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. 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 initiative grown out of Jesse Mostipak and the R4DS Online Learning Community[3], started by Thomas Mock. 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[4]. 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,

[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.rfordatasci.com/

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

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.

Among other things, the book covers the following aspects:

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

- Discussion of best practices in data visualization (e.g. colour 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

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

Cédric Scherer is a graduated computational ecologist with a passion for 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 to automate data analyses and plot generation, following the philosophy of a reproducible workflow.

Webpage: cedricscherer.com
Twitter: @CedScherer
GitHub: z3tt

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: http://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 <- "http://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 without*

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

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, ~
```

1.2 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.
3. **Aesthetics** `aes()`: Aesthetics of the geometric and statistical objects, such as position, color, size, shape, and transparency

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

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.

1.3 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 *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.

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

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)))
```

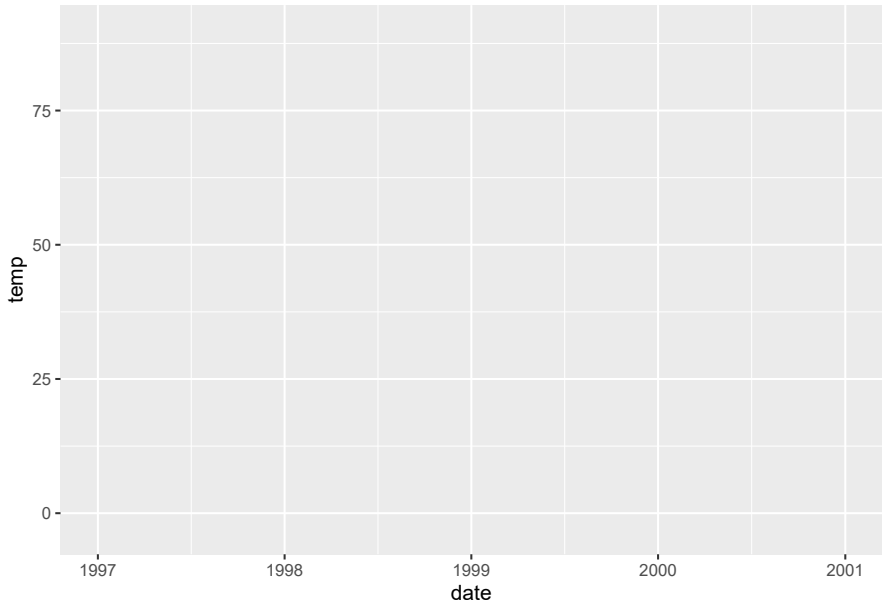

FIGURE 1.1: A default plot generated with the `{ggplot2}` package with variables mapped to `x` and `y`.

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 func-

tion 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()
```

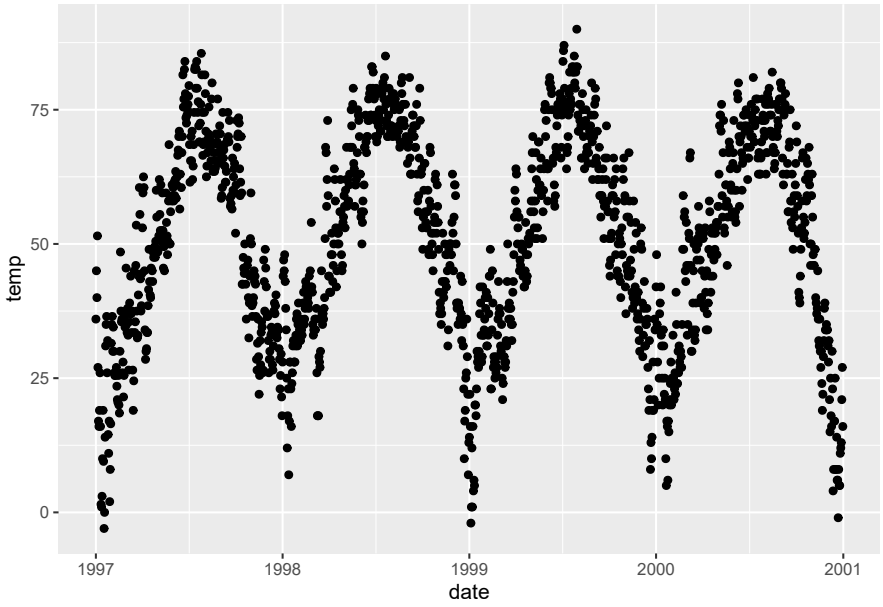

FIGURE 1.2: A default scatter plot of temperature measured in Chicago created with the `{ggplot2}` package.

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:

```
g + geom_line()
```

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

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

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

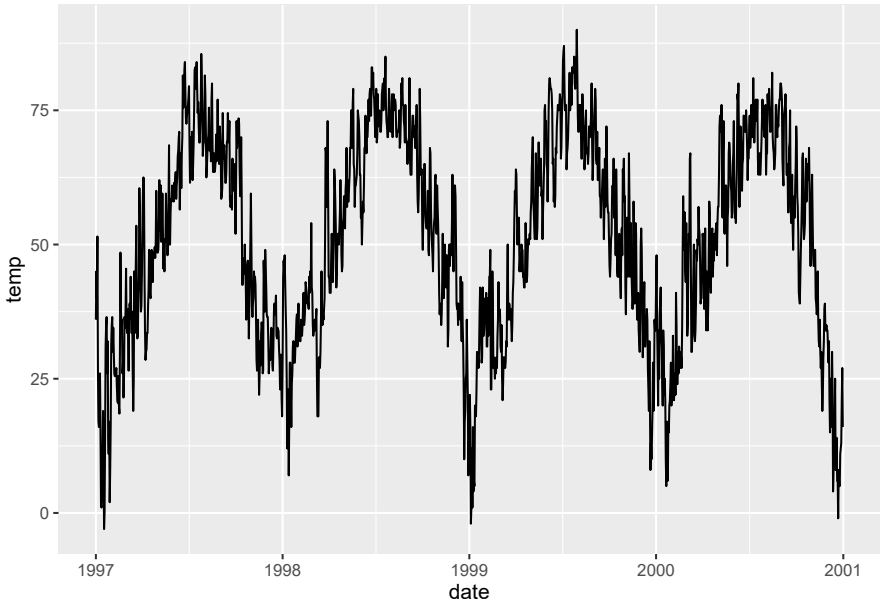

FIGURE 1.3: The same data shown as a line chart.

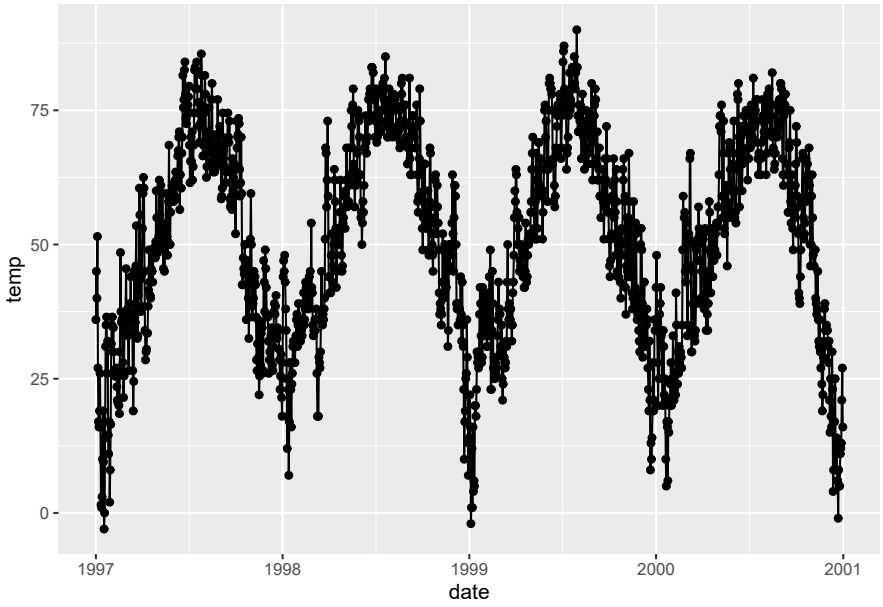

FIGURE 1.4: Again, the same data, now shown as a connected scatterplot as a combination of a points and lines.

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

1.4 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)
```

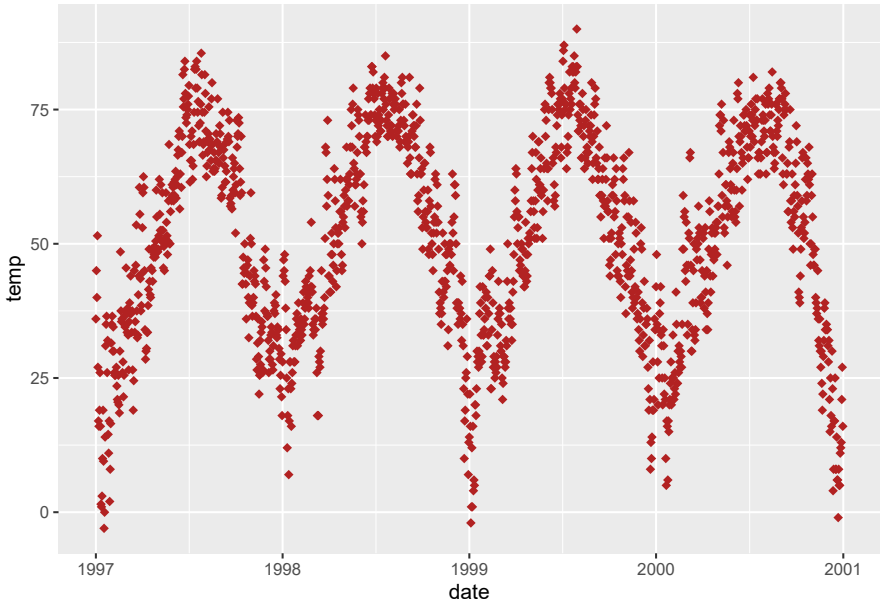

FIGURE 1.5: We can change the appearance of the geometry, here illustrated by turning the black dots in larger, red diamonds.

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.

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

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

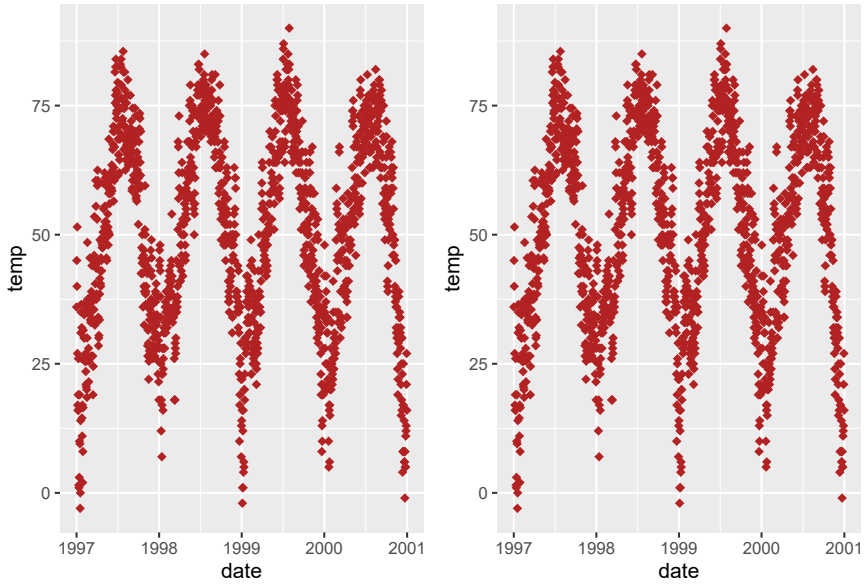

FIGURE 1.6: Comparing two ways of specifying the color in `{ggplot2}`.

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

1.5 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!

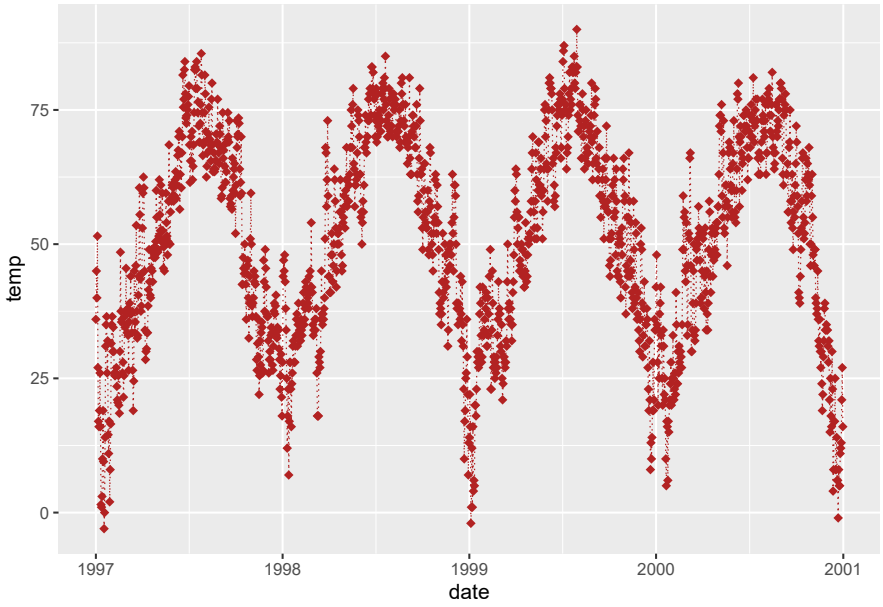

FIGURE 1.7: (#fig:ggplot-default-line_col-size-shape)You can style each geometrical layer on its own. Each geometry also comes with a set of individual properties.

```
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).

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

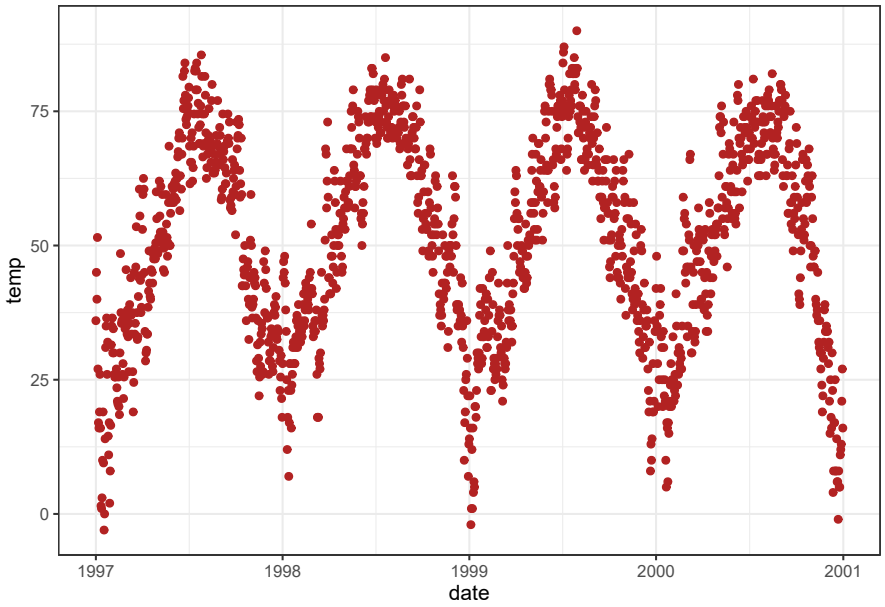

FIGURE 1.8: We can change the appearance for all following plots generated with `{ggplot2}` by globally setting another theme via `theme_set()`.

2

Chart types

2.1 Alternatives to a Box Plot

A box plot is probably the most commonly used chart type to compare distribution of several groups, at least in the scientific literature. Box plots are great, but they can be so incredibly boring. Also, even if you are used to looking at box plots, remember there might be plenty people looking at your plot that have never seen a box and whisker plot before.

A box-and-whisker plot (sometimes called simply a box plot) is a histogram-like method of displaying data, invented by J. Tukey. The thick **middle line** notates the median, also known as quartile. The limits of the **box** are determined by the lower and upper quartiles, Q^1 and Q^3. The box contains thus 50% of the data and is called *"interquartile range"* (IQR). The length of the **whiskers** is determined by the most extreme values that are not considered as outliers (i.e. values that are within 3/2 times the interquartile range).

```
knitr::include_graphics("img/boxplot.png")
```

There are alternatives, but first we are plotting a common box plot:

```
g <-
  ggplot(chic, aes(x = season, y = o3,
                   color = season)) +
    labs(x = "Season", y = "Ozone") +
    scale_color_brewer(palette = "Dark2", guide = "none")

g + geom_boxplot()
```

2.1.1 Dot Plots

Let's plot just each data point of the raw data:

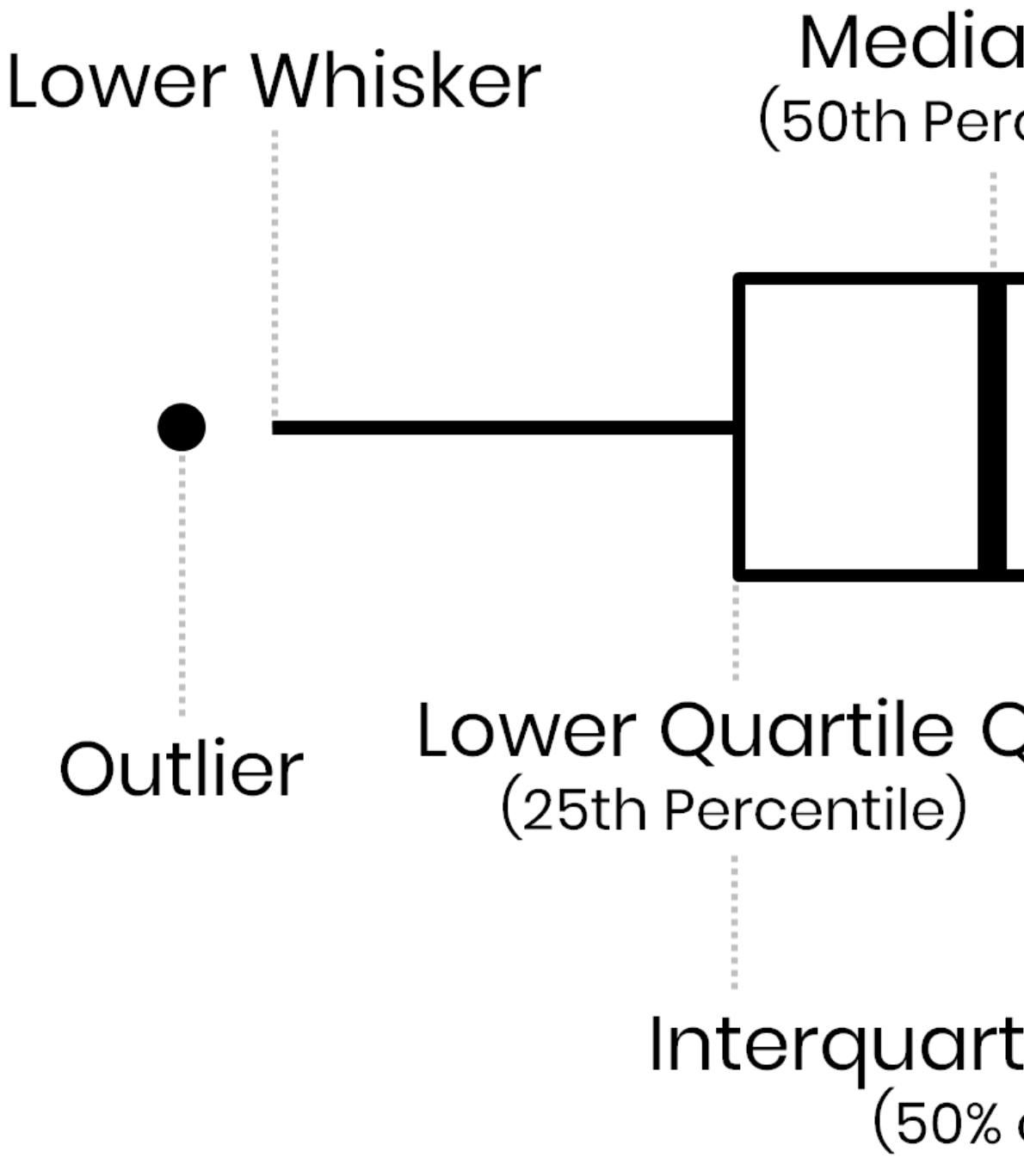

FIGURE 2.1: A boxplot is a standardized way of displaying the dataset based on a five-number summary: the minimum, the maximum, the sample median, and the first and third quartiles.

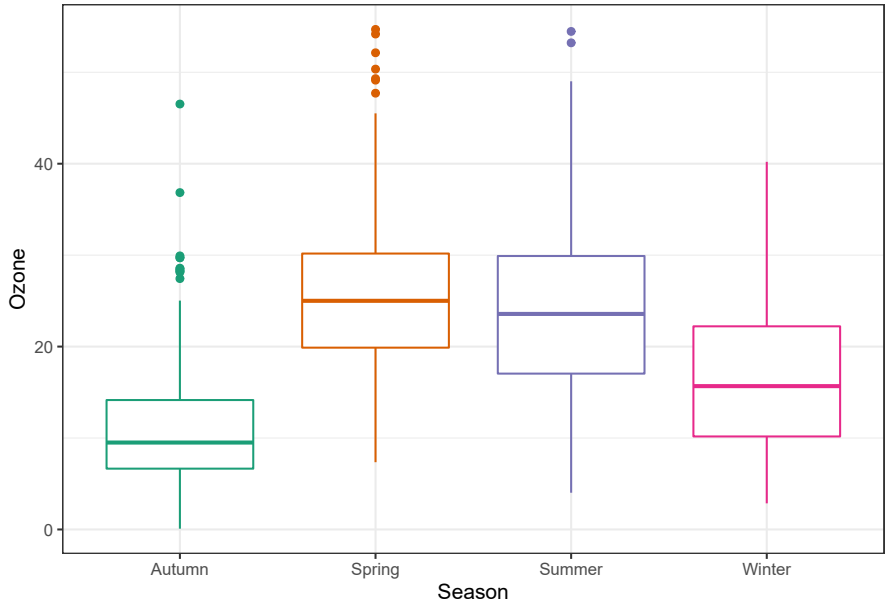

FIGURE 2.2: A box plot generated with `{ggplot2}` with customized axis labels and colors.

```
g + geom_point()
```

Not only boring but uninformative. To improve the plot, one could add transparency to deal with overplotting:

```
g + geom_point(alpha = .1)
```

However, setting transparency is difficult here since either the overlap is still too high or the extreme values are not visible. Bad, so let's try something else.

2.1.2 Strip Plots

Try adding a little jitter to the data. I like this for in-house visualization but be careful using jittering because you are purposely adding noise to your data and this can result in misinterpretation of your data.

```
g + geom_jitter(width = .3, alpha = .5)
```

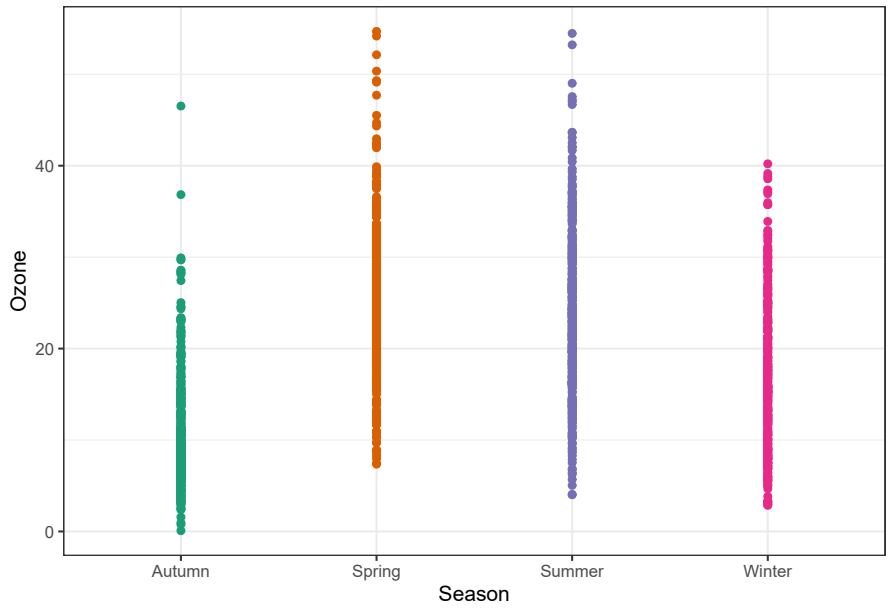

FIGURE 2.3: A dot plot visualizing the distribution of the raw data as an alternative to boxplots. However, the summary statistics are not shown anymore.

2.1.3 Violin Plots

Violin plots, similar to box plots except you are using a kernel density to show where you have the most data, are a useful visualization.

```
g + geom_violin(fill = "gray80", size = 1, alpha = .5)
```

2.1.4 Combining Violin Plots with Strip Charts

We can of course combine both, estimated densities and the raw data points:

```
g + geom_violin(fill = "gray80", size = 1, alpha = .5) +
    geom_jitter(alpha = .25, width = .3) +
    coord_flip()
```

The {ggforce} package[1] provides so-called sina functions where the width of the jitter is controlled by the density distribution of the data—that makes the jittering a bit more visually appealing:

[1] https://ggforce.data-imaginist.com/

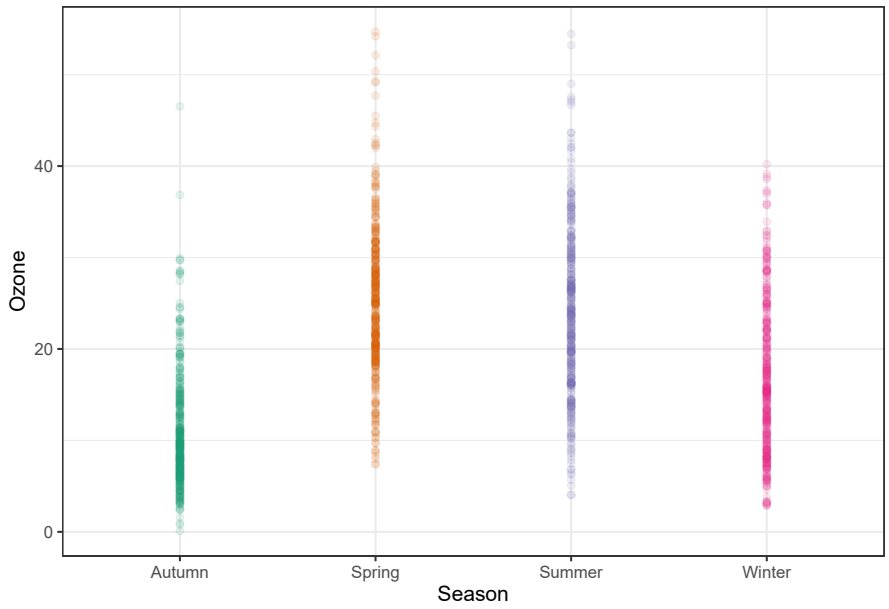

FIGURE 2.4: A dot plot with added transparency as an approach to make overlapping data points visible.

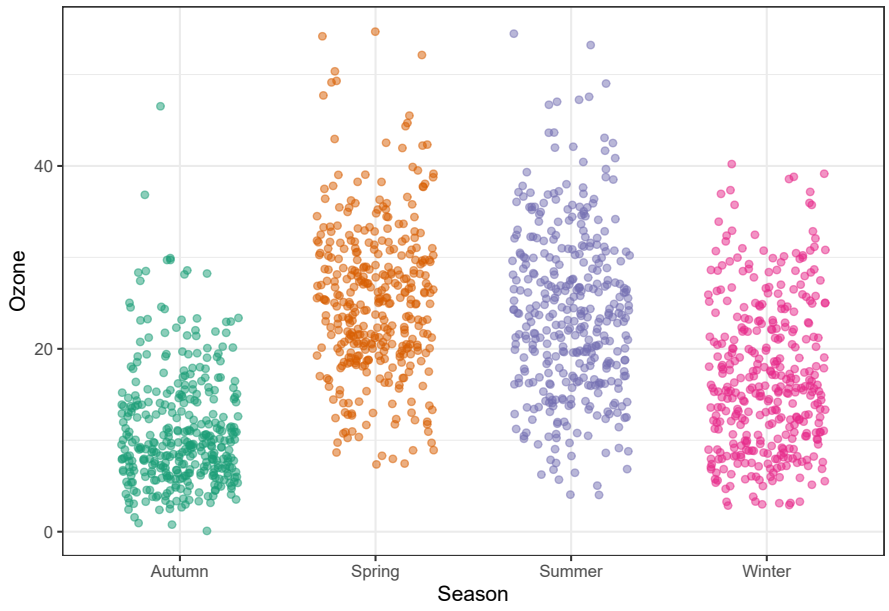

FIGURE 2.5: A jittered dot plot, also known as strip plot.

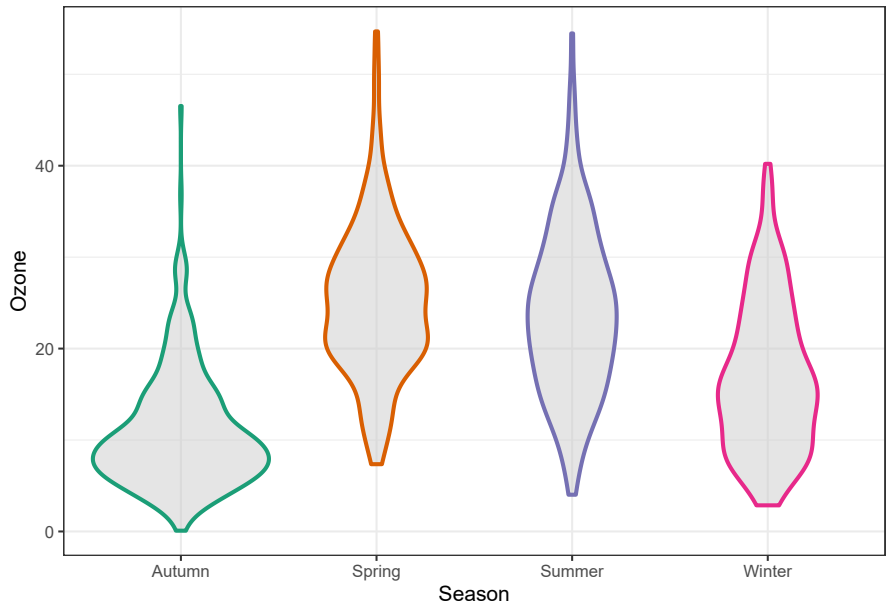

FIGURE 2.6: Violin plots are a great way to visualize the distribution values, especially in case of large sample sizes.

```
library(ggforce)

g + geom_violin(fill = "gray80", size = 1, alpha = .5) +
    geom_sina(alpha = .25) +
    coord_flip()
```

2.1.5 Combining Violin and Box Plots

To allow for easy estimation of quantiles, we can also add the box of the box plot inside the violins to indicate 25%-quartile, median and 75%-quartile:

```
g + geom_violin(aes(fill = season), size = 1, alpha = .5) +
    geom_boxplot(outlier.alpha = 0, coef = 0,
                 color = "gray40", width = .2) +
    scale_fill_brewer(palette = "Dark2", guide = "none") +
    coord_flip()
```

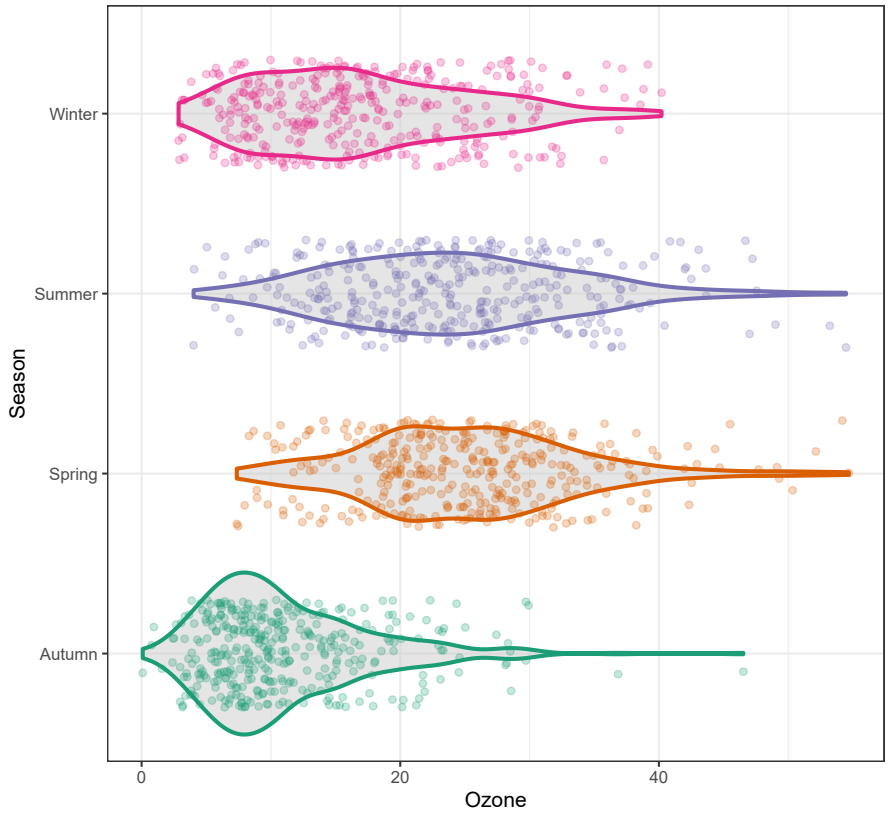

FIGURE 2.7: A combination of violin and strip plots to visualize the distribution and the raw values at the same time.

2.2 Create a Rug Representation to a Plot

A rug represents the data of a single quantitative variable, displayed as marks along an axis. In most cases, it is used in addition to scatter plots or heatmaps to visualize the overall distribution of one or both of the variables:

```
ggplot(chic, aes(x = date, y = temp,
                 color = season)) +
  geom_point(show.legend = FALSE) +
  geom_rug(show.legend = FALSE) +
  labs(x = "Year", y = "Temperature (°F)")
```

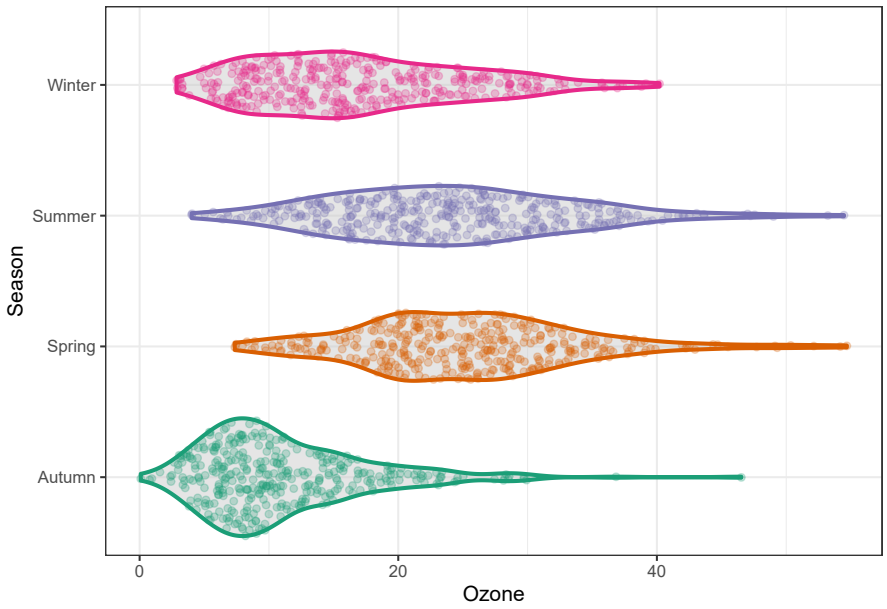

FIGURE 2.8: A sina plot places the dots according to the distribution and is basically a *raw data violin plot.*

2.3 Create a Correlation Matrix

There are several packages that allow to create correlation matrix plots, some also using the`{ggplot2}` infrastructure and thus returning ggplots. I am going to show you how to do this without extension packages.

First step is to create the correlation matrix. Here, we use the `{corrr}` package that works nicely with pipes but there are also many others out there. We are using Pearson because all the variables are fairly normally distributed (but you may consider Spearman if your variables follow a different pattern). Note that since a correlation matrix has redundant information we are setting half of it to `NA`.

```
library(tidyverse)

corm <-
  chic %>%
  select(death, temp, dewpoint, pm10, o3) %>%
```

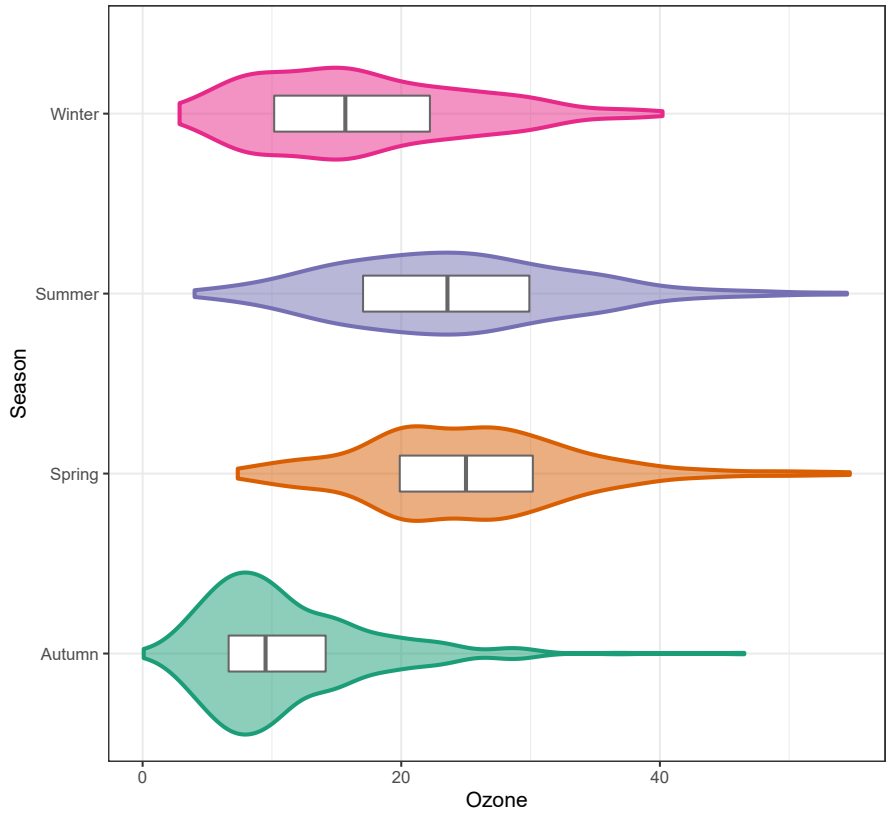

FIGURE 2.9: A combination of violin and box plots to visualize the distribution and the summary statistics of each group.

```
  corrr::correlate(diagonal = 1) %>%
  corrr::shave(upper = FALSE)
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
## # A tibble: 5 x 6
##   term      death   temp dewpoint     pm10     o3
##   <chr>     <dbl>  <dbl>    <dbl>    <dbl>  <dbl>
## 1 death         1 -0.486   -0.465 -0.00294 -0.238
## 2 temp         NA  1         0.958  0.368    0.535
## 3 dewpoint     NA NA         1      0.327    0.454
## 4 pm10         NA NA        NA      1        0.206
```

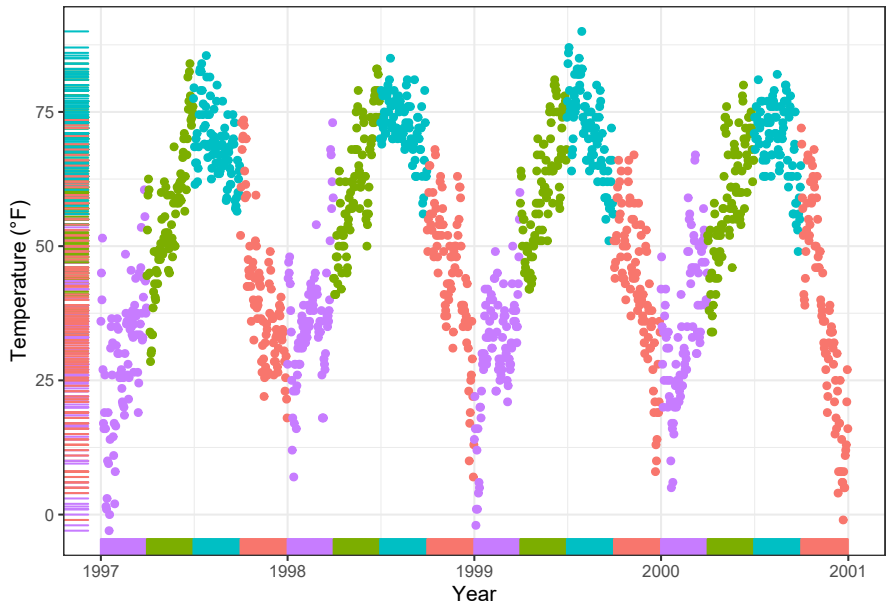

FIGURE 2.10: A so-called rug or bardcode plot is a powerful way to highlight the distribution along one axis without requiring much more space.

```
## 5 o3          NA NA       NA     NA        1
```

Now we put the resulting matrix in **long** format using the `pivot_longer()` function from the `{tidyr}` package:

```
corm <- corm %>%
  pivot_longer(
    cols = -term,
    names_to = "colname",
    values_to = "corr"
  ) %>%
  mutate(rowname = fct_inorder(term),
         colname = fct_inorder(colname))
```

```
## # A tibble: 25 x 4
##    term  colname      corr rowname
##    <chr> <fct>       <dbl> <fct>
##  1 death death     1       death
##  2 death temp     -0.486   death
##  3 death dewpoint -0.465   death
##  4 death pm10     -0.00294 death
```

```
##  5 death o3       -0.238   death
##  6 temp  death    NA       temp
##  7 temp  temp      1       temp
##  8 temp  dewpoint  0.958   temp
##  9 temp  pm10      0.368   temp
## 10 temp  o3        0.535   temp
## # ... with 15 more rows
```

For the plot we will use `geom_tile()` for the heatmap and `geom_text()` for the labels:

```
ggplot(corm, aes(rowname, fct_rev(colname),
                 fill = corr)) +
  geom_tile() +
  geom_text(aes(label = round(corr, 2))) +
  coord_fixed() +
  labs(x = NULL, y = NULL)
```

```
## Warning: Removed 10 rows containing missing values
## (geom_text).
```

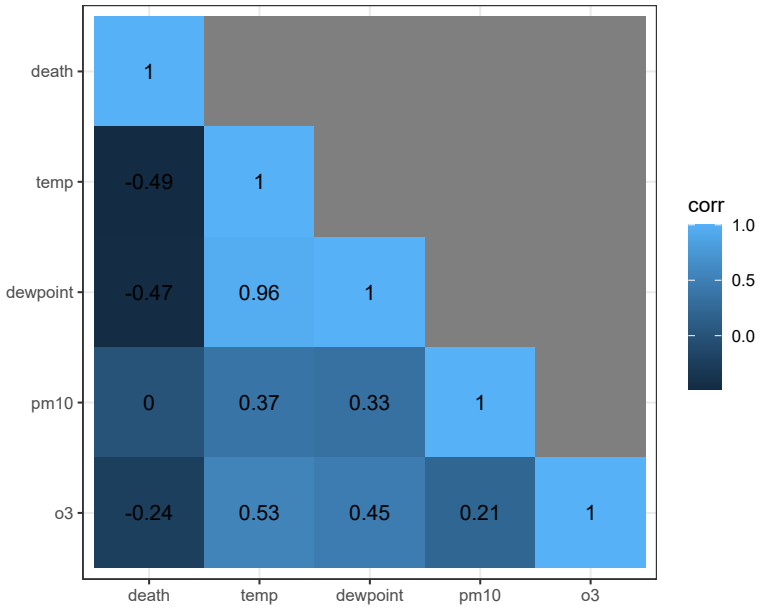

FIGURE 2.11: A basic correlation plot, created with our prepared correlation data set and the help of `geom_tile()`.

I like to have a diverging color palette, centered at zero correlation, with white

indicating missing data. Also I like to have no grid lines and padding around the heatmap as well as nicely formatted labels that are colored depending on the underlying fill:

```
ggplot(corm, aes(rowname, fct_rev(colname),
                 fill = corr)) +
  geom_tile() +
  geom_text(aes(
    label = format(round(corr, 2), nsmall = 2),
    color = abs(corr) < .75
  )) +
  coord_fixed(expand = FALSE) +
  scale_color_manual(values = c("white", "black"),
                     guide = "none") +
  scale_fill_distiller(
    palette = "PuOr", na.value = "white",
    direction = 1, limits = c(-1, 1)
  ) +
  labs(x = NULL, y = NULL) +
  theme(panel.border = element_rect(color = NA, fill = NA),
        legend.position = c(.85, .8))
```

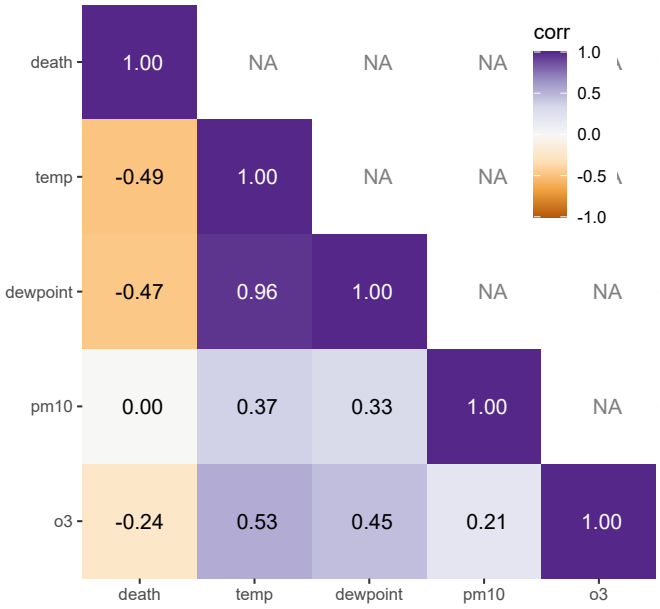

FIGURE 2.12: A polished version of the same correlation plot with custom color palettes and theme adjustments.

3

Themes

You can change the entire look of the plots by using themes. `{ggplot2}` comes with eight built-in themes:

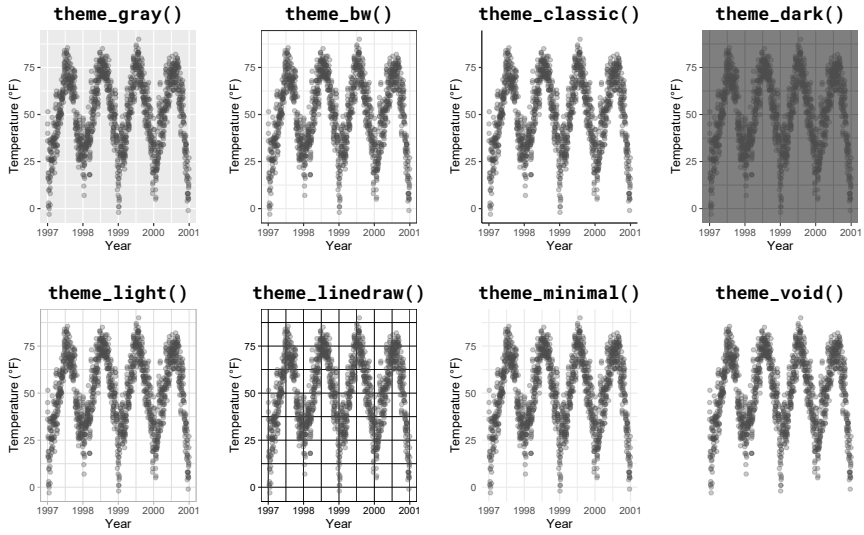

FIGURE 3.1: An overview of all in-build themes that are contained in the `{ggplot2}` package.

There are several packages that provide additional themes, some even with different default color palettes. As an example, Jeffrey Arnold has put together the library `{ggthemes}` with several custom themes imitating popular designs. For a list you can visit the `{ggthemes}` package site[1]. Without any coding you can just adapt several styles, some of them well known for their style and aesthetics.

[1] https://github.com/jrnold/ggthemes

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