STAT 545A
Class meeting #6
Monday, September 24, 2012

Dr. Jennifer (Jenny) Bryan

Department of Statistics and Michael Smith Laboratories

Review of last class

Quantitative summaries of a quantitative variable X (e.g. mean, median, variance, MAD, min, max, ...)

Above especially interesting when executed for levels of categorical variable(s) Y(, Z) via data aggregation techniques (e.g. tapply, by, ... or the plyr package?)

For small to medium datasets, stripplot is the way to go; SHOW ME THE DATA! SHOW ME THE DATA!

stripplot bells & whistles: jitter, type = "a" to add, e.g. the median, groups to superpose another categorical variable, auto.key = TRUE to get basic legend

Review of last class

For medium-to-large datasets, stripplot is either not enough or not even useful ➙ densityplot is my favorite way to convey an empirical distribution

Kernel density estimate at x = sum of bumps centered at observed data x_i. Shape of bumps = kernel; surprisingly not that important. Width of bumps = bandwidth; main tuning parameter.

Other options include boxplot, violin plot, histogram, ecdfplot

Sidebars: "<-" for assignment, formula interface

Sources for further study of topics covered:

Chapter 4 (“Graphics”) of Venables & Ripley (2002) has some good material on base R graphics. Sadly not available via SpringerLink.

Sources for further study of topics covered:

Chapters 2 ("Simple Usage of Traditional Graphics") and 3 ("Customizing Traditional Graphics") of Murrell (2006). This whole book is extremely valuable. Author's webpage* (for example, code to produce all figs in book is here). Google books search.

I'm sure there are others -- I learned what I know about base R graphics a long time ago. So I'd welcome feedback if students find more or better references that are more current.

* An issue with exporting from Keynote to PDF breaks this link. Use The Google and "Paul Murrell R graphics" to find the page. Also the relevant chapter(s) may have different number(s) in the 2nd edition, which now exists.

Code you see in this lecture can be found in these files:

bryan-a01-10-baseGraphicsStepByStep.R

bryan-a01-11-baseGraphicsPlotGapminderOneYear.R

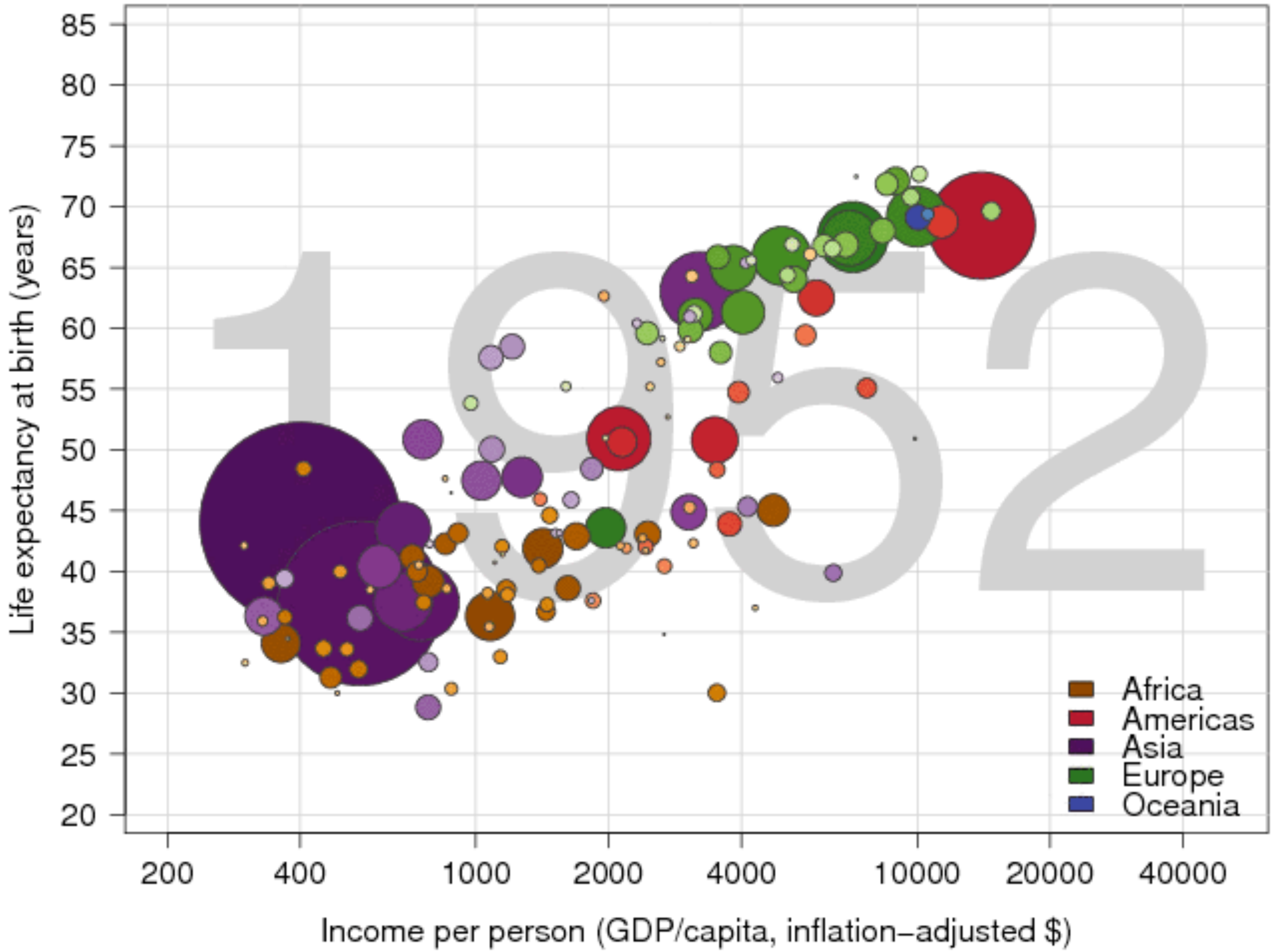
bryan-a01-12-baseGraphicsSoln.R

bryan-a01-30-makeGapminderColorScheme.R

bryan-a01-50-basicColorDemo.R

in this directory:
http://www.stat.ubc.ca/~jenny/notOcto/STAT545A/examples/gapminder/code/

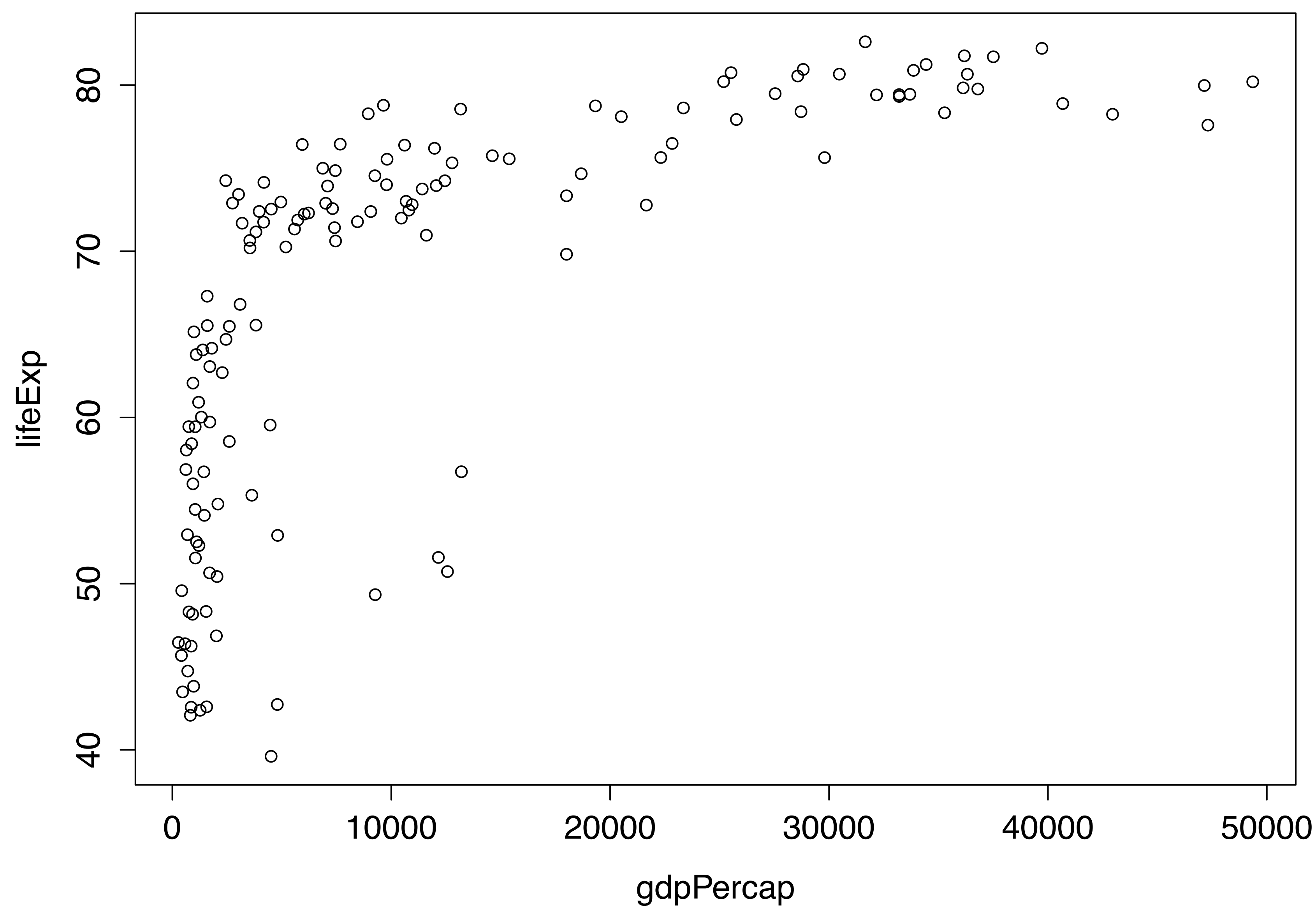
A JB 'solution' using base or traditional R graphics.



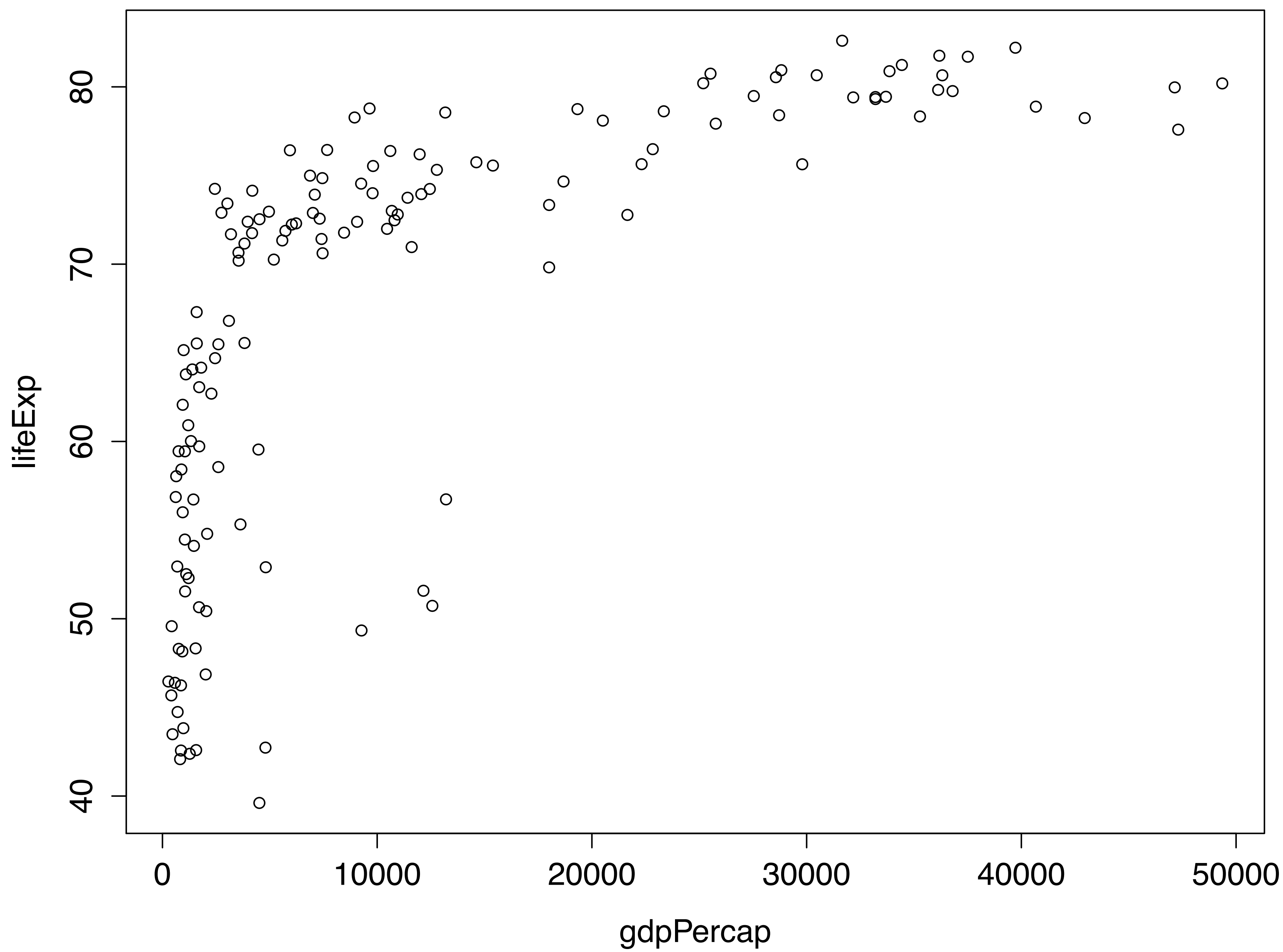
The animation is lost when exported to PDF.

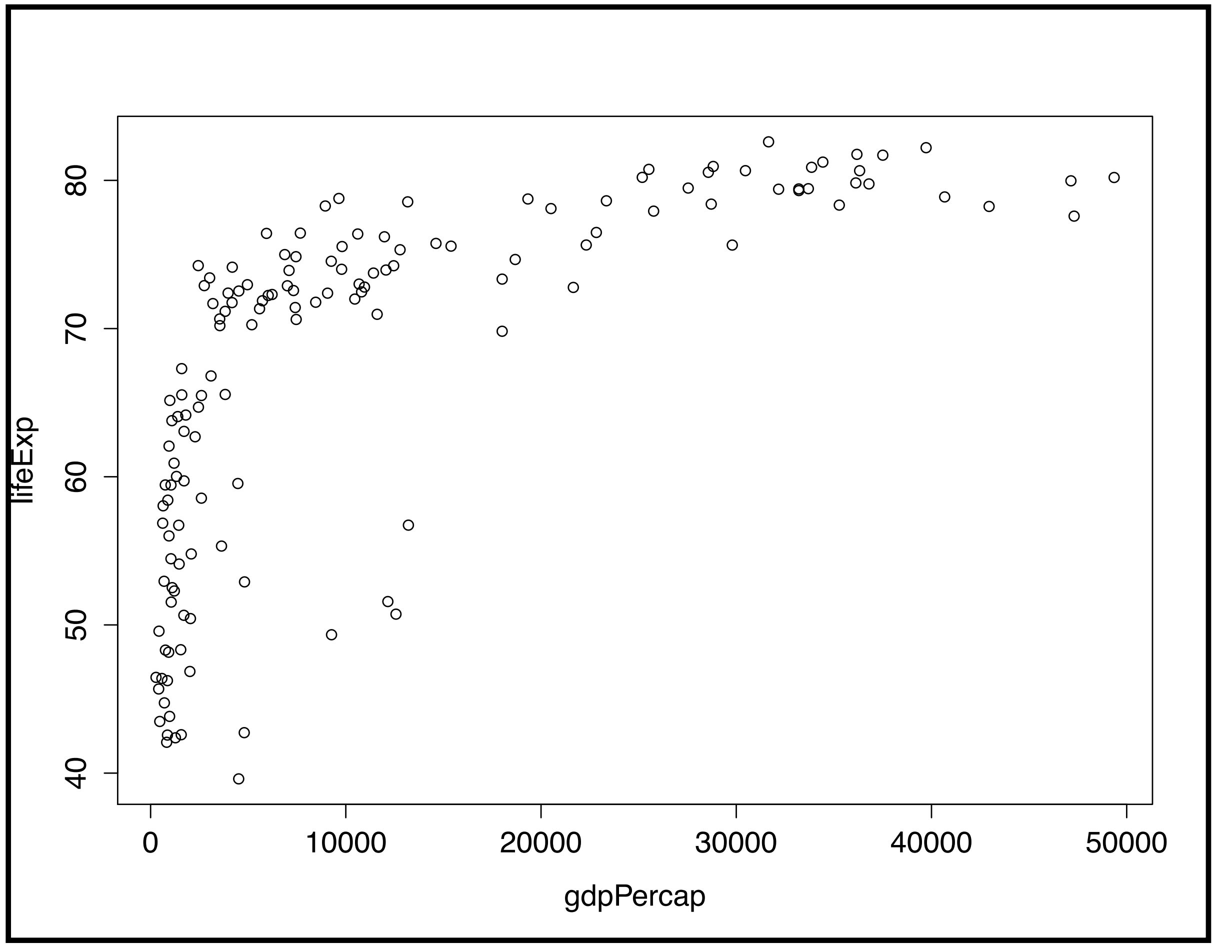
step-by-step development of the Gapminder figure/animation using base R graphics commands

```
(jYear <- max(gDat$year))
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear)
```

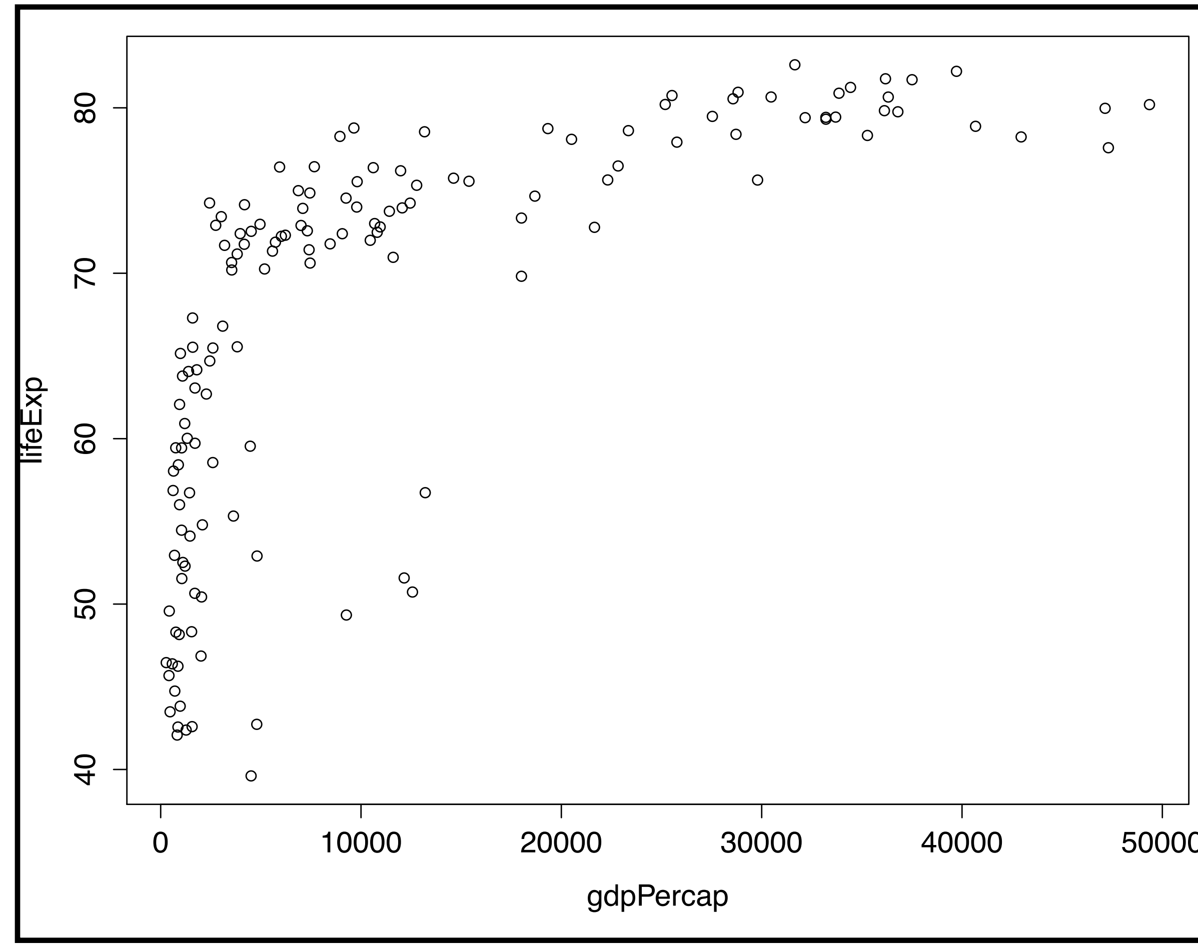


```
## take control of whitespace around plot
op <- par(mar = c(5, 4, 1, 1) + 0.1)
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear)
par(op)
```






```
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear)
```




```
## take control of whitespace around plot
op <- par(mar = c(5, 4, 1, 1) + 0.1)
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear)
par(op)
```

By default, base R graphics commands leave an excessive amount of whitespace around the plot. This -- and many other things -- will need explicit management via the par() command.

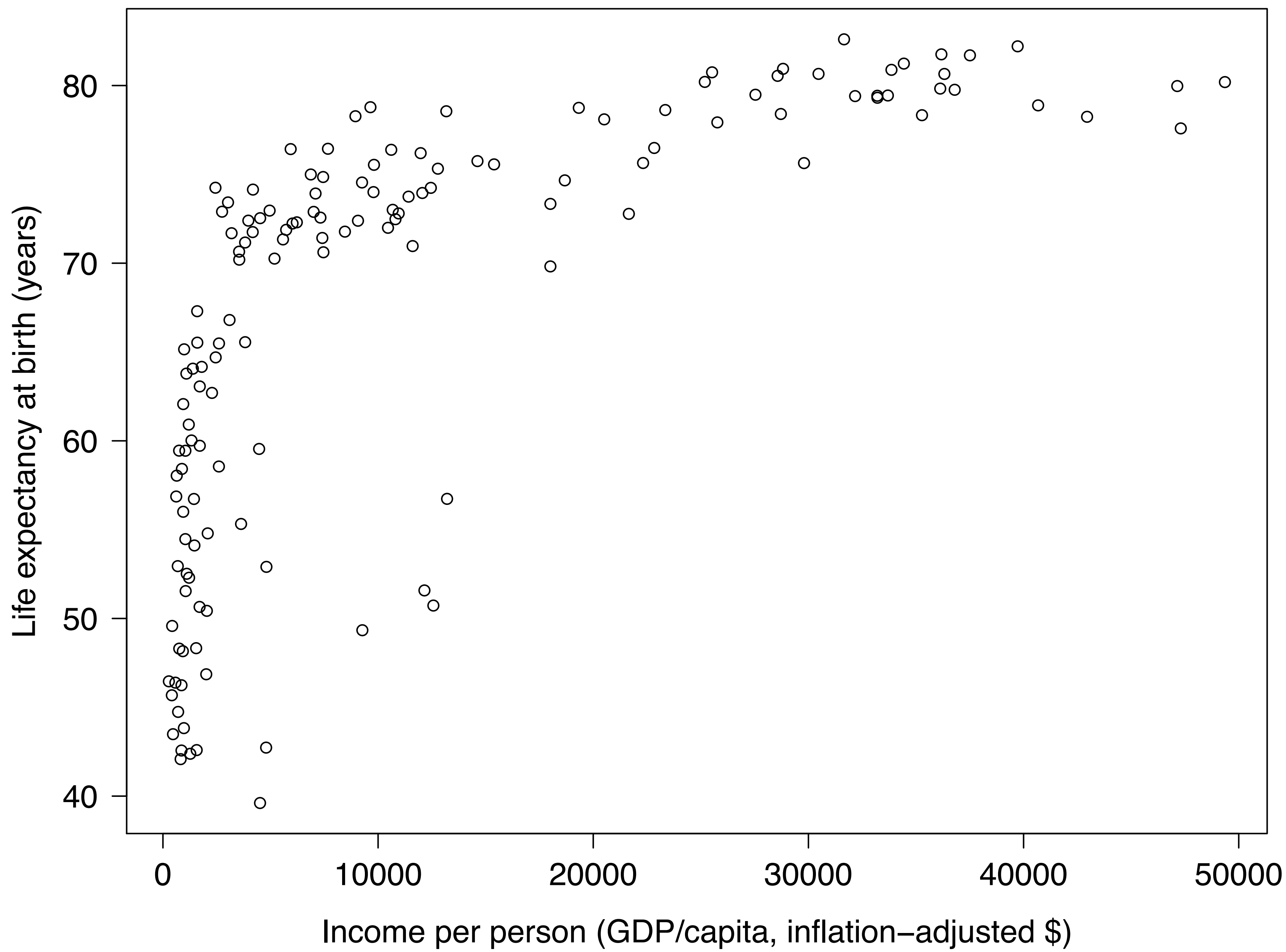
par() is used to set and query base R graphics parameters.

Read the documentation for par()!

To exert fine control over base R graphics, you will use par() alot. Which should tip you off why most figure-lovers are turning to lattice and ggplot2 these days.

Nonetheless, let's keep going. It's "best practice" to capture the current value of par when you begin to modify (current value is returned by the modification / new assignment) and then to restore that value when you're done. I will suppress this repetitive bit of code from here on.

```
## take control of axis labels, orientation of tick labels
jXlab <- "Income per person (GDP/capita, inflation-adjusted $)"
jYlab <- "Life expectancy at birth (years)"
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab)
```



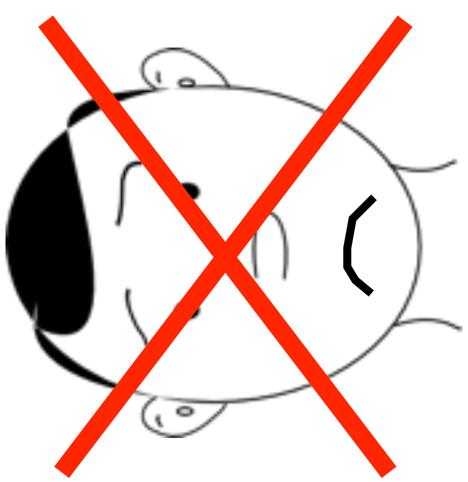
```r
## take control of axis labels
jXlab <- "Income per person (GDP/capita, inflation-adjusted $)"
jYlab <- "Life expectancy at birth (years)"
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab)
```

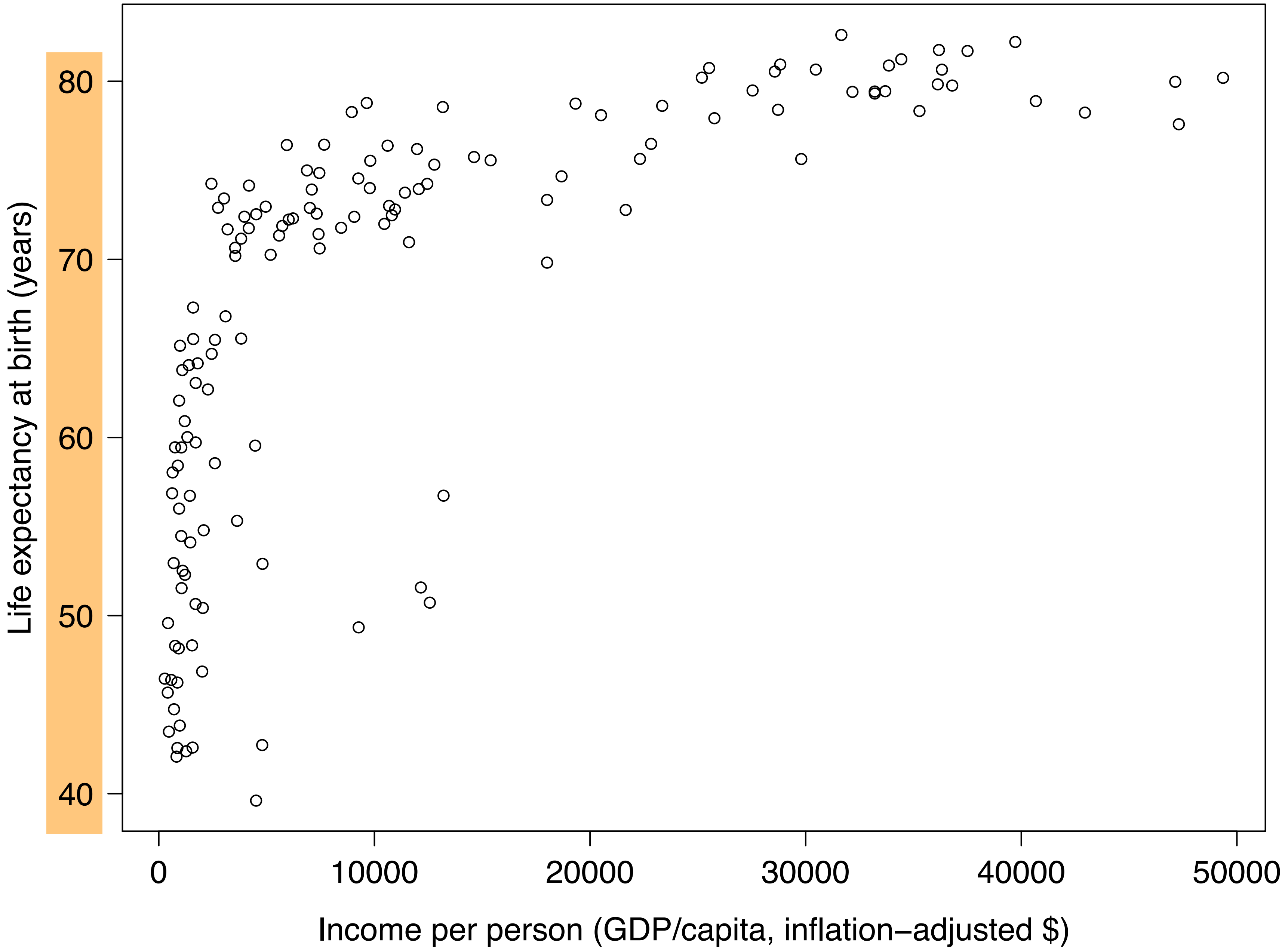
If you give good variable names, the default axis labels will be good enough most of the time.

When preparing a figure for a talk or paper, you will want to exert greater control.

Collect these sorts of Magic Text Strings at the top of a script that makes a Very Important Figure, for ease of modification and code re-use.

```
## take control of axis labels, orientation of tick labels
jXlab <- "Income per person (GDP/capita, inflation-adjusted $)"
jYlab <- "Life expectancy at birth (years)"
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab)
```





Recall your frustrations with axis manipulation?

For example, it would be nice if there can be more grid lines on the x-axis. It is easy to do on the original scale, but not on the log scale.

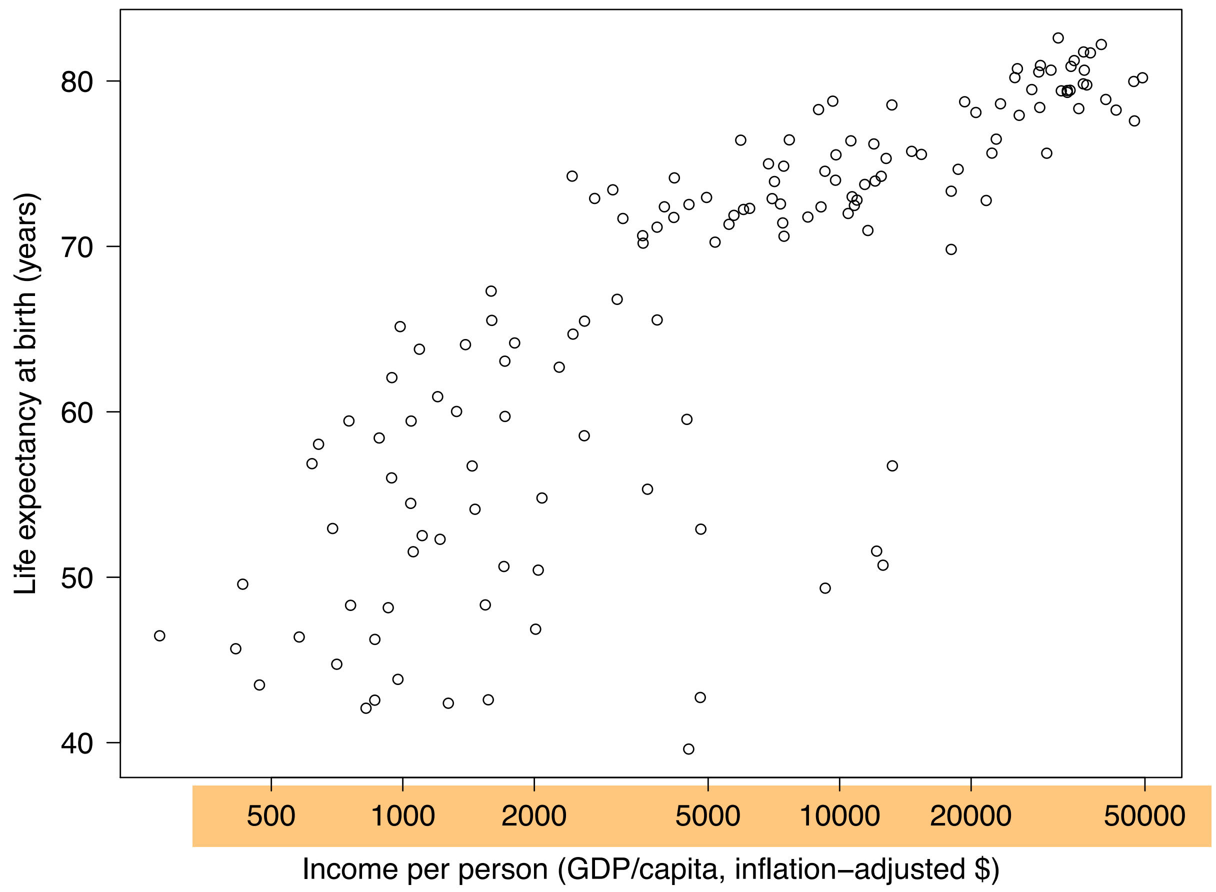
The X axis is not uniformly distributed

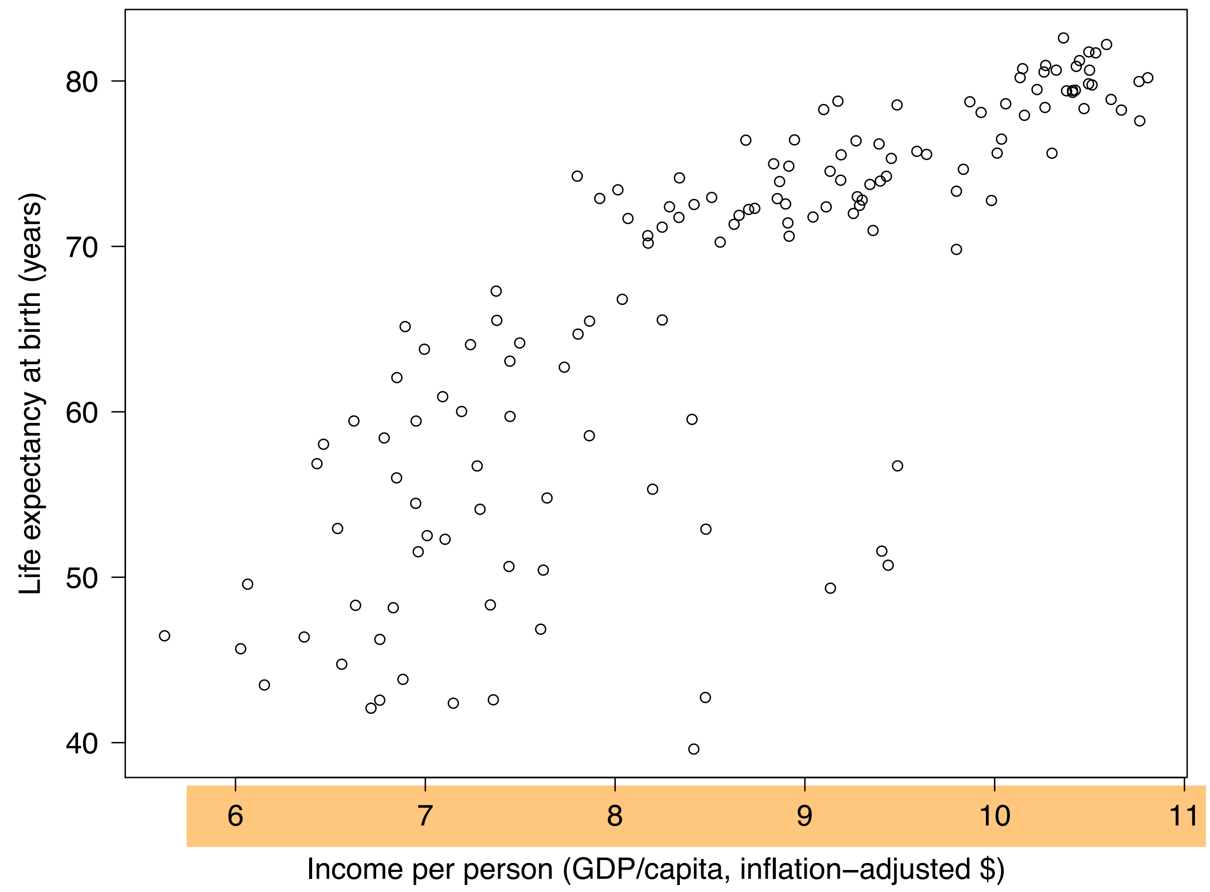
Also, I could not figure out how display the countries which have small populations on the graphs. I did not find out the actual range of Income per Person, I just applied the Logarithmic function on Income per Person.

some countries have an extreme amount of income relative to the other countries

For example, I do not know how to use log scale but still label the axis with original values.

*Note: These frustrations expressed by past STAT 545A students. Your mileage may vary.






```
## log transform the x = gdpPercap
## axis using the 'log' argument
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x')
```

```
## log transform the x = gdpPercap
## axis 'by hand'
plot(lifeExp ~ log(gdpPercap), gDat,
     subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab)
```

```
## log transform the x = gdpPercap
## axis using the 'log' argument
plot(lifeExp ~ gdpPercap, gDat,
     subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x')
```

This is the preferred way to log transform the x variable. Works same way for y variable. Results in axis tick marks and labels that are easier for reader to understand, i.e. are based on the original scale.

```
## log transform the x = gdpPercap
## axis 'by hand'
plot(lifeExp ~ log(gdpPercap), gDat,
     subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab)
```

Recall the frustration over drawing and sizing circles?

When I was trying to relate the population size to the size of points, it takes me about 1 hour,
because I need to scale the population properly. I use two scale method.
1) size=10*[pop-min(pop)]/[max(pop)-min(pop)]
2) size= sqrt(pop)/4000

Method(2) works better.

Find the right function or parameter to determine the radius of the circle symbols

Found the use of "symbols" and its documentation helps me to set circles and colors!!! I can set different colors for different countries, but the same country always uses the same color. The size of circles is increasing function of its population of "current data" or "the most recent available data".... I feel very lucky to find 'symbols'.

Finally, I tried to vary the size of the dots. The basic principle was simple, because there is a parameter to the 'points' function to scale the size of the marker ('cex'). What took me a surprisingly long time was getting the formula for the size of the marker 'right'.

I tried various ratios, scalings, and log transforms, and most of them yielded points that were far too uniform in size. Eventually, I decided that making this proportional to the ratio of population to the smallest value was the right approach, but that the proportion should be in area of the marker. Taking a square root and scaling it to keep the circles from getting too big ended up with effect pretty similar to GapMinder. This feature alone probably took me an hour.

*Note: These frustrations expressed by past STAT 545A students. Your mileage may vary.

The next task: conveying two more pieces of information

- color ↔ continent / country
- circle size ↔ population

Big picture: It's quite easy to depict a 4-dimensional dataset with a scatterplot.

```
## map pop into circle radius
jPopRadFun <- function(jPop) {            # make area scale with pop
  sqrt(jPop/pi)
}
plot(jPopRadFun(pop) ~ pop, gDat)         # looks promising

with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```

It can be surprisingly vexing to transform a variable into ... for example, circle radii or colors ... for an effective display! Expect to give this careful attention.

```
## map pop into circle radius
jPopRadFun <- function(jPop) {          # make area scale with pop
  sqrt(jPop/pi)
}
```

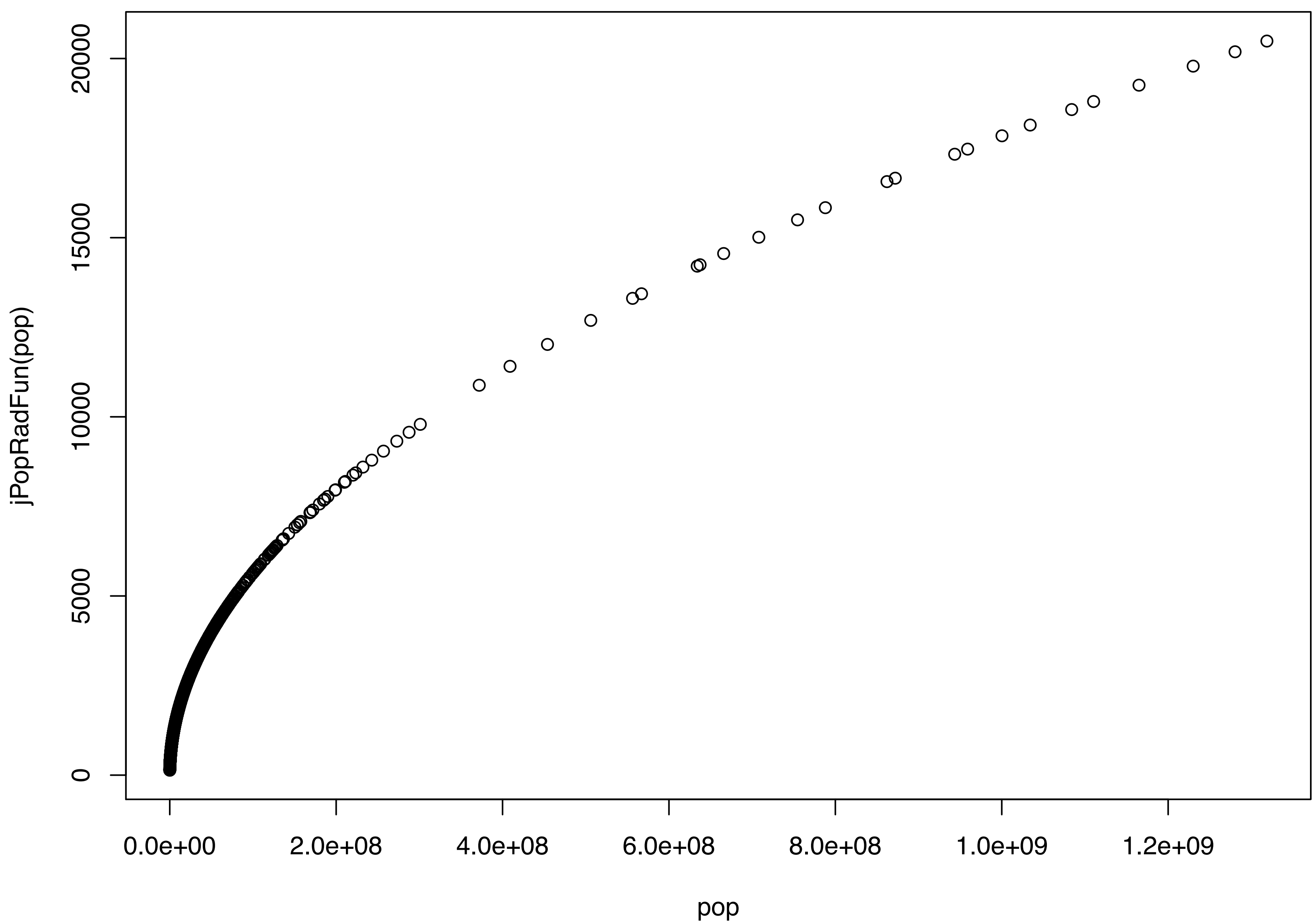
$$area = \pi r^2$$

$$area \Leftrightarrow pop$$

$$r = \sqrt{pop / \pi}$$

Try to find a principled way to proceed. In this case, I claim that area of circle should correspond to population, which implies the above transformation.

```
## map pop into circle radius
jPopRadFun <- function(jPop) {           # make area scale with pop
  sqrt(jPop/pi)
}
plot(jPopRadFun(pop) ~ pop, gDat)        # looks promising
```



Plot this for a sanity check before throwing into main figure command.

```r
## map pop into circle radius
jPopRadFun <- function(jPop) {             # make area scale with pop
  sqrt(jPop/pi)
}
plot(jPopRadFun(pop) ~ pop, gDat)          # looks promising

with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```

The symbols() command plots ... symbols! You can specify a shape, e.g. circle, and more, e.g. size.

I won't talk about this a lot because we risk getting hyper-specific about the Gapminder example.

Frankly, this doesn't come up often in real life for me.

```
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop),
             inches = 0.7,
             fg = jDarkGray, bg = color,
             las = 1, xlab = jXlab, ylab = jYlab,
             log = 'x'))
```

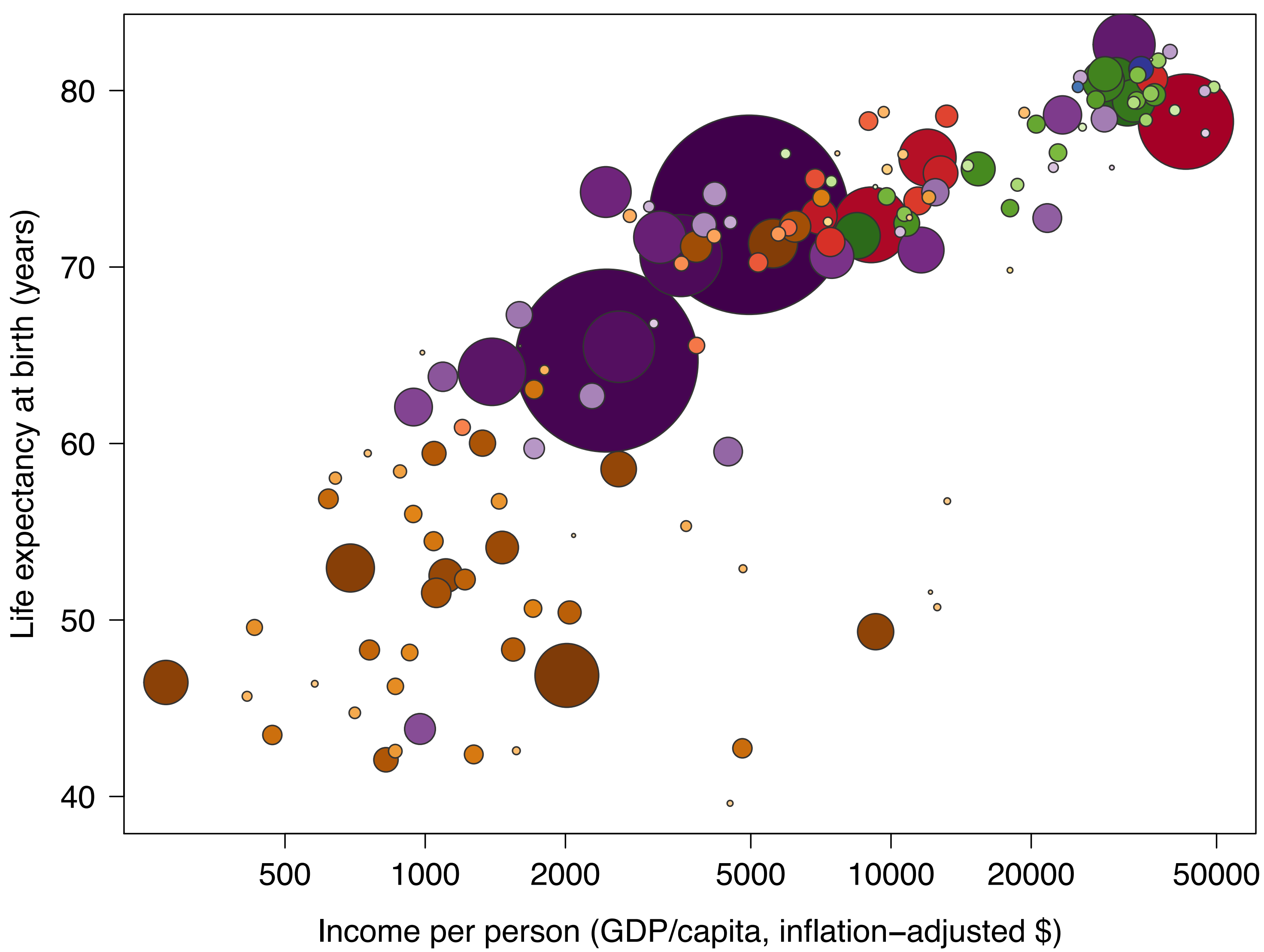
```
Error in plot.window(...) : Logarithmic
axis must have positive limits
```

```
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop),
             inches = 0.7,
             fg = jDarkGray, bg = color,
             las = 1, xlab = jXlab, ylab = jYlab,
             log = 'x'))
```

Morally, the above should work. But, in practice, it does not. I suppose due to the fact that the circle centres are in ‘legal’ places, but the entire circle is not.

More hints about what’s irritating about base graphics .... You have to do *everything* yourself.

```r
## use 'plot()' to set things up and then add other elements
plot(lifeExp ~ gdpPercap, gDat, subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x', type = "n")
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```

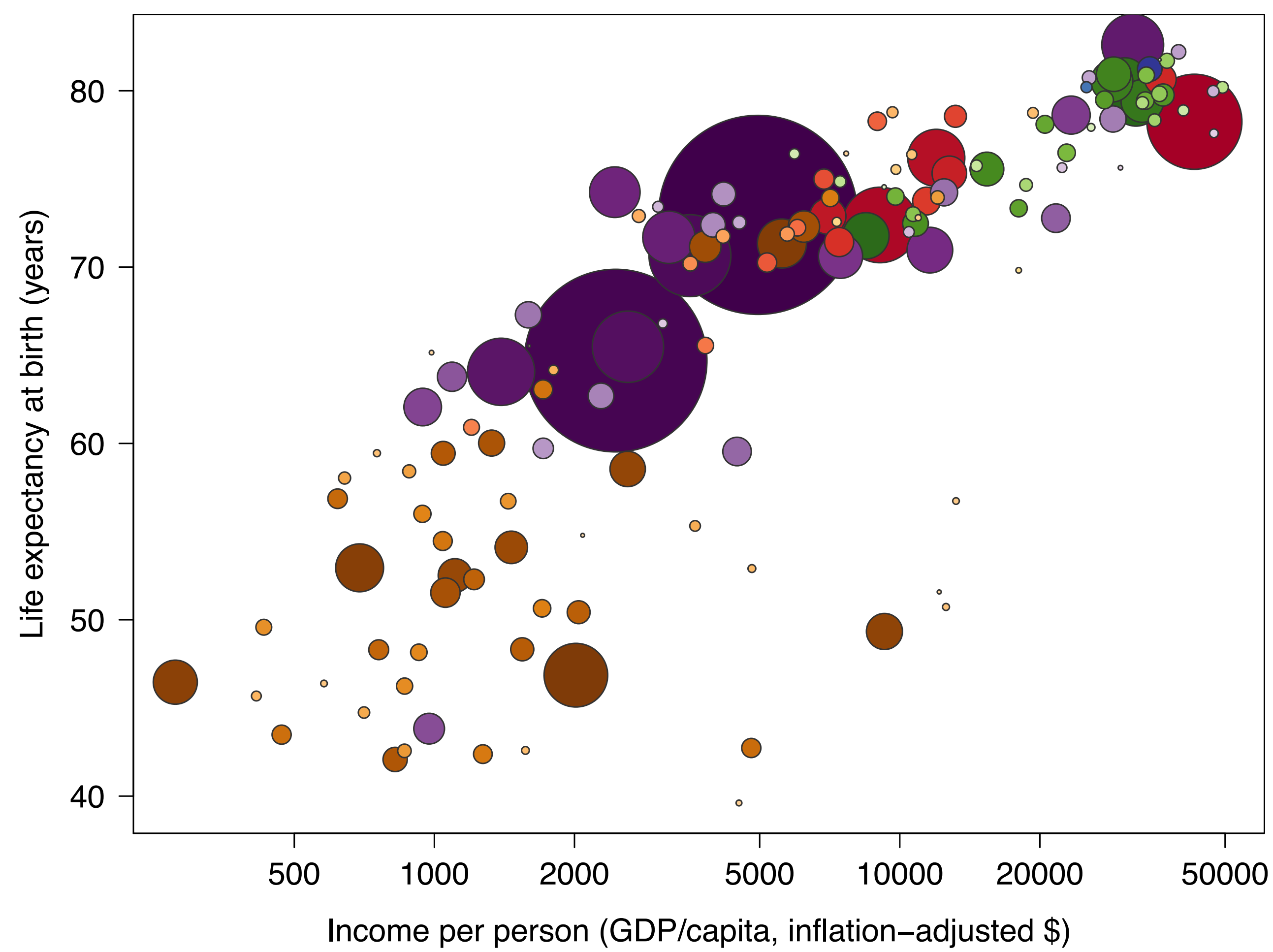


```r
## use 'plot()' to set things up and then add other elements
plot(lifeExp ~ gdpPercap, gDat, subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x', type = "n")
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```

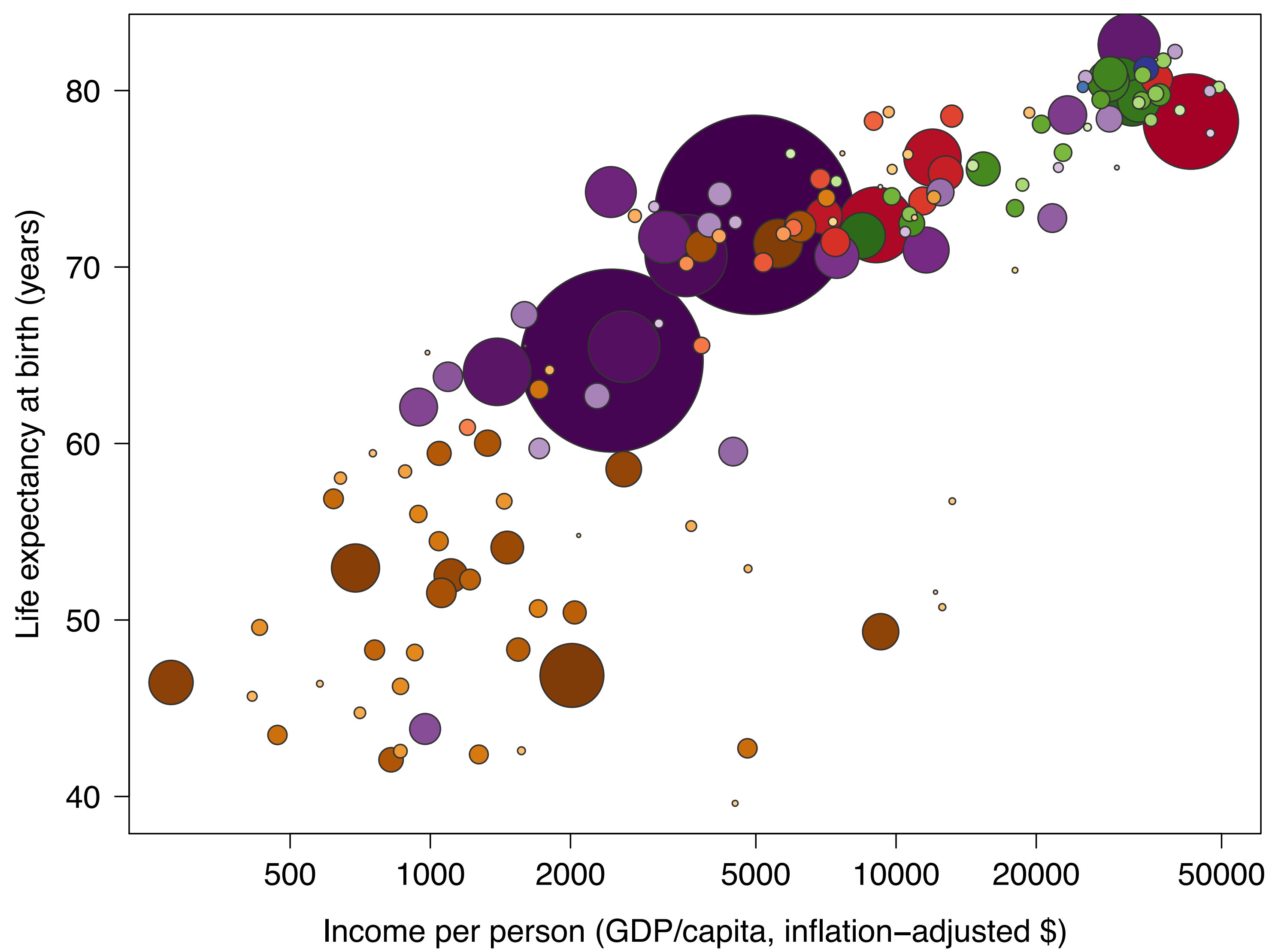
This is a typical workflow in ambitious plots made with base R graphics commands: call plot() to set up a coordinate system and do precious little else. Then call other functions to add desired elements.

```
## Sort by year (increasing) and population (decreasing)
## Why? So larger countries will be plotted "under" smaller ones.
gDat <- with(gDat, gDat[order(year, -1 * pop),])
```

Sidebar: I changed the order of the rows in the dataset to address overplotting. Example where the result (a figure) is unavoidably sensitive to the row order of the input data.



```
## use 'plot()' to set things up and then add other elements
plot(lifeExp ~ gdpPercap, gDat, subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x', type = "n")
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```

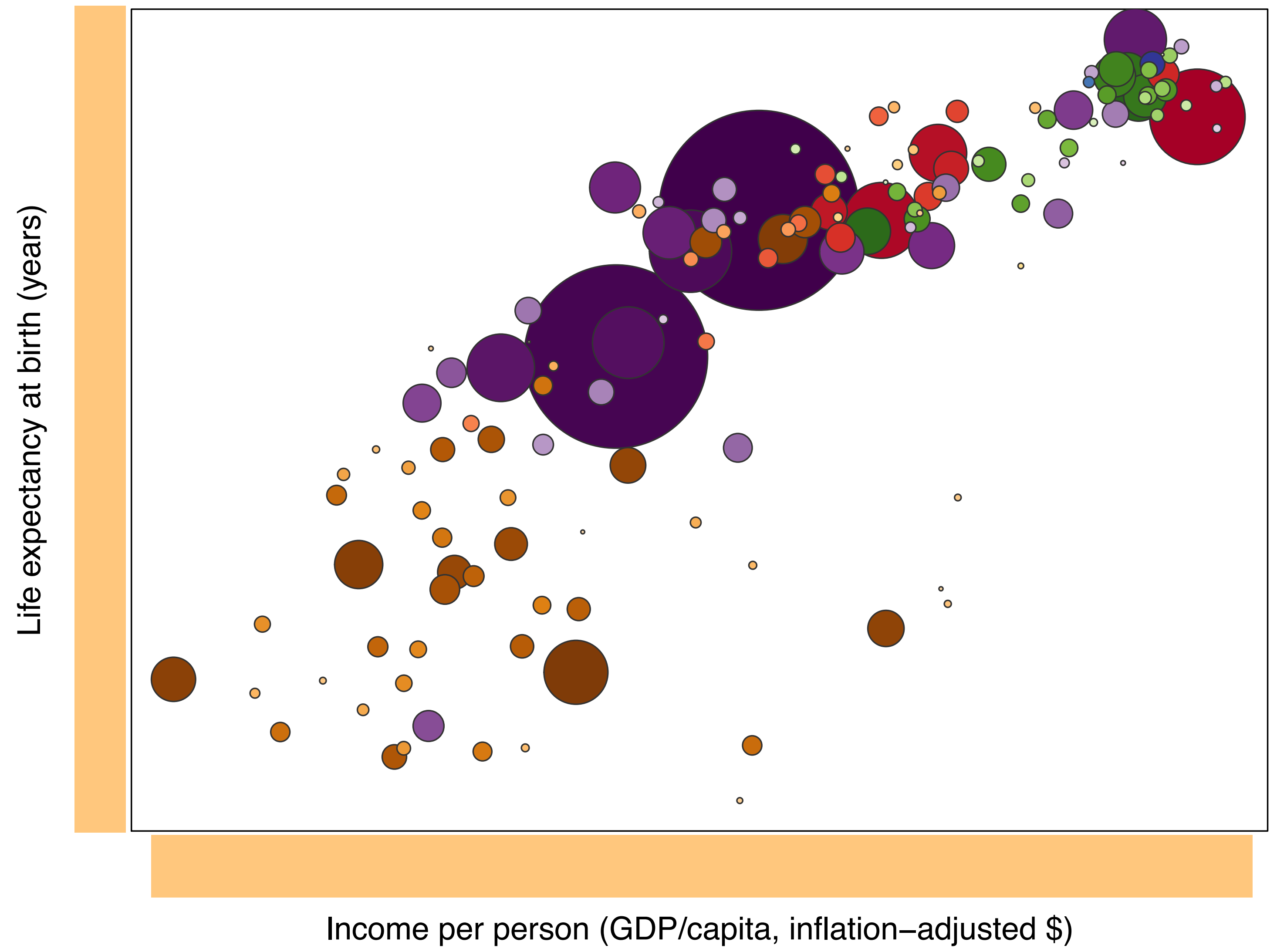


```r
## use 'plot()' to set things up and then add other elements
plot(lifeExp ~ gdpPercap, gDat, subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x', type = "n")
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```

I have added a variable that holds the color I wish each circle to be filled with. Telling symbols() to use that color is trivial. Creating the color scheme and constructing this color variable is not. Shown later.

```
> peek(gDat)
     continent       country   color year      pop lifeExp gdpPercap
1356    Europe       Belgium #6DAD35 1962  9218400  70.250 10991.207
137     Africa Congo, Rep. #F7AE55 1967  1179760  52.040  2677.940
418     Africa       Namibia #FDBA67 1972   821782  53.867  3746.081
118     Africa       Comoros #FDD6A2 1992   454429  57.939  1246.907
168     Africa      Djibouti #FDDCAF 1992   384156  51.604  2377.156
1319      Asia  Yemen, Rep. #A883B8 2007 22211743  62.698  2280.770
963       Asia      Cambodia #B797C6 2007 14131858  59.723  1713.779
```

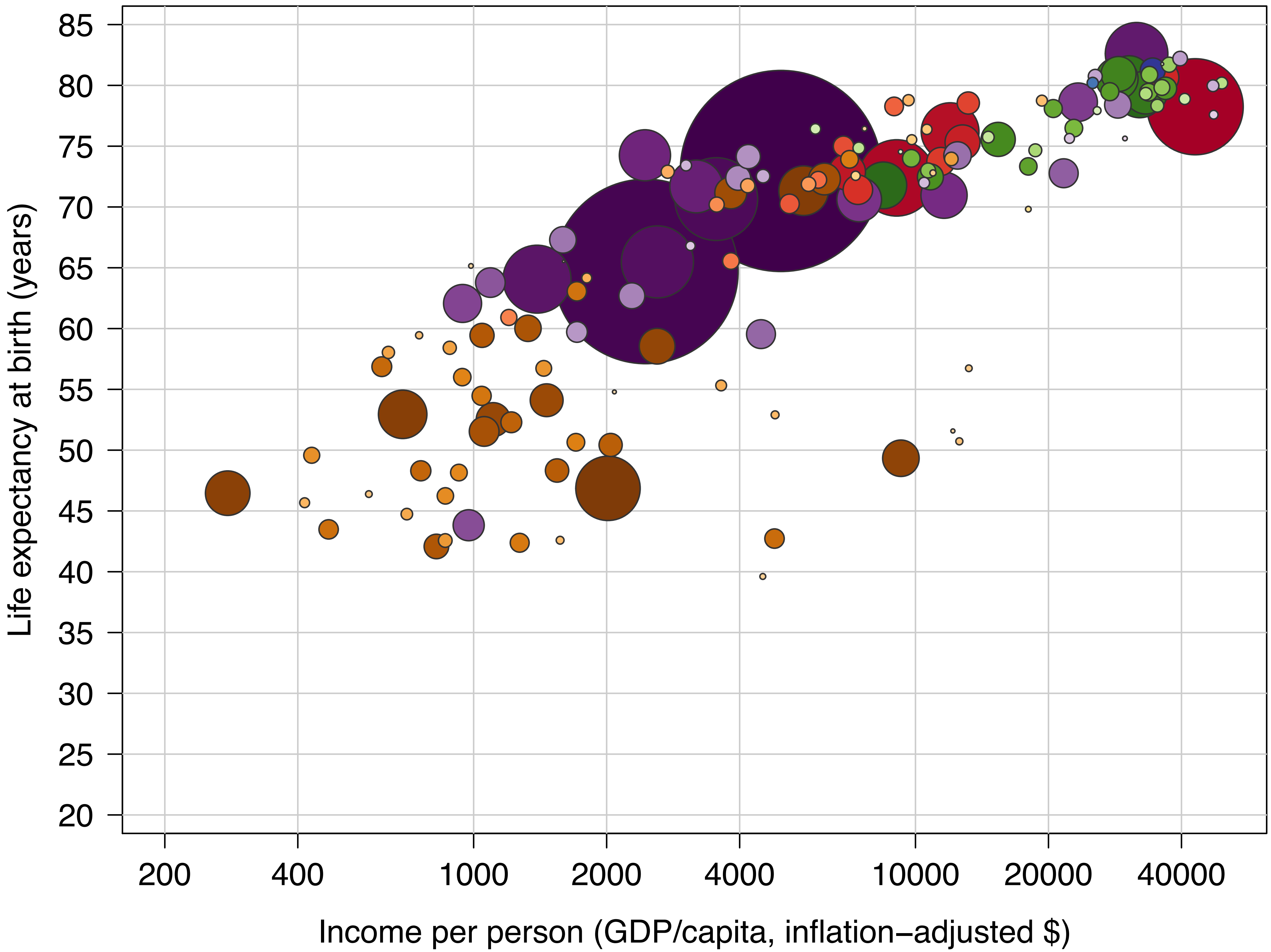
```
## suppress the automatic axes (tick marks)
## in anticipation of taking direct control
plot(lifeExp ~ gdpPercap, gDat, subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x', type = "n",
     xaxt = "n", yaxt = "n")
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```



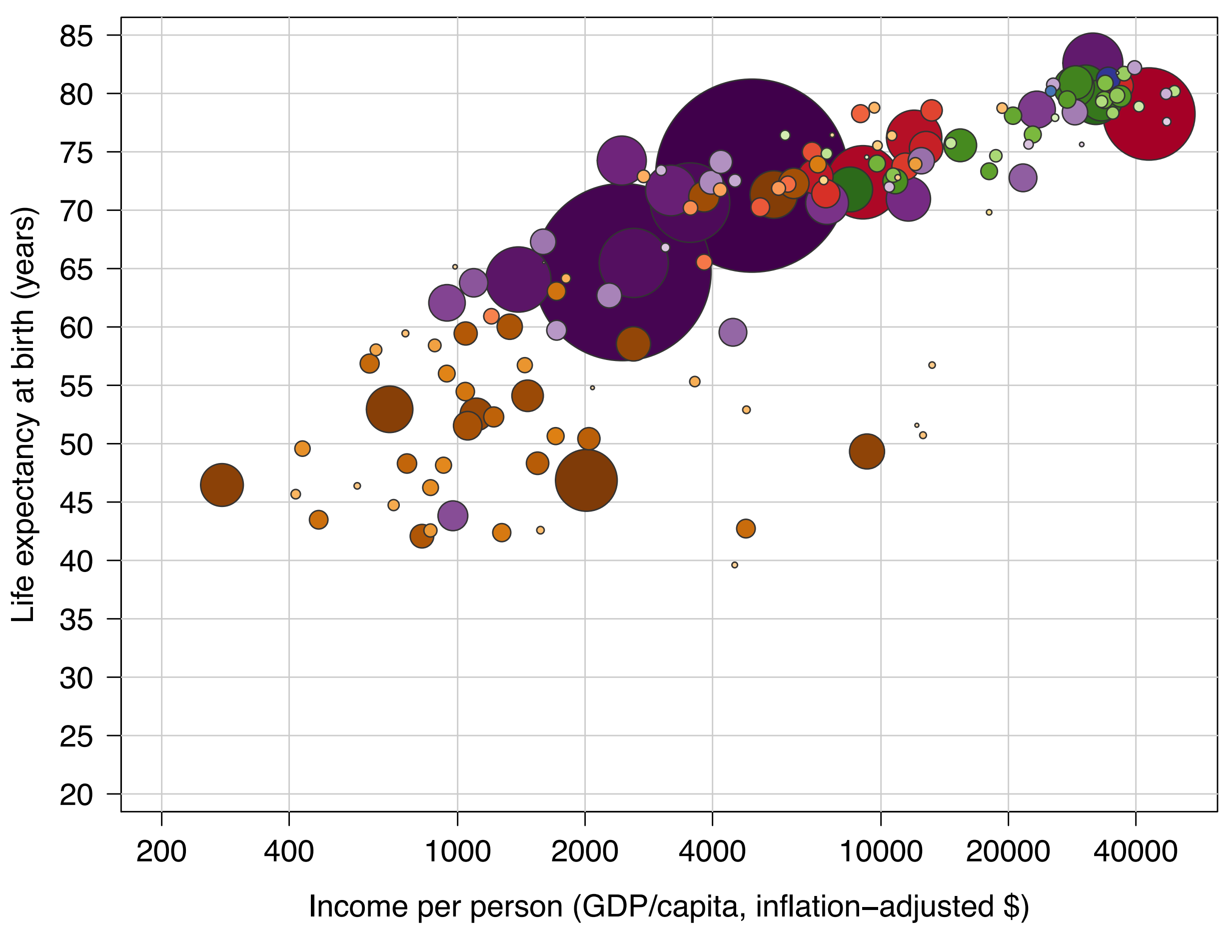
```
## suppress the automatic axes (tick marks)
## in anticipation of taking direct control
plot(lifeExp ~ gdpPercap, gDat, subset = year == jYear,
     las = 1, xlab = jXlab, ylab = jYlab,
     log = 'x', type = "n",
     xaxt = "n", yaxt = "n")
with(subset(gDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```

Another example of suppressing default plot elements. Fancy figures made with R graphics often have this counter-intuitive feel: two steps backward, then one step forward. Then another forward and so on. More ways to suppress stuff include ‘ann = FALSE’ and ‘bty = “n”’.

Axis tick marks & labels are back!
Reference grid has appeared.



```
jXlim <- c(200, 50000)
jYlim <- c(21, 84)
gdpTicks <- c(200, 400, 1000, 2000, 4000, 10000, 20000, 40000)
lifeExpTicks <- seq(from = 20, to = 85, by = 5)
jGray <- 'grey80'
plot(lifeExp ~ gdpPercap, <same old stuff here>,
     xlim = jXlim, ylim = jYlim)
axis(side = 1, at = gdpTicks, labels = gdpTicks)
axis(side = 2, at = lifeExpTicks, labels = lifeExpTicks, las = 1)
abline(v = gdpTicks, col = jGray)
abline(h = lifeExpTicks, col = jGray)
with(subset(gDat, year == jYear),
     symbols(<same old stuff here>))
```



```
> sapply(gDat[c('gdpPercap','lifeExp')], range)
       gdpPercap lifeExp
[1,]    241.1659  23.599
[2,] 113523.1329  82.603
> sapply(gDat[c('gdpPercap','lifeExp')], quantile,
+        probs = c(0.9, 0.95, 0.98))
    gdpPercap lifeExp
90%  19449.14 75.0970
95%  26608.33 77.4370
98%  33682.22 79.3694
```

Once you take a certain amount of control, it's almost inevitable that you will have to finish the job. For example, you may need to explicitly specify axis limits. There will be some trial-and-error, but commands like the above are helpful to get things rolling.

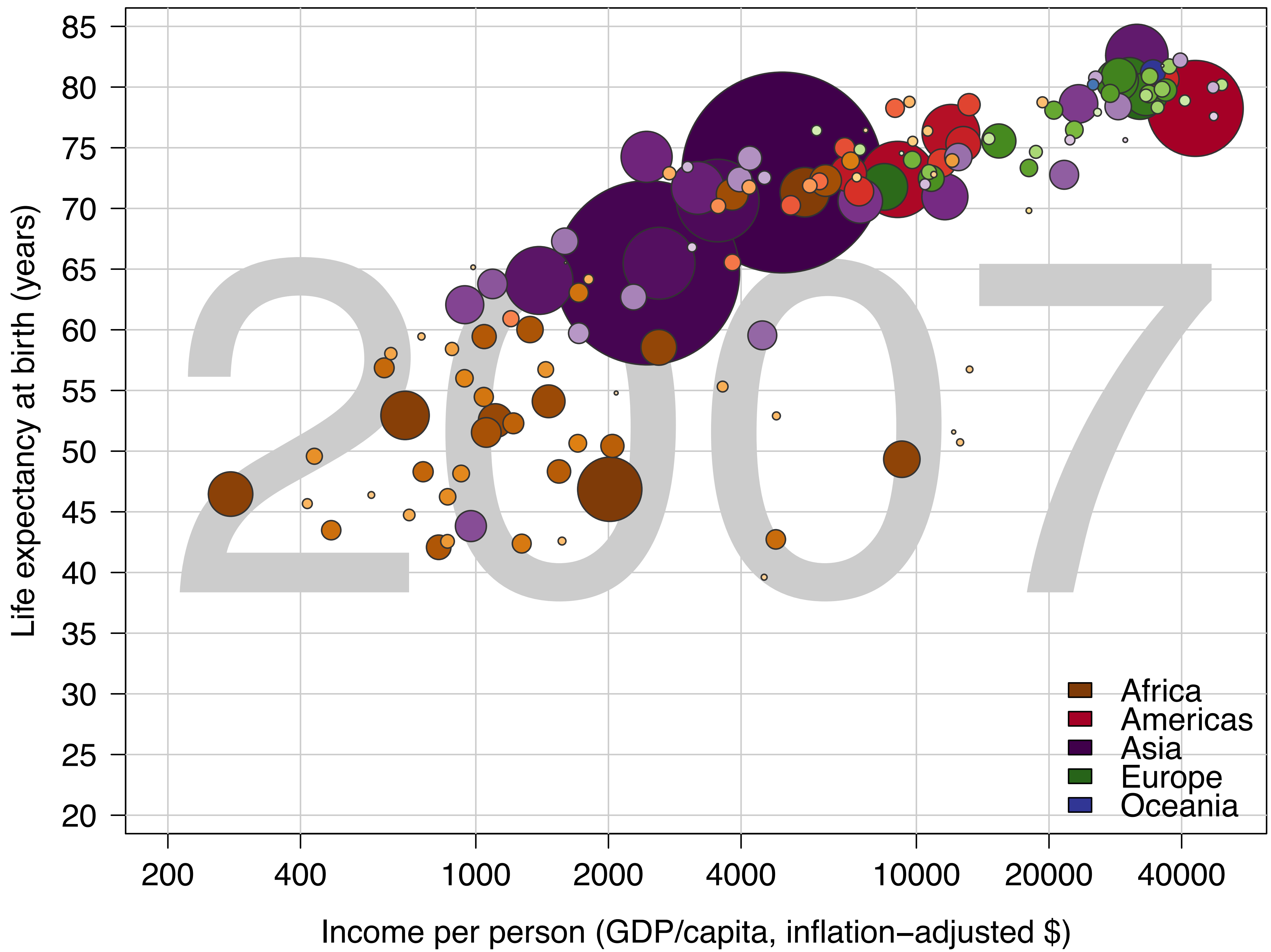
Recall your frustrations with a legend?

I tried to add a legend for the colours and continents, but it was quite the disaster. The function call seems simple enough but it doesn't behave as I'd expect.

legend (colors do not correspond to the data points

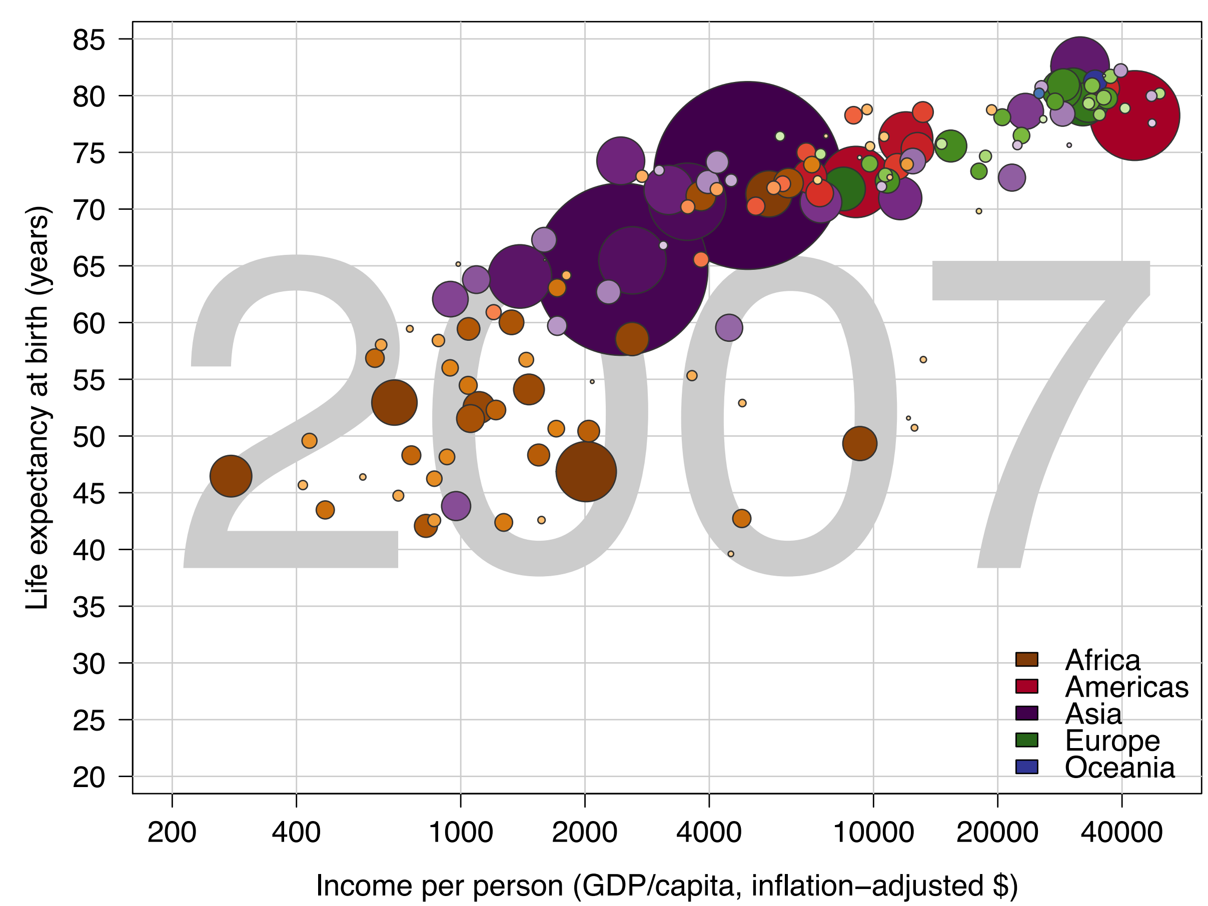
*Note: These frustrations expressed by past STAT 545A students. Your mileage may vary.

The year has been placed in the plot background. We have a legend linking a color family to a continent.



```
## place YEAR as a watermark in background,
## include a legend
yearCex <- 15
plot(lifeExp ~ gdpPercap, ...)
text(x = sqrt(prod(jXlim)), y = mean(jYlim),
     jYear, adj = c(0.5, 0.5), cex = yearCex, col = jGray)
<snip, snip>
legend(x = 'bottomright', bty = 'n',
       legend = names(colorAnchors),
       fill = sapply(colorAnchors, function(z) z[1]))
```

Details on colorAnchors will become clear when we go back and construct the color scheme.



The really last frontier: conveying one more piece of information

- time ↔ ‘frame’ in an animation

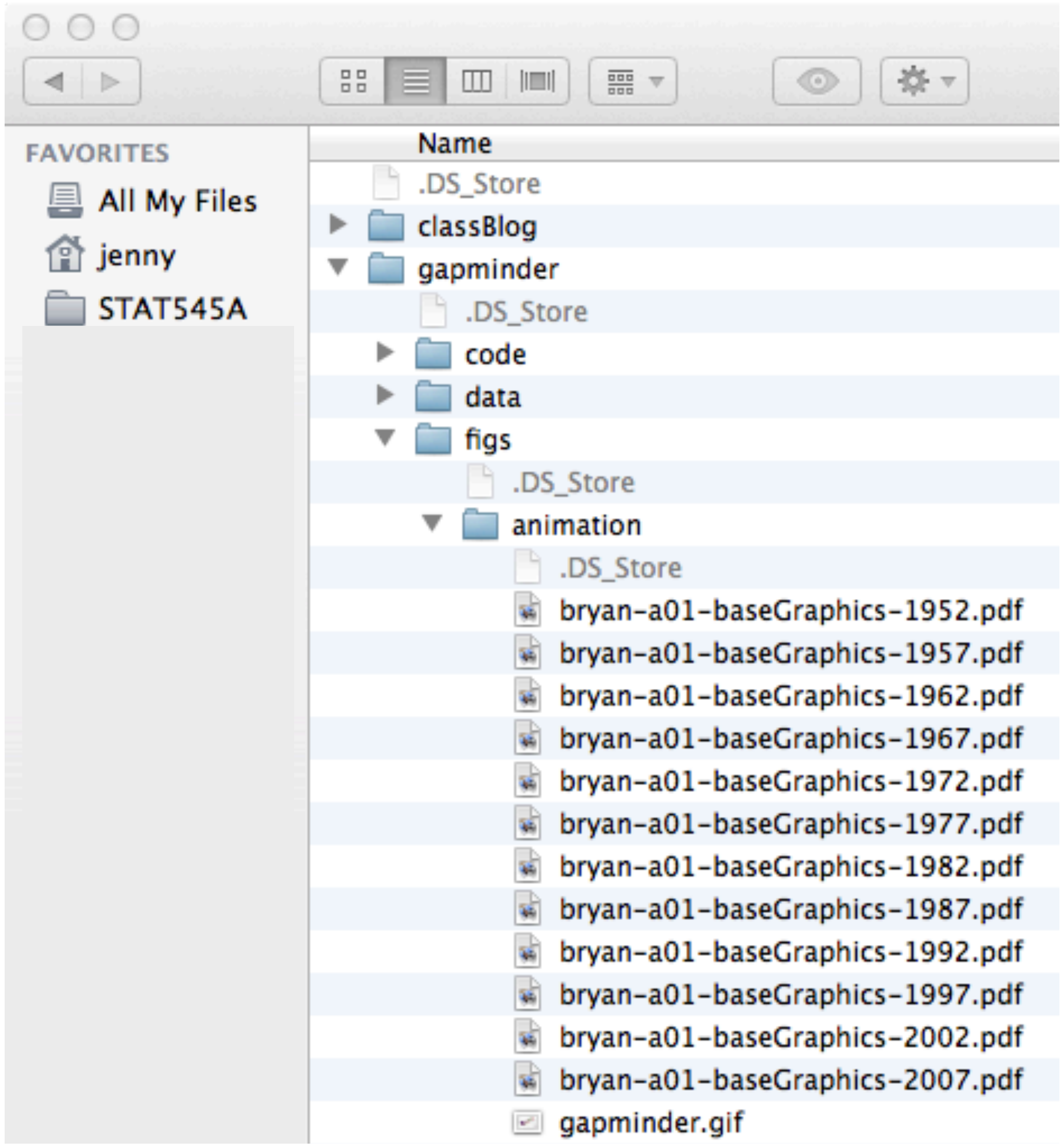
Big picture: It’s ~~quite easy~~ somewhat easy to depict a 5-dimensional dataset with a series of scatterplots.

```r
writeToFile <- TRUE                    # write a figure file for each year?

for(jYear in sort(unique(gDat$year))) {
  plot(lifeExp ~ gdpPercap, ...)
  <snip, snip>
  symbols(gDat$gdpPercap[gDat$year == jYear],
          gDat$lifeExp[gDat$year == jYear],
          circles = sqrt(gDat$pop[gDat$year == jYear]/pi),
          add = TRUE, fg = jDarkGray,
          bg = gDat$color[gDat$year == jYear],
          inches = 0.7)
  legend(x = 'bottomright', bty = 'n',
         legend = names(colorAnchors),
         fill = sapply(colorAnchors, function(z) z[1]))
  if(writeToFile) {
    dev.print(pdf,
              file = paste0(whereAmI,"figs/animation/bryan-a01-baseGraphics-",
                jYear, ".pdf"),
              width = 9, height = 7)
  }
  Sys.sleep(0.5)                       # gives 'live' figures an
                                       # animated feel
}
```

Code developed earlier is easily inserted inside a loop over year. Nice to build in a toggle for writing to file. Construct informative file names programmatically.

```r
writeToFile <- TRUE                         # write a figure file for each year?

for(jYear in sort(unique(gDat$year))) {
  op <- par(mar = c(5, 4, 1, 1) + 0.1)
  plotGapminderOneYear(jYear, gDat, continentColors)
  if(writeToFile) {
    dev.print(pdf,
              file = paste0(whereAmI,"figs/animation/bryan-a01-baseGraphics-",
                jYear, ".pdf"),
              width = 9, height = 7)
  }
  Sys.sleep(0.5)                                      # gives 'live' figures an
                                                      # animated feel
}
par(op)
```

After incremental, interactive development, figure-making code is easily packaged in a function and inserted inside a loop over year. Nice to build in a toggle for writing to file. Construct informative file names programmatically using paste() and relevant variables, such as year.



Figures are created for each year.
Filename tells me what the figure is.

I cannot stress enough how useful it is to

[1] write figures to file with a line of R code, not a casual spontaneous mouse event

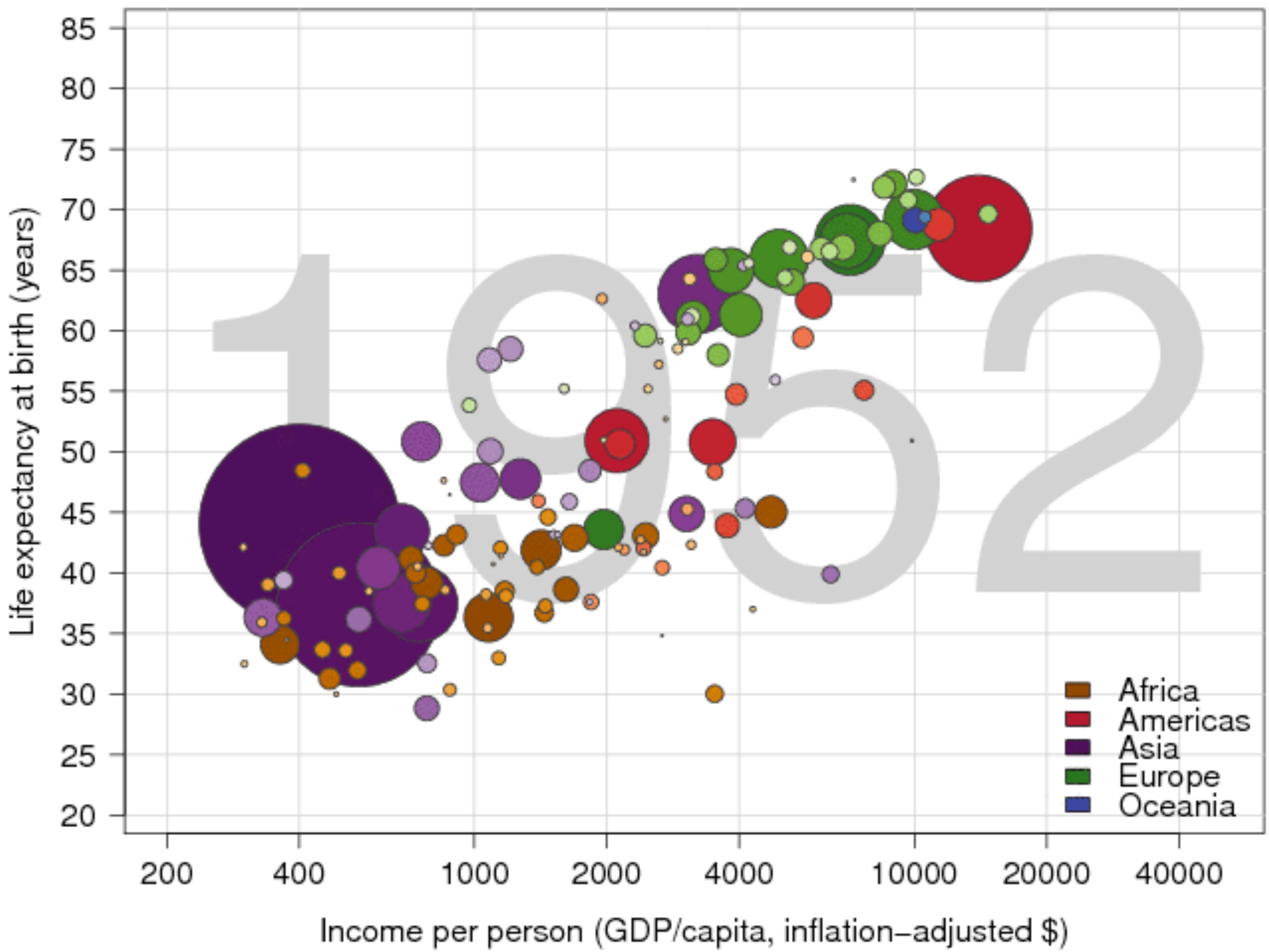
[2] give figure files excruciatingly informative names, not "figure1" or "final version" or "figure for meeting" or "scatterplot"

Your ability to navigate your own work products in the future will be <u>GREATLY</u> enhanced by these practices. I have learned this the hard way.

```
## BEGIN: stitch figures together into an animation

setwd(paste0(whereAmI, "figs/animation/"))
system("convert -delay 100 -loop 0 *.pdf gapminder.gif")
## NOTE: convert is part of ImageMagick
## I view the resulting gif animation with a browser or Xee
## most browsers work and it can also be pasted into Keynote, which
## suggests it might work in PowerPoint too?

## END: stitch figures together into an animation
```

For a final touch, stitch together the year-by-year 'stills' into a dorky animated GIF.

To be clear, I know this is low-tech and has lots of short-comings. But I think it has good hassle:result ratio.


1952
Life expectancy at birth (years)
Income per person (GDP/capita, inflation−adjusted $)
85
80
75
70
65
60
55
50
45
40
35
30
25
20
200
400
1000
2000
4000
10000
20000
40000
Africa
Americas
Asia
Europe
Oceania

Greatest hits of the base R solution

plot(y ~ x, myData, subset = sthgLogical)	axis()	legend()
par()	abline()	
symbols()	text() mtext()	

using colors in R

mostly focused on base/traditional R graphics

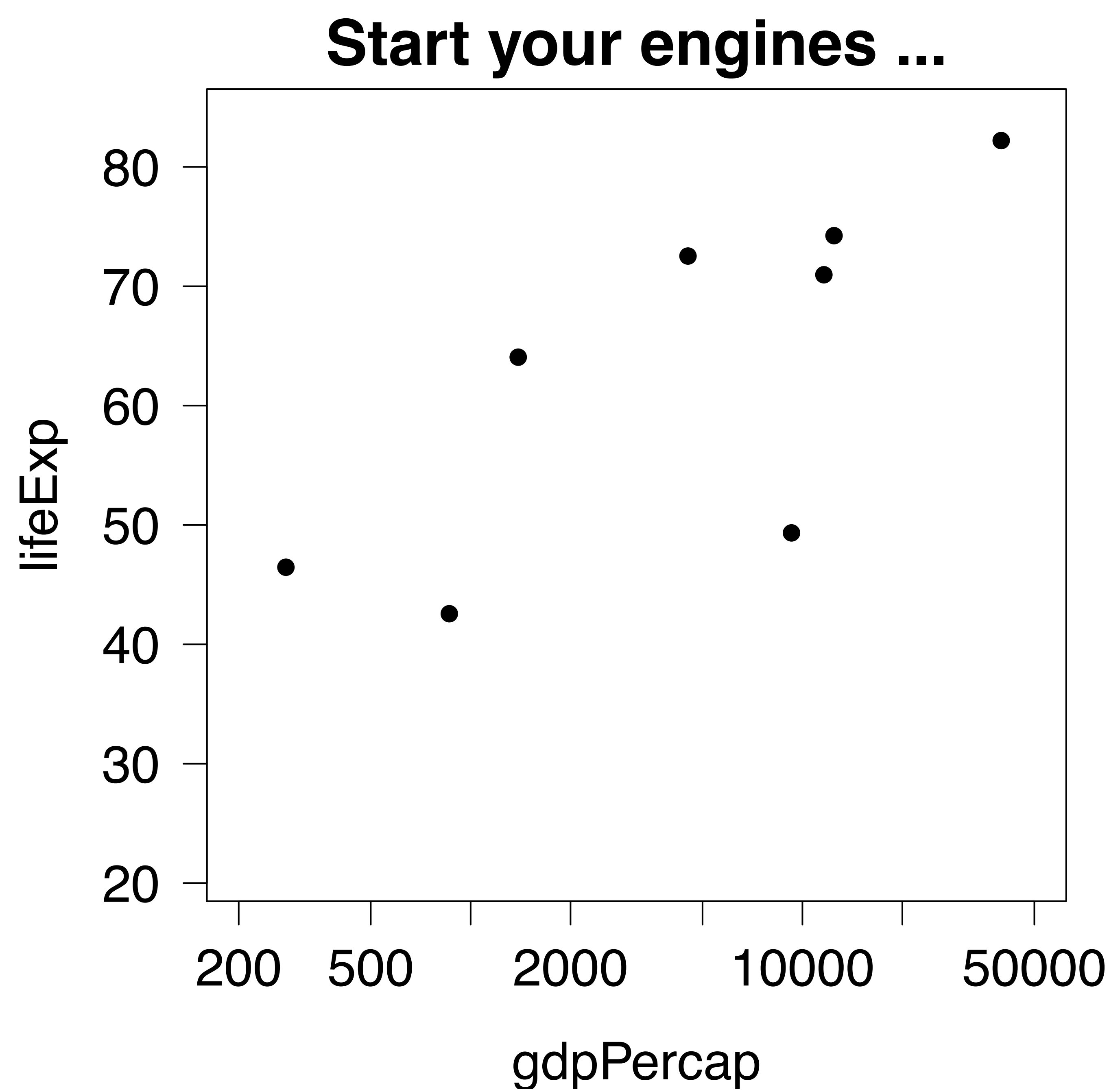
will revisit when we cover lattice

```
> jDat
               country year       pop continent lifeExp  gdpPercap
336  Congo, Dem. Rep. 2007  64606759    Africa  46.462   277.5519
1356     Sierra Leone 2007   6144562    Africa  42.568   862.5408
108        Bangladesh 2007 150448339      Asia  64.062  1391.2538
816            Jordan 2007   6053193      Asia  72.535  4519.4612
1416     South Africa 2007  43997828    Africa  49.339  9269.6578
732              Iran 2007  69453570      Asia  70.964 11605.7145
948          Malaysia 2007  24821286      Asia  74.241 12451.6558
672  Hong Kong, China 2007   6980412      Asia  82.208 39724.9787
```

I randomly drew 8 countries and kept their Gapminder data from 2007.

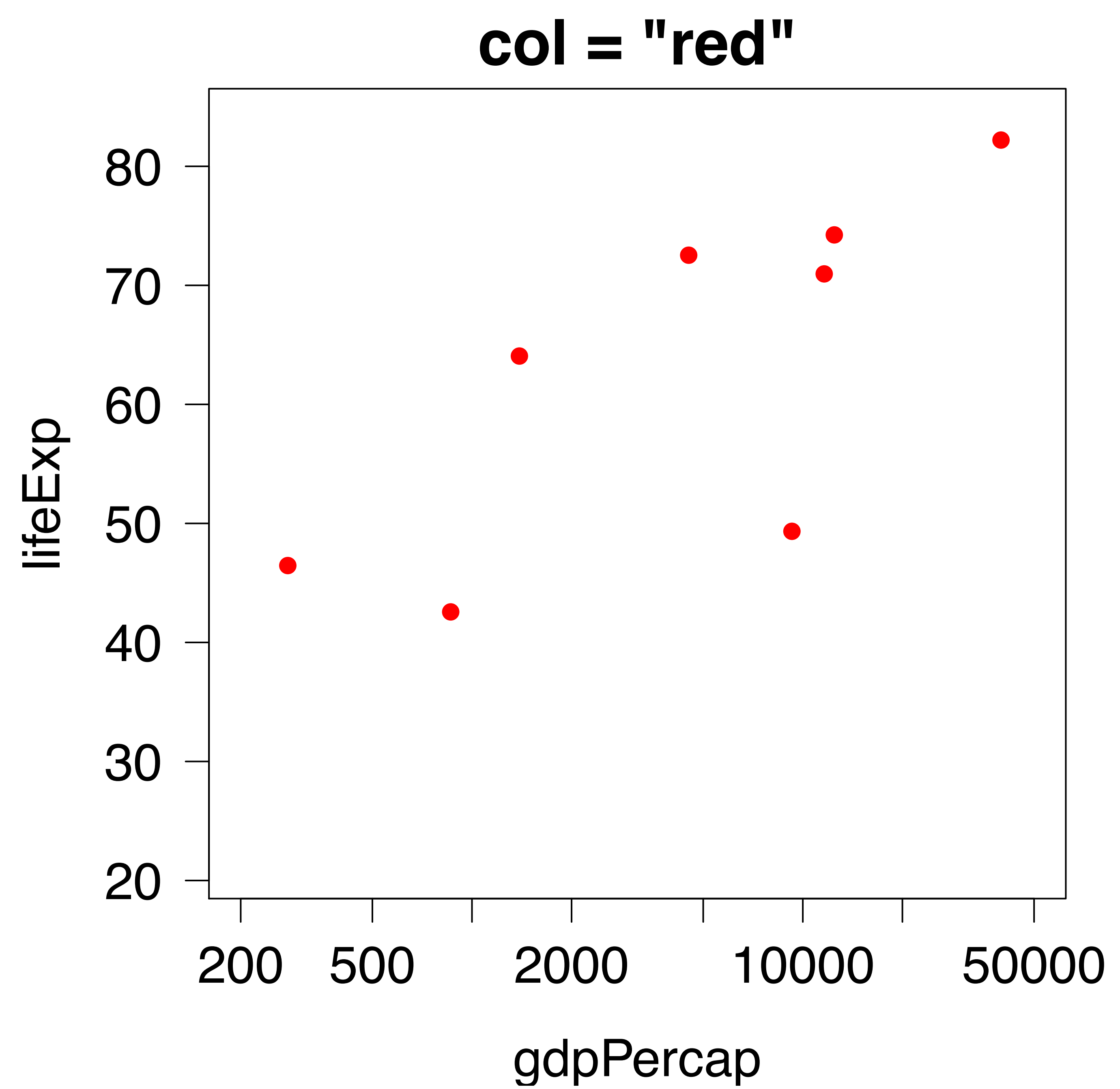
I sorted the rows by gdpPercap, so the points are added to plots from left to right.

```
plot(lifeExp ~ gdpPercap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     main = "Start your engines ...")
```



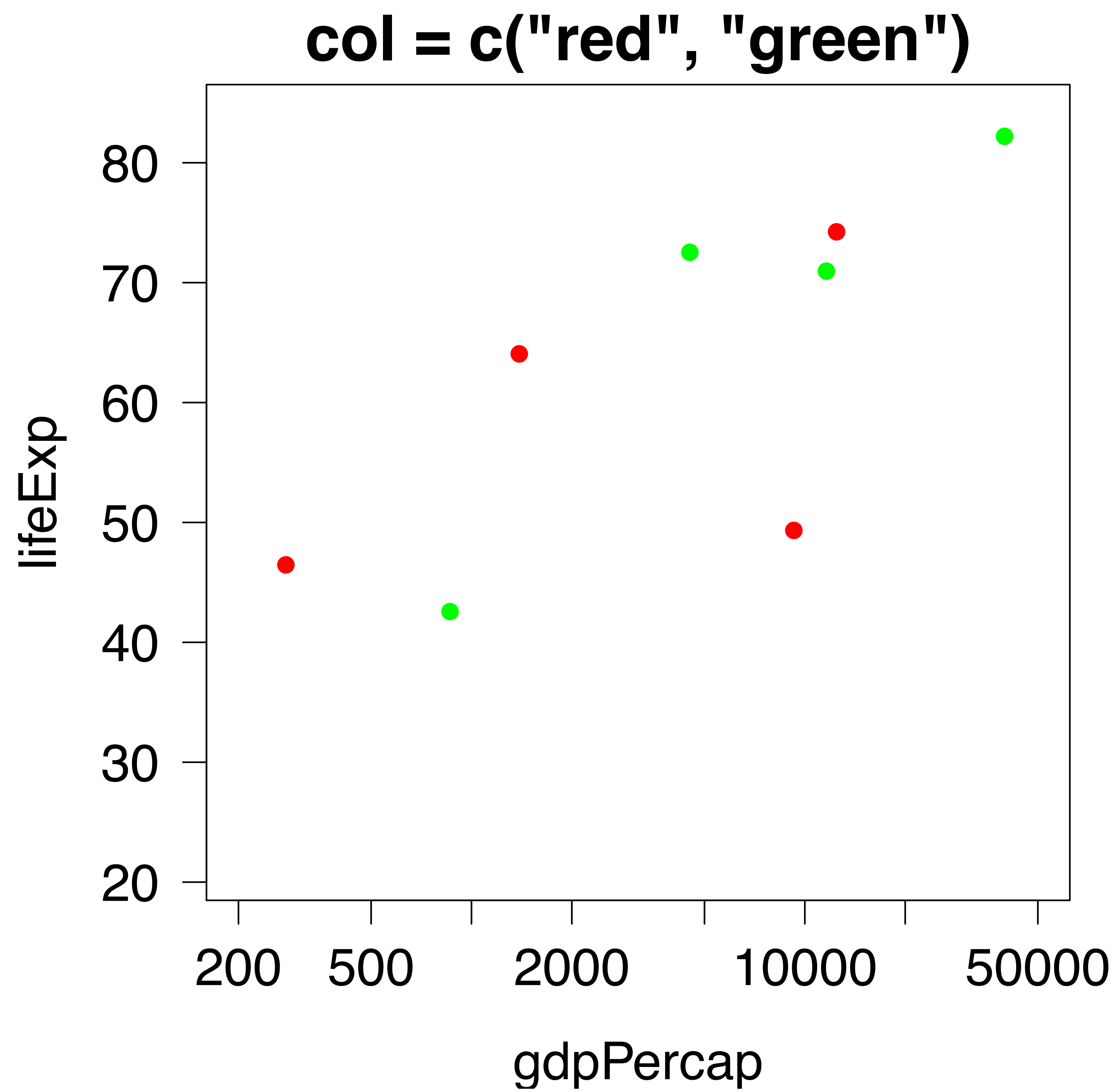
```
plot(lifeExp ~ gdpPercap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = "red", main = 'col = "red"')
```

You can tell R the color you want by name.



```
plot(lifeExp ~ gdpPercap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = c("red", "green"),
     main = 'col = c("red", "green")')
```

Recycling happens.

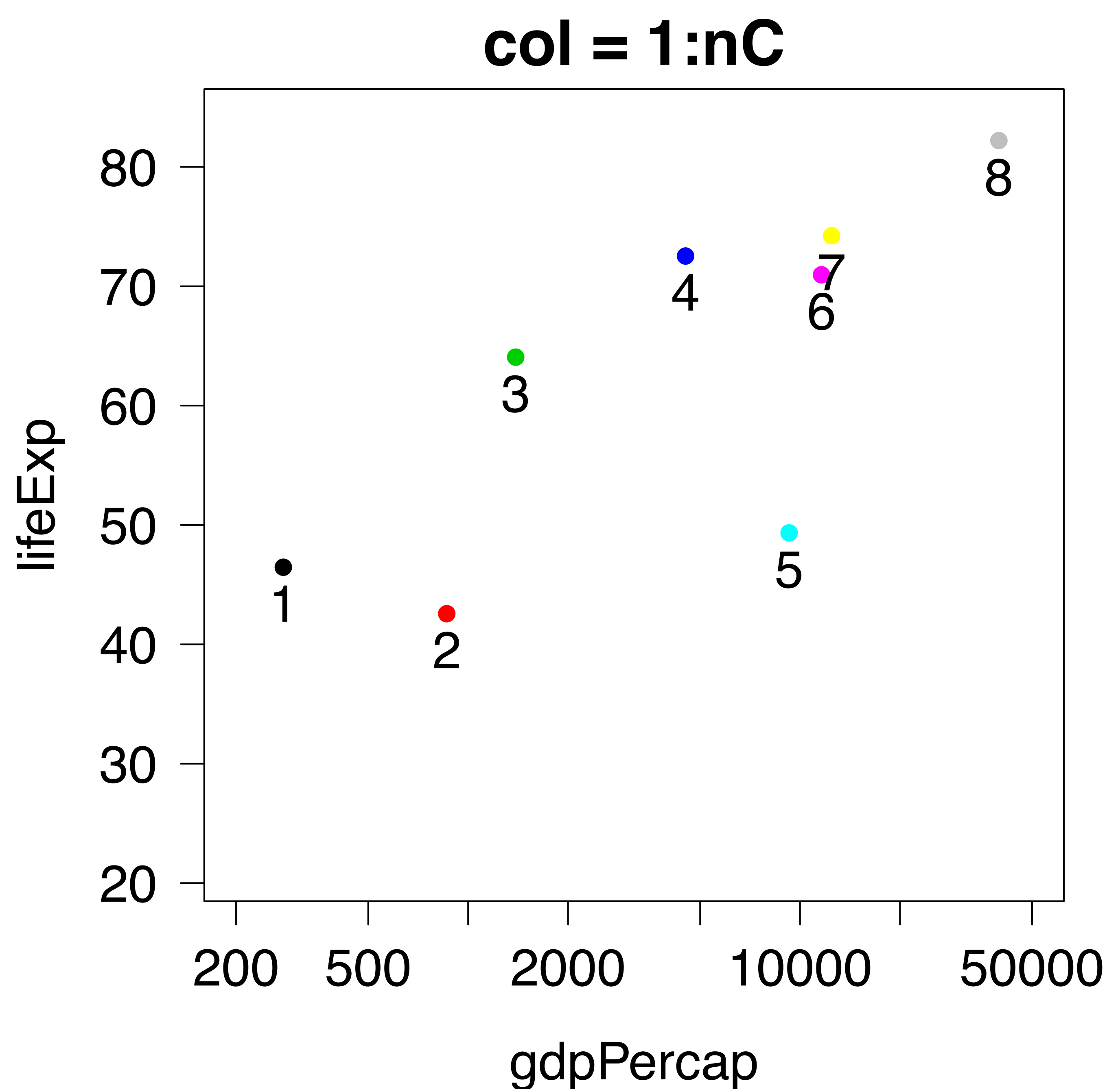


```
plot(lifeExp ~ gdpPercap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = 1:nC,
     main = 'col = 1:nC')
with(jDat,
     text(x = gdpPercap, y = lifeExp, pos = 1))
```

You can specify a color via an integer.

This specifies colors within the current palette.

You're looking at the default palette.

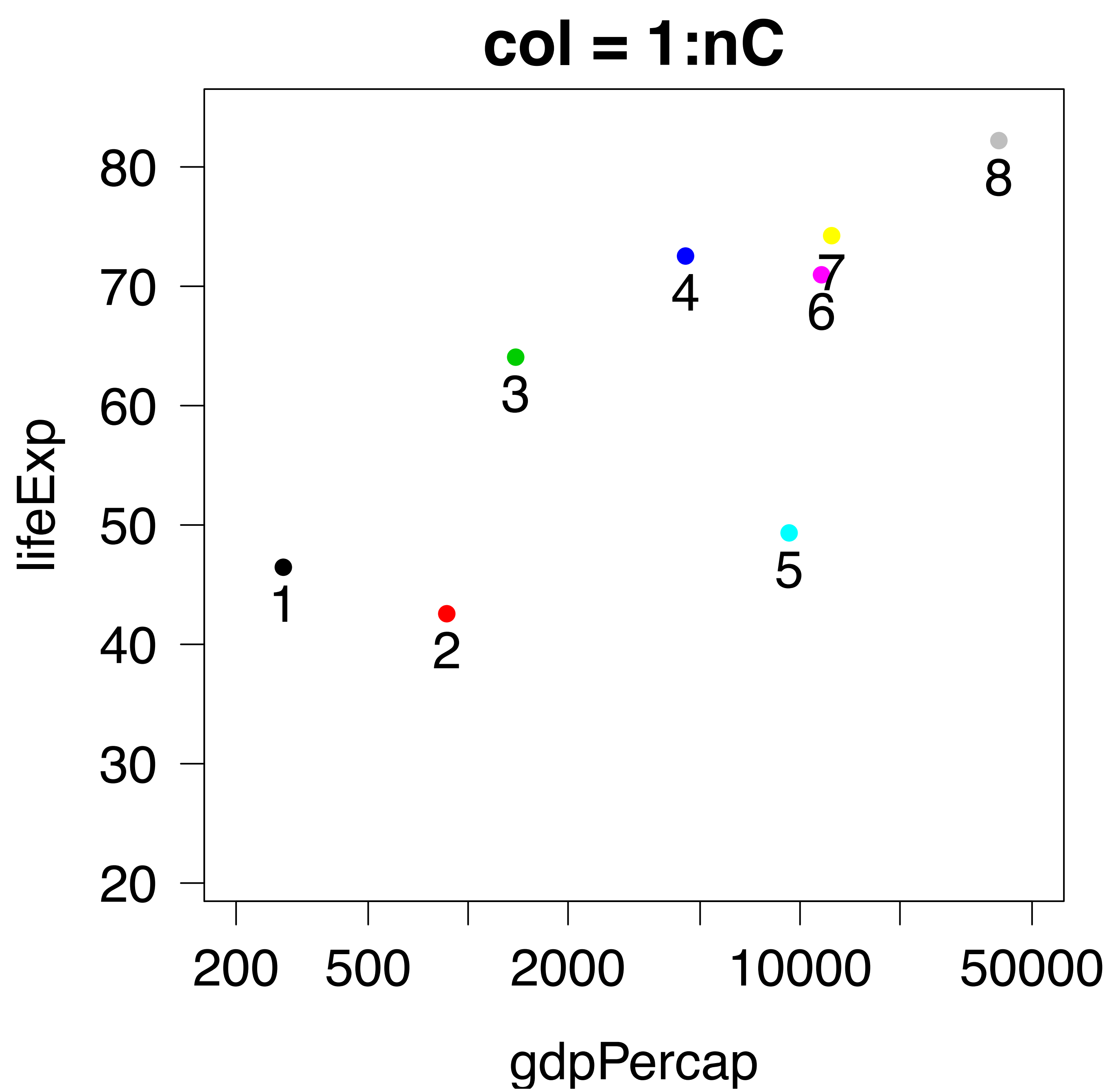


```
> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

View and modify the palette with palette().

Read documentation to see examples of changing the active palette.

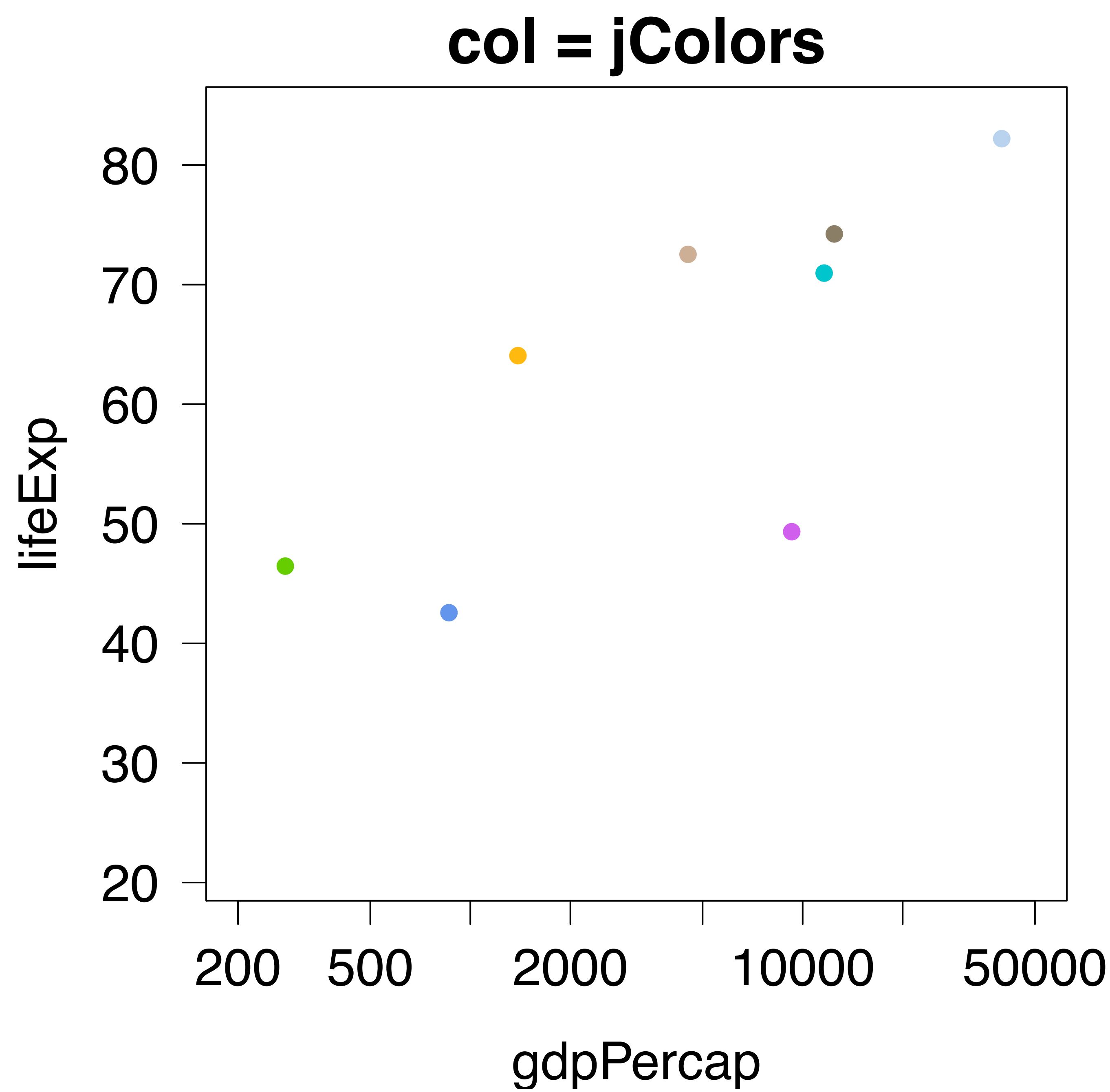
The default palette is ugly.



```r
jColors <- c('chartreuse3', 'cornflowerblue',
             'darkgoldenrod1', 'peachpuff3',
             'mediumorchid2', 'turquoise3',
             'wheat4', 'slategray2')
plot(lifeExp ~ gdpPercap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = jColors,
     main = 'col = jColors')
```

Express your inner artist!

Save the colors you plan to use to an R object, then pass to graphing functions.



```
> colors()
  [1] "white"                "aliceblue"            "antiquewhite"
  [4] "antiquewhite1"        "antiquewhite2"        "antiquewhite3"
  [7] "antiquewhite4"        "aquamarine"           "aquamarine1"
 [10] "aquamarine2"          "aquamarine3"          "aquamarine4"

<snip, snip>

[643] "violetred2"           "violetred3"           "violetred4"
[646] "wheat"                "wheat1"               "wheat2"
[649] "wheat3"               "wheat4"               "whitesmoke"
[652] "yellow"               "yellow1"              "yellow2"
[655] "yellow3"              "yellow4"              "yellowgreen"
```

colors() will show you the 657 colors you can refer to by name.

Page 1 out of 6

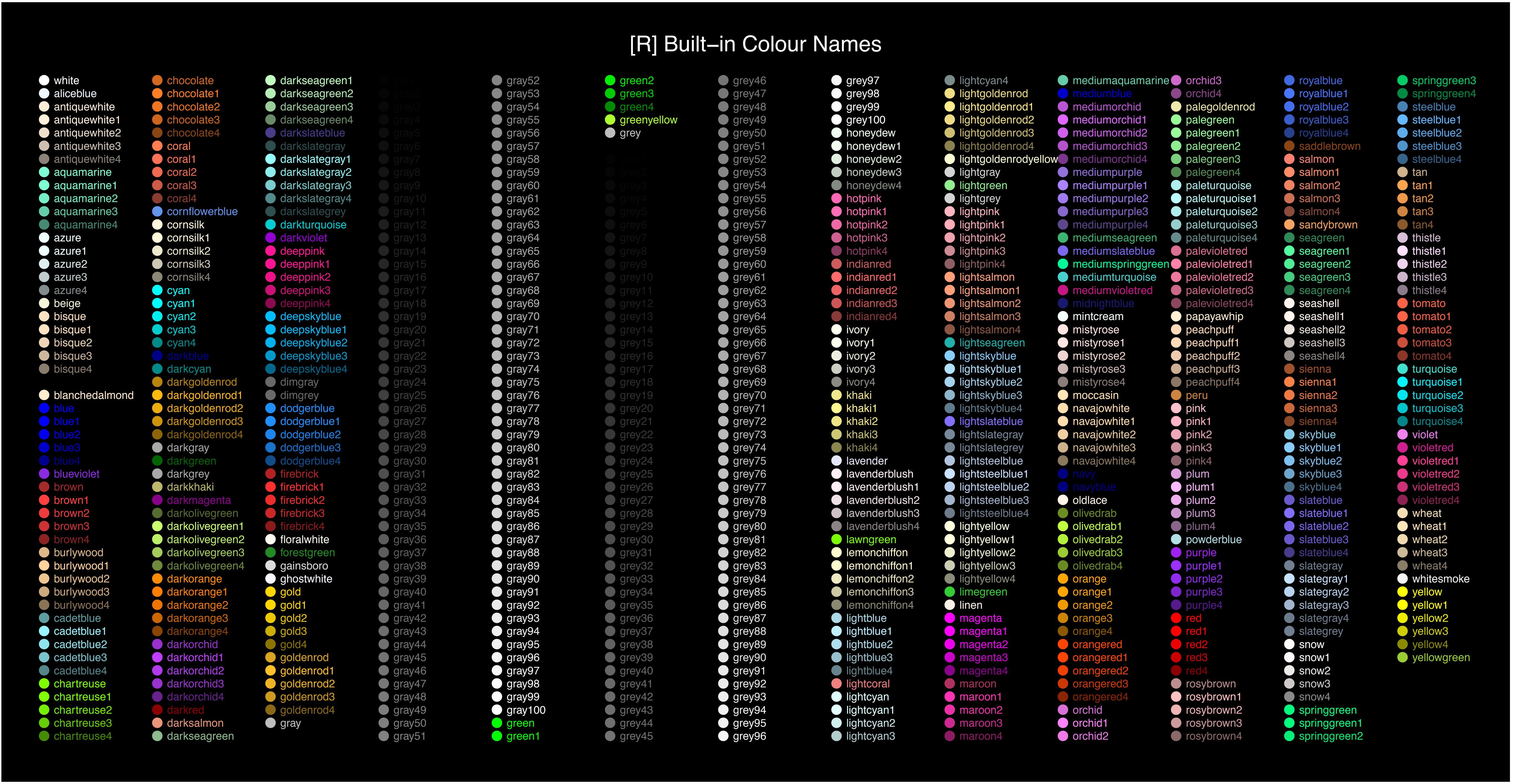
	azure2	blue3	cadetblue1	coral	cyan3	darkolivegreen	darkorchid4
aliceblue	azure3	blue4	cadetblue2	coral1	cyan4	darkolivegreen1	darkred
antiquewhite	azure4	blueviolet	cadetblue3	coral2	darkblue	darkolivegreen2	darksalmon
antiquewhite1	beige	brown	cadetblue4	coral3	darkcyan	darkolivegreen3	darkseagreen
antiquewhite2	bisque	brown1	chartreuse	coral4	darkgoldenrod	darkolivegreen4	darkseagreen1
antiquewhite3	bisque1	brown2	chartreuse1	cornflowerblue	darkgoldenrod1	darkorange	darkseagreen2
antiquewhite4	bisque2	brown3	chartreuse2	cornsilk	darkgoldenrod2	darkorange1	darkseagreen3
aquamarine	bisque3	brown4	chartreuse3	cornsilk1	darkgoldenrod3	darkorange2	darkseagreen4
aquamarine1	bisque4	burlywood	chartreuse4	cornsilk2	darkgoldenrod4	darkorange3	darkslateblue
aquamarine2	black	burlywood1	chocolate	cornsilk3	darkgray	darkorange4	darkslategray
aquamarine3	blanchedalmond	burlywood2	chocolate1	cornsilk4	darkgreen	darkorchid	darkslategray1
aquamarine4	blue	burlywood3	chocolate2	cyan	darkgrey	darkorchid1	darkslategray2
azure	blue1	burlywood4	chocolate3	cyan1	darkkhaki	darkorchid2	darkslategray3
azure1	blue2	cadetblue	chocolate4	cyan2	darkmagenta	darkorchid3	darkslategray4

A long time ago I made a 6 page document for myself. Good times.

Page 1 out of 6

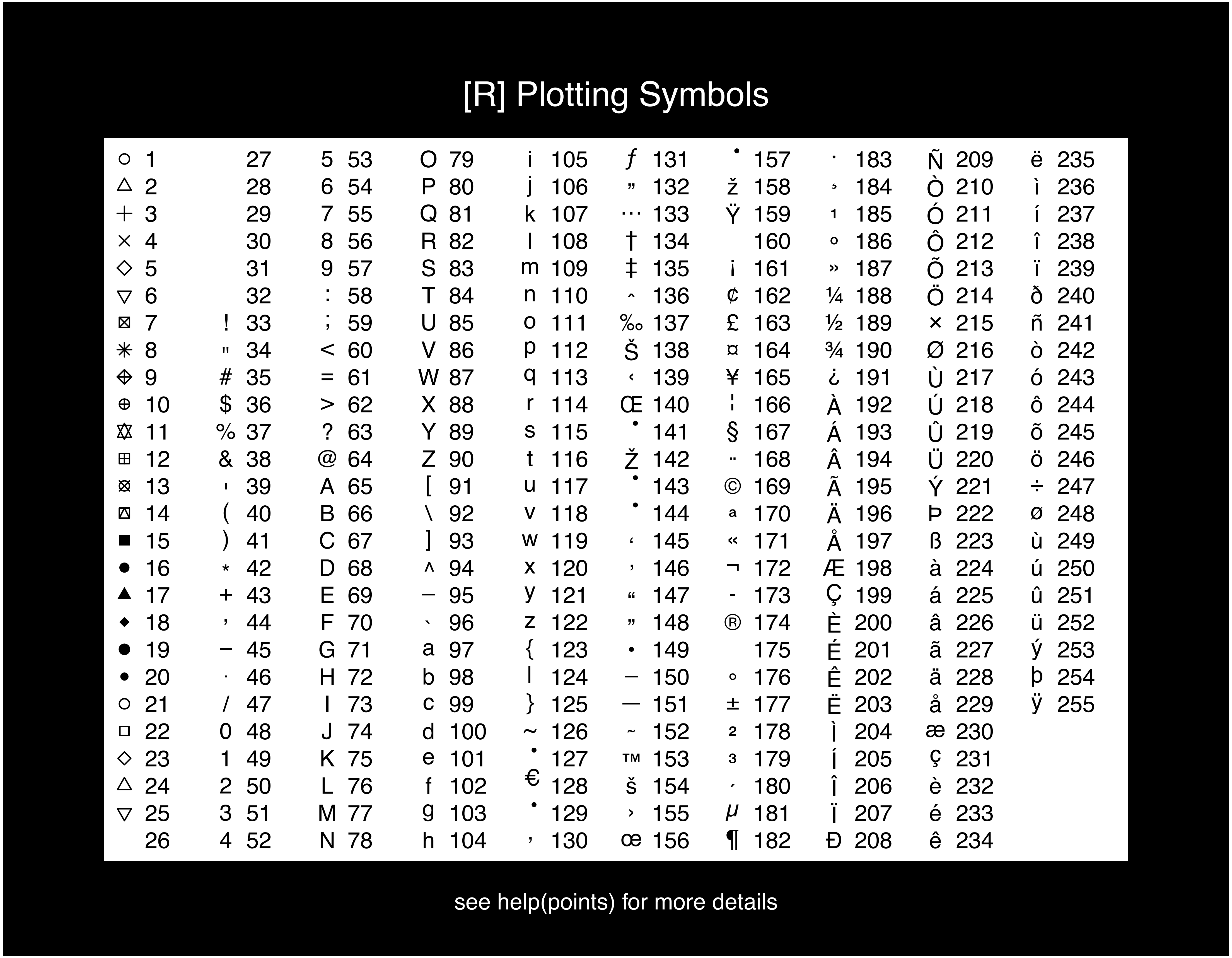
white	azure2	blue3	cadetblue1	coral	cyan3	darkolivegreen	darkorchid4
aliceblue	azure3	blue4	cadetblue2	coral1	cyan4	darkolivegreen1	darkred
antiquewhite	azure4	blueviolet	cadetblue3	coral2	darkblue	darkolivegreen2	darksalmon
antiquewhite1	beige	brown	cadetblue4	coral3	darkcyan	darkolivegreen3	darkseagreen
antiquewhite2	bisque	brown1	chartreuse	coral4	darkgoldenrod	darkolivegreen4	darkseagreen1
antiquewhite3	bisque1	brown2	chartreuse1	cornflowerblue	darkgoldenrod1	darkorange	darkseagreen2
antiquewhite4	bisque2	brown3	chartreuse2	cornsilk	darkgoldenrod2	darkorange1	darkseagreen3
aquamarine	bisque3	brown4	chartreuse3	cornsilk1	darkgoldenrod3	darkorange2	darkseagreen4
aquamarine1	bisque4	burlywood	chartreuse4	cornsilk2	darkgoldenrod4	darkorange3	darkslateblue
aquamarine2		burlywood1	chocolate	cornsilk3	darkgray	darkorange4	darkslategray
aquamarine3	blanchedalmond	burlywood2	chocolate1	cornsilk4	darkgreen	darkorchid	darkslategray1
aquamarine4	blue	burlywood3	chocolate2	cyan	darkgrey	darkorchid1	darkslategray2
azure	blue1	burlywood4	chocolate3	cyan1	darkkhaki	darkorchid2	darkslategray3
azure1	blue2	cadetblue	chocolate4	cyan2	darkmagenta	darkorchid3	darkslategray4

On a black background too, just in case!

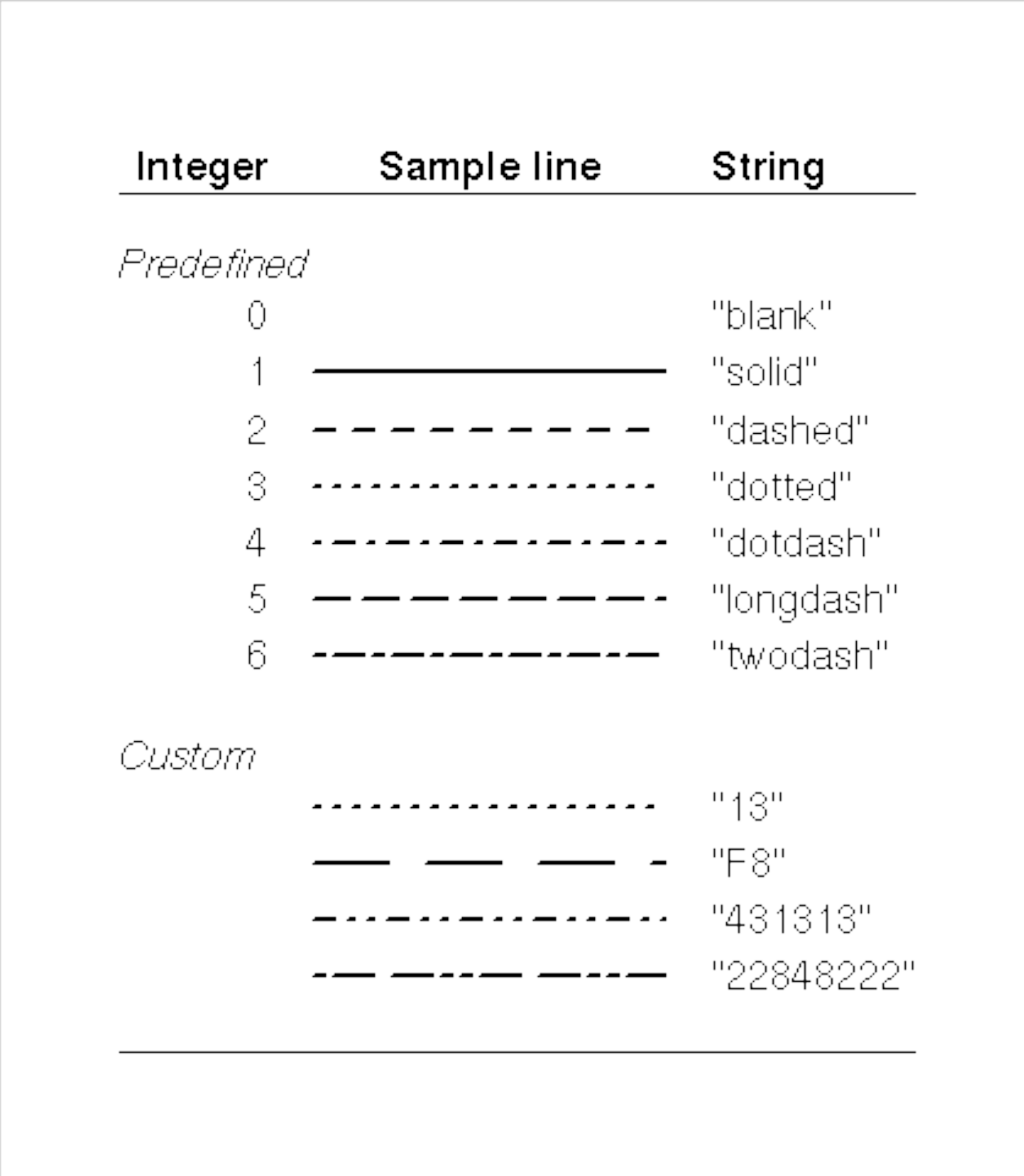


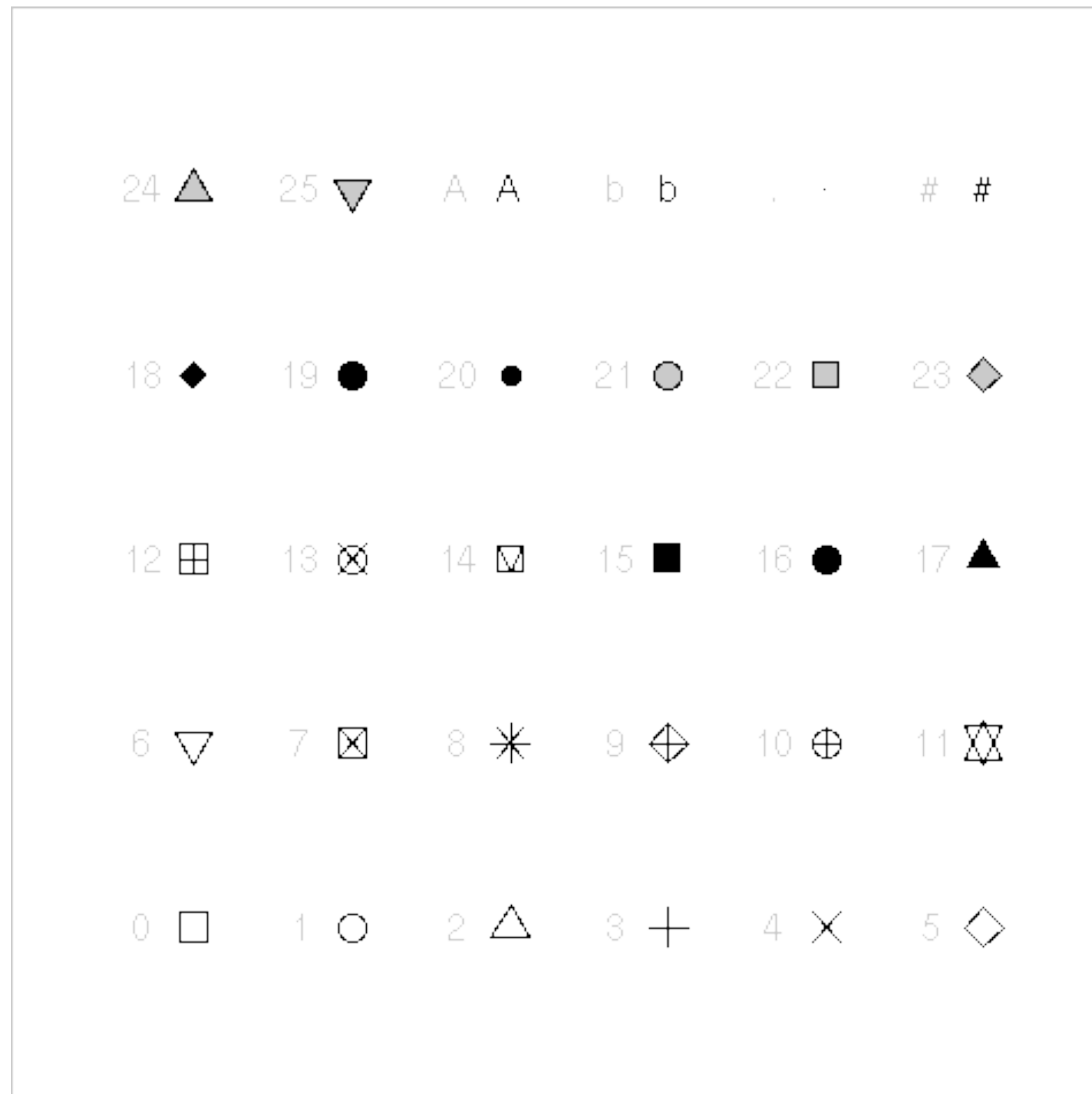
created by a STAT 545A student in past

you can also find lots of these on the interwebs


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

symbols, too ....





From Ch.3 of
Murrell 'R Graphics'

Temperature (°C) in 2003

```
expression(paste("Temperature (", degree, "C) in 2003"))
```

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

```
expression(bar(x) == sum(frac(x[i], n), i==1, n))
```

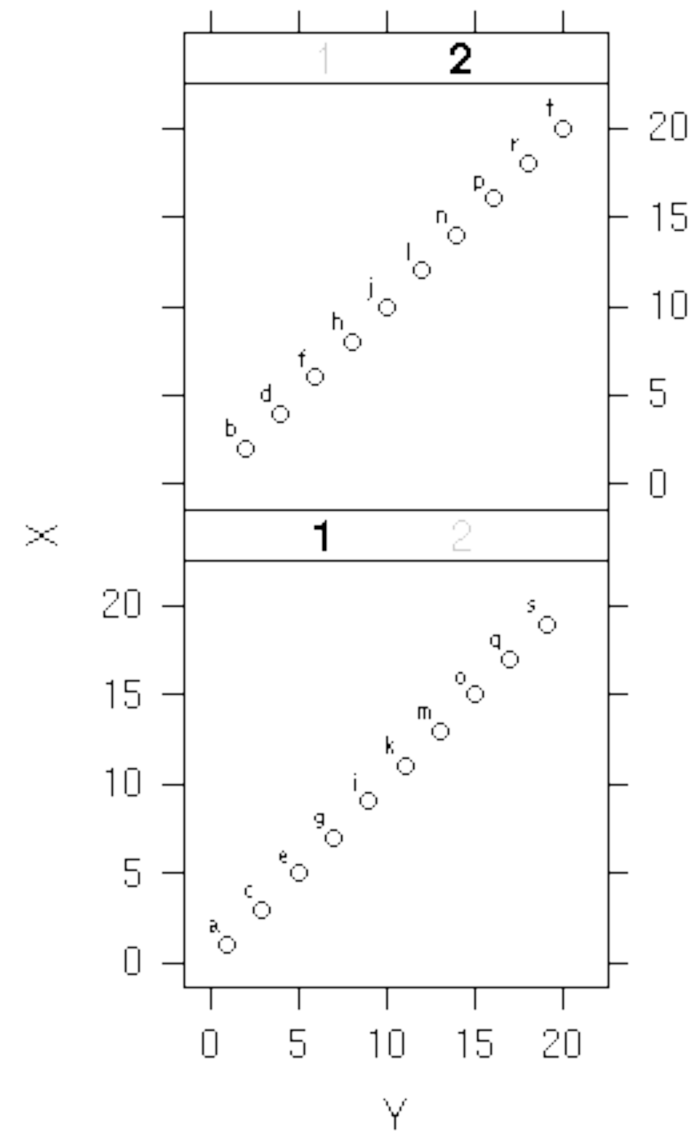
$$\hat{\beta} = (X^t X)^{-1} X^t y$$

```
expression(hat(beta) == (X^t * X)^{-1} * X^t * y)
```

$$z_i = \sqrt{x_i^2 + y_i^2}$$

```
expression(z[i] == sqrt(x[i]^2 + y[i]^2))
```

From <u>Ch.3</u> of
Murrell ‘<u>R Graphics</u>’



From <u>Ch. 4</u> of
Murrell ‘<u>R Graphics</u>’

Honestly, hand-picking colors is not sustainable.

Time-consuming.

Most of us are actually terrible at it.

Trust a professional.

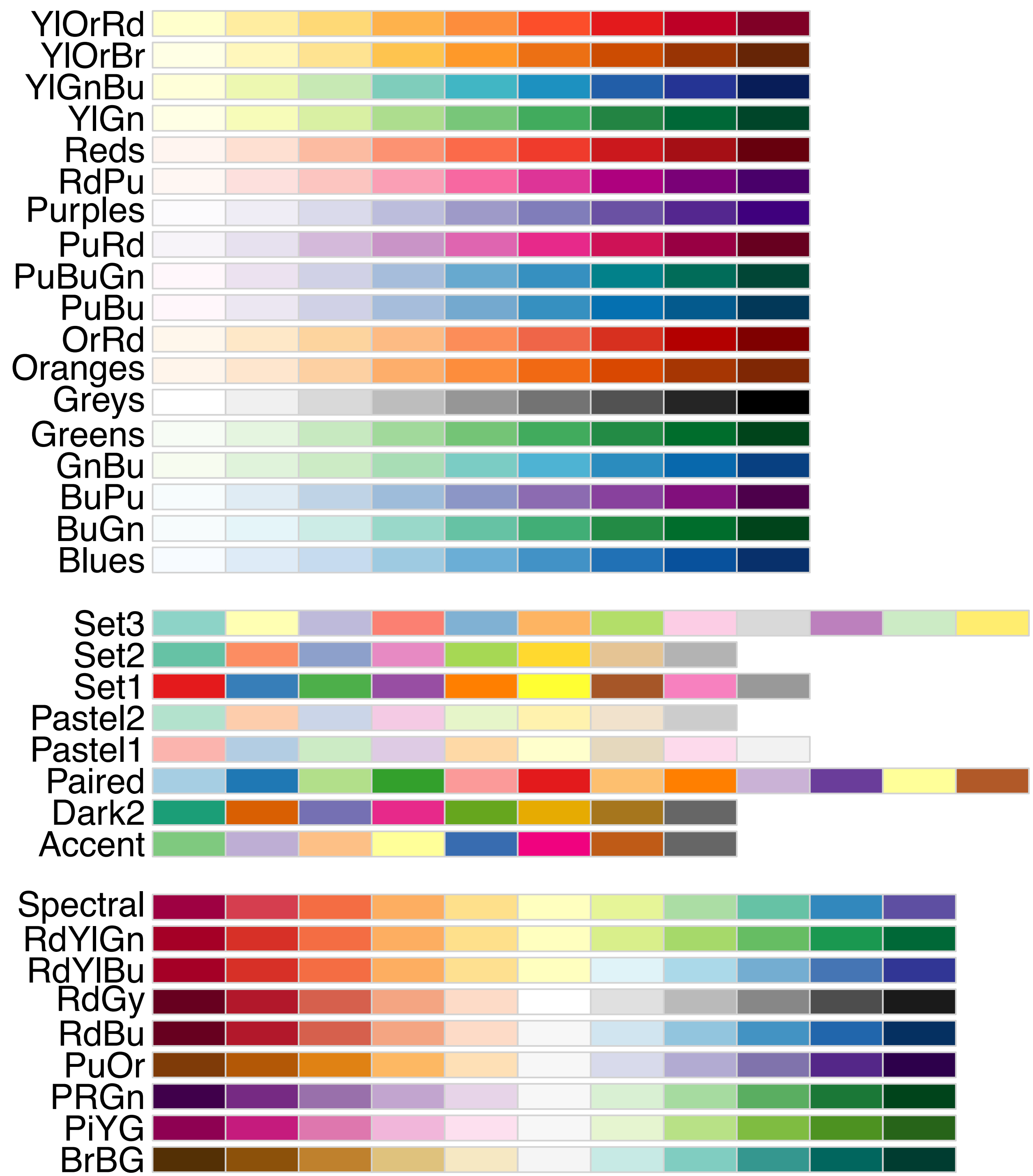
Consider the RColorBrewer package, based on the work of Cynthia Brewer.

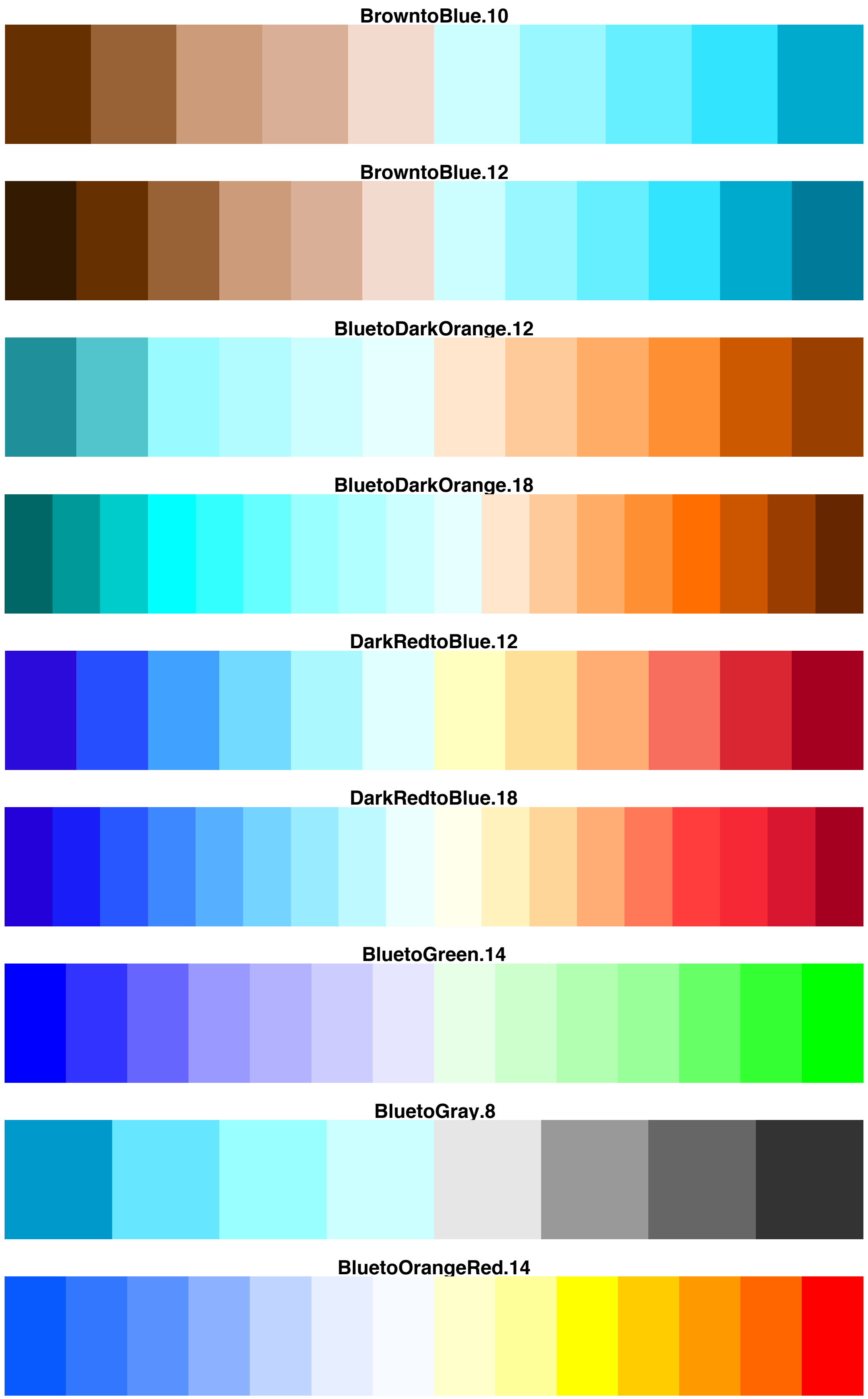
```
library(RColorBrewer)
display.brewer.all()
```

sequential

qualitative

diverging





Another source of color palettes suitable for colorblind people is the package `dichromat`

```
library(RColorBrewer)
display.brewer.pal(n = 8, name = 'Dark2')
```

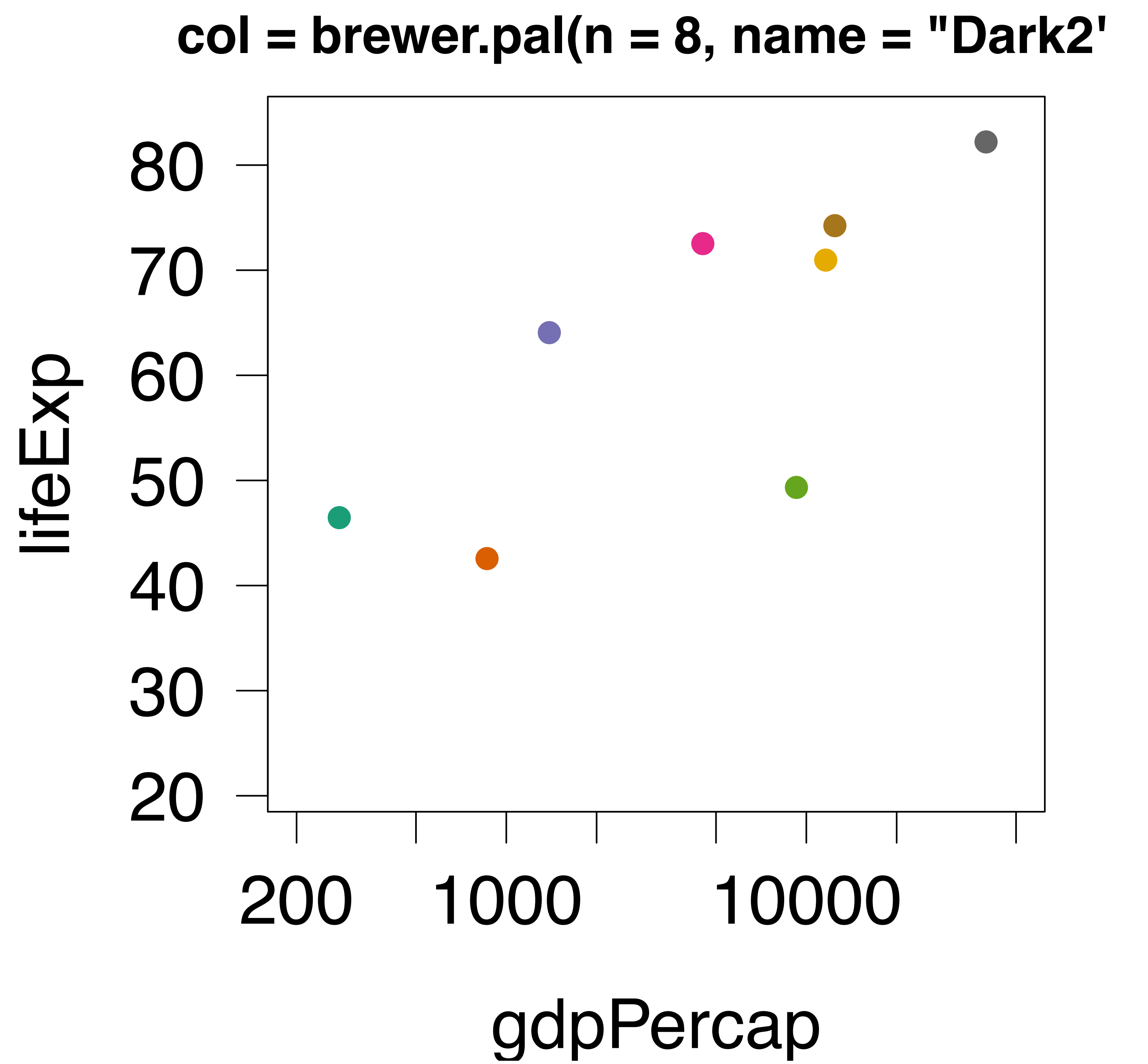
Dark2 (qualitative)

Focusing in on one of the qualitative palettes ....

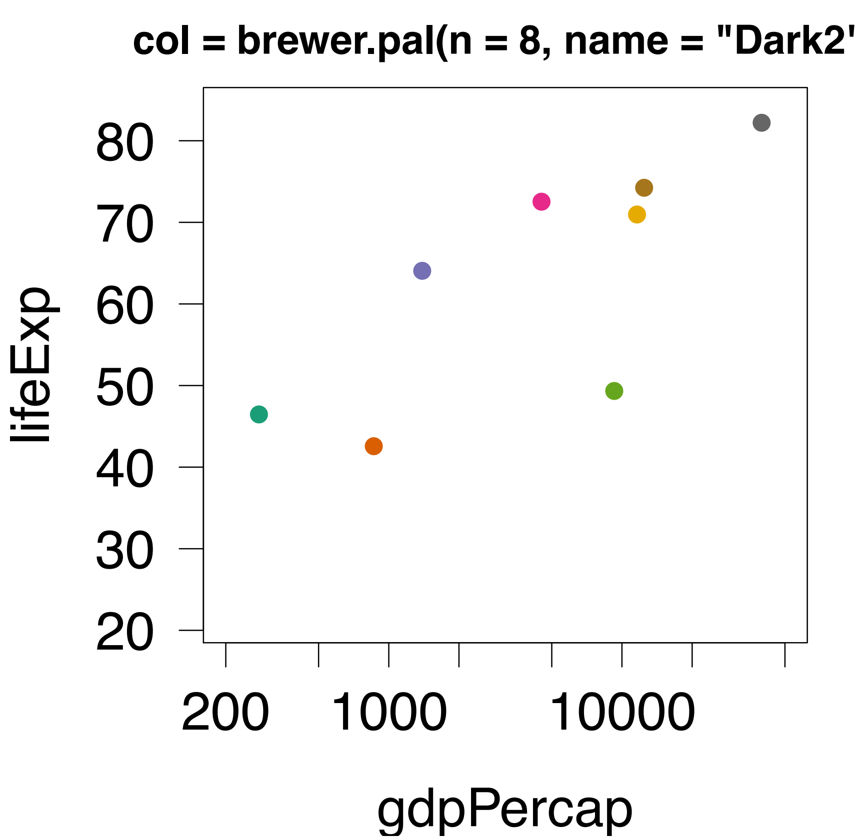
```
plot(lifeExp ~ gdpPercap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = brewer.pal(n = 8, name = "Dark2"),
     main = 'col = brewer.pal(n = 8, name = "Dark2")',
     cex.main = 0.75)
```

RColorBrewer-based color choices are more sustainable, higher quality than built-in or self-made color schemes.

But I still recommend storing the scheme as an object ....



```
> (jColors <- brewer.pal(n = 8, name = "Dark2"))
[1] "#1B9E77" "#D95F02" "#7570B3" "#E7298A" "#66A61E" "#E6AB02" "#A6761D"
[8] "#666666"
> plot(lifeExp ~ gdpPercap, jDat, log = 'x',
+      xlim = jXlim, ylim = jYlim,
+      col = jColors,
+      main = 'col = brewer.pal(n = 8, name = "Dark2")',
+      cex.main = 0.75)
```



Notice the form in which the RColorBrewer colors are stored.

Let's demystify that ....

```
> (jColors <- brewer.pal(n = 8, name = "Dark2"))
[1] "#1B9E77" "#D95F02" "#7570B3" "#E7298A" "#66A61E" "#E6AB02" "#A6761D"
[8] "#666666"
```

hex	decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

These colors are expressed as Red-Blue-Green (RBG) hexadecimal triples.

Parse like so: #rrbbgg.

Each element -- such as the ‘rr’ -- specifies the intensity of a color component as a two digit base 16 number.

How to interpret a hexadecimal value ....
$9E = 9 * 16^1 + 14 * 16^0 = 9 * 16 + 14 = 158$

Lowest value is 00 = 0.
Highest values is FF = 255.

Some basic facts re: RBG hexadecimal triples.

hex	decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

“unsaturated”, shades of gray

color name	#rrggbb	red	green	blue
white	#FFFFFF	255	255	255
gray50	#7F7F7F	127	127	127
black	#000000	0	0	0

“saturated”, primary colors

color name	#rrggbb	red	green	blue
blue	#0000FF	0	0	255
green	#00FF00	0	255	0
red	#FF0000	255	0	0

R is expecting colors to be specified in one of these ways:
- an integer, used as an index into current palette
- a character string, i.e. one of the color names in colors()
- a hexadecimal RGB triple

Under the hood, colors are always expressed in one of several color models or color spaces. RGB is just one example. Another is Hue-Saturation-Value (HSV).

Turns out RGB is a rather lousy color model (arguably, so it HSV). Good for generating colors on a computer screen but doesn't facilitate color picking with respect to human perception.

Zeileis et al advocate using Hue-Chroma-Luminance (HCL) triplets. "Less flashy (than HSV) and more perceptually balanced." Check out their interesting paper and the colorspace R package.

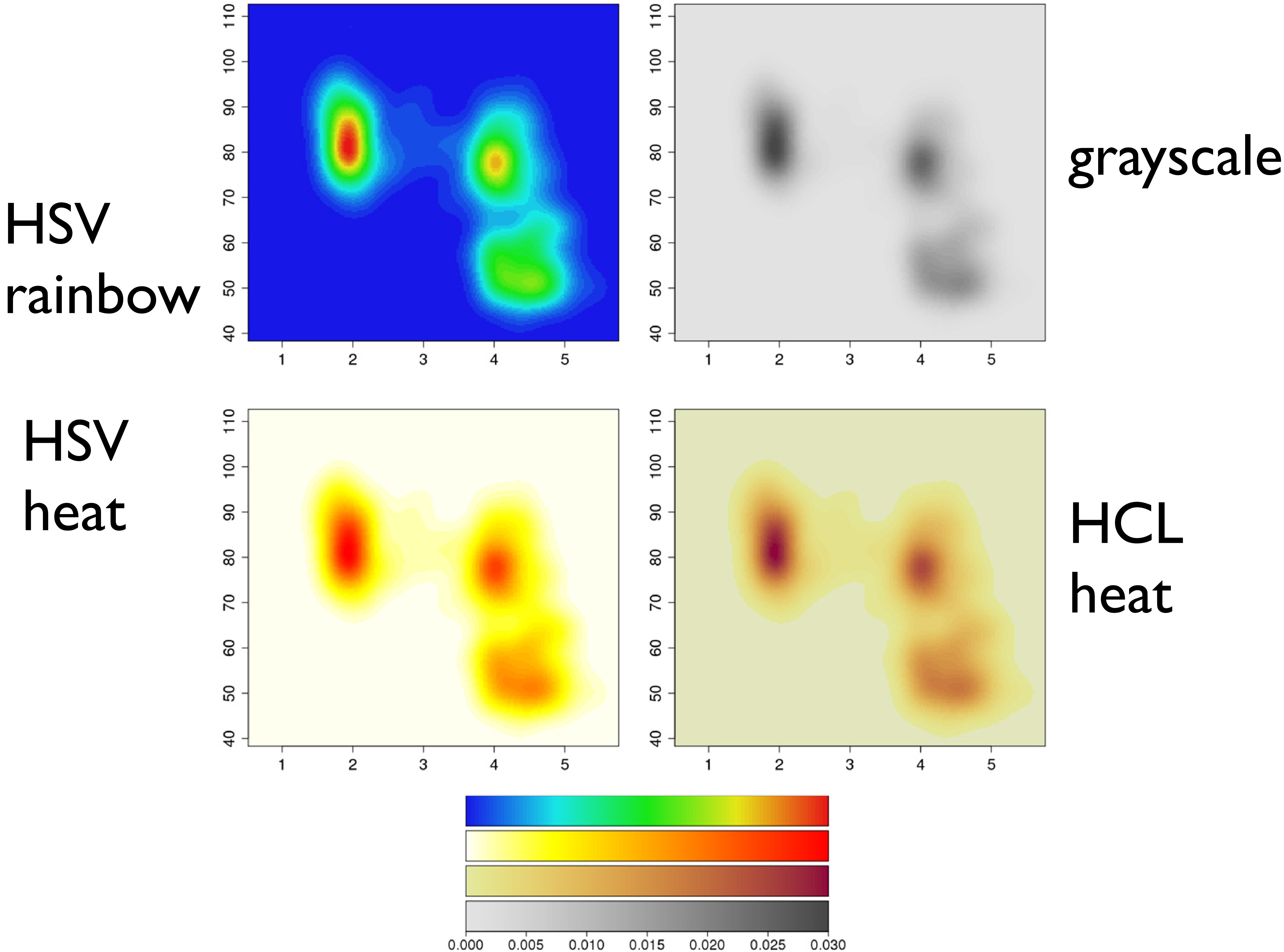


Fig. 1. Bivariate density estimation of duration (*x*-axis) and waiting time (*y*-axis) for Old Faithful geyser eruptions. The palettes employed are (counterclockwise from top left) an HSV-based rainbow, HSV-based heat colors, HCL-based heat colors and grayscales.

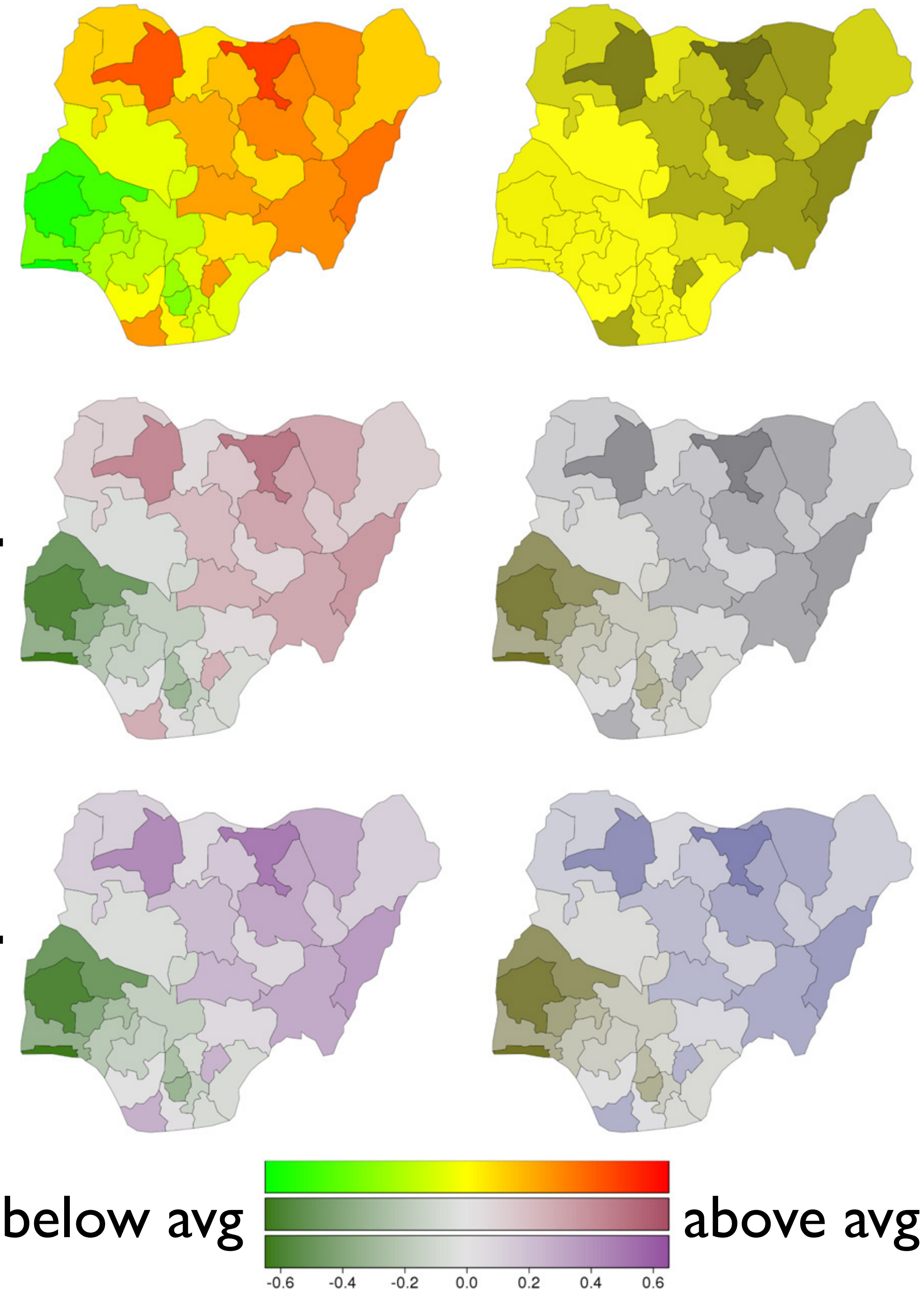
childhood mortality in Nigeria

original HSV palette

proposed HCL palette #1

proposed HCL palette #2

what a red-green colorblind person would see



below avg

above avg

Fig. 2. Posterior mode estimates for childhood mortality in Nigeria. The color palettes employed are (from top to bottom) an HSV-based rainbow and two HCL-based diverging palettes. In the right panels red–green contrasts are collapsed to emulate protanopic vision.

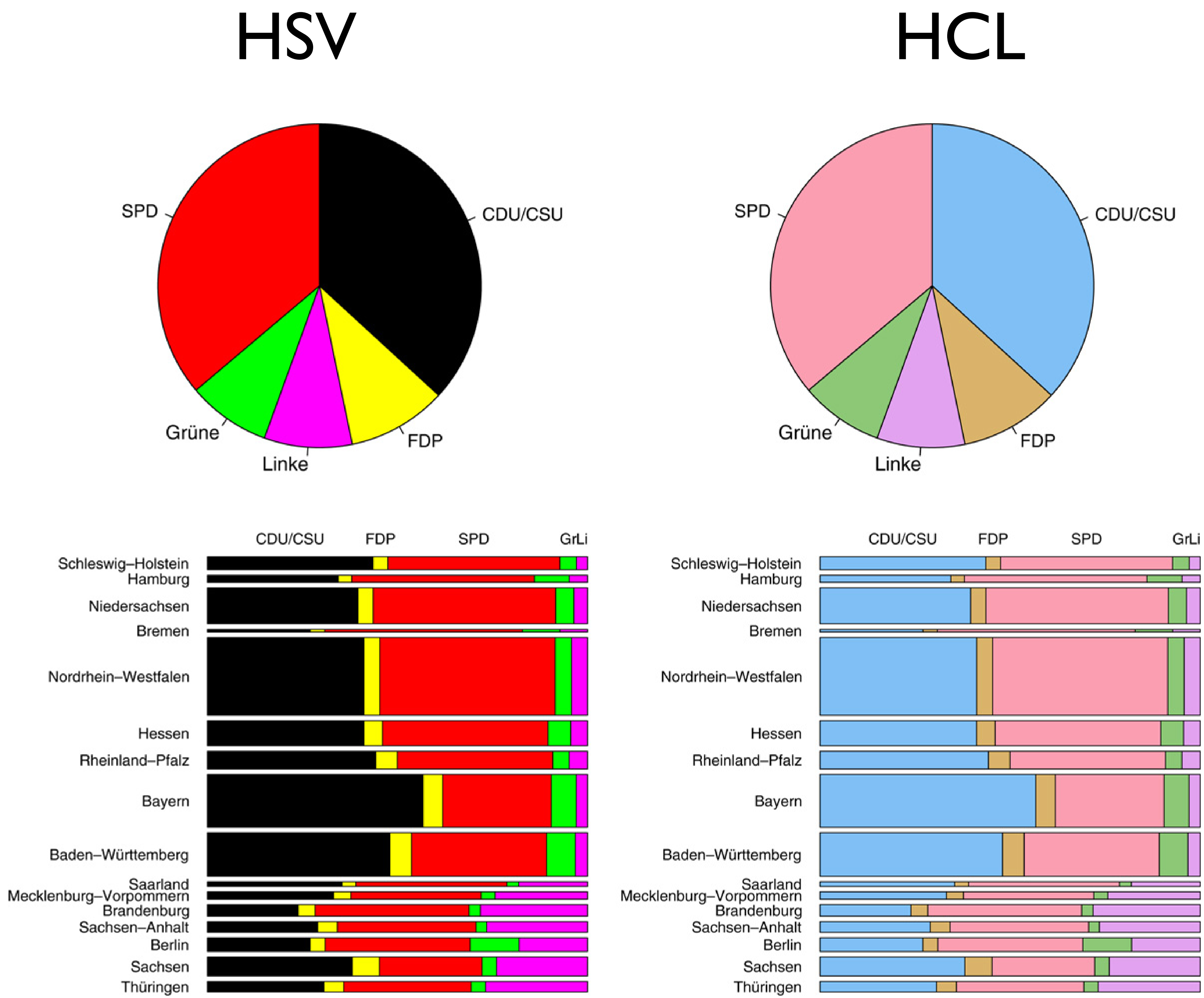


Fig. 7. German election 2005 with HSV-based (left) and HCL-based (right) qualitative palette. Top: Pie chart for seats in the parliament. Bottom: Mosaic display for votes by province.

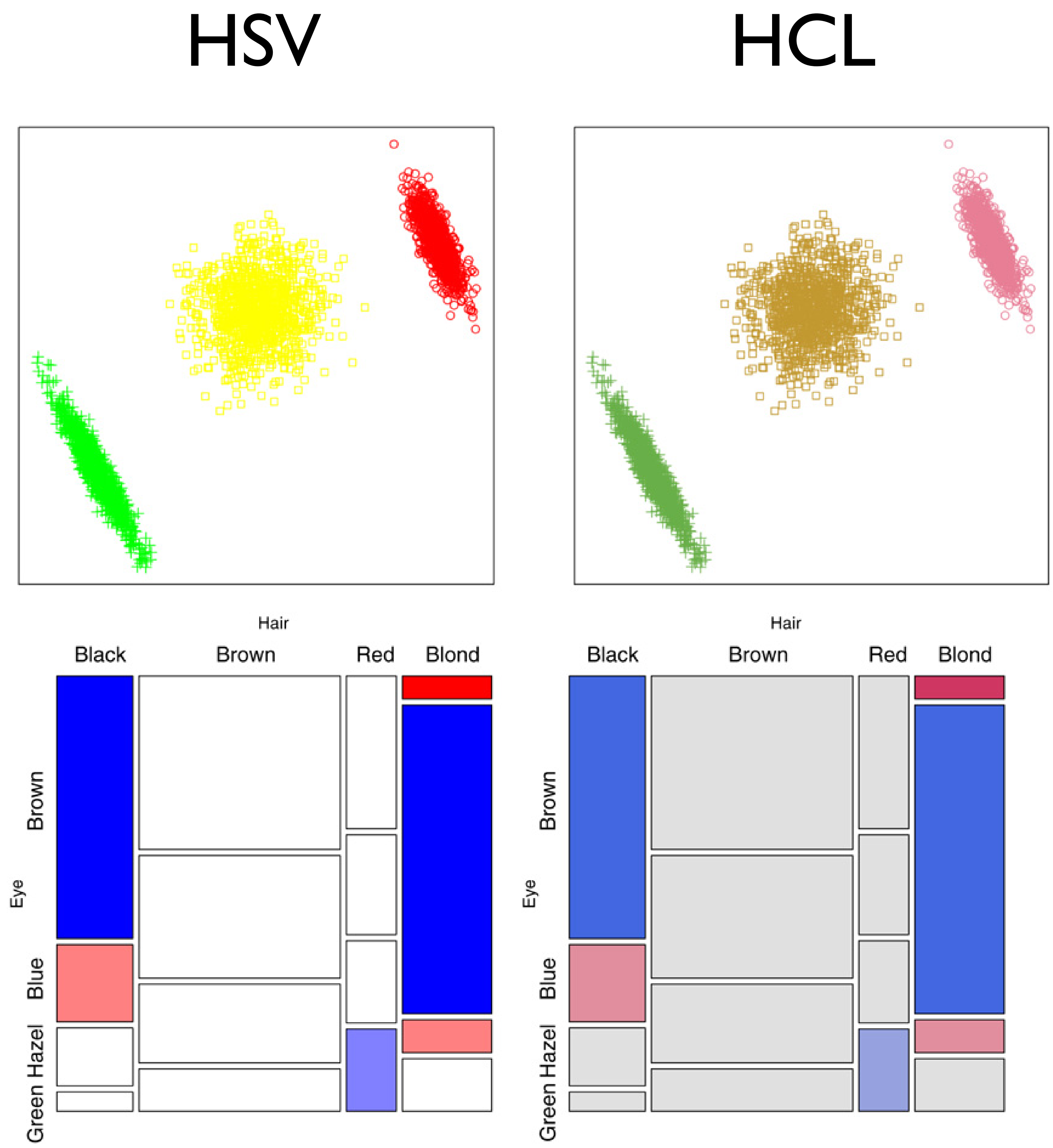


Fig. 8. Further examples for HSV-based (left) and HCL-based (right) palettes. Top: Scatter plot with three clusters and qualitative palette. Bottom: Extended mosaic display for hair and eye color data with diverging palette.

Zeileis, A., Hornik, K., & Murrell, P. (2009). Escaping RGBland: Selecting colors for statistical graphics. Computational Statistics & Data Analysis, 53(9), 3259–3270. doi:10.1016/j.csda.2008.11.033

The R system for statistical computing (R Development Core Team, 2008) provides an open-source implementation of HCL (and other color spaces) in the package **colorspace**, originally written by Ross Ihaka. The coordinate transformations mentioned above are contained in C code within **colorspace** that are easy to port to other statistical software systems. Version 1.0-0 of **colorspace** (Ihaka et al., 2008) also includes an implementation of all palettes discussed above. (Originally, the code for the palettes was in the **vcd** package, Meyer et al. (2006) but it was recently moved to **colorspace** to be more easily acessible.) Qualitative palettes are provided by `rainbow_hcl()` (named after the HSV-based function `rainbow()` in base R). Sequential palettes based on a single hue are implemented in the function `sequential_hcl()` while `heat_hcl()` offers sequential palettes based on a range of hues. Diverging palettes can be obtained by `diverge_hcl()`. Technical documentation along with a large collection of example palettes is available via `help("rainbow_hcl", package = "colorspace")`. Furthermore, R code for reproducing the example palettes in Figs. 4–6 (and some illustrations) can be accessed via `vignette("hcl-colors", package = "colorspace")`.

The default color palettes in the **ggplot2** package (Wickham, 2008) are also based on HCL colors, using similar ideas to those discussed in this article.

http://cran.r-project.org/web/packages/colorspace/index.html

Bottom-line:
Consider going beyond the R's default colors, color palettes, and color palette-building functions. They're pretty bad.

Ready-made palettes exist in RColorBrewer and dichromat and HCL-color-model based tools exist in colorspace for building your own palettes.

The example up til now is unrealistic (who really wants each point to have its own color?) and elementary (it's not that hard to get that far by yourself).

Typical task: encode the information in a factor with color.

How to do?

we paused here ... continuing in next class

```
> (jLevels <- paste0("grp", 1:3))
[1] "grp1" "grp2" "grp3"
> jDat$group <- factor(sample(jLevels, nC, replace = TRUE))
> jDat
                country year       pop continent lifeExp  gdpPercap group
336   Congo, Dem. Rep. 2007  64606759    Africa  46.462   277.5519  grp1
1356      Sierra Leone 2007   6144562    Africa  42.568   862.5408  grp2
108         Bangladesh 2007 150448339      Asia  64.062  1391.2538  grp3
816             Jordan 2007   6053193      Asia  72.535  4519.4612  grp1
1416      South Africa 2007  43997828    Africa  49.339  9269.6578  grp1
732               Iran 2007  69453570      Asia  70.964 11605.7145  grp3
948           Malaysia 2007  24821286      Asia  74.241 12451.6558  grp1
672   Hong Kong, China 2007   6980412      Asia  82.208 39724.9787  grp2

> (jColors <- data.frame(group = jLevels,
+                        color = I(brewer.pal(n = 3, name = 'Dark2'))))
  group   color
1  grp1 #1B9E77
2  grp2 #D95F02
3  grp3 #7570B3
```

I randomly created a grouping factor, with 3 levels: grp1, grp2, and grp3.

In a separate data.frame, I've associated those levels with colors drawn from the Dark2 RColorBrewer palette.

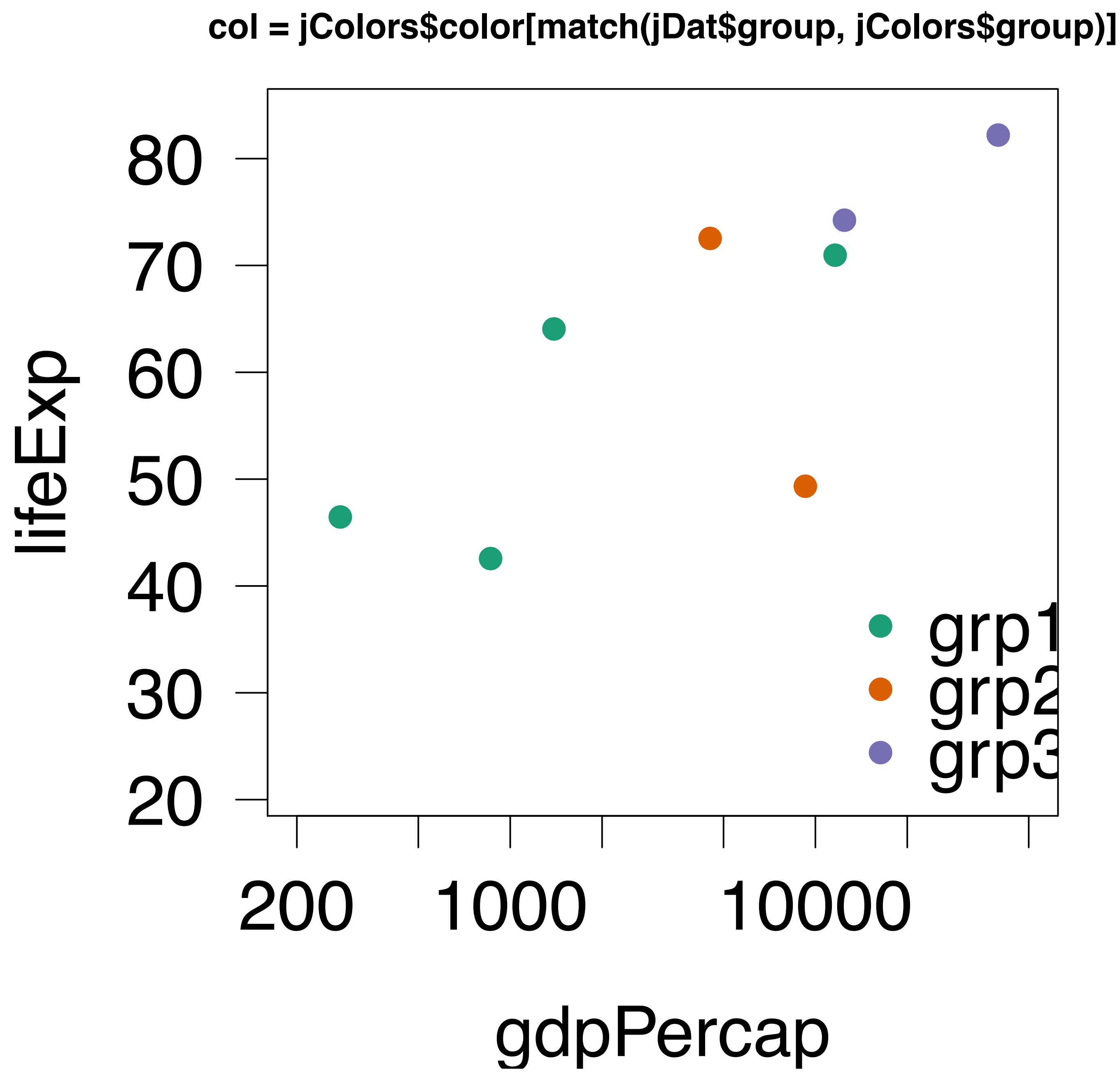
```
> (jLevels <- paste0("grp", 1:3))
[1] "grp1" "grp2" "grp3"
> jDat$group <- factor(sample(jLevels, nC, replace = TRUE))
> jDat
              country year       pop continent lifeExp  gdpPercap group
336  Congo, Dem. Rep. 2007  64606759    Africa  46.462   277.5519  grp1
1356     Sierra Leone 2007   6144562    Africa  42.568   862.5408  grp2
108        Bangladesh 2007 150448339      Asia  64.062  1391.2538  grp3
816            Jordan 2007   6053193      Asia  72.535  4519.4612  grp1
1416     South Africa 2007  43997828    Africa  49.339  9269.6578  grp1
732              Iran 2007  69453570      Asia  70.964 11605.7145  grp3
948          Malaysia 2007  24821286      Asia  74.241 12451.6558  grp1
672  Hong Kong, China 2007   6980412      Asia  82.208 39724.9787  grp2

> (jColors <- data.frame(group = jLevels,
+                        color = I(brewer.pal(n = 3, name = 'Dark2'))))
  group   color
1  grp1 #1B9E77
2  grp2 #D95F02
3  grp3 #7570B3
```

Example of protecting a variable with I() that I want to keep as character, i.e. want to suppress R's tendency to convert to factor.

```
> jDat
                country year       pop continent lifeExp  gdpPercap group
336  Congo, Dem. Rep. 2007  64606759    Africa  46.462   277.5519  grp1
1356     Sierra Leone 2007   6144562    Africa  42.568   862.5408  grp1
108        Bangladesh 2007 150448339      Asia  64.062  1391.2538  grp1
816            Jordan 2007   6053193      Asia  72.535  4519.4612  grp2
1416     South Africa 2007  43997828    Africa  49.339  9269.6578  grp2
732              Iran 2007  69453570      Asia  70.964 11605.7145  grp1
948          Malaysia 2007  24821286      Asia  74.241 12451.6558  grp3
672  Hong Kong, China 2007   6980412      Asia  82.208 39724.9787  grp3

> (jColors <- data.frame(group = jLevels,
+                        color = I(brewer.pal(n = 3, name = 'Dark2'))))
  group   color
1  grp1 #1B9E77
2  grp2 #D95F02
3  grp3 #7570B3

> match(jDat$group, jColors$group)
[1] 1 1 1 2 2 1 3 3

> jColors$color[match(jDat$group, jColors$group)]
[1] "#1B9E77" "#1B9E77" "#1B9E77" "#D95F02" "#D95F02" "#1B9E77" "#7570B3"
[8] "#7570B3"
```

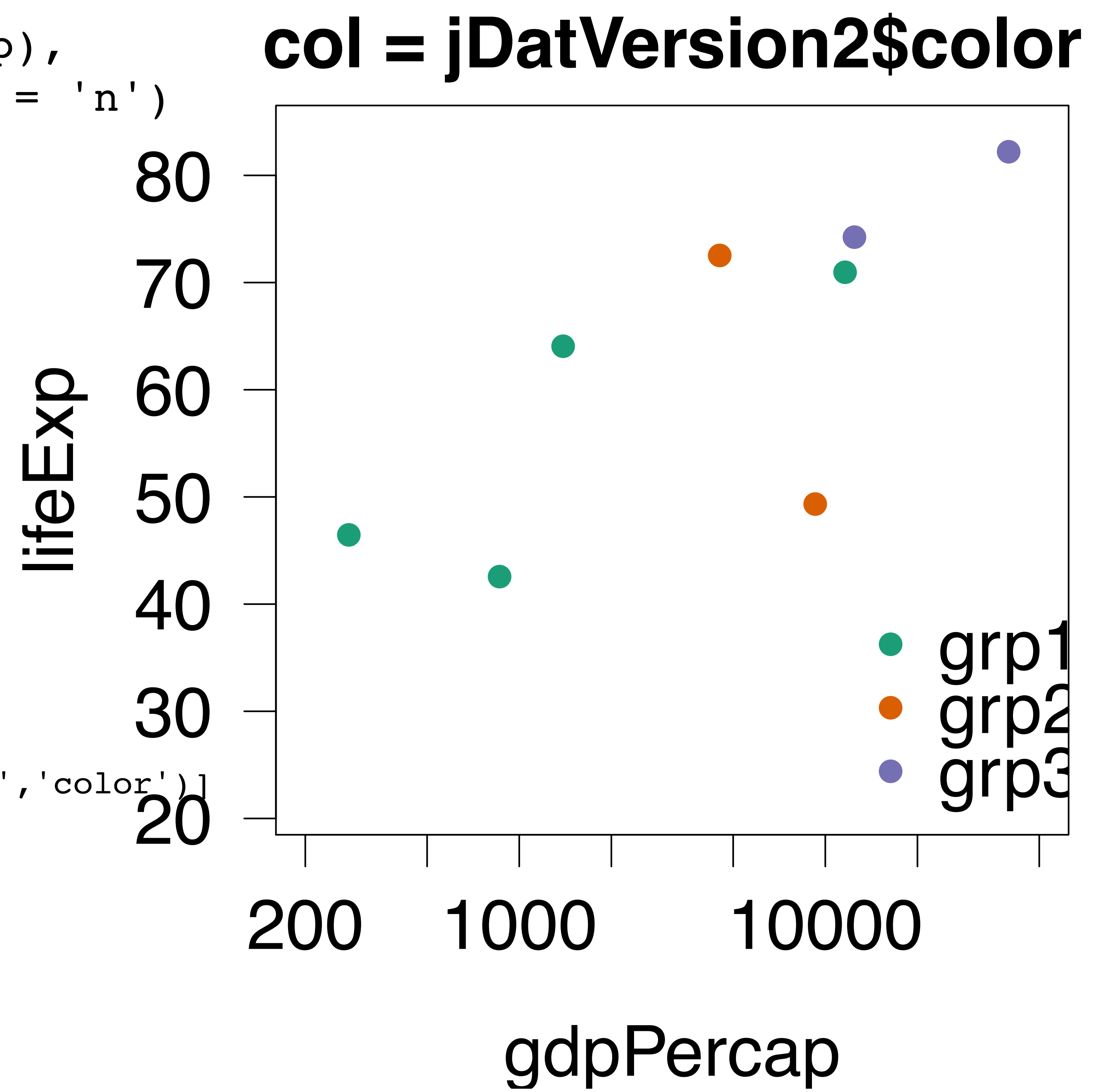
match() gets you a vector of indices which can then be used to index the vector colors. Part of R's toolkit for "table look-up" operations.

```
plot(lifeExp ~ gdpPercap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = jColors$color[match(jDat$group, jColors$group)],
     main = 'col = jColors$color[match(jDat$group, jColors$group)]',
     cex.main = 0.5)
legend(x = 'bottomright',
       legend = as.character(jColors$group),
       col = jColors$color, pch = 16, bty = 'n', xjust = 1)
```



```r
jDatVersion2 <- merge(jDat, jColors)
plot(lifeExp ~ gdpPercap, jDatVersion2, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = color,
     main = 'col = jDatVersion2$color',
     cex.main = 1)
legend(x = 'bottomright',
       legend = as.character(jColors$group),
       col = jColors$color, pch = 16, bty = 'n')
```

If you're willing to bring color info into the data.frame, merge() makes this incredibly easy.




```
> jDatVersion2[c('country','gdpPercap','lifeExp','group','color')]
             country  gdpPercap lifeExp group   color
1 Congo, Dem. Rep.   277.5519  46.462  grp1 #1B9E77
2     Sierra Leone   862.5408  42.568  grp1 #1B9E77
3       Bangladesh  1391.2538  64.062  grp1 #1B9E77
4             Iran 11605.7145  70.964  grp2 #D95F02
5           Jordan  4519.4612  72.535  grp2 #D95F02
6     South Africa  9269.6578  49.339  grp1 #1B9E77
7         Malaysia 12451.6558  74.241  grp3 #7570B3
8 Hong Kong, China 39724.9787  82.208  grp3 #7570B3
```

My recommendations:

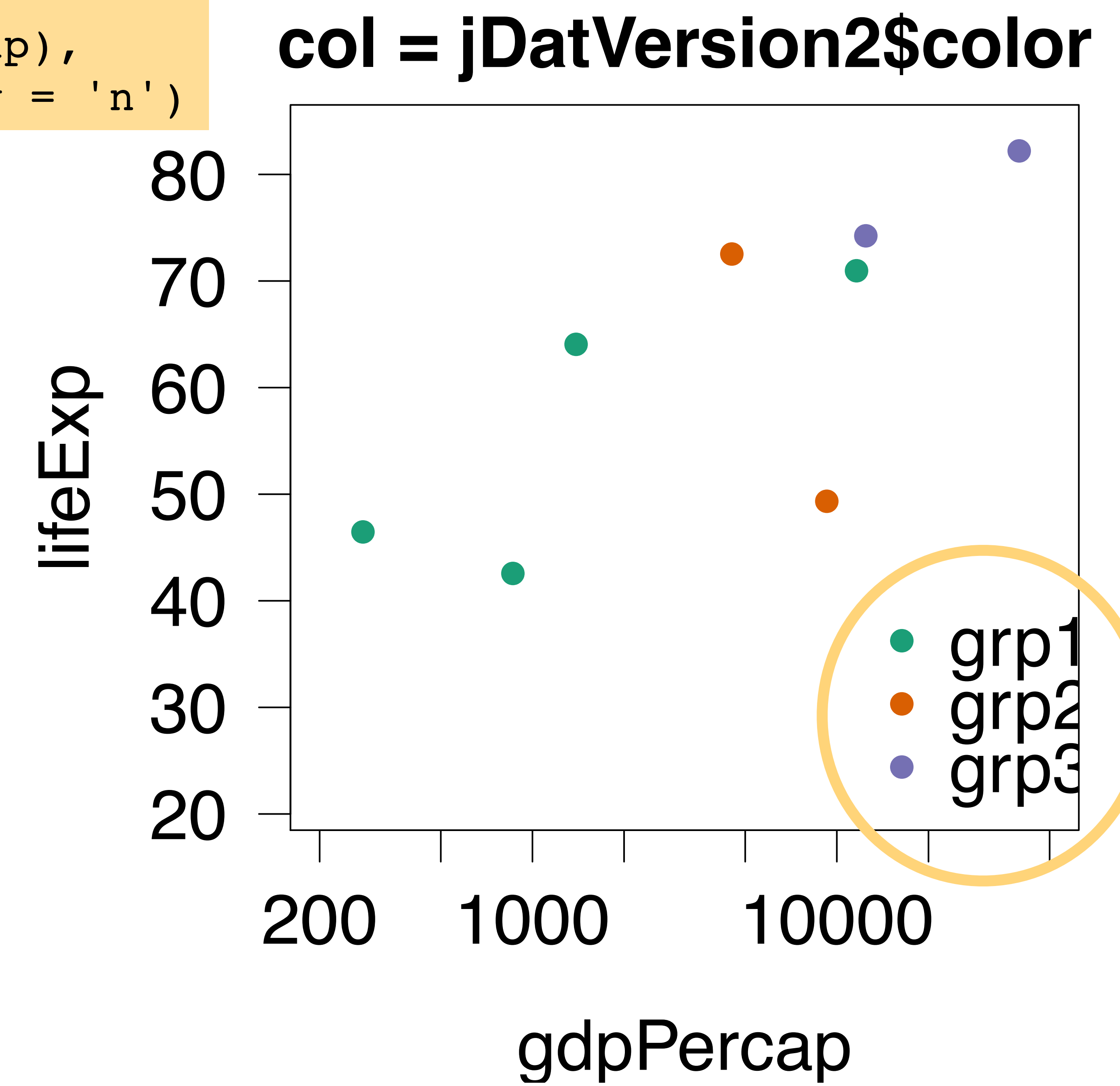
Use RColorBrewer or dichromat for your schemes (or as the basis of complicated schemes -- see Gapminder example next).

Store your scheme in an R object, like a vector or data.frame. Will be handy for code re-use, making legends, keeping colors consistent over several figures, etc.

Use match() to map a factor into colors or, often more useful, merge() to integrate the color variable with the data itself. The need for you to get personally involved in this is greatly reduced / delayed if you use lattice and the "groups" argument. Suspect something similar is true for ggplot2. Another downside of base graphics.

```
jDatVersion2 <- merge(jDat, jColors)
plot(lifeExp ~ gdpPercap, jDatVersion2, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = color,
     main = 'col = jDatVersion2$color',
     cex.main = 1)
legend(x = 'bottomright',
       legend = as.character(jColors$group),
       col = jColors$color, pch = 16, bty = 'n')
```

legend() is ... how you make a legend! Read the documentation and gradually build up the legend you want. Too fiddly and figure-specific to discuss here.



End: encoding the information in a factor with color ‘by hand’.

Continent / country colors vexed almost everyone in Assignment 1.

"I failed to assign different colors to countries from different continent"

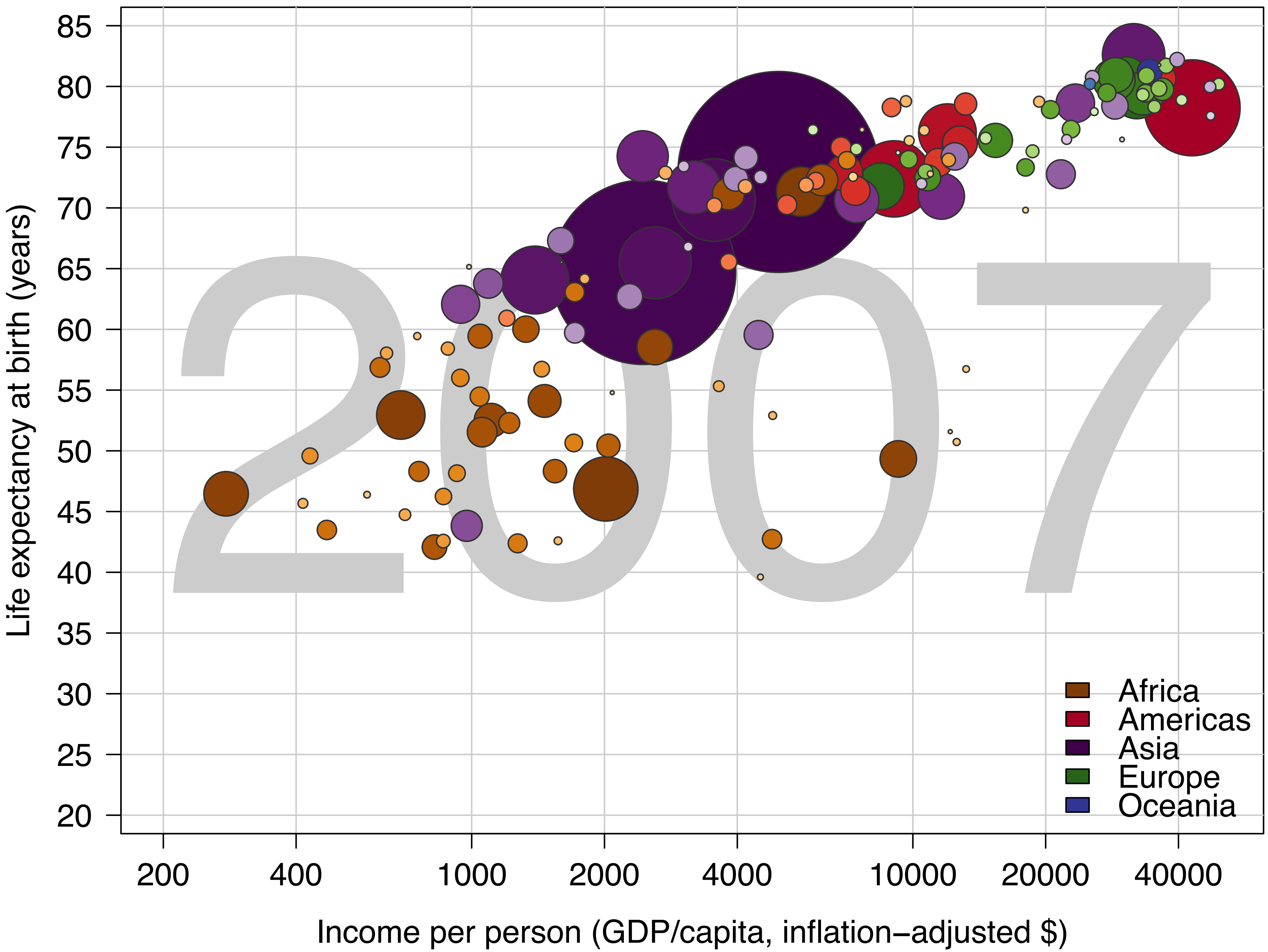
I know there are six continents in total and the command col=1:6 represents 6 different colors. But I really do not understand how to assign the different colors to each continent,

An extremely difficult step was to figure out how to relate the geographical are with the color coding.

I didn't use continent at all.

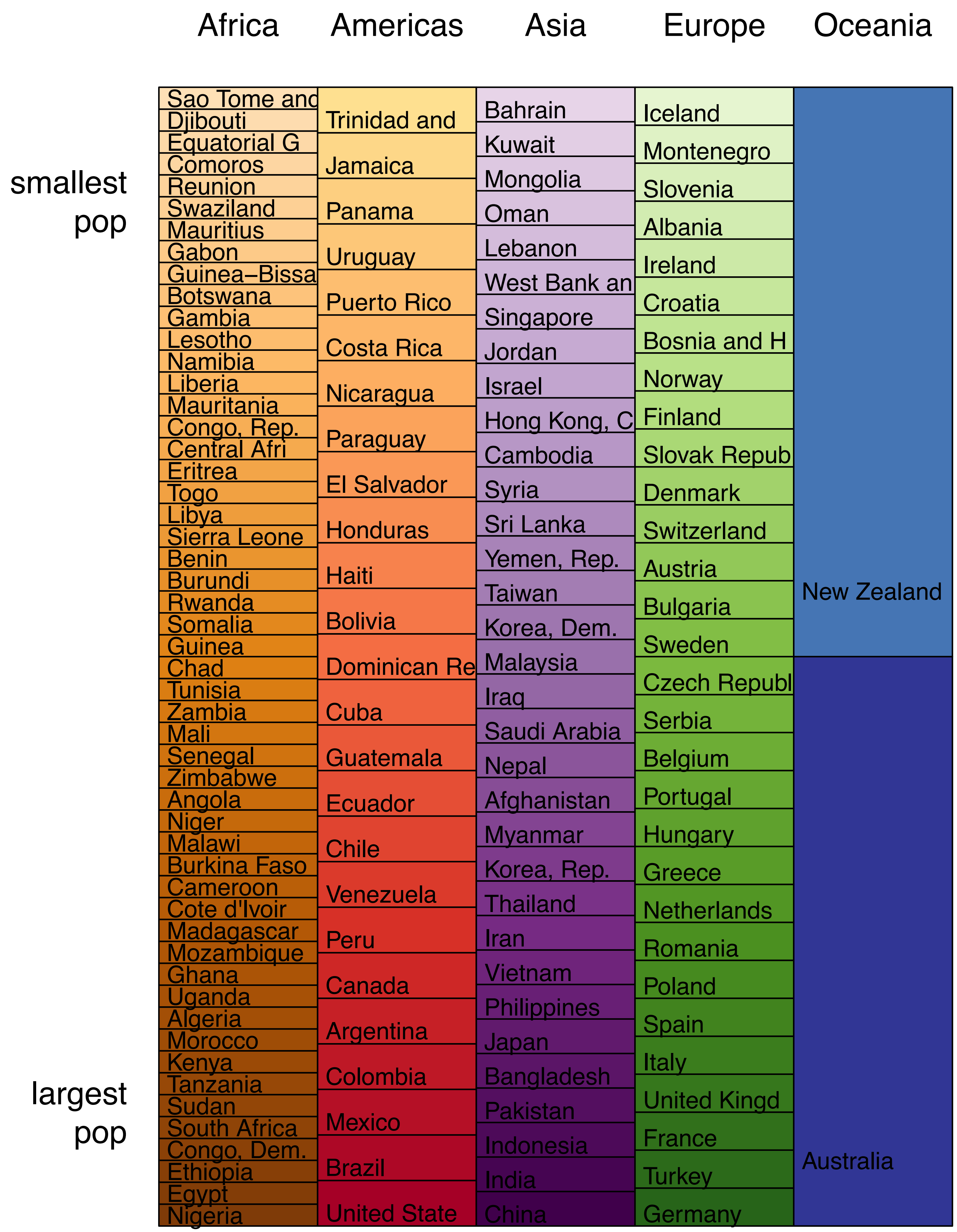
began trying to figure out how to re-color each dot based on continent. This proved to be beyond me at the moment, though I did end up with some interesting looking plots with col=rainbow(##) (of course the colors were then meaningless, but still progress nonetheless). I left the dots monotone, but I will try to figure out how to specify color by parameter this weekend at some point.

*Note: These frustrations expressed by past STAT 545A students. Your mileage may vary.



The Gapminder Color Scheme: How did JB construct it?

Gapminder Color Scheme



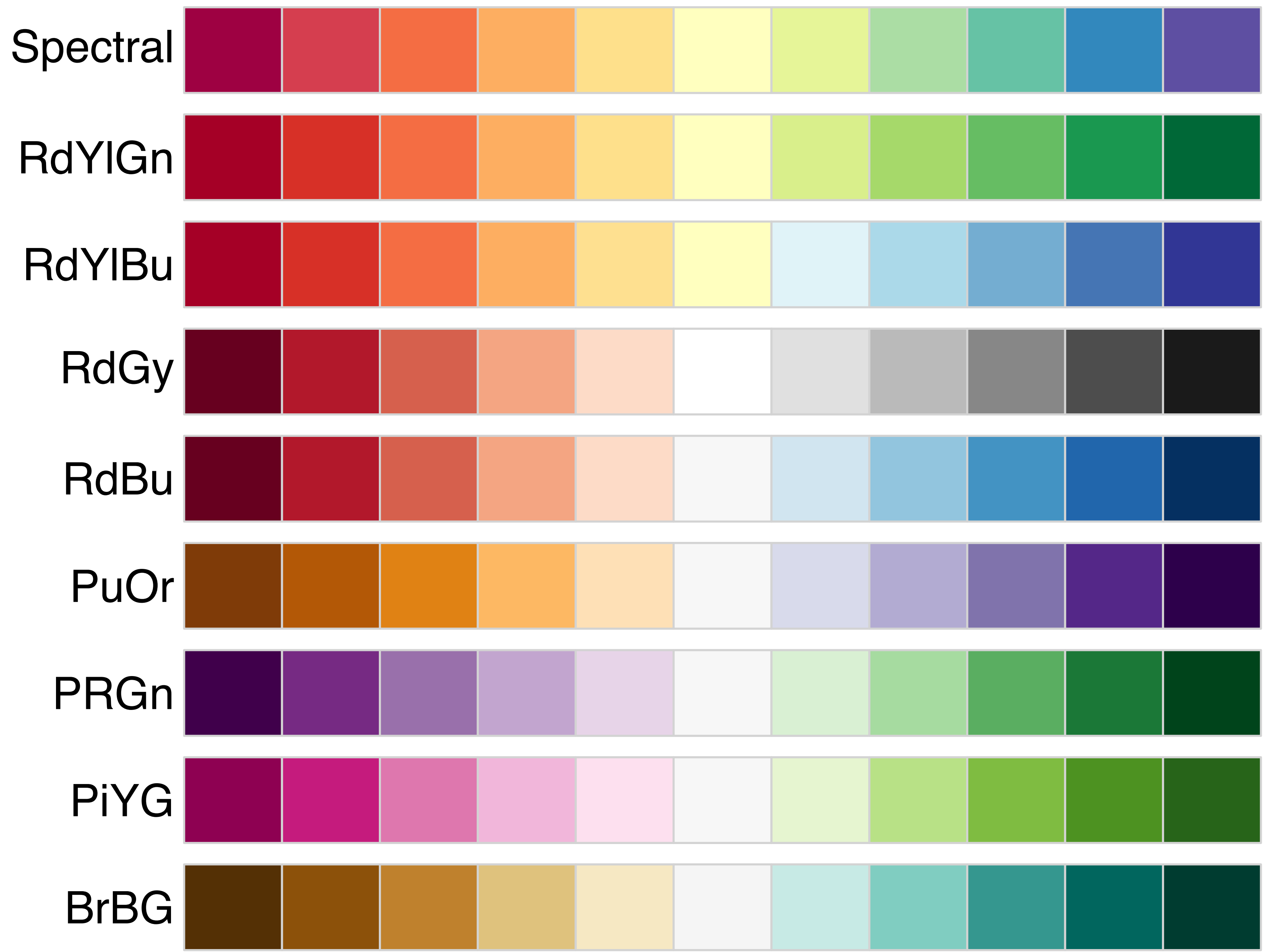
Caveat: This took a lot of time, a lot of tricks.

I don't regard this as a core basic skill of figure-making in R. It's rather advanced.

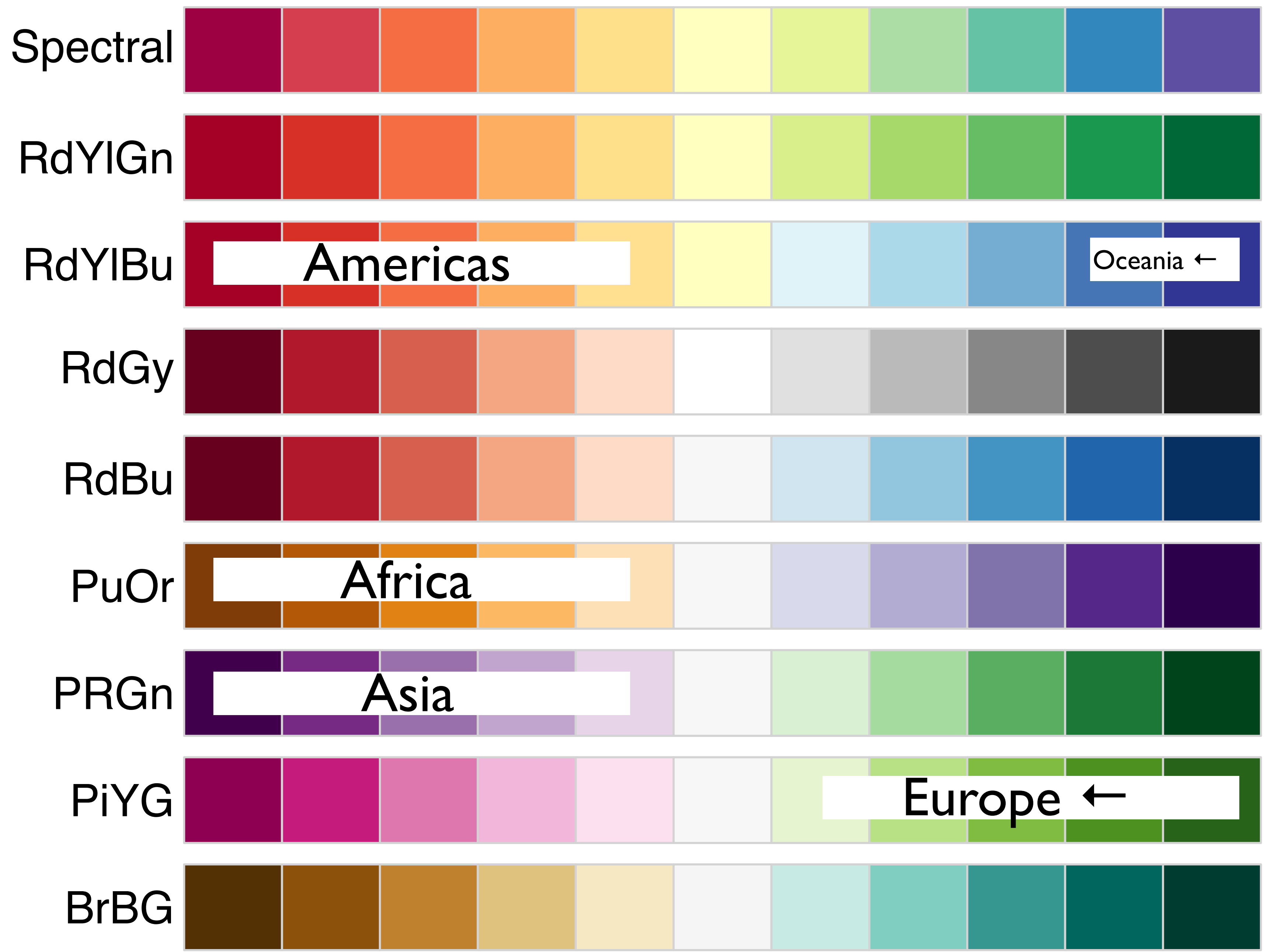
I'll show here for completeness, but we may not even go through all of this in class.

Takeaway #1: start with a professional palette.

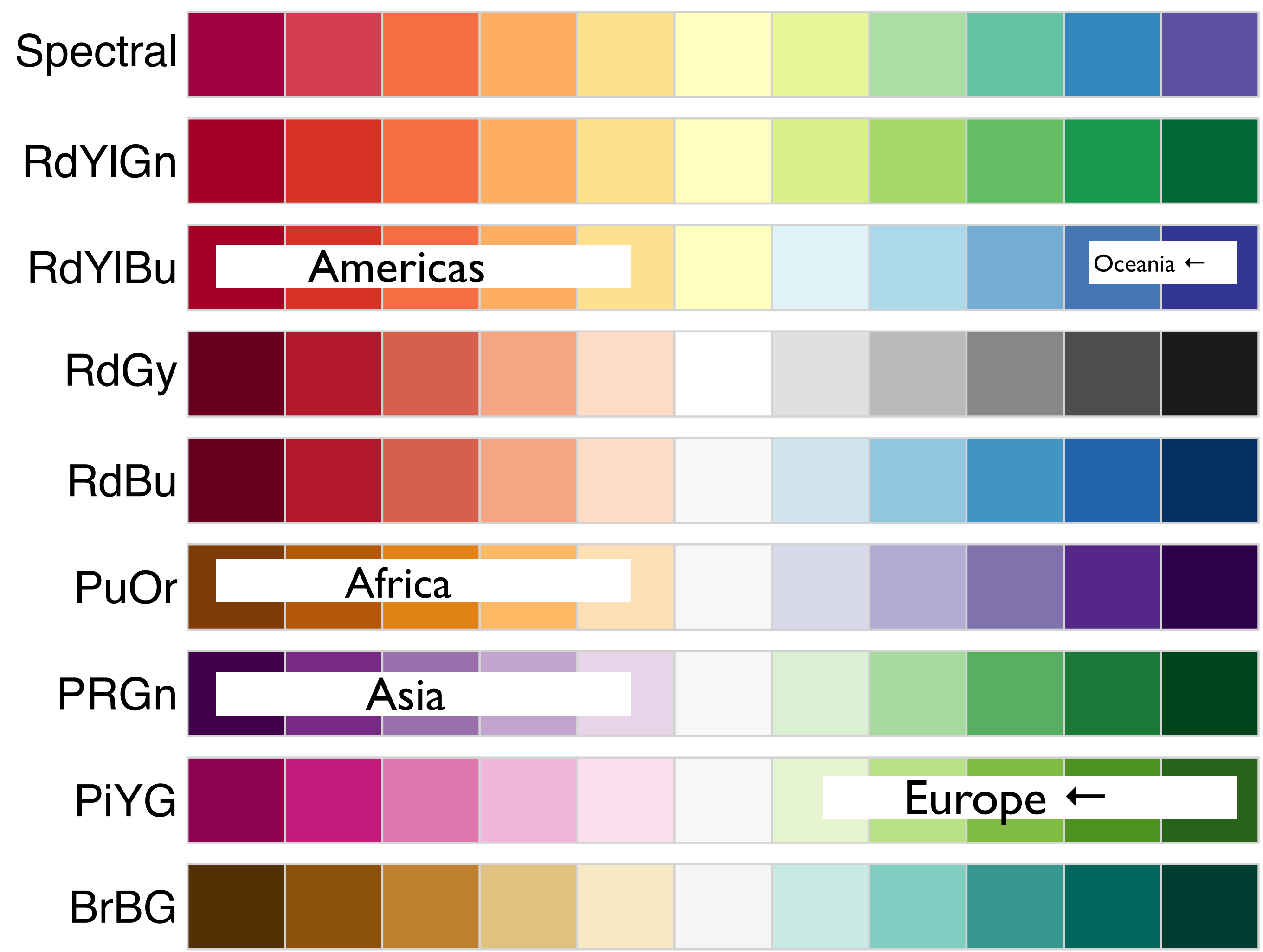
```
library(RColorBrewer)
display.brewer.all(type = "div")
```



```
colorAnchors <-
  list(Africa = brewer.pal(n = 11, 'PuOr')[1:5], # orange/brown/gold
       Americas = brewer.pal(n = 11, 'RdYlBu')[1:5],      # red
       Asia = brewer.pal(n = 11, 'PRGn')[1:5],            # purple
       Europe = brewer.pal(n = 11, 'PiYG')[11:7],         # green
       Oceania = brewer.pal(n = 11, 'RdYlBu')[11:10])     # blue
```



```
> colorAnchors
$Africa
[1] "#7F3B08" "#B35806" "#E08214" "#FDB863" "#FEE0B6"

$Americas
[1] "#A50026" "#D73027" "#F46D43" "#FDAE61" "#FEE090"

$Asia
[1] "#40004B" "#762A83" "#9970AB" "#C2A5CF" "#E7D4E8"

$Europe
[1] "#276419" "#4D9221" "#7FBC41" "#B8E186" "#E6F5D0"

$Oceania
[1] "#313695" "#4575B4"
```



```
## turn those into a palette big enough to cover each country in a
## continent
countryColors <- lapply(seq_len(nCont), function(i) {
  yo <- droplevels(subset(gDat, continent == cDat$continent[i]))
  countriesBigToSmall <- rev(levels(reorder(yo$country, yo$pop, max)))
  colorFun <- colorRampPalette(colorAnchors[[i]])
  return
                                                      $mall))))
})
```

Takeaway #2:
Use colorRampPalette() or colorRamp() to expand a professional palette (or excerpt thereof) into the full range of colors you need.

Above

Isolate
biggest

Expand e with one entry for each country. Store as a data.frame and return.

*There's a reason I use lapply in this way but let's stay focused on the colors.

```r
## turn those into a palette big enough to cover each country in a
## continent
countryColors <- lapply(seq_len(nCont), function(i) {
  yo <- droplevels(subset(gDat, continent == cDat$continent[i]))
  countriesBigToSmall <- rev(levels(reorder(yo$country, yo$pop, max)))
  colorFun <- colorRampPalette(colorAnchors[[i]])
  return(data.frame(continent = cDat$continent[i],
                    country = I(countriesBigToSmall),
                    color = I(colorFun(length(countriesBigToSmall)))))
})
```

Above is essentially a loop over the continents*.

Isolate the countries for the continent and sort from biggest to smallest.

Expand the previously set colorAnchors into a palette with one entry for each country. Store as a data.frame and return.

*There's a reason I use lapply in this way but let's stay focused on the colors.

```r
## turn those into a palette big enough to cover each country in a
## continent
countryColors <- lapply(seq_len(nCont), function(i) {
  yo <- refactor(subset(gDat, continent == cDat$continent[i]))
  countriesBigToSmall <- rev(levels(reorder(yo$country, yo$pop, max)))
  colorFun <- colorRampPalette(colorAnchors[[i]])
  return(data.frame(continent = cDat$continent[i],
                    country = I(countriesBigToSmall),
                    color = I(colorFun(length(countriesBigToSmall)))))
})
```

The key functionality -- the interpolation of colors -- comes from colorRampPalette().

Input = colors to interpolate
Output = a function (!) that takes an integer as input and outputs a vector of colors with that length

A close relative is colorRamp(), which is helpful for mapping the interval [0, 1] to to colors. Will see later in course.

```
> countryColors[['Europe']]
   continent                    country   color
1     Europe                    Germany #276419
2     Europe                     Turkey #2C6A1A
3     Europe                     France #31701B
4     Europe             United Kingdom #36771C
5     Europe                      Italy #3B7D1D
6     Europe                      Spain #41831E
7     Europe                     Poland #468A1F
8     Europe                    Romania #4B9020
9     Europe                Netherlands #529624
10    Europe                     Greece #599C28
11    Europe                    Hungary #5FA12D
12    Europe                   Portugal #66A731
13    Europe                    Belgium #6DAD35
14    Europe                     Serbia #74B33A
15    Europe             Czech Republic #7BB93E
16    Europe                     Sweden #82BE45
17    Europe                   Bulgaria #8AC34F
18    Europe                    Austria #92C858
19    Europe                Switzerland #9ACD62
20    Europe                    Denmark #A2D26B
21    Europe            Slovak Republic #AAD875
22    Europe                    Finland #B2DD7E
23    Europe                     Norway #B9E188
24    Europe Bosnia and Herzegovina #BFE492
25    Europe                    Croatia #C6E79C
26    Europe                    Ireland #CCE9A7
27    Europe                    Albania #D2ECB1
28    Europe                   Slovenia #D9EFBB
29    Europe                 Montenegro #DFF2C5
30    Europe                    Iceland #E6F5D0
```

Each country is now associated with a color.

Furthermore, this was enacted within continent, so all countries in, say, Europe, will be some shade of green.

And last but not least, within continent the dark colors are for big countries and the lighter colors are for small ones. Another measure to help see the small countries.

```
> i <- 2
>   yo <- refactor(subset(gDat, continent == cDat$continent[i]))
>   countriesBigToSmall <- rev(levels(reorder(yo$country, yo$pop, max)))

> countriesBigToSmall
 [1] "United States"       "Brazil"              "Mexico"
 [4] "Colombia"            "Argentina"           "Canada"
 [7] "Peru"                "Venezuela"           "Chile"
[10] "Ecuador"             "Guatemala"           "Cuba"
[13] "Dominican Republic"  "Bolivia"             "Haiti"
[16] "Honduras"            "El Salvador"         "Paraguay"
[19] "Nicaragua"           "Costa Rica"          "Puerto Rico"
[22] "Uruguay"             "Panama"              "Jamaica"
[25] "Trinidad and Tobago"


>   colorFun <- colorRampPalette(colorAnchors[[i]])


> colorFun
function (n)
{
    x <- ramp(seq.int(0, 1, length.out = n))
    rgb(x[, 1], x[, 2], x[, 3], maxColorValue = 255)
}
<environment: 0x10219fe40>
```

The key functionality -- the interpolation of colors -- comes from colorRampPalette().

Input = colors to interpolate
Output = a function (!) that takes an integer as input and outputs a vector of colors with that length

```
> colorAnchors
$Africa
[1] "#7F3B08" "#B35806" "#E08214" "#FDB863" "#FEE0B6"

$Americas
[1] "#A50026" "#D73027" "#F46D43" "#FDAE61" "#FEE090"

$Asia
[1] "#40004B" "#762A83" "#9970AB" "#C2A5CF" "#E7D4E8"

$Europe
[1] "#276419" "#4D9221" "#7FBC41" "#B8E186" "#E6F5D0"

$Oceania
[1] "#313695" "#4575B4"
```

This interpolation / expansion is what colorRampPalette() helps you to do.

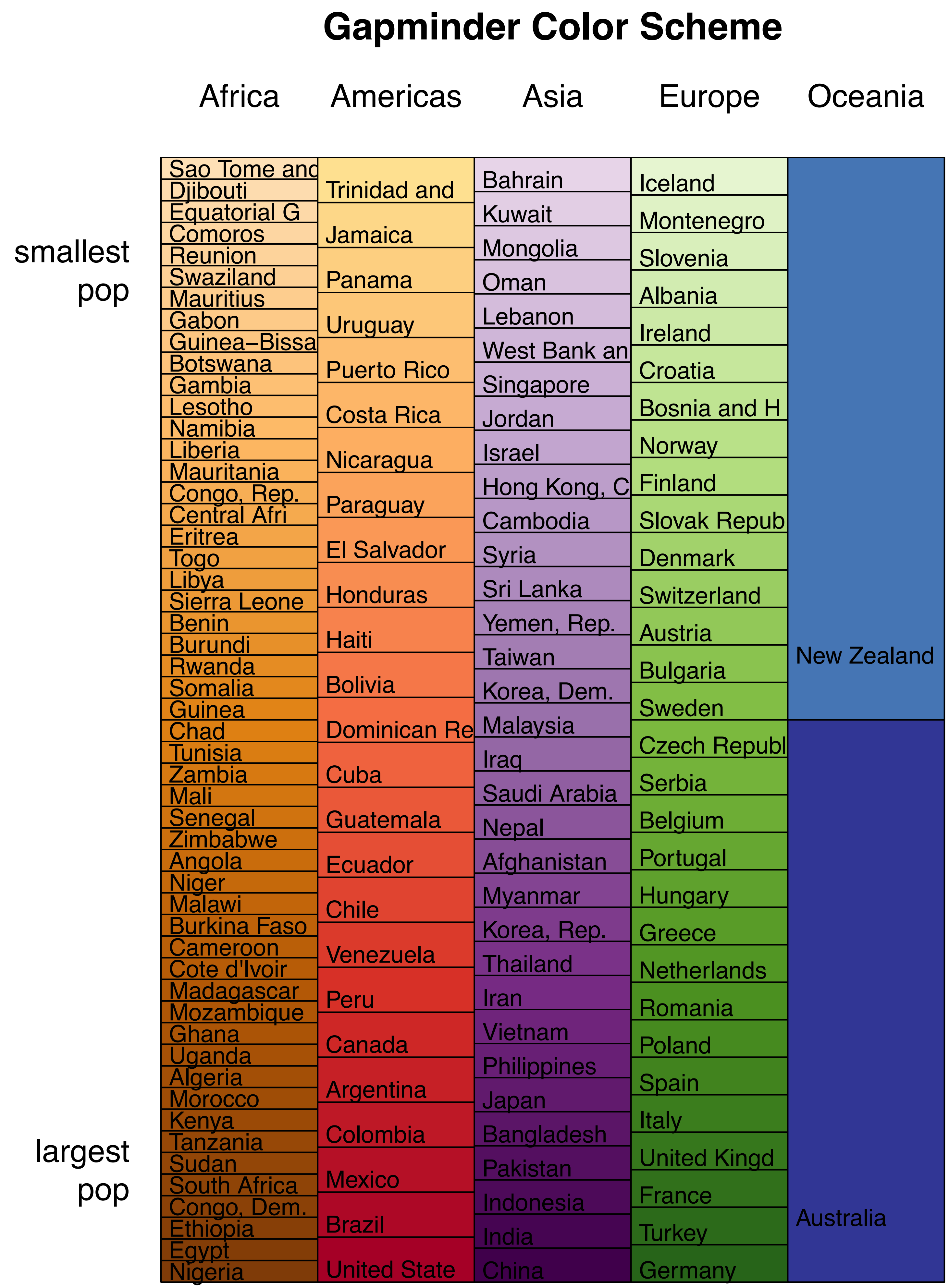
```
> countryColors[['Europe']]
   continent                     country   color
1     Europe                     Germany #276419
2     Europe                      Turkey #2C6A1A
3     Europe                      France #31701B
.....
29    Europe                  Montenegro #DFF2C5
30    Europe                     Iceland #E6F5D0
```

```
## I would like to stack these up, row-wise, into a data.frame that
## holds my color scheme

countryColors <- do.call(rbind, countryColors)

str(countryColors)
## 'data.frame':142 obs. of  3 variables:
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ country  :Class 'AsIs'  chr [1:142] "Nigeria" "Egypt" "Ethiopia" "Congo, De..
##  $ color    :Class 'AsIs'  chr [1:142] "#7F3B08" "#833D07" "#873F07" "#8B4107"..
```

do.call() trick helps us re-assemble the continent specific color schemes into one united color scheme.

```
> peek(countryColors)
    continent                  country   color
22     Africa                  Senegal #D0730F
27     Africa                   Guinea #E18417
38     Africa               Mauritania #FAB25B
50     Africa        Equatorial Guinea #FDD9A8
82       Asia               Bangladesh #5B1567
121    Europe                  Hungary #5FA12D
134    Europe Bosnia and Herzegovina #BFE492
```

This is what countryColors holds.



```
write.table(countryColors,
            paste0(whereAmI, "data/gapminderCountryColors.txt"),
            quote = FALSE, sep = "\t", row.names = FALSE)

write.table(cDat,
            paste0(whereAmI, "data/gapminderContinentColors.txt"),
            quote = FALSE, sep = "\t", row.names = FALSE)
```

Write the country color scheme to file, for re-use in all my "solutions". A very useful practice in many graphics-heavy analyses.

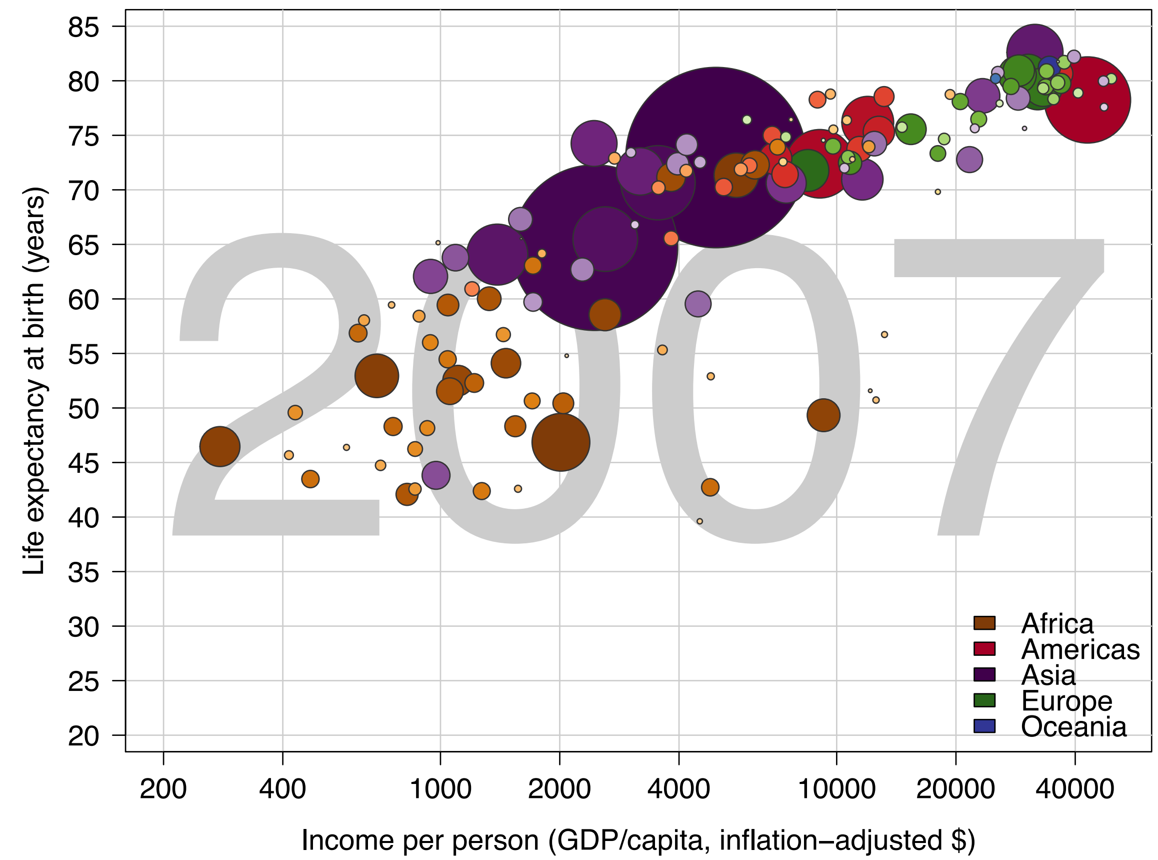
Read them back in whenever you need.

```
## use the color scheme created in
## bryan-a01-30-makeGapminderColorScheme.R
continentColors <-
  read.delim(paste0(whereAmI, "data/gapminderContinentColors.txt"),
             as.is = 3)                    # protect color

countryColors <-
  read.delim(paste0(whereAmI, "data/gapminderCountryColors.txt"),
             as.is = 3)                    # protect color
```

merge() merges the data (gDat) and the color scheme (countryColors) on the common variables, making the variable color available for plot(), symbols(), etc.

```
> peek(countryColors)
     continent                  country   color
5       Africa             South Africa #8F4407
15      Africa            Cote d'Ivoire #B75C07
18      Africa                   Malawi #C2650A
27      Africa                   Guinea #E18417
52      Africa Sao Tome and Principe #FEE0B6
81        Asia                 Pakistan #540F60
98        Asia                Sri Lanka #AD8ABD
> peek(gDat)
          country year      pop continent lifeExp  gdpPercap
189      Bulgaria 1992  8658506    Europe  71.190  6302.6234
194  Burkina Faso 1957  4713416    Africa  34.906   617.1835
571       Germany 1982 78335266    Europe  73.800 22031.5327
768        Israel 2007  6426679      Asia  80.745 25523.2771
779         Italy 2002 57926999    Europe  80.240 27968.0982
842   Korea, Rep. 1957 22611552      Asia  52.681  1487.5935
1519     Tanzania 1982 19844382    Africa  50.608   874.2426


> gDat <- merge(gDat, countryColors)
> peek(gDat)
                      country continent year      pop lifeExp gdpPercap   color
109                   Belgium    Europe 1952  8730405  68.000  8343.105 #6DAD35
255  Central African Republic    Africa 1962  1523478  39.475  1193.069 #F5AA4E
788                   Jamaica  Americas 1987  2326606  71.770  6351.237 #FDD788
1147                   Norway    Europe 1982  4114787  75.970 26298.635 #B9E188
1153                     Oman      Asia 1952   507833  37.578  1828.230 #D9C2DE
1572                  Tunisia    Africa 2007 10276158  73.923  7092.923 #DA7D12
1656                  Vietnam      Asia 2007 85262356  74.249  2441.576 #6F247B
```

```
plot(lifeExp ~ gdpPercap, gapDat, ...)
with(subset(gapDat, year == jYear),
     symbols(x = gdpPercap, y = lifeExp,
             circles = jPopRadFun(pop), add = TRUE,
             inches = 0.7,
             fg = jDarkGray, bg = color))
```




```
> gDat <- merge(gDat, countryColors)
> peek(gDat)
                      country continent year      pop lifeExp gdpPercap   color
109                   Belgium    Europe 1952  8730405  68.000  8343.105 #6DAD35
255  Central African Republic    Africa 1962  1523478  39.475  1193.069 #F5AA4E
788                   Jamaica  Americas 1987  2326606  71.770  6351.237 #FDD788
1147                   Norway    Europe 1982  4114787  75.970 26298.635 #B9E188
1153                     Oman      Asia 1952   507833  37.578  1828.230 #D9C2DE
1572                  Tunisia    Africa 2007 10276158  73.923  7092.923 #DA7D12
1656                  Vietnam      Asia 2007 85262356  74.249  2441.576 #6F247B
```

Core ideas for color schemes:

Use RColorBrewer or dichromat palettes as the basis for your schemes. And/or use colorspace package to develop more complicated schemes.

colorRampPalette() and colorRamp() help you interpolate colors.

Store the scheme as a data.frame, associating each level of the relevant factor with a color. Save it to file for re-use throughout a multi-script analysis.

Use that scheme with merge() to populate a color vector in the main data.frame. This will then be available when calling graphics functions.

Use the scheme again to make a legend.

Note this template generalizes to line types, etc.