Cédric Scherer

Graphic Design with ggplot2

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. I planned to create a visualization using small multiples to visualize the outcomes of my simulation model. I was already using the R programing 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 felt in love with the idea of ggplot'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 over the time I added 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]. TidyTuesday is an initative grown out of Jesse Mostipak[3] and the R4DS Online Learning Community[4], started by Thomas Mock[5]. Every week, a raw data set is 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 started working as a freelance data visualization specialist. I am now using ggplot2 every day: for my scientific work, design requests, reproducible reports, and personal data visualization projects. Since the blog entry by Zev Ross was not updated sine January 2016, I decided to add more examples and tricks to my version, which was now hosted on my personal blog[6]. It became step by step a unique tutorial that now contains also the

[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

[3] https://www.twitter.com/kierisi

[4] https://www.rfordatasci.com/

[5] https://www.twitter.com/thomas_mock

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

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 August 2021, this tutorial serves as the basis 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 cannot believe what one can achieve with ggplot2—and I want to show them 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 goal is to offer a book that contains everything needed to create appealing data visualizations with the help of the R library "ggplot2". The aim is to include the basics of ggplot2, a detailed how-to section on improving the overall design and readability, an overview of useful extension packages, and unique reference implementations of high-quality, state-of-the-art data visualizations. The book will also include a comprehensive inventory of data visualization techniques and good practice along the code examples.

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.

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 19041)
##
## 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., **rmarkdown**), and inline code and filenames are formatted in a typewriter font (e.g., `knitr::knit('foo.Rmd')`). Function names are followed by parentheses (e.g., `bookdown::render_book()`).

Acknowledgments

A lot of people helped me when I was writing the book.

Cédric Scherer
Berlin, Germany

About the Author

Frida Gomam is a famous lady. Police will always let her go.

1 Introduction

1.1 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[2] (but you don't have to).

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

```
chic <- readr::read_csv("https://raw.githubusercontent.com/Z3tt/R-Tutorials/master/ggplot2/chicago
```

```
##
## -- 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()
## )
```

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

[1] http://www.springer.com/de/book/9780387781662
[2] https://github.com/Z3tt/R-Tutorials/blob/master/ggplot2/chicago-nmmaps.csv

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, ~
```

```
head(chic, 10)
```

```
## # A tibble: 10 x 10
##    city  date       death  temp dewpoint  pm10    o3
##    <chr> <date>     <dbl> <dbl>    <dbl> <dbl> <dbl>
##  1 chic  1997-01-01   137  36      37.5  13.1   5.66
##  2 chic  1997-01-02   123  45      47.2  41.9   5.53
##  3 chic  1997-01-03   127  40      38    27.0   6.29
##  4 chic  1997-01-04   146  51.5    45.5  25.1   7.54
##  5 chic  1997-01-05   102  27      11.2  15.3  20.8
##  6 chic  1997-01-06   127  17       5.75  9.36 14.9
##  7 chic  1997-01-07   116  16       7    20.2  11.9
##  8 chic  1997-01-08   118  19      17.8  33.1   8.68
##  9 chic  1997-01-09   148  26      24    12.1  13.4
## 10 chic  1997-01-10   121  16       5.38 24.8  10.4
## # ... with 3 more variables: time <dbl>, season <chr>,
## #   year <dbl>
```

1.2 The `{ggplot2}` Package

`ggplot2` is a system for declaratively creating graphics, based on The Grammar of Graphics[3]. You provide the data, tell `ggplot2` how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

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.
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.

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

1.3 A Default 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[4]):

```
#library(ggplot2)
library(tidyverse)
```

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

[3] https://www.amazon.com/Grammar-Graphics-Statistics-Computing/dp/0387245448/ref=as_li_ss_tl?ie=UTF8&qid=1477928463&sr=8-1&keywords=the+grammar+of+graphics&linkCode=sl1&tag=ggplot2-20&linkId=f0130e557161b83fbe97ba0e9175c431

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

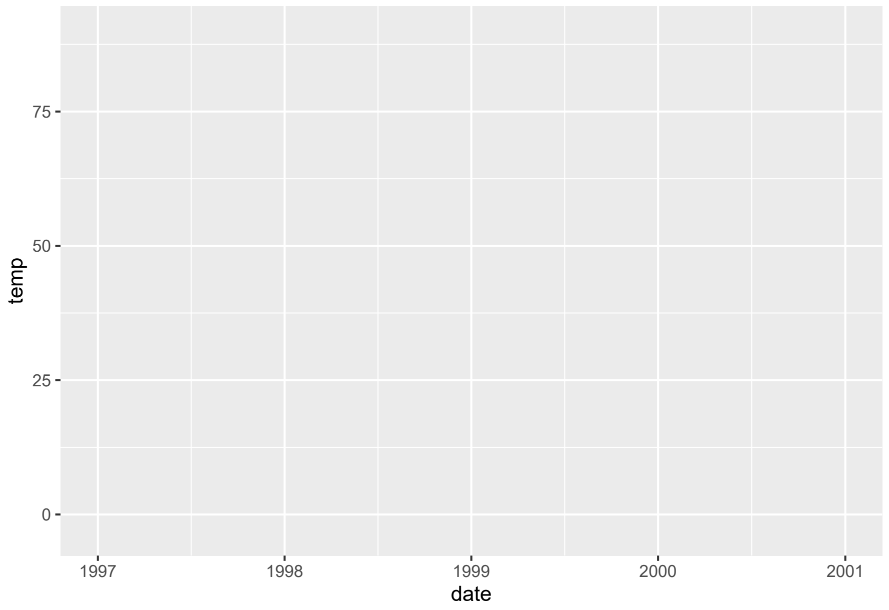
```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- 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 *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 one or even three or more variables.

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.

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



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!

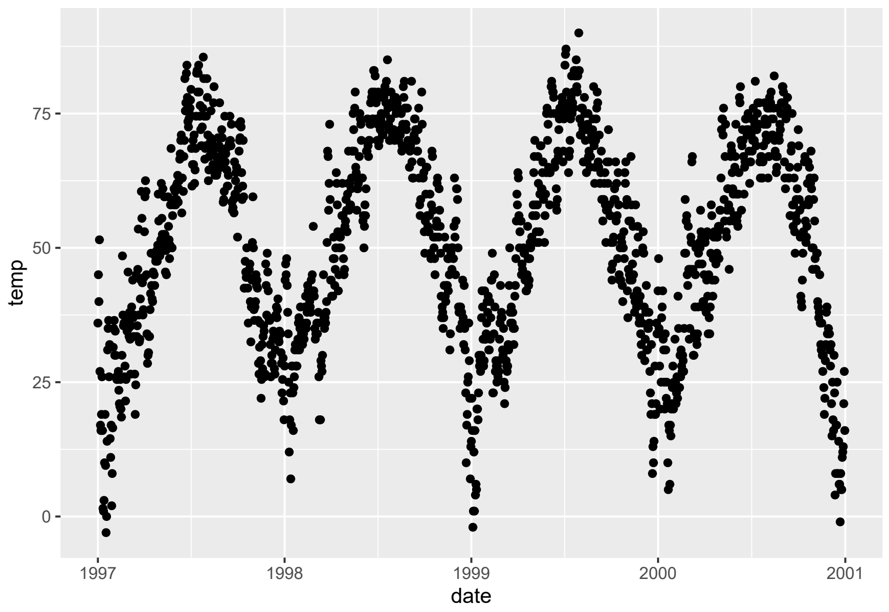
`ggplot2` allows you to store the current `ggobject` in a variable of your choice by assigning it to a variable, in our case called `g`. You can extend this `ggobject` later by adding other layers, either all at once or by assigning it to the same or another variable.

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[5] for a full list) and even more provided by extension packages (see here[6] 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:

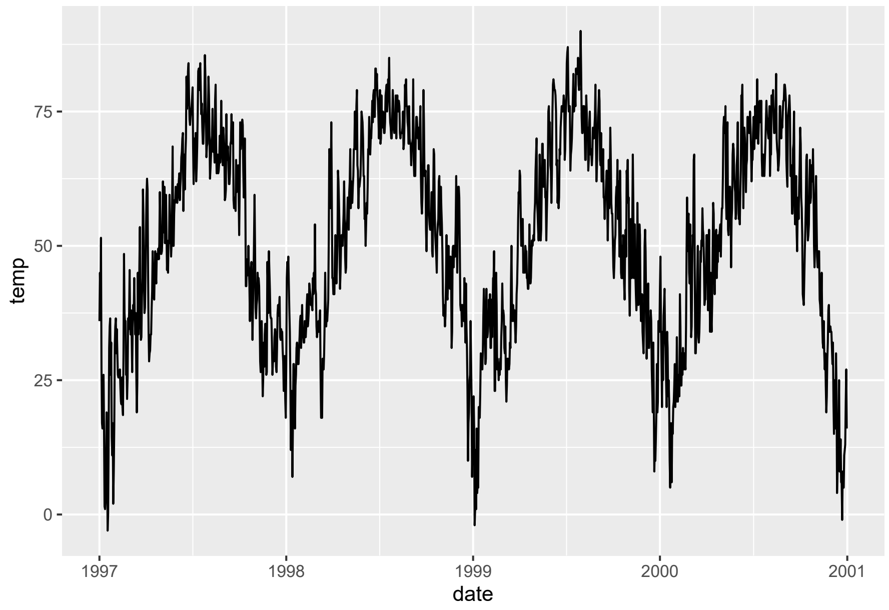
```
g + geom_point()
```

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



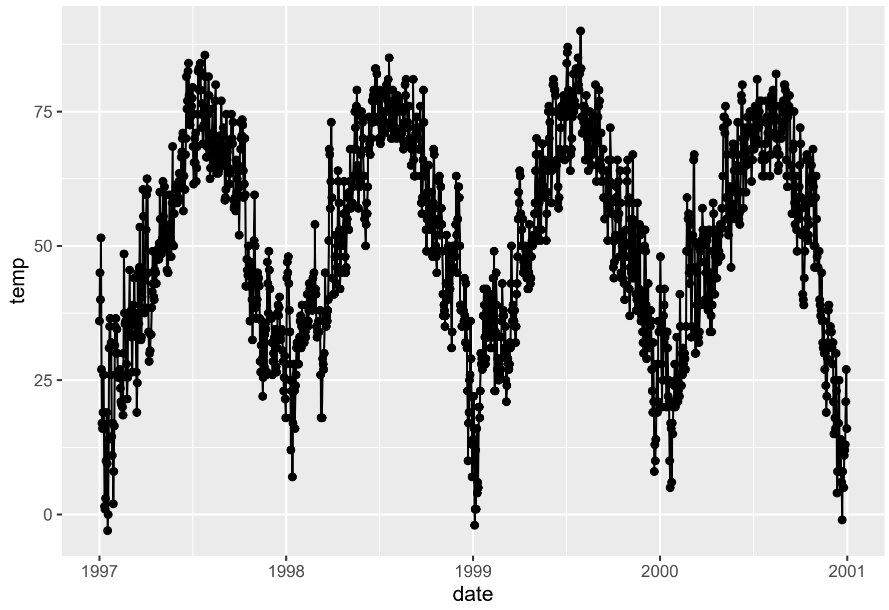
Nice! But this data could be also visualized as a line plot (not optimal, but people do things like this all the time). So we simply add `geom_line()` instead and voilá:

```
g + geom_line()
```



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

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

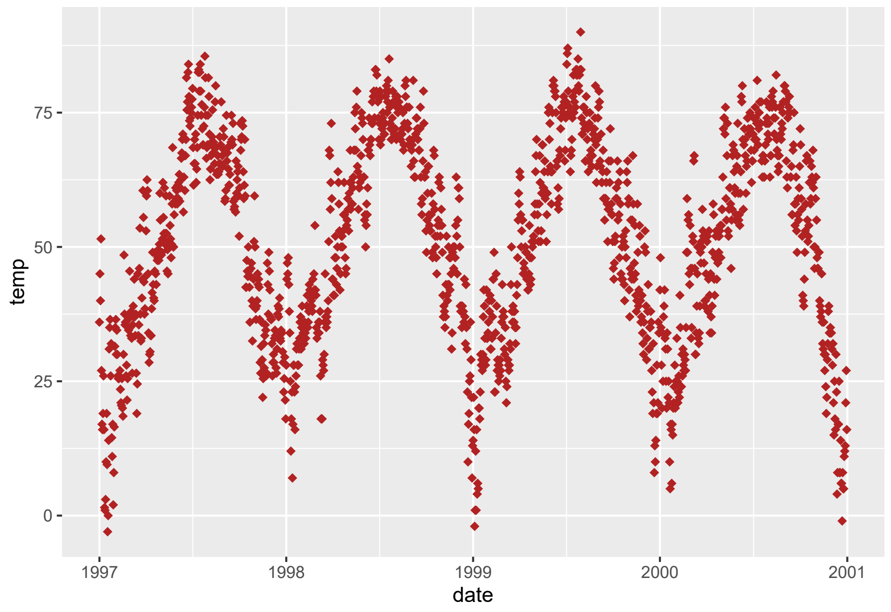


That's it for now about geometries. No worries, we are going to learn several plot types in ??.

1.3.0.0.1 *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)
```

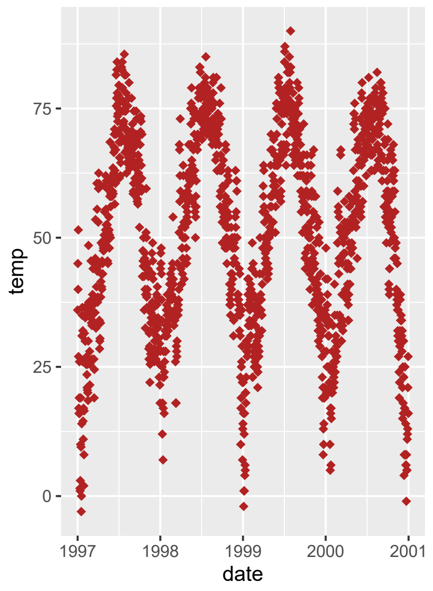


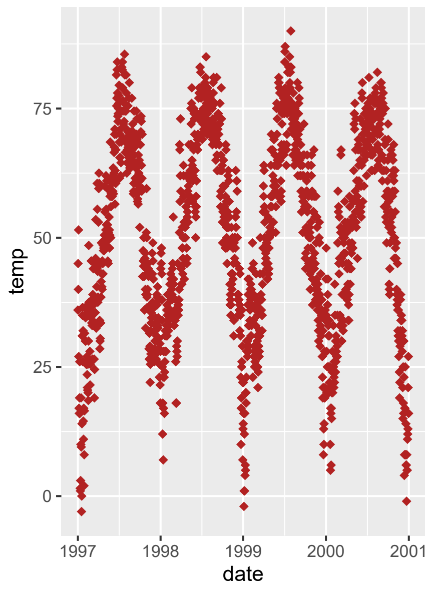
`{ggplot2}` understands both `color` and `colour` as well as the short version `col`.

You can use preset colors (here is a full list[7]) or hex color codes[8], 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)
```

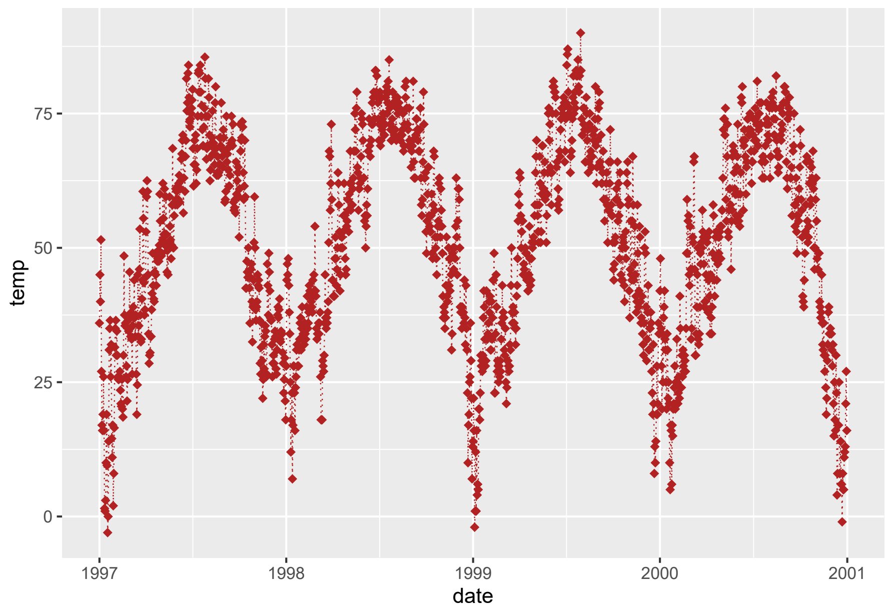
[7]http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf
[8]https://www.techopedia.com/definition/29788/color-hex-code





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.

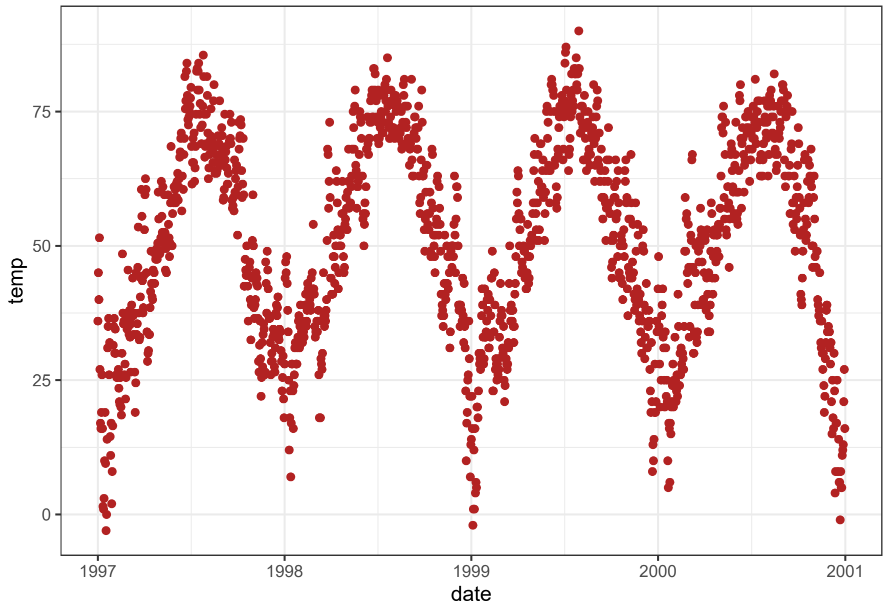
```
g + geom_point(color = "firebrick", shape = "diamond", size = 2) +
    geom_line(color = "firebrick", linetype = "dotted", size = .3)
```



1.3.0.0.2 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 "Working with Themes". From the next chapter on, we will also use the `theme()` function to customize particular elements of the theme.

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

To see which details of a ggplot theme can be modified have a look here[9]—and take some time, this is a looong list.

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

2 *The FOO Method*

We talk about the *FOO* method in this chapter.

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

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