ggvis tutorial

Winston Chang
RStudio
R Day at Strata NYC 2014

Slides and code: http://bit.ly/rday-strata14

What is ggvis?

A new package for interactive data visualization - a synthesis of ideas:

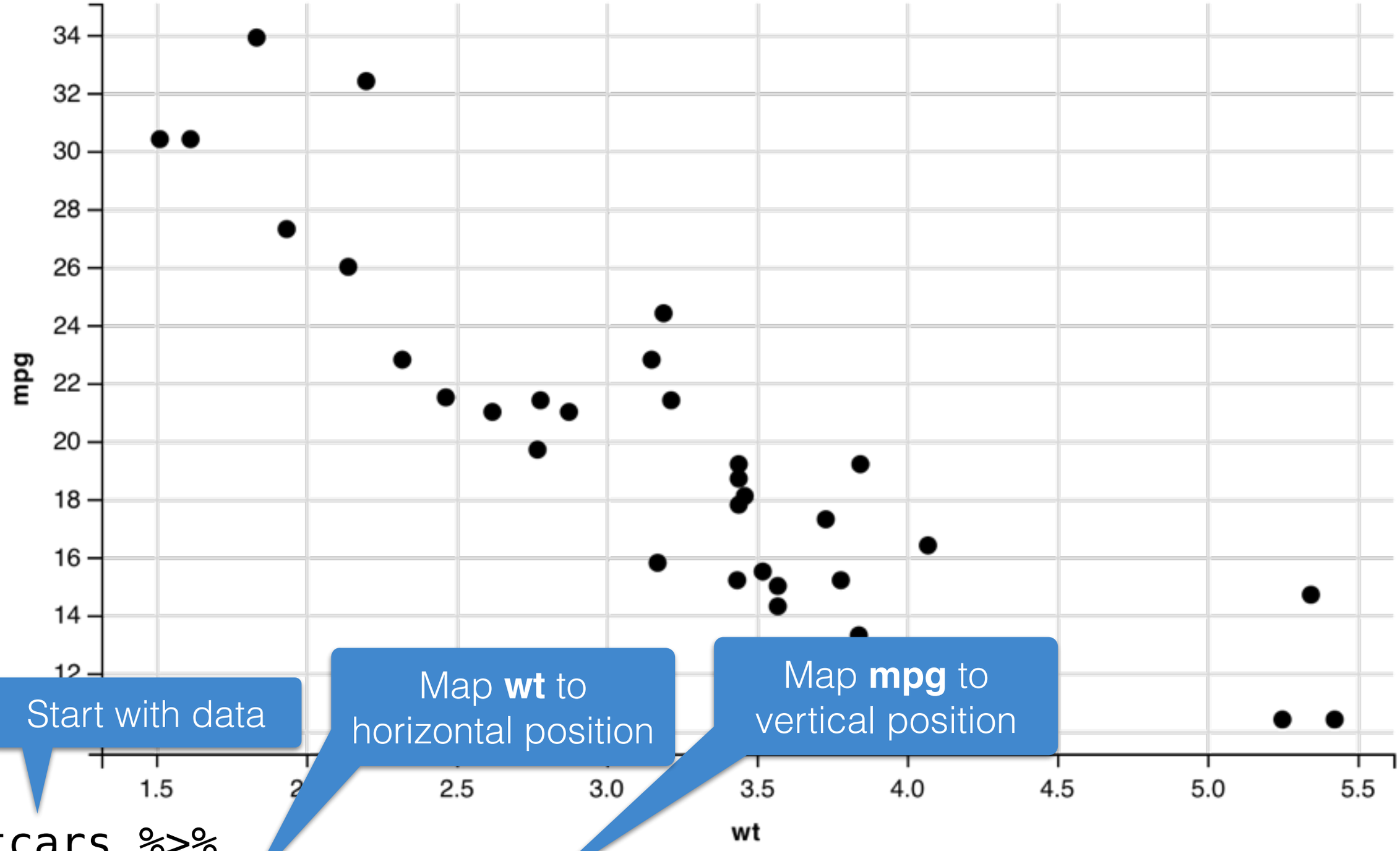
- Grammar of graphics (ggplot2)
- Reactivity (Shiny)
- Data pipeline (dplyr)
- Of the web (vega.js)

Basic plots

Getting started

```
# Load packages
library(dplyr)
library(ggvis)

# Inspect some data sets
str(mtcars)
str(cocaine)
```



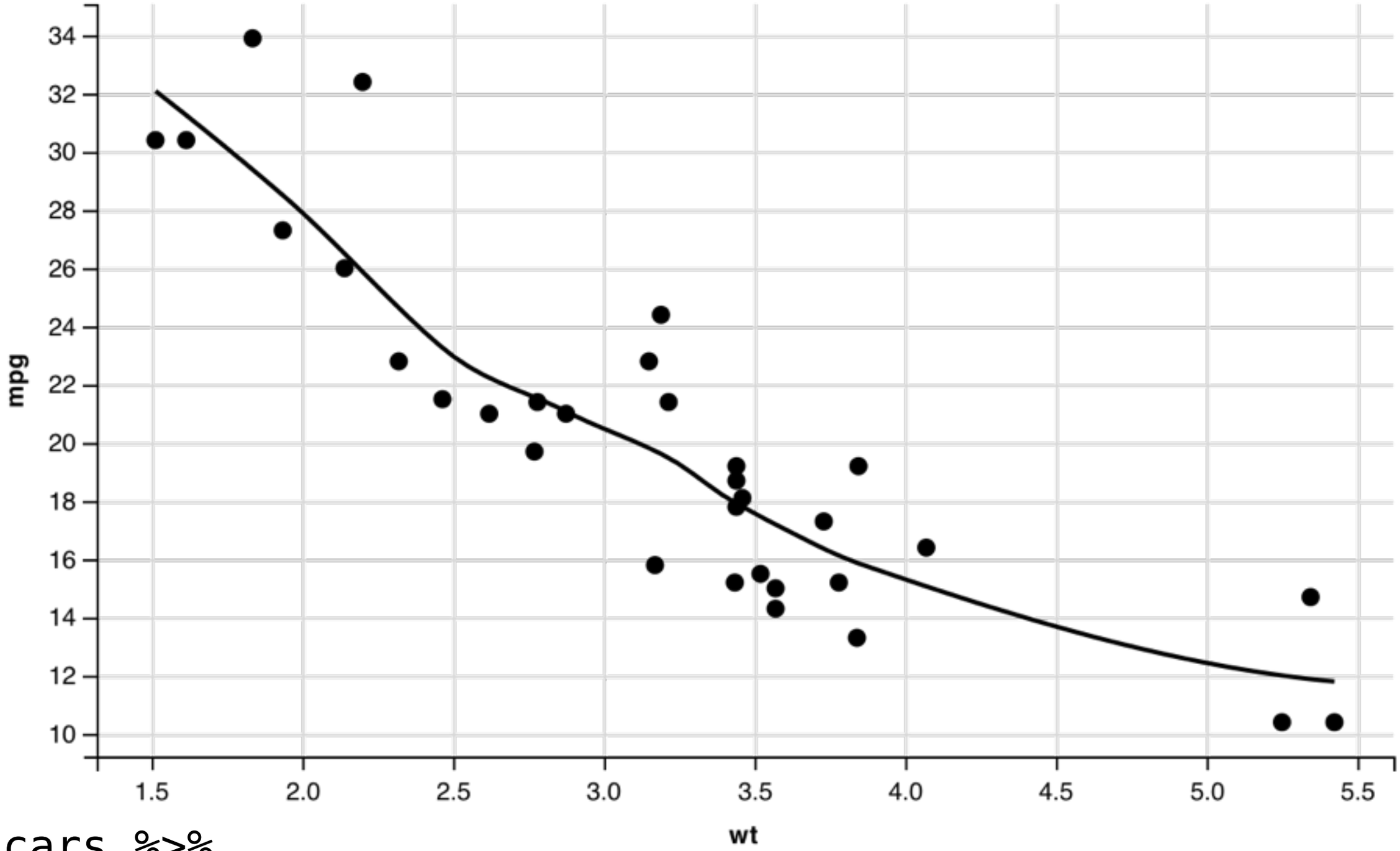
Start with data

Map **wt** to horizontal position

Map **mpg** to vertical position

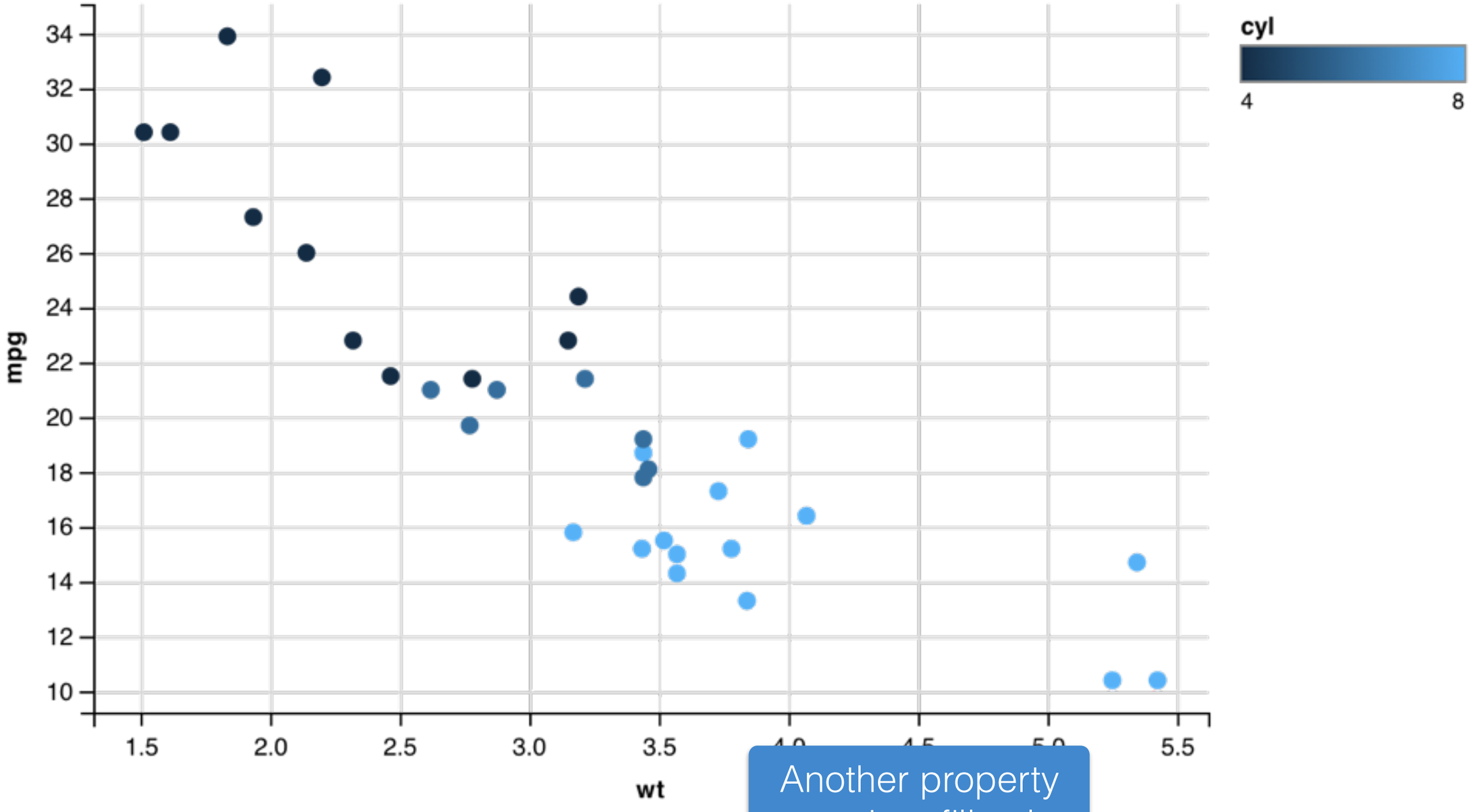
```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points()
```

Add a layer of points



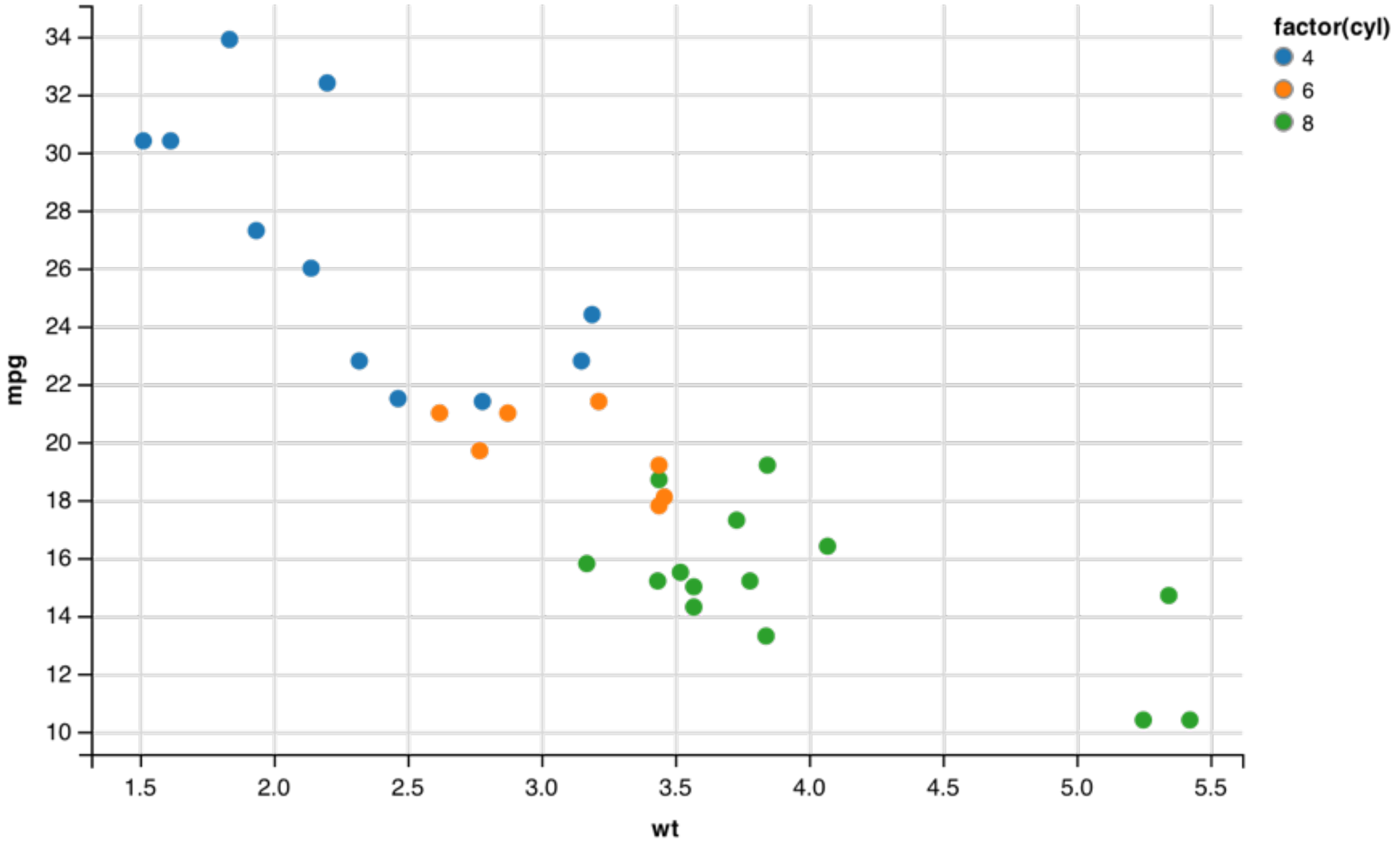

```
mtcars %>%
   ggvis(x = ~wt, y = ~mpg) %>%
   layer_points() %>%
   layer_smooths()
```

Also add a layer of smoothing lines



Another property mapping: fill color

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg, fill = cyl) %>%
  layer_points()
```




```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg, fill = ~factor(cyl)) %>%
  layer_points()
```

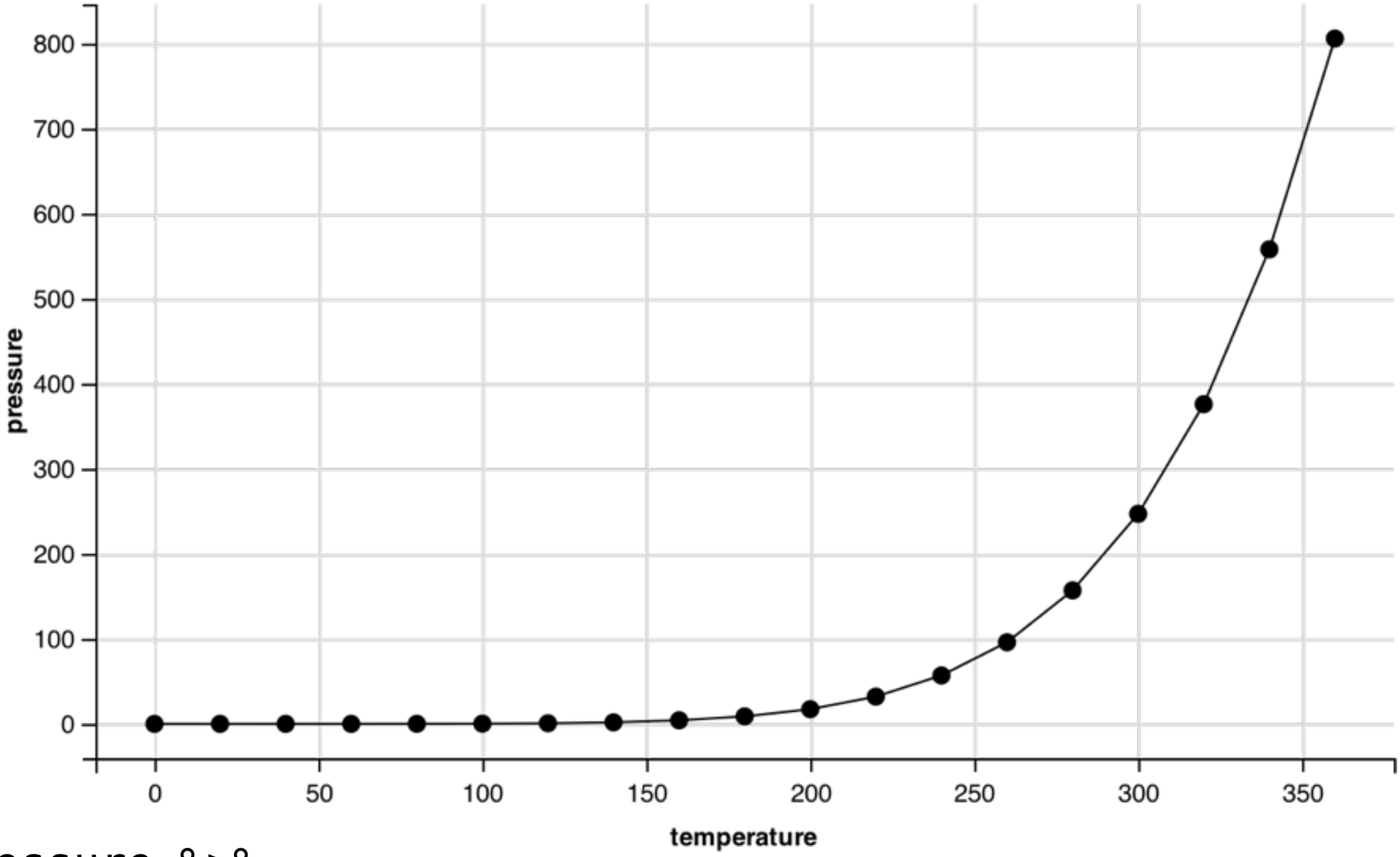
Use factor() to treat cyl as categorical

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points(x = ~wt, y = ~mpg)
```

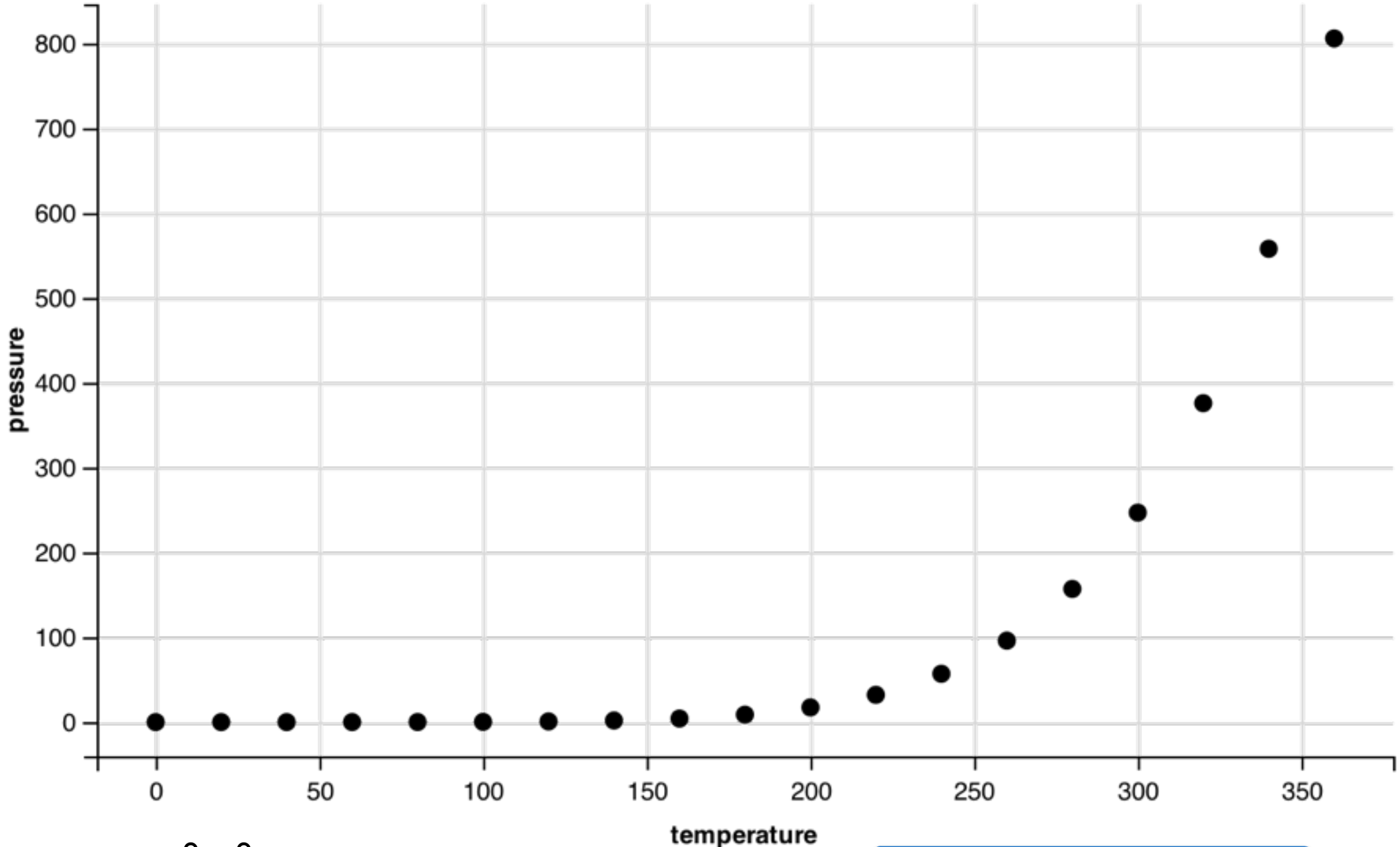
```
mtcars %>%
  ggvis(~wt, ~mpg) %>%
  layer_points()
```

x and y are defaults

layers inherit property mappings

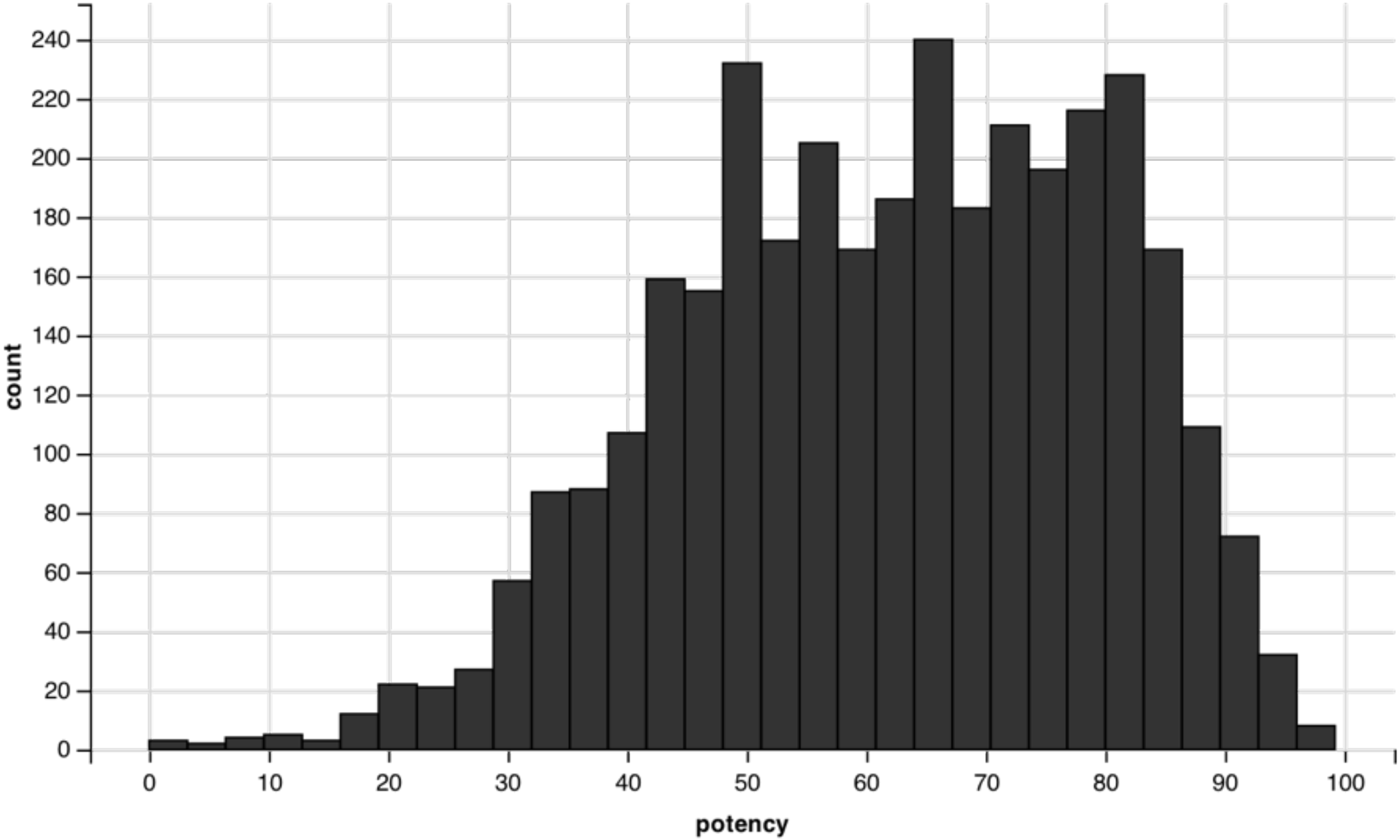



```
pressure %>%
  ggvis(~temperature, ~pressure) %>%
  layer_lines() %>%
  layer_points()
```




```
pressure %>%
  ggvis(~temperature, ~pressure)
```

When no layer specified, ggvis will guess




```
cocaine %>% ggvis(x = ~potency) %>% layer_histograms()

cocaine %>% ggvis(~potency)

cocaine %>% ggvis(x = ~potency) %>% layer_histograms(width = 2)
```

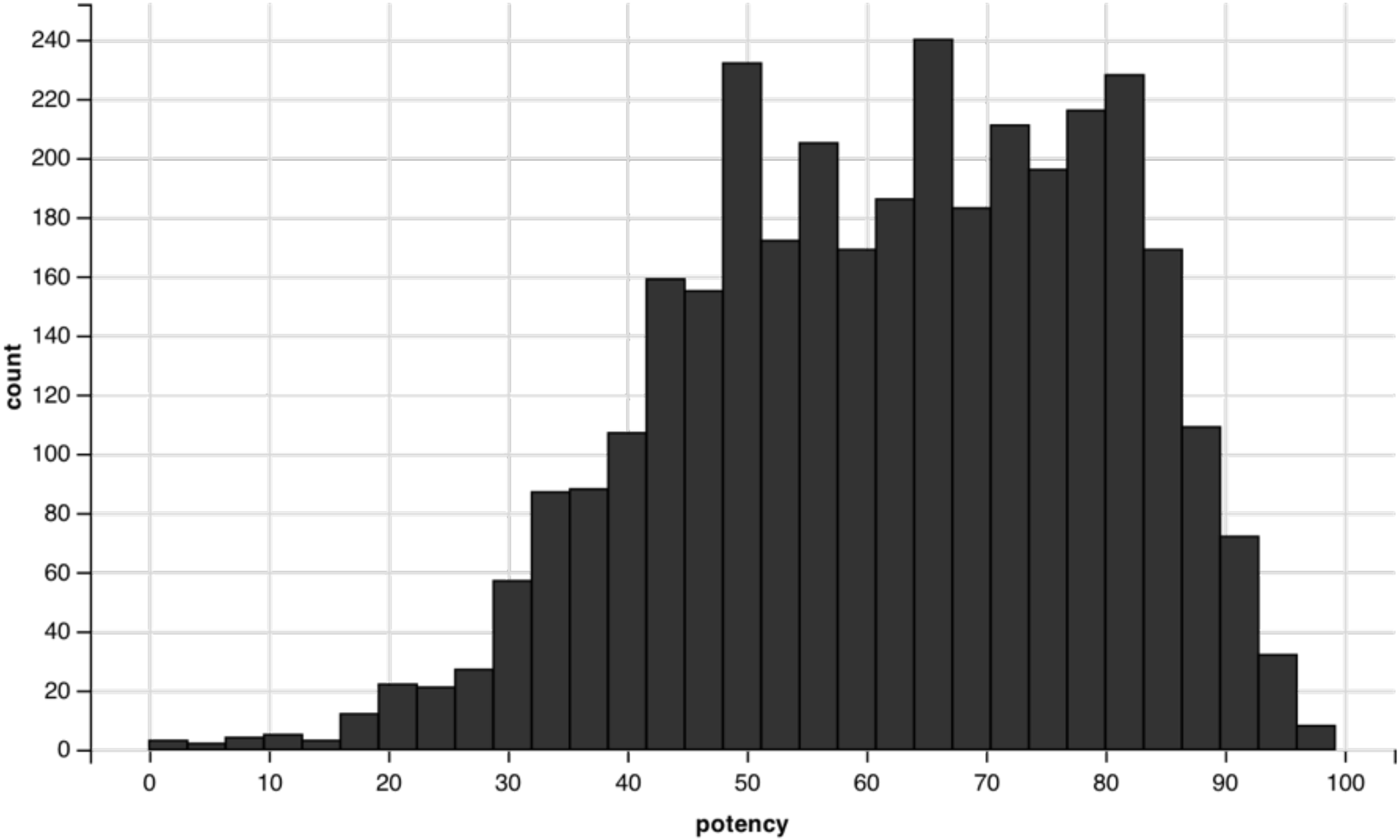
Can also change width of bins

Exercises

- Make scatter plots from the `iris` data set.
- Try adding smoothing lines
- Try mapping other variables to `size`, `stroke`, and `fill`.
- With a histogram of the `price` variable from the `cocaine` data set, experiment with different values of `width`. What do they reveal about the data?

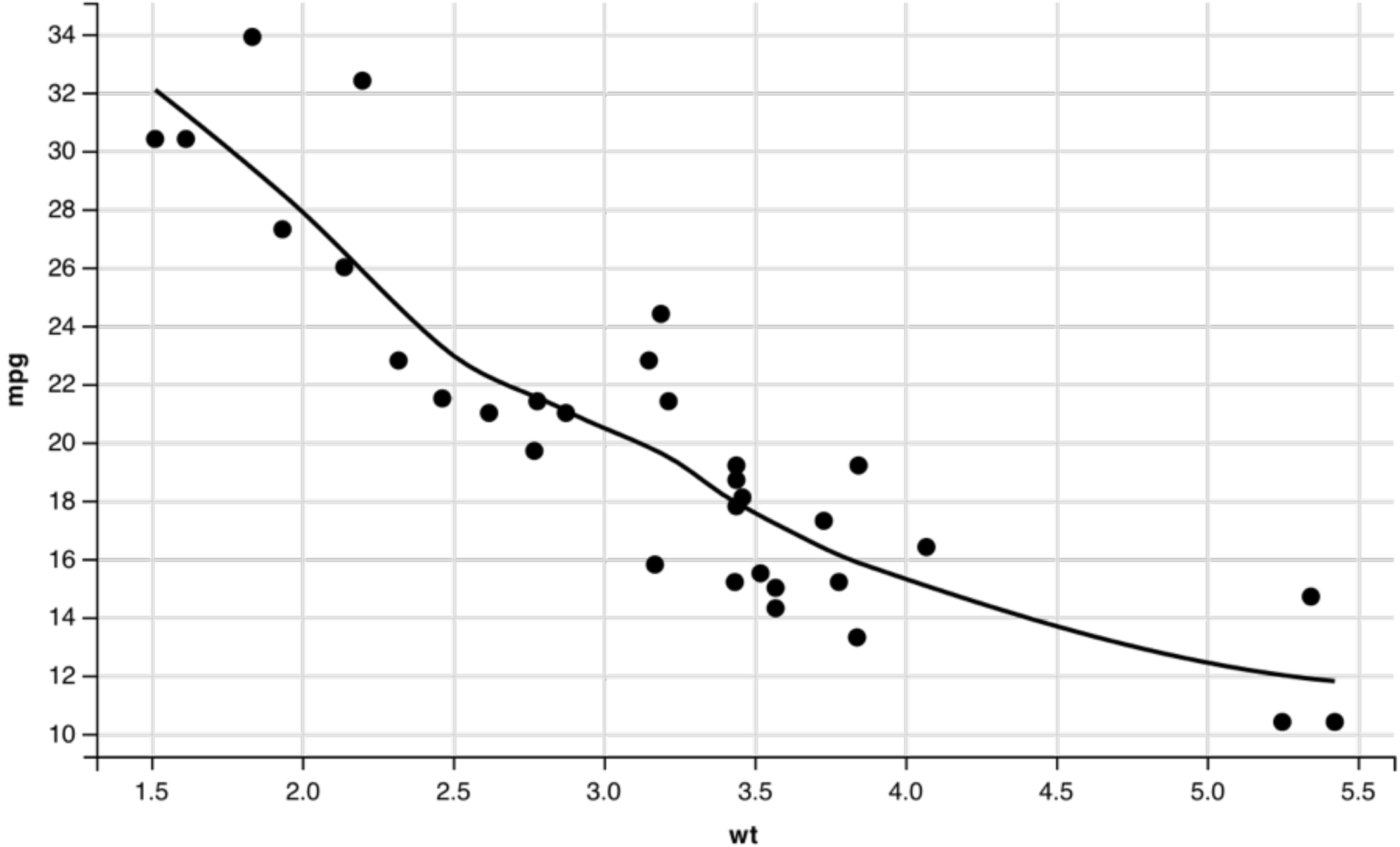
Basic and compound layers

- Basic layers consist of just the marks on the plot.
  - `layer_points()`
  - `layer_paths()`
  - `layer_rects()`
- Compound layers consist of a computation and a mark.
  - `layer_histograms()`
  - `layer_smooths()`
  - `layer_model_predictions()`



layer_histograms():

```
  compute_bin()
  layer_rects()
```



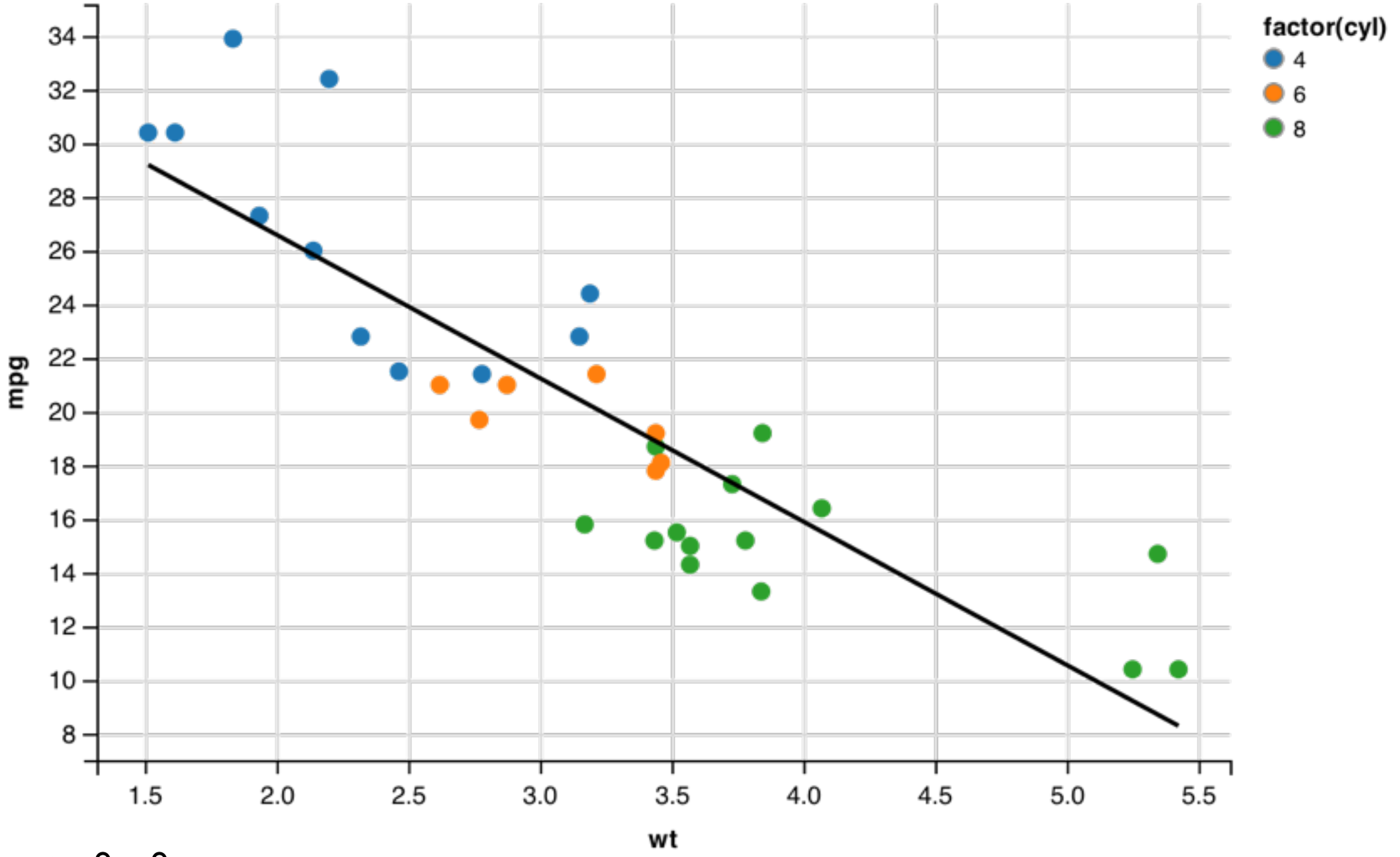
layer_smooths():

```
  compute_model_prediction(model="loess")
  layer_paths()
```

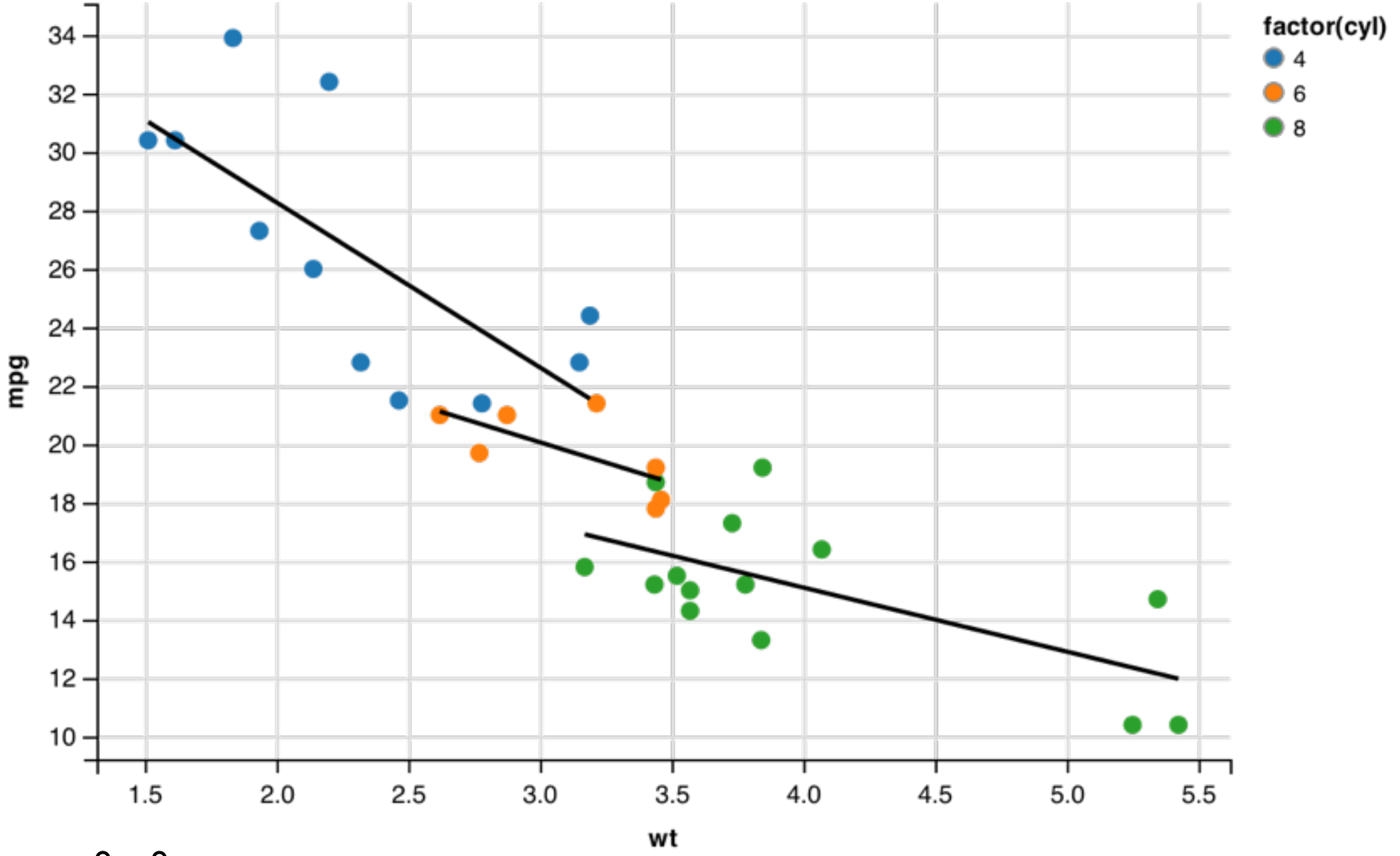
Grouping

- Sometimes layers with computations need to be told how to group the data.

- Usually ggvis tries to be smart about grouping, but it doesn't always get it right.

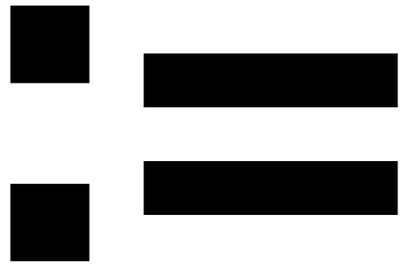



```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg, fill = ~factor(cyl)) %>%
  layer_points() %>%
  layer_model_predictions(model = "lm")
```

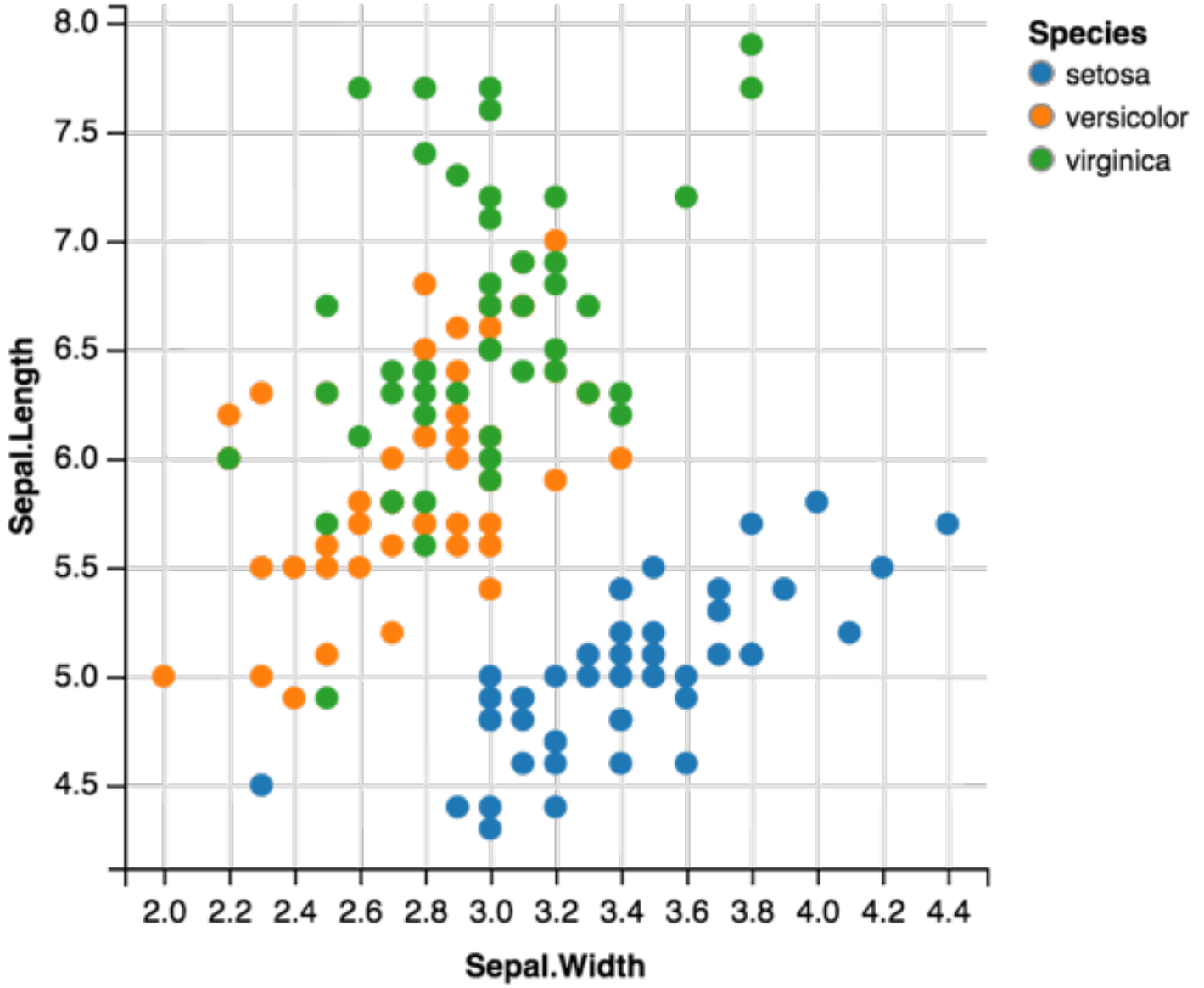



```
mtcars %>%
  group_by(cyl) %>%
  ggvis(x = ~wt, y = ~mpg, fill = ~factor(cyl)) %>%
  layer_points() %>%
  layer_model_predictions(model = "lm")
```

Explicit grouping

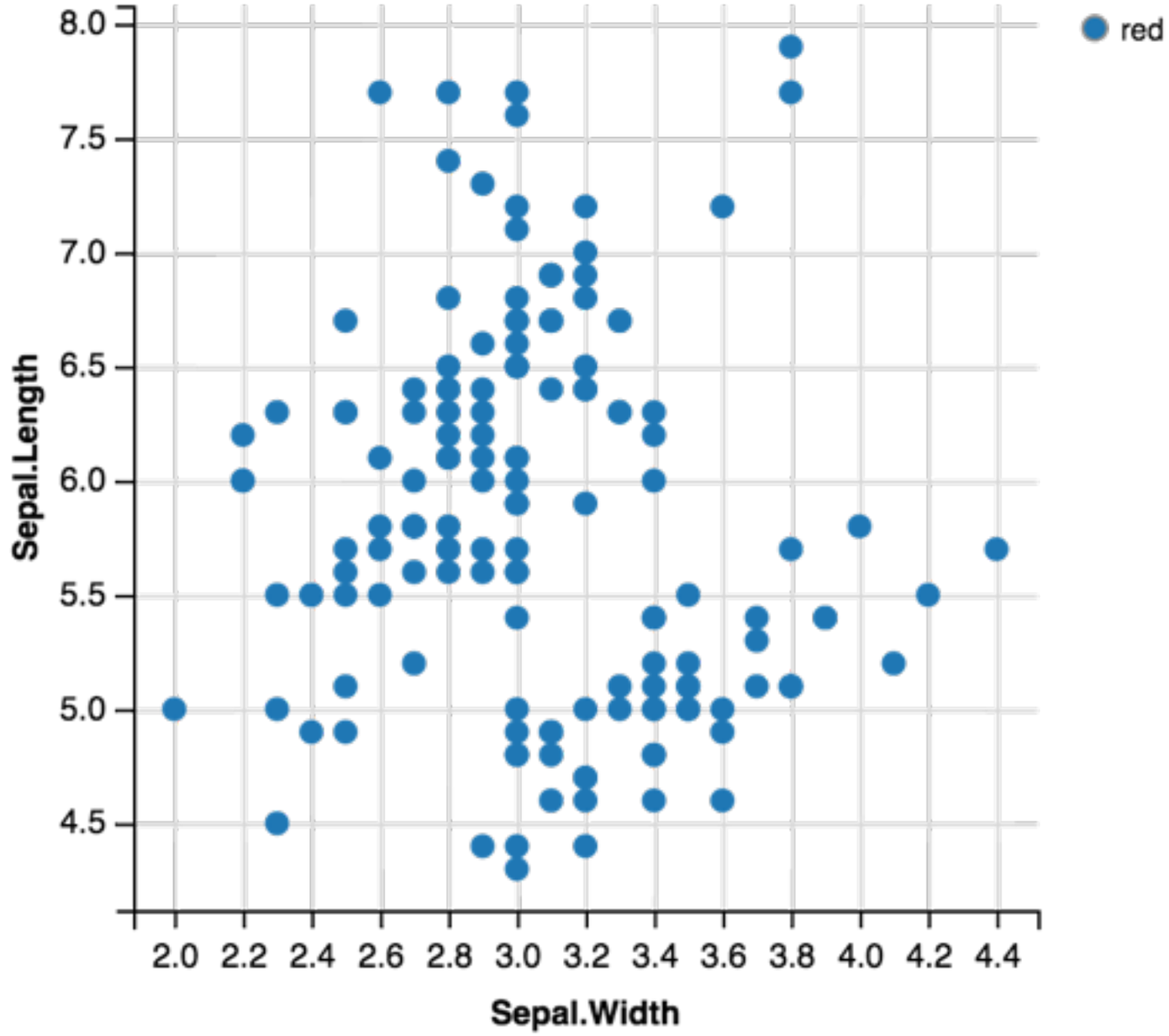


```
iris %>%
ggvis(x = ~Sepal.Width,
      y = ~Sepal.Length,
      fill = ~Species) %>%
layer_points()
```



Data Space (Species)		Visual Space (Fill)
setosa	⟷	blue
versicolor	⟷	orange
virginica	⟷	green

```
iris %>%
ggvis(x = ~Sepal.Width,
      y = ~Sepal.Length,
      fill = "red") %>%
layer_points()
```

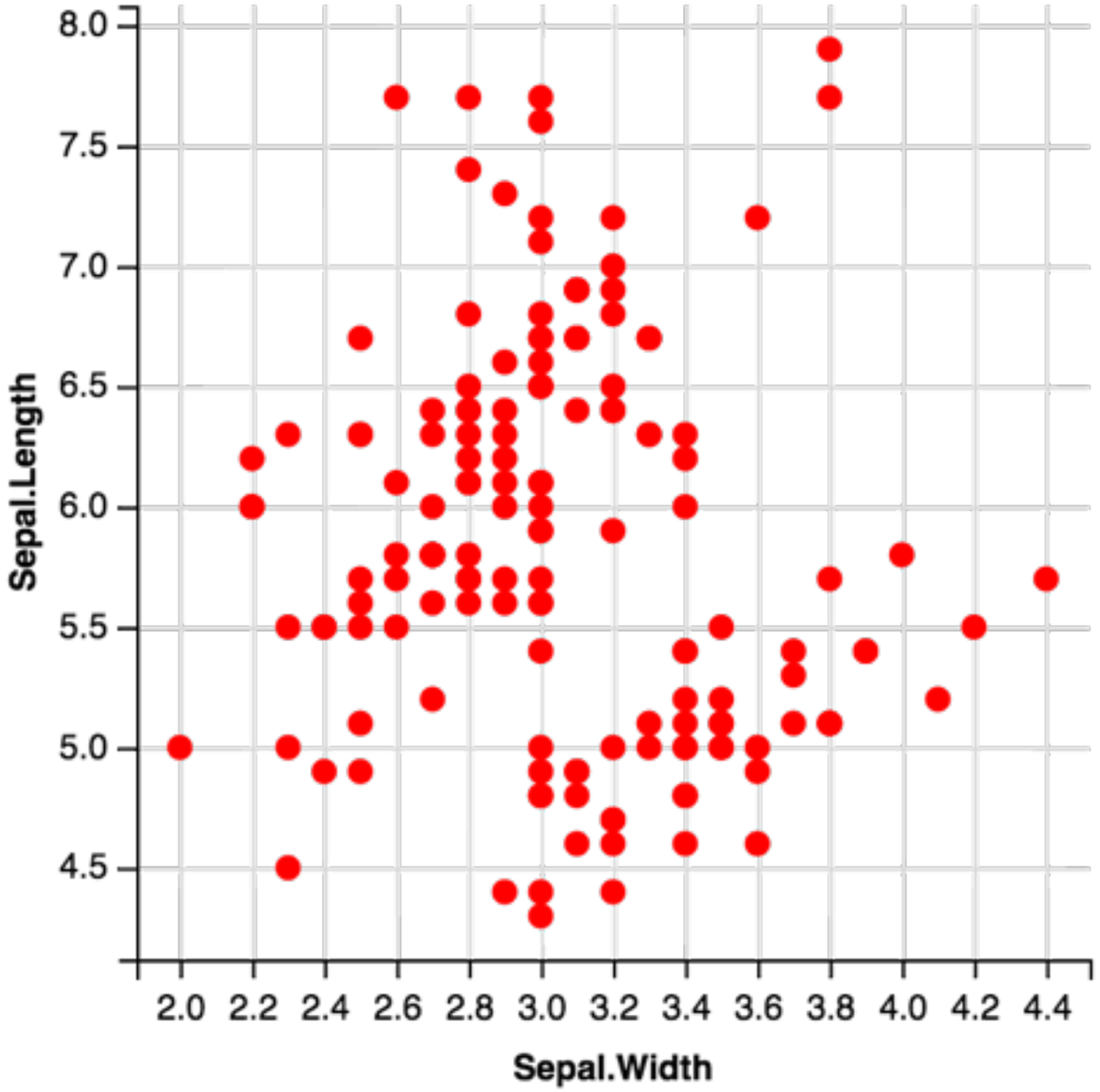


Data Space **Visual Space**

(Fill)

“red” ⟷ blue

```
iris %>%
ggvis(x = ~Sepal.Width,
      y = ~Sepal.Length,
      fill := "red") %>%
layer_points()
```



Data Space

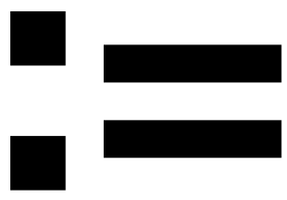
Visual Space

(Fill)

red

map

data values
to
visual values



set

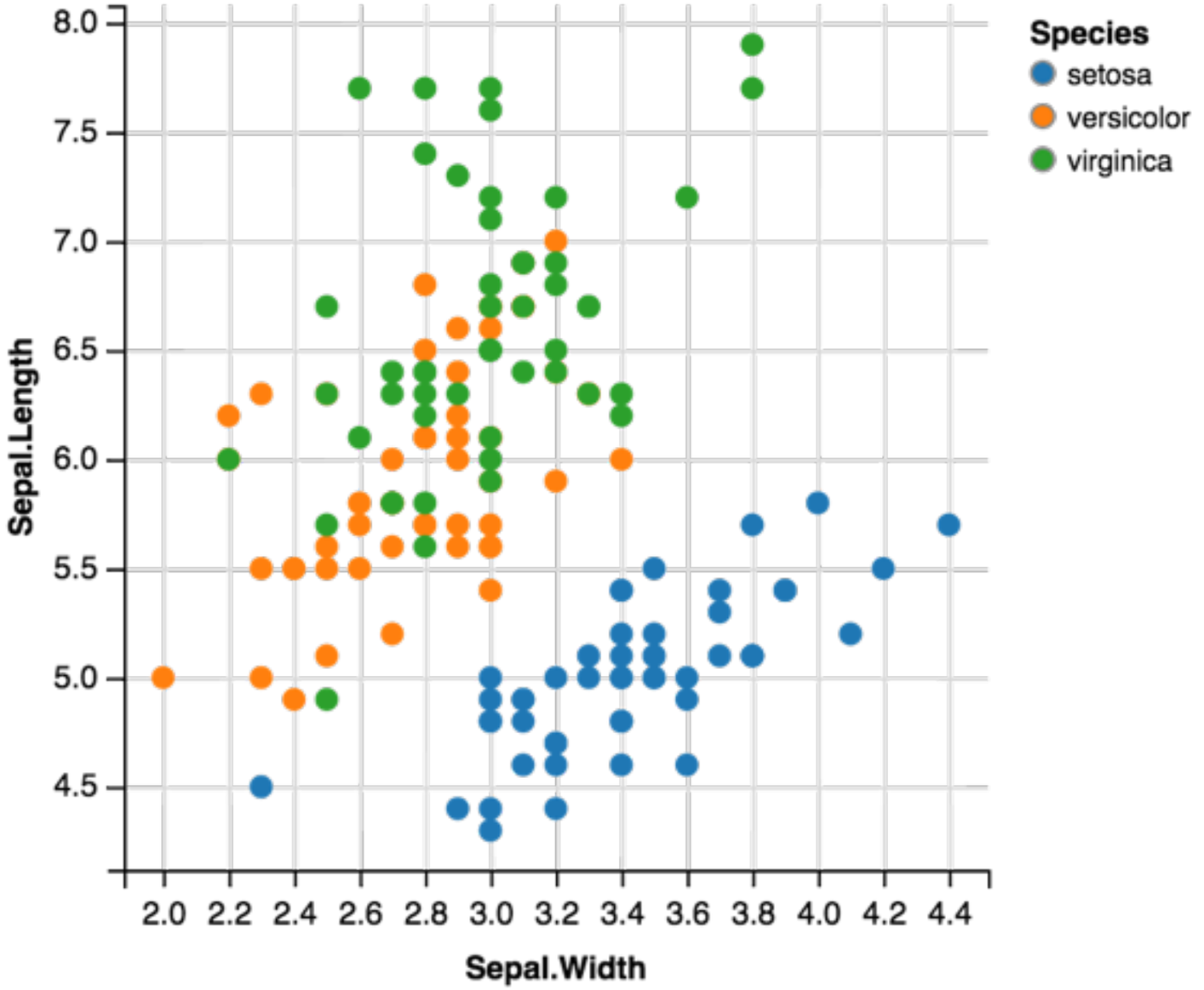
visual values

~

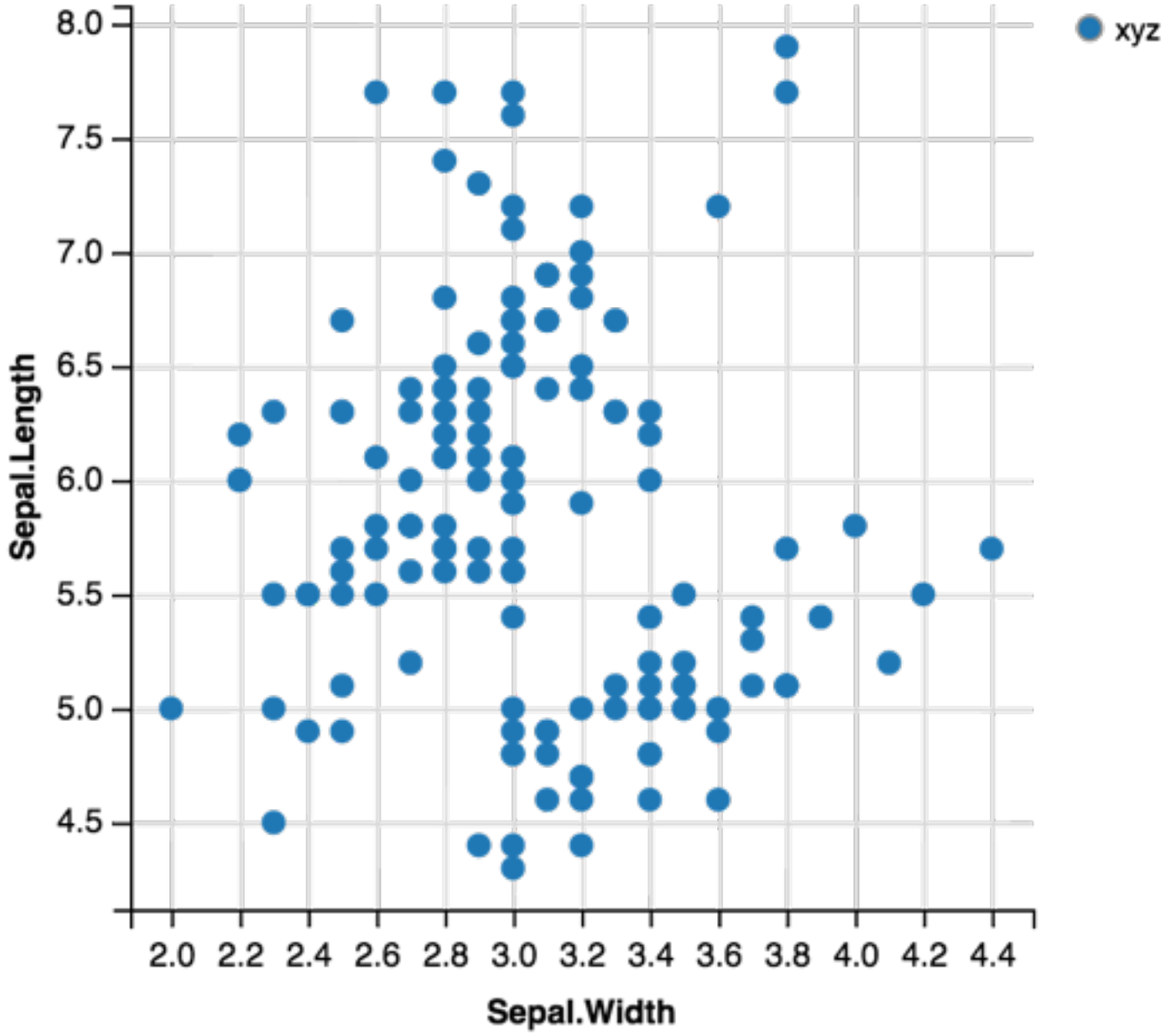
- ~ means evaluate in the context of the data

- No ~ means evaluate the expression in the current environment

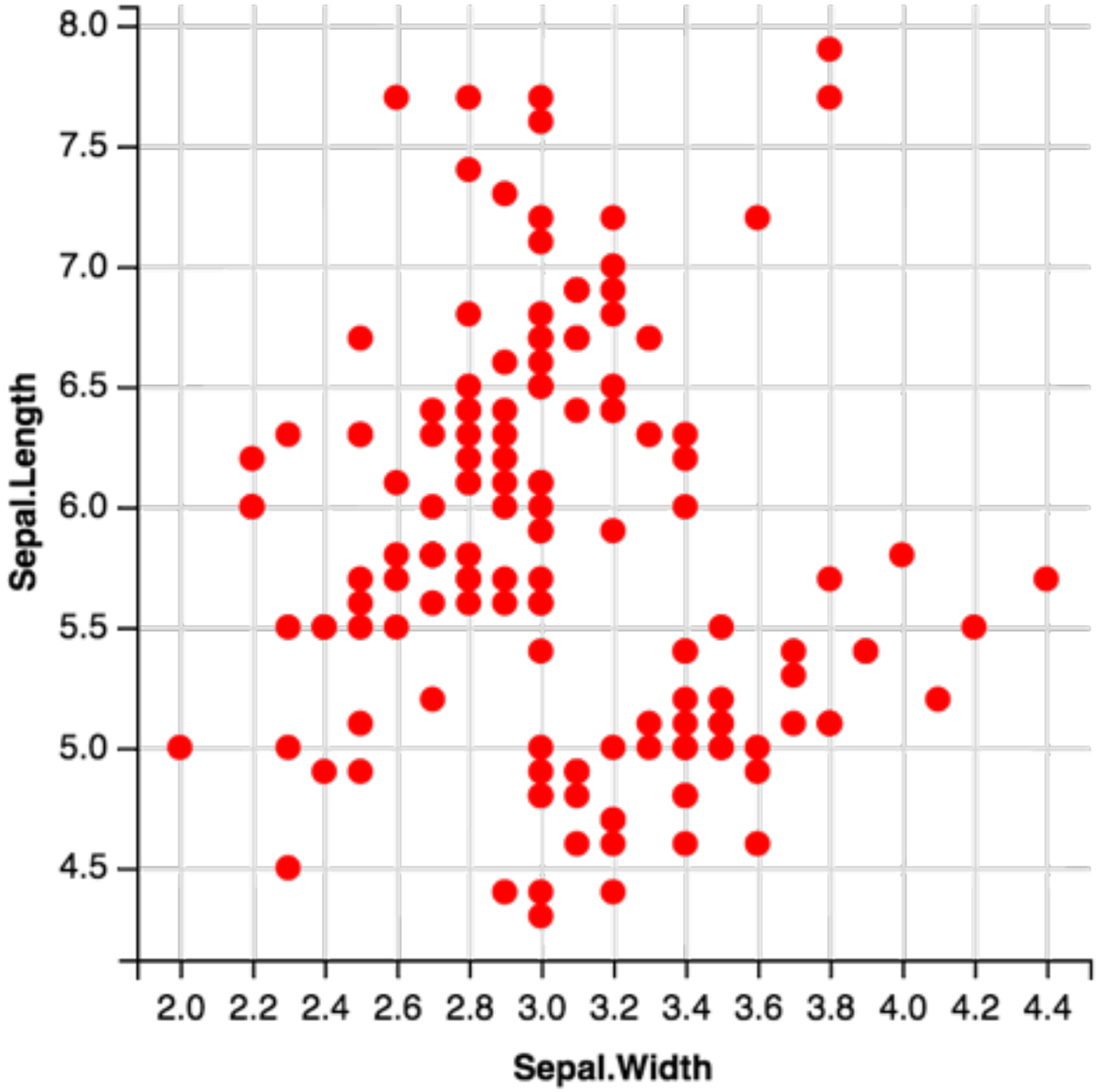
```
Species <- "xyz"
iris %>%
ggvis(x = ~Sepal.Width,
      y = ~Sepal.Length,
      fill = ~Species) %>%
layer_points()
```



```
Species <- "xyz"
iris %>%
ggvis(x = ~Sepal.Width,
      y = ~Sepal.Length,
      fill = Species) %>%
layer_points()
```



```
iris %>%
ggvis(x = ~Sepal.Width,
      y = ~Sepal.Length,
      fill := "red") %>%
layer_points()
```



Common patterns

`property = ~variable`

`property := constant`

Exercises

- Change all the points on a scatterplot to a different color.

- Add a new column to the `iris` dataset which contains color strings like `"red"`, `"green"`, `"#9900CC"`, and use that column with `fill:=` and `fill=`

- Make a scatter plot of `iris`, with `fill=~Species` but choose custom colors (see `?scale_nominal`)

Data pipeline

%>%

From **magrittr** package. Used extensively in **dplyr**.

%>% is a piping operator.

It takes the output of the left side, and uses it as the first argument of the function on the right side.

```
subset(mtcars, cyl == 6)
mtcars %>% subset(cyl == 6)

summary(subset(mtcars, cyl == 6), digits=2)
mtcars %>% subset(cyl == 6) %>% summary(digits=2)
```

Functional interface

Each ggvis function takes a visualization object as an input and returns a modified visualization as an output:

```
p <- ggvis(mtcars, x = ~wt, y = ~mpg)
p <- layer_points(p)
p <- layer_smooths(p)
p
```

Create a ggvis object with mtcars data.

Layer on points

Layer on smoothing lines

Print

```
# Three equivalent forms

layer_smooths(layer_points(ggvis(mtcars, ~wt, ~mpg)),
  span = 0.5)



p <- ggvis(mtcars, ~wt, ~mpg)
p <- layer_points(p)
p <- layer_smooths(p, span = 0.5)
p



mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points() %>%
  layer_smooths(span = 0.5)
```

Functions for controlling plot appearance

- `set_options()`
- `add_axis()`
- `add_legend()`

What's missing (for now)

- More layer types, like violins, contours

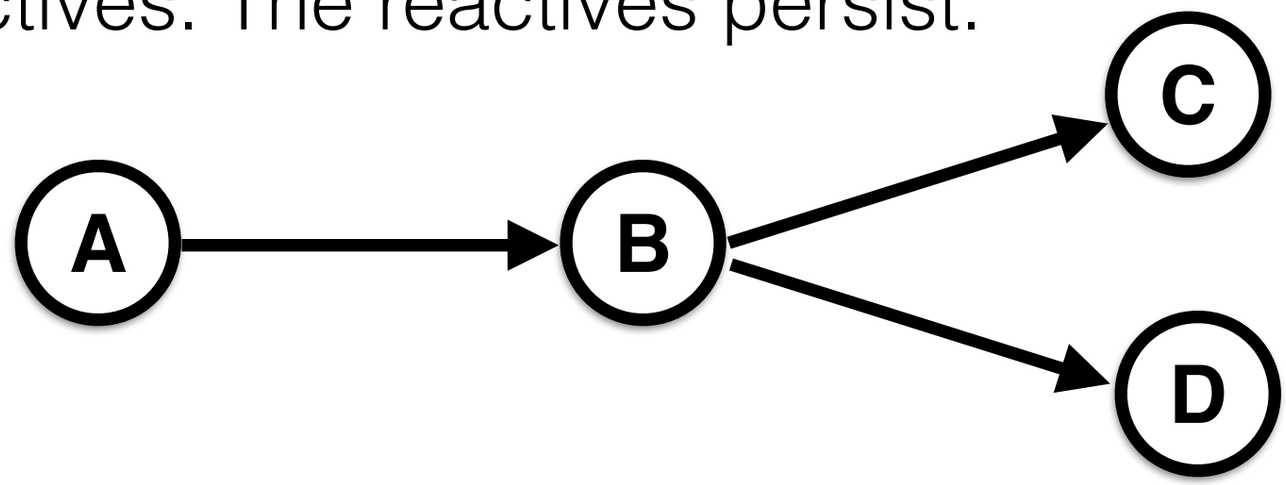
- Faceting (in progress)

Reactivity and Interactivity

Reactives from Shiny

- In “regular” programming, function calls happen once. The function takes in a value and returns a value.

- In functional reactive programming, a reactive can use a value from another reactive; this creates a dependency graph of reactives. The reactives persist.



- When the value of an ancestor node changes, it triggers recomputation of all its descendants.

Reactive computation parameters

```
faithful %>%
  ggvis(x = ~waiting) %>%
  layer_histograms(binwidth =
    input_slider(min=1, max=20, value=11))
```

Reactive properties

```
mtcars %>%
  ggvis(x = ~wt, y = ~mpg) %>%
  layer_points(
    size := input_slider(10, 400, value=50, label="size"),
    fill := input_select(c("red", “blue"), label="color")
  )
```

Reactive data sources

```
dat <- data.frame(time = 1:10, value = runif(10))

# Create a reactive that returns a data frame, adding a new
# row every 2 seconds
ddat <- reactive({
  invalidateLater(2000, NULL)
  dat$time  <<- c(dat$time[-1], dat$time[length(dat$time)] + 1)
  dat$value <<- c(dat$value[-1], runif(1))
  dat
})

ddat %>% ggvis(x = ~time, y = ~value, key := ~time) %>%
  layer_points() %>%
  layer_paths()
```

key is used to line up new and old data for visual transitions

Direct interaction

```
# This function receives information about the hovered
# point and returns an HTML string to display
all_values <- function(x) {
  if(is.null(x)) return(NULL)
  paste0(names(x), ": ", format(x), collapse = "<br />")
}

mtcars %>% ggvis(x = ~wt, y = ~mpg) %>%
  layer_points(fill.hover := "red") %>%
  add_tooltip(all_values, "hover")
```

Exercises

- Create a scatter plot with `layer_model_predictions()`, and add an `input_select()` to choose the model type (e.g., `lm` and `loess`)

- Modify the dynamic data example to not use a key. What happens, and why?

- Create a histogram that shows an informative tooltip when you hover over a bar.

Using ggvis with Shiny

Interactive (Shiny) docs

- R Markdown documents + Shiny

- More information: http://bit.ly/TkiPhR

Exercises*

*For Shiny / R markdown users

- Create an interactive document that explores a data set of your choice. Consider:
  - Using brushing to select a subset and perform an analysis on that subset.
  - Adding a reactive data set that you can filter with interactive controls.

The future

- Zooming and panning
- Subvisualizations (Faceting)
- ggplot2 feature parity
- Performance improvements
- Rendering without a web browser

More information

- http://ggvis.rstudio.com/
- http://rmarkdown.rstudio.com/
- More examples at: https://github.com/rstudio/ggvis/tree/master/demo
- Slides + code at: http://bit.ly/rday-strata14