A *ggplot2* Primer

Ehssan Ghashim[1], Patrick Boily[1,2,3,4]

Abstract
R has become one of the world's leading languages for statistical and data analysis. While the base R installation does suppor simple visualizations, its plots are rarely of high-enough quality for publication. Enter Hadley Wickam's *ggplot2*, an aesthetically and logical approach to data visualization. In this short report, we introduce *gglot2*'s graphic grammar elements, and present a number of examples.

Keywords
R, *ggplot2*, data visualization

[1]Centre for Quantitative Analysis and Decision Support, Carleton University, Ottawa
[2]Sprott School of Business, Carleton University, Ottawa
[3]Department of Mathematics and Statistics, University of Ottawa, Ottawa
[4]Idlewyld Analytics and Consulting Services, Wakefield, Canada
Email: patrick.boily@carleton.ca

Contents

1. Introduction

There are currently four graphical systems available in R.

1. The *base* graphics system, written by Ross Ihaka, is included in every R installation. Most of the graphs produced in the 'Basics of R' report rely on base graphics functions.

2. The *grid* graphics system, written by Paul Murrell in 2011, is implemented through the `grid` package, which offers a lower-level alternative to the standard graphics system. The user can create arbitrary rectangular regions on graphics devices, define coordinate systems for each region, and use a rich set of drawing primitives to control the arrangement and appearance of graphic elements.

   This flexibility makes grid a valuable tool for software developers. But the grid package doesn't provide functions for producing statistical graphics or complete plots. As a result, it is rarely used directly by data analysts and won't be discussed further (see Dr. Murrell's Grid website at **http://mng.bz/C86p**).

3. The *lattice* package, written by Deepayan Sarkar in 2008, implements trellis graphs, as outlined by Cleveland (1985, 1993). Basically, trellis graphs display the distribution of a variable or the relationship between variables, separately for each level of one or more other variables. Built using the *grid* package, the *lattice* package has grown beyond Cleveland's original approach to visualizing multivariate data and now provides a comprehensive alternative system for creating statistical graphics in R.

4. Finally, the *ggplot2* package, written by Hadley Wickham [2], provides a system for creating graphs based on the grammar of graphics described by Wilkinson (2005) and expanded by Wickham [3]. The intention of the *ggplot2* package is to provide a comprehensive, grammar-based system for generating graphs in a unified and coherent manner, allowing users to create new and innovative data visualizations. The power of this approach has led to *ggplot2* becoming one of the most common R data visualization tool.

Access to the four systems differs: they are all included in the base installation, except for *ggplot2*, and they must all be explicitly loaded, except for the *base* graphics system.

2. How *ggplot2* Works

As we saw in *Basics of R for Data Analysis*, visualization involves representing data using various elements, such as lines, shapes, colours, etc.. There is a structured relationship – some mapping – between the variables in the data and their representation in the displayed plot. We also saw that not all mappings make sense for all types of variables, and (independently), that some representations are harder to interpret than others.

ggplot2 provides a set of tools to map data to visual display elements and to specify the desired type of plot, and subsequently to control the fine details of how it will be displayed. Figure 1 shows a schematic outline of the process starting from data, at the top, down to a finished plot at the bottom.

The most important aspect of *ggplot2* is the way it can be used to think about the logical structure of the plot. The code allows the user to explicitly state the connections between the variables and the plot elements that are seen on the screen – items such as points, colors, and shapes.

In *ggplot2*, these logical connections between the data and the plot elements are called **aesthetic mappings**, or simply **aesthetics**.

After installing and loading the package, a plot is created by telling the `ggplot()` function what the data is, and how the variables in this data logically map onto the plot's aesthetics.

The next step is to specify what sort of plot is desired (scatterplot, boxplot, bar chart, etc), also known as a **geom**. Each geom is created by a specific function:

- `geom_point()` for scatterplots
- `geom_bar()` for barplots
- `geom_boxplot()` for boxplots,
- and so on.

These two components are combined, literally adding them together in an expression, using the "+" symbol.

At this point, *ggplot2* has enough information to draw a plot – the other components (see Figure 1) provide additional design elements.

If no further details are specified, *ggplot2* uses a set of sensible default parameters; usually, however, the user will want to be more specific about, say, the scales, the labels of legends and axes, and other guides that can improve the plot readability.

These additional pieces are added to the plot in the same manner as the `geom_ function()` component, with specific arguments, again using the "+" symbol. Plots are built systematically in this manner, piece by piece.

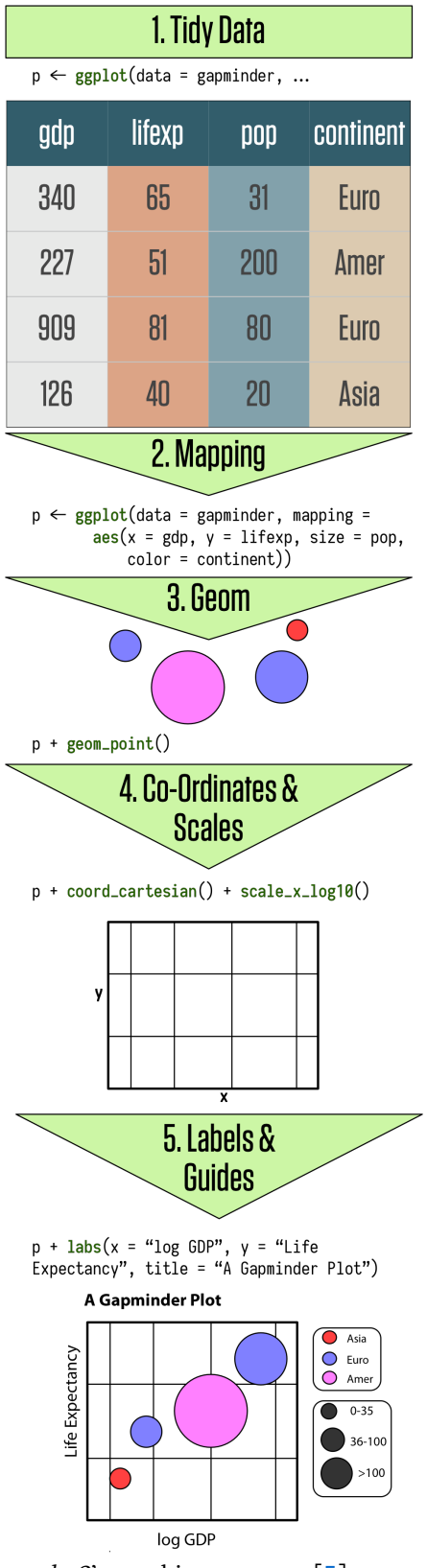

Figure 1. *ggplot2*'s graphics grammar [5].

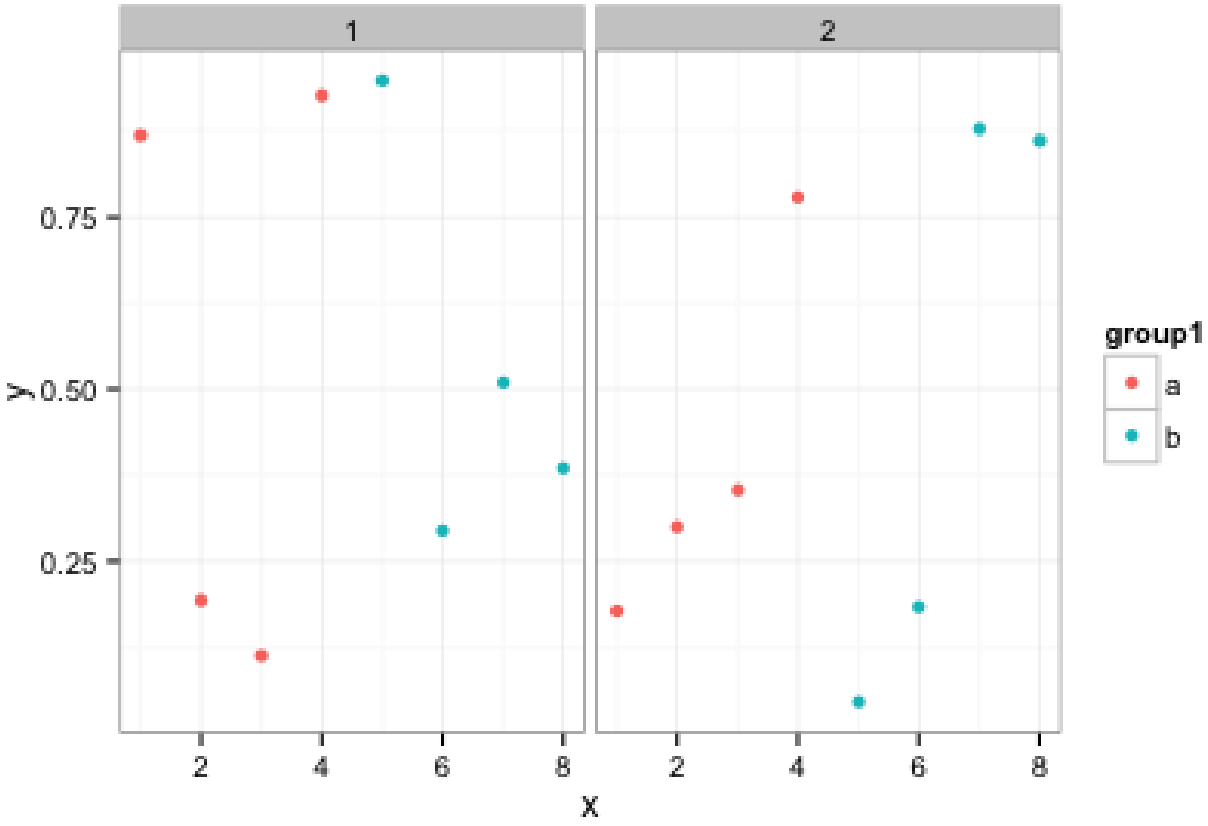

Figure 2. Artificial data - visualization.

3. Basics of *ggplot2* Grammar

Let's look at some illustrative *ggplot2* code:

```
library("ggplot2")
theme_set(theme_bw()) # use the black
    and white theme throughout
# artificial data:
d <- data.frame(x = c(1:8, 1:8), y =
    runif(16),
  group1 = rep(gl(2, 4, labels = c("a",
      "b")), 2),
  group2 = gl(2, 8))
head(d)
## R output
## x     y       group1 group2
## 1 1 0.8683116 a     1
## 2 2 0.1934542 a     1
## 3 3 0.1131743 a     1
## 4 4 0.9260514 a     1
## 5 5 0.9476787 b     1
## 6 6 0.2949107 b     1
ggplot(data = d) + geom_point(aes(x, y,
    colour = group1)) +
  facet_grid(~group2)
```

The data is plotted in Figure 2.

A basic display call contains the following elements:

- `ggplot()`: start an object and specify the data
- `geom_point()`: we want a scatter plot; this is called a "geom"
- `aes()`: specifies the "aesthetic" elements; a legend is automatically created
- `facet_grid()`: specifies the "faceting" or panel layout

Other components include statistics, scales, and annotation options. At a bare minimum, charts require a dataset, some aesthetics, and a geom, combined, as above, with "+" symbols! This non-standard approach has the advantage of allowing *ggplot2* plots to be proper R objects, which can modified, inspected, and re-used.

ggplot2's main plotting functions are `qplot()` and `ggplot()`; `qplot()` is short for "quick plot" and is meant to mimic the format of base R's `plot()`; it requires less syntax for many common tasks, but has limitations – it's essentially a wrapper for `ggplot()`, which is not itself that complicated to use.

We will focus on this latter function.

Function	Adds	Options
`geom_bar()`	Bar chart	color, fill, alpha
`geom_boxplot()`	Box plot	color, fill, alpha, notch, width
`geom_density()`	Density plot	color, fill, alpha, linetype
`geom_histogram()`	Histogram	color, fill, alpha, linetype, binwidth
`geom_hline()`	Horizontal lines	color, alpha, linetype, size
`geom_jitter()`	Jittered points	color, size, alpha, shape
`geom_line()`	Line graph	colorvalpha, linetype, size
`geom_point()`	Scatterplot	color, alpha, shape, size
`geom_rug()`	Rug plot	color, side
`geom_smooth()`	Fitted line	method, formula, color, fill, linetype, size
`geom_text()`	Text annotations	Many; see the help for this function
`geom_violin()`	Violin plot	color, fill, alpha, linetype
`geom_vline()`	Vertical lines	color, alpha, linetype, size

Option	Specifies
color	colour of points, lines, and borders around filled regions
fill	colour of filled areas such as bars and density regions
alpha	transparency of colors, ranging from 0 (fully transparent) to 1 (opaque)
linetype	pattern for lines (1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash)
size	point size and line width
shape	point shapes (same as pch, with 0 = open square, 1 = open circle, 2 = open triangle, and so on)
position	position of plotted objects such as bars and points. For bars, "dodge" places grouped bar charts side by side, "stacked" vertically stacks grouped bar charts, and "fill" vertically stacks grouped bar charts and standardizes their heights to be equal; for points, "jitter" reduces point overlap
binwidth	bin width for histograms
notch	indicates whether box plots should be notched (TRUE/FALSE)
sides	placement of rug plots on the graph ("b" = bottom, "l" = left, "t" = top, "r" = right, "bl" = both bottom and left, and so on)
width	width of box plots

Table 1. Commonly-used `geom` functions (top); common options for the various geom functions (bottom).

4. Specifying Plot Types with geoms

Whereas `ggplot()` specifies the data source and variables to be plotted, the various `geom` functions specify how these variables are to be visually represented (using points, bars, lines, and shaded regions). There are currently 37 available geoms. Table 1 lists the more common ones, along with frequently used options (most of the graphs shown in this report can be created using those `geoms`).

For example, the next bit of code produces a histogram of the heights of singers in the 1979 edition of the New York Choral Society (Figure 4), and a display of height by voice part for the same data (Figure 5).

```
library("ggplot2")
data(singer, package="lattice")
ggplot(singer, aes(x=height)) +
    geom_histogram()
ggplot(singer, aes(x=voice.part,
    y=height)) + geom_boxplot()
```

From Figure 5, it appears that basses tend to be taller and sopranos tend to be shorter. Although the singers' gender was not recorded, it probably accounts for much of the variation seen in the diagram.

Note that only the `x` variable (height) was specified when creating the histogram, but that both the `x` (voice part) and the `y` (height) variables were specified for the box plot – indeed, `geom_histogram()` defaults to counts on the y–axis when no `y` variable is specified (each function's documentation contains details and additional examples, but there's a lot of value to be found in playing around with data in order to determine their behaviour).

Let's examine the use of some of these options using the `Salaries` dataset (from package "car"). The dataframe contains information on the salaries of university professors collected during the 2008–2009 academic year. Variables include rank (AsstProf, AssocProf, Prof), sex (Female, Male), yrs.since.phd (years since Ph.D.), yrs.service (years of service), and salary (nine-month salary in dollars). The next code produces the plot in Figure 3.

```
data(Salaries, package="car")
library(ggplot2)
ggplot(Salaries, aes(x=rank, y=salary)) +
      geom_boxplot(fill="cornflowerblue",
      color="black", notch=TRUE)+
      geom_point(position="jitter",
          color="blue", alpha=.5)+
      geom_rug(side="l", color="black")
```

Figure 3 displays notched box plots of salary by academic rank. The actual observations (teachers) are overlaid and given some transparency so they don't obscure the box plots. They're also littered to reduce their overlap. Finally, a rug plot is provided on the left to indicate the general spread of salaries. From Figure 3, we see that the salaries of assistant, associate, and full professors differ significantly from each other (there is no overlap in the box plot notches).

Additionally, the variance in salaries increases with greater rank, with a larger range of salaries for full professors. In fact, at least one full professor earns less than all assistant professors. There are also three full professors whose salaries are so large as to make them outliers (as indicated by the black dots in the Prof box plot).

5. Aesthetics

Aesthetics refer to the displayed attributes of the data. They map the data to an attribute (such as the size or shape of a marker) and generate an appropriate legend. Aesthetics are specified with the `aes()` function.

The aesthetics available for `geom_point()`, as an example are:

- `x`
- `y`
- `alpha`
- `color`
- `fill`
- `shape`
- `size`

Note that `ggplot()` tries to accommodate the user who's never "suffered" through base graphics before by using intuitive arguments like `color`, `size`, and `linetype`, but `ggplot()` also accepts arguments such as `col`, `cex`, and `lty`. The documentation goes some way towards explaining aesthetic options exist for each geom (they're generally self-explanatory).

Aesthetics can be specified within the data function or within a geom. If they're specified within the data function then they apply to all specified `geoms`.

Note the important difference between specifying characteristics like colour and shape inside or outside the `aes()` function: those inside it are assigned colour or shape automatically based on the data. If characteristics like colour or shape are defined outside the `aes()` function, then they will not be mapped to data.

Here's an example, using the `mpg` dataset:

```
ggplot(mpg, aes(cty, hwy)) +
    geom_point(aes(colour = class))
ggplot(mpg, aes(cty, hwy)) +
    geom_point(colour = "red")
```

The outputs are shown in Figure 6.

6. Facets

In *ggplot2* parlance, small multiples are referred to as *facets*. There are two kinds:

- `facet_wrap()`
- `facet_grid()`

The former plots the panels in the order of the factor levels – when it gets to the end of a column it wraps to the next column (the number of columns and rows can be specified with `nrow` and `ncol`. The grid layout `facet_grid()` produces a grid with explicit `x` and `y` positions.

By default, the panels all share the same x and y axes. Note, however, that the various y—axes are allowed to vary *via*

```
facet_wrap(scales = "free_y"),
```

and that all axes are allowed to vary

via `facet_wrap(scales = free)`.

To specify the data frame columns that are mapped to the rows and columns of the facets, separate them with a tilde. Usually, only a row or a column is fed to `facet_wrap()`. What happens if both are fed to that component?

Going back to the choral example, a faceted graph can be produced using the following code:

```
data(singer, package="lattice")
library(ggplot2)
ggplot(data=singer, aes(x=height)) +
      geom_histogram() +
      facet_wrap(~voice.part, nrow=4)
```

The resulting plot (Figure 7) displays the distribution of singer heights by voice part. Separating the height distribution into their own small, side-by-side plots makes them easier to compare.

As a second example, let's create a graph that has faceting and grouping:The resulting graph is presented in Figure 8. It contains the same information, but separating the plot into facets makes it somewhat easier to read.

```
library(ggplot2)
ggplot(Salaries, aes(x=yrs.since.phd,
    y=salary, color=rank,
      shape=rank)) + geom_point() +
          facet_grid(.~sex)
```

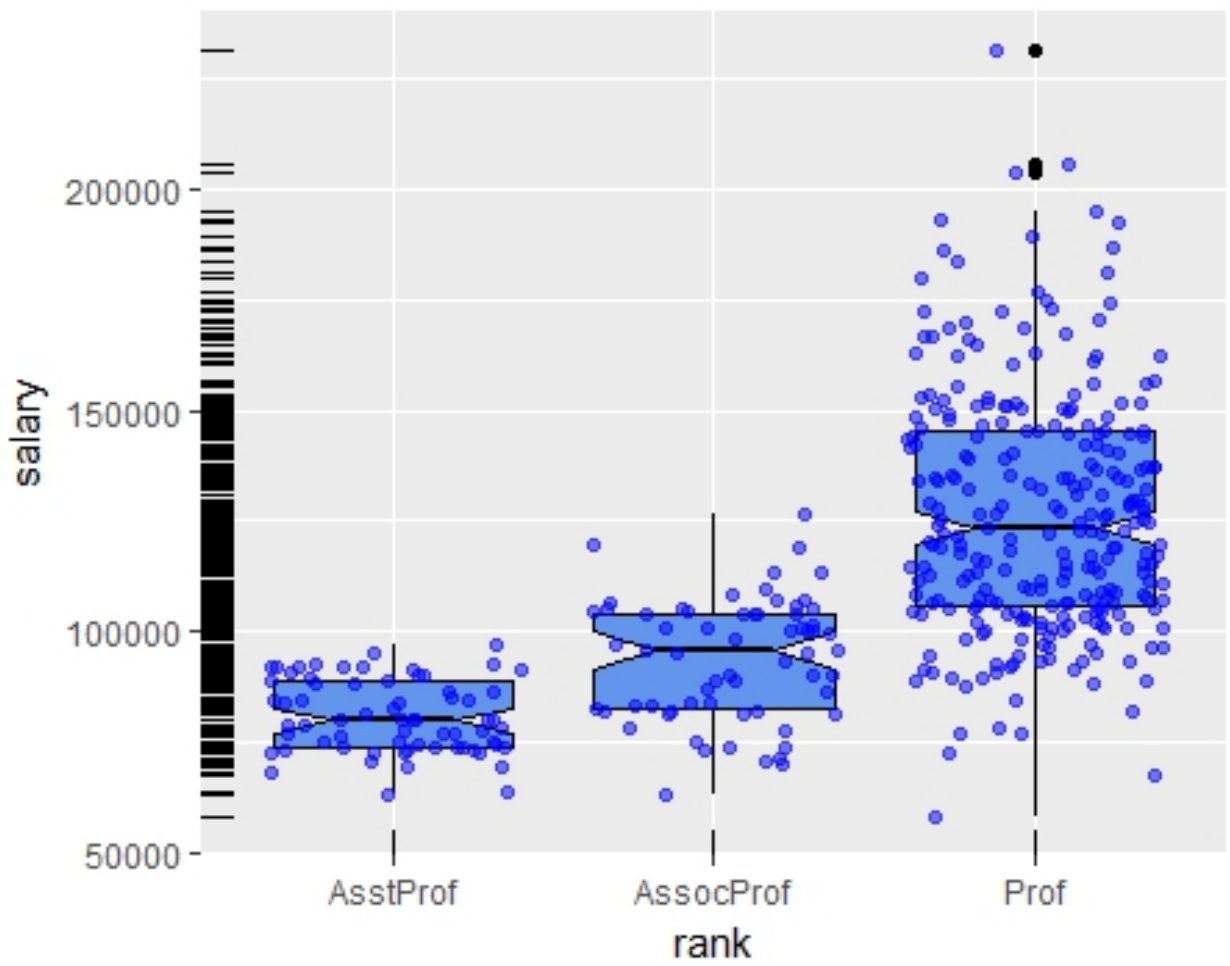

Figure 3. Notched box plots with superimposed points describing the salaries of college professors by rank. A rug plot is provided on the vertical axis.

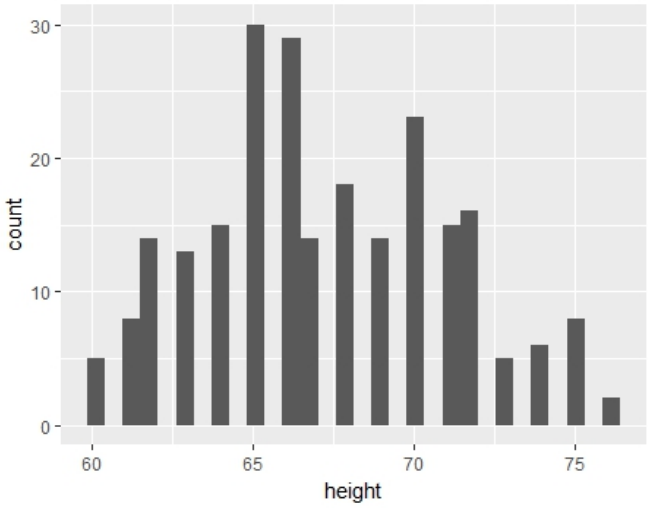

Figure 4. Histogram of singer heights

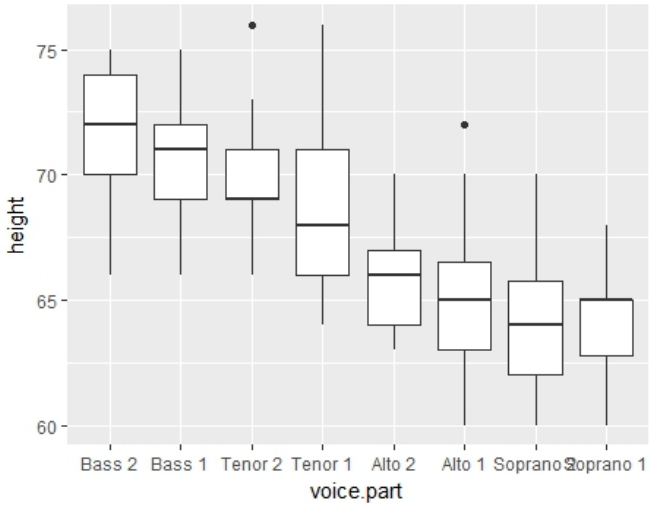

Figure 5. Box plot of singer heights by voice part

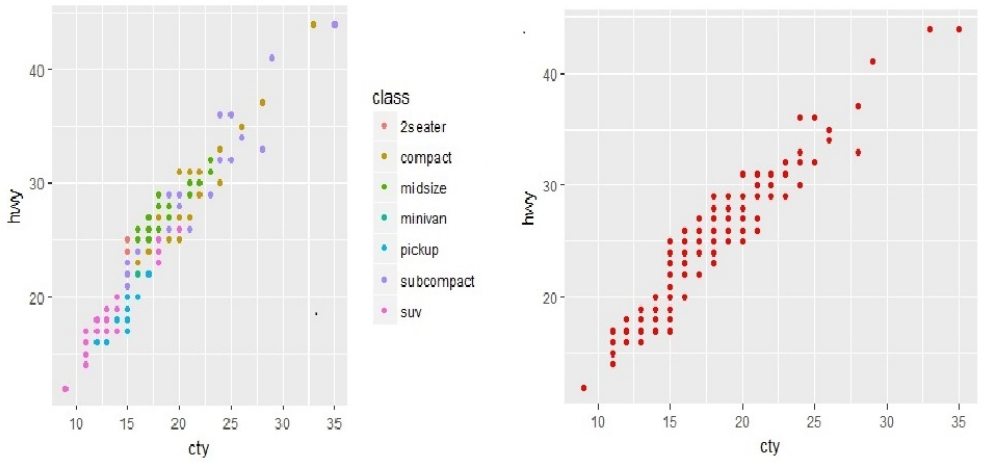

Figure 6. Visualizations of the `mpg` dataset – with `aes()` on the left, without on the right.

7. Multiple Graphs per Page

In basic R, the graphic parameter `mfrow` and the base function `layout()` are used to combine two or more base graphs into a single plot. This approach will not work with plots created with the *ggplot2* package, however. The easiest way to place multiple *ggplot2* graphs in a single figure is to use the `grid.arrange()` function found in the `gridExtra` package.

This code places three *ggplot2* charts based on the `Salaries` dataset onto a single graph.

```
data(Salaries, package="car")
library(ggplot2)
p1 <- ggplot(data=Salaries, aes(x=rank))
    + geom_bar()
p2 <- ggplot(data=Salaries, aes(x=sex))
    + geom_bar()
p3 <- ggplot(data=Salaries,
    aes(x=yrs.since.phd, y=salary)) +
    geom_point()

library(gridExtra)
grid.arrange(p1, p2, p3, ncol=3)
```

The resulting graph is shown in Figure 9. Each graph is saved as an object and then arranged into a single plot *via* `grid.arrange()`. Note the difference between faceting and multiple graphs: faceting creates an array of plots based on one or more categorical variables, but the components of a multiple graph could be completely independent plots arranged into a single display.

8. Themes

Themes allow the user to control the overall appearance of *ggplot2* charts; `theme()` options are used to change fonts, backgrounds, colours, gridlines, and more. Themes can be used once or saved and applied to multiple charts. See below for an example.

```
data(Salaries, package="car")
library(ggplot2)
mytheme <- theme(plot.title=element_text(
            face="bold.italic",
                size="14",
            color="brown"), axis.title=
            element_text(
                face="bold.italic",
               size=10, color="brown"),
            axis.text=element_text(
               face="bold", size=9,
               color="darkblue"),
            panel.background=element_rect(
               fill="white",color="darkblue"),
            panel.grid.major.y=element_line(
               color="grey", linetype=1),
            panel.grid.minor.y=element_line(
               color="grey", linetype=2),
            panel.grid.minor.x=element_blank(),
            legend.position="top")

ggplot(Salaries, aes(x=rank, y=salary,
    fill=sex)) +
      geom_boxplot() +
      labs(title="Salary by Rank and
          Sex", x="Rank", y="Salary") +
      mytheme
```

Adding "`+ mytheme`" to the plotting statement generates the graph shown in Figure 10; `mytheme` specifies that plot titles are printed in brown 14-point bold italics; axis titles in brown 10-point bold italics; axis labels in dark blue 9-point bold; the plot area should have a white fill and dark blue borders; major horizontal grids should be solid grey lines; minor horizontal grids should be dashed grey lines; vertical grids should be suppressed; and the legend should appear at the top of the graph. The `theme()` function gives you great control over the look of the finished product (consult `help(theme)` to learn more about these options).

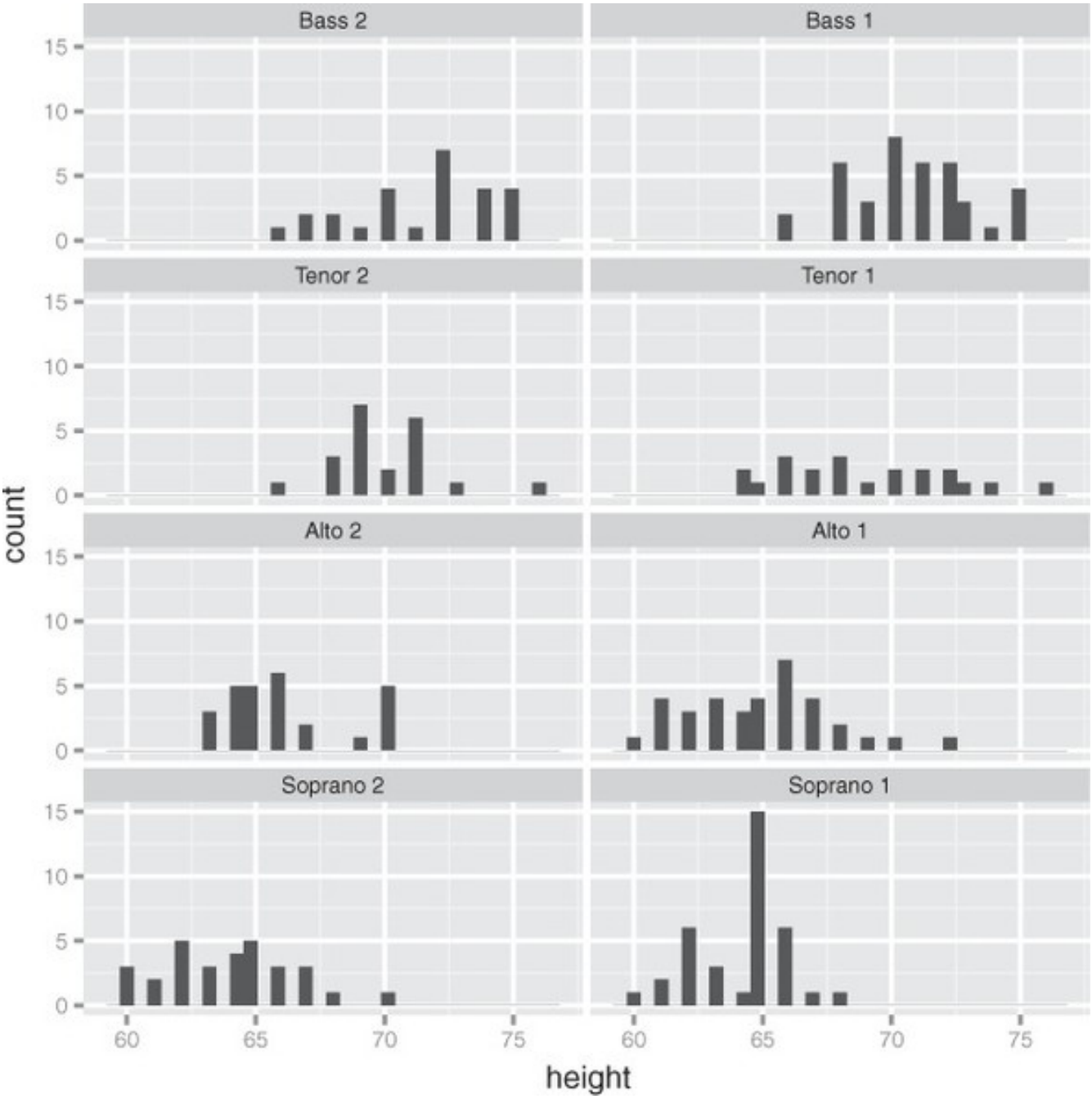

Figure 7. Faceted graph showing the distribution (histogram) of singer heights by voice part

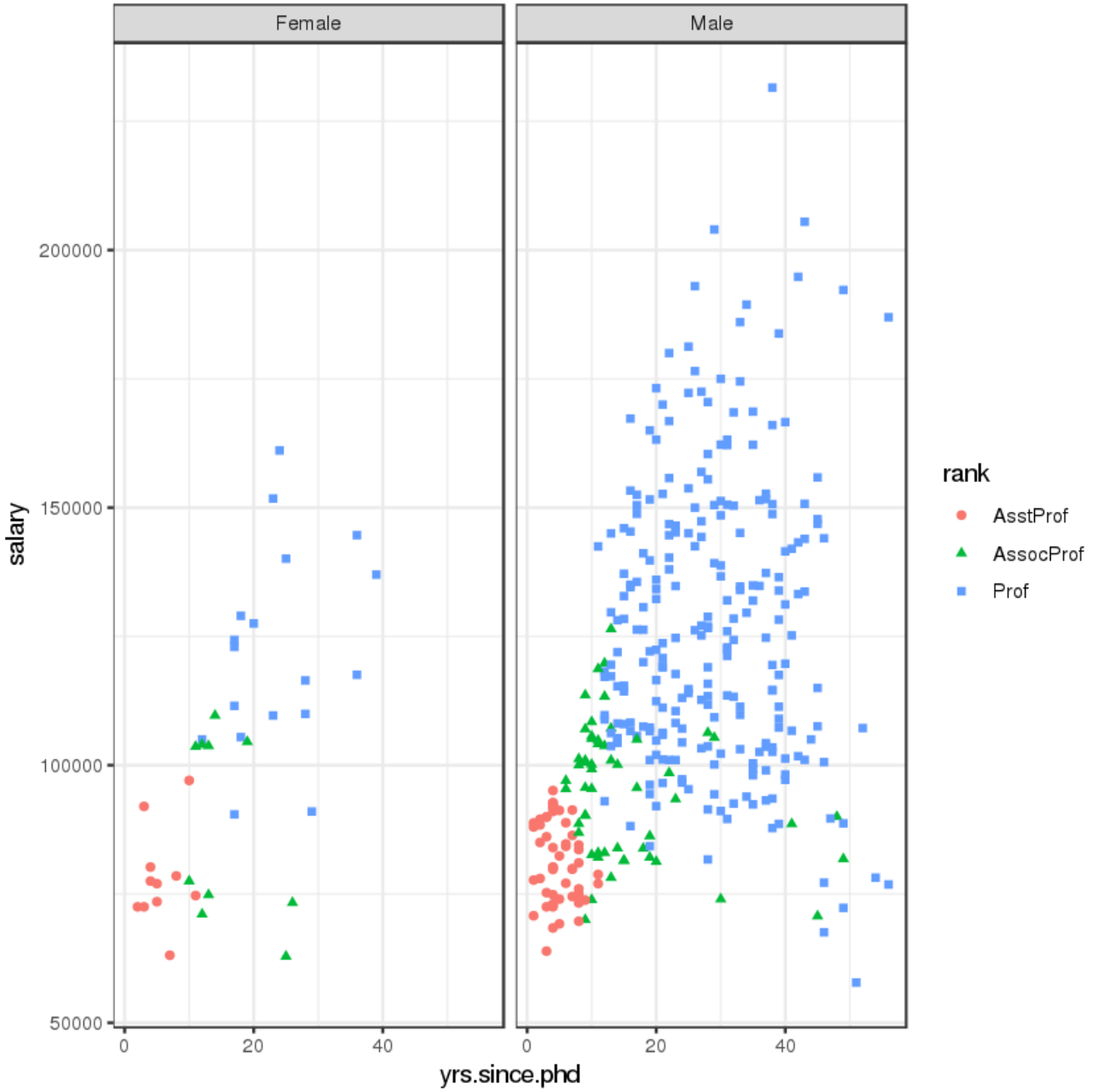

Figure 8. Scatterplot of years since graduation and salary. Academic rank is represented by color and shape, and sex is faceted.

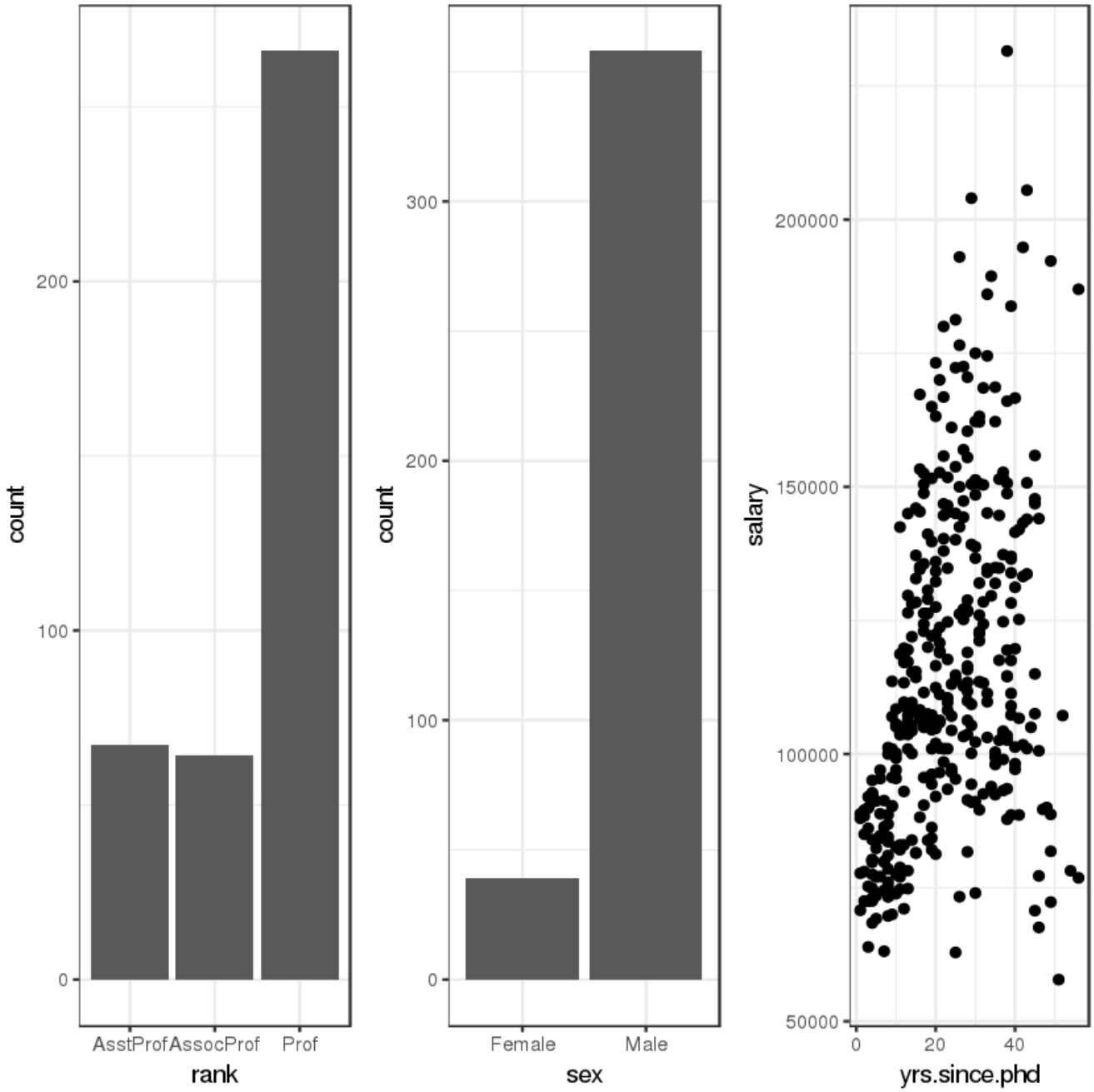

Figure 9. Placing three ggplot2 plots in a single graph

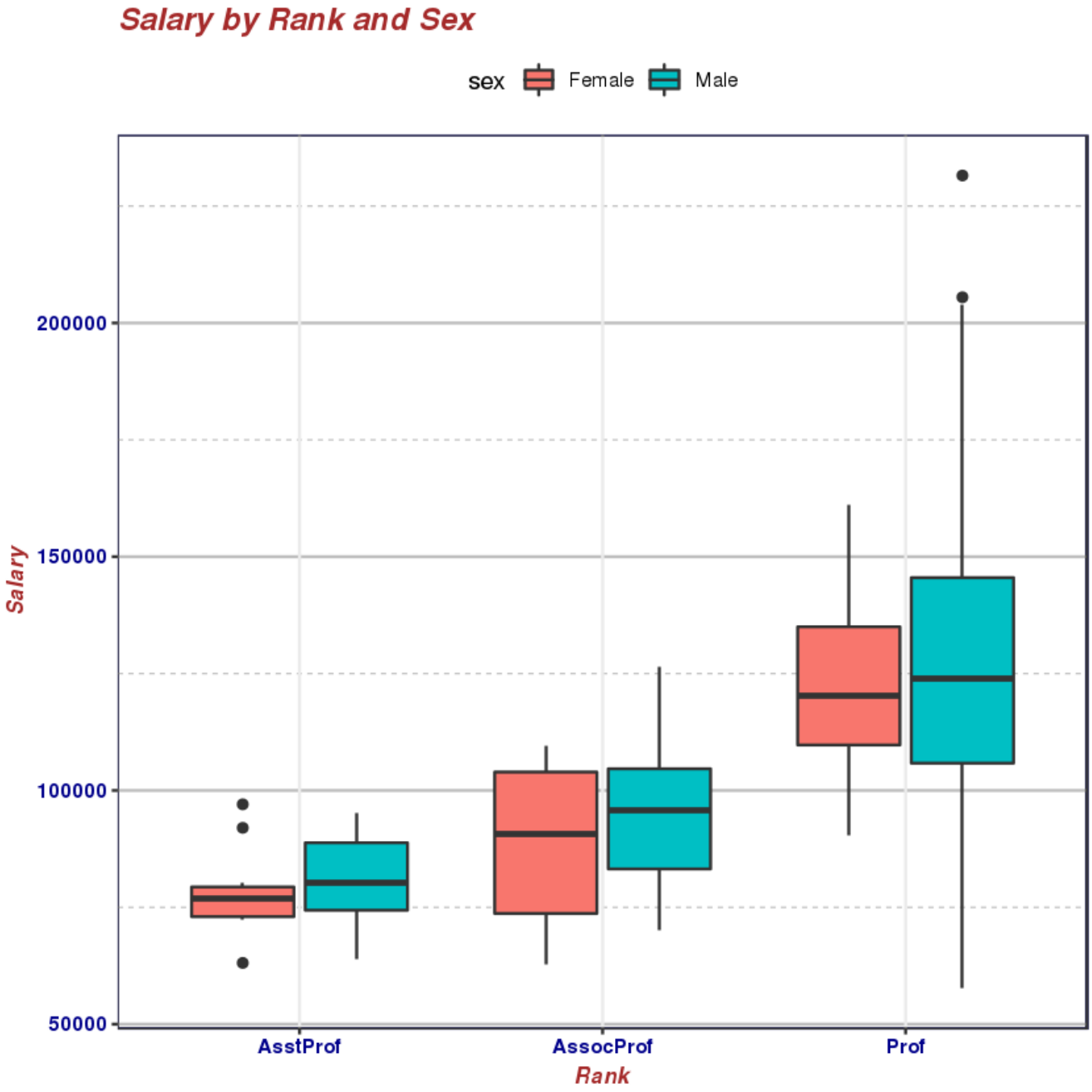

Figure 10. Box plots with a customized theme

9. Tidy Data: Getting Data into the Right Format

ggplot2 is compatible with what is generally referred to as the *tidyverse* [22]. Social scientists will likely be familiar with the distinction between data in *wide format* and in *long format*:

- in a long format table, every column represents a variables, and every row an observation,
- whereas in a wide format table, some variables are spread out across columns, perhaps along some other characteristic such as the year, say.

The plots that have been produced so far were simple to create because the data points were given in the format of one observation per row which we call a "tall" format. But many datasets come in a "wide"" format, i.e. there is more than one observation – more than one point on the scatterplot – in each row.

Consider, for instance, the `WorldPhones` dataset, one of R's built-in dataset:

```
data("WorldPhones")
```

This dataset records the number of telephones, in thousands, on each continent for several years in the 1950s (see Table 2).

Each column represents a different continent, and each row represents a different year. This wide format seems like a reasonable way to store data, but suppose that we want to compare increases in phone usage between continents, with time on the horizontal axis. In that case, each point on the plot is going to represent a continent during one year – there are seven observations in each row, which makes it very difficult to plot using *ggplot2*.

Fortunately, the *tidyvers* provides an easy way to convert this wide dataset into a tall dataset, by *melting* the data. This can be achieved by loading a thrid-party package called *reshape2*. The `WorldPhones` dataset can now be melted from a wide to a tall dataset with the `melt()` function. Let's assign the new, melted data to an object called `WorldPhones.m`, where the `m` reminds us that the data has been melted.

```
library(reshape2)
WorldPhones.m = melt(WorldPhones)
```

The new, melted data looks like:

```
head(WorldPhones.m)

##  Var1 Var2 value
## 1 1951 N.Amer 45939
## 2 1956 N.Amer 60423
## 3 1957 N.Amer 64721
## ...
```

Notice that while there were originally seven columns, there are now only three: `Var1`, `Var2`, and `value`; `Var1` represents the year, `Var2` the continents, and `value` the number of phones. Every data cell – every observation – every number of phones per year per continent – in the original dataset now has its own row in the melted dataset.

In 1951, in North America, for instance, there were 45,939,000 phones, which is the same value as in the original unmelted data – the data has not changed, it just got *reshaped*.

Changing the column names might make the data more intuitive to read:

```
colnames(WorldPhones.m) = c("Year",
    "Continent", "Phones")
head(WorldPhones.m)

##   Year Continent Phones
## 1 1951 N.Amer 45939
## 2 1956 N.Amer 60423
## 3 1957 N.Amer 64721
## ...
```

Now that the data has been melted into a tall dataset, it is easy to create a plot with *ggplot2*, with the usual steps of a `ggplot()` call, but with `WorldPhones.m` instead of `WorldPhones`:

```
ggplot(WorldPhones.m, aes(x=Year,
    y=Phones, color=Continent)) +
    geom_point()
```

We place the `Year` on the x—axis, in order to see how the numbers change over time, while the number of `Phones` (the variable of interest) is displayed on the y—axis. The `Continent` factor will be represented with colour. A scatterplot is obtained by adding a `geom_point()` layer.

Scatterplots can also be used to show trends over time, by drawing lines between points for each continent. This only require a change to a `geom_line()` layer.

```
ggplot(WorldPhones.m, aes(x=Year,
    y=Phones, color=Continent)) +
    geom_line()
```

The result is shown in Figure 11. Incidentally, one might expect the number of phones to increase exponentially over time, rather than linearly (a fair number of observations are clustered at the bottom of the chart).

When that's the case, it's a good idea to plot the vertical axis on a log scale. This can be done adding a logarithm scale to the chart.

```
ggplot(WorldPhones.m, aes(x=Year,
    y=Phones, color=Continent)) +
    geom_line() + scale_y_log10()
```

	N.Amer	Europe	Asia	S.Amer	Oceania	Africa	Mid.Amer
1951	45939	21574	2876	1815	1646	89	555
1956	60423	29990	4708	2568	2366	1411	733
1957	64721	32510	5230	2695	2526	1546	773
1958	68484	35218	6662	2845	2691	1663	836
1959	71799	37598	6856	3000	2868	1769	911
1960	76036	40341	8220	3145	3054	1905	1008
1961	79831	43173	9053	3338	3224	2005	1076

Table 2. `WorldPhones` dataset in wide format.

Now each of the phone trends looks linear, and the lower values are spotted more easily; for example, it is now clear that Africa has overtaken Central America by 1956 (see Figure 12).

Notice how easy it was to build this plot once the data was in the tall format: one row for every point – that's every combination of year and continent – on the graph.

10. Saving Graphs

Plots might look great on the screen, but they typically have to be embedded in other documents (Markdown, LaTeX, Word, etc.). In order to do so, they must first be saved in an appropriate format, with a specific resolution and size. Default size settings can be saved within the `.Rmd` document by declaring them in the first chunk of code. For instance, this would tell `knitr` to produce 8 in. × 5 in. charts:

```
knitr::opts_chunk$set(fig.width=8,
    fig.height=5)
```

A convenience function named `ggsave()` can be particularly useful. Options include which plot to save, where to save it, and in what format. For example,

```
myplot <- ggplot(data=mtcars,
    aes(x=mpg)) + geom_histogram()
ggsave(file="mygraph.png", plot=myplot,
    width=5, height=4)
```

saves the `myplot` object as a 5-inch by 4-inch `.png` file named `mygraph.png` in the current working directory. The available formats include `.ps`, `.tex`, `.jpeg`, `.pdf`, `.jpg`, `.tiff`, `.png`, `.bmp`, `.svg`, or `.wmf` (the latter only being available on Windows machines).

Without the `plot=` option, the most recently created graph is saved. The following code, for instance, the following bit of code would also save the `mtcars` plot (the latest plot) to the current working directory (see the `ggsave()` helf file for additional details):

```
ggplot(data=mtcars, aes(x=mpg)) +
    geom_histogram()
ggsave(file="mygraph.pdf")
```

Within RStudio, an alternative is to click on *Export*, then "Save Plot As Image" to open a GUI.

11. Summary

The first 10 sections reviewed the *ggplot2* package, which provides advanced graphical methods based on a comprehensive grammar of graphics. The package is designed to provide the use with a complete and comprehensive alternative to the native graphics provided with R. It offers methods for creating attractive and meaningful data visualizations that are difficult to generate in other ways.

It does come with some drawbacks, however: the *ggplot2* and *tidyverse* design teams have fairly strong opinions about how data should be visualized and processed. As a result, it can sometimes be difficult to produce charts that go against their design ideals. In the same vein, the various package updates do not always preserve the functionality of working code, sending the analysts scurrying to figure how the new functions work, which can cause problems with legacy code. Still, the versatility and overall simplicity of *ggplot2* cannot be overstated.

A list of all *ggplot2* functions, along with examples, can be found at **http://docs.ggplot2.org**. The theory underlying *ggplot2* is explained in great deatil in [2]; useful examples and starting points can also be found in [1, 5].

The *ggplot2* action flow is always the same: start with data in a table, map the display variables to various *aesthetics* (position, colour, shape, etc.), and select one or more *geoms* to draw the graph. This is accomplished in the code by first creating an object with the basic data and mappings information, and then by adding or layering additional information as needed.

Once this general way of thinking about plots is understood (especially the aesthetic mapping part), the drawing process is simplified significantly. There is no need to think about how to draw particular shapes or colours in the chart; the many (self-explanatory) `geom_` functions do all the heavy lifting.

Similarly, learning how to use new `geoms` is easier when they are viewed as ways to display specific aesthetic mappings.

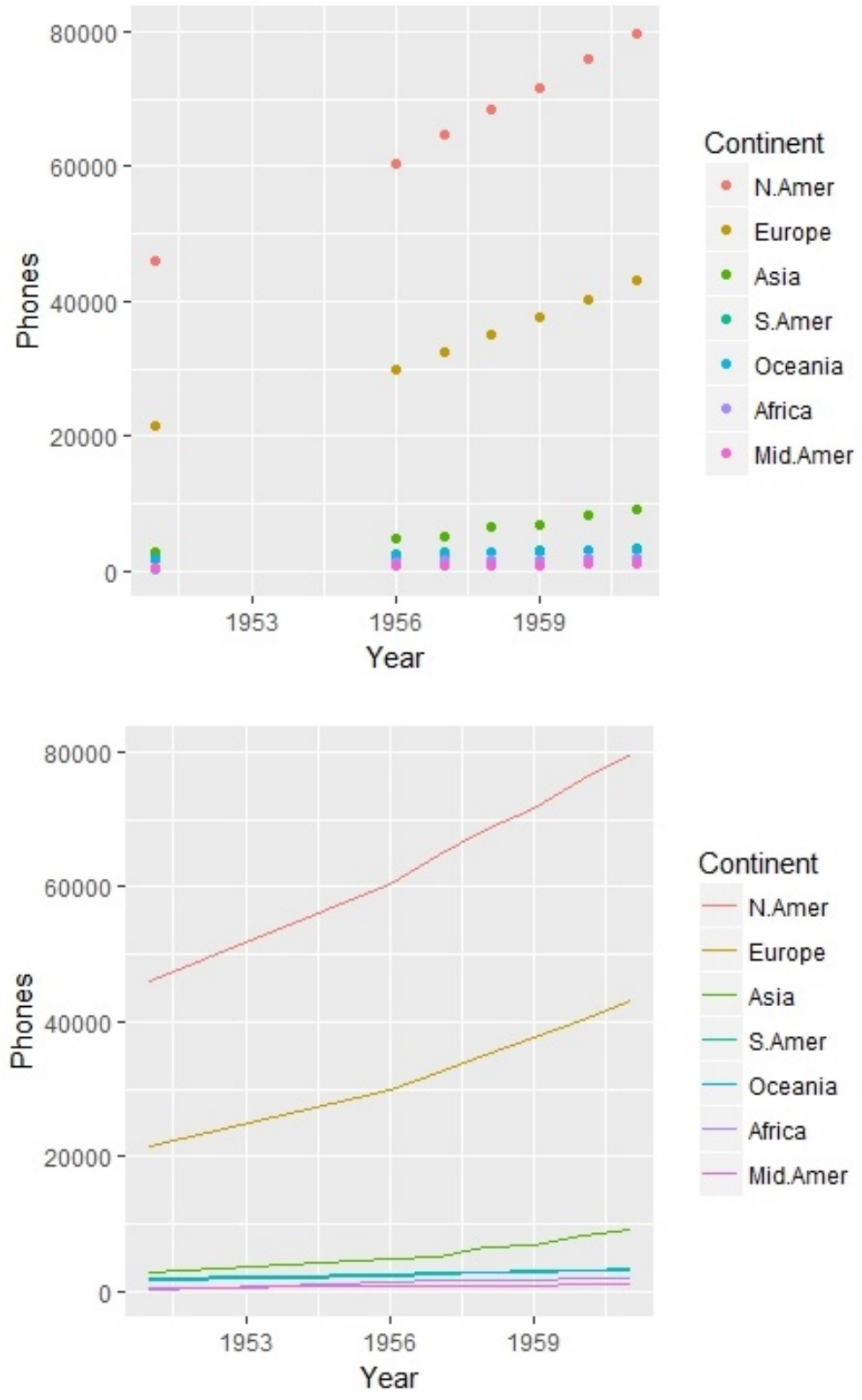

Figure 11. `WorldPhones.m` plots, using `geom_points()` (above) and `geom_lines()` (below).

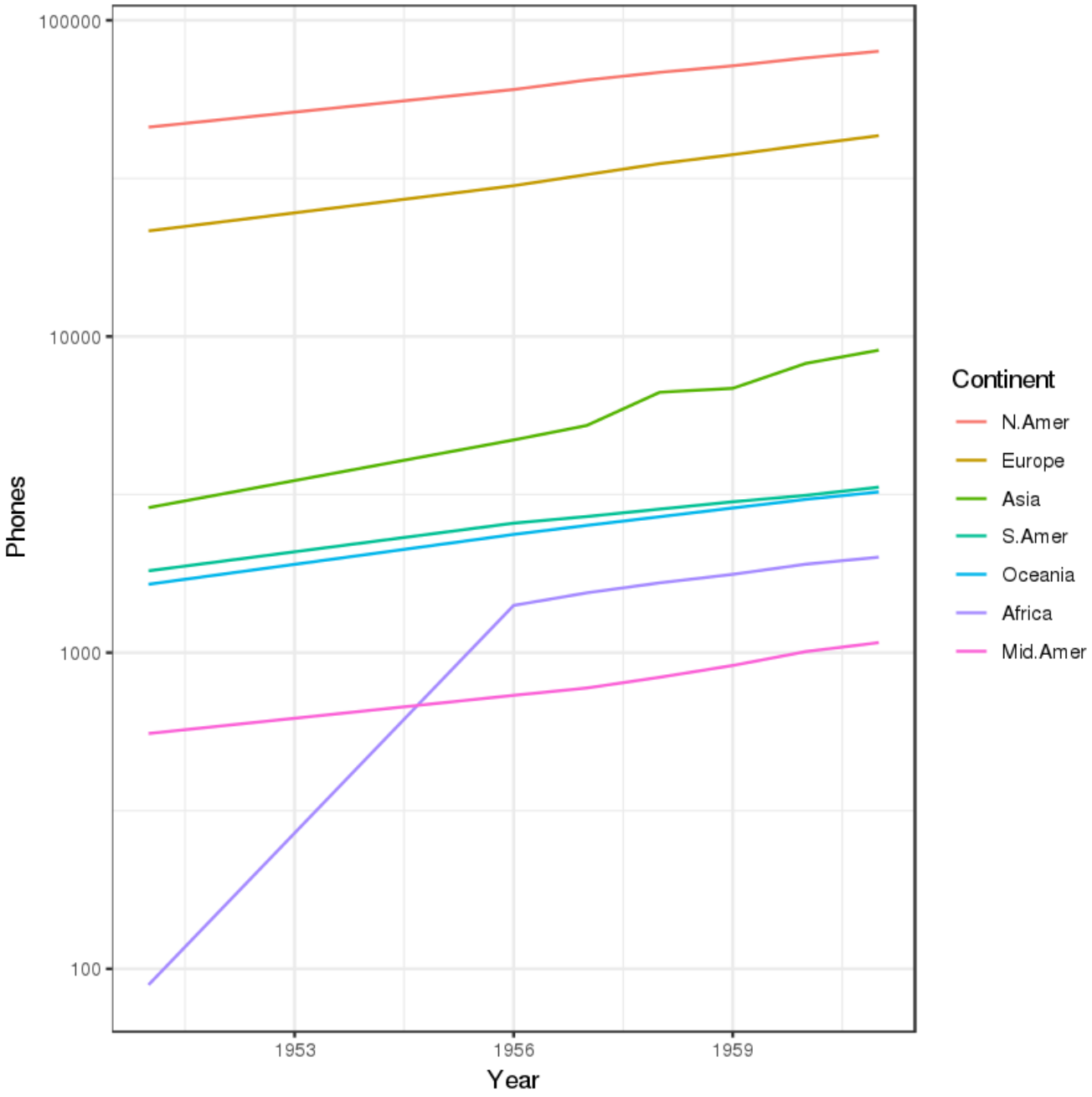

Figure 12. `WorldPhones.m` on a vertical logarithmic scale.

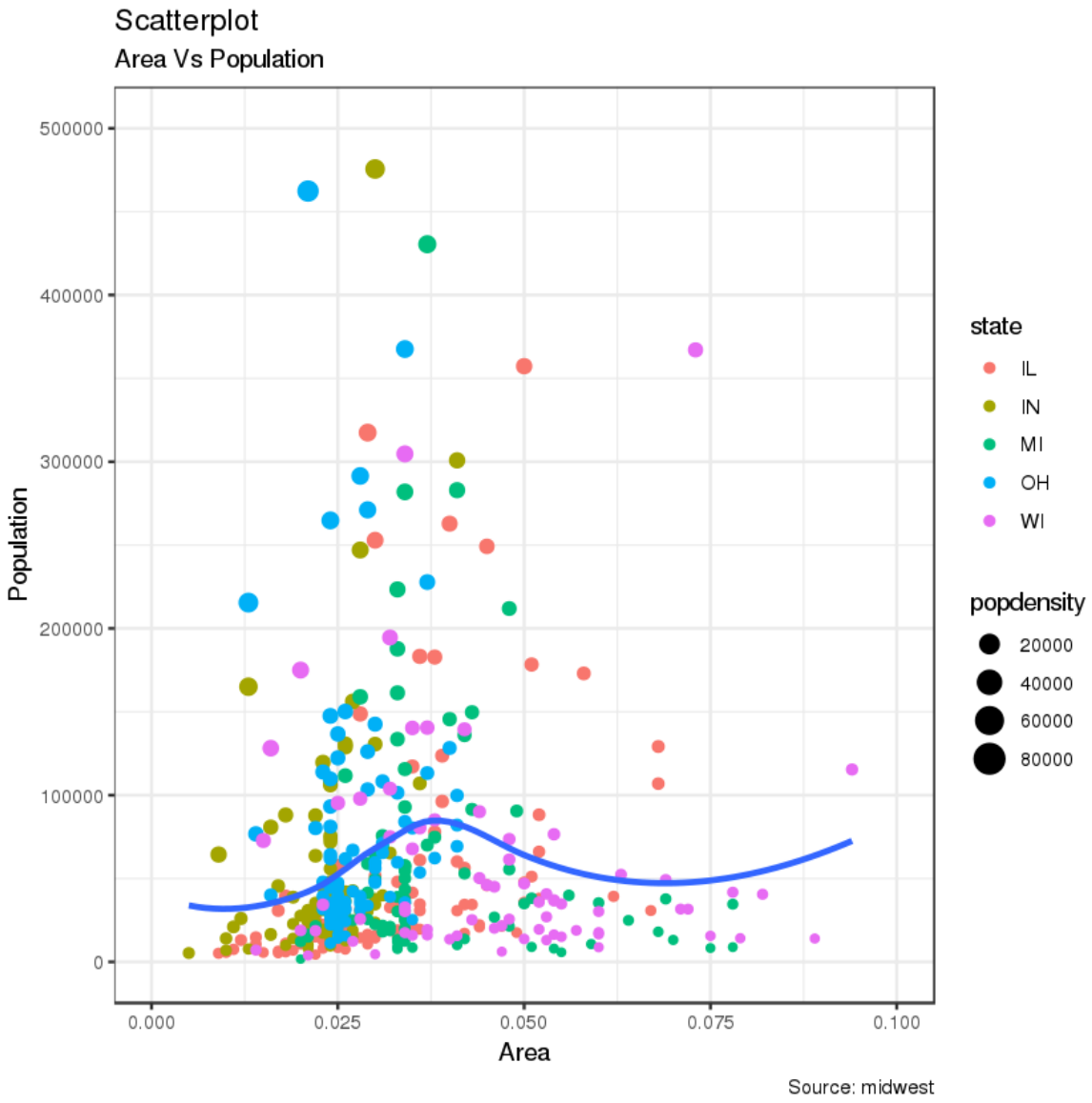

Figure 13. Scatterplot of the `midwest` dataset.

12. Examples

In this final section, we provide 30 additional examples of *ggplot2* visualizations. Some of those examples have been taken directly (and modified) from various online sources (see references).

Example 1 (Scatterplot)

The most frequently used plot for data analysis is undoubtedly the scatterplot. Whenever you want to understand the nature of relationship between two variables, invariably the first choice is the scatterplot, which is drawn using `geom_point()`. Additionally, the `geom_smooth` default will draw a "loess" smoothing line, which can be tweaked to draw the line of best fit instead by setting `method='lm'` (see Figure 13).

```
# load package and data
options(scipen=999) # turn-off
    scientific notation like 1e+48
library(ggplot2)
theme_set(theme_bw()) # pre-set
data("midwest", package = "ggplot2")
# Scatterplot
ggplot(midwest, aes(x=area, y=poptotal))
    + geom_point(aes(col=state,
    size=popdensity)) +
  geom_smooth(method="loess", se=F) +
  xlim(c(0, 0.1)) + ylim(c(0, 500000)) +
  labs(subtitle="Area Vs Population",
      y="Population",
      x="Area",
      title="Scatterplot",
      caption = "Source: midwest")
```

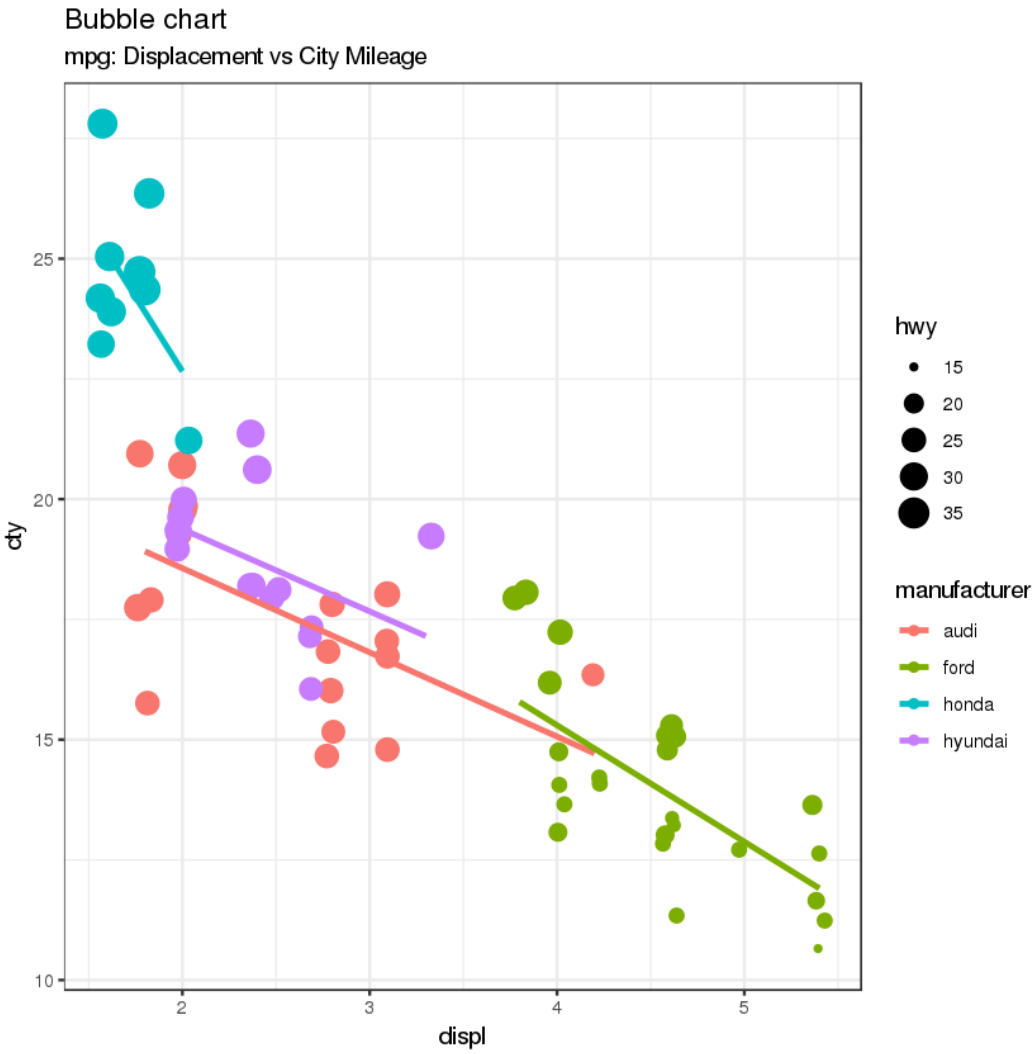

Figure 14. Bubble chart of the `mpg` dataset.

Example 2 (Bubble Chart)

While scatterplots allows for comparisons between 2 continuous variables, bubble charts extend the principles to 4 or more variables using various marker elements:

- the colour of the marker can be mapped to a categorical variable (finite colour choices) or a continuous variable (gradient scale);
- the size of the marker is typically mapped to a positive continuous variable.

Additionally, the shape of the marker can be mapped to a categorical variable.

With the `mpg` dataset, a bubble chart can help to clearly distinguish the range of the `displ` feature for the various manufacturers, and can be used to show how the slope of the lines of best fit vary by manufacturer, providing a better visual comparison between the groups (see Figure 14).

```
library(ggplot2)
data(mpg, package="ggplot2")
mpg_select <- mpg[mpg$manufacturer %in%
    c("audi", "ford", "honda",
    "hyundai"), ]

# Scatterplot
theme_set(theme_bw()) # pre-set the bw
    theme.
g <- ggplot(mpg_select, aes(displ, cty))
    +
  labs(subtitle="mpg: Displacement vs
      City Mileage",
       title="Bubble chart")
g + geom_jitter(aes(col=manufacturer,
    size=hwy)) +
  geom_smooth(aes(col=manufacturer),
      method="lm", se=F)
```

Example 3 (Animated Bubble Chart)

An animated bubble chart can be implemented using the *gganimate* package. It works quite the same way as a bubble chart, but allows the user to show how the bubble chart changes with an additional variable (typically time). The key element is to set `aes(frame)` to the desired column on which to animate. The rest of the plot construction procedure is the same as before. Once the plot is constructed, it can be animated by using `gganimate()` and setting the "time" variable appropriately.

Note that *ImageMagick* (**http://imagemagick.org**) must be installed in order to use the `anim_save()` function.

```
library(ggplot2)
library(gganimate)
library(gapminder)
theme_set(theme_bw()) # pre-set bw theme
head(gapminder)

ggplot(gapminder, aes(gdppc,
    life_expectancy, size = population,
    colour = country)) +
  geom_point(alpha = 0.7, show.legend =
     FALSE) +
  #scale_colour_manual(values =
     country_colors) +
  scale_size(range = c(2, 12)) +
  scale_x_log10() +
  facet_wrap(~continent) +
  # Here is the gganimate bits
  labs(title = 'Year: {frame_time}', x =
     'GDP per capita', y = 'life
     expectancy') +
  transition_time(year) +
  ease_aes('linear')
anim_save(file="gapminder.gif") # saved,
    not plotted
```

Example 4 (Maps)

The *ggmap* package provides facilities to interact with the google maps api and get the coordinates (latitude and longitude) of places you want to plot. The example below provides road (Figure 16), hybrid (Figure 17) and satellite (Figure 18) maps of the city of Ottawa, encircling some locations of interest (the google maps api has changed its functionality since the images were originally created; the example below will require some tweaking). The `geocode()` function is used to get the coordinates of the locations and `qmap()` is used to retrieve the maps. The type of map to fetch is determined by the value set to `maptype`.

The map supports zooms; the default value of 10 is suitable for large cities. It can be reduced to 3 for zooming out, and increased to 21 to zoom in at the building level.

```
library(ggplot2)
library(ggmap)
library(ggalt)

# Get Ottawa's Coordinates
    -------------------------------
ottawa<-geocode("Ottawa") # get
    longitude and latitude

# Get Coordinates for Ottawa's Places
    --------------------
ottawa_places<-c("Canadian War
    Museum","Rideau Centre","University
    of Ottawa","Carleton University")
places_loc <- geocode(ottawa_places) #
    get longitudes and latitudes

# Get the Map
    ----------------------------
# Google Satellite Map
ottawa_ggl_sat_map <- qmap("ottawa",
    zoom=13, source = "google",
    maptype="satellite")

# Google Hybrid Map
ottawa_ggl_hybrid_map <- qmap("ottawa",
    zoom=13, source = "google",
    maptype="hybrid")

# Google Road Map
ottawa_ggl_road_map <- qmap("ottawa",
    zoom=13, source = "google",
    maptype="roadmap")

# Plot Google Road Map
    ------------------------------------
ottawa_ggl_road_map +
    geom_point(aes(x=lon, y=lat), data =
    places_loc, alpha = 0.8, size = 7,
    color = "tomato") +
    geom_encircle(aes(x=lon, y=lat),
    data = places_loc, size = 2, color =
    "blue")

# Google Hybrid Map
    ---------------------------------------
ottawa_ggl_hybrid_map +
    geom_point(aes(x=lon, y=lat), data =
    places_loc, alpha = 0.7, size = 7,
    color = "tomato") +
    geom_encircle(aes(x=lon, y=lat),
    data = places_loc, size = 2, color =
    "blue")

# Google Satellite Map
    ---------------------------------------
ottawa_ggl_sat_map +
    geom_point(aes(x=lon, y=lat), data =
    places_loc, alpha = 0.7, size = 7,
    color = "tomato") +
  geom_encircle(aes(x=lon, y=lat), data
     = places_loc, size = 2, color =
     "blue")
```

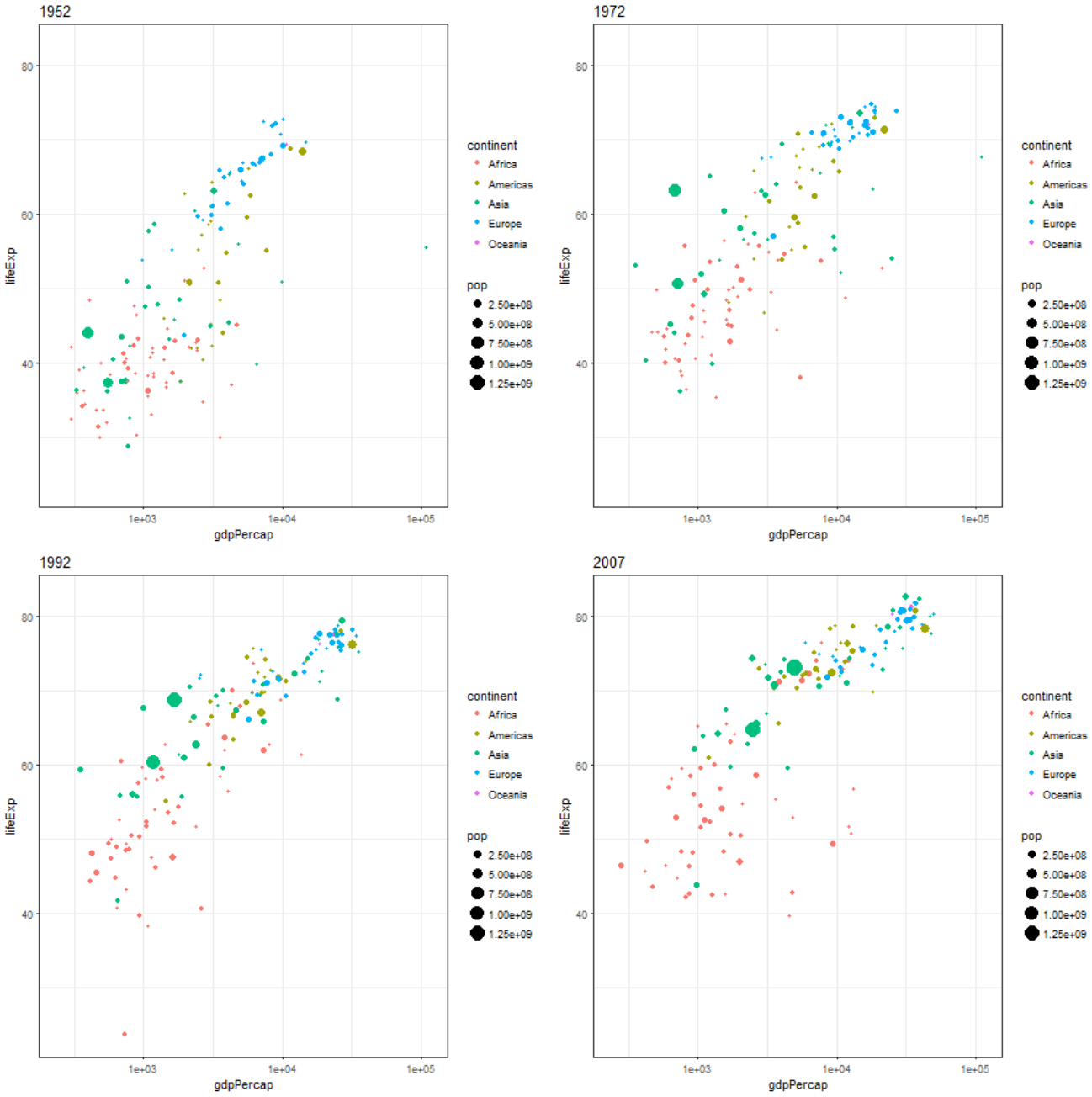

Figure 15. Bubble chart of the `gapminder` dataset (selected frames).

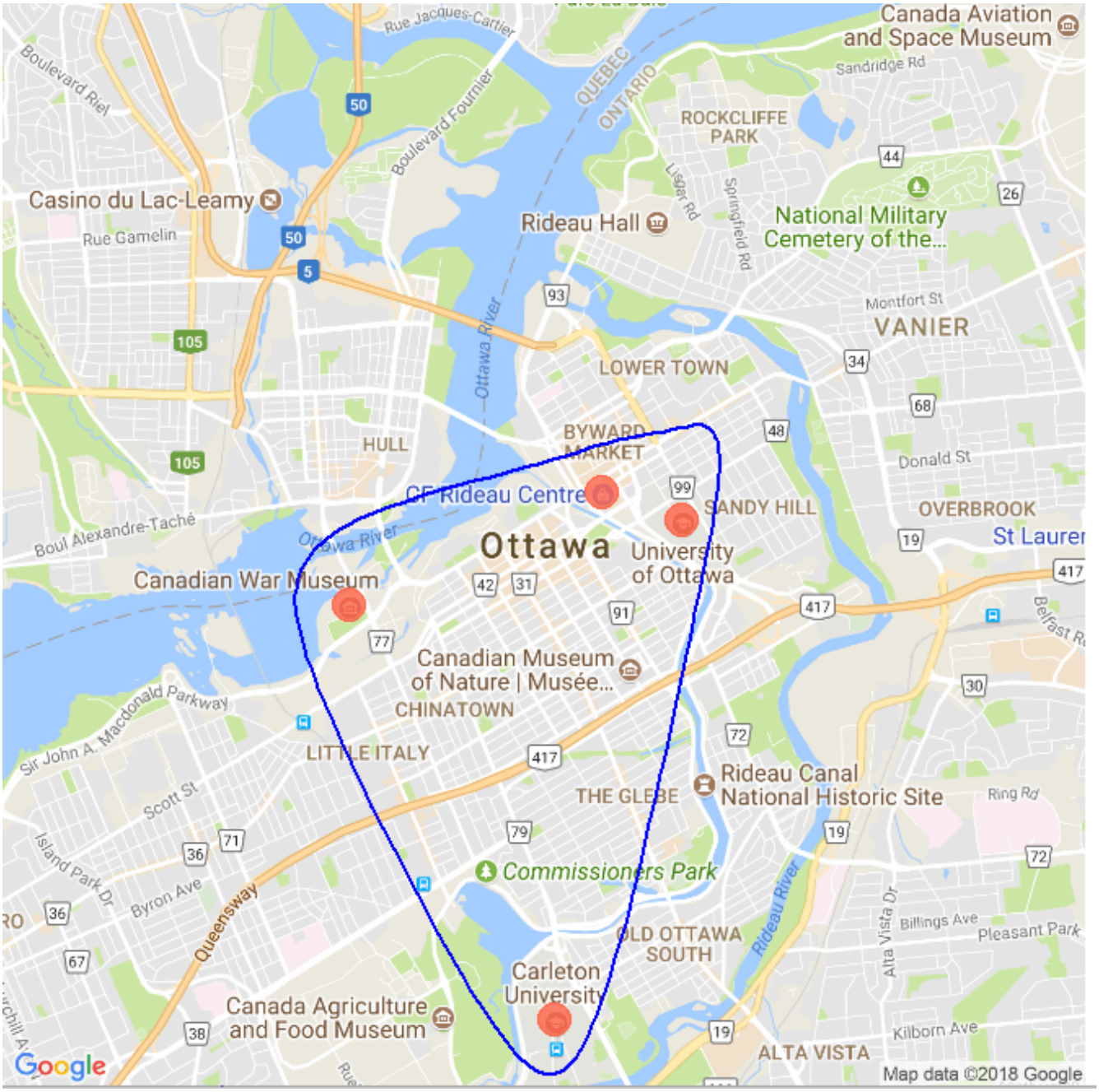

Figure 16. Ottawa Road Map

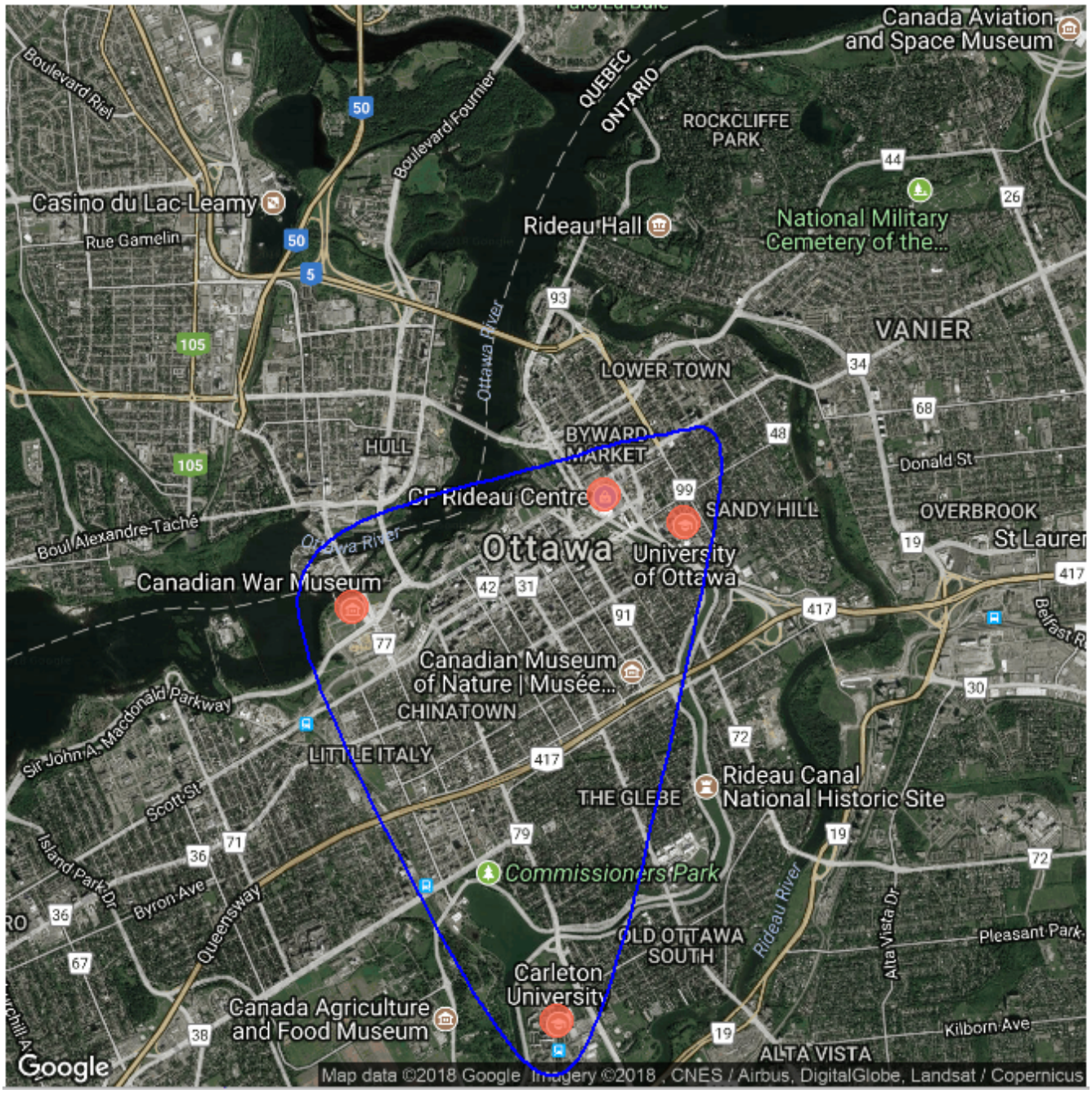

Figure 17. Ottawa Hybrid Map

Figure 18. Ottawa Satellite Map

Example 5 (Marginal Histogram / Boxplot)
A marginal histogram is a chart on which the scatterplot of two variables X and Y are shown, together with the distribution of each of the variables. This can be implemented using the `ggMarginal()` function from the *ggExtra* package. Other marginal plots are available. The output of the following code is shown in Figure 19.

```
# load package and data
library(ggplot2)
library(ggExtra)
data(mpg, package="ggplot2")

# Scatterplot
theme_set(theme_bw()) # pre-set the bw
    theme.
mpg_select <- mpg[mpg$hwy >= 35 &
    mpg$cty > 27, ]
g <- ggplot(mpg, aes(cty, hwy)) +
  geom_count() +
  geom_smooth(method="lm", se=F)

ggMarginal(g, type = "histogram",
    fill="transparent")
ggMarginal(g, type = "boxplot",
    fill="transparent")
```

Example 6 (Diverging Bars)
We might want bar charts that can handle both negative and positive values (diverging bars); this can be implemented by providing a tweak to `geom_bar()` (the histogram function):

- set `stat=identity`
- provide both `x` and `y` inside the `aes()` call, where `x` is a character or a factor and `y` is numeric.

In order to guarantee diverging bars (instead of simple bars), the categorical variable must have two levels whose values change at a given threshold of the continuous variable. In the display of Figure 20, `mpg` (from `mtcars`) is normalised by computing the z score – vehicles with `mpg` above zero are shown in green; those below in red.

```
library(ggplot2)
theme_set(theme_bw())
data("mtcars") # load data
mtcars$`car name` <- rownames(mtcars) #
    create new column for car names
mtcars$mpg_z <- round((mtcars$mpg -
    mean(mtcars$mpg))/sd(mtcars$mpg), 2)
    # compute normalized mpg
mtcars$mpg_type <- ifelse(mtcars$mpg_z <
    0, "below", "above") # above / below
    avg flag
mtcars <- mtcars[order(mtcars$mpg_z), ]
    # sort
mtcars$`car name` <- factor(mtcars$`car
    name`, levels = mtcars$`car name`)
# convert to factor to retain sorted
    order in plot.

# Diverging Barcharts
ggplot(mtcars, aes(x=`car name`,
    y=mpg_z, label=mpg_z)) +
  geom_bar(stat='identity',
      aes(fill=mpg_type), width=.5) +
  scale_fill_manual(name="Mileage",
    labels = c("Above Average", "Below
        Average"),
    values = c("above"="#00ba38",
        "below"="#f8766d")) +
  labs(subtitle="Normalised mileage",
      title= "Diverging Bars - mtcars") +
  coord_flip()
```

Example 7 (Area Chart)
Area charts are typically used to visualize how a particular metric (such as % returns from a stock) performed compared to a certain baseline. Other types of % returns or % change data are commonly used; area charts are implemented with `geom_area()` (see Figure 21).

```
library(ggplot2)
#install.packages("quantmod")
library(quantmod)
data("economics", package = "ggplot2")

# Compute % Returns
economics$returns_perc <- c(0,
    diff(economics$psavert)/
    economics$psavert[
    -length(economics$psavert)])

# Create break points and labels for
    axis ticks
brks <- economics$date[seq(1,
    length(economics$date), 12)]
#install.packages("lubridate")
lbls <-
    lubridate::year(economics$date[seq(1,
    length(economics$date), 12)])

# Plot
ggplot(economics[1:100, ], aes(date,
    returns_perc)) +
  geom_area() +
  scale_x_date(breaks=brks, labels=lbls)+
  theme(axis.text.x =
      element_text(angle=90)) +
  labs(title="Area Chart",
      subtitle = "Perc Returns for
          Personal Savings",
      y="% Returns for Personal Savings",
      caption="Source: economics")
```

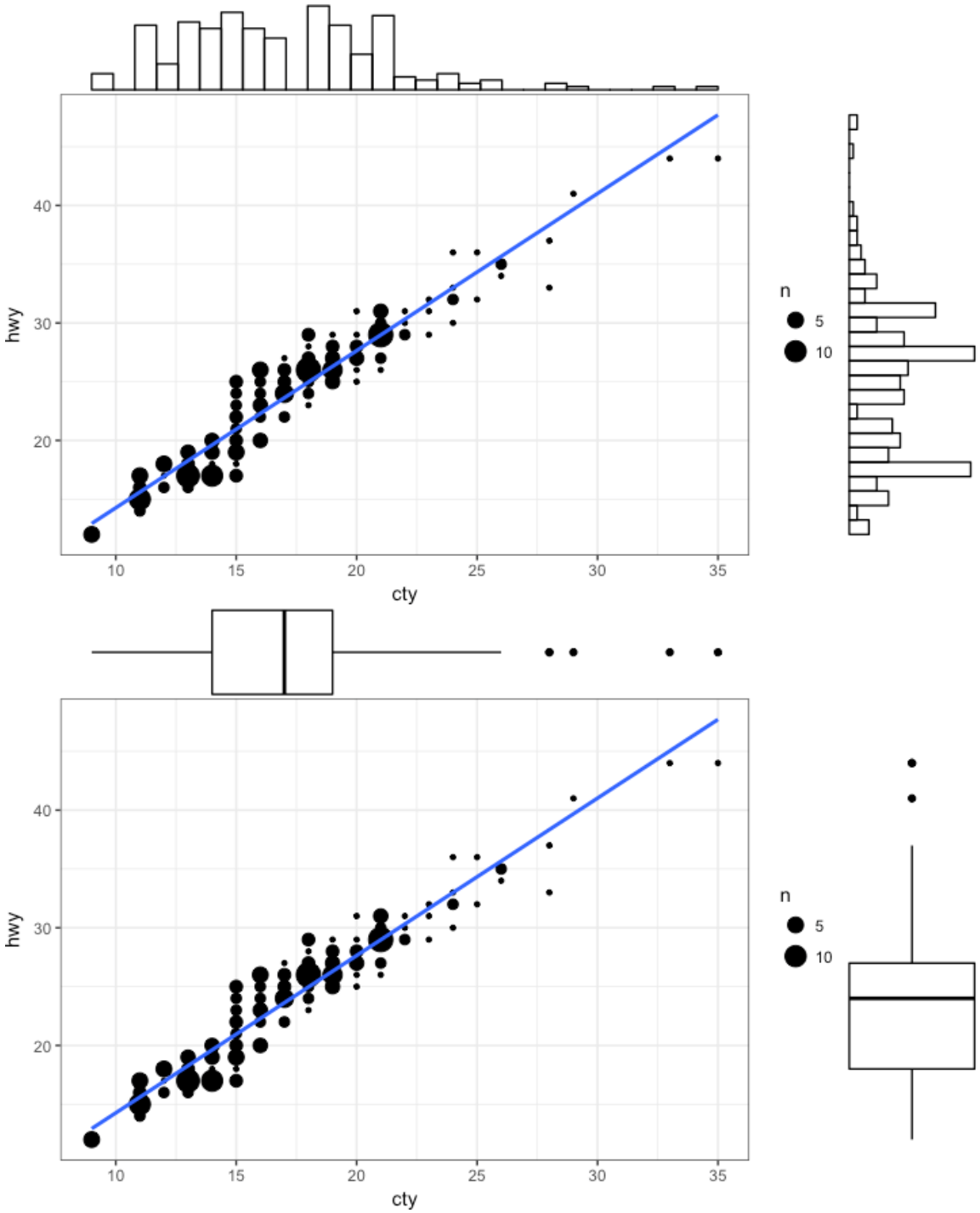

Figure 19. Marginal histograms / boxplots for the `mtcats` dataset.

Example 8 (Population Pyramid)

Population pyramids offer a way to visualize how much of the population (or what percentage of the population) falls under a certain category. The pyramid of Figure 22 is an excellent example, showing how many users are retained at each stage of an email marketing campaign funnel.

```
library(ggplot2)
library(ggthemes)
options(scipen = 999) # turns off
    scientific notations (like 1e+40)

# Read data
email_campaign_funnel <-
read.csv("https://raw.githubusercontent.com/
selva86/datasets/master/
```

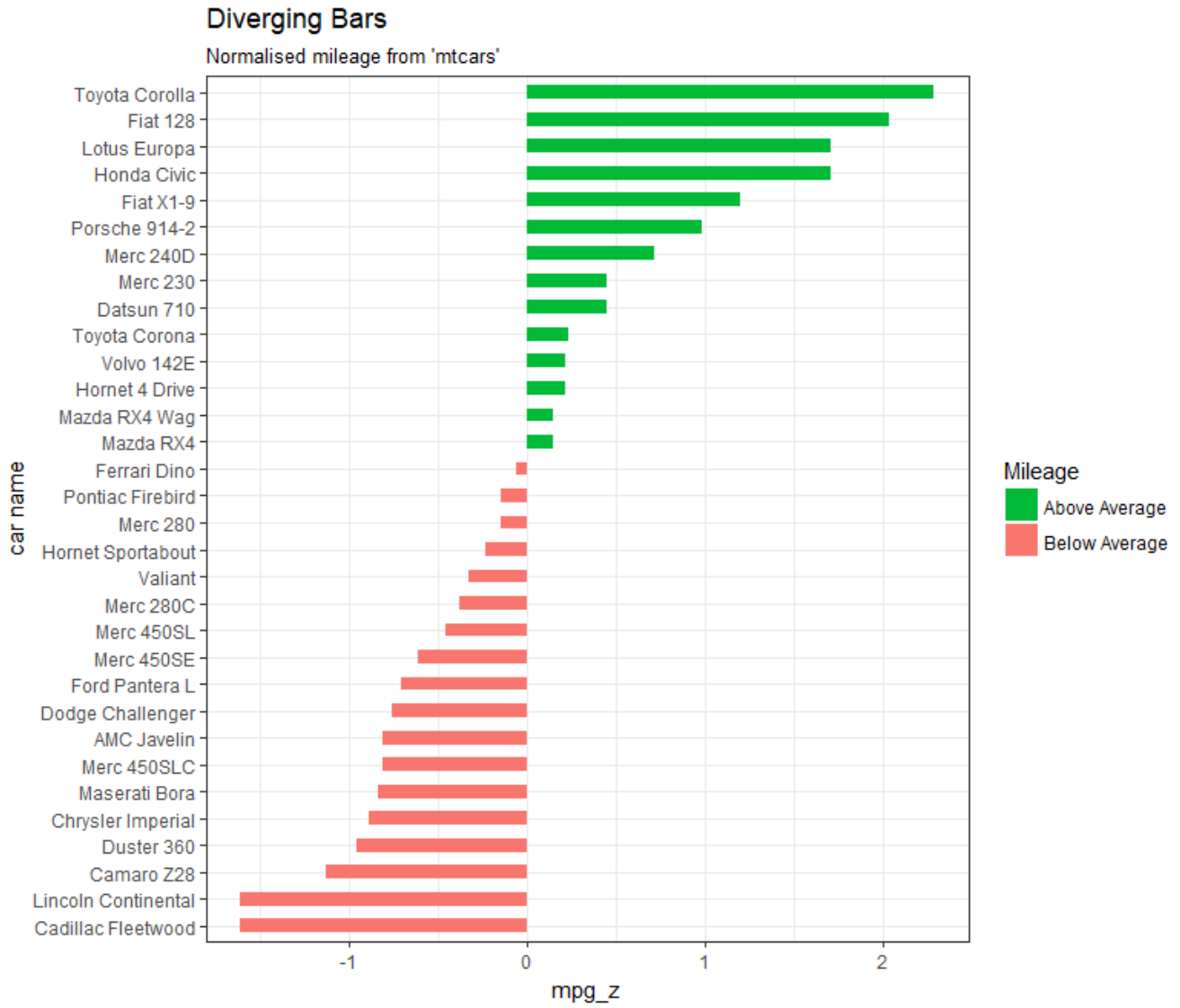

Figure 20. Diverging bars for the `mtcars` dataset.

```
email_campaign_funnel.csv")
# X Axis Breaks and Labels
brks <- seq(-15000000, 15000000, 5000000)
lbls = paste0(as.character(c(seq(15, 0,
    -5), seq(5, 15, 5))), "m")

# Plot
ggplot(email_campaign_funnel, aes(x =
    Stage, y = Users, fill = Gender)) +
    # Fill column
geom_bar(stat = "identity", width = .6)
    + # draw the bars
scale_y_continuous(breaks = brks, #
    Breaks
    labels = lbls) + # Labels
coord_flip() + # Flip axes
labs(title="Email Campaign Funnel") +
theme_tufte() + # Tufte theme from
    ggfortify
theme(plot.title = element_text(hjust =
    .5), axis.ticks = element_blank()) +
    # Centre plot title
scale_fill_brewer(palette = "Dark2")
# Colour palette
```

Example 9 (Calendar Heatmap)

The calendar heat map is a great tool to see the daily variation (especially the highs and lows) of a variable like stock price, as it emphasizes the variation over time rather than the actual value itself. It can (with a fair amount of data preparation) be produced with `geom_tile`.

```
# http://margintale.blogspot.in/2012/04/
    ggplot2-time-series-heatmaps.html
```

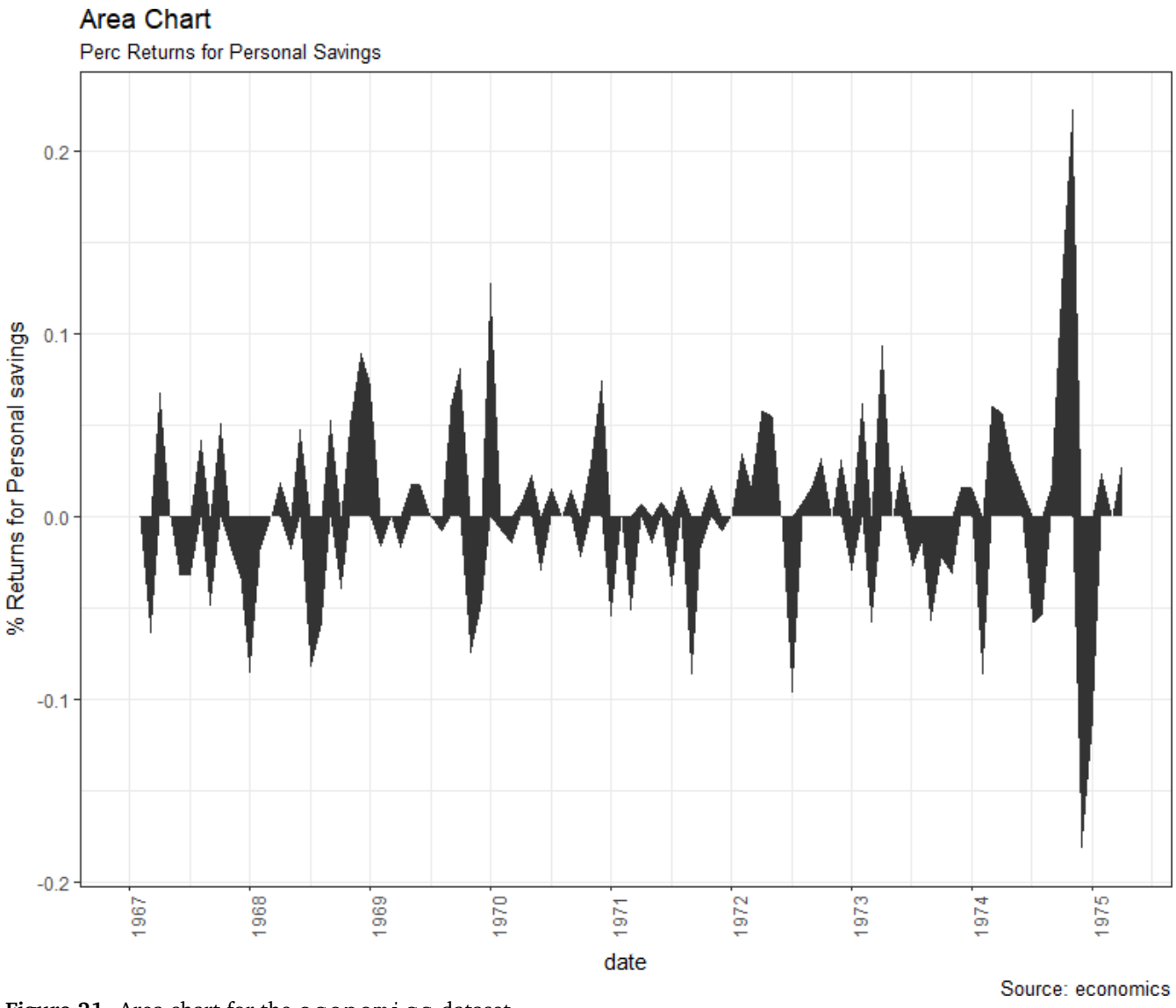

Figure 21. Area chart for the `economics` dataset.

```
library(ggplot2)
library(plyr)
library(scales)
library(zoo)

df <- read.csv(
    "https://raw.githubusercontent.com/
    selva86/datasets/master/yahoo.csv")
df$date <- as.Date(df$date) # format date
df <- df[df$year >= 2012, ] # filter
    years

# Create Month Week
df$yearmonth <- as.yearmon(df$date)
df$yearmonthf <- factor(df$yearmonth)
df <- ddply(df,.(yearmonthf), transform,
    monthweek=1+week-min(week))
# compute week number of month
df <- df[, c("year", "yearmonthf",
    "monthf", "week", "monthweek",
    "weekdayf", "VIX.Close")]
head(df)

# Plot
ggplot(df, aes(monthweek, weekdayf, fill
    = VIX.Close)) +
  geom_tile(colour = "white") +
  facet_grid(year~monthf) +
  scale_fill_gradient(low="red",
      high="green") +
  labs(x="Week of Month",
       y="",
       title = "Time-Series Calendar
          Heatmap",
       subtitle="Yahoo Closing Price",
       fill="Close")
```

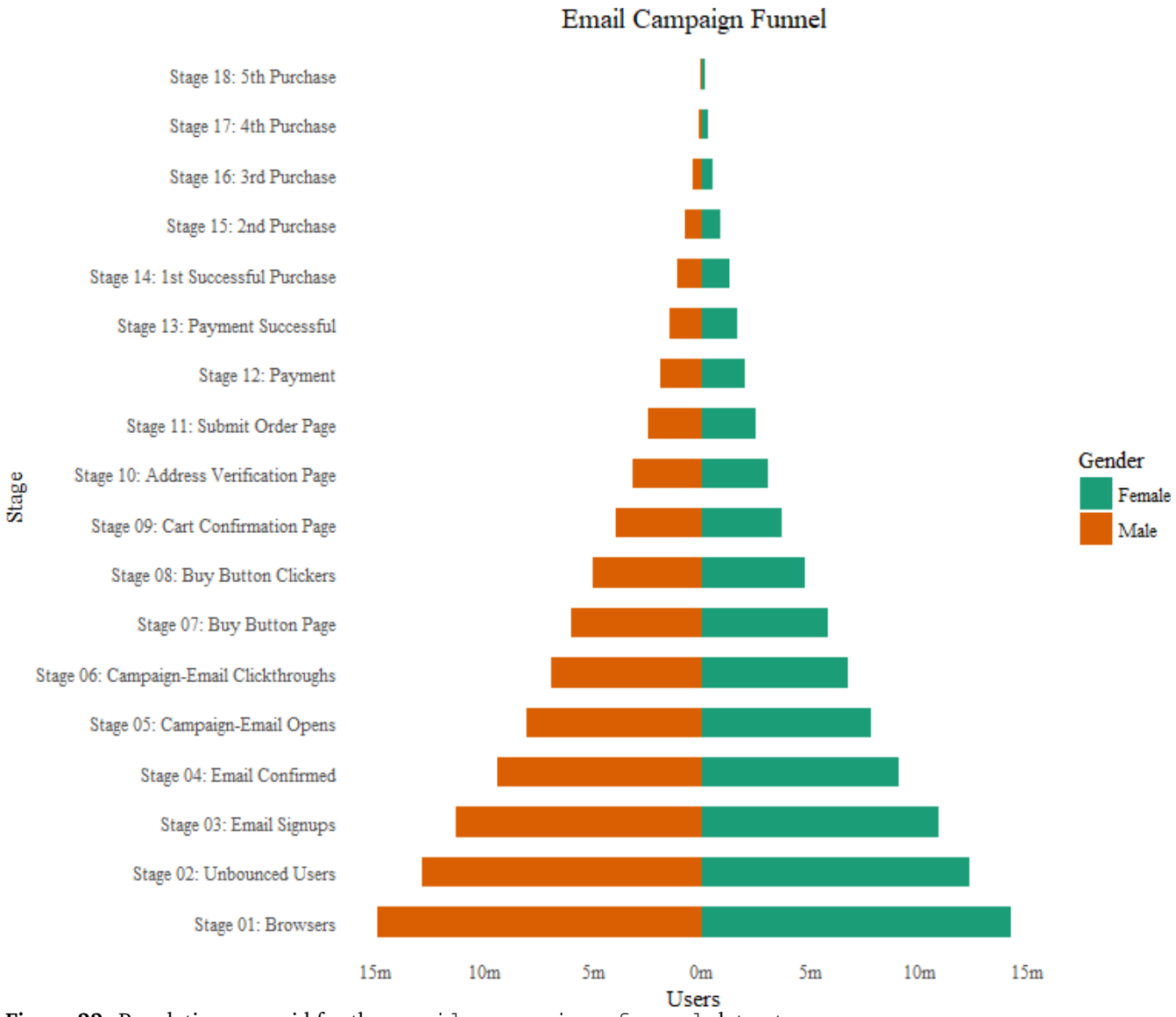

Figure 22. Population pyramid for the `email campaign funnel` dataset.

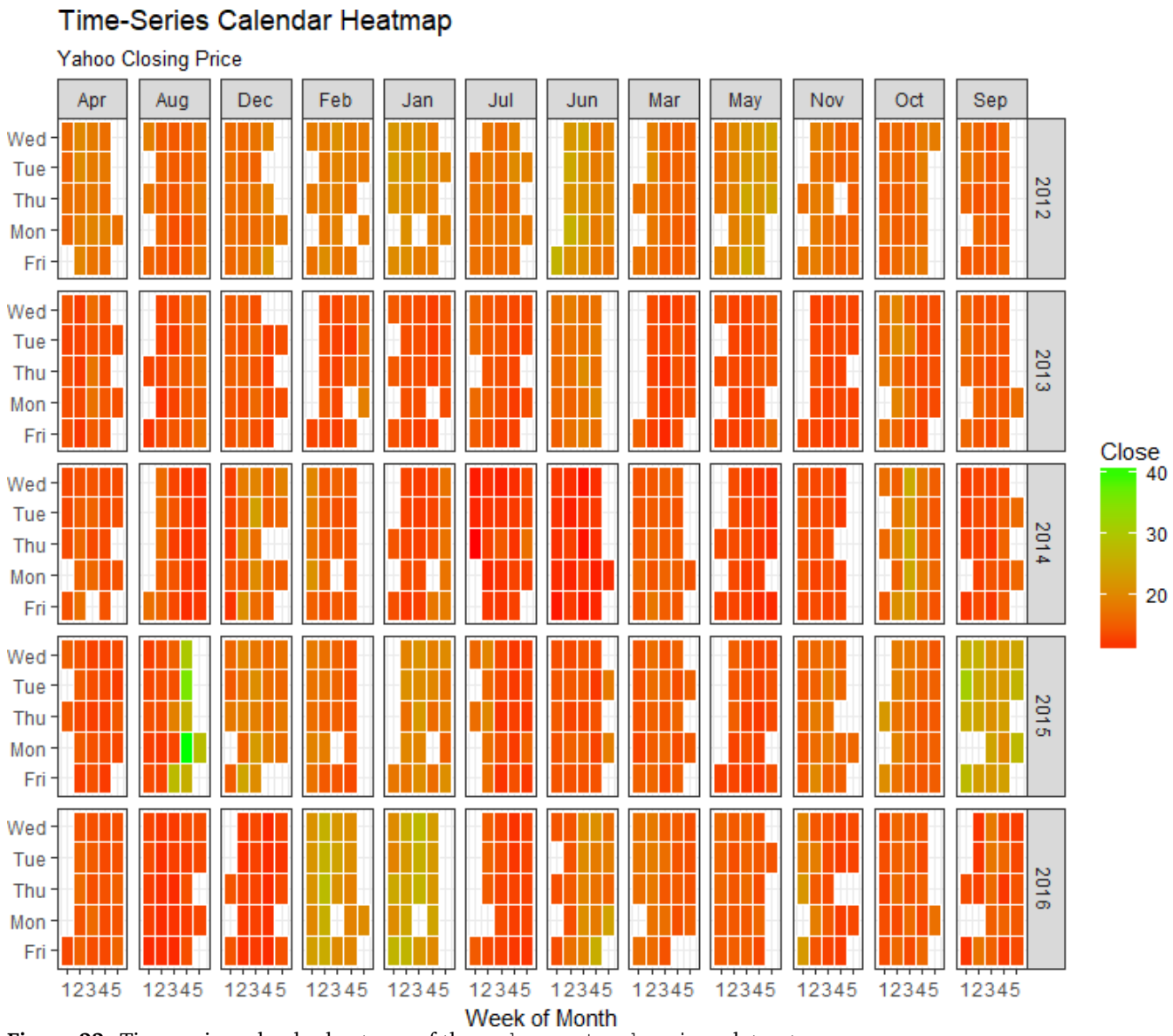

Figure 23. Time-series calendar heatmap of the `yahoo stockprice` dataset.

Example 10 (Ordered Bar Chart)
An ordered bar chart is a bar chart that is ordered by the y—axis variable. It is not sufficient to sort the dataframe by the variable of interest; in order for the bar chart to retain the row ordering, the x—axis variable (i.e. the categories) has to be converted into a factor object.

This is shown in Figure 24, for the mean city mileage of each manufacturer in the `mpg` dataset. The data is first aggregated and sorted, and the x—variable is converted to a factor.

```
# Prepare data: group mean city mileage
    by manufacturer.
cty_mpg <- aggregate(mpg$cty,
    by=list(mpg$manufacturer), FUN=mean)
    # aggregate
colnames(cty_mpg) <- c("make",
    "mileage") # change column names
cty_mpg <- cty_mpg[
    order(cty_mpg$mileage),] # sort
cty_mpg$make <- factor(cty_mpg$make,
    levels = cty_mpg$make) # to retain
    the order in plot.

# Plot
library(ggplot2)
theme_set(theme_bw())

# Draw plot
ggplot(cty_mpg, aes(x=make, y=mileage)) +
  geom_bar(stat="identity", width=.5,
      fill="tomato3") +
  labs(title="Ordered Bar Chart",
       subtitle="Make Vs Avg. Mileage",
       caption="source: mpg") +
  theme(axis.text.x =
      element_text(angle=65, vjust=0.6))
```

Example 11 (Correlogram)
Correlograms can be used to test the level of correlation among the data variables. The cells of the matrix can be shaded or coloured to show the correlation value (the darker the colour, the higher the magnitude of the correlation between a pair of variables). Positive correlations are displayed in one colour, and negative correlations in another, with intensity proportional to the actual correlation value. This is conveniently implemented using the *ggcorrplot* package (see Figure 25).

```
#install.packages("ggcorrplot")
library(ggplot2)
library(ggcorrplot)

# Correlation matrix
data(mtcars)
corr <- round(cor(mtcars), 1)

# Plot
ggcorrplot(corr, hc.order = TRUE,
        type = "lower",
        lab = TRUE,
        lab_size = 3,
        method="circle",
        colors = c("tomato2", "white",
            "springgreen3"),
        title="Correlogram of mtcars",
        ggtheme=theme_bw)
```

Example 12 (Treemap)
A treemap requires a data frame with (at least) the following columns:

- a numeric column, which determines the area of each treemap rectangle, and
- another numeric column, which determines the fill colour of each treemap rectangle.

The *treemapify* package includes, as an example, a dataset containing statistics about the G20 world economies. For this example, we will further use two optional columns: a factor column, containing labels for each rectangle (Country) and a second factor column, containing labels for groups of rectangles (Region) – see Figure 26 for the final display.

We start by drawing a treemap where each tile represents a G20 country. The area of the tile will be mapped to the country's GDP, and the tile's fill colour mapped to its HDI (Human Development Index). The basic geom used for that purpose is `geom_treemap()`, but without a label to identify each country, the display will not be very insightful. To add a text label to each tile, use `geom_treemap_text()`, which uses the *ggfittext* package to resize the text so that it fits inside the tile.

In addition to standard text formatting aesthetics in `geom_text()` (like `fontface` or `color`), *ggfittext*-specific options are available; for example, we can centre the text in the tile with `place = "centre"`, and expand it to fill as much of the tile as possible with `grow = TRUE`.

The `geom_treemap` geom supports subgrouping by passing a subgroup aesthetic. Countries can be subdivided by region, say; `geom_treemap_subgroup_border` can be used to draw a border around these regions, with labels given by `geom_treemap_subgroup_text` (the latter takes the same input arguments for text placement and resizing as `geom_treemap_text`).

Like any *ggplot2* plot, *treemapify* plots can be faceted, scaled, themed, etc.

```
library(devtools)
#devtools::install_github("wilkox/treemapify")
library(treemapify)
library(ggplot2)
data(G20)
```

```
ggplot(G20, aes(area = gdp_mil_usd, fill
    = region, label = country)) +
  geom_treemap() +
  geom_treemap_text(grow = T, reflow =
     T, colour = "black") +
  facet_wrap( ~ econ_classification) +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position = "bottom") +
  labs(
    title = "The G20 Economies",
    caption = "The area of each country
       is proportional to its relative
       GDP
    within the economic group (advanced
       or developing)",
    fill = "Region"
  )
```

Example 13 (Network Visualization)

This example using the *geomnet* package has been chosen from a social network from the popular television show *Mad Men* (which we have never actually seen, for the record). The network it displays uses data and code that made available at CRAN's gcookbook page [1]; it consists of 52 vertices and 87 edges. Each vertex represents a character on the show; there is an edge between two characters if they have shared a romantic relationship.

The network visualization of Figure 27 is provided by the *ggnetwork* package under the *ggplot2* framework, using layering..

```
library(ggplot2)
library(ggnetwork)
library(geomnet)
library(network)
# make the data available
data(madmen, package = 'geomnet')
# create undirected network
mm.net <- network(madmen$edges[, 1:2],
    directed = FALSE)
# mm.net # glance at network object

# create node attribute (gender)
rownames(madmen$vertices) <-
    madmen$vertices$label
mm.net %v% "gender" <- as.character(
  madmen$vertices[
     network.vertex.names(mm.net),
     "Gender"])
# gender color palette
mm.col <- c("female" = "#ff0000", "male"
    = "#00ff00")
set.seed(10052016)
ggplot(data = ggnetwork(mm.net, layout =
    "kamadakawai"),
      aes(x, y, xend = xend, yend =
          yend)) +
  geom_edges(color = "grey50") + # draw
     edge layer
  geom_nodes(aes(colour = gender), size
     = 2) + # draw node layer
  geom_nodetext(aes(colour = gender,
     label = vertex.names),
            size = 3, vjust = -0.6) + #
               draw node label layer
  scale_colour_manual(values = mm.col) +
  xlim(c(-0.05, 1.05)) +
  theme_blank() +
  theme(legend.position = "bottom")
```

In the plot, we can see that there is one central character who has many more relationships than any other character. This vertex represents the main character of the show, Don Draper, who is apparently quite the Lothario. Networks can be found practically in all data environments; *ggnetwork* provides the curious reader with a straightforward way to visualize any network.

Colouring the vertices or edges in a graph is a quick way to visualize grouping and helps with pattern or cluster detection. The vertices in a network and the edges between them compose the structure of a network, and being able to visually discover patterns among them is a key part of network analysis.

Example 14 (Time Series Plot From a Time Series Object)

The *ggfortify* package allows autoplot to automatically plot directly from a `ts` object (see Figure 28).

```
## From Timeseries object (ts)
library(ggplot2)
library(ggfortify)
# Plot
autoplot(AirPassengers) +
  labs(title="AirPassengers") +
  theme(plot.title =
      element_text(hjust=0.5))
```

Example 15 (Time Series Plot From a Data Frame)

Using `geom_line()`, a time series (or line chart) can be drawn from a data frame as well. The horizontal axis breaks are generated by default. In the example below (Figure 29), the breaks are formed once every 10 years.

```
library(ggplot2)
theme_set(theme_classic())
# Allow Default X Axis Labels
ggplot(economics, aes(x=date)) +
  geom_line(aes(y=unemploy)) +
  labs(title="Time Series Chart",
      subtitle="Number of unemployed in
         thousands from 'Economics-US'
         Dataset",
      caption="Source: Economics",
      y="unemploy")
```

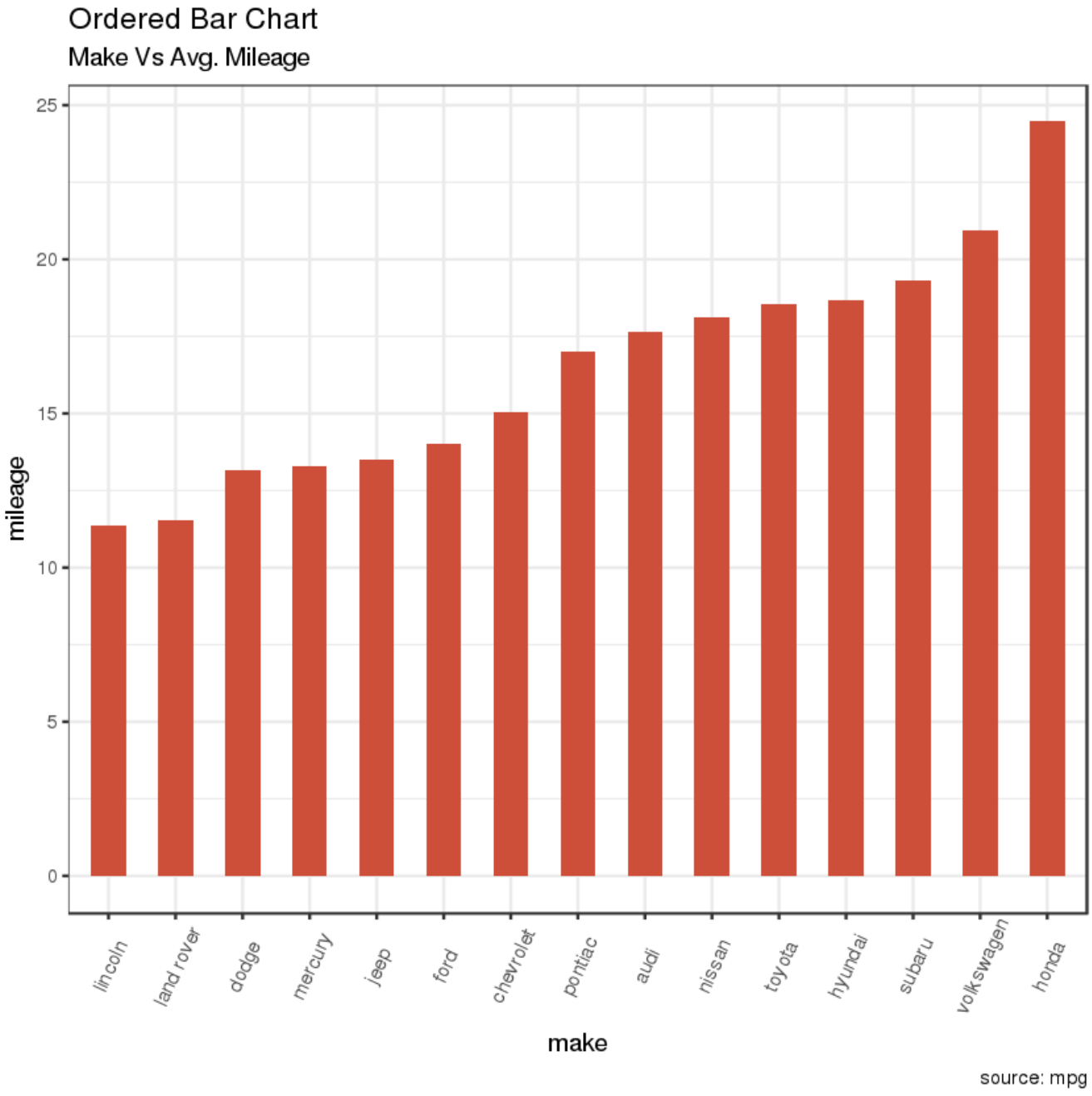

Figure 24. Ordered bar plot of the `mpg` dataset.

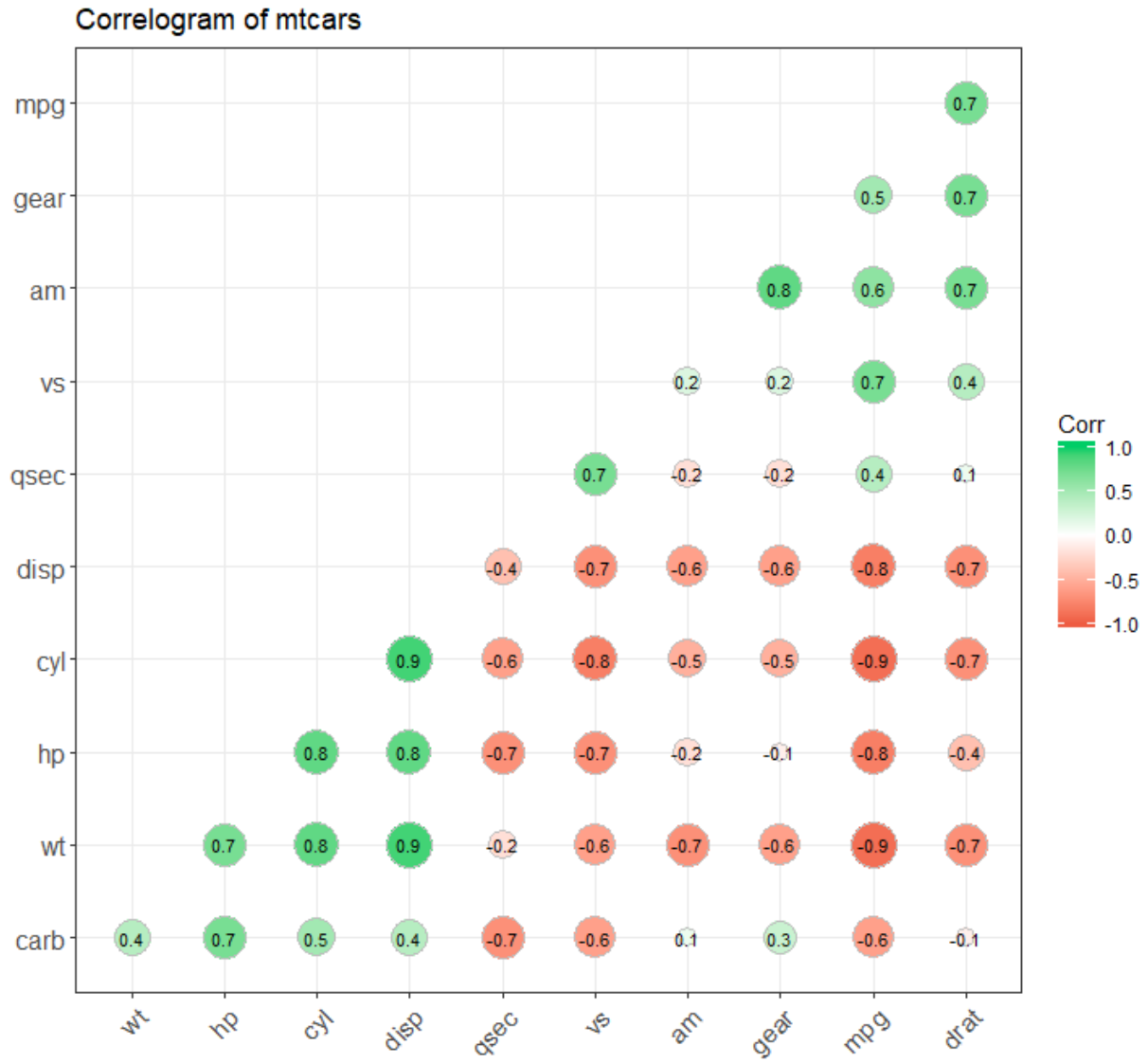

Figure 25. Correlogram of the `mtcars` dataset.

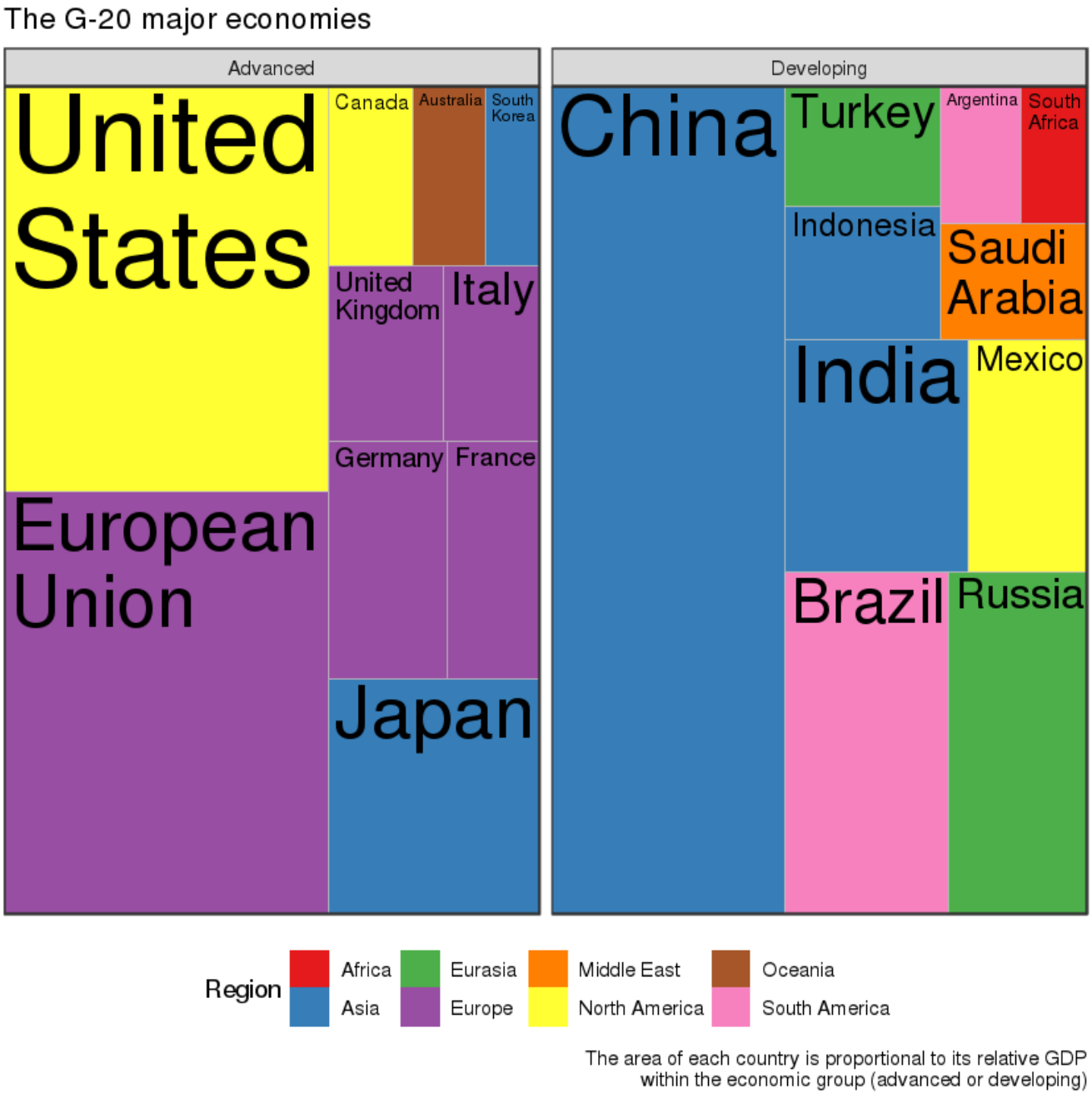

Figure 26. Treemap for the G20 economies.

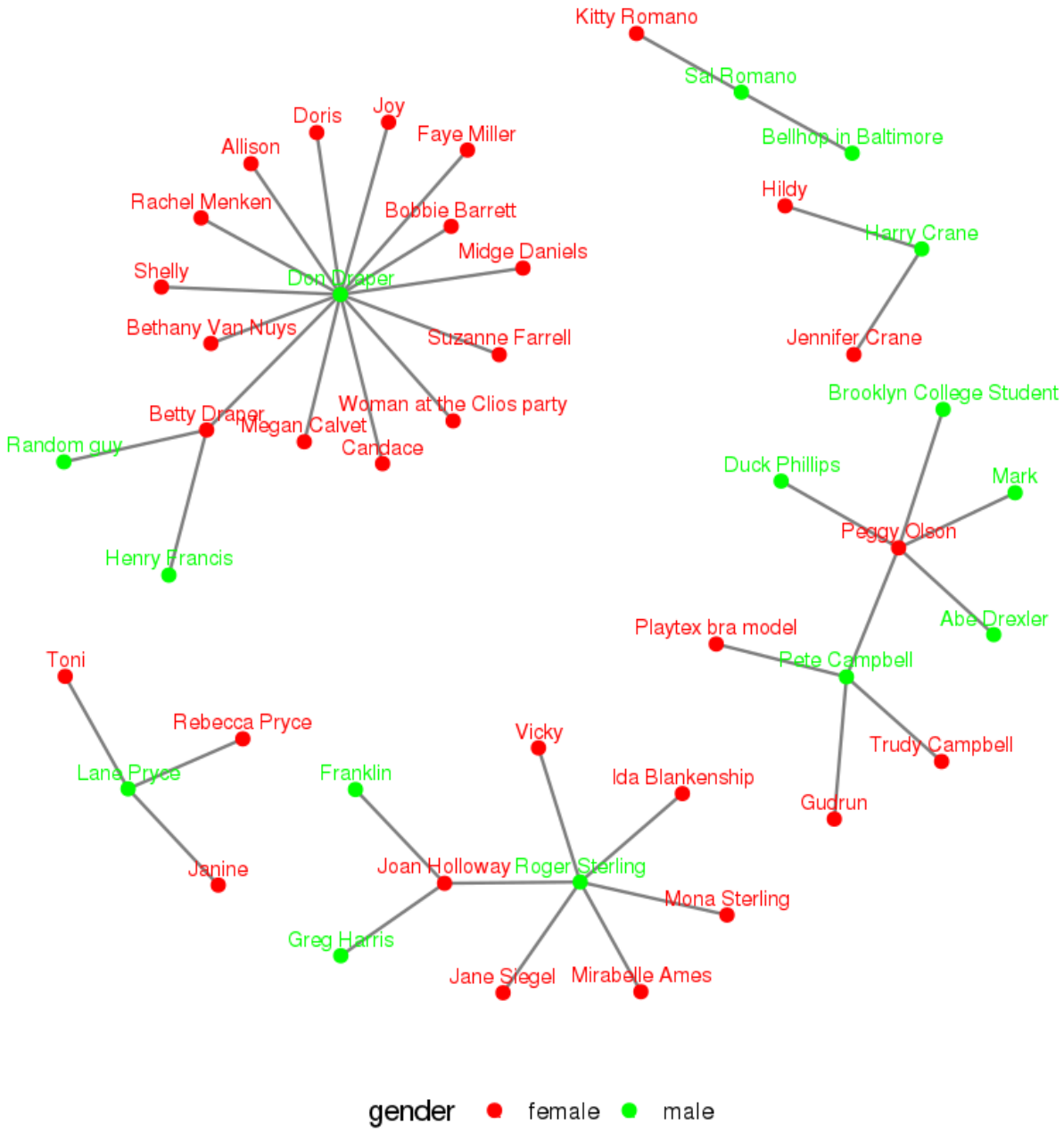

Figure 27. Graph of *Mad Men* characters who are linked by a romantic relationship.

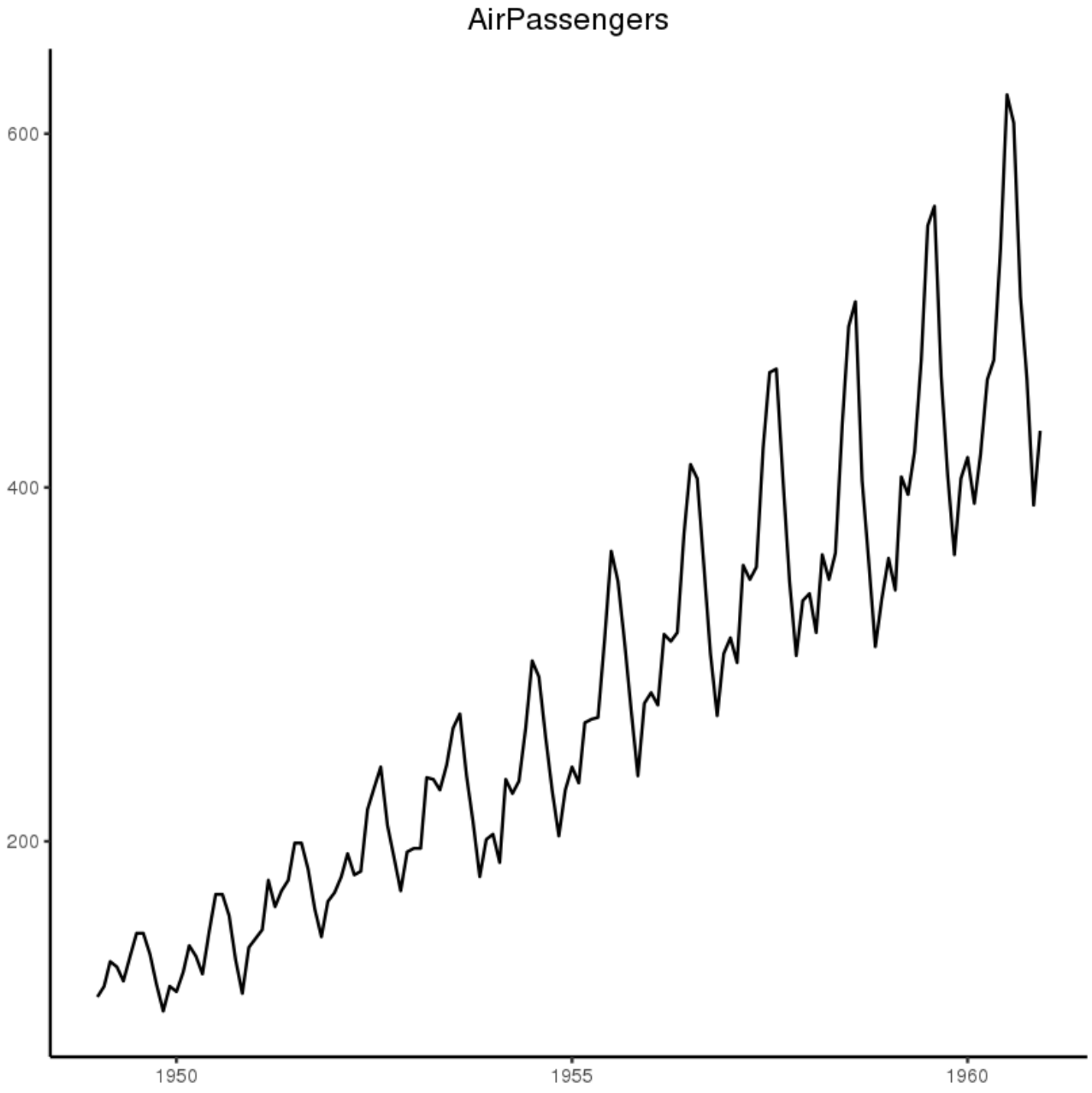

Figure 28. Time series plot for the `AirPassengers` dataset.

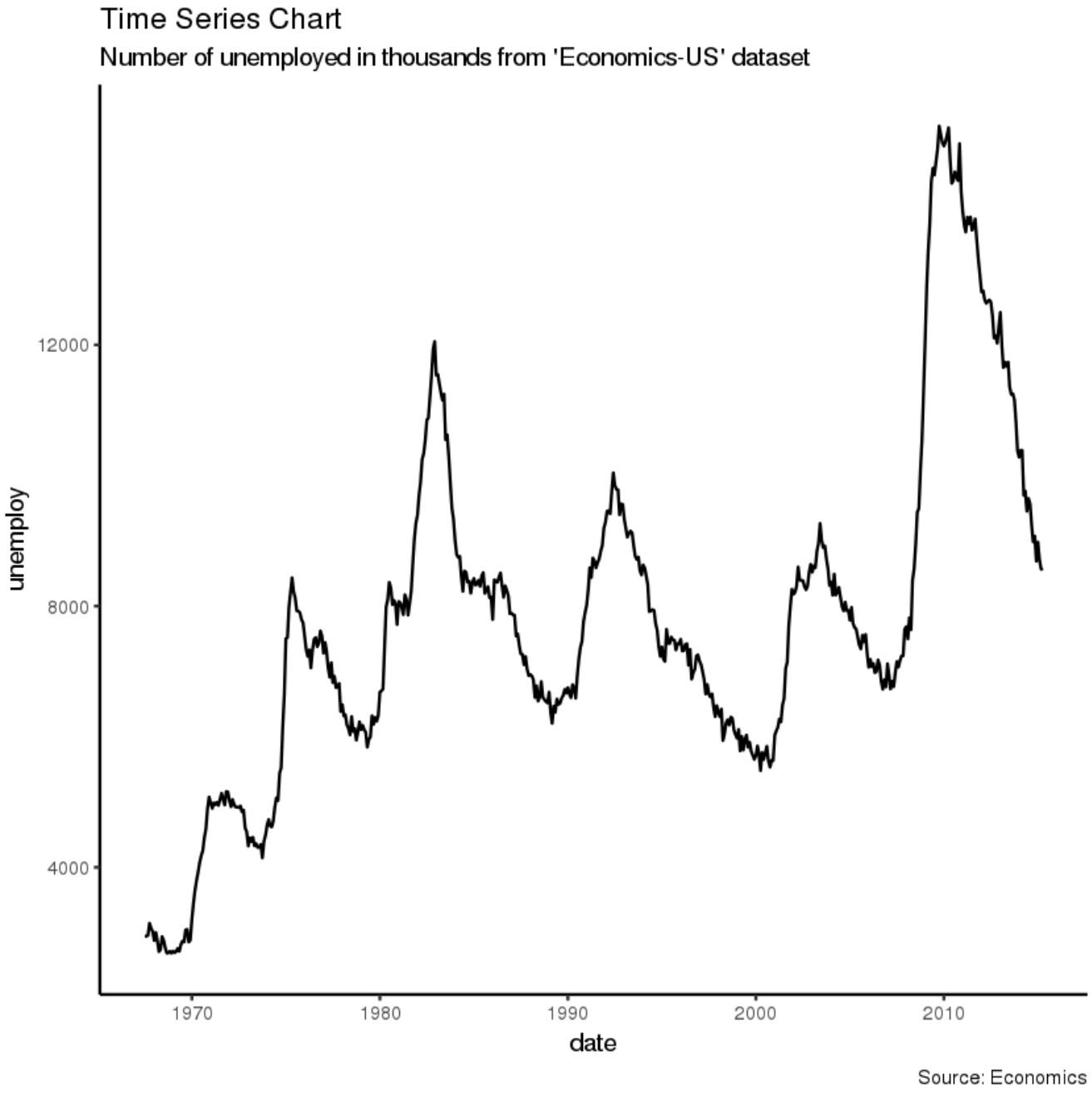

Figure 29. Time series plot for the `economics` dataset.

Example 16 (Time Series Plot For a Monthly Time Series)
In order to select specific breaks on the x—axis, consider the functionality offered by `scale_x_date()` (plot shown in Figure 30).

```
library(ggplot2)
library(lubridate)
theme_set(theme_bw())

economics_m <- economics[1:24, ]

# labels and breaks for X axis text
lbls <-
    paste0(month.abb[month(economics_m$date)
    ], " ",
    lubridate::year(economics_m$date))
brks <- economics_m$date

# plot
ggplot(economics_m, aes(x=date)) +
  geom_line(aes(y=pce)) +
  labs(title="Monthly Time Series",
      subtitle="Personal consumption
          expenditures, in billions of
          dollars",
      caption="Source: Economics",
      y="pce") + # title and caption
  scale_x_date(labels = lbls,
            breaks = brks) + # change to
               monthly ticks and labels
  theme(axis.text.x = element_text(angle
      = 90, vjust=0.5), # rotate x axis
      text
       panel.grid.minor =
          element_blank()) # turn off
          minor grid
```

Example 17 (Time Series Plot For a Yearly Time Series)
Here's the same, but with a yearly breakdown (plot shown in Figure 31).

```
library(ggplot2)
library(lubridate)
theme_set(theme_bw())

economics_y <- economics[1:90, ]

# labels and breaks for X axis text
brks <- economics_y$date[seq(1,
    length(economics_y$date), 12)]
lbls <- lubridate::year(brks)

# plot
ggplot(economics_y, aes(x=date)) +
  geom_line(aes(y=psavert)) +
  labs(title="Yearly Time Series",
      subtitle="Personal savings rate",
      caption="Source: Economics",
      y="psavert") + # title and caption
  scale_x_date(labels = lbls,
            breaks = brks) + # change to
               monthly ticks and labels
  theme(axis.text.x = element_text(angle
      = 90, vjust=0.5), # rotate x axis
      text
       panel.grid.minor =
          element_blank()) # turn off
          minor grid
```

Example 18 (Time Series Plot From Long Data Format)
In this example, we construct the plot from a long data format (i.e. the column names and respective values of all the columns are stacked in only 2 variables – `variable` and `value`, respectively). In the wide format, the data would takee the appearance of the `economics` dataset.

Below, the `geom_line` objects are drawn using `value` and `aes(col)` is set to `variable`. In this way, multiple coloured lines are plotted (one for each unique `variable` level) with a single call; `scale_x_date()` changes the x—axis breaks and labels, while the line colours are changed by `scale_color_manual`.

```
data(economics_long, package = "ggplot2")
# head(economics_long)
library(ggplot2)
library(lubridate)
theme_set(theme_bw())
df <-
    economics_long[economics_long$variable
    %in% c("psavert", "uempmed"), ]
df <- df[lubridate::year(df$date) %in%
    c(1967:1981), ]

# labels and breaks for X axis text
brks <- df$date[seq(1, length(df$date),
    12)]
lbls <- lubridate::year(brks)

# plot
ggplot(df, aes(x=date)) +
  geom_line(aes(y=value, col=variable)) +
  labs(title="Time Series of Returns
      Percentage",
      subtitle="Drawn from Long Data
          format",
      caption="Source: Economics",
      y="Returns %",
      color=NULL) + # title and caption
  scale_x_date(labels = lbls, breaks =
      brks) + # change to monthly ticks
      and labels
  scale_color_manual(labels =
      c("psavert", "uempmed"),
                values =
                    c("psavert"="#00ba38",
                    "uempmed"="#f8766d"))
                    + # line color
```

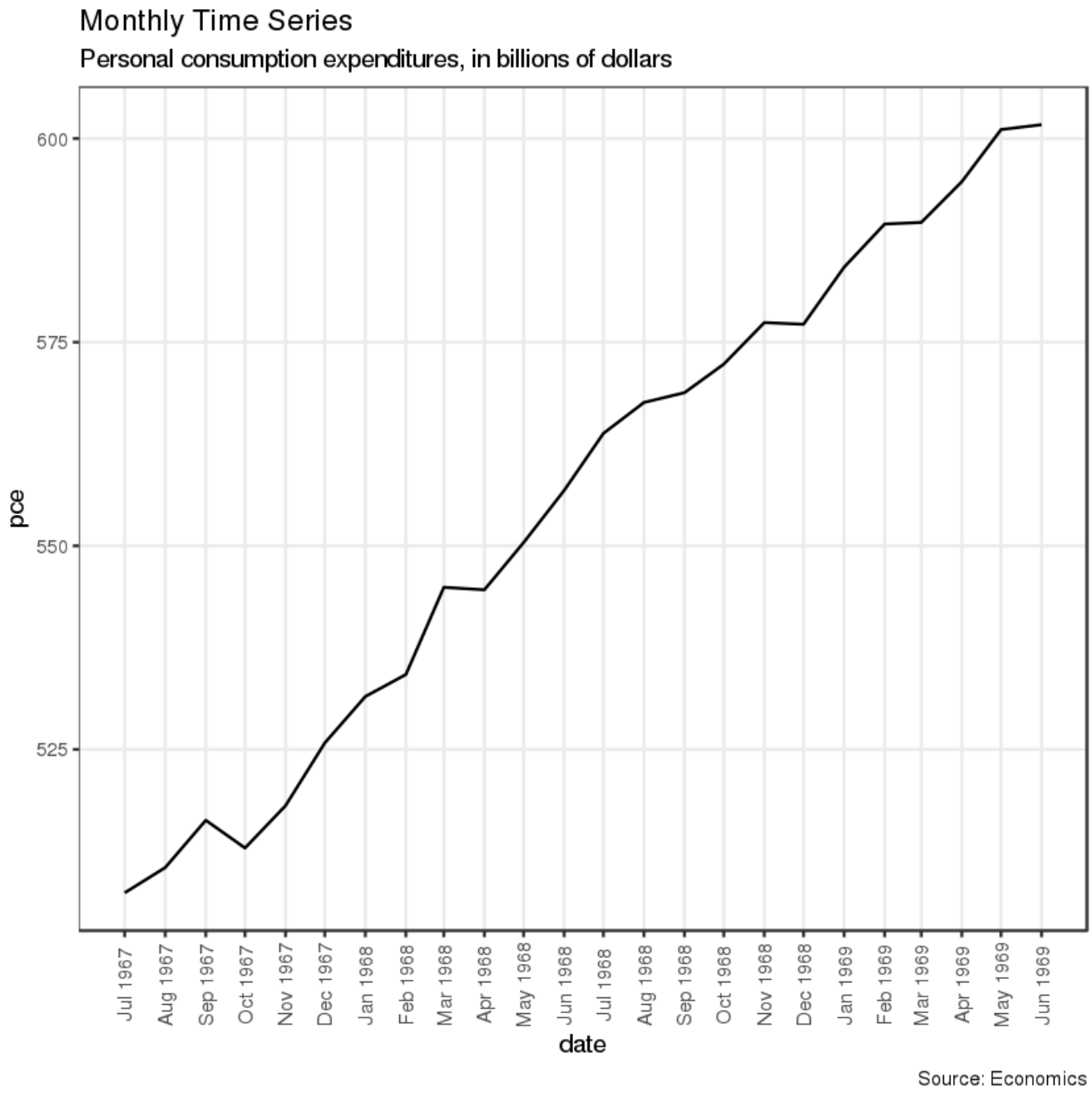

Figure 30. Monthly time series for the `economics` dataset.

```
theme(axis.text.x = element_text(angle
    = 90, vjust=0.5, size = 8), #
    rotate x axis text
     panel.grid.minor =
         element_blank()) # turn off
         minor grid
```

Example 19 (Stacked Area Chart)
A stacked area chart is just like a line chart, except that the region below the plot is filled in. This is typically used to:

- describe how a quantity or volume (rather than something like a price) changes over time;
- when the data contains a "large" number of points (for "small" datasets, consider using a bar chart), or
- when the respective contributions of each individual component needs to be highlighted.

The appropriate call uses `geom_area`, which works very much like `geom_line`, with an important difference – by default, each `geom_area` starts from the bottom of $y-$axis (which is typically set at 0), but in order to show

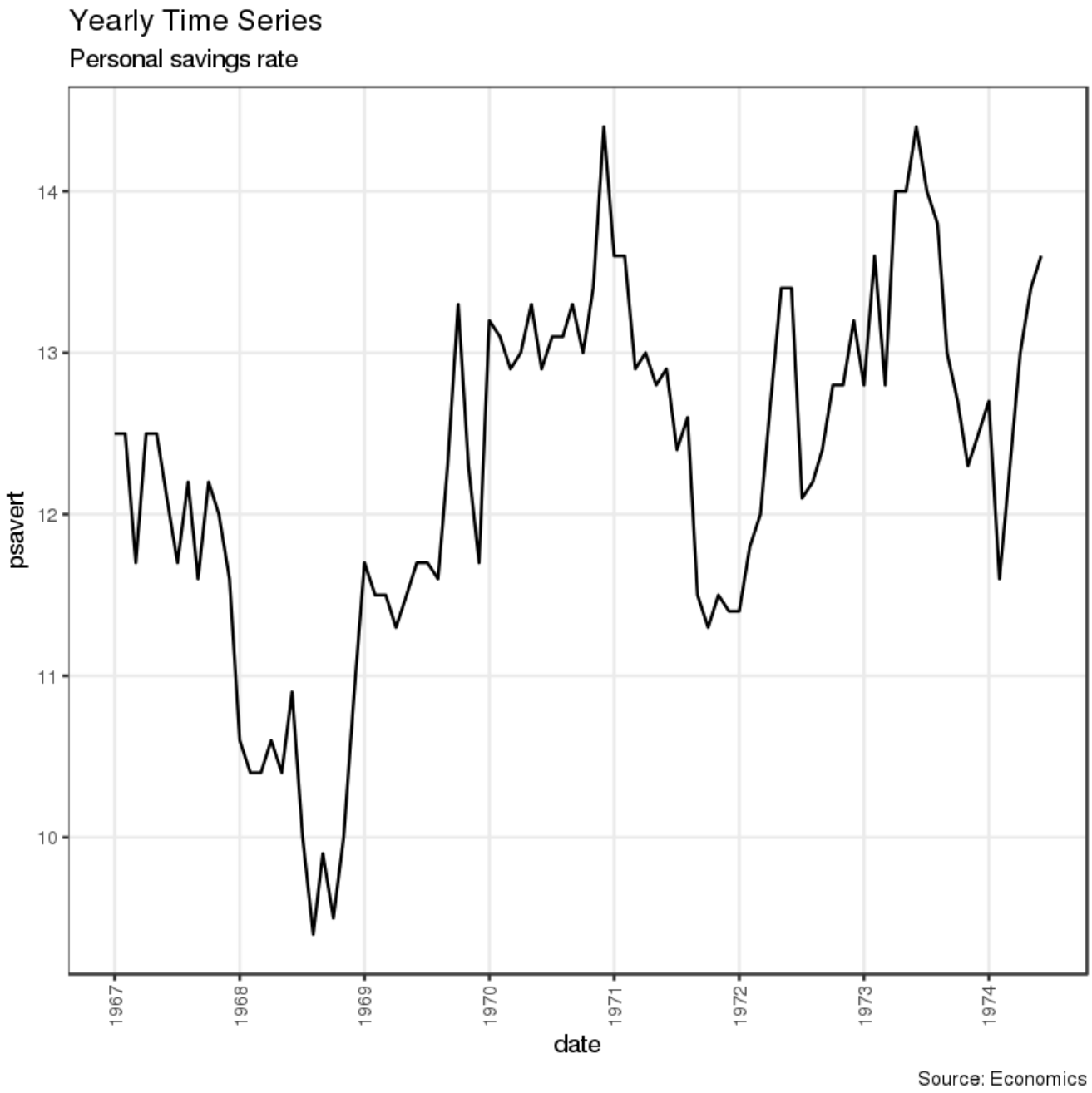

Figure 31. Yearly time series for the `economics` dataset.

the contribution from individual components, the area has to be stacked on top of the previous component, rather than relative to the floor of the plot. All the bottom layers have to be added to the y value of a new area.

In the example below (and in Figure 33), the top layer is `y=psavert+uempmed`. However nice the plot might look, keep in mind that it can be difficult to interpret.

```
library(ggplot2)
library(lubridate)
theme_set(theme_bw())
df <- economics[, c("date", "psavert",
    "uempmed")]
df <- df[lubridate::year(df$date) %in%
    c(1967:1981), ]
# labels and breaks for X axis text
brks <- df$date[seq(1, length(df$date),
    12)]
lbls <- lubridate::year(brks)
# plot
ggplot(df, aes(x=date)) +
  geom_area(aes(y=psavert+uempmed,
```

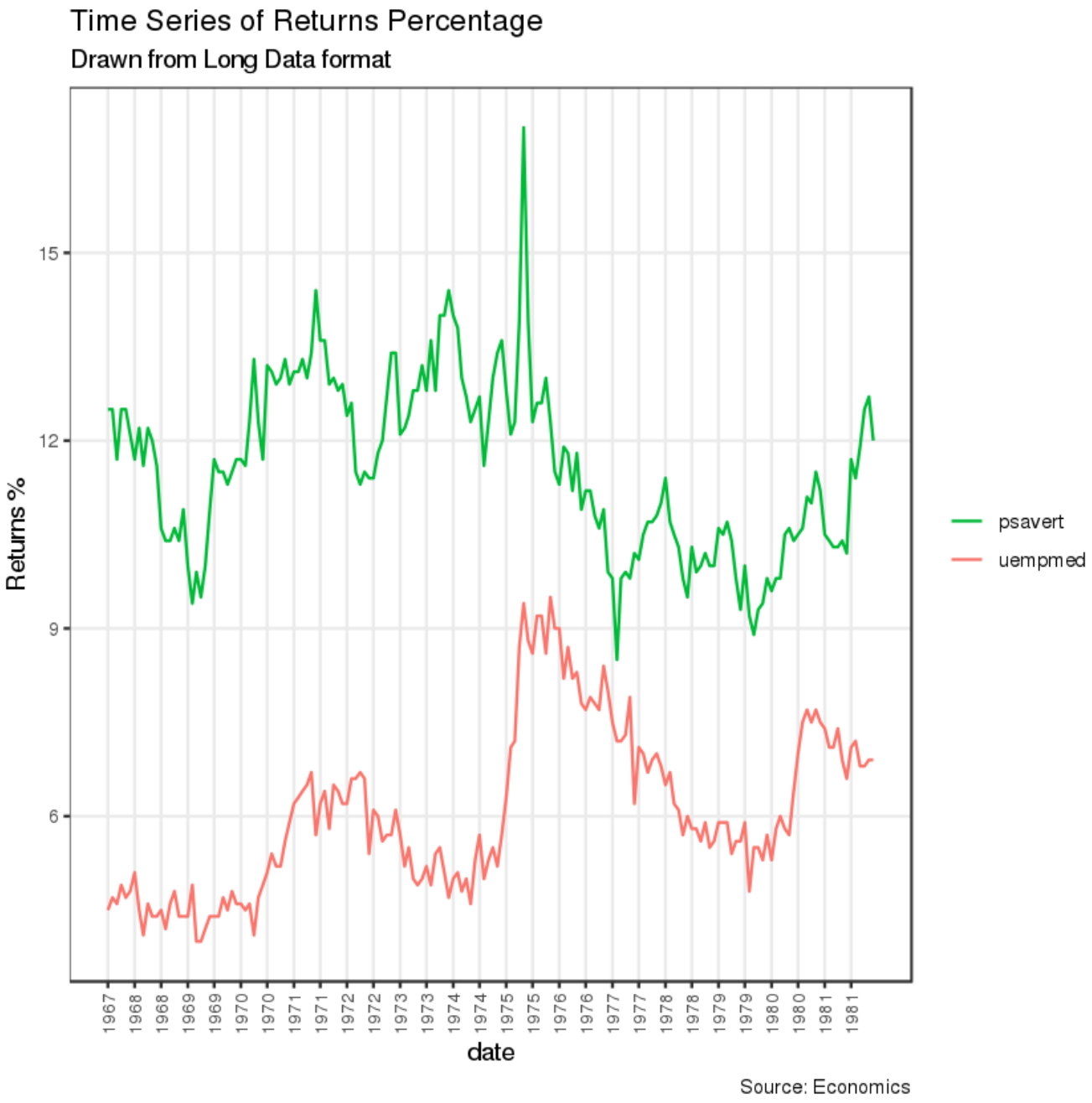

Figure 32. Time series from long data format for the `economics` dataset.

```
    fill="psavert")) +
geom_area(aes(y=uempmed,
    fill="uempmed")) +
labs(title="Area Chart of Returns
    Percentage",
     subtitle="From Wide Data format",
     caption="Source: Economics",
     y="Returns %") + # title and caption
scale_x_date(labels = lbls, breaks =
    brks) + # change to monthly ticks
    and labels
scale_fill_manual(name="",
                  values =
                     c("psavert"="#00ba38",
                     "uempmed"="#f8766d"))
                     + # line color
theme(panel.grid.minor =
    element_blank()) # turn off minor
    grid
```

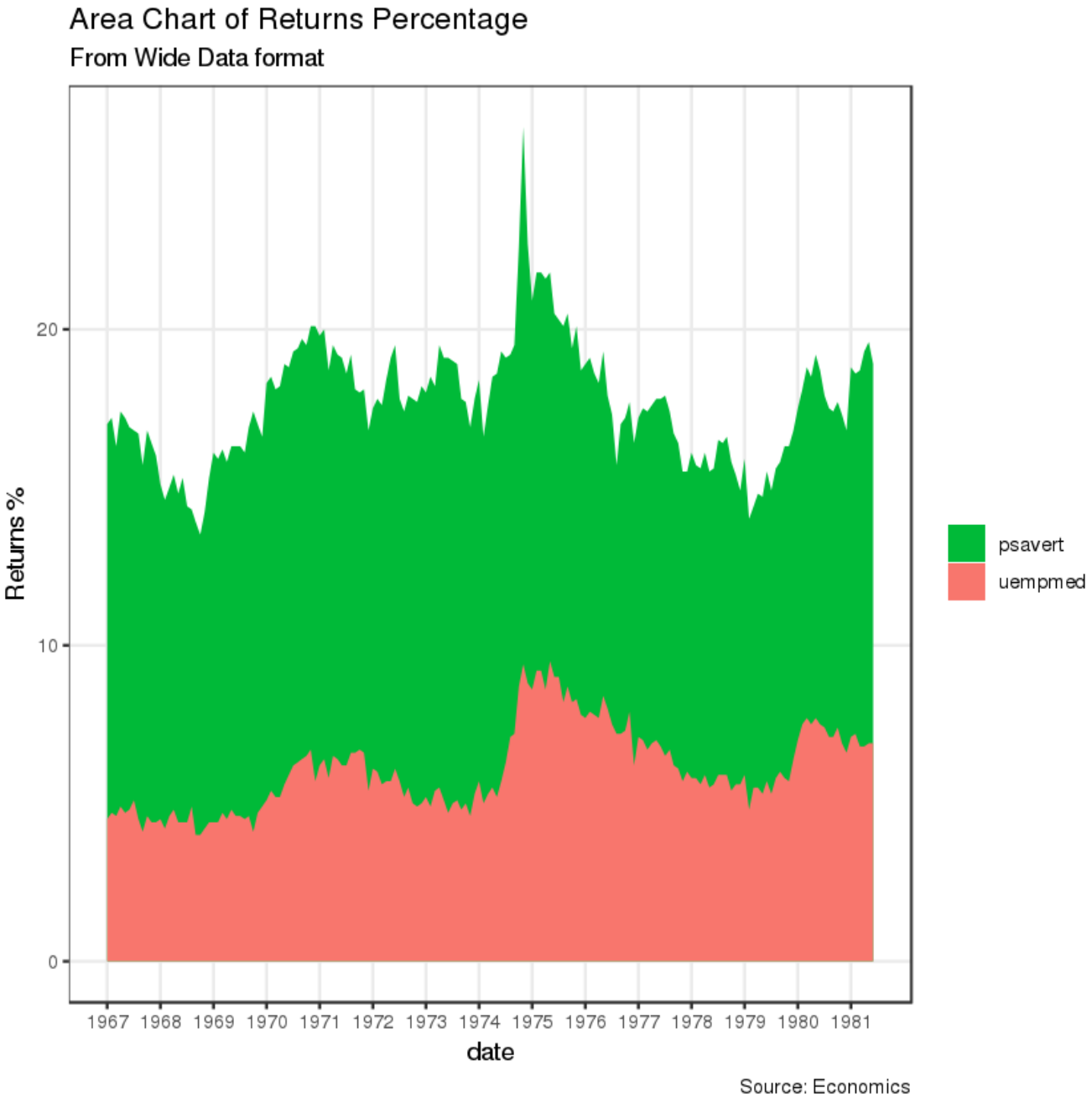

Figure 33. Stacked area chart for the `economics` dataset (green area is the sum of `psavert` and `uempmed`).

Example 20 (Seasonal Plot)
When working with a time series object of class `ts` or `xts`, the seasonal fluctuations can be viewed through a seasonal plot using *forecast::ggseasonplot*.

You can see the traffic increase in air passengers in `AirPassengers` over the years along with the repetitive seasonal patterns in traffic; in the same vein, the temperatures in `nottem` do not increase over time, but they definitely follow a seasonal pattern (see below, and Figure 34).

```
library(ggplot2)
library(forecast)
theme_set(theme_classic())
# Subset data
nottem_small <- window(nottem,
    start=c(1920, 1), end=c(1925, 12)) #
    subset a smaller timewindow
# Plot
ggseasonplot(AirPassengers) +
    labs(title="Seasonal plot:
    International Airline Passengers")
ggseasonplot(nottem_small) +
    labs(title="Seasonal plot: Air
    temperatures at Nottingham Castle")
```

Example 21 (Parallel Coordinate Plots)
Parallel coordinate plots are useful to visualize multivariate data. As a practical example, assume that a survey has been conducted, with a variety of questions. Each question is asked three times – in a different context – and is answered on a discrete scale from 1 to 7. Consequently, each question has three "dimensions". The distribution of answers across the three dimensions should be displayed for each question. Because the three dimensions have the same unit and scale, they can easily be compared on parallel coordinates (it would be possible to display more than three dimensions, of course).

```
library(triangle)
set.seed(0)
q1_d1 <- round(rtriangle(1000, 1, 7, 5))
q1_d2 <- round(rtriangle(1000, 1, 7, 6))
q1_d3 <- round(rtriangle(1000, 1, 7, 2))
df <- data.frame(q1_d1 = factor(q1_d1),
    q1_d2 = factor(q1_d2), q1_d3 =
    factor(q1_d3))
```

We are using the triangular distribution to get random integers $r \in [1,7]$, around a different mode c for each dimension (5, 6 and 2). To plot the main "answer paths" (i.e. the most frequent answer combination across the three dimensions), we need to group by all dimensions, and then to count the frequency of each unique answer combinations. This can be done with the *dplyr* package.

```
library(dplyr)
# group by combinations and count
df_grouped <- df %>% group_by(q1_d1,
    q1_d2, q1_d3) %>% count()
# set an id string that denotes the
    value combination
df_grouped <- df_grouped %>% mutate(id =
    factor(paste(q1_d1, q1_d2, q1_d3,
    sep = '-')))
order.freq <-
    order(df_grouped[,4],decreasing=TRUE)
# sort by count and select top rows
df_grouped <-
    df_grouped[order.freq[1:25],]
```

The count per group is automatically stored in a column `n`. We additionally set an `id` column which denotes the unique answer combination. The dataset is sorted by decreasing count and the 25 most frequent paths are retained (optional).

We can now plot the data by using `geom_path`, after processing the data appropriately. We need to convert our grouped data frame into a "long format" using *melt()* from the package *reshape2* so that our three dimensions are contained in a column named "variable" and the respective values are in the column "values":

```
library(reshape2)
library(ggplot2)
# create long format
df_pcp <- melt(df_grouped, id.vars =
    c('id', 'freq'))
df_pcp$value <- factor(df_pcp$value)
```

We can then specify what levels should be drawn on the y—axis (1 to 7). In the `ggplot()` function we define an aesthetic that uses the "variable" column for the x—axis and the "value" column for the y—axis. We also specify that the values should be grouped by using the `id` column. This is required, as the connections between the three dimensions won't be drawn otherwise. We use `geom_path()` to draw the connection lines and make the width and colour of the connection dependent on the `n` and `id` columns, respectively,

```
y_levels <- levels(factor(1:7))
ggplot(df_pcp, aes(x = variable, y =
    value, group = id)) + # group = id
    is important!
  geom_path(aes(size = freq, color = id),
          alpha = 0.5,
          lineend = 'round', linejoin =
              'round') +
  scale_y_discrete(limits = y_levels,
      expand = c(0.5, 0)) +
  scale_size(breaks = NULL, range = c(1,
      7))
```

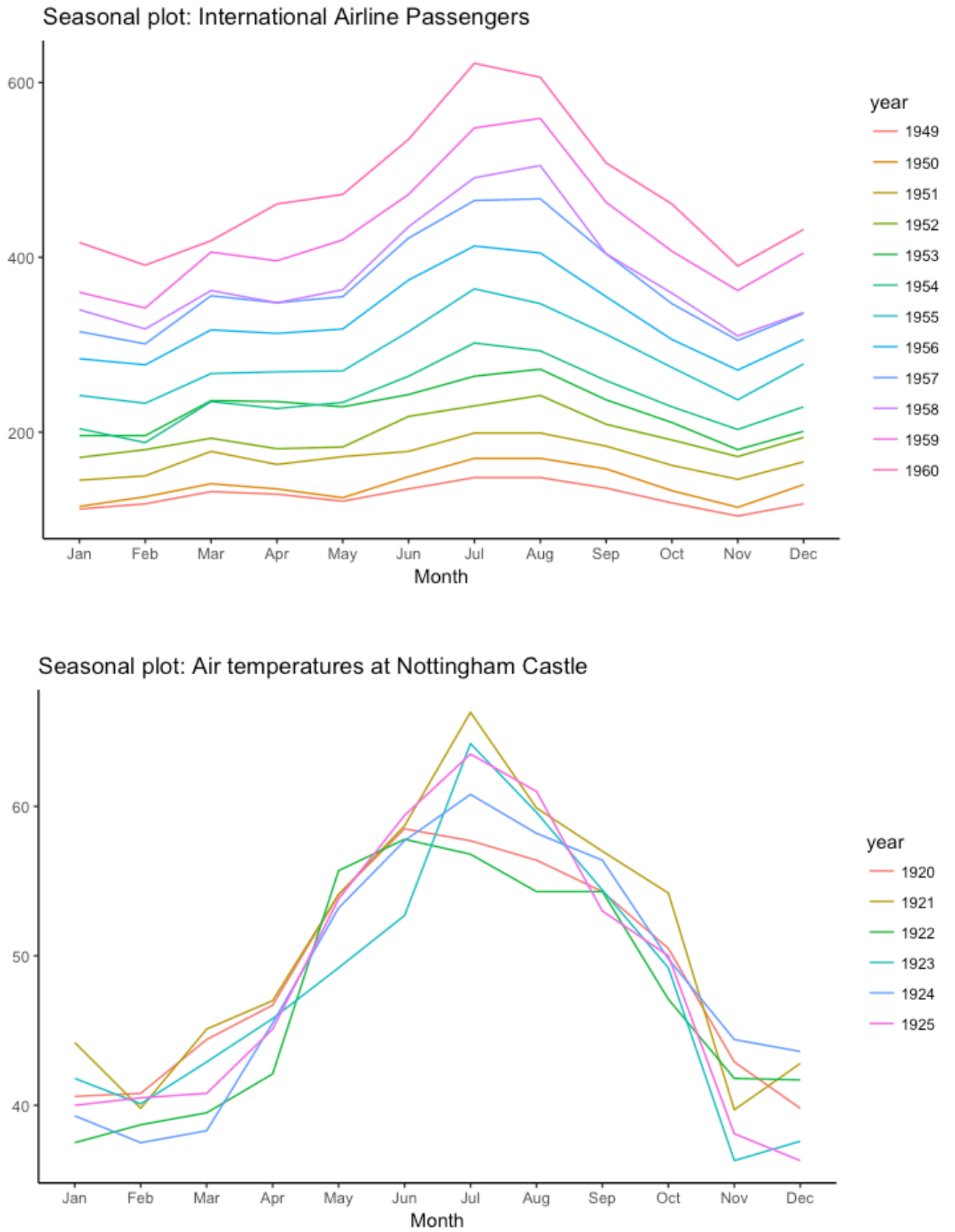

Figure 34. Seasonal plots for the `AirPassengers` dataset (above) and the `nottem` dataset (below).

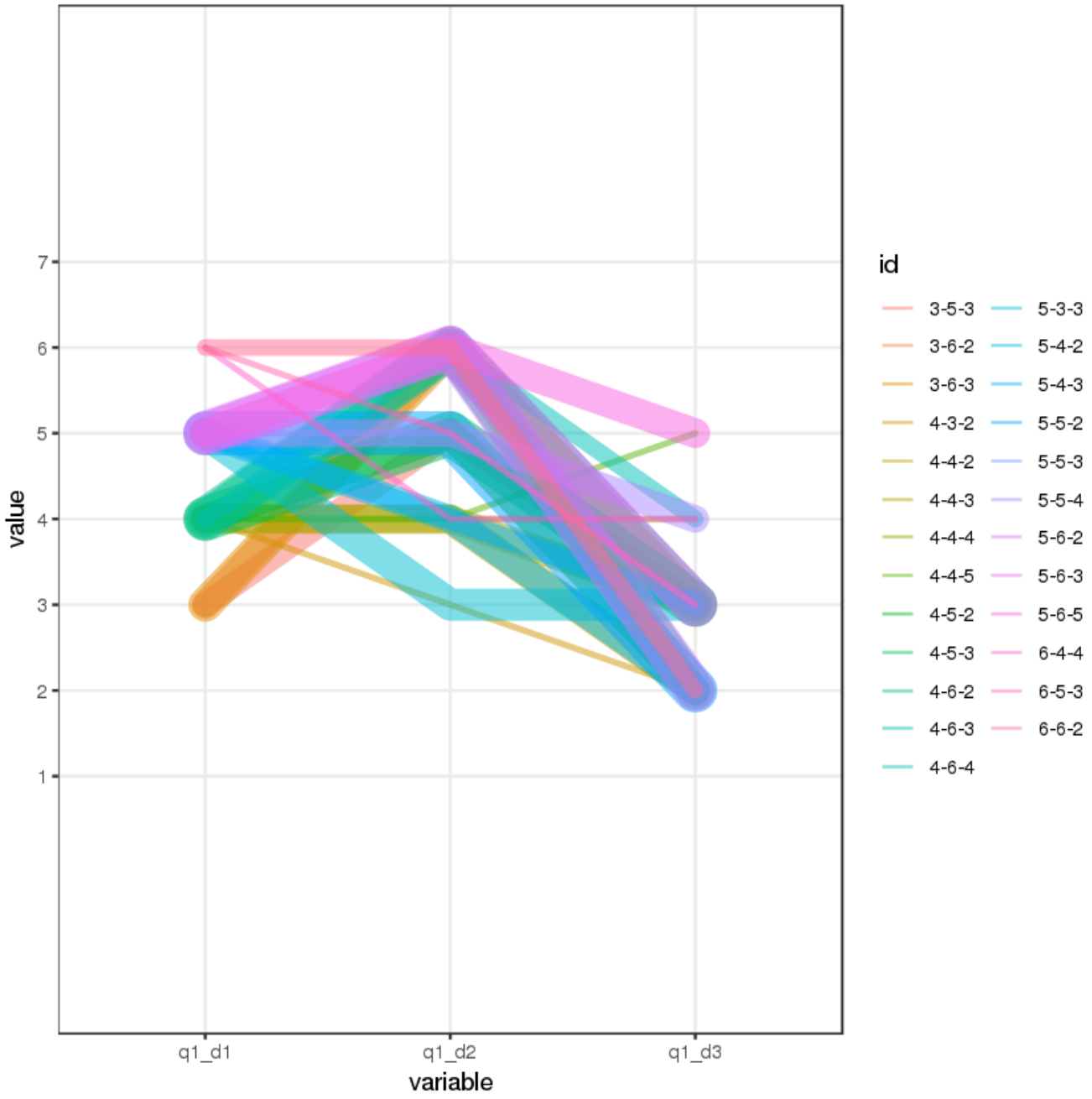

Figure 35. Parallel coordinate plots for randomly generated data.

Example 22 (Clusters)
It is possible to show the distinct clusters or groups using `geom_encircle()`. If the dataset has multiple "weak" features, we often first find the principal components and display the dataset as a scatterplot using PC_1 and PC_2 for the x and y axes, respectively.

The geom `geom_encircle()` can be used to encircle the desired groups. The only thing to note is the data argument to `geom_circle()` – a subsetted dataframe containing only the observations (rows) beloning to the group as the data argument.

```
# devtools::
    install_github("hrbrmstr/ggalt")
library(ggplot2)
library(ggalt)
library(ggfortify)
theme_set(theme_classic())
# Compute data with principal components
df <- iris[c(1, 2, 3, 4)]
pca_mod <- prcomp(df) # compute
    principal components
# Data frame of principal components
df_pc <- data.frame(pca_mod$x,
    Species=iris$Species) # dataframe of
    principal components
df_pc_vir <- df_pc[df_pc$Species ==
    "virginica", ] # df for 'virginica'
df_pc_set <- df_pc[df_pc$Species ==
    "setosa", ] # df for 'setosa'
df_pc_ver <- df_pc[df_pc$Species ==
    "versicolor", ] # df for 'versicolor'

# Plot
clustering<-ggplot(df_pc, aes(PC1, PC2,
    col=Species)) +
  geom_point(aes(shape=Species), size=2)
      + # draw points
  labs(title="Iris Clustering",
       subtitle="With principal components
           PC1 and PC2 as X and Y axis",
       caption="Source: Iris") +
  coord_cartesian(xlim = 1.2 *
      c(min(df_pc$PC1), max(df_pc$PC1)),
                ylim = 1.2 *
                    c(min(df_pc$PC2),
                    max(df_pc$PC2))) + #
                    change axis limits
  geom_encircle(data = df_pc_vir,
      aes(x=PC1, y=PC2)) + # draw circles
  geom_encircle(data = df_pc_set,
      aes(x=PC1, y=PC2)) +
  geom_encircle(data = df_pc_ver,
      aes(x=PC1, y=PC2))

ggsave(file="clusters.png",
    plot=clustering, width=5, height=4)
```

Example 23 (Dumbbell Plot)
Dumbbell charts are a great tool to visualize relative positions (like growth and decline) between two points in time, and compare distances between two categories.

In order to get the correct ordering of the dumbbells, the y—axis variable should be a factor and the levels of the factor variable have to be in the same order as they should appear in the plot.

```
#
    devtools::install_github("hrbrmstr/ggalt")
library(ggplot2)
library(ggalt)
theme_set(theme_classic())

health <-
    read.csv("https://raw.githubusercontent.com/
          selva86/datasets/master/health.csv")

# for right ordering of the dumbells
health$Area <- factor(health$Area,
    levels=as.character(health$Area))
# health$Area <- factor(health$Area)
gg <- ggplot(health, aes(x=pct_2013,
    xend=pct_2014, y=Area, group=Area)) +
    geom_dumbbell(color="#a3c4dc",
       size=0.75,
       point.colour.l="#0e668b") +
    scale_x_continuous(label=waiver()) +
    labs(x=NULL,
       y=NULL,
       title="Dumbbell Chart",
       subtitle="Pct Change: 2013 vs
           2014",
       caption="Source:
           https://github.com/hrbrmstr/ggalt")
           +
    theme(plot.title =
        element_text(hjust=0.5,
        face="bold"),
       plot.background=element_rect(
           fill="#f7f7f7"),
       panel.background=element_rect(
           fill="#f7f7f7"),
       panel.grid.minor=element_blank(),
       panel.grid.major.y=element_blank(),
       panel.grid.major.x=element_line(),
       axis.ticks=element_blank(),
       legend.position="top",
       panel.border=element_blank())
plot(gg)
```

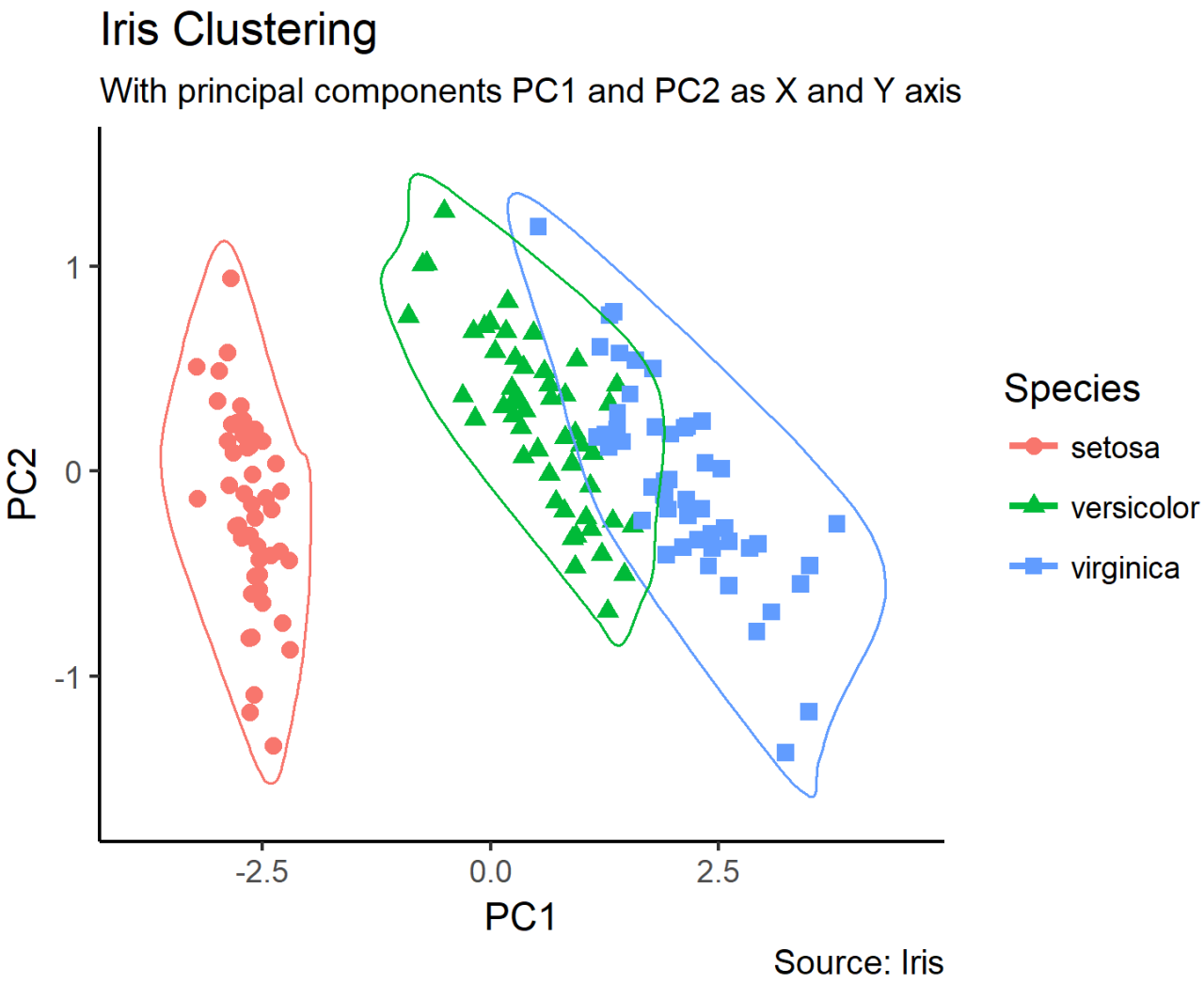

Figure 36. Clusters in the `iris` dataset, projected on the first 2 principal components.

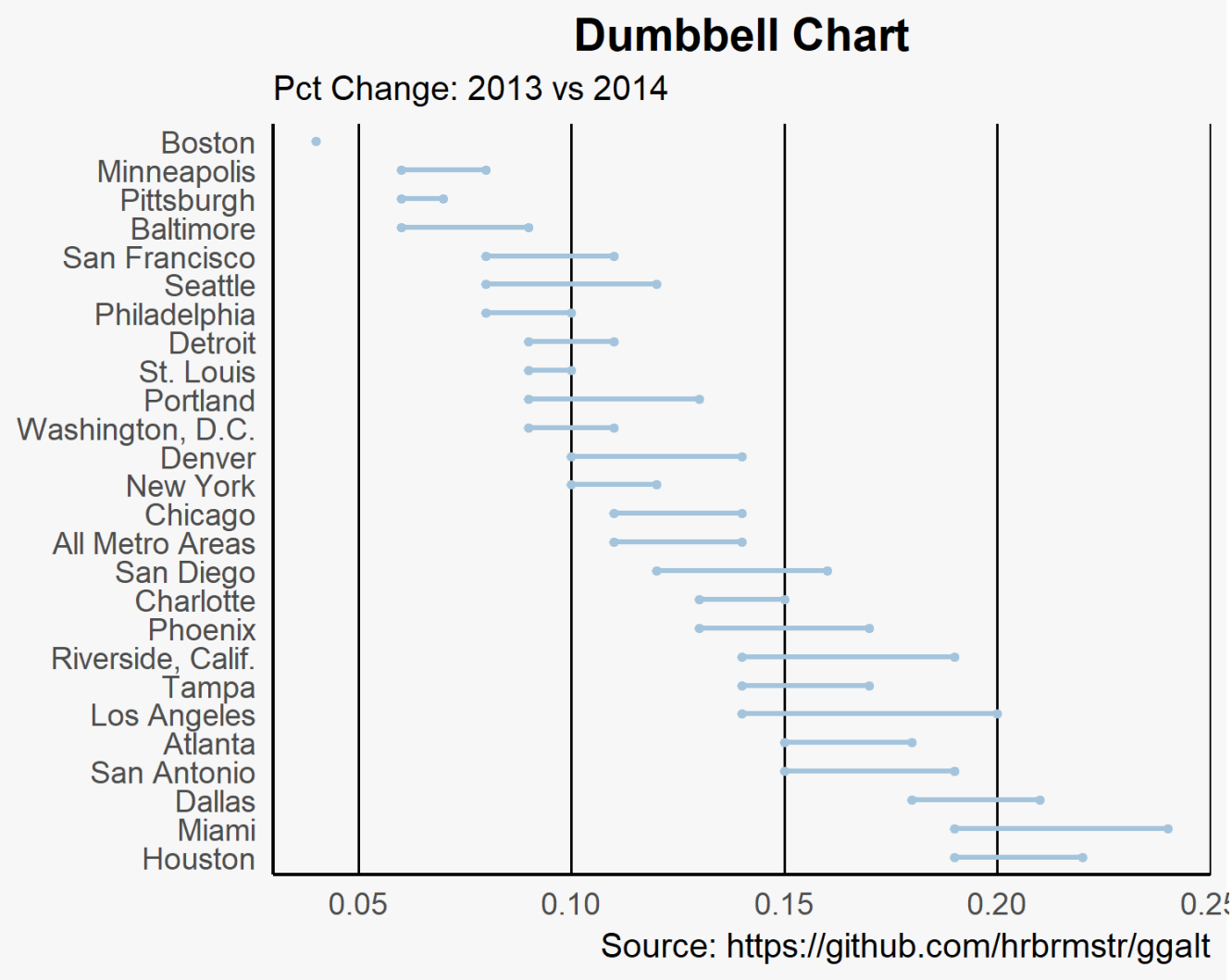

Figure 37. Dumbbell plot for the `health` dataset.

Example 24 (Slope Chart)

A slope chart is a great tool to visualize changes in value and ranking between categories. It is more suitable than a time series when very few time points are present.

```
library(dplyr)
theme_set(theme_classic())
source_df <- read.csv("
    https://raw.githubusercontent.com/
    jkeirstead/r-slopegraph/master/
        cancer_survival_rates.csv")

# Define functions. Source:
    https://github.com/jkeirstead/r-slopegraph
tufte_sort <- function(df, x="year",
    y="value", group="group",
    method="tufte", min.space=0.05) {
    ## First rename the columns for
        consistency
    ids <- match(c(x, y, group),
        names(df))
    df <- df[,ids]
    names(df) <- c("x", "y", "group")

    ## Expand grid to ensure every
        combination has a defined value
    tmp <- expand.grid(x=unique(df$x),
        group=unique(df$group))
    tmp <- merge(df, tmp, all.y=TRUE)
    df <- mutate(tmp, y=ifelse(is.na(y),
        0, y))

    ## Cast into a matrix shape and
        arrange by first column
    require(reshape2)
    tmp <- dcast(df, group ~ x,
        value.var="y")
    ord <- order(tmp[,2])
    tmp <- tmp[ord,]

    min.space <-
        min.space*diff(range(tmp[,-1]))
    yshift <- numeric(nrow(tmp))
    ## Start at "bottom" row
    ## Repeat for rest of the rows until
        you hit the top
    for (i in 2:nrow(tmp)) {
        ## Shift subsequent row up by
            equal space so gap between
        ## two entries is >= minimum
        mat <- as.matrix(tmp[(i-1):i, -1])
        d.min <- min(diff(mat))
        yshift[i] <- ifelse(d.min <
            min.space, min.space - d.min,
            0)}

    tmp <- cbind(tmp,
        yshift=cumsum(yshift))
    scale <- 1
    tmp <- melt(tmp, id=c("group",
        "yshift"), variable.name="x",
        value.name="y")
    ## Store these gaps in a separate
        variable so that they can be
        scaled ypos = a*yshift + y

    tmp <- transform(tmp, ypos=y +
        scale*yshift)
    return(tmp)

}
plot_slopegraph <- function(df) {
    ylabs <- subset(df,
        x==head(x,1))$group
    yvals <- subset(df, x==head(x,1))$ypos
    fontSize <- 3
    gg <- ggplot(df,aes(x=x,y=ypos)) +
        geom_line(aes(group=group),colour="grey80")
            +
        geom_point(colour="white",size=8) +
        geom_text(aes(label=y),
            size=fontSize,
            family="American Typewriter") +
        scale_y_continuous(name="",
            breaks=yvals, labels=ylabs)
    return(gg)
}

## Prepare data
df <- tufte_sort(source_df,
                 x="year",
                 y="value",
                 group="group",
                 method="tufte",
                 min.space=0.05)

df <- transform(df,
        x=factor(x, levels=c(5,10,15,20),
        labels=c("5 years","10 years","15
            years","20 years")),
        y=round(y))

## Plot
plot_slopegraph(df) +
    labs(title="Estimates of % survival
    rates") +
        theme(axis.title=element_blank(),
            axis.ticks = element_blank(),
            plot.title =
                element_text(hjust=0.5,
                family = "American
                    Typewriter",
                    face="bold"),
                axis.text =
                    element_text(family =
                    "American Typewriter",
                    face="bold"))
```

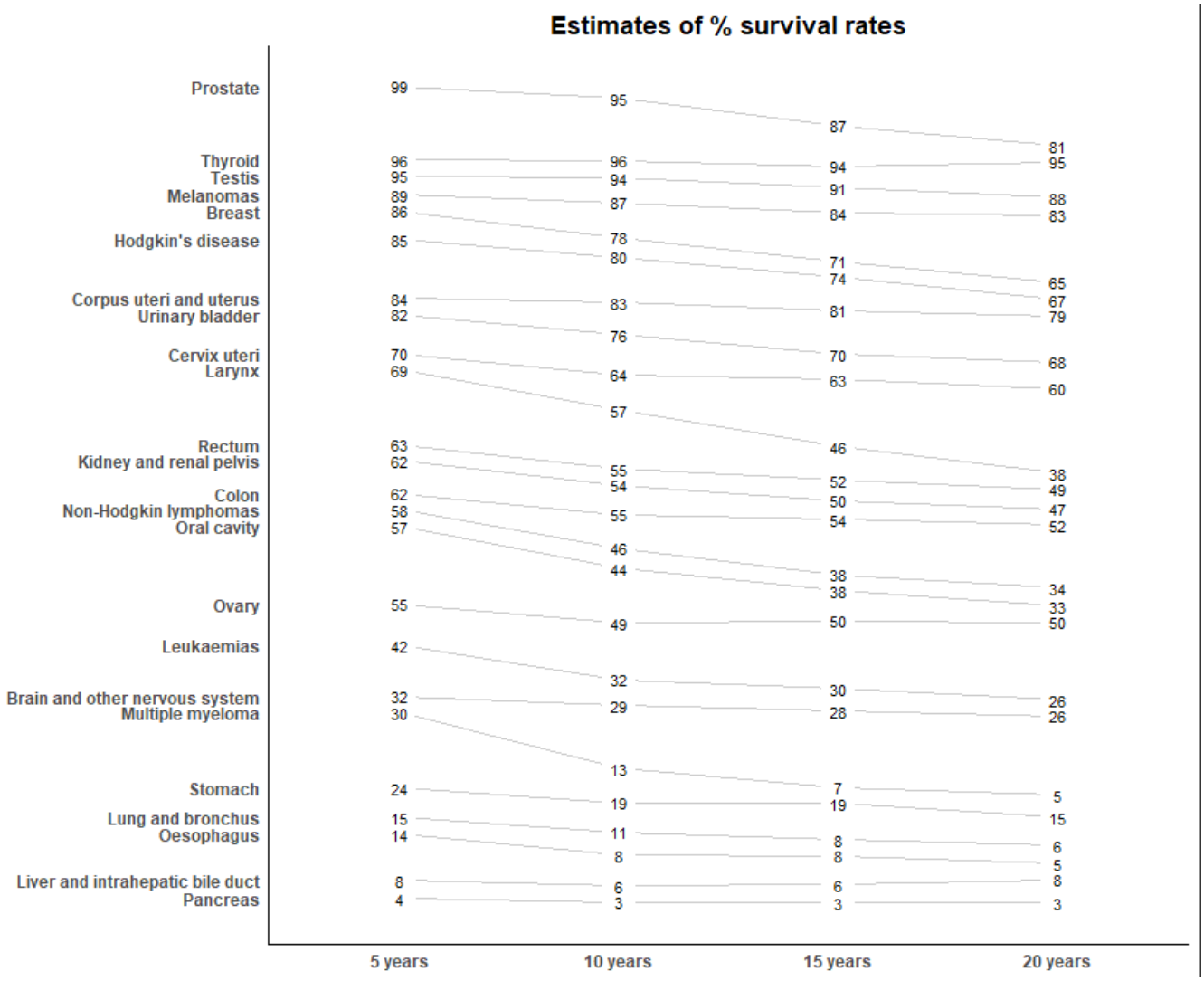

Figure 38. Slope chart for the `cancer survival rates` dataset.

Example 25 (Hierarchical Dendrogram)
A dendrogram is a tree-structured graph used to visualize the result of a hierarchical clustering calculation, *via* `ggdendrogram()` (see Figure 39).

```
#install.packages("ggdendro")
library("ggplot2")
library("ggdendro")
theme_set(theme_bw())
hc <- hclust(dist(USArrests), "ave") #
    hierarchical clustering
# plot
ggdendrogram(hc, rotate = TRUE, size = 2)
```

Example 26 (Density Plot)
Density plots can be viewed as a smoothed histograms (see Figure 40).

```
library(ggplot2)
theme_set(theme_classic())

# Plot
g <- ggplot(mpg, aes(cty))
g + geom_density(aes(fill=factor(cyl)),
    alpha=0.8) +
    labs(title="Density plot",
        subtitle="City Mileage Grouped by
            Number of cylinders",
        caption="Source: mpg",
        x="City Mileage",
        fill="# Cylinders")

h <- ggplot(mpg, aes(cty))
h +
    geom_density(aes(x=cty,fill=factor(cyl)),
    alpha=0.8) + facet_wrap(~cyl) +
    labs(title="Density plot",
        subtitle="City Mileage by Number
            of cylinders",
        caption="Source: mpg",
        x="City Mileage",
        fill="# Cylinders")
```

Example 27 (Box Plot)
Boxplots are an excellent tool to study a univariate distribution. It can also be used to show the distribution within multiple groups, along with the median, range, and suspected outliers (assuming the underlying distribution is normal).

The dark line inside the box represents the median. The top of box is the 75th percentile (the 3rd quartile) and the is the 25th percentile (the 1st quartile). The end points of the lines (the whiskers) are plotted at a distance of 1.5 × the interquartie range (3rd quartile - 1st quartile). The points outside the whiskers are marked as dots and are normally considered as extreme points.

Setting `varwidth=T` in the `geom_boxplot` geom adjusts the width of the boxes to be proportional to the number of observation it contains (see Figure 41).

```
library(ggplot2)
theme_set(theme_classic())

# Plot
g <- ggplot(mpg, aes(class, cty))
g + geom_boxplot(varwidth=T,
    fill="plum") +
    labs(title="Box plot",
        subtitle="City Mileage grouped by
            Class of vehicle",
        caption="Source: mpg",
        x="Class of Vehicle",
        y="City Mileage")
```

Example 28 (Dot + Box Plot)
On top of the information provided by a box plot, the dot plot can provide more clear information in the form of summary statistics by each group. The dots are staggered such that each dot represents one observation. In Figure 42 the number of dots for a given manufacturer will match the number of rows of that manufacturer in source data.

```
library(ggplot2)
theme_set(theme_bw())

# plot
g <- ggplot(mpg, aes(manufacturer, cty))
g + geom_boxplot() +
  geom_dotplot(binaxis='y',
            stackdir='center',
            dotsize = .5,
            fill="red") +
  theme(axis.text.x =
      element_text(angle=65, vjust=0.6)) +
  labs(title="Box plot + Dot plot",
      subtitle="City Mileage vs Class:
          Each dot represents 1 row in
          source data",
      caption="Source: mpg",
      x="Class of Vehicle",
      y="City Mileage")
```

Example 29 (Waffle Chart)
Waffle charts provide a nice way to show the categorical composition in the overall population. There is no direct waffle chart geom, but they can be produced using `geom_tile()`, as shown below (result in Figure 43).

```
library(ggplot2)
var <- mpg$class # the categorical data
nrows <- 10
df <- expand.grid(y = 1:nrows, x =
    1:nrows)
```

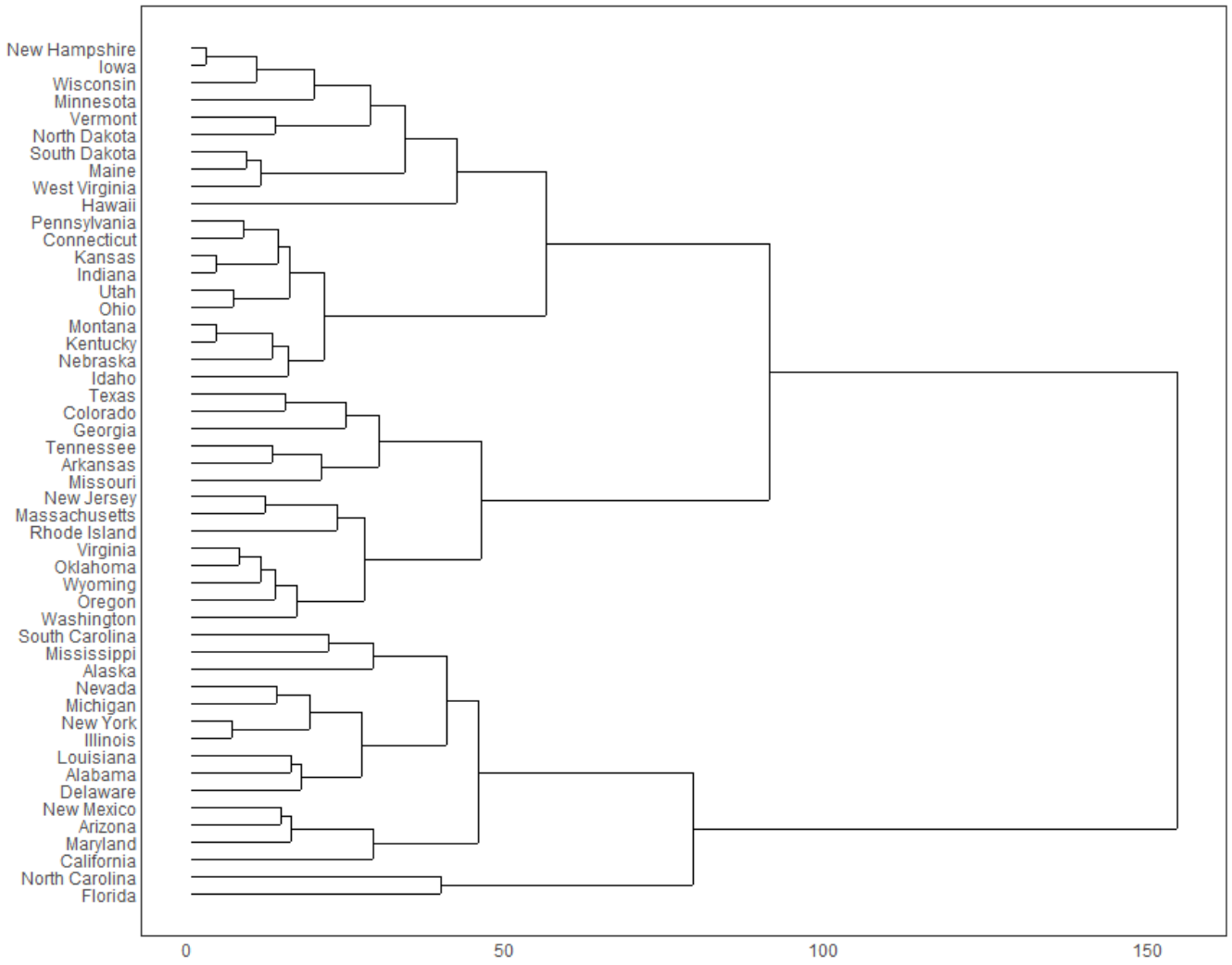

Figure 39. Hierarchical dendrogram for the `USArrests` dataset.

```
categ_table <- round(table(var) *
    ((nrows*nrows)/(length(var))))
# categ_table
df$category <-
    factor(rep(names(categ_table),
    categ_table))
# NOTE: if sum(categ_table) is not 100
    (i.e. nrows^2), it will need
    adjustment to make the sum to 100.
## Plot
ggplot(df, aes(x = x, y = y, fill =
    category)) +
       geom_tile(color = "black", size =
           0.5) +
       scale_x_continuous(expand = c(0,
           0)) +
       scale_y_continuous(expand = c(0,
           0), trans = 'reverse') +
       scale_fill_brewer(palette =
           "Set3") +
       labs(title="Waffle Chart",
           subtitle="'Class' of vehicles",
           caption="Source: mpg") +
       theme(panel.border =
           element_rect(size = 2),
           plot.title =
               element_text(size =
               rel(1.2)),
           axis.text = element_blank(),
           axis.title = element_blank(),
           axis.ticks = element_blank(),
           legend.title =
               element_blank(),
           legend.position = "right")
```

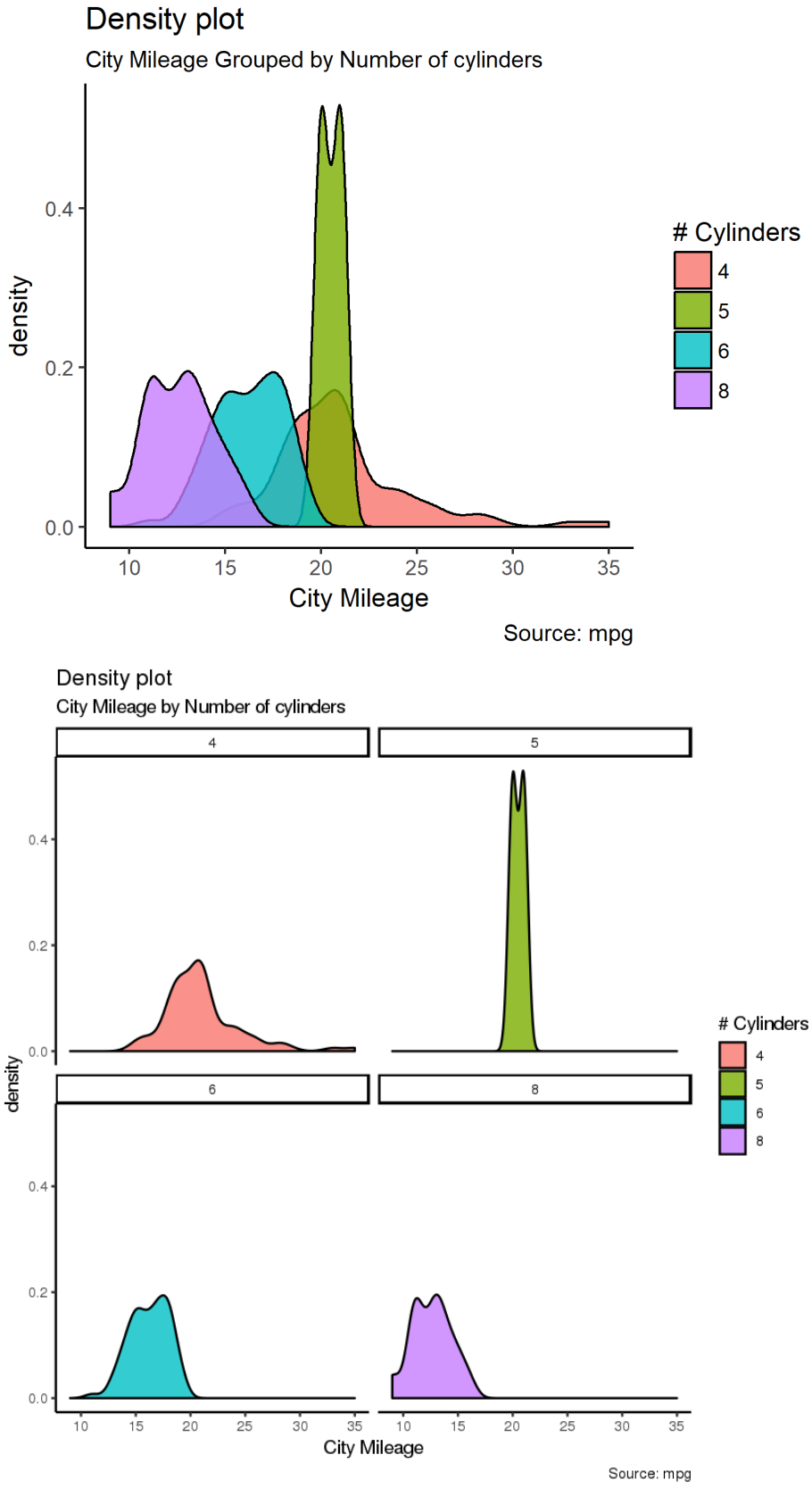

Figure 40. Density plot for the `mpg` dataset; simultaneous (top), faceted (bottom).

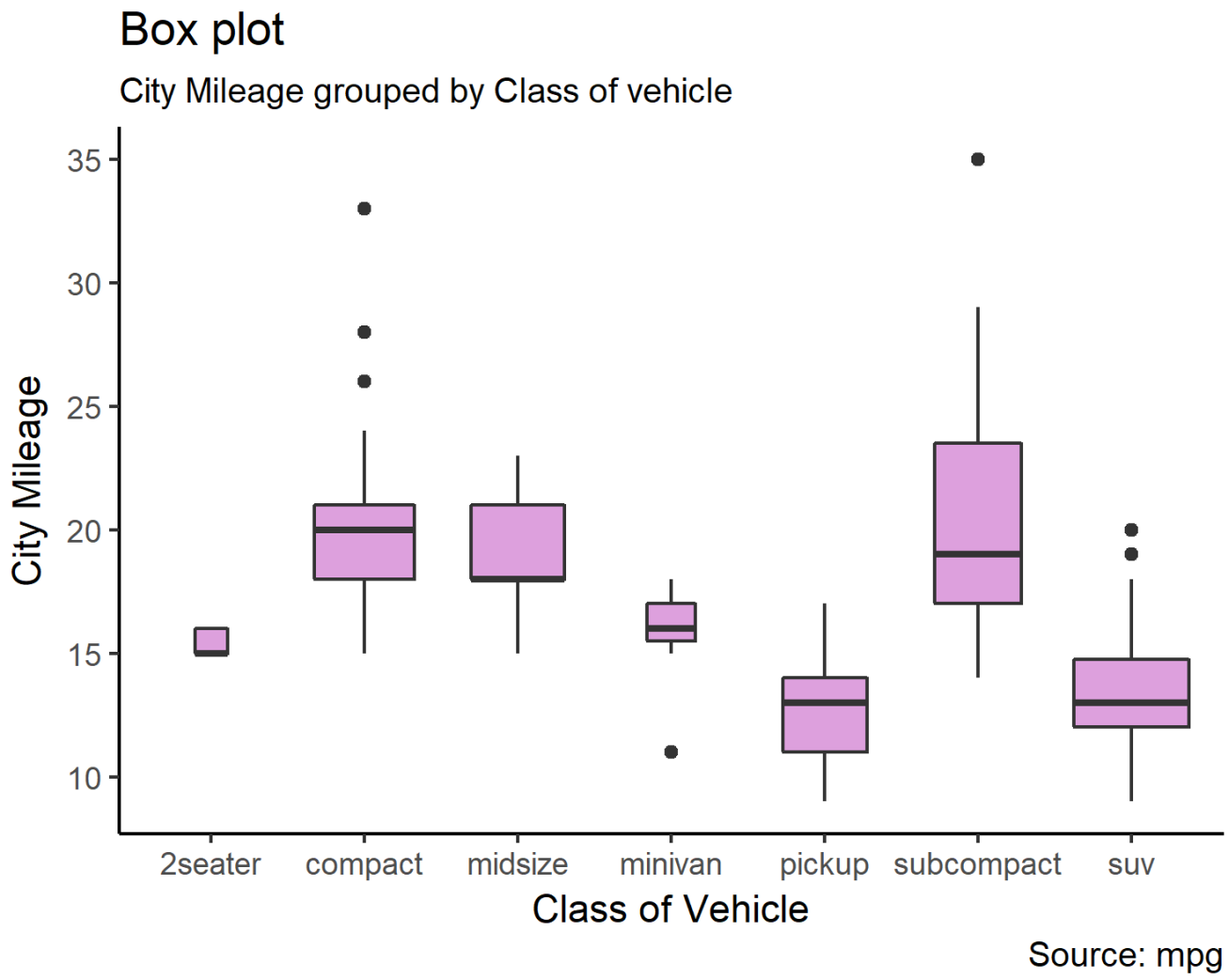

Figure 41. Boxplots of the `mpg` dataset.

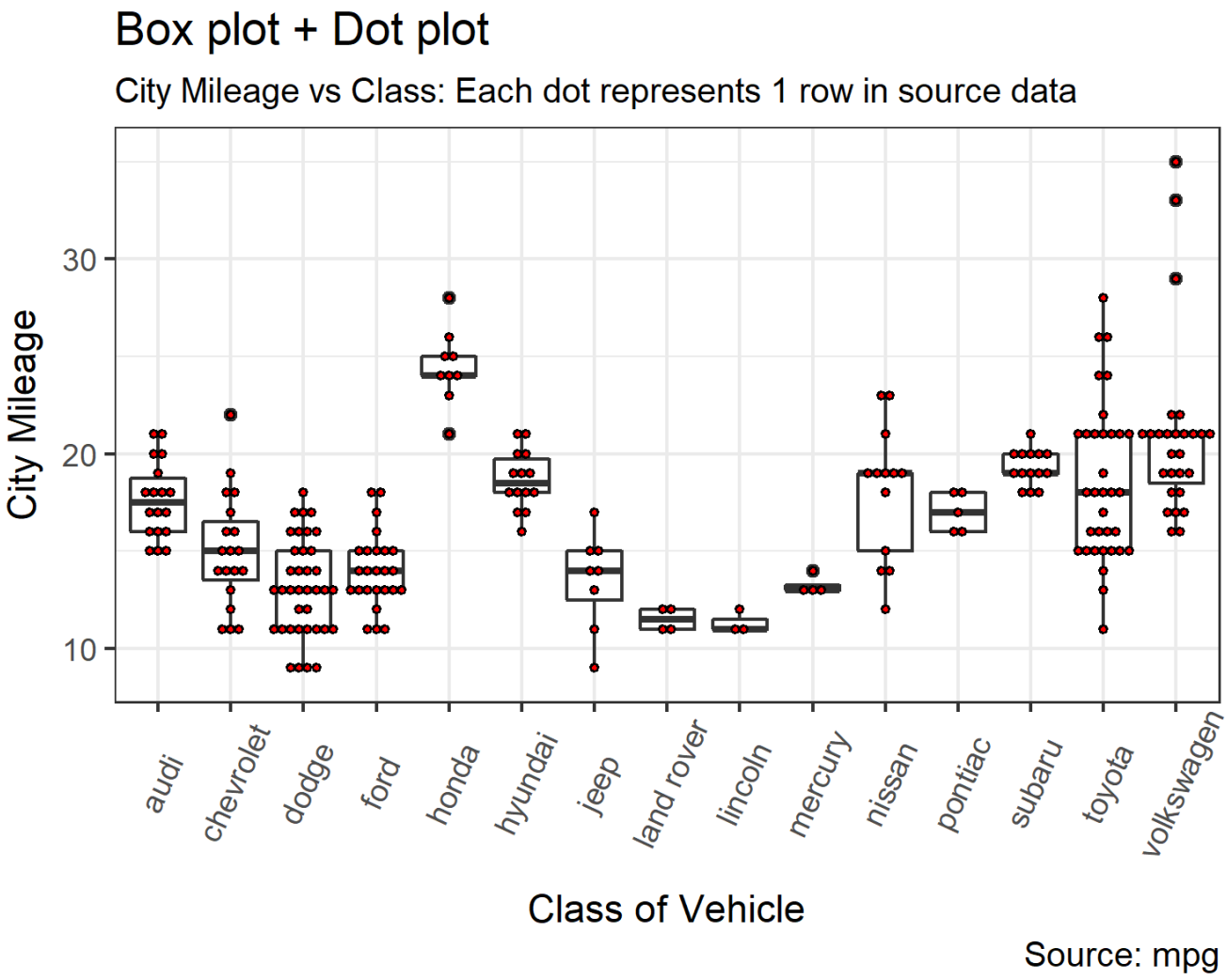

Figure 42. Dot boxplot of the `mpg` dataset.

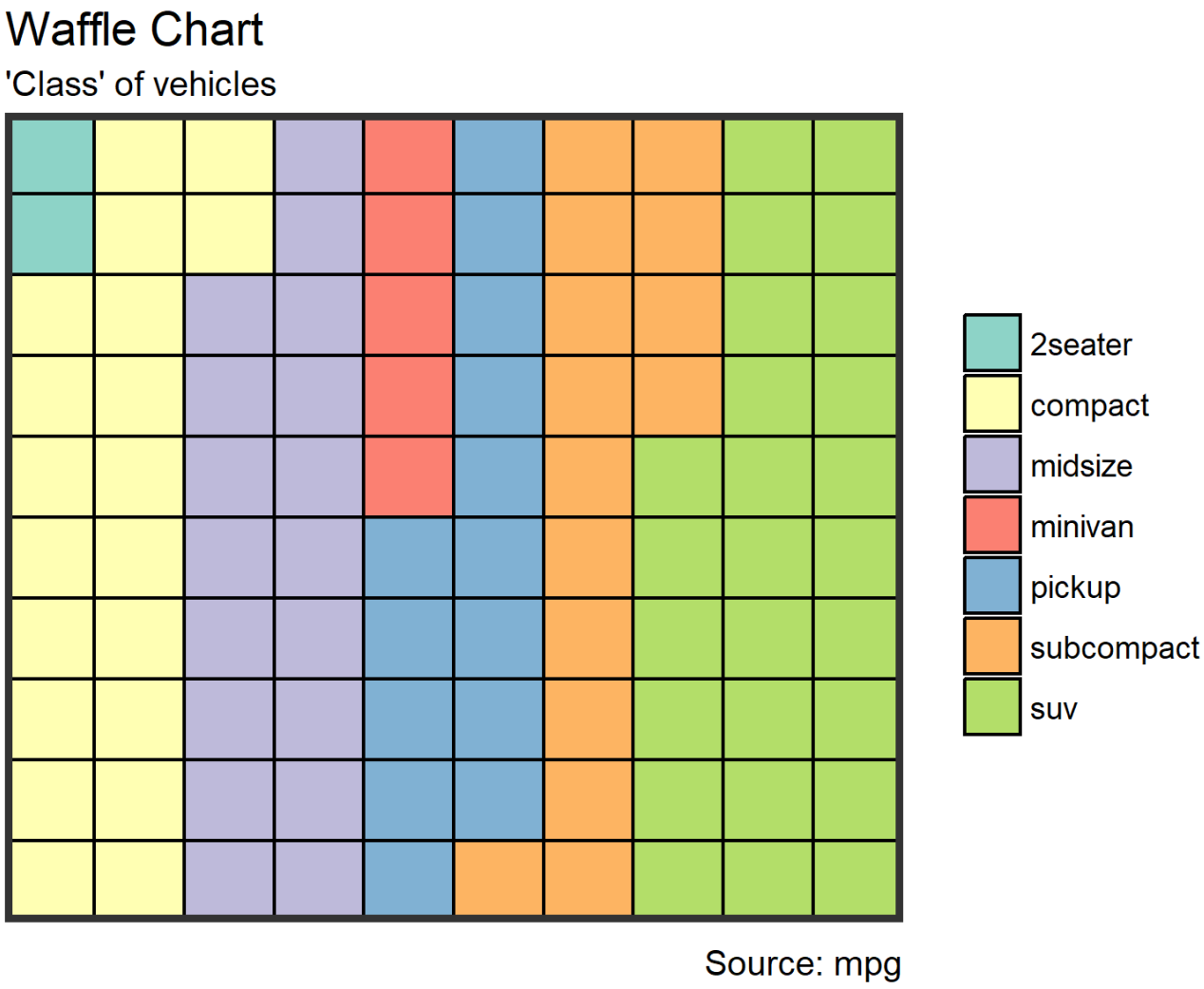

Figure 43. Waffle chart of the `mpg` dataset.

Example 30 (Text Visualization)
In the following example, we will process the subtitles of *The Green Mile*, saved in a plain text (`.txt`) file.

```
# Load
library("tm")
library("SnowballC")
library("wordcloud")
library("RColorBrewer")
library("ggplot2")
```

Loading the text: The text is loaded using `Corpus()` from the *tm* (text mining) package. A corpus is a list of a documents (in this case, a single document). Typically, this would be done using code such as:

```
text <- readLines(file.choose())
# Read the text file from internet
#filePath <- "http://..."
#text <- readLines(filePath)
```

The corpus can be inspected using:

```
# Load the data as a corpus
docs <- Corpus(VectorSource(text))
inspect(docs)
```

Text transformation: text processing is performed using various `tm_map()` calls to replace, for instance, the special characters "/", "@" and "|" with a blank space.

```
toSpace <- content_transformer(function
    (x , pattern ) gsub(pattern, " ", x))
docs <- tm_map(docs, toSpace, "/")
docs <- tm_map(docs, toSpace, "@")
docs <- tm_map(docs, toSpace, "\\|")
```

Cleaning the text: the `tm_map()` function can also be used to remove unnecessary white spaces, to convert the text to lower case, to remove common stopwords like "the" , or "we".

The information content of these stopwords is basically nil due to the fact that they are used so commonly in a given language. Removing such terms simplifies the final analysis (there are numerous supported language, whose names are case-sensitive). Numbers and punctuation can also be removed with `removeNumbers` and `removePunctuation` arguments.

Another important pre-processing step is to stem words to reduce them to their root form. This process removes word suffixes to get the common origin. For example, "moving", "moved" and "movement" would all be stemmed to the root word "move" (stemming requires the package *SnowballC*).

```
# Convert the text to lower case
docs <- tm_map(docs,
    content_transformer(tolower))
# Remove numbers
docs <- tm_map(docs, removeNumbers)
# Remove english common stopwords
docs <- tm_map(docs, removeWords,
    stopwords("english"))
# Remove your own stop word
# specify your stopwords as a character
    vector
docs <- tm_map(docs, removeWords,
    c("blabla1", "blabla2"))
# Remove punctuations
docs <- tm_map(docs, removePunctuation)
# Eliminate extra white spaces
docs <- tm_map(docs, stripWhitespace)
# Text stemming
# docs <- tm_map(docs, stemDocument)
```

Building a term-document matrix: a tdf is a table containing the frequency of the words per document. Column names are words (or terms) and row names are documents. The function `TermDocumentMatrix()` can be used as follow :

```
dtm <- TermDocumentMatrix(docs)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)
```

Generating a word cloud: The relative importance of words can be illustrated *via* a word cloud.

```
set.seed(1234)
wordcloud(words = d$word, freq = d$freq,
    min.freq = 1,
        max.words=200,
            random.order=FALSE,
            rot.per=0.35,
        colors=brewer.pal(8, "Dark2"))
```

Plotting: *ggplot2* can be used to provide bar plots of the most frequent words

```
p <- ggplot(subset(d, freq>30), aes(x =
    reorder(word, -freq), y = freq)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x=element_text(angle=45,
      hjust=1))
p
```

A word cloud and a bar plot for *The Green Mile* are shown in Figures 44 and 45.

Figure 44. Word cloud for *The Green Mile* (English subtitles).

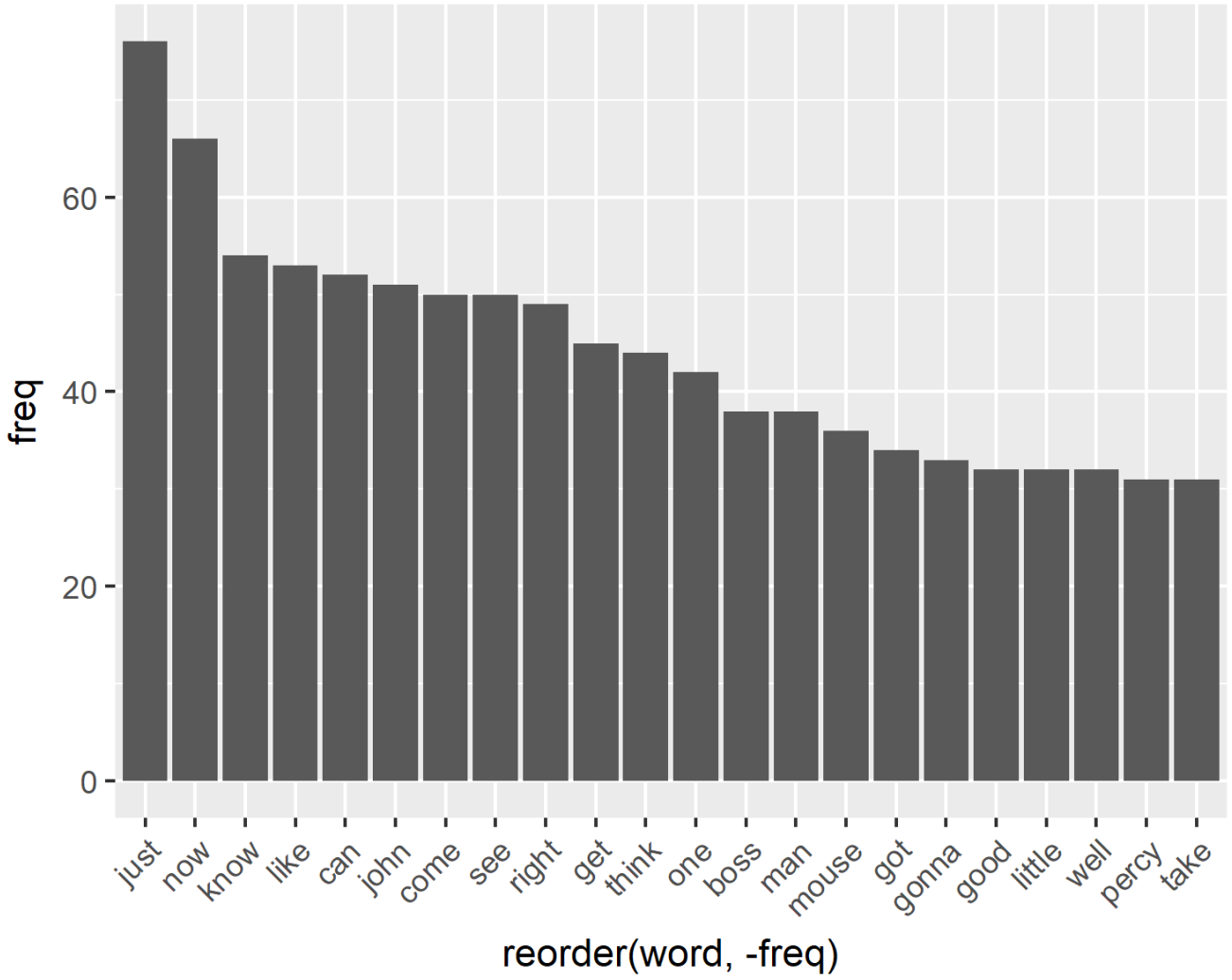

Figure 45. Bar chart for *The Green Mile* (English subtitles).

References

[1] Chang, W. [2013], *R Graphics Cookbook*, O'Reilly.

[2] Wickham, H. [2009], *ggplot2: Elegant Graphics for Data Analysis*, Springer.

[3] Wickham, H. [2009], *A Layered Grammar of Graphics*. Journal of Computational and Graphical Statistics 19:3–28.

[4] Horton, N.J., Kleinman, K. [2016], *Using R and RStudio for Data Management, Statistical Analysis, and Graphics*, 2nd ed., CRC Press.

[5] Healey, K. [2018], **Data Visualization: A Practical Introduction**.

[6] Kabacoff, R.I. [2011], *R in Action, Second Edition: Data analysis and graphics with R*, Live.

[7] Maindonald, J.H. [2008], **Using R for Data Analysis and Graphics Introduction, Code and Commentary**.

[8] Tyner, S., Briatte, F., Hofmann, H. [2017], **Network Visualization with *ggplot2***, The R Journal, vol. 9(1).

[9] Broman, K. [2016], **Data Visualization with *ggplot2***.

[10] ***ggplot2* Extensions.**

[11] **R Graph Gallery**.

[12] Anderson, S.C. [2015], **An Introduction to *ggplot2***.

[13] Prabhakaran, S., **Top-50 *ggplot2* Visualization (with Master List R Code)**.

[14] Konrad, M. [2016], **Parallel Coordinate Plots for Discrete and Categorical Data in R: A Comparison**.

[15] Wilkins, D., **treemapify** github repository.

[16] Wilkins, D., **treemapify R package (v. 0.2.1)**.

[17] STHDA, **Beautiful Dendrogram Visualizations in R: 5+ must-know methods - Unsupervised Machine Learning**.

[18] **Text Mining and Word Cloud Fundamentals in R: 5 simple steps you should know**, on Easy Guides.

[19] Harvard tutorial notes, **R graphics with *ggplot2* workshop**.

[20] Robinson, D., **Visualizing Data Using ggplot2**, on varianceexplained.org.

[21] **Manipulating, analyzing and exporting data with tidyverse**, on datacarpentry.org.

[22] Wickham, H. [2014], Tidy Data, *Journal of Statistical Software*, v59, n10.