Plotting with the Tidyverse

Deborah Figueiredo Nacer de Oliveira

2022-06-09

Contents

Introduction

Now that you have seen some plot examples and are more comfortable working with data in R, it is time to understand better how it all works with ggplot2. As always, we start by loading the tidyverse package and by setting our work directory. We will use two data sets you have seen before (flights and diamonds), as well as the built-in iris data. Go back to the previous material if you don't remember which command to use to take a glimpse at their structure.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
setwd("/Users/de0580ol/PhD/Basic R course/basic-R-course/")
flights <- read_tsv("data/nycflights13_flights.txt")
```

```
## Rows: 336776 Columns: 19
```

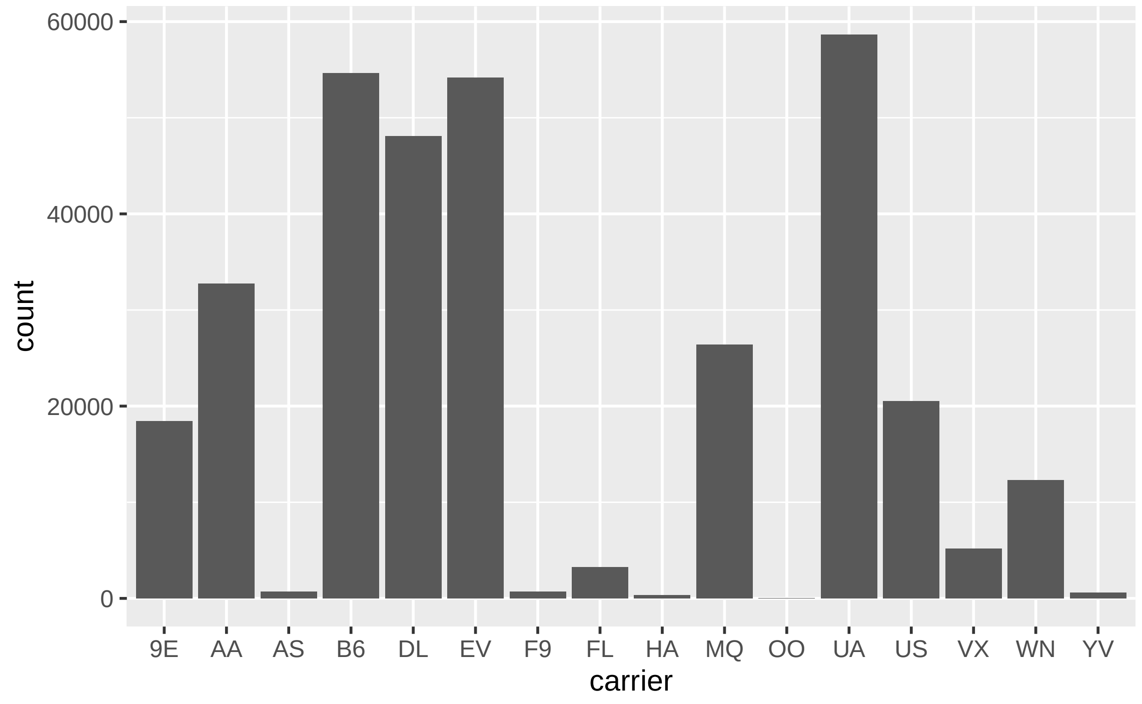
```
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr   (4): carrier, tailnum, origin, dest
## dbl  (14): year, month, day, dep_time, sched_dep_time, dep_delay, arr_time, ...
## dttm  (1): time_hour
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Basic syntax

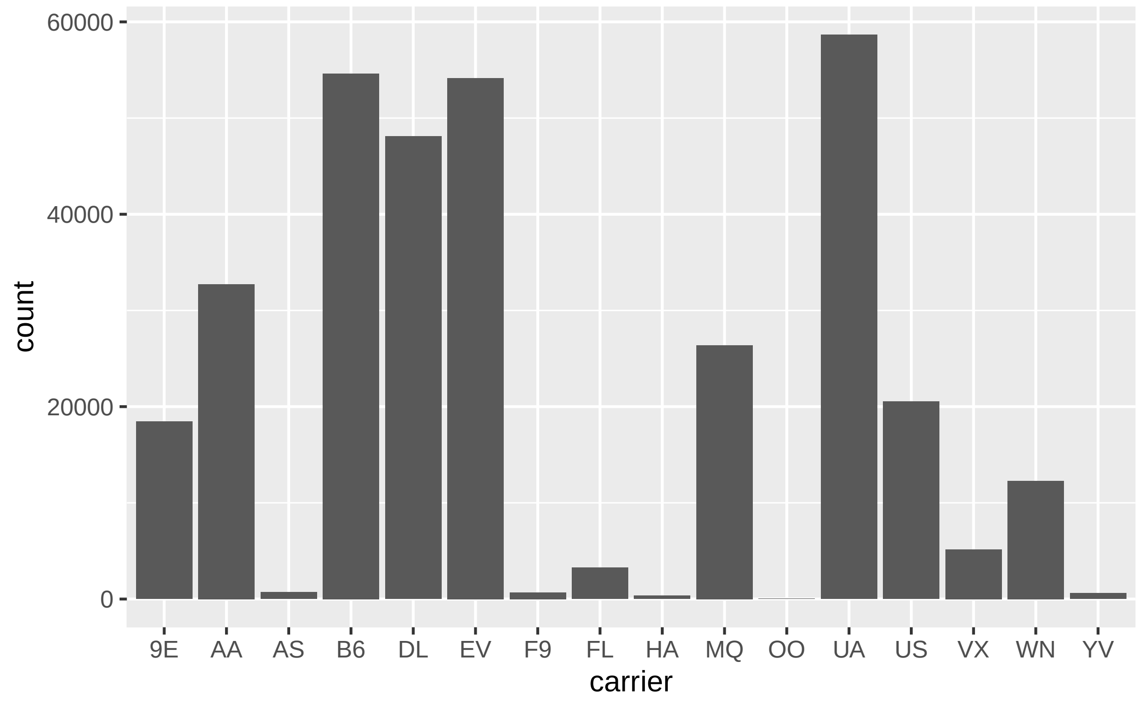
As you have seen, getting a basic plot is very easy and you can do it with as little as two lines of code. However, the package we are using lets you change almost every aspect of your plot, so you might end up with dozens of lines of code for just one plot that looks exactly as you want it.

Take a look at the examples below where we are plotting how many flights each carrier made in 2013. Do not simply look at the plots, see how the code is written.

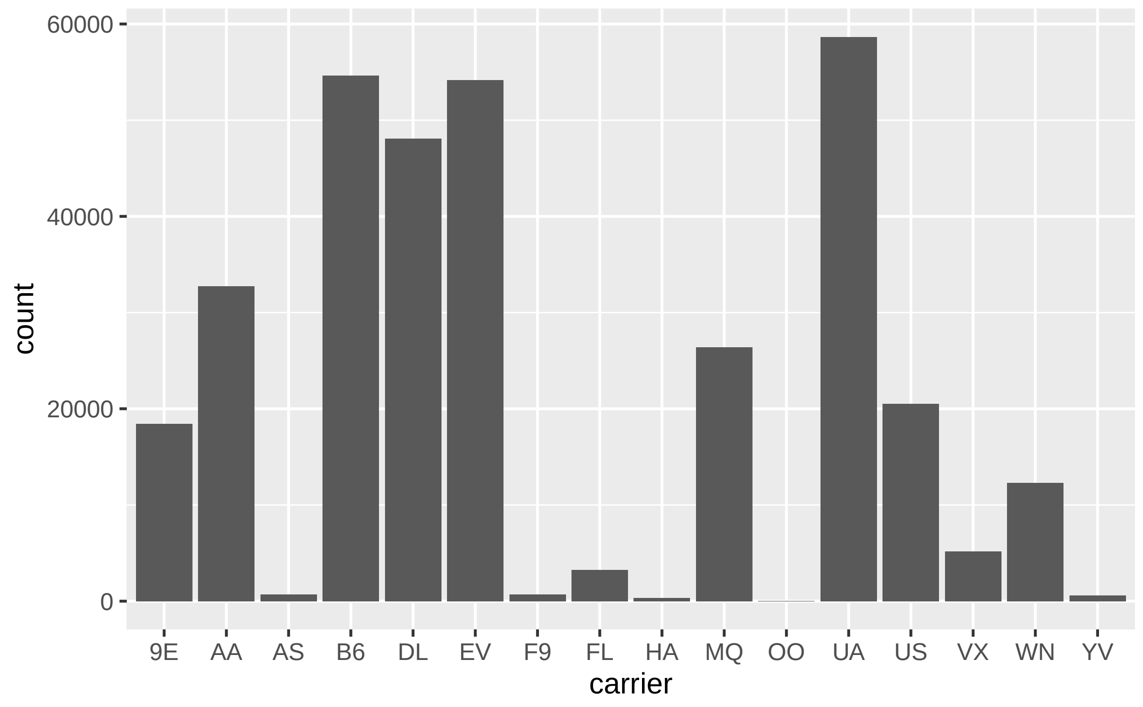
```r
ggplot(data=flights, aes(x=carrier)) +
  geom_bar() # 1
```



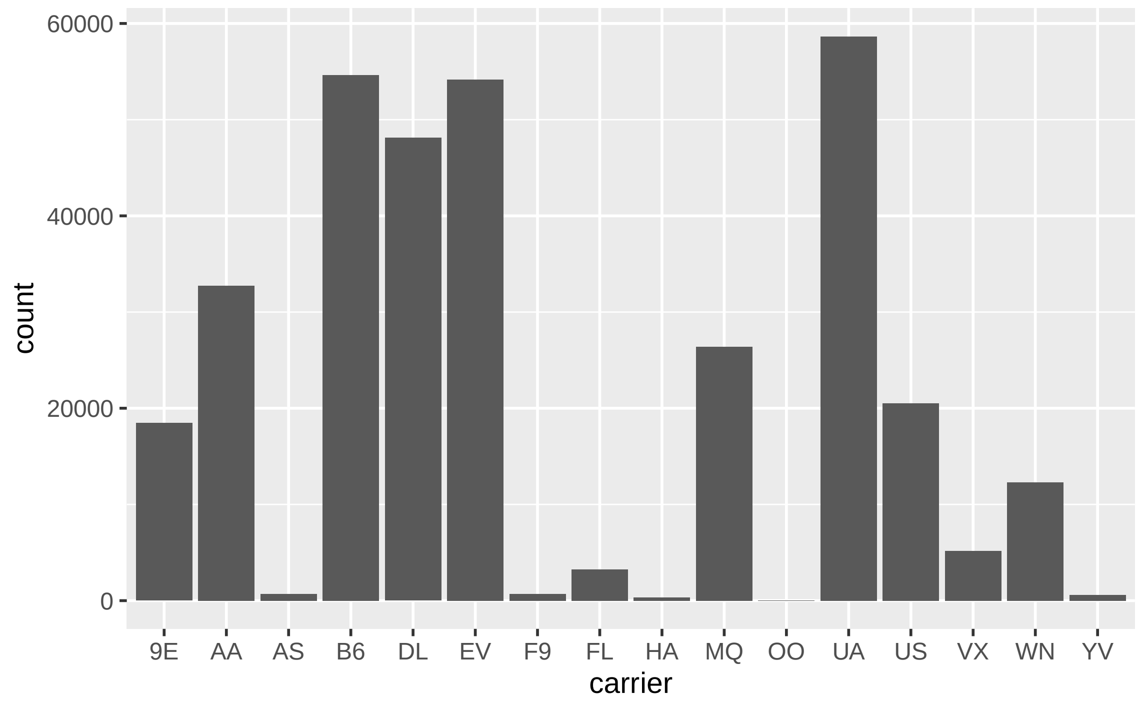
```r
ggplot(data=flights) +
  geom_bar(aes(x=carrier)) # 2
```



```
ggplot() +
  geom_bar(data=flights, aes(x=carrier)) # 3
```



```
flights %>%
  ggplot(aes(x=carrier)) +
  geom_bar() # 4
```



As you can see, all plots look the same and there are many ways of passing data in your code. You can choose which one you like best, but I generally use the last one. I find it is very easy to see what is happening to the data before it is plotted.

The variables to be displayed in the x and y axes are given with the aesthetics function, `aes()`. You can also see in the examples that subsequent layers (the geoms) inherit the information given in the first ggplot line, that is, they will use the same data, x, y, and other information you set with the `ggplot()` function unless you change it directly in the geom. If you haven't figured out this yet, geoms are how you choose which plot type should be used for displaying the data. Those were bar plots because we used `geom_bar()`.

Common **mistakes** here include forgetting to add `aes()` around the variable names, using quotes around variable names, and using '%>%' instead of '+' when adding layers within the ggplot code. If you can't plot something during this course and the error message didn't help you much, start debugging by checking these before you call a teaching assistant.

Exercises

1. Make a barplot with 'origin' in the x axis instead of 'carrier'. Try different ways of passing data.

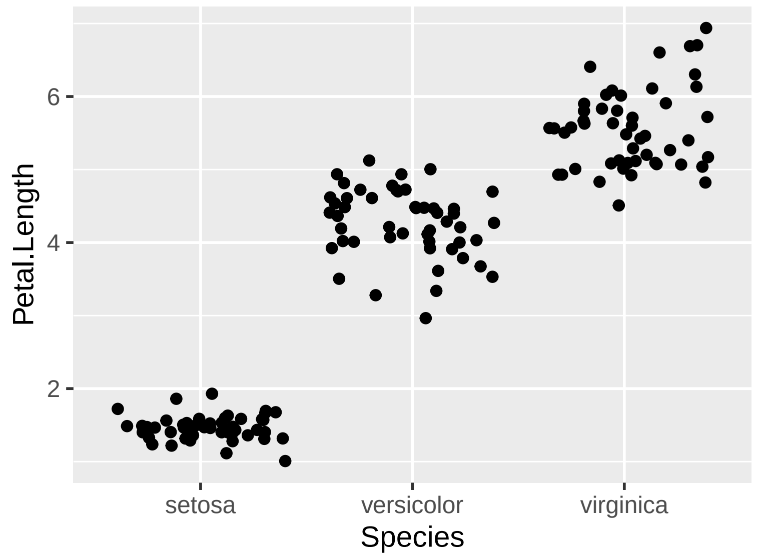
Geometric elements (geoms)

Different plots (and therefore geoms) need different information in the code: some like histograms only need one variable, while others like boxplots need both x and y variables. Try the different plots below to have a taste of how this works. It is important to know that you can use more than one geom in a plot. The iris data set is new in the course, but it is quite simple as it only contains different flower species and sepal/petal measurements.

If you have no idea which plot is good for your type of data, take a look at the from Data to Viz website and see some options based on variable types. Once you have decided on a plot type, see code examples in the R Graph Gallery. Always think about exactly what information you want readers to see in your plot and perhaps try some options to find the best way of showing it.

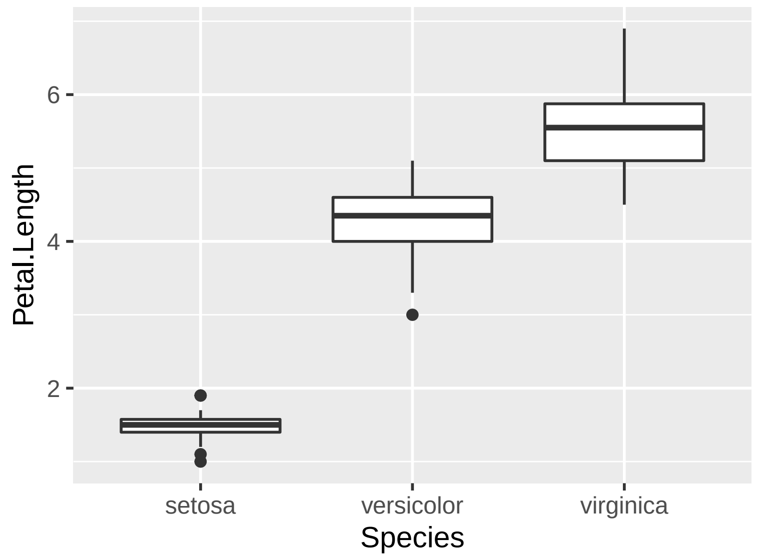
Jitter

```
iris %>%
  ggplot(aes(x=Species, y=Petal.Length)) +
  geom_jitter()
```



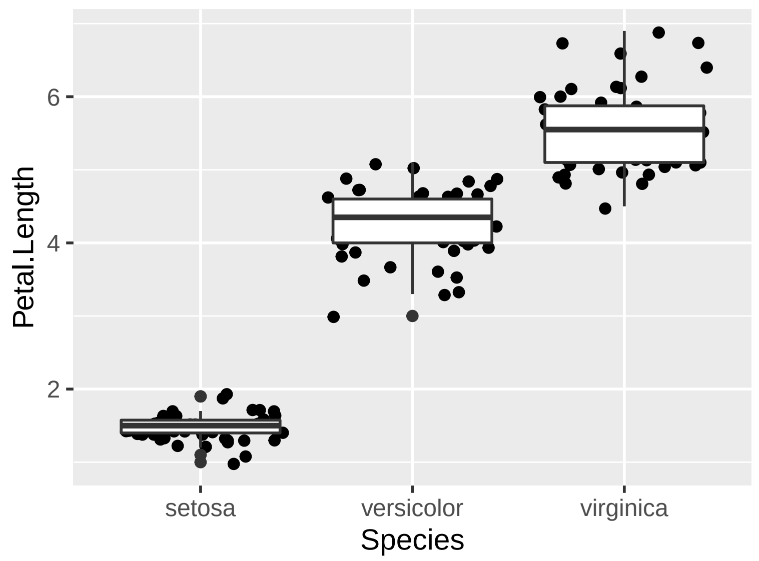
Boxplot

```
iris %>%
  ggplot(aes(x=Species, y=Petal.Length)) +
  geom_boxplot()
```



Jitter + Boxplot

```
iris %>%
  ggplot(aes(x=Species, y=Petal.Length)) +
  geom_jitter() +
  geom_boxplot()
```



Notice how both geoms inherited the x and y aesthetics as given in the beginning.

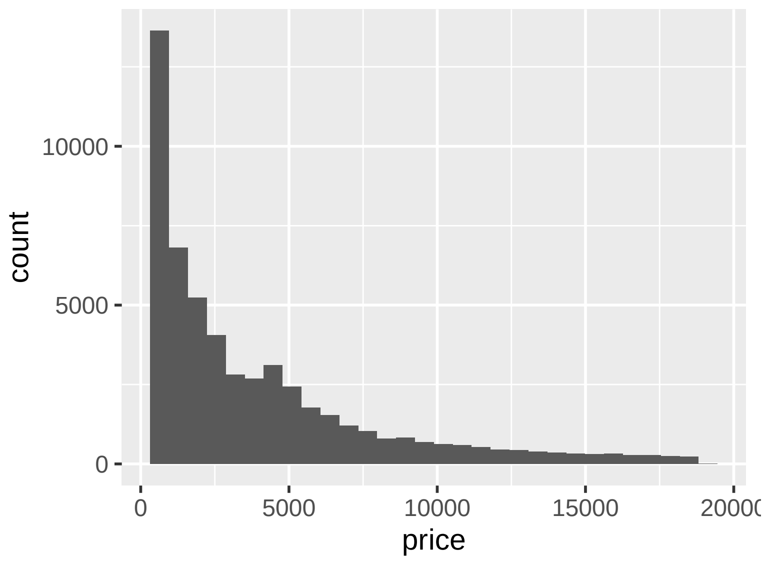
Exercises

2. Try inverting the order of the jitter and boxplot geoms in the code. Does geom order matter?

Histogram

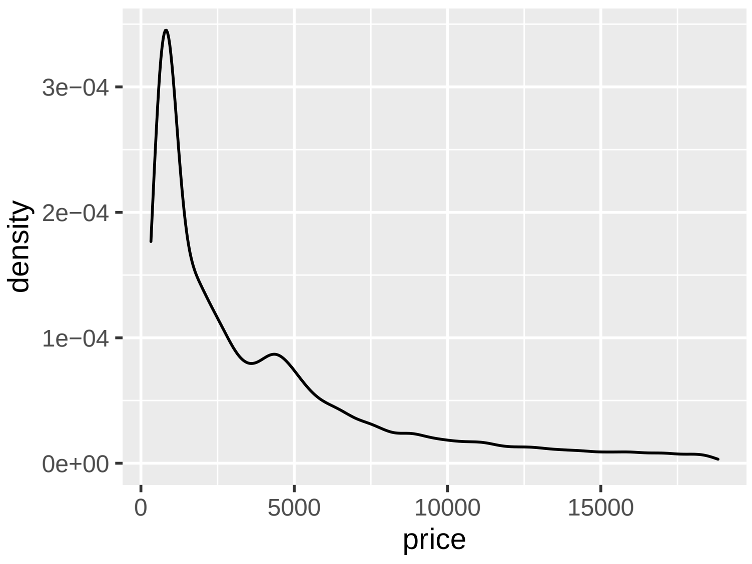
```
diamonds %>%
  ggplot(aes(x=price)) +
  geom_histogram()
```

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



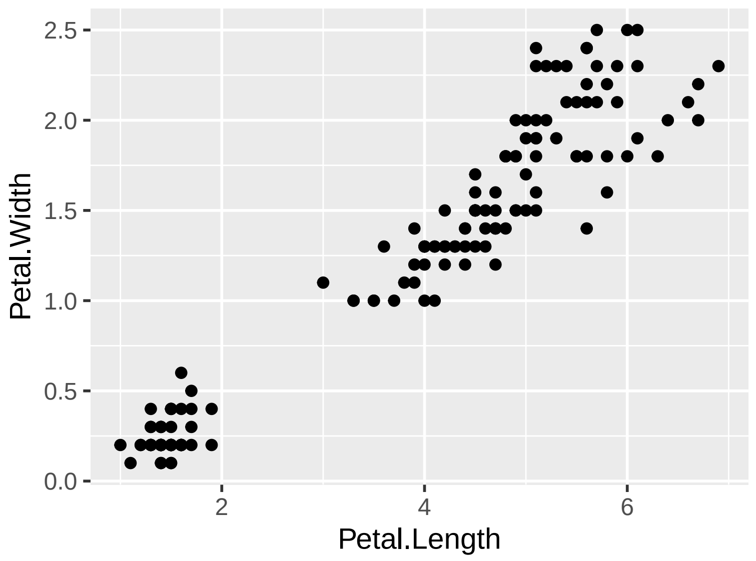
Density plot

```
diamonds %>%
  ggplot(aes(x=price)) +
  geom_density()
```



Scatter plot

```
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width)) +
  geom_point()
```



Exercises

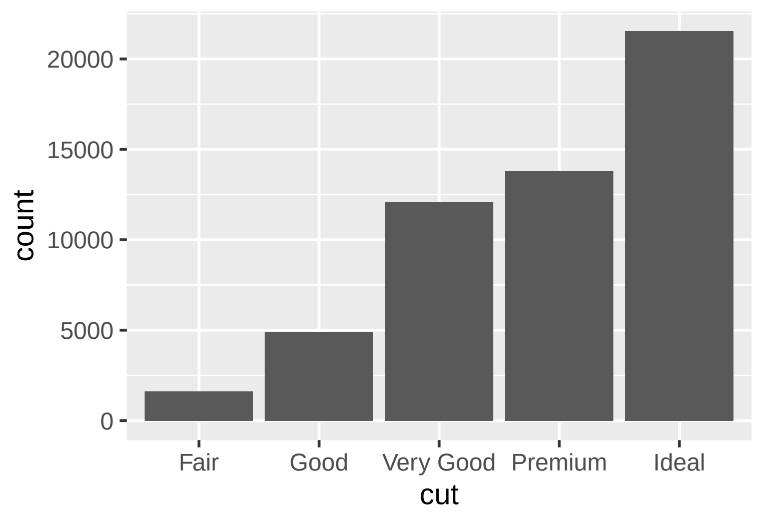
3. What geom is used for adding a horizontal line to a plot? And a vertical line? Add a horizontal line at y = 0.75 and a vertical line at x = 2.5.

Pie chart

Just kidding, we will not learn pie charts in this course. Why, you ask? Because humans are bad at estimating quantity from angles, and it is even worse if you are comparing slices in different pies. Try any of the plots below instead, they are better at conveying the same information.

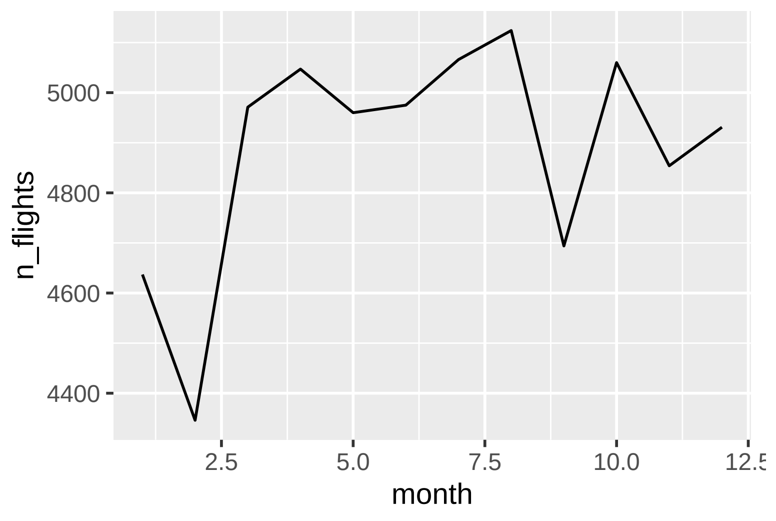
Barplot

```
diamonds %>%
  ggplot(aes(x=cut)) +
  geom_bar()
```



Line chart

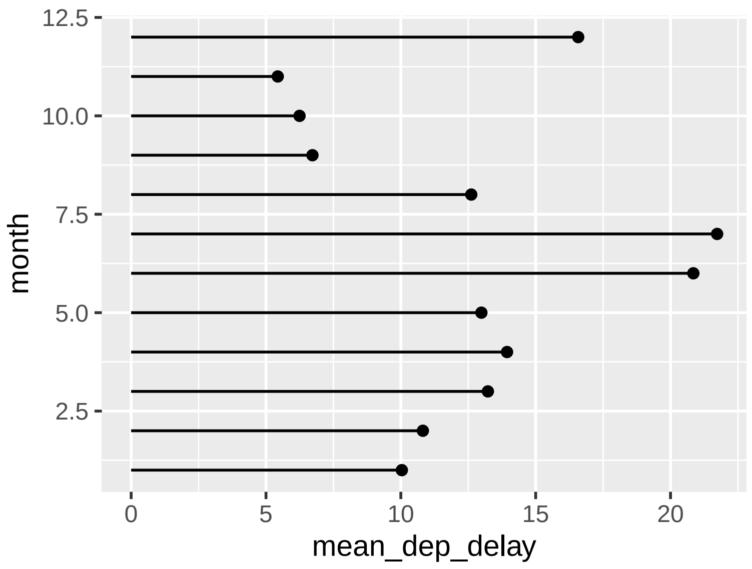
```
flights %>%
  filter(carrier == 'UA') %>%
  group_by(month) %>%
  summarise(n_flights=n()) %>%
  ggplot(aes(x=month, y=n_flights)) +
  geom_line()
```



Notice here how we first generated the data we wanted to plot (a summary of how many flights per month the carrier UA did) by piping the `filter()`, `group_by()` and `summarise()` commands, and only then passed the result to `ggplot()`.

Lollipop

```
flights %>%
  drop_na(dep_delay) %>%
  group_by(month) %>%
  summarise(mean_dep_delay = mean(dep_delay, na.rm = T)) %>%
  ggplot(aes(x=mean_dep_delay, y=month)) +
  geom_segment(aes(x=0, xend=mean_dep_delay, y=month, yend=month)) +
  geom_point()
```



Another example of inheritance can be seen in this lollipop plot. Notice how `geom_point()` inherited x and y from `ggplot()`, but we set different parameters for `geom_segment()`.

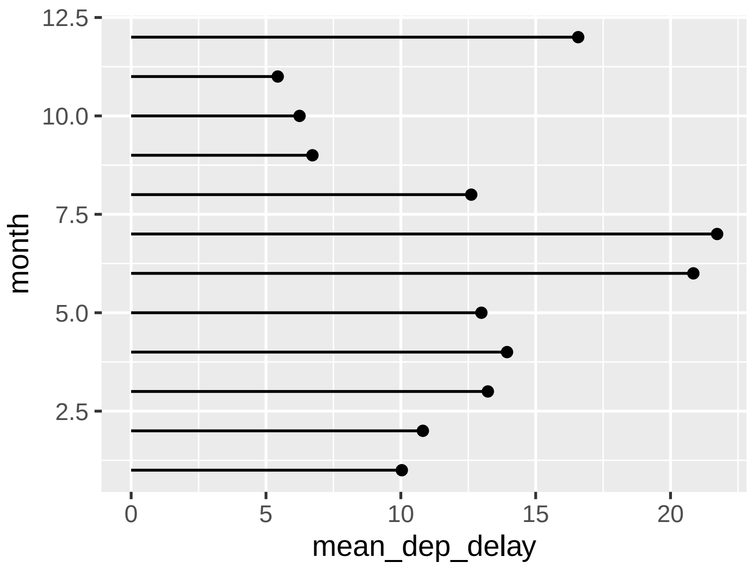
Different geoms have different parameters you can change and we will go through some in this course. For a full list for any specific geom, look for their function name in the Help section or write e.g., `?geom_bar` in the Console.

Heatmaps are also often used in science, but we will not cover it in this course. If that interests you, take a look at ggplot's `geom_tile()` or other packages such as pheatmap and ComplexHeatmap.

Variable type

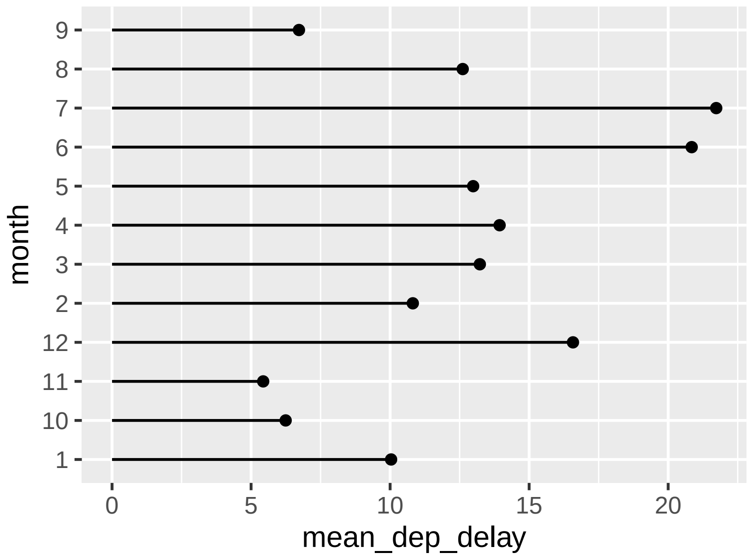
Knowing what are the variables' types is also very important when plotting since you can have very different results according to it. Try the examples below to see the differences.

```
# Mean departure delay by month with month as numeric
flights %>%
  group_by(month) %>%
  summarise(mean_dep_delay = mean(dep_delay, na.rm = T)) %>%
  mutate(month = as.numeric(month)) %>%
  ggplot(aes(x=mean_dep_delay, y=month)) +
  geom_segment(aes(x=0, xend=mean_dep_delay, y=month, yend=month)) +
  geom_point()
```



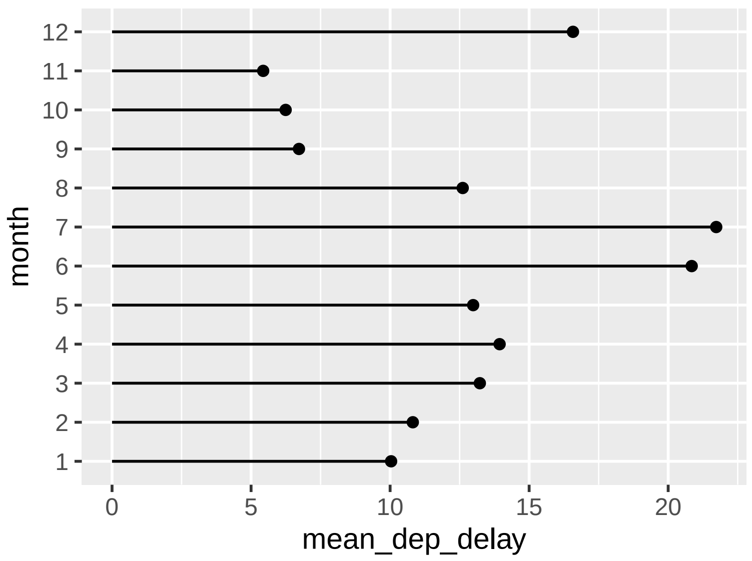
Numbers on the y axis don't correspond to what we want to show as the breaks were automatically arranged.

```
# with month as character
flights %>%
  group_by(month) %>%
  summarise(mean_dep_delay = mean(dep_delay, na.rm = T)) %>%
  mutate(month = as.character(month)) %>%
  ggplot(aes(x=mean_dep_delay, y=month)) +
  geom_segment(aes(x=0, xend=mean_dep_delay, y=month, yend=month)) +
  geom_point()
```



Now each month is annotated as we would want, but the order is not what you would expect for a month variable. Notice that the automatic sorting order for characters in ggplot is alphanumeric (arranged by the first numeral, so 12 comes before 2).

```
# with month as factor
flights %>%
  group_by(month) %>%
  summarise(mean_dep_delay = mean(dep_delay, na.rm = T)) %>%
  mutate(month = as.factor(month)) %>%
  ggplot(aes(x=mean_dep_delay, y=month)) +
  geom_segment(aes(x=0, xend=mean_dep_delay, y=month, yend=month)) +
  geom_point()
```



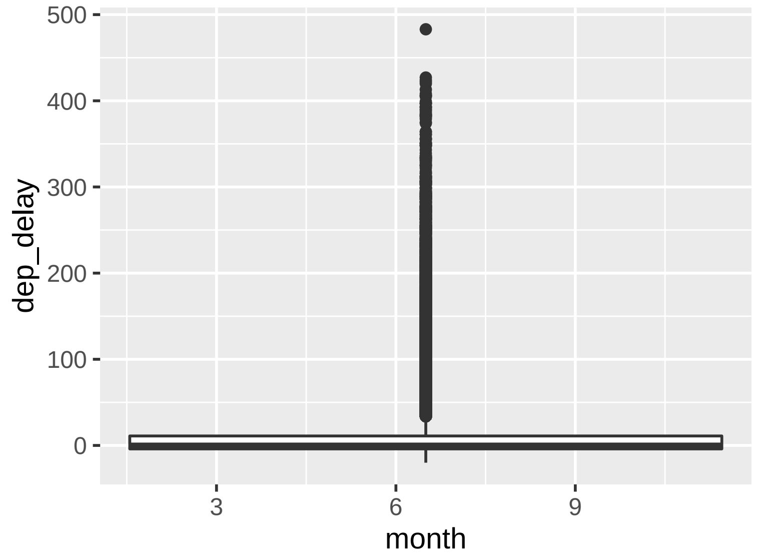
This last plot is closer to what we want. Again, do not just run this code and move on. Look at each line and understand what it means before proceeding. Run the code in steps if you need. Call a teaching assistant if you get lost. Tidyverse gives you all the freedom in the world to do whatever you want with data and plot it however you wish it if you only ask nicely.

Exercises

4. Fix the code below to plot departure delay distribution by month. Notice how R is nice and gives you a warning.

```
flights %>%
  filter(carrier == 'UA') %>%
  drop_na(dep_delay) %>%
  ggplot(aes(x=month, y=dep_delay)) +
  geom_boxplot()
```

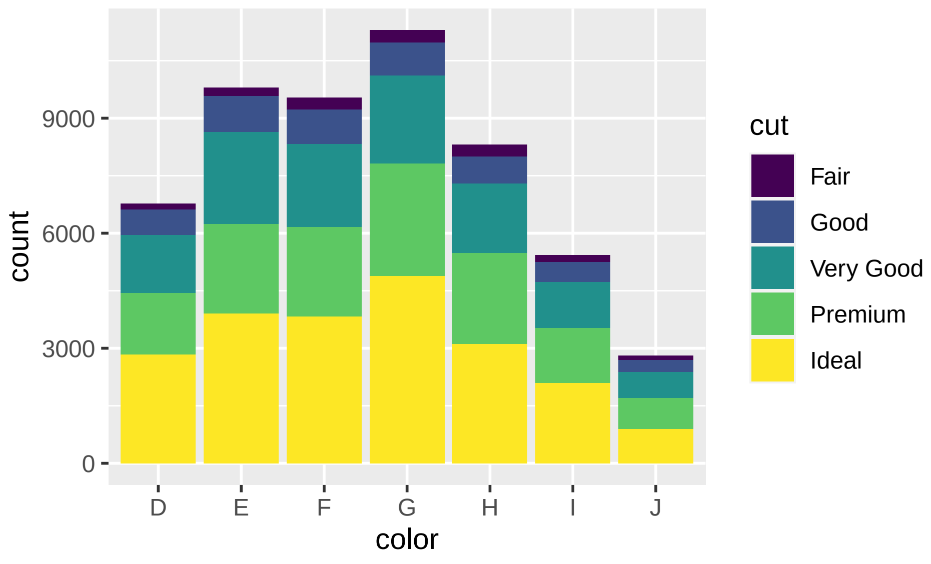
```
## Warning: Continuous x aesthetic -- did you forget aes(group=...)?
```



Color, size, and shape

We have been plotting data without changing many parameters, but of course you can (and many times should) have different colors/shapes/sizes in a plot to make clear whatever you want to show. For color, be aware that there are both a `color` and a `fill` parameter you can change depending on the geom. Also, ggplot accepts both British and American spellings of the word color.

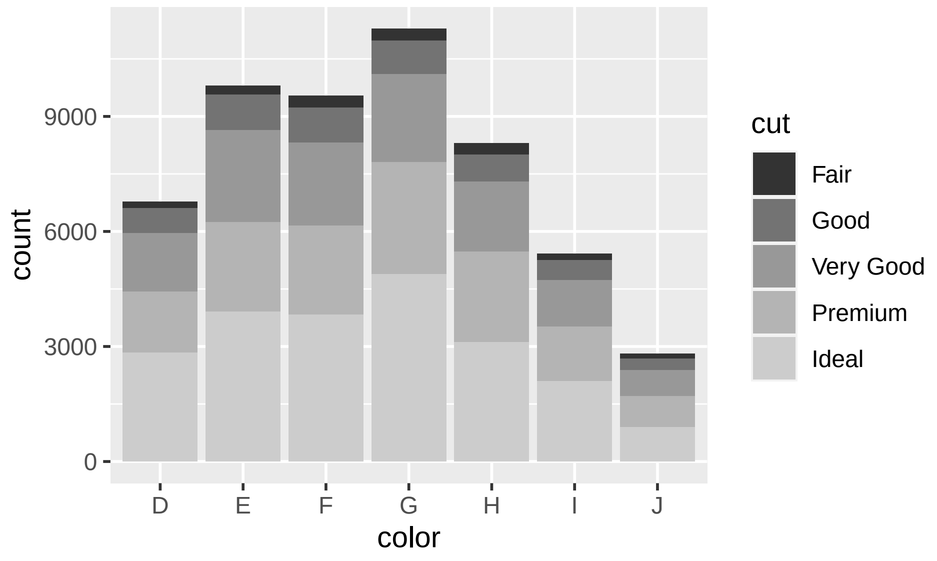
```
## Color
diamonds %>%
  ggplot(aes(x=color, fill=cut)) +
  geom_bar()
```



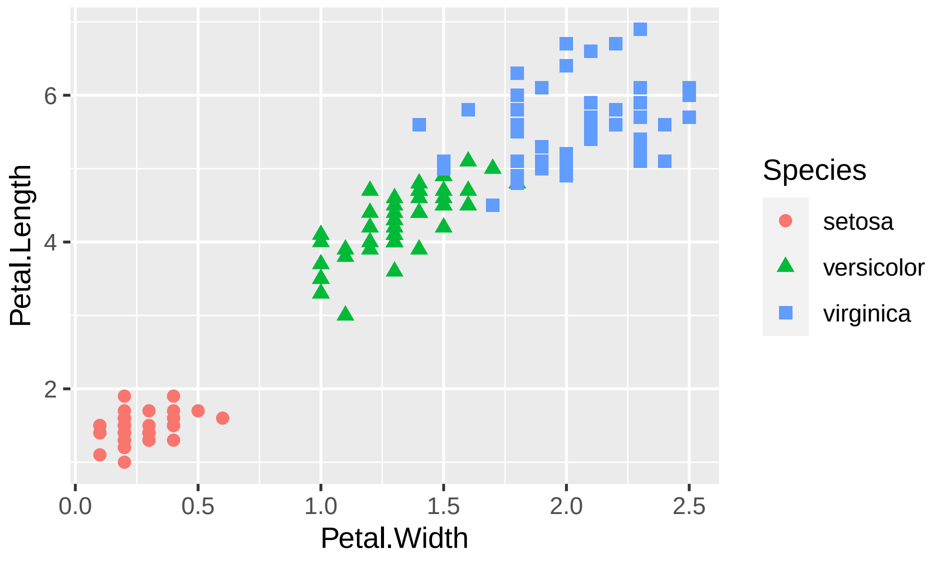
Exercises

5. Substitute the argument 'fill' with 'color' to see what happens.

```r
diamonds %>%
  ggplot(aes(x=color, fill=cut)) +
  geom_bar() +
  scale_fill_grey()
```



```r
## Shape and size
iris %>%
  ggplot(aes(x=Petal.Width, y=Petal.Length, color = Species, shape = Species)) +
  geom_point(size = 2)
```



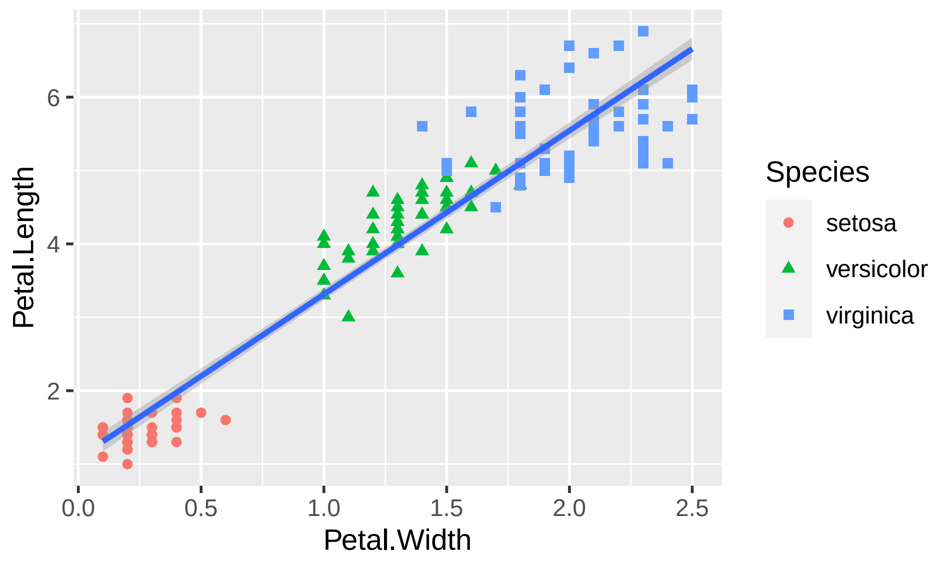
Common **mistakes** here include setting the chosen color outside `aes()` and setting the chosen color in the fill argument when it should have been in the color argument (or the other way around).

Exercises

6. Fix the code below so you have one regression line per species. Make it with as few command lines as possible.

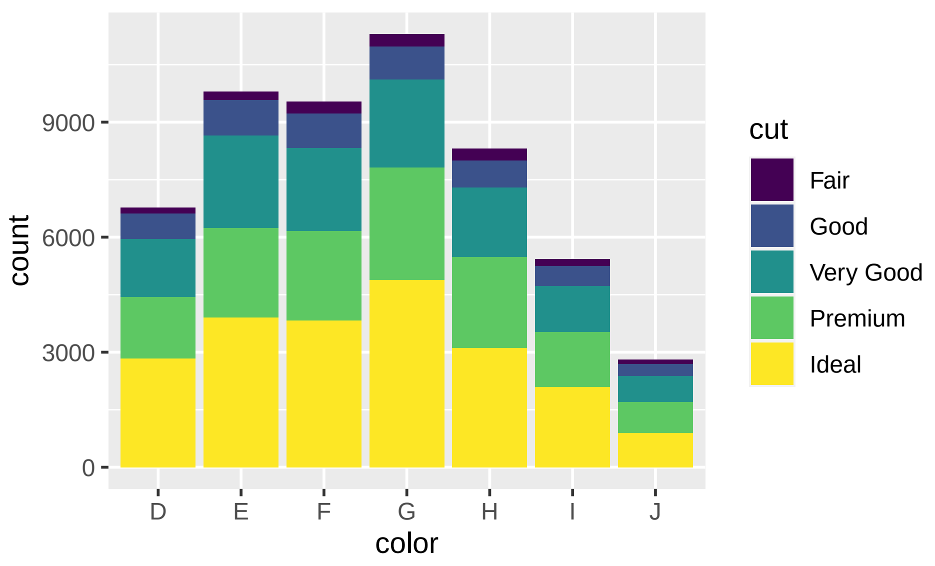
```
iris %>%
  ggplot(aes(x=Petal.Width, y=Petal.Length)) +
  geom_point(aes(color = Species, shape = Species)) +
  geom_smooth(method="lm")
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```



7. Change the barplot below to take ‘dodge’ as value for the argument ‘position’ in `geom_bar()`. Change it again to take ‘fill’ as value for the same argument. See what changed. If you don’t understand this question, read the Usage information in the Help section.

```
diamonds %>%
  ggplot(aes(x=color, fill=cut)) +
  geom_bar()
```



8. Change the colors in a plot using a viridisLite function such as `scale_color_viridis_d()`. Choose the 'magma' option.

9. Set colors manually using `scale_fill_manual()`. Do it by order first, then change the code to have specific colors to specific variable values such as 'Fair' = 'deepskyblue'. Use this online pdf to pick color names.

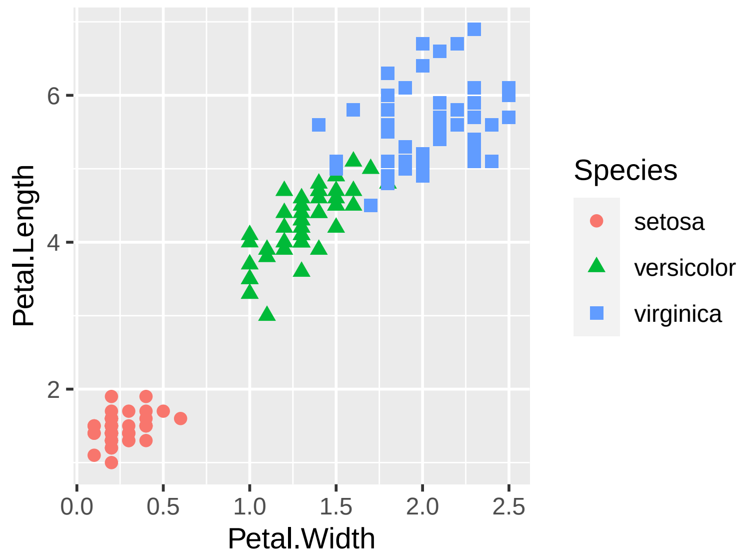
10. Curious to see how this plot would look if the variables were in the other axes? Try flipping it using `coord_flip()`.

Quick words on the use of color. It is true what they say that less is more. It is very easy to get excited and make a plot super colorful, but remember that colors are supposed to help the visualization, not distract you from the information you are trying to convey. Color theory is a course in itself and there is much you can read online. For now, you can use one of the many color palettes already out there - take a look at for instance viridisLite and RColorBrewer. Lastly, avoid using a red and green combination - lots of people are colorblind (and what do you think your plot is, Christmas?).

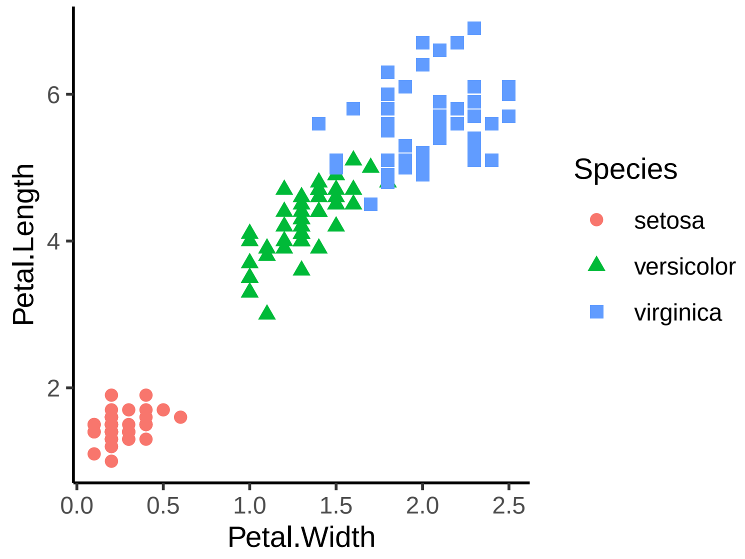
Themes

There are several built-in themes in ggplot. Try some with the code below and choose the one you like the best.

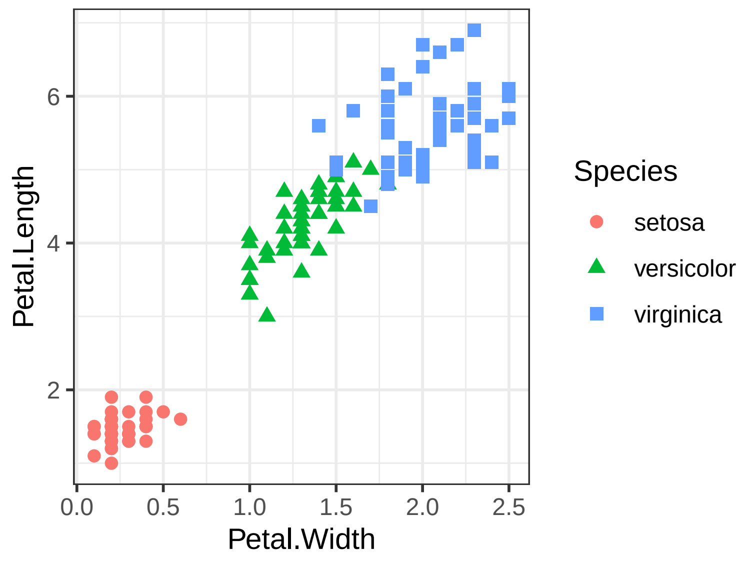
```
iris %>%
  ggplot(aes(x=Petal.Width, y=Petal.Length, color = Species, shape = Species)) +
  geom_point(size = 2) +
  theme_grey() # the default theme
```



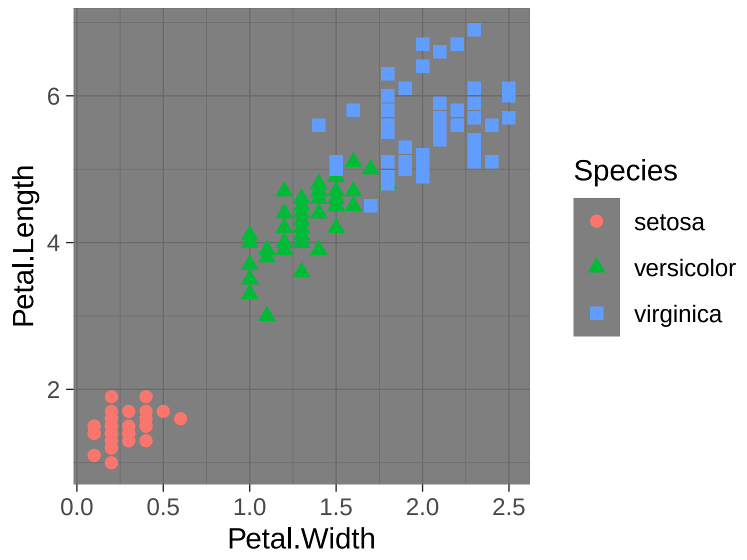
```
iris %>%
  ggplot(aes(x=Petal.Width, y=Petal.Length, color = Species, shape = Species)) +
  geom_point(size = 2) +
  theme_classic()
```



```r
iris %>%
  ggplot(aes(x=Petal.Width, y=Petal.Length, color = Species, shape = Species)) +
  geom_point(size = 2) +
  theme_bw()
```



```r
iris %>%
  ggplot(aes(x=Petal.Width, y=Petal.Length, color = Species, shape = Species)) +
  geom_point(size = 2) +
  theme_dark()
```



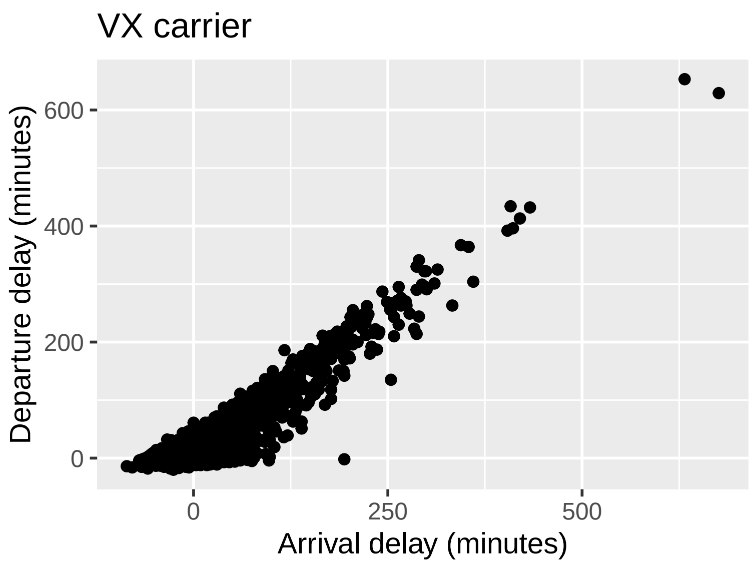
Exercises

11. There are other options. Try `theme_nothing()` and another one of your choosing.

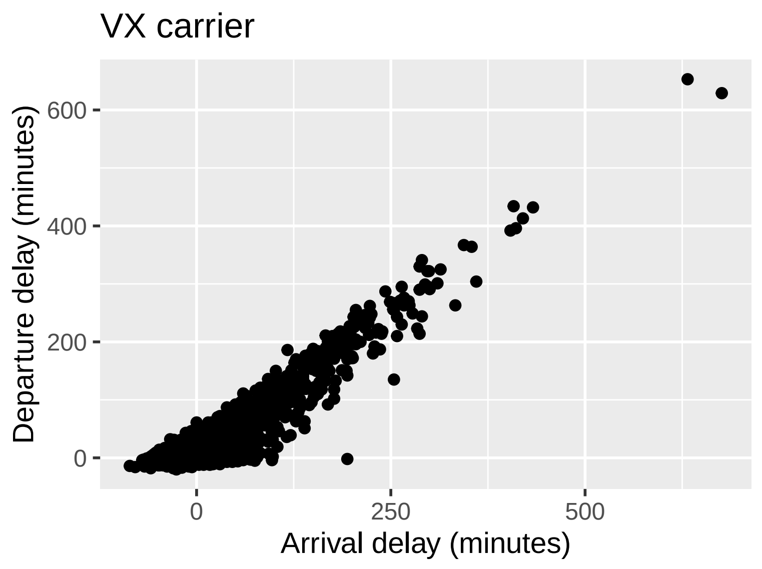
Axes

Now that you know how to make the theme and colors of the plot look nicer, we go into the text part. The most straightforward to change axes' labels is using `labs()`, but we show a second way as well.

```
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(x = "Arrival delay (minutes)",
       y = "Departure delay (minutes)",
       title = "VX carrier")
```



```
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(title = "VX carrier") +
  scale_x_continuous(name="Arrival delay (minutes)") +
  scale_y_continuous(name="Departure delay (minutes)")
```



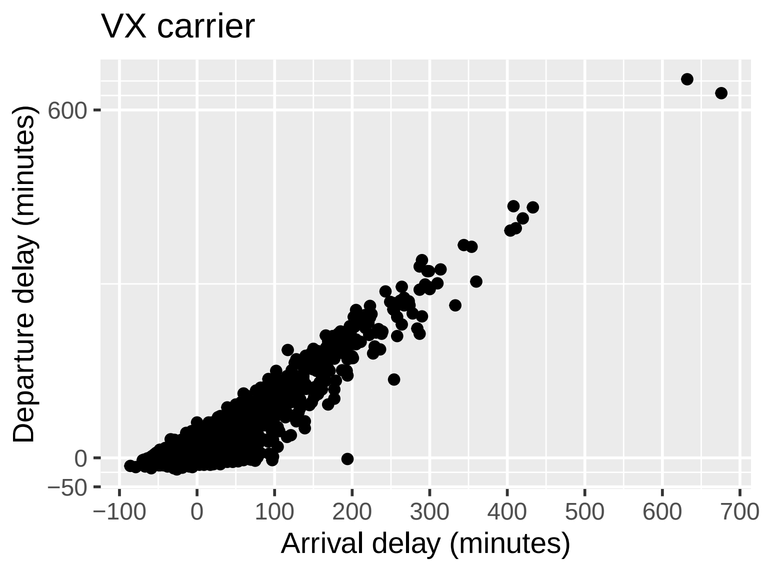
Exercises

12. Let's assume you would like to have '(minutes)' on a new line instead of how it is now. Do it by substituting the space just before that word with '\n'.

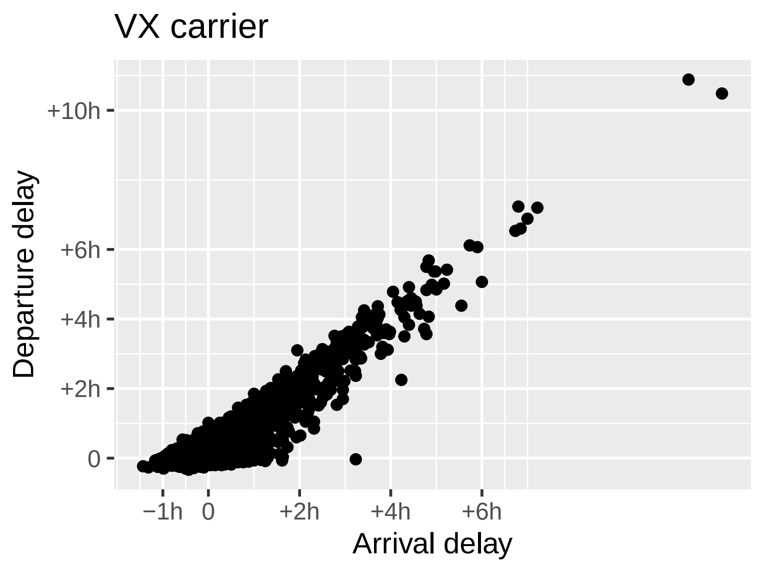
13. For the second way we showed, it matters whether your numeric variable is discrete or continuous. Try changing it to `scale_x_discrete()` and see what happens.

You can also change the breaks on the axes and their labels. Notice how the white lines on the background of the plot change according to the breaks you choose.

```
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(title = "VX carrier") +
  scale_x_continuous(name="Arrival delay (minutes)",
                     breaks=seq(-100,700,100)) +
  scale_y_continuous(name="Departure delay (minutes)",
                     breaks=c(-50,0,600))
```



```
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(title = "VX carrier") +
  scale_x_continuous(name="Arrival delay",
                     breaks=c(-60,0,120,240,360),
                     labels=c('-1h', '0', '+2h', '+4h', '+6h')) +
  scale_y_continuous(name="Departure delay",
                     breaks=c(-60,0,120,240,360,600),
                     labels=c('-1h', '0', '+2h', '+4h', '+6h', '+10h'))
```

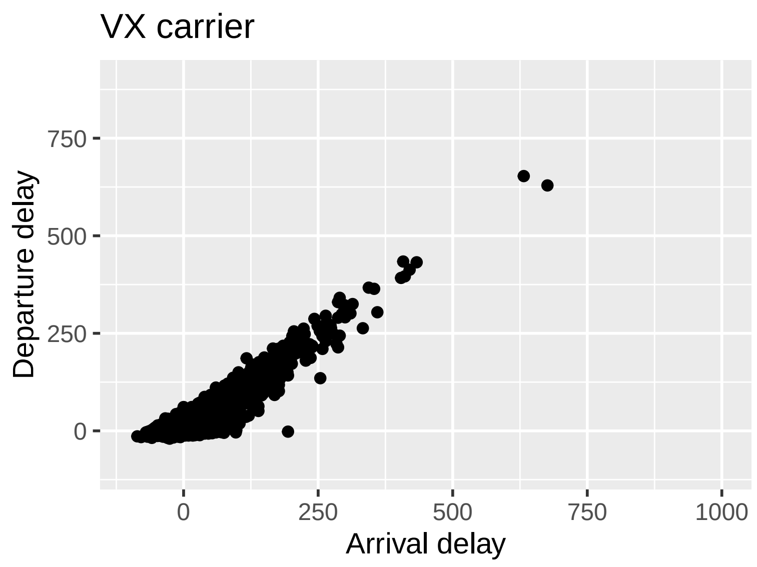


Exercises

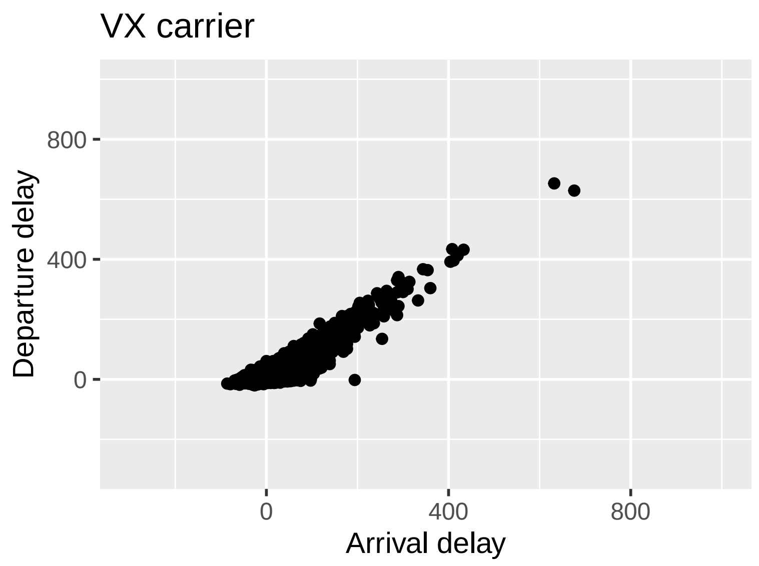
14. Why did we use `seq()`? Use the Help to understand this base R function if you have never seen it before.

You might want to extend a plot to show a certain axis break even though there are no observations with that value. This is easily done by using `lims()` or the 'limits' argument within scale.

```
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(title = "VX carrier",
       x = "Arrival delay",
       y = "Departure delay") +
  lims(x = c(-100, 1000),
       y = c(-100, 900))
```



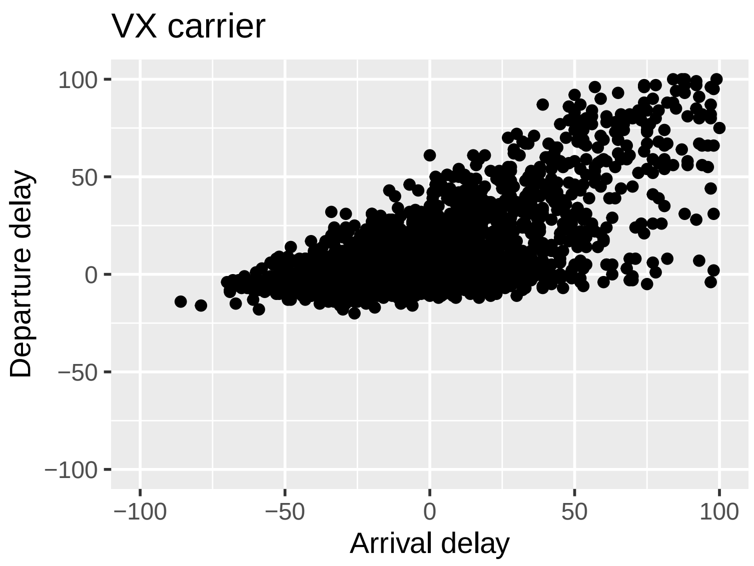
```
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(title = "VX carrier") +
  scale_x_continuous(name="Arrival delay",
                     limits=c(-300,1000)) +
  scale_y_continuous(name="Departure delay",
                     limits = c(-300,1000))
```



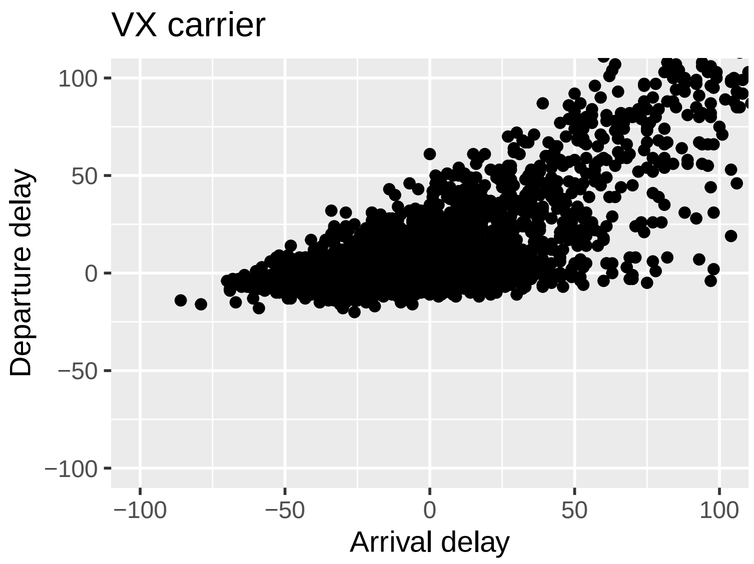
Finally, you might want to zoom in to a specific part of a plot. This should **not** be done with limits as observations outside the limits would turn to NA changing the distribution and statistics associated with the data. To zoom in properly, use `coord_cartesian()` as shown below. Notice how points above the limits disappear in the first plot but they remain in the second, as they should.

```r
# WRONG way of zooming in, it turns observations outside the limits to NA
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(title = "VX carrier",
       x = "Arrival delay",
       y = "Departure delay") +
  lims(x = c(-100, 100),
       y = c(-100, 100)) # WRONG!
```

```
## Warning: Removed 258 rows containing missing values (geom_point).
```



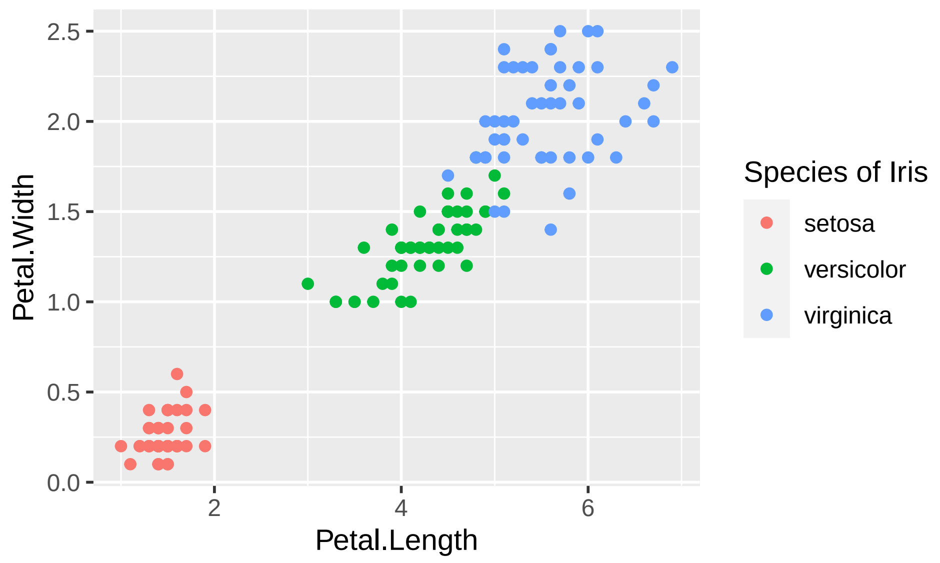
```
# Correct way of zooming in, all observations remain in the plot but are not the focus
flights %>%
  filter(carrier == 'VX') %>%
  drop_na(arr_delay, dep_delay) %>%
  ggplot(aes(x=arr_delay, y=dep_delay)) +
  geom_point() +
  labs(title = "VX carrier",
       x = "Arrival delay",
       y = "Departure delay") +
  coord_cartesian(xlim=c(-100,100), ylim=c(-100,100)) # Correct
```



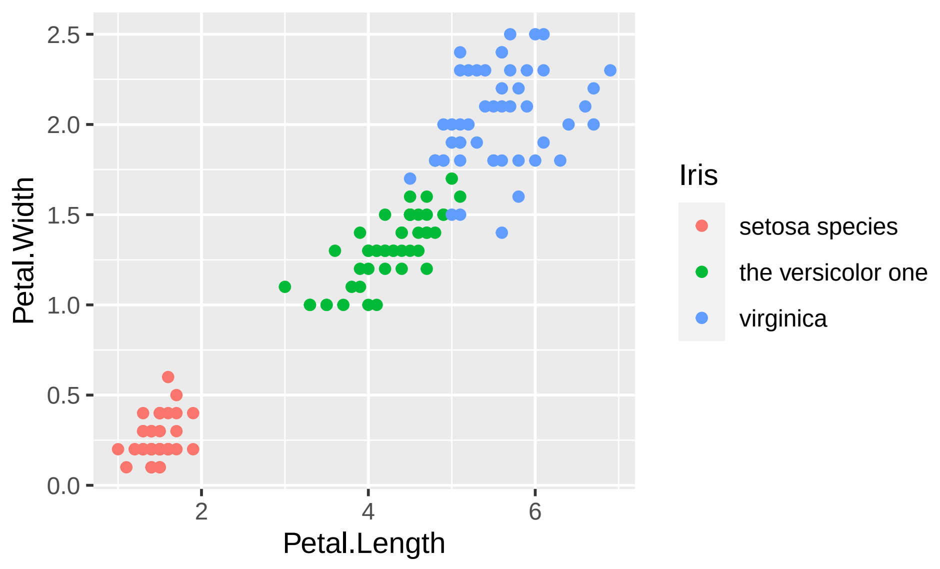
Legends

You will probably want to modify legends as well. Here are some ways of changing their titles and labels, as well as their position. Notice the `theme()` function. You can change a lot in a plot using it, but we won't go into details in this basic course. Feel free to explore if you have the time.

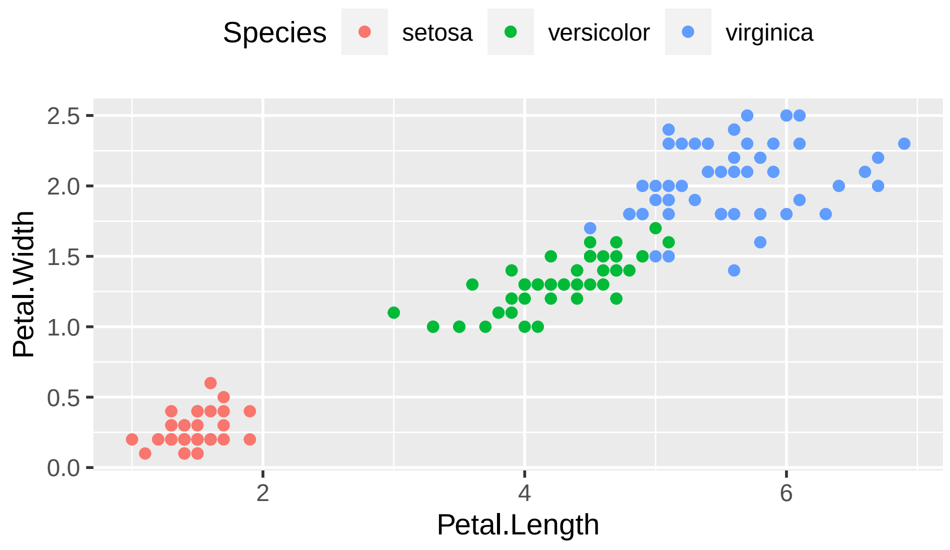
```
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width, color=Species)) +
  geom_point() +
  labs(color = 'Species of Iris')
```



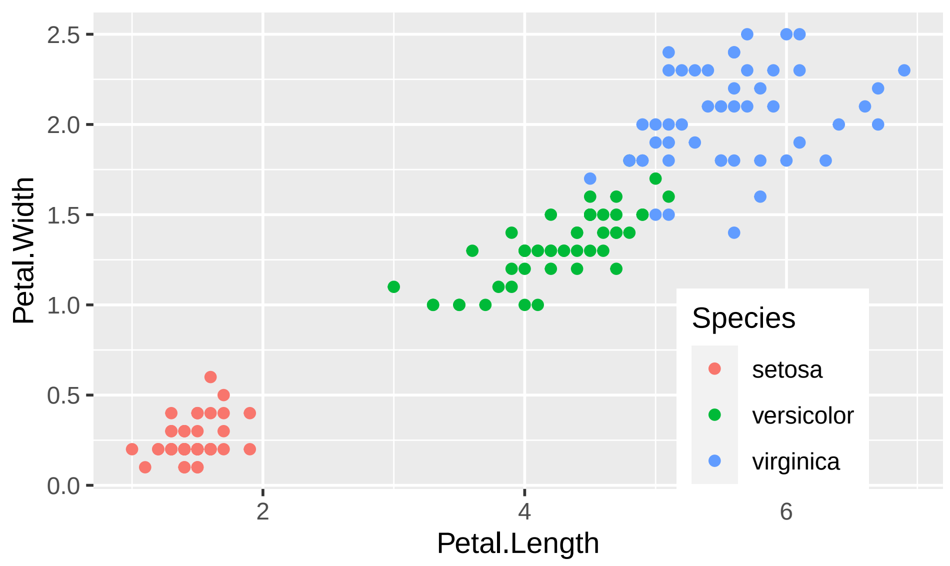
```
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width, color=Species)) +
  geom_point() +
  scale_color_discrete(name = 'Iris',
                       labels = c('setosa' = 'setosa species',
                                  'versicolor' = 'the versicolor one'))
```



```
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width, color=Species)) +
  geom_point() +
  theme(legend.position = 'top')
```



```
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width, color=Species)) +
  geom_point() +
  theme(legend.position = c(0.8,0.2))
```



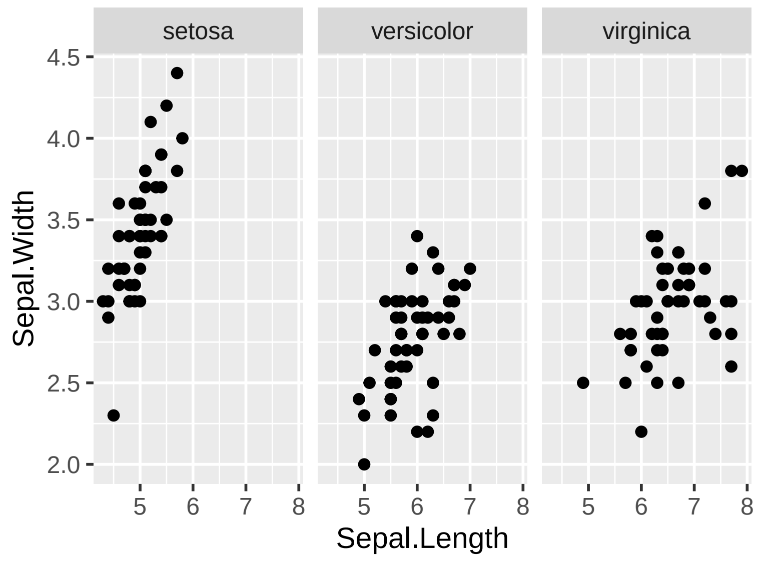
Exercises

15. Change the legend to the bottom of the plot by changing the 'legend.position' argument. Then remove it completely by setting the same argument to 'none'.
16. Move the legend to the upper left corner within the plot.
17. Add an argument to have different point shapes for different species and rename that legend as well.

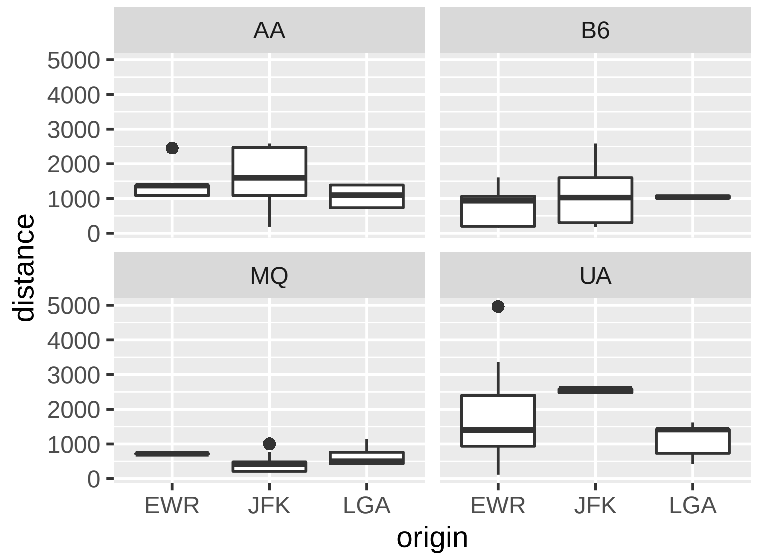
Facets

Sometimes you want to have different plots for different values of a variable. This can be done very easily by faceting. See the examples below.

```
iris %>%
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point() +
  facet_wrap(~Species)
```



```
flights %>%
  filter(carrier %in% c('UA', 'AA', 'B6', 'MQ')) %>%
  ggplot(aes(x=origin, y=distance)) +
  geom_boxplot() +
  facet_wrap(~carrier)
```



Exercises

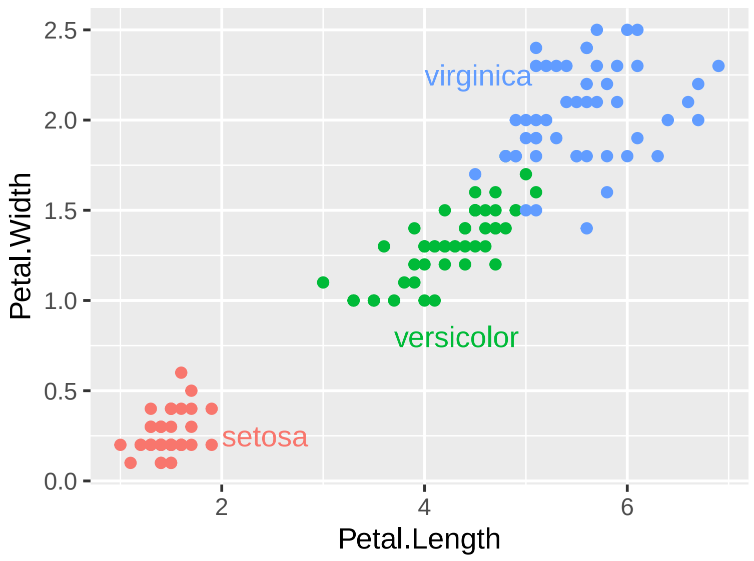
18. Look at the Usage of `facet_wrap()`. Set a different number of rows and/or columns when faceting.
19. By default all plots have the same scale. Make it that the scale in the y axis is free to fit its data.
20. Change the position of the strip on top that contains the variable name.

21. What if you want to facet by two variables instead? Create a scatter plot with the diamonds data set, where Price is in x axis and Carat is in the y axis. Now use `facet_grid()` to split plots by Cut and Color. Not sure how to? Use the Help section.

Annotations

You can add manual annotations to plots using `annotate()`. See the code below where we add species' names directly to the plot instead of in a legend to the side.

```
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width, color=Species)) +
  geom_point() +
  theme(legend.position = 'none') +
  annotate('text',
           x=c(2, 3.7, 4),
           y=c(0.25, 0.8, 2.25),
           label=c('setosa', 'versicolor', 'virginica'),
           color=c("#F8766D", "#00BA38", "#619CFF"),
           hjust=0)
```



Exercises

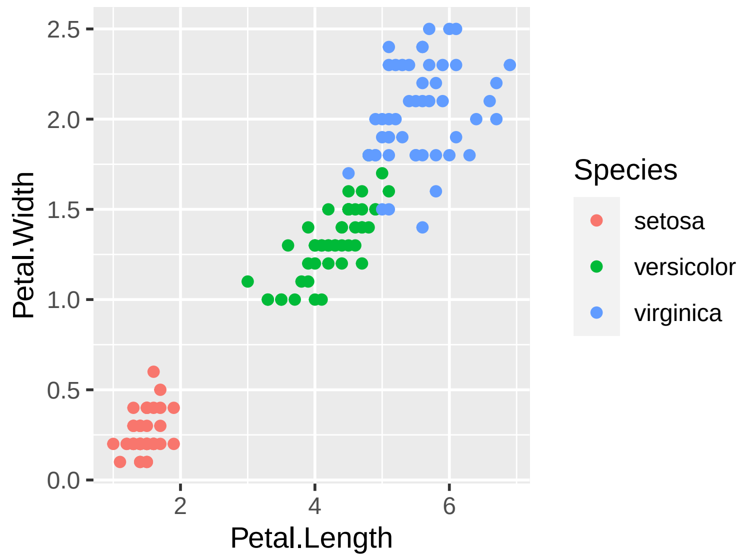
22. You can add more than text to a plot using `annotate()`. Add a red segment to the plot above with whatever coordinates you'd like.

23. Now add a rectangle around the dots belonging to the *setosa* species.

Saving plots

The final point to learn within ggplot basics is how to save a plot you have generated. Here we show two ways of doing it.

```
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width, color=Species)) +
  geom_point()
```



```
ggsave('PetalLength_Width.pdf')
```

```
## Saving 4 x 3 in image
```

```
# or

pdf("PetalLength_Width2.pdf", width = 6, height = 6) # open a graphics device
iris %>%
  ggplot(aes(x=Petal.Length, y=Petal.Width, color=Species)) +
  geom_point() # generate plot
dev.off() # close the device
```

```
## pdf
##   2
```

Exercises

24. Take a look at the Help section for `ggsave()` to learn other extensions you can use, as well as how to set the width and height.

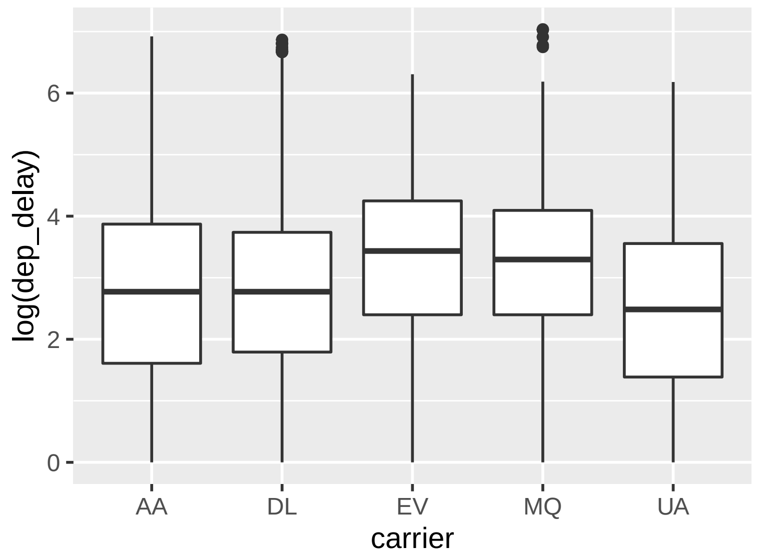
25. What happens if you generate two different plots between `pdf()` and `dev.off()`? Try it.

Final tips and tricks

It is also good to be aware of some other things that can be done with ggplot2, so keep reading.

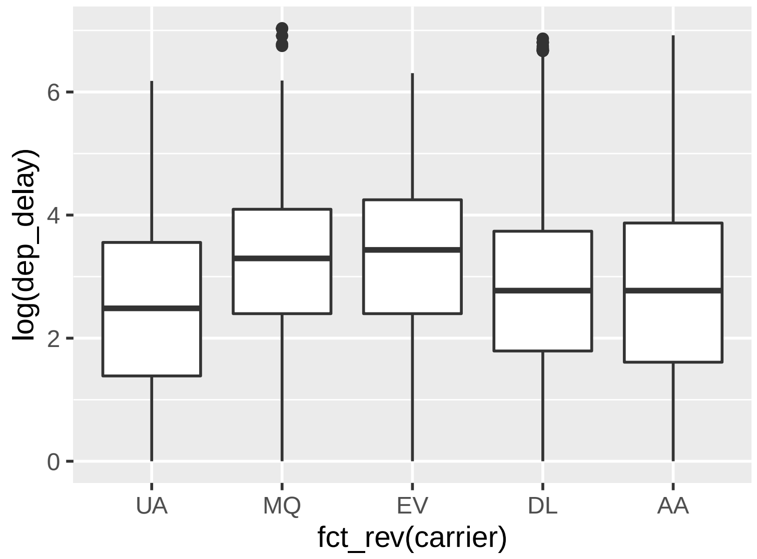
You can modify variables directly in the code without needing to do a `mutate()` beforehand. See how we log transform the variable 'dep_delay' here mid-plotting.

```
flights %>%
  filter(carrier %in% c('UA', 'DL', 'EV', 'AA', 'MQ'),
         dep_delay > 0) %>%
  ggplot(aes(x=carrier, y=log(dep_delay))) +
  geom_boxplot()
```

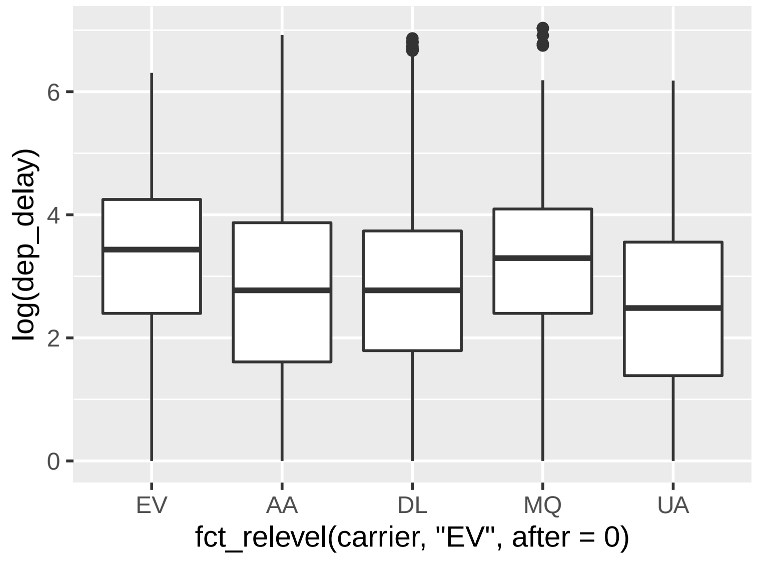


You can reorder factors in a plot using the forcats package that comes in tidyverse. Here we show ways of reversing (`fct_rev()`), reordering manually (`fct_relevel()`), and reordering automatically according to some value ().

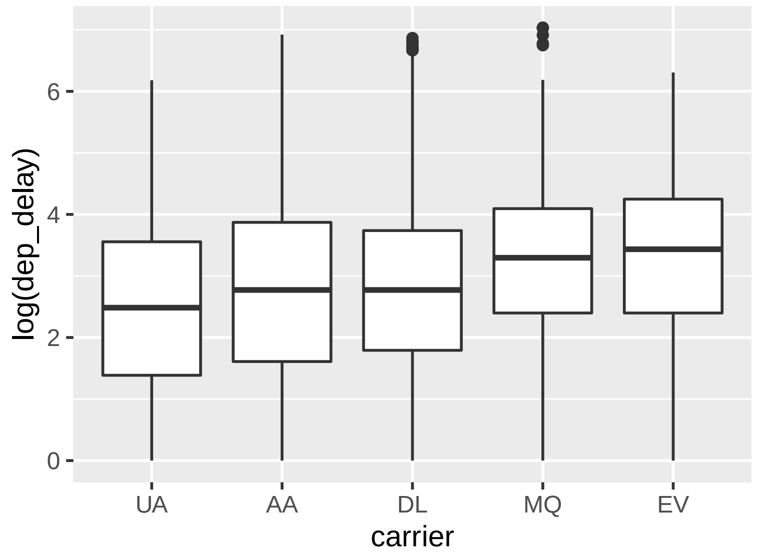
```
flights %>%
  filter(carrier %in% c('UA', 'DL', 'EV', 'AA', 'MQ'),
         dep_delay > 0) %>%
  # reverse
  ggplot(aes(x=fct_rev(carrier), y=log(dep_delay))) +
  geom_boxplot()
```



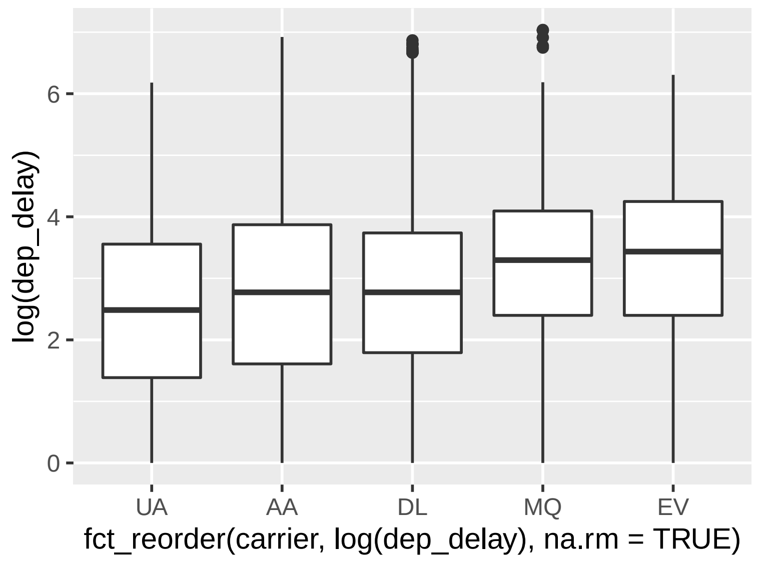
```
flights %>%
  filter(carrier %in% c('UA', 'DL', 'EV', 'AA', 'MQ'),
         dep_delay > 0) %>%
  # bring EV to the beginning
  ggplot(aes(x=fct_relevel(carrier, 'EV', after=0), y=log(dep_delay))) +
  geom_boxplot()
```



```
flights %>%
  filter(carrier %in% c('UA', 'DL', 'EV', 'AA', 'MQ'),
         dep_delay > 0) %>%
  # arrange in a mutate
  mutate(carrier = fct_reorder(carrier, log(dep_delay), na.rm=TRUE)) %>%
  ggplot(aes(x=carrier, y=log(dep_delay))) +
  geom_boxplot()
```



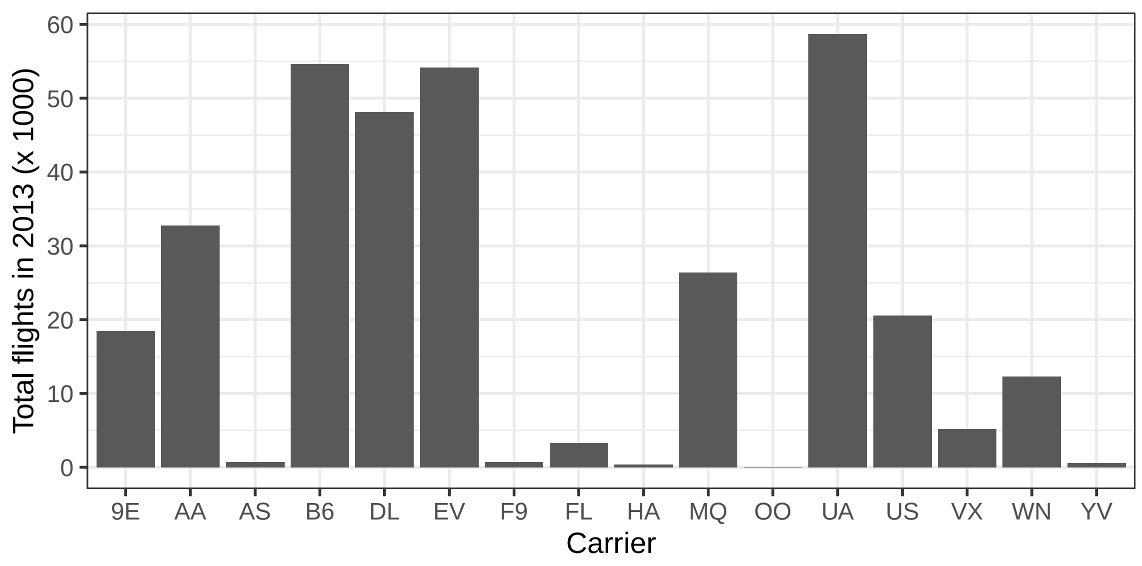
```
flights %>%
  filter(carrier %in% c('UA', 'DL', 'EV', 'AA', 'MQ'),
         dep_delay > 0) %>%
  # arrange in ggplot
  ggplot(aes(x=fct_reorder(carrier, log(dep_delay), na.rm=TRUE), y=log(dep_delay))) +
  geom_boxplot()
```



Exercises

26. Arrange the very first plot you had in this section from higher to lower values, which is easily done with a different functions from forcats. Look at the factors cheatsheet or write 'forcats reorder geom_bar count' in a search engine to solve this. Search engines and Stack Overflow are the programmer's best friends and learning how to structure your queries to find the answers for your questions will give you a huge freedom.

```
flights %>%
  ggplot(aes(x=carrier)) +
  geom_bar() +
  labs(x='Carrier', y='Total flights in 2013 (x 1000)') +
  theme_bw() +
  scale_y_continuous(breaks=seq(0,600000,10000),
                     labels=seq(0,600000,10000)/1000)
```



Resources

See? Plotting with ggplot is fun! You already know the basics and can generate very nice plots, but of course there is much more you can learn. Here are some other resources to explore:
* Data visualization cheatsheet
* R for Data Science
* ggplot2: Elegant Graphics for Data Analysis
* R Graphics cookbook