Hands–On Data Visualization with ggplot2

Segment 1: The Grammar of Graphics

Dr. Cédric Scherer

Pearson Live Training Session for O'Reilly

September 3, 2021

Photo by Richard Strozynski

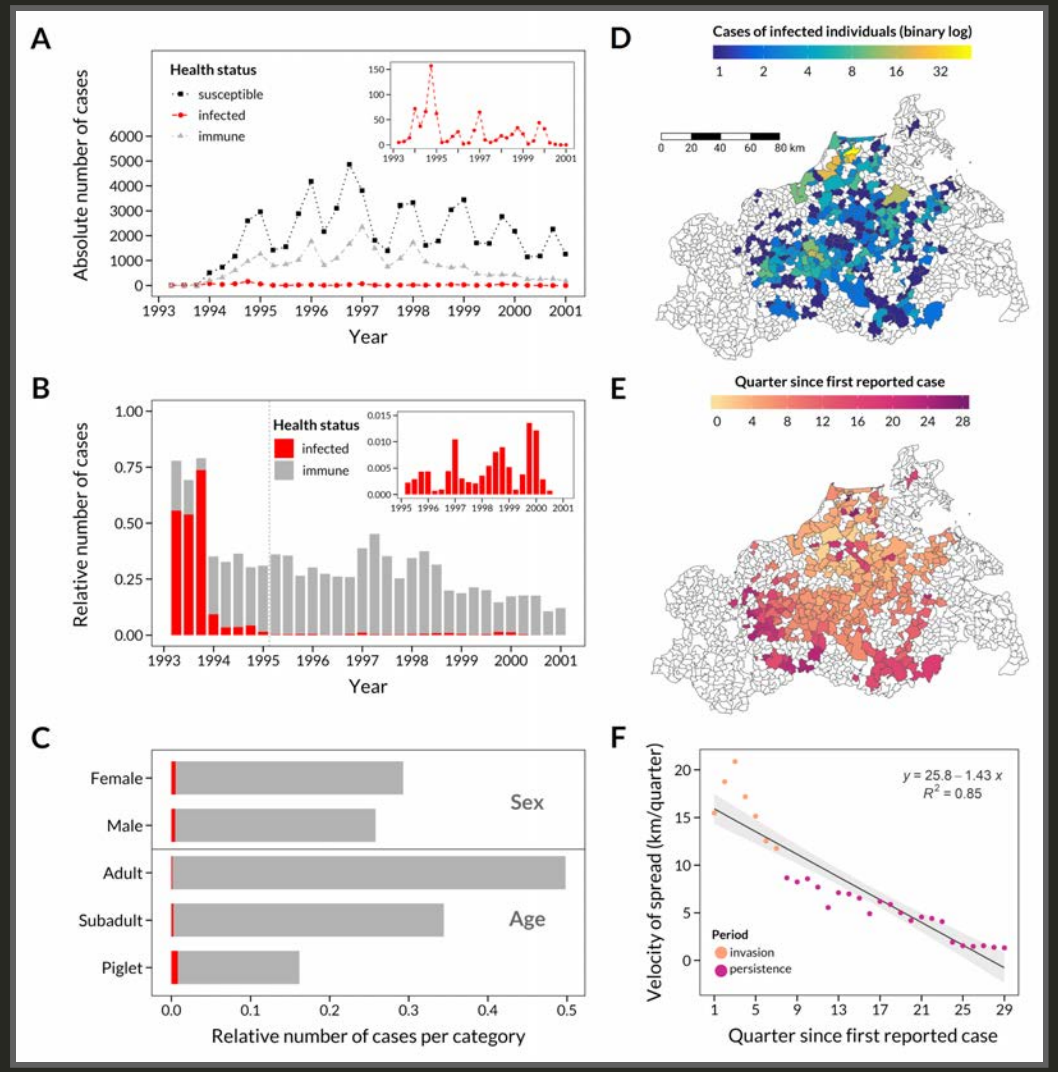
Cédric Scherer

Freelance Data Visualization Specialist
Computational Ecologist at IZW Berlin

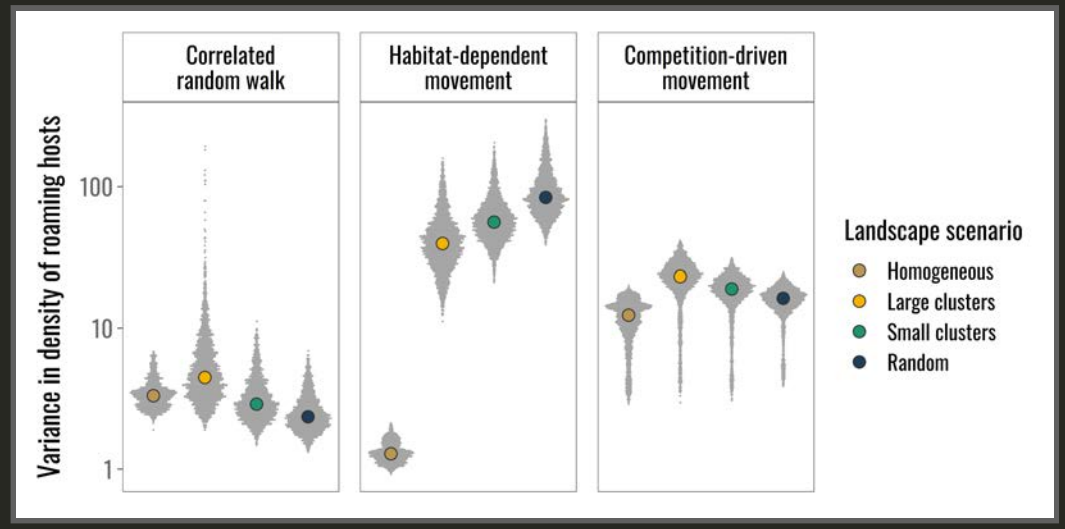
cedricscherer.com @CedScherer @z3tt @cedscherer


tidyverse
dplyr
glue
xaringan
lubridate
ggplot2
forcats
tibble
broom
rmarkdown
readr
stringr
tidyr
Shiny
purrr
RStudio
R

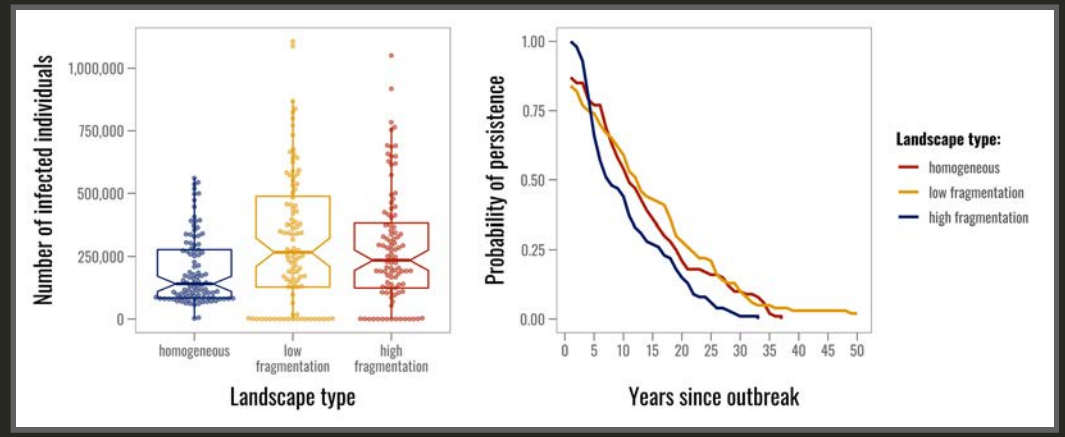
Data Visualizations for Scientific Publications & Talks



Scherer et al. 2019 Journal of Animal Ecology

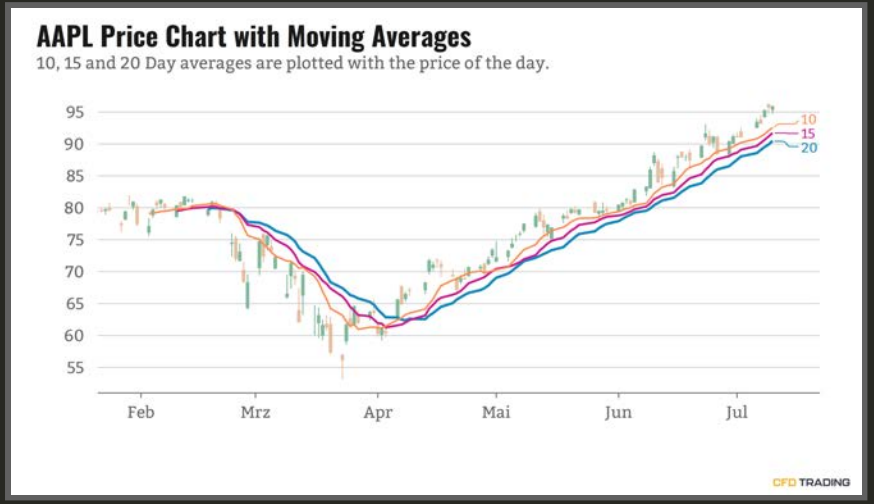


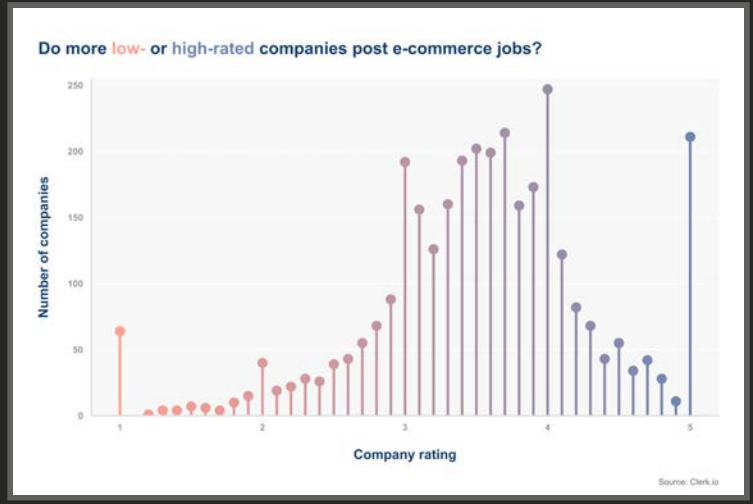
Scherer et al. 2020 Oikos

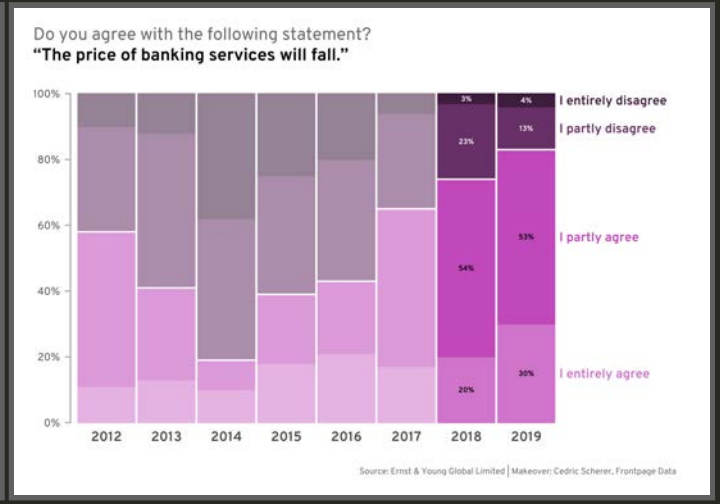


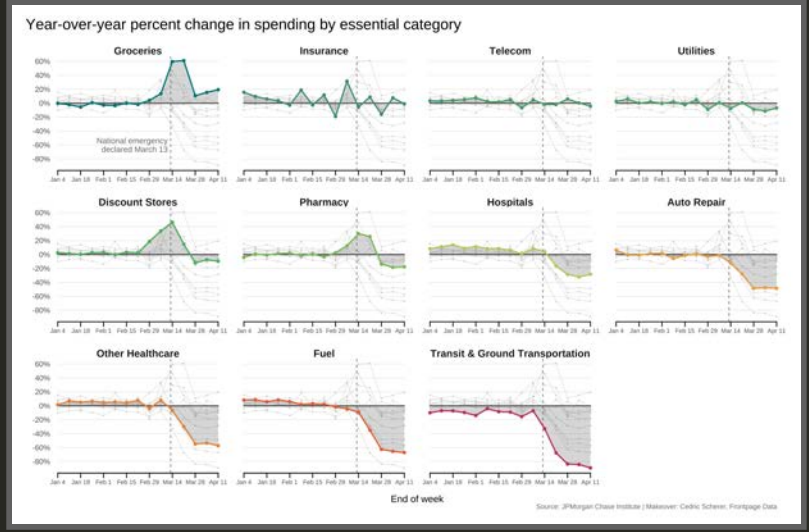
Sciaini et al. 2019 Methods in Ecology & Evolution

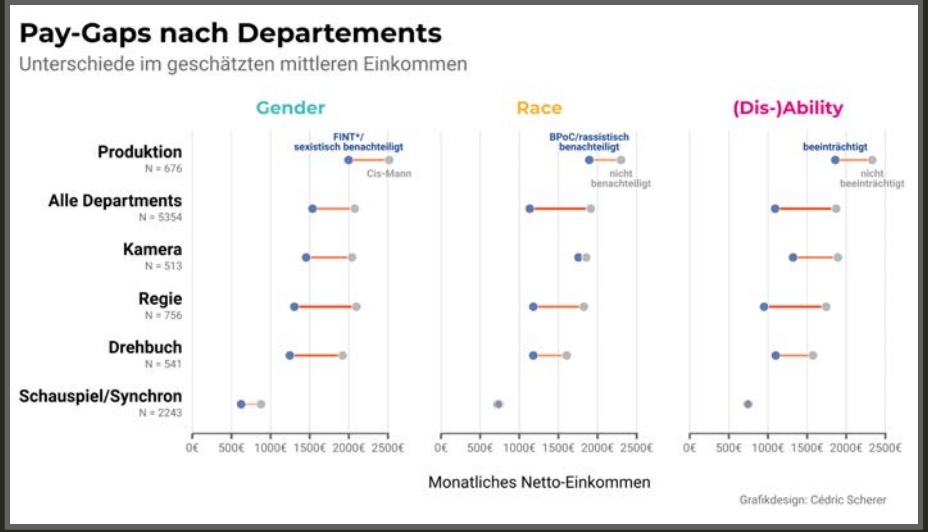
Data Visualizations for Client Projects



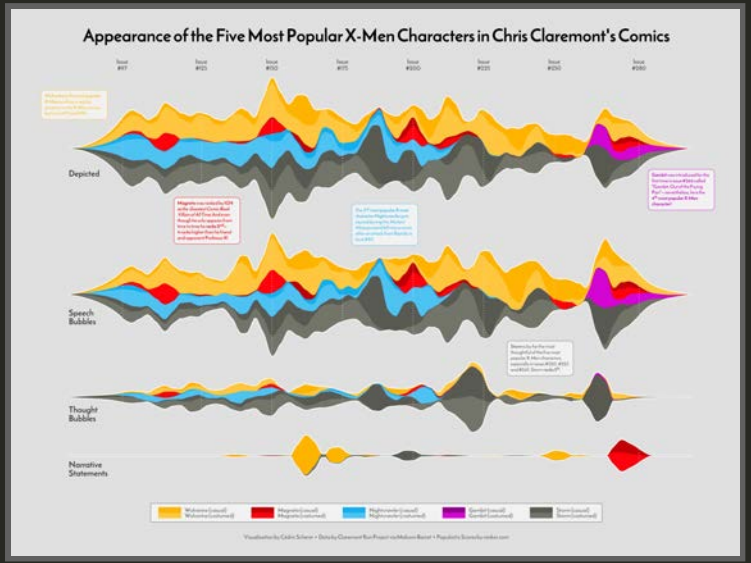


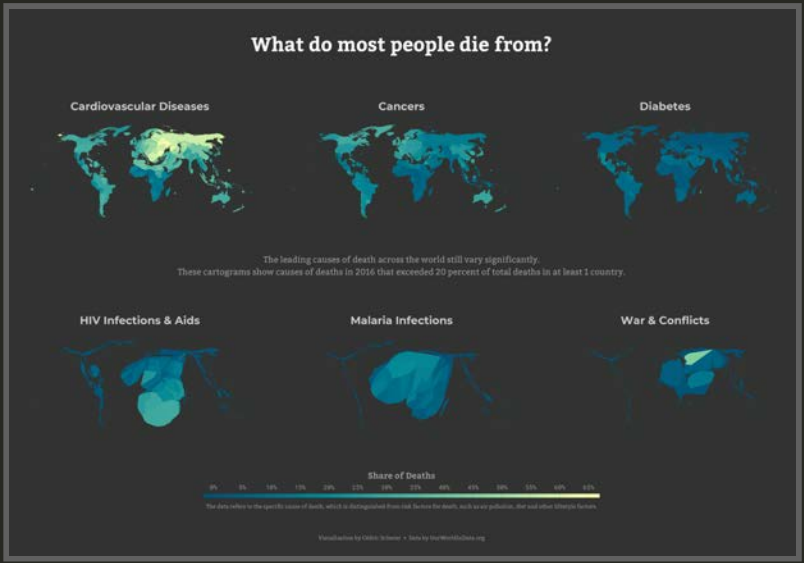


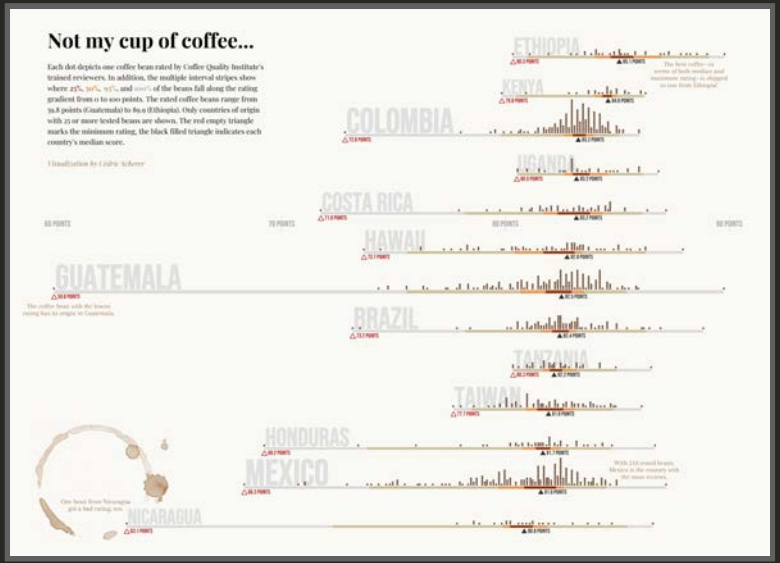


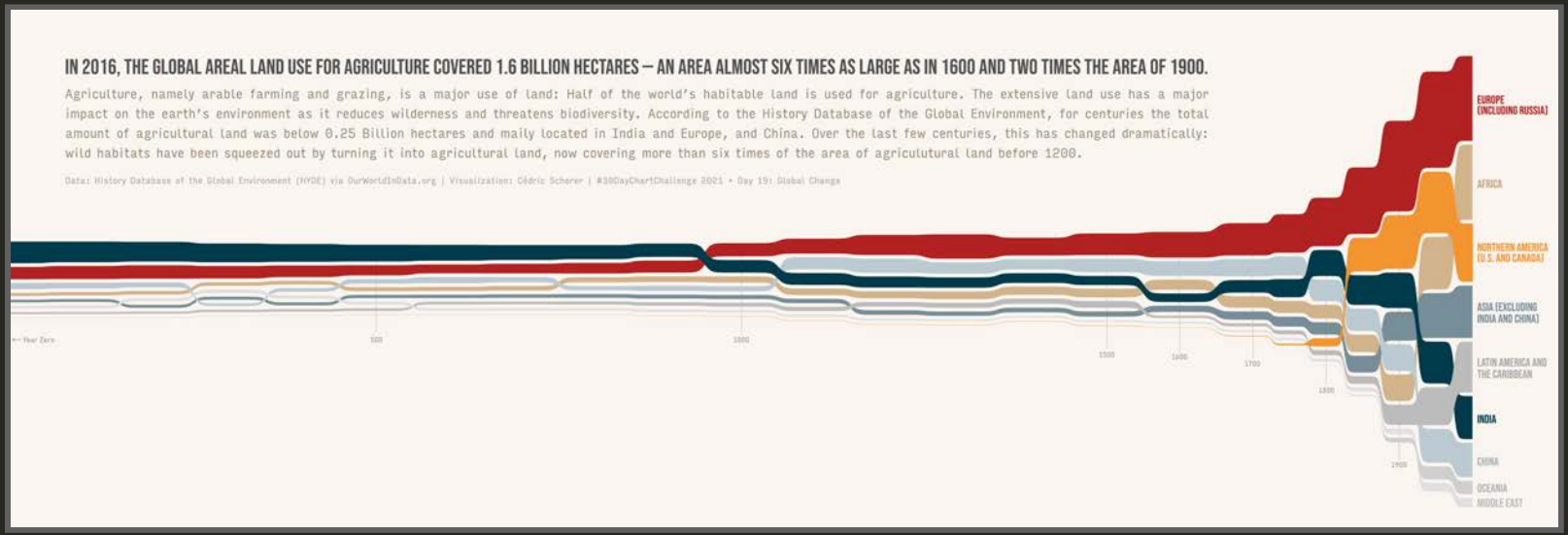


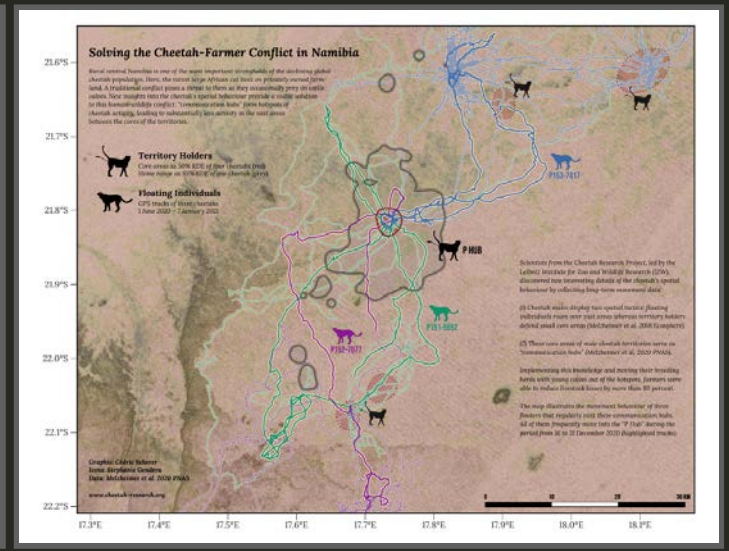
Data Visualizations as Personal Projects











`{ggplot2}` is a system for declaratively creating graphics, based on "The Grammar of Graphics" (Wilkinson, 2005).

You provide the data, tell **`{ggplot2}`** how to **map variables to aesthetics**, what graphical primitives to use, and it takes care of the details.

Advantages of `{ggplot2}`

- consistent underlying grammar of graphics (Wilkinson, 2005)

Advantages of `{ggplot2}`

- consistent underlying grammar of graphics (Wilkinson, 2005)
- very flexible, layered plot specification

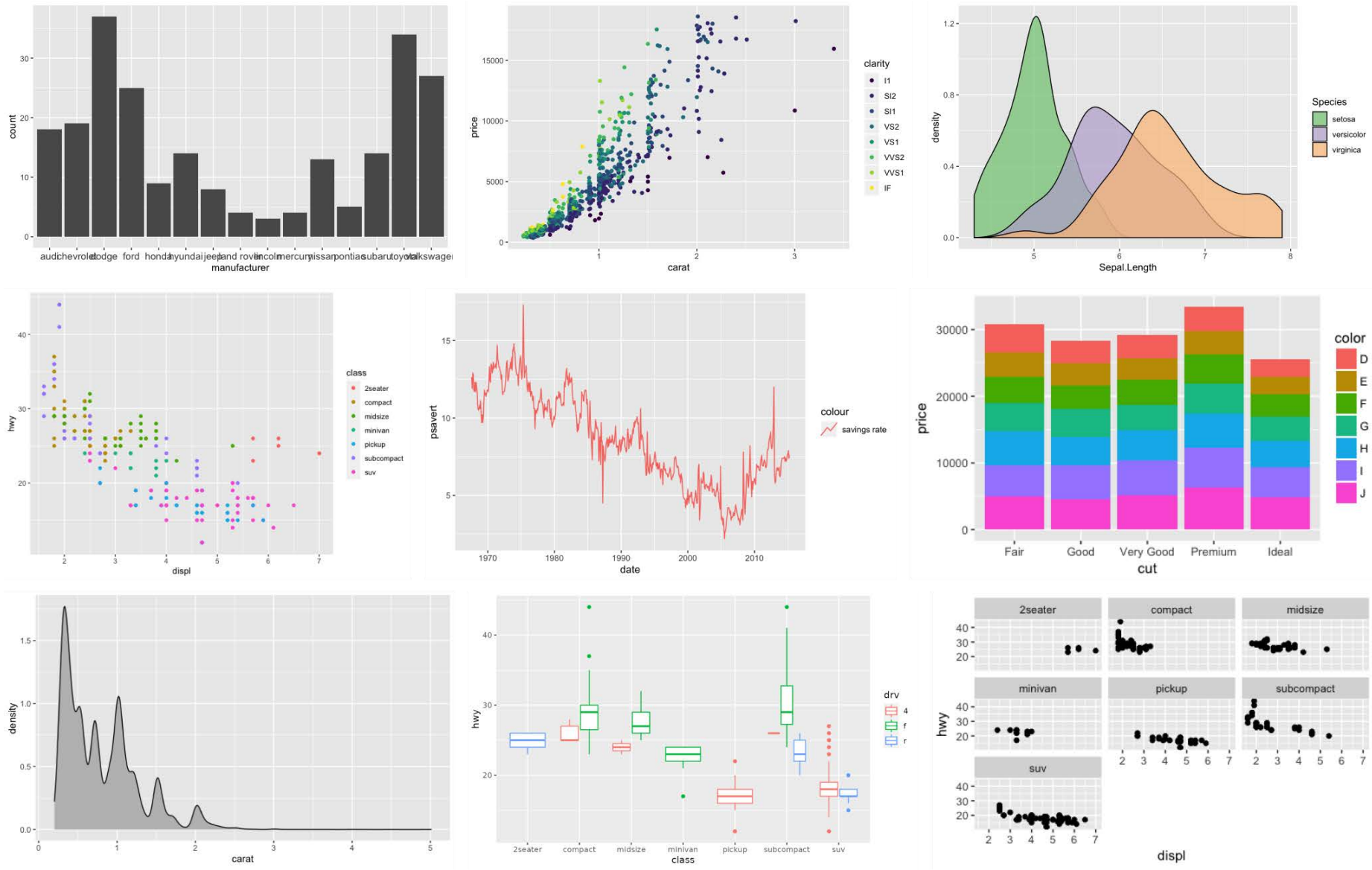
Advantages of `{ggplot2}`

- consistent underlying grammar of graphics (Wilkinson, 2005)
- very flexible, layered plot specification
- theme system for polishing plot appearance

Advantages of `{ggplot2}`

- consistent underlying grammar of graphics (Wilkinson, 2005)
- very flexible, layered plot specification
- theme system for polishing plot appearance
- active and helpful community

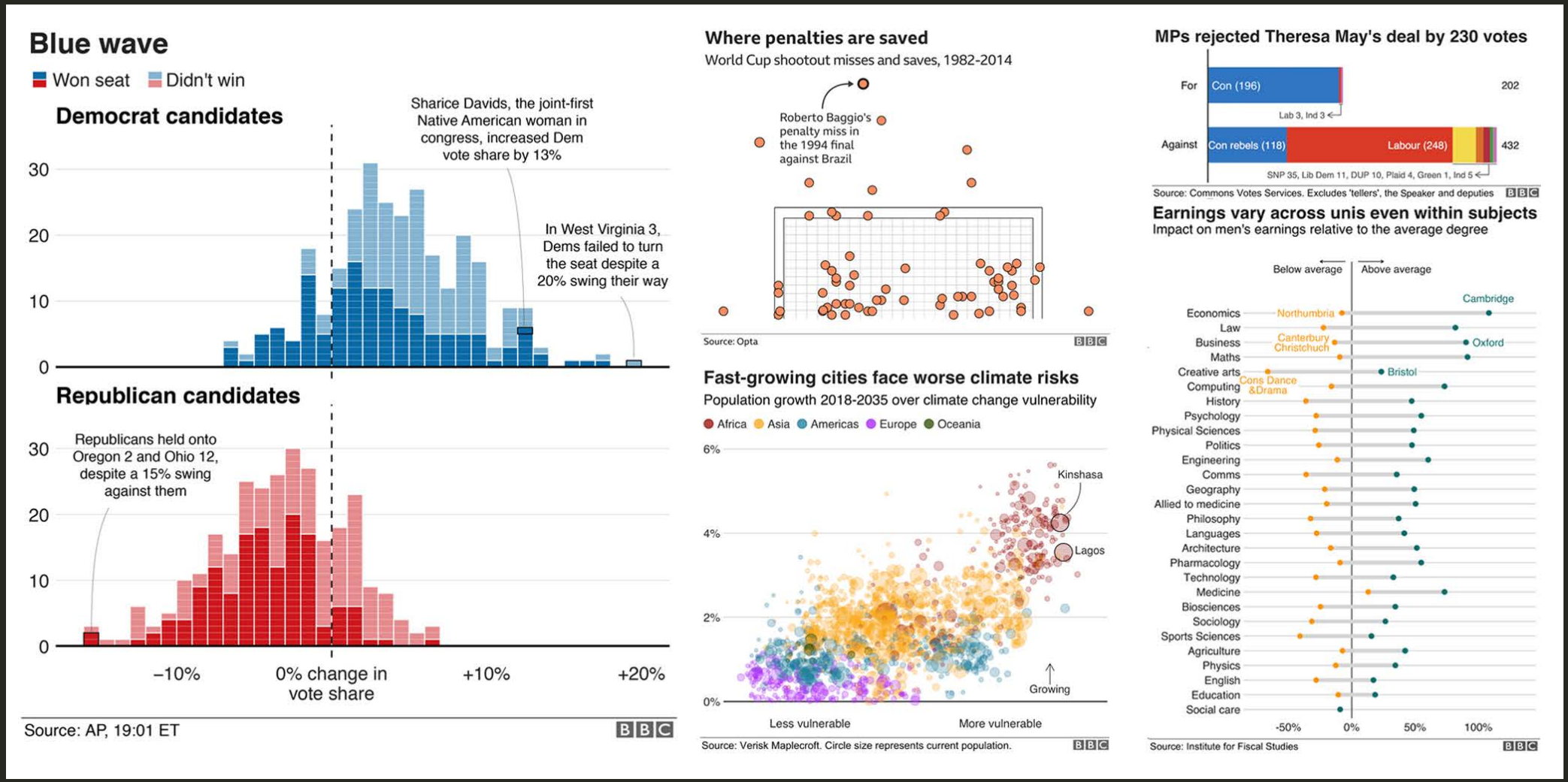
Illustration by Allison Horst



Source: Rstudio (ggplot2.tidyverse.org)

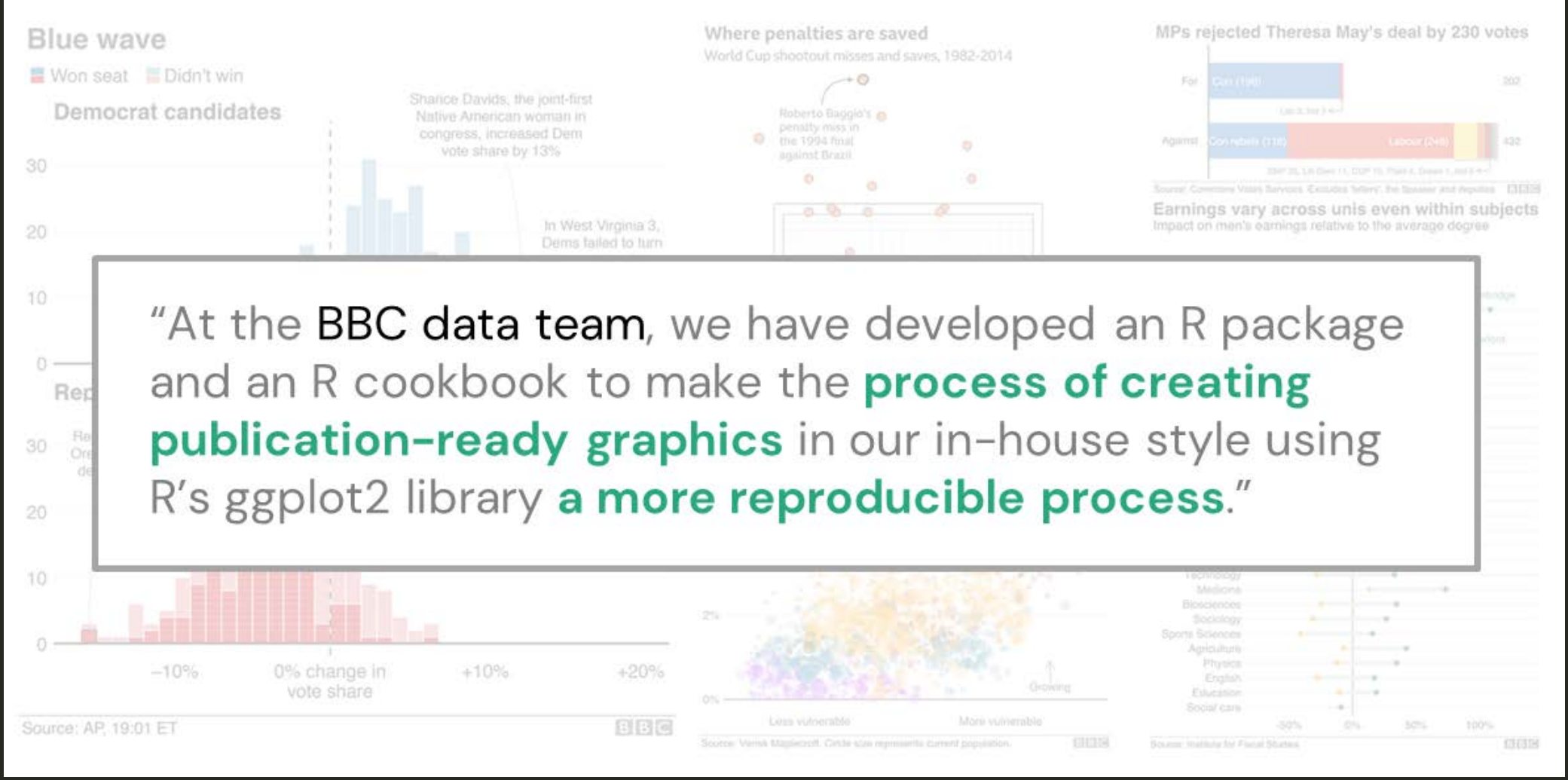
Illustration by Allison Horst

The {ggplot2} Showcase



Collection of BBC Graphics

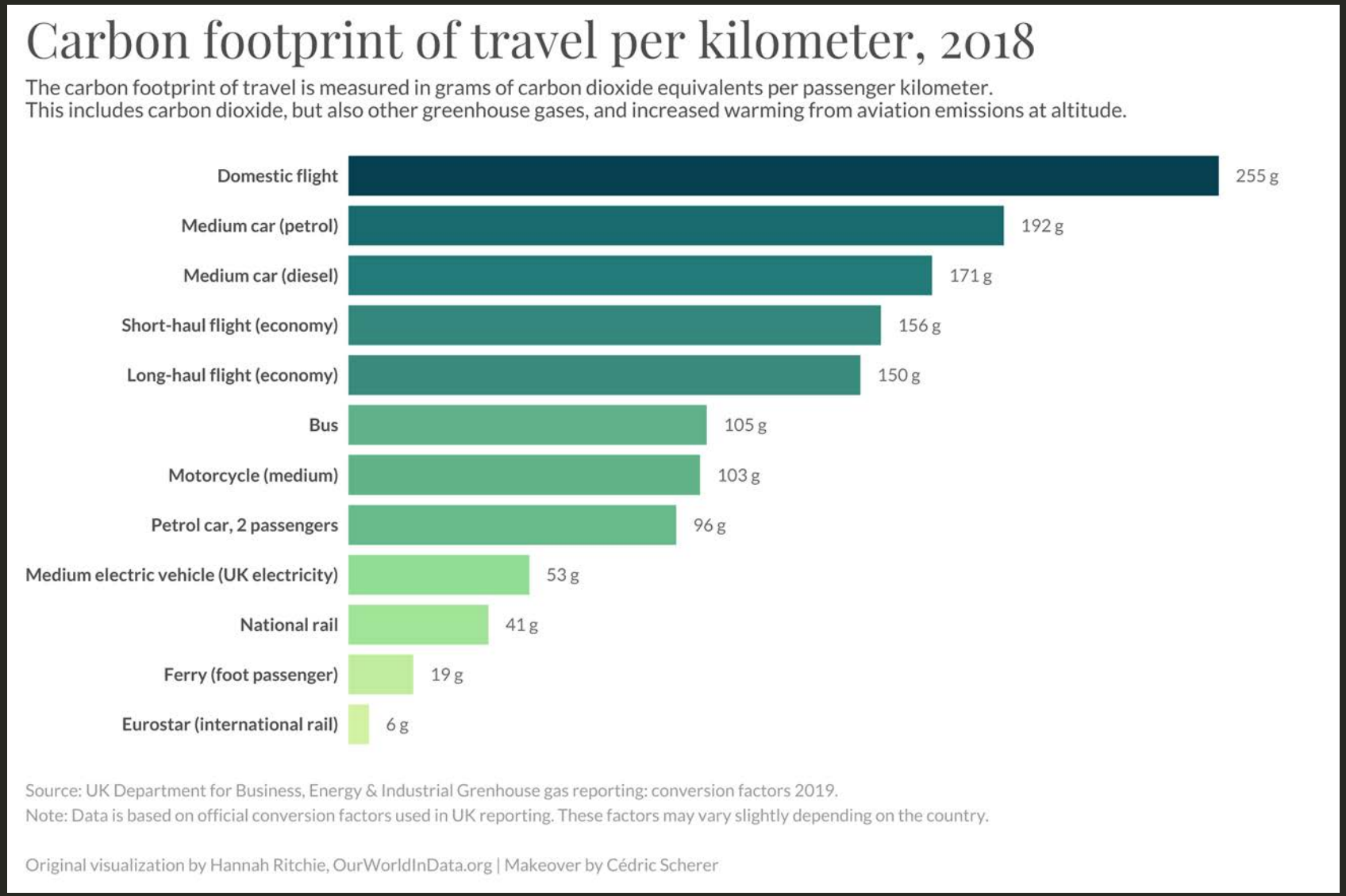
(modified from bbc.github.io/rcookbook)



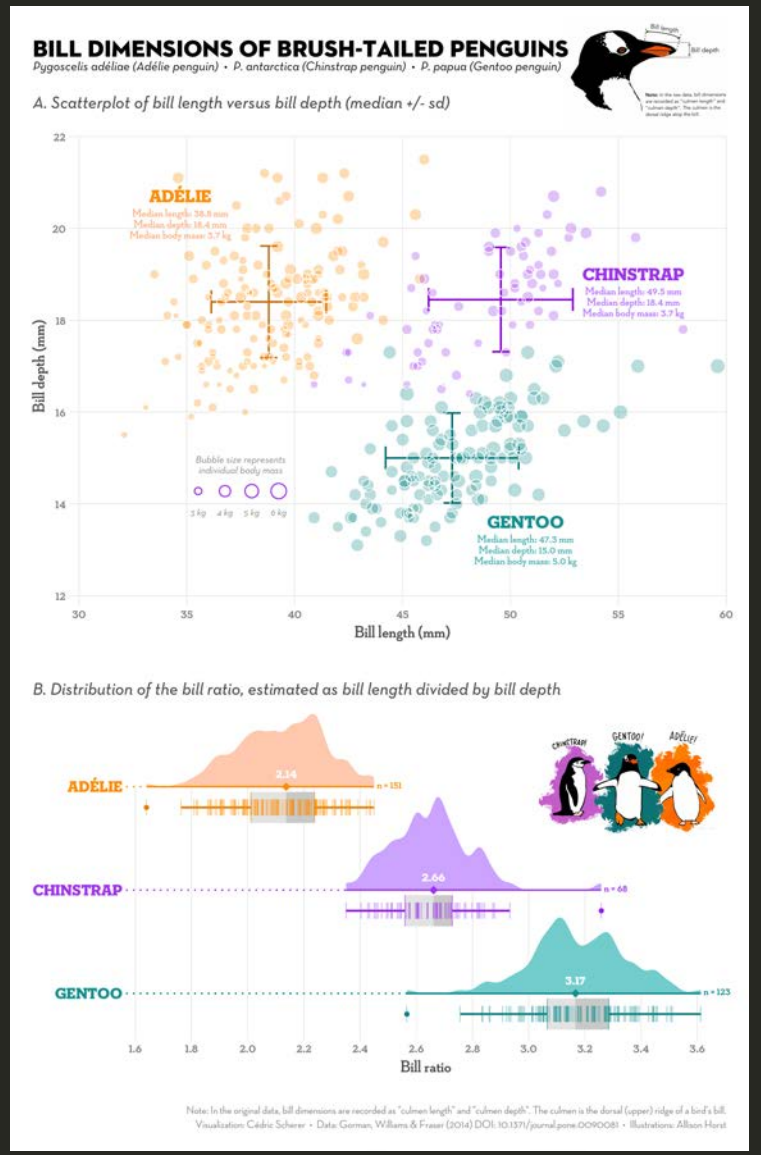
"At the **BBC data team**, we have developed an R package and an R cookbook to make the **process of creating publication-ready graphics** in our in-house style using R's ggplot2 library **a more reproducible process**."

Collection of BBC Graphics

(modified from bbc.github.io/rcookbook)



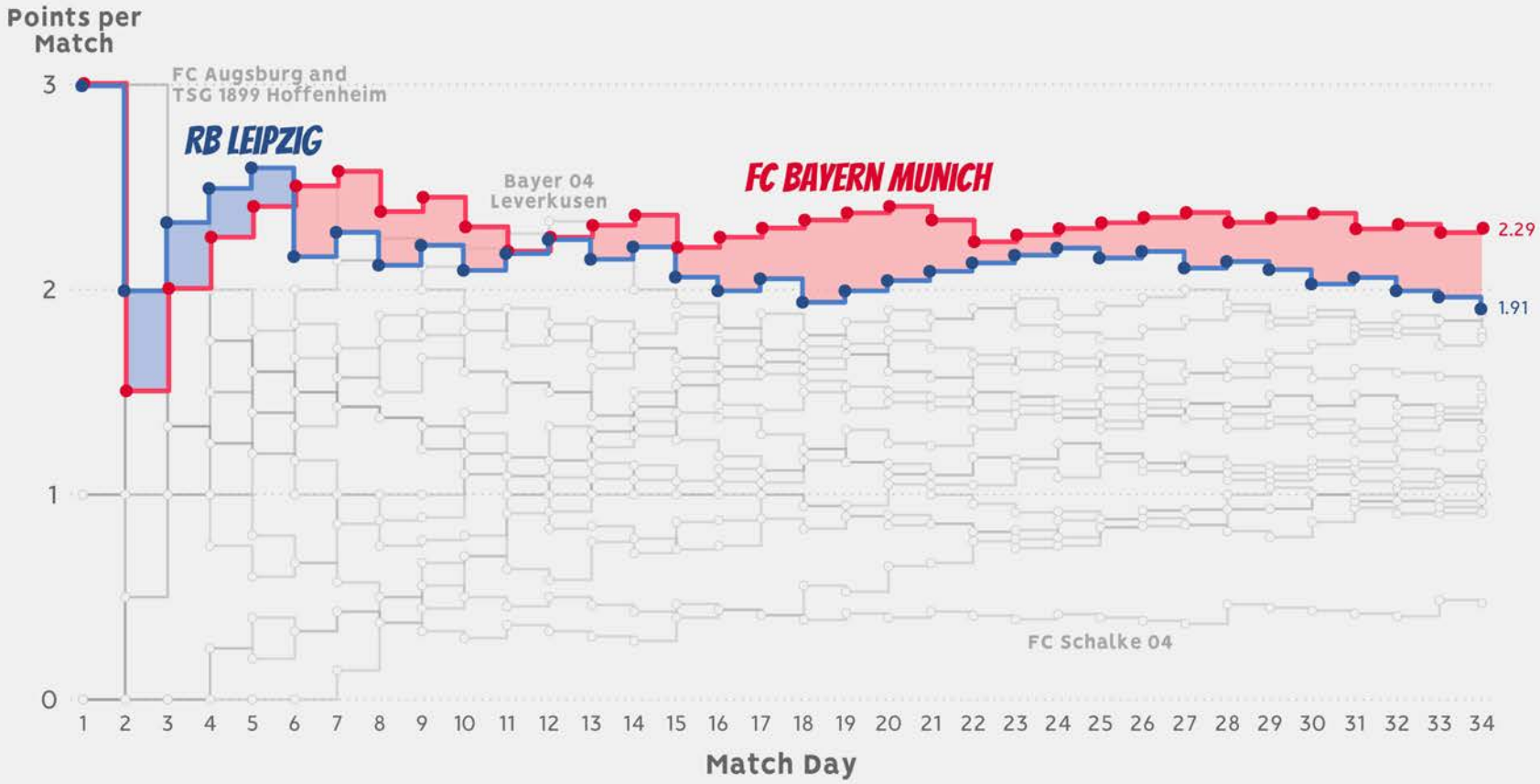
Reworked OurWorldInData Graph



Contribution to #TidyTuesday 2020/31

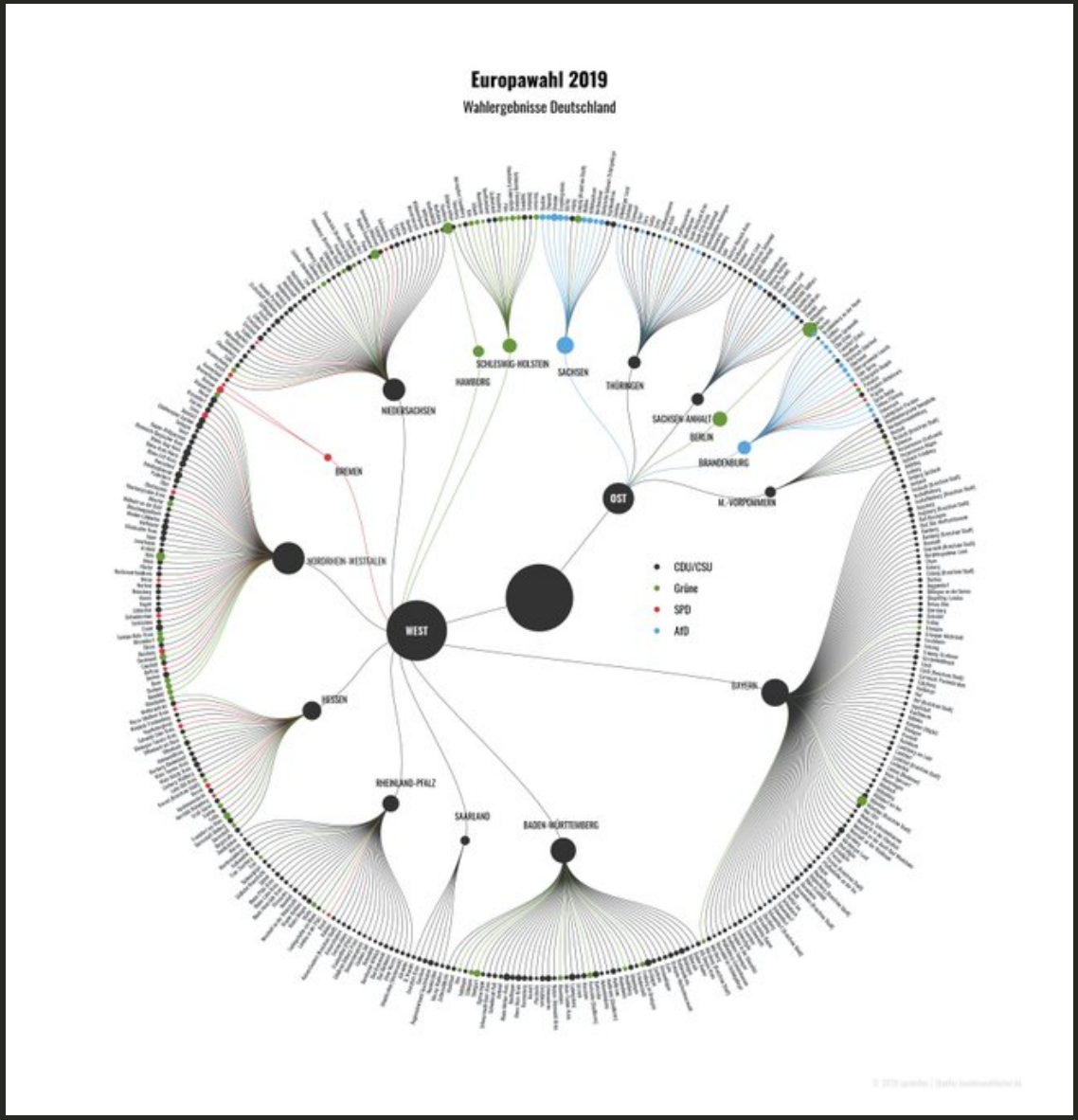
Bayern Munich's Road to the 31st Championship

FC Bayern Munich is the champion of the Bundesliga season 2020/21 with incredible 2.29 points per match, more than 0.3 points more than the best chasers. RB Leipzig (1.91 points per match) was getting close several times but has lost the trace again in the last matches.



Visualization: Cédric Scherer

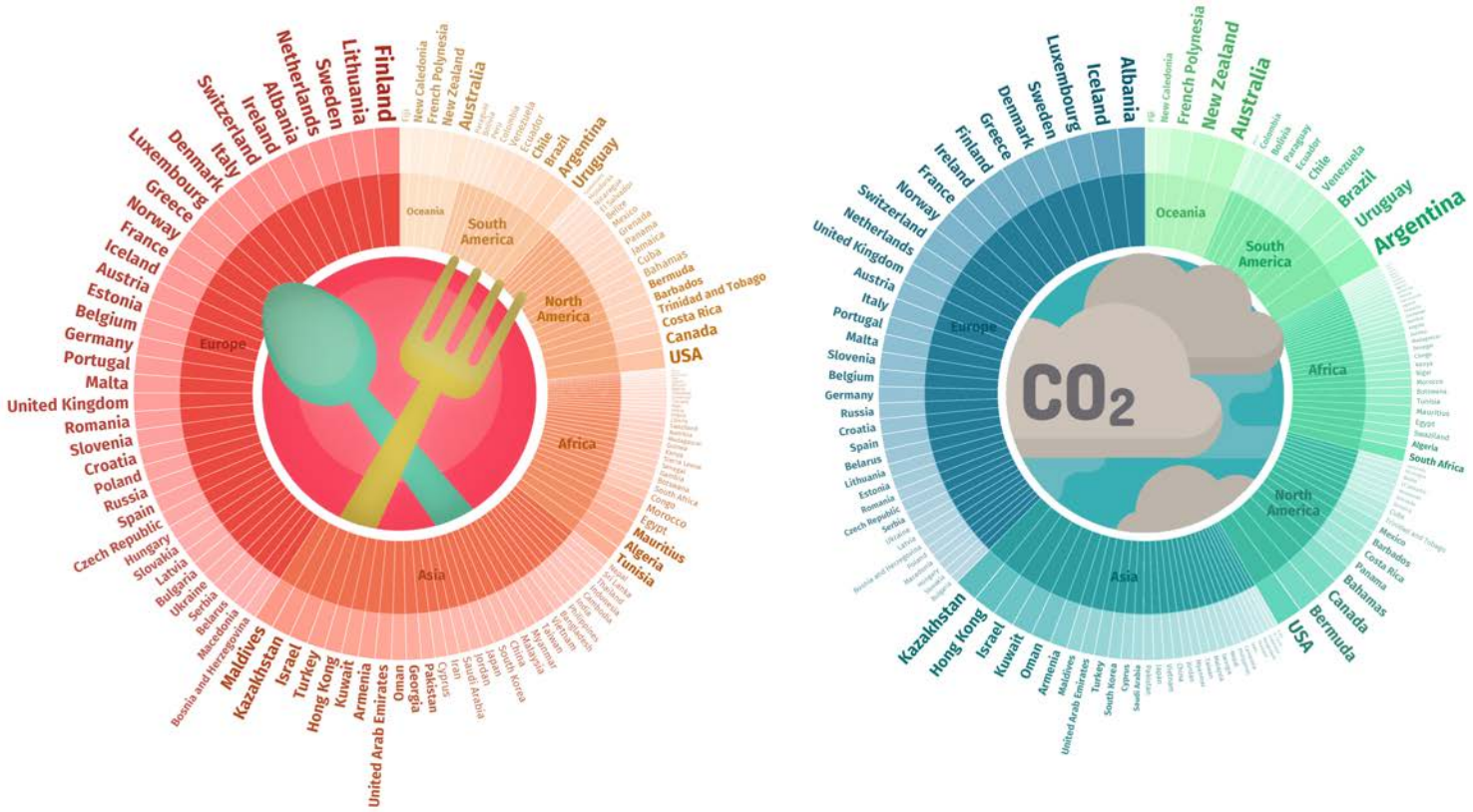
Contribution to #30DayChartChallenge 2021



European Elections by Torsten Sprenger

Food Carbon Footprint Index 2018

Global comparison of different diets in terms of **Average Consumption** (kg/person/year) of both animal and non-animal products as well as **Carbon Emissions** (kg CO_2/person/year) per continent and country. Font size and color intensity indicate each country's estimate with **countries printed in bold** belonging to the upper 50% of consumers and CO_2 emissioners, respectively.

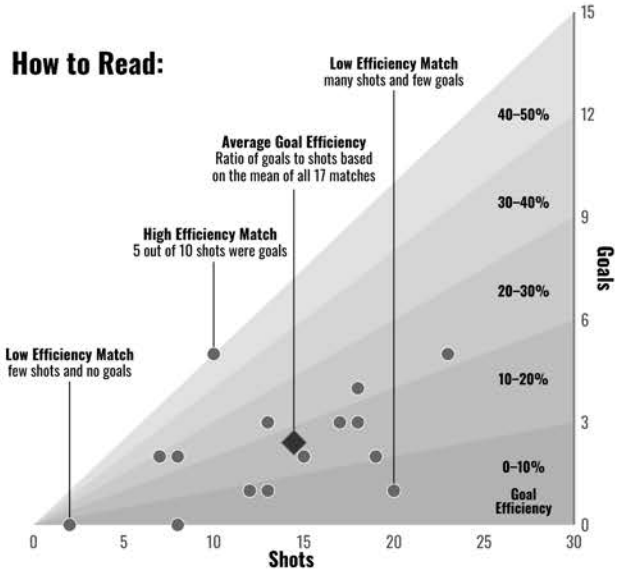


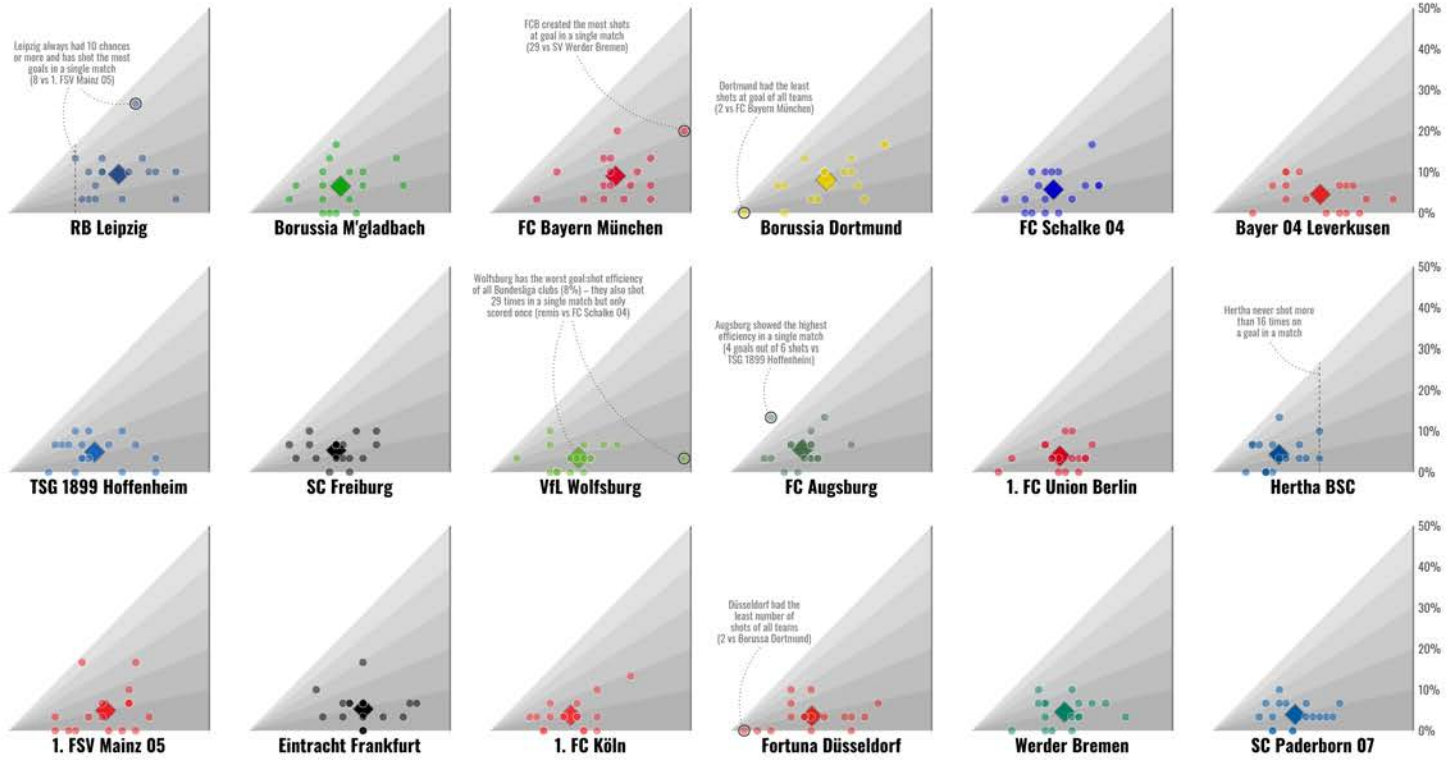
Visualization by Cédric Scherer • Data by Food and Agriculture Organization of the United Nations (FAO) via nu3 • Icons by FreePik

Contribution to #TidyTuesday

RB Leipzig and Borussia Dortmund Make the Most of Their Opportunities

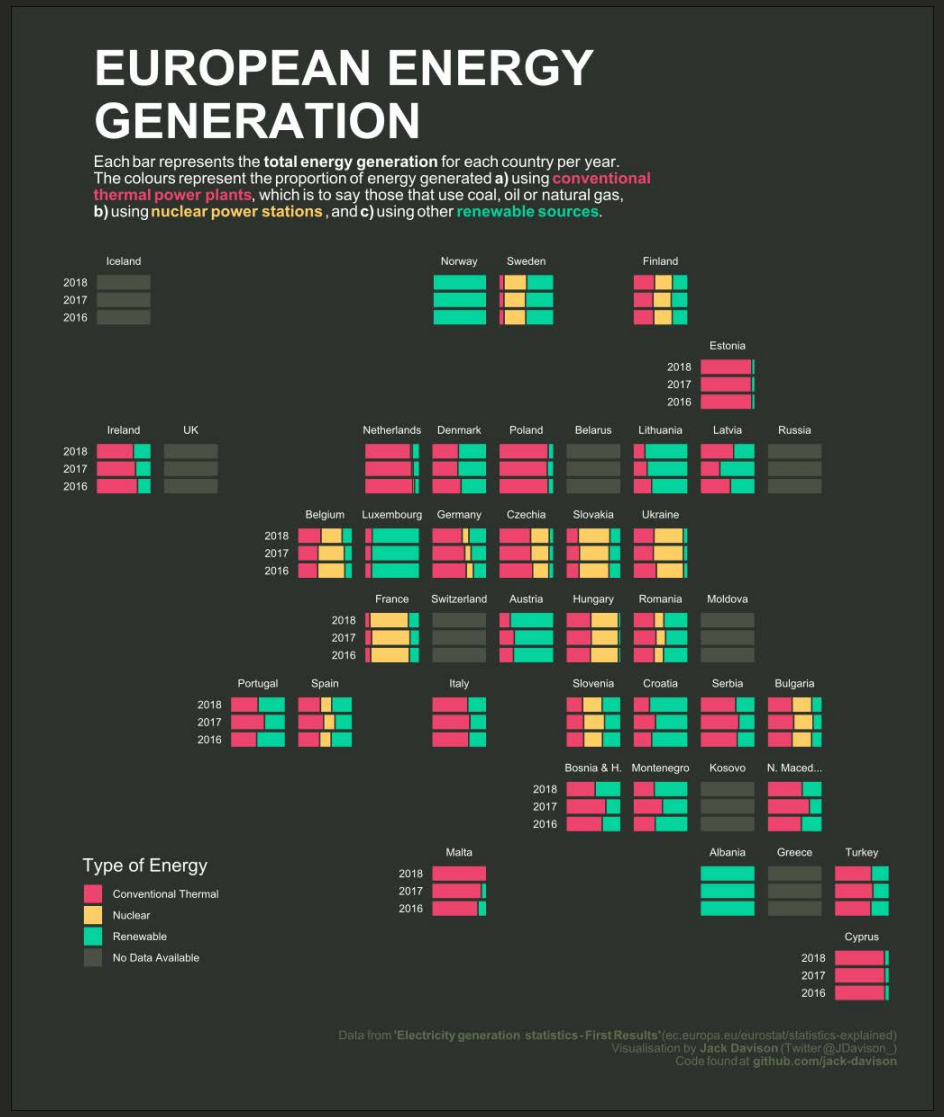
The small multiples show each club's goal:shot efficiency in the first season 2019/2020 of the 1. Bundesliga. However, while Borussia Dortmund also had very bad matches with only 2 chances, the Autumn champion RB Leipzig always scored minimum one goal and shot at least ten times on the opponent's goal in all 17 matches! Of all Bundesliga clubs, RB Leipzig also shot the most goals – 8 against Mainz.





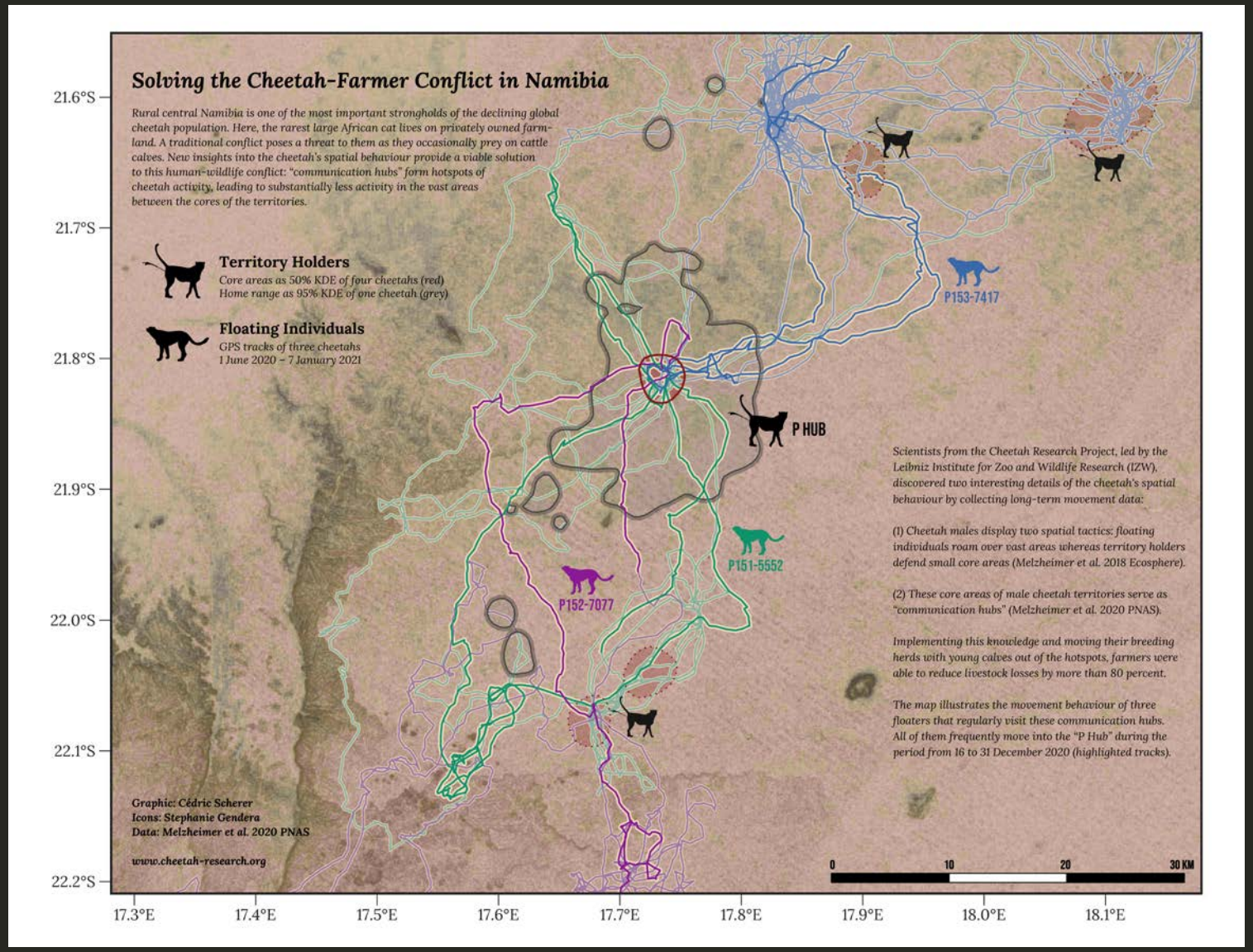
Visualization by Cédric Scherer

Contribution to #SWDchallenge

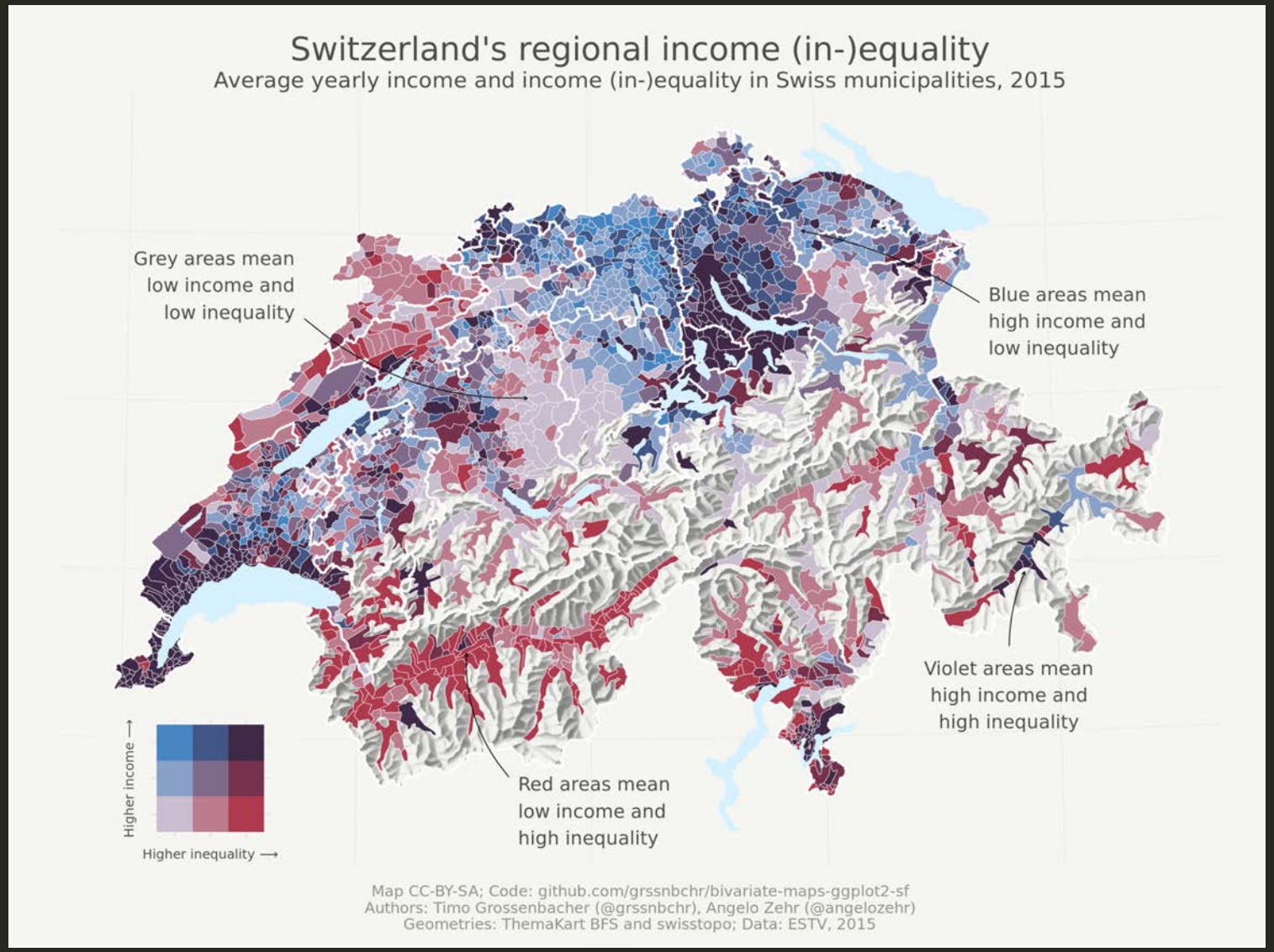


#TidyTuesday Contribution by Jack Davison

(github.com/jack-davison/tidytuesday)

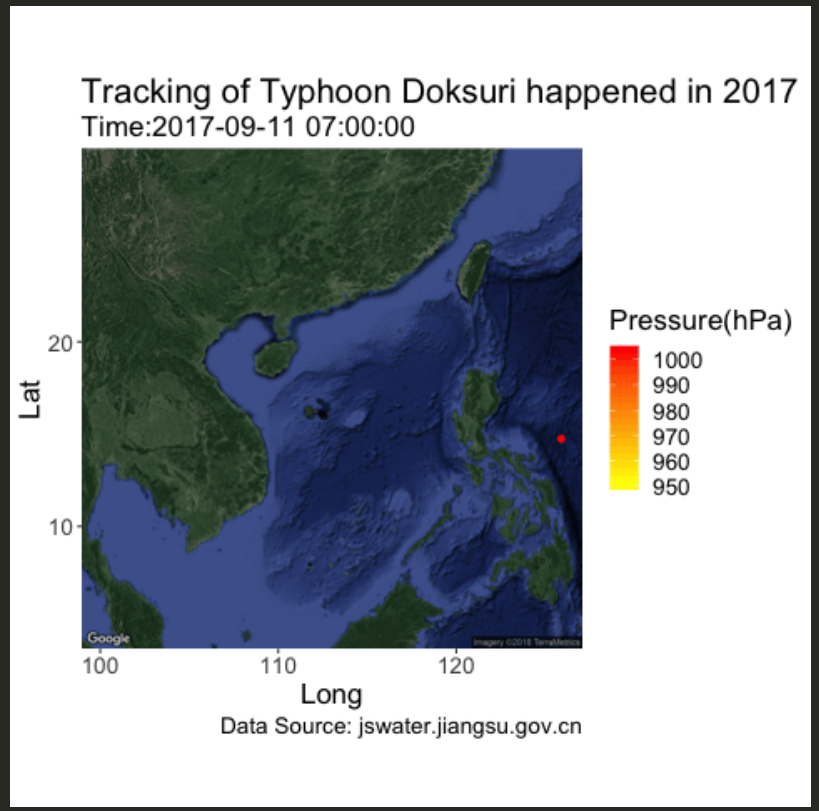


Contribution to the BES MoveMap Contest



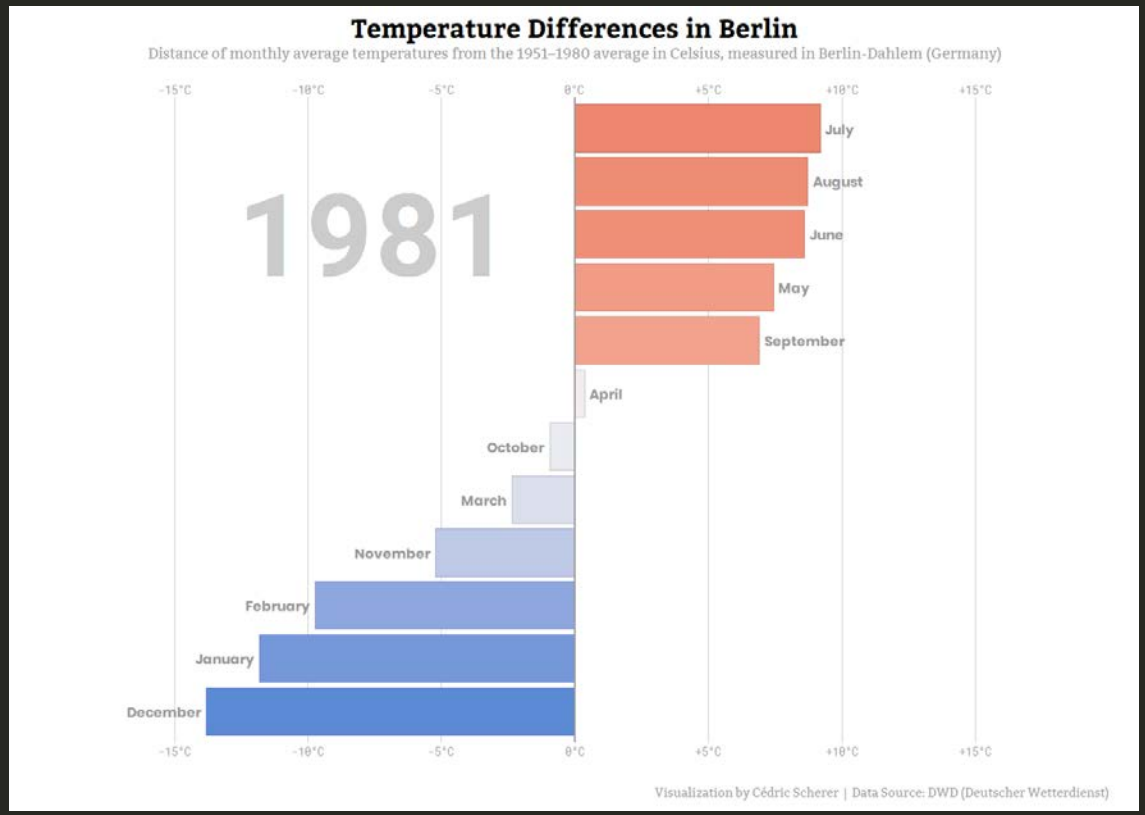
Bivariate Choropleth Map by Timo Gossenbacher

(timogrossenbacher.ch/2019/04/bivariate-maps-with-ggplot2-and-sf)

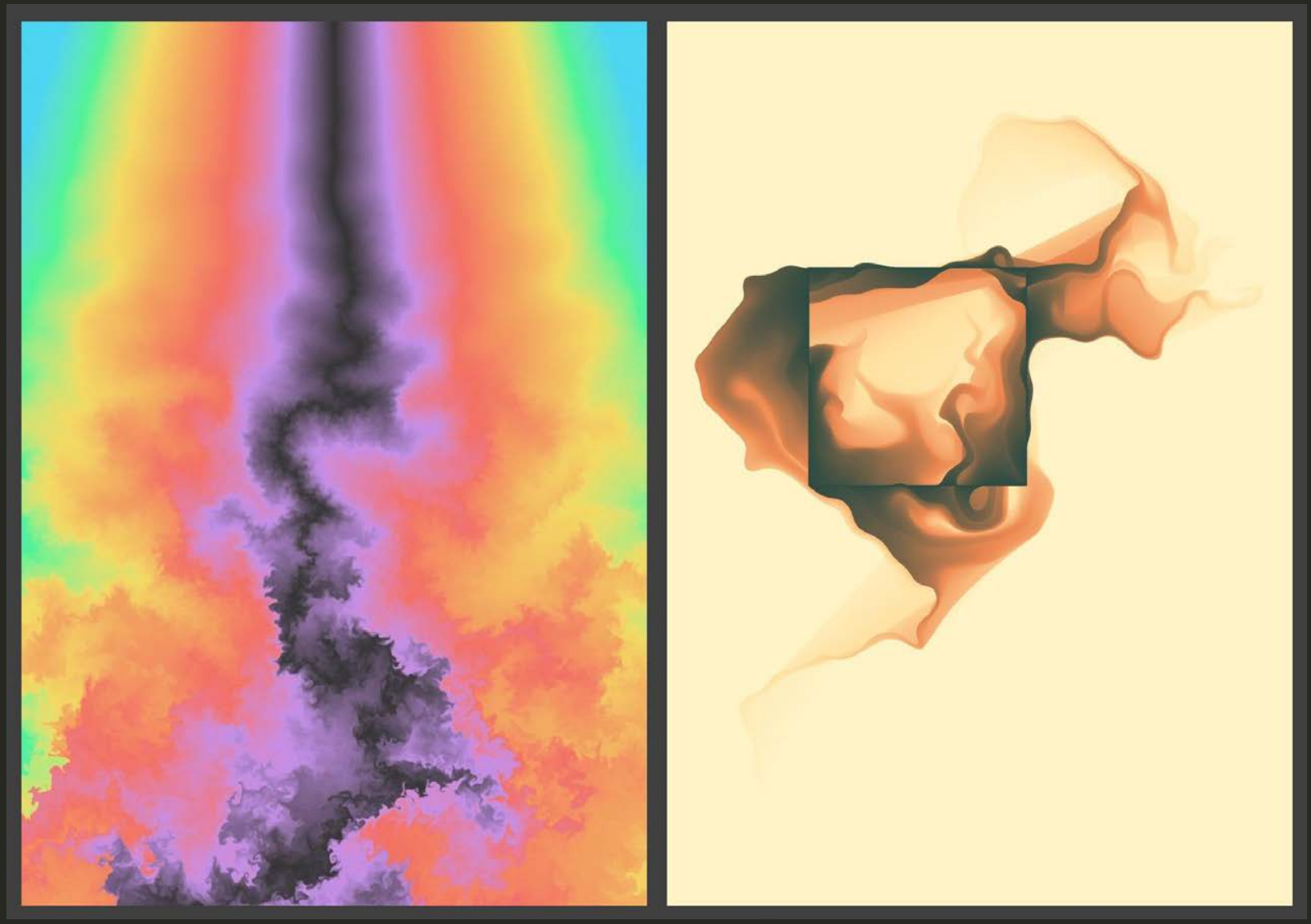


Example of the {gganimate} Package

(github.com/thomasp85/gganimate/wiki)



Personal Project

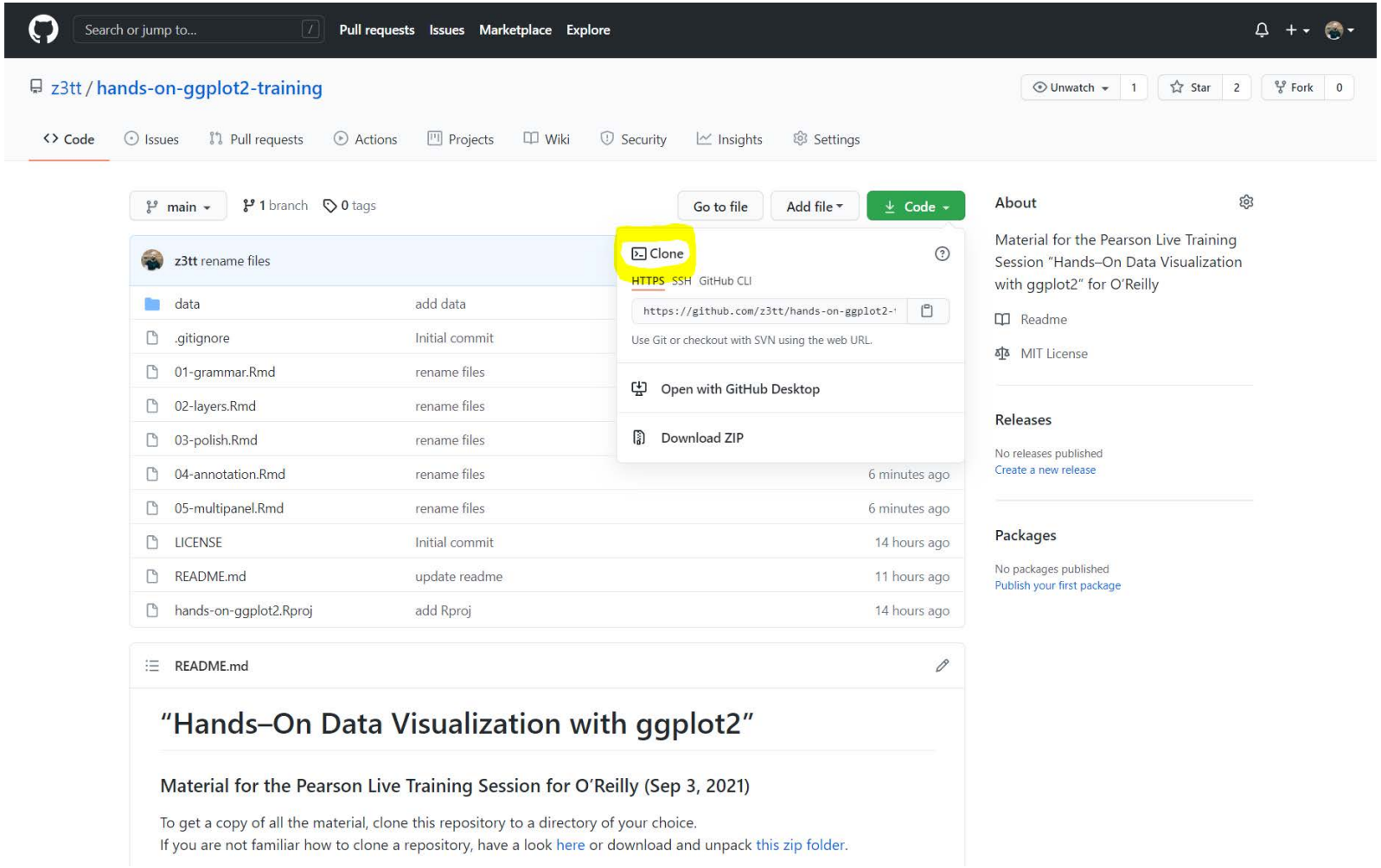


Generative art by Thomas Lin Pedersen

The Setup

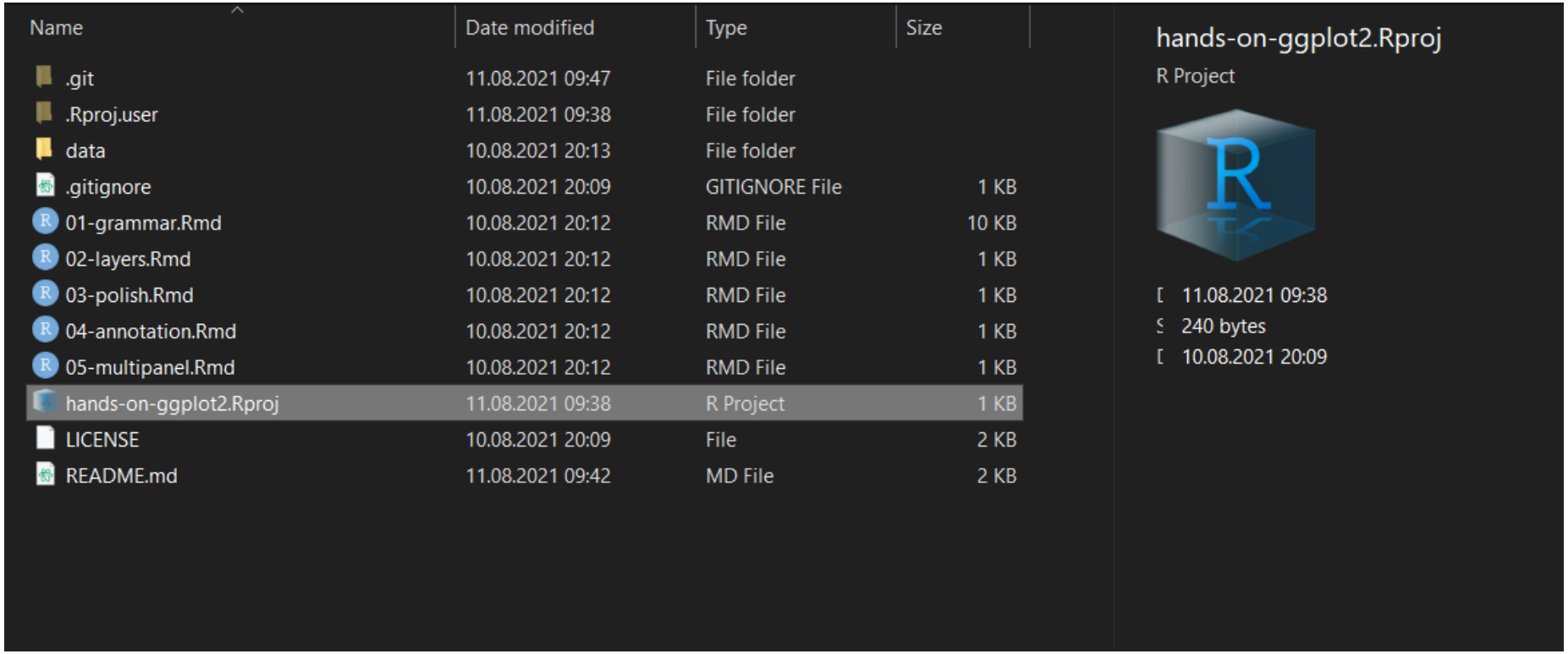
The Course Material

- **Clone the** `repository`



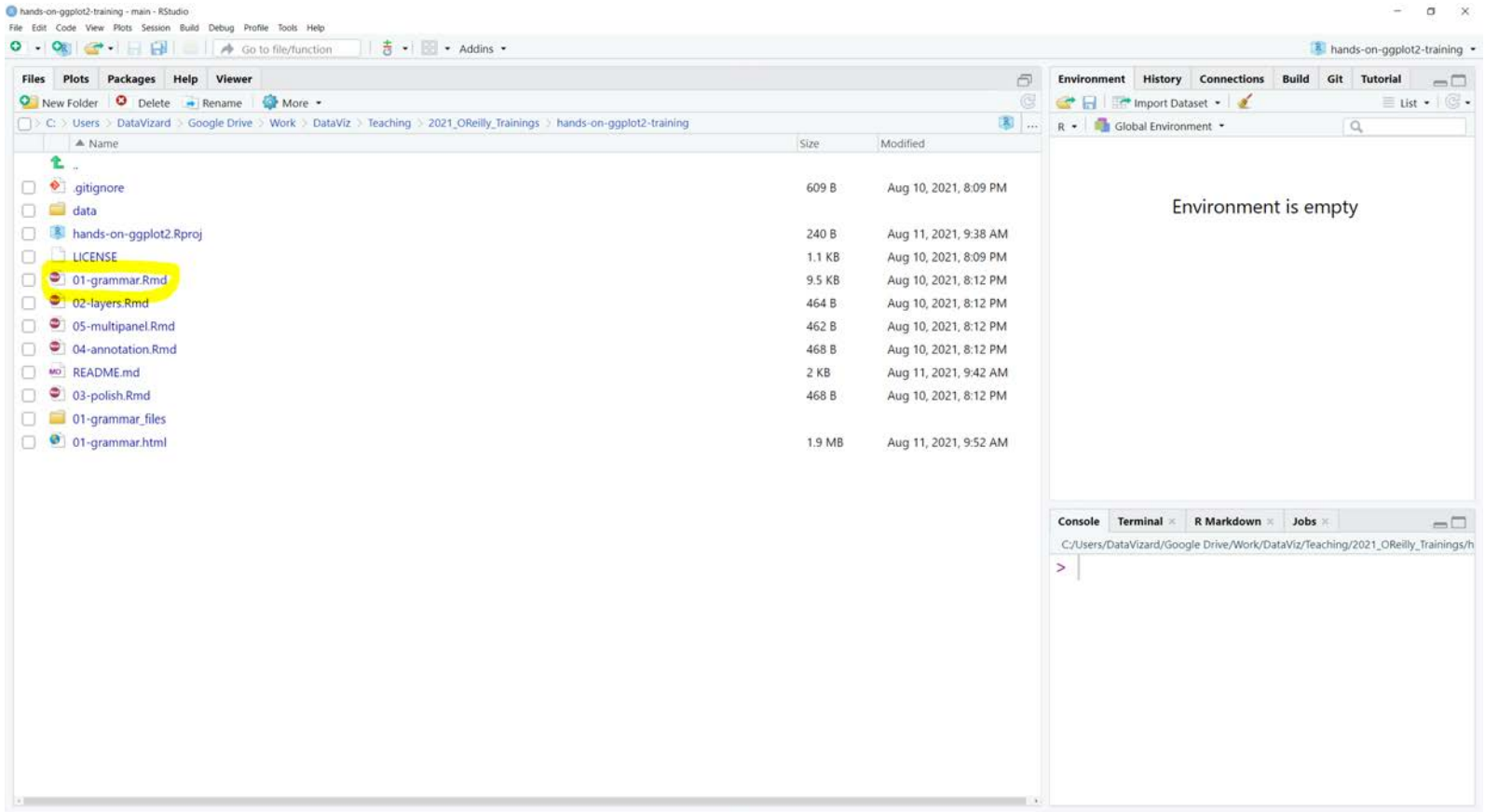
The Course Material

- Clone the `repository`.
- **Open the R Project:** `hands-on-ggplot2.Rproj`



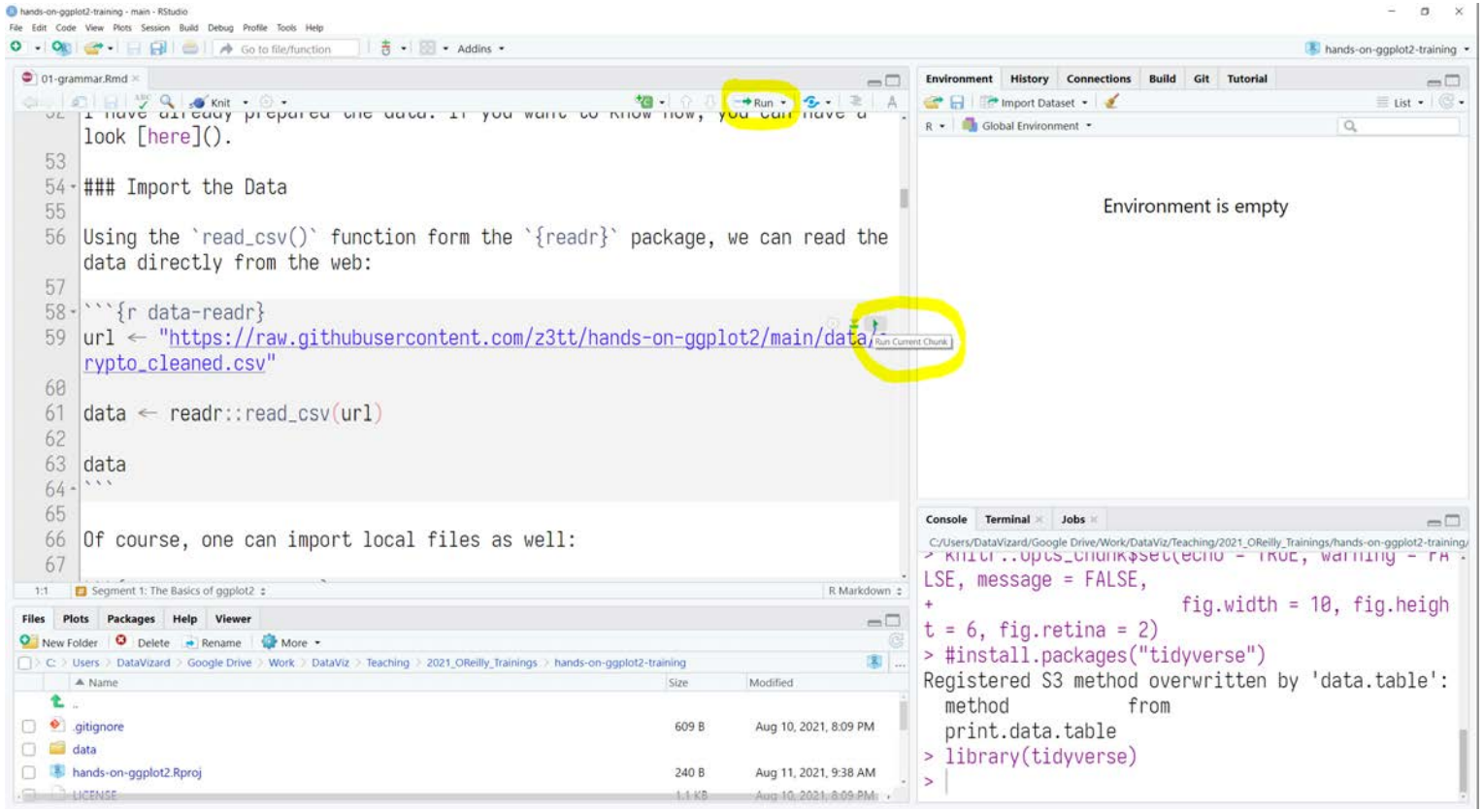
The Course Material

- Clone the `repository`.
- Open the R Project: `hands-on-ggplot2.Rproj`
- **Open the first script:** `01-grammar.Rmd`



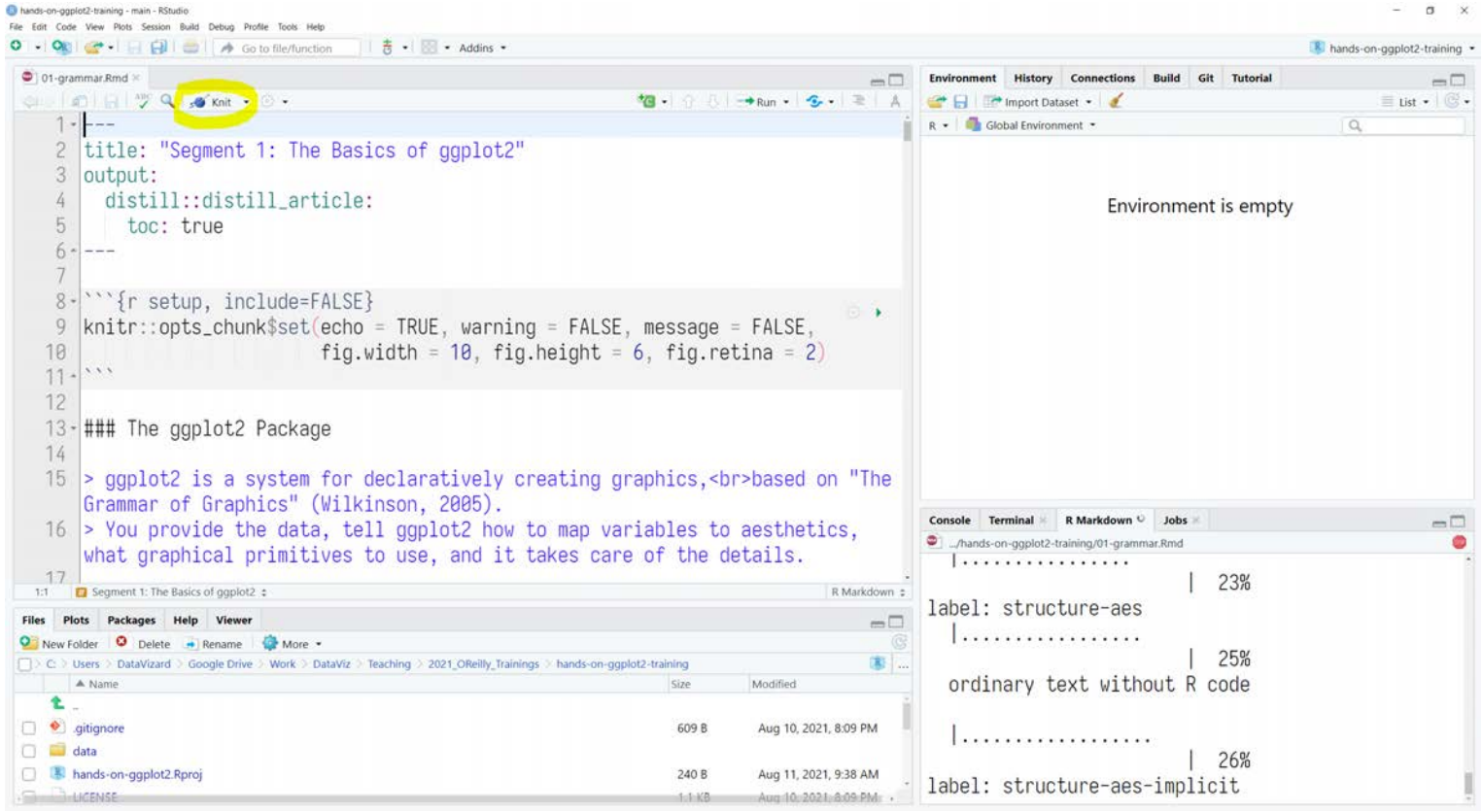
The Course Material

- Clone the `repository`.
- Open the R Project: `hands-on-ggplot2.Rproj`
- Open the first script: `01-grammar.Rmd`
- **Run code locally**



The Course Material

- Clone the `repository`.
- Open the R Project: `hands-on-ggplot2.Rproj`
- Open the first script: `01-grammar.Rmd`
- Run code locally or **knit the report**



The Course Material

- Clone the `repository`.
- Open the R Project: `hands-on-ggplot2.Rproj`
- Open the first script: `01-grammar.Rmd`
- Run code locally or **knit the report**

POLL: Do you use Rmarkdown reports?

- No, never heard of it.
- No, but I am aware they exist.
- Yes, rarely.
- Yes, extensively.

The Package

{ggplot2} is a **data visualization package** for the programming language R created by Hadley Wickham.

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

{ggplot2} is part of the tidyverse, a set of packages to manipulate and explore data.

```
install.packages("tidyverse")
library(tidyverse)
```

Source: dicook.org/files/rstudio/#3

The Data Set

We use **cryptocurrency financial data**, pulled from `CoinMarketCap.com`.

Data Explorer

2.17 MB

consolidated_coin_data.csv

Summary

1 file

8 columns

consolidated_coin_data.csv (2.17 MB)

Detail | Compact | Column | 8 of 8 columns

Currency	Date	Open	High	Low	
Name of currency	Date refers to the calendar date for the particular row - 24 hours midnight to midnight	Open is what the price was at the beginning of the day	Highest recorded trading price of the day	Lowest recorded trading price of the day	Cl w
12 unique values	28Apr13 – 4Dec19	0 – 19.5k	0 – 20.1k	0 – 19k	0
tezos	Aug 03, 2019	1.43	1.46	1.38	1.
tezos	Aug 02, 2019	1.41	1.47	1.37	1.
tezos	Aug 01, 2019	1.27	1.45	1.25	1.
tezos	Jul 31, 2019	1.22	1.29	1.21	1.
tezos	Jul 30, 2019	1.01	1.27	1.00	1.
tezos	Jul 29, 2019	1.02	1.04	1.01	1.
tezos	Jul 28, 2019	1.01	1.02	0.983322	1.
tezos	Jul 27, 2019	1.04	1.06	0.992433	1.
tezos	Jul 26, 2019	1.03	1.04	1.01	1.
tezos	Jul 25, 2019	1.01	1.06	0.999385	1.
tezos	Jul 24, 2019	1.07	1.13	0.984626	1.
tezos	Jul 23, 2019	1.12	1.12	1.00	1.
tezos	Jul 22, 2019	1.08	1.15	1.08	1.

We limit the data to the period 08/2017–12/2019 and the top 4 cryptocurrencies:

POLL: Are you into cryptocurrencies?

- No, I don't care.
- Yes, I'm keeping an eye on that new technology (passively).
- Yes, I am part of the movement.

Import the Data

Import the Data

Using the `read_csv()` function form the `{readr}` package, we can read the data directly from the web:

```
url <- "https://raw.githubusercontent.com/z3tt/hands-on-ggplot2/main/data/crypto_cleaned.csv"

data <- readr::read_csv(url)
```

Short Explanation of `::`

The so–called **namespace** allows to access functions from a package directly without loading it first.

```
packagename::function(argument)
```

Furthermore, it helps readers to understand from which package a function is imported from.

Import the Data

Using the `read_csv()` function form the `{readr}` package, we can read the data directly from the web:

```
url <- "https://raw.githubusercontent.com/z3tt/hands-on-ggplot2/main/data/crypto_cleaned.csv"

data <- readr::read_csv(url)
```

```
data
```

```
## # A tibble: 2,812 x 9
##    currency     date        open  high   low close  year month  yday
##    <chr>        <date>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 binance-coin 2019-12-04  15.4  15.7  15.0  15.3  2019    12   338
##  2 binance-coin 2019-12-03  15.2  15.6  15.0  15.3  2019    12   337
##  3 binance-coin 2019-12-02  15.5  15.7  15.2  15.2  2019    12   336
##  4 binance-coin 2019-12-01  15.7  15.7  15.0  15.5  2019    12   335
##  5 binance-coin 2019-11-30  16.3  16.4  15.5  15.7  2019    11   334
##  6 binance-coin 2019-11-29  15.7  16.3  15.6  16.3  2019    11   333
##  7 binance-coin 2019-11-28  16.1  16.2  15.6  15.7  2019    11   332
##  8 binance-coin 2019-11-27  15.5  16.2  14.9  16.1  2019    11   331
##  9 binance-coin 2019-11-26  15.3  15.9  15.2  15.5  2019    11   330
## 10 binance-coin 2019-11-25  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 2,802 more rows
```

Import the Data

Of course, one can import local files as well:

```
data_local <- readr::read_csv("data/crypto_cleaned.csv")

data_local
## # A tibble: 2,812 x 9
##    currency     date        open  high   low close  year month  yday
##    <chr>        <date>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 binance-coin 2019-12-04  15.4  15.7  15.0  15.3  2019    12   338
##  2 binance-coin 2019-12-03  15.2  15.6  15.0  15.3  2019    12   337
##  3 binance-coin 2019-12-02  15.5  15.7  15.2  15.2  2019    12   336
##  4 binance-coin 2019-12-01  15.7  15.7  15.0  15.5  2019    12   335
##  5 binance-coin 2019-11-30  16.3  16.4  15.5  15.7  2019    11   334
##  6 binance-coin 2019-11-29  15.7  16.3  15.6  16.3  2019    11   333
##  7 binance-coin 2019-11-28  16.1  16.2  15.6  15.7  2019    11   332
##  8 binance-coin 2019-11-27  15.5  16.2  14.9  16.1  2019    11   331
##  9 binance-coin 2019-11-26  15.3  15.9  15.2  15.5  2019    11   330
## 10 binance-coin 2019-11-25  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 2,802 more rows
```

This assumes that you have placed the file in a folder called `data` in your working directory.
You can specify this directory via `setwd()` or, and preferably, use R projects.

Exercise 1:

- Import the data set on Chicago weather data: `https://raw.githubusercontent.com/z3tt/ggplot-courses/master/data/chicago-nmmaps.csv`
- Inspect the data.

Exercise 1: Import the Data

```
chic <- readr::read_csv(
  "https://raw.githubusercontent.com/z3tt/ggplot-courses/master/data/chicago-nmmaps.csv"
)

# chic <- readr::read_csv("data/chicago-nmmaps.csv")
```

Exercise 1: Inspect the Data

```
tibble::glimpse(chic)
## Rows: 1,461
## Columns: 10
## $ city     <chr> "chic", "chic", "chic", "chic", "chic", "chic", "chic", "chic~
## $ date     <date> 1997-01-01, 1997-01-02, 1997-01-03, 1997-01-04, 1997-01-05, ~
## $ death    <dbl> 137, 123, 127, 146, 102, 127, 116, 118, 148, 121, 110, 127, 1~
## $ temp     <dbl> 36.0, 45.0, 40.0, 51.5, 27.0, 17.0, 16.0, 19.0, 26.0, 16.0, 1~
## $ dewpoint <dbl> 37.500, 47.250, 38.000, 45.500, 11.250, 5.750, 7.000, 17.750,~
## $ pm10     <dbl> 13.052268, 41.948600, 27.041751, 25.072573, 15.343121, 9.3646~
## $ o3       <dbl> 5.659256, 5.525417, 6.288548, 7.537758, 20.760798, 14.940874,~
## $ time     <dbl> 3654, 3655, 3656, 3657, 3658, 3659, 3660, 3661, 3662, 3663, 3~
## $ season   <chr> "Winter", "Winter", "Winter", "Winter", "Winter", "Winter", "~
## $ year     <dbl> 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1~
```

Exercise 1: Inspect the Data

```
summary(chic)
##      city                 date                 death             temp
##  Length:1461        Min.   :1997-01-01   Min.   : 69.0   Min.   :-3
##  Class :character   1st Qu.:1998-01-01   1st Qu.:101.0   1st Qu.:36
##  Mode  :character   Median :1999-01-01   Median :110.0   Median :52
##                     Mean   :1999-01-01   Mean   :111.3   Mean   :51
##                     3rd Qu.:2000-01-01   3rd Qu.:120.0   3rd Qu.:67
##                     Max.   :2000-12-31   Max.   :164.0   Max.   :90
##
##     dewpoint            pm10                 o3                time
##  Min.   :-10.38   Min.   :  0.2033   Min.   : 0.08584   Min.   :3654
##  1st Qu.: 28.10   1st Qu.: 18.3844   1st Qu.:10.92419   1st Qu.:4019
##  Median : 41.20   Median : 28.8499   Median :18.65149   Median :4384
##  Mean   : 41.36   Mean   : 32.0928   Mean   :19.23597   Mean   :4384
##  3rd Qu.: 56.25   3rd Qu.: 41.0738   3rd Qu.:26.20476   3rd Qu.:4749
##  Max.   : 77.10   Max.   :125.3844   Max.   :54.68764   Max.   :5114
##                   NA's   :17
##     season               year
##  Length:1461        Min.   :1997
##  Class :character   1st Qu.:1998
##  Mode  :character   Median :1999
##                     Mean   :1999
##                     3rd Qu.:2000
```

Exercise 1: Inspect the Data

```
range(chic$date)
## [1] "1997-01-01" "2000-12-31"

range(chic$temp)
## [1] -3 90

unique(chic$season)
## [1] "Winter" "Spring" "Summer" "Autumn"

unique(chic$year)
## [1] 1997 1998 1999 2000
```

The Structure of {ggplot2}

"The Grammar of Graphics"

The Structure of {ggplot2}

Layer	Function	Explanation
Data	`ggplot(data)`	The raw data that you want to visualise.
Aesthetics	`aes()`	Aesthetic mappings of the geometric and statistical objects.
Layers	`geom_*()` and `stat_*()`	The geometric shapes and statistical summaries representing the data.

The Structure of `{ggplot2}`

Layer	Function	Explanation
Data	`ggplot(data)`	The raw data that you want to visualise.
Aesthetics	`aes()`	Aesthetic mappings of the geometric and statistical objects.
Layers	`geom_*()` and `stat_*()`	The geometric shapes and statistical summaries representing the data.
Scales	`scale_*()`	Maps between the data and the aesthetic dimensions.
Coordinate System	`coord_*()`	Maps data into the plane of the data rectangle.
Facets	`facet_*()`	The arrangement of the data into a grid of plots.
Visual Themes	`theme()` and `theme_*()`	The overall visual defaults of a plot.

The Structure of {ggplot2}

Layer	Function	Explanation
Data	`ggplot(data)`	The raw data that you want to visualise.
Aesthetics	`aes()`	Aesthetic mappings of the geometric and statistical objects.
Layers	`geom_*()` and `stat_*()`	The geometric shapes and statistical summaries representing the data.
Scales	`scale_*()`	Maps between the data and the aesthetic dimensions.
Coordinate System	`coord_*()`	Maps data into the plane of the data rectangle.
Facets	`facet_*()`	The arrangement of the data into a grid of plots.
Visual Themes	`theme()` and `theme_*()`	The overall visual defaults of a plot.
Annotations	`annotate()`	Add additional labels, geometries or images to a plot.

Data: `ggplot(data)`

We need to specify the data in the `ggplot()` call:

```
ggplot(data = data)
```

Data: `ggplot(data)`

We need to specify the data in the `ggplot()` call:

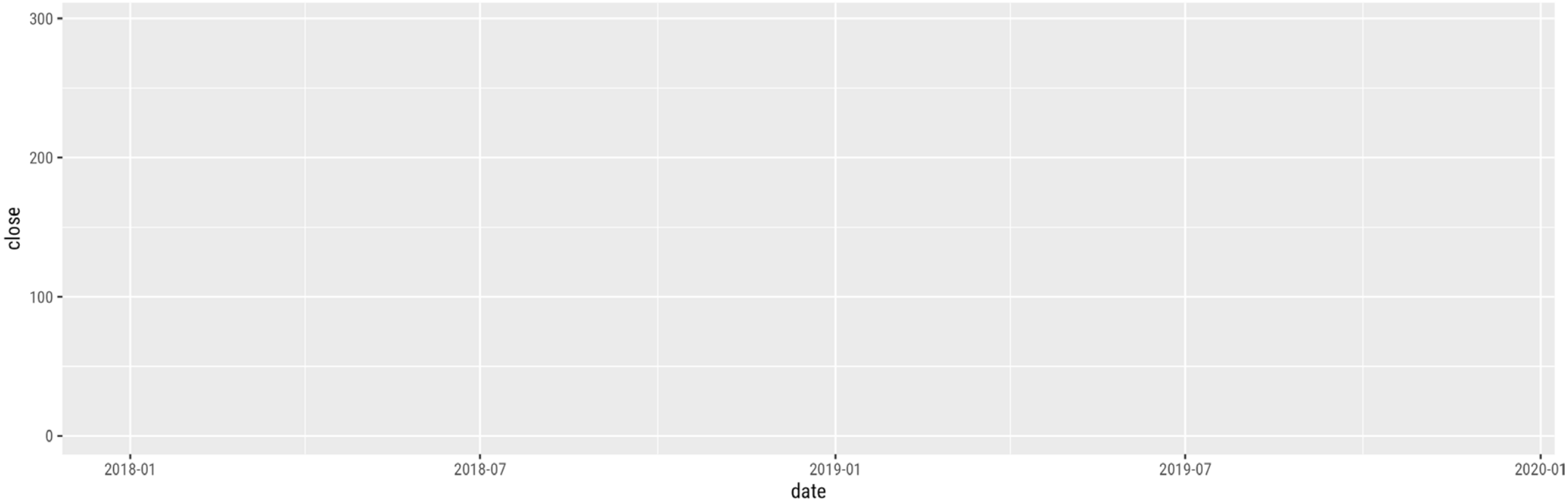
```
ggplot(data = data)
```


There is only an empty panel because `{ggplot2}` doesn't know **what** of the data it should plot.

Aesthetics: `aes()`

We need to specify two variables we want to plot as positional aesthetics:

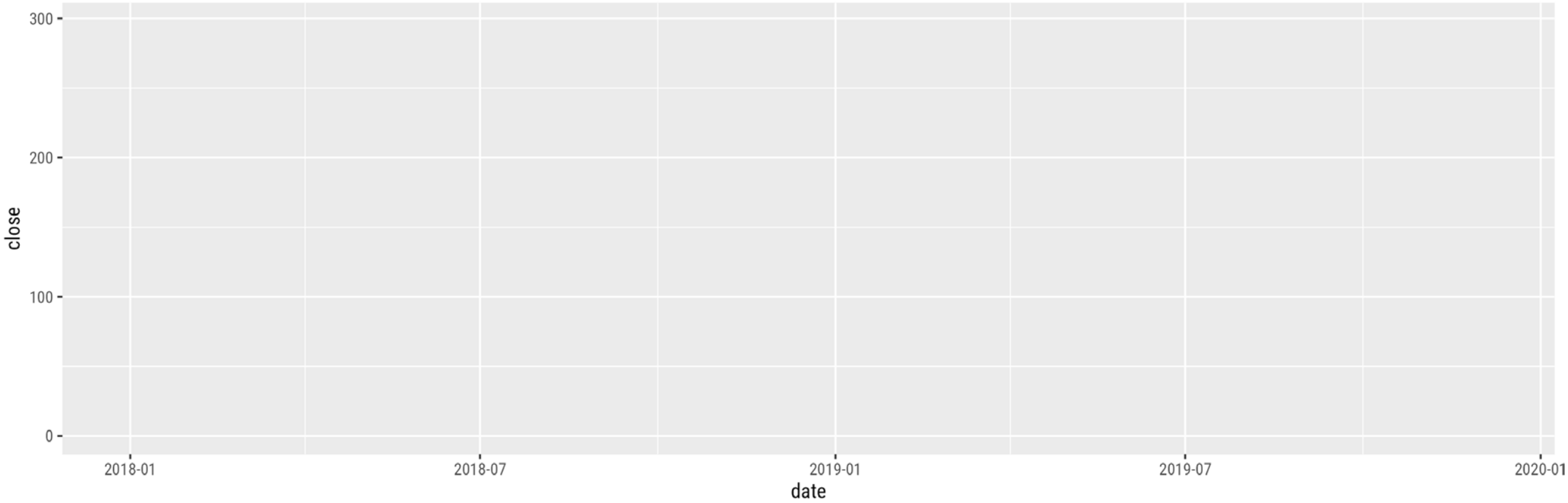
```
ggplot(data = data, mapping = aes(x = date, y = close))
```



Aesthetics: `aes()`

We need to specify two variables we want to plot as positional `aes`thetics:

```
ggplot(data = data, mapping = aes(x = date, y = close))
```

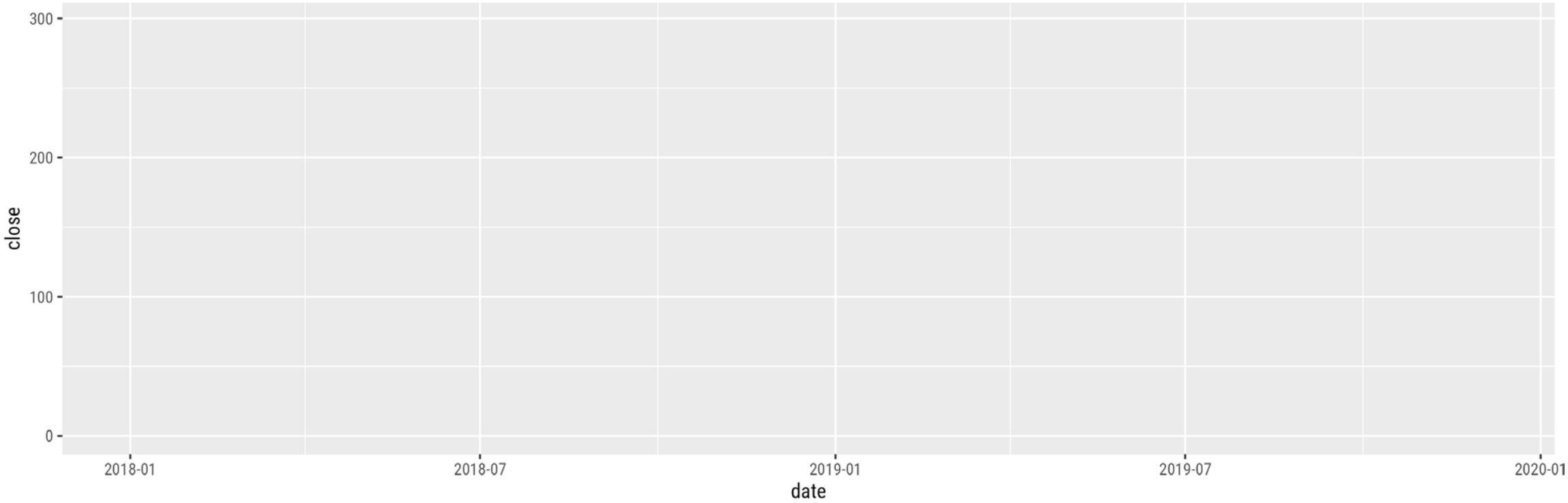


There is only an empty panel because `{ggplot2}` doesn't know **how** it should plot the data.

`ggplot(data, aes(x, y))`

Thanks to implicit matching of arguments in `ggplot()` and `aes()`, we can also write:

```
ggplot(data, aes(date, close))
```



Layers: `geom_*()` and `stat_*()`

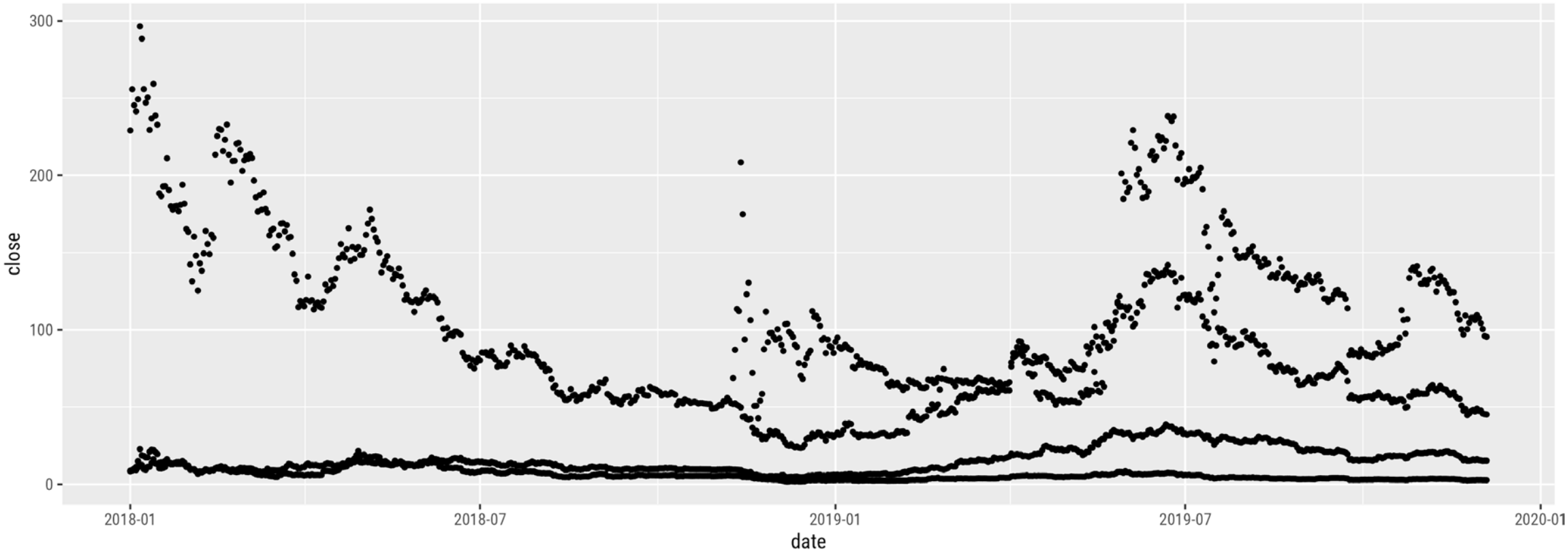
By adding one or multiple layers we can tell `{ggplot2}` *how* to represent the data.
There are lots of built-in geometric elements (`geom's`) and statistical transformations (`stat's`):

Adapted from ggplot2.tidyverse.org/reference

Layers: `geom_*()`

We can tell `{ggplot2}` to represent the data for example as a **scatter plot**:

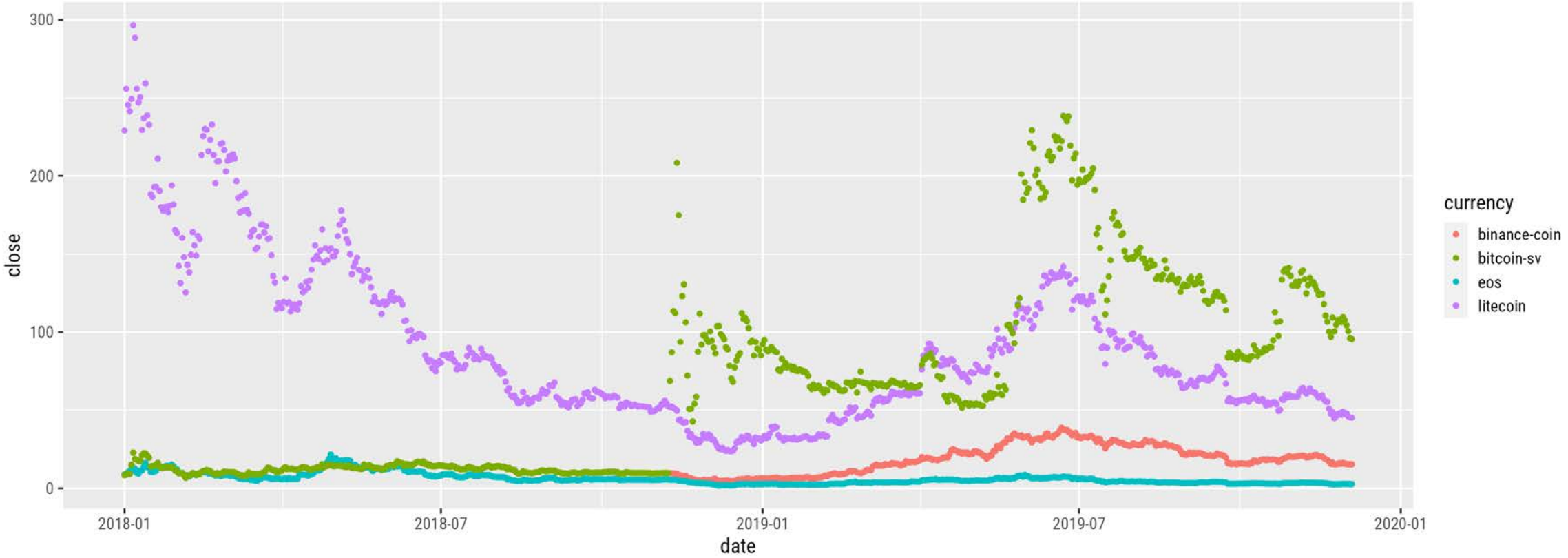
```
ggplot(data, aes(date, close)) +
  geom_point()
```



Aesthetics: `aes()`

Aesthetics do not only refer to x and y positions, but also groupings, colors, fills, shapes etc.

```
ggplot(data = data, mapping = aes(x = date, y = close, color = currency)) +
  geom_point()
```

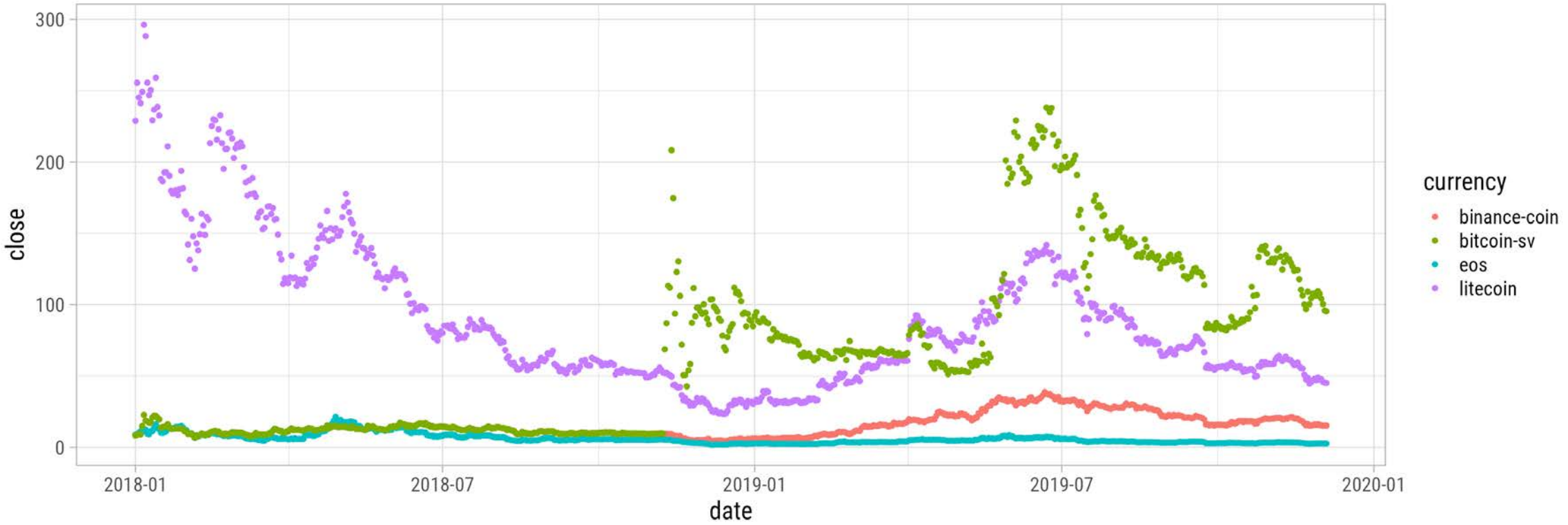


Important Things First: Change the Default Theme

```
theme_set(theme_light(base_size = 18))
```

Important Things First: Change the Default Theme

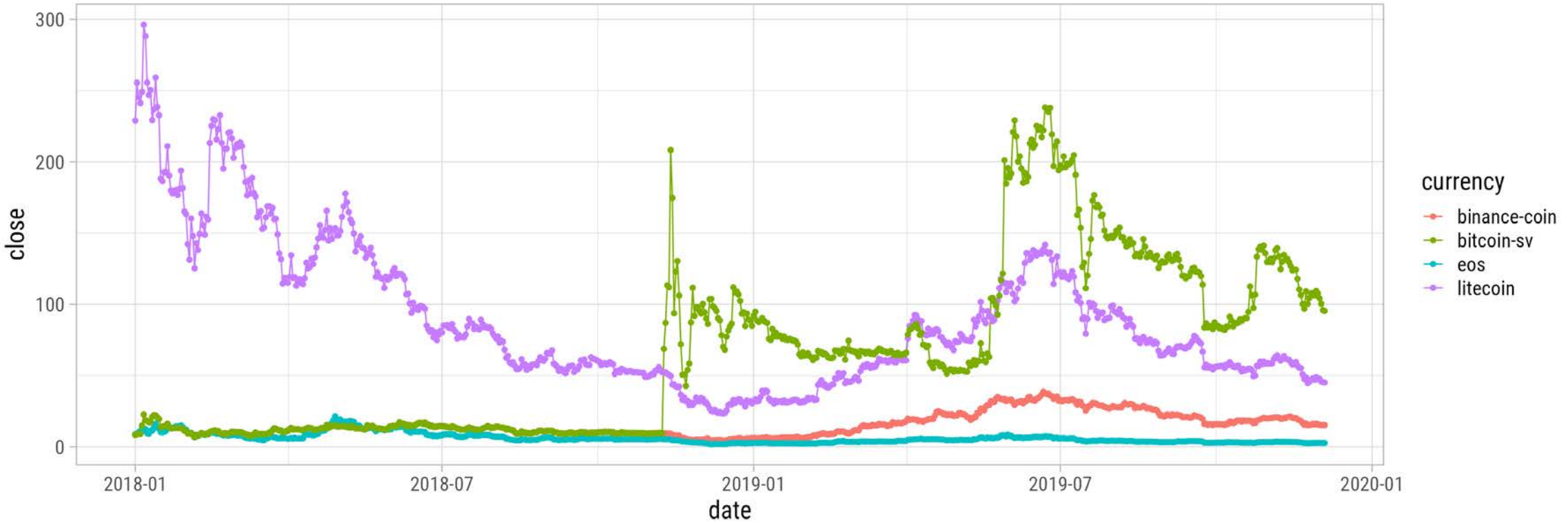
```
theme_set(theme_light(base_size = 18))

ggplot(data = data, mapping = aes(x = date, y = close, color = currency)) +
  geom_point()
```



Layers: `geom_*()` and `stat_*()`

The exciting thing about layers is that you can combine several `geom_*()` and `stat_*()` calls:

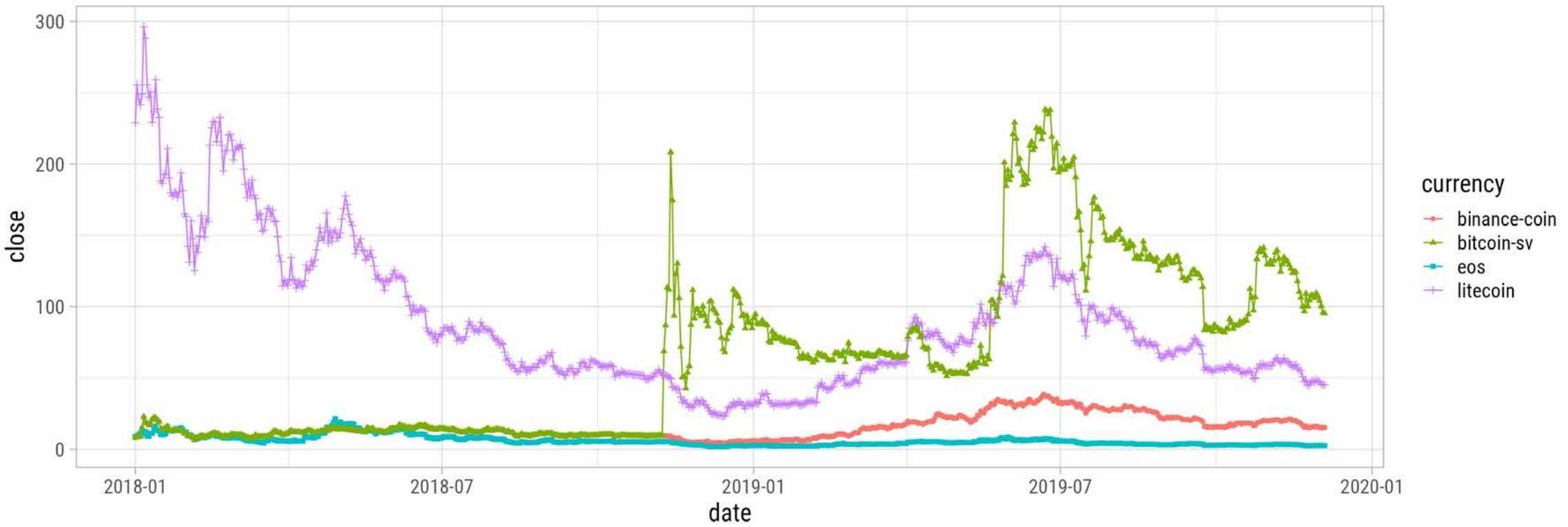
```
ggplot(data, aes(date, close, color = currency)) +
  geom_line() +
  geom_point()
```



Layers: `geom_*()` and `stat_*()`

... and aesthetics can be applied either **globally** or for each layer individually:

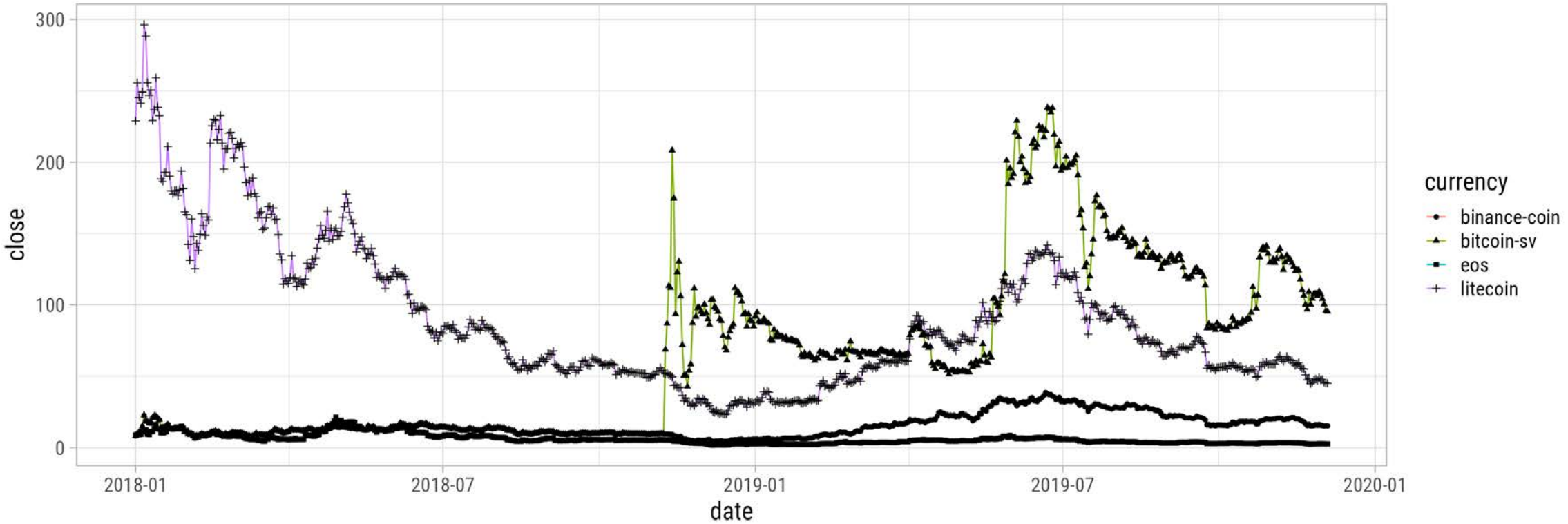
```
ggplot(data, aes(date, close, color = currency, shape = currency)) +
  geom_line() +
  geom_point()
```



Layers: `geom_*()` and `stat_*()`

... and aesthetics can be applied either globally or for each layer **individually**:

```
ggplot(data, aes(date, close)) +
  geom_line(aes(color = currency)) +
  geom_point(aes(shape = currency))
```



Exercise 2:

- If needed, import the Chicago weather data again:

```
chic <- readr::read_csv(
  "https://raw.githubusercontent.com/z3tt/ggplot-courses/master/data/chicago-nmmaps.csv"
)
```

- Create a scatter plot of temperature (`temp`) versus day (`date`).
- Color the points by season (`season`).
- Color the points by year (`year`).
    - What's the problem? How could you fix it?

Exercise 2:

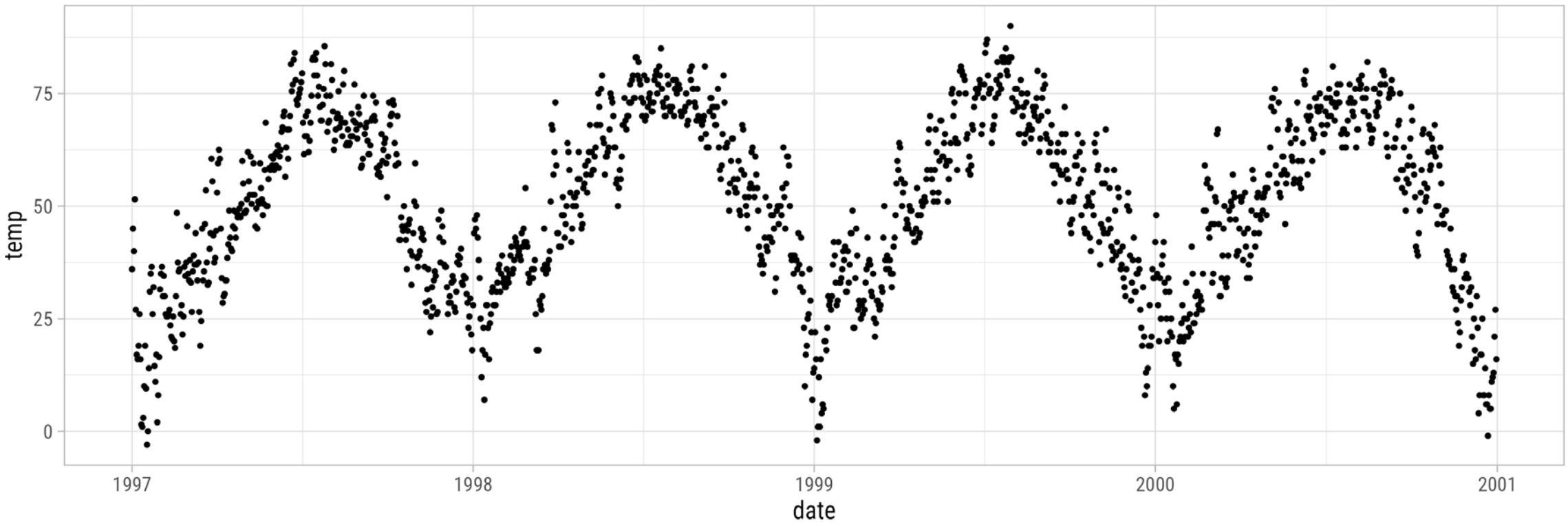
- If needed, import the Chicago weather data again:

```
chic <- readr::read_csv(
  "https://raw.githubusercontent.com/z3tt/ggplot-courses/master/data/chicago-nmmaps.csv"
)
```

- Create a scatter plot of temperature (`temp`) versus day (`date`).
- Color the points by season (`season`).
- Color the points by year (`year`).
    - What's the problem? How could you fix it?
- **Bonus:** Turn the scatter plot, colored by season, into a line plot.
    - What's the problem? How could you fix it?

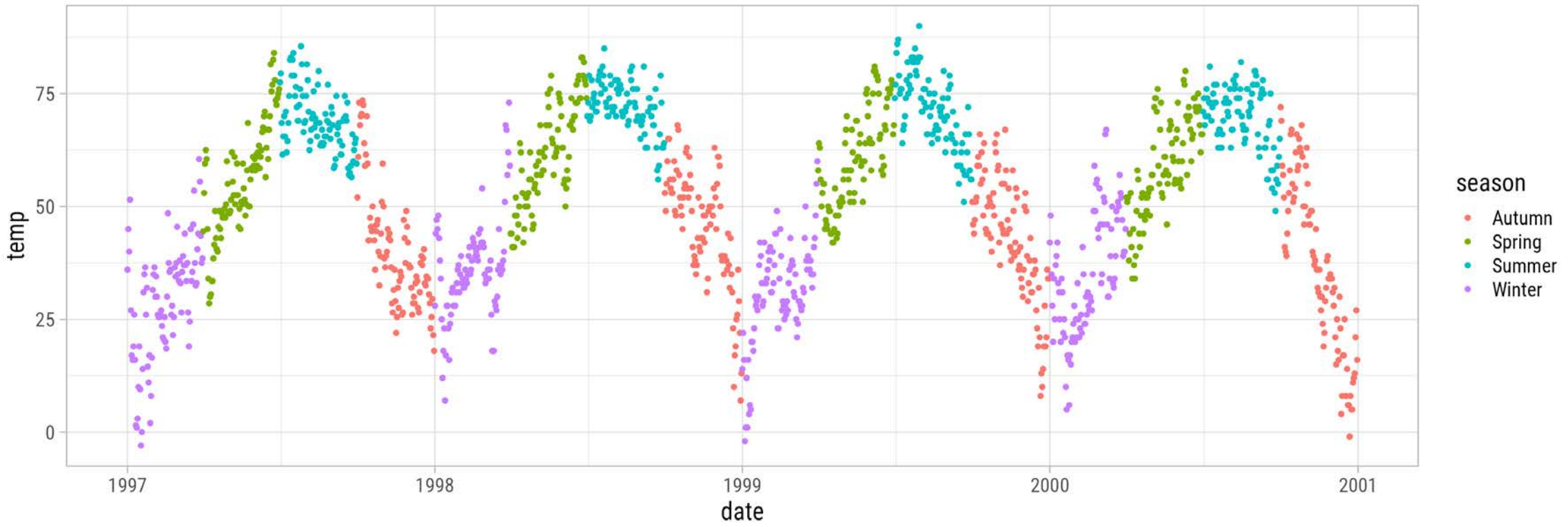
Exercise 2: Scatter Plot Temperature vs Day

```
ggplot(chic, aes(date, temp)) +
  geom_point()
```



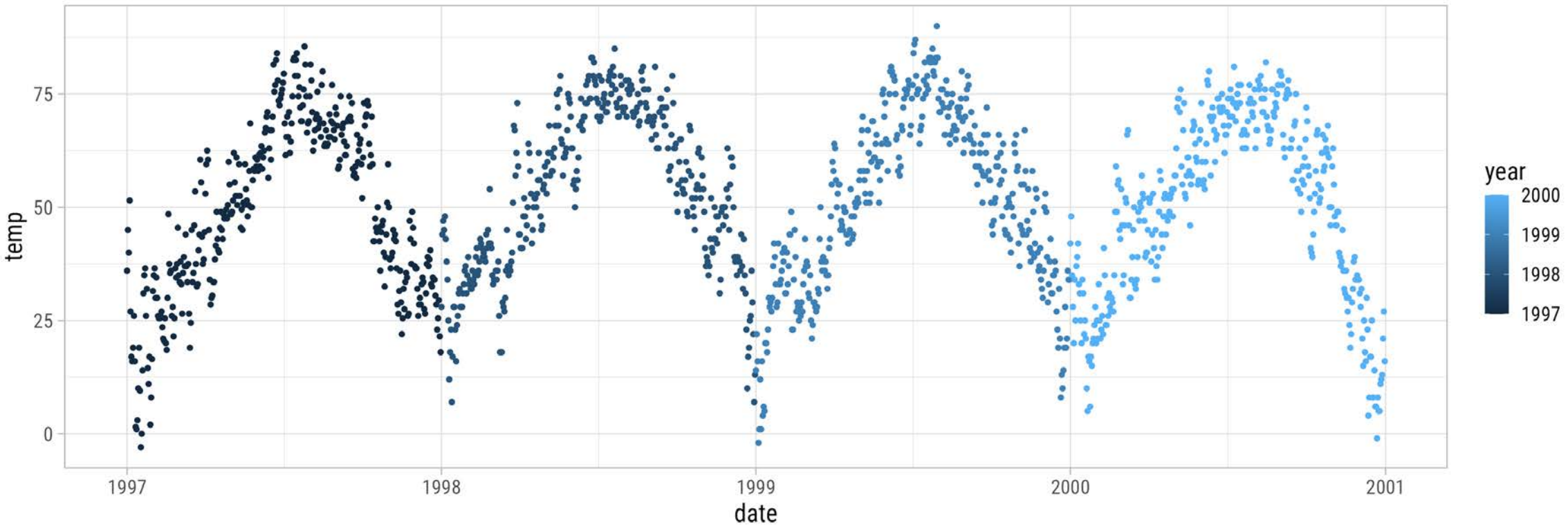
Exercise 2: Scatter Plot Colored by Season

```
ggplot(chic, aes(date, temp, color = season)) +
  geom_point()
```



Exercise 2: Scatter Plot Colored by Year

```
ggplot(chic, aes(date, temp, color = year)) +
  geom_point()
```

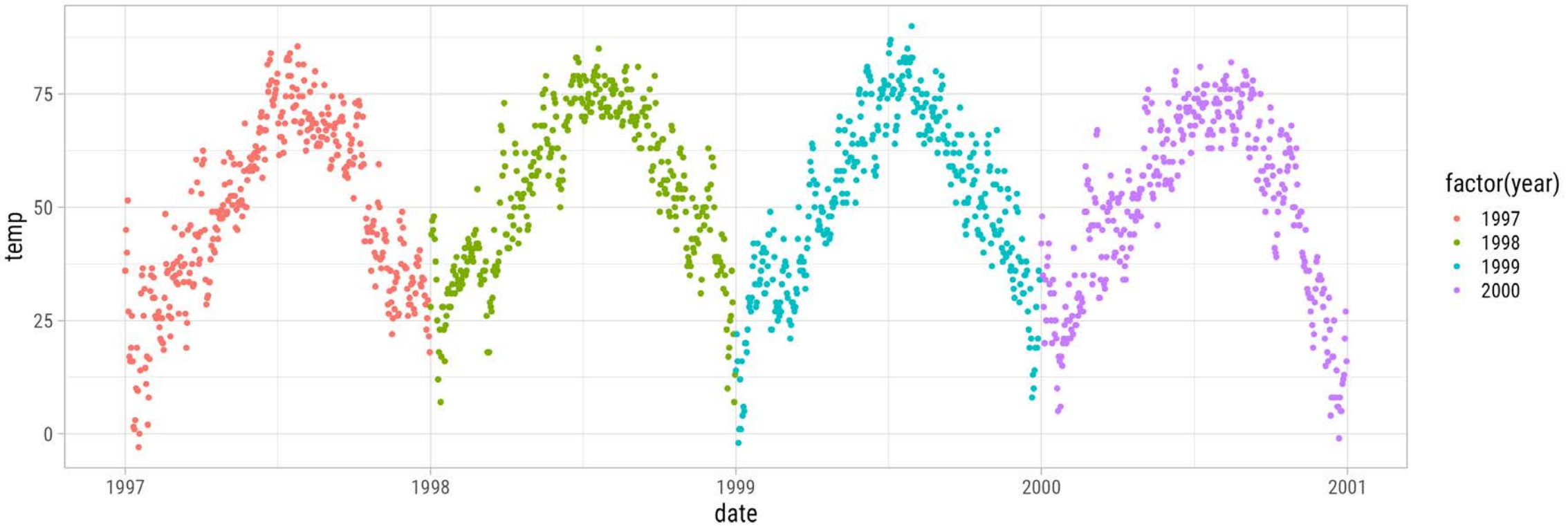


POLL: The variable `year` should be of type... (MC)

- Integer
- Character
- Factor
- Date

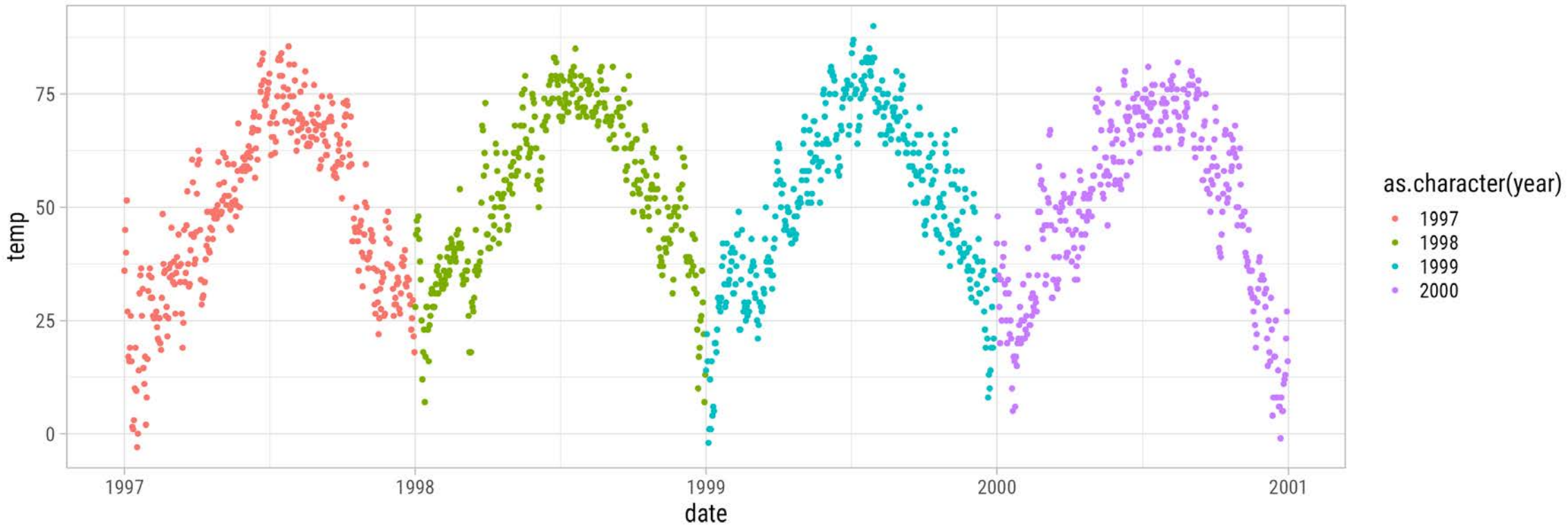
Exercise 2: Scatter Plot Colored by Year

```
ggplot(chic, aes(date, temp, color = factor(year))) +
  geom_point()
```



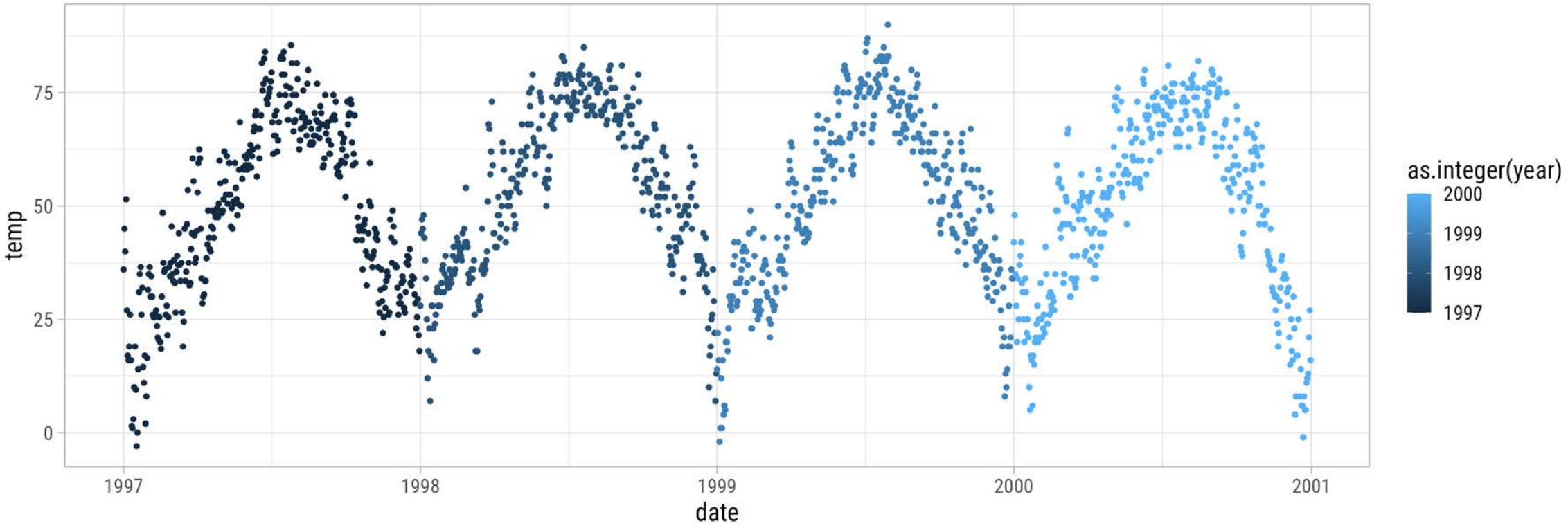
Exercise 2: Scatter Plot Colored by Year

```
ggplot(chic, aes(date, temp, color = as.character(year))) +
  geom_point()
```



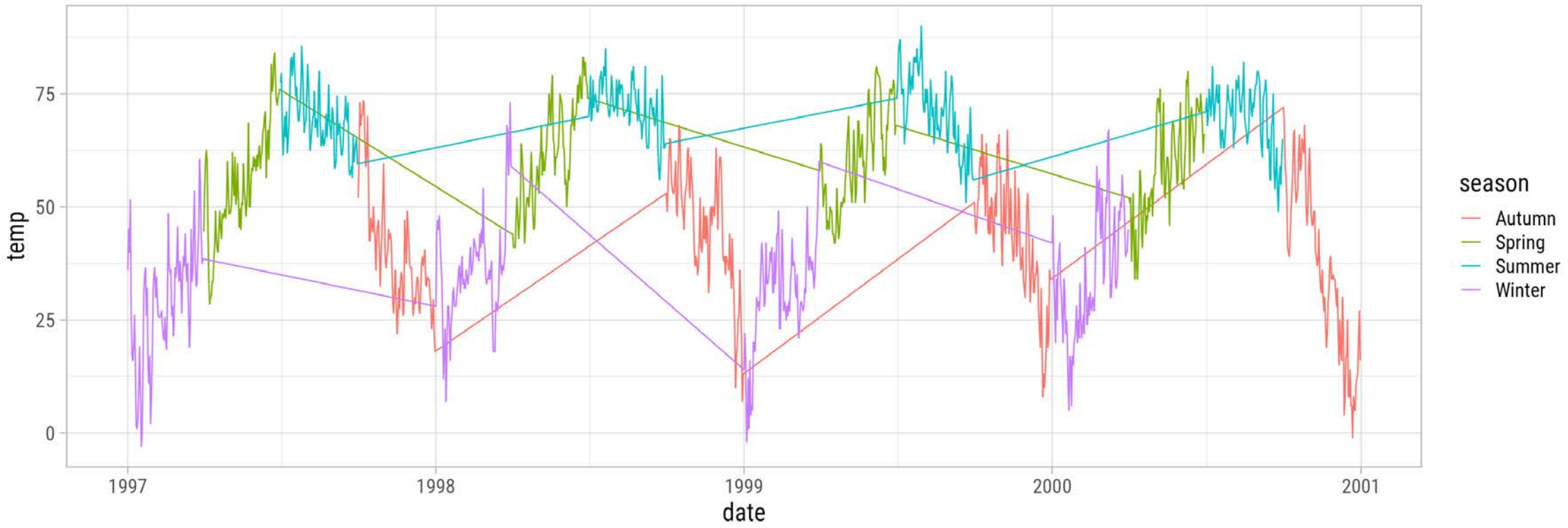
Exercise 2: Scatter Plot Colored by Year

```
ggplot(chic, aes(date, temp, color = as.integer(year))) +
  geom_point()
```



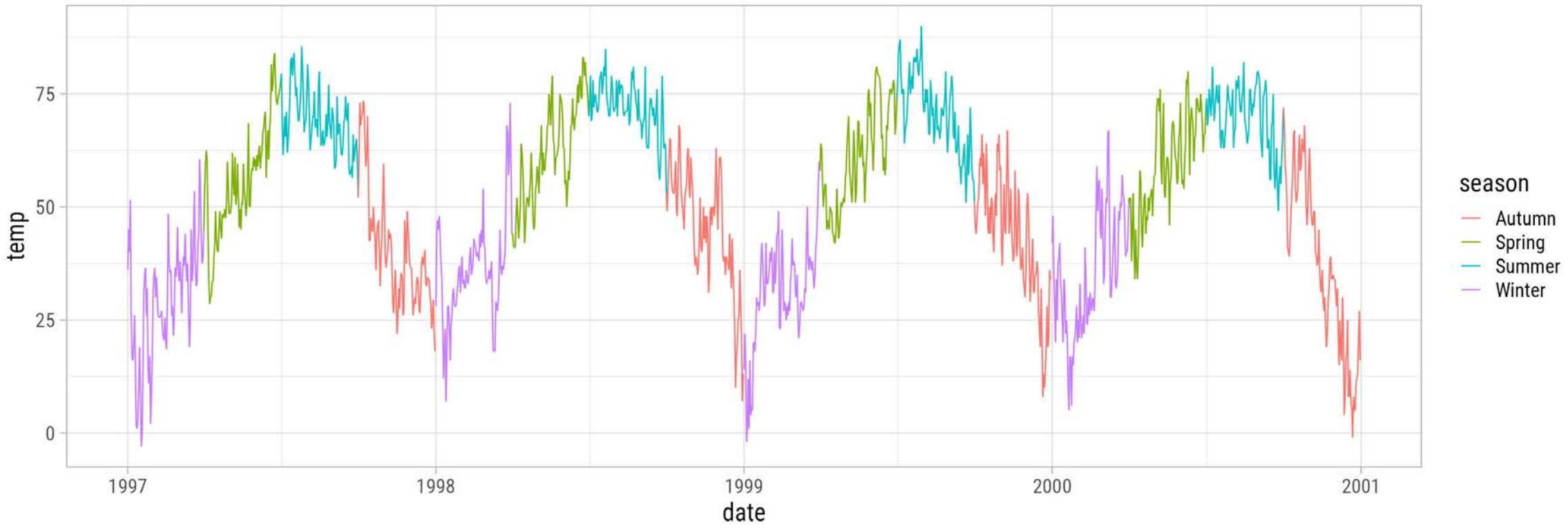
Exercise 2 — Bonus: Scatter Plot Colored by Year

```
ggplot(chic, aes(date, temp, color = season)) +
  geom_line()
```



Exercise 2 — Bonus: Scatter Plot Colored by Year

```
ggplot(chic, aes(date, temp, color = season, group = year)) +
  geom_line()
```



Scales, Coordinate Systems, Facets, Themes, and Annotations will follow later!

Saving a `ggplot`

You can export your plot via the `ggsave()` function:

```
ggsave(filename = "my_ggplot.pdf",
       width = 10, height = 7,
       device = cairo_pdf)

ggsave(filename = "my_ggplot.png",
       width = 10, height = 7,
       dpi = 700)
```

Source: canva.com

Resources

- “ggplot2: Elegant Graphics for Data Analysis”, free–access book by Hadley Wickham et al.
- {ggplot2} reference
- {ggplot2} cheatsheet
- “R for Data Science”, free–access book by Hadley Wickham
- “Data Visualization: A Practical Introduction”, free–access book by Kieran Healy
- “A {ggplot2} Tutorial for Beautiful Plotting in R”, my extensive "how to"–tutorial

APPENDIX

Import the Data

A good workflow when working with local files is offered by the {here} package in combination with R projects:

```
here::here()
## [1] "C:/Users/DataVizard/Google Drive/Work/DataViz/Teaching/2021_OReilly_Trainings/hands-on-ggplot2"

data_local <- readr::read_csv(here::here("data", "crypto_cleaned.csv"))

data_local
## # A tibble: 2,812 x 9
##    currency     date        open  high   low close  year month  yday
##    <chr>        <date>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 binance-coin 2019-12-04  15.4  15.7  15.0  15.3  2019    12   338
##  2 binance-coin 2019-12-03  15.2  15.6  15.0  15.3  2019    12   337
##  3 binance-coin 2019-12-02  15.5  15.7  15.2  15.2  2019    12   336
##  4 binance-coin 2019-12-01  15.7  15.7  15.0  15.5  2019    12   335
##  5 binance-coin 2019-11-30  16.3  16.4  15.5  15.7  2019    11   334
##  6 binance-coin 2019-11-29  15.7  16.3  15.6  16.3  2019    11   333
##  7 binance-coin 2019-11-28  16.1  16.2  15.6  15.7  2019    11   332
##  8 binance-coin 2019-11-27  15.5  16.2  14.9  16.1  2019    11   331
##  9 binance-coin 2019-11-26  15.3  15.9  15.2  15.5  2019    11   330
## 10 binance-coin 2019-11-25  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 2,802 more rows
```

Import the Data

The base R function `read.csv()` works in the same way as `readr::read_csv()`:

```
data <- read.csv(here::here("data", "crypto_cleaned.csv"))

head(data)
##       currency       date  open  high   low close year month yday
## 1 binance-coin 2019-12-04 15.35 15.69 15.01 15.28 2019    12  338
## 2 binance-coin 2019-12-03 15.19 15.55 15.05 15.31 2019    12  337
## 3 binance-coin 2019-12-02 15.51 15.71 15.15 15.19 2019    12  336
## 4 binance-coin 2019-12-01 15.74 15.74 15.05 15.50 2019    12  335
## 5 binance-coin 2019-11-30 16.26 16.37 15.54 15.72 2019    11  334
## 6 binance-coin 2019-11-29 15.68 16.34 15.65 16.27 2019    11  333
```

Import the Data

... and we can turn it into a **tibble** afterwards:

```
data <- tibble::as_tibble(data)

data
## # A tibble: 2,812 x 9
##    currency     date        open  high   low close  year month  yday
##    <chr>        <chr>      <dbl> <dbl> <dbl> <dbl> <int> <int> <int>
##  1 binance-coin 2019-12-04  15.4  15.7  15.0  15.3  2019    12   338
##  2 binance-coin 2019-12-03  15.2  15.6  15.0  15.3  2019    12   337
##  3 binance-coin 2019-12-02  15.5  15.7  15.2  15.2  2019    12   336
##  4 binance-coin 2019-12-01  15.7  15.7  15.0  15.5  2019    12   335
##  5 binance-coin 2019-11-30  16.3  16.4  15.5  15.7  2019    11   334
##  6 binance-coin 2019-11-29  15.7  16.3  15.6  16.3  2019    11   333
##  7 binance-coin 2019-11-28  16.1  16.2  15.6  15.7  2019    11   332
##  8 binance-coin 2019-11-27  15.5  16.2  14.9  16.1  2019    11   331
##  9 binance-coin 2019-11-26  15.3  15.9  15.2  15.5  2019    11   330
## 10 binance-coin 2019-11-25  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 2,802 more rows
```

However, note that by default the date column is turned into type `character`.

Import the Data

The `import()` function from the `{rio}` package allows to load all kind of data formats:

```
data <- rio::import(here::here("data", "crypto_cleaned.csv"), setclass = "tbl")

data
## # A tibble: 2,812 x 9
##    currency     date        open  high   low close  year month  yday
##    <chr>        <date>     <dbl> <dbl> <dbl> <dbl> <int> <int> <int>
##  1 binance-coin 2019-12-04  15.4  15.7  15.0  15.3  2019    12   338
##  2 binance-coin 2019-12-03  15.2  15.6  15.0  15.3  2019    12   337
##  3 binance-coin 2019-12-02  15.5  15.7  15.2  15.2  2019    12   336
##  4 binance-coin 2019-12-01  15.7  15.7  15.0  15.5  2019    12   335
##  5 binance-coin 2019-11-30  16.3  16.4  15.5  15.7  2019    11   334
##  6 binance-coin 2019-11-29  15.7  16.3  15.6  16.3  2019    11   333
##  7 binance-coin 2019-11-28  16.1  16.2  15.6  15.7  2019    11   332
##  8 binance-coin 2019-11-27  15.5  16.2  14.9  16.1  2019    11   331
##  9 binance-coin 2019-11-26  15.3  15.9  15.2  15.5  2019    11   330
## 10 binance-coin 2019-11-25  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 2,802 more rows
```

Import the Data

The `import()` function from the `{rio}` package allows to load all kind of data formats:

```
data_json <- rio::import(here::here("data", "crypto_cleaned.json"))

data_json <- as_tibble(data_json) ## somehow `setclass` doesn't work with json

data_json
## # A tibble: 3,424 x 9
##    currency     date        open  high   low close  year month  yday
##    <chr>        <chr>      <dbl> <dbl> <dbl> <dbl> <int> <int> <int>
##  1 binance-coin 2019-12-04  15.4  15.7  15.0  15.3  2019    12   338
##  2 binance-coin 2019-12-03  15.2  15.6  15.0  15.3  2019    12   337
##  3 binance-coin 2019-12-02  15.5  15.7  15.2  15.2  2019    12   336
##  4 binance-coin 2019-12-01  15.7  15.7  15.0  15.5  2019    12   335
##  5 binance-coin 2019-11-30  16.3  16.4  15.5  15.7  2019    11   334
##  6 binance-coin 2019-11-29  15.7  16.3  15.6  16.3  2019    11   333
##  7 binance-coin 2019-11-28  16.1  16.2  15.6  15.7  2019    11   332
##  8 binance-coin 2019-11-27  15.5  16.2  14.9  16.1  2019    11   331
##  9 binance-coin 2019-11-26  15.3  15.9  15.2  15.5  2019    11   330
## 10 binance-coin 2019-11-25  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 3,414 more rows
```

Import the Data

The `import()` function from the `{rio}` package allows to load all kind of data formats:

```
data_xlsx <- rio::import(here::here("data", "crypto_cleaned.xlsx"), setclass = "tbl")

data_xlsx
## # A tibble: 2,812 x 10
##    ...1  currency  date                open  high   low close  year month  yday
##    <chr> <chr>     <dttm>             <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 1     binance-~ 2019-12-04 00:00:00  15.4  15.7  15.0  15.3  2019    12   338
##  2 2     binance-~ 2019-12-03 00:00:00  15.2  15.6  15.0  15.3  2019    12   337
##  3 3     binance-~ 2019-12-02 00:00:00  15.5  15.7  15.2  15.2  2019    12   336
##  4 4     binance-~ 2019-12-01 00:00:00  15.7  15.7  15.0  15.5  2019    12   335
##  5 5     binance-~ 2019-11-30 00:00:00  16.3  16.4  15.5  15.7  2019    11   334
##  6 6     binance-~ 2019-11-29 00:00:00  15.7  16.3  15.6  16.3  2019    11   333
##  7 7     binance-~ 2019-11-28 00:00:00  16.1  16.2  15.6  15.7  2019    11   332
##  8 8     binance-~ 2019-11-27 00:00:00  15.5  16.2  14.9  16.1  2019    11   331
##  9 9     binance-~ 2019-11-26 00:00:00  15.3  15.9  15.2  15.5  2019    11   330
## 10 10    binance-~ 2019-11-25 00:00:00  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 2,802 more rows
```

Import the Data

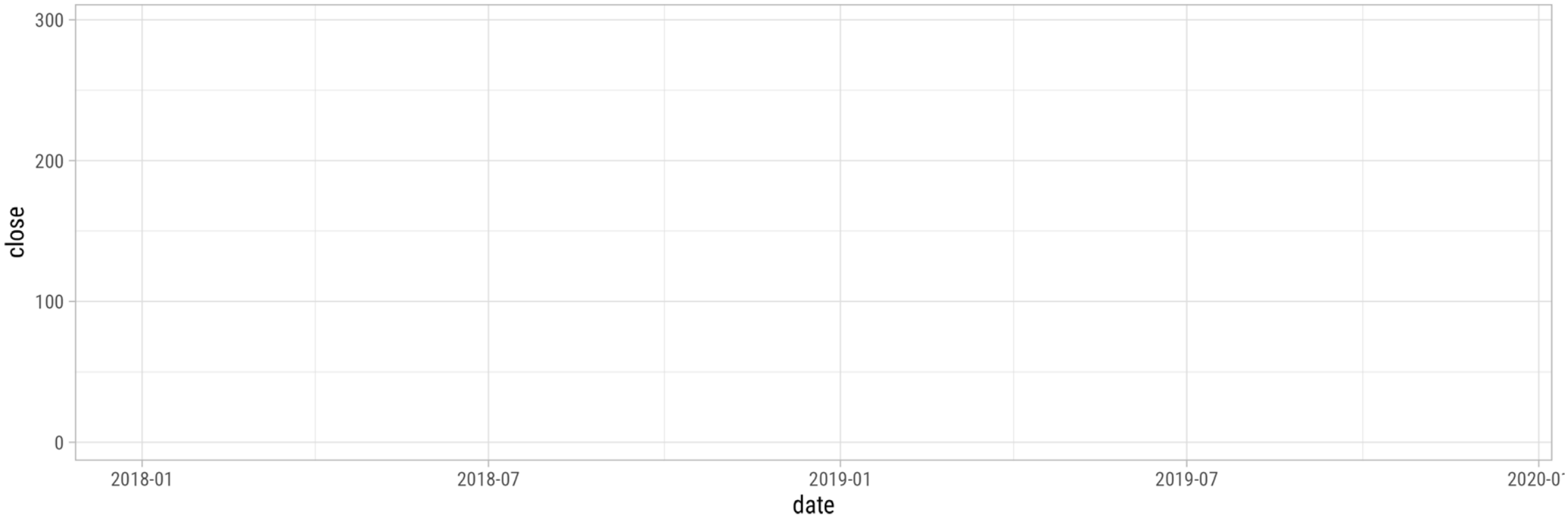
We can remove the first column by using the `select()` function from the `{dplyr}` package:

```
data_xlsx <- dplyr::select(data_xlsx, -1)
#data_xlsx <- dplyr::select(data_xlsx, currency:yday)

data_xlsx
## # A tibble: 2,812 x 9
##    currency     date                 open  high   low close  year month  yday
##    <chr>        <dttm>              <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 binance-coin 2019-12-04 00:00:00  15.4  15.7  15.0  15.3  2019    12   338
##  2 binance-coin 2019-12-03 00:00:00  15.2  15.6  15.0  15.3  2019    12   337
##  3 binance-coin 2019-12-02 00:00:00  15.5  15.7  15.2  15.2  2019    12   336
##  4 binance-coin 2019-12-01 00:00:00  15.7  15.7  15.0  15.5  2019    12   335
##  5 binance-coin 2019-11-30 00:00:00  16.3  16.4  15.5  15.7  2019    11   334
##  6 binance-coin 2019-11-29 00:00:00  15.7  16.3  15.6  16.3  2019    11   333
##  7 binance-coin 2019-11-28 00:00:00  16.1  16.2  15.6  15.7  2019    11   332
##  8 binance-coin 2019-11-27 00:00:00  15.5  16.2  14.9  16.1  2019    11   331
##  9 binance-coin 2019-11-26 00:00:00  15.3  15.9  15.2  15.5  2019    11   330
## 10 binance-coin 2019-11-25 00:00:00  15.3  15.7  14.2  15.3  2019    11   329
## # ... with 2,802 more rows
```

ggplot(data) + aes(x, y)

Some prefer to place the `aes()` outside the `ggplot()` call:

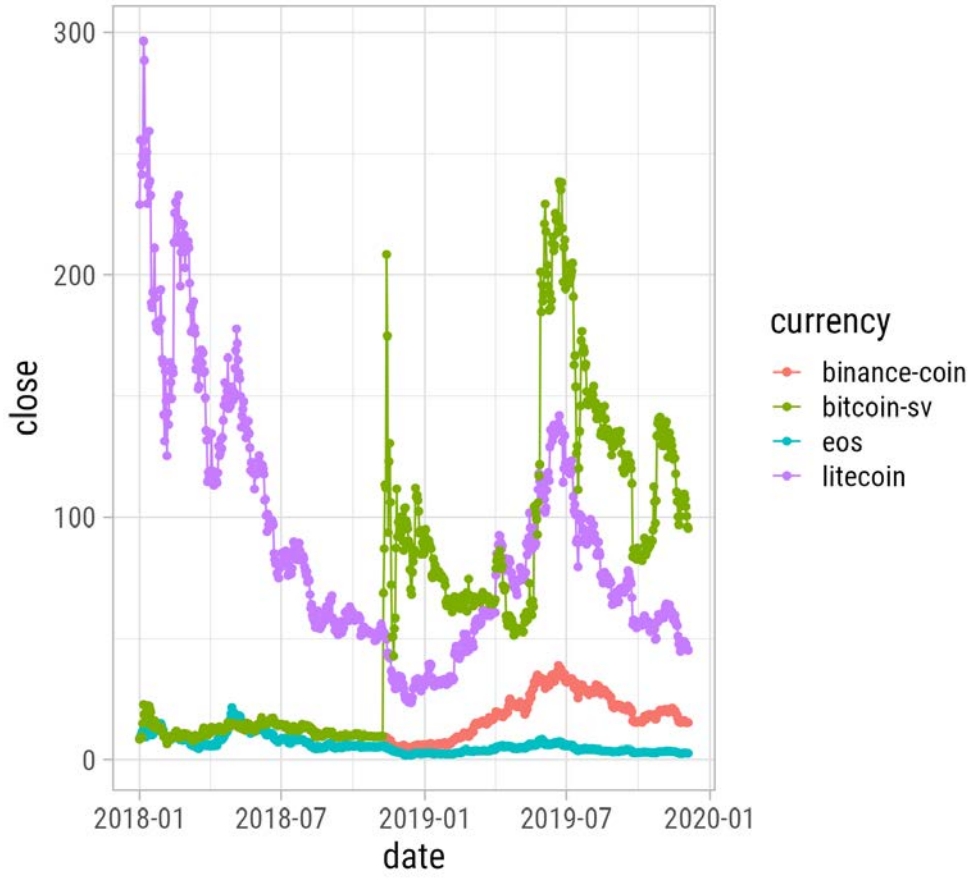
```
ggplot(data) +
  aes(x = date, y = close)
```



Coordinate Systems: `coord_*()`

The coordinate system maps the two position to a 2d position on the plot:

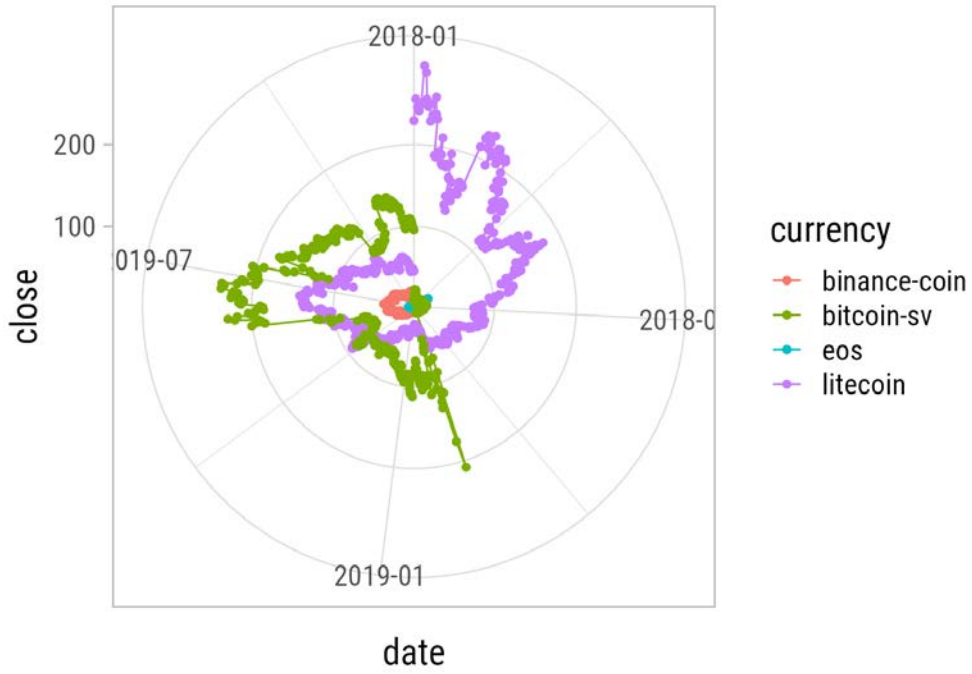
```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  geom_point() +
  scale_x_date() +
  scale_y_continuous() +
  scale_color_discrete() +
  coord_cartesian()
```



Coordinate Systems: `coord_*()`

The coordinate system maps the two position to a 2d position on the plot:

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  geom_point() +
  scale_x_date() +
  scale_y_continuous() +
  scale_color_discrete() +
  coord_polar()
```



Coordinate Systems: `coord_*()`

Changing the limits on the coordinate system allows to zoom in:

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  geom_point() +
  scale_x_date() +
  scale_y_continuous() +
  scale_color_discrete() +
  coord_cartesian(
    xlim = c(as.Date("2018-11-01"),
             as.Date("2019-11-01")),
    ylim = c(NA, 100)
  )
```