Data Visualization in R with ggplot2

The Structure of ggplot2 (Part 4)

Cédric Scherer

Physalia Courses | November 9-13 2020

Photo by Richard Strozynski

Part 4

Visual Themes

Illustration by Allison Horst (github.com/allisonhorst/stats-illustrations)

The Structure of `ggplot2`

Layer	Function	Explanation
Data	`ggplot(data)`	The raw data that you want to visualise.
Aesthetics	`aes()`	Aesthetic mappings of the geometric and statistical objects.
Layers	geom_*() and `stat_*()`	The geometric shapes and statistical summaries representing the data.
Scales	`scale_*()`	Maps between the data and the aesthetic dimensions.
Coordinate System	`coord_*()`	Maps data into the plane of the data rectangle.
Facets	`facet_*()`	The arrangement of the data into a grid of plots.
Visual Themes	`theme()` and `theme_*()`	The overall visual defaults of a plot.
Annotations	`annotate()`	Add additional labels, geometries or images to a plot.

Visual Themes

`theme()` and `theme_*`

`theme_*()`

- Themes control the display of all non-data elements of the plot
    - titles
    - labels
    - fonts
    - background + borders
    - gridlines
    - legends
- Themes can be modified for each plot by calling `theme()` and `element_` functions
- Themes can be used to give plots a consistent customized look by defining a custom theme using `theme_update()` or `theme_set()`

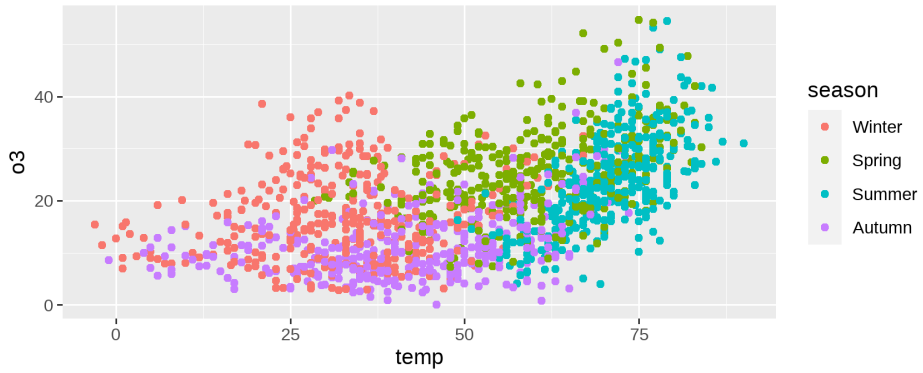
Visual Themes

`theme_*()` and `theme()`

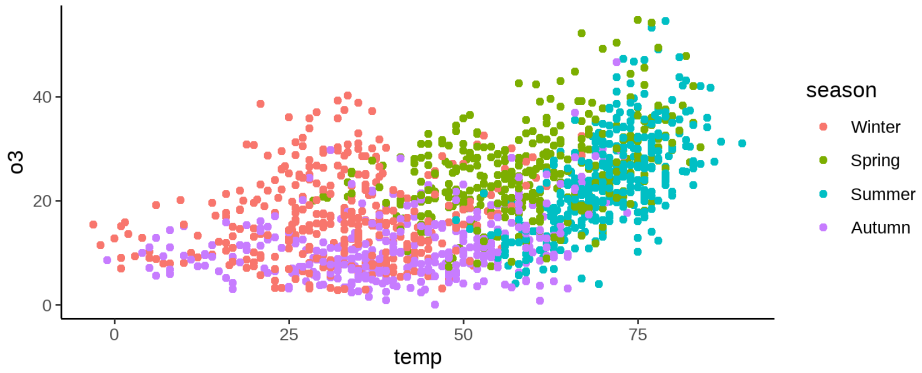
theme_*()

We have already seen several build-in theme examples in the first `ggplot2` lecture:

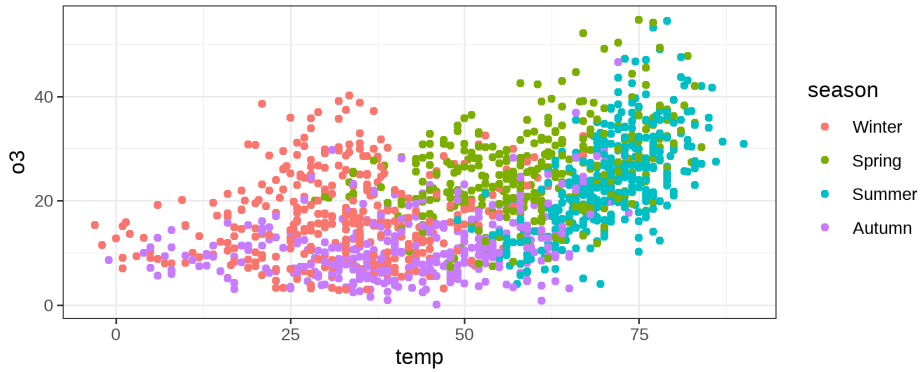
theme_grey() or theme_gray()



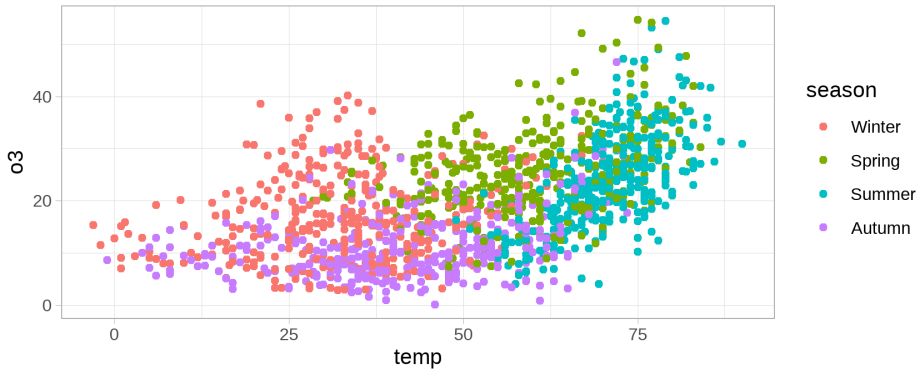
theme_classic()



theme_bw()

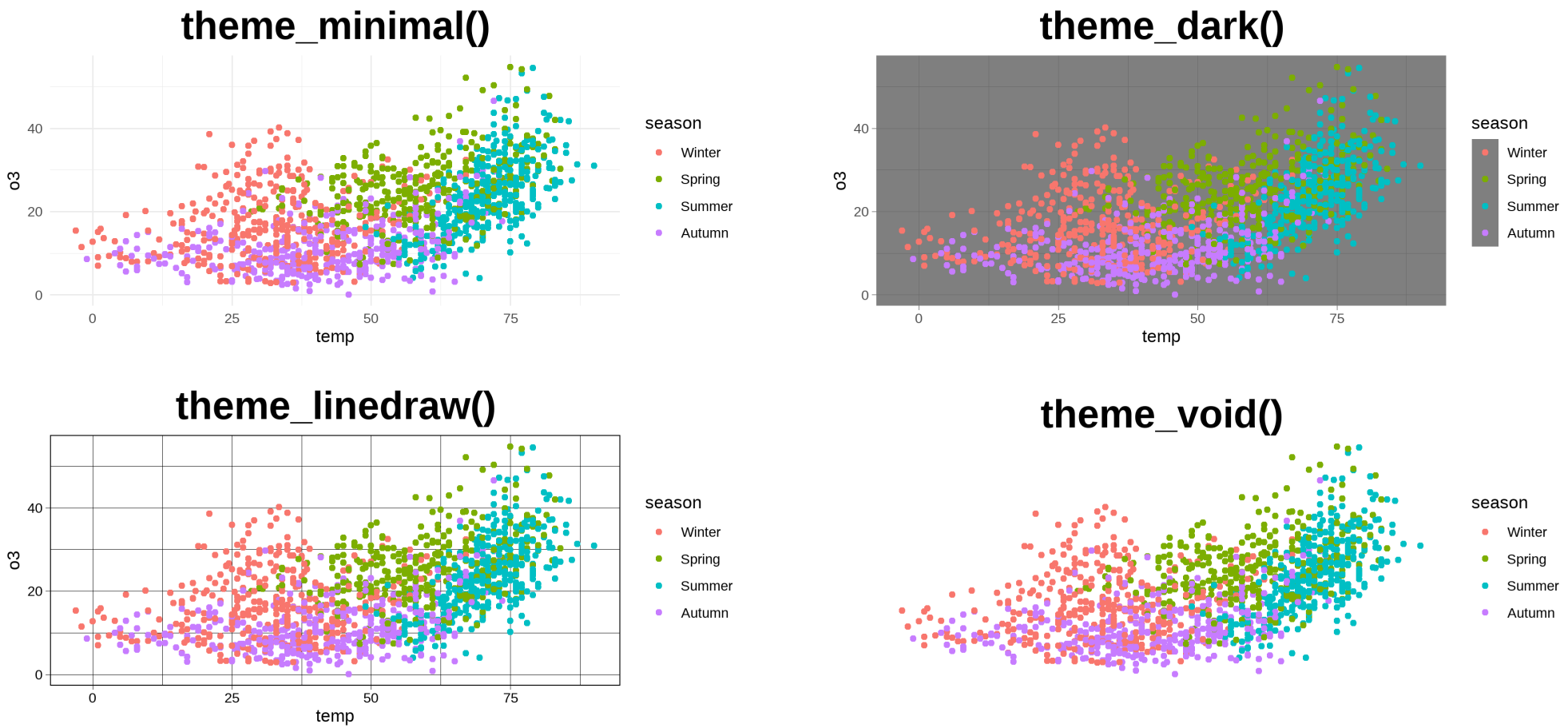


theme_light()



`theme_*()`

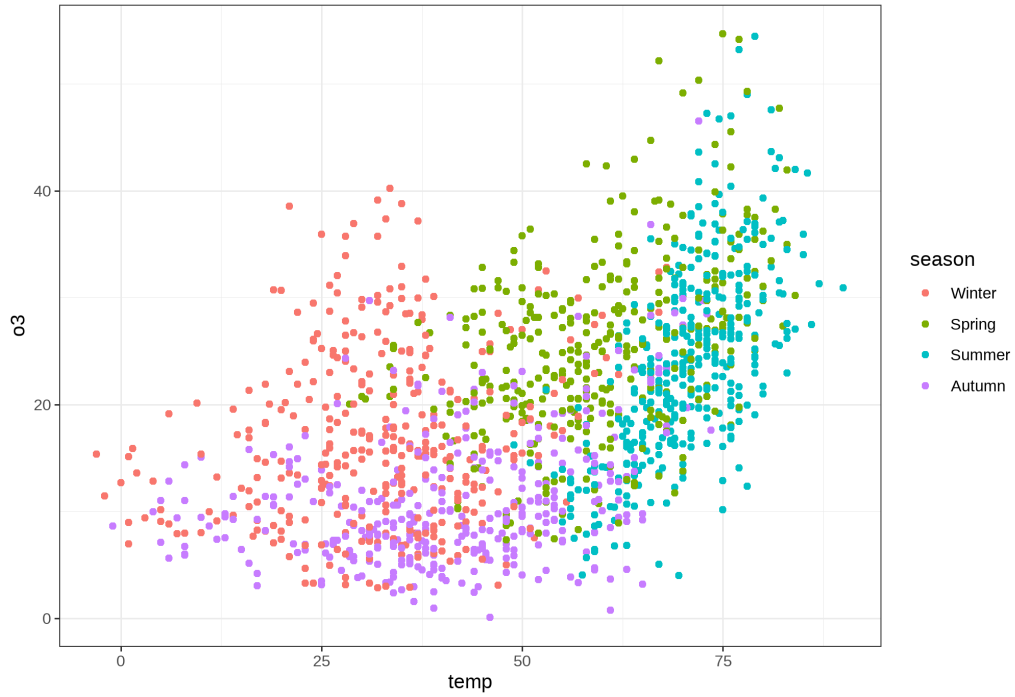
We have already seen several build-in theme examples in the first `ggplot2` lecture:



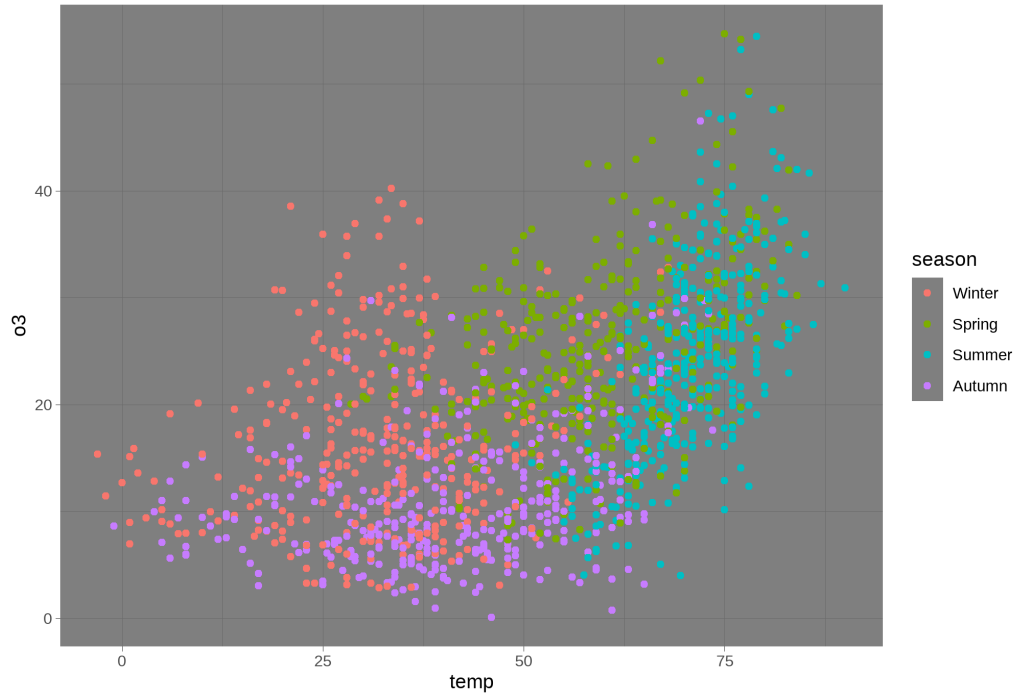
theme_*()

You can set themes **for each plot** or globally for the current session with `theme_set()`:

```
ggplot(chic, aes(temp, o3)) +
  geom_point(aes(color = season)) +
  theme_bw()
```



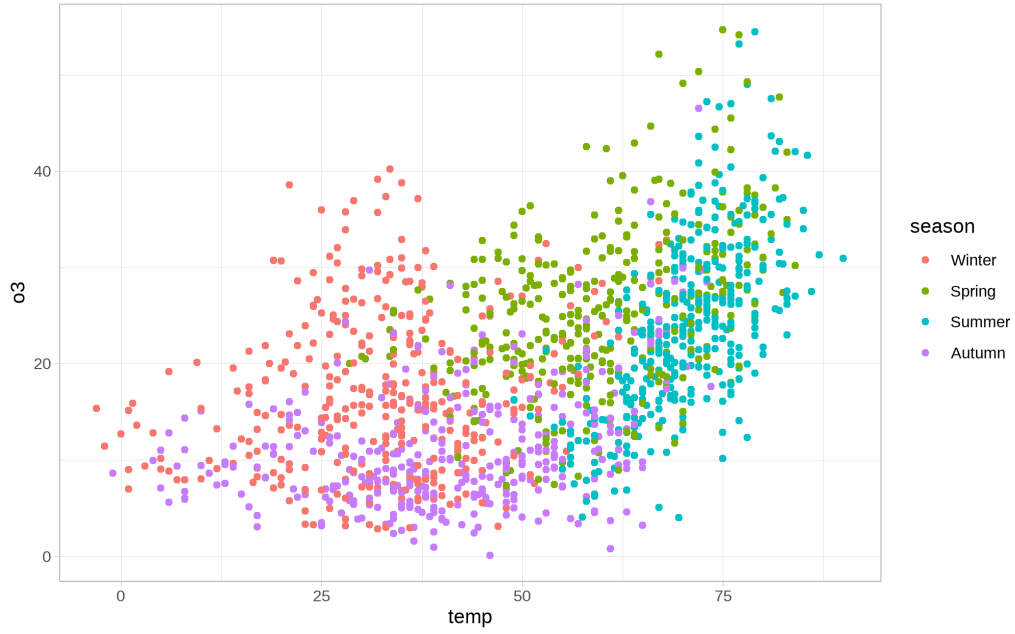

```
ggplot(chic, aes(temp, o3)) +
  geom_point(aes(color = season)) +
  theme_dark()
```



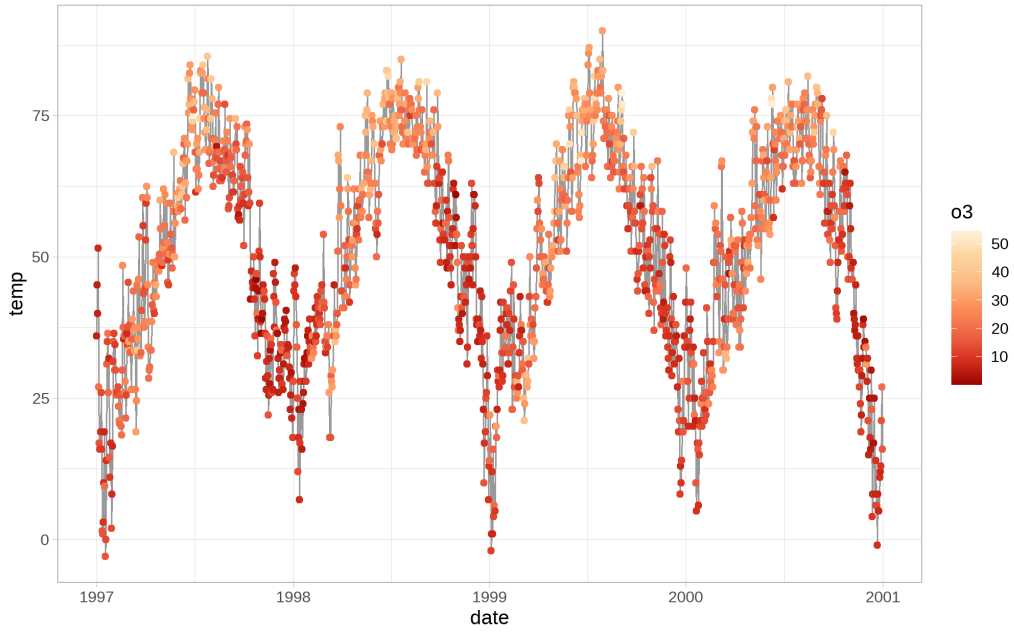
theme_set()

You can set themes for each plot or **globally for the current session and thus all following plots** with `theme_set()`:

```
theme_set(theme_light())

(g <- ggplot(chic, aes(temp, o3)) +
  geom_point(aes(color = season)))
```



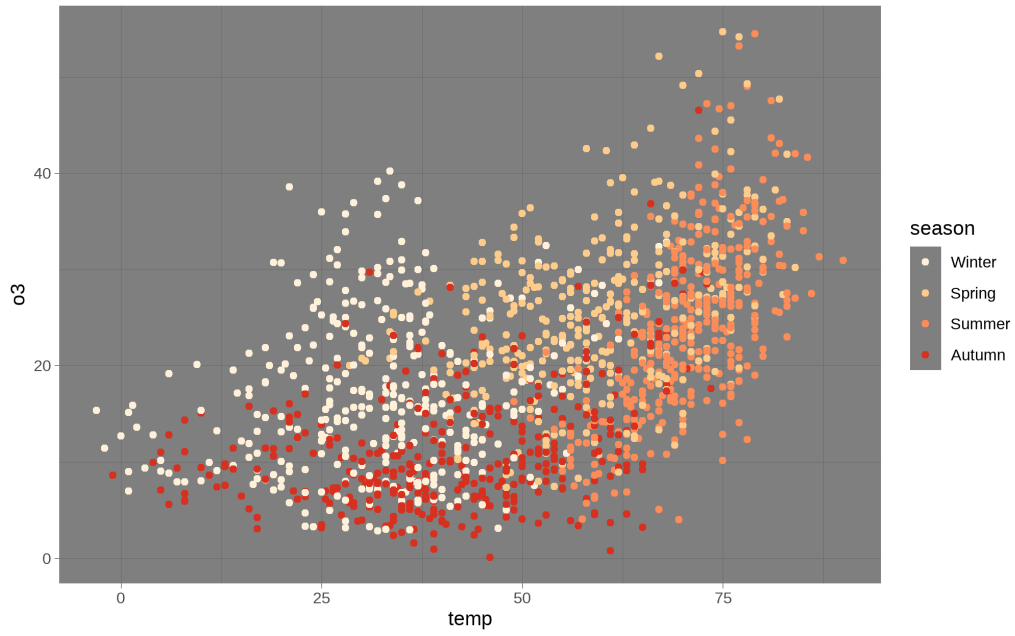

```
ggplot(chic, aes(date, temp)) +
  geom_line(color = "grey60") +
  geom_point(aes(color = o3)) +
  scale_color_distiller(palette = 8)
```



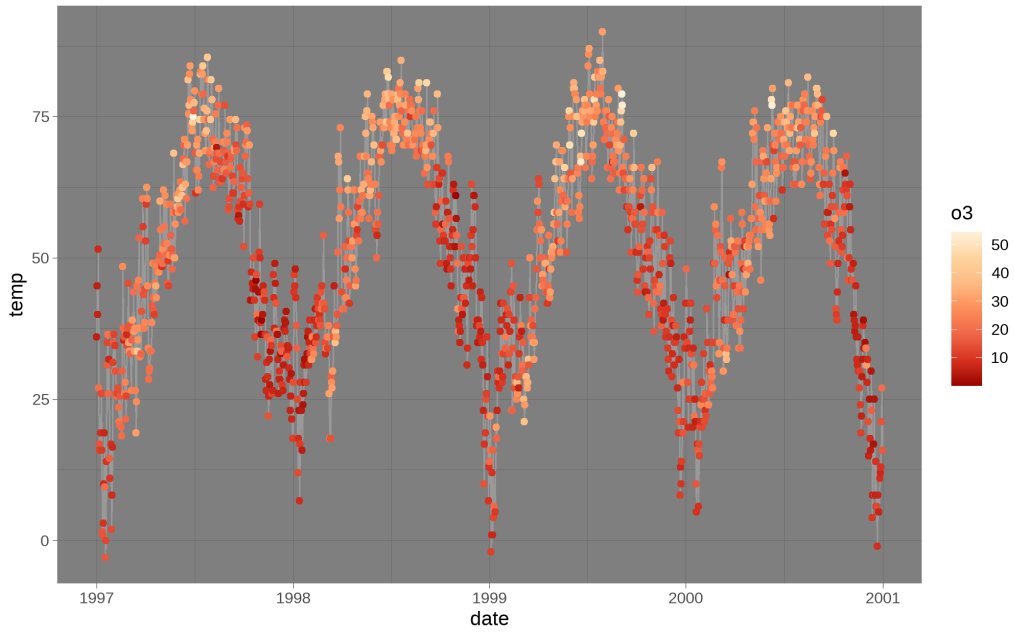
theme_set()

You can set themes for each plot or **globally for the current session and thus all following plots** with `theme_set()`:

```
old_theme <- theme_set(theme_dark())

g +
  scale_color_brewer(palette = 8)
```



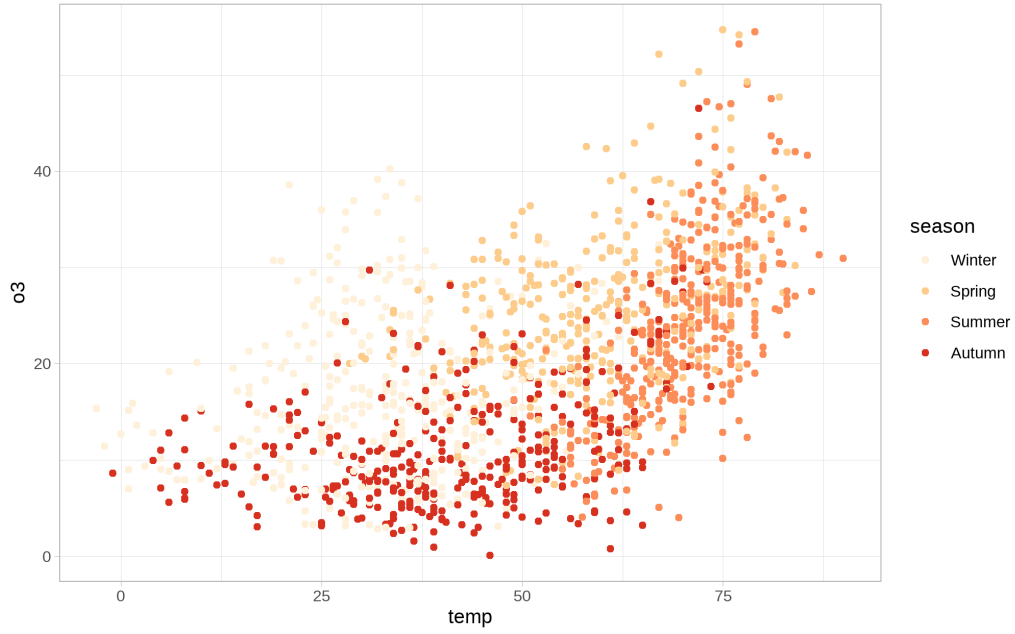

```
ggplot(chic, aes(date, temp)) +
  geom_line(color = "grey60") +
  geom_point(aes(color = o3)) +
  scale_color_distiller(palette = 8)
```



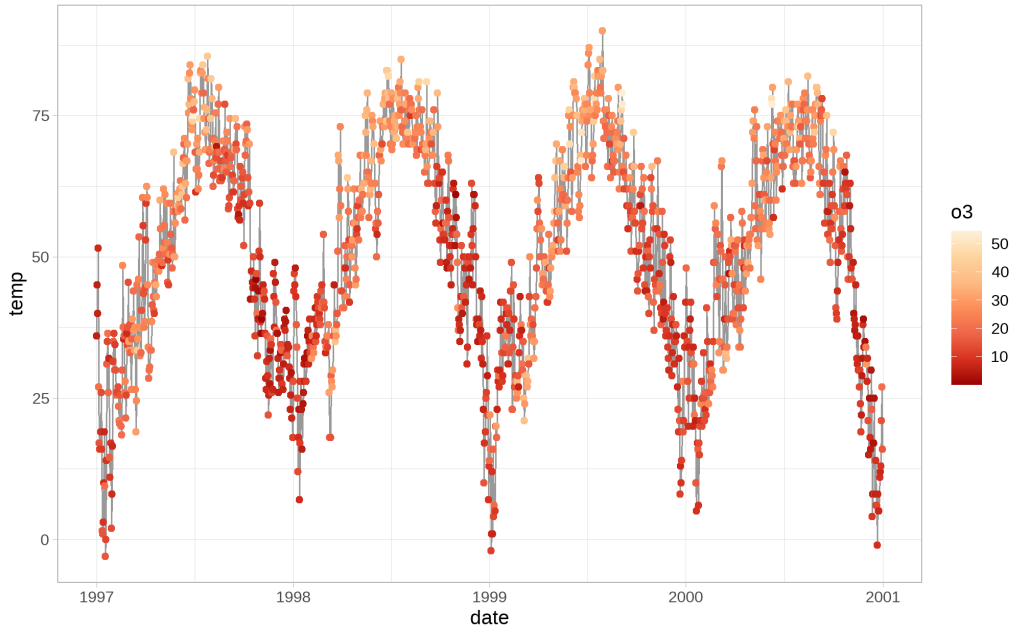
theme_set()

`theme_set()` invisibly returns the previous theme so you can easily save it, then later restore it:

```
theme_set(old_theme)

g +
  scale_color_brewer(palette = 8)
```



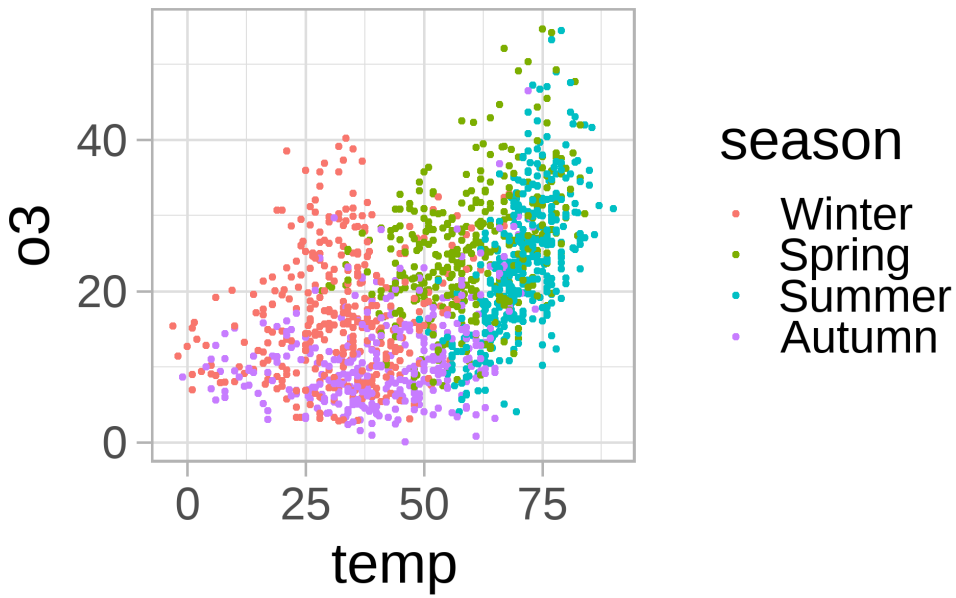

```
ggplot(chic, aes(date, temp)) +
  geom_line(color = "grey60") +
  geom_point(aes(color = o3)) +
  scale_color_distiller(palette = 8)
```



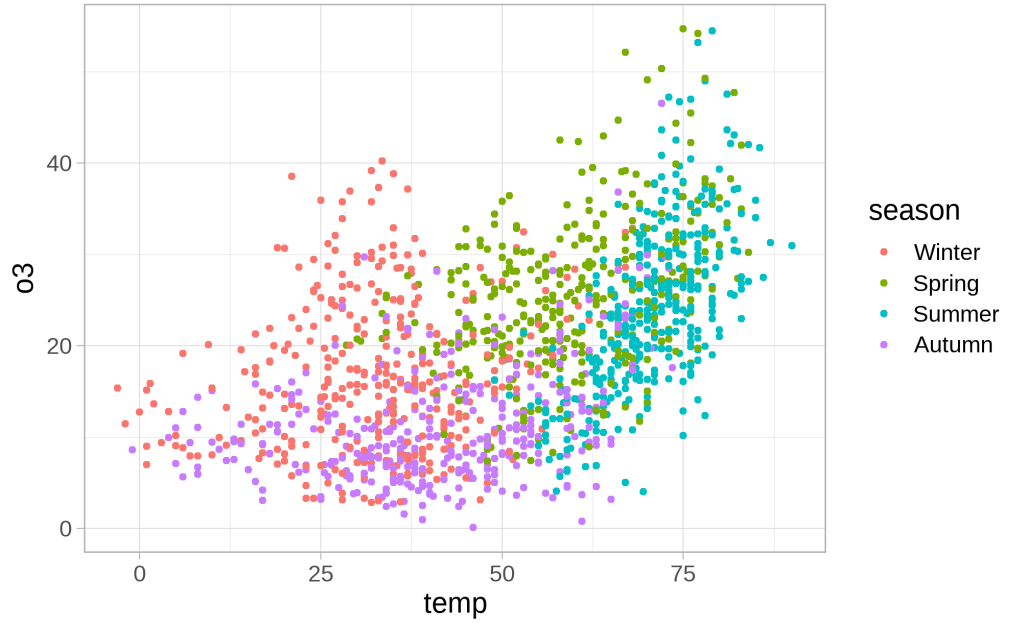
theme_*(base_size)

Themes have a `base_size` argument which is a simple way to increase or decrease the font size:

```
theme_set(theme_light(base_size = 32))

g
```



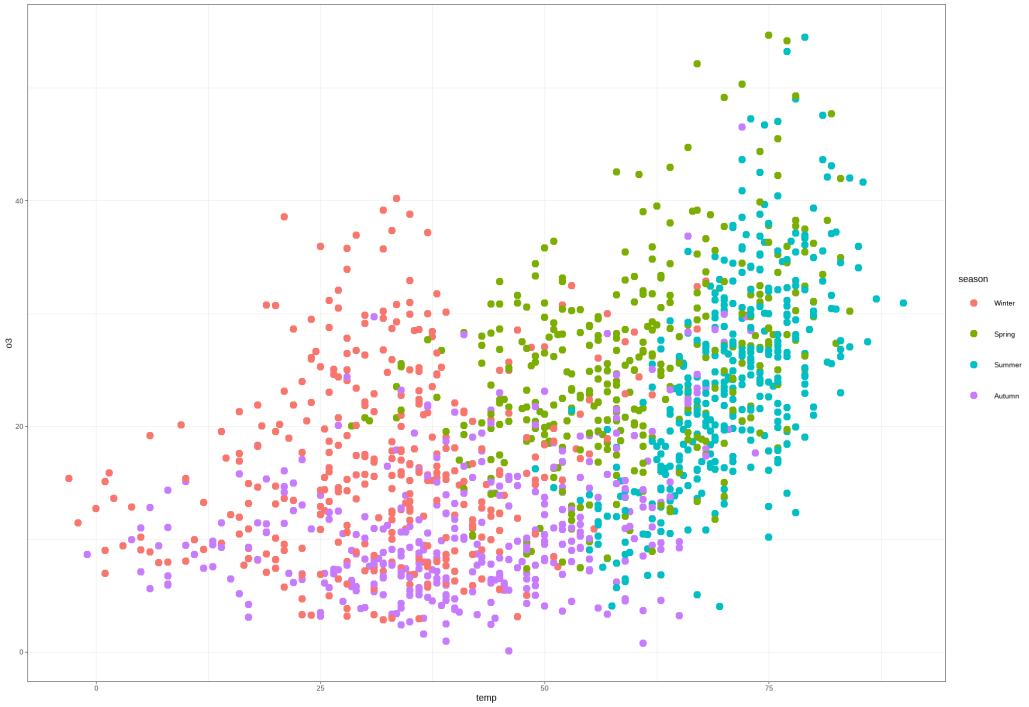

```
theme_set(theme_light(base_size = 16))

g
```



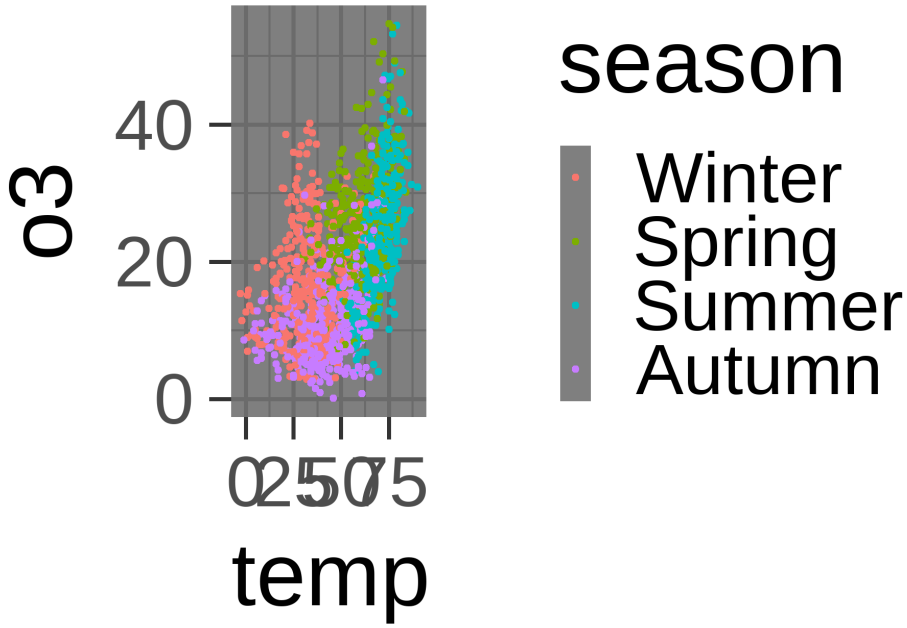
theme_*(base_size)

Themes have a `base_size` argument which is a simple way to increase or decrease the font size:

```
g +
  theme_bw(base_size = 5)
```



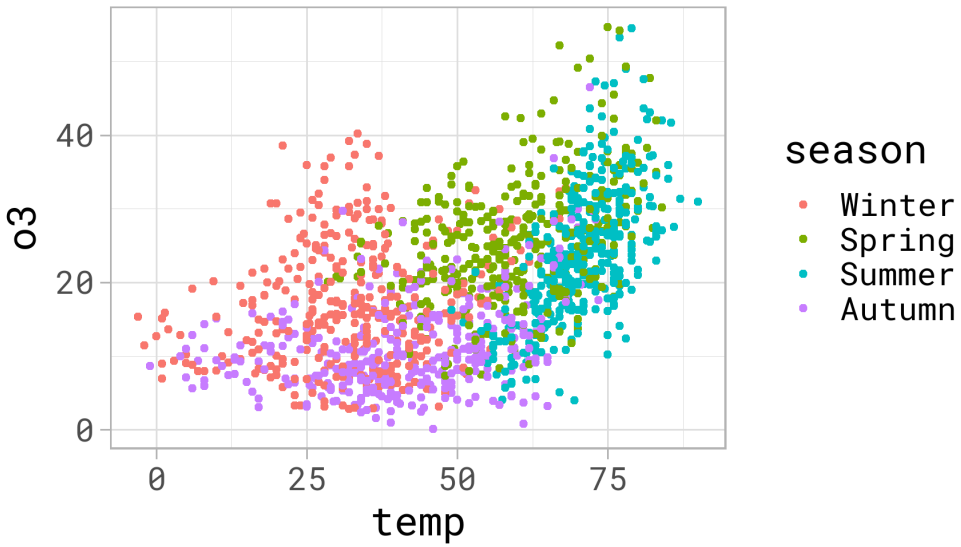

```
g +
  theme_dark(base_size = 50)
```



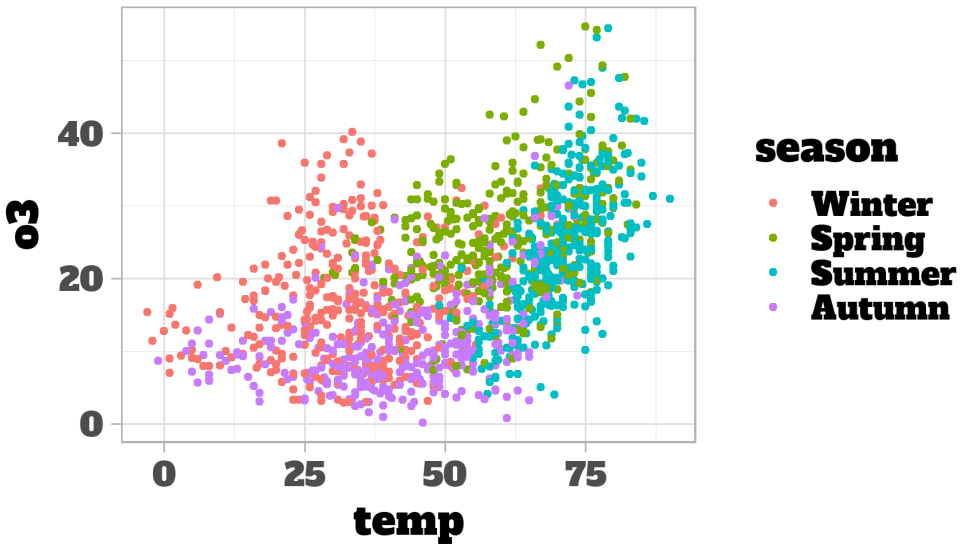
theme_*(base_family)

You can also set the font of the theme via `base_family`:

```
theme_set(theme_light(
    base_size = 20,
    base_family = "Roboto Mono"
  ))
g
```




```
theme_set(theme_light(
    base_size = 20,
    base_family = "Alfa Slab One"
  ))
g
```



Fonts: The `showtext` Package

> The showtext package, is able to support more font formats and more graphics devices, and avoids using external software such as Ghostscript. showtext makes it even easier to use various types of fonts (TrueType, OpenType, Type 1, web fonts, etc.) in R graphs.

```
install.packages("showtext")
library(showtext)

## Loading Google fonts
## (https://fonts.google.com/)
## installed on your system
font_add_google("Roboto", "Roboto")
font_add_google("Roboto Mono", "Roboto Mono")
```

Fonts: The `systemfonts` Package

> The systemfonts package locates installed fonts. It uses the system-native libraries on Mac (CoreText) and Linux (FontConfig), and uses Freetype to parse the fonts in the registry on Windows.

```
## install.packages("systemfonts")
library(systemfonts)

## Fonts installed on your system
system_fonts()
## # A tibble: 1,083 x 9
##    path             index name      family   style  weight width italic monospace
##    <chr>            <int> <chr>     <chr>    <chr>  <ord>  <ord> <lgl>  <lgl>
##  1 "C:\\WINDOWS\\~      0 ArialMT   Arial    Regul~ normal norm~ FALSE  FALSE
##  2 "C:\\WINDOWS\\~      0 Arial-Bl~ Arial    Black  heavy  norm~ FALSE  FALSE
##  3 "C:\\WINDOWS\\~      0 Arial-Bo~ Arial    Bold   bold   norm~ FALSE  FALSE
##  4 "C:\\WINDOWS\\~      0 Arial-Bo~ Arial    Bold ~ bold   norm~ TRUE   FALSE
##  5 "C:\\WINDOWS\\~      0 Arial-It~ Arial    Italic normal norm~ TRUE   FALSE
##  6 "C:\\WINDOWS\\~      0 Bahnschr~ Bahnsch~ Regul~ normal norm~ FALSE  FALSE
##  7 "C:\\WINDOWS\\~      0 Calibri   Calibri  Regul~ normal norm~ FALSE  FALSE
##  8 "C:\\WINDOWS\\~      0 Calibri-~ Calibri  Bold   bold   norm~ FALSE  FALSE
##  9 "C:\\WINDOWS\\~      0 Calibri-~ Calibri  Bold ~ bold   norm~ TRUE   FALSE
## 10 "C:\\WINDOWS\\~      0 Calibri-~ Calibri  Italic normal norm~ TRUE   FALSE
## # ... with 1,073 more rows
```

Fonts: The `systemfonts` Package

> The systemfonts package locates installed fonts. It uses the system-native libraries on Mac (CoreText) and Linux (FontConfig), and uses Freetype to parse the fonts in the registry on Windows.

```
## install.packages("systemfonts")
## library(systemfonts)

## Check font family
font_info("Oswald")
## # A tibble: 1 x 22
##   path  index family style italic bold  monospace kerning color scalable
##   <chr> <int> <chr>  <chr> <lgl>  <lgl> <lgl>     <lgl>   <lgl> <lgl>
## 1 "C:\~     0 Oswald Regu~ FALSE  FALSE FALSE     FALSE   FALSE TRUE
## # ... with 12 more variables: vertical <lgl>, n_glyphs <int>, n_sizes <int>,
## #   n_charmaps <int>, bbox <list>, max_ascend <dbl>, max_descend <dbl>,
## #   max_advance_width <dbl>, max_advance_height <dbl>, lineheight <dbl>,
## #   underline_pos <dbl>, underline_size <dbl>
```

Fonts: The `extrafont` Package

> Fonts that are imported into extrafont can be used with PDF or PostScript output files. On Windows, extrafont will also make system fonts available for bitmap output.

```
## install.packages("extrafont")
library(extrafont)
##
## import your system fonts
## ## (only 1 time)
## font_import() ## press "y"
## loadfonts()
```

Fonts: The `extrafont` Package

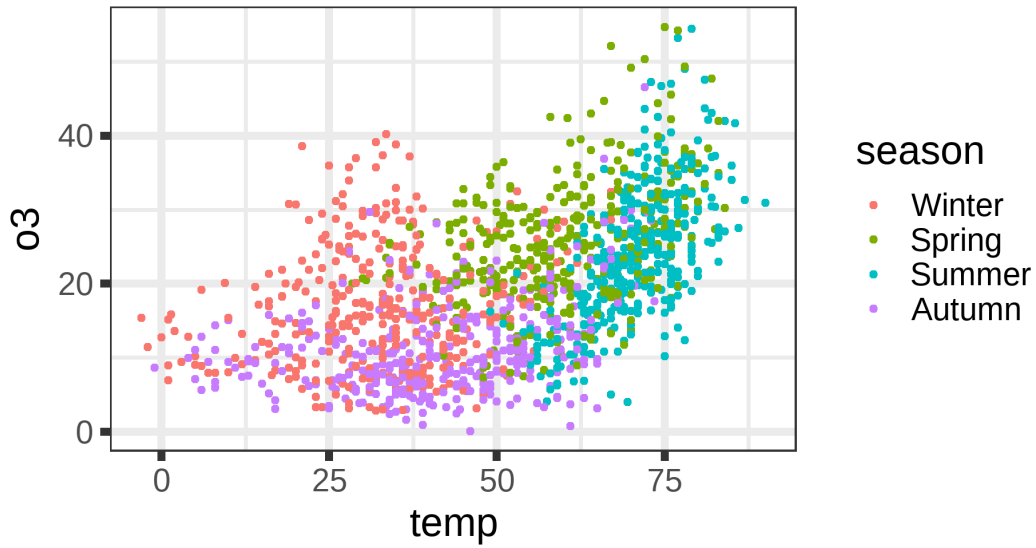
> Fonts that are imported into extrafont can be used with PDF or PostScript output files. On Windows, extrafont will also make system fonts available for bitmap output.

```
## which fonts are available?
fonts()
##   [1] "Agency FB"                    "Algerian"
##   [3] "Arial Black"                  "Arial"
##   [5] "Arial Narrow"                 "Arial Rounded MT Bold"
##   [7] "Arial Unicode MS"             "Bahnschrift"
##   [9] "Baskerville Old Face"         "Bauhaus 93"
##  [11] "Bell MT"                      "Berlin Sans FB"
##  [13] "Berlin Sans FB Demi"          "Bernard MT Condensed"
##  [15] "Blackadder ITC"               "Bodoni MT"
##  [17] "Bodoni MT Black"              "Bodoni MT Condensed"
##  [19] "Bodoni MT Poster Compressed"  "Book Antiqua"
##  [21] "Bookman Old Style"            "Bookshelf Symbol 7"
##  [23] "Bradley Hand ITC"             "Britannic Bold"
##  [25] "Broadway"                     "Brush Script MT"
##  [27] "Calibri"                      "Calibri Light"
##  [29] "Californian FB"               "Calisto MT"
##  [31] "Cambria"                      "Candara"
##  [33] "Candara Light"                "Castellar"
```

theme_*(base_line_size)

... and change the size of geometric elements via `base_line_size` and `base_rect_size`:

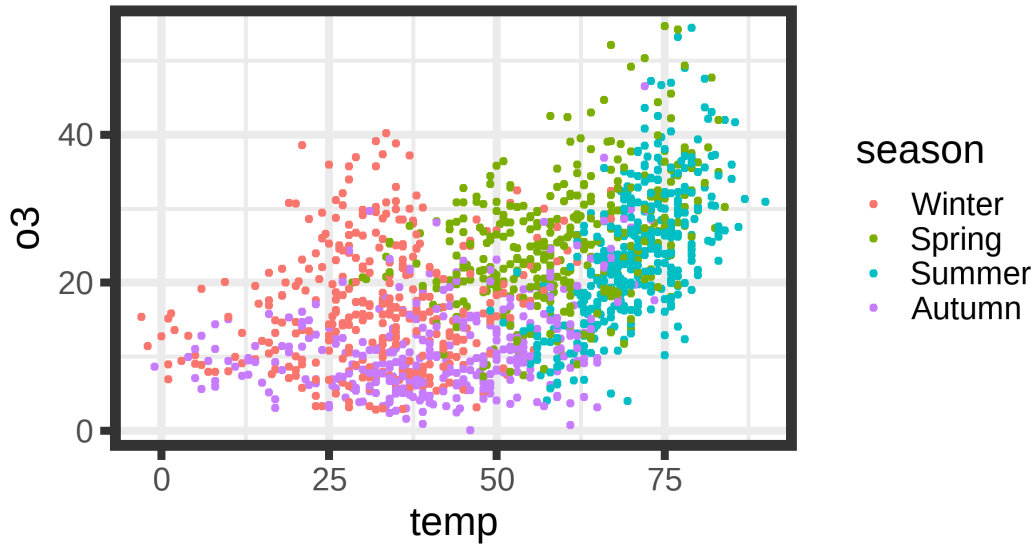
```
g +
  theme_bw(
    base_size = 20,
    base_line_size = 2
  )
```



theme_*(base_rect_size)

... and change the size of geometric elements via `base_line_size` and `base_rect_size`:

```
g +
  theme_bw(
    base_size = 20,
    base_line_size = 2,
    base_rect_size = 5
  )
```



`theme_*()`

- `theme_set()` completely overrides the current theme.
- `theme_update()` modifies a particular element of the current theme using the + operator.
- `theme_replace()` modifies a particular element of the current theme using the `%+replace%` operator.

For the last two, you first need to know what is available to *update* or *replace*...

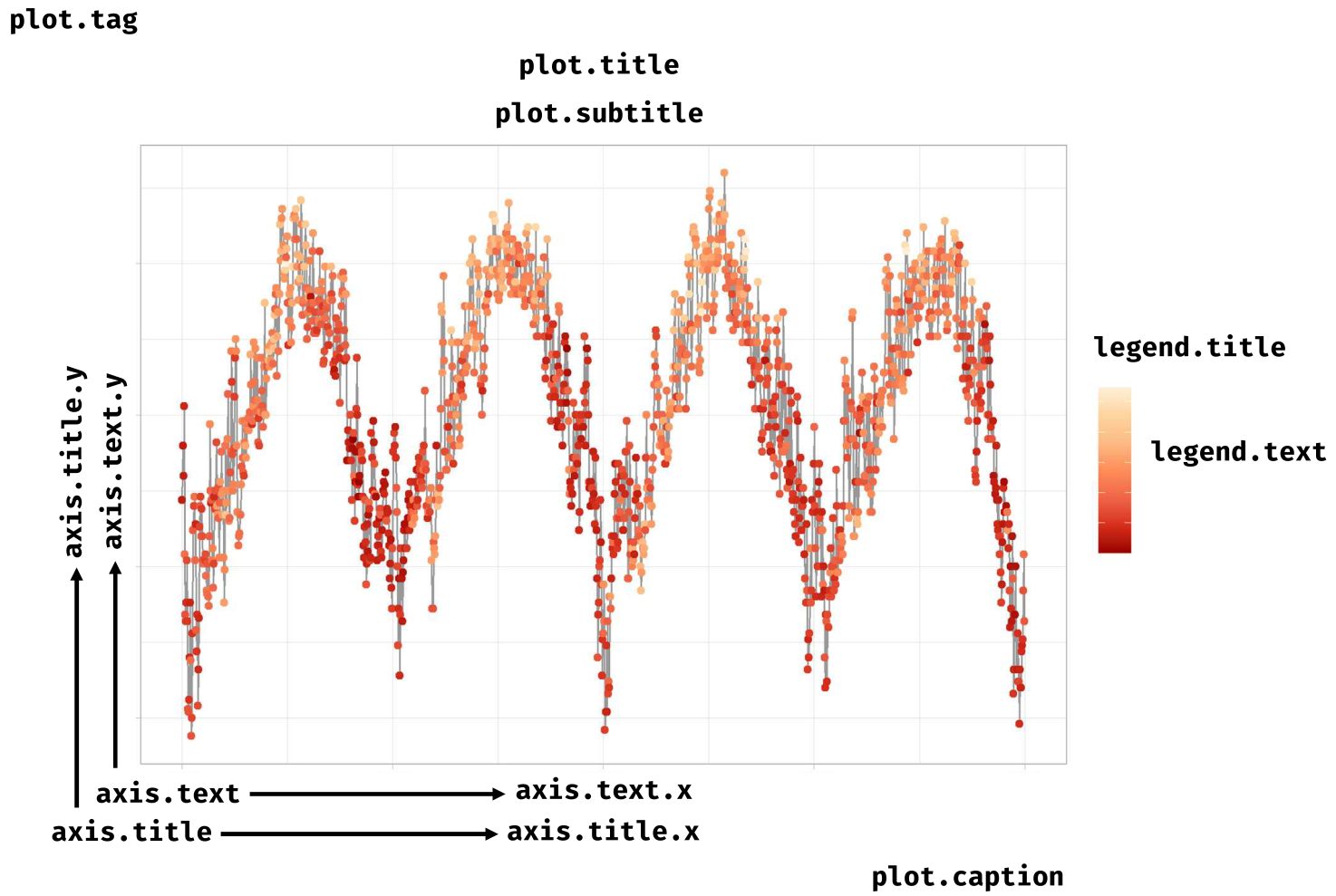
```
theme_grey
## function (base_size = 11, base_family = "", base_line_size = base_size/22,
##     base_rect_size = base_size/22)
## {
##     half_line <- base_size/2
##     t <- theme(line = element_line(colour = "black", size = base_line_size,
##         linetype = 1, lineend = "butt"), rect = element_rect(fill = "white",
##         colour = "black", size = base_rect_size, linetype = 1),
##         text = element_text(family = base_family, face = "plain",
##             colour = "black", size = base_size, lineheight = 0.9,
##             hjust = 0.5, vjust = 0.5, angle = 0, margin = margin(),
##             debug = FALSE), axis.line = element_blank(), axis.line.x = NULL,
##         axis.line.y = NULL, axis.text = element_text(size = rel(0.8),
##             colour = "grey30"), axis.text.x = element_text(margin = margin(t = 0.8
##             half_line/2), vjust = 1), axis.text.x.top = element_text(margin = margi
##             half_line/2), vjust = 0), axis.text.y = element_text(margin = margin(r
##             half_line/2), hjust = 1), axis.text.y.right = element_text(margin = mar
##             half_line/2), hjust = 0), axis.ticks = element_line(colour = "grey20"),
##         axis.ticks.length = unit(half_line/2, "pt"), axis.ticks.length.x = NULL,
##         axis.ticks.length.x.top = NULL, axis.ticks.length.x.bottom = NULL,
##         axis.ticks.length.y = NULL, axis.ticks.length.y.left = NULL,
##         axis.ticks.length.y.right = NULL, axis.title.x = element_text(margin = marg
##             vjust = 1), axis.title.x.top = element_text(margin = margin(b = half_li
##             vjust = 0), axis.title.y = element_text(angle = 90,
```

`theme()` Arguments

There are many elements you can customize. You can either group them by their type or by their category:

- element types:
  - `text` → all labels, axis text, legend title and text
  - `line` → axis lines, ticks, grid lines
  - `rect` → plot area, panel area, legend and legend keys, facets
- element category:
  - `axis.*` → titles, text, ticks, lines
  - `legend.*` → background, margin, spacing, keys, text, title, position, direction, box
  - `panel.*` → background, border, margin, spacing, grid (major and minor)
  - `plot.*` → background, title, subtitle, caption, tag, margin
  - `strip.*` → background, placement, text

Text Elements via `element_text()`



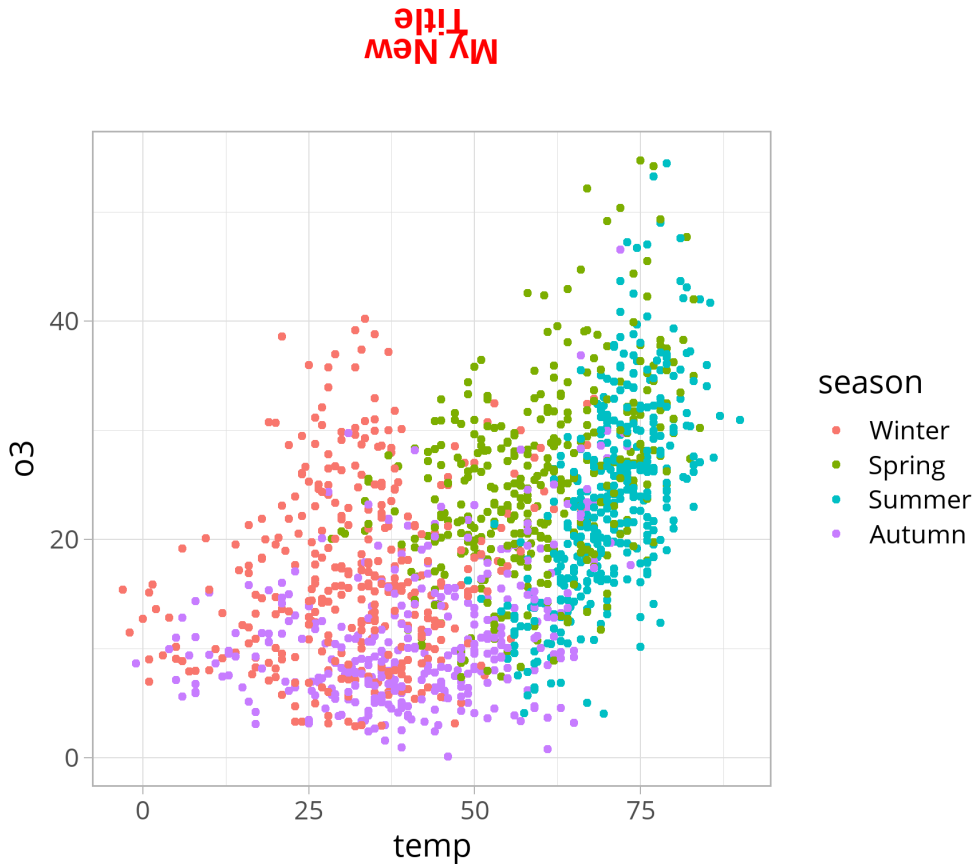
Text Elements via `element_text()`

```
element_text(
  family = "Roboto",
  face = "bold", ## plain, italic, bolditalic
  size = 18,
  color = "red",
  lineheight = .7,
  angle = 180,
  hjust = .5,
  vjust = .0,
  margin = margin(
    10, ## t (top)
    0,  ## r (right)
    30, ## b (bottom)
    0   ## l (left)
  )
)
```

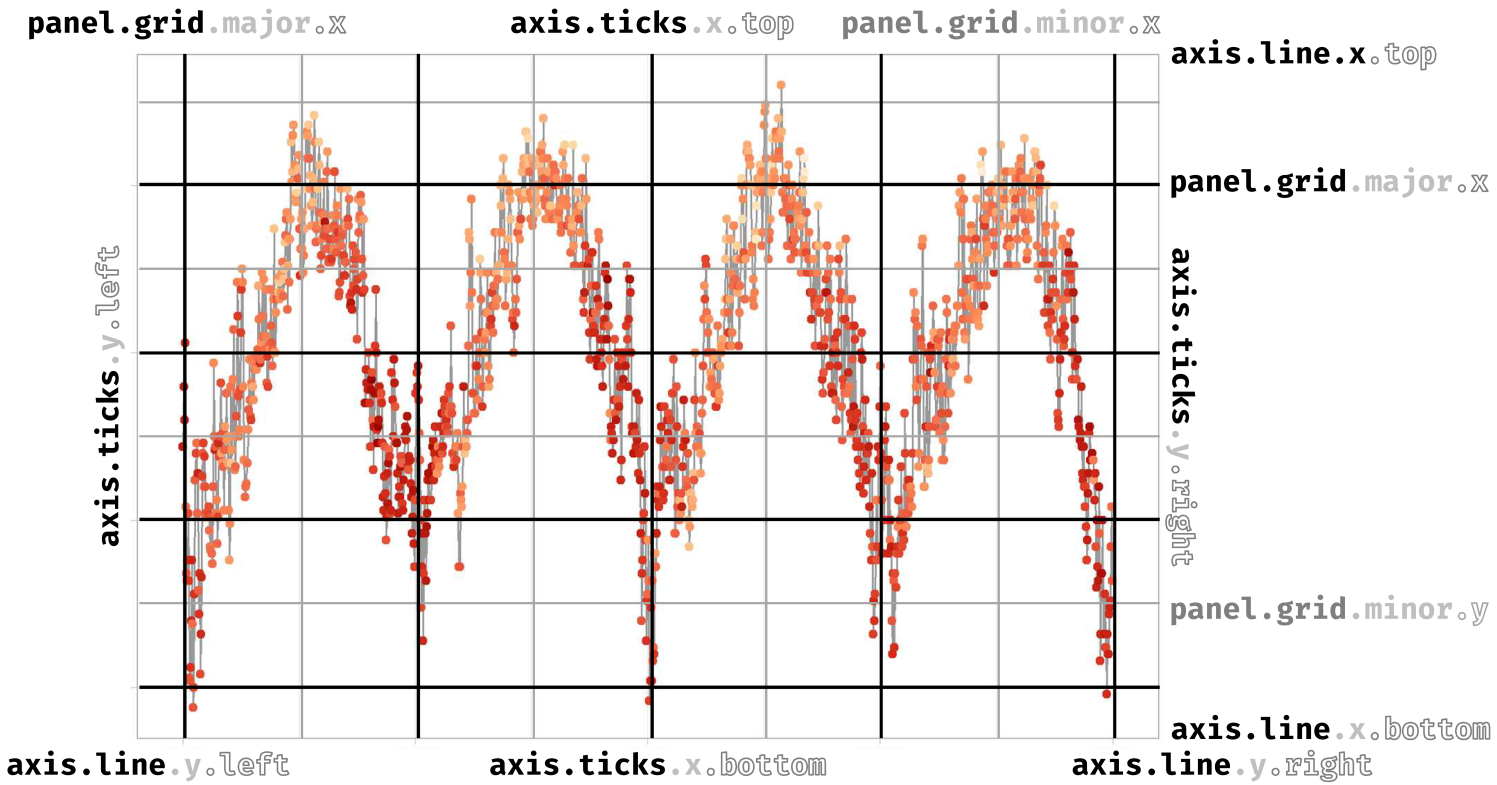
Text Elements via `element_text()`

You can directly alter the appearance by adding `theme()` to a ggplot:

```
g +
  ggtitle("My New\nTitle") +
  theme(
    plot.title = element_text(
      family = "Roboto",
      face = "bold",
      size = 18,
      color = "red",
      lineheight = .7,
      angle = 180,
      hjust = .5,
      vjust = .0,
      margin = margin(
        10,
        0,
        30,
        0
      )
    )
  )
```



Line Elements via `element_line()`



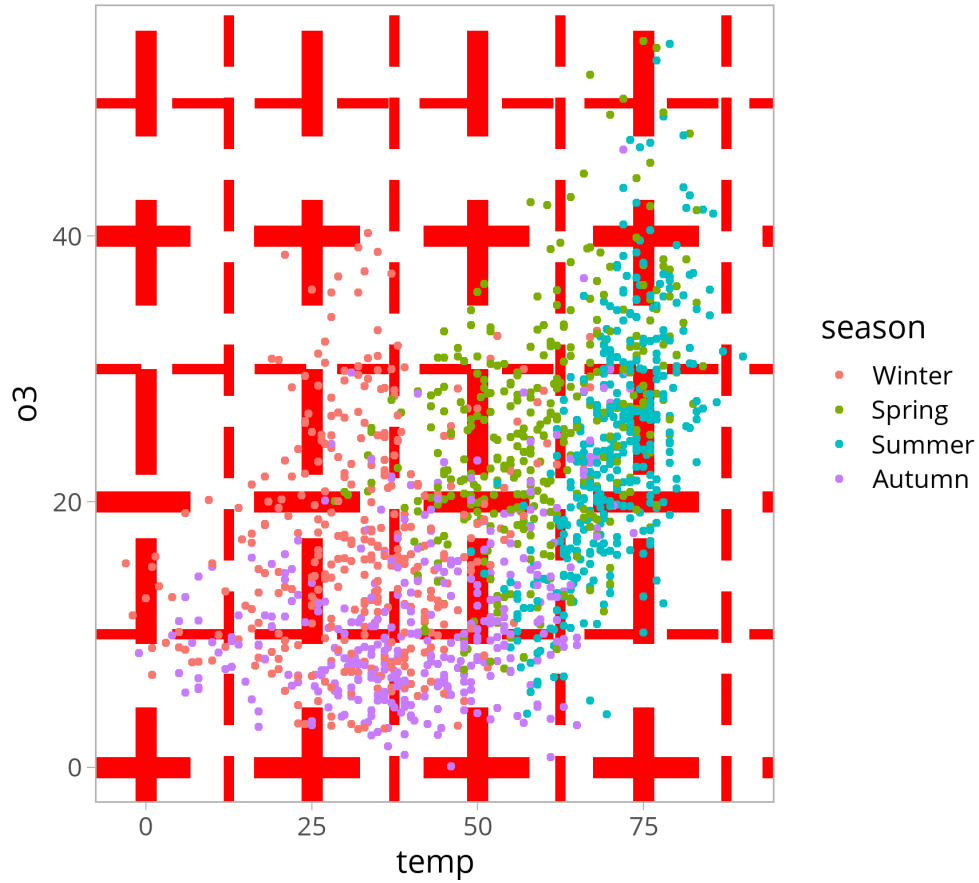
Line Elements via `element_line()`

```
element_line(
  color = "red",
  size = 10,
  linetype = "dashed",
  lineend = "square", # round, butt
  arrow = arrow(
    angle = 30,
    length = unit(0.25, "inches")
  )
)
```

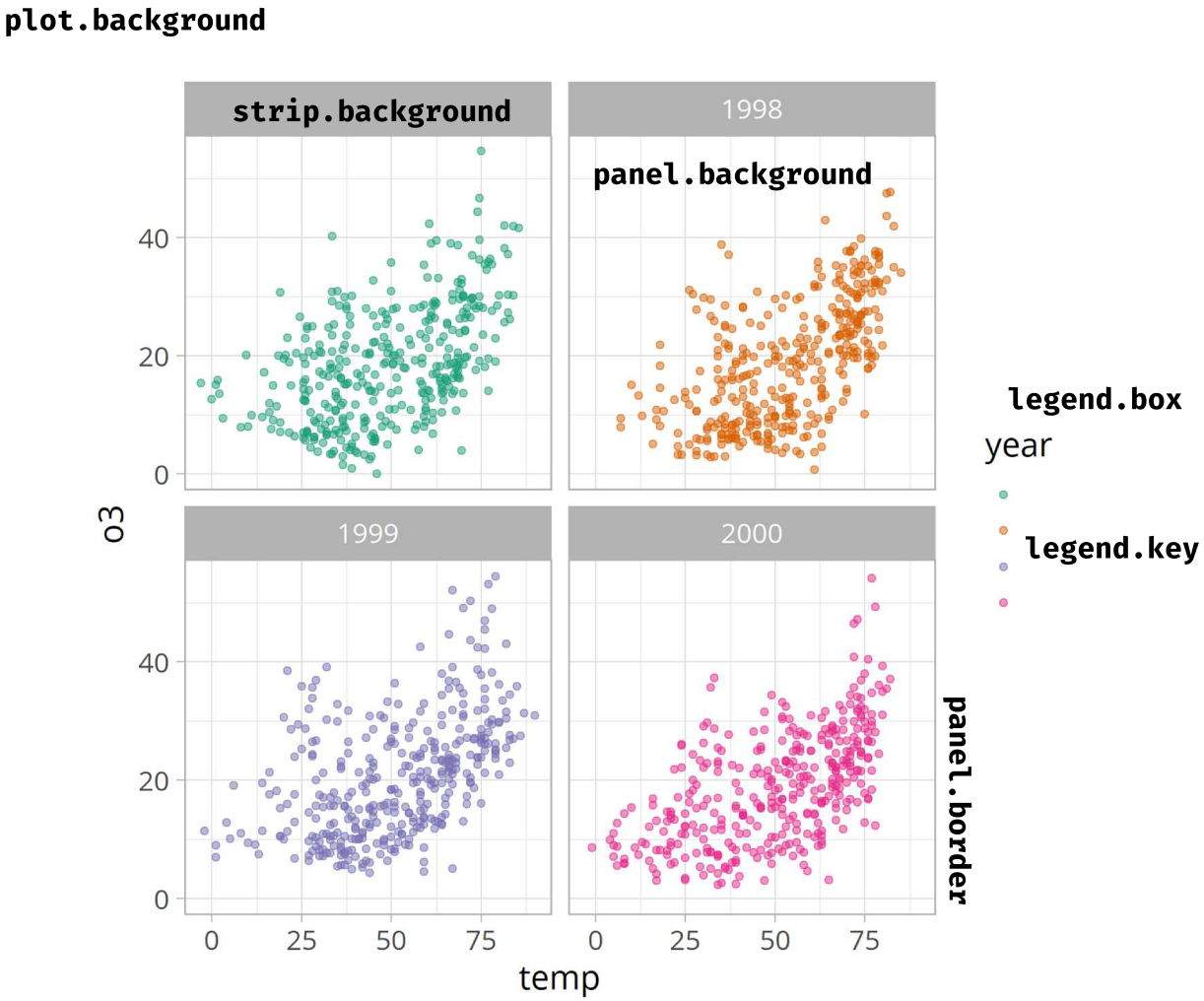
Line Elements via `element_line()`

You can directly alter the appearance by adding `theme()` to a ggplot:

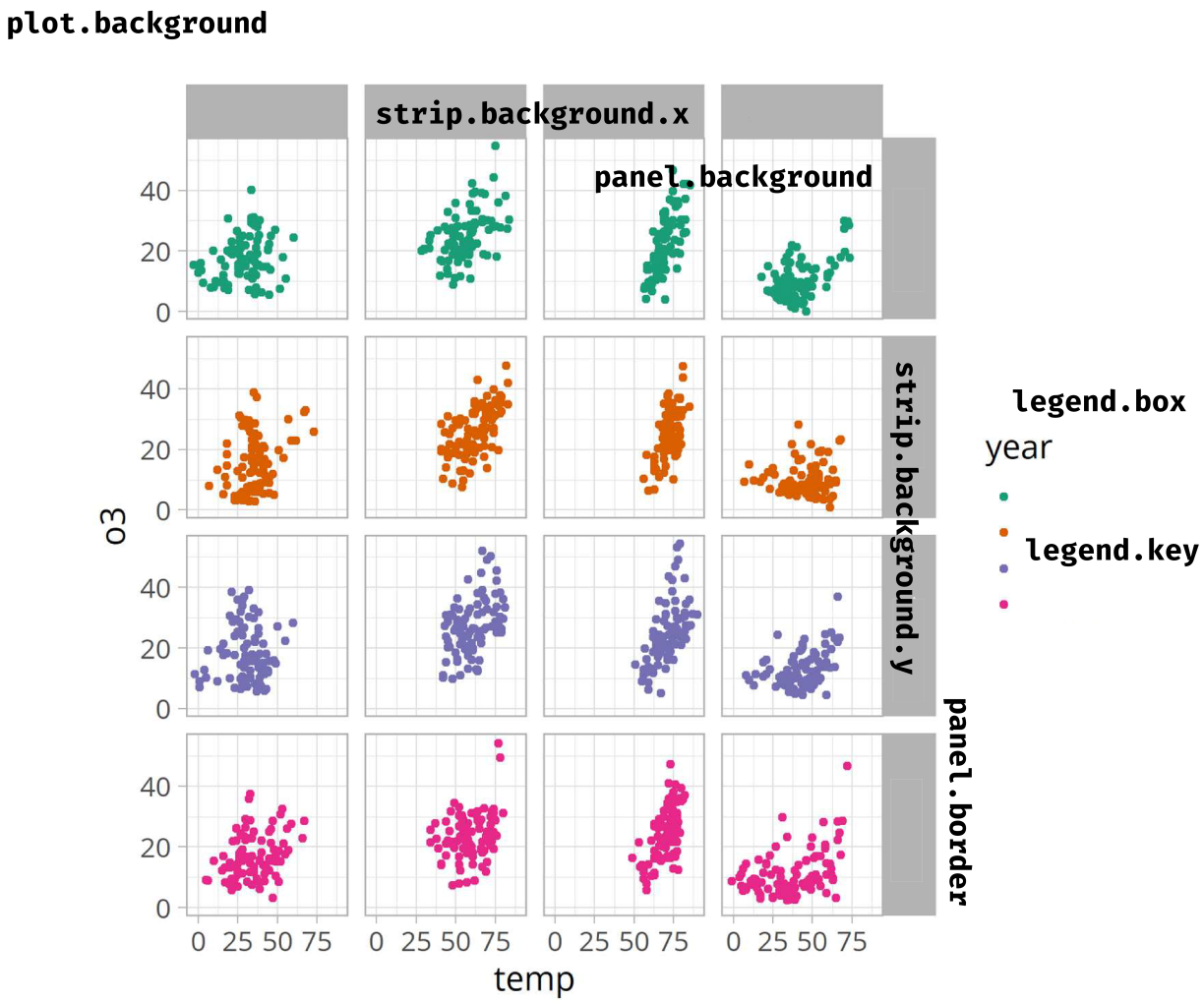
```
g +
  theme(
    panel.grid = element_line(
      color = "red",
      size = 10,
      linetype = "dashed",
      lineend = "square", # round, butt
      arrow = arrow(
        angle = 30,
        length = unit(0.25, "inches")
      )
    )
  )
```



Rectangular Elements via `element_rect()`



Rectangular Elements via `element_rect()`



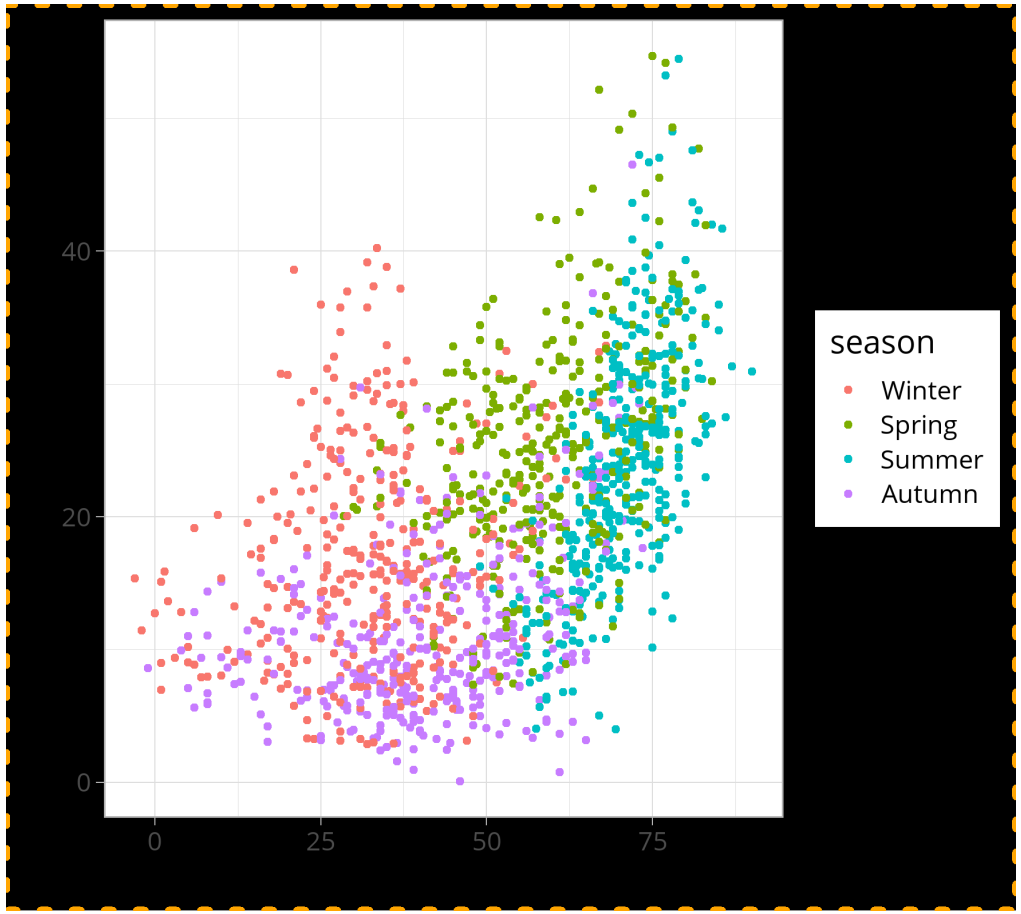
Rectangular Elements via `element_rect()`

```
element_rect(
  color = "orange",
  fill = "black",
  size = 2,
  linetype = "dotted"
)
```

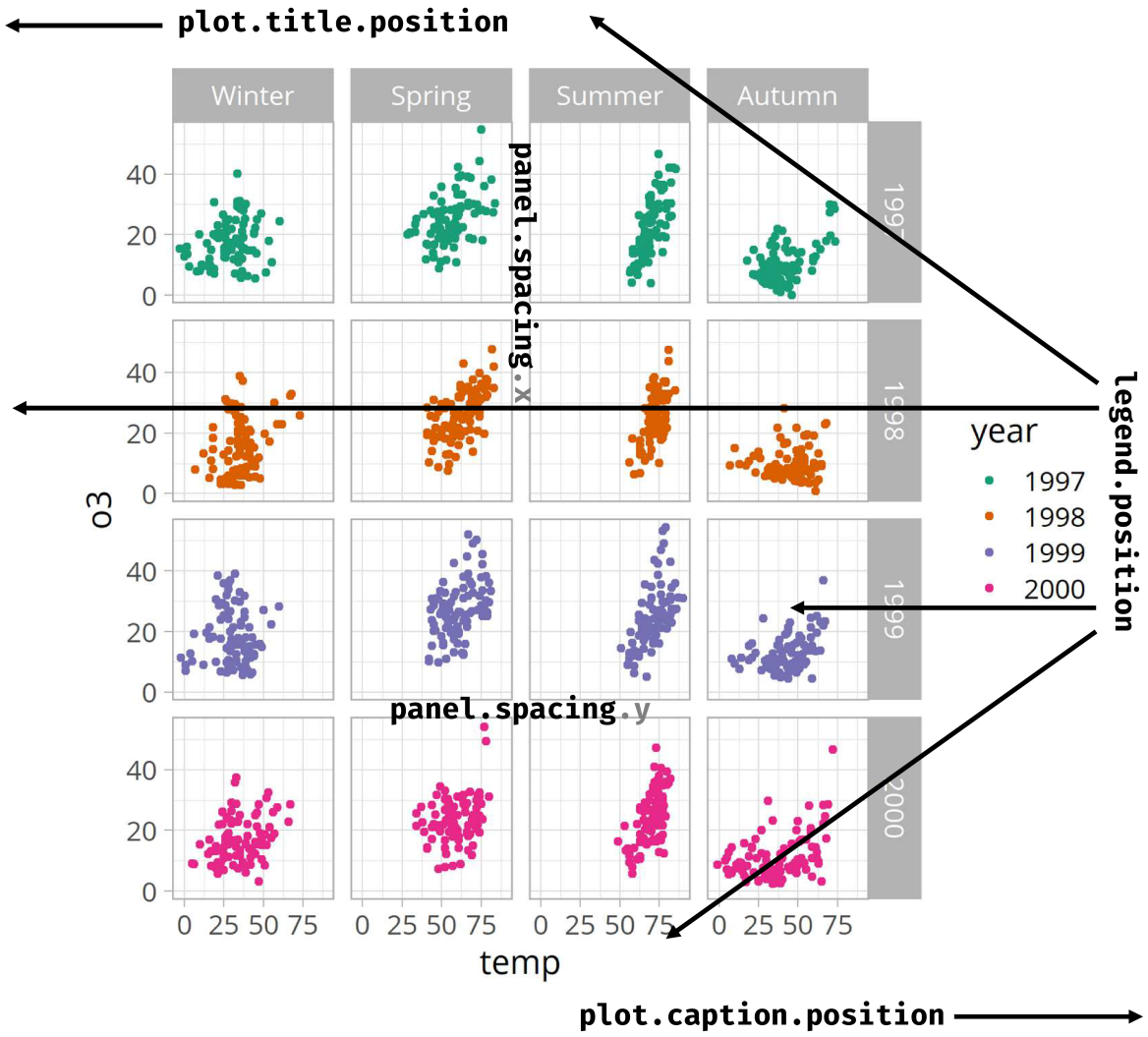
Rectangular Elements via `element_rect()`

You can directly alter the appearance by adding `theme()` to a ggplot:

```
g +
  theme(
    plot.background = element_rect(
      color = "orange",
      fill = "black",
      size = 2,
      linetype = "dotted"
    )
  )
```



Other Elements: Positions



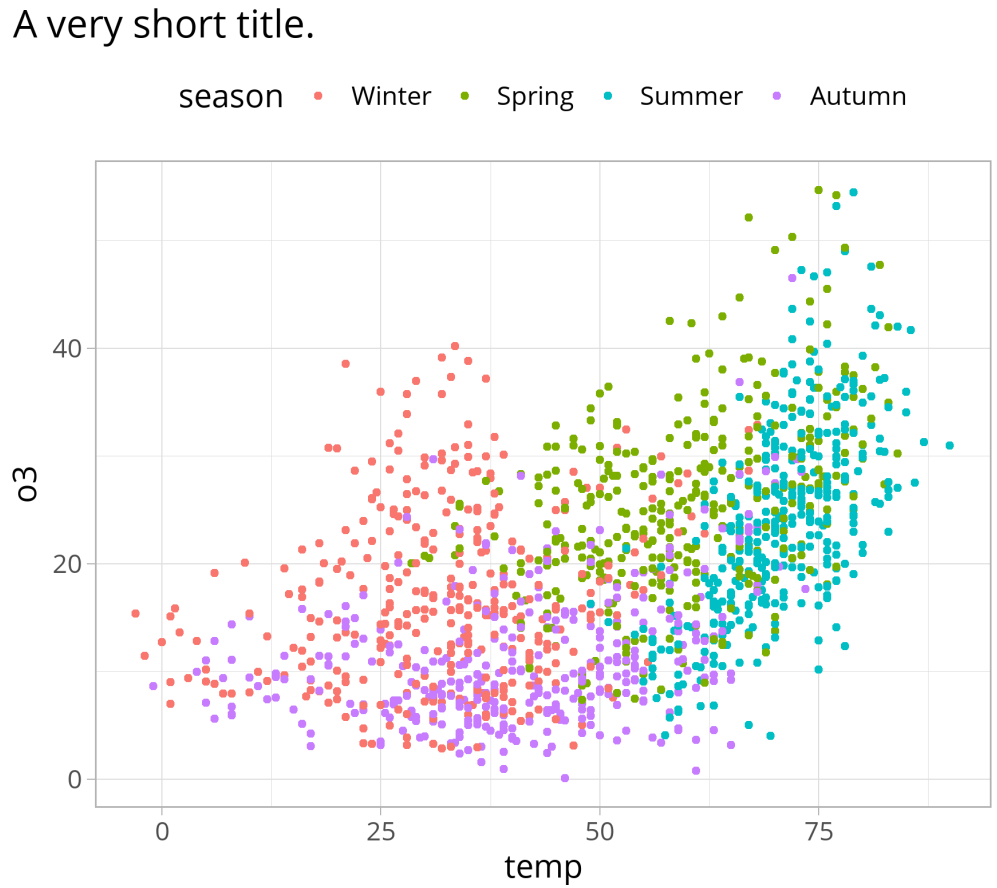
Other Elements: Positions

```
legend.position = "top" # bottom, right, left
legend.position = "none"
legend.position = c(.2, .8) # x, y

plot.title.position = "plot" # panel
plot.caption.position = "plot"
```

Other Elements: Positions

You can directly alter the appearance by adding `theme()` to a ggplot:

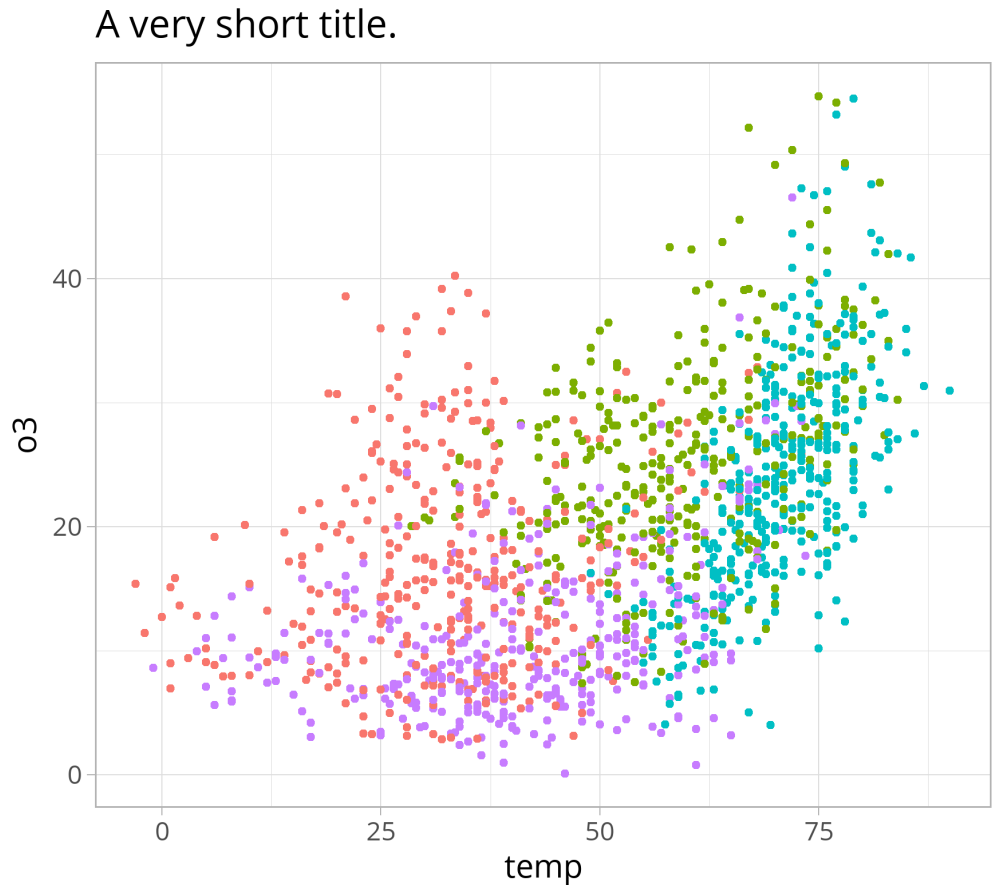
```
g +
  ggtitle("A very short title.") +
  theme(
    legend.position = "top",
    plot.title.position = "plot"
  )
```



Other Elements: Positions

You can directly alter the appearance by adding `theme()` to a ggplot:

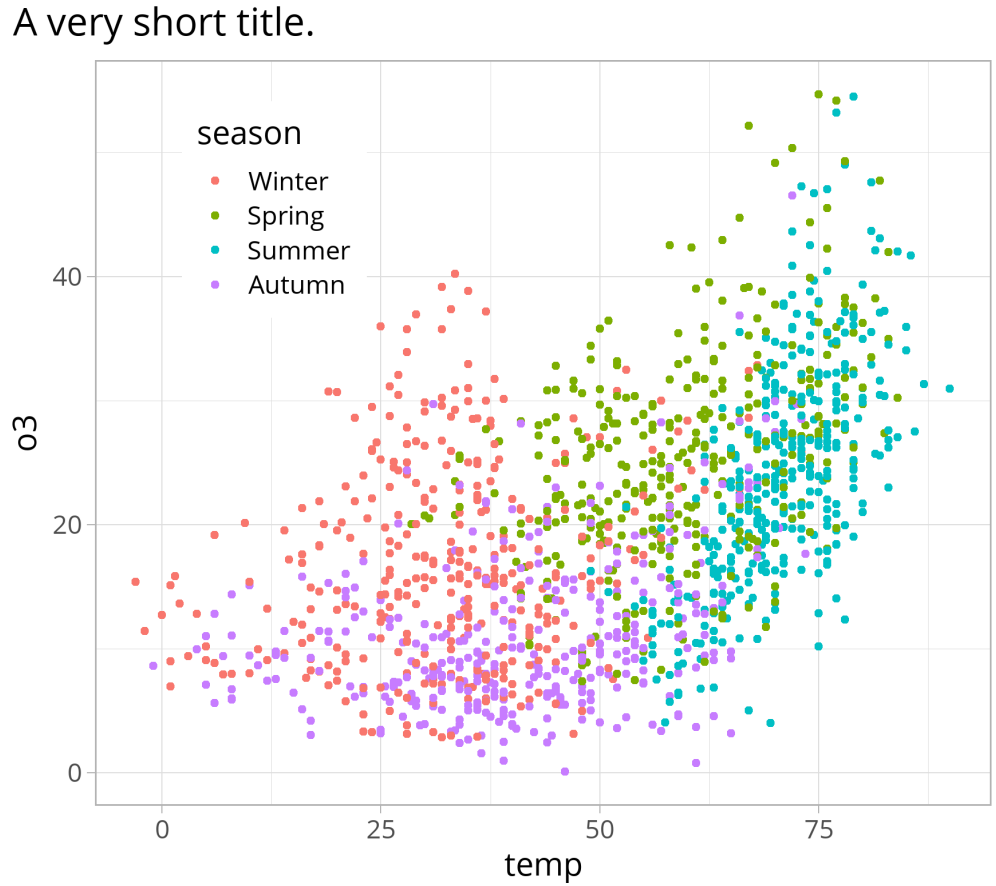
```
g +
  ggtitle("A very short title.") +
  theme(
    legend.position = "none",
    plot.title.position = "panel"
  )
```



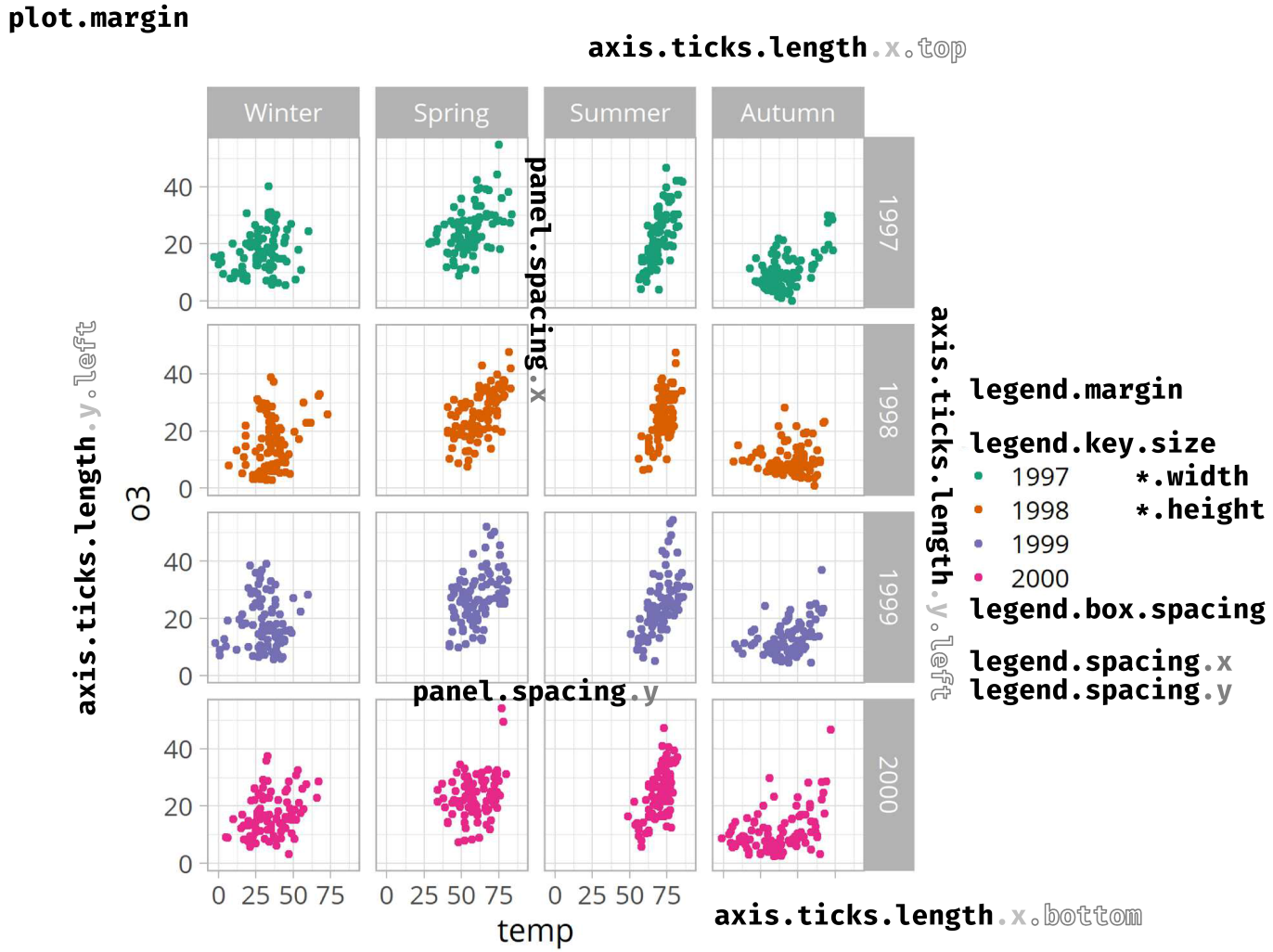
Other Elements: Positions

You can directly alter the appearance by adding `theme()` to a ggplot:

```
g +
  ggtitle("A very short title.") +
  theme(
    legend.position = c(.2, .8),
    plot.title.position = "plot"
  )
```



Other Elements: Spacing, Length, Size etc.



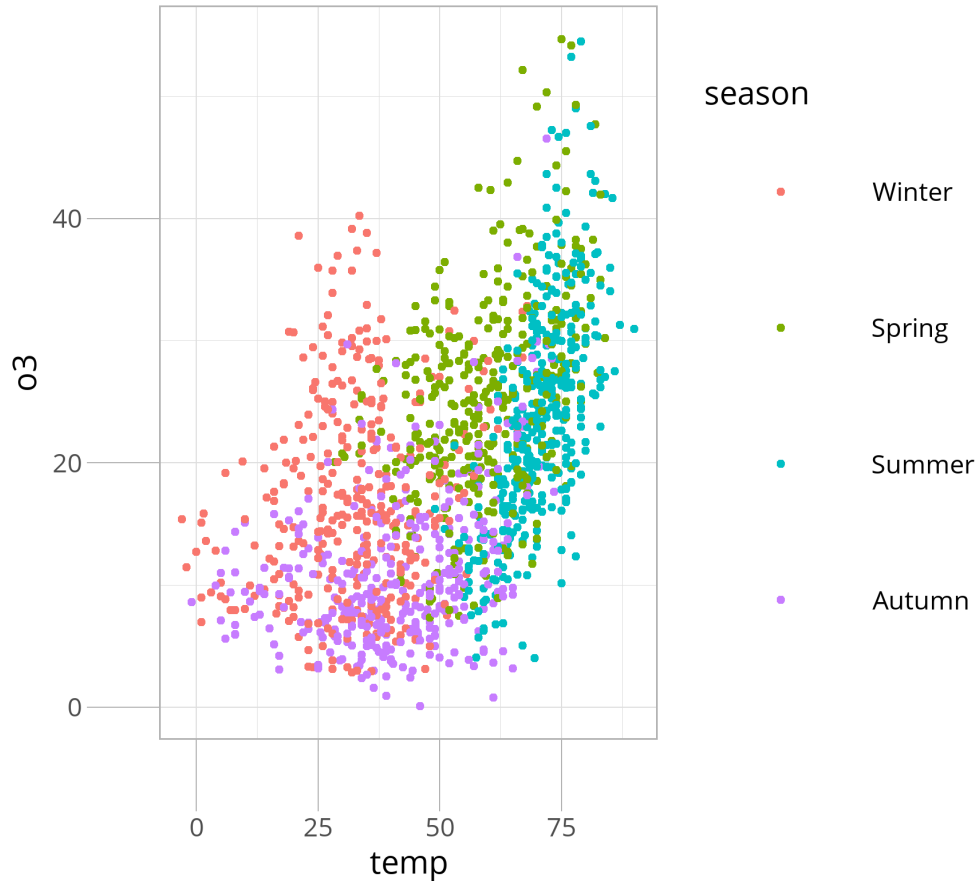
Other Elements: Spacing, Length, Size etc.

```
## ?grid::unit
unit(50, "pt")
unit(10, "mm")
unit(2.5, "lines")
unit(.15, "npc")  ## Normalised Parent Coordinates
```

Other Elements: Positions

You can directly alter the appearance by adding `theme()` to a ggplot:

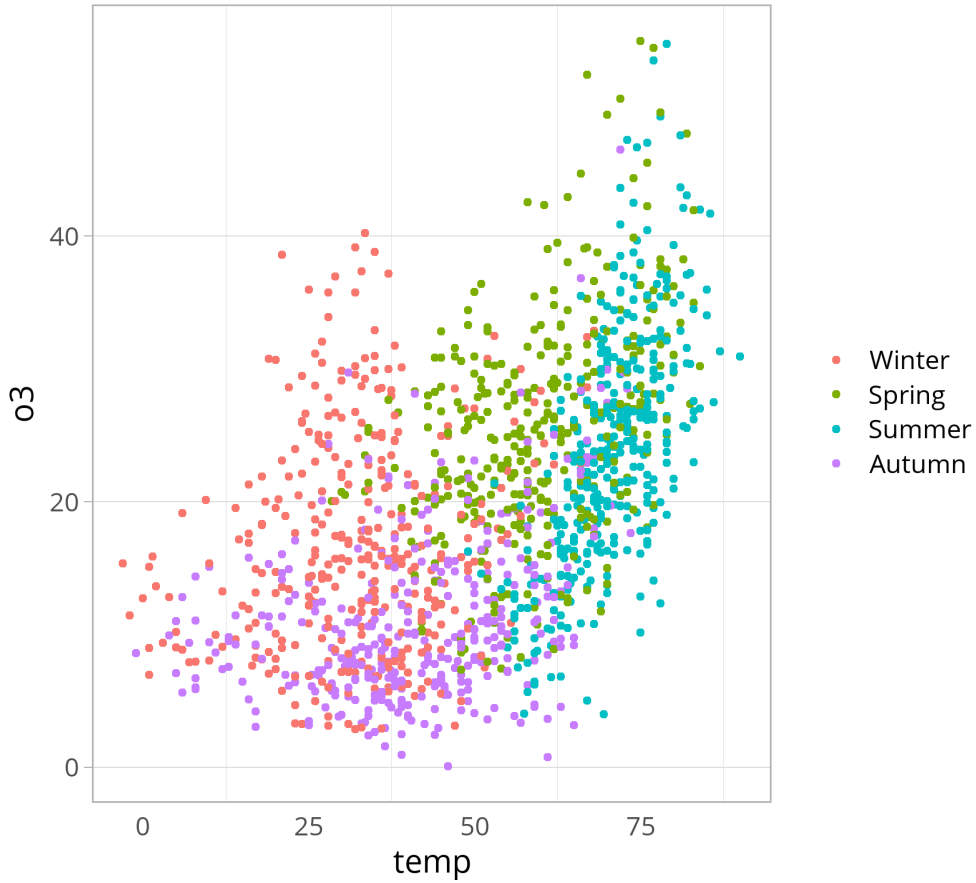
```
g +
  theme(
    axis.ticks.length =
      unit(2.5, "lines"),
    legend.key.size =
      unit(.15, "npc")
  )
```



Draw Nothing: `element_blank()`

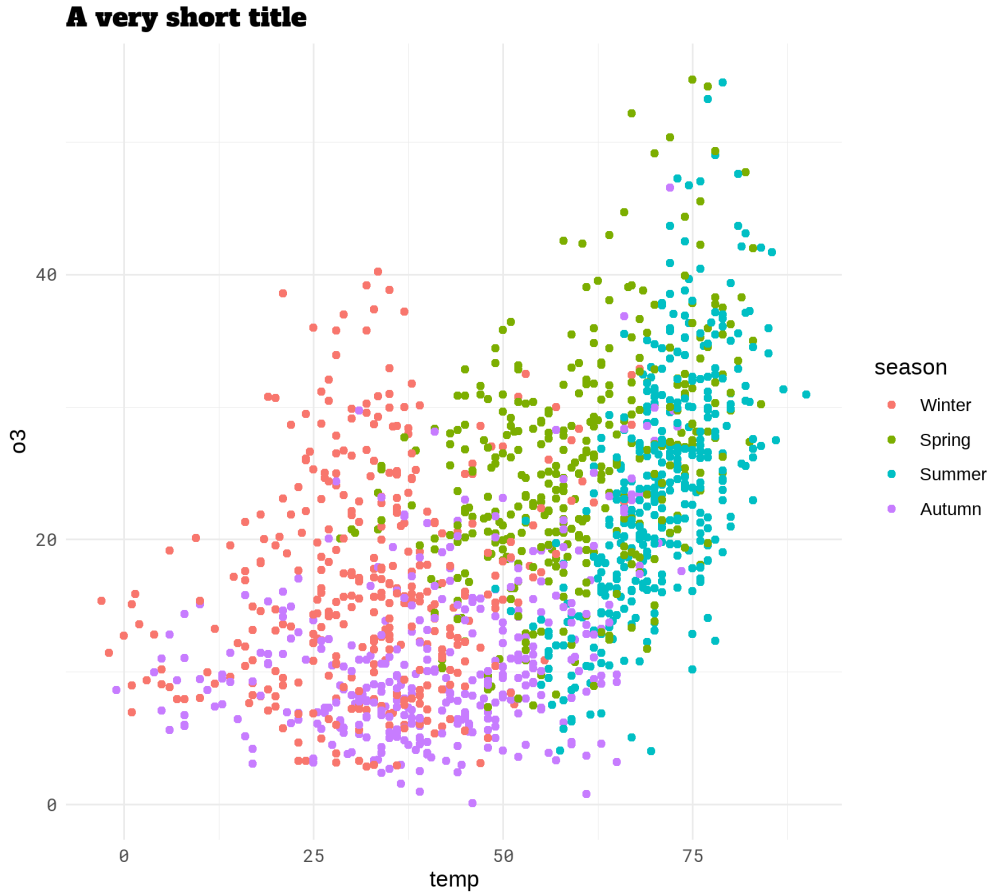
You can remove elements via `element_blank()`:

```
g +
  theme(
    axis.ticks.x = element_blank(),
    panel.grid.major.x = element_blank(),
    panel.grid.minor.y = element_blank(),
    legend.title = element_blank()
  )
```



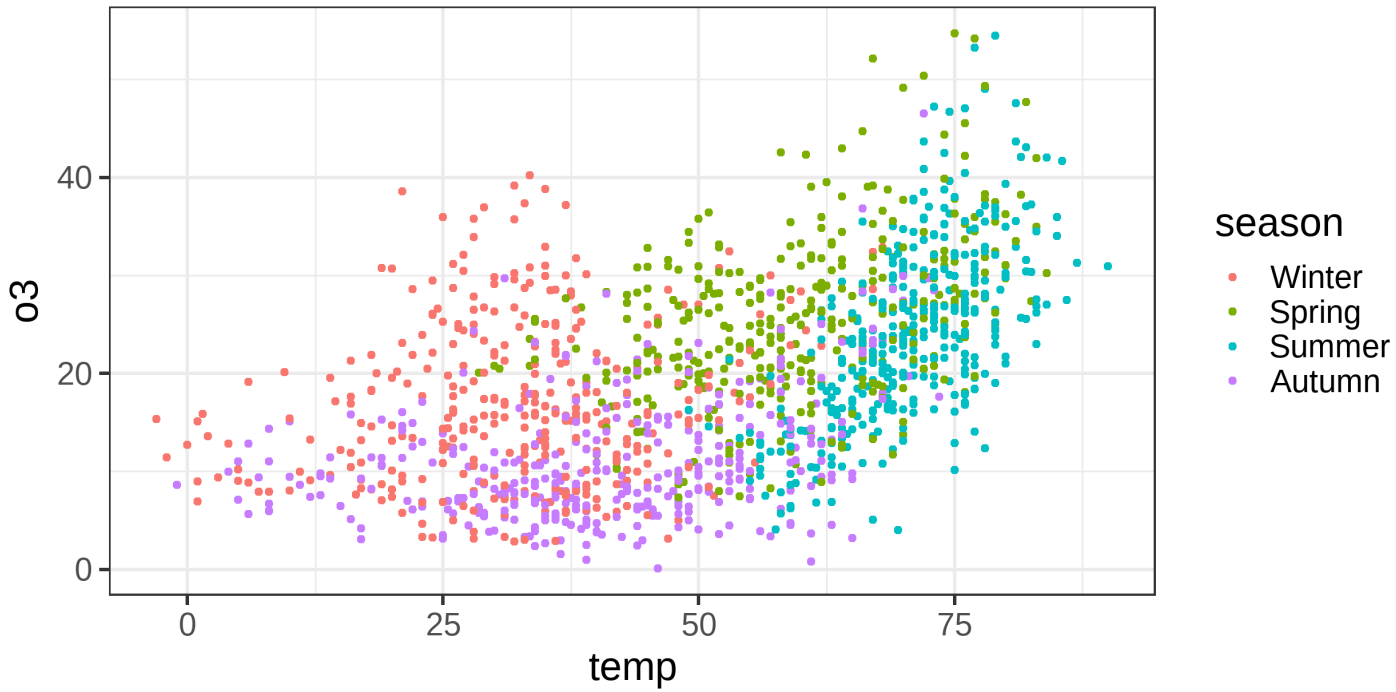
Changing Fonts of Your ggplot2

```
g +
  labs(
    title = "A very short title"
  ) +
  theme_minimal() +
  theme(
    plot.title =
      element_text(
        family = "Alfa Slab One"
      ),
    axis.text =
      element_text(
        family = "Roboto Mono"
      )
  )
```



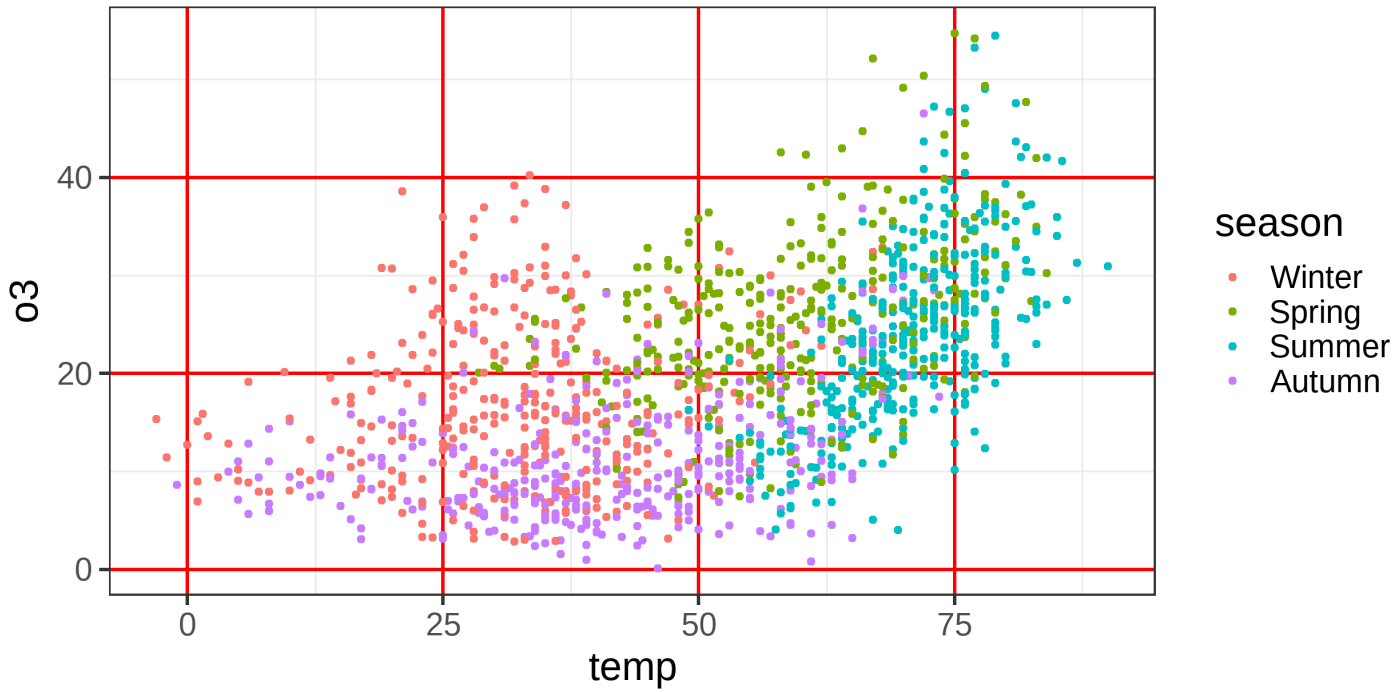
theme_update()

```
theme_set(theme_bw(base_size = 20))

g
```



theme_update()

```
theme_update(
  panel.grid.major = element_line(color = "red")
)

g
```



Your Turn!

- Have a closer look at the code of `theme_grey` and o the one you have chosen in lecture 3. What's the difference?
- Create a plot and change the theme to be as ugly as possible!
- If you like, build your own theme by editing `theme_grey` or updating one of the others. You could also search for other themes provided by extension packages.