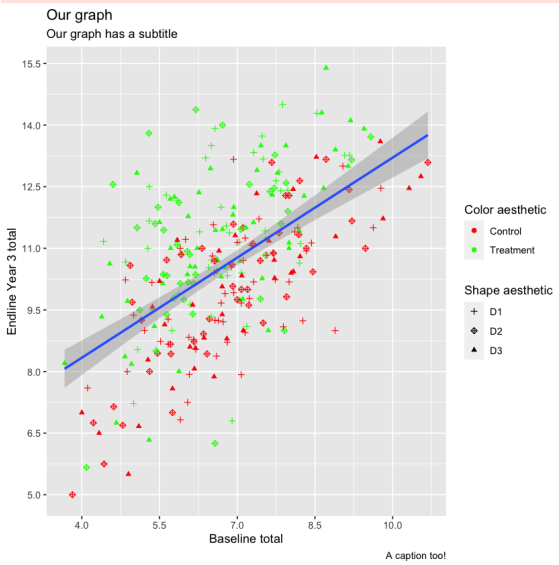
Data visualizations using R

Customizing `ggplot` graphs #2

In this lesson, we will discuss non-data visual changes to our graphs using the `theme()` layer.
Let's start by recreating the graph from the previous section.

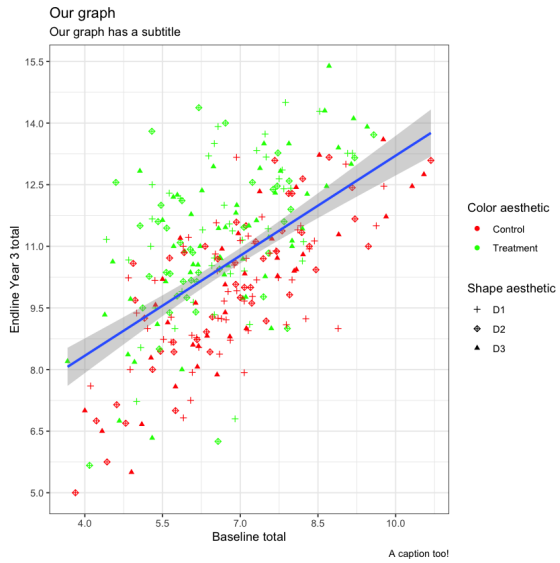
In [1]:
```
library("tidyverse")

mydata <- read_csv("https://raw.githubusercontent.com/arkadeep/R-Bootcamp-Data-Visualization/main/Assets/2_data/EG_DIB.cs
mydata$district <- factor(sample.int(3, nrow(mydata), replace = T))

labs <- c("1" = "Treatment", "0" = "Control")
colors <- c("0" = "red", "1" = "green")
labs_district <- c("1" = "D1", "2" = "D2", "3" = "D3")


plot1 <- 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"
       ) +
       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")
       ) +
       labs(
              title = "Our graph",
              subtitle = "Our graph has a subtitle",
              caption = "A caption too!",
              x = "Baseline total",
              y = "Endline Year 3 total"
       )

plot1
```

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──

✔ ggplot2 3.3.6     ✔ purrr   0.3.4
✔ tibble  3.1.7     ✔ dplyr   1.0.9
✔ 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”
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

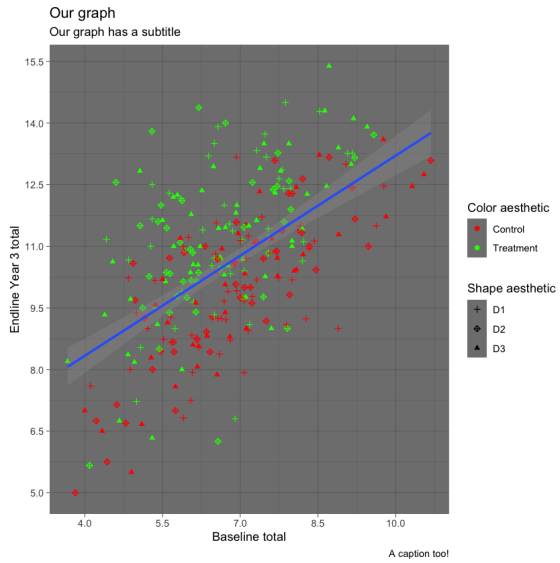


`ggplot` comes inbuilt with several default themes.

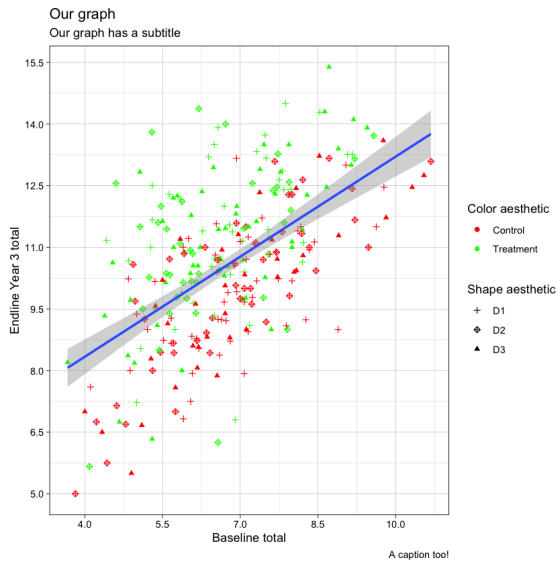
In [2]:
```
plot1 + theme_bw()
```



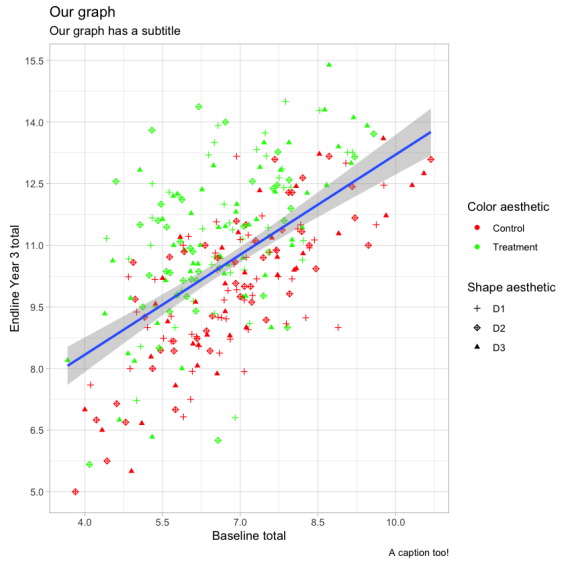
In [3]:
```
plot1 + theme_dark()
```



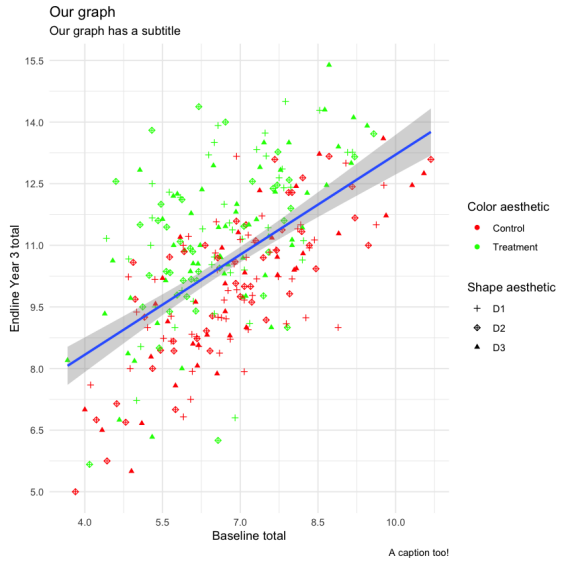
In [4]:
```
plot1 + theme_linedraw()
```



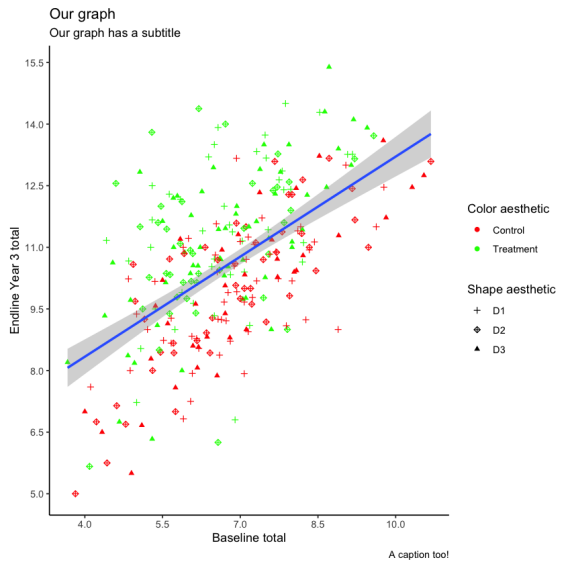
In [5]:
```
plot1 + theme_light()
```



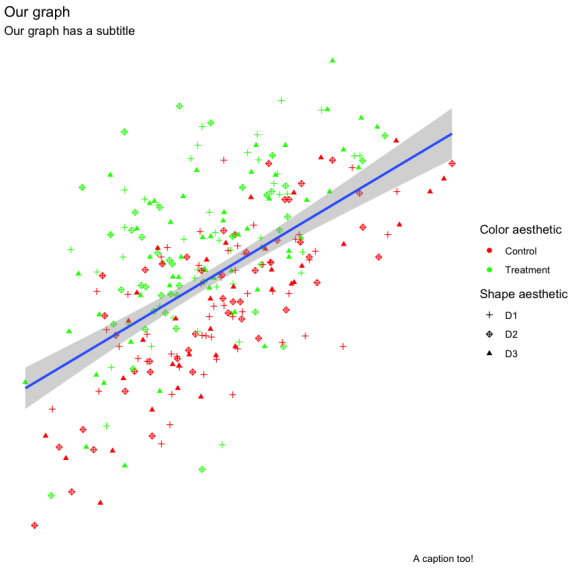
In [6]:
```
plot1 + theme_minimal()
```



In [7]:
```
plot1 + theme_classic()
```



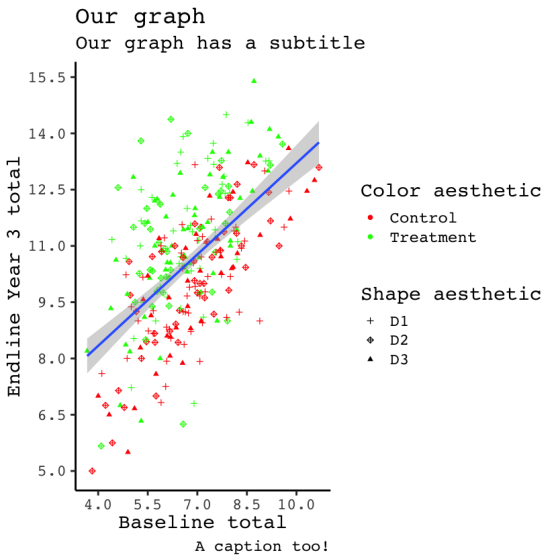
In [8]:
```
plot1 + theme_void()
```



Each `theme_xxx` function can take two arguments:

1. `base_size` : to change the font size
2. `base_family` : to change the font face

In [9]:
```
plot1 +
    theme_classic(
        base_size = 18,
        base_family = "mono"
    )
```



While the inbuilt themes can be a quick way to change the way your graph looks, you will probably want to customize it further. This can be done using the `theme()` layer. It is usually convenient to build your the modifications on top of an inbuilt theme. `theme_classic()` serves as a good starting point as it removes most elements which would otherwise be considered chart junk. Let's discuss some of the most commonly used parameters available to us.

Super parameters

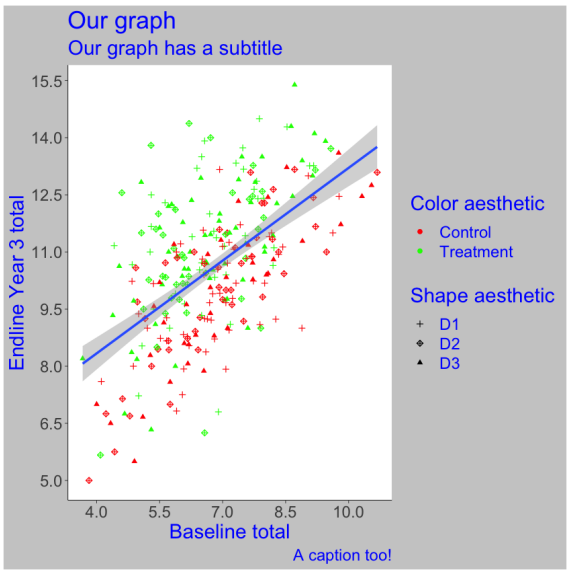
There are 3 "super" parameters. If you specify arguments for these, those would be carried over to every visual element they are mapped to.

- `line` : Used to change *all* line elements. This includes the axis lines, axis ticks and the panel grids. Done through the `element_line()` function.
- `rect` : Used to change *all* border and background elements. This includes the panel background, plot background, etc. Done through the `element_rect()` function.
- `text` : Used to change *all* text elements. This includes plot titles, axis titles, legend key text, etc. Done through the `element_text()` function.

This page details the full capabilities of the `element_xxx` functions.

In [10]:
```
plot1 +
    theme_classic() +
    theme(
        text = element_text(
            size = 18,
            color = "blue"
```

```
        ),
        rect = element_rect(
            fill = "grey80",
            color = "green"
        ),
        line = element_line(
            size = 0.25,
            color = "red"
        )
    )
```



Let's go over each element:

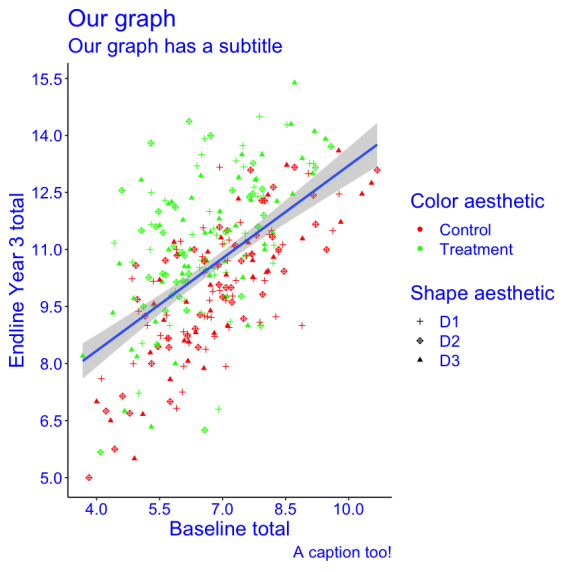
`text`

```
text = element_text(
            size = 18,
            color = "blue"
        )
```

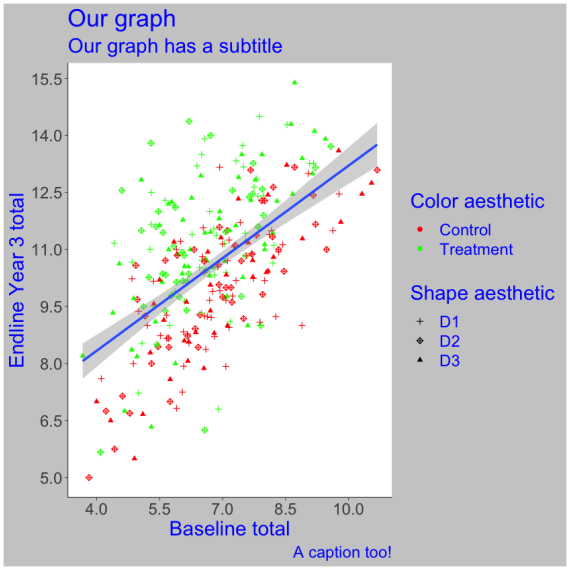
We are changing the size and color of every text element in the plot. But you might have noticed that not *everything* changes. The axis tick labels are still appearing in black. This is surprising, since the `axis.text` is supposed to inherit from the `text` parameter.

The reason why the text formatting does not get applied to the axis tick labels is because our `theme()` function has been applied on top of the the inbuilt `theme_classic()`. `theme_classic()` applies certain axis tick label formatting which overrides what we specify in the `text` parameter. Therefore, if we want to change the axis label text, we will need to pass arguments to the `axis.text` parameter.

In [11]:
```
plot1 +
    theme_classic() +
    theme(
        text = element_text(
            size = 18,
            color = "blue"
        ),
        axis.text = element_text(
            color = "blue"
        )
    )
```



In [12]:
```
plot1 +
    theme_classic() +
    theme(
        text = element_text(
            size = 18,
            color = "blue"
        ),
        rect = element_rect(
            fill = "grey80",
            color = "green"
        ),
        line = element_line(
            size = 0.25,
            color = "red"
        )
    )
```

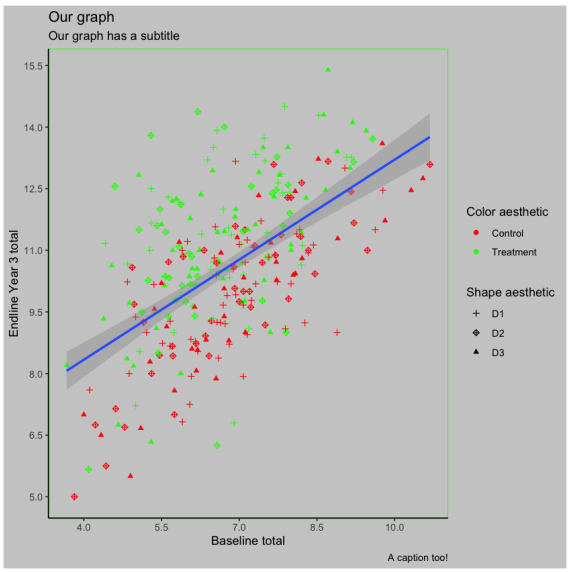


`rect`

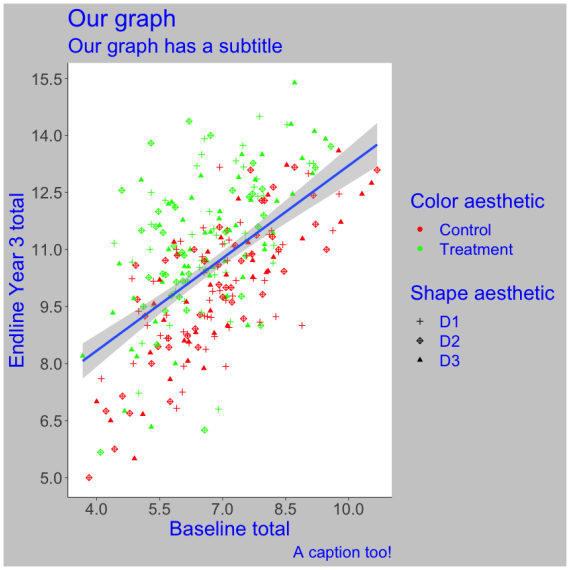
```
rect = element_rect(
            fill = "grey80",
            color = "green"
        )
```

We are changing the fill to a shade of grey and the outline to green for each element covered under `rect`. However as with `text`, you will notice that some elements, such as the panel background is unchanged. This is again because of modifications applied by `theme_classic()`. Specifying these through `panel.background` will do the trick.

In [13]:
```
plot1 +
    theme_classic() +
    theme(
        rect = element_rect(
            fill = "grey80",
            color = "green"
        ),
        panel.background = element_rect(
            fill = "grey80",
            color = "green"
        )
    )
```



In [14]:
```
plot1 +
    theme_classic() +
    theme(
        text = element_text(
            size = 18,
            color = "blue"
        ),
        rect = element_rect(
            fill = "grey80",
            color = "green"
        ),
        line = element_line(
            size = 0.25,
            color = "red"
        )
    )
```

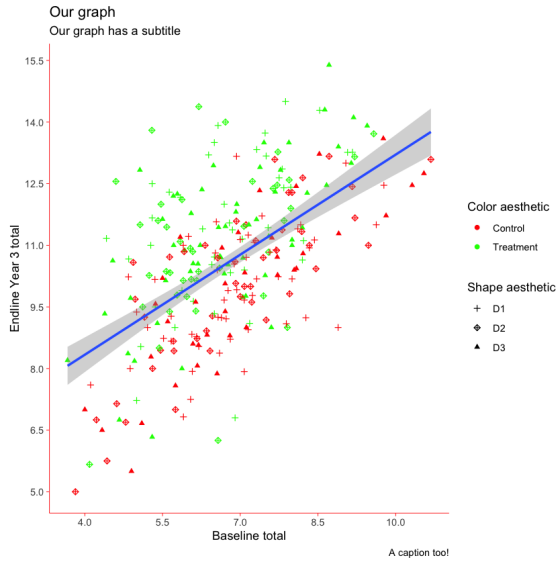


`line`

```
line = element_line(
            size = 0.25,
            color = "red"
        )
```

We are changing the size, which in this case is reflected as the width of the axis line and axis ticks and setting the color as red. But again, due to modifications set by `theme.classic()`, the color is not applied. To set the axis line and axis ticks as red, we will have to specify them through `axis.line` and `axis.ticks`.

In [15]:
```
plot1 +
    theme_classic() +
    theme(
        line = element_line(
            size = 0.25,
            color = "red"
        ),
        axis.line = element_line(
            color = "red"
        ),
        axis.ticks = element_line(
            color = "red"
        )
    )
```



While these super parameters are a handy way to change the look of our graph very quickly, they have some limitations:

- Inbuilt themes will override certain modifications.
  - This can overcome by modifying the theme, will be discussed later.
- No control over specific elements. In most use cases, you will probably want to specify exactly which elements you are interested in modifying.

Axis parameters

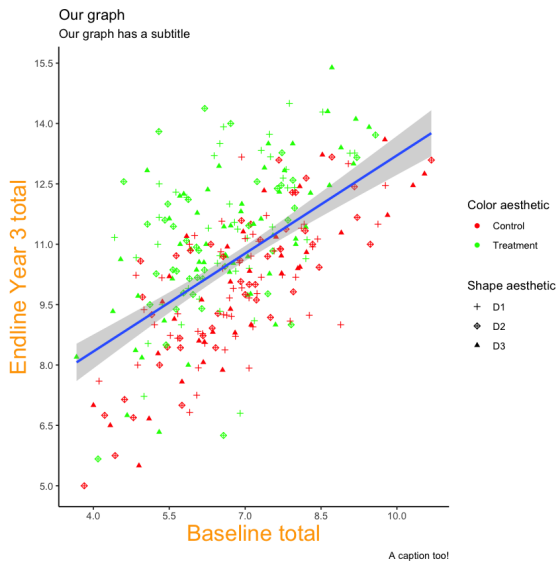
Axis title

Modifications are done through `element_text`. Some parameters include:

- `axis.title`
- `axis.title.x`: inherits from `axis.title`
- `axis.title.y`: inherits from `axis.title`

If you want to apply the same visual modifications to both axes, you can use `axis.title` only as `axis.title.x` and `axis.title.y` will automatically inherit those properties.

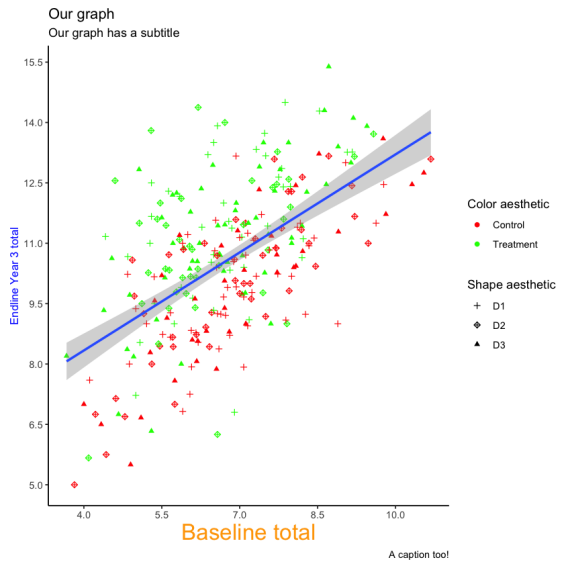
In [16]:
```
plot1 +
    theme_classic() +
    theme(
        axis.title = element_text(
            size = 20,
            color = "orange"
        )
    )
```



However if the axes are to have distinct visual modifications, you will need to use both `axis.title.x` and `axis.title.y`.

In [17]:
```
plot1 +
    theme_classic() +
    theme(
        axis.title.x = element_text(
            size = 20,
```

```
            color = "orange"
        ),
        axis.title.y = element_text(
            size = 10,
            color = "blue"
        )
    )
```



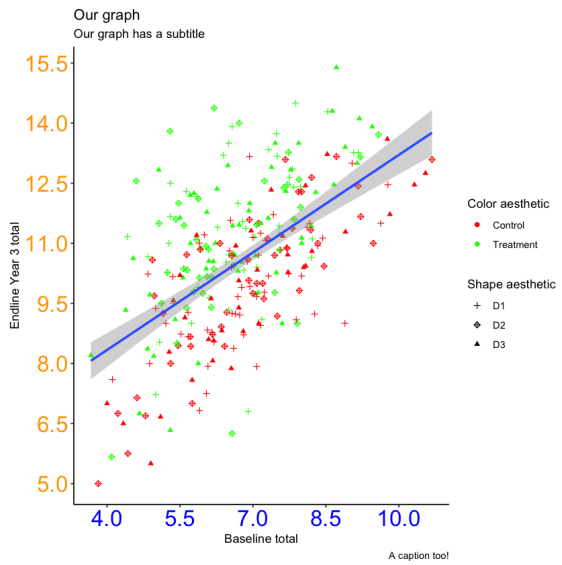
Axis text

Modifications are done through `element_text`. Some parameters include:

- `axis.text`
- `axis.text.x`: inherits from `axis.text`
- `axis.text.y`: inherits from `axis.text`

Inheritance works the same way as `axis.title`.

In [18]:
```
plot1 +
    theme_classic() +
    theme(
        axis.text = element_text(
            size = 20
        ),
        axis.text.x = element_text(
            color = "blue"
        ),
        axis.text.y = element_text(
            color = "orange"
        )
    )
```



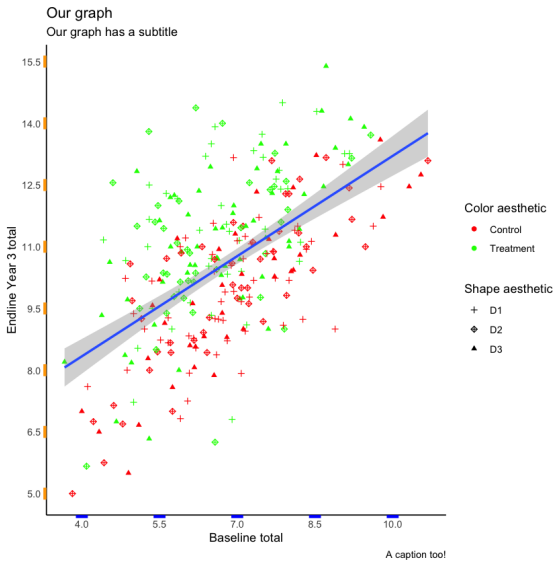
Axis tick marks

Modifications are done through `element_line`. Some parameters include:

- `axis.ticks`
- `axis.ticks.x`: inherits from `axis.ticks`
- `axis.ticks.y`: inherits from `axis.ticks`

Inheritance works the same way as `axis.title`.

In [19]:
```
plot1 +
    theme_classic() +
    theme(
        axis.ticks = element_line(
            size = 5
        ),
        axis.ticks.x = element_line(
            color = "blue"
        ),
        axis.ticks.y = element_line(
            color = "orange"
        )
    )
```



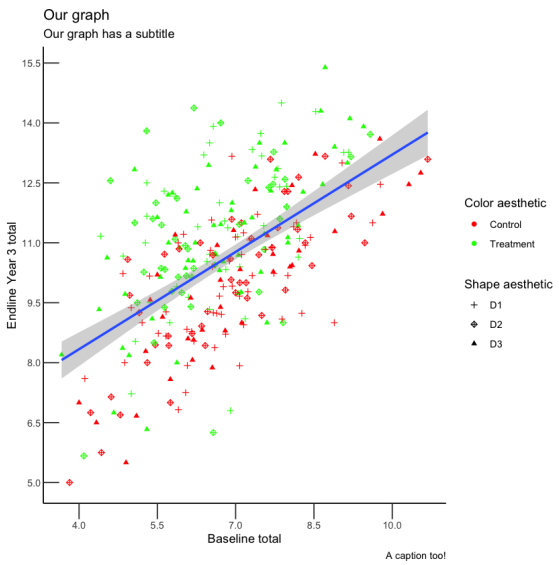
Axis tick length

Modifications done through the `unit` function. Some parameters include:

- `axis.ticks.length`
- `axis.ticks.length.x` : inherits from `axis.ticks.length`
- `axis.ticks.length.y` : inherits from `axis.ticks.length`

Inheritance works the same way as `axis.title`.

In [20]:
```
plot1 +
    theme_classic() +
    theme(
        axis.ticks.length.x = unit(.5, "cm"),
        axis.ticks.length.y = unit(-0.5, "cm")
    )
```

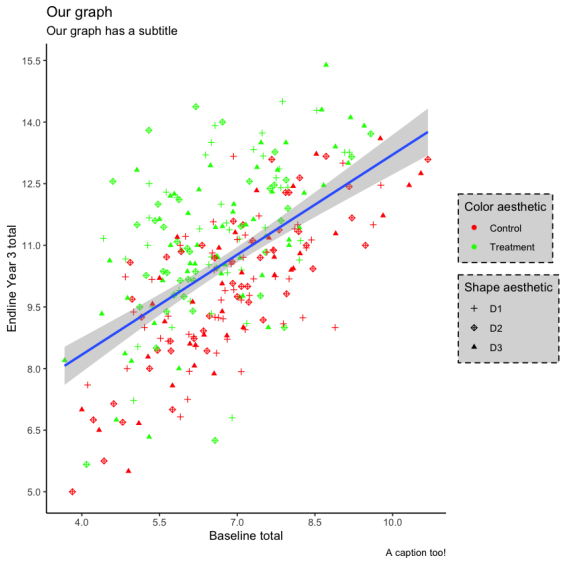


Legend parameters

Legend background

`legend.background` with `element_rect`

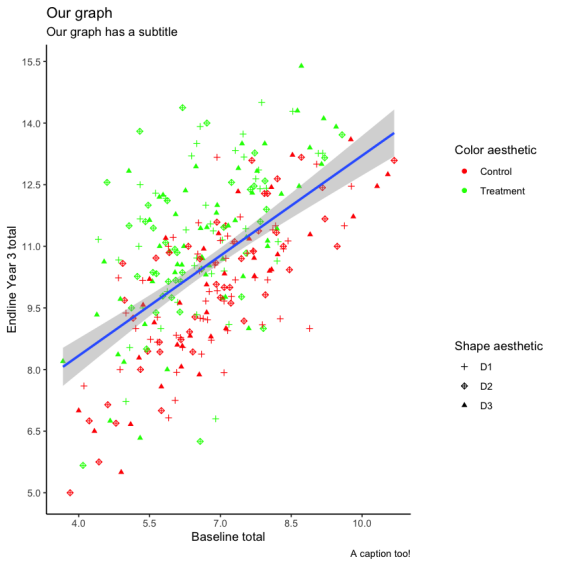
In [21]:
```
plot1 +
    theme_classic() +
    theme(
        legend.background = element_rect(
            fill = "grey85",
            color = "black",
            linetype = 8
        )
    )
```



Legend margin

`legend.margin` with `margin()`

In [22]:
```
plot1 +
    theme_classic() +
    theme(
        legend.margin = margin(
            t = 2,
            r = 0.5,
            b = 2,
            l = 1,
            unit = "cm"
        )
    )
```



Legend spacing

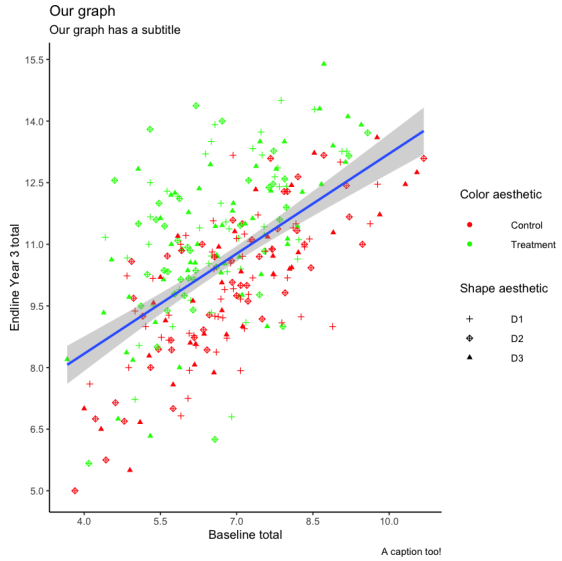
Modifications done through the `unit` function. Parameters include:

- `legend.spacing`
- `legend.spacing.x` : inherits from `legend.spacing`
- `legend.spacing.y` : inherits from `legend.spacing`

Inheritance works the same way as `axis.title` .

In [23]:
```
plot1 +
    theme_classic() +
```

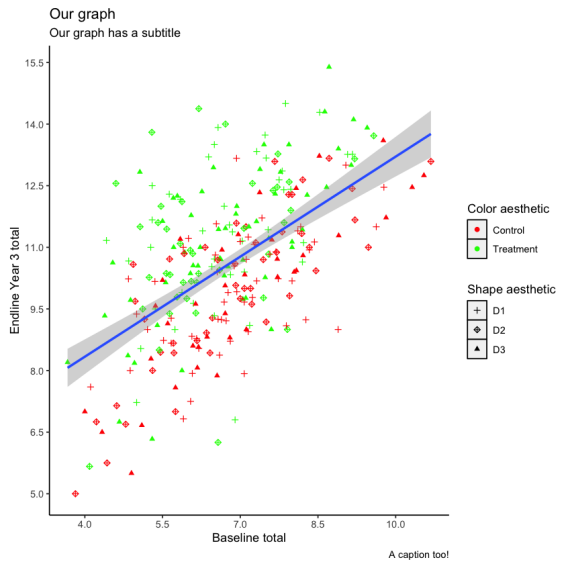
```
    theme(
        legend.spacing.x = unit(1, "cm"),
        legend.spacing.y = unit(0.5, "cm")
    )
```



Legend key

Background underneath the legend keys. `legend.key` with `element_rect`.

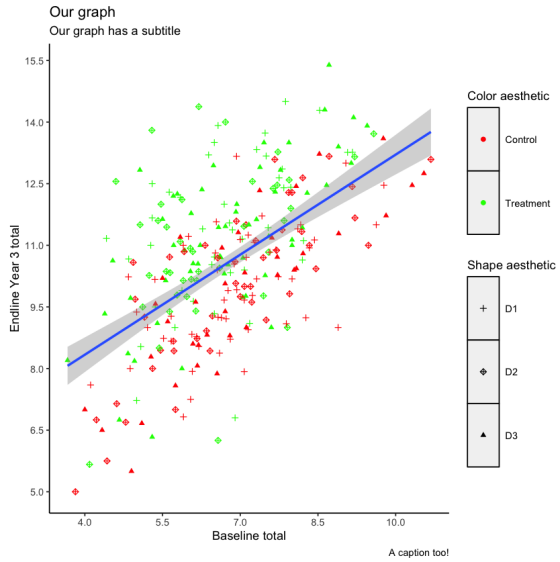
In [24]:
```
plot1 +
    theme_classic() +
    theme(
        legend.key = element_rect(
            fill = "grey95",
            color = "black"
        )
    )
```



Legend key size, height and width

Use with `unit`. `legend.key.height` and `legend.key.width` inherit from `legend.key.size` or you can specify them separately.

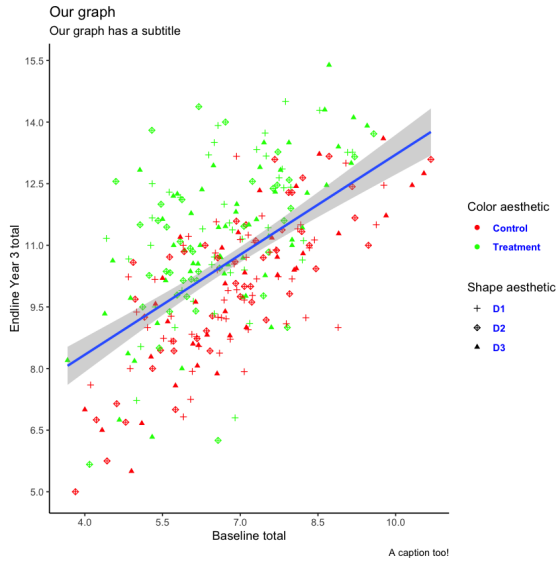
In [25]:
```
plot1 +
    theme_classic() +
    theme(
        legend.key = element_rect(
            fill = "grey95",
            color = "black"
        ),
        legend.key.height = unit(2, "cm"),
        legend.key.width = unit(1, "cm")
    )
```



Legend text

Refers to the legend labels. Use `legend.text` with `element_text`.

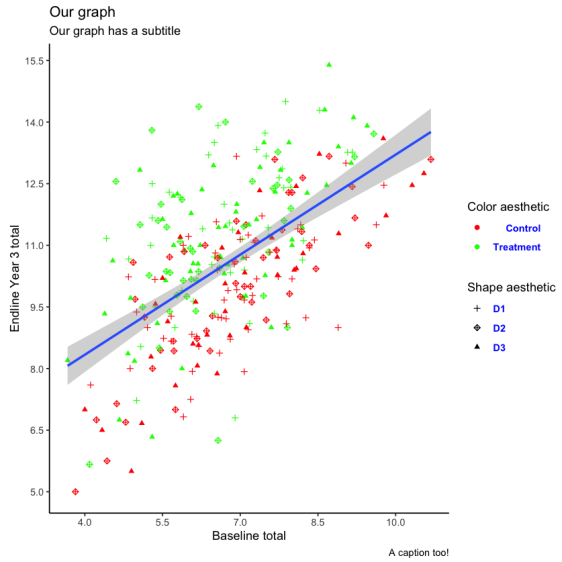
In [26]:
```
plot1 +
    theme_classic() +
    theme(
        legend.text = element_text(
            face = "bold",
            color = "blue"
        )
    )
```



Legend text alignment

Use a number between 0 (left) and 1 (right) with `legend.text.align`.

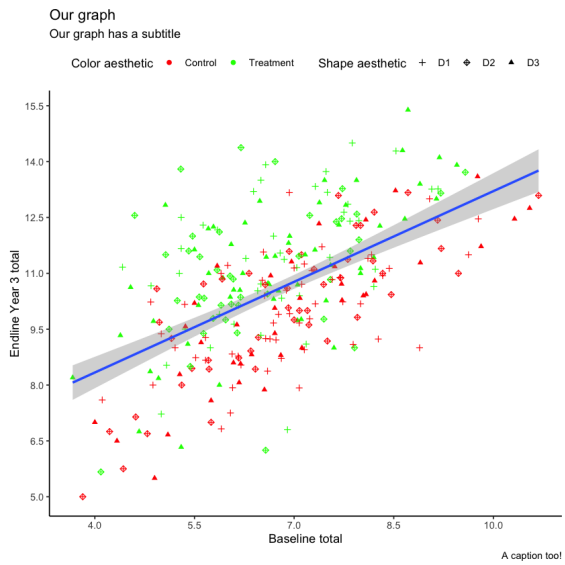
In [27]:
```
plot1 +
    theme_classic() +
    theme(
        legend.text = element_text(
            face = "bold",
            color = "blue"
        ),
        legend.text.align = 1
    )
```



Legend position

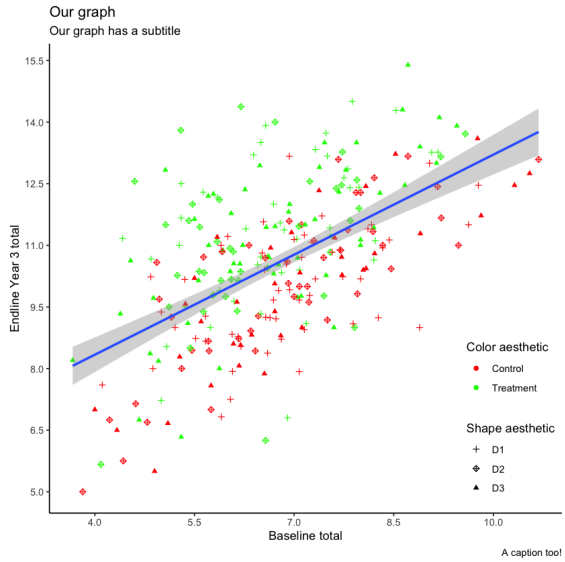
Changes the position of the legend. Can be used with text ("top", "right", "bottom", "left") or a numeric vector containing 2 elements.

In [28]:
```
plot1 +
    theme_classic() +
    theme(
        legend.position = "top"
    )
```



Use the numeric vector to place the legend inside the plot panel.

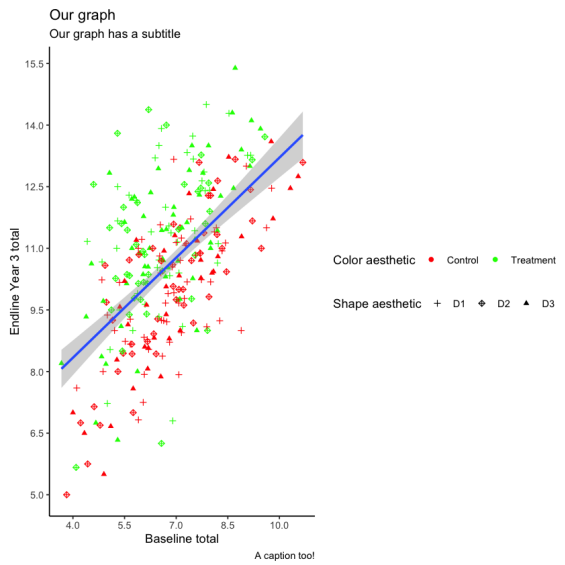
In [29]:
```
plot1 +
    theme_classic() +
    theme(
        legend.position = c(0.90, .2)
    )
```



Legend direction

To switch between horizontal and vertical layouts.

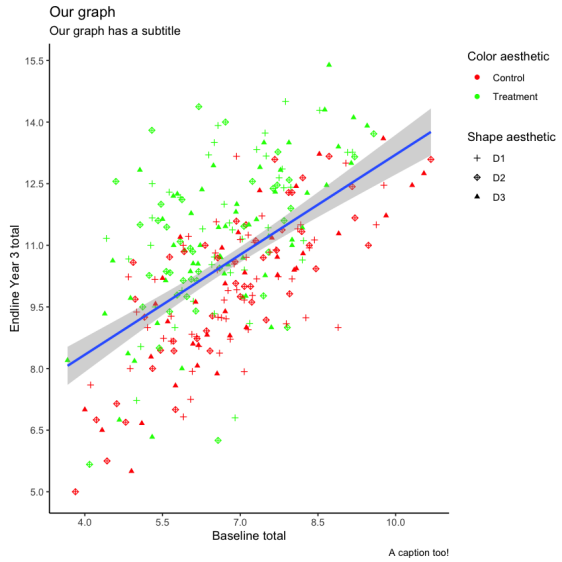
In [30]:
```
plot1 +
    theme_classic() +
    theme(
        legend.direction = "horizontal"
    )
```



Legend justification

Changes the justification of the legend.

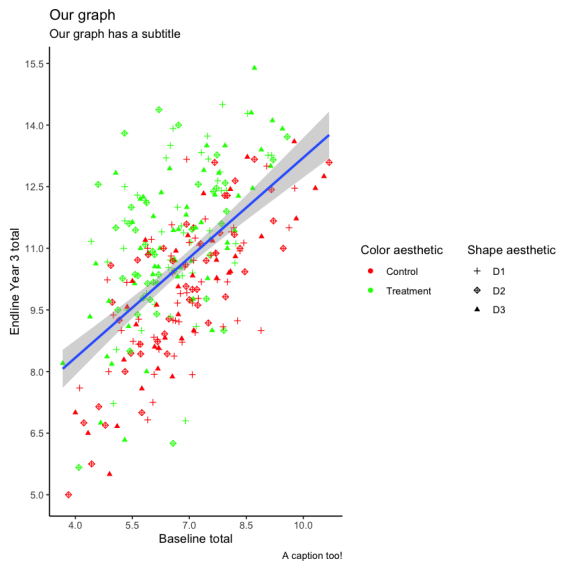
In [31]:
```
plot1 +
    theme_classic() +
    theme(
        legend.justification = "top"
    )
```



Legend box

Arrangement of multiple legend boxes, horizontal or vertical. Note the difference between this and `legend.direction`.

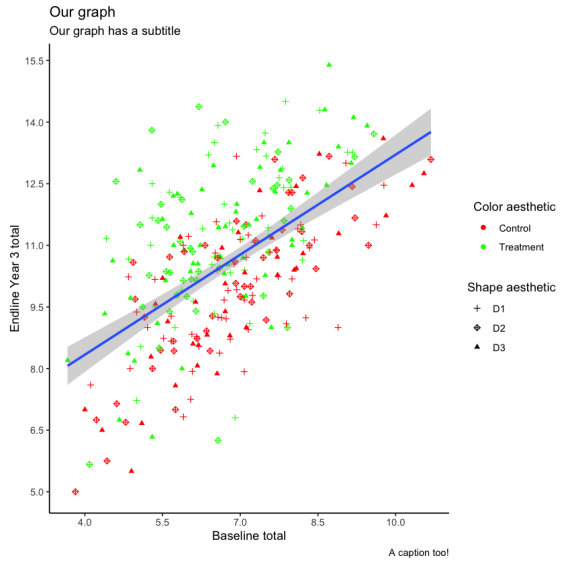
In [32]:
```
plot1 +
    theme_classic() +
    theme(
        legend.box = "horizontal"
    )
```



Legend box justification

To change the justification of each legend within the overall legend bounding box. Used when there are multiple legends.

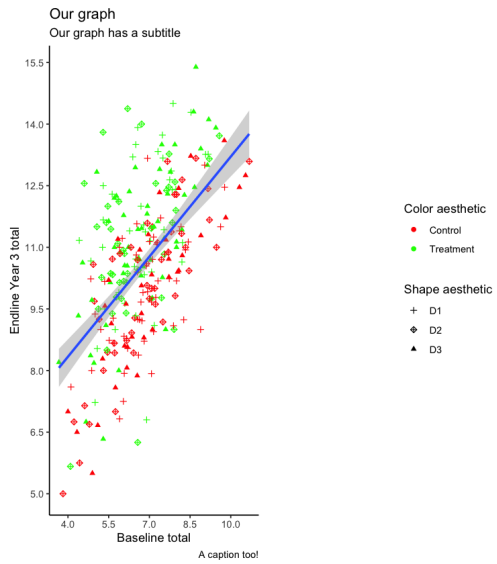
In [33]:
```
plot1 +
    theme_classic() +
    theme(
        legend.box.just = "right"
    )
```



Legend box margin

Use `margin()` to set the margin around the overall legend area.

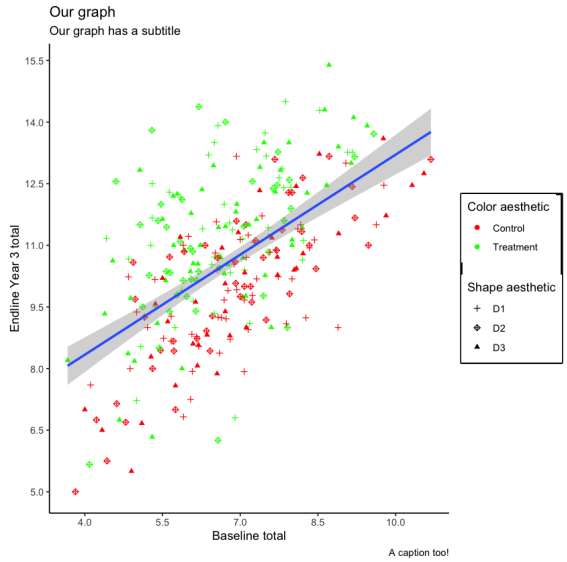
In [34]:
```
plot1 +
    theme_classic() +
    theme(
        legend.box.margin = margin(
            r = 2,
            l = 4,
            unit = "cm"
        )
    )
```



Legend box background

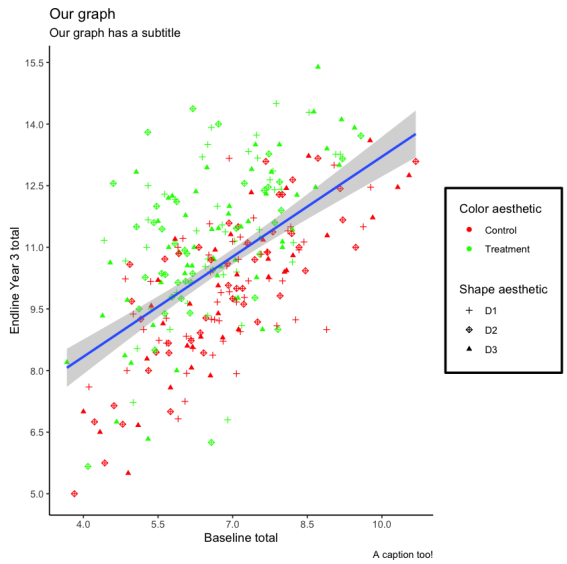
Use `legend.box.background` with the `element_rect()` function to set the background of the legend.

In [35]:
```
plot1 +
    theme_classic() +
    theme(
        legend.box.background = element_rect(
            color = "black",
            size = 1
        )
    )
```



The output does not look very good. Use `legend.box.margin` to increase the margins so that the boundary is clearly visible.

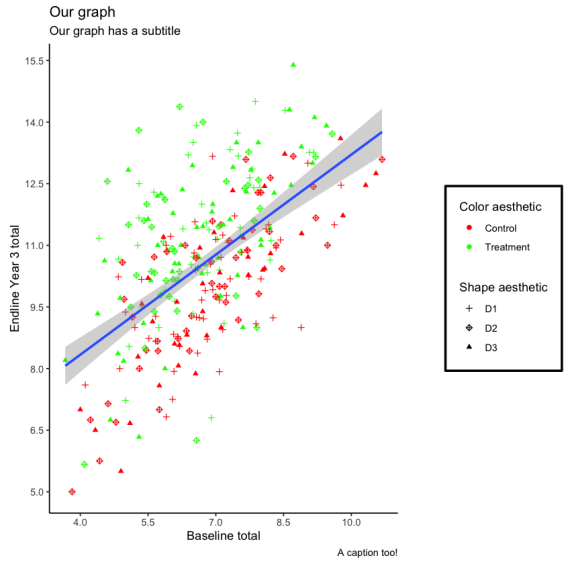
In [36]:
```
plot1 +
    theme_classic() +
    theme(
        legend.box.background = element_rect(
            color = "black",
            size = 1
        ),
        legend.box.margin = margin(
            t = 0.25,
            r = 0.25,
            b = 0.25,
            l = 0.25,
            unit = "cm"
        )
    )
```



Legend box spacing

Use the `unit()` function along with `legend.box.spacing` to change the space between the legend box and plotting area.

In [37]:
```
plot1 +
    theme_classic() +
    theme(
        legend.box.background = element_rect(
            color = "black",
            size = 1
        ),
        legend.box.margin = margin(
            t = 0.25,
            r = 0.25,
            b = 0.25,
            l = 0.25,
            unit = "cm"
        ),
        legend.box.spacing = unit(1.5, "cm")
    )
```

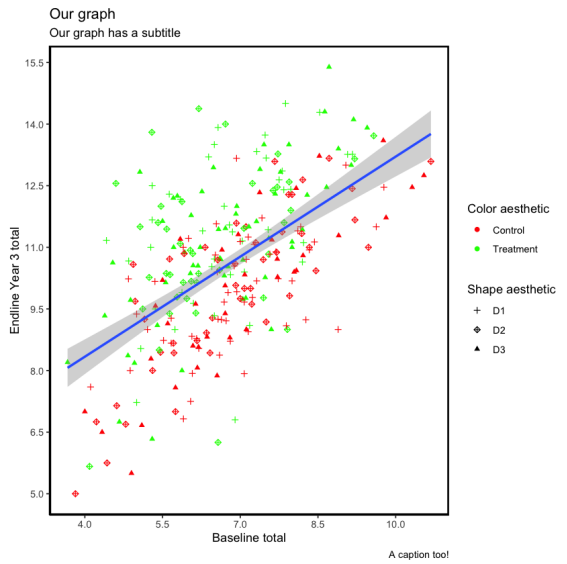


Panel parameters

Panel background

Use `panel.background` with `element_rect` to change the background of the panel. Note that this is drawn underneath the plot that you have created. It is advisable to use the `fill` parameter with `NA` as argument so that the visualization is not distorted.

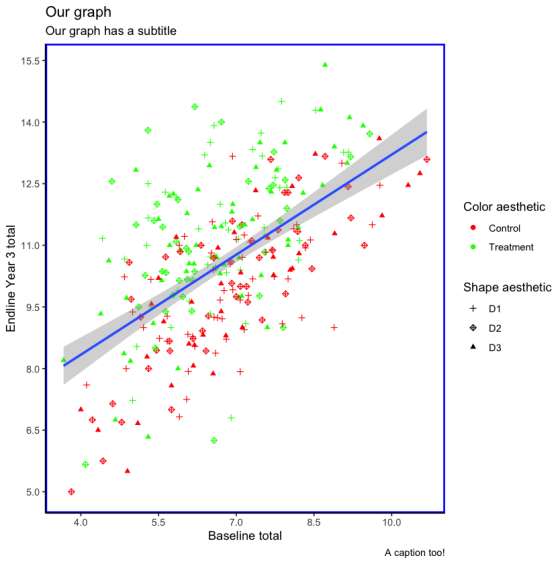
In [38]:
```
plot1 +
    theme_classic() +
    theme(
        panel.background = element_rect(
            color = "black",
            size = 1.5,
            fill = NA
        )
    )
```



Panel border

`panel.border` behaves quite similarly to `panel.background`, with the difference between that the former draws on top of the plot. As you can see below, the axis line have been drawn over. Again, it is advisable to specify `fill` as `NA` so that the data is not distorted.

In [39]:
```
plot1 +
    theme_classic() +
    theme(
        panel.border = element_rect(
            color = "blue",
            size = 1.5,
            fill = "NA"
        )
    )
```



Panel grids

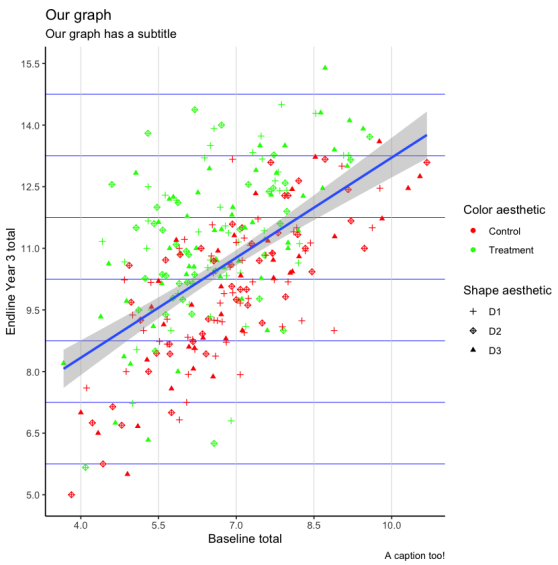
To change the grid lines on the plot. Use with `element_line`.

Available parameters:

- `panel.grid`
- `panel.grid.major` : inherits from `panel.grid`
- `panel.grid.minor` : inherits from `panel.grid`
- `panel.grid.major.x` : inherits from `panel.grid.major`
- `panel.grid.major.y` : inherits from `panel.grid.major`
- `panel.grid.minor.x` : inherits from `panel.grid.minor`
- `panel.grid.minor.y` : inherits from `panel.grid.minor`

Inheritance works as expected.

In [40]:
```
plot1 +
    theme_classic() +
    theme(
        panel.grid.major.x = element_line(
            color = "grey85",
            size = 0.25
        ),
        panel.grid.minor.y = element_line(
            color = "blue",
            size = 0.25
        )
    )
```



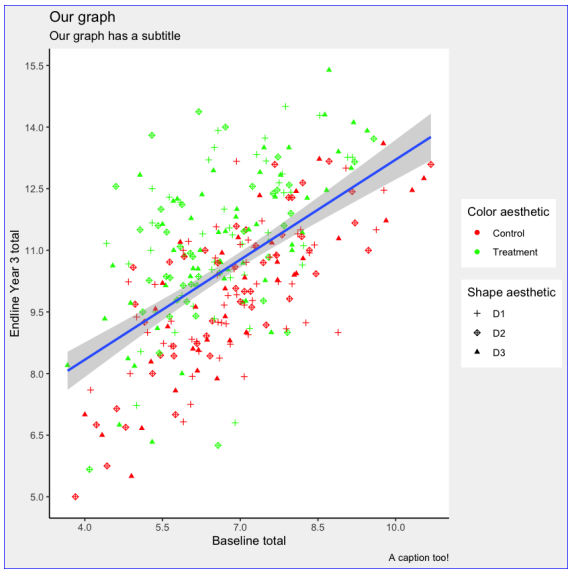
Plot parameters

Plot background

Use `plot.background` with `element_rect` to change the background of the entire plot.

In [41]:
```
plot1 +
```

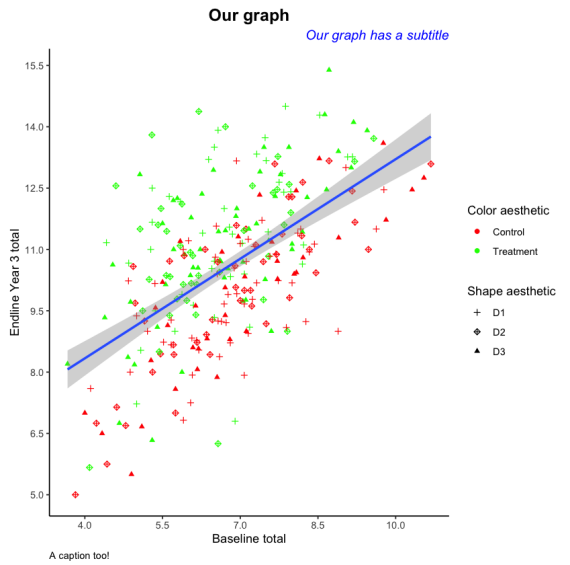
```
    theme_classic() +
    theme(
        plot.background = element_rect(
            color = "blue",
            fill = "grey95"
        )
    )
```



Title, subtitle and caption

Use `plot.title`, `plot.subtitle` and `plot.caption` to change the appearance of the title, subtitle and caption respectively. Changes done through `element_text`.

In [42]:
```
plot1 +
    theme_classic() +
    theme(
        plot.title = element_text(
            size = 15,
            face = "bold",
            hjust = 0.5
        ),
        plot.subtitle = element_text(
            size = 12,
            face = "italic",
            hjust = 1,
            color = "blue"
        ),
        plot.caption = element_text(
            vjust = 1,
            hjust = 0
        )
    )
```

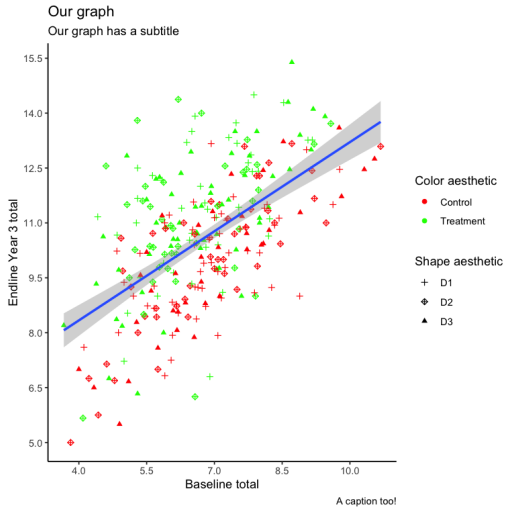


Plot margin

Plot margins can be changed to add space around the plot. Use the `unit()` function.

In [43]:
```
plot1 +
    theme_classic() +
    theme(
```

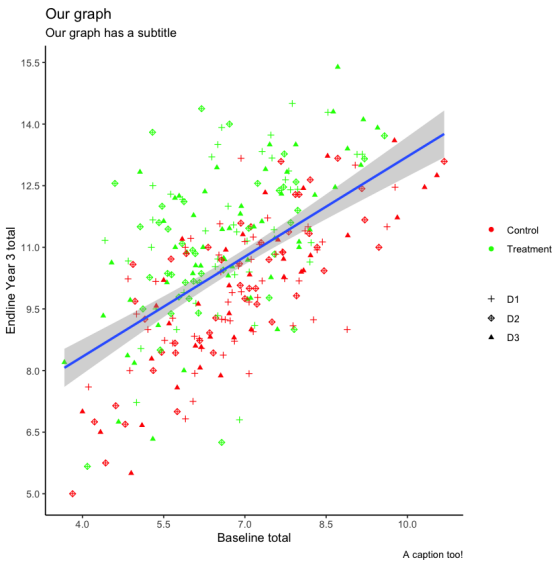
```
        plot.margin = unit(
            c(1,1,1,1),
            "cm"
        )
    )
```



Removing an element

Finally, if you have added an element but would like to remove it, you can do so from within the `theme` function by setting the relevant parament to `element_blank()`. Suppose we want to remove the legends titles.

In [44]:
```
plot1 +
    theme_classic() +
    theme(
        legend.title = element_blank()
    )
```



Final graph

We will now recreate the graph according to IDinsight formatting. If you are interested in using the Inter font, you will have add it to R using the `showtext` package as it is not available by default. Since Inter is a google font, we can add it easily using the `font_add_google()` function.

In [45]:
```
library('showtext')

font_add_google("Inter", "Inter")

showtext_auto()
```

```
Warning message:
“package ‘showtext’ was built under R version 4.0.5”
Loading required package: sysfonts

Warning message:
“package ‘sysfonts’ was built under R version 4.0.5”
Loading required package: showtextdb
```

Changing the colors vector to contain IDinsight brand colors and then creating the plot.

In [46]:
```
colors <- c("0" = "#A8BFEB", "1" = "#173363")

final_plot <- 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(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"
       ) +
       guides(
              color = guide_legend(title = "Group")
       ) +
       labs(
              title = "Correlation between Baseline and Endline 3 scores",
              x = "Baseline total",
              y = "Endline Year 3 total",
              caption = "Source: EG DIB Dataset"
       )
```

Adding the theme.

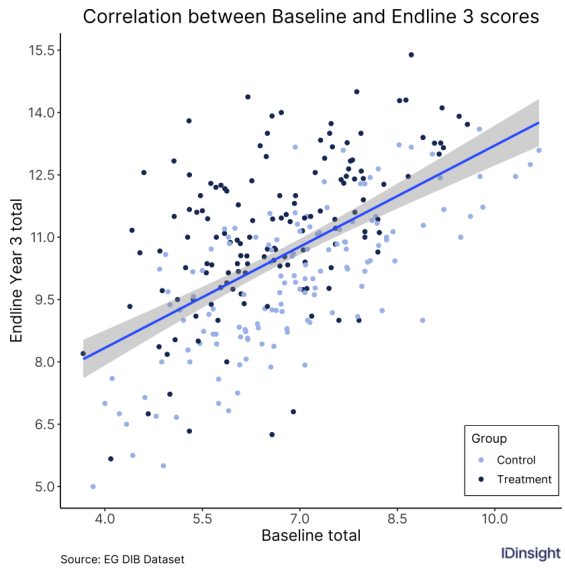
In [47]:
```
final_plot <- final_plot +
    theme_classic() +
    theme(
        text = element_text(
            family = "Inter"
        ),
        axis.text = element_text(
            size = 12
        ),
        axis.title = element_text(
            size = 14
        ),
        legend.box.background = element_rect(
            color = "black"
        ),
        legend.box.margin = margin(
              t = 0.5,
              r = 0.5,
              b = 0.5,
              l = 0.5,
              unit = "pt"
        ),
        legend.box = "horizontal",
        legend.position = c(0.9, 0.1),
        legend.text = element_text(
            size = 10
        ),
        plot.title = element_text(
            size = 17,
            hjust = 0.5
        ),
        plot.caption = element_text(
            size = 10,
            vjust = -1,
            hjust = 0
        )
    )
```

Adding the IDinsight logo

An optional final step would be to add the IDinsight logo to the bottom right corner of the plot. This relies on loading the logo as an image and then applying it to the plot. You will need the `magick` and `grid` packages.

In [48]:
```
library('magick')
library('grid')

final_plot

logo <- image_read("https://github.com/arkadeep/R-Bootcamp-Data-Visualization/blob/main/Assets/1_images/IDi-Logo-Master-R

grid.raster(logo, x = 0.935, y = 0.025, width = unit(1, 'inch'))
```

```
Linking to ImageMagick 6.9.12.3
Enabled features: cairo, fontconfig, freetype, heic, lcms, pango, raw, rsvg, webp
Disabled features: fftw, ghostscript, x11
```



Next steps

In the upcoming lessons, we will go over how to create some other common visualizations, such as Bar graphs, Line graphs, Histograms and Kernel density graphs.

Notes

You can check out the `ggthemes` package to greatly extend the number of themes available to use.