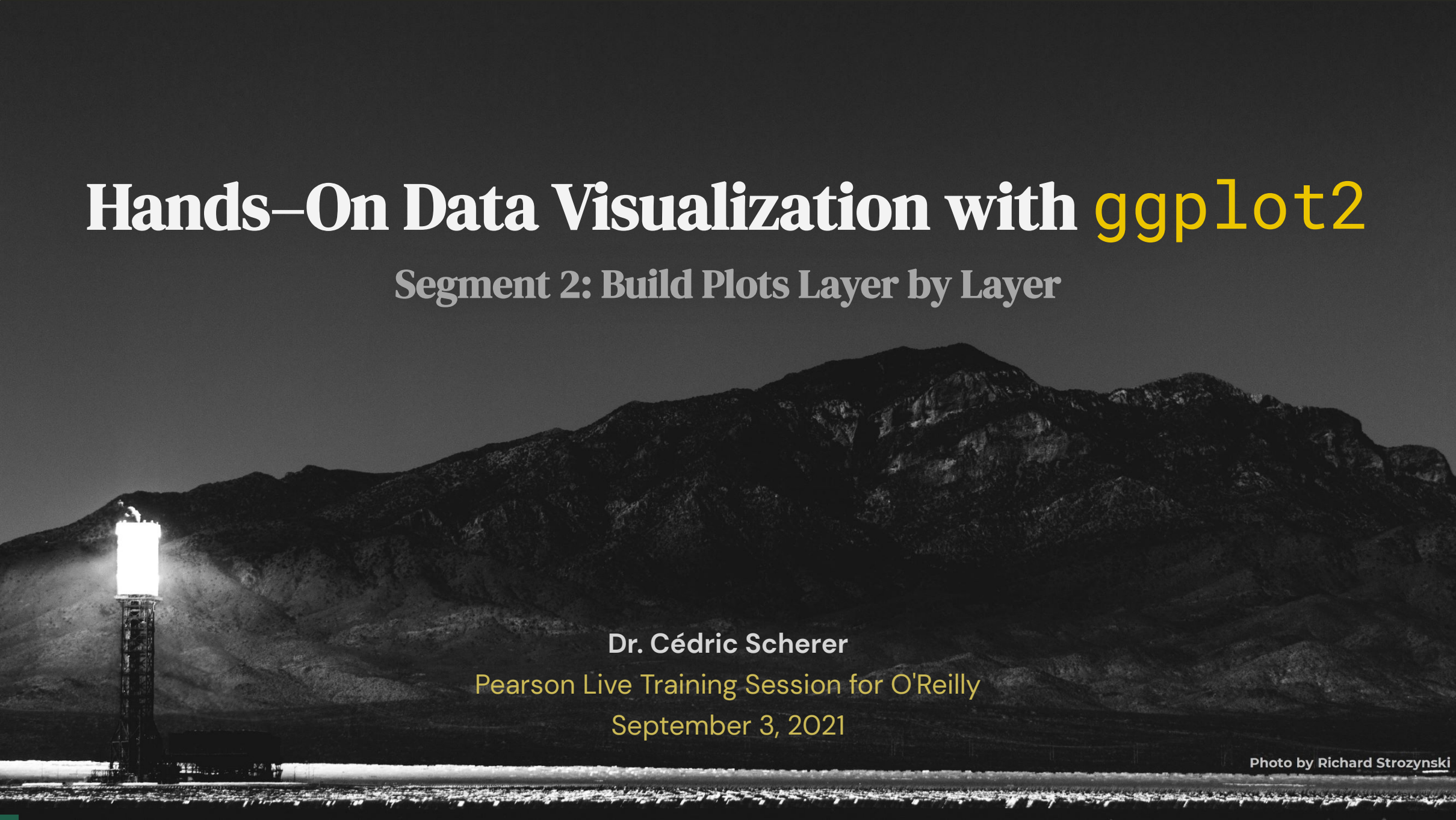
Hands–On Data Visualization with ggplot2

Segment 2: Build Plots Layer by Layer



Dr. Cédric Scherer

Pearson Live Training Session for O'Reilly

September 3, 2021

Photo by Richard Strozynski

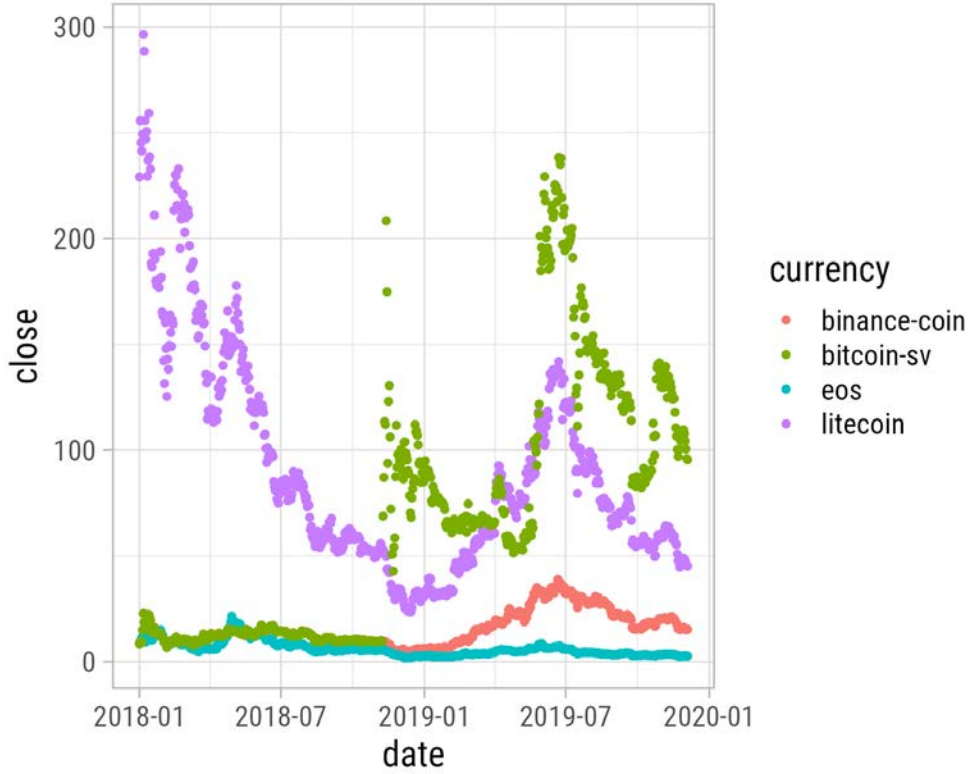
POLL: Which chart types do you use regularly? (MC)

- Pie or donut chart
- Bubble chart
- Dumbbell plot
- Heatmap
- Treemap
- Network graph
- Dendrogram
- Gantt chart
- Candlestick chart
- Radar/spider chart
- Sankey diagram
- Maps (bubble, choropleth, ...)

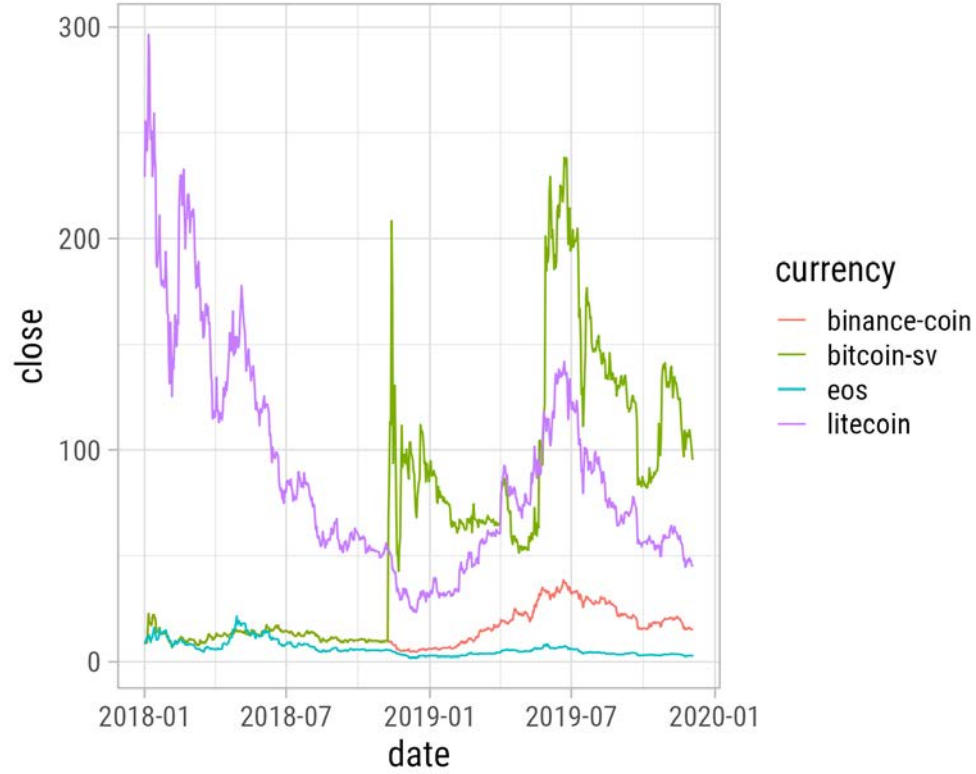
Create Any Chart Type

As you've seen, we can quickly switch through different chart types:

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



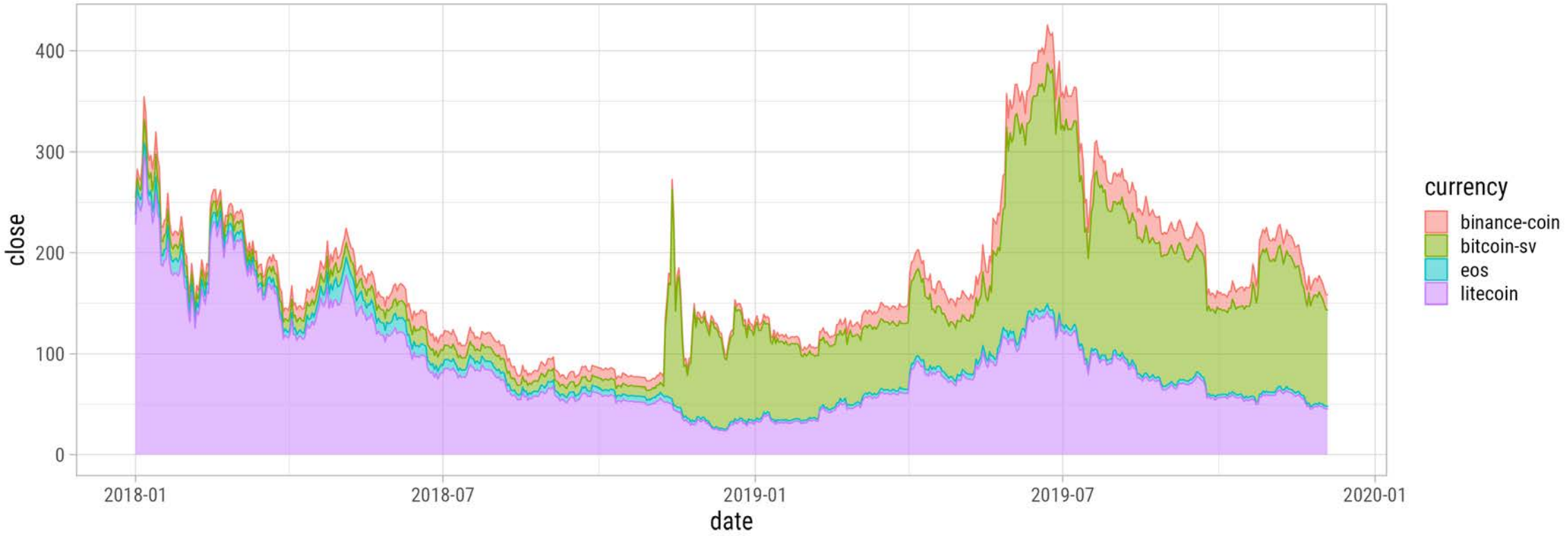

```
ggplot(data, aes(date, close, color = currency)) +
  geom_line()
```



Create Any Chart Type: Area Charts

As you've seen, we can quickly switch through different chart types:

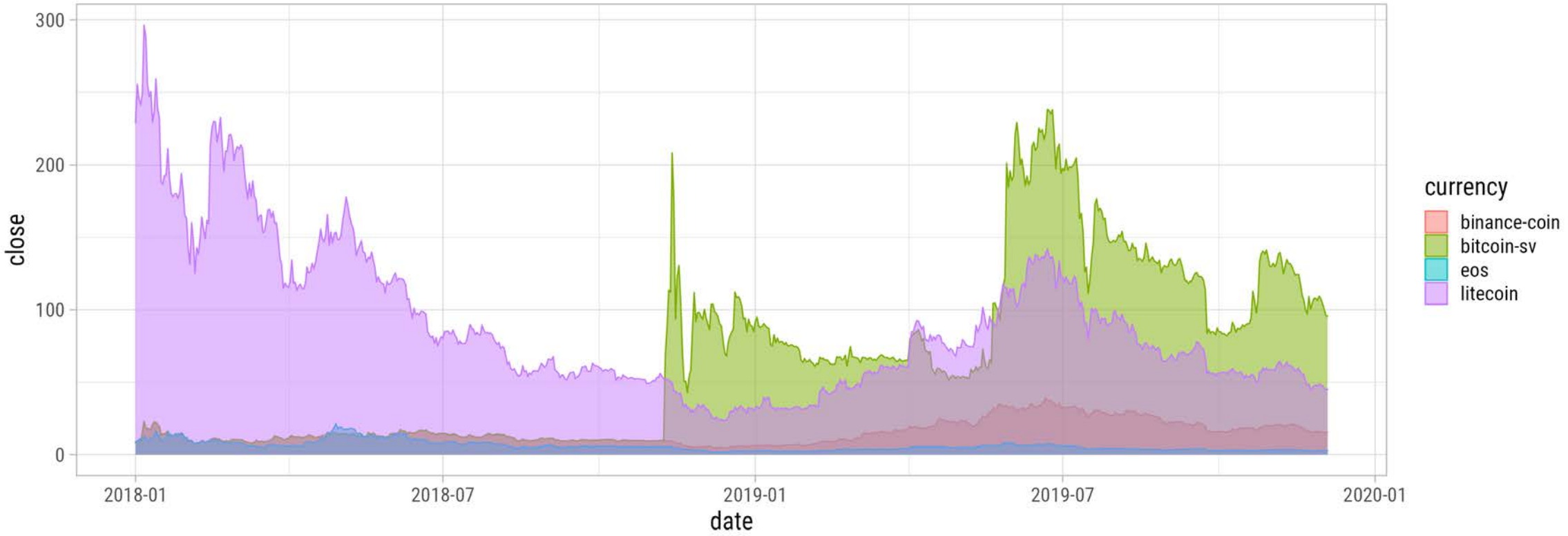
```
ggplot(data, aes(date, close, color = currency)) +
  geom_area(aes(fill = currency), alpha = .5)
```



Create Any Chart Type: Area Charts

Many of the geom's can further adjusted:

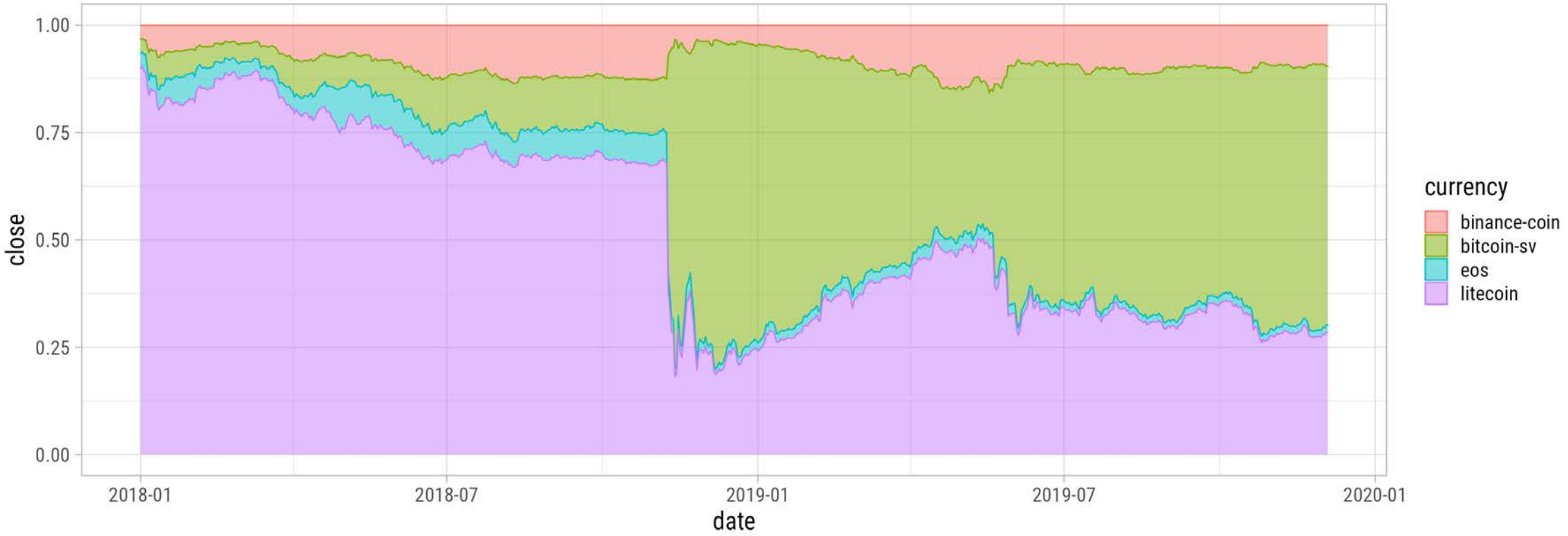
```
ggplot(data, aes(date, close, color = currency)) +
  geom_area(aes(fill = currency), alpha = .5, position = "dodge")
```



Create Any Chart Type: Area Charts

Many of the geom's can further adjusted:

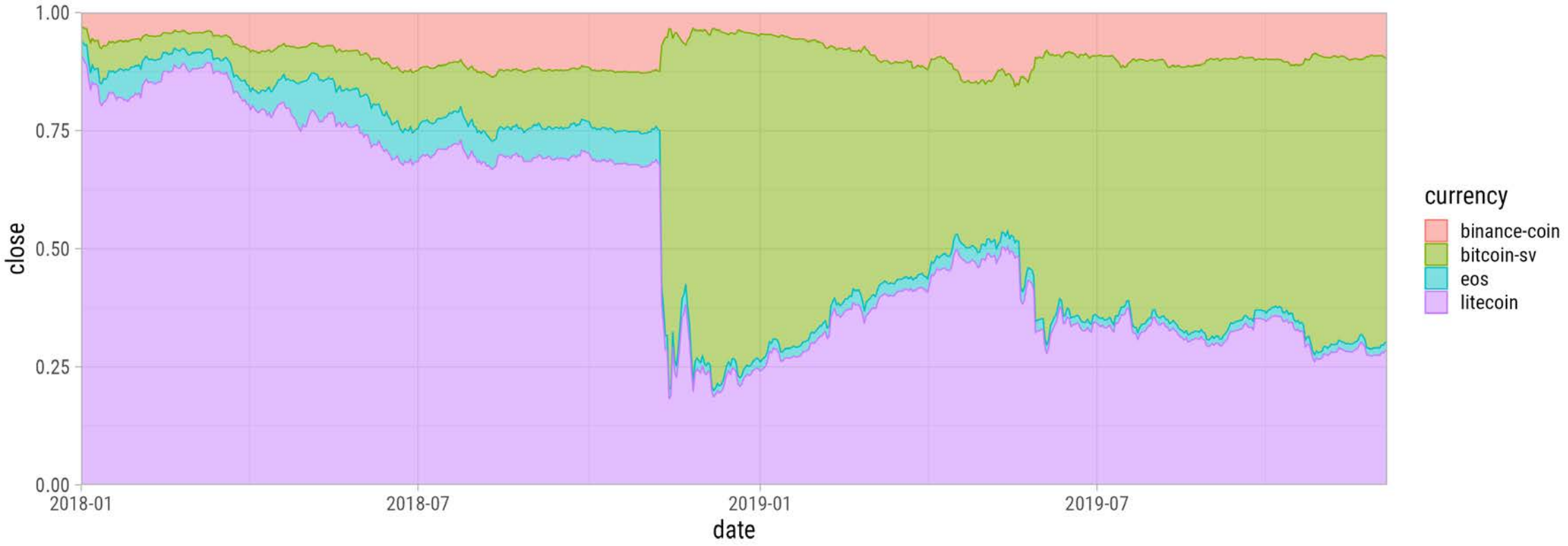
```
ggplot(data, aes(date, close, color = currency)) +
  geom_area(aes(fill = currency), alpha = .5, position = "fill")
```



Create Any Chart Type: Area Charts

Many of the geom's can be further adjusted:

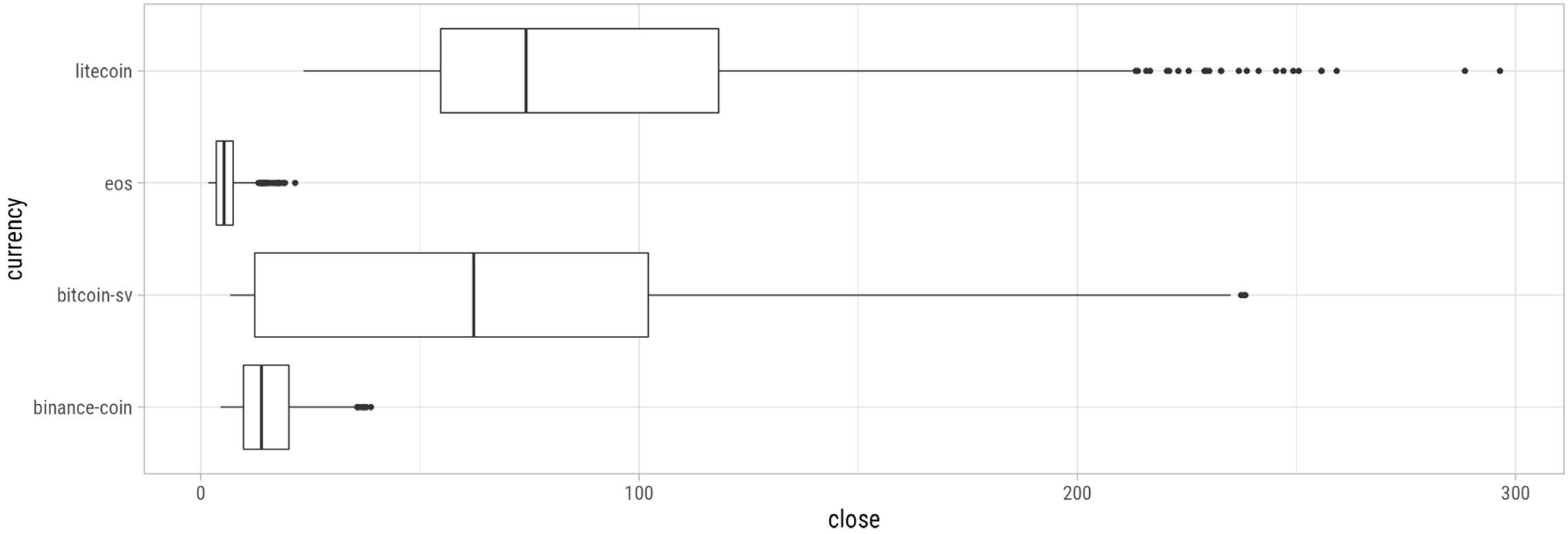
```
ggplot(data, aes(date, close, color = currency)) +
  geom_area(aes(fill = currency), alpha = .5, position = "fill") +
  coord_cartesian(expand = FALSE)
```



Create Any Chart Type: Box Plots

Some geom's drastically change the appearance of the data:

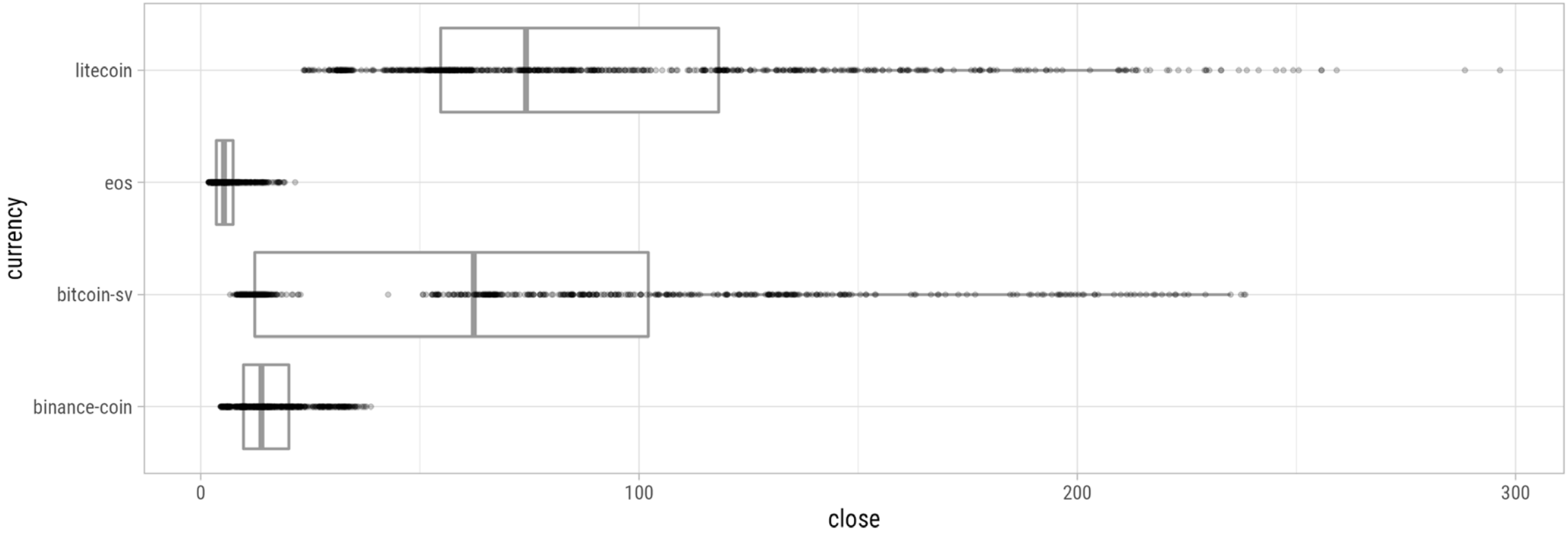
```
ggplot(data, aes(close, currency)) +
  geom_boxplot()
```



Create Any Chart Type: Box Plots

And, as you already know, we can combine geom's to make the plot more insightful:

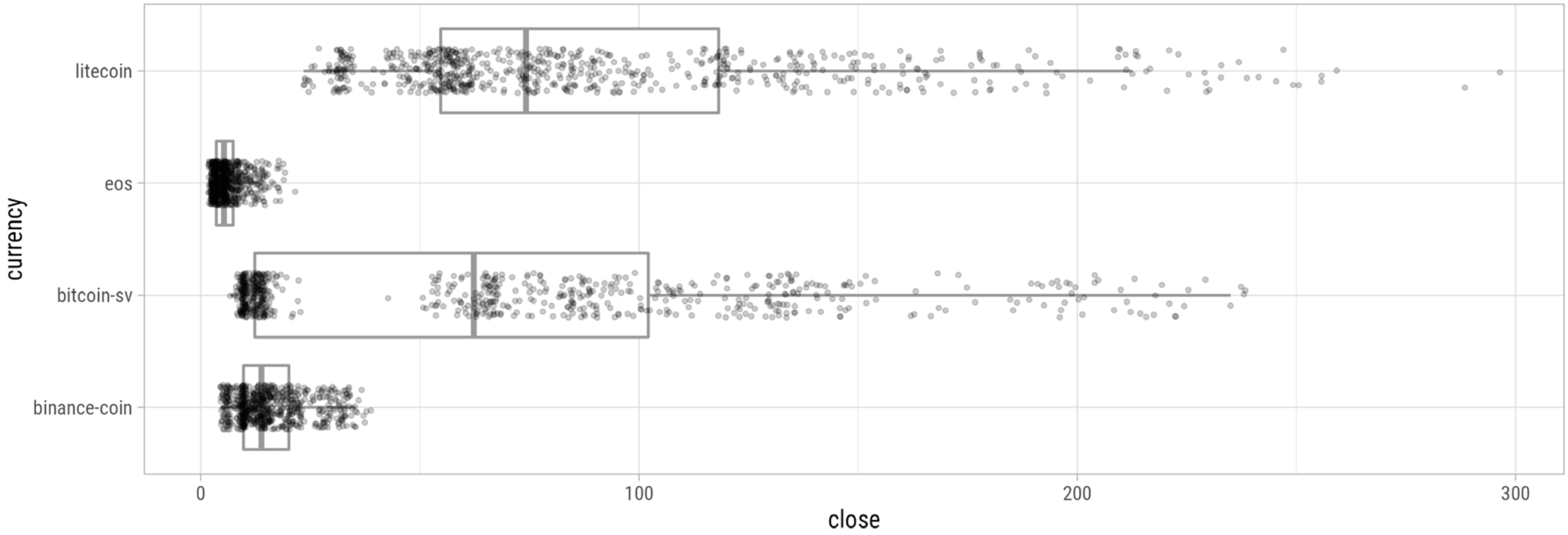
```
ggplot(data, aes(close, currency)) +
  geom_boxplot(outlier.alpha = 0, color = "grey60", size = 1) +
  geom_point(alpha = .2)
```



Create Any Chart Type: Jitterstrips

... and also the point geom can be adjusted:

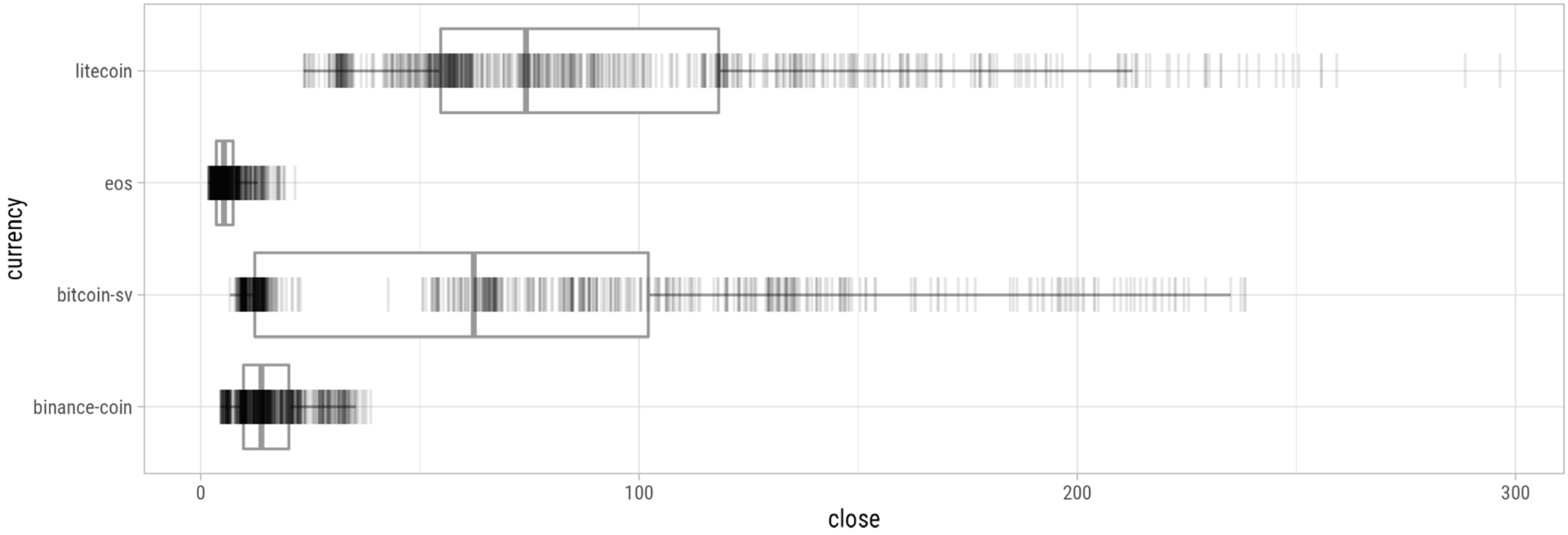
```
ggplot(data, aes(close, currency)) +
  geom_boxplot(outlier.alpha = 0, color = "grey60", size = 1) +
  geom_point(alpha = .2, position = position_jitter(height = .2))
```



Create Any Chart Type: Barcode Plots

... and also the point geom can be adjusted:

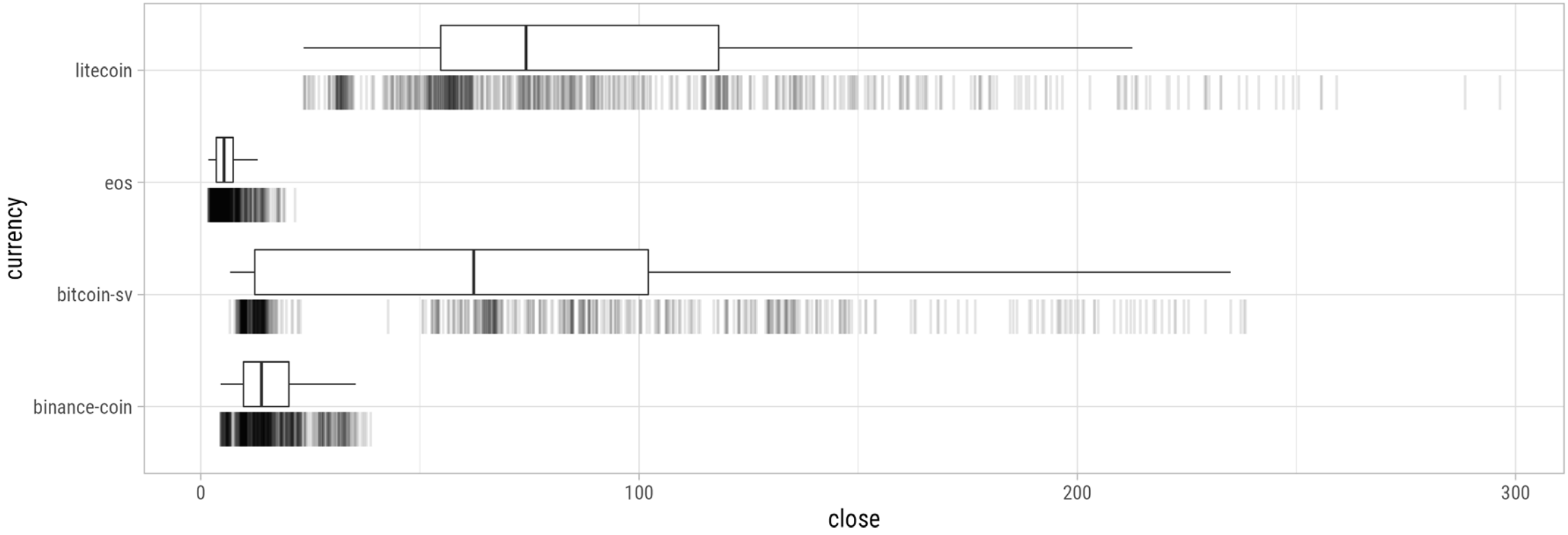
```
ggplot(data, aes(close, currency)) +
  geom_boxplot(outlier.alpha = 0, color = "grey60", size = 1) +
  geom_point(alpha = .1, shape = "|", size = 9)
```



Create Any Chart Type: Barcode Plots

... and also the point geom can be adjusted:

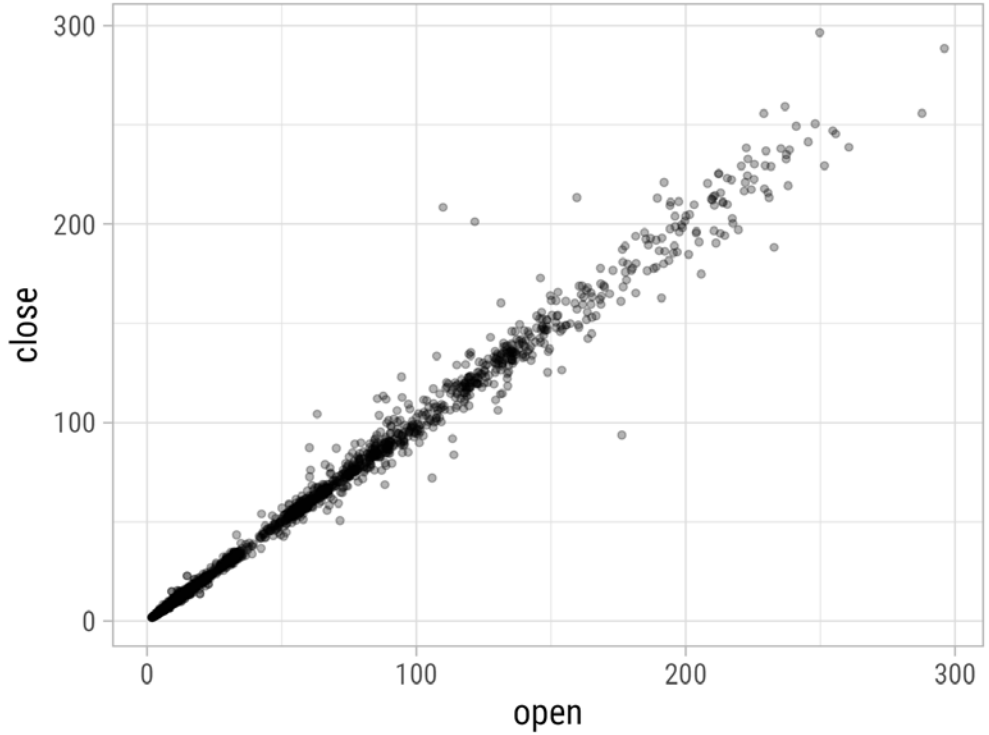
```
ggplot(data, aes(close, currency)) +
  geom_boxplot(outlier.alpha = 0, width = .4, position = position_nudge(y = .2)) +
  geom_point(alpha = .1, shape = "|", size = 9, position = position_nudge(y = -.2))
```



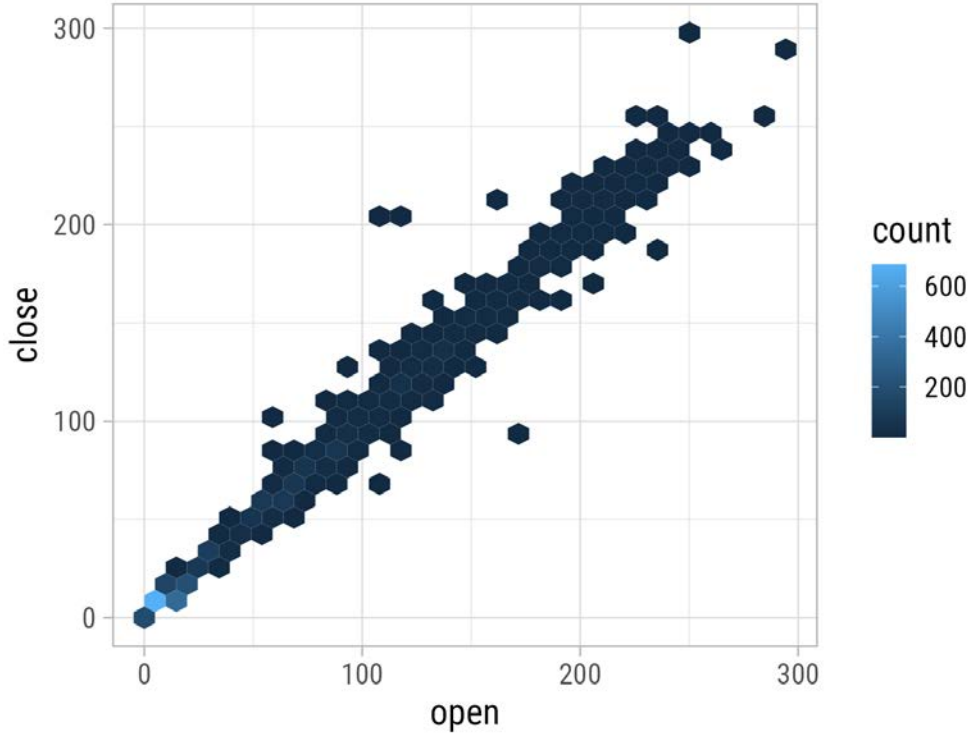
Create Any Chart Type: Hexagonal Bin Plot

Plots that bin the data can be helpful to deal with overplotting:

```
ggplot(data, aes(open, close)) +
  geom_point(alpha = .3)
```



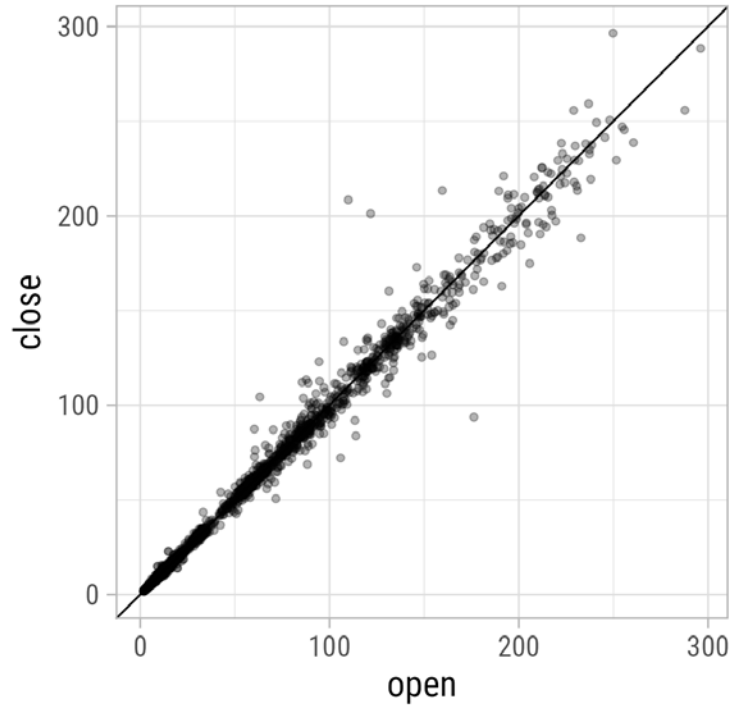

```
ggplot(data, aes(open, close)) +
  geom_hex()
```



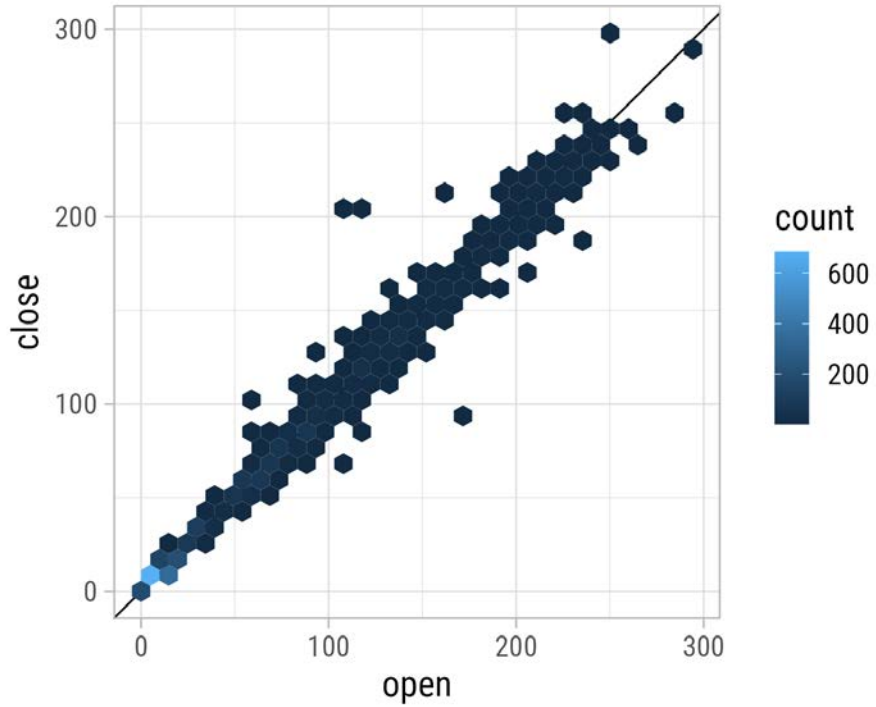
Create Any Chart Type: Hexagonal Bin Plot

Let's add the diagonal to the plot:

```
ggplot(data, aes(open, close)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_point(alpha = .3) +
  coord_fixed()
```



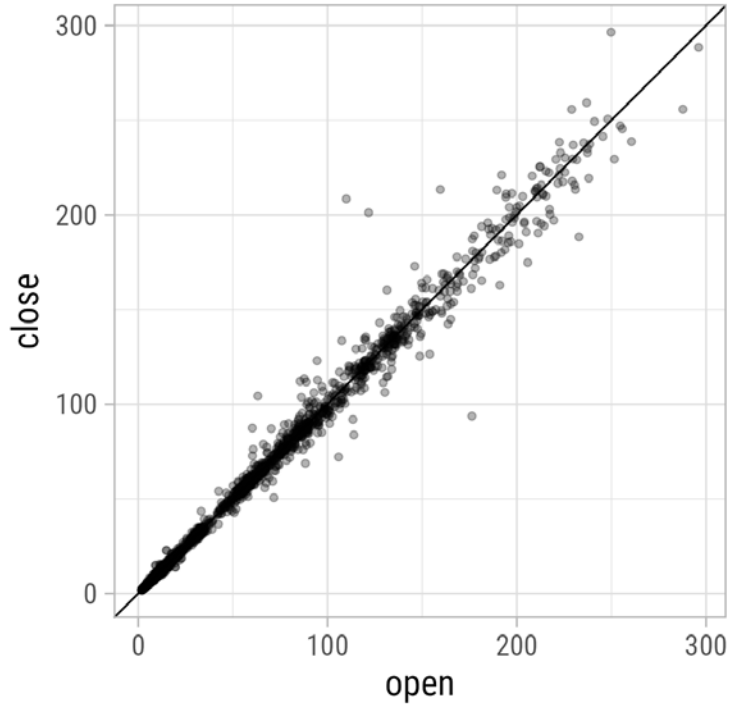

```
ggplot(data, aes(open, close)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_hex() +
  coord_fixed()
```



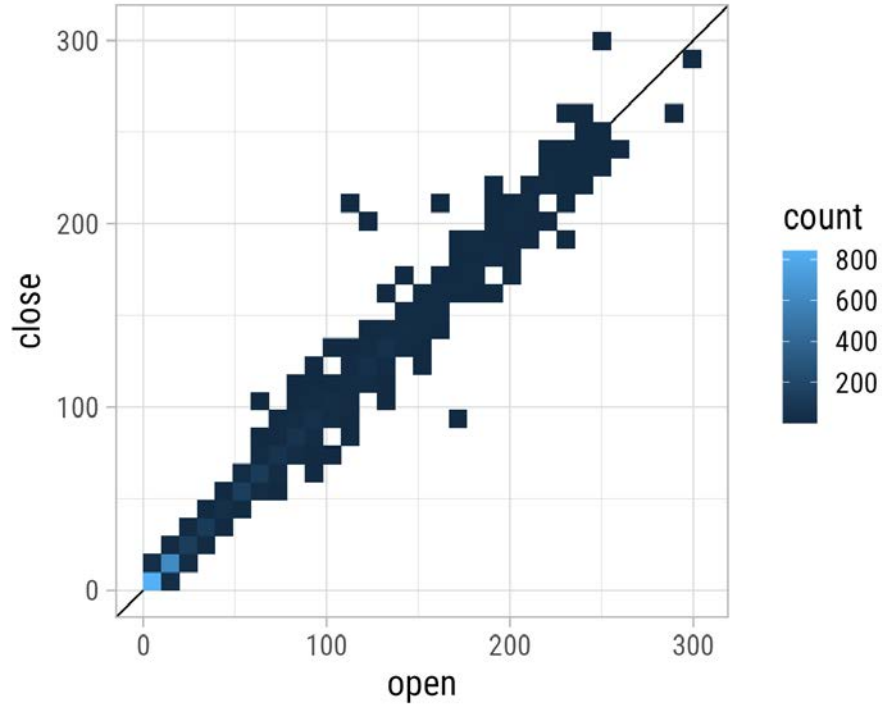
Create Any Chart Type: Rectangular Bin Plot

If you don't like hexagonals, use rectangles!

```
ggplot(data, aes(open, close)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_point(alpha = .3) +
  coord_fixed()
```



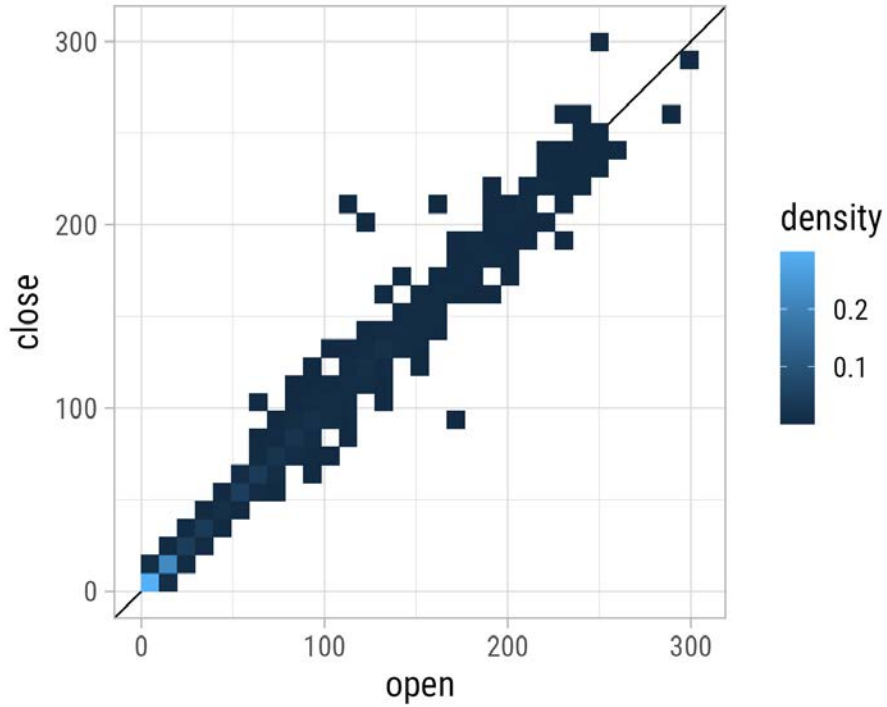

```
ggplot(data, aes(open, close)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_bin2d() +
  coord_fixed()
```



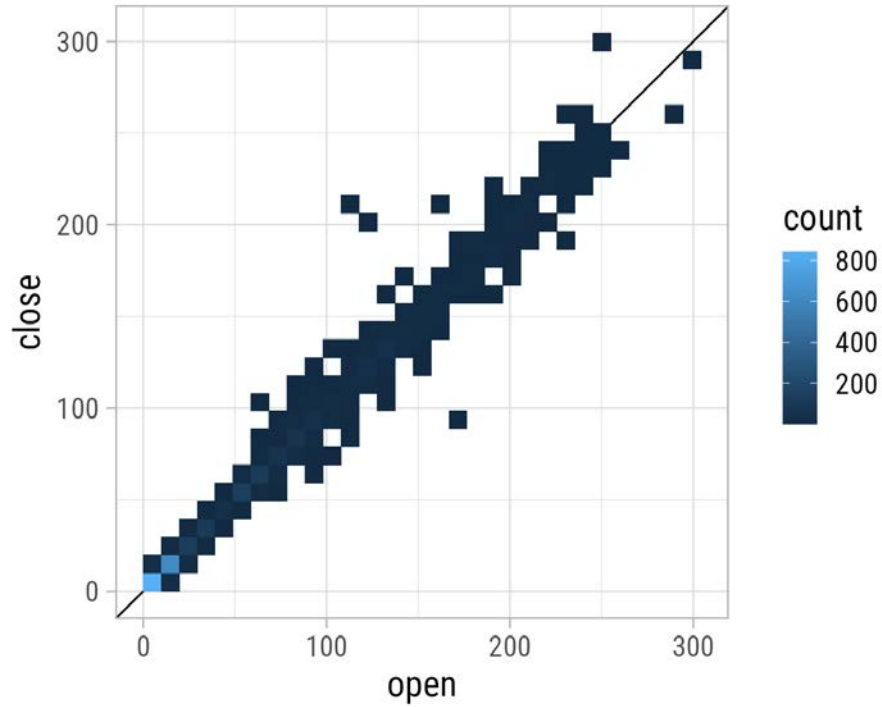
Create Any Chart Type: Rectangular Bin Plot

Actually, they map to a hidden summary statistic that is calculated by {ggplot2}:

```
ggplot(data, aes(open, close)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_bin2d(aes(fill = ..density..)) +
  coord_fixed()
```



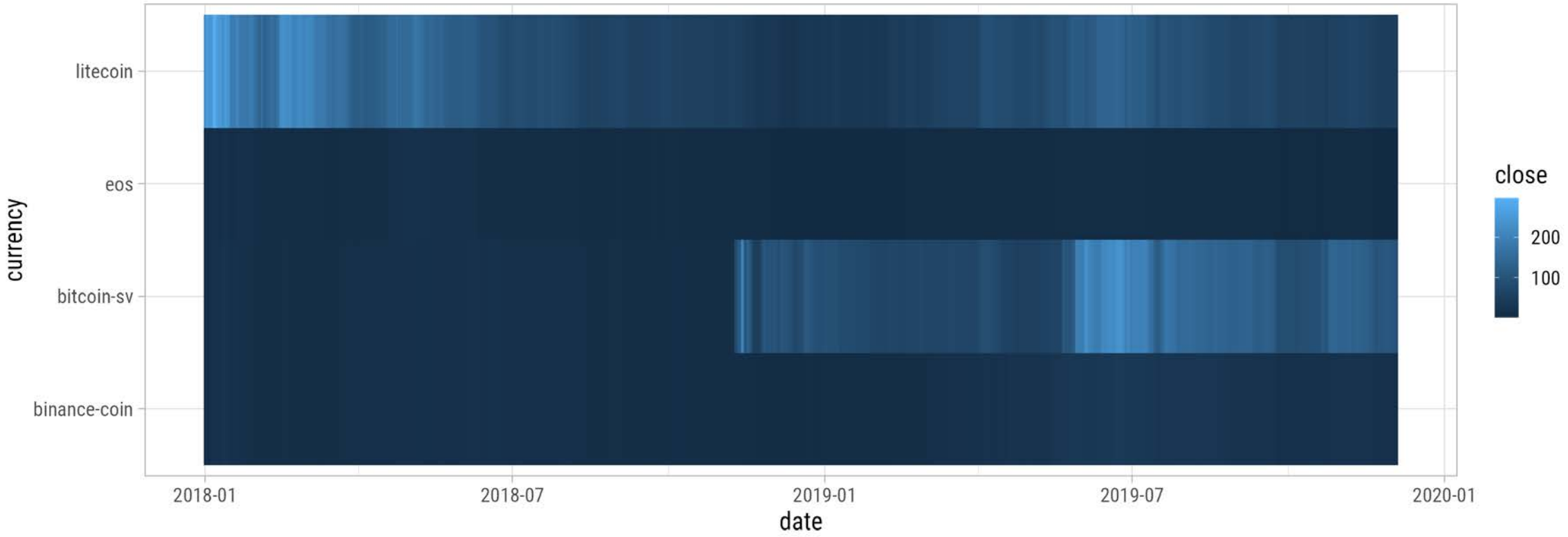

```
ggplot(data, aes(open, close)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_bin2d(aes(fill = ..count..)) +
  coord_fixed()
```



Create Any Chart Type: Heatmaps

With `geom_tile()` one can create heatmaps:

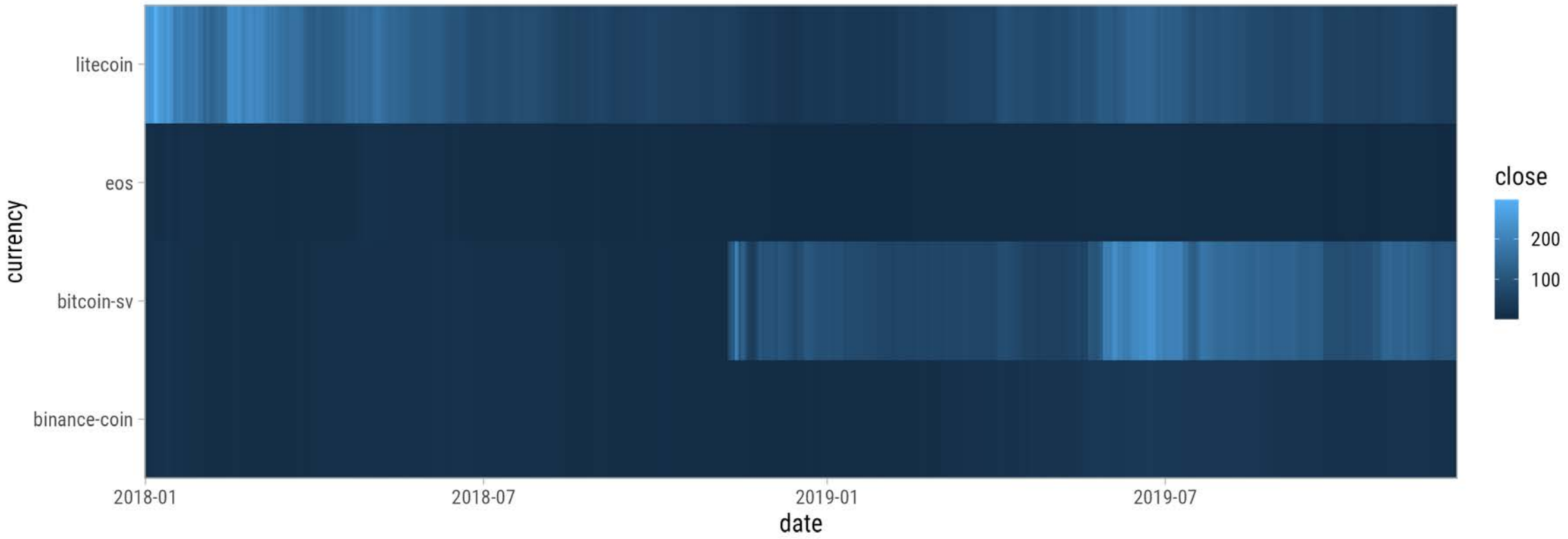
```
ggplot(data, aes(date, currency, fill = close)) +
  geom_tile()
```



Create Any Chart Type: Heatmaps

With `geom_tile()` one can create heatmaps:

```
ggplot(data, aes(date, currency, fill = close)) +
  geom_tile() +
  coord_cartesian(expand = FALSE)
```

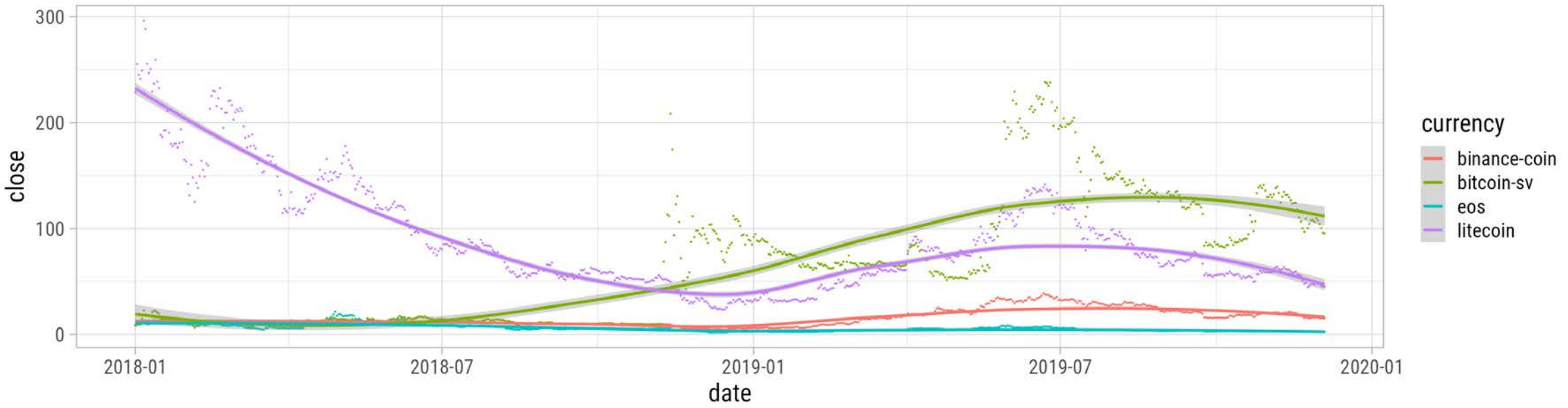


Statistical Transformations

Statistical Transformations: Draw Fittings

You can directly add smoothed conditional means:

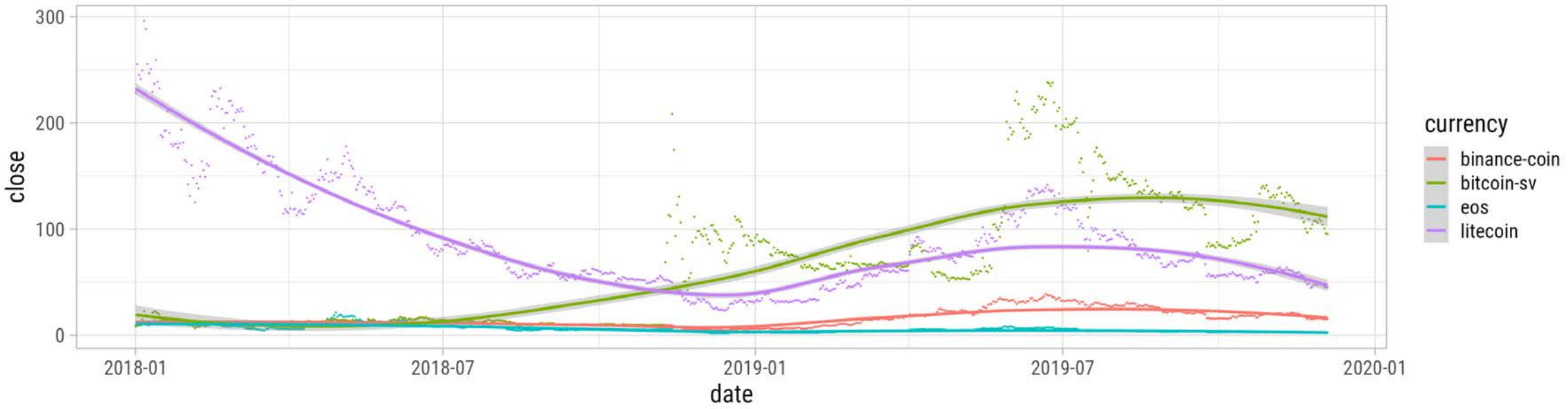
```
ggplot(data, aes(date, close, color = currency)) +
  geom_point(size = .1) +
  stat_smooth()
```



Statistical Transformations: Draw Fittings

You can directly add smoothed conditional means:

```
ggplot(data, aes(date, close, color = currency)) +
  geom_point(size = .1) +
  stat_smooth()
```

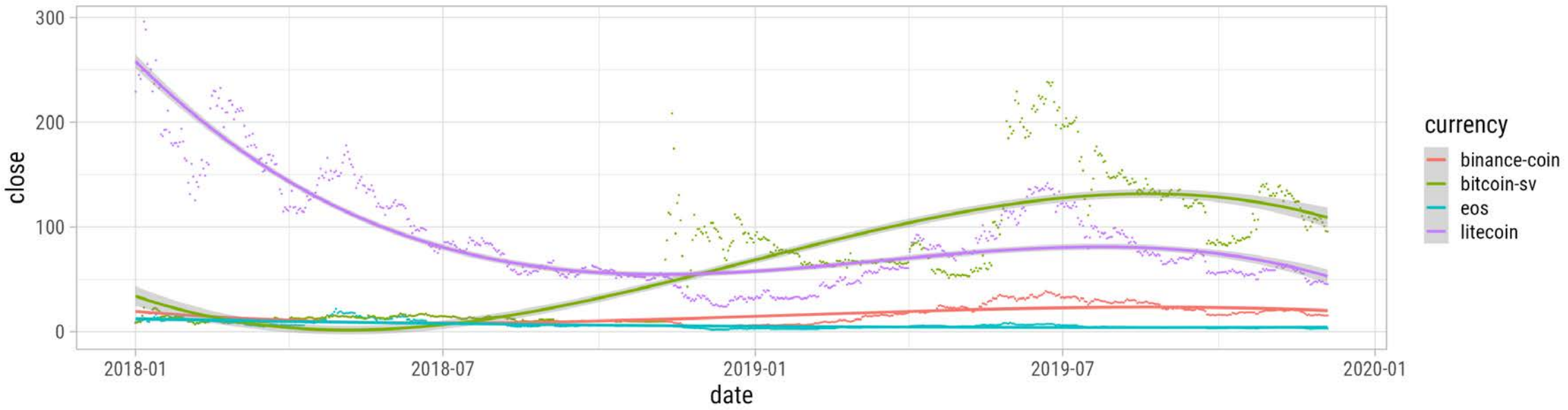


By default this adds a **LOESS** (locally weighted scatter plot smoothing) or a **GAM** (generalized additive model) depending on the number of data points (GAM in case of ≥ 1000 observations).

Statistical Transformations: Draw Fittings

You can specify the fitting method and the formula:

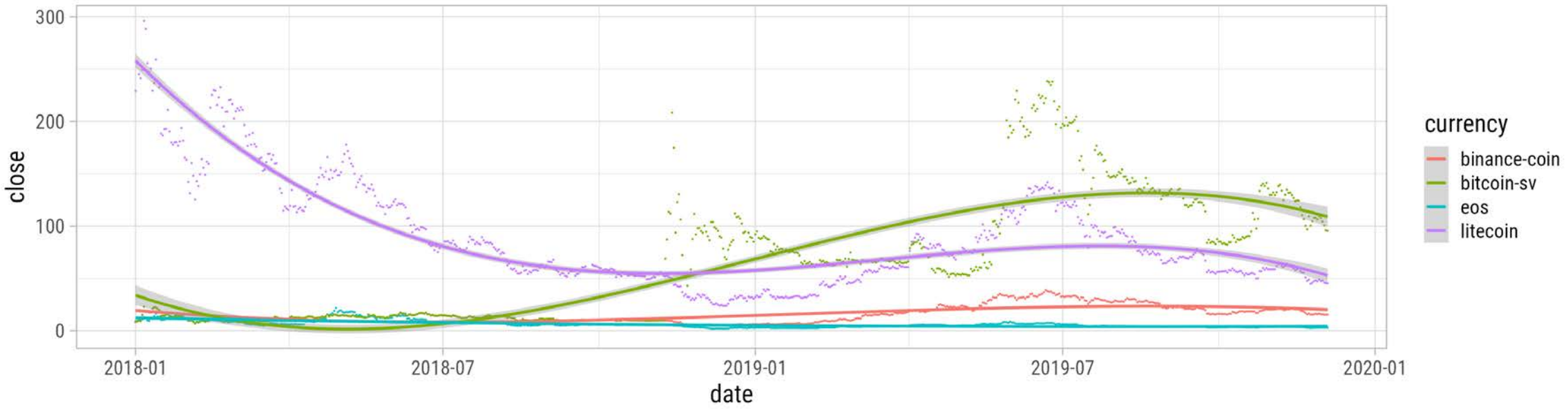
```
ggplot(data, aes(date, close, color = currency)) +
  geom_point(size = .1) +
  stat_smooth(method = "lm", formula = y ~ x + I(x^2) +  I(x^3) + I(x^4))
```



Statistical Transformations: Draw Fittings

You can specify the fitting method and the formula:

```
ggplot(data, aes(date, close, color = currency)) +
  geom_point(size = .1) +
  stat_smooth(method = "lm", formula = y ~ x + I(x^2) +  I(x^3) + I(x^4))
```

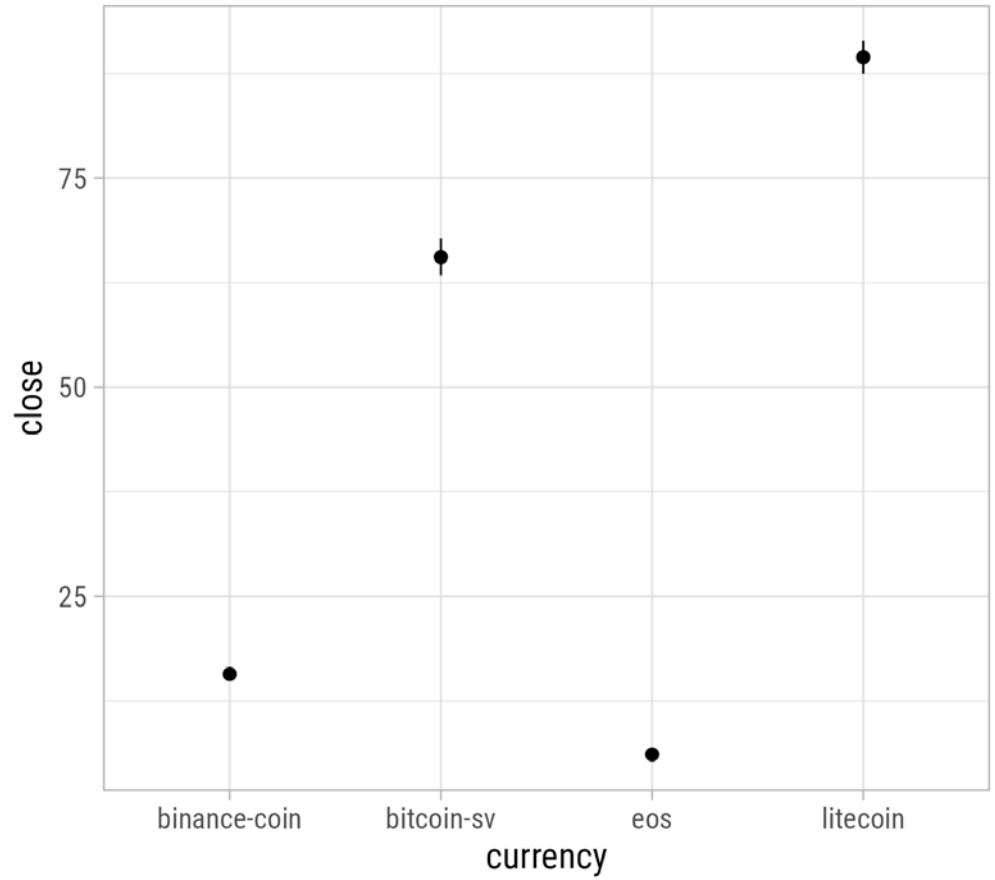


Other methods such as `method = "lm"` (without an explicit formula) for simple linear regressions and `method = "glm"` for generalized linear models are available as well.

Statistical Transformations: Calculate Summaries

Without pre-calculations one can easily plot data summaries:

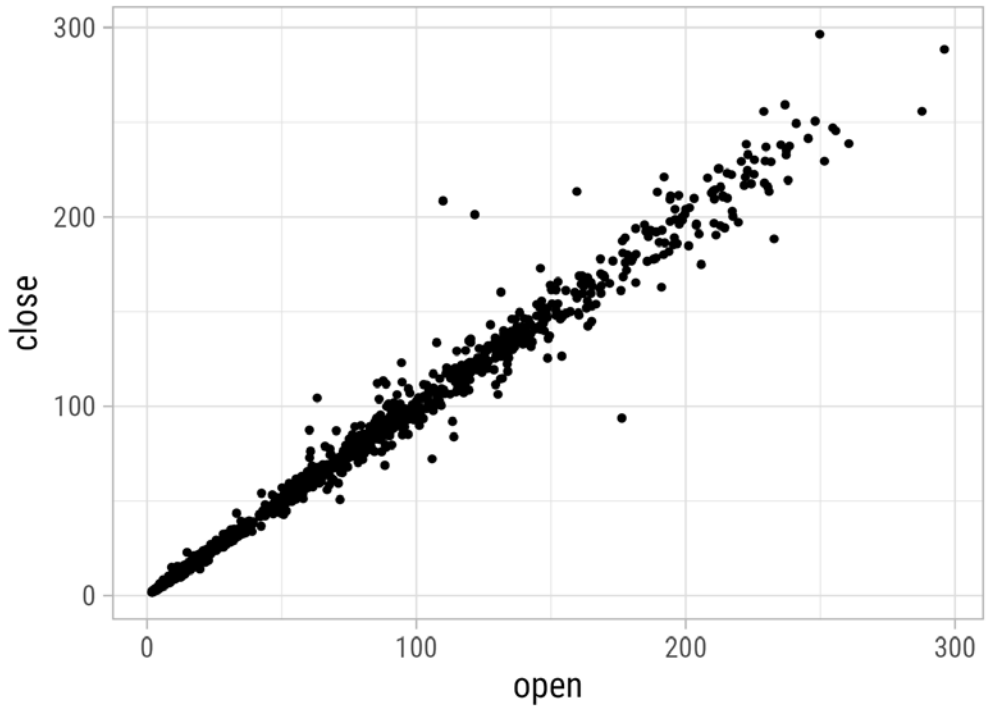
```
ggplot(data, aes(currency, close)) +
  stat_summary()
```



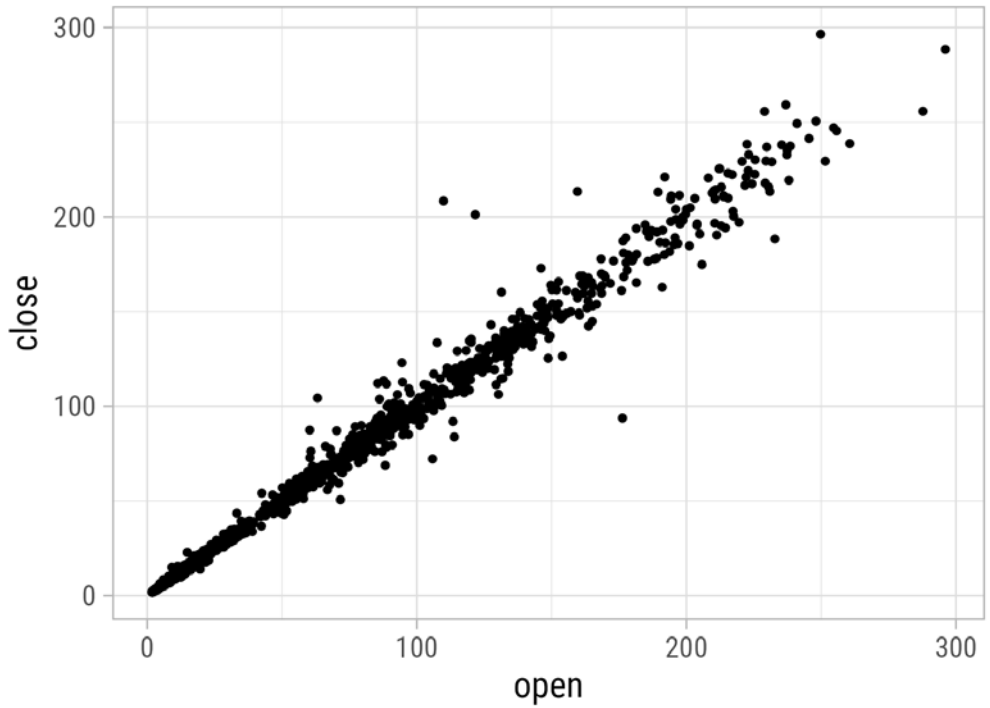
`stat_*()` versus `geom_*()`

You can go both ways — `stat_x(geom = "y") == geom_y(stat = "x")`:

```
ggplot(data, aes(open, close)) +
  stat_identity(geom = "point")
```



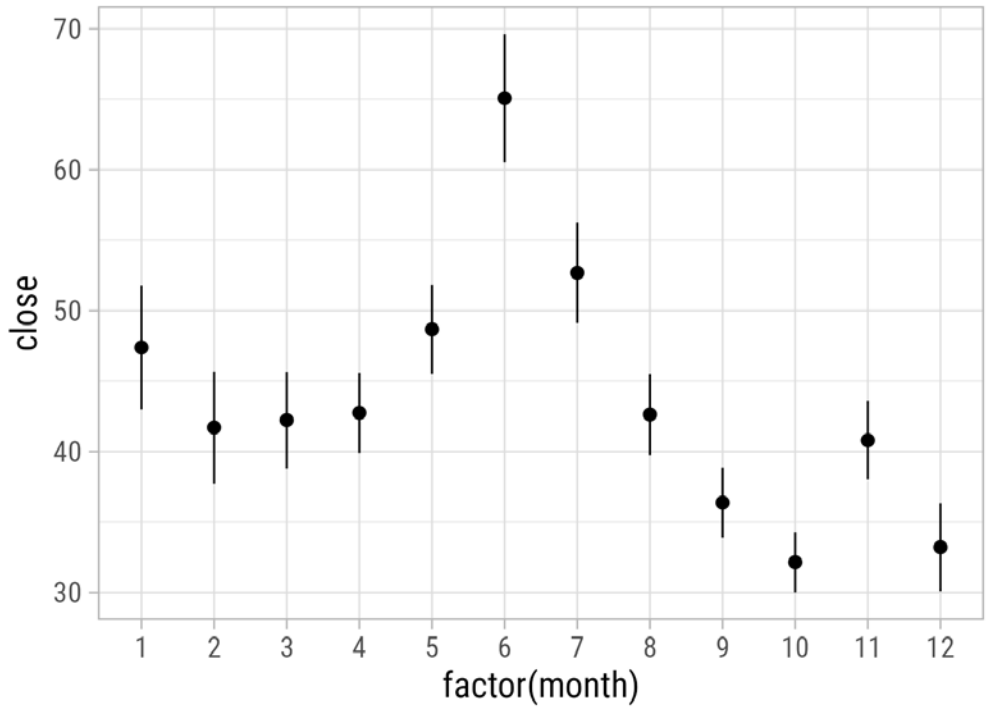

```
ggplot(data, aes(open, close)) +
  geom_point(stat = "identity")
```



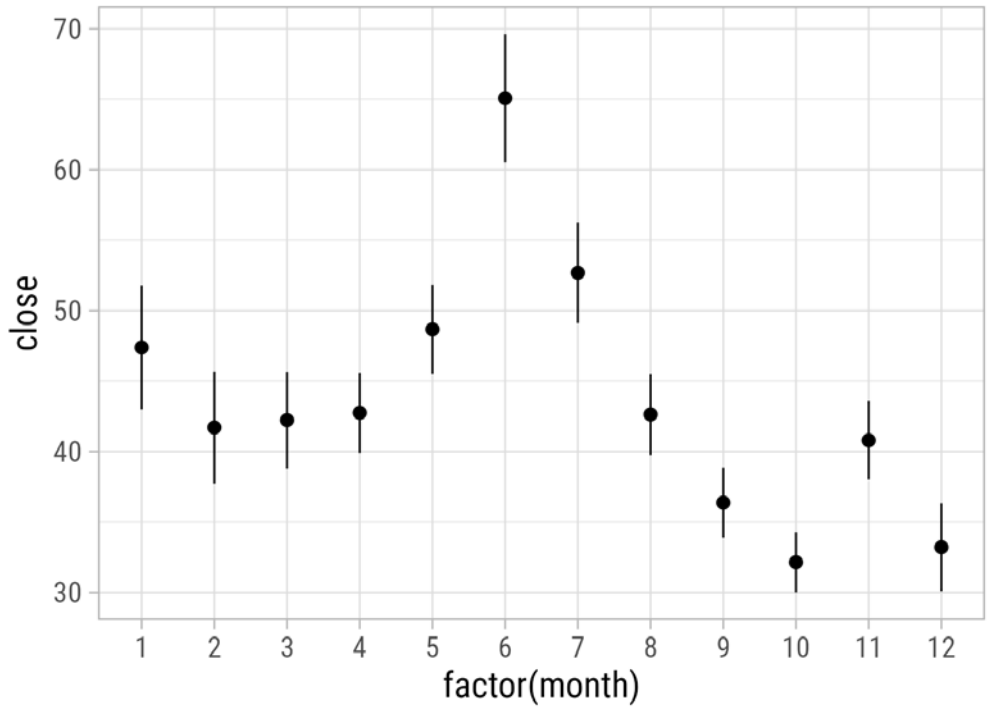
`stat_*()` versus `geom_*()`

You can go both ways — `stat_x(geom = "y") == geom_y(stat = "x")`:

```
ggplot(data, aes(factor(month), close)) +
  stat_summary(geom = "pointrange")
```



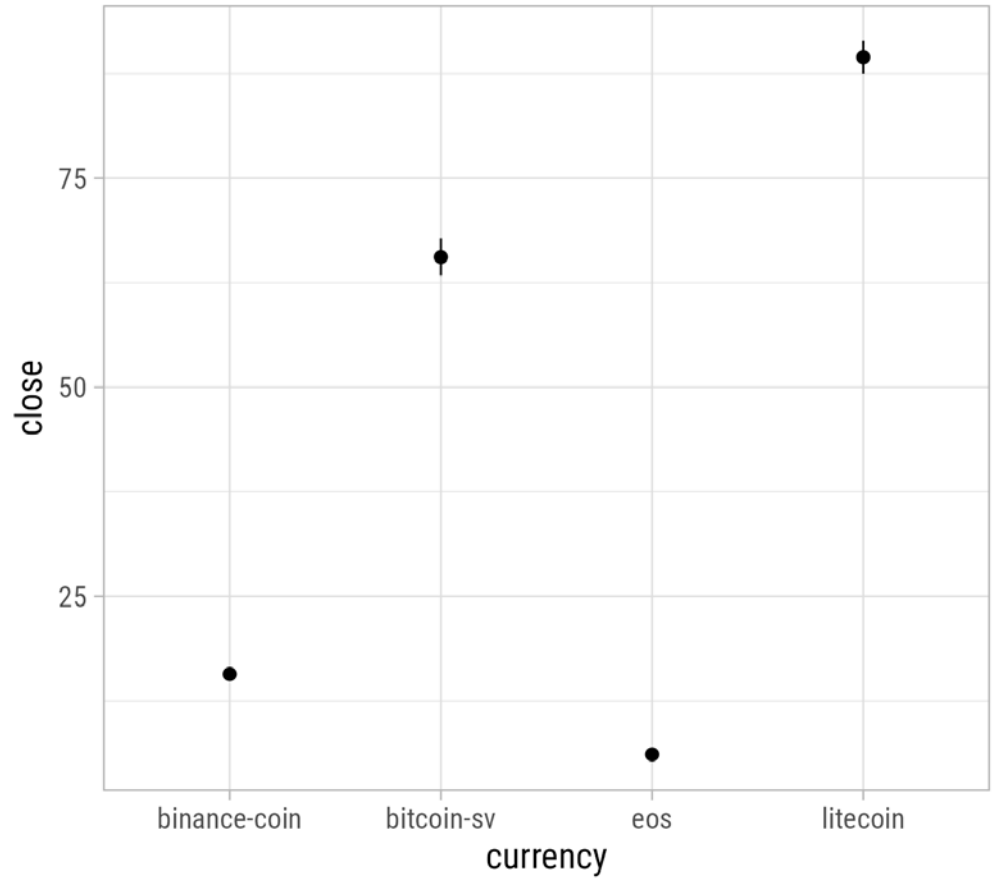

```
ggplot(data, aes(factor(month), close)) +
  geom_pointrange(stat = "summary")
```



Statistical Transformations: Calculate Summaries

Without pre-calculations one can easily plot data summaries:

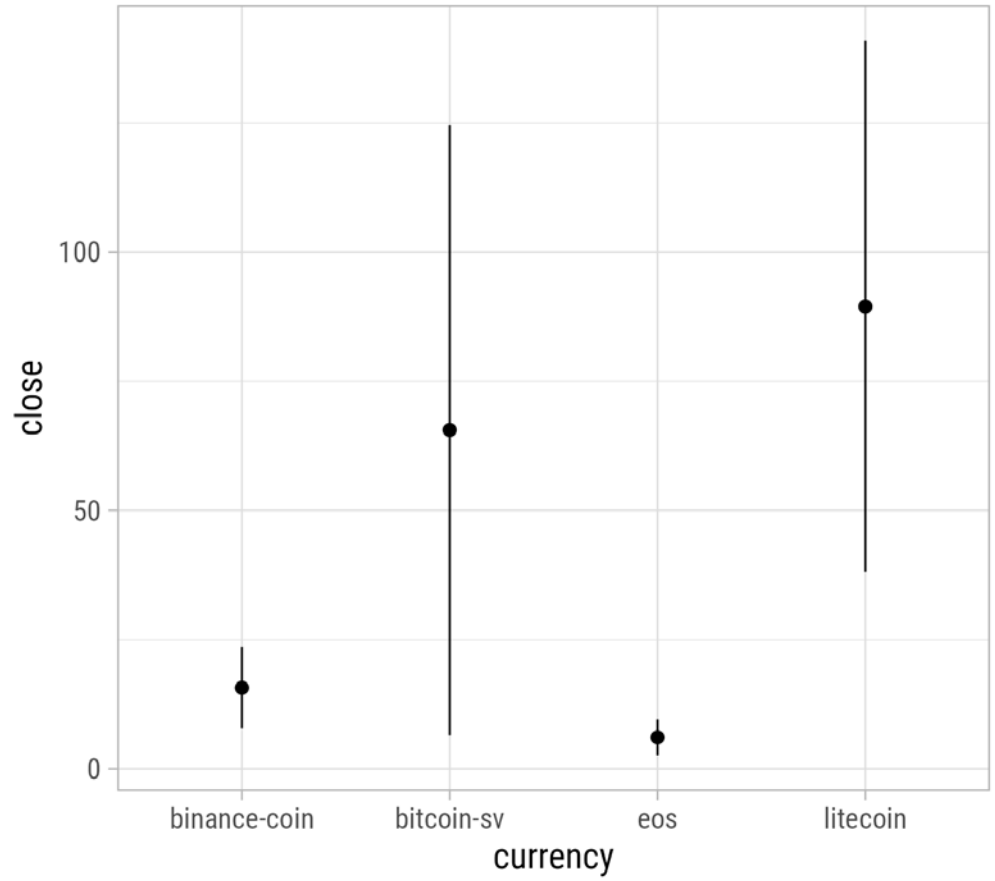
```
ggplot(data, aes(currency, close)) +
  stat_summary(
    geom = "pointrange",
    fun.data = "mean_se"
  )
```



Statistical Transformations: Calculate Summaries

Without pre-calculations one can easily plot data summaries:

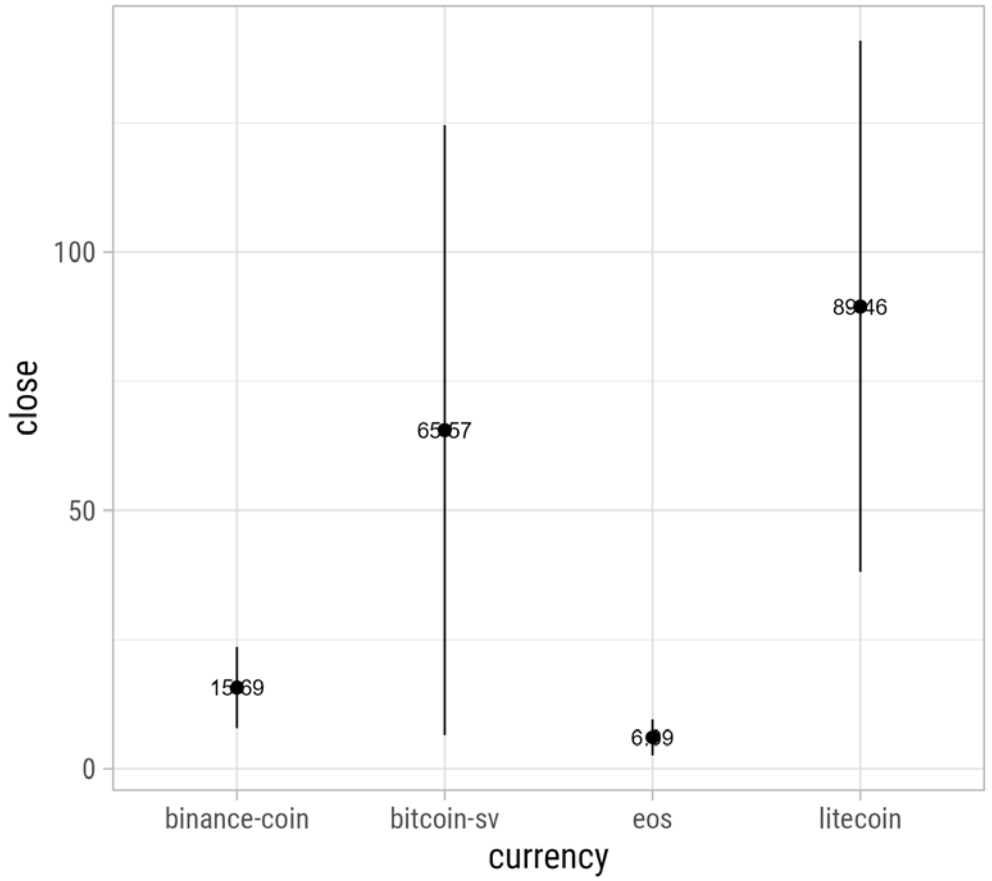
```
ggplot(data, aes(currency, close)) +
  stat_summary(
    fun = function(x) mean(x),
    fun.min = function(x) mean(x) - sd(x),
    fun.max = function(x) mean(x) + sd(x)
  )
```



Statistical Transformations: Calculate Summaries

Without pre-calculations one can easily plot data summaries:

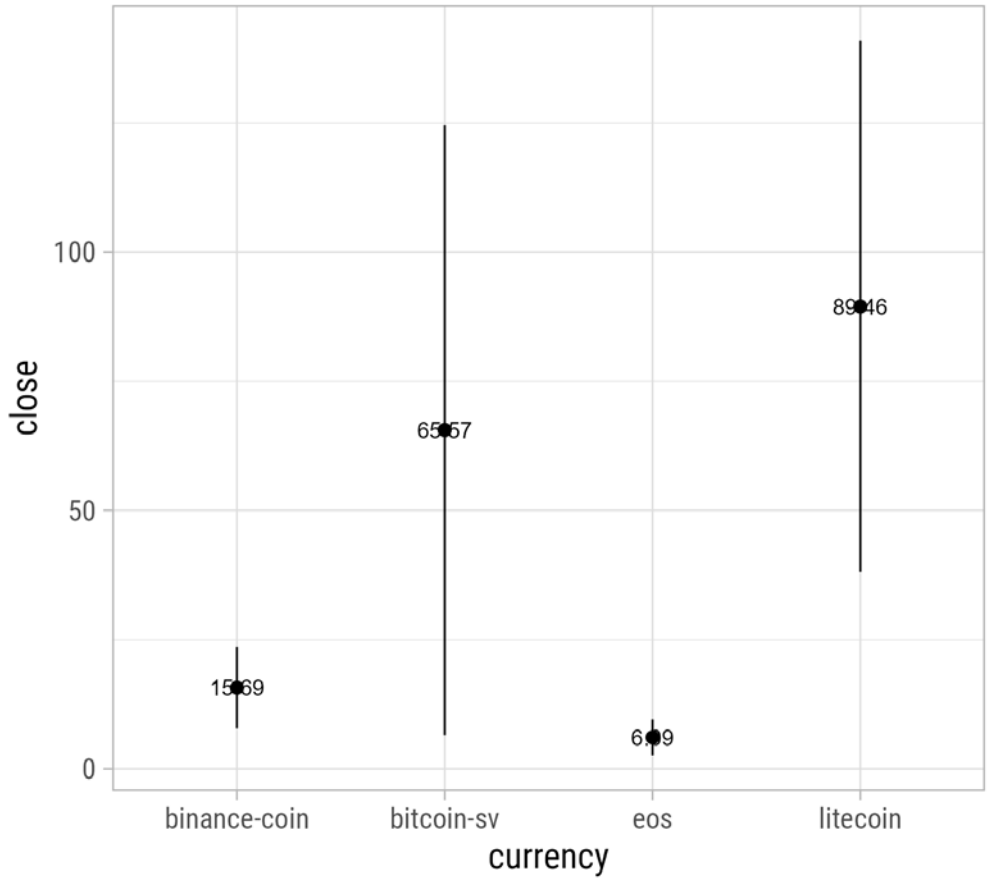
```
ggplot(data, aes(currency, close)) +
  stat_summary(
    fun = function(x) mean(x),
    fun.min = function(x) mean(x) - sd(x),
    fun.max = function(x) mean(x) + sd(x)
  ) +
  stat_summary(
      geom = "text",
      aes(label = round(..y.., 2))
  )
```



Statistical Transformations: Calculate Summaries

Without pre-calculations one can easily plot data summaries:

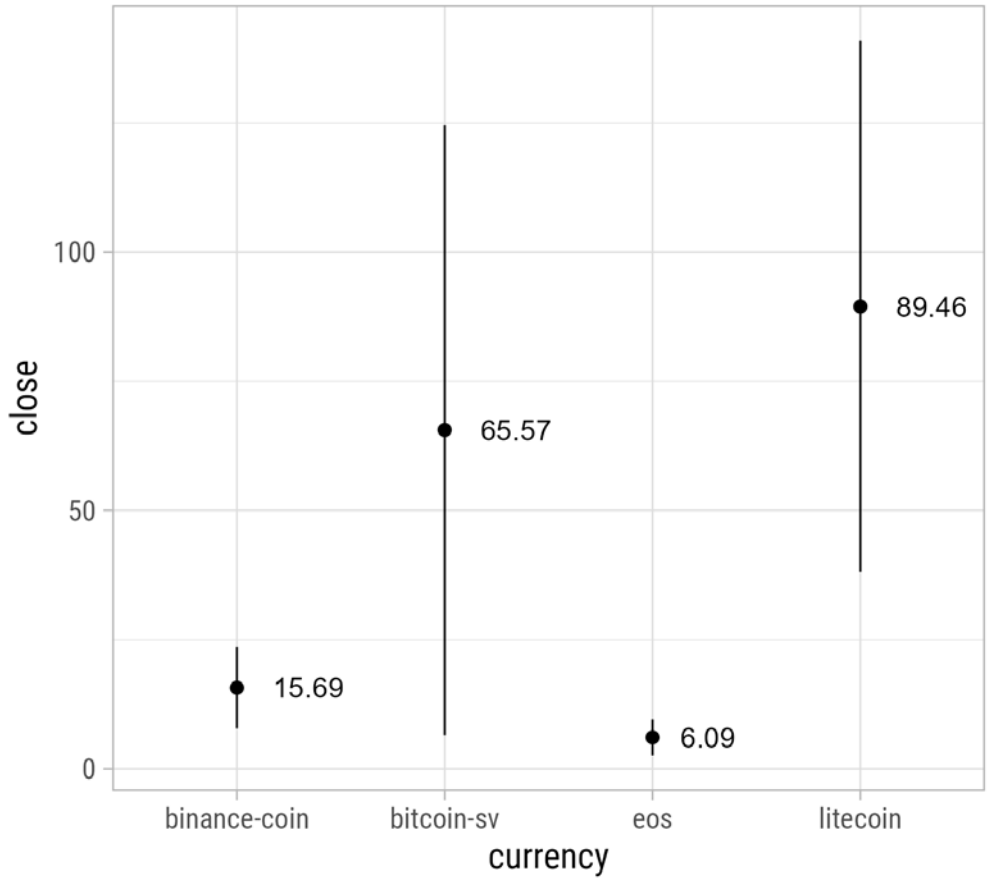
```
ggplot(data, aes(currency, close)) +
  stat_summary(
    fun = function(x) mean(x),
    fun.min = function(x) mean(x) - sd(x),
    fun.max = function(x) mean(x) + sd(x)
  ) +
  stat_summary(
      geom = "text",
      fun = function(x) mean(x),
      aes(label = round(..y.., 2))
  )
```



Statistical Transformations: Calculate Summaries

Without pre-calculations one can easily plot data summaries:

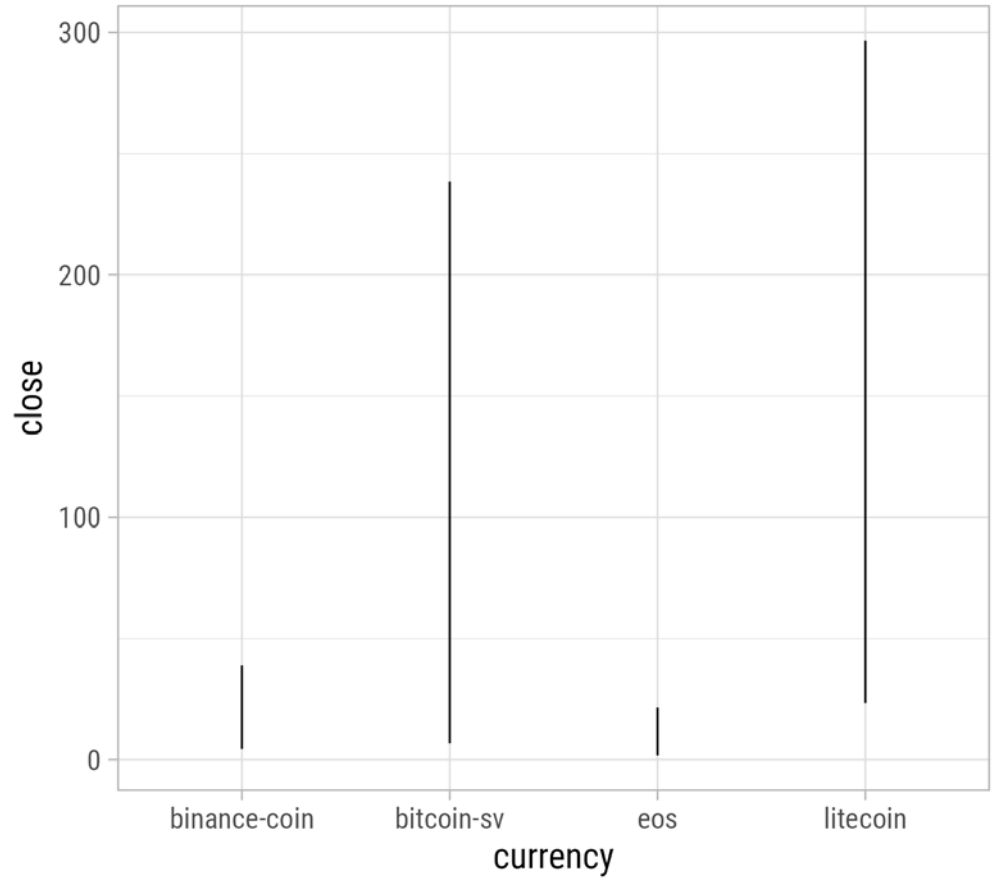
```
ggplot(data, aes(currency, close)) +
  stat_summary(
    fun = function(x) mean(x),
    fun.min = function(x) mean(x) - sd(x),
    fun.max = function(x) mean(x) + sd(x)
  ) +
  stat_summary(
      geom = "text",
      fun = function(x) mean(x),
      aes(label = round(..y.., 2)),
      size = 5,
      hjust = -.5
  )
```



Create Any Chart Type: Dumbbell Plot

Without pre-calculations one can easily plot data summaries:

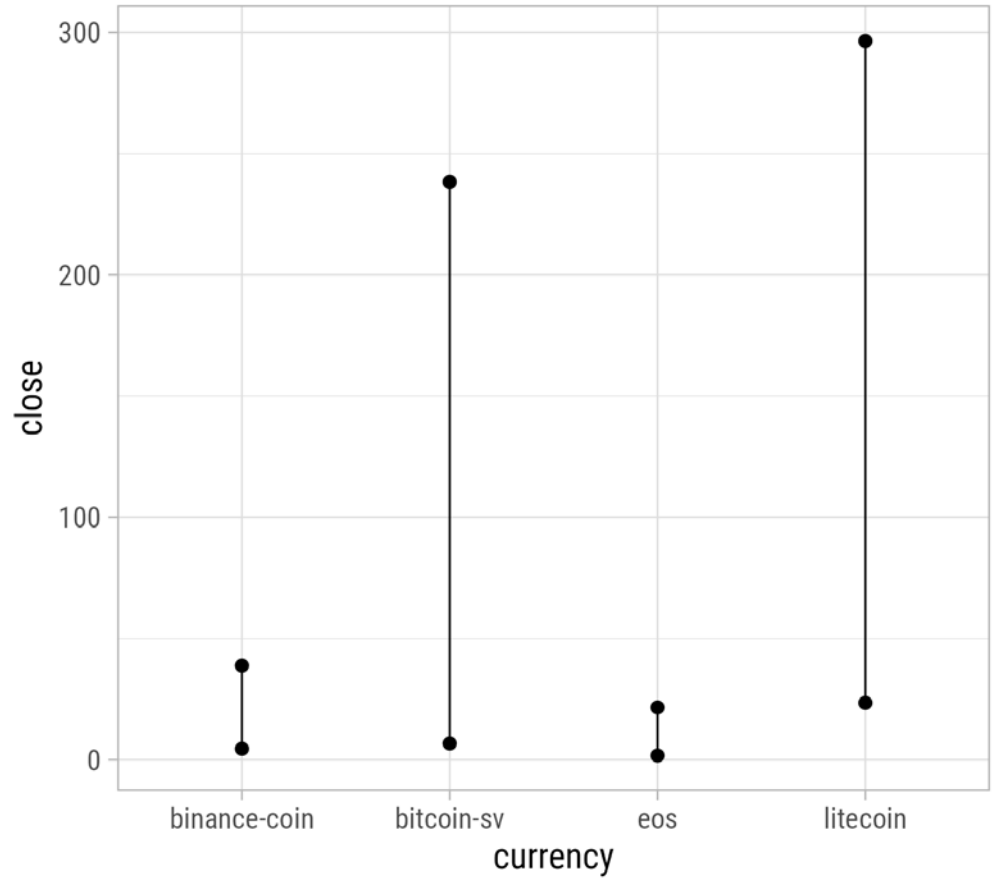
```
ggplot(data, aes(currency, close)) +
  stat_summary(
    geom = "linerange",
    fun.min = "min",
    fun.max = "max"
  )
```



Create Any Chart Type: Dumbbell Plot

Without pre-calculations one can easily plot data summaries:

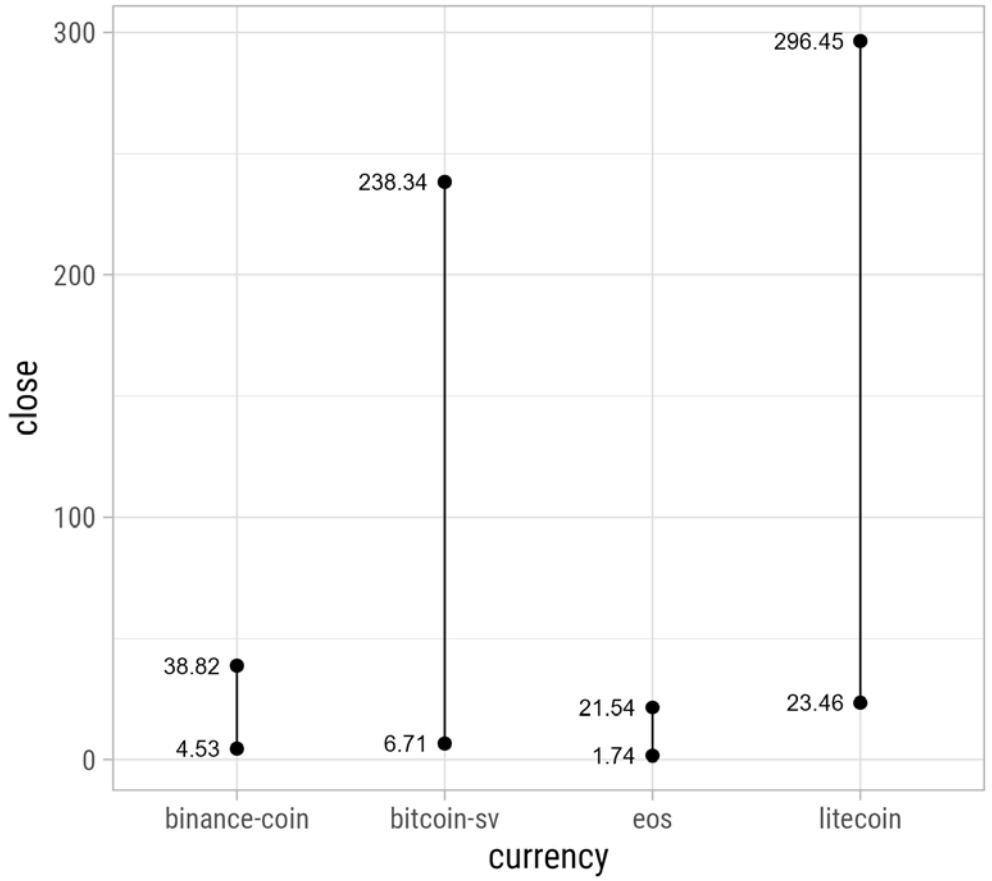
```
ggplot(data, aes(currency, close)) +
  stat_summary(
    geom = "linerange",
    fun.min = "min",
    fun.max = "max"
  ) +
  stat_summary(
    fun = "range"
  )
```



Create Any Chart Type: Dumbbell Plot

Without pre-calculations one can easily plot data summaries:

```
ggplot(data, aes(currency, close)) +
  stat_summary(
    geom = "linerange",
    fun.min = "min",
    fun.max = "max"
  ) +
  stat_summary(
    fun = "range"
  ) +
  stat_summary(
    fun = "range",
    aes(label = round(..y.., 2)),
    geom = "text",
    size = 4,
    hjust = 1,
    position = position_nudge(x = -.08)
  )
```



Statistical Transformations: `stat_*()`

There are several other statistical transformations available:

- `stat_count` to count observations (e.g. for bar charts)
- `stat_bin` and `stat_bin2d` to count observations per bin (e.g. for histograms)
- `stat_density` and `stat_density2d` to compute kernel density estimates (e.g. for density curves)
- `stat_contour` and `stat_contour_filled` to retrieve threshold levels (e.g. for contour maps)
- `stat_function` to draw functions as continuous curves
- `stat_boxplot` to calculate the five summary statistics for a box plot
- `stat_ydensity` to compute vertical density estimates (e.g. for violin plots)

Exercise 1:

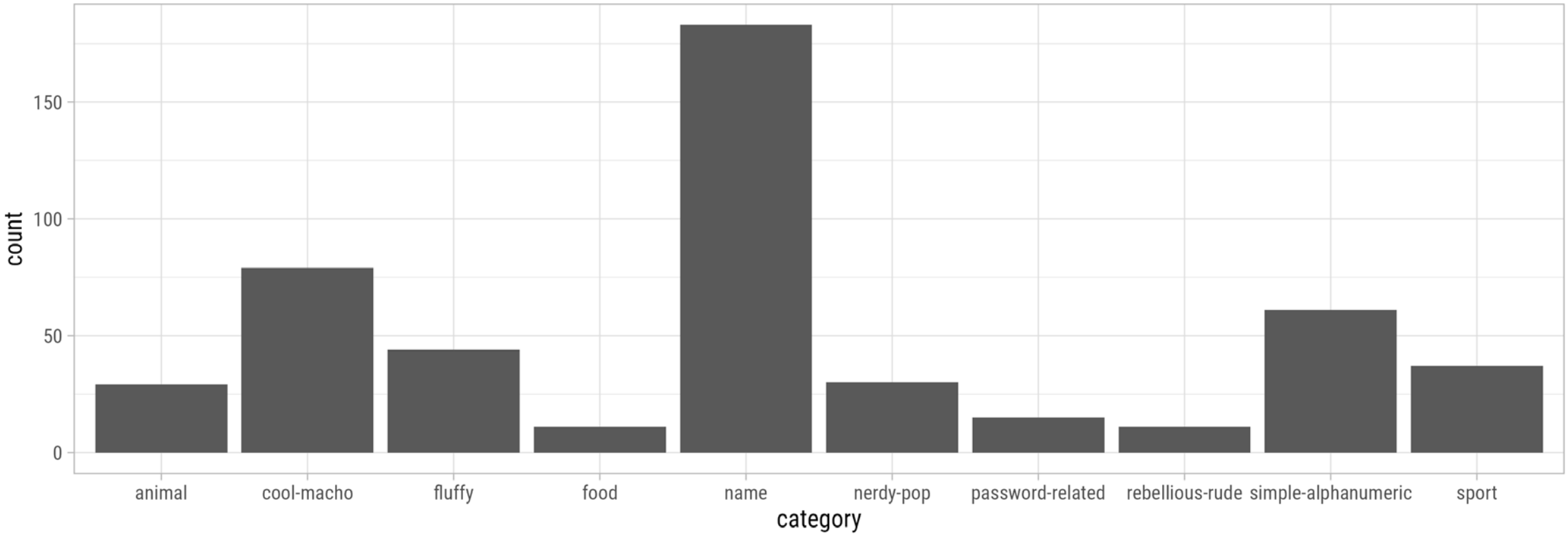
- Import the password data set: `https://raw.githubusercontent.com/Z3tt/ggplot-courses/master/data/passwords.csv`
- Visualize the number of passwords per category as bar chart.
- Add to each bar a label of the number of passwords.

Exercise 1: Import the Data

```
passwords <- readr::read_csv(
  "https://raw.githubusercontent.com/Z3tt/ggplot-courses/master/data/passwords.csv"
)

glimpse(passwords)
## Rows: 500
## Columns: 3
## $ rank     <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18~
## $ password <chr> "password", "123456", "12345678", "1234", "qwerty", "12345", ~
## $ category <chr> "password-related", "simple-alphanumeric", "simple-alphanumer~
```

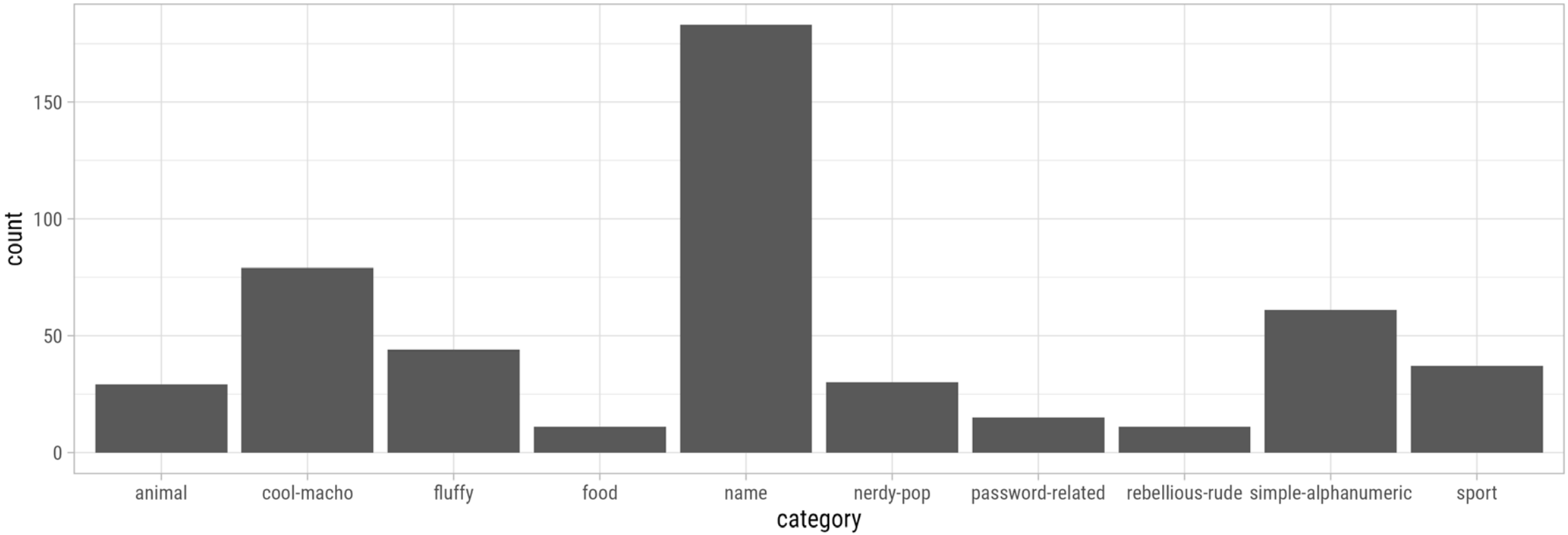
Exercise 1: Number of Passwords per Category

```
ggplot(passwords, aes(category)) +
  geom_bar() ## stat = "count" is the default
```



Exercise 1: Number of Passwords per Category

```
ggplot(passwords, aes(category)) +
  stat_count() ## geom = "bar" is the default
```



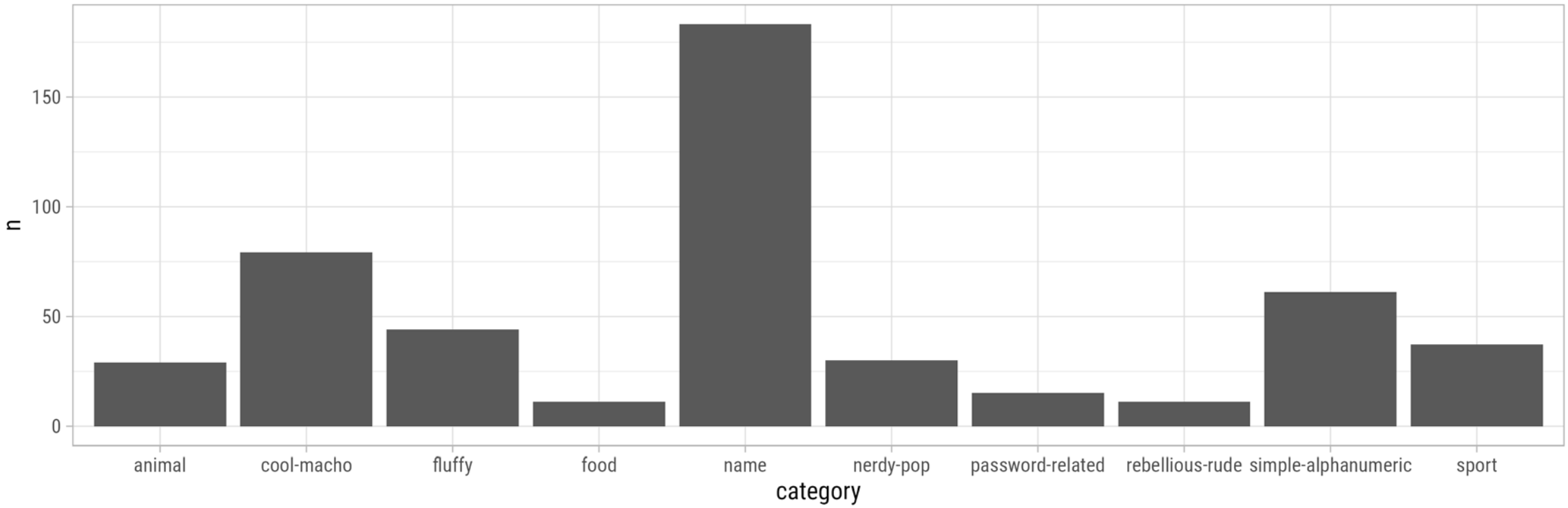
Exercise 1: Number of Passwords per Category

```
passwords %>%
  count(category)
```

```
## # A tibble: 10 x 2
##    category               n
##    <chr>              <int>
##  1 animal                29
##  2 cool-macho            79
##  3 fluffy                44
##  4 food                  11
##  5 name                 183
##  6 nerdy-pop             30
##  7 password-related      15
##  8 rebellious-rude       11
##  9 simple-alphanumeric   61
## 10 sport                 37
```

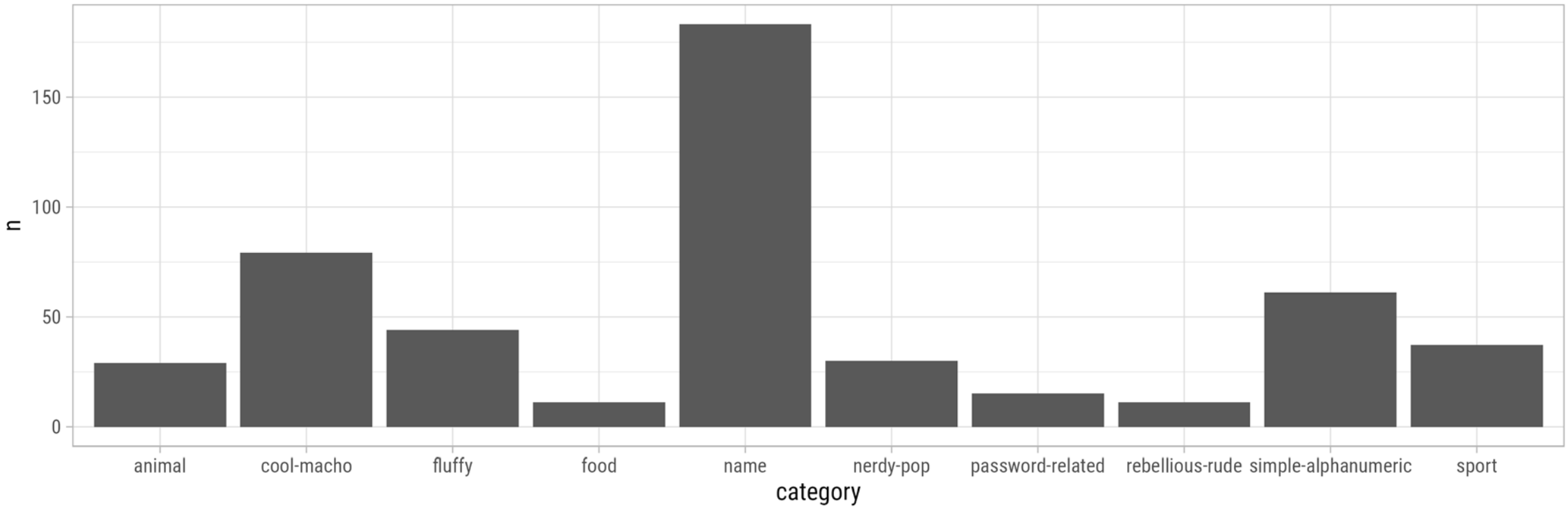
Exercise 1: Number of Passwords per Category

```
passwords %>%
  count(category) %>%
  ggplot(aes(category, n)) +
  geom_col()
```



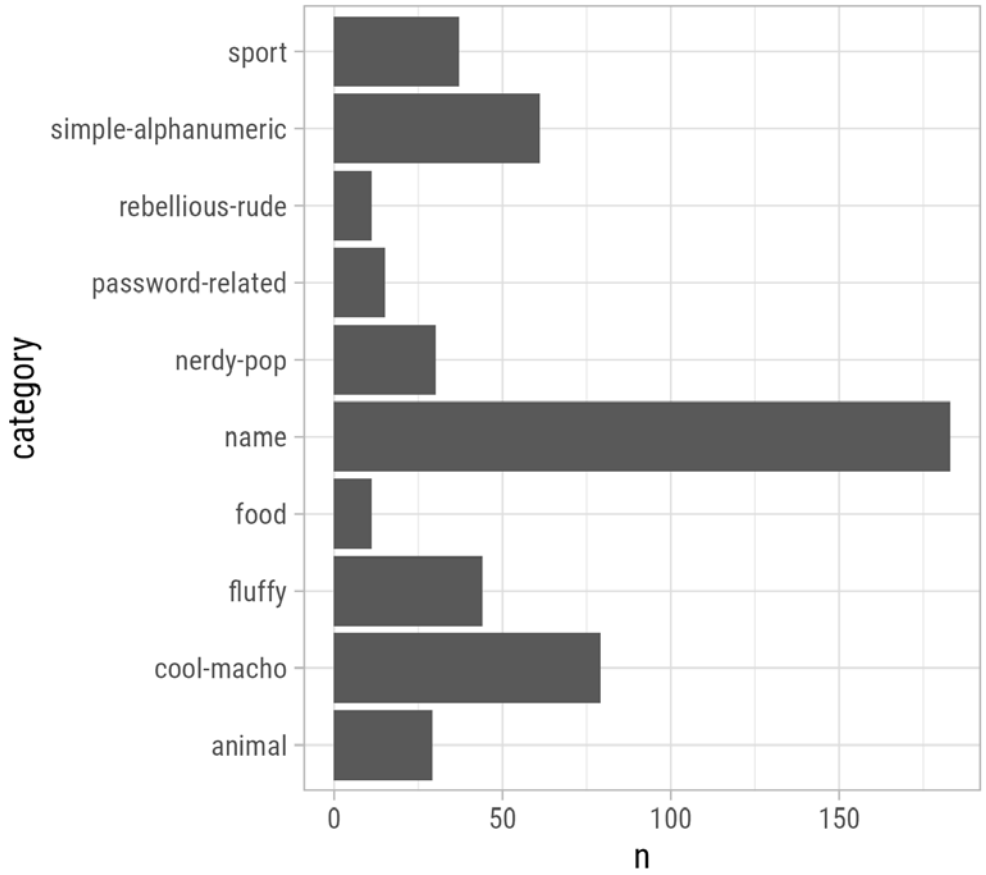
Exercise 1: Number of Passwords per Category

```
pw_count <- count(passwords, category)

ggplot(pw_count, aes(category, n)) +
  geom_col()
```

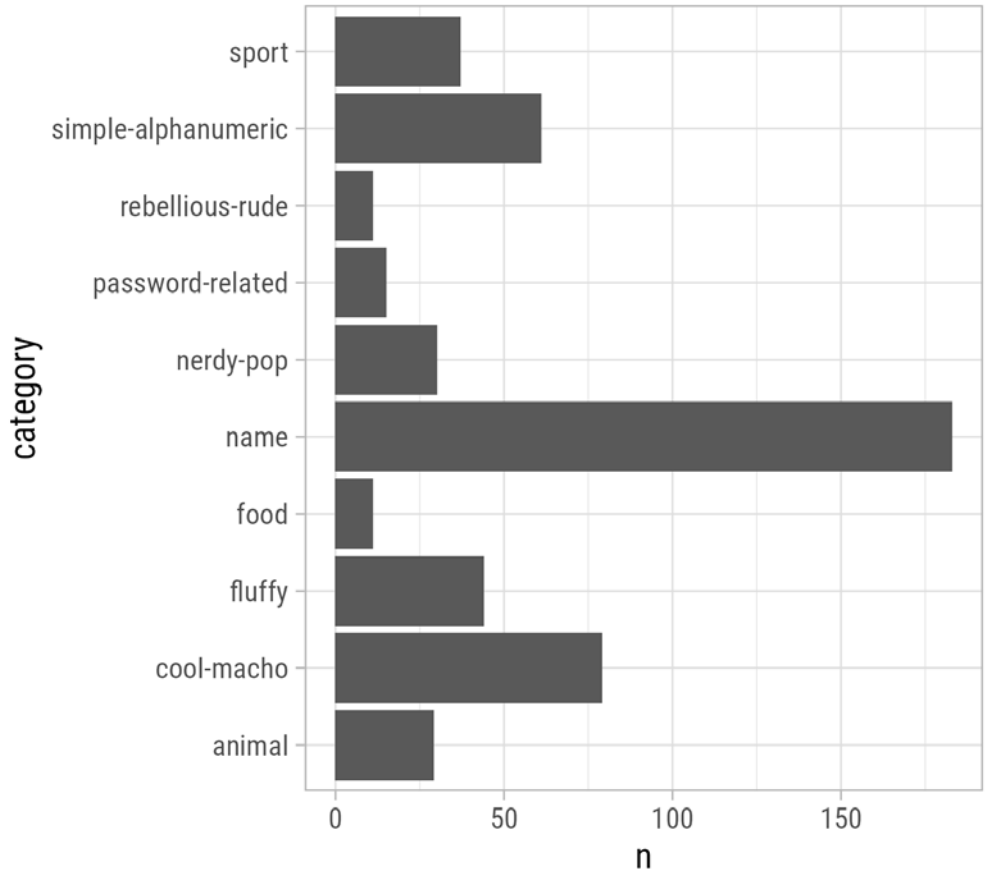


Exercise 1: Number of Passwords per Category

```
ggplot(pw_count, aes(n, category)) +
  geom_col()
```

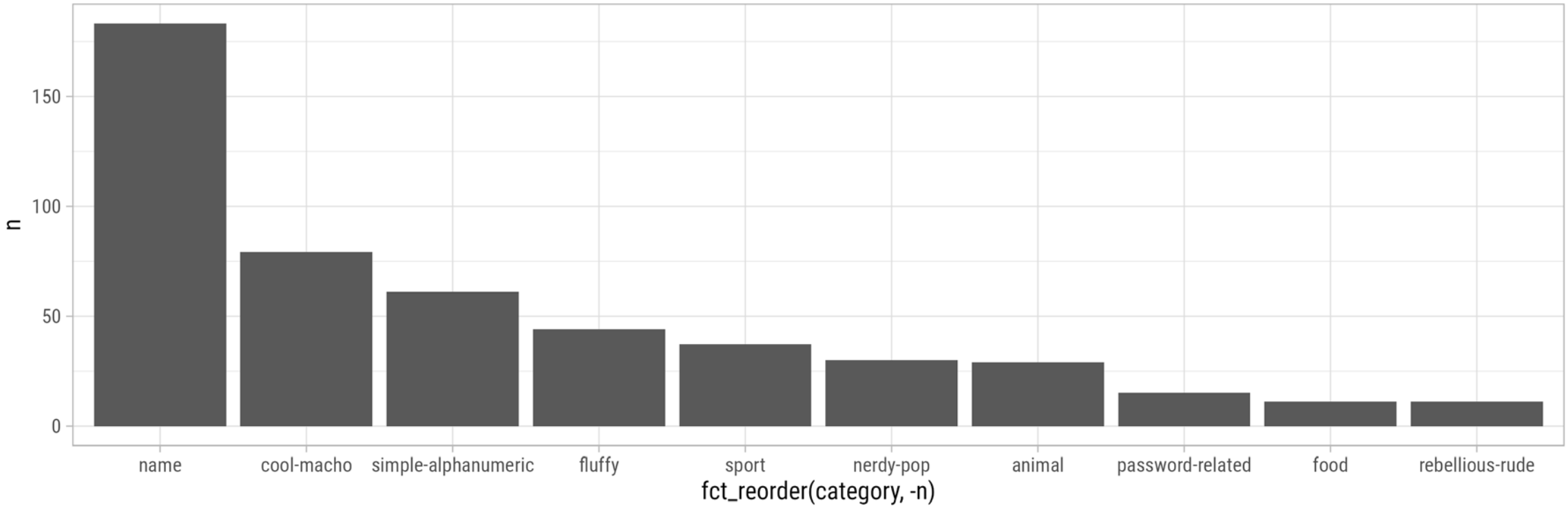



```
ggplot(pw_count, aes(n, category)) +
  geom_bar(stat = "identity")
```



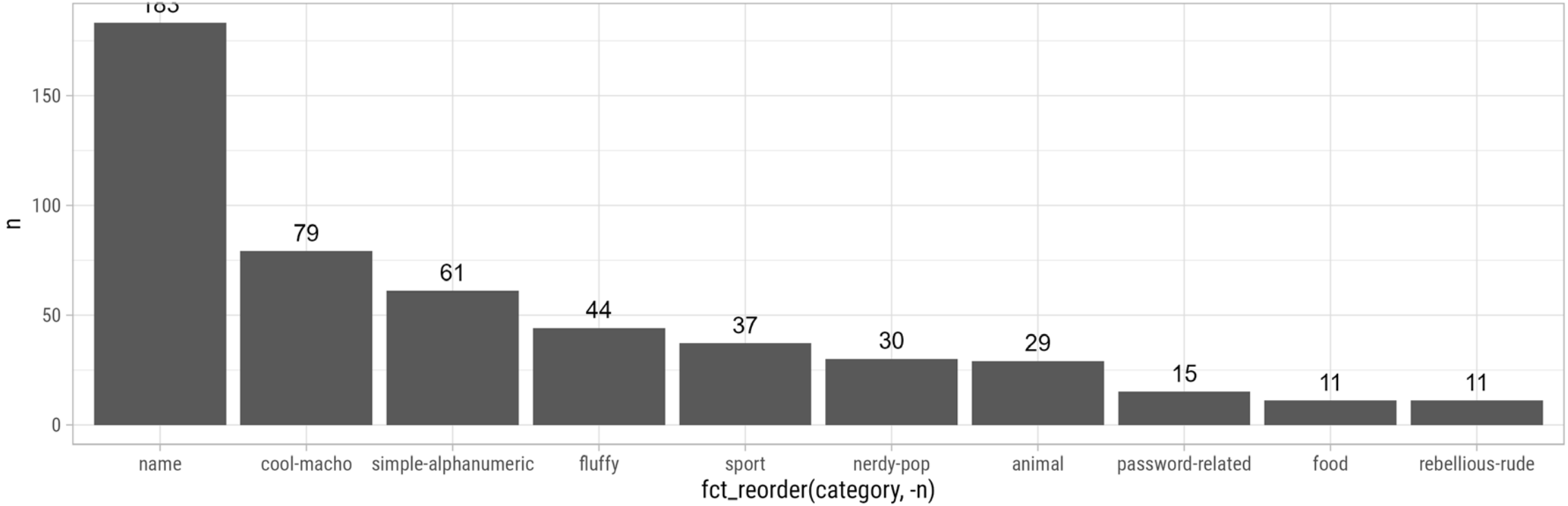
Exercise 1: Number of Passwords per Category

```
pw_count <- count(passwords, category)

ggplot(pw_count, aes(fct_reorder(category, -n), n)) +
  geom_col()
```



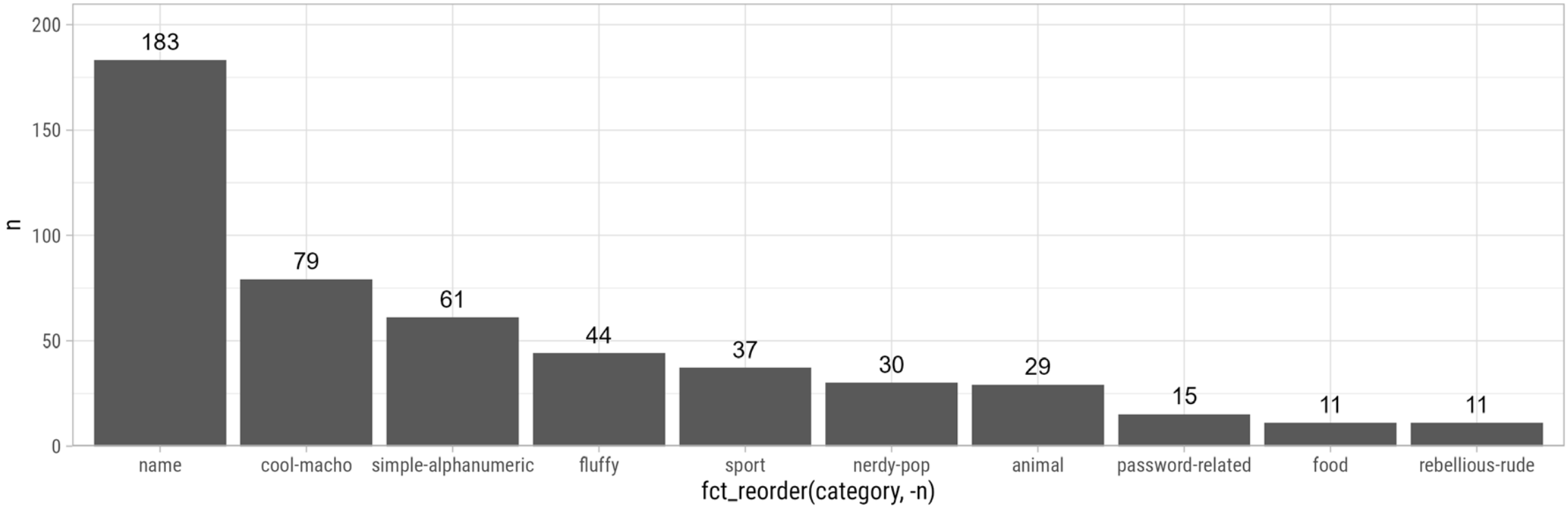
Exercise 1: Labels Number of Passwords per Category

```
ggplot(pw_count, aes(fct_reorder(category, -n), n)) +
  geom_col() +
  geom_text(aes(label = n), vjust = -.6, size = 6)
```



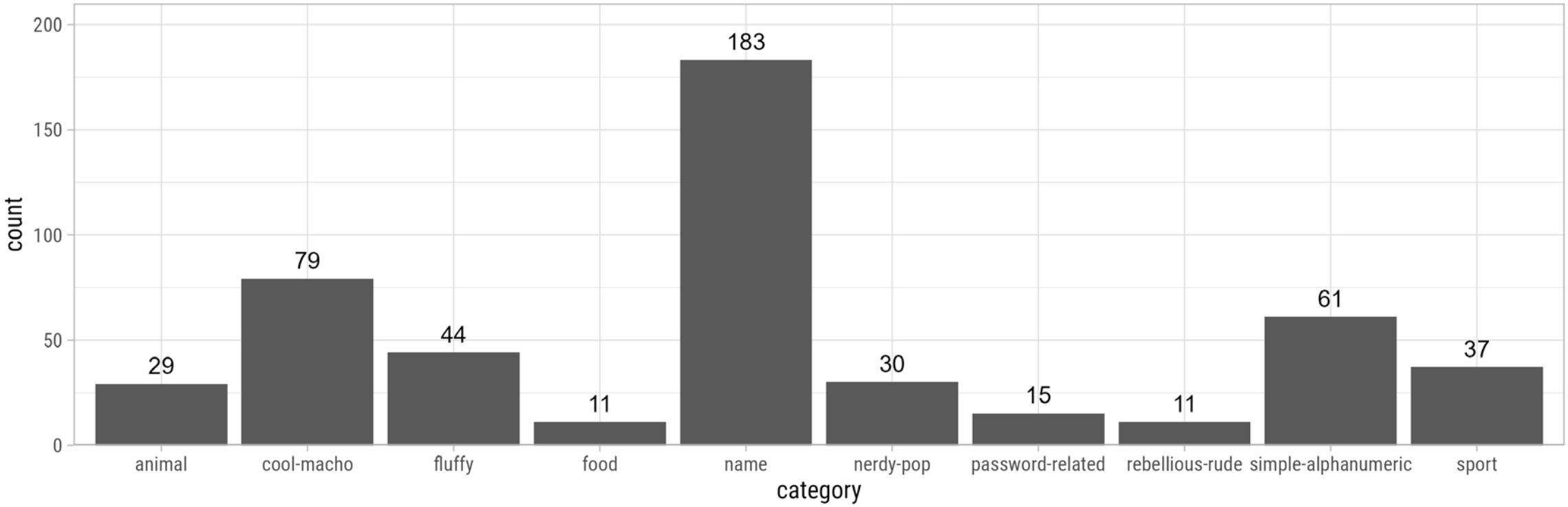
Exercise 1: Fix Limits

```
ggplot(pw_count, aes(fct_reorder(category, -n), n)) +
  geom_col() +
  geom_text(aes(label = n), vjust = -.6, size = 6) +
  scale_y_continuous(limits = c(NA, 210), expand = c(0, 0))
```



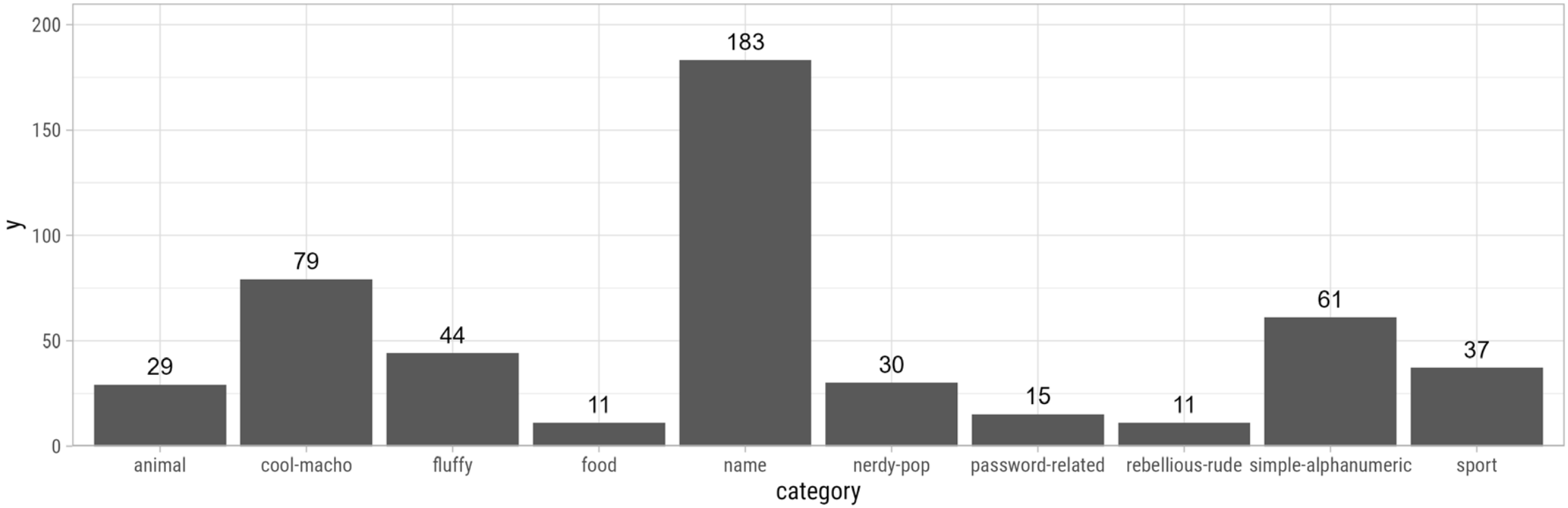
Exercise 1: Labels Number of Passwords per Category

```
ggplot(passwords, aes(category)) +
  stat_count() +
  stat_count(geom = "text", aes(label = ..count..), vjust = -.6, size = 6) +
  scale_y_continuous(limits = c(NA, 210), expand = c(0, 0))
```



Exercise 1: Labels Number of Passwords per Category

```
ggplot(passwords, aes(category, 1)) +
  stat_summary(geom = "bar", fun = length) +
  stat_summary(geom = "text", aes(label = ..y..), fun = length, vjust = -.6, size = 6) +
  scale_y_continuous(limits = c(NA, 210), expand = c(0, 0))
```



Collection of some chart types provided by ggplot2 extension packages.

Illustration by Allison Horst

Spatial Maps

Spatial Maps

Simple Features (Access)

`Simple features` or `simple features access` refers to a formal standard (ISO 19125-1:2004) that describes how objects in the real world can be represented in computers, with emphasis on the *spatial geometry*.

`{sf}` is an R package that provides simple features access for R.

Spatial Maps

Simple Features (Access)

Simple features or **simple features access** refers to a formal standard (ISO 19125-1:2004) that describes how objects in the real world can be represented in computers, with emphasis on the *spatial geometry*.

{sf} is an R package that provides simple features access for R.

Natural Earth

Natural Earth is a public domain map data set of *physical and cultural features*, available at 1:10m, 1:50m, and 1:110 million scales.

{rnaturalearth} is an R package to hold and facilitate interaction with Natural Earth map data.

POLL: Have you used R to visualize spatial data? (MC)

- No, I never deal with spatial data.
- No, I use other tools for spatial data.
- Yes, `ggplot2::geom_polygon()`.
- Yes, `ggplot2::geom_raster()`.
- Yes, `ggplot2::geom_sf()`.
- Yes, the `{tmap}` package.
- Yes, the `{ggmap}` package.
- Yes, some other R packages that are not mentioned here.

Setup

```
#install.packages("sf")
library(sf)

#install.packages("rnaturalearth")
library(rnaturalearth)
```

Get the Data

The {rnaturalearth} allows downloading the Natural Earth data sets in simple feature format.

```
sf_world <- ne_countries(returnclass = "sf")
sf_airports <- ne_download(scale = "large", category = "cultural", type = "airports", returnclass = "sf")
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\DataVizard\AppData\Local\Temp\RtmpoxZTGc", layer: "ne_10m_airports"
## with 891 features
## It has 35 fields
## Integer64 fields read as strings:  ne_id

tibble::glimpse(sf_world)
## Rows: 177
## Columns: 64
## $ scalerank  <int> 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
## $ featurecla <chr> "Admin-0 country", "Admin-0 country", "Admin-0 country", "A~
## $ labelrank  <dbl> 3, 3, 6, 4, 2, 6, 4, 6, 2, 4, 5, 6, 2, 5, 3, 3, 4, 4, 5, 4,~
## $ sovereignt <chr> "Afghanistan", "Angola", "Albania", "United Arab Emirates",~
## $ sov_a3     <chr> "AFG", "AGO", "ALB", "ARE", "ARG", "ARM", "ATA", "FR1", "AU~
## $ adm0_dif   <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ level      <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,~
## $ type       <chr> "Sovereign country", "Sovereign country", "Sovereign countr~
## $ admin      <chr> "Afghanistan", "Angola", "Albania", "United Arab Emirates",~
## $ adm0_a3    <chr> "AFG", "AGO", "ALB", "ARE", "ARG", "ARM", "ATA", "ATF", "AU~
```

Plot the Map

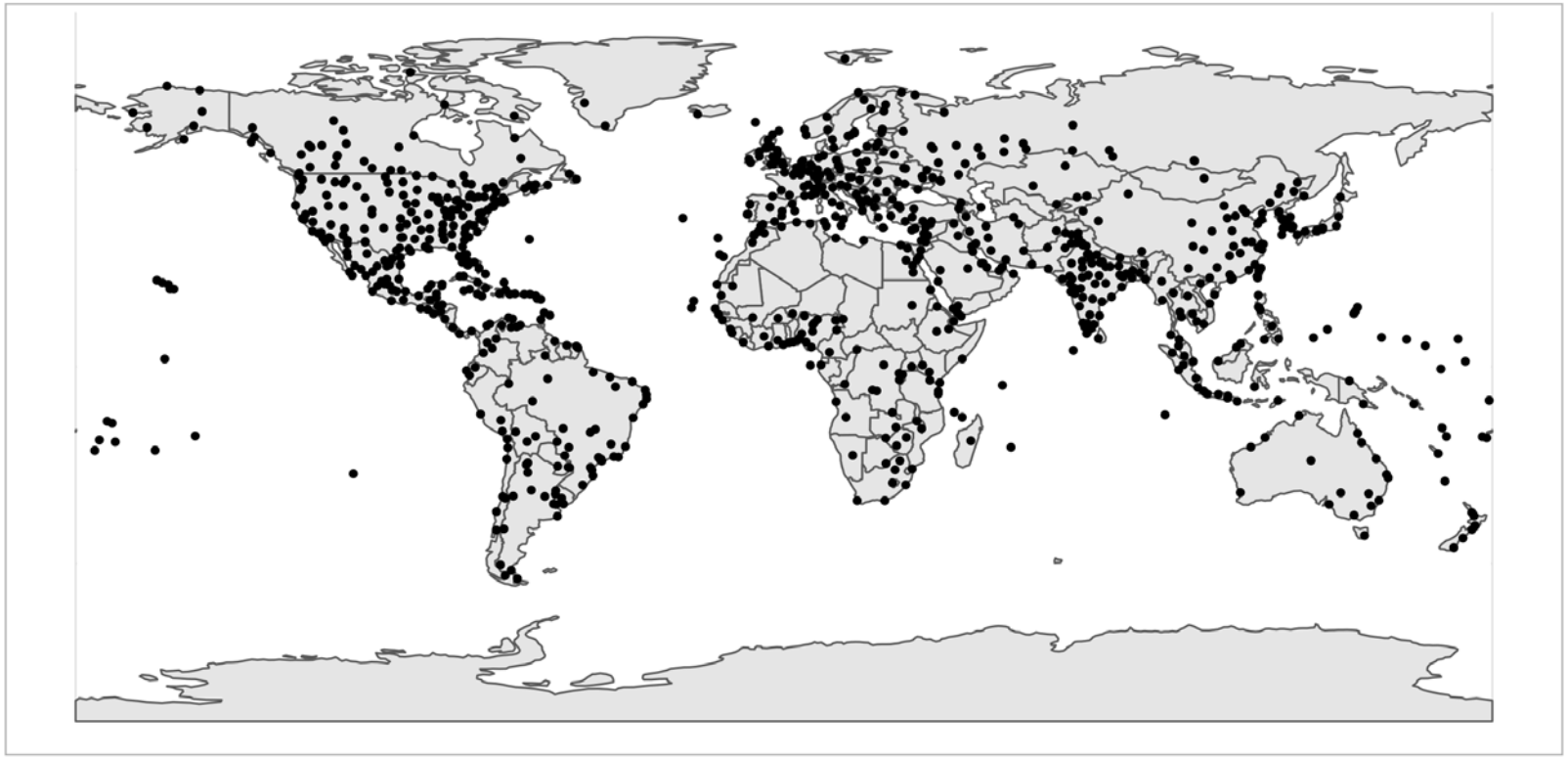
{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

```
ggplot(sf_world) +
  geom_sf()
```

Plot the Map

`geom_sf()` works with all types of vector data and returns geometries depending on the simple features:

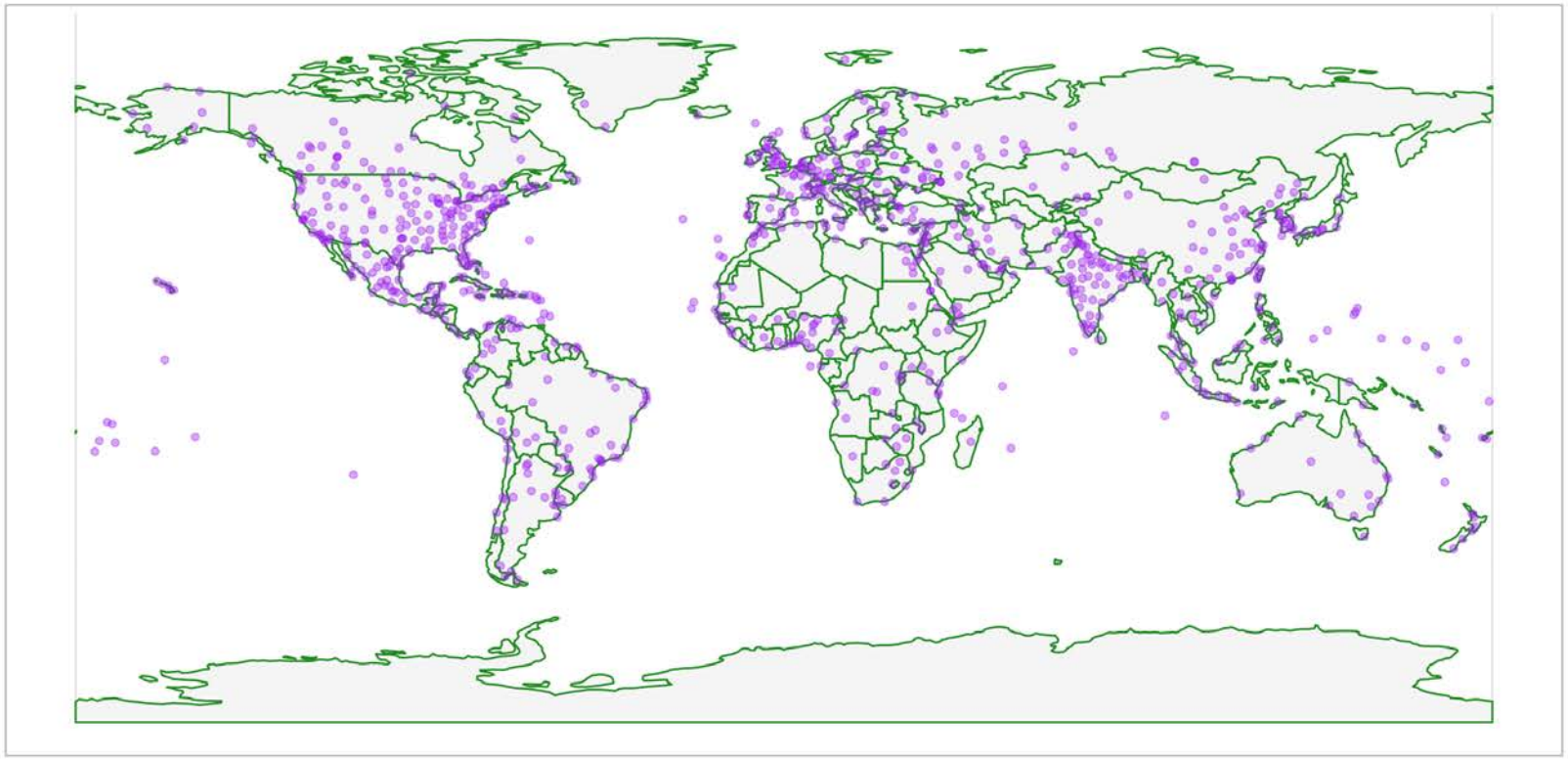
```
ggplot(sf_world) +
  geom_sf() +
  geom_sf(data = sf_airports)
```



Plot the Map

`geom_sf()` works with all types of vector data and returns geometries depending on the simple features:

```
ggplot(sf_world) +
  geom_sf(color = "forestgreen", alpha = .4) +
  geom_sf(data = sf_airports, color = "purple", alpha = .4)
```



Generate Own Simple Features Objects

`geom_sf()` works with all types of vector data and returns geometries depending on the simple features:

```
sf_bln <- sf::st_sfc(sf::st_point(c(13.4050, 52.5200)), crs = sf::st_crs(sf_world))

sf_bln
## Geometry set for 1 feature
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 13.405 ymin: 52.52 xmax: 13.405 ymax: 52.52
## CRS:           +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

Plot the Map

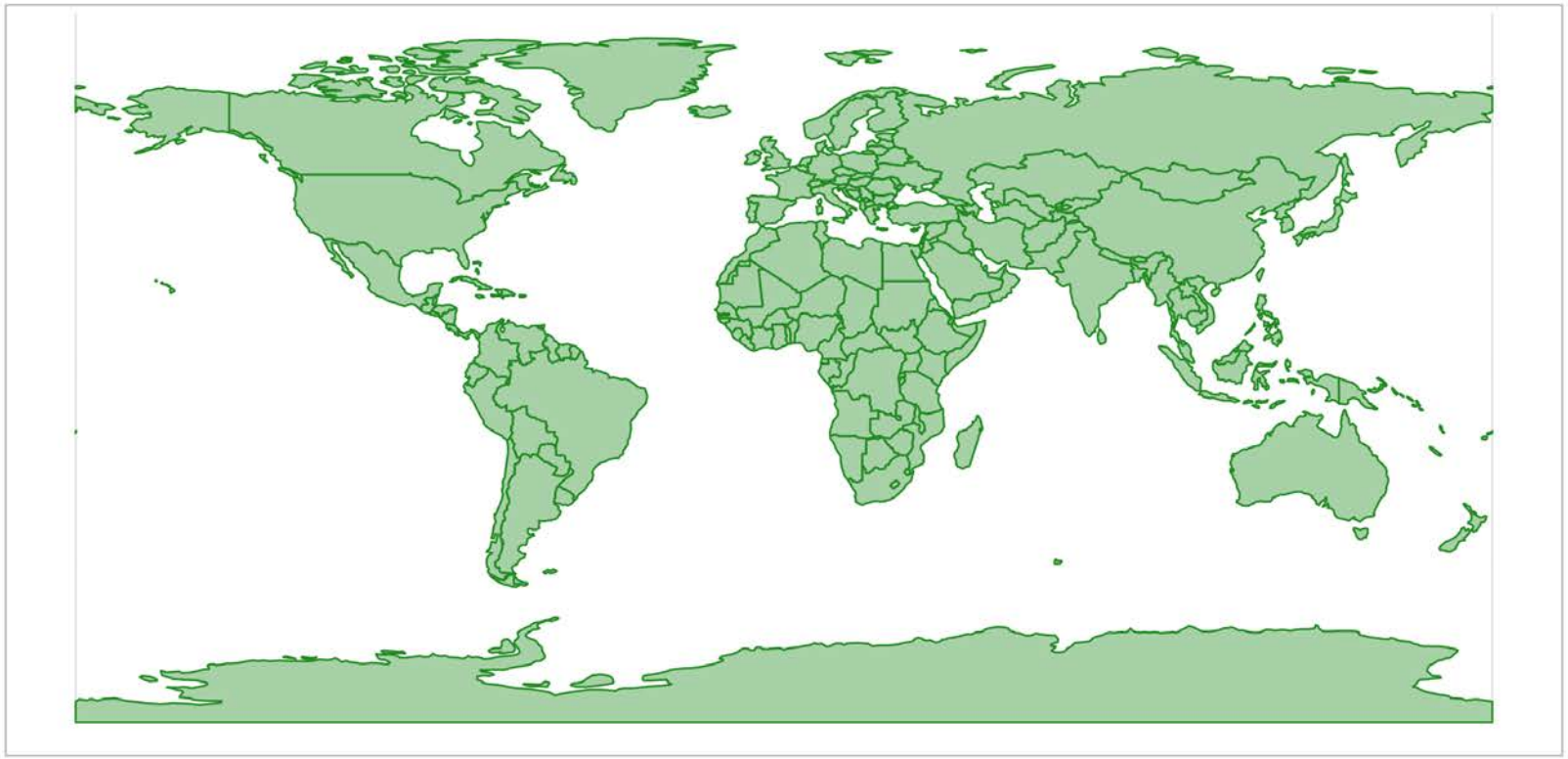
`geom_sf()` works with all types of vector data and returns geometries depending on the simple features:

```
ggplot(sf_world) +
  geom_sf(color = NA) +
  geom_sf(data = sf_bln, shape = 21, color = "firebrick", fill = NA, size = 5, stroke = 2)
```

Plot the Map

{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

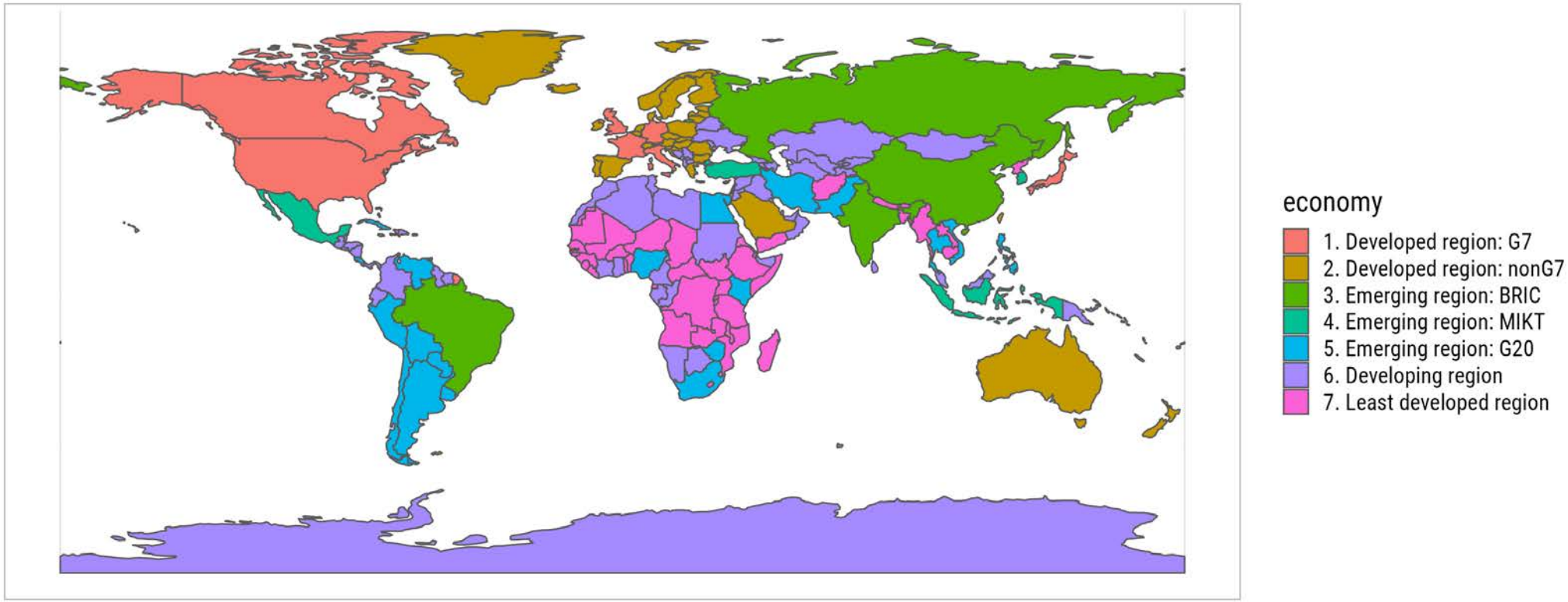
```
ggplot(sf_world) +
  geom_sf(color = "forestgreen", fill = "forestgreen", alpha = .4)
```



Plot the Map

{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

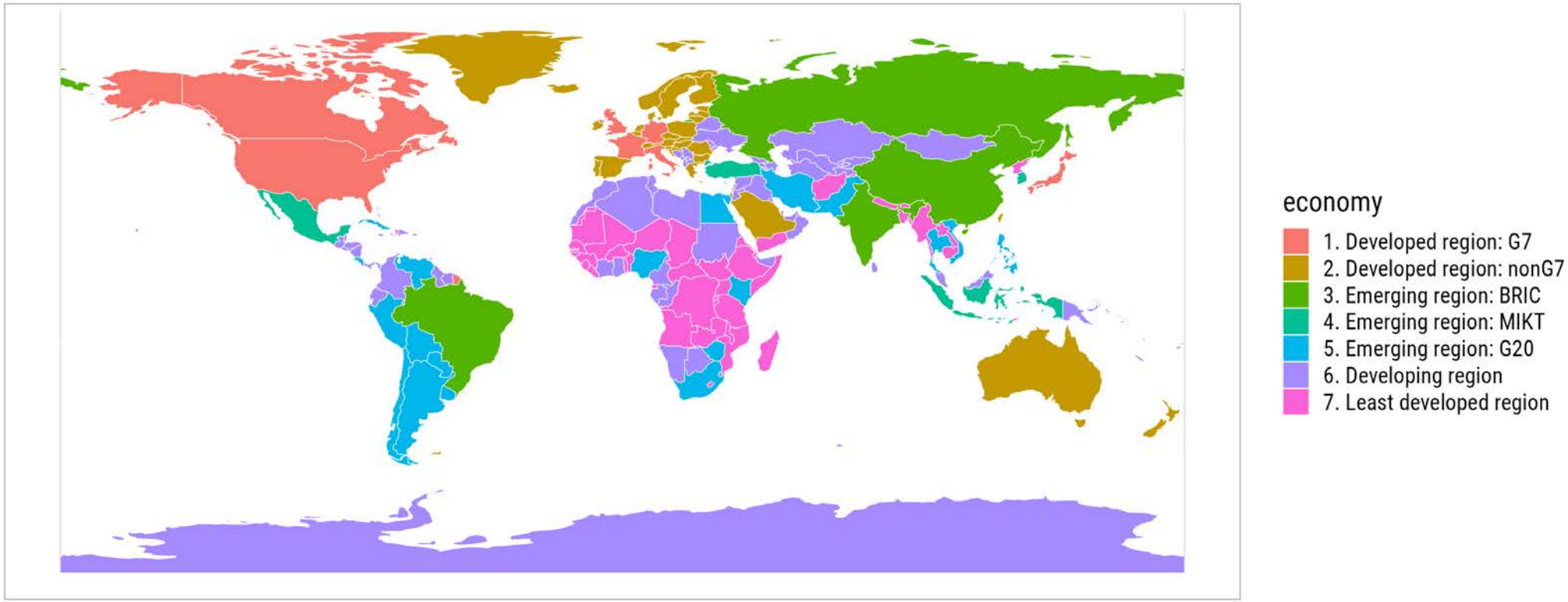
```
ggplot(sf_world) +
  geom_sf(aes(fill = economy))
```



Plot the Map

{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

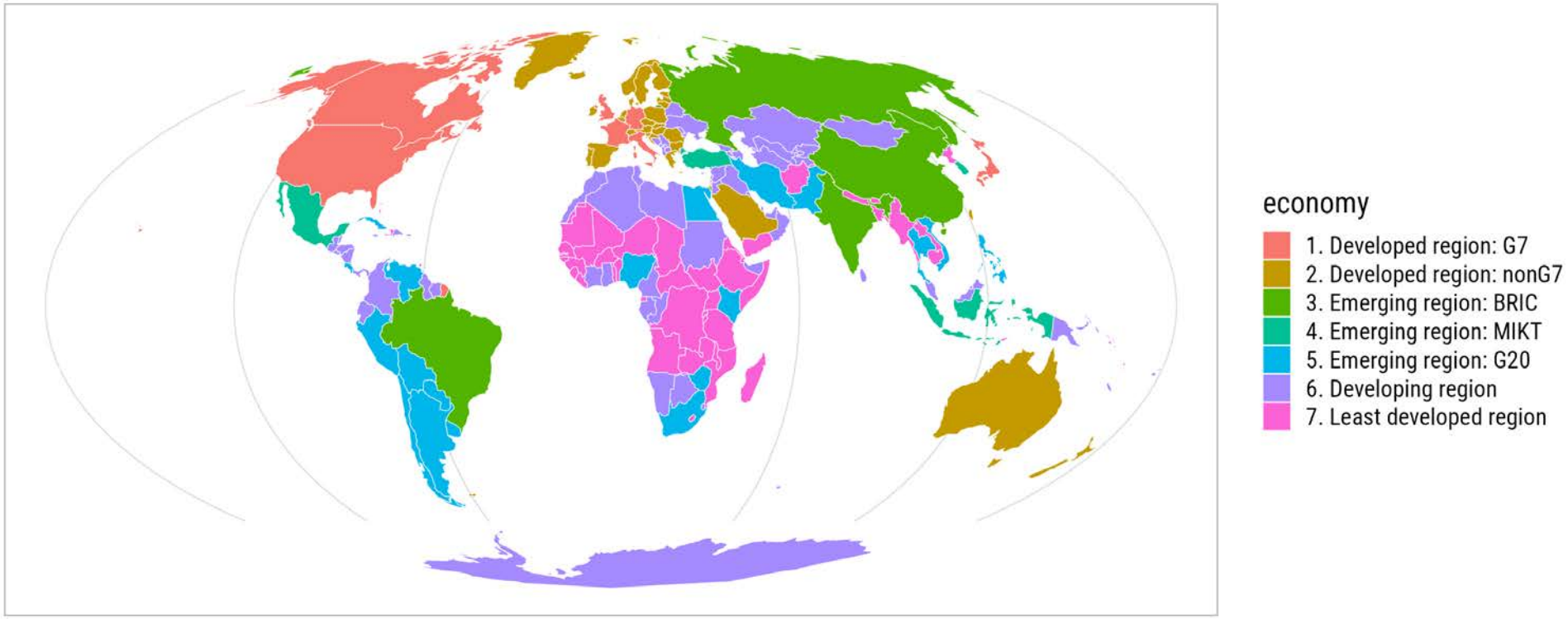
```
ggplot(sf_world) +
  geom_sf(aes(fill = economy), color = "white", size = .2)
```



Change the Projection

{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

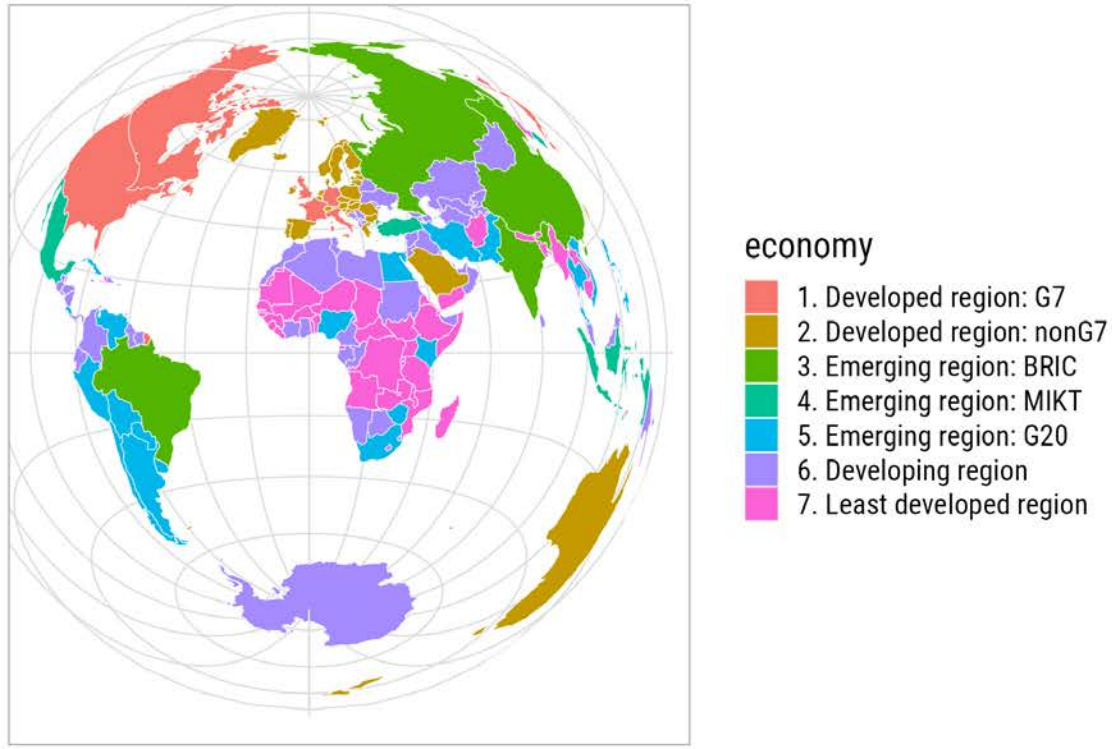
```
ggplot(sf_world) +
  geom_sf(aes(fill = economy), color = "white", size = .2) +
  coord_sf(crs = "+proj=moll")
```



Change the Projection

{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

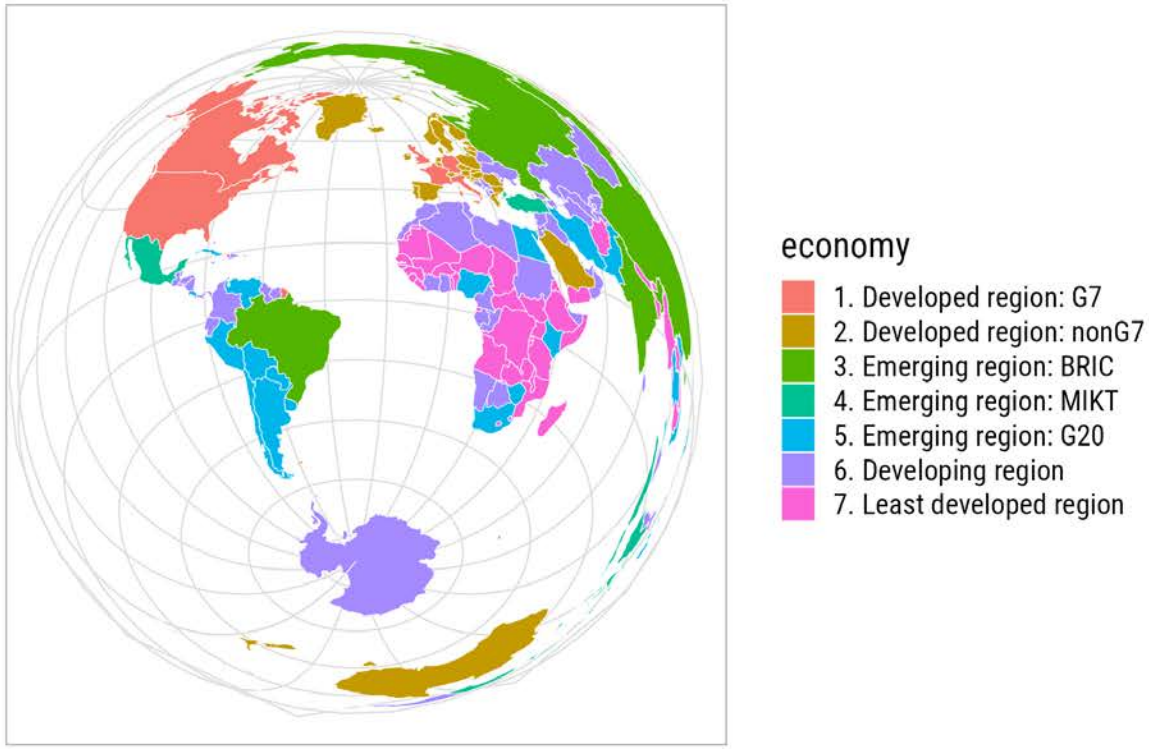
```
ggplot(sf_world) +
  geom_sf(aes(fill = economy), color = "white", size = .2) +
  coord_sf(crs = "+proj=laea +x_0=0 +y_0=0 +lon_0=0 +lat_0=0")
```



Change the Projection

{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

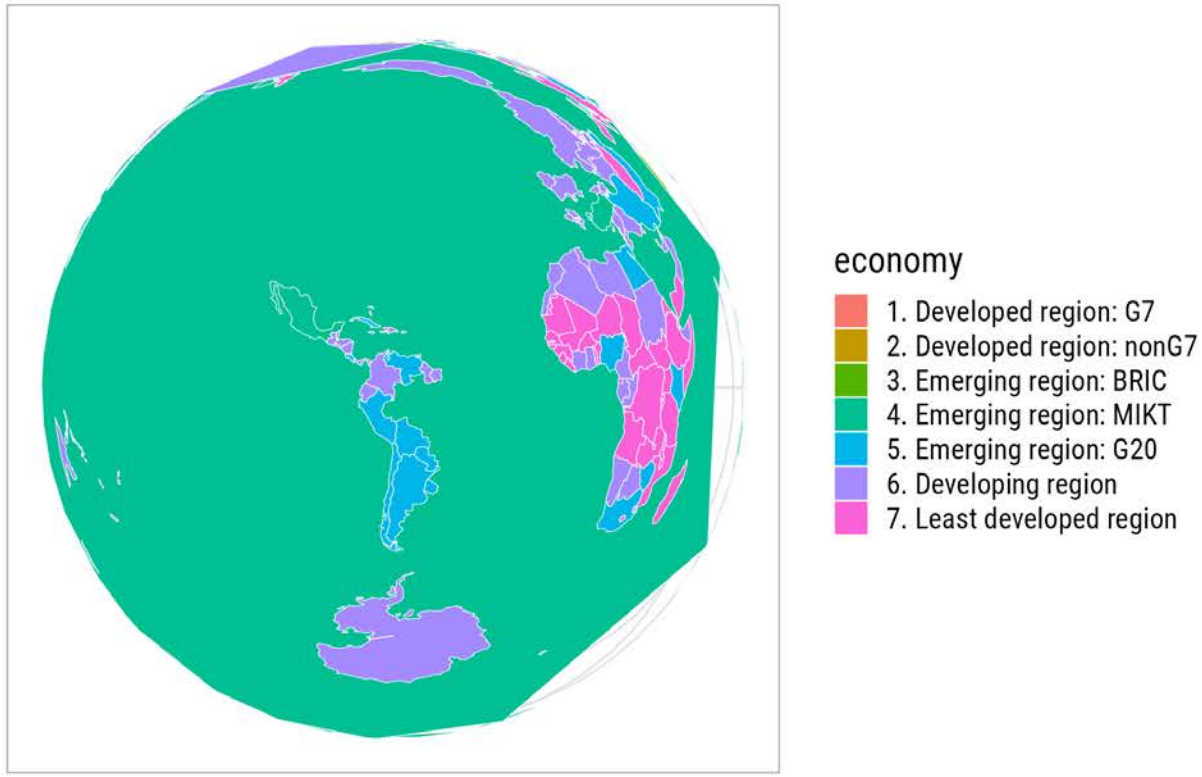
```
ggplot(sf_world) +
  geom_sf(aes(fill = economy), color = "white", size = .2) +
  coord_sf(crs = "+proj=laea +x_0=0 +y_0=0 +lon_0=-30 +lat_0=-25")
```



Change the Projection

{ggplot2} comes with a set of geom, stat, and coord are used to visualize sf objects.

```
ggplot(sf_world) +
  geom_sf(aes(fill = economy), color = "white", size = .2) +
  coord_sf(crs = "+proj=laea +x_0=0 +y_0=0 +lon_0=-70 +lat_0=0")
```



Exercise 2:

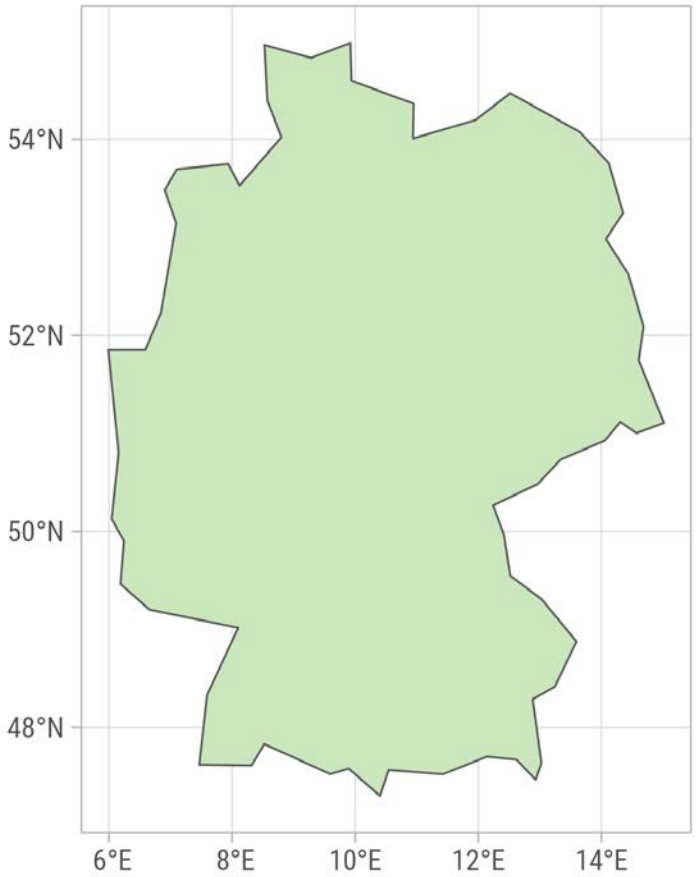
- Download data set(s) for your home country and plot it/them.
- Change the projection with `coord_sf` and observe how the spatial mapping changes.

Exercise 2: Load the Data

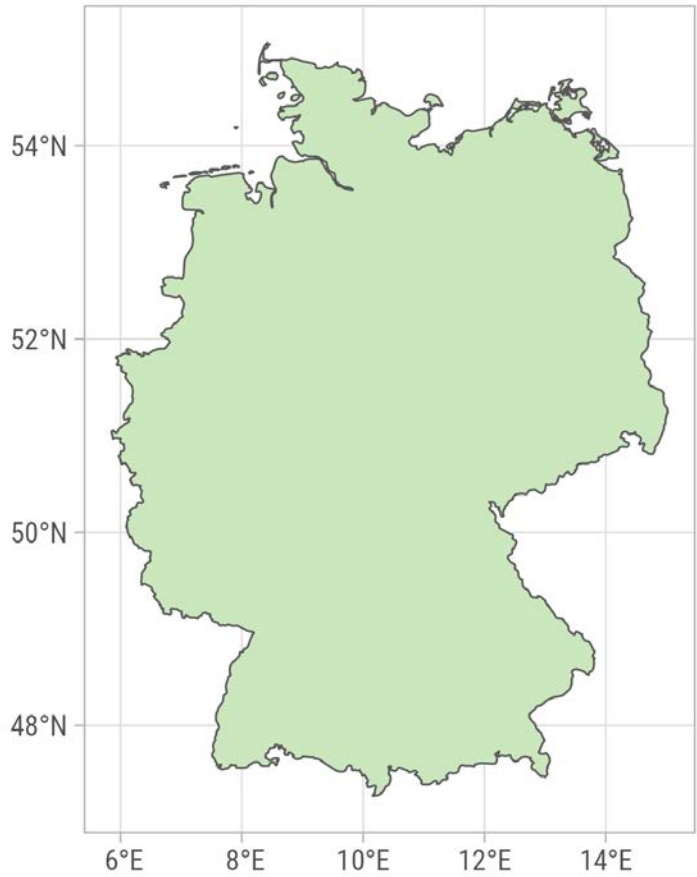
```
sf_germany <- rnaturalearth::ne_countries(country = "Germany", returnclass = "sf")
sf_germany_hd <- rnaturalearth::ne_countries(country = "Germany", scale = 10, returnclass = "sf")

class(sf_germany)
## [1] "sf"         "data.frame"
names(sf_germany)
##  [1] "scalerank"  "featurecla" "labelrank"  "sovereignt" "sov_a3"
##  [6] "adm0_dif"   "level"      "type"       "admin"      "adm0_a3"
## [11] "geou_dif"   "geounit"    "gu_a3"      "su_dif"     "subunit"
## [16] "su_a3"      "brk_diff"   "name"       "name_long"  "brk_a3"
## [21] "brk_name"   "brk_group"  "abbrev"     "postal"     "formal_en"
## [26] "formal_fr"  "note_adm0"  "note_brk"   "name_sort"  "name_alt"
## [31] "mapcolor7"  "mapcolor8"  "mapcolor9"  "mapcolor13" "pop_est"
## [36] "gdp_md_est" "pop_year"   "lastcensus" "gdp_year"   "economy"
## [41] "income_grp" "wikipedia"  "fips_10"    "iso_a2"     "iso_a3"
## [46] "iso_n3"     "un_a3"      "wb_a2"      "wb_a3"      "woe_id"
## [51] "adm0_a3_is" "adm0_a3_us" "adm0_a3_un" "adm0_a3_wb" "continent"
## [56] "region_un"  "subregion"  "region_wb"  "name_len"   "long_len"
## [61] "abbrev_len" "tiny"       "homepart"   "geometry"
```

Exercise 2: Plot the Data

```
ggplot(sf_germany) +
  geom_sf(fill = "#CBE7BE")
```



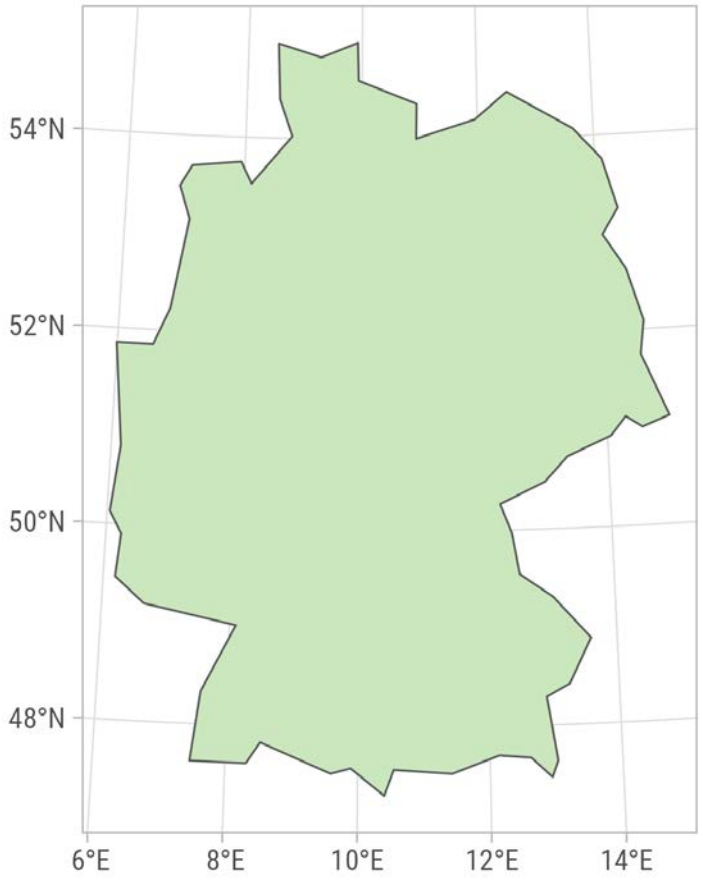

```
ggplot(sf_germany_hd) +
  geom_sf(fill = "#CBE7BE")
```



Exercise 2: Change Projection

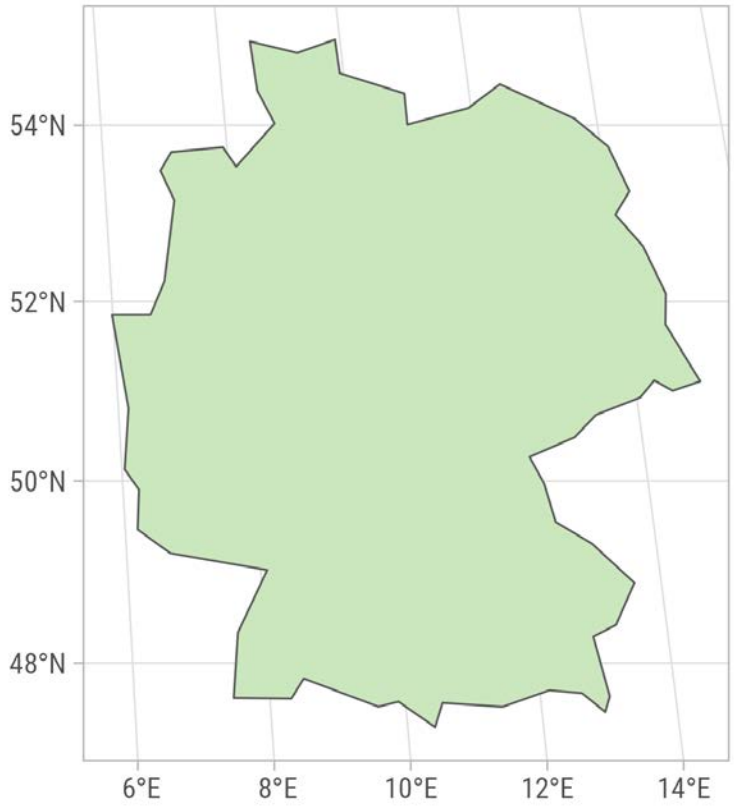
```
ggplot(sf_germany) +
  geom_sf(fill = "#CBE7BE") +
  coord_sf(crs = "+proj=lcc +lat_1=48.66 +lat_2=53.6
```

```
ggplot(sf_germany) +
  geom_sf(fill = "#CBE7BE") +
  coord_sf(crs = st_crs(5243))
```

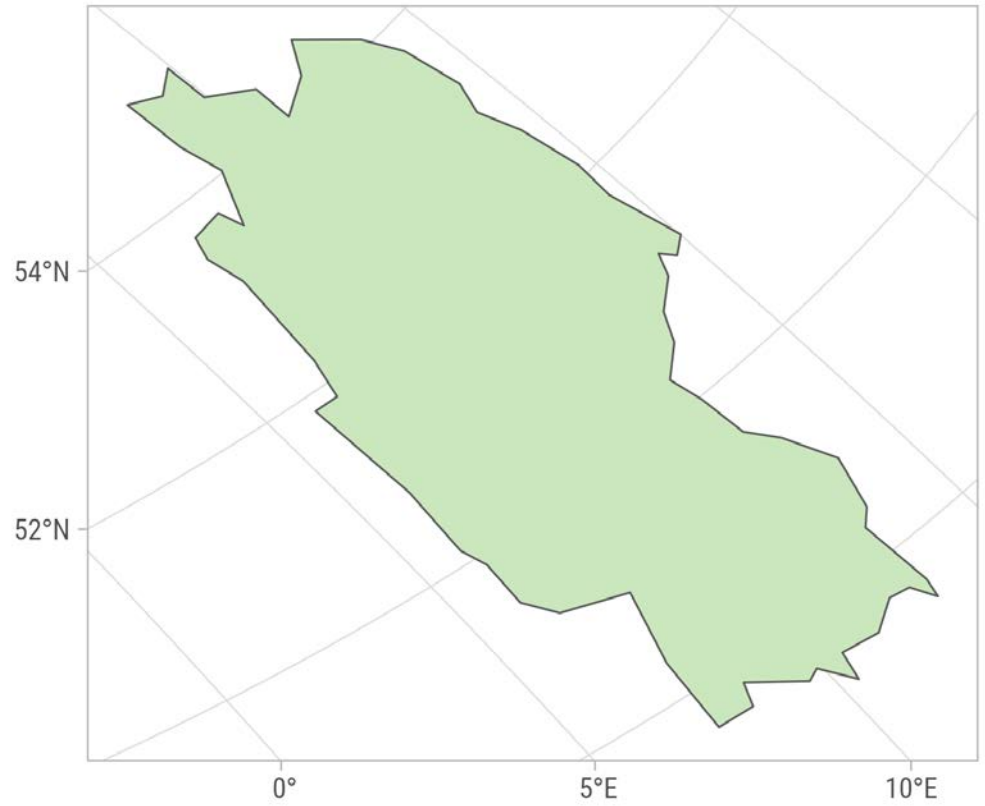


Exercise 2: Change Projection

```
ggplot(sf_germany) +
  geom_sf(fill = "#CBE7BE") +
  coord_sf(crs = "+proj=moll")
```

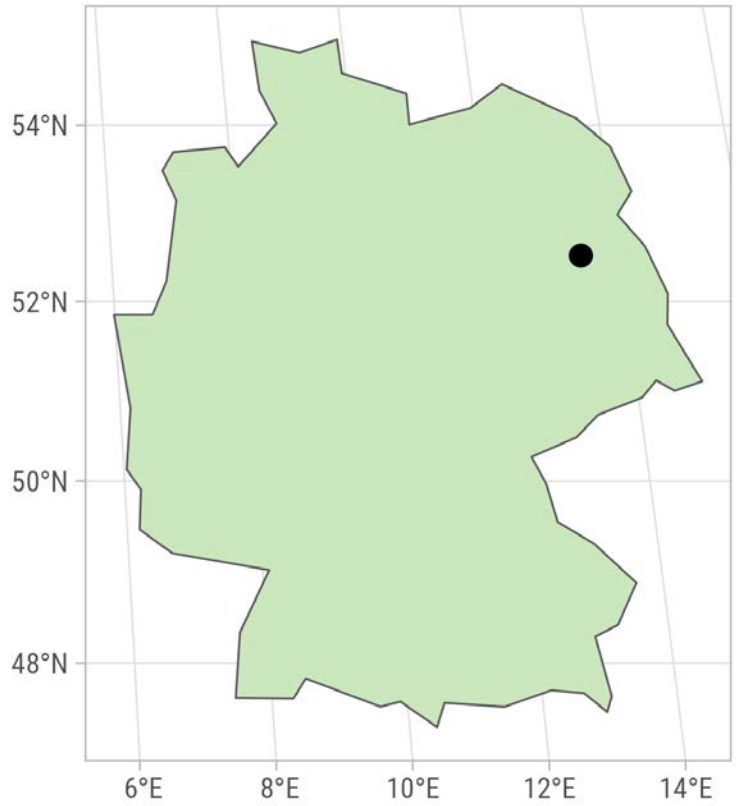



```
ggplot(sf_germany) +
  geom_sf(fill = "#CBE7BE") +
  coord_sf(crs = "+proj=laea +lon_0=-70 +lat_0=0")
```

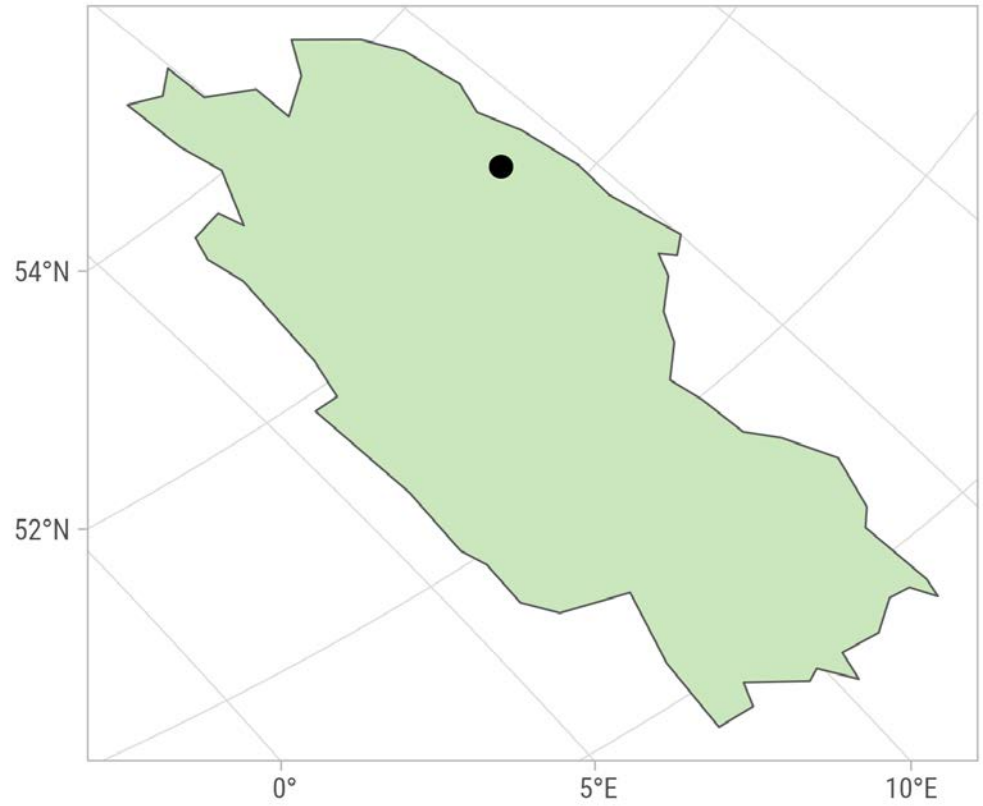


Exercise 2: Add Location

```
ggplot(sf_germany) +
  geom_sf(fill = "#CBE7BE") +
  geom_sf(data = sf_bln, size = 5) +
  coord_sf(crs = "+proj=moll")
```




```
ggplot(sf_germany) +
  geom_sf(fill = "#CBE7BE") +
  geom_sf(data = sf_bln, size = 5) +
  coord_sf(crs = "+proj=laea +lon_0=-70 +lat_0=0")
```



Resources

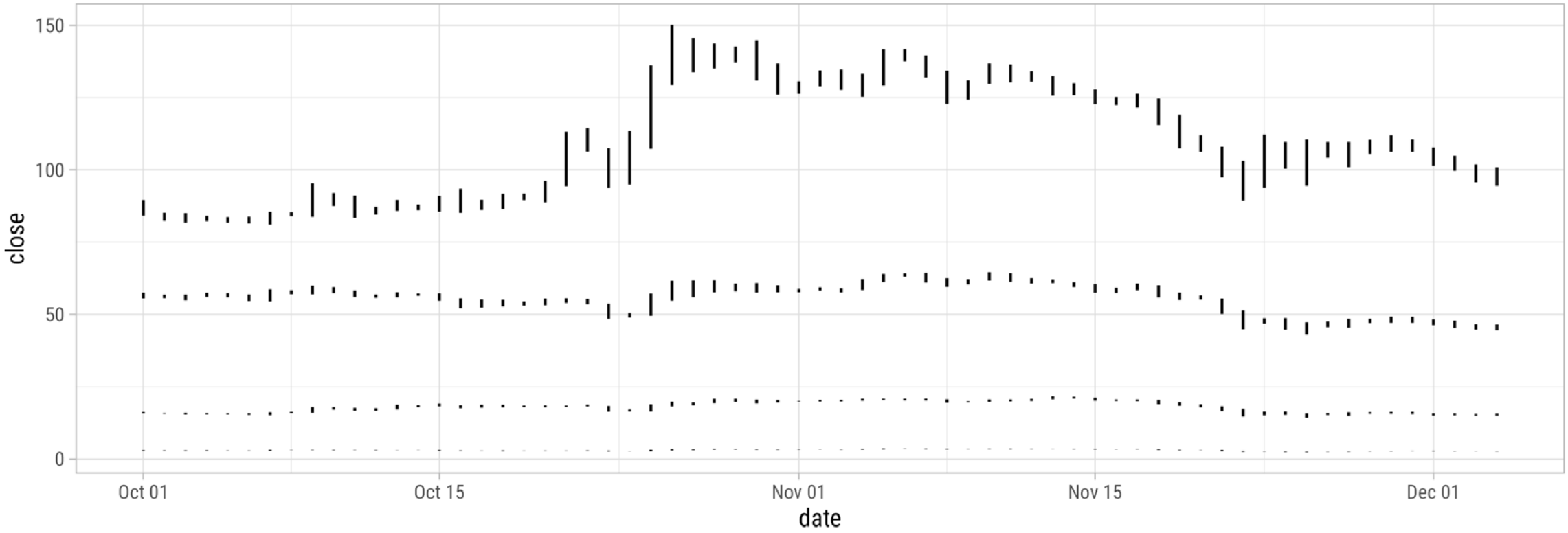
- Chapters on individual geom's, collective geom's, statistical summaries, and maps of the “ggplot2” book by Hadley Wickham et al.
- Overview of layers contained in the {ggplot2} package
- R Graph Gallery that provides hundreds of charts with their reproducible code
- List of {ggplot2} extension packages
- “Geocomputation with R”, free-access book by Robin Lovelace et al.
- {sf} package page
- StackOverflow discussion and code to plot a true globe
- “A {ggplot2} Tutorial for Beautiful Plotting in R”, my extensive "how to"-tutorial

APPENDIX

Create Any Chart Type: Candlestick Charts

There is a package for that — but it's simple enough to do it manually!

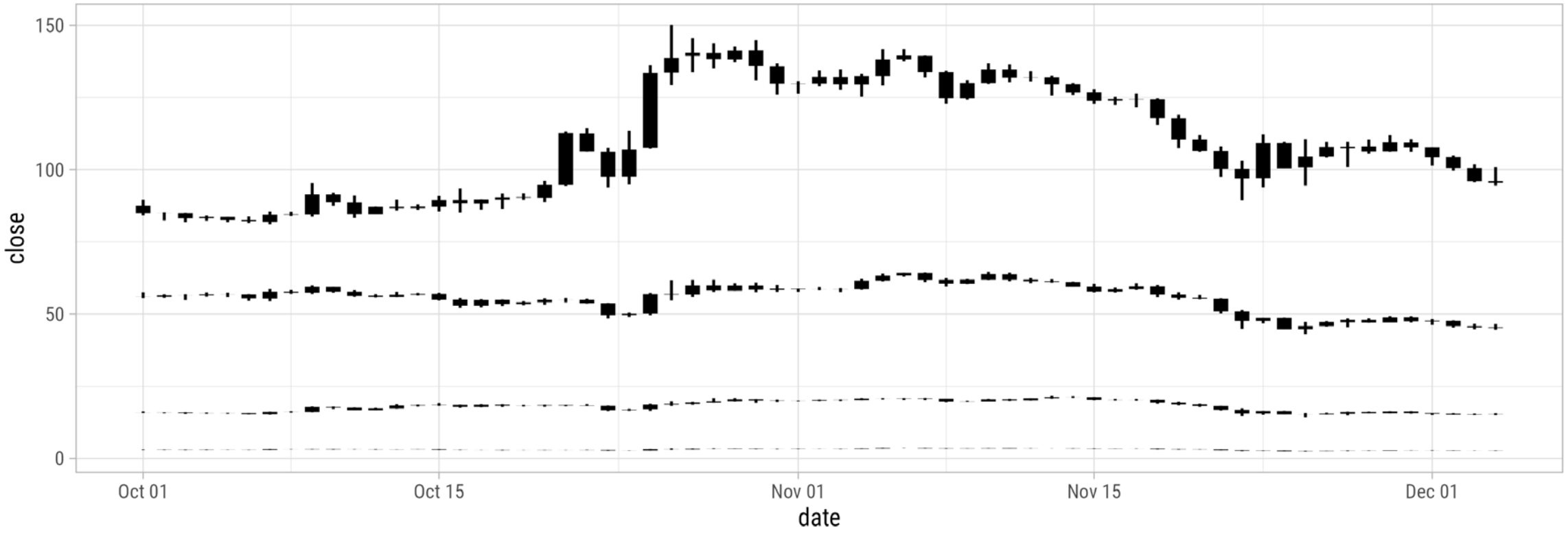
```
ggplot(filter(data, year == 2019, month > 9), aes(date, close)) +
  geom_linerange(aes(ymin = low, ymax = high), size = 1)
```



Create Any Chart Type: Candlestick Charts

There is a package for that — but it's simple enough to do it manually!

```
ggplot(filter(data, year == 2019, month > 9), aes(date, close)) +
  geom_linerange(aes(ymin = low, ymax = high), size = 1) +
  geom_linerange(aes(ymin = open, ymax = close), size = 5)
```



Create Any Chart Type: Candlestick Charts

There is a package for that — but it's simple enough to do it manually!

```
ggplot(filter(data, year == 2019, month > 9), aes(date, close)) +
  geom_linerange(aes(ymin = low, ymax = high, color = close > open), size = 1) +
  geom_linerange(aes(ymin = open, ymax = close, color = close > open), size = 5)
```

Create Any Chart Type: Candlestick Charts

There is a package for that — but it's simple enough to do it manually!

```
ggplot(filter(data, year == 2019, month > 9), aes(date, close)) +
  geom_linerange(aes(ymin = low, ymax = high, color = close > open), size = 1) +
  geom_linerange(aes(ymin = open, ymax = close, color = close > open), size = 5) +
  scale_x_date(date_labels = "%b %d, %Y")
```

Statistical Transformations: Draw Functions

`stat_function()` makes it easy to add a function to a plot, either continuous or discrete:

```
ggplot(tibble(x = c(-8, 8)), aes(x)) +
  stat_function(fun = dnorm) +
  stat_function(
    fun = dcauchy,
    color = "red",
    n = 75
  )
```