Data Visualization in R

Eloi Vilella

20 September 2024

Contents

0.1 Introduction

The following examples will walk you through the basic components of the `ggplot2` grammar. The examples use data from the `datasets` package, which is already loaded by default in the R session, as well as some data sets loaded with `ggplot2` package. `ggplot2` requires data to be stored in data frames and in a **long format** (one observation per row and one variable per column). In some cases, the **wide format** is also used. For example, `mtcars` dataset is in **wide format**:

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

where each column represents a different variable.

0.1.1 Organization of the practical

You will see different icons through the document, the meaning of which is:

: additional or useful information : a worked example : a practical exercise : a space to answer the exercise
: a hint to solve an exercise : a more challenging exercise

`ggplot2` is a data visualization package for the statistical programming language R. Created by Hadley Wickham in 2005, `ggplot2` is an implementation of Leland Wilkinson's Grammar of Graphics — a general scheme for data visualization which breaks up graphs into semantic components such as scales and layers. We will further learn about `ggplot2` in our next theoretical session.

0.2 Example 1 | Creating a scatter plot

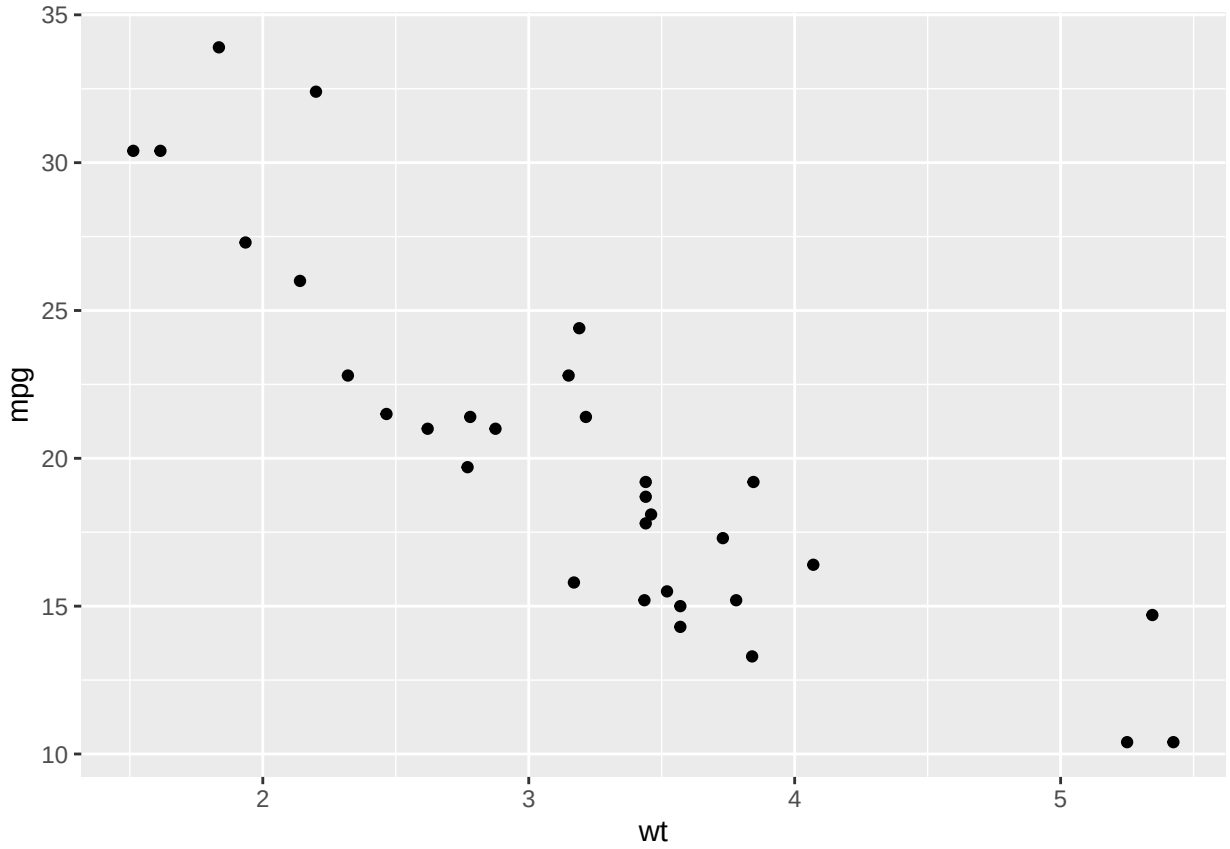
0.2.1 1a | Basic scatter plot

For the first problem we want to represent the relationship between the variables `wt` (weight) and `mpg` (miles/gallon) from the `mtcars` data frame.

The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models). You can type `?mtcars` in the R console to read a description of the data.

To represent any graph in `ggplot2` we need two basic functions that are combined with a + sign.

```
# Run install.packages("ggplot2") if ggplot2 is not yet installed
library(ggplot2)
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) +
  geom_point()
```



The variables that we want to represent are wrapped within an `aes()` function, that specifies the ***mapping*** between the variables and the ***aesthetic attributes*** (in this case we map them to spatial positions, `x` and `y`). We call the variables directly by their names, because we also pass the entire data frame to the call with the data argument, so `ggplot` knows were to get them from. Finally, we need to add the ***geometric object*** we want to represent. In this case, points.

0.2.2 1b | Represent extra variables

Another variable in the data indicates the number of cylinders of the car engines (`cyl`). There are cars with 4, 6 or 8 cylinders.

```
table(mtcars$cyl)
```

```
##
##  4  6  8
## 11  7 14
```

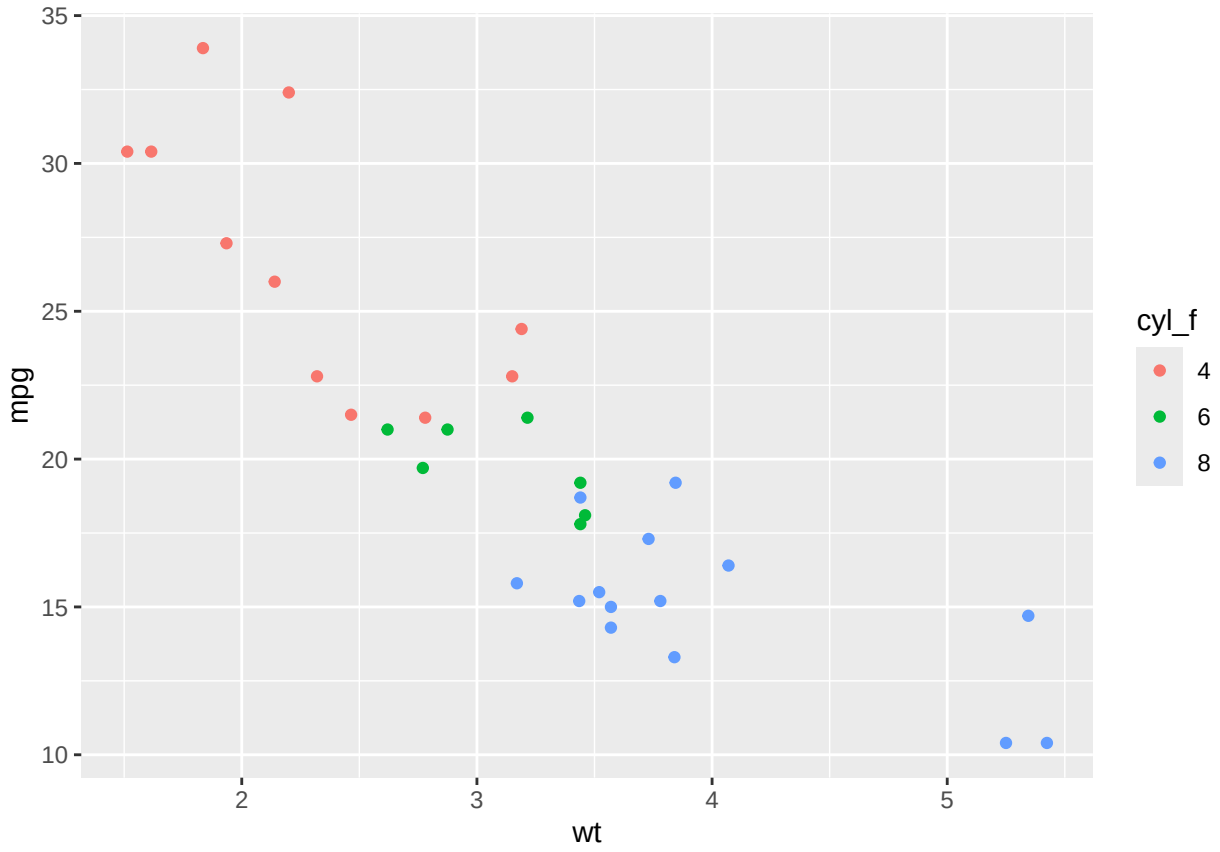
Let's say we want to represent the different types of cylinders in different colours. In this case we want to use `cyl` as a categorical variable, distinguishing groups rather than indicating a value in a continuous scale. For that, we need to change its class before giving it to `ggplot` using the `factor()` function.

```r
# First we check the class of cyl
class(mtcars$cyl)
```

```
## [1] "numeric"
```

```r
# Because it is numeric, let's make cyl a factor so that we represent it as a categor
# We create a new variable in the dataframe, cyl_f, that is cyl converted to factor
mtcars$cyl_f <- factor(mtcars$cyl)

ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, colour = cyl_f)) +
  geom_point()
```

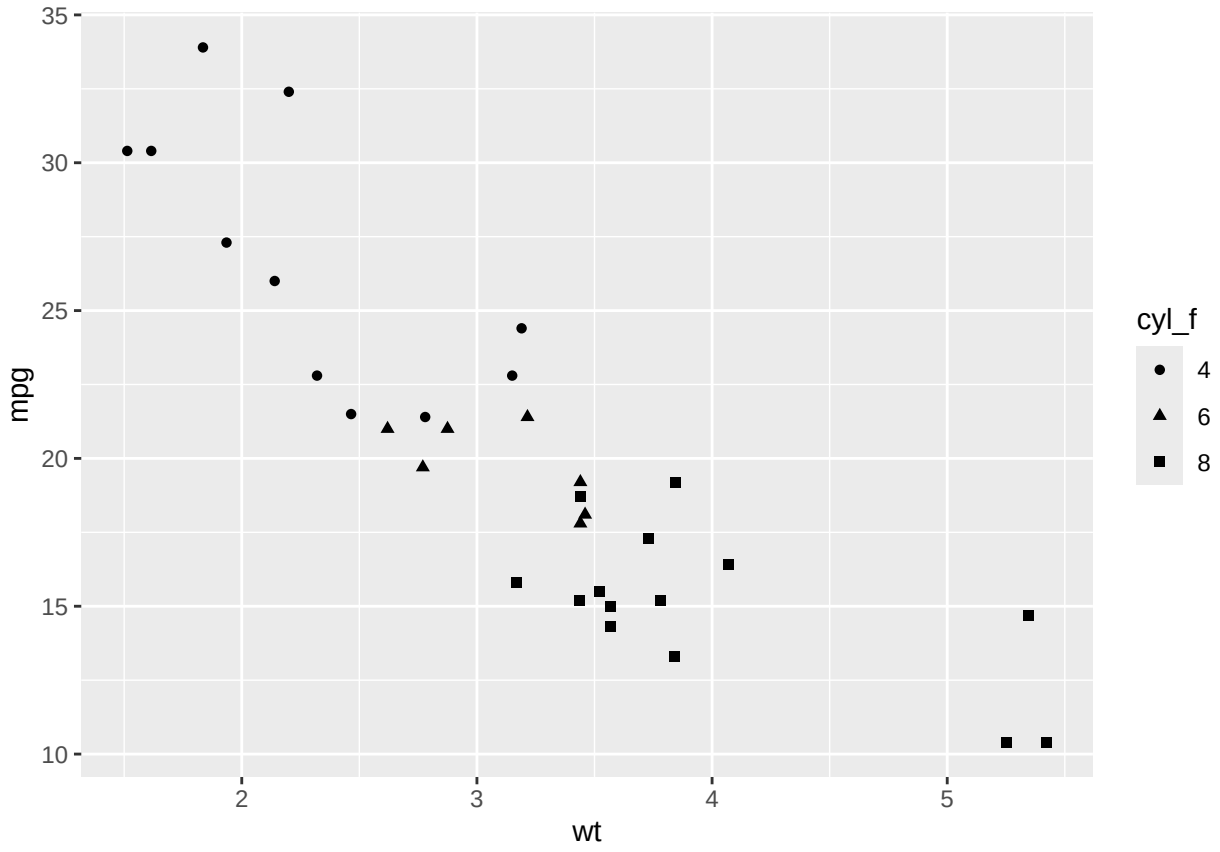


Note that `ggplot` adds a **legend** by default for all the variables that have been mapped to some aesthetic attribute. This way we can read all the variables without extra effort.

0.2.3 Exercise

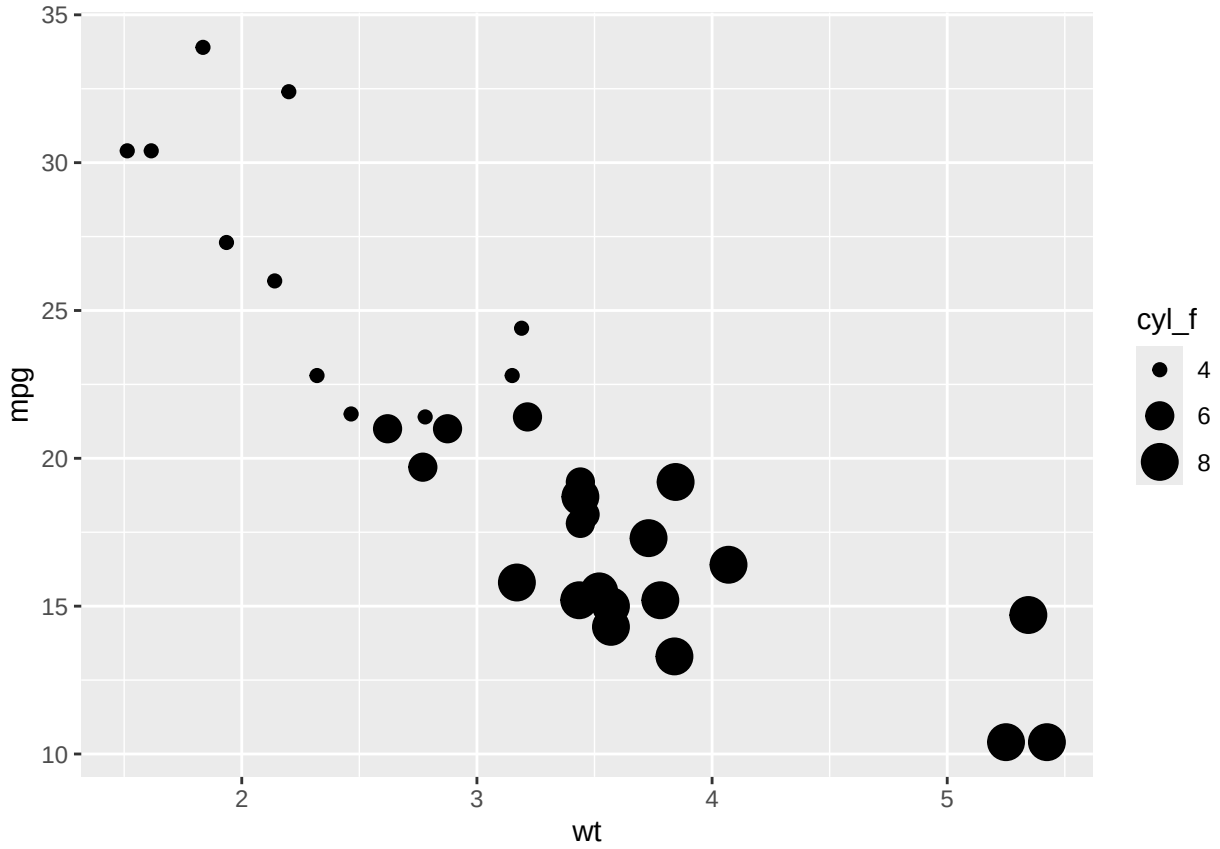
Try mapping `cyl_f` to another aesthetic attribute instead of `colour`, such as `shape` or `size`.

```
# Shape
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, shape = cyl_f)) +
  geom_point()
```




```
# Size
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, size = cyl_f)) +
  geom_point()
```

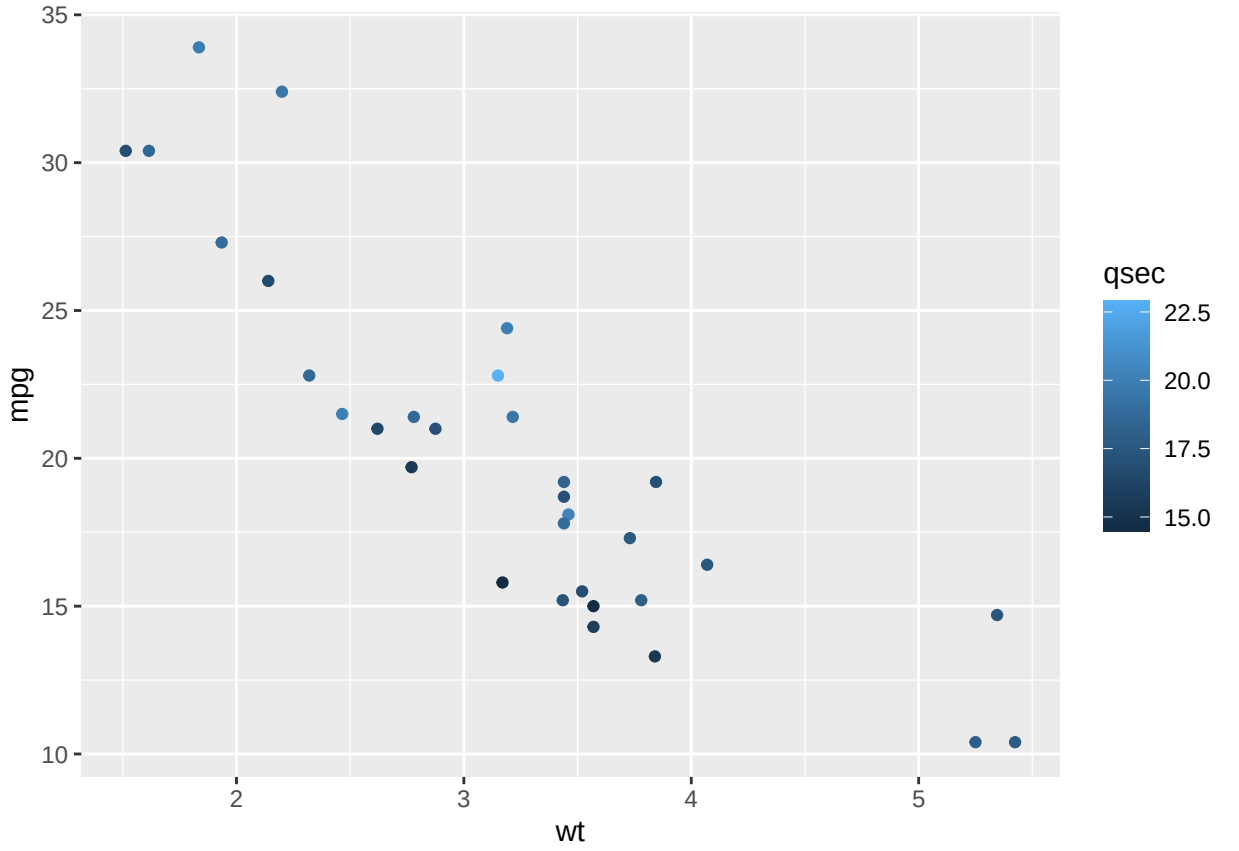
```
## Warning: Using size for a discrete variable is not advised.
```



0.2.3.0.1 Answer:

What happens if you map a continuous variable such as `qsec`, instead of `cyl`, to `colour`? And to `shape`?

```
#Instead of categorical colours, it appears a gradient of 2 shades of blue which enco
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, colour = qsec)) +
  geom_point()
```




```
# To shape is not possible due to the continous nature of our variable
```

0.2.3.0.2 Answer:

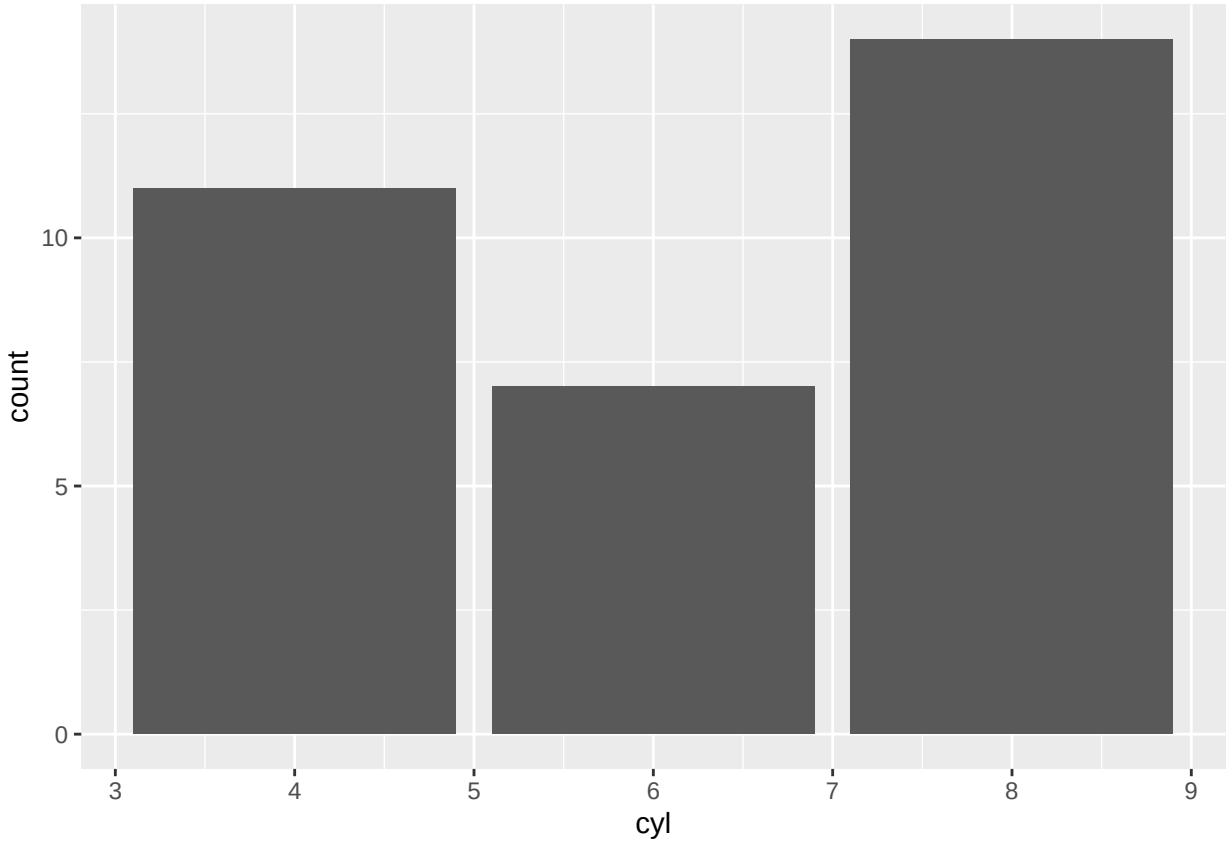
0.3 Example 2 | Creating a bar plot

0.3.1 2a | Basic bar plot

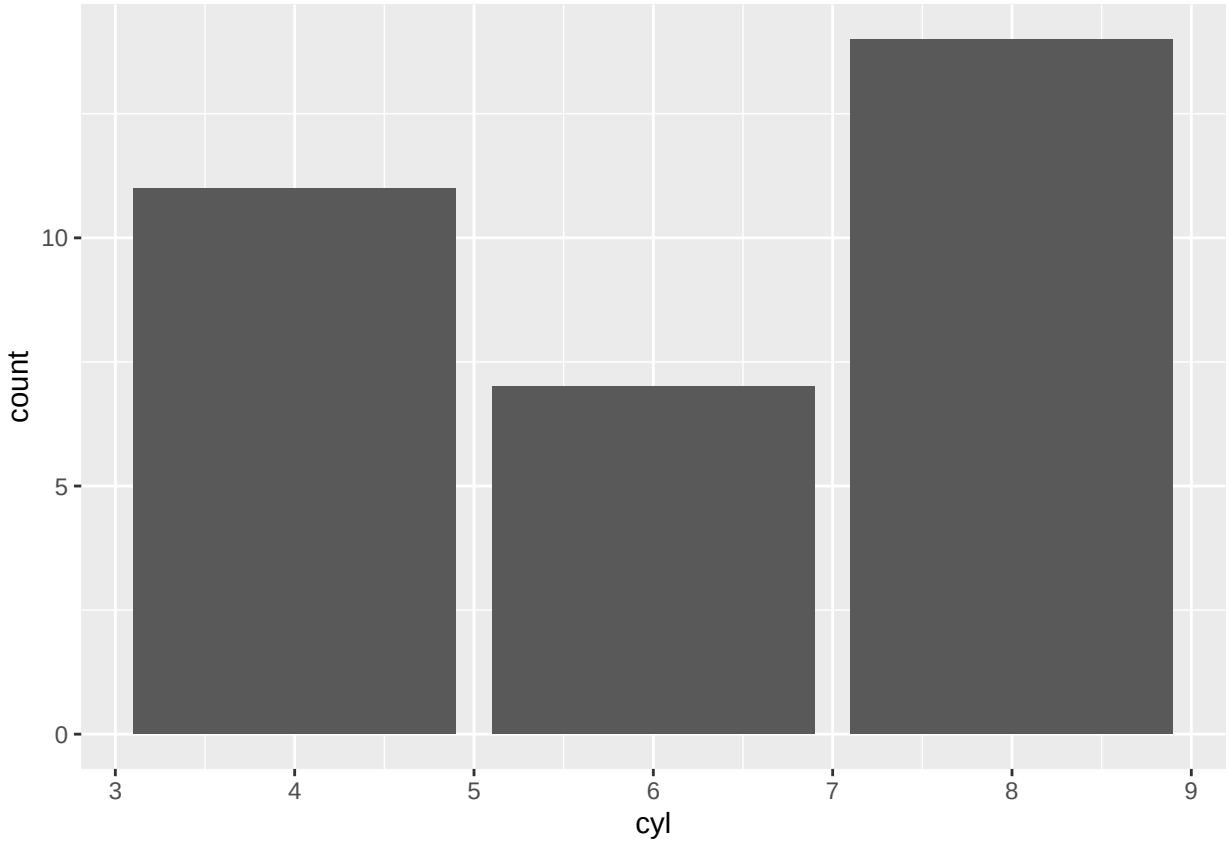
We now want to summarise our data in a simple bar plot representing the number of cars in each cylinder category. However, the number of cars with 4 cylinders is not a piece of information present in the data set, for example. To know the number it is necessary to count the rows where `cyl = 4`.

`ggplot2` is capable to do simple summary operations with the input variables, refered as **statistical tranformations**. One of them is to count the occurrences of each value in a variable. And `geom_bar` function happen to use the `count` statistical transformation by default on the variable mapped to the `x` axis.

```
ggplot(data = mtcars, mapping = aes(x = cyl)) +
  geom_bar()
```




```
# # The same as
 ggplot(data = mtcars, mapping = aes(x = cyl)) +
   geom_bar(stat = "count")
```



If we had a precomputed data frame with `cyl` and `number_of_cars` instead, we could pass `number_of_cars` variable to `geom_col` function, that by default takes the variables mapped to `x` and `y` without transformation.

```
# Let's create the data frame
counts_by_cyl_data_frame <- as.data.frame(table(mtcars$cyl))
names(counts_by_cyl_data_frame) <- c("cyl", "number_of_cars")

#ggplot(data = counts_by_cyl_data_frame, mapping = aes(x = cyl, y = number_of_cars))
 # geom_col()
```

Alternatively, we could remove the default statistical transformation of `geom_bar` with `stat = "indentity"` and use the precomputed data frame.

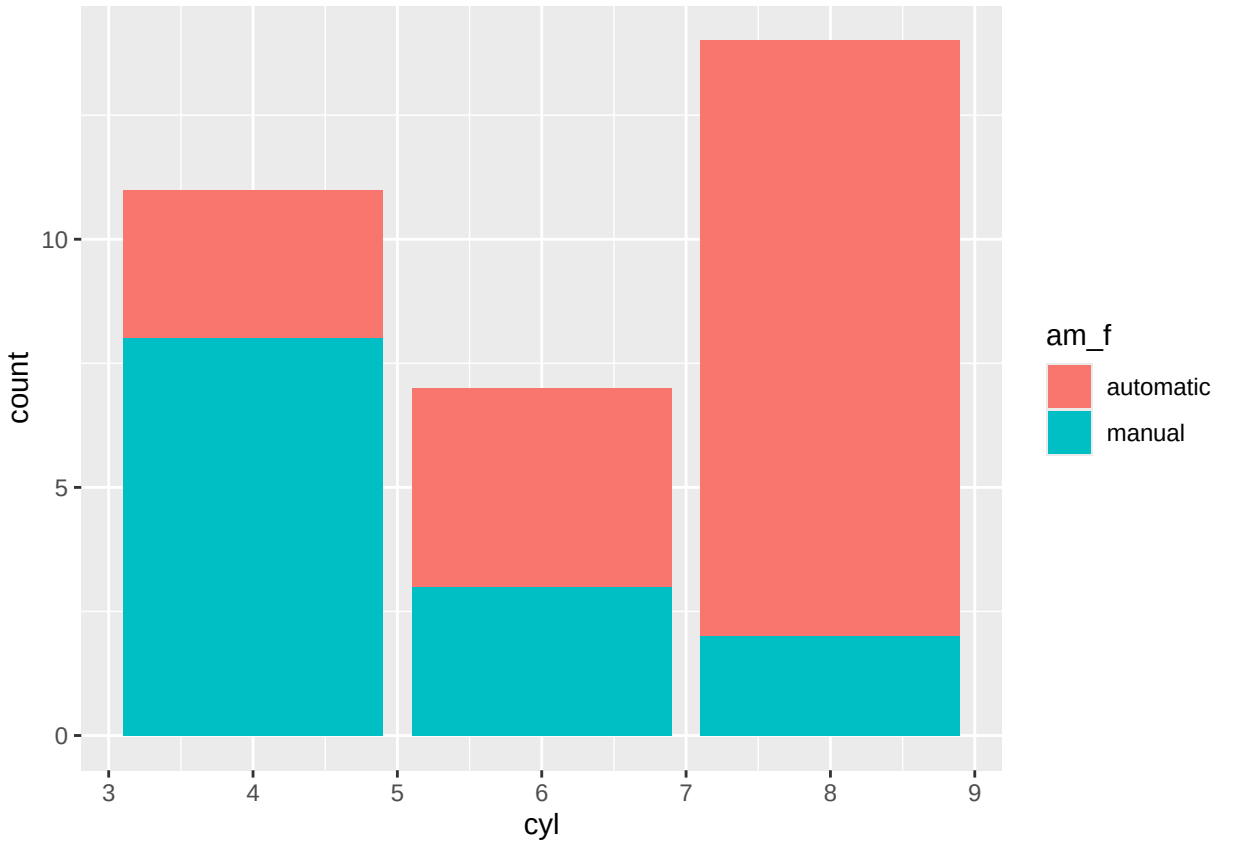
```
# # The same as
# ggplot(data = counts_by_cyl_data_frame, mapping = aes(x = cyl, y = number_of_cars))
#   geom_bar(stat = "identity")
```

0.3.2 2b | Groups and position

We have seen in the scatter plot example how to represent groups encoded in extra variables as colours. Say we now want to show transmission type (`am`) in the bar plot, in addition to the number of cylinders. We can map `am` to the filling colour of the bars, `fill`. (`colour` would change the edges of the rectangles.)

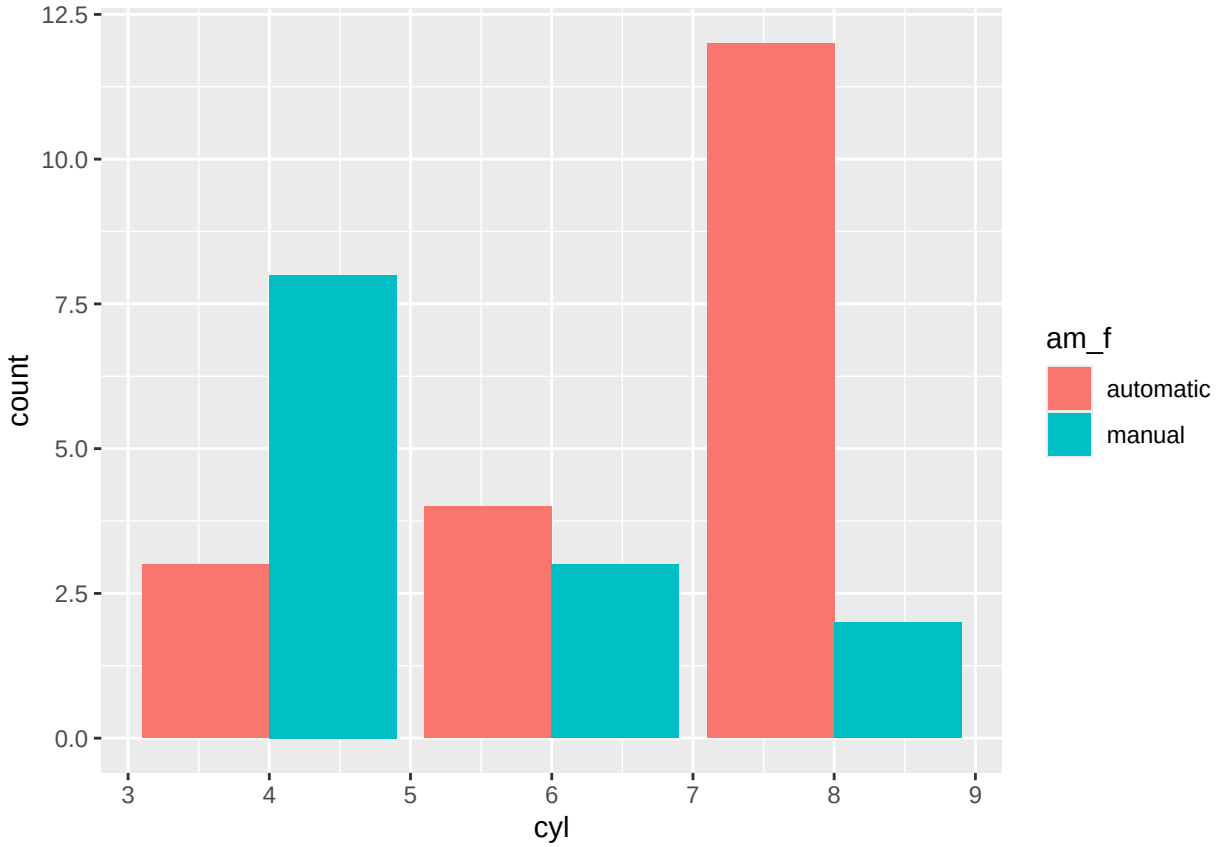
```r
# First we make am factor, and we can change the 0/1 notation for a more informative
mtcars$am_f <- factor(mtcars$am, levels = c(0, 1), labels = c("automatic","manual"))

ggplot(data = mtcars, mapping = aes(x = cyl, fill = am_f)) +
  geom_bar()
```



Each geometric object in `ggplot2` also has a **position** argument that controls how groups are arranged. In `geom_bar` the default position is to stack any groups. We can change it for a side-by-side position with `position = "dodge"`.

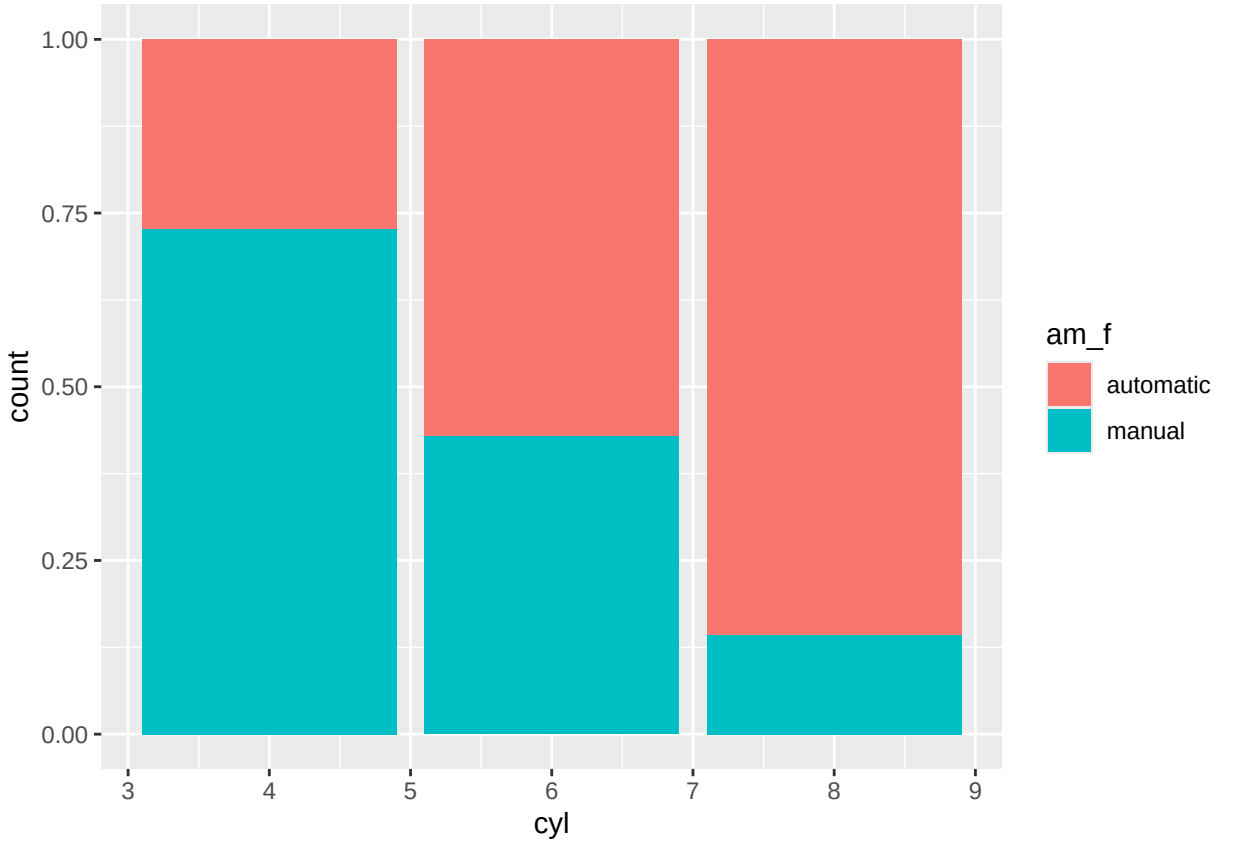
```r
ggplot(data = mtcars, mapping = aes(x = cyl, fill = am_f)) +
  geom_bar(position = "dodge")
```



0.3.3 Exercise

Which is the `position` argument in the `ggplot2` bar plot that standardises the bars to the same height? Update the plot above with the new position adjustment.

```
# The argument is fill
ggplot(data = mtcars, mapping = aes(x = cyl, fill = am_f)) +
  geom_bar(position = "fill")
```



0.3.3.0.1 Answer:

Which position adjustment would you choose if you still wanted to compare the total amount of cars with each cylinder category? And if you were interested in knowing the relative abundance of each type of transmission?

0.3.3.0.2 Answer:

0.4 Example 3 | Showing the distribution of a variable

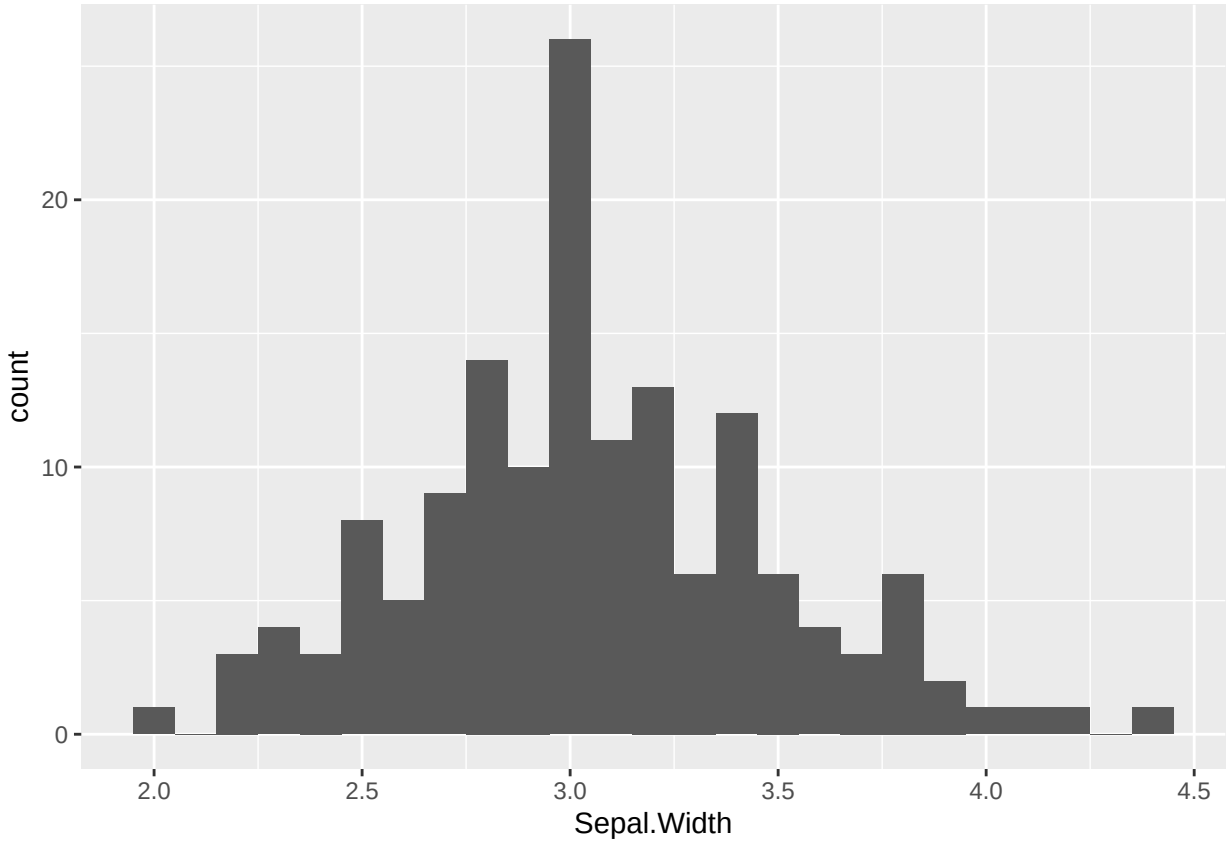
0.4.1 3a | Simple histogram

Now we have a new data set called `iris` and we need to understand the distribution of some of its continuous variables. A good place to start is a histogram, that represents the number of observations in different ranges as bars.

Note that histograms deal with *continuous* variables while bar plots with *discrete*, but are sometimes confused.

The function that we need is called `geom_histogram` and has the statistical transformation `bin` by default. In this case, `bin` divides the variable mapped to `x` in ranges and counts the number of values in each bin. The number of bins is controlled with the argument `binwidth`.

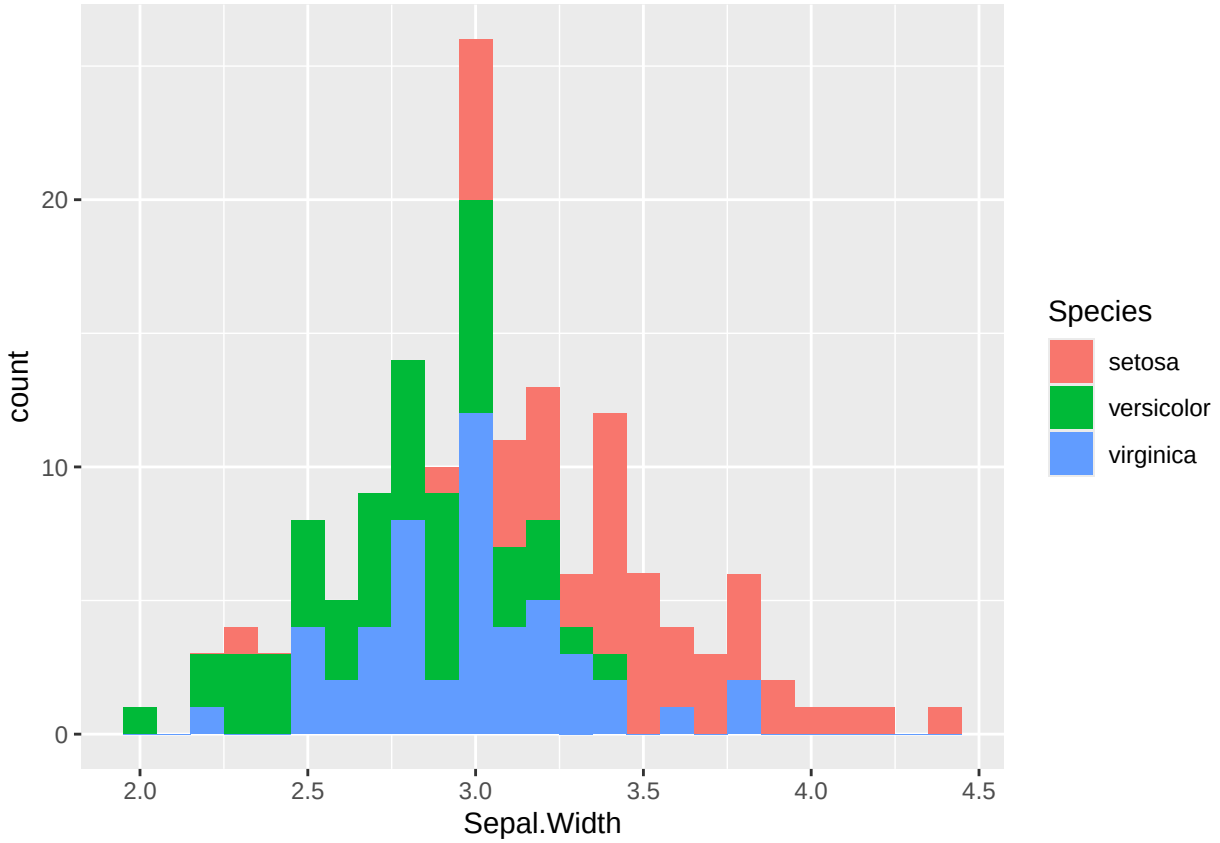
```r
ggplot(data = iris, mapping = aes(x = Sepal.Width)) +
  geom_histogram(binwidth = 0.1)
```



0.4.2 3b | Multiple histograms

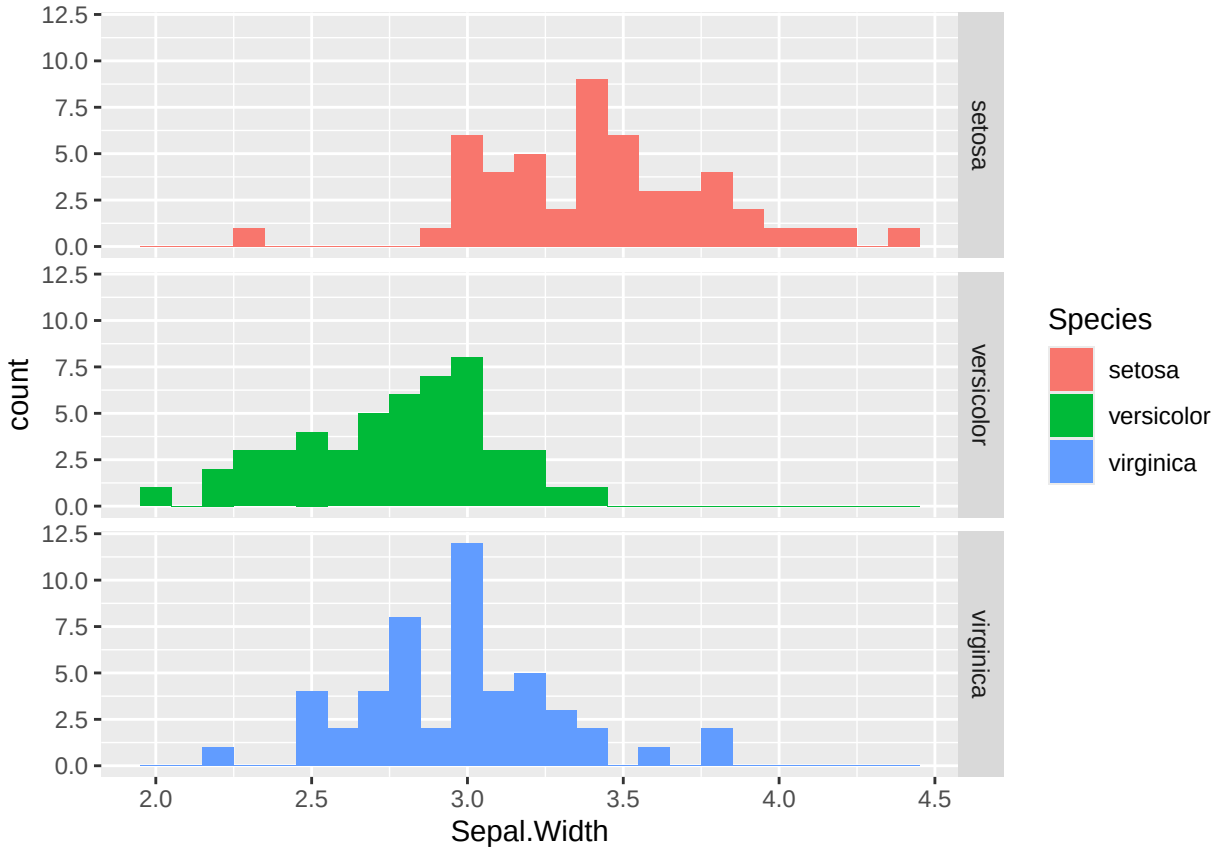
`iris` data contains information about three species of iris: *setosa*, *versicolor* and *virginica*. To see the distribution of the different species we can try to map the species to the filling colour. That's easy with `ggplot2`!

```r
ggplot(data = iris, mapping = aes(x = Sepal.Width, fill = Species)) +
  geom_histogram(binwidth = 0.1)
```



Stacked histograms are difficult to interpret and three separated subplots could actually work better. `ggplot2` provides a simple way of creating small multiples or **facets** with the functions `facet_grid` and `facet_wrap`.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, fill = Species)) +
  geom_histogram(binwidth = 0.1) +
  facet_grid(Species ~ .)
```



0.4.3 Exercise

Experiment with `facet_grid` and `facet_wrap`. For testing purposes, we can create an extra categorical variable by splitting `Petal.Length` in two groups.

```
iris$Petal.Type[iris$Petal.Length >= 4 ] <- "Long"
iris$Petal.Type[iris$Petal.Length < 4 ] <- "Short"
```

0.4.3.0.1 Answer:

Which is the best subplot configuration to compare the distributions and why?

0.4.3.0.2 Answer:

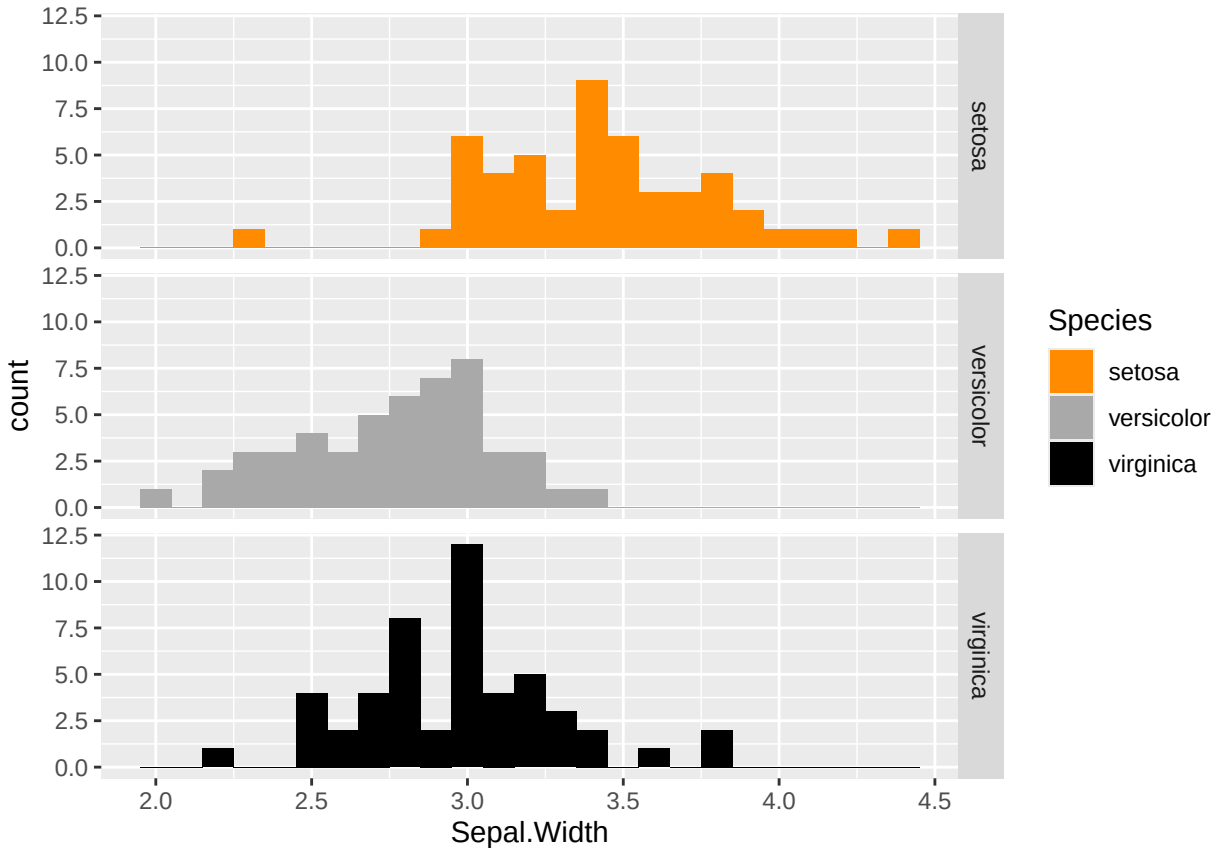
0.5 Example 4 | Customizing a plot

0.5.1 4a | Modify colours

So far we have used the default colour palettes for all our representations. We may need to change them to make them accessible to colourblind people, match the colour palette of our project or give meaningful values (e.g., red

for positive and blue for negative). We can control the exact mapping of a variable to an aesthetic attribute with the functions `scale_*`.

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, fill = Species)) +
  geom_histogram(binwidth = 0.1) +
  facet_grid(Species ~ .) +
  scale_fill_manual(values = c("darkorange", "darkgray", "black"))
```



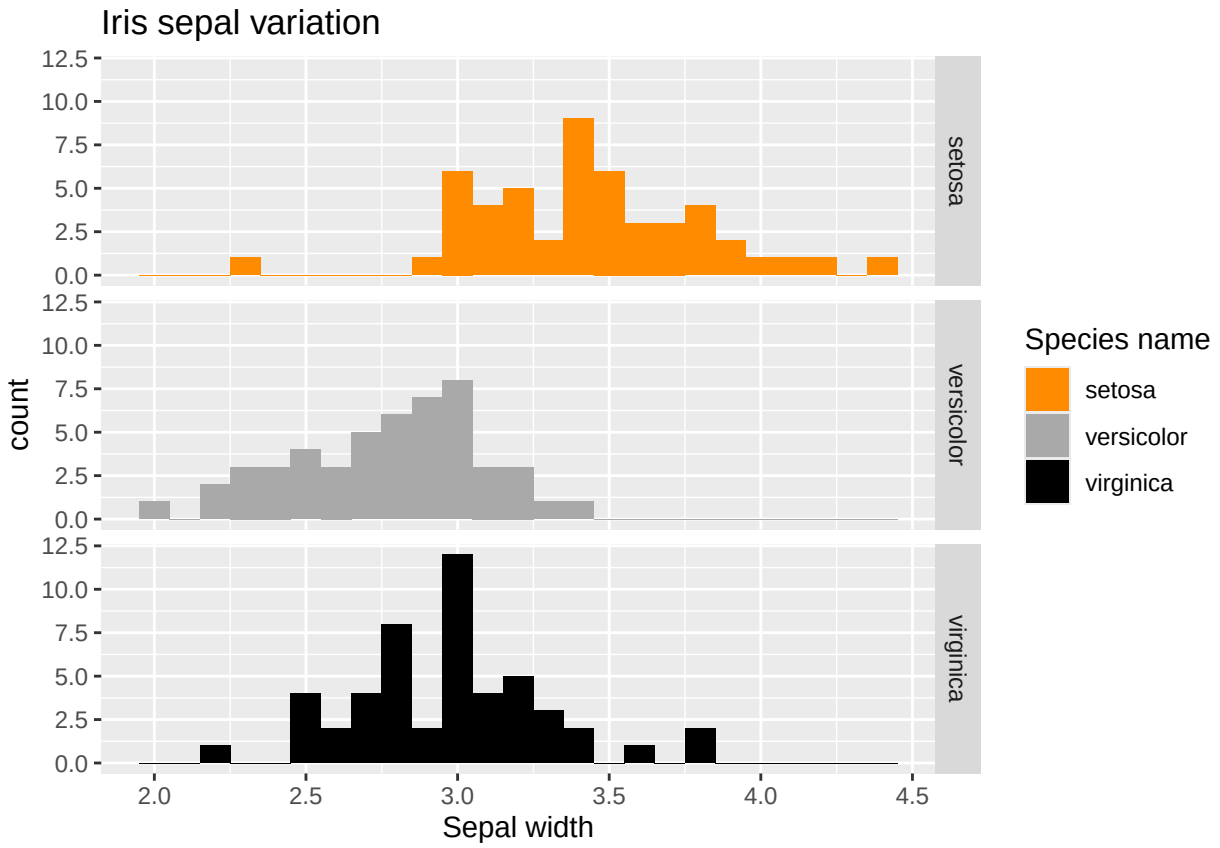
Note that scale functions update both the aesthetic mappings in the plot and in the legend.

0.5.2 4b | Change (or add) axis, legend and plot titles

We may also need to add a title to the plot or change the axis title. In `ggplot2` axis and legend titles can be specified with `name` argument within a `scale_*` function. The title is set with `ggtitle`. You can also use the convenience function `labs`.

```
# We save the common part of the plot in a variable and then we can add more componen
p <- ggplot(data = iris, mapping = aes(x = Sepal.Width, fill = Species)) +
  geom_histogram(binwidth = 0.1) +
  facet_grid(Species ~ .)
```

```r
# Option A:
p + scale_fill_manual(values = c("darkorange", "darkgray", "black"), name = "Species
  scale_x_continuous(name = "Sepal width") +
  ggtitle("Iris sepal variation")
```

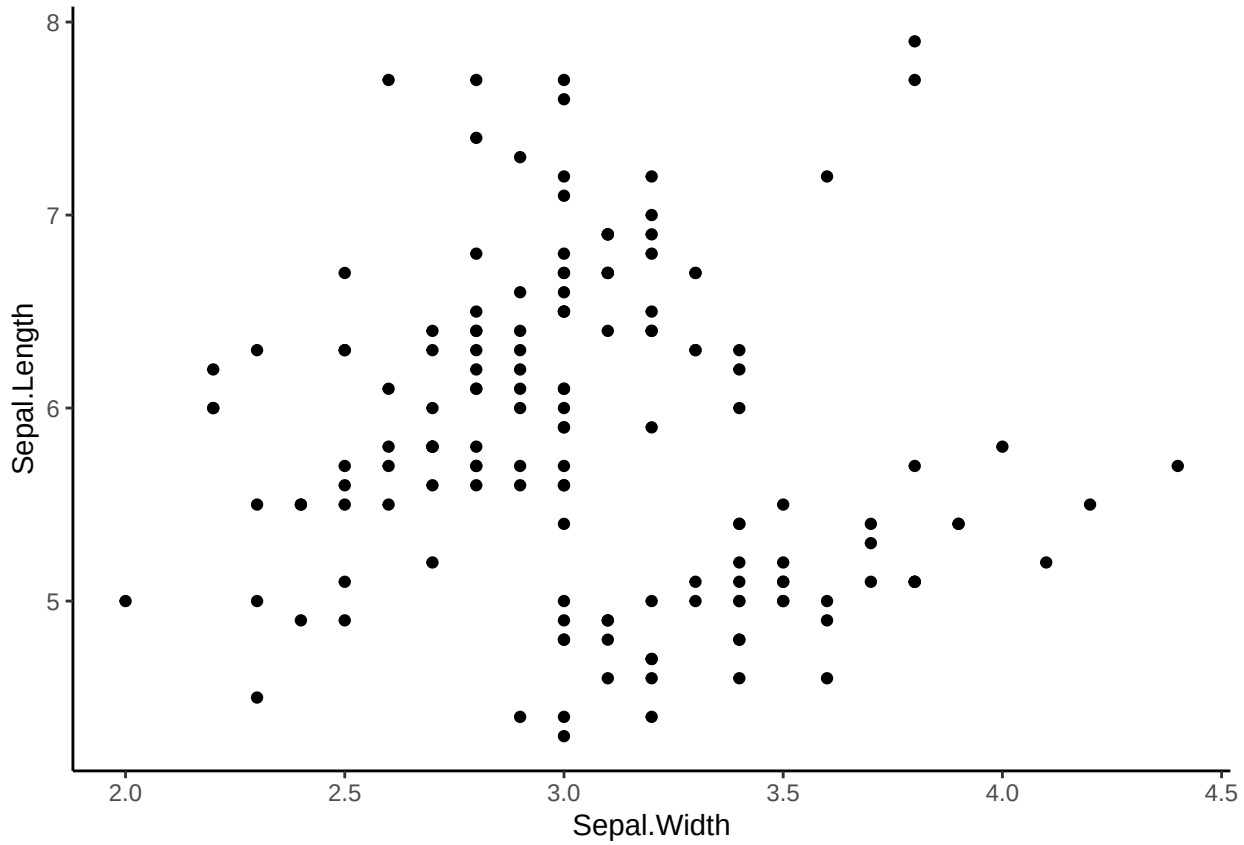



```r
# Option B:
# p + scale_fill_manual(values = c("darkorange", "darkgray", "black")) +
#   labs(title = "Iris sepal variation", x = "Sepal width", fill = "Species name")
```

0.5.3 4c | Change theme

The appearence of `ggplot2` plots is controlled by the **themes**. The default `ggplot2` theme has a gray background and "is designed to put the data forward yet make comparisons easy". You can change the general appearence by choosing a different theme with `theme_*` functions.

```r
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point() +
  theme_classic()
```



0.5.4 Exercise

Try other `scale_fill_*` functions in `ggplot2` with pre-defined palettes, such as `scale_fill_brewer` and `scale_fill_viridis_d`. Which palette would you use to ensure that colourblind people can distinguish the colours?

#Viridis palet in most colorblindcases

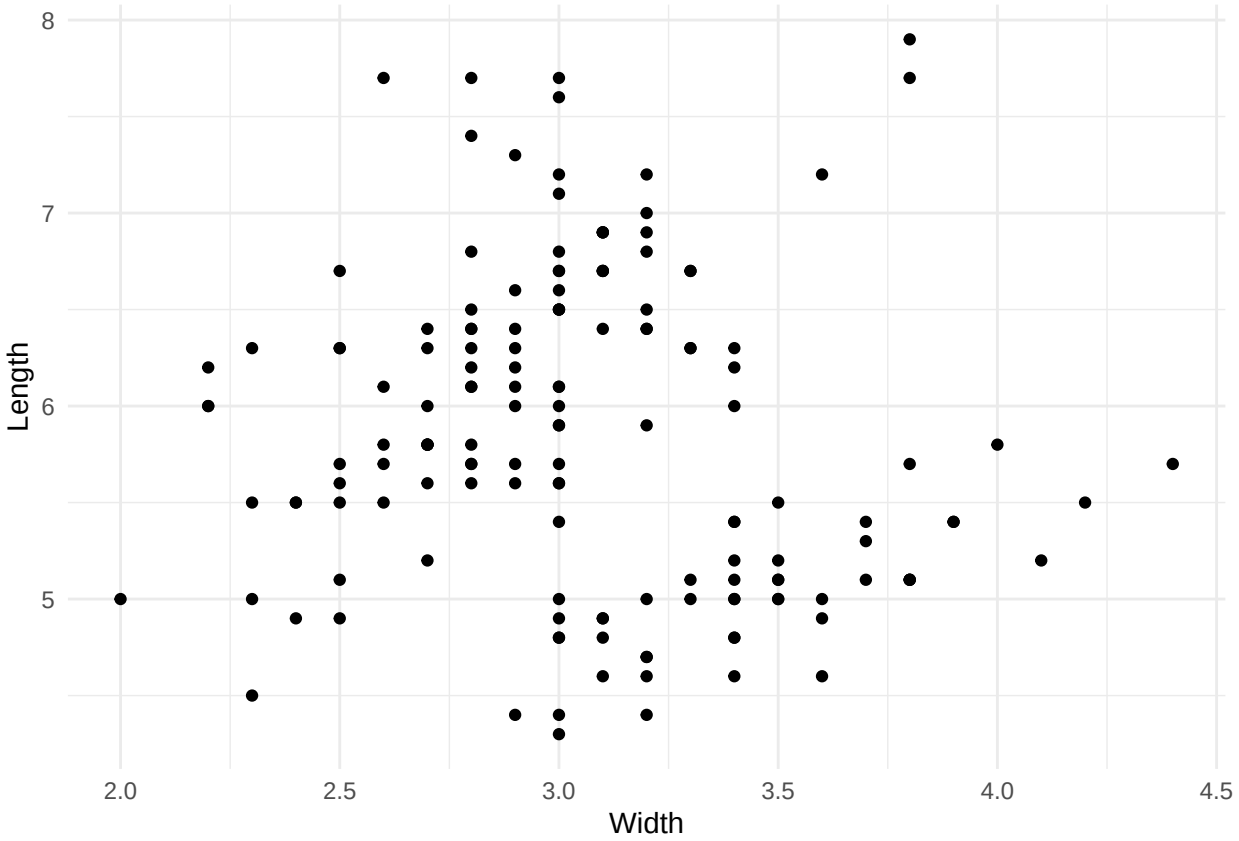
0.5.4.0.1 Answer:

Try `subtitle`, `caption` and `tag` arguments from the `labs` function. What are they for?

0.5.4.0.2 Answer:

Which theme do you think that maximises the data-ink ratio?

```
ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point() +
  theme_minimal() +
  labs(x = "Width", y = "Length")
```




```
#Theme minimal
```

0.5.4.0.3 Answer:

0.6 Saving the plots

There are three ways to save a plot to a file (from easy to difficult):

A. Export button from `RStudio` plot pane

B. `ggsave` function from `ggplot2` package

```
p <- ggplot(data = iris, mapping = aes(x = Sepal.Width, y = Sepal.Length)) + geom_poi
ggsave(filename = "plot.png", plot = p, width = 6, height = 4) # in inches by default
```

C. Opening > Ploting > Closing a graphic device

```
png(filename = "plot.png", width = 600, height = 400, res = 150) # In pixels by defau
p
dev.off()
```

Plots can be saved using different image file formats. Option **A** gives you the format options in a drop list, option **B** guesses the format from the filename extension, and in option **C** the function that is used to open the graphic device determines the format of the output (in the example `png()`).

The main formats can be classified into:

- Raster/bitmat formats, where information is stored in pixels and have a maximum resolution.
  - **PNG**: extension .png, supports transparent background, good compression, doesn't lose quality
  - **JPEG**: extensions .jpg and .jpeg, very good compression, used in personal photography but suffers from quality degradation with repeated modifications
  - **TIFF**: extensions .tif and .tiff, preferred format for professional printing
- Vector formats, where information is encoded in geometric shapes that can be rendered at any size without losing resolution.
  - **SVG**: extension .svg, standard for vector graphics, requires `svglite` package
- Hybrid
  - **PDF**: can contain both vector graphics and raster images

0.6.1 Exercise

Save the plot `p` in a raster and a vector format with the same size. What differences do you observe?

Note: svg devices require `svglite` R package and other system libraries. Skip the exercise if you get an error!

```
#install.packages("sgvlite")

#ggsave(filename = "plot.png", plot = p, width = 6, height = 4)
#ggsave(filename = "plot.svg", plot = p, width = 6, height = 4)

#svg is ligther and you can do infinite zoom to its objects due to information not be
```

0.6.1.0.1 Answer:

0.7 Wrap up exercise

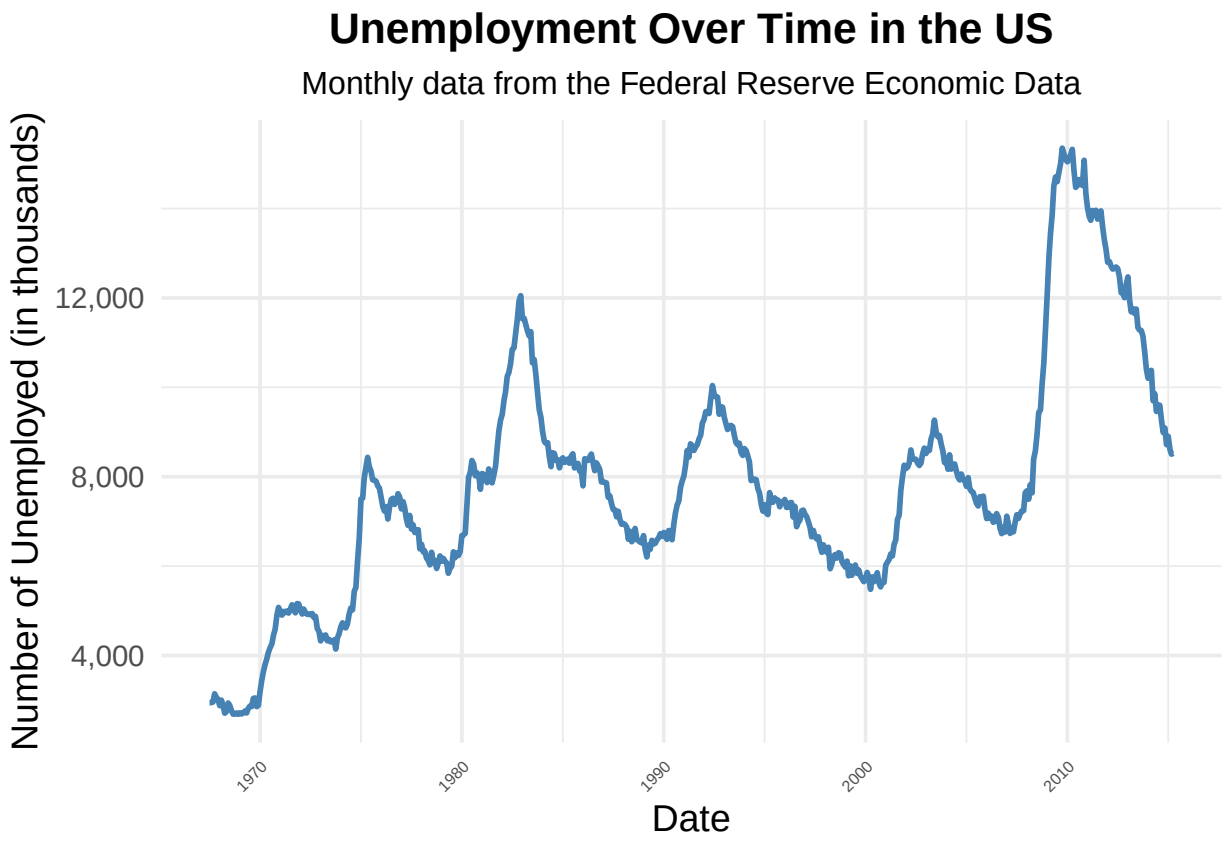
Could you guess how to represent a line plot with `ggplot2` syntax?

- Represent how `unemploy` variable changes over time (`date` variable) from `economics` data frame with a line plot using `ggplot2` syntax
- Modify axis and legend names and add a title
- Save the plot to a file using a raster image format

```r
p <- ggplot(data = economics, aes(x = date, y = unemploy)) +
  geom_line(color = "steelblue", size = 1) +  # Enhanced line color and thickness
  labs(
    title = "Unemployment Over Time in the US",
    subtitle = "Monthly data from the Federal Reserve Economic Data",
    x = "Date",
    y = "Number of Unemployed (in thousands)"
  ) +
  theme_minimal(base_size = 14) +  # Cleaner theme with larger base font
  theme(
    plot.title = element_text(face = "bold", size = 16, hjust = 0.5),  # Centering th
    plot.subtitle = element_text(size = 12, hjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1, size= 6)  # Tilt x-axis labels
  ) +
  scale_y_continuous(labels = scales::comma)  # Add comma separators to large numbers
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
# Print the plot
print(p)
```




```
# Save the plot
ggsave(filename = "unemployment_plot_improved.png", plot = p, width = 8, height = 5)
```

0.7.0.0.1 Answer: