Module 6: Advanced R for Statgen Software Package Development

Ken Rice
Thomas Lumley

Universities of Washington and Auckland

Seattle, July 2017

Introduction: Course Aims

- Programming with R
  - Efficient coding
  - Code that other people can use

- Using R for sophisticated analyses
  - Some useful tools for large-scale problems
  - Making R play nicely with others
  - Knowing where to look when you need more

Introduction: About Prof Lumley

- Prof, University of Auckland
- R Core developer
- Genetic/Genomic research in Cardiovascular Epidemiology
- Sings bass (sometimes)

Introduction: About Prof Rice

- Associate Prof, UW Biostat
- Not an authoR, but a useR (and a teacheR)
- Genetic/Genomic research in Cardiovascular Epidemiology
- Sings bass (in Seattle!)

... and you?

(who are you, what area of genomics, what are you looking for from the course)

Introduction: Course structure

10 sessions over 2.5 days

- Day 0.5; Graphics, Programming in R
- Day 1; Objects, Packages, The Web
- Day 2; C code, large datasets

Download everything from here;

`http://faculty.washington.edu/kenrice/sisg-adv`

Introduction: Session structure

We will alternate teaching (questions welcome) and hands-on exercises (questions and discussions welcome!)

For some topics, within a single 90 minute session;

- 45 mins teaching (Questions welcome! Please interrupt!)
- 30 mins hands-on
- 15 mins summary, discussion

For other topics, we'll separate sessions (90 mins) and hands-on exercises (90 mins)

1. Tools for Graphics

Ken Rice
Thomas Lumley

Universities of Washington and Auckland

Seattle, July 2015

Important pre-takeoff announcement:

We are assuming you know;

- How to use `R` from the command line, and how to write and use script files (and spot e.g. missing commas and }'s)
- How to manipulate basic data structures in `R`; in particular vectors and data frames
- How to write functions
- How to look up help files
- How to make some simple plots e.g. making a scatterplot with `plot()`, adding to existing plots using `points()`, `lines()`, `text()`, and `legend()`

Of course, familiarity with (non-advanced) statistical & genetic concepts will also help.

A thousand words?

Graphics are vital to presenting complex data

R's defaults are not bad, but are from a long time ago

- Facts about graphics
- Tools (to build on) for graphics

Plotting large & high-dimensional data

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

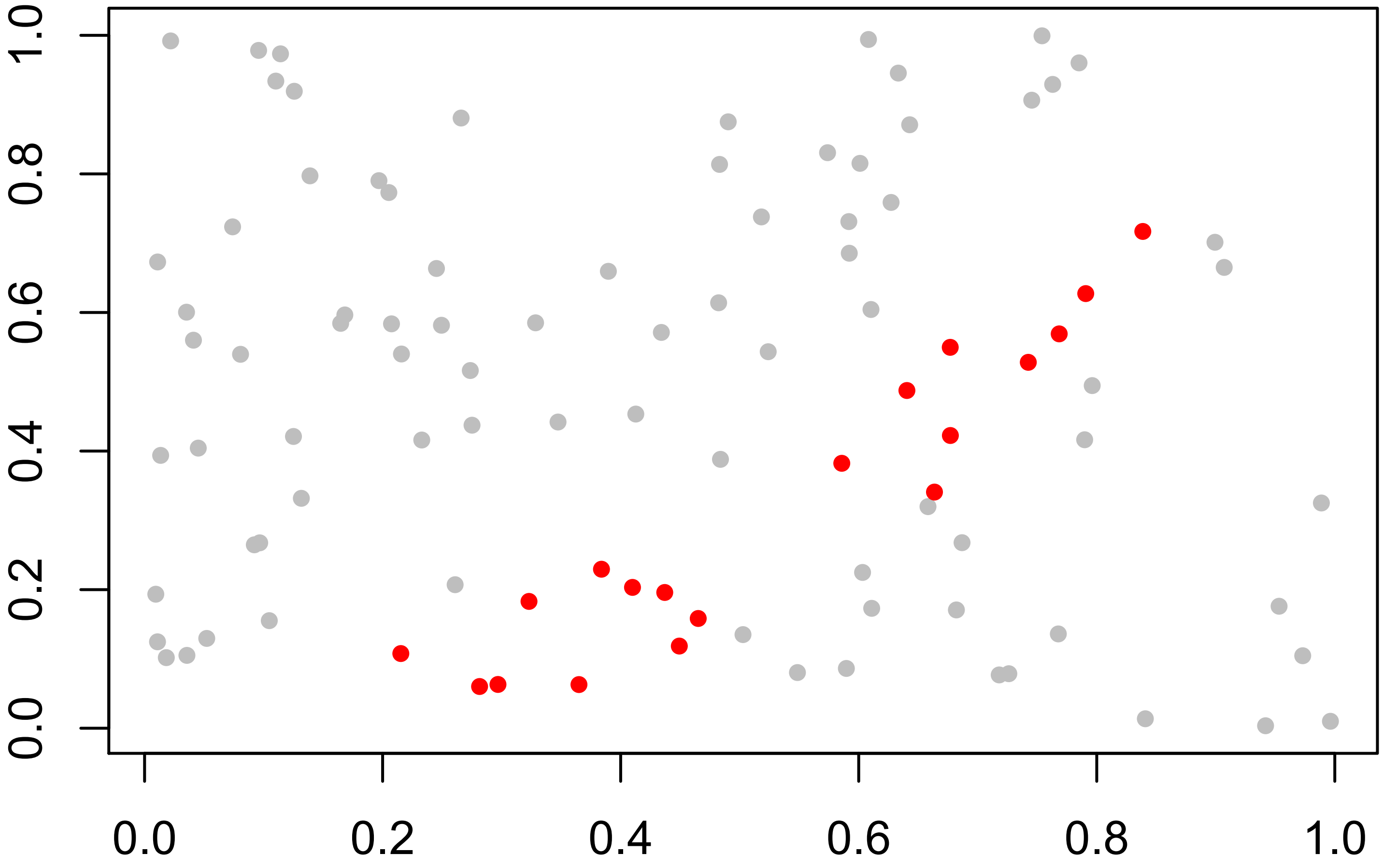
For higher-dimensions, some traditional approaches are;

- 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;

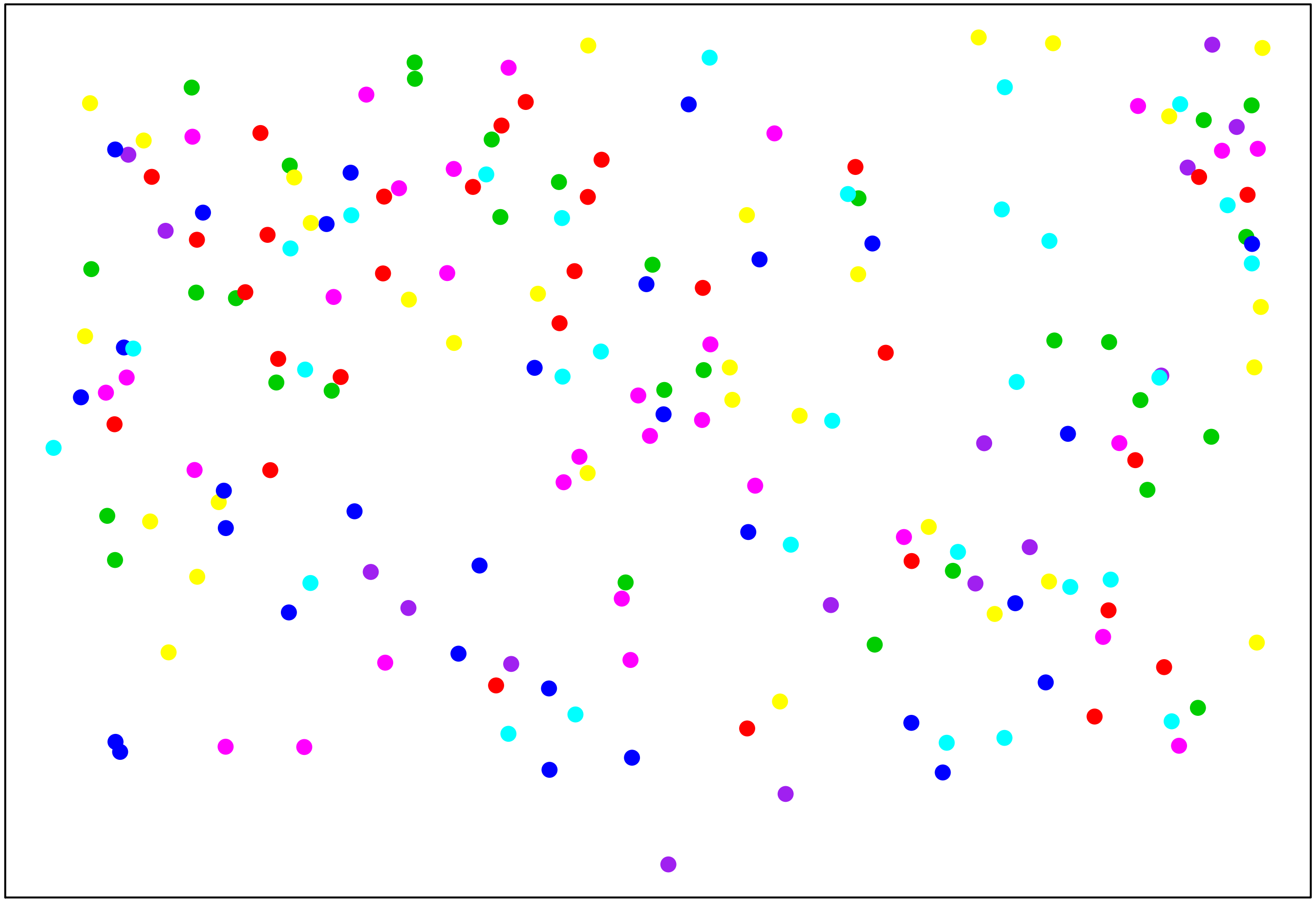
Plotting large & high-dimensional data

Some of these points are not like the others...



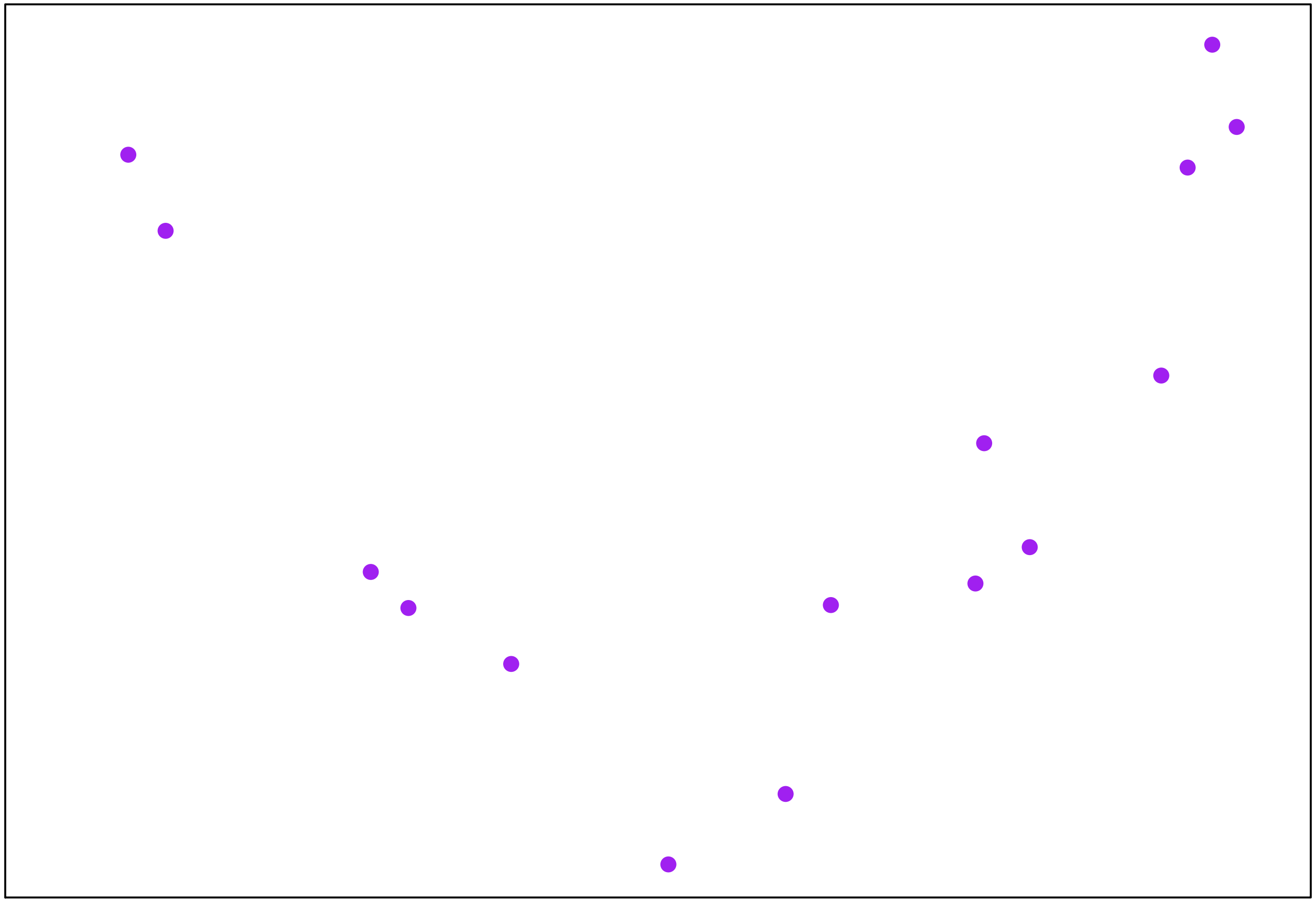
Plotting large & high-dimensional data

Some of these points are not like the others...



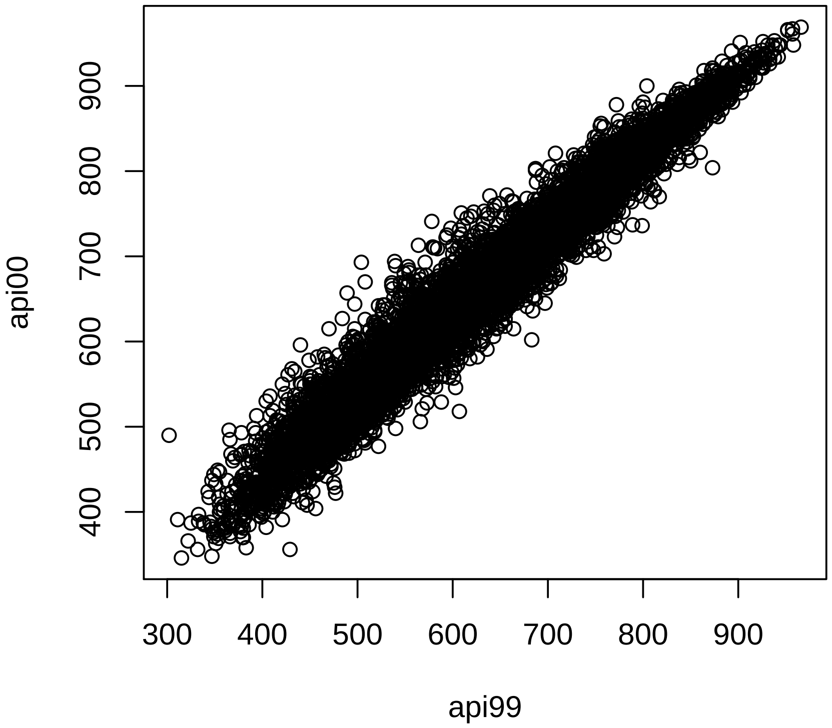
Plotting large & high-dimensional data

Some of these points are not like the others...



Plotting large & high-dimensional data

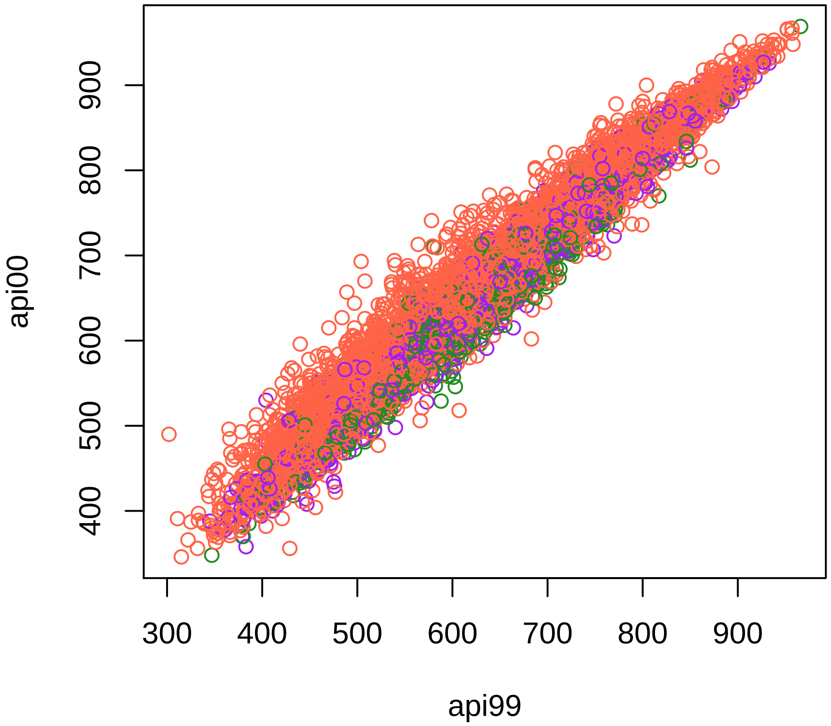
For large(ish) data, ‘overlap’ is a fundamental problem...



(California Academic Performance Index on 6194 schools)

Plotting large & high-dimensional data

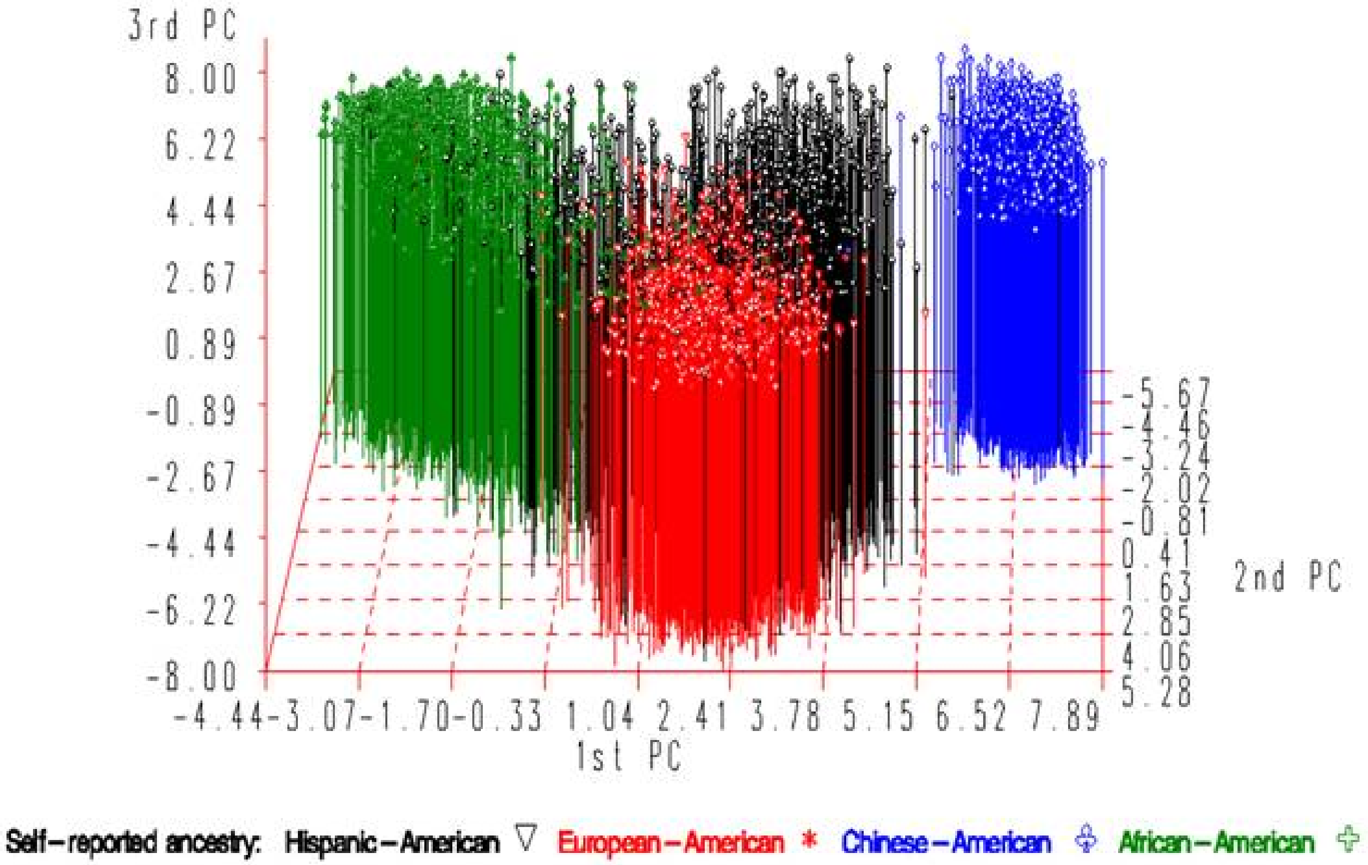
... which remains, when we color-code.



Colors denote Elementary, Middle & High Schools

Plotting large & high-dimensional data

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



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

Conditioning plots

A typical goal for measuring Z is to see whether the $Y - X$ relationship changes at different values of Z. For example, we might want to see if a Blood Pressure/genotype association varies by Body Mass Index (weight/height^2)

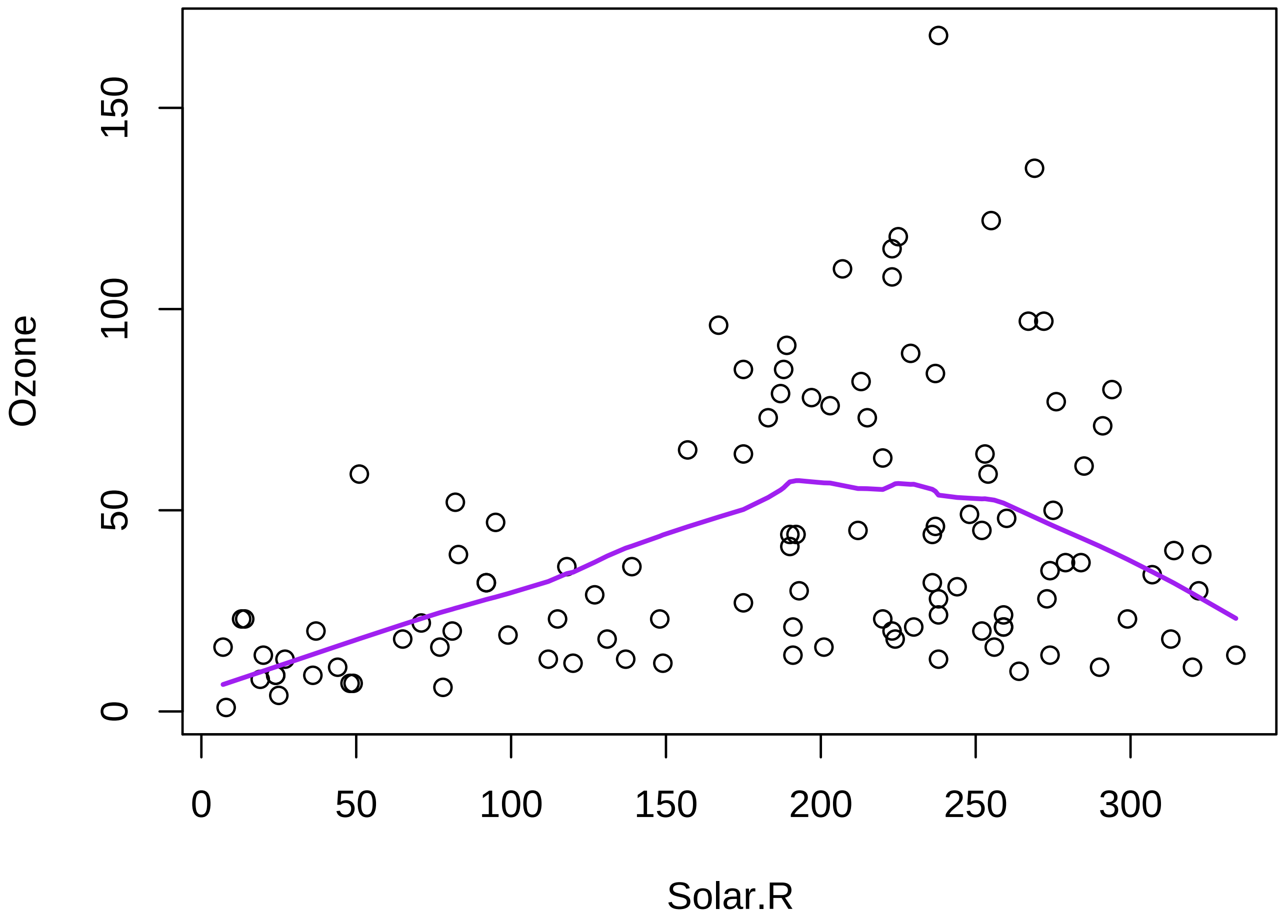
In this case, it's useful to show plots of Y against X conditioned on the value of Z, i.e. Y versus X for all data with Z in a small range. This is known as a conditioning plot, and can be produced with `coplot()`.

Conditioning plots

Ozone is a secondary pollutant, it is produced from organic compounds and atmostpheric oxygen in reactions catalyzed by nitrogen oxides and powered by sunlight.

However, looking at ozone concentrations in NY in summer (Y) we see a non-monotone relationship with sunlight (X)

Conditioning plots

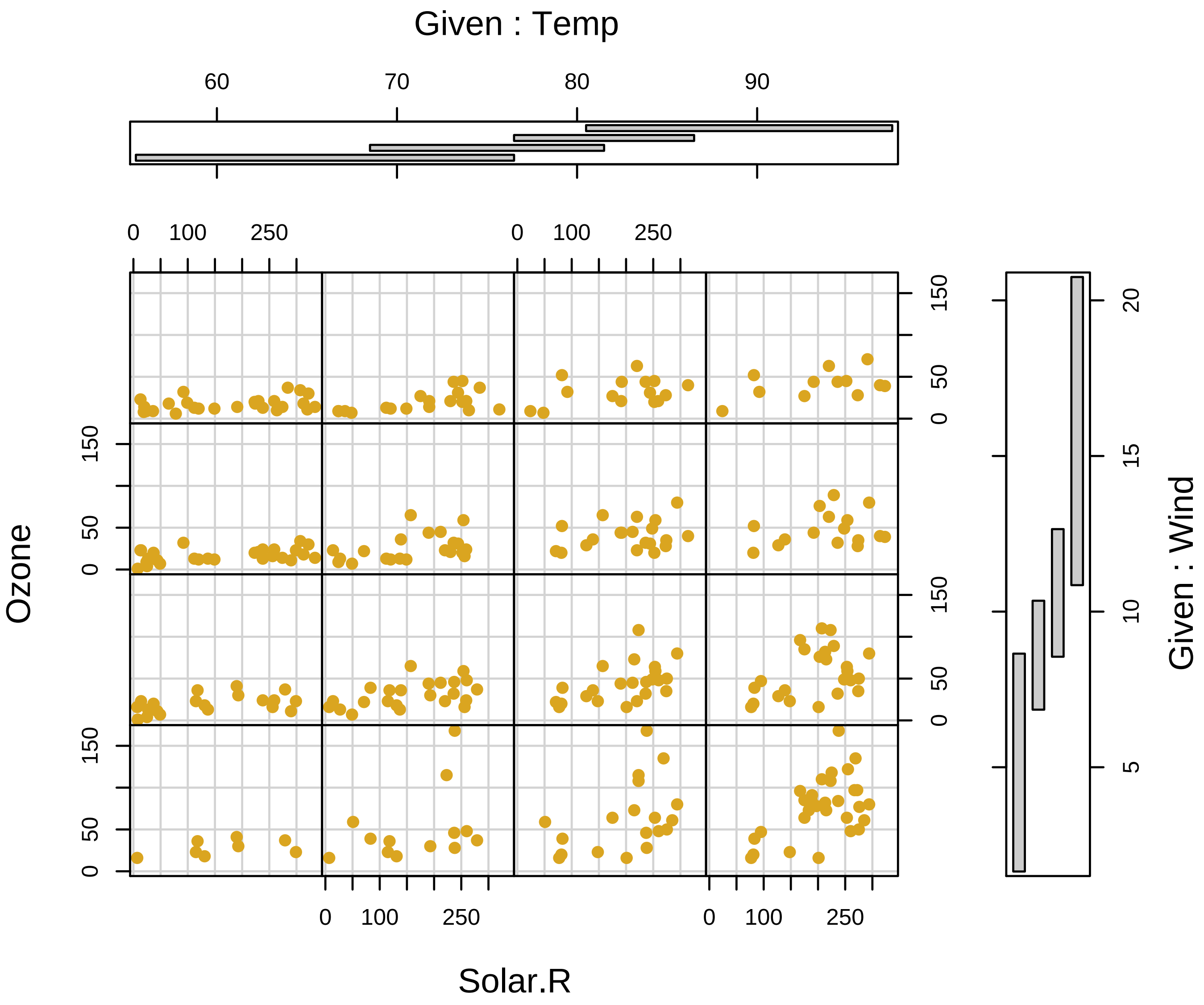


Conditioning plots

Here we draw a scatterplot of `Ozone` vs `Solar.R` for various subranges of `Temp` and `Wind`. For more examples like this, see the commands in the `lattice` package.

```
data(airquality)
coplot(Ozone ~ Solar.R | Temp * Wind, number = c(4, 4),
     data = airquality,
    pch = 21, col = "goldenrod", bg = "goldenrod")
```

Conditioning plots

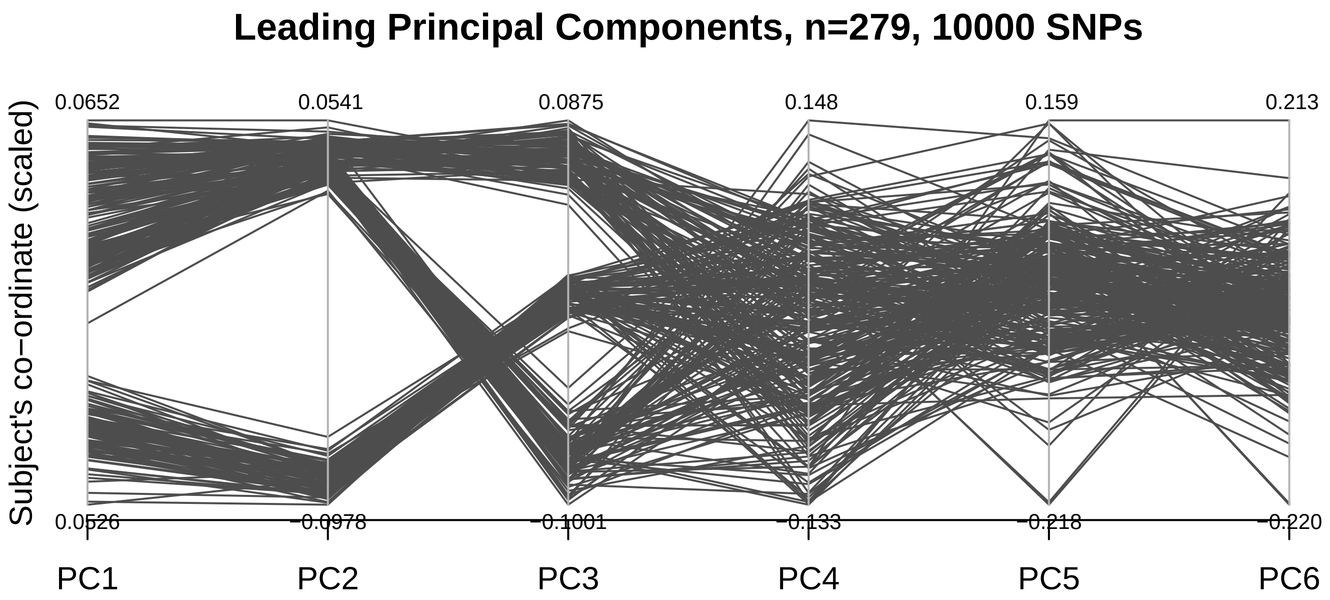


Conditioning plots

- A 4-D relationship is illustrated; the Ozone/sunlight relationship changes in strength depending on both the Temperature and Wind
- The vertical bar | is statistician-speak for ‘conditioning on’ (nb this is different to use of |’s meaning as Boolean ‘OR’)
- The horizontal/vertical ‘shingles’ tell you which data appear in which plot. The overlap can be set to zero, if preferred
- `coplot()`’s default layout is a bit odd; try setting `rows`, `columns` to different values
- For more plotting commands that support conditioning, see `library(help="lattice")`

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;

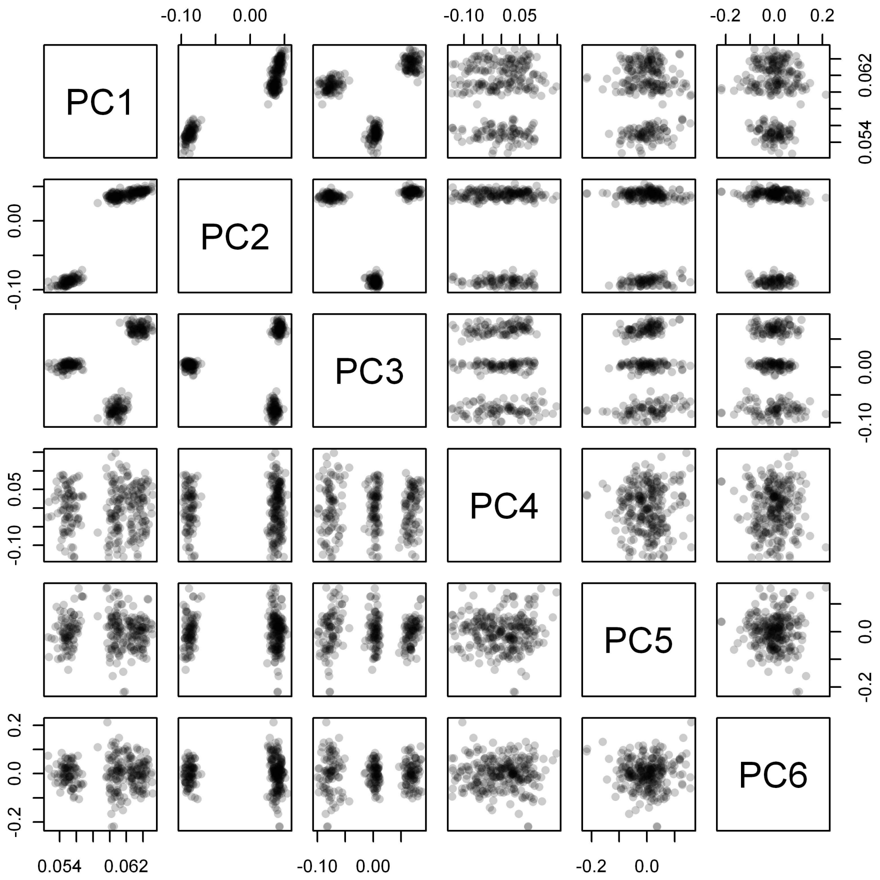


Parallel Coordinate Plots

- 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, ...

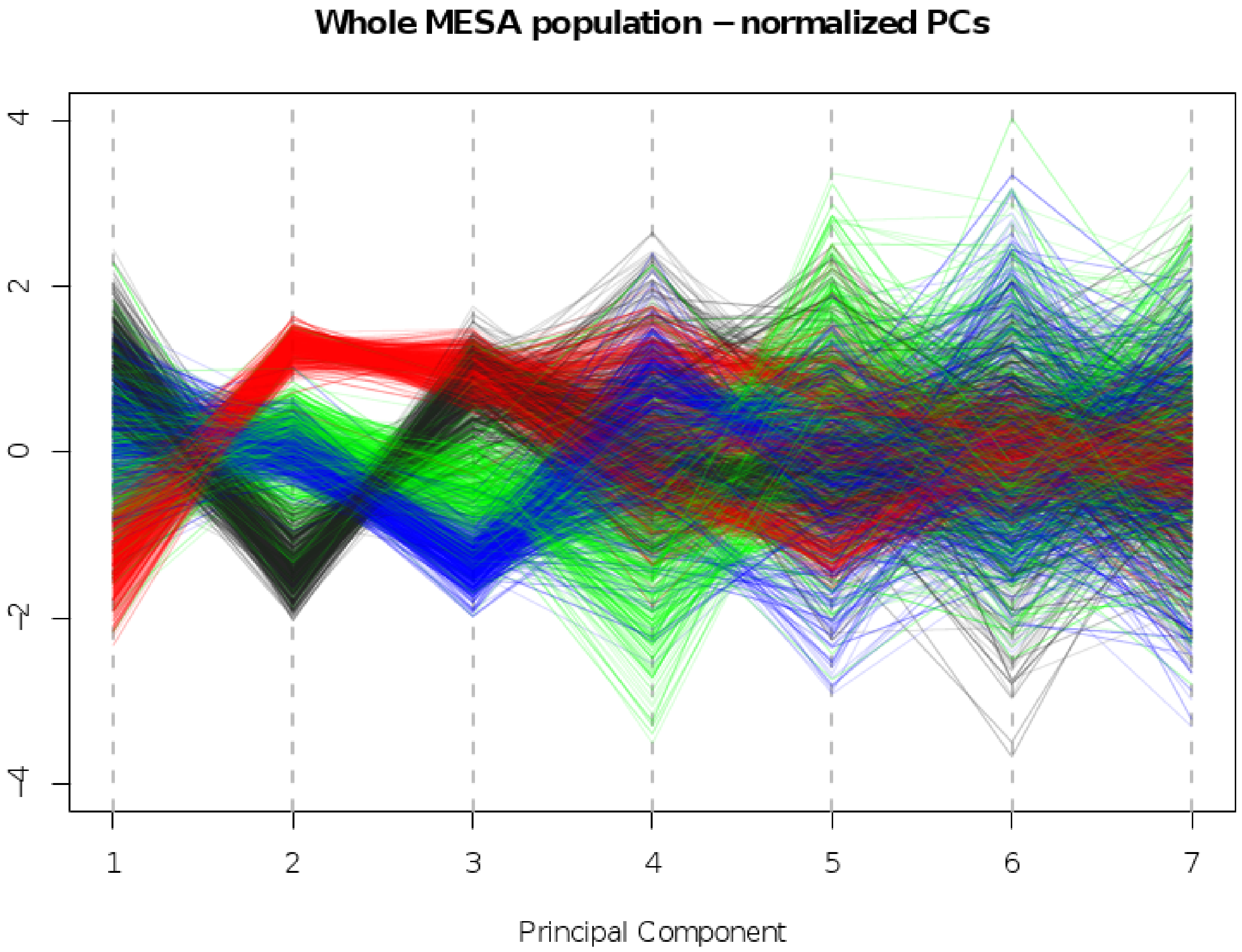
Parallel Coordinate Plots

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)`

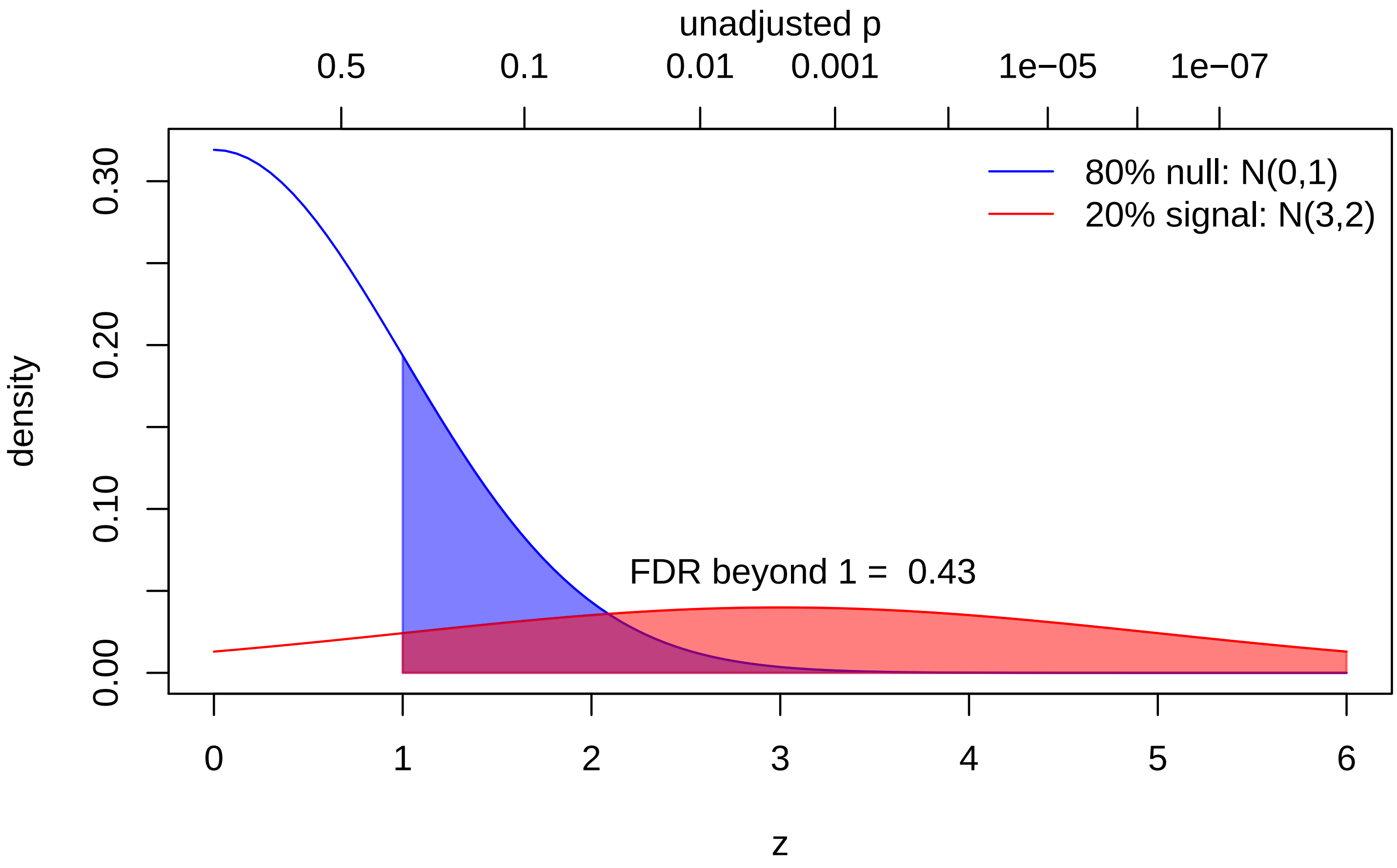
Transparency

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;



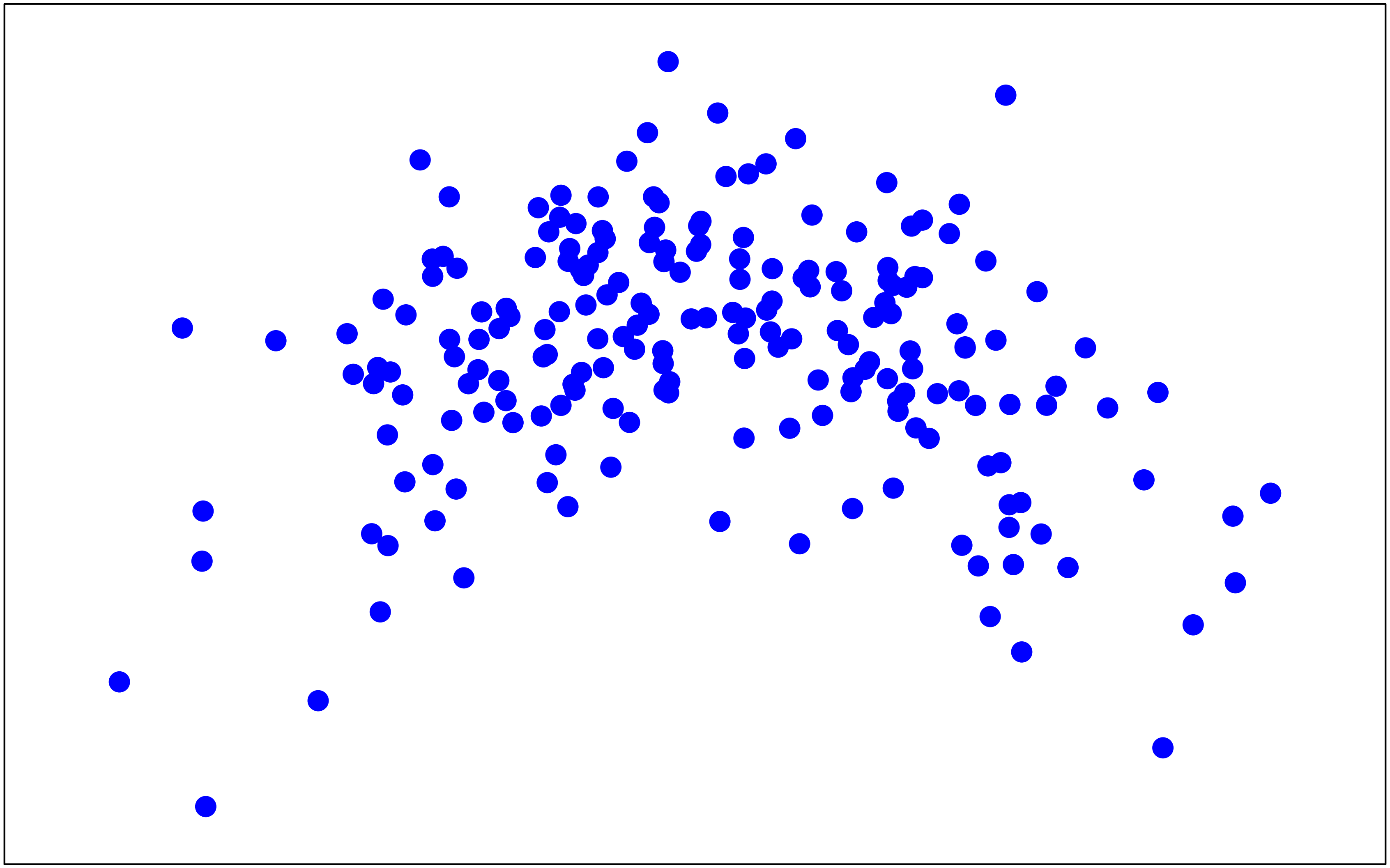
Hexagonal binning

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.

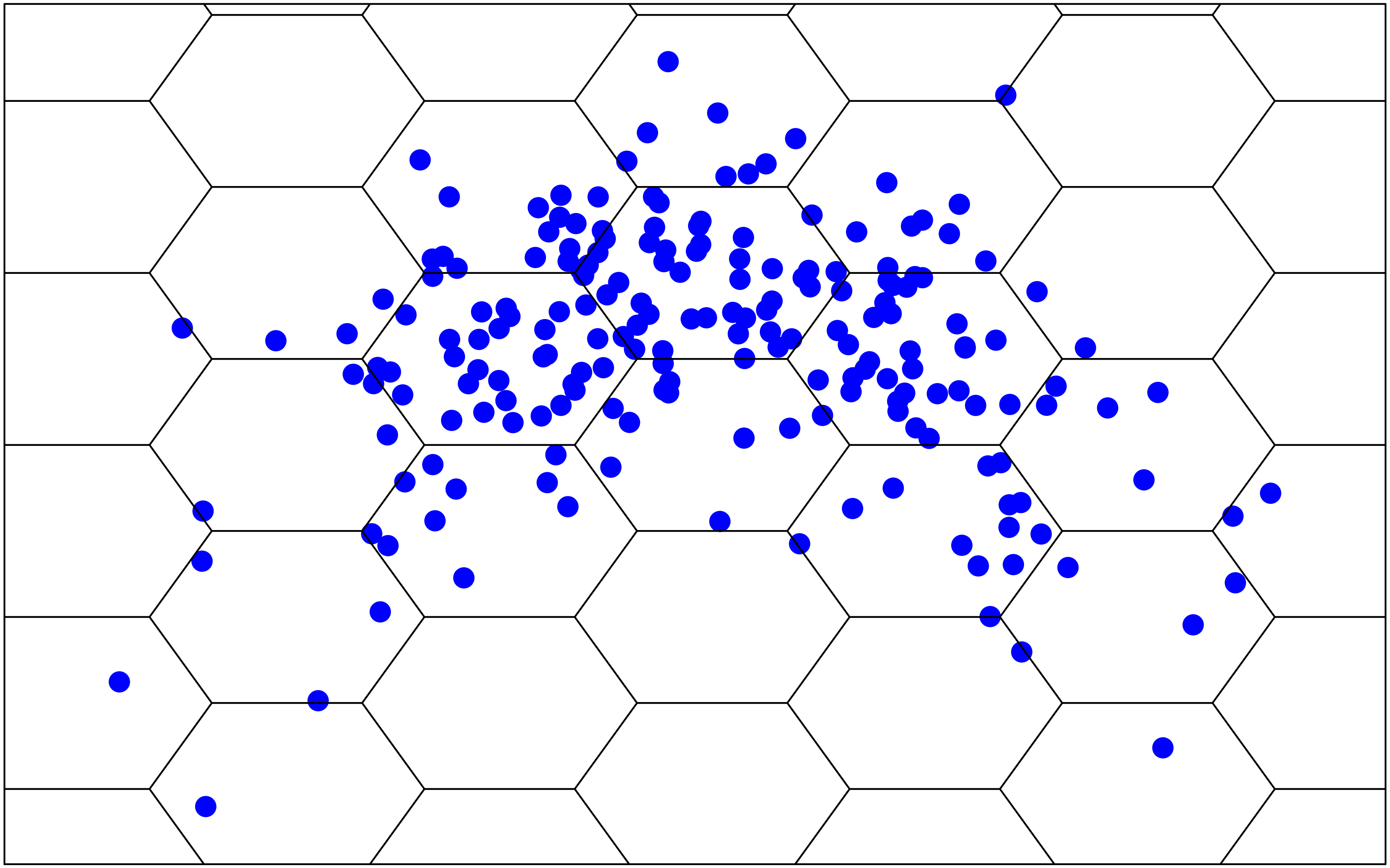
Hexagonal binning

Binning in two dimensions;



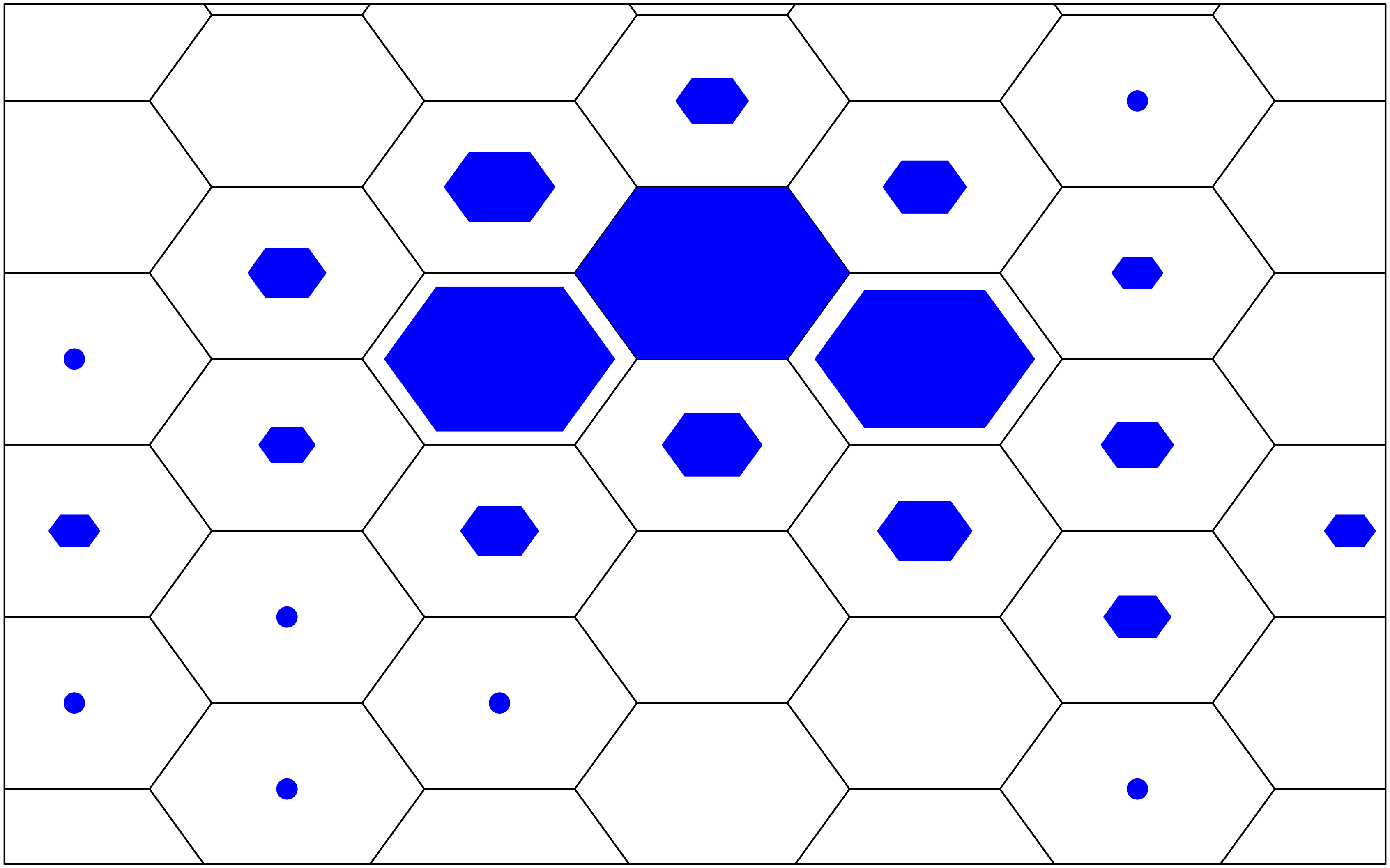
Hexagonal binning

Binning in two dimensions;



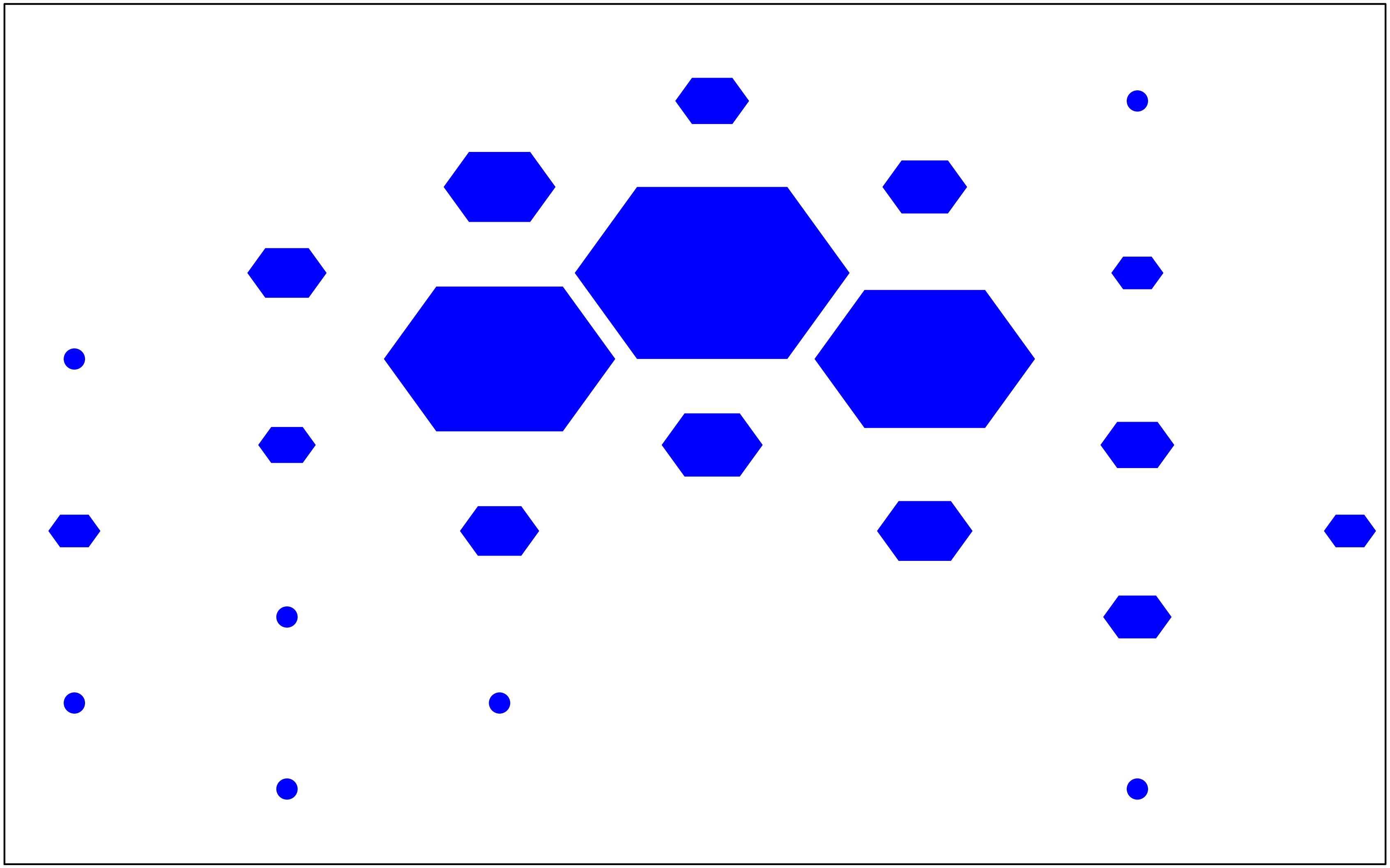
Hexagonal binning

Binning in two dimensions;



Hexagonal binning

Binning in two dimensions;

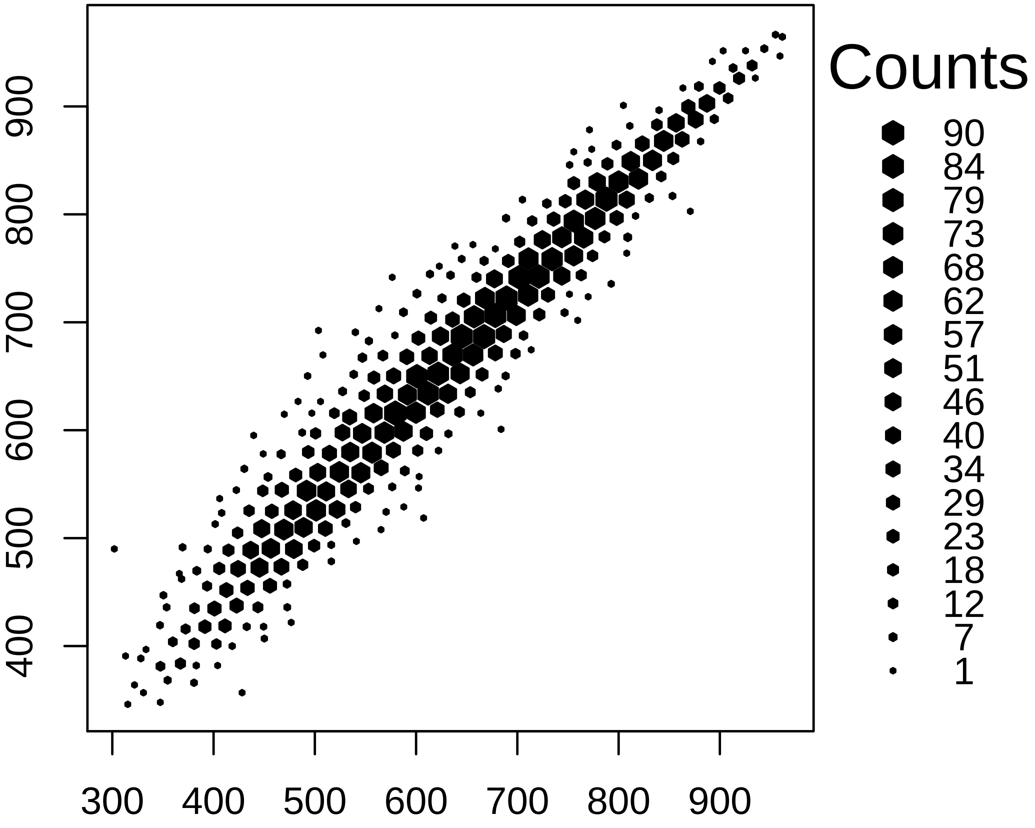


Hexagonal binning

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

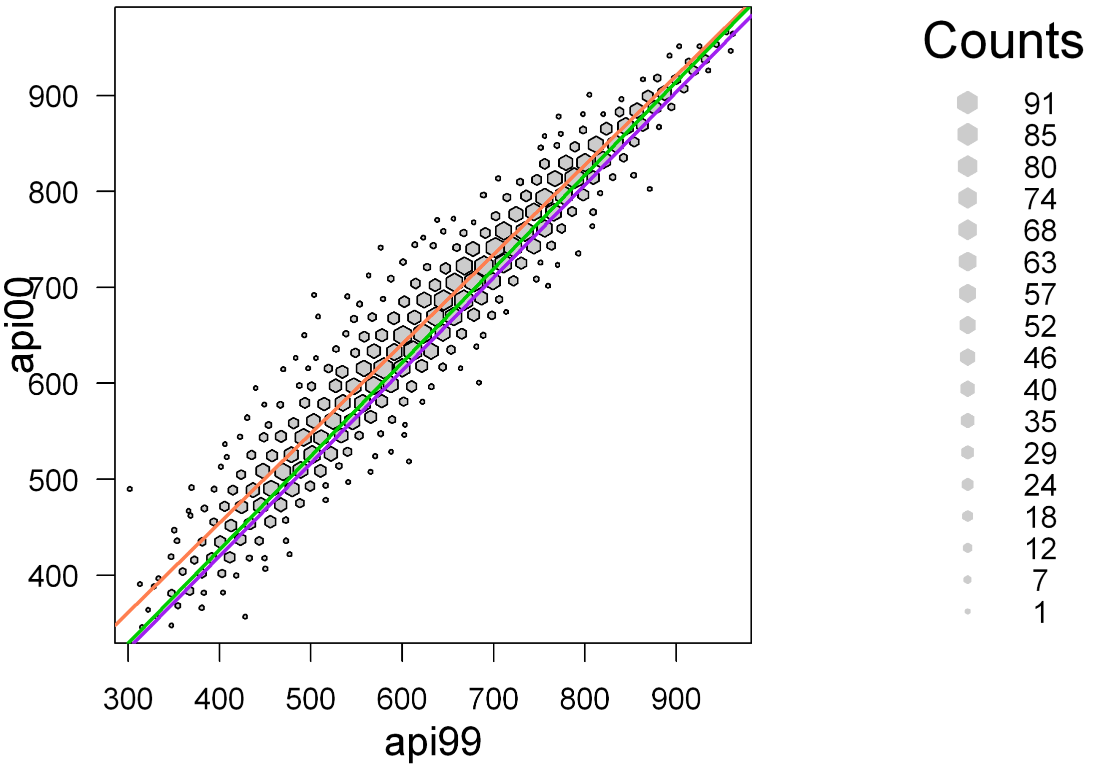
Hexagonal binning

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
- Use the hextri package, which also allows for multiple colours.

Hexagonal binning

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

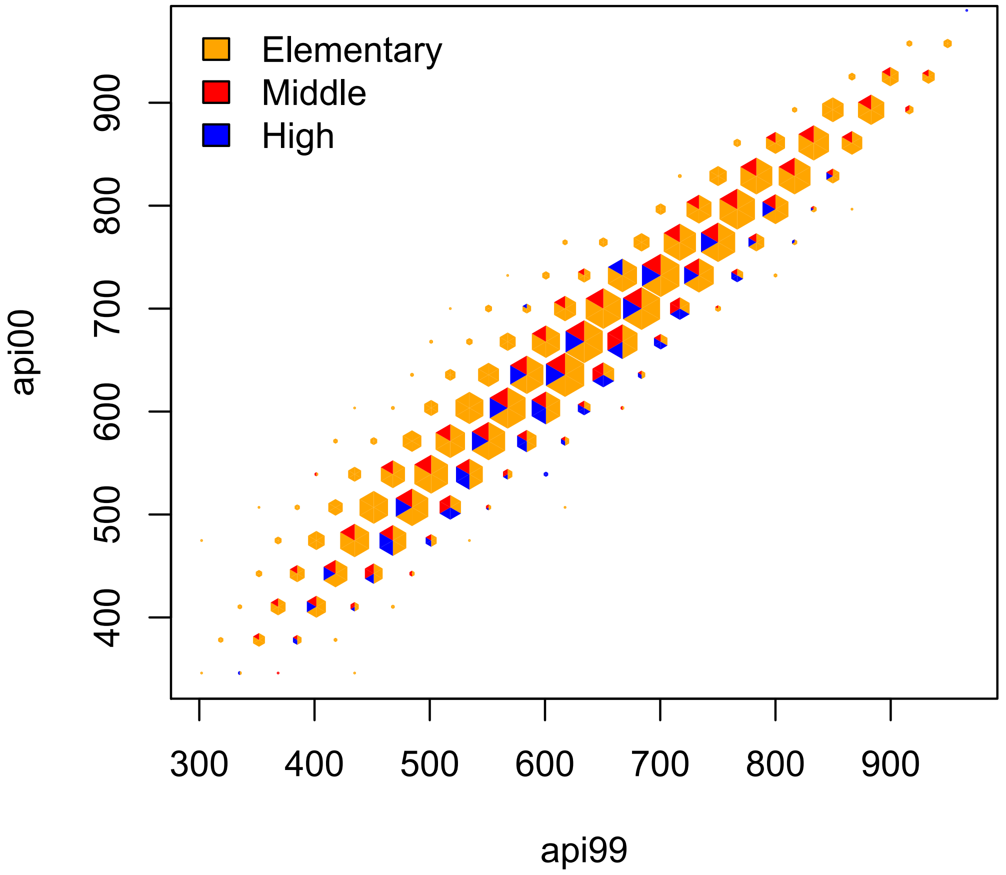


```
lm.e <- coef(lm(api00~api99, data=apipop, subset=stype=="E"))
lm.m <- coef(lm(api00~api99, data=apipop, subset=stype=="M"))
lm.h <- coef(lm(api00~api99, data=apipop, subset=stype=="H"))

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

Hexagonal binning

```
library(hextri)
hextri(api00~api99, data=apipop, class=stype,
   col=c("orange","blue","red"),nbins=20,style="size")
legend("topleft",bty="n",fill=c("orange","red","blue"),
   legend=c("Elementary","Middle","High"))
```

Hexagonal binning



File formats

Ultimately, we want to output the graph in an appropriate file format. (Cut-and-paste is possible, but not recommended)

R knows more about font sizes and spacing than most users – so first design the graph at the size it will end up, eg:

```
## on Windows
windows(height=4,width=6)
## on Unix
x11(height=4,width=6)
## on Mac
quartz(height=4, width=6
```

... and, when that's done, write a version to a file

File formats

For example, for a 6×4 PDF file;

```
pdf("myprettypic.pdf", height=4, width=6) # inches
... plotting commands here ...
dev.off()                    # close the file
```

Some other formats: (see `?Devices` for a full list)

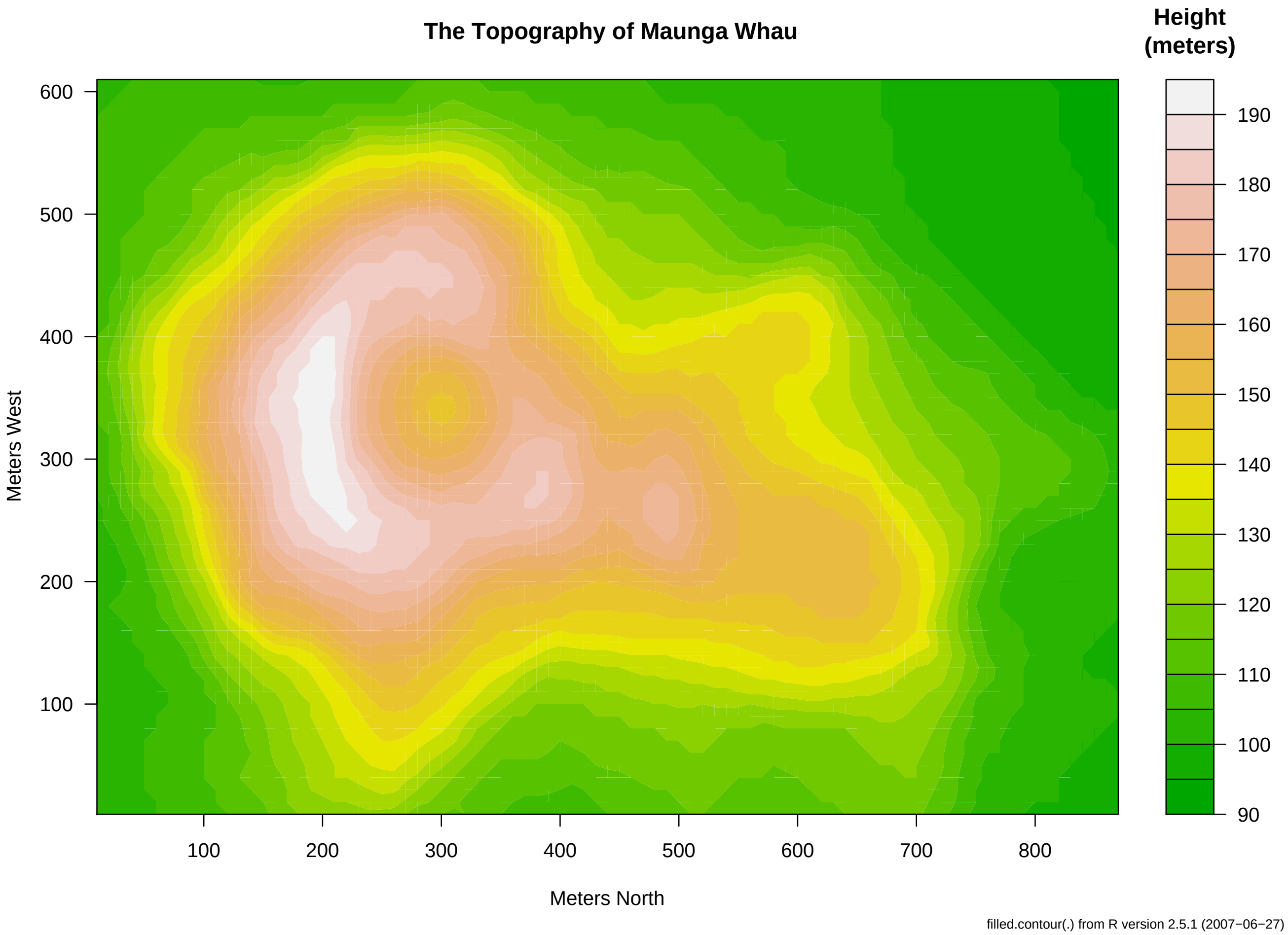
- `jpeg("mypic.jpg", w=6*288, h=4*288, res=288)` – lossy
- `png("mypic.png", w=6*288, h=4*288, res=288)` – lossless

– point size of text can also be manipulated, which can be useful when making posters

PowerPoint, or Word, or LaTeX can all rescale graphs. But when the graph gets smaller, so do the axis labels...

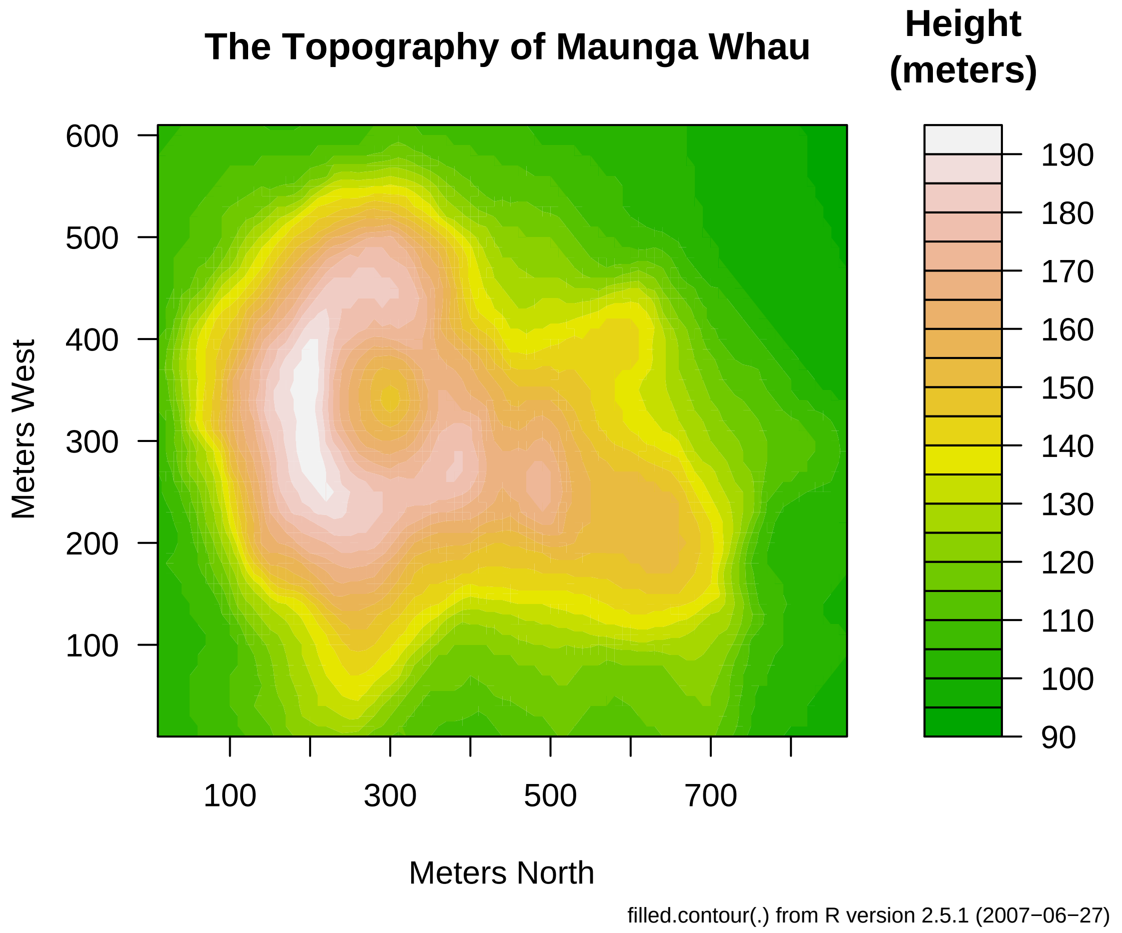
File formats

Created at full-page size (11×8.5 inches)



File formats

Created at 6×5 inches



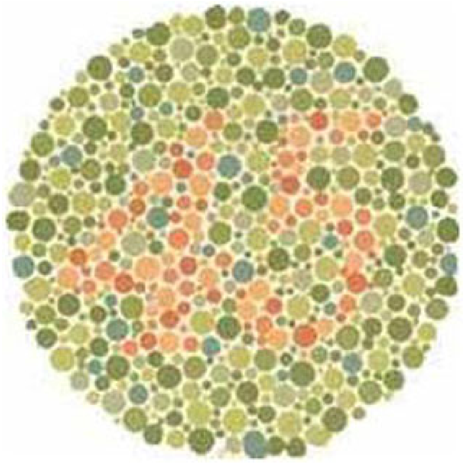
Color schemes

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`

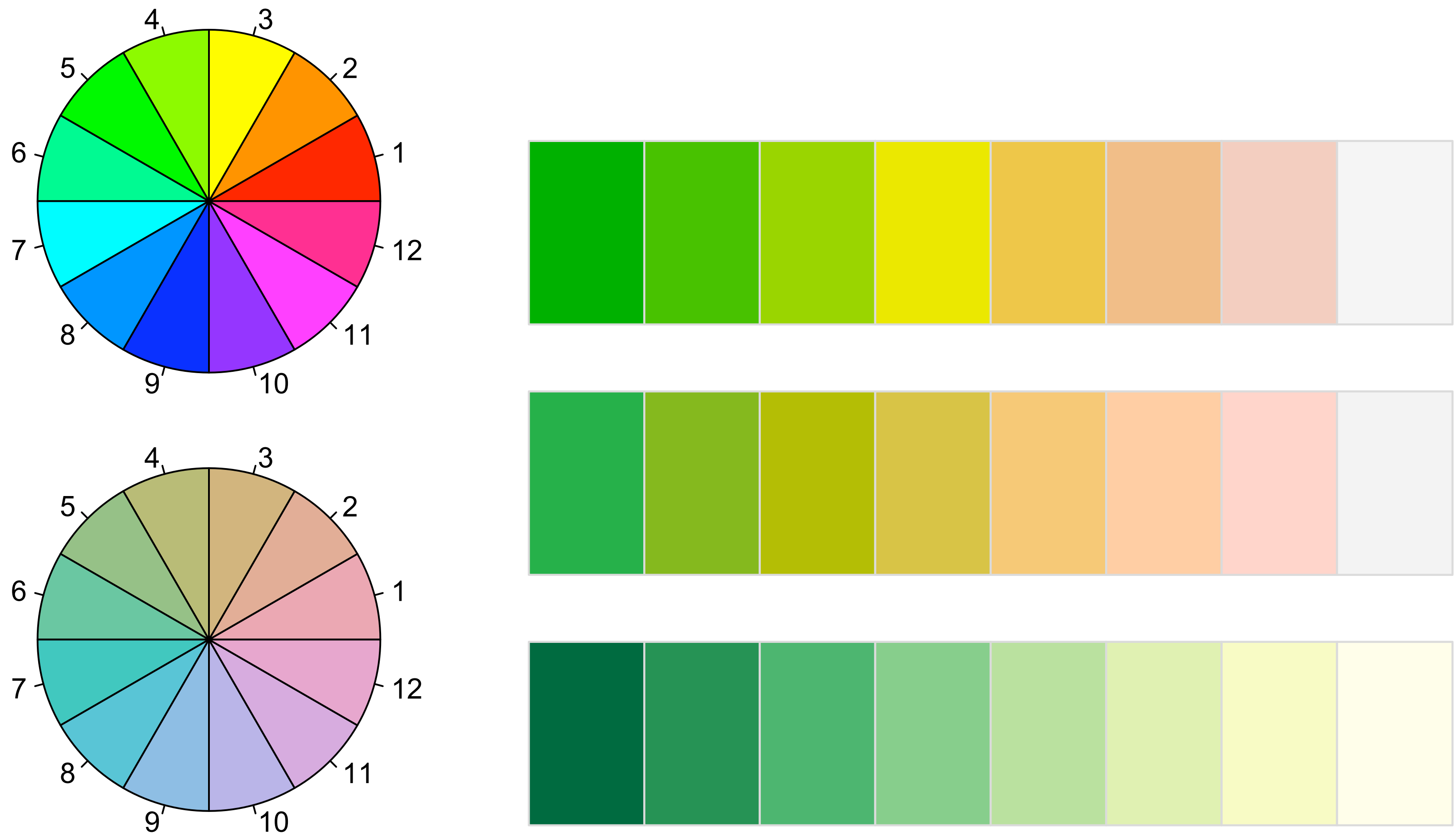
`colorspace` package has color schemes based on straight lines in a perceptually-based color space (rather than RGB).

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



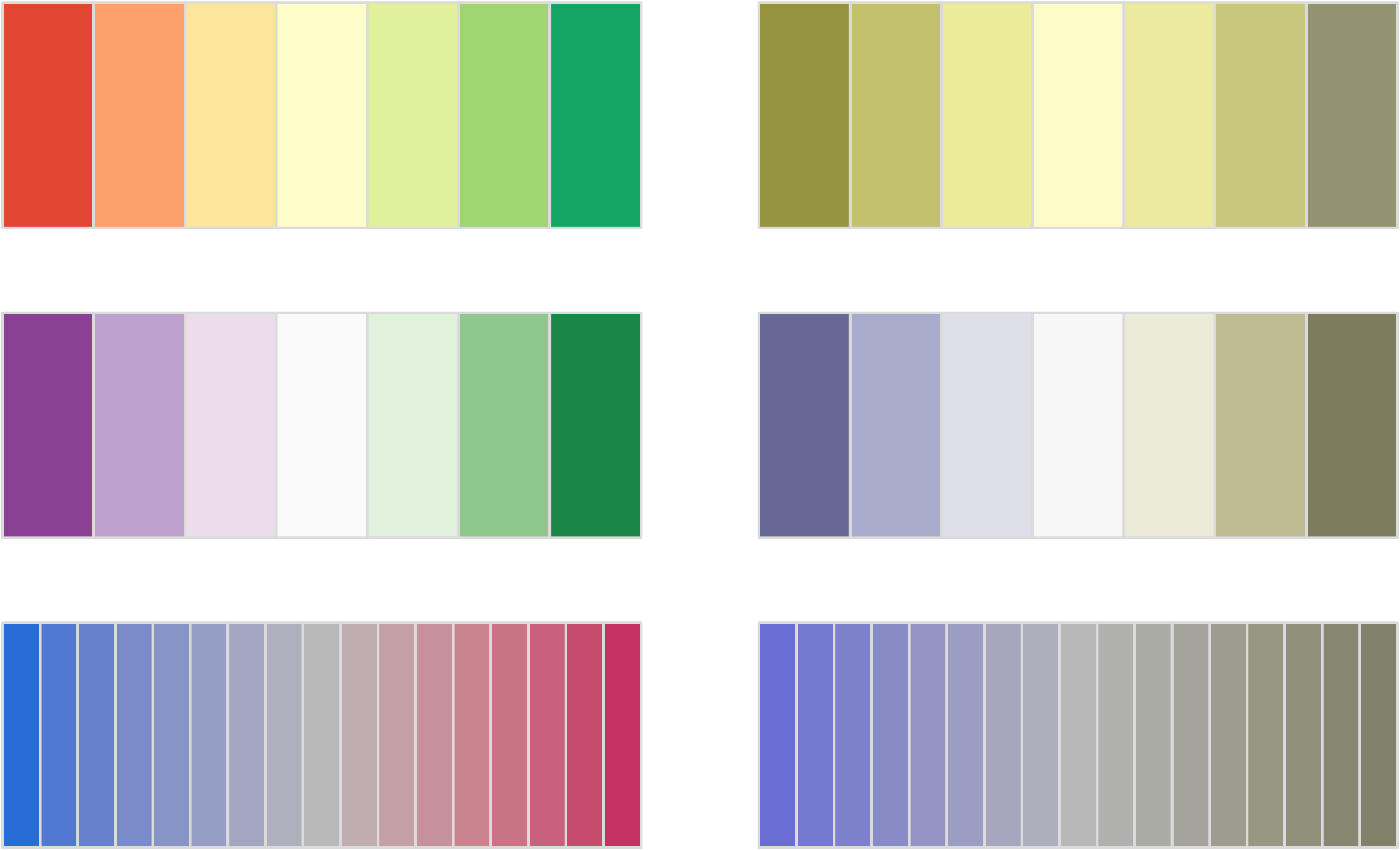
[Code for examples is in file `colorpalettes.R` on course website]

Color 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!)