data_visualization_2

June 17, 2022

1 Data visualizations using R

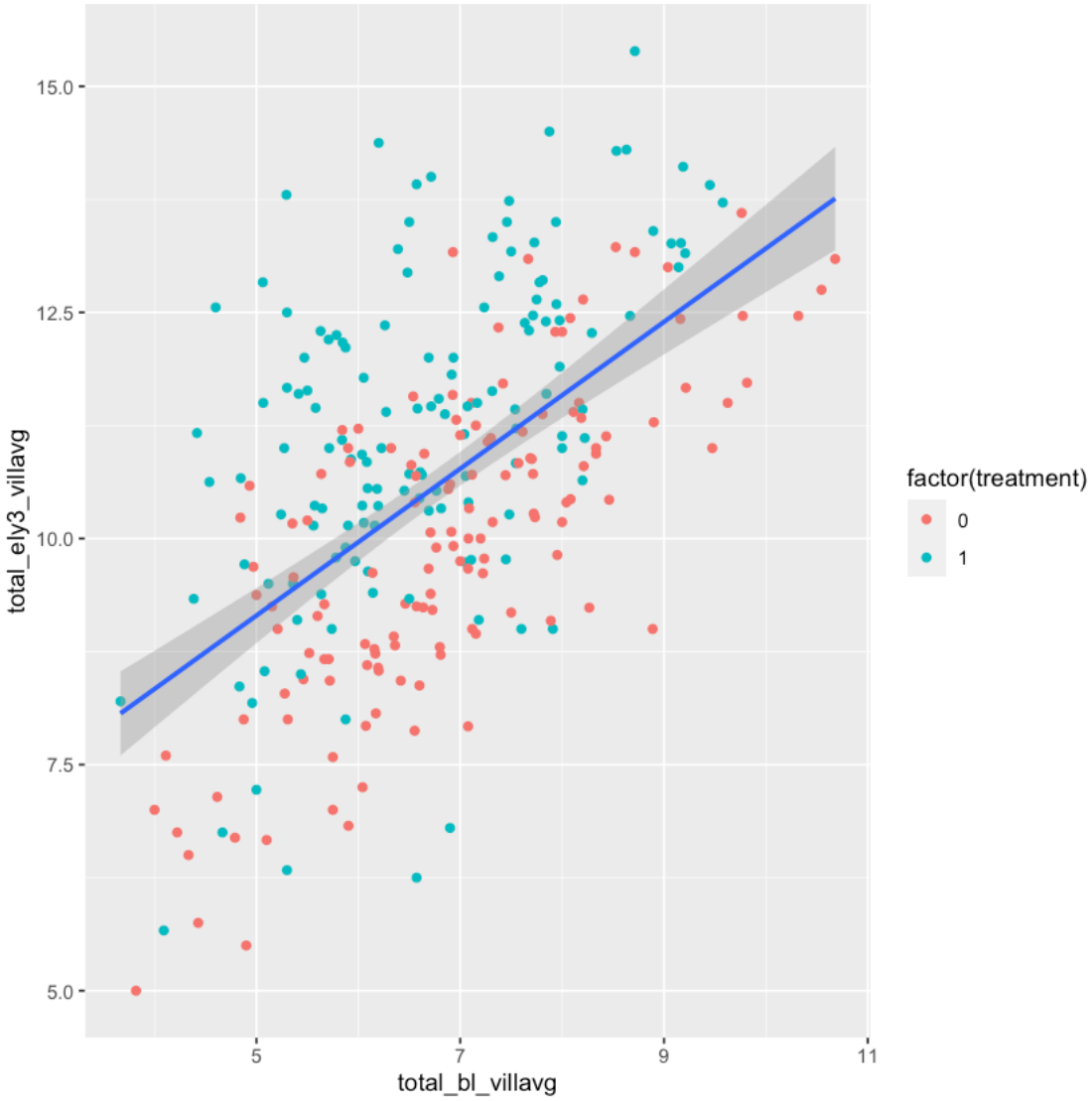
1.1 Customizing ggplot graphs

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, legends and more.

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

[2]:
```
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
```

```
  Attaching packages                                   tidyverse
1.3.1

 ggplot2 3.3.6      purrr   0.3.4
 tibble  3.1.6      dplyr   1.0.8
 tidyr   1.2.0      stringr 1.4.0
 readr   2.1.2      forcats 0.5.1

Warning message:
"package 'tidyr' was built under R version 4.0.5"
Warning message:
"package 'readr' was built under R version 4.0.5"
Warning message:
"package 'dplyr' was built under R version 4.0.5"
  Conflicts
tidyverse_conflicts()
 dplyr::filter() masks stats::filter()
 dplyr::lag()    masks stats::lag()
```



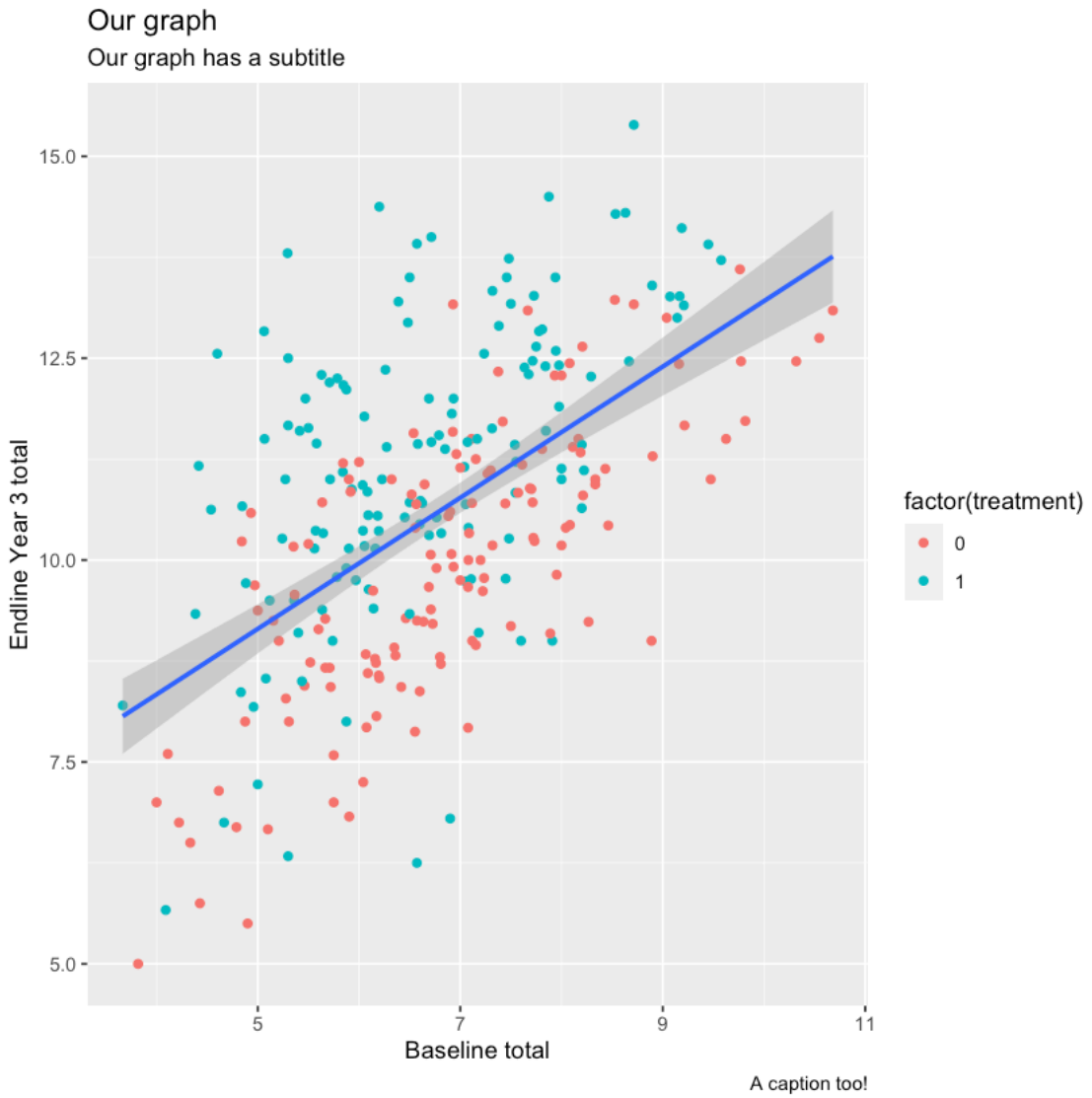
In `ggplot`, visual modifications are split between data and non-data elements. This is not the easiest distinction to put in words. But let's say we are modifying the x-axis labels. Changing the labels themselves would be considered a data element visual change. But if we only change the rotation of 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.

1.1.1 Data elements

Graph and axis titles `labs`

[3]:
```
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
```

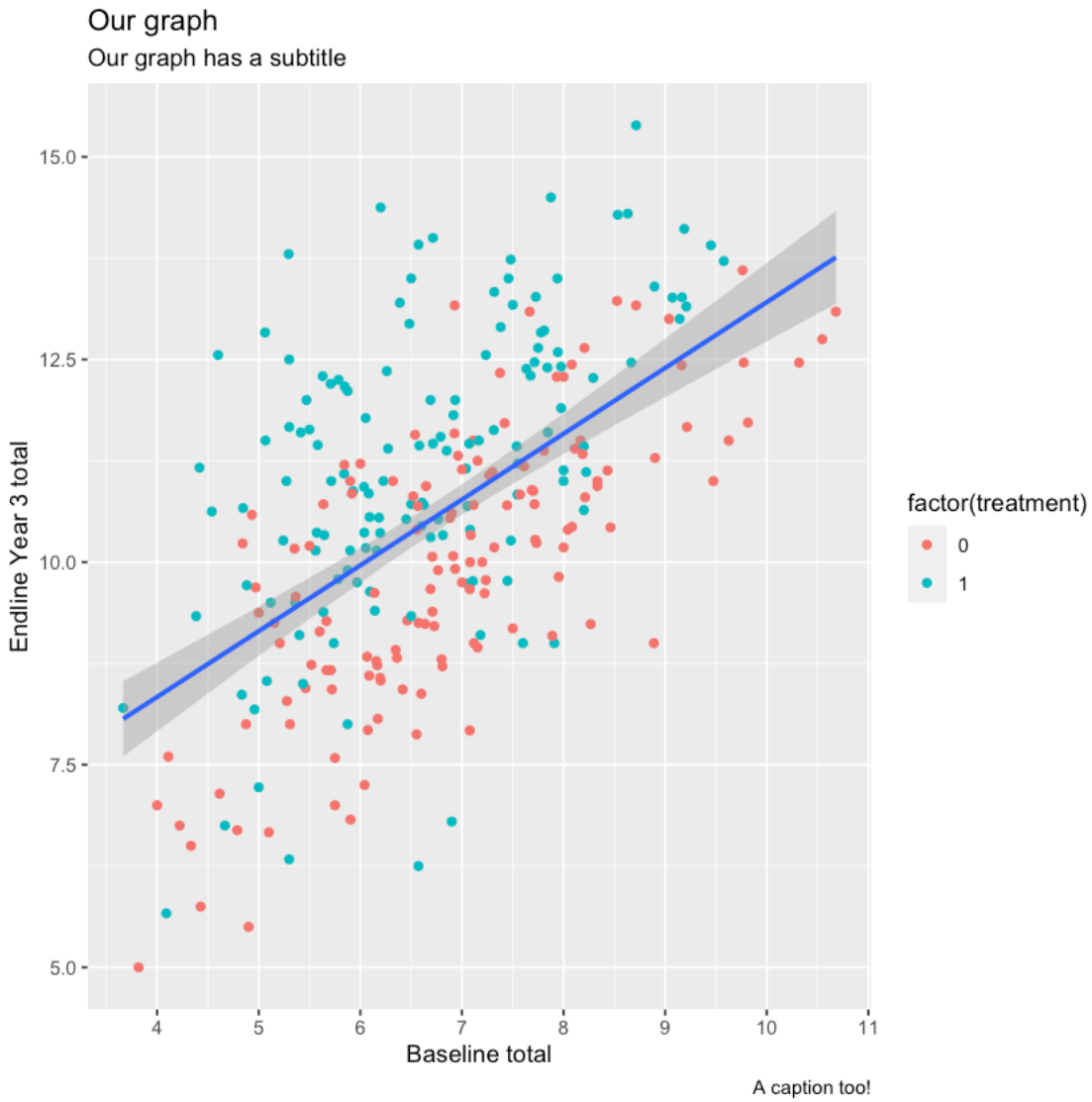


Axis labels `scale_x_continous` and `scale_y_continuous`

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`).

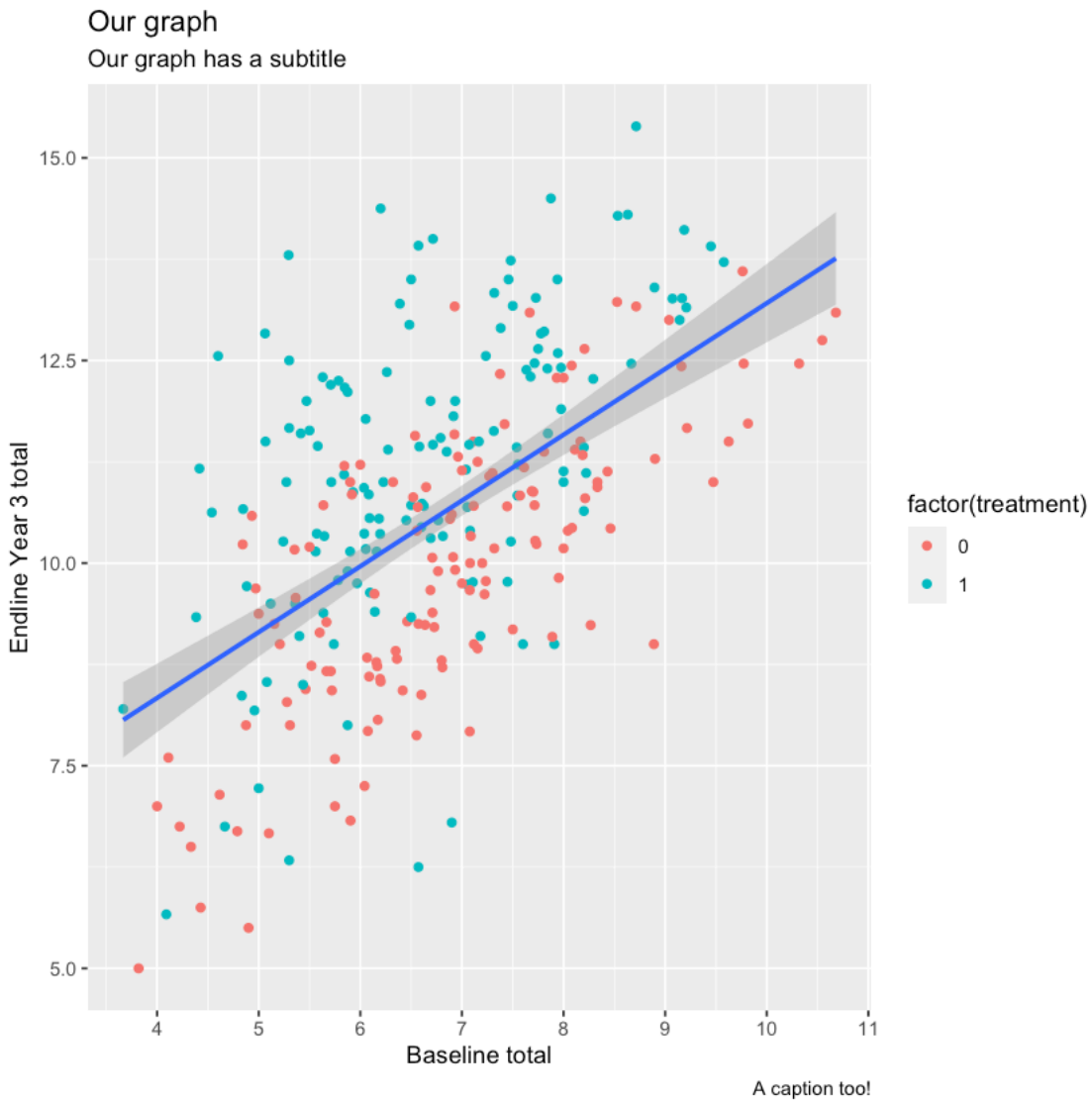
[11]:
```
plot3 <- plot2 +
    scale_x_continuous(breaks = c(4, 5, 6, 7, 8, 9, 10, 11))
```

```
plot3
```



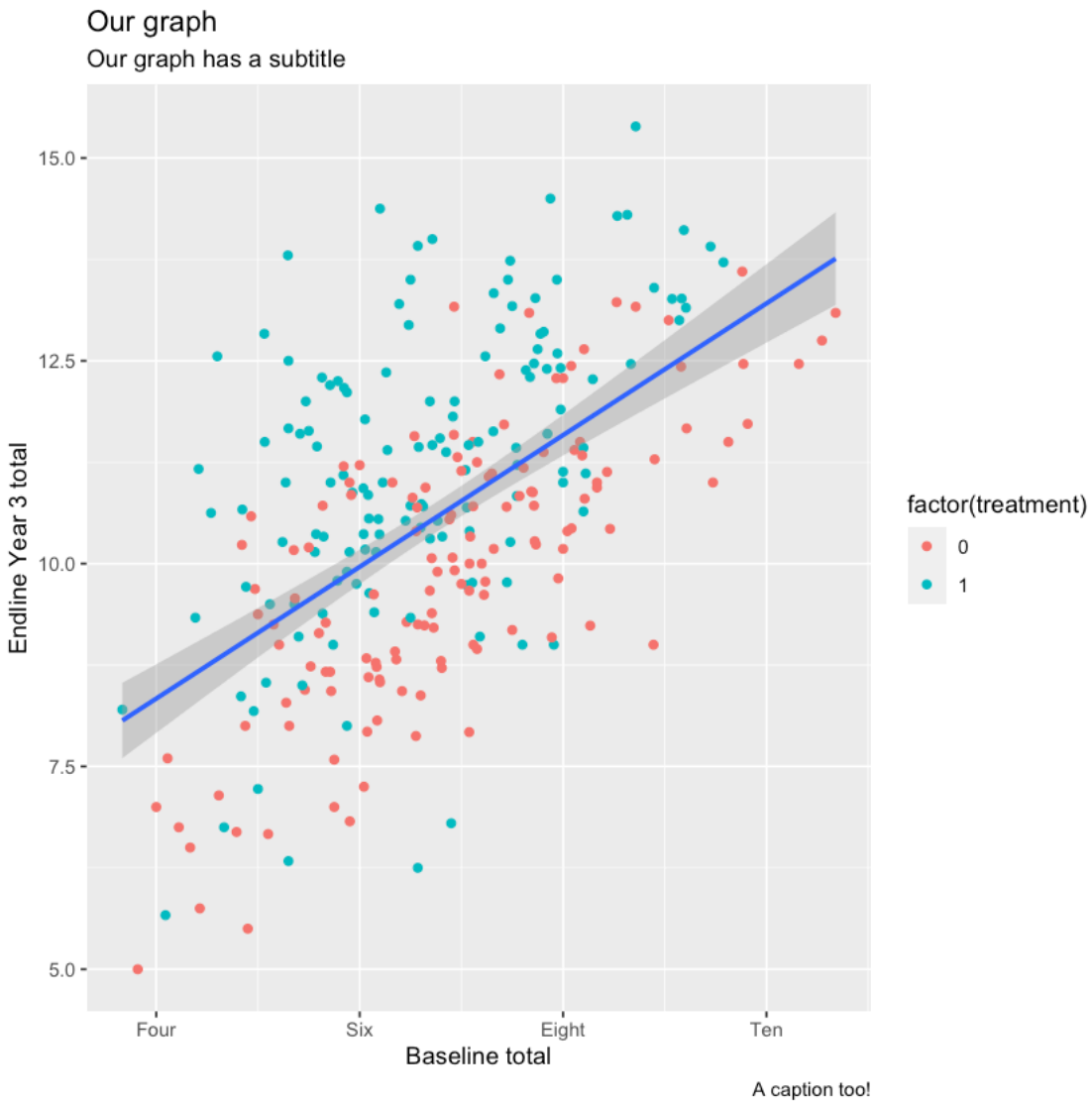
Writing down each label might be tedious, so we can use the `seq` function:

[29]:
```
plot3 <- plot2 +
    scale_x_continuous(breaks = seq(from = 4, to = 11, by = 1))

plot3
```



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

[30]:
```
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:

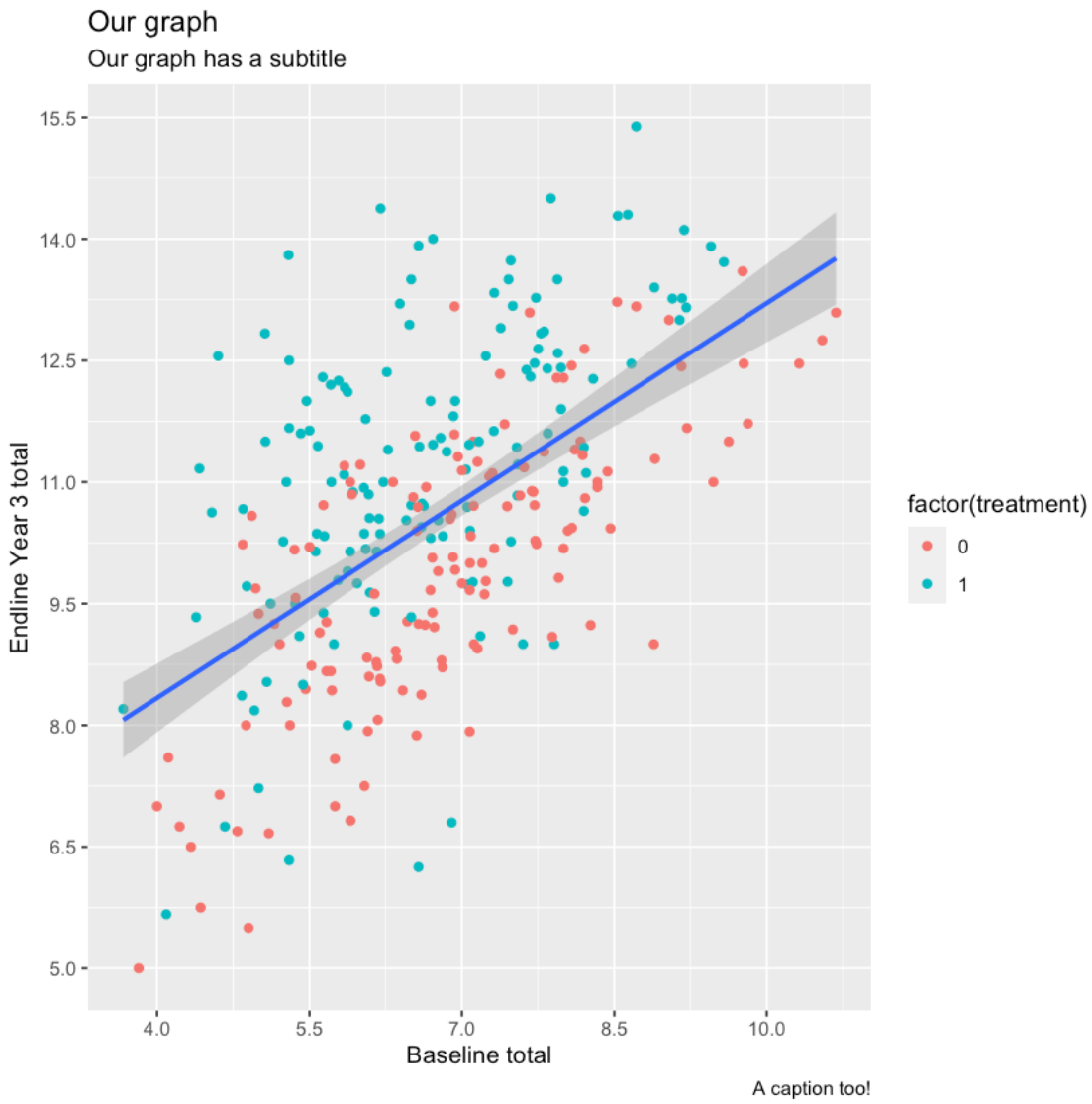
[31]:
```
plot2 +
    scale_x_continuous(breaks = seq(from = 4, to = 11, by = 1),
                       labels = c("4" = "Four",
                                  "6" = "Six",
                                  "8" = "Eight",
                                  "10" = "Ten"))
```

```
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:

[45]:
```
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
```



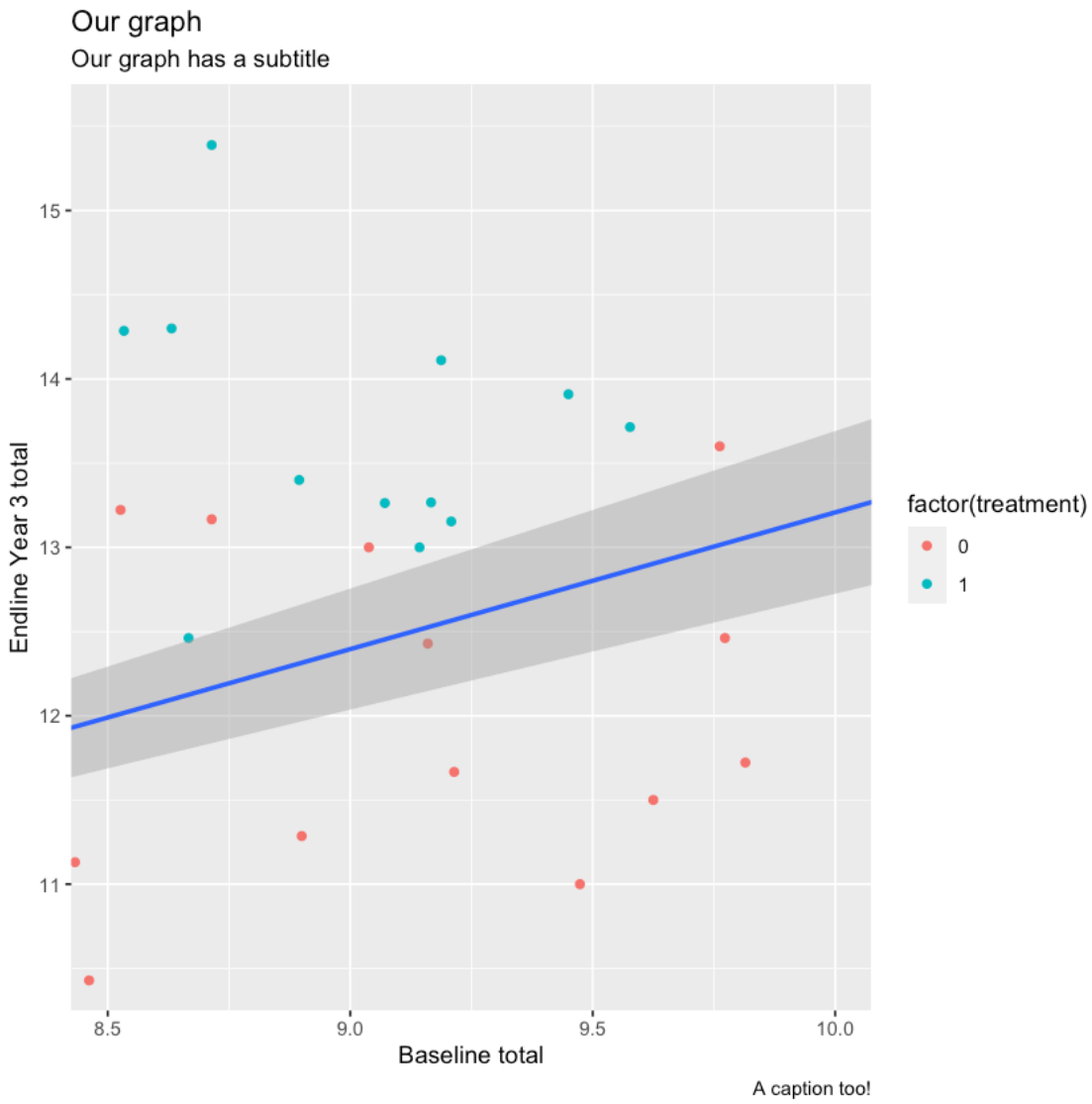
Axis limits Suppose we are interested in changing how much of the axis we are interested in showing. We'll discuss two ways of doing this:

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

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

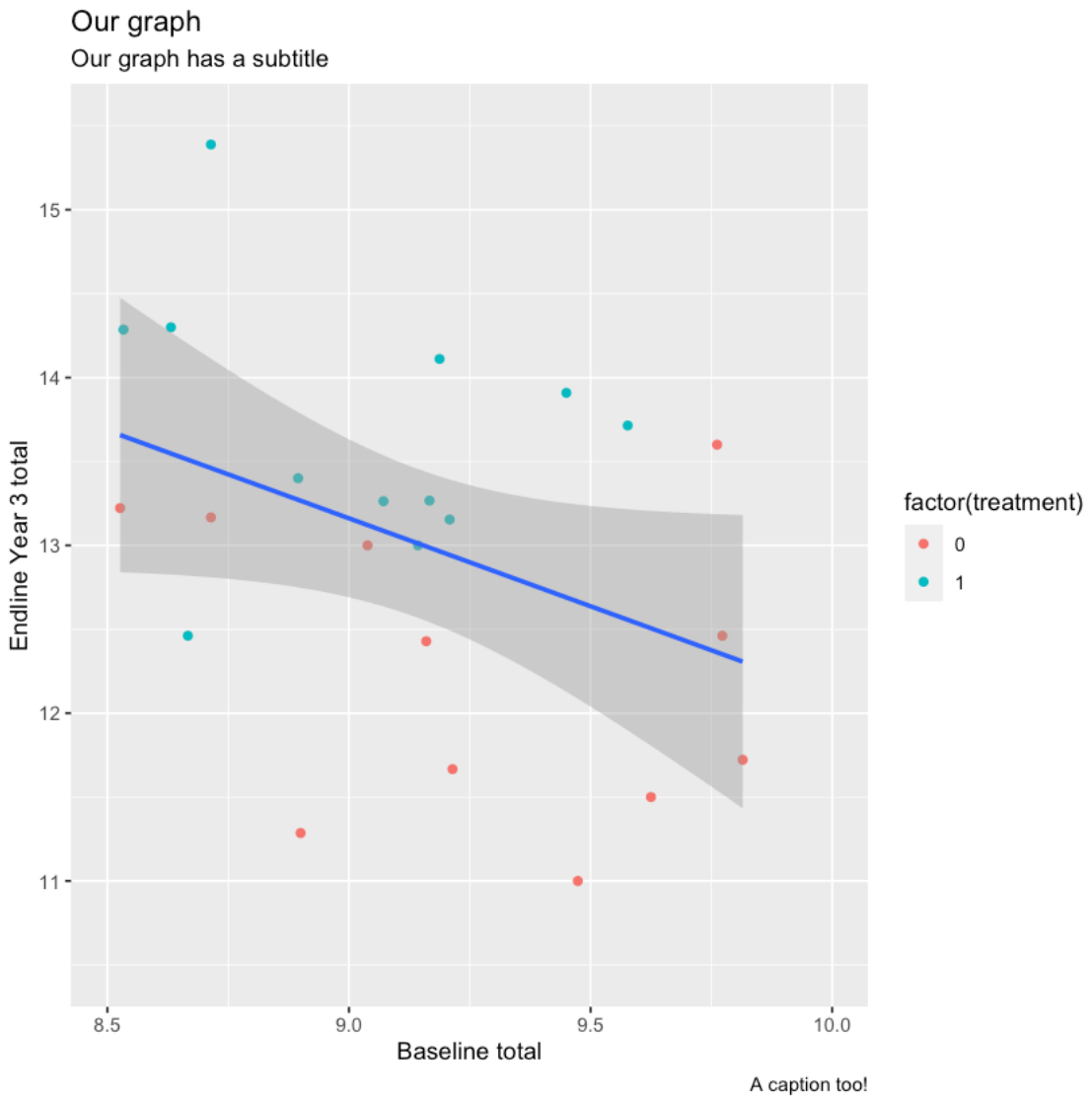
`coord_cartesian`

[50]:
```
plot2 +
    coord_cartesian(xlim = c(8.5, 10),
                    ylim = c(10.5, 15.5))
```



`scale_x_continuous` and `scale_y_continuous`

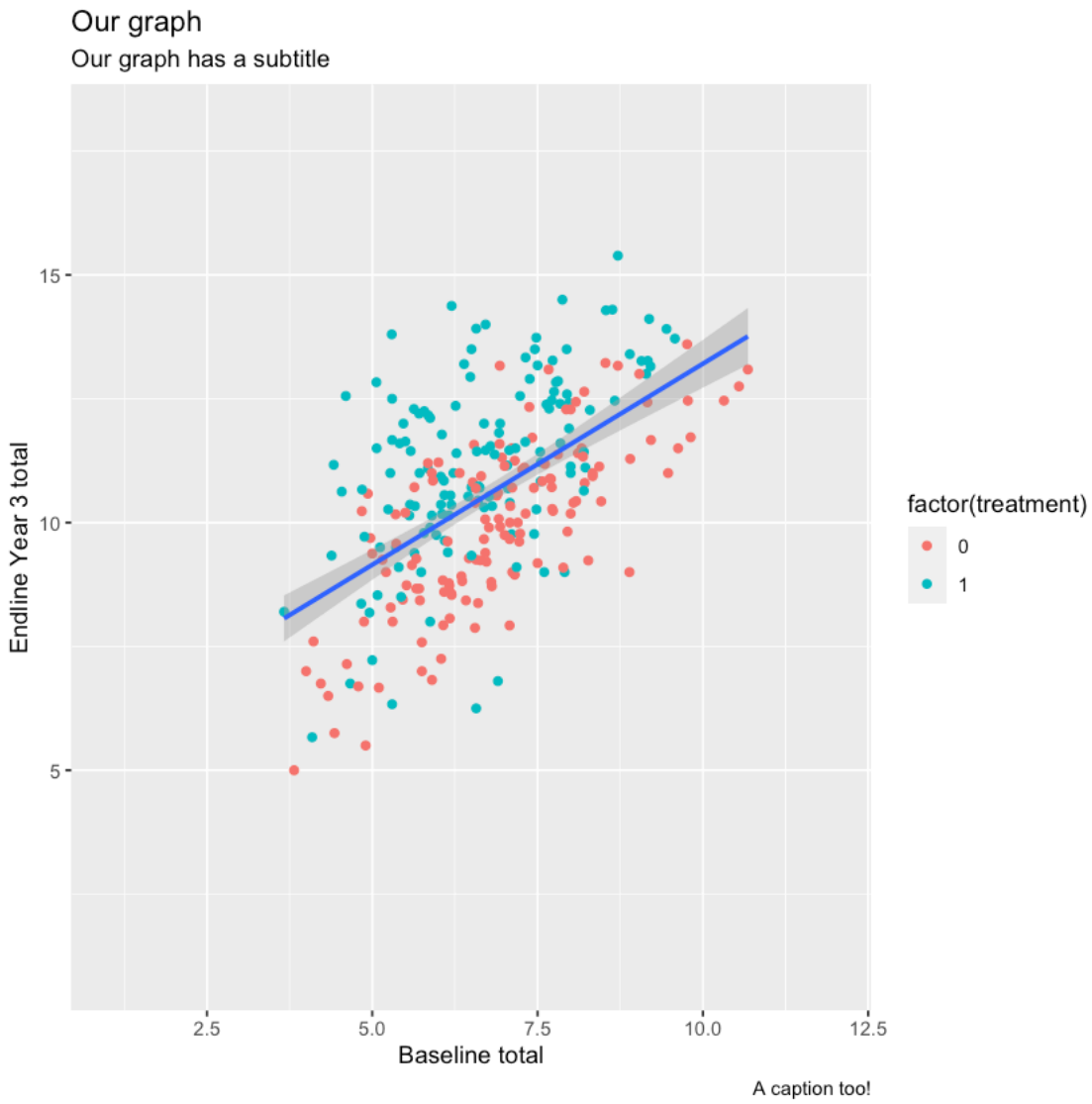
[51]:
```
plot2 +
    scale_x_continuous(limits = c(8.5, 10)) +
    scale_y_continuous(limits = c(10.5, 15.5))
```



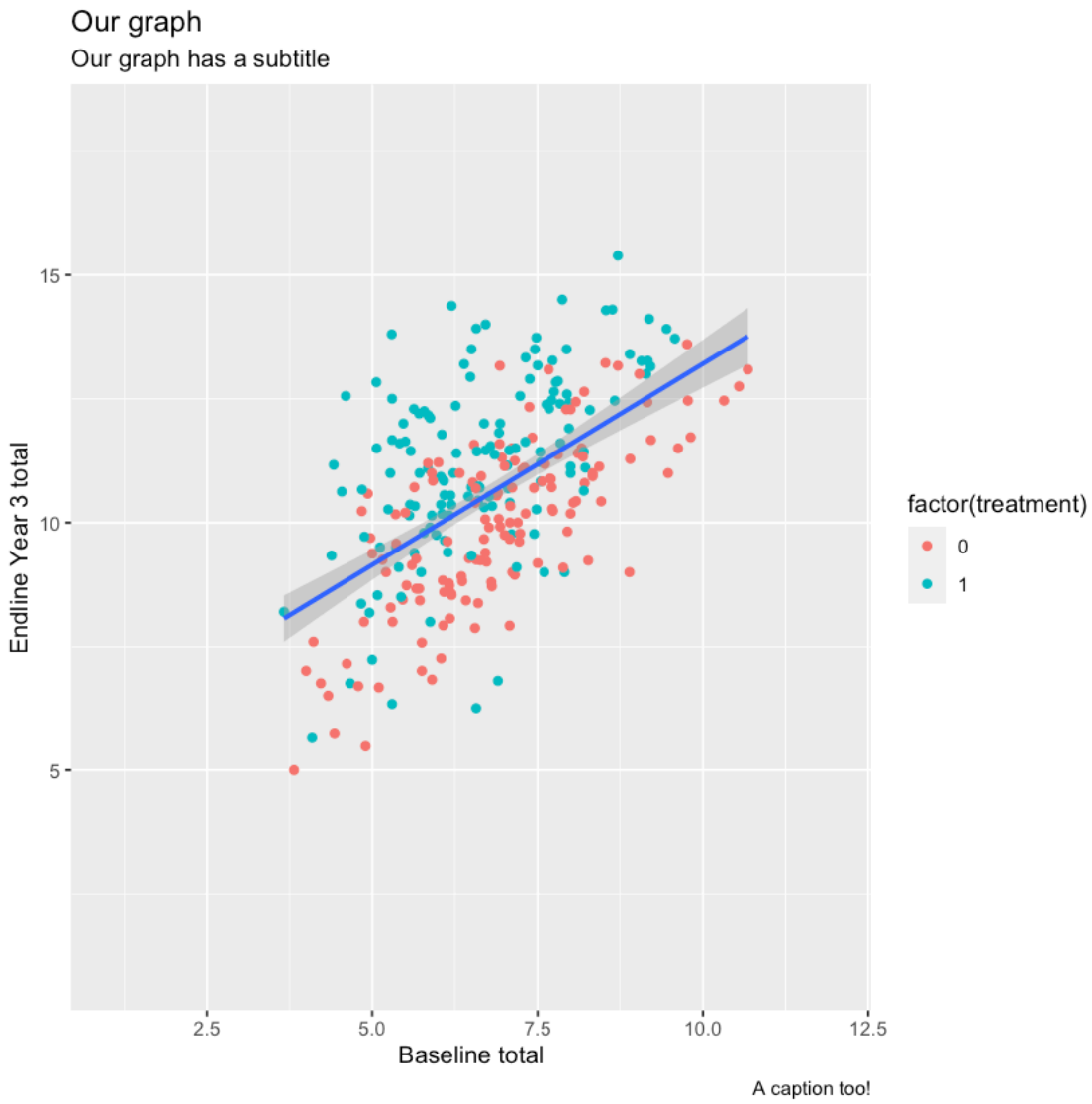
Two completely different graphs!
By using `coord_cartesian`, we are able to *zoom in* to the graph. However, if we use `scale_x_continuous` (or `scale_y_continuous`, we are *limiting* the data points which can be plotted. Consequently, the fitted line and its accompanying CI envelope also changes. So 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.

[53]:
```
plot2 +
    coord_cartesian(xlim = c(1, 12),
                    ylim = c(1, 18))
```



[54]:
```
plot2 +
    scale_x_continuous(limits = c(1, 12)) +
    scale_y_continuous(limits = c(1, 18))
```



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)
```

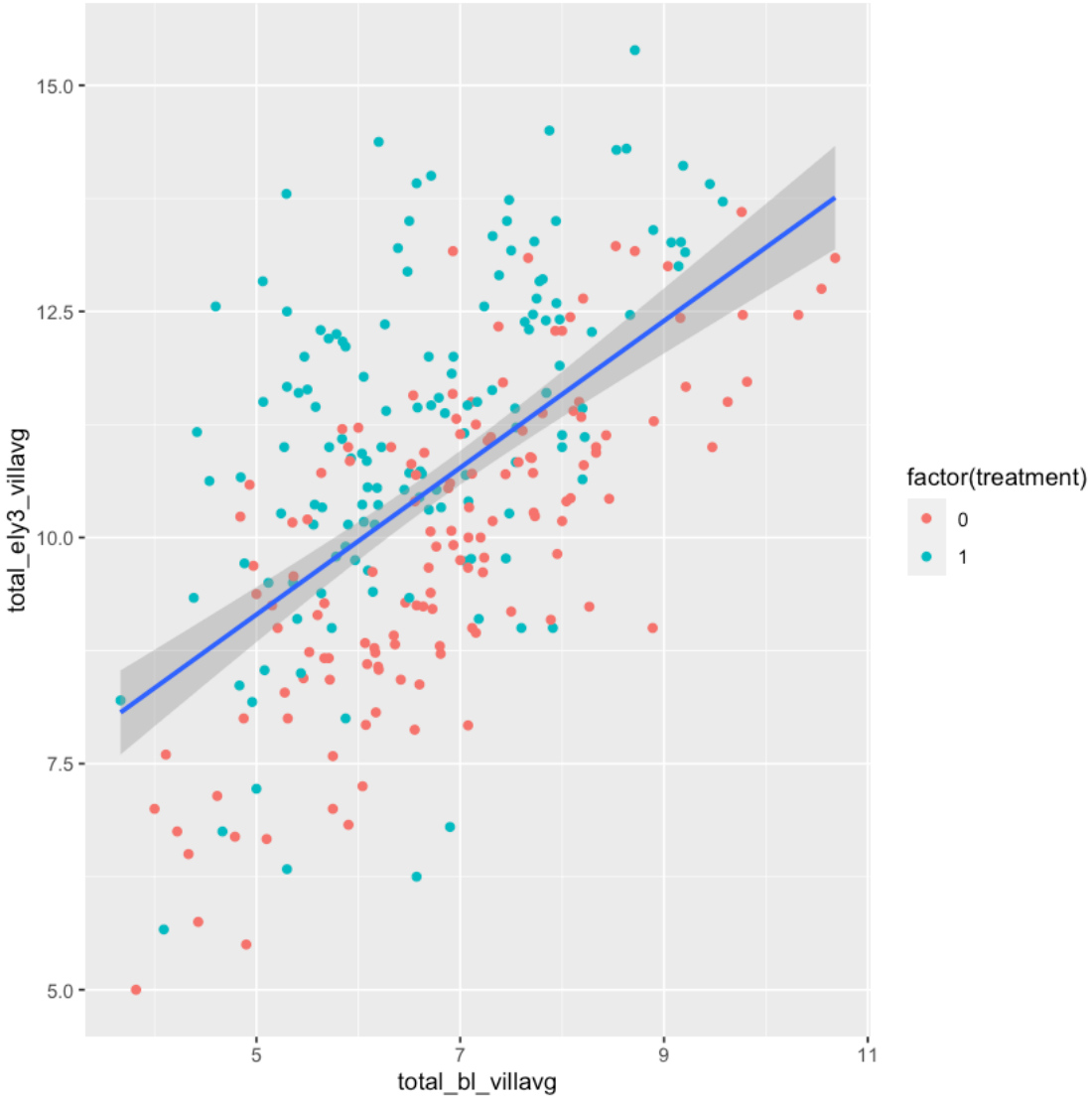
`ggplot` adds default scales equivalent to:

```
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()
```

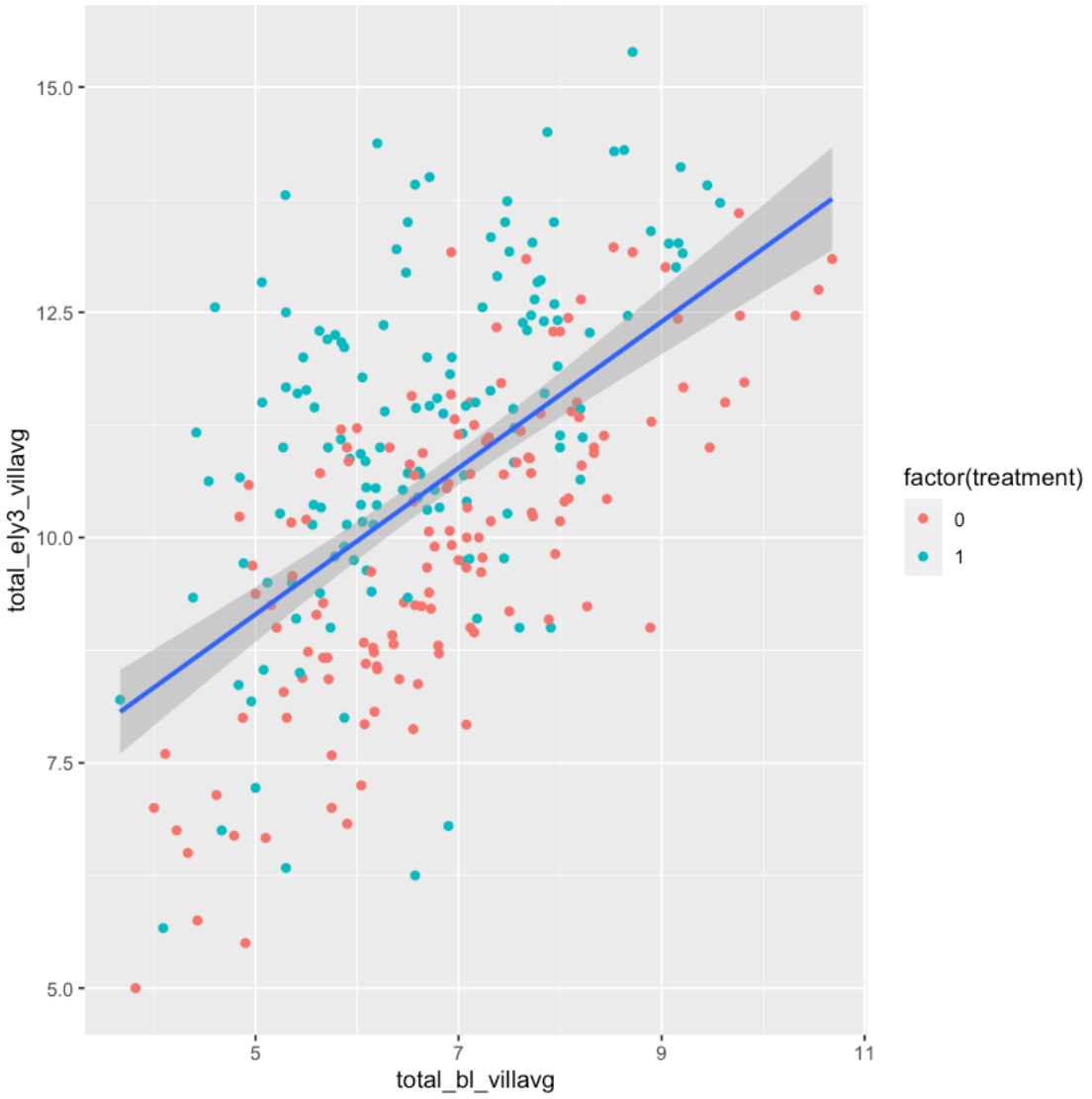
Let's execute and verify that they are indeed the same.

[55]:
```
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)
```



[56]:
```
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()
```



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:

[57]:
```
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()
```

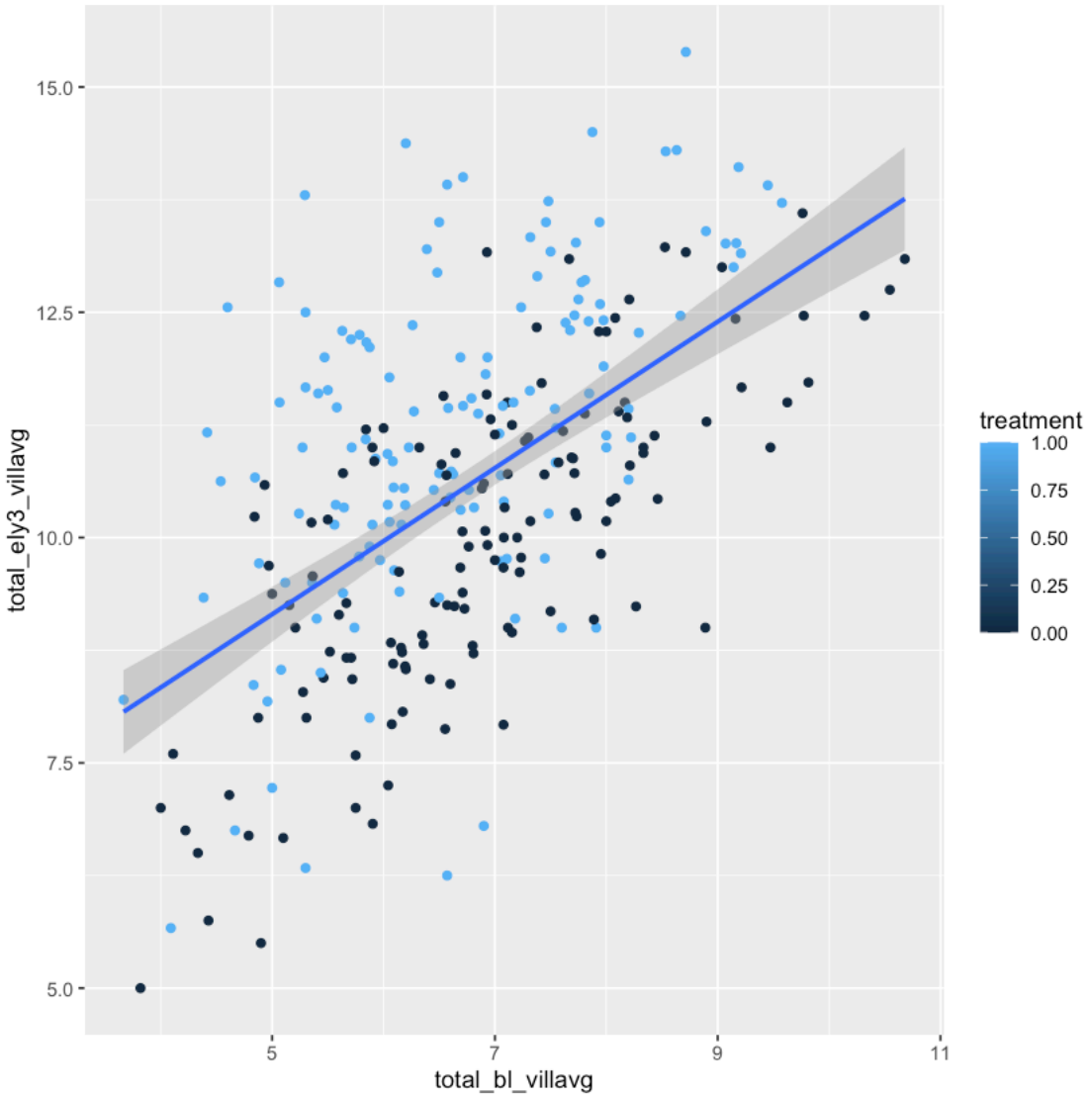
```
ERROR while rich displaying an object: Error: Continuous value supplied to
discrete scale

Traceback:
1. FUN(X[[i]], …)
2. 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)
3. tryCatchList(expr, classes, parentenv, handlers)
4. tryCatchOne(expr, names, parentenv, handlers[[1L]])
5. doTryCatch(return(expr), name, parentenv, handler)
6. 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)
7. repr::mime2repr[[mime]](obj)
8. repr_text.default(obj)
9. paste(capture.output(print(obj)), collapse = "\n")
10. capture.output(print(obj))
11. evalVis(expr)
12. withVisible(eval(expr, pf))
13. eval(expr, pf)
14. eval(expr, pf)
15. print(obj)
16. print.ggplot(obj)
17. ggplot_build(x)
18. ggplot_build.ggplot(x)
19. lapply(data, scales_train_df, scales = npscales)
20. FUN(X[[i]], …)
21. lapply(scales$scales, function(scale) scale$train_df(df = df))
22. FUN(X[[i]], …)
23. scale$train_df(df = df)
24. f(…, self = self)
25. self$train(df[[aesthetic]])
```

```
26. f(…, self = self)
27. self$range$train(x, drop = self$drop, na.rm = !self$na.translate)
28. f(…, self = self)
29. scales::train_discrete(x, self$range, drop = drop, na.rm = na.rm)
30. 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:

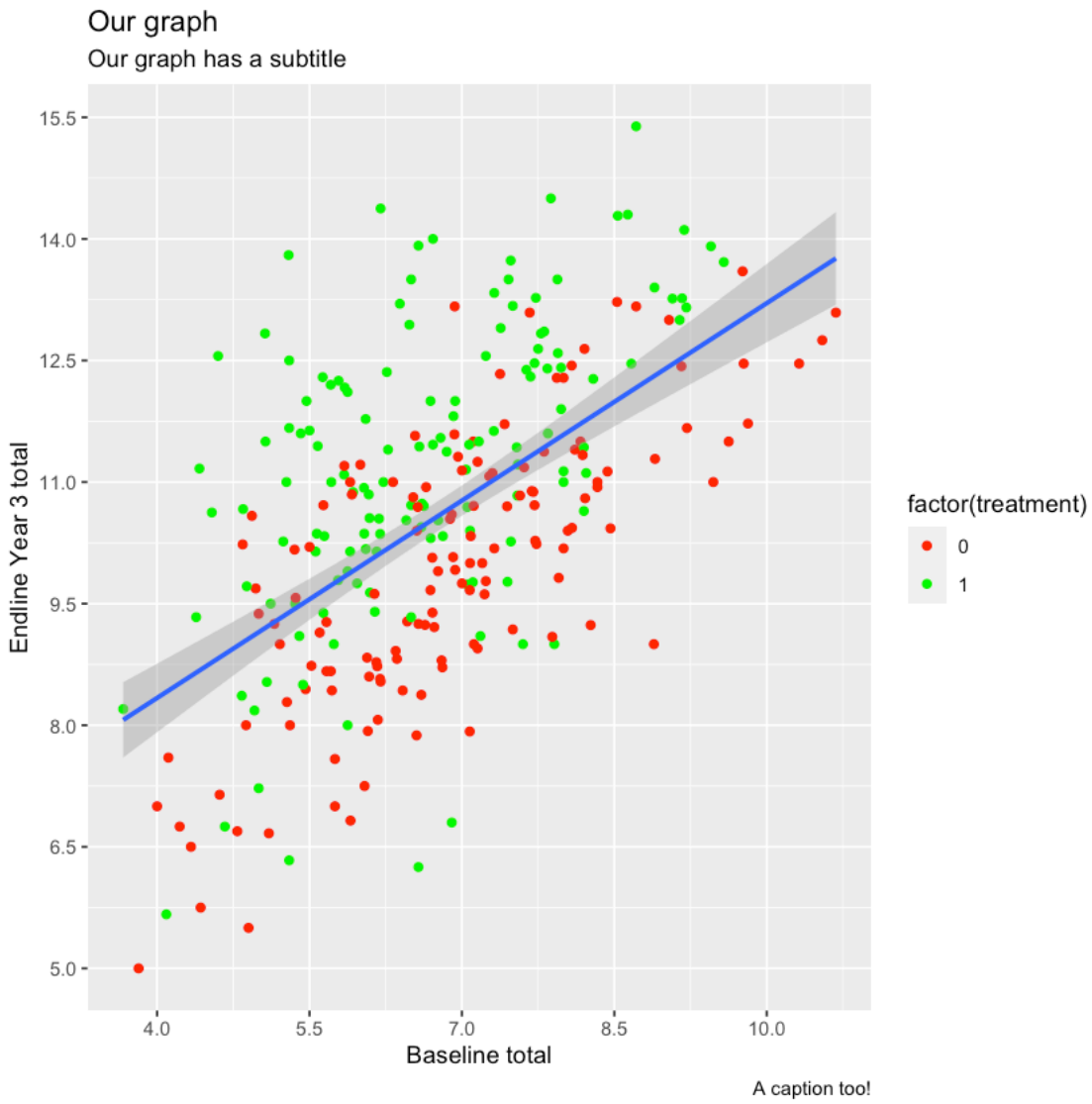
[63]:
```
ggplot(mydata, aes(x = total_bl_villavg, y = total_ely3_villavg)) +
            geom_point(aes(color = treatment), na.rm = TRUE) +
            geom_smooth(method = lm, formula = y ~ x, na.rm = TRUE) +
            scale_x_continuous() +
```

```
        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. `values` for changing the color of the legend levels

[64]:
```
plot3 +
    scale_color_discrete(type = c("red", "green"))
```


Our graph
Our graph has a subtitle
Endline Year 3 total
15.5
14.0
12.5
11.0
9.5
8.0
6.5
5.0
4.0
5.5
7.0
8.5
10.0
Baseline total
A caption too!
factor(treatment)
0
1

[ ]: