R crash course - Visualization

Jongbin Jung

`ggplot2` Basics

Single-variable Plots (usually distributions)

Two-variable Plots (points and lines)

Labels, Scales, and More

Maps

Dependencies

- Latest version of `R` (*free* from https://www.r-project.org/)
- Latest version of `Rstudio` (also *free* from https://www.rstudio.com/)
- A bunch of *free* packages

```
# for general plotting
install.packages('tidyverse')
# spacial visualization on maps
install.packages('ggmap')
```

- Alternatively, use cloud services such as MatrixDS (http://matrixds.com) [beta]
  - Consistent access to Rstudio via browser (on all devices)
  - Easily manage/share files with others

```
install.packages('ggmap')
```

Visualization: Introduction

- There is more than one framework for thinking about data visualization, e.g.,
  1. Mapping (function) of points/vectors in 2D/3D surface/space
  2. Function of **inputs** given as variables of a data set, **geometries** and **aesthetics** that describe visual markings, and a **coordinate** system that defines the location of each marking
- The first approach is widely used in scientific visualization (e.g., `MATLAB`, classical plotting function in `R`), but doesn't scale well with data
- The second approach, implemented in `R` with the `ggplot2` package, is prefered when working with large scale data, but requires the data frame to be formatted in a specific manner (i.e., in the *long* format)

Quick Comparison: An Example

- We're given the following data as a result of some experiment

Time	Group A Score	Group B Score
1	2	3
2	6	5

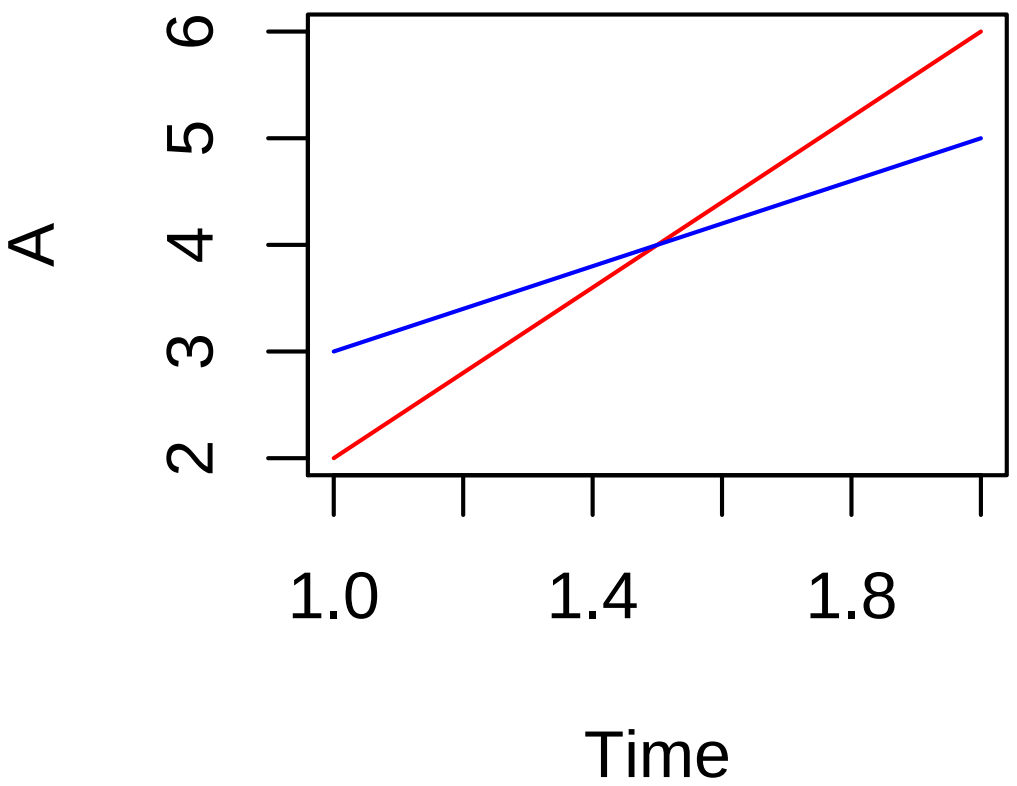
- We wish to plot the scores of each group, i.e., A and B on the vertical axis, with respect to *Time* on the horizontal axis, with different colors for each group
- First, create the data

```
Time <- c(1, 2)
A <- c(2, 6)
B <- c(3, 5)
```

Quick Comparison: The "Classic" Way

- Plot the coordinates of each vector `A` and `B` (no need to understand the code)

```
plot(Time, A, type = 'l', col = 'red')
lines(B, col = 'blue')
```



Quick Comparison: The ggplot2 Way

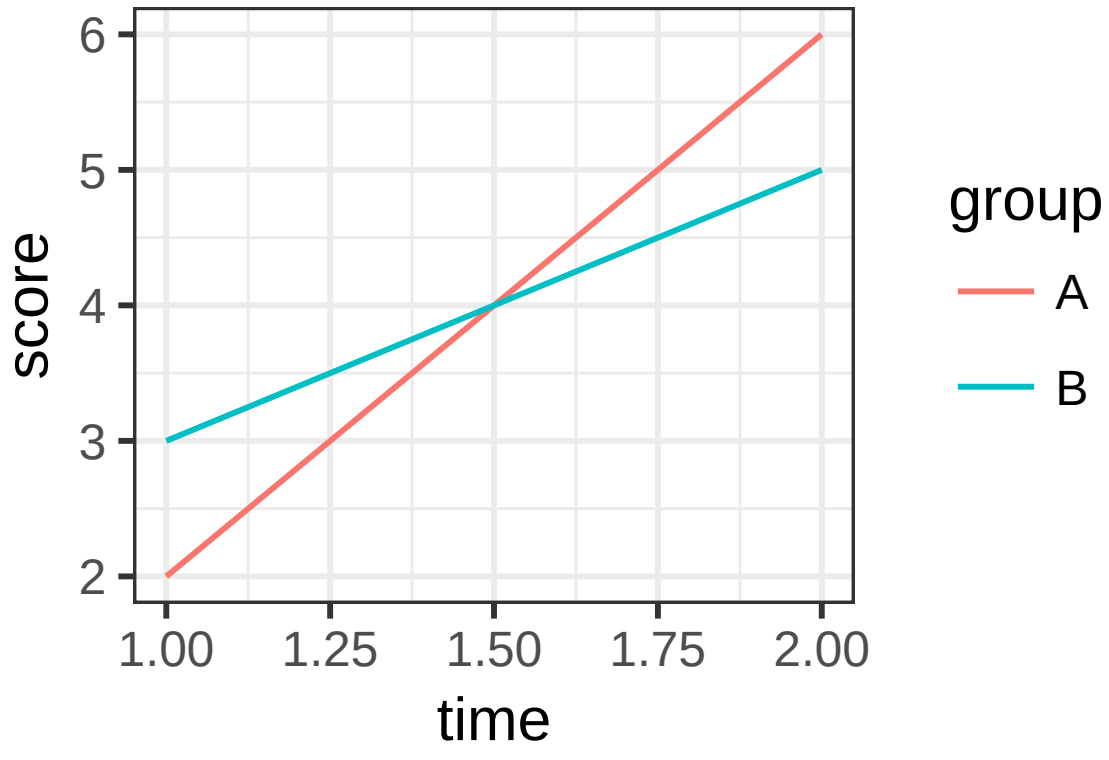
- Create data frame from the vectors, and *tidy* into *long* format (Note that the variables of interest are `time`, `score`, and `group`)

```
## # A tibble: 4 x 3
##    time group score
##   <dbl> <chr> <dbl>
## 1     1     A     2
## 2     1     B     3
## 3     2     A     6
## 4     2     B     5
```

- Then, use ggplot2 to *visualize* the data frame (this is what we'll cover in this session, so you're not supposed to understand the following code)

ggplot2 Code and Plot

```
p <- ggplot(df_tidy, aes(x = time, y = score)) +
  geom_line(aes(color = group))
p
```



Quick Comparison: Thoughts?

- What would be the benefits/downsides of either approach?
- In what situation would you rather use one over the other?
- Can you come up with a better approach?

Some Common Visualization Tasks

- Most visualization tasks of a data scientist will fall into some combination of the following
  - Explore the distribution of some data with histograms/density plots
  - Plot points on a grid, lines in a plane with meaningful shape/linetype/size/colors
  - Transform coordinates (e.g., log-transform)
  - Make axis labels, tick-marks, etc. concise and meaningful
  - Plot geographic locations on a map
- The goal of this session is to become familiar with the basic concepts and building blocks, such that
  1. you can complete most of the required tasks by yourself
  2. when you need help, you know what to Google (and how to make sense of whatever it is you find)

ggplot2 Basics

Install and Load ggplot2

- Install and load the ggplot2 package like you would any other R package
- ggplot2 is a part of the tidyverse, so if you load tidyverse, ggplot2 is also loaded
- You can also, but don't have to, install and load the ggplot2 as a standalone package like you would any other R package

```
install.packages('ggplot2')
library('ggplot2')
```

Datasets

- For this session, we'll mainly use the `quakes` dataset included with your `R` installation and the `mpg` and `economics` datasets included with `ggplot2`
- The `quakes` dataset contains the location (long/lat), depth (Km), Richter Magnitude, and ID of reporting station for 1,000 seismic events near Fiji since 1964
- The `economics` dataset contains monthly US economic time series data
- The `mpg` dataset contains a subset of the fuel economy data made public via http://fueleconomy.gov
- Take a look at each data set and accompanying documentation

```
?quakes
?economics
?mpg
```

The ggplot Object

- The basic concept of ggplot2 is that you define a ggplot object, to which you can *add* various elements (e.g., data, visual markings, labels) as layers
- First, you start by defining an empty ggplot object with the initializing function ggplot(data)

```
p <- ggplot(data = quakes)
```

- Note that
  - The ggplot object is assigned to a variable (in this case p). The object exists in the workspace, and the *plot* is only generated when you *print* the object itself (i.e., if you type p in the console).
  - An initial ggplot object is blank, equivalent to a brand new canvas.

aesthetic Mappings

- A key concept that follows the `ggplot` object is `aesthetic` (`aes`) mappings
- `aes` mappings tell the `ggplot` object where to find the inputs for certain elements of the plot (e.g., x-axis coordinates, colors)
- For example, from the `quakes` data set, if we want to have the `depth` on the x-axis and `mag` on the y-axis, we could initialize our `ggplot` object as

```
p <- ggplot(quakes, aes(x = depth, y = mag))
```

- Note that
  - `aes()` itself is a function that returns a mapping object, which is used as an argument in the `ggplot()` intialization
  - arguments within the `aes()` call can be column (variable) names
  - the `ggplot` object `p` is still blank: we haven't specified how we want x and y to be visualized

Adding geometries (and other elements)

- The building blocks of visual elements in `ggplot2` are `geometries`
- `geometries` define markings (e.g., points, lines) to be made on the *canvas*
- Elements such as `geometries` are (literally) **added** to existing `ggplot` objects
- For example

```
p <- ggplot(quakes, aes(x = depth, y = mag)) +
  geom_point()  # add 'point' geometry to p
```

- We'll explore different `geometries` and visual markings that can be **addedd** to `ggplot` objects in the following sections

Saving Plots

- You can save any plot from RStudio with Export > Save As ... or something like that
- That method of saving plots doesn't scale well, for obvious reasons, **so don't do it!**
- Use `ggsave()` to save plots to files

```
ggsave('my_plot.png',
       width = 5,
       height = 5,
       plot = p)
```

Saving Plots (cont'd)

- `ggsave()` is smart enough to determine the filetype from the extension of the filename that you specify (`png` in the above example)
- While many formats are supported, `png` and `pdf` are most commonly used
- Read the docs to harness the full power of `ggsave()`

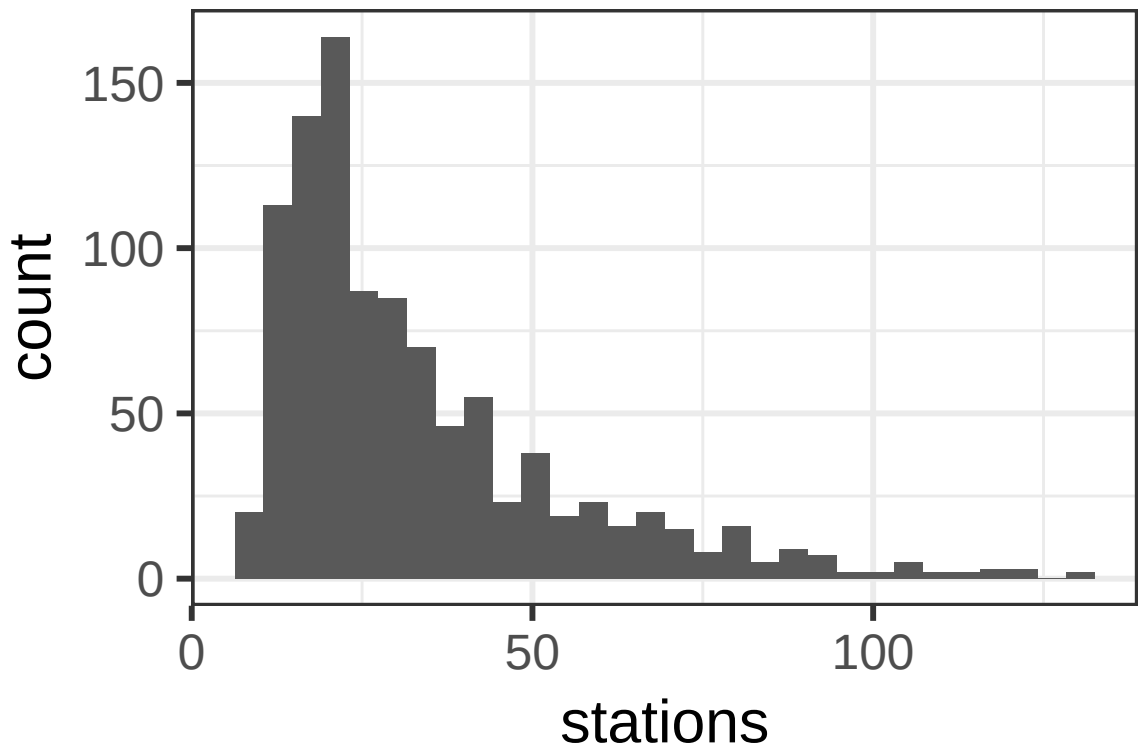
```
?ggsave
```

Single-variable Plots (usually distributions)

Histograms

- Plot a simple histogram by specifying the x-axis variable, and adding the histogram geometry with `geom_histogram()`

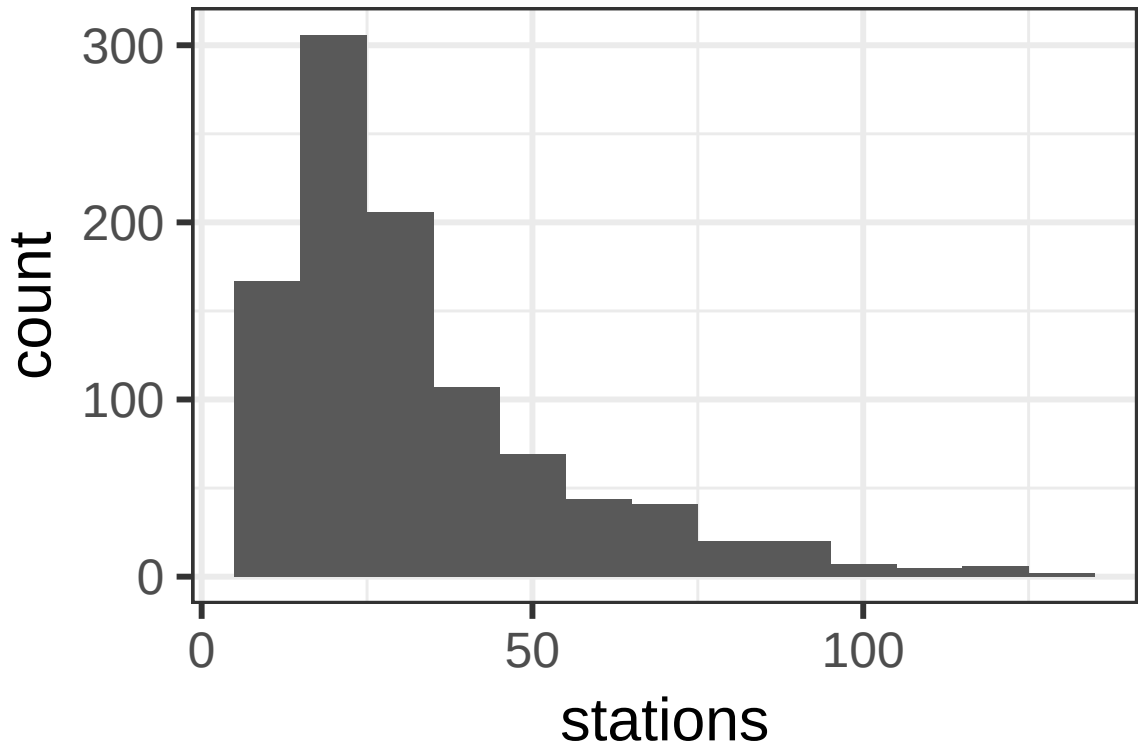
```
p <- ggplot(quakes, aes(x = stations)) +
  geom_histogram()
p
```



Histograms (cont'd)

- Specify the size of each bin in the histogram with the `binwidth` argument in `geom_histogram()`

```
p <- ggplot(quakes, aes(x = stations)) +
  geom_histogram(binwidth = 10)
p
```



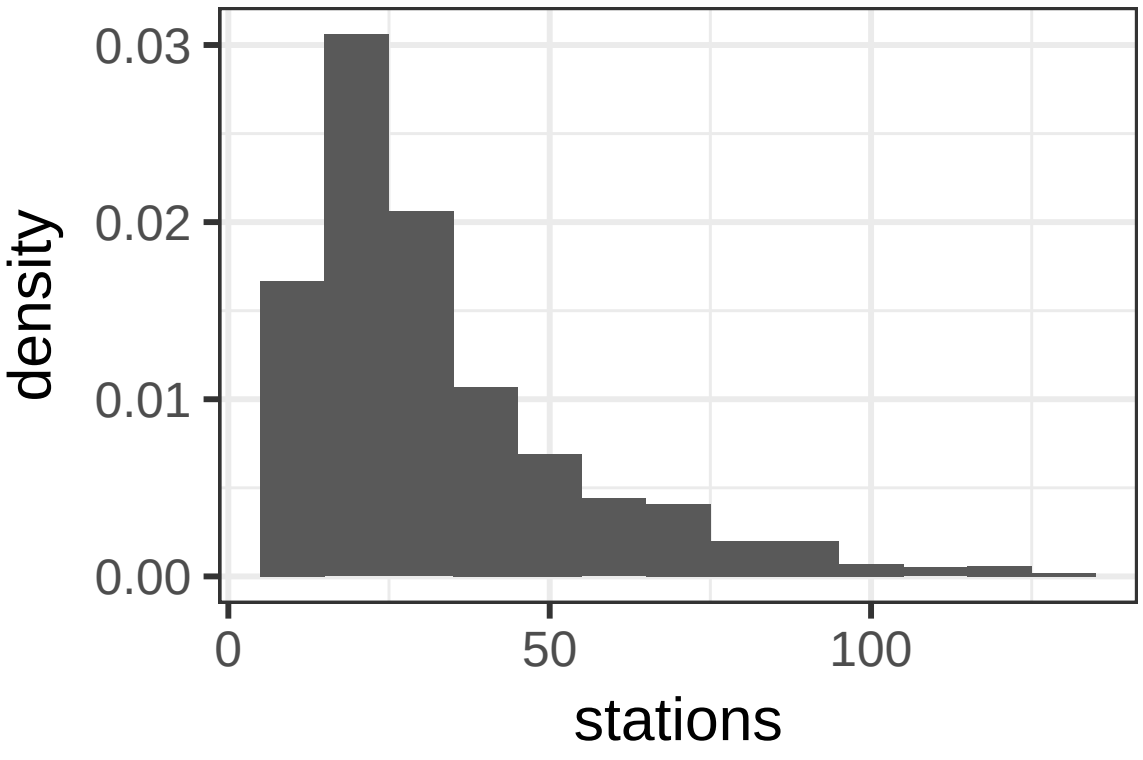
Histograms (cont'd)

- Notice that the default y-axis is `count`, i.e., the observation count of each bin
- This can be changed by specifying the `aes()` mapping of `y`
- For example, to generate a density histogram such that the points of each bin integrates to 1, set `aes(y=..density..)`
- For more options, see

```
?geom_histogram
```

Histogram with aes(y=..density..)

```
p <- ggplot(quakes, aes(x = stations)) +
  geom_histogram(binwidth = 10, aes(y = ..density..))
p
```



Exercise

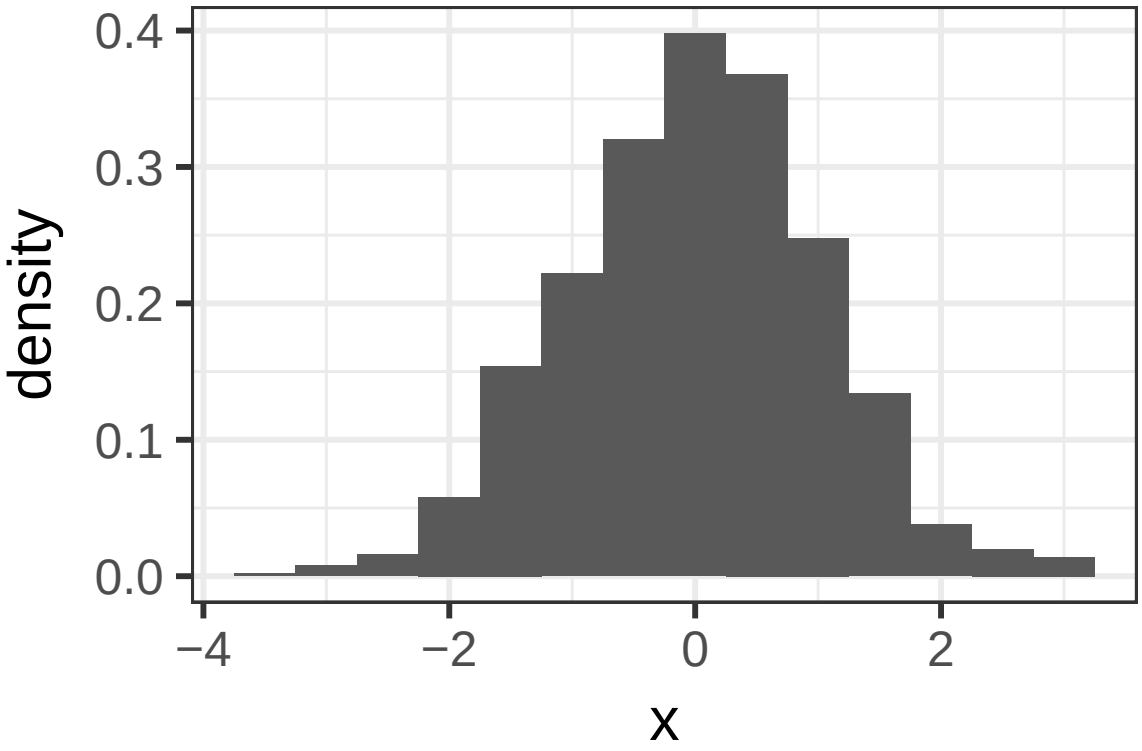
1. Plot a density histogram of 1,000 random samples from a standard normal distribution using binwidth 0.5 (hint: use `rnorm()`)
2. Use `geom_density()` to plot the (smooth) density of the population (`pop`) variable from the `economics` data

WARNING

- Solutions to the exercise are presented in the next slide
- Try the exercise before proceeding!

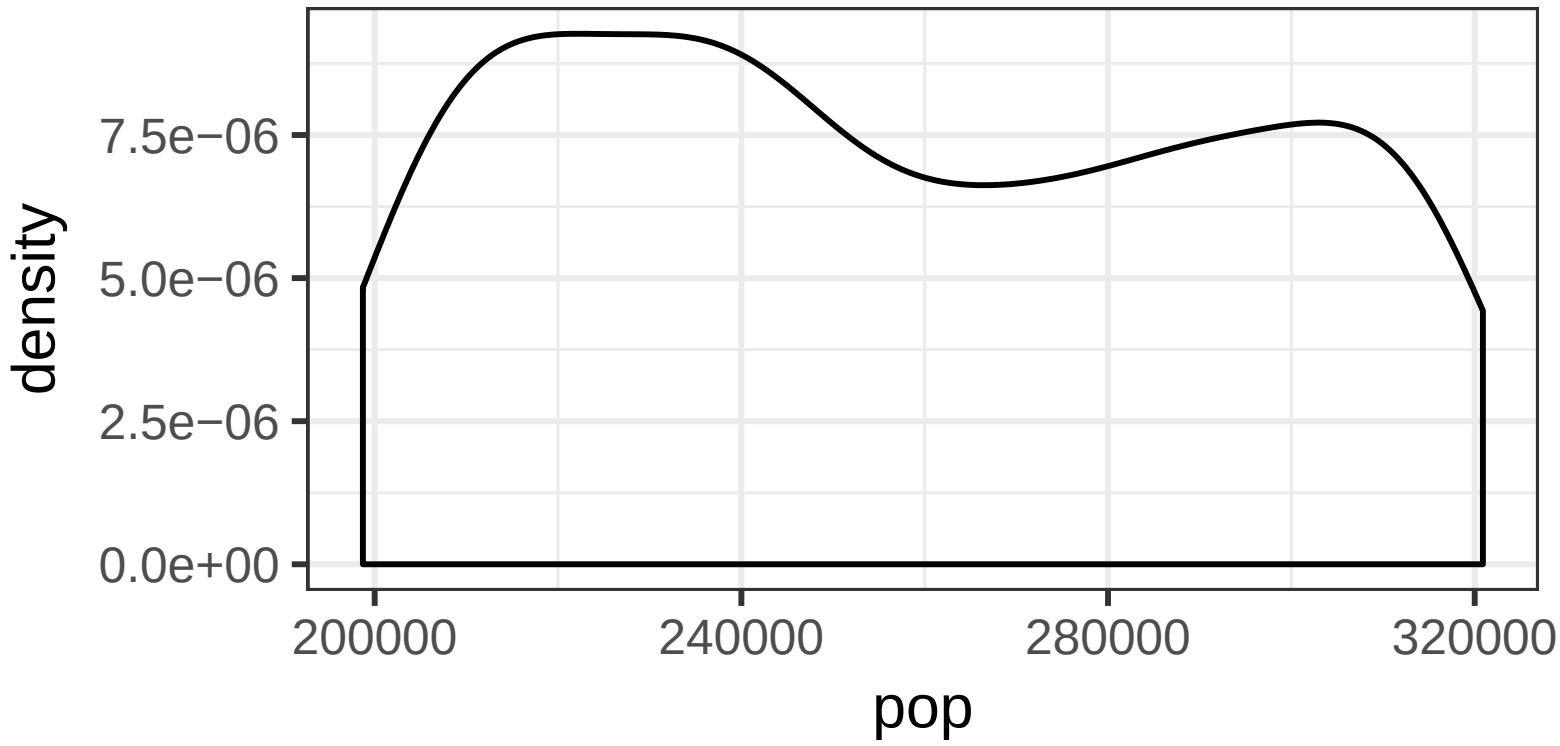
Solution 1

```
X <- tibble(x = rnorm(1000))
p <- ggplot(data = X, aes(x = x)) +
  geom_histogram(binwidth = 0.5, aes(y = ..density..))
p
```



Solution 2

```
p <- ggplot(data = economics, aes(x = pop)) +
  geom_density()
p
```

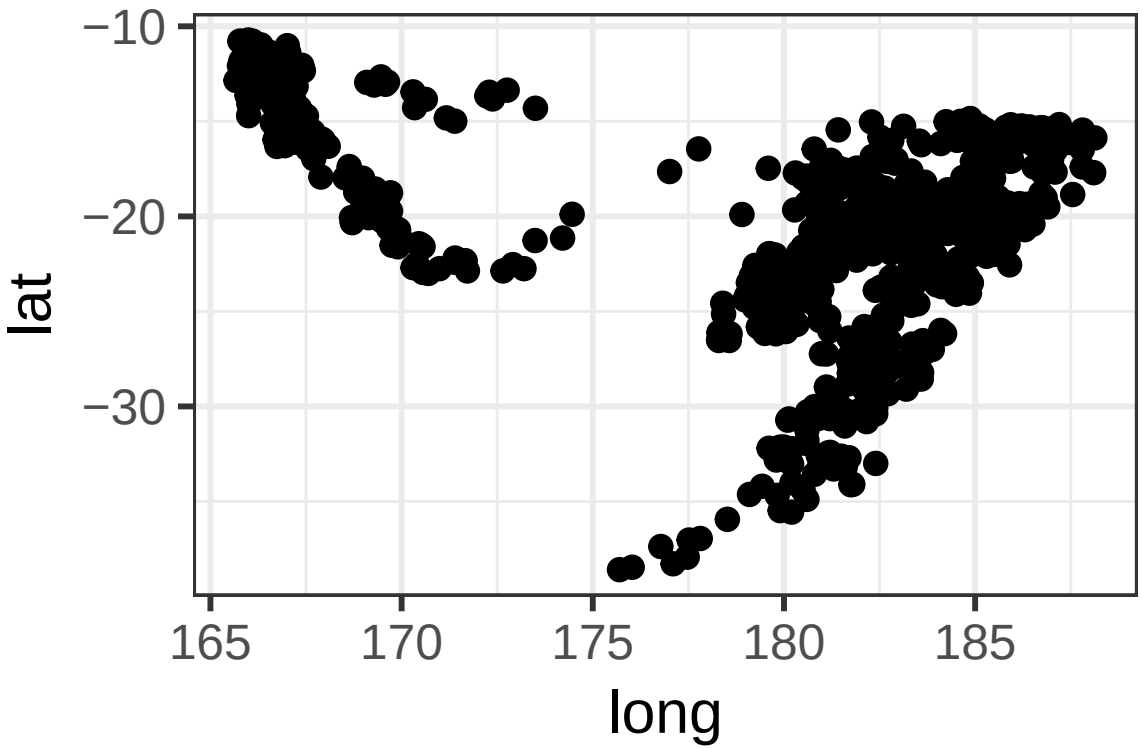


Two-variable Plots (points and lines)

Points with geom_point()

- Plot points on a 2D plane by specifying variables corresponding to the x and y-axis, and adding the point geometry with `geom_point()`

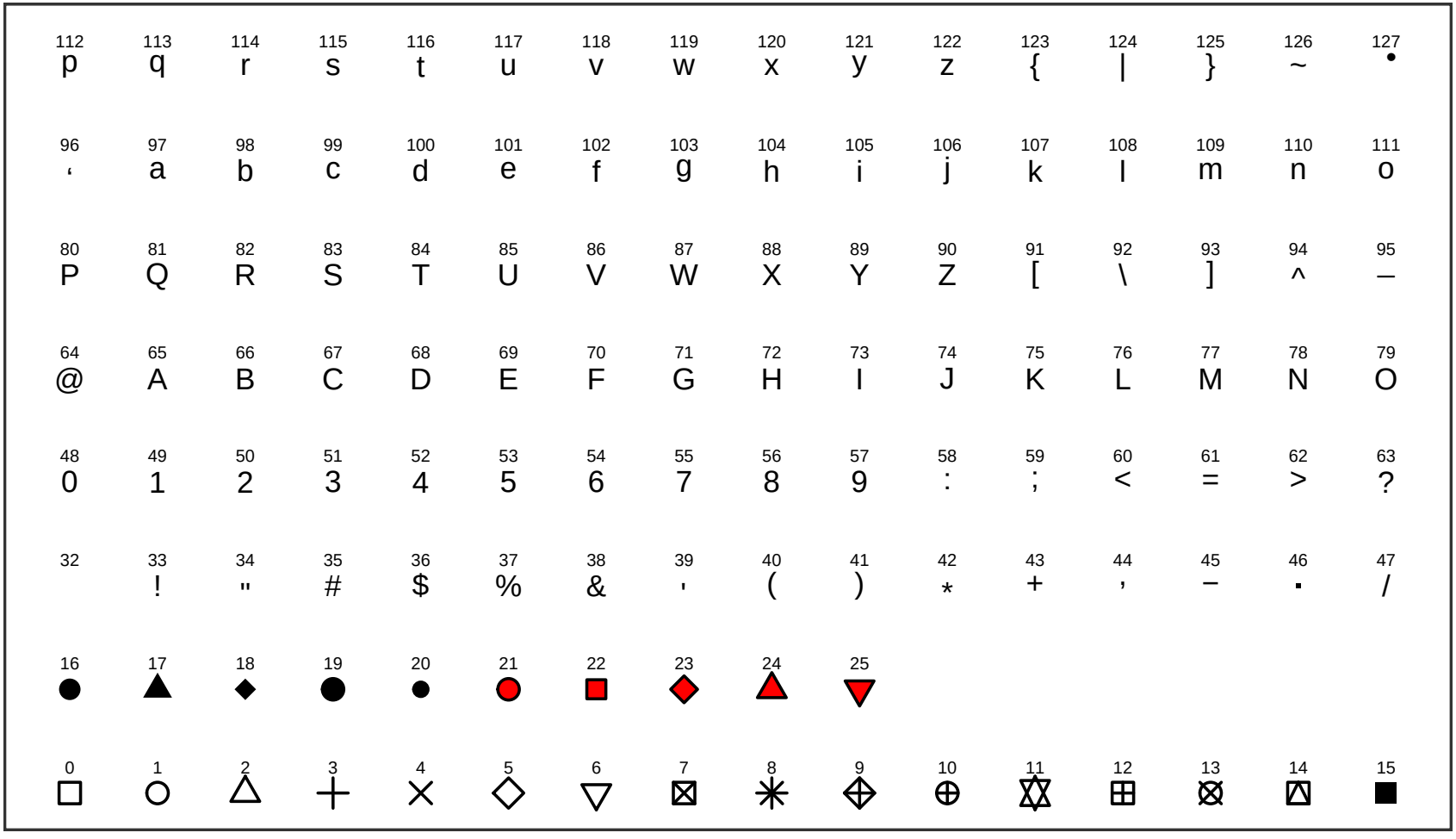
```
p <- ggplot(quakes, aes(x = long, y = lat)) +
  geom_point()
p
```



aesthetics for `geom_point()`

- Popular `aesthetics` for `geom_point()` are
  - `alpha`: point visibility; 0 = invisible, 1 = opaque
  - `color`: color of the points (try `colors()` to see a list of some pre-defined colors)
  - `shape`: shape of the points (predefined, see next slide for reference)
  - `size`: size of the points
  - `fill`: color used to fill-in the points (only applies to certain shapes, i.e., shape numbers 21 to 25)

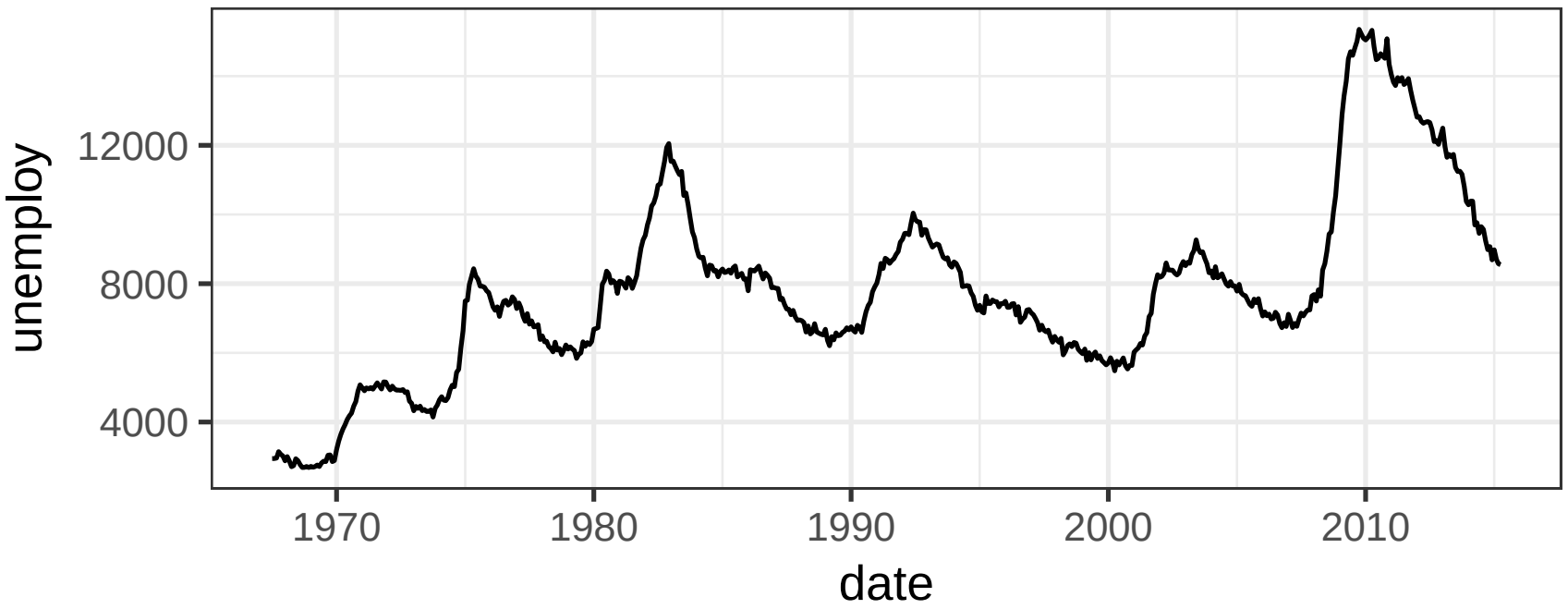
Reference: Shapes



Lines with `geom_line()`

- Similarly, plot lines on a 2D plane by specifying variables corresponding to the x and y-axis, and adding the line geometry with `geom_line()`

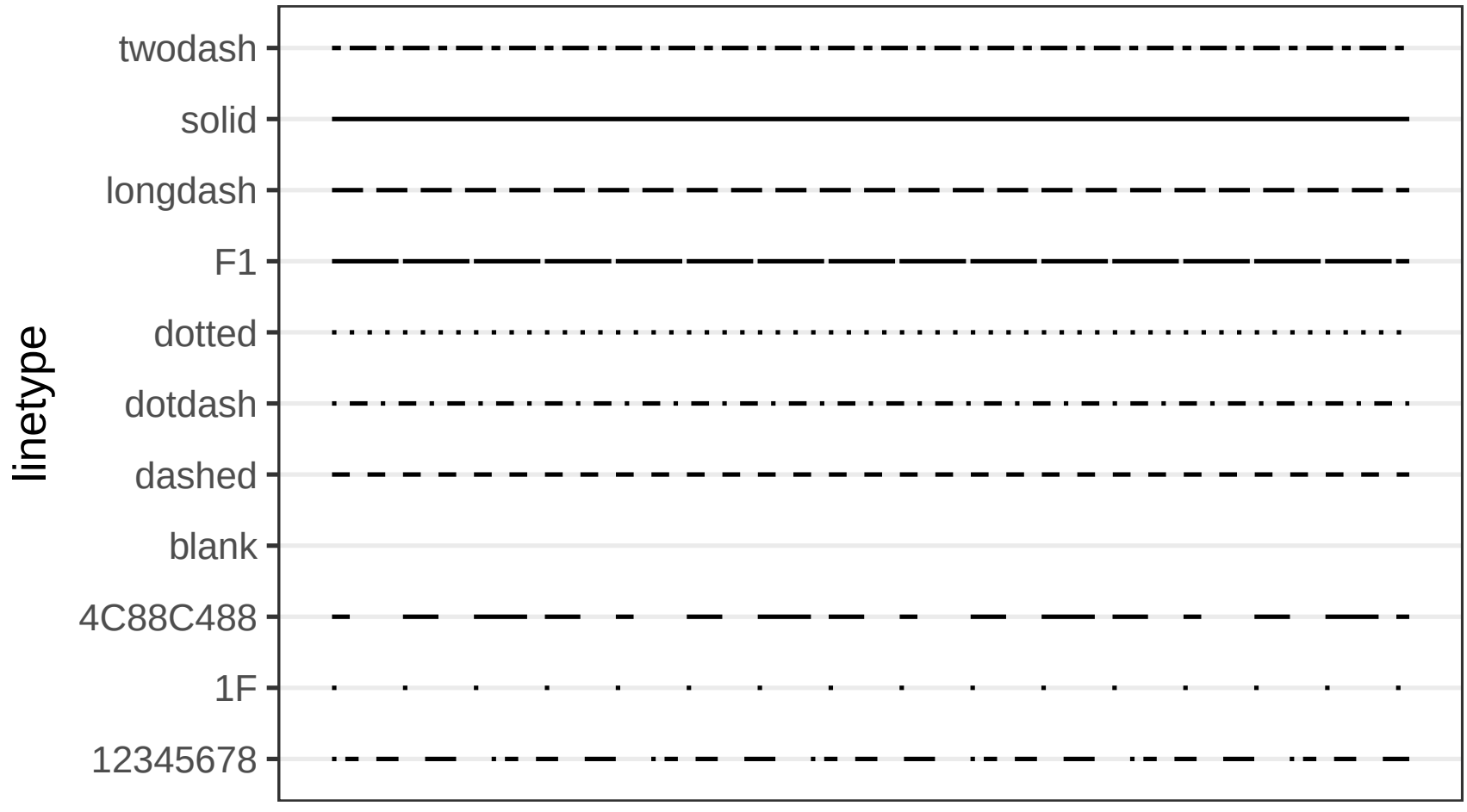
```
p <- ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line()
p
```



aesthetics for `geom_line()`

- Popular `aesthetics` for `geom_point()` are
  - `alpha`: line visibility; 0 = invisible, 1 = opaque
  - `color`: color of the lines
  - `linetype`: shape of lines (predefined, see next slide for reference)
  - `size`: size (thickness) of the lines

Reference: Linetypes



A Note on `data` and `aes()` Arguments

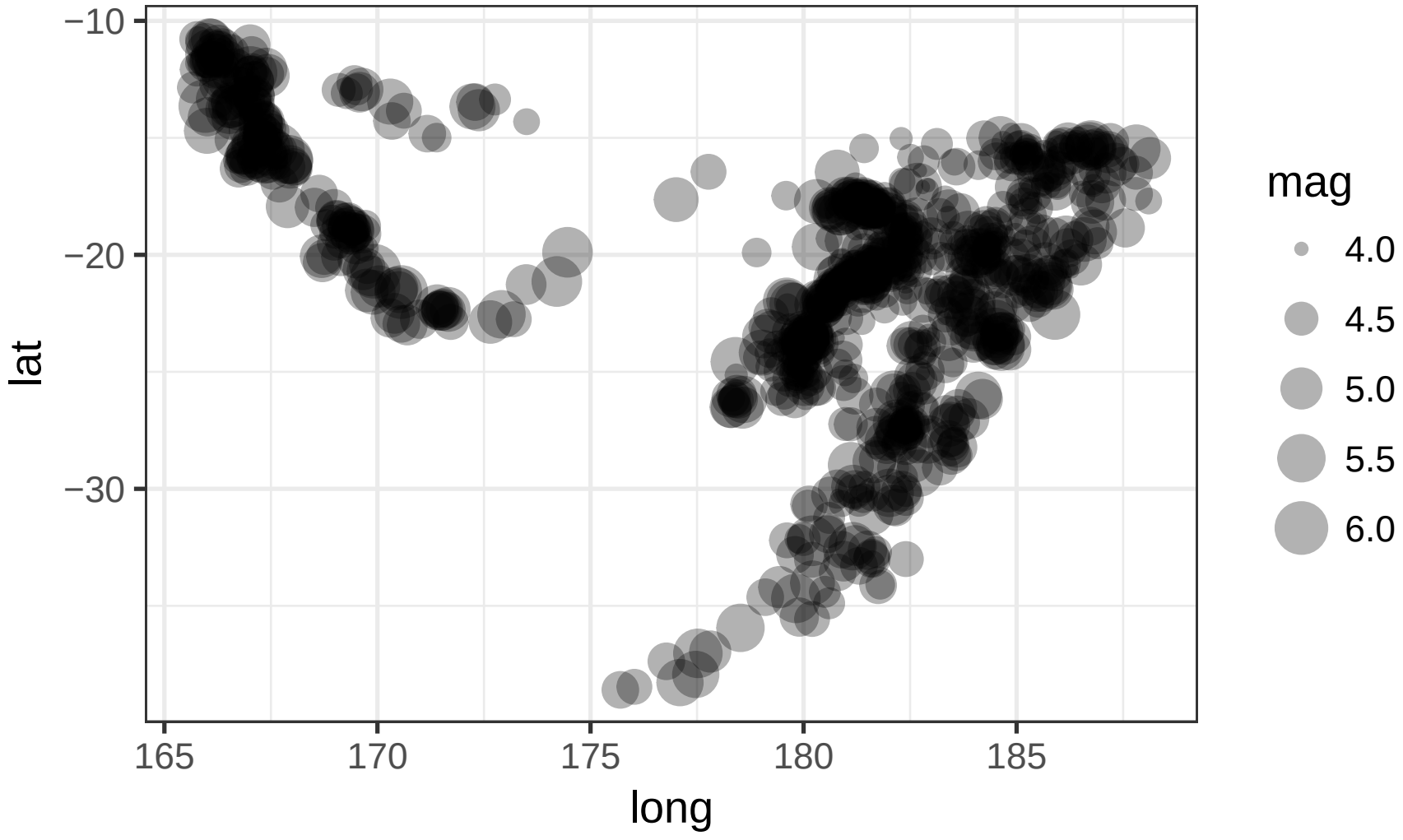
- The `data` and `aes()` arguments, can be declared globally in the `ggplot()` function, or locally in each `geometry` function
- Also, `aesthetics` can be either
  - mapped to a variable *globally*, i.e., in `ggplot(aes())`
  - mapped to a variable *locally*, i.e., in `geom_*(aes())`, or
  - defined explicitly for a local `geom_*()`, outside of `aes()`
- It's critical to understand the difference between 'mapping' an aesthetic and 'explicitly defining' one as an argument

Example: Global aes() mapping

```
p <- ggplot(quakes, aes(x = long, y = lat, size = mag)) +
  geom_point(alpha = .3)
p
```

- the data and aesthetic mappings for x, y, and size are defined globally in ggplot()
- this means any geom_* added to this ggplot will have the specified x, y, and size aesthetic *mappings*, unless assigned otherwise within their own geom_*() function
- the alpha aesthetic for geom_point, on the other hand, is defined **explicitly** (i.e., it is **set** to 0.3, and not mapped to a variable)

Example: Global aes() mapping (figure)

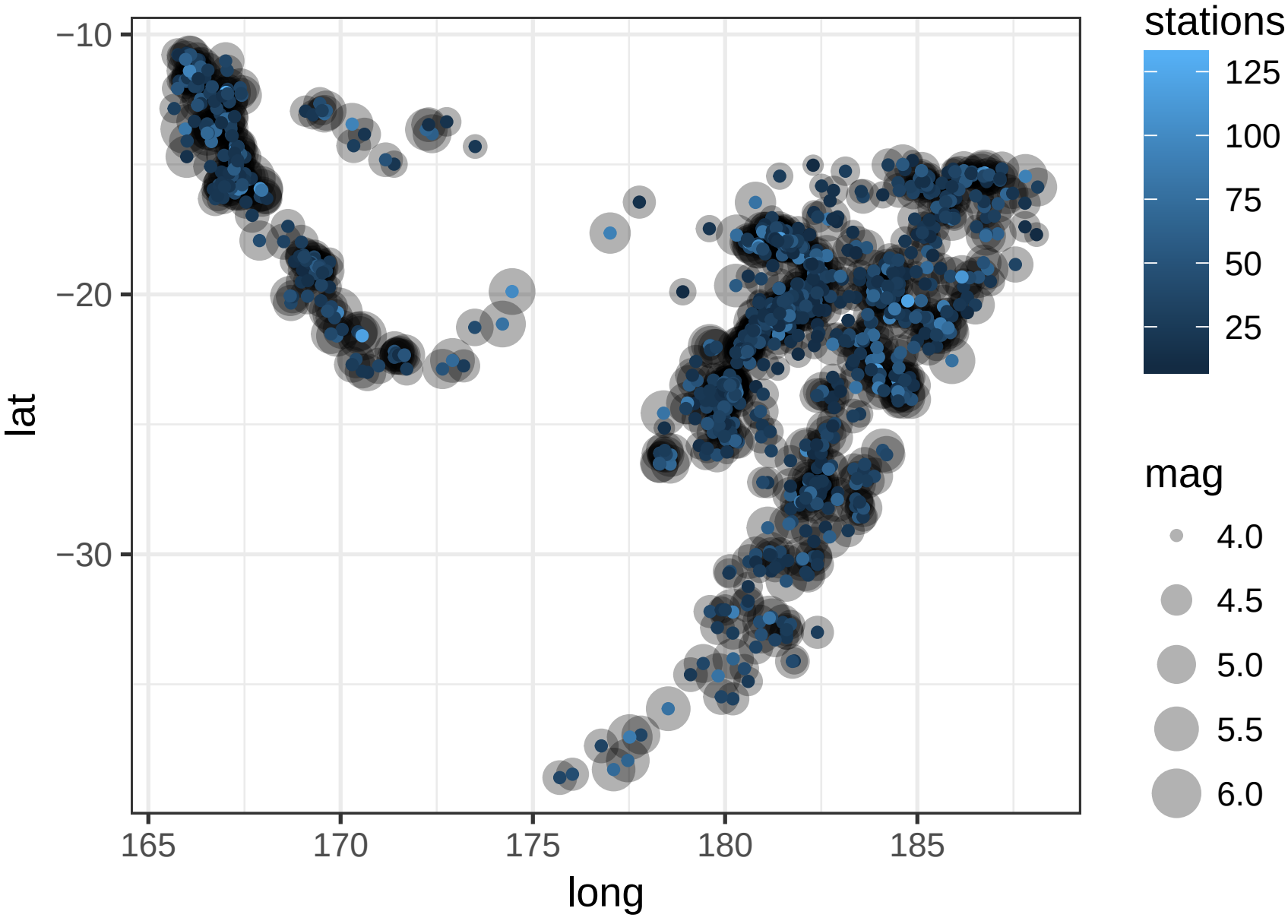


Example: Local aes() mapping

```
p <- ggplot(quakes, aes(x = long, y = lat)) +
  geom_point(alpha = .3, aes(size = mag)) +
  geom_point(size = 1, aes(color = stations))
p
```

- Here, the data and aesthetic mappings for x and y are defined globally in ggplot()
- But the aesthetic mapping/value for size is defined locally for each specific geom_point()
- The first geom_point() maps size to the mag variable, which means the size of the points will depend on the corresponding value of mag
- The second geom_point() explicitly assigns size to the fixed value 1, but maps the color aesthetic to the stations variable
- What do you think the plot will look like?

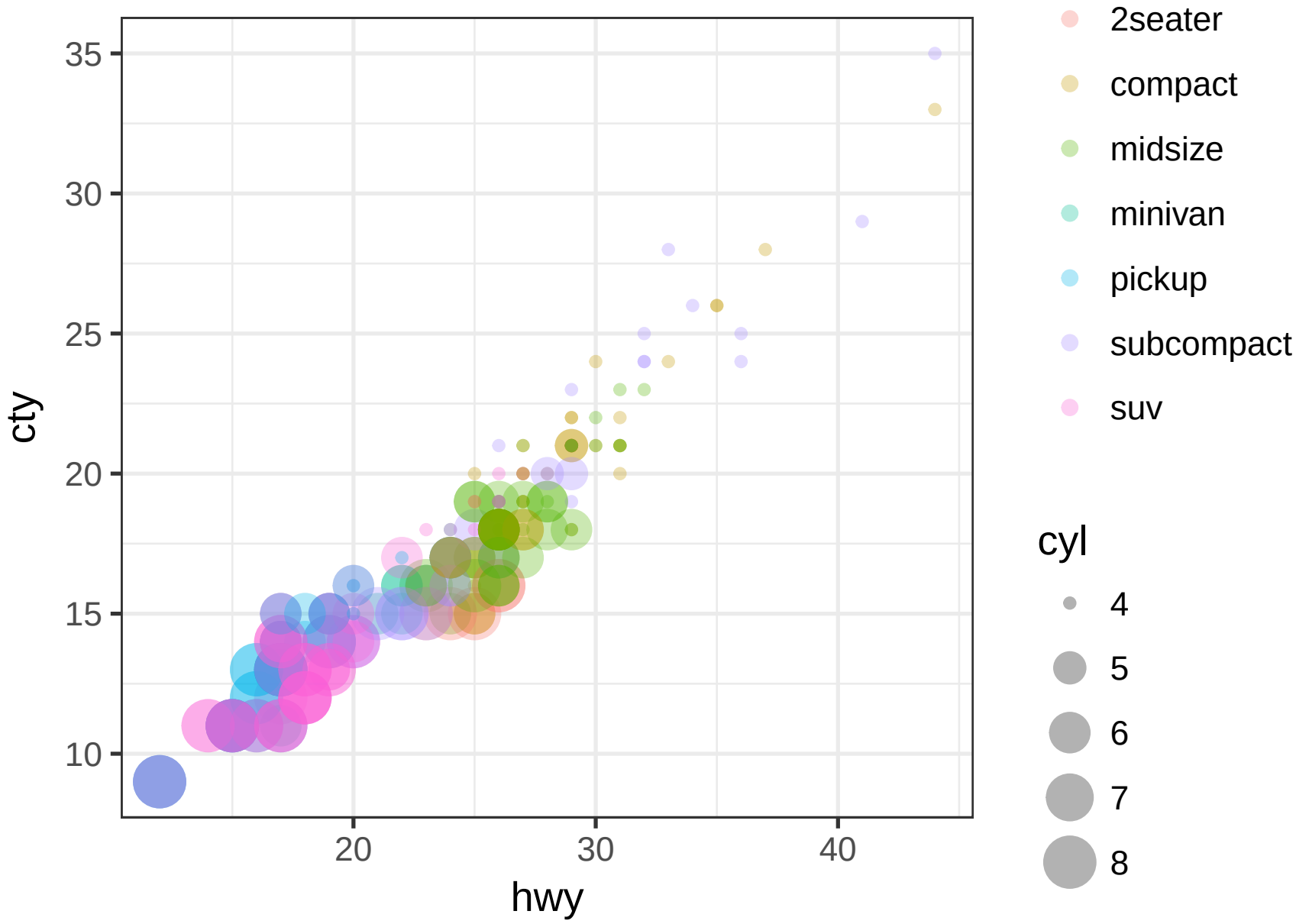
Example: Local aes() mapping (figure)



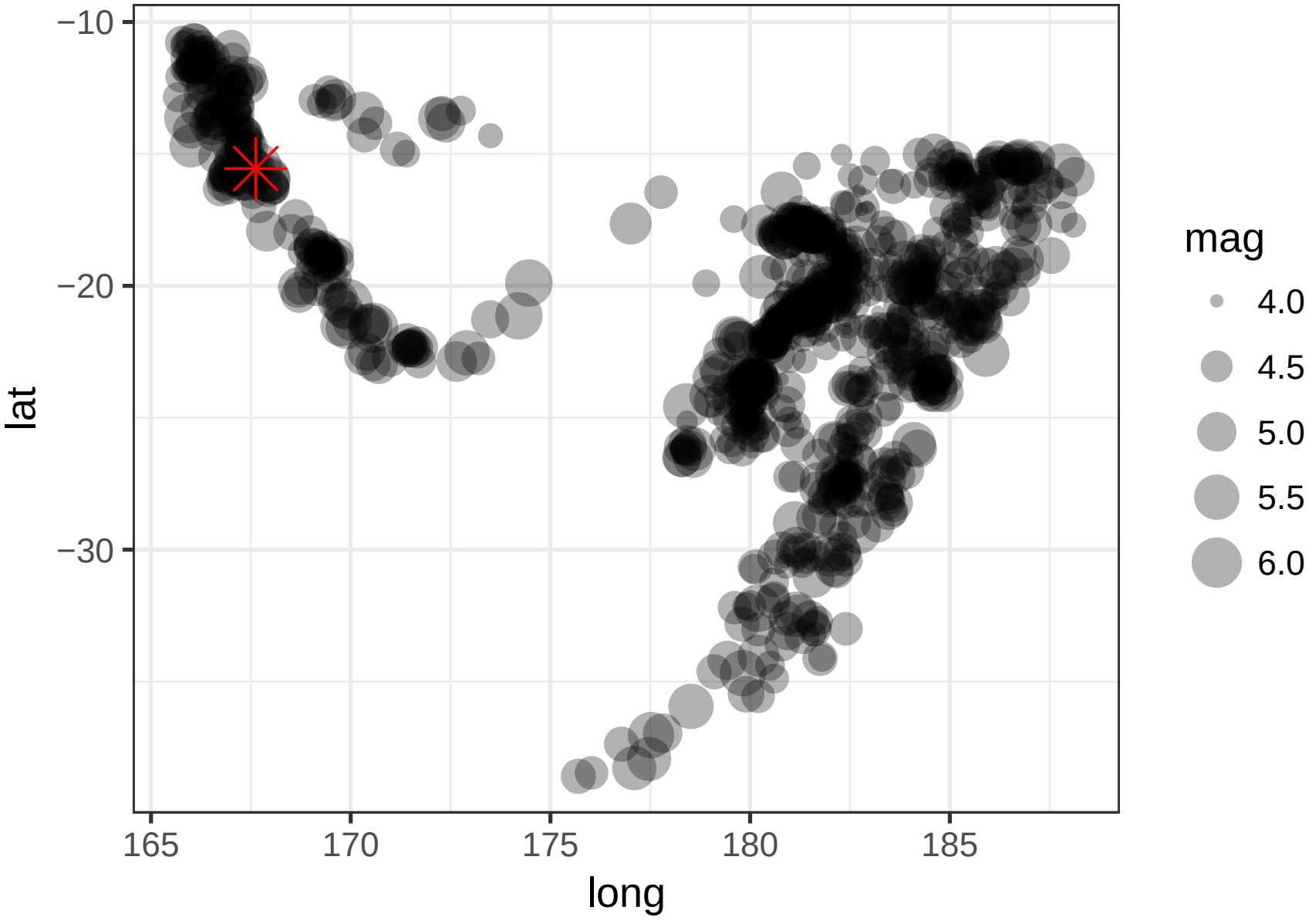
Exercise

1. With the `mpg` dataset, generate a scatter plot with `hwy` on the x-axis and `cty` on the y-axis. Let the `color` represent to `class` and `size` represent `cyl`. Also set `alpha=.3`.
2. From the `quakes` plot on the previous slide, replace the blue colored points with *one big red star* (size=5, shape=8) showing which location had the largest magnitude (hint: you'll need to crate another data frame consisting of the single observation you wish to plot)
3. [OPTIONAL, requires `tidyr`] Using the `economics` dataset, plot lines for the values of `unemploy` and `pop` with different linetypes, against `date` as the horizontal axis. (hint: you'll need to select the variables you need, and tidy the data into long format)

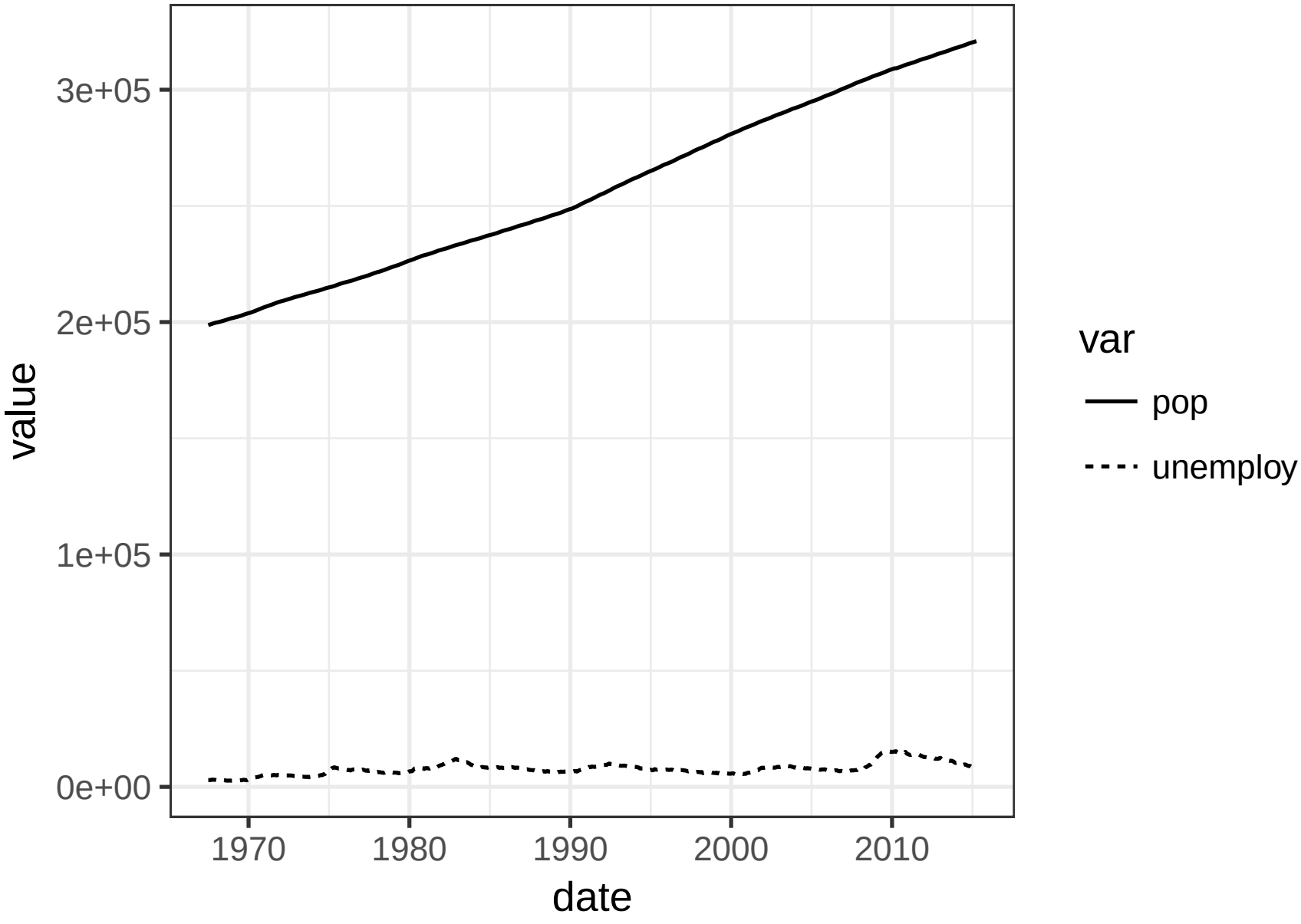
Goal: Exercise 1



Goal: Exercise 2



Goal: Exercise 3



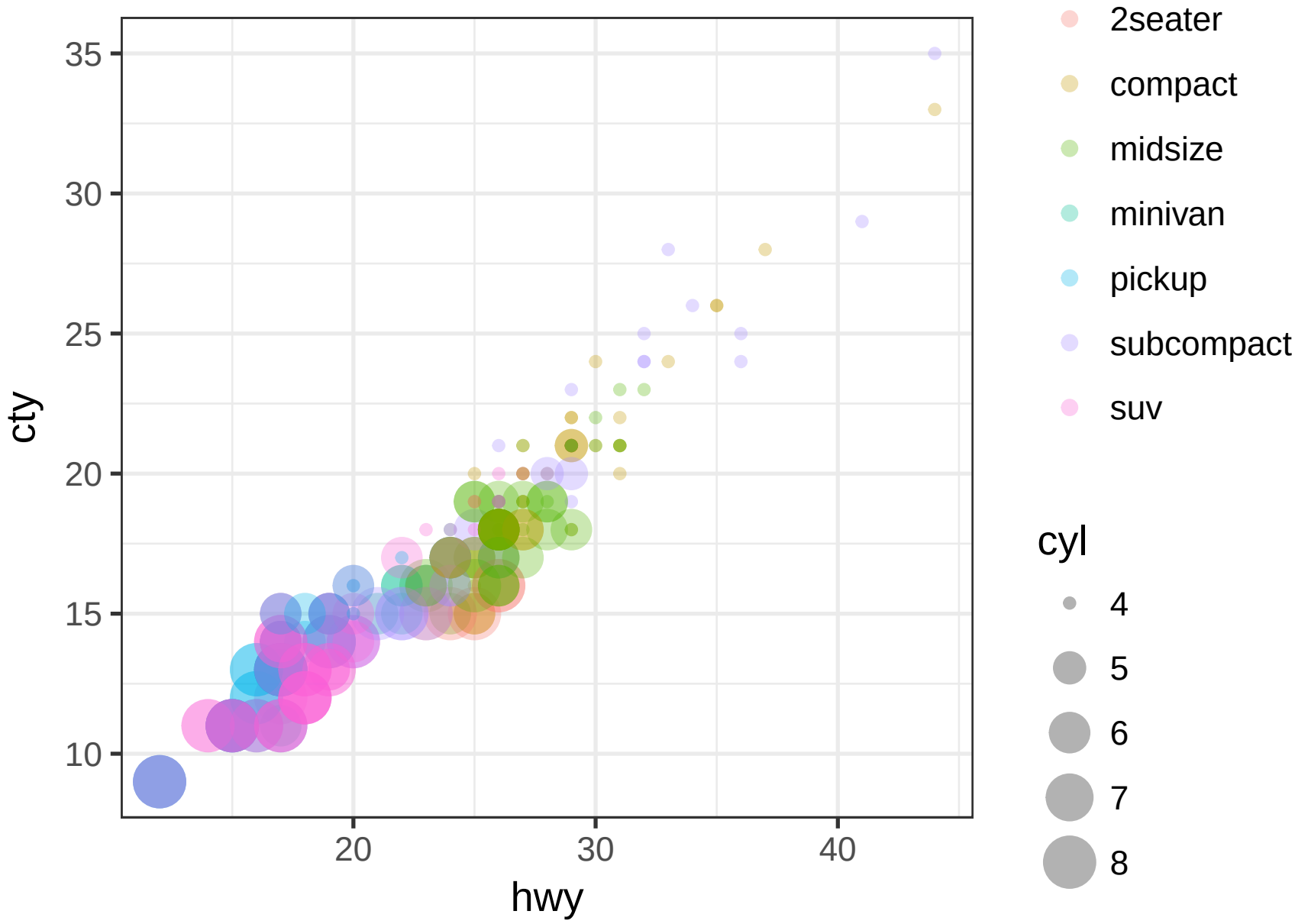
WARNING

- Solutions to the exercise are presented in the next slide
- Try the exercise before proceeding!

Solution 1

```
p <- ggplot(mpg, aes(x = hwy, y = cty)) +
  geom_point(alpha = .3,
             aes(color = class, size = cyl))
p
```

Solution 1 (figure)

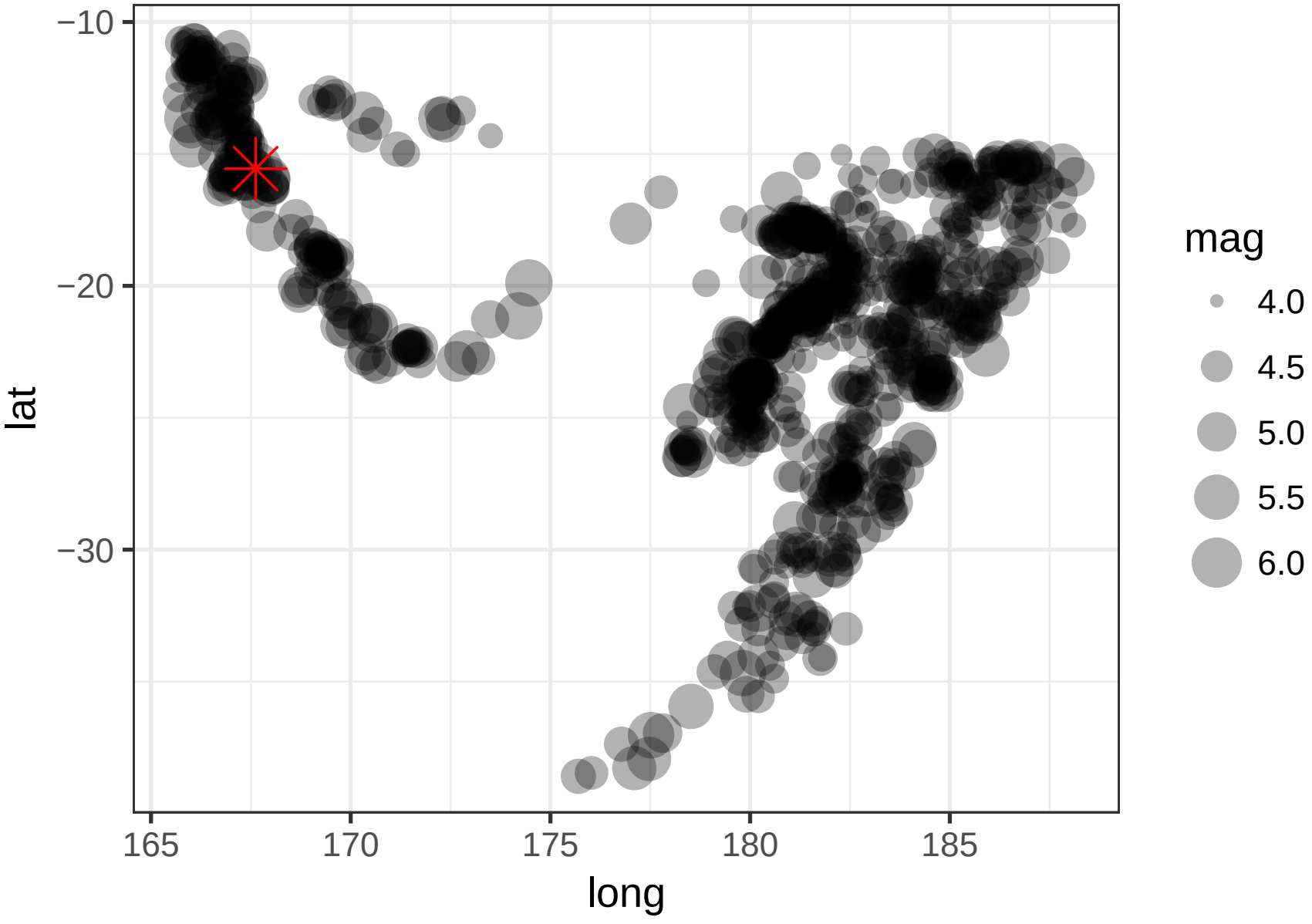


Solution 2

```
# Find the largest magnitude observation
quakes_max_mag <- quakes %>%
  arrange(desc(mag)) %>%
  slice(1)

# Generate plot
p <- ggplot(quakes, aes(x = long, y = lat)) +
  geom_point(alpha = .3, aes(size = mag)) +
  geom_point(data = quakes_max_mag,
             size = 5,
             color = 'red',
             shape = 8)
p
```

Solution 2 (figure)

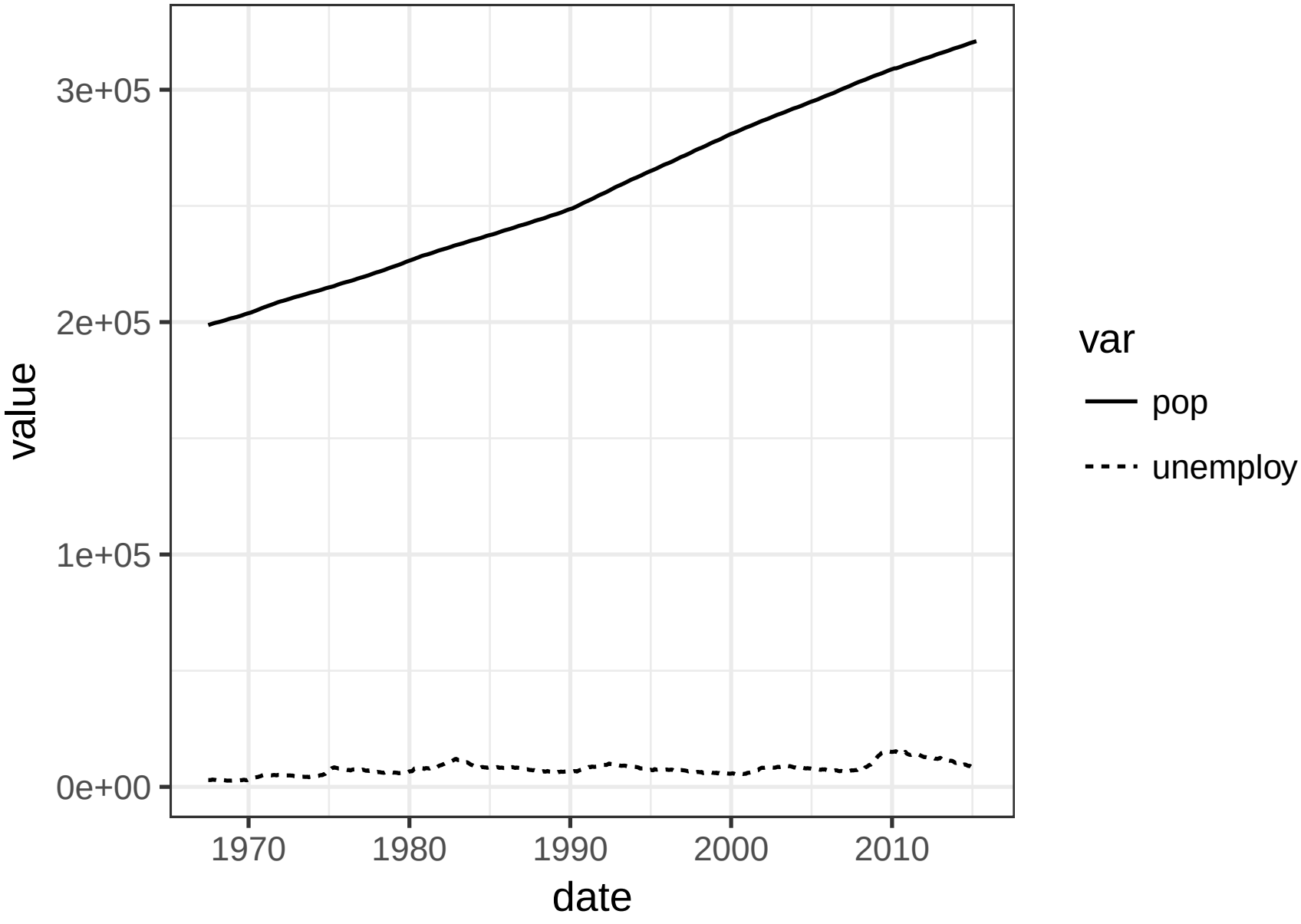


Solution 3

```
# First, get the data into the right format
econ_tidy <- economics %>%
  select(date, unemploy, pop) %>%
  gather(var, value, unemploy:pop)

# Generate the plot
p <- ggplot(econ_tidy, aes(x = date, y = value)) +
  geom_line(aes(linetype = var))
p
```

Solution 3 (figure)



Labels, Scales, and More

Labels

- Three major labels on a plot would be
  1. Plot title: use `ggtitle('title')`
  2. x-axis label: use `xlab('x label')`
  3. y-axis label; use `ylab('y label')`
- Or, to do all of the above in a single line, use

```
labs(title='title', x='x label', y='y label')
```

- `paste()` or `sprintf()` functions could be useful if you want to construct labels from data

Labels (Example)

```
p <- ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line() +
  labs(title = 'Unemployment by time',
       x = 'Date (by month)',
       y = 'Unemployment')
p
```

Labels: Pro tip!

```
p <- ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line() +
  labs(title = 'Unemployment by time',
       x = '\nDate (by month)',
       y = 'Unemployment\n')
p
```

Scales

- The mapping between data and aesthetics is controled via **scales**
- The `scales` package provides many helpful functions when working with scales

```
# Install, if you haven't already.
# Only need to do this once on a single machine.
install.packages('scales')
# load package into workspace
library('scales')
```

Scales: General Syntax

- The general syntax for setting scales is

```
scales_{aes name}_{some option}(
    # name of legend/axis
    name = ...
    # range of data values to map
    limits = ...
    # what aesthetic values to use for the limits
    values = ...
    # where to mark legend/axis
    breaks = ...
    # what to label the marks (breaks)
    labels = ...
)
```

- Some scales have specific arguments/convenience functions
- Be sure to checkout the documentation!

Axis Specific Scales

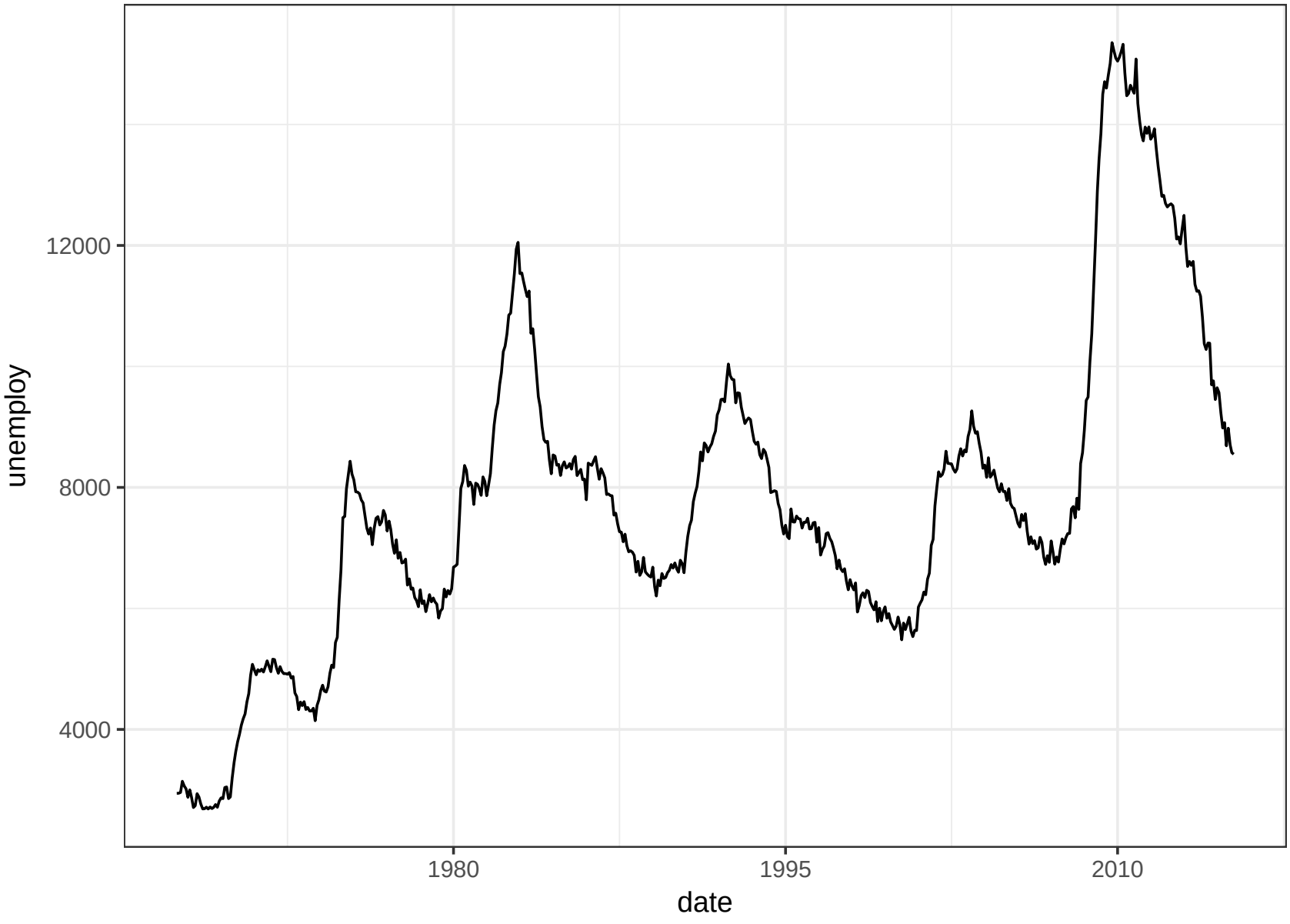
- There are scales specifically for controling the x/y-axes (examples are given for x, but can be used for y by substituting x for y)
  - `scale_x_log10()`: Plot x on `log10` scale
  - `scale_x_reverse()`: Reverse the direction of the x axis
  - `scale_x_sqrt()`: Plot x on square root scale
  - `scale_x_date(labels=date_format("%m/%d"), breaks=date_breaks("2 weeks"))`: For axes in date format. See `?strptime` for label formats, and `scale_x_date` for more details

Axis Scales Example

- To plot `unemploy` against `date`, but mark the x axis for every 15 years, we would write

```
p <- ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line() +
  scale_x_date(labels = date_format('%Y'),
               breaks = date_breaks("15 years"))
p
```

Axis Scales Example (figure)



General Purpose Scales

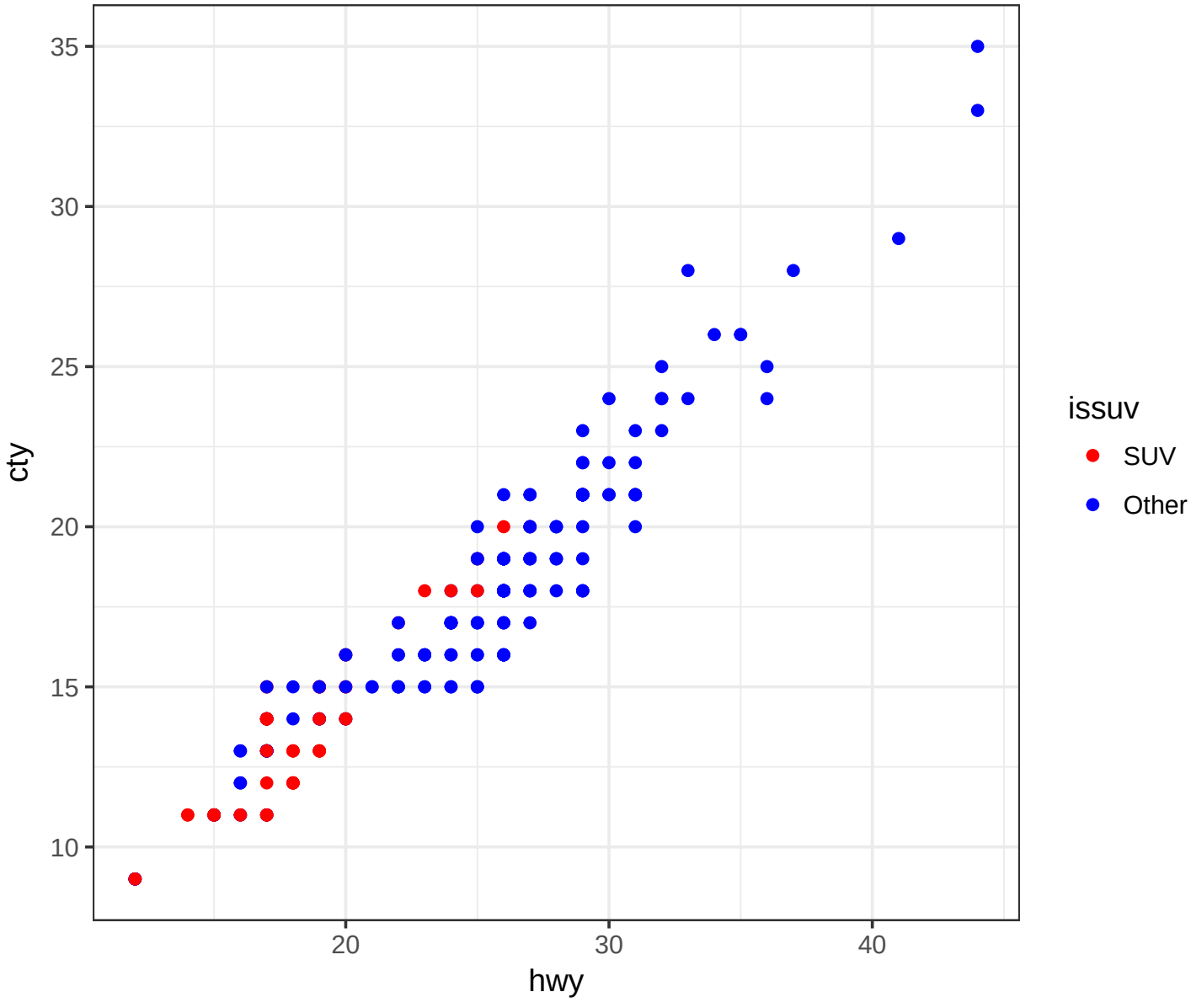
- Scales for specific `aesthetics` can be controled with the following functions (where `*` should be replaced with the desired `aesthetic`, e.g., `color`, `shape`)
  - `scale_*_continuous()`: map continuous values to visual values
  - `scale_*_discrete()`: map discrete values to visual values
  - `scale_*_identity()`: map raw data to visual values (e.g., a data value of 1 equals shape #1)
  - `scale_*_manual()`: map discrete values to manually assigned visual values

Scales: Example

- For example, with the `mpg` data, we wish to plot `hwy` against `cty`, with red points for `class=suv` and blue points otherwise

```
# First, create a new column in the data indicating
# whether the entry has class suv or not
mpg_suv <- mpg %>%
  mutate(issuv = class == 'suv')
# Generate the plot with a manual color scale
p <- ggplot(mpg_suv, aes(x = hwy, y = cty)) +
  geom_point(aes(color = issuv)) +
  scale_color_manual(limits = c(FALSE, TRUE),
                     values = c('blue', 'red'),
                     breaks = c(TRUE, FALSE),
                     labels = c('SUV', 'Other'))
p
```

Scales: Example (figure)



Legends: Positions

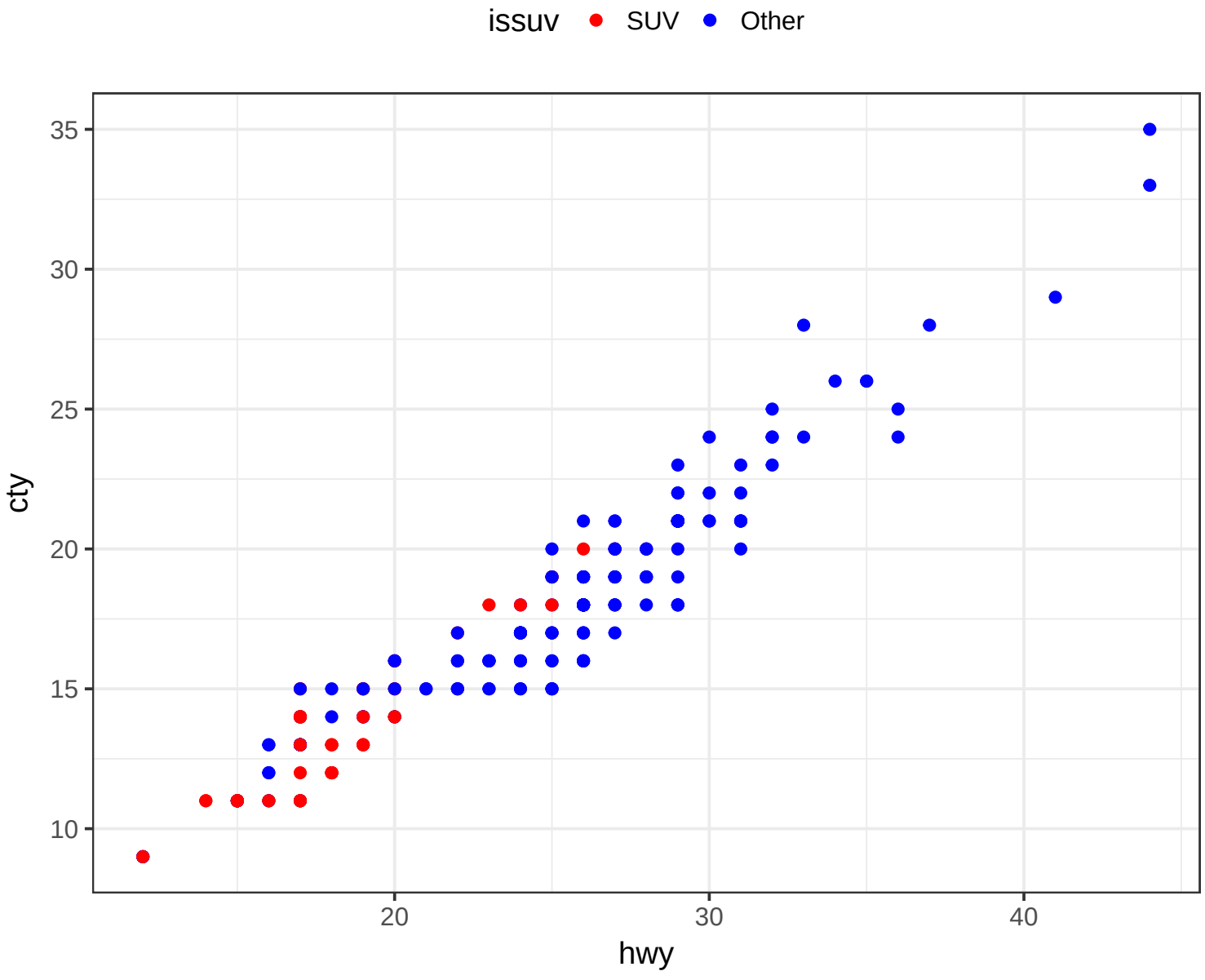
- Changing legend position
- The `legend.justification` determines the 'anchor point' of the legend
- Takes a vector of two numbers between 0 and 1, where the first determines left(0)/right(1) and the second determines bottom(0)/top(1)
- For example, setting `legend.justification` to `c(0, 0)` would anchor the legend at left/bottom, while `c(1, 0)` would anchor the legend at right/bottom

```
# specify either top/left/bottom/right, e.g.,
p <- p + theme(legend.position='top')
# or set x/y coordinates to position within plot
p <- p + theme(legend.position=c(0,1),
               legend.justification=c(0,0))
```

Legends: Position Top Example

```
p <- ggplot(mpg_suv, aes(x = hwy, y = cty)) +
  geom_point(aes(color = issuv)) +
  scale_color_manual(limits = c(FALSE, TRUE),
                     values = c('blue', 'red'),
                     breaks = c(TRUE, FALSE),
                     labels = c('SUV', 'Other')) +
  theme(legend.position = 'top')
p
```

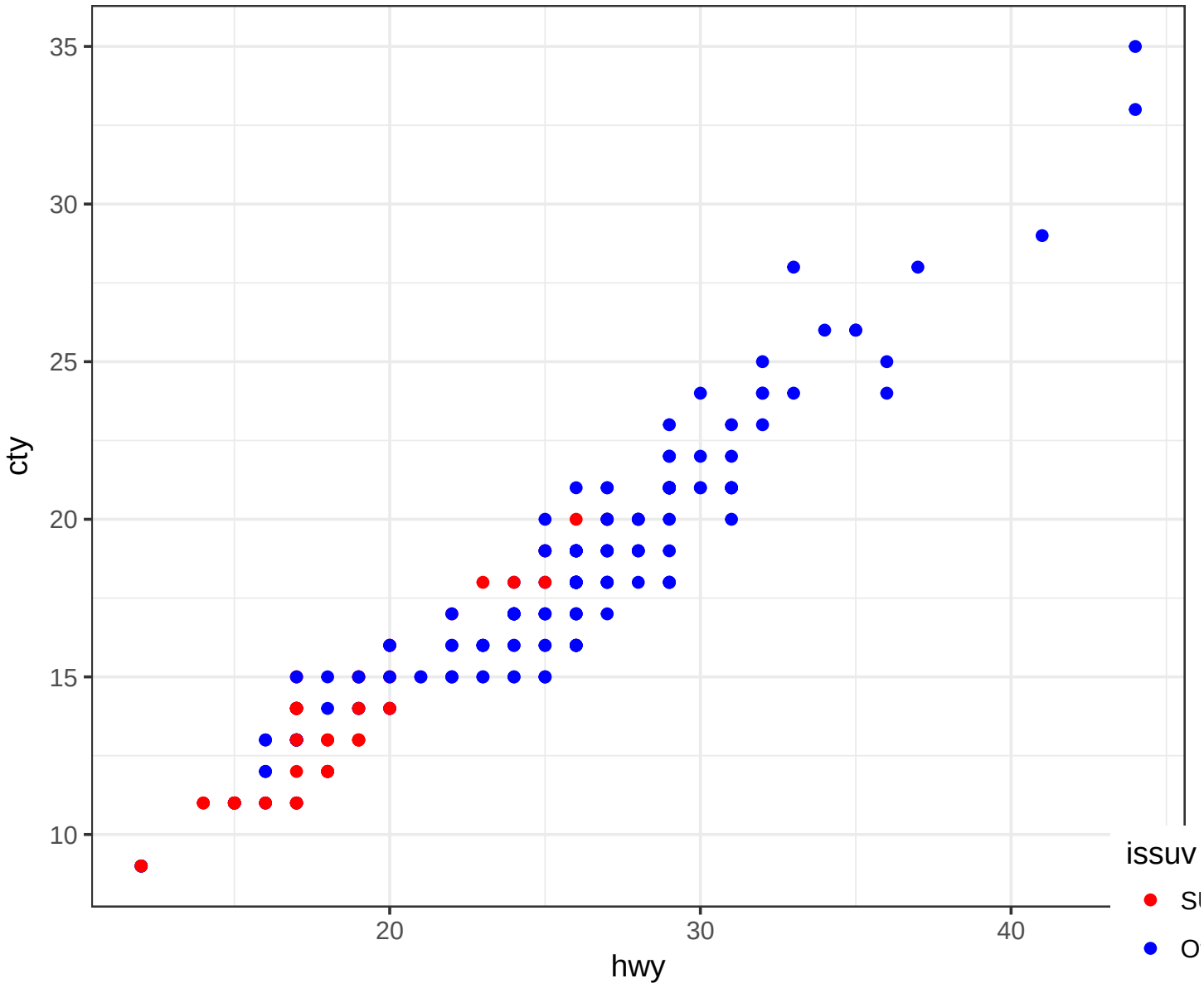
Legends: Position Top Example (figure)



Legends: Position Example

```
p <- ggplot(mpg_suv, aes(x = hwy, y = cty)) +
  geom_point(aes(color = issuv)) +
  scale_color_manual(limits = c(FALSE, TRUE),
                     values = c('blue', 'red'),
                     breaks = c(TRUE, FALSE),
                     labels = c('SUV', 'Other')) +
  theme(legend.position = c(1, 0))
p
```

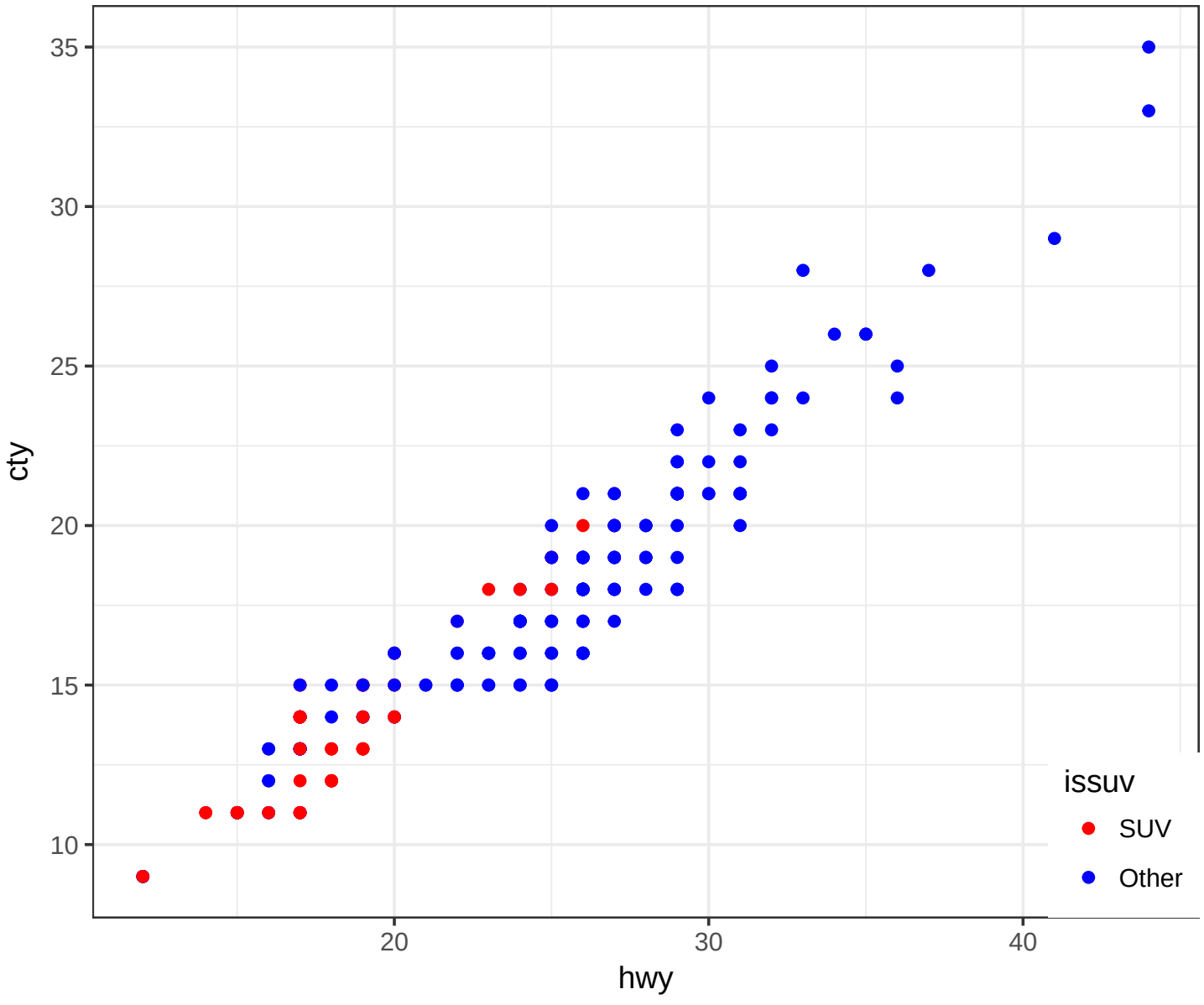
Legends: Position Example (figure)



Legends: Position Example

```
p <- ggplot(mpg_suv, aes(x = hwy, y = cty)) +
  geom_point(aes(color = issuv)) +
  scale_color_manual(limits = c(FALSE, TRUE),
                     values = c('blue', 'red'),
                     breaks = c(TRUE, FALSE),
                     labels = c('SUV', 'Other')) +
  theme(legend.position = c(1, 0),
        legend.justification = c(1, 0))
p
```

Legends: Position Example (figure)



Legends: Guides

- `guides()` is used to set the legend type for specific `aesthetics`
- A common use-case is, for example, to hide legends by setting the `aesthetic` to `none` in `guides()`, e.g.,

```
# Hide color legends
p <- p + guides(color = 'none')
```

Legends: Scales

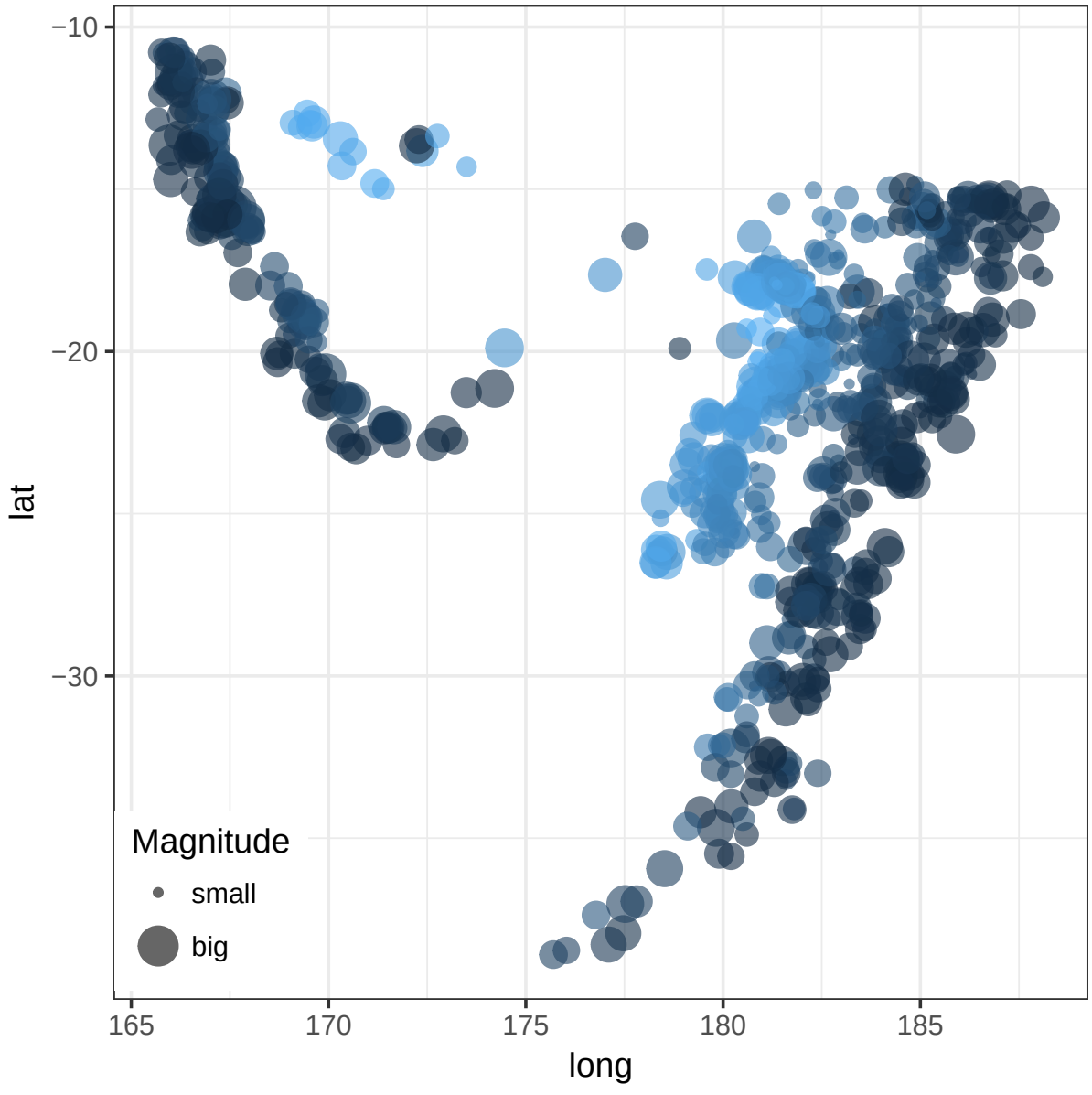
- As shown in the scales section, we can specify the title/breaks/labels for a legend, using the `scale_*_` of the corresponding aesthetic, e.g.,

```
# To modify legend title/labels for a
# continuous variable applied to color aes()
p <- p + scale_color_continuous(
  name = 'legend title',
  breaks = c(1, 5),  # vector of points to label
  labels = c('low', 'high')  # corresponding labels
  )
```

Legends (Example)

```
p <- ggplot(quakes, aes(x = long, y = lat)) +
  geom_point(alpha = .6,
             aes(size = mag, color = depth)) +
  scale_size_continuous(name = 'Magnitude',
                        breaks = c(4, 6),
                        labels = c('small', 'big')) +
  theme(legend.position = c(0, 0),
        legend.justification = c(0, 0)) +
  guides(color = 'none')
p
```

Legends (Example Figure)



Legends: Remarks

- *Lots* of legend options (and more!) can be controlled via `theme()`
- Get familiar with the options in `?theme`

Facets

- Facets let you divide a plot into subplots based on the values of a (discrete) variable
- The notation is

```
# Divide into subplots of 2D grid
# using upto two variables
facet_grid( var.row ~ var.col)

# Wrap into a rectangular layout
# using a single variable
facet_wrap( ~ var, nrow = NULL, ncol = NULL)
```

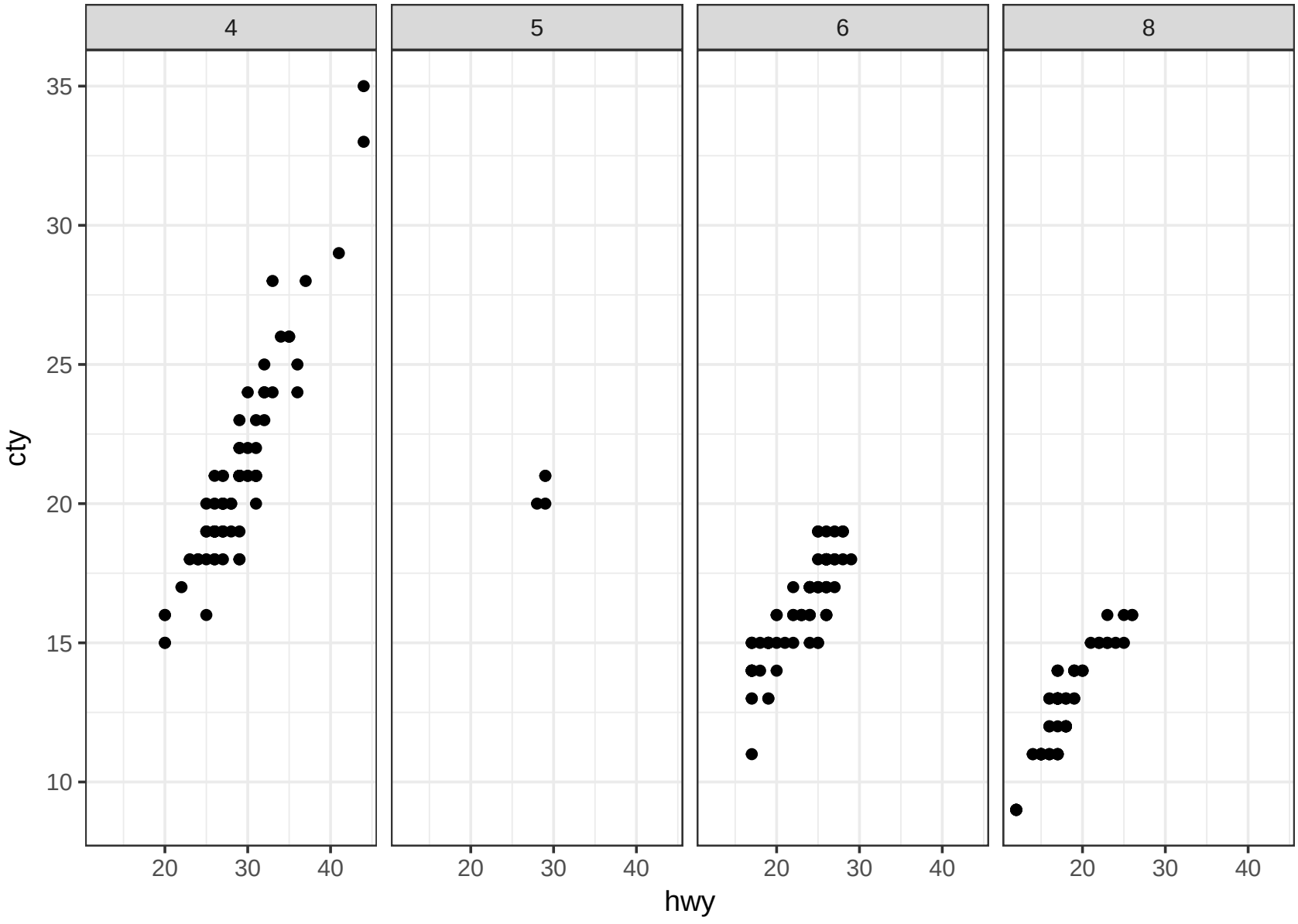
- `var.row` is the name of the variable to use to divide the rows
- `var.col` is the name of the variable to use to divide the columns
- Substitute with '.' to skip either row or column division
- Best demonstrated with examples . . .

Facets: Examples

- `facet_grid()` with one variable (columns)

```
p <- ggplot(mpg, aes(x = hwy, y = cty)) +
  geom_point() +
  facet_grid(. ~ cyl)
p
```

Facets: Examples (figure)

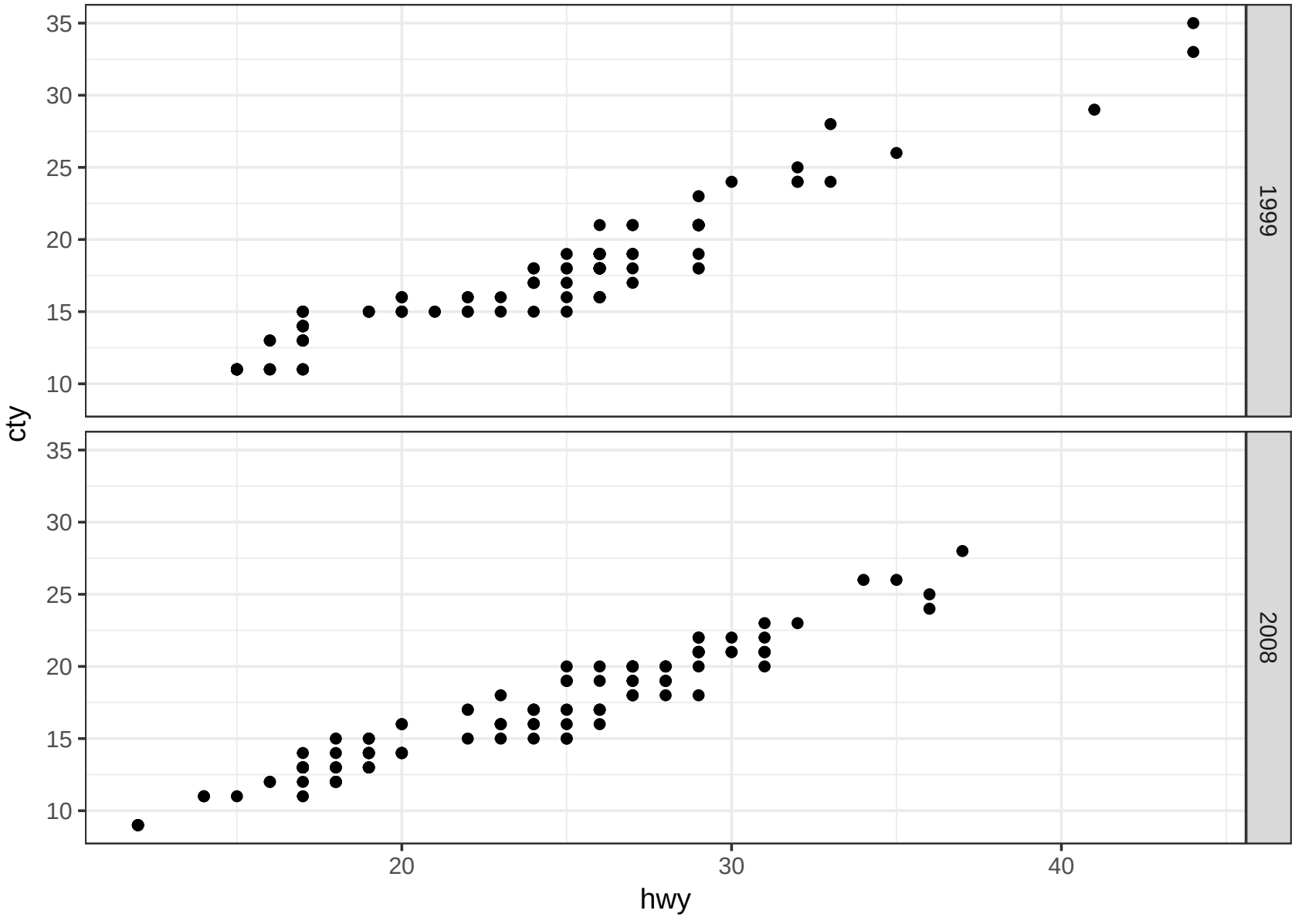


Facets: Examples

- `facet_grid()` with one variable (rows)

```
p <- ggplot(mpg, aes(x = hwy, y = cty)) +
  geom_point() +
  facet_grid(year ~ .)
p
```

Facets: Examples (figure)

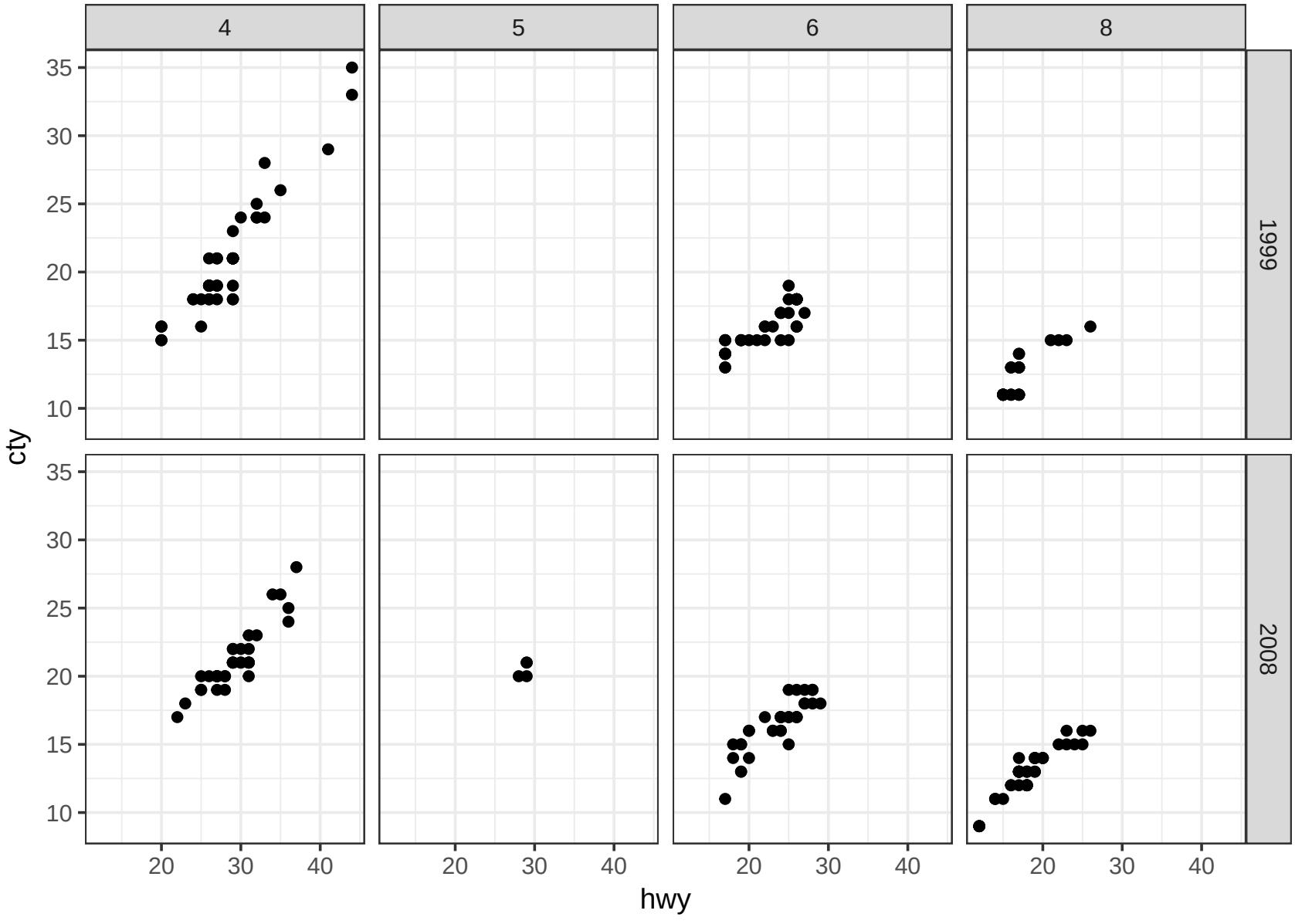


Facets: Examples

- `facet_grid()` with two variables

```
p <- ggplot(mpg, aes(x = hwy, y = cty)) +
  geom_point() +
  facet_grid(year ~ cyl)
p
```

Facets: Examples (figure)

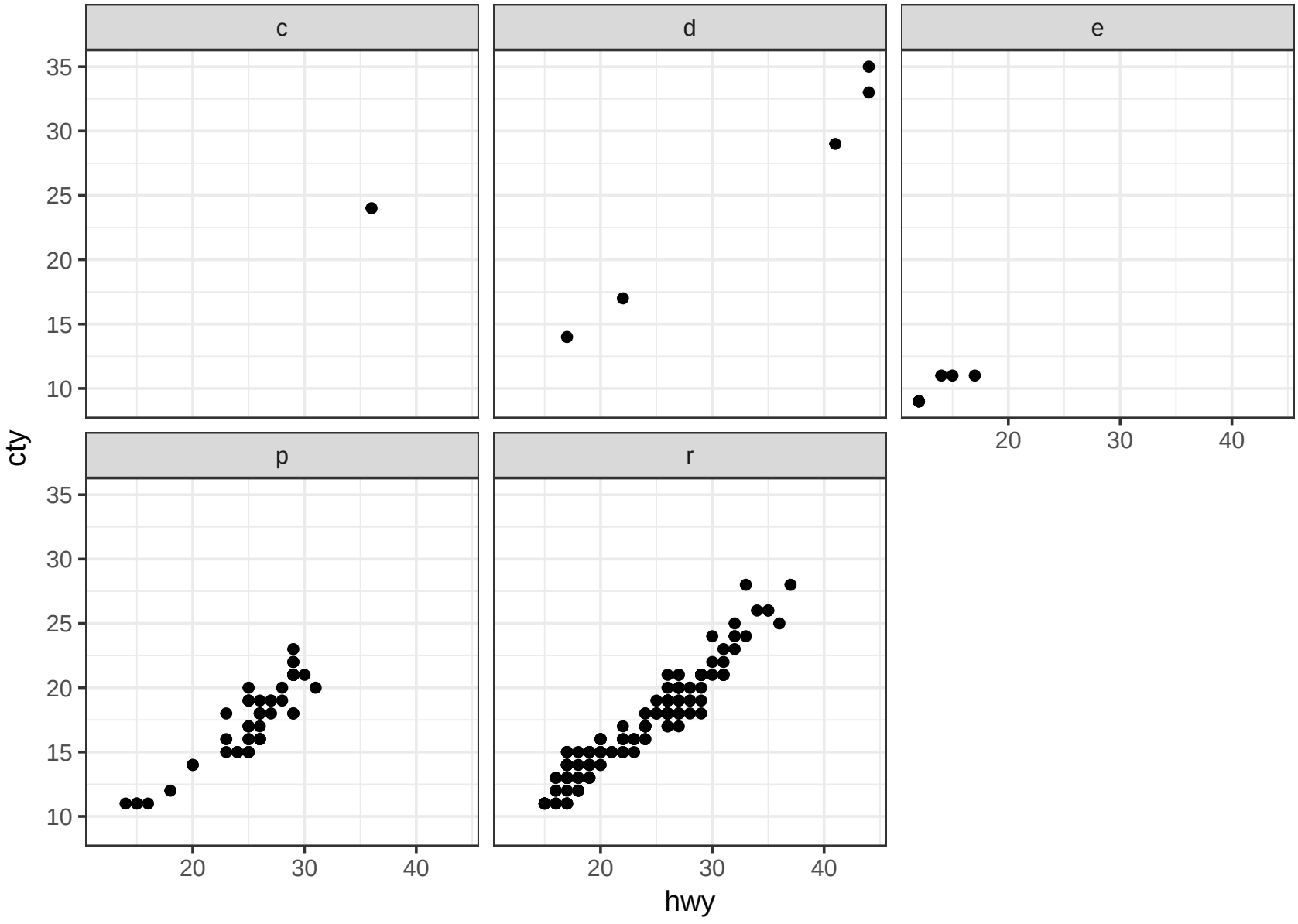


Facets: Examples

- `facet_wrap()`

```
p <- ggplot(mpg, aes(x = hwy, y = cty)) +
  geom_point() +
  facet_wrap(~ fl)
p
```

Facets: Examples (figure)



Exercise

- Now we've covered almost all of the basic concepts of ggplot2!
- Using any of the tools we've covered, how would you best visualize the `mpg` dataset to compare `hwy` against `cty`?
- How would you best visualize any potential insights there might be with regard to unemployment in the `economics` dataset?
- There's no "answer" to this one. Explore and share!

Maps

The map layer with ggmap

- With data that involve spacial coordinates (i.e., long/lat), you might want to use the corresponding geographic map as a 'canvas'
- This is easily achieved in `ggplot` with the `ggmap` package
- Install and load the `ggmap` package like you would any other `R` package

```
# Install, if you haven't already.
# Only need to do this once on a single machine.
install.packages('ggmap')
# load package into workspace
library('ggmap')
```

Initializing a map layer

- Use the `qmap` function to initialize a `ggplot` object with a map layer
- The main arguments for `qmap` are
  - `location`: an address, vector of longitude/latitude pair (in that order), or vector of left/bottom/right/top bounding box
  - `zoom`: the zoom level in integer values from 3 (continent) to 21 (building)
  - `color`: either 'color' or 'bw' (for grayscale)
  - `maptype`: character string providing map theme, e.g., 'terrain', 'satellite', 'roadmap', 'hybrid', etc.
  - `legend`: position for the legend, e.g., 'top', 'right', 'topleft', 'bottomright', 'none', etc.
- `qmap` loads maps from Google maps by default, but can be changed to other sources
- see `?qmap`, `?ggmap`, and `?get_map` for more details

A Map Layer Example: Location Search

```
# Map of stanford by search
p <- qmap(location = 'Stanford, CA',
          zoom = 15, maptype = 'satellite')
p
```

A Map Layer Example

```
# Bounding box around North America
# (I Googled the numbers)
box <- c(left = -52.233, bottom = 5.4996,
         right = -167.2764, top = 83.1621)
p <- qmap(zoom = 3, location = box)
p
```

Exercise

- Plot the `quakes` data as a scatter plot on a map, using a bounding box with an appropriate zoom level. Let the `size` of points represent the `mag` variable, and `color` represent `depth`. Set `alpha` to 0.6. (hint: use the `max`/`min` of `long`/`lat` variables to define the bounding box)

WARNING

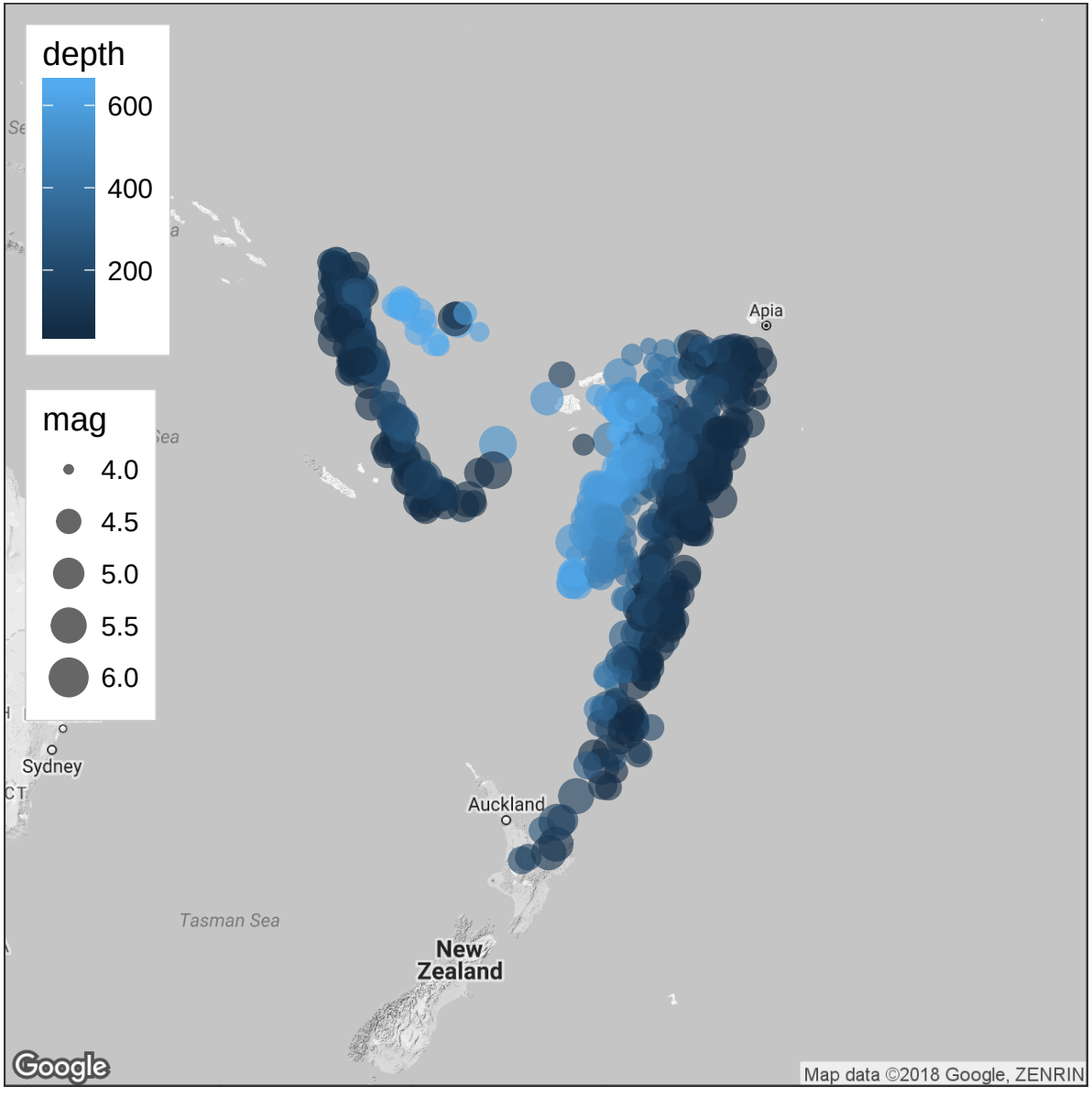
- Solutions to the exercise are presented in the next slide
- Try the exercise before proceeding!

Solution

```
box <- c(left = min(quakes$long),
         bottom = min(quakes$lat),
         right = max(quakes$long),
         top = max(quakes$lat)

p <- qmap(location = box, zoom = 4,
          color = 'bw', legend = 'topleft') +
  geom_point(data = quakes, alpha = 0.6,
             aes(x = long, y = lat,
                 size = mag, color = depth))
p
```

Solution (figure)



Reference

- A great "cheat sheet" for data visualization provided with Rstudio
- See Help $>$ Cheatsheets $>$ Data visualization ...