Creatings Plots & Graphs in R

Joshua Wiley

Health Psychology Program
University of California, Los Angeles
wileyj@ucla.edu

Thursday, 26 April, 2012

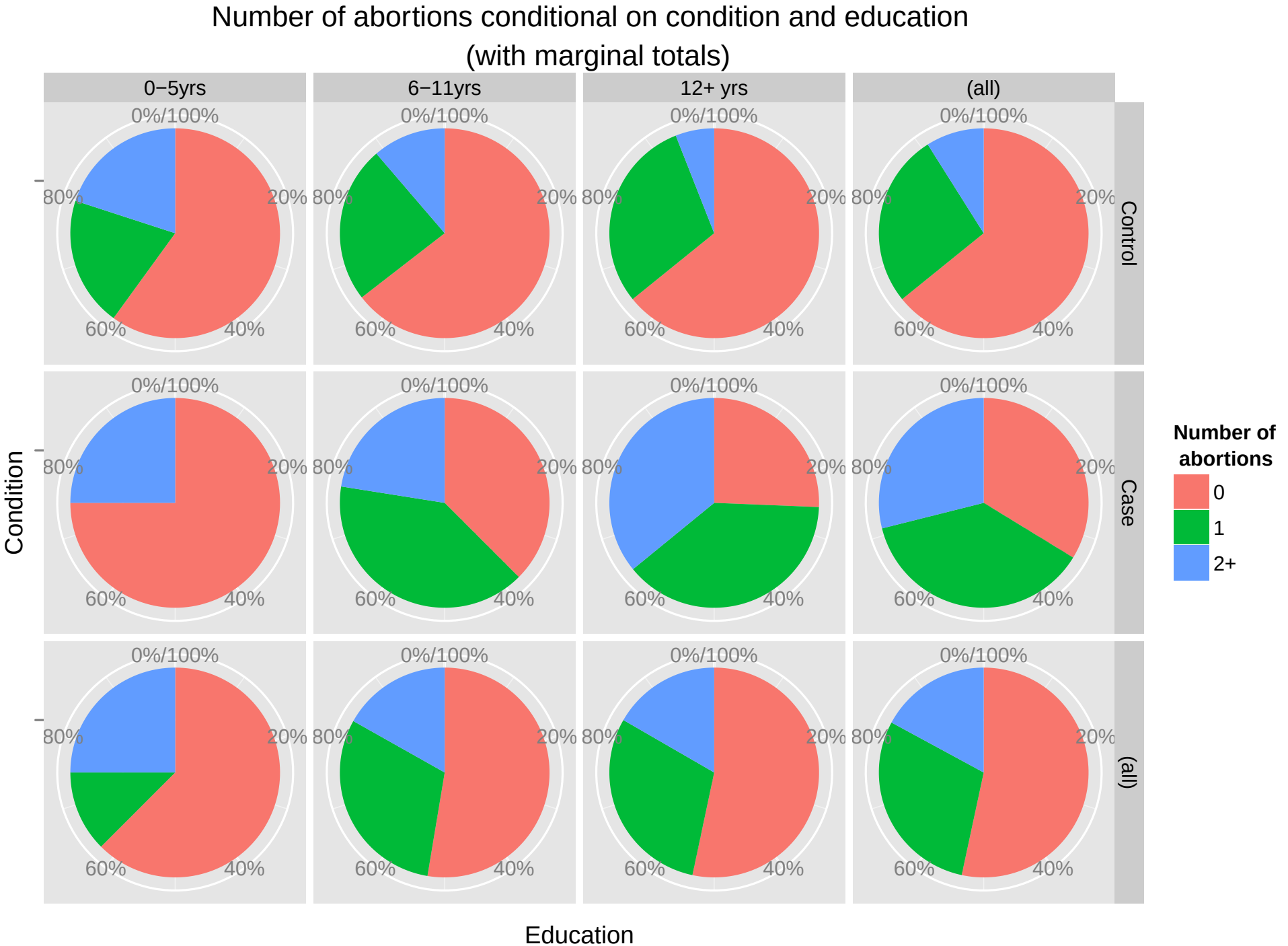

Number of abortions conditional on condition and education
(with marginal totals)
0−5yrs
6−11yrs
12+ yrs
(all)
Control
Case
(all)
Condition
Education
0%/100%
20%
40%
60%
80%
Number of abortions
0
1
2+

Table of Contents

Getting Started I

R is composed of a core set of packages that provide infrastructure and basic functionality and over 3,671 packages. Most regular R users use some interface. Popular ones include Rstudio (which I will demonstrate here) and Emacs + ESS. These provide features to make working with R easier and more efficient.

R	http://www.r-project.org/
Rstudio	http://rstudio.org/
Emacs	http://www.gnu.org/software/emacs/
ESS	http://ess.r-project.org/
Emacs+ESS	http://vgoulet.act.ulaval.ca/en/emacs/

(a prebuilt version of Emacs and ESS for Windows or Mac)

Getting Started II

Tutorial 1 http://www.burns-stat.com/pages/tutorials.html

read the tutorials for R beginning ($\sim$ 1 day)

Tutorial 2 http://cran.r-project.org/doc/manuals/R-intro.pdf

(through Appendix A) ($\sim$ 1 day)

Tutorial 3 http://www.burns-stat.com/pages/tutorials.html

intermediate guide, "The R Inferno" ($\sim$ 1-3 days)

Please stop me if you have questions!

R Basics

- Nearly everything in R is an *object*
- Objects can be assigned using the assignment operator, "<-"
- *Objects have classes* ranging from basic to complex, such as
  - numeric vectors
  - a class that contains coordinates (latitude, longitude, elevation) and time
  - a graphical object containing all necessary information to produce a graph
- Functions do things, and *functions have methods*
- Depending on the *class* of an object passed to a function, different methods will be dispatched.

Examples

An object's class affects how a function behaves

```
> x <- c(1, 2, 3, 1, 2, 3)
> class(x)
[1] "numeric"
> print(x)
[1] 1 2 3 1 2 3
> x <- factor(x)
> class(x)
[1] "factor"
> print(x)
[1] 1 2 3 1 2 3
Levels: 1 2 3
```

R Basics II

- This is just Object Oriented Programming (OOP). Whether numbers are numeric or factor class can dramatically change the graph created.
- The most common data structure in R is a data frame; a matrix where each column can be a vector of a different class.
- `str()` is a useful function to show the **str**ucture of an object. We could look at the data frame (called **infert**) that the pie charts were based on.

Examples II

```
> str(infert)

'data.frame': 248 obs. of  8 variables:
 $ education     : Factor w/ 3 levels "0-5yrs","6-11yrs",..: 1
 $ age           : num  26 42 39 34 35 36 23 32 21 28 ...
 $ parity        : num  6 1 6 4 3 4 1 2 1 2 ...
 $ induced       : num  1 1 2 2 1 2 0 0 0 0 ...
 $ case          : Factor w/ 2 levels "Control","Case": 2 2 2
 $ spontaneous   : Factor w/ 3 levels "0","1","2+": 3 1 1 1 2
 $ stratum       : int  1 2 3 4 5 6 7 8 9 10 ...
 $ pooled.stratum: num  3 1 4 2 32 36 6 22 5 19 ...
```

R Graphics

There are three main graphing systems in R

- traditional graphics (included in the core R packages)
- lattice (in the **lattice** package)
- ggplot2 (in the **ggplot2** package)

I am going to focus on ggplot2 in this presentation. For a more detailed overview, I suggest *R Graphics* by Paul Murrell (2nd Ed.).

Table of Contents

ggplot2 background I

- Before we dive into examples of graphics in `ggplot2`, I want to give a bit of background.
- `ggplot2` is based on the **g**rammar of **g**raphics, a framework for creating graphs.
- The idea is that graphics or data visualization generally can be broken down into basic low level pieces and then combined, like language, into a final product.

ggplot2 background II

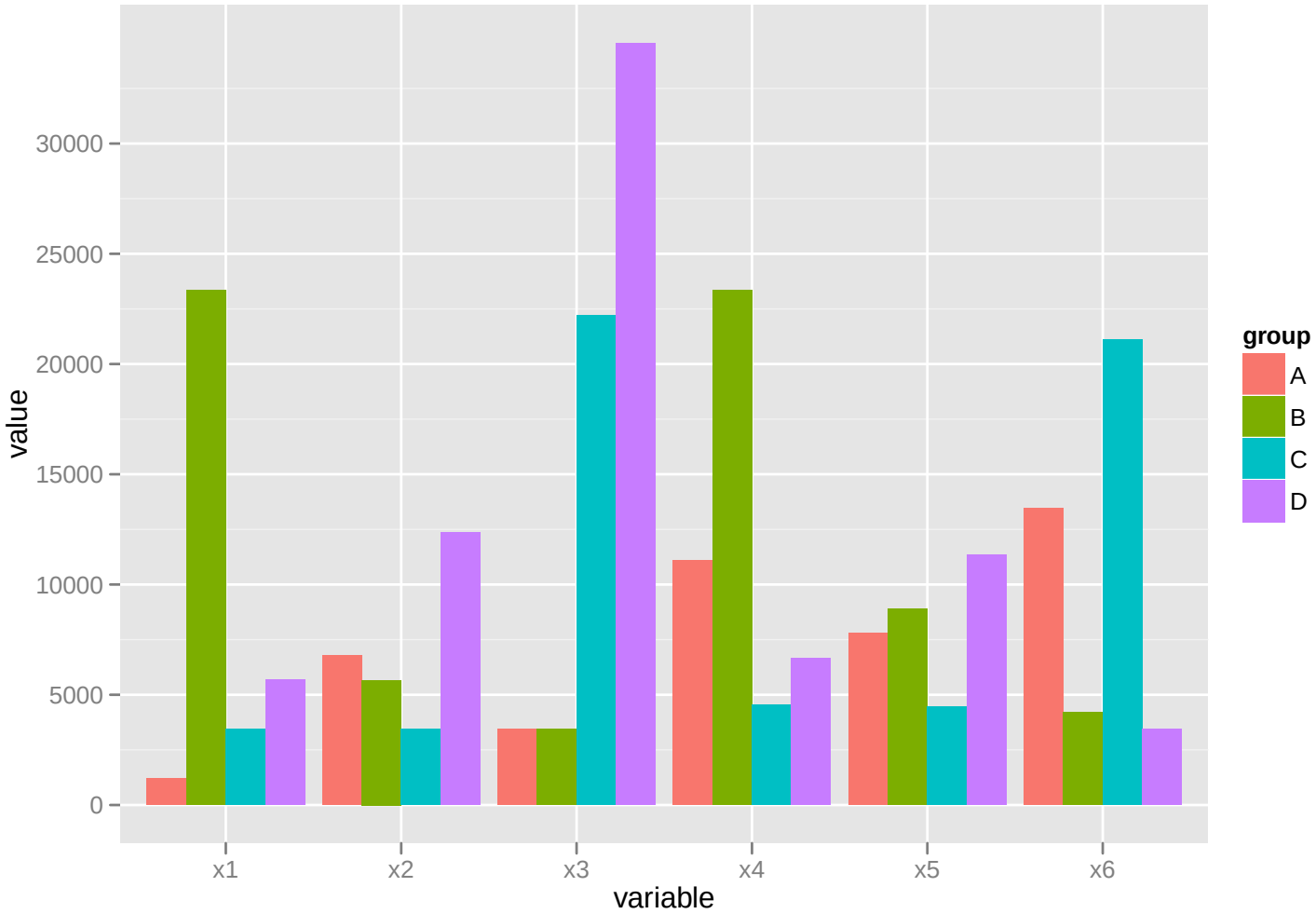
- For example, this sentence is valid: "This is a lamp." but the sentence can be modified without recreating it: "This is a red lamp.".
- This powerful concept allows basic aspects of graphs to be reused and slight modifications added rather than starting from scratch each time.
- Under this system, line plots and scatter plots are essentially the same. Both have data mapped to the x and y axes. The difference is the plotting symbol (called a **geom** in `ggplot2`) is a point or line. The data, axes, labels, titles, etc. can be identical in both cases.

Dodged Bar Graph: Code

```
> require(rJava)
> require(xlsx)
> require(reshape2)
> dat <- read.xlsx(file = "barchart.xlsx", 1)
> ldat <- melt(dat)
> colnames(ldat)[1] <- "group"
> p <- ggplot(ldat,
+   aes(x = variable, y = value, fill = group)) +
+   geom_bar(stat = "identity", position = "dodge")
```

Dodged Bar Graph: Graph

```
> print(p); presplots <- list(bar1 = p)
```



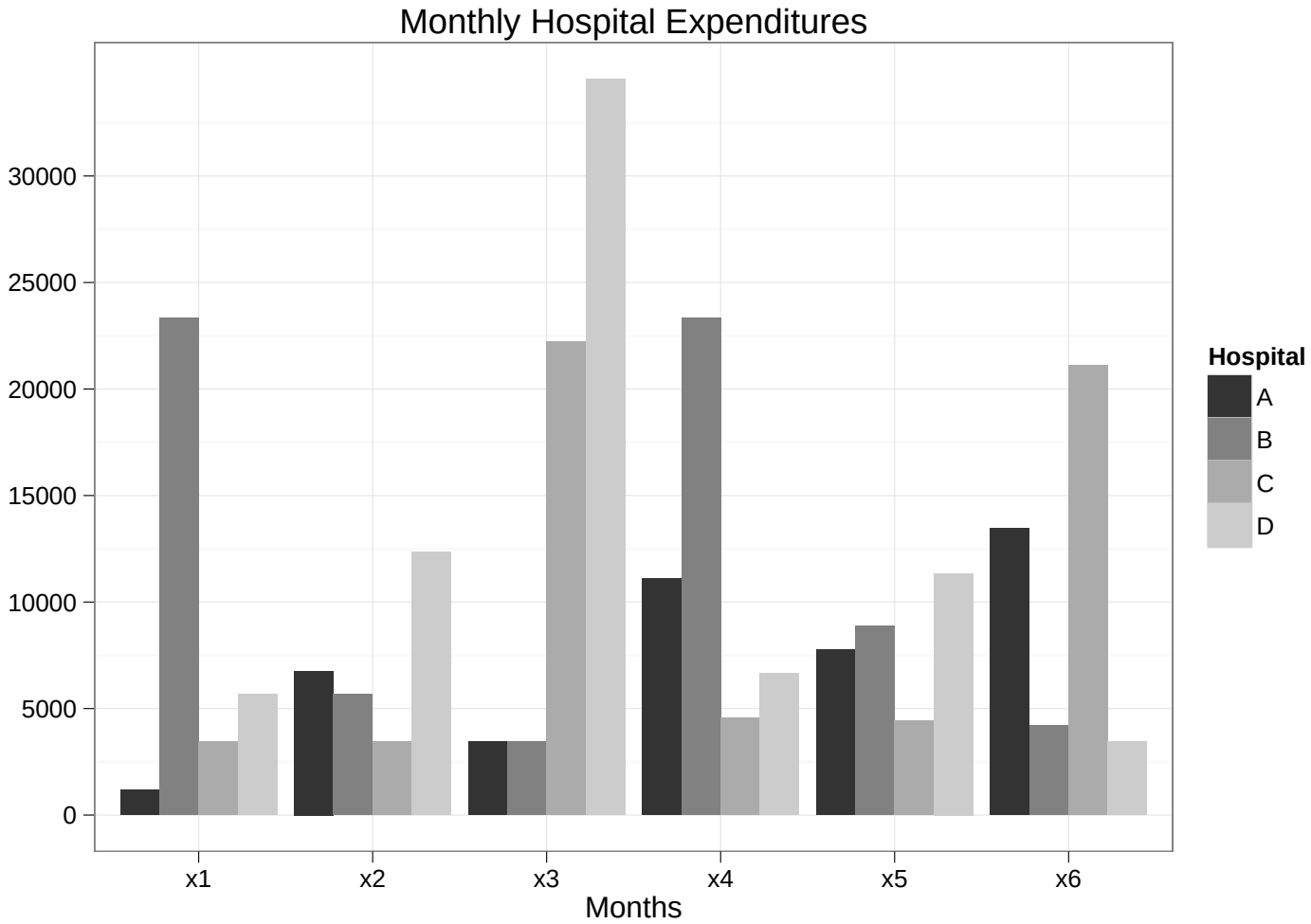
Customizing Labels: Code

We can customize the barplot adding our own axis labels, changing the colour scheme, adding an overall title, and changing the theme to black and white.

```
> p <-
+   p +
+   labs(x = "Months", y = "") +
+   scale_fill_grey(name = "Hospital") +
+   opts(title = "Monthly Hospital Expenditures") +
+   theme_bw()
```

Customizing Labels: Graph

```
> print(p)
```



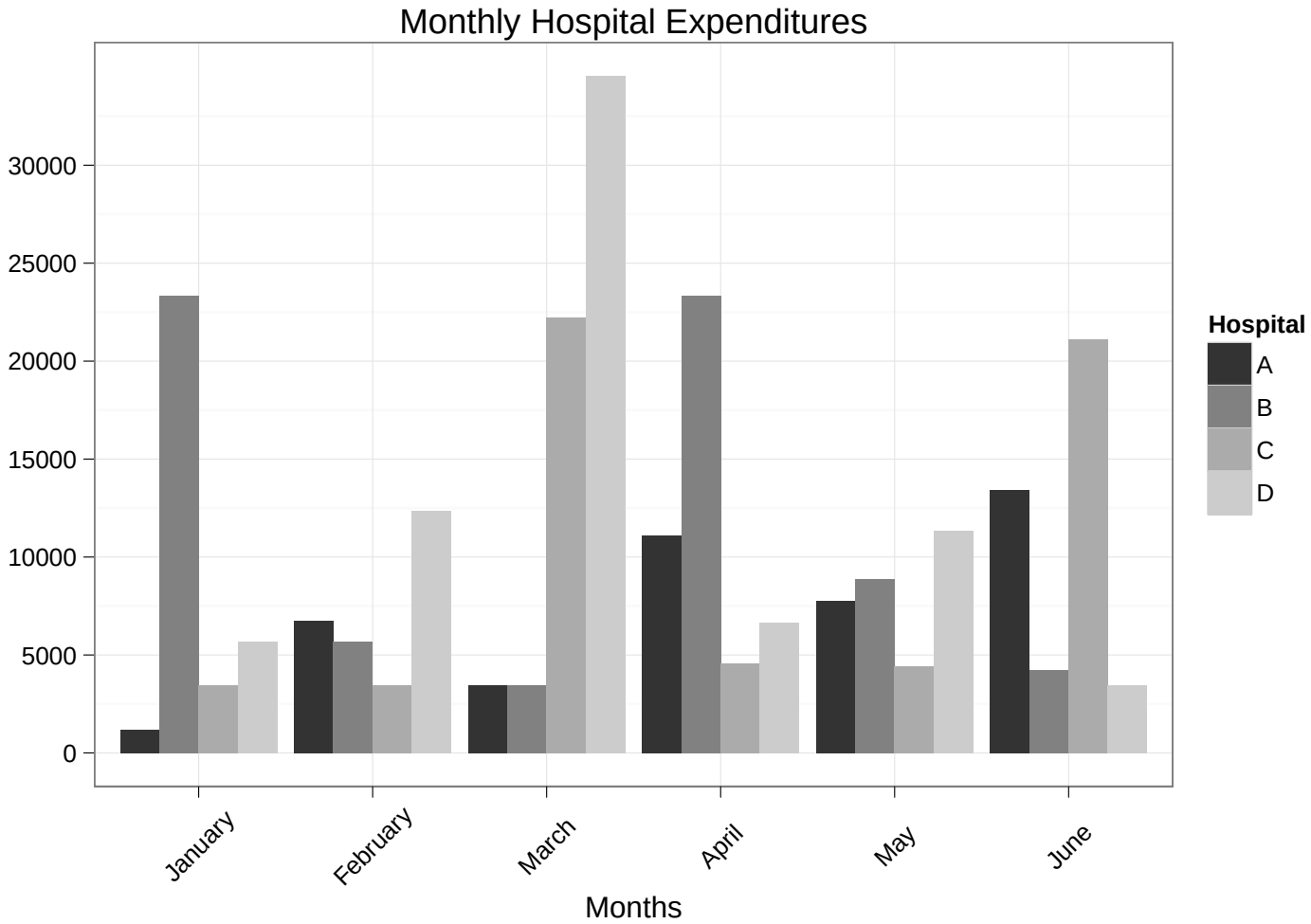
Customizing Labels II: Code

- This demonstrates how you can even label strange things (e.g., "x1 - x6") by creating dates and extracting the month names.
- The code looks complex because the original data were called "x1 - x6". Had it been date class data (e.g., "2012/6/2"), it would be easy to make the labels just the month names.

```
> p <-
+   p +
+   scale_x_discrete(breaks = paste0("x", 1:6), labels =
+   months(as.Date(paste0("2012/", 1:6, "/01"), "%Y/%m/%d")))
+   opts(axis.text.x = theme_text(angle = 45))
```

Customizing Labels II: Graph

```
> print(p)
```

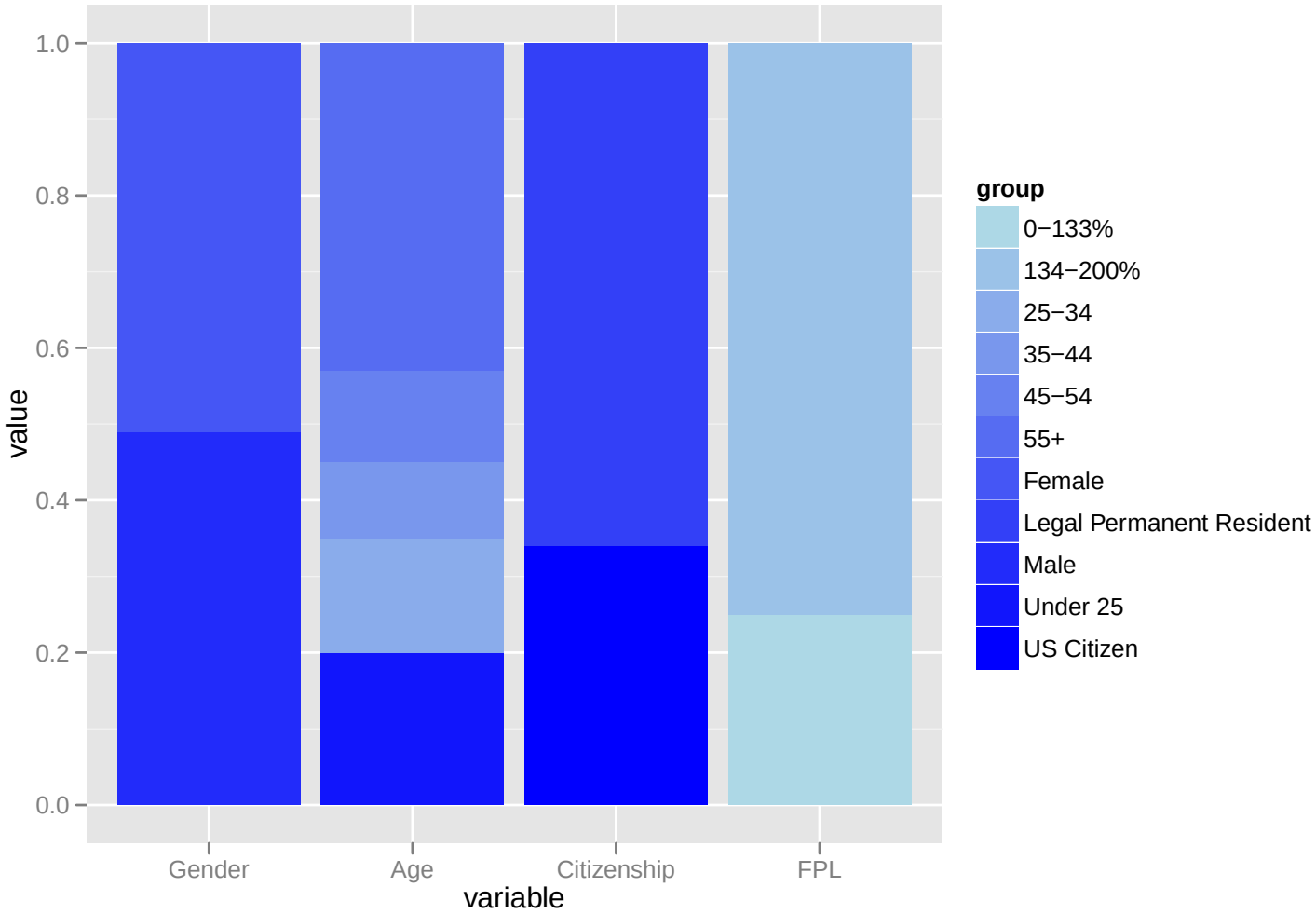


Stacked Bar: Code

```
> dat2 <- read.xlsx(file = "barchart2.xlsx", 1)
> ldat2 <- na.omit(melt(dat2))
> colnames(ldat2)[1] <- "group"
> cols <- colorRamp(c("lightblue", "blue"))(
+   seq(0, 1, length.out = 11))
> cols <- rgb(cols[, 1], cols[, 2], cols[, 3],
+   maxColorValue = 255)
> p <- ggplot(ldat2,
+   aes(x = variable, y = value, fill = group)) +
+   geom_bar(stat = "identity", position = "stack") +
+   scale_fill_manual(values = cols)
```

Stacked Bar: Graph

```
> print(p)
```



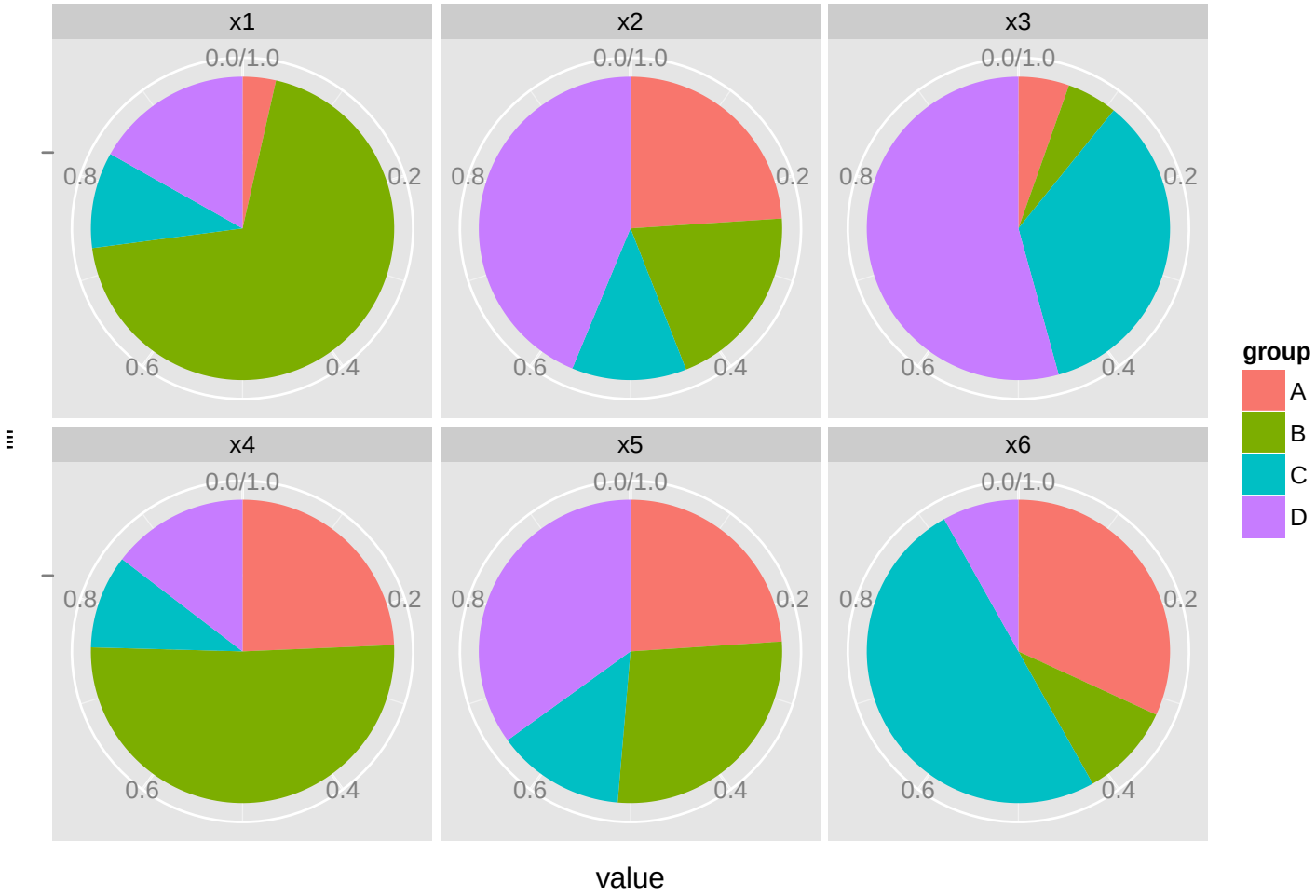
Pie Graph: Code

Remember this? Using polar coordinates, it is a pie chart.

```
> p <- ggplot(ldat,
+    aes(x = "", y = value, fill = group)) +
+    geom_bar(width=1, position = "fill") +
+    coord_polar(theta = "y") +
+    facet_wrap(~variable)
```

Pie Graph: Graph

```
> print(p)
```

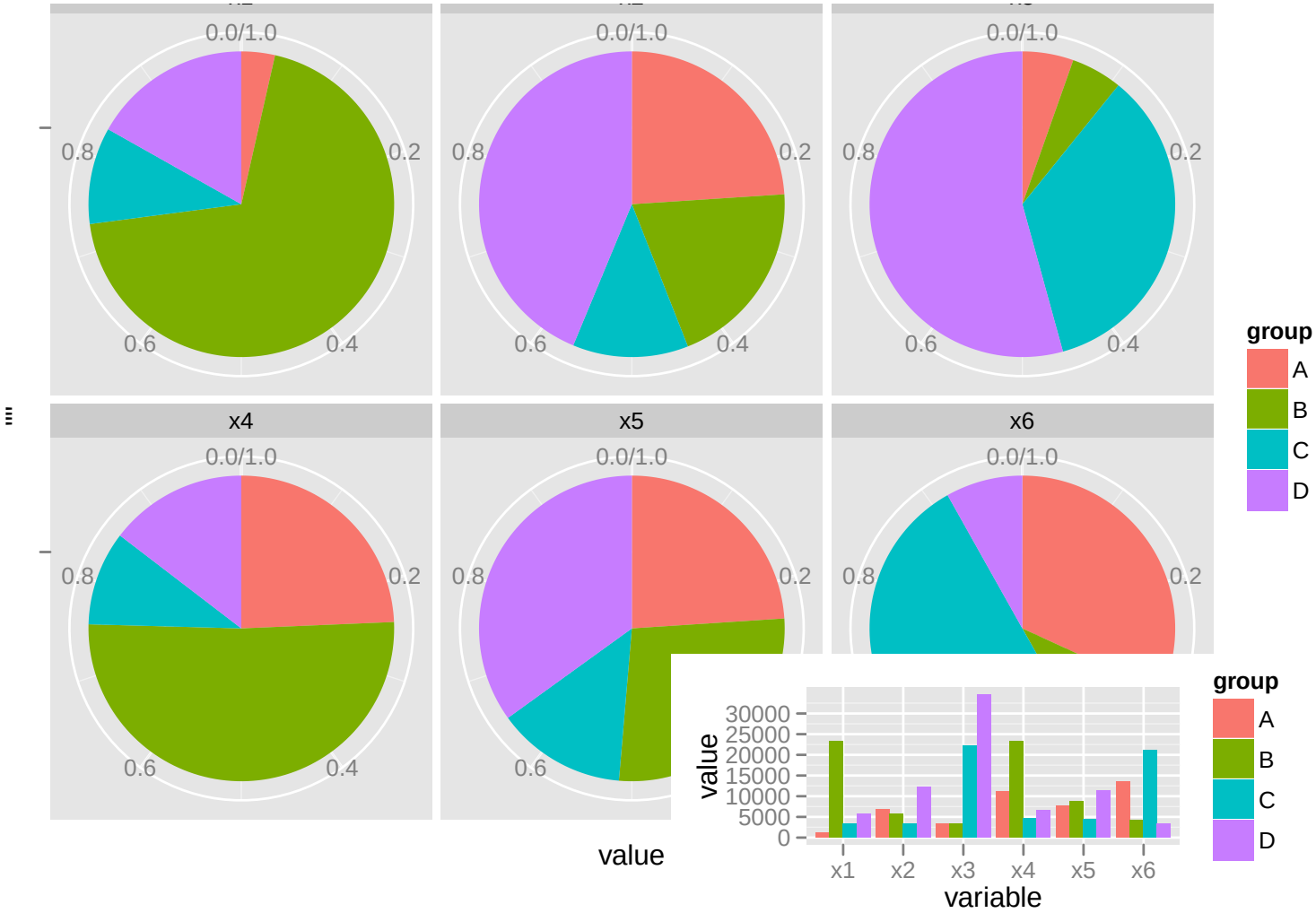


Pie Graph: Advanced Code

```
> require(grid) # has viewport function
> vp <- viewport(width = .5, height = .3, x = 1, y = 0,
+   just = c("right", "bottom"))
> p <- p +
+   opts(title = "Pie chart with Inset Bar chart\n",
+     plot.margin = unit(c(-5, 0, 0, 0), "lines"))
```

Pie Graph: Advanced Graph

```
> print(p); print(presplots$bar1, vp = vp)
```



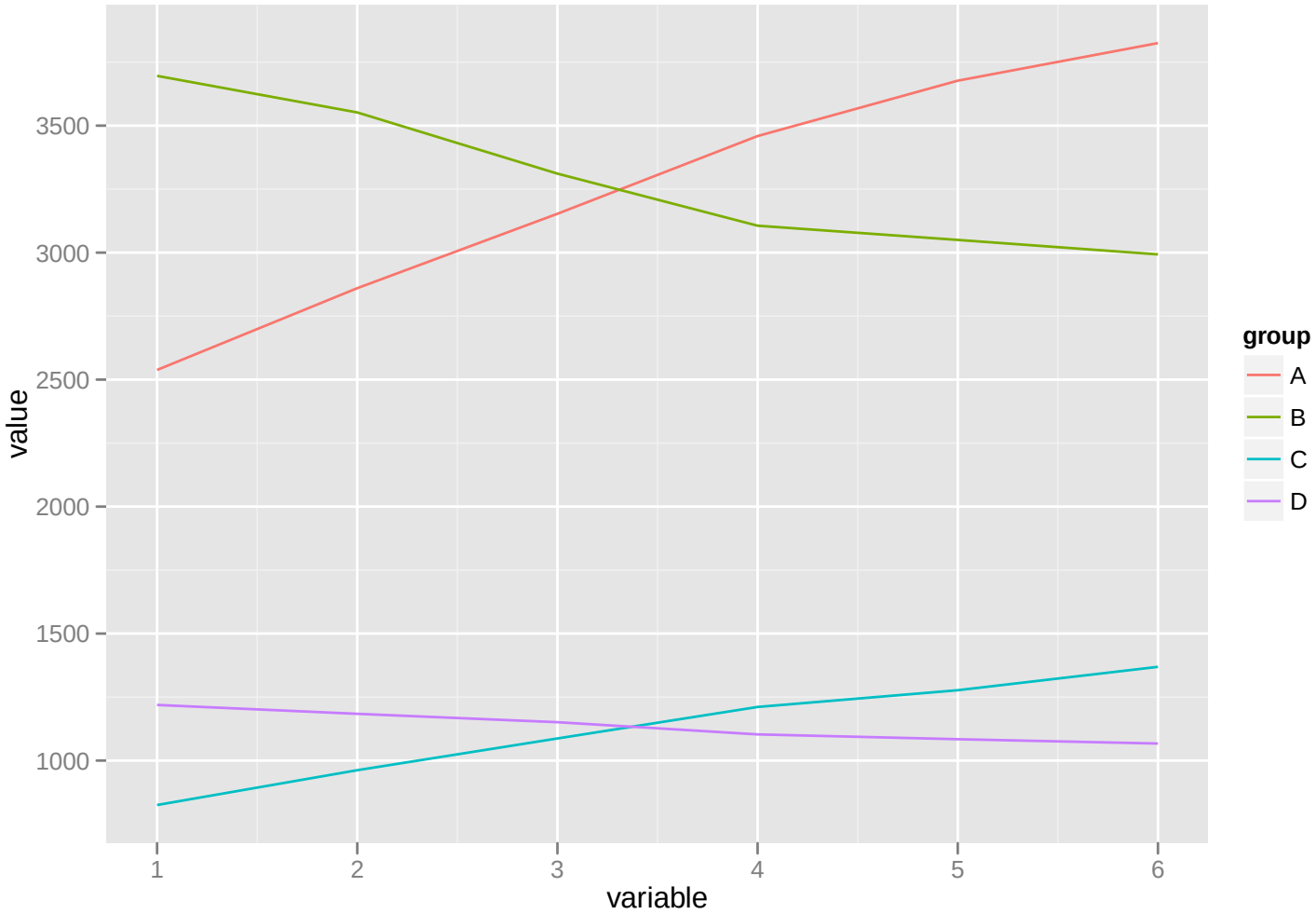
Line Chart: Code

Here we are going to look at line and scatter plots. We create a base plot, but do not add shapes (lines or points, so we can demonstrate them.

```
> dat3 <- read.xlsx(file = "linechart.xlsx",
+   1, header = FALSE)
> ldat3 <- melt(dat3)
> colnames(ldat3)[1] <- "group"
> ldat3$variable <- as.numeric(ldat3$variable)
> p <- ggplot(ldat3, aes(x = variable, y = value,
+   colour = group))
```

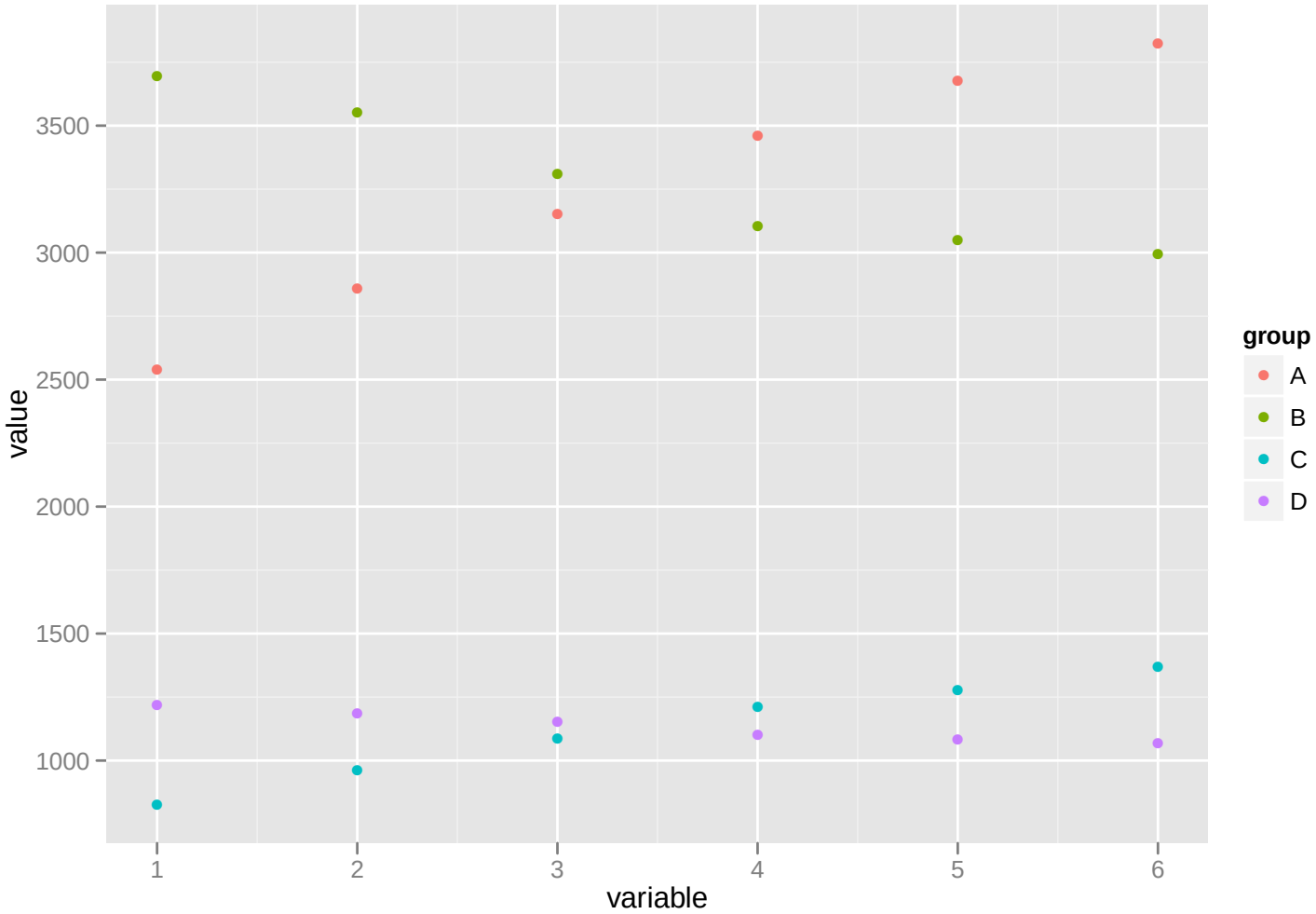
Line Chart: Graph

```
> print(p + geom_line())
```



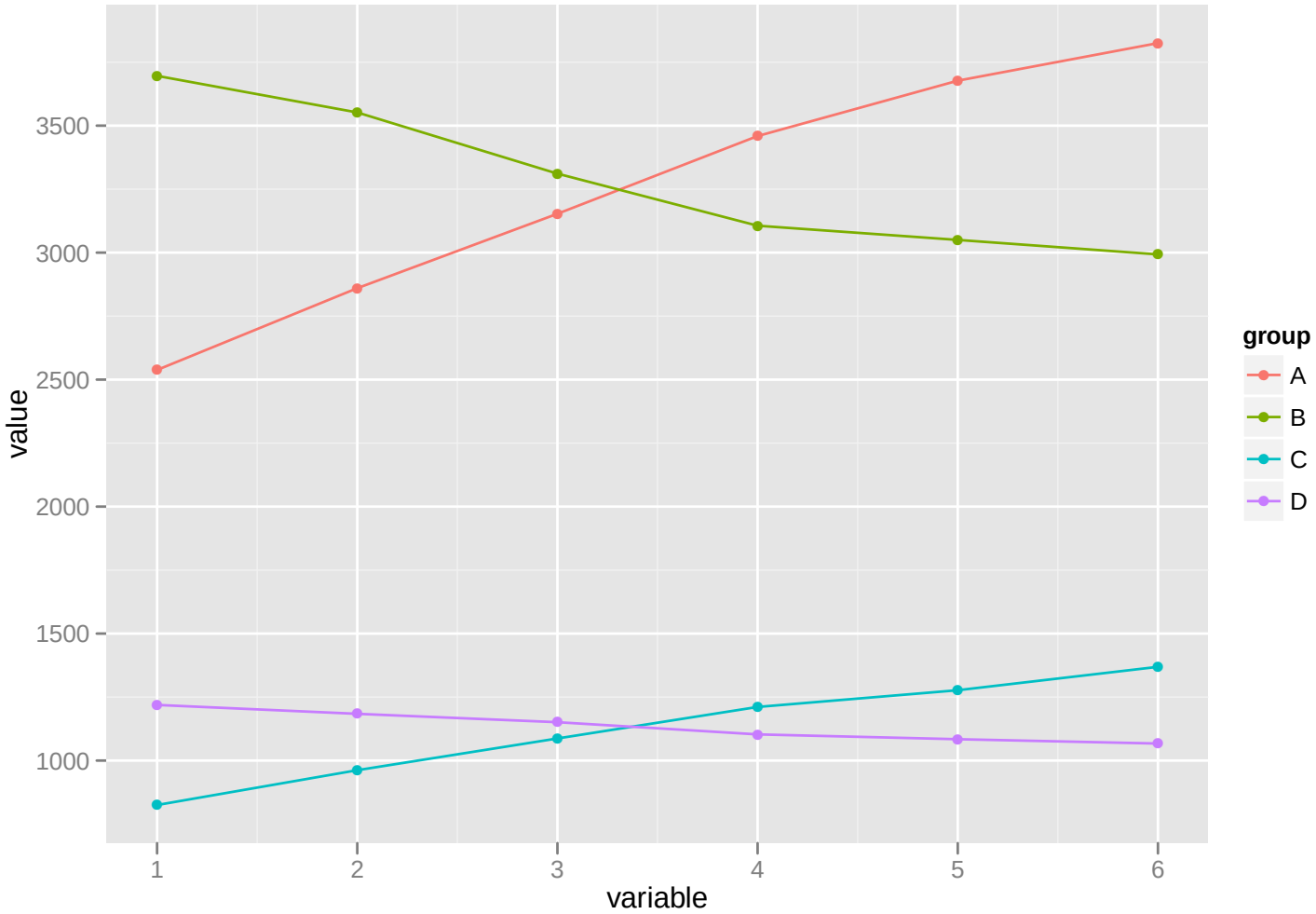
Scatter: Graph

```
> print(p + geom_point())
```



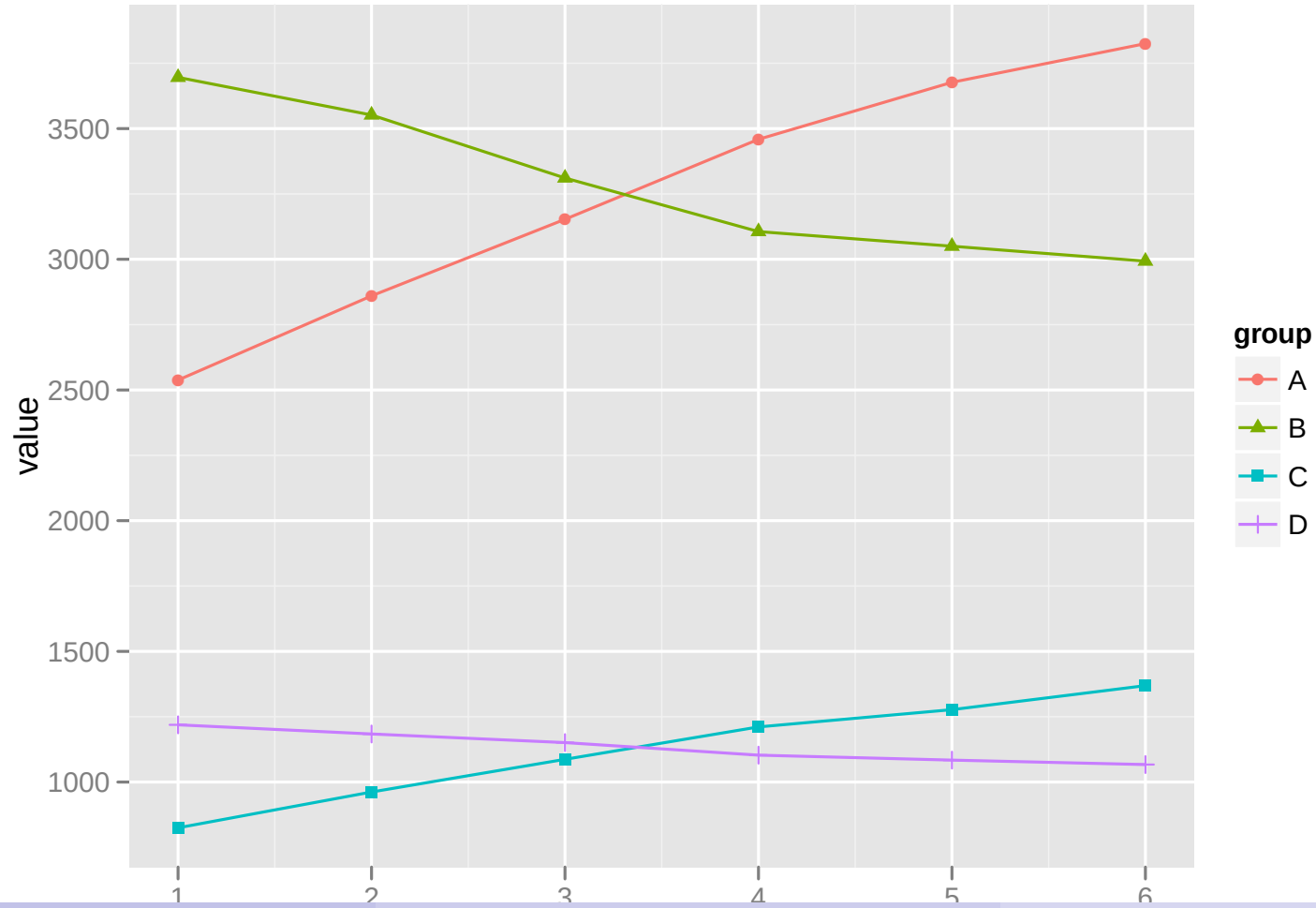
Scatter & Line: Graph

```
> print(p + geom_line() + geom_point())
```



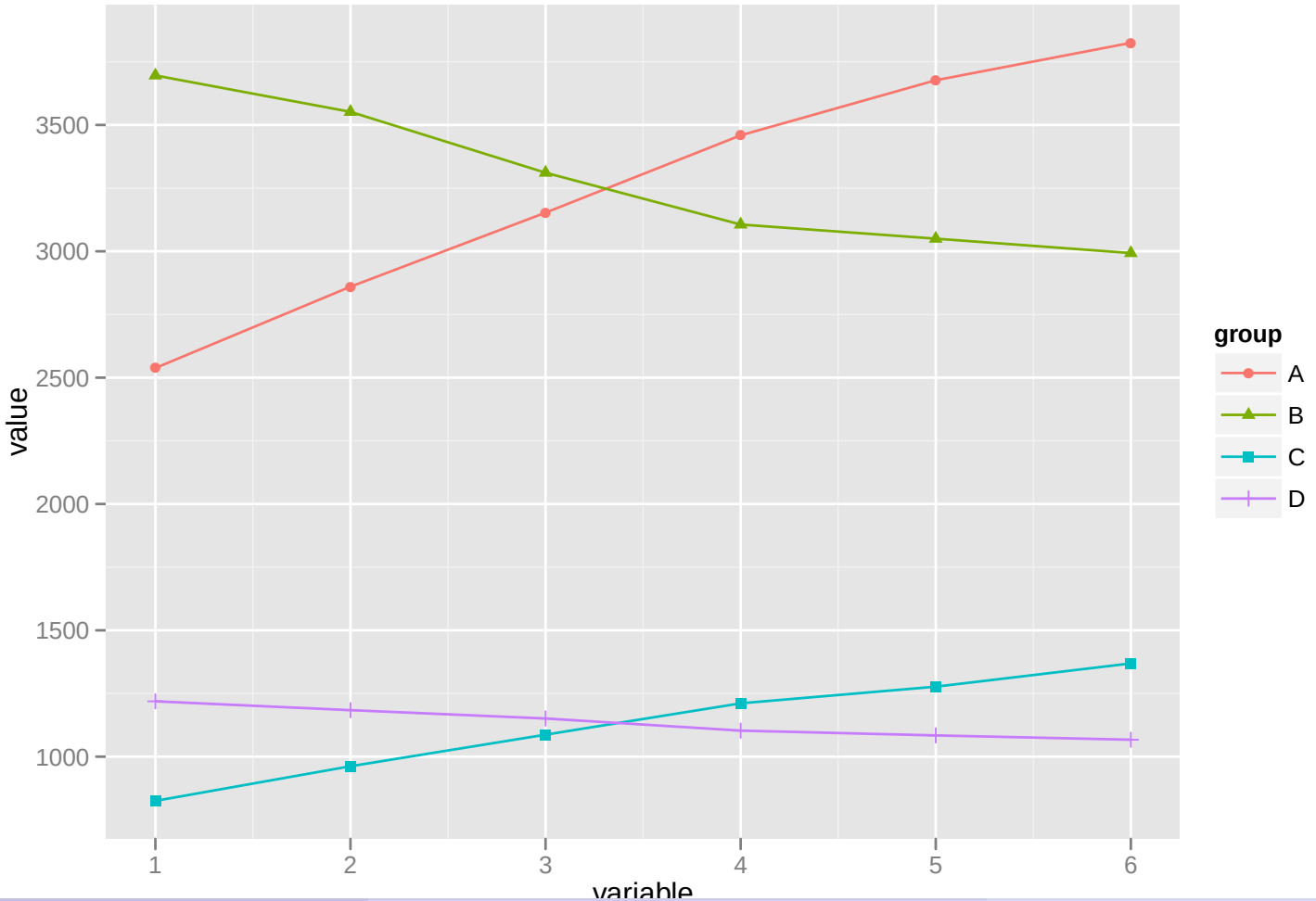
Scatter & Line: Graph

```
> p <- p + geom_line() + geom_point(aes(shape = group))
> print(p)
```



Customizing Legend: Graph

```
> p <- p + opts(legend.key.width = unit(1, "cm"))
> print(p)
```



Chloropleth Map: Data Setup I

```
> require(maps)
> require(mapproj)
> states <- map_data("state")[,
+   c("long", "lat", "group", "order", "region")]
> colnames(states)[5] <- "state"
> ## original data source
> ## dat <- read.csv("http://www.census.gov/popest/data/
> ##  national/totals/2011/files/NST_EST2011_ALLDATA.csv")
>
> dat <- read.csv("popdata.csv")
```

Chloropleth Map: Data Setup II

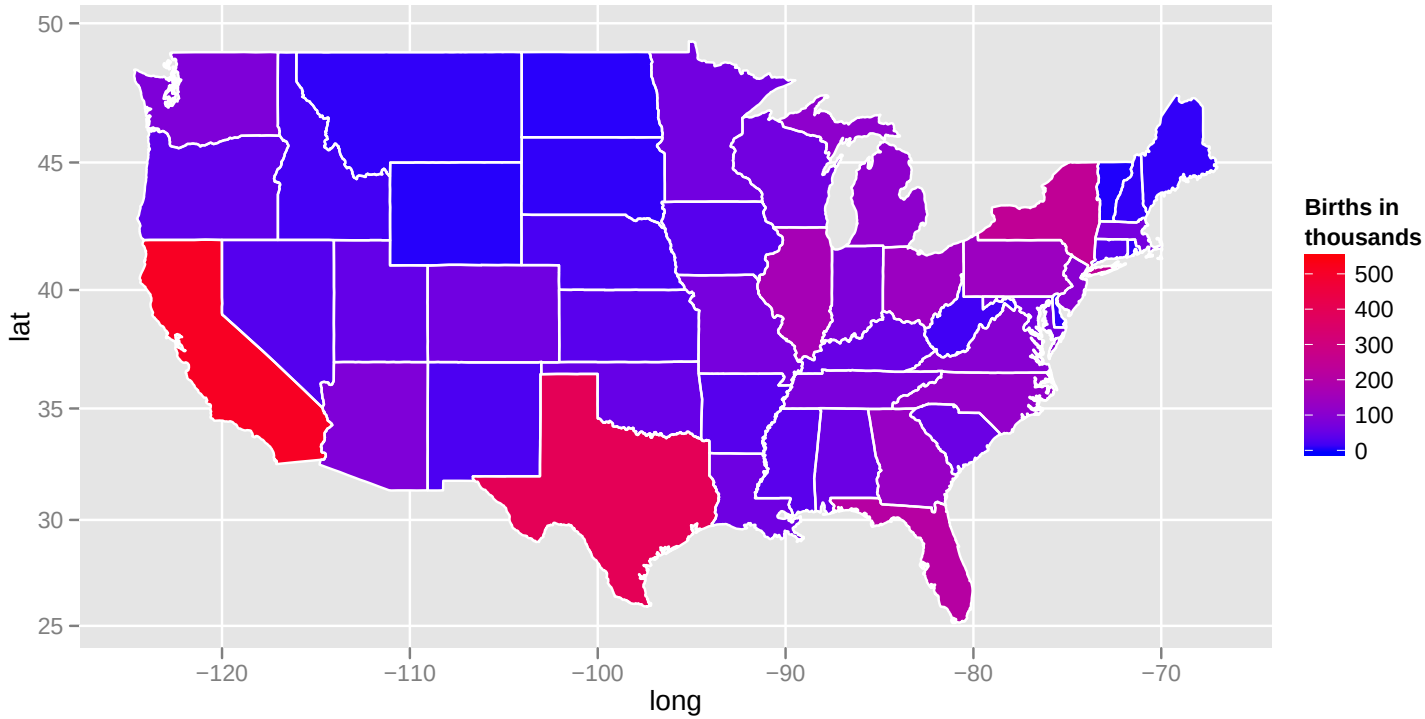
```
> births <- with(dat, {
+   data.frame(state = tolower(NAME),
+   kbirths2011 = BIRTHS2011/1000,
+   mpop2010 = CENSUS2010POP/1000000)
+ })
> births <- subset(births, state %in% states$state)
```

Chloropleth Map: Code

```
> chloropleth <- merge(states, births, "state")
> chloropleth <- chloropleth[order(chloropleth$order), ]
> p <- ggplot(chloropleth, aes(long, lat, group = group)) +
+   geom_polygon(aes(fill = kbirths2011)) +
+   geom_polygon(data = states, colour = "white", fill=NA) +
+   scale_fill_gradientn(name = "Births in\nthousands",
+     guide = guide_colorbar(),
+     colours = c("blue", "red"),
+     limits = c(0, 550)) +
+   coord_map(projection = "mercator")
```

Chloropleth Map: Graph

```
> print(p)
```



Chloropleth Map Advanced: Code

```
> tmp <- chloropleth[cumsum(rle(chloropleth$state)$lengths),]
> p <- p +
+   labs(x = "Longitude", y = "Latitude") +
+   geom_point(data = tmp,
+     aes(long, lat, size = mpop2010, group = 1)) +
+   scale_size_continuous(guide = FALSE, range = c(4, 10)) +
+   opts(title =
+   "2011 US Births, 2010 population shown in bubbles\n")
```

Chloropleth Map Advanced: Graph

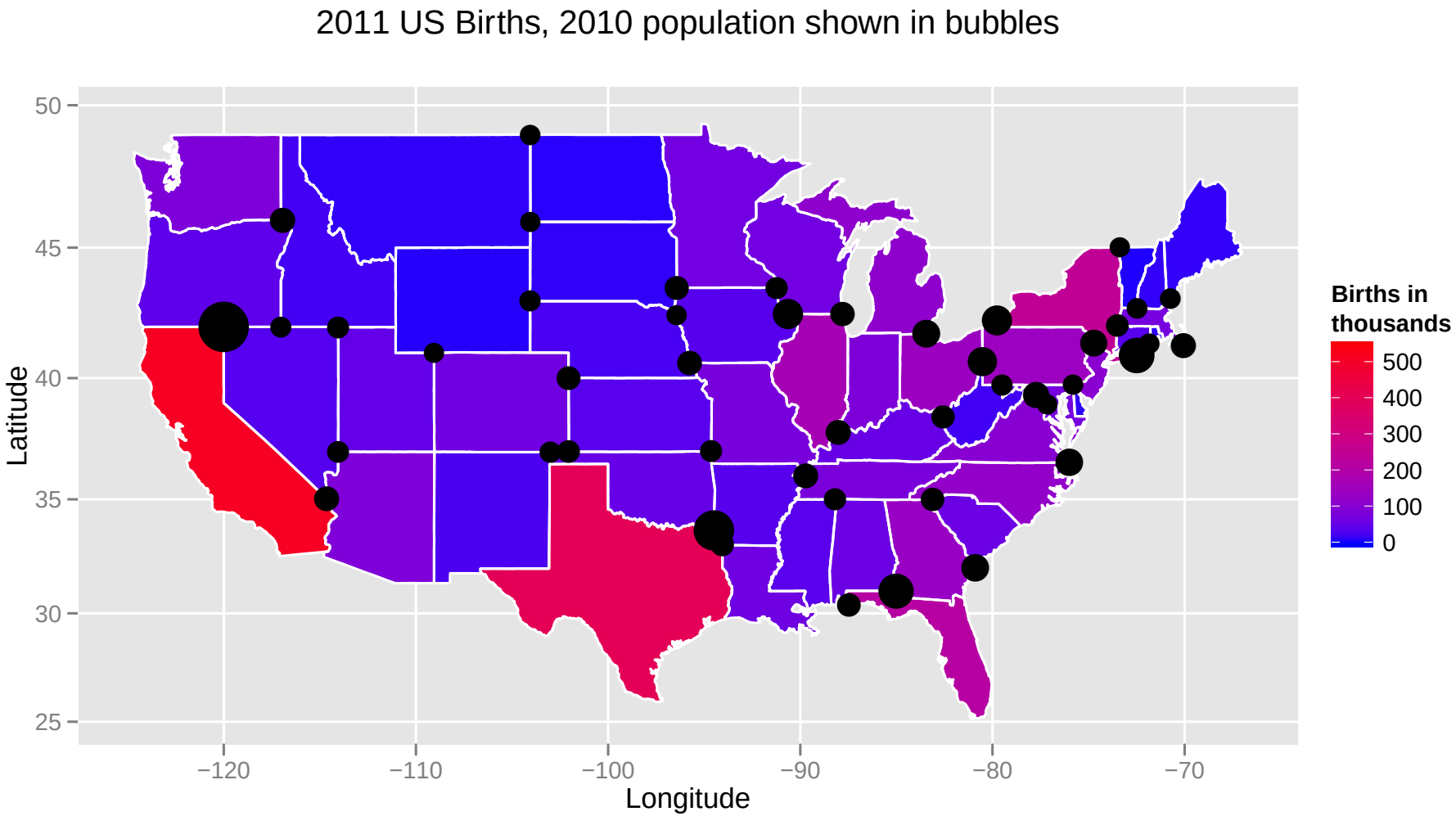


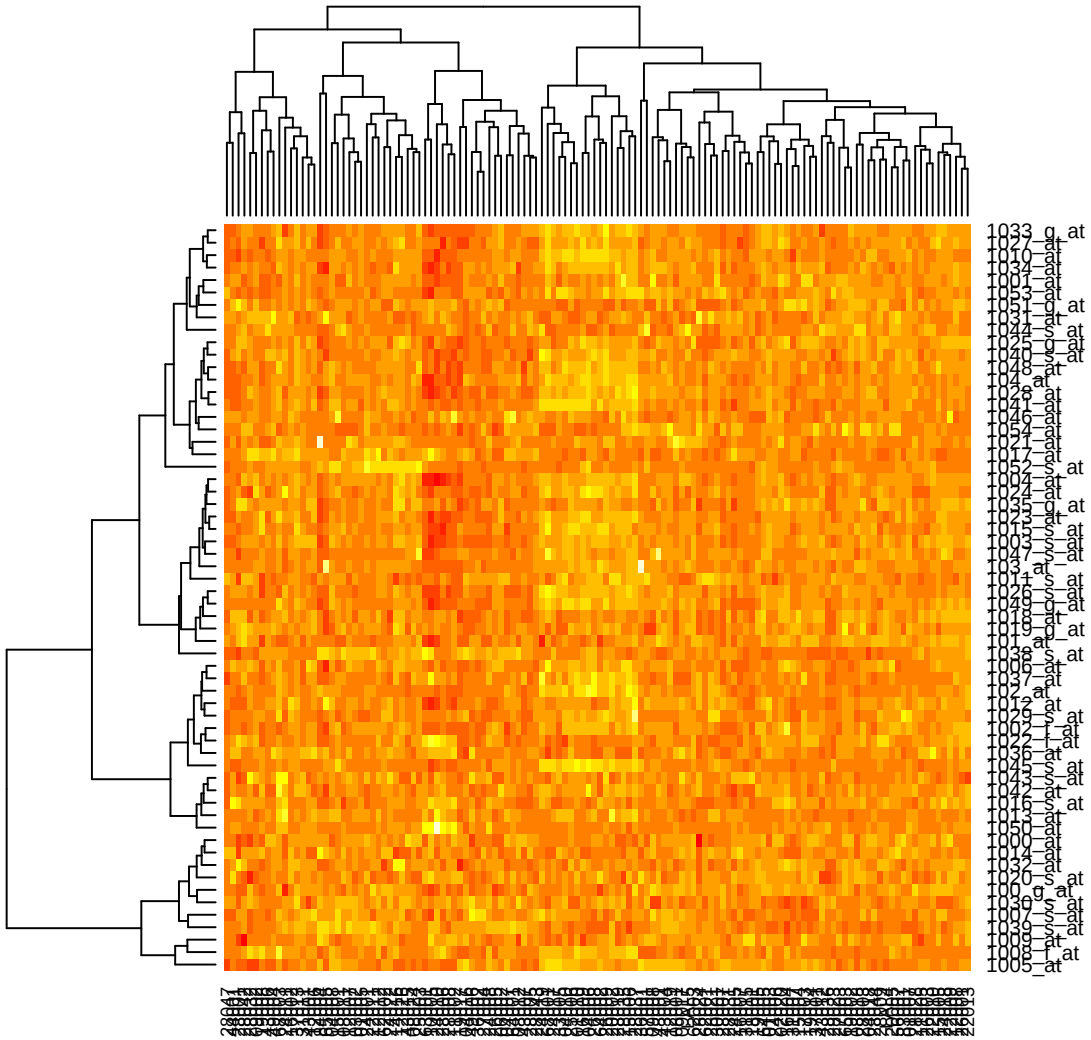
Table of Contents

Gene Heatmap: Code

```
> ## install.packages("BiocInstaller", repos =
> ## "http://www.bioconductor.org/packages/2.11/bioc")
> ## require(BiocInstaller)
> ## biocLite("ALL")
>
> require(ALL)
> require("gplots")
> data("ALL")
> x <- exprs(ALL)[1:60, ]
```

Gene Heatmap: Graph I

```
> heatmap(x)
```



Gene Heatmap: Graph IV

```
> heatmap.2(x, scale = "row", symkey=TRUE,
+    col = colorpanel(256, "blue", "black", "yellow"),
+    trace = "none", cexCol = 0.4, cexRow = 0.5)
```