Iterative Plotting for Easy and Aesthetic Figures

Author: Cole Brookson **Date:** 20 July 2022

When making a plot, typically the best way to go about it is to iteratively make a version of the plot, observe the outcome, and then make changes in an iterative fashion. This prevents us from making mistakes in a long set of code that we then can't diagnose the problem with, and further, helps us catch mistakes (typos, colour problems, etc) as we only have to take in a few new components each time, and not a whole pile.

For our example, let's use the example from the (LINK) Intro to `ggplot2` section, looking at salamanders and trout:

```
library(tidyverse)
library(lterdatasampler)

df <- lterdatasampler::and_vertebrates

names(df)
```

```
##  [1] "year"         "sitecode"     "section"      "reach"        "pass"
##  [6] "unitnum"      "unittype"     "vert_index"   "pitnumber"    "species"
## [11] "length_1_mm"  "length_2_mm"  "weight_g"     "clip"         "sampledate"
## [16] "notes"
```

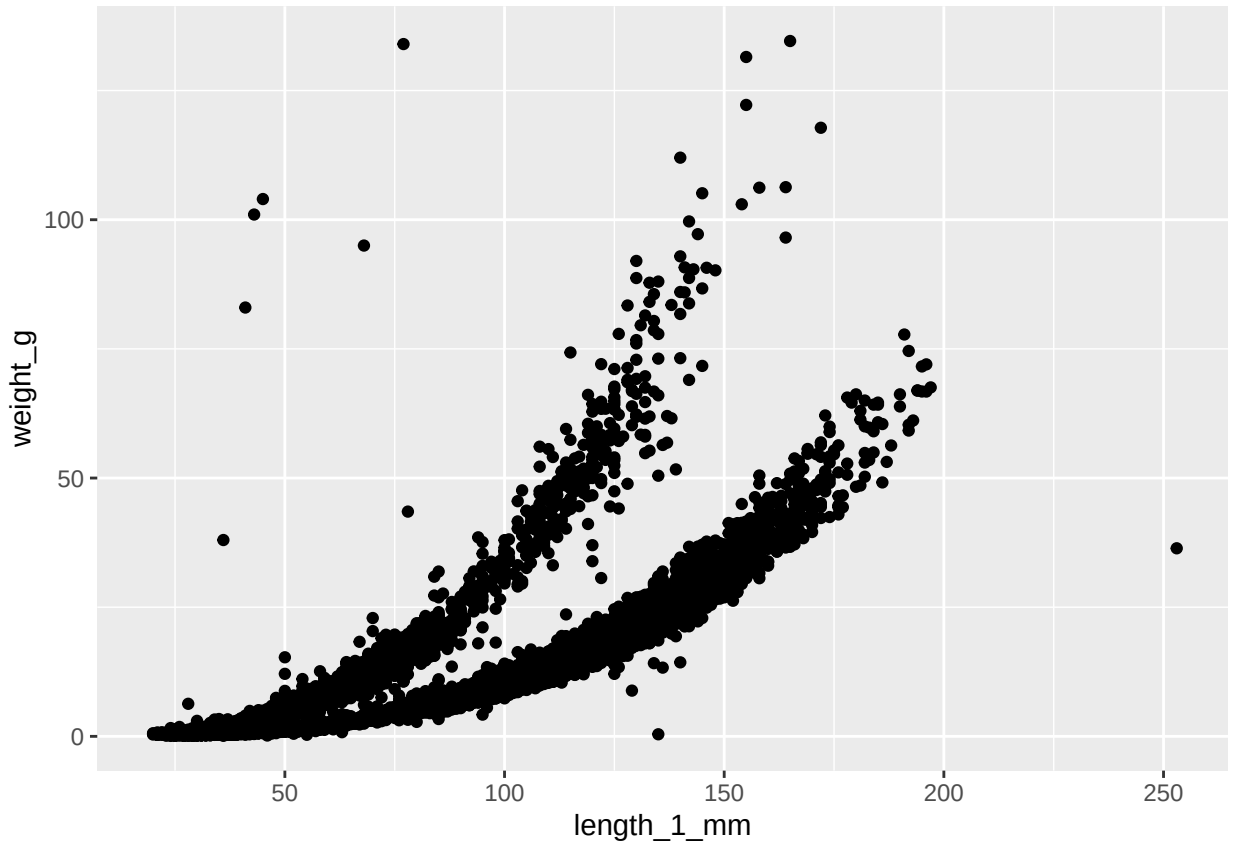
Building a Basic Plot

Let's again plot a scatter plot of `length_1_mm` vs. `weight_g`. We'll start out with the absolute minimum number of arguments:

1. The call to `ggplot()`
2. The aesthetics (i.e. what variables we want on the x- and y-axis)
3. The `geom_` function denoting what *type* of plot we're going for

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g))
```

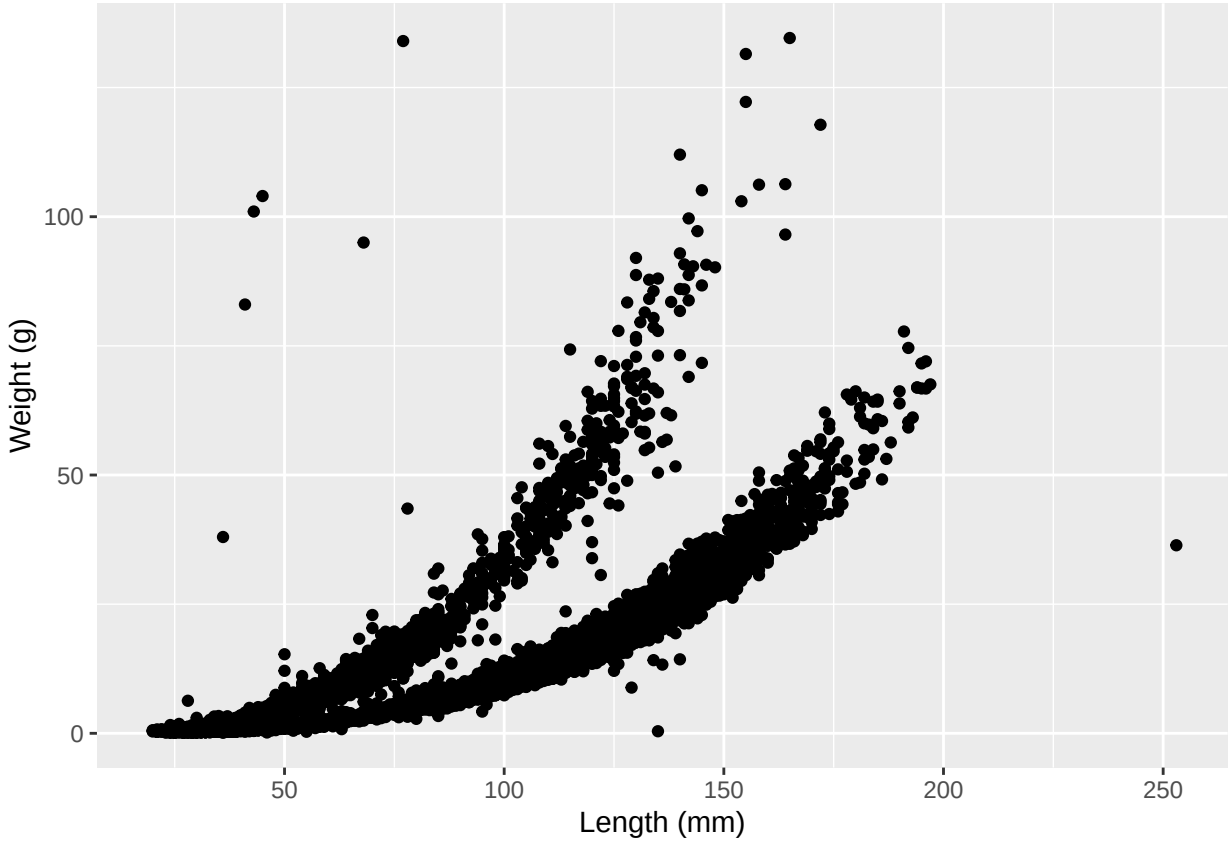
```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



Ok so we see our plot! Perfect. So the first thing to do, before we make any stylistic additions, is to deal with the components of the plot that are *must-haves* to have a legible, complete plot. First step is the axis labels. So to iteratively work up our plot, we'll add that one component on, by using the new function `labs()`:

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g)) +
  labs(x = "Length (mm)", y = "Weight (g)")
```

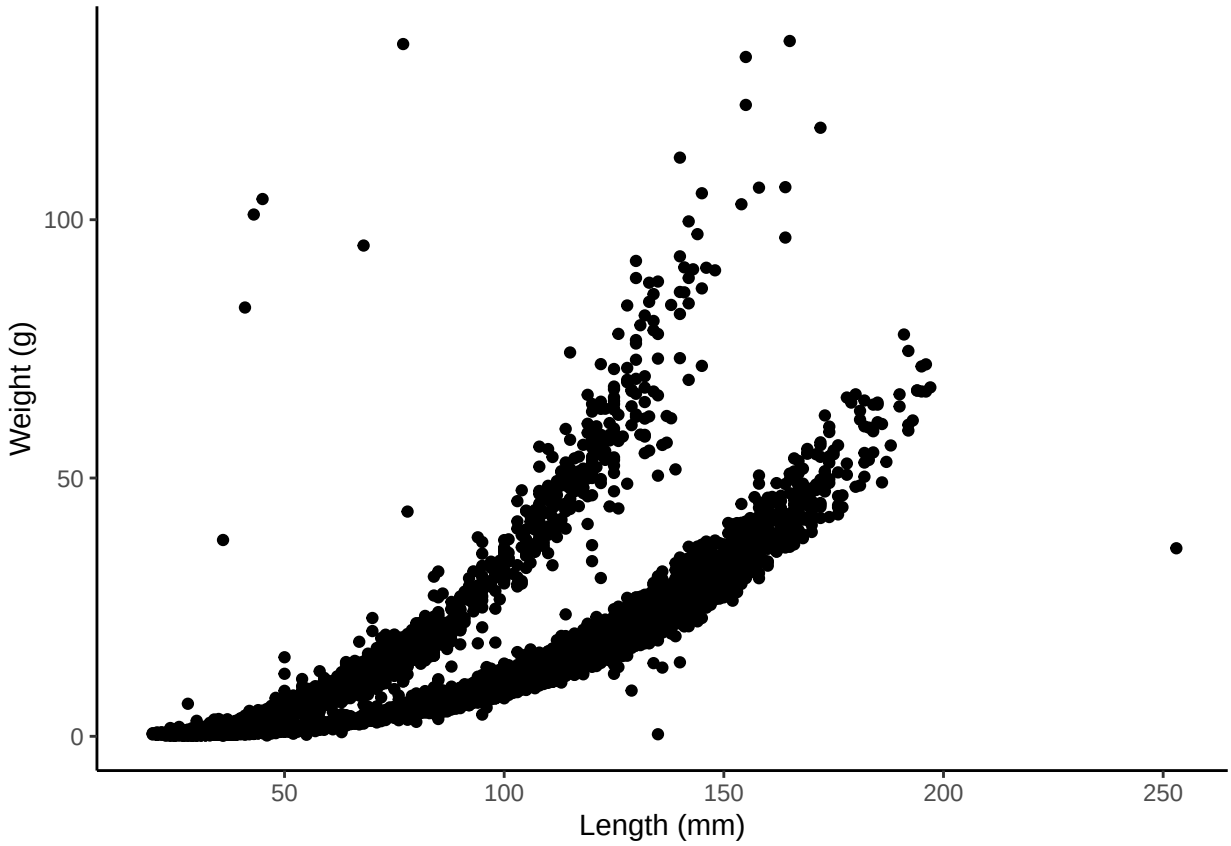
```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



OK good. Now, this is more aesthetic than anything, but we might want to change the appearance of this plot away from this ugly grey background and grid lines. This is a more personal decision regarding what exactly you want your plot to look like. However, there are lots of `themes` available in the `ggplot2` package to choose from. For example, we could use the `theme_classic()` like so:

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g)) +
  labs(x = "Length (mm)", y = "Weight (g)") +
  theme_classic()
```

```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



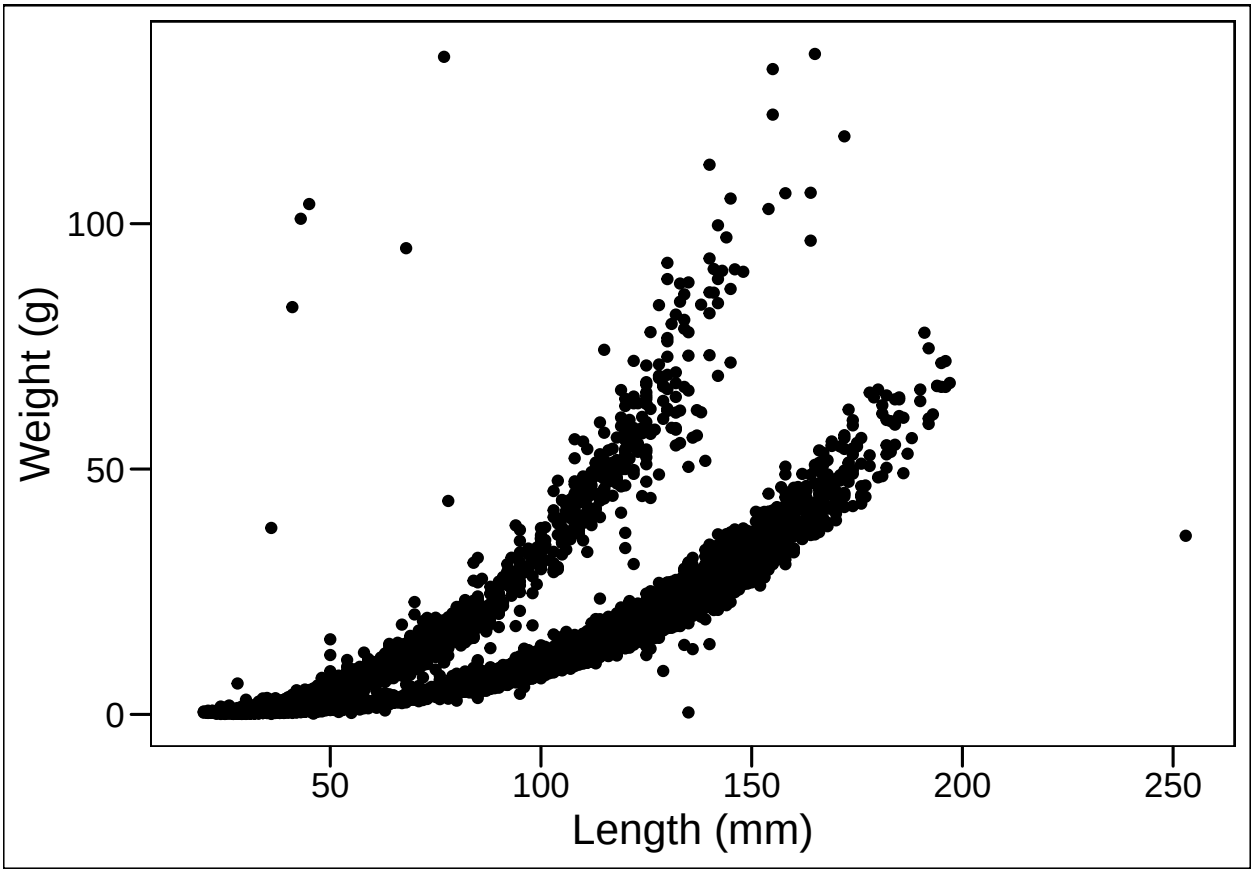
But I *personally* prefer a different theme that comes in the `ggthemes` package, which we can install now:

```
#install.packages("ggthemes")
library(ggthemes)
```

Now, let's use the `theme_base()` option, which will give us a very classic look and feel to the plot:

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g)) +
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base()
```

```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



Perfect! You'll also notice the text sizes are now larger which is preferable for reading on a screen or in a report.

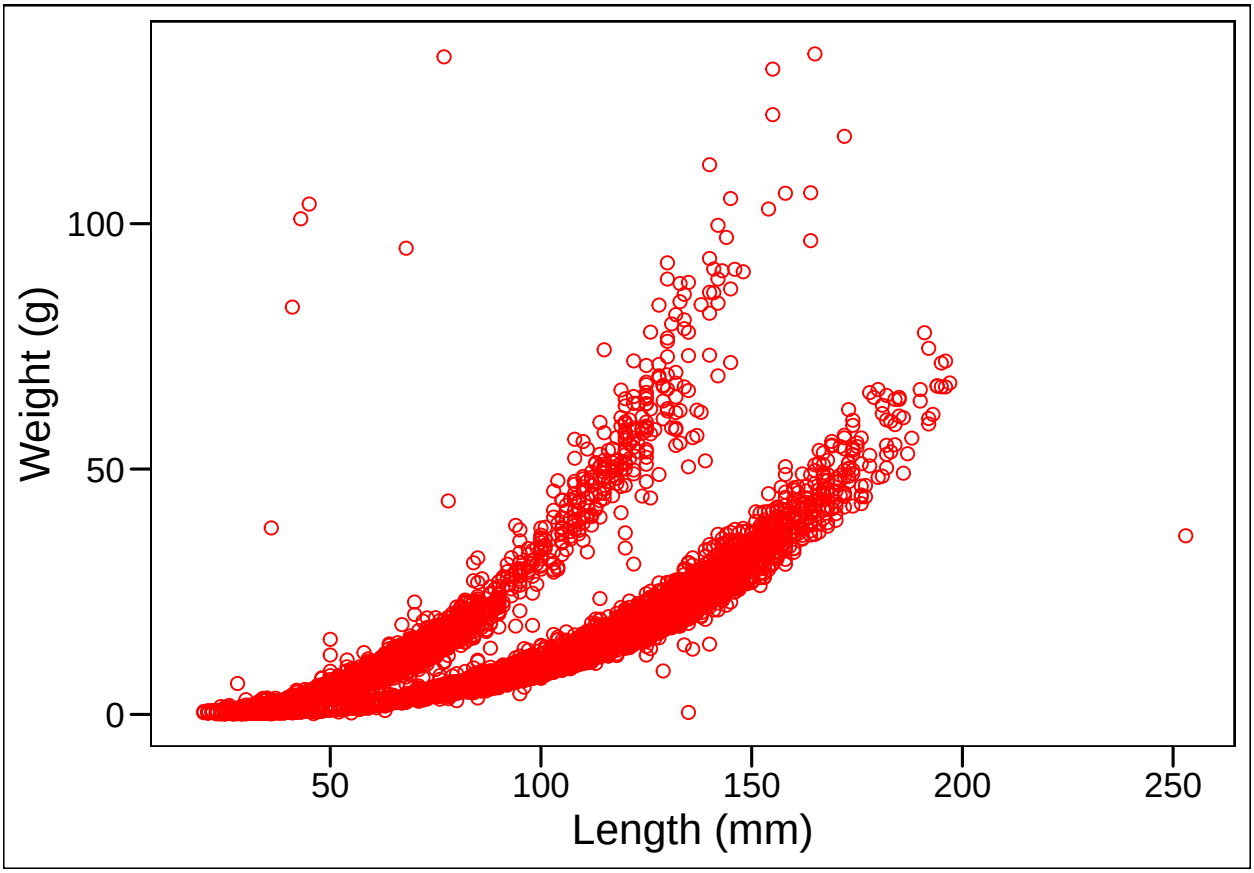
Adding Colour & Shape

Let's now add a colour to our points and change the shape we're using as well.

Note: Arguments like this go in the `aes()` section, but there is a difference between simply assigning a colour and using colour to group points. To start, we'll just make the colour *red*, change the shape to my personal favourite, and up the size just a smidge:

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g),
             # here's where we'll add our colour etc
             colour = "red", shape = 21, size = 2) +
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base()
```

```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



So this is interesting! We see that somehow we've made our points hollow on the inside. Well, that has to do with the shape we picked. There are lots of options for shapes we *could* use, and we can see all our options in this handy graphic here:

So notice we specified point # 21. This point has a dark outer circle and a fill colour in the middle. **When using a point option with an outter boundary and an inner fill, the argument `colour` refers to the outter boundary line, and the argument `fill` refers to the colour filling inside**. Let's see how this works by changing `colour = "black"` and `fill = "red"`:

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g),
             # here's where we'll add our colour etc
             colour = "black", fill = "red", shape = 21, size = 2) +
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base()
```

```
## Warning: Removed 13279 rows containing missing values (geom_point).
```

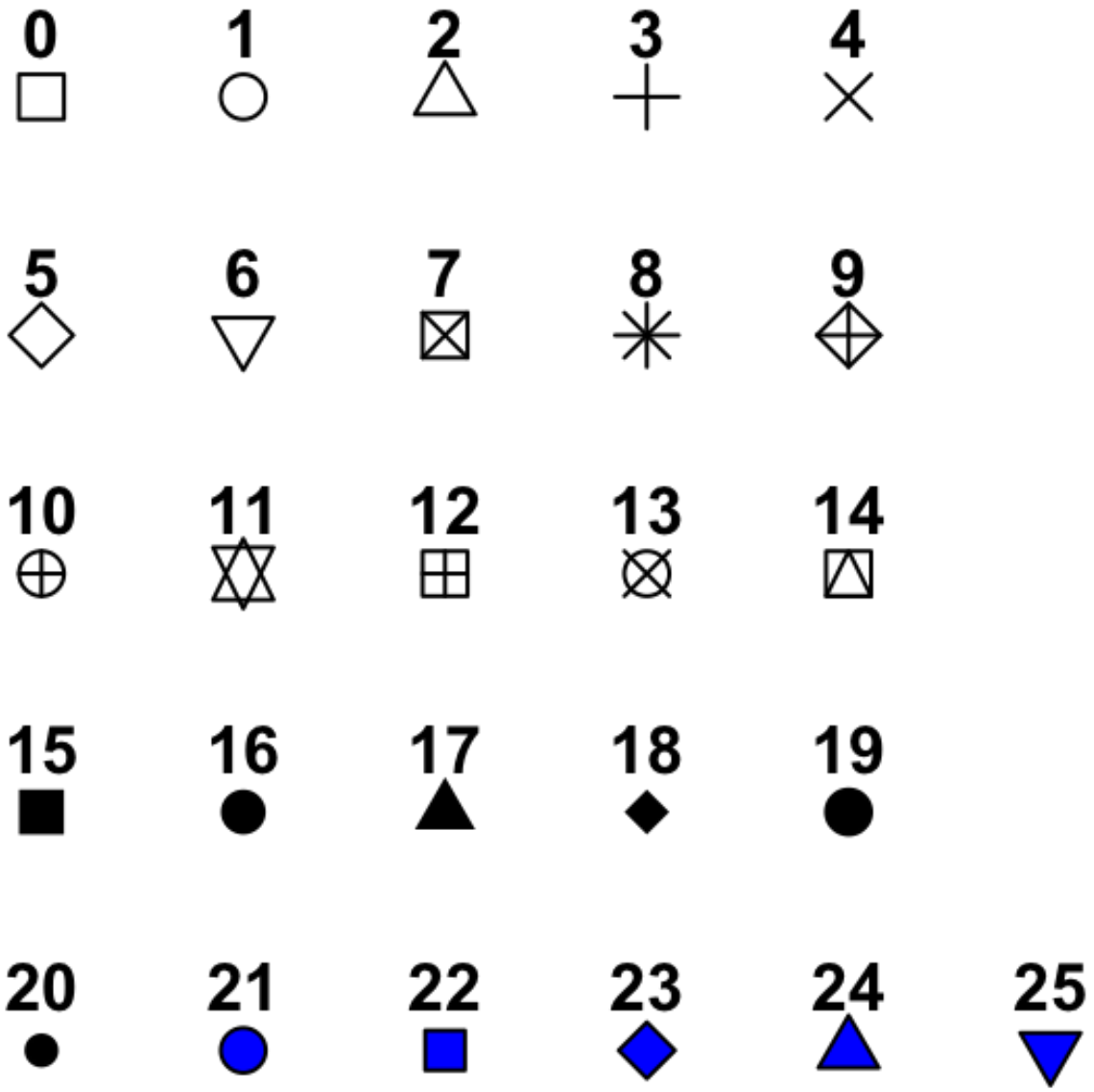
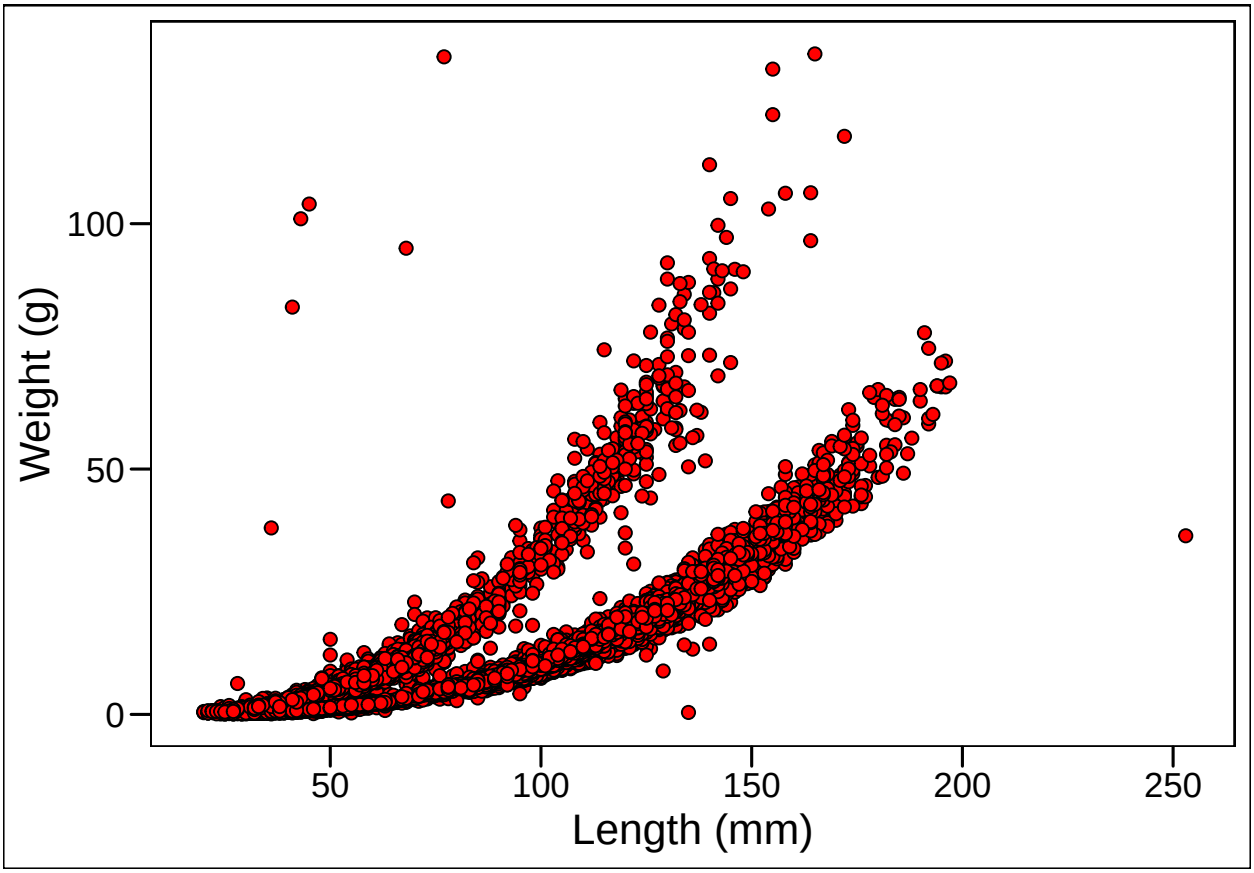


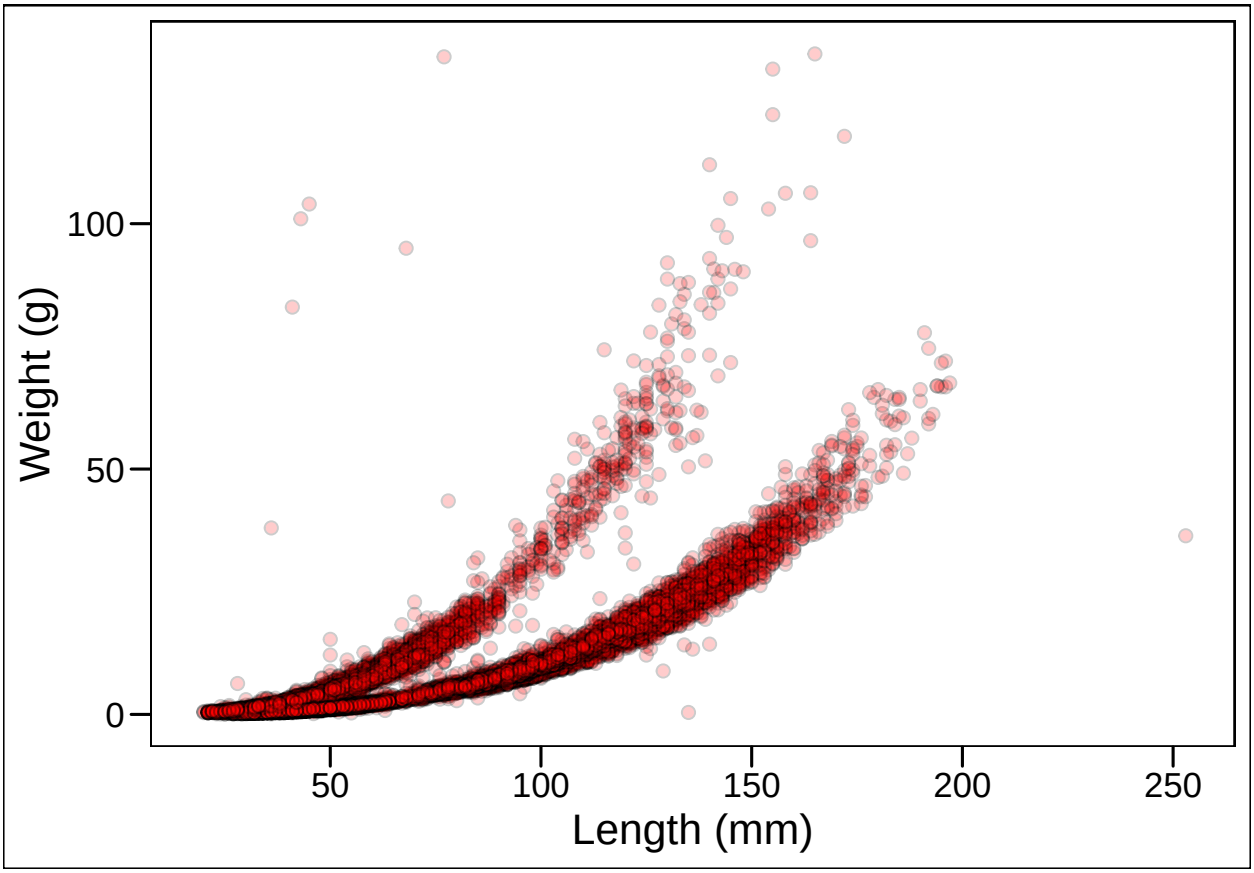
Figure 1: shapes



Okay, so this looks like maybe more along the lines of what we were expecting! One more thing though is that some spots on the plot have a high density of points. To better identify those high density areas, let's make our points slightly transparent, with the `alpha` commmand:

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g),
             # here's where we'll add our colour etc
             colour = "black", fill = "red", shape = 21, alpha = 0.2, size = 2)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base()
```

```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



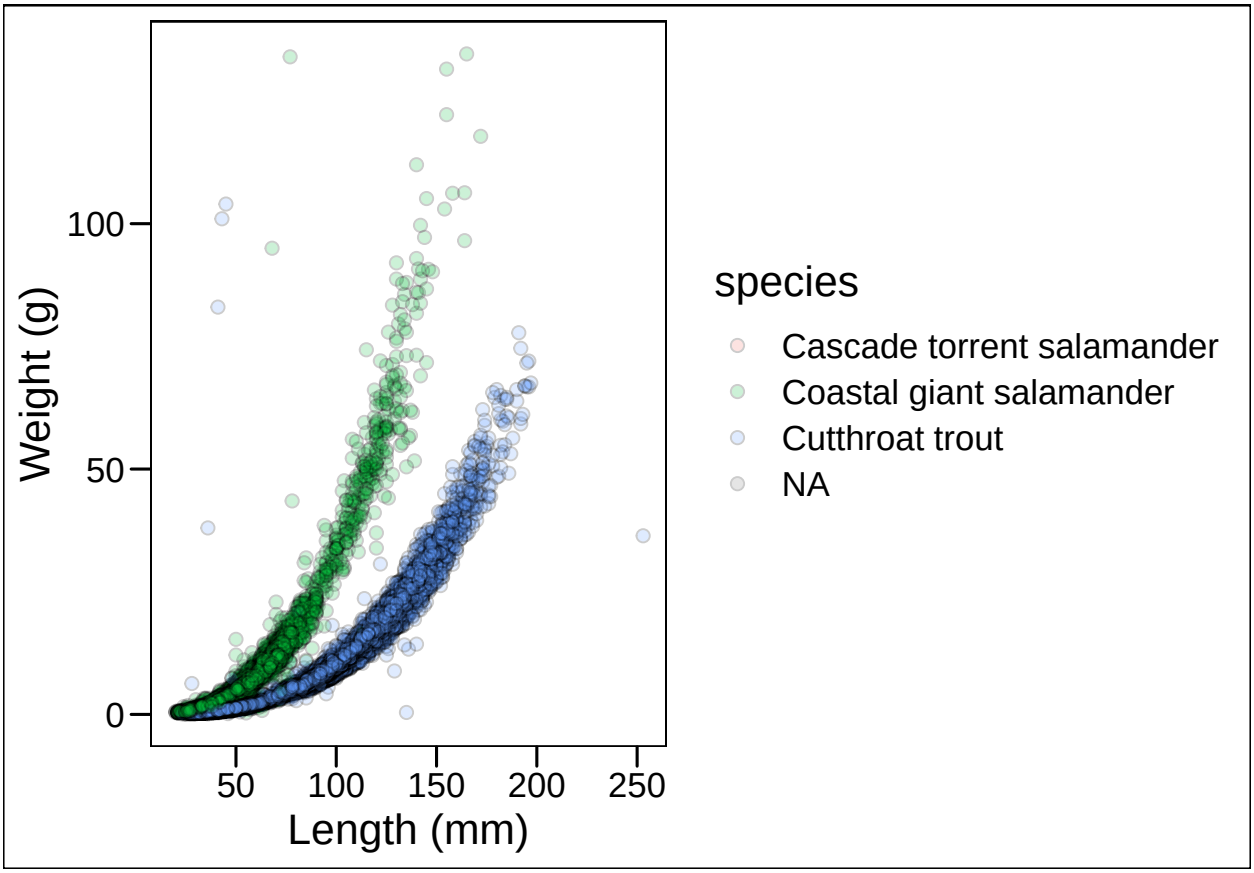
And now we can see where there really are many points.

Variables by Group

One of the most useful things to be able to do is group our points by either shape, fill, colour, or even size to show some difference betwen them. For example, in these data, we can see two very clearly different trends.

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g, fill = species),
             # here's where we'll add our colour etc
             colour = "black", shape = 21, alpha = 0.2, size = 2)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base()
```

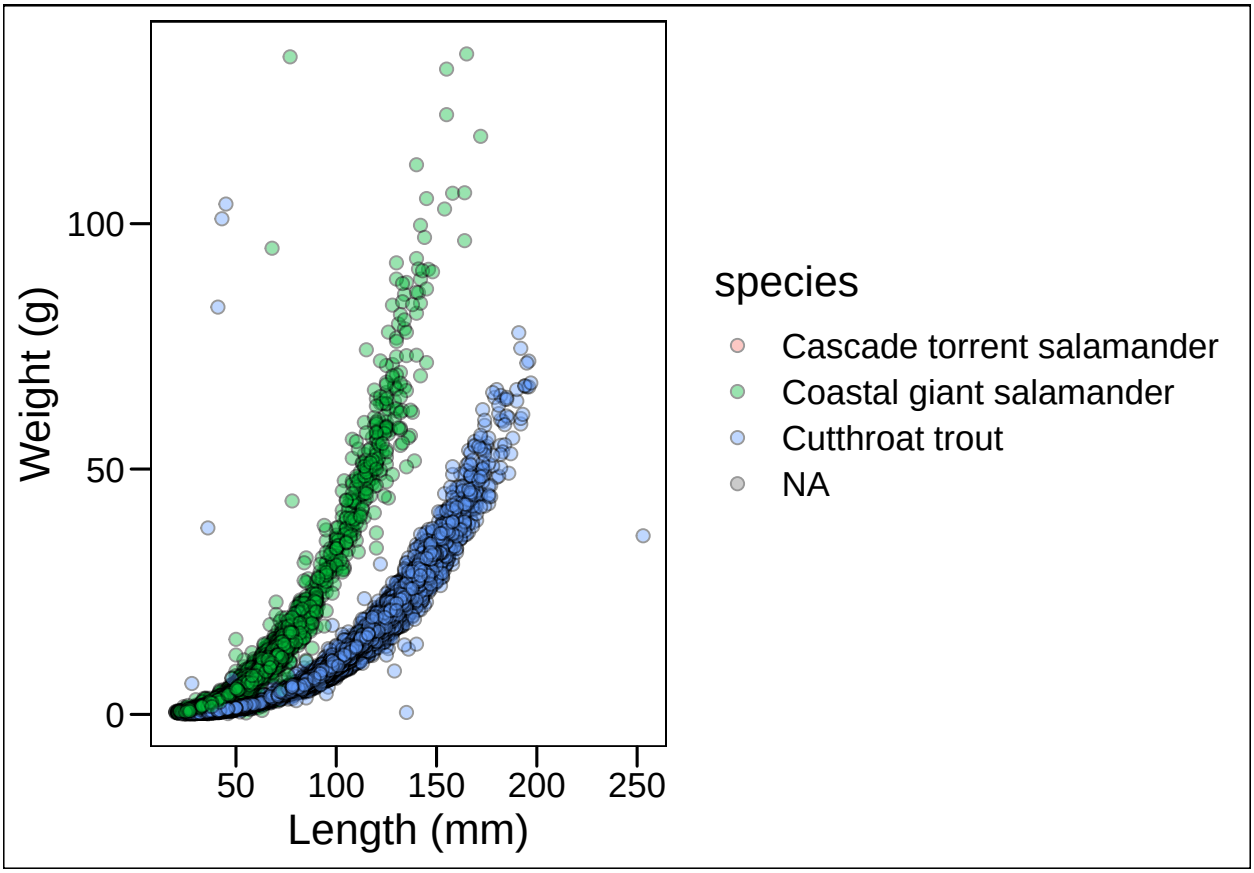
```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



So we can see our two main point colours plotted out nicely here. Although, there should be four colours and we can't really see them. Let's change our `alpha` to fix that:

```
ggplot() +
  geom_point(data = df, aes(x = length_1_mm, y = weight_g, fill = species),
             # here's where we'll add our colour etc
             colour = "black", shape = 21, alpha = 0.4, size = 2)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base()
```

```
## Warning: Removed 13279 rows containing missing values (geom_point).
```



That's better. Looking at this plot, this brings up a few common tasks we might want to do from here. First of all, it's clear from this plot that the vast majority of these points are only from two species. We can see how the samples in our dataframe fall out with the simple `table()` function:

```
table(df$species)
```

```
##
## Cascade torrent salamander   Coastal giant salamander
##                         15                      11758
##            Cutthroat trout
##                      20433
```

So there are only 15 of the Cascade salamanders, and we know from our plot, at least a few NA's. Now the following decisions depend HIGHLY on what message one is trying to communicate with the plot, but we could imagine a scenario wherein perhaps it's actually important to have the Cascade salamander points be visible. But, it's clear from our plot, we may first want to get rid of NA's at least fo this plot. So let's do that via a `filter()` (LINK TO FILTERING AND SUBSETTING)

```
df_filtered <- df %>%
  dplyr::filter(!is.na(species))
head(df_filtered)
```

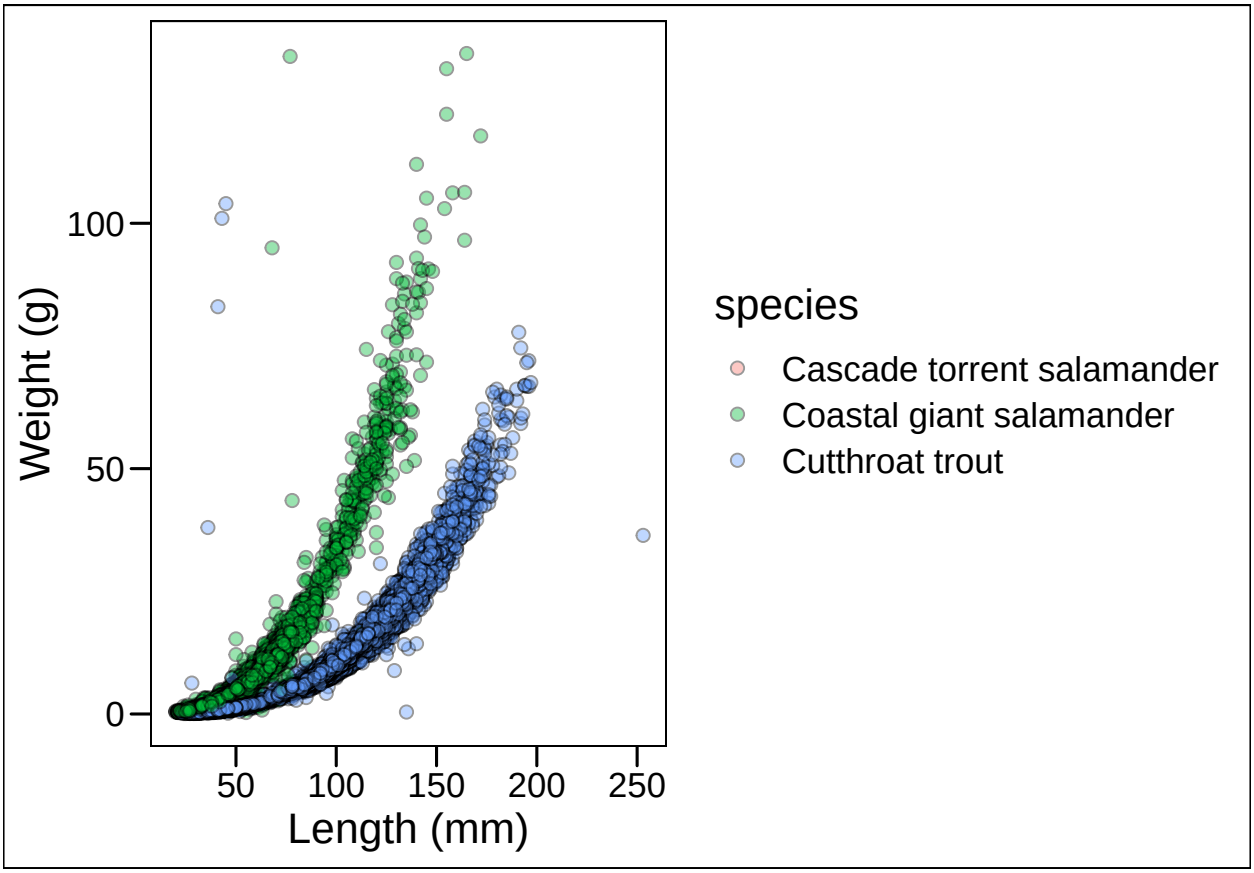
```
## # A tibble: 6 x 16
##    year sitecode section reach  pass unitnum unittype vert_index pitnu~1 species
##   <dbl> <chr>    <chr>   <chr> <dbl>   <dbl> <chr>         <dbl>   <dbl> <chr>
```

```
## 1  1987 MACKCC-L CC      L         1       1 R                 1      NA Cutthr~
## 2  1987 MACKCC-L CC      L         1       1 R                 2      NA Cutthr~
## 3  1987 MACKCC-L CC      L         1       1 R                 3      NA Cutthr~
## 4  1987 MACKCC-L CC      L         1       1 R                 4      NA Cutthr~
## 5  1987 MACKCC-L CC      L         1       1 R                 5      NA Cutthr~
## 6  1987 MACKCC-L CC      L         1       1 R                 6      NA Cutthr~
## # ... with 6 more variables: length_1_mm <dbl>, length_2_mm <dbl>,
## #   weight_g <dbl>, clip <chr>, sampledate <date>, notes <chr>, and abbreviated
## #   variable name 1: pitnumber
## # i Use `colnames()` to see all variable names
```

Okay, let's go ahead and make our plot again:

```
ggplot() +
  geom_point(data = df_filtered, # using the new dataframe
             aes(x = length_1_mm, y = weight_g, fill = species),
             # here's where we'll add our colour etc
             colour = "black", shape = 21, alpha = 0.4, size = 2)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base()
```

```
## Warning: Removed 13276 rows containing missing values (geom_point).
```



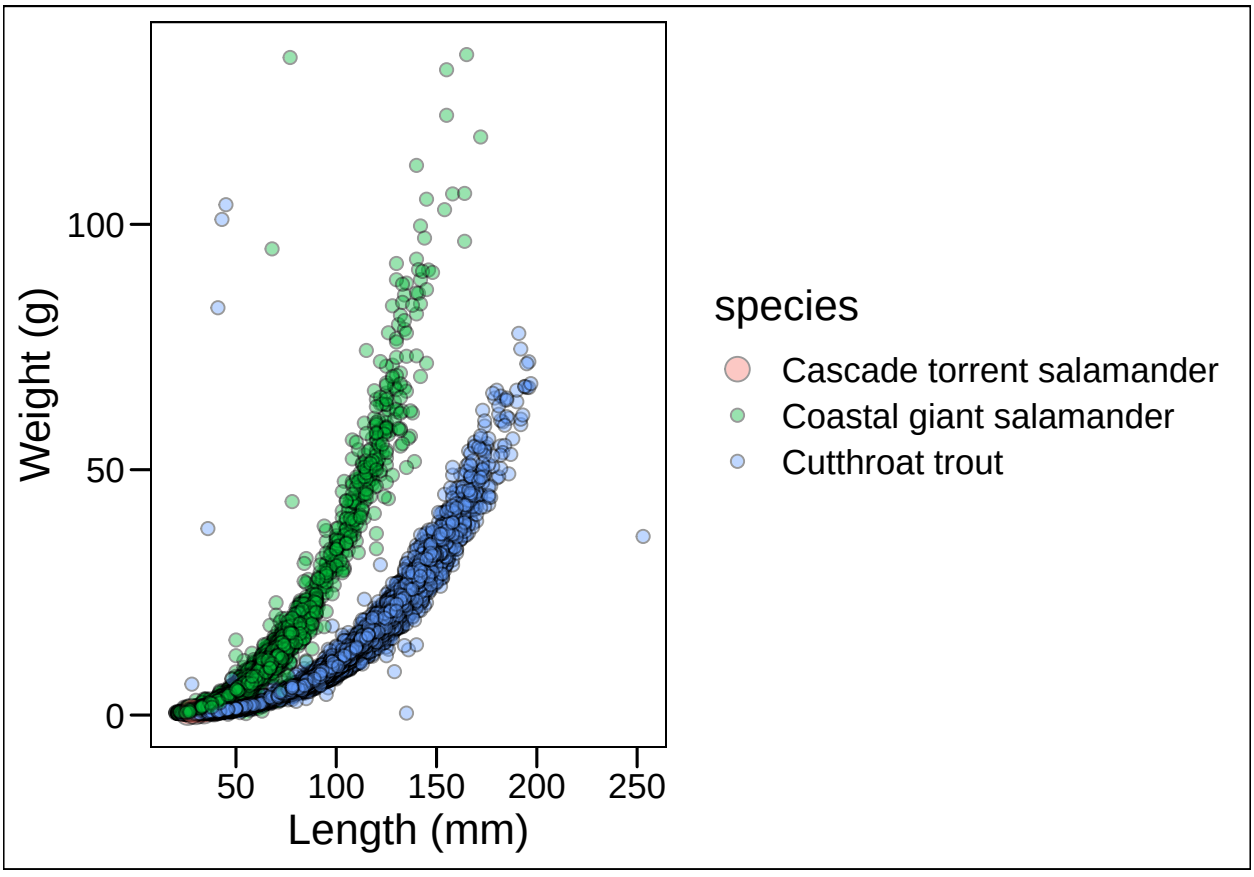
This looks better. Now perhaps one way we could make the Cascade salamander points stand out is by making them significantly larger in size. We can do that my scaling our size manually according to the levels

of our grouping variable. **Note, that scaling, whether manually or otherwise, can be done for all grouping aesthetics such as fill, colour, alpha, etc. Further documentation on this topic can be found on the ggplot2 website**

Since we see that the Cascade Torrent Salmander is first in our list of the legend, that means we will need to make the larger size first up in our `values` argument:

```
ggplot() +
  geom_point(data = df_filtered, # using the new dataframe
             aes(x = length_1_mm, y = weight_g, fill = species, size = species),
             # here's where we'll add our colour etc
             colour = "black", shape = 21, alpha = 0.4)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base() +
  scale_size_manual(values = c(4, 2, 2))
```

```
## Warning: Removed 13276 rows containing missing values (geom_point).
```



Hmm. We can vaguely see in the bottom corner the points we want, but they're hard to see. Why is this? Well, the problem here is that `ggplot()` will automatically plot data in the order they are passed. we can see by a quick `head()` call ...

```
head(df_filtered)
```

```
## # A tibble: 6 x 16
```

```
##    year sitecode section reach  pass unitnum unittype vert_index pitnu~1 species
##   <dbl> <chr>    <chr>   <chr> <dbl>   <dbl> <chr>         <dbl>   <dbl> <chr>
## 1  1987 MACKCC-L CC      L         1       1 R                 1      NA Cutthr~
## 2  1987 MACKCC-L CC      L         1       1 R                 2      NA Cutthr~
## 3  1987 MACKCC-L CC      L         1       1 R                 3      NA Cutthr~
## 4  1987 MACKCC-L CC      L         1       1 R                 4      NA Cutthr~
## 5  1987 MACKCC-L CC      L         1       1 R                 5      NA Cutthr~
## 6  1987 MACKCC-L CC      L         1       1 R                 6      NA Cutthr~
## # ... with 6 more variables: length_1_mm <dbl>, length_2_mm <dbl>,
## #   weight_g <dbl>, clip <chr>, sampledate <date>, notes <chr>, and abbreviated
## #   variable name 1: pitnumber
## # i Use `colnames()` to see all variable names
```

That the first species in the list is Cutthroat trout, and with a quick `tail()` call (`tail()` returns the last six rows instead of the first six) . . .

```
tail(df_filtered)
```

```
## # A tibble: 6 x 16
##    year sitecode section reach  pass unitnum unittype vert_index pitnu~1 species
##   <dbl> <chr>    <chr>   <chr> <dbl>   <dbl> <chr>         <dbl>   <dbl> <chr>
## 1  2019 MACKOG-U OG      U         2      16 C                21      NA Coasta~
## 2  2019 MACKOG-U OG      U         2      16 C                22      NA Coasta~
## 3  2019 MACKOG-U OG      U         2      16 C                23 1043503 Coasta~
## 4  2019 MACKOG-U OG      U         2      16 C                24 1043547 Coasta~
## 5  2019 MACKOG-U OG      U         2      16 C                25 1043583 Coasta~
## 6  2019 MACKOG-U OG      U         2      16 C                26 1043500 Coasta~
## # ... with 6 more variables: length_1_mm <dbl>, length_2_mm <dbl>,
## #   weight_g <dbl>, clip <chr>, sampledate <date>, notes <chr>, and abbreviated
## #   variable name 1: pitnumber
## # i Use `colnames()` to see all variable names
```

. . . we can see that the Coastal salamander is the last. That means that the Coastal salamander will be plotted after (i.e "on top of") our focal larger points. Luckily this is an easy fix by simply rearranging the dataframe such that the focal species is the *last* one to be plotted. This can be accomplished with `dplyr::arrange()`. We'll pass first the dataframe we're working with, then the column we want to sort by. To check that this will work, let's print the calll to `tail()` from the `arrange()` version of our dataframe.

```
tail(dplyr::arrange(df_filtered, species))
```

```
## # A tibble: 6 x 16
##    year sitecode section reach  pass unitnum unittype vert_index pitnu~1 species
##   <dbl> <chr>    <chr>   <chr> <dbl>   <dbl> <chr>         <dbl>   <dbl> <chr>
## 1  2019 MACKOG-U OG      U         2      16 C                 1      NA Cutthr~
## 2  2019 MACKOG-U OG      U         2      16 C                 2      NA Cutthr~
## 3  2019 MACKOG-U OG      U         2      16 C                 3      NA Cutthr~
## 4  2019 MACKOG-U OG      U         2      16 C                 4 1043522 Cutthr~
## 5  2019 MACKOG-U OG      U         2      16 C                 5 1043513 Cutthr~
## 6  2019 MACKOG-U OG      U         2      16 C                 6 1043521 Cutthr~
## # ... with 6 more variables: length_1_mm <dbl>, length_2_mm <dbl>,
## #   weight_g <dbl>, clip <chr>, sampledate <date>, notes <chr>, and abbreviated
## #   variable name 1: pitnumber
## # i Use `colnames()` to see all variable names
```

That doesn't work! That's because `arrange()` will automatically default to sortin alphabetically. Luckily we can convert this column into factor to change the order in a custom way. Lets do so like this (and again check with `tail()`):

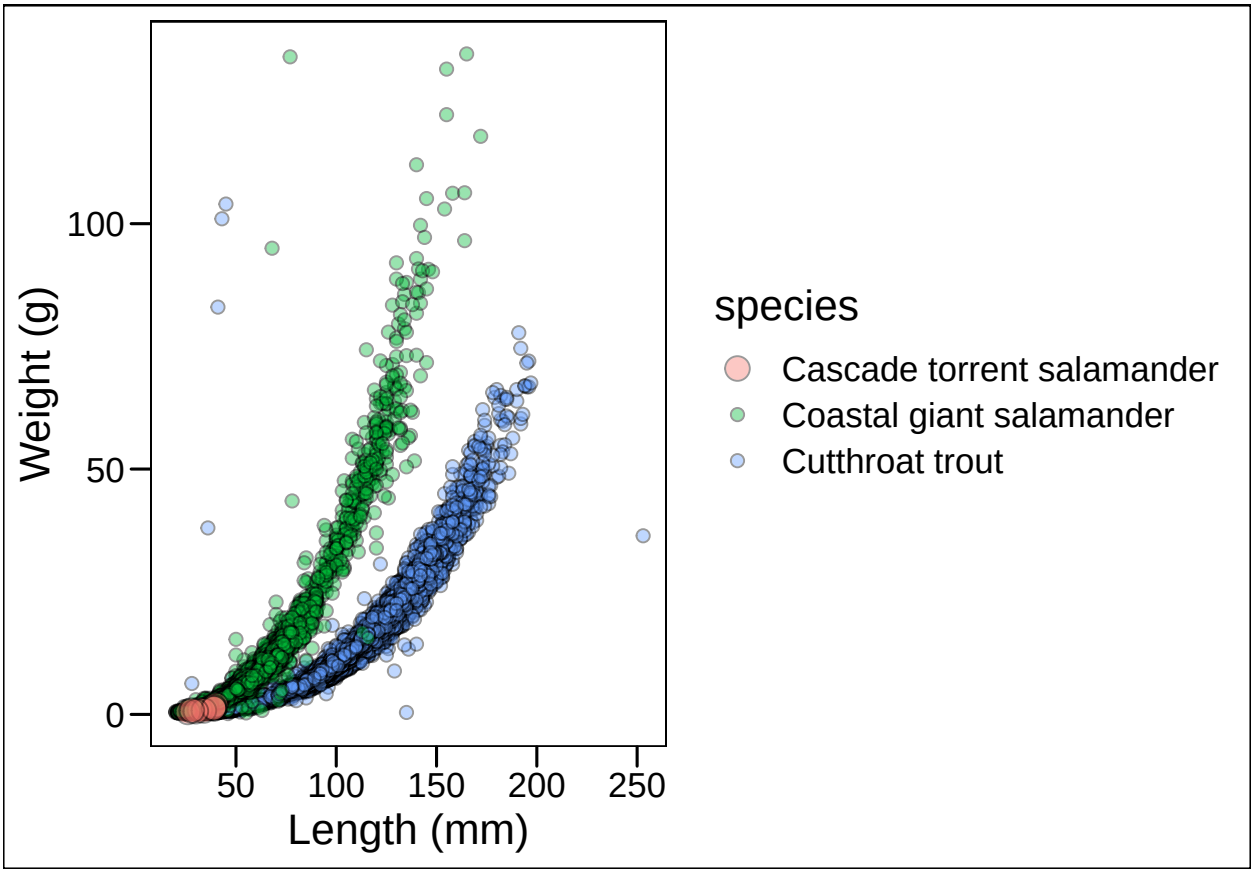
```
# wrap our pipe arrange() with tail
tail(
  df_filtered %>%
  dplyr::arrange(factor(species,
                        levels = c("Cutthroat trout",
                                   "Coastal giant salamander",
                                   "Cascade torrent salamander")))
)
```

```
## # A tibble: 6 x 16
##    year sitecode section reach  pass unitnum unittype vert_index pitnu~1 species
##   <dbl> <chr>    <chr>   <chr> <dbl>   <dbl> <chr>         <dbl>   <dbl> <chr>
## 1  2012 MACKOG-U OG      U         1      16 SC                2      NA Cascad~
## 2  2013 MACKCC-L CC      L         2       2 SC               12      NA Cascad~
## 3  2013 MACKOG-U OG      U         1      14 P                 1      NA Cascad~
## 4  2017 MACKCC-L CC      L         1       2 SC               29      NA Cascad~
## 5  2017 MACKOG-M OG      M         1       8 SC               20      NA Cascad~
## 6  2019 MACKOG-M OG      M         1       8 SC               17      NA Cascad~
## # ... with 6 more variables: length_1_mm <dbl>, length_2_mm <dbl>,
## #   weight_g <dbl>, clip <chr>, sampledate <date>, notes <chr>, and abbreviated
## #   variable name 1: pitnumber
## # i Use `colnames()` to see all variable names
```

And we see our friend the Cascade salamander is at the end. Now we don't necessarily want to reassign our dataframe to take this rearranged form, so we can simply wrap our call to the dataframe `df_filtered` in our `arrange()` function and see if that works:

```
ggplot() +
  geom_point(data = df_filtered %>%
               dplyr::arrange(factor(species,
                        levels = c("Cutthroat trout",
                                   "Coastal giant salamander",
                                   "Cascade torrent salamander"))),
             aes(x = length_1_mm, y = weight_g, fill = species, size = species),
             # here's where we'll add our colour etc
             colour = "black", shape = 21, alpha = 0.4)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base() +
  scale_size_manual(values = c(4, 2, 2))
```

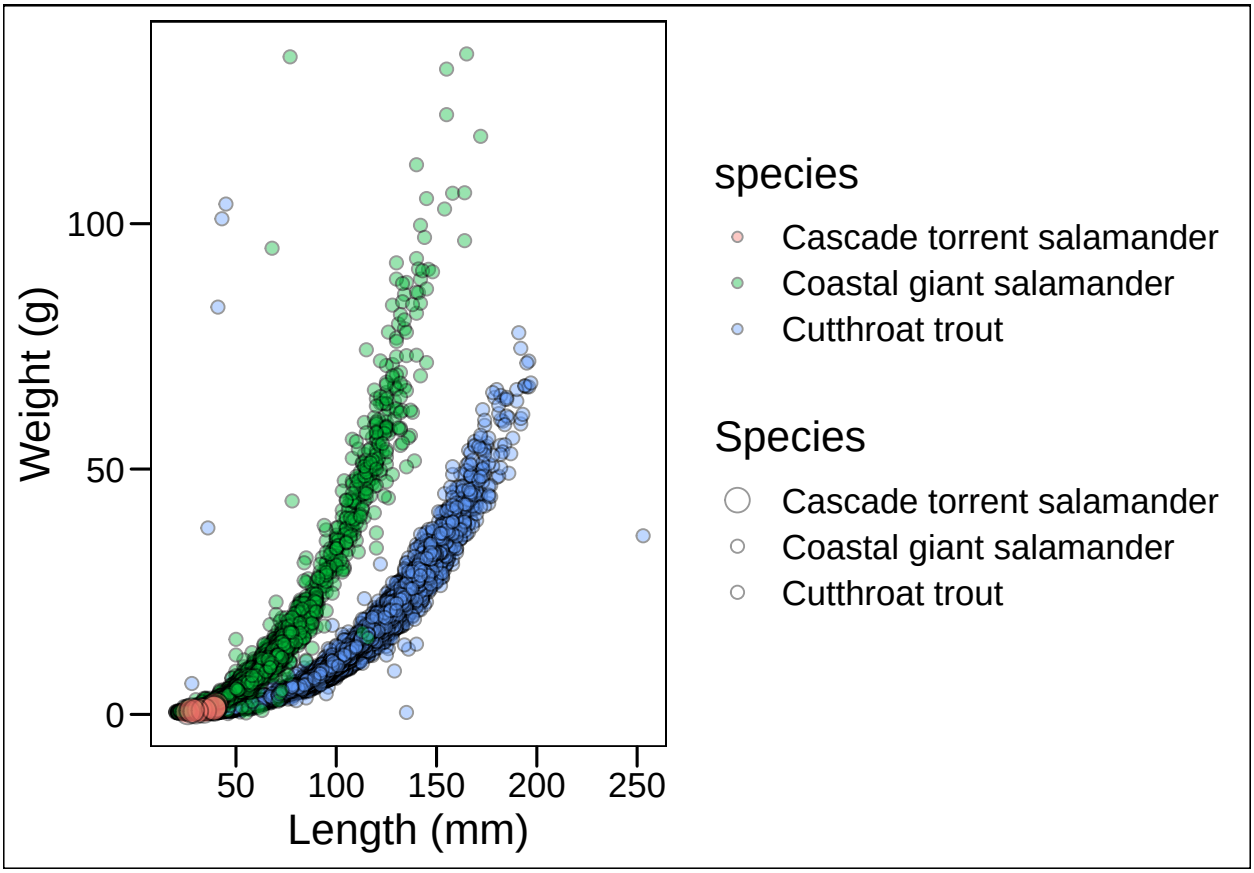
```
## Warning: Removed 13276 rows containing missing values (geom_point).
```



We can now see our Cascade salamander data points! While we could stop here, there are a few other things we can do. We can capitalize the legend easily enough, by simply adding it as the first argument in the `scale_size_manual()` function. We put it here since the `size` parameter is what's being displayed in our plot:

```
ggplot() +
  geom_point(data = df_filtered %>%
               dplyr::arrange(factor(species,
                         levels = c("Cutthroat trout",
                                    "Coastal giant salamander",
                                    "Cascade torrent salamander"))),
             aes(x = length_1_mm, y = weight_g, fill = species, size = species),
             # here's where we'll add our colour etc
             colour = "black", shape = 21, alpha = 0.4)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base() +
  scale_size_manual("Species", values = c(4, 2, 2))
```

```
## Warning: Removed 13276 rows containing missing values (geom_point).
```



Ahh! Now we've run into a new problem. We have two legends! This is annoying. However, we can fix this, but first, let's choose different colours, these are not my favourite. We can do that (in this case) with `scale_fill_manual()`. We use the `fill` because the colour for all the ponts is *black* since it refers to the outer border of the points.

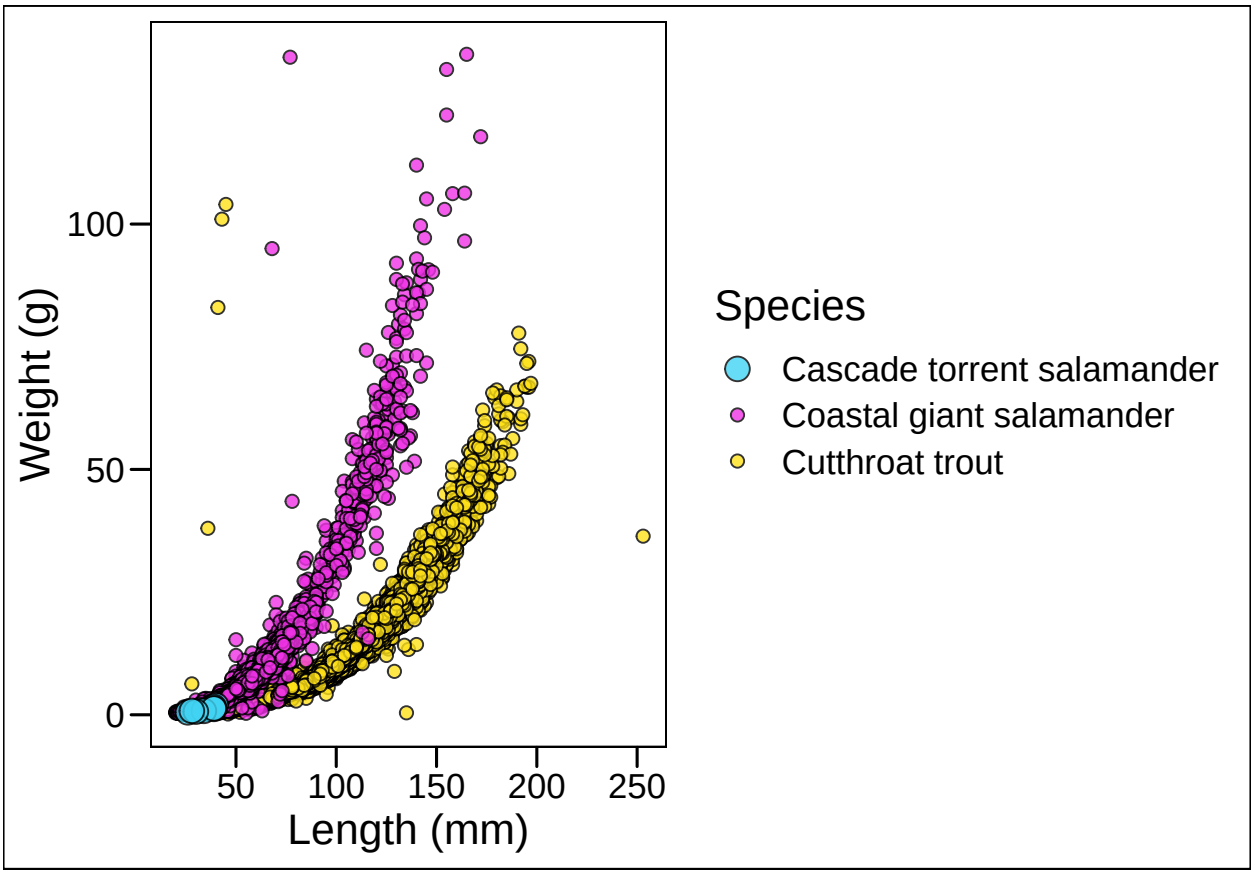
SIDE NOTE: Note that when making figures, when possible, choose colours that if printed in black and white are still differentiable, and that also are distinguishable for colour blind people.

Also, note that often the easiest way to add in specific colours (if you're only using a handful of specific ones like we are here) is to pass the RGB code for that colour. If you have never heard of RGB colours, essentially it's a code of alphanumeric characters that form a specific colour on the colourwheel. Learn more on RGB colours here. Interestingly,

```
ggplot() +
  geom_point(data = df_filtered %>%
               dplyr::arrange(factor(species,
                        levels = c("Cutthroat trout",
                                   "Coastal giant salamander",
                                   "Cascade torrent salamander"))),
             aes(x = length_1_mm, y = weight_g, fill = species, size = species),
             # here's where we'll add our colour etc
             colour = "black", shape = 21, alpha = 0.8)+
  labs(x = "Length (mm)", y = "Weight (g)") +
   # note here we use `ggthemes::` to indicate which package we're using
  ggthemes::theme_base() +
  scale_size_manual("Species", values = c(4, 2, 2)) +
```

```
  scale_fill_manual("Species", values = c("#42d4f4", "#f032e6", "#ffe119"))
```

```
## Warning: Removed 13276 rows containing missing values (geom_point).
```



While these colours are harsh, they're easy to see. We could choose others if we want to be more aesthetic :) Also, we notice that our legend problem is gone! Why? Well, what created the problem before was having *technically* two different legends being asked for from our `aes()` arguments - size & fill. However, by manually providing values for both and then giving them the *same* legend title, we fixed up the problem. *On your own, try adding a different title for one of them and see what happens!.*

For this plot, we are pretty much done! This looks good to go. This section has hopefully showed you how to iteratively add components to your plot to troubleshoot throughout and end up with a nice plot quickly. At this point, we can save our plot if we want to (and if it has been assigned to a variable). (LINK TO SAVING SECTION OF GGSAVE).