Starting DataViz using R: ggplot2

Abhijit Dasgupta, PhD

What is ggplot2?

- A second (and final) iteration of the ggplot
- Implementation of Wilkerson's Grammar of Graphics in R
- Conceptually, a way to layer different elements onto a canvas to create a data visualization
- Started as Dr. Hadley Wickham's PhD thesis (with Dr. Dianne Cook)

- Won the John M. Chambers Statistical Software Award in 2006

- Mimicked in other software platforms
    - `ggplot` and `seaborn` in Python
    - Translated in `plotly`

ggplot2 uses the grammar of graphics

A grammar ...

- compose and re-use small parts
- build complex structures from simpler units

of graphics ...

- Think of yourself as a painter
- Build a visualization using layers on a canvas
- Draw layers on top of each other

A dataset

```
library(tidyverse) # do this once per session
beaches <- read_csv('../data/sydneybeaches3.csv')
```

```
#>  # A tibble: 344 x 12
#>     date        year month   day season rainfall temperature
#>     <date>     <dbl> <dbl> <dbl>  <dbl>    <dbl>       <dbl>
#>   1 2013-01-02  2013     1     2      1      0          23.4
#>   2 2013-01-06  2013     1     6      1      0          30.3
#>   3 2013-01-12  2013     1    12      1      0          31.4
#>   4 2013-01-18  2013     1    18      1      0          46.4
#>   5 2013-01-24  2013     1    24      1      0          27.5
#>   6 2013-01-30  2013     1    30      1      0.6        26.6
#>   7 2013-02-05  2013     2     5      1      0.1        25.7
#>   8 2013-02-11  2013     2    11      1      8          22.2
#>   9 2013-02-17  2013     2    17      1     13.6        26.3
#>  10 2013-02-23  2013     2    23      1      7.2        24.8
#>  # … with 334 more rows, and 5 more variables:
#>  #   enterococci <dbl>, day_num <dbl>, month_num <dbl>,
#>  #   month_name <chr>, season_name <chr>
```

Credit: D. J. Navarro

> Download the `sydneybeaches3.csv` file from the website and save it in your project's data folder

Building a graph

Start with a blank canvas

```
ggplot()
```

Add a data set

```
ggplot(
  data = beaches # Tell ggplot the data you're using
)
```

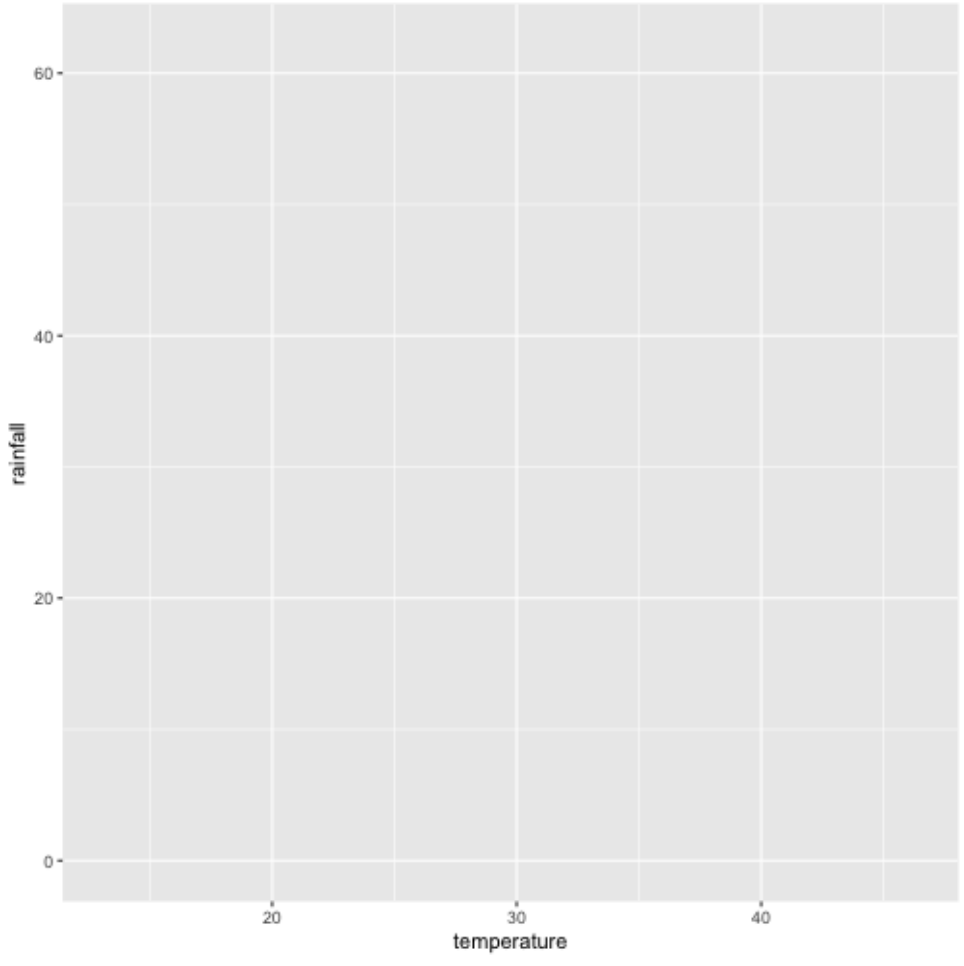
> Nothing has really happened yet, since we haven't said what we want to plot from the data set
>
> The # symbol tells R that anything after it is a *comment* and should be ignored. We'll make a lot of use of comments in our code.

Add a mapping from data to elements

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
)
```

What goes in

- the x and y axes
- the color of markers
- the shape of markers



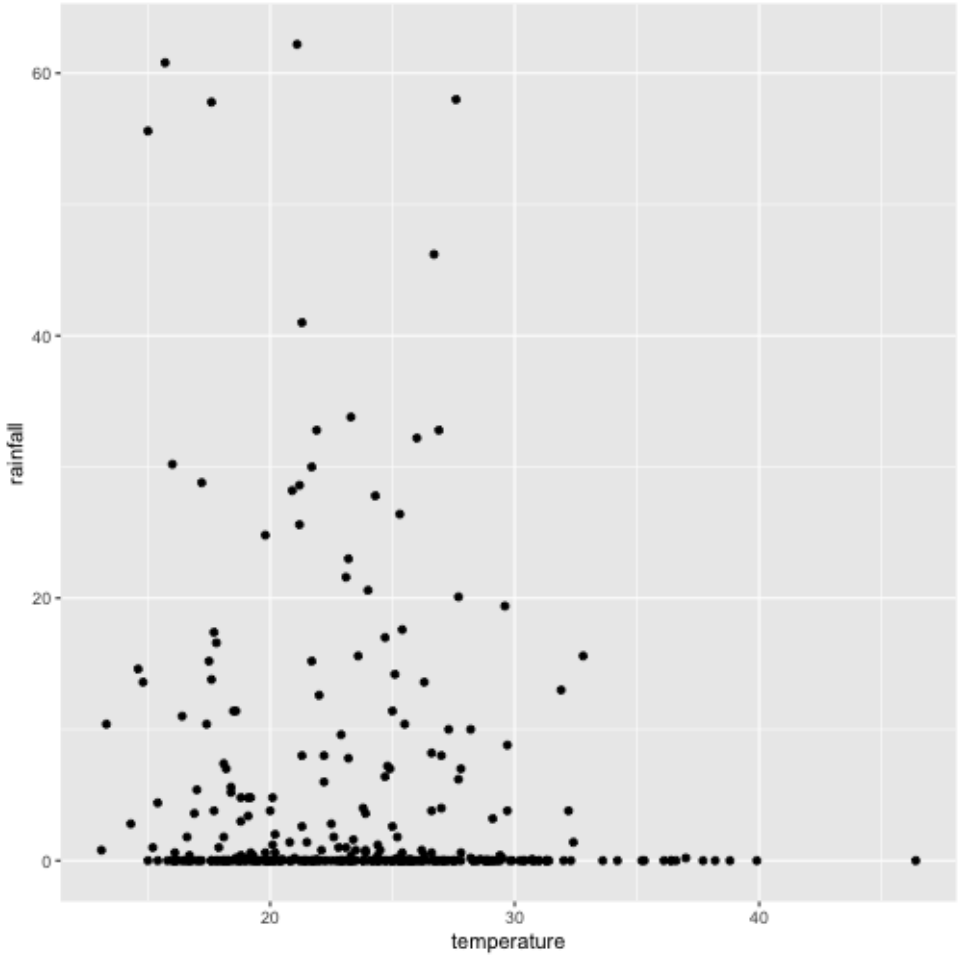
> Now we've said what we want to plot, so axes get drawn. We haven't yet specified how we want to plot it

Add a geometry to draw

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point()
```

What to draw:

- Points, lines
- histogram, bars, pies

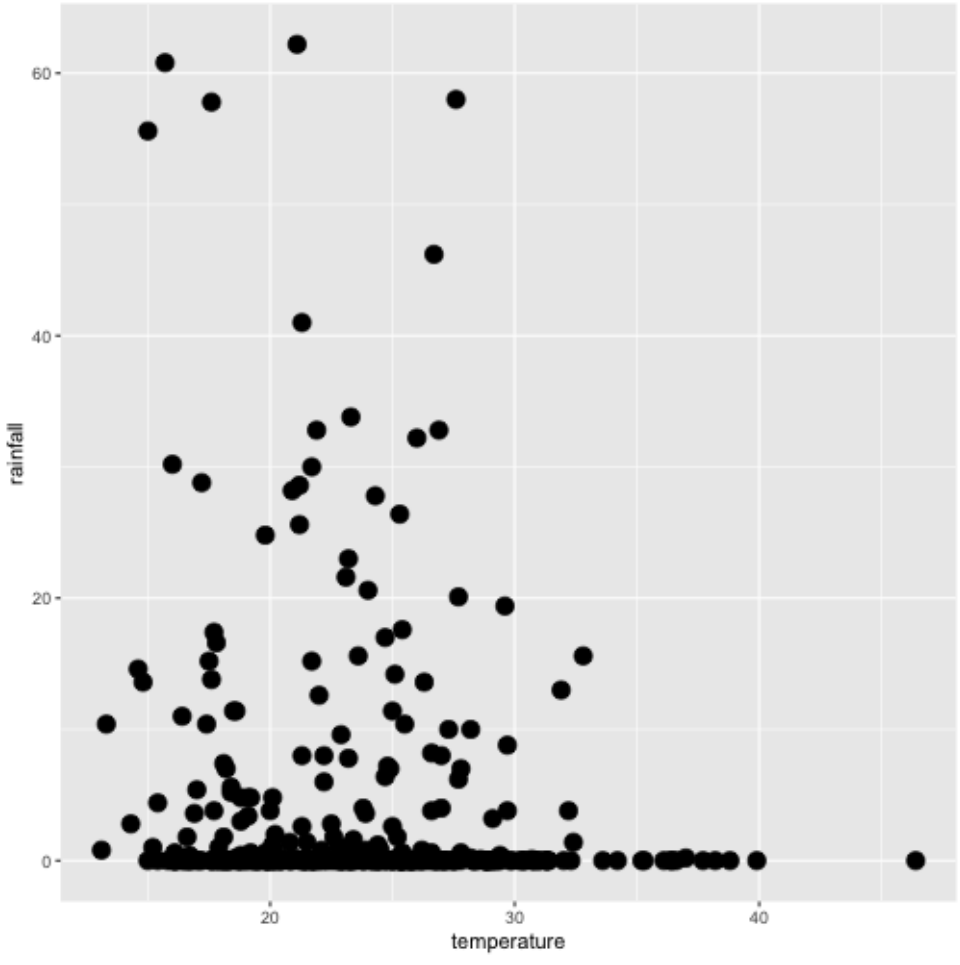


> Now we've said **how** we want to plot the things we want from the data

Add options for the geom

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(size = 4)
```

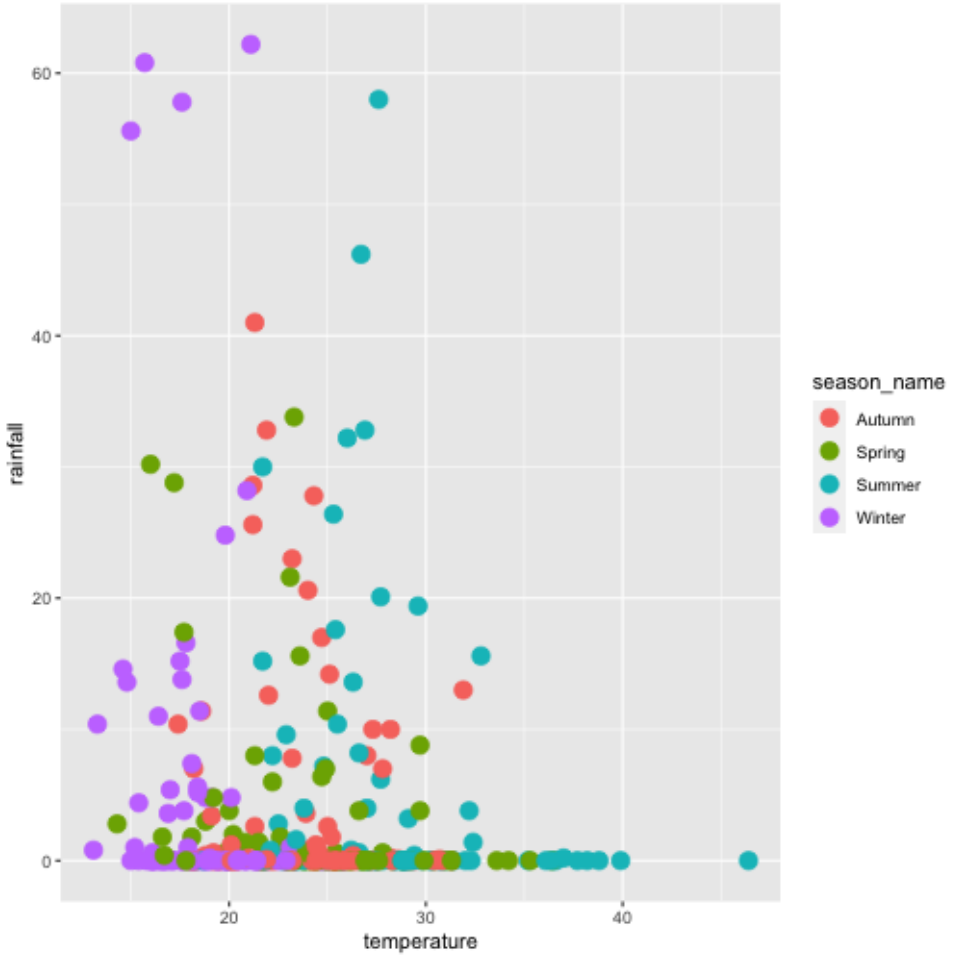
> Options pertaining to how we want things drawn are usually put in the function for the geometry, e.g. `geom_point`, `geom_line`, etc. If the option is based on any element from the dataset, we have to wrap it in a `aes()` function. We'll see this later



Add a mapping to modify the geom

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4
  )
```

> Anything data-driven has to be a mapping, driven by the `aes` function



A side note on aes

The aes function

The aes function and when it needs to be used creates quite a bit of confusion initially. Let's start with the documentation for this function

```
?aes
```

> **Description**
>
> Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms. Aesthetic mappings can be set in ggplot2() and in individual layers.

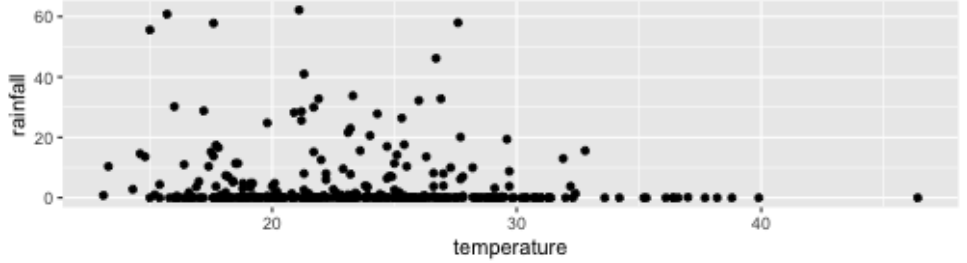
So, anytime we want to represent some aspect of the data on the plot in some form (color, shape, etc.), we have to use the aes function to *map* the data to the plot

The aes function

The `aes` function can occur in one of two places:

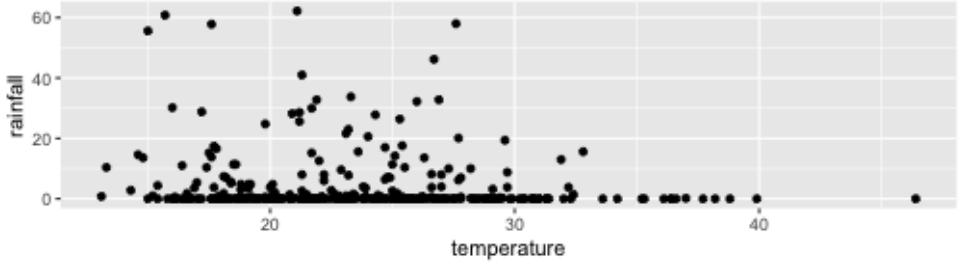
Within the `ggplot` function:

```
ggplot(
  data=beaches,
  mapping=aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point()
```



In the actual `geom` layer:

```
ggplot(
  data = beaches
) +
  geom_point(
    mapping = aes(
      x = temperature,
      y = rainfall
    )
  )
```



The `aes` function

The `aes` function can occur in one of two places:

Within the `ggplot` function:

- You do this if the same mapping will be common to **all** the subsequent geometry layers

In the actual `geom` layer:

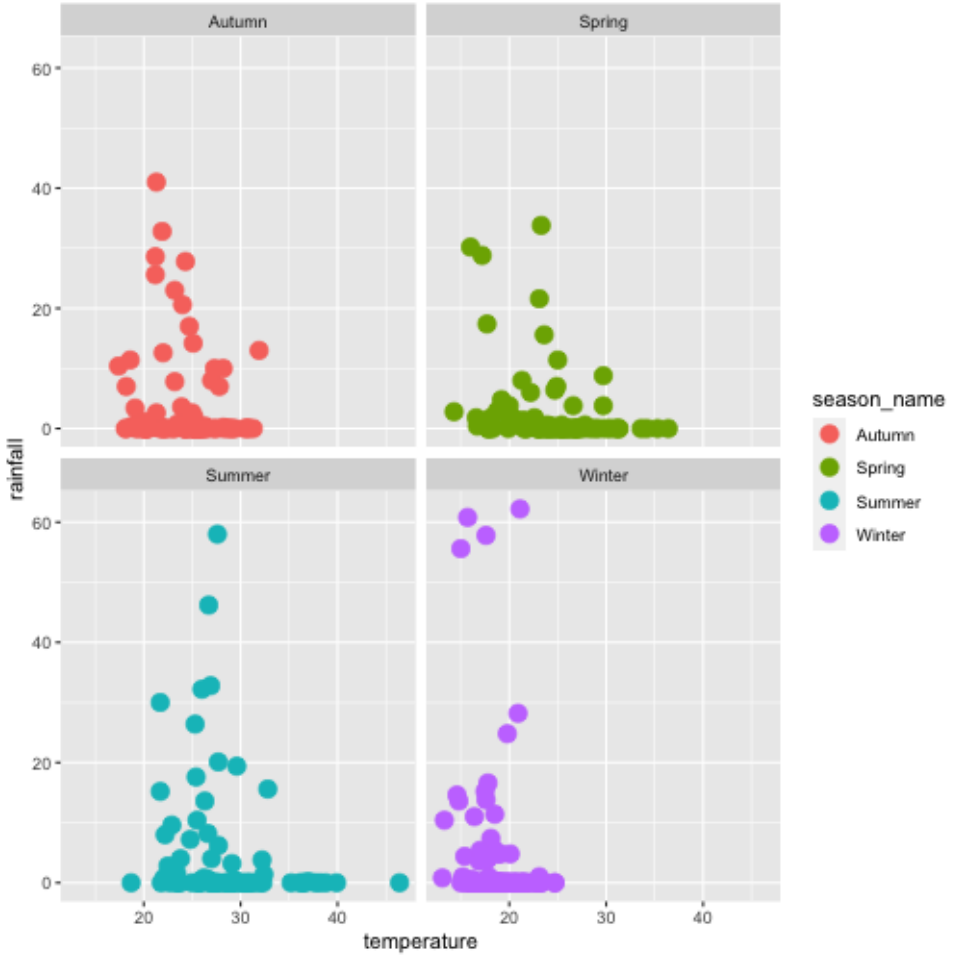
- You do this if the mapping will apply **only** to that layer
- You do it if it makes more sense to put it in the `geom`.

Facets / Trellis / Small multiples

Split into facets

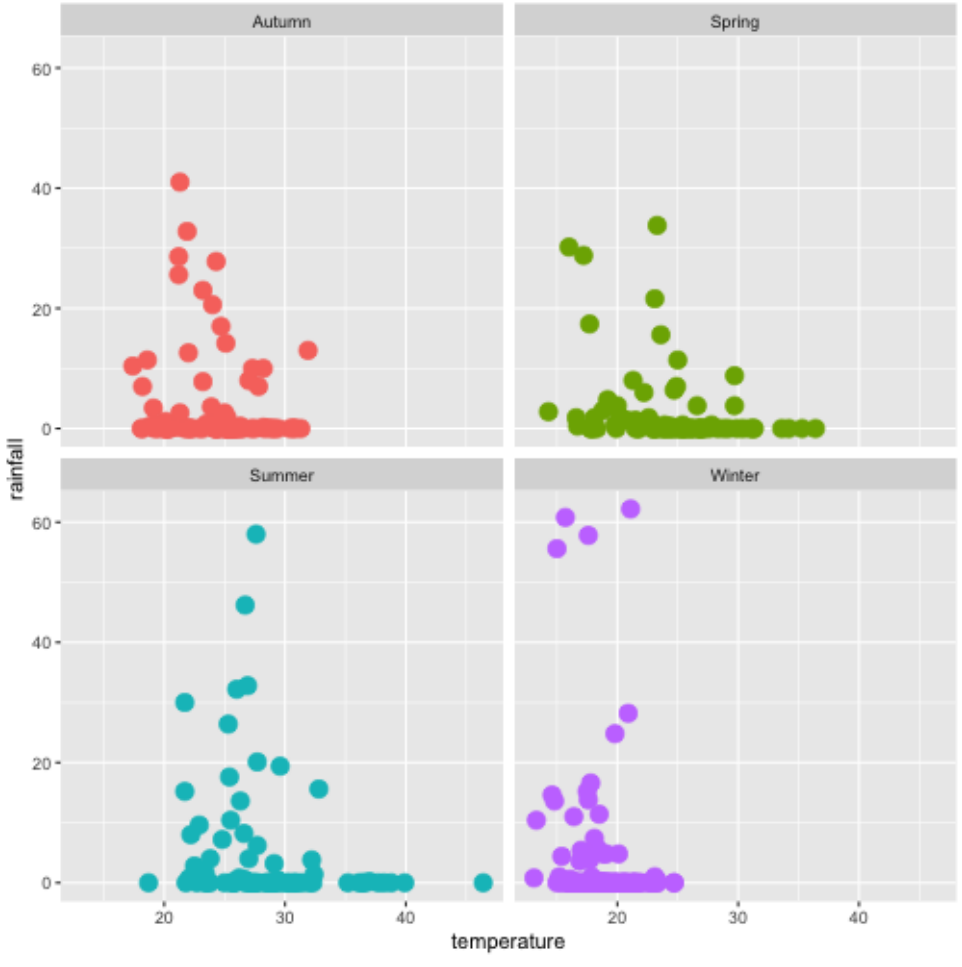
```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4
  ) +
  facet_wrap( ~ season_name)
```

> Create separate plots based on unique values of some variable (`season_name`) in your dataset
>
> Typically this variable should a few distinct values



Remove the legend

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4,
    show.legend = FALSE
  ) +
  facet_wrap( ~ season_name)
```

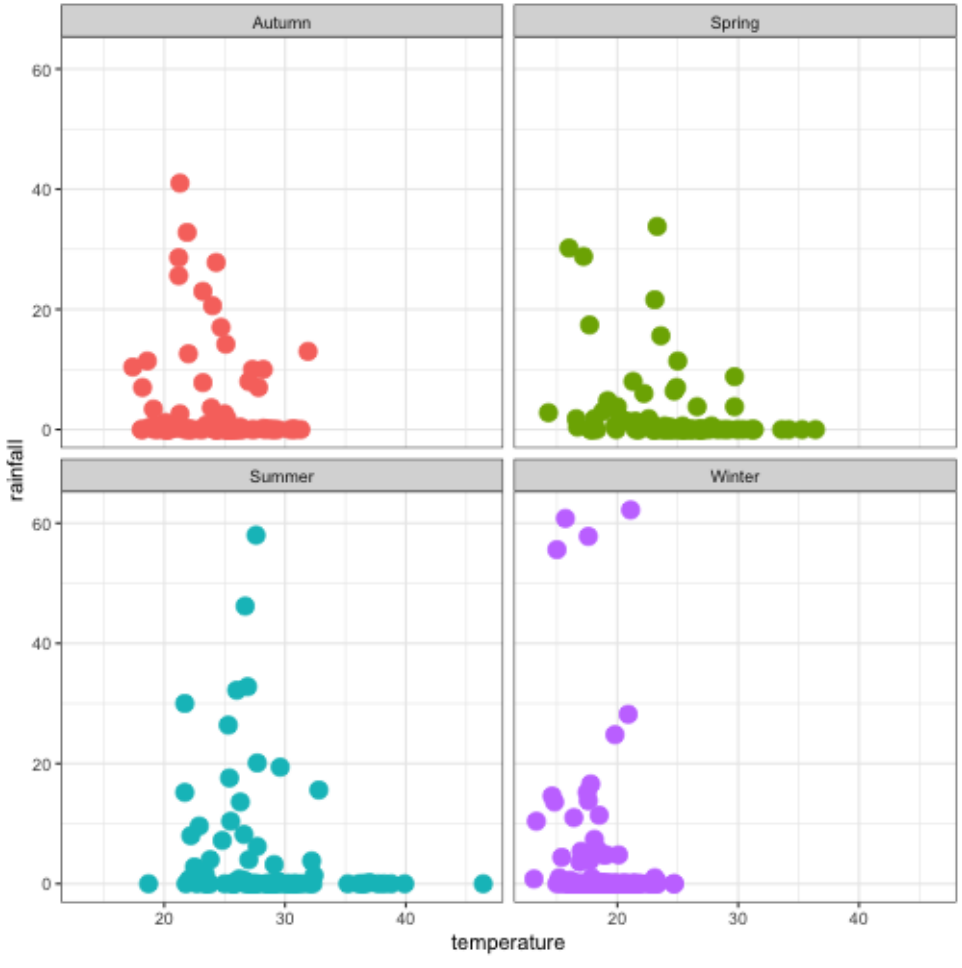


> Now we're getting more into the look of the plot and how much information should be on it
>
> `show.legend` option is in `geom_point` here since it would be based on the colors of the points. If you had a different geometry like `geom_line`, you would put the `show.legend` option there if the legend was based on that geom.

Change the background

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4,
    show.legend = FALSE
  ) +
  facet_wrap( ~ season_name) +
  theme_bw()
```

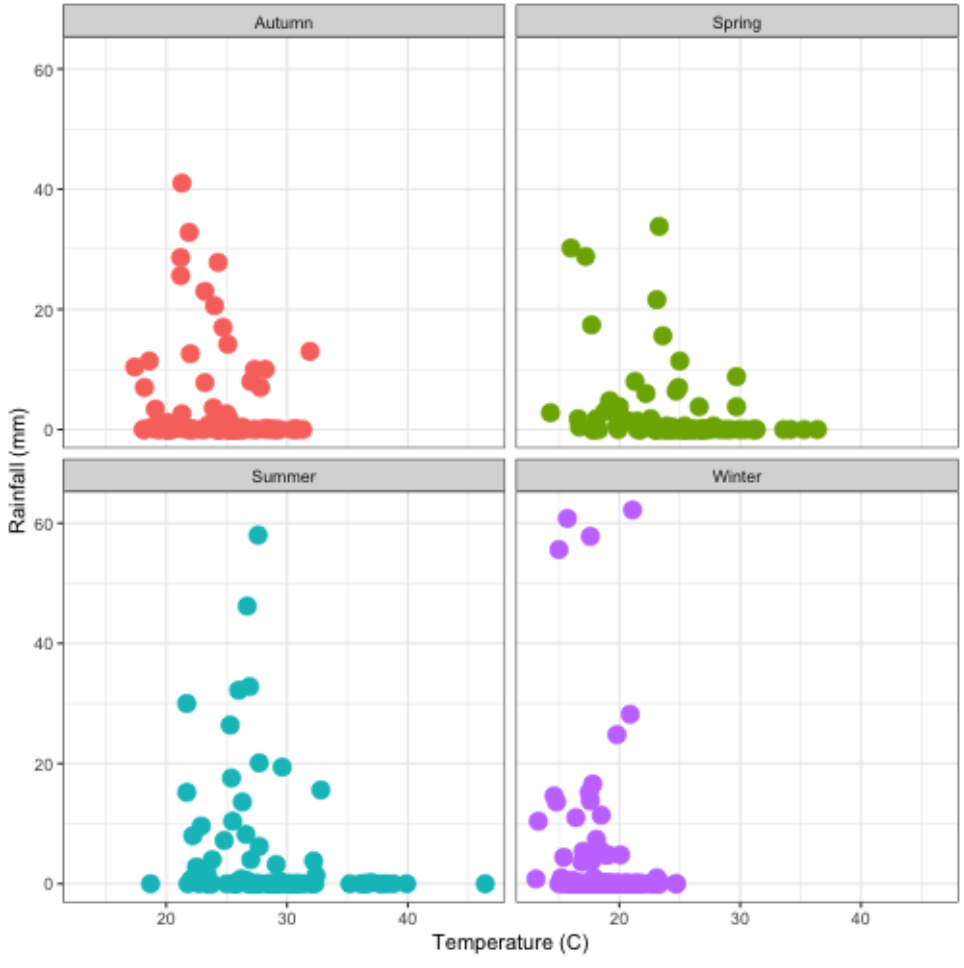
> Again, a look-and-feel choice
>
> In-built `ggplot` themes are described here. Other themes are availabe via packages `ggthemes` and others.



Update the labels

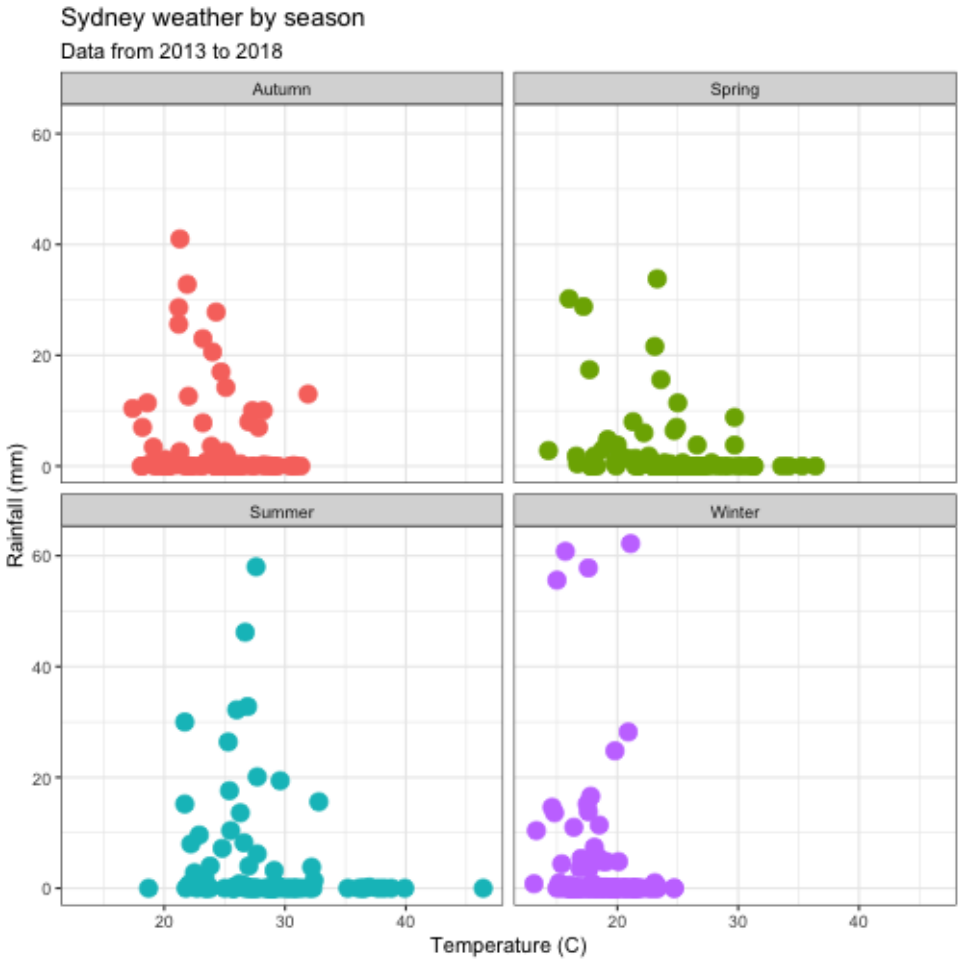
```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4,
    show.legend = FALSE
  ) +
  facet_wrap( ~ season_name) +
  theme_bw() +
  labs(x = 'Temperature (C)', y = 'Rainfall (mm)')
```

> This is **important**. Make sure the information in your plot is self-contained by putting appropriate labels and titles on it.



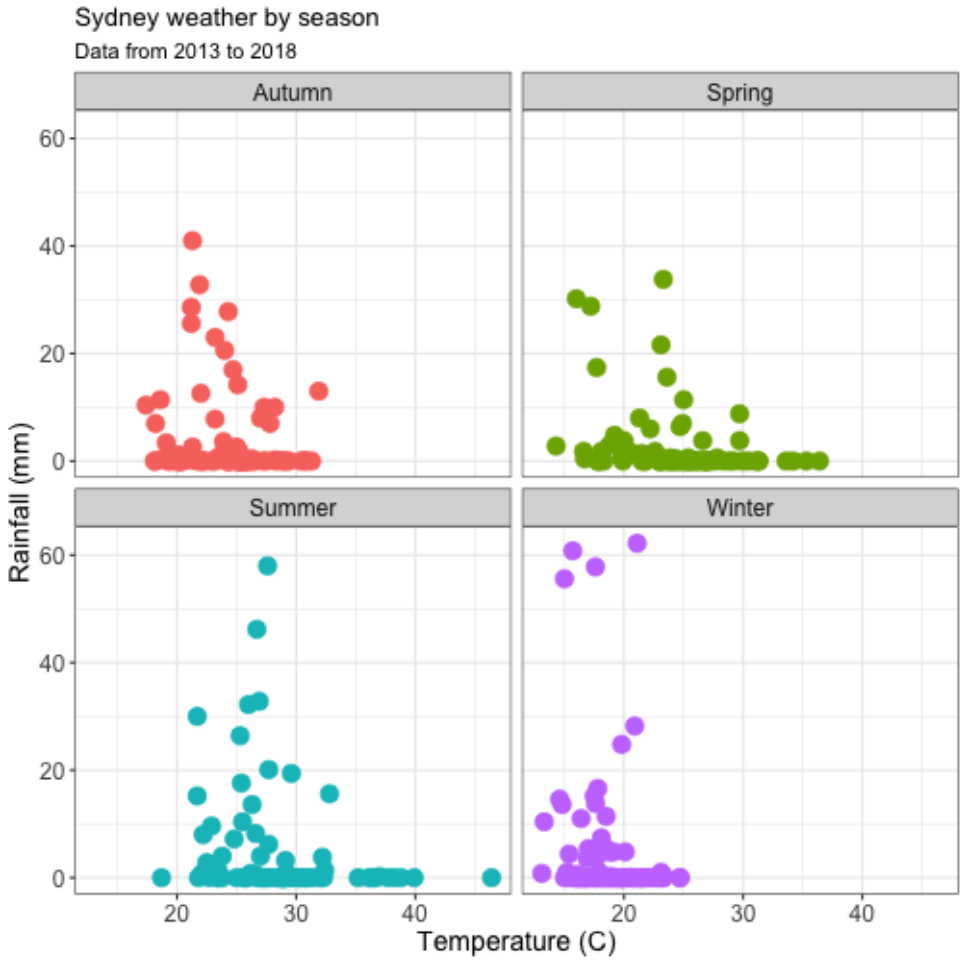
Add titles

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4,
    show.legend = FALSE
  ) +
  facet_wrap( ~ season_name) +
  theme_bw() +
  labs(x = 'Temperature (C)',
       y = 'Rainfall (mm)',
       title = 'Sydney weather by season',
       subtitle = "Data from 2013 to 2018")
```



Customize

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4,
    show.legend = FALSE
  ) +
  facet_wrap( ~ season_name) +
  theme_bw() +
  labs(x = 'Temperature (C)',
       y = 'Rainfall (mm)',
       title = 'Sydney weather by season',
       subtitle = "Data from 2013 to 2018") +
  theme(axis.title = element_text(size = 14),
        axis.text = element_text(size = 12),
        strip.text = element_text(size = 12))
```



> Once again, a look-and-feel choice, but this is very useful for publications when the actual figure will be smaller but we need the labels to be legible.

Understanding the structure of ggplot

The grammar

- Data
- Aesthetics (or aesthetic mappings)
- Geometries (as layers)
- Facets
- Themes
- (Coordinates)
- (Scales)

> Data, Aesthetics and Geometries are **required** to actually create a plot

Peeking under the hood

If I write...

```
ggplot(
  data = beaches,
  aes(x = temperature,
      y = rainfall)
) +
  geom_point()
```

what's really run is ...

```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature, y = rainfall)) +
layer(
  geom = "point",
  stat = "identity",
  position = "identity") +
facet_null() +
theme_grey() +
coord_cartesian() +
scale_x_continuous() +
scale_y_continuous()
```

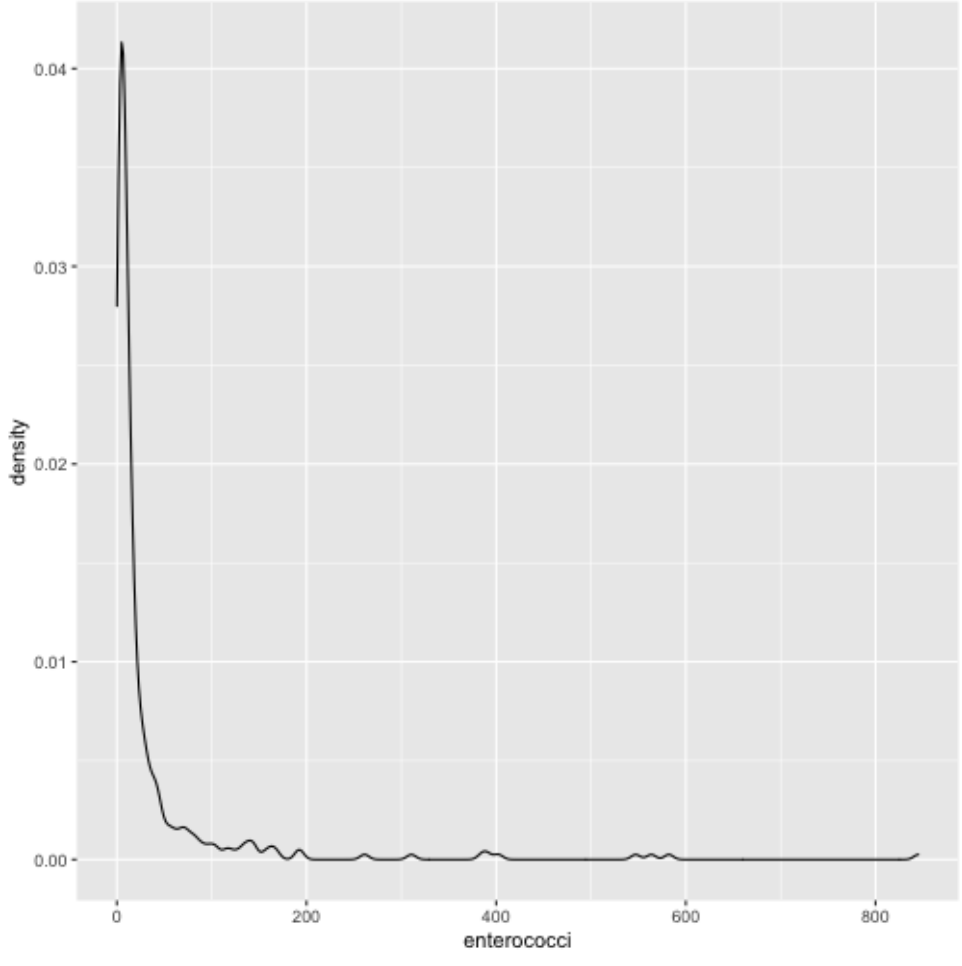
Each element can be adapted and tweaked to create graphs

Exploring aesthetics, mappings and their uses

Let's start

```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci
  )
) +
  geom_density()
```

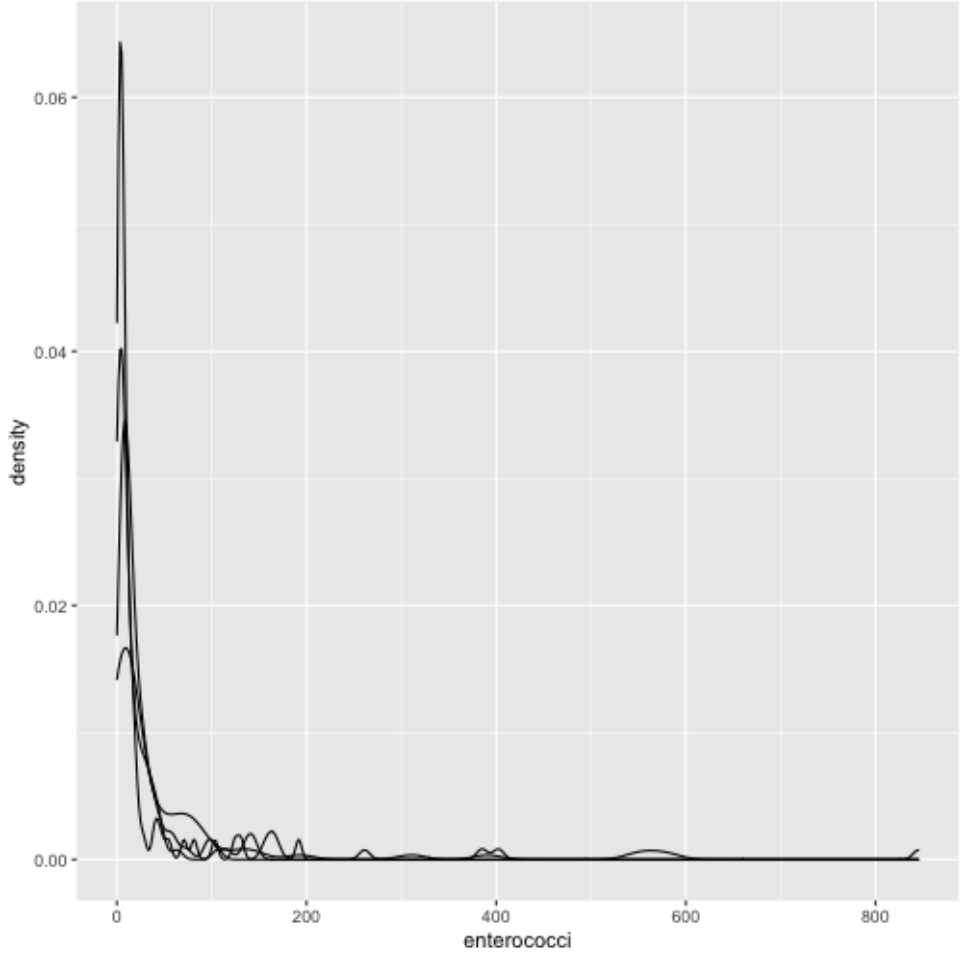
> This is looking at the distribution of enterococci concentration (x-axis) in this dataset



Bacteria growth should depend on temperature....

```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci,
    group = season_name
  )
) +
  geom_density()
```

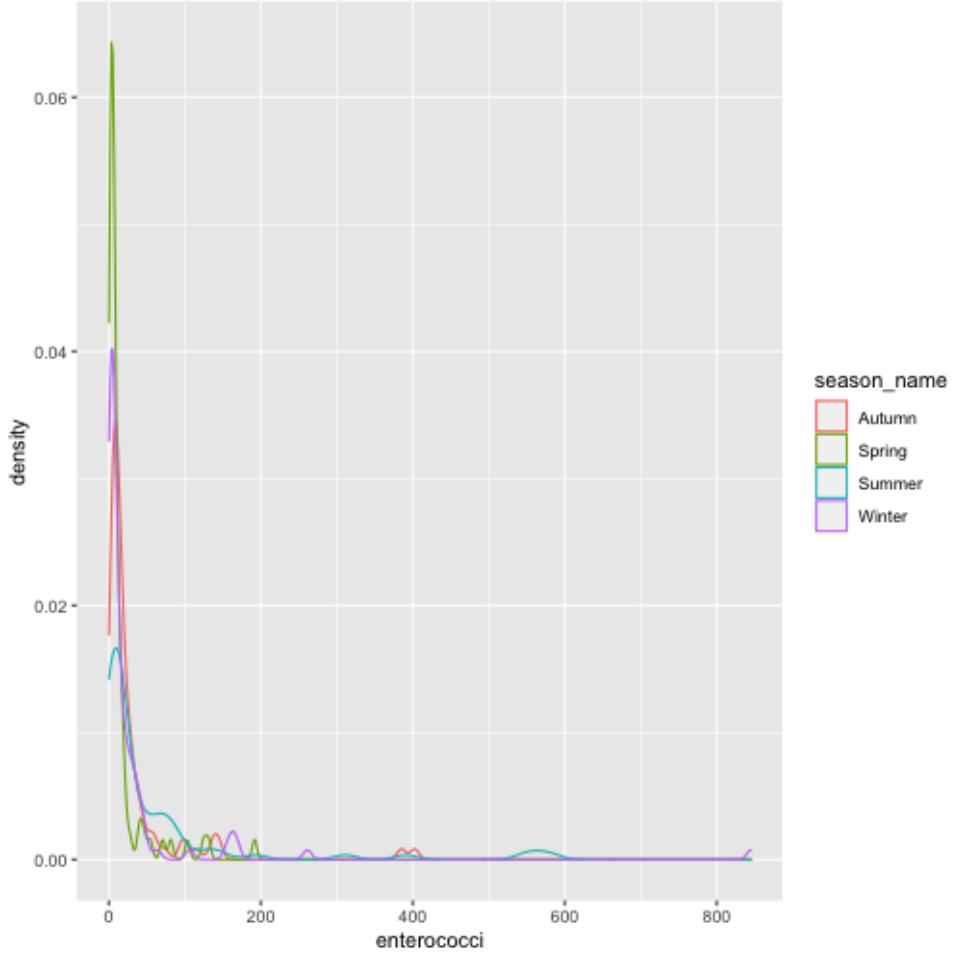
> Can't really parse the seasons out from the graph.



Add some color

```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci,
    group = season_name,
    color = season_name
  )
) +
  geom_density()
```

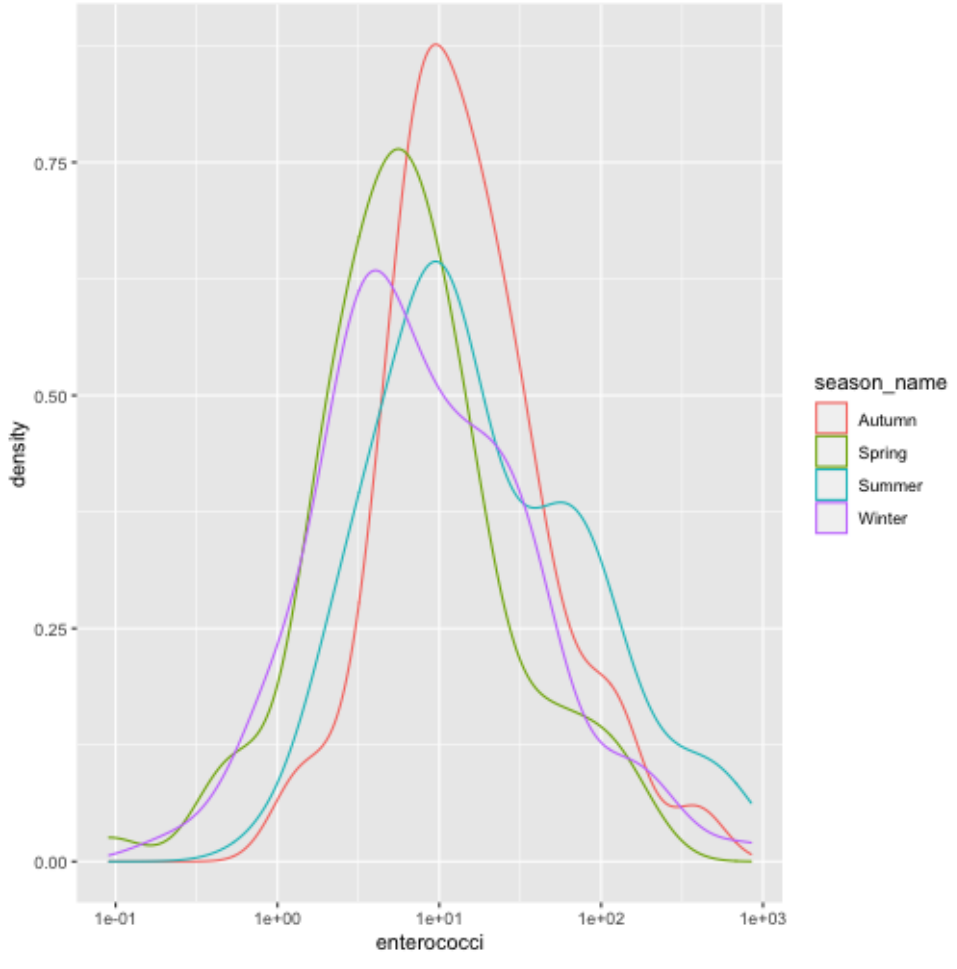
> This does better distinguish the seasons (and we get a legend as a by-product).
>
> But.... things are too crammed on the left of the plot. This is because the bacterial concentrations are low, with some high values. A transformation may be nice.



A better choice of scale

```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci,
    group = season_name,
    color = season_name
  )
) +
  geom_density() +
  scale_x_log10()
```

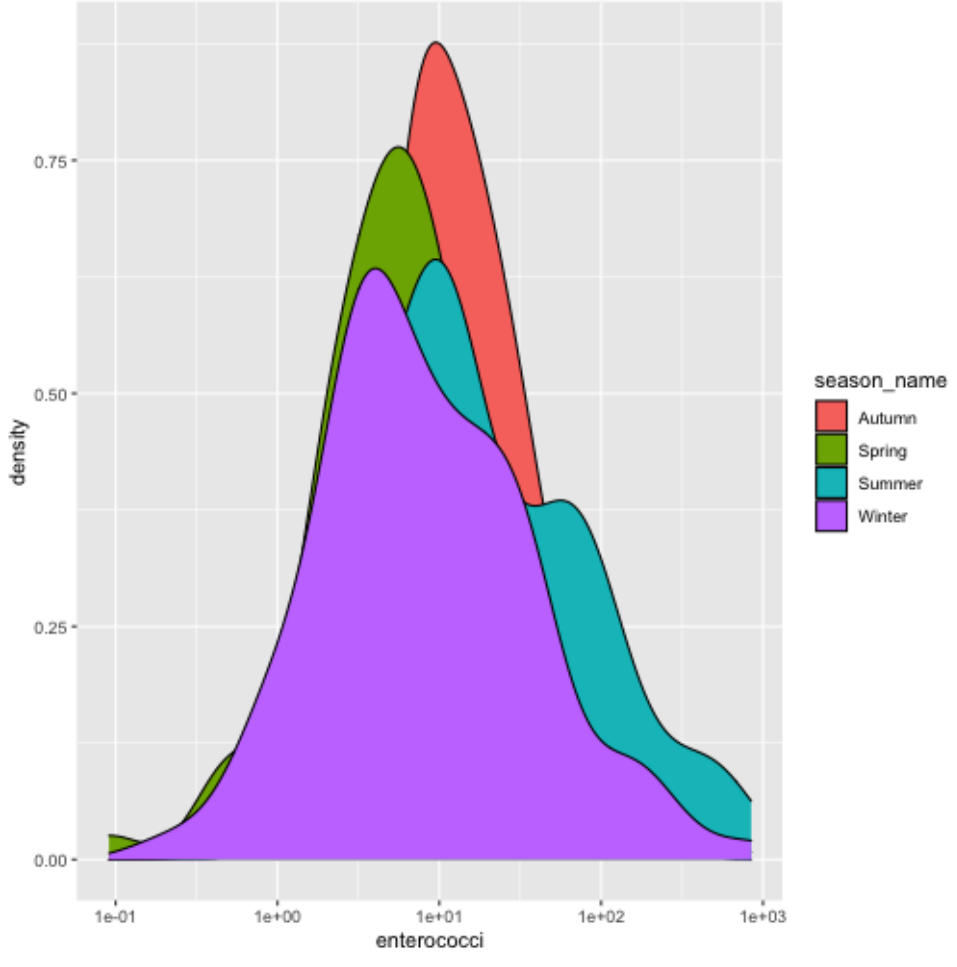
> This makes things a bit clearer, but still a bunch of squiggly lines



A better choice of where to put the color

```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci,
    group = season_name,
    fill = season_name
  )
) +
  geom_density() +
  scale_x_log10()
```

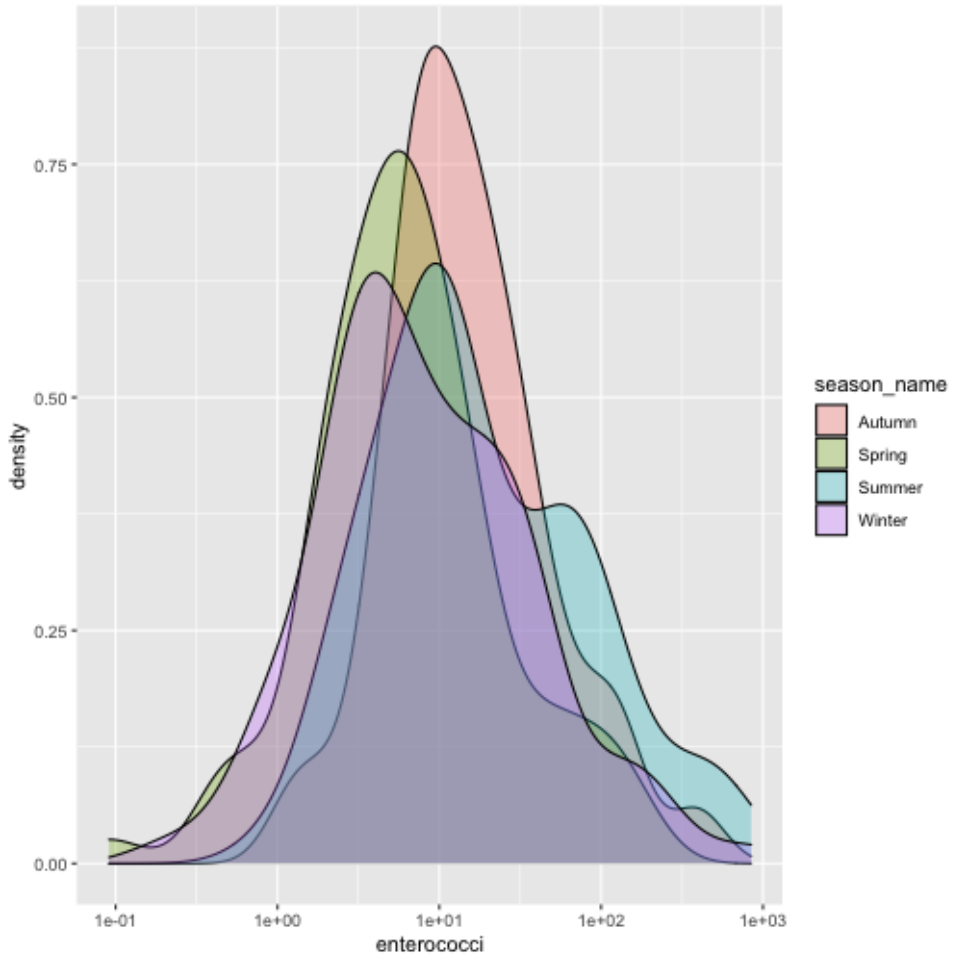
> Things are covered up!
>
> For geometries with "insides", `color` puts color on the outlines, and `fill` puts color on the insides.



Let's play peek-a-boo

```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci,
    group = season_name,
    fill = season_name
  )
) +
  geom_density(alpha = 0.3) +
  scale_x_log10()
```

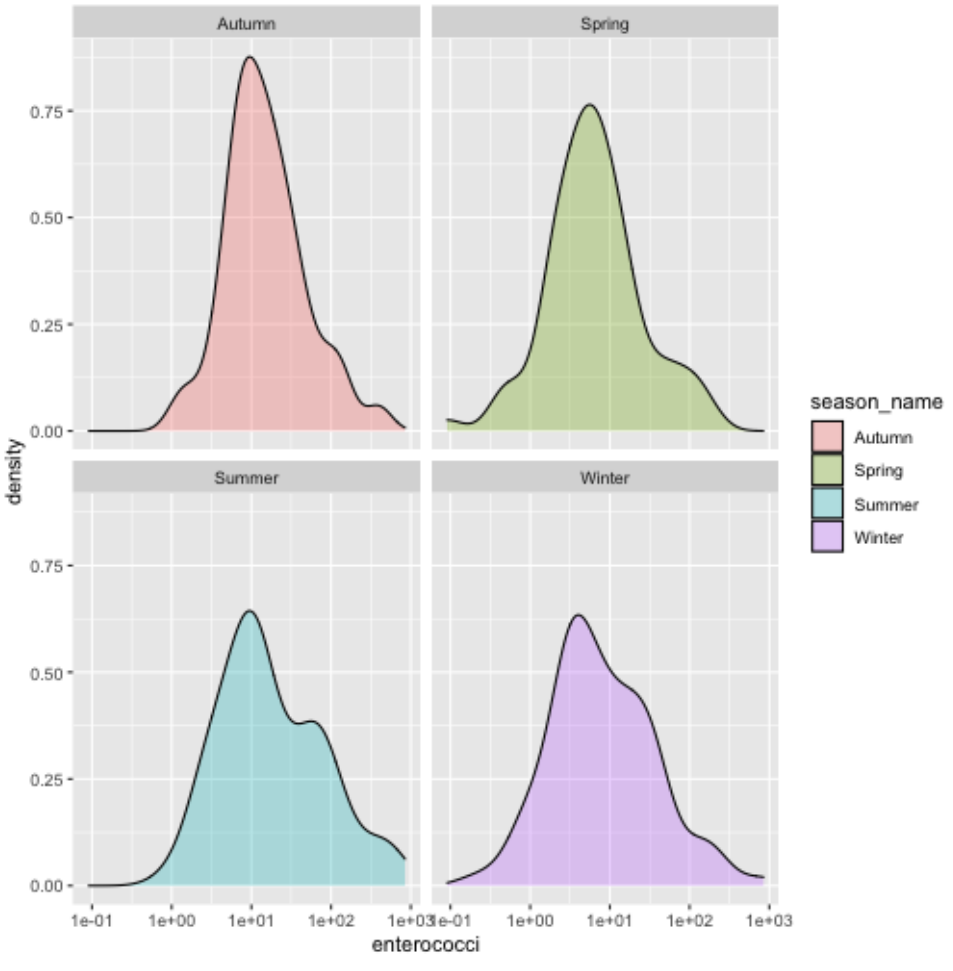
> A bit better, but not great



Let's break things out

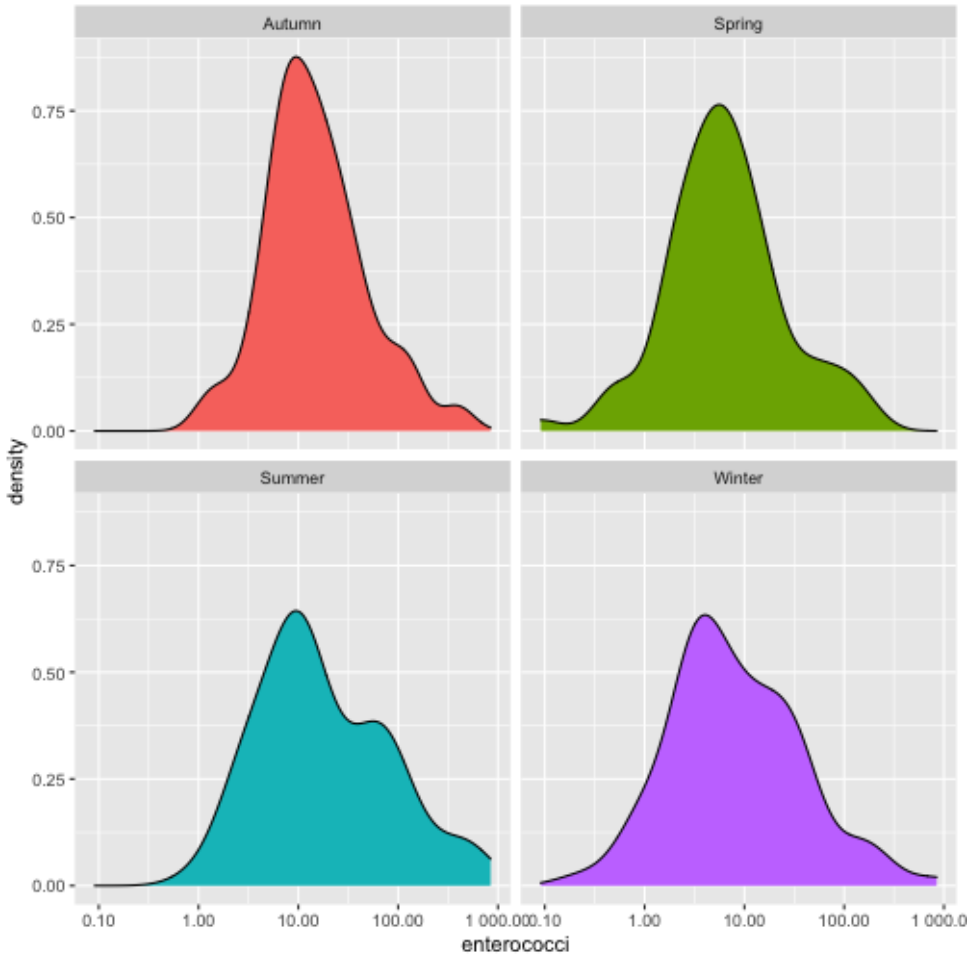
```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci,
    group = season_name,
    fill = season_name
  )
) +
  geom_density(alpha = 0.3) +
  scale_x_log10() +
  facet_wrap(~ season_name)
```

> A lot clearer!!
>
> There's stuff we don't need now, like the legend Also, can we please make the numbers human-readable



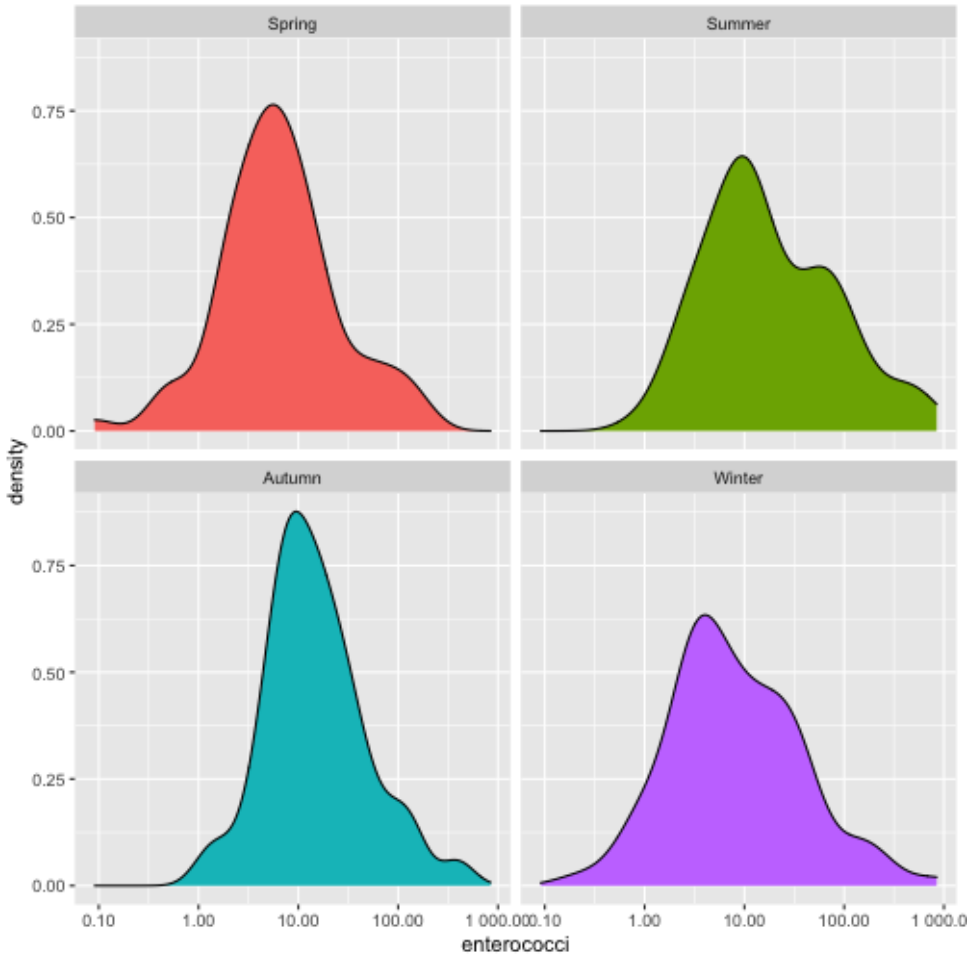
Cleaning up

```
ggplot(
  data = beaches,
  mapping = aes(
    x = enterococci,
    fill = season_name
  )
) +
  geom_density(show.legend = FALSE) +
  scale_x_log10(labels = scales::label_number()) +
  facet_wrap(~season_name)
```



Fixing the legend ordering

```
beaches <- beaches %>%
  mutate(season_name = fct_relevel(season_name,
                                   c('Spring','Summer
ggplot(
  data=beaches,
  aes(x = enterococci,
      fill = season_name
  )
) +
  geom_density(show.legend = F) +
  facet_wrap(~ season_name) +
  scale_x_log10(labels = scales::label_number())
```



Exploring geometries

Univariate plots

Histograms

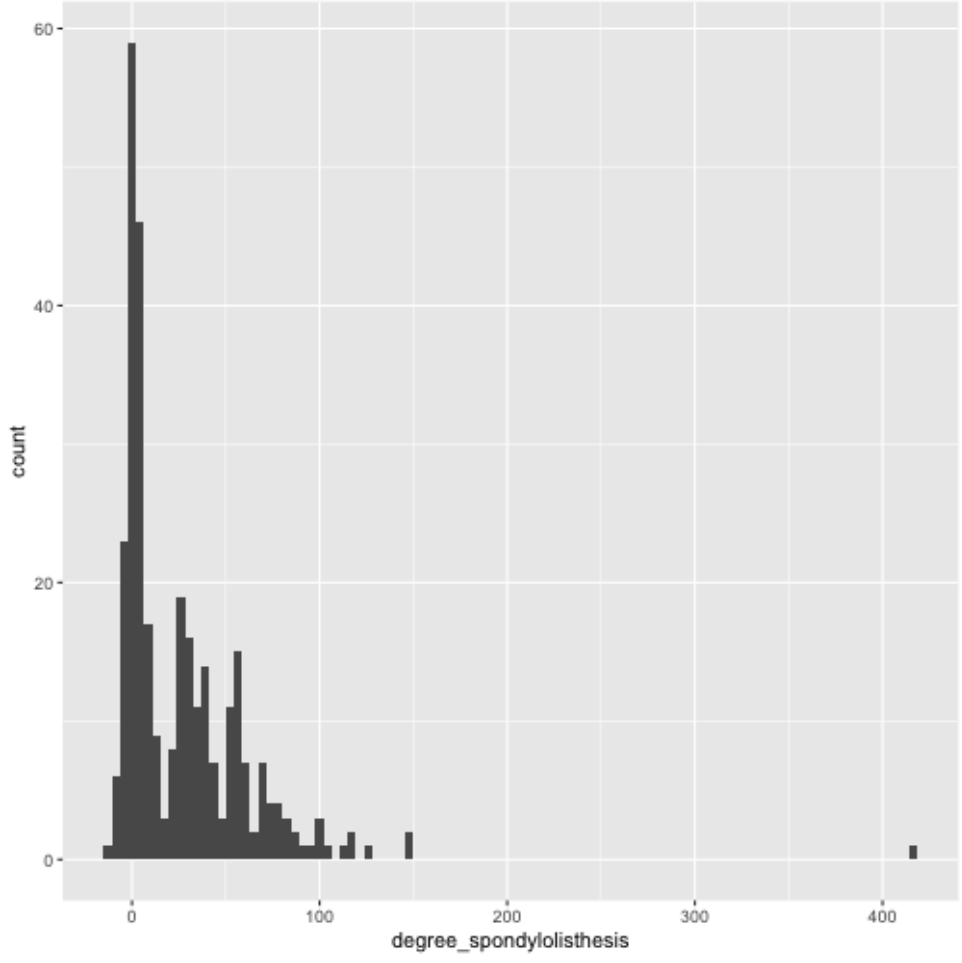
```
library(tidyverse)
library(rio)
dat_spine <- import('data/Dataset_spine.csv',
                    check.names = T)

ggplot(
  data=dat_spine,
  aes(x = degree_spondylolisthesis))+
  geom_histogram()
```

```
#>  `stat_bin()` using `bins = 30`. Pick better value
#>  `binwidth`.
```

Histograms

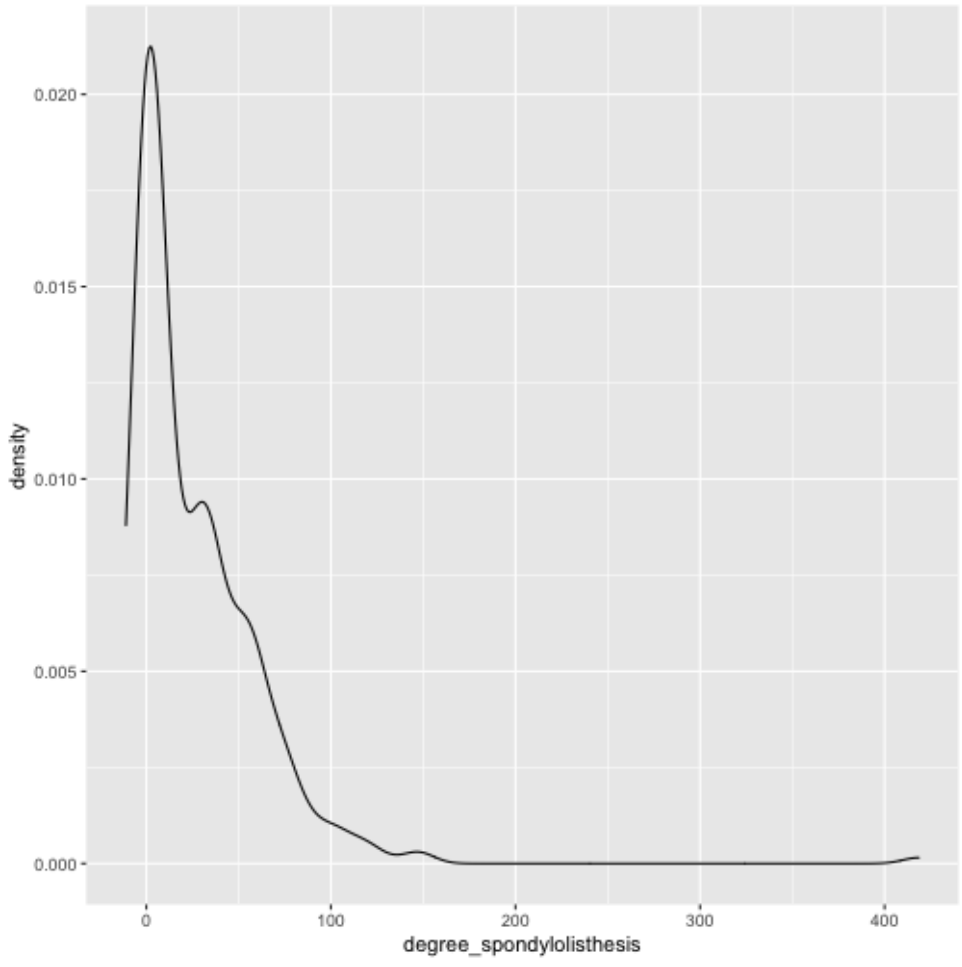
```
ggplot(
  data=dat_spine,
  aes(x = degree_spondylolisthesis))+
  geom_histogram(bins = 100)
```

This gives a very different view of the data



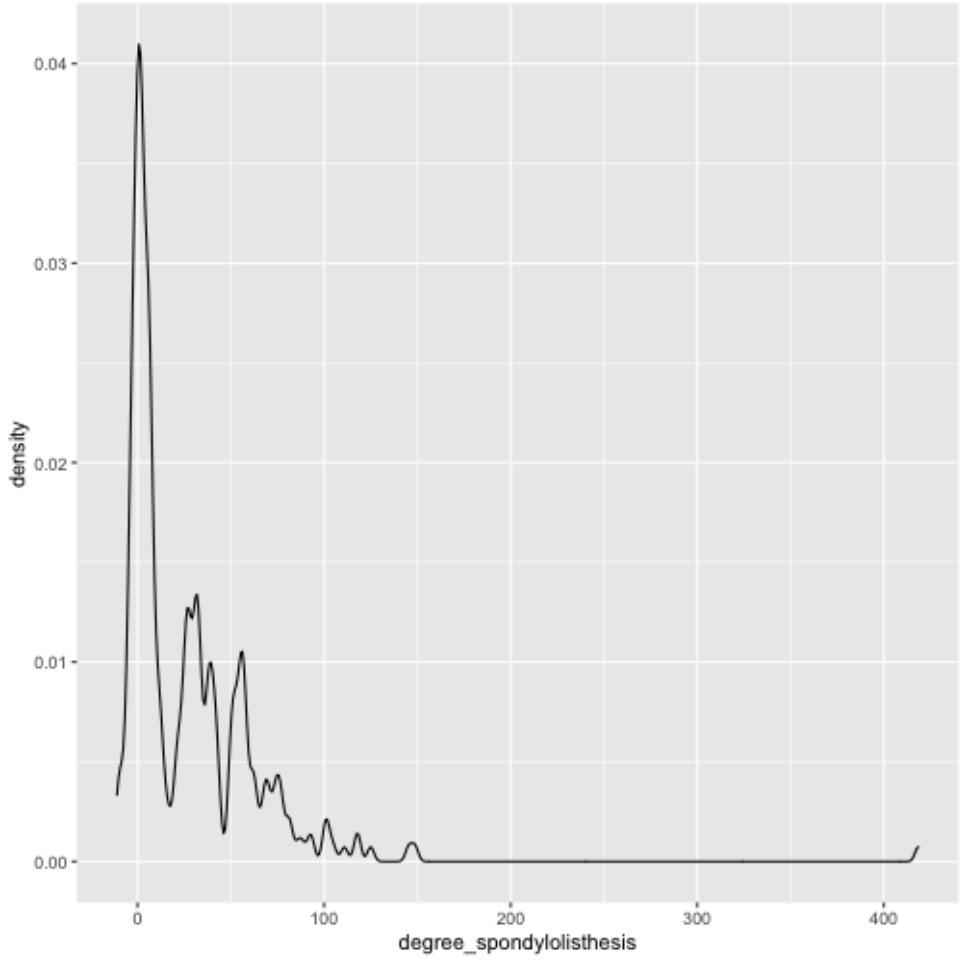
Density plots

```
ggplot(
  data=dat_spine,
  aes(x = degree_spondylolisthesis))+
  geom_density()
```



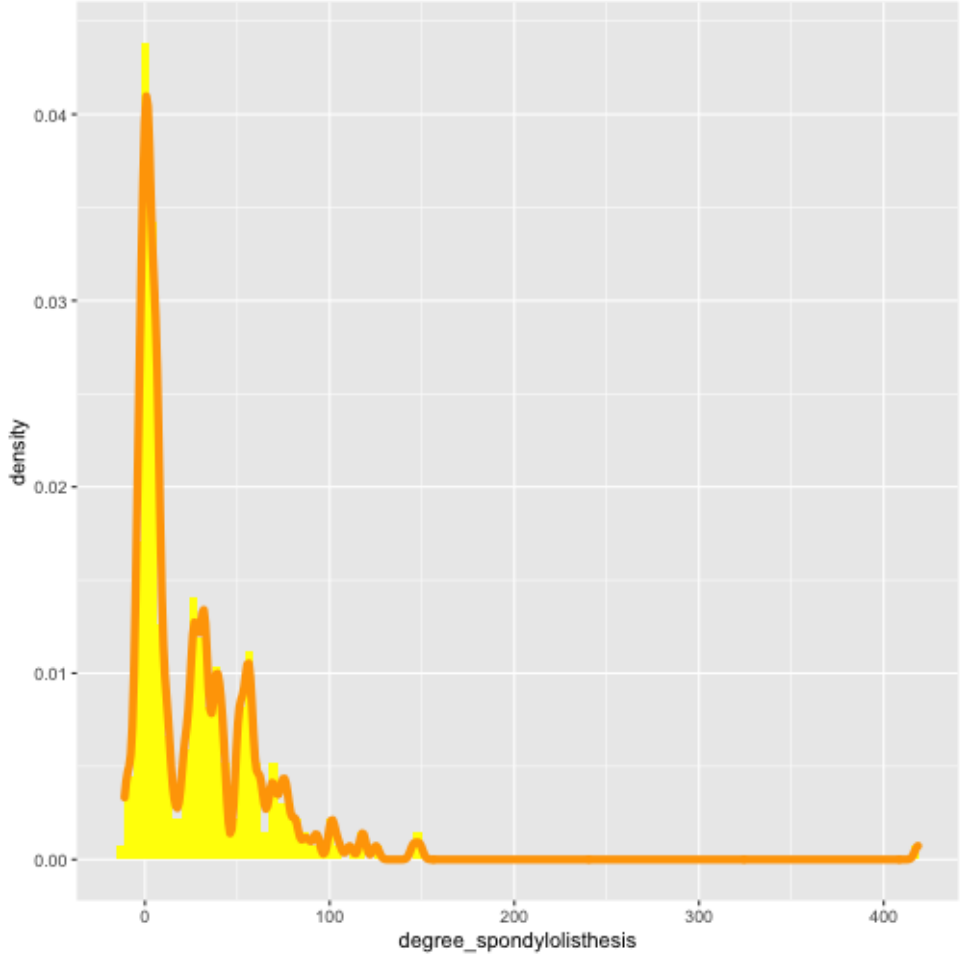
Density plots

```
ggplot(
  data=dat_spine,
  aes(x = degree_spondylolisthesis))+
  geom_density(adjust = 1/5) # Use 1/5 the bandwidth
```



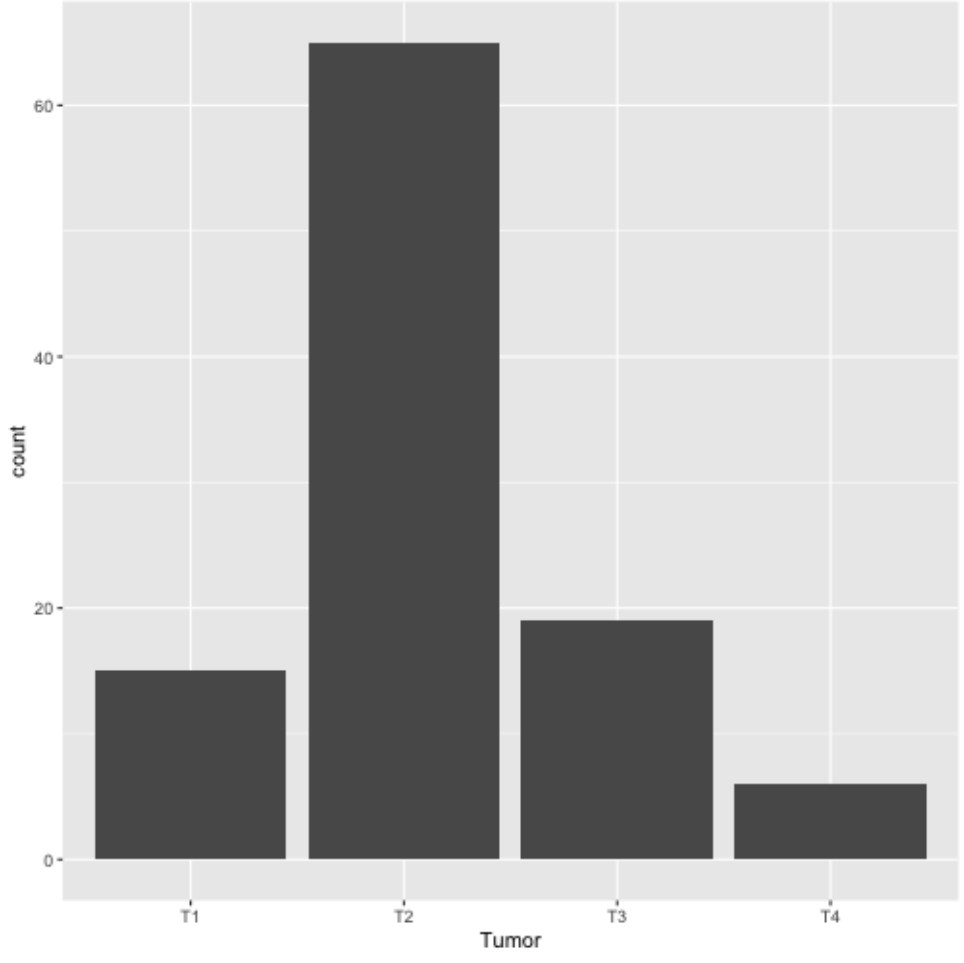
Layering geometries

```
ggplot(
  data=dat_spine,
  aes(x = degree_spondylolisthesis,
      y = stat(density)))+  # Re-scales histogram
  geom_histogram(bins = 100, fill='yellow') +
  geom_density(adjust = 1/5, color = 'orange', size =
```



Bar plots (categorical variable)

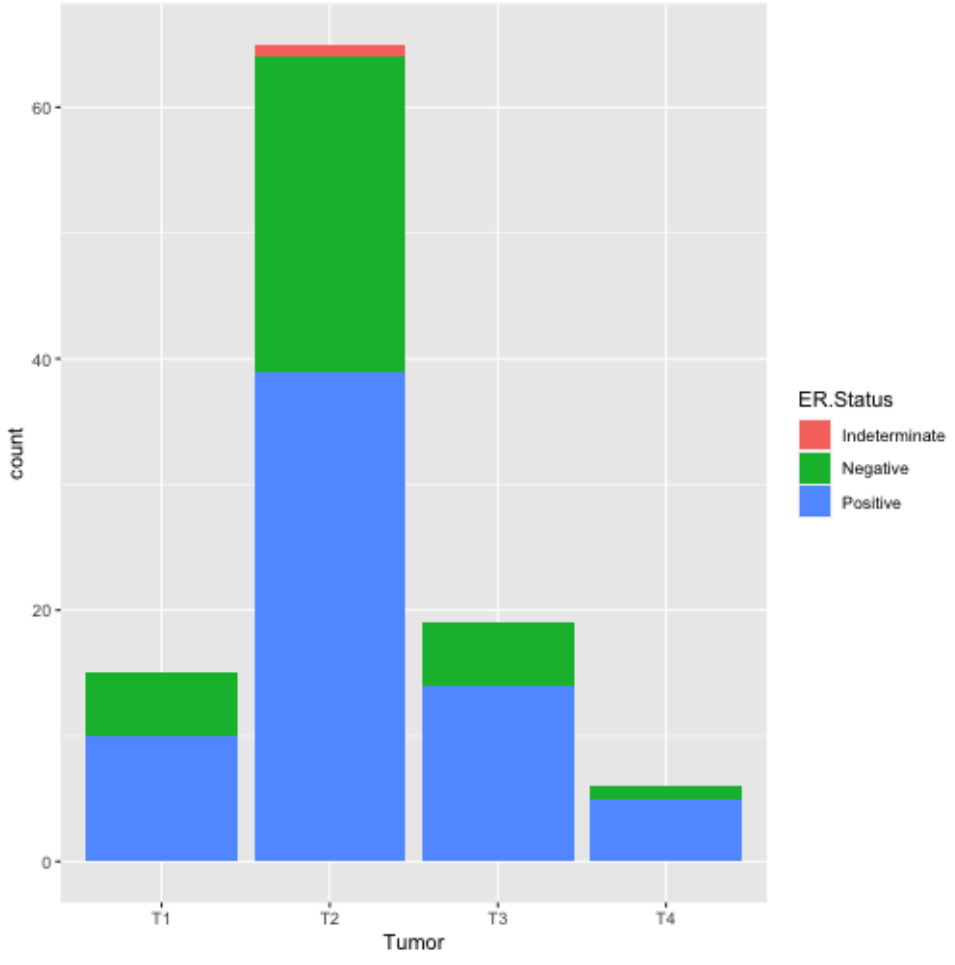
```
dat_brca <- rio::import('data/clinical_data_breast_ca
                        check.names = T)
ggplot(
  data=dat_brca,
  aes(x = Tumor))+
  geom_bar()
```



Bar plots (categorical variable)

```
dat_brca <- import('data/clinical_data_breast_cancer_
                   check.names = T)
ggplot(
  data=dat_brca,
  aes(x = Tumor,
      fill = ER.Status))+
  geom_bar()
```

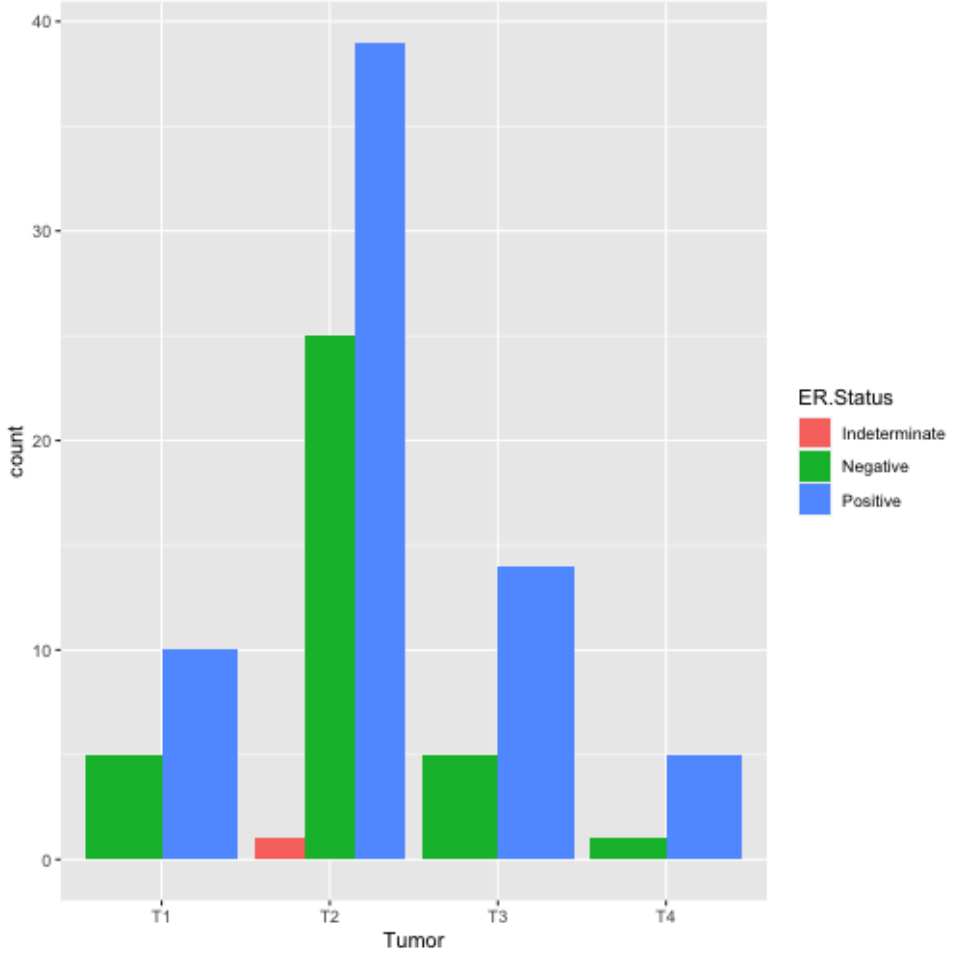
Add additional information via mapping



Bar plots (categorical variable)

```
dat_brca <- import('data/clinical_data_breast_cancer_
                   check.names = T)
ggplot(
  data=dat_brca,
  aes(x = Tumor,
      fill = ER.Status))+
  geom_bar(position = 'dodge')
    # Default is position = "stack"
```

Change the nature of the geometry



Graphing tabulated data

```
tabulated <- dat_brca %>% count(Tumor)
tabulated
```

```
#>    Tumor  n
#>  1    T1 15
#>  2    T2 65
#>  3    T3 19
#>  4    T4  6
```

```
ggplot(
  data = tabulated,
  aes(x = Tumor, y = n)) +
  geom_bar()
```

```
#>  Error: stat_count() can only have an x or y aesth
```

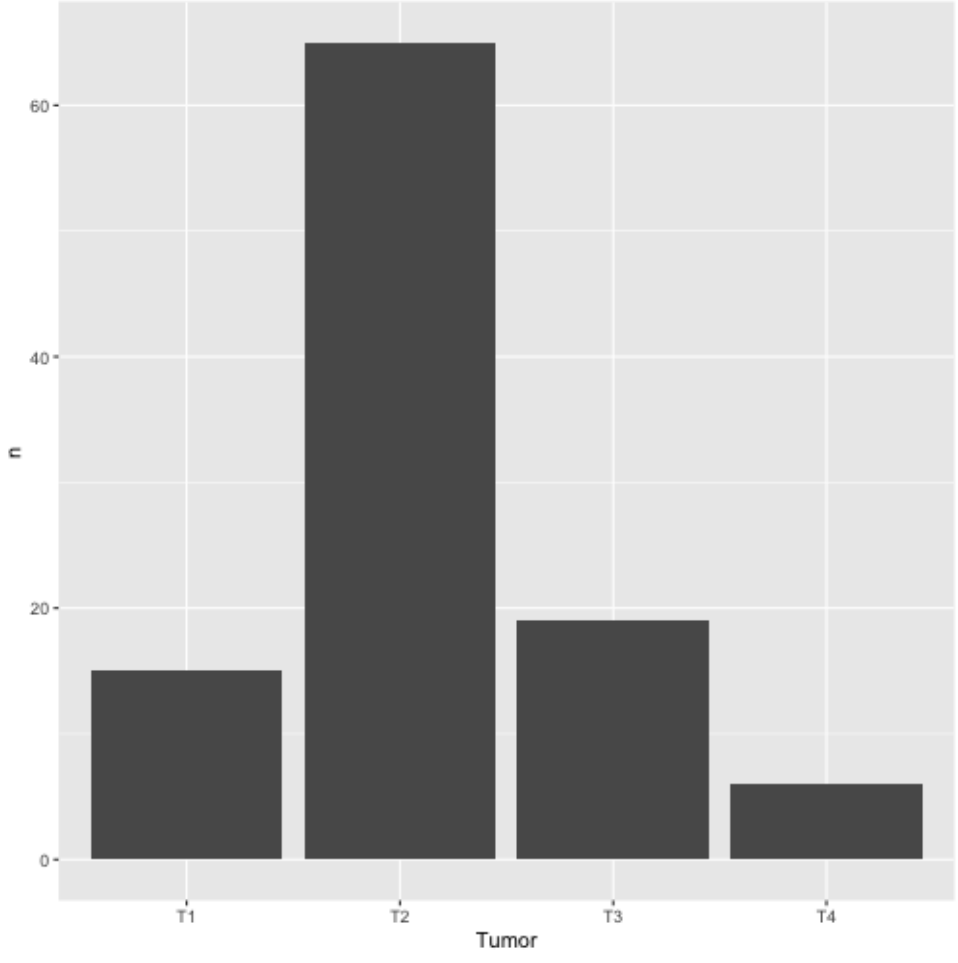
Graphing tabulated data

```
tabulated <- dat_brca %>% count(Tumor)
tabulated

ggplot(
  data = tabulated,
  aes(x = Tumor, y = n)) +
  geom_bar(stat = 'identity')
```

Here we need to change the default computation

The barplot usually computes the counts (`stat_count`)

We suppress that here since we have already done the computation



Peeking under the hood

```
plt <- ggplot(
  data = tabulated,
  aes(x = Tumor, y = n)) +
  geom_bar()

plt$layers
```

```
#>  [[1]]
#>  geom_bar: width = NULL, na.rm = FALSE, orientatic
#>  stat_count: width = NULL, na.rm = FALSE, orientat
#>  position_stack
```

```
plt <- ggplot(
  data = tabulated,
  aes(x = Tumor, y = n)) +
  geom_bar(stat = 'identity')

plt$layers
```

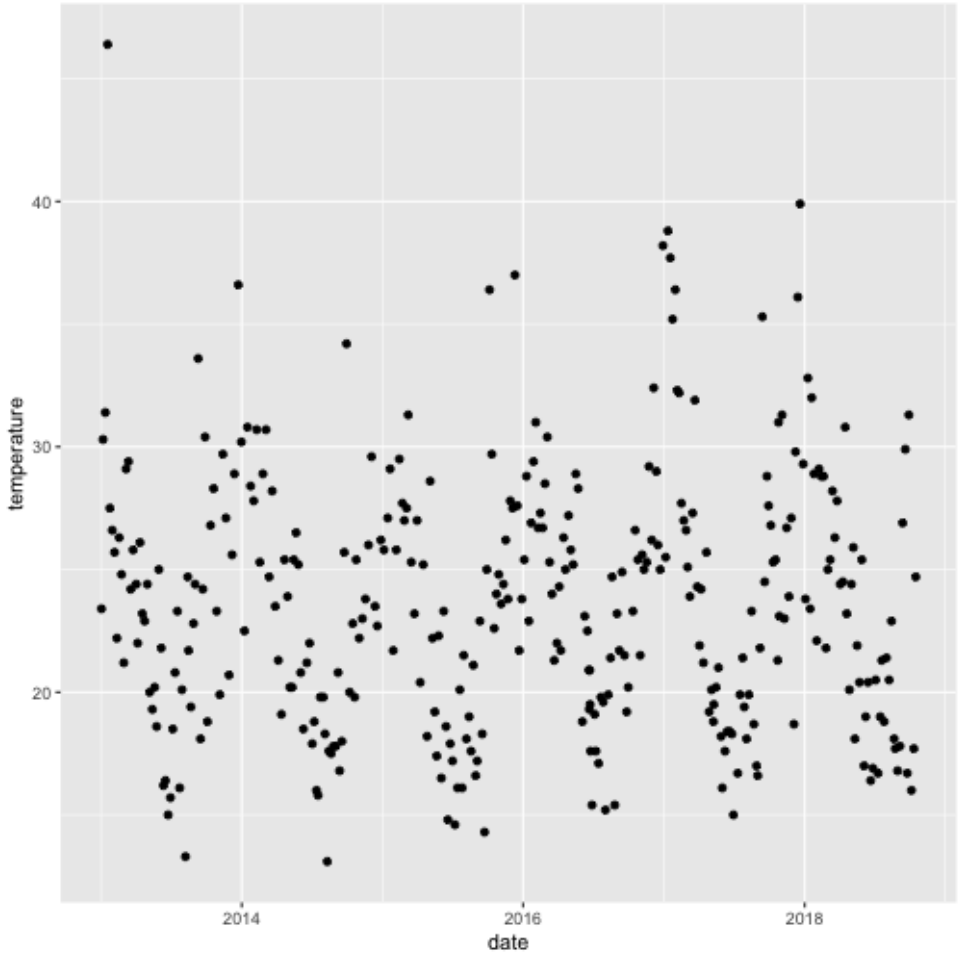
```
#>  [[1]]
#>  geom_bar: width = NULL, na.rm = FALSE, orientatic
#>  stat_identity: na.rm = FALSE
#>  position_stack
```

> Each layer has a geometry, statistic and position associated with it

Bivariate plots

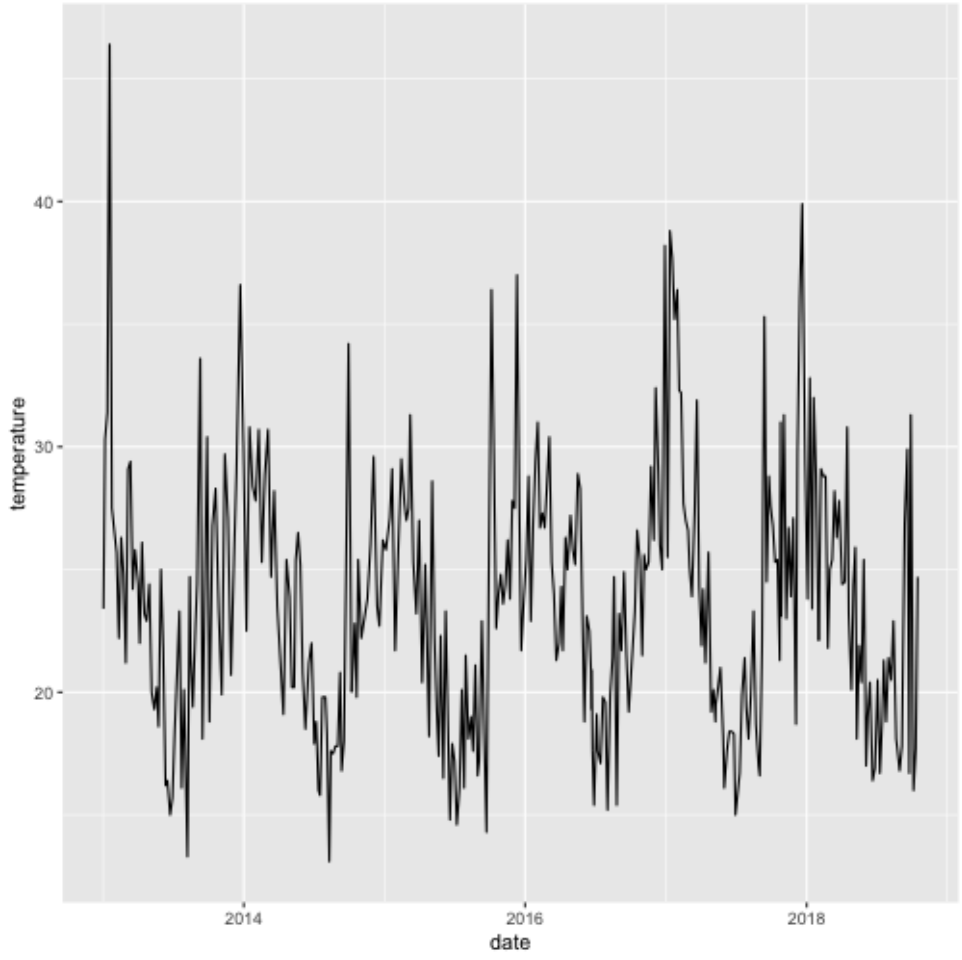
Scatter plots

```
ggplot(
  data = beaches,
  aes(x = date, y = temperature))+
  geom_point()
```



Scatter plots

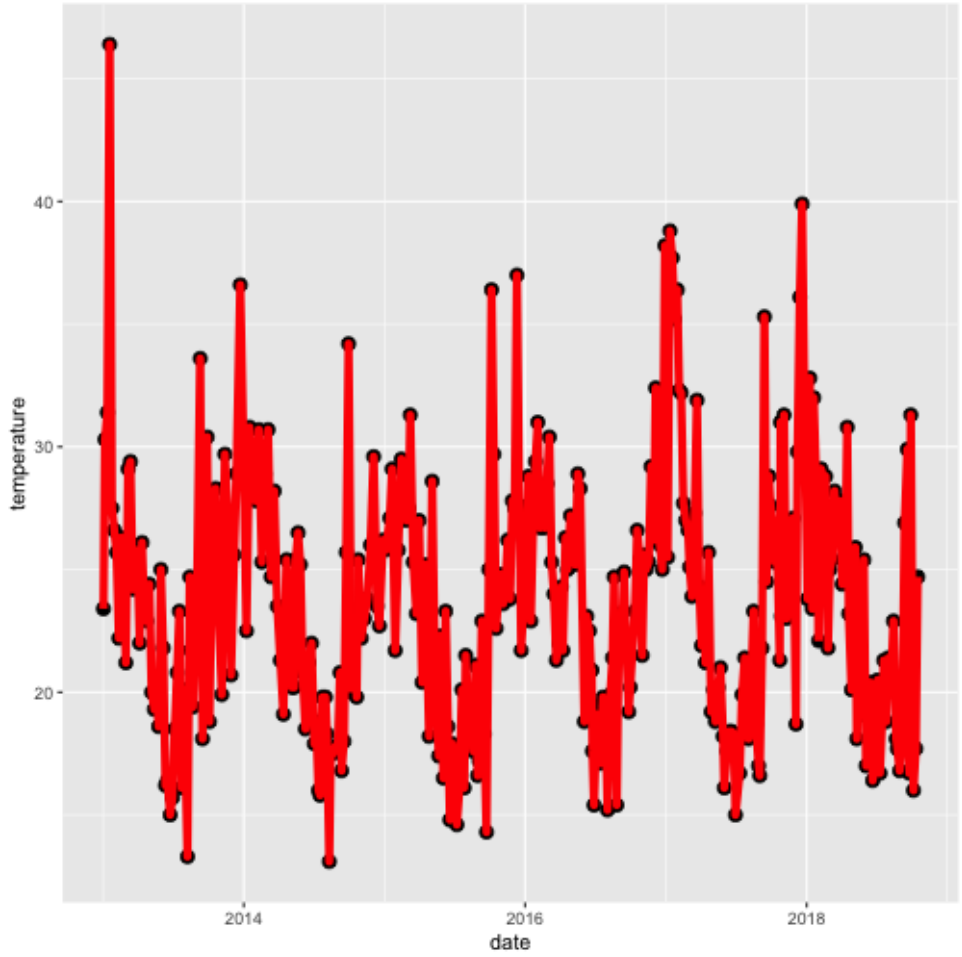
```
ggplot(
  data = beaches,
  aes(x = date, y = temperature))+
  geom_line()
```



Scatter plots

```
ggplot(
  data = beaches,
  aes(x = date, y = temperature))+
  geom_point(color='black', size = 3) +
  geom_line(color='red',size=2)
```

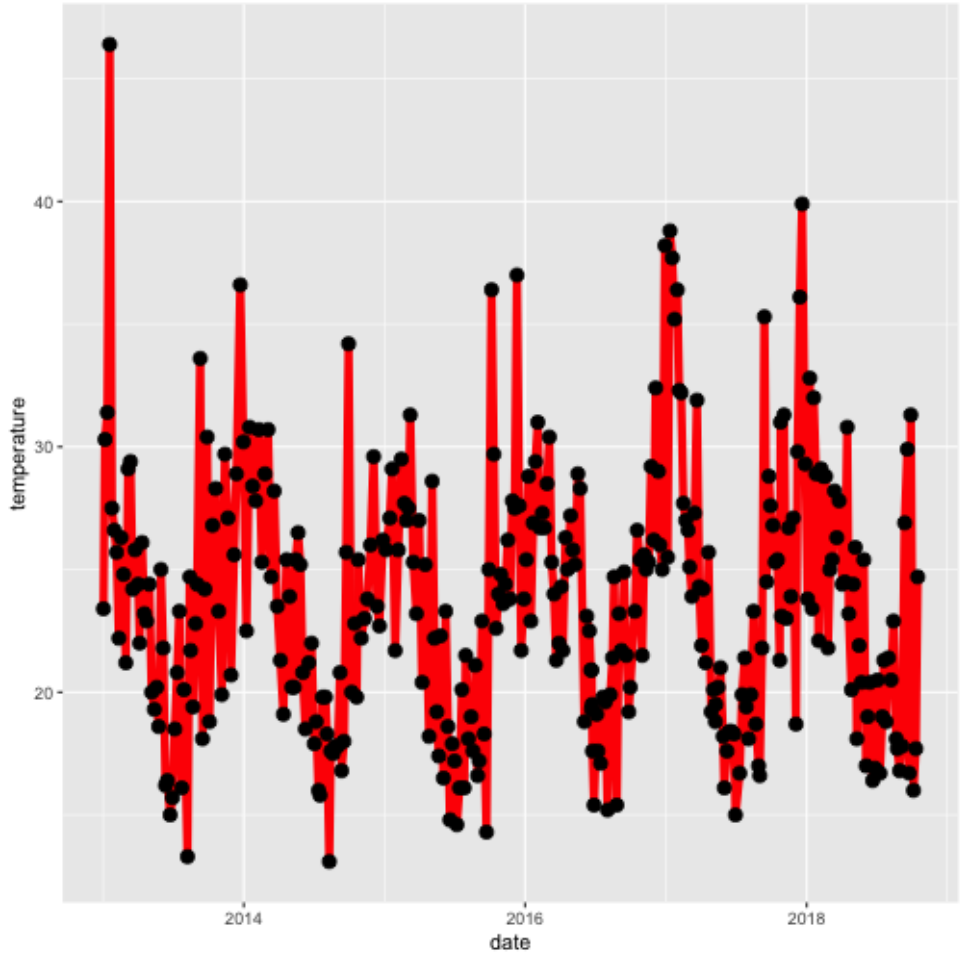
Layer points and lines



Scatter plots

```
ggplot(
  data = beaches,
  aes(x = date, y = temperature))+
  geom_line(color='red',size=2) +
  geom_point(color='black', size = 3)
```

Order of laying down geometries matters



Doing some computations

```
ggplot(
  data = beaches,
  aes(x = date, y = temperature)) +
  geom_point() +
  geom_smooth()
```

```
#>  `geom_smooth()` using method = 'loess' and formul
```

Averages over 75% of the data

Doing some computations

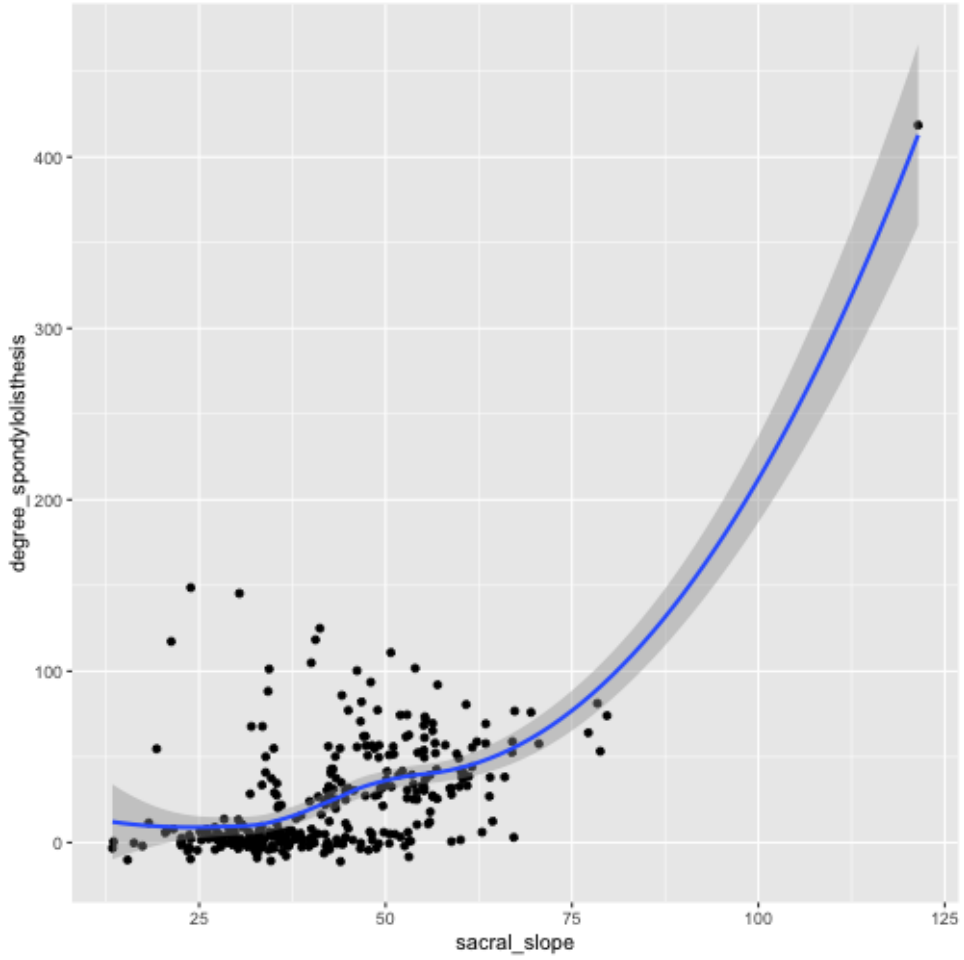
```
ggplot(
  data = beaches,
  aes(x = date, y = temperature)) +
  geom_point() +
  geom_smooth(span = 0.1)
```

```
#>  `geom_smooth()` using method = 'loess' and formul
```

Averages over 10% of the data

Computations over groups

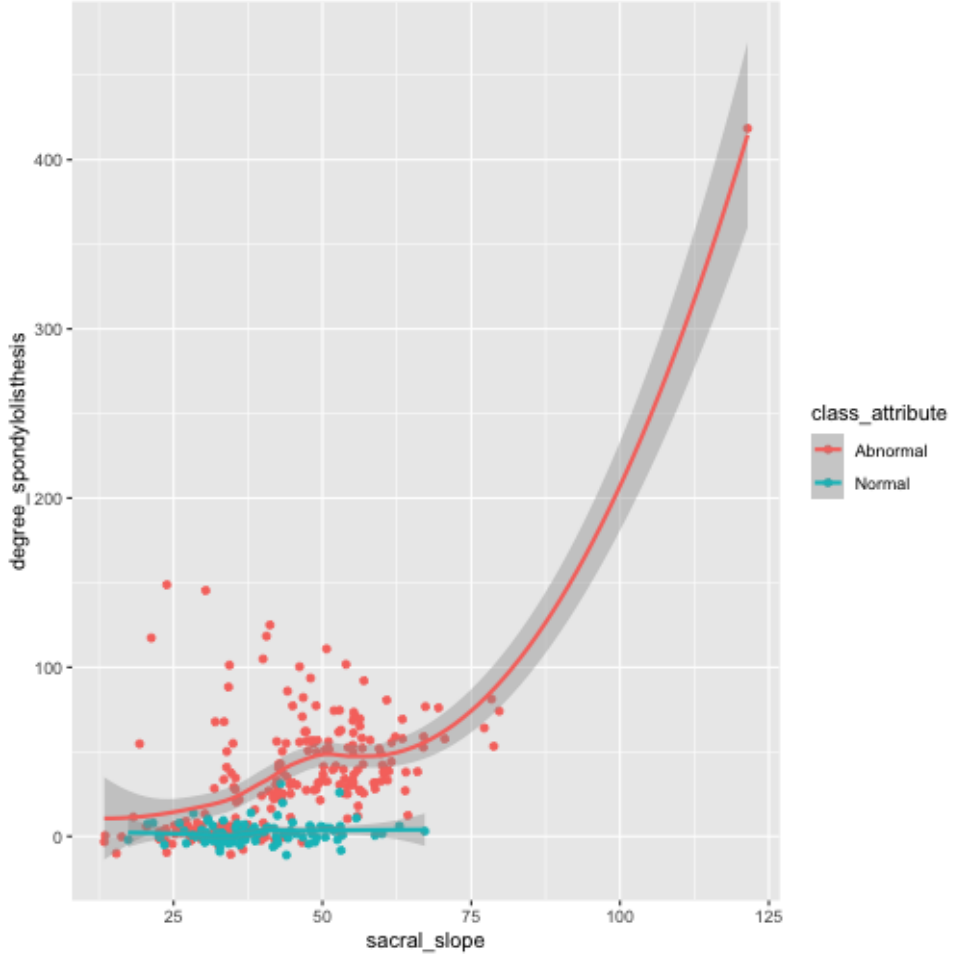
```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope,
      y = degree_spondylolisthesis)) +
  geom_point() +
  geom_smooth()
```



Computations over groups

```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope,
      y = degree_spondylolisthesis,
      color = class_attribute)) +
  geom_point() +
  geom_smooth()
```

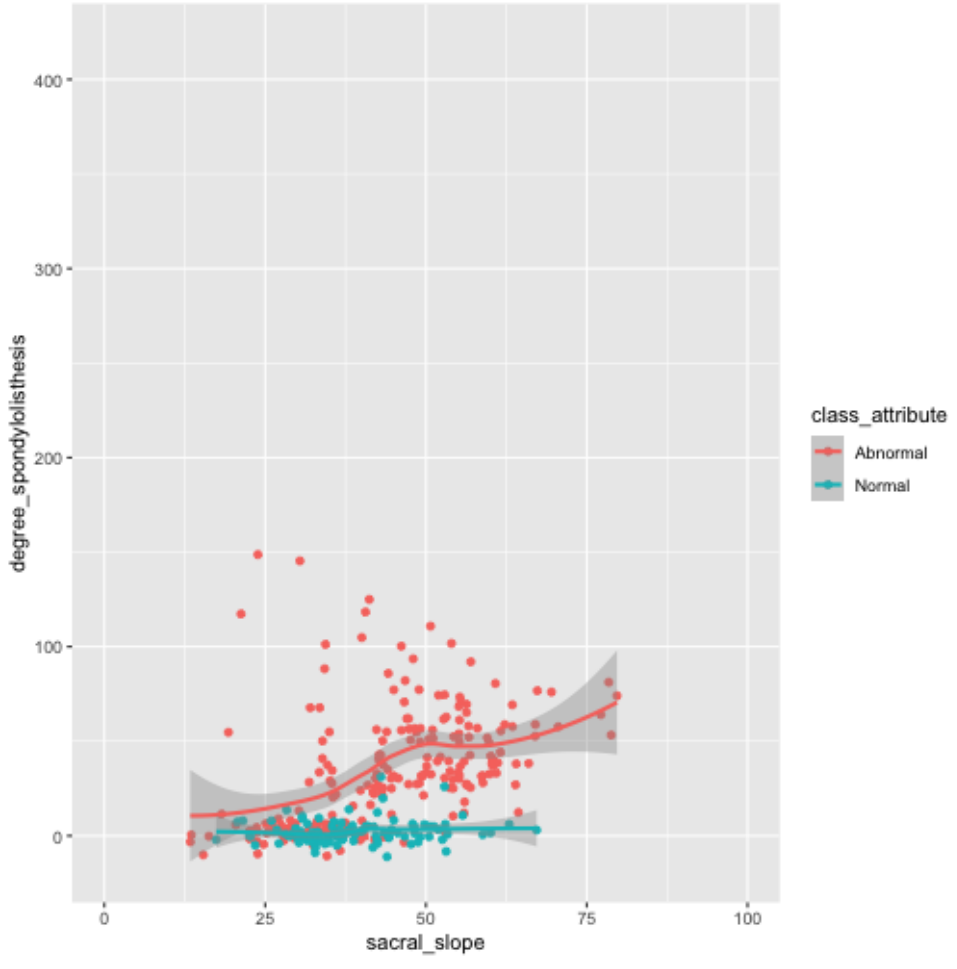
Computation is done by groups



Computations over groups

```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope,
      y = degree_spondylolisthesis,
      color = class_attribute)) +
  geom_point() +
  geom_smooth() +
  xlim(0, 100)
```

Ignore the outlier for now

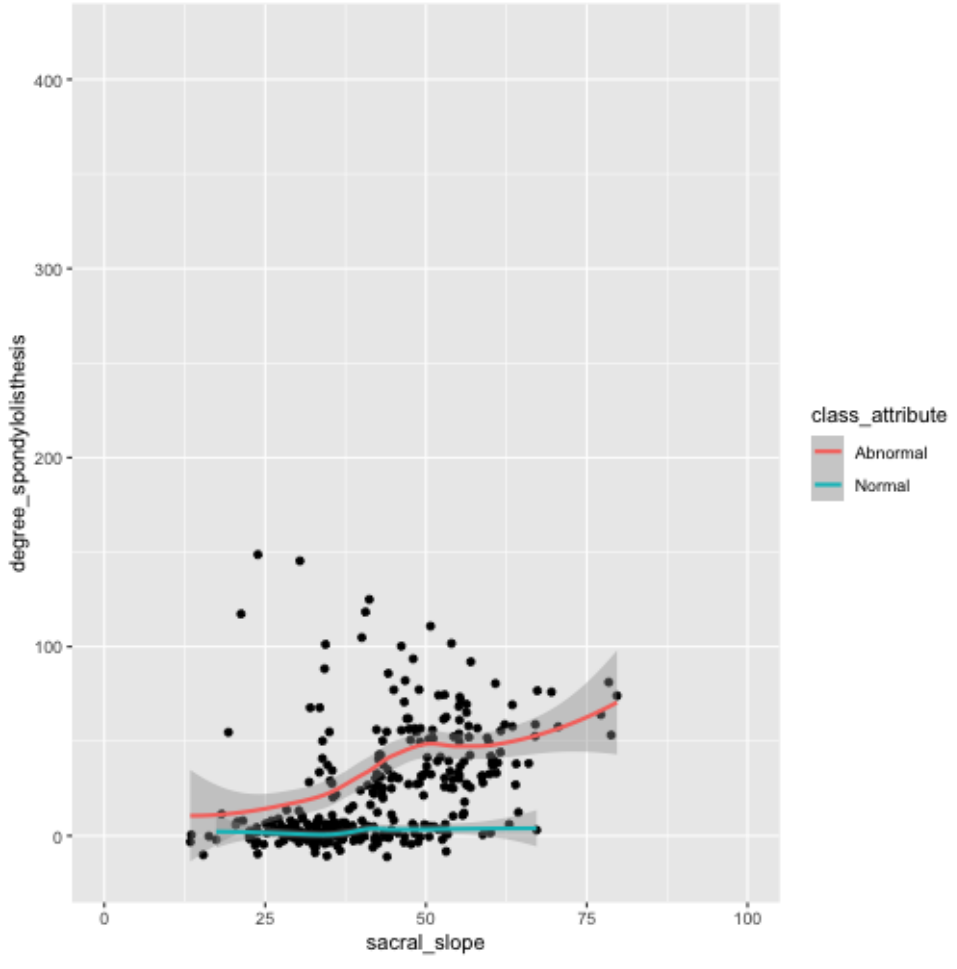


Computations over groups

```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope,
      y = degree_spondylolisthesis)) +
  geom_point() +
  geom_smooth(aes(color = class_attribute)) +
  xlim(0, 100)
```

Only color-code the smoothers

You can change the plot based on where you map the aesthetic

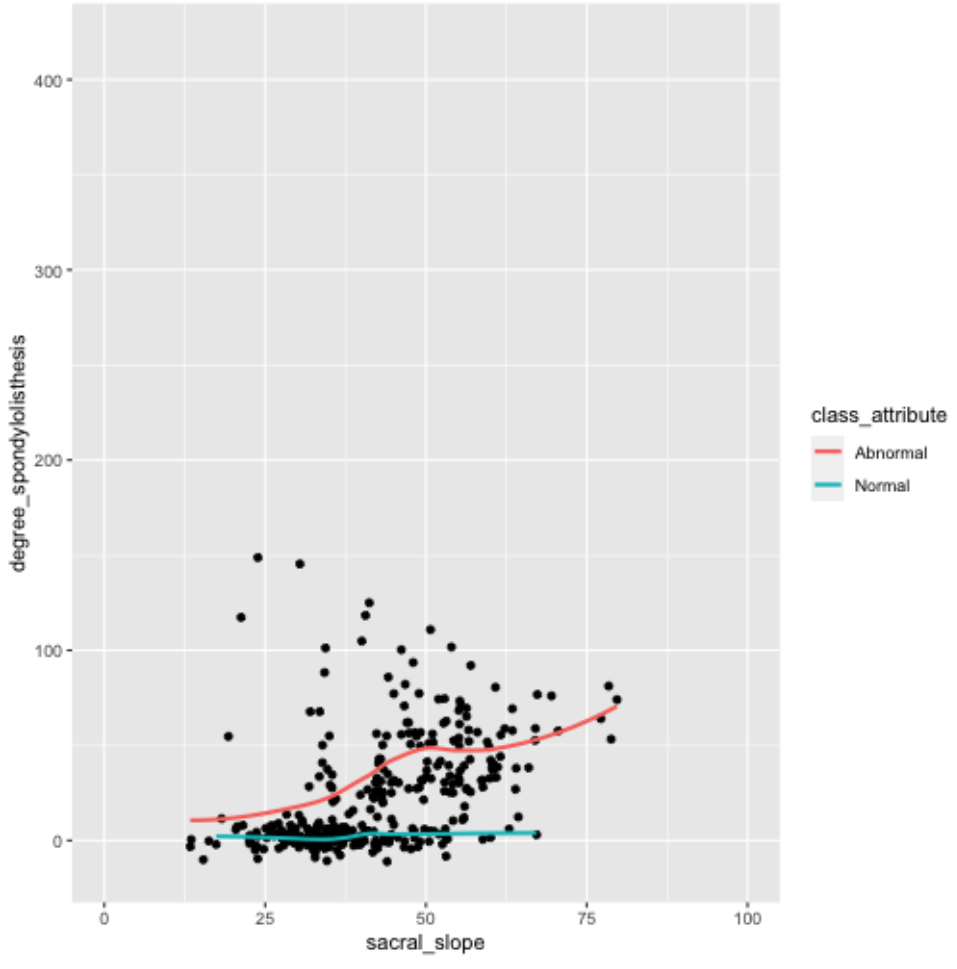


Computations over groups

```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope,
      y = degree_spondylolisthesis)) +
  geom_point() +
  geom_smooth(aes(color = class_attribute),
              se = F) +
  xlim(0, 100)
```

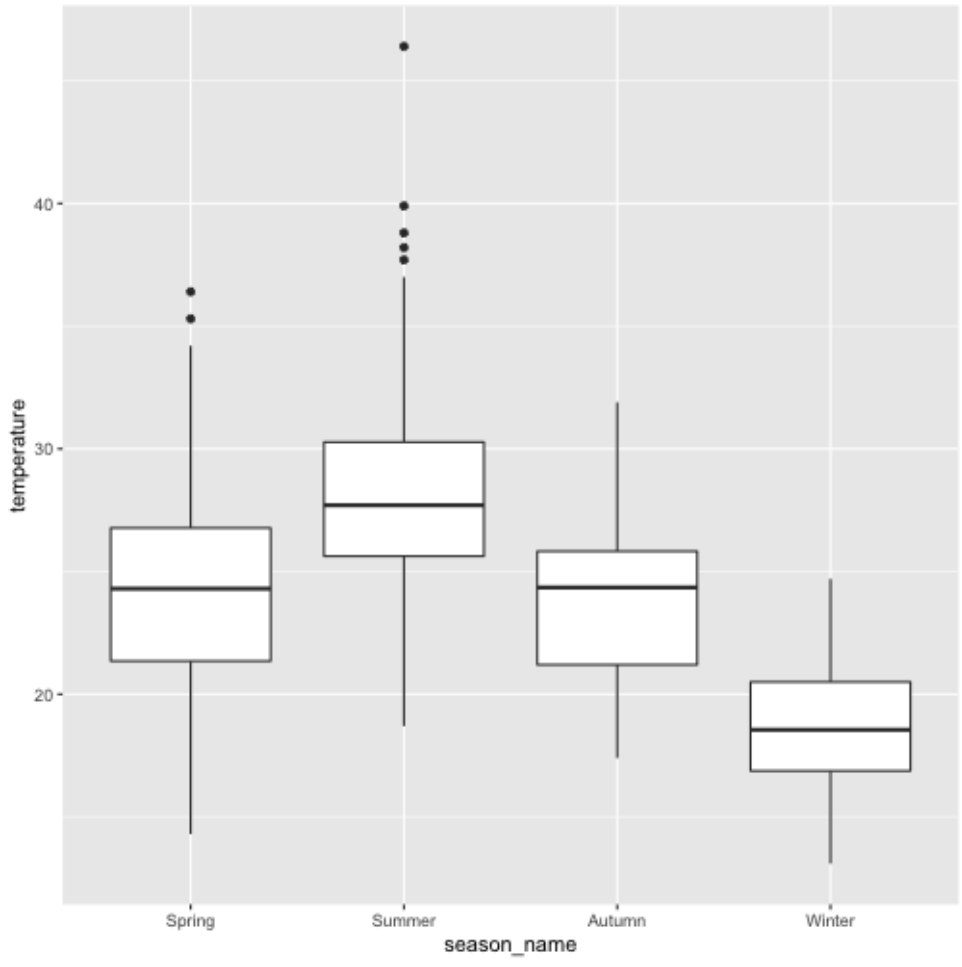
Remove the confidence bands

Maybe a cleaner look



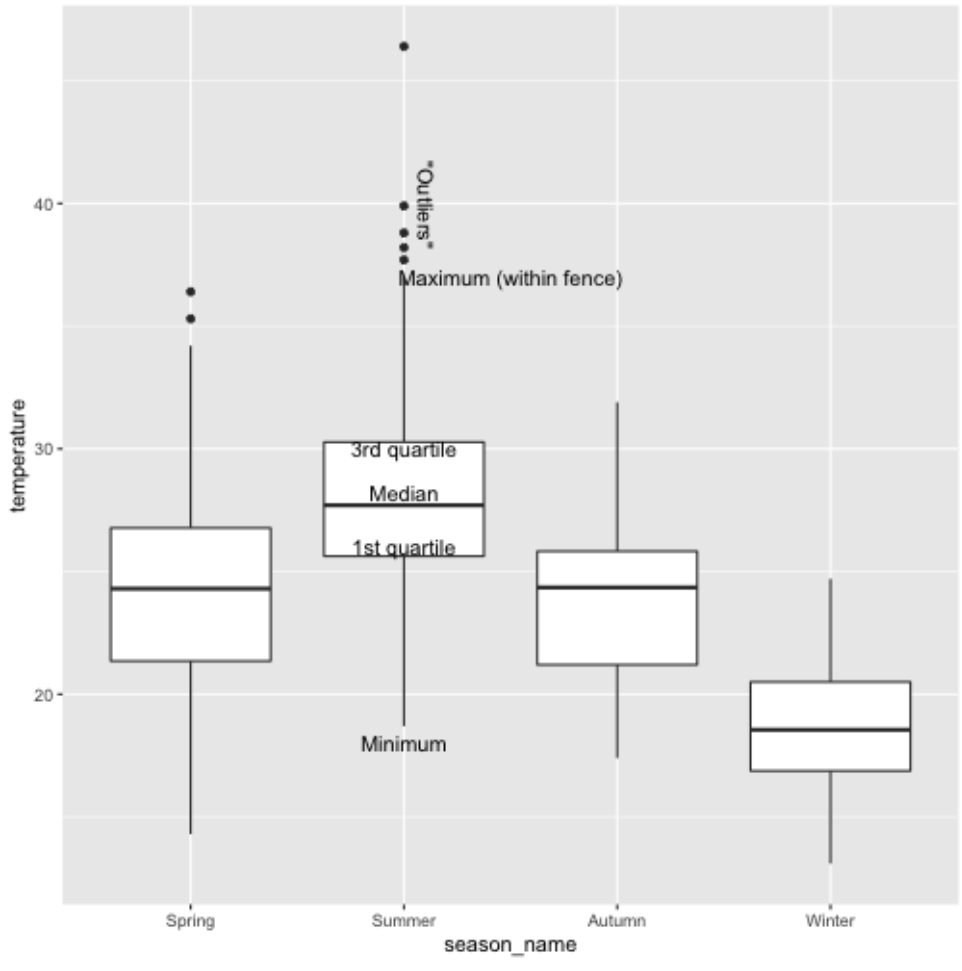
Box Plots

```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_boxplot()
```



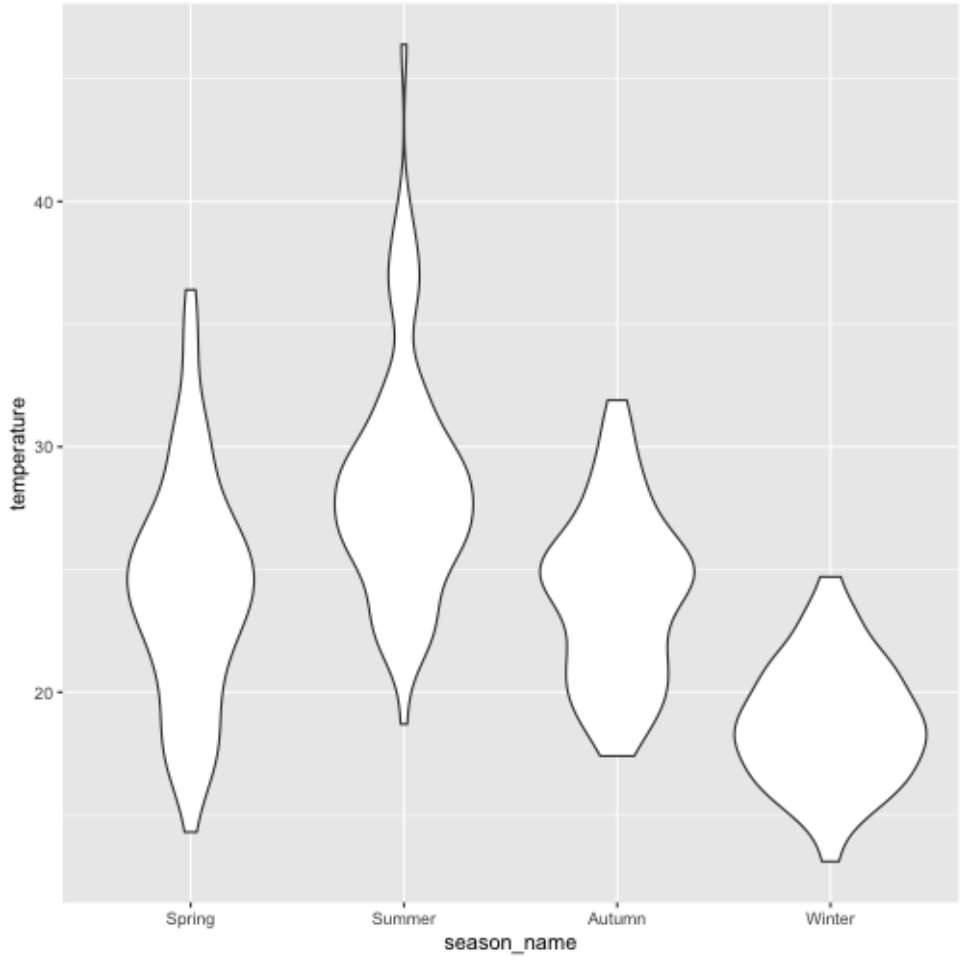
Box Plots

```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_boxplot()
```



Violin plots

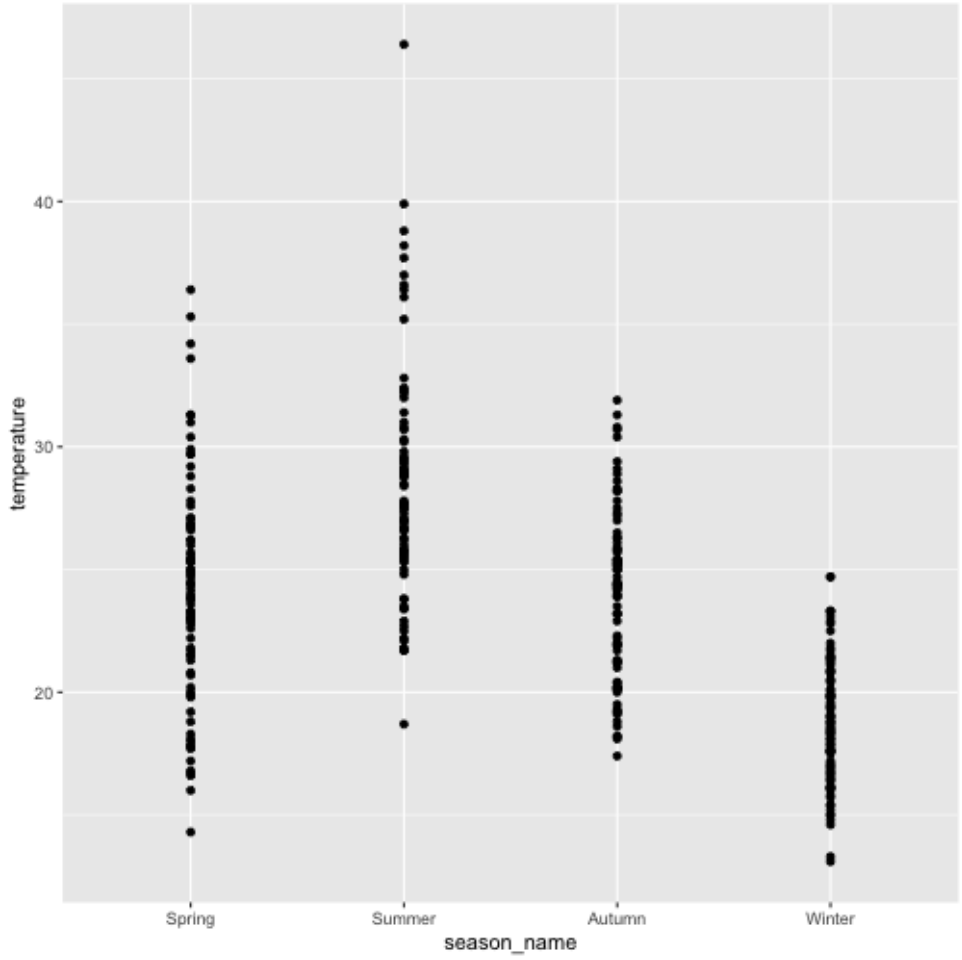
```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_violin()
```



Strip plots

```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_point()
```

Points are overlayed on each other

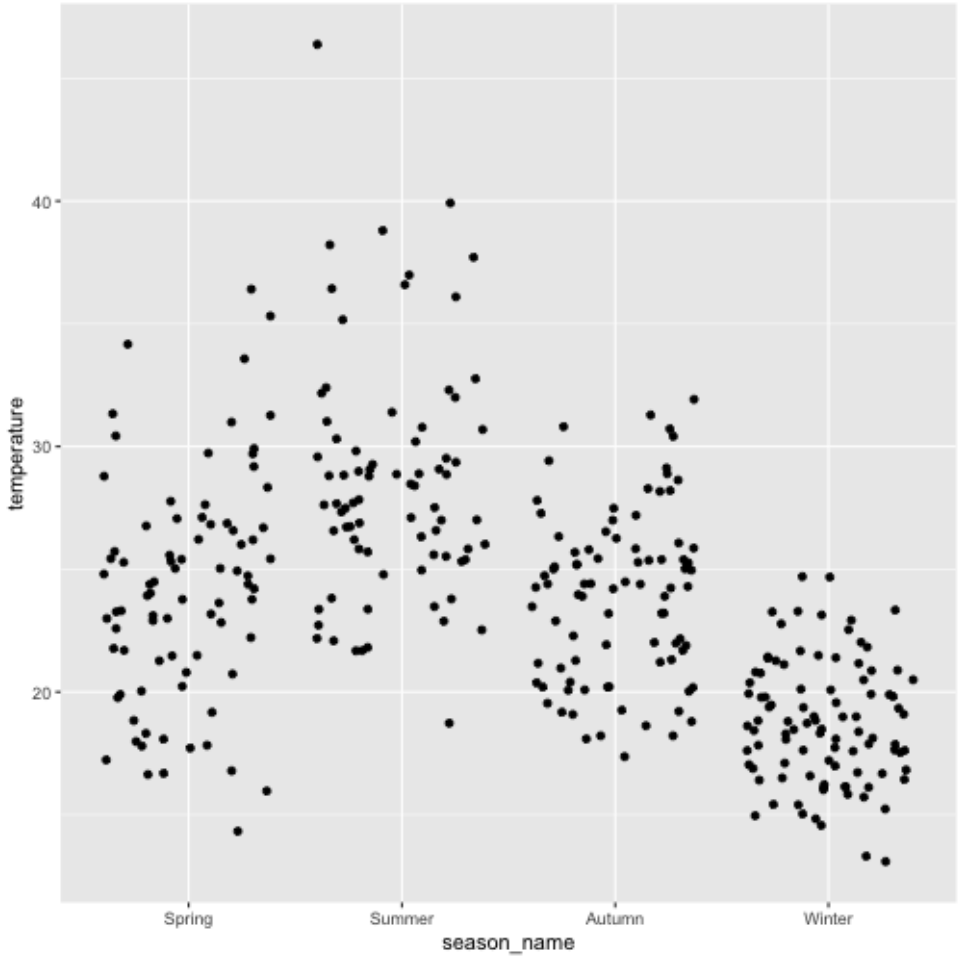


Strip plots

```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_jitter()
```

Jittering allows all points to be seen

Maybe too much

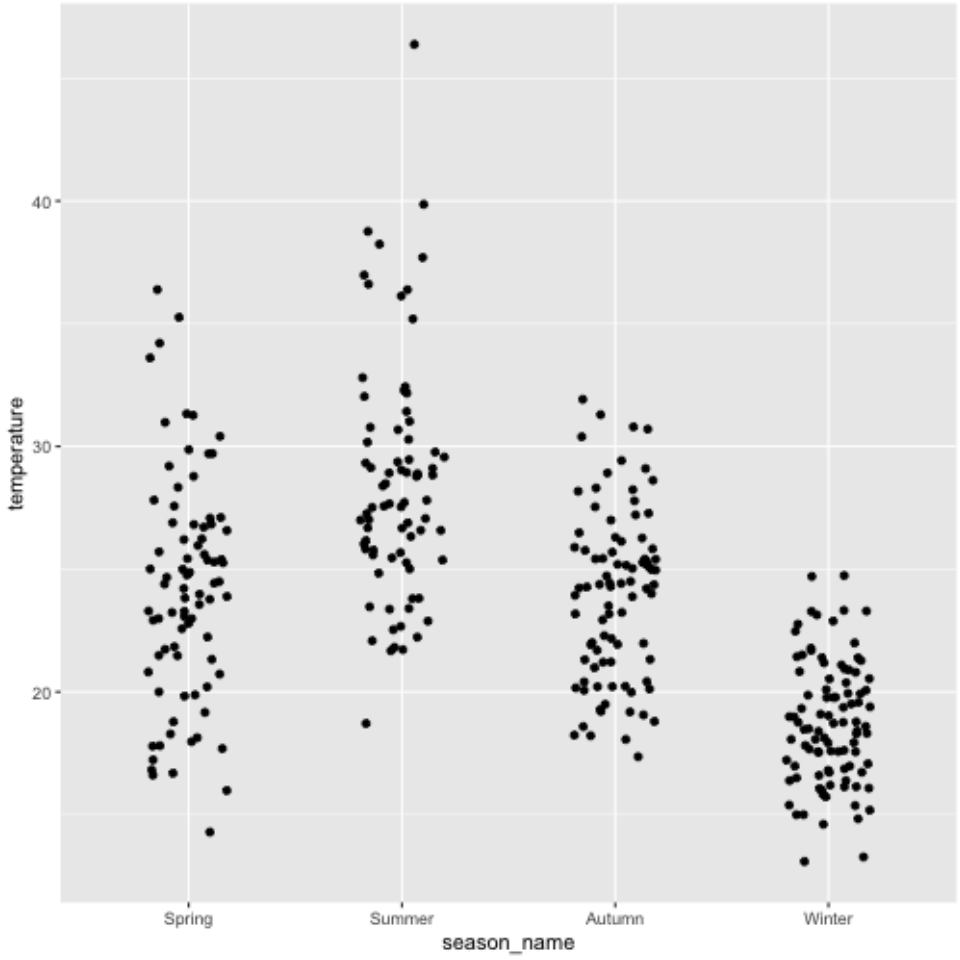


Strip plots

```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_jitter(width = 0.2)
```

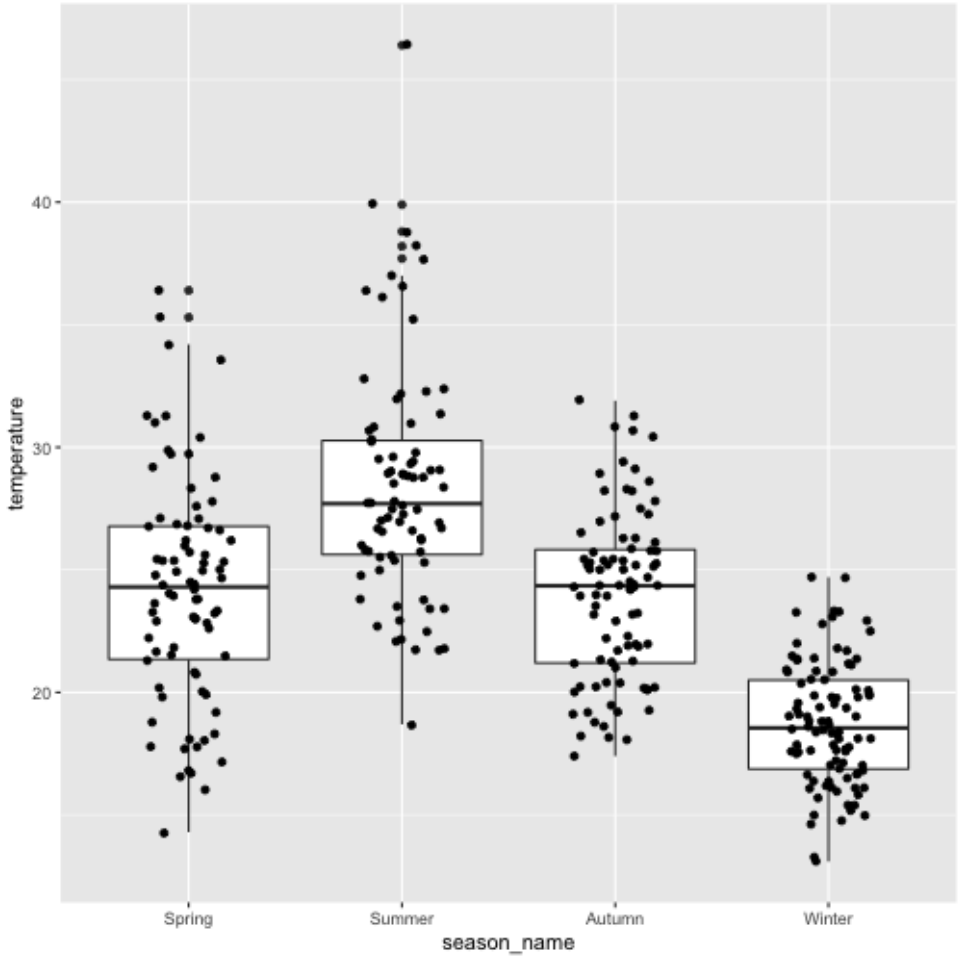
Jittering allows all points to be seen

Maybe too much



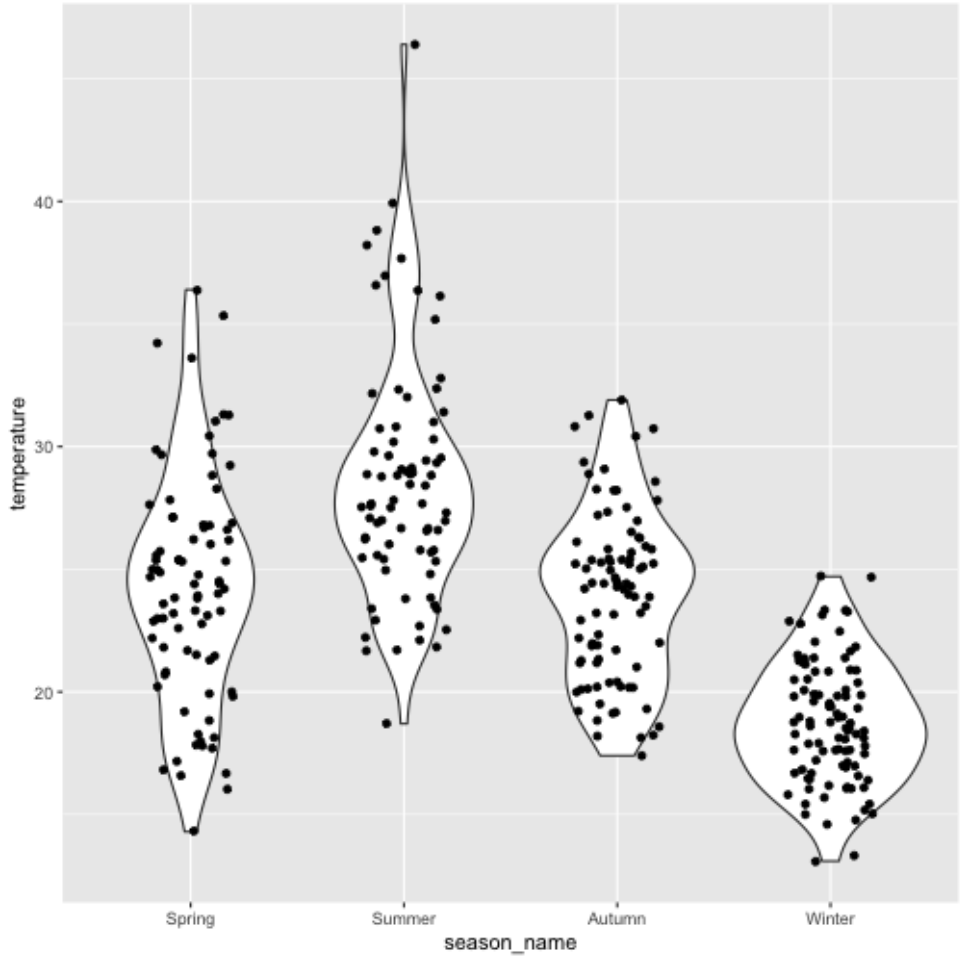
Layers, again

```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_boxplot() +
  geom_jitter(width = 0.2)
```



Layers, again

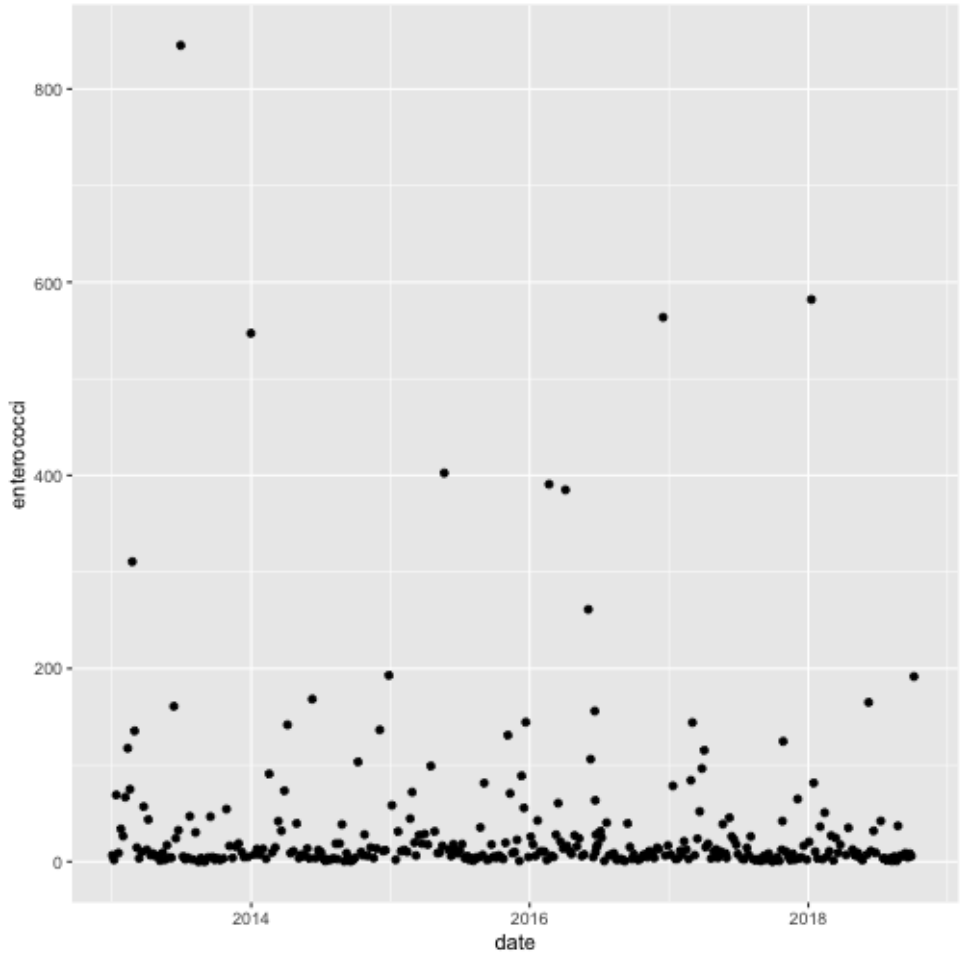
```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_violin() +
  geom_jitter(width = 0.2)
```



Scales

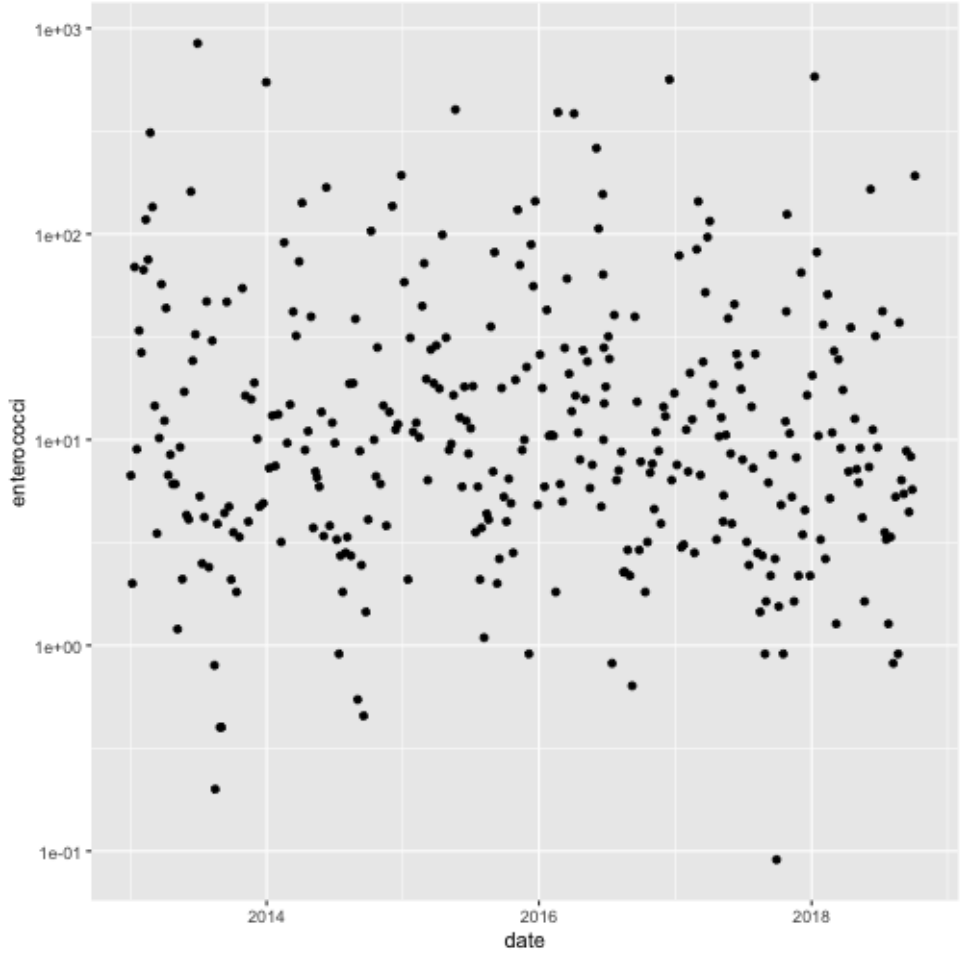
```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci)) +
  geom_point()
```

All the action is happening in the bottom bit



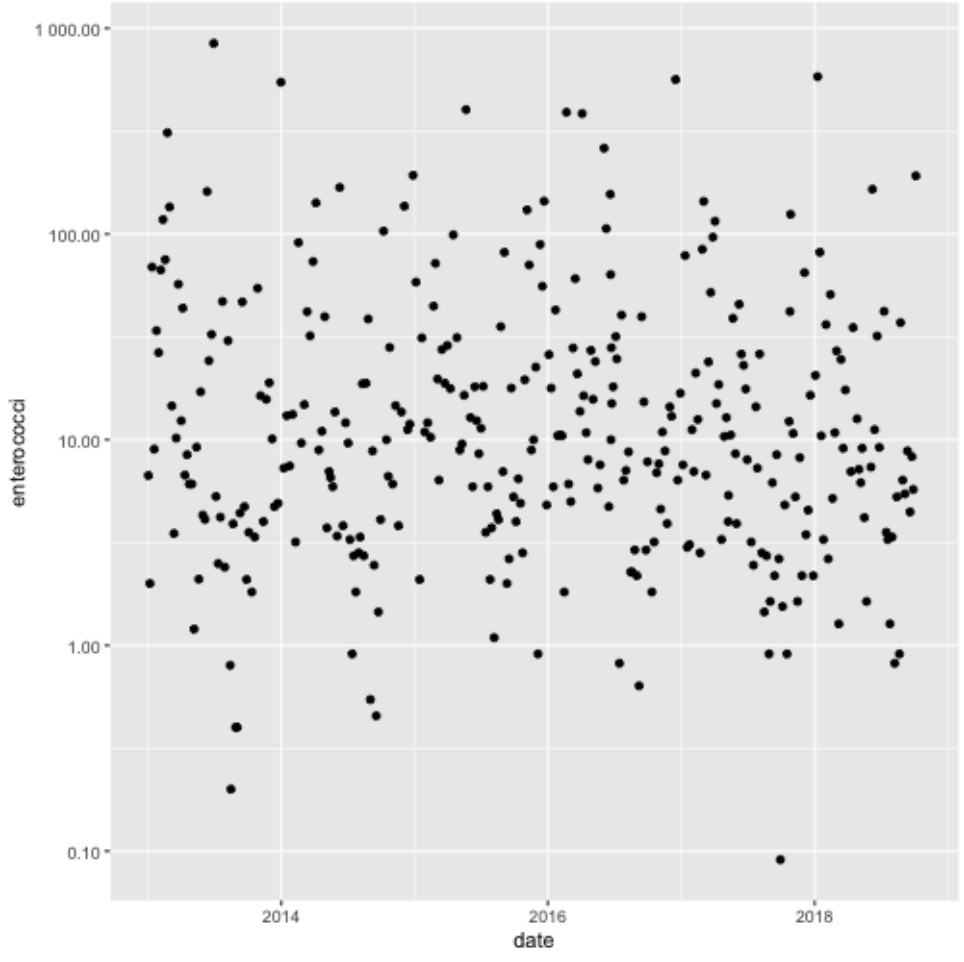
```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci)) +
  geom_point() +
  scale_y_log10()
```

Log-transforming an axis can make things easier to see



```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci)) +
  geom_point() +
  scale_y_log10(
    labels = scales::number_format(digits=3))
```

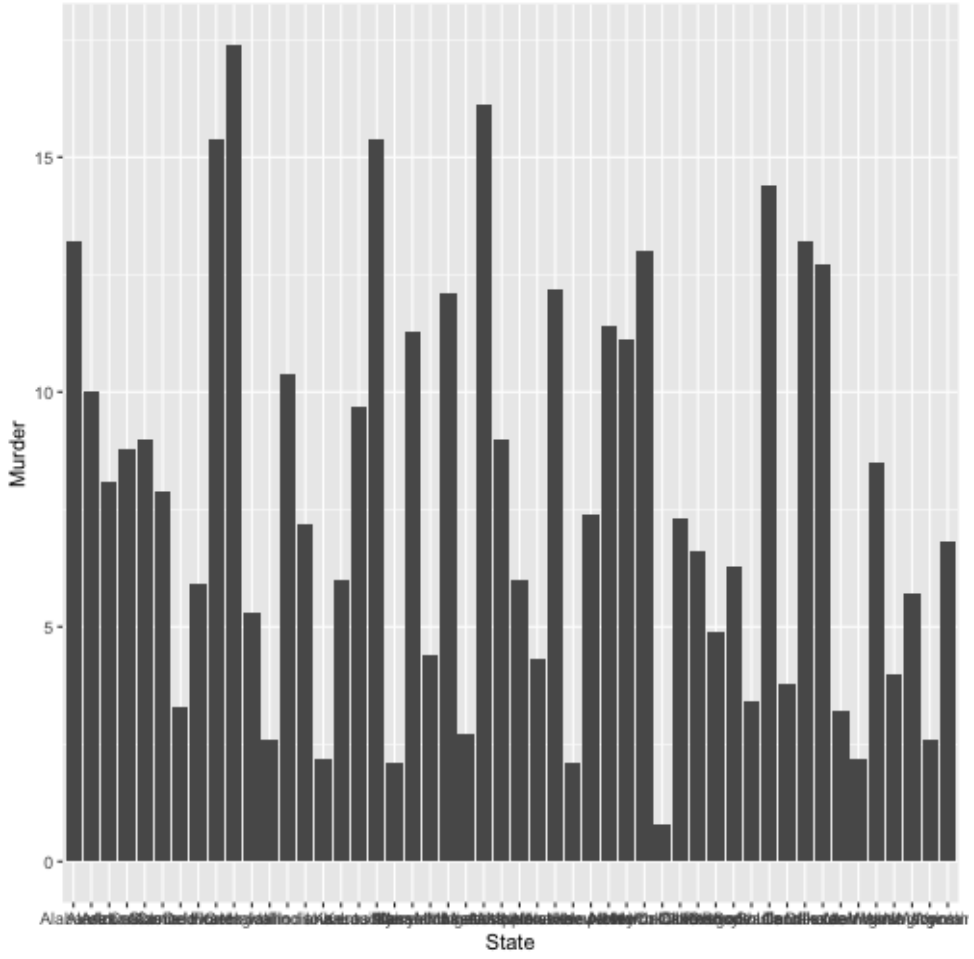
Making the labels a bit easier to read



Order and orientation

Arrests in the USA in 1973

```
arrests <- import('data/USArrests.csv')
ggplot(
  data = arrests,
  aes(x = State,
      y = Murder)) +
  geom_bar(stat = 'identity')
```



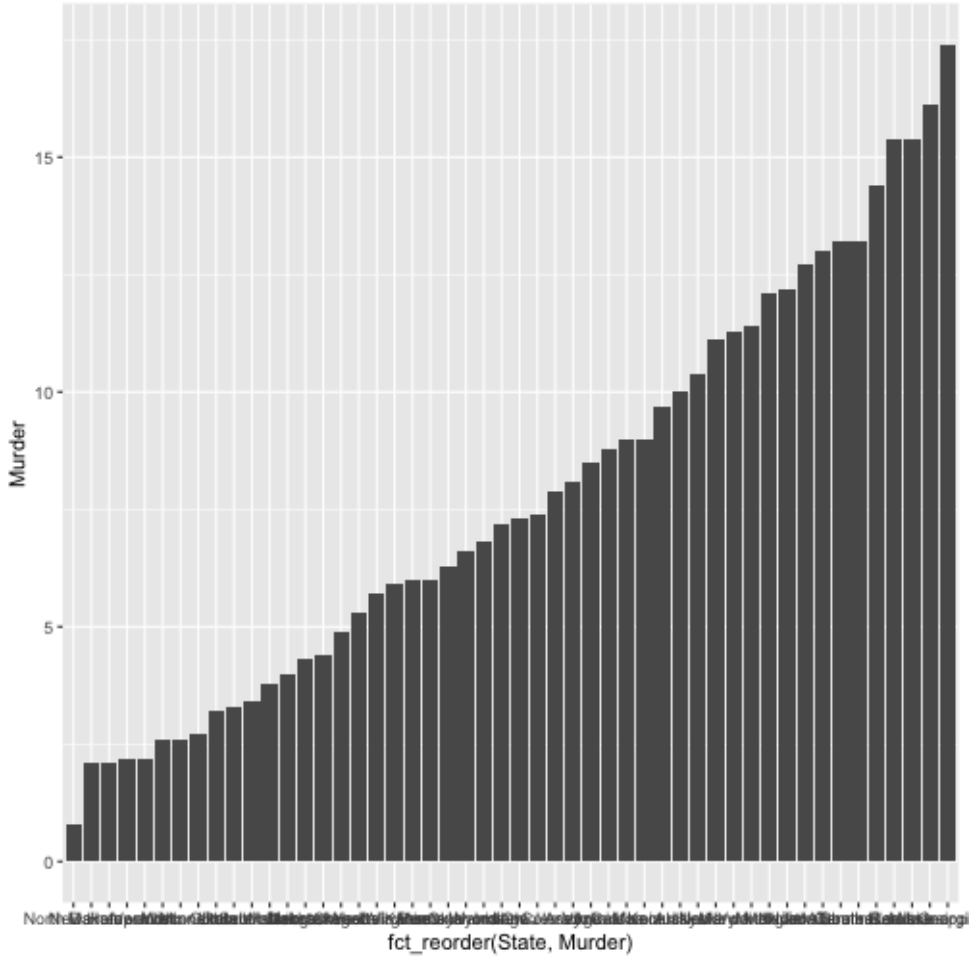
This plot is very hard to read

There is no ordering, and states can't be read

Arrests in the USA in 1973

```
arrests <- import('data/USArrests.csv')
ggplot(
  data = arrests,
  aes(x = fct_reorder(State, Murder), # Order by murd
      y = Murder)) +
  geom_bar(stat = 'identity')
```

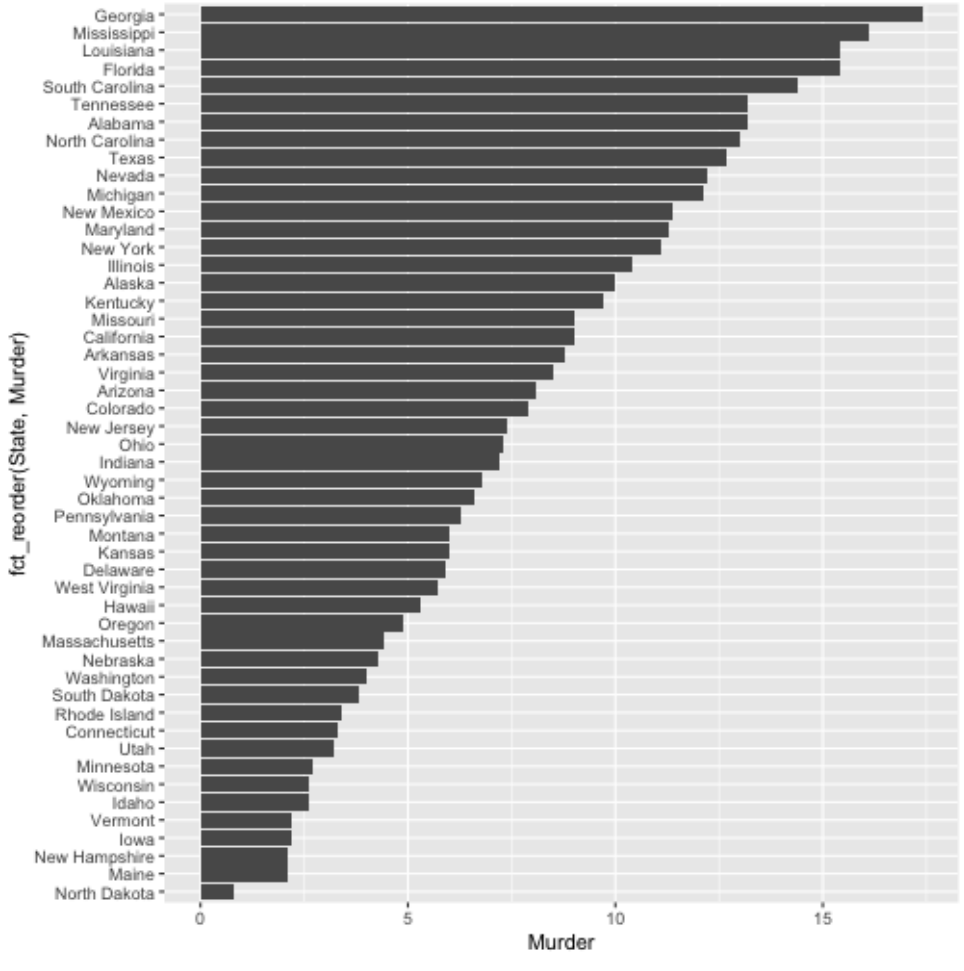
We see the pattern, but its still unreadable



Arrests in the USA in 1973

```
arrests <- import('data/USArrests.csv')
ggplot(
  data = arrests,
  aes(x = fct_reorder(State, Murder), # Order by murd
      y = Murder)) +
  geom_bar(stat = 'identity') +
  coord_flip()
```

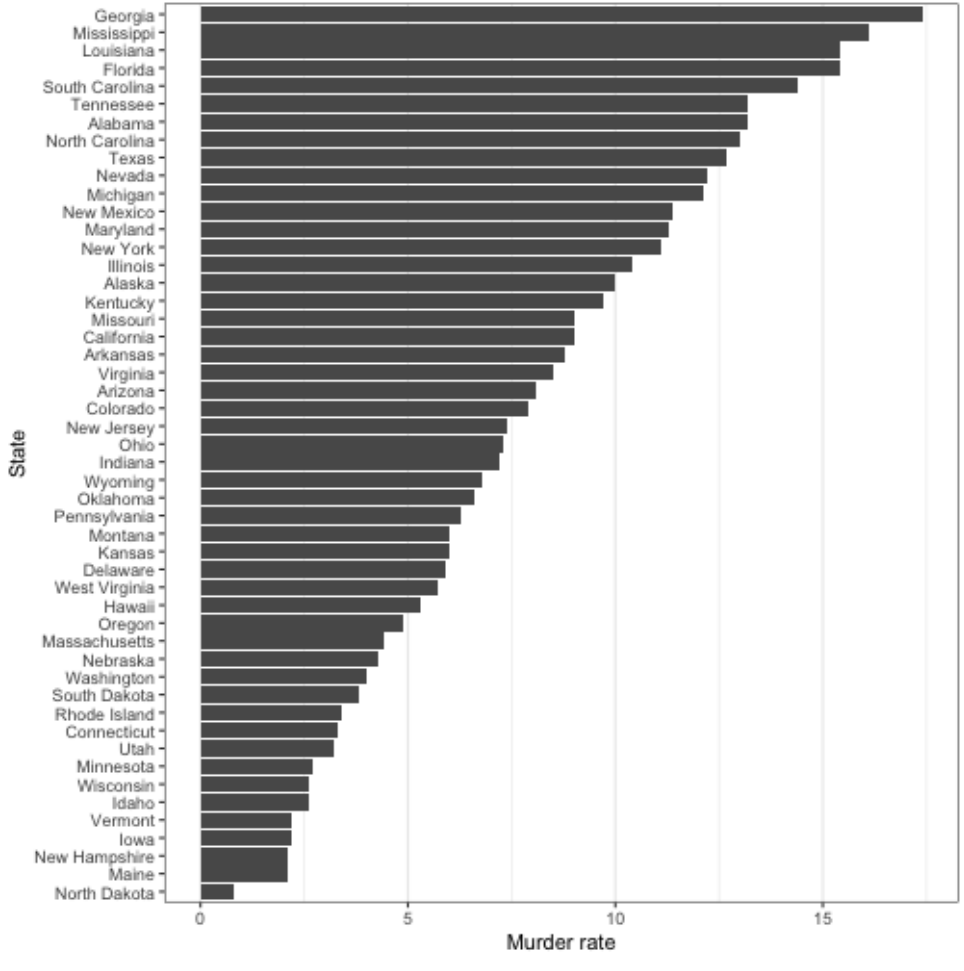
Flipping the axes makes the states readable



Arrests in the USA in 1973

```
arrests <- import('data/USArrests.csv')
ggplot(
  data = arrests,
  aes(x = fct_reorder(State, Murder), # Order by murd
      y = Murder)) +
  geom_bar(stat = 'identity') +
  labs(x = 'State', y = 'Murder rate') +
  theme_bw() +
  theme(panel.grid.major.y = element_blank(),
        panel.grid.minor.y = element_blank()) +
  coord_flip()
```

Cleaning it up a little



Themes

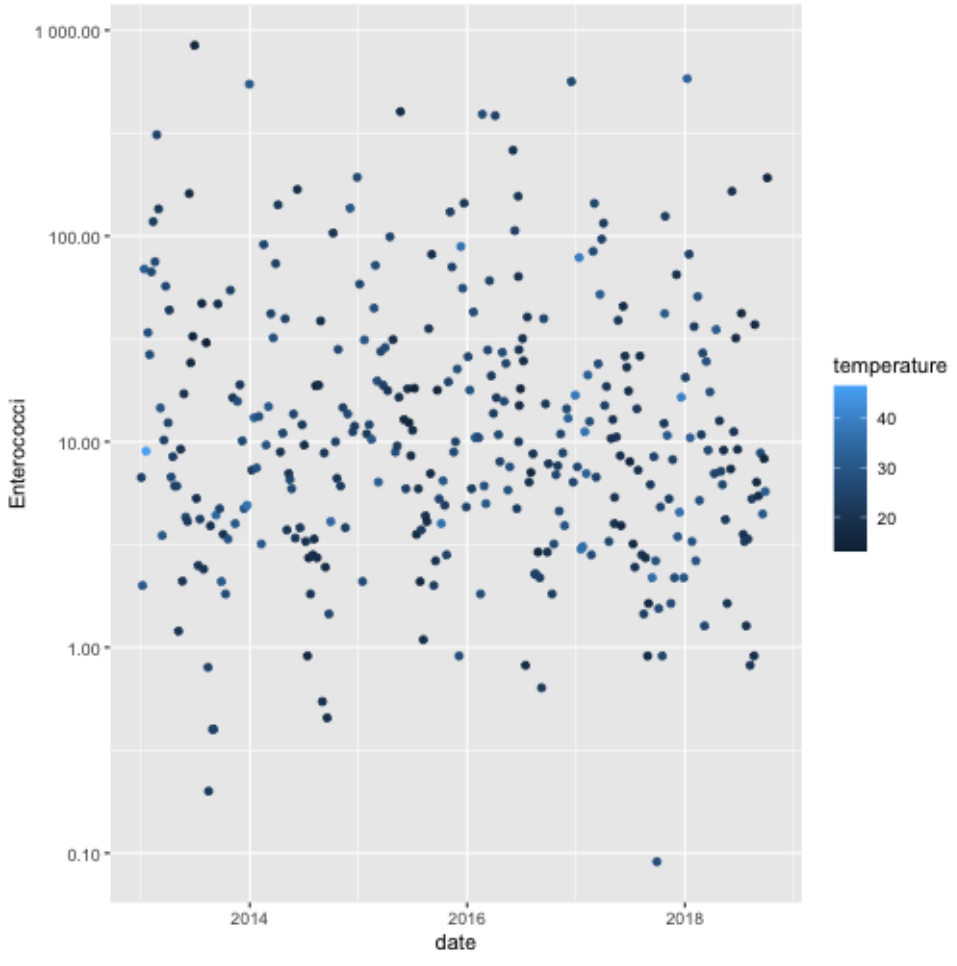
Color schemes

ggplot comes with a default color scheme. There are several other schemes available

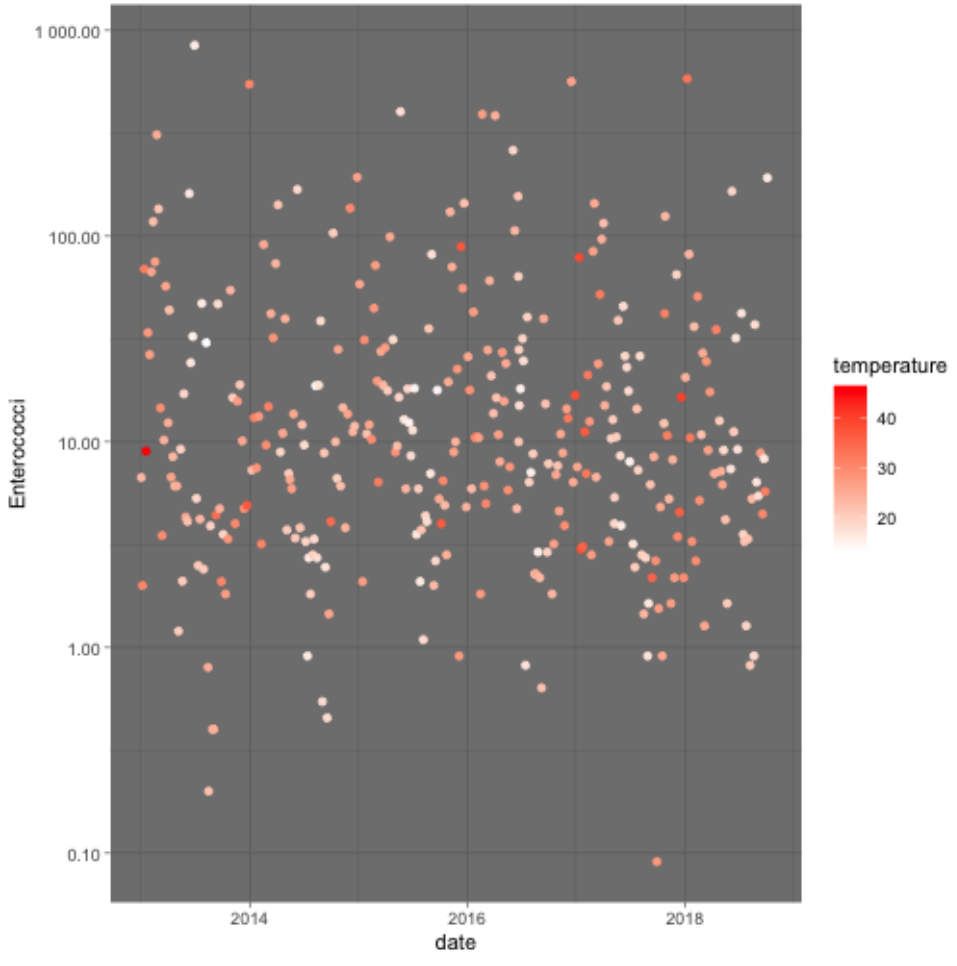
- `scale_*_brewer` uses the ColorBrewer palettes
- `scale_*_gradient` uses gradients
- `scale_*_distill` uses the ColorBrewer palettes, for continuous outcomes

> Here * can be `color` or `fill`, depending on what you want to color
>
> Note `color` refers to the outline, and `fill` refers to the inside

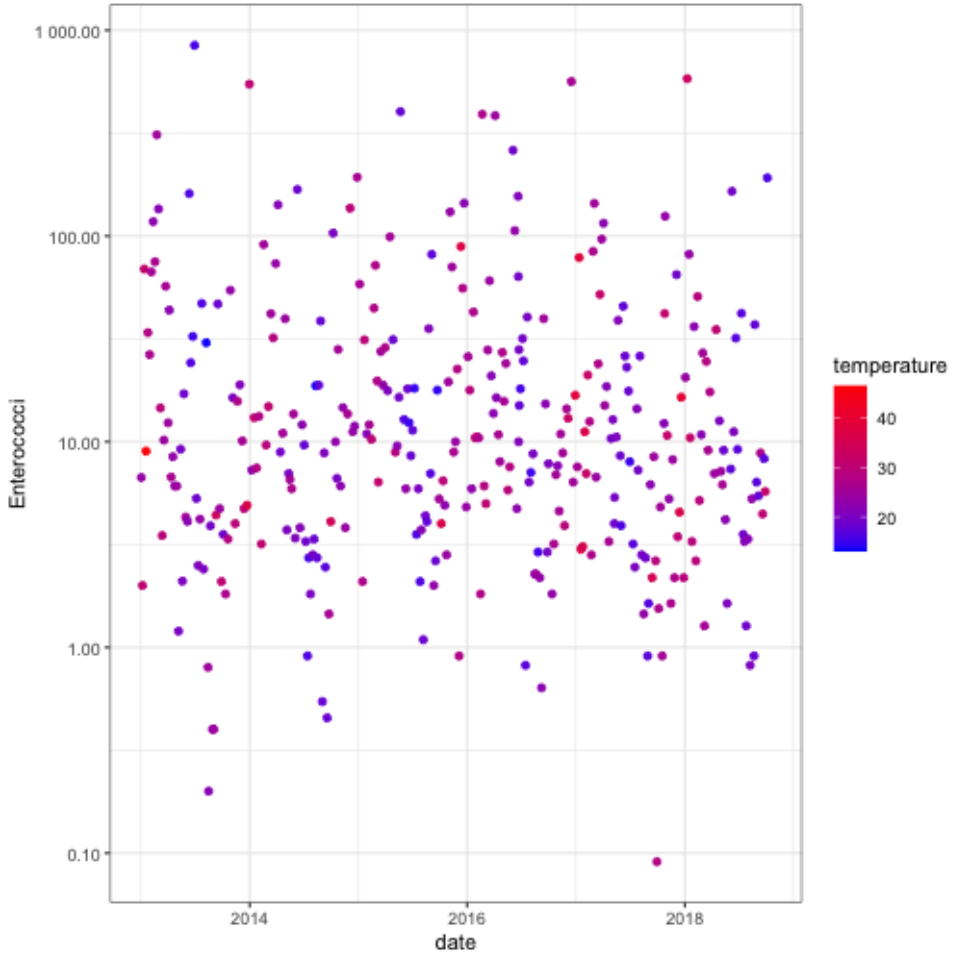
```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci,
      color = temperature)) +
  geom_point() +
  scale_y_log10(name = 'Enterococci',
                label = scales::number_format(digits=
```



```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci,
      color = temperature)) +
  geom_point() +
  scale_y_log10(name = 'Enterococci',
                label = scales::number_format(digits=
  scale_color_gradient(low = 'white', high='red') +
  theme_dark()
```

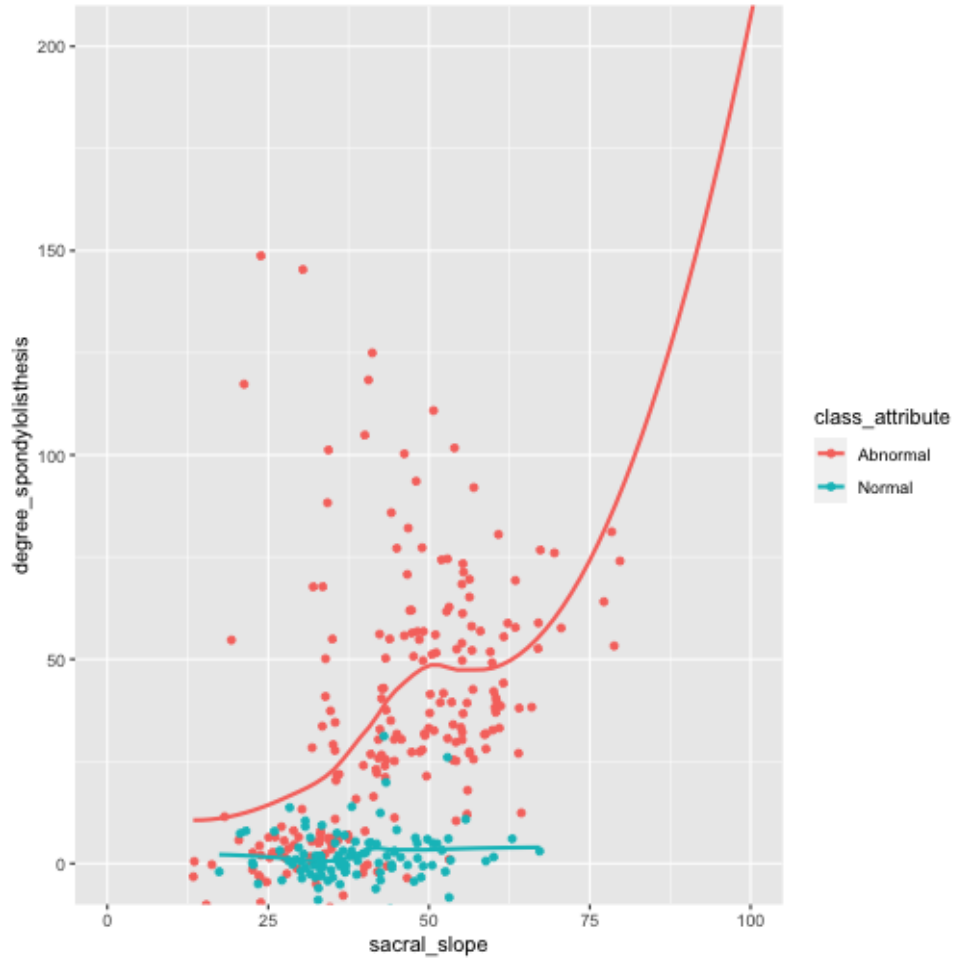


```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci,
      color = temperature)) +
  geom_point() +
  scale_y_log10(name = 'Enterococci',
                label = scales::number_format(digits=
  scale_color_gradient(low = 'blue', high='red') +
  theme_bw()
```



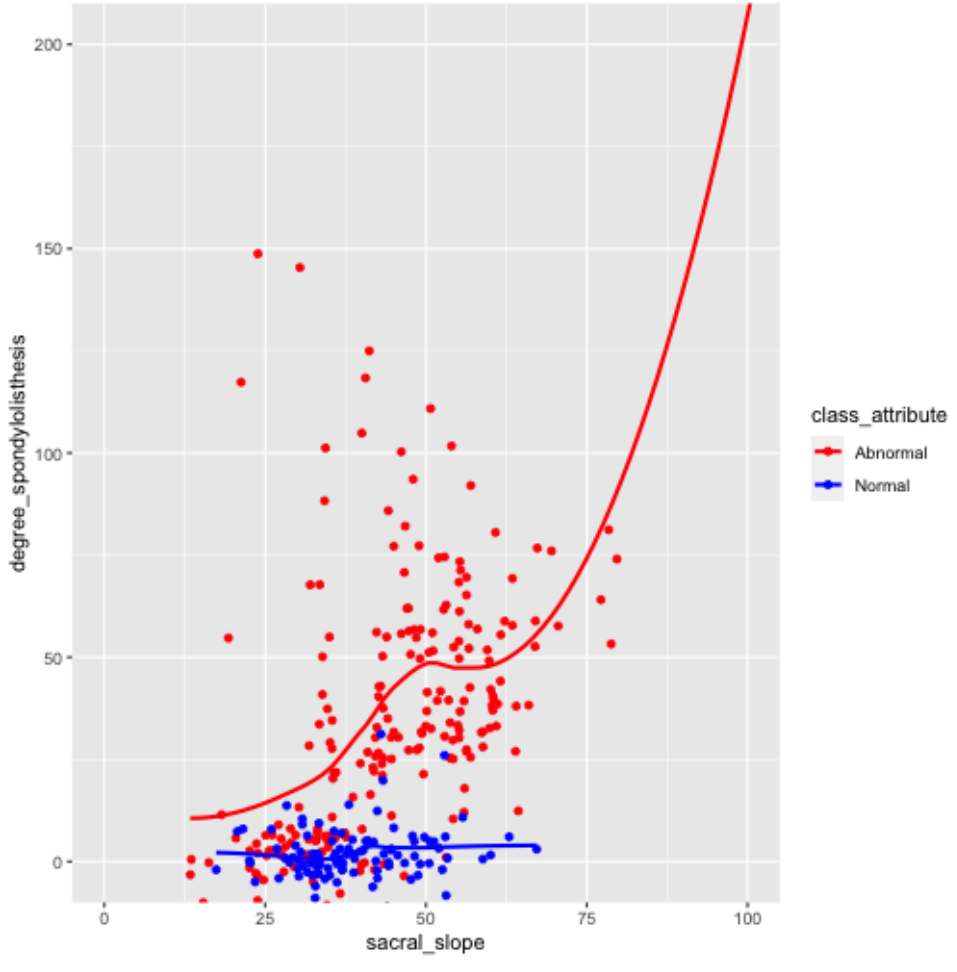
Specifying colors

```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope, y = degree_spondylolisthesis,
      color = class_attribute)) +
  geom_point() +
  geom_smooth(se = F) +
  coord_cartesian(xlim = c(0, 100), ylim = c(0,200))
```



Specifying colors

```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope, y = degree_spondylolisthesis,
      color = class_attribute)) +
  geom_point() +
  geom_smooth(se = F) +
  coord_cartesian(xlim = c(0, 100), ylim = c(0,200))
  scale_color_manual(values = c("Normal"="blue", 'Abn
```



Themes

You can create your own custom themes to keep a unified look to your graphs

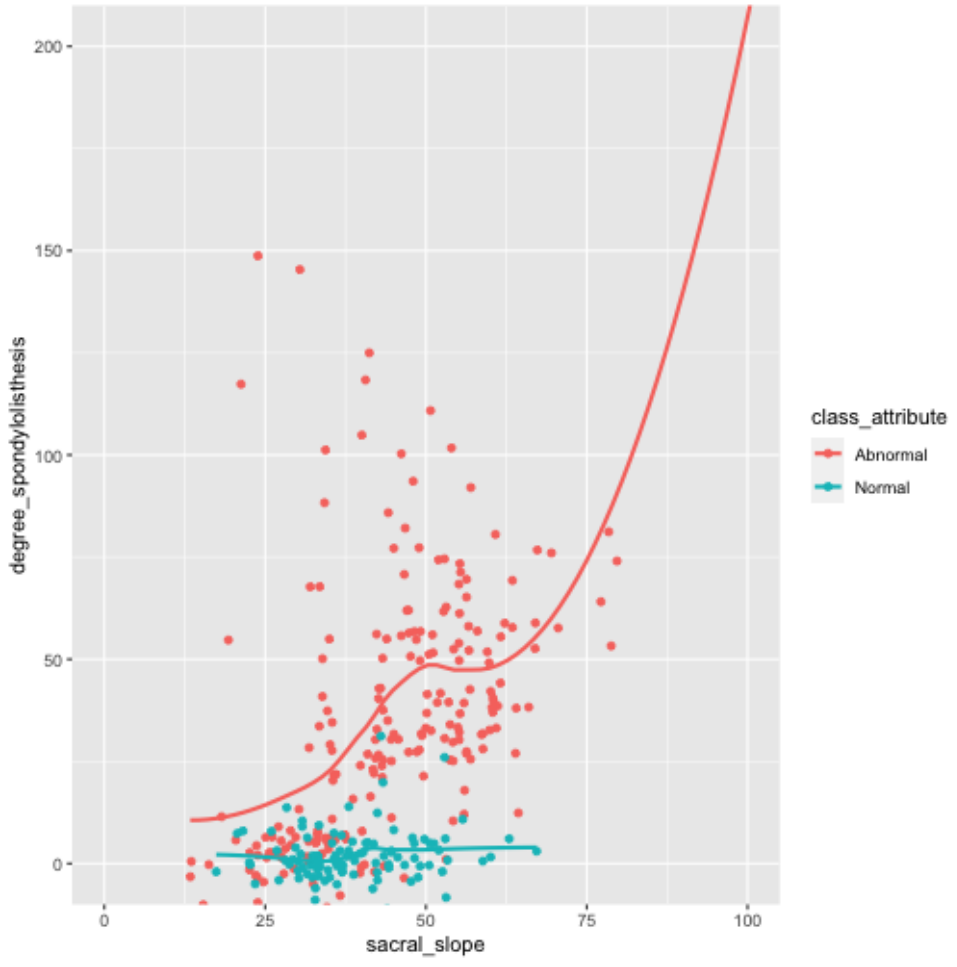
ggplot comes with

- theme_classic
- theme_bw
- theme_void
- theme_dark
- theme_gray
- theme_light
- theme_minimal

Themes

Create your own

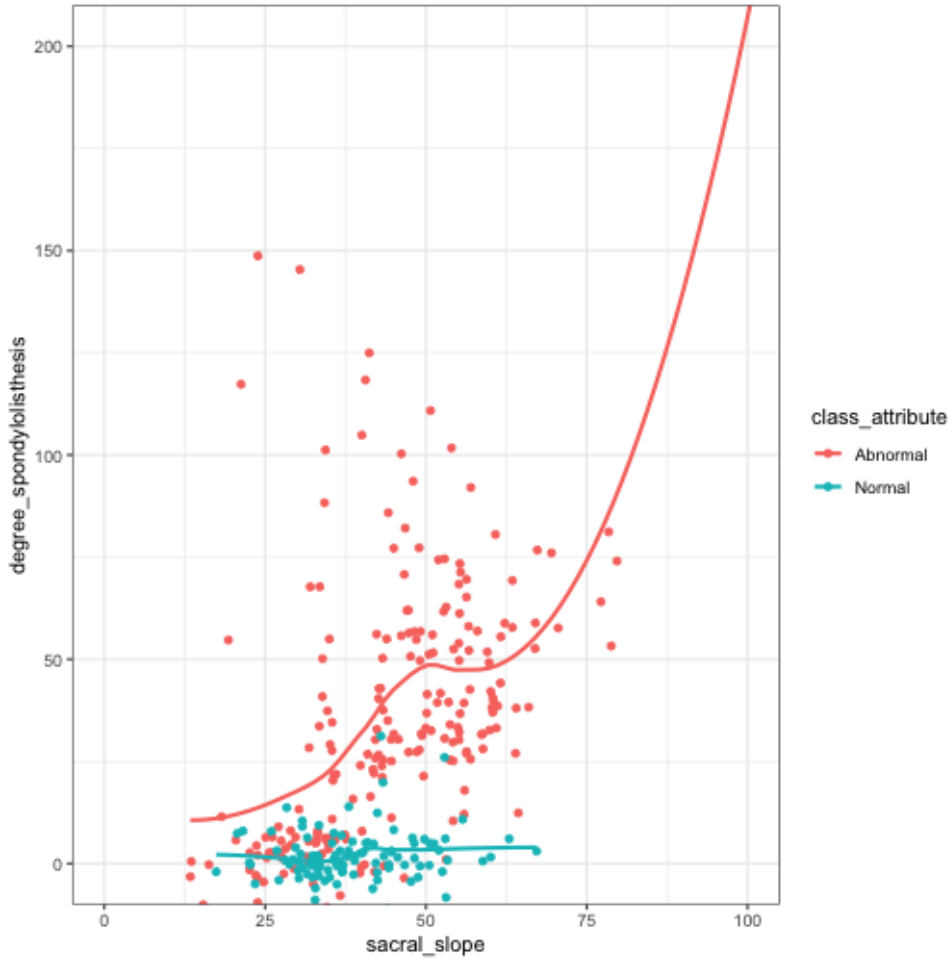
```
ggplot(
  data = dat_spine,
  aes(x = sacral_slope, y = degree_spondylolisthesis,
      color = class_attribute)) +
  geom_point() +
  geom_smooth(se = F) +
  coord_cartesian(xlim = c(0, 100),
                  ylim = c(0,200))
```



Themes

Create your own

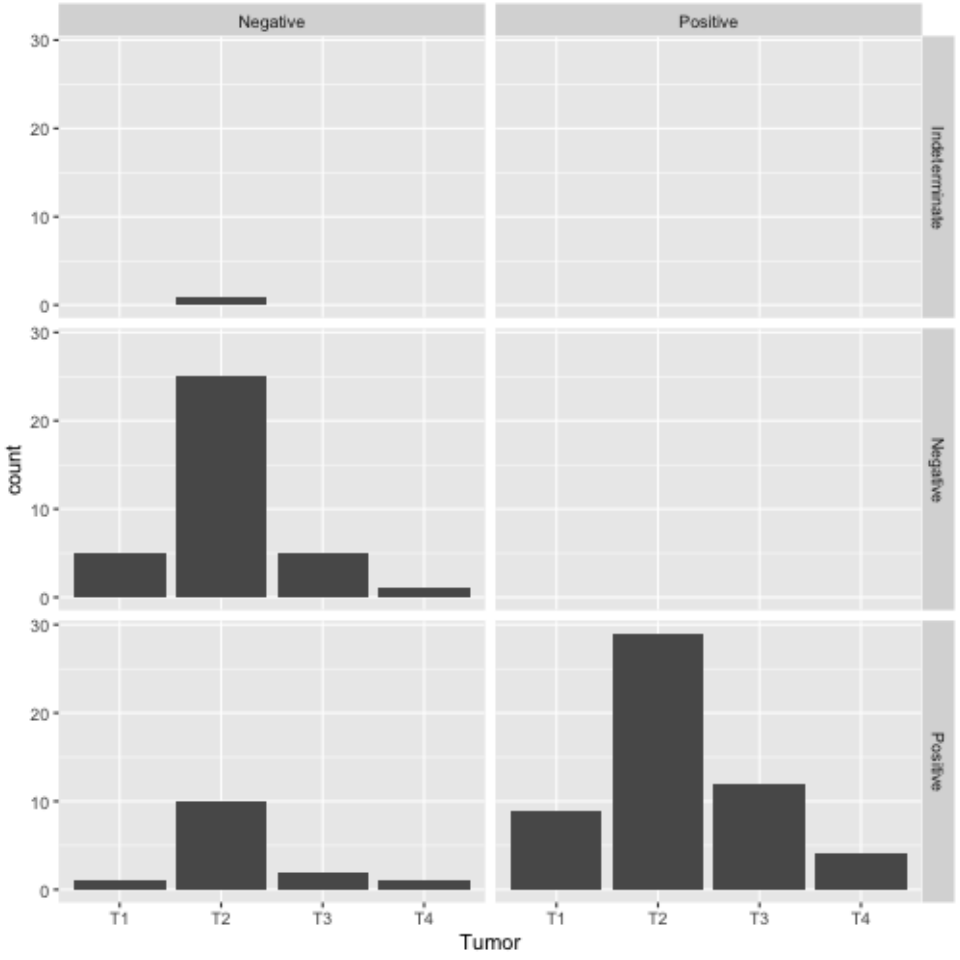
```
my_theme <- function(){
  theme_bw()
}

ggplot(
  data = dat_spine,
  aes(x = sacral_slope, y = degree_spondylolisthesis,
      color = class_attribute)) +
  geom_point() +
  geom_smooth(se = F) +
  coord_cartesian(xlim = c(0, 100),
                  ylim = c(0,200)) +
  my_theme()
```



Themes

Create your own

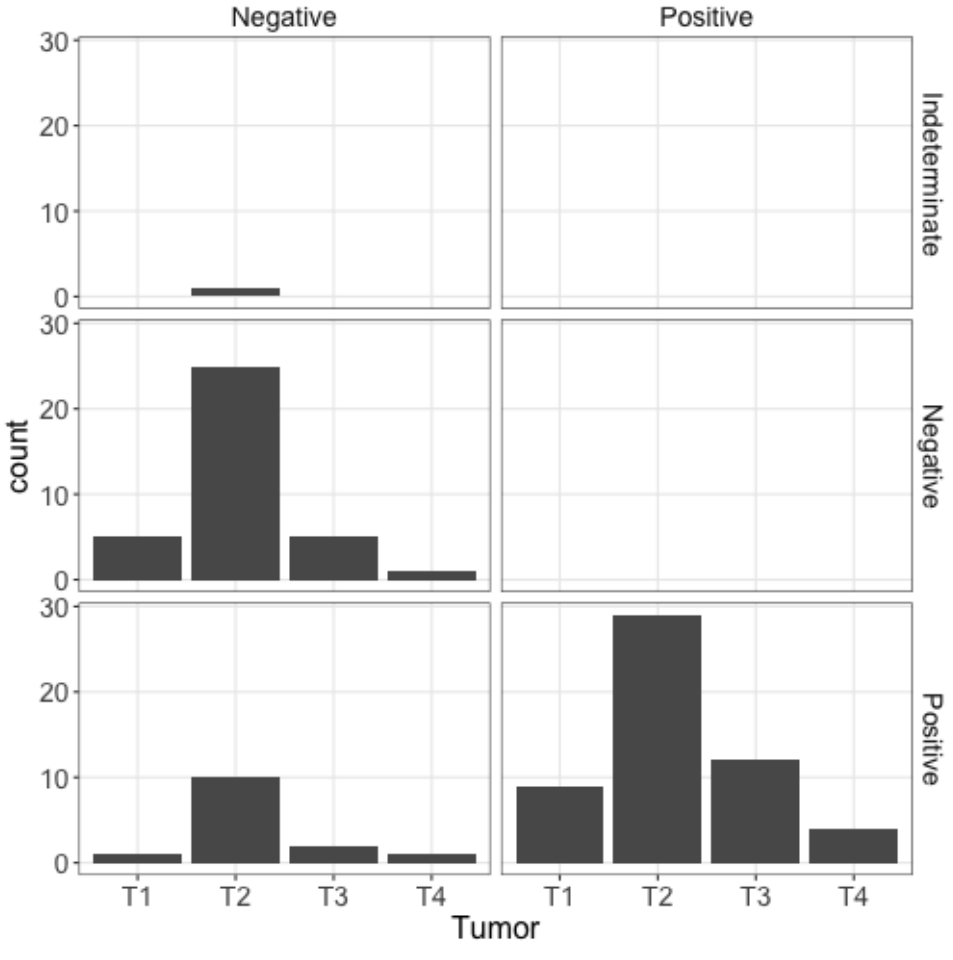
```
my_theme <- function(){
  theme_bw() +
  theme(axis.text = element_text(size = 14),
        axis.title = element_text(size = 16),
        panel.grid.minor = element_blank(),
        strip.text = element_text(size=14),
        strip.background = element_blank())
}

ggplot(
  data = dat_brca,
  aes(x = Tumor))+
  geom_bar() +
  facet_grid(rows = vars(ER.Status),
             cols = vars(PR.Status))
```



Themes

Create your own

```
my_theme <- function(){
  theme_bw() +
  theme(axis.text = element_text(size = 14),
        axis.title = element_text(size = 16),
        panel.grid.minor = element_blank(),
        strip.text = element_text(size=14),
        strip.background = element_blank())
}

ggplot(
  data = dat_brca,
  aes(x = Tumor))+
  geom_bar() +
  facet_grid(rows = vars(ER.Status),
             cols = vars(PR.Status)) +
  my_theme()
```



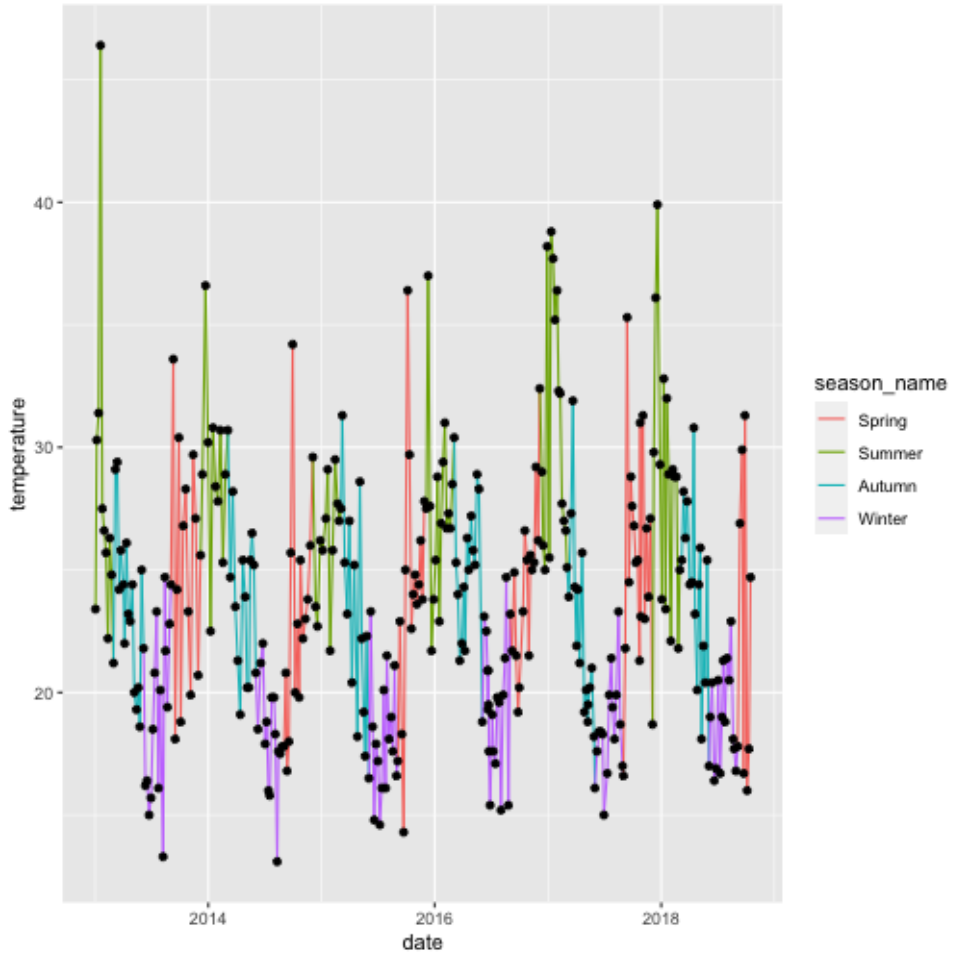
Animations

gganimate

The new `gganimate` package has made it very easy to create animations

It's literally a few lines

```
library(gganimate)
plt <- ggplot(beaches,
              aes(date, temperature))+
  geom_path(aes(color = season_name, group = 1))+
  geom_point()

plt
```



```
library(gganimate)
plt <- ggplot(beaches,
              aes(date, temperature))+
  geom_path(aes(color = season_name, group = 1))+
  geom_point()

plt + transition_reveal(date)
```