An Introduction to {ggplot2}

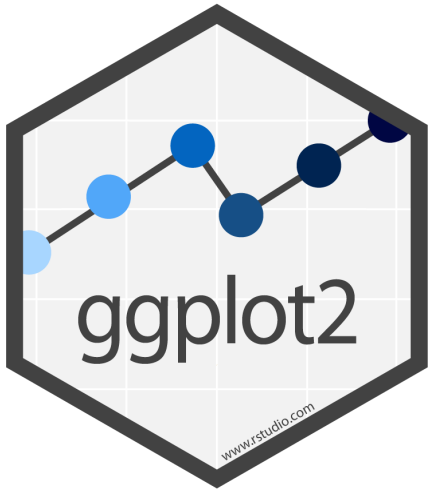
A Graphics Framework for Elegant Plotting in R

Cédric Scherer

Leibniz Institute for Zoo and Wildlife Research Berlin

IZW Stats Group | 28th of August 2019

Image by Richard Strozynski



{ggplot2} is a system for declaratively creating graphics, based on "The Grammar of Graphics" (Wilkinson, 2005).

You provide the data, tell **{ggplot2}** how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

ggplot2 package description

Advantages of `{ggplot2}`

- consistent underlying grammar of graphics (Wilkinson, 2005)
- very flexible, layered plot specification
- theme system for polishing plot appearance
- active and helpful community
  (e.g. community.rstudio.com, R4DS Learning Community, Twitter)

The {ggplot2} Showcase

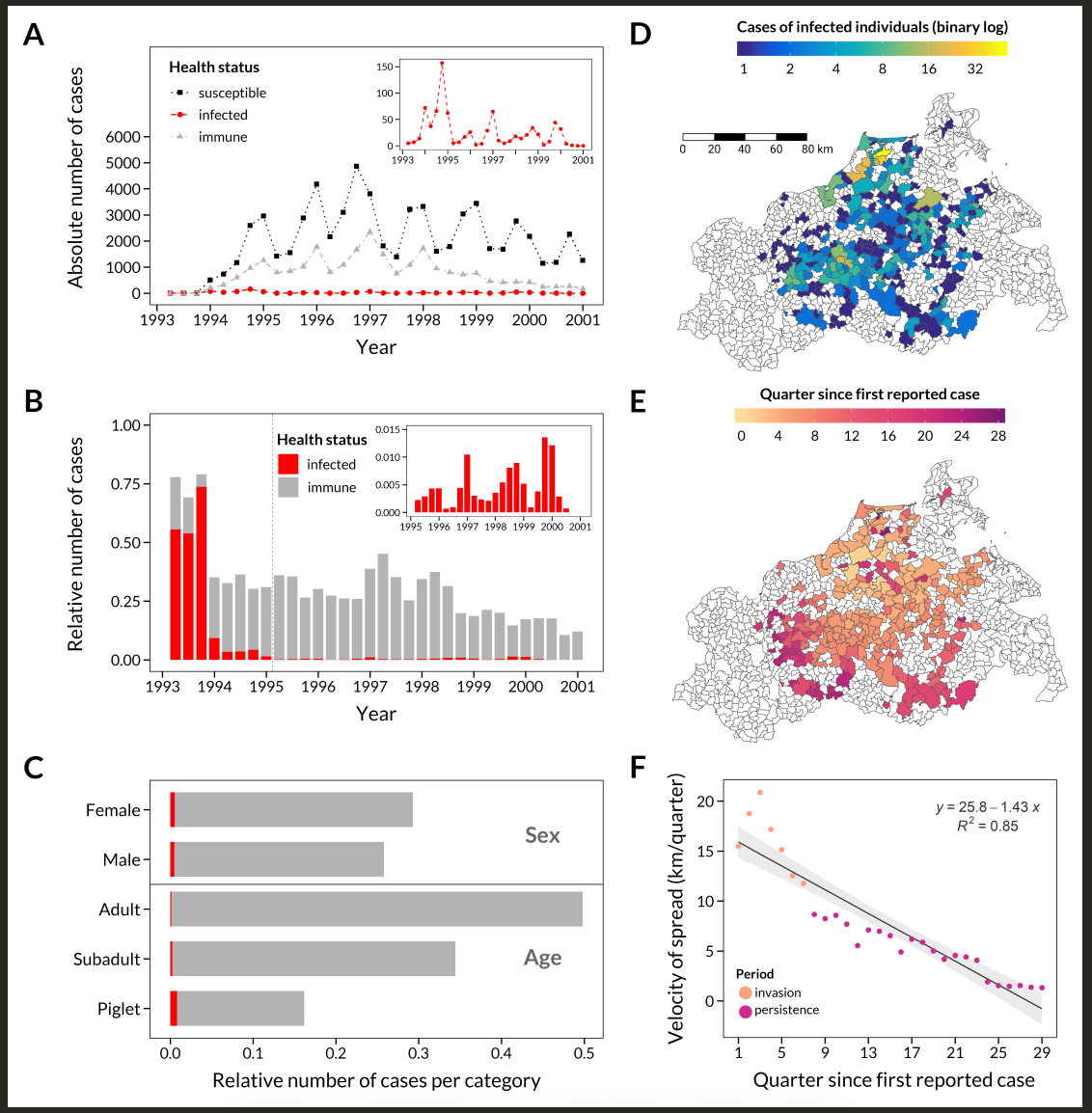
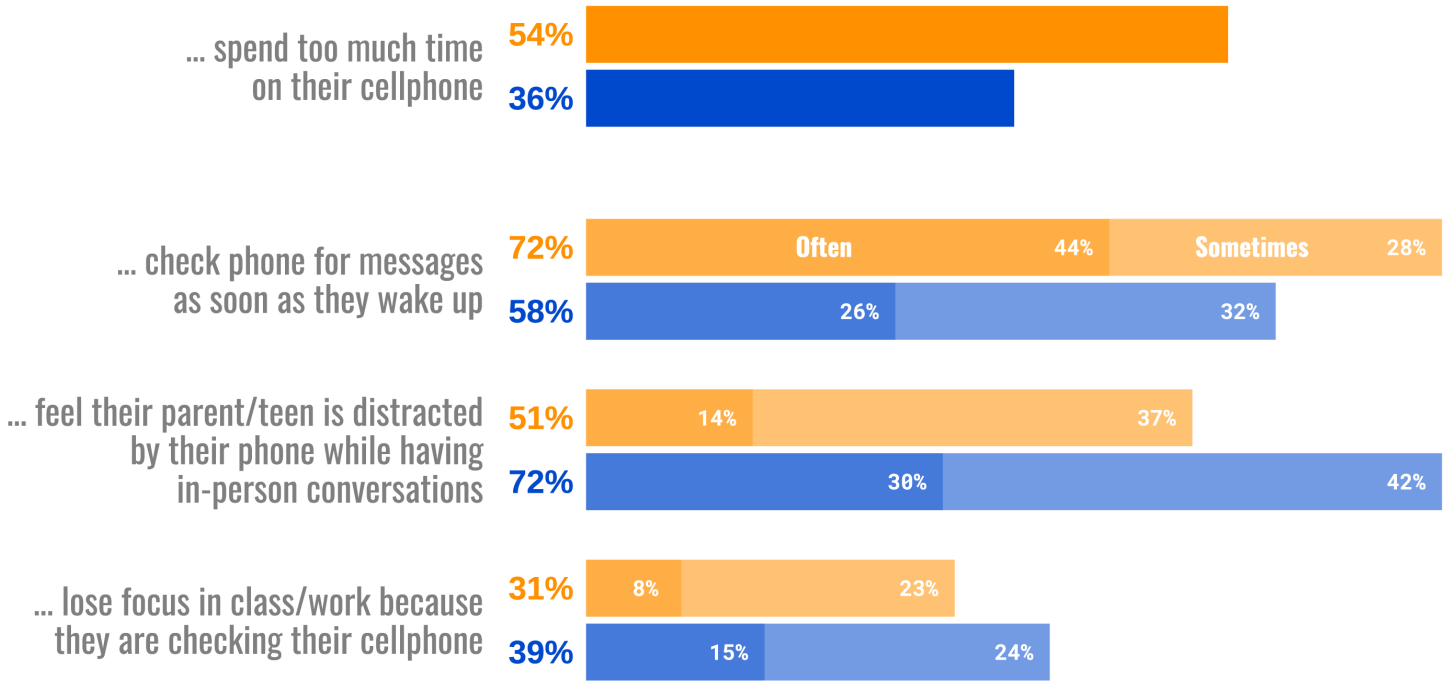


Figure in Scherer et al. 2019 J. Anim. Ecol.

Parents and teens report varying levels of attachment and distraction due to their cellphones

Percent of U.S. **teens** and **parents** who say they...



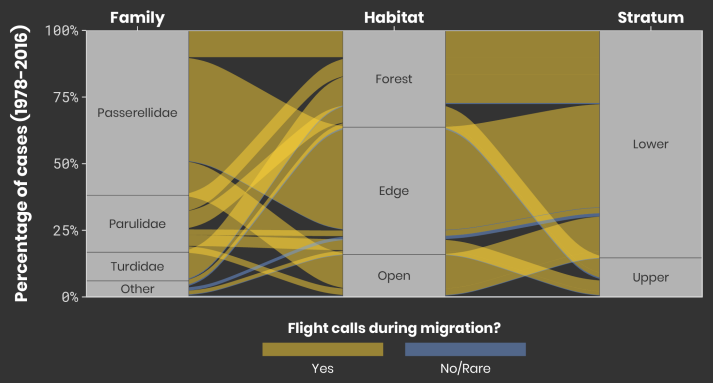
Source: "How Parents and Teens Navigate Screen Time and Device Distraction" by Pew Research Center
Survey conducted March 7-April 10, 2018, with 1058 parents of teens and 743 teens (teens are between 13 and 17 years old)

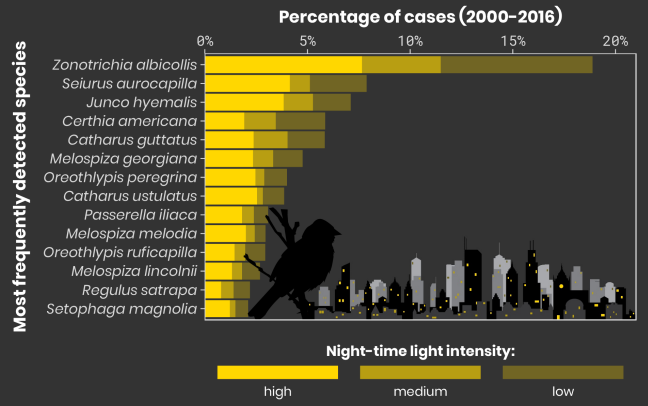
DataViz by @CedScherer

Contribution to #MakeoverMonday

What drives collision risk in nocturnally migrating passerine birds?

Nocturnal bird collisions at McCormick Place, Chicago, IL

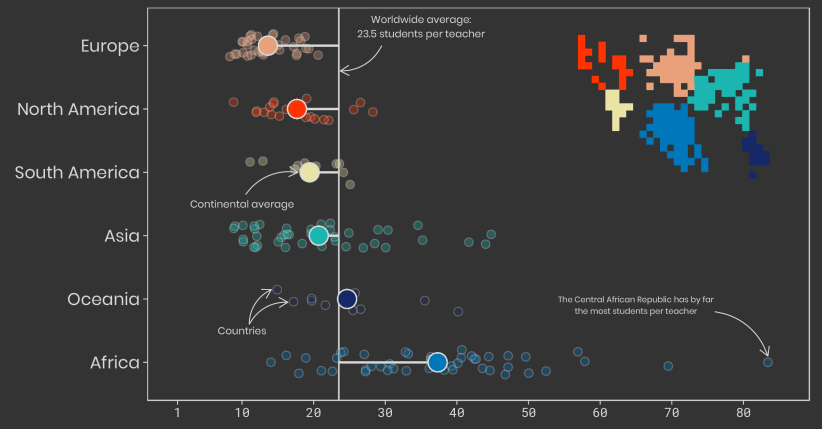


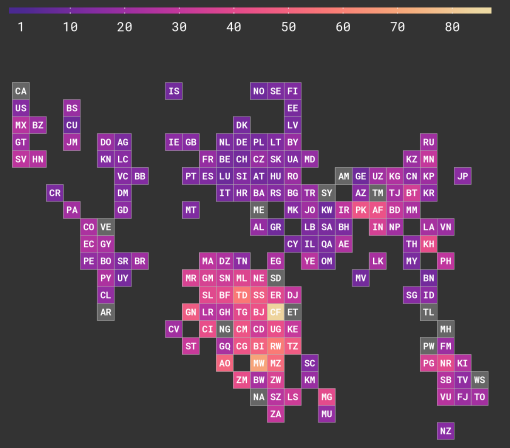


Visualization by @CedScherer | Data: Winger et al. 2019 (doi: 10.1098/rspb.2019.0364)

Global student to teacher ratios in primary education

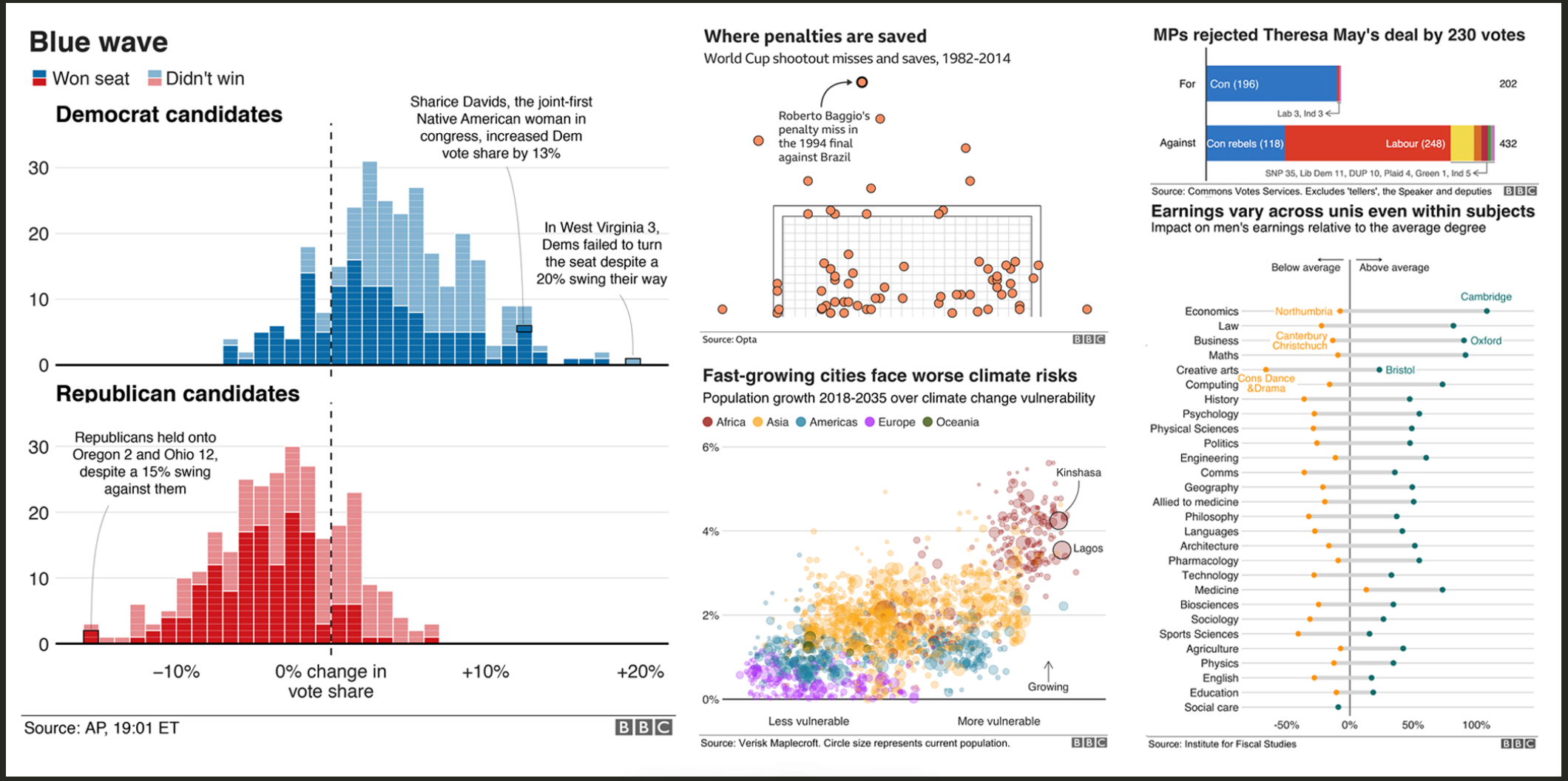
Latest reported student to teacher ratio per country and continent (2012–2018)





Visualization by @CedScherer | Data: "eAtlas of Teachers" by UNESCO

Contribution to #TidyTuesday



Collection of BBC Graphics

(adapted from bbc.github.io/rcookbook)

The Setup

The package

{ggplot2} is a **data visualization package** for the statistical programming language **R** created by Hadley Wickham in 2005.

```
install.packages("ggplot2")
library(ggplot2)
```

{ggplot2} is part of the **{tidyverse}**, a set of packages that work in harmony to manipulate and explore data.

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

{tidyverse} contains **{ggplot2}**, **{dplyr}**, **{tidyr}**, **{readr}**, **{purrr}**, **{tibble}** and some more packages.

Source: www.dicook.org/files/rstudio/#3

The data

We use data from the *National Morbidity and Mortality Air Pollution Study* (NMMAPS), filtered for the city of Chicago and the timespan January 1997 to December 2000.

```
chic <- readr::read_csv("https://raw.githubusercontent.com/Z3tt/ggplot-courses/master/
chic$year <- factor(chic$year, levels = as.character(1997:2000))
tibble::glimpse(chic)
```

```
## Observations: 1,461
## Variables: 10
## $ city     <chr> "chic", "chic", "chic", "chic", "chic", "chic", "chic...
## $ date     <date> 1997-01-01, 1997-01-02, 1997-01-03, 1997-01-04, 1997...
## $ death    <dbl> 137, 123, 127, 146, 102, 127, 116, 118, 148, 121, 110...
## $ temp     <dbl> 36.0, 45.0, 40.0, 51.5, 27.0, 17.0, 16.0, 19.0, 26.0,...
## $ dewpoint <dbl> 37.500, 47.250, 38.000, 45.500, 11.250, 5.750, 7.000,...
## $ pm10     <dbl> 13.052268, 41.948600, 27.041751, 25.072573, 15.343121...
## $ o3       <dbl> 5.659256, 5.525417, 6.288548, 7.537758, 20.760798, 14...
## $ time     <dbl> 3654, 3655, 3656, 3657, 3658, 3659, 3660, 3661, 3662,...
## $ season   <fct> Winter, Winter, Winter, Winter, Winter, Winter, Winte...
## $ year     <fct> 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997, 1997,...
```

The Structure of {ggplot2}

"The Grammar of Graphics"

The Structure of `{ggplot2}`

1. **Data** → The raw data that you want to visualise
2. **Layers `geom_` and `stat_`** → The geometric shapes and statistical summaries representing the data
3. **Aesthetics `aes()`** → Aesthetic mappings of the geometric and statistical objects
4. **Scales `scale_`** → Maps between the data and the aesthetic dimensions
5. **Coordinate system `coord_`** → Maps data into the plane of the data rectangle
6. **Facets `facet_`** → The arrangement of the data into a grid of plots
7. **Visual themes `theme() and theme_`** → The overall visual defaults of a plot

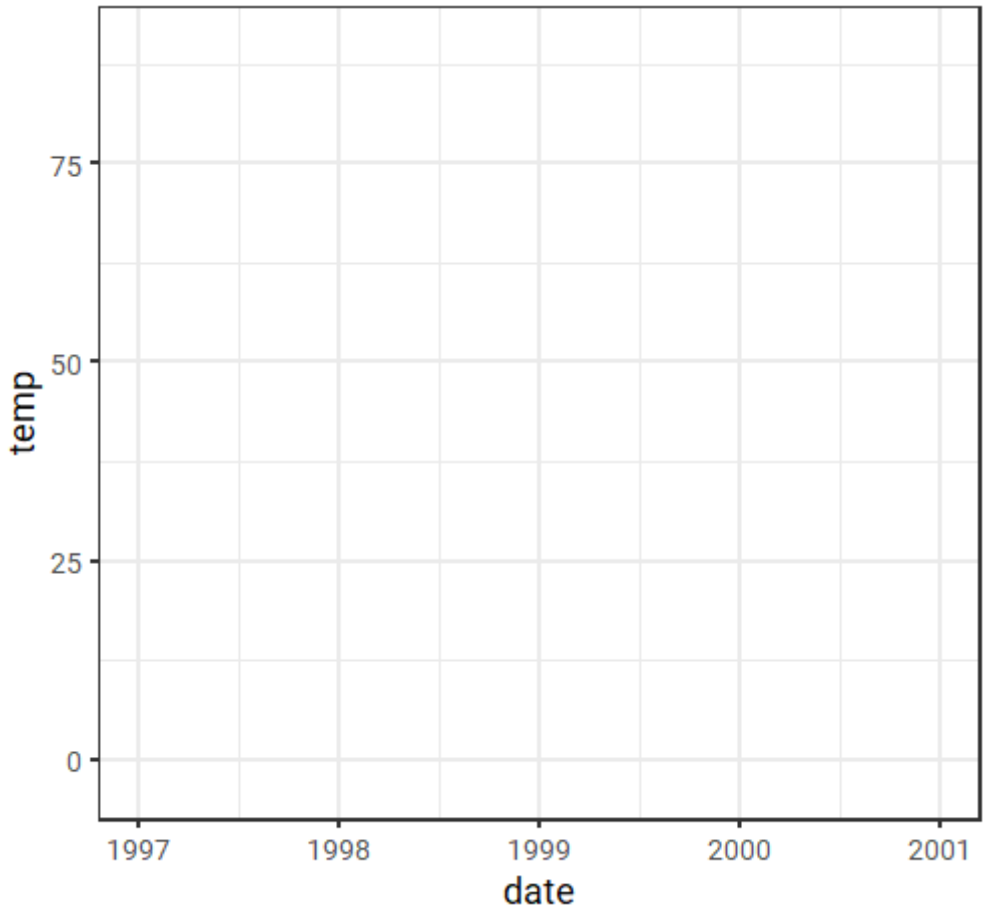
1. Data

```
ggplot(data, aes(x, y))
```

1. Data: `ggplot()`

We need to specify data and the two variables we want to plot as `aes`thetics of the `ggplot()` call:

```
ggplot(data = chic,
       mapping =
         aes(
          x = date,
          y = temp
         )
      )
```



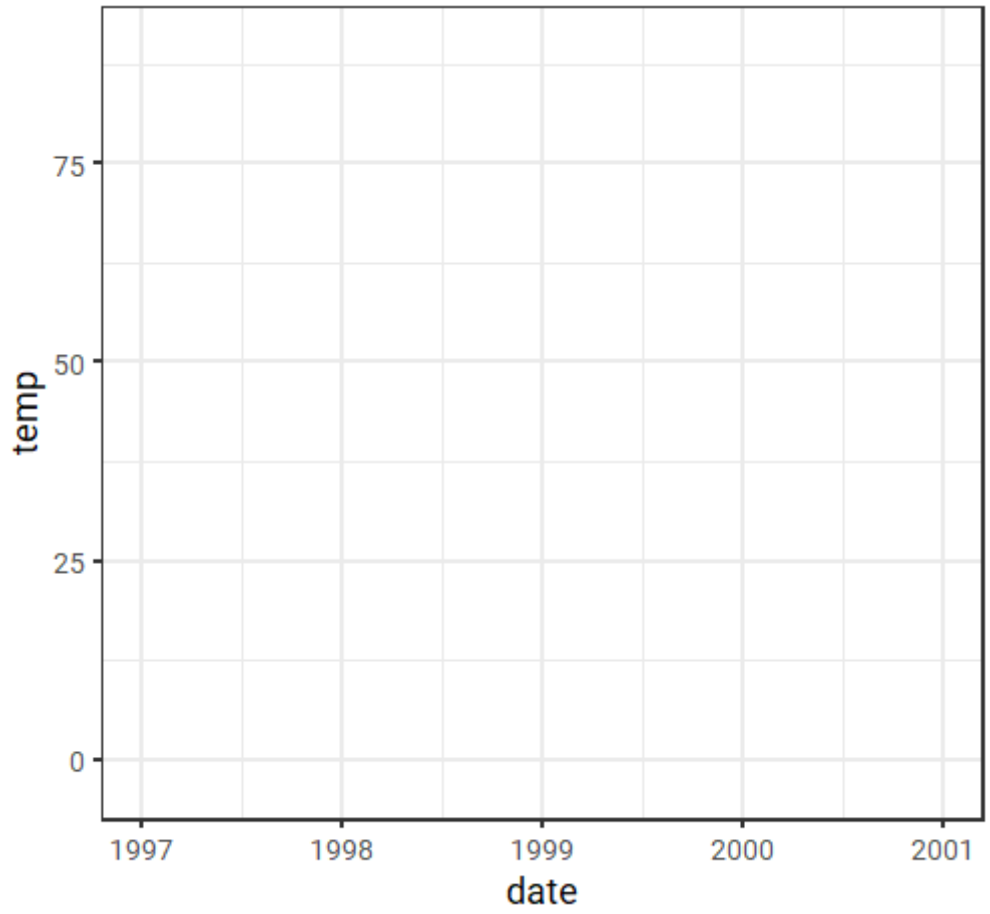
There is only an empty panel because `{ggplot2}` **doesn't know *how* it should plot the data**.

1. Data: `ggplot()`

We need to specify data and the two variables we want to plot as `aes`thetics of the `ggplot()` call:

Since almost every `ggplot()` takes the same arguments (`data, mapping = aes(x, y)`), we can also write:

```
ggplot(chic, aes(date, temp))
```



1. Data: `ggplot()`

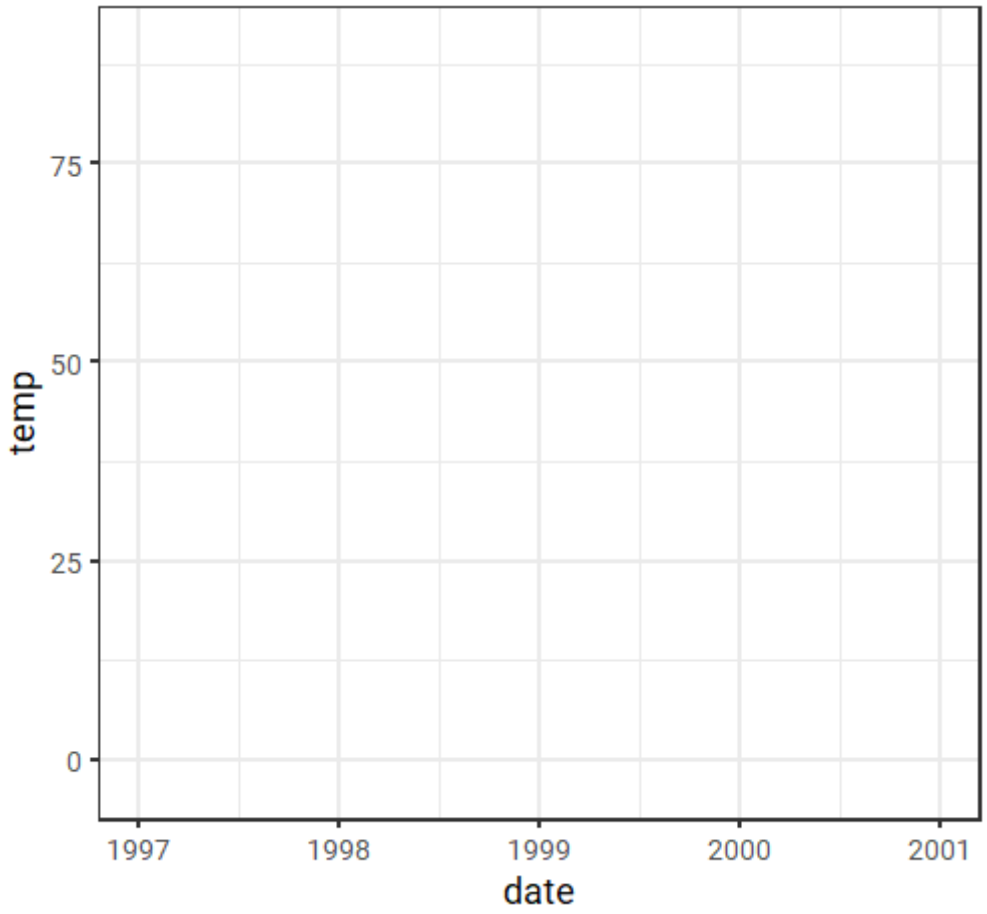
We need to specify data and the two variables we want to plot as `aes`thetics of the `ggplot()` call:

Since almost every `ggplot()` takes the same arguments (`data, mapping = aes(x, y)`), we can also write:

```
ggplot(chic, aes(date, temp))
```

... or add the `aes`thetics outside the `ggplot` function:

```
ggplot(chic) +
  aes(date, temp)
```



2. Layers

`geom_*()` and `stat_*()`

2. Layers: `geom_*()` and `stat_*()`

By adding one or multiple layers we can tell `{ggplot2}` *how* to represent the data.
There are lots of build-in geometrics elements (`geoms`) and statistical transformations (`stats`):

Layer: geoms

- `geom_abline()` `geom_hline()` `geom_vline()`
- `geom_bar()` `geom_col()` `stat_count()`
- `geom_bin2d()` `stat_bin_2d()`
- `geom_blank()`
- `geom_boxplot()` `stat_boxplot()`
- `geom_contour()` `stat_contour()`
- `geom_count()` `stat_sum()`
- `geom_density()` `stat_density()`
- `geom_density_2d()` `stat_density_2d()`
- `geom_dotplot()`
- `geom_errorbarh()`
- `geom_hex()` `stat_bin_hex()`
- `geom_freqpoly()` `geom_histogram()` `stat_bin()`
- `geom_jitter()`
- `geom_crossbar()` `geom_errorbar()` `geom_linerange()` `geom_pointrange()`
- `geom_map()`
- `geom_path()` `geom_line()` `geom_step()`
- `geom_point()`
- `geom_polygon()`
- `geom_qq_line()` `stat_qq_line()` `geom_qq()` `stat_qq()`
- `geom_quantile()` `stat_quantile()`
- `geom_ribbon()` `geom_area()`
- `geom_rug()`
- `geom_segment()` `geom_curve()`
- `geom_smooth()` `stat_smooth()`
- `geom_spoke()`
- `geom_label()` `geom_text()`
- `geom_raster()` `geom_rect()` `geom_tile()`
- `geom_violin()` `stat_ydensity()`
- `coord_sf()` `geom_sf()` `geom_sf_label()` `geom_sf_text()` `stat_sf()`

Layer: stats

- `stat_ecdf()`
- `stat_ellipse()`
- `stat_function()`
- `stat_identity()`
- `stat_summary_2d()` `stat_summary_hex()`
- `stat_summary_bin()` `stat_summary()`
- `stat_unique()`
- `stat_sf_coordinates()`
- `stat()`

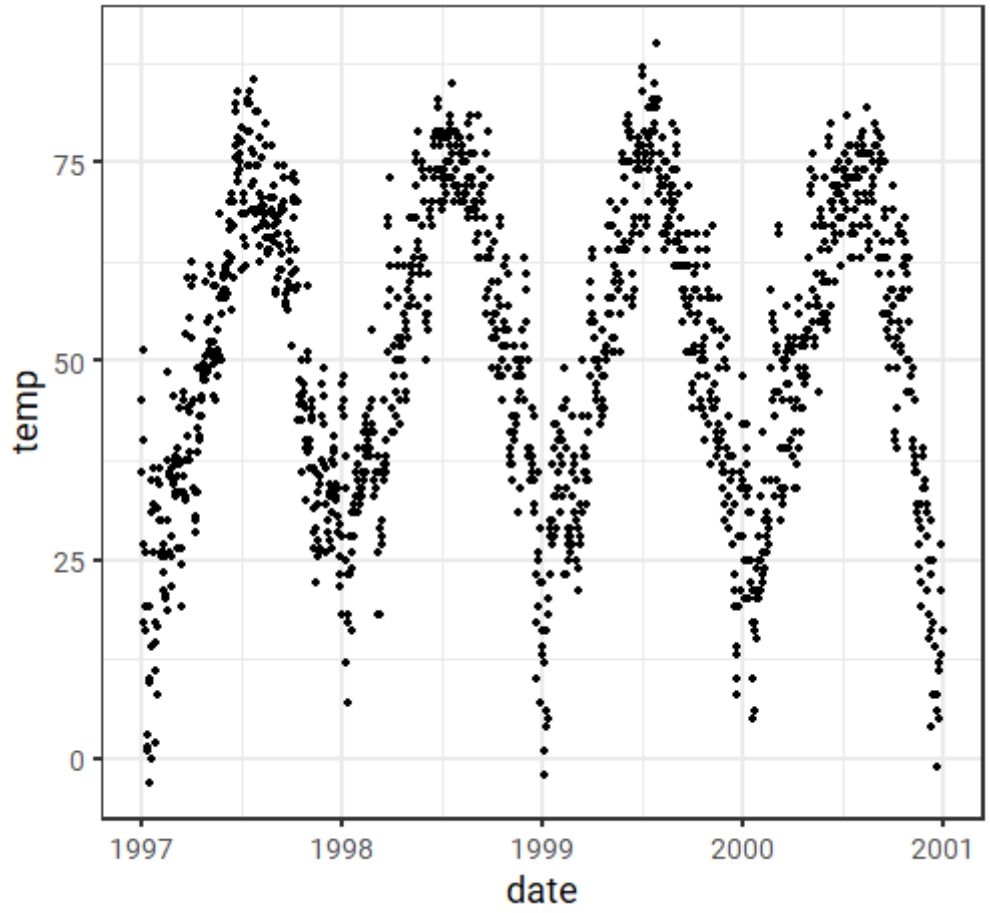
Adapted from https://ggplot2.tidyverse.org/reference/

... and several more in extension packages, e.g. `ggforce`, `ggalt`, `ggridges`, `ggrepel`, `ggcorrplot`, `ggraph` and `ggdendro`.

2. Layers — Geometries: `geom_point()`

We can tell `{ggplot2}` to represent the data for example as a **scatterplot**:

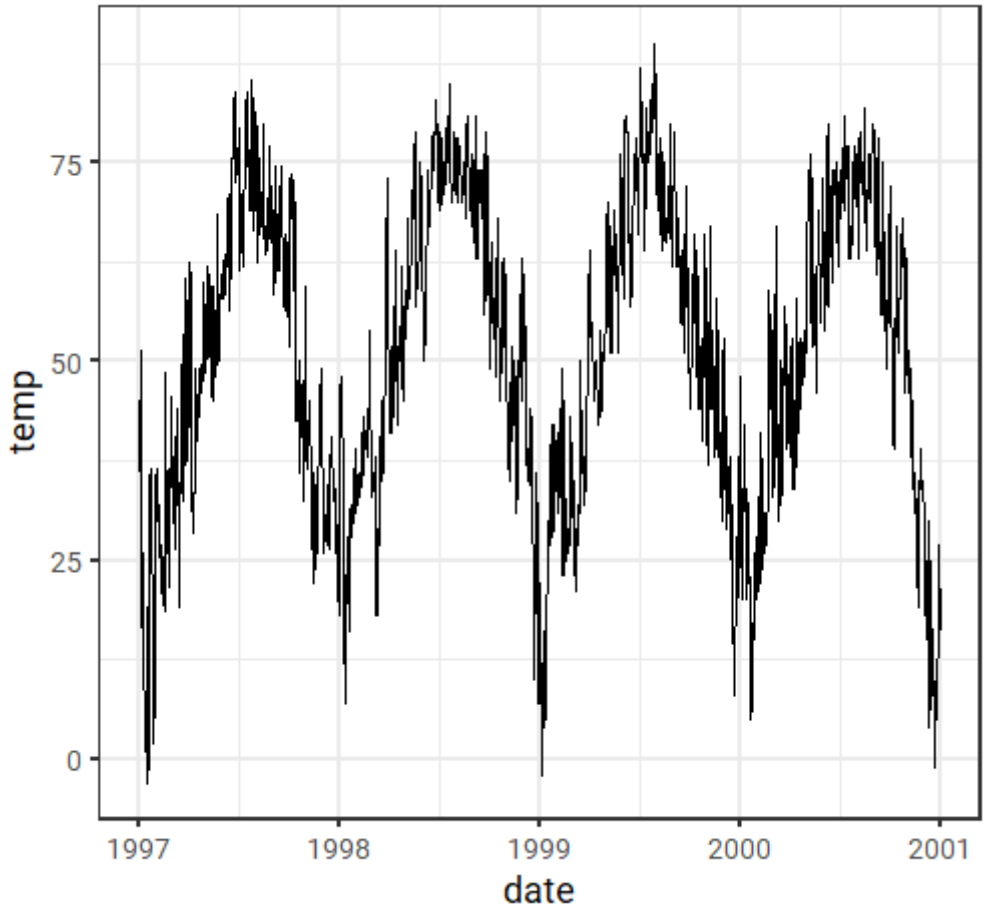
```
ggplot(chic, aes(date, temp)) +
  geom_point()
```



2. Layers — Geometries: `geom_line()`

... or a **line plot**:

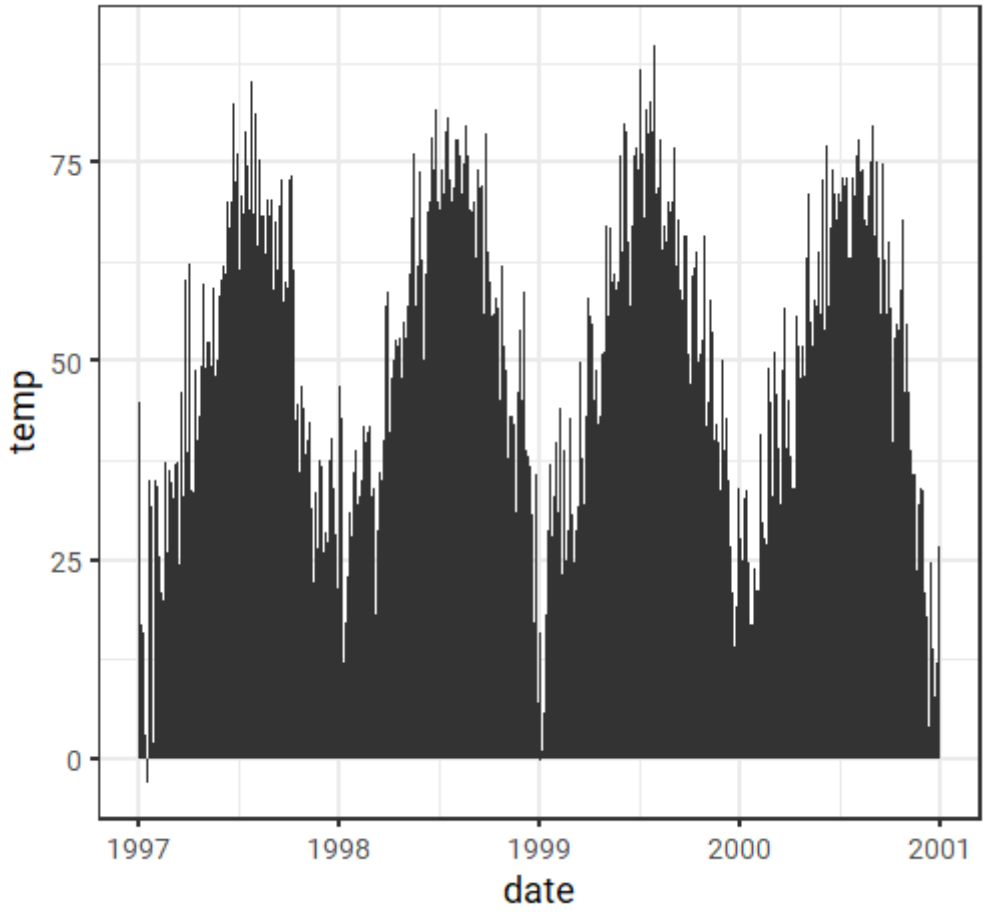
```
ggplot(chic, aes(date, temp)) +
  geom_line()
```



2. Layers — Geometries: `geom_area()`

... or an **area plot**:

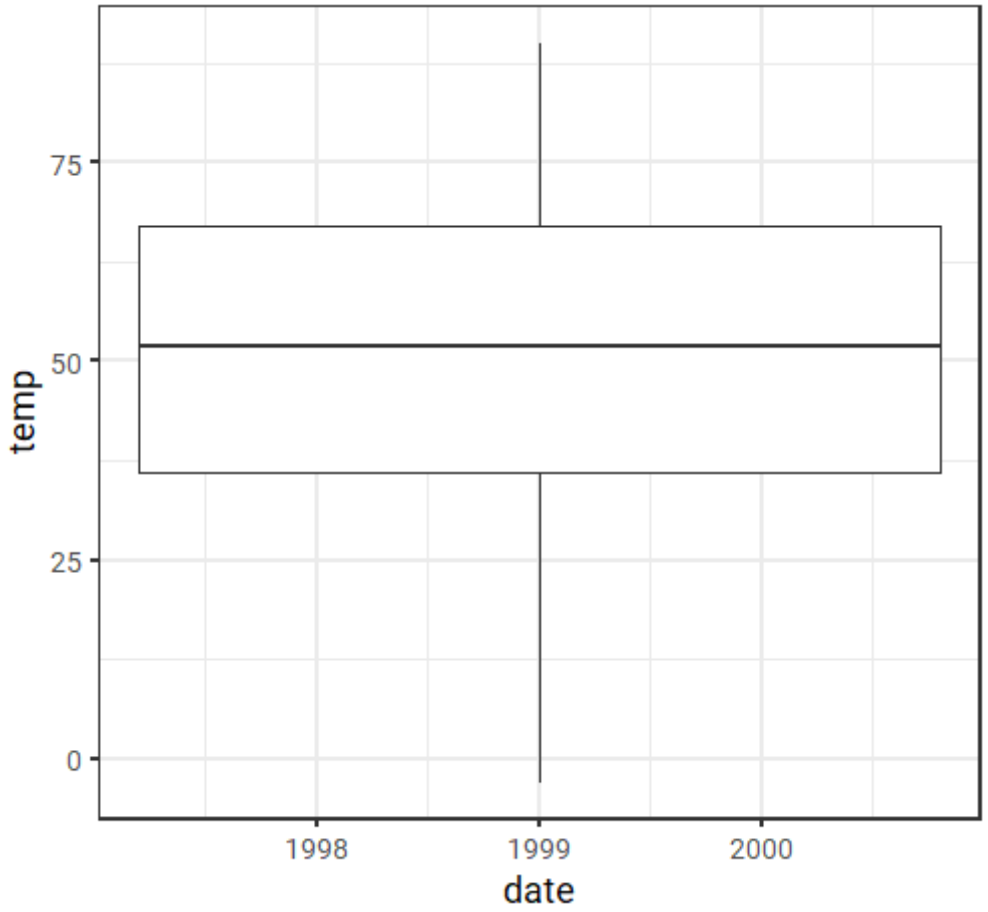
```
ggplot(chic, aes(date, temp)) +
  geom_area()
```



2. Layers — Geometries: `geom_boxplot()`

... or a **box and whiskers plot**:

```
ggplot(chic, aes(date, temp)) +
  geom_boxplot()
```

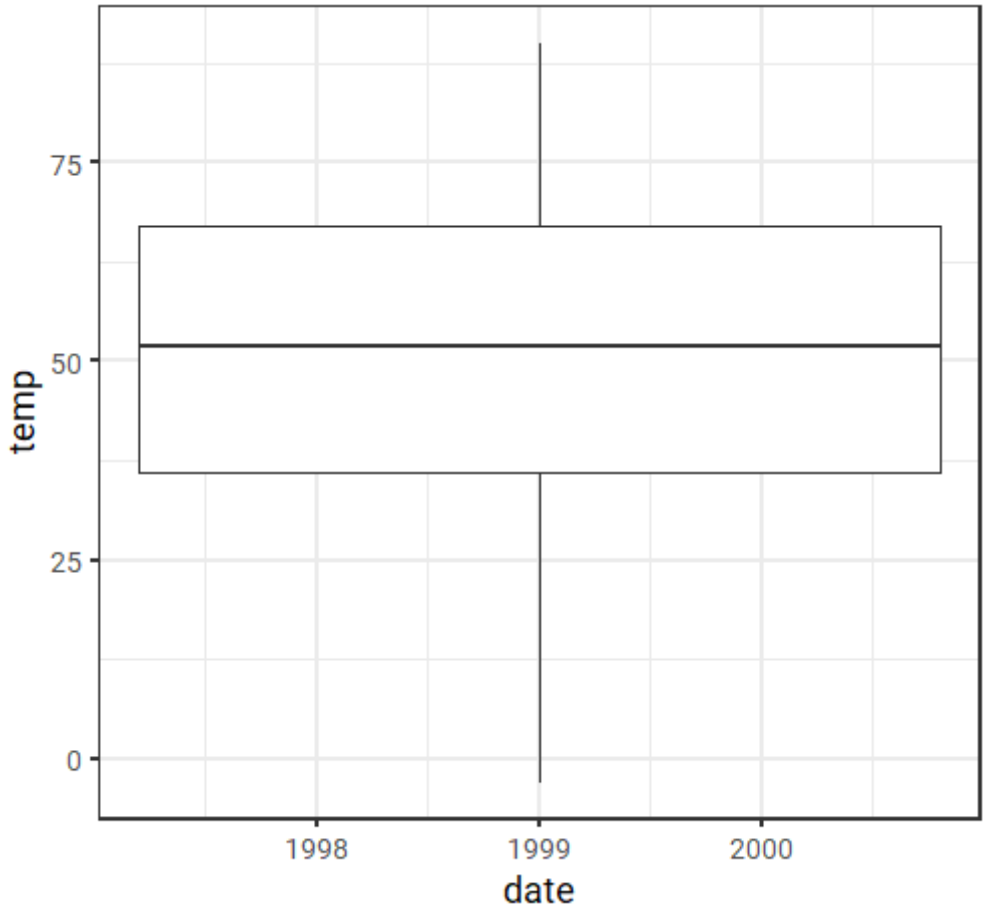


2. Layers — Geometries: `geom_boxplot()`

... or a **box and whiskers plot**:

```
ggplot(chic, aes(date, temp)) +
  geom_boxplot()
```

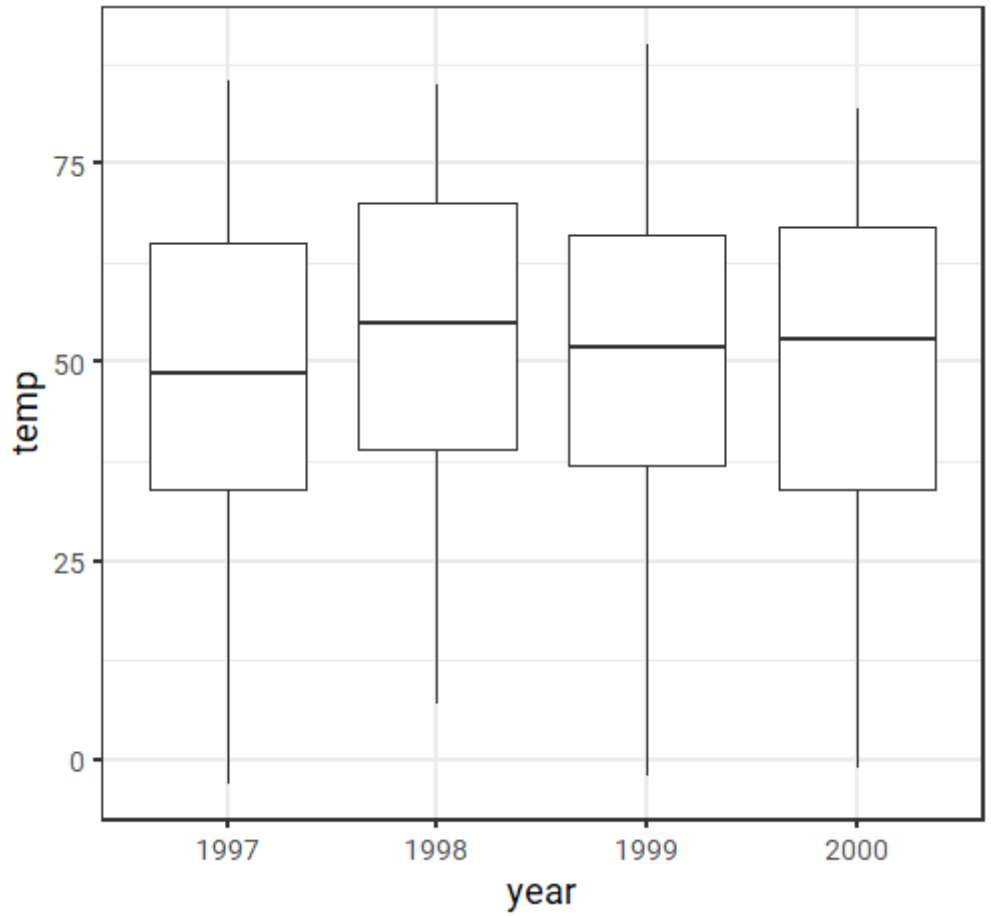
What's going on here?!



2. Layers — Geometries: `geom_boxplot()`

We need to specify the variable as **categorial** (`year`), not as **continuous** (`date`):

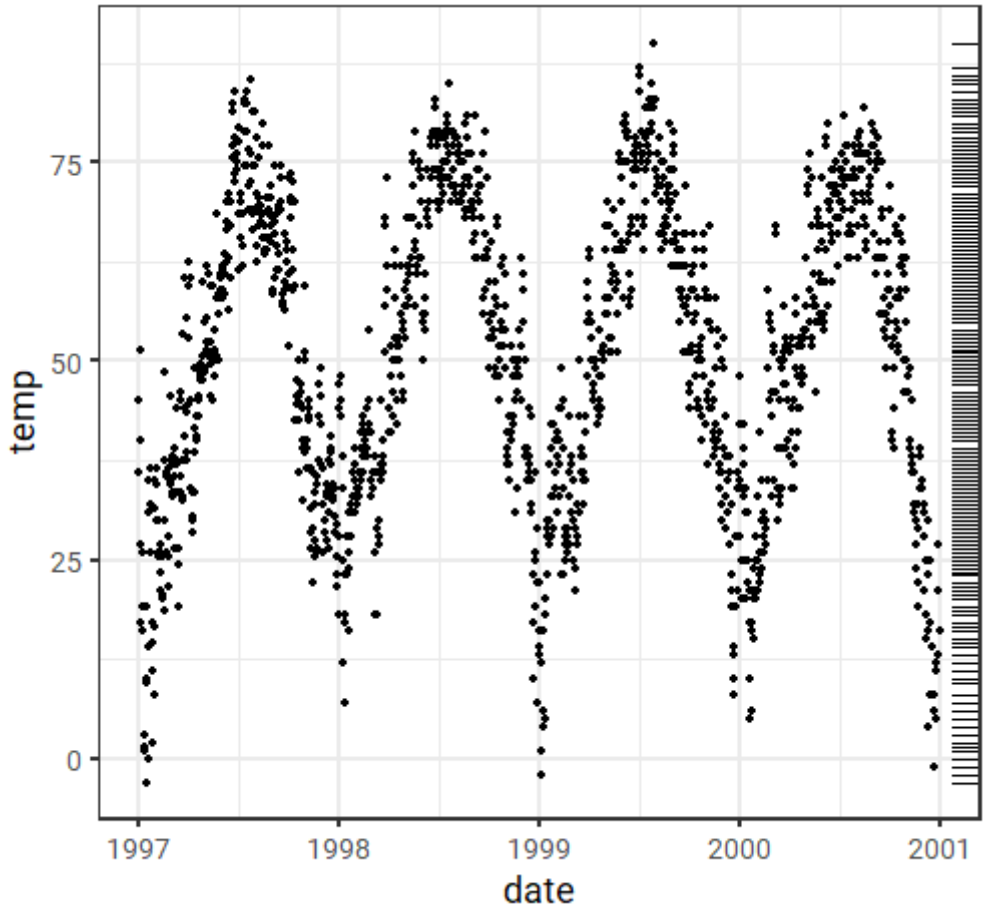
```
ggplot(chic, aes(year, temp)) +
    geom_boxplot()
```



2. Layers — Geometries: `geom_rug()`

Other layers can be added to an existing plot - a rug representation for example:

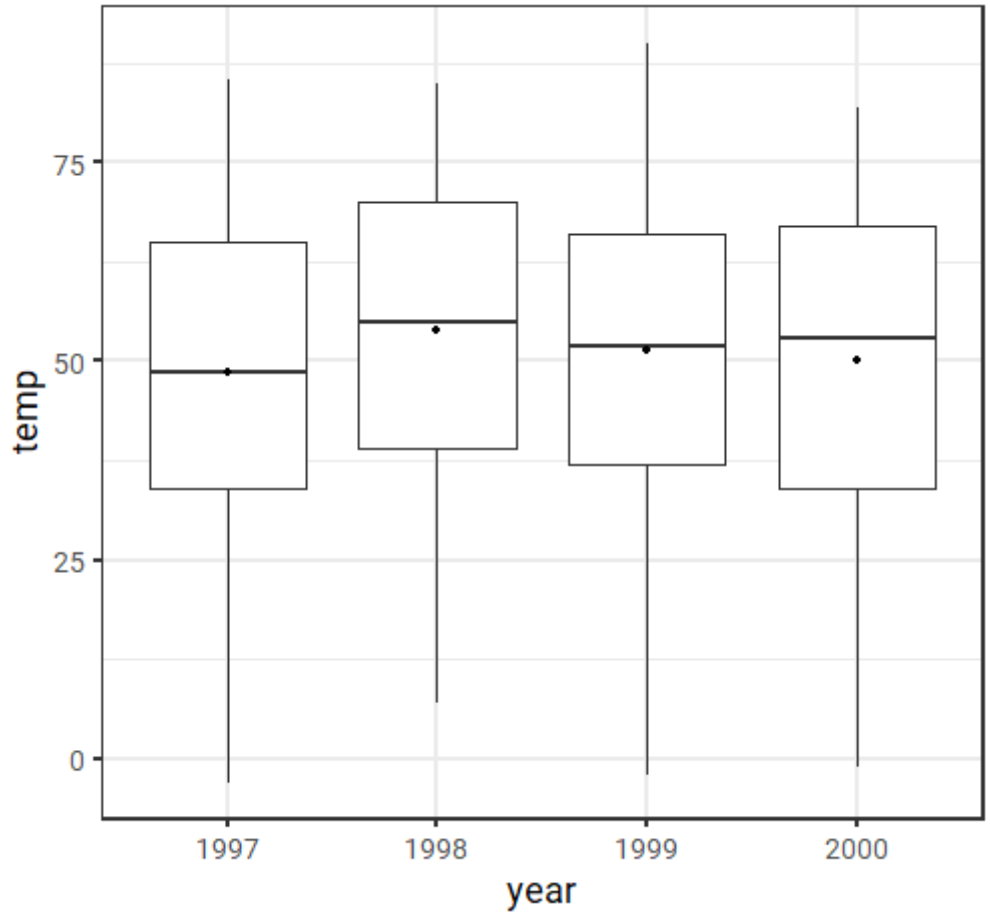
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  geom_rug(sides = "r")
```



2. Layers — Statistical transformations: `stat_summary()`

A handful of layers with attention to the statistical transformation rather than the visual appearance:

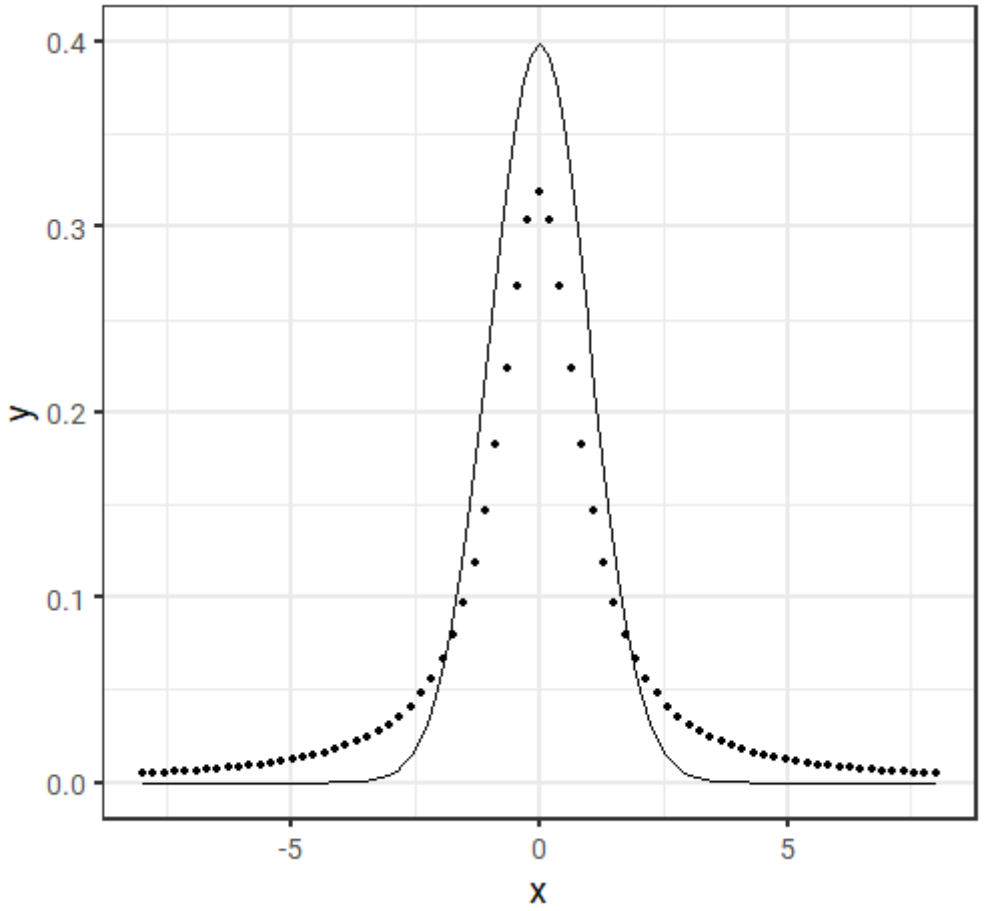
```
ggplot(chic, aes(year, temp)) +
  geom_boxplot() +
  stat_summary(
    fun.y = mean,
    geom = "point"
   )
```



2. Layers — Statistical transformations: `stat_function()`

`stat_function()` makes it easy to add a function to a plot, either continuous or discrete:

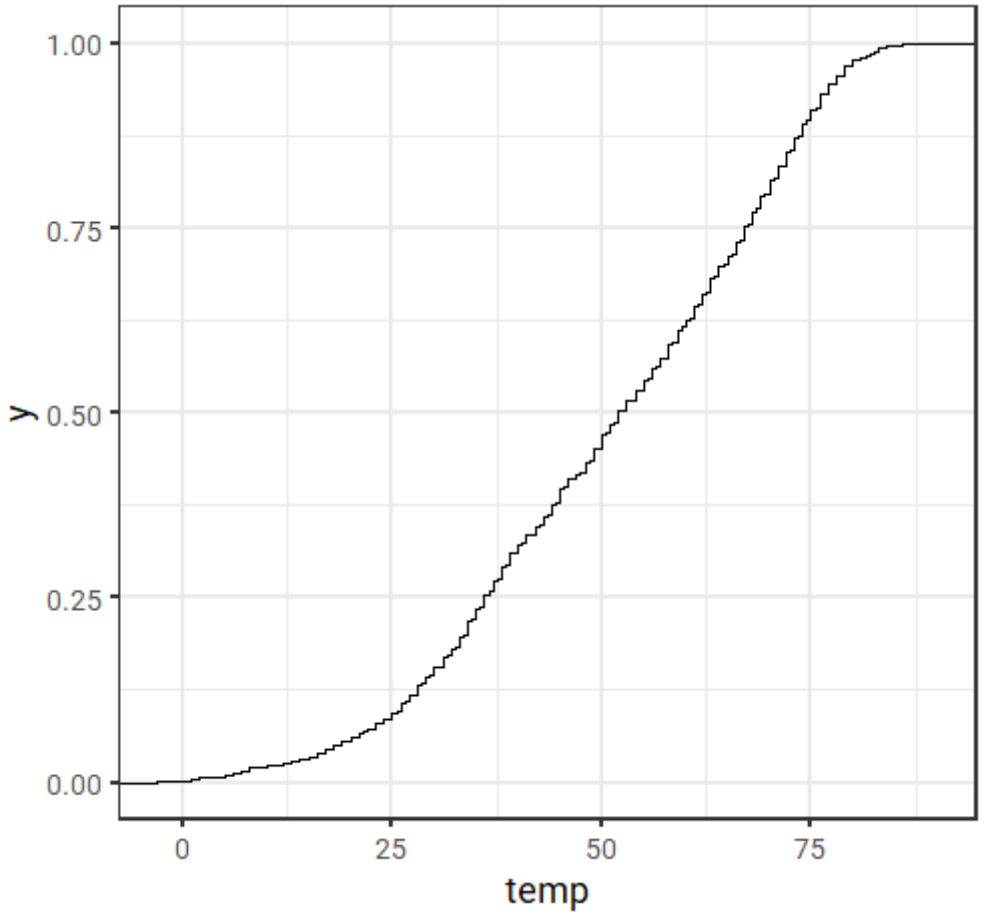
```
ggplot(tibble(x = c(-8, 8)), aes(x)) +
  stat_function(fun = dnorm) +
  stat_function(
    fun = dcauchy,
    geom = "point",
    n = 75
  )
```



2. Layers — Statistical transformations: `stat_ecdf()`

You can also easily plot the empirical cumulative distribution function (ECDF) of a variable:

```
ggplot(chic, aes(temp)) +
  stat_ecdf(geom = "step")
```

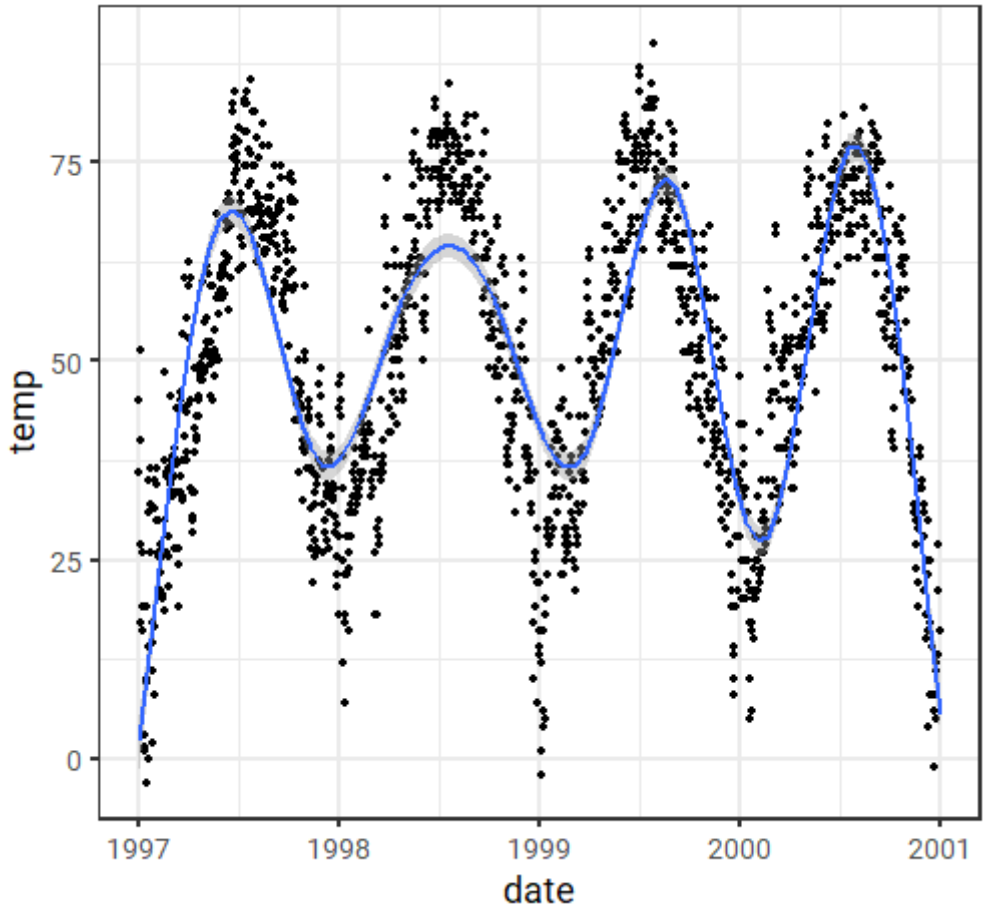


2. Layers — Statistical transformations: `stat_smooth()`

You can directly add smoothed conditional means:

```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  stat_smooth()
```

By default this adds a **LOESS** (locally weighted scatterplot smoothing, `method = "loess"`) or a **GAM** (generalized additive model, `method = "gam"`) depending on the number of data points (GAM in case of ≥ 1000 observations).

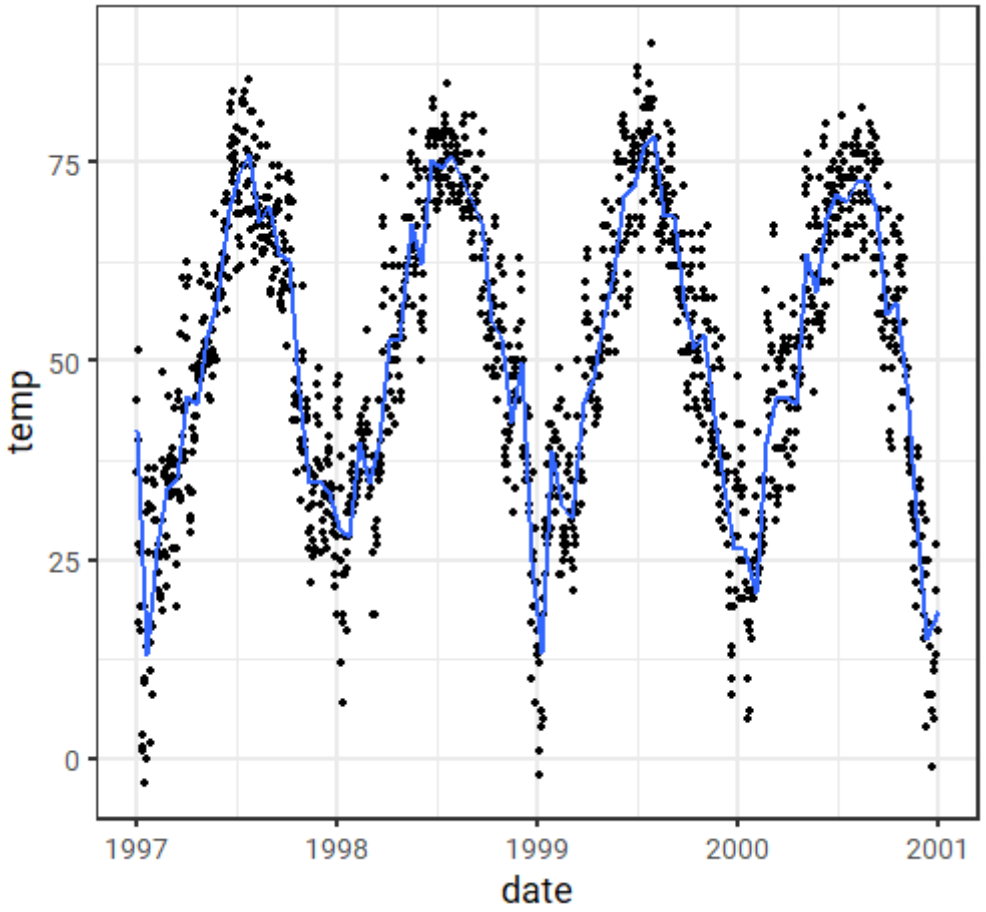


2. Layers — Statistical transformations: `stat_smooth()`

You can specify the fitting method and the formula:

```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  stat_smooth(
    method = "gam",
    formula = y ~ s(x, k = 100),
    se = F
  )
```

Other methods such as `method = "lm"` for linear regressions and `method = "glm"` for generalized linear models are available as well.



3. Aesthetics

`aes()`

3. Aesthetics: `aes()`

Aesthetics of the geometric and statistical objects, such as

- **position** via `x`, `y`, `xmin`, `xmax`, `ymin`, `ymax`, ...
- **colors** via `color` and `fill`
- **transparency** via `alpha`
- **sizes** via `size` and `width`
- **shapes** via `shape` and `linetype`

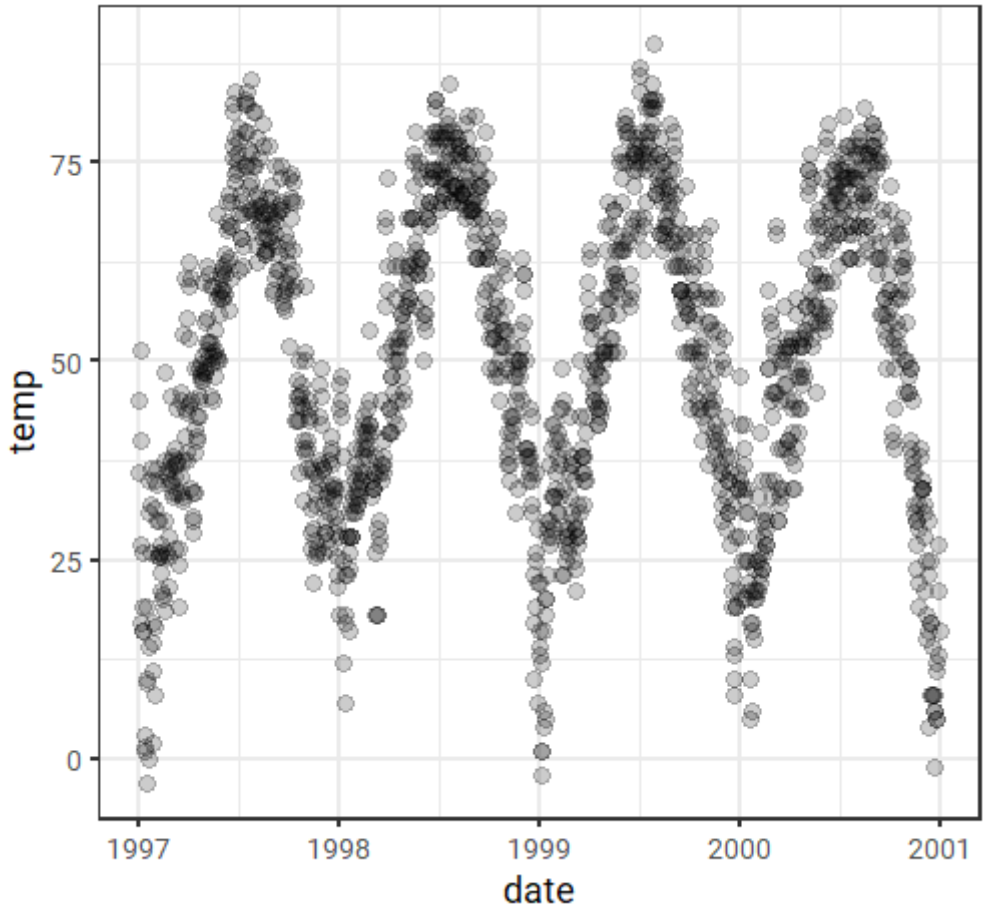
In general, everything which maps to the data needs to be wrapped in `aes()`
while static arguments are placed outside the `aes()`.

e.g.
`geom_point(aes(color = season))` to color points based on the variable `season`
`geom_point(color = "grey")` to color all points in the same color

3. Aesthetics: `aes(color/fill/alpha/size/shape)`

This way, we can enlarge all points and add some transparency:

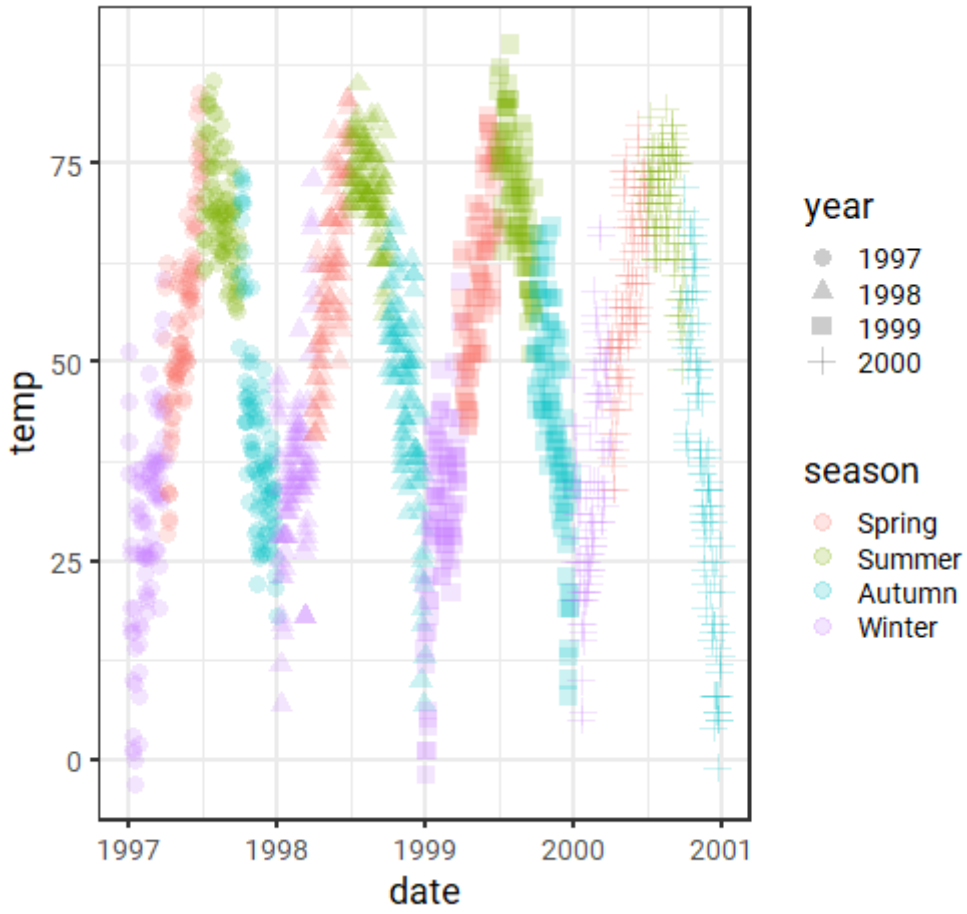
```
ggplot(chic, aes(date, temp)) +
  geom_point(
    size = 4,
    alpha = 0.2
  )
```



3. Aesthetics: `aes(color/fill/alpha/size/shape)`

... and change the color and the shape based on **season** and **year**:

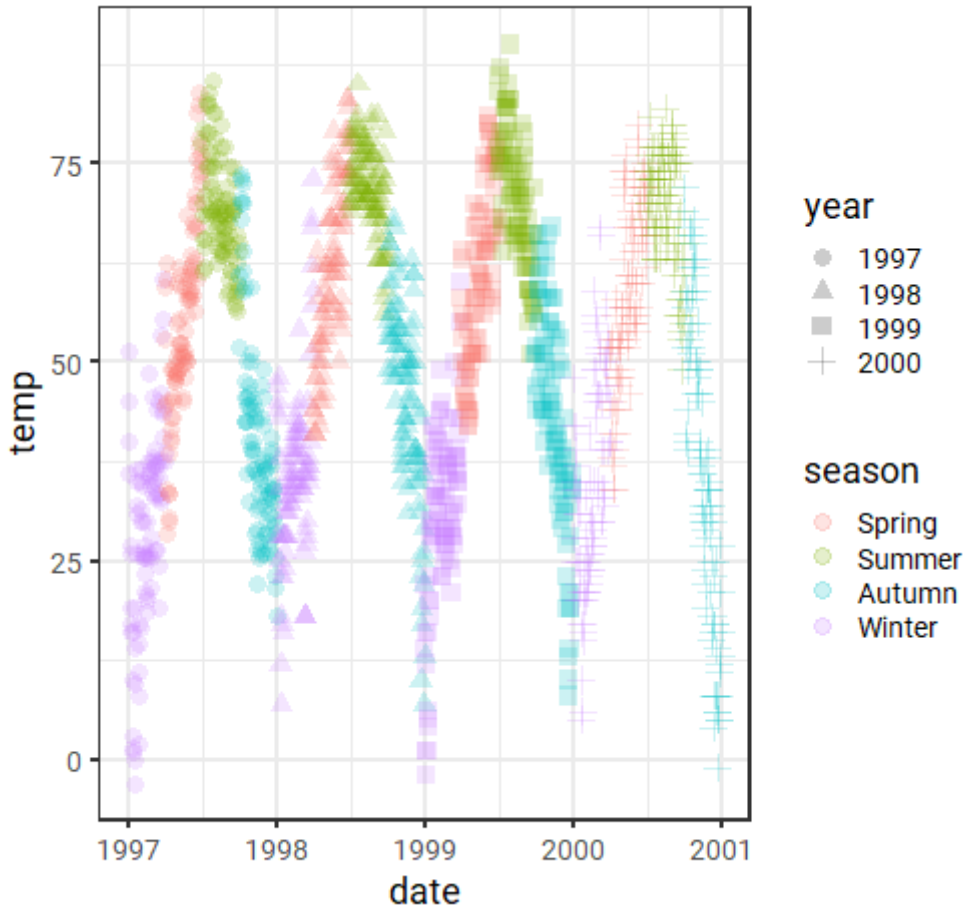
```
ggplot(chic, aes(date, temp)) +
  geom_point(
    aes(
      color = season,
      shape = year
    ),
    size = 4,
    alpha = 0.2
  )
```



3. Aesthetics: `aes(color/fill/alpha/size/shape)`

Alternatively, all `aes`thetics can be grouped together (and are then applied to all `geoms` and `stats`):

```
ggplot(
  chic,
  aes(date,
      temp,
      color = season,
      shape = year)
  ) +
  geom_point(
    size = 4,
    alpha = 0.2
  )
```

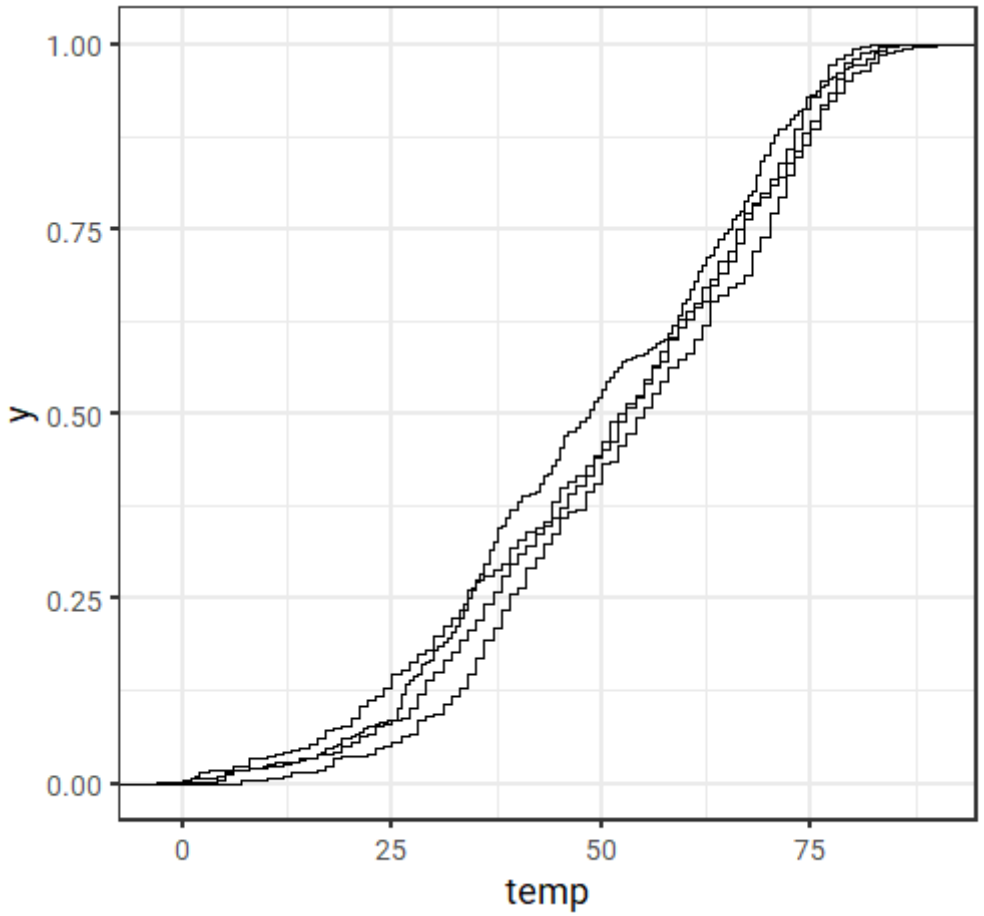


3. Aesthetics: `aes(group)`

You can create subsets of the data by specifying a grouping variable via **`group`**:

```
ggplot(chic, aes(temp)) +
  stat_ecdf(
    aes(group = year),
    geom = "step"
  )
```

However, for most applications you can simply specify the grouping using visual aesthetics (**`color`**, **`fill`**, **`alpha`**, **`shape`**, **`linetype`**).

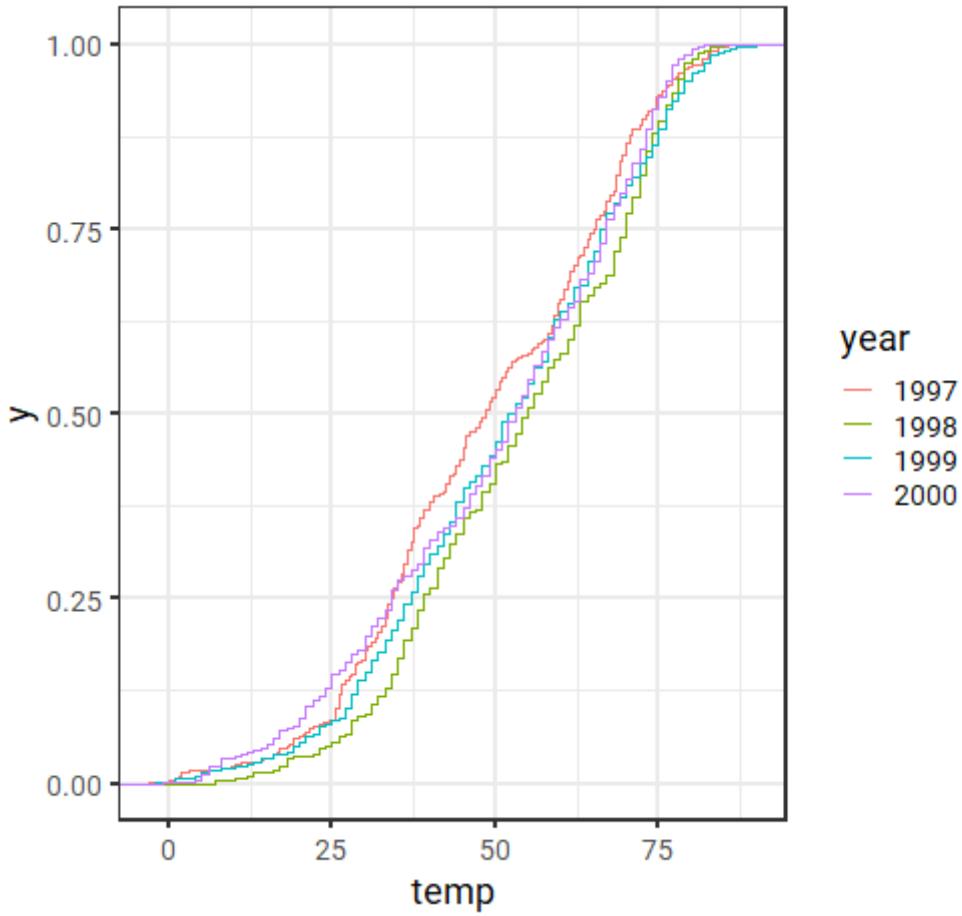


3. Aesthetics: `aes(group)`

You can create subsets of the data by specifying a grouping variable via **`group`**:

```
ggplot(chic, aes(temp)) +
  stat_ecdf(
    aes(color = year),
    geom = "step"
  )
```

However, for most applications you can simply specify the grouping using visual aesthetics (**`color`**, **`fill`**, **`alpha`**, **`shape`**, **`linetype`**).



4. Scales

`scale_*()`

4. Scales: `scale_`

One can use `scale_*()` to change properties of all the **aesthetic dimensions mapped to the data**.

Consequently, there are `scale_*()` functions for all aesthetics such as:

- **position** via `scale_x_*()` and `scale_y_*()`
- **colors** via `scale_color_*()` and `scale_fill_*()`
- **transparency** via `scale_alpha_*()`
- **sizes** via `scale_size_*()`
- **shapes** via `scale_shape_*()` and `scale_linetype_*()`

4. Scales: `scale_`

One can use `scale_*()` to change properties of all the **aesthetic dimensions mapped to the data**.

Consequently, there are `scale_*()` functions for all aesthetics such as:

- **position** via `scale_x_*()` and `scale_y_*()`
- **colors** via `scale_color_*()` and `scale_fill_*()`
- **transparency** via `scale_alpha_*()`
- **sizes** via `scale_size_*()`
- **shapes** via `scale_shape_*()` and `scale_linetype_*()`

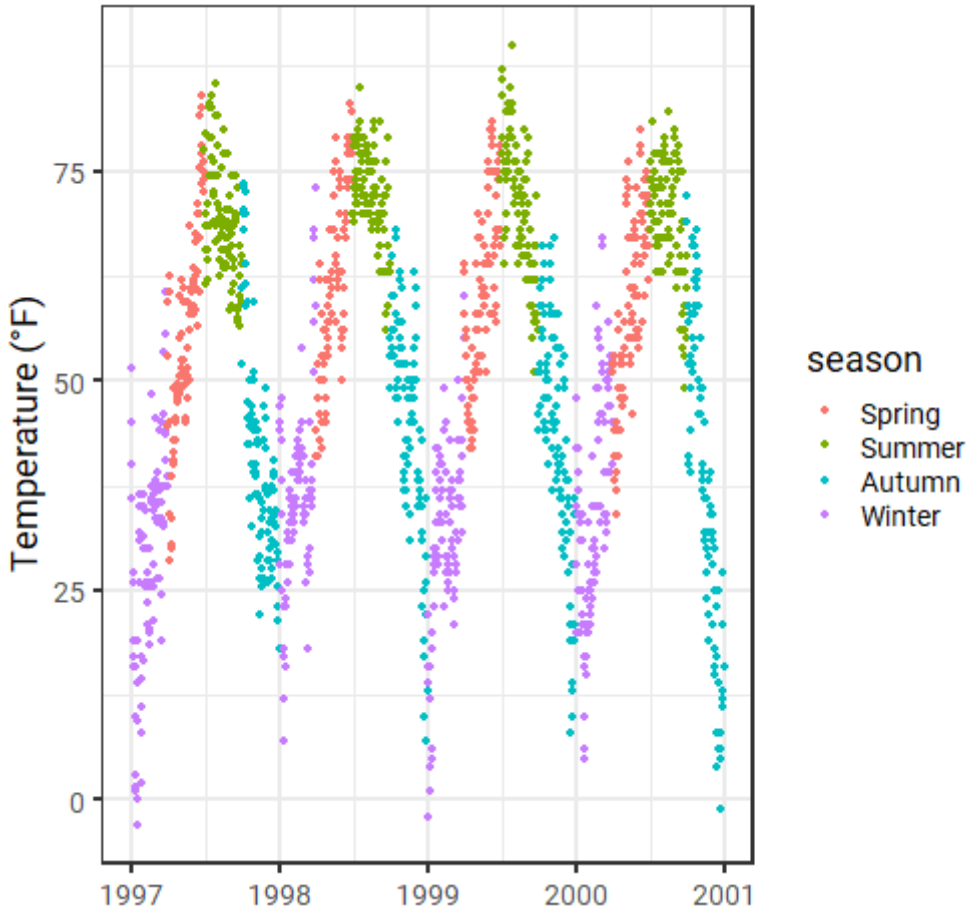
... with extensions (`*`) such as

- `continuous()`, `discrete()`, `reverse()`, `log10()`, `squrt()`, `date()`, `time()` for axes
- `continuous()`, `discrete()`, `manual()`, `gradient()`, `hue()`, `brewer()` for colors and fills
- `continuous()`, `discrete()`, `manual()`, `ordinal()`, `identity()`, `date()` for transparencies
- `continuous()`, `discrete()`, `manual()`, `ordinal()`, `identity()`, `area()`, `date()` for sizes
- `continuous()`, `discrete()`, `manual()`, `ordinal()`, `identity()` for shapes and linetypes

4. Scales: `scale_x_*()` and `scale_y_*()`

For example, to change the titles of the axes...

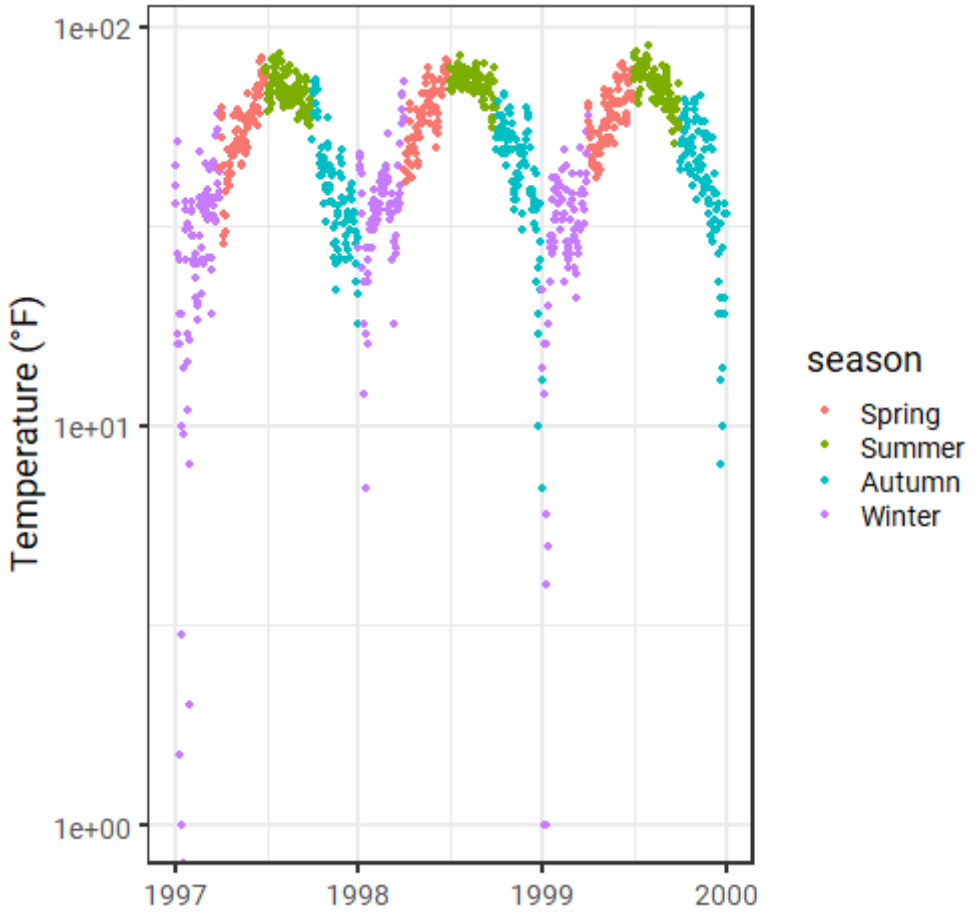
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_x_date(
    name = NULL
  ) +
  scale_y_continuous(
    name = "Temperature (°F)"
  )
```



4. Scales: `scale_x_*()` and `scale_y_*()`

... and their properties such as the range, scaling, labels, and axis breaks:

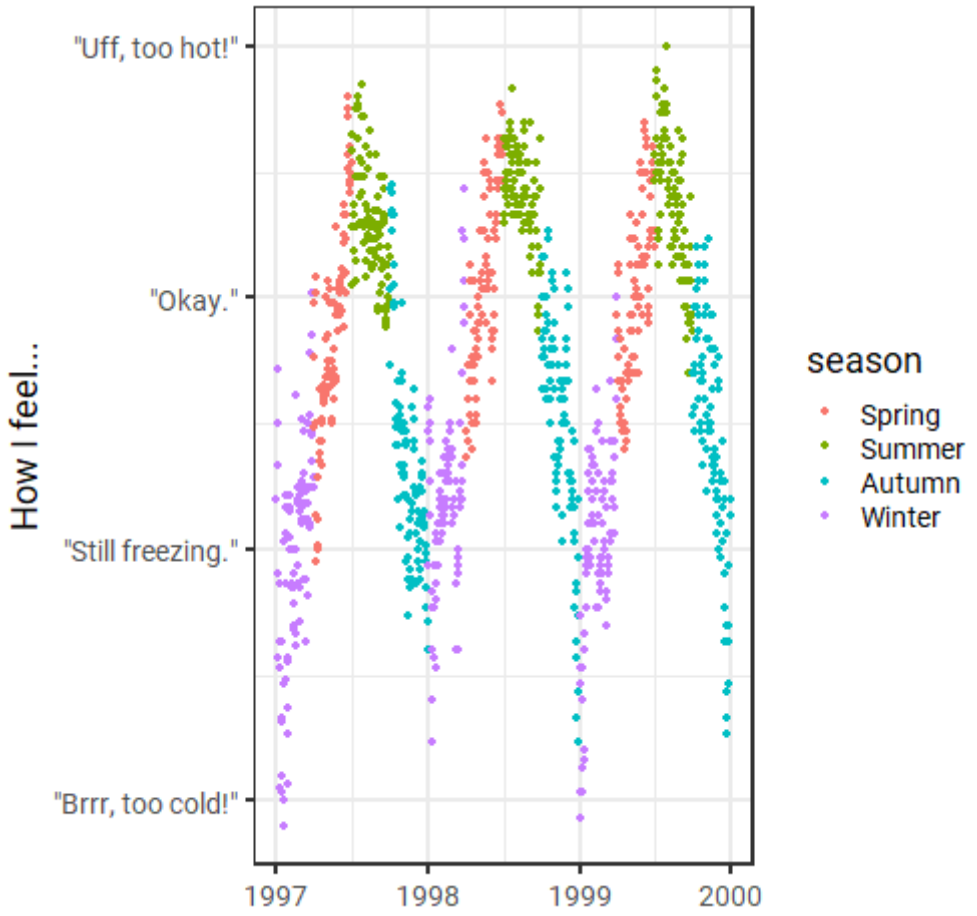
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_x_date(
    name = NULL,
    limits = c(
      as.Date("1997-01-01"),
      as.Date("1999-12-31")
    )
  ) +
  scale_y_log10(
    name = "Temperature (°F)",
    breaks = c(1, 10, 100),
    labels = scales::scientific
  )
```



4. Scales: `scale_x_*()` and `scale_y_*()`

With `breaks` and `labels` it's also possible to alter the labels without changing the data:

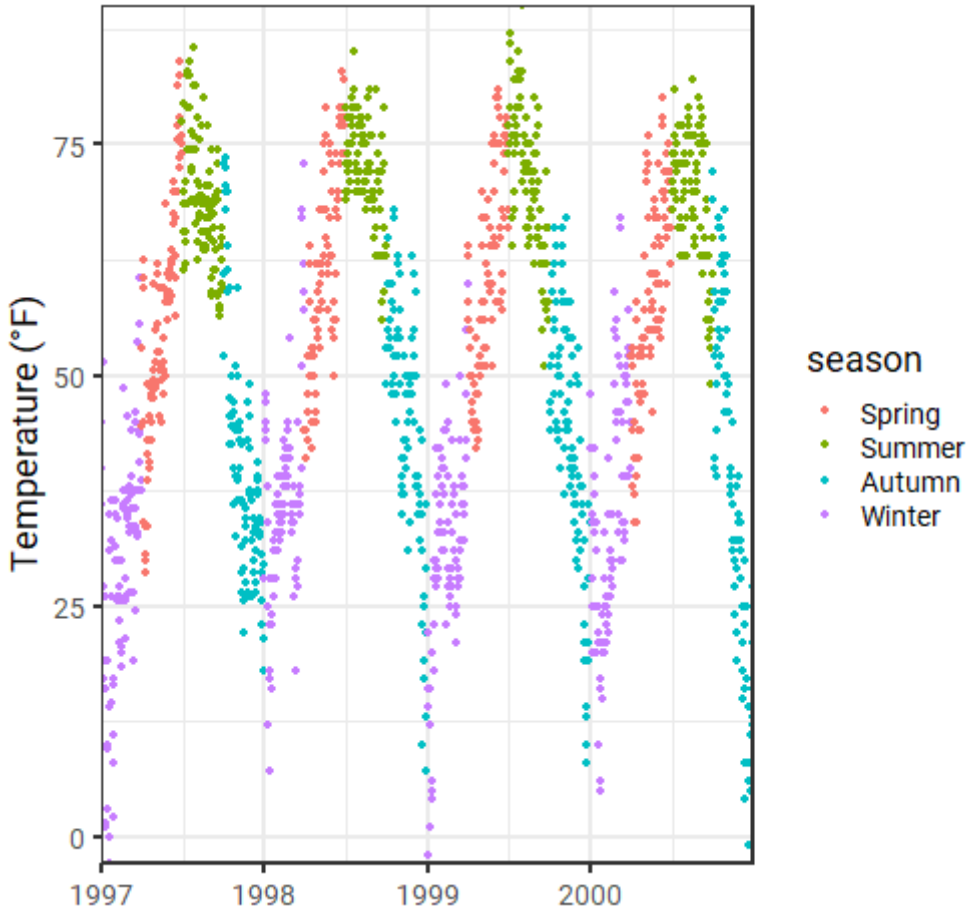
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_x_date(
    name = NULL,
    limits = c(
      as.Date("1997-01-01"),
      as.Date("1999-12-31")
    )
  ) +
  scale_y_continuous(
    name = "How I feel...",
    breaks = c(0, 30, 60, 90),
    labels = c(
      '"Brrr, too cold!"',
      '"Still freezing."',
      '"Okay."',
      '"Uff, too hot!"'
    )
  )
```



4. Scales: `scale_x_*()` and `scale_y_*()`

Some people are annoyed by the extra spacing around the data but we can remove that:

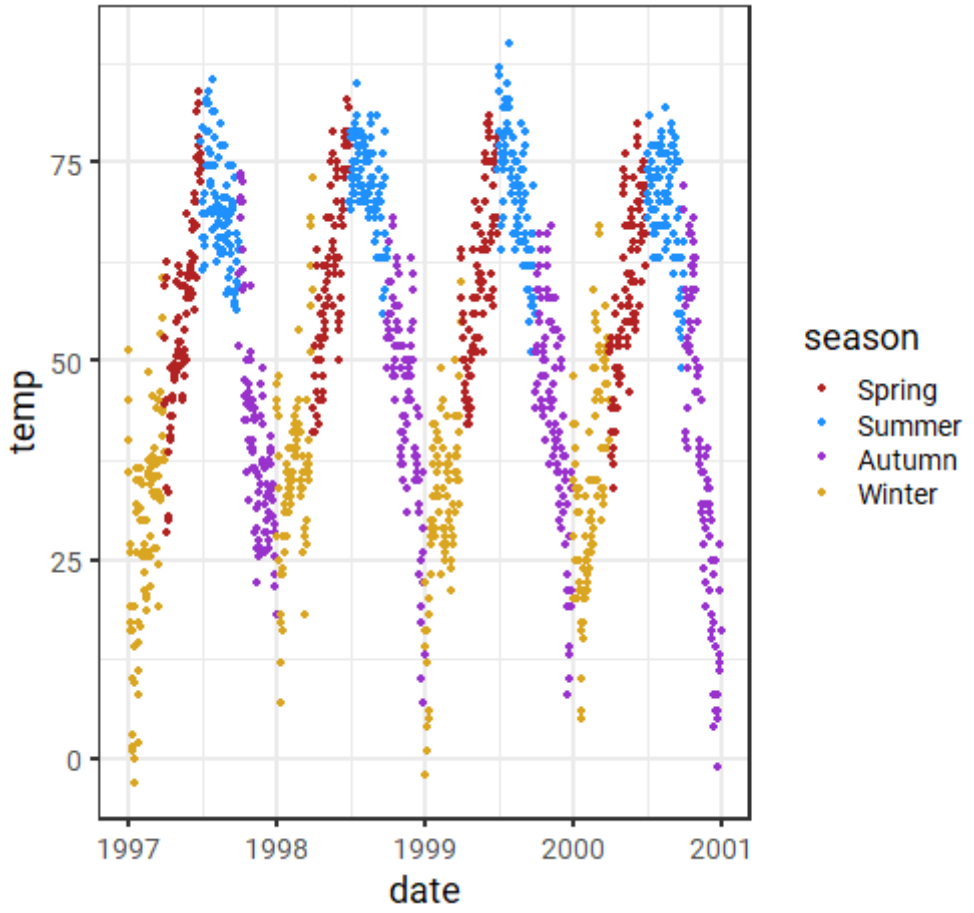
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_x_date(
    name = NULL,
    expand = c(0, 0)
  ) +
  scale_y_continuous(
    name = "Temperature (°F)",
    expand = c(0, 0)
  )
```



4. Scales: `scale_color_*()` and `scale_fill_*()`

Similarly, we can change properties of the other aesthetics. for example get rid of the default colors:

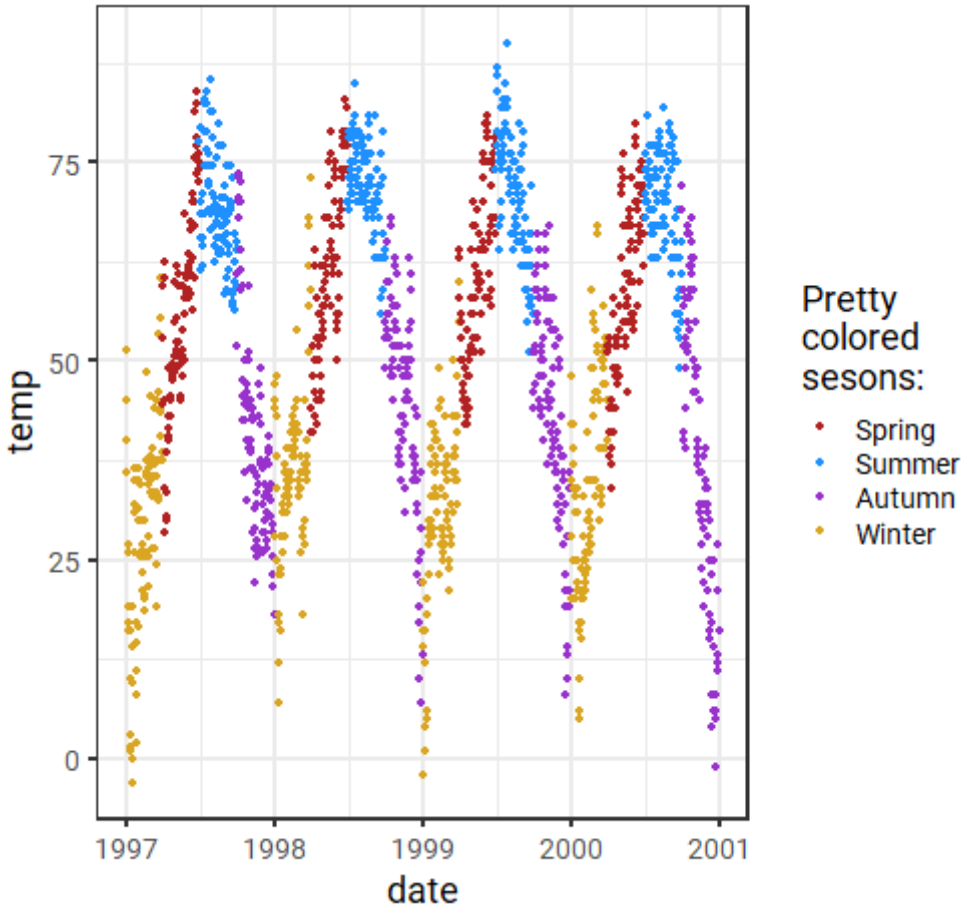
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_color_manual(
    values = c(
      "firebrick",
      "dodgerblue",
      "darkorchid",
      "goldenrod"
    )
  )
```



4. Scales: `scale_color_*()` and `scale_fill_*()`

... and change the title of the legend:

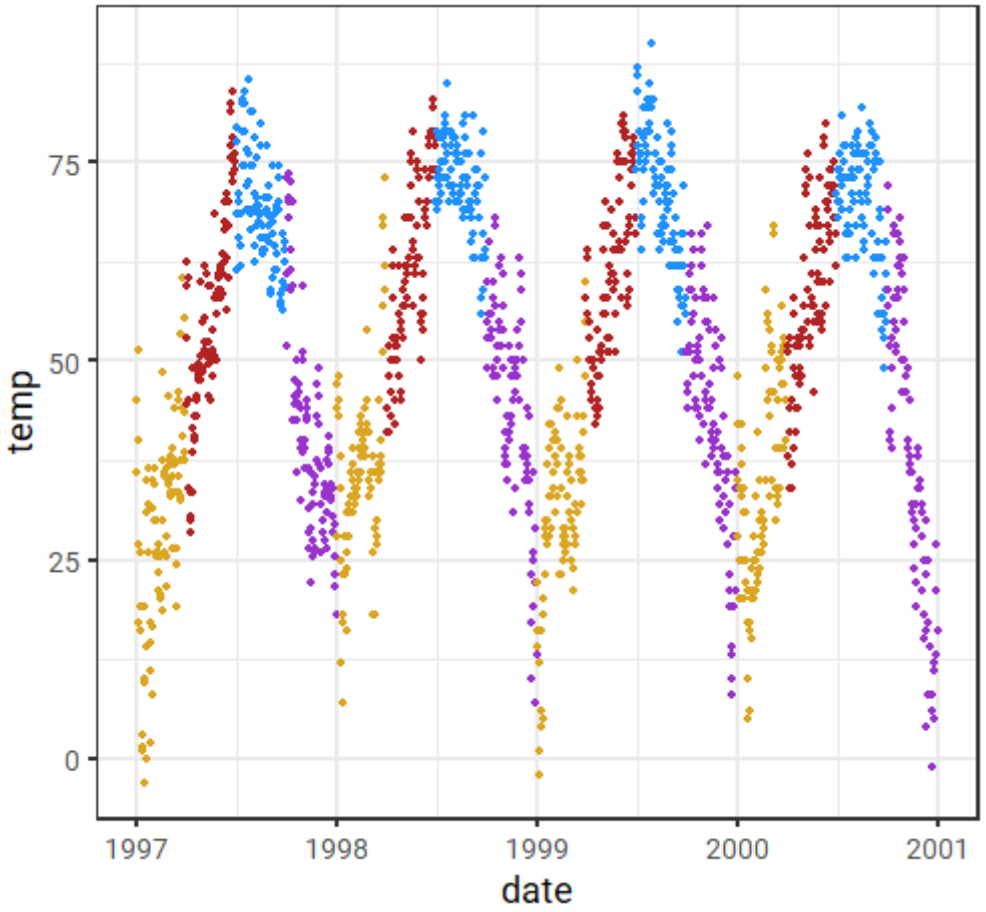
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_color_manual(
    values = c(
      "firebrick",
      "dodgerblue",
      "darkorchid",
      "goldenrod"
    ),
    name = "Pretty\ncolored\nsesons:"
  )
```



4. Scales: `scale_color_*()` and `scale_fill_*()`

... or remove the legend for a specific aesthetic:

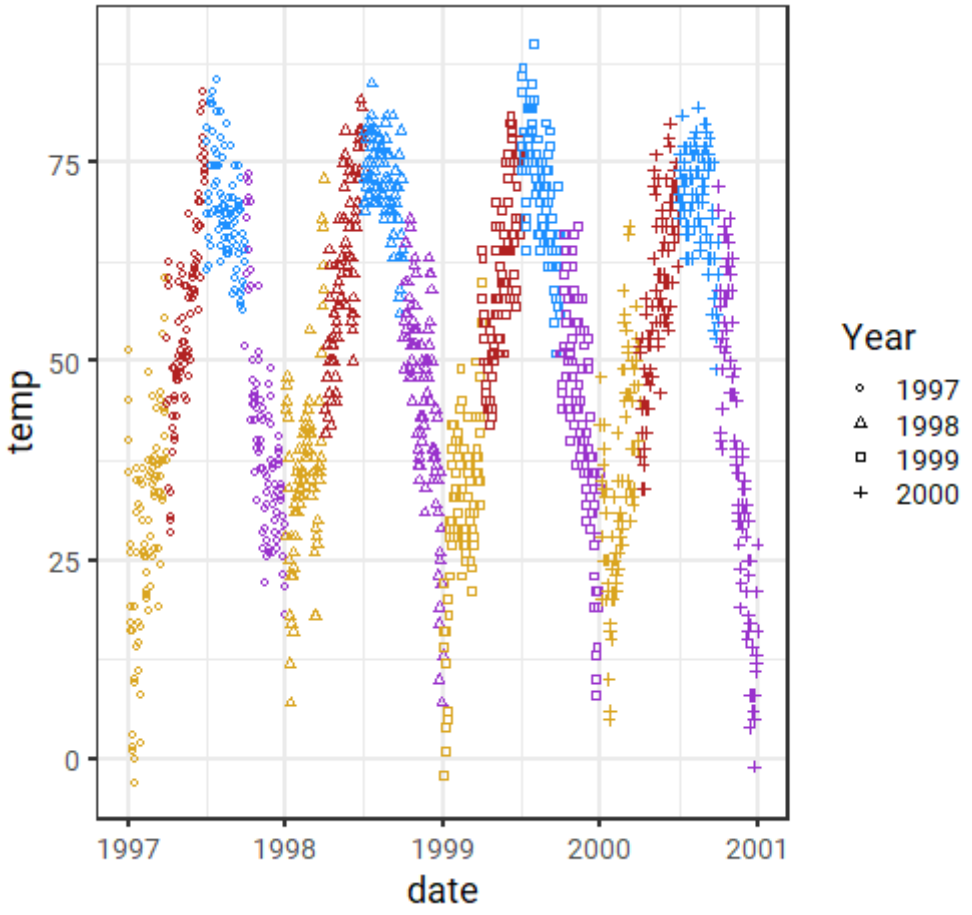
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_color_manual(
    values = c(
      "firebrick",
      "dodgerblue",
      "darkorchid",
      "goldenrod"
    ),
    guide = FALSE
  )
```



4. Scales: `scale_shape_*()`

... or remove the legend for a **specific** aesthetic:

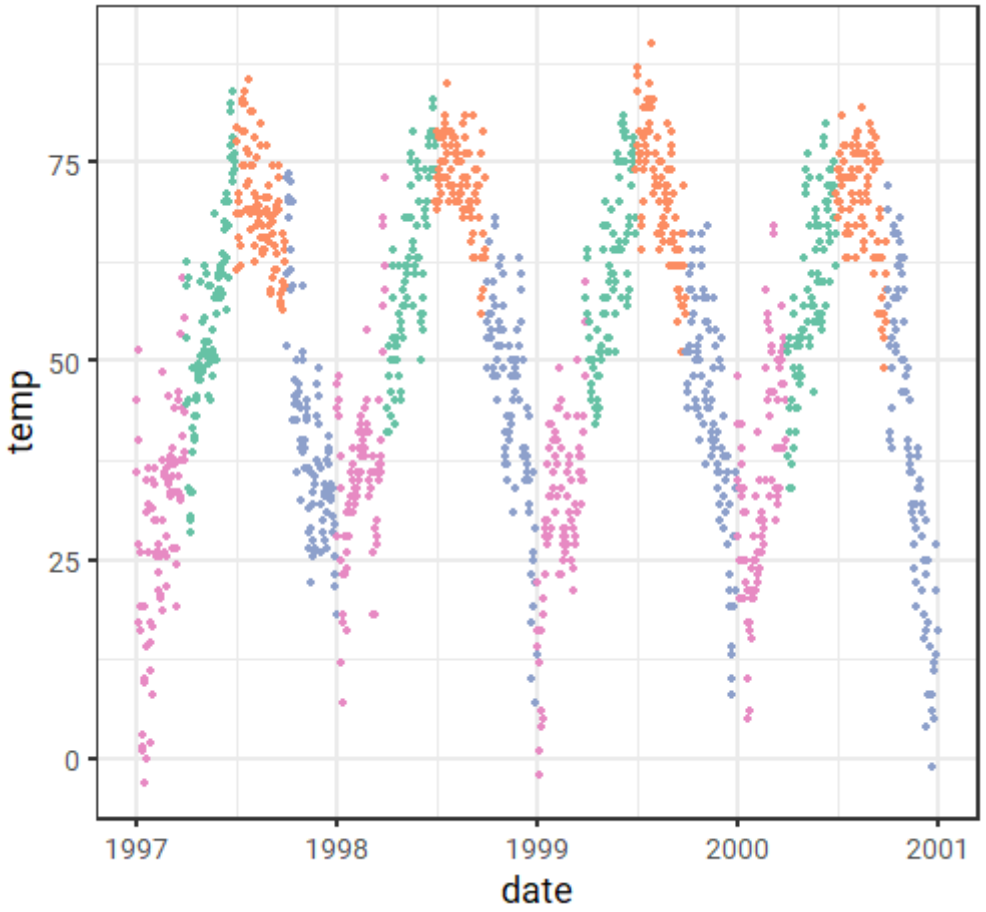
```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season,
                 shape = year)) +
  scale_color_manual(
    values = c(
      "firebrick",
      "dodgerblue",
      "darkorchid",
      "goldenrod"
    ),
    guide = FALSE
  ) +
  scale_shape_discrete(
    solid = FALSE,
    name = "Year"
  )
```



4. Scales: `scale_color_*()` and `scale_fill_*()`

One can of course also use preset color palettes:

```
ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_color_brewer(
    palette = "Set2",
    guide = FALSE
  )
```

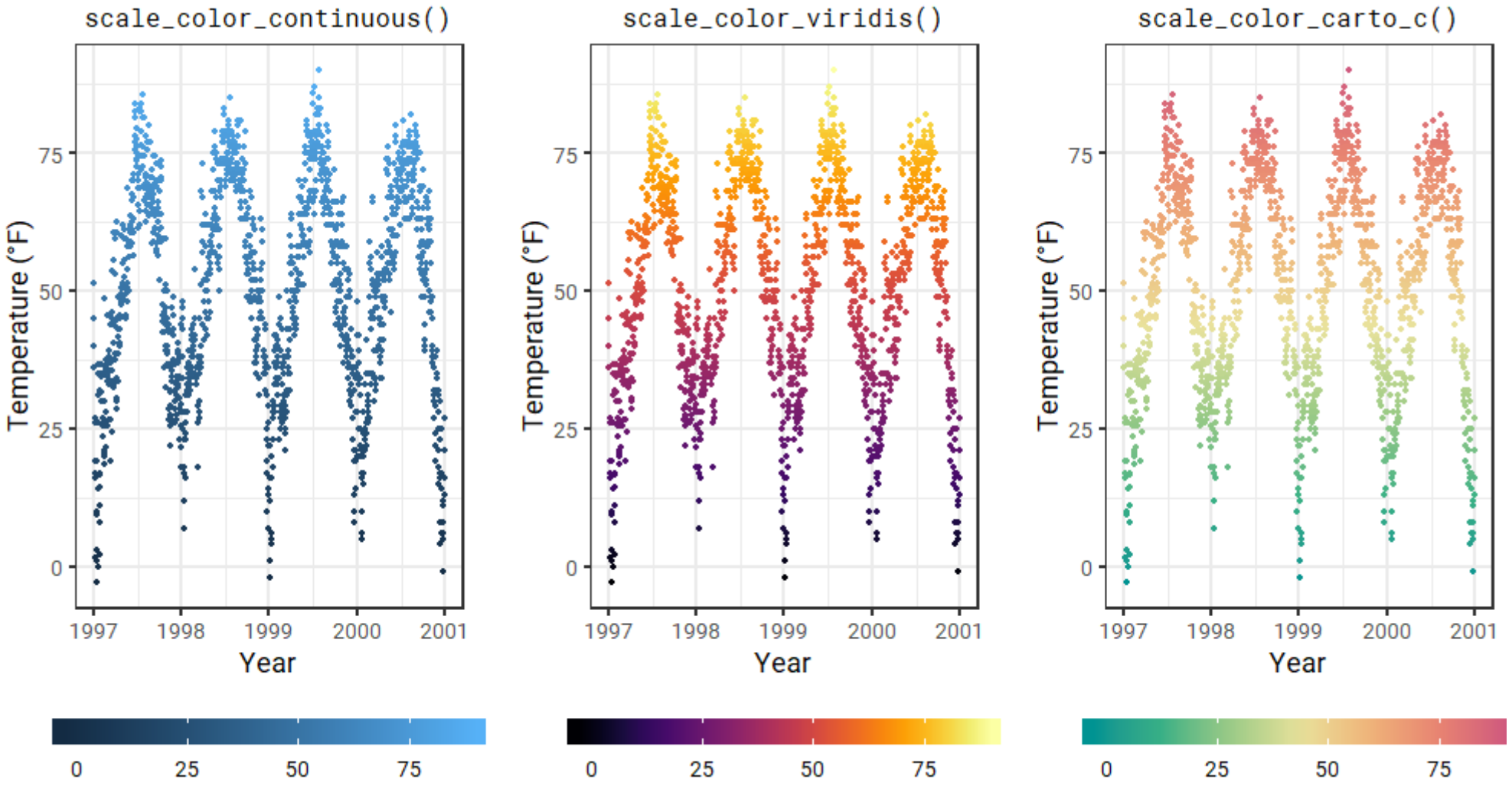


Several packages offer predefined palettes, e.g.:

- `{viridis}` for color-blind safe palettes
- `{rcartocolor}` for map color schemes
- `{LaCroixColoR}` for vibrant summery colors
- `{ggthemes}` for colors of popular software and publishers

Check the collection by Emil Hvitfeldt for a list of color palettes available in `R`.

4. Scales: `scale_color_*()` and `scale_fill_*()` — Color Palettes



5. Coordinate System

`coord_*()`

5. Coordinate System: `coord_*()`

Coordinate systems combine the two position aesthetics (usually `x` and `y`) to produce a 2d position on the plot.

The meaning of the position aesthetics depends on the coordinate system used:

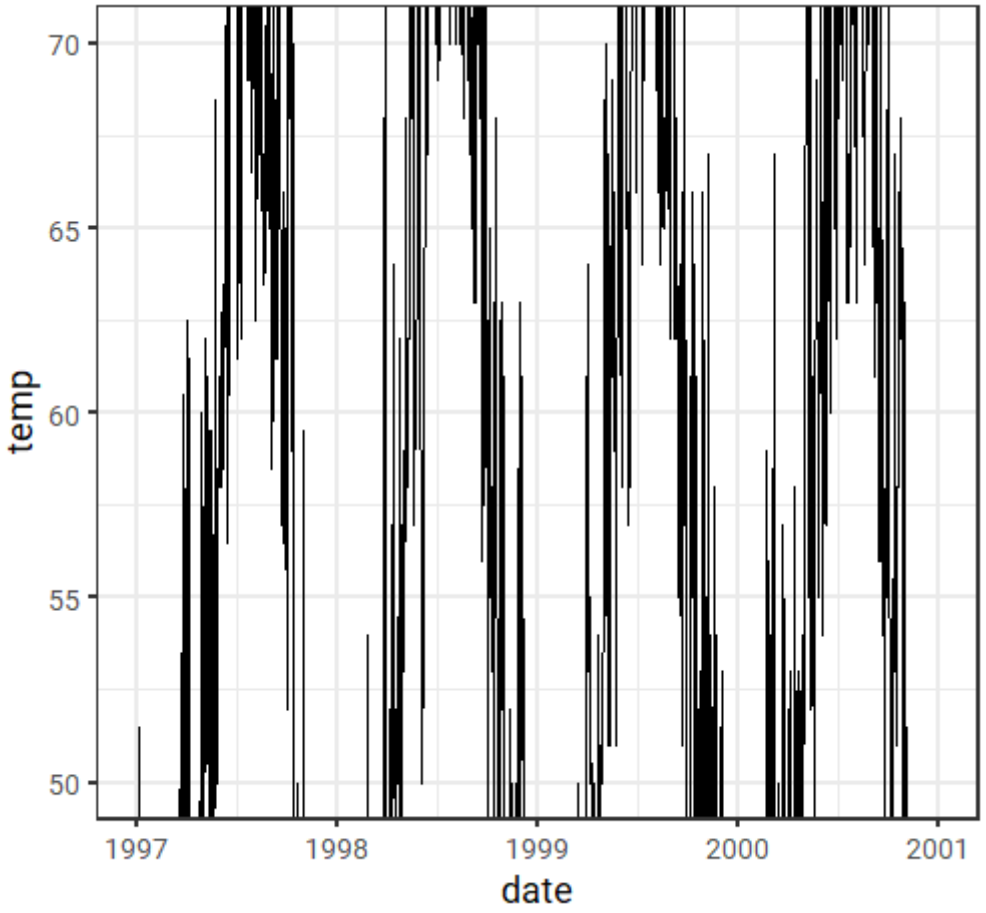
- Linear coordinate systems that preserve the shape of geoms:
    - `coord_cartesian()`: the default with two fixed perpendicular oriented axes
    - `coord_flip()`: a Cartesian coordinate system with flipped axes
    - `coord_fixed()`: a Cartesian coordinate system with a fixed aspect ratio
- Non-linear coordinate systems that likely change the shapes:
    - `coord_map()`: map projections
    - `coord_polar()`: a polar coordinate system
    - `coord_trans()`: arbitrary transformations to x and y positions

5. Coordinate System: `coord_cartesian()`

`coord_*()` functions allow you to zoom in a plot:

```
ggplot(chic, aes(date, temp)) +
  geom_line() +
  coord_cartesian(
    ylim = c(50, 70)
  )
```

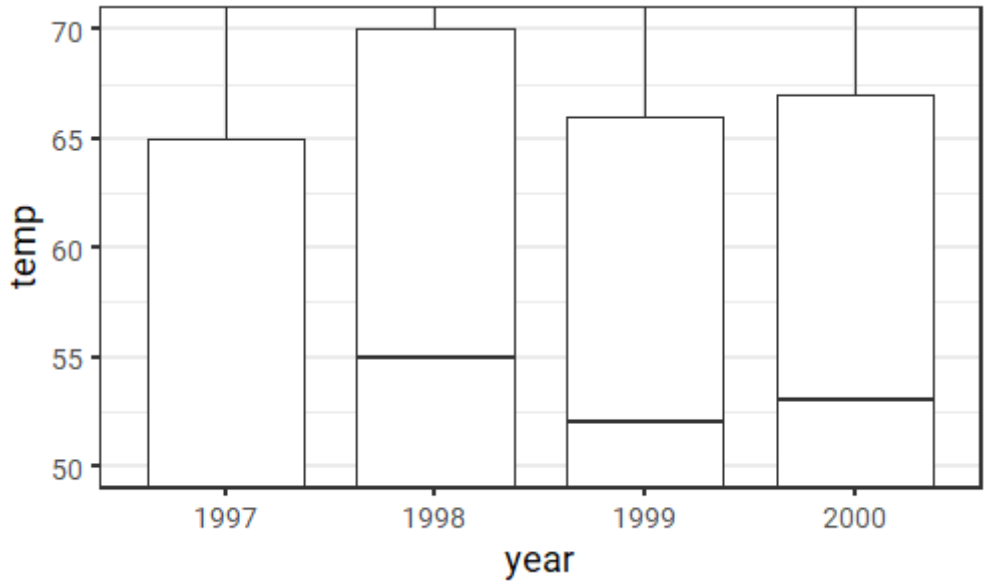
Note that this does prevent a removal of the data points outside the plot area!



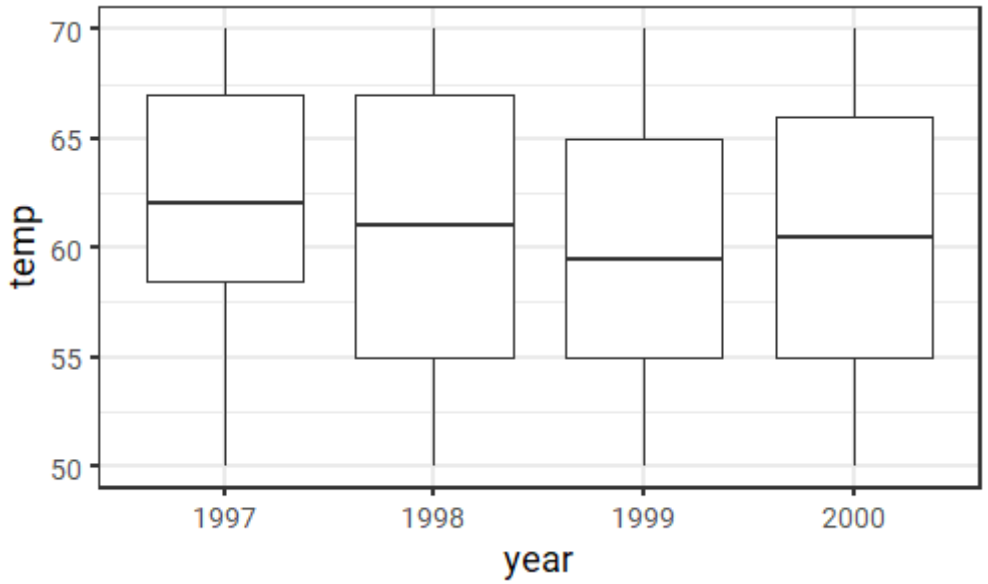
5. Coordinate System: `coord_cartesian()`

In case you want to remove those data points, use `scale_y_continuous(limits = c(min, max))`:

```
ggplot(chic, aes(year, temp)) +
  geom_boxplot()  +
  coord_cartesian(
    ylim = c(50, 70)
  )
```



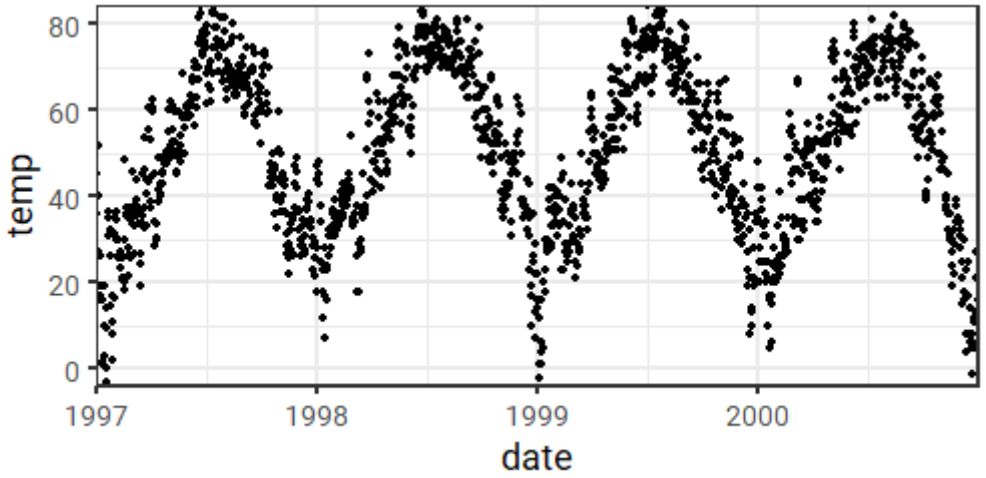

```
ggplot(chic, aes(year, temp)) +
  geom_boxplot()  +
  scale_y_continuous(
    limits = c(50, 70)
  )
```

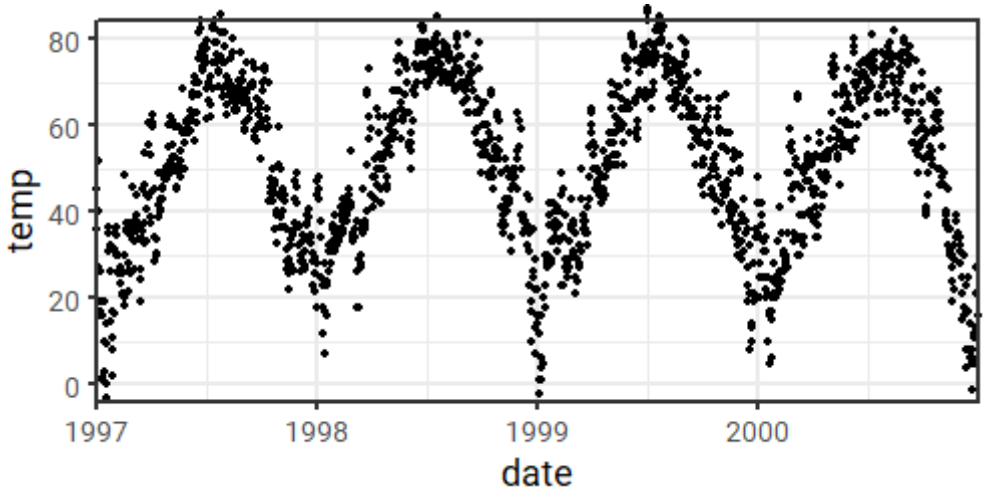


5. Coordinate System: `coord_cartesian()`

`coord_*()` is also used to *prevent clipping* of points and correct alignment of axis ticks:

```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  scale_x_date(
    expand = c(0, 0)
  ) +
  coord_cartesian(ylim = c(0, 80))

ggplot(chic, aes(date, temp)) +
  geom_point() +
  scale_x_date(
    expand = c(0, 0)
  ) +
  coord_cartesian(
    ylim = c(0, 80),
    clip = "off"
  )
```

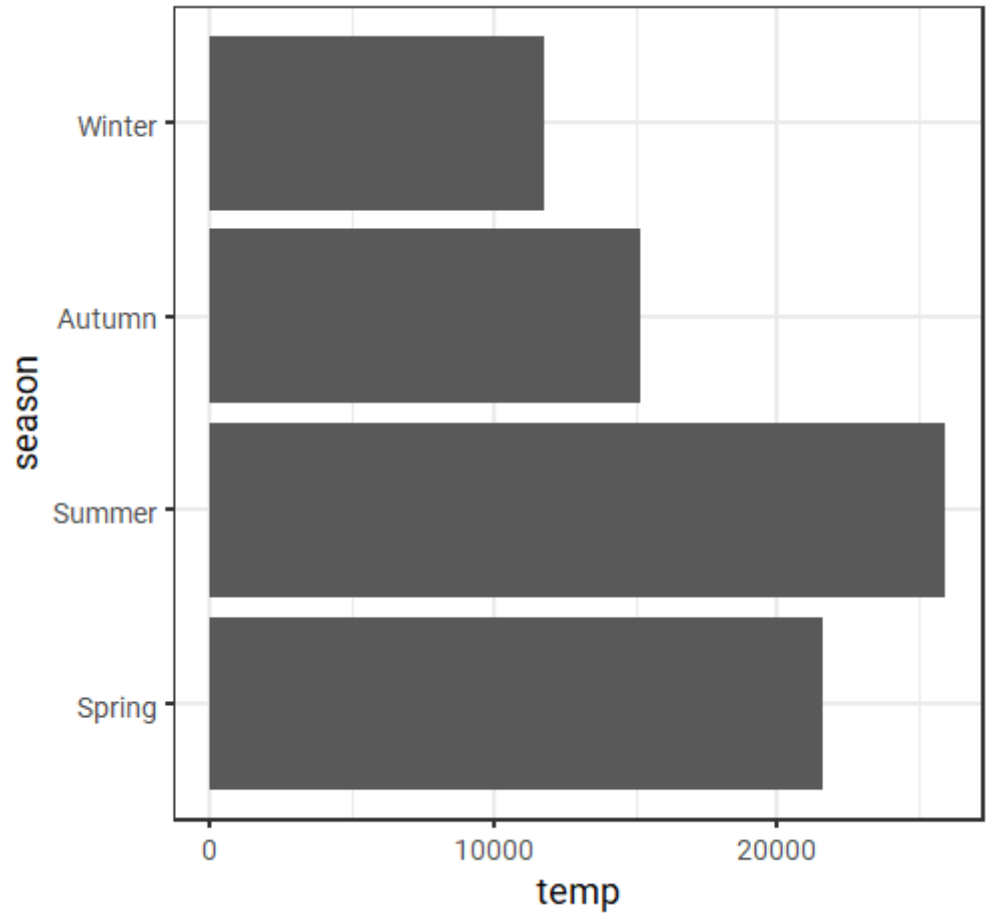




5. Coordinate System: `coord_flip()`

`coord_flip()` allows you to flip a Cartesian coordinate system:

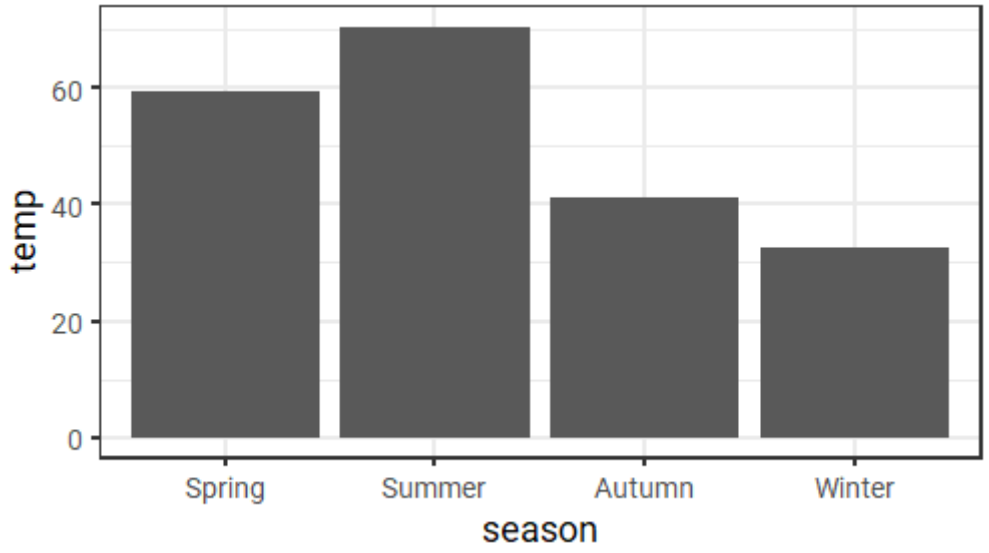
```
ggplot(chic, aes(season, temp)) +
  geom_col() +
  coord_flip()
```



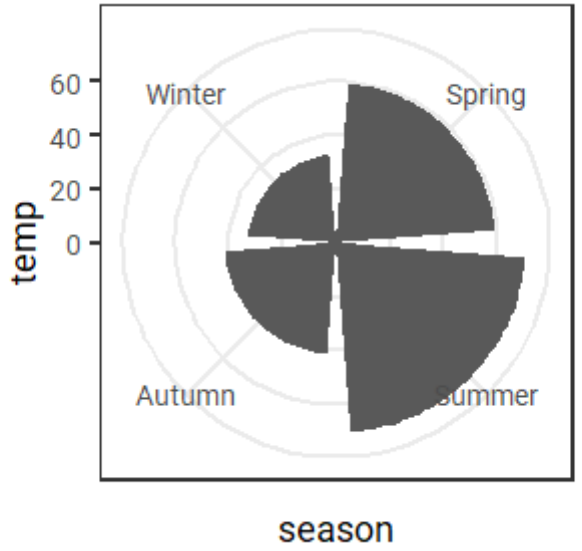
5. Coordinate System: `coord_polar()`

You can easily transform a rectangular coordinate system into a polar one:

```
ggplot(chic, aes(season, temp)) +
  stat_summary(fun.y = mean,
               geom = "bar")
```



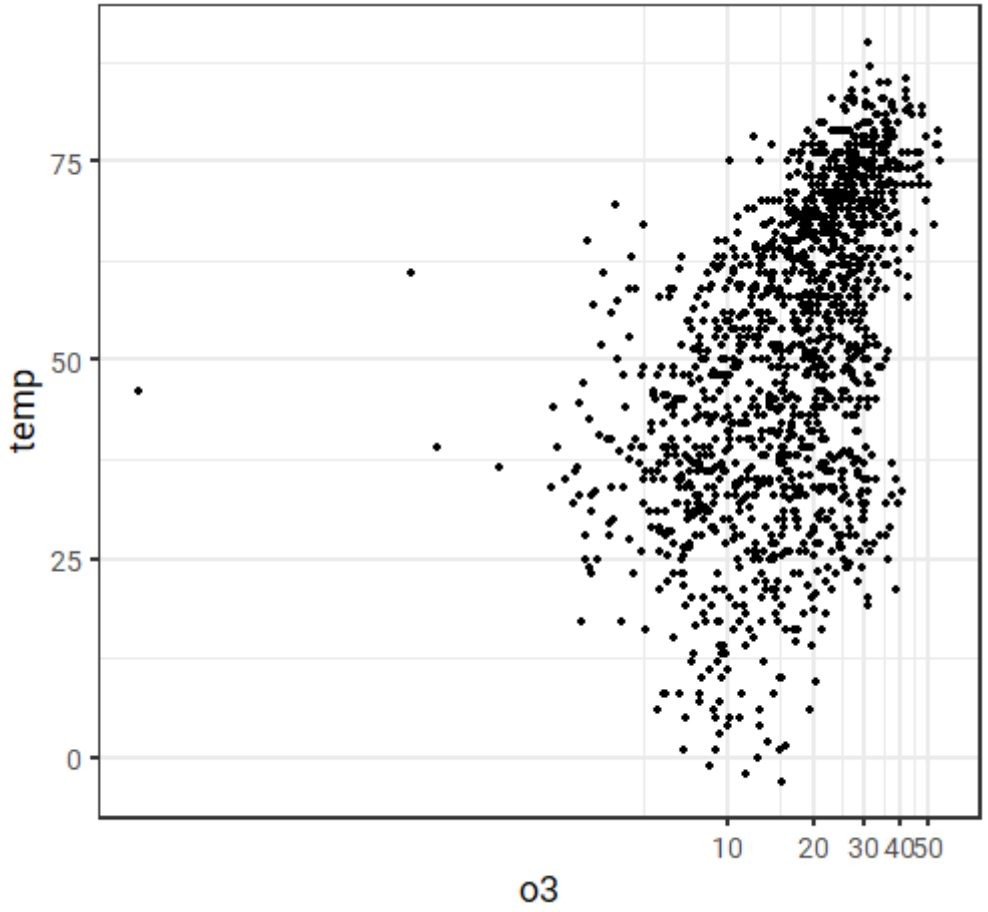

```
ggplot(chic, aes(season, temp)) +
  stat_summary(fun.y = mean,
               geom = "bar") +
  coord_polar()
```



5. Coordinate System: `coord_trans()`

The difference between transforming the scales and transforming the coordinate system is that coordinate transformation occurs **after** the statistics:

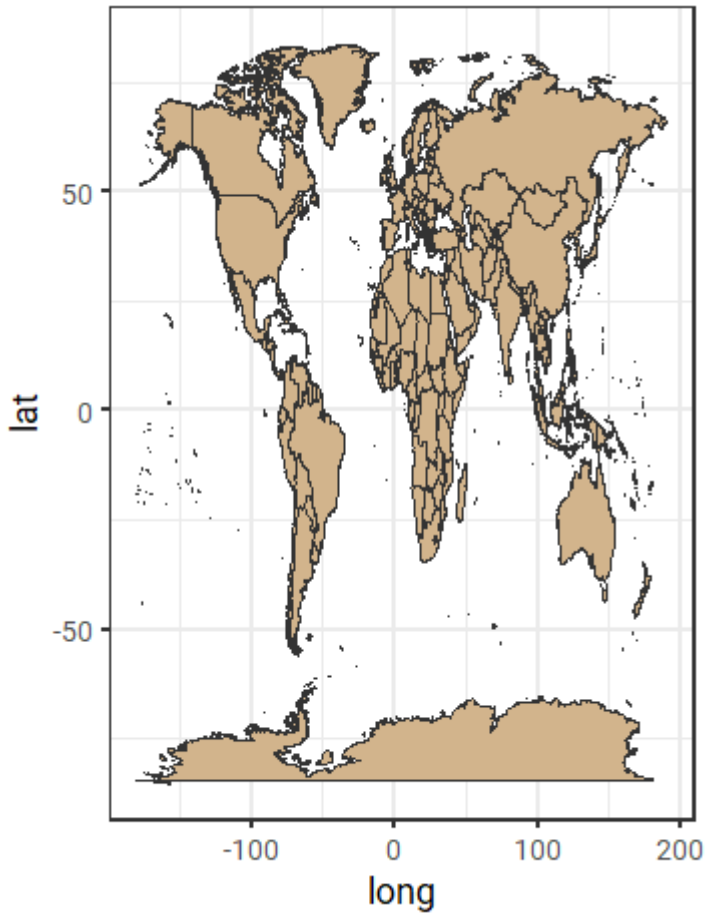
```
ggplot(chic, aes(o3, temp)) +
  geom_point() +
  coord_trans(x = "log2")
```



5. Coordinate System: `coord_map()`

Maps often come as polygons and can be plotted via `geom_polygon()`:

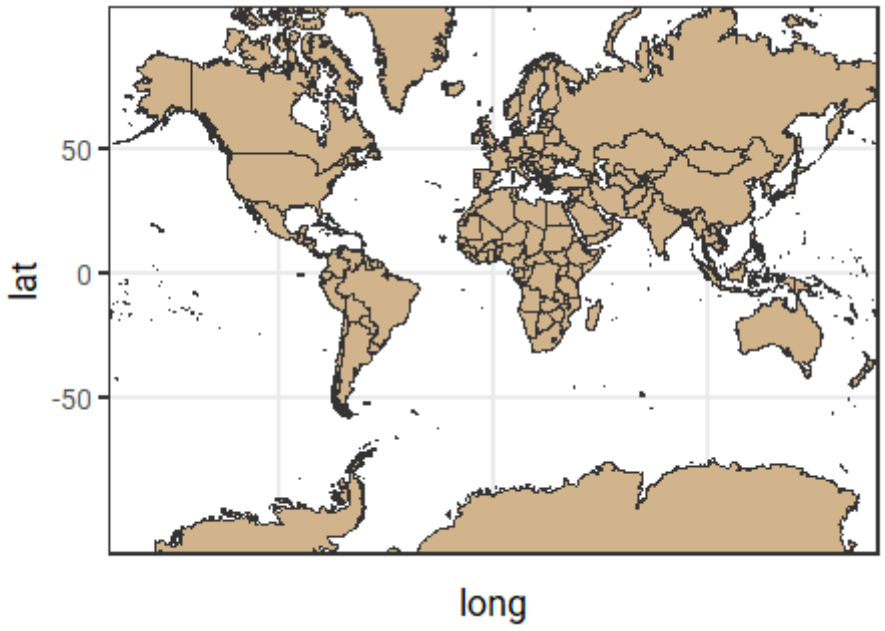
```
(g_world <-
  ggplot(map_data("world"),
         aes(long, lat,
             group = group)) +
    geom_polygon(
      fill = "tan",
      color = "grey20"
    )
)
```



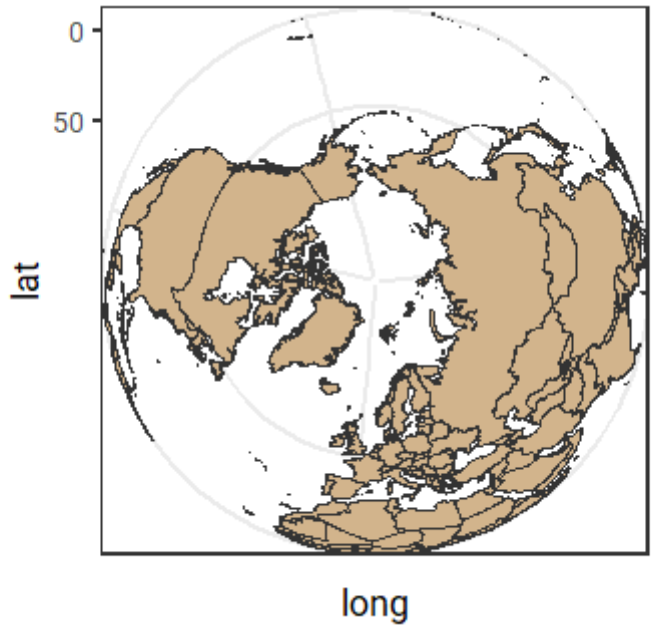
5. Coordinate System: `coord_map()`

Since maps are displaying spherical data, we must project the data via `coord_map()`:

```
g_world +
  coord_map(xlim = c(-180,180))
```




```
g_world +
  coord_map("ortho")
```

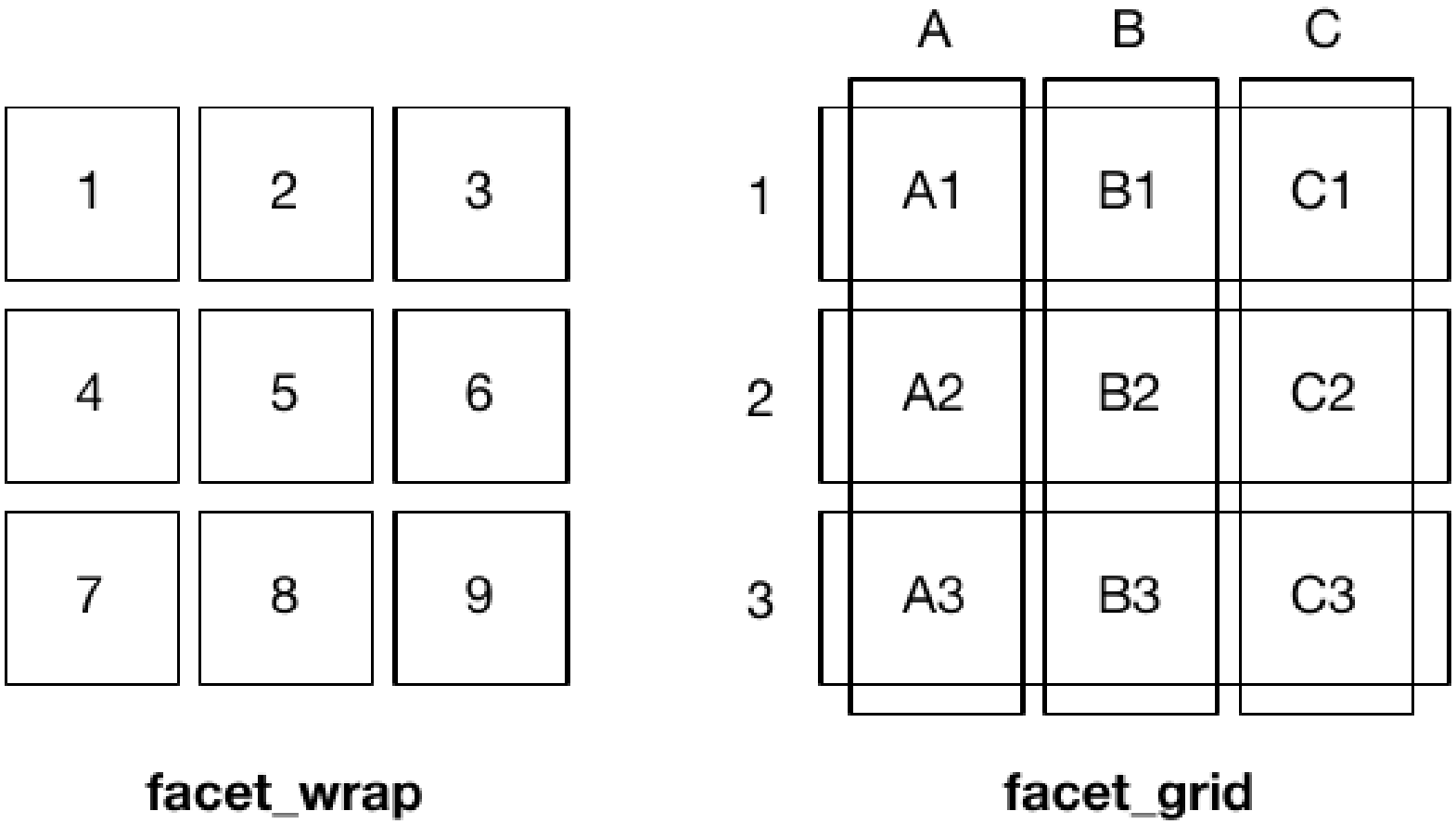


6. Facets

`facet_*()`

6. Facets: `facet_*()`

Facetting generates small multiples each showing a different subset of the data:

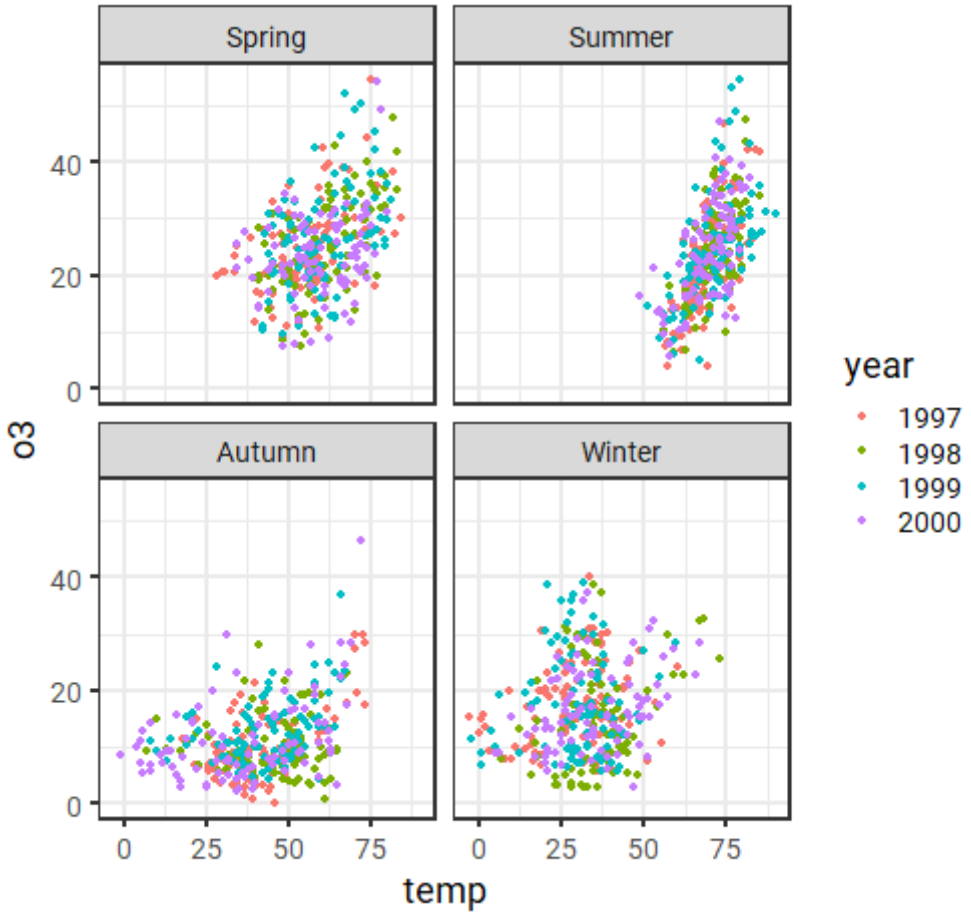


Adapted from "ggplot2: Elegant Graphics for Data Analysis" by Hadley Wickham

6. Facets: `facet_wrap()`

`facet_wrap()` splits the data into small multiples based on *one grouping variable*:

```
ggplot(chic, aes(temp, o3)) +
  geom_point(aes(color = year)) +
  facet_wrap(~ season)
```

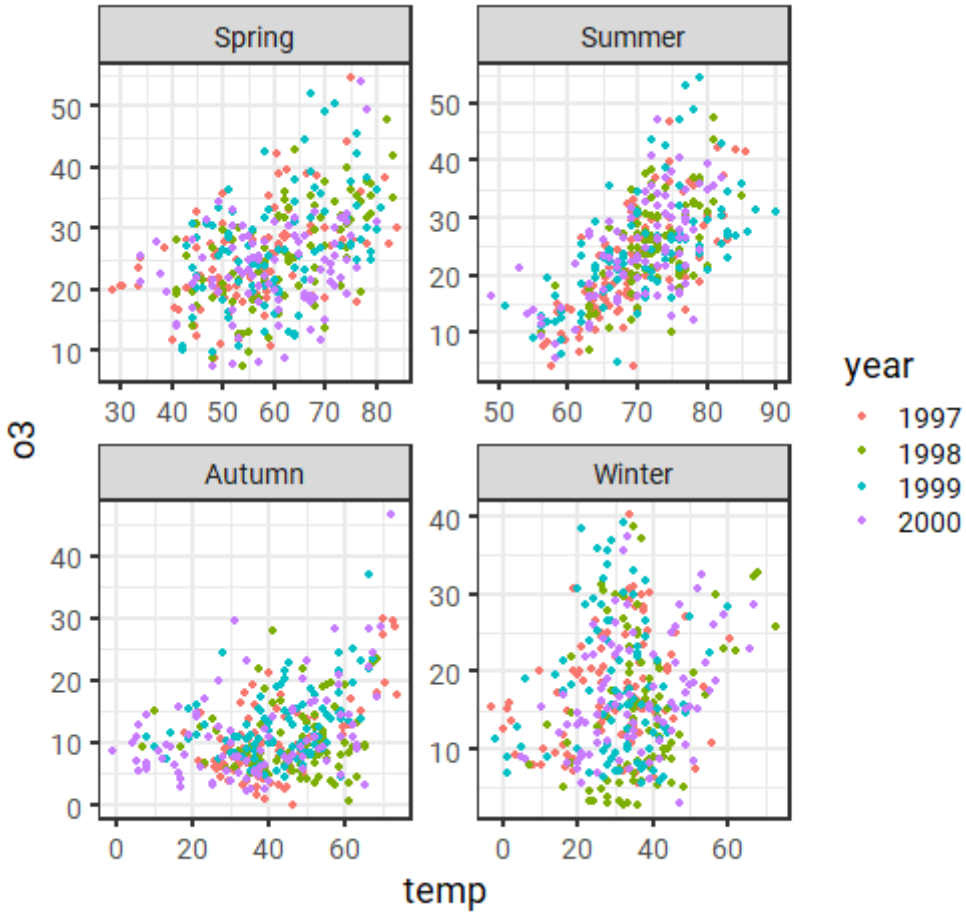


6. Facets: `facet_wrap()`

`facet_wrap()` splits the data into small multiples based on *one grouping variable*:

```
ggplot(chic, aes(temp, o3)) +
  geom_point(aes(color = year)) +
  facet_wrap(~ season,
             scales = "free")
```

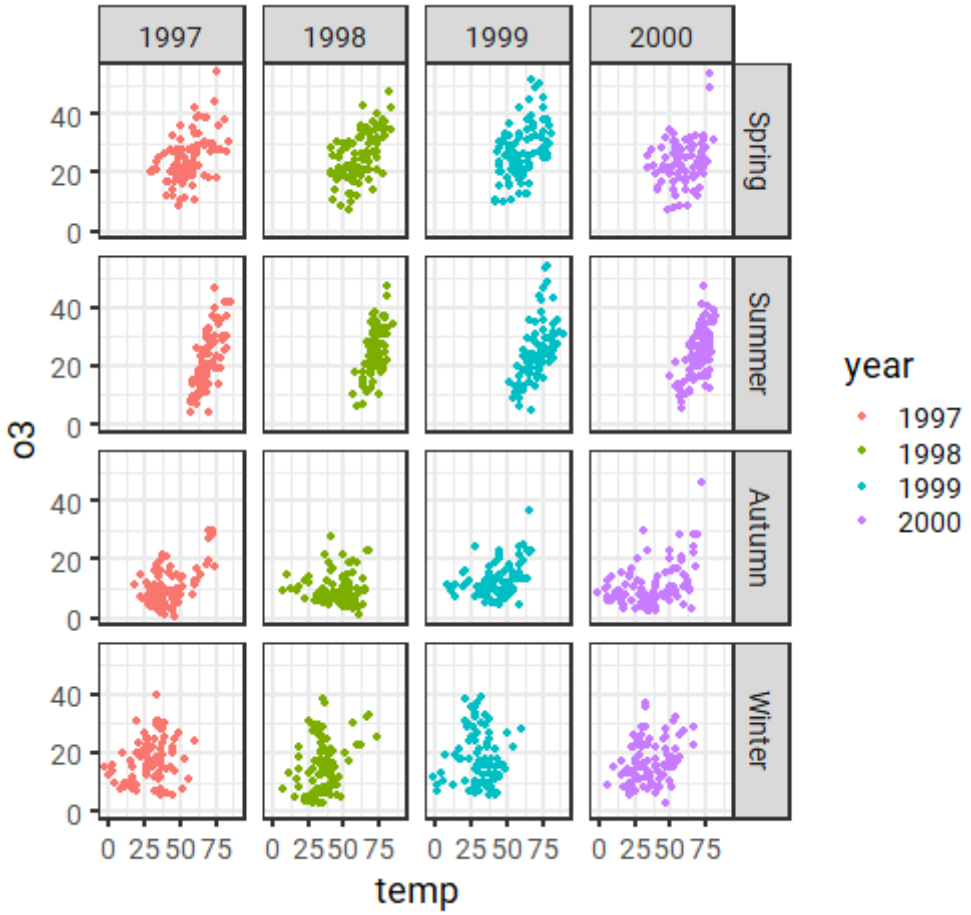
It is possible to change the axes range to scale free for each subset (for only one axis to scale free use `"free_x"` or `"free_y"`.



6. Facets: `facet_grid()`

`facet_grid()` spans a grid of each combination of *two grouping variables*:

```
ggplot(chic, aes(temp, o3)) +
  geom_point(aes(color = year)) +
  facet_grid(season ~ year)
```



7. Visual Themes

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

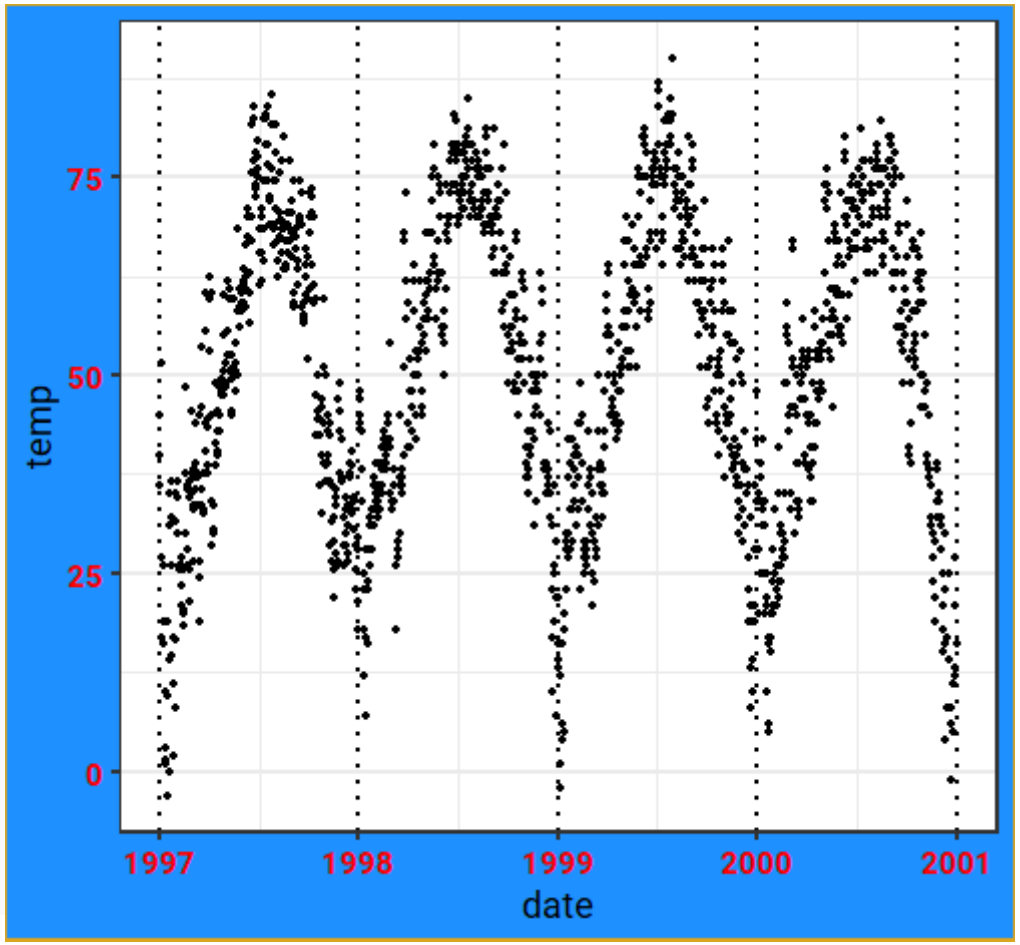
7. Visual Themes: `theme()` and `theme_*()`

- Themes control the display of all non-data elements of the plot
    - titles
    - labels
    - fonts
    - background
    - gridlines
    - legends
    - base size!
- 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()`

7. Visual Themes: `theme()`

To modify the theme of a plot use `theme()` in combination with `element_*()`:

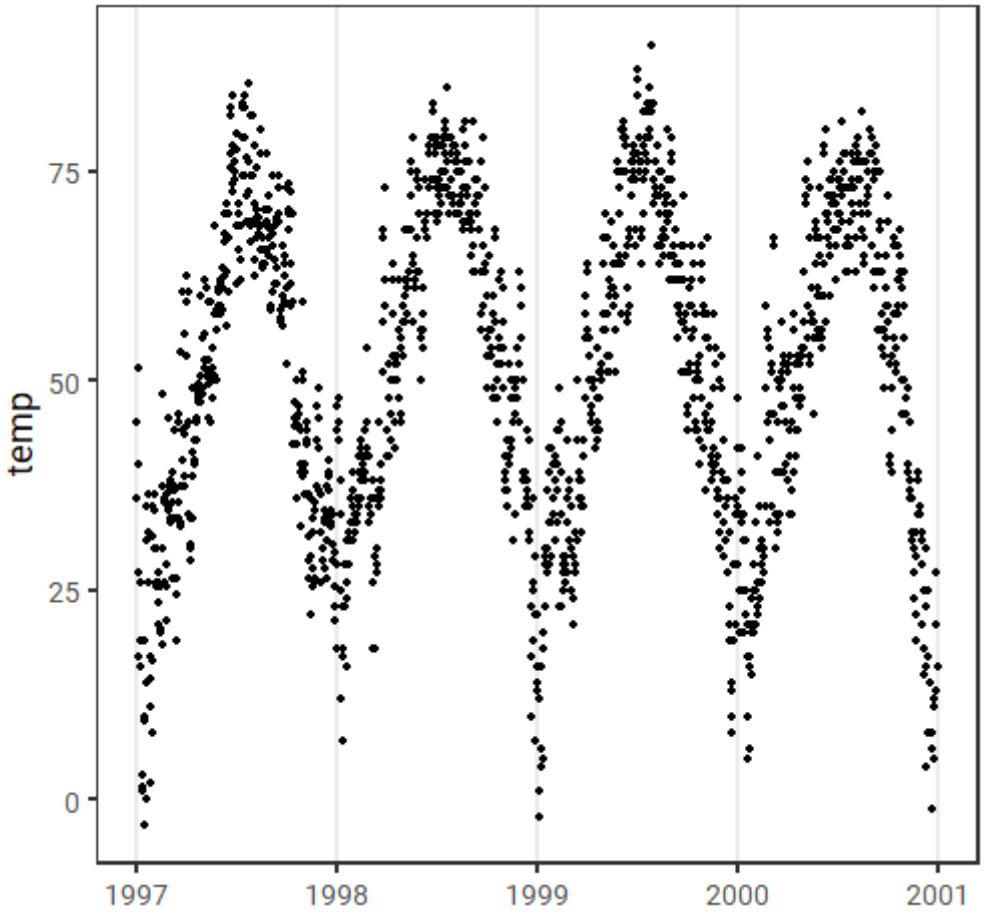
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  theme(
    axis.text = element_text(
      size = 15,
      face = "bold",
      color = "red"
    ),
    panel.grid.major.x = element_line(
      linetype = "dotted",
      color = "black"
    ),
    plot.background = element_rect(
      fill = "dodgerblue",
      color = "goldenrod"
    )
  )
```



7. Visual Themes: `theme()`

`element_blank()` can be used to remove certain theme elements:

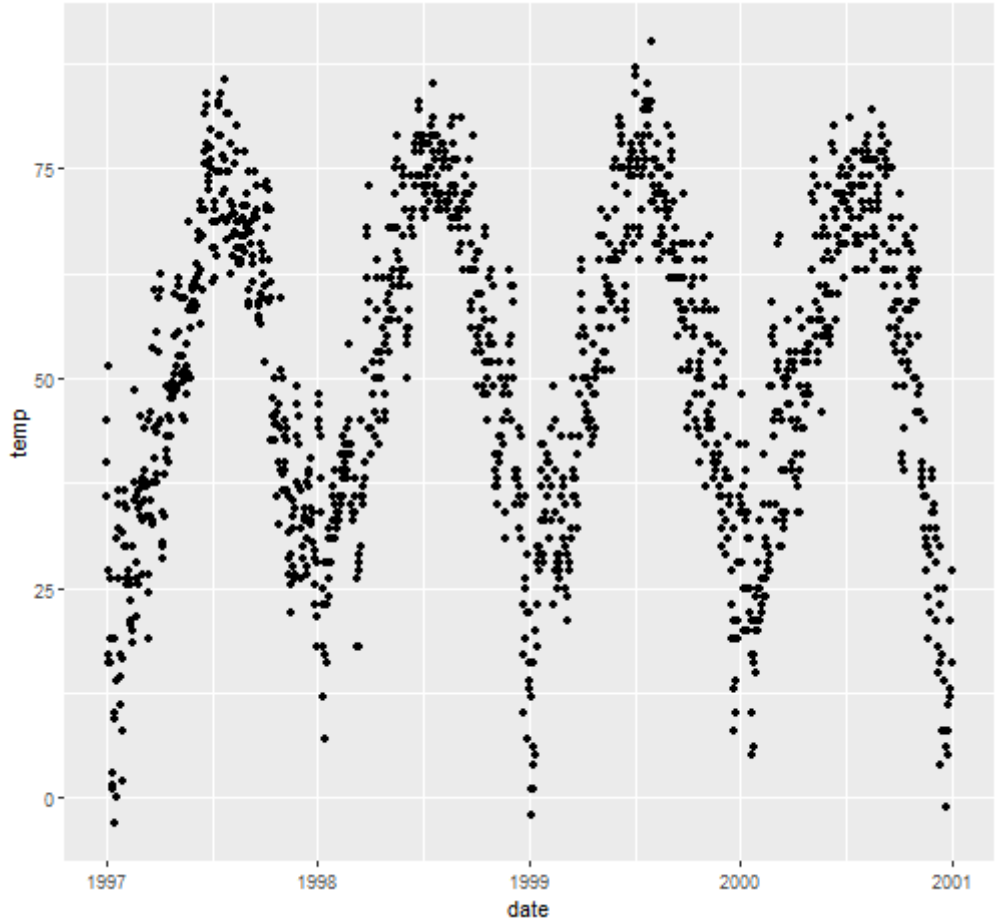
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  theme(
    axis.title.x = element_blank(),
    panel.grid.major.y = element_blank(),
    panel.grid.minor = element_blank()
  )
```



7. Visual Themes: `theme_*()`

Use a built-in theme of **{ggplot2}**:

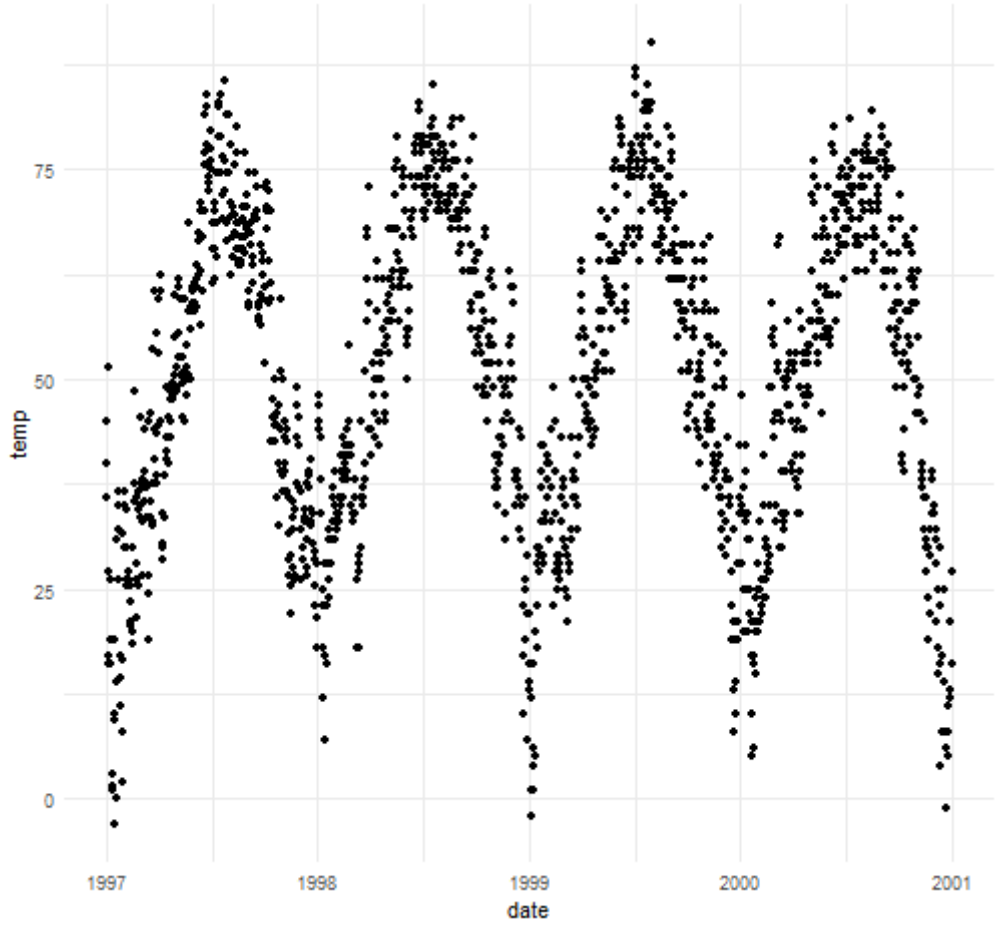
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  theme_gray()
```



7. Visual Themes: `theme_*()`

Use a built-in theme of `{ggplot2}`:

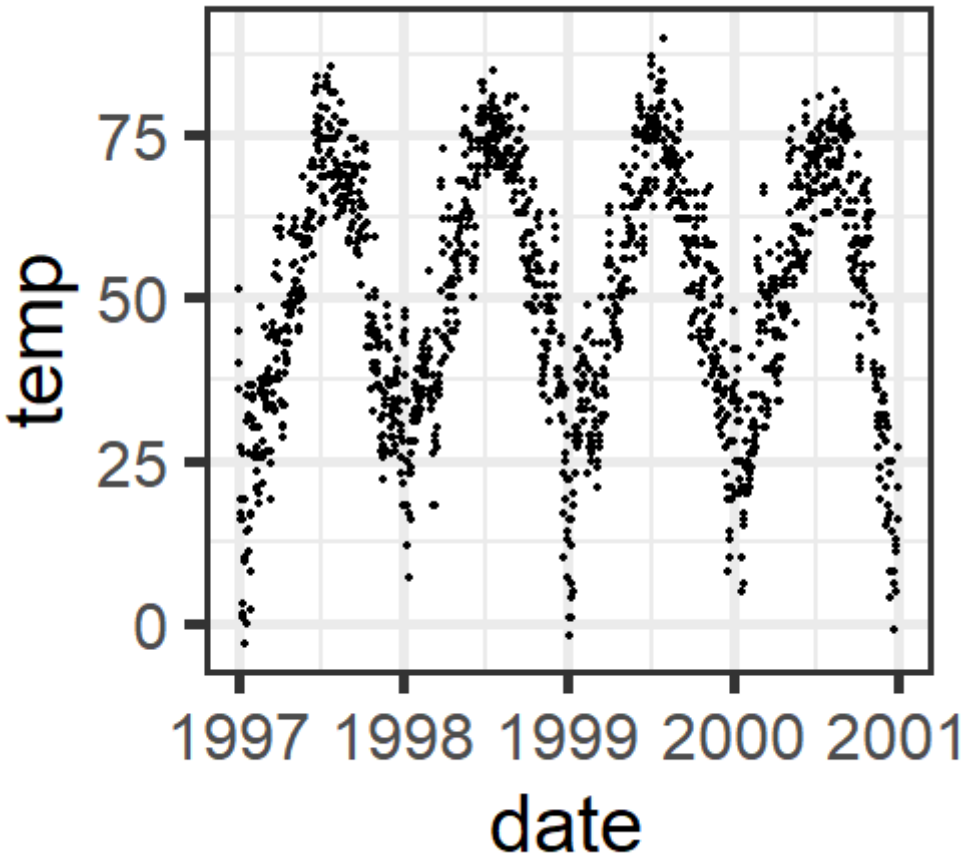
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  theme_minimal()
```



7. Visual Themes: `theme_*()`

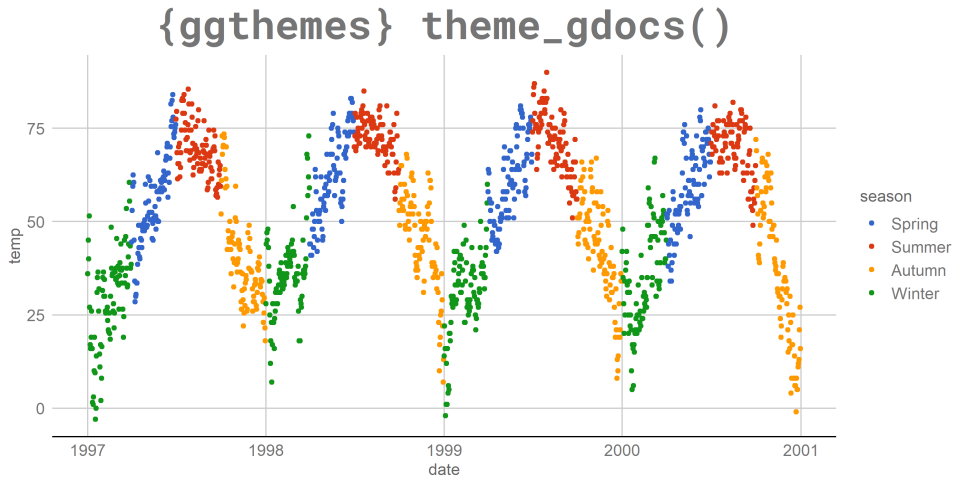
Using the argument `base_size()` you can change the size of the text:

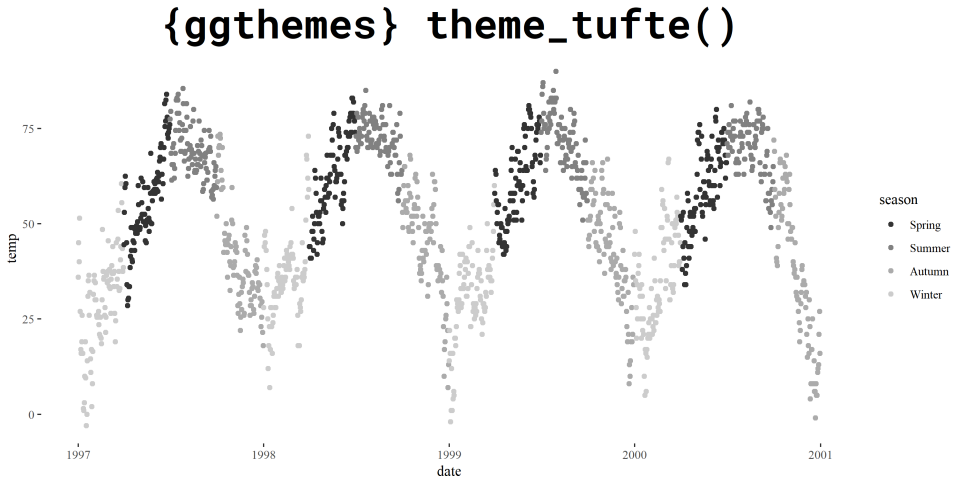
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  theme_bw(base_size = 40)
```

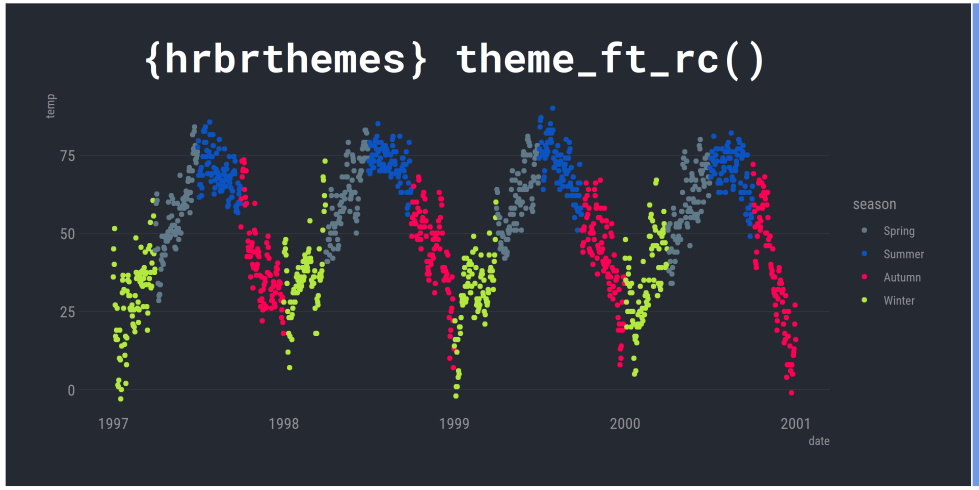


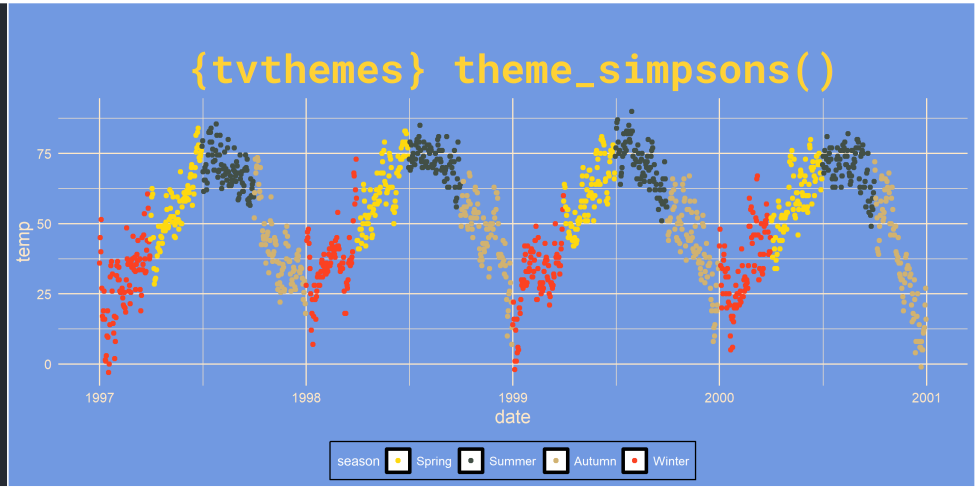
7. Visual Themes: `theme_*()`

There are also extension packages such as `{ggthemes}`, `{hrbrthemes}` and `{tvthemes}`:





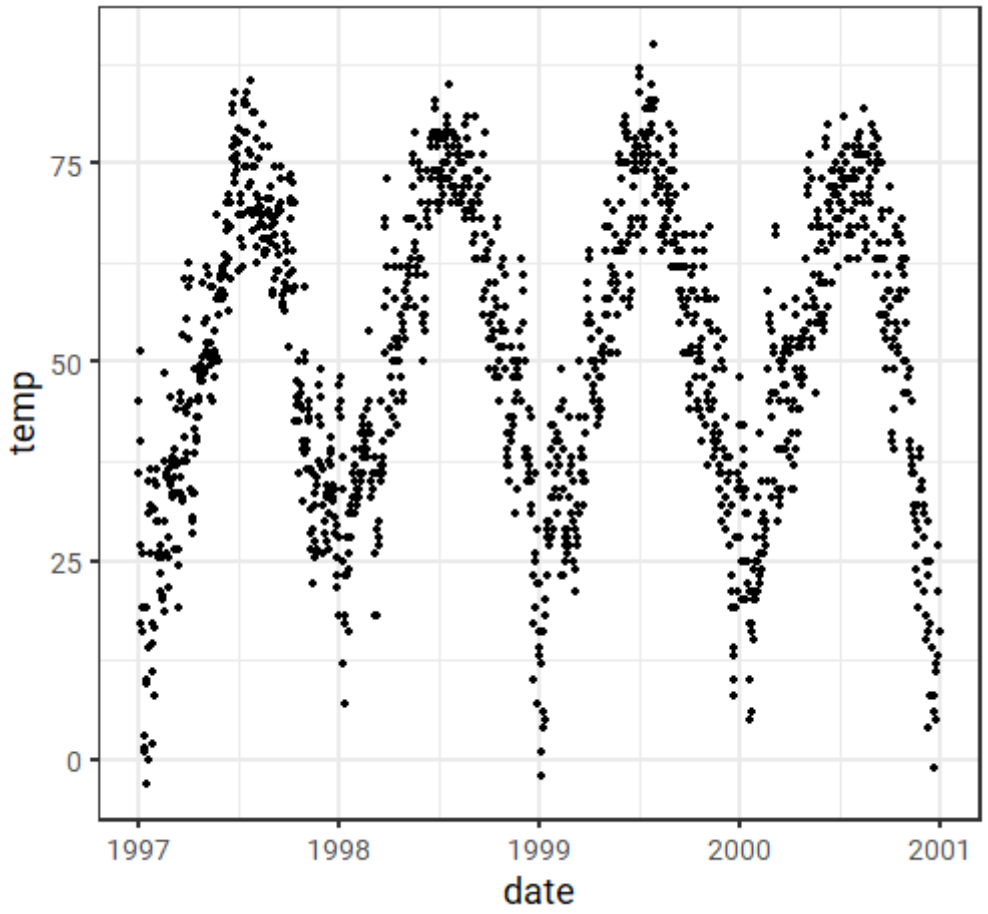




7. Visual Themes: `theme_set()` and `theme_update()`

`theme_set()` and `theme_update()` override settings completely or partly:

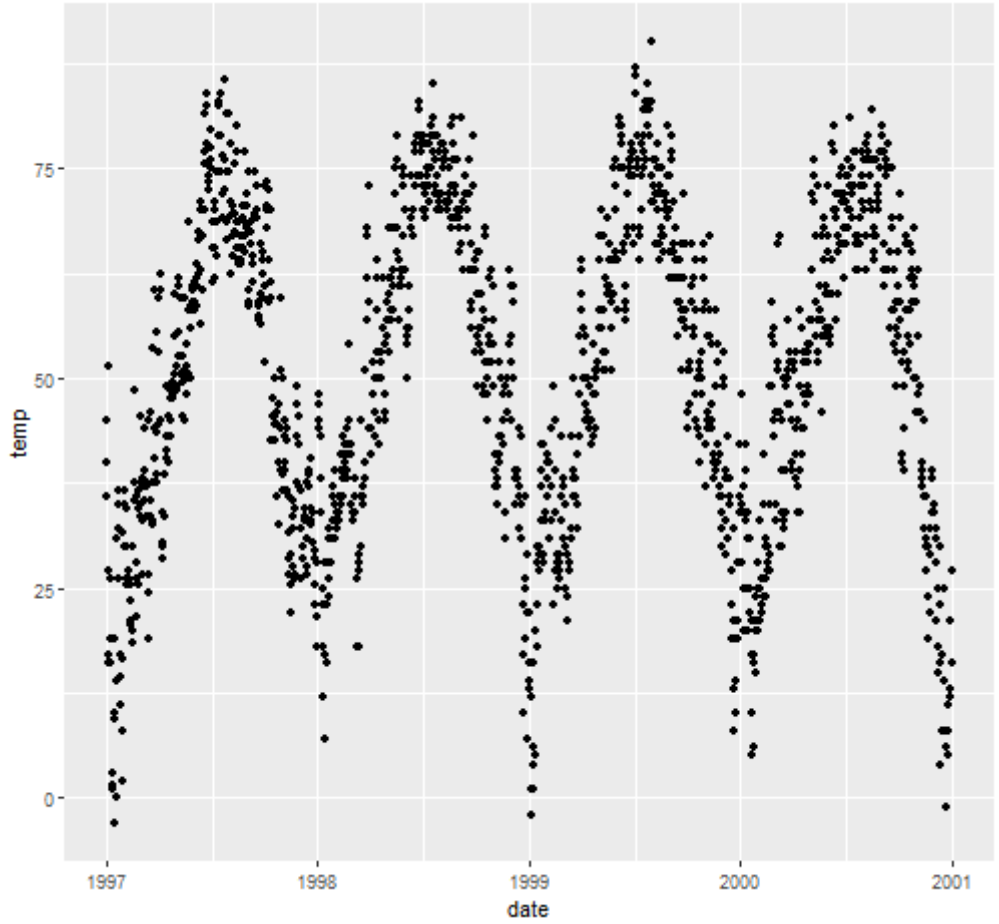
```
(g <- ggplot(chic, aes(date, temp)) +
  geom_point())
```



7. Visual Themes: `theme_set()` and `theme_update()`

`theme_set()` and `theme_update()` override settings completely or partly:

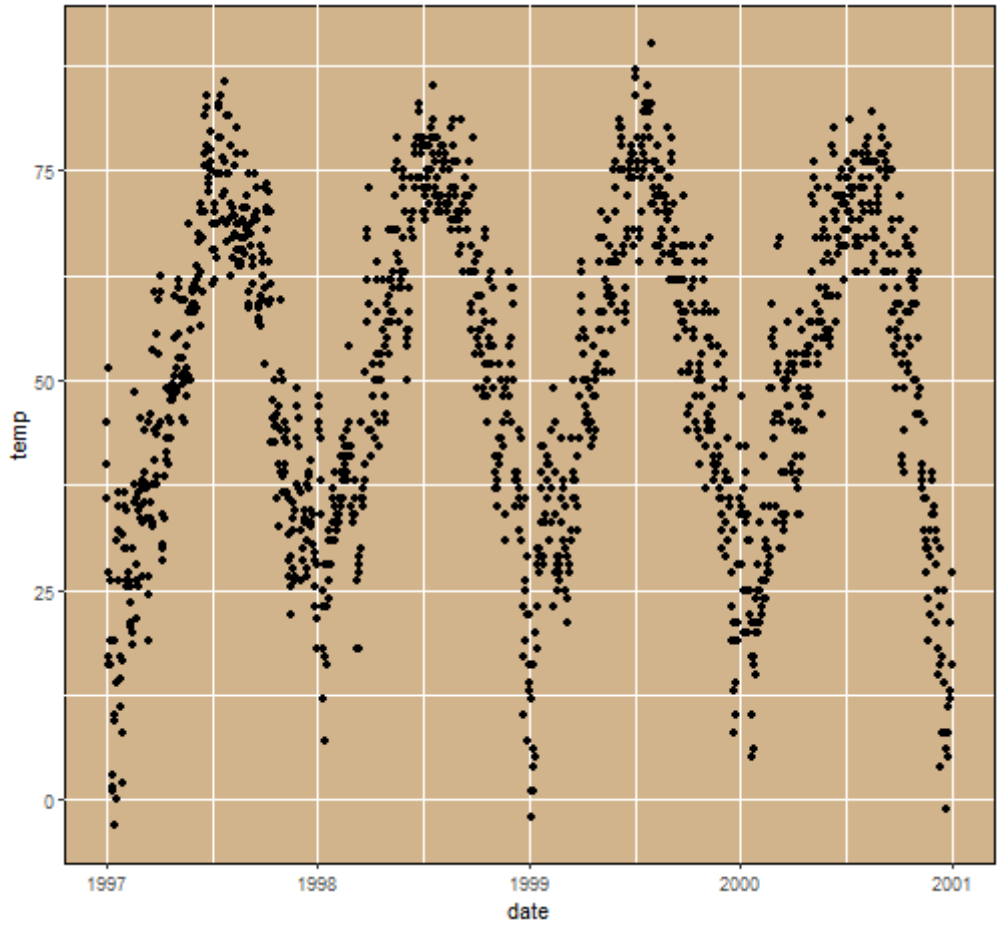
```
(g <- ggplot(chic, aes(date, temp)) +
  geom_point())

old <- theme_set(theme_grey())
g
```



7. Visual Themes: `theme_set()` and `theme_update()`

`theme_set()` and `theme_update()` override settings completely or partly:

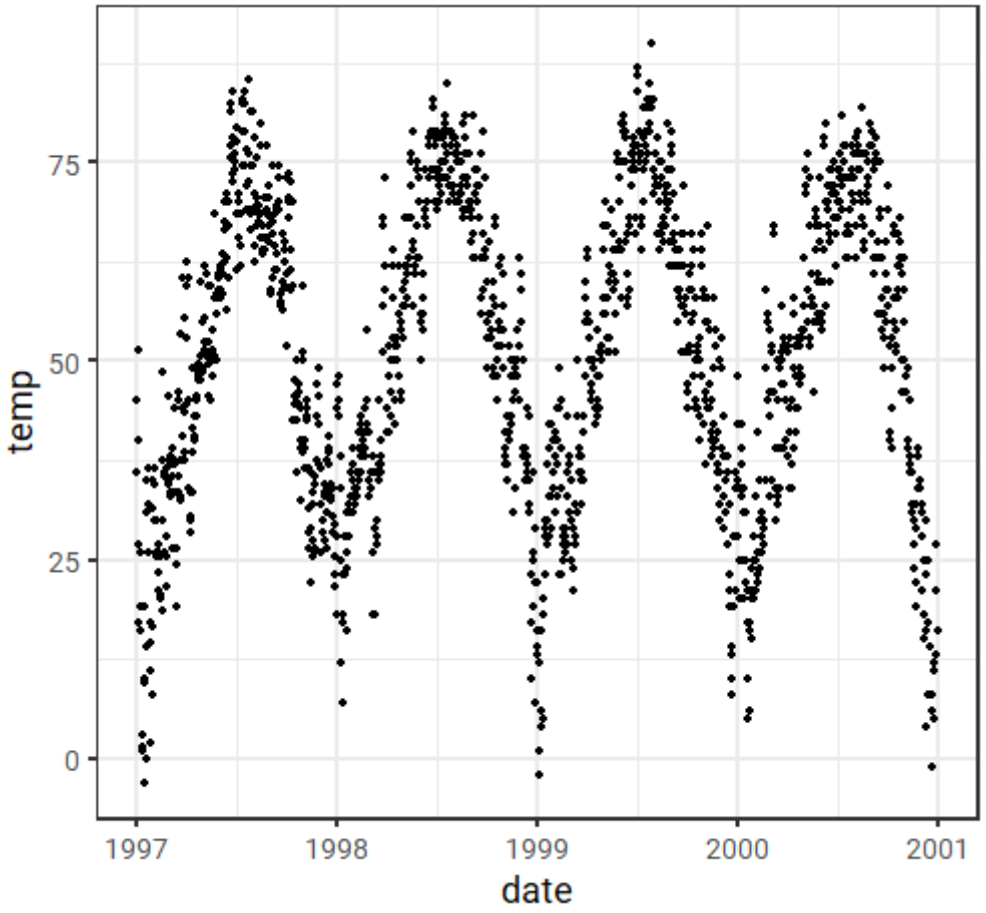
```
(g <- ggplot(chic, aes(date, temp)) +
  geom_point())

old <- theme_set(theme_grey())
g

theme_update(
  panel.background = element_rect(
    fill = "tan",
    color = "black"
  )
)
g
```



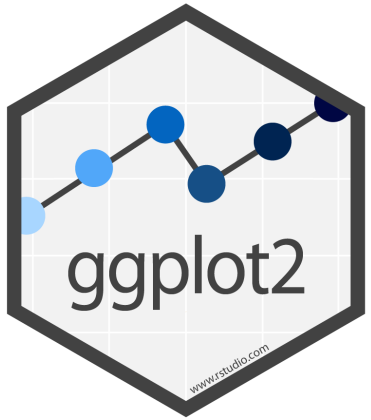
7. Visual Themes: `theme_set()` and `theme_update()`

`theme_set()` and `theme_update()` override settings completely or partly:

```
(g <- ggplot(chic, aes(date, temp)) +
  geom_point())

old <- theme_set(theme_grey())
g

theme_update(
  panel.background = element_rect(
    fill = "tan",
    color = "black"
  )
)
g

theme_set(old)
g
```



Some more?



Ressources

- **Free Online Books:**
  - "ggplot2: Elegant Graphics for Data Analysis" by Hadley Wickham → ggplot2-book.org
  - "R for Data Science" by Hadley Wickham → r4ds.had.co.nz
  - "Data Visualization: A Practical Introduction" by Kieran Healy → socviz.co
  - "Fundamentals of Data Visualization" by Claus Wilke → serialmentor.com/dataviz
- **Galleries:**
  - R Graph Gallery → r-graph-gallery.com
  - ggplot2 Extension Gallery → ggplot2-exts.org/gallery
  - Color Palettes in R → github.com/EmilHvitfeldt/r-color-palettes
  - #TidyTuesday Collection → tidytuesday.rocks
- **Get Help:**
  - RStudio Community → community.rstudio.com
  - R4DS Learning Community → rfordatasci.com
  - #TidyTuesday Community → #TidyTuesday on Twitter
- **Useful packages:**
  - `{ggtext}` — rich-text rendering → github.com/clauswilke/ggtext
  - `{ggforce}` — several interesting add-on features → ggforce.data-imaginist.com
  - `{ggridges}` — geoms for joy plots → github.com/clauswilke/ggridges
  - `{ggrepel}` — prevent overlapping text labels → github.com/slowkow/ggrepel
  - `{gganimate}` — create animations → gganimate.com
  - `{ggplotly}` — create interactive plots → plot.ly/ggplot2
  - `{ggraph}` — networks, graphs & trees → github.com/thomasp85/ggraph
  - `{ggalt}` — alternative coords, geoms, stats & scales → github.com/hrbrmstr/ggalt
  - `{ggpubr}` — publication-ready plot in one line → github.com/kassambara/ggpubr
  - `{ggmaps}` — access to Google & Stamen maps → github.com/dkahle/ggmap
  - `{ggthemes}` — additional themes, sclaes & geoms → github.com/jrnold/ggthemes
  - `{hrbrthemes}` — typography-centric themes → github.com/hrbrmstr/hrbrthemes
  - `{lemon}` — axis & legend add-ons → github.com/stefanedwards/lemon
  - `{cowplot}` — plot arrangements, themes & annotations → wilkelab.org/cowplot
  - `{patchwork}` — combine ggplots → github.com/thomasp85/patchwork
  - `{showtext}` — use custom fonts → github.com/yixuan/showtext
  - `{rayshader}` — hillshaded maps in 2D & 3D → github.com/tylermorganwall/rayshader

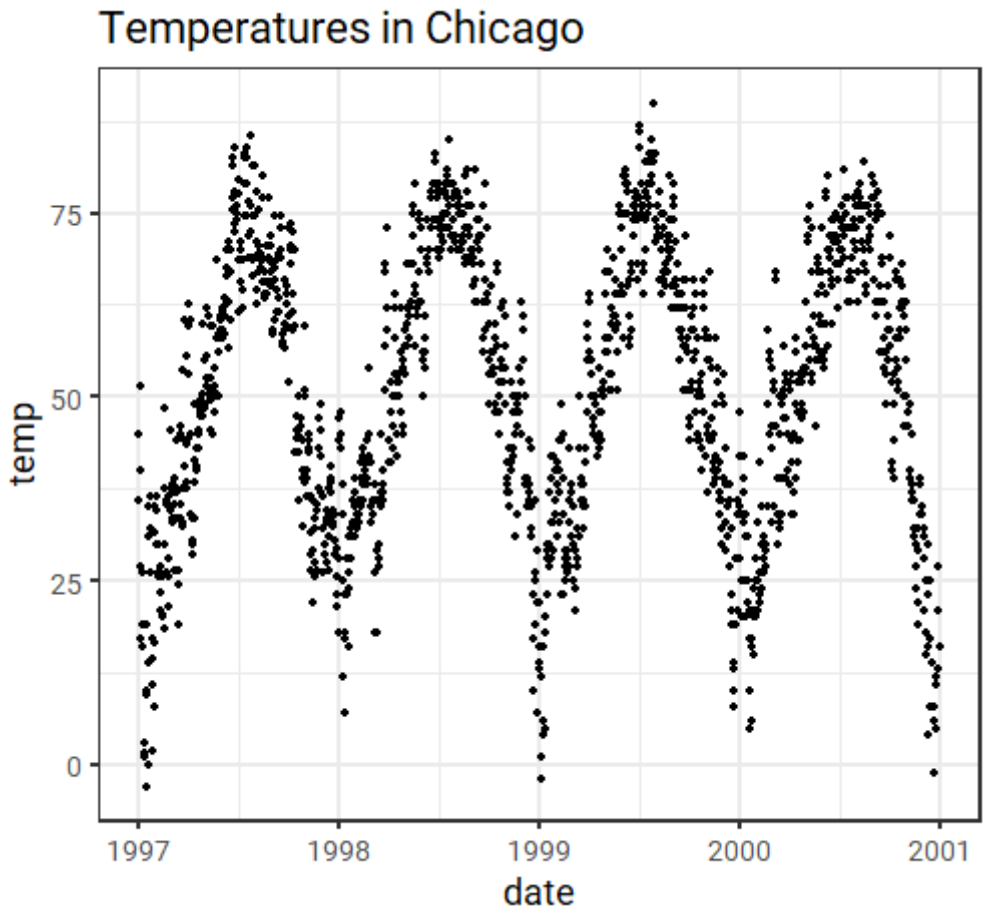
You want even more?

Working with Text

Working with Text: Title, Subtitle, Caption, and Tag

To quickly add a title, use `ggtitle()`:

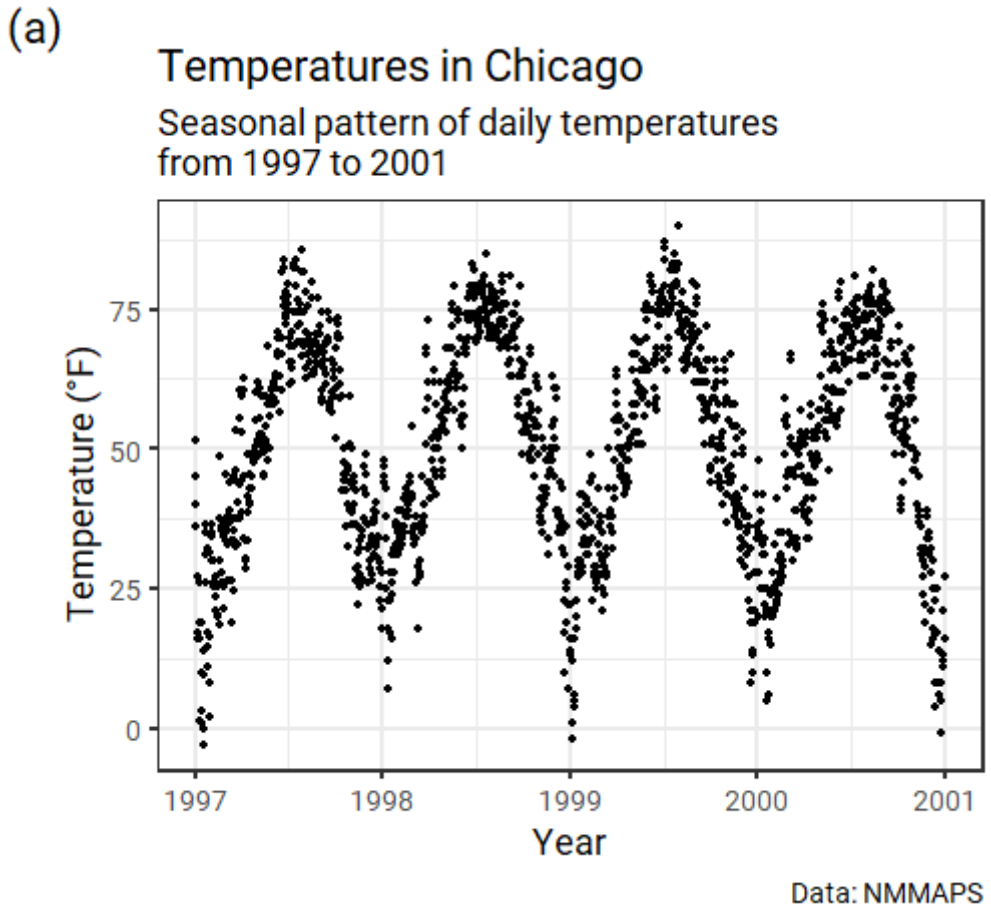
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  ggtitle("Temperatures in Chicago")
```



Working with Text: Title, Subtitle, Caption, and Tag

{ggplot2} has a built-in structure for title, subtitle, caption and tags via **labs()**:

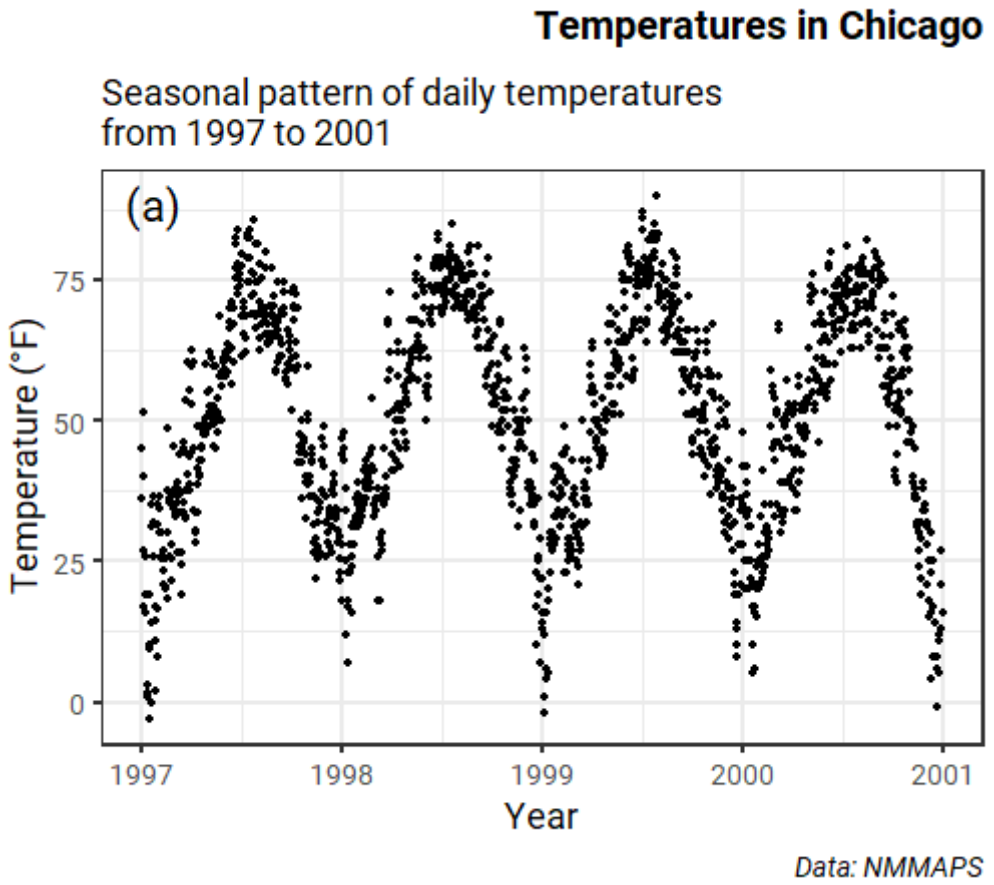
```
(g <- ggplot(chic, aes(date, temp)) +
  geom_point() +
  labs(
    x = "Year",
    y = "Temperature (°F)",
    title = "Temperatures in Chicago",
    subtitle = "Seasonal pattern of daily
    caption = "Data: NMMAPS",
    tag = "(a)"
  )
)
```



Working with Text: Title, Subtitle, Caption, and Tag

{ggplot2} has a built-in structure for title, subtitle, caption and tags:

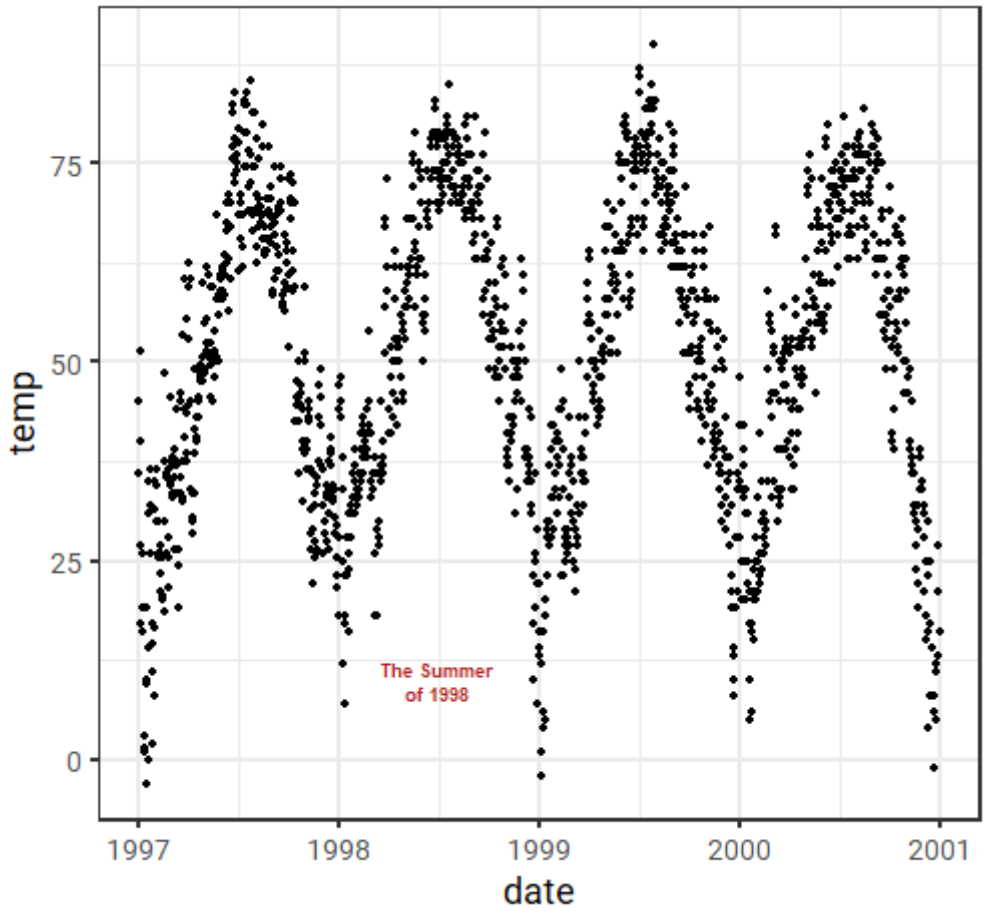
```
g +
  theme(
    plot.title = element_text(
      size = 20,
      face = "bold",
      hjust = 1,
      margin = margin(15, 0, 15, 0)
    ),
    plot.caption = element_text(
      face = "italic"
    ),
    plot.tag.position = c(0.15, 0.75)
  )
```



Working with Text: `annotate("text")`

`annotate()` allows to add text inside the plot area:

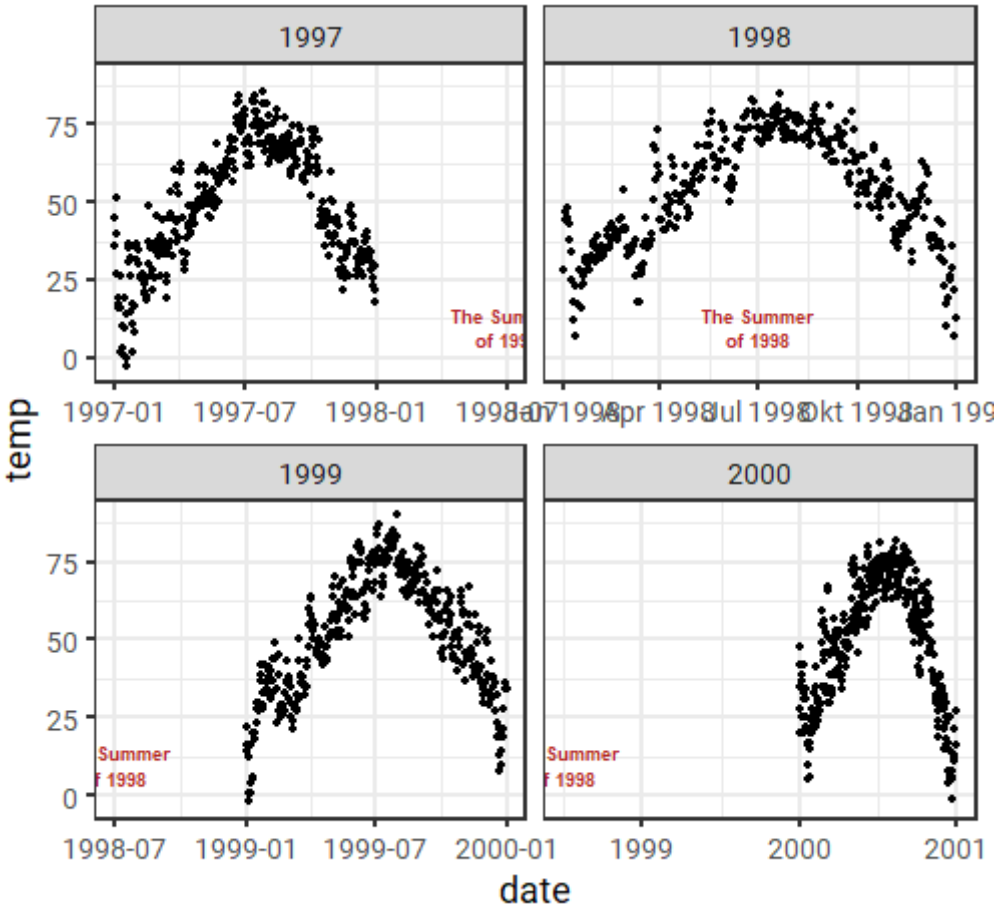
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  annotate(
    "text",
    x = as.Date("1998-07-01"),
    y = 10,
    label = "The Summer\nof 1998",
    size = 3,
    fontface = "bold",
    color = "firebrick"
  )
```



Working with Text: `annotate("text")`

The `annotate()` function comes from ggplot2 and is designed to use a so-called **grob** as input:

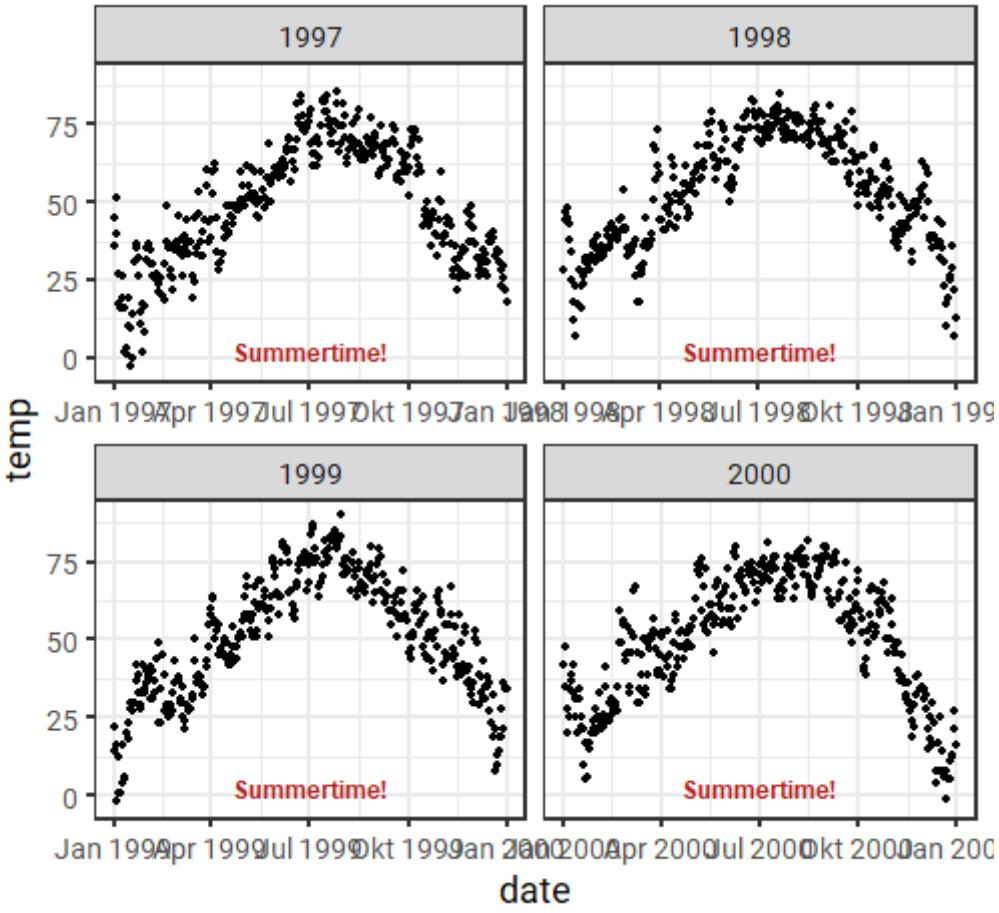
```
ggplot(chic, aes(date, temp)) +
  geom_point() +
  annotate(
    "text",
    x = as.Date("1998-07-01"),
    y = 10,
    label = "The Summer\nof 1998",
    size = 3,
    fontface = "bold",
    color = "firebrick"
  ) +
  facet_wrap(~year, scales = "free_x")
```



Working with Text: `annotation_custom(grob)`

`annotation_custom()` allows to add text by relative positions which solves the problem:

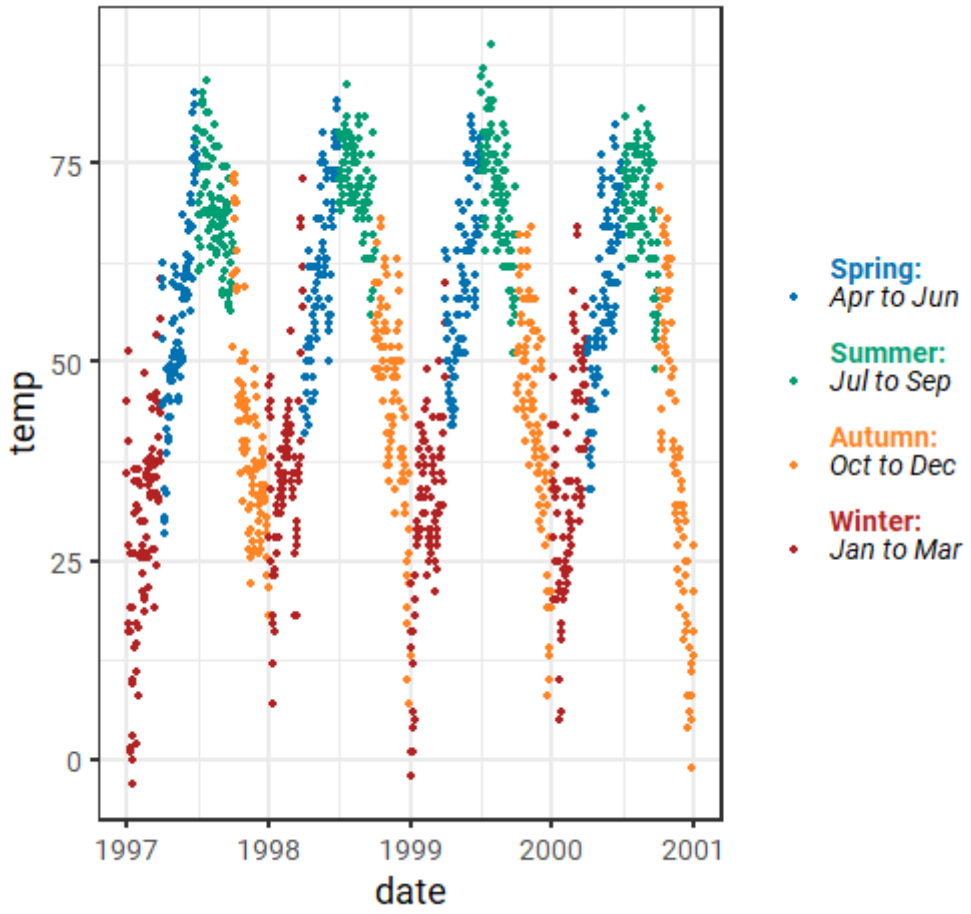
```
text <- grid::grobTree(
  grid::textGrob(
    "Summertime!",
    x = 0.5, y = 0.1, hjust = 0.5,
    gp = grid::gpar(
      fontsize = 12,
      fontface = "bold",
      col = "firebrick"
    )
  )
)

ggplot(chic, aes(date, temp)) +
  geom_point() +
  annotation_custom(text) +
  facet_wrap(~ year, scales = "free_x")
```



Working with Text: `element_markdown()` via `{ggtext}`

With the new `{ggtext}` package, it is possible to use Markdown and basic HTML within strings:

```
library(ggtext)

ggplot(chic, aes(date, temp)) +
  geom_point(aes(color = season)) +
  scale_color_manual(
    name = NULL,
    values = c("#0072B2", "#009E73", "#FF8
    labels = c(
      "<b style='color:#0072B2'>Spring:<br
      "<b style='color:#009E73'>Summer:</b
      "<b style='color:#FF8423'>Autumn:</b
      "<b style='color:#B22222'>Winter:</b
    )
  ) +
  theme(legend.text = element_markdown())
```

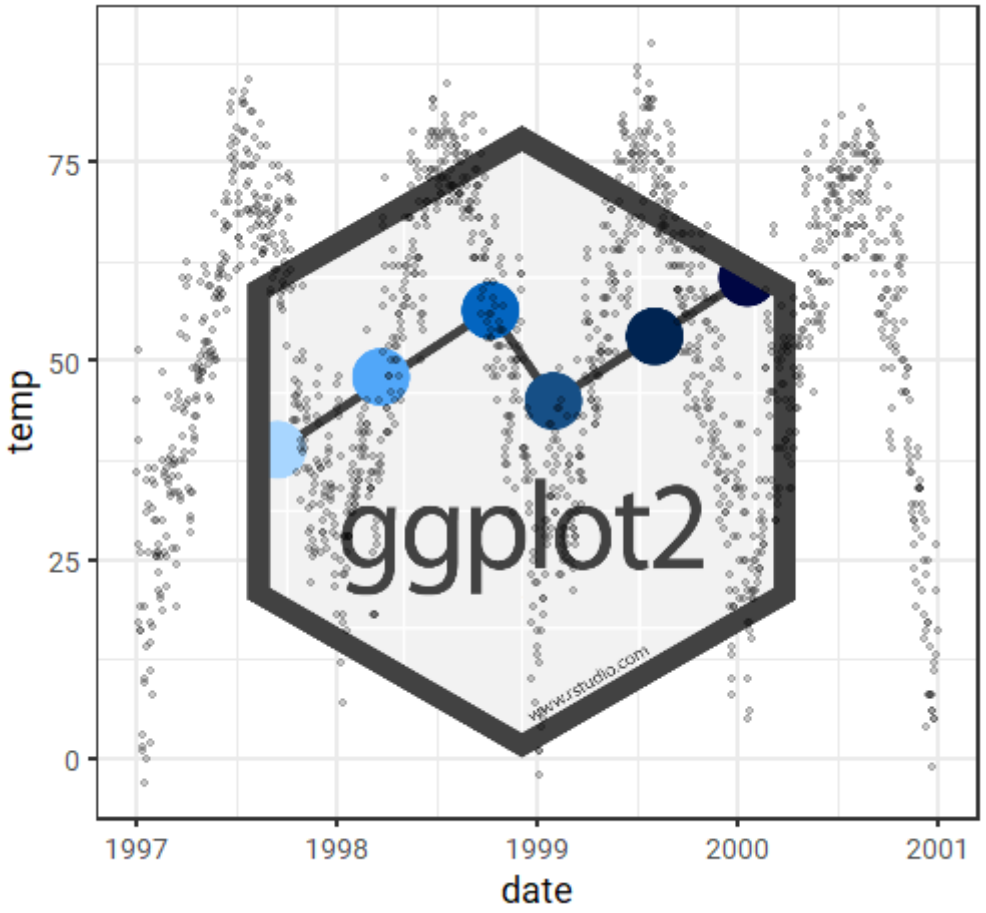


Working with Images

Working with Images: `annotation_custom(grob)`

The `annotation_custom()` function comes with ggplot2 and is designed to use a so-called **grob** as input:

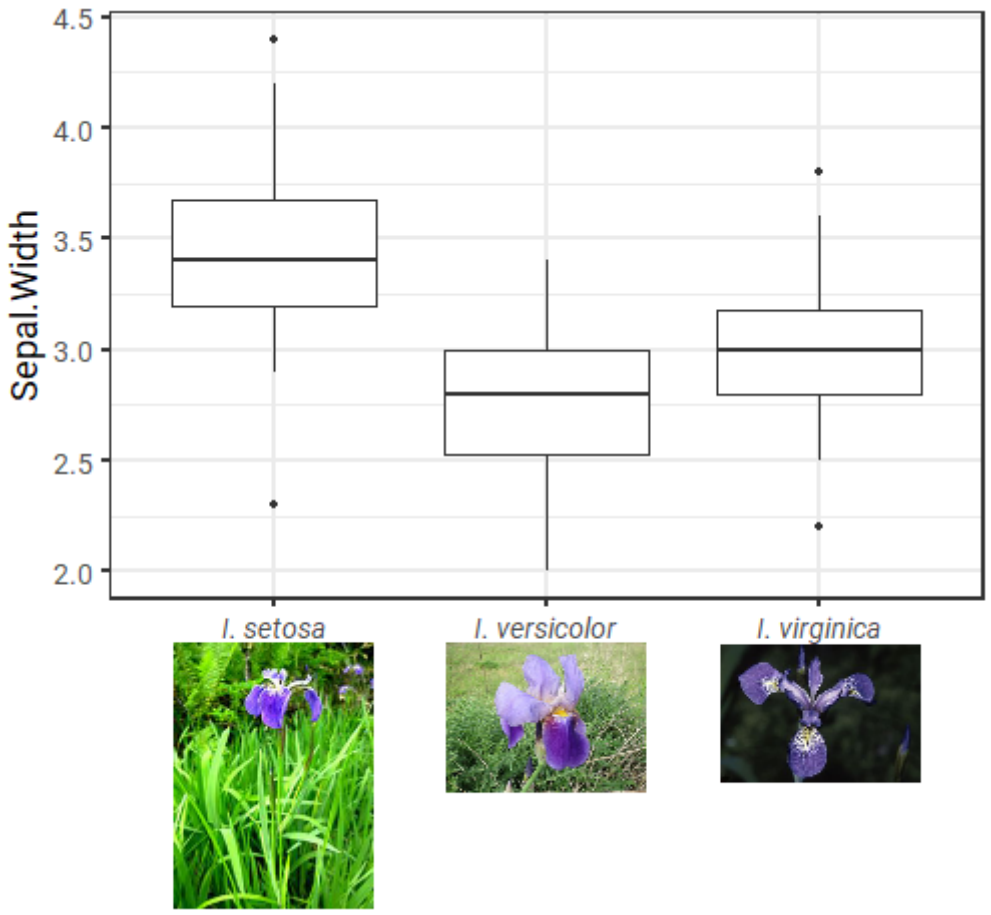
```
img <- png::readPNG("./img/logo.png")
logo <- grid::rasterGrob(img, interpolate

ggplot(chic, aes(date, temp)) +
  annotation_custom(
    logo,
    xmin = as.Date("1997-06-01"),
    xmax = as.Date("2000-06-01"),
    ymin = 0,
    ymax = 80
  ) +
  geom_point(alpha = 0.2)
```



Working with Images: `element_markdown()` via `{ggtext}`

It also has basic support for the html tag:

```
ggplot(iris, aes(Species, Sepal.Width)) +
  geom_boxplot() +
  scale_x_discrete(
    name = NULL,
    labels = c(
      setosa = "*I. setosa*<br><img src='h
      virginica = "*I. virginica*<br><img
      versicolor = "*I. versicolor*<br><im
    )
  ) +
  theme(axis.text.x = element_markdown())
```

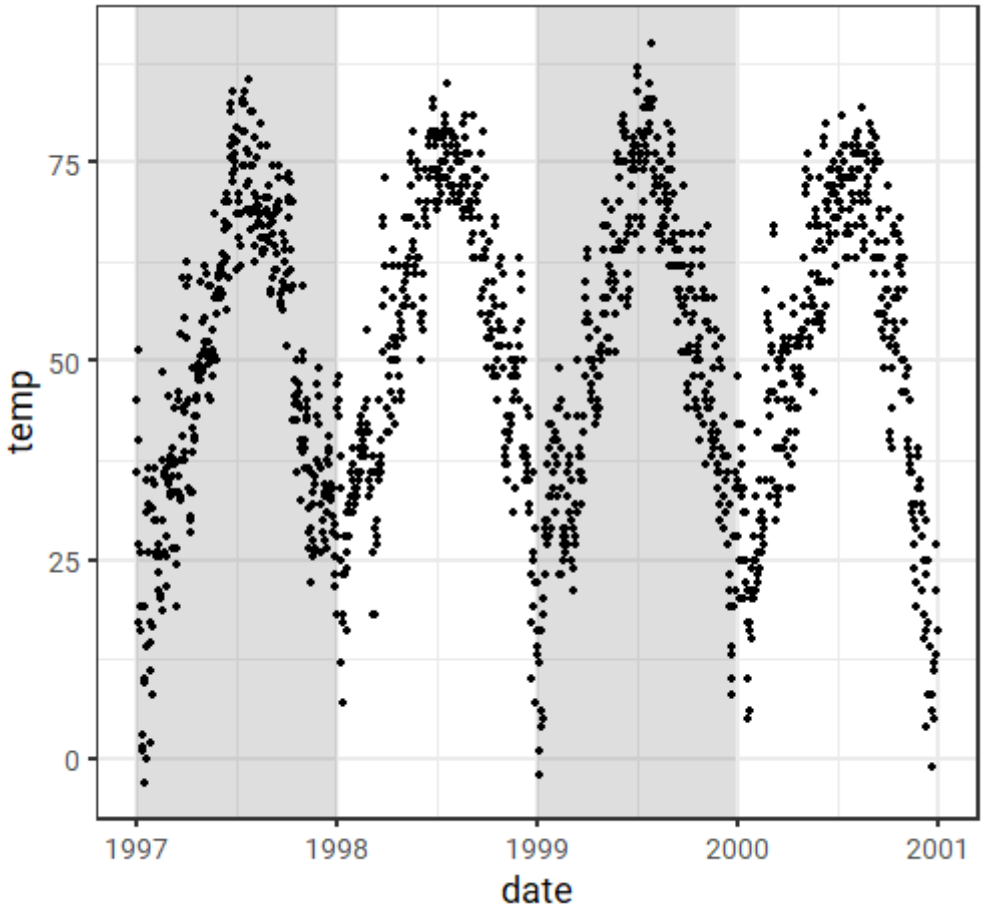


Working with Geometric Forms

Working with Geometric Forms: `annotate("rect")`

Let's add some background indicating the years:

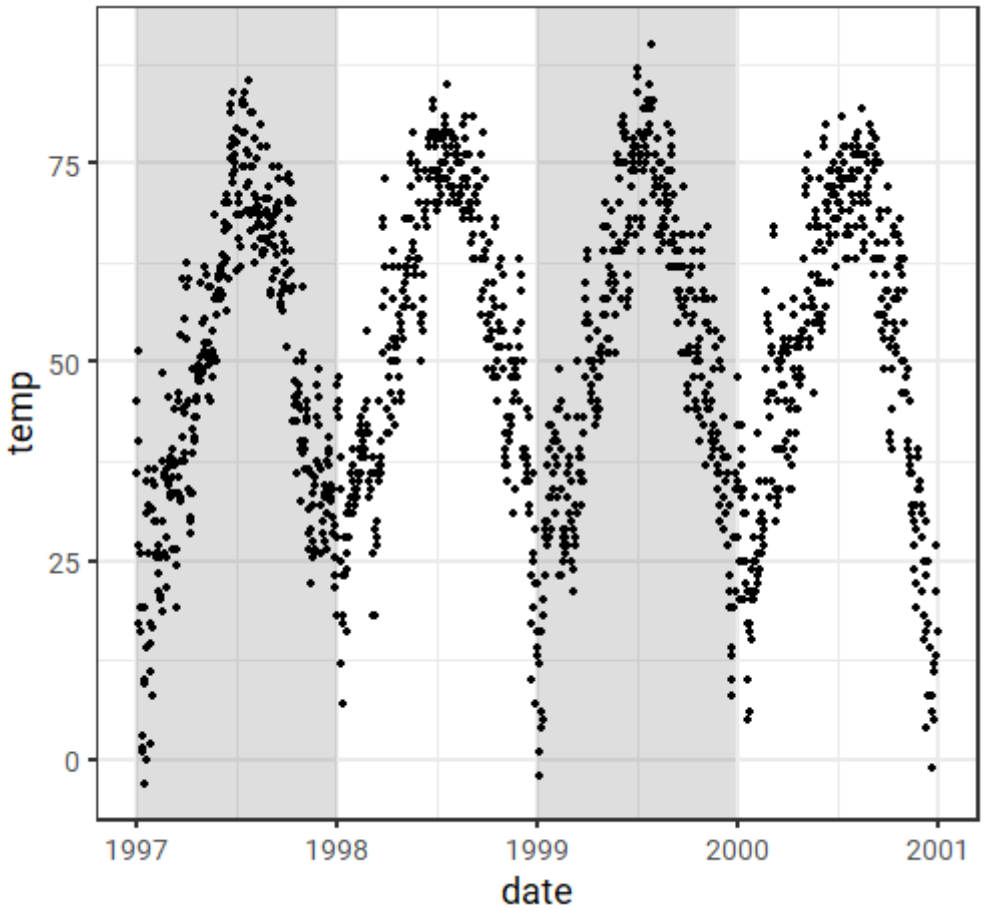
```
ggplot(chic, aes(date, temp)) +
  annotate(
    "rect",
    xmin = as.Date("1997-01-01"),
    xmax = as.Date("1997-12-31"),
    ymin = -Inf, ymax = Inf,
    color = NA, alpha = 0.2
  ) +
    annotate(
    "rect",
    xmin = as.Date("1999-01-01"),
    xmax = as.Date("1999-12-31"),
    ymin = -Inf, ymax = Inf,
    color = NA, alpha = 0.2
  ) +
  geom_point()
```



Working with Geometric Forms: `annotate("rect")` or `geom_rect()`?

If you plot many elements, use `geom_rect()`:

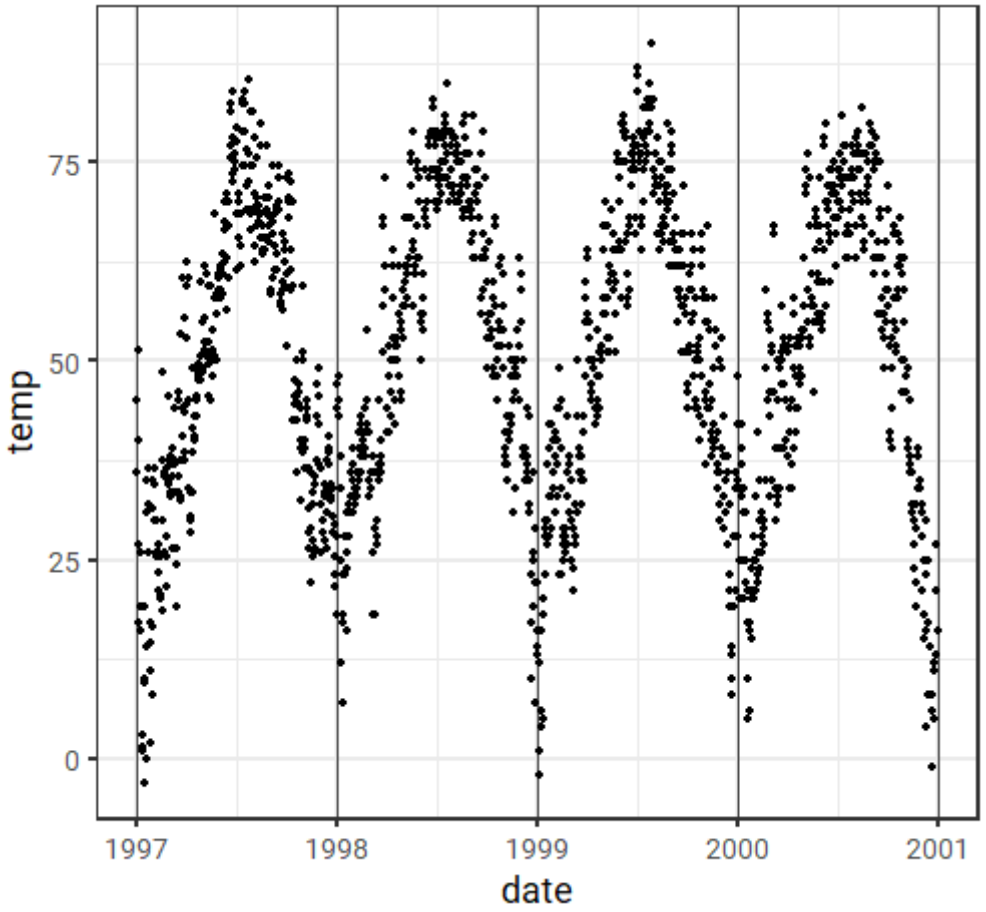
```
rect <- tibble(
  xmin = as.Date(c("1997-01-01",
                   "1999-01-01")),
  xmax = as.Date(c("1997-12-31",
                   "1999-12-31")),
  ymin = rep(-Inf, 2),
  ymax = rep(Inf, 2)
)

ggplot(chic) +
  geom_rect(
    data = rect,
    aes(xmin = xmin, xmax = xmax,
        ymin = ymin, ymax = ymax),
    color = NA,
    alpha = 0.2
  ) +
  geom_point(aes(date, temp))
```



Working with Geometric Forms: `geom_vline()`

We could also indicate new years by a vertical line:

```
lines <- tibble(
  xintercept = seq(
    as.Date("1997-01-01"),
    as.Date("2001-01-01"),
    length.out = 5)
)

ggplot(chic, aes(date, temp)) +
  geom_vline(
    data = lines,
    aes(xintercept = xintercept),
    color = "grey30") +
  geom_point()
```



There are also `geom_abline()` and `geom_hline()`.