Introduction to Data Visualization with ggplot2

Knitted by SL in RStudio

2023-01-13

Part 1. Introduction

1. Introduction

```
1. Introduction
Hi and welcome the first course in DataCamp's data visualization with ggplot2 series!

2. Your instructor - Rick Scavetta
My name is Rick Scavetta and I'll be the instructor for this series. I've been training scientists on h

3. Data visualization & data science
Data visualization is an essential skill for data scientists. It combines statistics and design in mean

4. Exploratory versus explanatory
It's important to understand the distinction between exploratory and explanatory visualizations. Explor

5. MASS::mammals
This data set contains the average brain and body weights of 62 land mammals. To understand the relatio

6. A scatter plot
Two mammals, the African and Asian Elephants have both very large brain and body weights, leading to a

7. Explore with a linear model
Here, applying a linear model is a poor choice since a few extreme values have a large influence.

8. Explore: fine-tuning
A log transformation of both variables allows for a better fit. So, although we began with a rough expl

9. Publication-ready plot
In the end, we'd probably want a cleaned-up explanatory plot.

10. Anscombe's plots
Here's a classic example from Francis Anscombe, first published in 1973. When we imagine a linear model

11. Anscombe's plots
something like this. But this same model could be describing a very different set of data

12. Anscombe's plots
such as a parabolic relationship.

13. Anscombe's plots
which calls for a different model.
```

14. Anscombe's plots
or data in which an extreme value has a large effect.

15. Anscombe's plots
which becomes clear when the outlier is removed. And sometimes

16. Anscombe's plots
the model may be describing a relationship where in fact there is none at all

17. Anscombe's plots
because some extreme values may be incorrect.

18. Anscombe's plots
If we relied solely on the numerical output without plotting our data, we'd have missed distinct and in

2. Explore and explain

We made the distinction between plots for exploring and plots for explaining data.

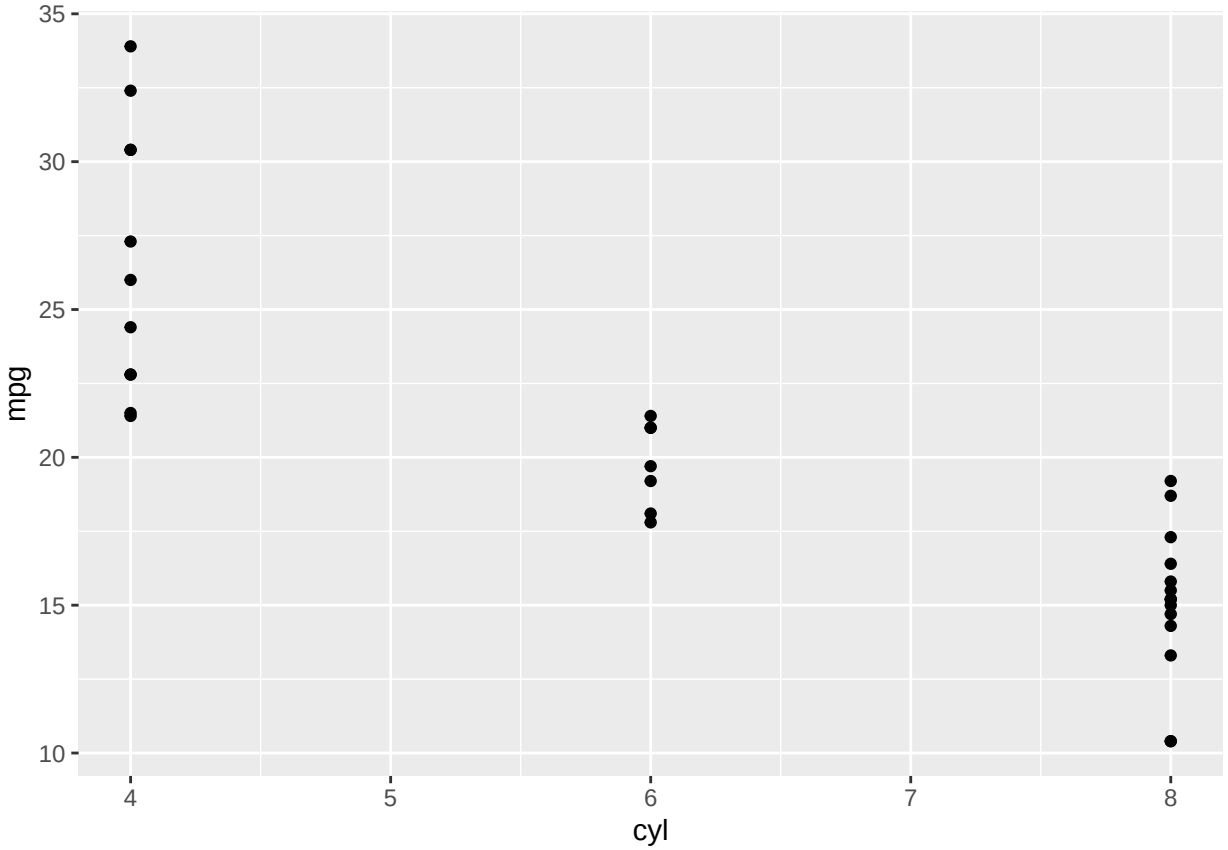
Which of the following are exploratory plots typically NOT?
A. Meant for a specialist audience.
B. Data-heavy.
-> C. Pretty.
D. Rough first drafts.
E. Part of our data science toolkit as graphical data analysis.

3. Drawing your first plot

```
# Load the ggplot2 package
library(ggplot2)

# Explore the mtcars data frame with str()
str(mtcars)
```

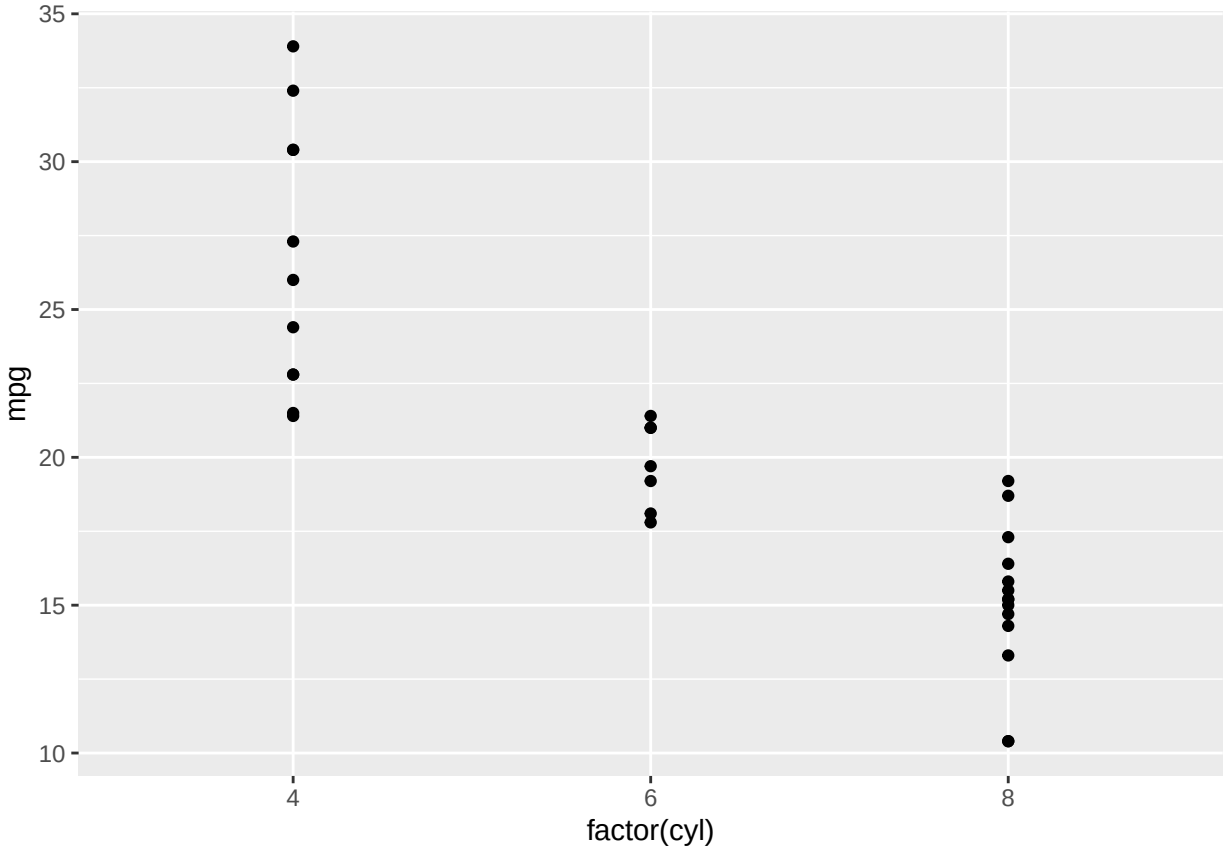
```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
# Execute the following command
ggplot(mtcars, aes(cyl, mpg)) +
  geom_point()
```



4. Data columns types affect plot types

```
# Load the ggplot2 package
library(ggplot2)

# Change the command below so that cyl is treated as factor
ggplot(mtcars, aes(x=factor(cyl), mpg)) +
  geom_point()
```



5. The grammar of graphics

1. The grammar of graphics
The first step in thinking creatively about data visualization is to appreciate that graphics are built

2. The quick brown fox jumps over the lazy dog
To begin, let's consider one of the most well-known sentences in English. The quick brown fox jumps ove

3. The quick brown fox jumps over the lazy dog
Every word in the sentence has a clear grammatical definition and when we write text, we take great car

4. Grammar of graphics
The same concept holds true for data visualization - graphics are built on an underlying grammar. The g

5. The three essential grammatical elements
Let's explore grammatical elements first. There are three essential grammatical elements: data, aesthet

6. Course 1: core competency
The rest are optional layers. This includes the theme layer, which controls all the non-data ink. In th

7. The seven grammatical elements
In the next course we'll explore the remaining grammatical elements: the statistics, coordinates and fa

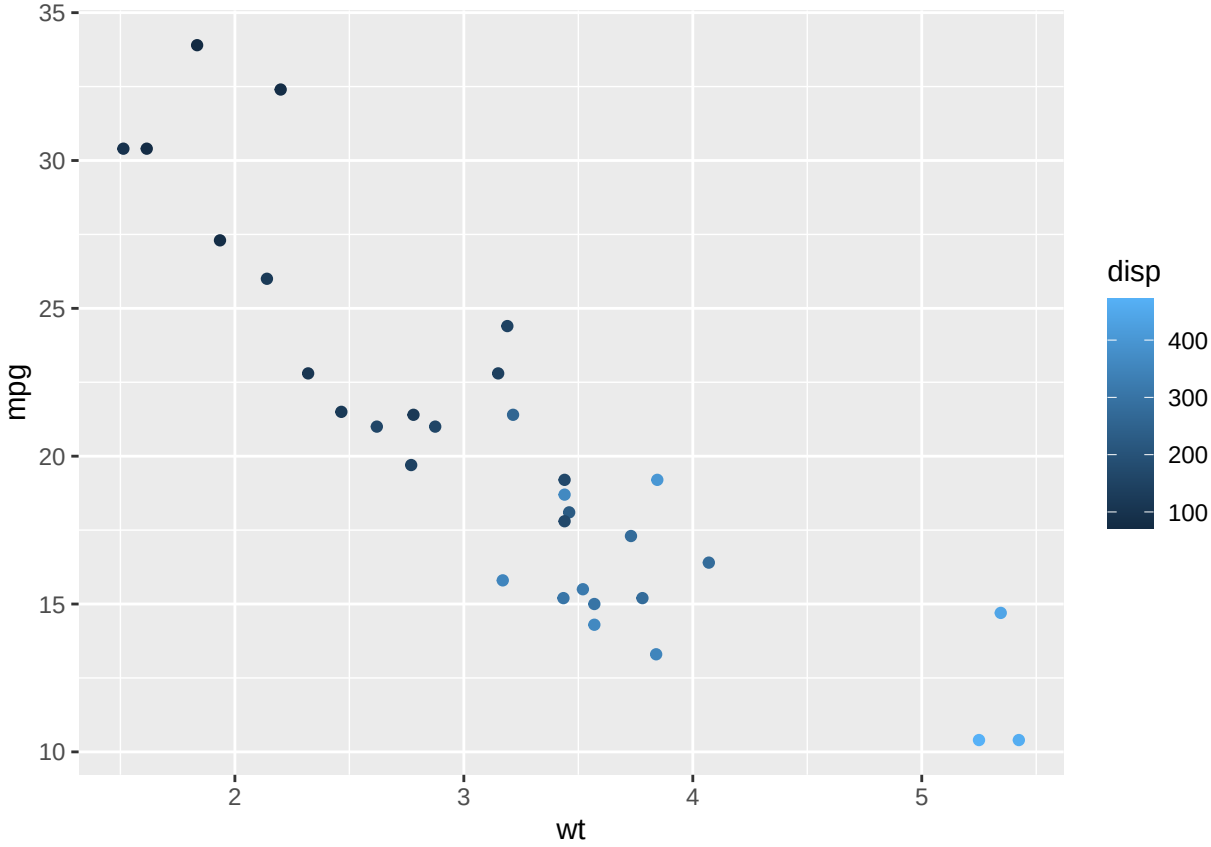
8. Jargon for each element
This diagram gives an example of some of the terms we'll encounter in each element. Whenever we make a

9. Course 2: Tools for EDA

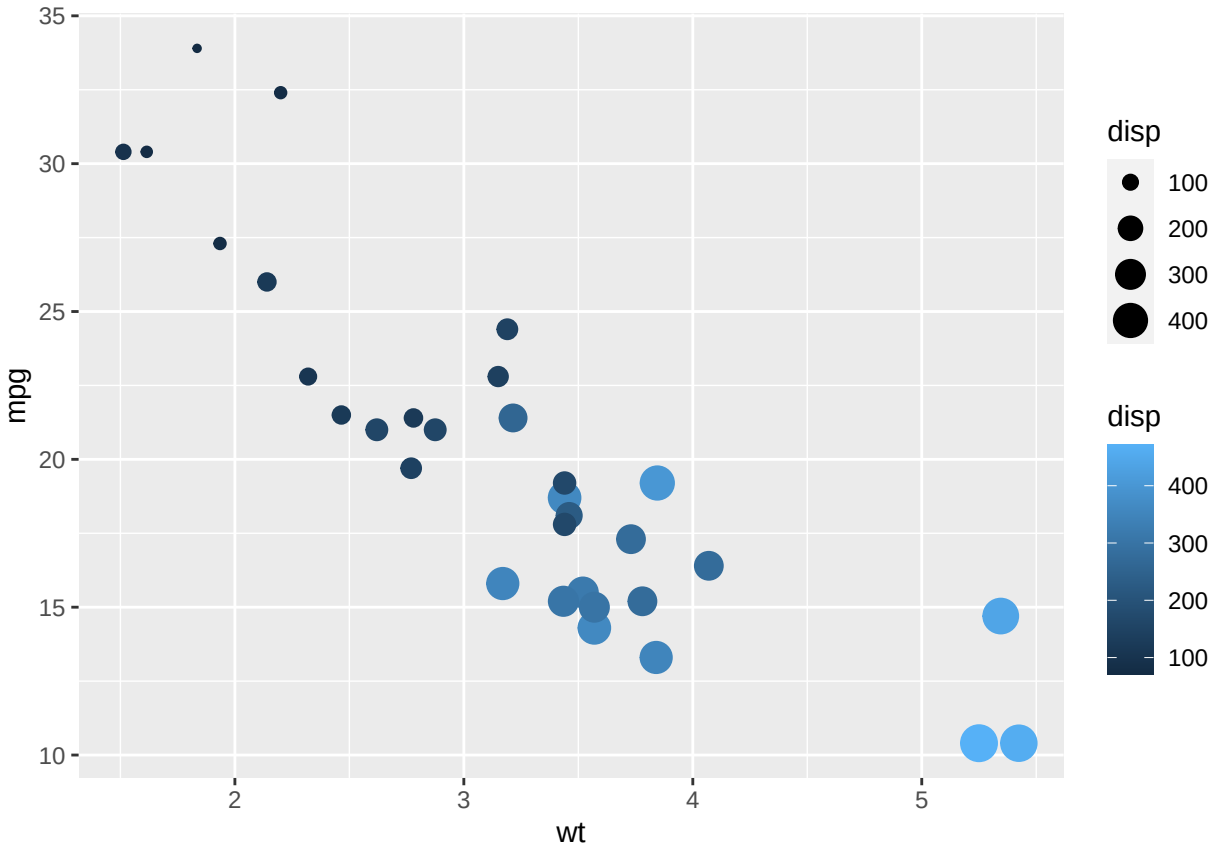
Once we've covered the remaining three layers in the second course, we'll be using data viz as a tool f

6. Mapping data columns to aesthetics

```
# Edit to add a color aesthetic mapped to disp
ggplot(mtcars, aes(x=wt, y=mpg, color=disp)) +
  geom_point()
```




```
# Change the color aesthetic to a size aesthetic
ggplot(mtcars, aes(wt, mpg, color = disp, size=disp)) +
  geom_point()
```



7. Understanding variables

```
If you try this command in the console:
ggplot(mtcars, aes(wt, mpg, shape = disp)) +
  geom_point()
It gives an error. What does this mean?
-> shape only makes sense with categorical data, and disp is continuous.
```

8. ggplot2 layers

```
1. ggplot2 layers
Now that we have some idea about the different grammatical elements of graphics, let's see how this work

2. ggplot2 package
The grammar of graphic is implemented in R using the ggplot2 package. There are two key functions that g

3. Data
The bottom layer is the data element. Obviously we need some data to plot. I'm going to use several diff

4. Iris dataset
one of which is the classic iris data set collected by Edgar Anderson in the 1930s and thereafter popula

1 Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7, Pa
2 Anderson, Edgar (1935). The irises of the Gaspe Peninsula, Bulletin of the American Iris Society, 59,
5. Iris dataset
The data is stored in an object called iris, there are five variables: the species and one for each of t
```

6. Aesthetics
The next layer we'll add is the aesthetics element, which tells us which scales we should map our data

7. Iris aesthetics
In this case we are going to make a scatter plot so we're going to map Sepal-dot-Length onto the X aest

8. Geometries
The next element is the geometry element. This allows us to choose how the plot will look.

9. Iris geometries
After we've established our three essential layers, we have enough instructions to make a basic scatter

10. Themes
The next layer we'll look at is the themes element. It controls all the non-data ink on our plot.

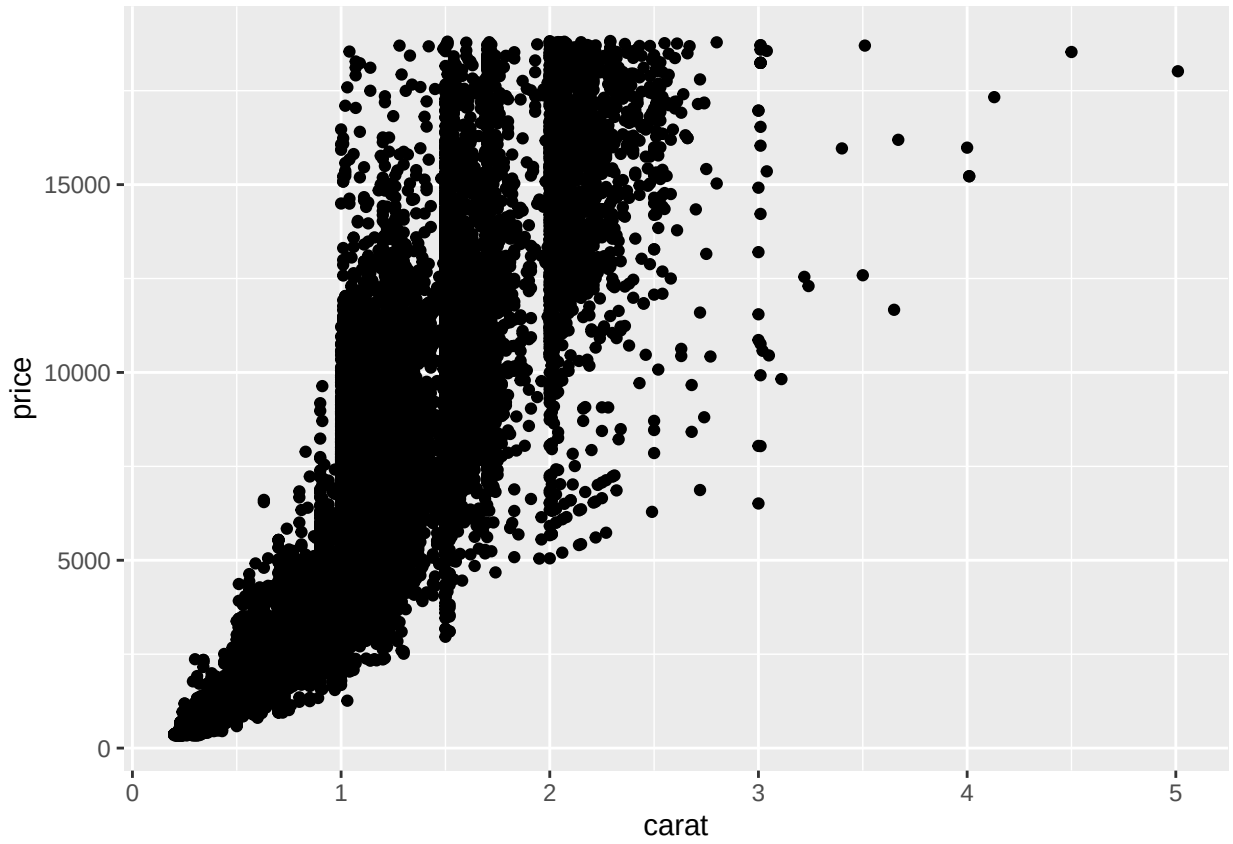
11. Iris themes
Which allows us to get a nice looking, meaningful and publication-quality plot directly in R.

9. Adding geometries

```
# Explore the diamonds data frame with str()
str(diamonds)
```

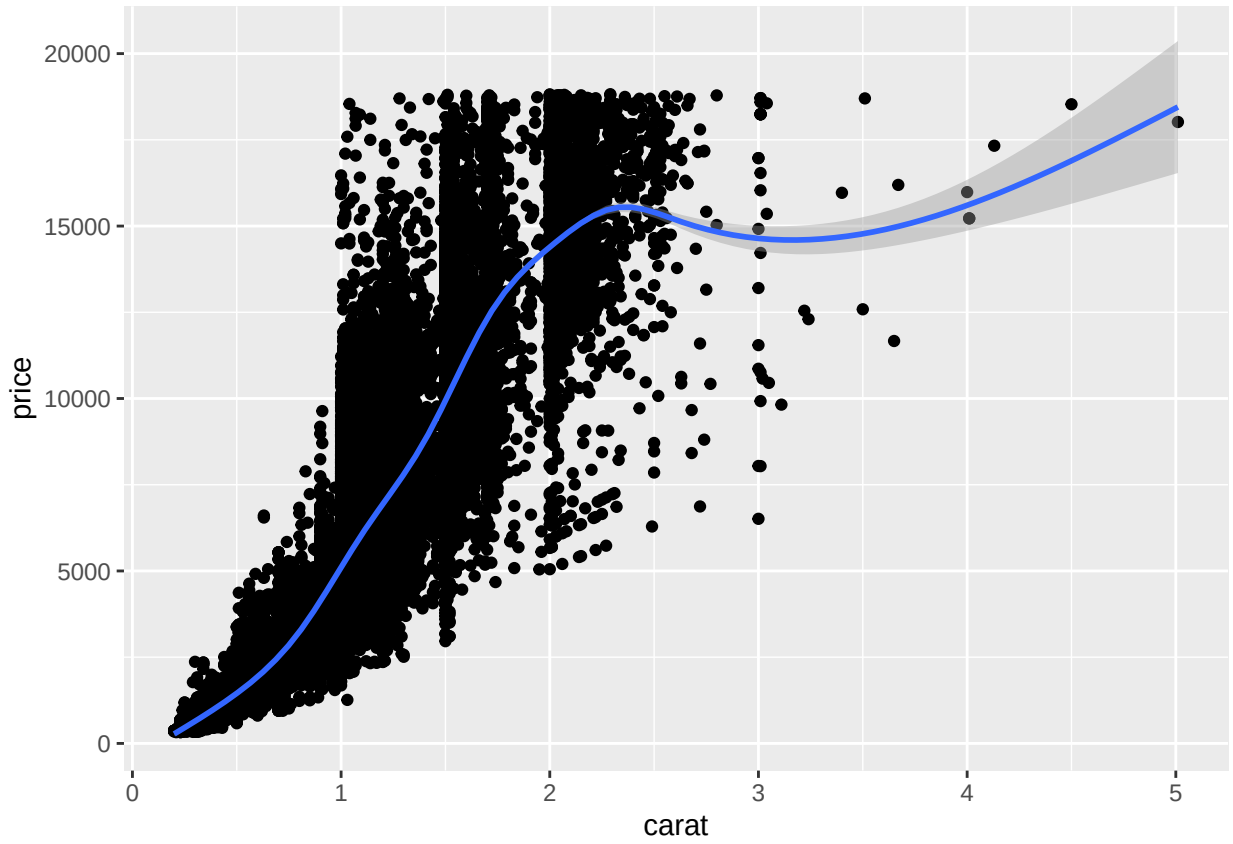
```
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
# Add geom_point() with +
ggplot(diamonds, aes(carat, price)) +
  geom_point()
```




```
# Add geom_smooth() with +
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  geom_smooth()
```

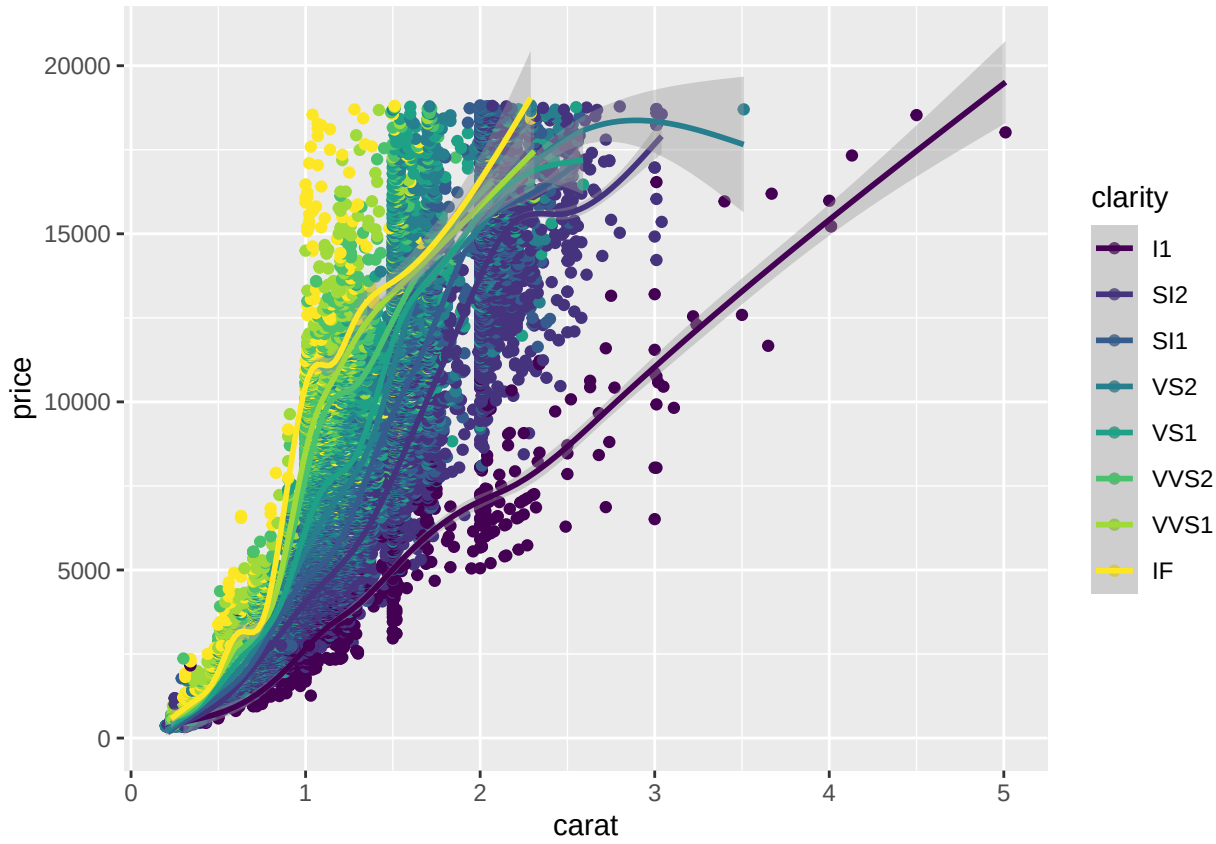
```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



10. Changing one geom or every geom

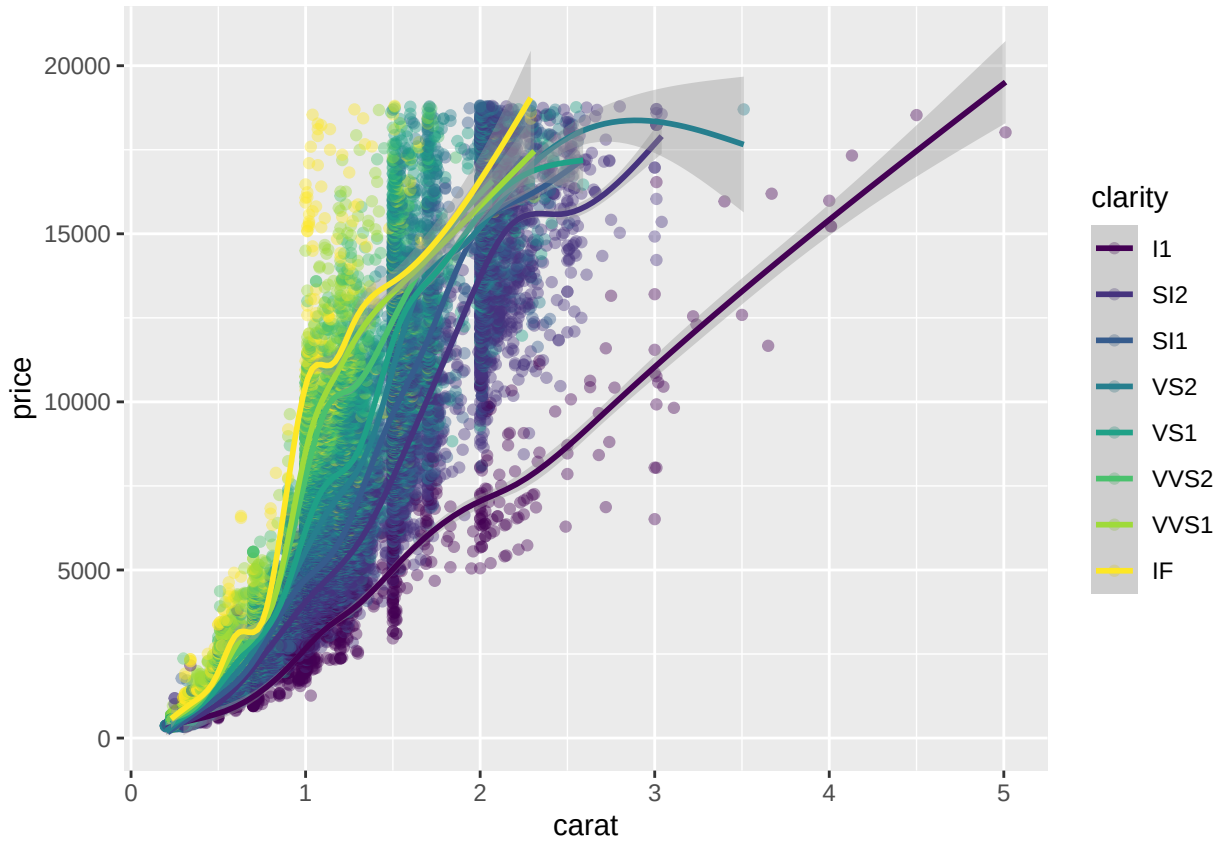
```
# Map the color aesthetic to clarity
ggplot(diamonds, aes(carat, price, color=clarity)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



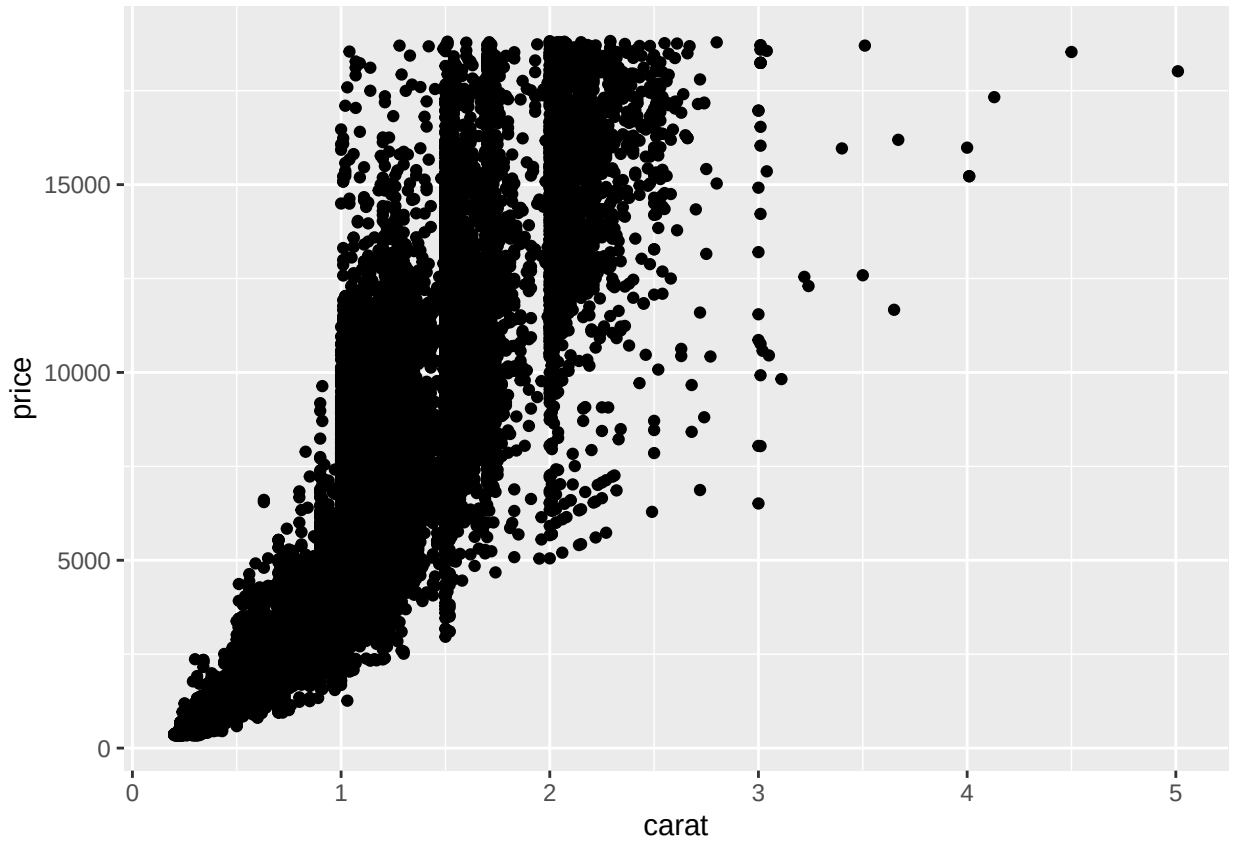

```
# Make the points 40% opaque
ggplot(diamonds, aes(carat, price, color = clarity)) +
  geom_point(alpha=.4) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```

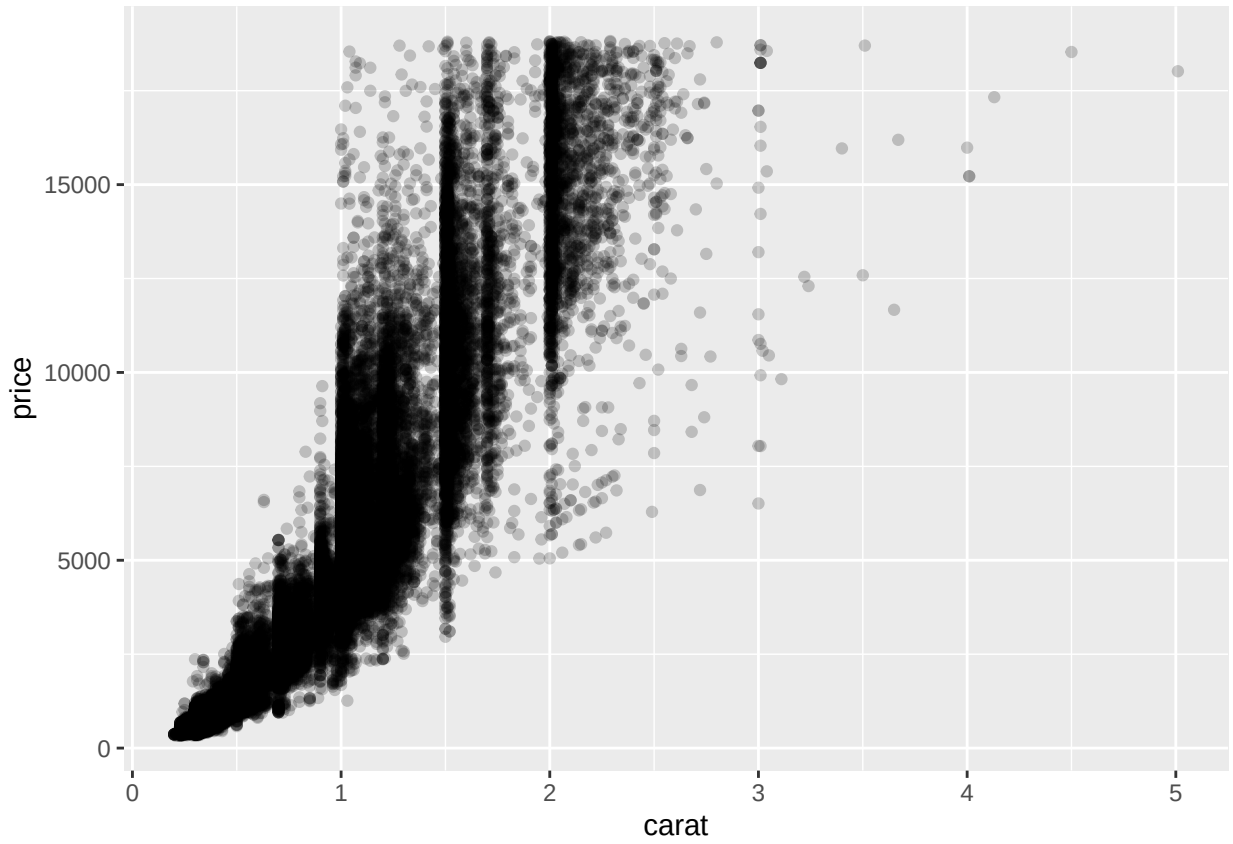


11. Saving plots as variables

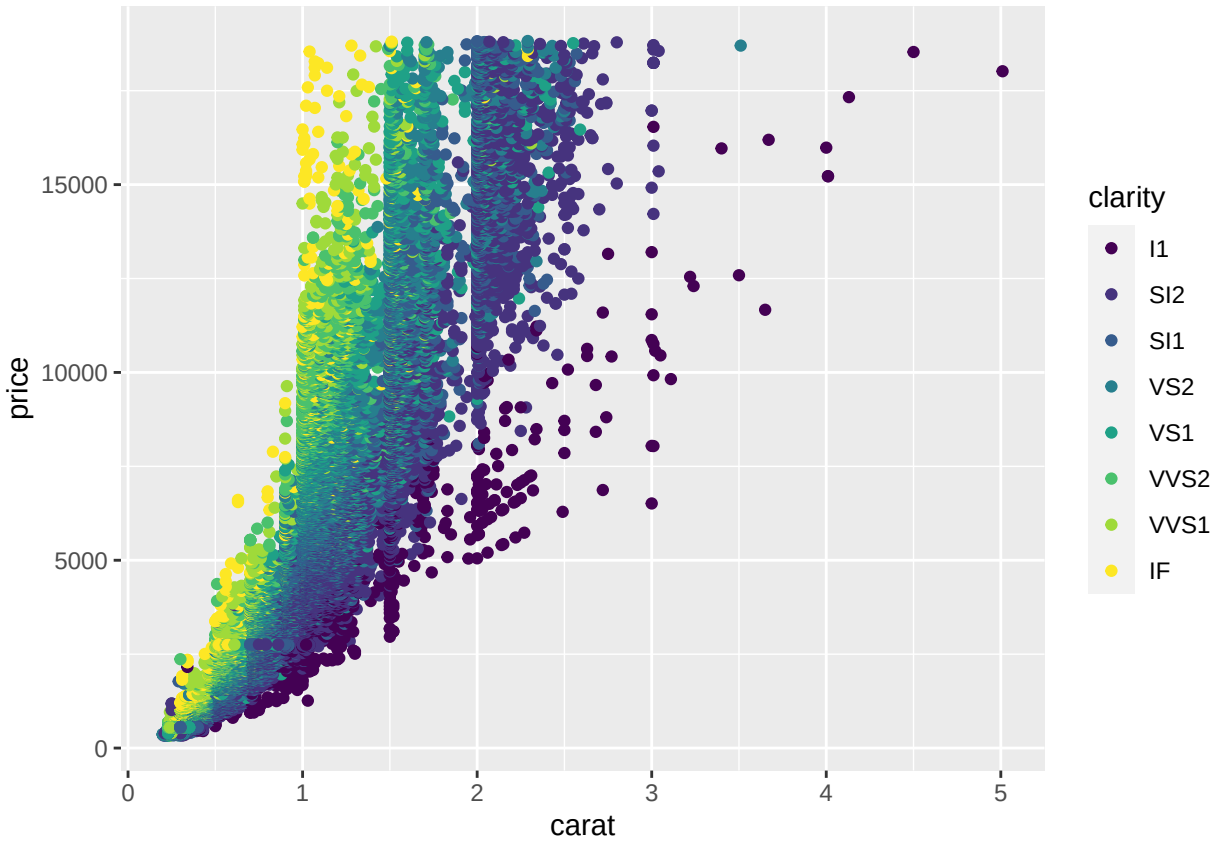
```
# Draw a ggplot
plt_price_vs_carat <- ggplot(
  # Use the diamonds dataset
  diamonds,
  # For the aesthetics, map x to carat and y to price
  aes(carat, price)
)

# Add a point layer to plt_price_vs_carat
plt_price_vs_carat + geom_point()
```




```
# From previous step
plt_price_vs_carat <- ggplot(diamonds, aes(carat, price))

# Edit this to make points 20% opaque: plt_price_vs_carat_transparent
plt_price_vs_carat_transparent <- plt_price_vs_carat + geom_point(alpha=.2)

# See the plot
plt_price_vs_carat_transparent
```




```
# From previous step
plt_price_vs_carat <- ggplot(diamonds, aes(carat, price))

# Edit this to map color to clarity,
# Assign the updated plot to a new object
plt_price_vs_carat_by_clarity <- plt_price_vs_carat + geom_point(aes(color=clarity))

# See the plot
plt_price_vs_carat_by_clarity
```



Part 2. Aesthetics

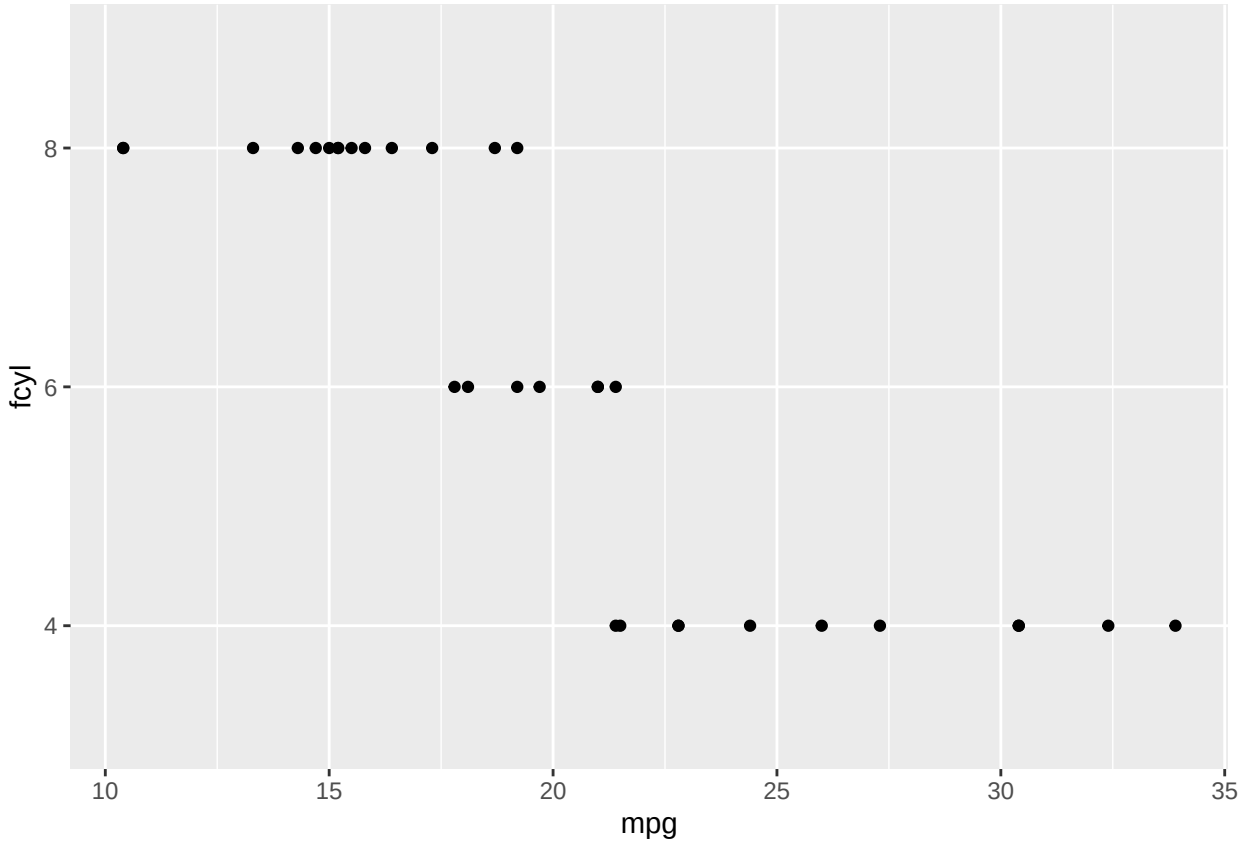
1. Visible aesthetics

```
1. Visible aesthetics
In this section we'll explore aesthetics, and understand how they are distinct from attributes.

2. Mapping onto the X and Y axes
In ggplot2, the mapping of aesthetics elements is a key concept to master. So what do we mean by mapping

3. Mapping onto color
For example, the variable Species can be mapped onto the color aesthetic, which colors the points accor

4. Mapping onto the color aesthetic
That is, we map a variable from our dataframe onto one of the visible aesthetics. We call a column in ou

5. Mapping onto the color aesthetic
Importantly, we call aesthetics in the aes function. We could have also called aesthetics in the geom la

6. Mapping onto the color aesthetic in geom
as shown here, and get the same result. This is typically only done if we don't want all layers to inhe

7. Typical visible aesthetics
In addition to the X and Y axes and color, typical visible aesthetics include

8. Typical visible aesthetics
fill, which is distinct from
```

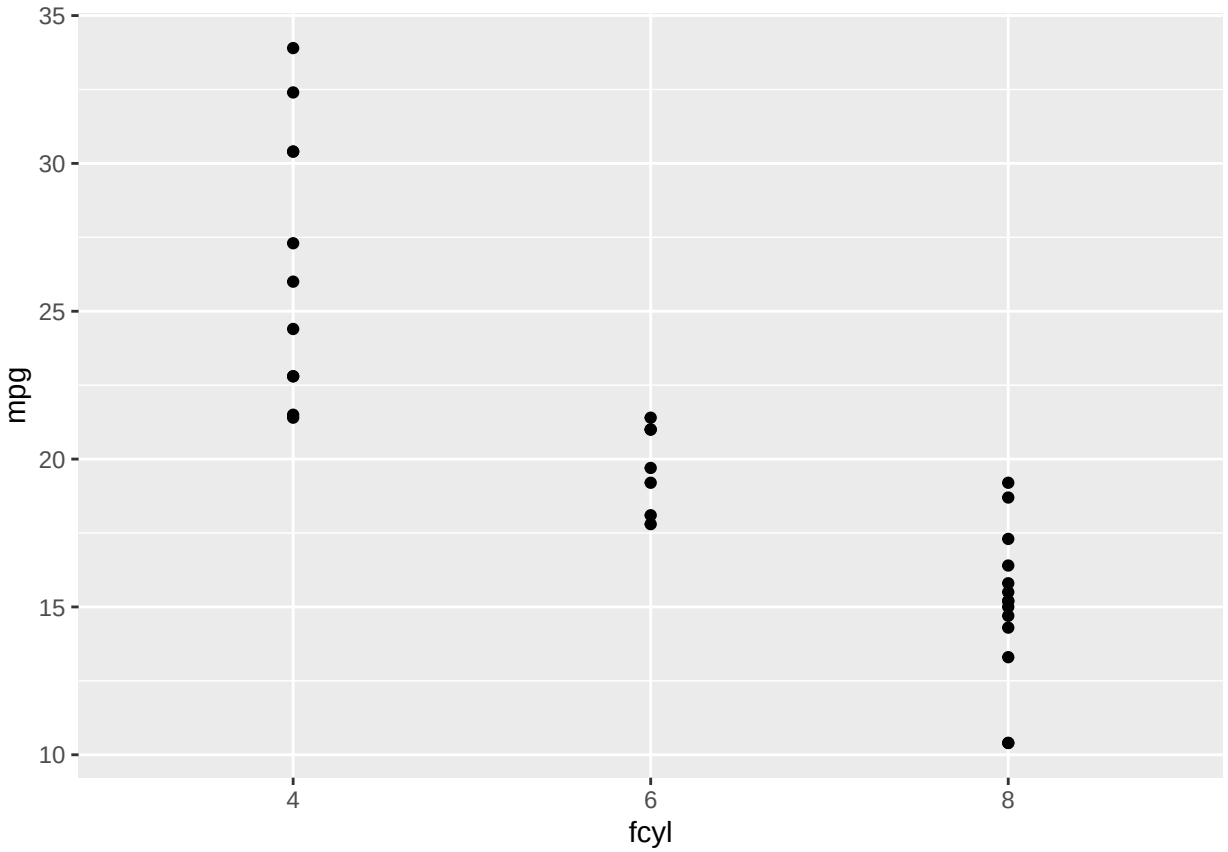
```
9. Typical visible aesthetics
color in that color usually, but not always, refers to the outline of a shape.

10. Typical visible aesthetics
Size adjusts the area or radius of points, the thickness of lines and the font size of text.

11. Typical visible aesthetics
alpha refers to alpha-blending, which adjusts the transparency of a shape.

12. Typical visible aesthetics
line type refers to the dash pattern of a line and

13. Typical visible aesthetics
labels are direct labels of an item, directly on the plot. Like printing an item's name on a scatter pl

14. Typical visible aesthetics
Shape refers to the shape of a point. Many of these aesthetics function as both aesthetic mappings as w
```

2. All about aesthetics: color, shape and size

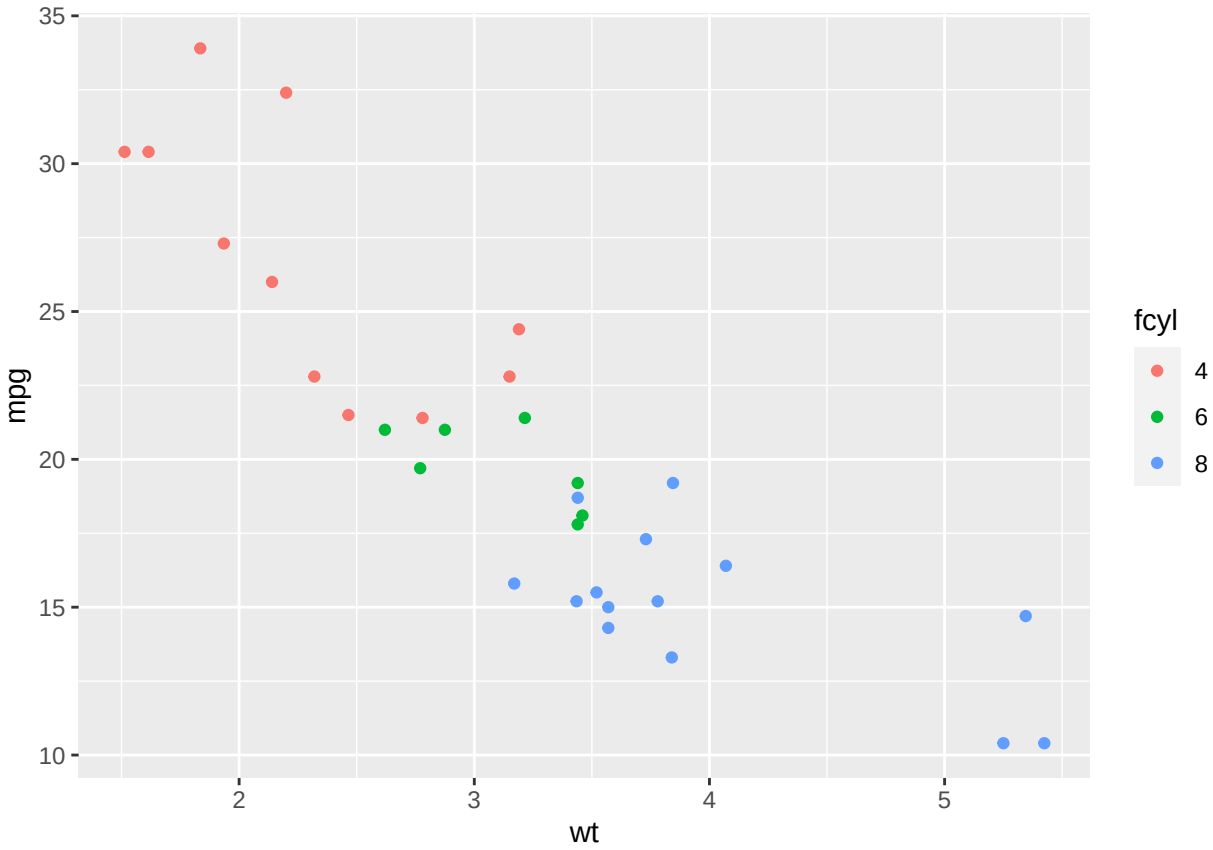
```r
# create factor fcyl
mtcars$fcyl <- as.factor(mtcars$cyl)

# Map x to mpg and y to fcyl
ggplot(mtcars, aes(mpg, fcyl)) +
  geom_point()
```



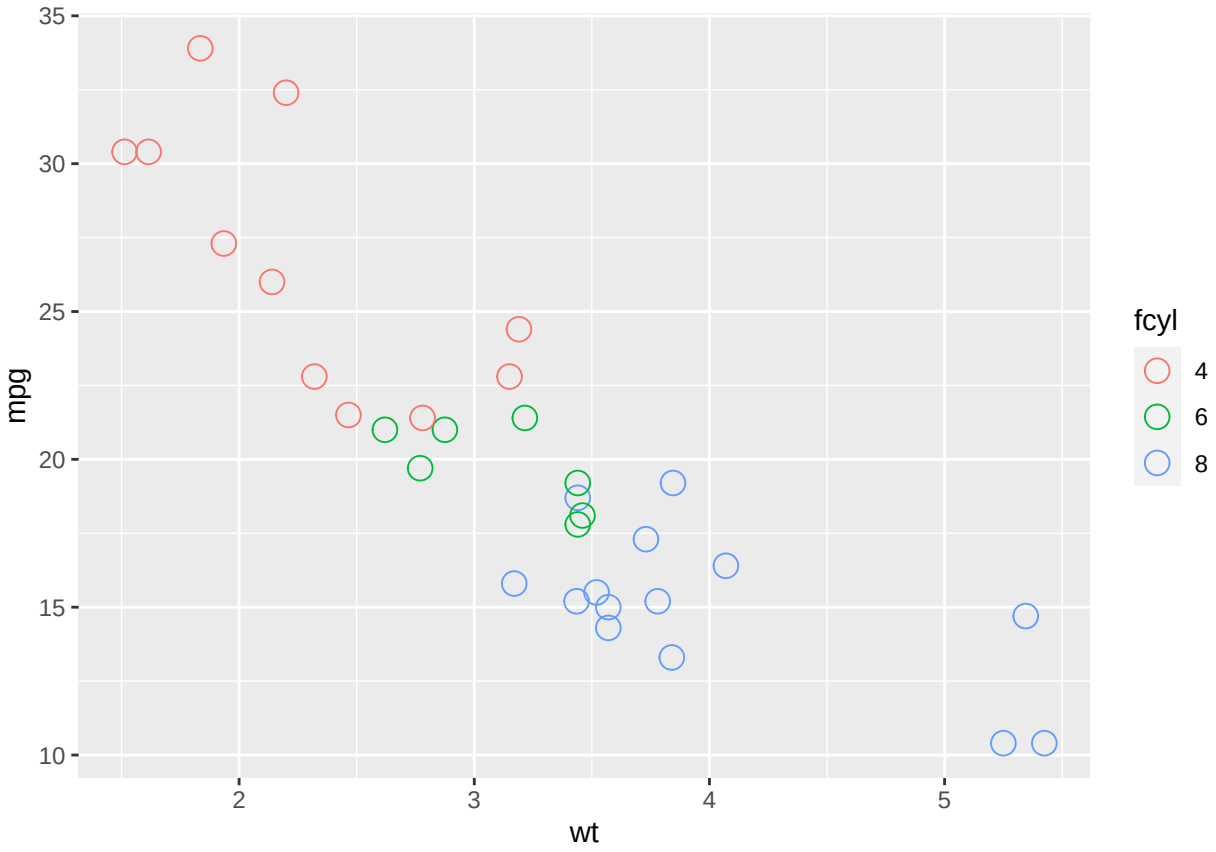
```
# Swap mpg and fcyl
ggplot(mtcars, aes(fcyl, mpg)) +
  geom_point()
```




```
# Map x to wt, y to mpg and color to fcyl
ggplot(mtcars, aes(x=wt, y=mpg, color=fcyl)) +
  geom_point()
```

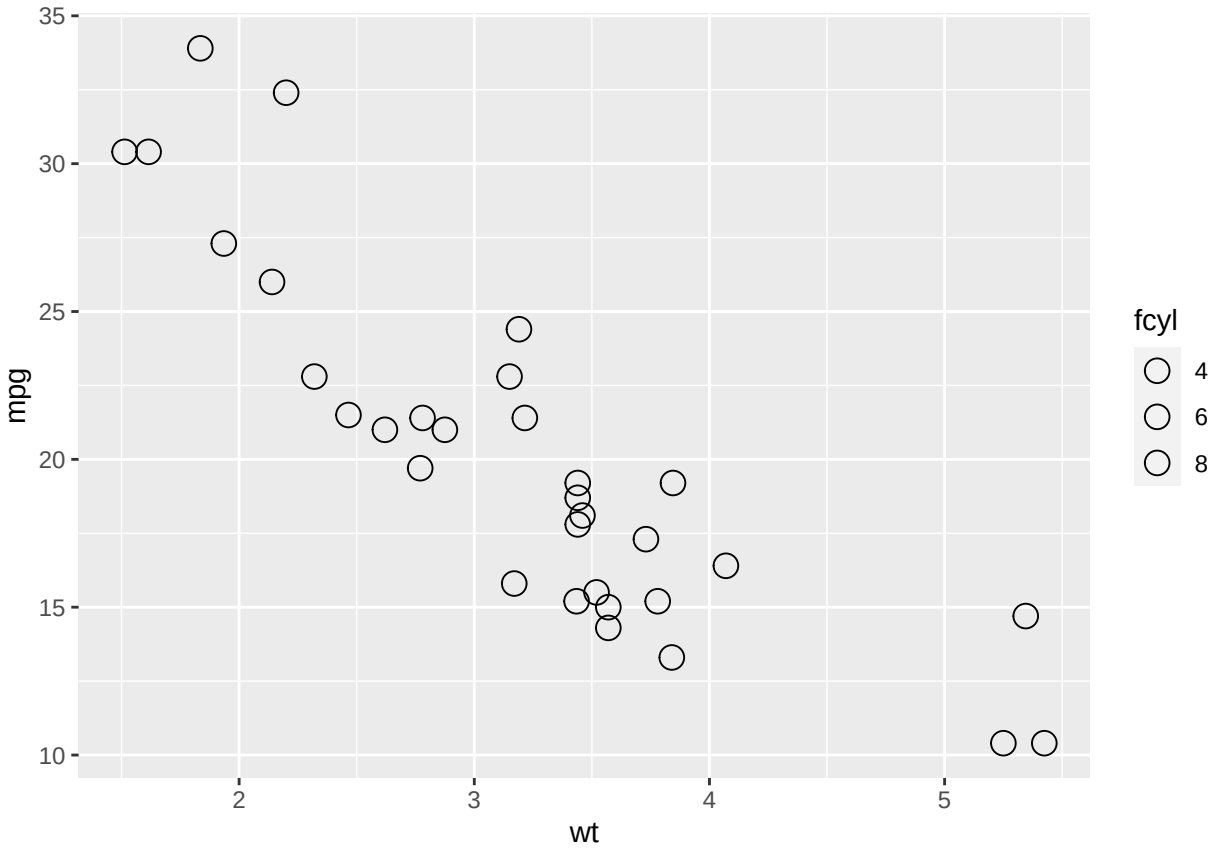



```
ggplot(mtcars, aes(wt, mpg, color = fcyl)) +
  # Set the shape and size of the points
  geom_point(shape=1, size=4)
```

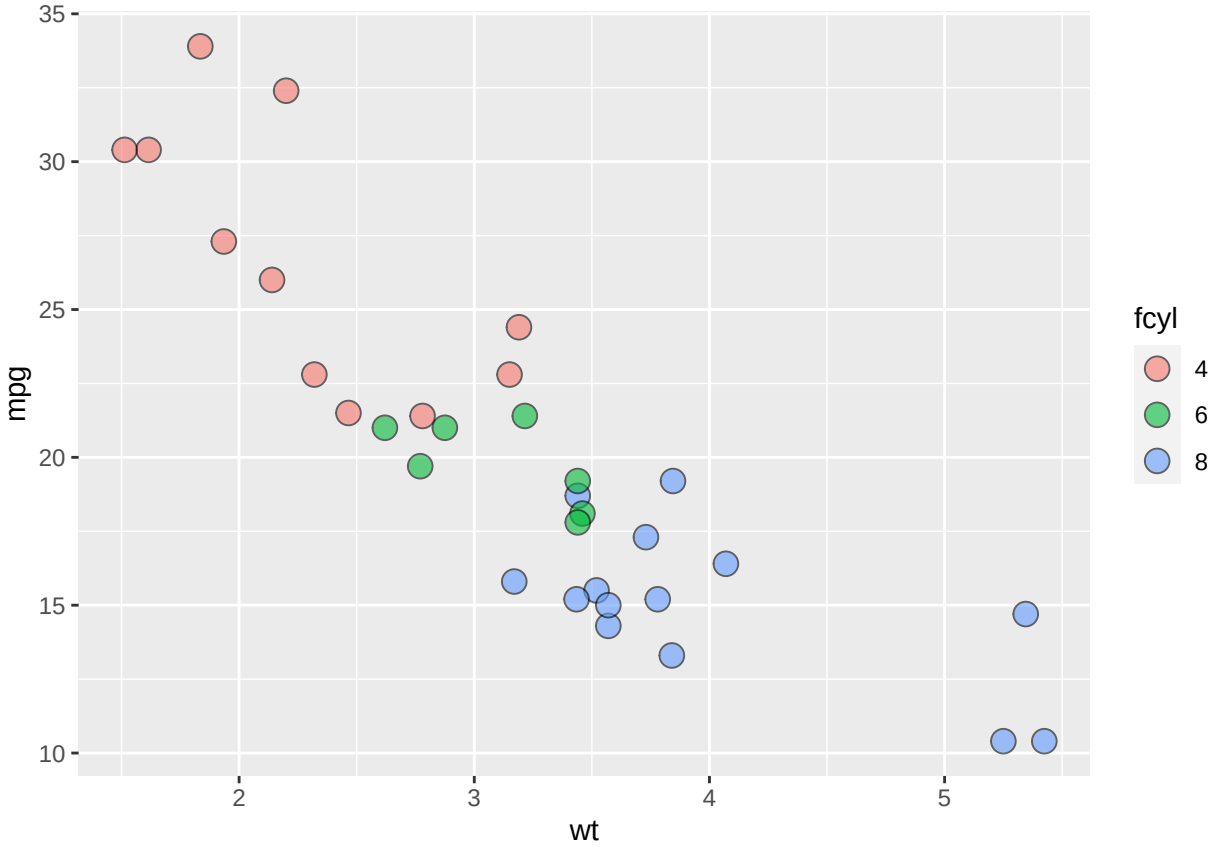


3. All about aesthetics: color vs. fill

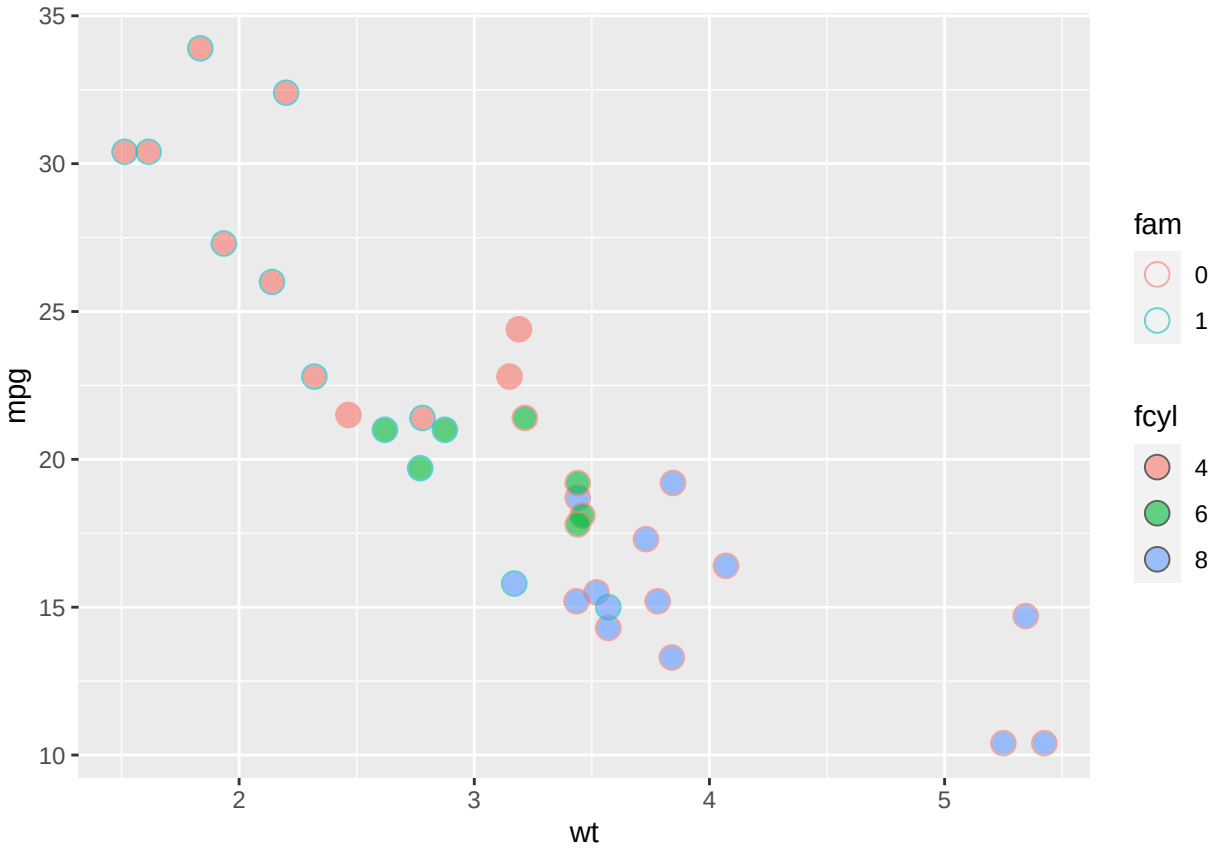
```
# create fcyl
mtcars$fcyl <- as.factor(mtcars$cyl)


# Map fcyl to fill
ggplot(mtcars, aes(wt, mpg, fill = fcyl)) +
  geom_point(shape = 1, size = 4)
```




```
ggplot(mtcars, aes(wt, mpg, fill = fcyl)) +
  # Change point shape; set alpha
  geom_point(shape = 21, size = 4, alpha=.6)
```



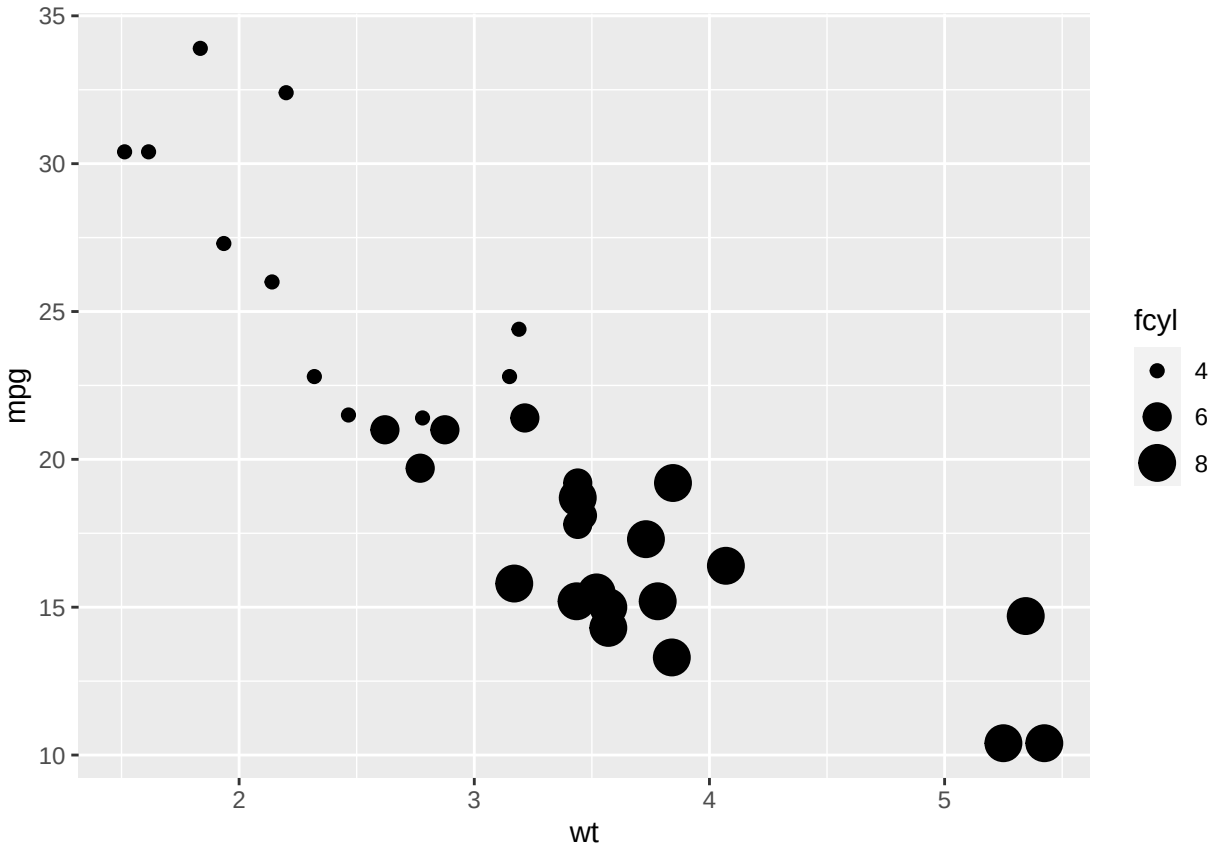

```
#create fam
mtcars$fam <- as.factor(mtcars$am)

# Map color to fam
ggplot(mtcars, aes(wt, mpg, fill = fcyl, color=fam)) +
  geom_point(shape = 21, size = 4, alpha = 0.6)
```



4. All about aesthetics: comparing aesthetics

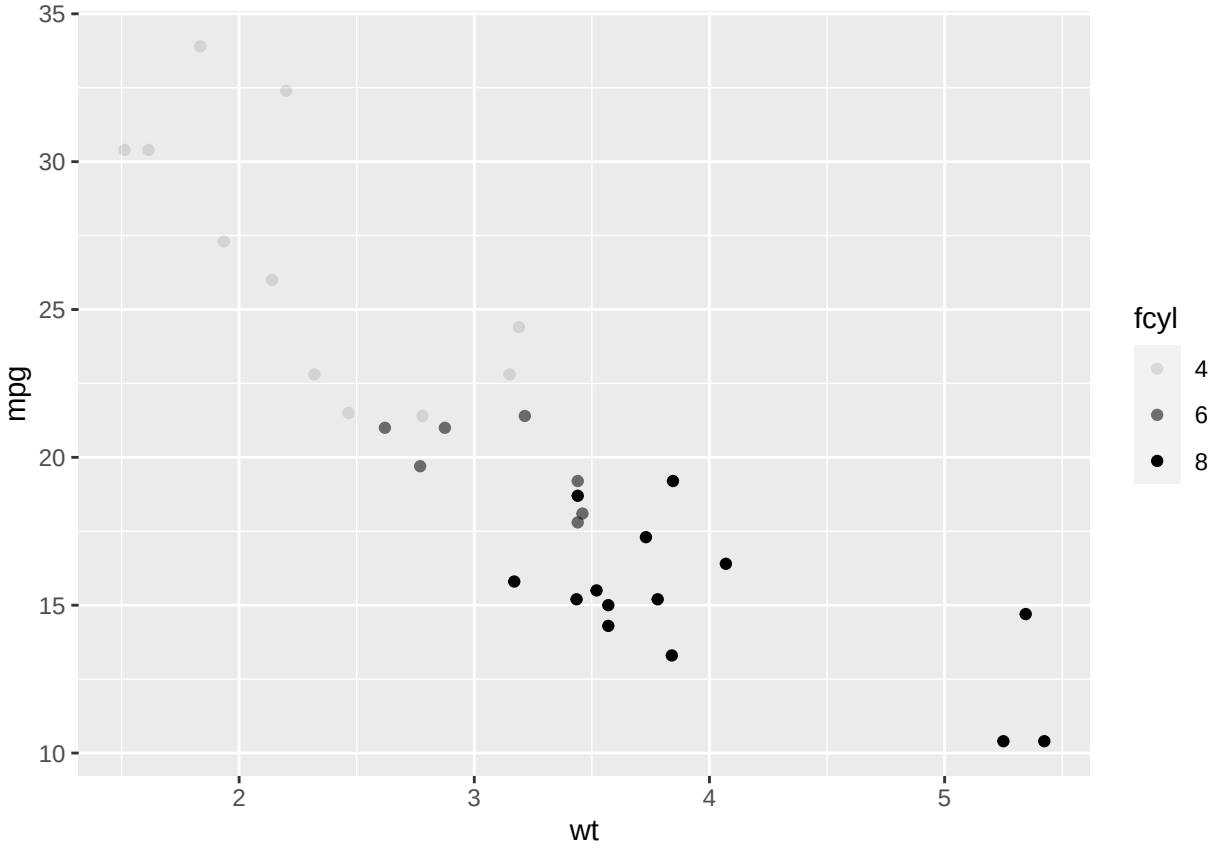
```
# Establish the base layer
plt_mpg_vs_wt <- ggplot(mtcars, aes(wt, mpg))

# Map fcyl to size
plt_mpg_vs_wt +
  geom_point(aes(size=fcyl))
```

```
## Warning: Using size for a discrete variable is not advised.
```

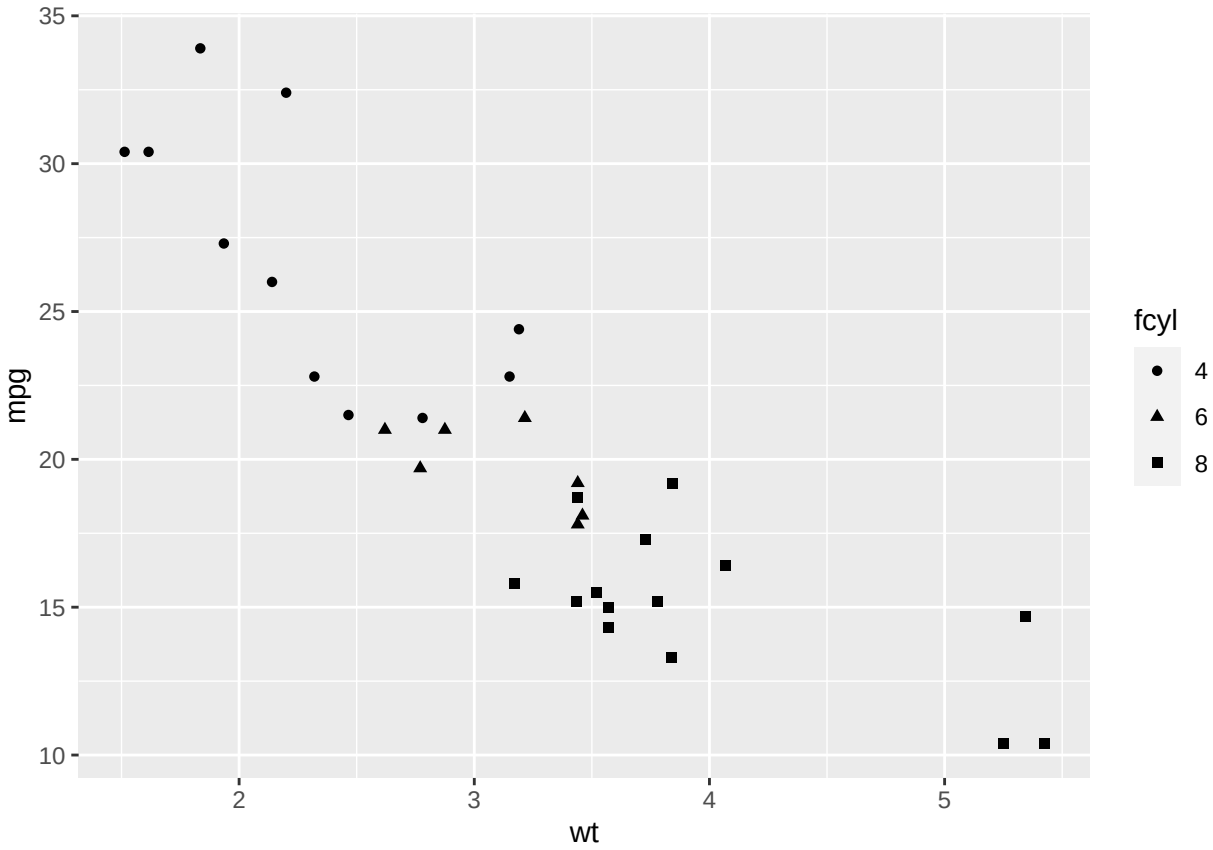



```
# Map fcyl to alpha, not size
plt_mpg_vs_wt +
  geom_point(aes(alpha = fcyl))
```

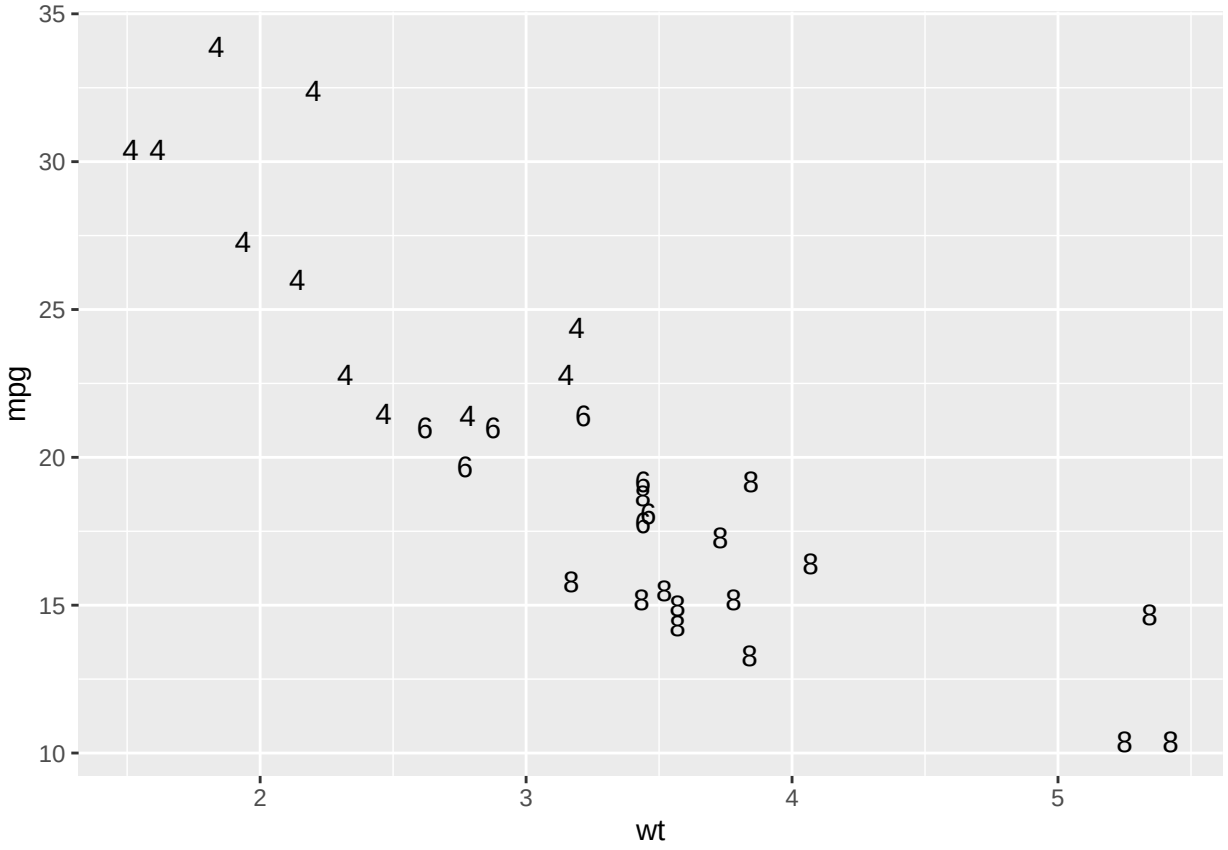
```
## Warning: Using alpha for a discrete variable is not advised.
```




```
# Map fcyl to shape, not alpha
plt_mpg_vs_wt +
  geom_point(aes(shape = fcyl))
```




```
# Use text layer and map fcyl to label
plt_mpg_vs_wt +
  geom_text(aes(label = fcyl))
```



5. Aesthetics for categorical & continuous variables

```
Which aesthetics are only applicable to categorical data?
A. color & fill
B. alpha & size
-> C. label & shape
D. alpha & label
E. x & y
```

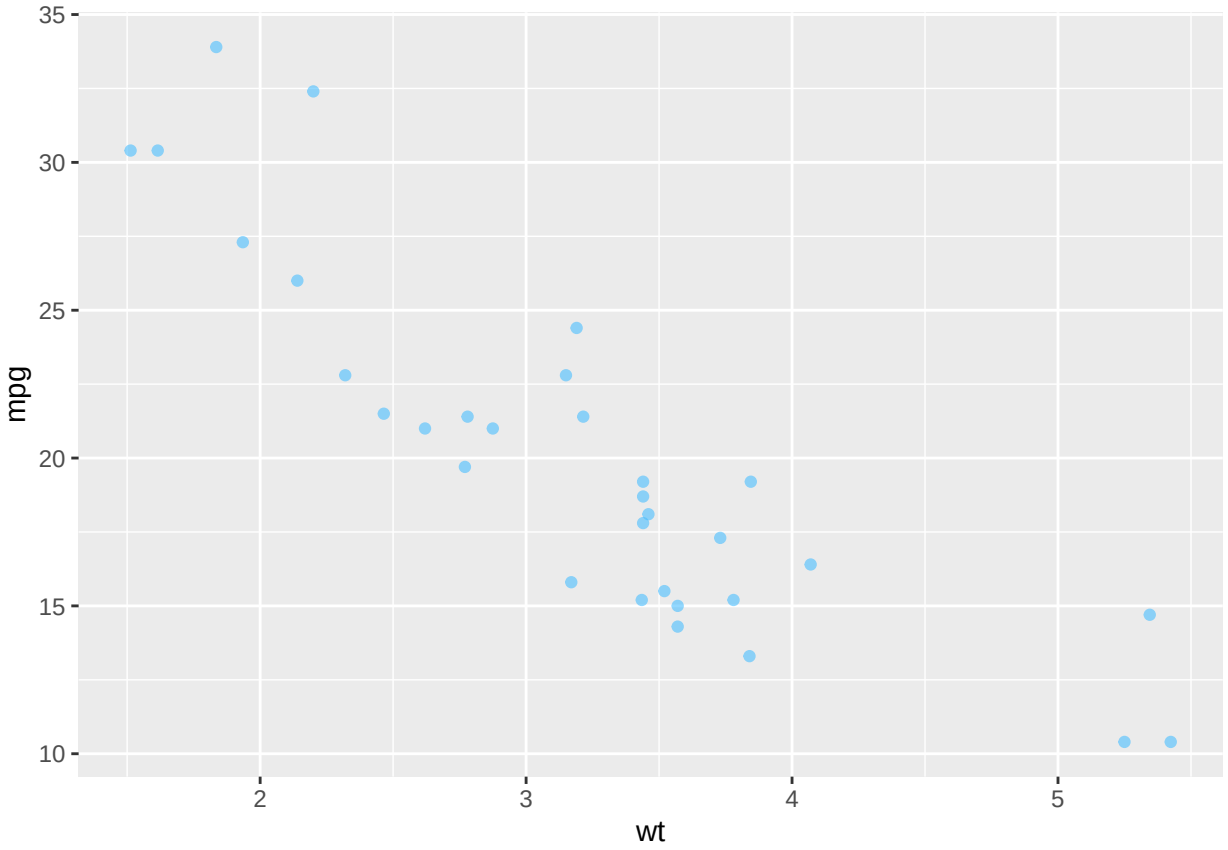
6. Using attributes

```
1. Using attributes
In the last exercises you learned a fundamental concept of ggplot2: aesthetic mappings. Colloquially, wh

2. Aesthetics? Attributes!
so it's easy to mix up the two! Attributes are always called in the geom layer (which we'll discuss in n

3. Aesthetics? Attributes!
For example, it's color attribute is set by the color argument, its size by the size argument

4. Aesthetics? Attributes!
and its shape by the shape argument. The distinction between aesthetics and attributes is subtle but imp
```

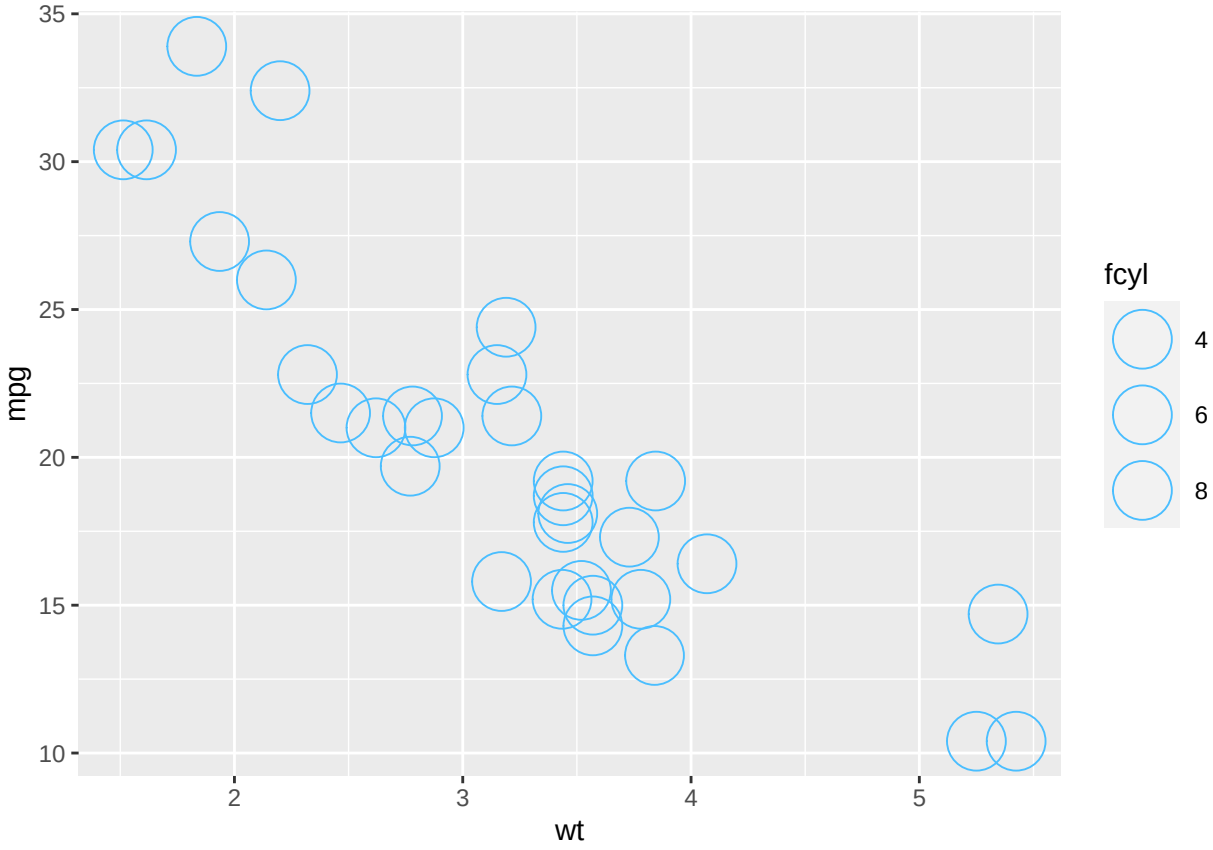
7. All about attributes: color, shape, size and alpha

```
# A hexadecimal color
my_blue <- "#4ABEFF"
```

```
ggplot(mtcars, aes(wt, mpg)) +
  # Set the point color and alpha
  geom_point(color=my_blue, alpha=.6)
```

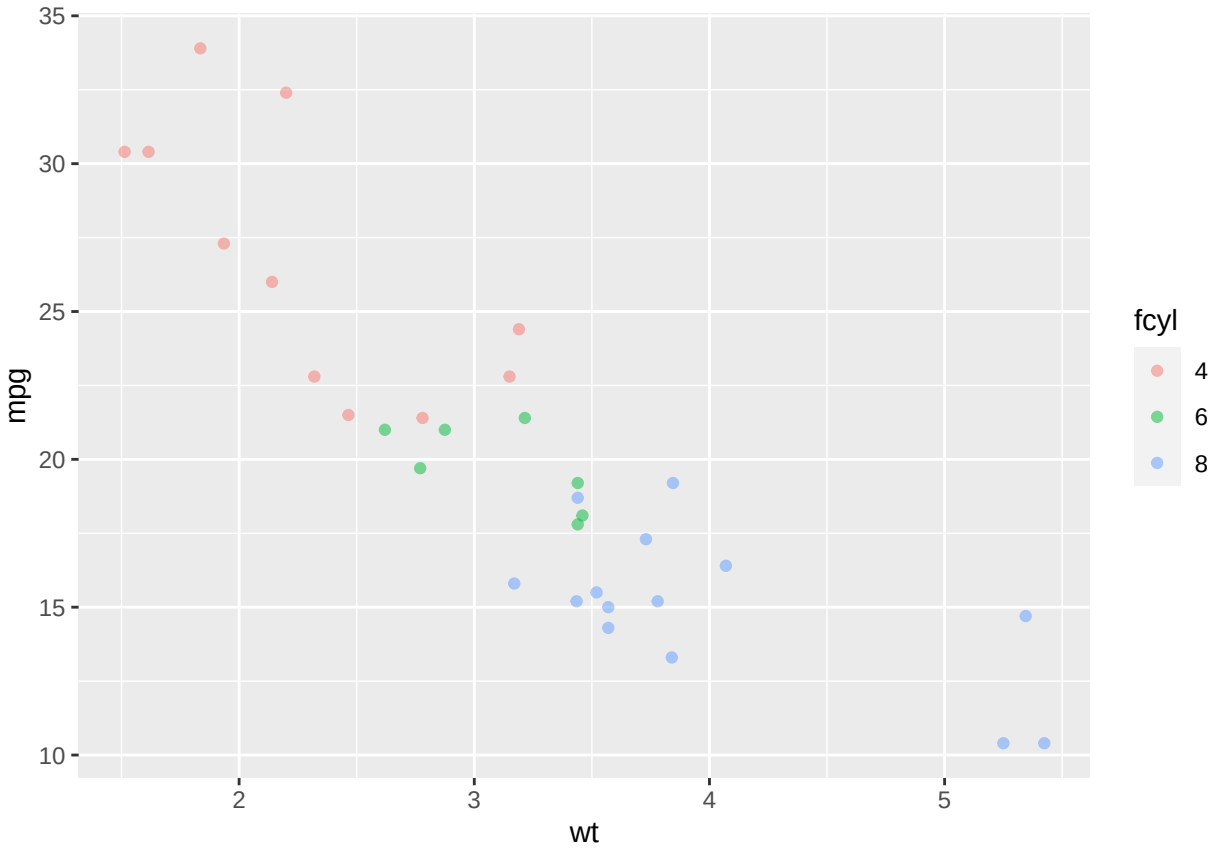



```
# Change the color mapping to a fill mapping
ggplot(mtcars, aes(wt, mpg, fill = fcyl)) +
  # Set point size and shape
  geom_point(color = my_blue, size = 10, shape = 1)
```

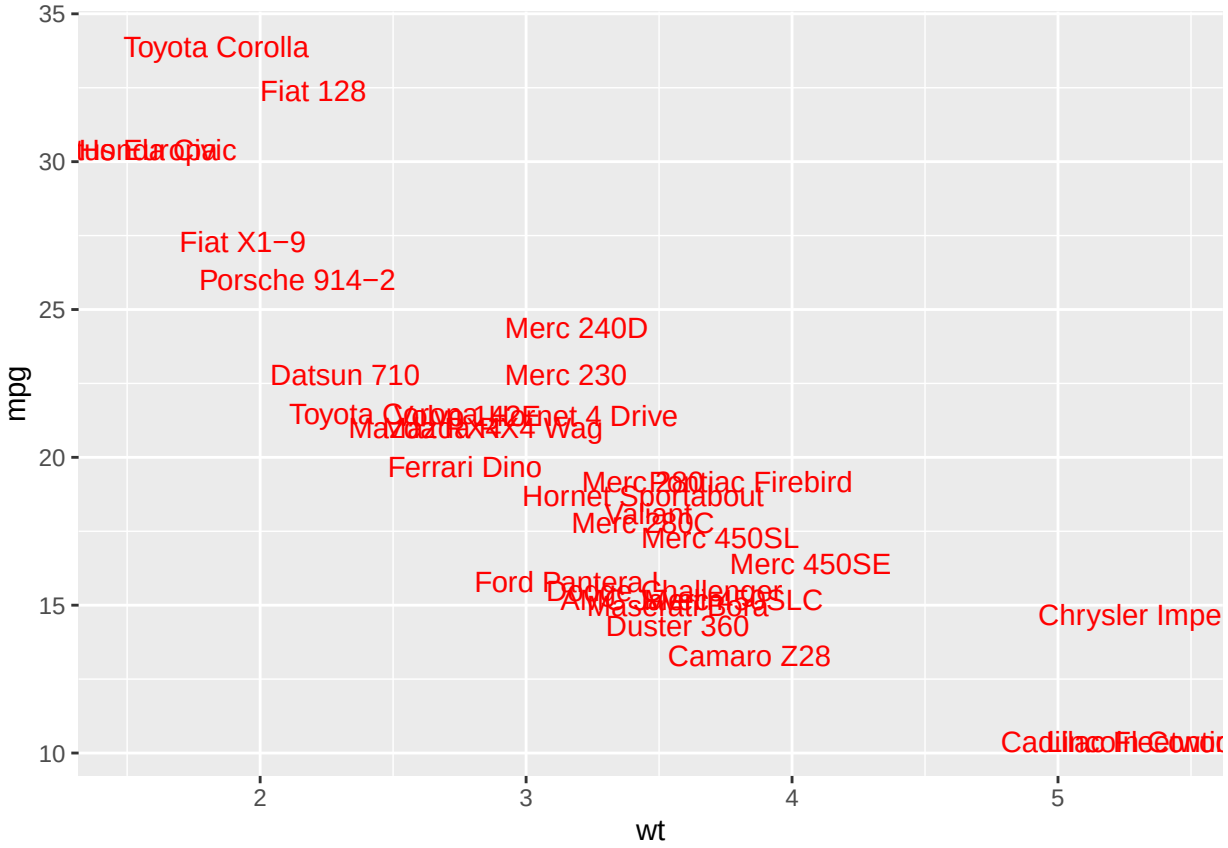


8. All about attributes: conflicts with aesthetics

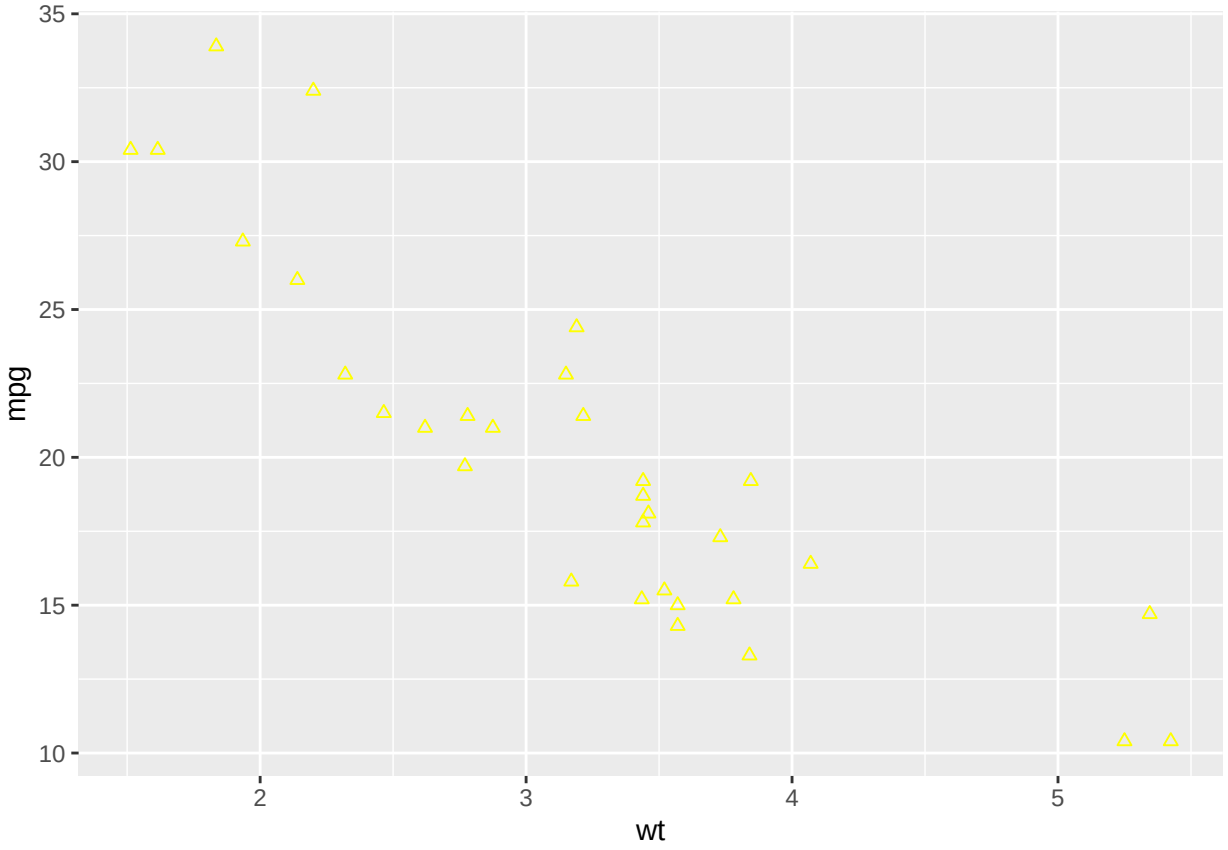
```
ggplot(mtcars, aes(wt, mpg, color = fcyl)) +
  # Add point layer with alpha 0.5
  geom_point(alpha=.5)
```




```
ggplot(mtcars, aes(wt, mpg, color = fcyl)) +
  # Add text layer with label rownames(mtcars) and color red
  geom_text(label=rownames(mtcars), color="red")
```

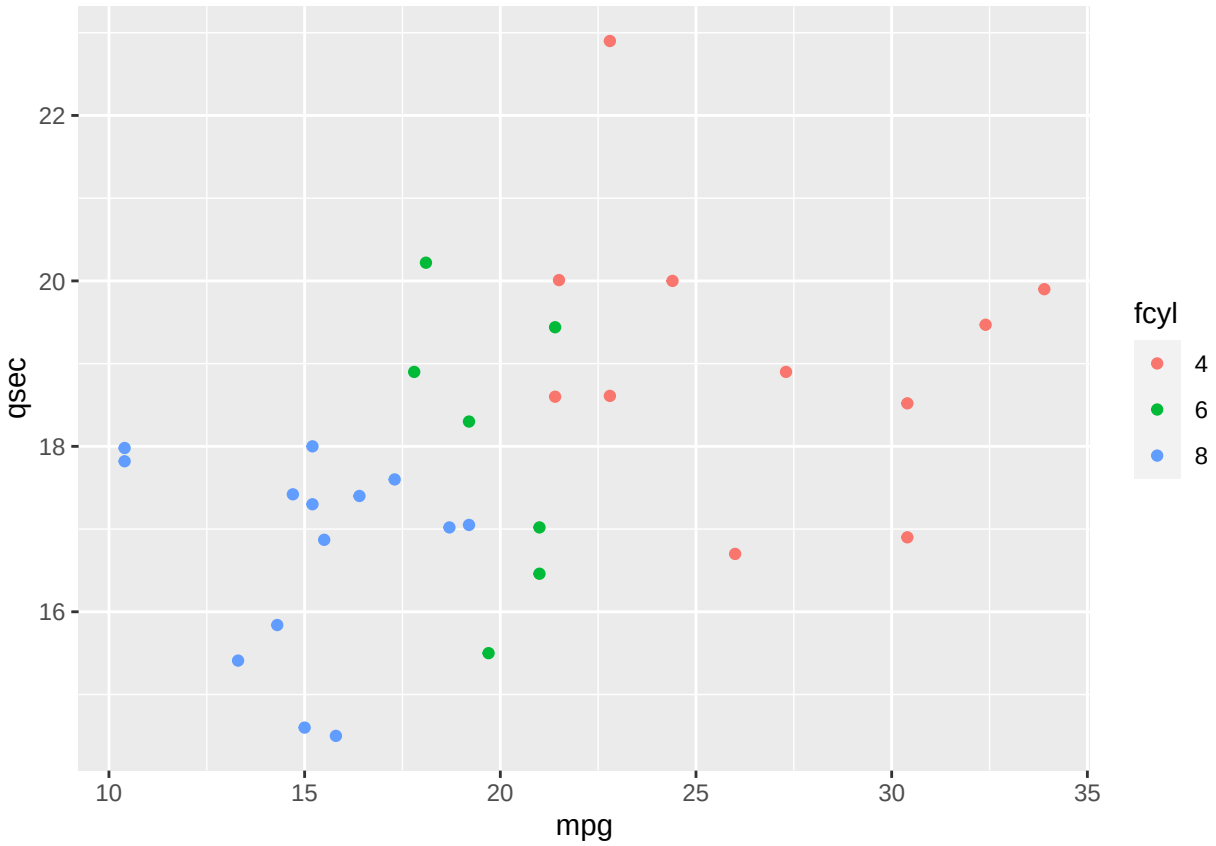



```
ggplot(mtcars, aes(wt, mpg, color = fcyl)) +
  # Add points layer with shape 24 and color yellow
  geom_point(shape=24, color="yellow")
```

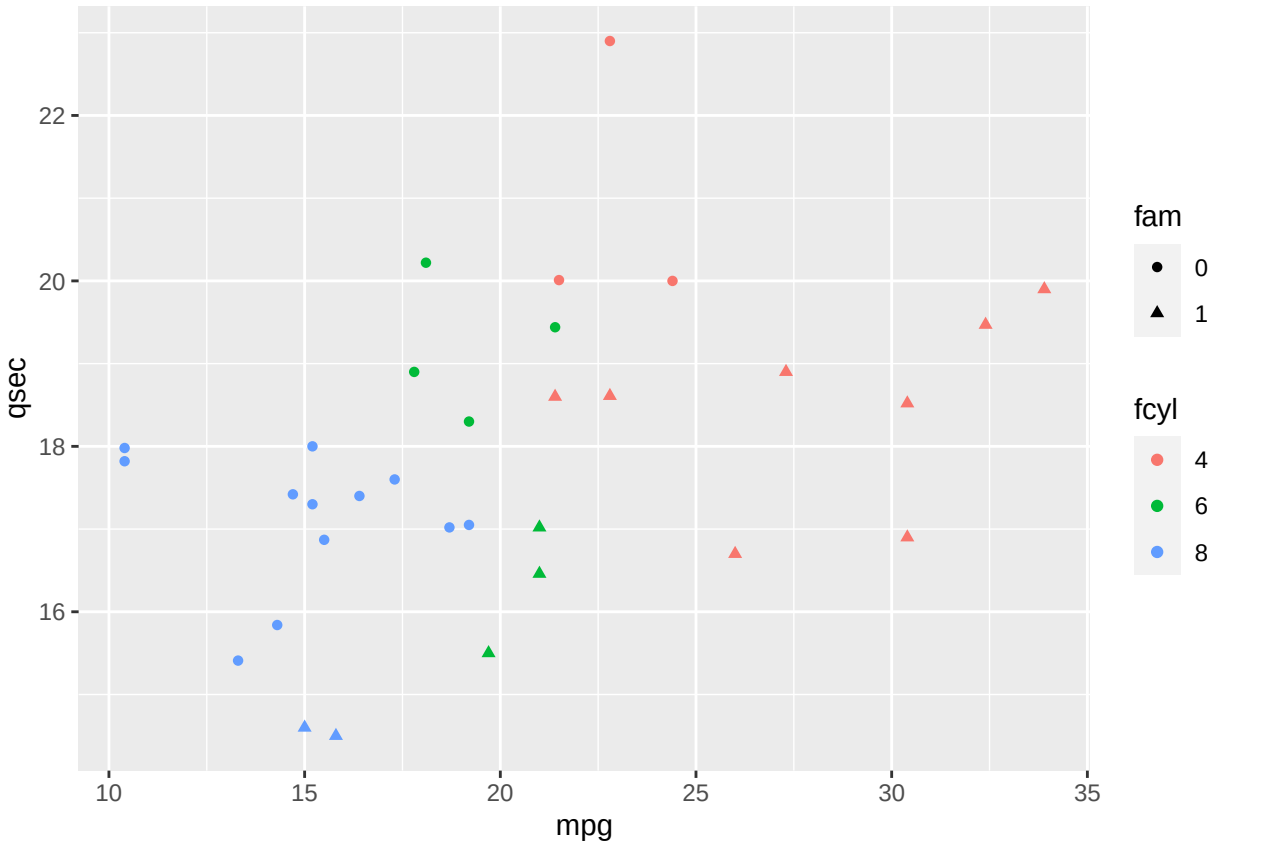


9. Going all out

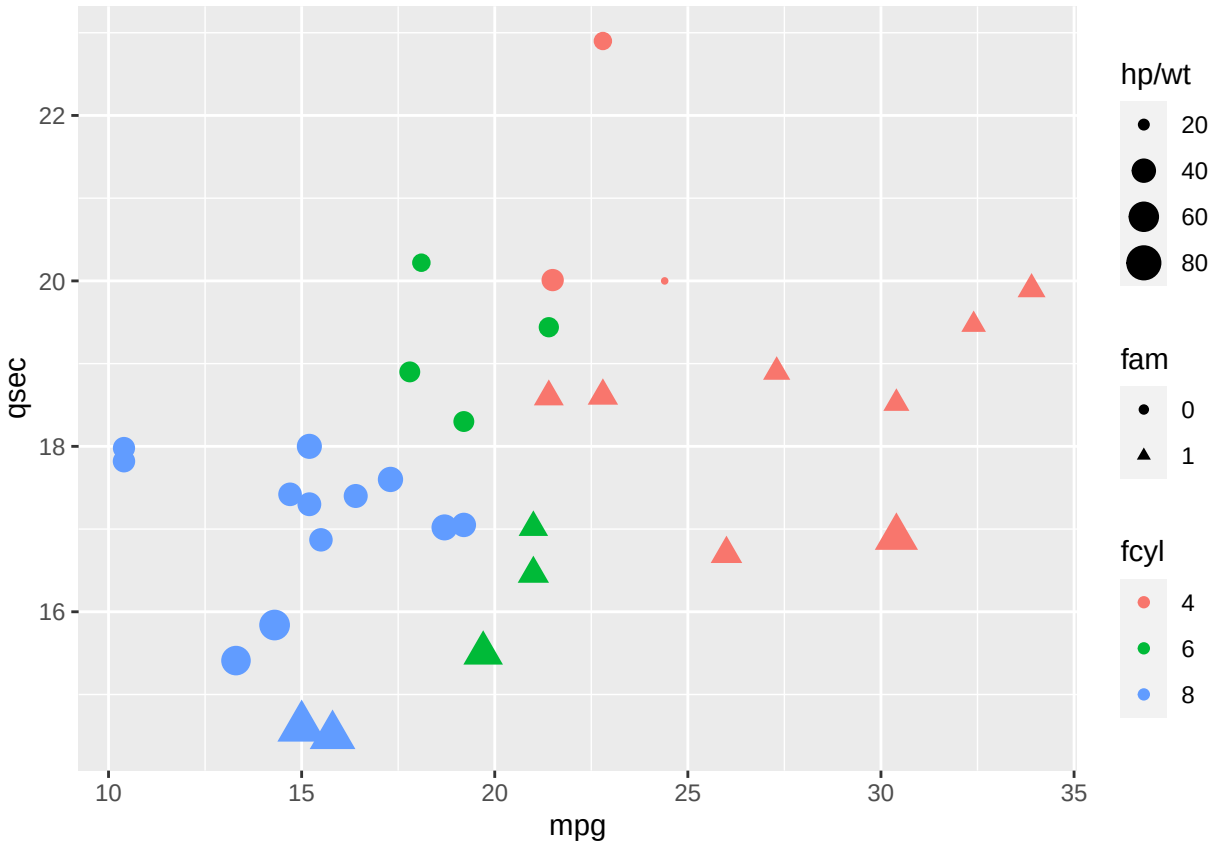
```
# 3 aesthetics: qsec vs. mpg, colored by fcyl
ggplot(mtcars, aes(mpg,qsec, color=fcyl)) +
  geom_point()
```




```
# 4 aesthetics: add a mapping of shape to fam
ggplot(mtcars, aes(mpg, qsec, color = fcyl, shape=fam)) +
  geom_point()
```




```
# 5 aesthetics: add a mapping of size to hp / wt
ggplot(mtcars, aes(mpg, qsec, color = fcyl, shape = fam, size=hp/wt)) +
  geom_point()
```



10. Modifying aesthetics

```
1. Modifying Aesthetics
Now that we know what aesthetics are and have some idea about choosing them appropriately, let's explor

2. Positions
A common adjustment is the position. Position specifies how ggplot will adjust for overlapping bars or

3. position = "identity" (default)
The most straightforward position is identity, which we've actually already seen. It's the default posi

4. position = "identity" (default)
We could have written it explicitly, but it's not necessary. There is an issue with the precision in th

5. position = "jitter"
"jitter" can be used as an argument, but each position type can also be accessed as a function. For exa

6. position_jitter()
position jitter can be defined in a function before we call our plot, as shown here. This has two advan

7. position_jitter()
Now we can set specific arguments for the position, such as the width, which defines how much random no

8. Scale functions
Recall that each of the aesthetics is a scale which we mapped data onto, so color is just a scale, like

9. Scale functions
```

That means we have to choose our axis dependent on the type of data we have. Here, we'll consider the co

10. scale_*_*()
There are many arguments for the scale functions. The first argument is always the name of the scale, a

11. The limits argument
limits describe the scale's range.

12. The breaks argument
breaks control the tick mark positions.

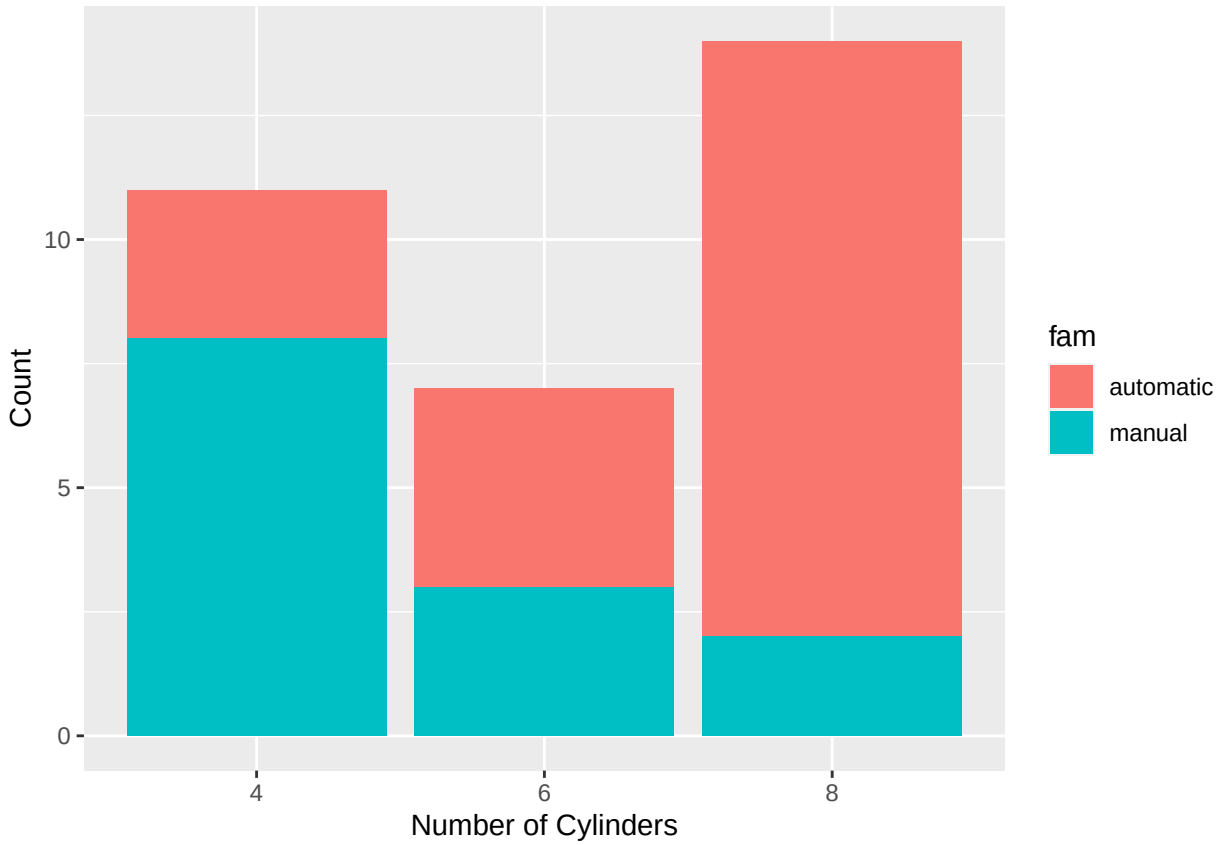
13. The expand argument
expand is a numeric vector of length two, giving a multiplicative and additive constant used to expand

14. The labels argument
and labels adjust the category names.

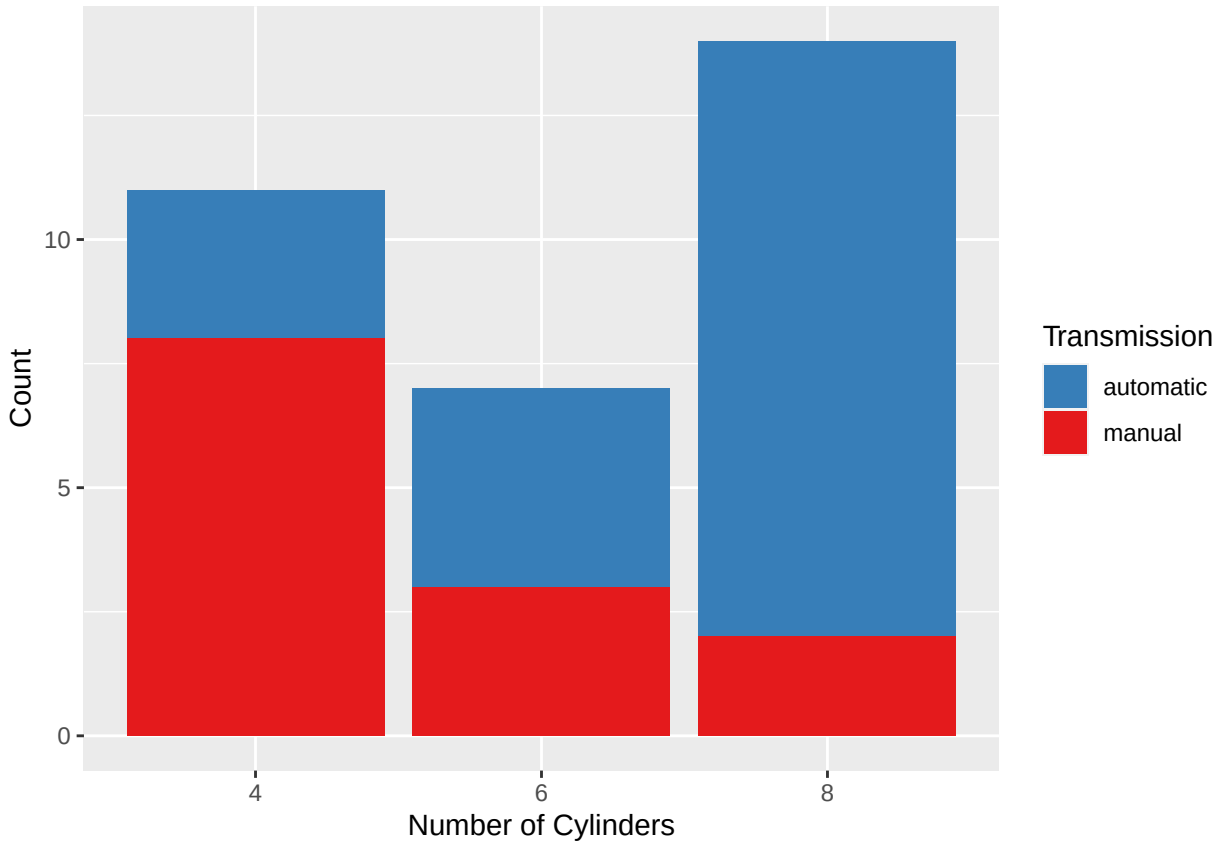
15. labs()
Note that if we just want to quickly change the axis labels, we can do this with the labs function.

11. Updating aesthetic labels

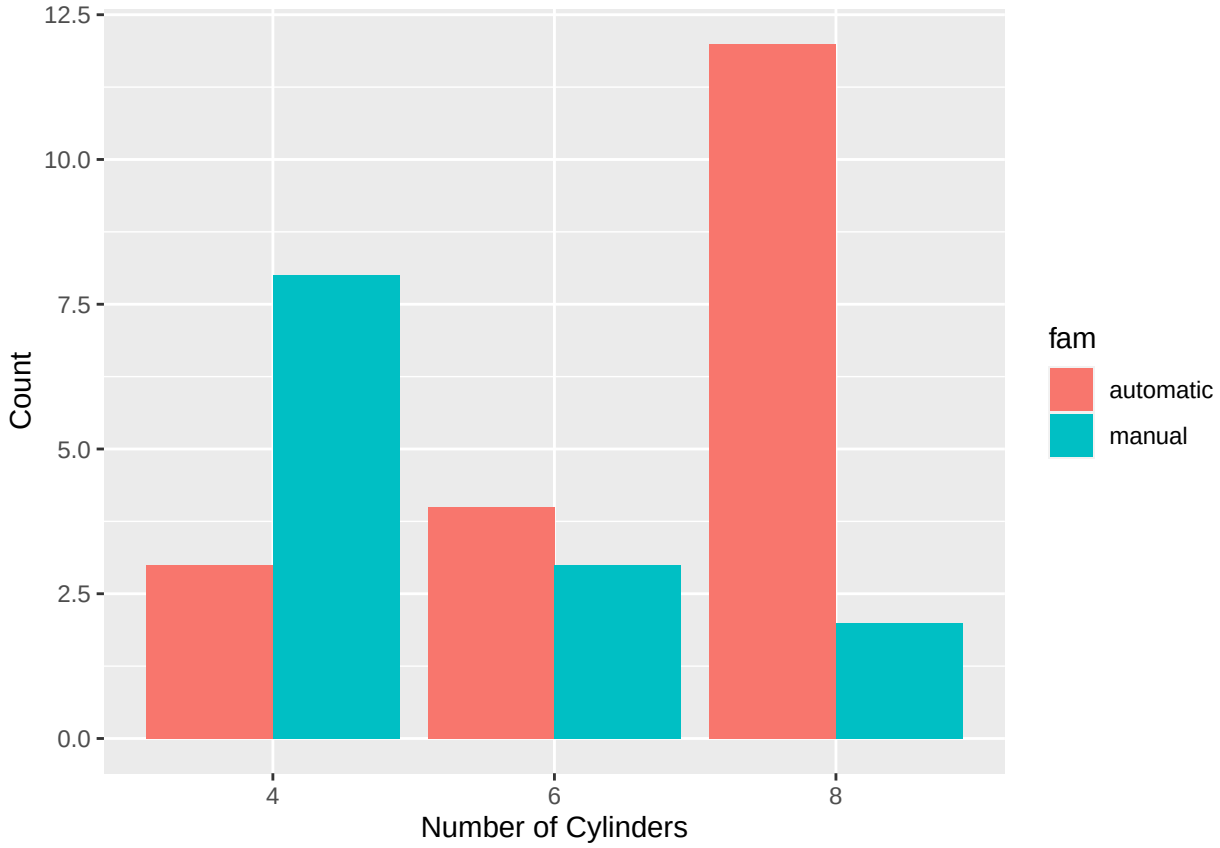
```
# use levels to assign & access attribute of fam.
levels(mtcars$fam) <-  c("automatic", "manual")

library(ggplot2)
ggplot(mtcars, aes(fcyl, fill = fam)) +
  geom_bar() +
  # Set the axis labels
  labs(
    x="Number of Cylinders",
    y="Count"
  )
```




```
palette <- c(automatic = "#377EB8", manual = "#E41A1C")

ggplot(mtcars, aes(fcyl, fill = fam)) +
  geom_bar() +
  labs(x = "Number of Cylinders", y = "Count") +
  # Set the fill color scale
  scale_fill_manual("Transmission", values = palette)
```




```
# Set the position
ggplot(mtcars, aes(fcyl, fill = fam)) +
  geom_bar(position="dodge") +
  labs(x = "Number of Cylinders", y = "Count")
```



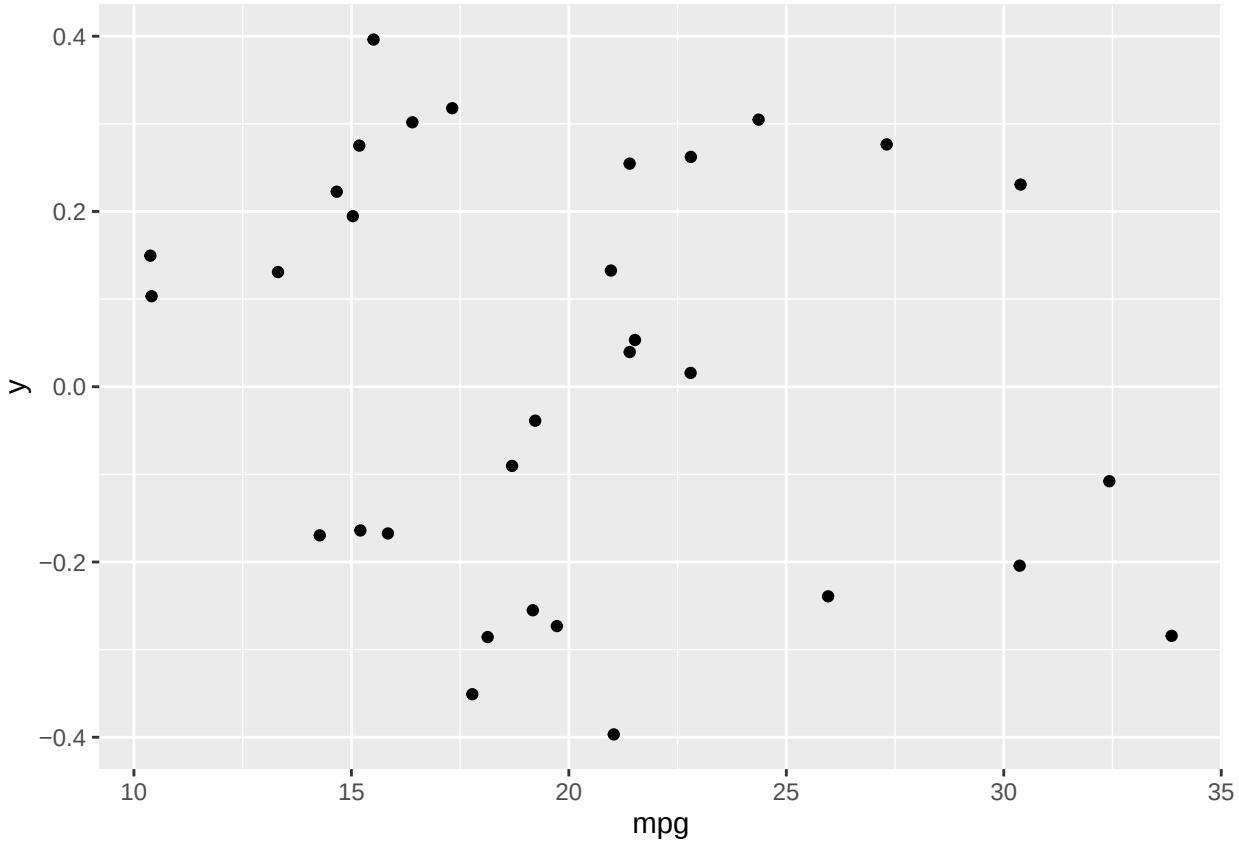

```
  scale_fill_manual("Transmission", values = palette)
```

```
## <ggproto object: Class ScaleDiscrete, Scale, gg>
##     aesthetics: fill
##     axis_order: function
##     break_info: function
##     break_positions: function
##     breaks: waiver
##     call: call
##     clone: function
##     dimension: function
##     drop: TRUE
##     expand: waiver
##     get_breaks: function
##     get_breaks_minor: function
##     get_labels: function
##     get_limits: function
##     guide: legend
##     is_discrete: function
##     is_empty: function
##     labels: waiver
##     limits: function
##     make_sec_title: function
##     make_title: function
##     map: function
##     map_df: function
##     n.breaks.cache: NULL
```

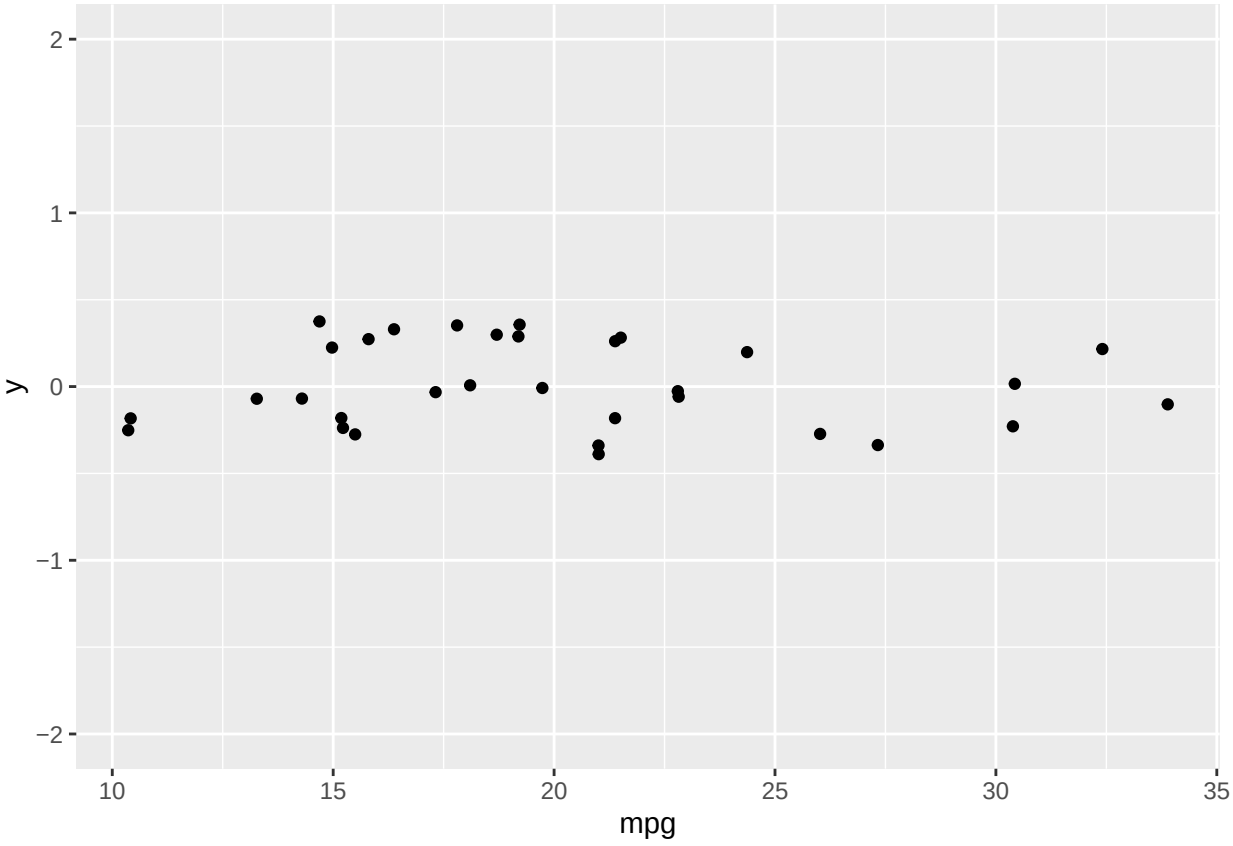
```
##     na.translate: TRUE
##     na.value: grey50
##     name: Transmission
##     palette: function
##     palette.cache: NULL
##     position: left
##     range: <ggproto object: Class RangeDiscrete, Range, gg>
##         range: NULL
##         reset: function
##         train: function
##         super:  <ggproto object: Class RangeDiscrete, Range, gg>
##     rescale: function
##     reset: function
##     scale_name: manual
##     train: function
##     train_df: function
##     transform: function
##     transform_df: function
##     super:  <ggproto object: Class ScaleDiscrete, Scale, gg>
```

12. Setting a dummy aesthetic

```
# Plot 0 vs. mpg
ggplot(mtcars, aes(mpg, 0)) +
  # Add jitter
  geom_point(position="jitter")
```



```r
ggplot(mtcars, aes(mpg, 0)) +
  geom_jitter() +
  # Set the y-axis limits
  ylim(c(-2,2))
```



13. Aesthetics best practices

1. Aesthetics best practices
Now that we know what visual aesthetics are, how do we choose the right one?

2. Which aesthetics?
There is some creativity involved, but there are some helpful guidelines. This chapter is informed by tl

3. Form follows function
The best data viz serves a purpose - that is, form follows function. So what is the function in data viz

4. Form follows function
First and foremost, our function is the accurate and efficient representation of data. Beautiful is nice

5. Calculating statistics
Let's look at a simple example. In this data set, I have two continuous variables, x & y, y is a functio

6. Calculating statistics
It's pretty difficult to obtain summary statistics just by looking at the data.

7. Extracting information from Data
So we have two choices, numeric summaries, which are precise but offer a poor overview, or data visualiz

8. Encoding numbers into plots
To make a plot, we encode data in numbers and text into a visual medium. That's what we do with aesthet

9. Various aesthetic mappings
These plots differ in their aesthetic mappings and other values that we'll explore throughout the cours

10. Decoding to data
These visuals are then decoded to form an image of the original data. It's inherently imprecise - kind

11. The best choices for aesthetics
We consider the best choices to be those which are both efficient, in that they are faster than numeric

12. Aesthetics - continuous variables
The choice of aesthetic mapping depends on the type of variable. The scatter plot is so easy to underst

13. Aesthetics - continuous variables
Imagine if we switch the aesthetic mappings for x and color. This is possible, but is neither accurate

14. Efficiency of decoding
There are many choices for mapping continuous variables. For example, position on unaligned scales, as

15. Three iris scatter plots
like this example where we had three plots from the iris wide2 data frame, one for each of our iris spe

16. Three iris scatter plots, unaligned y-axes
on unaligned y axes, is less efficient and makes it difficult to compare plots compared to

17. Single faceted plot, common y-axis
an aligned scale.

18. Decoding categorical
There are also a variety of choices for categorical data.

19. Aesthetics - categorical variables
Color is often used to good effect for a categorical variables. But efficiency and accuracy are not onl

20. Aesthetics - categorical variables
We'll see lots of examples of over-plotting and how to deal with it in the exercises. Here, we'll want

14. Appropriate mappings

Typically, the dependent variable is mapped onto the the y-axis and the independent variable is mapped
In the ToothGrowth data set, we have three variables:
len: Tooth length
supp: Supplement type (VC or OJ)
dose: Dose in milligrams/day

From the six possible ways to map three variables, one solution is shown in the viewer. Which of the op
-> x = supp, y = len, color = dose

Part 3. Geometrics

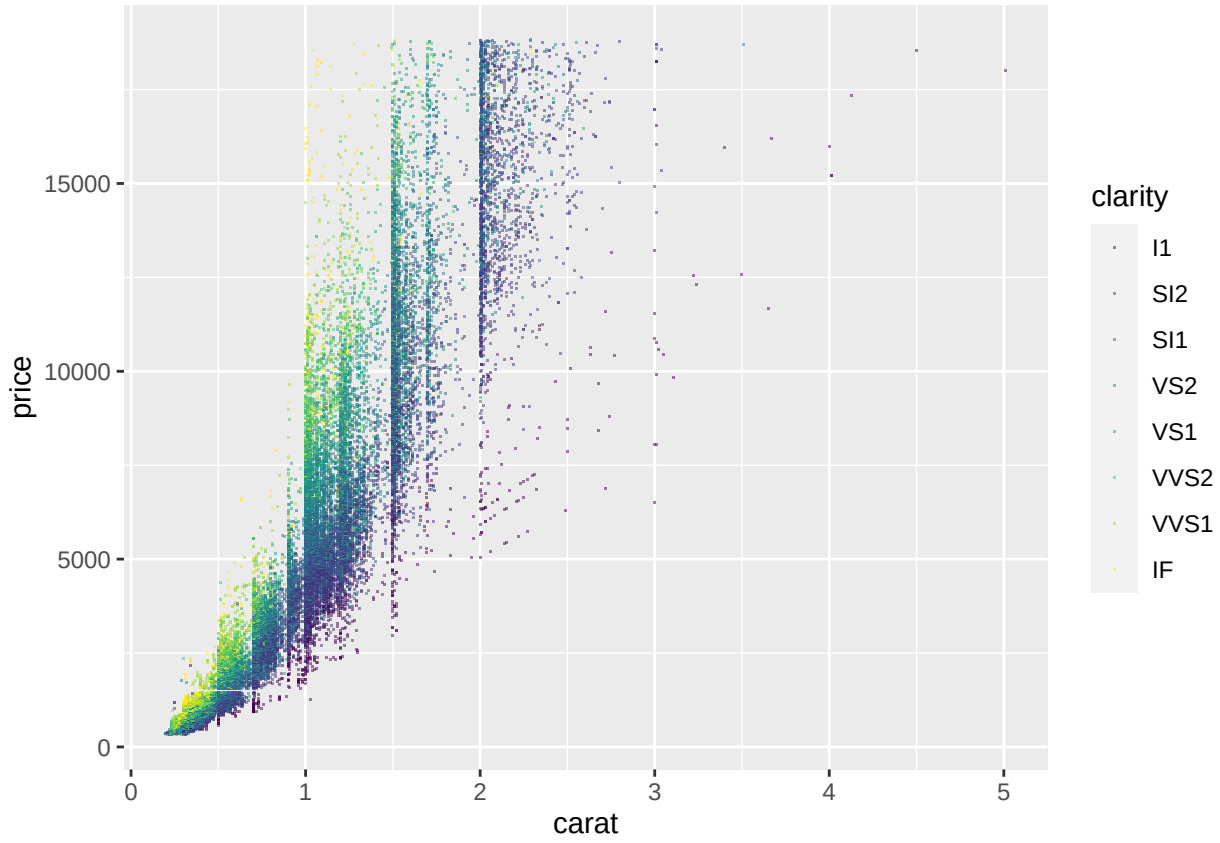
1. Scatter plots

```
1. Scatter plots
The third essential layer is the geometry layer. This determines how the plot actually looks. We've alr

2. 48 geometries
At present there are almost 50 different geometries to choose from, although there are some redundancie

3. Common plot types
Let's begin with scatter plots.

4. Scatter plots
Each geom is associated with specific aesthetic mappings, some of which are essential. To use geom_poin

5. Scatter plots
In addition to the essential aesthetics, we can also choose optional aesthetics, like alpha, color, fil

6. Geom-specific aesthetic mappings
We can specify both geom-specific data and aesthetics. This allows us to control the information for ea

7. iris demo
Imagine I have a data frame which contains summary statistics, such as the mean, for each of my variabl

8. iris plot
In this plot one geom_point layer inherits the data and aesthetics from the parent ggplot function, and

9. Shape attribute values
The possible values are shown here. 15 is a solid square. Numbers 21 - 25 are not simply repeats of ear

10. Example
For example, I can have a black fill and use a stroke of 2 for a thick outline. The color aesthetic is

11. On-the-fly stats by ggplot2
It's not fair to plot the mean without some measure of spread, like the standard deviation. We'll get i

12. position = "jitter"
Recall that in the last chapter we used the position argument to change the position from identity to j

13. geom_jitter()
We could have also done this with the geom_jitter function directly. geom_jitter is just a wrapper for

14. Don't forget to adjust alpha
On top of jittering, we would also need to deal with overplotting of points by adjusting the alpha-blen

15. Hollow circles also help
Yet another way to deal with overplotting is to change the symbol to a hollow circle, which is shape 1.
```

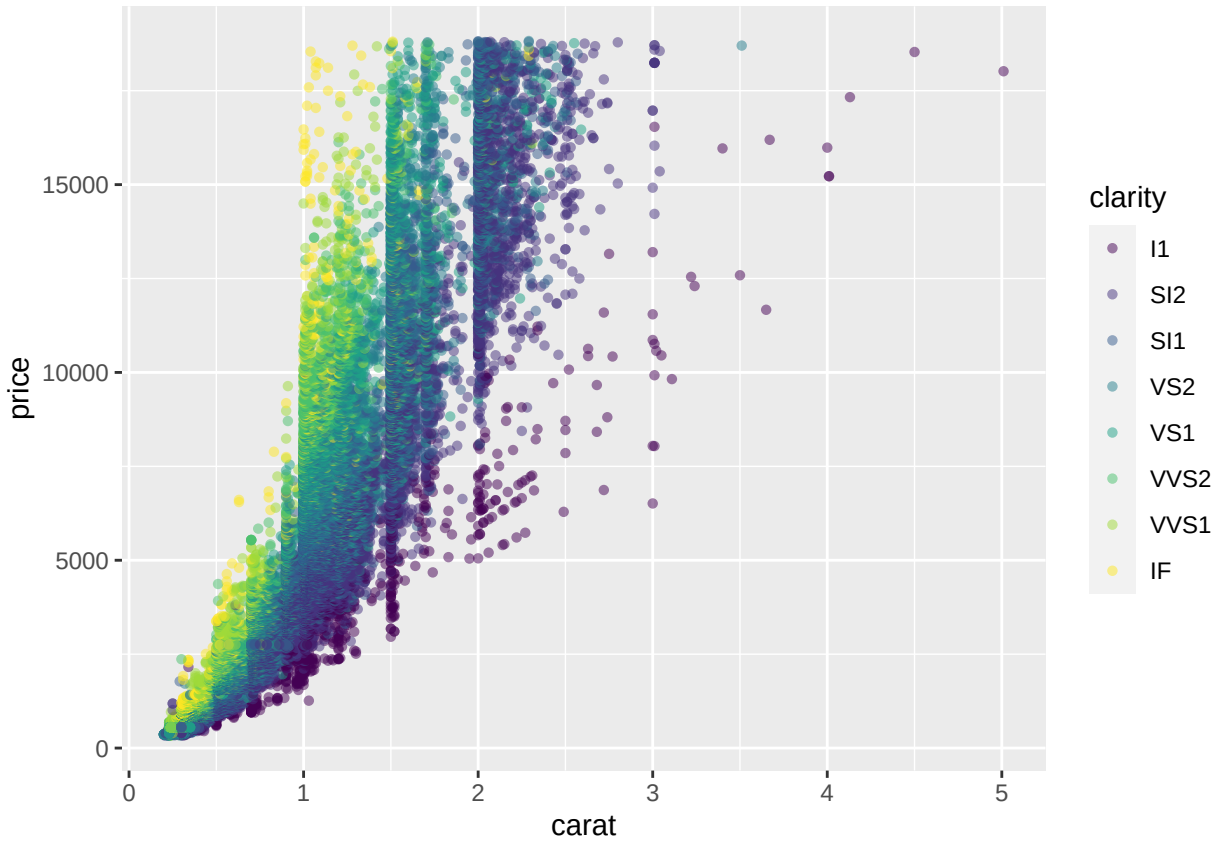
2. Overplotting 1: large datasets

```
# Plot price vs. carat, colored by clarity
plt_price_vs_carat_by_clarity <- ggplot(diamonds, aes(carat, price, color = clarity))

# Add a point layer with tiny points
```

```
plt_price_vs_carat_by_clarity +
    geom_point(alpha=.5, shape=".")
```

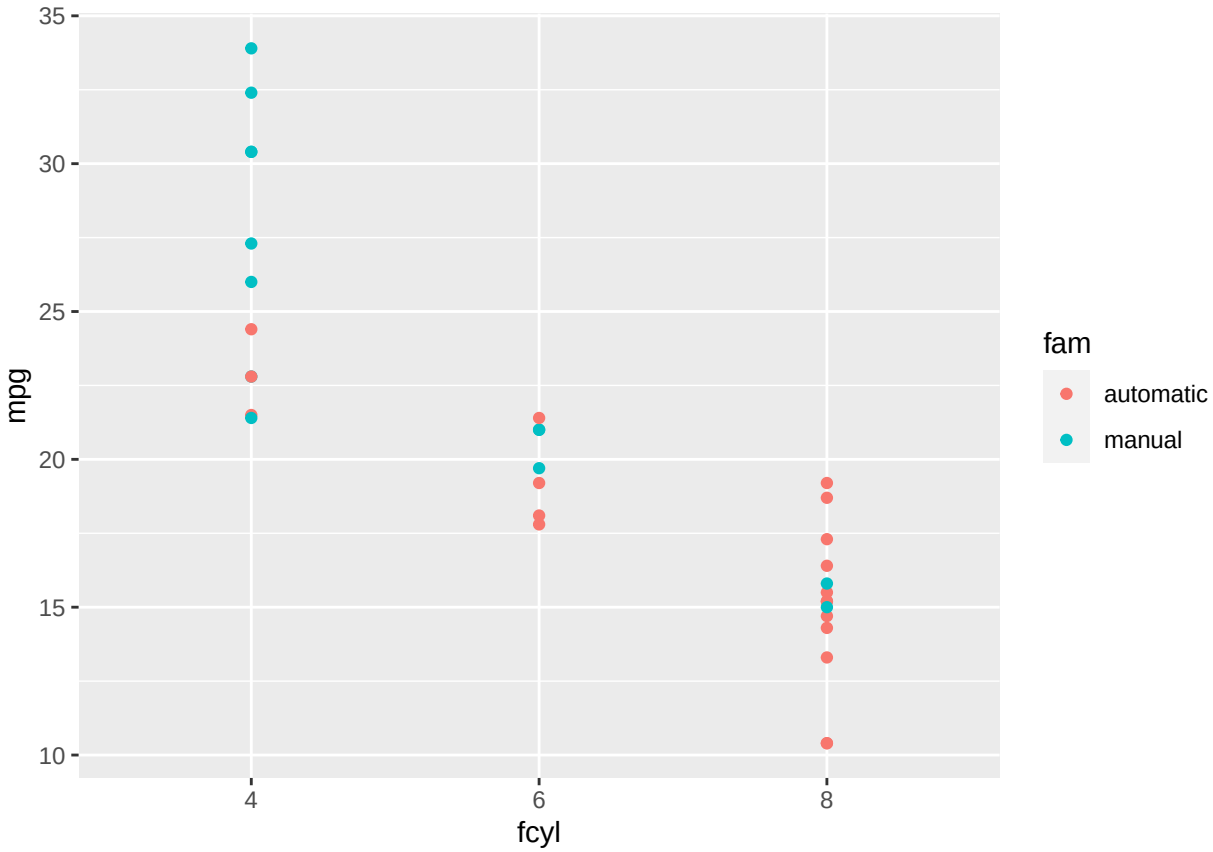



```
# Set transparency to 0.5, shape to 16
plt_price_vs_carat_by_clarity + geom_point(alpha = 0.5, shape = 16)
```

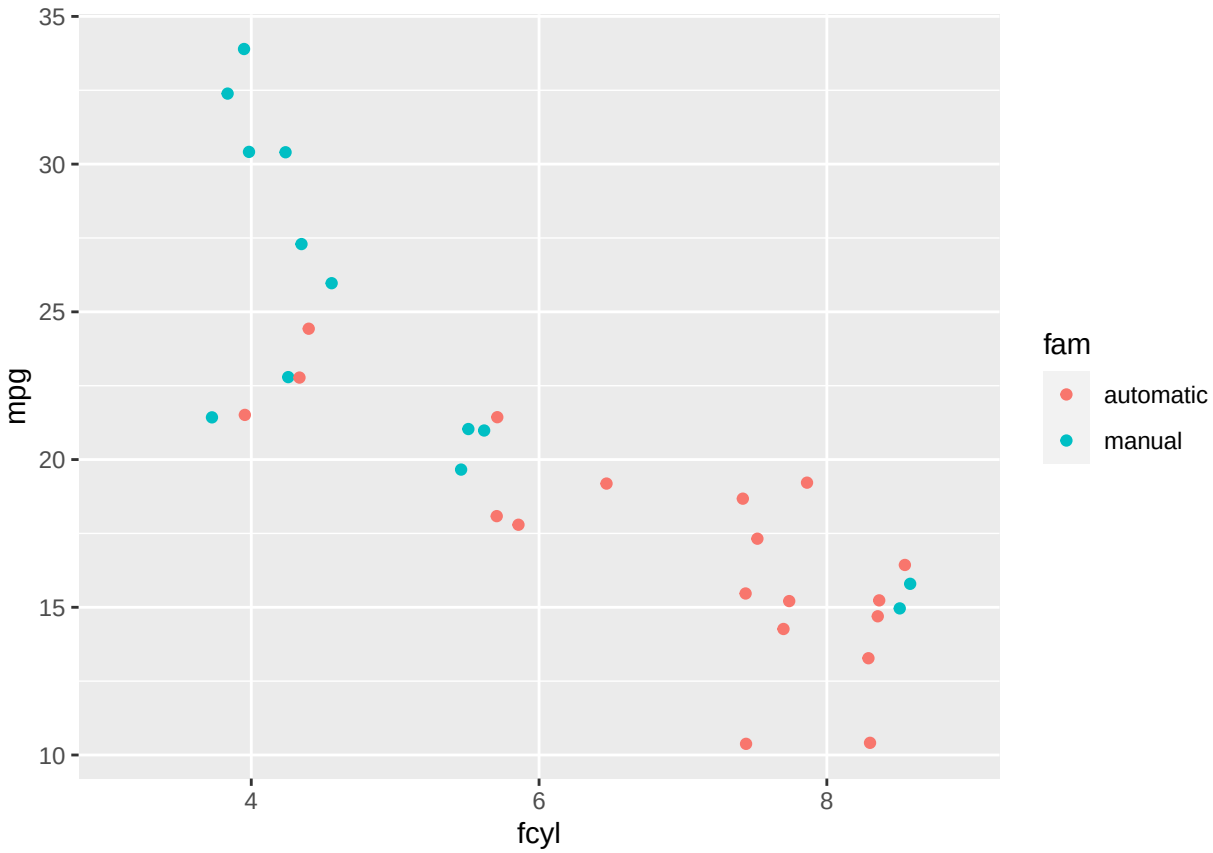


3. Overplotting 2: Aligned values

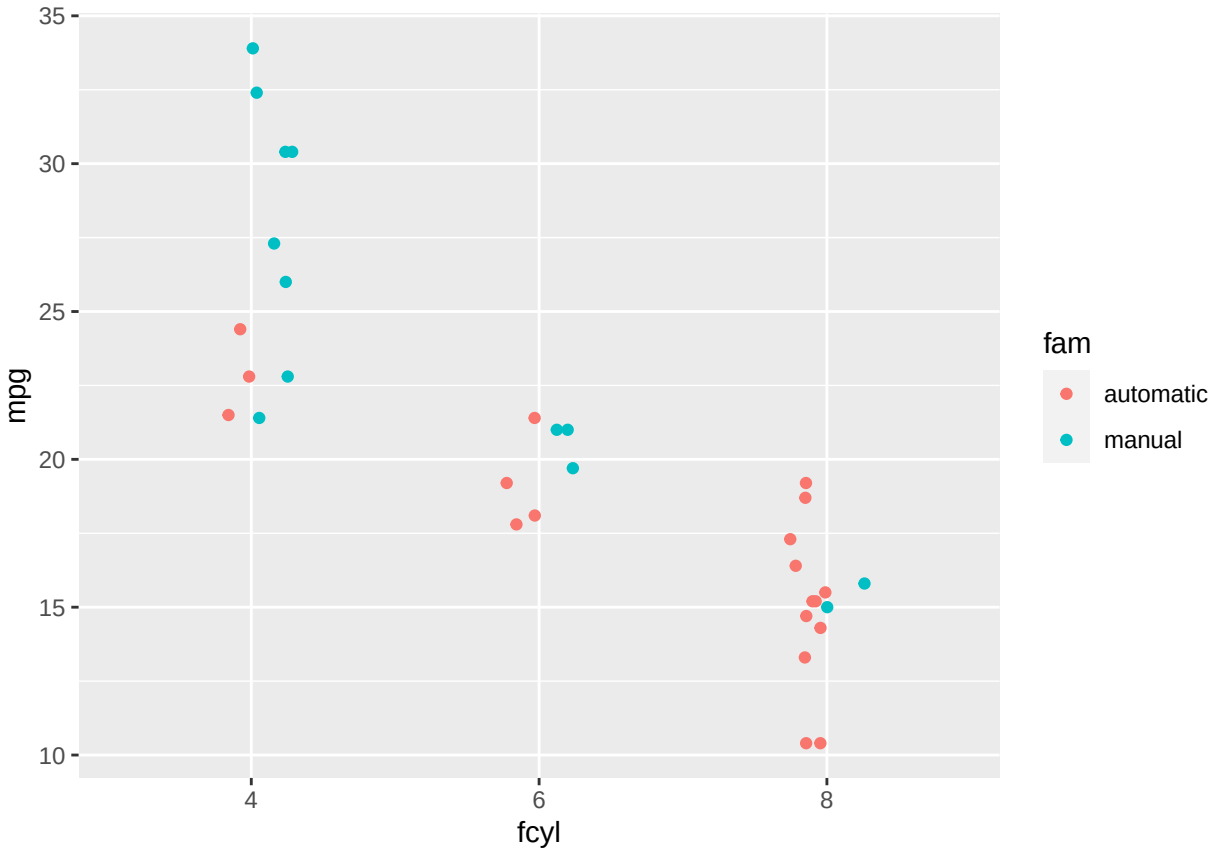
```
# Plot base
plt_mpg_vs_fcyl_by_fam <- ggplot(mtcars, aes(fcyl, mpg, color = fam))

# Default points are shown for comparison
plt_mpg_vs_fcyl_by_fam +
    geom_point()
```




```
# Alter the point positions by jittering, width 0.3
plt_mpg_vs_fcyl_by_fam +
  geom_point(
    position = position_jitter(0.3)
  )
```

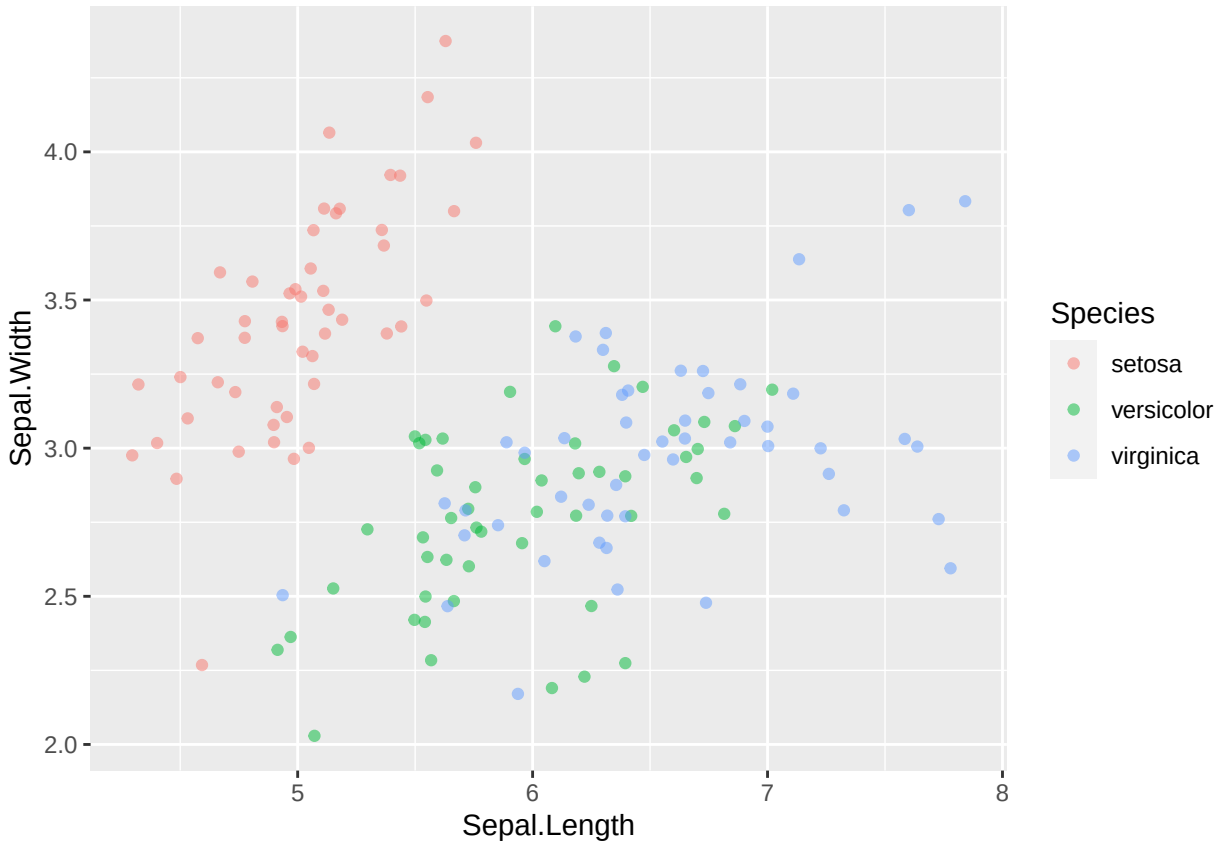



```
# Now jitter and dodge the point positions
plt_mpg_vs_fcyl_by_fam +
    geom_point(
        position = position_jitterdodge(jitter.width=0.3, dodge.width=0.3)
    )
```

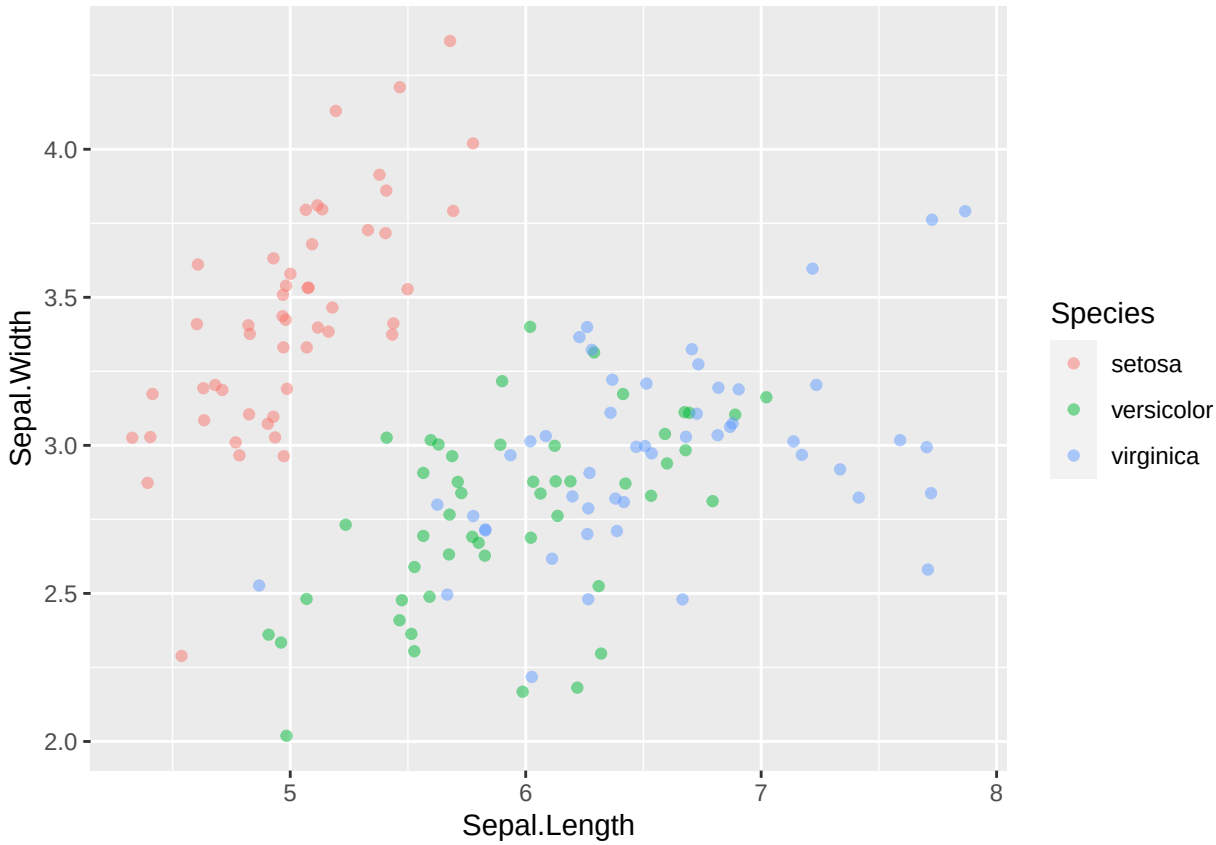


4. Overplotting 3: Low-precision data

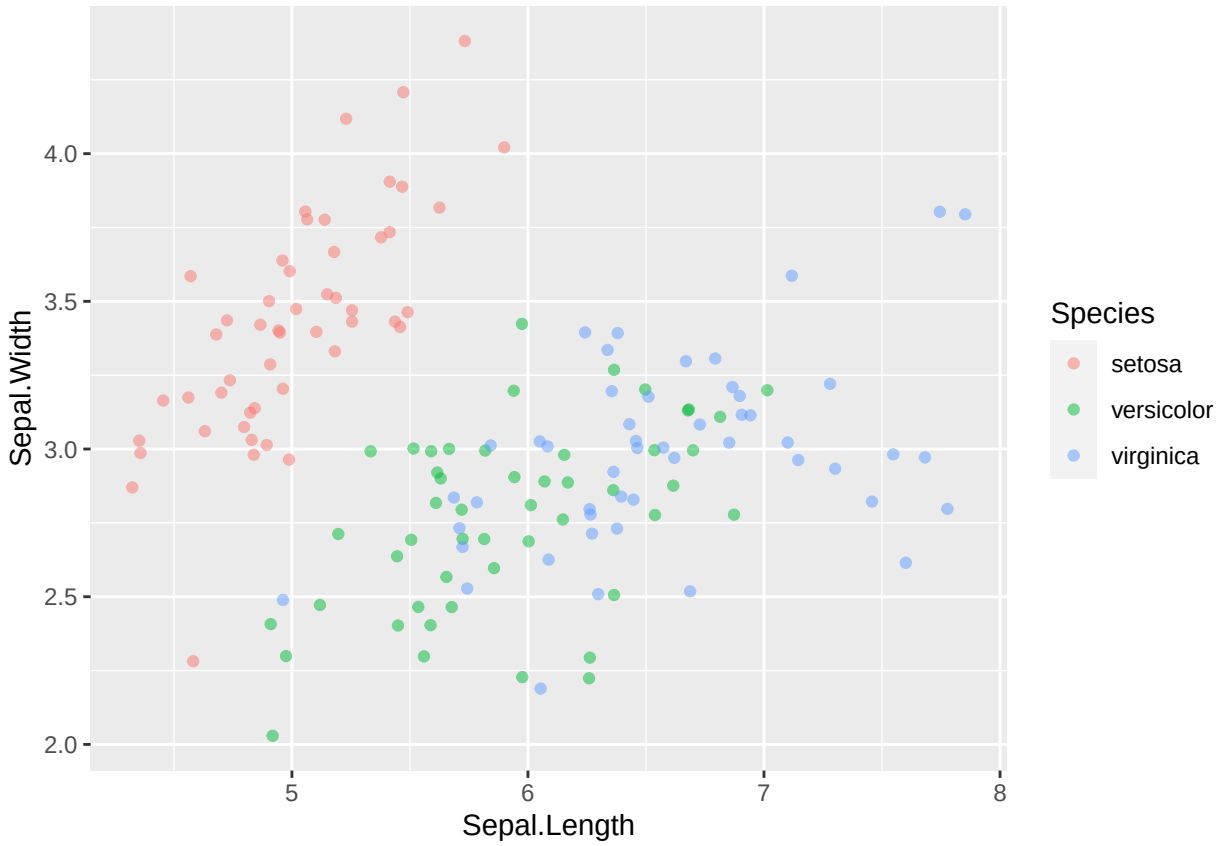
```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  # Swap for jitter layer with width 0.1
  geom_jitter(alpha = 0.5, width = 0.1)
```




```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  # Set the position to jitter
  geom_point(alpha = 0.5, position = "jitter")
```




```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  # Use a jitter position function with width 0.1
  geom_point(alpha = 0.5, position = position_jitter(0.1))
```



5. Overplotting 4: Integer data

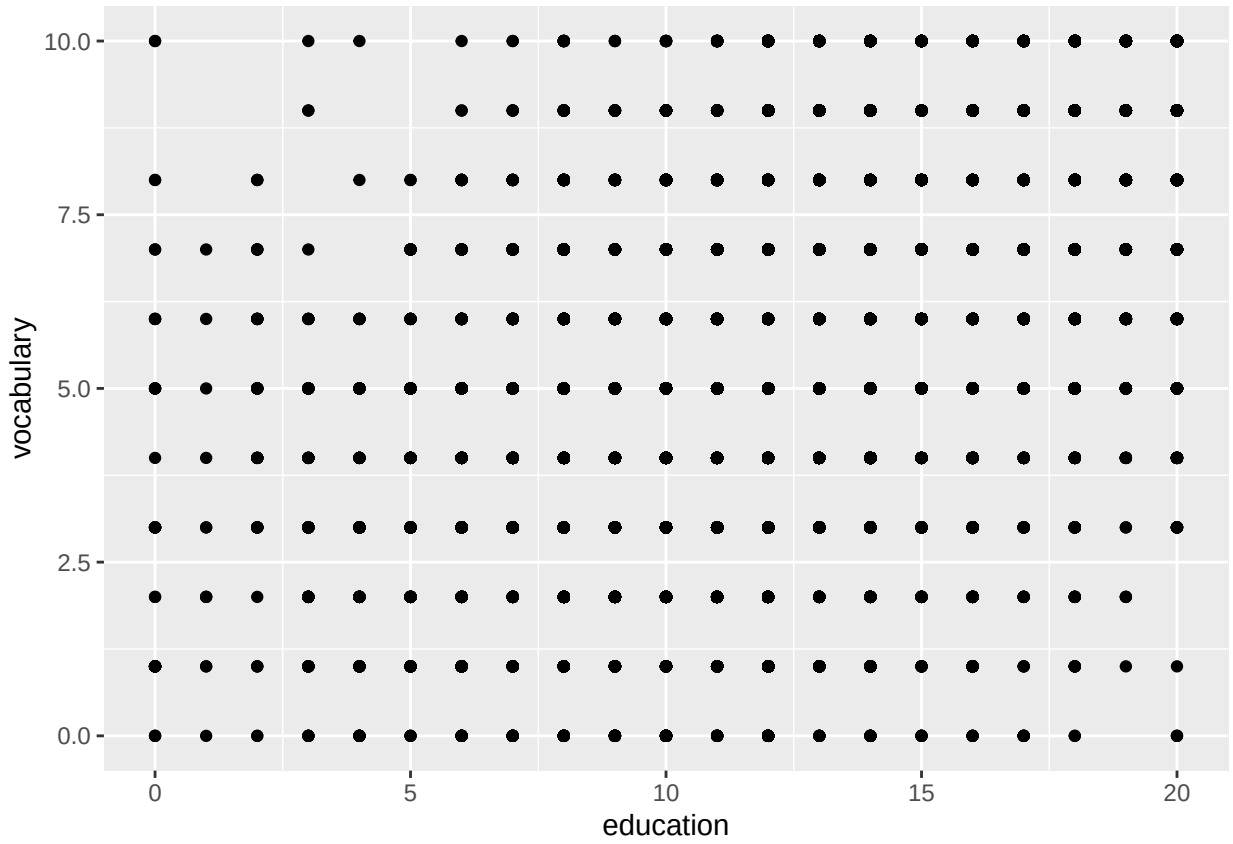
```
# load ggplot2
library(ggplot2)

# install car package @https://cran.r-project.org/web/packages/car/index.html
library(car)
```

```
## Loading required package: carData
```

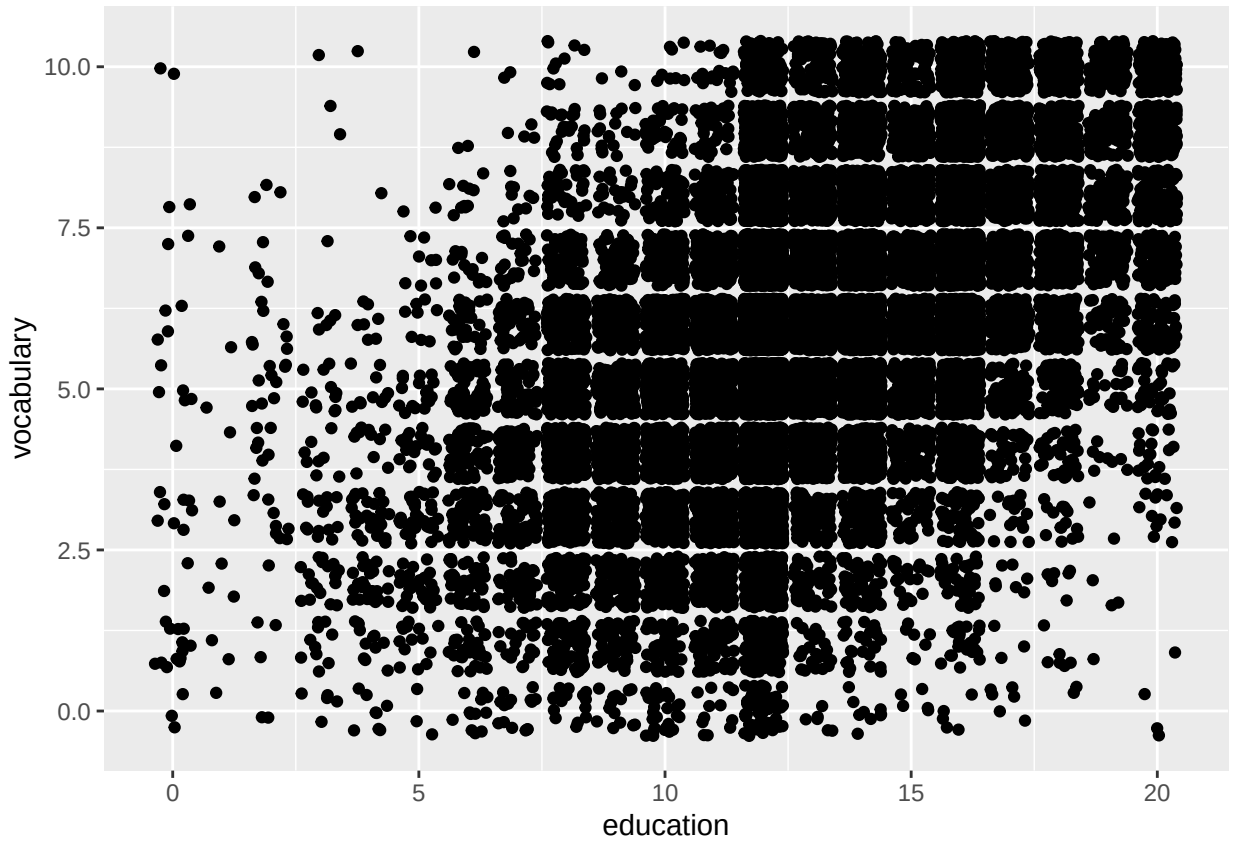
```
data(Vocab)

# Examine the structure of Vocab
str(Vocab)
```

```
## 'data.frame':    30351 obs. of  4 variables:
##  $ year      : num  1974 1974 1974 1974 1974 ...
##  $ sex       : Factor w/ 2 levels "Female","Male": 2 2 1 1 1 2 2 2 1 1 ...
##  $ education : num  14 16 10 10 12 16 17 10 12 11 ...
##  $ vocabulary: num  9 9 9 5 8 8 9 5 3 5 ...
##  - attr(*, "na.action")= 'omit' Named int [1:32115] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "names")= chr [1:32115] "19720001" "19720002" "19720003" "19720004" ...
```

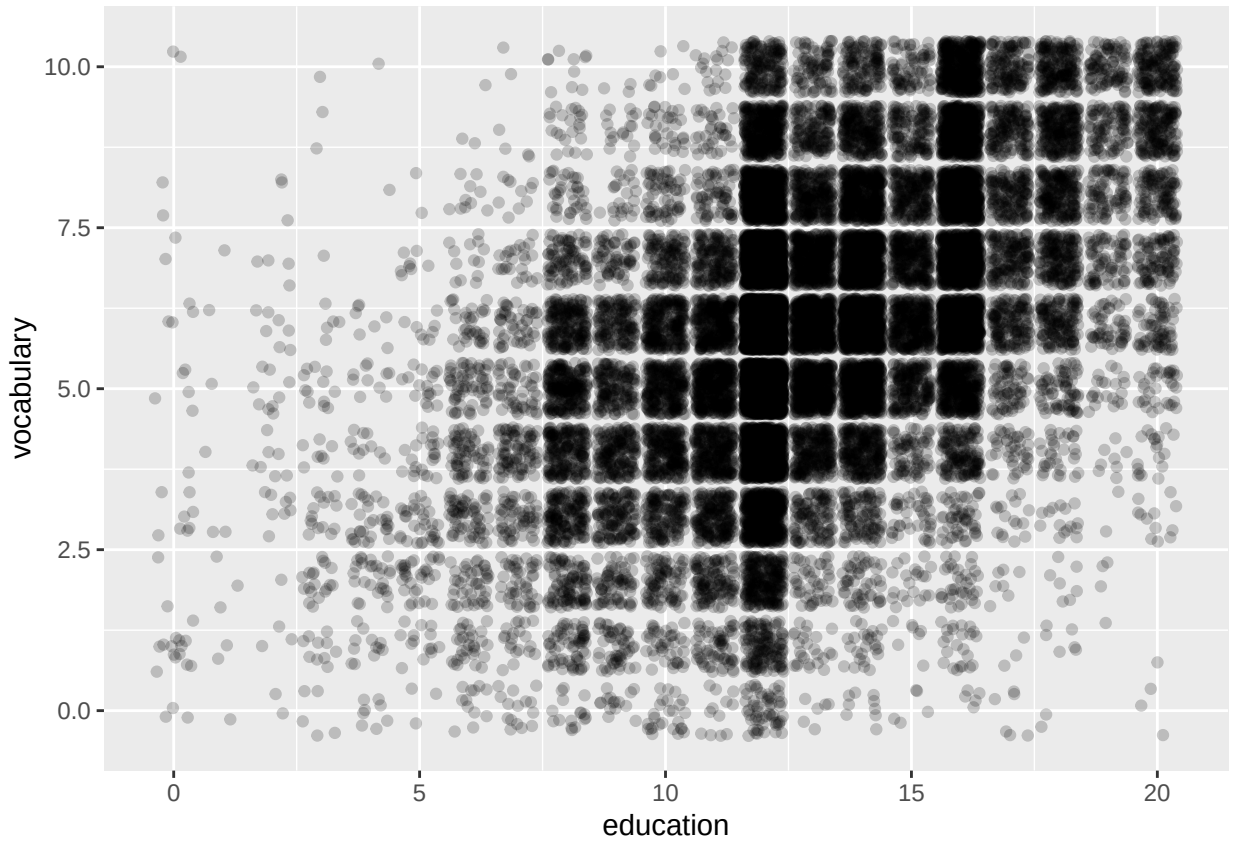
```
# Plot vocabulary vs. education
ggplot(Vocab, aes(education, vocabulary)) +
  # Add a point layer
  geom_point()
```



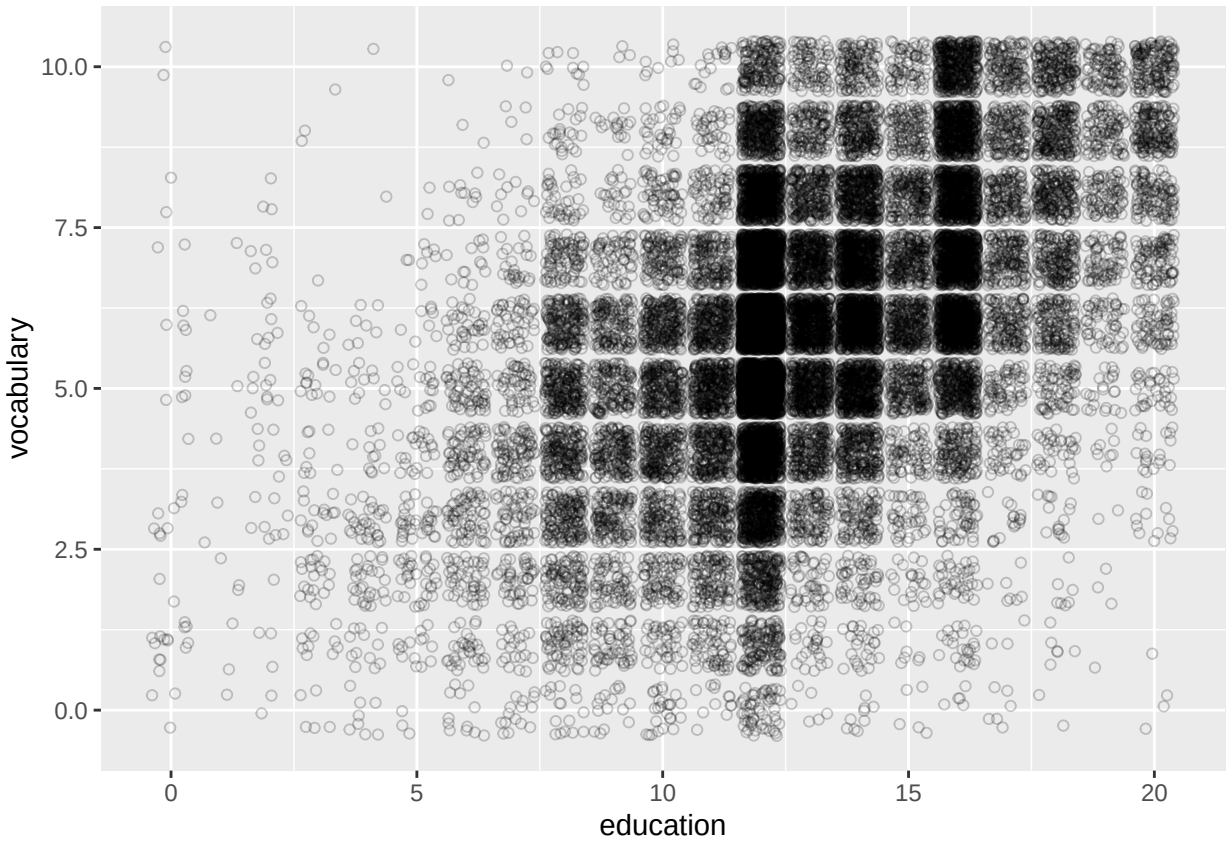

```
ggplot(Vocab, aes(education, vocabulary)) +
  # Change to a jitter layer
  geom_jitter()
```




```
ggplot(Vocab, aes(education, vocabulary)) +
  # Set the transparency to 0.2
  geom_jitter(alpha=0.2)
```




```
ggplot(Vocab, aes(education, vocabulary)) +
  # Set the shape to 1
  geom_jitter(alpha = 0.2, shape=1)
```



6. Historgrams

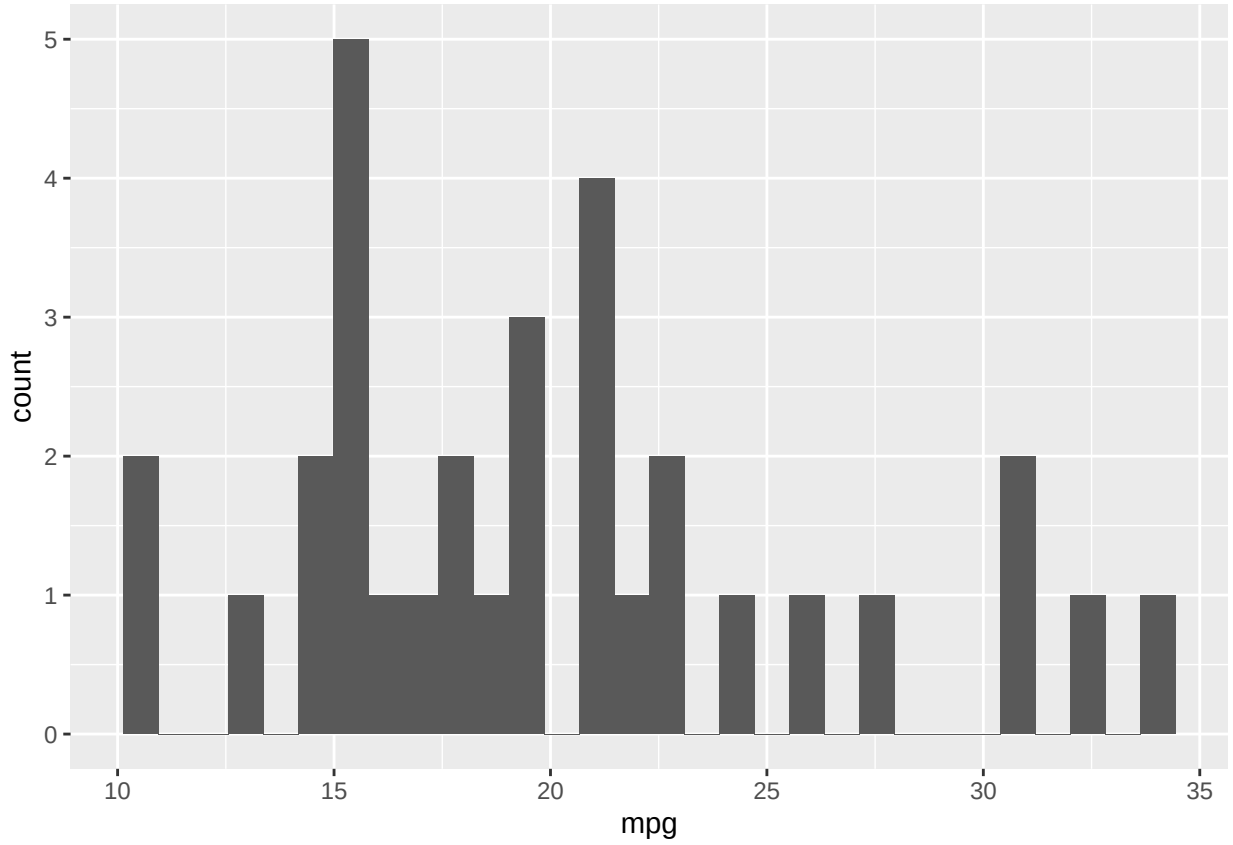
```
1. Histograms
In this section we'll take a look at the typical uses of bar plots and their associated geoms.

2. Common plot types
A histogram is a special type of bar plot that shows the binned distribution of a continuous variable.

3. Histograms
Here, we only need a single aesthetic: X, a continuous variable. geom_histogram plots a a binned versio

4. Default of 30 even bins
This is a good starting point, but we don't need to settle for defaults! Let's change it and see what ha

5. Intuitive and meaningful bin widths
Changing the binwidth argument to 0-point-1 gives us a more intuitive impression of our data. Note that

6. Re-position tick marks
That's also why the labels on the x axis shouldn't fall directly on the bars, but between the bars. The

7. Different Species
Remember that we have three species in our data set? We can fill the bars according to each species. Th

8. Default position is "stack"
The default position is stack. In some cases, this may not be clear, so don't risk confusing your viewe

9. position = "dodge"
We can "dodge" our bars, which is a data viz term that simply means to off-set set each data point in a
```

10. position = "fill"
The fill position normalizes each bin to represent the proportion of all observations in each bin. The y

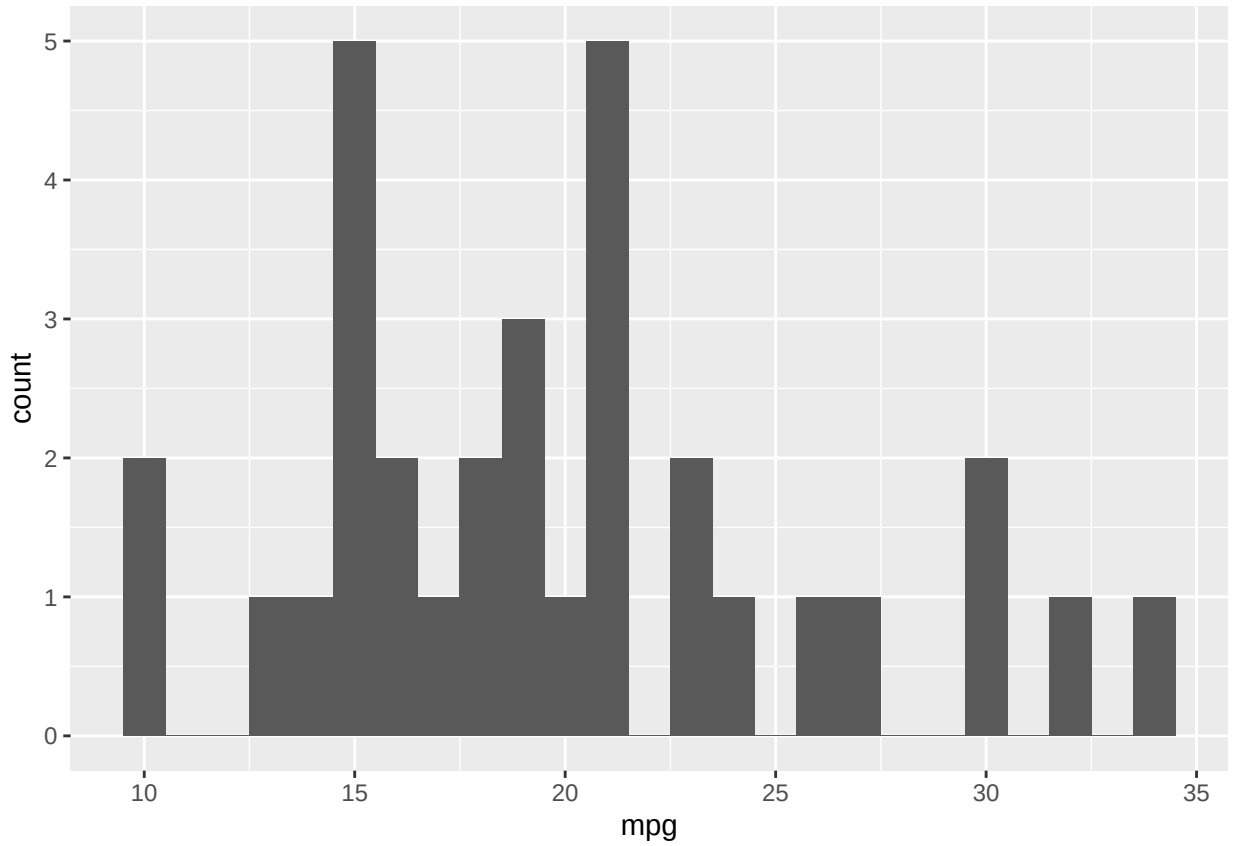
7. Drawing histograms

```
# Plot mpg
ggplot(mtcars, aes(mpg)) +
  # Add a histogram layer
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

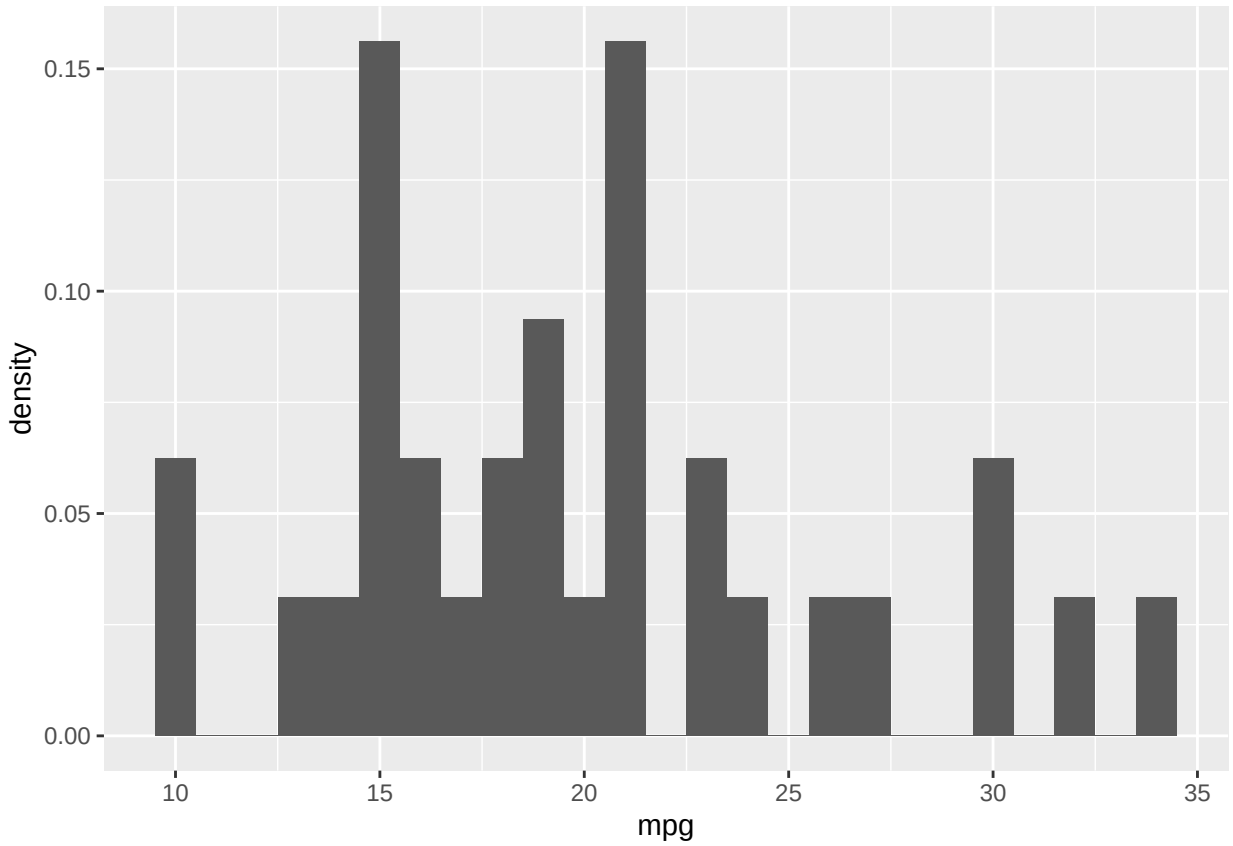



```
ggplot(mtcars, aes(mpg)) +
  # Set the binwidth to 1
  geom_histogram(binwidth=1)
```

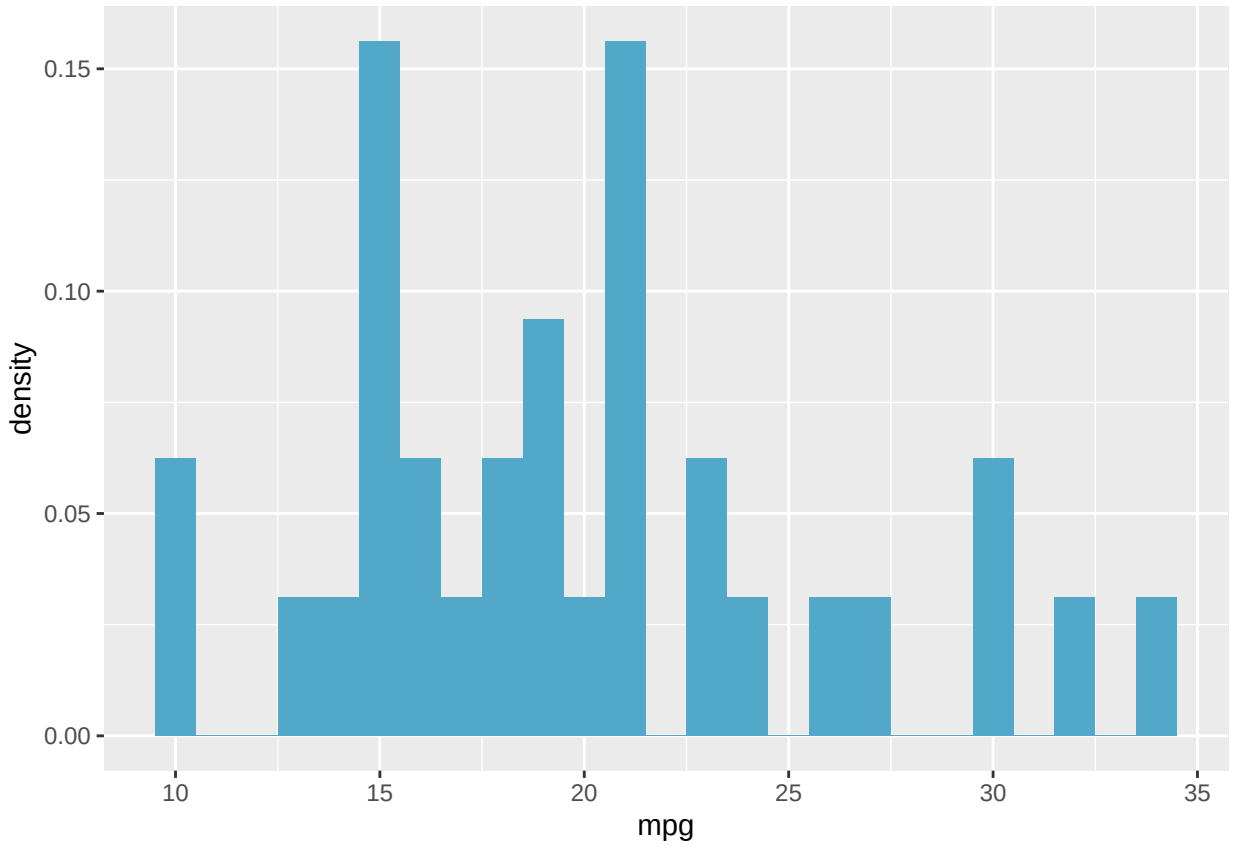



```
# Map y to ..density..
ggplot(mtcars, aes(mpg, ..density..)) +
  geom_histogram(binwidth = 1)
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
```

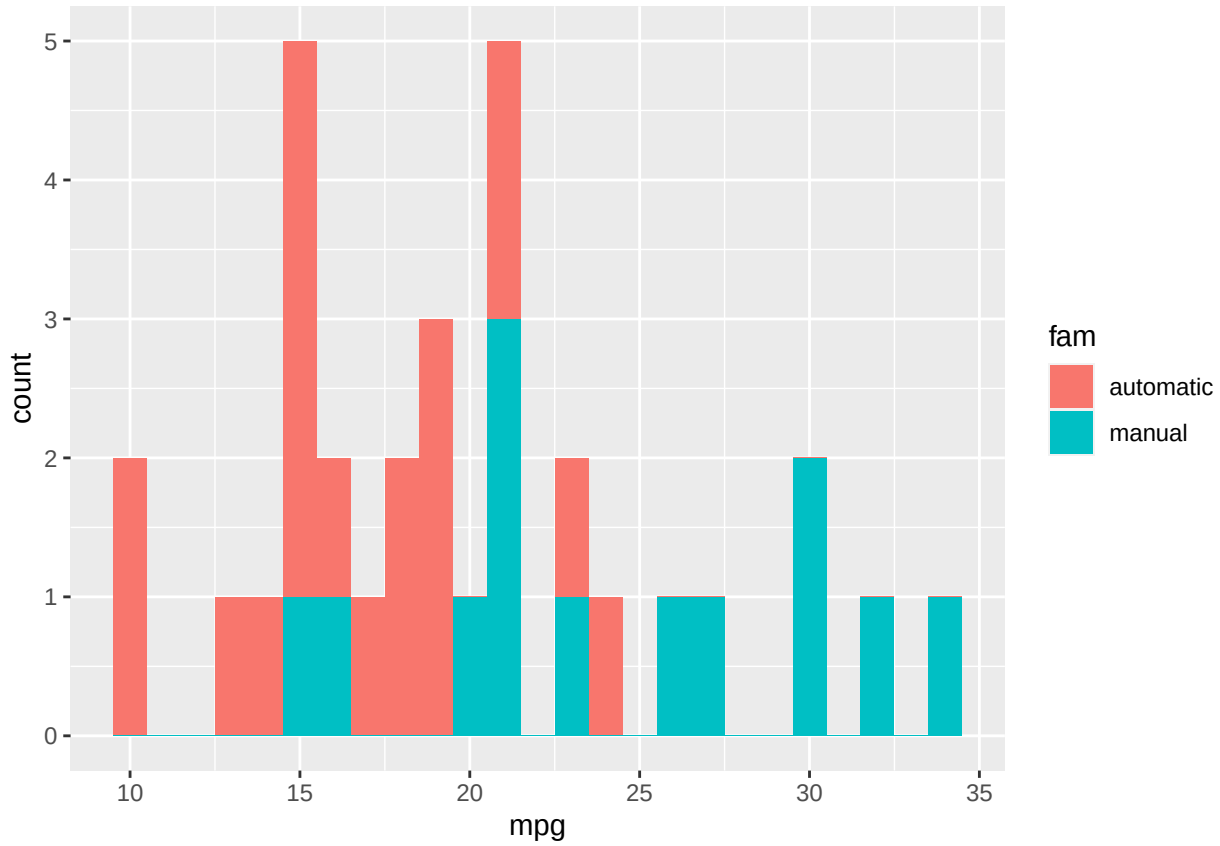



```
datacamp_light_blue <- "#51A8C9"

ggplot(mtcars, aes(mpg, ..density..)) +
  # Set the fill color to datacamp_light_blue
  geom_histogram(binwidth = 1, fill=datacamp_light_blue)
```

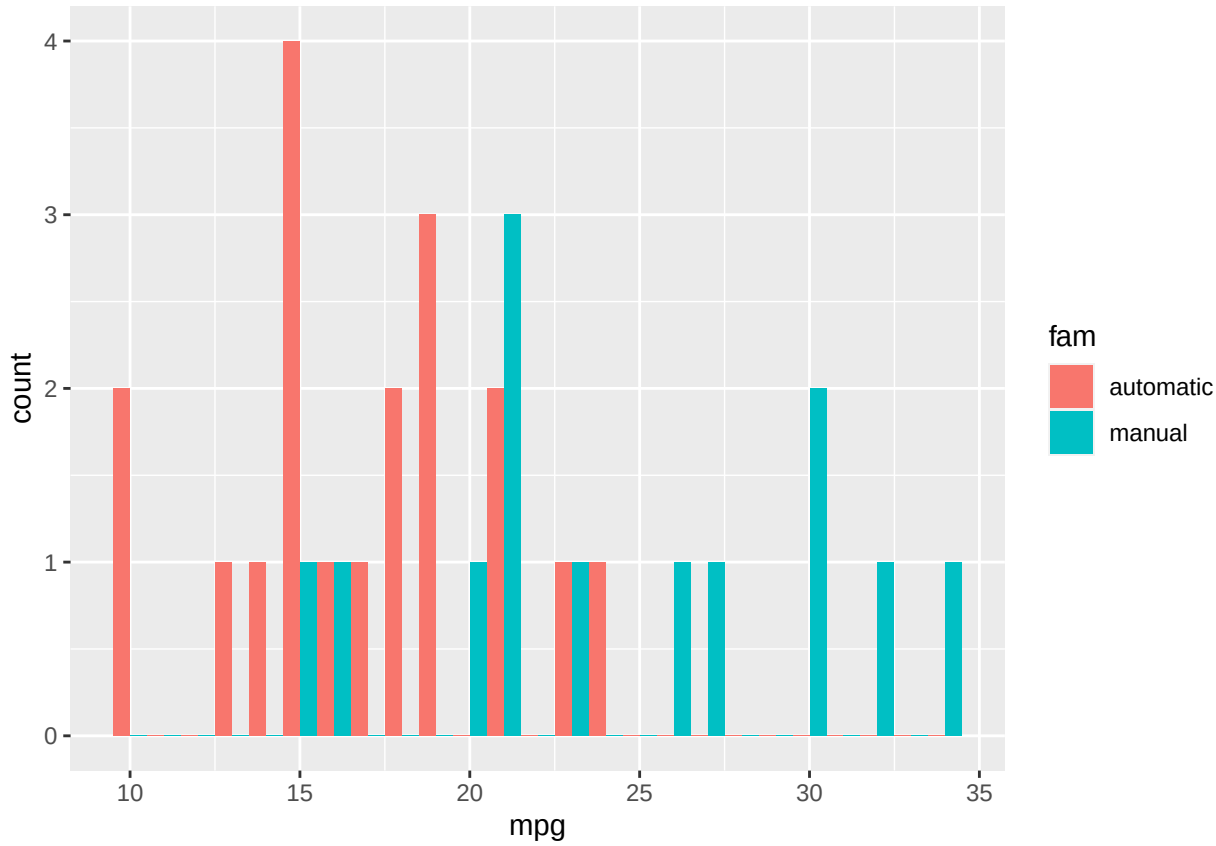


8. Positions in histograms

```
# Update the aesthetics so the fill color is by fam
ggplot(mtcars, aes(mpg, fill=fam)) +
  geom_histogram(binwidth = 1)
```

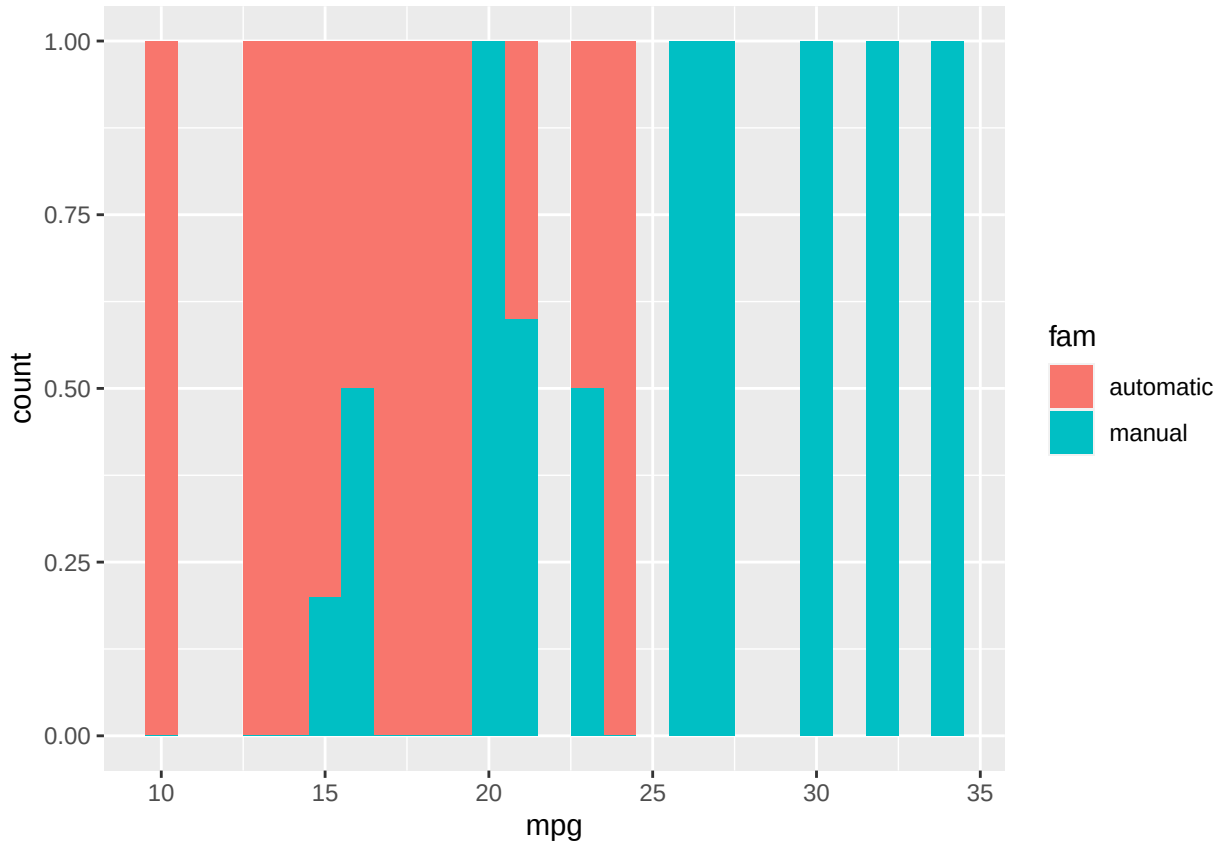



```
ggplot(mtcars, aes(mpg, fill = fam)) +
  # Change the position to dodge
  geom_histogram(binwidth = 1, position="dodge")
```

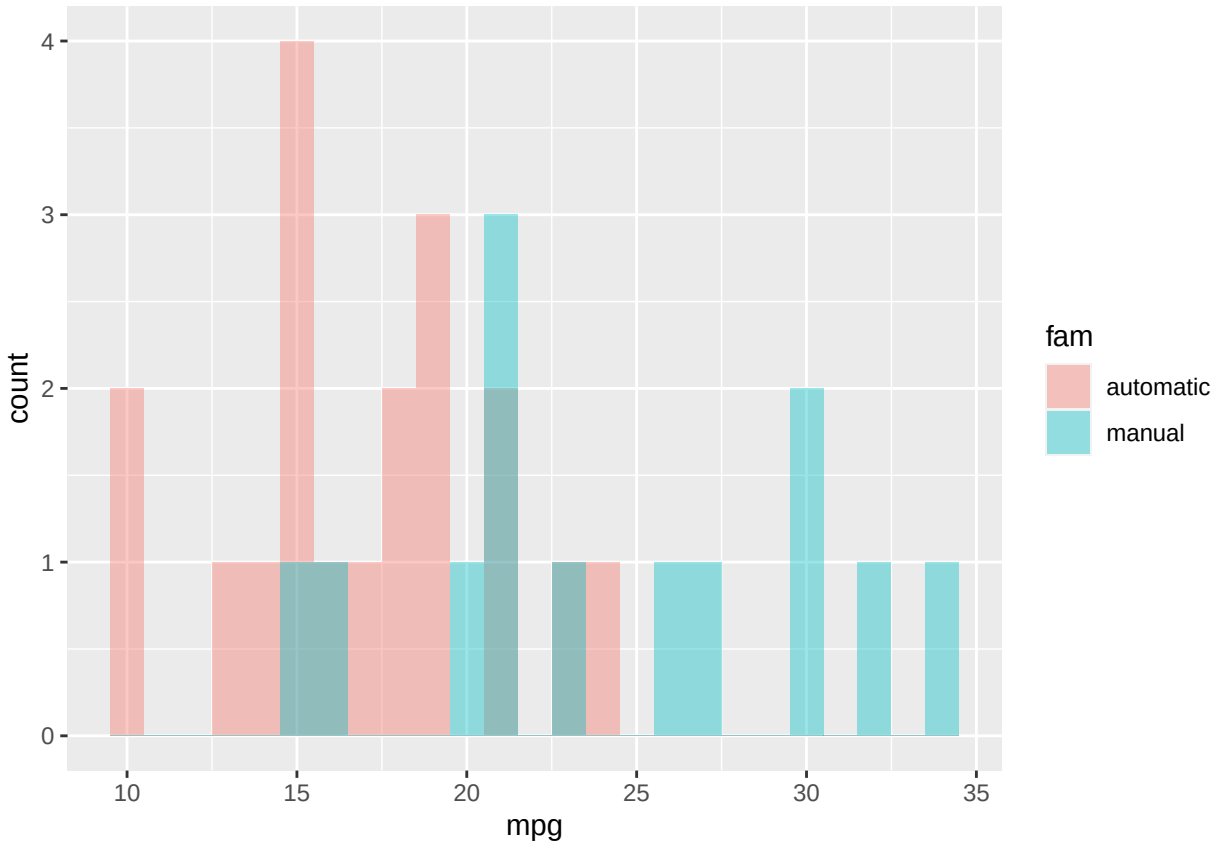



```
ggplot(mtcars, aes(mpg, fill = fam)) +
  # Change the position to fill
  geom_histogram(binwidth = 1, position = "fill")
```

```
## Warning: Removed 16 rows containing missing values (`geom_bar()`).
```




```
ggplot(mtcars, aes(mpg, fill = fam)) +
  # Change the position to identity, with transparency 0.4
  geom_histogram(binwidth = 1, position = "identity", alpha=0.4)
```



9. Bar plots

1. Bar plots
In the last video, we saw that histograms are a specialized version of bar plots, where we have binned a

2. Bar Plots, with a categorical X-axis
Classic bar plots refer to a categorical X-axis. Here we need to use either geom_bar or geom_col.

3. Bar Plots, with a categorical X-axis
geom_bar will count the number of cases in each category of the variable mapped to the x-axis, whereas g

4. Bar Plots, with a categorical X-axis
All the positions we just looked at are available in bar plots. You will encounter two types of bar plot

5. Habits of mammals
We'll use a data set containing information on the REM sleep time and eating habits of a variety of mamm

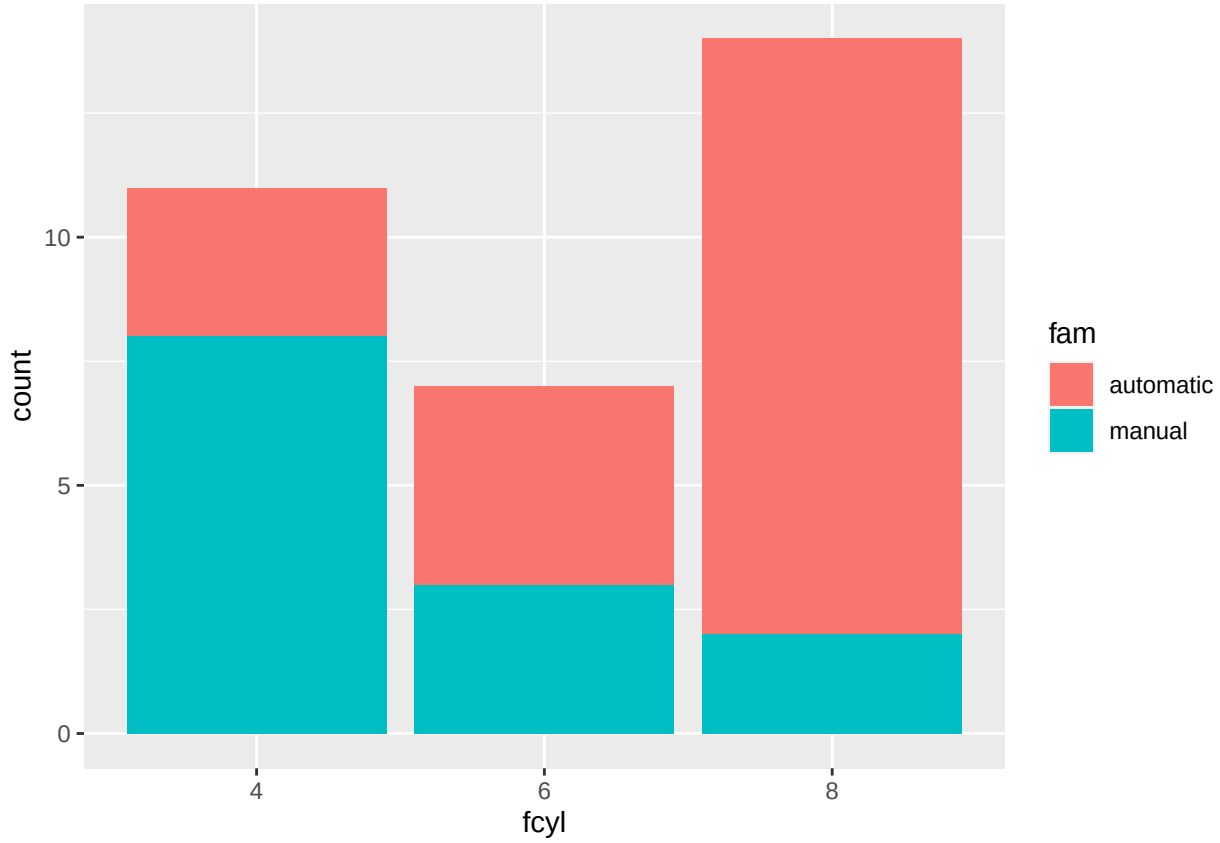
6. Bar plot
In this bar plot, we've split our data set according to eating behavior and simply asked how many observ

7. Plotting distributions instead of absolute counts
Often times this is the case - you will have descriptive statistics already calculated, but remember tha

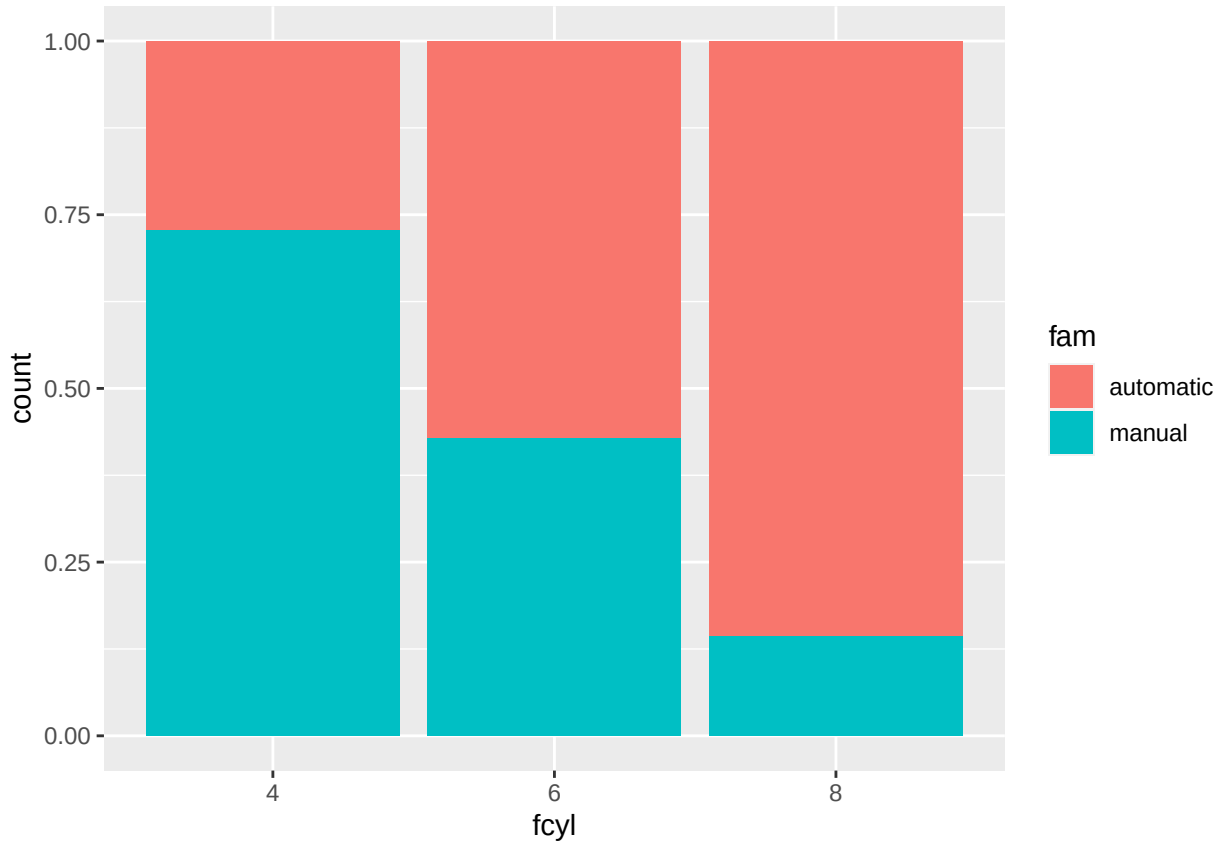
8. Plotting distributions
If we want to plot the average sepal width for each species, we can map the avg column in our dataset or

9. Position in bar and col plots

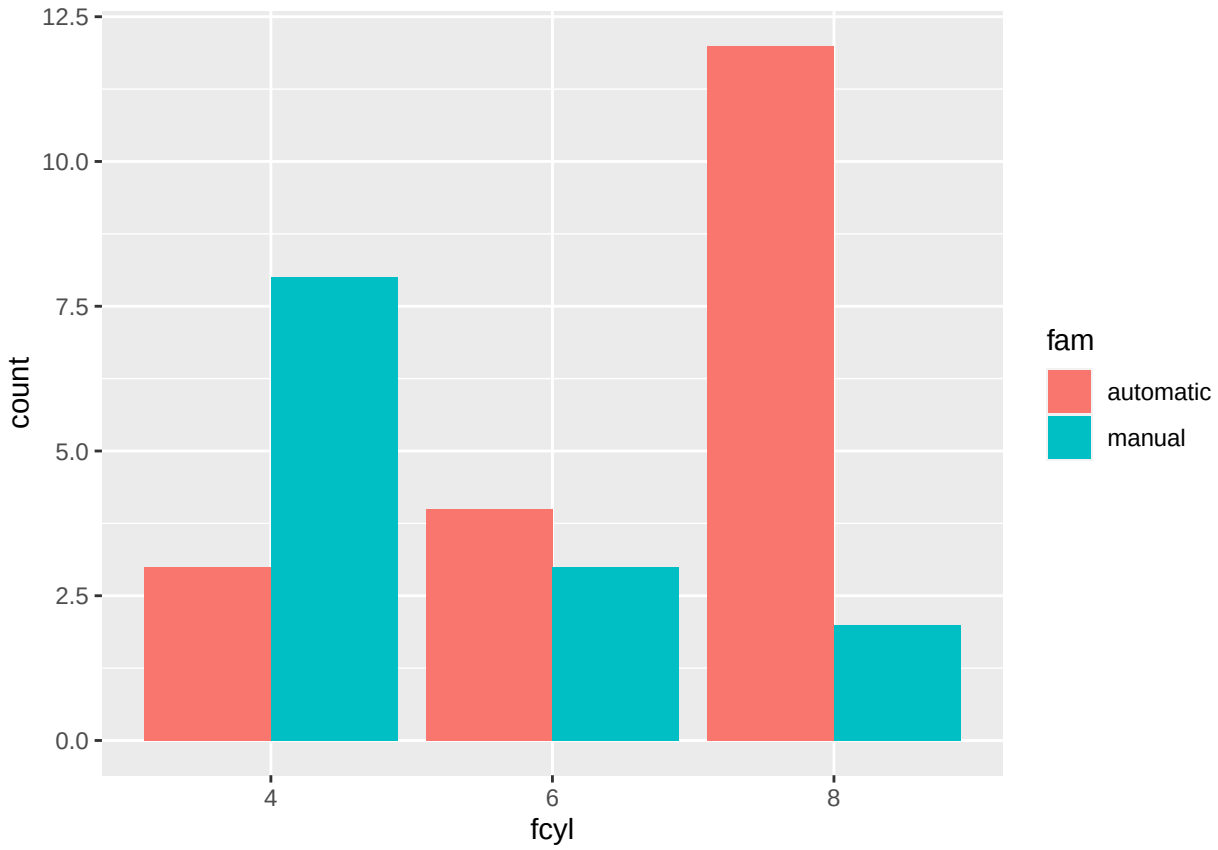
```
# Plot fcyl, filled by fam
ggplot(mtcars, aes(fcyl, fill=fam)) +
  # Add a bar layer
  geom_bar()
```




```
ggplot(mtcars, aes(fcyl, fill = fam)) +
  # Set the position to "fill"
  geom_bar(position="fill")
```

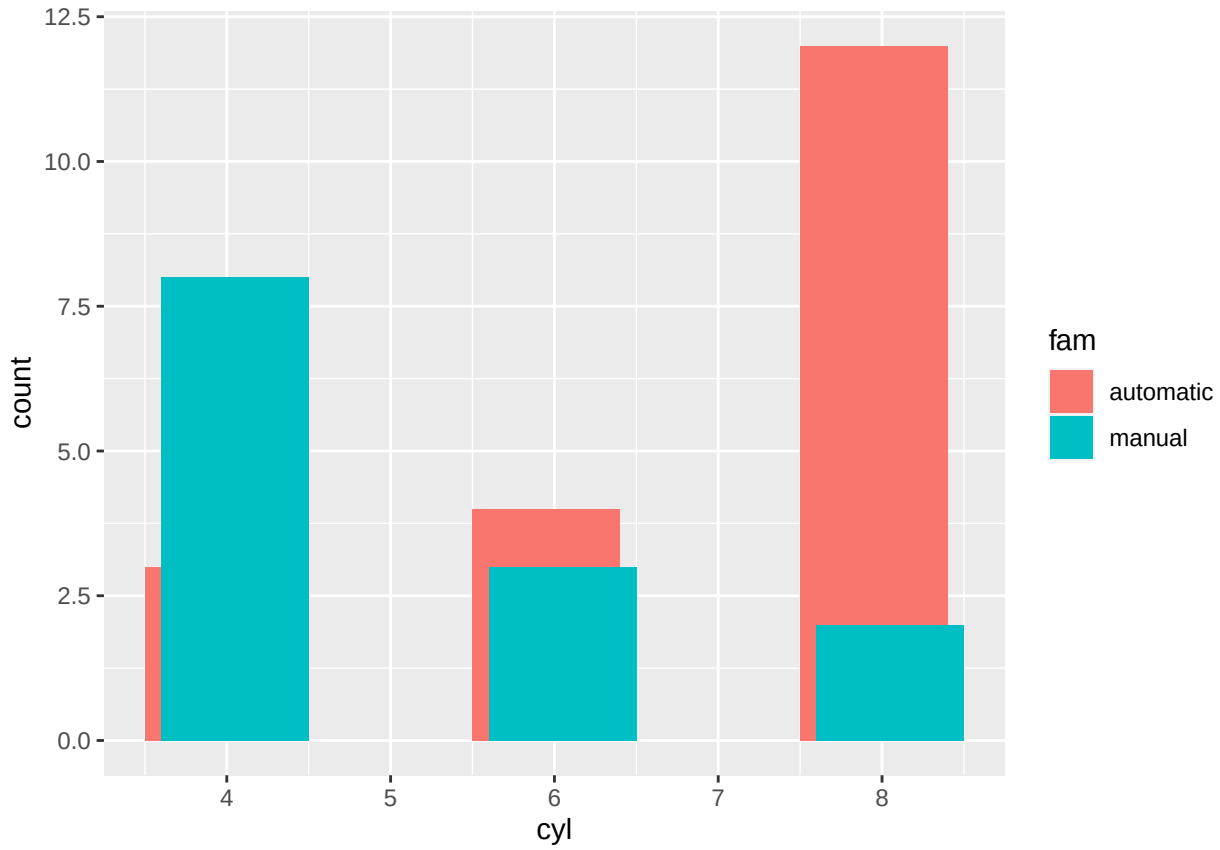



```
ggplot(mtcars, aes(fcyl, fill = fam)) +
  # Change the position to "dodge"
  geom_bar(position = "dodge")
```

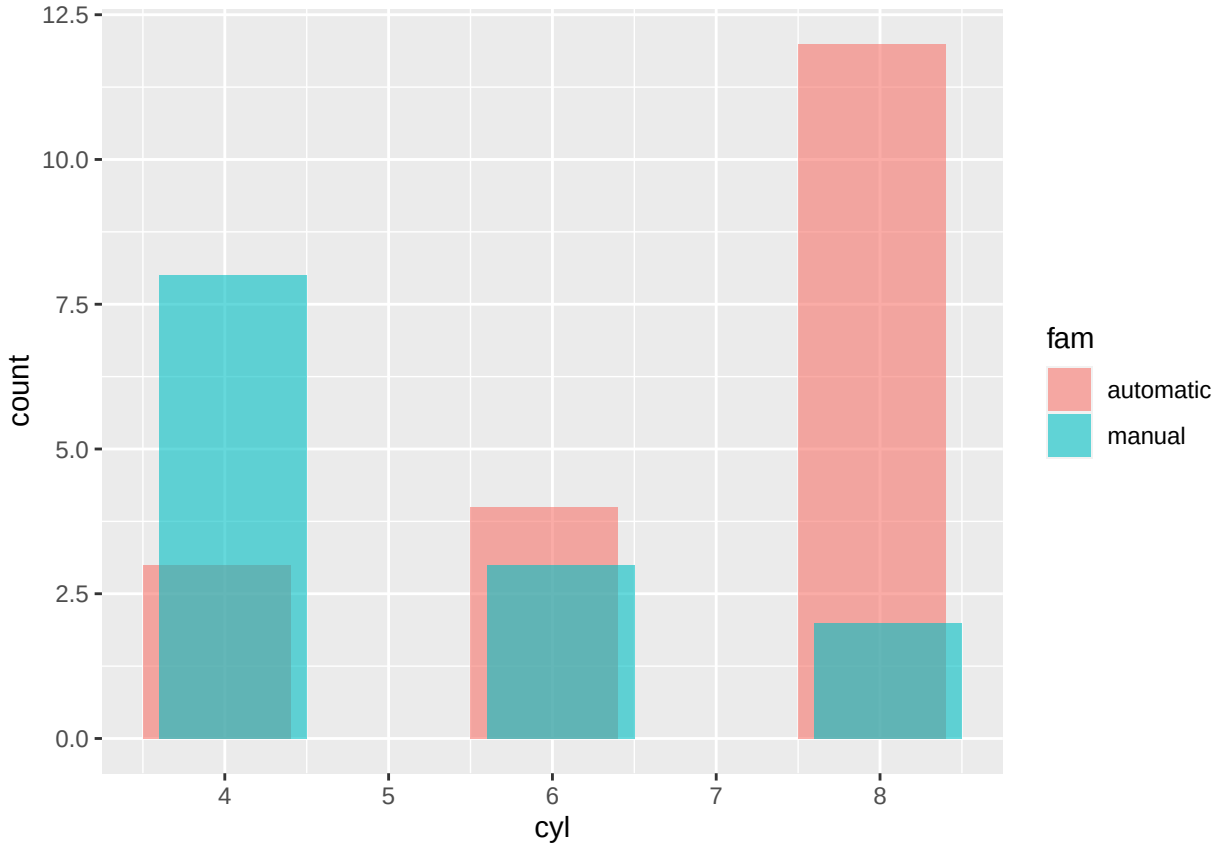


10. Overlapping bar plots

```
ggplot(mtcars, aes(cyl, fill = fam)) +
  # Change position to use the functional form, with width 0.2
  geom_bar(position = position_dodge(width=0.2))
```

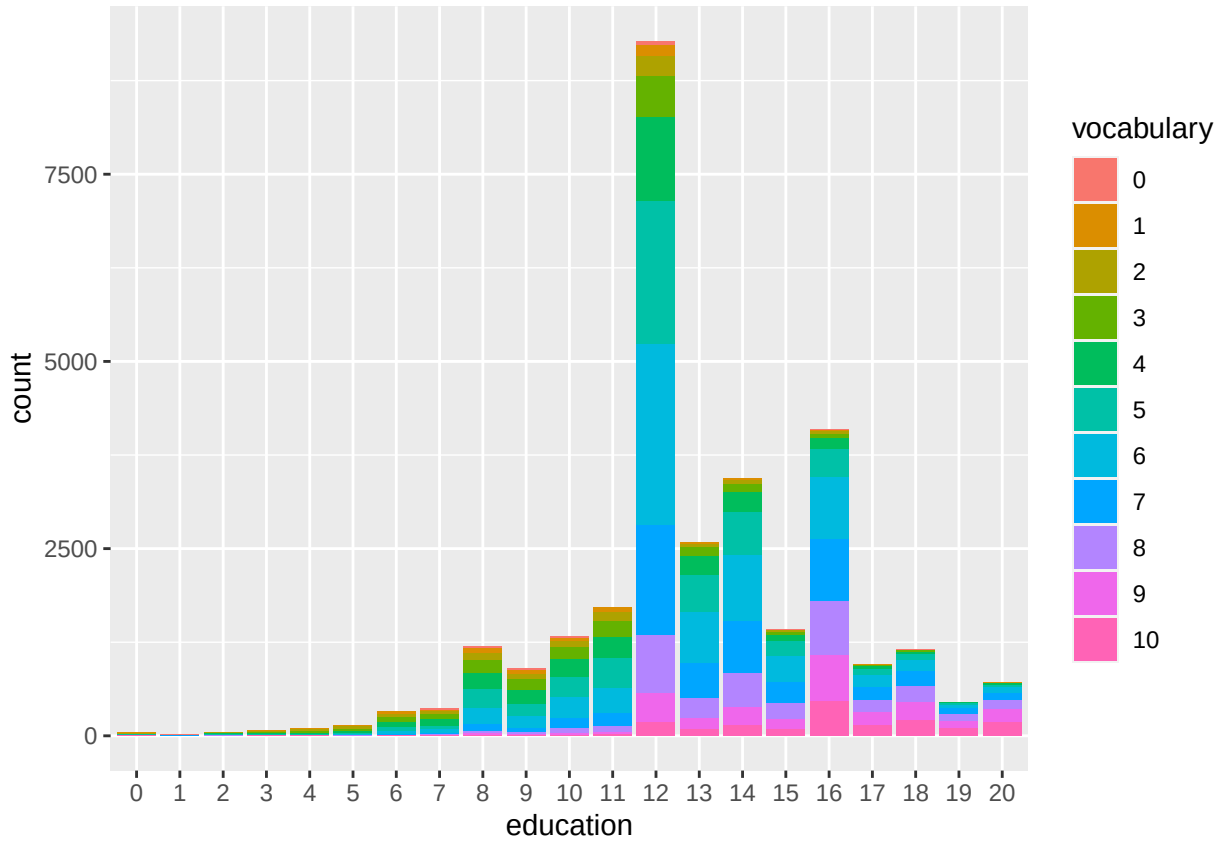



```
ggplot(mtcars, aes(cyl, fill = fam)) +
  # Set the transparency to 0.6
  geom_bar(position = position_dodge(width = 0.2), alpha=0.6)
```

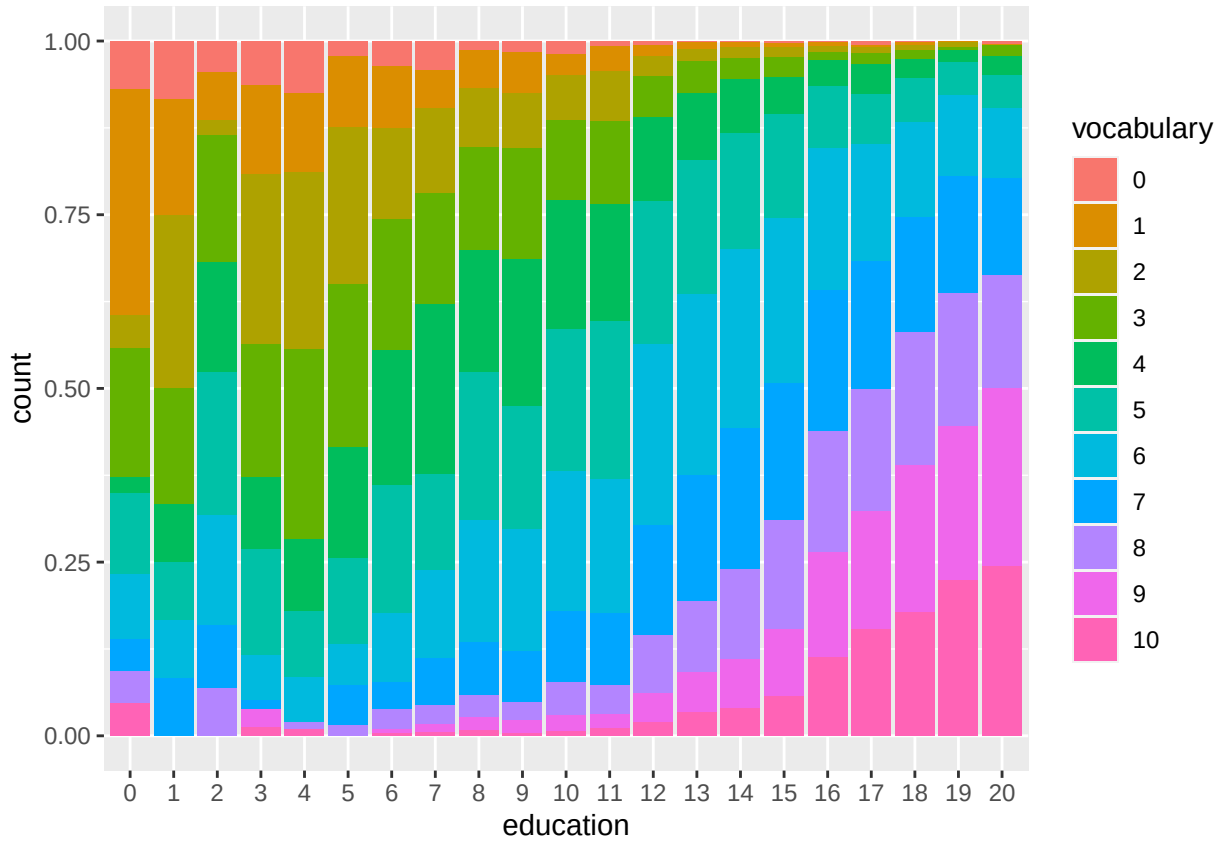


11. Bar plots: sequential color palette

```
Vocab$education <- as.factor(Vocab$education)
Vocab$vocabulary<- as.factor(Vocab$vocabulary)

# Plot education, filled by vocabulary
ggplot(Vocab, aes(education, fill=vocabulary)) +
    geom_bar()
```

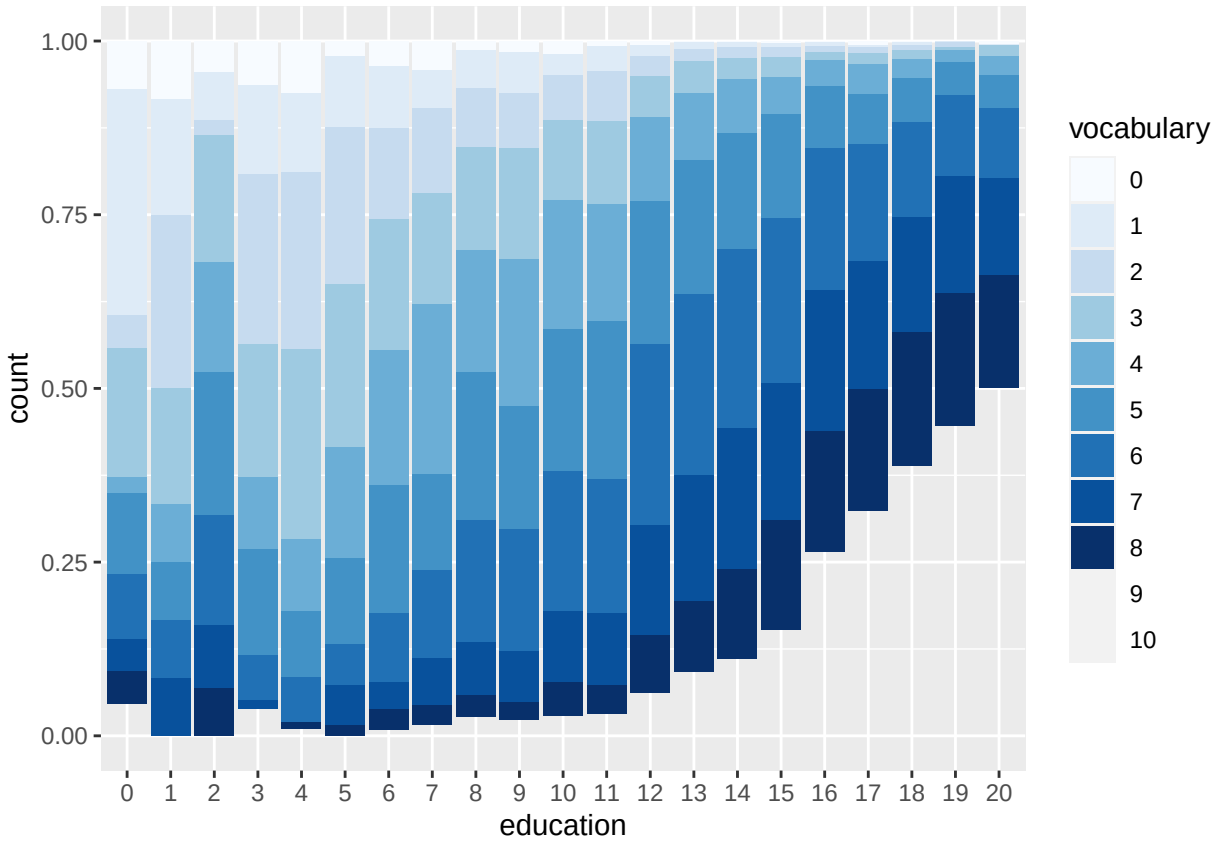



```
# Plot education, filled by vocabulary
ggplot(Vocab, aes(education, fill = vocabulary)) +
  # Add a bar layer with position "fill"
  geom_bar(position="fill")
```




```
# Plot education, filled by vocabulary
ggplot(Vocab, aes(education, fill = vocabulary)) +
  # Add a bar layer with position "fill"
  geom_bar(position = "fill") +
  # Add a brewer fill scale with default palette
  scale_fill_brewer()
```

```
## Warning in RColorBrewer::brewer.pal(n, pal): n too large, allowed maximum for palette Blues is 9
## Returning the palette you asked for with that many colors
```



12. Line plots

```
1. Line plots
Line plots are another very common plot type.

2. Common plot types
We'll take a look at two examples of lines plots in situations in which they are very well-suited - time

3. Beaver
In this first example our data set contains temperature measurements of a beaver in 10 minute intervals

4. Beaver
Our basic line plot follows the syntax we've see so far. This is the simplest case scenario. Let's look

5. Beaver
We can directly color each segment of our line according to another variable which is set as a color aes

6. The fish catch dataset
In the fish data set we have the global catch of 7 varieties of salmon over a 60 year period.

7. Linetype aesthetic
When we have multiple lines, we have to consider which aesthetic is more appropriate in allowing us to d

8. Size aesthetic
Using size is even worse! Don't forget to use your common sense here.

9. Color aesthetic
```

Using color allows for easily distinguishable groups. There are a couple other ways of showing lines.

10. Aesthetics for categorical variables
The most salient choice is color, when available, since it allows the easiest way of distinguishing betw

11. Fill aesthetic with geom_area()
For example, we could have used an area fill, with geom_area, which defaults to position "stack", so ins

12. Using position = "fill"
If we use position"fill" we'll get a proportion the total capture for each fish at each time-point. Note

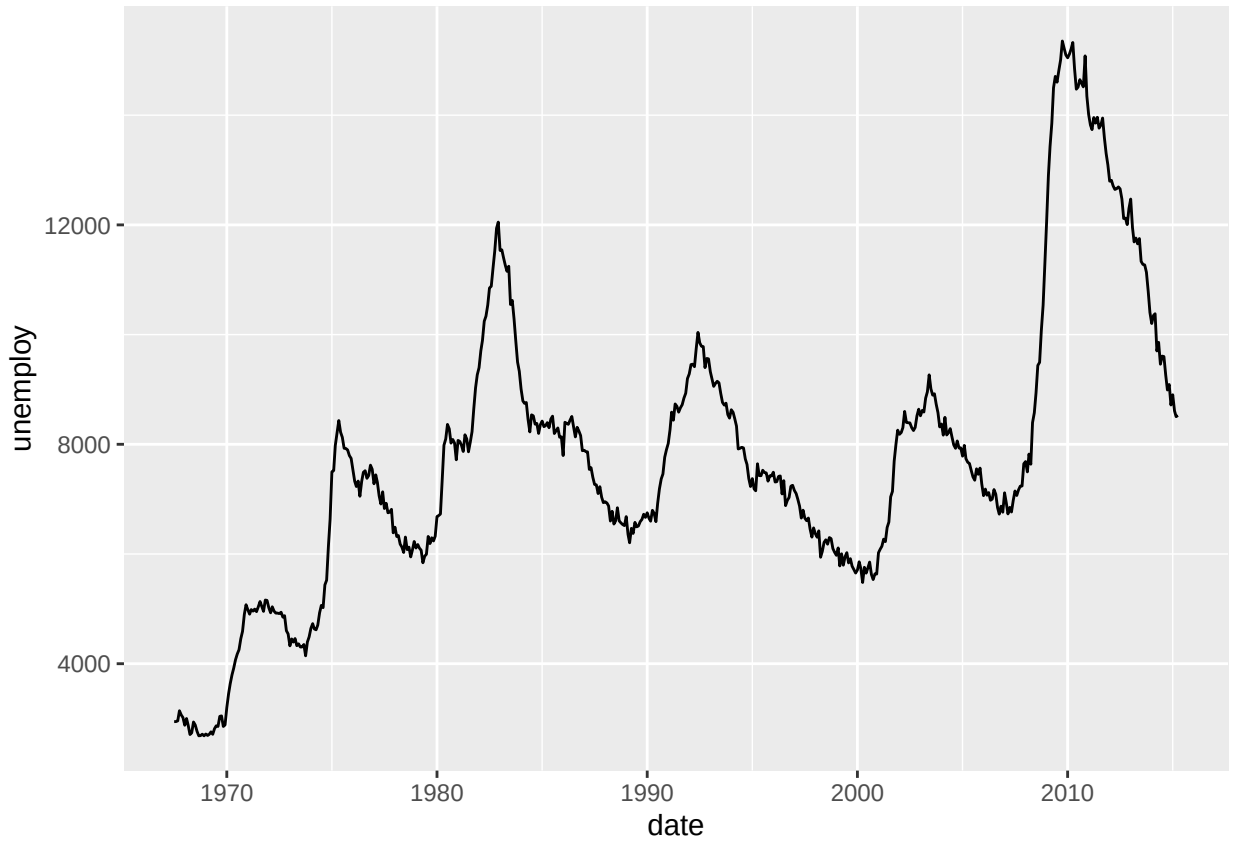
13. geom_ribbon()
The final type of plot we'll look at is when we would want to have overlapping areas plots. In this case

13. Basic line plots

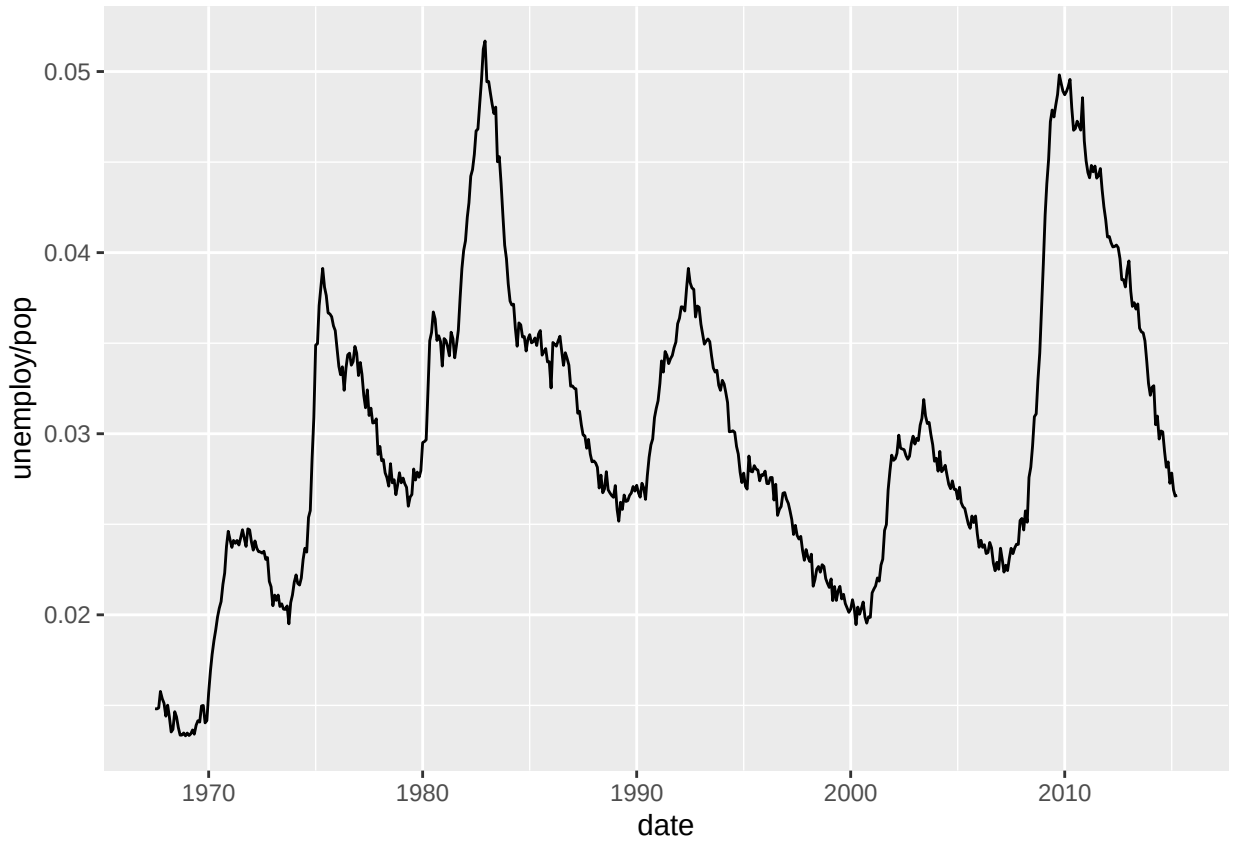
```
# Print the head of economics
head(economics)
```

```
## # A tibble: 6 x 6
##   date         pce    pop psavert uempmed unemploy
##   <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
## 1 1967-07-01  507. 198712    12.6     4.5     2944
## 2 1967-08-01  510. 198911    12.6     4.7     2945
## 3 1967-09-01  516. 199113    11.9     4.6     2958
## 4 1967-10-01  512. 199311    12.9     4.9     3143
## 5 1967-11-01  517. 199498    12.8     4.7     3066
## 6 1967-12-01  525. 199657    11.8     4.8     3018
```

```
# Using economics, plot unemploy vs. date
ggplot(economics, aes(date, unemploy)) +
  # Make it a line plot
  geom_line()
```




```
# Change the y-axis to the proportion of the population that is unemployed
ggplot(economics, aes(date, unemploy/pop)) +
  geom_line()
```



14. Multiple time series

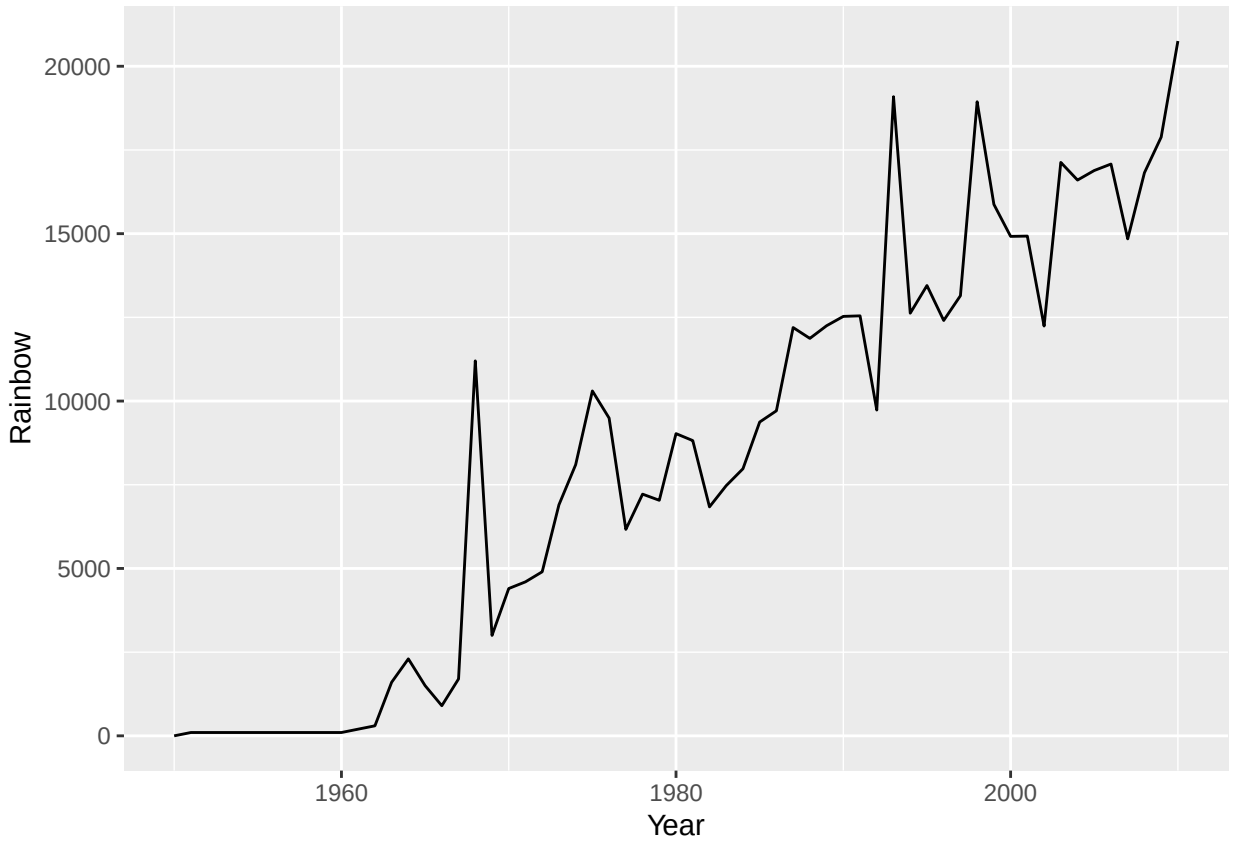
```
# load fish dataset with absolute path or select from files
load("/Users/sarahlin/Desktop/DataCamp/1_5_ggplot2_r/datasets/fish.RData")

str(fish.species)
```

```
## 'data.frame':    61 obs. of  8 variables:
##  $ Year    : int  1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 ...
##  $ Pink    : int  100600 259000 132600 235900 123400 244400 203400 270119 200798 200085 ...
##  $ Chum    : int  139300 155900 113800 99800 148700 143700 158480 125377 132407 113114 ...
##  $ Sockeye : int  64100 51200 58200 66100 83800 72000 84800 69676 100520 62472 ...
##  $ Coho    : int  30500 40900 33600 32400 38300 45100 40000 39900 39200 32865 ...
##  $ Rainbow : int  0 100 100 100 100 100 100 100 100 100 ...
##  $ Chinook : int  23200 25500 24900 25300 24500 27700 25300 21200 20900 20335 ...
##  $ Atlantic: int  10800 9701 9800 8800 9600 7800 8100 9000 8801 8700 ...
```

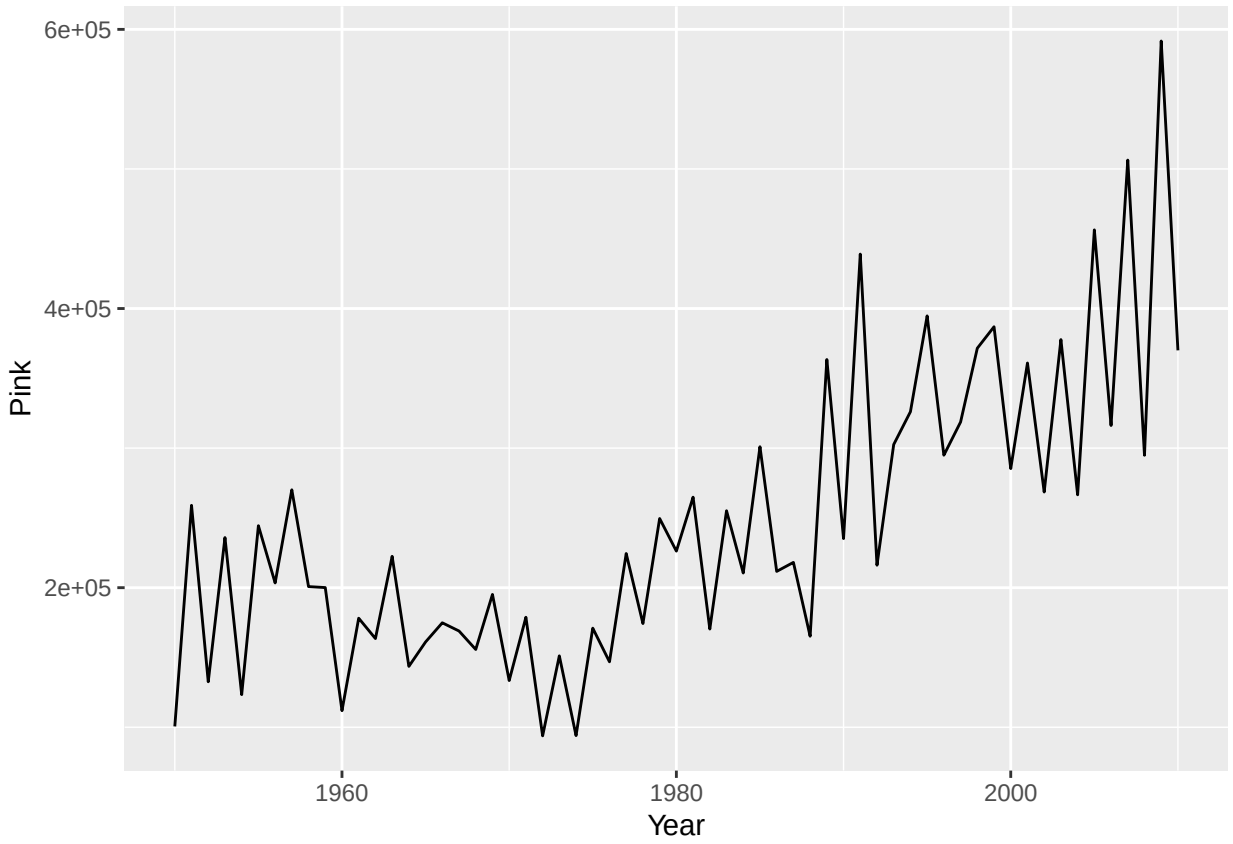
```
str(fish.tidy)
```

```
## 'data.frame':    427 obs. of  3 variables:
##  $ Species: Factor w/ 7 levels "Pink","Chum",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Year   : int  1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 ...
##  $ Capture: int  100600 259000 132600 235900 123400 244400 203400 270119 200798 200085 ...
```

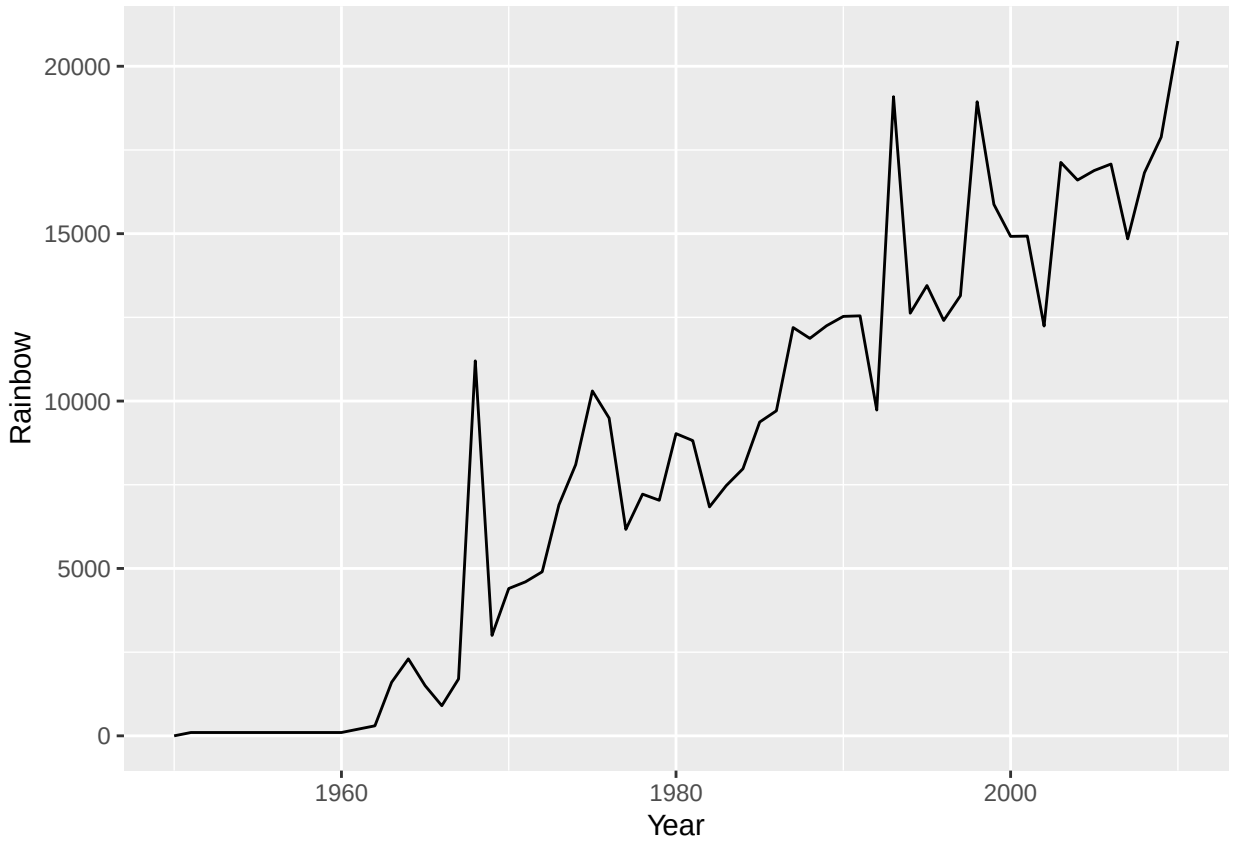
```
# Plot the Rainbow Salmon time series
ggplot(fish.species, aes(x = Year, y = Rainbow)) +
  geom_line()
```



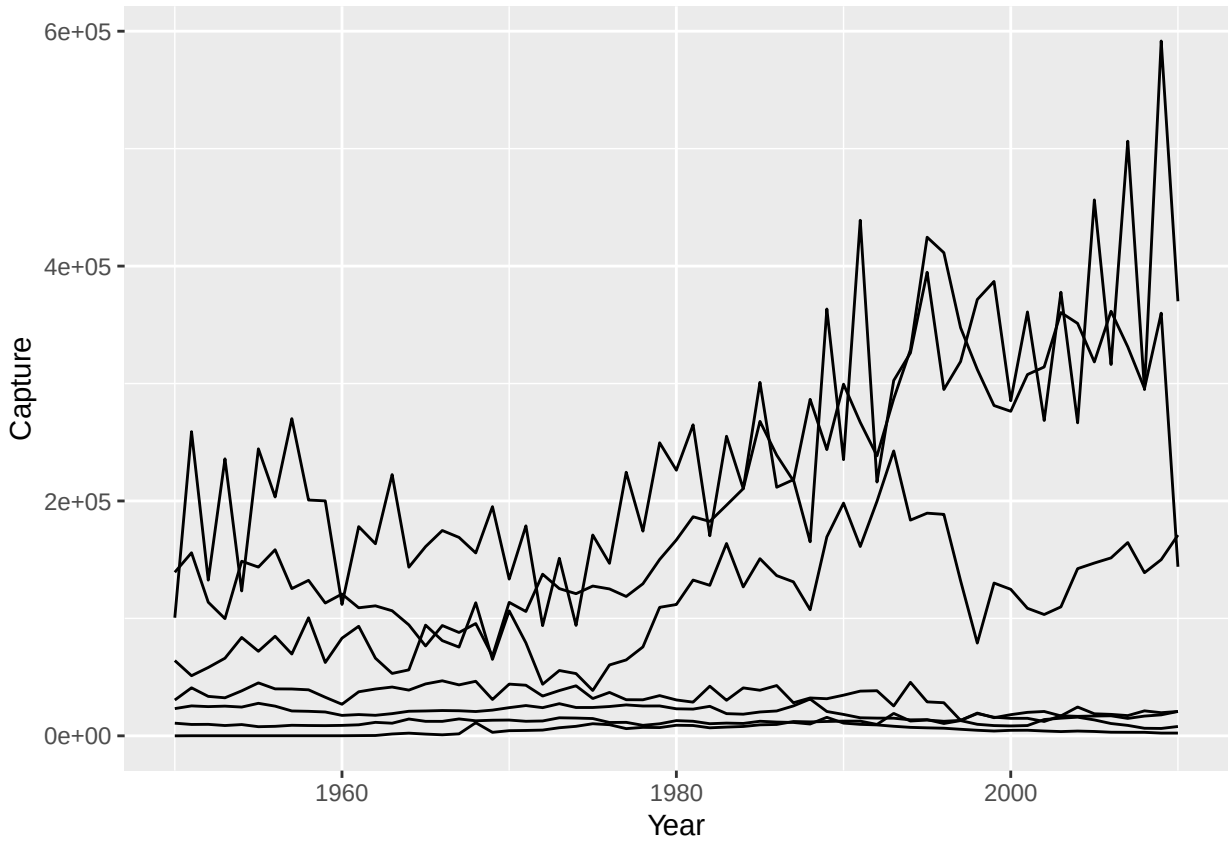

```
# Plot the Pink Salmon time series
ggplot(fish.species, aes(Year, Pink)) +
  geom_line()
```



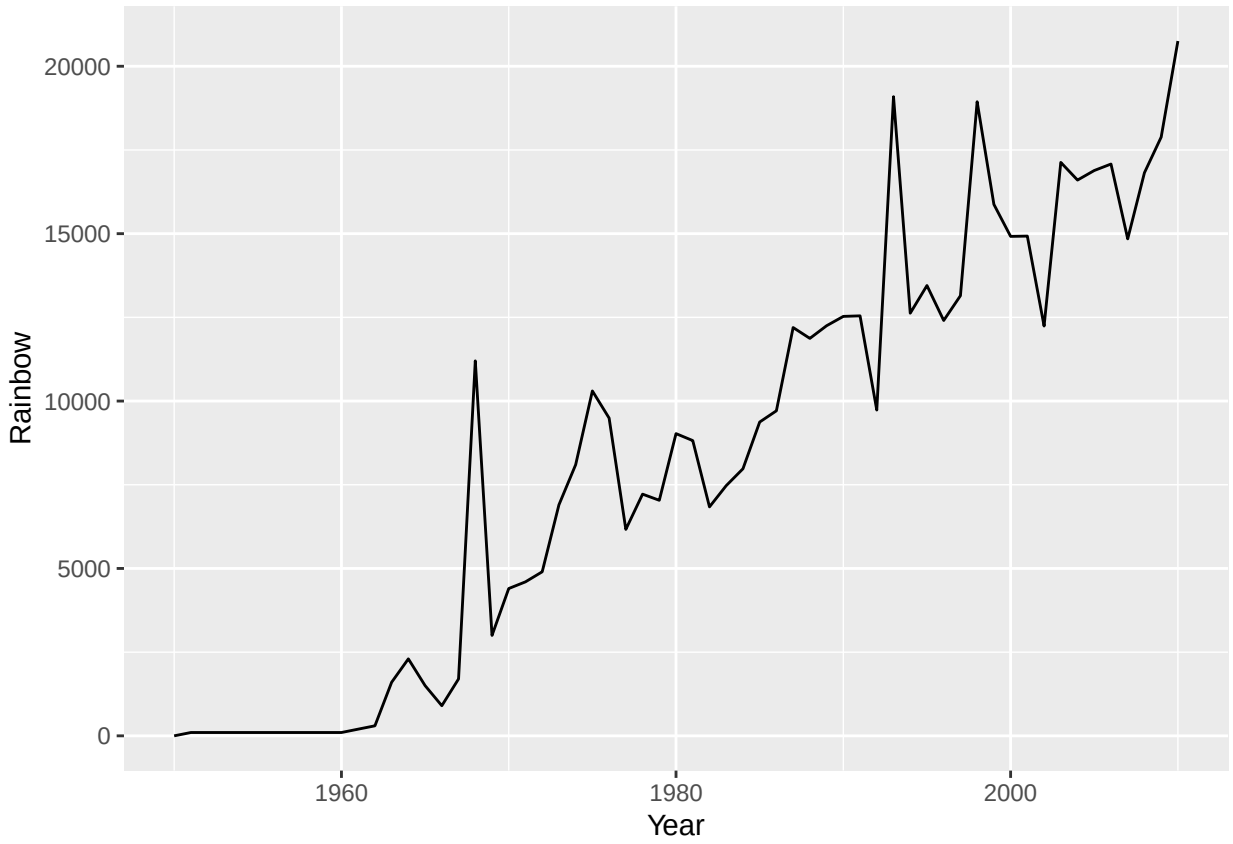

```
# Plot the Rainbow Salmon time series
ggplot(fish.species, aes(x = Year, y = Rainbow)) +
  geom_line()
```



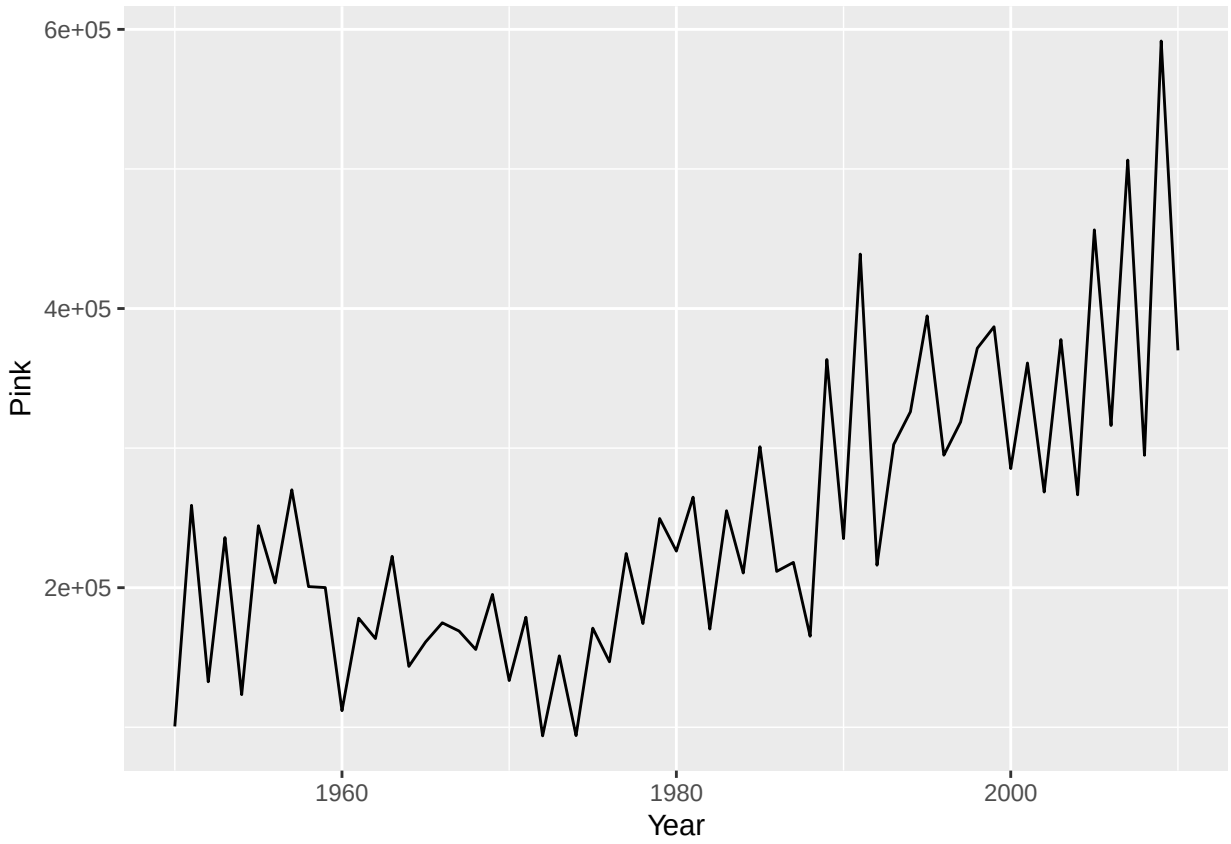

```
# Plot multiple time-series by grouping by species
ggplot(fish.tidy, aes(Year, Capture)) +
  geom_line(aes(group=Species))
```



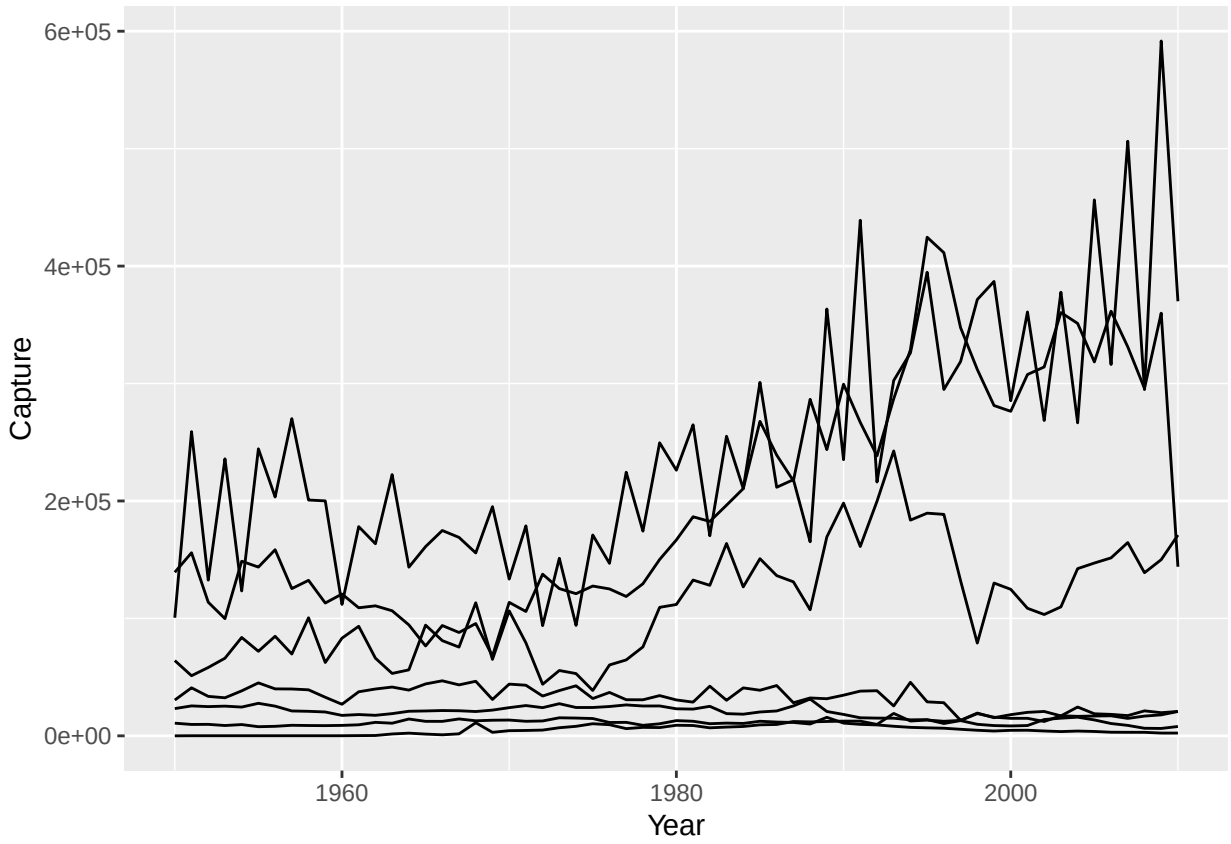

```
# Plot the Rainbow Salmon time series
ggplot(fish.species, aes(x = Year, y = Rainbow)) +
  geom_line()
```



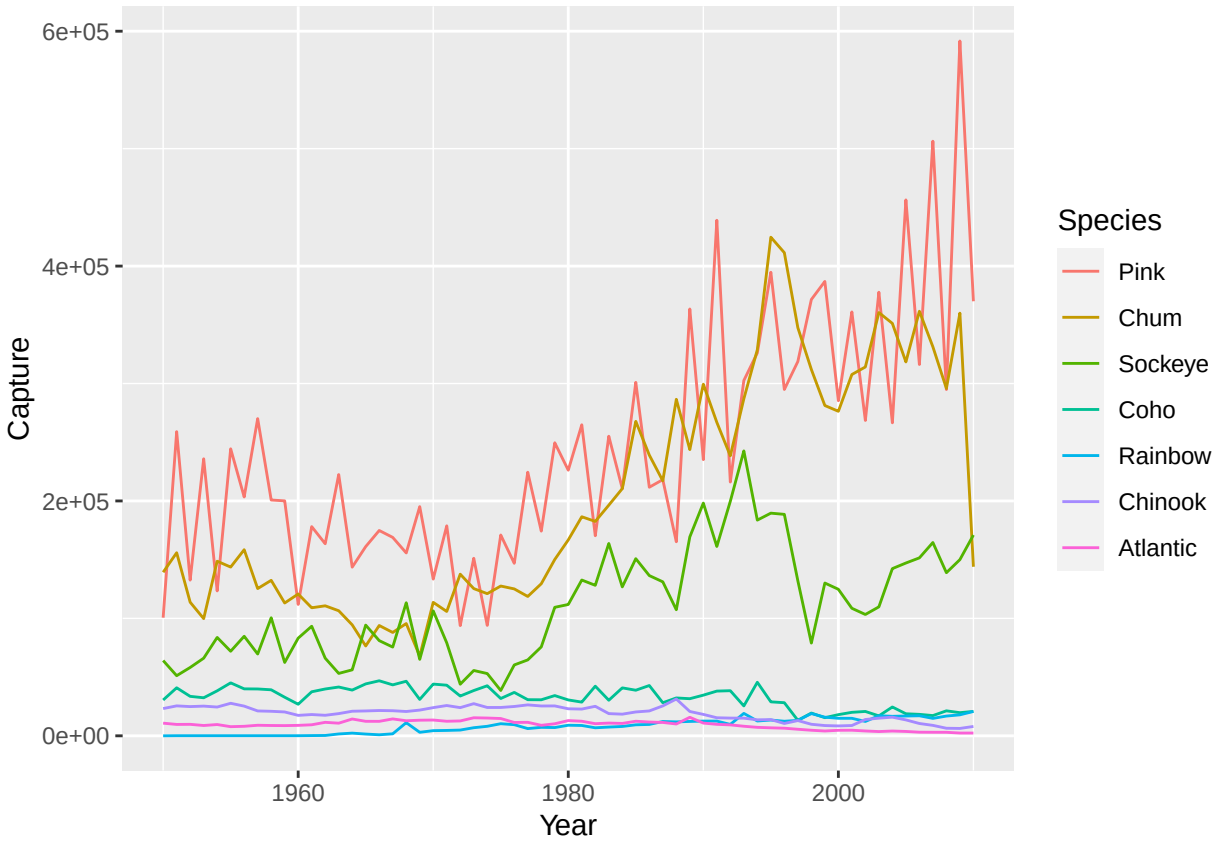

```
# Plot the Pink Salmon time series
ggplot(fish.species, aes(x = Year, y = Pink)) +
  geom_line()
```




```
# Plot multiple time-series by grouping by species
ggplot(fish.tidy, aes(Year, Capture)) +
  geom_line(aes(group = Species))
```




```
# Plot multiple time-series by coloring by species
ggplot(fish.tidy, aes(x=Year, y=Capture, color=Species)) +
  geom_line(aes(group = Species))
```



Part 4. Themes

1. Themes from scratch

```
1. Themes from scratch
The themes layer controls all the non-data ink on your plot.

2. The themes layer
Which are all the visual elements that are not actually part of the data.

3. The themes layer
Visual elements can be classified as one of three different types - text, line or rectangle.

4. The themes layer
Each type can be modified by using the appropriate function, which all begin with element_ followed by

5. A starting plot...
For example, consider this plot that we've already encounter a few times. It's composed out of a combina

6. The text elements
Each element has it's own unique name. We can access all the text in general, all titles in general but

7. The text elements
This is used like all other layers in ggplot, by adding a plus to our plot.

8. Adjusting theme elements
To modify an element, just call its argument in the theme function and use the appropriate element_ func
```

9. A starting plot...
Lines include the tick marks on the axes, the axis lines themselves and all grid lines, both major and m

10. Line elements
These are also all just arguments within the theme function and are modified by the element_line argumen

11. A starting plot...
The remaining non-data ink on our plot are all rectangles of various sizes.

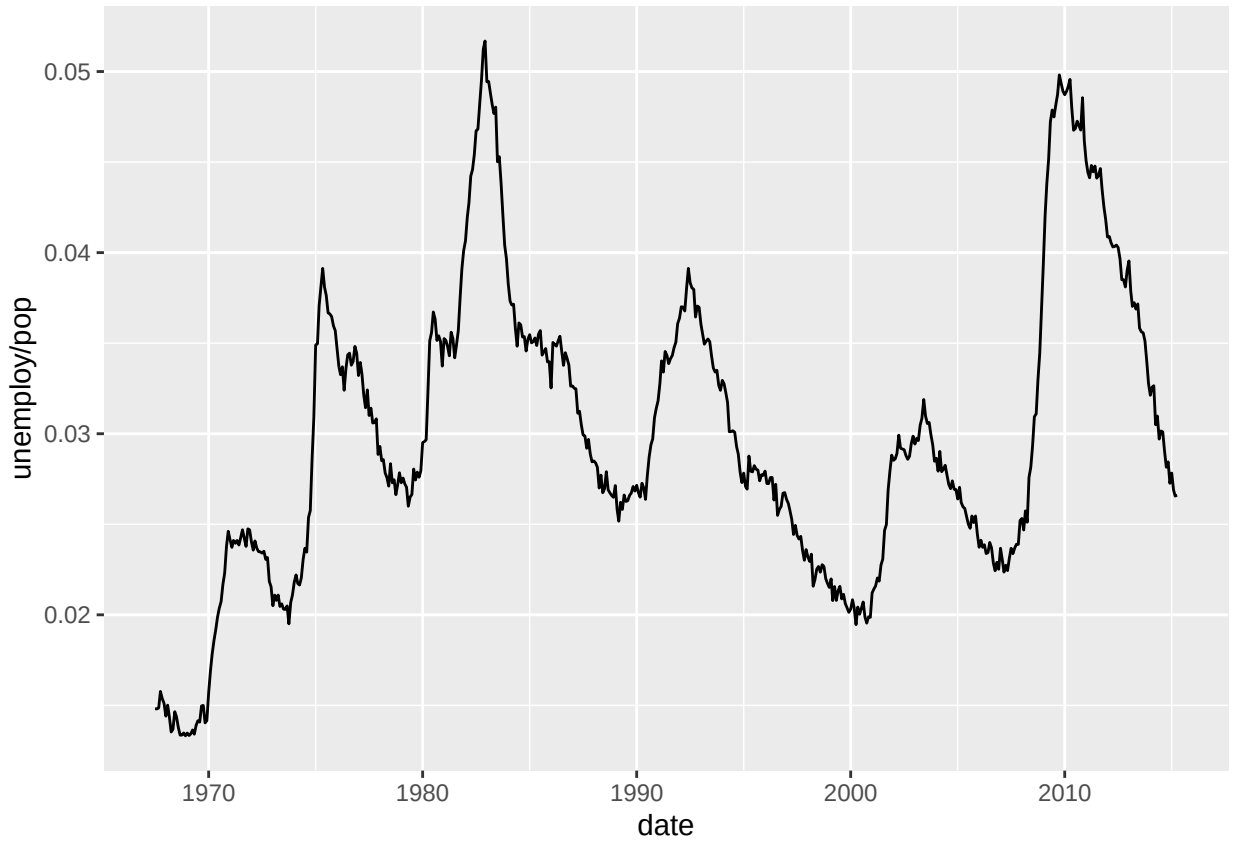
12. Rect elements
Access rectangles using arguments in the theme function and modify them using element_rect.

13. Hierarchical naming reflects inheritance rules
Although we have access to every item, we don't need to modify them individually. They inherent from eac

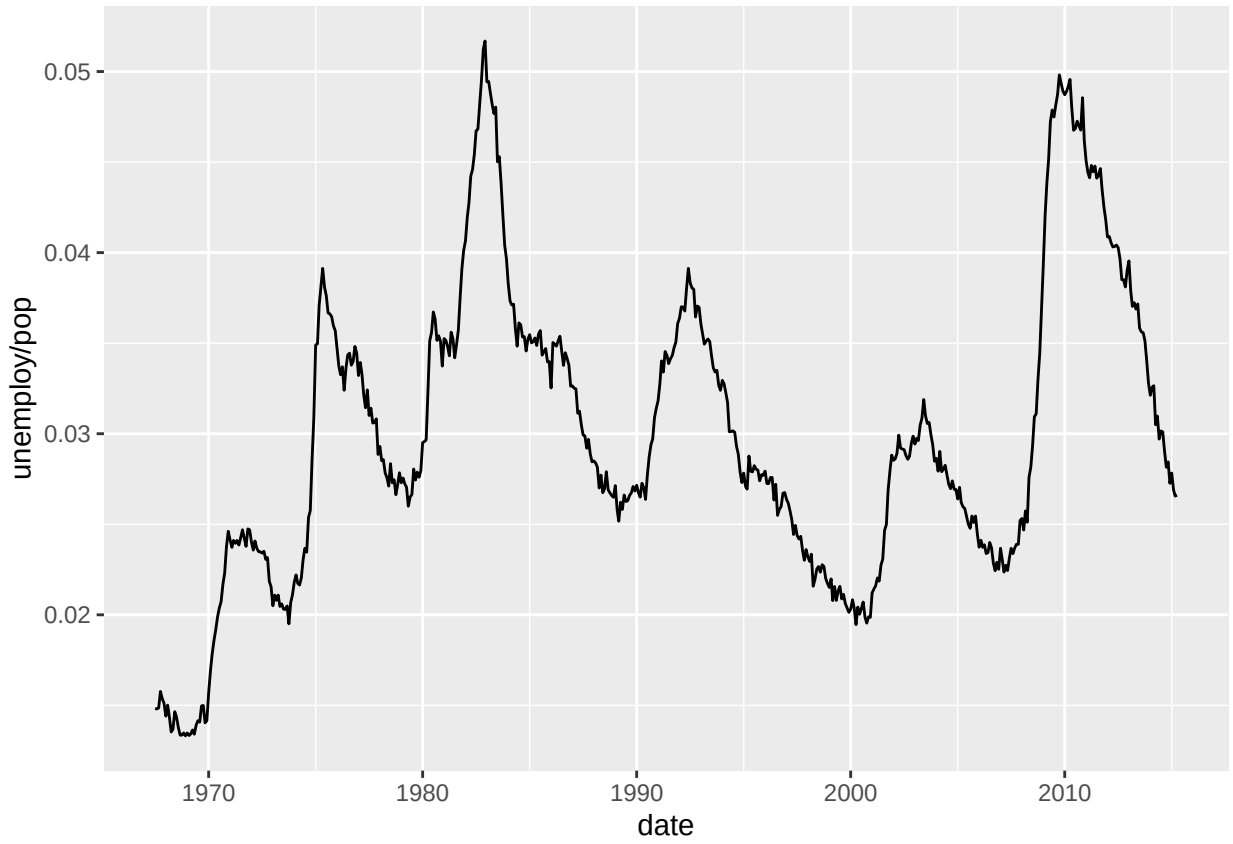
14. element_blank()
There is one other element function that we haven't discussed yet: element_blank. We can use this in a p

2. Moving the legend

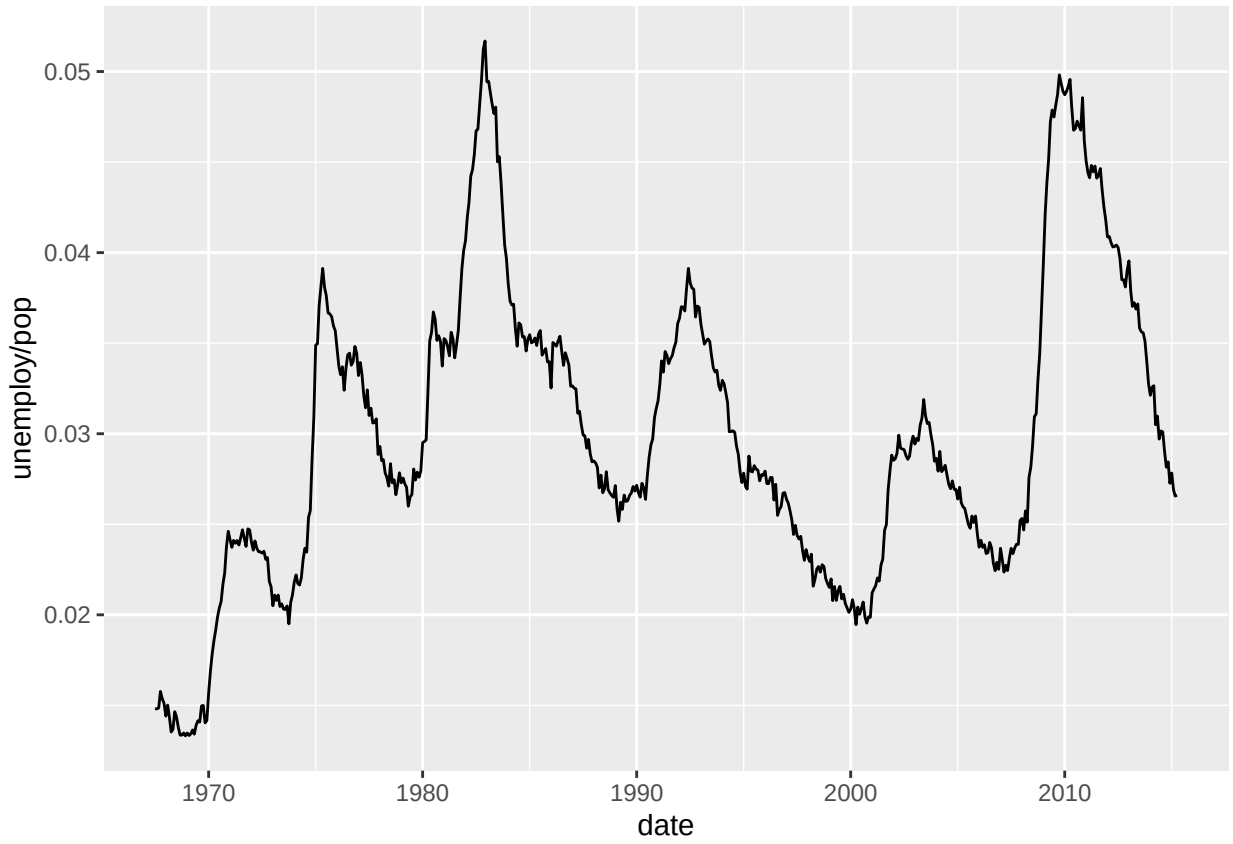
```
library(ggplot2)

# View the default plot
plt_prop_unemployed_over_time <- ggplot(economics, aes(x=date, y=unemploy/pop)) +
  geom_line()

# Remove legend entirely
plt_prop_unemployed_over_time +
  theme(legend.position ="none")
```



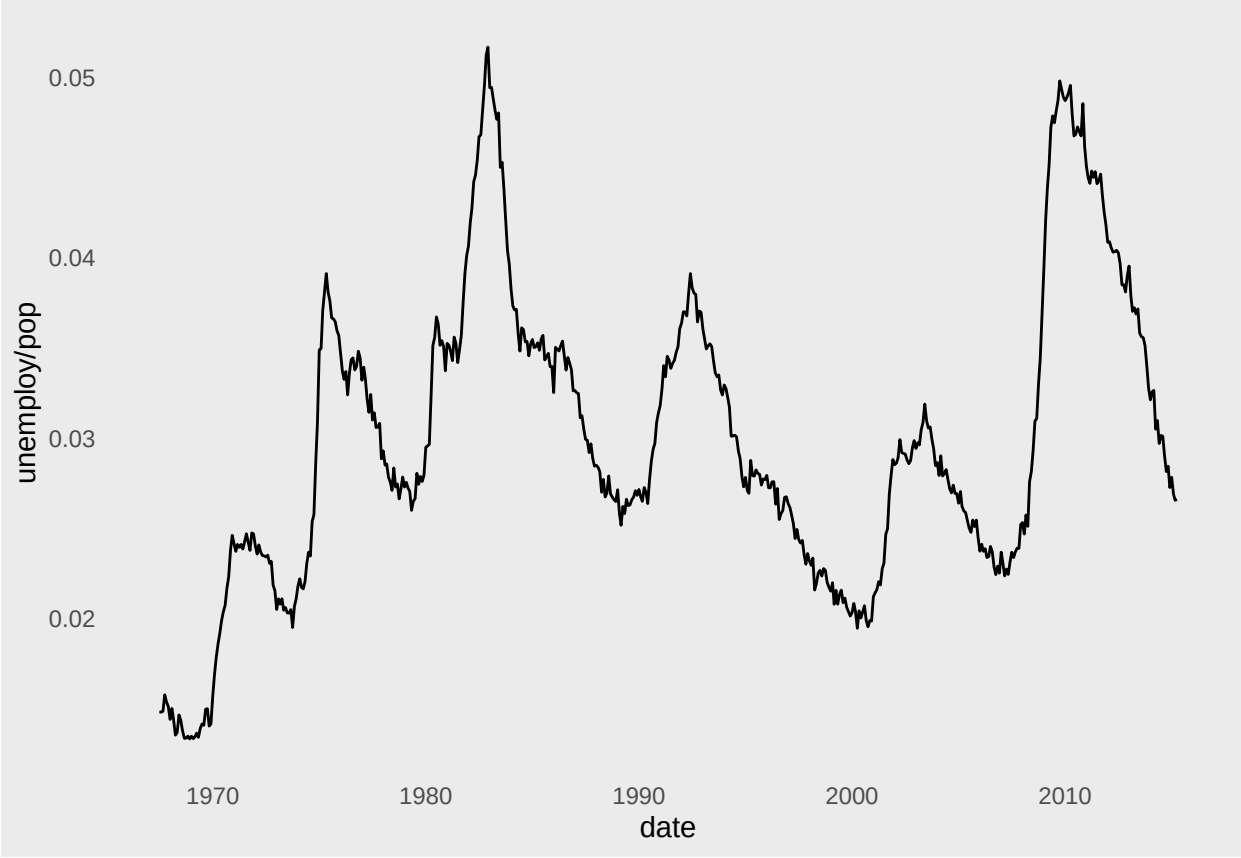

```
# Position the legend at the bottom of the plot
plt_prop_unemployed_over_time +
  theme(legend.position ="bottom")
```




```
# Position the legend inside the plot at (0.6, 0.1)
plt_prop_unemployed_over_time +
  theme(legend.position = c(0.6, 0.1))
```

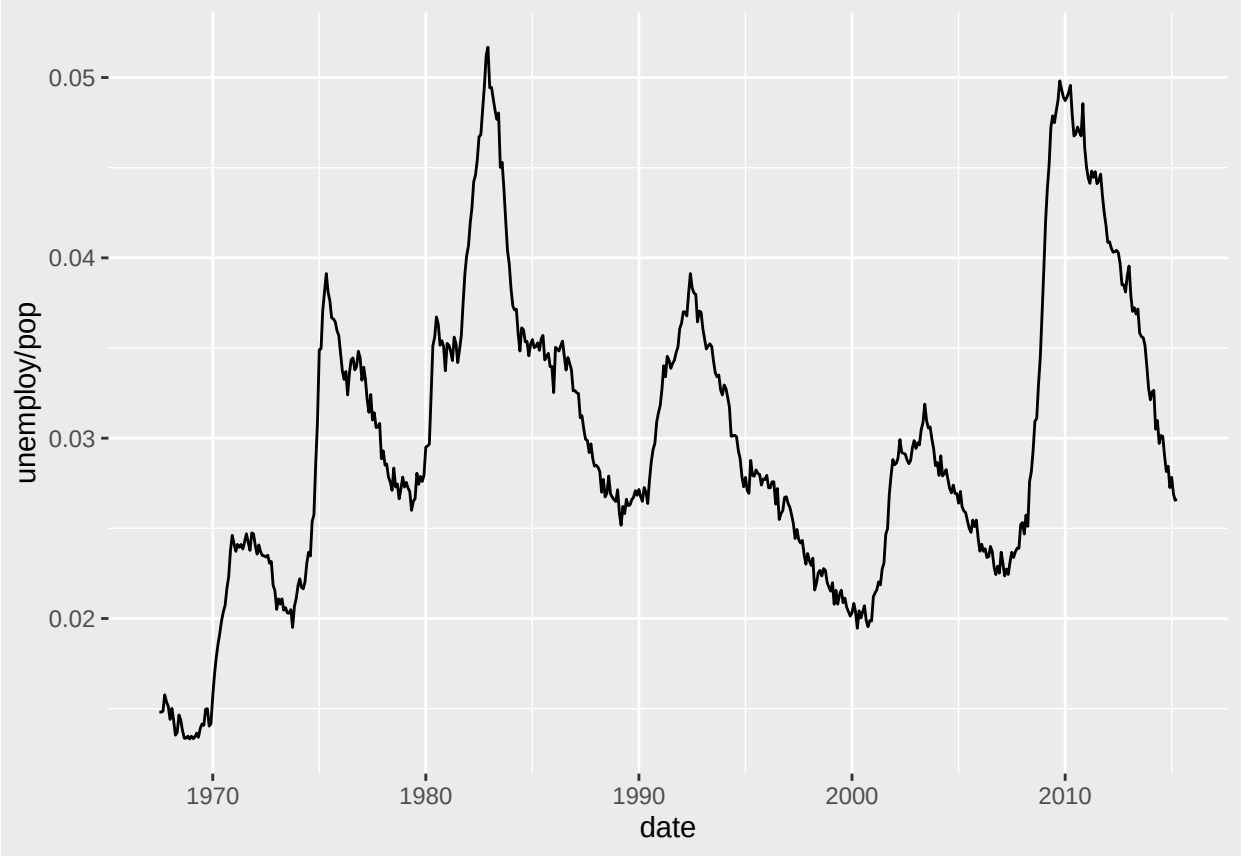



```
plt_prop_unemployed_over_time +
  theme(
    rect = element_rect(fill = "grey92"),
    legend.key = element_rect(color = NA),
    # Turn off axis ticks
    axis.ticks = element_blank(),
    # Turn off the panel grid
    panel.grid = element_blank()
  )
```



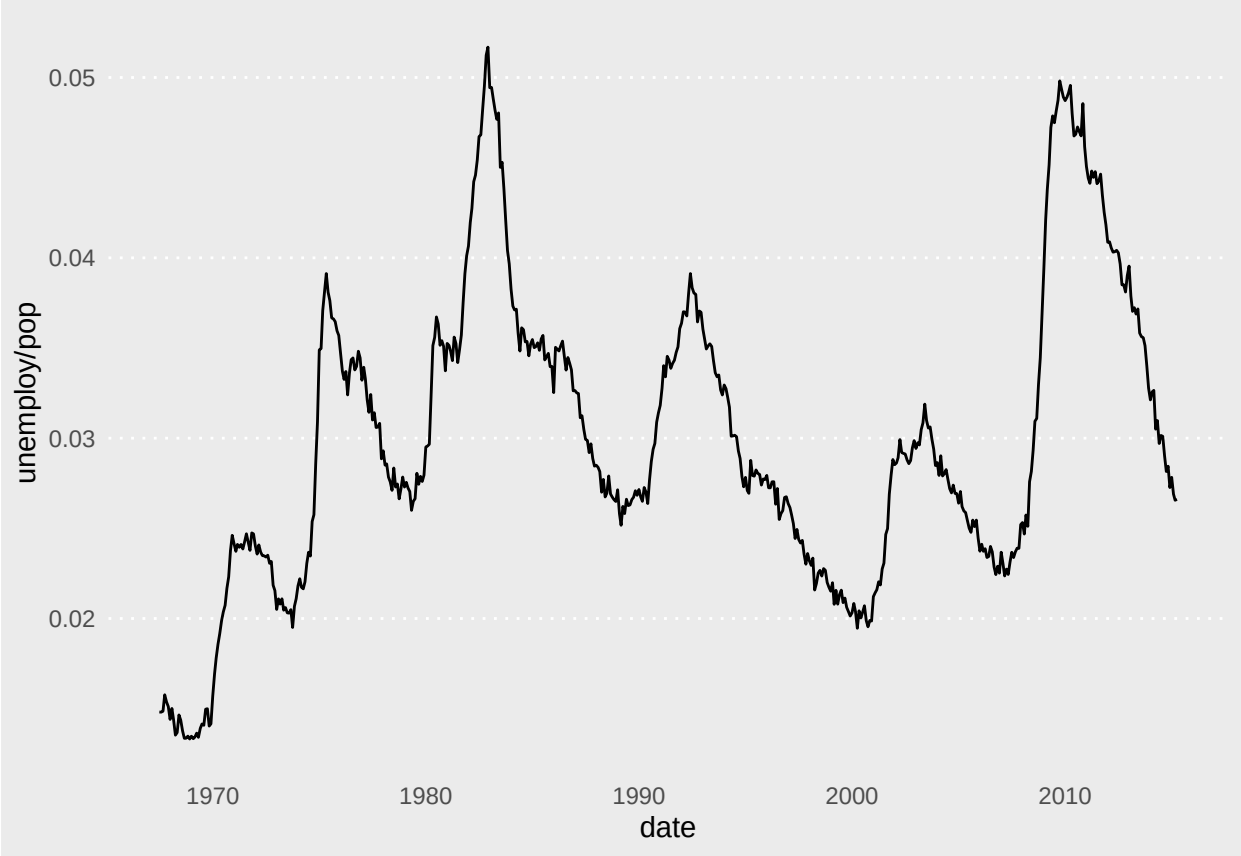
3. Modifying theme elements

```
plt_prop_unemployed_over_time +
  theme(
    # For all rectangles, set the fill color to grey92
    rect = element_rect(fill = "grey92"),
    # For the legend key, turn off the outline
    legend.key = element_rect(color = NA)
  )
```

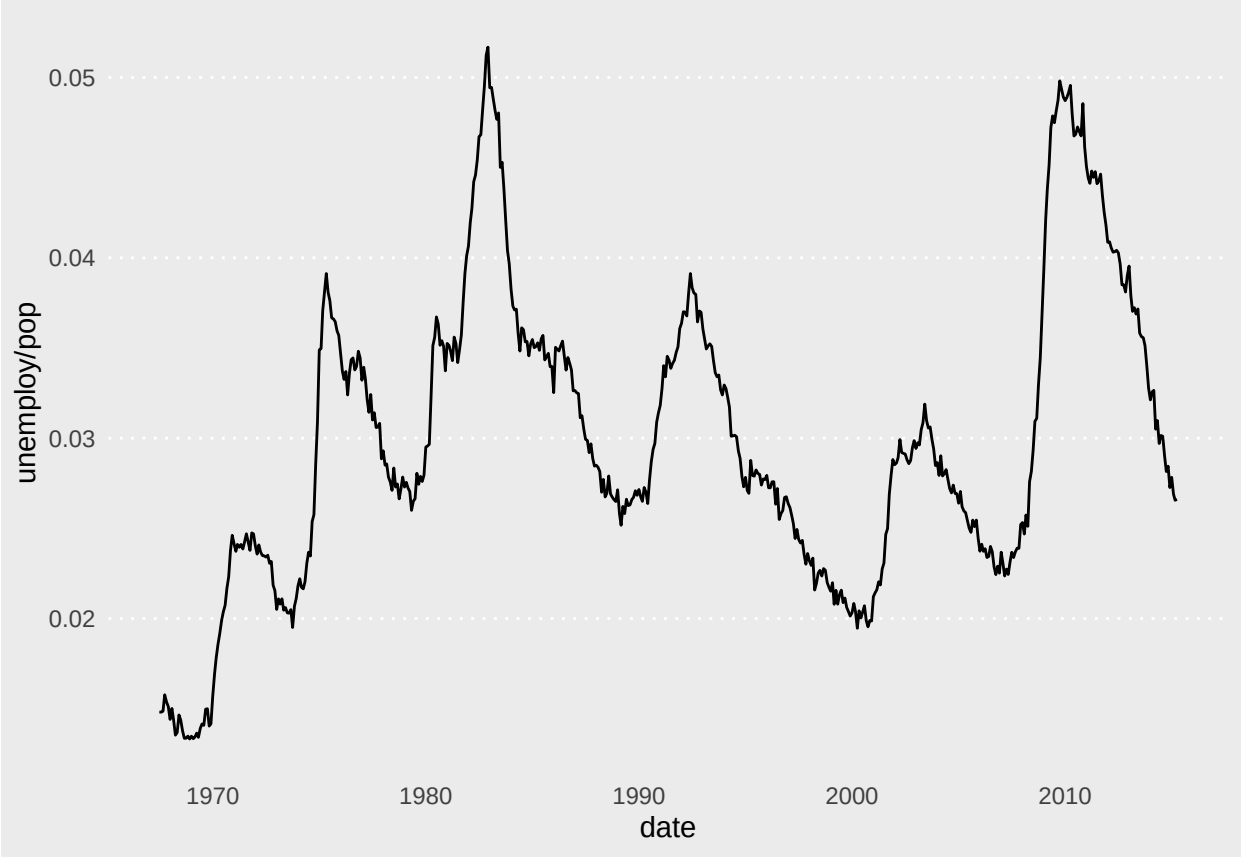



```
plt_prop_unemployed_over_time +
  theme(
    rect = element_rect(fill = "grey92"),
    legend.key = element_rect(color = NA),
    axis.ticks = element_blank(),
    panel.grid = element_blank(),
    # Add major y-axis panel grid lines back
    panel.grid.major.y = element_line(
      # Set the color to white
      color="white",
      # Set the size to 0.5
      size=0.5,
      # Set the line type to dotted
      linetype="dotted"
    )
  )
```

```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
```

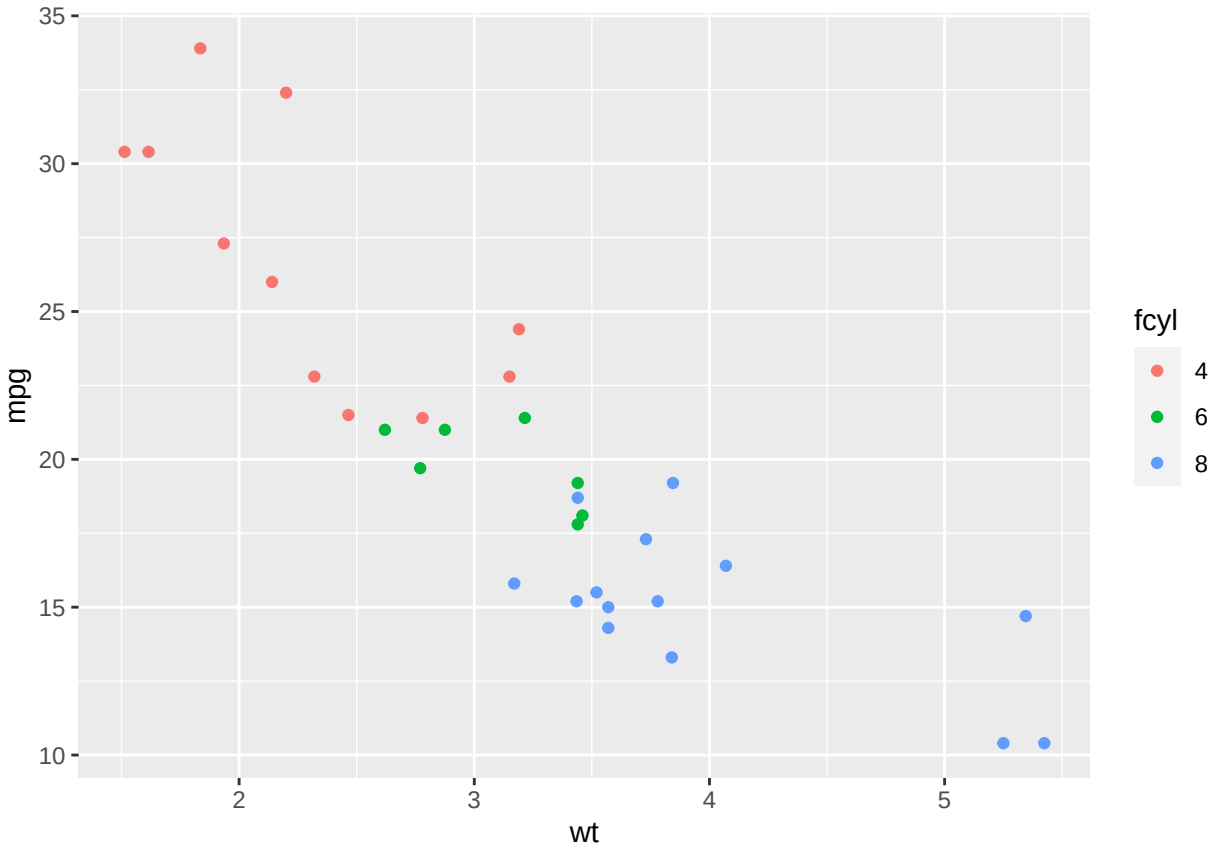



```
plt_prop_unemployed_over_time +
  theme(
    rect = element_rect(fill = "grey92"),
    legend.key = element_rect(color = NA),
    axis.ticks = element_blank(),
    panel.grid = element_blank(),
    panel.grid.major.y = element_line(
      color = "white",
      size = 0.5,
      linetype = "dotted"
    ),
    # Set the axis text color to grey25
    axis.text = element_text(color = "grey25"),
    # Set the plot title font face to italic and font size to 16
   plot.title = element_text(face = "italic", size =16)
  )
```

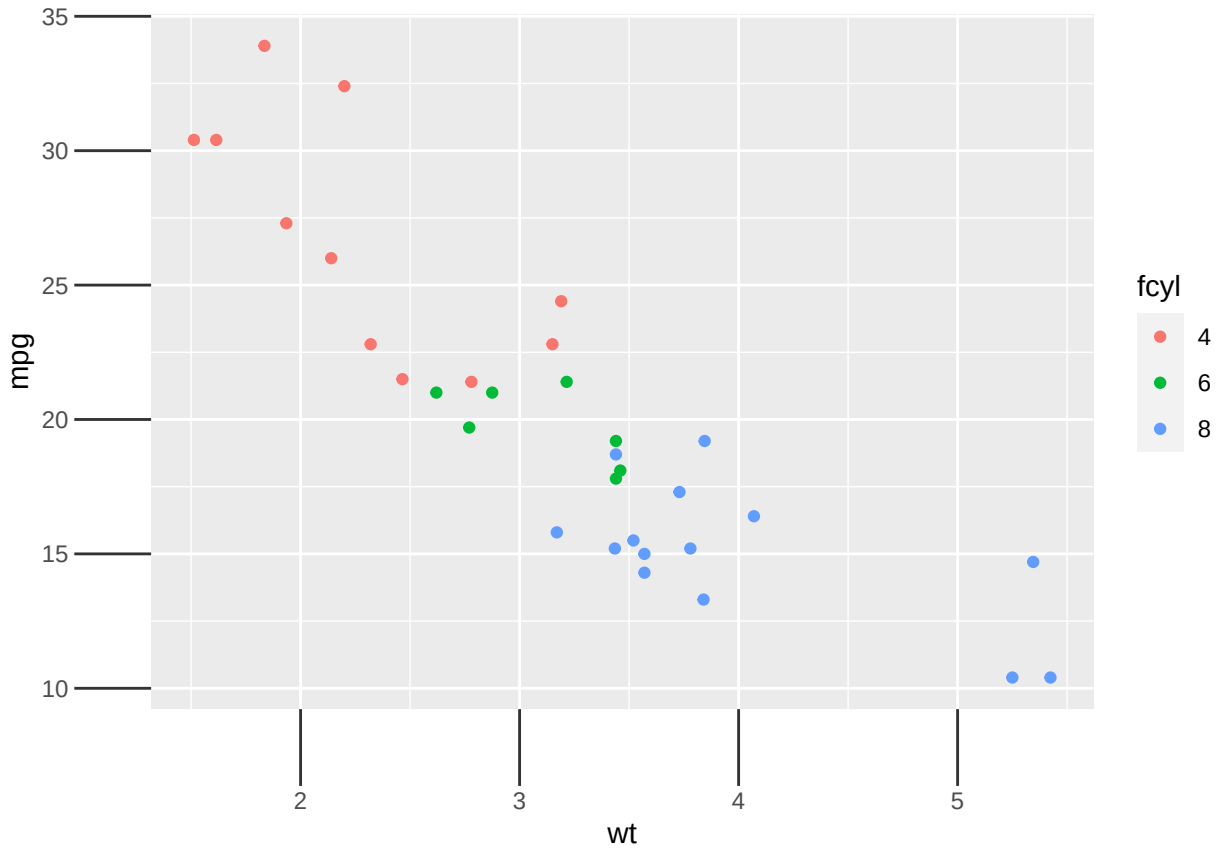


4. Modifying whitespace

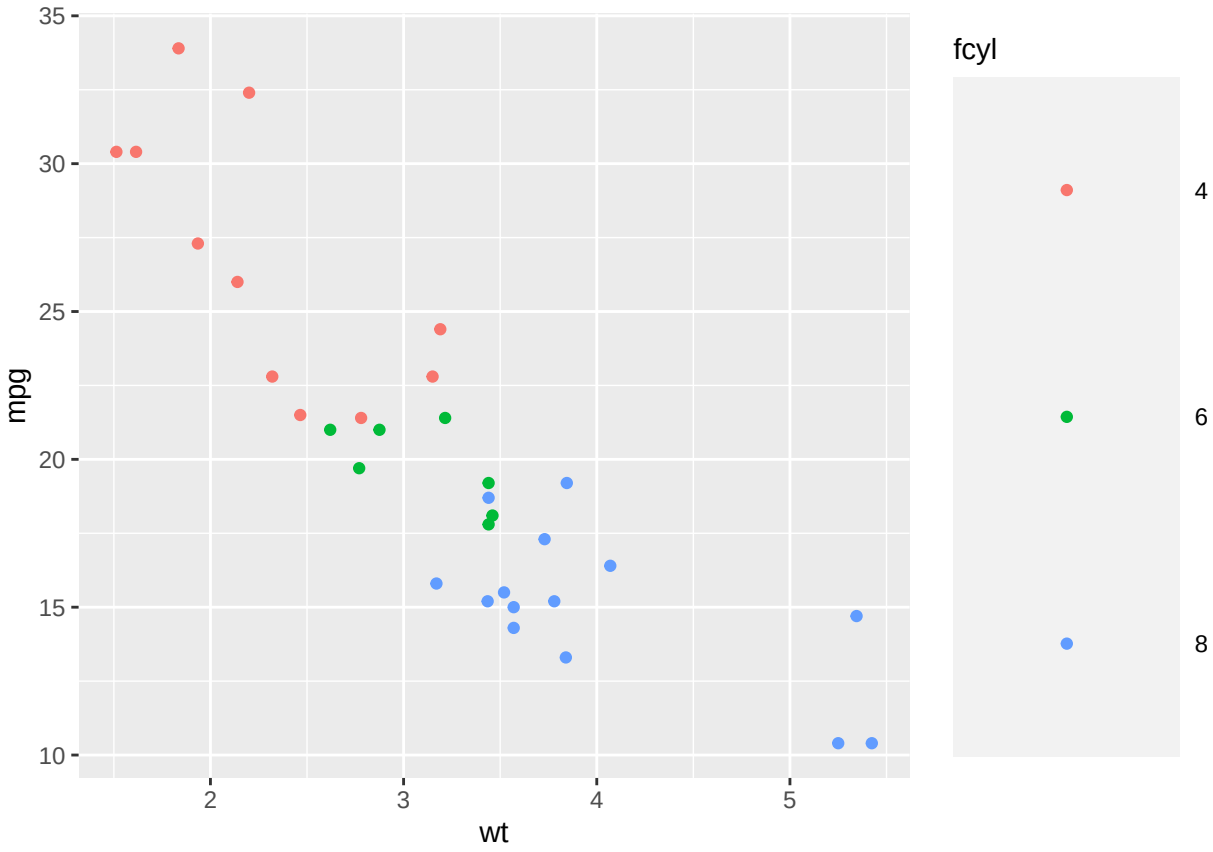
```
# from previous exercise
plt_mpg_vs_wt_by_cyl <- ggplot(mtcars, aes(x=wt, y=mpg, color=fcyl)) +
  geom_point()

plt_mpg_vs_wt_by_cyl
```



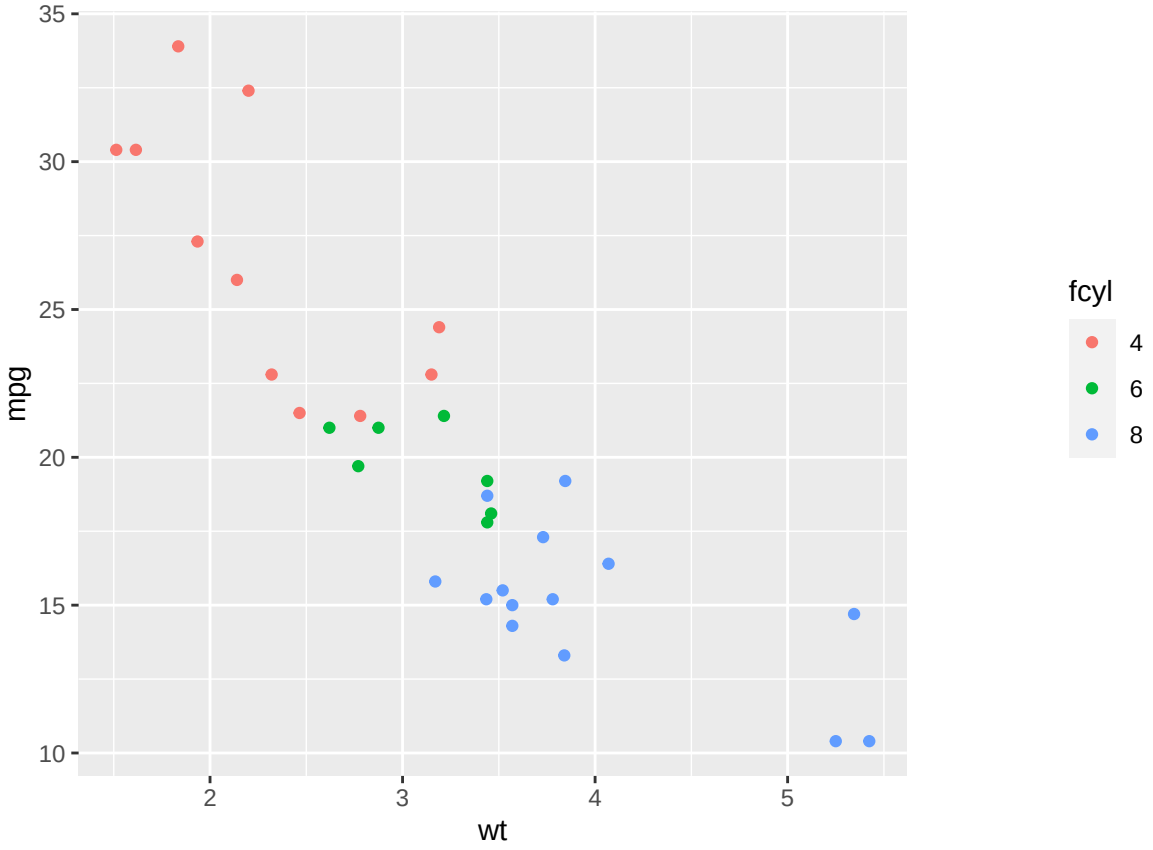

```
plt_mpg_vs_wt_by_cyl +
  theme(
    # Set the axis tick length to 2 lines
    axis.ticks.length = unit(2, "lines")
  )
```



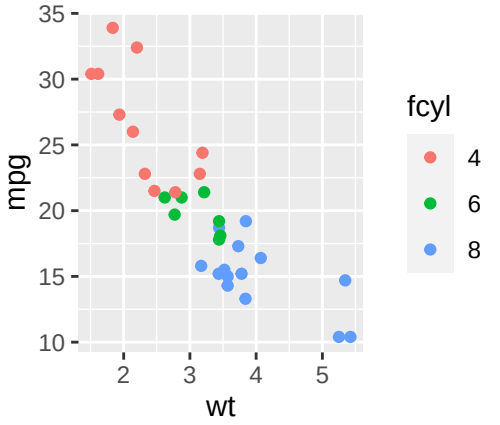

```
plt_mpg_vs_wt_by_cyl +
  theme(
    # Set the legend key size to 3 centimeters
    legend.key.size = unit(3, "cm")
  )
```




```
plt_mpg_vs_wt_by_cyl +
  theme(
    # Set the legend margin to (20, 30, 40, 50) points
    legend.margin = margin(20, 30, 40, 50, "pt")
  )
```




```
plt_mpg_vs_wt_by_cyl +
  theme(
    # Set the plot margin to (10, 30, 50, 70) millimeters
    plot.margin = margin(10, 30, 50, 70, "mm")
  )
```



5. Theme flexibility

```
1. Theme flexibility
In the last video and exercises

2. Ways to use themes
we saw how to fine-tune every part of our plot using the theme layer.
```

3. Ways to use themes
There are a few other ways of changing theme elements, so let's take a look. We'll begin with defining

4. Defining theme objects
If you're using many plots within a presentation or publication, you'll want to have consistency in you

5. Defining theme objects
To see how this works let's return to a plot we've already seen in the last video. For convenience, we'

6. Defining theme objects
We can adjust specific theme arguments to get the desired plot style. Here, I've changed the font famil

7. Defining theme objects
The first method in automating this process is to save our layer as an object. Here we're just going to

8. Reusing theme objects
Just as we've seen throughout the course, we can add individual layers to any ggplot object. This means

9. Reusing theme objects
Remember this histogram of the iris Sepal widths?

10. Reusing theme objects
Now it has the same style as our scatter plot, without having to retype the whole theme layer.

11. Reusing theme objects
But let's say that on occasion I wanted to modify some other specific elements of a plot. Not a problem

12. Ways to use themes
So far we just used our theme as an object itself. This is a pretty flexible way of working with theme,

13. Using built-in themes
Built-in theme functions begin with theme_*. theme classic is my go-to template for great publication-q

14. Using built-in themes
Of course we can always modify any specific element, as we've already seen.

15. Ways to use themes
There are also packages with pre-defined themes, like the ggthemes package.

16. The ggthemes package
The tufte theme mimics Tufte's classic style, which removes all non-data ink and sets the font to a ser

17. Ways to use themes
Finally, we can update and set the default theme

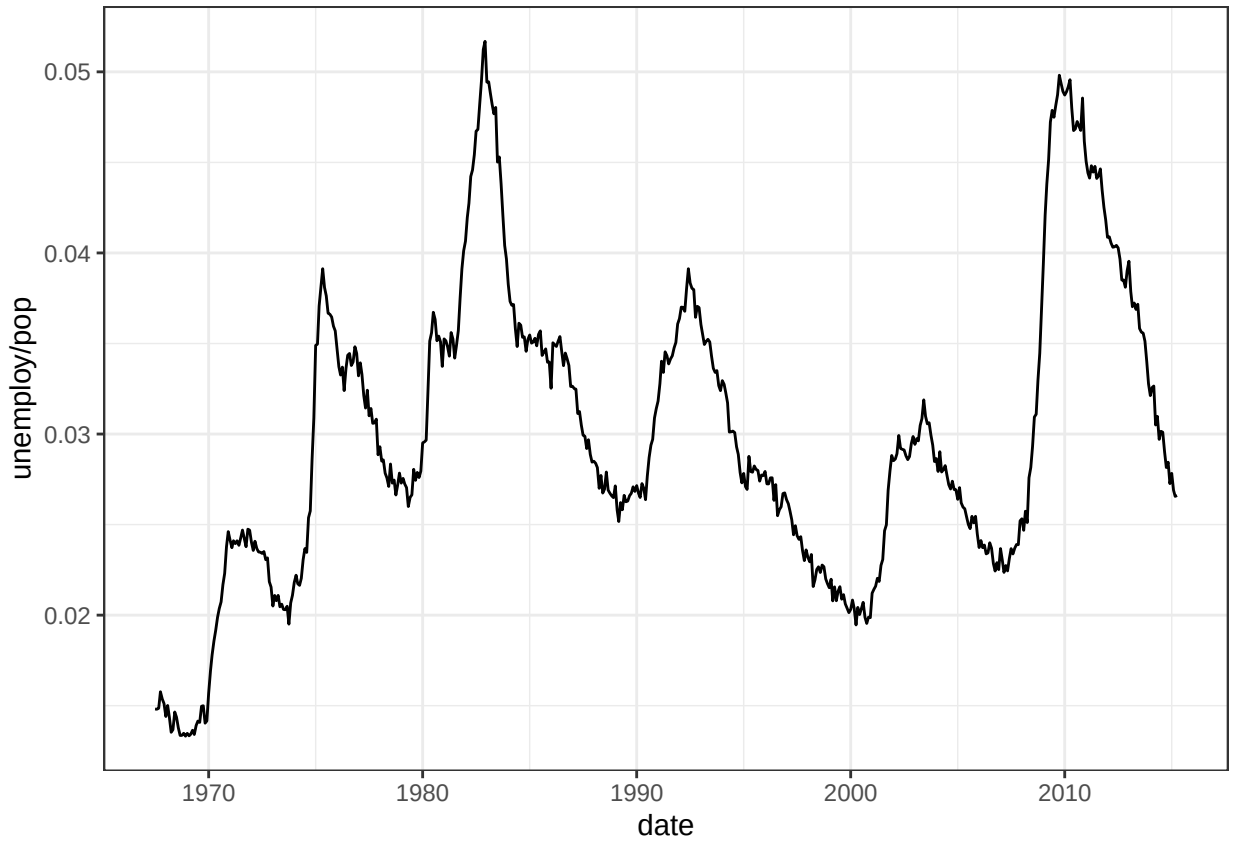
18. Updating themes
Updating is done with the theme_update function. This function behaves differently from a typical funct

19. Updating themes
Now, all plots will automatically have the same theme.

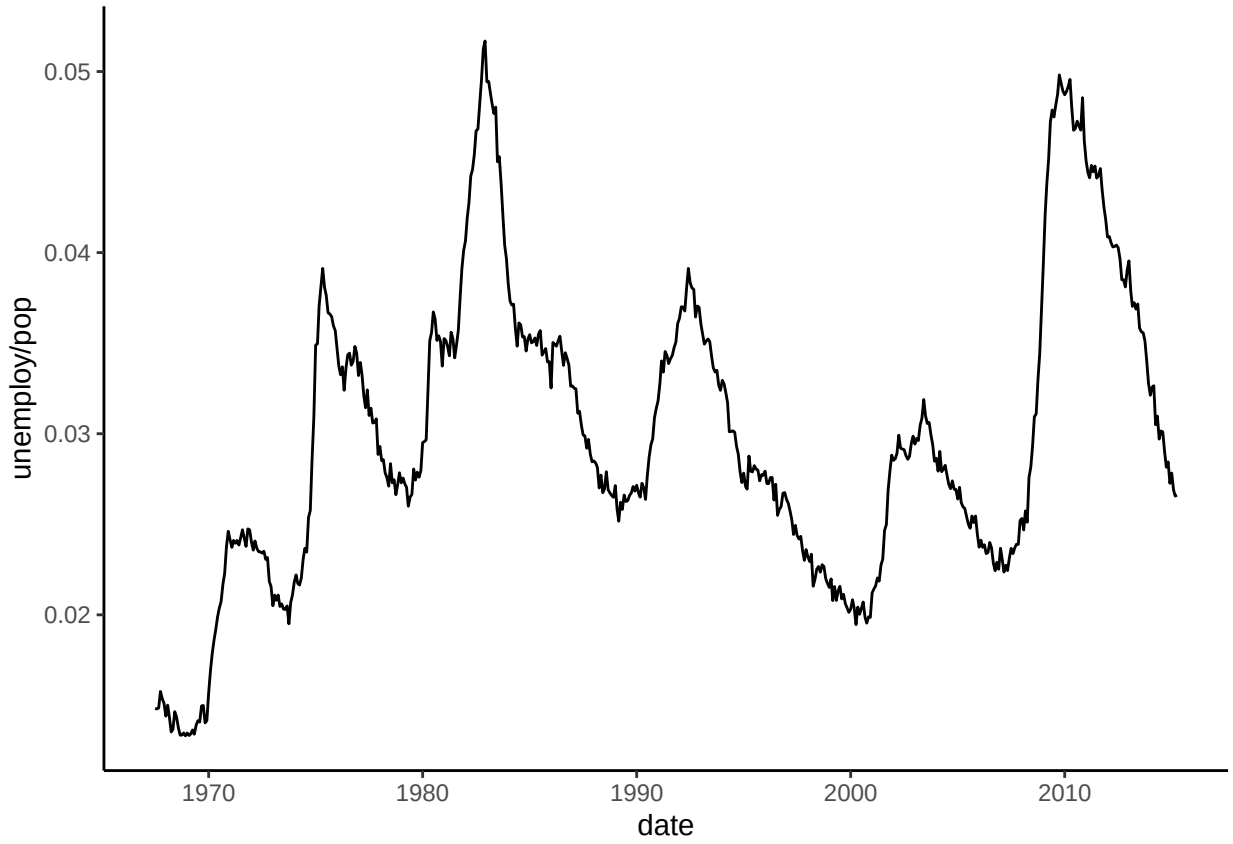
20. Setting themes
We can set a complete theme object, like original, using the theme_set function.

6. Built-in themes

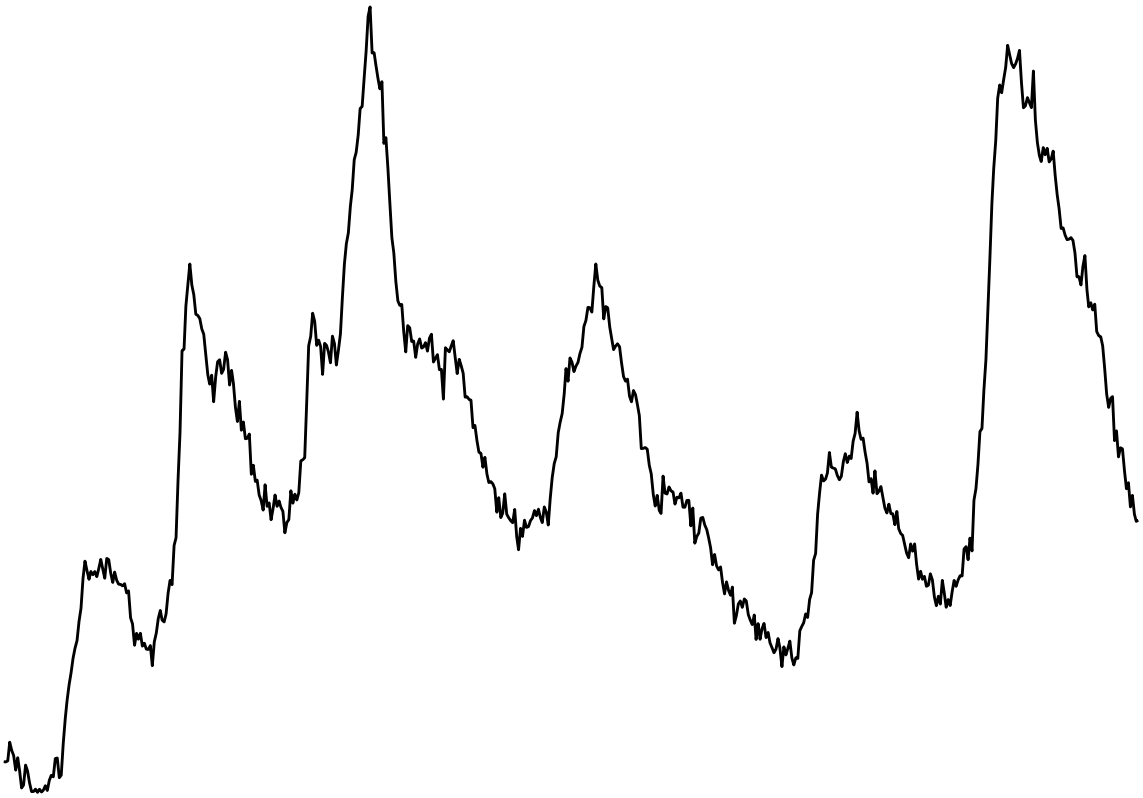
```
# Add a black and white theme
plt_prop_unemployed_over_time +
  theme_bw()
```




```
# Add a classic theme
plt_prop_unemployed_over_time +
  theme_classic()
```

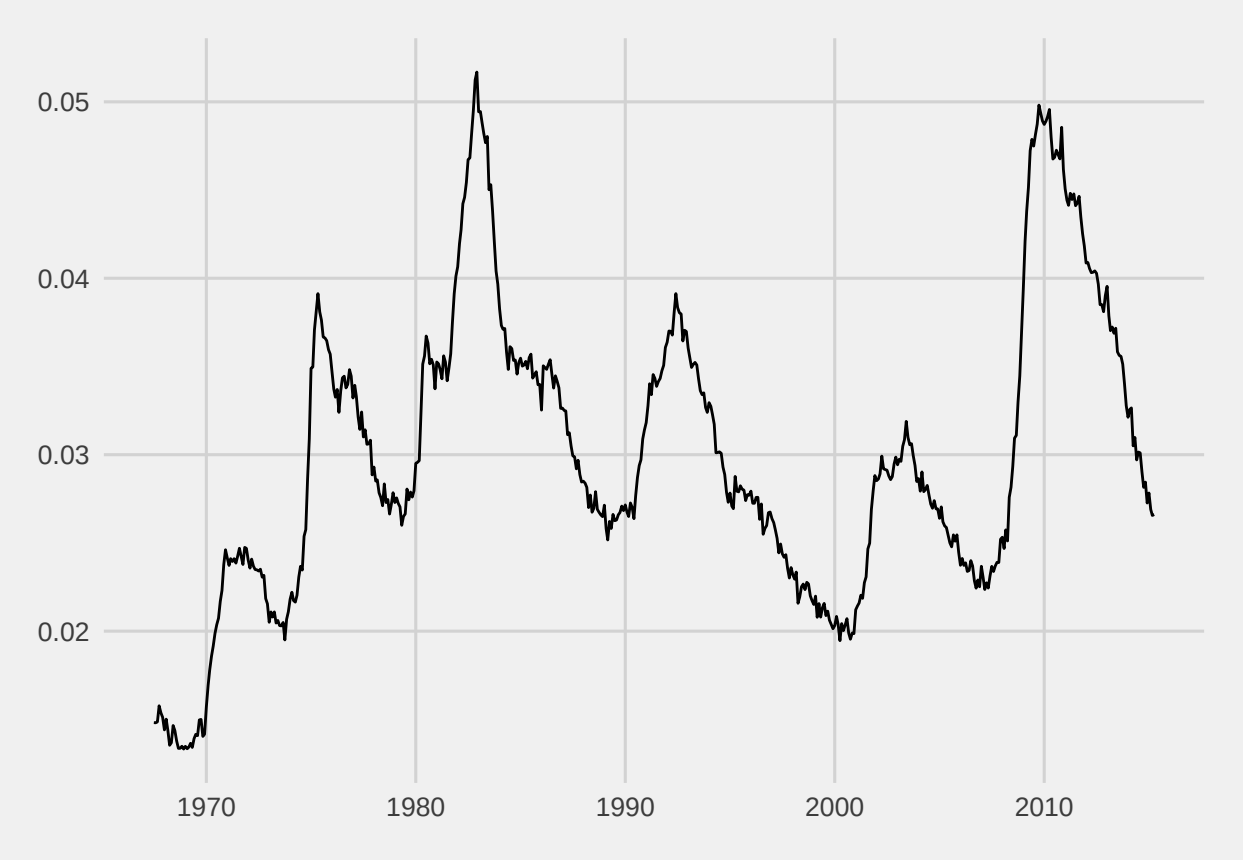



```
# Add a void theme
plt_prop_unemployed_over_time +
  theme_void()
```

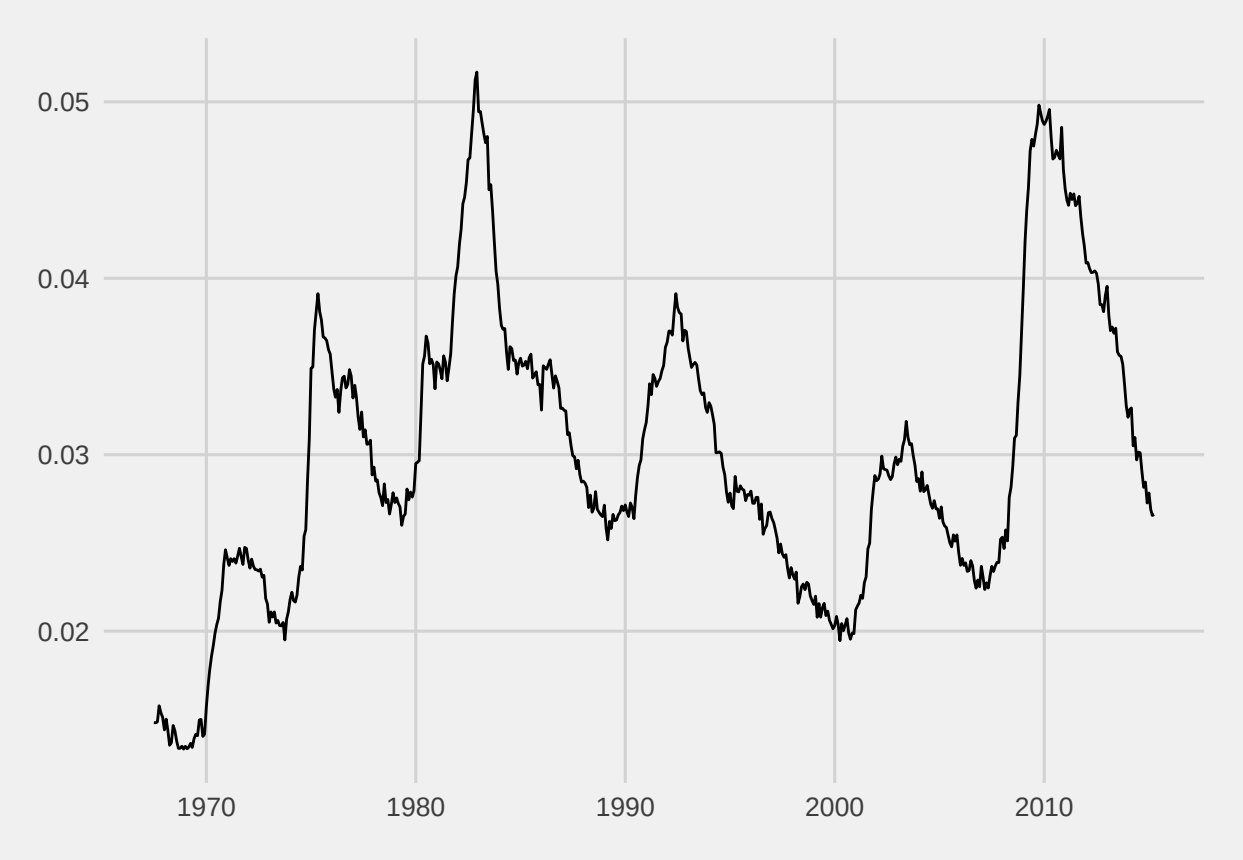


7. Exploring ggthemes

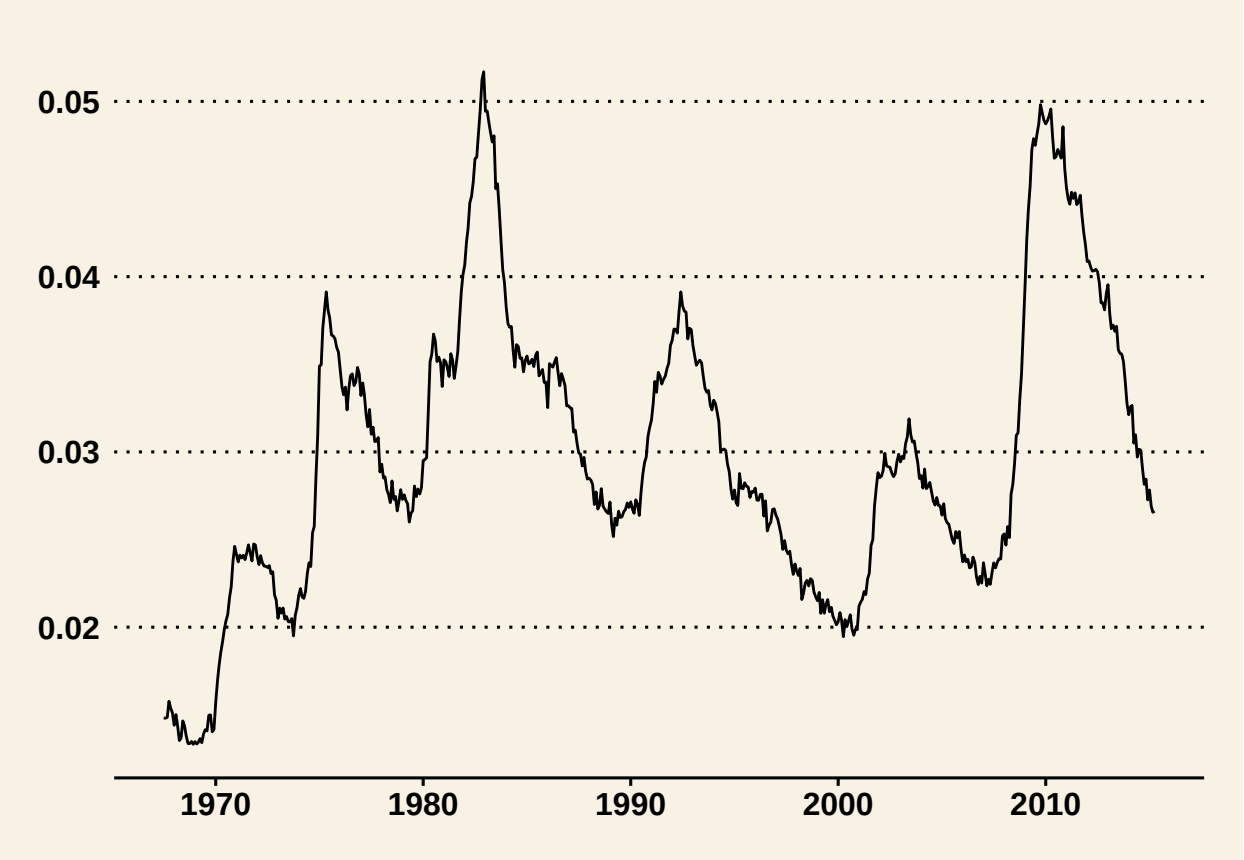
```
library(ggthemes)
# Use the fivethirtyeight theme
plt_prop_unemployed_over_time +
  theme_fivethirtyeight()
```




```
# Use the fivethirtyeight theme
plt_prop_unemployed_over_time +
  theme_fivethirtyeight()
```

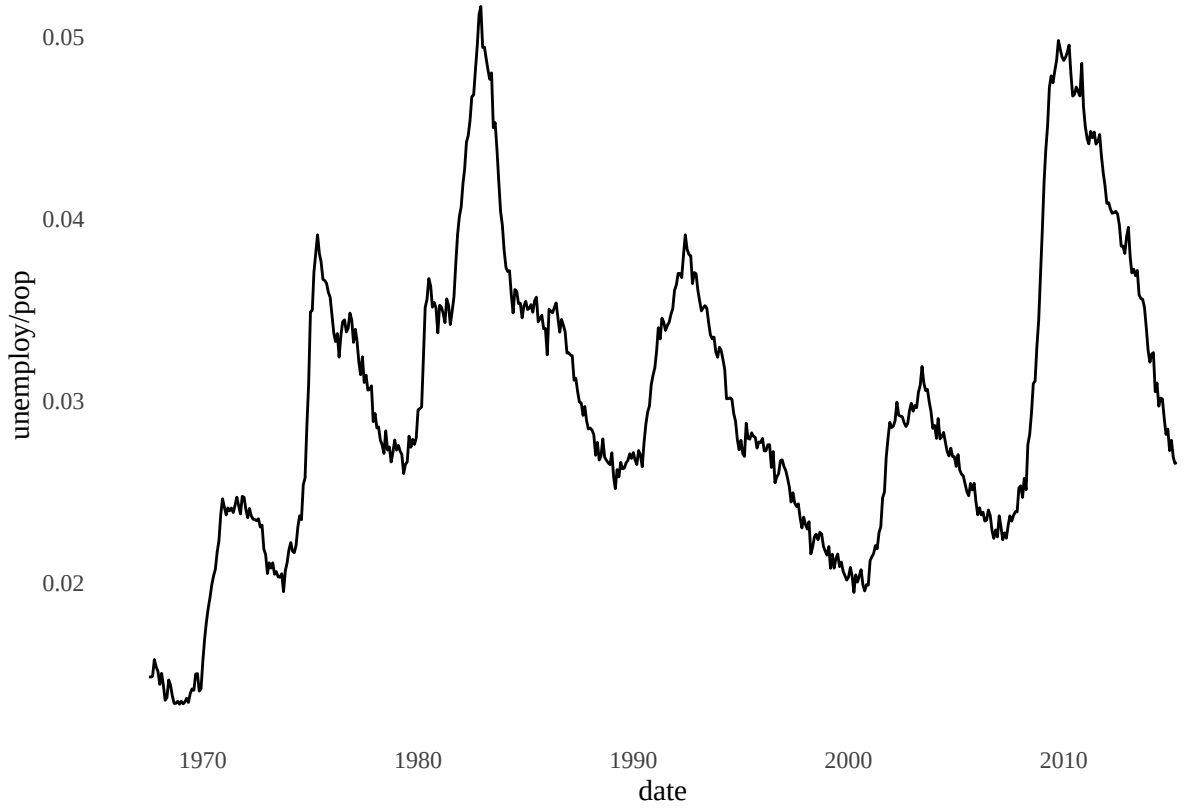



```
# Use the Wall Street Journal theme
plt_prop_unemployed_over_time +
  theme_wsj()
```

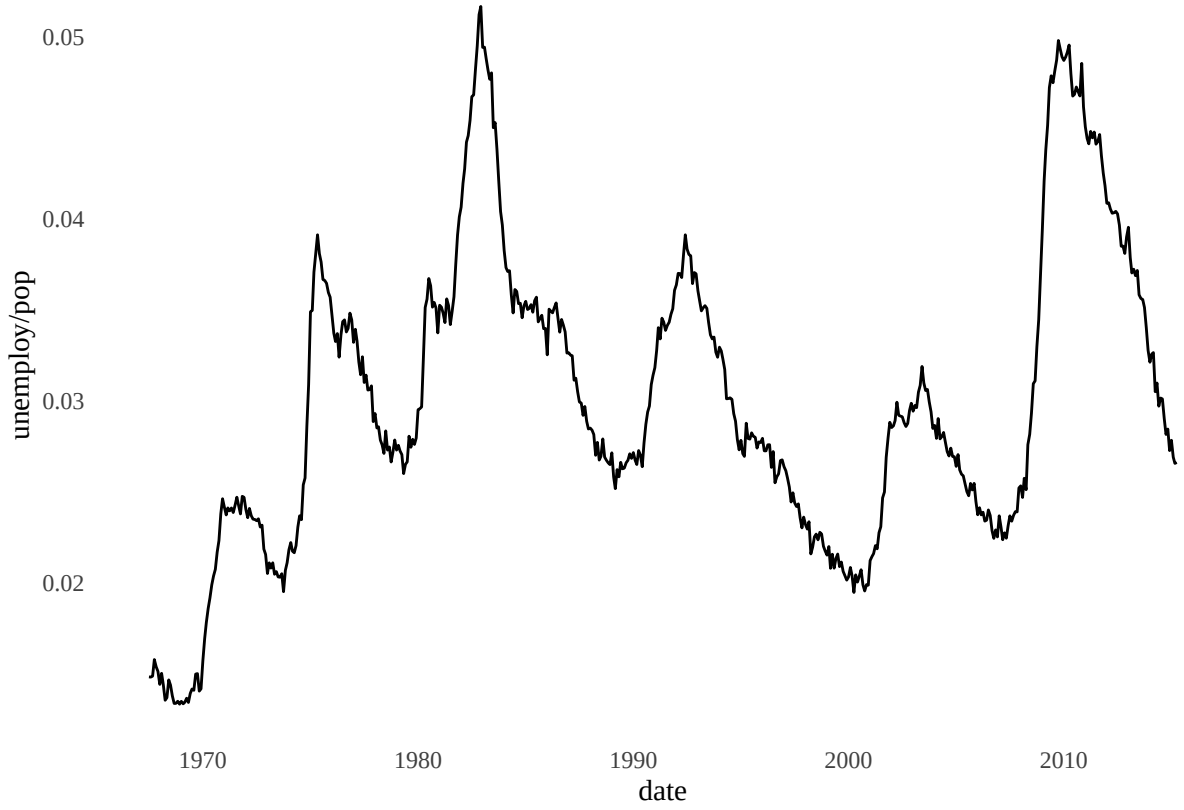


8. Setting themes

```
library(ggplot2)
library(ggthemes)

# Save the theme as theme_recession
theme_recession <- theme(
  rect = element_rect(fill = "grey92"),
  legend.key = element_rect(color = NA),
  axis.ticks = element_blank(),
  panel.grid = element_blank(),
  panel.grid.major.y = element_line(color = "white", size = 0.5, linetype = "dotted"),
  axis.text = element_text(color = "grey25"),
  plot.title = element_text(face = "italic", size = 16),
  legend.position = c(0.6, 0.1)
)

# Combine the Tufte theme with theme_recession
theme_tufte_recession <- theme_tufte() + theme_recession

# Add the Tufte recession theme to the plot
plt_prop_unemployed_over_time + theme_tufte_recession
```

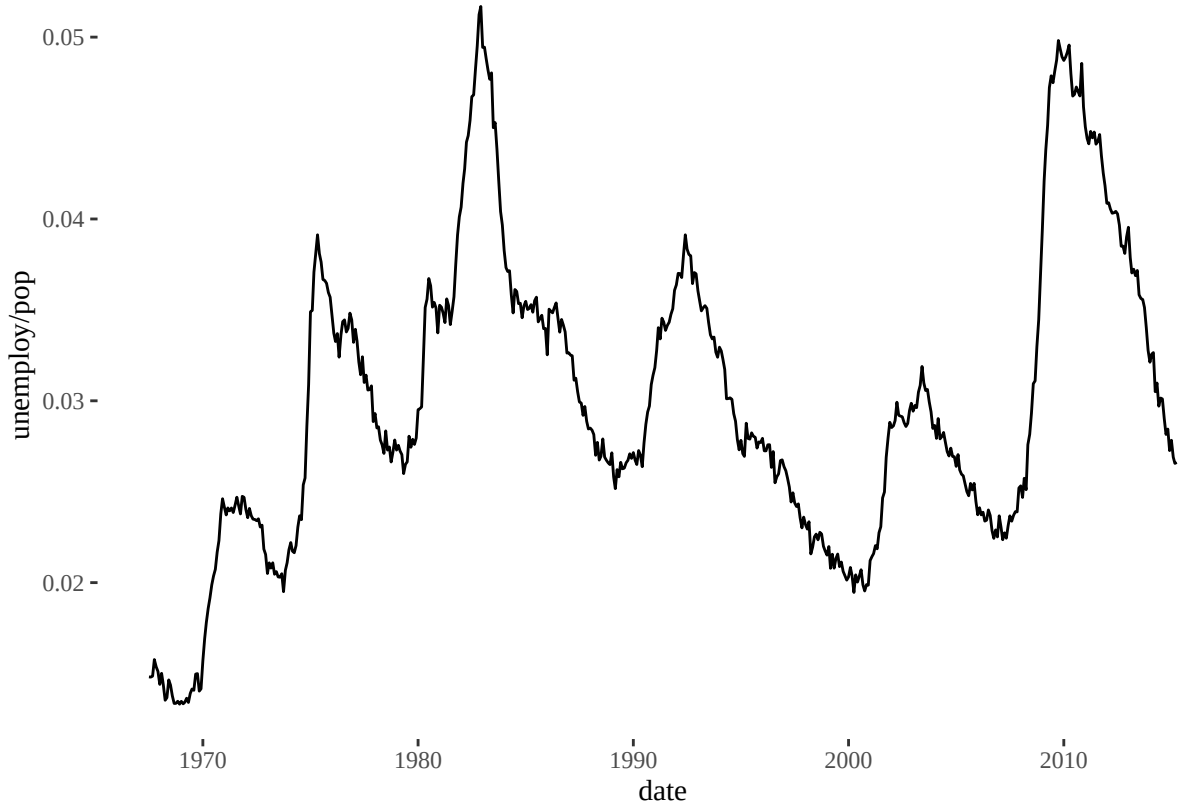



```
# Set theme_tufte_recession as the default theme
theme_set(theme_tufte_recession)

# Draw the plot (without explicitly adding a theme)
plt_prop_unemployed_over_time
```

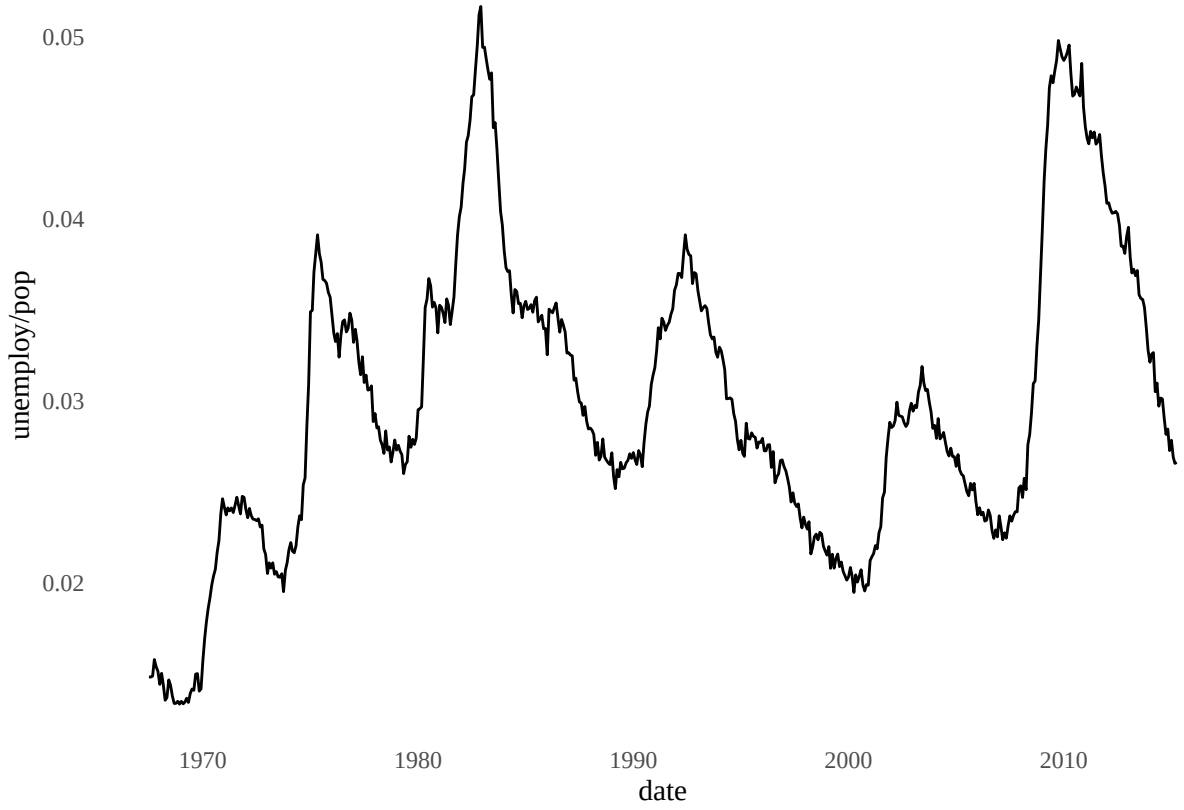


9. Publication-quality plots

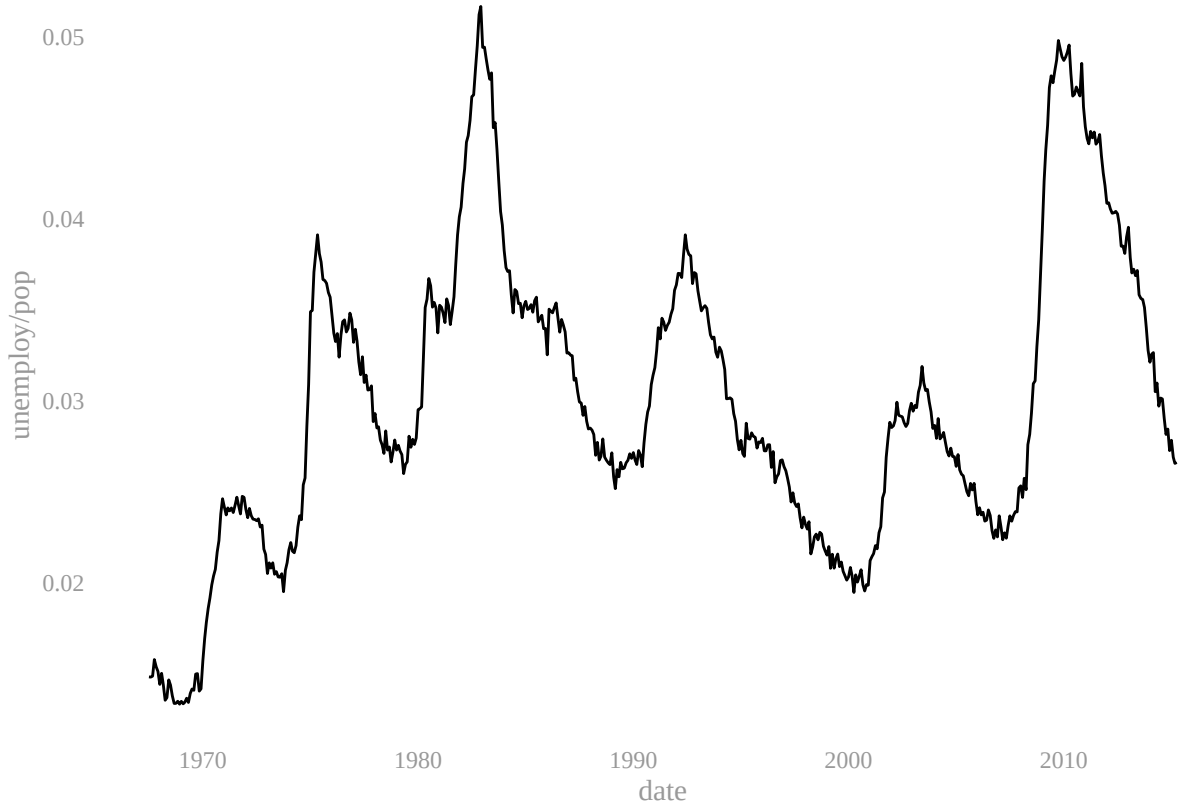
```
plt_prop_unemployed_over_time +
  # Add Tufte's theme
  theme_tufte()
```



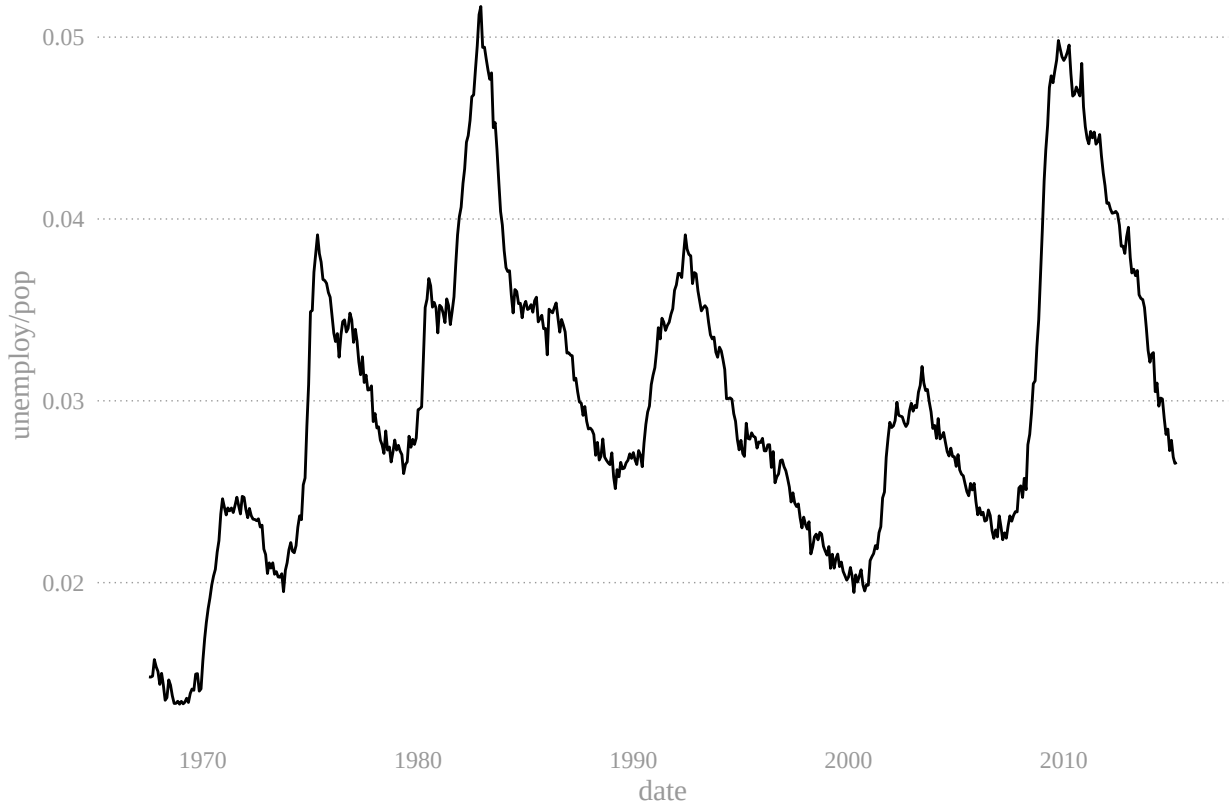

```
plt_prop_unemployed_over_time +
  theme_tufte() +
  # Add individual theme elements
  theme(
    # Turn off the legend
    legend.position = "none",
    # Turn off the axis ticks
    axis.ticks = element_blank()
  )
```




```
plt_prop_unemployed_over_time +
  theme_tufte() +
  theme(
    legend.position = "none",
    axis.ticks = element_blank(),
    # Set the axis title's text color to grey60
    axis.title = element_text(color = "grey60"),
    # Set the axis text's text color to grey60
    axis.text = element_text(color = "grey60")
  )
```




```
plt_prop_unemployed_over_time +
  theme_tufte() +
  theme(
    legend.position = "none",
    axis.ticks = element_blank(),
    axis.title = element_text(color = "grey60"),
    axis.text = element_text(color = "grey60"),
    # Set the panel gridlines major y values
    panel.grid.major.y = element_line(
      # Set the color to grey60
      color = "grey60",
      # Set the size to 0.25
      size = 0.25,
      # Set the linetype to dotted
      linetype = "dotted"
    )
  )
```



10. Effective explanatory plots

```
1. Effective explanatory plots
For our last exercises, I want to go through an example of producing explanatory plots in an info viz s

2. Our goal, an effective explanatory plot
These plots tend to have both a small number of observations and variables, have embellishments and typ

3. Complete data
1 We would begin with our complete data set, which contains three variables for 142 countries.
4. First exploratory plots - distributions
Our first exploratory plot would probably be a histogram, which isn't a bad choice. Recall that we have

5. First exploratory plots - distributions
An alternative would be to arrange the data according life expectancy and plot that as an index, which a

6. First exploratory plots - distributions
This has the advantage that we can color each point according to continent. This is already a quite inf

7. Our data
In this form we only have 20 observations, the top 10 and bottom 10 observations.

8. life expectancy plot
Here, I'd map the country to the y axis, so that it's easy to read, I mapped life expectancy onto both

9. Use intuitive and attractive geoms
The line segments add some perspective and is sometimes referred to as a lollipop plot when used with p
```

10. Add text labels to your plot
Typically, we're happy to just read a value from the axis, but adding the actual value using a geom_tex

11. Use appropriate scales
Next, I'd clean up the scales, using an intuitive color palette, removing unnecessary buffering, and cha

12. Add useful titles and citations
Titles and captions help to make the plot complete, if it will be viewed alone.

13. Remove non-data ink
And of course, removing non-data ink makes for a great looking plot. Notice that I removed the x and y a

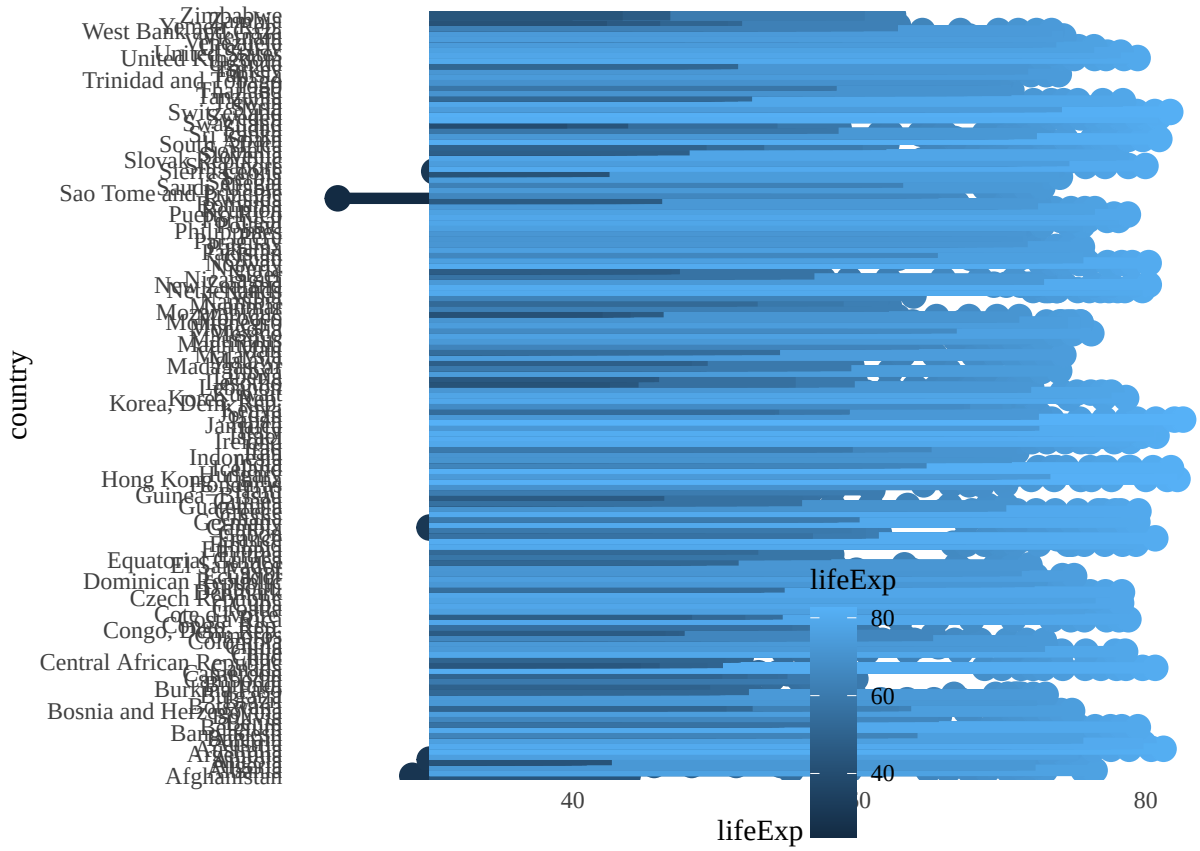
14. Add threshold lines
Adding a threshold line helps to orientate the viewer. Here, it's the global mean from 2007.

15. Add informative text
Of course, it's also helpful to label the threshold line. We'll do this with the annotate function, whi

16. Add embellishments
For example, another geom we haven't seen yet is geom_curve, for drawing curved lines. This is really g

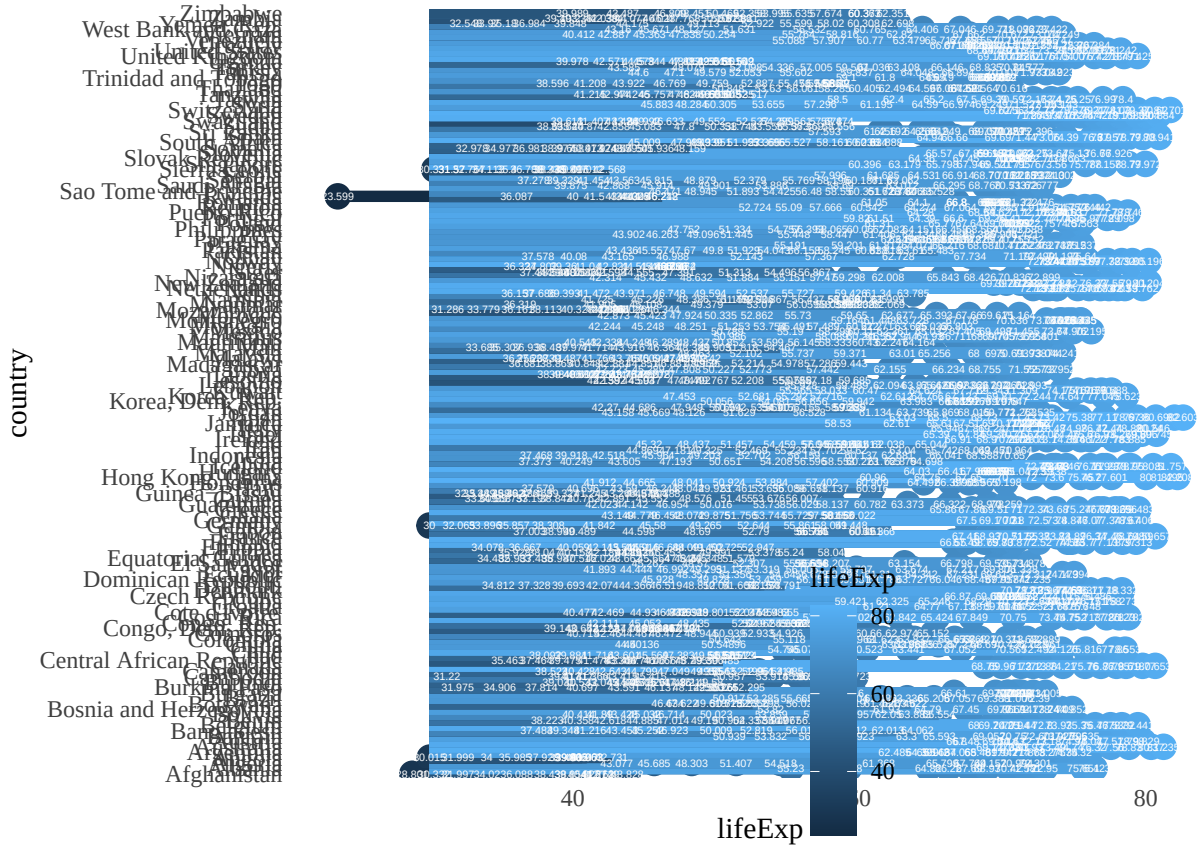
11. Using geoms for explanatory plots

```
# install.packages("gapminder")
library(ggplot2)
library(gapminder)
data(gapminder)
gm2007 <- gapminder

# Add a geom_segment() layer
ggplot(gm2007, aes(x = lifeExp, y = country, color = lifeExp)) +
  geom_point(size = 4) +
  geom_segment(aes(xend = 30, yend = country), linewidth = 2)
```




```
# Add a geom_text() layer
ggplot(gm2007, aes(x = lifeExp, y = country, color = lifeExp)) +
  geom_point(size = 4) +
  geom_segment(aes(xend = 30, yend = country), size = 2) +
  geom_text(aes(label = lifeExp), color = "white", size = 1.5)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
```

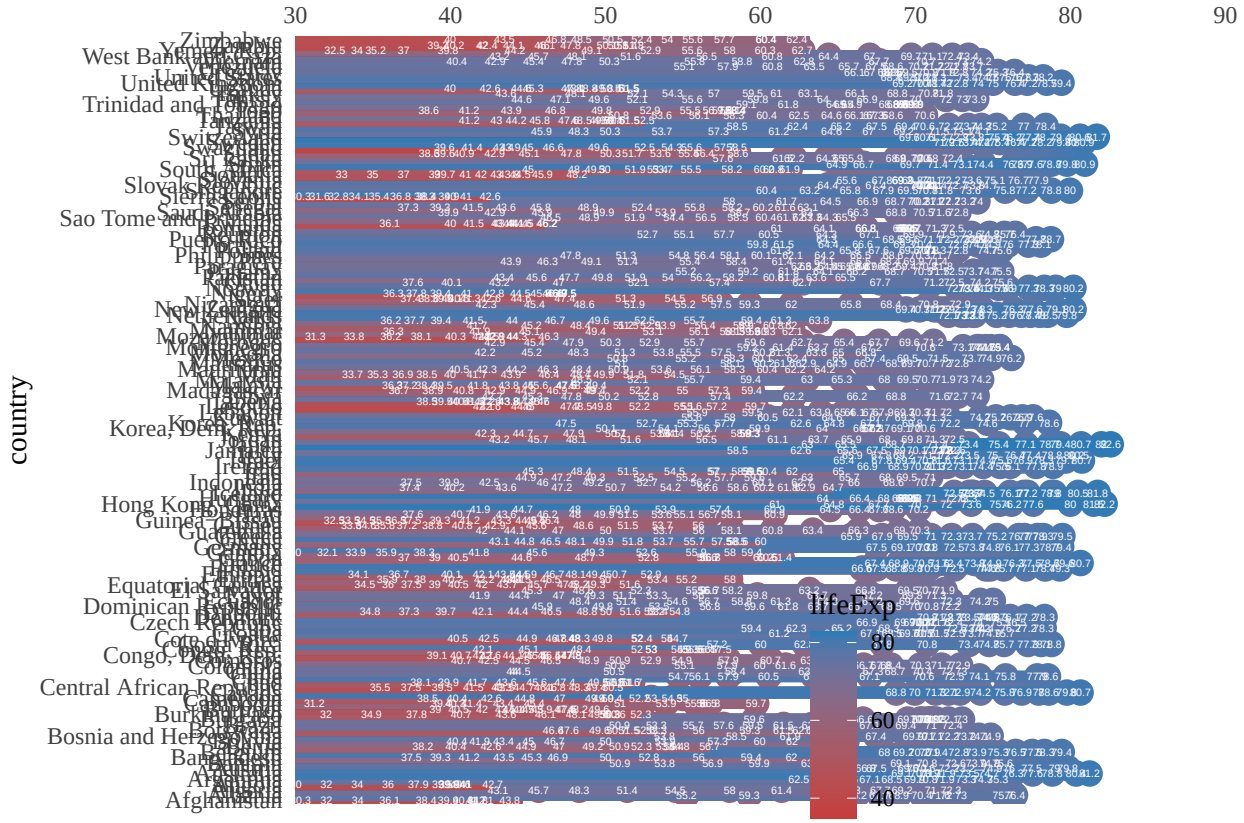



```
# Set the color scale
library(RColorBrewer)
palette <- brewer.pal(5, "RdYlBu")[-(2:4)]

# Modify the scales
ggplot(gm2007, aes(x = lifeExp, y = country, color = lifeExp)) +
  geom_point(size = 4) +
  geom_segment(aes(xend = 30, yend = country), size = 2) +
  geom_text(aes(label = round(lifeExp,1)), color = "white", size = 1.5) +
  scale_x_continuous("", expand = c(0, 0), limits = c(30, 90), position = "top") +
  scale_color_gradientn(colors = palette)
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```

```
## Warning: Removed 2 rows containing missing values (`geom_segment()`).
```

```
## Warning: Removed 2 rows containing missing values (`geom_text()`).
```

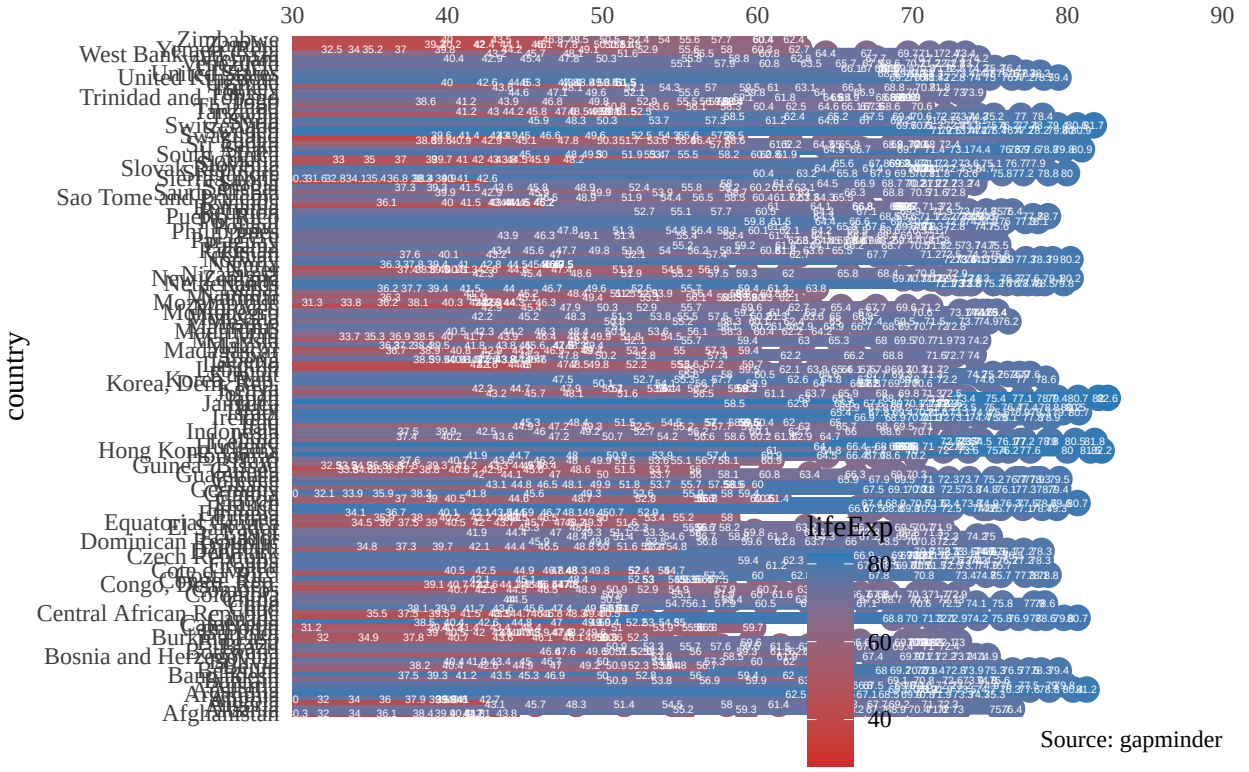



```
# Set the color scale
palette <- brewer.pal(5, "RdYlBu")[-(2:4)]

# Add a title and caption
ggplot(gm2007, aes(x = lifeExp, y = country, color = lifeExp)) +
  geom_point(size = 4) +
  geom_segment(aes(xend = 30, yend = country), size = 2) +
  geom_text(aes(label = round(lifeExp,1)), color = "white", size = 1.5) +
  scale_x_continuous("", expand = c(0,0), limits = c(30,90), position = "top") +
  scale_color_gradientn(colors = palette) +
  labs(title = "Highest and lowest life expectancies, 2007", caption = "Source: gapminder")
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```

```
## Warning: Removed 2 rows containing missing values (`geom_segment()`).
```

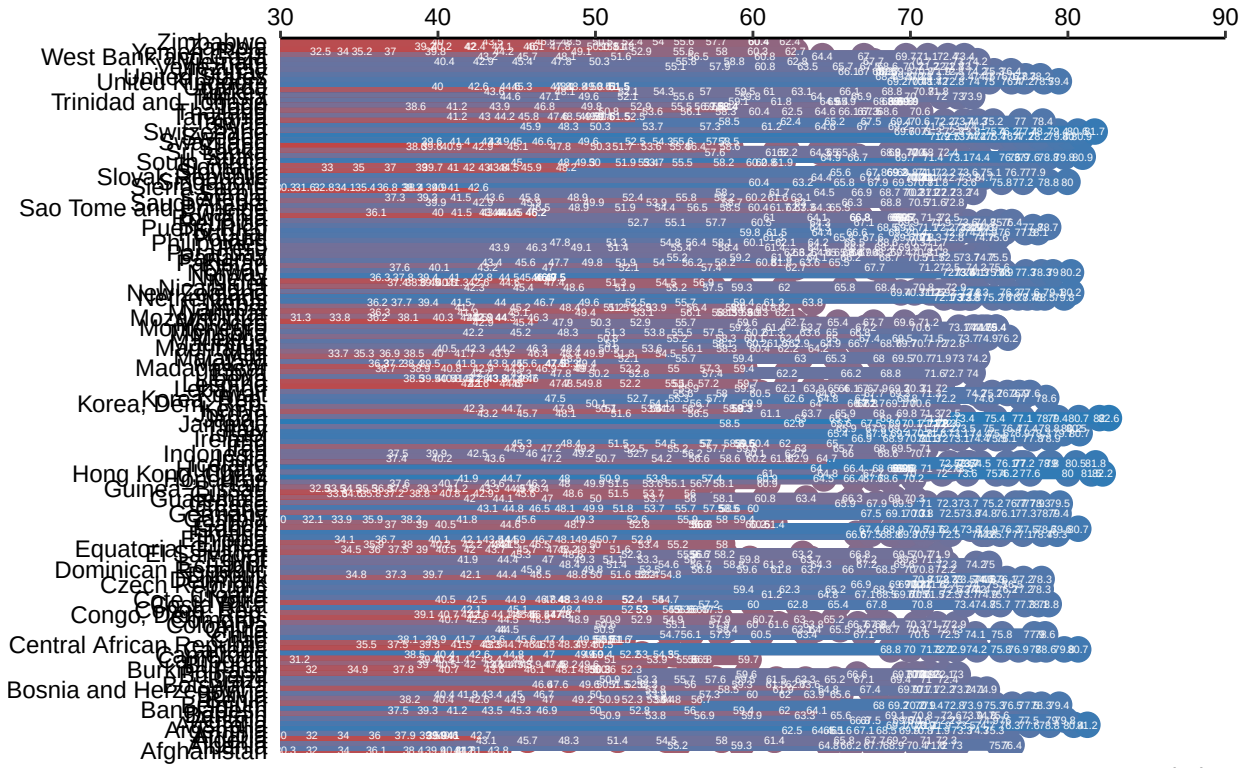
```
## Warning: Removed 2 rows containing missing values (`geom_text()`).
```



11. Using annotate() for embellishments

```
#
plt_country_vs_lifeExp <- ggplot(gm2007, aes(x = lifeExp, y = country, color = lifeExp)) +
  geom_point(size = 4) +
  geom_segment(aes(xend = 30, yend = country), size = 2) +
  geom_text(aes(label = round(lifeExp,1)), color = "white", size = 1.5) +
  scale_x_continuous("", expand = c(0,0), limits = c(30,90), position = "top") +
  scale_color_gradientn(colors = palette) +
  labs(title = "Highest and lowest life expectancies, 2007", caption = "Source: gapminder")

# Define the theme
plt_country_vs_lifeExp +
  theme_classic() +
  theme(axis.line.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text = element_text(color = "black"),
        axis.title = element_blank(),
        legend.position = "none")
```

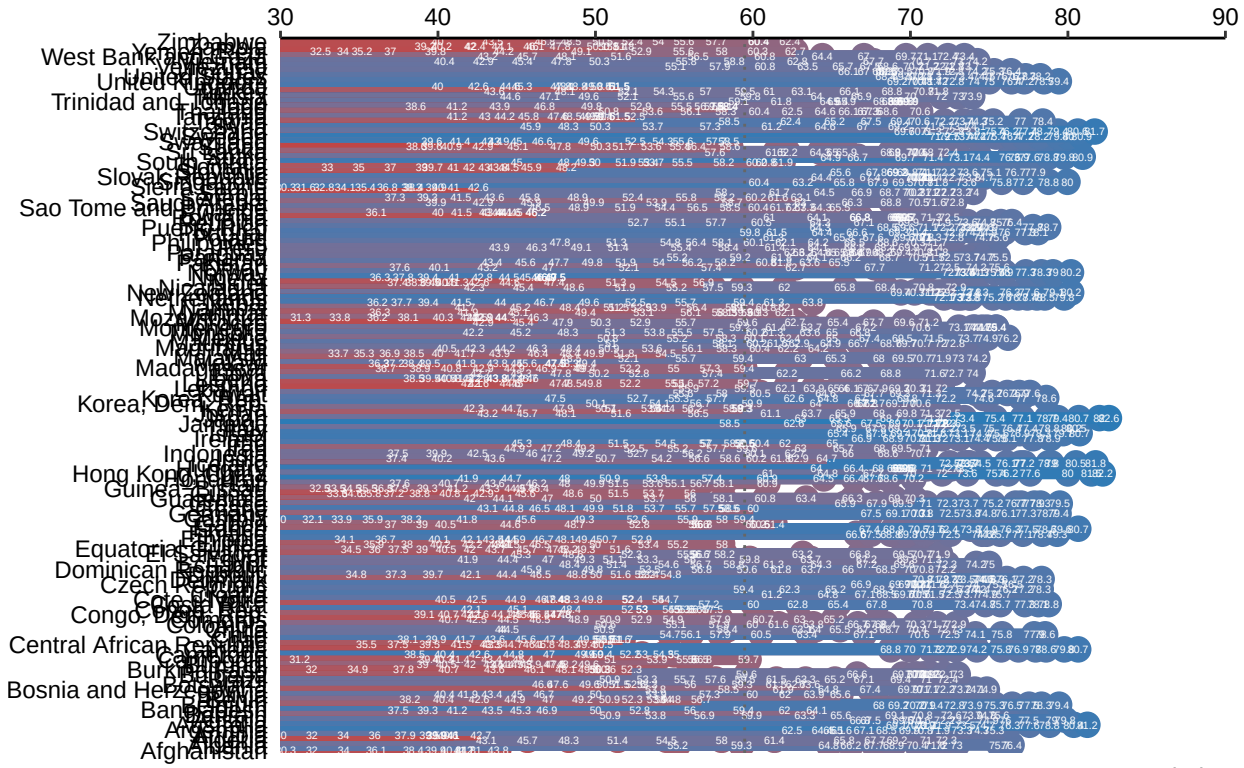
```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
## Warning: Removed 2 rows containing missing values (`geom_segment()`).
## Warning: Removed 2 rows containing missing values (`geom_text()`).
```



Source: gapminder

```
# gm2007(gm2007_full) + other embellishments calculations
gm2007_full <- gm2007
global_mean <- mean(gm2007_full$lifeExp)
x_start <- global_mean + 4
y_start <- 5.5
x_end <- global_mean
y_end <- 7.5

# step_1_themes
step_1_themes <- theme_classic() +
  theme(axis.line.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text = element_text(color = "black"),
        axis.title = element_blank(),
        legend.position = "none")

# Add a vertical line
plt_country_vs_lifeExp +
  step_1_themes +
  geom_vline(xintercept = global_mean, color = "grey40", linetype = 3)
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).

## Warning: Removed 2 rows containing missing values (`geom_segment()`).

## Warning: Removed 2 rows containing missing values (`geom_text()`).
```



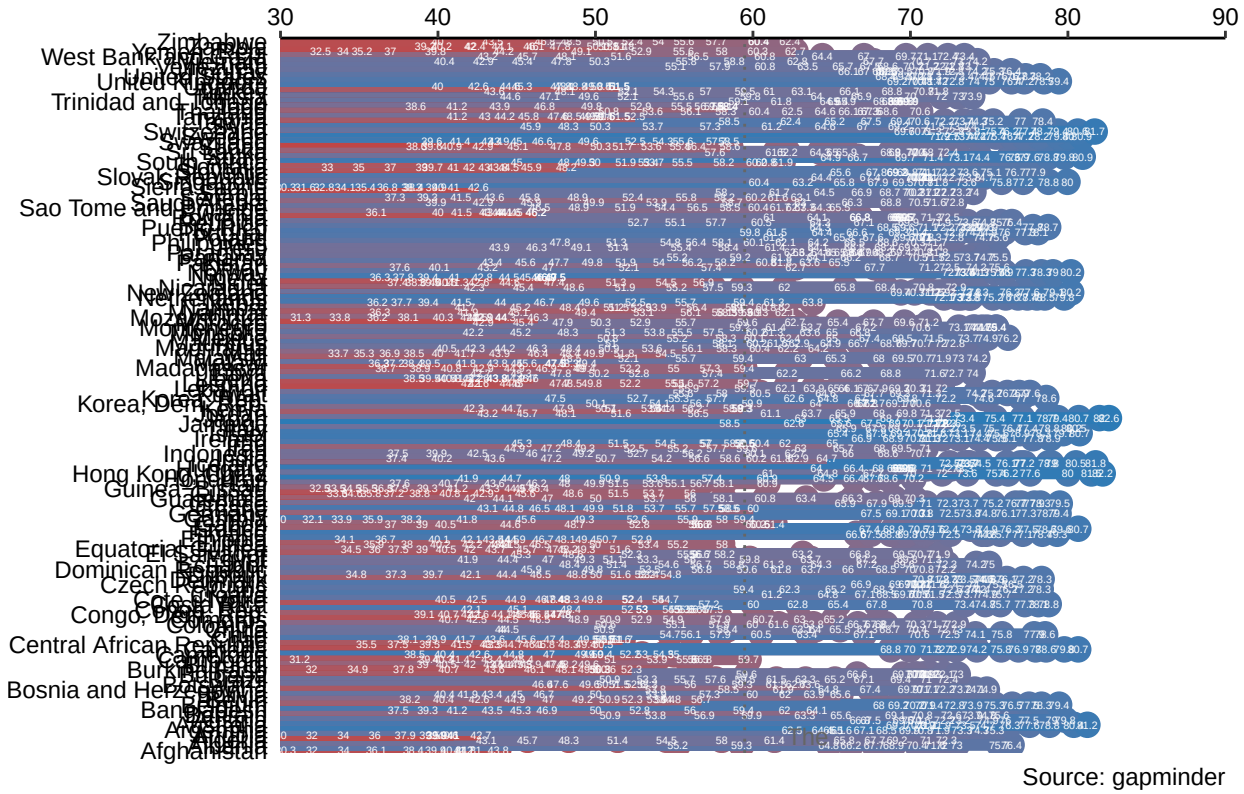
Source: gapminder

```
# Add text
plt_country_vs_lifeExp +
  step_1_themes +
  geom_vline(xintercept = global_mean, color = "grey40", linetype = 3) +
  annotate(
    "text",
    x = x_start, y = y_start,
    label = "The\nglobal\naverage",
    vjust = 1, size = 3, color = "grey40"
  )
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).
```

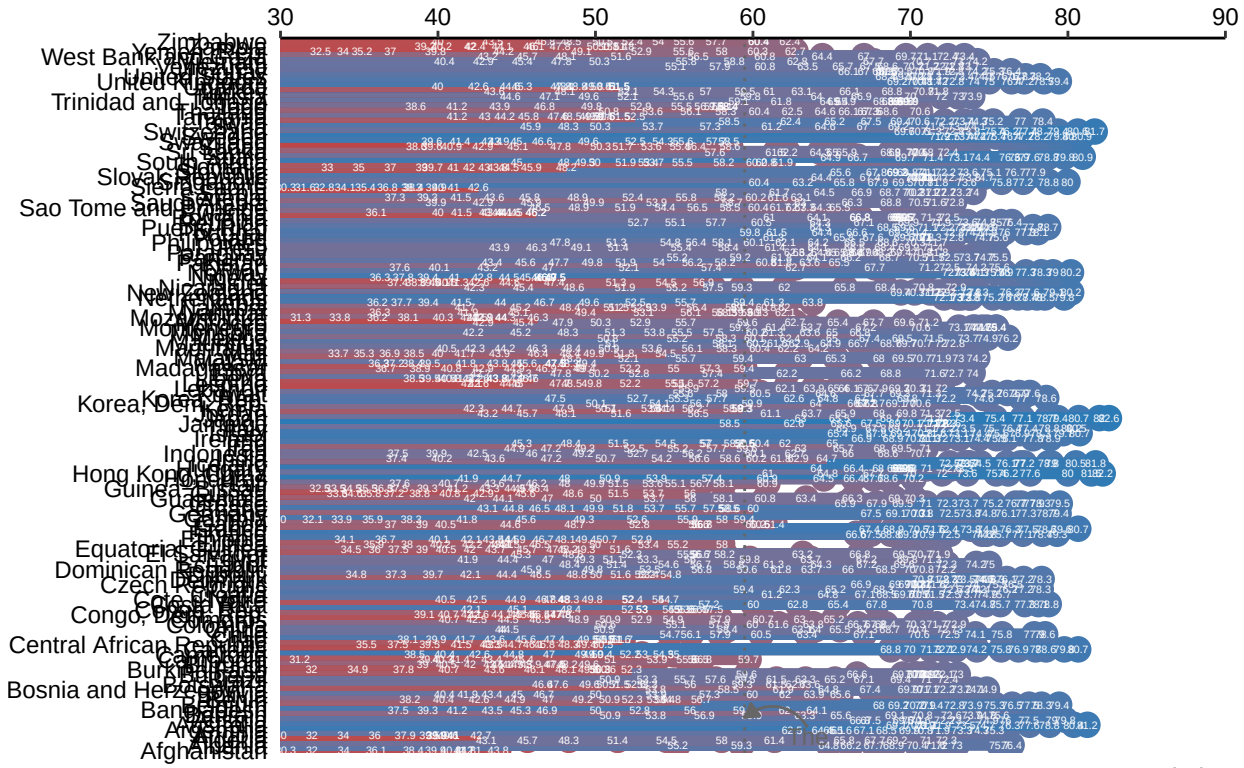
```
## Warning: Removed 2 rows containing missing values (`geom_segment()`).
```

```
## Warning: Removed 2 rows containing missing values (`geom_text()`).
```




```
# store as step_3_annotation
step_3_annotation <- annotate(
    "text",
    x = x_start, y = y_start,
    label = "The\nglobal\naverage",
    vjust = 1, size = 3, color = "grey40"
  )

# Add a curve
plt_country_vs_lifeExp +
  step_1_themes +
  geom_vline(xintercept = global_mean, color = "grey40", linetype = 3) +
  step_3_annotation +
  annotate(
    "curve",
    x = x_start, y = y_start,
    xend = x_end, yend = y_end,
    arrow = arrow(length = unit(0.2, "cm"), type = "closed"),
    color = "grey40"
  )
```

```
## Warning: Removed 2 rows containing missing values (`geom_point()`).

## Warning: Removed 2 rows containing missing values (`geom_segment()`).

## Warning: Removed 2 rows containing missing values (`geom_text()`).
```


Highest and lowest life expectancies, 2007
30
40
50
60
70
80
90
Zimbabwe
Afghanistan
Source: gapminder