Reporting data results #1

Plot guidelines

Guidelines for good plots

There are a number of very thoughtful books and articles about creating graphics that effectively communicate information.

Some of the authors I highly recommend (and from whose work I've pulled the guidelines for good graphics we'll talk about this week) are:

- Edward Tufte
- Howard Wainer
- Stephen Few
- Nathan Yau

You should plan, in particular, to read *The Visual Display of Quantitative Information* by Edward Tufte before you graduate.

Guidelines for good plots

This week, we'll focus on six guidelines for good graphics, based on the writings of these and other specialists in data display.

The guidelines are:

1. Aim for high data density.
2. Use clear, meaningful labels.
3. Provide useful references.
4. Highlight interesting aspects of the data.
5. Make order meaningful.
6. When possible, use small multiples.

Packages for examples

For the examples, I'll use `dplyr` for data cleaning and, for plotting, the packages `ggplot2`, `gridExtra`, and `ggthemes`.

```
library(dplyr)

library(ggplot2)
library(gridExtra)
library(ggthemes)
```

Example data

You can load the data for today's examples with the following code:

```
library(faraway)
data(nepali)
data(worldcup)

library(dlnm)
data(chicagoNMMAPS)
chic <- chicagoNMMAPS
chic_july <- chic %>%
  filter(month == 7 & year == 1995)
```

High data density

Guideline 1: **Aim for high data density.**

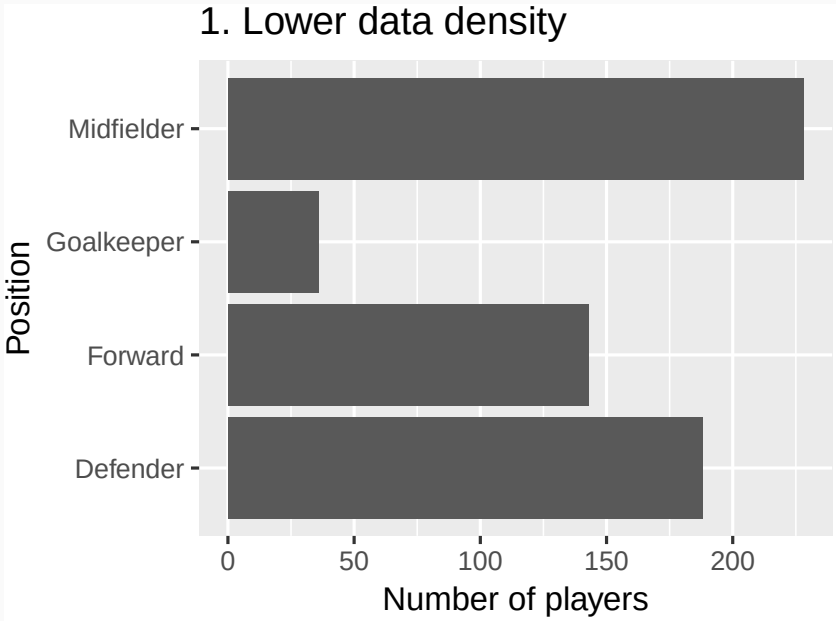
You should try to increase, as much as possible, the **data to ink ratio** in your graphs. This is the ratio of "ink" providing information to all ink used in the figure.

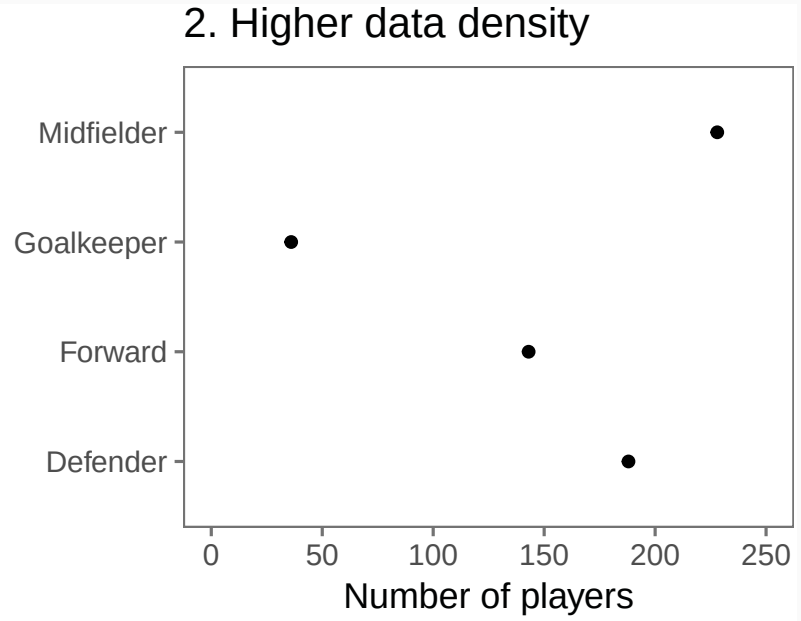
One way to think about this is that the only graphs you make that use up a lot of your printer's ink should be packed with information.

High data density

Guideline 1: **Aim for high data density.**

The two graphs below show the same information. Compare the amount of ink used in the left plot to the amount used in the right plot to see how graphs with the same information can have very different data densities.

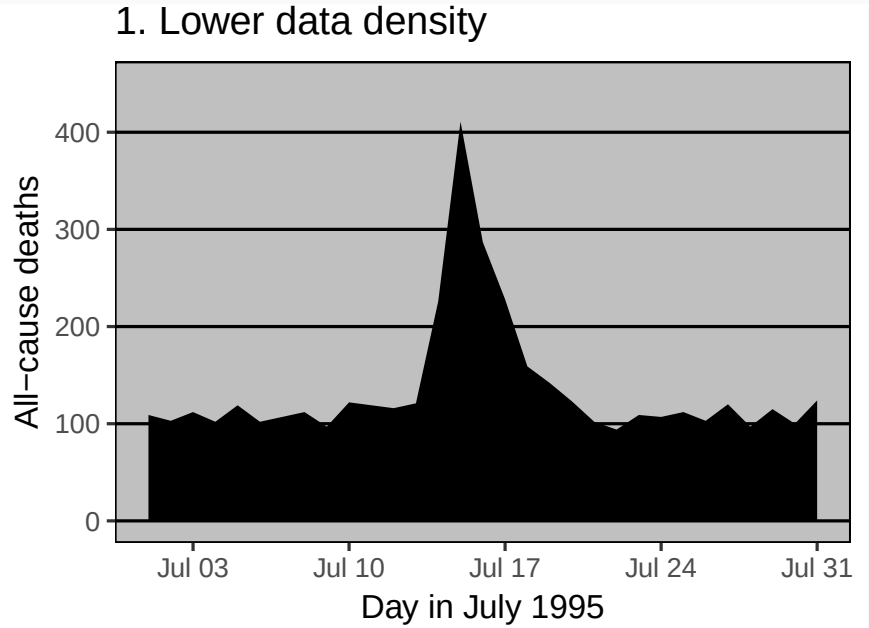


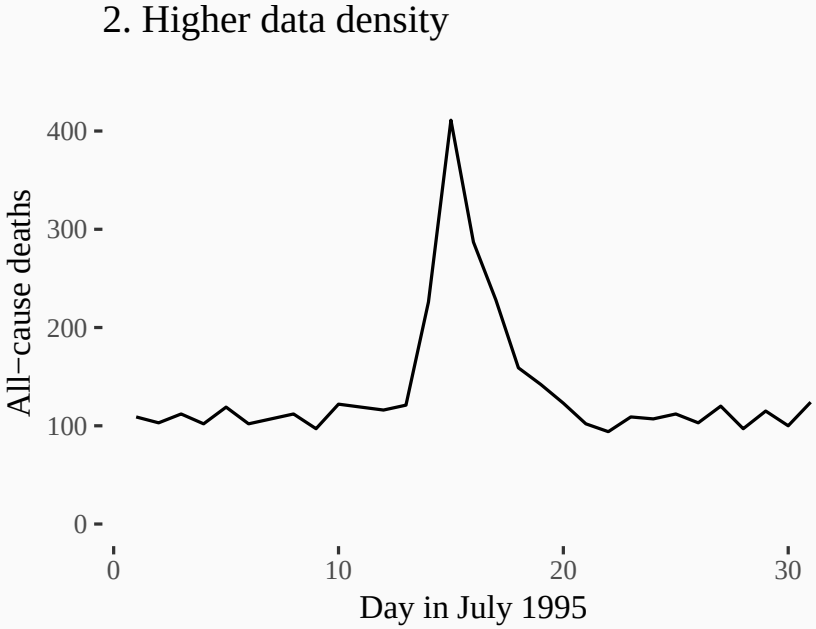


High data density

Guideline 1: **Aim for high data density.**

The two graphs below show another example of very different data densities in two plots showing the same information:





Data density

One quick way to increase data density in `ggplot2` is to change the *theme* for the plot. This essentially changes the "background" elements to a plot, including elements like the plot grid, background color, and the font used for labeling.

Some themes come with `ggplot2`, including:

- `theme_classic`
- `theme_bw`
- `theme_minimal`
- `theme_void`

The `ggthemes` packages has some excellent additional themes.

Data density

The following slides show some examples of the effects of using different themes. The following code creates a plot of daily deaths in Chicago in July 1995:

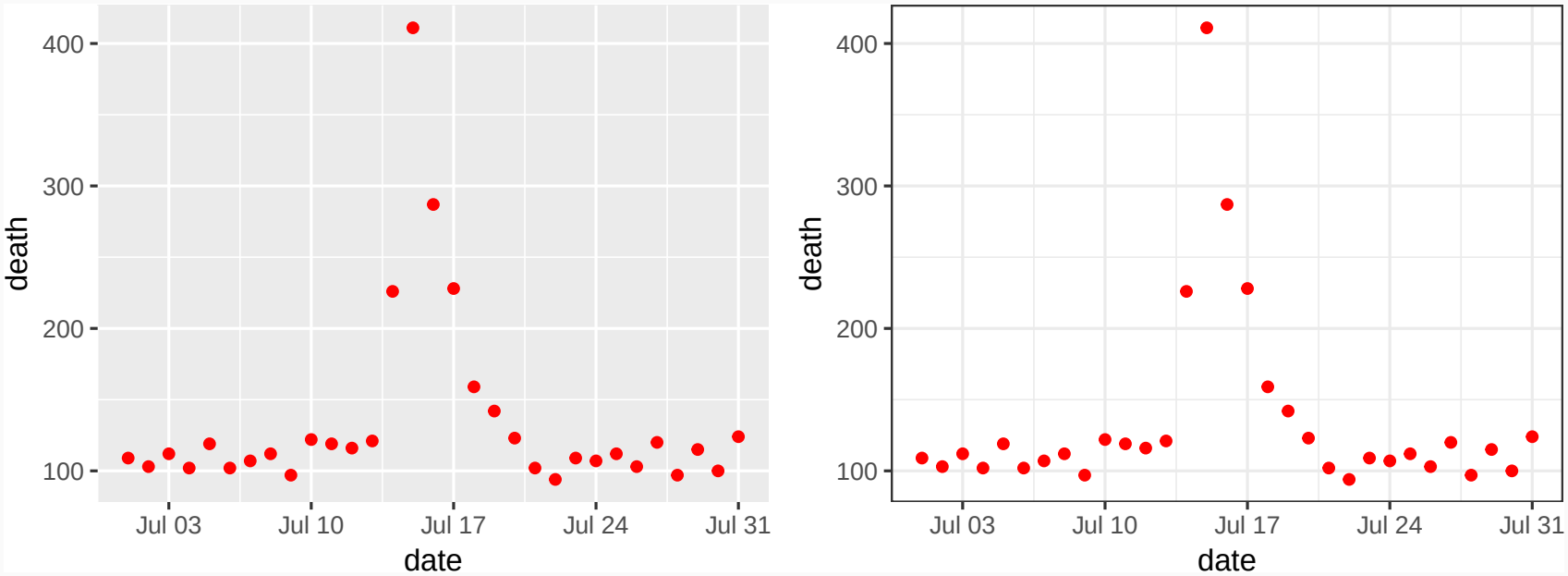
```
chic_plot <- ggplot(chic_july, aes(x = date, y = death))  +
        geom_point(color = "red")
```

Next, we can see how the graph looks with the default theme and with other themes.

Themes

The left graph shows the graph with the default theme, while the right shows the effect of adding the black-and-white theme that comes with ggplot2 as `theme_bw`:

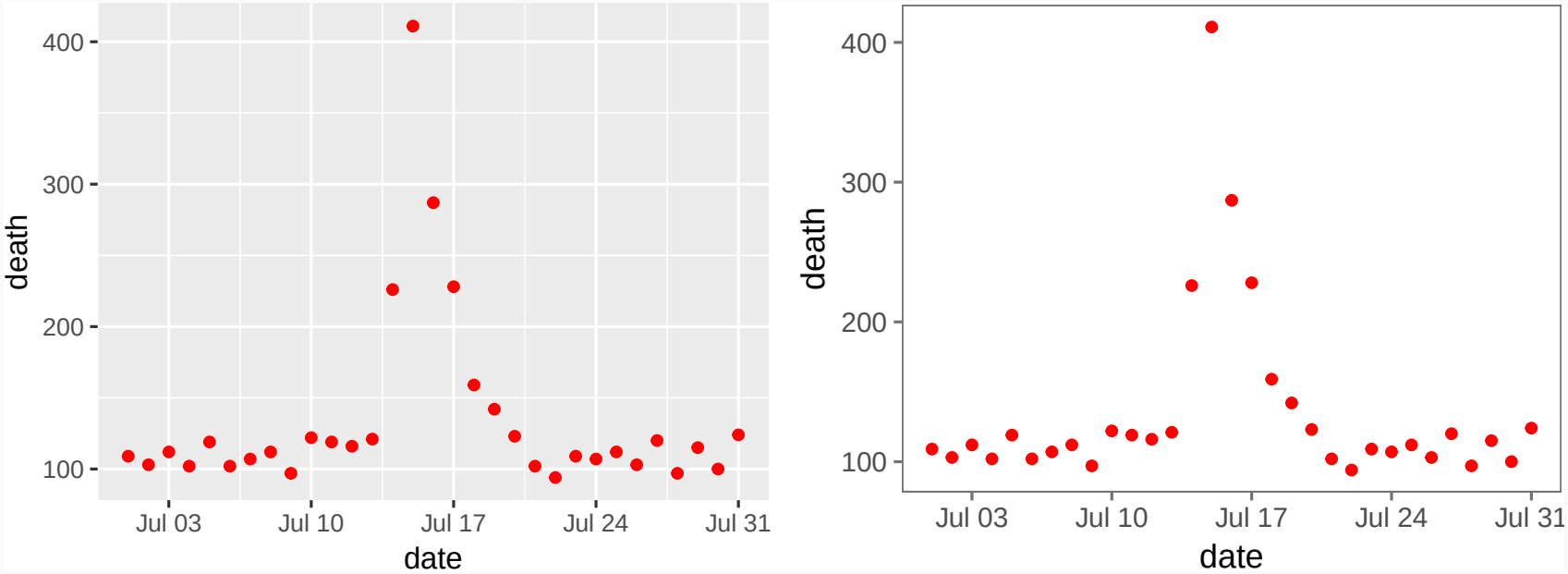
```
a <- chic_plot
b <- chic_plot + theme_bw()
grid.arrange(a, b, ncol = 2)
```



Themes

Stephen Few theme:

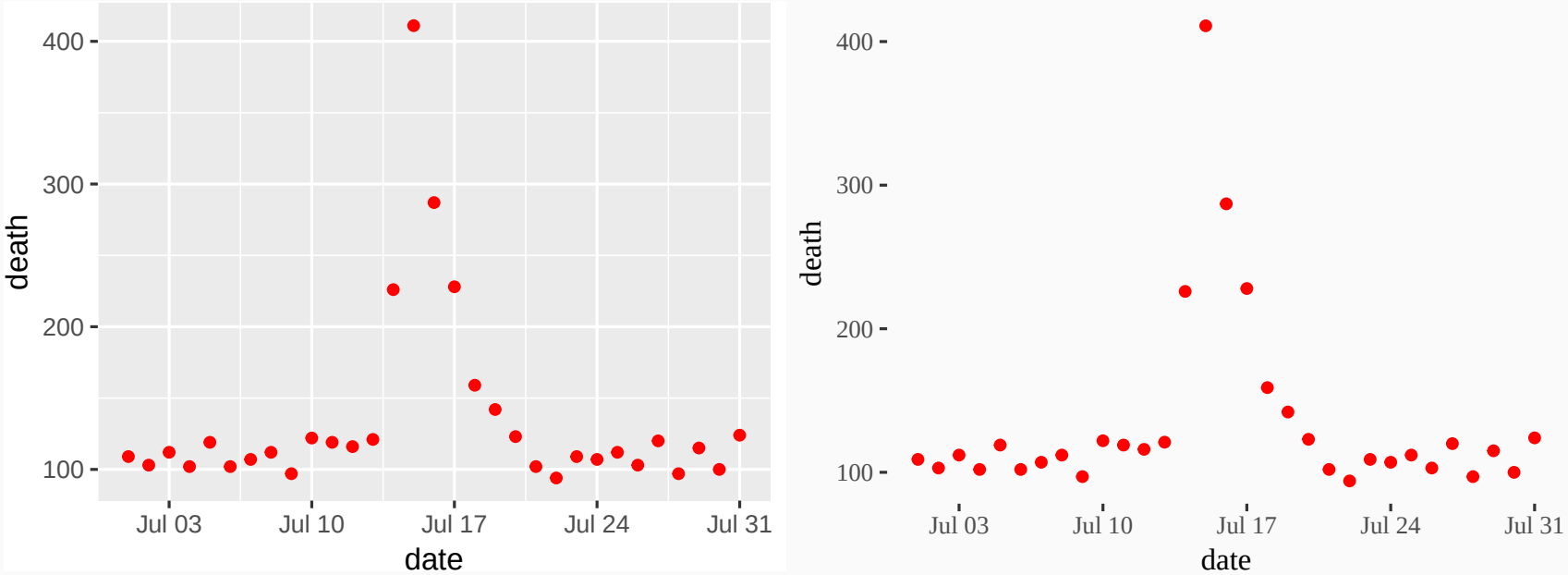
```
a <- chic_plot
b <- chic_plot + theme_few()
grid.arrange(a, b, ncol = 2)
```



Themes

Edward Tufte theme:

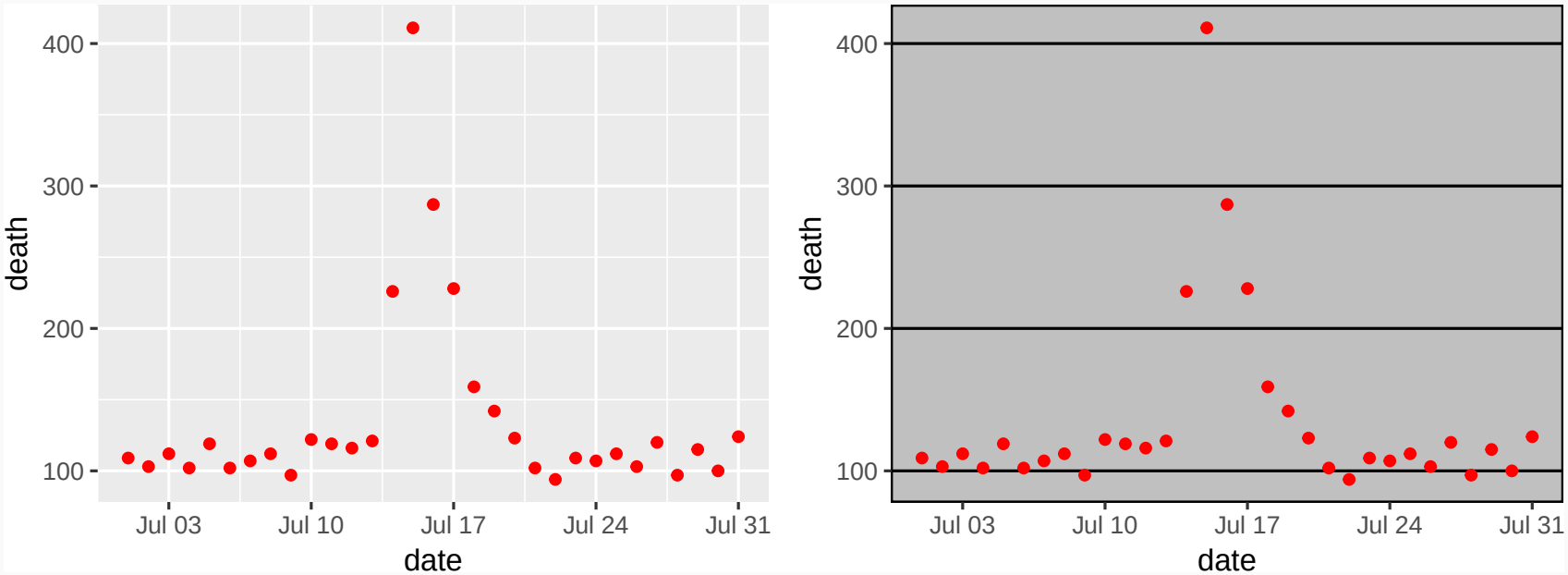
```
a <- chic_plot
b <- chic_plot + theme_tufte()
grid.arrange(a, b, ncol = 2)
```



Themes

You can even use themes to add some questionable choices for different elements. For example, `ggthemes` includes an Excel theme:

```
a <- chic_plot
b <- chic_plot + theme_excel()
grid.arrange(a, b, ncol = 2)
```

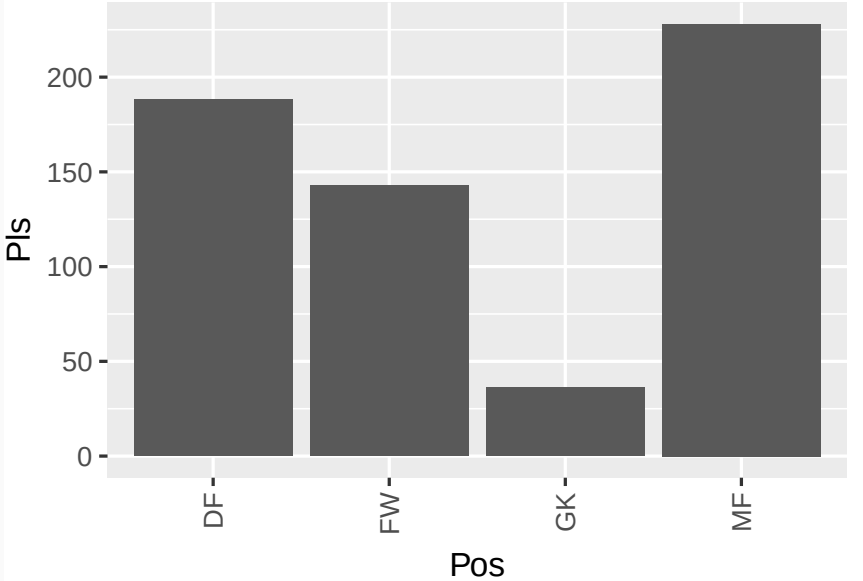


Meaningful labels

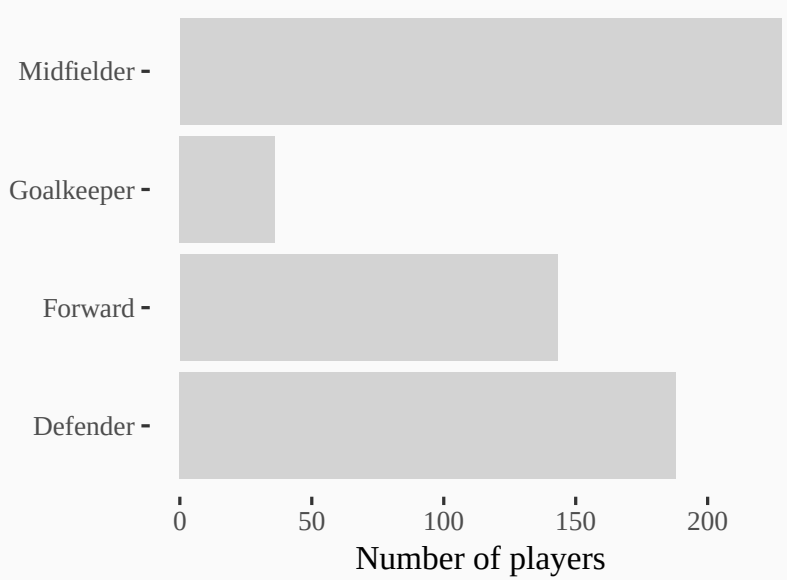
Guideline 2: **Use clear, meaningful labels.**

Graph defaults often use abbreviations for axis labels and other labeling. Further, text labels can sometimes be aligned in a way that makes them hard to read. The plots below give an example of the same information shown without (left) and with (right) clear, meaningful labels.

1. Murkier labels



2. Clearer labels



Meaningful labels

There are a few strategies you can use to make labels clearer:

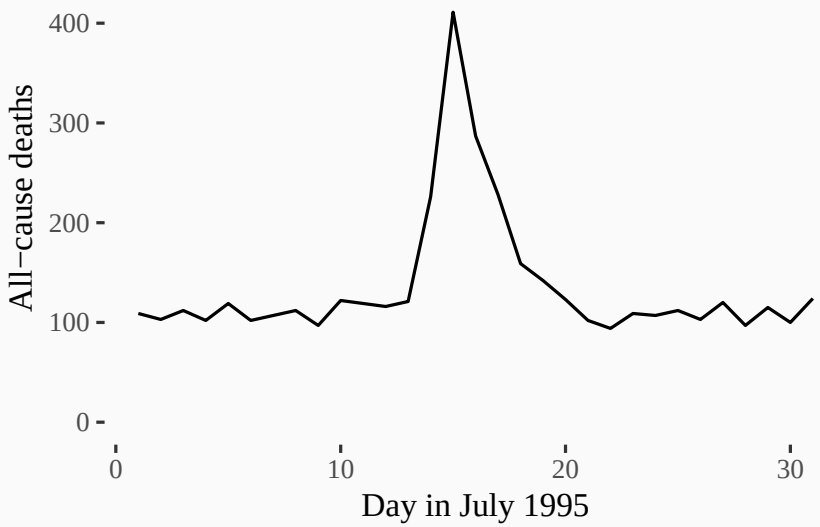
- Add a `labs()` element to the plot, rather than relying on the column names in the original data. This can also be done with `scale` elements (e.g., `scale_x_continuous`), which give you more power to also make other changes to these scales.
- Include units of measurement in axis titles when relevant. If units are dollars or percent, check out the `scales` package, which allows you to add labels directly to axis elements by including arguments like `labels = percent` in `scale` elements.
- If the x-variable requires longer labels (player positions in the example above), consider flipping the coordinates, rather than abbreviating or rotating the labels. You can use `coord_flip` to do this.

References

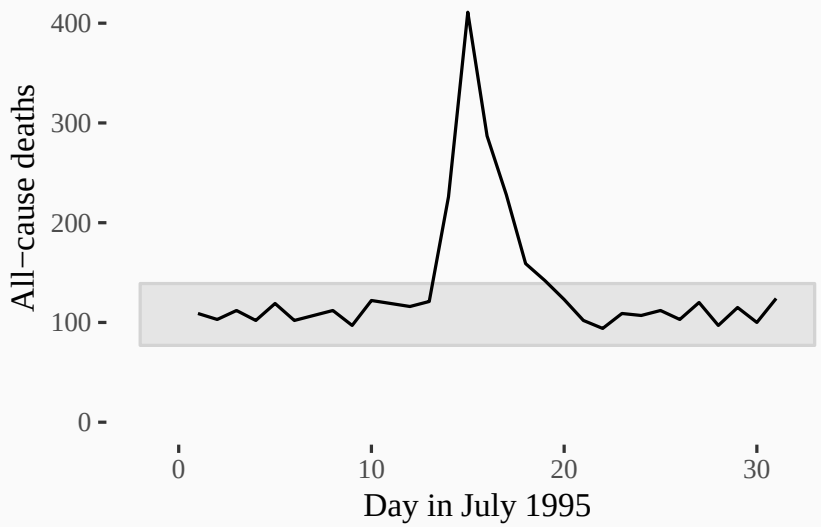
Guideline 3: **Provide useful references.**

Data is easier to interpret when you add references. For example, if you show what it typical, it helps viewers interpret how unusual outliers are. The graph below on the right has added shading showing the range of daily deaths in July in Chicago for 1990–1994 and 1996–2000, to clarify how unusual July 1995 was.

1. No reference



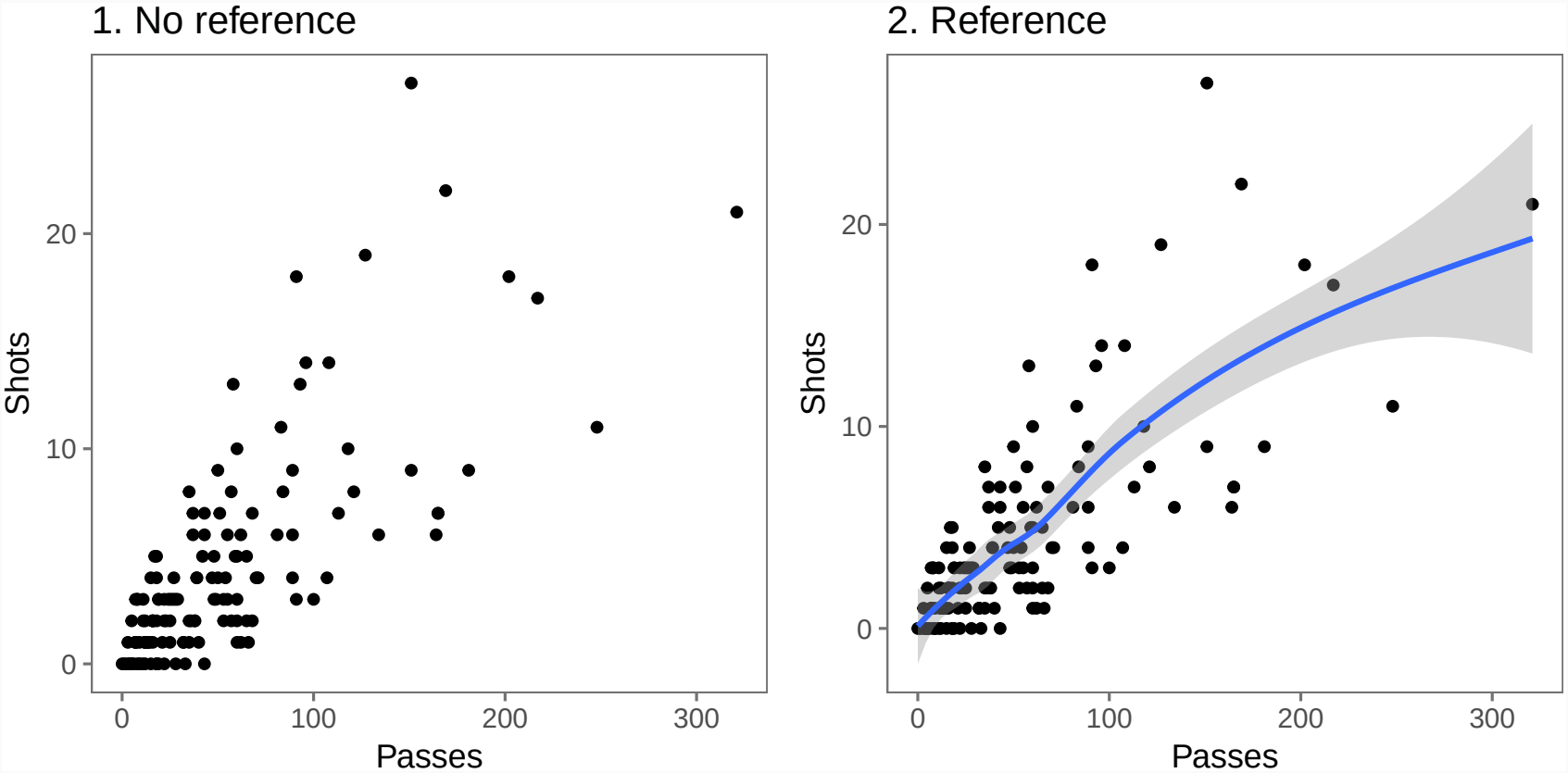
2. Reference



References

Guideline 3: **Provide useful references.**

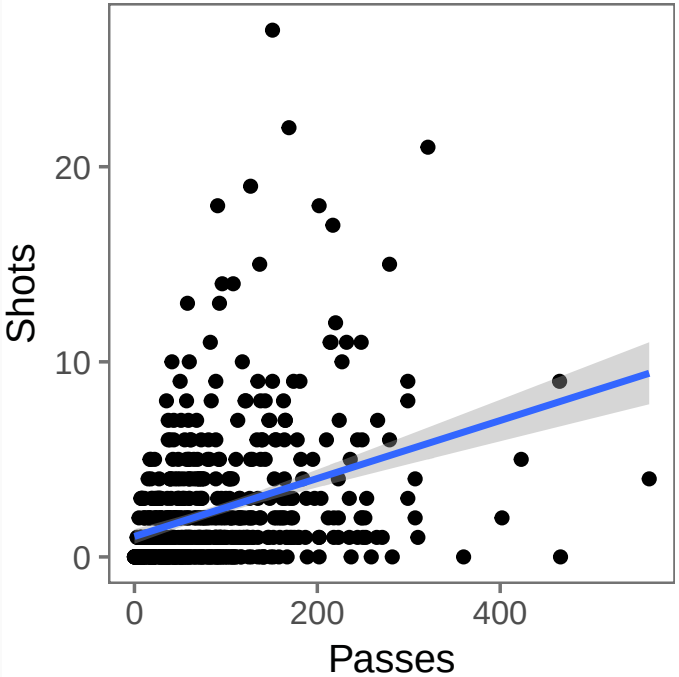
Another useful way to add references is to add a linear or smooth fit to the data, to help clarify trends in the data.



References

You can use the function `geom_smooth` to add a smooth or linear reference line:

```
ggplot(worldcup, aes(x = Passes, y = Shots)) +
  geom_point() + theme_few() +
  geom_smooth(method = "lm")
```



References

The most useful `geom_smooth` parameters to know are:

- `method`: The default is to add a loess curve if the data includes less than 1000 points and a generalized additive model for 1000 points or more. However, you can change to show the fitted line from a linear model using `method = "lm"` or from a generalized linear model using `method = "glm"`.
- `span`: How wiggly or smooth the smooth line should be (smaller value: more wiggly; larger value: more smooth)
- `se`: TRUE or FALSE, indicating whether to include shading for 95% confidence intervals.
- `level`: Confidence level for confidence interval (e.g., `0.90` for 90% confidence intervals)

References

Lines and polygons can also be useful for adding references. Useful geoms include:

- `geom_hline`, `geom_vline`: Add a horizontal or vertical line
- `geom_abline`: Add a line with an intercept and slope
- `geom_polygon`: Add a filled polygon
- `geom_path`: Add an unfilled polygon

References

When adding these references:

- Add reference elements first, so they will be plotted under the data, instead of on top of it.
- Use `alpha` to add transparency to these elements.
- Use colors that are unobtrusive (e.g., grays)
- For lines, consider using non-solid line types (e.g., `linetype = 3`)

In-course exercise

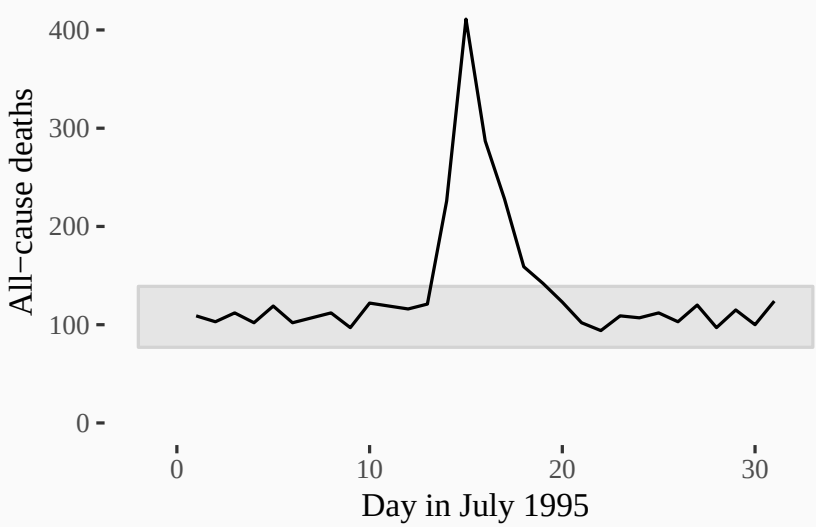
We'll take a break now to do Sections 4.10.1 and 4.10.2 of the In-Course Exercise.

Highlighting

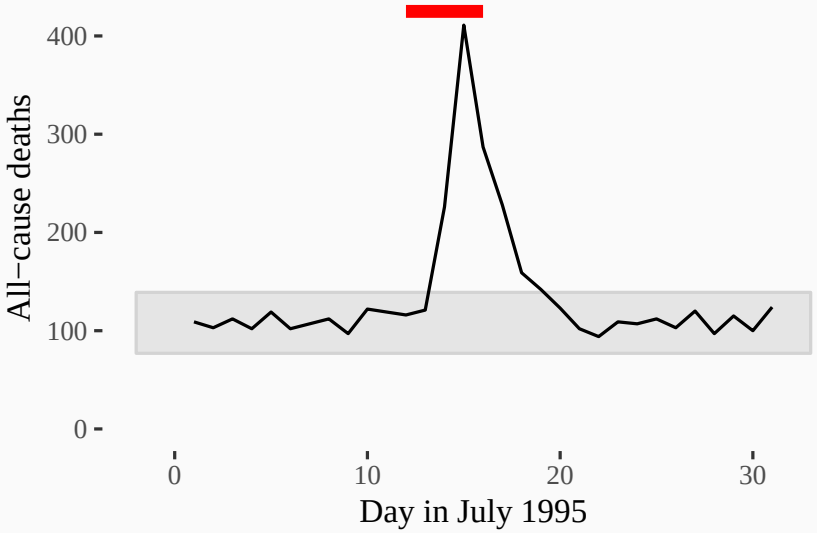
Guideline 3: **Highlight interesting aspects.**

Consider adding elements to highlight noteworthy elements of the data. For example, in the graph on the right, the days of a major heat wave have been highlighted with a red line.

1. No highlighting



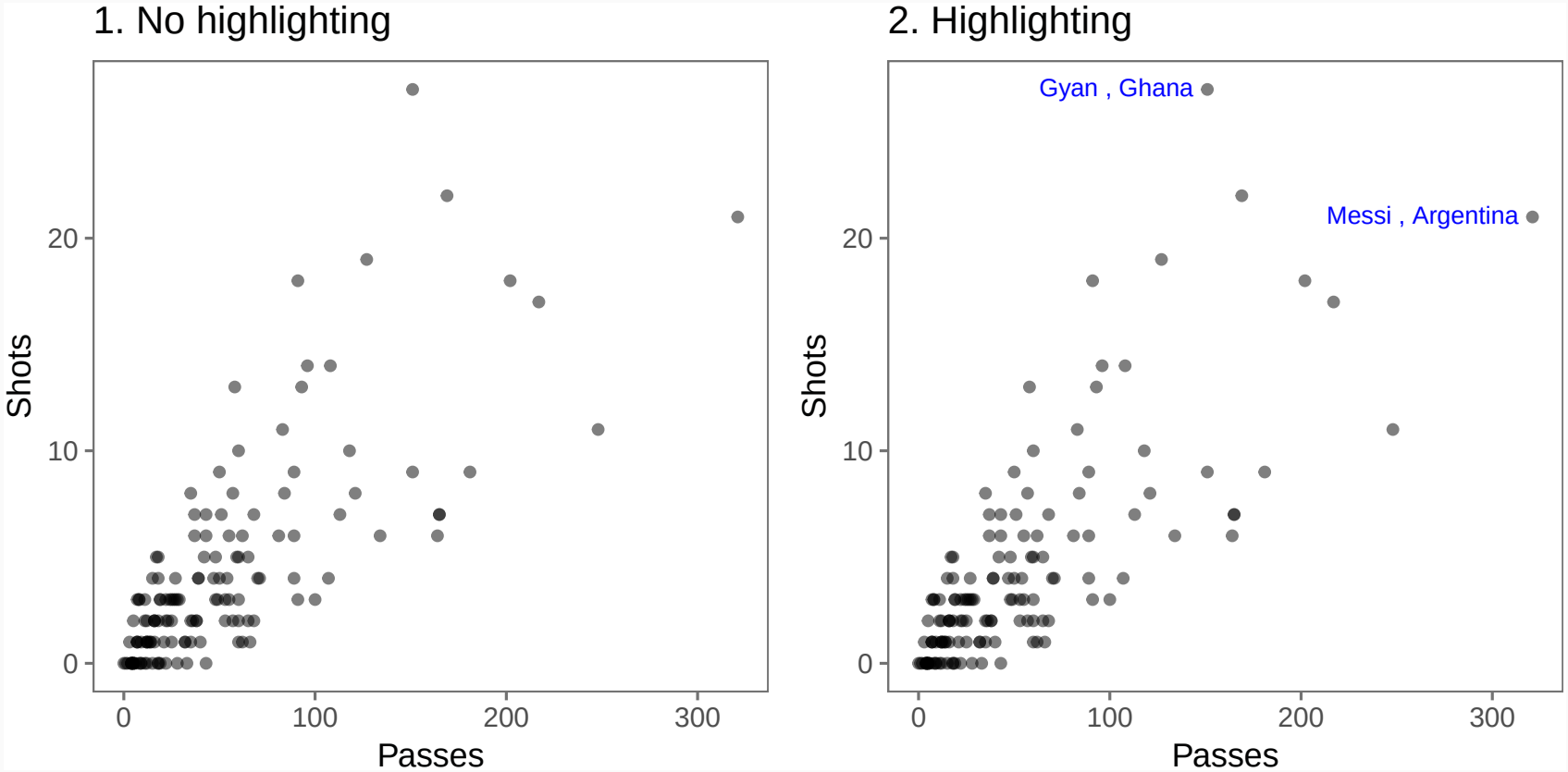
2. With highlighting



Highlighting

Guideline 3: **Highlight interesting aspects.**

In the below graphs, the names of the players with the most shots and passes have been added to highlight these unusual points.



Highlighting

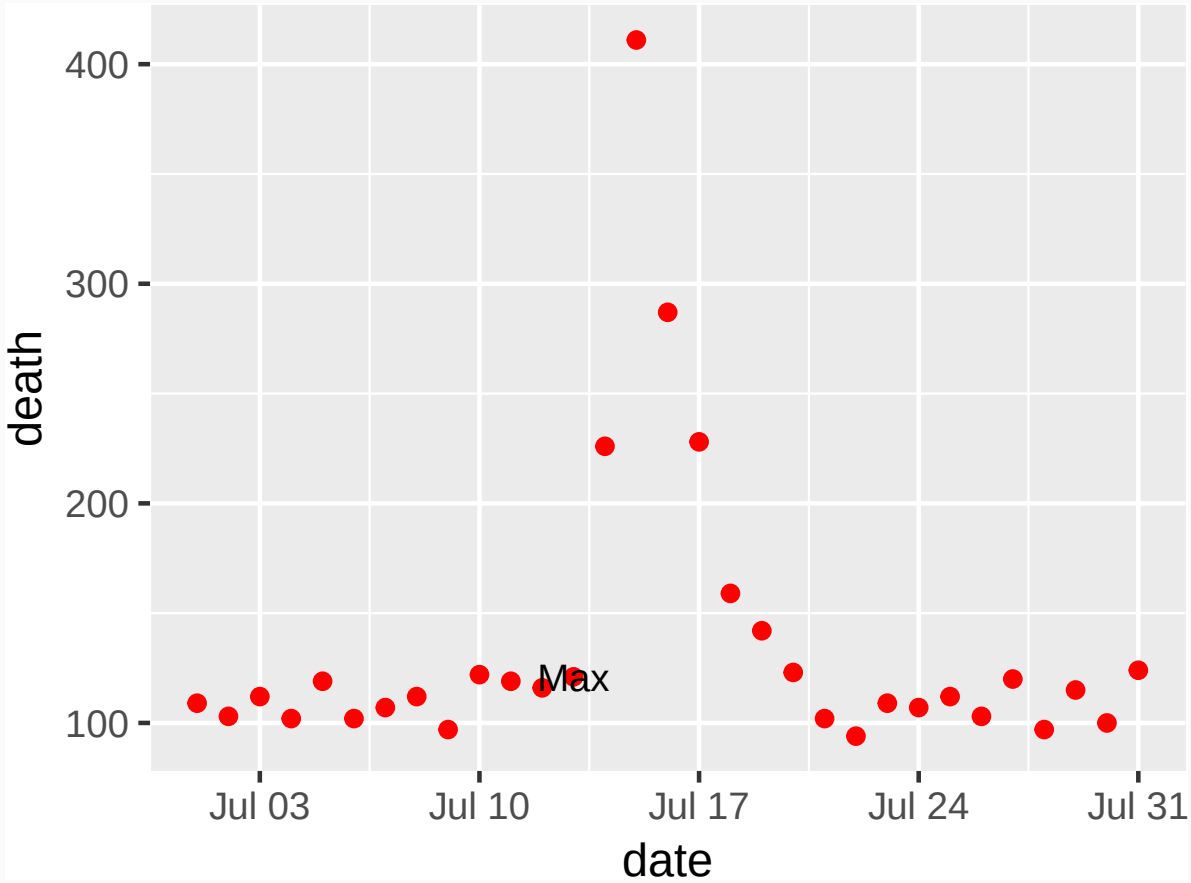
One helpful way to annotate is with text, using `geom_text()`. For this, you'll first need to create a dataframe with the hottest day in the data:

```
hottest_day <- chic_july %>%
  filter(temp == max(temp))
hottest_day[ , 1:6]
```

```
##         date time year month doy       dow
## 1 1995-07-13 3116 1995     7 194 Thursday
```

Highlighting

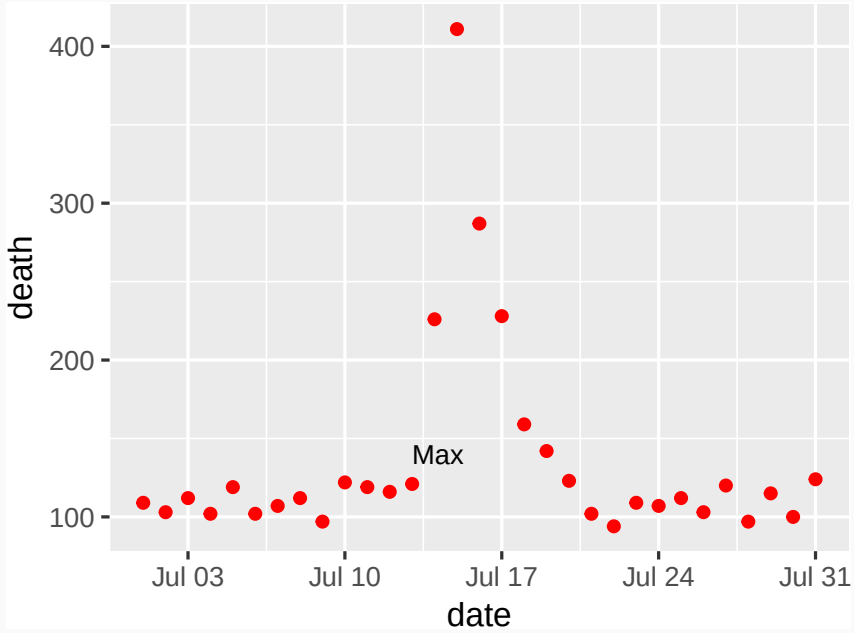
```
chic_plot + geom_text(data = hottest_day,
                      label = "Max",
                      size = 3)
```



Highlighting

With `geom_text`, you'll often want to use position adjustment (the `position` parameter) to move the text so it won't be right on top of the data points:

```
chic_plot + geom_text(data = hottest_day,
                      label = "Max",
                      size = 3, hjust = 0, vjust = -1)
```



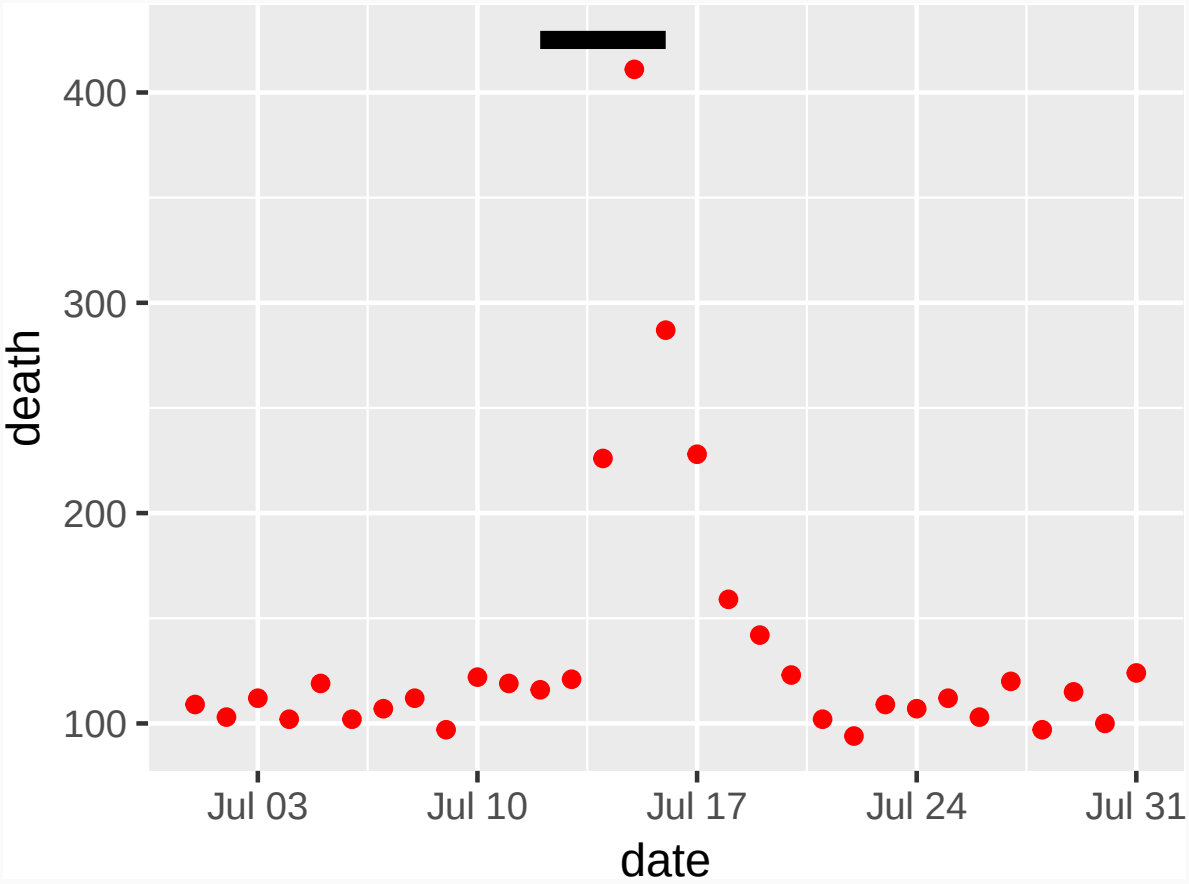
Highlighting

You can also use lines to highlight. For this, it is often useful to create a new dataframe with data for the reference. To add a line for the Chicago heat wave, I've added a dataframe called `hw` with the relevant date range. I'm setting the y-value to be high enough (425) to ensure the line will be placed above the mortality data.

```
hw <- data.frame(date = c(as.Date("1995-07-12"),
                          as.Date("1995-07-16")),
                 death = c(425, 425))

b <- chic_plot +
       geom_line(data = hw,
                 aes(x = date, y = death),
                 size = 2)
```

b



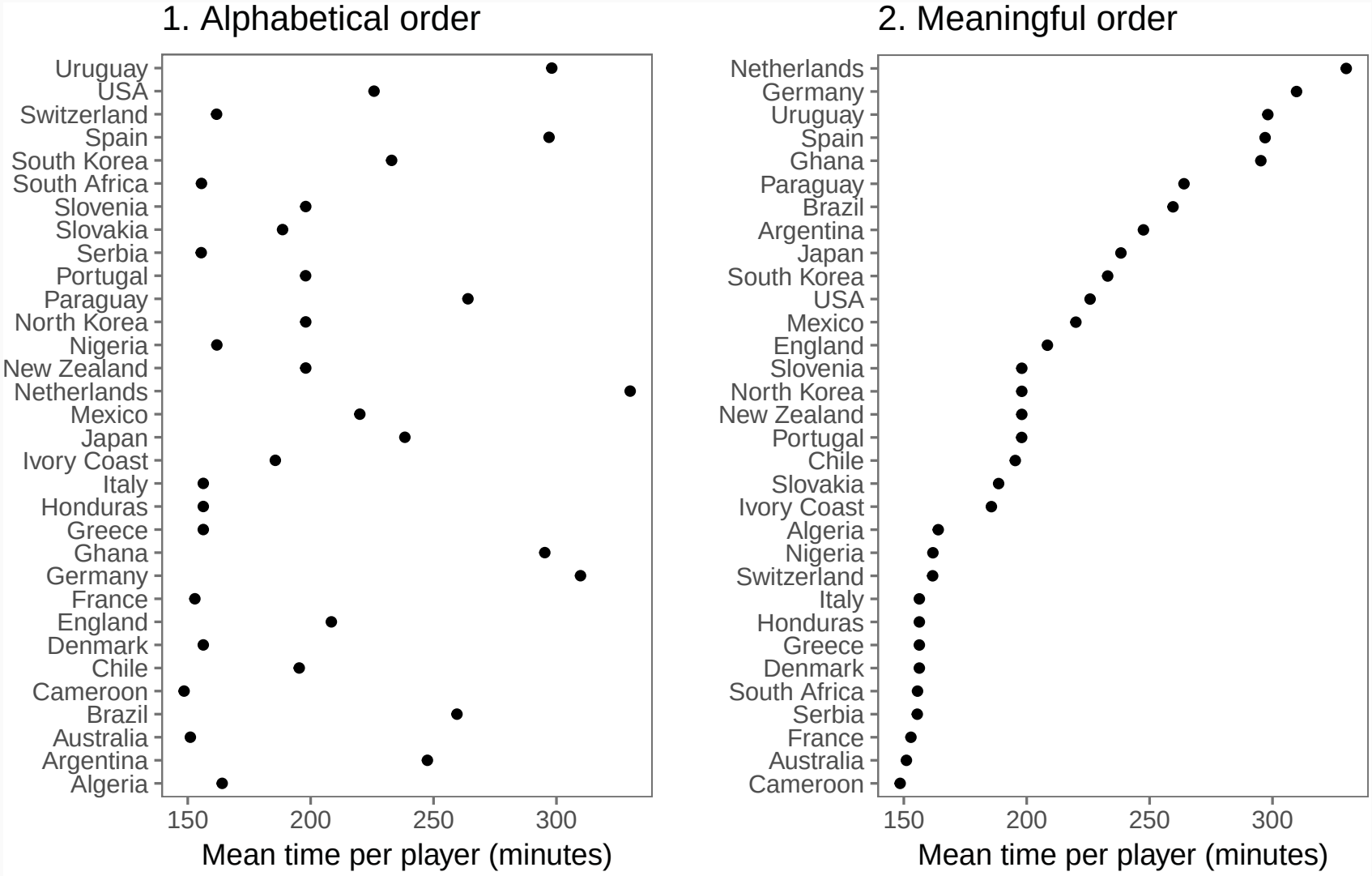
Order

Guideline 4: **Make order meaningful.**

You can make the ranking of data clearer from a graph by using order to show rank. Often, factor or categorical variables are ordered by something that is not interesting, like alphabetical order.

Order

Guideline 4: **Make order meaningful.**



Order

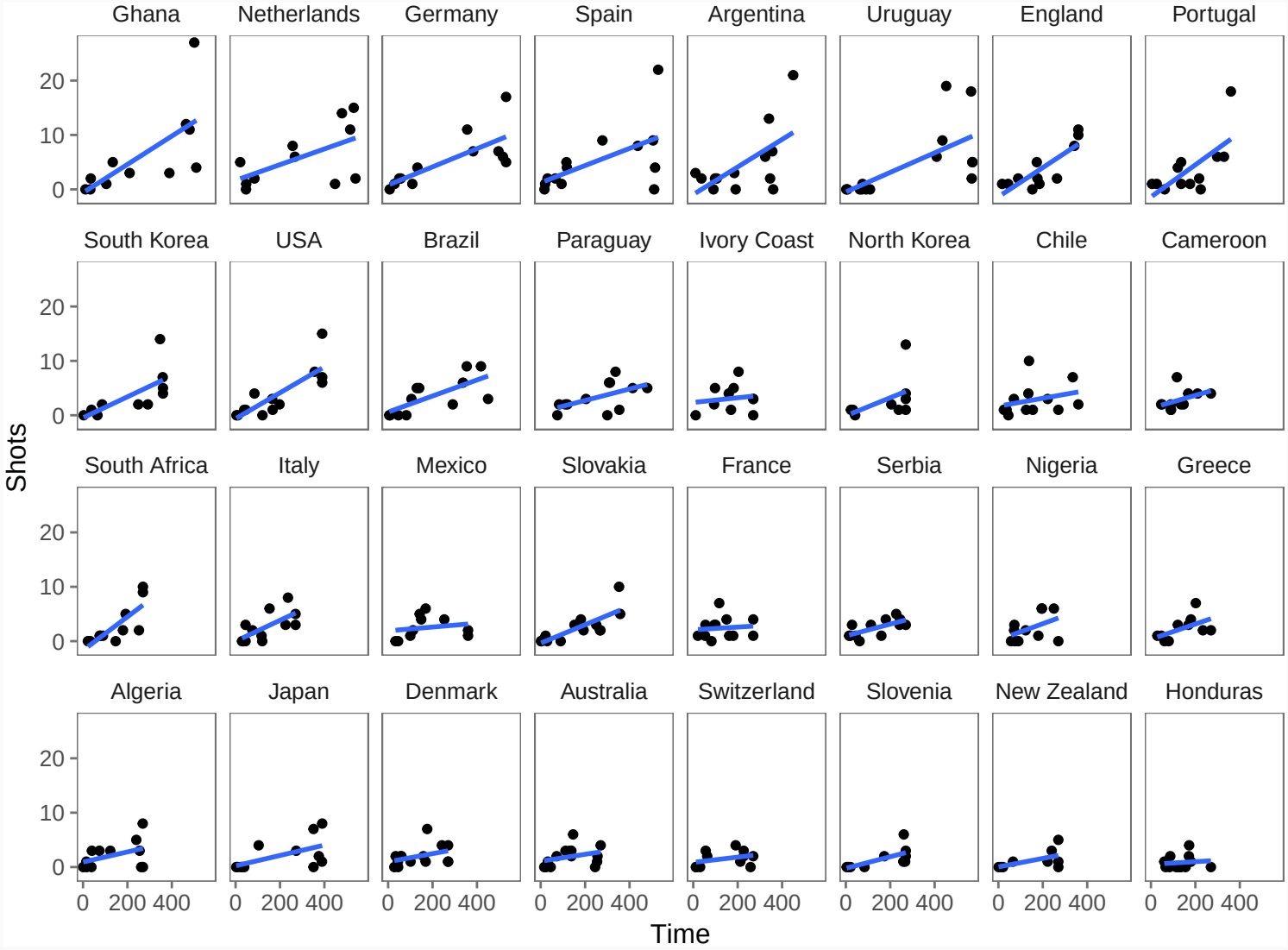
You can re-order factor variables in a graph by resetting the factor using the `factor` function and changing the order that levels are included in the `levels` parameter.

Small multiples

Guideline 5: **When possible, use small multiples.**

Small multiples are graphs that use many small plots showing the same thing for different facets of the data. For example, instead of using color in a single plot to show data for males and females, you could use two small plots, one each for males and females.

Typically, in small multiples, all plots with use the same x- and y-axes. This makes it easier to compare across plots, and it also allows you to save room by limiting axis annotation.


Ghana
Netherlands
Germany
Spain
Argentina
Uruguay
England
Portugal
South Korea
USA
Brazil
Paraguay
Ivory Coast
North Korea
Chile
Cameroon
South Africa
Italy
Mexico
Slovakia
France
Serbia
Nigeria
Greece
Algeria
Japan
Denmark
Australia
Switzerland
Slovenia
New Zealand
Honduras
Shots
0
10
20
0
200
400
Time

Small multiples

You can use the `facet` functions to create small multiples. This separates the graph into several small graphs, one for each level of a factor.

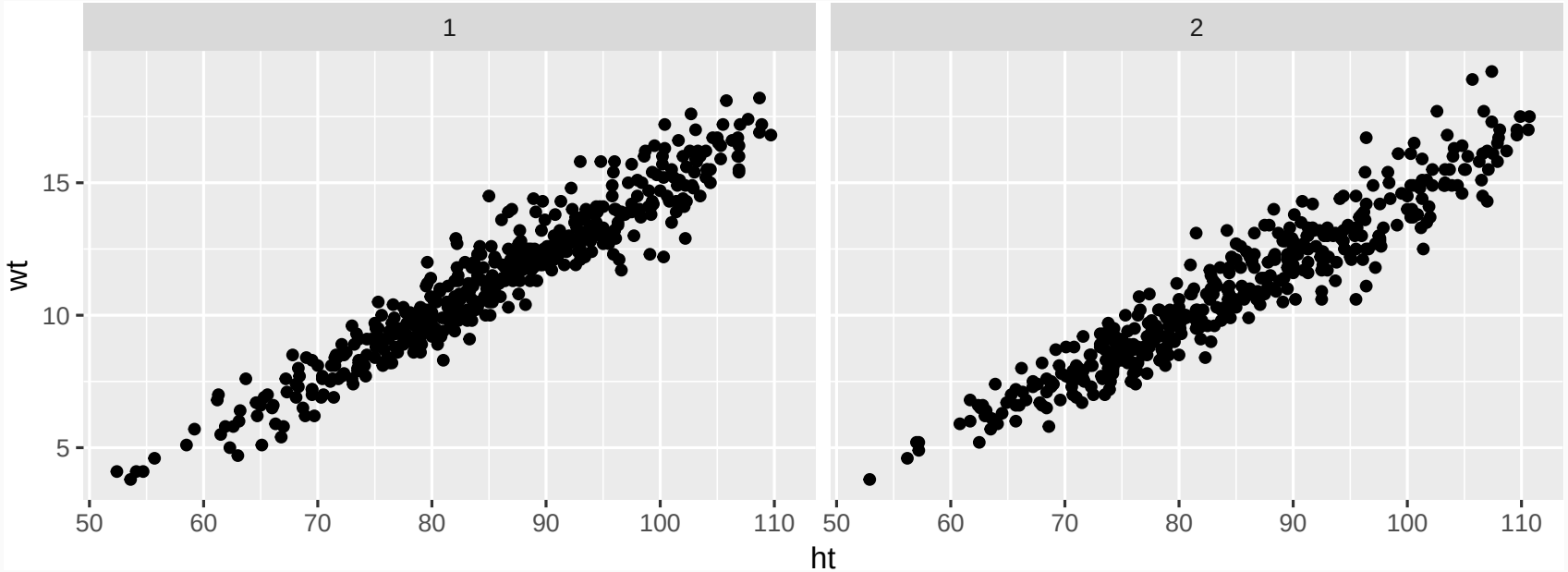
The `facet` functions are:

- `facet_grid()`
- `facet_wrap()`

Small multiples

For example, to create small multiples by sex for the Nepali dataset, when plotting height versus weight, you can call:

```
ggplot(nepali, aes(ht, wt)) +
        geom_point() +
        facet_grid(. ~ sex)
```



Small multiples

The `facet_grid` function can facet by one or two variables. One will be shown by rows, and one by columns:

```
## Generic code
facet_grid([factor for rows] ~ [factor for columns])
```

The `facet_wrap()` function can only facet by one variable, but it can "wrap" the small graphs for that variable, so the don't all have to be in one row or column:

```
## Generic code
facet_wrap(~ [factor for faceting], ncol = [# of columns])
```

Small multiples

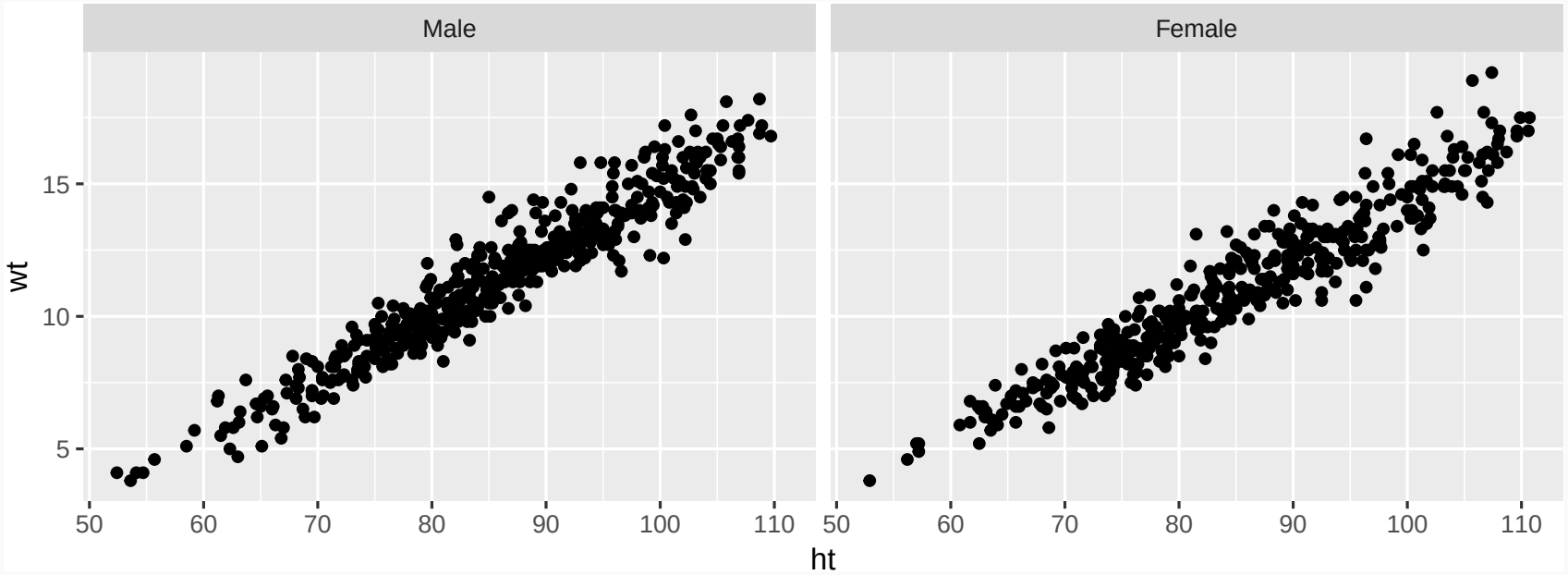
Often, when you do faceting, you'll want to re-name your factors levels or re-order them. For this, you'll need to use the `factor()` function on the original vector. For example, to rename the `sex` factor levels from "1" and "2" to "Male" and "Female", you can run:

```
nepali <- nepali %>%
  mutate(sex = factor(sex, levels = c(1, 2),
                      labels = c("Male", "Female")))
```

Small multiples

Notice that the labels for the two graphs have now changed:

```
ggplot(nepali, aes(ht, wt)) +
        geom_point() +
        facet_wrap(~ sex, ncol = 2)
```



Small multiples

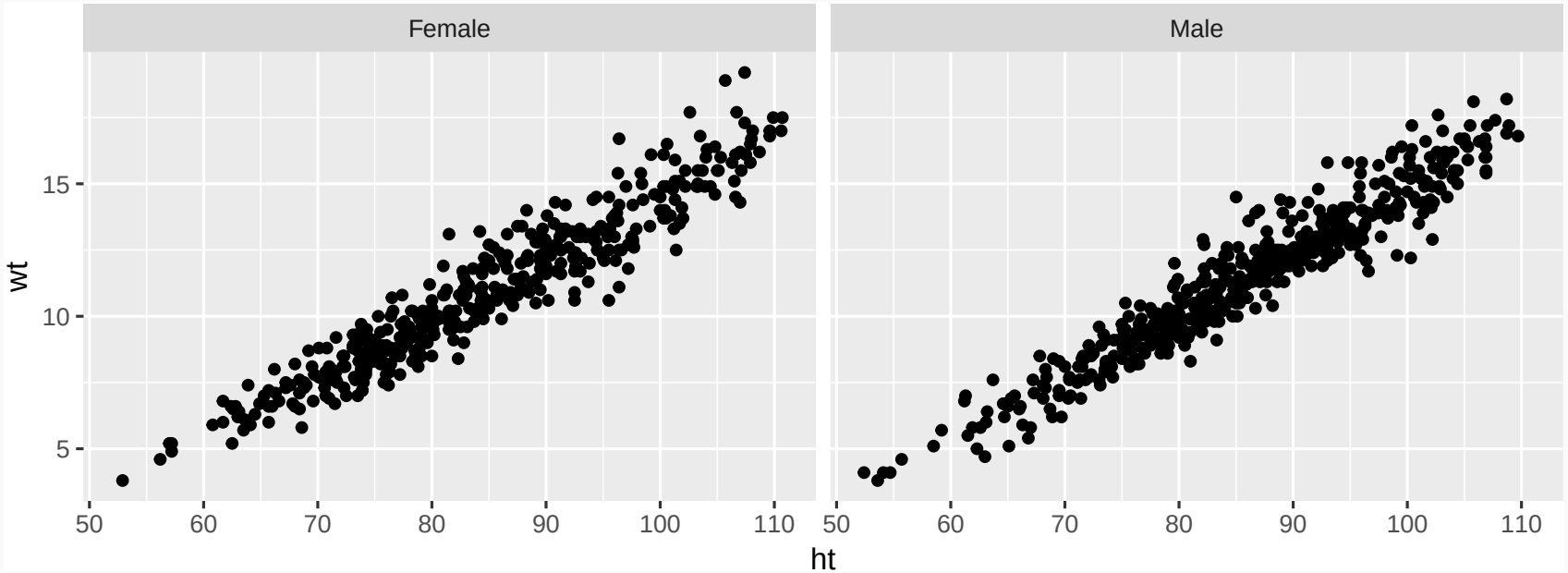
To re-order the factor, and show the plot for "Female" first, you can use `factor` to change the order of the levels:

```
nepali <- nepali %>%
  mutate(sex = factor(sex, levels = c("Female", "Male")))
```

Small multiples

Now notice that the order of the plots has changed:

```
ggplot(nepali, aes(ht, wt)) +
        geom_point() +
        facet_grid(. ~ sex)
```



In-course exercise

We'll now take a break to do Section 4.10.3 of the In-Course Exercise.

Advanced customization

Scales

There are a number of different functions for adjusting scales. These follow the following convention:

```
## Generic code
scale_[aesthetic]_[vector type]
```

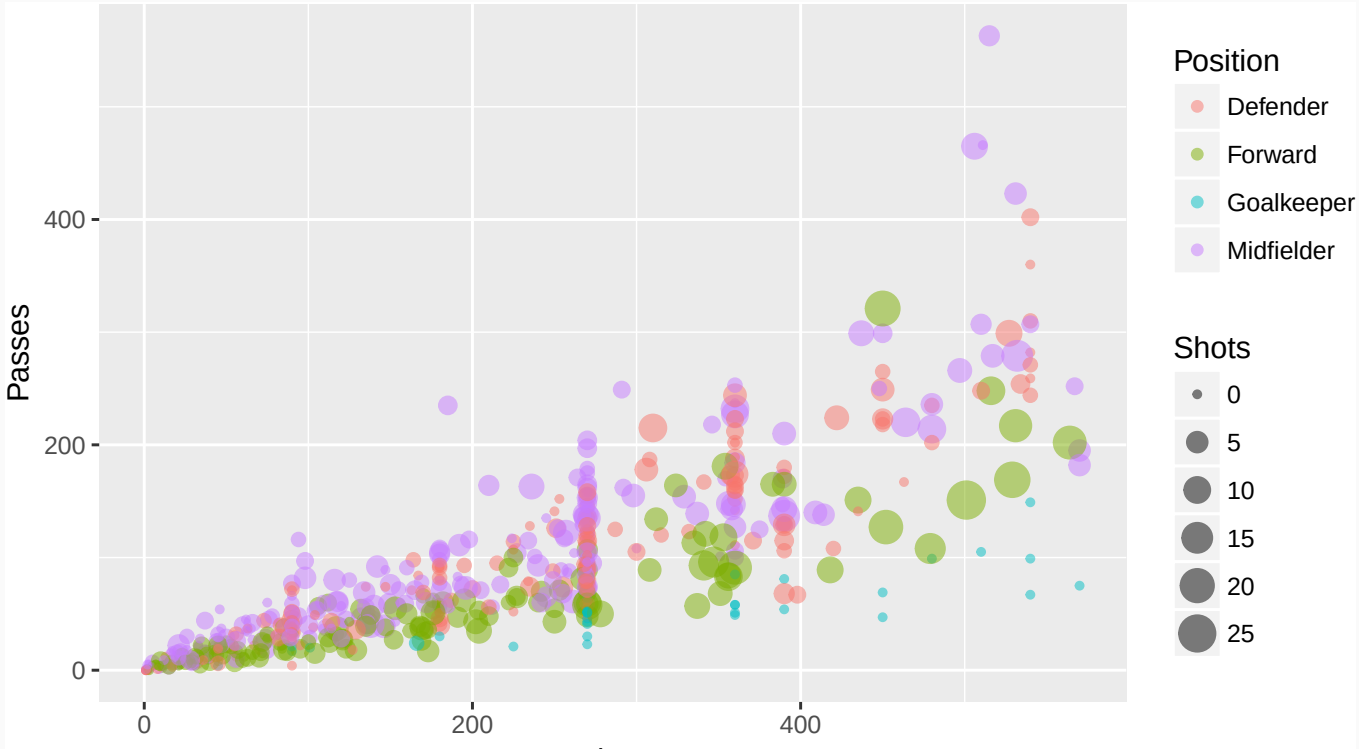
For example, to adjust the x-axis scale for a continuous variable, you'd use `scale_x_continuous`.

You can use a `scale` function for an axis to change things like the axis label (which you could also change with `xlab` or `ylab`) as well as position and labeling of breaks.

Scales

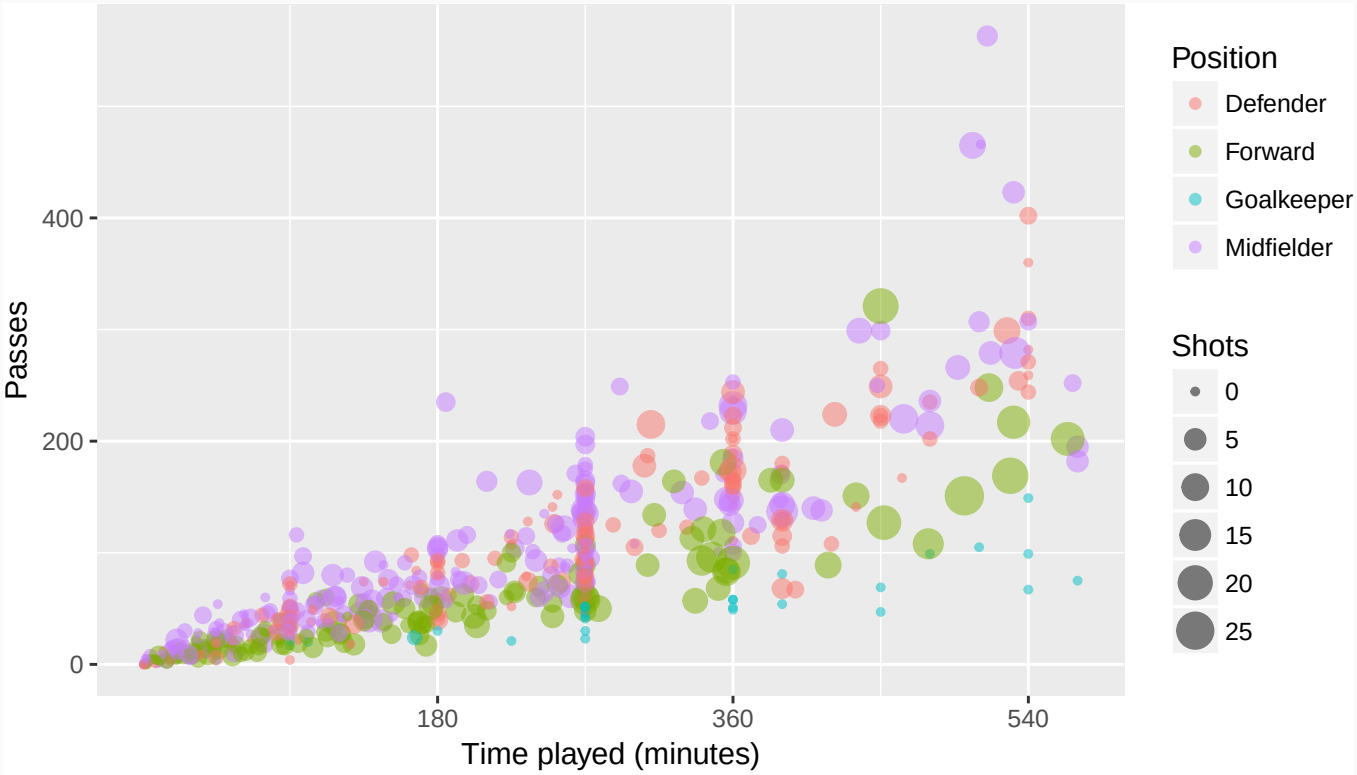
For example, here is the default for plotting time versus passes for the `worldcup` dataset, with the number of shots taken shown by size and position shown by color:

```
ggplot(worldcup, aes(x = Time, y = Passes,
                     color = Position, size = Shots)) +
  geom_point(alpha = 0.5)
```



Scales

```
ggplot(worldcup, aes(x = Time, y = Passes,
                     color = Position, size = Shots)) +
  geom_point(alpha = 0.5) +
  scale_x_continuous(name = "Time played (minutes)",
                     breaks = 90 * c(2, 4, 6),
                     minor_breaks = 90 * c(1, 3, 5))
```



Scales

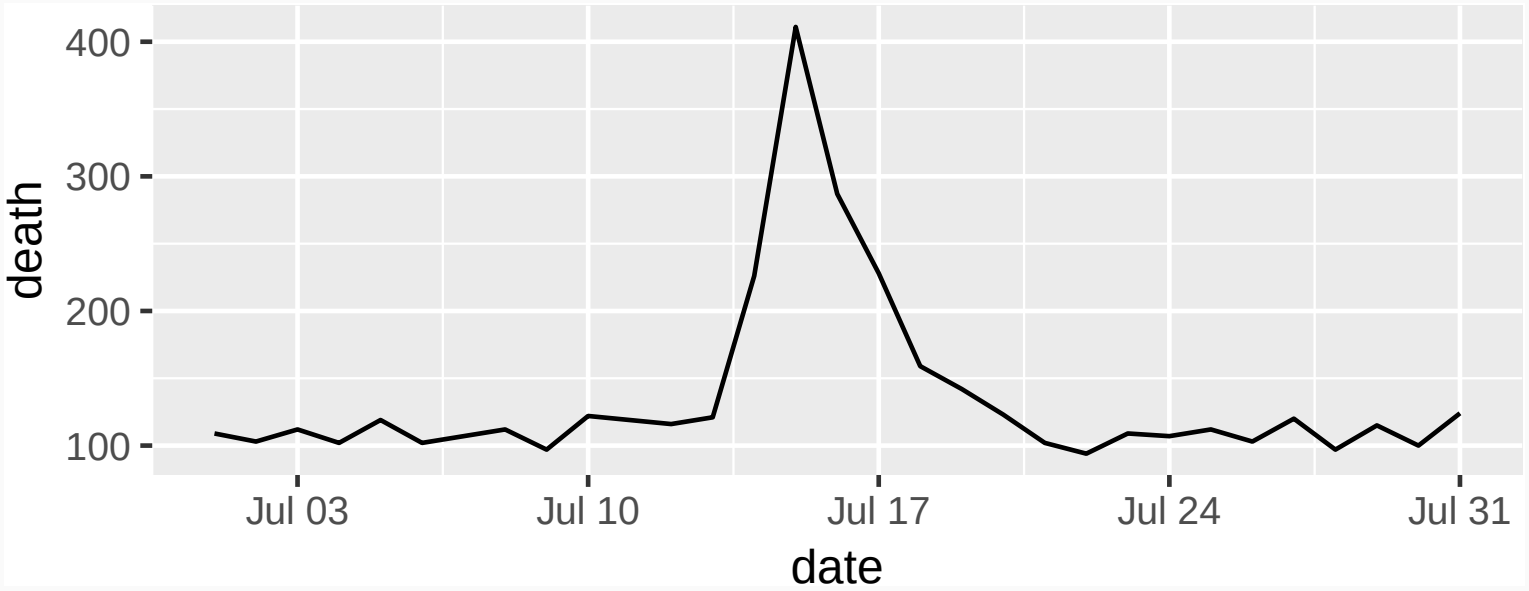
Parameters you might find useful in `scale` functions include:

Parameter	Description
name	Label or legend name
breaks	Vector of break points
minor_breaks	Vector of minor break points
labels	Labels to use for each break
limits	Limits to the range of the axis

Scales

For dates, you can use `scale` functions like `scale_x_date` and `scale_x_datetime`. For example, here's a plot of deaths in Chicago in July 1995 using default values for the x-axis:

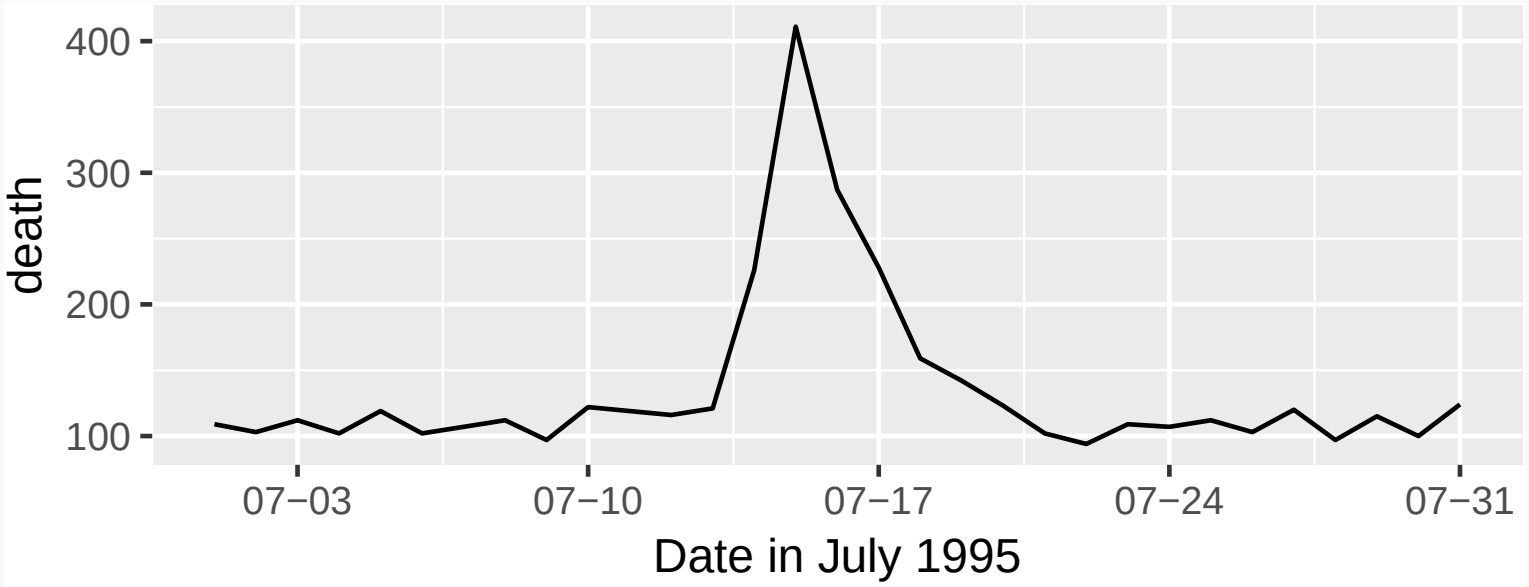
```
ggplot(chic_july, aes(x = date, y = death)) +
  geom_line()
```



Scales

And here's an example of changing the formatting and name of the x-axis:

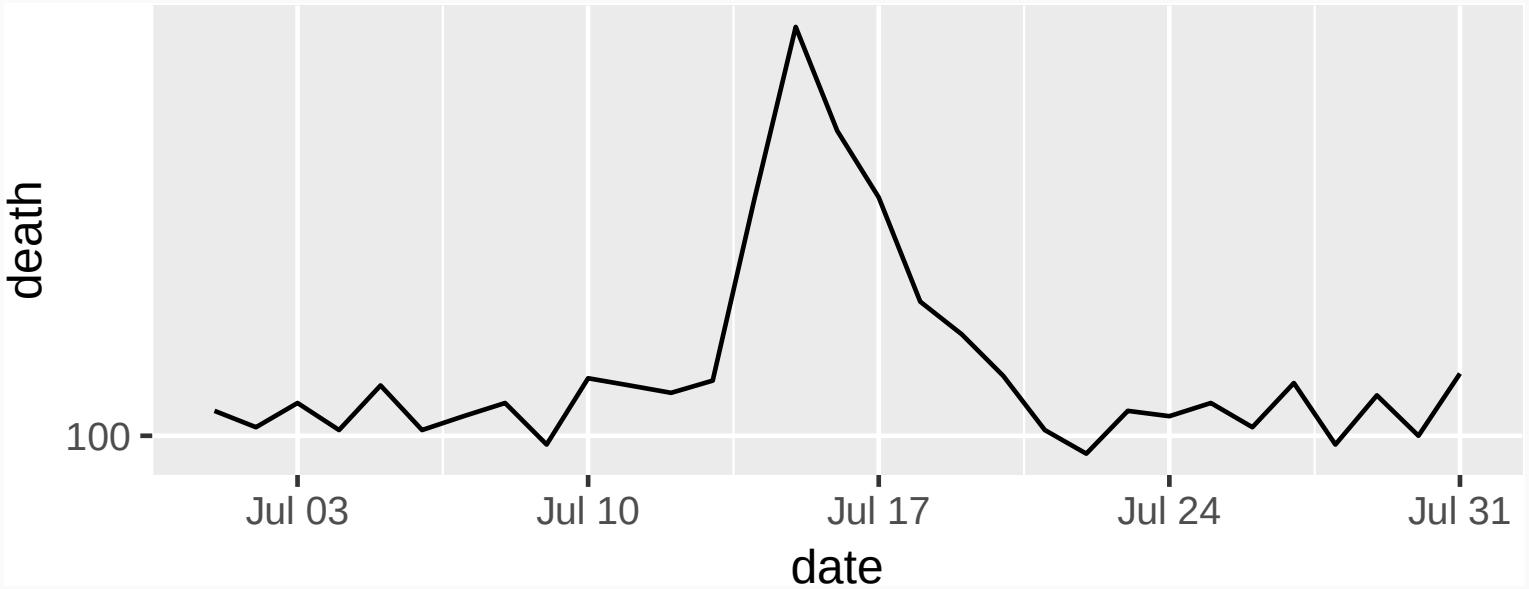
```
ggplot(chic_july, aes(x = date, y = death)) +
  geom_line() +
  scale_x_date(name = "Date in July 1995",
               date_labels = "%m-%d")
```



Scales

You can also use the `scale` functions to transform an axis. For example, to show the Chicago plot with "deaths" on a log scale, you can run:

```
ggplot(chic_july, aes(x = date, y = death)) +
  geom_line() +
  scale_y_log10()
```



Scales

For colors and fills, the conventions for the names of the `scale` functions can vary.

For example, to adjust the color scale when you're mapping a discrete variable (i.e., categorical, like gender or animal breed) to color, you'd use `scale_color_hue`. To adjust the color scale for a continuous variable, like age, you'll use `scale_color_gradient`.

Scales

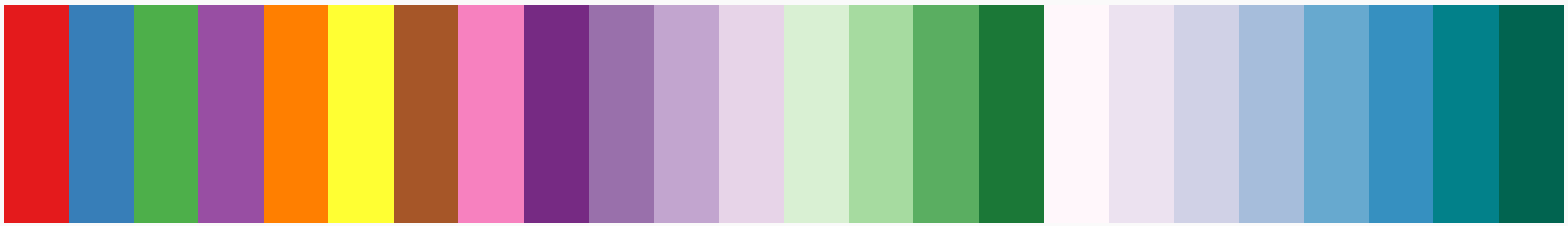
For any color scales, consider starting with `brewer` first (e.g., `scale_color_brewer`).

Scale functions from `brewer` allow you to set colors using different palettes. You can explore these palettes at `http://colorbrewer2.org/`.

Scales

The Brewer palettes fall into three categories: sequential, divergent, and qualitative. You should use sequential or divergent for continuous data and qualitative for categorical data. Use `display.brewer.pal` to show the palette for a given number of colors.

```
library(RColorBrewer)
display.brewer.pal(name = "Set1", n = 8)
display.brewer.pal(name = "PRGn", n = 8)
display.brewer.pal(name = "PuBuGn", n = 8)
```

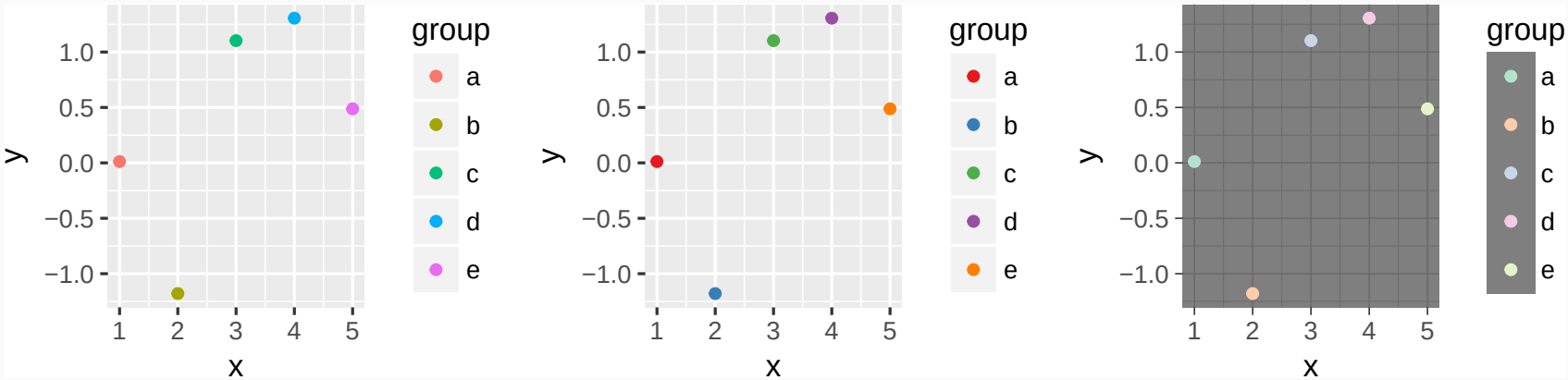


Set1 (qualitative) PRGn (divergent) PuBuGn (sequential)

Scales

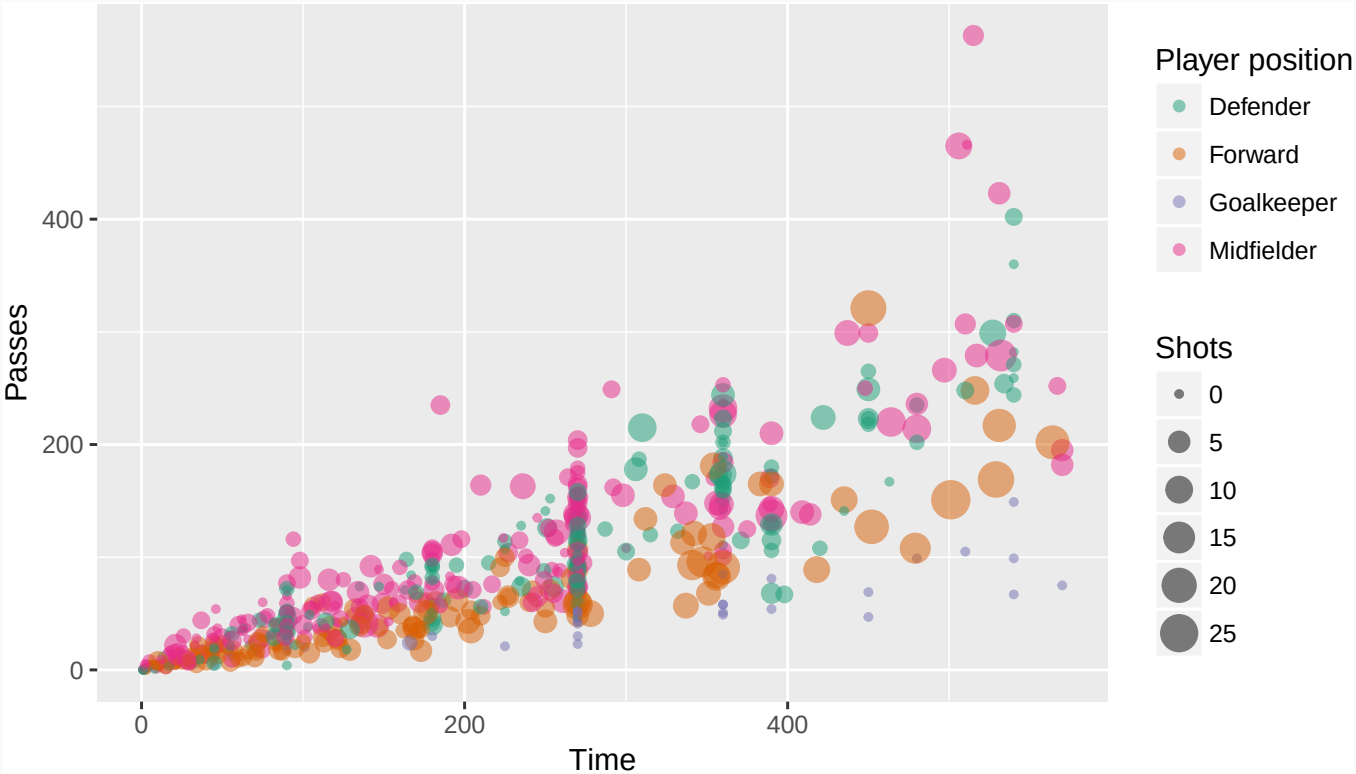
Use the `palette` argument within a `scales` function to customize the palette:

```
a <- ggplot(data.frame(x = 1:5, y = rnorm(5),
                       group = letters[1:5]),
            aes(x = x, y = y, color = group)) +
  geom_point()
b <- a + scale_color_brewer(palette = "Set1")
c <- a + scale_color_brewer(palette = "Pastel2") +
  theme_dark()
grid.arrange(a, b, c, ncol = 3)
```



Scales

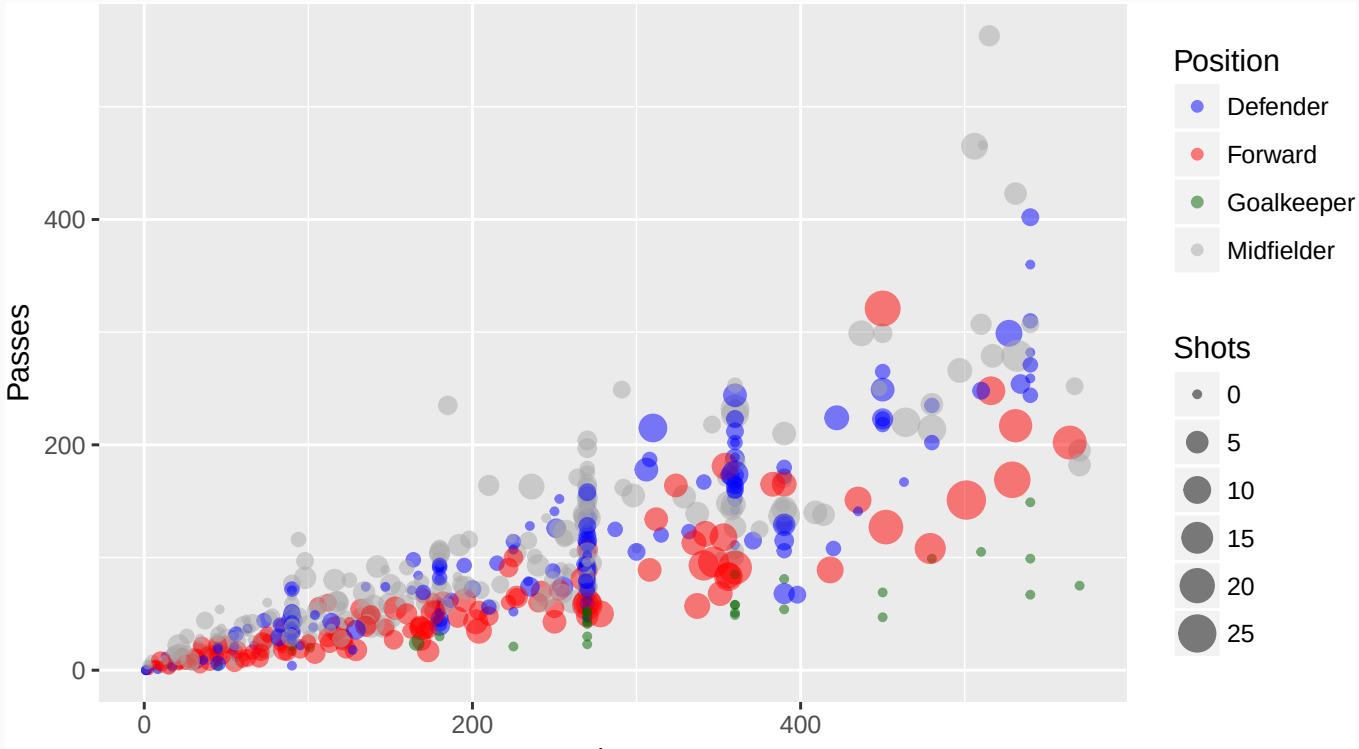
```
ggplot(worldcup, aes(x = Time, y = Passes,
                     color = Position, size = Shots)) +
  geom_point(alpha = 0.5) +
  scale_color_brewer(palette = "Dark2",
                     name = "Player position")
```



Scales

You can also set colors manually:

```
ggplot(worldcup, aes(x = Time, y = Passes,
                     color = Position, size = Shots)) +
  geom_point(alpha = 0.5) +
  scale_color_manual(values = c("blue", "red",
                                "darkgreen", "darkgray"))
```



Excellent references

- Chapter 3 of "R for Data Science", `http://r4ds.had.co.nz/`

Some excellent further references for plotting are:

- R Graphics Cookbook (book and website)
- Google images

For more technical details about plotting in R:

- ggplot2: Elegant Graphics for Data Analysis, Hadley Wickham
- R Graphics, Paul Murrell