As noted elsewhere, sometimes beauty matters. A plot that’s pleasing to the eye will be considered more  
gladly, and thus might be understood more thoroughly. 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) +

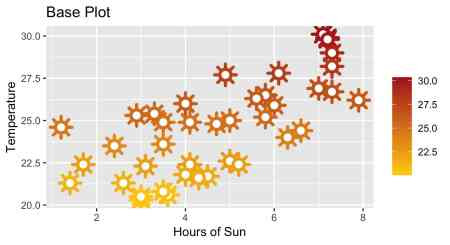
geom\_point(aes(x = sun\_hours, y = temperature, color = temperature),

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() +

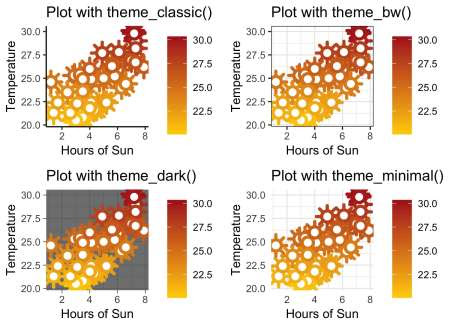
ggtitle("Plot with theme\_dark()")

p4 <- base\_plot +

theme\_light() +

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

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



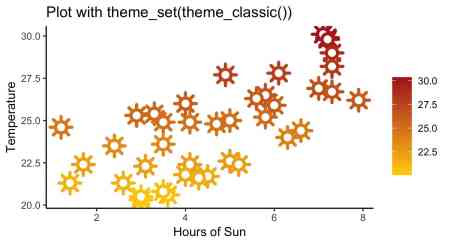
Alternatively, the default theme that’s automatically added to any plot, can be set or get with the functions theme\_set() or theme\_get().

# making the classic theme the default

theme\_set(theme\_classic())

base\_plot +

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



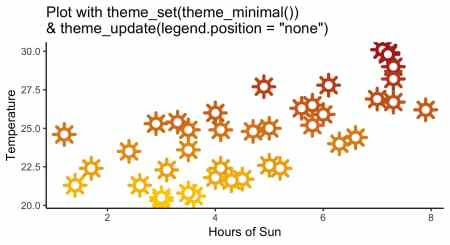
While predefined themes are very convenient, there’s always the option to (additionally) tweak the appearance of any non-data detail of a plot via the various arguments of theme(). This can be done for a specific plot, or the currently active default theme. The default theme can be updated or partly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

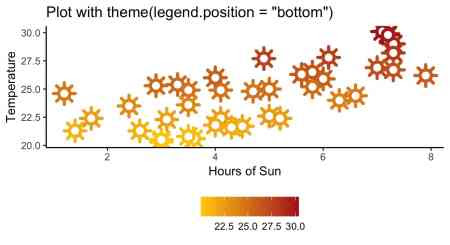


# changing the theme directly applied to the plot

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"),

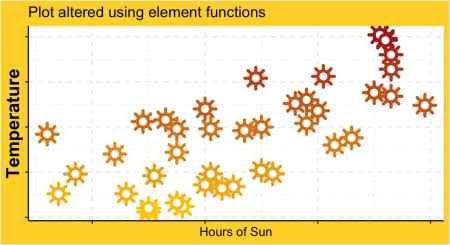
panel.grid.major = element\_line(linetype = "dashed"),

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

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

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

ggtitle("Plot altered using element functions")



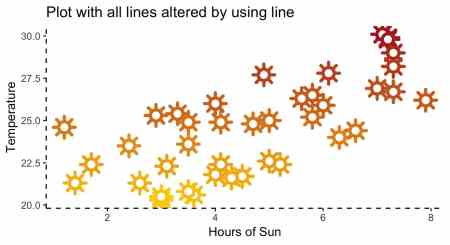
If we don’t want to change the display of some specific plot elements, but of all text, lines, titles or rectangular elements we can do so by specifying the arguments text, line, rect and title. Specifications passed to these arguments are inherited by all elements of the respective type. This inheritance 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")

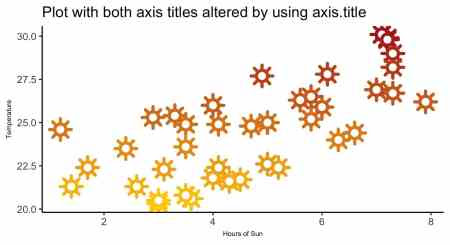


# using overreaching arguments #2

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")



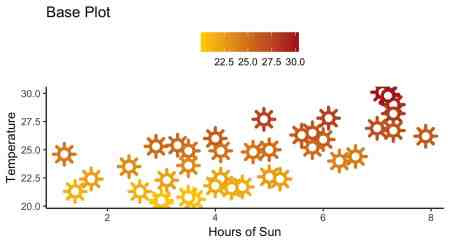
**Outlook**

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 +

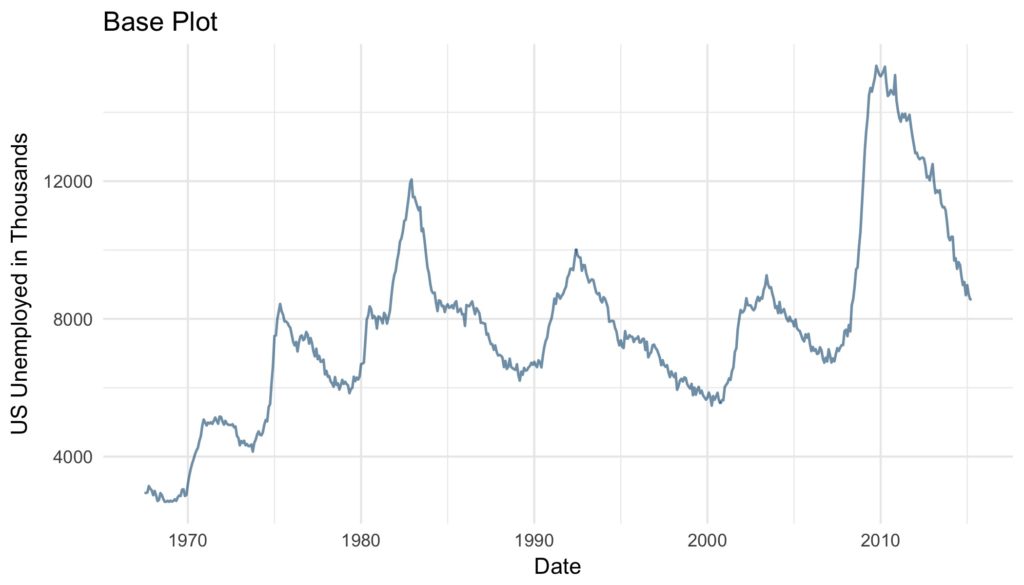
theme(legend.position = "top")



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.

ggplot2 – Customizing Date and Time Scales

We are using ggplot’s economics data set. Our base Plot looks like this:



base\_plot <- ggplot(data = economics) +

geom\_line(aes(x = date, y = unemploy),

color = "#09557f",

alpha = 0.6,

size = 0.6) +

labs(x = "Date",

y = "US Unemployed in Thousands",

title = "Base Plot") +

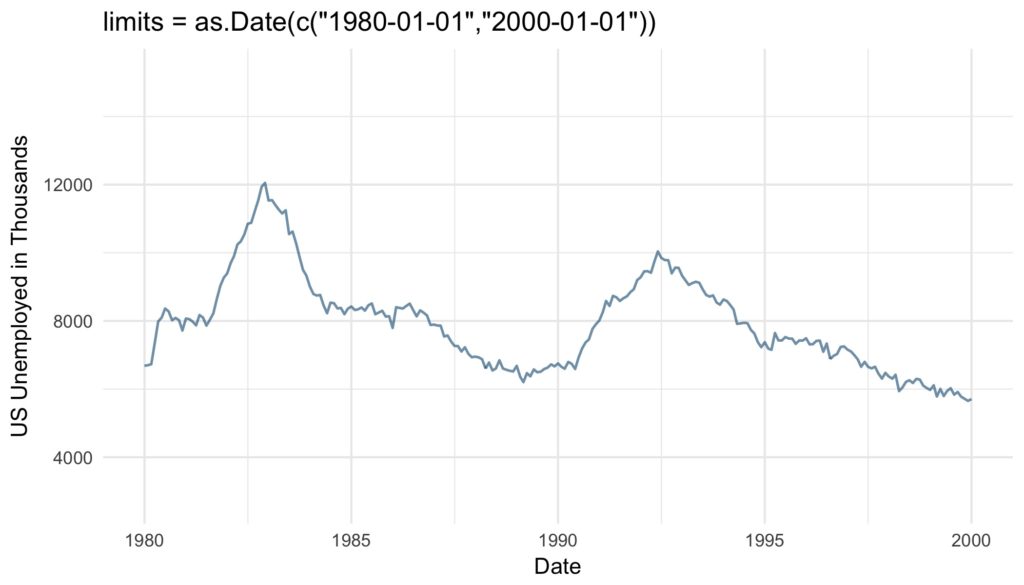
theme\_minimal()

**Scale Types**

As of now, ggplot2 supports three date and time classes: POSIXct, Date and hms. Depending on the class at hand, axis ticks and labels can be controlled by using scale\_\*\_date, scale\_\*\_datetime or scale\_\*\_time, respectively. Depending on whether one wants to modify the x or the y axis scale\_x\_\* or scale\_y\_\* are to be employed. For sake of simplicity, in the examples only scale\_x\_date is employed, but all discussed arguments work just the same for all mentioned scales.

**Minor Modifications**

Let’s start easy. With the argument limits the range of the displayed dates or time can be set. Two values of the correct date or time class have to be supplied.

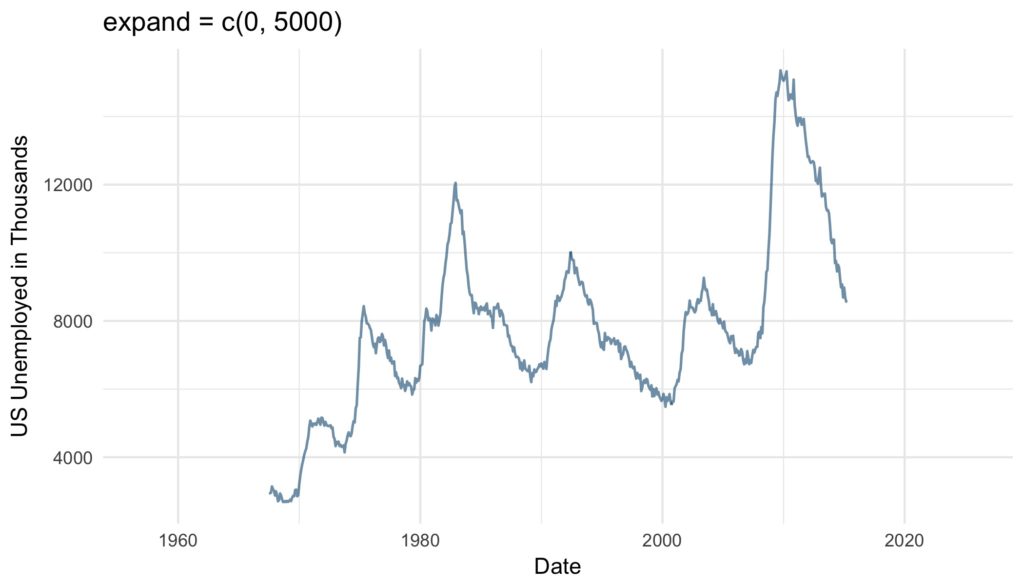


base\_plot +

scale\_x\_date(limits = as.Date(c("1980-01-01","2000-01-01"))) +

ggtitle("limits = as.Date(c("1980-01-01","2000-01-01"))")

The expand argument ensures that there is some distance between the displayed data and the axes. The multiplicative constant is multiplied with the range of the displayed data, the additive is multiplied with one unit of the depicted data. The sum of the two resulting distances is added to the axis limits as padding. The resulting empty space is added at the left and right end of the x-axis or the top and bottom of the y-axis.

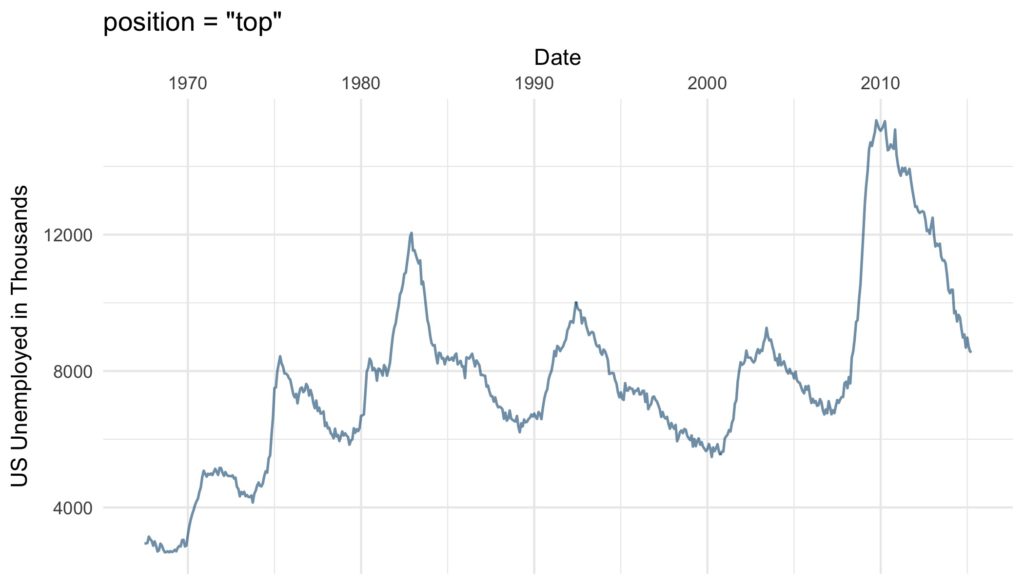


base\_plot +

scale\_x\_date(expand = c(0, 5000)) + #5000/365 = 13.69863 years

ggtitle("expand = c(0, 5000)")

position argument defines where the labels are displayed: Either “left” or “right” from the y-axis or on the “top” or on the “bottom” of the x-axis.



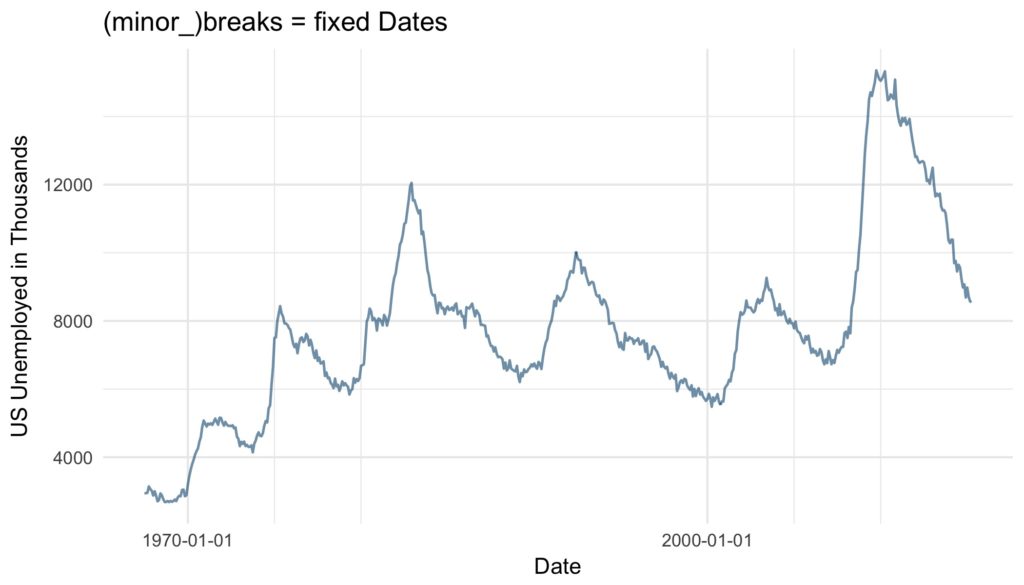
base\_plot +

scale\_x\_date(position = "top") +

ggtitle("position = "top"")

**Axis Ticks and Grid Lines**

More essential than the cosmetic modifications discussed so far are the axis ticks. There are several ways to define the axis ticks of dates and times. There are the labelled *major breaks* and further the *minor breaks*, which are not labeled but marked by grid lines. These can be customized with the arguments breaks and minor\_breaks, respectively. The breaks as the well as minor\_breaks can be defined by a numeric vector of exact positions or a function with the axis limits as inputs and breaks as outputs. Alternatively, the arguments can be set to NULL to display (minor) breaks at all. These options are especially handy if irregular intervals between breaks are desired.



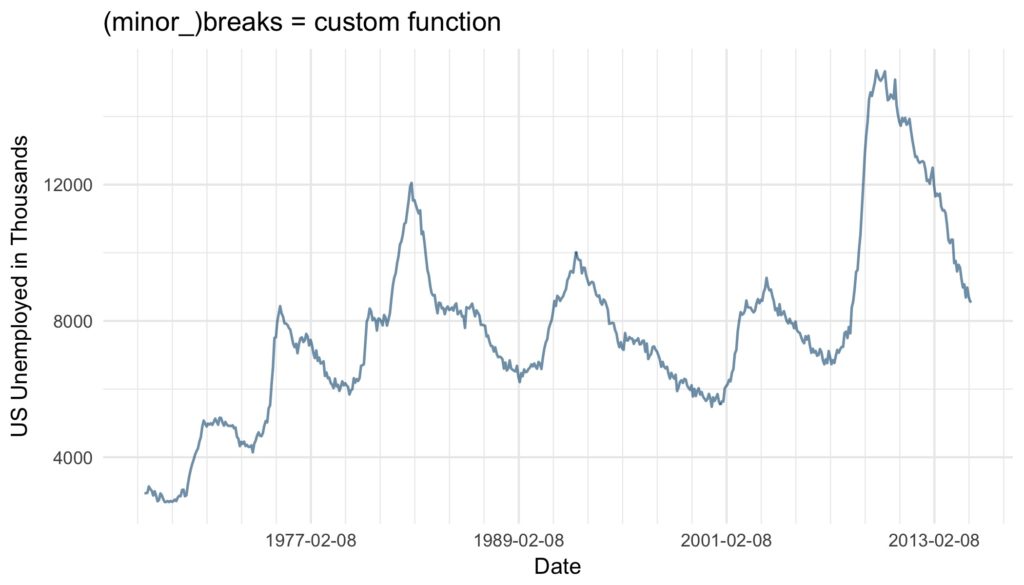
base\_plot +

scale\_x\_date(breaks = as.Date(c("1970-01-01", "2000-01-01")),

minor\_breaks = as.Date(c("1975-01-01", "1980-01-01",

"2005-01-01", "2010-01-01"))) +

ggtitle("(minor\_)breaks = fixed Dates")



base\_plot +

scale\_x\_date(breaks = function(x) seq.Date(from = min(x),

to = max(x),

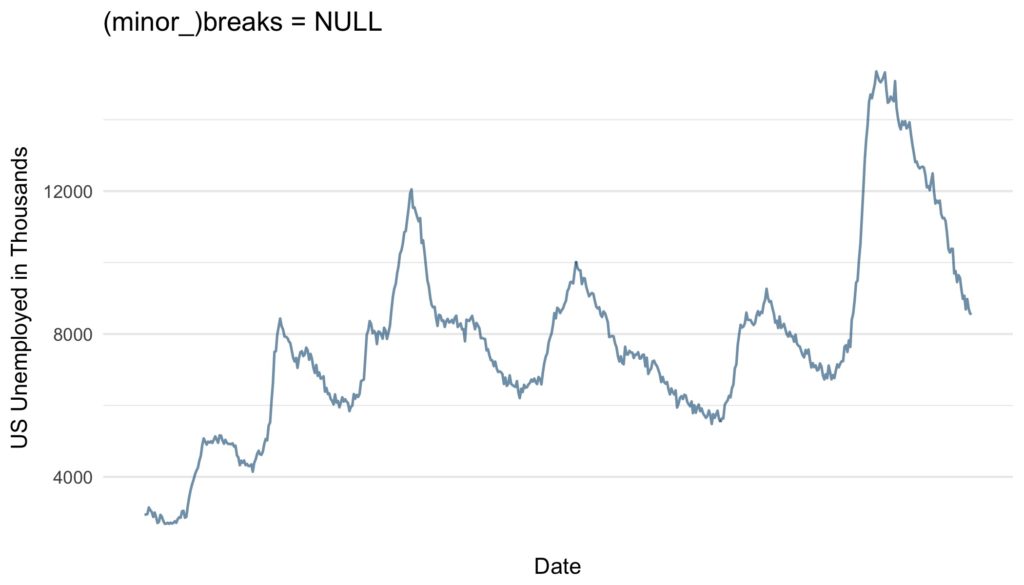
by = "12 years"),

minor\_breaks = function(x) seq.Date(from = min(x),

to = max(x),

by = "2 years")) +

ggtitle("(minor\_)breaks = custom function")



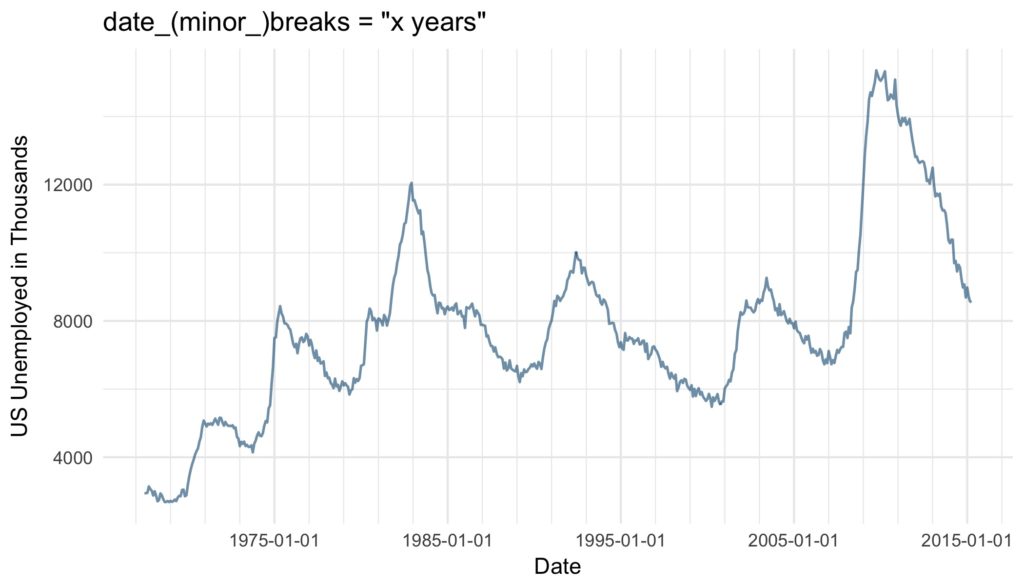
base\_plot +

scale\_x\_date(breaks = NULL,

minor\_breaks = NULL) +

ggtitle("(minor\_)breaks = NULL")

Another and very convenient way to define regular breaks are the date\_breaks and the date\_minor\_breaks argument. As input both arguments take a character vector combining a string specifying the time unit (either “sec”, “min”, “hour”, “day”, “week”, “month” or “year”) and an integer specifying number of said units specifying the break intervals.



base\_plot +

scale\_x\_date(date\_breaks = "10 years",

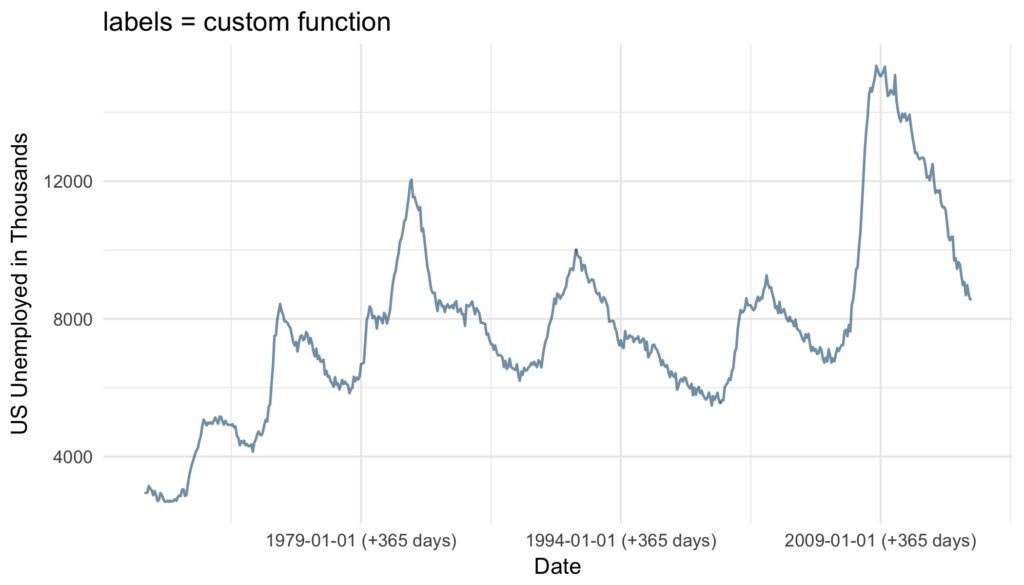
date\_minor\_breaks = "2 years") +

ggtitle("date\_(minor\_)breaks = "x years"")

If both are given, date(\_minor)\_breaks overrules (minor\_)breaks.

**Axis Labels**

Similar to the axis ticks, the format of the displayed labels can either be defined via the labels or the date\_labels argument. The labels argument can either be set to NULL if no labels should be displayed, with the breaks as inputs and the labels as outputs. Alternatively, a character vector with labels for all the breaks can be supplied to the argument. This can be very useful, since like this virtually any character vector can be used to label the breaks. The number of labels must be the same as the number of breaks. If the breaks are defined by a function, date\_breaks or by default the labels must be defined by a function as well.

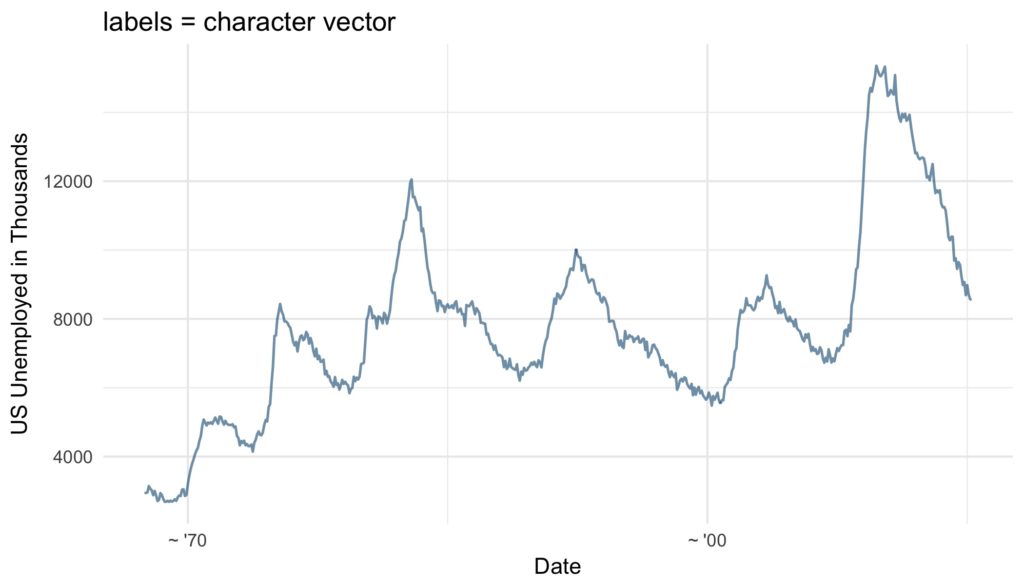


base\_plot +

scale\_x\_date(date\_breaks = "15 years",

labels = function(x) paste((x-365), "(+365 days)")) +

ggtitle("labels = custom function")



base\_plot +

scale\_x\_date(breaks = as.Date(c("1970-01-01", "2000-01-01")),

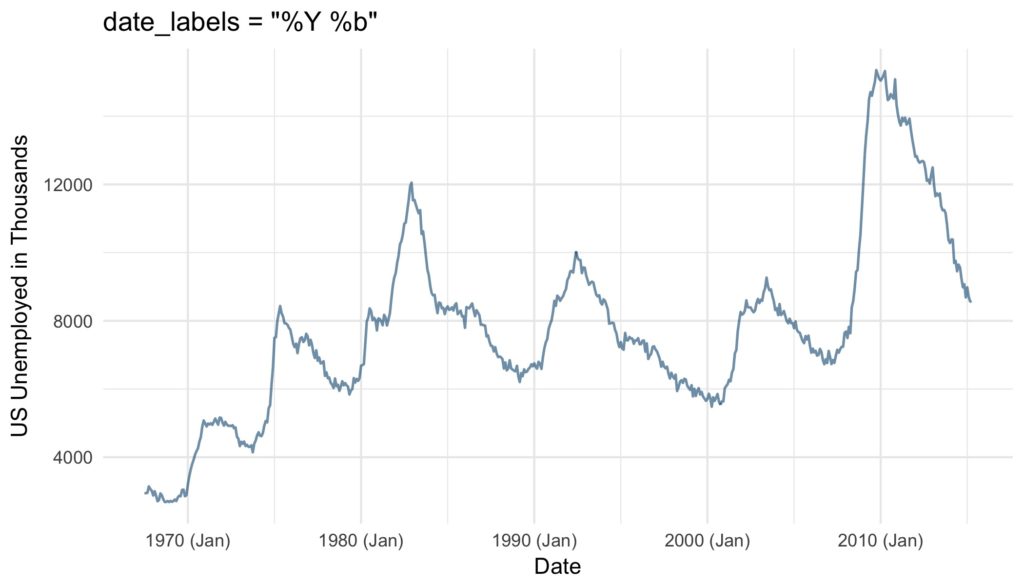
labels = c("~ '70", "~ '00")) +

ggtitle("labels = character vector")

Furthermore and very conveniently, the format of the labels can be controlled via the argument date\_labels set to a string of formatting codes, defining order, format and elements to be displayed:

| **Code** | **Meaning** |
| --- | --- |
| %S | second (00-59) |
| %M | minute (00-59) |
| %l | hour, in 12-hour clock (1-12) |
| %I | hour, in 12-hour clock (01-12) |
| %H | hour, in 24-hour clock (01-24) |
| %a | day of the week, abbreviated (Mon-Sun) |
| %A | day of the week, full (Monday-Sunday) |
| %e | day of the month (1-31) |
| %d | day of the month (01-31) |
| %m | month, numeric (01-12) |
| %b | month, abbreviated (Jan-Dec) |
| %B | month, full (January-December) |
| %y | year, without century (00-99) |
| %Y | year, with century (0000-9999) |

Source: Wickham 2009 p. 99



base\_plot +

scale\_x\_date(date\_labels = "%Y (%b)") +

ggtitle("date\_labels = "%Y (%b)"")

The choice of axis ticks and labels might seem trivial. However, one should not underestimate the amount of confusion that can be caused by too many, too less or poorly positioned axis ticks and labels. Further, economical yet clear labeling of axis ticks can increase the readability and visual appeal of any time series plot immensely. Since it is so easy to tweak the date and time axes in ggplot2 there is simply no excuse not to do so.