Plotting in R

EC 425/525, Lab 5

Edward Rubin
10 May 2019

Prologue

Schedule

Last time

Regession

Today

Plotting in `R` (especially `ggplot2`)

Plotting

Plotting

The default option: `plot()`

While we'll quickly move on to other options, `R`'s `plot()` function (in the default `graphics` package) is a great tool for basic data exploration—it's fast, simple, and flexible.

Plotting

The default option: `plot()`

While we'll quickly move on to other options, `R`'s `plot()` function (in the default `graphics` package) is a great tool for basic data exploration—it's fast, simple, and flexible.

In fact, `plot()` is a generic function, that works for many classes.

Plotting

The default option: `plot()`

While we'll quickly move on to other options, `R`'s `plot()` function (in the default `graphics` package) is a great tool for basic data exploration—it's fast, simple, and flexible.

In fact, `plot()` is a generic function, that works for many classes.

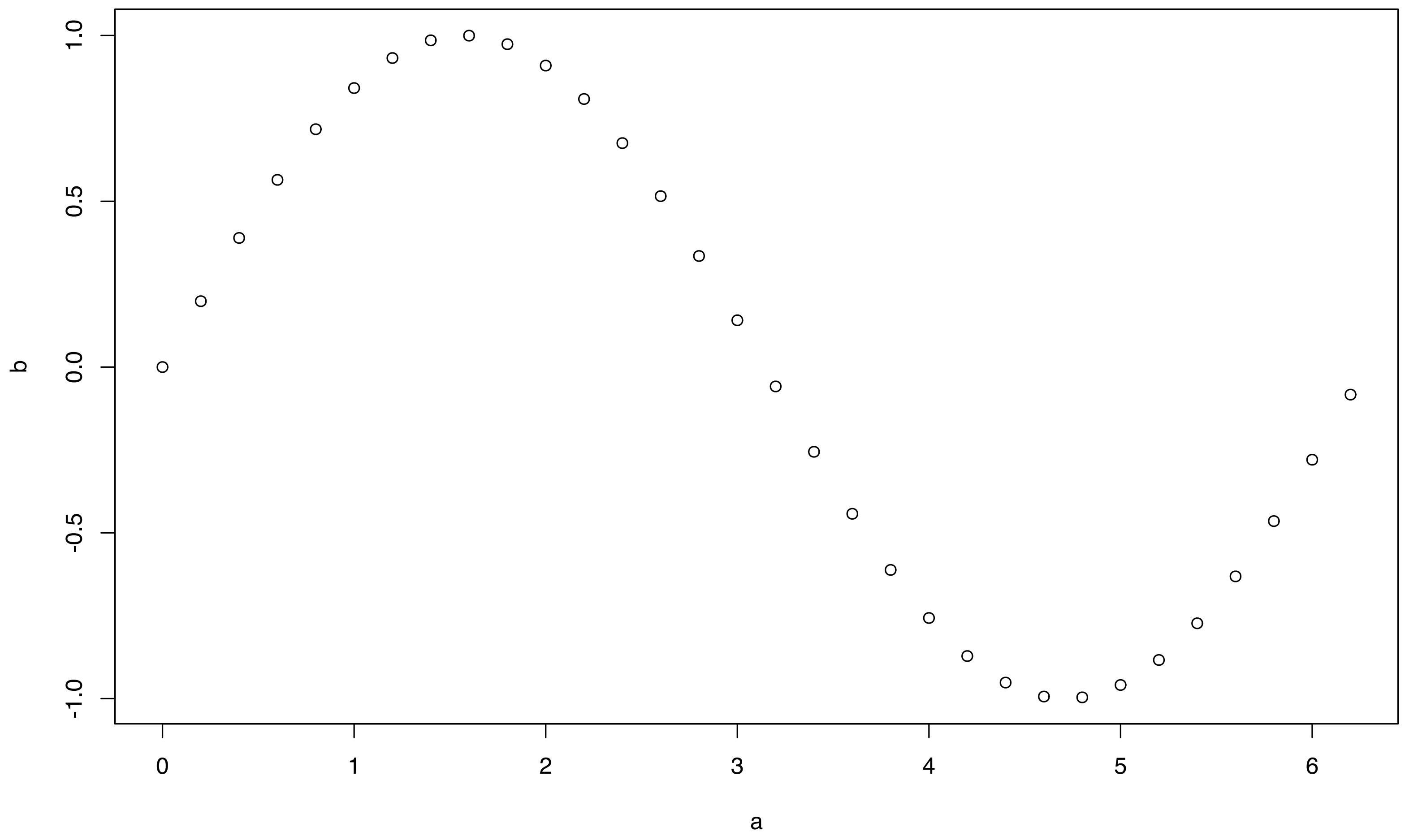
General arguments for `plot()`:

- `x` and `y` for coordinates
- `type =` {`"p"`oints, `"l"`ines, *etc.*} *(optional)*
- `xlab`, `ylab`, `main`, and `sub` for axis labels and (sub)title *(optional)*
- `col` and `pch` for color and plot character *(optional)*
- `lty` and `lwd` for line type, and line width *(optional)*

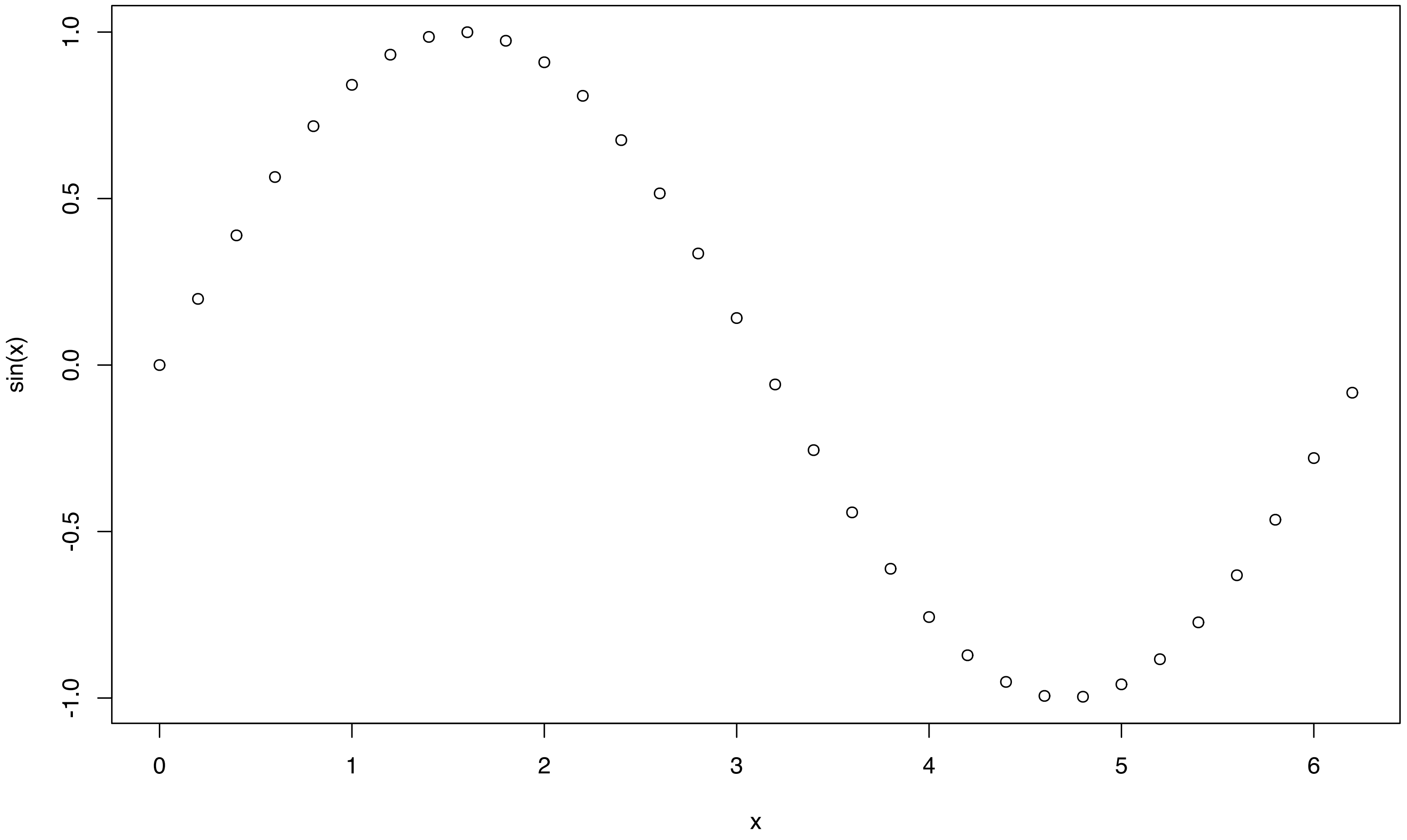
Let's see `plot()` in action.

```r
# Define two vectors
a ← seq(from = 0, to = 2*pi, by = 0.2)
b ← sin(a)
```

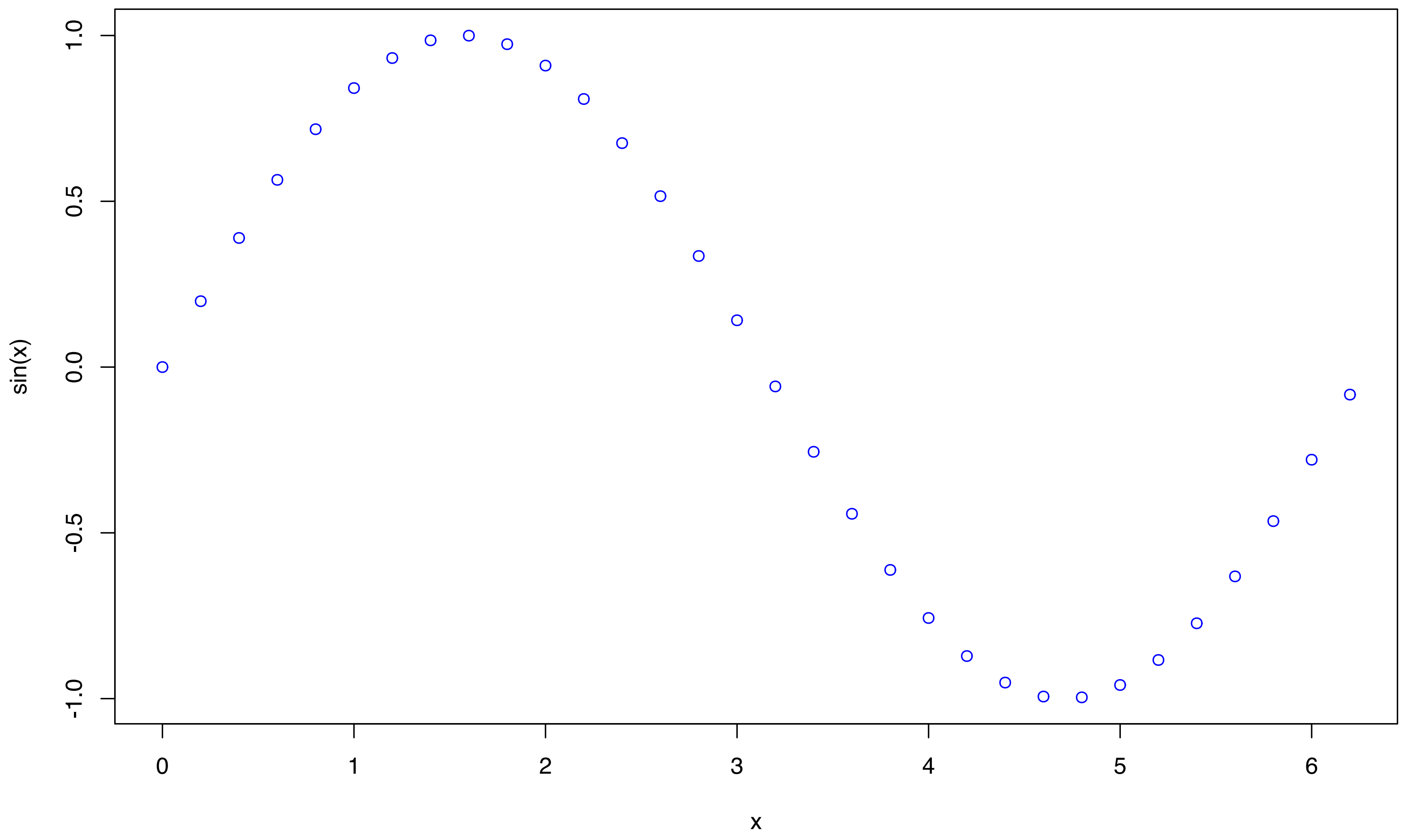
```
plot(x = a, y = b)
```



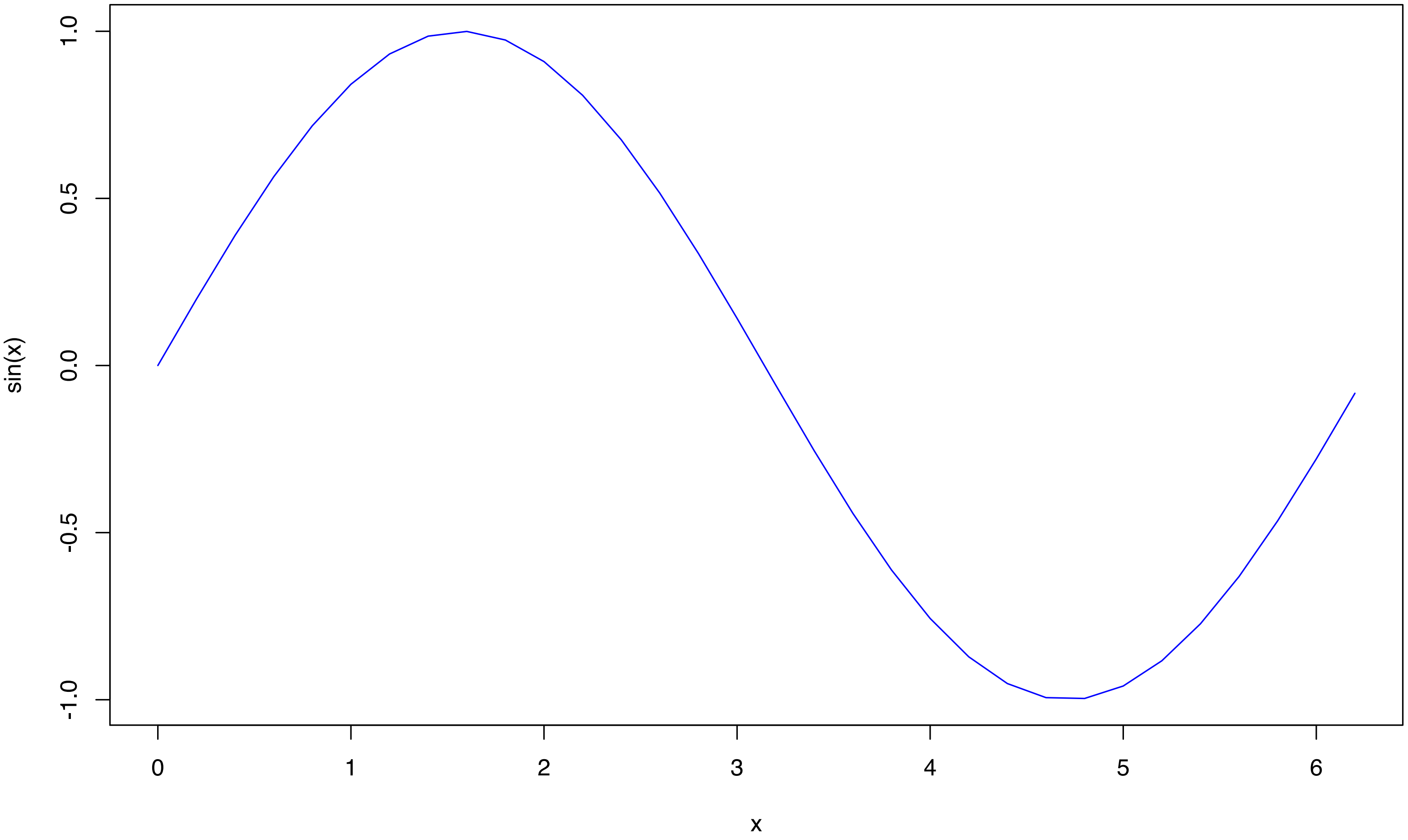
```r
plot(x = a, y = b, xlab = "x", ylab = "sin(x)")
```



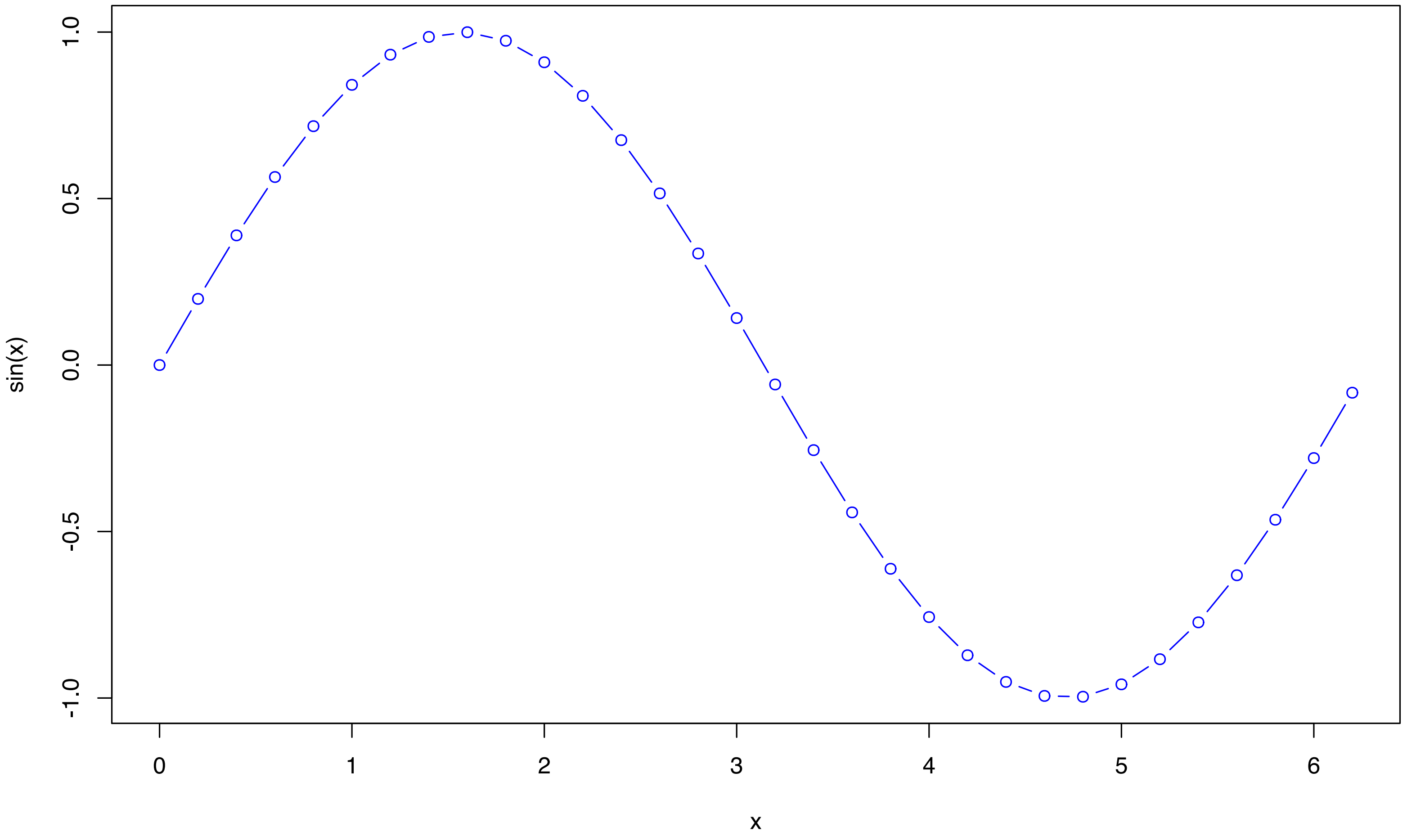
```r
plot(x = a, y = b, xlab = "x", ylab = "sin(x)", col = "blue")
```



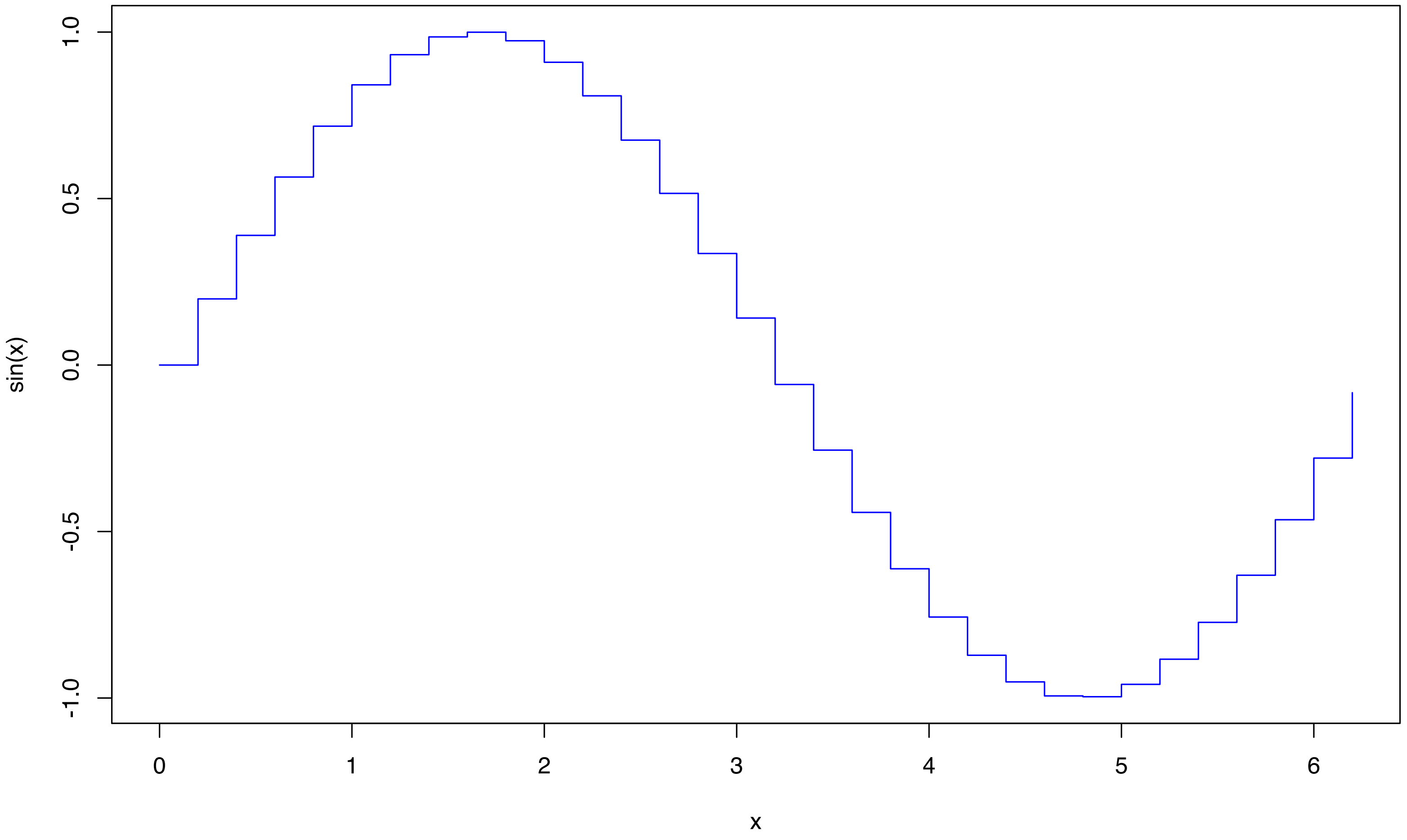
```r
plot(x = a, y = b, xlab = "x", ylab = "sin(x)", col = "blue", type = "l")
```



```r
plot(x = a, y = b, xlab = "x", ylab = "sin(x)", col = "blue", type = "b")
```



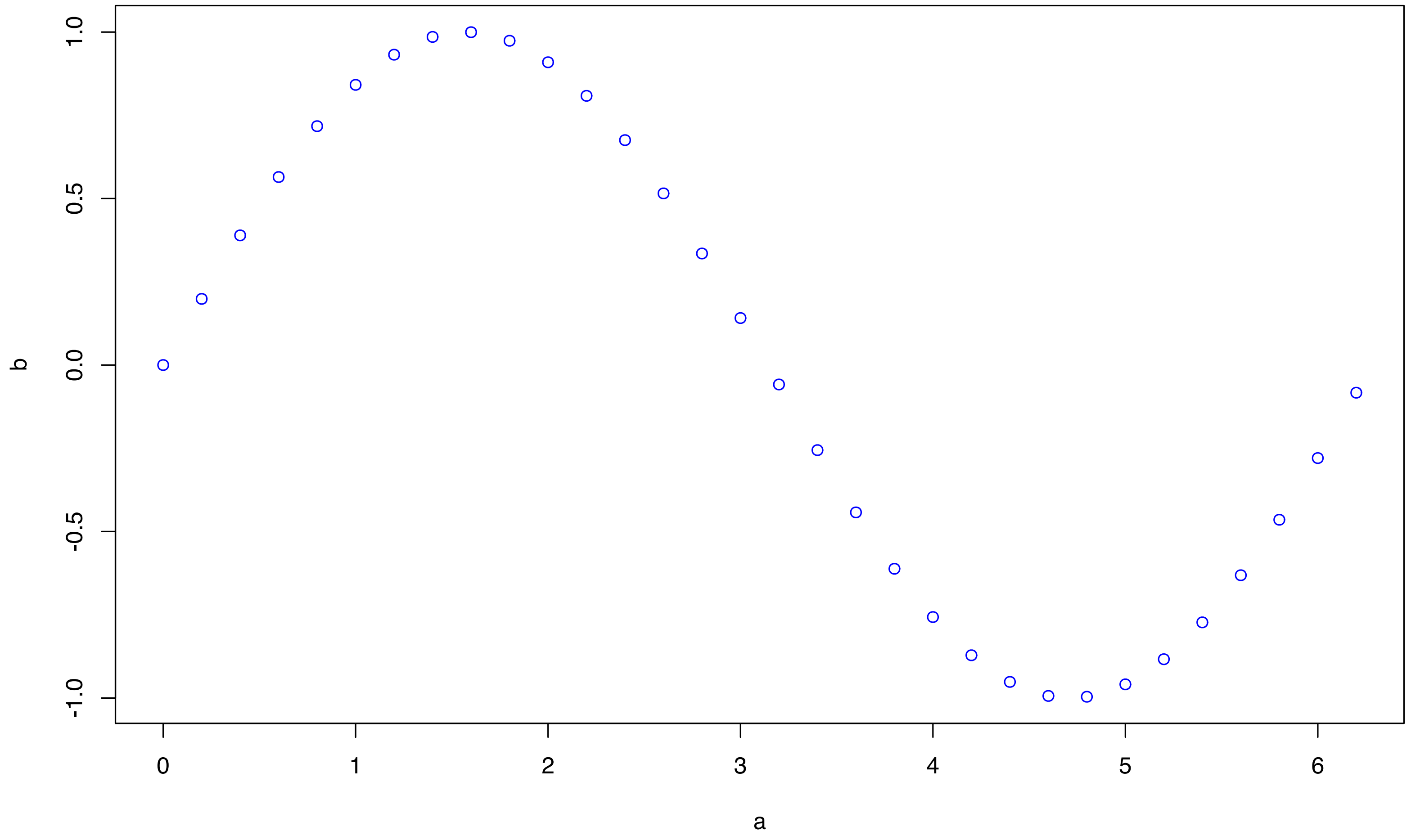
```r
plot(x = a, y = b, xlab = "x", ylab = "sin(x)", col = "blue", type = "s")
```



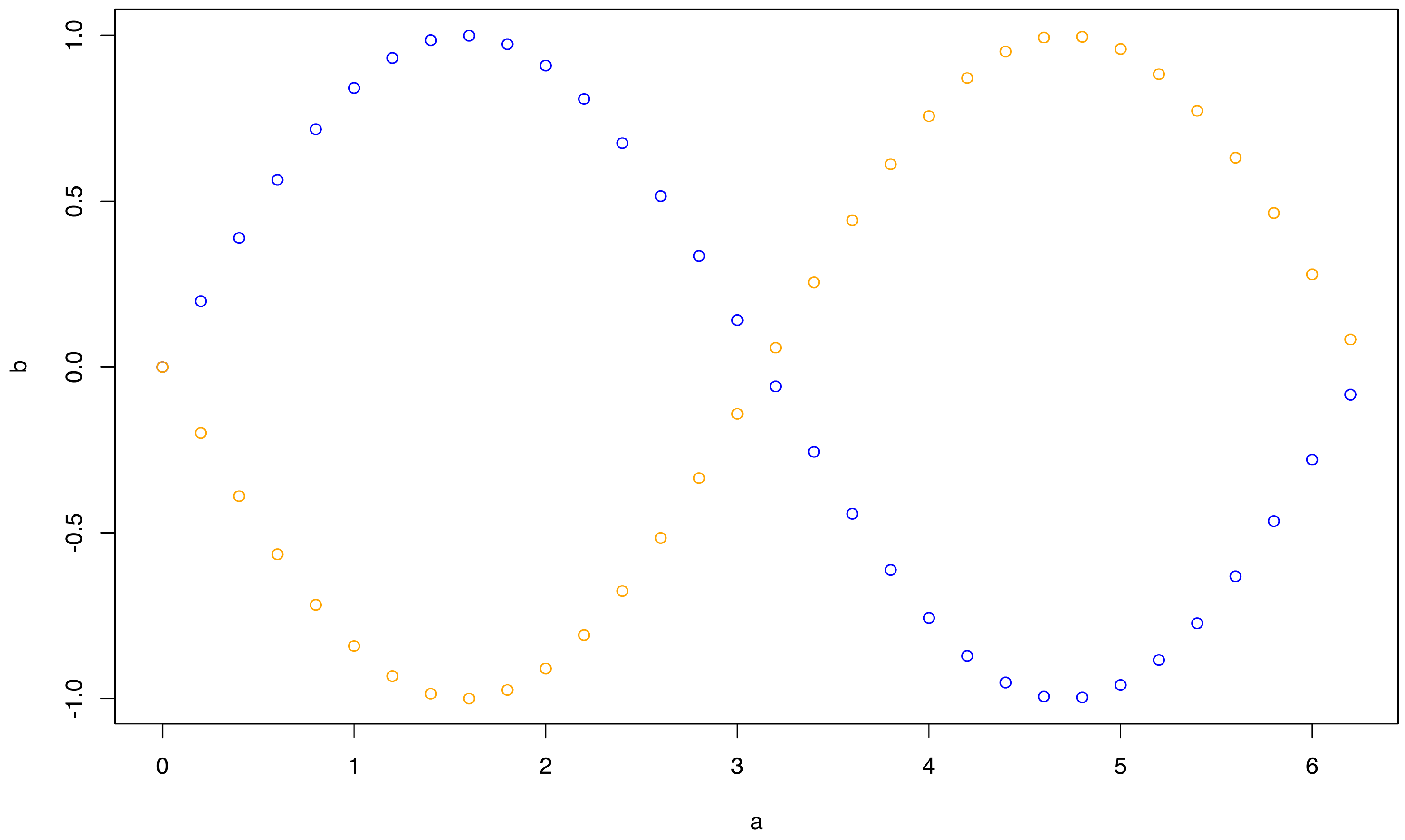
`plot()` is essentially calling `points()` or `lines()`.

You can layer plots by using these individual functions.

```r
plot(x = a, y = b, col = "blue")
```

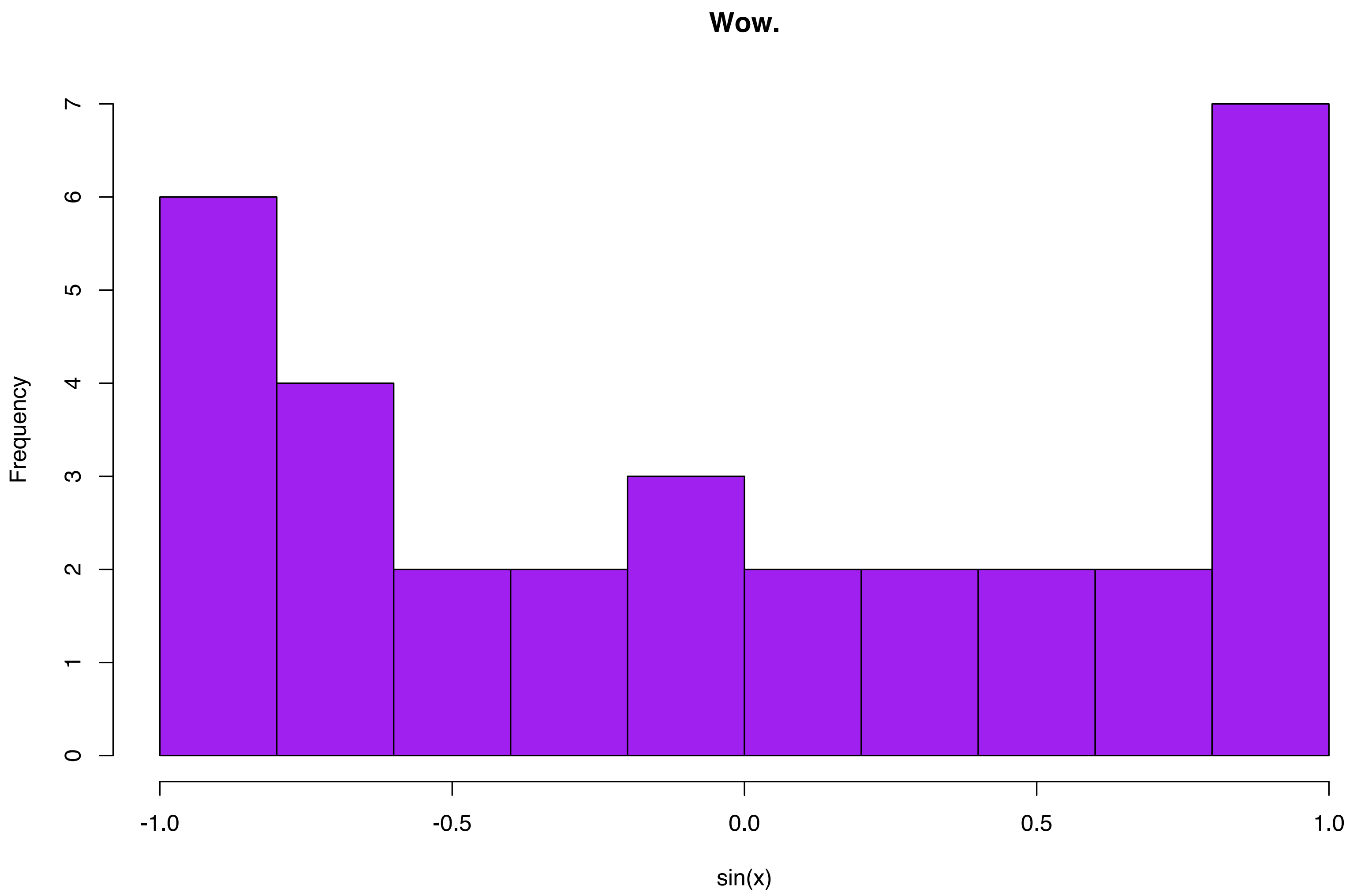


```r
plot(x = a, y = b, col = "blue"); points(x = a, y = -b, col = "orange")
```



`graphics` also offers a nice histogram function in `hist()`.

```r
hist(x = b, breaks = 10, col = "purple", xlab = "sin(x)", main = "Wow.")
```



That said/done, further customization/manipulation of your graphics using `graphics` plotting functions can become quite difficult.

Enter `ggplot2`

ggplot2

ggplot2

The grammar

The `ggplot2` package offers an incredibly flexible, diverse, and powerful set of functions for creating graphics in `R`.

ggplot2

The grammar

The `ggplot2` package offers an incredibly flexible, diverse, and powerful set of functions for creating graphics in `R`.

The `gg` stands for the *grammar of graphics*.

ggplot2

The grammar

The `ggplot2` package offers an incredibly flexible, diverse, and powerful set of functions for creating graphics in `R`.

The `gg` stands for the *grammar of graphics*.

`ggplot2`

1. centers on a **data frame** (the `data` argument)
2. maps variables to **aesthetics** (the `aes` argument)
3. **layers geometries** to *build up* your graphic

Note The package is called `ggplot2`, but the main function is `ggplot()`.

ggplot2

`ggplot()`

Main arguments

1. **`data`** Your dataset. As a data frame (or `tibble`).

ggplot2

`ggplot()`

Main arguments

1. **`data`** Your dataset. As a data frame (or `tibble`).
2. **`aes()`** Maps variables in `data` to "aesthetics" like `x`, `color`, `shape`.

ggplot2

`ggplot()`

Main arguments

1. `data` Your dataset. As a data frame (or `tibble`).
2. `aes()` Maps variables in `data` to "aesthetics" like `x`, `color`, `shape`.

Example A time series of problems, `color` defined by money

```
library(ggplot2)
ggplot(
  data = pretend_df,
  aes(x = time, y = problems, color = money)
)
```

ggplot2

Layers

The `ggplot()` function doesn't plot anything—it *sets up* the plot.

To create the actual figure, you layer **geometries** (*e.g.*, `geom_point()`),

ggplot2

Layers

The `ggplot()` function doesn't plot anything—it *sets up* the plot.

To create the actual figure, you layer **geometries** (*e.g.*, `geom_point()`), **scales** (*e.g.*, `scale_color_manual()`),

ggplot2

Layers

The `ggplot()` function doesn't plot anything—it *sets up* the plot.

To create the actual figure, you layer **geometries** (*e.g.*, `geom_point()`), **scales** (*e.g.*, `scale_color_manual()`), and other **options** (*e.g.*, `xlab()`).

ggplot2

Layers

The `ggplot()` function doesn't plot anything—it *sets up* the plot.

To create the actual figure, you layer **geometries** (*e.g.*, `geom_point()`), **scales** (*e.g.*, `scale_color_manual()`), and other **options** (*e.g.*, `xlab()`).

You **add layers** using the addition sign (`+`).

Layers

The `ggplot()` function doesn't plot anything—it *sets up* the plot.

To create the actual figure, you layer **geometries** (*e.g.*, `geom_point()`), **scales** (*e.g.*, `scale_color_manual()`), and other **options** (*e.g.*, `xlab()`).

You **add layers** using the addition sign (`+`).

Example A time series of problems, `color` defined by money

```
library(ggplot2)
ggplot(
  data = pretend_df,
  aes(x = time, y = problems, color = money)
) +
geom_point() + geom_line()
```

Alright, let's build a plot.

We'll use the `economics` dataset that comes with `ggplot2` (because economics).

Show 8 entries Search:

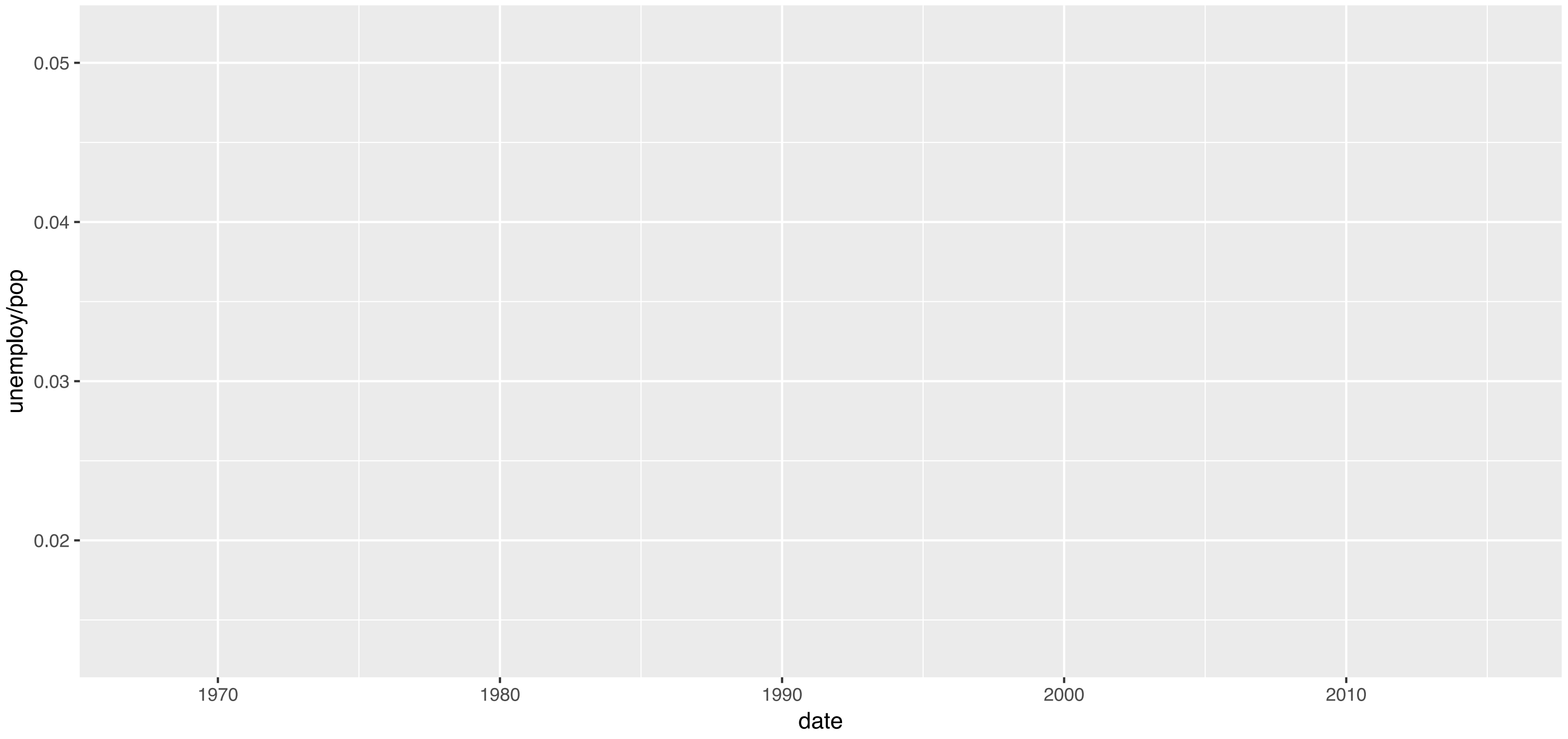
	date	pce	pop	psavert	uempmed	unemploy
1	1967-07-01	507.4	198712	12.5	4.5	2944
2	1967-08-01	510.5	198911	12.5	4.7	2945
3	1967-09-01	516.3	199113	11.7	4.6	2958
4	1967-10-01	512.9	199311	12.5	4.9	3143
5	1967-11-01	518.1	199498	12.5	4.7	3066
6	1967-12-01	525.8	199657	12.1	4.8	3018
7	1968-01-01	531.5	199808	11.7	5.1	2878
8	1968-02-01	534.2	199920	12.2	4.5	3001

Showing 1 to 8 of 574 entries

Previous 1 2 3 4 5 … 72 Next

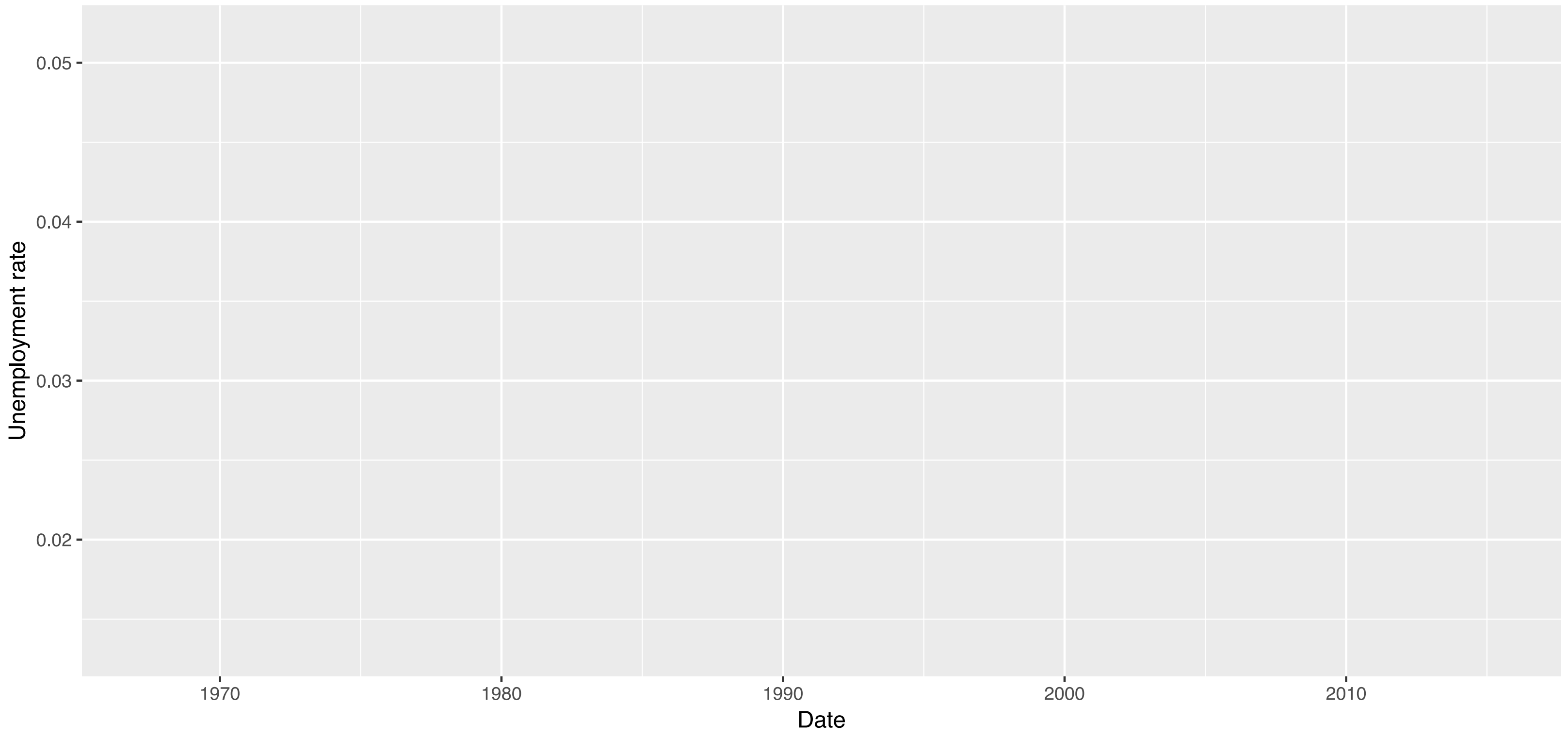
Set up the plot.

```
ggplot(data = economics, aes(x = date, y = unemploy/pop))
```



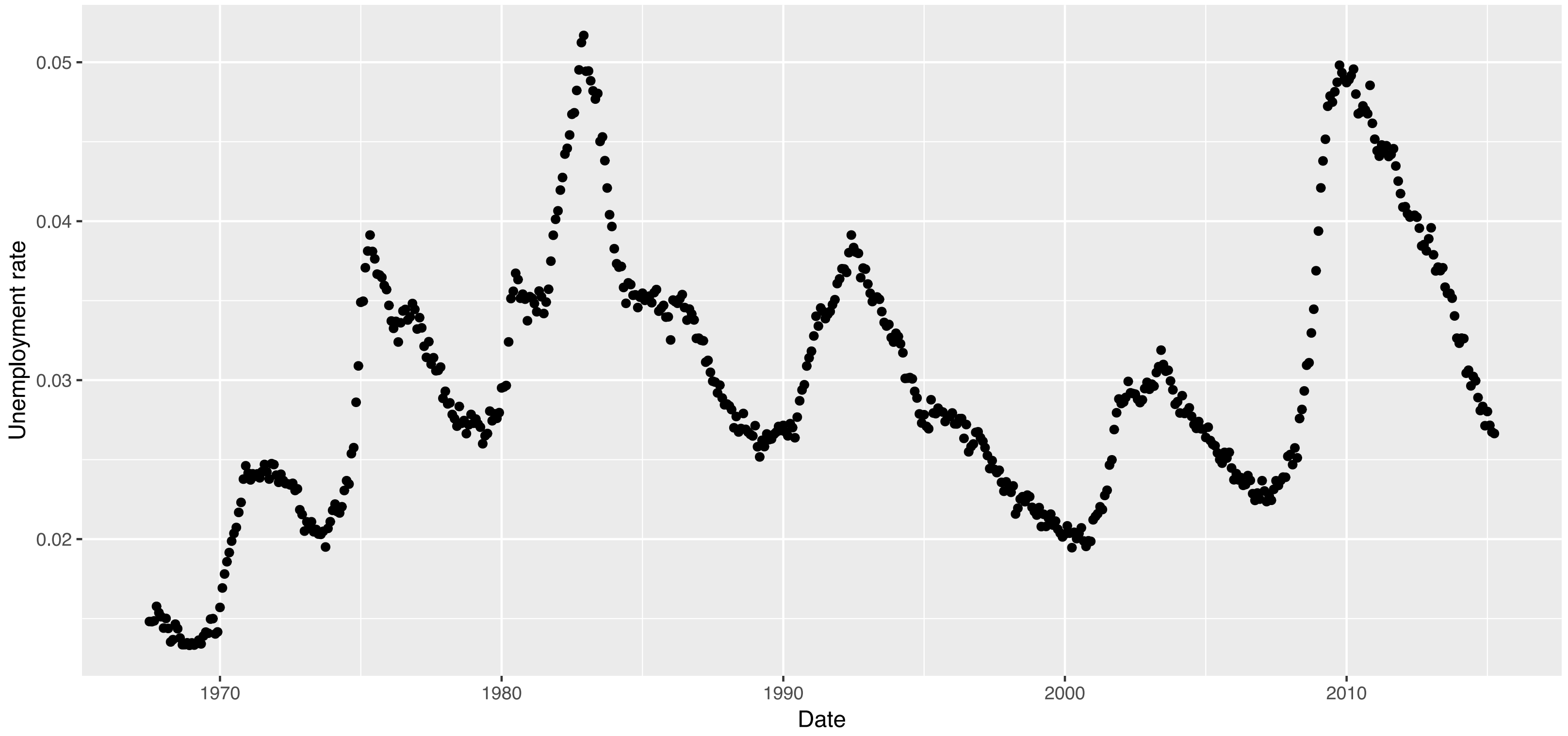
Label the axes.

```r
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date")
```



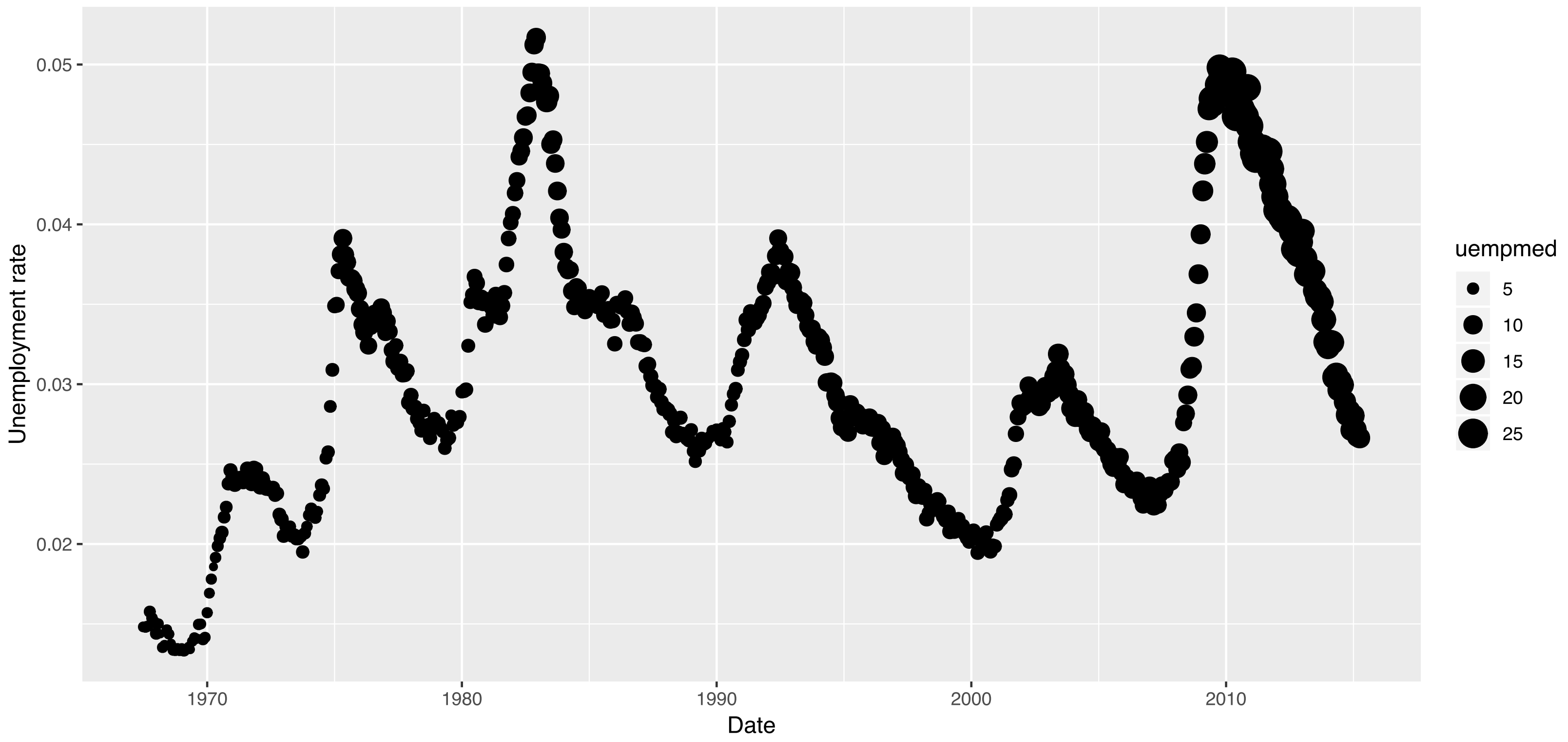
Draw some points.

```r
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point()
```



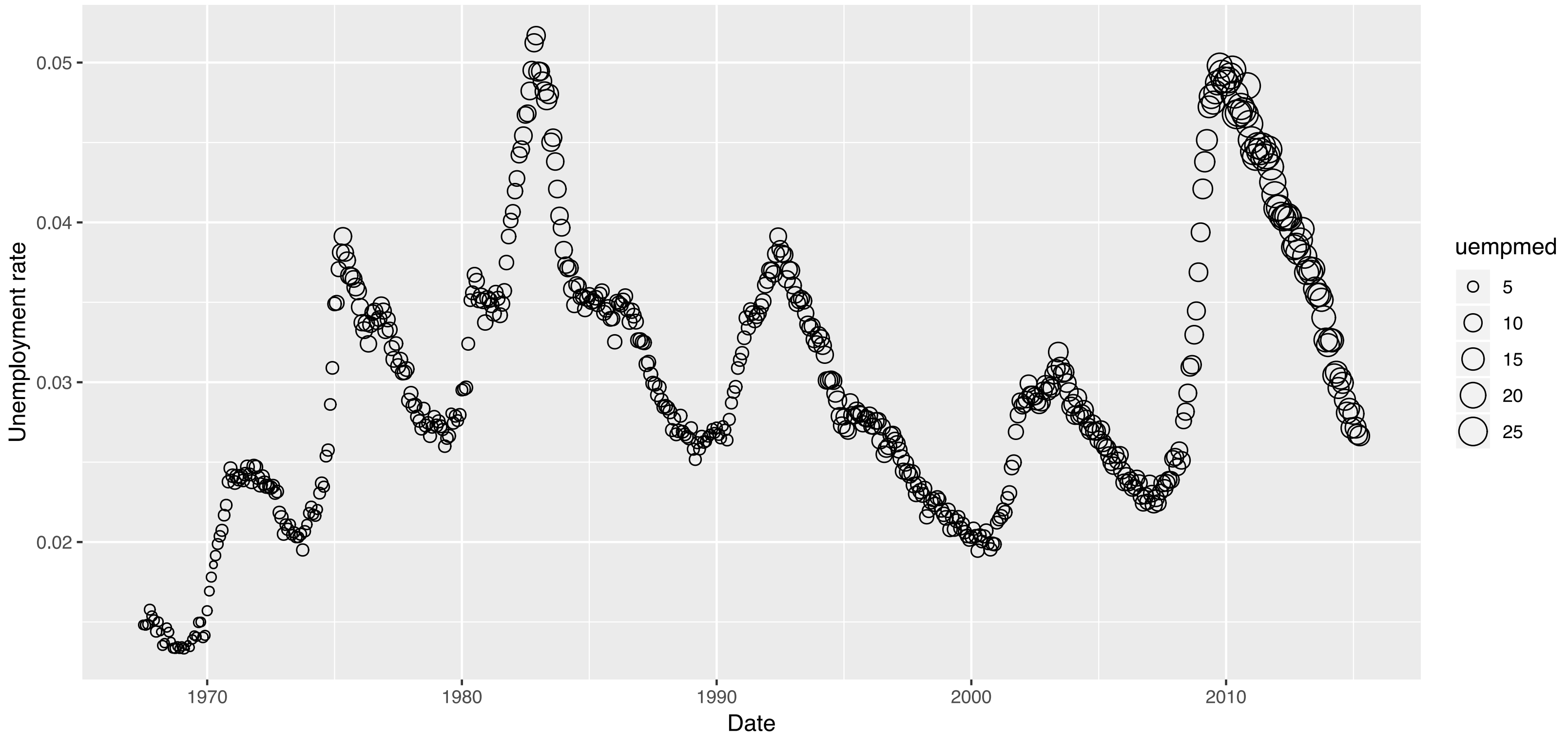
Map the `size` to the median duration of unemployment.

```
ggplot(data = economics, aes(x = date, y = unemploy/pop, size = uempmed)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point()
```



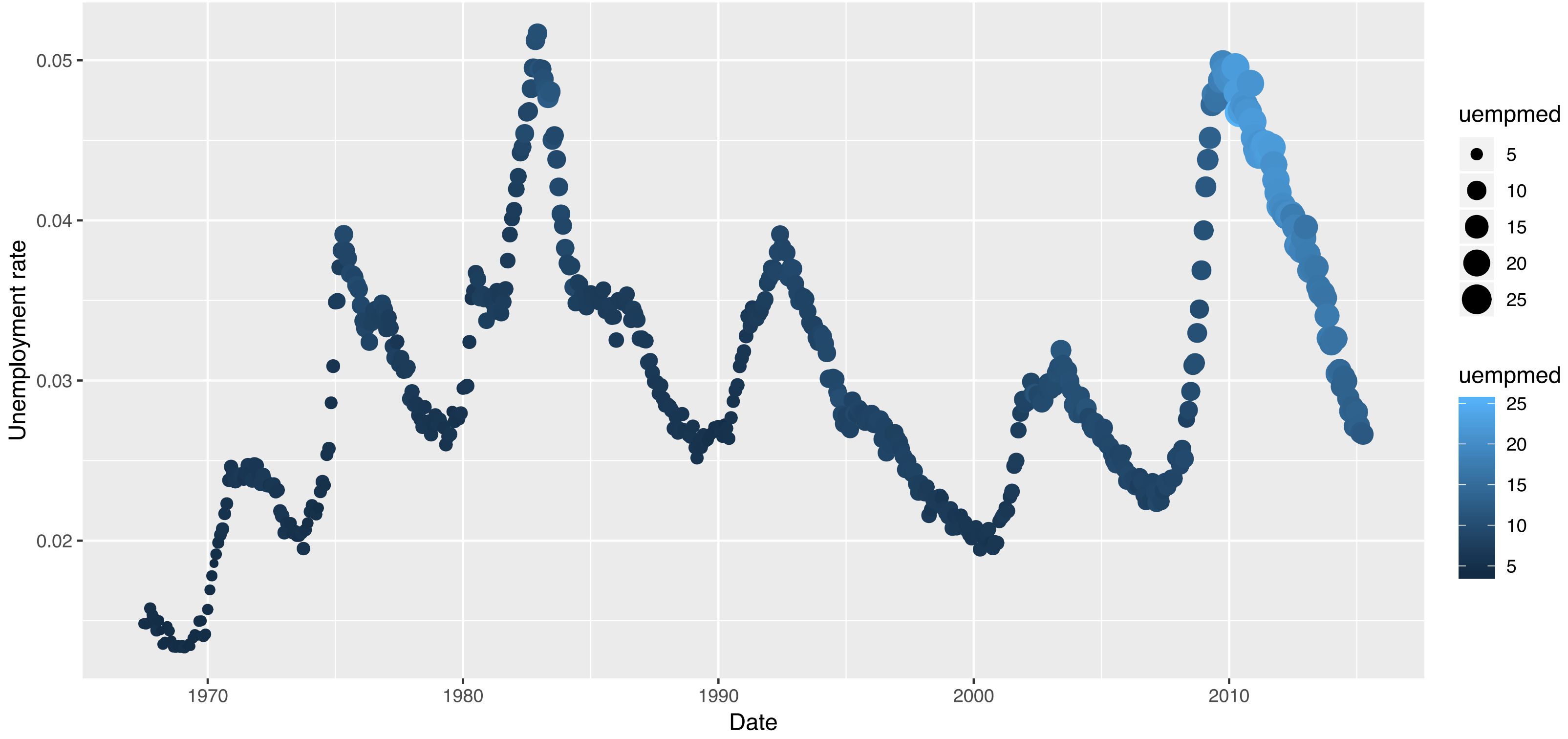
Change the `shape` of the points.

```r
ggplot(data = economics, aes(x = date, y = unemploy/pop, size = uempmed)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(shape = 1)
```



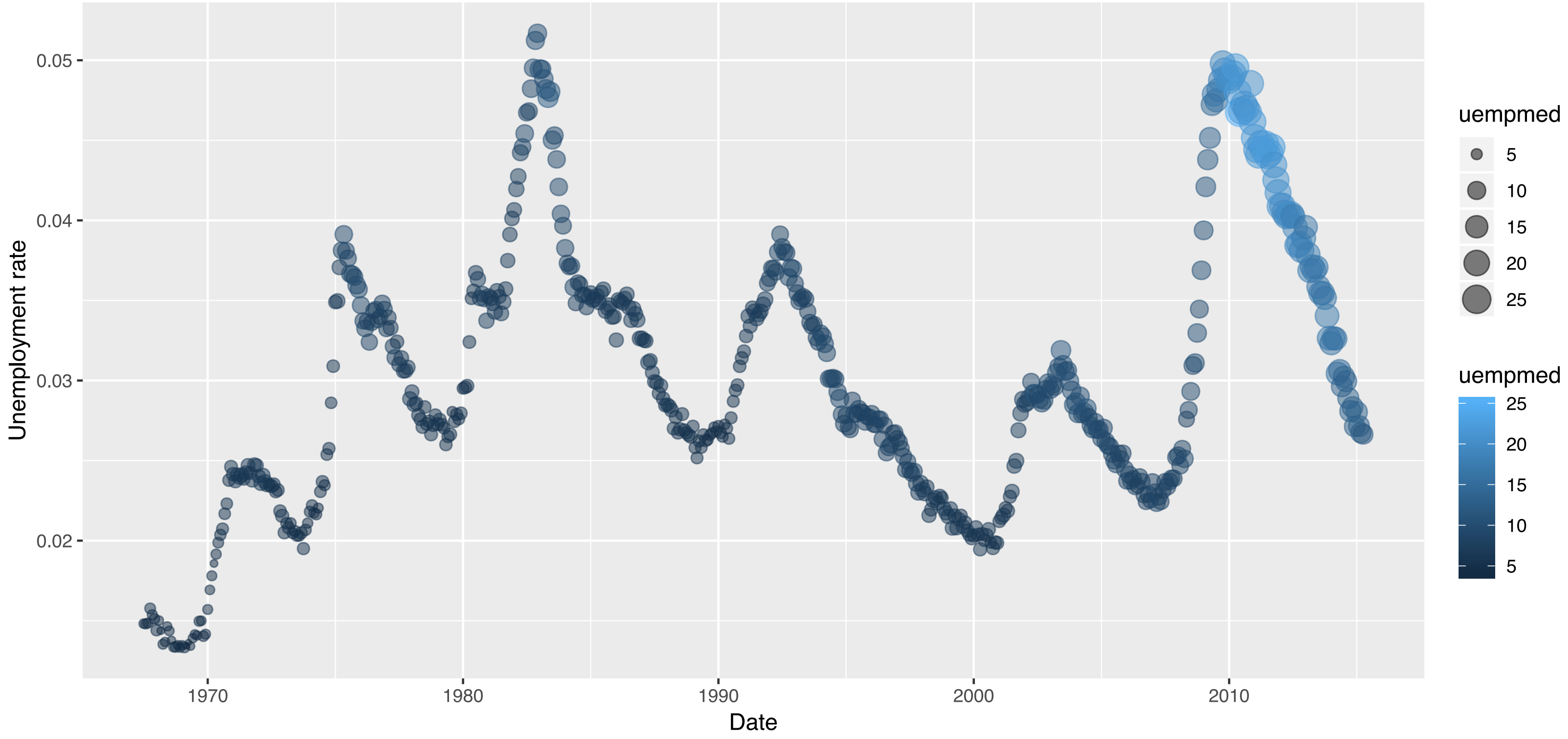
Map points' `color` to the median duration of unemployment.

```r
ggplot(data = economics, aes(x = date, y = unemploy/pop, size = uempmed)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed))
```



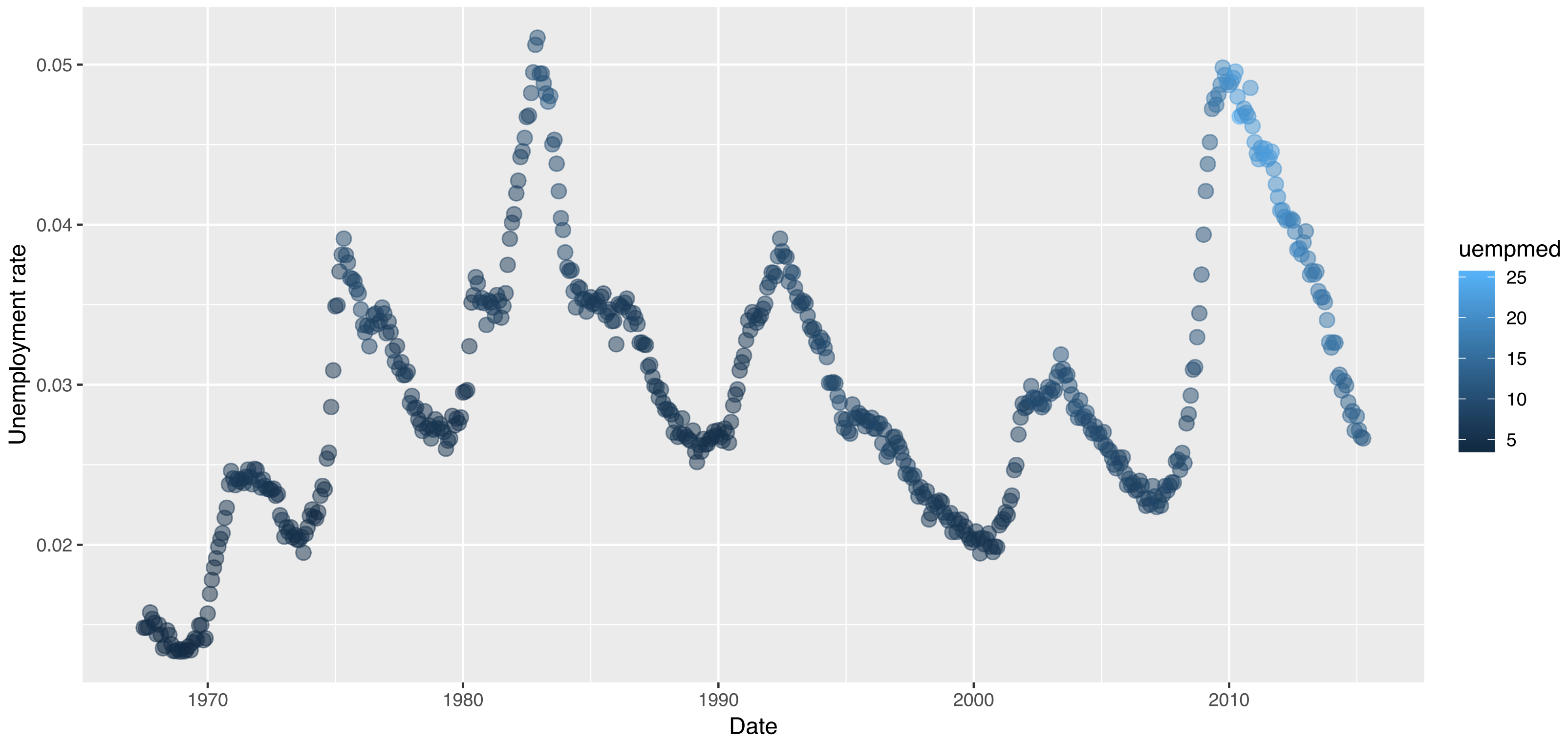
Add some transparency (`alpha`) to our points.

```
ggplot(data = economics, aes(x = date, y = unemploy/pop, size = uempmed)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5)
```



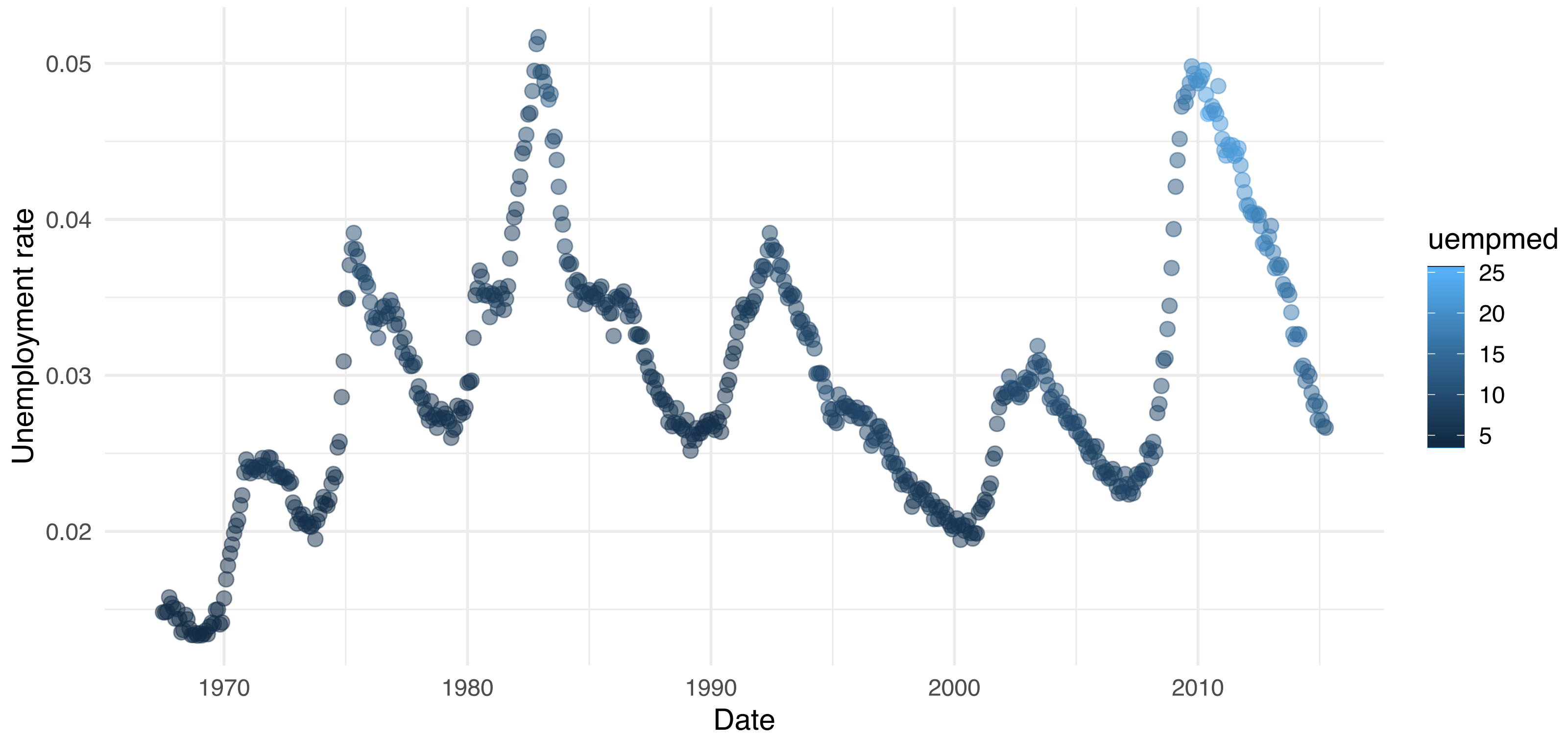
Same size points; all bigger.

```
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3)
```



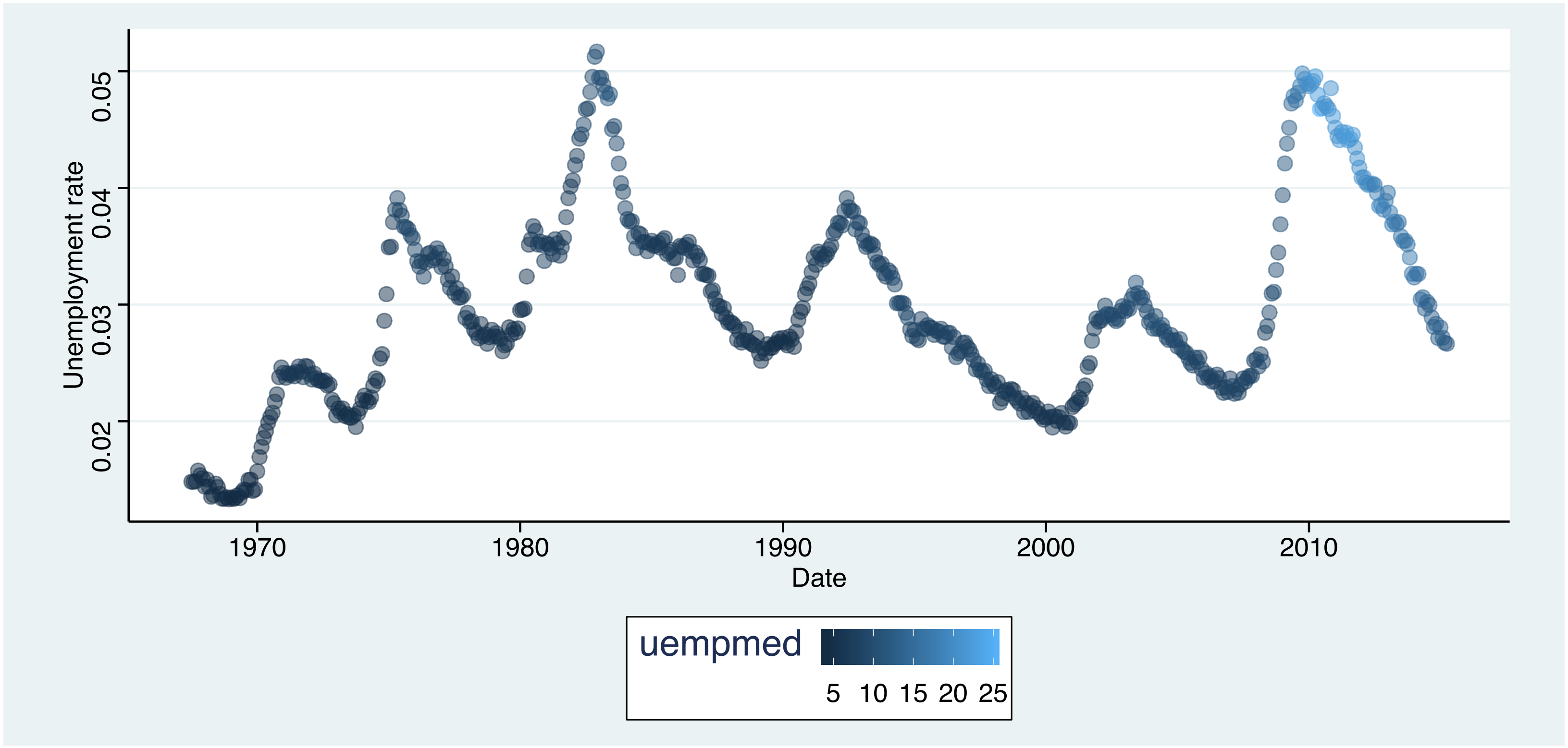
Change our theme—maybe you're a minimalist (but want slightly larger fonts)?

```r
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3) +
theme_minimal(base_size = 14)
```



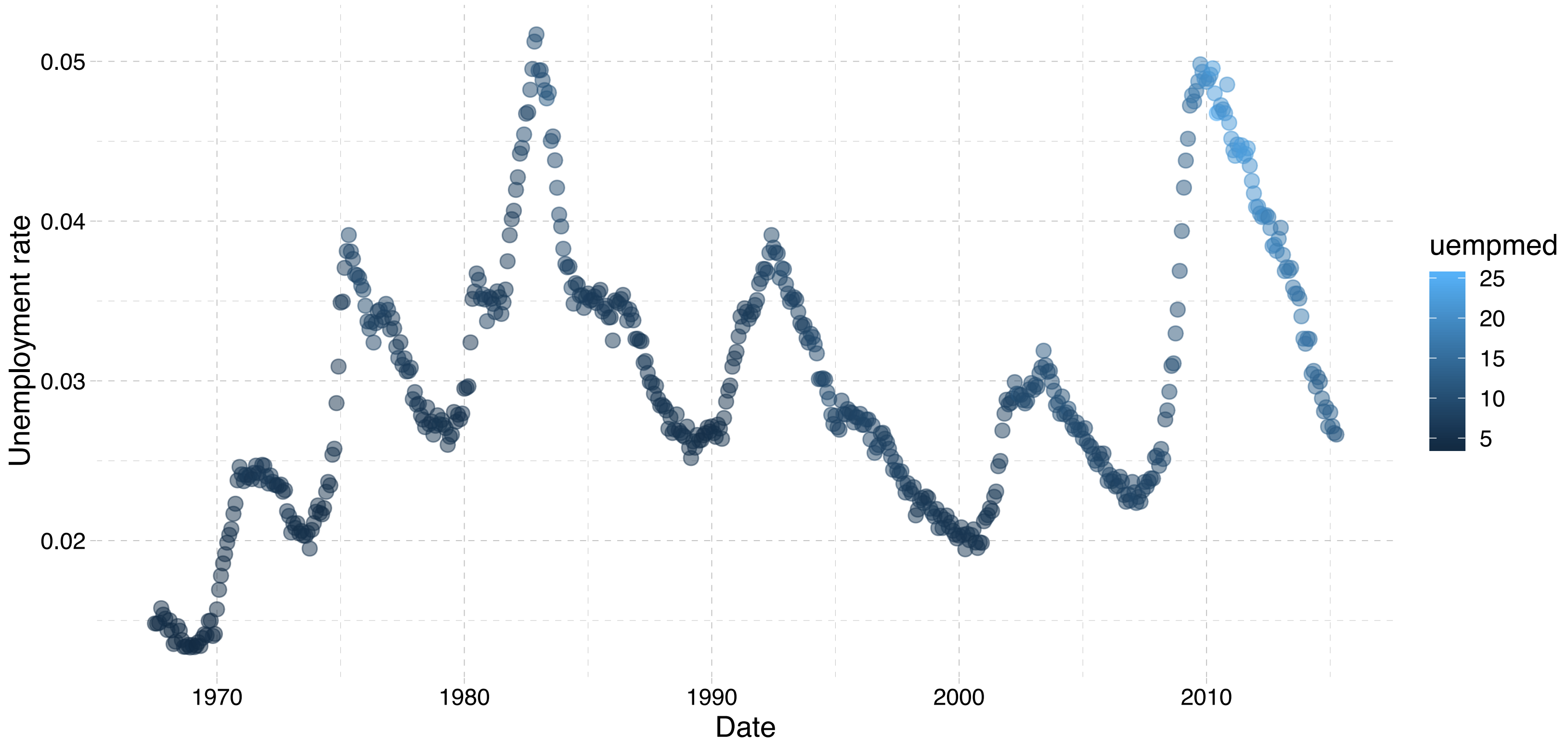
Want your figure to look like Stata made it?

```
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3) +
ggthemes::theme_stata(base_size = 14)
```



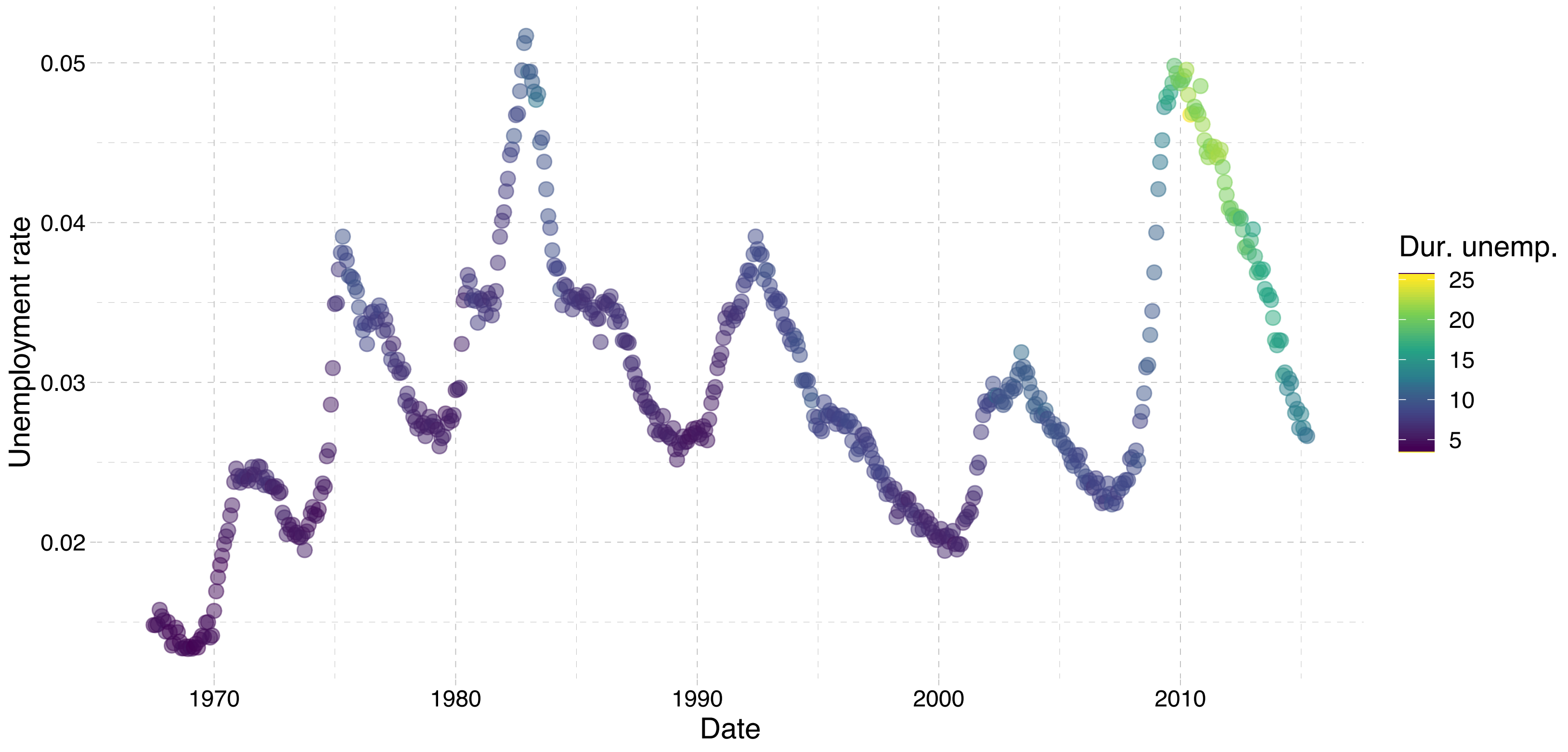
The "pander" theme from the `ggthemes` package.

```
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3) +
ggthemes::theme_pander(base_size = 14)
```



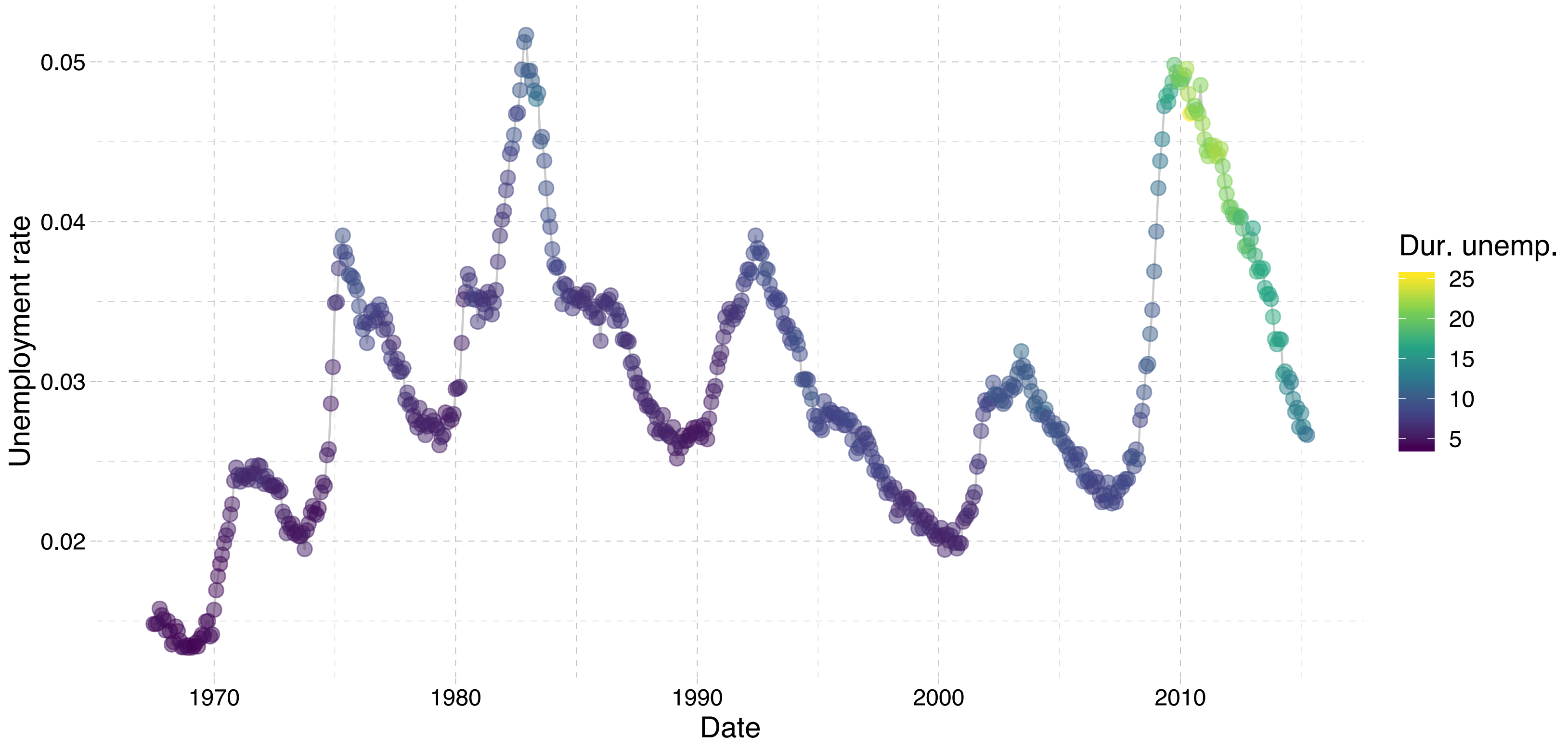
Change (and label) our color scale. *Note* `viridis` is the best.

```
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3) +
ggthemes::theme_pander(base_size = 14) +
scale_color_viridis_c("Dur. unemp.")
```



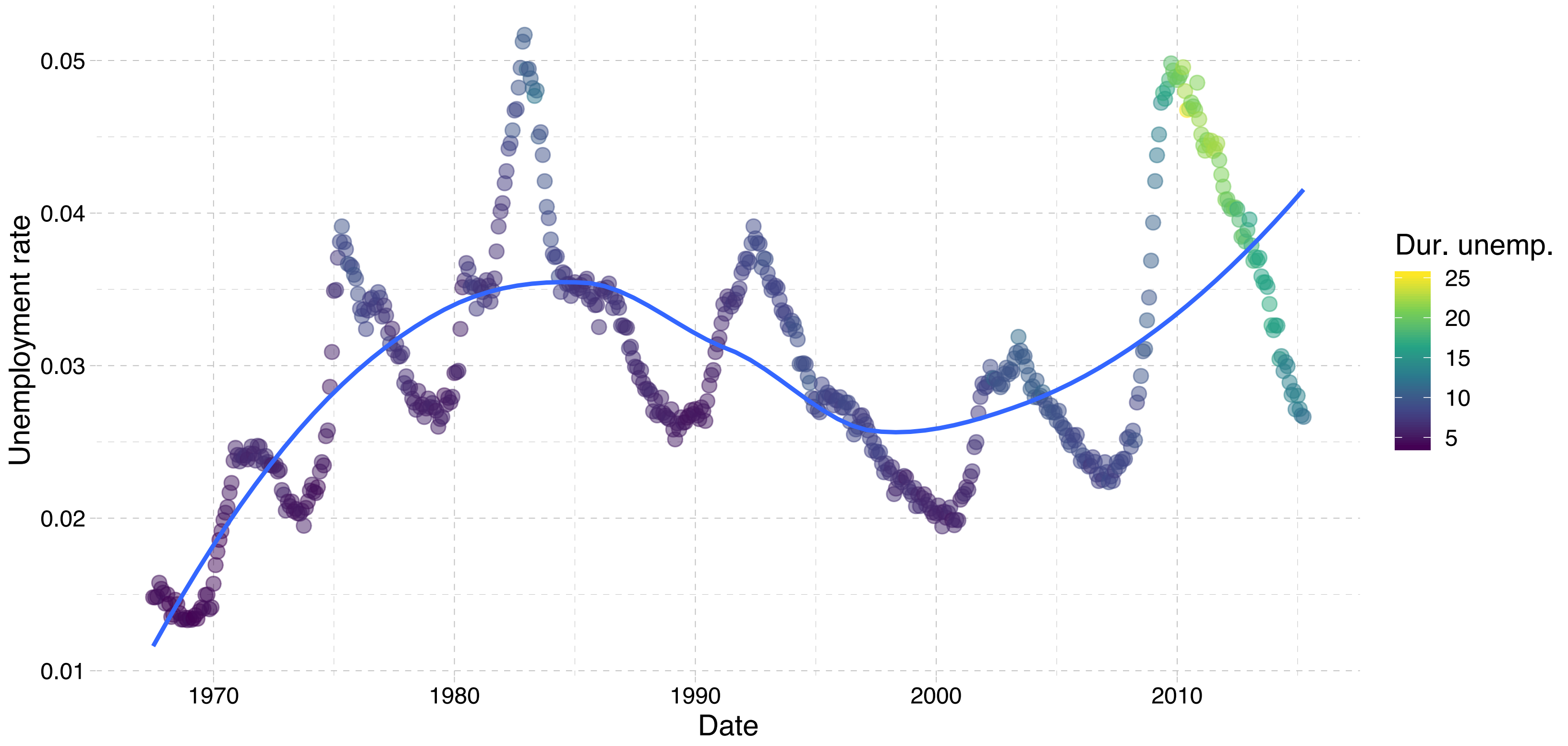
Connect the dots.

```r
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_line(color = "grey80") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3) +
ggthemes::theme_pander(base_size = 14) +
scale_color_viridis_c("Dur. unemp.")
```



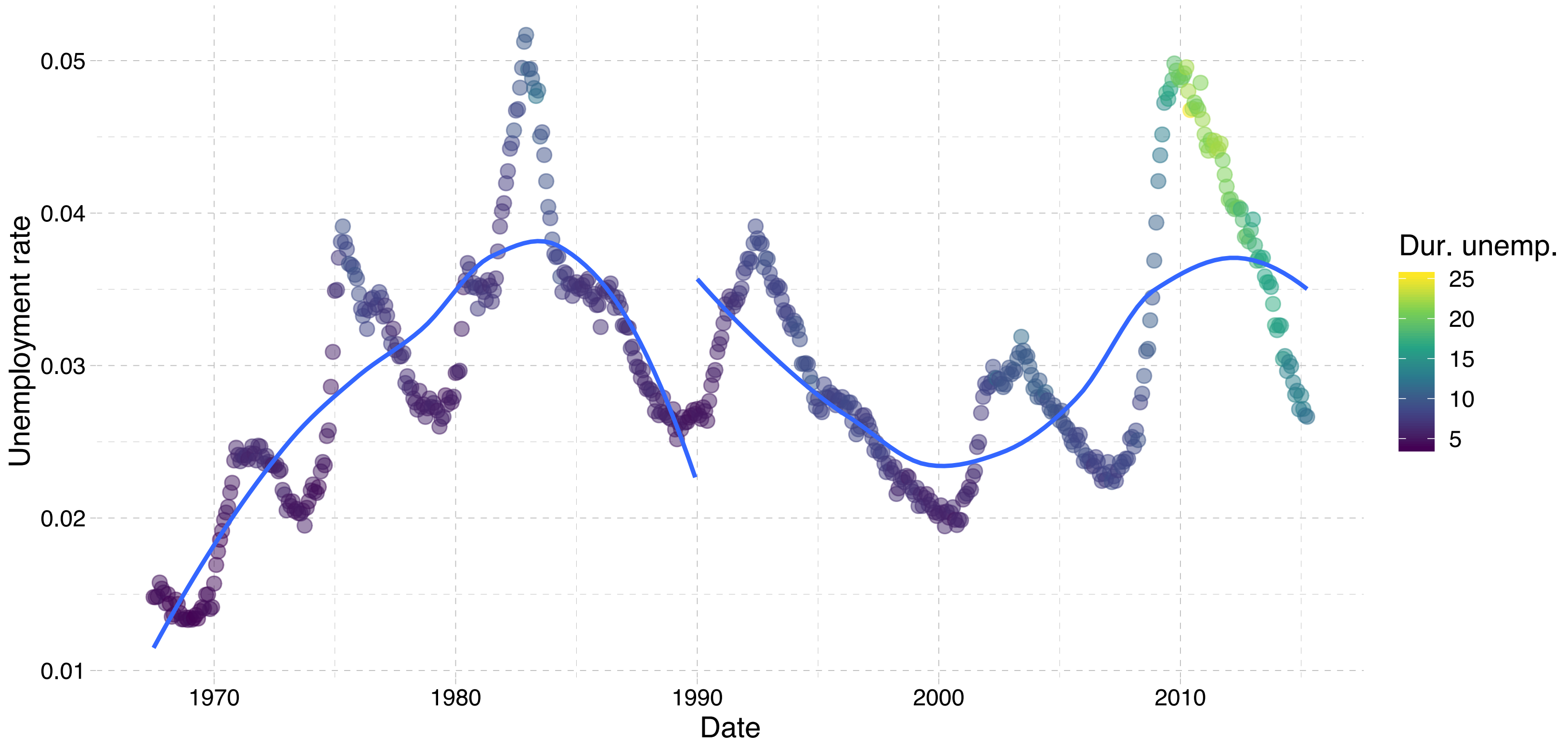
How about a smoother?

```
ggplot(data = economics, aes(x = date, y = unemploy/pop)) +
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3) +
geom_smooth(se = F) +
ggthemes::theme_pander(base_size = 14) +
scale_color_viridis_c("Dur. unemp.")
```



The `group` aesthetic separates groups.

```
ggplot(data = economics, aes(x = date, y = unemploy/pop, group = date < ymd(19900101)))
ylab("Unemployment rate") + xlab("Date") +
geom_point(aes(color = uempmed), alpha = 0.5, size = 3) +
geom_smooth(se = F) +
ggthemes::theme_pander(base_size = 14) +
scale_color_viridis_c("Dur. unemp.")
```

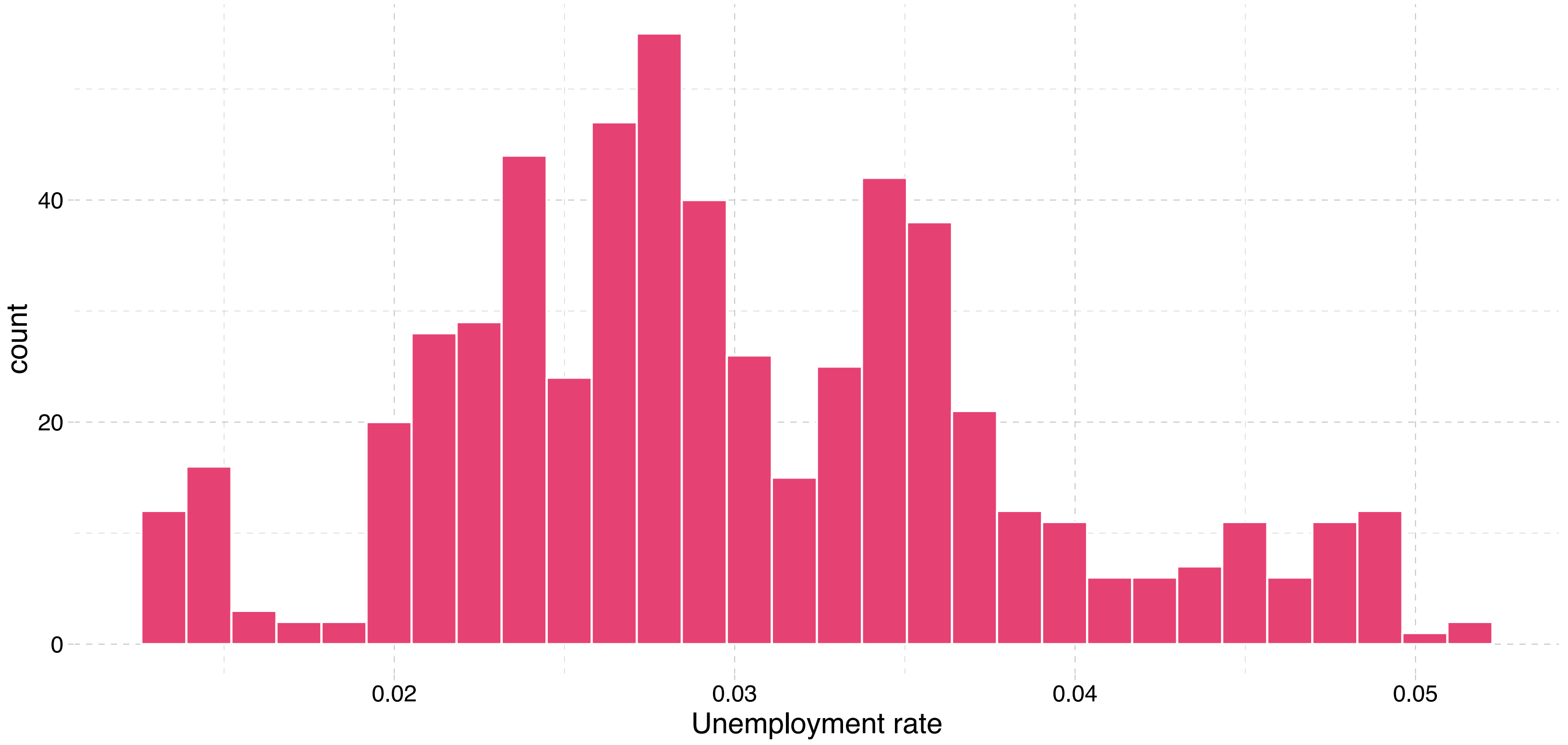


Note The `ymd()` function comes from the `lubridate` package.

`ggplot2` knows histogams.

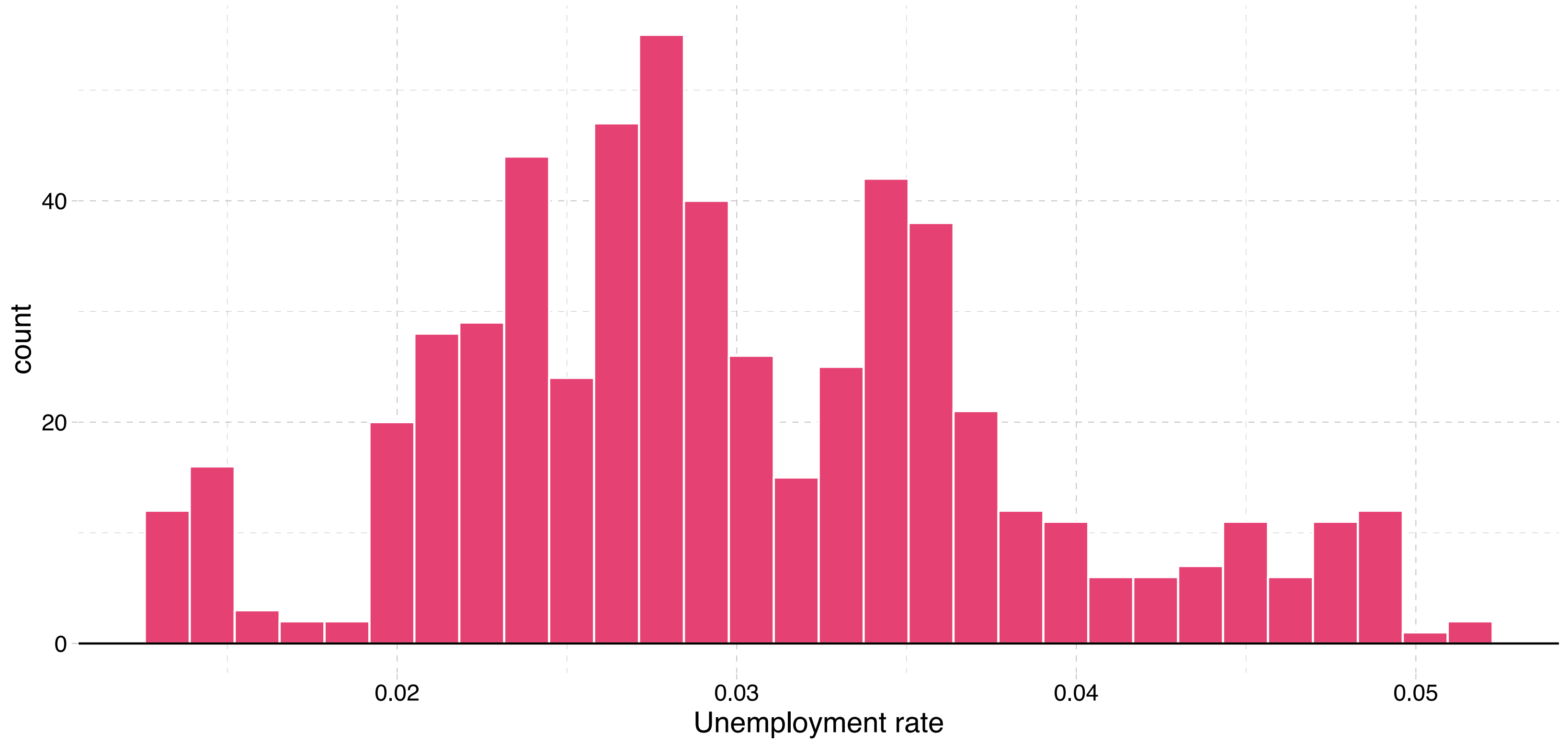
A histogram.

```
ggplot(data = economics, aes(x = unemploy/pop)) +
xlab("Unemployment rate") +
geom_histogram(color = "white", fill = "#e64173") +
ggthemes::theme_pander(base_size = 14)
```



Add a horizontal line where count = 0.

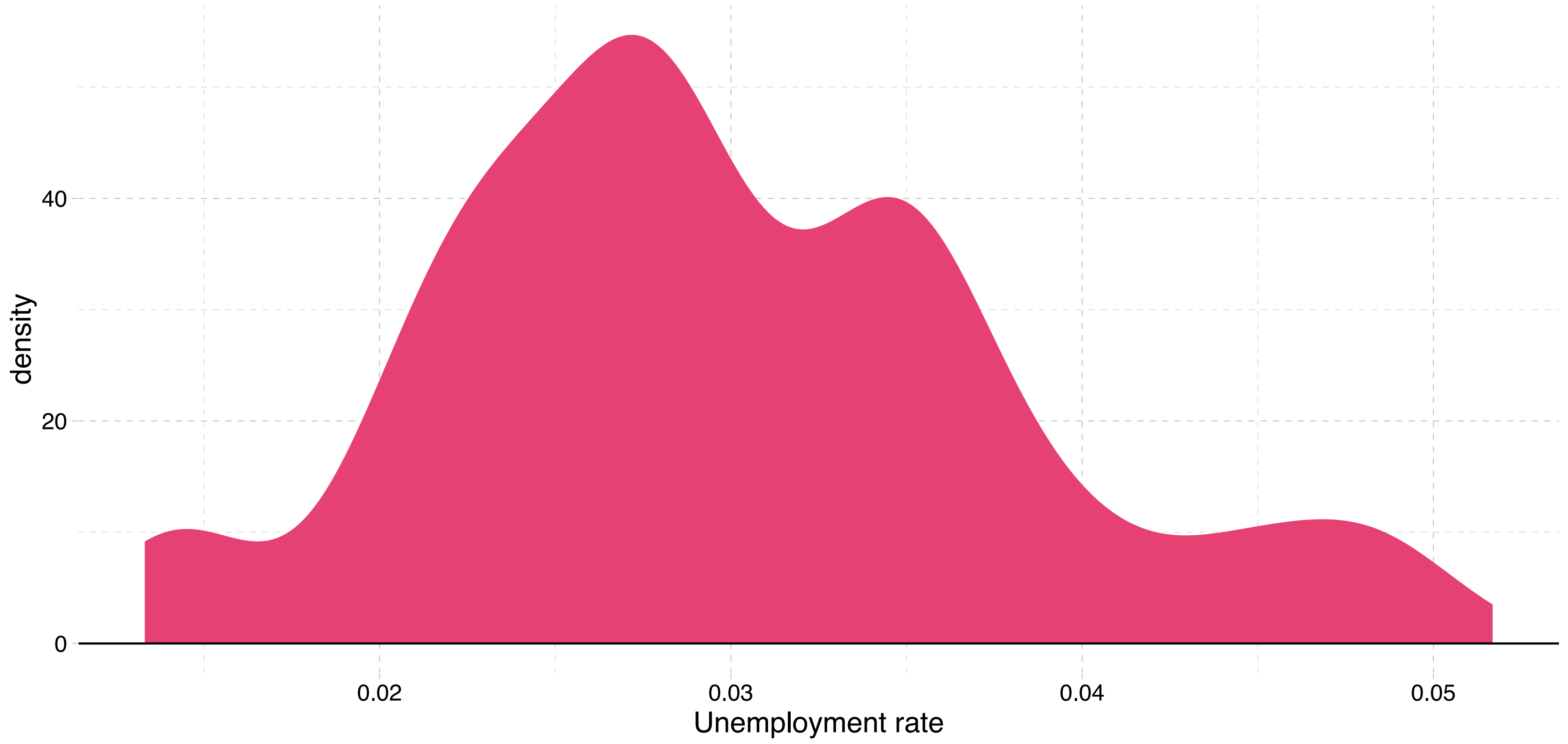
```r
ggplot(data = economics, aes(x = unemploy/pop)) +
xlab("Unemployment rate") +
geom_histogram(color = "white", fill = "#e64173") +
geom_hline(yintercept = 0) +
ggthemes::theme_pander(base_size = 14)
```



`ggplot2` knows densities.

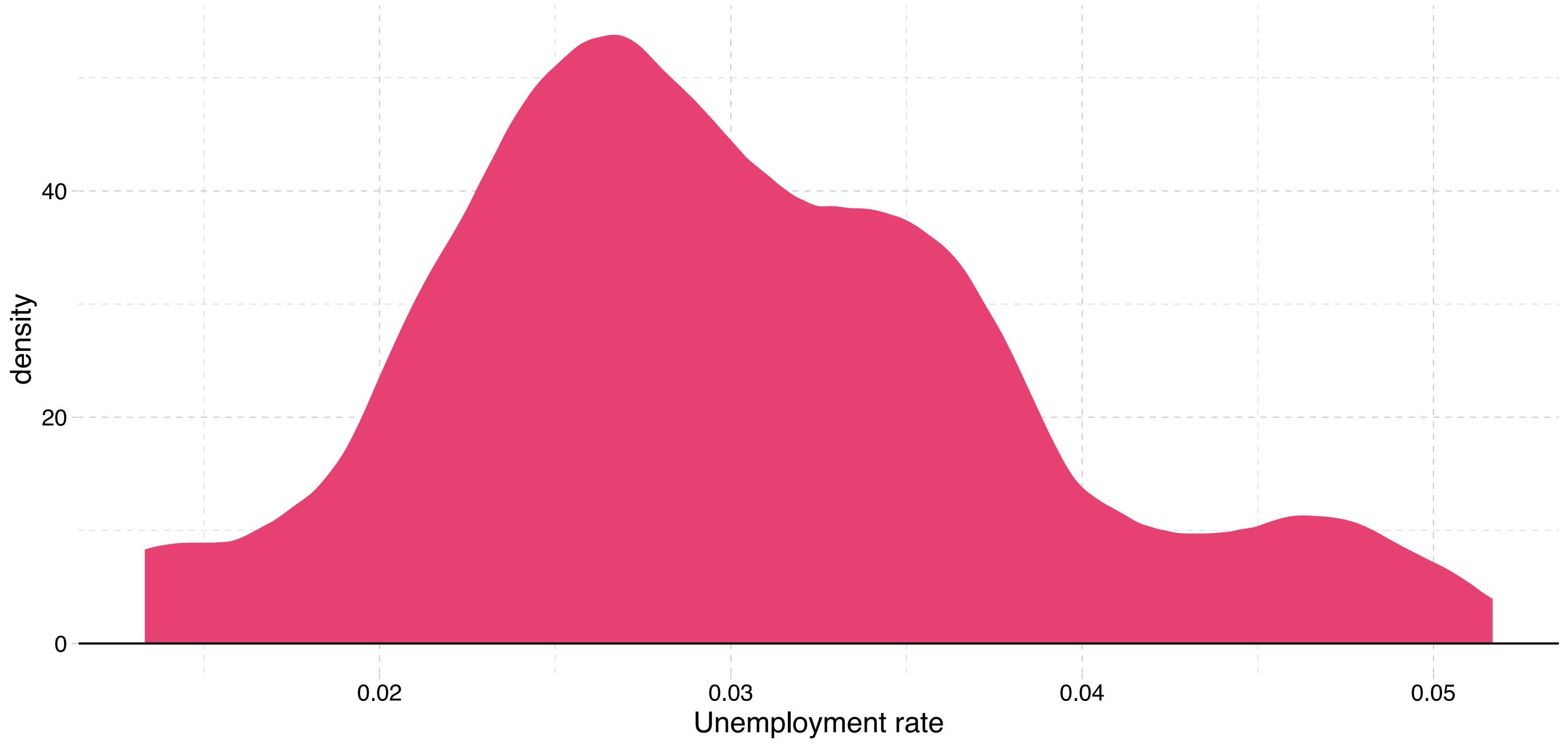
A density plot.

```r
ggplot(data = economics, aes(x = unemploy/pop)) +
xlab("Unemployment rate") +
geom_density(color = NA, fill = "#e64173") +
geom_hline(yintercept = 0) +
ggthemes::theme_pander(base_size = 14)
```



Now with Epanechnikov kernel!

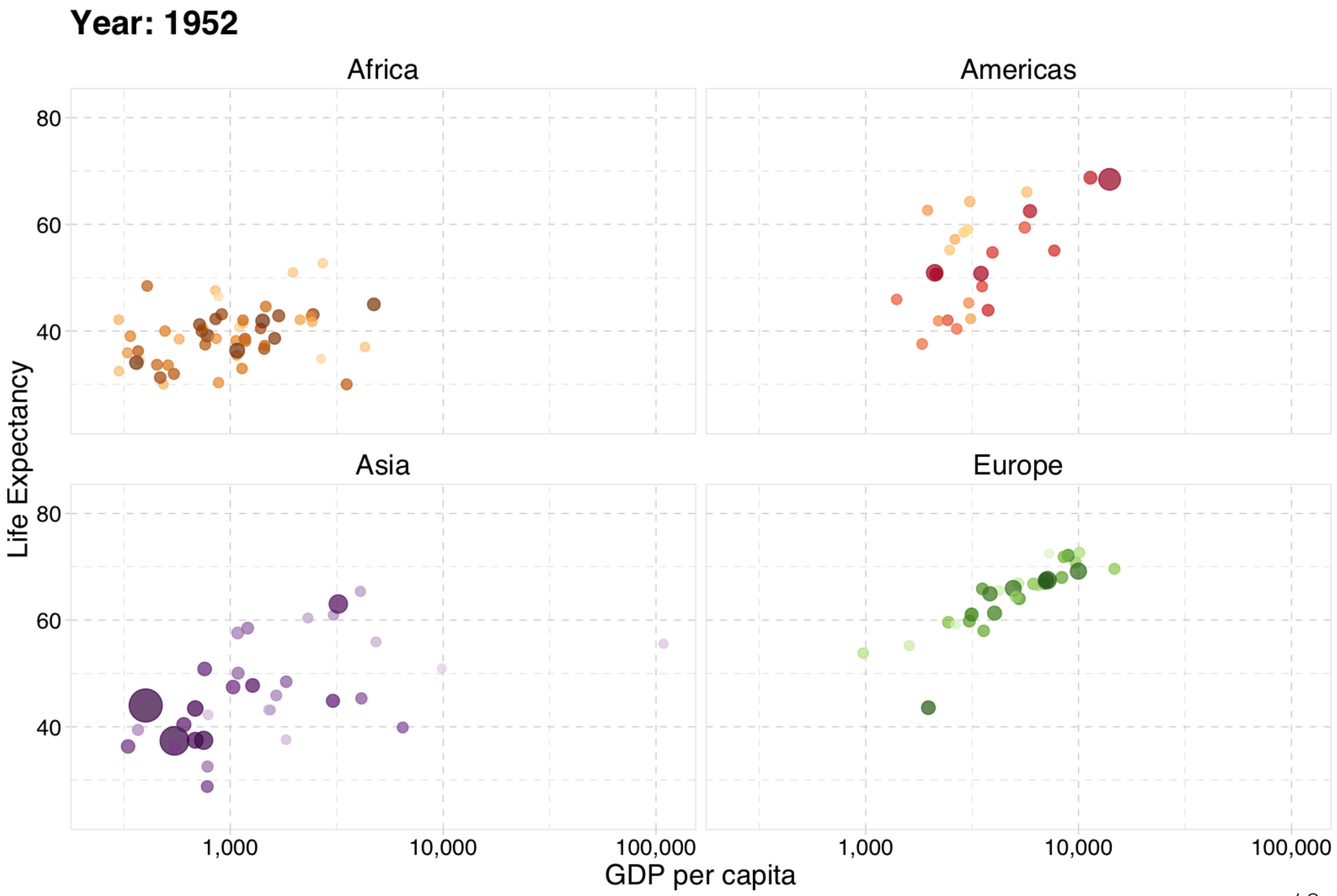
```
ggplot(data = economics, aes(x = unemploy/pop)) +
xlab("Unemployment rate") +
geom_density(kernel = "epanechnikov", color = NA, fill = "#e64173") +
geom_hline(yintercept = 0) +
ggthemes::theme_pander(base_size = 14)
```



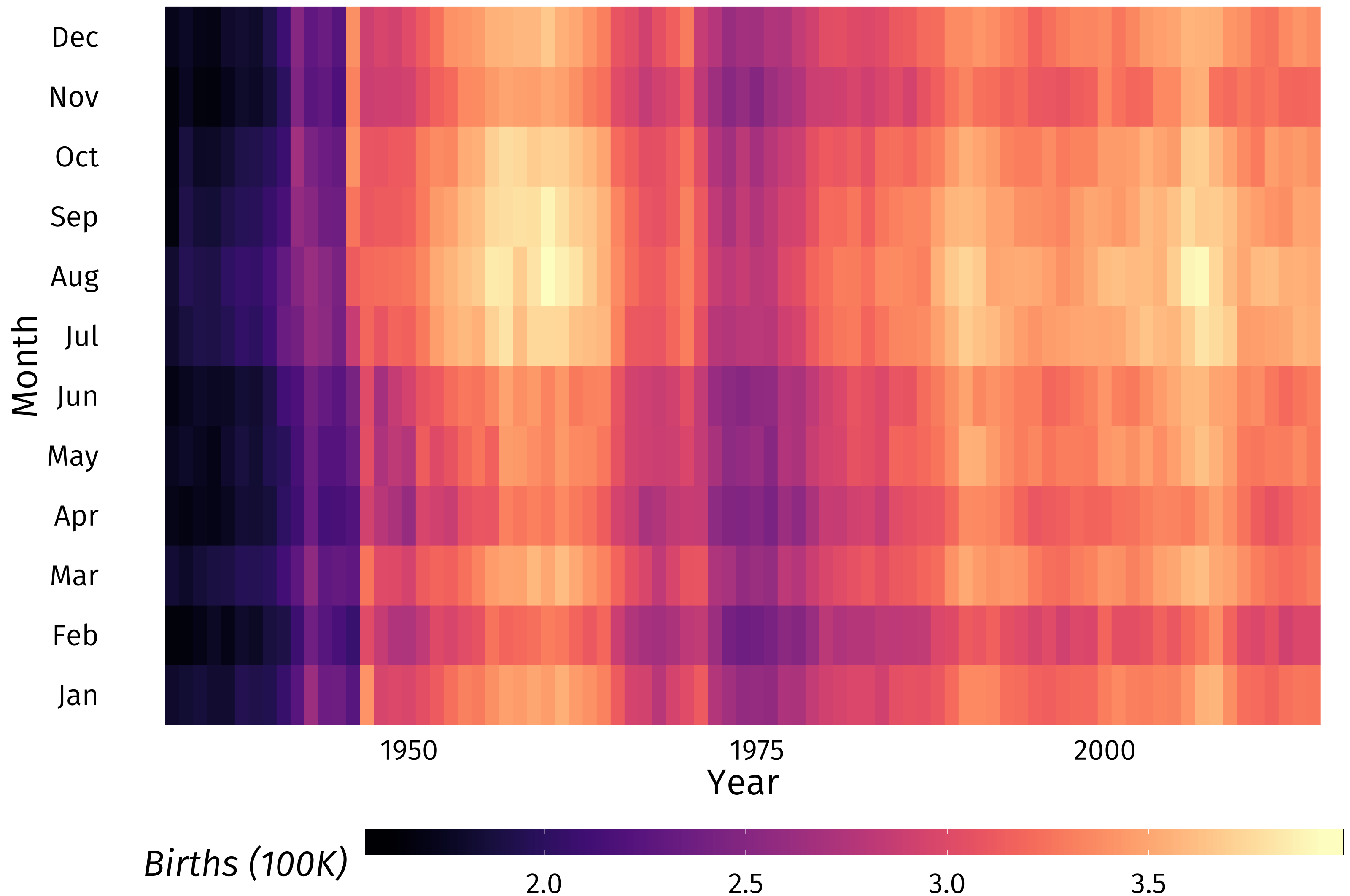
`ggplot2` itself is incredibly flexible/powerful.

But there are even more packages that extend its power—*e.g.*, `ggthemes`, `gganimate`, `cowplot`, `ggmap`, `ggExtra`, and (of course) `viridis`.

Gapminder meets `gganimate`



US births by month since 1933



ggplot2

Saving plots

You can save your `ggplot2`-based figures using `ggsave()`.

ggplot2

`ggsave()` Option 1

By default, `ggsave()` saves the last plot printed to the screen.

```
# Create a simple scatter plot
ggplot(data = fun_df, aes(x = x, y = y)) +
geom_point()
# Save our simple scatter plot
ggsave(filename = "simple_scatter.pdf")
```

ggplot2

`ggsave()` Option 1

By default, `ggsave()` saves the last plot printed to the screen.

```r
# Create a simple scatter plot
ggplot(data = fun_df, aes(x = x, y = y)) +
geom_point()
# Save our simple scatter plot
ggsave(filename = "simple_scatter.pdf")
```

Notes

- This example creates a PDF. Change to `".png"` for PNG, *etc.*
- There several helpful, optional arguments: `path`, `width`, `height`, `dpi`.

ggplot2

`ggsave()` Option 2

You can assign your `ggplot()` objects to memory

```r
# Create a simple scatter plot named 'gg_points'
gg_points ← ggplot(data = fun_df, aes(x = x, y = y)) +
geom_point()
```

ggplot2

`ggsave()` Option 2

You can assign your `ggplot()` objects to memory

```
# Create a simple scatter plot named 'gg_points'
gg_points ← ggplot(data = fun_df, aes(x = x, y = y)) +
geom_point()
```

We can then save this figure with the name `gg_points` using `ggsave()`

```
# Save our simple scatter plot name 'ggsave'
ggsave(
  filename = "simple_scatter.pdf",
  plot = gg_points
)
```

Resources

There's always more

`ggplot2`

1. `RStudio`'s cheat sheet for `ggplot2`.
2. `ggplot2` reference index
3. The `tidyverse` page on `ggplot2`.
4. Hadley Wickham's on *Data visualization* in his data science book.

Table of contents

Default options

ggplot2