As noted elsewhere, sometimes [beauty matters](https://www.statworx.com/blog/make-rstudio-look-the-way-you-want-because-beauty-matters/). A plot that’s pleasing to the eye will be considered more  
gladly, and thus might be understood more thoroughly. Also, since we at [STATWORX](https://www.statworx.com/de/) oftentimes need to  
subsume and communicate our results, we have come to appreciate how a nice plot can upgrade any presentation.

So how make a plot look good? How make it accord with given style guidelines? In ggplot2 the display of all non-data components is controlled by the theme system. Other than in some other packages, the appearance of plots is edited after all the data-related elements of the plot have been determined. The theme system of ggplot2 allows the manipulation of titles, labels, legends, grid lines and backgrounds. There are various build-in themes available that already have an all-around consistent style, pertaining to any detail of a plot.

**Pre-defined themes**

There are two ways to apply bulid-in (or otherwise predefined) themes (e.g. theme\_grey, theme\_bw, theme\_linedraw, theme\_light, theme\_dark, theme\_minimal or theme\_classic).  
For one, they can be added as an additional layer to individual plots:

rm(list = ls())

library(gridExtra)

library(ggplot2)

# generating a fictional data set containing hours of sunshine and temperature

sun\_hours <- sample(seq(from = 1, to = 8, by = 0.1), size = 40, replace = TRUE)

noise <- sample(seq(from = 17, to = 24, by = 0.1), size = 40, replace = TRUE)

temperature <- sun\_hours + noise

df\_sun <- data.frame(sun\_hours, temperature)

# generate the plot base

base\_plot <- ggplot(df\_sun) +

geom\_point(aes(x = sun\_hours, y = temperature, color = temperature),

shape = 6, size = 5, stroke = 2) +

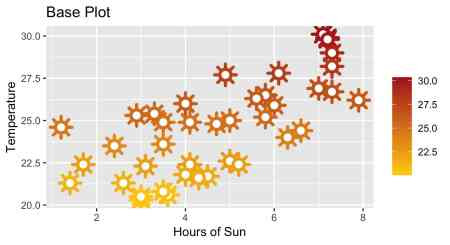
geom\_point(aes(x = sun\_hours, y = temperature, color = temperature),

shape = 21, size = 3.3, fill = "white", stroke = 2) +

labs(x = "Hours of Sun", y = "Temperature") +

scale\_color\_gradient(high = "firebrick", low = "#ffce00", name = " ") +

ggtitle("Base Plot")



# adding predefined themes

p1 <- base\_plot +

theme\_classic() +

ggtitle("Plot with theme\_classic()")

p2 <- base\_plot +

theme\_bw() +

ggtitle("Plot with theme\_bw()")

p3 <- base\_plot +

theme\_dark() +

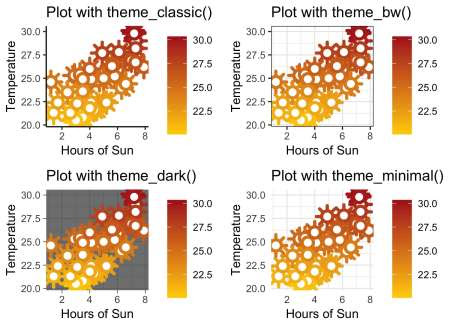
ggtitle("Plot with theme\_dark()")

p4 <- base\_plot +

theme\_light() +

ggtitle("Plot with theme\_light()")

gridExtra::grid.arrange(p1, p2, p3, p4)



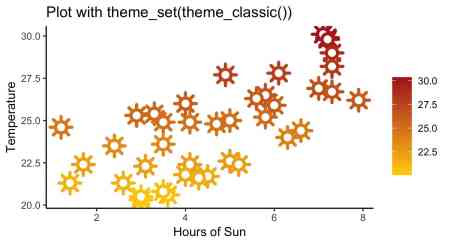
Alternatively, the default theme that’s automatically added to any plot, can be set or get with the functions theme\_set() or theme\_get().

# making the classic theme the default

theme\_set(theme\_classic())

base\_plot +

ggtitle("Plot with theme\_set(theme\_classic())")



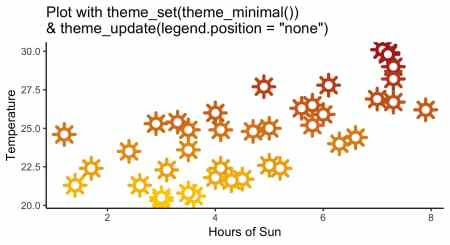
While predefined themes are very convenient, there’s always the option to (additionally) tweak the appearance of any non-data detail of a plot via the various arguments of theme(). This can be done for a specific plot, or the currently active default theme. The default theme can be updated or partly replaced via theme\_update and theme\_replace, respectively.

# changing the default theme

theme\_update(legend.position = "none")

base\_plot +

ggtitle("Plot with theme\_set(theme\_classic()) \n& theme\_update(legend.position = \"none\")")

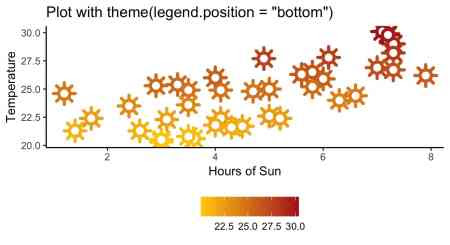


# changing the theme directly applied to the plot

base\_plot +

theme(legend.position = "bottom") +

ggtitle("Plot with theme(legend.position = \"bottom\")")



**Element functions**

There’s a wide range of arguments for theme(), in fact such a wide range, that not all arguments can be discussed here. Therefore, this blog post is far from exhaustive and only deals with the general principles of the theme system and only provides some illustrative examples for a few of all the available arguments. The appearance of many elements needs to be specified via one of the four element functions: element\_blank, element\_text, element\_line or element\_rect.

* How labels and titles are displayed, is controlled by the element\_text function. For example, we can make the title of the y axis bold and increase its size.
* Borders and backgrounds can be manipulated using element\_rect. For example, we can choose the color of the plot’s background.
* Lines can be defined via the element\_line function. For example, we can change the line types of the mayor and minor grid.
* Further, with element\_blank() it is possible to remove an object completely, without having any space dedicated to the plot element.

# using element\_text, element\_rect, element\_line, element\_blank

base\_plot +

theme(axis.title.y = element\_text(face = "bold", size = 16),

plot.background = element\_rect(fill = "#FED633"),

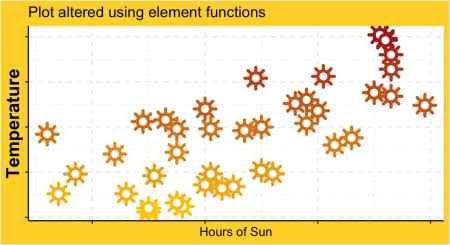
panel.grid.major = element\_line(linetype = "dashed"),

panel.grid.minor = element\_line(linetype = "dotted"),

axis.text.y = element\_blank(),

axis.text.x = element\_blank()) +

ggtitle("Plot altered using element functions")



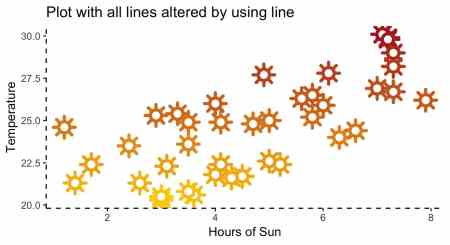
If we don’t want to change the display of some specific plot elements, but of all text, lines, titles or rectangular elements we can do so by specifying the arguments text, line, rect and title. Specifications passed to these arguments are inherited by all elements of the respective type. This inheritance principle also holds true for other 'parent' arguments. 'Parent' arguments oftentimes are easily identifiable, as their names are used as prefixes for all subordinate arguments.

# using overreaching arguments #1

base\_plot +

theme(line = element\_line(linetype = "dashed")) +

ggtitle("Plot with all lines altered by using line")

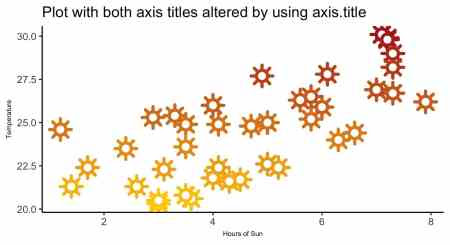


# using overreaching arguments #2

base\_plot +

theme(axis.title = element\_text(size = 6)) + # here axis.title is the parent

ggtitle("Plot with both axis titles altered by using axis.title")



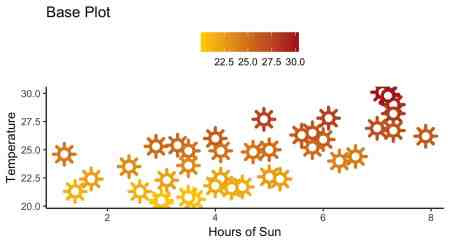
**Outlook**

Margins, spaces, sizes and orientations of elements are not specified with element functions but have their own sets of possible parameters. For example, the display of legends is controlled by such arguments and specific parameters.

# using parameters instead of element functions

base\_plot +

theme(legend.position = "top")



Since ggplot2 enables to manipulate the appearance of non-data elements of plots in great detail, there is a multitude of arguments. This blog post only tries to give a first impression of the many, many possibilities to design a plot. Some additional occupation with the topic, might be advisable, but any time invested in understanding how to style plots, surely is well spent. If you want read more on making pretty plots in ggplot2, check out my other posts on [coordinate stystems](https://www.statworx.com/blog/coordinate-systems-in-ggplot2-easily-overlooked-and-rather-underrated/) or [customizing date and time scales](https://www.statworx.com/blog/customizing-time-and-date-scales-in-ggplot2/).

**Referenzen**

* Wickham, H. (2009). *ggplot2: elegant graphics for data analysis*. Springer.