Hands–On Data Visualization with ggplot2

Segment 3: Polish Your Visualization

Dr. Cédric Scherer

Pearson Live Training Session for O'Reilly

September 3, 2021

Photo by Richard Strozynski

Illustration by Allison Horst

Aesthetics + Scales

Adjust Colors and Other `aes`thetics

One can use `scale_*()` to change properties of all the aesthetic dimensions mapped to the data.

Adjust Colors and Other `aes`thetics

One can use `scale_*()` to change properties of all the aesthetic dimensions mapped to the data.

Consequently, there are `scale_*()` functions for all aesthetics such as:

- **positions via `scale_x_*()` and `scale_y_*()`**
- **colors via `scale_color_*()` and `scale_fill_*()`**
- **sizes via `scale_size_*()` and `scale_radius_*()`**
- **shapes via `scale_shape_*()` and `scale_linetype_*()`**
- **transparency via `scale_alpha_*()`**

Adjust Colors and Other `aes`thetics

One can use `scale_*()` to change properties of all the aesthetic dimensions mapped to the data

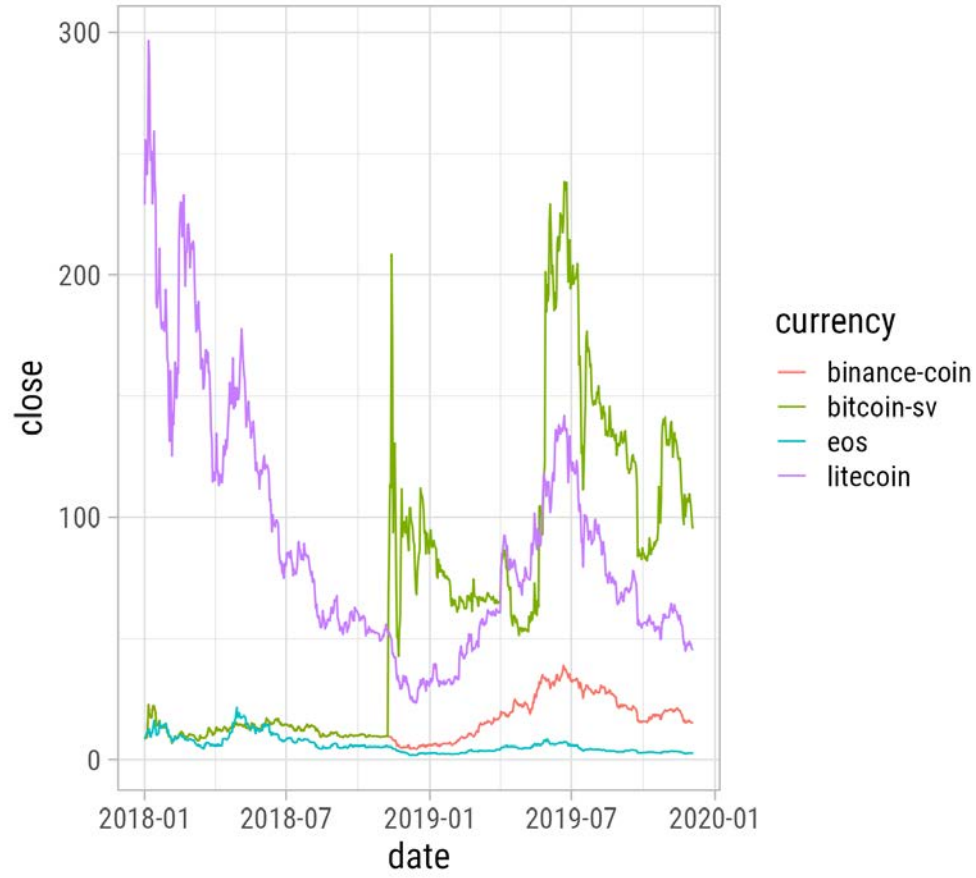
The extensions (`*`) can be filled by e.g.:

- `continuous()`,`discrete()`,`reverse()`,`log10()`,`sqrt()`,`date()` **for positions**
- `continuous()`,`discrete()`,`manual()`,`gradient()`,`gradient2()`,`brewer()` **for colors**
- `continuous()`,`discrete()`,`manual()`,`ordinal()`,`area()`,`date()` **for sizes**
- `continuous()`,`discrete()`,`manual()`,`ordinal()` **for shapes**
- `continuous()`,`discrete()`,`manual()`,`ordinal()`,`date()` **for transparency**

Scales: `scale_*()`

Scales are directly connected to aesthetics:

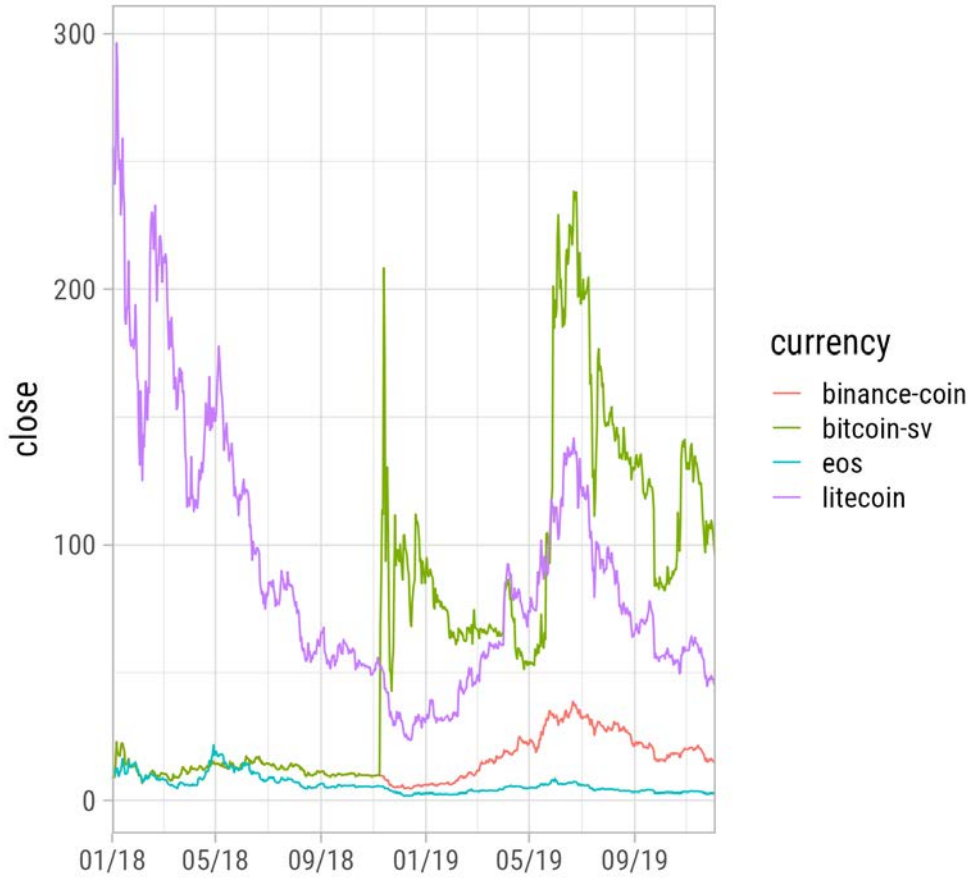
```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_x_date() +
  scale_y_continuous() +
  scale_color_discrete()
```



Scales: `scale_*()`

All scales come with some general and specific arguments to change the appearance:

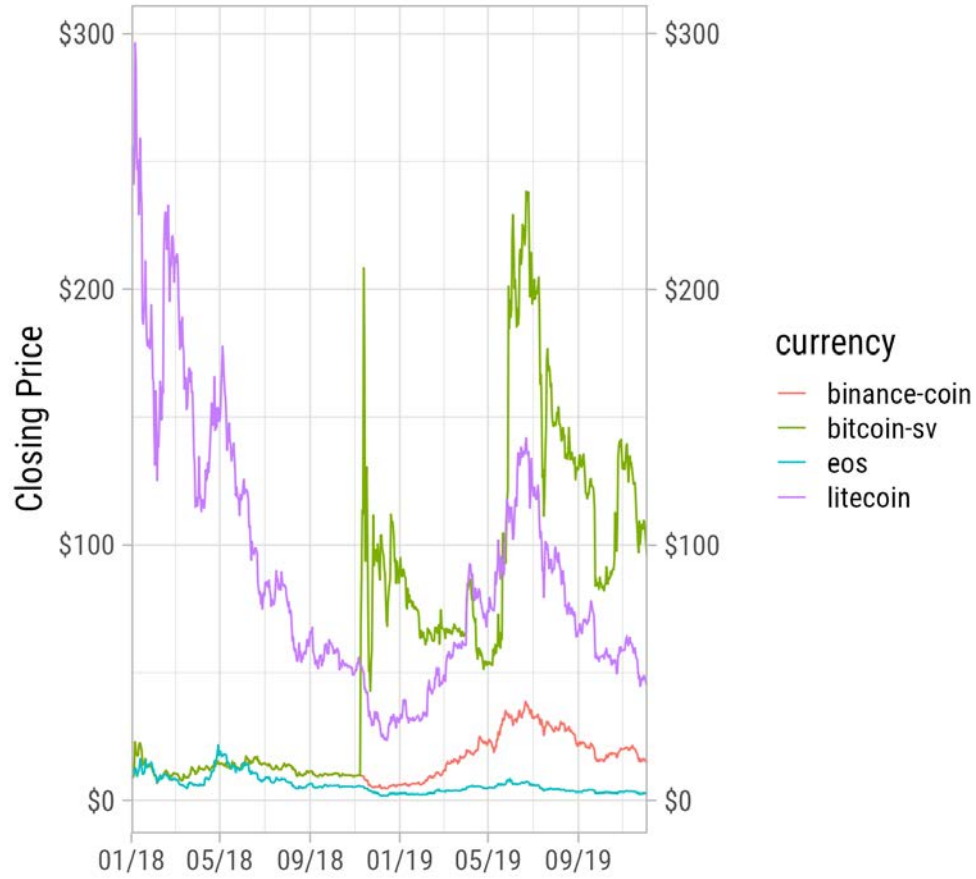
```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_x_date(
    expand = c(0, 0), ## general
    date_breaks = "4 months", ## date-only
    date_labels = "%m/%y", ## date only
    name = NULL ## general
  ) +
  scale_y_continuous() +
  scale_color_discrete()
```



Scales: `scale_*()`

All scales come with some general and specific arguments to change the appearance:

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_x_date(
    expand = c(0, 0), ## general
    date_breaks = "4 months", ## date-only
    date_labels = "%m/%y", ## date only
    name = NULL ## general
  ) +
  scale_y_continuous(
    labels = scales::dollar_format(), ## general
    sec.axis = dup_axis(name = NULL), ## axis only
    name = "Closing Price" ## general
  ) +
  scale_color_discrete()
```



Scales: `scale_*()`

All scales come with some general and specific arguments to change the appearance:

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_x_date(
    expand = c(0, 0), ## general
    date_breaks = "4 months", ## date-only
    date_labels = "%m/%y", ## date only
    name = NULL ## general
  ) +
  scale_y_continuous(
    labels = scales::dollar_format(), ## general
    sec.axis = dup_axis(name = NULL), ## axis only
    name = "Closing Price" ## general
  ) +
  scale_color_discrete(
    type = c("#F0B90B", "#4d4d4e",
             "#810080", "#00aeff"),  ## color only
    name = "Cryptocurrency:" ## general
  )
```

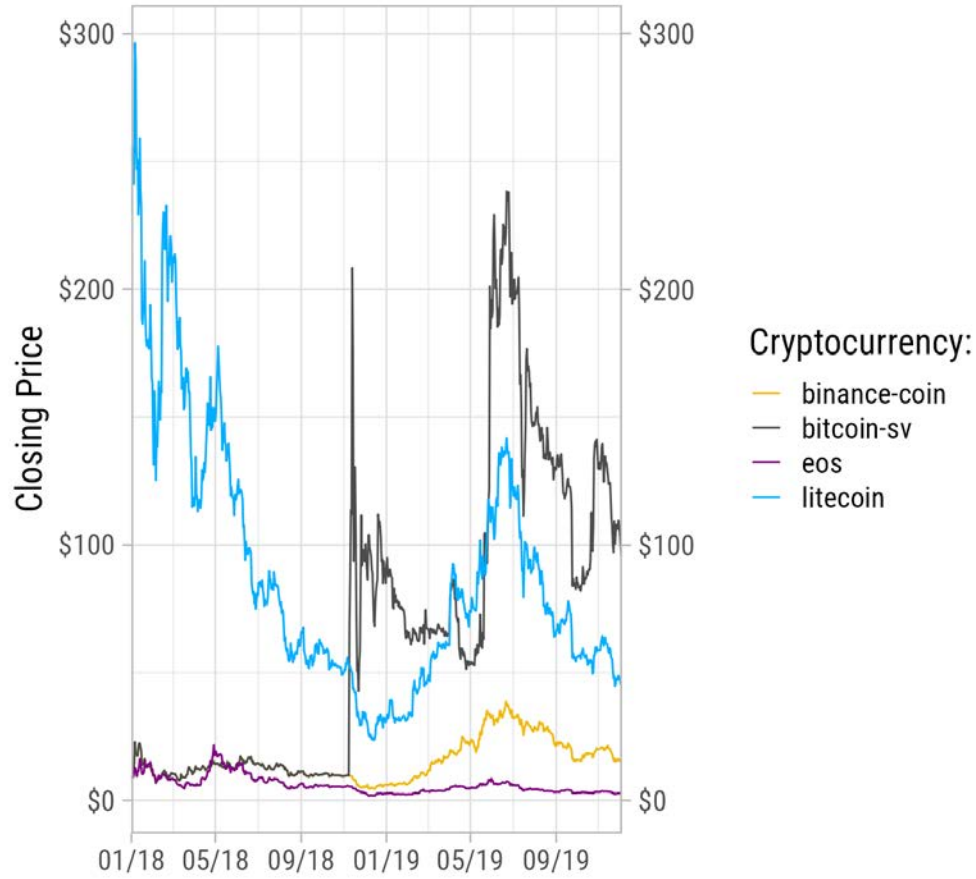


Illustration by Allison Horst

Categorical

default_uchicago

Sequential: Single-Hue

cyan_material

Sequential: Multi-Hue

Sunset

Diverging

Geyser

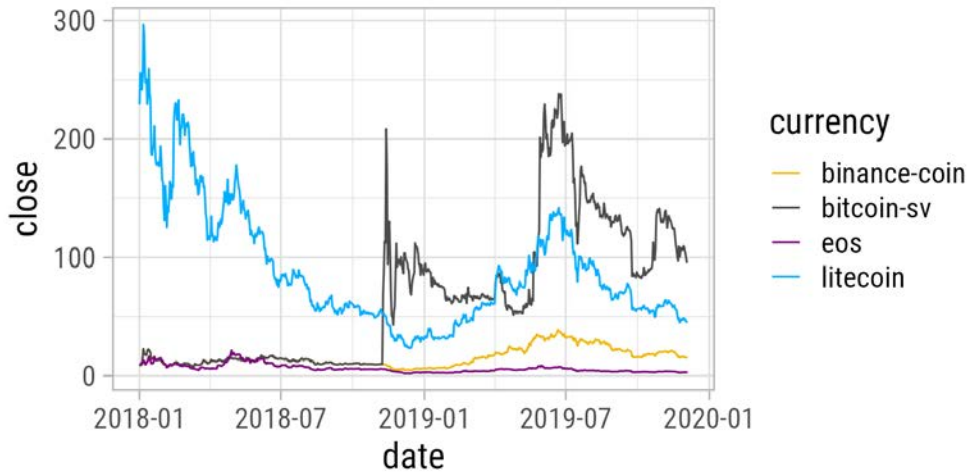
Cyclical

ocean.phase

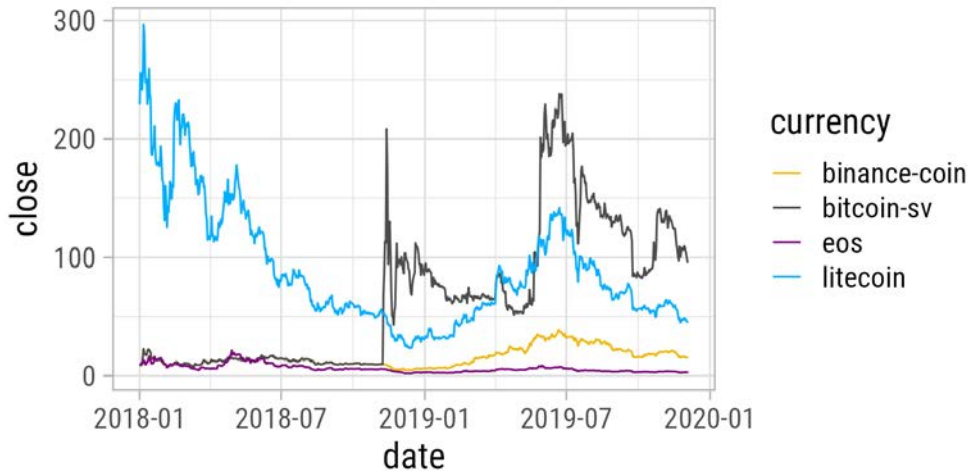
Scales: `scale_color_*()`

All colors and fills that are mapped to **categorical** variables can be manipulated with either `scale_color|fill_discrete()` or `scale_color|fill_manual()`.

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_color_discrete(
    type = c("#F0B90B", "#4d4d4e",
             "#810080", "#00aeff")
  )
```



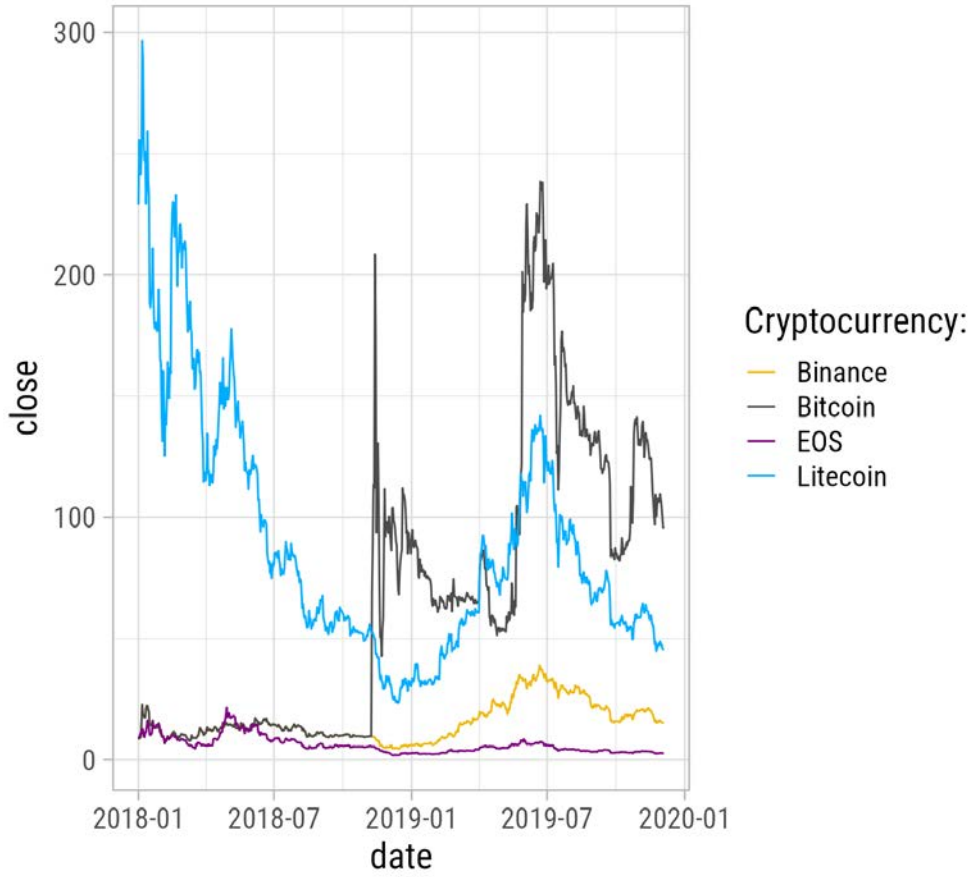

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_color_manual(
    values = c("#F0B90B", "#4d4d4e",
               "#810080", "#00aeff")
  )
```



Scales: scale_color_*()

Here, you can overwrite the labels of your legend items—no need to manipulate the data itself!

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_color_manual(
    values = c("#F0B90B", "#4d4d4e",
               "#810080", "#00aeff"),
    labels = c("Binance", "Bitcoin",
               "EOS", "Litecoin"),
    name = "Cryptocurrency:"
  )
```

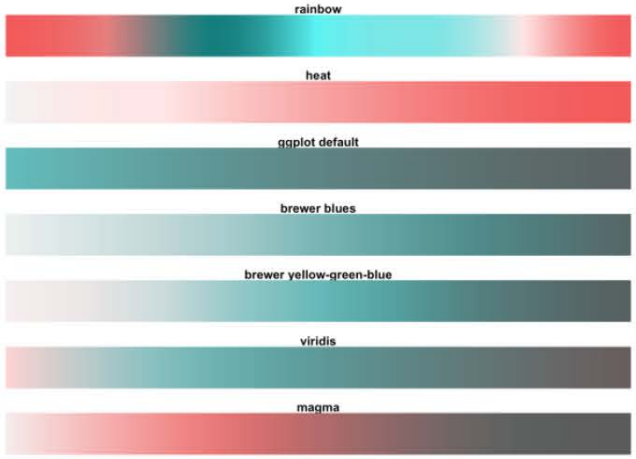


{viridis} package reference

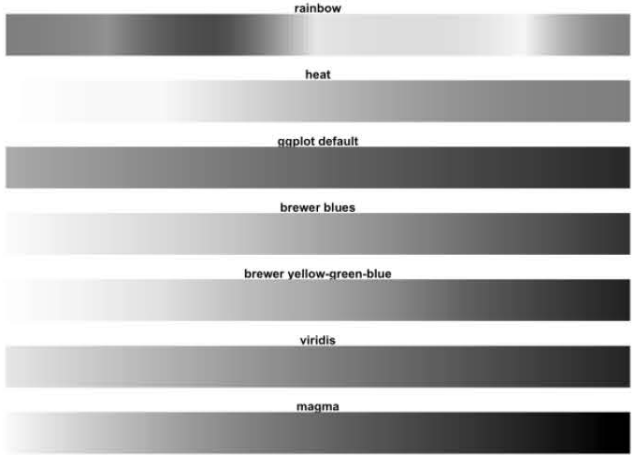
Deutranopia: present in 6% of males

Protanopia: present in 1% of males

Tritanopia: present in 0.008% of humans



Monochromacy: present in 0.001% of humans



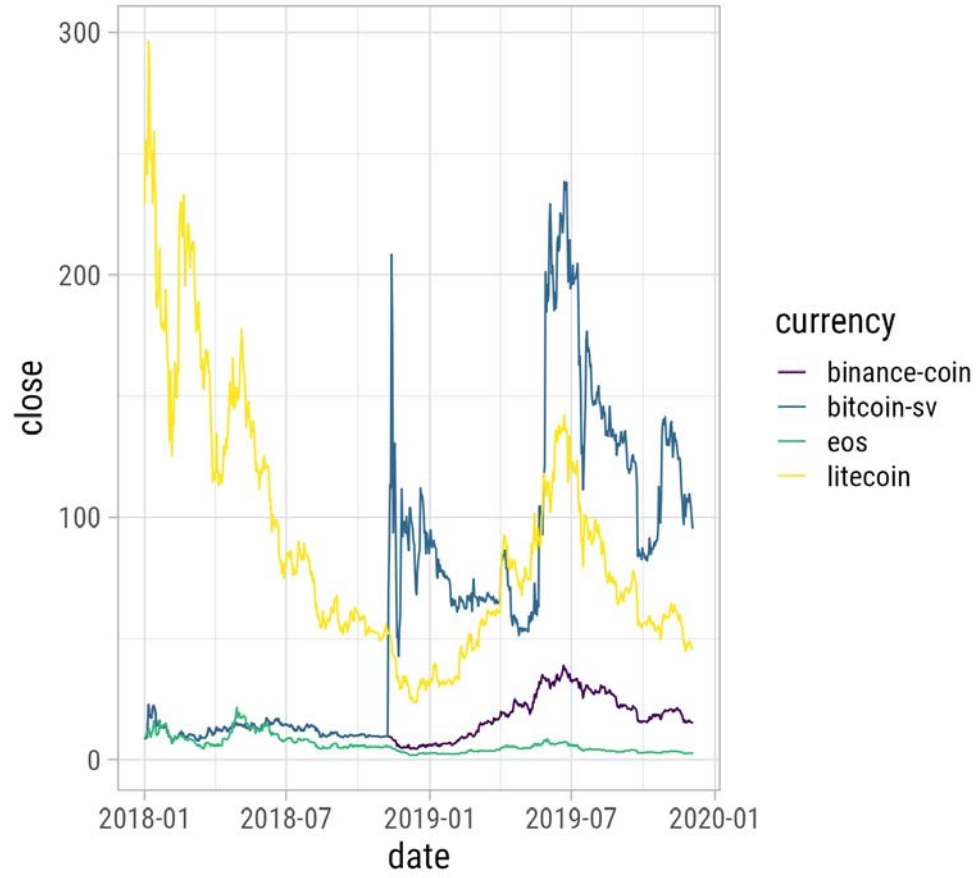
... and present in ~75% of university printers! ☺

Modified from the {viridis} package reference

Scales: `scale_color_viridis_*()`

There are a few and well crafted built-in palettes you can use as well (the defaults are pretty bad).

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

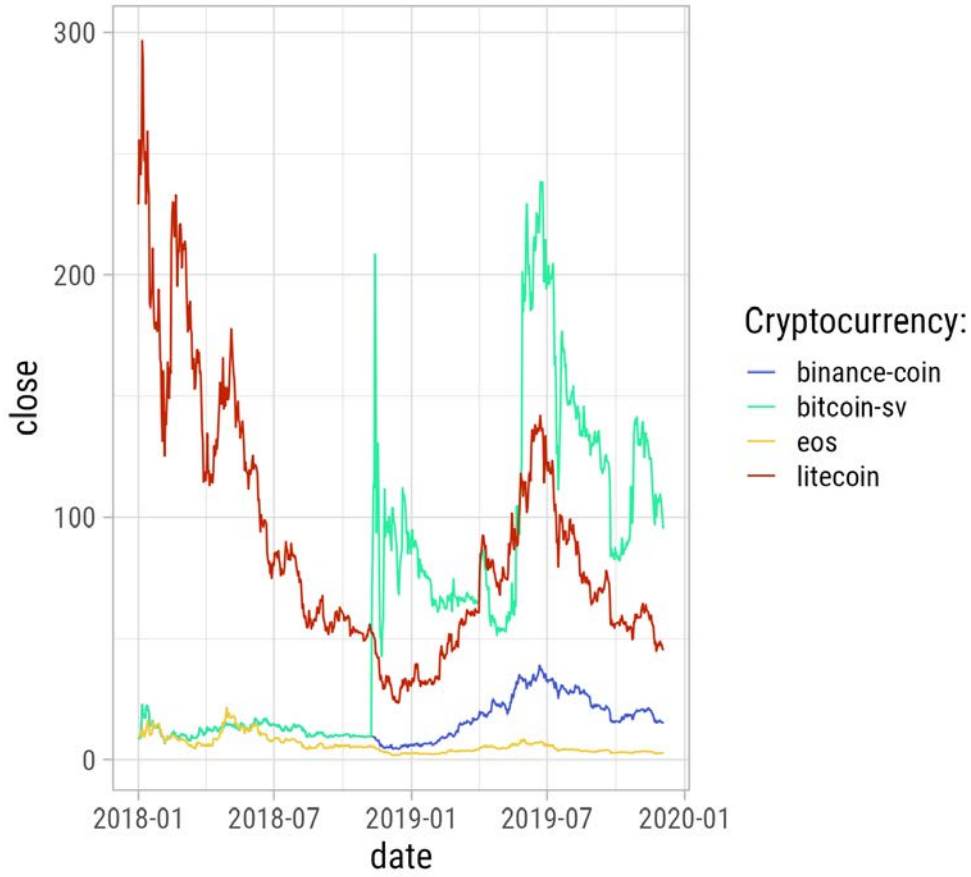


{viridis} package reference

Scales: `scale_color_viridis_*()`

There are a few and well crafted built-in palettes you can use as well (the defaults are pretty bad).

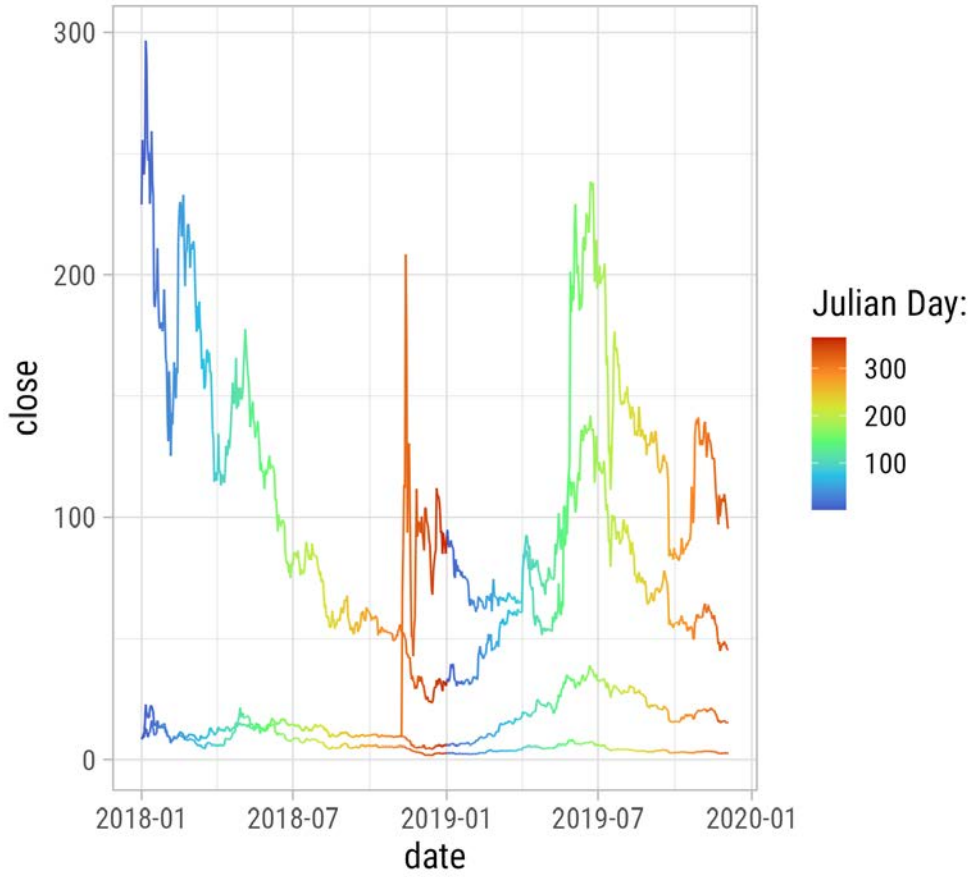
```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
  geom_line() +
  scale_color_viridis_d(
    option = "turbo",
    begin = .1,
    end = .9,
    name = "Cryptocurrency:"
  )
```



Scales: `scale_color_viridis_*()`

The viridis palettes are more suitable for continuous data:

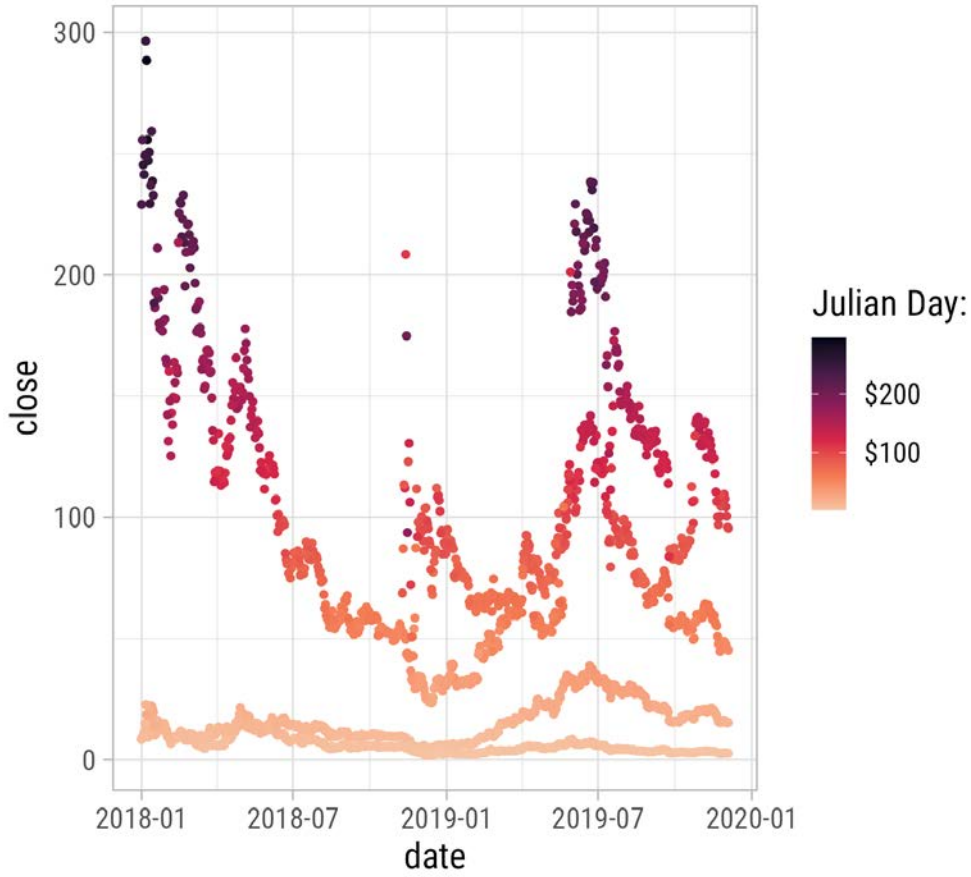
```
ggplot(data, aes(x = date, y = close,
                 color = yday,
                 group = currency)) +
  geom_line() +
  scale_color_viridis_c(
    option = "turbo",
    begin = .1,
    end = .9,
    name = "Julian Day:"
  )
```



Scales: `scale_color_viridis_*()`

The viridis palettes are more suitable for continuous data:

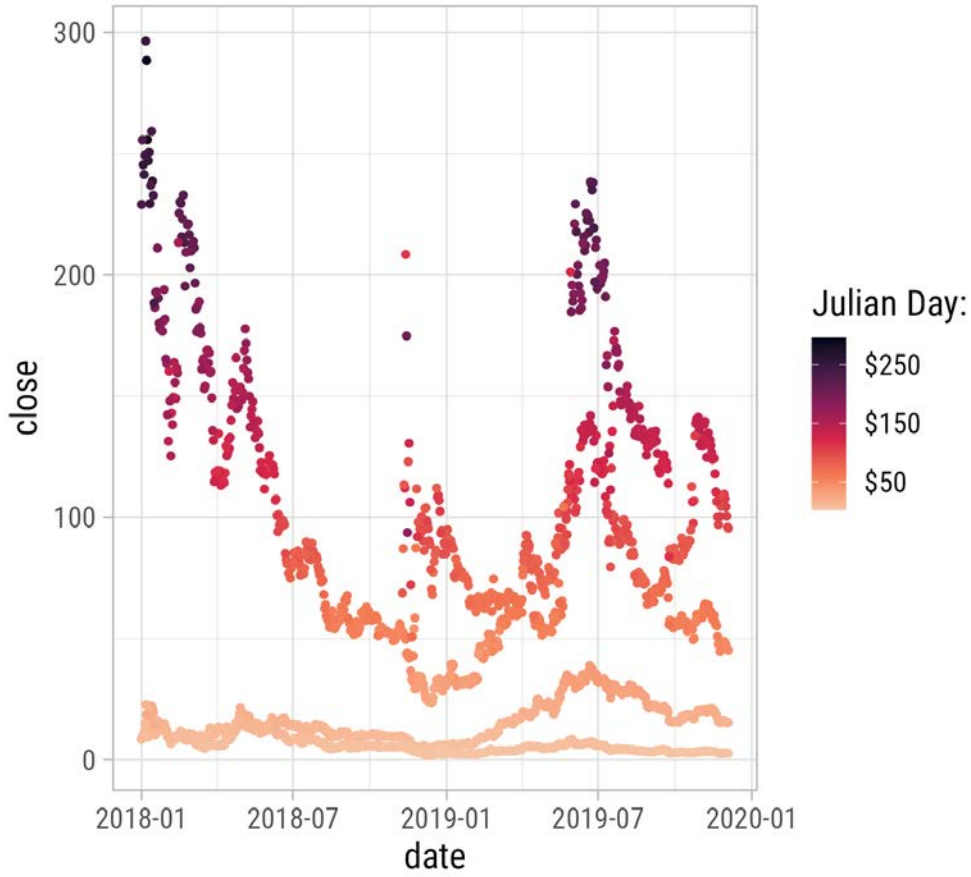
```
ggplot(data, aes(x = date, y = close,
                 color = open,
                 group = currency)) +
  geom_point() +
  scale_color_viridis_c(
    option = "rocket",
    direction = -1,
    end = .9,
    labels = scales::dollar_format(),
    name = "Julian Day:"
  )
```



Scales: `scale_color_viridis_*()`

The viridis palettes are more suitable for continuous data:

```
ggplot(data, aes(x = date, y = close,
                 color = open,
                 group = currency)) +
  geom_point() +
  scale_color_viridis_c(
    option = "rocket",
    direction = -1,
    end = .9,
    breaks = seq(50, 300, by = 100),
    labels = scales::dollar_format(),
    name = "Julian Day:"
  )
```



Scales: `scale_color_brewer_*()`

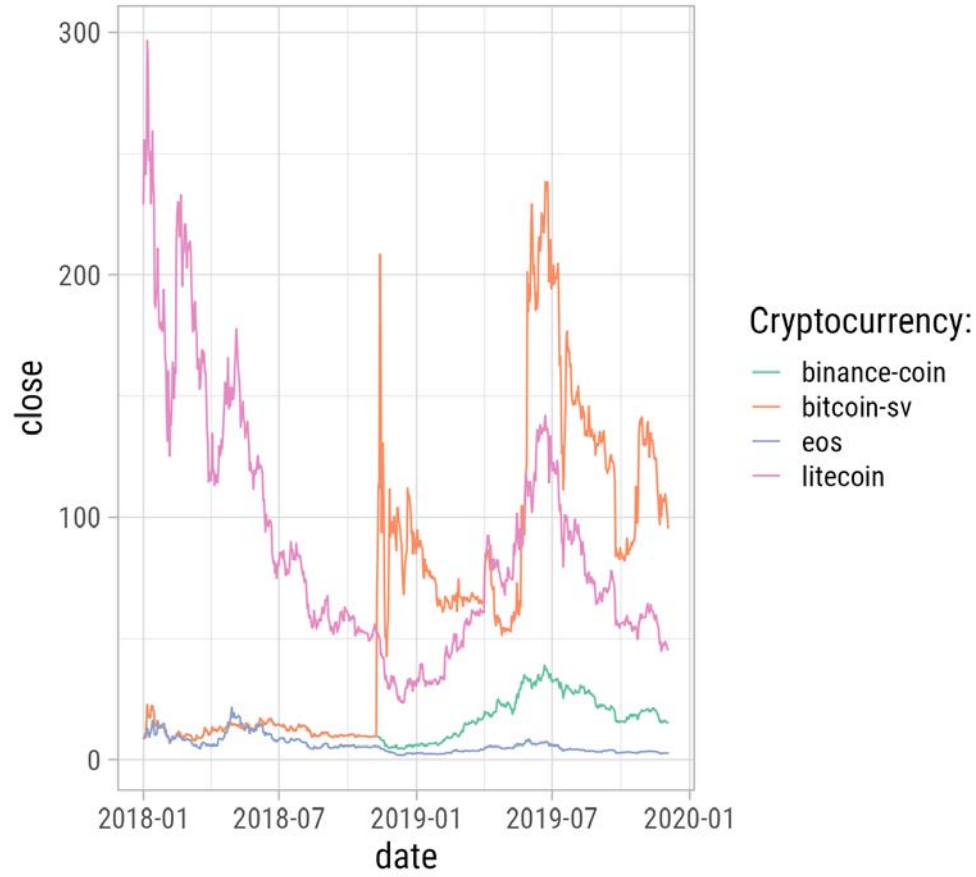
Colorbrewer provides color schemes for maps.

```
RColorBrewer::display.brewer.all()
```

Scales: `scale_color_brewer_*()`

Colorbrewer provides color schemes for maps.

```
ggplot(data, aes(x = date, y = close,
                 color = currency)) +
    geom_line() +
    scale_color_brewer(
        palette = "Set2",
        name = "Cryptocurrency:"
    )
```



Exercise 1:

Colorbrewer provides color schemes for **maps**.

- Download the country data set from `{rnaturalearth}`.
    - see segment 2 if you don't remember how
- Visualize the economy classes (`economy`) as a choropleth map and use one of the categorical colorbrewer palettes.
- Now change the code so the fill colors encode the estimated population (`pop_est`).
    - What's the problem? How can you fix it?

Exercise 1:

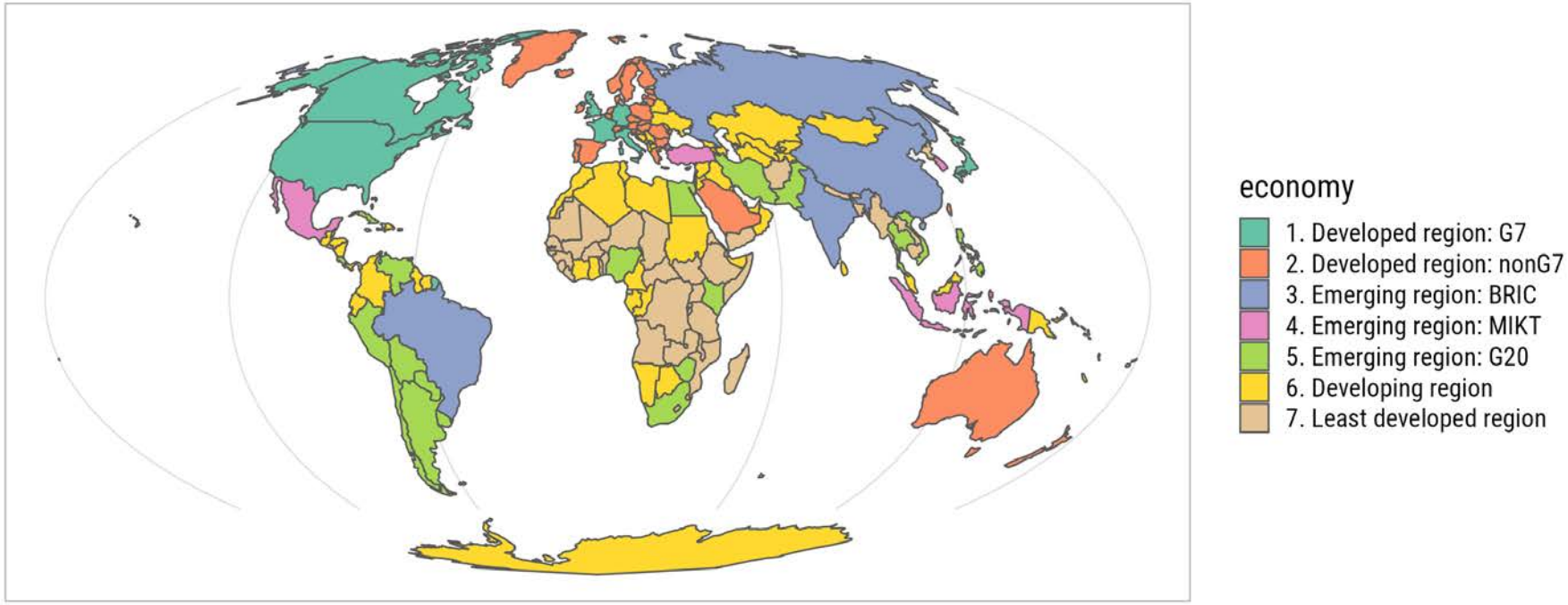
Colorbrewer provides color schemes for **maps**.

- Download the country data set from `{rnaturalearth}`.
    - see segment 2 if you don't remember how
- Visualize the economy classes (`economy`) as a choropleth map and use one of the categorical colorbrewer palettes.
- Now change the code so the fill colors encode the estimated population (`pop_est`).
    - What's the problem? How can you fix it?
    - Tip: Have a look at the note and examples on the help page: `?scale_fill_brewer`

Exercise 1: Choropleth Map of Economy Classes

Colorbrewer provides color schemes for **maps**.

```
sf_world <- rnaturalearth::ne_countries(returnclass = "sf")
ggplot(sf_world) +
  geom_sf(aes(fill = economy)) +
  scale_fill_brewer(palette = "Set2") +
  coord_sf(crs = "+proj=moll")
```



Exercise 1: Choropleth Map of Estimated Population

Colorbrewer provides color schemes for **maps**.

```
ggplot(sf_world) +
  geom_sf(aes(fill = pop_est)) +
  scale_fill_brewer(palette = "Set2") +
  coord_sf(crs = "+proj=moll")
```

```
Error: Continuous value supplied to discrete scale
```

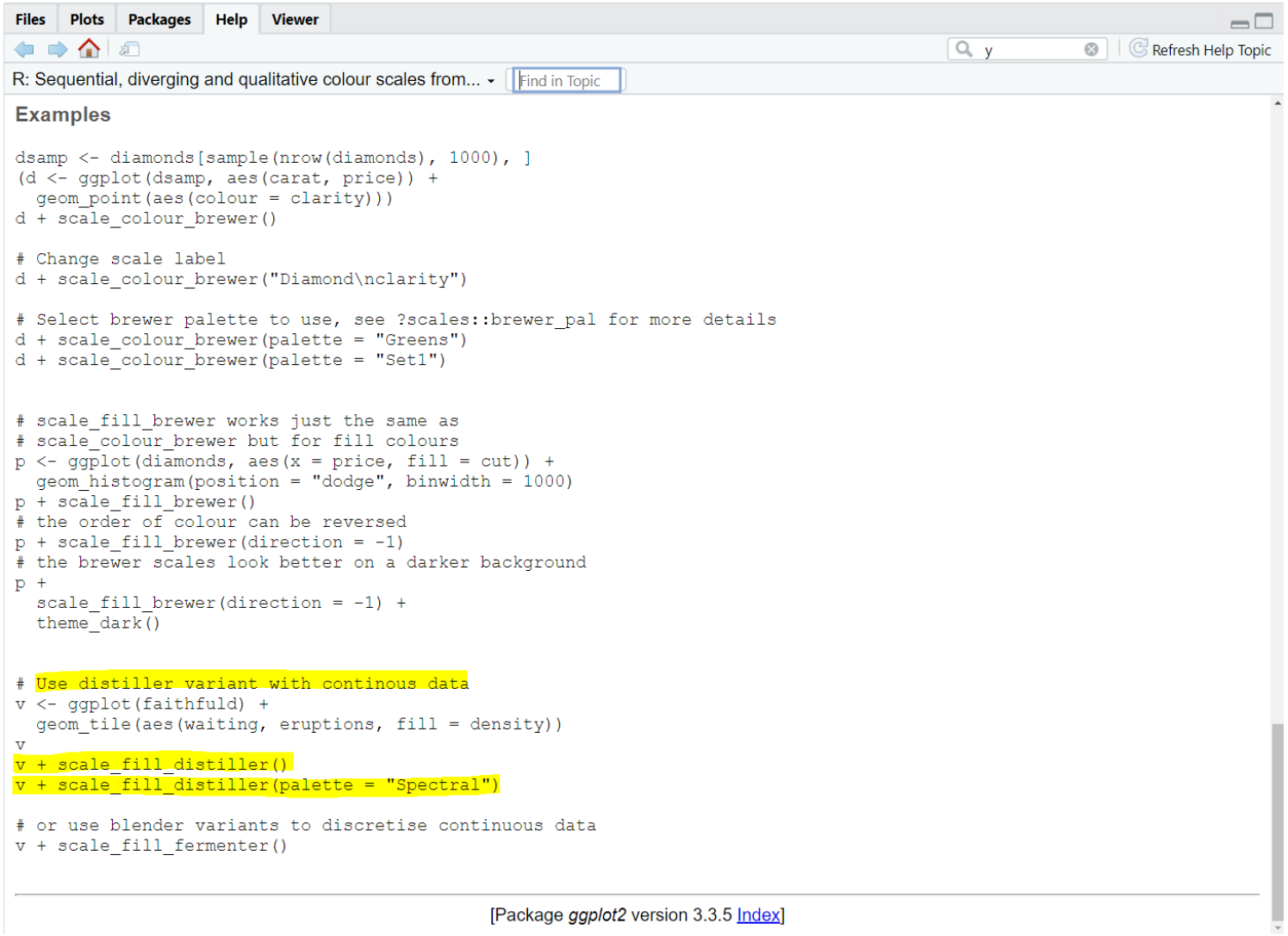
Exercise 1: Choropleth Map of Estimated Population

Colorbrewer provides color schemes for **maps**.

```
ggplot(sf_world) +
  geom_sf(aes(fill = pop_est)) +
  scale_fill_brewer(palette = "RdPu") +
  coord_sf(crs = "+proj=moll")
```

```
Error: Continuous value supplied to discrete scale
```

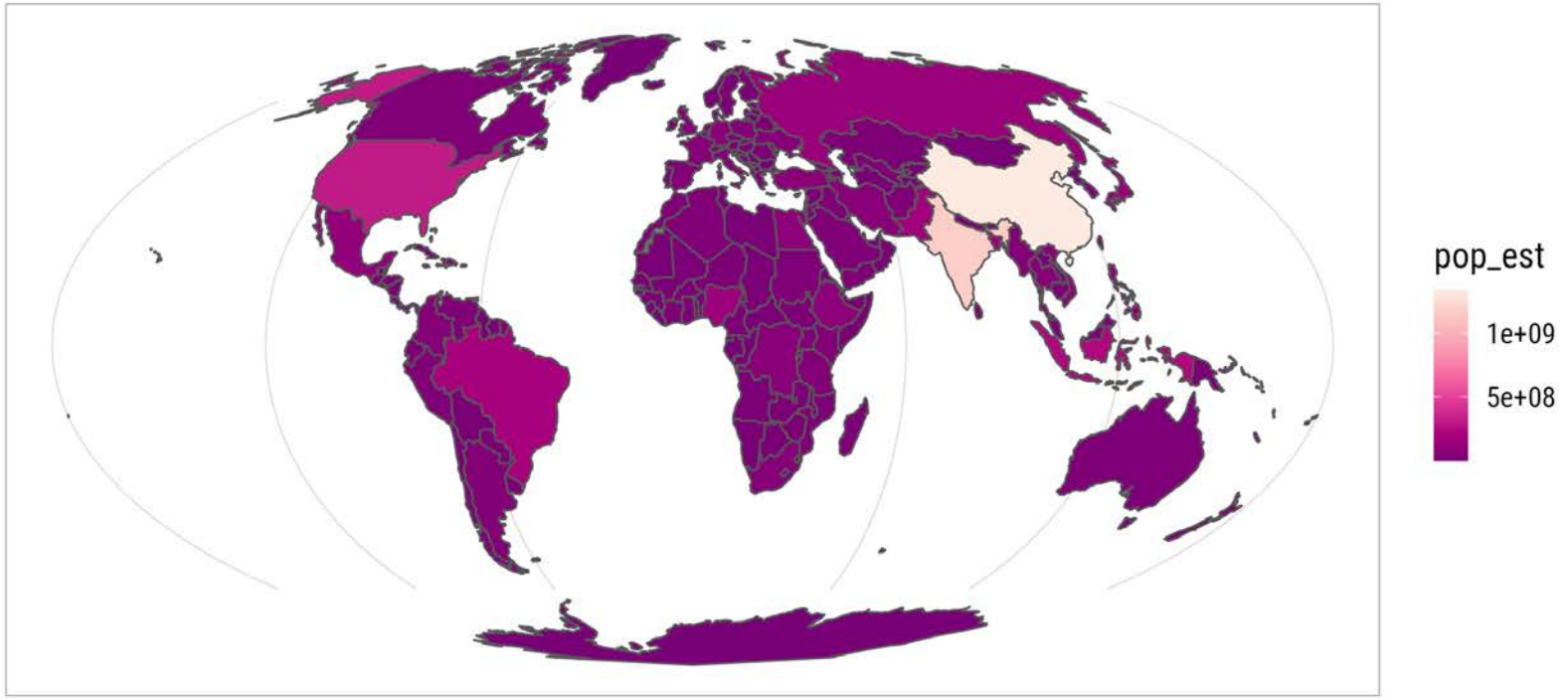
Exercise 1: Choropleth Map of Estimated Population



Exercise 1: Choropleth Map of Estimated Population

Colorbrewer provides color schemes for **maps**.

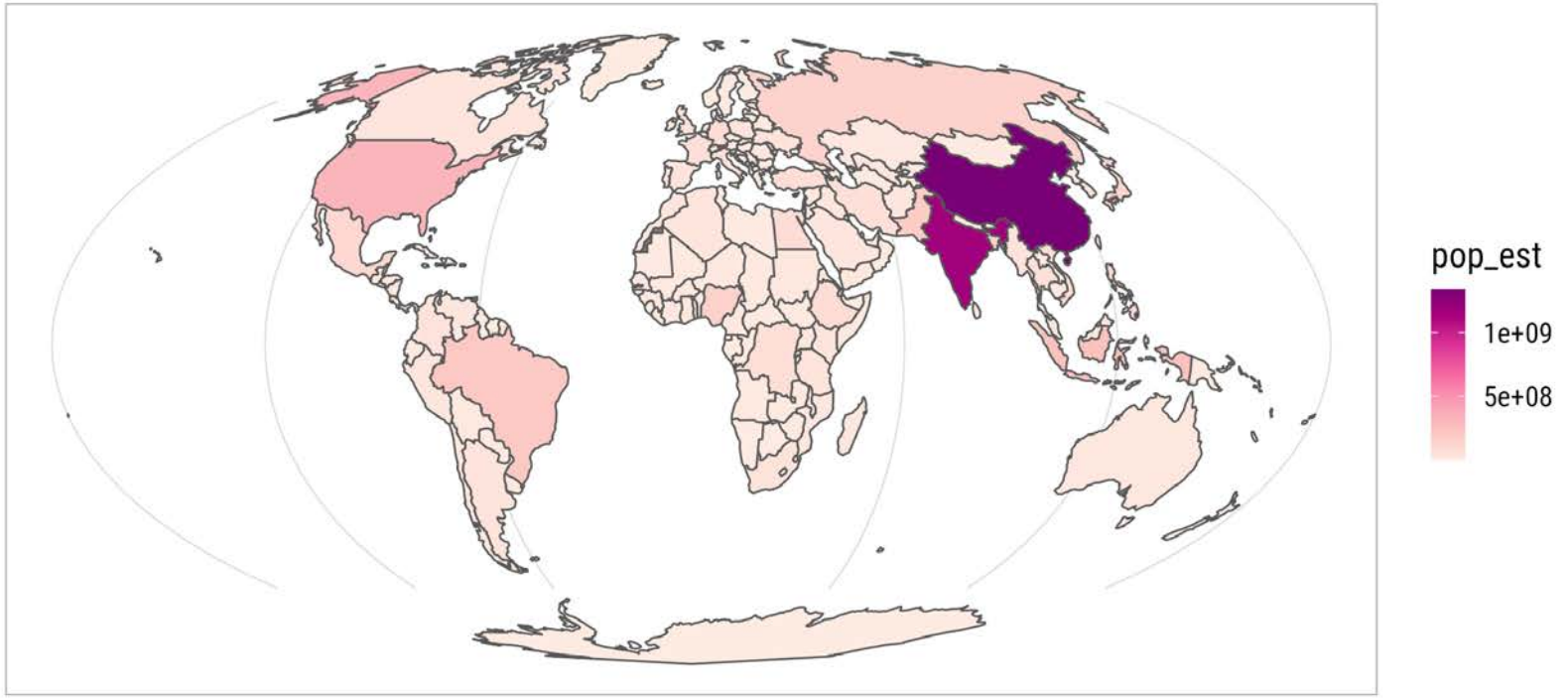
```
ggplot(sf_world) +
  geom_sf(aes(fill = pop_est)) +
  scale_fill_distiller(palette = "RdPu") +
  coord_sf(crs = "+proj=moll")
```



Exercise 1: Choropleth Map of Estimated Population

Colorbrewer provides color schemes for **maps**.

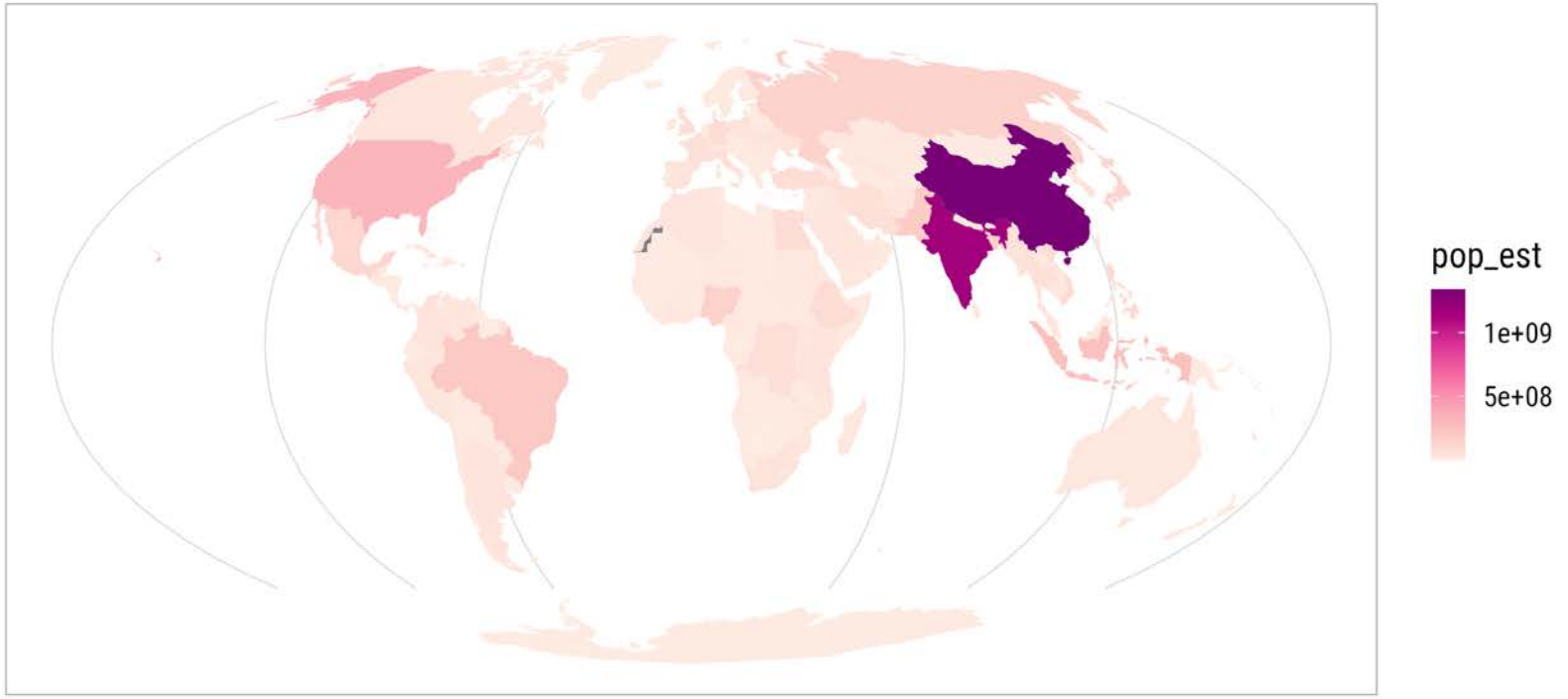
```
ggplot(sf_world) +
  geom_sf(aes(fill = pop_est)) +
  scale_fill_distiller(palette = "RdPu", direction = 1) +
  coord_sf(crs = "+proj=moll")
```



Exercise 1: Choropleth Map of Estimated Population

Colorbrewer provides color schemes for **maps**.

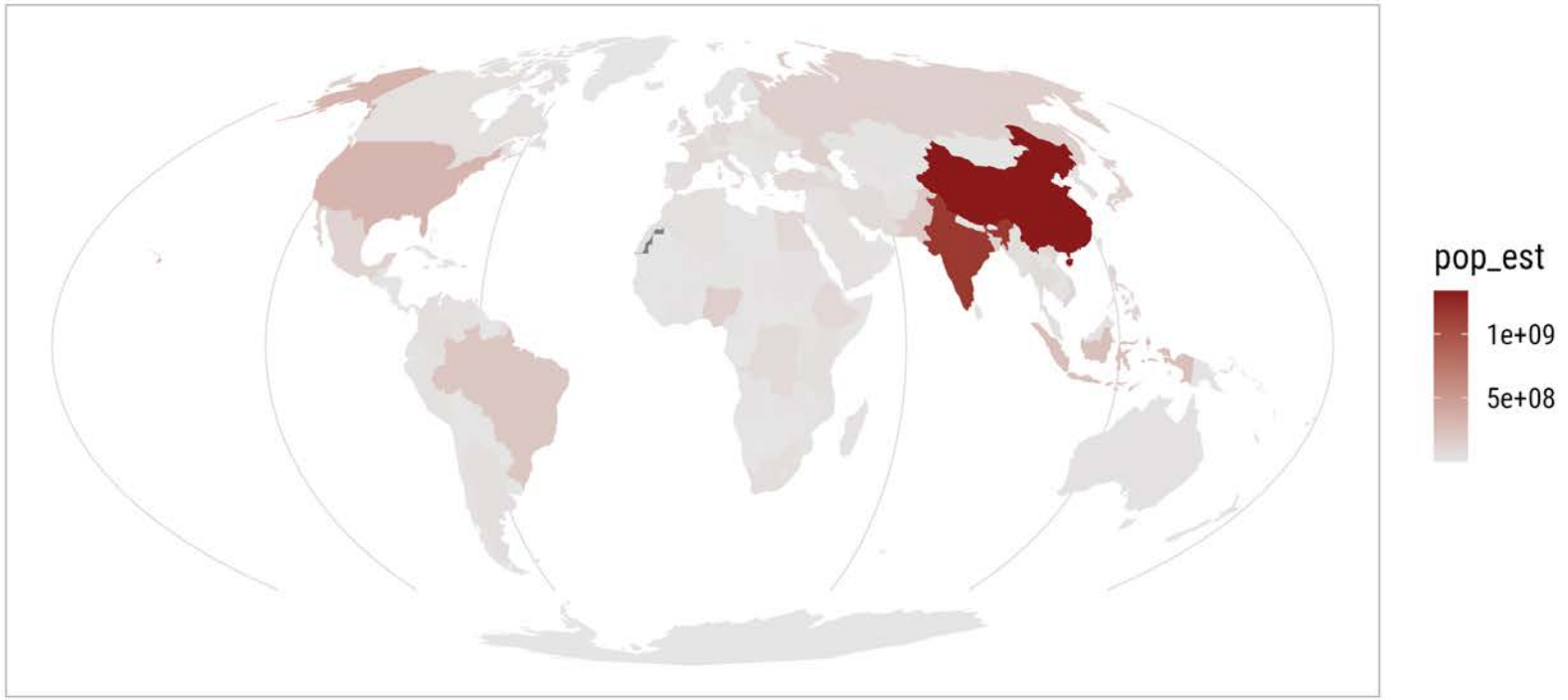
```
ggplot(sf_world) +
  geom_sf(aes(fill = pop_est), color = NA) +
  scale_fill_distiller(palette = "RdPu", direction = 1) +
  coord_sf(crs = "+proj=moll")
```



Scales: `scale_fill_gradient()`

You can also build your own **sequential color palettes**:

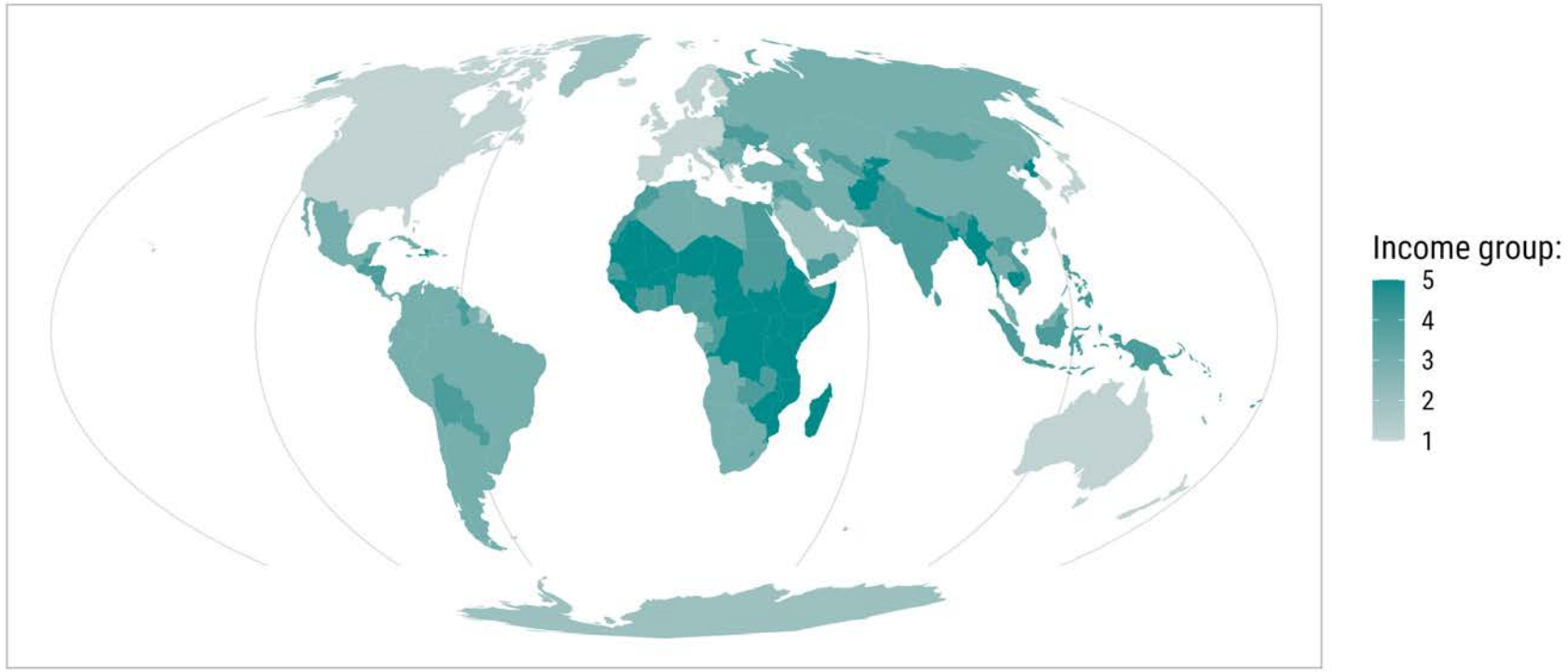
```
ggplot(sf_world) +
  geom_sf(aes(fill = pop_est), color = NA) +
  scale_fill_gradient(low = "grey90", high = "firebrick4") +
  coord_sf(crs = "+proj=moll")
```



Scales: `scale_fill_gradient2()`

You can also build your own **diverging color palettes**:

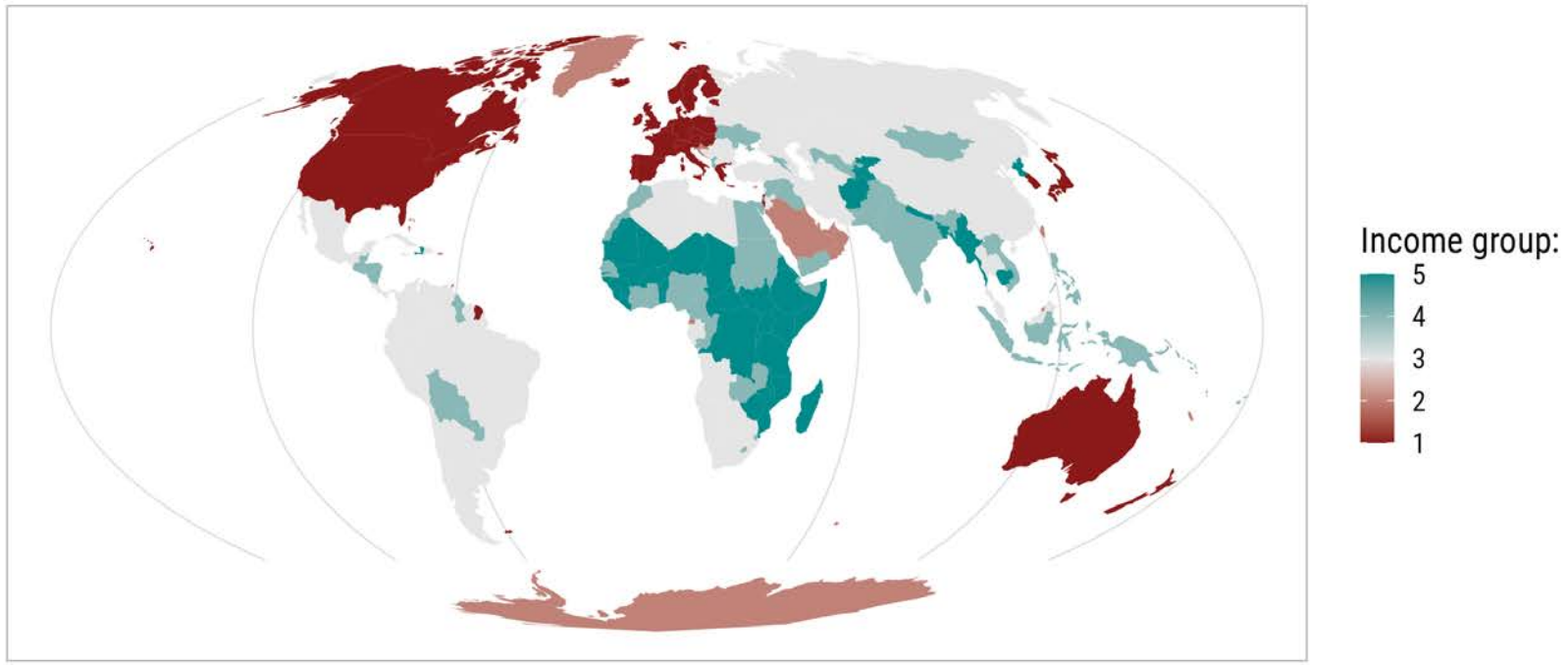
```
ggplot(sf_world) +
  geom_sf(aes(fill = as.numeric(factor(income_grp))), color = NA) +
  scale_fill_gradient2(low = "firebrick4", high = "darkcyan", mid = "grey90", name = "Income group:") +
  coord_sf(crs = "+proj=moll")
```



Scales: `scale_fill_gradient2()`

You can also build your own **diverging color palettes**:

```
ggplot(sf_world) +
  geom_sf(aes(fill = as.numeric(factor(income_grp))), color = NA) +
  scale_fill_gradient2(low = "firebrick4", high = "darkcyan", mid = "grey90", name = "Income group:",
                       midpoint = 3) +
  coord_sf(crs = "+proj=moll")
```



Color Palettes in R

Several packages offer predefined palettes, e.g.:

- `{viridis}` for perceptually uniform palettes
- `{scico}` for more perceptually uniform palettes
- `{rcartocolor}` for map color palettes
- `{ggsci}` for scientific journal and sci-fi themed color
- `{ggthemes}` for colors of popular software & publishers
- `{LaCroixColoR}` for vibrant summery colors

Check the `collection by Emil Hvitfeldt` for an extensive list of color palettes available in R

A ggplot Object

By the way, you can store ggplots in objects and extend them later:

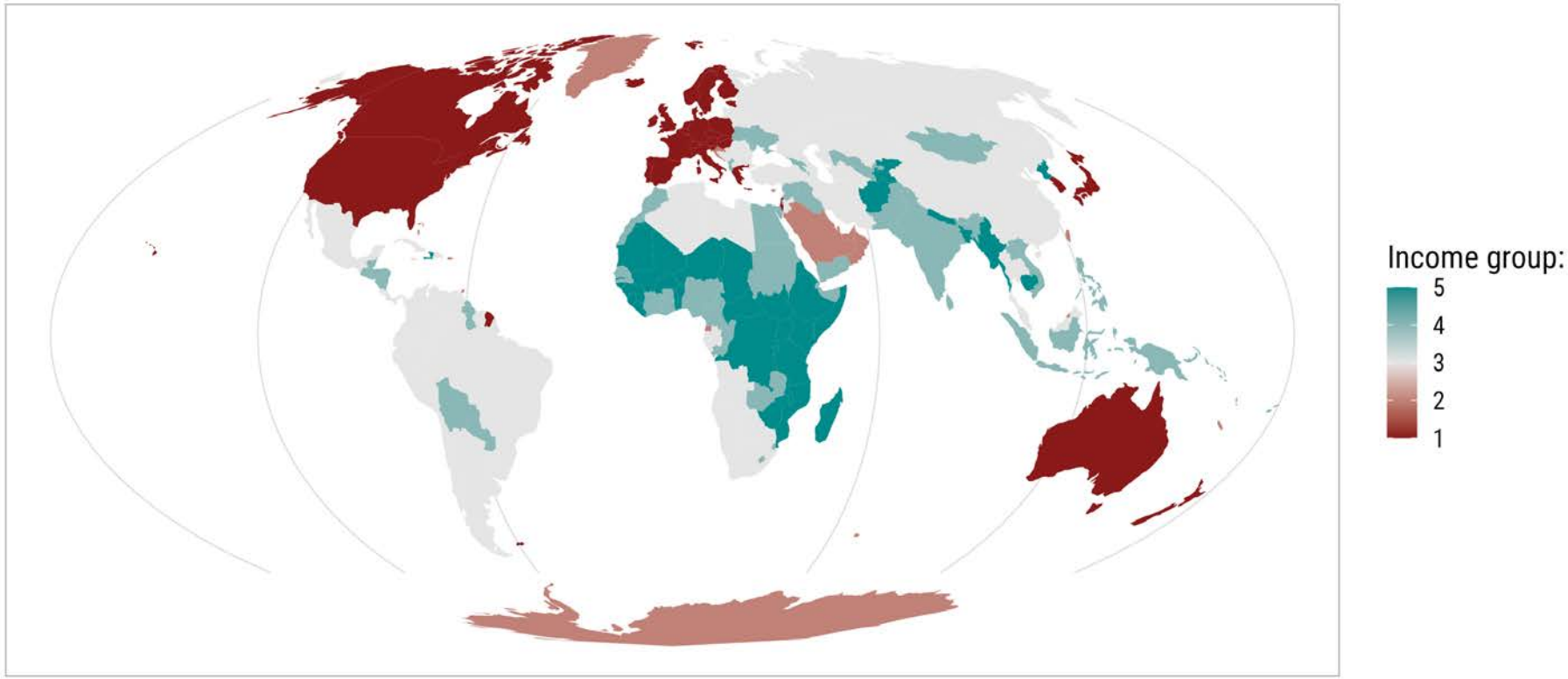
```
g <- ggplot(sf_world) +
  geom_sf(aes(fill = as.numeric(factor(income_grp))), color = NA) +
  scale_fill_gradient2(low = "firebrick4", high = "darkcyan", mid = "grey90", name = "Income group:",
                       midpoint = 3) +
  coord_sf(crs = "+proj=moll")

class(g)
## [1] "gg"     "ggplot"
```

Legends
(Guides)

Style Legends: `guides()`

The `guides()` function allows to adjust the appearance of each legend:

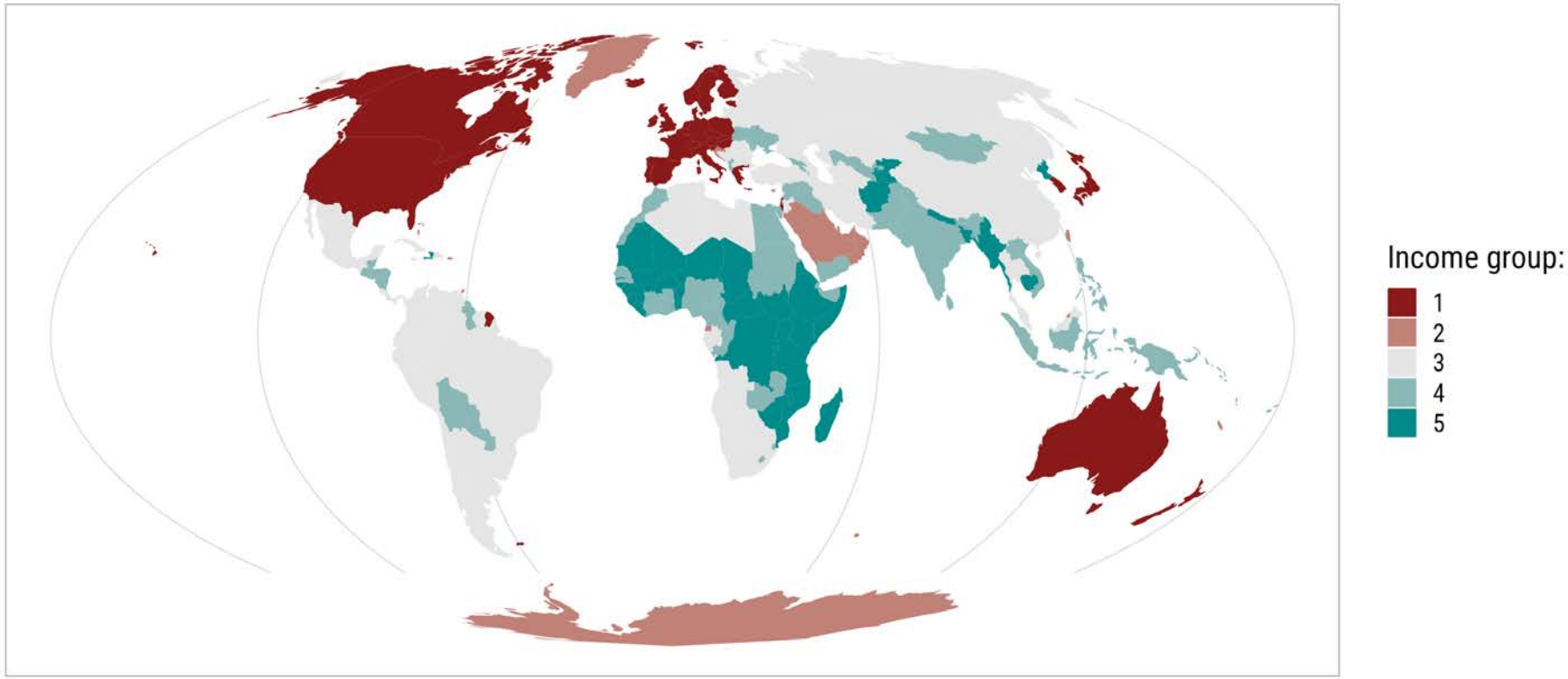
```
g +
  guides(fill = guide_colorbar())
```



Style Legends: `guides()`

The `guides()` function allows to adjust the appearance of each legend:

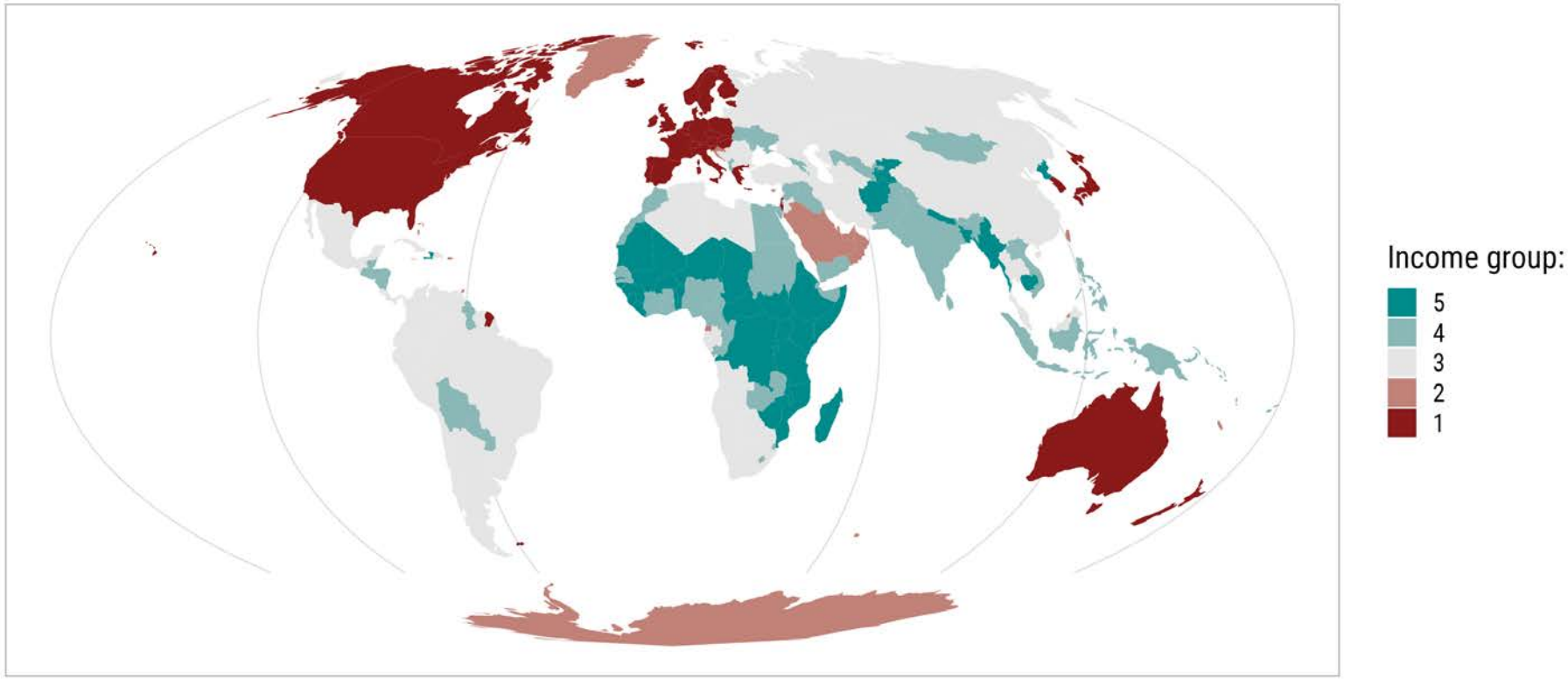
```
g +
  guides(fill = guide_legend())
```



Style Legends: `guides()`

The `guides()` function allows to adjust the appearance of each legend:

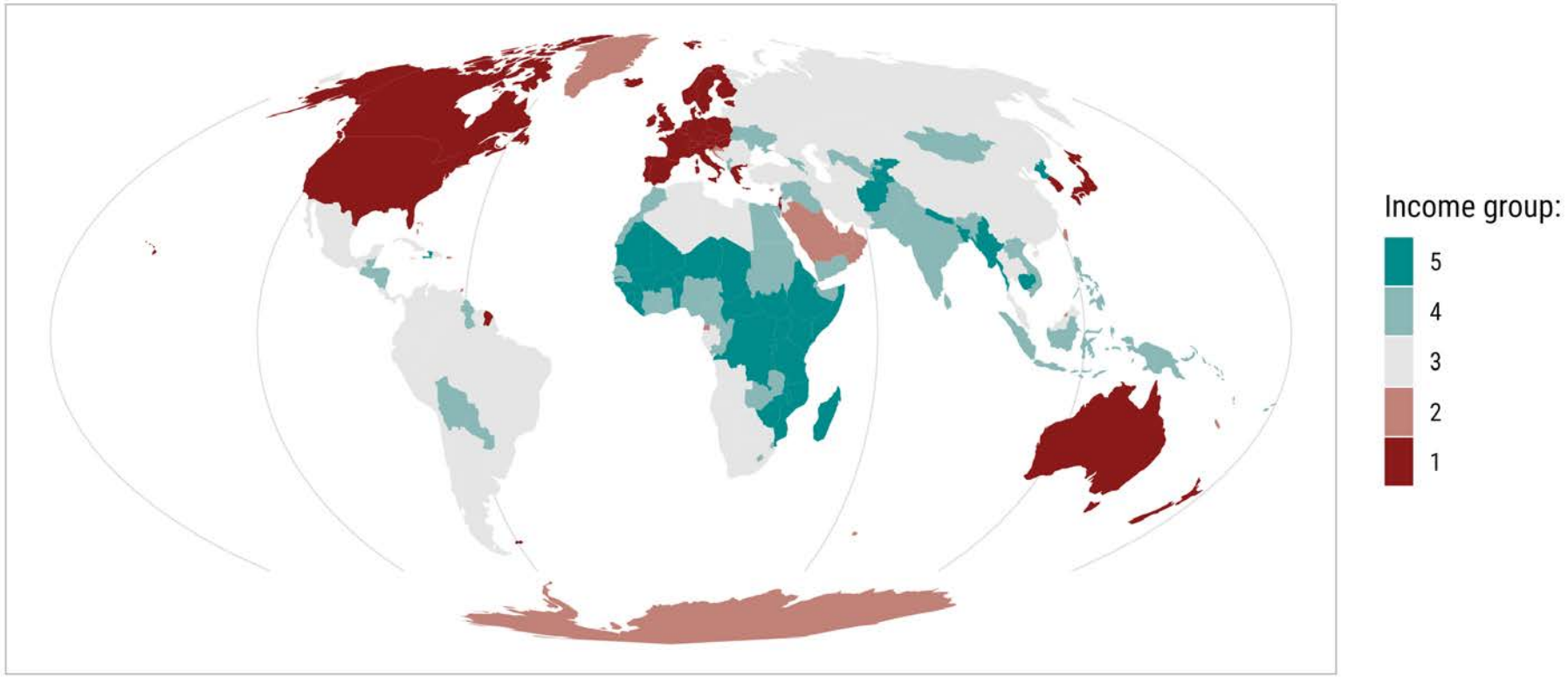
```
g +
  guides(fill = guide_legend(reverse = TRUE))
```



Style Legends: `guides()`

The `guides()` function allows to adjust the appearance of each legend:

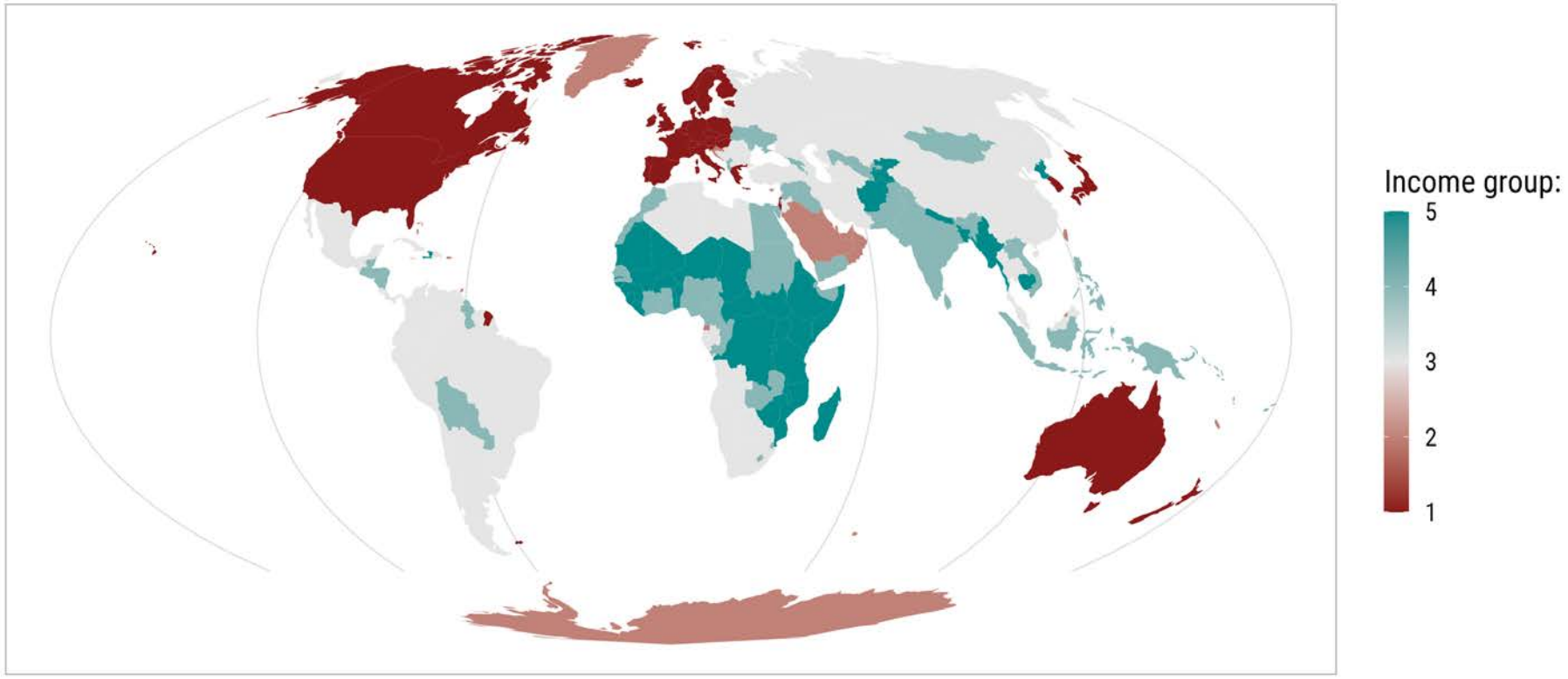
```
g +
  guides(fill = guide_legend(reverse = TRUE, keyheight = unit(2, "lines")))
```



Style Legends: `guides()`

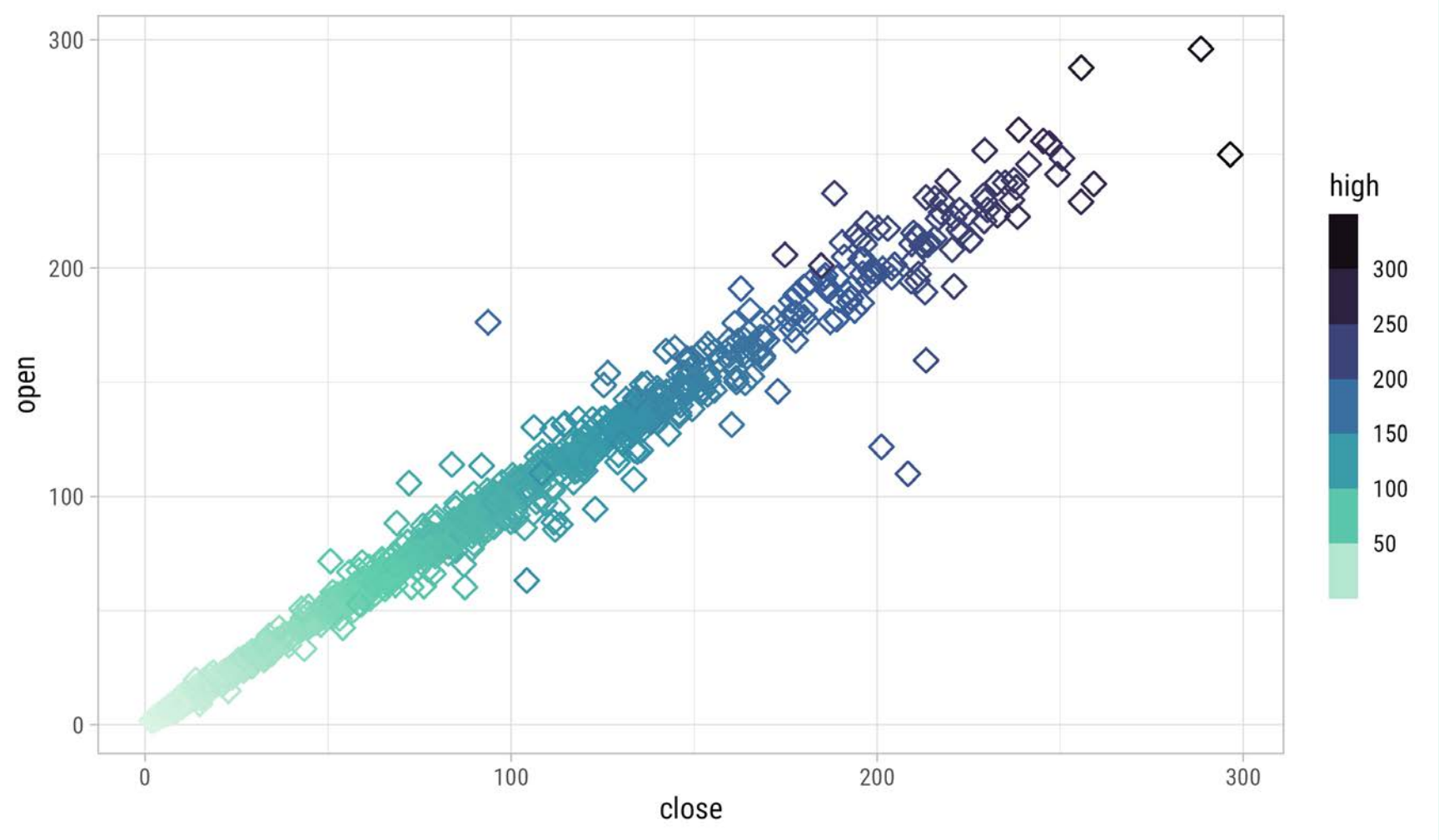
The `guides()` function allows to adjust the appearance of each legend:

```
g +
  guides(fill = guide_colorbar(barheight = unit(12, "lines"),
                               barwidth = unit(1, "lines")))
```



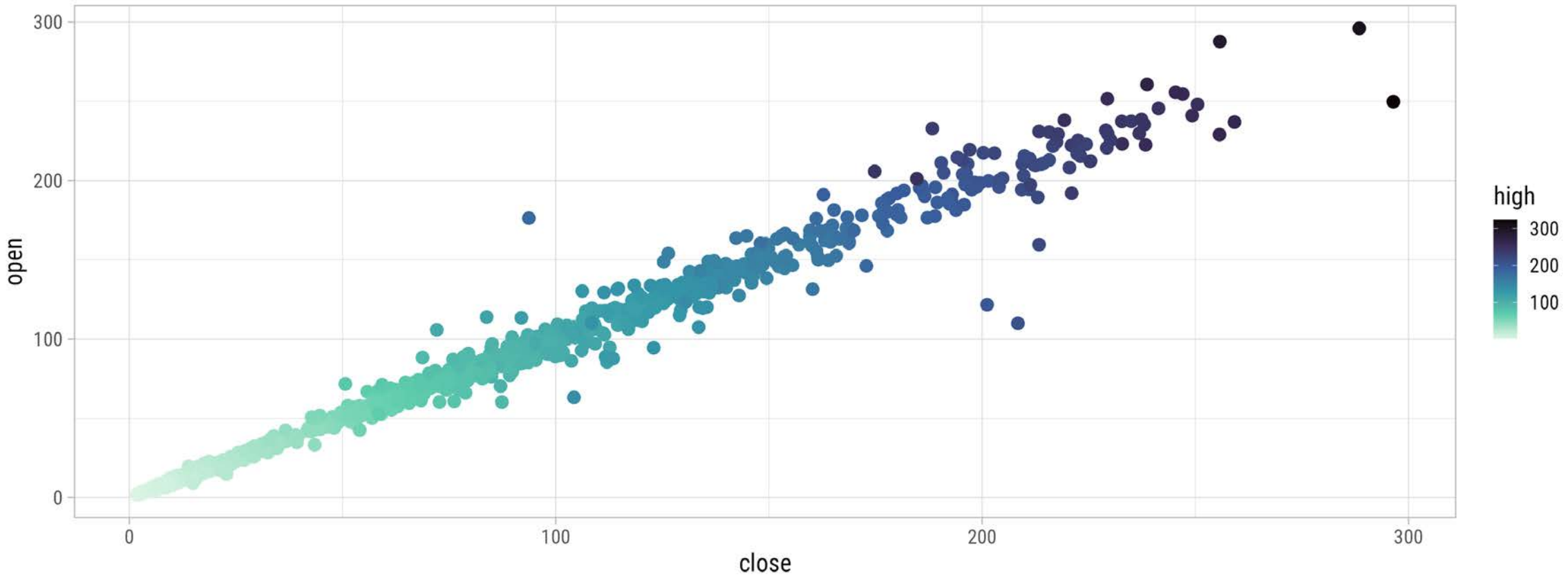
Exercise 2:

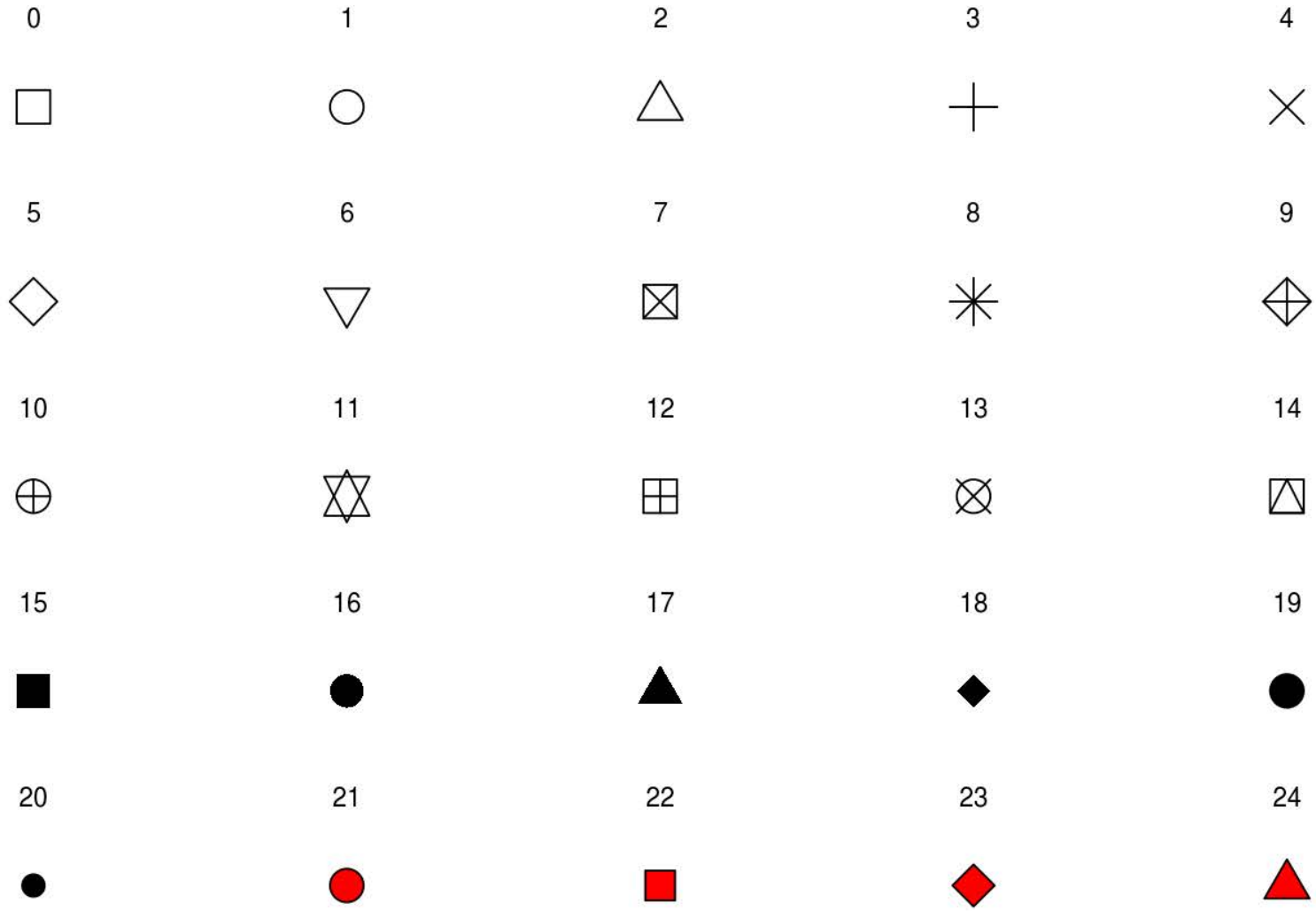
- Create the following visualization with a viridis palette:



Exercise 2: Create a Colored Scatterplot

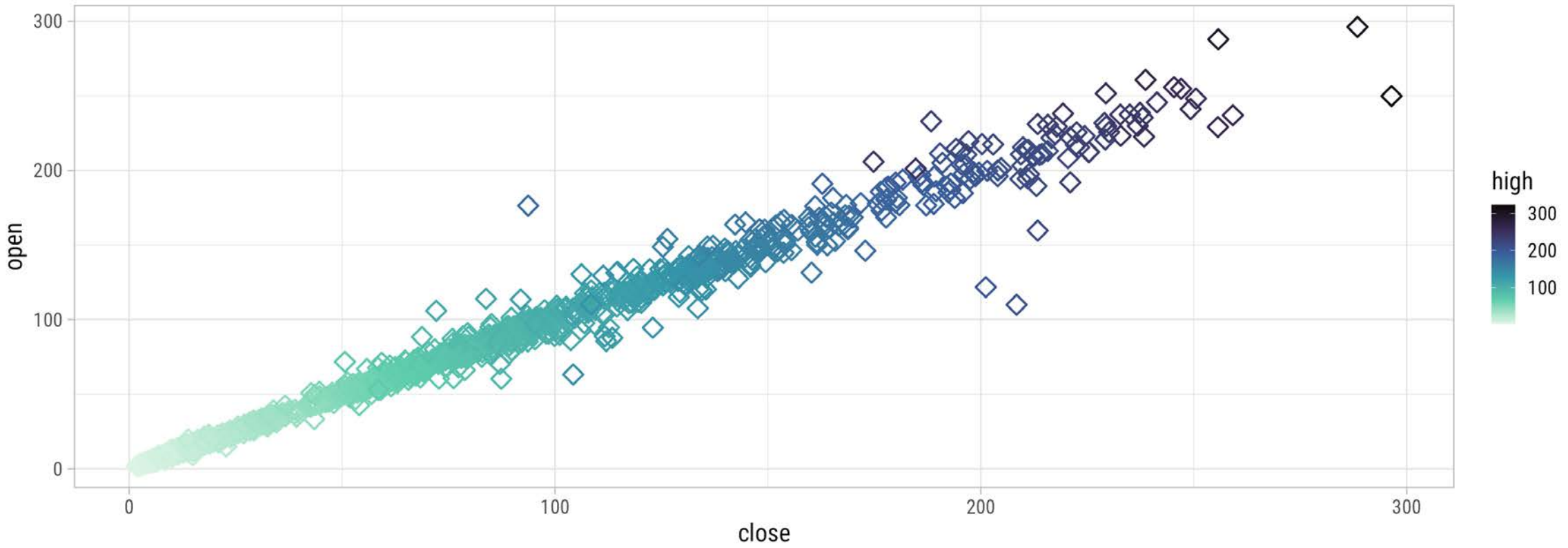
```
ggplot(data, aes(close, open, color = high)) +
  geom_point(size = 4) +
  scale_color_viridis_c(option = "mako", direction = -1)
```




0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

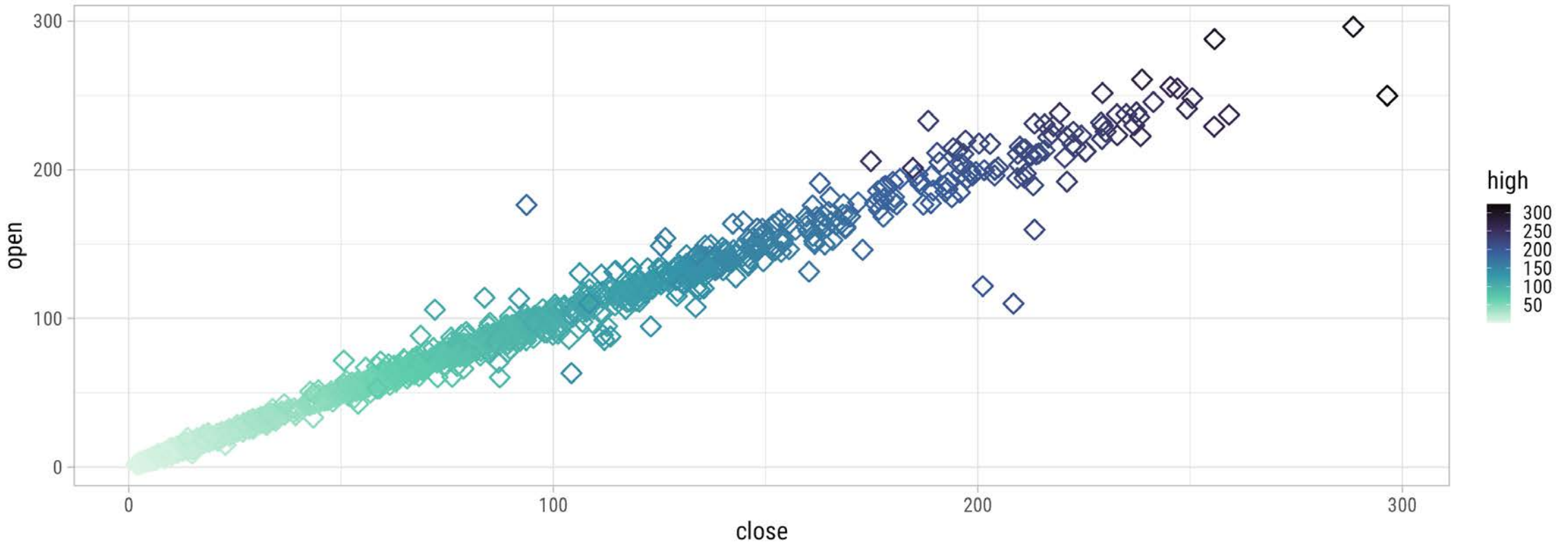
Exercise 2: Adjust Point Style

```
ggplot(data, aes(close, open, color = high)) +
  geom_point(shape = 5, size = 4, stroke = 1.2) +
  scale_color_viridis_c(option = "mako", direction = -1)
```



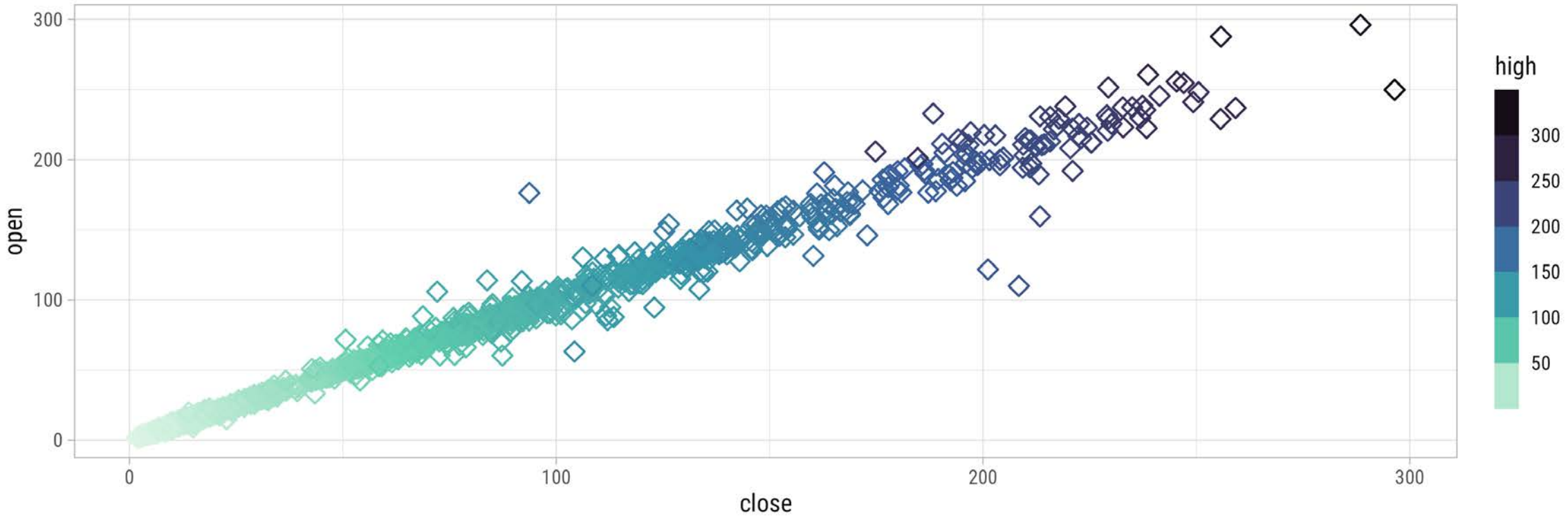
Exercise 2: Adjust Breaks

```
ggplot(data, aes(close, open, color = high)) +
  geom_point(shape = 5, size = 4, stroke = 1.2) +
  scale_color_viridis_c(option = "mako", direction = -1, breaks = seq(50, 350, by = 50))
```



Exercise 2: Adjust Guide Type and Style

```
ggplot(data, aes(close, open, color = high)) +
  geom_point(shape = 5, size = 4, stroke = 1.2) +
  scale_color_viridis_c(option = "mako", direction = -1, breaks = seq(50, 350, by = 50)) +
  guides(color = guide_colorsteps(barheight = unit(16, "lines")))
```



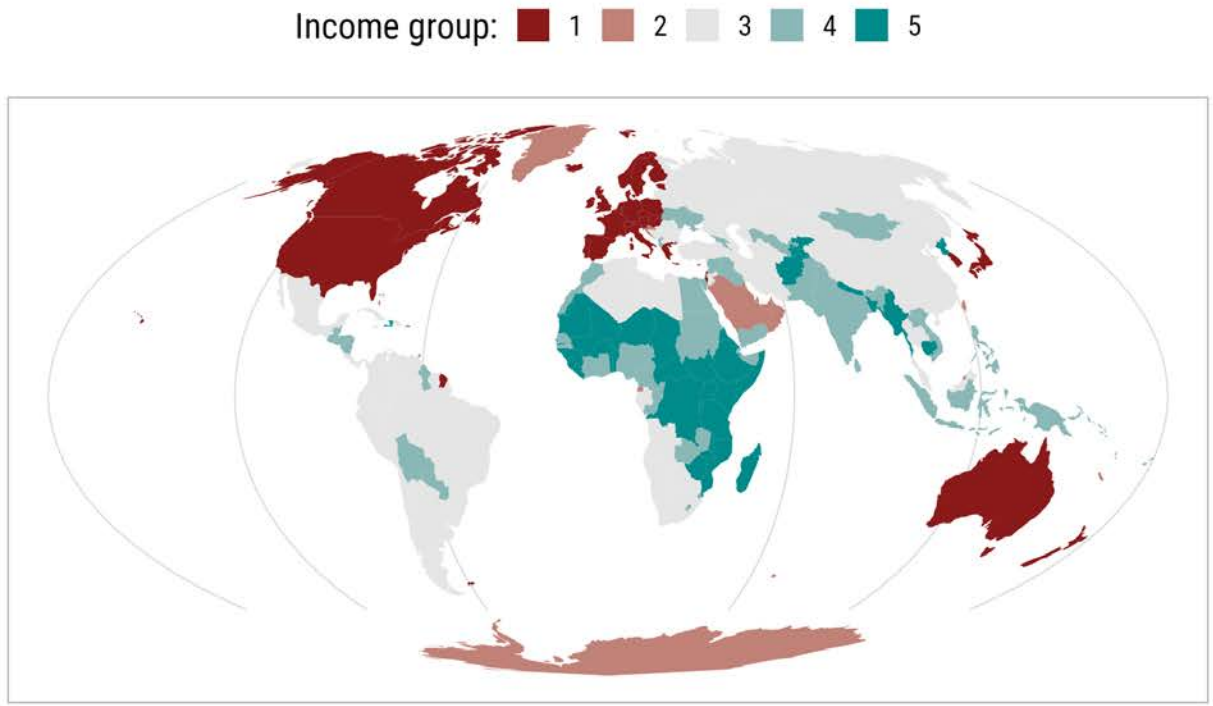
POLL: What isthe best position for the legend?

- On the right.
- On the left.
- At the top.
- At the bottom.
- Inside the chart.

Style Legends: `theme()`

The position of the legend and some other properties can be changed via `theme()`:

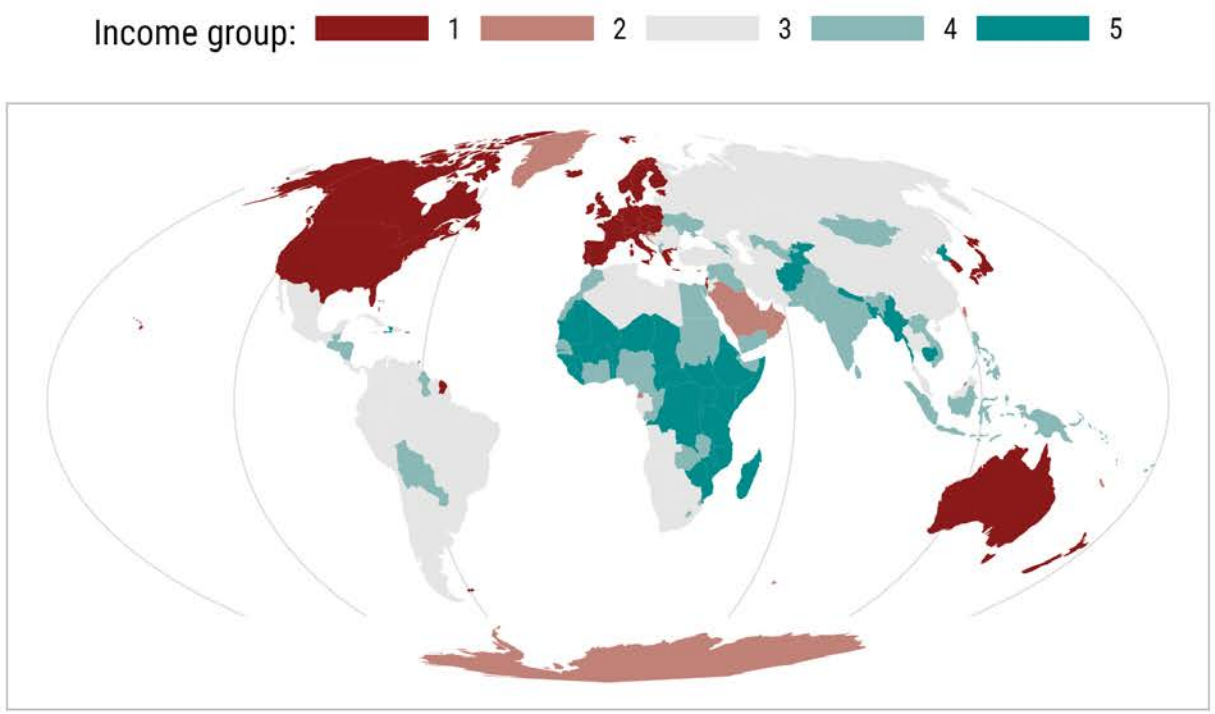
```
g +
  guides(fill = guide_legend()) +
  theme(legend.position = "top")
```



Style Legends: `theme()`

The position of the legend and some other properties can be changed via `theme()`:

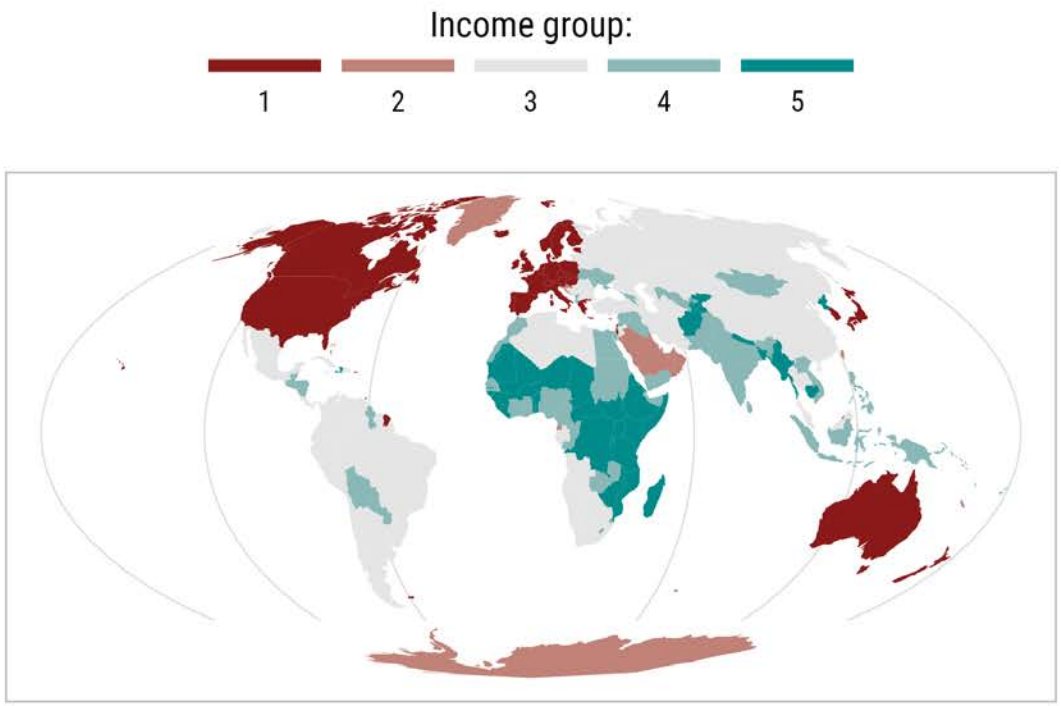
```
g +
  guides(fill = guide_legend()) +
  theme(legend.position = "top", legend.key.height = unit(.5, "lines"),
        legend.key.width = unit(4, "lines"))
```



Style Legends: `theme()`

You can change the position of the legend title and labels in `guides()`:

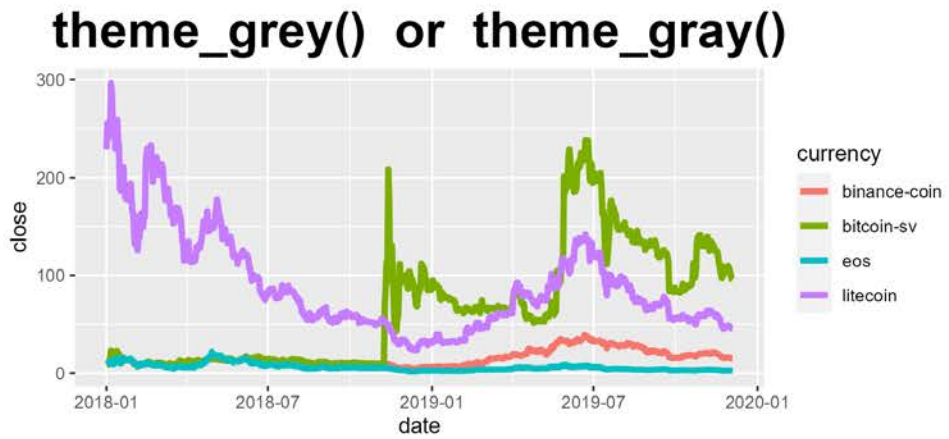
```
g +
  guides(fill = guide_legend(title.position = "top", title.hjust = .5, label.position = "bottom")) +
  theme(legend.position = "top", legend.key.height = unit(.5, "lines"),
        legend.key.width = unit(4, "lines"))
```

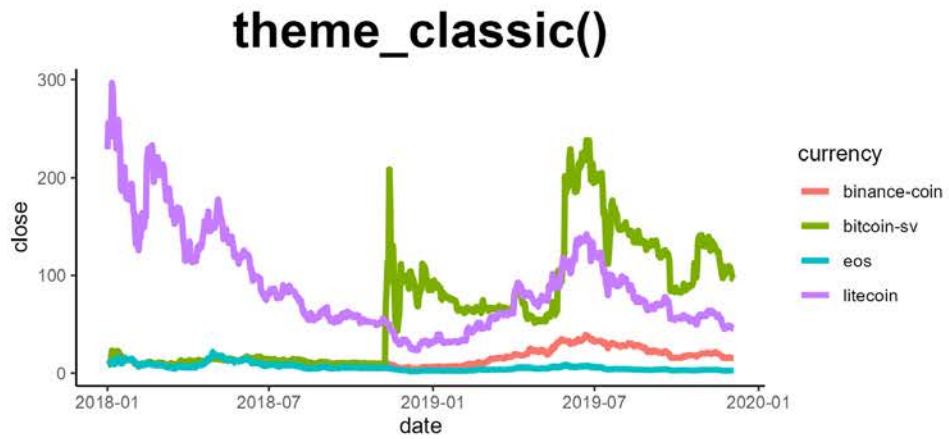


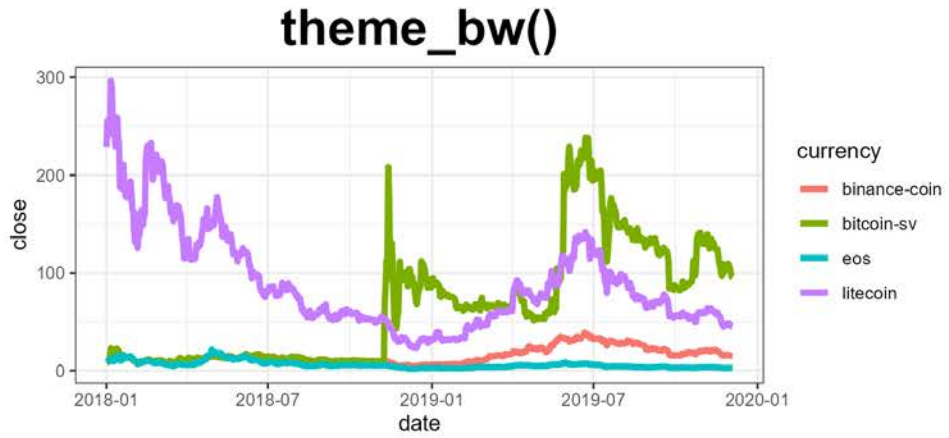
Template and Custom Theming

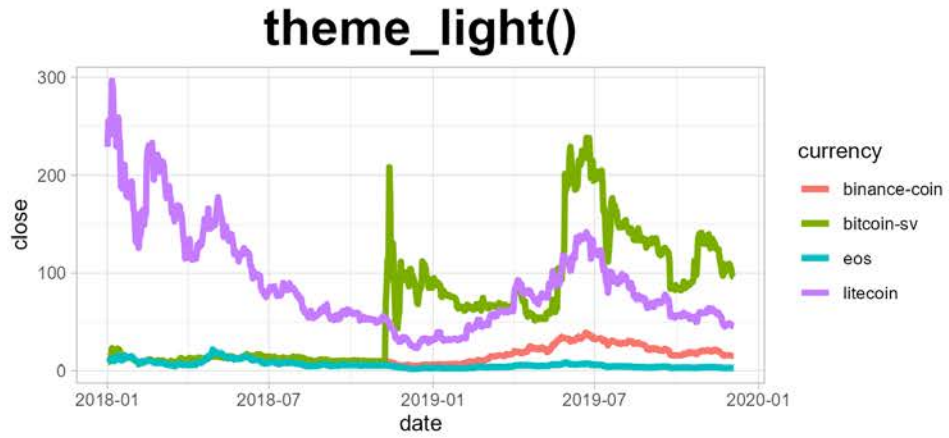
Style Legends: `theme()`

You have already seen built-in themes in segment 1:



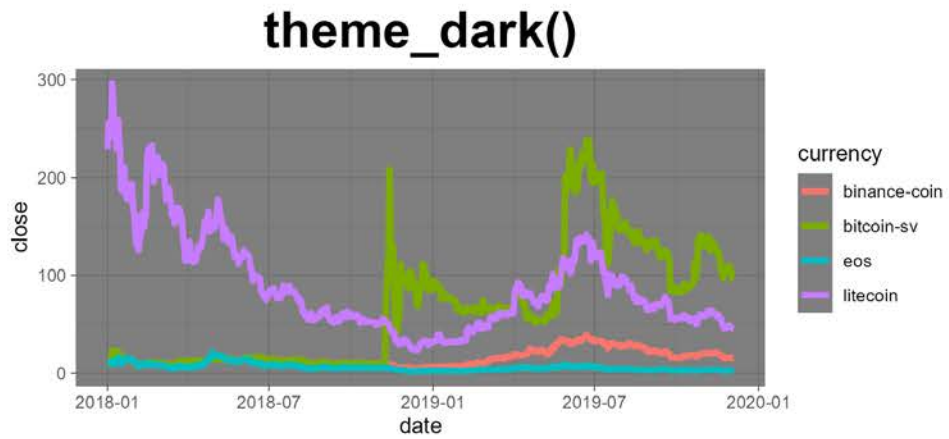


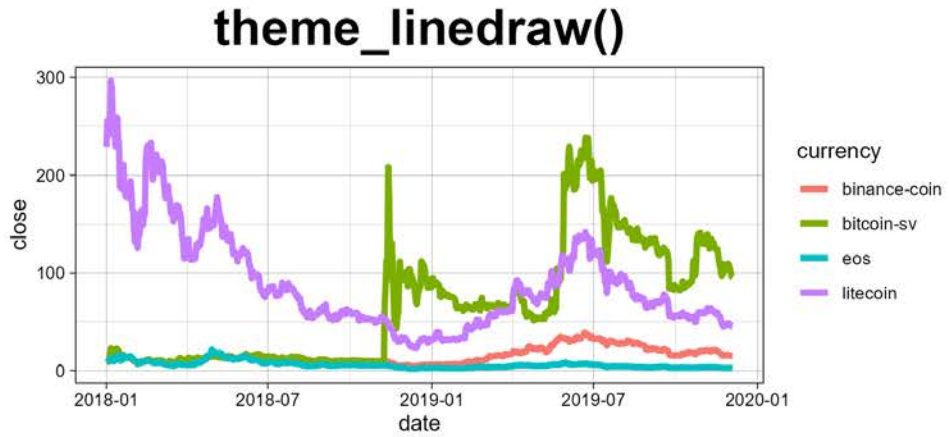


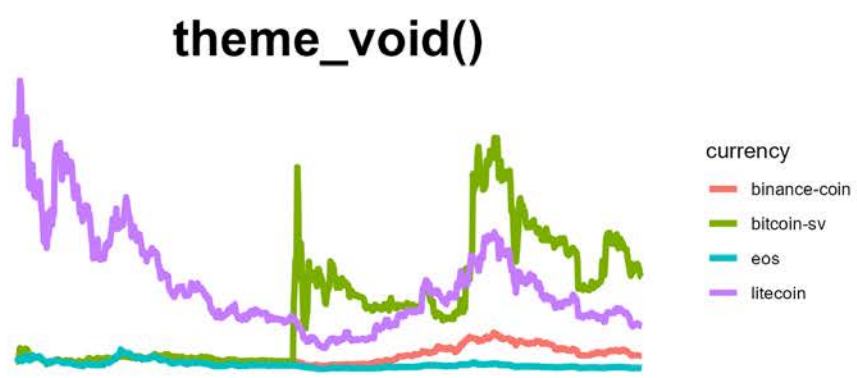


Style Legends: `theme()`

You have already seen built-in themes in segment 1:



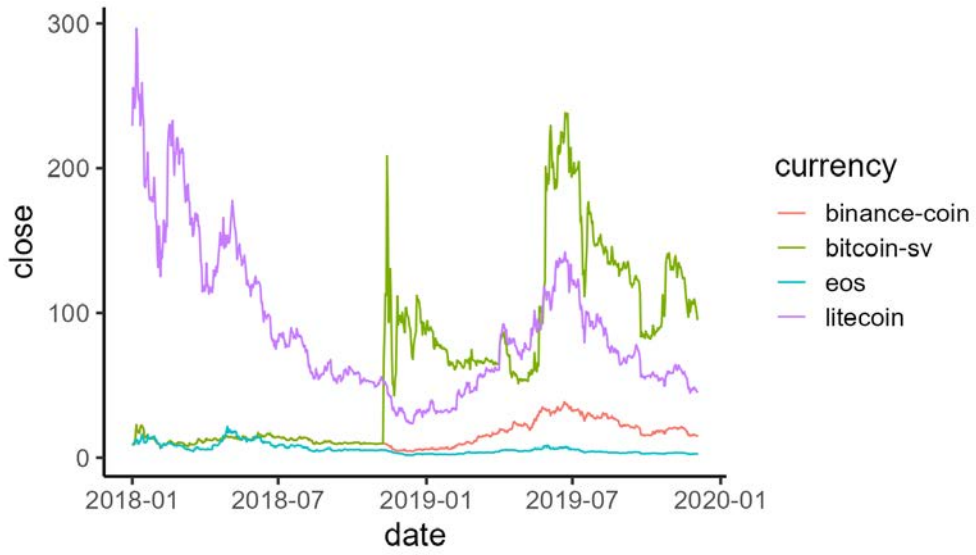




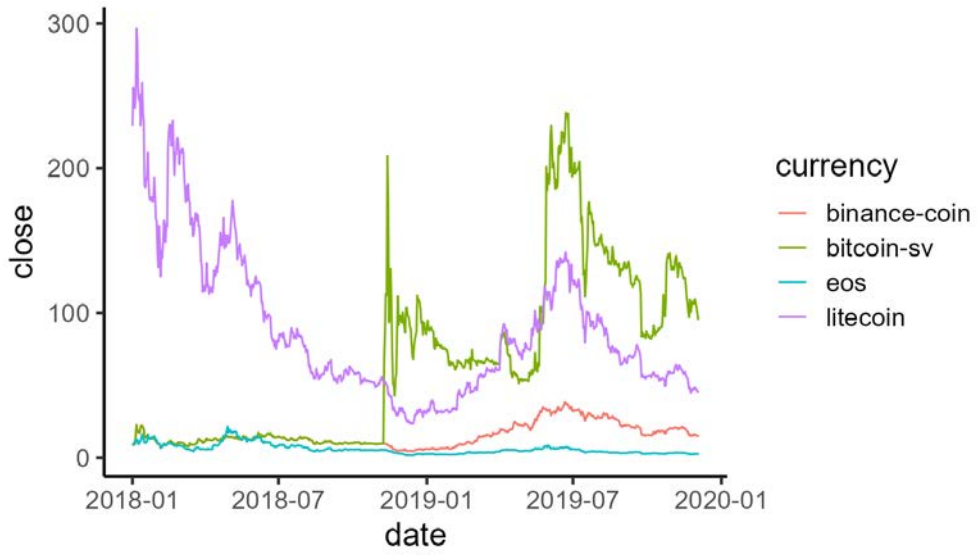
Style Legends: `theme()`

You can either add theme changes to each plot or globally for all plots:

```
ggplot(data, aes(x = date, y = close)) +
  geom_line(aes(color = currency)) +
  theme_classic(base_size = 16)
```




```
theme_set(theme_classic(base_size = 16))
ggplot(data, aes(x = date, y = close)) +
  geom_line(aes(color = currency))
```



```
theme_grey
## function (base_size = 11, base_family = "", base_line_size = base_size/22,
##     base_rect_size = base_size/22)
## {
##     half_line <- base_size/2
##     t <- theme(line = element_line(colour = "black", size = base_line_size,
##         linetype = 1, lineend = "butt"), rect = element_rect(fill = "white",
##         colour = "black", size = base_rect_size, linetype = 1),
##         text = element_text(family = base_family, face = "plain",
##             colour = "black", size = base_size, lineheight = 0.9,
##             hjust = 0.5, vjust = 0.5, angle = 0, margin = margin(),
##             debug = FALSE), axis.line = element_blank(), axis.line.x = NULL,
##         axis.line.y = NULL, axis.text = element_text(size = rel(0.8),
##             colour = "grey30"), axis.text.x = element_text(margin = margin(t = 0.8 *
##             half_line/2), vjust = 1), axis.text.x.top = element_text(margin = margin(b = 0.8 *
##             half_line/2), vjust = 0), axis.text.y = element_text(margin = margin(r = 0.8 *
##             half_line/2), hjust = 1), axis.text.y.right = element_text(margin = margin(l = 0.8 *
##             half_line/2), hjust = 0), axis.ticks = element_line(colour = "grey20"),
##         axis.ticks.length = unit(half_line/2, "pt"), axis.ticks.length.x = NULL,
##         axis.ticks.length.x.top = NULL, axis.ticks.length.x.bottom = NULL,
##         axis.ticks.length.y = NULL, axis.ticks.length.y.left = NULL,
##         axis.ticks.length.y.right = NULL, axis.title.x = element_text(margin = margin(t = half_line/2),
##             vjust = 1), axis.title.x.top = element_text(margin = margin(b = half_line/2),
##             vjust = 0), axis.title.y = element_text(angle = 90,
##             margin = margin(r = half_line/2), vjust = 1), axis.title.y.right = element_text(angle = -90,
##             margin = margin(l = half_line/2), vjust = 0), legend.background = element_rect(colour = NA),
```

`theme()` Arguments

There are many elements you can customize. You can either group them by their type or by their category:

- element types:
  - `text` → all labels, axis text, legend title and text
  - `line` → axis lines, ticks, grid lines
  - `rect` → plot area, panel area, legend and legend keys, facets
- element category:
  - `axis.*` → titles, text, ticks, lines
  - `legend.*` → background, margin, spacing, keys, text, title, position, direction, box
  - `panel.*` → background, border, margin, spacing, grid (major and minor)
  - `plot.*` → background, title, subtitle, caption, tag, margin
  - `strip.*` → background, placement, text

Rectangular Elements via `element_rect()`

Rectangular Elements via `element_rect()`

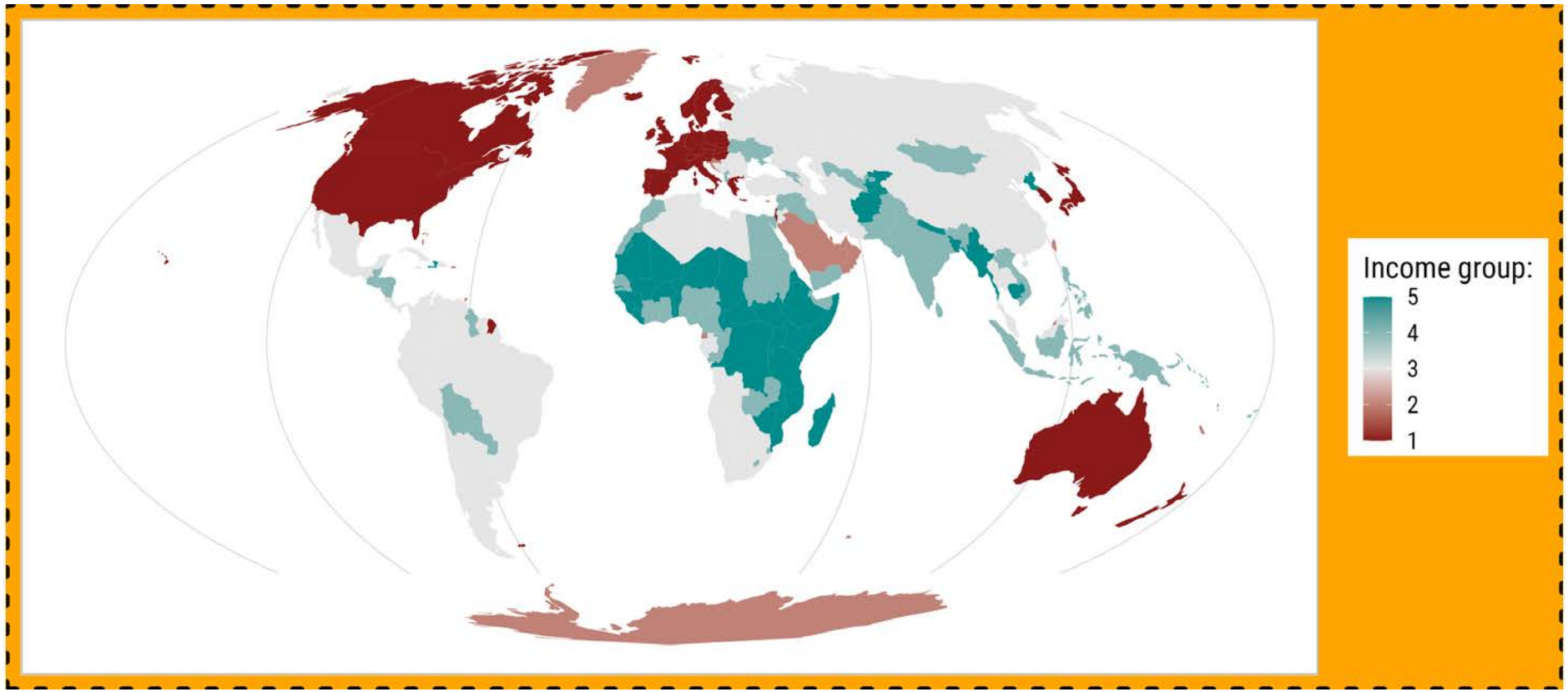
Rectangular Elements via `element_rect()`

```
my_rect_element <-
  element_rect(
    color = "black",
    fill = "orange",
    size = 2,
    linetype = "dotted"
  )
```

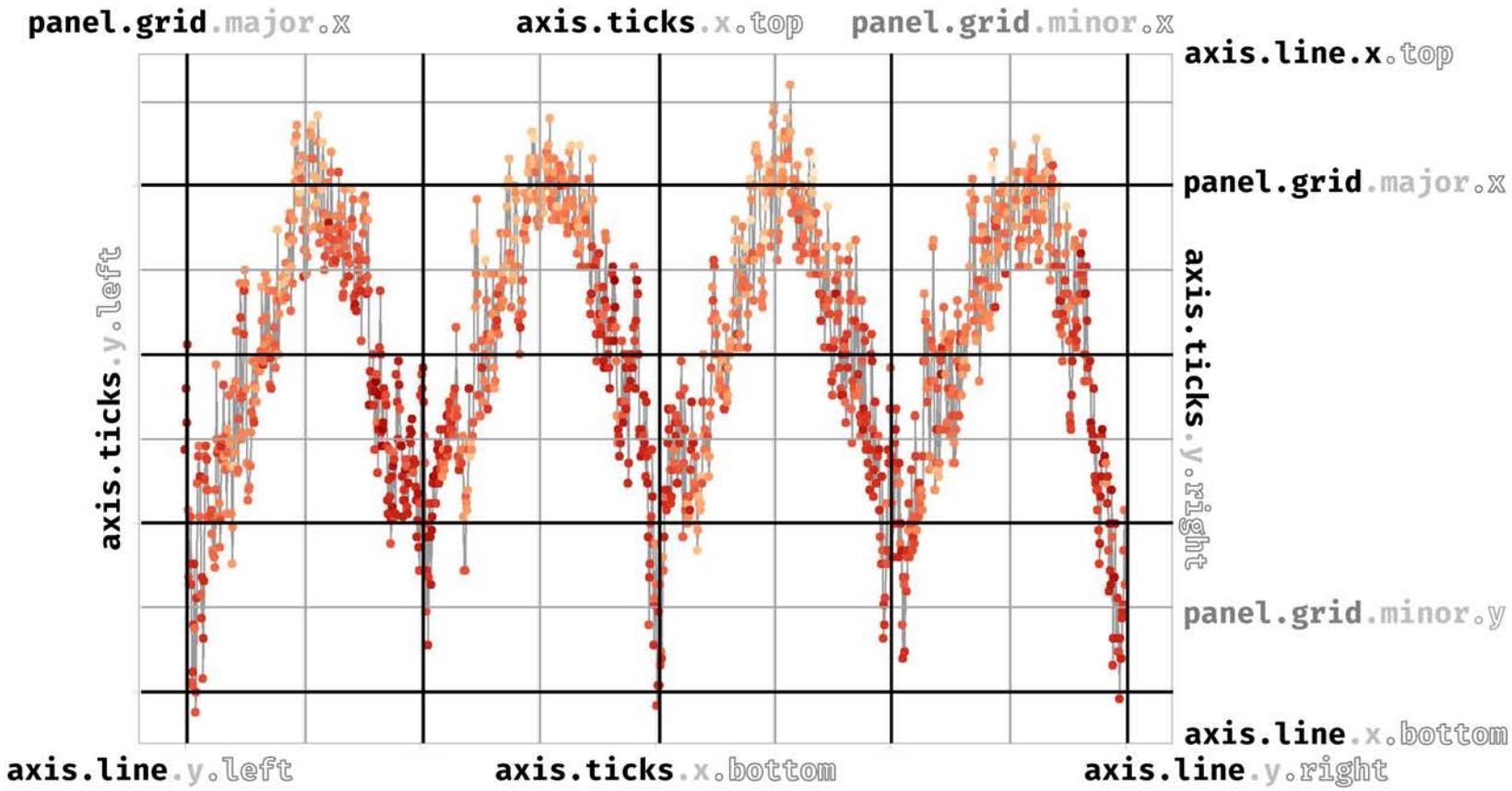
Rectangular Elements via `element_rect()`

You can directly alter the appearance by adding `theme()` to a ggplot:

```
g +
  theme(plot.background = my_rect_element)
```



Line Elements via `element_line()`



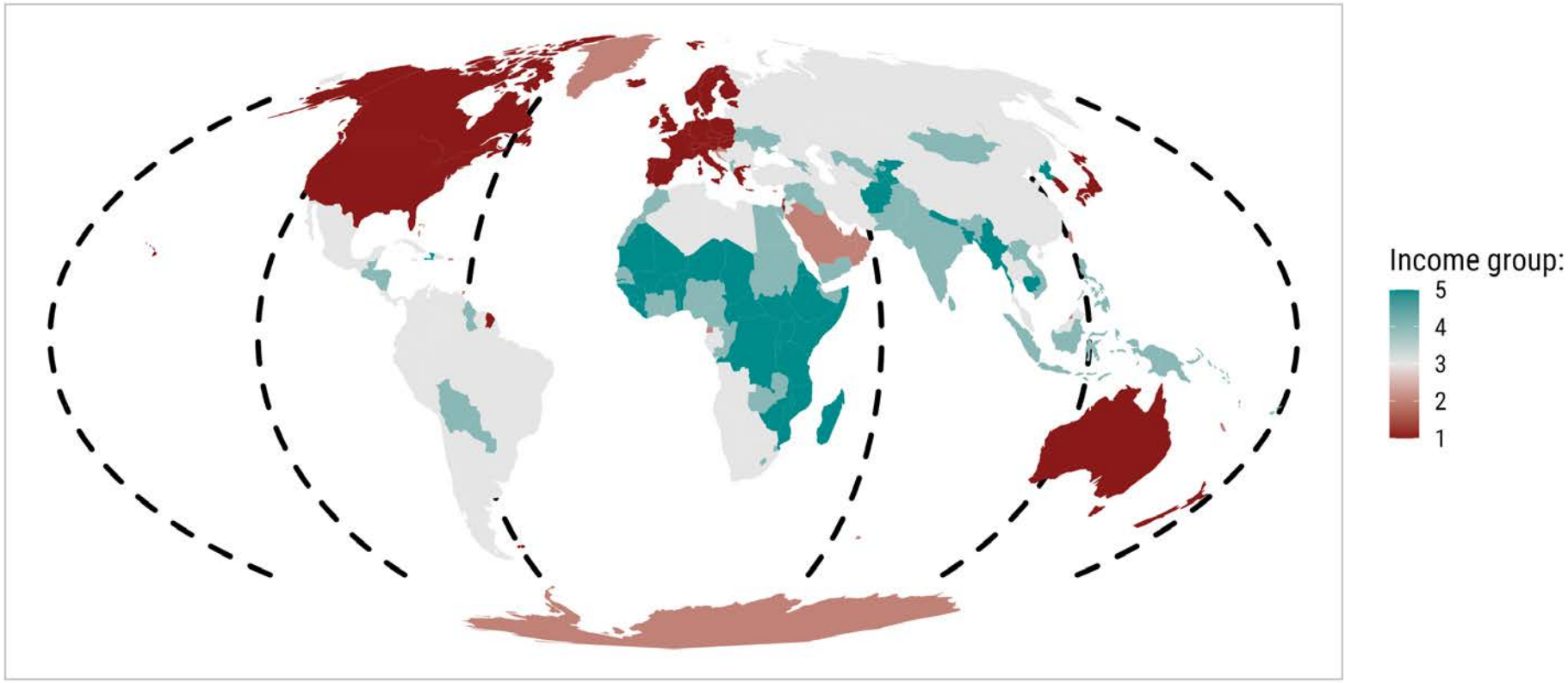
Line Elements via `element_line()`

```
my_line_element <-
  element_line(
    color = "black",
    size = 3,
    linetype = "dashed",
    lineend = "square", # round, butt
    arrow = arrow(
      angle = 30,
      length = unit(0.25, "inches")
    )
  )
```

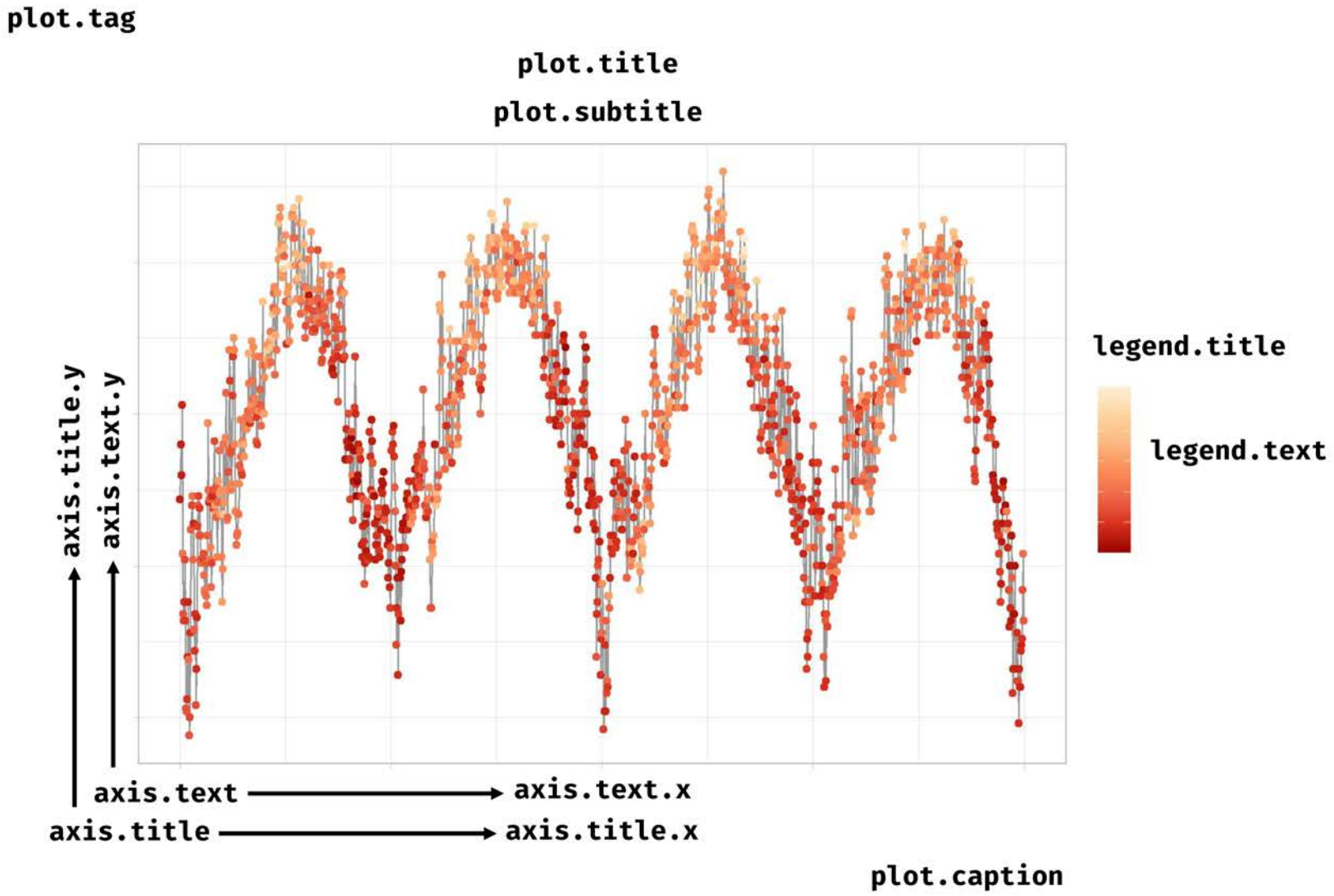
Line Elements via `element_line()`

You can directly alter the appearance by adding `theme()` to a ggplot:

```
g +
  theme(panel.grid = my_line_element)
```



Text Elements via `element_text()`



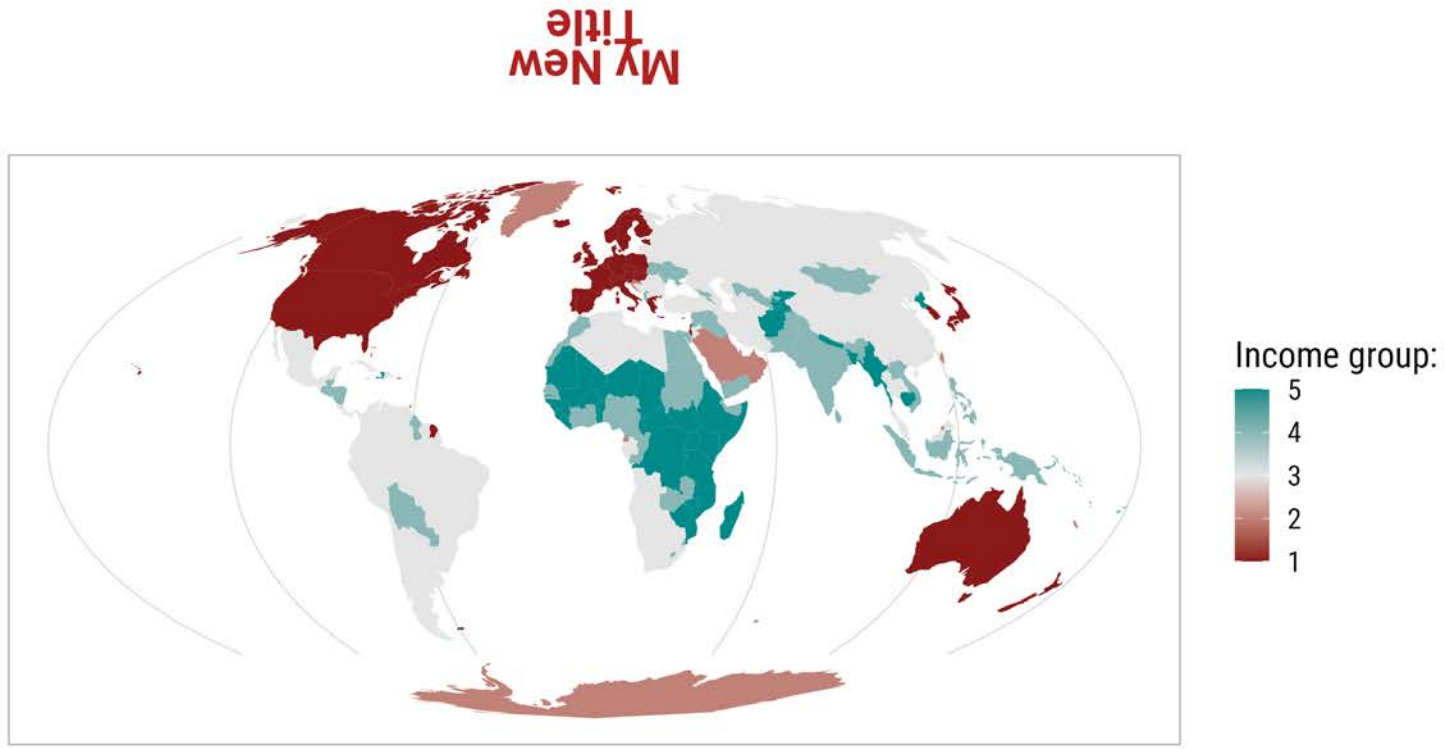
Text Elements via `element_text()`

```
my_text_element <-
  element_text(
    family = "Roboto", ## you need {systemfonts} for this
    face = "bold", ## plain, italic, bolditalic
    size = 24,
    color = "firebrick",
    lineheight = .7,
    angle = 180,
    hjust = .5,
    vjust = .0,
    margin = margin(
      10, ## t (top)
      0,  ## r (right)
      30, ## b (bottom)
      0   ## l (left)
    )
  )
```

Text Elements via `element_text()`

You can directly alter the appearance by adding `theme()` to a ggplot:

```
g +
  ggtitle("My New\nTitle") +
  theme(plot.title = my_text_element)
```



Exercise 3:

- Have a closer look at the codes of `theme_grey`/`theme_gray` and one of the other themes. What's the difference?
- Create a theme that is as ugly as possible!

  or:

  Create a theme that you can use ass personal theme.
- Apply it to a plot.

Exercise 3: theme_grey|gray()

```
theme_grey
## function (base_size = 11, base_family = "", base_line_size = base_size/22,
##     base_rect_size = base_size/22)
## {
##     half_line <- base_size/2
##     t <- theme(line = element_line(colour = "black", size = base_line_size,
##         linetype = 1, lineend = "butt"), rect = element_rect(fill = "white",
##         colour = "black", size = base_rect_size, linetype = 1),
##         text = element_text(family = base_family, face = "plain",
##             colour = "black", size = base_size, lineheight = 0.9,
##             hjust = 0.5, vjust = 0.5, angle = 0, margin = margin(),
##             debug = FALSE), axis.line = element_blank(), axis.line.x = NULL,
##         axis.line.y = NULL, axis.text = element_text(size = rel(0.8),
##             colour = "grey30"), axis.text.x = element_text(margin = margin(t = 0.8 *
##             half_line/2), vjust = 1), axis.text.x.top = element_text(margin = margin(b = 0.8 *
##             half_line/2), vjust = 0), axis.text.y = element_text(margin = margin(r = 0.8 *
##             half_line/2), hjust = 1), axis.text.y.right = element_text(margin = margin(l = 0.8 *
##             half_line/2), hjust = 0), axis.ticks = element_line(colour = "grey20"),
##         axis.ticks.length = unit(half_line/2, "pt"), axis.ticks.length.x = NULL,
##         axis.ticks.length.x.top = NULL, axis.ticks.length.x.bottom = NULL,
##         axis.ticks.length.y = NULL, axis.ticks.length.y.left = NULL,
##         axis.ticks.length.y.right = NULL, axis.title.x = element_text(margin = margin(t = half_line/2),
##             vjust = 1), axis.title.x.top = element_text(margin = margin(b = half_line/2),
```

Exercise 3: `theme_dark()`

```
theme_dark
## function (base_size = 11, base_family = "", base_line_size = base_size/22,
##     base_rect_size = base_size/22)
## {
##     half_line <- base_size/2
##     theme_grey(base_size = base_size, base_family = base_family,
##         base_line_size = base_line_size, base_rect_size = base_rect_size) %+replace%
##         theme(panel.background = element_rect(fill = "grey50",
##             colour = NA), panel.grid = element_line(colour = "grey42"),
##             panel.grid.major = element_line(size = rel(0.5)),
##             panel.grid.minor = element_line(size = rel(0.25)),
##             axis.ticks = element_line(colour = "grey20", size = rel(0.5)),
##             legend.key = element_rect(fill = "grey50", colour = NA),
##             strip.background = element_rect(fill = "grey15",
##                 colour = NA), strip.text = element_text(colour = "grey90",
##                 size = rel(0.8), margin = margin(0.8 * half_line,
##                   0.8 * half_line, 0.8 * half_line)),
##             complete = TRUE)
## }
## <bytecode: 0x000000002d518158>
## <environment: namespace:ggplot2>
```

Exercise 3: Create Your Own Theme!

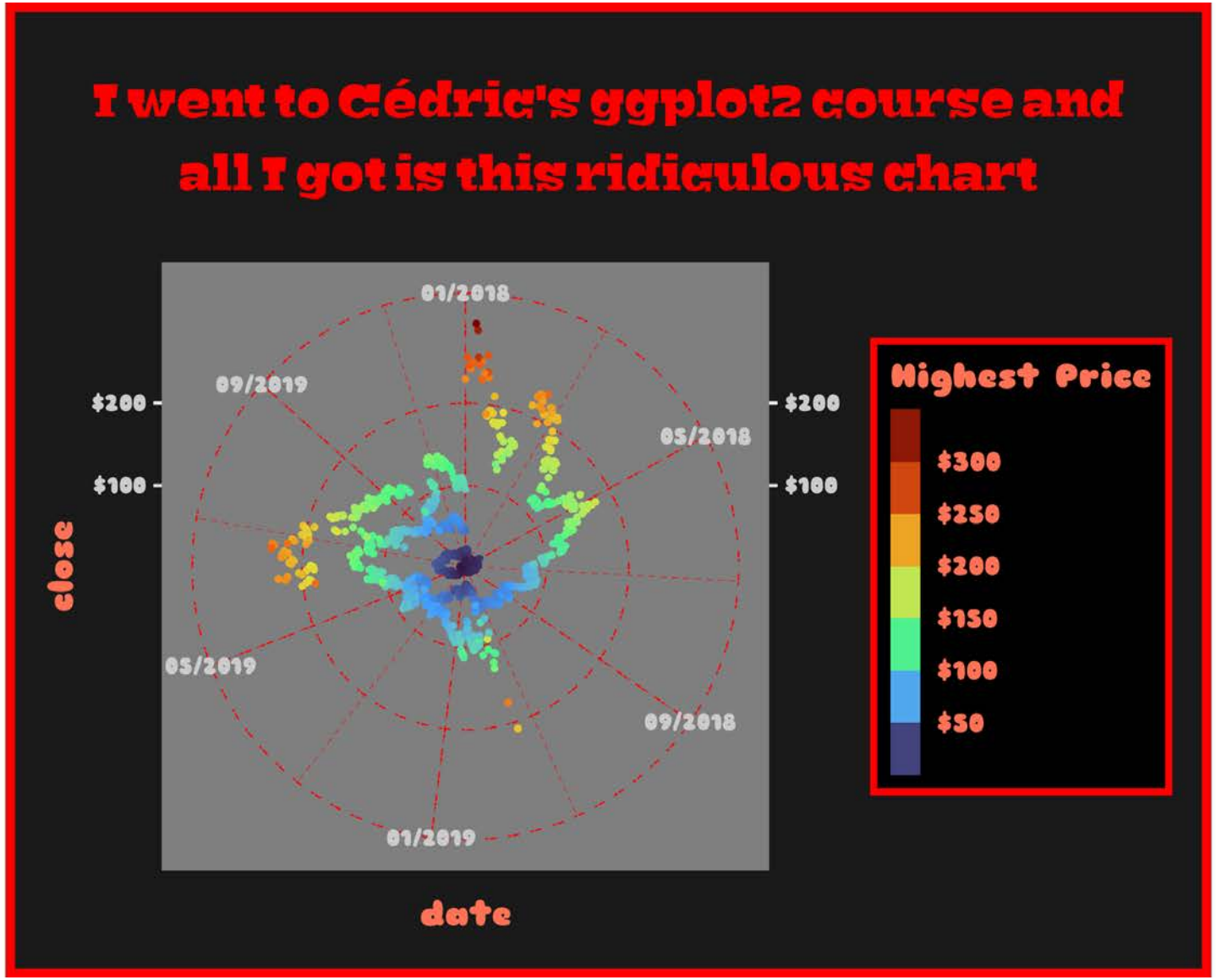
```
theme_ugly <-
  function(base_size = 14, ...) {
    theme_dark(base_size = base_size) %+replace%
        theme(
          text = element_text(color = "coral1", family = "Sniglet", size = base_size),
          axis.text = element_text(color = "grey80", size = rel(.7)),
          axis.ticks = element_line(color = "grey90"),
          plot.title = element_text(
            family = "Bigfish Black",
            color = "red", size = rel(1.5), lineheight = 1.2,
            margin = margin(base_size, 0, base_size * 2, 0)
          ),
          plot.title.position = "plot",
          panel.grid = element_line(color = "red", linetype = "4127"),
          plot.background = element_rect(fill = "grey10", color = "red", size = 5),
          plot.margin = margin(rep(25, 4)),
          legend.background = element_rect(fill = "black", color = "red", size = 2)
        )
}
```

Exercise 3: Create Your Own Theme!

```
theme_ugly <-
  function(base_size = 14, ...) {
    theme_dark(base_size = base_size) %+replace%
        theme(
          text = element_text(color = "coral1", family = "Sniglet", size = base_size),
          axis.text = element_text(color = "grey80", size = rel(.7)),
          axis.ticks = element_line(color = "grey90"),
          plot.title = element_text(
            family = "Bigfish Black",
            color = "red", size = rel(1.5), lineheight = 1.2,
            margin = margin(base_size, 0, base_size * 2, 0)
          ),
          plot.title.position = "plot",
          panel.grid = element_line(color = "red", linetype = "4127"),
          plot.background = element_rect(fill = "grey10", color = "red", size = 5),
          plot.margin = margin(rep(25, 4)),
          legend.background = element_rect(fill = "black", color = "red", size = 2)
        )
}
```

Exercise 3: Create Your Own Theme!

```
theme_ugly <-
  function(base_size = 14, ...) {
    theme_dark(base_size = base_size) %+replace%
        theme(
          text = element_text(color = "coral1", family = "Sniglet", size = base_size),
          axis.text = element_text(color = "grey80", size = rel(.7)),
          axis.ticks = element_line(color = "grey90"),
          plot.title = element_text(
            family = "Bigfish Black",
            color = "red", size = rel(1.5), lineheight = 1.2,
            margin = margin(base_size, 0, base_size * 2, 0)
          ),
          plot.title.position = "plot",
          panel.grid = element_line(color = "red", linetype = "4127"),
          plot.background = element_rect(fill = "grey10", color = "red", size = 5),
          plot.margin = margin(rep(25, 4)),
          legend.background = element_rect(fill = "black", color = "red", size = 2)
        )
}
```


I went to Cédric's ggplot2 course and all I got is this ridiculous chart
01/2018
09/2019
05/2018
05/2019
09/2018
01/2019
$200
$100
$200
$100
close
date
Highest Price
$300
$250
$200
$150
$100
$50

Exercise 3: Apply Your Own Theme!

```
ggplot(data, aes(x = date, y = close, color = high)) +
  geom_point() +
  scale_x_date(expand = c(0, 0), date_breaks = "4 months", date_labels = "%m/%Y") +
  scale_y_continuous(labels = scales::dollar_format(),
                     sec.axis = dup_axis(name = NULL)) +
  scale_color_viridis_c(option = "turbo", breaks = seq(50, 350, by = 50),
                        labels = scales::dollar_format(), name = "Highest Price") +
  guides(color = guide_colorsteps(barheight = unit(15, "lines"))) +
  coord_polar() +
  ggtitle("I went to Cédric's ggplot2 course and\nall I got is this ridiculous chart") +
  theme_ugly(base_size = 20)
```

Resources

- `Modify theme components` on the ggplot reference page
- Chapter 19 `Themes` of the “ggplot2” book by Hadley Wickham et al.
- `“Creating and Using Custom ggplot2 Themes”`, blog post by Thomas Mock
- `“Themes to Improve Your ggplot Figures”` from `R for the Rest of Us` with a collection of additional themes
- `“A {ggplot2} Tutorial for Beautiful Plotting in R”`, my extensive "how to"-tutorial