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 $x = \text{sum of bumps centered at observed data } x_i$. 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/
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

```r
op <- par(mar = c(5, 4, 1, 1) + 0.1)
plot(lifeExp ~ gdpPercap, gDat,
    subset = year == jYear)
par(op)
```

---

```r
gdpPercap
lifeExp
```
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

```r
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)
```
## 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

```r
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 'by hand'

```r
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.

```r
## 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

```r
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.
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

```r
jPopRadFun <- function(jPop) {
  sqrt(jPop/pi)
}
```

```r
plot(jPopRadFun(pop) ~ pop, gDat)  # looks promising
```

Plot this for a sanity check before throwing into main figure command.
## map pop into circle radius

```r
jPopRadFun <- function(jPop) {
  sqrt(jPop/pi)
}
```

```r
plot(jPopRadFun(pop) ~ pop, gDat)
```

```r
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.
## use 'plot()' to set things up and then add other elements

```r
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))
```
## 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

```r
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))
```
## use 'plot()' to set things up and then add other elements

```r
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)

<table>
<thead>
<tr>
<th>continent</th>
<th>country</th>
<th>color</th>
<th>year</th>
<th>pop</th>
<th>lifeExp</th>
<th>gdpPercap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe</td>
<td>Belgium</td>
<td>#6DAD35</td>
<td>1962</td>
<td>9218400</td>
<td>70.250</td>
<td>10991.207</td>
</tr>
<tr>
<td>Africa</td>
<td>Congo, Rep.</td>
<td>#F7AE55</td>
<td>1967</td>
<td>1179760</td>
<td>52.040</td>
<td>2677.940</td>
</tr>
<tr>
<td>Africa</td>
<td>Namibia</td>
<td>#FDBA67</td>
<td>1972</td>
<td>821782</td>
<td>53.867</td>
<td>3746.081</td>
</tr>
<tr>
<td>Africa</td>
<td>Comoros</td>
<td>#FDD6A2</td>
<td>1992</td>
<td>454429</td>
<td>57.939</td>
<td>1246.907</td>
</tr>
<tr>
<td>Africa</td>
<td>Djibouti</td>
<td>#FDDCAF</td>
<td>1992</td>
<td>384156</td>
<td>51.604</td>
<td>2377.156</td>
</tr>
<tr>
<td>Asia</td>
<td>Yemen, Rep.</td>
<td>#A883B8</td>
<td>2007</td>
<td>22211743</td>
<td>62.698</td>
<td>2280.770</td>
</tr>
<tr>
<td>Asia</td>
<td>Cambodia</td>
<td>#B797C6</td>
<td>2007</td>
<td>14131858</td>
<td>59.723</td>
<td>1713.779</td>
</tr>
</tbody>
</table>
## 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.
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.
writeToFile <- TRUE

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.
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`.

```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)
```
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 GREATLY enhanced by these
practices. I have learned this the hard way.
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.
Greatest hits of the base R solution

<table>
<thead>
<tr>
<th>plot(y ~ x, myData, subset = sthgLogical)</th>
<th>axis()</th>
<th>legend()</th>
</tr>
</thead>
<tbody>
<tr>
<td>par()</td>
<td>abline()</td>
<td></td>
</tr>
<tr>
<td>symbols()</td>
<td>text()</td>
<td>mtext()</td>
</tr>
</tbody>
</table>
using colors in R

mostly focused on base/traditional R graphics

will revisit when we cover lattice
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.
Recycling happens.

```r
plot(lifeExp ~ gdpPerCap, jDat, log = 'x',
     xlim = jXlim, ylim = jYlim,
     col = c("red", "green"),
     main = 'col = c("red", "green")')
```
You can specify a color via an integer.

This specifies colors within the current palette.

You’re looking at the default palette.
View and modify the palette with `palette()`.

Read documentation to see examples of changing the active palette.

The default palette is ugly.
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.
A long time ago I made a 6 page document for myself. Good times.
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
symbols, too ....
### Integer | Sample line | String
---|---|---
0 | | "blank"
1 | | "solid"
2 | | "dashed"
3 | | "dotted"
4 | | "dotdash"
5 | | "longdash"
6 | | "twodash"

**Predefined**

**Custom**

| | | |
---|---|---
| "13" | "F8" | "431313"
| "22848222" | | |

From Ch.3 of Murrell ‘R Graphics’
Temperature (°C) in 2003

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

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

\[ \hat{\beta} = (X'X)^{-1}X'y \]

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 Ch. 3 of Murrell ‘R Graphics’

From Ch. 4 of Murrell ‘R Graphics’
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()
Another source of color palettes suitable for colorblind people is the package `dichromat`
library(RColorBrewer)
display.brewer.pal(n = 8, name = 'Dark2')

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

<table>
<thead>
<tr>
<th>hex</th>
<th>decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>

### “unsaturated”, shades of gray

<table>
<thead>
<tr>
<th>color name</th>
<th>#rrggbhb</th>
<th>red</th>
<th>green</th>
<th>blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>white</td>
<td>#FFFFFF</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>gray50</td>
<td>#7F7F7F</td>
<td>127</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>black</td>
<td>#000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### “saturated”, primary colors

<table>
<thead>
<tr>
<th>color name</th>
<th>#rrggbhb</th>
<th>red</th>
<th>green</th>
<th>blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>blue</td>
<td>#0000FF</td>
<td>0</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>green</td>
<td>#00FF00</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>red</td>
<td>#FF0000</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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. A simple and very effective palette for such a display is a set of gray colors as in the top right panel of Fig. 1. This is often (appropriately) used in printed papers when the journal does not offer color graphics—however, in journals that support color graphics (or on presentation slides and in interactive usage in statistical software packages), many users prefer to have colored displays and most often use HSV palettes (as in the two left two panels). These palettes code the variable of interest by varying hue in an HSV color wheel, yielding a “rainbow” of colors, as done by Harezlak et al. (2007, Fig. 8) or Tenenhaus et al. (2007, Fig. 2). The palette in the upper left panel codes increasing density by going from a blue to a red hue (via green and yellow)—a similar strategy are the “heat colors” in the lower left panel that increase from yellow to red. The latter works somewhat better than the former, however both palettes exhibit several drawbacks. The modes in the map look much more like “rings” rather than a smoothly increasing/decreasing density. The heatmap looks very flashy which—although good for drawing attention to a plot—makes it hard to hold the attention of the viewer for a longer time because the large areas shaded with saturated colors can be distracting and produce after-image effects (Ihaka, 2003).

In contrast, the gray colors used in the top right panel do not exhibit the same disadvantages, coding the variable of interest much better and without flashy colors. If, however, the user wants to increase the contrast by adding some color to the plot, this could be done by using a better balanced version of the heat colors (as shown in the bottom right panel). These colors also increase from a yellow to a red hue while using the same brightness levels as in the grayscale palette (see also Ware, 1988; Levkowitz and Herman, 1992, for similar approaches). Thus, when converted to a grayscale or printed out on a grayscale printer (as in the printed version of Computational Statistics & Data Analysis), the upper and lower right panel would look (virtually) identical. Both palettes have in common that they give increasing perceptual emphasis to regions with...
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.
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 accessible.) 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
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.
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.
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.
```r
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)
```
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.
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.
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.

```r
jDatVersion2 <- merge(jDat, jColors)
plot(lifeExp ~ gdpPercap, jDatVersion2, log = 'x',
    xlab = jXlim, ylab = 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')
```
End: encoding the information in a factor with color ‘by hand’.
“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 area 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

<table>
<thead>
<tr>
<th>Africa</th>
<th>Americas</th>
<th>Asia</th>
<th>Europe</th>
<th>Oceania</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sao Tome and</td>
<td>Trinidad and</td>
<td>Bahrain</td>
<td>Iceland</td>
<td></td>
</tr>
<tr>
<td>Gabon</td>
<td>Zambé</td>
<td>Kazakstan</td>
<td>Montenegro</td>
<td></td>
</tr>
<tr>
<td>Comoros</td>
<td>Jamaica</td>
<td>Mongolia</td>
<td>Slovenia</td>
<td></td>
</tr>
<tr>
<td>Reunion</td>
<td>Panama</td>
<td>Oman</td>
<td>Albania</td>
<td></td>
</tr>
<tr>
<td>Swaziland</td>
<td>South Africa</td>
<td>Lebanon</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mauritius</td>
<td>Sudan</td>
<td>Ireland</td>
<td>Norway</td>
<td></td>
</tr>
<tr>
<td>Gabon</td>
<td>Tanzania</td>
<td>Croatia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guinea–Bissau</td>
<td>Kenya</td>
<td>Bosnia and Herzegovina</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Botswana</td>
<td>Madagascar</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gambia</td>
<td>Ethiopia</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesotho</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Namibia</td>
<td>Uruguay</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Liberia</td>
<td>Nicaragua</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mauritania</td>
<td>Paraguay</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Congo, Rep.</td>
<td>El Salvador</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Central Afri</td>
<td>United States</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eritrea</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eritrea</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Togo</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Libya</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sierra Leone</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Benin</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burundi</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rwanda</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Somalia</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guinea</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chad</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tunisia</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zambé</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mali</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Senegal</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zimbabwe</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Angola</td>
<td>Costa Rica</td>
<td>Jordan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nigeria</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malawi</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burkina Faso</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cameroon</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cote d'Ivoire</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Madagascar</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mozambique</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ghana</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uganda</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algeria</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Morocco</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kenya</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tanzania</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sudan</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>South Africa</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Congo, Dem.</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ethiopia</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Egypt</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nigeria</td>
<td>Chile</td>
<td>Malaysia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>China</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>United States</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brazil</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colombia</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peru</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ecuador</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guatemala</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canada</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Venezuela</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iran</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thailand</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vietnam</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indonesia</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Philippines</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>South Korea</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>North Korea</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>China</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Zealand</td>
<td>China</td>
<td>China</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Smallest Pop:** Sao Tome and Principe, Tuvalu, Nauru, Palau, Marshall Islands, Greenland

**Largest Pop:** China, United States, India, Brazil, Indonesia, Pakistan, Bangladesh, Nigeria, Ethiopia, Russia, Japan, Mexico, South Africa
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.

```r
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" "#FD8E61" "#FEE090"

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

$Europe
[1] "#276419" "#4D9221" "#7BC41" "#B8E186" "#E6F5D0"

$Oceania
[1] "#313695" "#4575B4"
### turn those into a palette big enough to cover each country in a continent

```r
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.

*Takeaway #2:*

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

*There’s a reason I use `lapply` in this way but let’s stay focused on the colors.*
## turn those into a palette big enough to cover each country in a continent

```r
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.
## 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.
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
This interpolation / expansion is what `colorRampPalette()` helps you to do.
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 country colors hold.
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.

```r
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)
```

```r
## 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)

<table>
<thead>
<tr>
<th>continent</th>
<th>country</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Africa</td>
<td>South Africa</td>
<td>#8F4407</td>
</tr>
<tr>
<td>Africa</td>
<td>Cote d'Ivoire</td>
<td>#B75C07</td>
</tr>
<tr>
<td>Africa</td>
<td>Malawi</td>
<td>#C2650A</td>
</tr>
<tr>
<td>Africa</td>
<td>Guinea</td>
<td>#E18417</td>
</tr>
<tr>
<td>Africa</td>
<td>Sao Tome and Principe</td>
<td>#FEE0B6</td>
</tr>
<tr>
<td>Asia</td>
<td>Pakistan</td>
<td>#540F60</td>
</tr>
<tr>
<td>Asia</td>
<td>Sri Lanka</td>
<td>#AD8ABD</td>
</tr>
</tbody>
</table>

> peek(gDat)

<table>
<thead>
<tr>
<th>country</th>
<th>continent</th>
<th>year</th>
<th>pop</th>
<th>lifeExp</th>
<th>gdpPercap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bulgaria</td>
<td>Europe</td>
<td>1992</td>
<td>8658506</td>
<td>71.190</td>
<td>6302.6234</td>
</tr>
<tr>
<td>Burkina Faso</td>
<td>Africa</td>
<td>1957</td>
<td>4713416</td>
<td>34.906</td>
<td>617.1835</td>
</tr>
<tr>
<td>Germany</td>
<td>Europe</td>
<td>1982</td>
<td>7833526</td>
<td>73.800</td>
<td>22031.5327</td>
</tr>
<tr>
<td>Israel</td>
<td>Asia</td>
<td>2007</td>
<td>6426679</td>
<td>80.745</td>
<td>25523.2771</td>
</tr>
<tr>
<td>Italy</td>
<td>Europe</td>
<td>2002</td>
<td>5792699</td>
<td>80.240</td>
<td>27968.0982</td>
</tr>
<tr>
<td>Korea, Rep.</td>
<td>Asia</td>
<td>1957</td>
<td>22611552</td>
<td>52.681</td>
<td>1487.5935</td>
</tr>
<tr>
<td>Tanzania</td>
<td>Africa</td>
<td>1982</td>
<td>19844382</td>
<td>50.608</td>
<td>7092.923</td>
</tr>
</tbody>
</table>

> gDat <- merge(gDat, countryColors)

> peek(gDat)

<table>
<thead>
<tr>
<th>country</th>
<th>continent</th>
<th>year</th>
<th>pop</th>
<th>lifeExp</th>
<th>gdpPercap</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belgium</td>
<td>Europe</td>
<td>1952</td>
<td>8730405</td>
<td>68.000</td>
<td>8343.105</td>
<td>#6DAD35</td>
</tr>
<tr>
<td>Central African Republic</td>
<td>Africa</td>
<td>1962</td>
<td>1523478</td>
<td>39.475</td>
<td>1193.069</td>
<td>#F5AA4E</td>
</tr>
<tr>
<td>Jamaica</td>
<td>Americas</td>
<td>1987</td>
<td>2326606</td>
<td>71.770</td>
<td>6351.237</td>
<td>#FDD788</td>
</tr>
<tr>
<td>Norway</td>
<td>Europe</td>
<td>1982</td>
<td>4114787</td>
<td>75.970</td>
<td>26298.635</td>
<td>#B9E188</td>
</tr>
<tr>
<td>Oman</td>
<td>Asia</td>
<td>1952</td>
<td>507833</td>
<td>37.578</td>
<td>1828.230</td>
<td>#D9C2DE</td>
</tr>
<tr>
<td>Tunisia</td>
<td>Africa</td>
<td>2007</td>
<td>10276158</td>
<td>73.923</td>
<td>7092.923</td>
<td>#DA7D12</td>
</tr>
<tr>
<td>Vietnam</td>
<td>Asia</td>
<td>2007</td>
<td>85262356</td>
<td>74.249</td>
<td>2441.576</td>
<td>#6F247B</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>country</th>
<th>continent</th>
<th>year</th>
<th>pop</th>
<th>lifeExp</th>
<th>gdpPercap</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belgium</td>
<td>Europe</td>
<td>1952</td>
<td>8730405</td>
<td>68.000</td>
<td>8343.105</td>
<td>#6DAD35</td>
</tr>
<tr>
<td>Central African Republic</td>
<td>Africa</td>
<td>1962</td>
<td>1523478</td>
<td>39.475</td>
<td>1193.069</td>
<td>#F5AA4E</td>
</tr>
<tr>
<td>Jamaica</td>
<td>Americas</td>
<td>1987</td>
<td>2326606</td>
<td>71.770</td>
<td>6351.237</td>
<td>#FDD788</td>
</tr>
<tr>
<td>Norway</td>
<td>Europe</td>
<td>1982</td>
<td>4114787</td>
<td>75.970</td>
<td>26298.635</td>
<td>#B9E188</td>
</tr>
<tr>
<td>Oman</td>
<td>Asia</td>
<td>1952</td>
<td>507833</td>
<td>37.578</td>
<td>1828.230</td>
<td>#D9C2DE</td>
</tr>
<tr>
<td>Tunisia</td>
<td>Africa</td>
<td>2007</td>
<td>10276158</td>
<td>73.923</td>
<td>7092.923</td>
<td>#DA7D12</td>
</tr>
<tr>
<td>Vietnam</td>
<td>Asia</td>
<td>2007</td>
<td>85262356</td>
<td>74.249</td>
<td>2441.576</td>
<td>#6F247B</td>
</tr>
</tbody>
</table>
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.