

Effective Graphs with Microsoft R Open

Naomi B. Robbins and Joyce Robbins

Contents

1	Introduction	4
1.1	Why visualize data?	4
1.2	Why use R?	5
1.3	Why use Microsoft R Open?	6
1.4	How do I start using Microsoft R Open?	7
1.5	Which graphics package should I use?	7
1.6	How should I size and save my graphs?	8
1.7	What is an effective graph?	10
2	Direct Comparisons	11
2.1	Bar charts (base)	11
2.2	Dot plots (base)	13
3	Distributions	16
3.1	Histograms (base)	16
3.2	Box plots (base)	18
4	Trends Over Time	20
4.1	Line charts (base)	20
4.2	Month plots (base)	22
5	Relationships	23
5.1	Scatterplots (base)	23
5.2	Scatterplot matrices (lattice)	26
5.3	Parallel coordinate plots (MASS)	29
6	Percents... or Parts of a Whole	32
6.1	Pie charts (base)	32
6.2	Bar percent charts (base)	33
6.3	Multiple pie charts (base)	34
6.4	Divided bar charts (base)	36
6.5	Grouped bar charts (base)	38
6.6	Faceted bar charts (ggplot2)	39
7	Special Cases	44
7.1	Diverging stacked bar charts (HH)	44
7.2	Linked micromaps (micromapST)	47
8	Conclusion	50

A	Data and Sources	51
A.1	countries2012.csv	51
A.2	fathers.txt	52
A.3	living.csv	52
A.4	acs2014.csv	54
B	Base Graphics Cheatsheet	55
	References	57
	Index	58

Book web site

A Github site, <https://www.github.com/nbrgraphs/mro>, is available for downloading an electronic version of this document, code scripts for the graphs here, and additional code scripts for **ggplot2** versions of many of the graphs.

1 Introduction

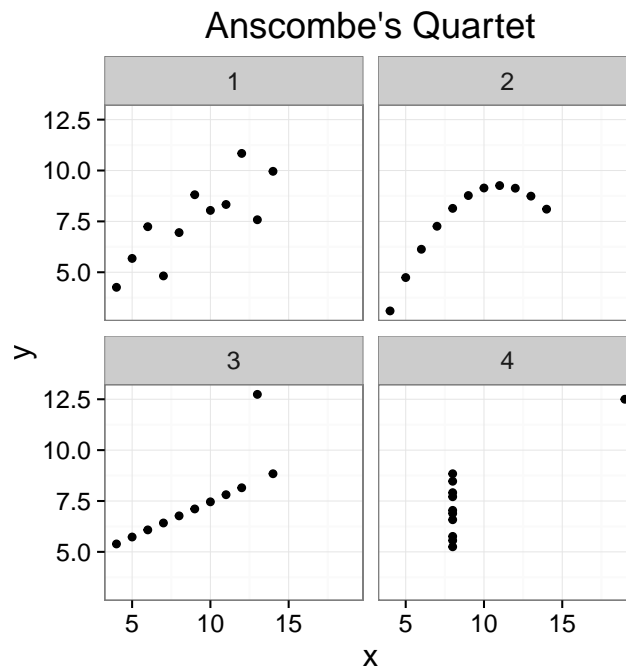
1.1 Why visualize data?

Graphs help us understand data in a way that simply cannot be matched with numbers and calculations alone. It's hard to find a data set that makes this case better than the one devised by Frank Anscombe to illustrate the power of visual representation. The set is one of R's many built-in data sets. It consists of 88 observations: four groups of eleven (x,y) pairs.

Summary statistics suggest that the groups are quite similar:

Property (by group)	Value
Mean of x	9 (exact)
Variance of x	11 (exact)
Mean of y	7.50 (to 2 decimal places)
Variance of y	4.122 or 4.127 (to 3 decimal places)
Correlation between x and y	0.816 (to 3 decimal places)
Linear regression line	$y = 3.00 + 0.500x$ (to 2 and 3 decimal places, respectively)

But if we graph the data, the results are striking:



In an instant each group appears distinct and unique, each clearly reflecting a different empirical reality. It is surely aha moments like this that led statistician and data visualization expert John Tukey to claim that his favorite part of analytics was "taking boring flat data and bringing it to life through visualization."

1.2 Why use R?

With computers we have the power to visualize data in ways that were unimaginable with only pencil and paper. There is a great variety of software packages and programming languages that provide the tools for drawing graphs. A bit of history is important for understanding one reason why R is so well-suited to the task. A dialect of the S language, R was developed by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand in the 1990s as an open source alternative to S-PLUS, the commercial version of S available at the time. The original S language was developed in the Statistical Research Department of Bell Laboratories, the same department in which William Cleveland and his colleagues studied how perception influences our ability to decode information from graphs.

Thanks to fruitful interaction between these projects, many graphing defaults in S were influenced by Cleveland’s research, and passed along to R. For this reason, it stands out among software options for drawing effective graphs.

R consists of a base language and user-contributed packages—of which over 8000 are currently available in the CRAN repository alone—that provide great depth and flexibility to the language. Packages can be conveniently explored on the [Microsoft R Portal](#).¹ There are R packages for most statistical methods and new ones are regularly being added. All of the code is publicly available so users can tweak it to their requirements and are not forced to accept preprogrammed options. This makes R very adaptable and extensible. Since R is a language rather than a program, what you can do is limited only by your imagination.

Furthermore, R users have formed an active and helpful community on and offline, a great resource for beginning and experienced programmers alike. User groups, blogs, R-Help, and Stack Overflow are a few of the many ways R users share knowledge and offer guidance. Package developers often respond directly to user questions and feedback, so the development of R is truly a collaborative effort. While many statistical and graphics software packages are very expensive, R is open source.

1.3 Why use Microsoft R Open?

With all these benefits, there are some deficits. The downside to having so many package authors is the lack of coordination among them. Many packages depend on others. Therefore, a change or update in one package may have a ripple effect, causing errors or faulty output. This can make it difficult to share code, or even rerun one’s own code at a later date. Another downside is R’s lack of ability to take advantage of parallel processing power.

Microsoft R Open (MRO), formerly Revolution R Open, was designed by Revolution Analytics to tackle these issues. It is an enhanced R distribution that standardizes the package landscape by fixing it in a particular point in time. Standardization provides consistency when installing packages, as all are downloaded from the same CRAN repository snapshot. Using a fixed snapshot allows for reproducibility over time and among users. It is

¹<https://mran.microsoft.com/packages>

a great relief to be able to reuse or share code without worrying about the compatibility of R packages. Reproducibility is further enhanced in MRO with the **checkpoint** package, which allows users to run R as it existed at a prior date. MRO is free and open source, and runs on Windows, Linux, and Mac systems.

To take advantage of the speed of multi-core processors, MRO provides optional multi-threaded math libraries. With these libraries, R makes use of all of the processing power available and computation times are significantly reduced. It is not necessary to change any code to benefit from the speed enhancements. Finally, MRO works well with RStudio: if MRO is installed, RStudio will automatically find it and use it as the R engine. Installing MRO doesn't change the way you interact with R at all. In fact, the only difference you'll notice is the Microsoft R Open message that appears after restarting R, indicating the version number and date of the default CRAN mirror snapshot.

1.4 How do I start using Microsoft R Open?

The first step is to [download Microsoft R Open](#).² We assume that the reader has a basic familiarity with the R language, but beginning R users can type in the examples here and run them, without any prior knowledge. This text focuses on graphical functions; for a general introduction to R, the Microsoft R Portal provides [a great list of resources for getting started](#).³ For data visualization, it is particularly important to understand how to work with data frames, vectors, and matrices, and to master the factor class. R Help can be easily accessed by entering a question mark before any R command or function in the console, for example: `?barplot`. Online R resources abound; [Stack Overflow](#)⁴ has become the go-to source for answers to R questions.

1.5 Which graphics package should I use?

An oft-asked question is whether to use the **graphics** package that comes with the R distribution, subsequently referred to as "base graphics," or an al-

²<https://mran.revolutionanalytics.com/download/>

³<https://mran.revolutionanalytics.com/documents/getting-started/>

⁴<http://stackoverflow.com>

ternative graphics package. Packages such as **ggplot2** or **lattice** are built on **grid** graphics, completely independent of the base graphics system. Therefore, it is not necessary to know base graphics in order to learn one of these **grid**-based packages. Starting with base graphics in our view, however, is worthwhile, as it will always be useful even if you end up doing most of your work with another package. For example, while **ggplot2** is quite powerful, it requires data to be in a data frame in a particular form. Therefore, for a quick plot of a vector, we are more likely to turn to base graphics and use functions like `plot()`, `barplot()`, `hist()`, etc. Base graphics is better as well for drawing highly specialized plots that combine a variety of different elements. We struggled with which to include here as both are popular choices, and decided to use base graphics in the main examples, but provide **ggplot2** code [on Github](#) for most of the plots.⁵ If you're learning **ggplot2** you may want to try replicating the plots on your own before looking at our code.

While the majority of the graphs here are drawn with base graphics, we also include some examples from **ggplot2**, **lattice**, **HH**, **MASS**, and **micromapST**. The learning curve for new packages varies widely. While **ggplot2** offers a new vocabulary and tools for constructing plots from the ground up, many packages are in reality collections of scripts that produce complex plots with minimal effort. By experimenting with functions from a variety packages, you will quickly expand your repertoire of graphs. At the same time, it's important to note that packages vary in their adherence to principles of good design. We encourage you not to assume that plot defaults are well-chosen, and override them as necessary to improve your graphs.

1.6 How should I size and save my graphs?

If no graphics device is specified, the default in R is to send the graphics to the on-screen device, "the plot window." In short, you type a graphics command into the console, such as `plot(1:10, 1:10)`, and you see it right away. The downside is that this on-screen device varies from system to system, and your images will not transfer well. While it's fine for experimentation purposes to use the plot window, once you start caring about specific dimensions and sizes, you'll need another option. There are two routes you can go from here. One is to call a device driver, with a command such as

⁵<https://github.com/nbrgraphs/mro>

bmp(), jpg(), pdf(), png(), jpeg(), or tiff(), and turn it off after the plot is complete:

```
pdf("savedplot.pdf", width = 7, height = 5)
plot(1:10, 1:10)
dev.off()
```

For more on getting your plots to look good with this method, see David Smith's ["10 tips for making your R graphics look their best."](#)⁶

A second route is to use Yihui Xie's [knitr package](#)⁷ to simultaneously create text and plots, and combine them in an output document, as we did to create this publication. We cannot go into detail about **knitr** here, but for anyone who is interested, we have posted the .Rnw document containing all of our code and text [on Github](#).⁸

Since sizing instructions are contained in the chunk options with **knitr** and not in the code itself, and since this is important information to have, we choose to make the figure height and width settings visible at the top of our code sections so you can reproduce our plot sizes, no matter what system you use to create your plots. For example, a line like this:

```
#+ fig.width = 6, fig.height = 3.5
```

indicates that we set the figure width to 6 inches and the figure height to 3.5 inches. There are many ways you can replicate this, depending on your workflow and image type preferences:

```
pdf("mypdf.pdf", width = 6, height = 3.5) #pdf
```

```
png("mypng.png", width = 6, height = 3.5, #png
     units = "in", res = 72)
```

```
```{r myplot, fig.width = 6, fig.height = 3.5} #R Markdown
```

Of course, the plots will not be identical in different image types, and you'll need to experiment to get the results you want. Our figure sizes, though, can be used as a starting point.

---

<sup>6</sup><http://blog.revolutionanalytics.com/2009/01/10-tips-for-making-your-r-graphics-look-their-best.html>

<sup>7</sup><http://yihui.name/knitr/>

<sup>8</sup><https://github.com/nbrgraphs/mro>

## 1.7 What is an effective graph?

Different authors assign different meanings to the term, "effective graph." To some, a graph is effective if it attracts attention. As data analysts, however, we are more concerned with comprehension and clarity than bells and whistles. We call one graph more effective than another if most readers can decode the information more quickly or more accurately from it than from the other. More often than not, fancier and more complex is less effective. Edward Tufte's *The Visual Display of Quantitative Information* and William Cleveland's *The Elements of Graphing Data* remain the classics for understanding what it takes for a graph to be effective and are must-reads for data visualization professionals. For an introduction to the subject, with a focus on putting principles into action, see *Creating More Effective Graphs* by Naomi B. Robbins.

In a nutshell, it is crucial that your chart type and design elements be compatible with the type of data you have, the message you're trying to communicate, and your audience. Many software packages offer "chart choosers" to help the designer select an appropriate chart. It would be convenient if these worked, but unfortunately, it is often the case that they do not. Each data set is unique, and what works for one may not work for another, even one that has the same size and shape and contains the same type of variables. More importantly, the best graph depends on the aspect of the data you wish to emphasize.

Experimenting with different chart types and options will lead to a deeper understanding of your data set, and help you select the graphics type that is best suited to your purposes. Naomi's *Forbes* blog post "[Thinking Outside the Chart Menu](http://www.forbes.com/sites/naomirobbins/2011/11/29/thinking-outside-the-chart-menu/)"<sup>9</sup> describes how thinking about the meaning of the data set led to an innovative use of diverging stacked bar charts (discussed in section 7.1). This chart type was coded by Richard Heiberger and added as the `likert()` function to the **HH** package, a testament to the creativity and flexibility of the R user community.

Finally, we reject the idea we often hear that data visualization as part of exploratory data analysis (EDA) is very different from data visualization for presentation. To be sure, details like font size are not part of EDA. However, the graphical forms that spark insights for you *will* in general be the same ones you'll want to share with others.

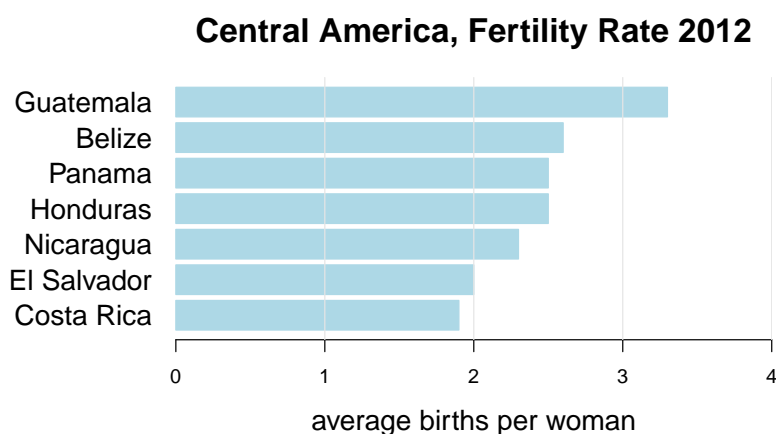
---

<sup>9</sup><http://www.forbes.com/sites/naomirobbins/2011/11/29/thinking-outside-the-chart-menu/>

## 2 Direct Comparisons

### 2.1 Bar charts (base)

Bar charts are a common and effective means to show direct comparisons for small data sets. With base graphics, bar charts are drawn with the `barplot()` function, which takes a numeric vector of lengths as the main argument. For a very simple bar chart, try `barplot(1:5)`. In this example we wish to compare the total fertility rate (measured in average total births per woman) of countries in Central America:



```
##+ fig.width = 7, fig.height = 4

TFR <- c(2.6, 1.9, 2.0, 3.3, 2.5, 2.3, 2.5)
names(TFR) <- c("Belize", "Costa Rica", "El Salvador",
 "Guatemala", "Honduras", "Nicaragua",
 "Panama")

TFR <- sort(TFR)
par(mar=c(5,8,4,2))
barplot (TFR, horiz = TRUE,
 col = "lightblue",
 border = "lightblue",
 main = "Central America, Fertility Rate 2012",
 xlab = "average births per woman",
 xlim = c(0,4),
 cex.lab = 1.4,
```

```
cex.main = 1.7,
cex.names = 1.4,
las = 1)
abline(v = 1:4, col = "grey90")
```

Here TFR is a vector of fertility rates. We use `names()` to assign country names to each vector element since `barplot()` automatically uses these names to label the bars. While `barplot(TFR)` is all that is needed to produce a simple bar chart, we make several adjustments to make the chart easier to interpret. Unless there is an order to the data that shouldn't be altered, the bars should be organized in length order. Therefore, before plotting we sort the data in numerical order.

For accurate perception, bars must start at 0. This may be the default... if it is, don't change it!

We generally prefer horizontal bars (`horiz = TRUE`) since the eye more readily perceives differences in length of bars stacked on top of each other rather than next to each other. In addition, a horizontal bar chart provides more space for the relative long country names, avoiding the problem of crowded, vertical names on the x-axis. We change the orientation of the axis labels to horizontal with (`las=1`).

Changing the plot margins with `par(mar=c(5, 8, 4, 2))` makes space for the country labels. These four numbers refer to the bottom, left, top, and right plot margins, respectively. The defaults are 5.1, 4.1, 4.1, and 2.1, measured in lines of text from the edge of the plot region. (See [Murrell \(2011\)](#), Ch. 3, for a thorough discussion of plotting regions.) Trial and error determines that a left margin of 8 produces enough space.

Next we set the x-axis limits with `xlim=c(0, 4)`. The defaults in base R may produce axes that don't include all the data, which is not ideal. The bar perimeter color is controlled with (`border=`) and the fill color with (`col=`). We use the same color for both since the default black borders are distracting.

The `cex.lab`, `cex.main` and `cex.names` parameters enlarge the text of the bar labels, axis labels, and title, respectively. The setting (1.4 or 1.7 in this case) represents the number of times larger (or smaller) to make the text relative to the default size. In general, we prefer font sizes that are large but not overpowering. The title size (`cex.main`) should be larger than the

axis label (`cex.lab`) and bar label size (`cex.names`), which in turn should be larger than the tick mark label size (`cex.axis`). (Since we wished to keep the default of 1, we did not set `cex.axis` in this example.)

Gridlines help the reader estimate the value of the data but should be minimal and subtle. In this case, only vertical gridlines are necessary. We add gridlines with the `abline()` function: vertical lines are drawn with (`v=`) and horizontal ones with (`h=`). Making use of the colon (`:`) to generate integer sequences in R, `abline(v=1:4)` draws vertical lines at  $x = 1$ ,  $x = 2$ ,  $x = 3$ , and  $x = 4$ . Note that `abline()` can only be called to add lines to an existing plot. We set the color of the lines to a very light grey—"grey90"—so the lines won't interfere with the data, the "main attraction." Colors can be set in many ways. Using the [657 named colors](#)<sup>10</sup> is convenient since it makes the code easier to read. A list of the color names can be obtained by typing `colors()` in the console.

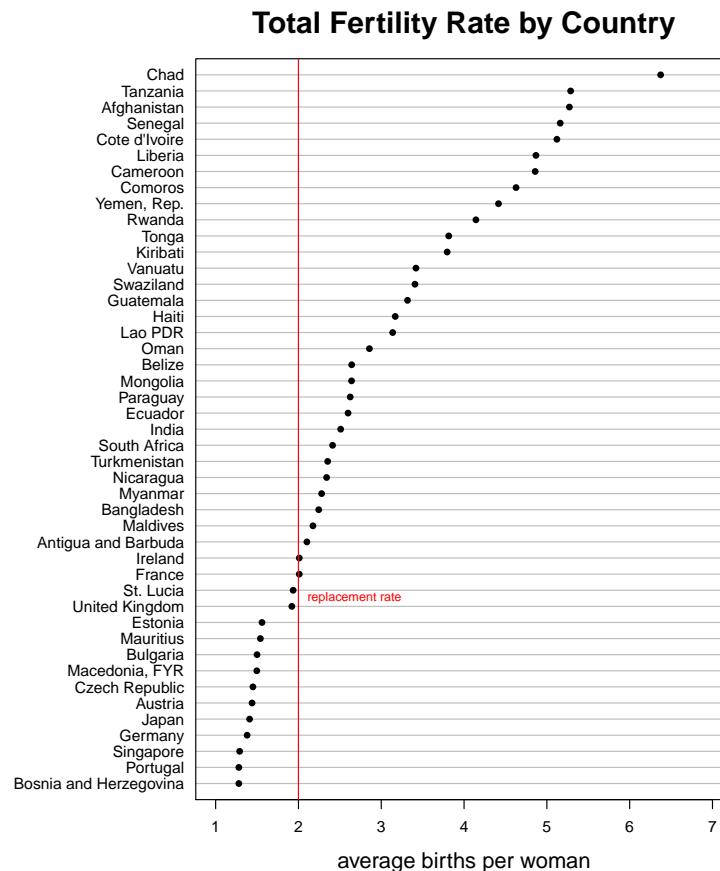
Finally, we wish to make the bars narrower as the thick bars take up space without adding any information. Narrow bars are also more pleasing to the eye. While `barplot()` takes a (`width=`) argument, it only works if (`xlim=`), or (`ylim=`) for a horizontal bar chart, is adjusted, and even then other adjustments need to be made to keep the plot proportional. Therefore, we prefer to adjust the bar width by changing the figure height. The method will change depending on the graphics device in use. With **knitr** we add `fig.width=7` and `fig.height=4` to the chunk options (see [section 1.6](#) for more on sizing graphics).

## 2.2 Dot plots (base)

Bars get cluttered quickly. It's hard to give an exact number but if the graph is looking crowded, consider switching to a Cleveland dot plot. It can accommodate many more data points, and is a better choice for log scales and showing error bars. Unlike bar charts, the axis scale does not need to start at zero, since we are judging position rather than length. Here we show the total fertility rate for a larger sample of countries (see [Appendix A.1](#) for the data and source). The sample is subsetting from the full data set with `index <- seq(from=1, to=179, by=4)`, which creates an index of every fourth value beginning at 1. If the paper size were larger or this list appeared online, there's no reason we couldn't use the full data set in a dot plot.

---

<sup>10</sup><http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf>



Despite the advantages of the dot plot, it is rare to find it as a built-in option in data visualization software packages. R base graphics, however, does have a function, `dotchart()`, for this purpose. One simple adjustment to the defaults for a single series is to set the plotting symbol to a filled rather than an open circle with `(pch=16)`. When plotting multiple series, however, if symbols overlap, the default open circles should be used.

We prefer right-justified labels and solid gridlines to the default dotted ones produced by `dotchart()`. Neither the line type nor the justification is a parameter that can be passed to this plot type. However, it's easy in R to make changes quickly to built-in functions—a feature we really like. In this case, type `dotchart`—without the parens—into the console to see the function code. Without delving deep into the code, you'll see two `abline()` calls that include `(lty="dotted")`. Copy the code into an R script, change the `lty` settings to `"solid"`, change the `(adj=0)` setting to `(adj=adj)` in the `text()`

call that plots the labels, so the desired justification setting can be passed to the dot plot function: (adj=1) for right justification. To save the function for future use, it's best to change the name. Ours is called dotchartsolid(). It's saved in a file called "dotchartsolid.R" and is available [on Github](#).<sup>11</sup>

Annotating plots with relevant information can boost the reader's ability to interpret the data. In this case, a total fertility rate of "2" is a benchmark since it represents the rate needed to replace the current population size through childbirth. Adding a line indicating this benchmark, as we have done in red with abline(), helps us readily identify which countries have total fertility rates above and below the replacement rate.

```
#+ fig.width = 7, fig.height = 8

source("dotchartsolid.R")
data <- read.csv("data/countries2012.csv")
index <- seq(from = 1, to = 179, by = 4)
sample <- data$TFR[index]
names(sample) <- data$COUNTRY[index]
sample <- sample[order(sample)]
par(mar = c(5, 10, 4, 2))
dotchartsolid(sample, cex = .8, pch = 16, xlim = c(1,7),
 main = "Total Fertility Rate by Country",
 xlab = "average births per woman",
 adj = 1, cex.main = 2, cex.lab = 1.5)
abline (v=2, col = "red")
text (2, 12.5, "replacement rate", cex = .7, pos = 4,
 col = "red")
```

---

<sup>11</sup><https://github.com/nbrgraphs/mro>

## 3 Distributions

Often we are interested in how a variable is distributed. Is it symmetric or skewed? Is it multimodal or not? How spread out is it? In these cases we need a method for viewing the full distribution of the data. For a single data set, a histogram is a good tool for the job and an easy chart to create in R. To compare the distributions of several data sets or groups within one set, boxplots are a great choice.

### 3.1 Histograms (base)

Each rectangle in a histogram is called a bin; it "holds" the number or percent of data points indicated on the y-axis. Be careful: although they look somewhat similar, a histogram is *not* a bar chart. For more details on the difference, see Naomi's [blog post on the topic](http://www.forbes.com/sites/naomirobbins/2012/01/04/a-histogram-is-not-a-bar-chart/).<sup>12</sup> The number of bins can greatly affect the appearance of the histogram. As [Aaron Schumacher points out](http://planspace.org/20141225-how_does_r_calculate_histogram_break_points/)<sup>13</sup>, the process by which the bin size in R is determined is complex and not always optimal. There are several ways to use the (breaks=) parameter to set your own bin breakpoints. It will take an integer for the number of bins, but it will be adjusted according to a *pretty* value, i.e., one that is 1, 2, or 5 times a power of 10. Tightest control of the bin limits is obtained by setting (breaks=) to a vector of positions such as: `seq(from=0, to=1000, by=50)`. We recommend trying different bin sizes. If an inconsistency appears, investigate further to determine its cause.

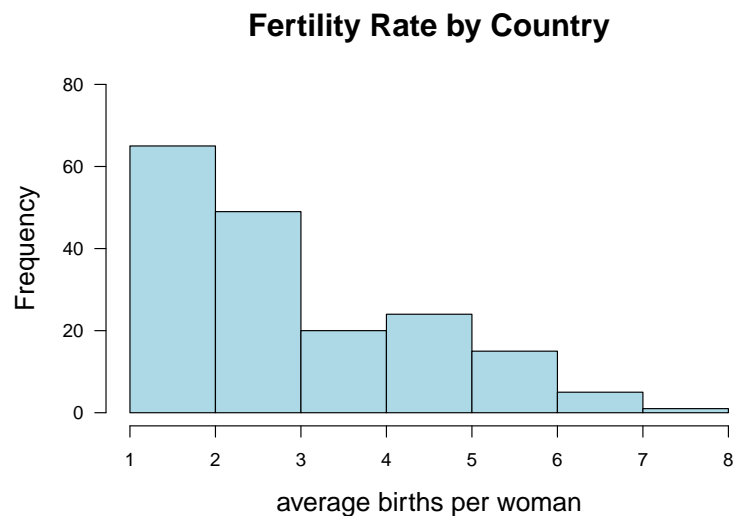
In this example we use the full country (n = 179) list of total fertility rate data discussed in the previous section:

---

<sup>12</sup><http://www.forbes.com/sites/naomirobbins/2012/01/04/a-histogram-is-not-a-bar-chart/>

<sup>13</sup>[http://planspace.org/20141225-how\\_does\\_r\\_calculate\\_histogram\\_break\\_points/](http://planspace.org/20141225-how_does_r_calculate_histogram_break_points/)

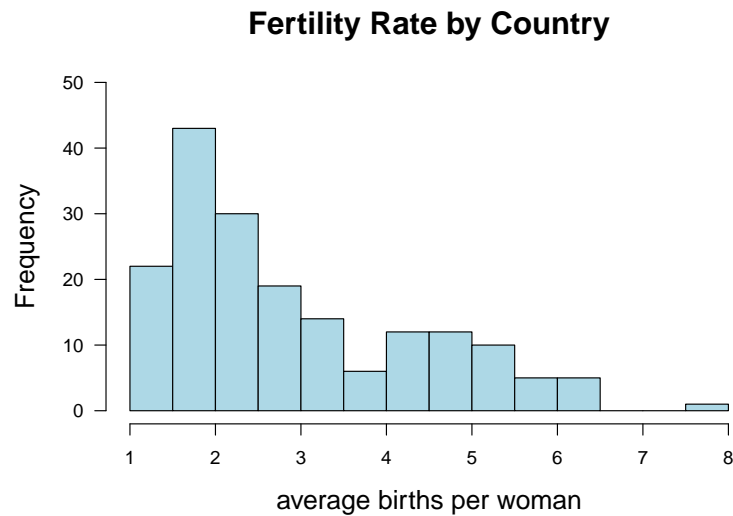




```
##+ fig.width = 7, fig.height = 5

x <- read.csv("data/countries2012.csv")
hist(x$TFR, breaks = seq(from = 1, to = 8), col = "lightblue",
 main = "Fertility Rate by Country",
 xlab = "average births per woman",
 xlim = c(1,8), ylim = c(0,80),
 cex.main = 1.7, cex.lab = 1.4, las = 1)
```

We can clearly see from the histogram that the most common total fertility rate is between 1 and 2, and the distribution skews right. Dividing the bins in half with `breaks=seq(from=0, to=7, by=.5)` provides more detail:



```
#+ fig.width = 7, fig.height = 5

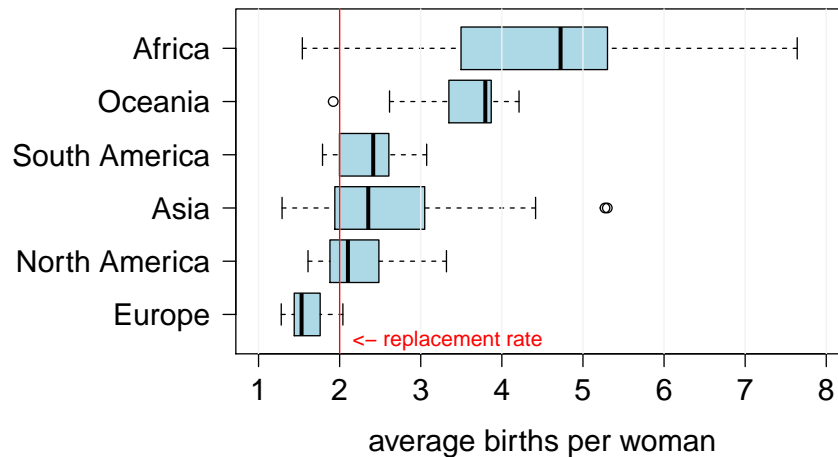
x <- read.csv("data/countries2012.csv")
hist(x$TFR, breaks = seq(from = 1, to = 8, by = .5),
 col = "lightblue",
 main = "Fertility Rate by Country",
 xlab = "average births per woman",
 xlim = c(1,8), ylim = c(0,50),
 cex.main = 1.7, cex.lab = 1.4, las = 1)
```

Again, we recommend caution when changing the bin size since not all choices represent the data well.

### 3.2 Box plots (base)

Box plots are far superior to histograms for displaying more than one distribution. Here we compare the distributions of total fertility rate by continent. We can clearly see that Africa not only has the highest median total fertility rate, but also the greatest range of rates. The opposite is the case for Europe.

## Fertility Rate Distributions by Continent



```
#+ fig.width = 7, fig.height = 4.5
```

```
Box plots
par(mar = c(5,9,4,2))
data <- read.csv("data/countries2012.csv")
data$CONTINENT <- reorder(data$CONTINENT, data$TFR, median)
boxplot(TFR ~ CONTINENT, data, horizontal = TRUE,
 ylim = c(1,8),
 col = "lightblue",
 main = "Fertility Rate Distributions by Continent",
 xlab = "average births per woman", cex.main = 1.7,
 cex.lab = 1.4, cex.axis = 1.4, las = 1)
abline(v = 1:8, col = "grey95")
abline(v = 2, col = "red")
text(x = 2, y = .5, "<- replacement rate", col = "red",
 pos = 4)
```

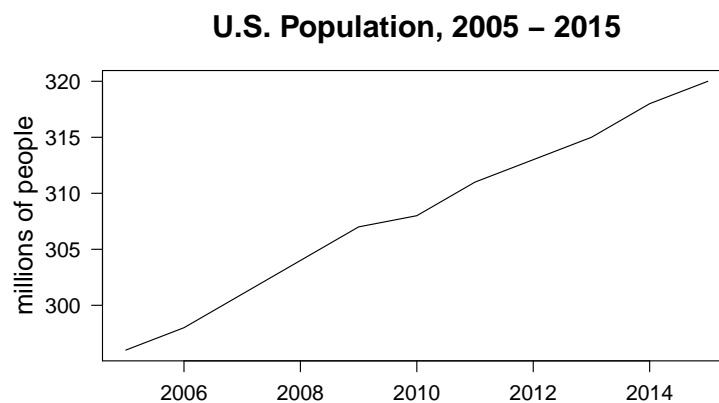
Passing the function `TFR~CONTINENT` as the first argument to `boxplot()` indicates how the full distribution of total fertility rates should be grouped. As we've seen previously, graphs are easier to decipher if the graphical elements are ordered by size. Since `boxplot()` plots the data in order of the factors, we sort the continent factor levels by median group total fertility rate with the `reorder()` function.

## 4 Trends Over Time

### 4.1 Line charts (base)

The use of line charts to depict trends over time is a tried-and-true method: there is evidence from the 10th or 11th century of [a line chart showing planetary movements plotted against time](#).<sup>14</sup> Unlike bar charts, line charts do not need to have a zero baseline. The most common mistake we see with line charts is using evenly spaced tick marks to represent different amounts of time.

The simplest way to create a line chart in R is to use (type="l") with the versatile plot() function.<sup>15</sup> Here we show the population of the U.S. from 2005-2015:



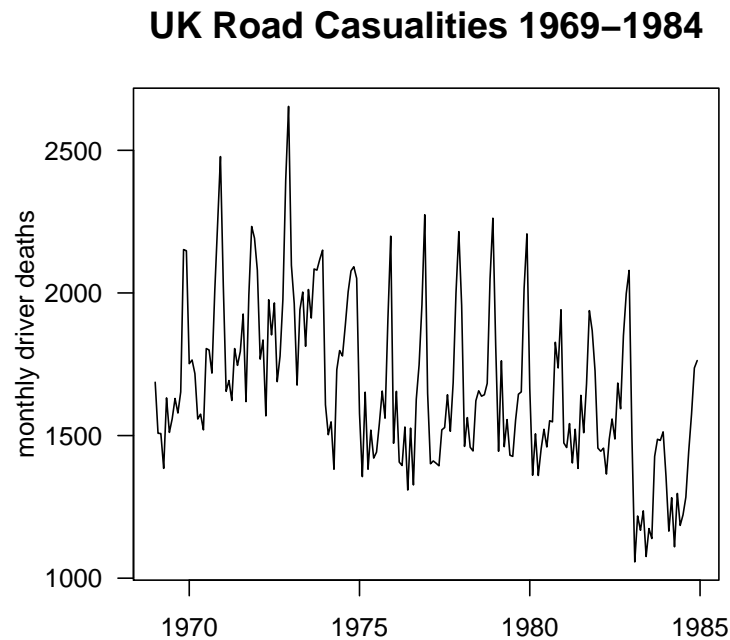
```
#+ fig.width = 7, fig.height = 4.5
```

```
Line chart
df <- data.frame (year = seq(2005,2015),
 pop = c(296, 298, 301, 304, 307, 308,
 311, 313, 315, 318, 320))
plot(df$year, df$pop, type = "l",
 main = "U.S. Population, 2005 - 2015", xlab = "",
 ylab = "millions of people", las = 1,
 cex.axis = 1.2, , cex.lab = 1.4, cex.main = 1.7)
```

<sup>14</sup><http://visage.co/data-visualization-101-line-charts/>

<sup>15</sup>To see how versatile plot() really is, try each of the following: plot(ChickWeight) plot(AirPassengers) plot(BOD) plot(HairEyeColor)

Time-series objects, such as the built-in data set `UKDriverDeaths`, are simple to plot in R. This plot requires only one line of code.



```
#+ fig.width = 5, fig.height = 5
```

```
plot(UKDriverDeaths,
 main = "UK Road Casualties 1969-1984",
 ylab = "monthly driver deaths", xlab = "", las = 1,
 cex.main = 1.4, cex.lab = 1.2, cex.axis = 1.2)
```

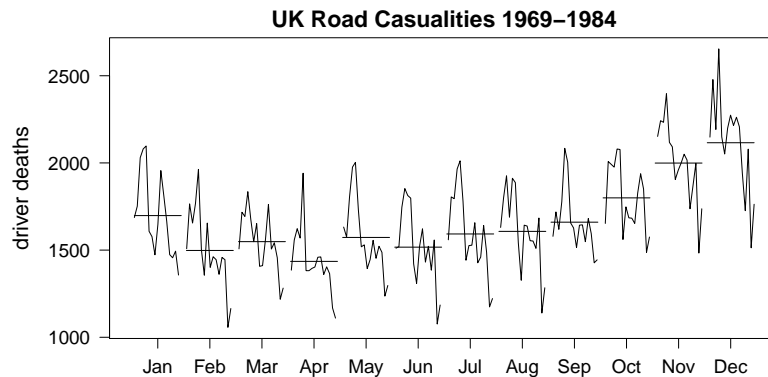
To convert a data set to a time-series object, we use the `ts()` function, for example:

```
data <- ts(data, start = c(2010,7), frequency = 12)
```

Time-series are particularly convenient for monthly (`frequency=12`) or quarterly (`frequency=4`) data.

## 4.2 Month plots (base)

A nice R feature for time-series is the `monthplot()` function, which groups the time series data by month and plots the data as 12 separate line charts on the same set of axes:



```
#+ fig.width = 8.5, fig.height = 4.5
```

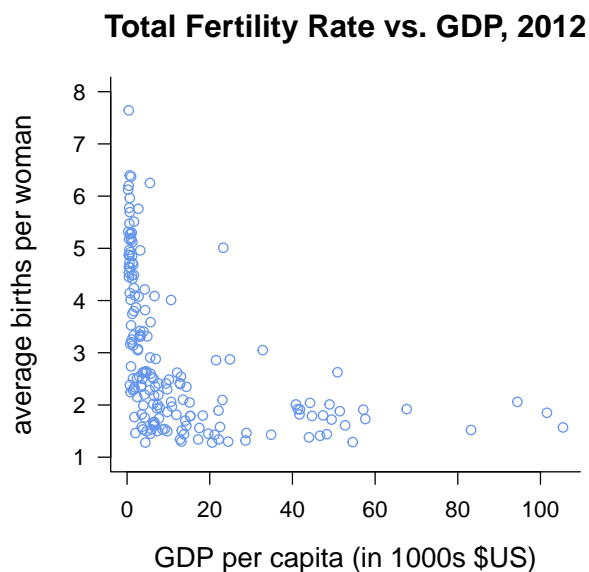
```
Month plot
par(mar=c(5,6,2,2))
monthplot(UKDriverDeaths,
 main="UK Road Casualties 1969-1984",
 labels = month.abb, las = 1, ylab="",
 cex.axis = 1.2, cex.main = 1.4)
mtext("driver deaths", side = 2, line = 4, cex = 1.3)
```

In the month plot, it is easy to see both the trend of decreasing deaths over time along with a pattern of increased deaths in the last several months of the year. We set `(labels=month.abb)` since the built-in month abbreviations are clearer than the `monthplot()` default of single letters to represent the months. Month plots are also called cycle plots.

## 5 Relationships

### 5.1 Scatterplots (base)

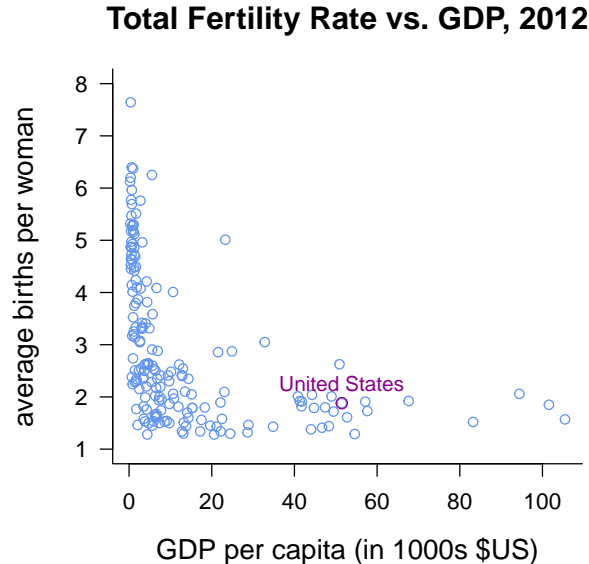
A scatterplot is a great tool for showing the relationship between two sets of data. Suppose, for example, we are interested in investigating whether there's a correlation between gross domestic product (GDP) and total fertility rate (TFR). In base graphics either `plot(x,y)` or `plot(y~x)` will plot the data on a coordinate system. Since some of the data points overlap, we use open circles (the default) for clarity. We divide GDP by 1000 to avoid either scientific notation or long strings of 0's on the x-axis, both of which may confuse readers. Of course, we must add "1000s" to the x-axis label for an accurate unit label.



```
#+ fig.width = 5, fig.height = 5

data <- read.csv("data/countries2012.csv")
plot(data$GDP/1000, data$TFR, xpd = TRUE, bty = "n",
 main = "Total Fertility Rate vs. GDP, 2012",
 xlab = "GDP per capita (in 1000s $US)",
 ylab = "average births per woman",
 ylim = c(1, 8), cex.main = 1.5,
 cex.lab = 1.3, cex.axis = 1.1,
 col = "cornflowerblue", las = 1)
```

We don't recommend overlabeling data points since the labels can interfere with the visibility of the data. If you feel a need to label the data points, it may be a sign that the plot type or parameters you've chosen aren't the best for showing off your data. That said, at times it may be useful to label one or more data points. To illustrate the method, in this example, we label the U.S. in magenta. We also replot the data point itself in magenta to avoid confusing it with nearby data points.



With the `text()` function it is easy to label the point with the same coordinates we used to plot the data point. We use `(pos=3)` to place the label above the point, so the two don't overlap. Alternatively, the position of the

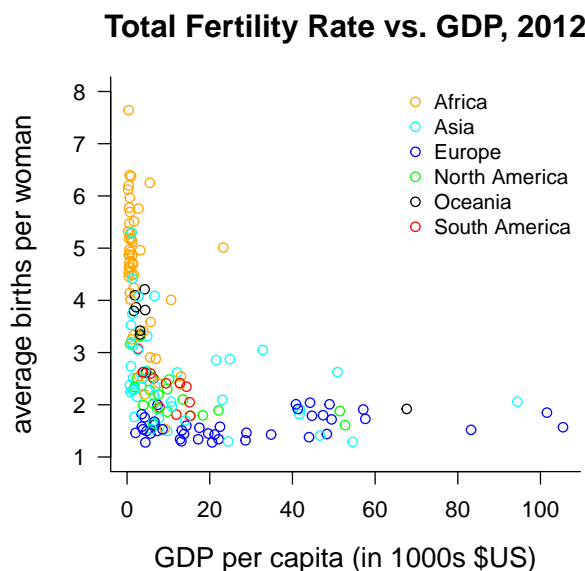


labels can be adjusted by making minor changes to the coordinates. Setting the legend border with (bty="l") removes the top and right plot box lines, providing a cleaner look.

```
#+ fig.width = 5, fig.height = 5

Scatterplot with one label
data <- read.csv("data/countries2012.csv")
plot(data$GDP/1000, data$TFR, xpd = TRUE, bty = "l",
 main = "Total Fertility Rate vs. GDP, 2012",
 xlab = "GDP per capita (in 1000s $US)",
 ylab = "average births per woman",
 ylim = c(1,8), cex.main = 1.5,
 cex.lab = 1.3, cex.axis = 1.1,
 col = "cornflowerblue", las = 1)
usdata <- data[data$COUNTRY=="United States",]
text(usdata$GDP/1000, usdata$TFR,
 labels = usdata$COUNTRY, pos = 3, col = "magenta4")
points(usdata$GDP/1000, usdata$TFR, cex = 1.2,
 col = "magenta4")
```

In the final scatter plot, we color the data points by continent:



```
#+ fig.width = 5, fig.height = 5

Scatterplot with continents labeled by color
data <- read.csv("data/countries2012.csv")
colors6 <- c("orange","cyan","blue","green",
 "black", "red")
plot(data$GDP/1000,data$TFR, bty = "l",
 col = colors6[data$CONTINENT],
 xlab = "GDP per capita (in 1000s $US)",
 ylab = "average births per woman",
 main = "Total Fertility Rate vs. GDP, 2012",
 las = 1, ylim = c(1,8), cex.main = 1.5, cex.lab = 1.3,
 cex.axis = 1.1)
legend("topright", pch = 21, legend=levels(data$CONTINENT),
 col = colors6, bty = "n")
```

Note that by setting the color to `colors6[data$CONTINENT]`, the vector of colors, `colors6`, is paired with the levels of the group variable `data$CONTINENT`. Therefore it may be helpful to view the levels of the factor when choosing colors :

```
levels(data$CONTINENT)
[1] "Africa" "Asia" "Europe"
[4] "North America" "Oceania" "South America"
```

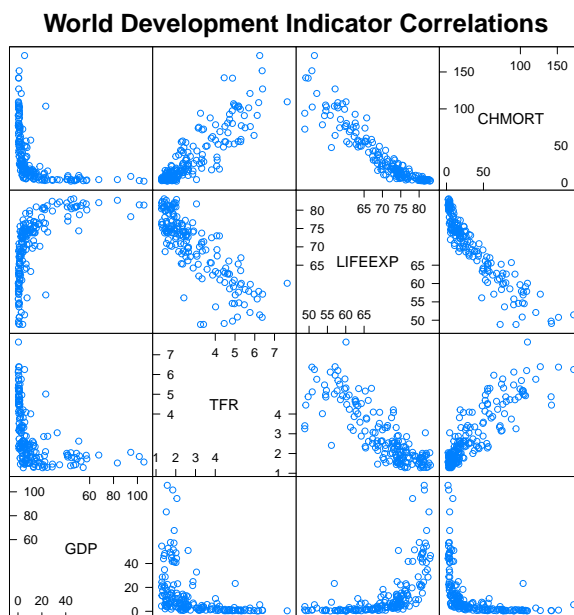
Colors should be rearranged so that nearby clusters such as Asia and Europe show as much contrast as possible. It is essential, though, that colors remain consistent within a report or presentation to avoid confusion.

A legend is added with the `legend()` function. The parameter `legend` is set to the levels of the group variable and the color to the corresponding vector of colors, `colors6`. The number of colors must be equal to the the number of factors for the legend to be drawn properly.

## 5.2 Scatterplot matrices (lattice)

A scatterplot matrix is useful for viewing all two-way relationships in a multidimensional data set. For this example, we use the `splom()` function in the **lattice** package since we like its default settings, in particular the lack of

margin space between panels. In the following plot, we can simultaneously look at the correlation of any of the six pairs of variables among the four in our data set: total fertility rate (TFR), life expectancy (LIFEEXP), gross domestic product (GDP), and child mortality (CHMORT). Note that the six plots above the diagonal in the top left half of the graph are the same as the six in the bottom right half, with the axes flipped. For this reason, some recommend placing additional plots such as single variable histograms on one half the graph to avoid repetition. However, since relationships may be clearer with a particular variable on a particular axis, we prefer to fill the entire square with scatterplots, as shown.<sup>16</sup>



The scatterplot matrix clearly shows the positive correlation between child mortality rate and total fertility rate, and the negative correlations between life expectancy and total fertility rate, and between life expectancy and child mortality. We also observe that all of the plots involving gross domestic product are L-shaped, indicating that the relationships are not linear.

The plot is drawn in **lattice** with a simple call to `splom()`. The data is passed to the function as columns of a data frame.

<sup>16</sup>For a more detailed explanation, see [Jacoby \(1998\)](#).

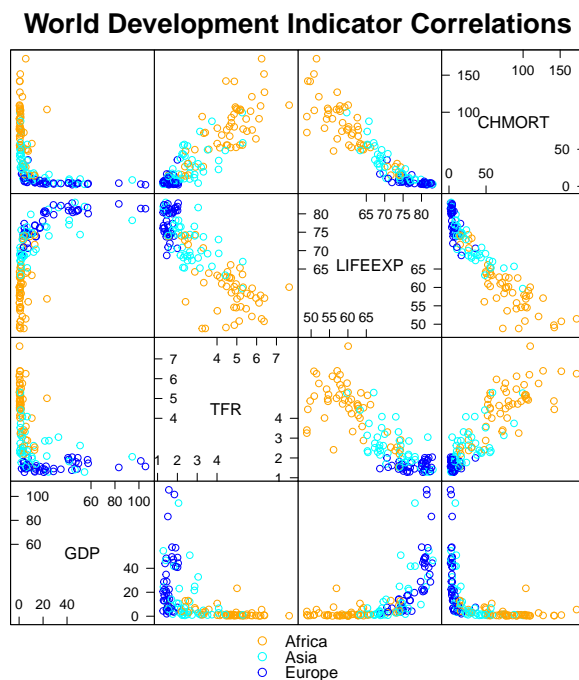
```

#+ fig.width = 7, fig.height = 7

library(lattice)
data <- read.csv ("data/countries2012.csv")
data$GDP <- data$GDP/1000
splom(data[,c("GDP", "TFR", "LIFEEXP", "CHMORT")],
 scales = list(y = list(tick.number = 0)),
 main = list(label = paste("World Development",
 "Indicator Correlations"),
 cex = 1.7), xlab=NULL)

```

If we wish to observe continent clusters in the scatterplot matrix, as with the simple scatterplot, we set `col=colors3[data$CONTINENT]`. We chose to limit the number of continents to three since any more would be hard to distinguish in small plots.



```

#+ fig.width = 7, fig.height = 7

Scatterplot matrix with continents in color
library(lattice)
colors3 <- c("orange","cyan","blue")
data <- read.csv ("data/countries2012.csv")
data$GDP <- data$GDP/1000
data <- droplevels(data[data$CONTINENT %in%
 c("Europe","Asia","Africa"),])
splom(data[,c("GDP","TFR","LIFEEXP","CHMORT")],
 col = colors3[data$CONTINENT],
 key = list(space="bottom",
 points=list(pch = 21, col=colors3),
 text=list(levels(data$CONTINENT))),
 main = list(label = paste("World Development",
 "Indicator Correlations"),
 cex = 1.7), xlab=NULL)

```

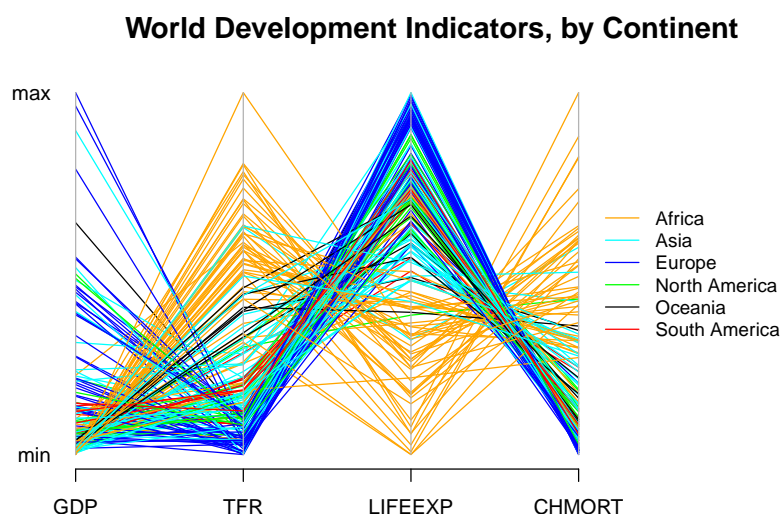
In addition to the correlations, the color scatterplot matrix shows a clear continent effect. In the top left box, for example, we see that the vertical piece of the plot represents countries in Africa and the horizontal piece countries in Europe. Another method for identifying group effects is the parallel coordinate plot, which we'll describe in the next section.

In **lattice**, (key=) creates the legend. It is a list of lists containing information on the location (space=), symbol color lines=list(col=colors3), and text text=list(levels(data\$CONTINENT)) for the legend. To ensure that the number of colors is equal to the number of factor levels, we used droplevels() when subsetting the data to remove unused factor levels. An alternative would be to subset the first three elements of the factor levels vector when drawing the legend.

### 5.3 Parallel coordinate plots (MASS)

Graphs based on the coordinate system, such as the scatterplot, represent different variables on perpendicular axes. The problem is that we are limited to at most three dimensions, and even a third dimension can be difficult to decipher in two dimensional space. To cope with the challenge of displaying multidimensional data, the parallel coordinate plot represents the

variables on parallel axes, greatly increasing the number of variables that can be conveniently and clearly displayed. Here we plot four world development indicators on a parallel coordinate plot, grouped by continent:



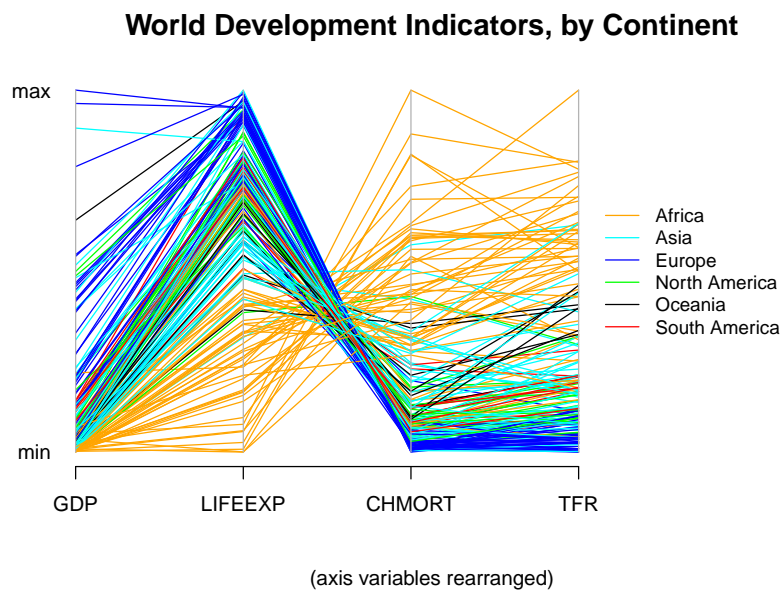
We use the `parcoord()` in the **MASS** package to create the parallel coordinate plot. The function scales each variable on a scale from 0 to 1. We use the `axis()` function to label the top and bottom of the y-axis "max" and "min" respectively; (`tick=FALSE`) turns off the unnecessary axis line and tick marks.

```
##+ fig.width = 7, fig.height = 5

library(MASS)
par(bg='white')
data <- read.csv ("data/countries2012.csv")
colors6 <- c("orange", "cyan", "blue", "green",
 "black", "red")
parcoord(data[,c("GDP", "TFR", "LIFEEXP", "CHMORT")],
 col = colors6[data$CONTINENT], xlim=c(1,5.25),
 main = "World Development Indicators, by Continent",
 cex.main = 1.4, xaxs = "i")
axis(2, at = c(0,1), labels = c("min", "max"), las = 1,
 tick = FALSE)
```

```
legend("right", lty = 1, legend = levels(data$CONTINENT),
 col = colors6, cex = .9, bty = "n")
```

The order of the axes can greatly influence the look of the graph, as the chart below demonstrates.



The code for the second plot is identical to the first, with the exception of the beginning of the `parcoord()` call:

```
parcoord(data[,c("GDP", "LIFEEXP", "CHMORT", "TFR")],
```

## 6 Percents... or Parts of a Whole

### *Unidimensional data*

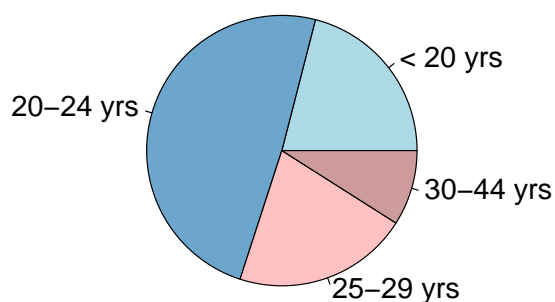
#### 6.1 Pie charts (base)

If you design charts, you're probably aware that the pie chart is quite controversial. The documentation for the `pie()` function in base graphics doesn't pull any punches: "Pie charts are a very bad way of displaying information.... A bar chart or dot chart is a preferable way of displaying this type of data." We agree. Experiments by Cleveland and McGill show that we judge position and length more accurately than angles or areas.

One positive feature of pie charts, however, is that they clearly show that the sum of the wedges is 100%. To retain this feature in bar charts we created a "bar percent chart" with percent labels, which we'll discuss in the next section.

Suffice it to say here that if you do use a pie chart, be sure that the data you're displaying represent parts of a whole. Otherwise the chart isn't just difficult to read, it's nonsensical. In this example, we use a pie chart to show the percentages of fathers without high school degrees who had their first child within particular age brackets. (See Appendix A.2 for a data table.)

**Fathers without High School Degrees:  
Age at First Child**



Documentation warnings notwithstanding, a pie chart is easily drawn in base graphics by sending a vector of values to `pie()`. The values are converted to percents, if necessary, before the plot is drawn. Note that for long



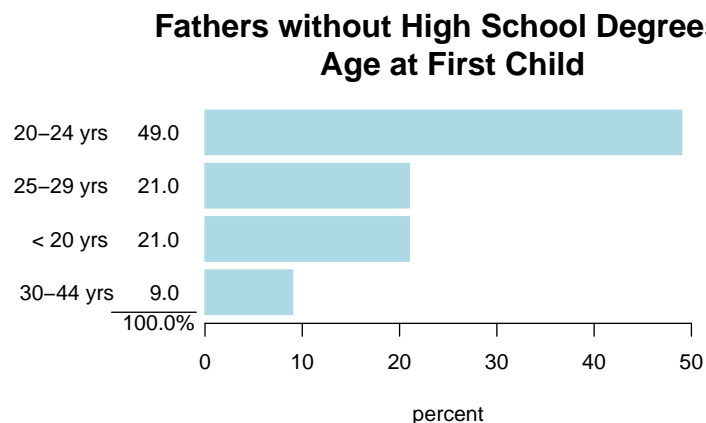
titles, creating a line break with `\n` is a convenient way to divide the title without adding a second title line.

```
#+ fig.width = 7, fig.height = 5

data <- read.table("data/fathers.txt")
colors4 <- c("lightblue", "skyblue3", "rosybrown1",
 "rosybrown3")
pie(data[, "NOHSDEG"], labels = data$AGE, col = colors4,
 cex = 1.5)
mtext(paste("Fathers without High School Degrees:\n",
 "Age at First Child"), side = 3,
 cex = 1.7, font = 2)
```

## 6.2 Bar percent charts (base)

For very small data sets, we generally prefer bar charts. To make it clear that our data points are parts of a whole, we label the bars with percents to show that they total to 100%. We created a function `barpercent()` to do this:



```
#+ fig.width = 7, fig.height = 3.5

source("barpercent.R")
data <- read.table("data/fathers.txt")
```

```
barpercent(data$NOHSDEG,data$AGE)
mtext(paste("Fathers without High School Degrees: \n",
 "Age at First Child"), side = 3, line = 1,
 font = 2, cex = 1.5)
```

We include the code here as it shows off what we love about R: you are never limited by a set number of chart types. If the chart you want isn't available, you can create it, or more often than not, find a package that has what you're looking for. The file "barpercent.R" is available [on Github](#).<sup>17</sup>

```
barpercent <- function (x, names = NULL) {
 if (is.null(names)) names <- names(x)
 par (las = 1, mar = c(5,10,4,6), xpd = NA, xaxs = "i", yaxs = "i")
 data <- x[order(x)]*100/sum(x)
 labels <- paste(names[order(x)], " ",
 formatC(data,digits=1,format="f",width=4))
 xmax <- round(max(data),digits = -1)
 ymax <- (length(x)*1.2)+.1
 plot(NULL, xlim = c(0,xmax), ylim=c(0,ymax), axes = FALSE,
 xlab = "percent", ylab="")
 barplot(data, horiz = TRUE, names.arg = labels,
 col = "lightblue", border = "lightblue",
 axes = FALSE, add = TRUE)
 axis (1, pos = 0, xlim = c(0,xmax), at = c(0,1:(xmax/10)*10))
 hat <- -xmax/16.66667
 text(hat, .2, "100.0", adj = c(1,1))
 text(hat, .2, "%", adj = c(0,1))
 lines(c(.3,3.2)*hat,c(.25,.25))
}
```

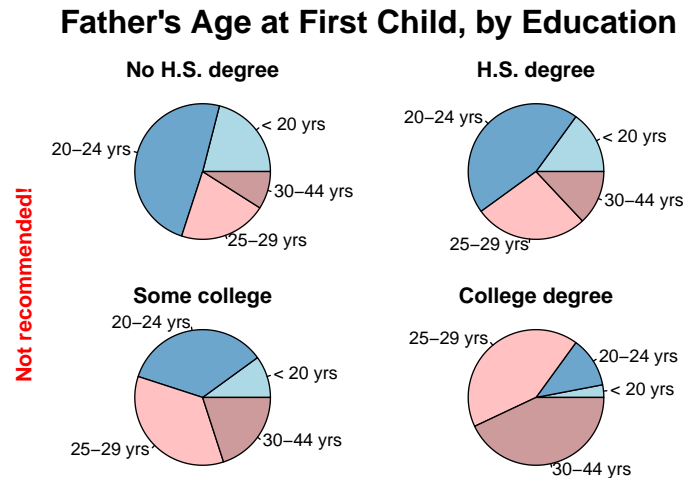
## Multidimensional data

### 6.3 Multiple pie charts (base)

What if we're interested not only in the age at which fathers who didn't complete high school had their first child, but the age at which fathers en-

<sup>17</sup><https://github.com/nbrgraphs/mro>

tered fatherhood for a range of educational levels? A common approach is to draw a series of pie charts:



```
#+ fig.width = 5.5, fig.height = 4

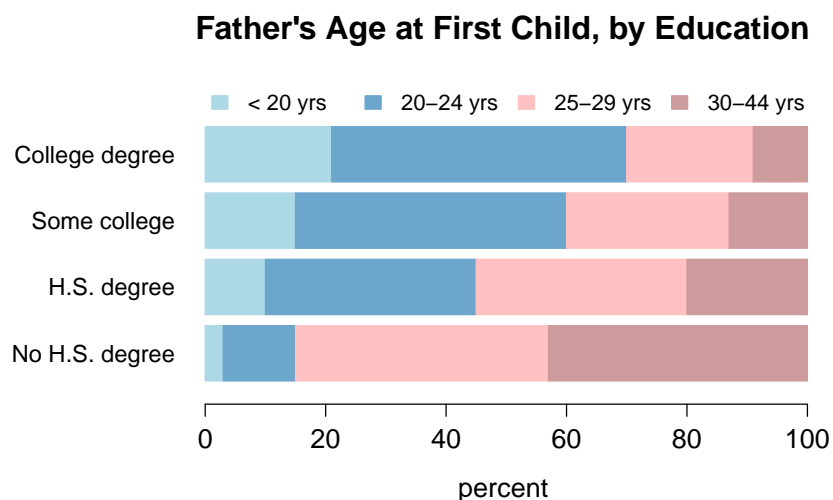
data <- read.table("data/fathers.txt")
colors4 <- c("lightblue", "skyblue3", "rosybrown1",
 "rosybrown3")
par(mfrow = c(2,2), mar=c(1,0,1,0), oma = c(0,2,3,0))
pietitles <- c("No H.S. degree", "H.S. degree",
 "Some college", "College degree")
for (i in 2:5) {
 pie(data[,i], labels = data$AGE,
 main = pietitles[i-1], col = colors4)
}
mtext("Father's Age at First Child, by Education",
 side=3, line=1, outer=TRUE, cex=1.5, font=2)
mtext("Not recommended!", side = 2, font = 2,
 outer = TRUE, col = "red")
```

We're able to put four graphs in one plot using `par(mfrow=c(2,2))`. So the labels aren't cut off, we eliminate the left and right margins, and for overall spacing shrink the top and bottom margins of each graph to 1 with `mar=c(1,0,1,0)`. Extra space is needed to fit the title and y-axis label, so we change the left outer margin to 2 and the top outer margin to 3: `oma=c(0,2,3,0)`.

(The default is 0 on all sides.) Note that with `mtext()` we need `(outer=TRUE)` in order for the text to appear in the outer margin. Getting the spacing right with pie charts is a challenge—another incentive not to use them.

In terms of the content, we find this graph particularly confusing. It is difficult to read one pie chart let alone compare one to another, particularly since certain categories, such as "25-29 yrs", appear in different places in different pies due to the variety of wedge or slice sizes. What are our alternatives?

## 6.4 Divided bar charts (base)



A divided bar chart is a common option for comparing parts of a whole in different groups. The coding is the same as for a stacked bar chart; what distinguishes it from a stacked bar chart is that the columns all sum to 100%, so the bar chart is even on both sides. While we're not big fans of stacked bar charts, the divided bar chart does a fair job at showing percents. We clearly see here that the percentage of fathers who had their first child before they turned 20 decreases significantly as education increases (light blue bars on the left). Likewise the percentage who had their first child in the 30-44 yrs age range increases with education (dark pink bars on the right). The dark blue and light pink bars representing middle age ranges are more difficult to compare since they don't have a common starting point. Still with this data trends are discernible.

```

#+ fig.width = 7, fig.height = 4.5

par(mar=c(5, 8, 4, 2))
fathers <- read.table("data/fathers.txt")
columns <- c("COLLDEG", "SOMECOLL", "HSDEG", "NOHSDEG")
data <- as.matrix(subset(fathers, select = columns))
colors4 <- c("lightblue", "skyblue3", "rosybrown1",
 "rosybrown3")
edugroups <- c("No H.S. degree", "H.S. degree",
 "Some college", "College degree")
barplot (data, names.arg = edugroups, horiz = TRUE,
 col = colors4, border = colors4, xlab = "percent",
 ylim = c(0,5.5), cex.axis = 1.3, cex.names = 1.2,
 cex.lab = 1.3, las = 1)
legend ("top", legend=fathers$AGE, fill = colors4,
 border = colors4, bty = "n", cex = 1.1,
 horiz = TRUE)
title ("Father's Age at First Child, by Education",
 cex.main = 1.6)

```

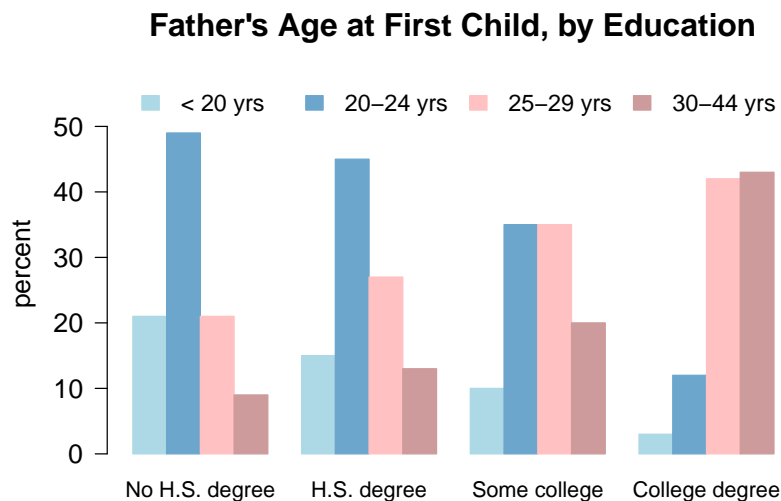
The divided bar chart is drawn in base graphics by passing a matrix of values to the `barplot()` function. The stacks represent the columns of the matrix and the colors represent the rows. When converting to matrix form with `as.matrix()`, we begin with the factor level we want to appear at the bottom of the chart, in this case `COLLDEG`, since horizontal bar charts are drawn in base graphics from the bottom up. Setting the border color (`border=`) to the same vector of colors as the fill color (`col=`) provides a cleaner look than does the default black border color. (You may have noticed that we kept the black borders in the histograms since the bars are adjacent.)

We extend the y-axis a bit (`ylim=`) so the legend doesn't overlap with the bars. If you wish to extend the x or y coordinate ranges with `xlim / ylim` but don't know what the current ranges are—it's not always obvious—you can get an approximation with `par("usr")`, which returns the x and y ranges, respectively.

The legend should be as easy to read as possible, so we placed it on the top of the graph with the color boxes in the same order as the bars in the chart. If the bars were vertically stacked, we would place the legend on the right, once again with the colors ordered as they are in the graph. If this doesn't happen automatically, resort the order of the (`legend=`) vector. The

default box drawn around the legend is distracting, so we remove it with (bty="n").

## 6.5 Grouped bar charts (base)



The advantage of the grouped bar chart is that values can easily be compared both within and across educational levels since all bars begin in the same place—the x-axis. The downside is that bars of the same color are not adjacent and therefore more concentration is required to observe trends.

The code for the grouped bar chart is nearly identical to that of the divided bar chart. We add (beside=TRUE) so bars are placed alongside each other rather than stacked, remove (horiz=TRUE) to draw vertical bars, and adjust labels and coordinate limits as needed.

```
##+ fig.width = 7, fig.height = 5

fathers <- read.table("data/fathers.txt")
columns <- c("NOHSDEG", "HSDEG", "SOMECOLL", "COLLDEG")
data <- as.matrix(subset(fathers, select = columns))
colors4 <- c("lightblue", "skyblue3", "rosybrown1",
 "rosybrown3")
edugroups <- c("No H.S. degree", "H.S. degree",
```

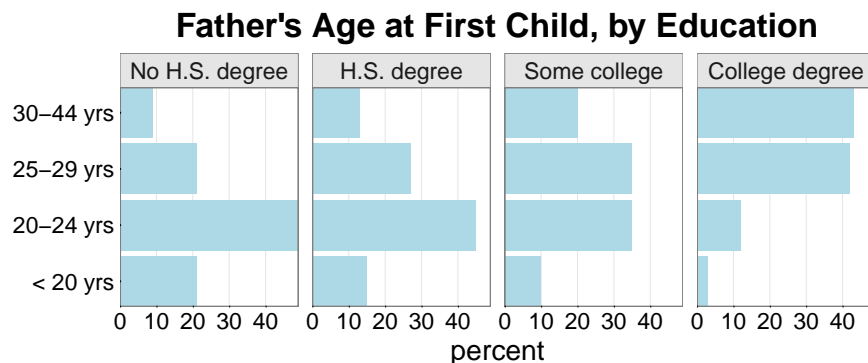
```

 "Some college", "College degree")
barplot(data, names.arg = edugroups, beside=TRUE,
 col = colors4, border = colors4, ylab = "percent",
 ylim = c(0,58), cex.axis = 1.3, cex.names = 1.1,
 cex.lab = 1.3, las = 1)
legend("top", fill = colors4, border = colors4, bty = "n",
 legend=fathers$AGE, cex = 1.2, horiz = TRUE)
title ("Father's Age at First Child, by Education",
 cex.main = 1.6)

```

## 6.6 Faceted bar charts (ggplot2)

Facets offer a powerful way to represent multidimensional data. To avoid the cluttering and confusion that inevitably accompanies attempts to display an entire multidimensional data set in one graph, the data is organized into multiple small plots or panels. Each contains subsets of the data conditioned on levels of one or more variables.<sup>18</sup> We switch to **ggplot2** here since R base graphics doesn't offer a system for faceting. We present a very simple case of faceting: a plot of the fatherhood data in four panels, conditioned on education level:



We use a single color since the layout of the panels allows us to compare

<sup>18</sup>The concept was developed and introduced to S by William Cleveland and colleagues as trellis displays, and implemented in R in the **lattice** package. In *The Grammar of Graphics*, on which the **ggplot2** package is based, Leland Wilkinson defines facets more generally as "frames of frames."

bars horizontally (across education levels) or vertically (within an education level) without having to focus on a particular color. While we like the overall structure, one downside is the condensed nature of the bars resulting from plotting four panels of horizontal bar charts in a single row.

```
#+ fig.width = 7, fig.height = 3

library(ggplot2)
library(tidyr)
fathers <- read.table("data/fathers.txt")
data <- gather(fathers, key = EDUCATION, value = PERCENT,
 -AGE)
levels(data$EDUCATION) <- c("No H.S. degree", "H.S. degree",
 "Some college", "College degree")
g <- ggplot(data, aes(x = AGE, y = PERCENT))
g + geom_bar(stat = "identity", fill = "lightblue") +
 coord_flip(expand = FALSE) +
 facet_grid(.~EDUCATION) + theme_bw(16) +
 theme(axis.line = element_blank(),
 axis.ticks.length = unit(0, "cm"),
 panel.grid.major.y = element_blank(),
 panel.grid.minor = element_blank(),
 strip.background = element_rect(fill="grey90"),
 strip.text.x =
 element_text(margin = margin(t = 5, b = 5)),
 plot.title = element_text(face = "bold")) +
 ggtitle("Father's Age at First Child, by Education") +
 xlab(NULL) + ylab("percent")
```

It is beyond the scope of this introduction to explain the **ggplot2** system from the ground up. Winston Chang's *R Graphics Cookbook* and package author Hadley Wickham's *ggplot2: Elegant Graphics for Data Analysis* are excellent resources for learning the package systematically. As indicated earlier, the ability to convert data into long form is key for **ggplot2**. Two other packages, also by Hadley Wickham, **tidyr** and **dplyr**, provide highly intuitive functions for getting the data in proper form before plotting. RStudio provides [excellent cheatsheets](https://www.rstudio.com/resources/cheatsheets/) for all three.<sup>19</sup>

Regarding our code, **ggplot2** users will note that the fill color is added to

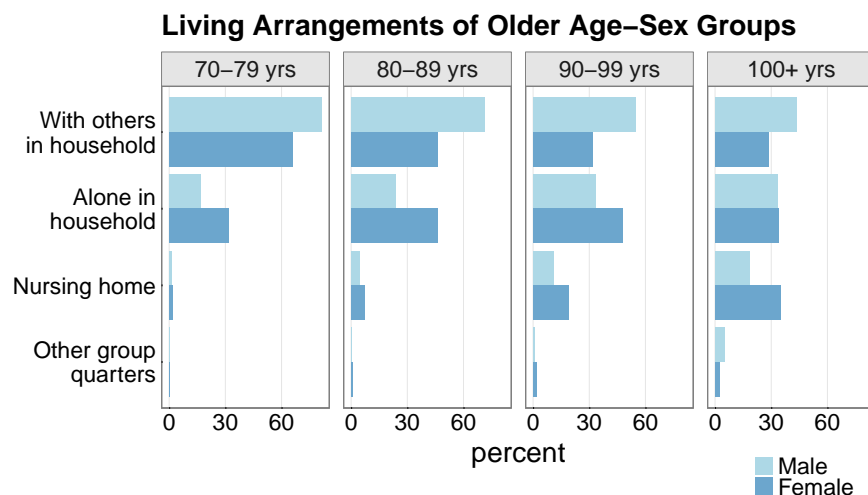
---

<sup>19</sup><https://www.rstudio.com/resources/cheatsheets/>



`geom_bar()`, not the `ggplot()` aesthetic `aes()`, since it doesn't vary. We create a horizontal chart using `coord_flip`, adding `(expand=FALSE)` to eliminate a gap between the chart and the axes, similar to `(xaxs="i")` and `(yaxs="i")` in base. Since we flipped the coordinates, the axis labels are the opposite of what we expect: we use `(ylab="percent")` to label the x-axis. In general the default font sizes in **ggplot2** are quite small, but they can be easily increased by passing a larger base font size to the theme, such as `theme_bw(16)`. The base font size is the font size of the axis labels; by default the plot title is 20% larger and the tick mark labels are 20% smaller. Of course, these relative sizes can be changed by adjusting the theme elements individually.

The next example shows how faceting can be combined with color to show living arrangements of four groups of older male and female age groups (see Appendix A.3 for a data table). Again the structure of the graph allows us to compare down the columns within one age group or along the rows for cross-age comparisons. In the next section we'll consider another way to present the same data.



```
#+ fig.width = 7, fig.height = 4
```

```
library(ggplot2)
data <- read.csv("data/living.csv")
colors2 <- c("lightblue", "skyblue3")
data$LIVING <- factor(data$LIVING, levels =
 c("Other group quarters",
```

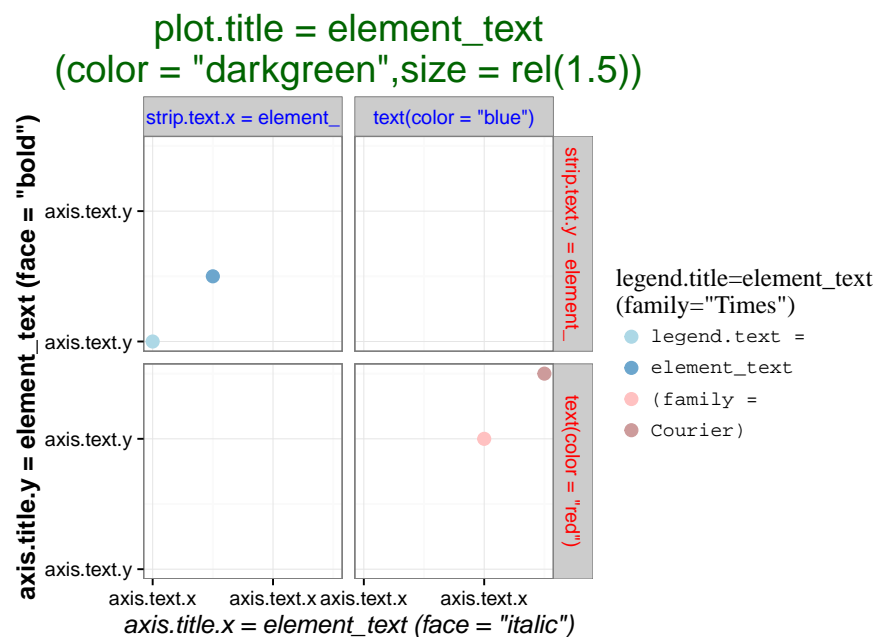
```

 "Nursing home",
 "Alone in household",
 "With others in household"))
twolinelabels <- c("Other group\nquarters",
 "Nursing home",
 "Alone in\nhousehold",
 "With others\nin household")
data$AGE <- factor(data$AGE, levels = c("70-79 yrs",
 "80-89 yrs",
 "90-99 yrs",
 "100+ yrs"))
g <- ggplot(data, aes(x = LIVING, y = PERCENT,
 fill = SEX))
g + geom_bar(stat="identity", position = "dodge") +
 scale_x_discrete(labels = twolinelabels) +
 scale_fill_manual(values = colors2,
 limits = c("Male", "Female")) +
 scale_y_continuous(breaks = c(0, 30, 60)) +
 coord_flip() + facet_grid(. ~ AGE) +
 theme_bw(16) +
 theme(axis.ticks.length = unit(0, "cm"),
 legend.title = element_blank(),
 legend.position = c(.9, -.2),
 legend.key = element_blank(),
 legend.key.size = unit(.8, "lines"),
 panel.grid.major.y = element_blank(),
 panel.grid.minor = element_blank(),
 strip.background = element_rect(fill="grey90"),
 strip.text.x=element_text
 (margin = margin(t=5, b=5)),
 plot.title = element_text(hjust = 0, size = 16,
 face = "bold"),
 plot.margin = margin(8, 8, 24, 8)) +
 xlab(NULL) + ylab("percent") +
 ggtitle ("Living Arrangements of Older Age-Sex Groups")

```

Since the bars will be plotted according to the factor levels, we use `factor()` to set the levels to the desired order. Two line labels are added with `(labels=)` in `scale_x_discrete()`. Colors are set with `(values=)` in `scale_fill_manual`;

we use `(limits=)` in the same line to set the order of the legend entries to match the order of the bars in the chart. In `scale_y_continuous()`, setting `(breaks=)` prevents the x-axis from becoming too crowded with tick mark labels. We make a number of changes to the theme, primarily removing grid lines, borders, and other unnecessary elements from the plot. The process can be expedited by creating a custom theme. We choose not to do that here to make the changes more transparent. The following chart shows additional textual elements that can be altered with `theme()`:



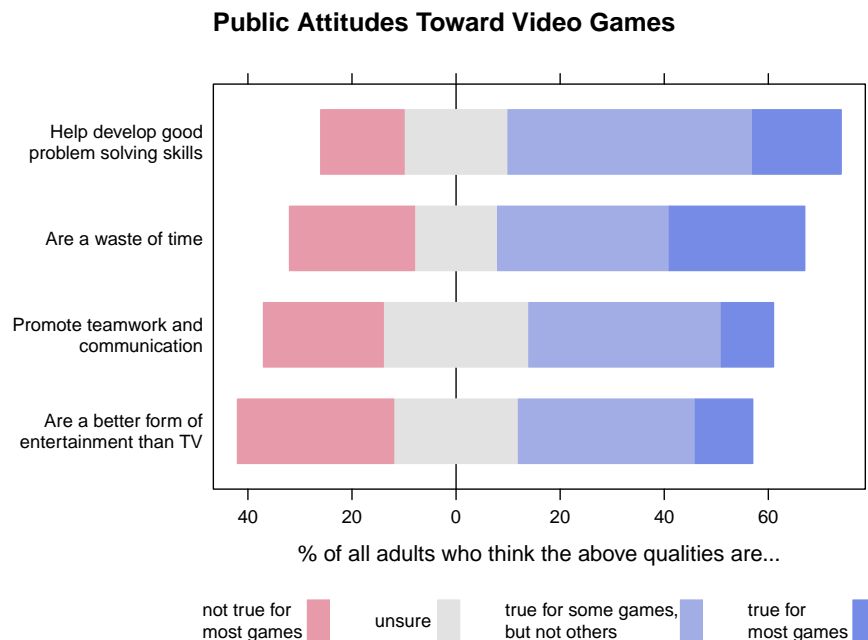
## 7 Special Cases

### 7.1 Diverging stacked bar charts (HH)

Some categorical data, particularly attitudinal research results, are derived from scales with diverging categories. It can be a challenge to plot such data. If, for example, we use bars to represent "Strongly Agree", "Agree", "Neutral", "Disagree", and "Strongly Disagree," we are not able to show that the categories are points along a scale with extremes at both ends. Diverging stacked bar charts are a preferred option in this case since they preserve this aspect of the data. They are well suited for attitudinal "Likert" data but can be used for any categorical data with categories that fall into two opposing groups.

Diverging stacked bar charts look similar to divided bar charts but rather than starting and ending in same place (see section 6.4), the bars are centered on a reference line in the middle of the "neutral" category, or between the least extreme opposing categories if no neutral category exists.

Here we show a diverging stacked bar chart for the results of a public attitude question about video games:



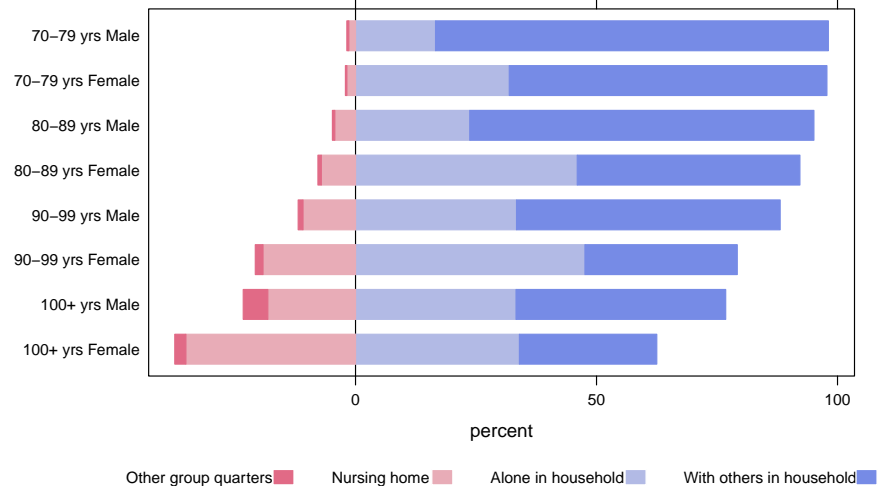
We clearly see from the bars which questions have responses that skew toward the "true" (blue) pole and those that skew to the "not true" (red) pole. The reference line is placed behind the data so as not to make the "unsure" category look like two groups.

```
#+ fig.width = 7, fig.height = 5

library(HH)
Question <- c("Are a waste of time",
 "Help develop good\nproblem solving skills",
 "Promote teamwork and\ncommunication",
 "Are a better form of\nentertainment than TV")
data <- data.frame("not true for\nmost games"
 = c(24,16,23,30),
 "unsure" = c(16,20,28,24),
 "true for some games,\nbut not others"
 = c(33,47,37,34),
 "true for\nmost games" = c(26,17,10,11),
 Question, check.names = FALSE)
likert(Question ~ ., data,
 ReferenceZero = 2,
 positive.order = TRUE,
 main = "Public Attitudes Toward Video Games",
 xlab = paste("% of all adults who think the",
 "above qualities are..."),
 ylab = "")
```

The `likert()` function is found in the **HH** package. The data passed to the function contains a column of responses for each value of the scale and an optional column of statements or questions associated with each row of data. If omitted, row names of the data structure are used. Since we have an even number of categories we specify that the second category ("unsure") is the neutral one with (`ReferenceZero=2`). The other parameter, besides the labels, is (`positive.order=TRUE`), which orders the bars in descending order from the top, beginning with the one that protrudes furthest to the right. Here we show the living arrangements data using the `likert()` function:

Living Arrangments of Older Age–Sex Groups



The diverging stacked bar chart clearly distinguishes between institutional living ("Other group quarters" and "Nursing home") and home living ("Alone in household" and "With others in household"). With the bars ordered, we see that age is the primary factor in determining whether individuals will live at home. In addition, within each age bracket, men are more likely to live at home than women.

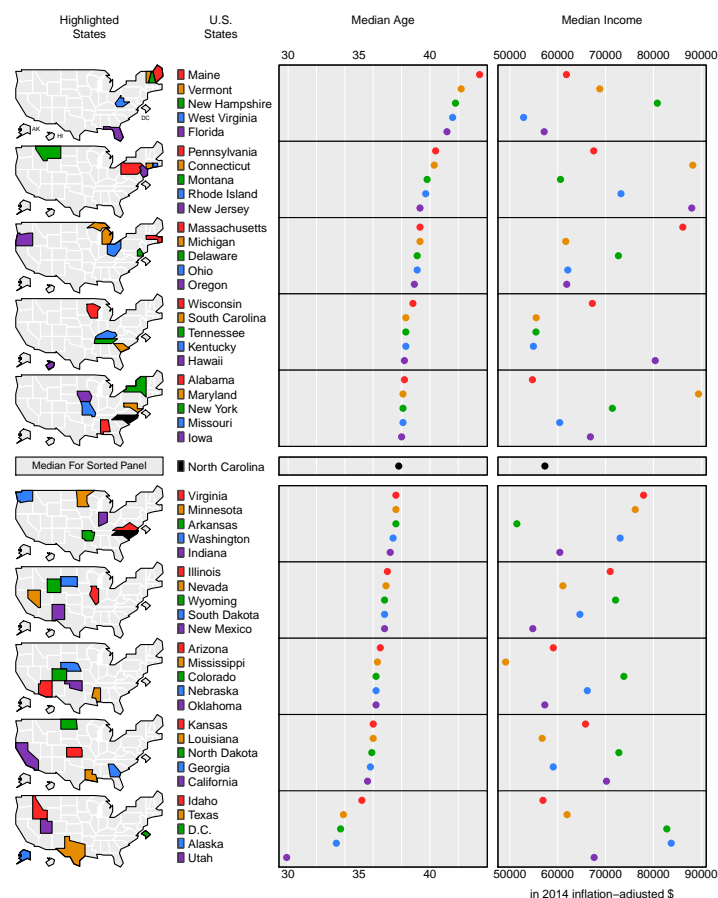
```
#+ fig.width = 8, fig.height = 5
```

```
library(HH)
library(tidyr)
library(dplyr)
data <- read.csv("data/living.csv")
newlevels <- c("Other group quarters", "Nursing home",
 "Alone in household",
 "With others in household")
data$LIVING <- factor(data$LIVING, levels = newlevels)
ldata <- spread(data, key = LIVING, value = PERCENT) %>%
 mutate(GROUP = paste(AGE, SEX))
likert(GROUP~., ldata, positive.order = TRUE,
 main = "Living Arrangments of Older Age-Sex Groups",
 xlab = "percent", ylab = "")
```

There are many creative uses for a diverging stacked bar chart, including age pyramids and profit/loss data. For more info, see [Heiberger and Robbins \(2014\)](#).<sup>20</sup>

## 7.2 Linked micromaps (micromapST)

2010–2014 American Community Survey: State Age and Income



Linked micromaps are extremely useful for geographically referenced data—data for regions such as states, countries, or counties—as they help us visualize geographic patterns in the data. In a linked micromap, a variable of interest is ordered by size and then the data is split into groups of about

<sup>20</sup><https://www.jstatsoft.org/article/view/v057i05>

five, depending on the size of the data set. This format works well since often information is digested better in small carefully chosen chunks than in, as Alberto Cairo puts it, "a single, large, ultra-complex, cluttered display."<sup>21</sup> Color is used to link the regions indicated in the map with the region names and associated data.

The linked micromap plot here was produced with the **micromapST**, a package for drawing linked micromaps for the 50 US states plus the District of Columbia. Other packages are available for other geographic regions.

```
#+ fig.width = 7.5, fig.height = 10

library(micromapST)
data <- read.csv("data/acs2014.csv", row.names = 1)
data <- data[order(data$Income),]

panelDesc <- data.frame(
 type = c("map", "id", "dot", "dot"),
 lab1 = c("", "", "Median Age", "Median Income"),
 lab3 = c("", "", "", "in 2014 inflation-adjusted $"),
 col1 = c(NA, NA, "Age", "Income"))

micromapST(data, panelDesc, rowNames = "full", sortVar="Age",
 ascend = FALSE,
 title = paste("2010-2014 American Community",
 "Survey: State Age and Income"),
 details = list(Title.cex = 1.4))
```

Before plotting, we created a simple data frame with two columns, one for median age and one for median income. See Appendix A.4 for the data source. The row names must contain the state names, plus the District of Columbia. Three formats are acceptable: the full names, two letter abbreviations or two digit FIPS codes. That is the only data the user supplies: all of the geographical data needed to draw the plots is built into the **micromapST** package.

Next we create a data frame, `panelDesc`, that contains information about the structure of the plot. Each element of the `type` variable specifies the graph-

---

<sup>21</sup>"Falling in Love with Micromaps" <http://www.thefunctionalart.com/2013/07/falling-in-love-with-micromaps.html>



ical type for each column of the plot. To keep it simple, we used dot for both of the plot columns though many other options are available: arrow charts, bar charts, stacked bar charts, box plots, dot with confidence intervals, scatterplots and time series plots, many of which require additional columns of data. The variables lab1, lab2, and lab3 provide the title, subtitle, and footer for each column. The last variable col1 specifies the data either by column name or number. For chart types that require more than one column of data, col2 and col3 are included. Finally we set the overall title and title font size with the (details=) parameter. As with many other plots we've described here, this is a basic example. Additional options can be found in the package documentation and Carr and Pickle (2010).

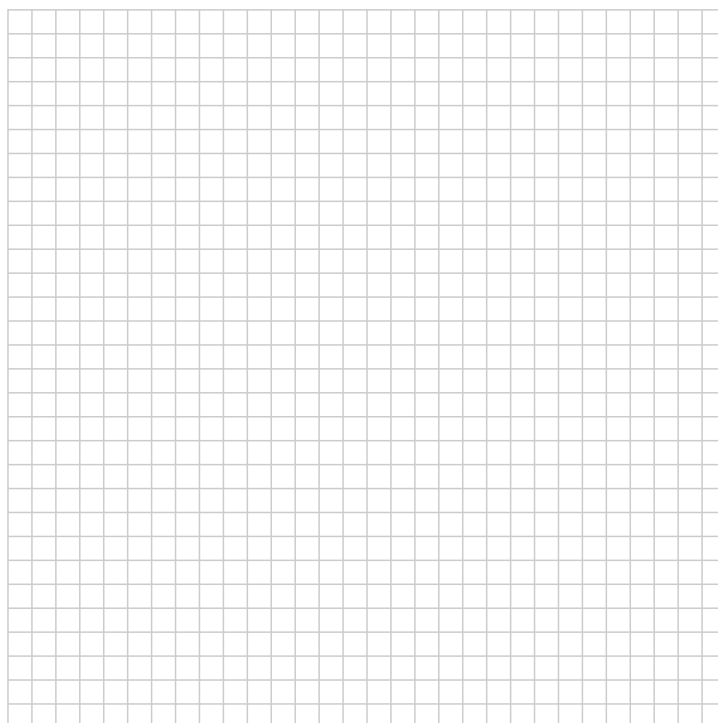
The call to micromapST() draws the plot. In addition to passing data and panelDesc to the function, we add a few other parameters. We indicate that we are using full state names as row names (rowNames="full"), that the variable on which to sort (sortVar=) is "Age", and set (ascend=FALSE) so the data are presented from top to bottom in descending order, i.e. Maine, with the highest median age, appears at the top of the chart.<sup>22</sup> Finally we add a title and specify the title font size with the details parameter.

---

<sup>22</sup>This behavior is different from base graphics, where dot charts are plotted from the bottom up rather than the top down.

## 8 Conclusion

We hope we have inspired you to try some new graphs in R. With over 8000 packages available, and the ability to create your own graphic forms, the possibilities are endless. What will your next graph look like?



We look forward to hearing from you. Please direct comments regarding effective graph design to [naomi@nbr-graphs.com](mailto:naomi@nbr-graphs.com) and R comments to [joycerobbins1@gmail.com](mailto:joycerobbins1@gmail.com).

## Appendices

### A Data and Sources

#### A.1 countries2012.csv

```
data <-read.csv("data/countries2012.csv")
str(data)

'data.frame': 179 obs. of 6 variables:
 $ COUNTRY : Factor w/ 179 levels "Afghanistan",...: 1 2
 3 4 5 6 7 8 9 10 ...
 $ CONTINENT: Factor w/ 6 levels "Africa","Asia",...: 2
 3 1 1 4 6 3 5 3 3 ...
 $ GDP : num 691 4247 5584 5532 13526 ...
 $ TFR : num 5.27 1.76 2.91 6.25 2.1 ...
 $ LIFEEXP : num 59.7 77.4 74.3 51.5 75.6 ...
 $ CHMORT : num 99.5 15.5 26.1 172.2 9.1 ...
```

(header only, numerical data rounded)

	COUNTRY	CONTINENT	GDP	TFR	LIFEEXP	CHMORT
1	Afghanistan	Asia	690.8	5.3	59.7	99.5
2	Albania	Europe	4247.5	1.8	77.4	15.5
3	Algeria	Africa	5583.6	2.9	74.3	26.1
4	Angola	Africa	5531.8	6.3	51.5	172.2

source: <http://databank.worldbank.org/>

## A.2 fathers.txt

```
data <-read.table("data/fathers.txt")
str(data)

'data.frame': 4 obs. of 5 variables:
 $ AGE : Factor w/ 4 levels "< 20 yrs","20-24 yrs",...:
 1 2 3 4
 $ NOHSDEG : int 21 49 21 9
 $ HSDEG : int 15 45 27 13
 $ SOMECOLL: int 10 35 35 20
 $ COLLDEG : int 3 12 42 43
```

	AGE	NOHSDEG	HSDEG	SOMECOLL	COLLDEG
1	< 20 yrs	21	15	10	3
2	20-24 yrs	49	45	35	12
3	25-29 yrs	21	27	35	42
4	30-44 yrs	9	13	20	43

Source: <http://www.pewresearch.org/fact-tank/2015/06/19/college-educated-men-take-their-time-becoming-dads/>

## A.3 living.csv

```
data <-read.csv("data/living.csv")
str(data)

'data.frame': 32 obs. of 4 variables:
 $ AGE : Factor w/ 4 levels "100+ yrs","70-79 yrs",...:
 2 2 2 2 2 2 2 2 3 3 ...
 $ SEX : Factor w/ 2 levels "Female","Male": 2 1 2 1 2
 1 2 1 2 1 ...
 $ LIVING : Factor w/ 4 levels "Alone in household",...:
 4 4 1 1 2 2 3 3 4 4 ...
 $ PERCENT: num 81.5 65.9 16.6 31.9 1.4 1.8 0.4 0.3
 71.4 46.2 ...
```

Source: <https://www.census.gov/prod/cen2010/reports/c2010sr-03.pdf>

living.csv (cont.)

	AGE	SEX	LIVING	PERCENT
1	70-79 yrs	Male	With others in household	81.5
2	70-79 yrs	Female	With others in household	65.9
3	70-79 yrs	Male	Alone in household	16.6
4	70-79 yrs	Female	Alone in household	31.9
5	70-79 yrs	Male	Nursing home	1.4
6	70-79 yrs	Female	Nursing home	1.8
7	70-79 yrs	Male	Other group quarters	0.4
8	70-79 yrs	Female	Other group quarters	0.3
9	80-89 yrs	Male	With others in household	71.4
10	80-89 yrs	Female	With others in household	46.2
11	80-89 yrs	Male	Alone in household	23.7
12	80-89 yrs	Female	Alone in household	46.0
13	80-89 yrs	Male	Nursing home	4.3
14	80-89 yrs	Female	Nursing home	7.1
15	80-89 yrs	Male	Other group quarters	0.5
16	80-89 yrs	Female	Other group quarters	0.7
17	90-99 yrs	Male	With others in household	54.7
18	90-99 yrs	Female	With others in household	31.6
19	90-99 yrs	Male	Alone in household	33.4
20	90-99 yrs	Female	Alone in household	47.6
21	90-99 yrs	Male	Nursing home	10.9
22	90-99 yrs	Female	Nursing home	19.2
23	90-99 yrs	Male	Other group quarters	1.0
24	90-99 yrs	Female	Other group quarters	1.6
25	100+ yrs	Male	With others in household	43.5
26	100+ yrs	Female	With others in household	28.5
27	100+ yrs	Male	Alone in household	33.3
28	100+ yrs	Female	Alone in household	34.0
29	100+ yrs	Male	Nursing home	18.2
30	100+ yrs	Female	Nursing home	35.2
31	100+ yrs	Male	Other group quarters	5.1
32	100+ yrs	Female	Other group quarters	2.3

## A.4 acs2014.csv

```
data <-read.csv("data/acs2014.csv", row.names = 1)
str(data)

'data.frame': 51 obs. of 2 variables:
 $ Age : num 38.2 33.4 36.5 37.6 35.6 36.2 40.3 39.1
 33.7 41.2 ...
 $ Income: int 54724 83714 59088 51464 70187 73817
 88217 72683 82791 57176 ...
```

(header only)

	Age	Income
Alabama	38.2	54724
Alaska	33.4	83714
Arizona	36.5	59088
Arkansas	37.6	51464
California	35.6	70187
Colorado	36.2	73817

Source: 2010-2014 American Community Survey 5-Year Estimates

Table: GCT0101 - "Median Age of the Total Population"

[http://factfinder.census.gov/bkmk/table/1.0/en/ACS/14\\_5YR/GCT0101.US01PR](http://factfinder.census.gov/bkmk/table/1.0/en/ACS/14_5YR/GCT0101.US01PR)

Table: GCT1902 - "Median Family Income"

[http://factfinder.census.gov/bkmk/table/1.0/en/ACS/14\\_5YR/GCT1902.US01PR](http://factfinder.census.gov/bkmk/table/1.0/en/ACS/14_5YR/GCT1902.US01PR)

## B Base Graphics Cheatsheet

### SET GRAPHICAL PARAMETERS

*the following can only be set with `par()`*

**`par(...)`**

<i>multiple plots</i>	<code>mfcop = c(nrow,ncol)</code>	<i>plot margins (outer)</i>	<code>oma = c(bottom, left, top, right)</code> default: <code>c(0, 0, 0, 0)</code> lines
<i>plot margins</i>	<code>mar = c(bottom, left, top, right)</code> default: <code>c(5.1, 4.1, 4.1, 2.1)</code> lines	<i>query x &amp; y limits</i>	<code>par("usr")</code>

### CREATE A NEW PLOT

<b>Bar charts</b>	<b><code>barplot(height, ...)</code></b>	<b>Histograms</b>	<b><code>hist(x, ...)</code></b>
<i>bar labels</i>	<code>names.arg =</code>	<i>breakpts</i>	<code>breaks =</code>
<i>border</i>	<code>border =</code>		
<i>fill color</i>	<code>col =</code>	<b>Line charts</b>	<b><code>plot(x, type = "l")</code></b>
<i>horizontal</i>	<code>horiz = TRUE</code>	<i>line type</i>	<code>lty =</code> "blank" 0 "solid" 1 "dashed" 2 "dotted" 3
<b>Box plots</b>	<b><code>boxplot(x, ...)</code></b>	<i>line width</i>	<code>lwd =</code>
<i>horizontal</i>	<code>horizontal = TRUE</code>		
<i>box labels</i>	<code>names =</code>		
<b>Dot plots</b>	<b><code>dotchart(x, ...)</code></b>	<b>Scatterplots</b>	<b><code>plot(x, ...)</code></b>
<i>dot labels</i>	<code>labels =</code>	<i>symbol</i>	<code>pch =</code>

### REMOVE

<i>axis labels</i>	<code>ann = FALSE</code>
<i>axis, tickmarks, and labels</i>	<code>xaxt = "n"</code> <code>yaxt = "n"</code>
<i>plot box</i>	<code>bty = "n"</code>

*NOTE: Many of the parameters here can be also be set in `par()`. See R help for more options.*

### ADJUST

<i>allow plotting out of plot region</i>	<code>xpd = TRUE</code>
<i>aspect ratio</i>	<code>asp =</code>
<i>axis limits</i>	<code>xlim =, ylim =</code>
<i>axis lines to match axis limits</i>	<code>xaxs = "i", yaxs = "i"</code> (internal axis calculation)

## ADD TEXT

	<b>location</b>		<b>size</b> <i>(magnification factor)</i>
<i>axis labels</i>	xlab =, ylab =	<i>all elements</i>	cex =
<i>subtitle</i>	sub =	<i>axis labels</i>	cex.lab =
<i>title</i>	main =	<i>subtitle</i>	cex.sub =
	<b>style</b>	<i>tick mark labels</i>	cex.axis =
<i>font face</i>	font = 1 ( <i>plain</i> ) 2 ( <i>bold</i> ) 3 ( <i>italic</i> ) 4 ( <i>bold italic</i> )	<i>title</i>	cex.main =
	<b>font family</b>		<b>position</b>
	family = "serif" "sans" "mono"	<i>text direction</i>	las = 1 (horizontal)
		<i>justification</i>	adj = 0 .5 1 (left, center, right)

## ADD TO AN EXISTING PLOT

<b>Add new plot</b>	<b>[any plot function]</b> (..., add = TRUE) ex. barplot(x, add = TRUE)	<b>Lines</b>	<b>lines (x,...)</b> line style lty = line width lwd = color col =
<b>Axes</b>	<b>axis (side,... )</b> location side = 1 2 3 4 (bottom, left, top, right)	<b>Points</b>	<b>points (x,...)</b> symbol pch = <div style="display: flex; flex-wrap: wrap; font-family: monospace; font-size: 0.8em;"> <div style="margin-right: 10px;"> □ ○ △ + × ◇ ▽ ☒ * ✎ ⊕ ⊗ ⊞  0 1 2 3 4 5 6 7 8 9 10 11 12 </div> <div> ☒ ☒ ■ ● ▲ ◆ ● ● ○ □ ◇ △ ▽  13 14 15 16 17 18 19 20 21 22 23 24 25 </div> </div> color col = fill color bg = (pch: 21-25 only)
<b>tick mark:</b>	<b>labels</b> labels = <b>location</b> at = <b>remove</b> tick = FALSE <b>rotate text</b> las = 1 (horizontal)	<b>Text</b>	<b>text (x, y, text,...)</b> position pos = 1 2 3 4 (rel. to x,y) (below, left, above, right) (default=center)
<b>Axis labels</b>	<b>mtext (text,... )</b> location side = 1 2 3 4 (bottom, left, top, right) lines to skip line = (from plot region, default = 0) position at = x or y-coord (depending on side) justification adj = 0 .5 1 (left, center, right)	<b>Title</b>	<b>title (main,...)</b> axis labels xlab =, ylab = subtitle sub = title main =



## References

- Carr, Daniel B. and Linda Williams Pickle. 2010. *Visualizing Data Patterns with Micromaps*. CRC Press.
- Chang, Winston. 2013. *R Graphics Cookbook*. O'Reilly Media.
- Cleveland, William S. 1994. *The Elements of Graphing Data*. 2nd ed. Hobart Press.
- Heiberger, Richard, and Burt Holland. 2015. *Statistical Analysis and Data Display: An Intermediate Course with Examples in R*, 2nd ed. Springer.
- Heiberger, Richard M., and Naomi B. Robbins. 2014. "[Design of Diverging Stacked Bar Charts for Likert Scales and Other Applications.](#)" *Journal of Statistical Software*, 57 (5).
- Jacoby, William G. 1998. *Statistical Graphics for Visualizing Multivariate Data*. Sage.
- Murrell, Paul. 2011. *R Graphics*, 2nd ed. CRC Press.
- Robbins, Naomi B. 2013 [2004]. *Creating More Effective Graphs*. Chart House.
- Tufte, Edward R. 2001. *The Visual Display of Quantitative Information*, 2nd ed. Graphics Press.
- Wickham, Hadley. 2016. *ggplot2: Elegant Graphics for Data Analysis*, 2nd ed. Springer.
- Wilkinson, Leland. 2005. *The Grammar of Graphics*, 2nd ed. Springer.
- Xie, Yihui. 2015. *Dynamic Documents with R and knitr*, 2nd ed. CRC Press.

## Index

- `abline()`, 13–15, 19
- Anscombe’s quartet, 4–5
- `as.matrix()`, 37
- `axis()`, 30
- axis limits, 12
- bar charts
  - bar percent charts, 33–34
  - diverging stacked bar charts, 44–47
  - divided bar charts, 36–38
  - faceted bar charts, 39–43
  - grouped bar charts, 38–39
  - simple, 11–13
- bar percent charts, 33–34
- `barplot()`, 11–13, 37
- base graphics, 7–8
- Bell Laboratories, 5
- box plots, 18–19
- `boxplot()`, 19
- Cairo, Alberto, 48
- Carr, Daniel, 49
- Chang, Winston, 40
- chart choosers, problems, 10
- Cleveland, William, 5, 10, 13, 32
- `colors()`, 13
- computation time, 7
- `coord_flip()`, 41
- correlation, 23, 27, 29
- CRAN, 6, 7
- datasets
  - fathers age and education, 32–41
  - fertility rates, 11–19, 23–31
  - living arrangements, 41–43, 45–46
  - road casualties, 21–22
  - US population, 20–21
  - US states, median age and income, 47–49
  - video game attitudes, 44–45
  - world development, 30–31
- distributions, 16
- diverging stacked bar charts, 44–47
- divided bar charts, 36–38
- dot plots, 13–15
- `dotchart()`, 14
- `droplevels()`, 29
- effective graphs, 10
- effective graphs blog on *Forbes*, 16
- examples, *see* datasets
- exploratory data analysis, 10
- faceted bar charts, 39–43
- `factor()`, 42
- factor level order, 19, 26, 37, 42
- figure height, 13
- figure width, 13
- functions, creating, 14
- functions, *see* code, 14
- `geom_bar()`, 41
- graphics devices, 8–9
- gridlines, 13, 14
- grouped bar charts, 38–39
- Heiberger, Richard, 10, 47
- histograms, 16–18
- horizontal axis labels, 12
- horizontal bars, 12
- horizontal lines, 13
- Jacoby, William, 27
- labeling data points, 24

- legend(), 26
- legends, 26, 29, 37–38
- length, judging, 12, 13, 32
- likert(), 10, 45
- line breaks in plot text, 33
- line charts, 20–22
- linked micromaps, 47–49
- margins, 12, 35
- micromapST(), 49
- Microsoft R Open, 6–7
- Microsoft R Portal, 6, 7
- month plots, 22
- monthplot(), 22
- mtext(), 36
- multidimensional data, 26, 29, 39
- multiple pie charts, 34–36
- Murrell, Paul, 12
- names(), 12
- overlapping symbols, 14
- packages, 6–8
  - checkpoint, 7
  - dplyr, 40
  - ggplot2, 8, 39–43
  - grid, 8
  - HH, 8, 10, 44–47
  - knitr, 9, 13
  - lattice, 8, 26, 27, 29, 39
  - MASS, 8, 30
  - micromapST, 8, 48
  - tidyr, 40
- par(), 35, 37
- parallel coordinate plots, 29–31
- parcoord(), 30, 31
- parts of a whole, 32–43
- perception, 5
- Pickle, Linda, 49
- pie(), 32
- pie charts, 32–33
  - multiple pie charts, 34–36
- plot(), 20, 23
- plot sizes, 9
- position, judging, 13, 32
- R Help, 7
- R, advantages of, 5–6
- relationships, 23–29
- reproducibility, 6
- Revolution Analytics, 6
- Robbins, Naomi, 10, 47
- RStudio cheatsheets, 40
- S language, 5–6, 39
- scale\_fill\_manual(), 42
- scale\_x\_discrete(), 42
- scale\_y\_continuous(), 43
- scatterplot matrices, 26–29
- scatterplots, 23–26
- Schumacher, Aaron, 16
- splom(), 26, 27
- text(), 24
- theme(), 43
- theme\_bw(), 41
- time series, 22, 49
- trellis displays, 39
- trends over time, 20–22
- ts(), 21
- Tufte, Edward, 10
- Tukey, John, 5
- vertical lines, 13
- Wickham, Hadley, 40
- Wilkinson, Leland, 39

## About the Authors

**Naomi B. Robbins** is a consultant and seminar leader who specializes in the graphical display of data. She trains employees of corporations and organizations on the effective presentation of data. She also reviews documents and presentations for clients, suggesting improvements or alternative presentations as appropriate. She is the author of *Creating More Effective Graphs*, first published by John Wiley (2005) and now by Chart House (2013). In addition to her one and two day seminars on creating more effective graphs, she offers short programs including "Recognizing Misleading and Deceptive Graphs" and "How to Avoid Common Graphical Mistakes." Dr. Robbins received her Ph.D. in mathematical statistics from Columbia University, M.A. from Cornell University, and A.B. from Bryn Mawr College. Naomi is active in the American Statistical Association and is the immediate past-chair of the Statistical Graphics Section. She is the organizer for the Data Visualization New York Meetup. She had a long career at Bell Laboratories before forming the consulting practice, NBR.

**Joyce Robbins** specializes in the communication of social data, with expertise in both quantitative and qualitative methods. She received her Ph.D. in sociology from Columbia University, her M.A. in sociology and anthropology from Tel Aviv University and her B.S.E. in civil engineering and operations research from Princeton University. Before joining NBR, she was Assistant Professor of Sociology at Touro College in New York City, and prior to that taught high school mathematics and computer science.

## Acknowledgments

Many thanks to David Smith of Microsoft for inviting us to write this monograph, and to Terry Christiani for coordinating the project. We are grateful for the detailed feedback we received from Karen Bryan, Keith Chamberlain, Richard Heiberger, John Rosenfelder, and David Smith. Finally, we thank Julian Glickman for assisting with research.