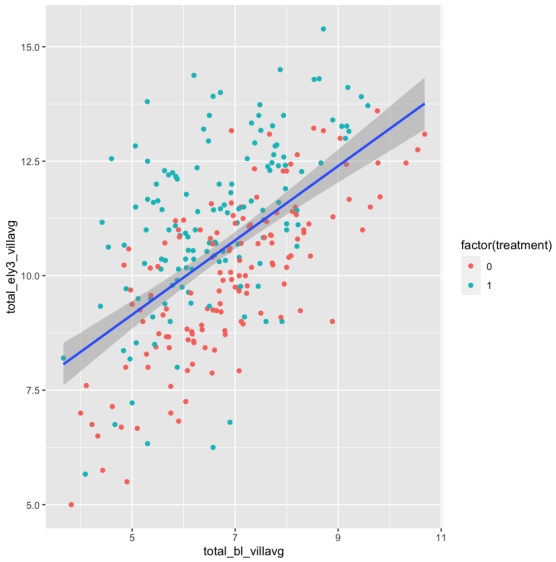
Data visualizations using R

Customizing `ggplot` graphs #1

So far, we have used aesthetic mappings and fixed aesthetic values to change visual elements of our graphs. In this section, we will modify elements such as the axis limits, labels and legend.

Let's start by loading our graph from the previous section.

In [19]:
```
library("tidyverse")

mydata <- read_csv("~/Dropbox (IDinsight)/Data visualization library/Data/EG_DIB.csv", show_col_types = FALSE)
mydata$district <- factor(sample.int(3, nrow(mydata), replace = T))

plot1 <- ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
    geom_point(aes(color = factor(treatment)), na.rm = TRUE) +
    geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE)
plot1
```



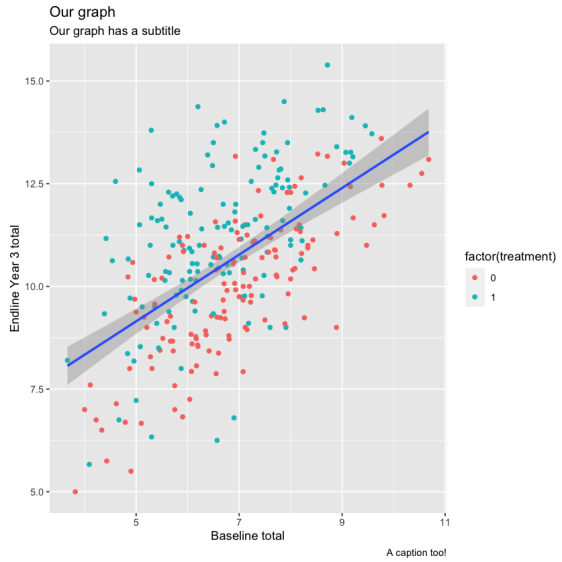
In `ggplot`, visual modifications are split between data and non-data elements. This is not the easiest distinction to put in words. But let's say we are modifying the x-axis labels. Changing the labels themselves would be considered a data element visual change. But if we change the rotation of the labels, it would be considered a non-data element visual change. Non-data element visual modifications require the use of the `theme()` function; more on that later.

Data elements

Graph and axis titles

`labs`

In [20]:
```
plot2 <- plot1 +
        labs(
                title = "Our graph",
                subtitle = "Our graph has a subtitle",
                caption = "A caption too!",
                x = "Baseline total",
                y = "Endline Year 3 total"
        )
plot2
```

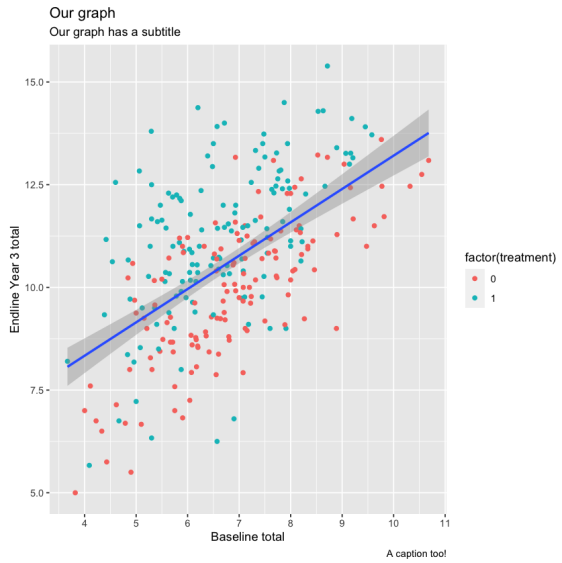


Axis labels

`scale_x_continous` and `scale_y_continuous`

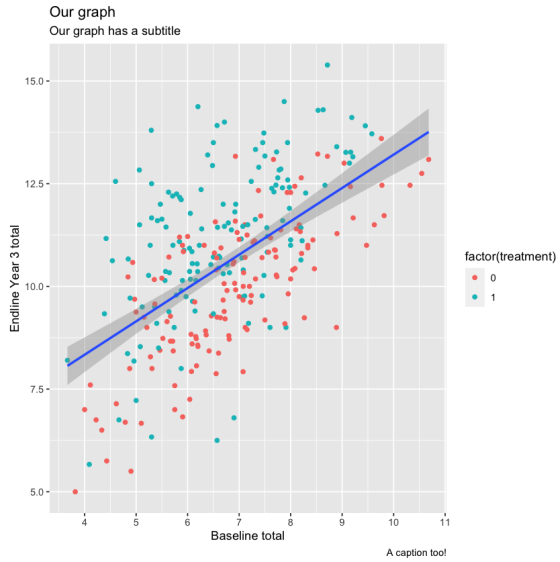
Note that our x and y axis variables are both *continuous*, that is, the variables have values such as 5.2, 7.8, 9.9, etc. This requires us to use the *continuous* form of the parameters. If our x (or y) variable was discrete (categorical, such as caste or religion), we would need to use `scale_x_discrete` (or `scale_y_discrete`).

In [21]:
```
plot3 <- plot2 +
    scale_x_continuous(breaks = c(4, 5, 6, 7, 8, 9, 10, 11))

plot3
```



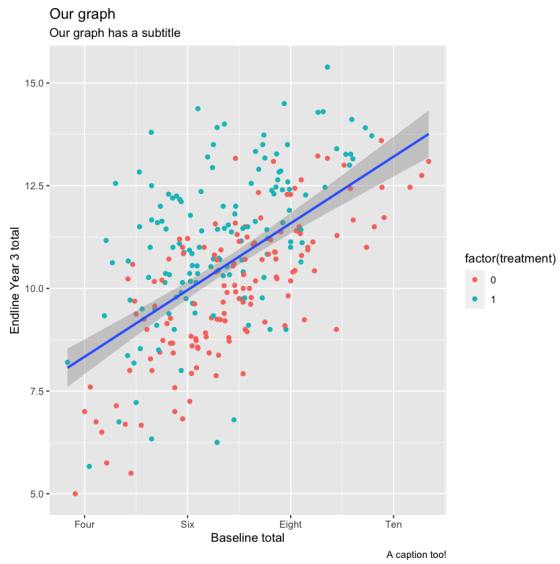
Writing down each label might be tedious, so we can use the `seq` function:

In [22]:
```
plot3 <- plot2 +
    scale_x_continuous(breaks = seq(from = 4, to = 11, by = 1))

plot3
```



We can change the axis tick labels using `labels` parameter:

In [23]:
```
plot2 +
    scale_x_continuous(
        breaks = seq(from = 4, to = 11, by = 2),
        labels = c(
            "4" = "Four",
            "6" = "Six",
            "8" = "Eight",
            "10" = "Ten"
        )
    )
```



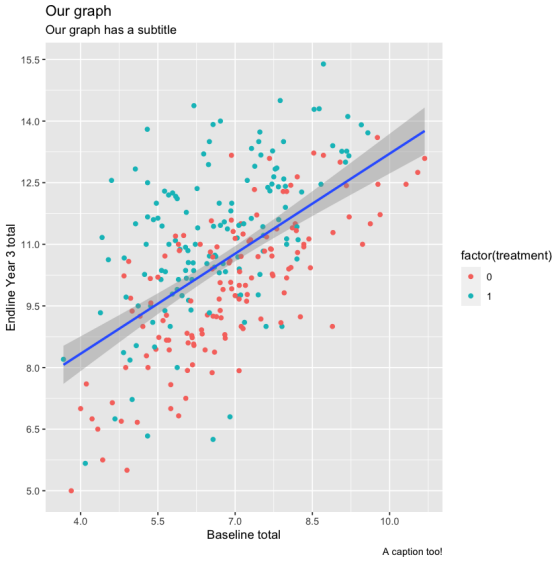
Important to note however that the number of labels should equal the number of breaks. The following code will return an error:

In [24]:
```
plot2 +
    scale_x_continuous(
        breaks = seq(from = 4, to = 11, by = 1),
        labels = c(
            "4" = "Four",
            "6" = "Six",
            "8" = "Eight",
            "10" = "Ten"
        )
    )
```

```
Error in `check_breaks_labels()`:
! `breaks` and `labels` must have the same length
Traceback:

1. scale_x_continuous(breaks = seq(from = 4, to = 11, by = 1), labels = c(`4` = "Four",
 .     `6` = "Six", `8` = "Eight", `10` = "Ten"))
2. continuous_scale(ggplot_global$x_aes, "position_c", identity,
 .     name = name, breaks = breaks, n.breaks = n.breaks, minor_breaks = minor_breaks,
 .     labels = labels, limits = limits, expand = expand, oob = oob,
 .     na.value = na.value, trans = trans, guide = guide, position = position,
 .     super = ScaleContinuousPosition)
3. check_breaks_labels(breaks, labels)
4. abort("`breaks` and `labels` must have the same length")
5. signal_abort(cnd, .file)
```

It doesn't make a lot of sense to show the axis labels as words:

In [25]:
```
plot3 <- plot2 +
        scale_x_continuous(breaks = seq(from = 4, to = 12, by = 1.5)) +
        scale_y_continuous(breaks = seq(from = 5, to = 16, by = 1.5))

plot3
```



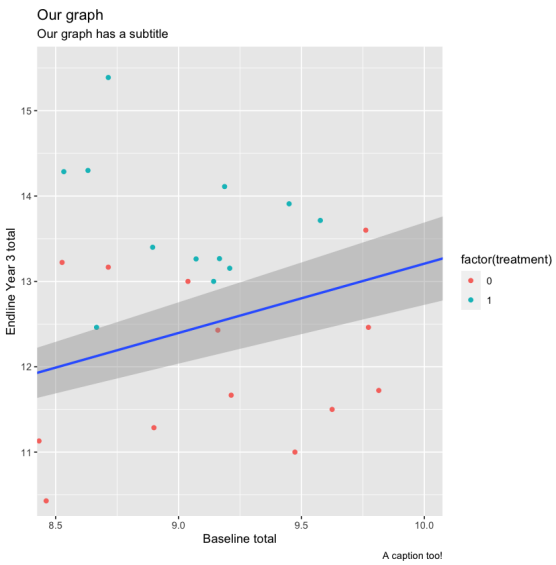
Axis limits

Suppose we are interested in changing how much of the axis we are interested in showing. We'll discuss two ways of doing this:

1. `xlim` and `ylim` in `coord_cartesian`
2. `limits` in `scale_x_continuous` and `scale_y_continuous`

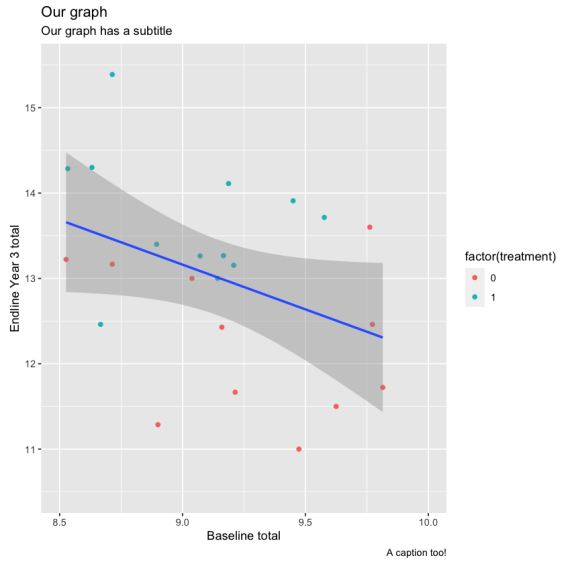
Let's use both and see how they differ. We will limit the x-axis between 8.5 and 10 and the y-axis to 10.5 and 15.5.
`coord_cartesian`

In [26]:
```
plot2 +
    coord_cartesian(
        xlim = c(8.5, 10),
        ylim = c(10.5, 15.5)
    )
```



`scale_x_continuous` and `scale_y_continuous`

In [27]:
```
plot2 +
    scale_x_continuous(limits = c(8.5, 10)) +
    scale_y_continuous(limits = c(10.5, 15.5))
```

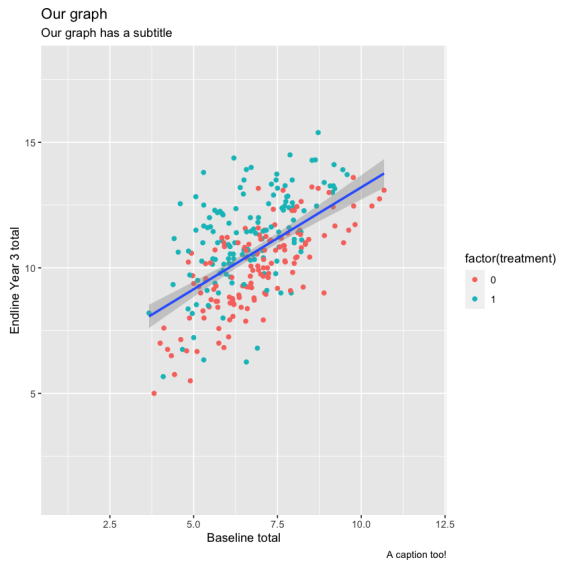


Two completely different graphs!

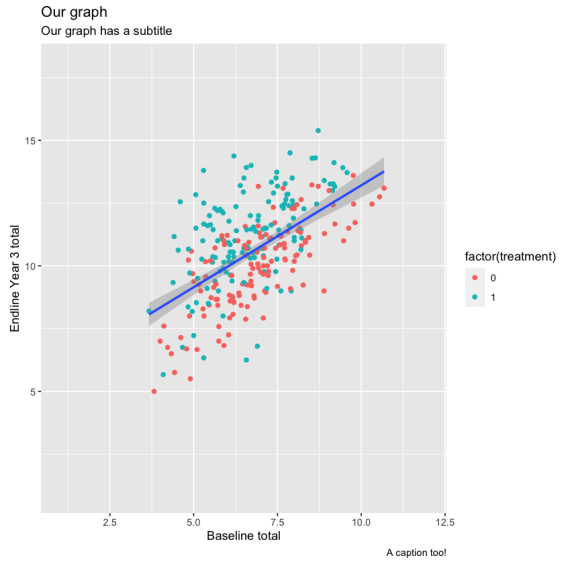
By using `coord_cartesian`, we are able to *zoom in* to the graph. However, if we use `scale_x_continuous` (or `scale_y_continuous`), we are *limiting* the data points being be plotted. Consequently, the fitted line and its accompanying CI envelope also changes. So if you are only interested in zooming in to a specific range of the graph, we recommend using `coord_cartesian`.

However if you are interested in *zooming out*, both options will work in the same way.

In [28]:
```
plot2 +
    coord_cartesian(
        xlim = c(1, 12),
        ylim = c(1, 18)
    )
```



In [29]:
```
plot2 +
    scale_x_continuous(limits = c(1, 12)) +
    scale_y_continuous(limits = c(1, 18))
```



Tip: You will find examples in the data visualization library where `scale_x_continuous` and `scale_y_continuous` have been used to *zoom out*.

Legend

Whenever an aesthetic mapping is provided, `ggplot` automatically associates it with **exactly** one *scale*. To explain this better, let us revisit the code used to create `plot1`:

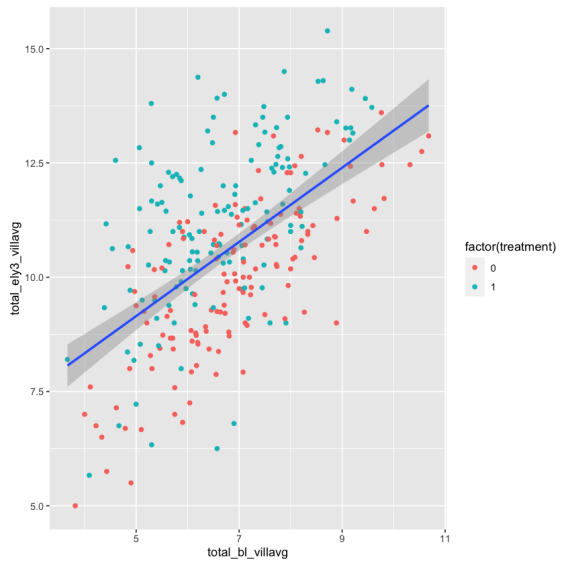
```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
            geom_point(aes(color = factor(treatment)), na.rm = TRUE) +
            geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE)
```

`ggplot` adds default scales equivalent to:

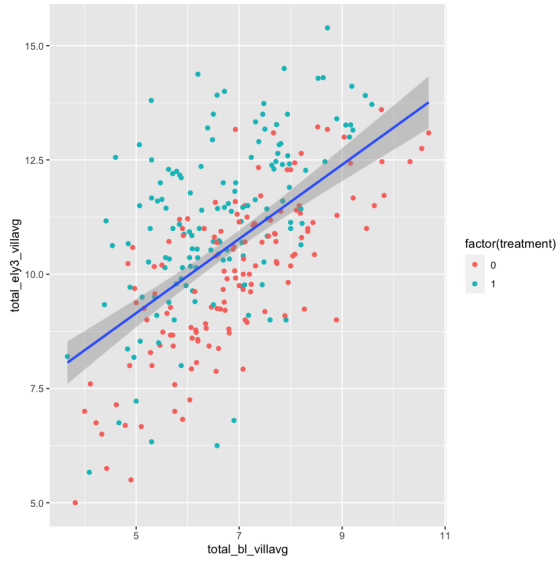
```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
            geom_point(aes(color = factor(treatment)), na.rm = TRUE) +
            geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE) +
            scale_x_continuous() +
            scale_y_continuous() +
            scale_color_discrete()
```

Let's execute and verify that they are indeed the same.

In [30]:
```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
    geom_point(aes(color = factor(treatment)), na.rm = TRUE) +
    geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE)
```



In [31]:
```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
    geom_point(aes(color = factor(treatment)), na.rm = TRUE) +
    geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE) +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_color_discrete()
```




```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
            geom_point(aes(color = factor(treatment)), na.rm = TRUE) +
            geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE) +
            scale_x_continuous() +
            scale_y_continuous() +
            scale_color_discrete()
```

In this code block, aesthetic mappings have been used 3 times:

1. `x = total_bl_villavg` : corresponding to `scale_x_continuous()`
2. `y = total_ely3_villavg` : corresponding to `scale_y_continuous()`
3. `color = factor(treatment)` : corresponding to `scale_color_discrete()`

But what does this have to do with legends?

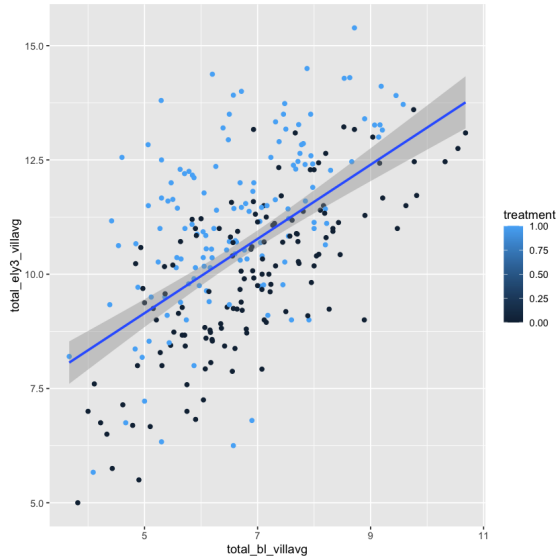
Every scale is associated with a guide that displays the relationship between the aesthetic (`x` , `y` , `color` , `fill` , `shape` , `size` , etc.) and the data. Positional scales are displayed using the axes. For color scales, this role is performed through a legend. Hence, to modify the legend, we need to use `scale_color_discrete` . Why discrete? Because the `color` aesthetic is mapped to a factor variable. Let's see what happens if we remove the `factor()` function:

In [32]:
```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
    geom_point(aes(color = treatment), na.rm = TRUE) +
    geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE) +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_color_discrete()
```

```
ERROR while rich displaying an object: Error: Continuous value supplied to discrete scale

Traceback:
1. tryCatch(withCallingHandlers({
 .     if (!mime %in% names(repr::mime2repr))
 .         stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
 .     rpr <- repr::mime2repr[[mime]](obj)
 .     if (is.null(rpr))
 .         return(NULL)
 .     prepare_content(is.raw(rpr), rpr)
 . }, error = error_handler), error = outer_handler)
2. tryCatchList(expr, classes, parentenv, handlers)
3. tryCatchOne(expr, names, parentenv, handlers[[1L]])
4. doTryCatch(return(expr), name, parentenv, handler)
5. withCallingHandlers({
 .     if (!mime %in% names(repr::mime2repr))
 .         stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
 .     rpr <- repr::mime2repr[[mime]](obj)
 .     if (is.null(rpr))
 .         return(NULL)
 .     prepare_content(is.raw(rpr), rpr)
 . }, error = error_handler)
6. repr::mime2repr[[mime]](obj)
7. repr_text.default(obj)
8. paste(capture.output(print(obj)), collapse = "\n")
9. capture.output(print(obj))
10. evalVis(expr)
11. withVisible(eval(expr, pf))
12. eval(expr, pf)
13. eval(expr, pf)
14. print(obj)
15. print.ggplot(obj)
16. ggplot_build(x)
17. ggplot_build.ggplot(x)
18. lapply(data, scales_train_df, scales = npscales)
19. FUN(X[[i]], ...)
20. lapply(scales$scales, function(scale) scale$train_df(df = df))
21. FUN(X[[i]], ...)
22. scale$train_df(df = df)
23. f(..., self = self)
24. self$train(df[[aesthetic]])
25. f(..., self = self)
26. self$range$train(x, drop = self$drop, na.rm = !self$na.translate)
27. f(..., self = self)
28. scales::train_discrete(x, self$range, drop = drop, na.rm = na.rm)
29. stop("Continuous value supplied to discrete scale", call. = FALSE)
```

We get an error saying that continuous values have been supplied to a discrete value. To fix this, we will need to use `scale_color_continuous` as shown here:

In [33]:
```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
    geom_point(aes(color = treatment), na.rm = TRUE) +
    geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE) +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_color_continuous()
```

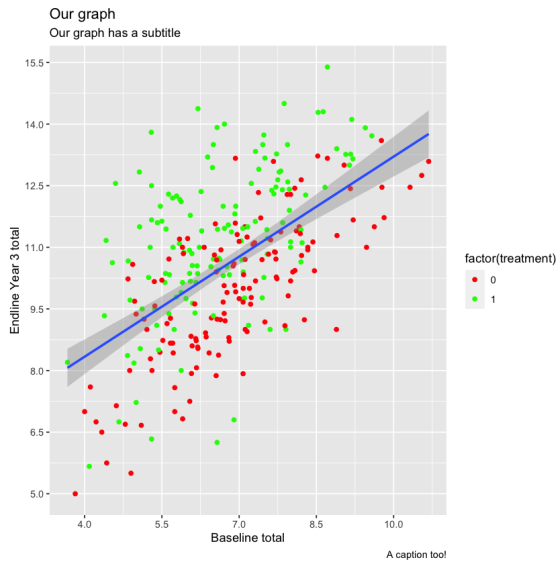


Now that we have cleared up the link between the legend and the aesthetic mappings, let us explore a few common visual modifications.

Being the default option, `scale_color_discrete` does not offer a lot of customization options and hence, we will be using **`scale_color_manual`**.

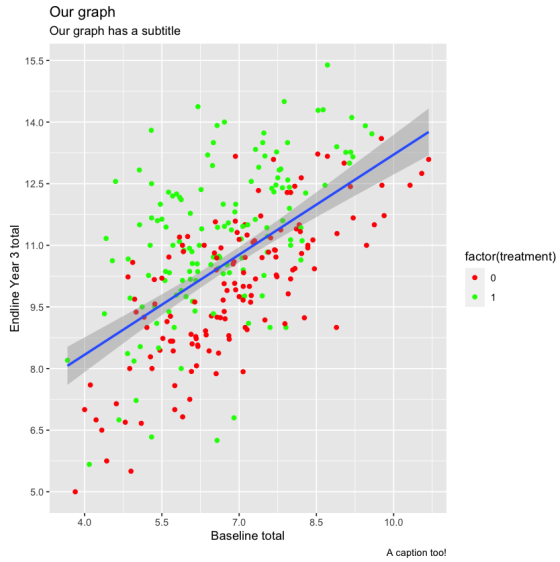
`values` for changing the color of the legend levels

In [34]:
```
plot3 +
    scale_color_manual(values = c("red", "green"))
```



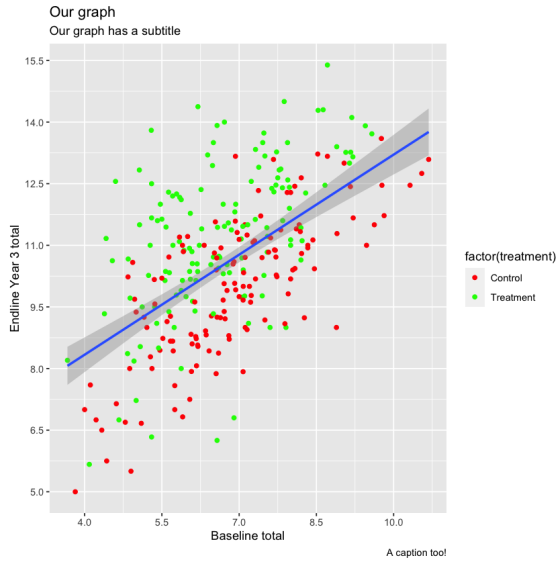
The treatment variable has only two possible values, 0 and 1, which are easy to track and you can write down the `values` directly. However, if there were multiple levels, for example different treatment arms, keeping track of each level would be difficult. In such situations, it is recommended to use a named vector.

In [35]:
```
colors <- c("0" = "red", "1" = "green")

plot3 +
    scale_color_manual(values = colors)
```



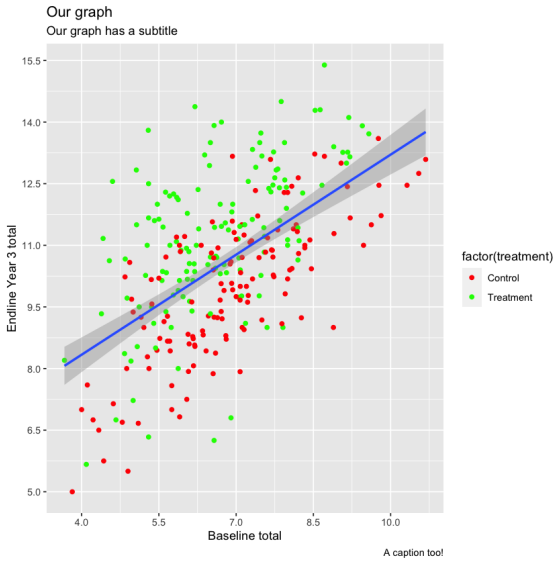
`labels` for changing the legend key text

In [36]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = c("Control", "Treatment")
    )
```



But as with the colors, here too, it is recommend to store the labels in a vector so that you can keep track of them.

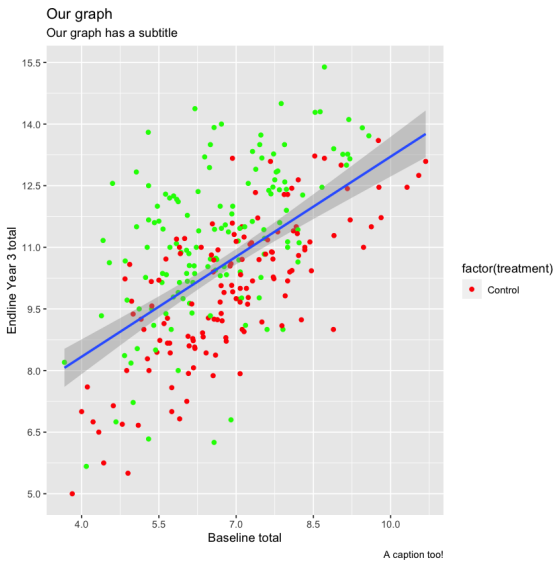
In [37]:
```
labs <- c("1" = "Treatment", "0" = "Control")

plot3 +
    scale_color_manual(
        values = colors,
        labels = labs
    )
```



`breaks` for changing which which keys get displayed

Please note that the number of labels must match the number of breaks. In the following example, since we are showing only key, we must also pass on one label.

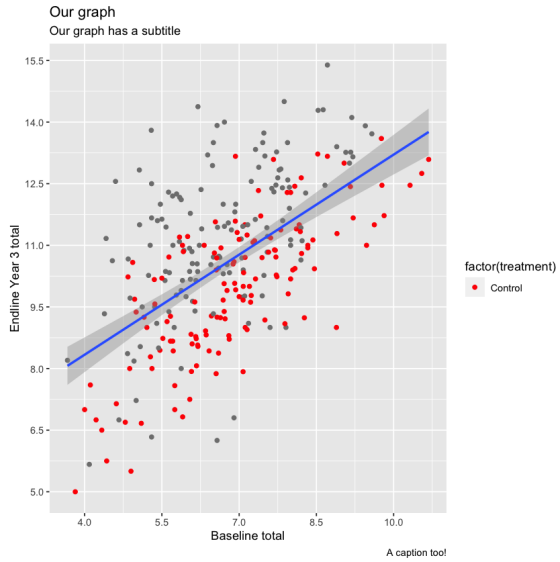
In [38]:
```
labs1 <- c("0" = "Control")

plot3 + scale_color_manual(
    values = colors,
    labels = labs1,
    breaks = c("0")
)
```



`limits` to change the possible values of the scale

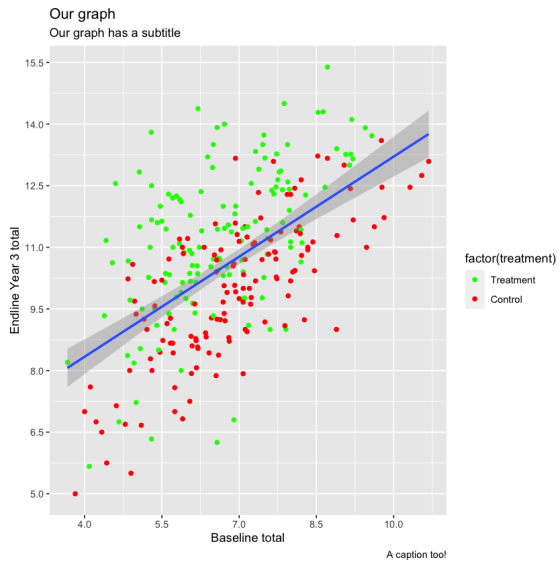
Note the difference between `breaks` and `limits`!

In [39]:
```
labs <- c("1" = "Treatment", "0" = "Control")

plot3 +
    scale_color_manual(
        values = colors,
        labels = labs,
        limits = c("0")
    )
```



`limits` can also be used to change the order of the legend keys.

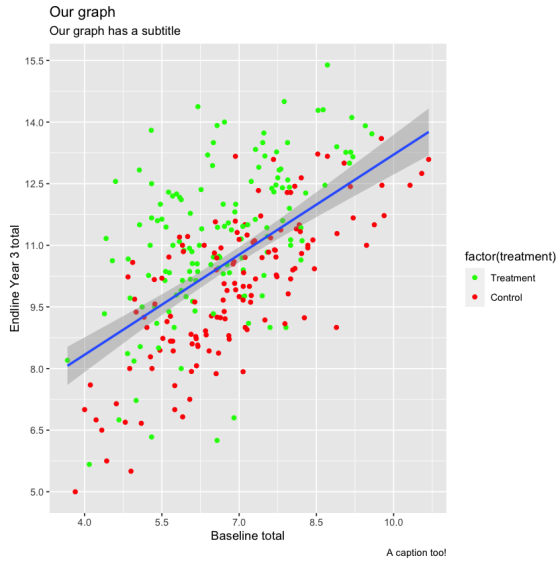
In [40]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs,
        limits = c("1", "0")
    )
```



`guides` layer to reverse the order of the legend

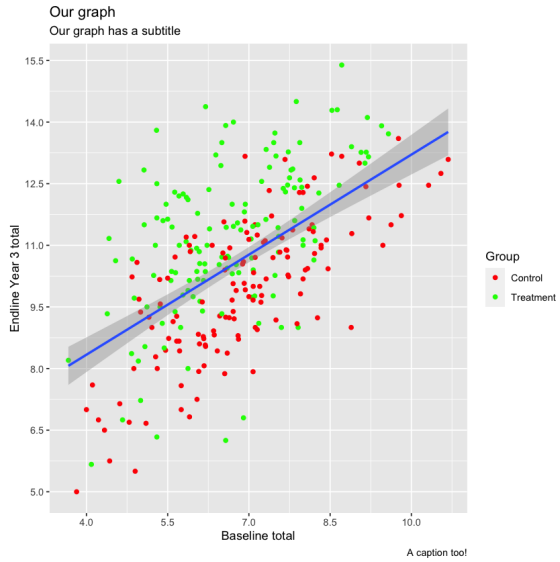
Useful for quickly reversing the order as you don't have to specify the position of each manually.

In [41]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs
    ) +
    guides(color = guide_legend(reverse = TRUE))
```



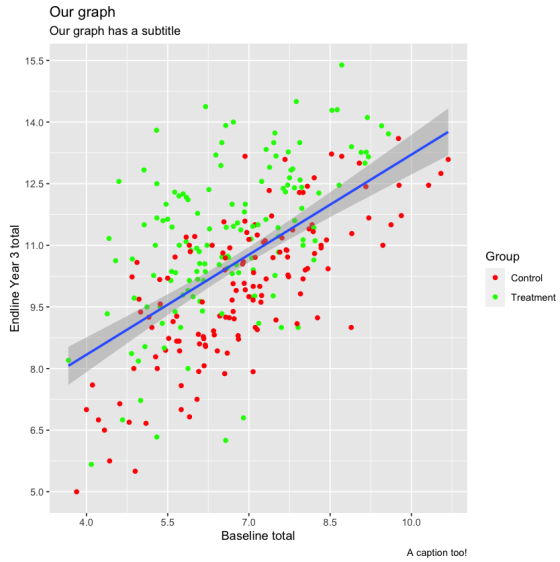
`name` to change the legend title

In [42]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs,
        name = "Group"
    )
```



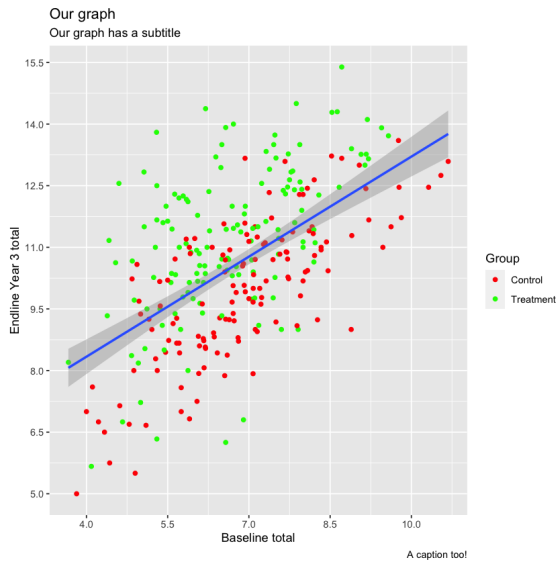
You can also use the `guides` layer to change the legend title

In [43]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs
    ) +
    guides(color = guide_legend(title = "Group"))
```



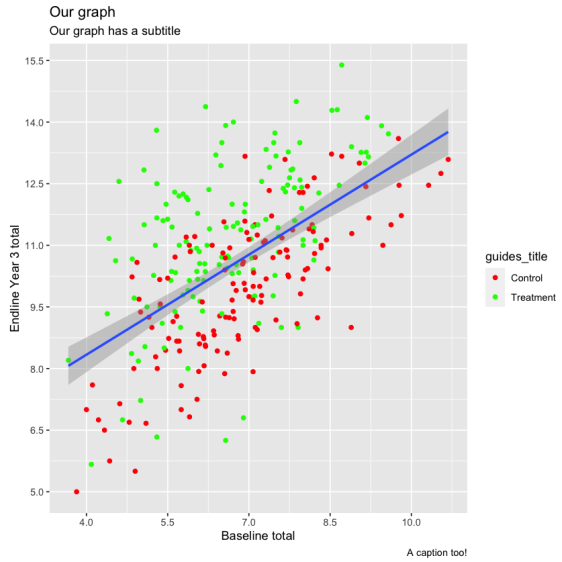
Alternatively, you can also use the `labs` layer.

In [44]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs
    ) +
    labs(color = "Group")
```



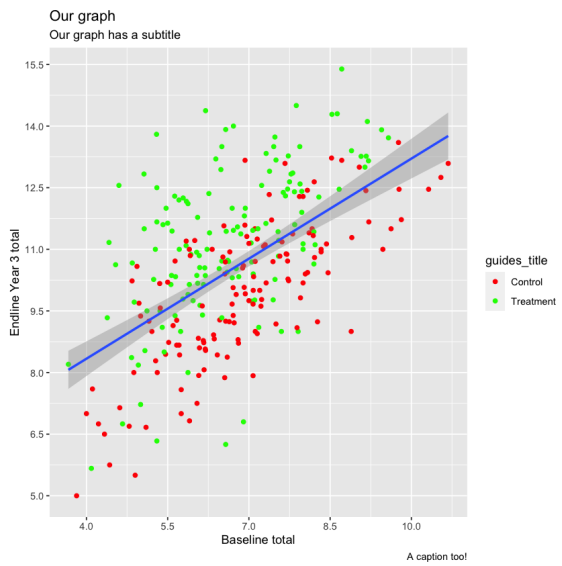
If you use `labs` , `name` and `guides` at the same time, the latter will be applied.

In [45]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs,
        name = "name_title"
    ) +
    guides(color = guide_legend(title = "guides_title")) +
    labs(color = "labs_title")
```



And of course, the order in which you write the layers don't matter.

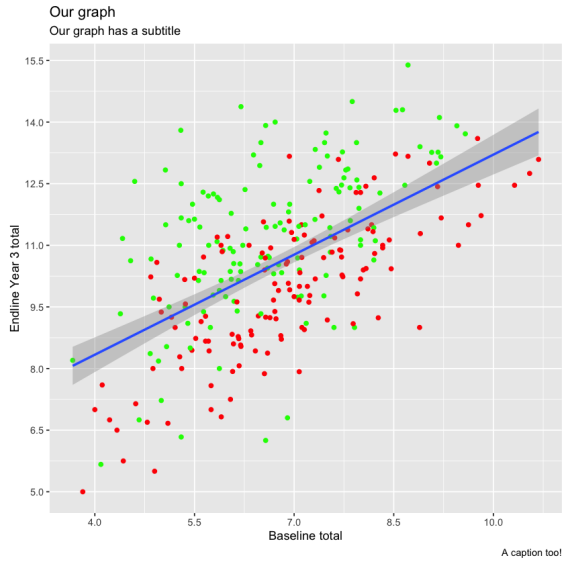
In [46]:
```
plot3 +
    labs(color = "labs_title") +
    guides(color = guide_legend(title = "guides_title")) +
    scale_color_manual(
        values = colors,
        labels = labs,
        name = "name_title"
    )
```



We would recommend using the `name` parameter of the `scale` layer since it is likely that you will be coding that out regardless.

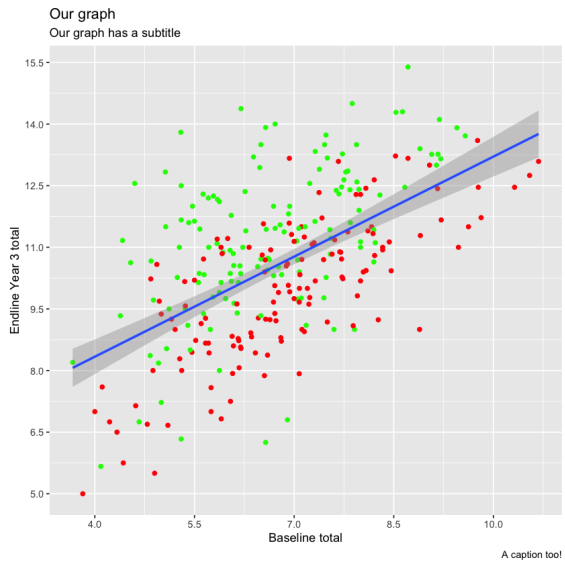
`guides` to remove a legend

In [47]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs
    ) +
    guides(color = "none")
```



Alternatively, you can also use the `guide` parameter from the `scale` layer to remove the legend.

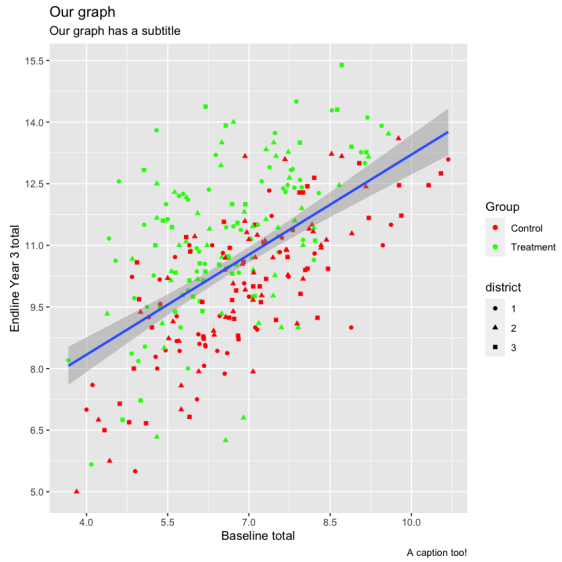
In [48]:
```
plot3 +
    scale_color_manual(
        values = colors,
        labels = labs,
        guide = "none"
    )
```



What if there is more than 1 legend?

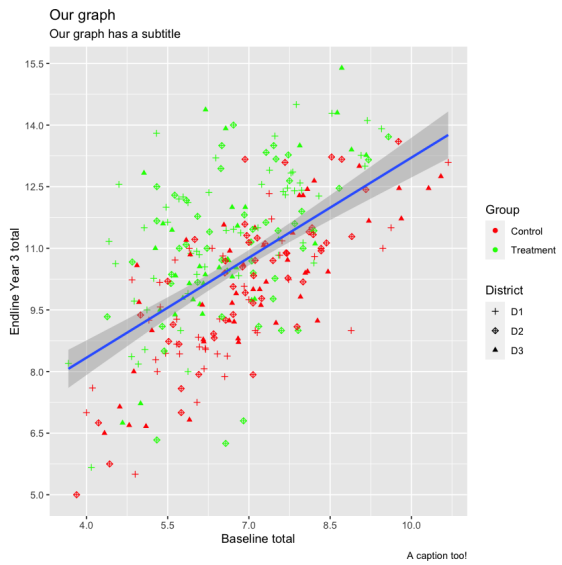
In our graph, we have used the `color` aesthetic mapping to distinguish the points between the treatment and control groups. As discussed previously, each aesthetic is linked to a scale. The `color` aesthetic produces a legend. Now suppose we had another aesthetic mapping, `shape` to denote the district of each point.

In [49]:
```
plot4 <- ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
     geom_point(aes(color = factor(treatment), shape = district), na.rm = TRUE) +
     geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE) +
     scale_x_continuous(breaks = seq(from = 4, to = 12, by = 1.5)) +
     scale_y_continuous(breaks = seq(from = 5, to = 16, by = 1.5)) +
     scale_color_manual(
          values = colors,
          labels = labs,
          name = "Group"
     ) +
     labs(
          title = "Our graph",
          subtitle = "Our graph has a subtitle",
          caption = "A caption too!",
          x = "Baseline total",
          y = "Endline Year 3 total"
     )
plot4
```



With the addition of the `shape` aesthetic mapping, `ggplot` adds a legend as well. We can manipulate this scale and legend using the **`scale_shape_manual`** layer.

In [50]:
```
labs_district <- c("1" = "D1", "2" = "D2", "3" = "D3")

plot4 + scale_shape_manual(
    values = c(3, 9, 17),
    name = "District",
    labels = labs_district
)
```

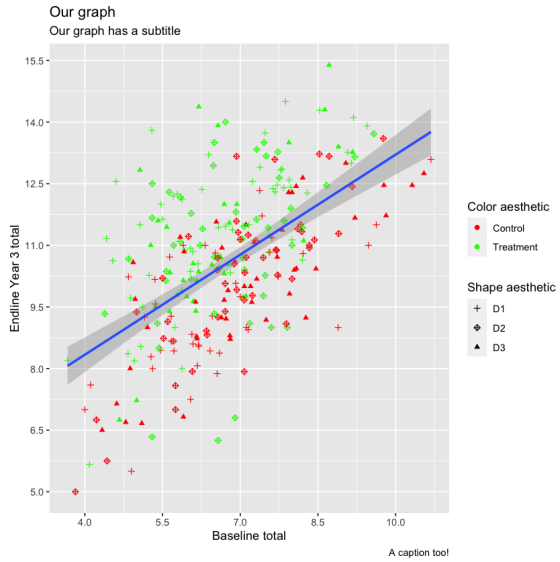


In the code block above, we have use the following parameters in the `scale_shape_manual` layer:

- `values` : to change the shapes
- `name` : to change the name of the legend for this scale
- `labels` : to change the legend key text

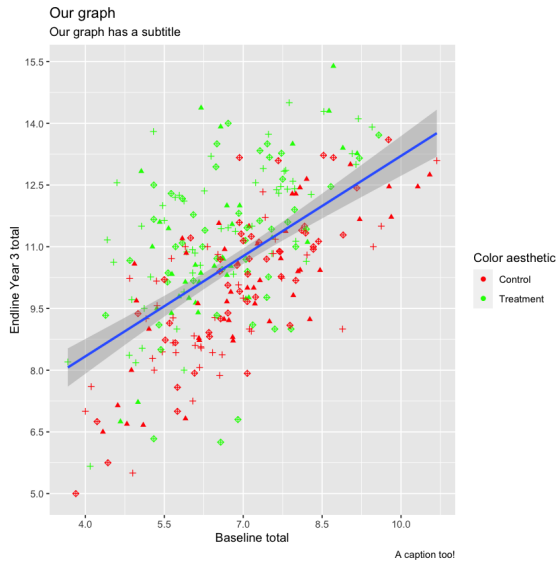
The functionality is similar to `scale_color_manual` . The `guides` layer works as anticipated:

In [51]:
```
labs_district <- c("1" = "D1", "2" = "D2", "3" = "D3")

plot4 + scale_shape_manual(
       values = c(3, 9, 17),
       name = "District",
       labels = labs_district
       ) +
       guides(
              color = guide_legend(title = "Color aesthetic"),
              shape = guide_legend(title = "Shape aesthetic")
       )
```



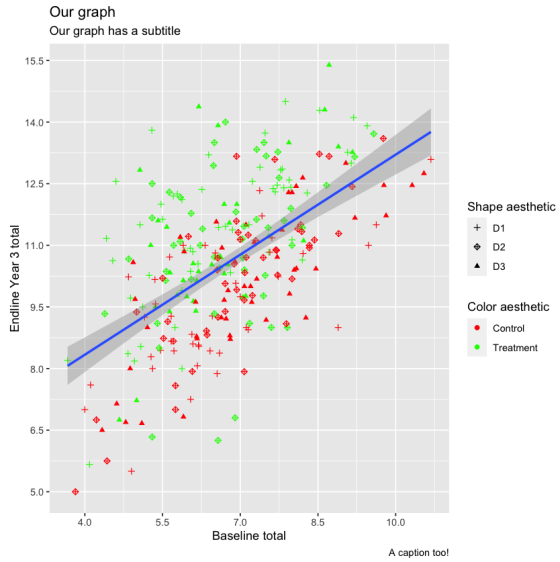
`guides` can be used to selectively remove one legend.

In [52]:
```
labs_district <- c("1" = "D1", "2" = "D2", "3" = "D3")

plot4 + scale_shape_manual(
       values = c(3, 9, 17),
       name = "District",
       labels = labs_district
) +
       guides(
              color = guide_legend(title = "Color aesthetic"),
              shape = "none"
       )
```



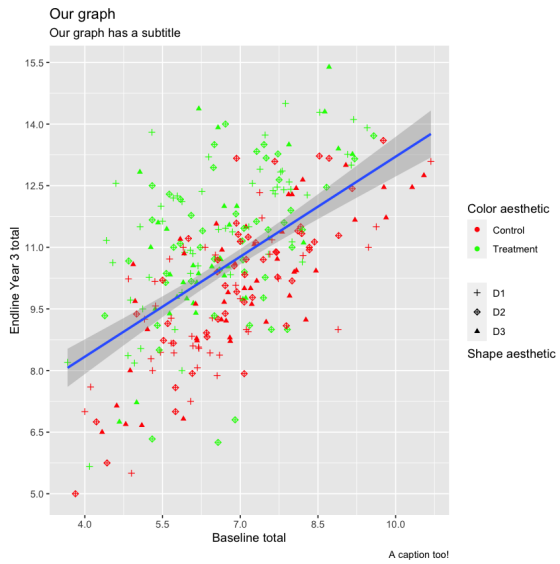
`guides` can also be used to change the order of the legends.

In [53]:
```
labs_district <- c("1" = "D1", "2" = "D2", "3" = "D3")

plot4 + scale_shape_manual(
    values = c(3, 9, 17),
    name = "District",
    labels = labs_district
) +
    guides(
        color = guide_legend(
            title = "Color aesthetic",
            order = 2
        ),
        shape = guide_legend(
            title = "Shape aesthetic",
            order = 1
        )
    )
```



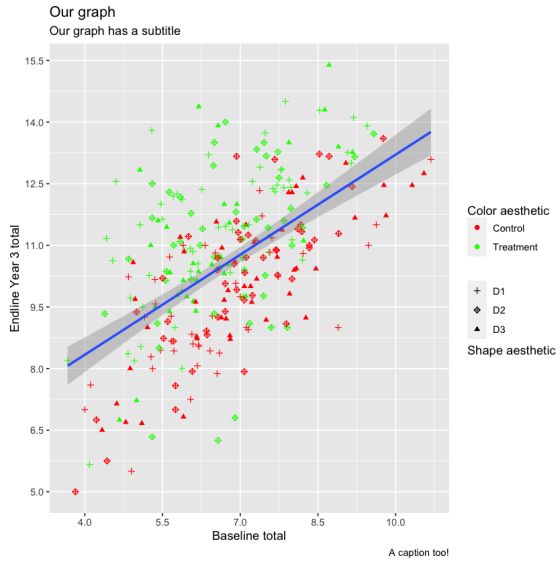
`title.position` can be used to change the position of the legend title.

In [54]:
```
plot4 + scale_shape_manual(
    values = c(3, 9, 17),
    name = "District",
    labels = labs_district
) +
    guides(
        color = guide_legend(
            title = "Color aesthetic",
            title.position = "top"
        ),
        shape = guide_legend(
            title = "Shape aesthetic",
            title.position = "bottom"
        )
    )
```



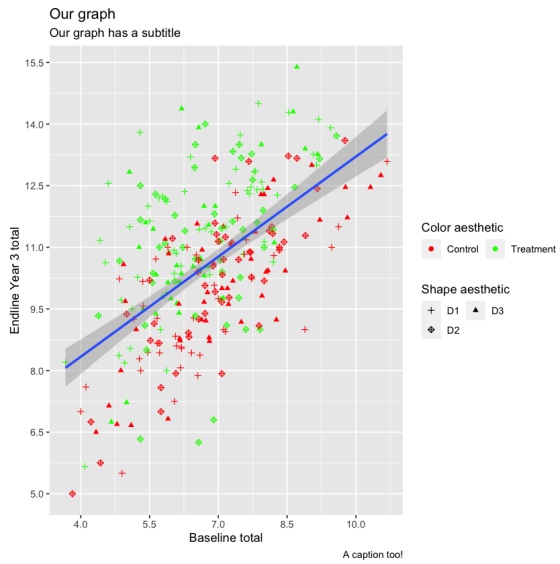
`title.hjust` and `title.vjust` can be used to change the horizontal and vertical justification of the legend title.

In [55]:
```
plot4 + scale_shape_manual(
    values = c(3, 9, 17),
    name = "District",
    labels = labs_district
) +
    guides(
        color = guide_legend(
            title = "Color aesthetic",
            title.position = "top",
            title.vjust = -1
        ),
        shape = guide_legend(
            title = "Shape aesthetic",
            title.position = "bottom",
            title.vjust = 1
        )
    )
```



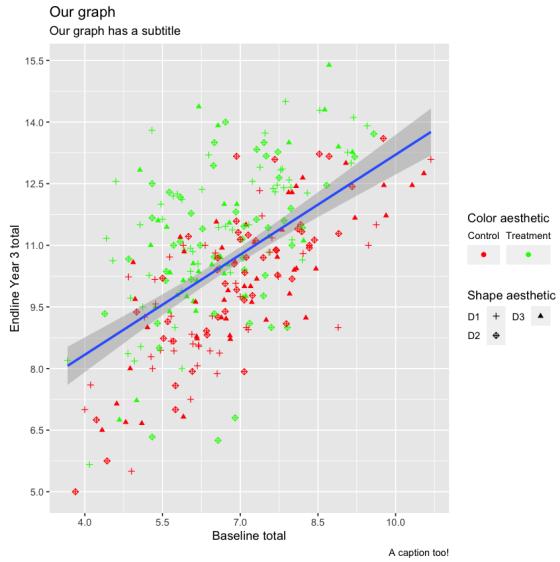
`ncol` and `nrow` can be used to change the number of rows and columns of the legend keys.

In [56]:
```
plot4 + scale_shape_manual(
    values = c(3, 9, 17),
    name = "District",
    labels = labs_district
) +
    guides(
        color = guide_legend(
            title = "Color aesthetic",
            ncol = 2
        ),
        shape = guide_legend(
            title = "Shape aesthetic",
            nrow = 2
        )
    )
```



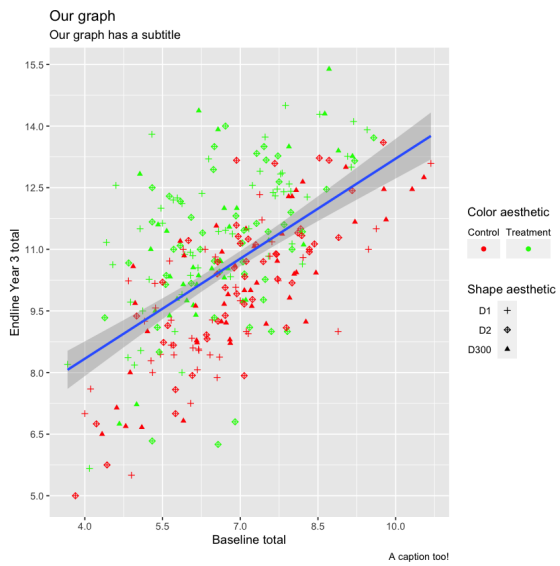
`label.position` can be used to change the position of the legend key labels.

In [57]:
```
plot4 + scale_shape_manual(
    values = c(3, 9, 17),
    name = "District",
    labels = labs_district
) +
    guides(
        color = guide_legend(
            title = "Color aesthetic",
            ncol = 2,
            label.position = "top"
        ),
        shape = guide_legend(
            title = "Shape aesthetic",
            nrow = 2,
            label.position = "left"
        )
    )
```



`label.hjust` and `label.vjust` can be used to change the horizontal and vertical justification of the legend key labels.

In [58]:
```
labs_district <- c("1" = "D1", "2" = "D2", "3" = "D300")

plot4 + scale_shape_manual(
    values = c(3, 9, 17),
    name = "District",
    labels = labs_district
) +
    guides(
        color = guide_legend(
            title = "Color aesthetic",
            ncol = 2,
            label.position = "top",
            label.vjust = -1
        ),
        shape = guide_legend(
            title = "Shape aesthetic",
            nrow = 3,
            label.position = "left",
            label.hjust = 1
        )
    )
```



Scales can get confusing, given the number of options you can choose from. This Stackoverflow article details the different options available, how they differ and their suggested usage.

In the next lesson, we will discuss more visual modifications available to us using the `theme` layer.

In [ ]:

In [ ]: