different?



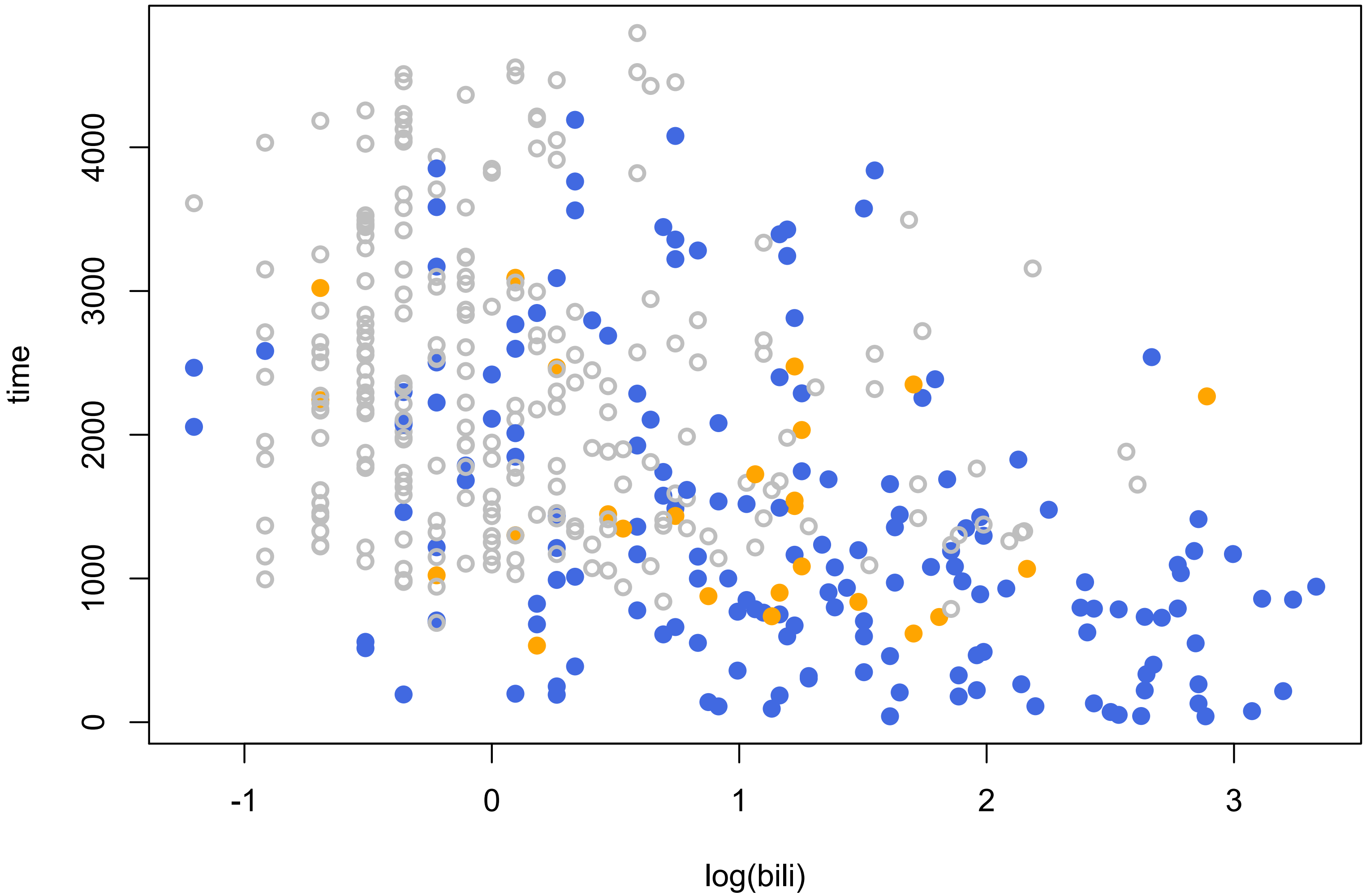
Preattentive perception

Some differences are processed by the brain before you get to see the image: pre-attentive perception.

Important because

- It's easier to see things
- You can look at just one subset of the points and see patterns
- Like colour-blindness, illustrates that there are **facts** about graphics, not just artistic taste

Preattentive perception



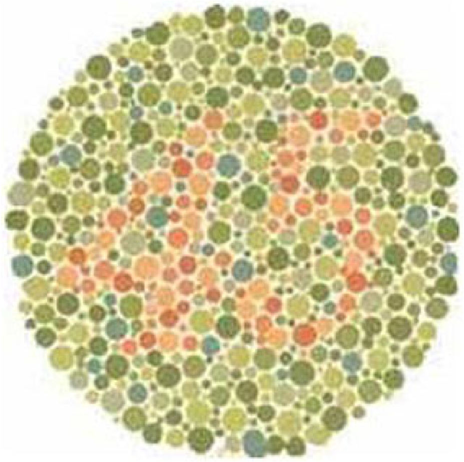
Color schemes

Color choice is best left to experts, or people with taste.

`http://www.colorbrewer.org` has color schemes designed for the National Cancer Atlas, also in package `RColorBrewer`

`colorspace` package has color schemes based on straight lines in a perceptually-based color space (rather than RGB).

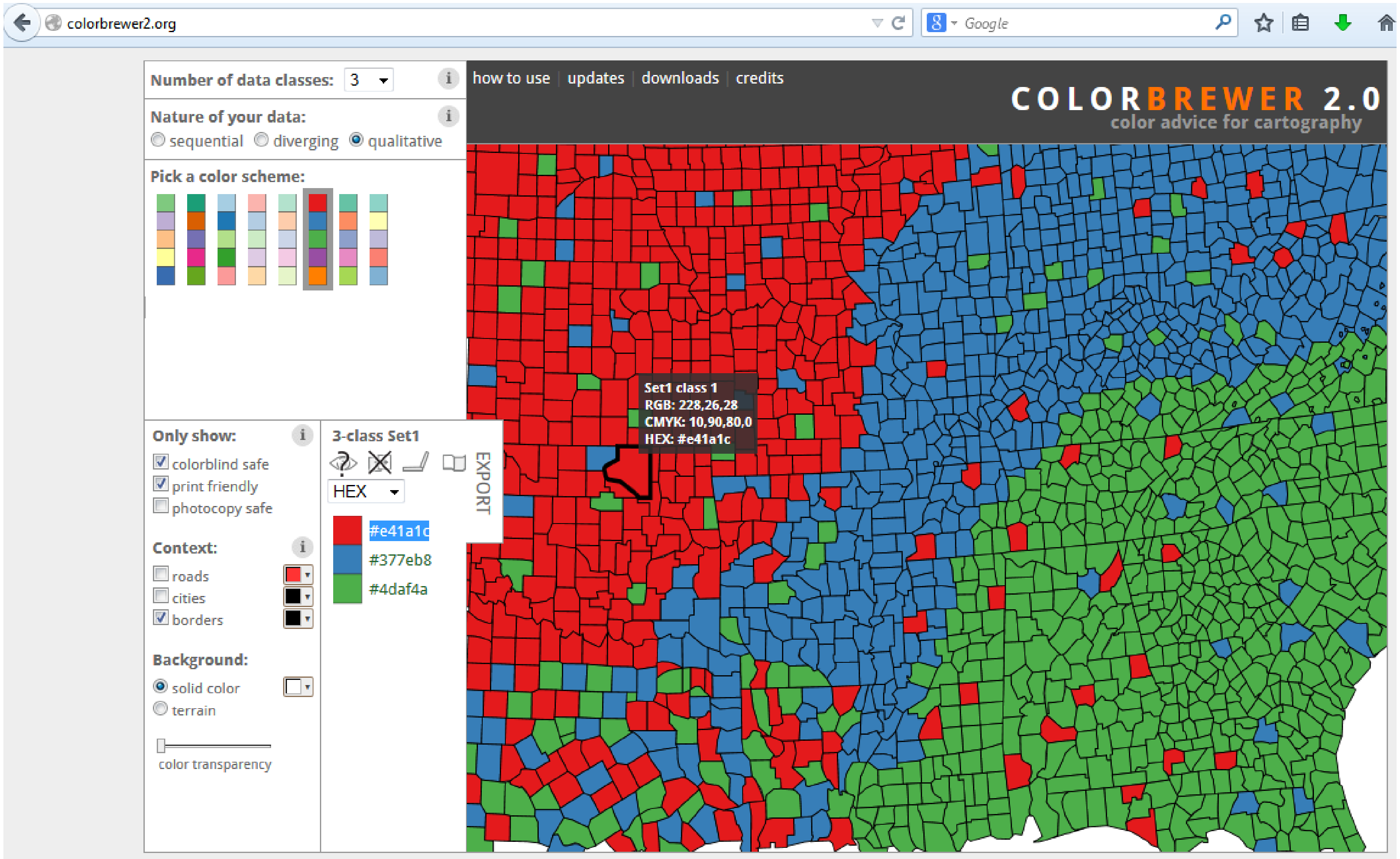
`dichromat` package attempts to show the impact of red:green color blindness on your R color schemes.



[Code for examples is in file `colorpalettes.R` on course website]

Color brewer

R accepts ‘hex’ colors, e.g. `col="#e41a1c"` here;



colorbrewer2.org

Number of data classes: 3

Nature of your data:
○ sequential ○ diverging ● qualitative

Pick a color scheme:

Only show:
☑ colorblind safe
☑ print friendly
☐ photocopy safe

Context:
☐ roads
☐ cities
☑ borders

Background:
● solid color
○ terrain

color transparency

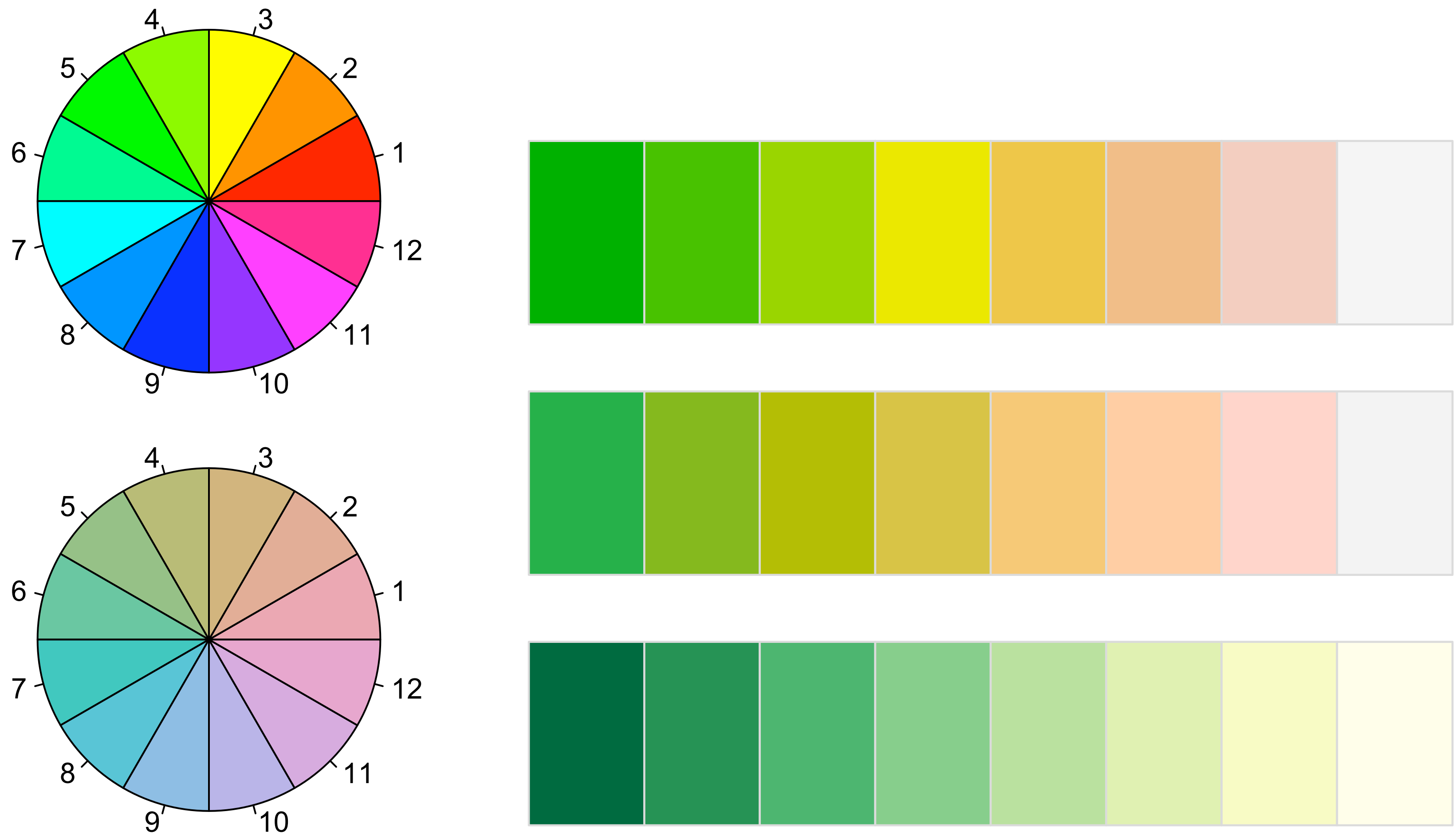
3-class Set1

HEX

🟥 #e41a1c
🟦 #377eb8
🟩 #4daf4a

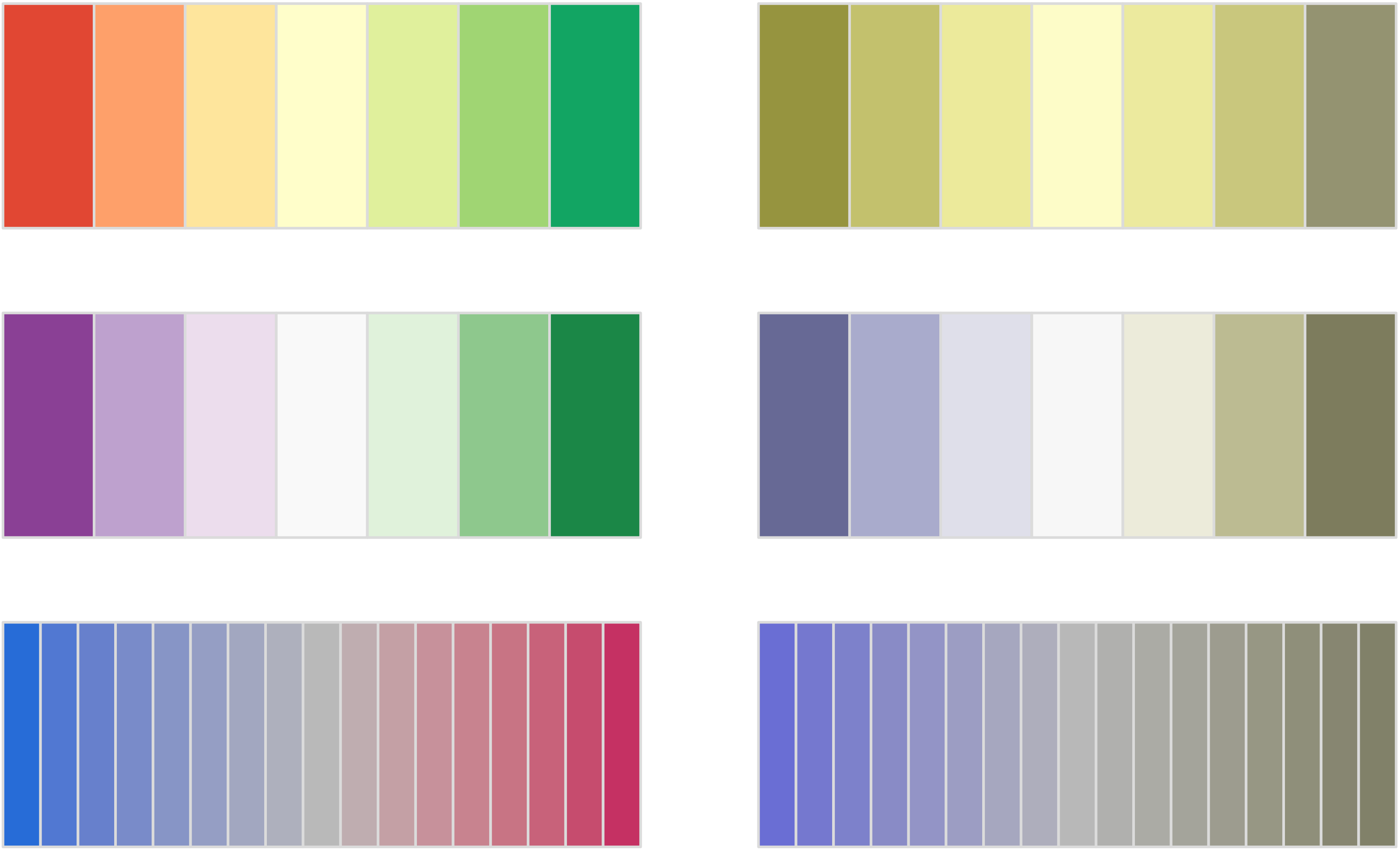
Set1 class 1
RGB: 228,26,28
CMYK: 10,90,80,0
HEX: #e41a1c

Color choice



(nb B&W printed copies of this slide may not be helpful!)

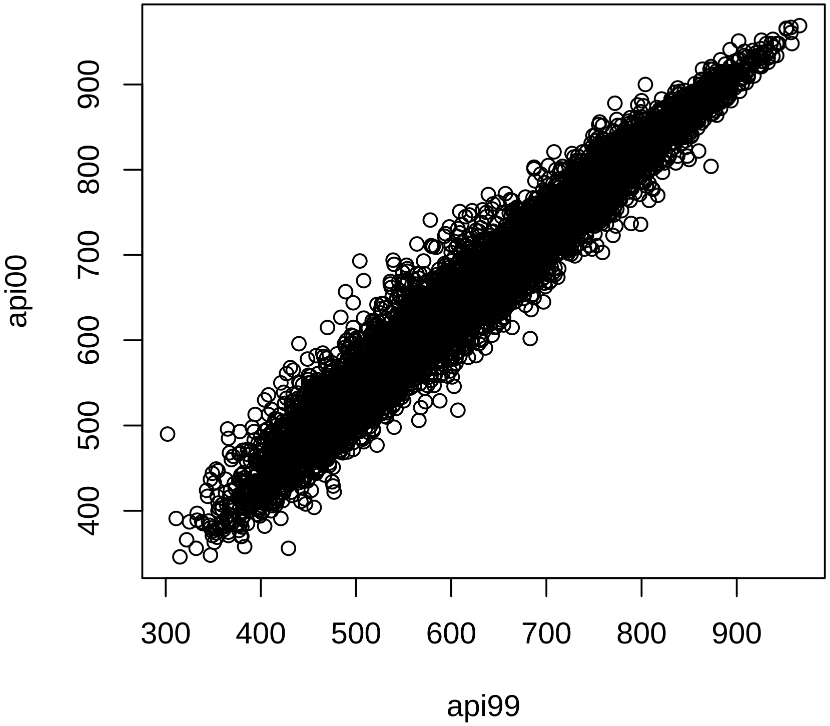
Color blindness



(nb B&W printed copies of this slide may not be helpful!)

Larger data

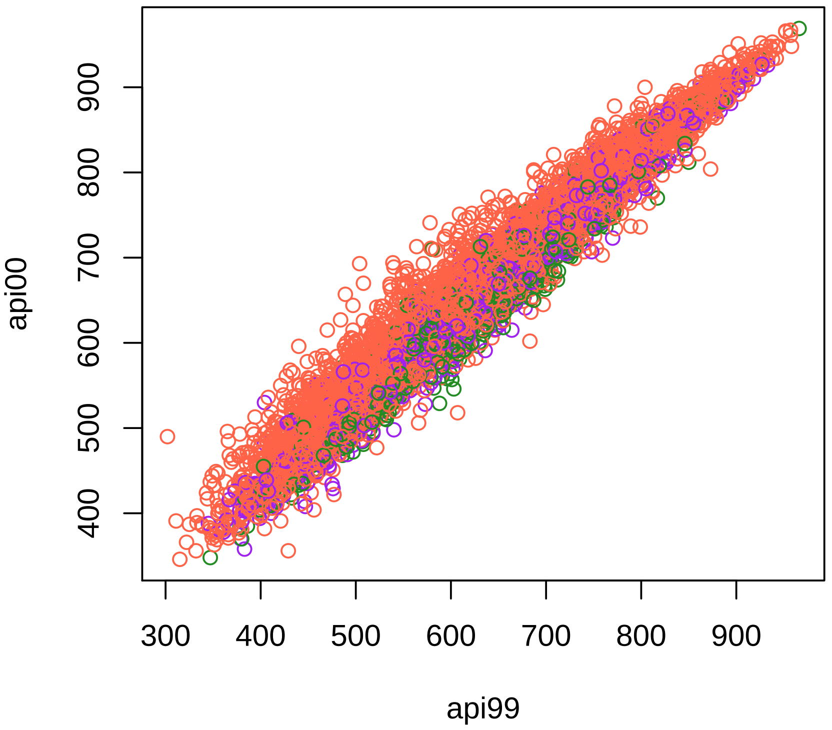
For large(ish) data, ‘overlap’ is a fundamental problem...



(California Academic Performance Index on 6194 schools)

Larger data

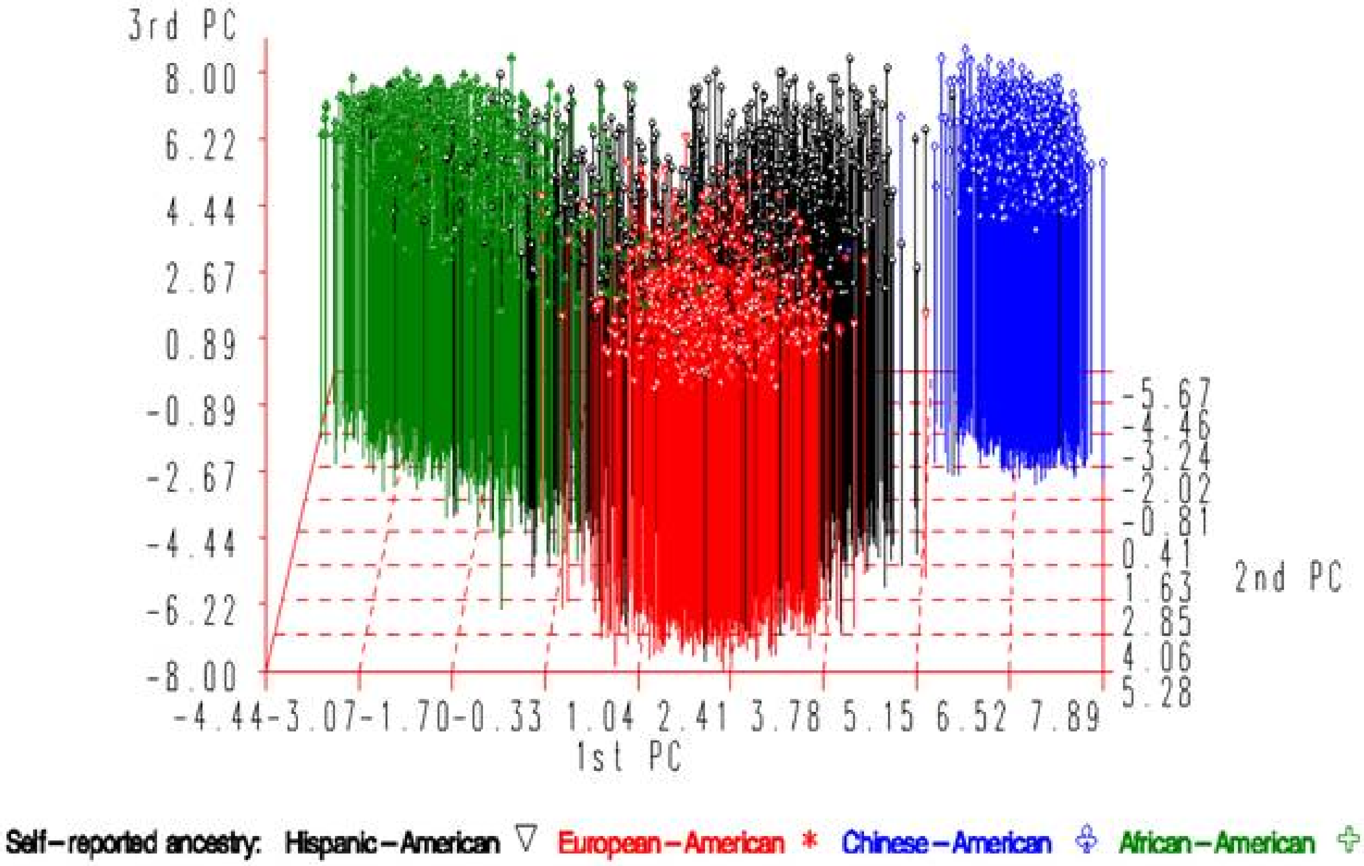
... which remains, when we color-code.



Colors denote Elementary, Middle & High Schools

Larger data

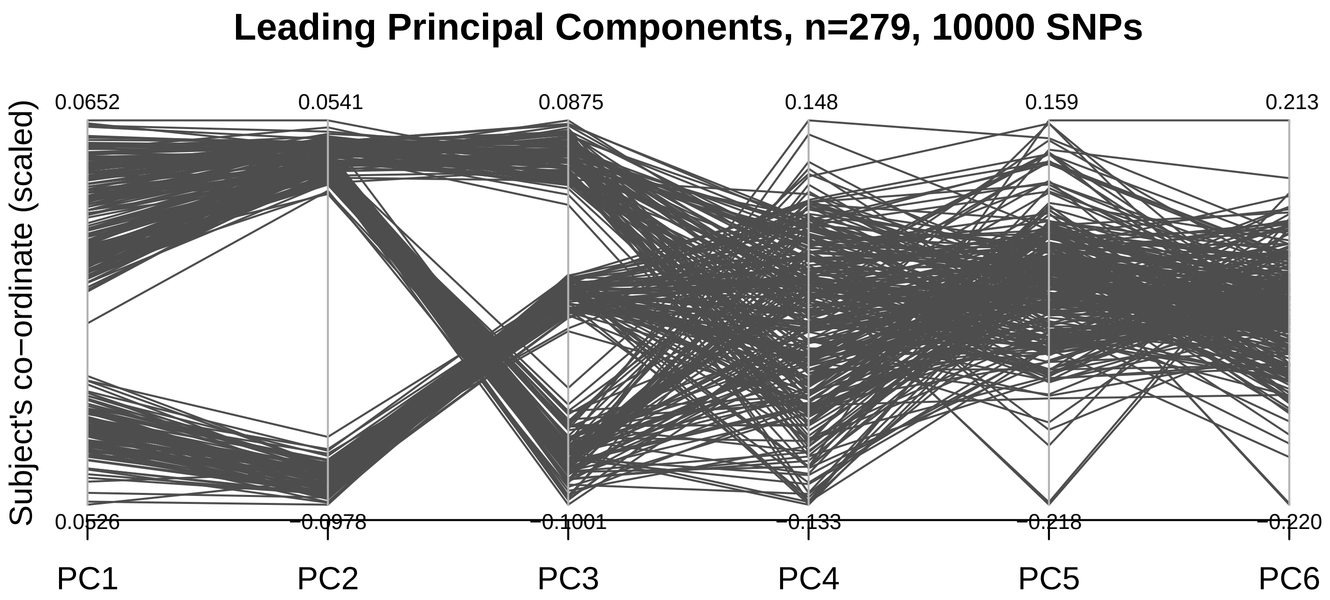
With three dimensions + color-codes, this can happen;



(R does have `persp()`, for occasional use)

Parallel Coordinate Plots

For even higher-dimensional data, scatterplots can not provide adequate summaries. For data where the dimensions can be ordered, the parallel co-ordinates plot is useful;

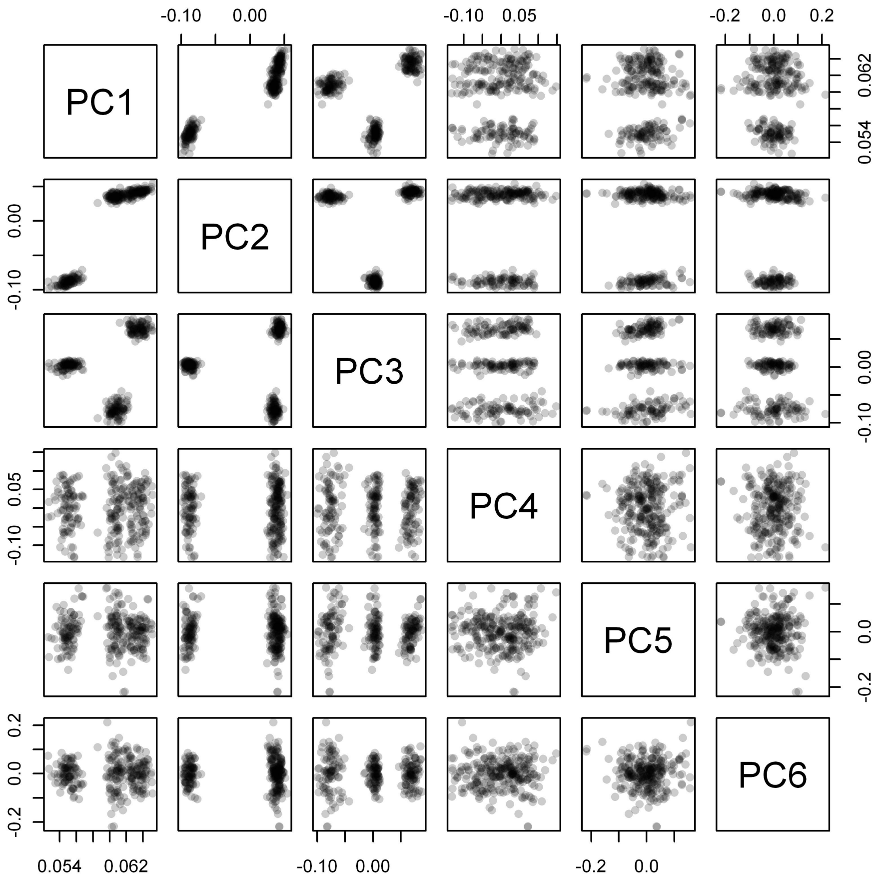


Parallel Coordinate Plots

- Each multi-dimensional data point (i.e. each person) is represented by a line – not a point
- `parcoord()` in the `MASS` package is one simple implementation – writing your own version is not a big job
- Coloring the lines also helps (example later)
- Scaling of axes, and their vertical positions are arbitrary
- Doing 'Principal Components Analysis' is just choosing axes for your data so that their variance is maximized on axis 1, then axis 2, ...

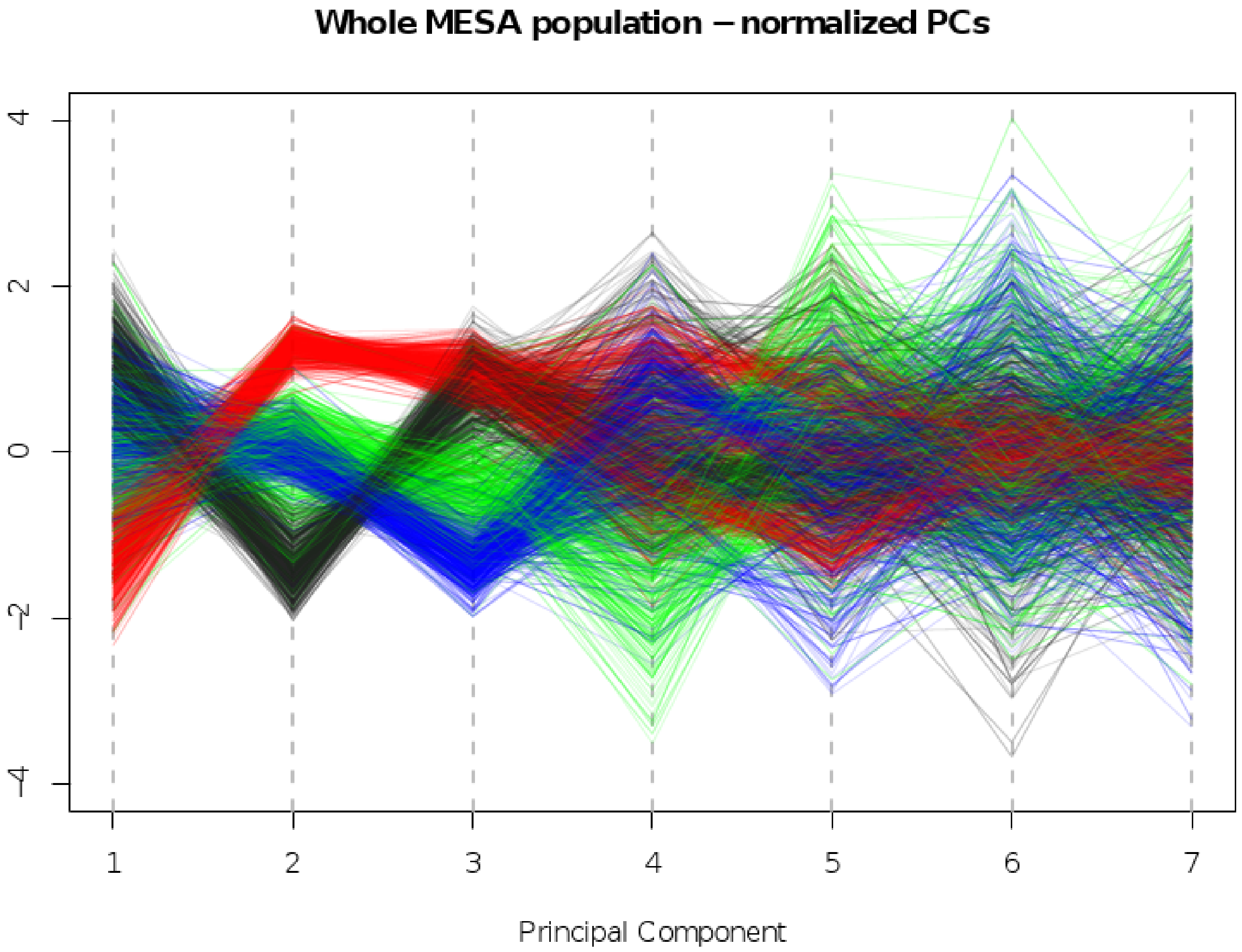
Parallel Coordinate Plots

A `pairs()` plot of the same thing; (nasty!)



Parallel Coordinate Plots

The pin cushion data++ : colors indicate self-report ancestry



Transparency

The colors in the last examples were transparent. As well as specifiying e.g. `col=2` or `col="red"`, you can also specify

`col="#FF000033"`

– coded as RRGGBB in hexadecimal, with transparency 33 (also hexadecimal). This is a 'pale' red – 33/FF ≈ 20%.

Get from color names to RGB with `col2rgb()`, and from base 10 to base 16 using `format(as.hexmode(11), width=2)`

(Or, go to colorbrewer or a similar site and take the hex from there!)

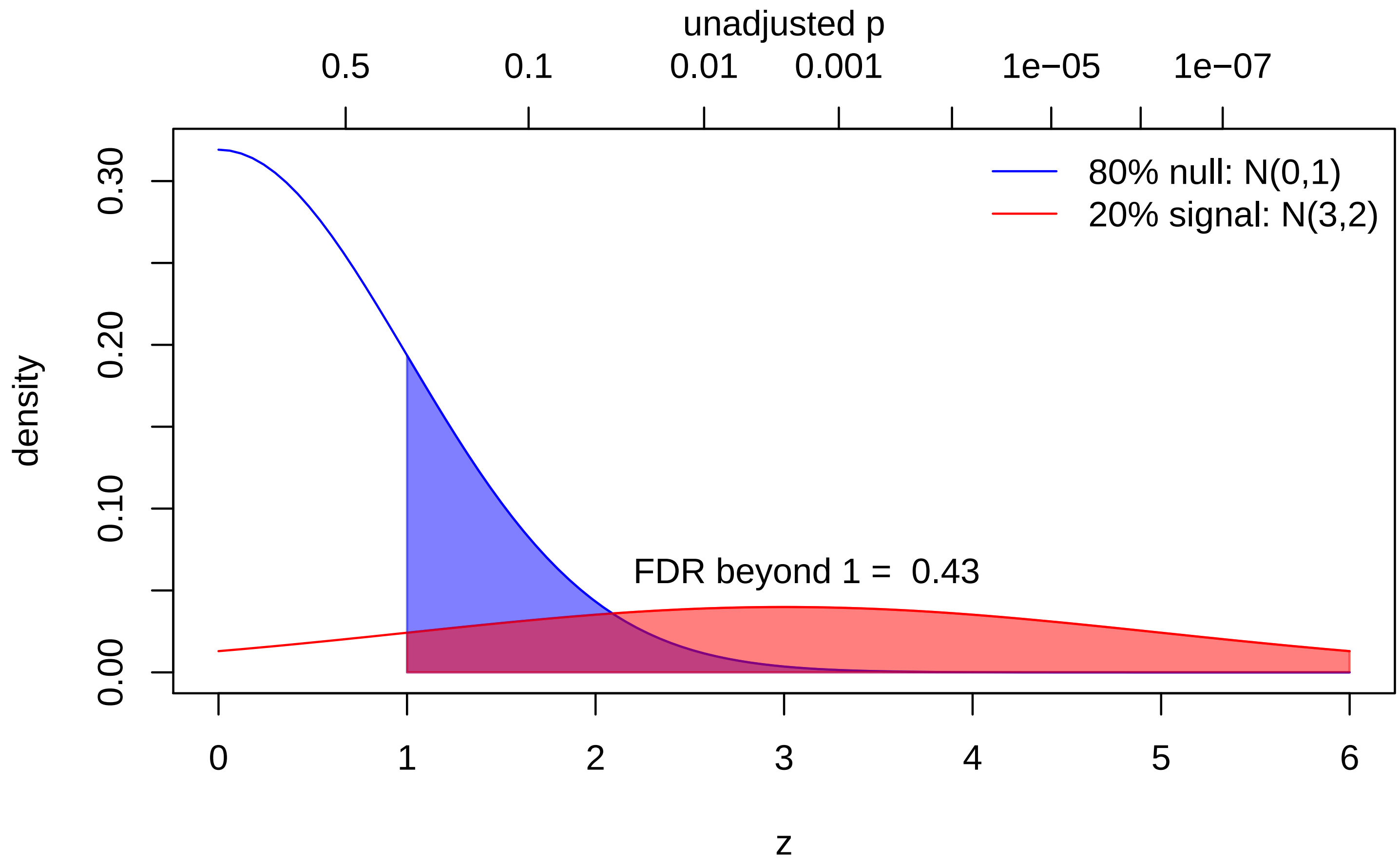
Transparency

An example; (also shows other graphics commands)

```
curve(0.8*dnorm(x), 0, 6, col="blue", ylab="density", xlab="z")
curve(0.2*dnorm(x,3,2), 0, 6, col="red", add=T)

xvals <- seq(1, 6, l=101)
polygon(
c(xvals,6,1), c(0.8*dnorm(xvals), 0,0),
density=NA, col="#0000FF80" ) # tranparent blue
polygon(
c(xvals,6,1), c(0.2*dnorm(xvals,3,2), 0,0),
density=NA, col="#FF000080" ) # tranparent red

legend("topright", bty="n", lty=1, col=c("blue","red"),
c("80% null: N(0,1)", "20% signal: N(3,2)"))
axis(3, at=qnorm(c(0.25, 0.5*10^(-1:-7)), lower=F), c(0.5, 10^(-1:-7)) )
mtext(side=3, line=2, "unadjusted p")

text(2.2, 0.07, adj=c(0,1), paste("FDR beyond 1 = ",
round(0.8*pnorm(1,lower=F)/(0.8*pnorm(1,lower=F) + 0.2*pnorm(1,3,2,lower=F)),3)))
```

Transparency

Here's the output;



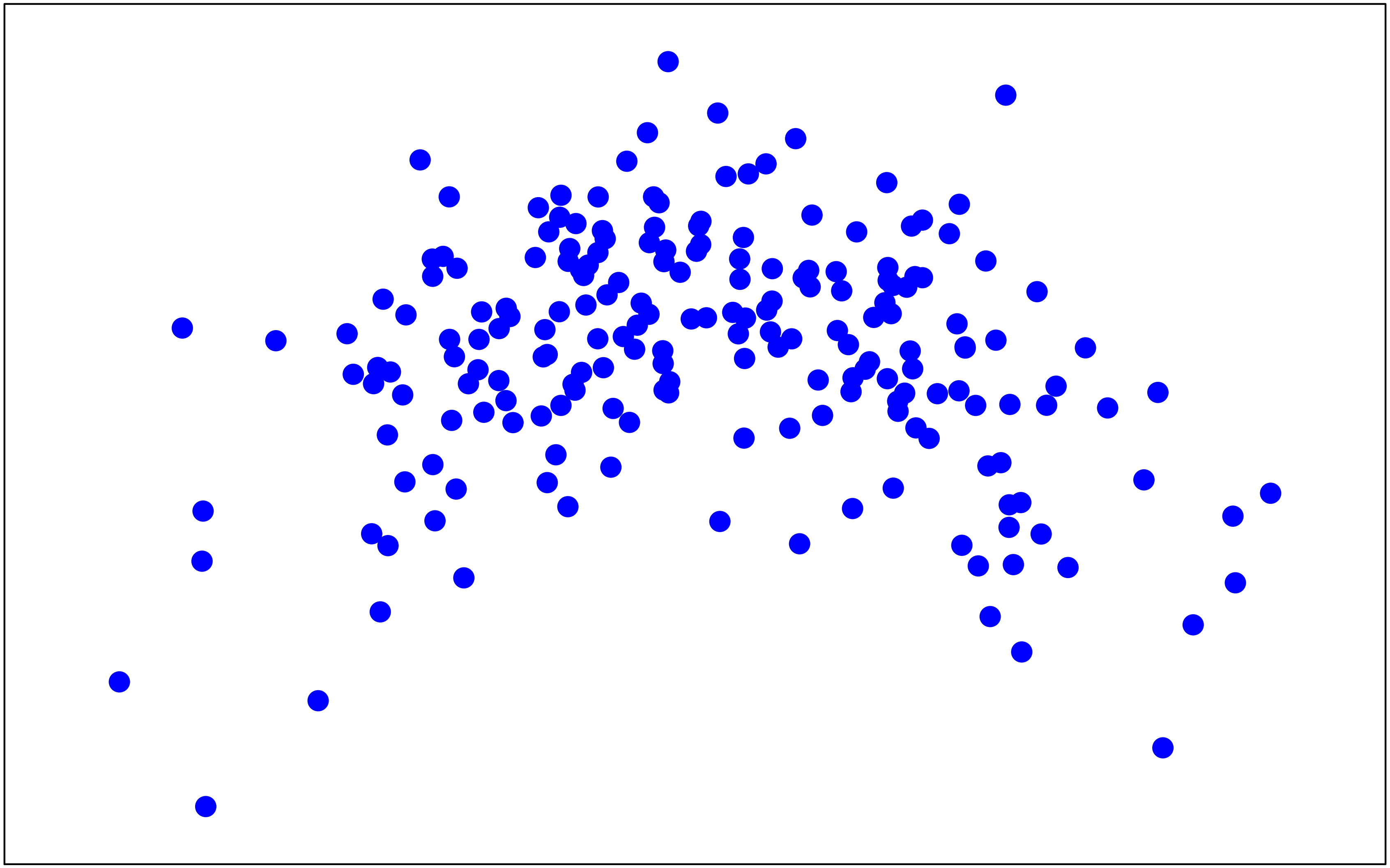
Hexagonal binning

Using transparent plotting symbols is a quick-and-dirty way to adapt scatterplots for use with large datasets.

A better method is ‘hexagonal binning’; this is a 2D analog of a histogram – where you would count the number of data in one area, and then draw a bar with height proportional to that count.

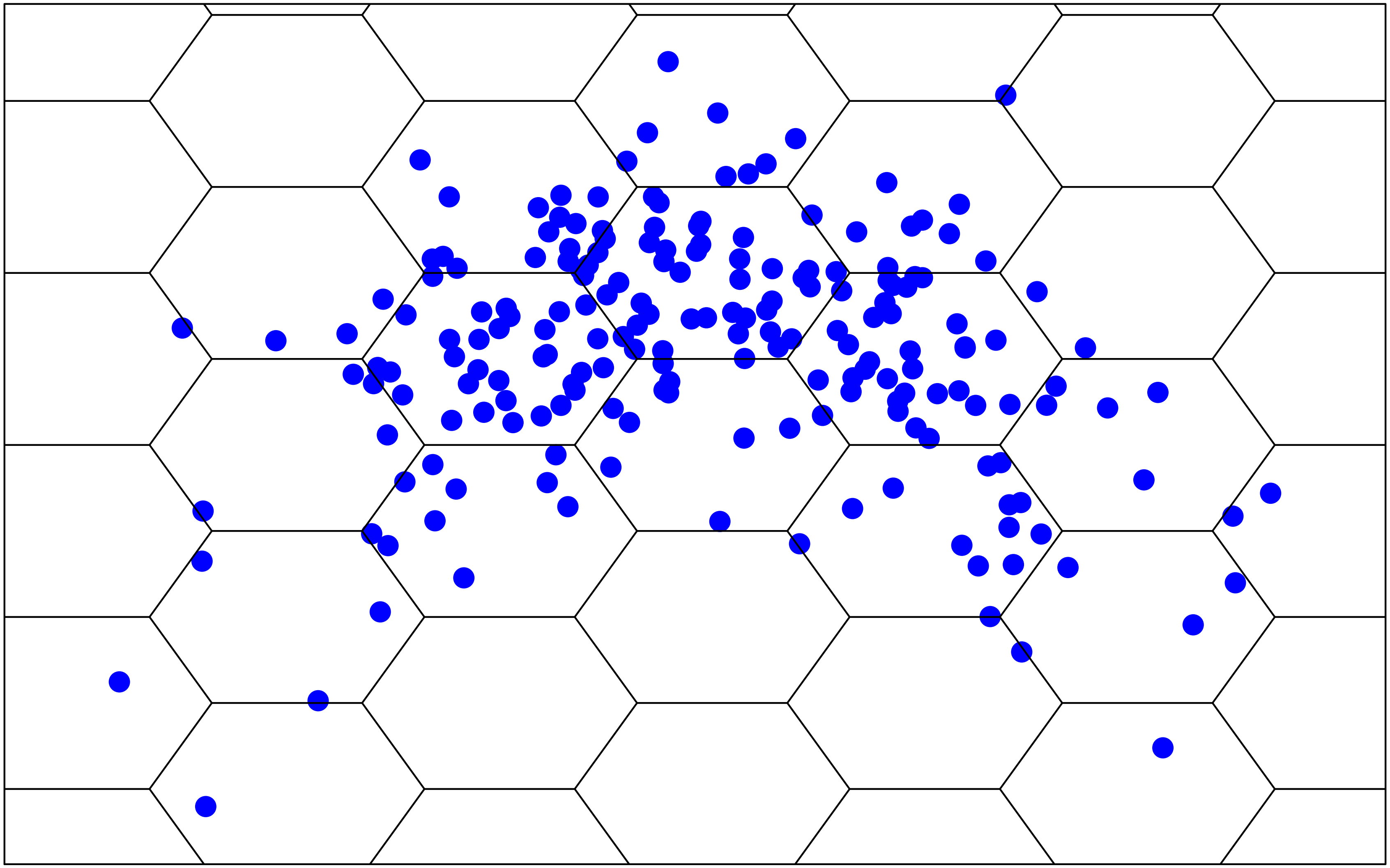
Hexagonal binning

Binning in two dimensions;



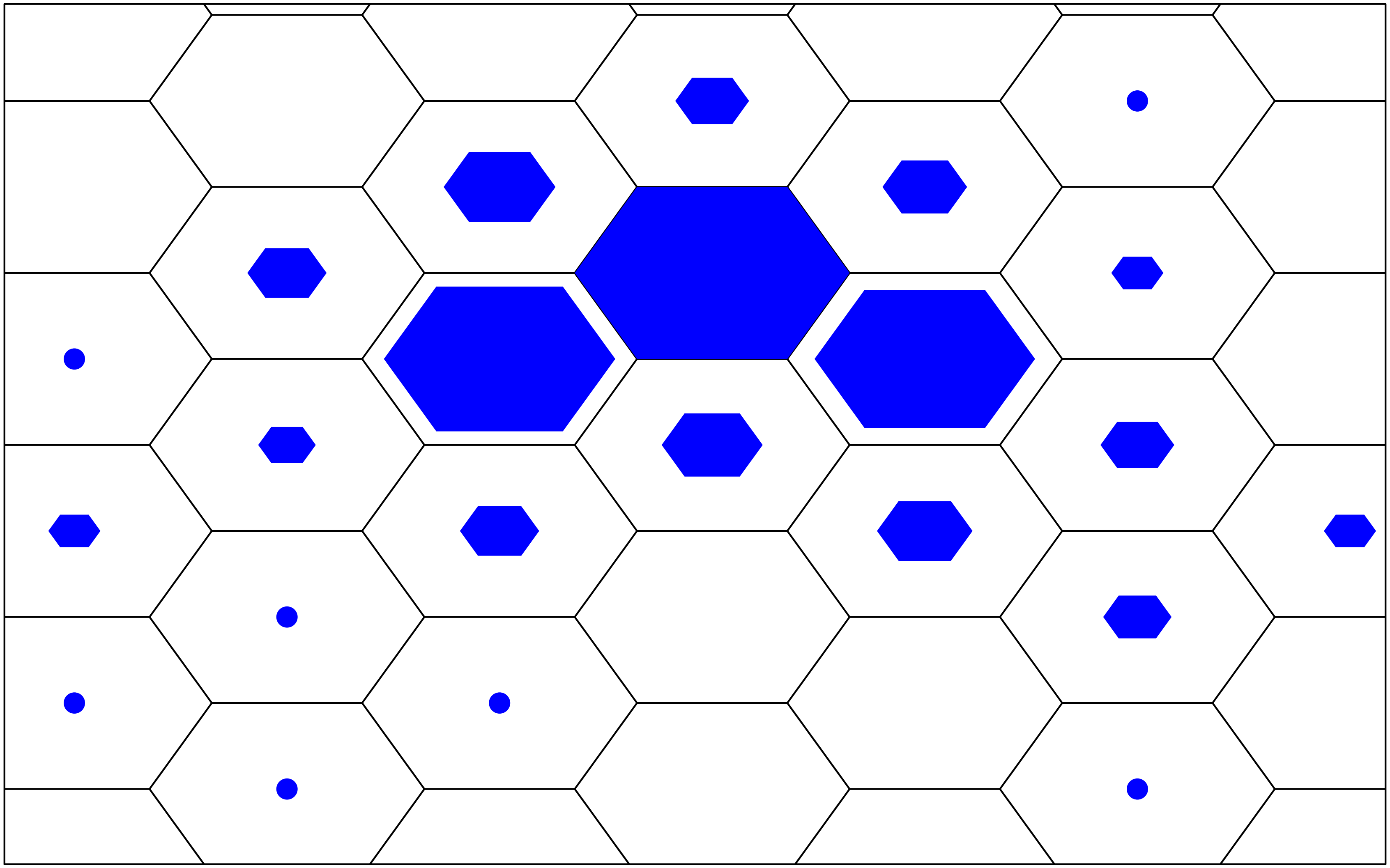
Hexagonal binning

Binning in two dimensions;



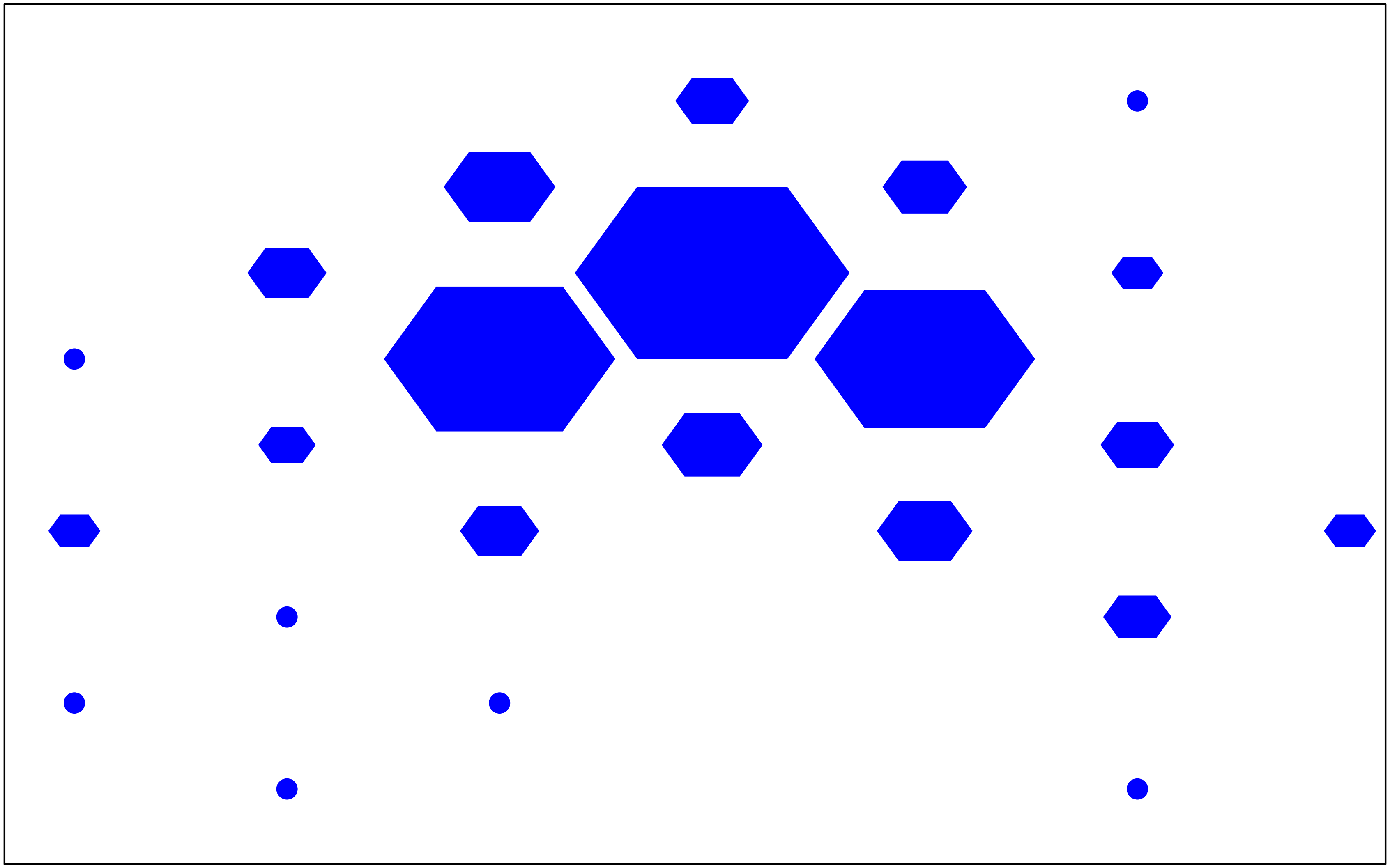
Hexagonal binning

Binning in two dimensions;



Hexagonal binning

Binning in two dimensions;

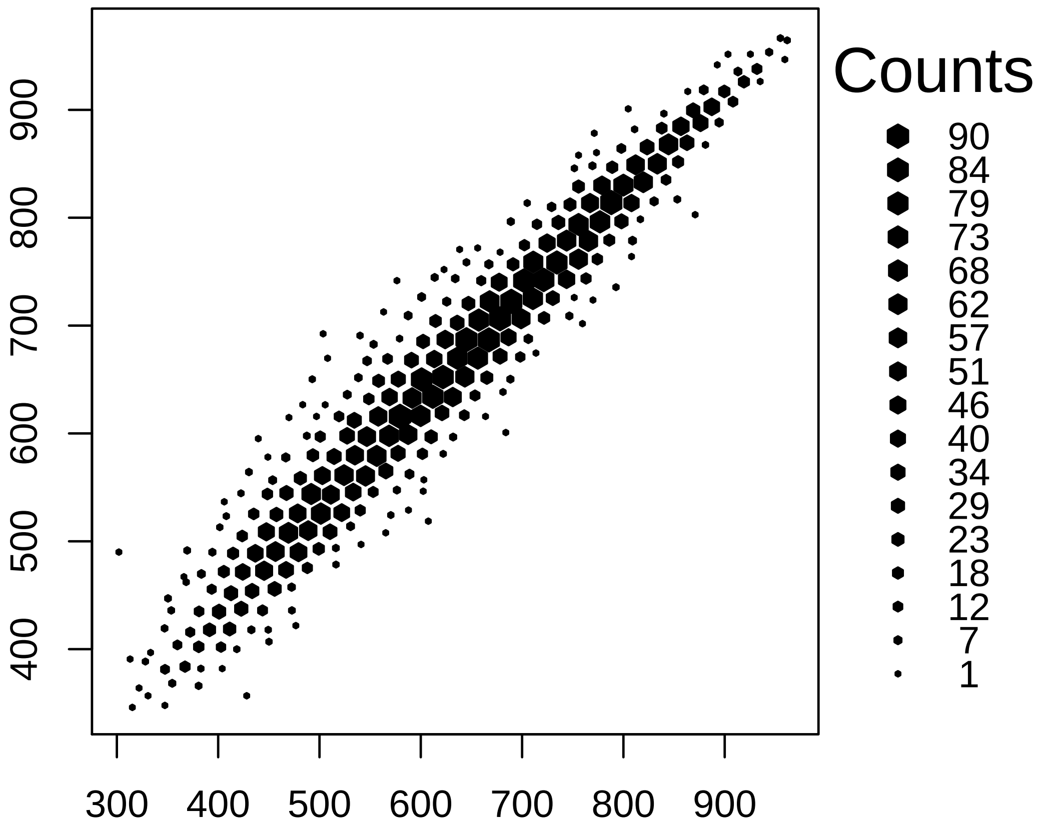


Hexagonal binning

The `hexbin()` package does all the bin construction, and counting. It has a `plot` method for its `hexbin` objects;

```
install.packages(c("hexbin","survey"))
library("hexbin")
library("survey")# for apipop data frame

with(apipop, plot(hexbin(api99,api00), style="centroids"))
```

Hexagonal binning



Hexagonal binning

Hexbin is used when you don't *really* care about the exact location of every single point

- Singleton points are plotted 'as usual'; you do (perhaps) care about them

- **`hexbin`** centers the 'ink' at the cell data's 'center of gravity'

- `style="centroids"` gives the center-of-gravity version; the default `style` is `colorscale` – usually grayscale. See `?gplot.hexagons` for more options

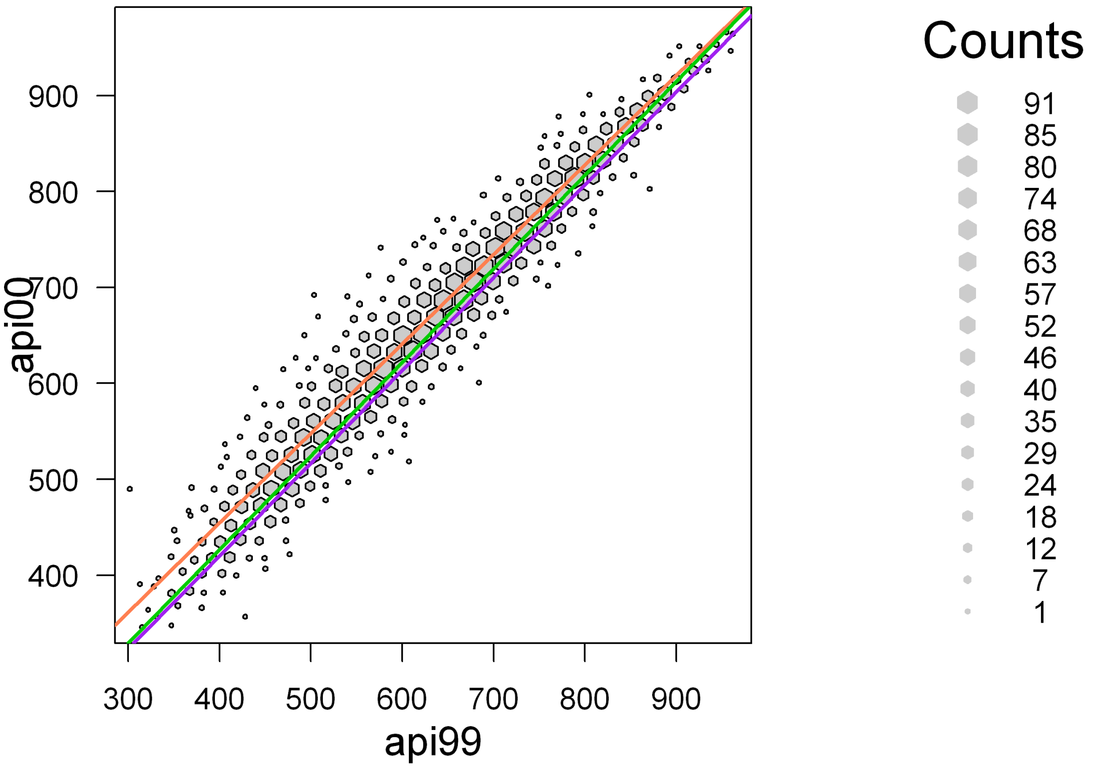
Hexagonal binning

For keen people: the `hexbin` package doesn't use the standard R graphics plotting devices; instead, it operates through the `Grid` system (in the `grid` package) which defines rectangular regions on a graphics device; these `viewport` regions can have a number of coordinate systems. To add lines to a hexbin plot, the options are;

- Use `hexVP.abline()` to add these directly
- Move everything into 'standard' graphics – not `Grid` graphics (see `?Grid`). The `Grid` system lets you alter graphics *after* plotting them
- Write your own plot method for hexbin objects, with standard R graphics commands
- Make do with `hexBinning()` in the `fMultivar` package

Hexagonal binning

An example; color-coded lines of best fit, by school type;



```
lm.e <- coef(lm(api00~api99, data=apipop, subset=stype=="E"))
lm.m <- coef(lm(api00~api99, data=apipop, subset=stype=="M"))
lm.h <- coef(lm(api00~api99, data=apipop, subset=stype=="H"))

hexVP.abline(vp1$plot.vp, lm.e[1], lm.e[2], col="coral")
```

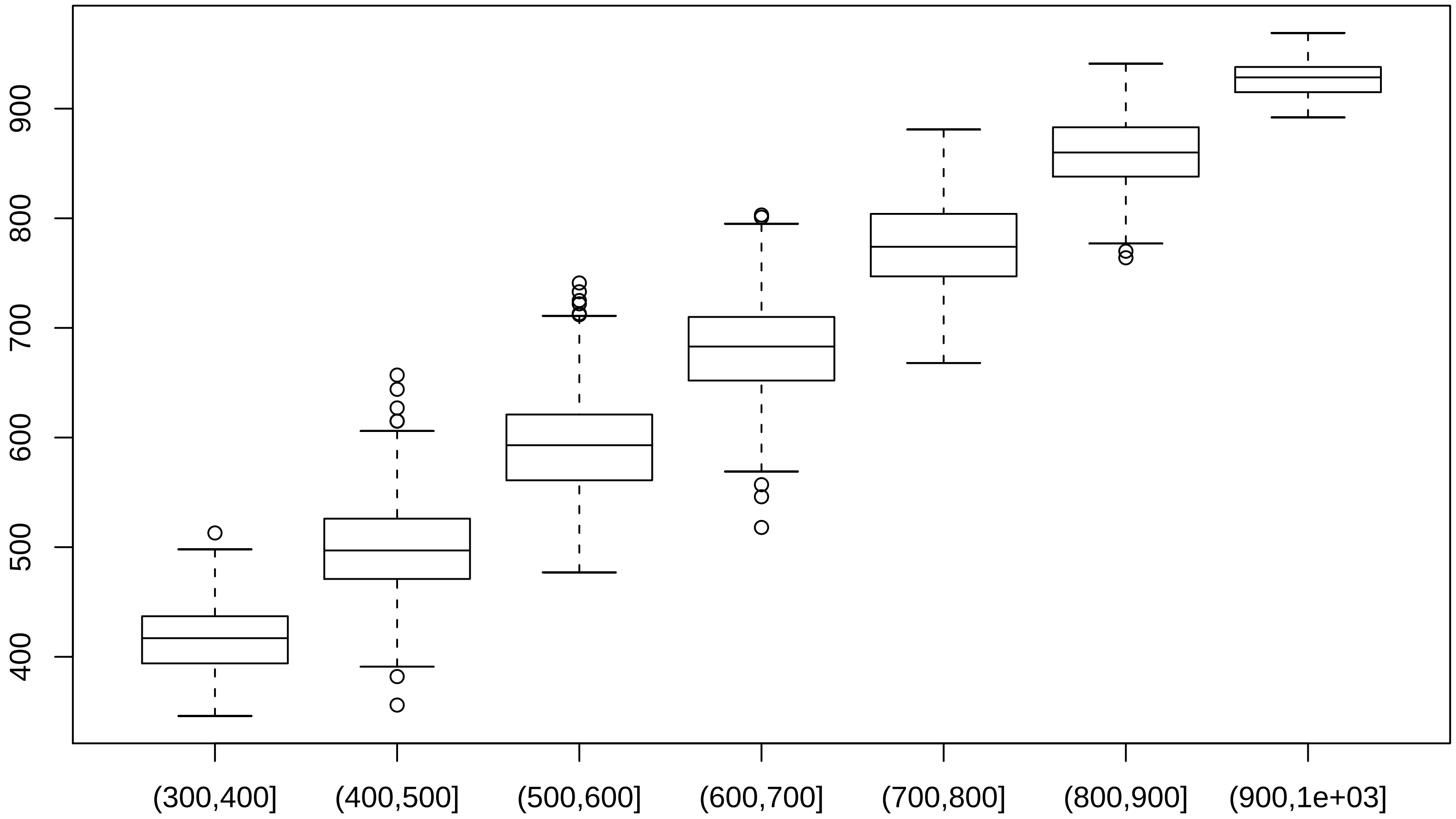
Large data: multiple groups

For showing multiple groups, a scatterplot smoother or perhaps boxplots or conditioning may be better.

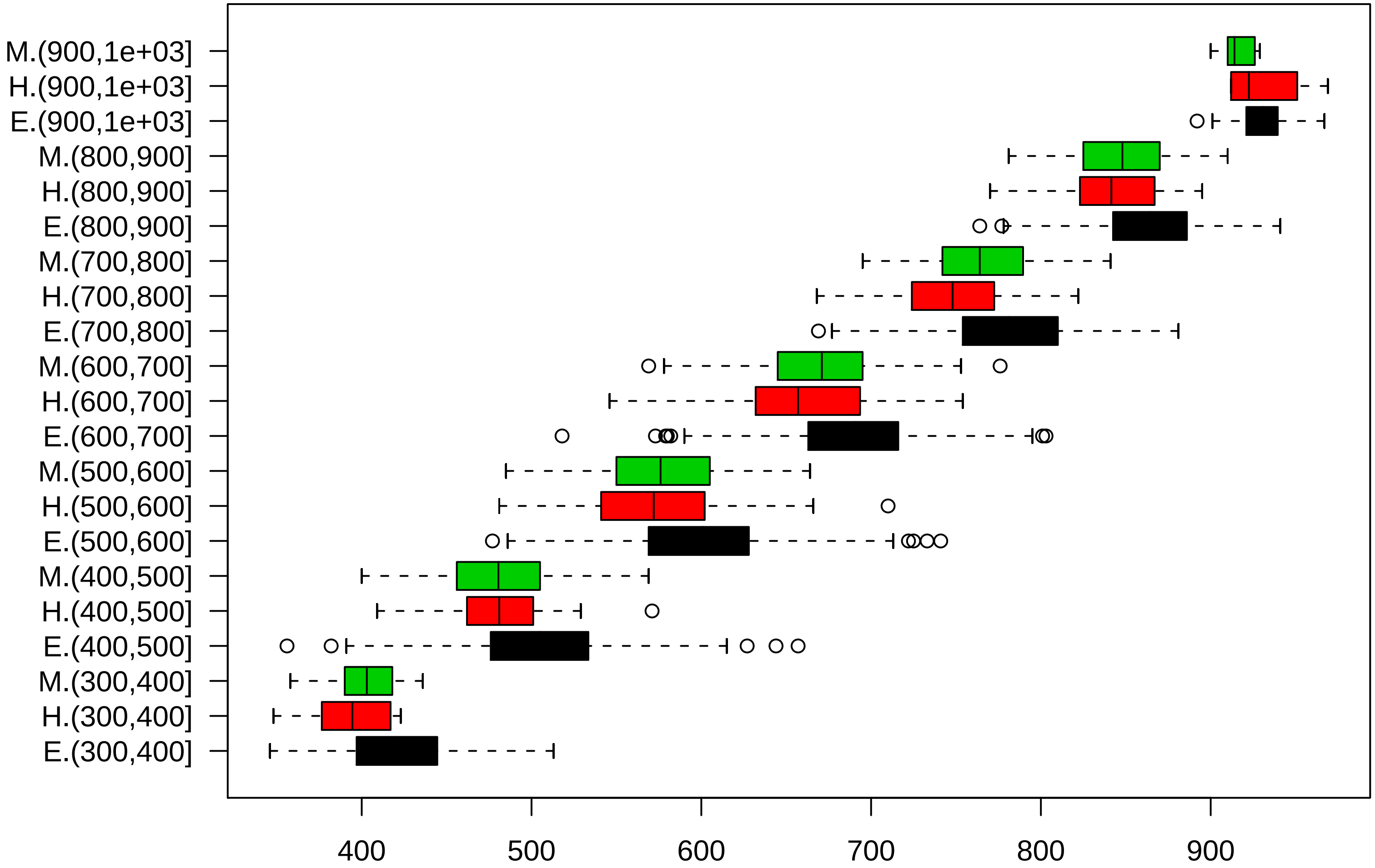
```
boxplot(api00~cut(api99,(3:10)*100), data=apipop)
par(las=1)
par(mar=c(5.1,10.1,2.1,2.1))
boxplot(api00~interaction(stype,
                           cut(api99,(3:10)*100)),
          data=apipop, horizontal=TRUE,col=1:3)
```

- `cut()` turns a variable into a factor by cutting it at the specified points.
- `par(mar=)` sets the margins around the plot. We need a large left margin for the labels.

Large data: multiple groups



Large data: multiple groups



Smoothers

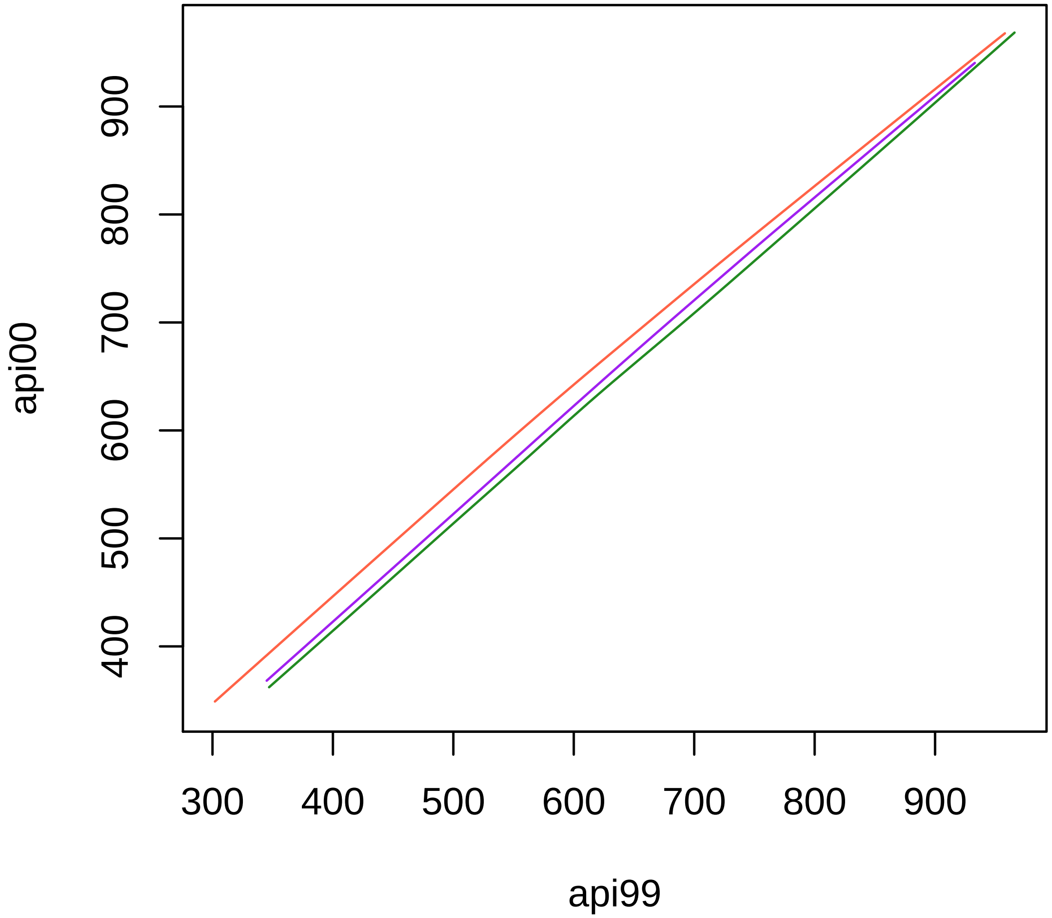
We don't plot the data at all, just means (could also plot quantiles with `quantreg` package)

```
plot(api00~api99,data=apipop,type="n")
with(subset(apipop, stype=="E"),
     lines(lowess(api99, api00), col="tomato"))
with(subset(apipop, stype=="H"),
     lines(lowess(api99, api00), col="forestgreen"))
with(subset(apipop, stype=="M"),
     lines(lowess(api99, api00), col="purple"))
```

Note the use of `type="n"`

`subset()` returns a subset of a data frame.

Smoothers



Conditioning

```
hexbinplot(api00~api99|stype,data=apipop,style="centroid")
```