Cédric Scherer

Graphic Design with ggplot2

Create Beautiful Data Visualizations in R

To my son,

without whom I should have finished this book two years earlier

Contents

List of Tables

List of Figures

Preface

Back in 2016, I had to prepare my PhD introductory talk to inform about my plans for the next three years and to showcase my first preliminary results. I planned to create a visualization using small multiples to show various outcomes of the scenarios I ran with my simulation model. I was already using the R programming language for years and quickly came across the graphics library **ggplot2** which comes with the functionality to easily create small multiples.to visualize my data. I never liked the syntax and style of base plots in R, so I immediately fell in love with the idea and implementation of **ggplot2**'s *Grammar of Graphics*. But because I was short on time, I plotted these figures by trial and error and with the help of lots of googling. The resource I came always back to was a blog entry called "Beautiful plotting in R: A **ggplot2** cheatsheet" by Zev Ross[1]. After giving the talk which contained some decent plots thanks to the blog post, I decided to go through this tutorial step-by-step. I learned so much from it and directly started modifying the codes and adding additional code snippets, chart types, and resources.

Fast forward to 2019. I successfully finished my PhD and started participating in a weekly data visualization challenge called #TidyTuesday[2].Every week, a raw data set is shared with the aim to explore and visualize the data with **ggplot2**. Thanks to my experience with the **tidyverse** and especially **ggplot2** during my PhD and the open-source approach of the challenge that made it possible to learn from other participants, my visualizations quickly became more advanced and complex.

A few months later, I had built a portfolio ofvarious charts and maps and decided to start working as an independent data visualization specialist. I am now using **ggplot2** every day: for my academic work, design requests, reproducible reports, educational purposes, and personal data visualization projects. What I especially love about my current job specification: It challenges and satisfies my creativity on different levels. Beside the creativity one can express in terms of chart choice and design, there is also creativity needed to come up with solutions and tricks to bring the most venturous ideas to life. At the same time, there is the gratification when your code works and *magically* translates code snippets to visuals.

[1] http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/
[2] https://github.com/rfordatascience/tidytuesday/blob/master/README.md

The blog entry by Zev Ross was not updated since January 2016, so I decided to add more examples and tricks to my version, which was now hosted on my personal blog[3]. It became step by step a unique tutorial that now contains also the fantastic **patchwork**, **ggtext** and **ggforce** packages, a section of custom fonts and colors, a collection of R packages tailored to create interactive charts, and several new chart types. The updated version now contains ~3.000 lines of code and 188 plots and received a lot of interest from **ggplot2** users from a wide range of fields.

Today, on a sunny day in July 2021, this tutorial serves as the starting point for the book you hold in your hands. I hope you enjoy it as much as I enjoyed learning and sharing **ggplot2** wizardry!

Why read this book

Often, people that use common graphic design and charting tools or have basic experience experience with **ggplot2** cannot believe what one can achieve with this graphics library—and I want to show you how one can create a publication-ready graphic that goes beyond the traditional scientific scatter or box plot.

ggplot2 is already used by a large and diverse group of graduates, researchers, and analysts and the current rise of R and the tidyverse will likely lead to an even increasing interest in this great plotting library. While there are many tutorials on **ggplot2** tips and tricks provided by the R community, to my knowledge there is no book that specifically addresses the complete design of specific details up to building an ambitious multi-panel graphic with **ggplot2**. As a blend of strong grounding in academic foundations of data visualization and hands-on, practical codes, and implementation material, the book can be used as introductory material as well as a reference for more experienced **ggplot2** practitioners.

The book is intended for students and professionals that are interested in learning **ggplot2** and/or taking their default ggplots to the next level. Thus, the book is potentially interesting for **ggplot2** novices and beginners, but hopefully also helpful and educational for proficient users.

Among other things, the book covers the following:

- Look-up resource for every-day and more specific ggplot adjustments and design options
- Practical hands-on introduction to **ggplot2** to quickly build appealing visualization

[3]https://www.cedricscherer.com/2019/08/05/a-ggplot2-tutorial-for-beautiful-plotting-in-r/

- Discussion of best practices in data visualization (e.g. color choice, direct labelling, chart type selection) along the way
- Coverage of useful **ggplot2** extension packages
- Ready-to-start code examples
- Reference implementations illustrating code solutions and design choices

Why R and ggplot2

As a computational ecologist, I've learned and used a range of different tools and programming languages for various purposes such as data wrangling, statistical analyses, and model building. The open-source language R was and is the programming language most widely used by ecologists to handle and analyze ecological data (Sciaini et al., 2018). Consequently, I was *of course* using R in my daily life as an scientific researcher.

Nowadays, R plays crucial part in many data-related workflows, no matter if for scientific, educational, or industrial use cases. Thanks to the ever growing R community and the rich collection of libraries that add additional functionality and simplify workflows, R is an attractive programming language that has outgrown of its original purpose: statistical analyses. Today, R can serve as tool to generate automated reports, develop stand-alone web apps, and draft presentation slides, books, and web pages. And to design high-level, publication–ready visualizations.

Even though R—similar to most programming languages—has a steep learning curve, the level of functionality, flexibility, automation, and reproducibility offered can be a major benefit also in a design context:

- The layered approach of **ggplot2** opens the possibility to build any type of visualisation.
- Various extension packages add missing functionality.
- Script-based workflows instead of *point–and–click* approaches allow for reproducibility—which means you can simply (in theory) run the code again after receiving new data or create thousands of visualizations for various data sets in no time.
- Sharing code is becoming the golden standard in many fields and thus facilitates transparency and credibility as well as modification and creative advancement.
- A helpful community and many free resources simplify learning experiences and the search for solutions.
- The visualisations created in R can be exported as vector files and thus allow for post–processing with graphic design tools.

Structure of the book

Chapters 1 introduces a new topic, and ...

Software information and conventions

The book was written with the **knitr** package (Xie, 2015) and the **bookdown** package (Xie, 2021) with the following setup:

```
xfun::session_info()
```

```
## R version 4.1.0 (2021-05-18)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19043)
##
## Locale:
##   LC_COLLATE=German_Germany.1252
##   LC_CTYPE=German_Germany.1252
##   LC_MONETARY=German_Germany.1252
##   LC_NUMERIC=C
##   LC_TIME=German_Germany.1252
## system code page: 65001
##
## Package version:
##   base64enc_0.1.3   bookdown_0.22
##   compiler_4.1.0    digest_0.6.27
##   evaluate_0.14     glue_1.4.2
##   graphics_4.1.0    grDevices_4.1.0
##   highr_0.9         htmltools_0.5.1.1
##   jsonlite_1.7.2    knitr_1.33
##   magrittr_2.0.1    markdown_1.1
##   methods_4.1.0     mime_0.11
##   rlang_0.4.11      rmarkdown_2.9
##   rstudioapi_0.13   stats_4.1.0
##   stringi_1.7.3     stringr_1.4.0
##   tinytex_0.32      tools_4.1.0
##   utils_4.1.0       xfun_0.24
##   yaml_2.2.1
```

Package names are in bold text (e.g., **ggplot2**), and inline code and filenames are formatted in a monospaced typewriter font (e.g., `readr::read_csv('data.csv')`). Function names are followed by parentheses (e.g., `ggplot2::ggplot()`).

Acknowledgments

Thanks to David Grubbs, Alberto Cairo, Emily Riederer, Oscar Baruffa, and Malcolm Barrett for all your constructive feedback.

Cédric Scherer
Berlin, Germany

About the Author

Cédric Scherer is a graduated computational ecologist with a passion for good design. In 2020, he combined his expertise in analyzing and visualizing large data sets in R with his interest in design and his perfectionism to become a freelance data visualization specialist.

Cédric has created visualizations across all disciplines, purposes, and styles. Due to regular participation to social data challenges, he is now well known for complex and visually appealing figures, entirely made with ggplot2, that look as they have been created with a vector design tool.

As a data visualization specialist, he does rarely create dashboards but acts as a consultant and designer improving chart and design choices and workshop coach teaching data visualization principles and courses on R and **ggplot2**. He also uses R and the **tidyverse** packages to automate data analyses and plot generation, following the philosophy of a reproducible workflow.

1
Introduction

2 Setup

2.1 The Dataset

We are using data from the *National Morbidity and Mortality Air Pollution Study* (NMMAPS). To make the plots manageable we are limiting the data to Chicago and 1997–2000. For more detail on this data set, consult Roger Peng's book Statistical Methods in Environmental Epidemiology with R[1]. You can download the data we are using during this tutorial here: `https://cedricscherer.com/data/chicago-nmmaps.csv` (but you don't have to).

We can import the data into our R session for example with `read_csv()` from the **readr** package which allows to import data directly from a web URL. To access the data later, we are storing it in a variable called `chic` by using the *assignment arrow* `<-`.

```
url_data <- "https://cedricscherer.com/data/chicago-nmmaps.csv"
chic <- readr::read_csv(url_data)
```

```
##
## -- Column specification -------------------------------
## cols(
##   city = col_character(),
##   date = col_date(format = ""),
##   death = col_double(),
##   temp = col_double(),
##   dewpoint = col_double(),
##   pm10 = col_double(),
##   o3 = col_double(),
##   time = col_double(),
##   season = col_character(),
##   year = col_double()
## )
```

Note: The `::` *is called "namespace" and can be used to access a function*

[1] http://www.springer.com/de/book/9780387781662

without loading the package. Here, you could also run `library(readr)` first and `chic <- read_csv(...)` afterwards.

```
tibble::glimpse(chic)
```

```
## Rows: 1,461
## Columns: 10
## $ city     <chr> "chic", "chic", "chic", "chic", "chi~
## $ date     <date> 1997-01-01, 1997-01-02, 1997-01-03,~
## $ death    <dbl> 137, 123, 127, 146, 102, 127, 116, 1~
## $ temp     <dbl> 36.0, 45.0, 40.0, 51.5, 27.0, 17.0, ~
## $ dewpoint <dbl> 37.500, 47.250, 38.000, 45.500, 11.2~
## $ pm10     <dbl> 13.052, 41.949, 27.042, 25.073, 15.3~
## $ o3       <dbl> 5.659, 5.525, 6.289, 7.538, 20.761, ~
## $ time     <dbl> 3654, 3655, 3656, 3657, 3658, 3659, ~
## $ season   <chr> "Winter", "Winter", "Winter", "Winte~
## $ year     <dbl> 1997, 1997, 1997, 1997, 1997, 1997, ~
```

2.2 The ggplot2 Package

When looking into the package description of the **ggplot2** package, it states the following:

> **ggplot2** is a system for declaratively creating graphics, based on The Grammar of Graphics[2]. You provide the data, tell **ggplot2** how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Consequently, a ggplot is built up from a few basic elements:

1. **Data**:
   The raw data that you want to plot.
2. **Geometries** `geom_`:
   The geometric shapes that will represent the data.

[2] https://link.springer.com/chapter/10.1007/978-3-642-21551-3_13

3. **Aesthetics** `aes()`:
   Aesthetics of the geometric and statistical objects, such as position, color, size, shape, and transparency
4. **Scales** `scale_`:
   Maps between the data and the aesthetic dimensions, such as data range to plot width or factor values to colors.
5. **Statistical transformations** `stat_`:
   Statistical summaries of the data, such as quantiles, fitted curves, and sums.
6. **Coordinate system** `coord_`:
   The transformation used for mapping data coordinates into the plane of the data rectangle.
7. **Facets** `facet_`:
   The arrangement of the data into a grid of plots.
8. **Visual themes** `theme()`:
   The overall visual defaults of a plot, such as background, grids, axes, default typeface, sizes and colors.

Note: The number of elements may vary depending on how you group them and whom you ask.

2.3 A Basic ggplot

First, to be able to use the functionality of **ggplot2** we have to load the package (which we can also load via the tidyverse package collection[3]):

```
library(tidyverse)
```

```
## -- Attaching packages -------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ----------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

The syntax of **ggplot2** is different from base R. In accordance with the basic elements, a default ggplot needs three things that you have to specify: the

[3]https://www.tidyverse.org/

data, *aesthetics*, and a *geometry*. We always start to define a plotting object by calling `ggplot(data = df)` which just tells **ggplot2** that we are going to work with that data. In most cases, you might want to plot two variables—one on the x and one on the y axis. These are *positional aesthetics* and thus we add `aes(x = var1, y = var2)` to the `ggplot()` call (yes, the `aes()` stands for aesthetics). However, there are also cases where one has to specify onkly one or even three or more variables.

Note: We specify the data outside `aes()` and add the variables that ggplot maps the aesthetics to inside `aes()`.

Here, we map the variable `date` to the x position and the variable `temp` to the y position. Later, we will also map variables to all kind of other aesthetics such as color, size, and shape.

```
(g <- ggplot(chic, aes(x = date, y = temp)))
```

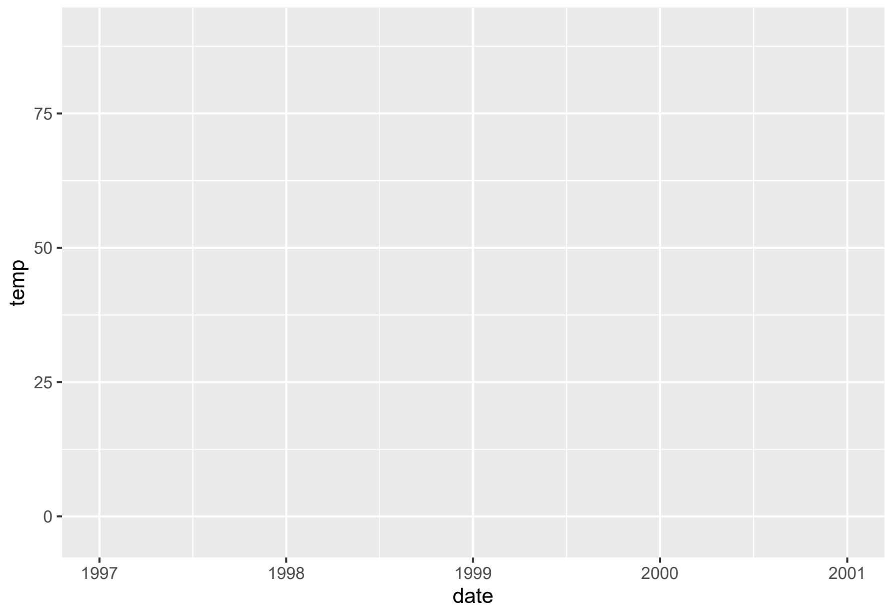

FIGURE 2.1: A default plot generated with the **ggplot2** package with variables mapped to `x` and `y`.

Hm, only a panel is created when running this. Why? This is because **ggplot2** does not know *how* we want to plot that data—we still need to provide a geometry!

ggplot2 allows you to store the current `ggobject` in a variable of your choice by assigning it to a variable, in our case called `g`. You can extend this `ggobject` later by adding other layers, either all at once or by assigning it to the same or another variable.

Note: By using parentheses while assigning an object, the object will be printed immediately (instead of writing `g <- ggplot(...)` and then `g` we simply write `(g <- ggplot(...))`).

There are many, many different geometries (called *geoms* because each function usually starts with `geom_`) one can add to a ggplot by default (see here[4] for a full list) and even more provided by extension packages (see here[5] for a collection of extension packages). Let's tell **ggplot2** which style we want to use, for example by adding `geom_point()` to create a scatter plot:

```
g + geom_point()
```

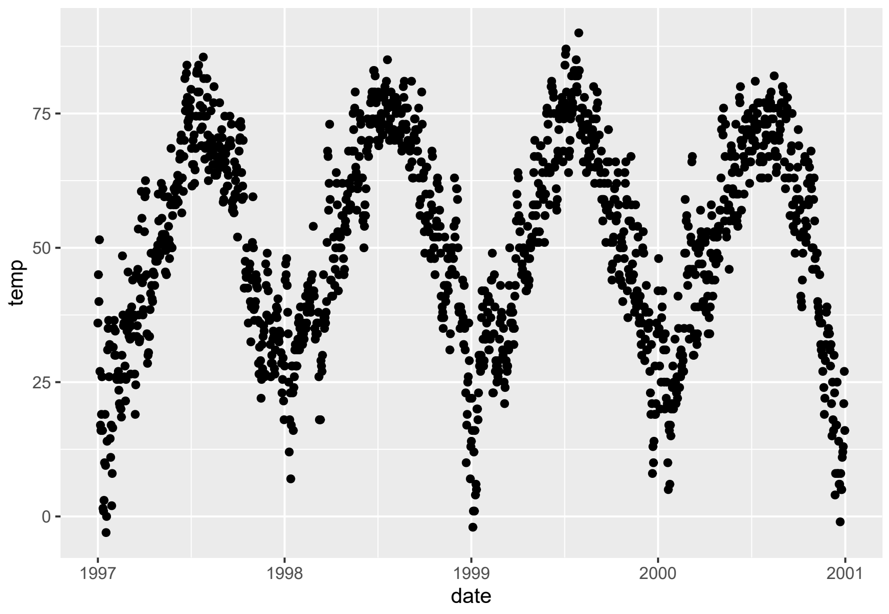

FIGURE 2.2: A default scatter plot of temperature measured in Chicago created with the **ggplot2** package.

Nice! But this data could be also visualized as a line plot (not optimal, but people do things like this all the time). So we simply add `geom_line()` instead and voilá:

[4] https://ggplot2.tidyverse.org/reference/
[5] https://exts.ggplot2.tidyverse.org/

```
g + geom_line()
```

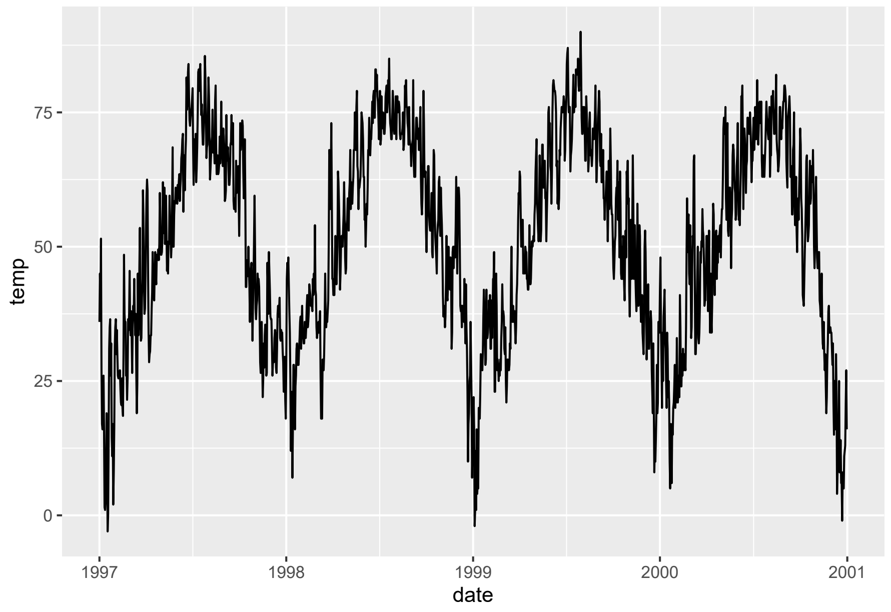

FIGURE 2.3: The same data shown as a line chart.

One can also combine several geometric layers—and this is where the magic and fun starts!

```
g + geom_line() + geom_point()
```

That's it for now about geometries. No worries, we are going to learn several plot types in Chapter `charts`.

2.4 Change Properties of Geometries

Within the `geom_*` command, you already can manipulate visual aesthetics such as the color, shape, and size of your points. Let's turn all points to large fire-red diamonds!

```
g + geom_point(color = "firebrick", shape = "diamond", size = 2)
```

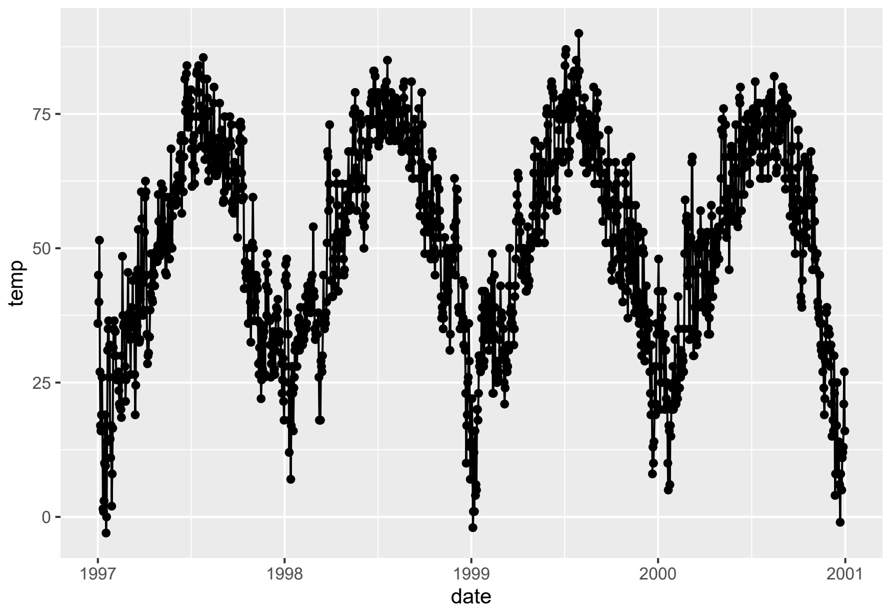

FIGURE 2.4: Again, the same data, now shown as a connected scatterplot as a combination of points and lines.

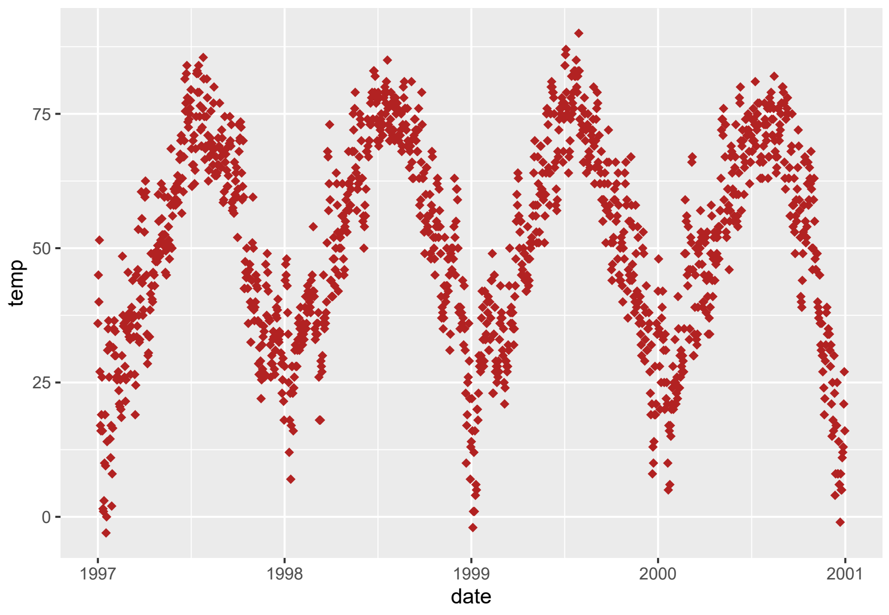

FIGURE 2.5: We can change the appearance of the geometry, here illustrated by turning the black dots in larger, red diamonds.

Note: ***ggplot2*** *understands both* `color` *and* `colour` *as well as the short version* `col`*.*

You can use preset colors (here is a full list[6]) or hex color codes[7], both in quotes, and even RGB/RGBA colors by using the `rgb()` function.

```
g + geom_point(color = "#b22222", shape = "diamond", size = 2)
g + geom_point(color = rgb(178, 34, 34, maxColorValue = 255), shape = "diamond", size = 2)
```

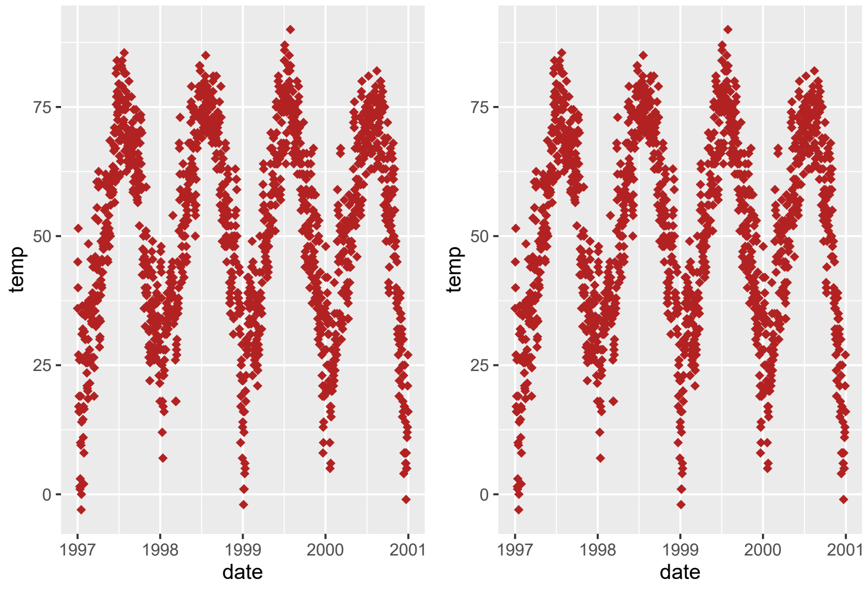

FIGURE 2.6: Comparing two ways of specifying the color in **ggplot2**.

Each geom comes with its own properties (called *arguments*) and the same argument may result in a different change depending on the geom you are using.

```
g + geom_line(color = "gray40", linetype = "dotted", size = .6) +
   geom_point(color = "firebrick", shape = "diamond", size = 2)
```

[6]http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
[7]https://www.techopedia.com/definition/29788/color-hex-code

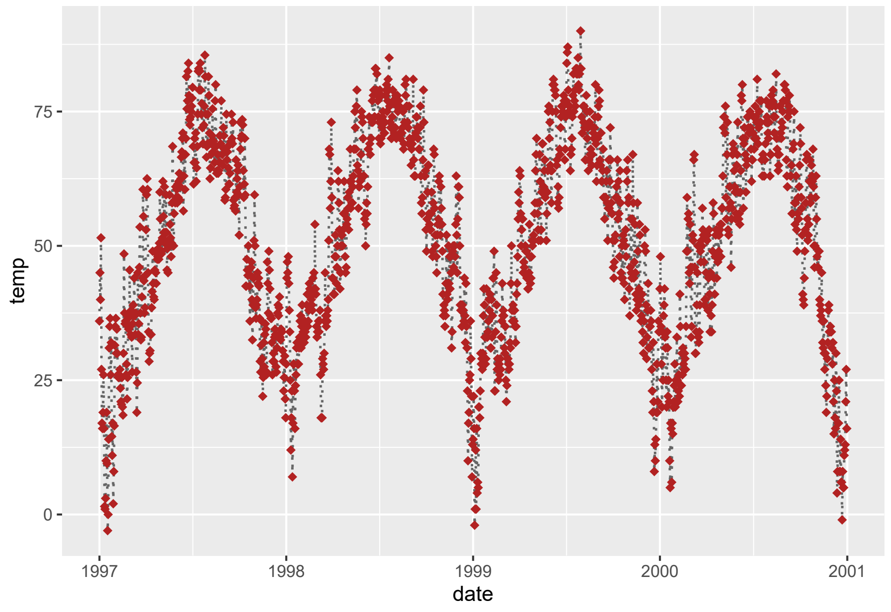

FIGURE 2.7: You can style each geometrical layer on its own. Each geometry also comes with a set of individual properties.

2.5 Replace the default ggplot2 theme

And to illustrate some more of ggplot's versatility, let's get rid of the grayish default **ggplot2** look by setting a different built-in theme, e.g. `theme_bw()`—by calling `theme_set()` all following plots will have the same black'n'white theme. The red points look way better now!

```
theme_set(theme_bw())

g + geom_point(color = "firebrick")
```

You can find more on how to use built-in themes and how to customize themes in the section `themes`. From the next chapter on, we will also use the `theme()` function to customize particular elements of the theme.

Note: `theme()` is an essential command to manually modify all kinds of theme elements (texts, rectangles, and lines).

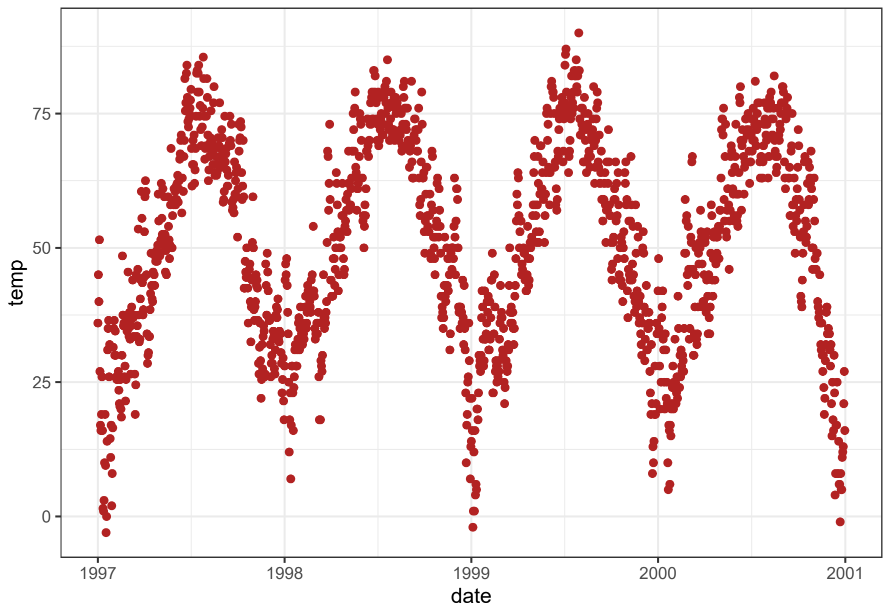

FIGURE 2.8: We can change the appearance for all following plots generated with **ggplot2** by globally setting another theme via `theme_set()`.

To see which details of a ggplot theme can be modified have a look here[8]—and take some time, this is a looong list.

[8]https://ggplot2.tidyverse.org/reference/theme.html

A

More to Say

Yeah! I have finished my book, but I have more to say about some topics. Let me explain them in this appendix.

To know more about **bookdown**, see `https://bookdown.org`.

Bibliography

Sciaini, M., Fritsch, M., Scherer, C., and Simpkins, C. E. (2018). Nlmr and landscapetools: An integrated environment for simulating and modifying neutral landscape models in r. *Methods in ecology and evolution*, 9(11):2240–2248.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2021). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.22.

Index