Starting DataViz using R: ggplot2

Abhijit Dasgupta, PhD

Spring 2019

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 enterococci
#>     <date>     <dbl> <dbl> <dbl>  <dbl>    <dbl>       <dbl>       <dbl>
#>   1 2013-01-02  2013     1     2      1      0          23.4         6.7
#>   2 2013-01-06  2013     1     6      1      0          30.3         2
#>   3 2013-01-12  2013     1    12      1      0          31.4        69.1
#>   4 2013-01-18  2013     1    18      1      0          46.4         9
#>   5 2013-01-24  2013     1    24      1      0          27.5        33.9
#>   6 2013-01-30  2013     1    30      1      0.6        26.6        26.5
#>   7 2013-02-05  2013     2     5      1      0.1        25.7        66.9
#>   8 2013-02-11  2013     2    11      1      8          22.2       118.
#>   9 2013-02-17  2013     2    17      1     13.6        26.3        75
#>  10 2013-02-23  2013     2    23      1      7.2        24.8       311.
#>  # … with 334 more rows, and 4 more variables: day_num <dbl>,
#>  #   month_num <dbl>, month_name <chr>, season_name <chr>
```

Credit: D. J. Navarro

Building a graph

Start with a blank canvas

```
ggplot()
```

Add a data set

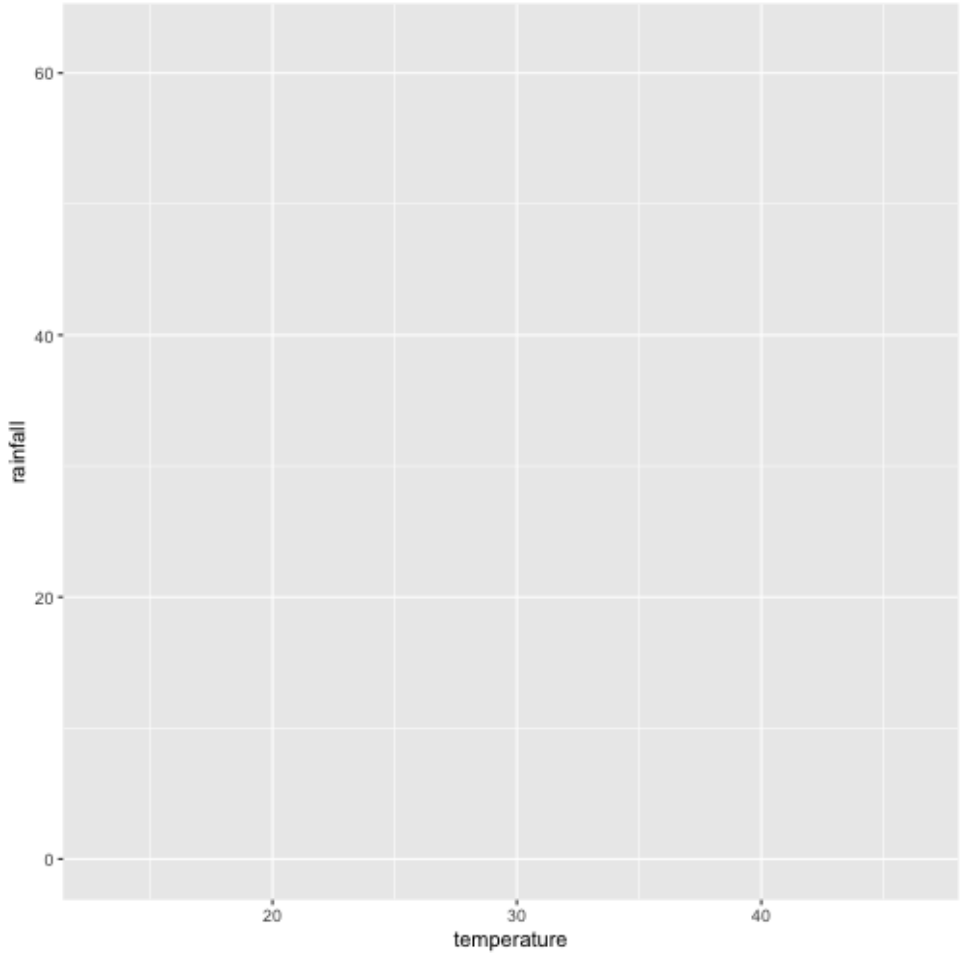
```
ggplot(
  data = beaches
)
```

Add a mapping from data to elements

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

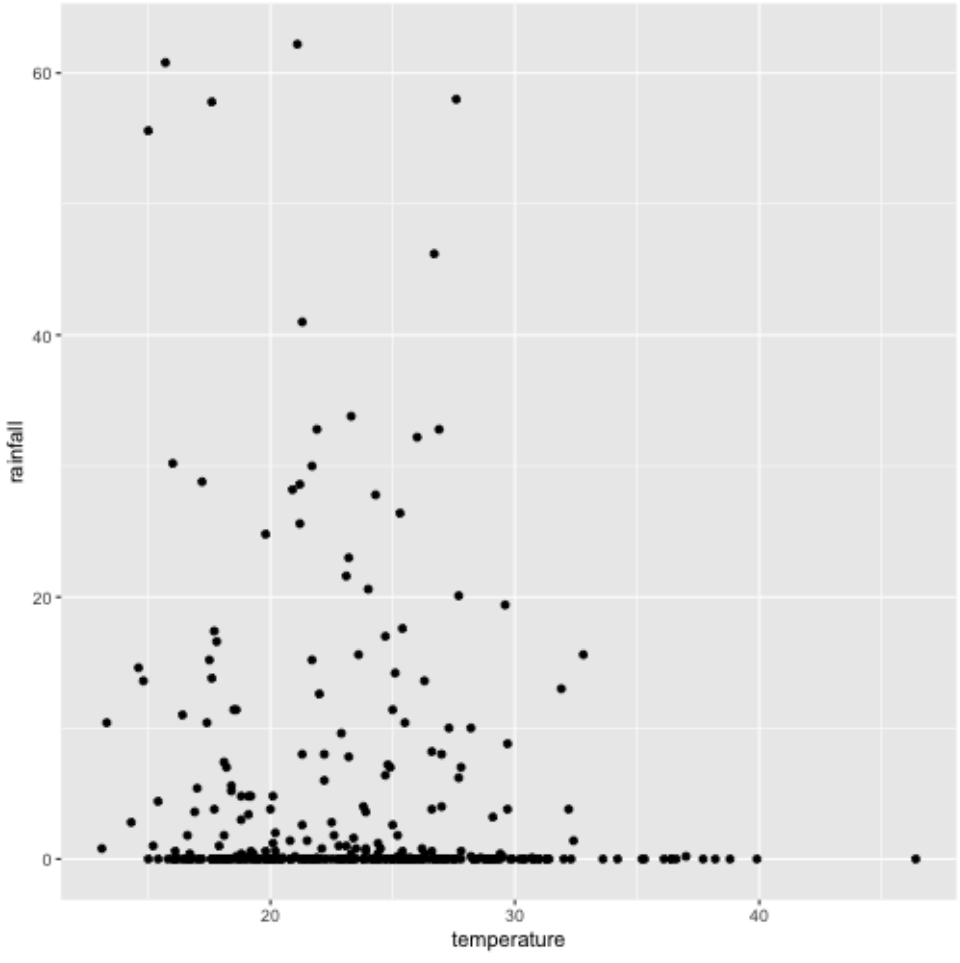
What goes in

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



Add a geometry to draw

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

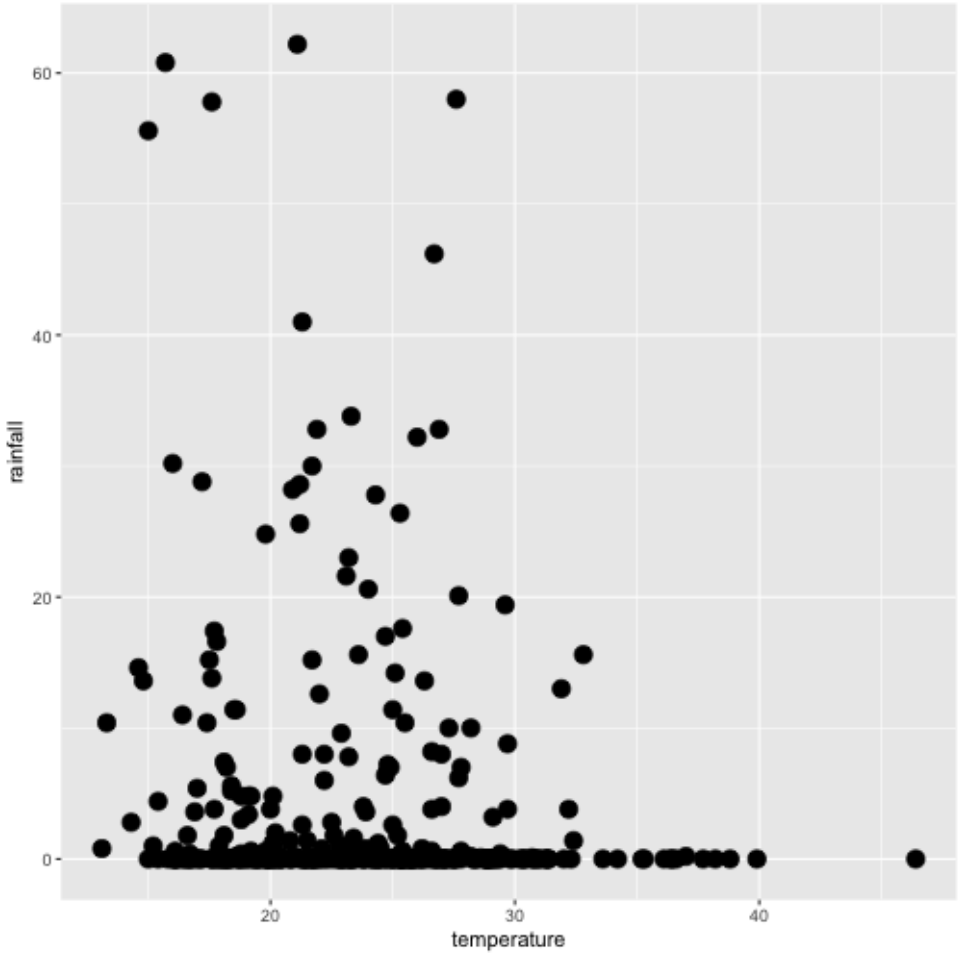


What to draw:

- Points, lines
- histogram, bars, pies

Add options for the geom

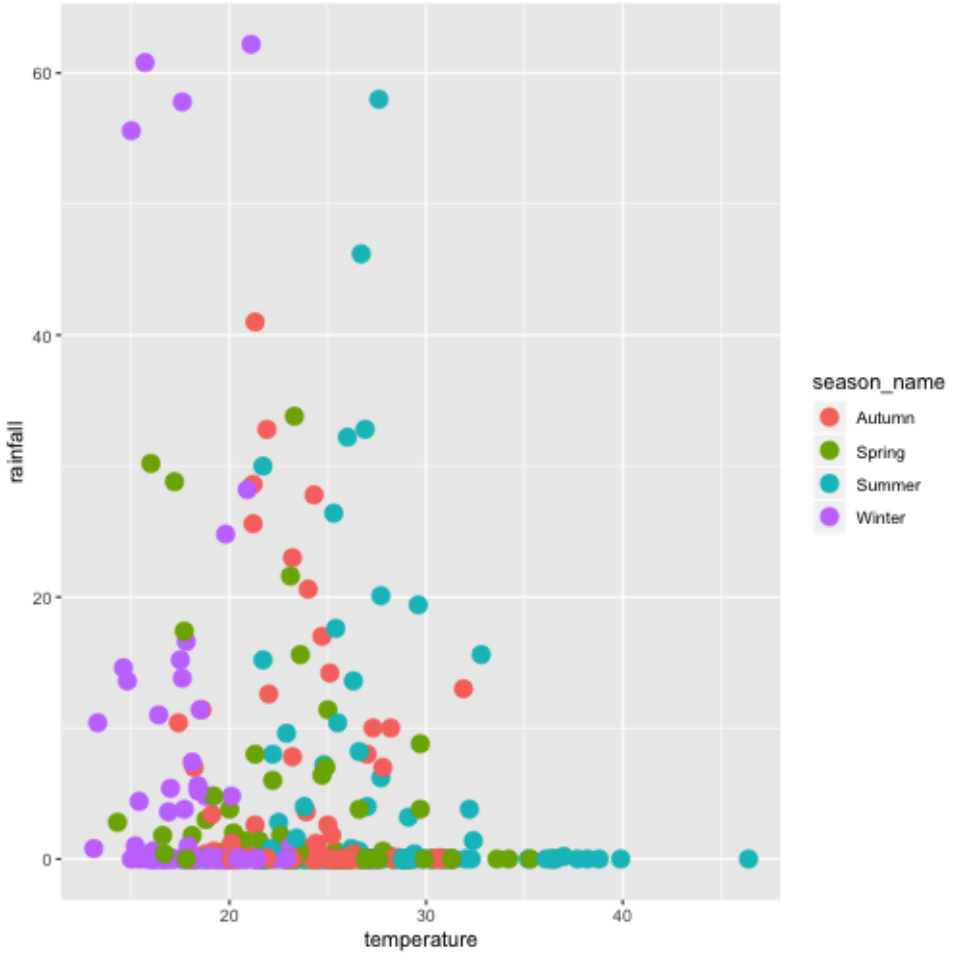
```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(size = 4)
```



Add a mapping to modify the geom

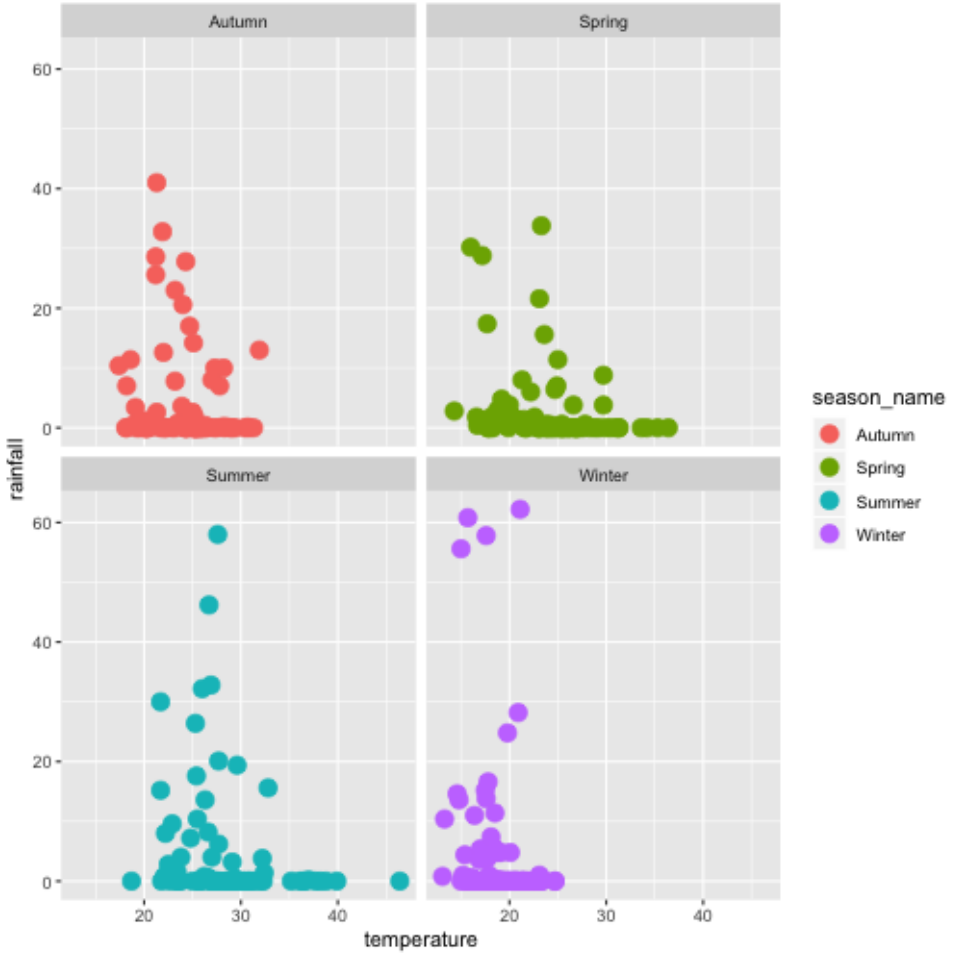
```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4
  )
```

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



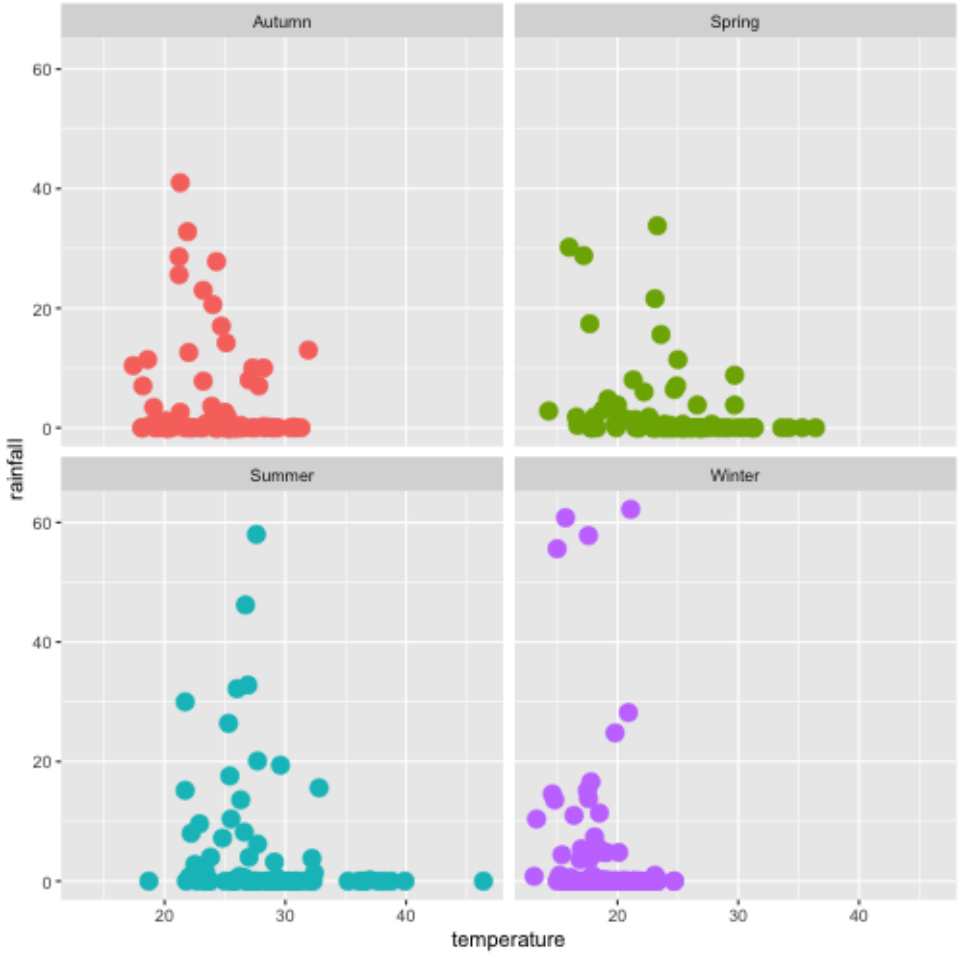
Split into facets

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



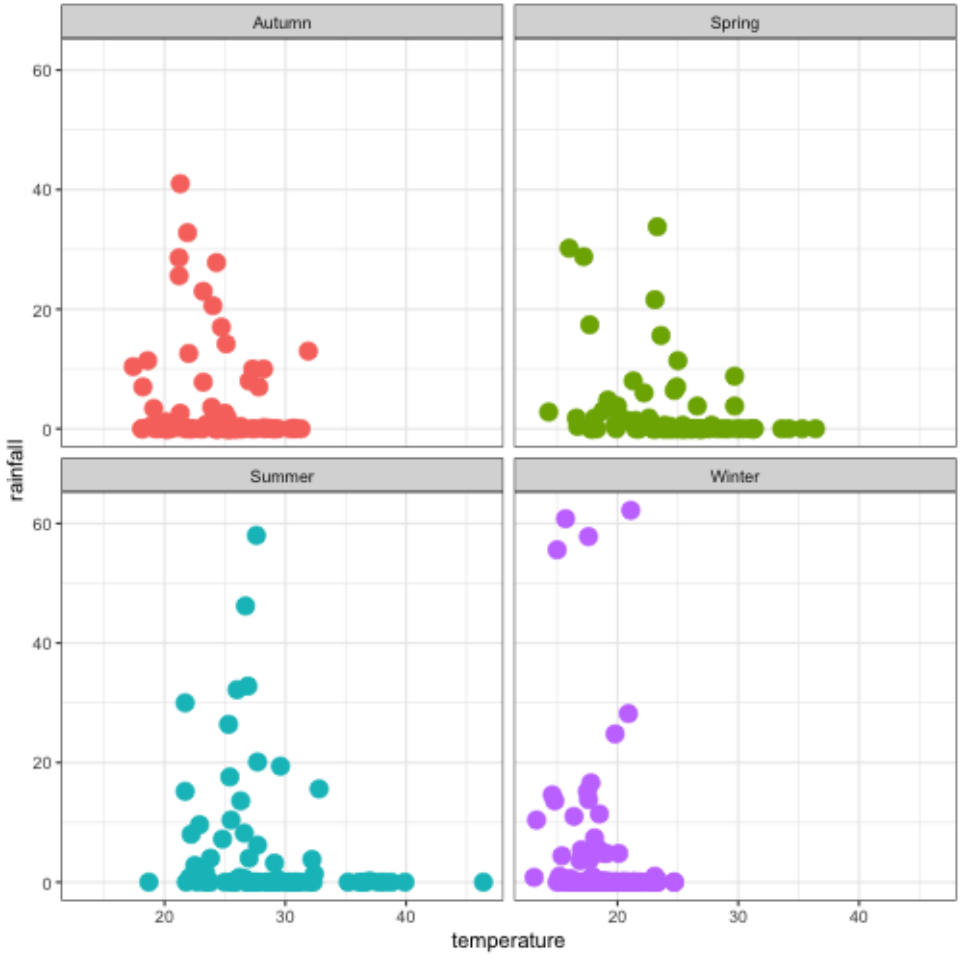
Remove the legend

```
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)
```



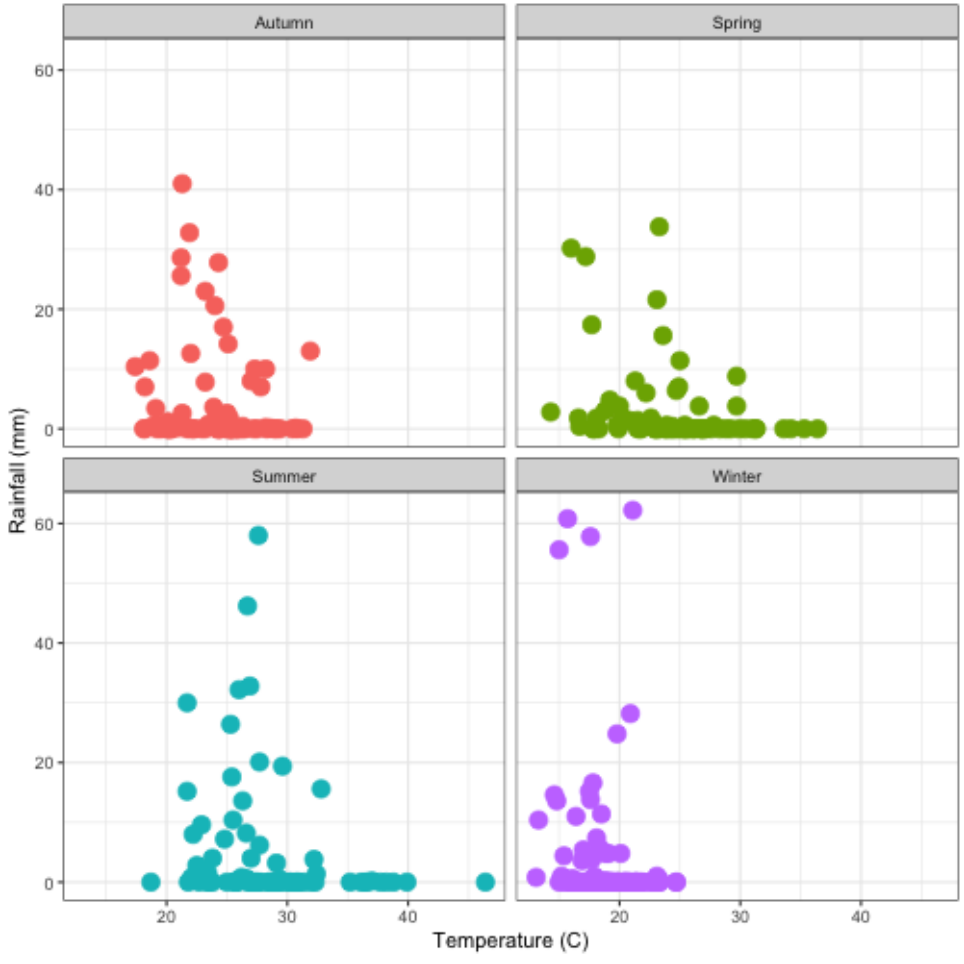
Change the background

```
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()
```



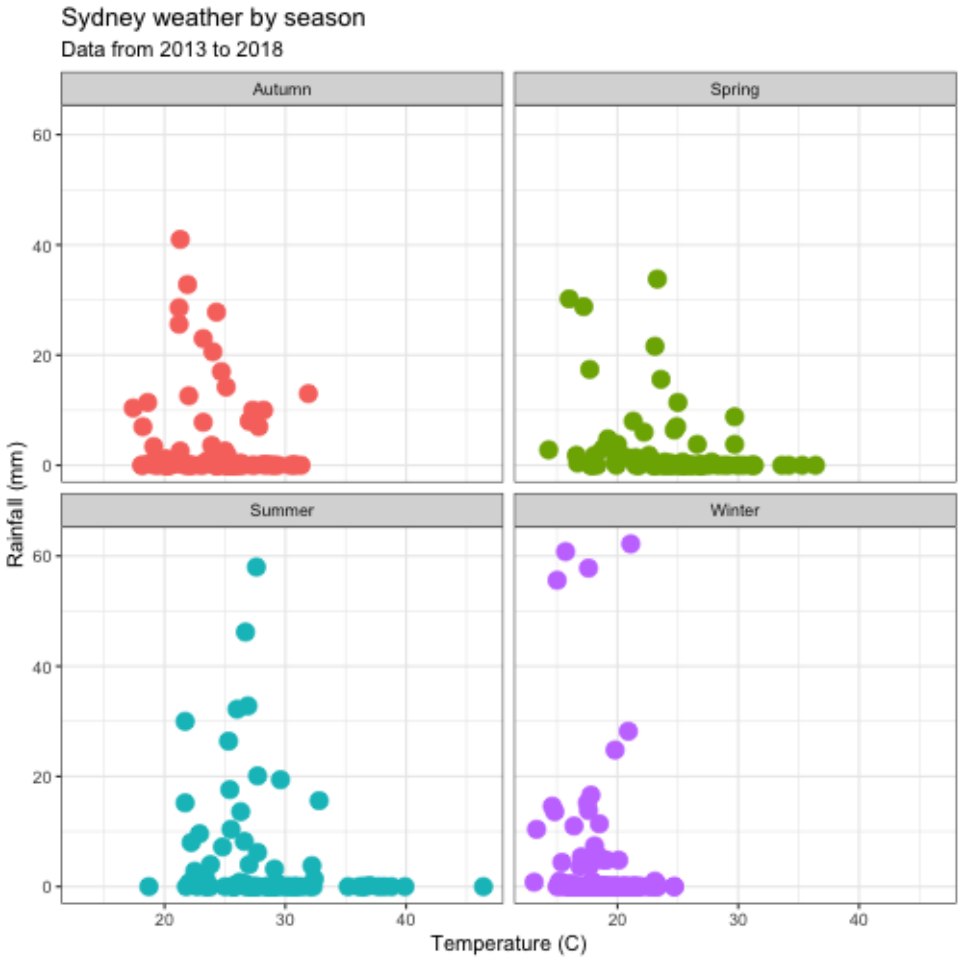
Update the labels

```
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)')
```



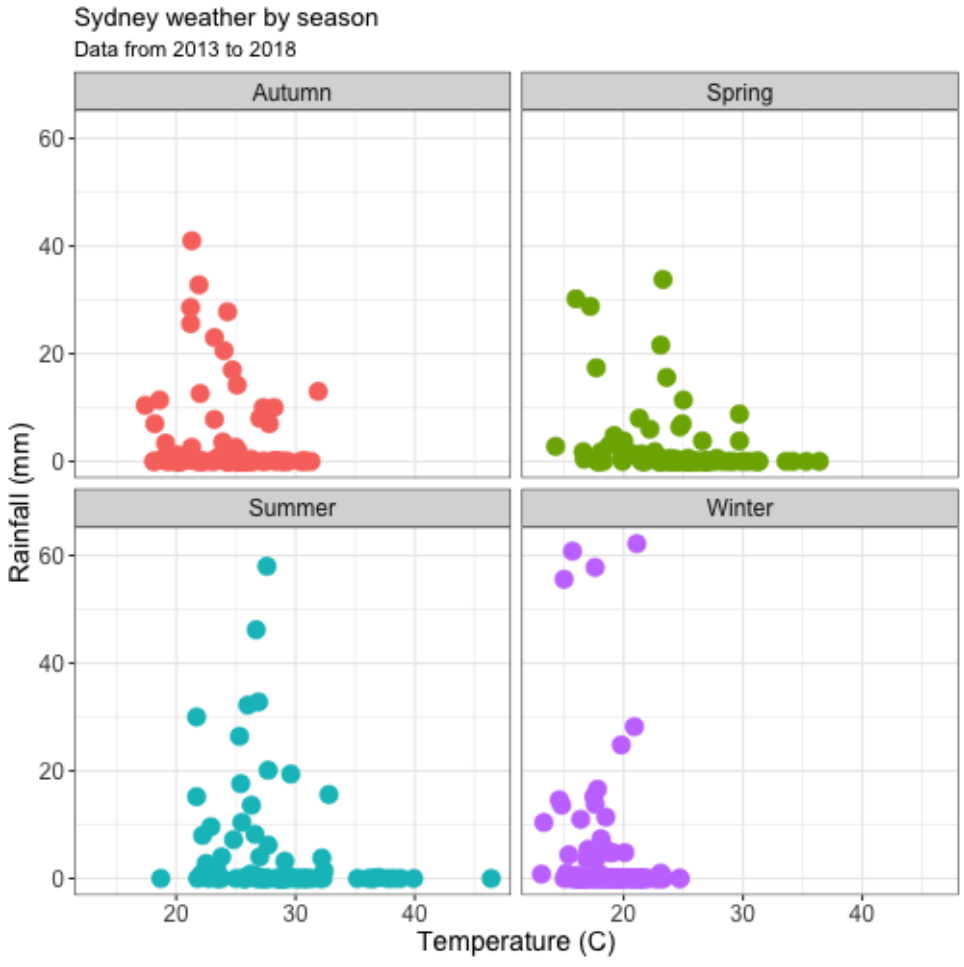
Add titles

```
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))
```



The grammar

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

Exercise 1

Peeking under the hood

If I write...

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

what's really run is ...

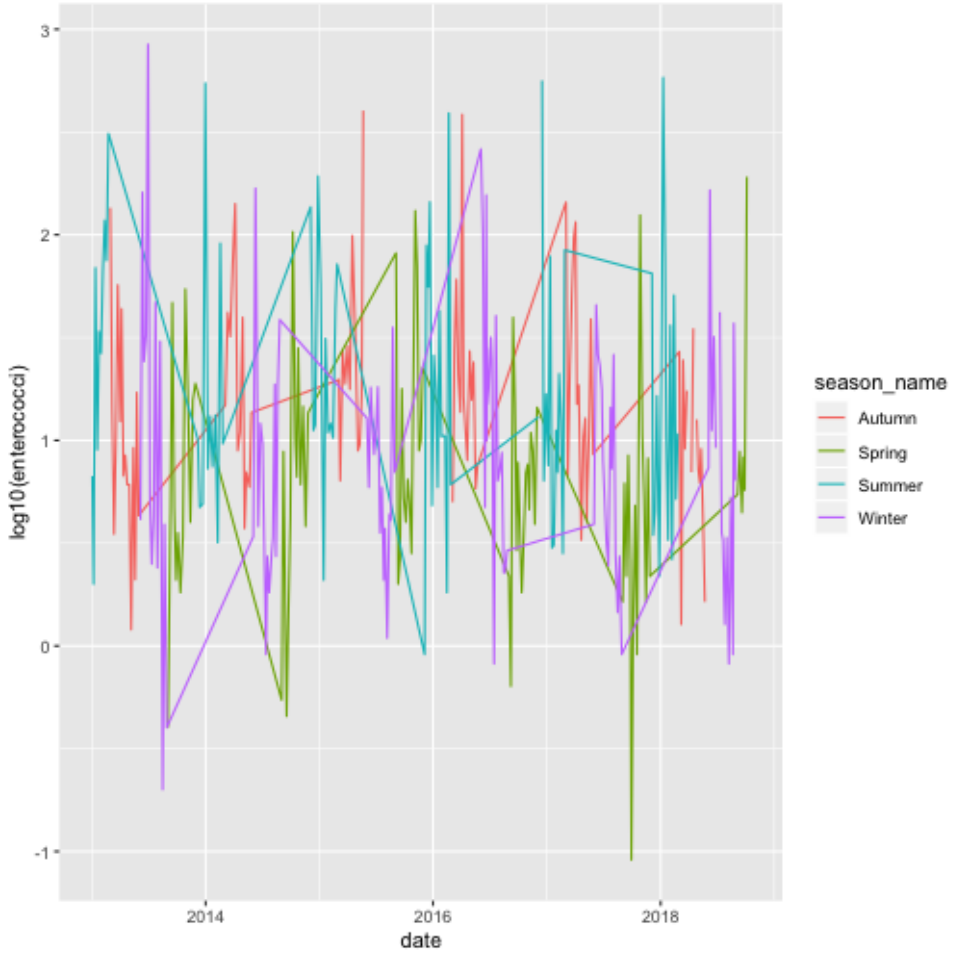
```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature, y = rainfall)) +
layer(
  geom = "point",
  stat = "identity",
  position = "identity") +
facet_null() +
theme_grey() +
coord_cartesian() +
scale_x_continuous() +
scale_y_continuous()
```

Each element can be adapted and tweaked to create graphs

Exploring aesthetics

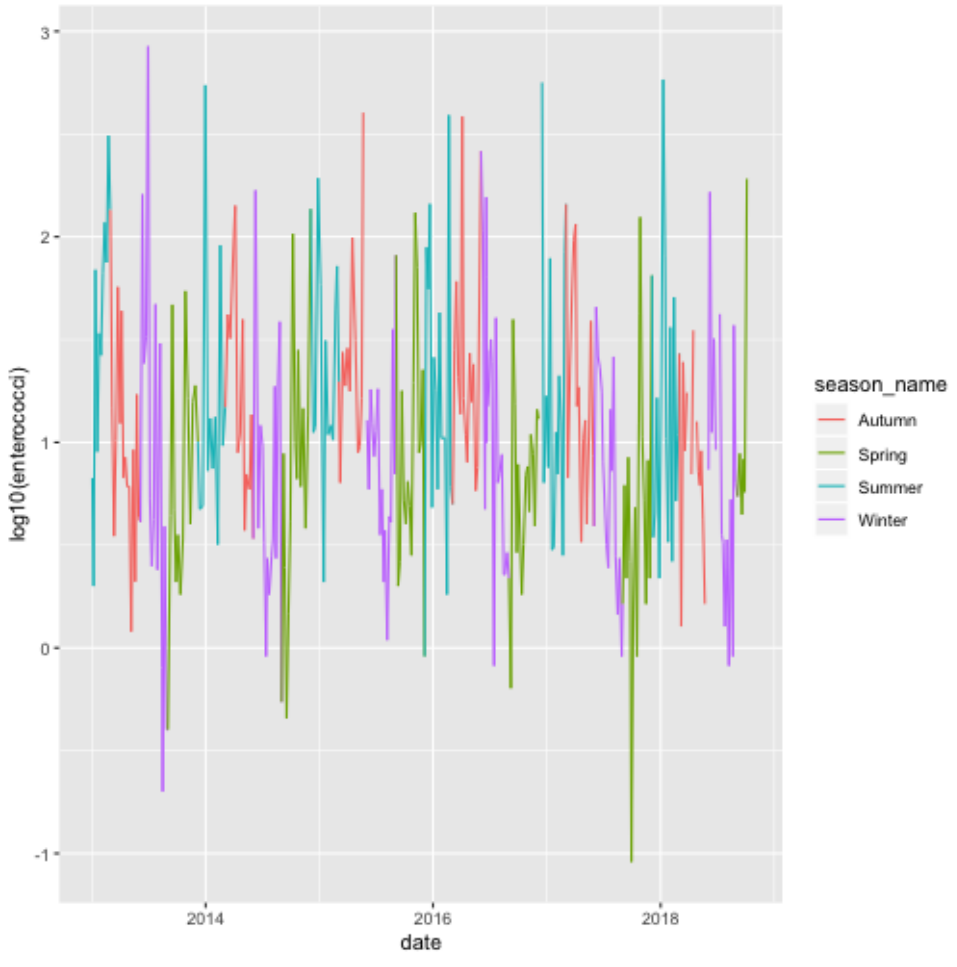
Mapping color

```
ggplot(
  data=beaches,
  aes(x = date,
      y = log10(enterococci),
      color = season_name)
) +
  geom_line()
```



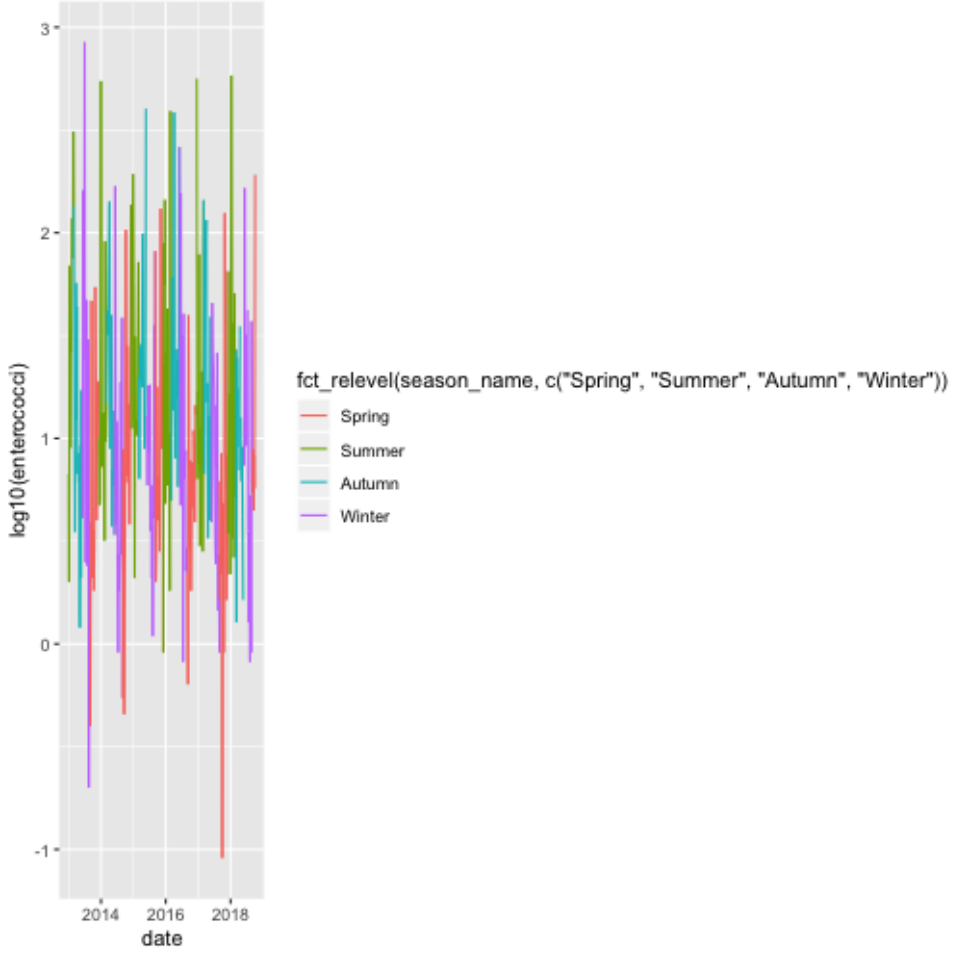
Adding groups to the mapping

```
ggplot(
  data=beaches,
  aes(x = date,
      y = log10(enterococci),
      color = season_name,
      group = 1)
) +
  geom_line()
```



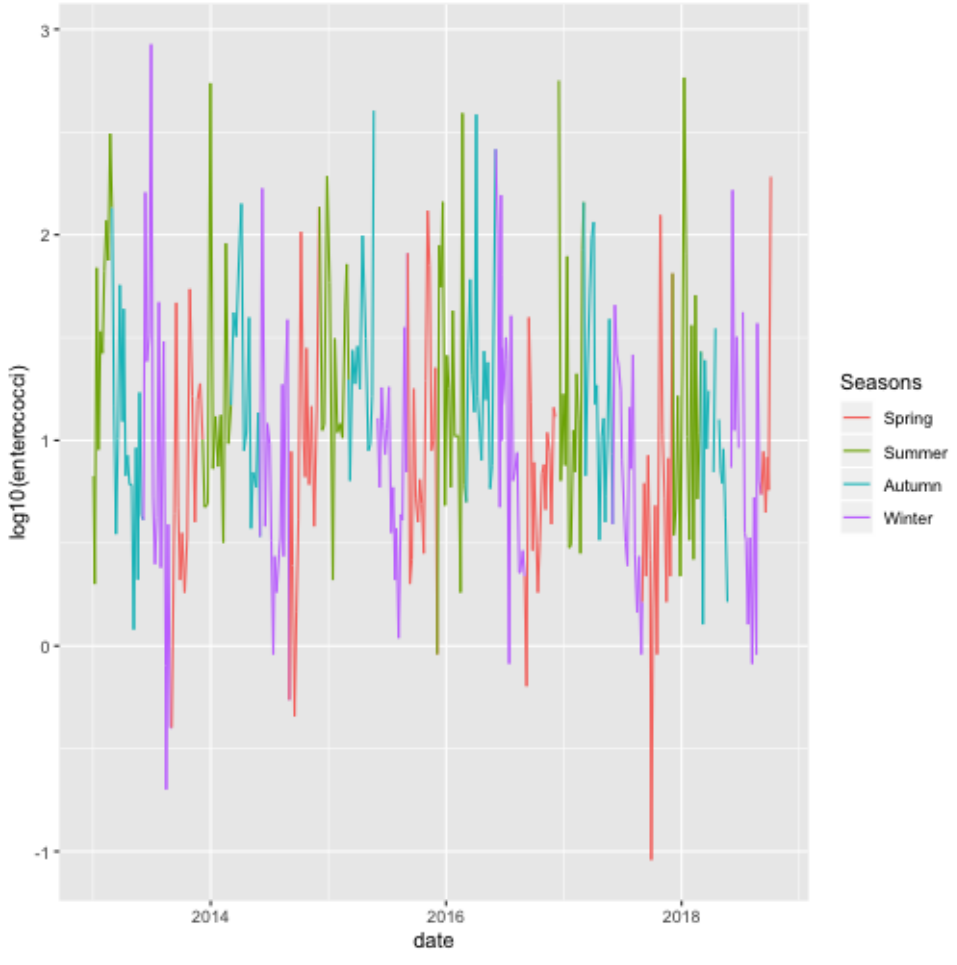
Fixing the legend ordering

```
ggplot(
  data=beaches,
  aes(x = date,
      y = log10(enterococci),
      color = fct_relevel(season_name, c('Spring','Su
      group = 1)
) +
  geom_line()
```



Fixing the legend ordering

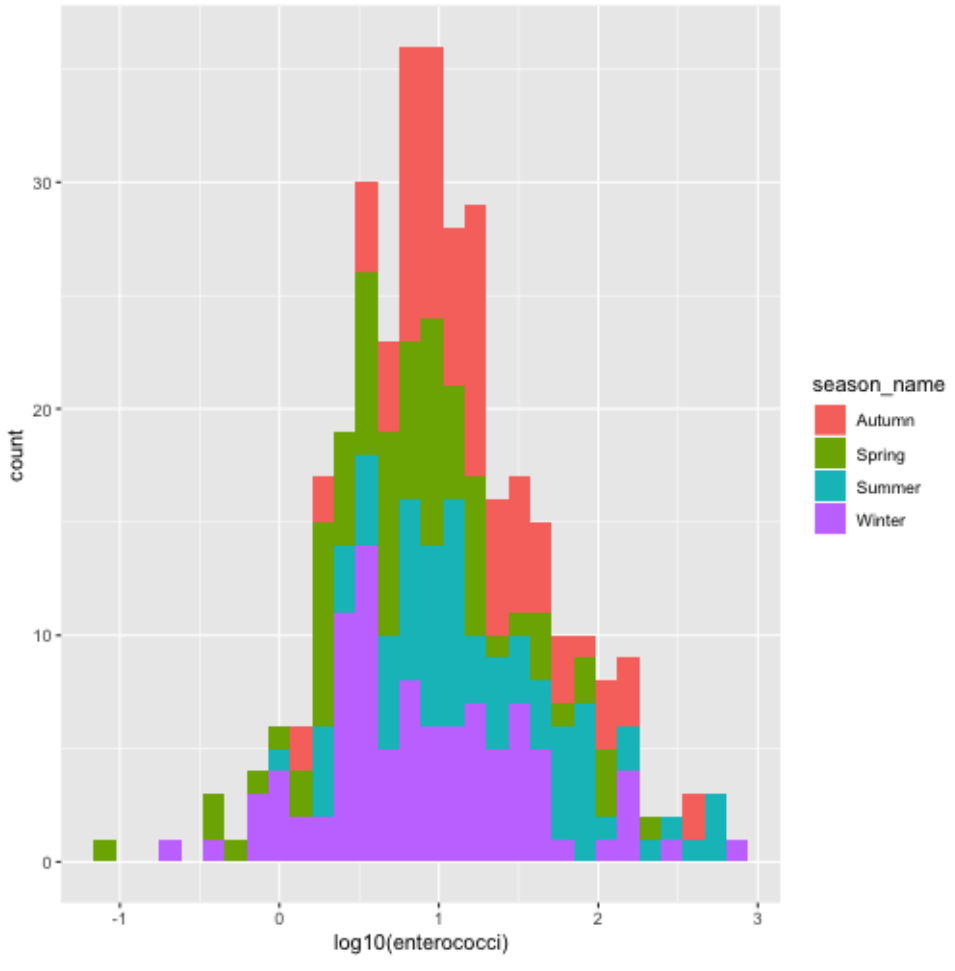
```
ggplot(
  data=beaches,
  aes(x = date,
      y = log10(enterococci),
      color = fct_relevel(season_name, c('Spring','Su
      group = 1)
) +
  geom_line()+
  labs(color = 'Seasons')
```



You can also fill based on data

```
ggplot(
  data=beaches,
  aes(x = log10(enterococci),
      fill = season_name))+
  geom_histogram()
```

This is not a great plot. We'll refine this idea later



Exercise 2

Exploring geometries

Univariate plots

Histograms

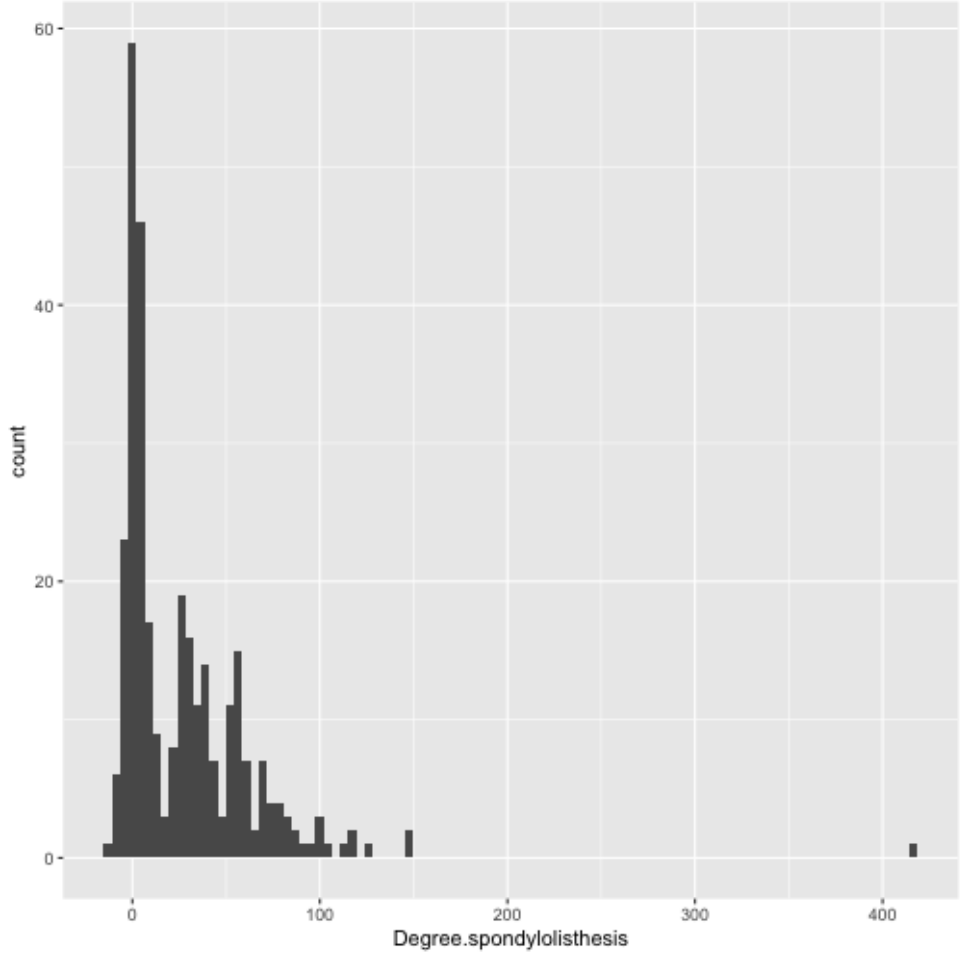
```
library(tidyverse)
library(rio)
dat_spine <- import('data/Dataset_spine.csv',
                    check.names = T)

ggplot(
  data=dat_spine,
  aes(x = Degree.spondylolisthesis))+
  geom_histogram()
```

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

Histograms

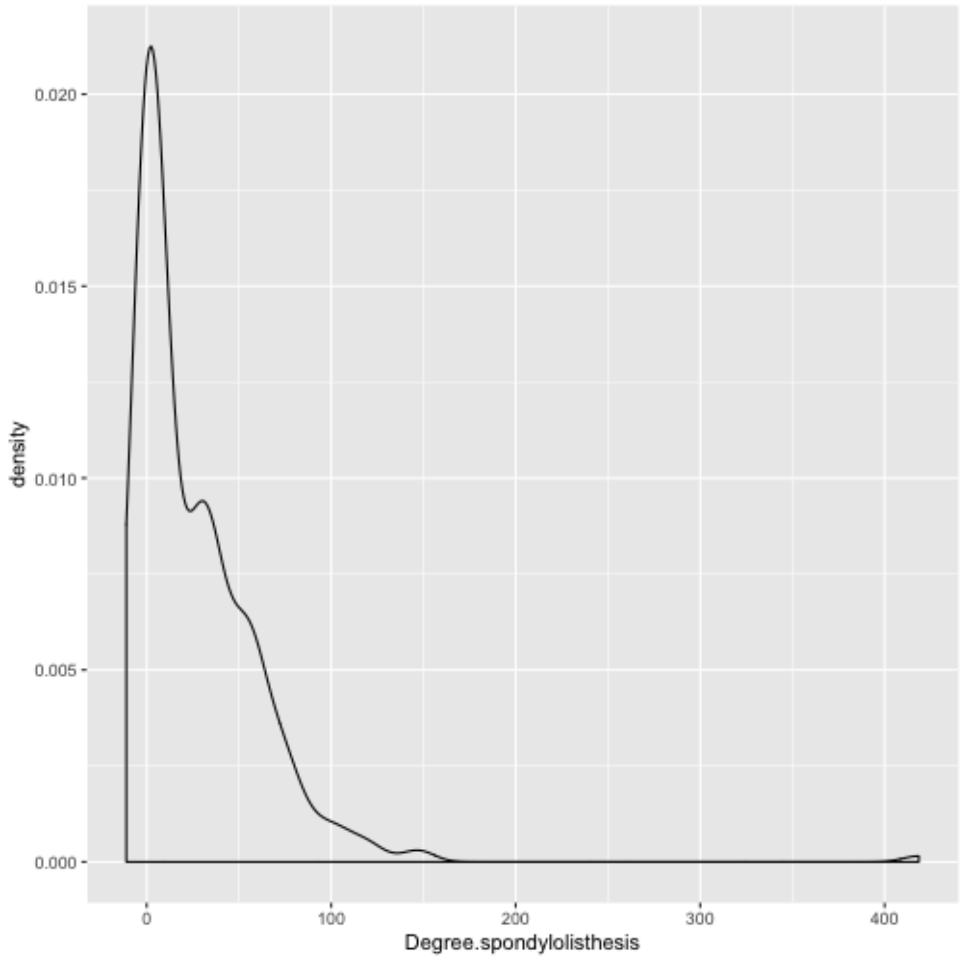
```
ggplot(
  data=dat_spine,
  aes(x = Degree.spondylolisthesis))+
  geom_histogram(bins = 100)
```

This gives a very different view of the data



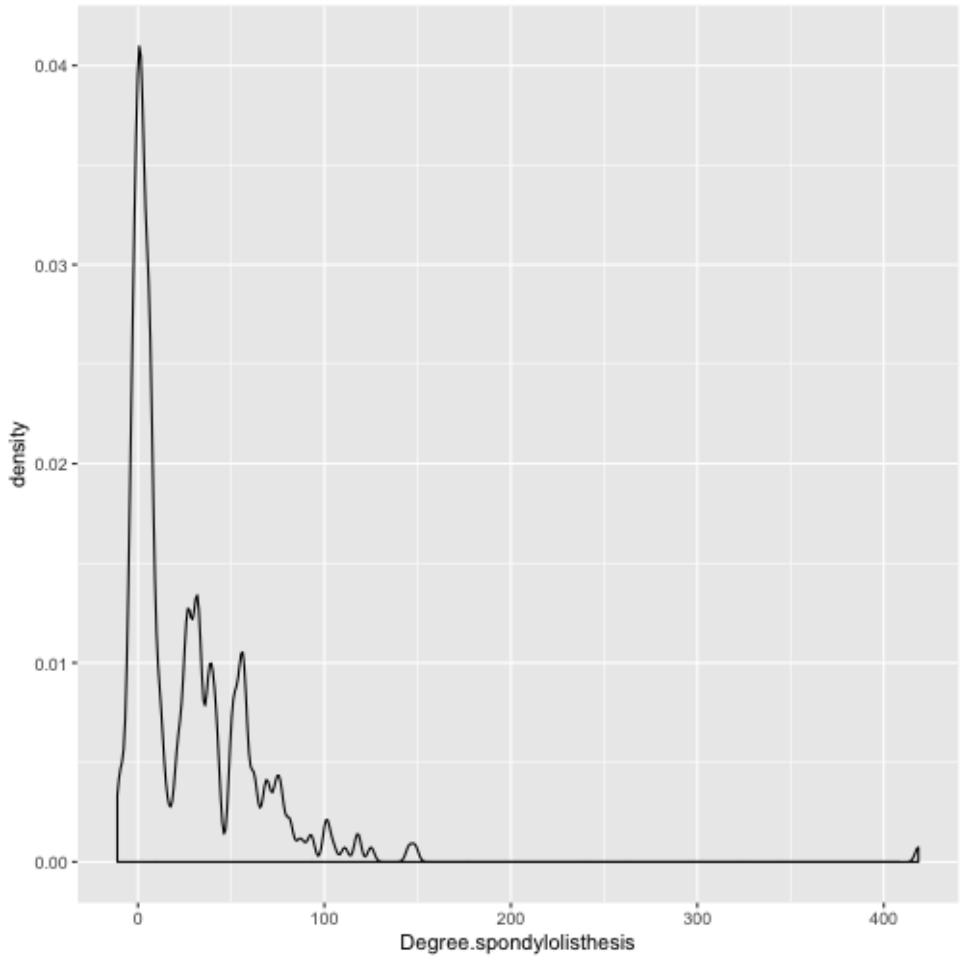
Density plots

```
ggplot(
  data=dat_spine,
  aes(x = Degree.spondylolisthesis))+
  geom_density()
```



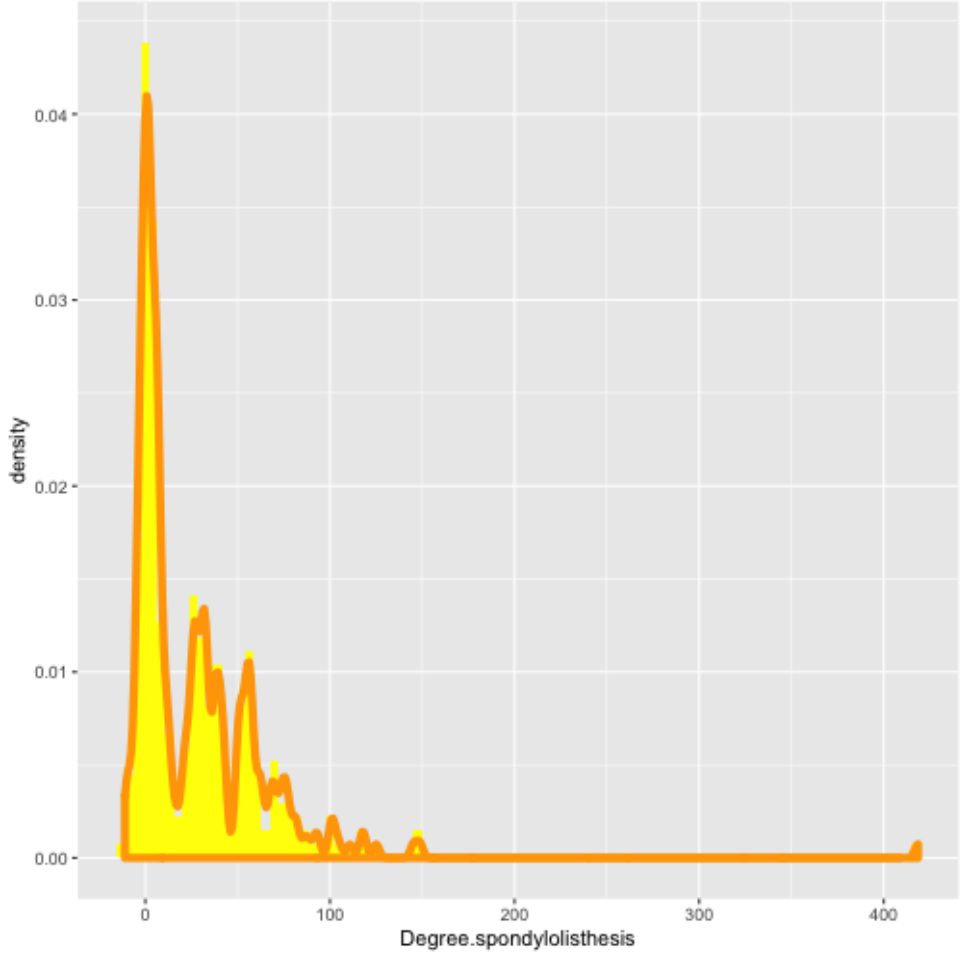
Density plots

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



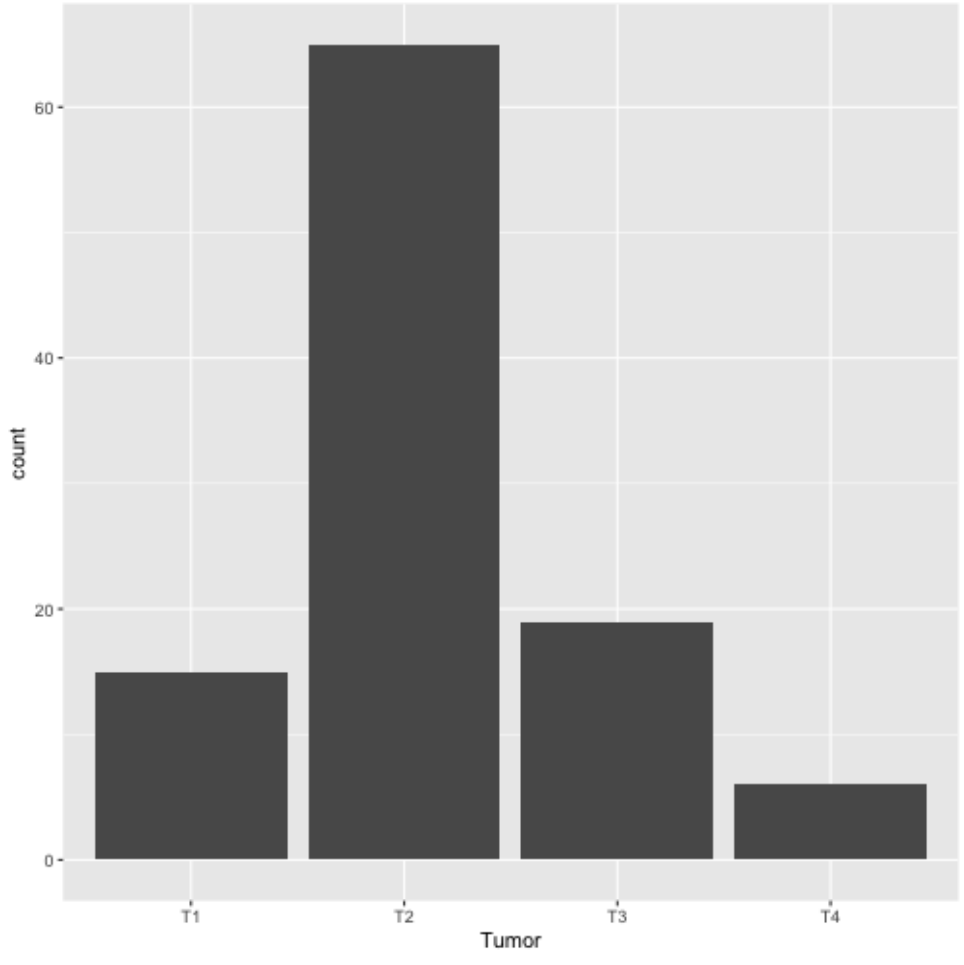
Layering geometries

```
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)

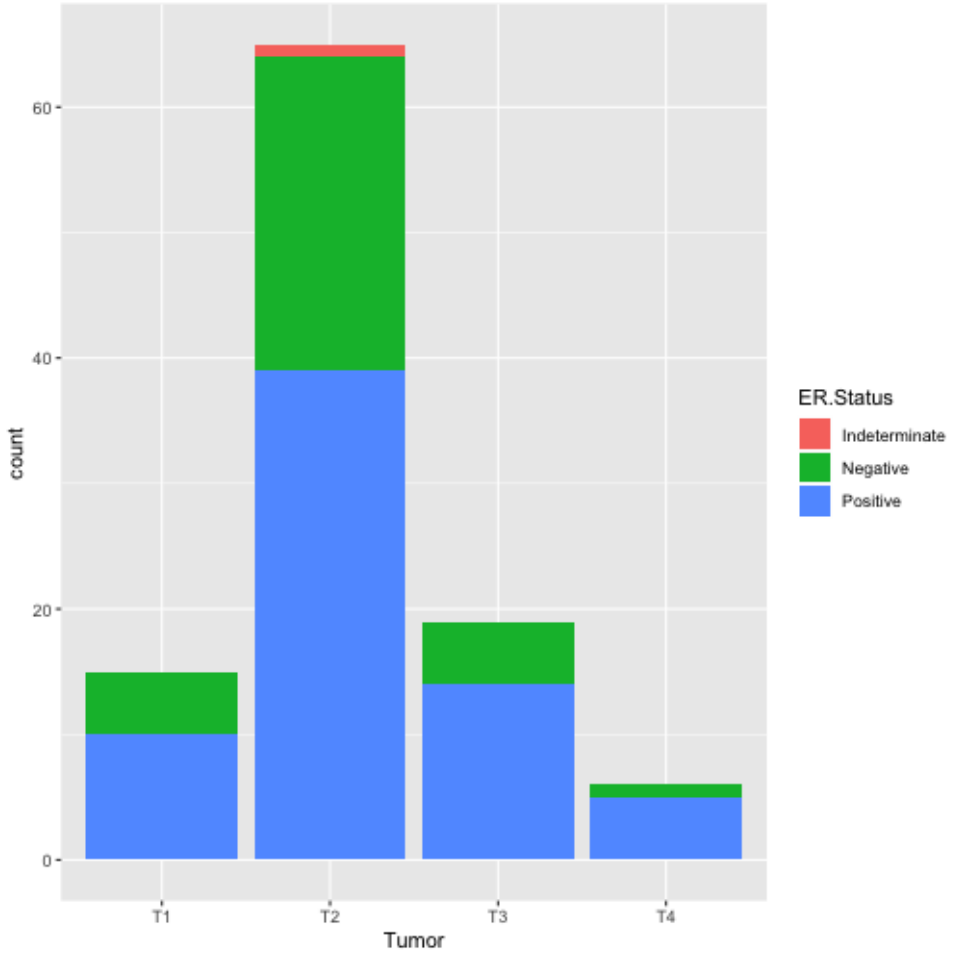
```
dat_brca <- rio::import('data/clinical_data_breast_ca
                        check.names = T)
ggplot(
  data=dat_brca,
  aes(x = Tumor))+
  geom_bar()
```



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()
```

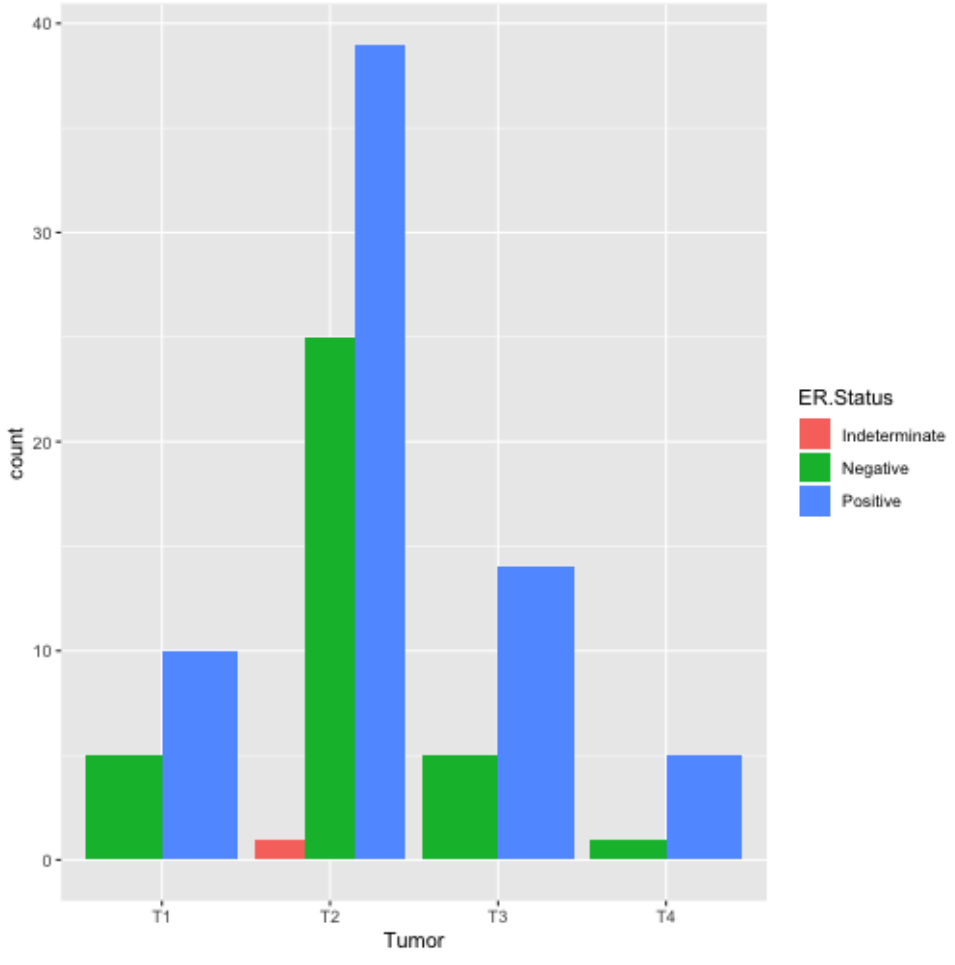
Add additional information via mapping



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
```

```
#>  # A tibble: 4 x 2
#>    Tumor     n
#>    <chr> <int>
#>  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() must not be used with a y aes
```

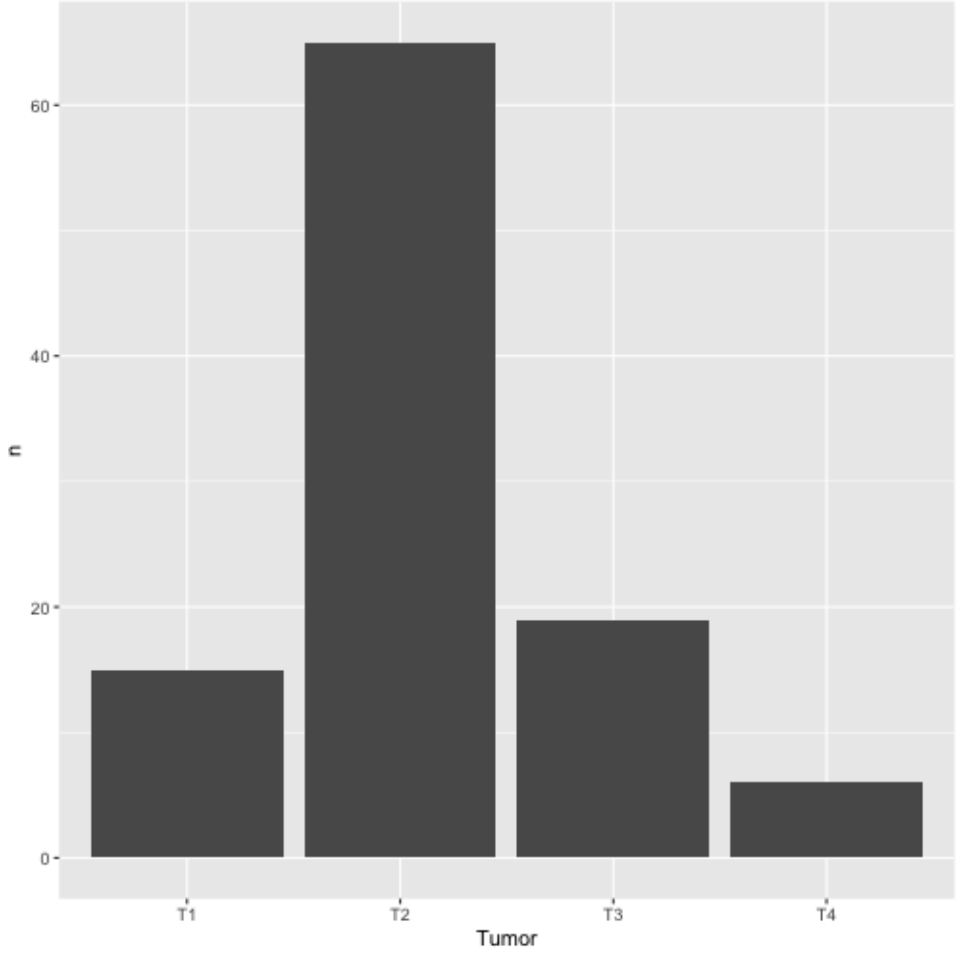
Graphing tabulated data

```
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
#>  stat_count: width = NULL, na.rm = FALSE
#>  position_stack
```

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

plt$layers
```

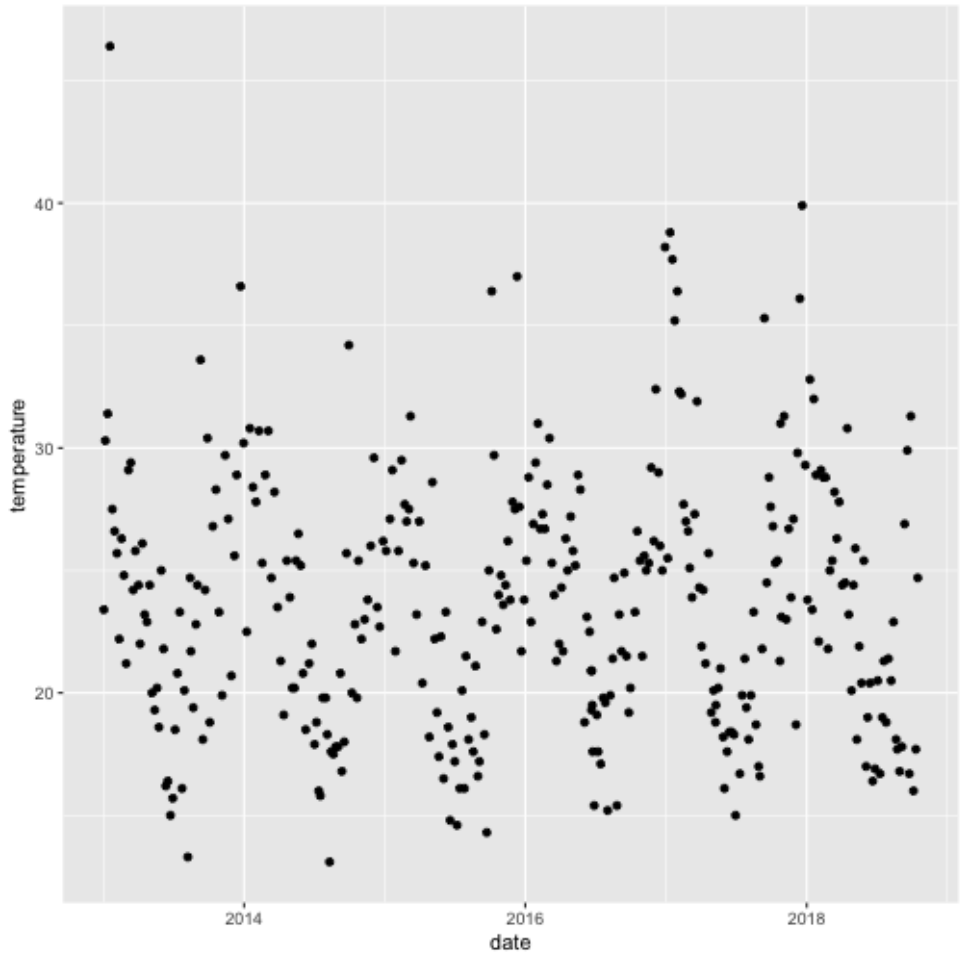
```
#>  [[1]]
#>  geom_bar: width = NULL, na.rm = FALSE
#>  stat_identity: na.rm = FALSE
#>  position_stack
```

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

Bivariate plots

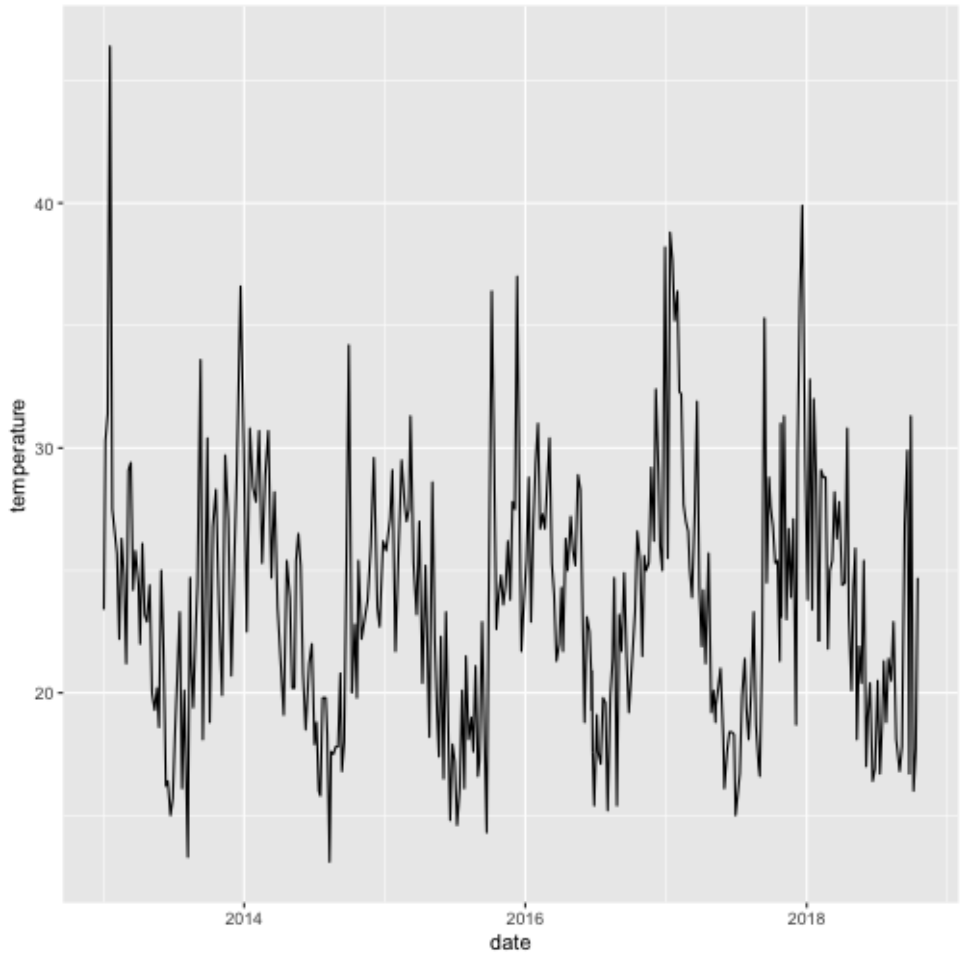
Scatter plots

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



Scatter plots

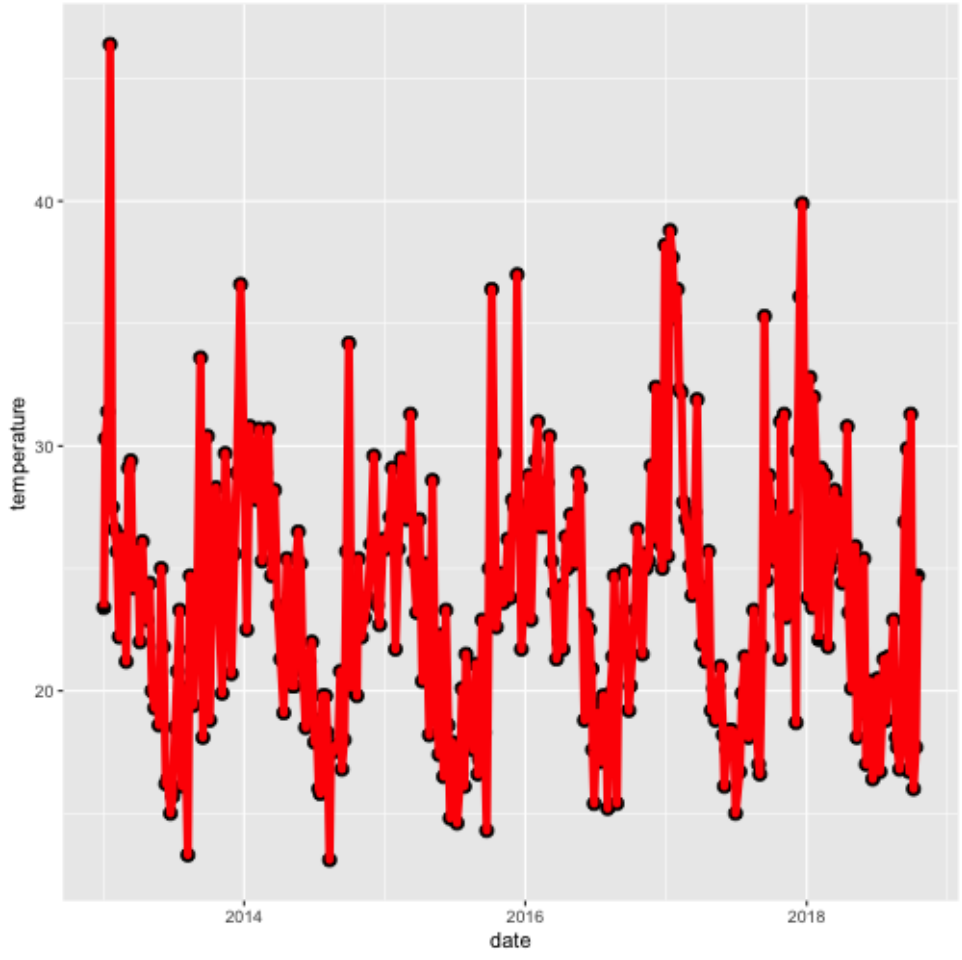
```
ggplot(
  data = beaches,
  aes(x = date, y = temperature))+
  geom_line()
```



Scatter plots

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

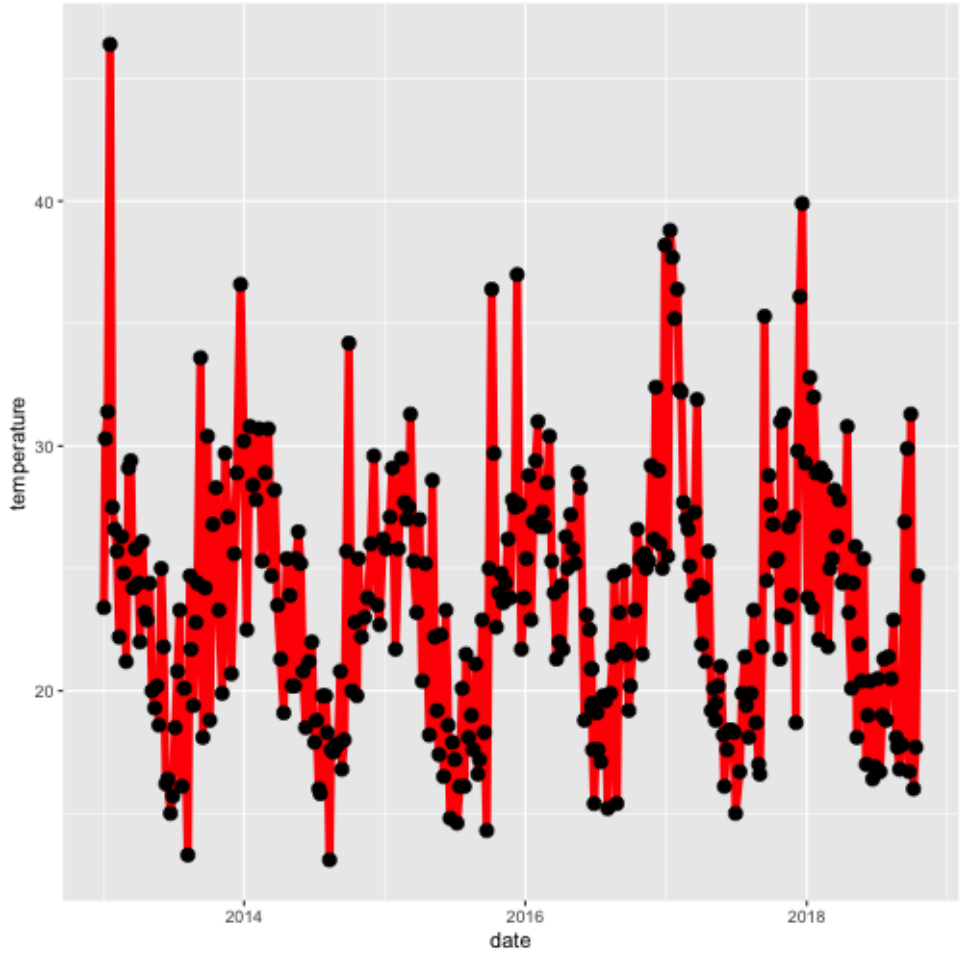
Layer points and lines



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

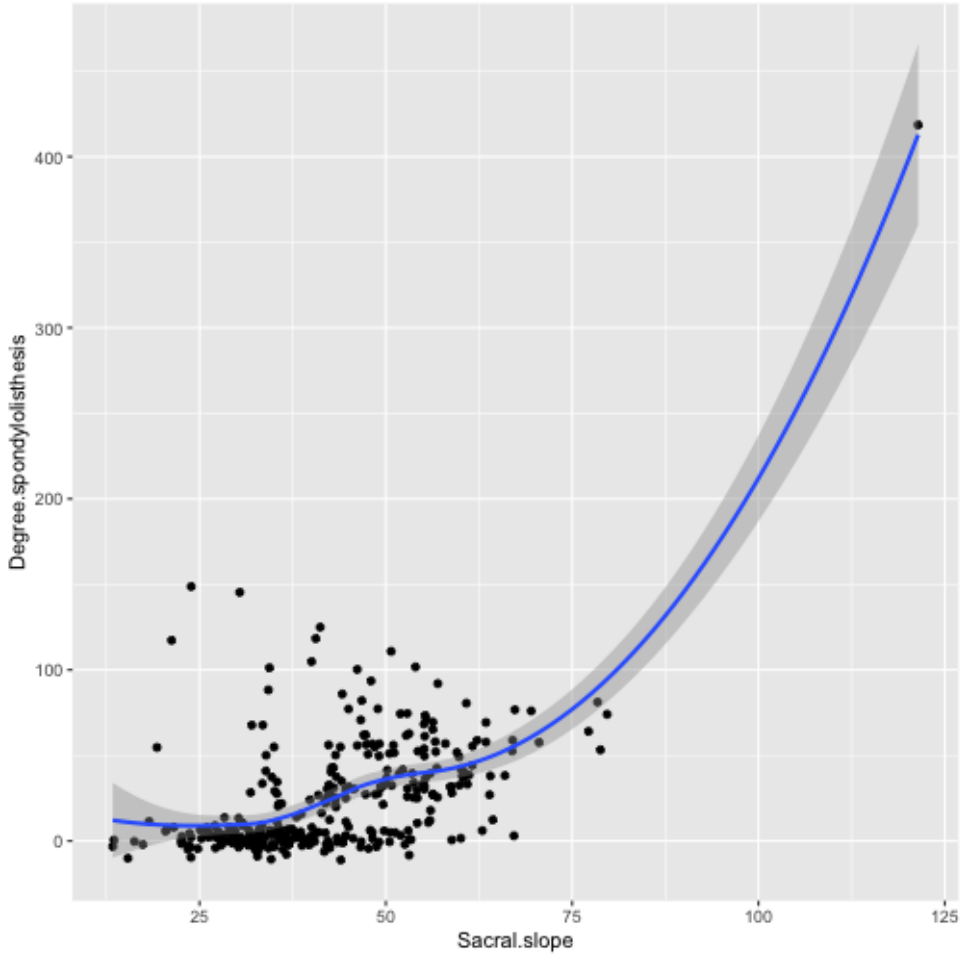
```
ggplot(
  data = beaches,
  aes(x = date, y = temperature)) +
  geom_point() +
  geom_smooth(span = 0.1)
```

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

Averages over 10% of the data

Computations over groups

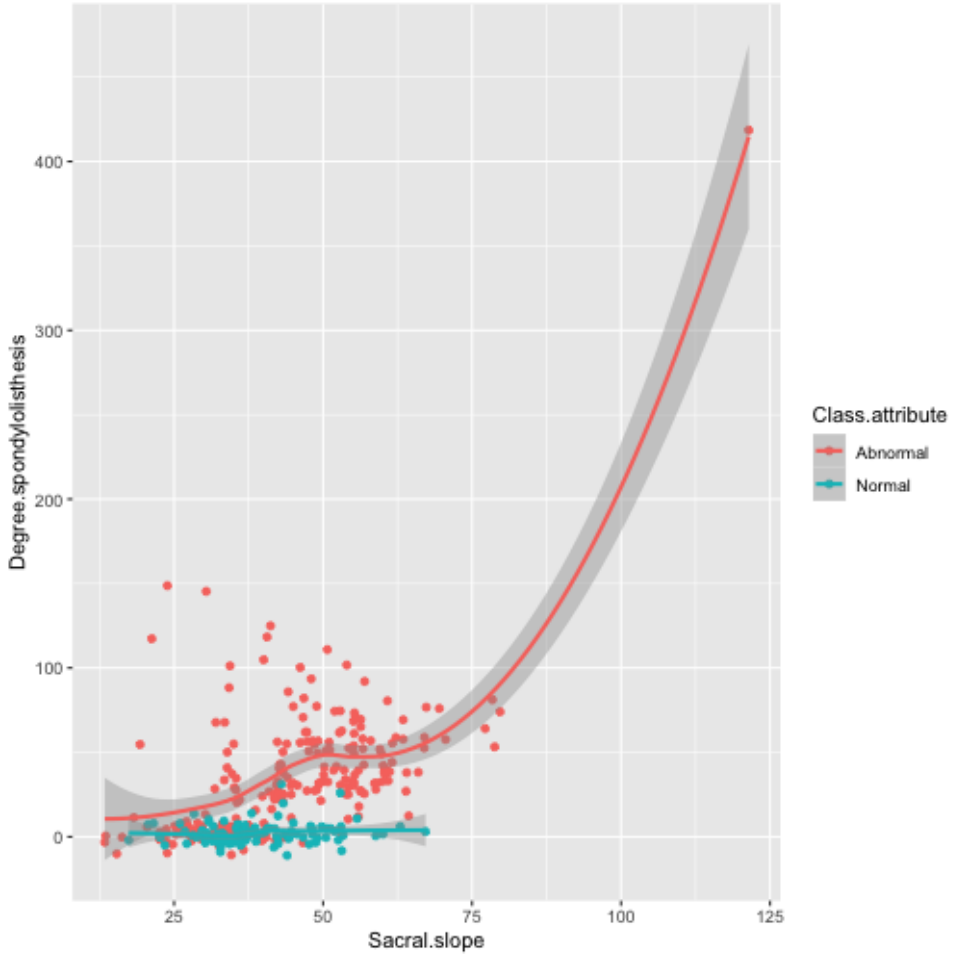
```
ggplot(
  data = dat_spine,
  aes(x = Sacral.slope,
      y = Degree.spondylolisthesis)) +
  geom_point() +
  geom_smooth()
```



Computations over groups

```
ggplot(
  data = dat_spine,
  aes(x = Sacral.slope,
      y = Degree.spondylolisthesis,
      color = Class.attribute)) +
  geom_point() +
  geom_smooth()
```

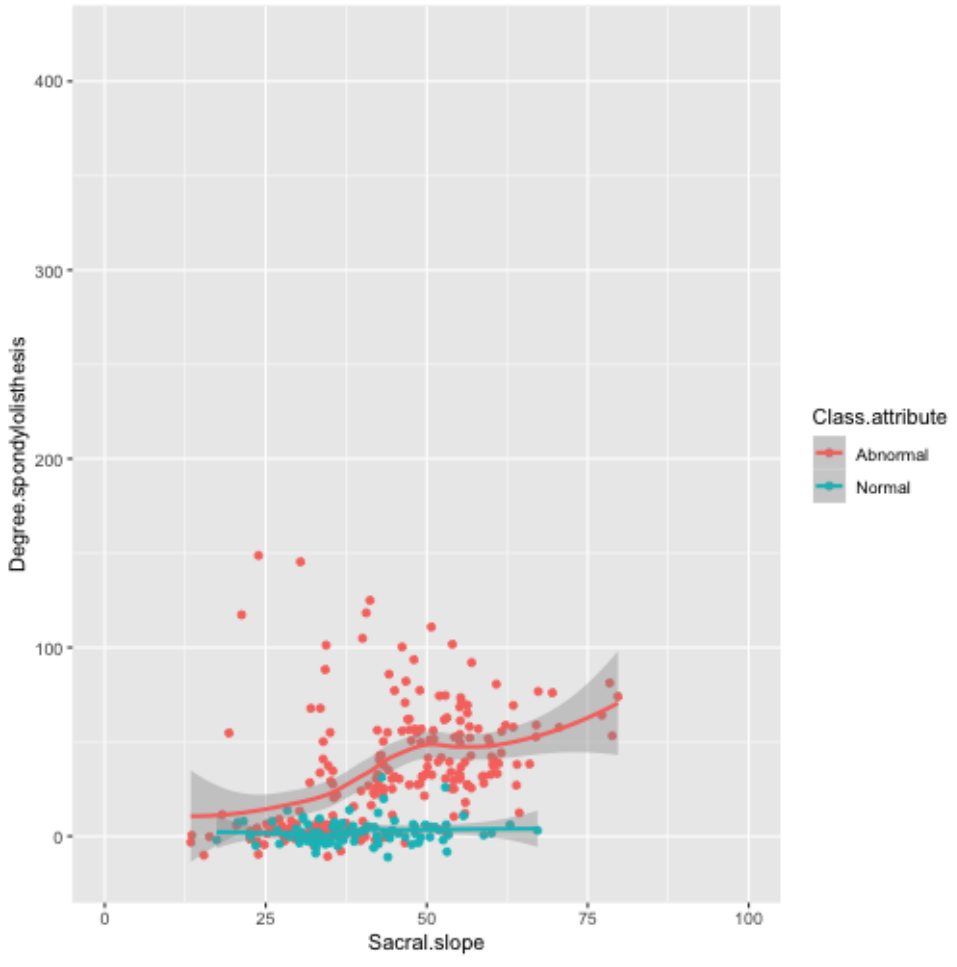
Computation is done by groups



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

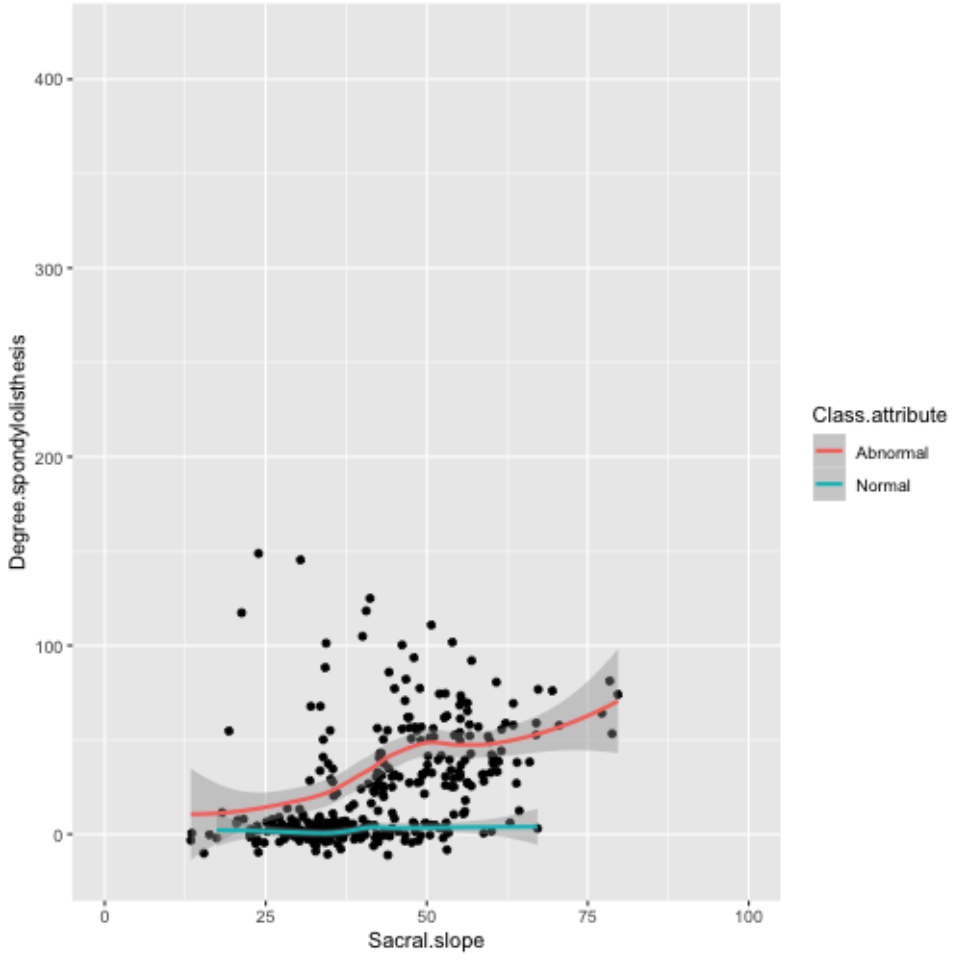


Computations over groups

```
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

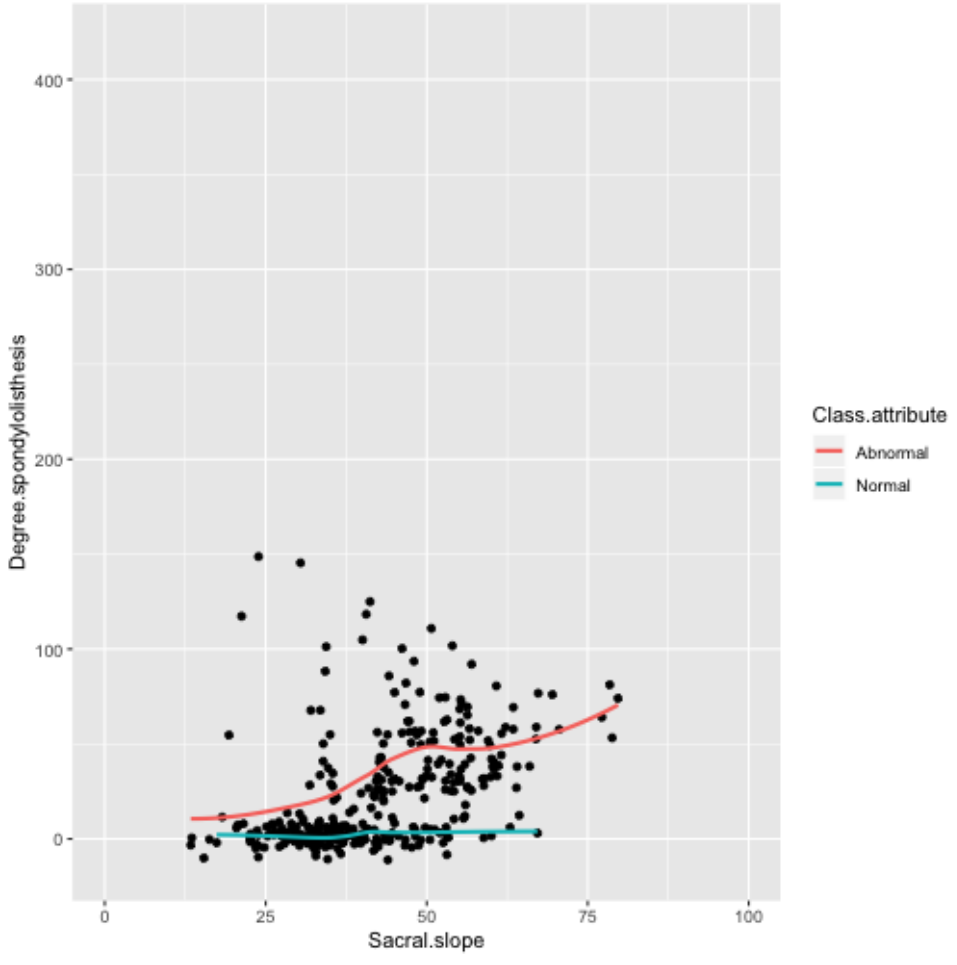


Computations over groups

```
ggplot(
  data = dat_spine,
  aes(x = Sacral.slope,
      y = Degree.spondylolisthesis)) +
  geom_point() +
  geom_smooth(aes(color = Class.attribute),
              se = F) +
  xlim(0, 100)
```

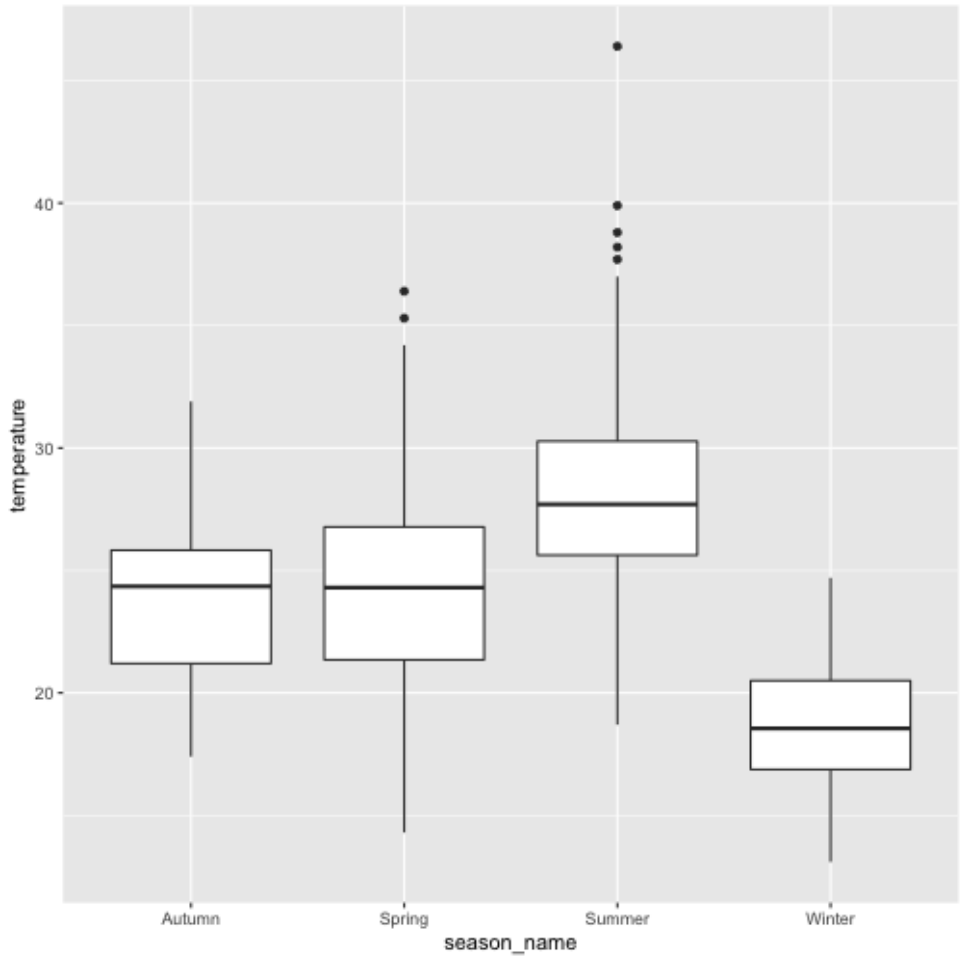
Remove the confidence bands

Maybe a cleaner look



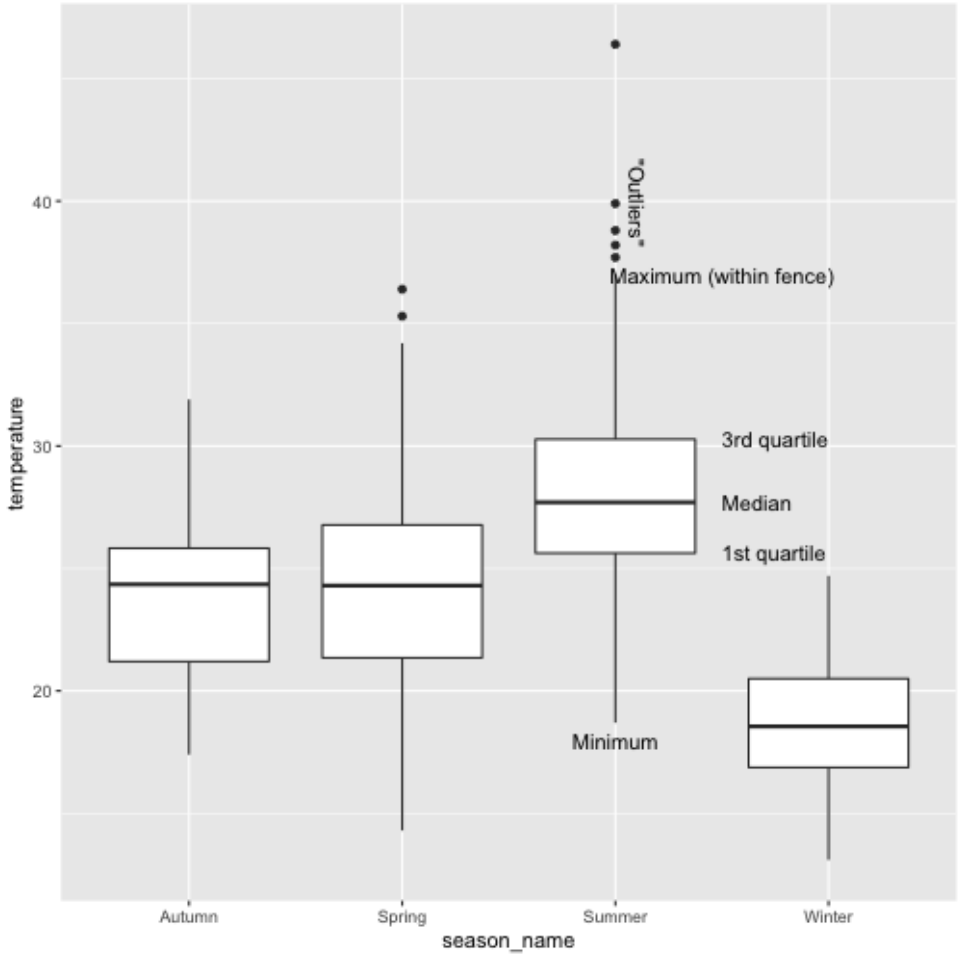
Box Plots

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



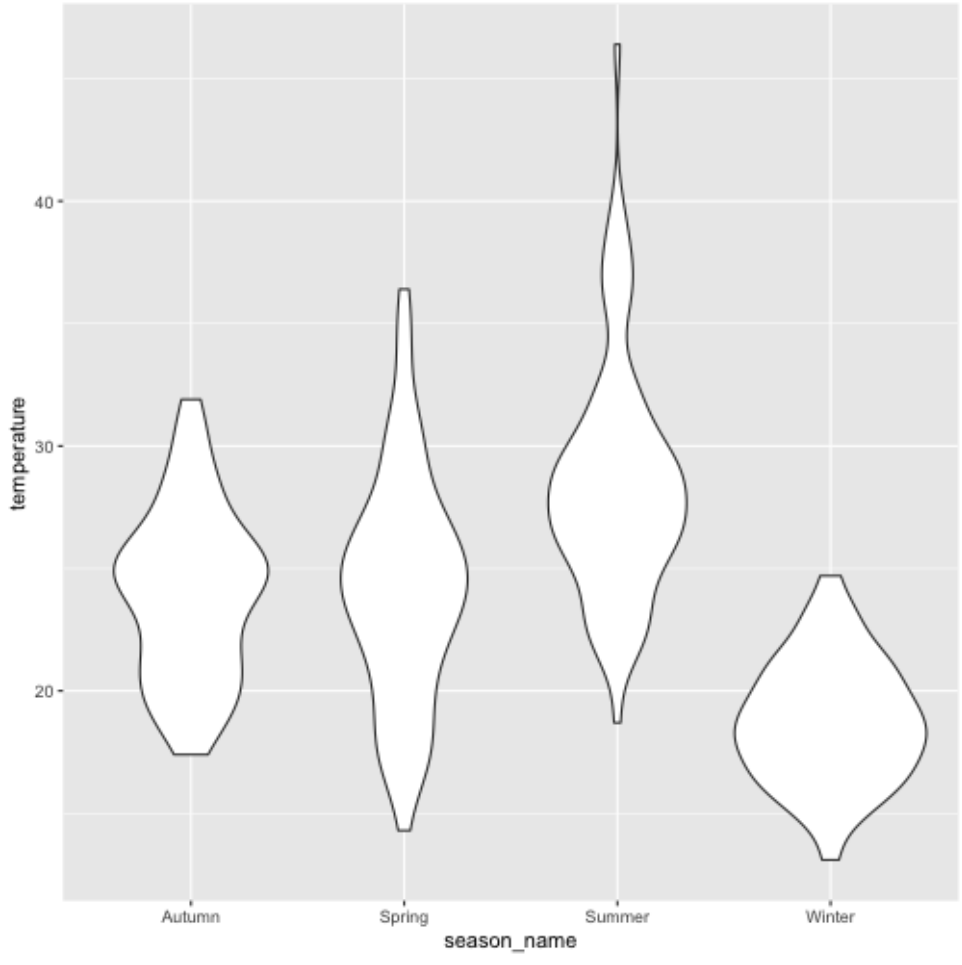
Box Plots

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



Violin plots

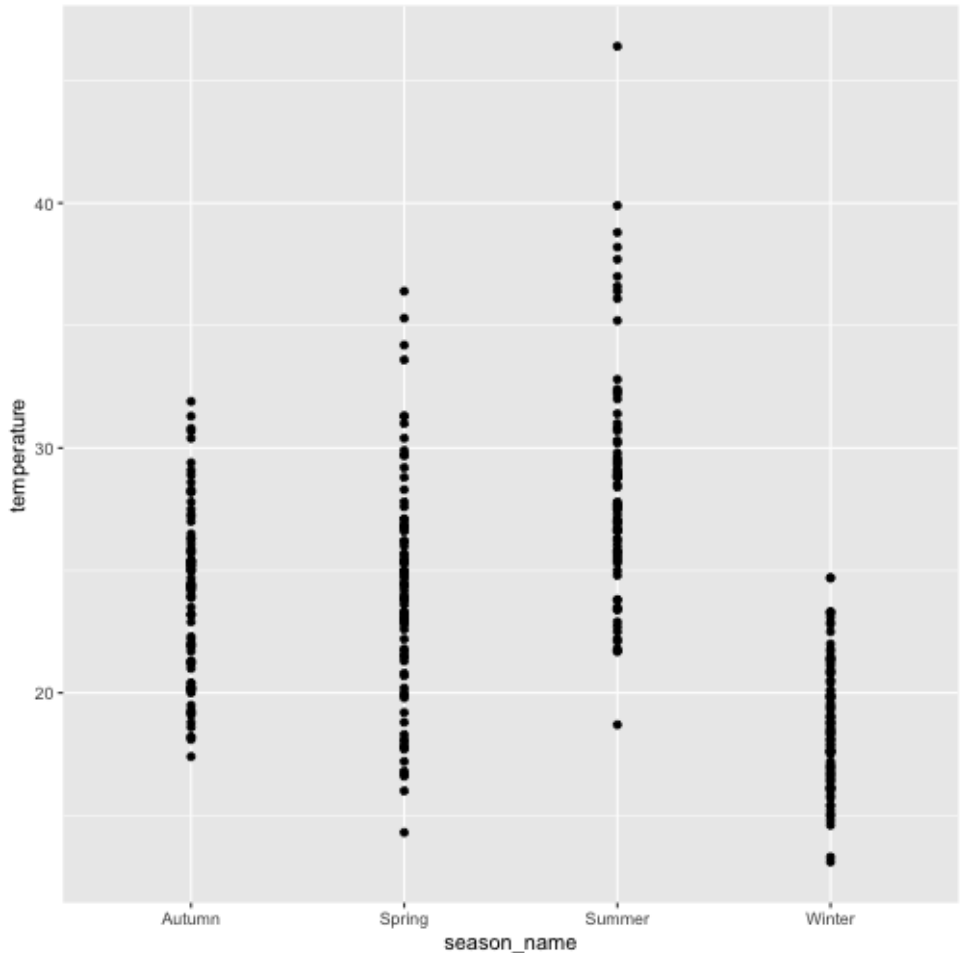
```
ggplot(
  data = beaches,
  aes(x = season_name,
      y = temperature)) +
  geom_violin()
```



Strip plots

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

Points are overlayed on each other

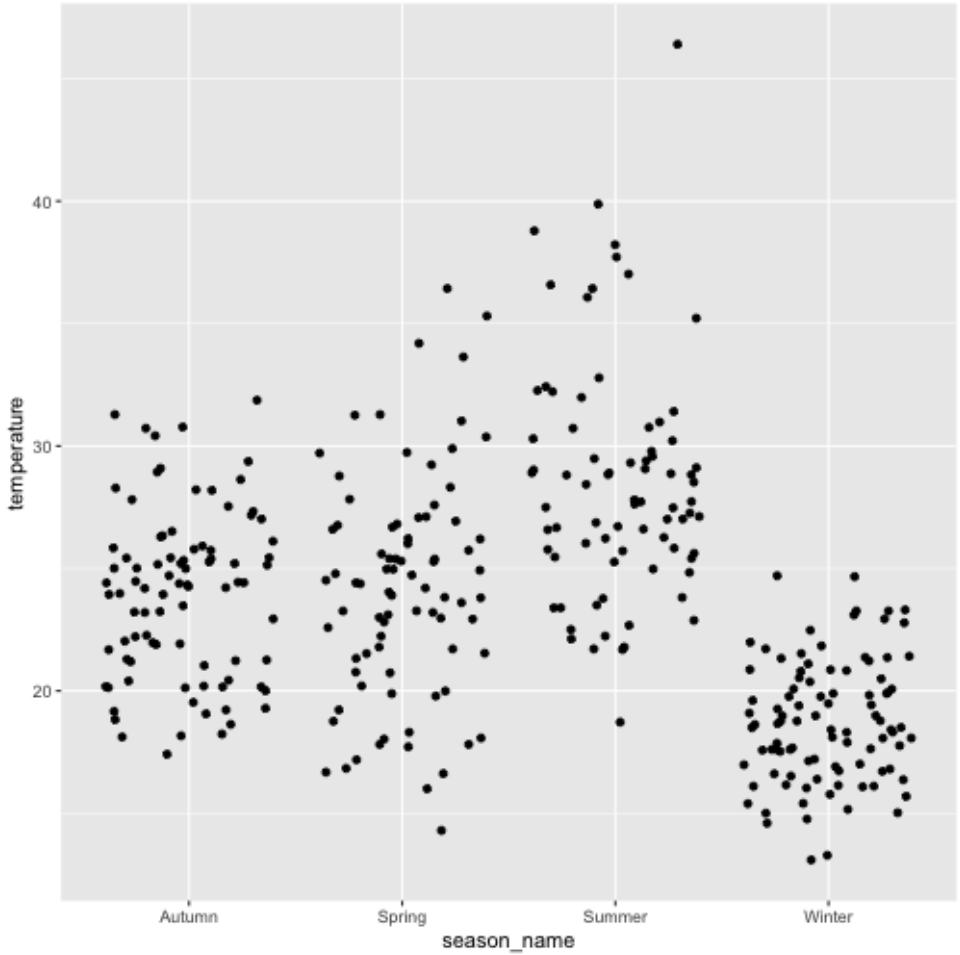


Strip plots

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

Jittering allows all points to be seen

Maybe too much

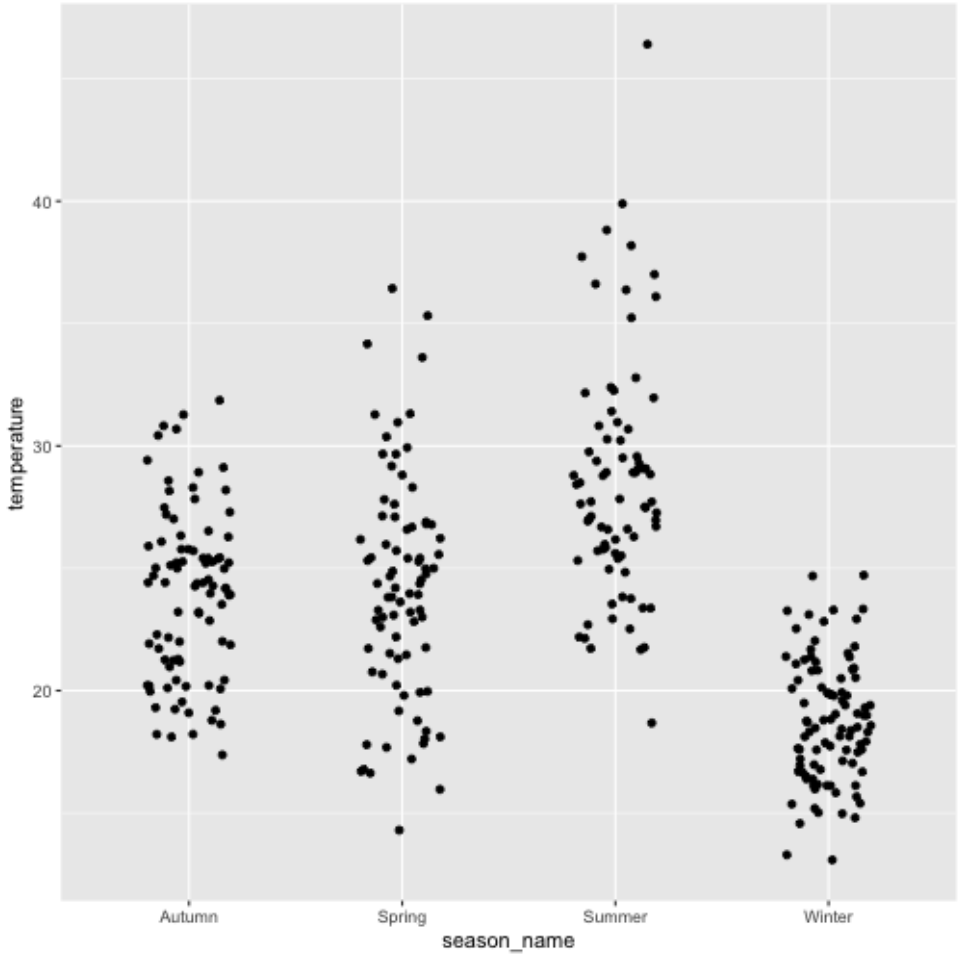


Strip plots

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

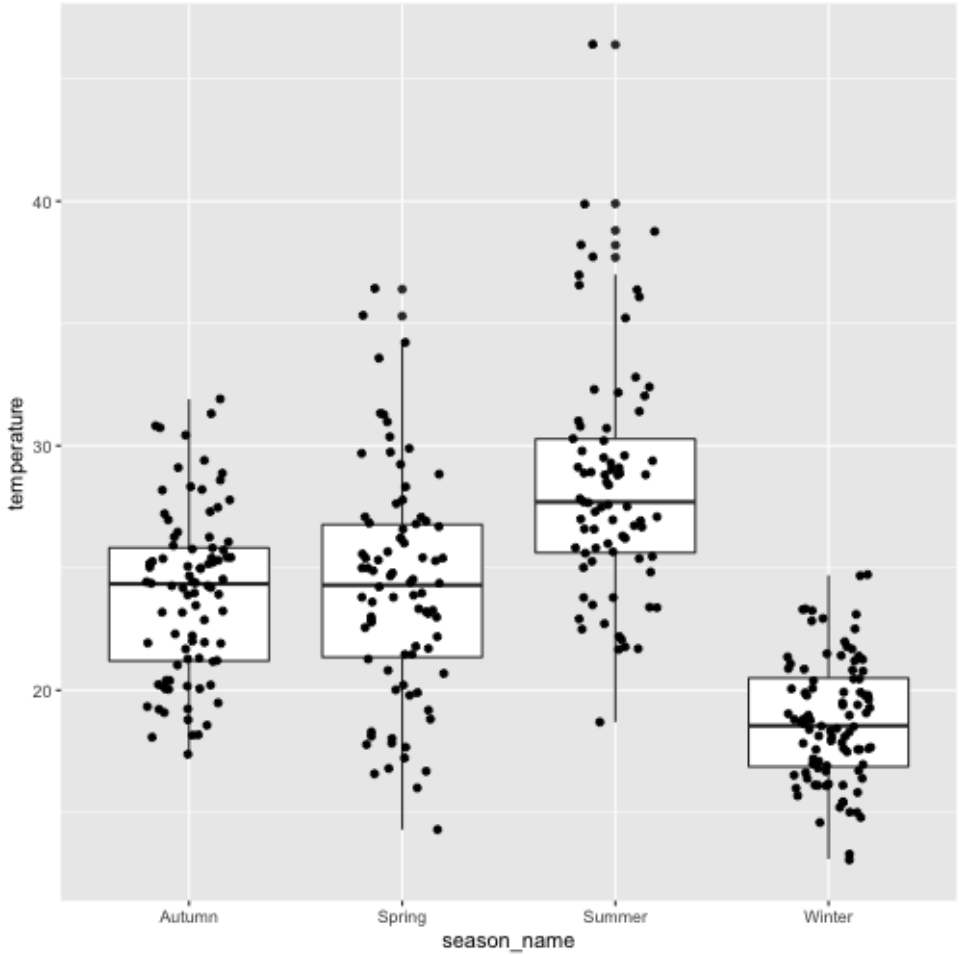
Jittering allows all points to be seen

Maybe too much



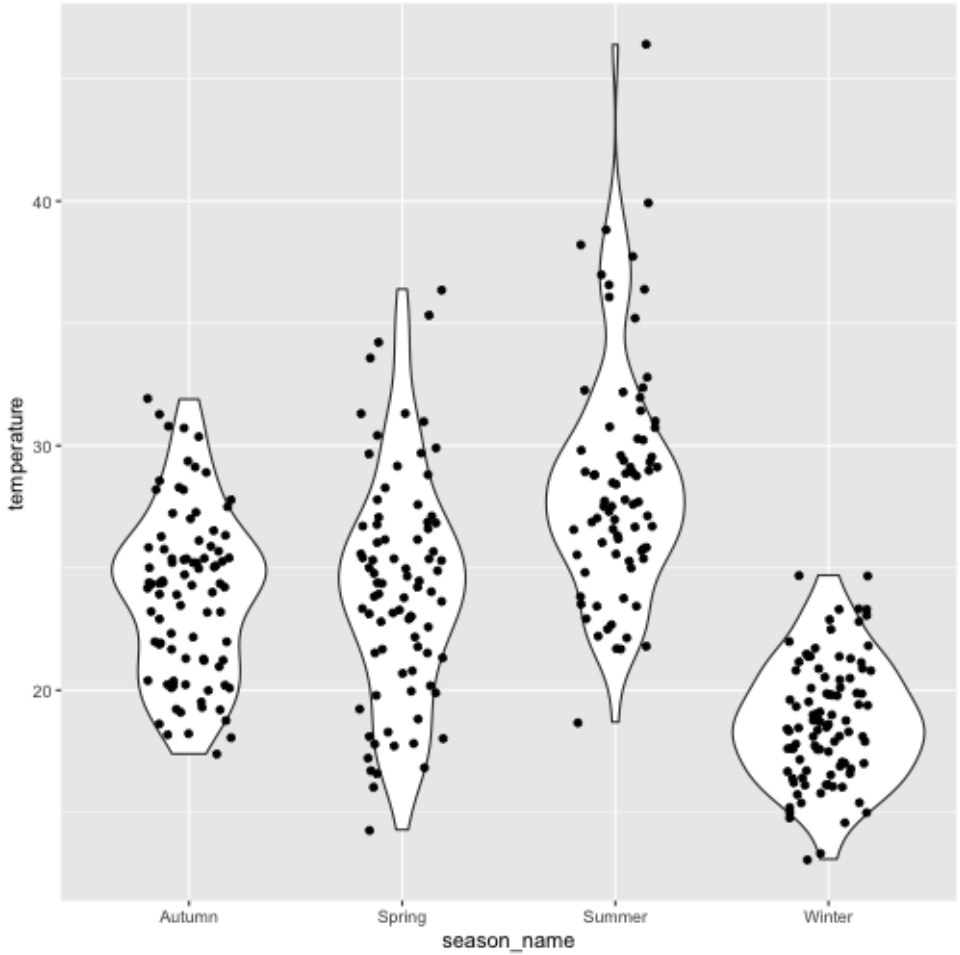
Layers, again

```
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)
```

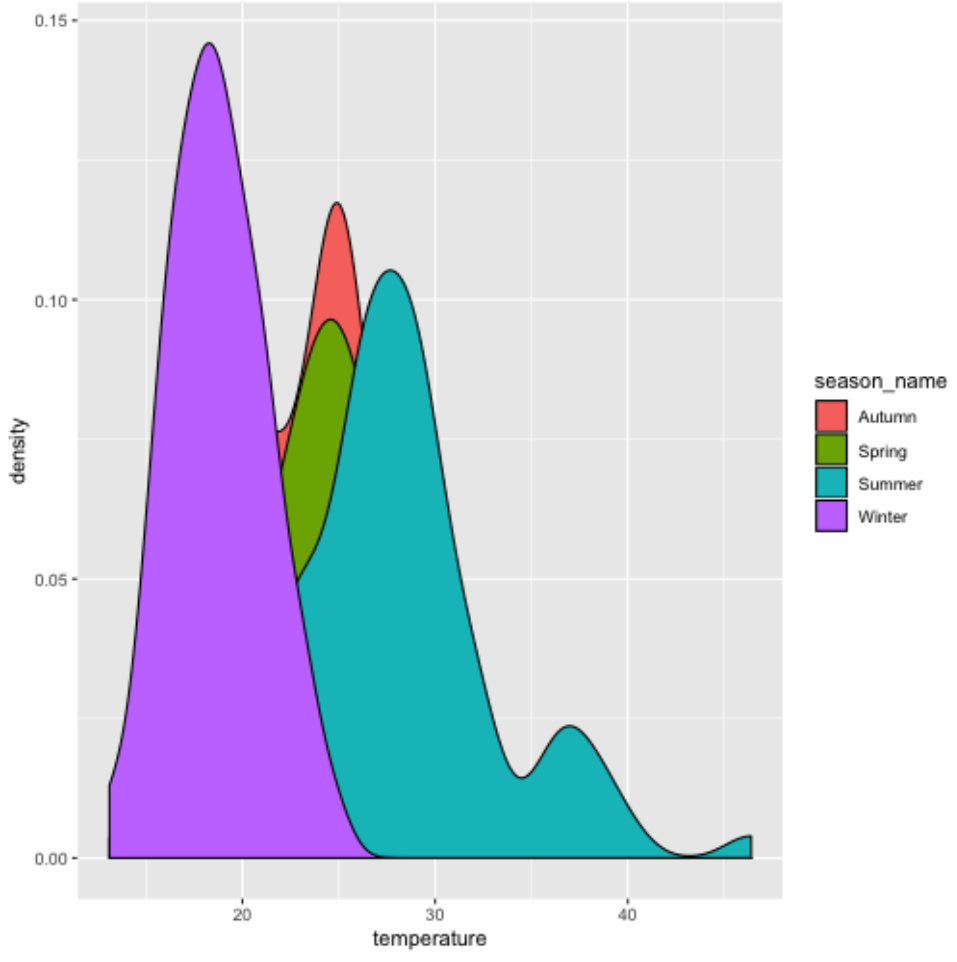


Exploring grouped data

Let's start here

```
ggplot(
  data = beaches,
  aes(x = temperature,
      fill = season_name)) +
  geom_density()
```

Not very useful

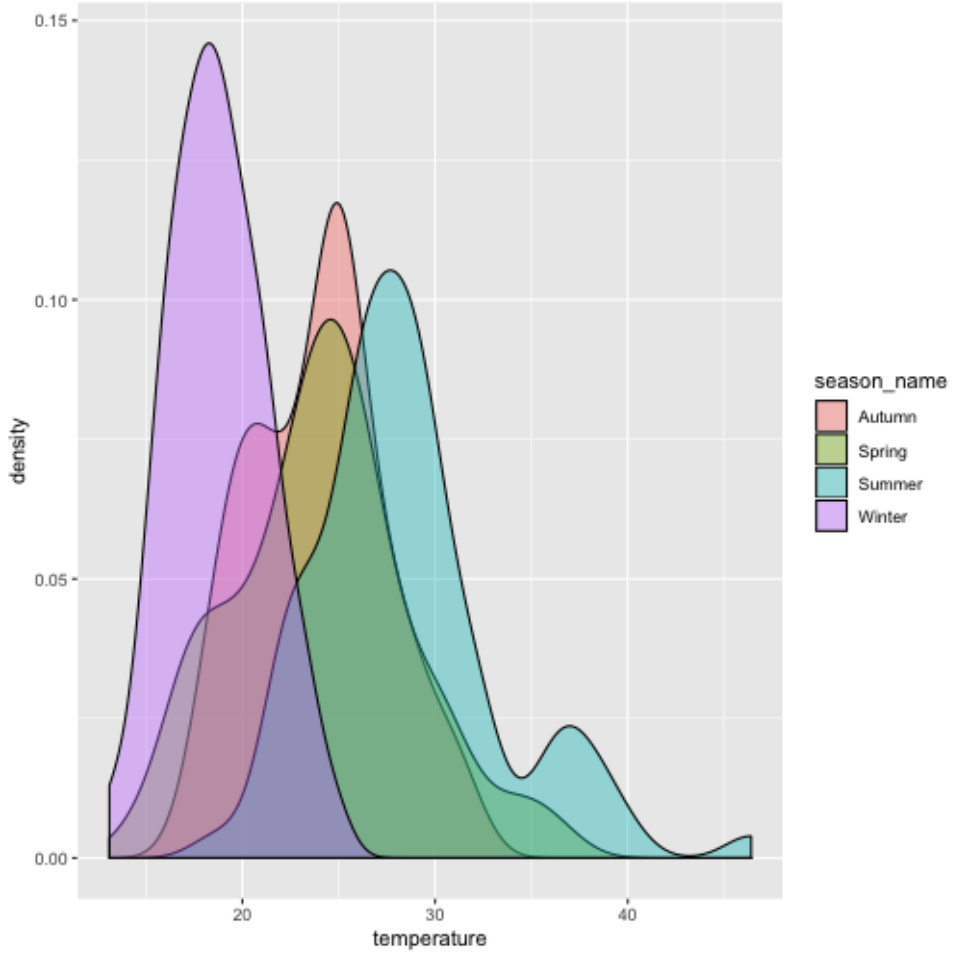


Overlaying graphs

```
ggplot(
  data = beaches,
  aes(x = temperature,
      fill = season_name)) +
  geom_density(alpha = 0.4)
```

Make graphs more transparent

Still pretty cluttered

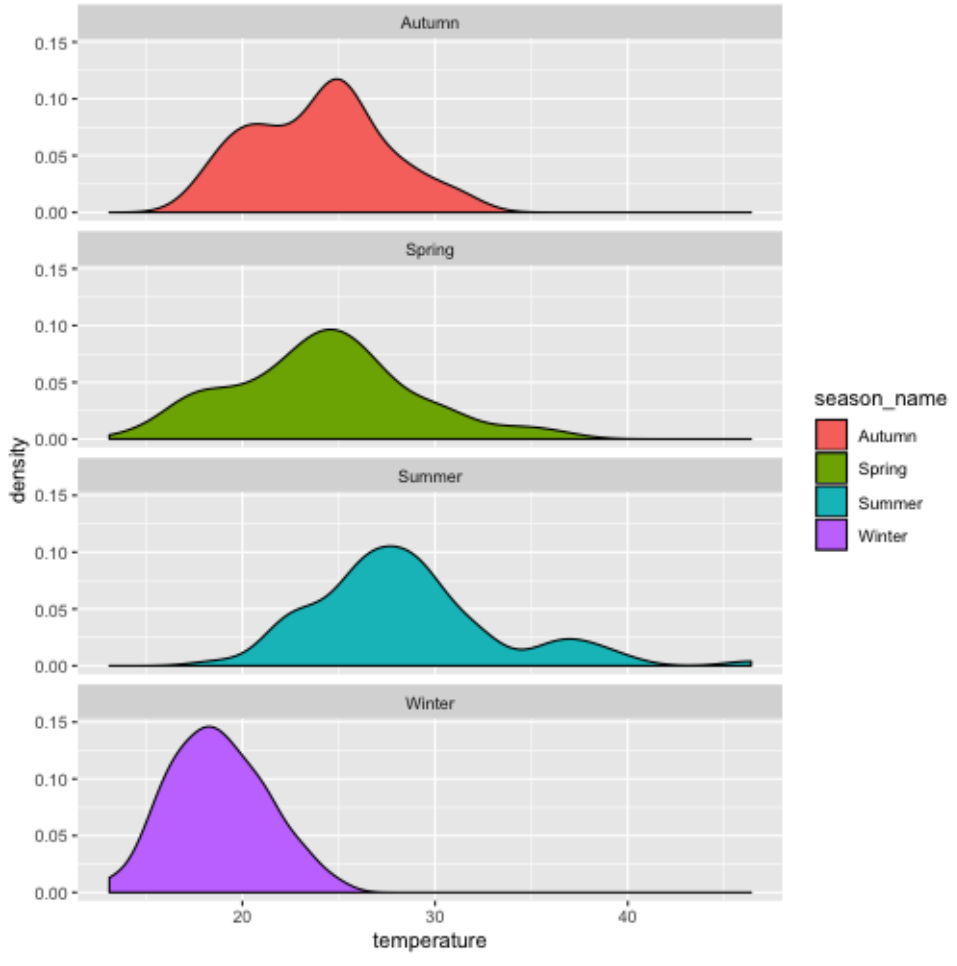


Exploding graphs

```
ggplot(
  data = beaches,
  aes(x = temperature,
      fill = season_name)) +
  geom_density() +
  facet_wrap(~ season_name, ncol = 1)
```

Let's explode these out

This is called "small multiples" (Tufte) or "trellis graphics" (Cleveland)

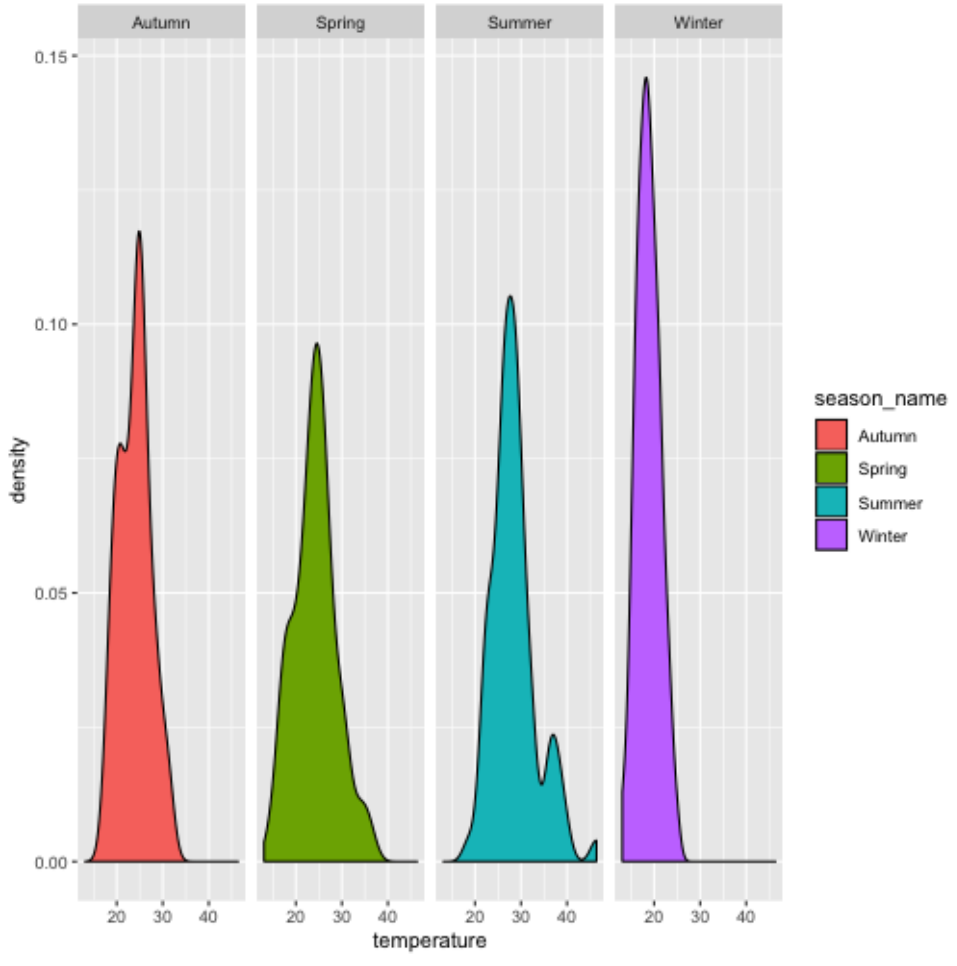


> Notice that all the graphs have the same x-axis. This is a good thing

Exploding graphs

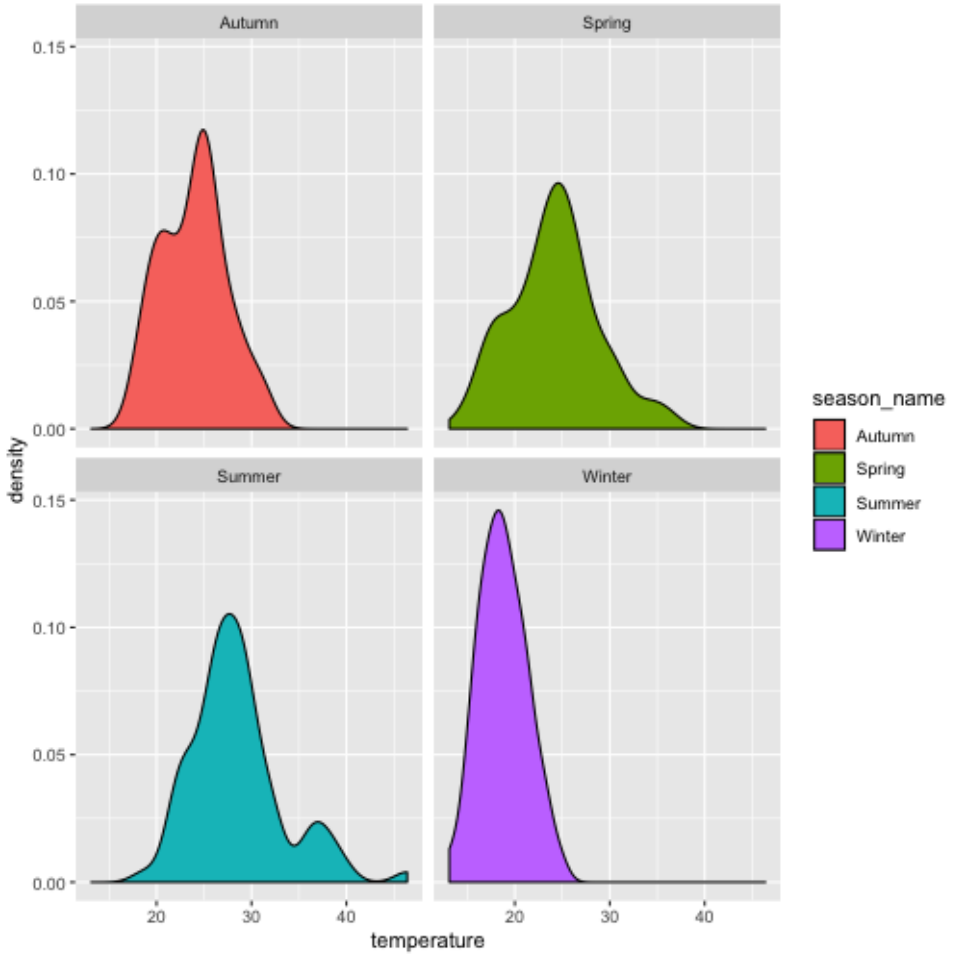
```
ggplot(
  data = beaches,
  aes(x = temperature,
      fill = season_name)) +
  geom_density() +
  facet_wrap(~ season_name, nrow = 1)
```

We can arrange them the other way too



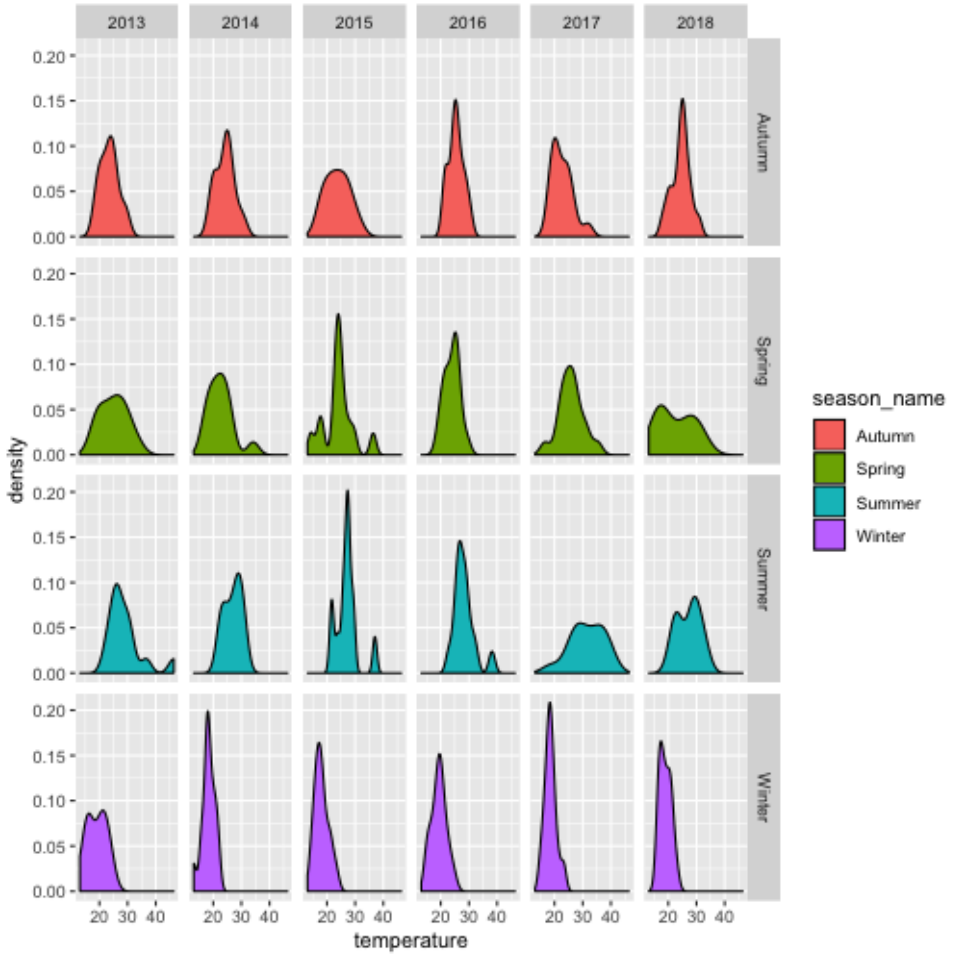
Re-arranging graphs

```
ggplot(
  data = beaches,
  aes(x = temperature,
      fill = season_name)) +
  geom_density() +
  facet_wrap(~season_name, ncol = 2)
```



Creating a grid of graphs

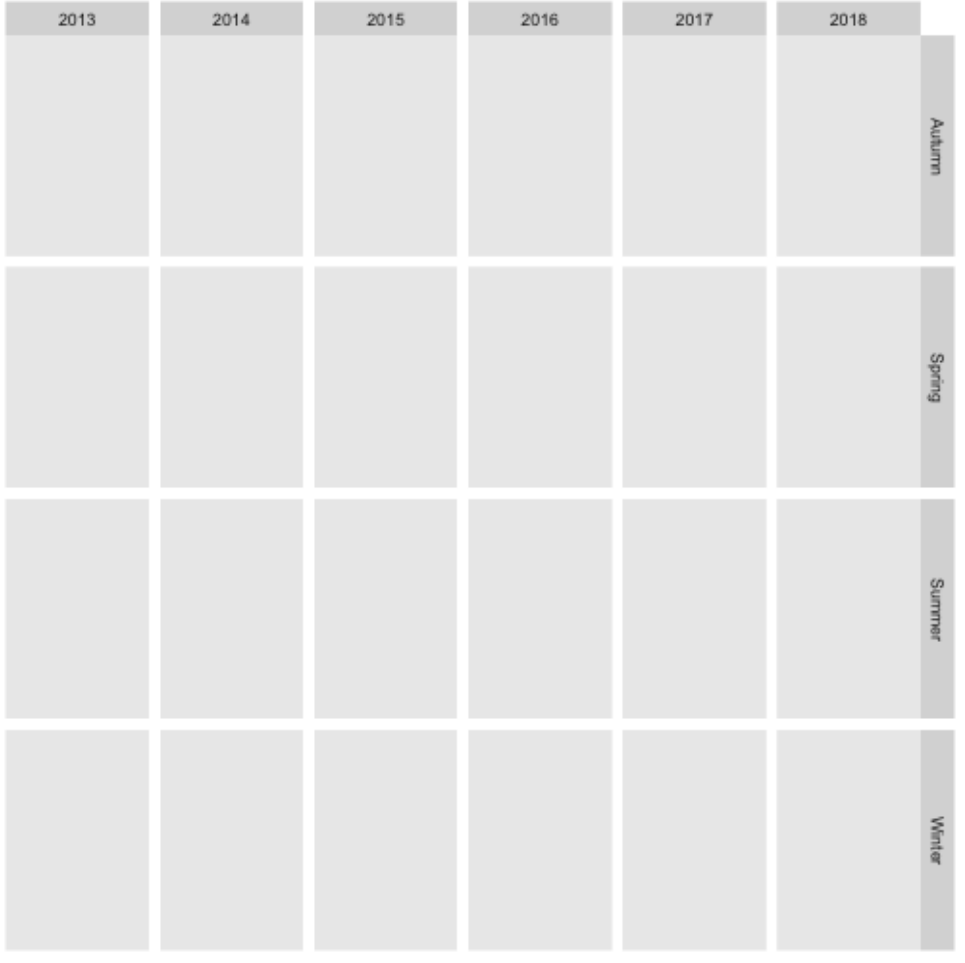
```
ggplot(
  data = beaches,
  aes(x = temperature)) +
  geom_density(aes(fill = season_name)) +
  facet_grid(rows = vars(season_name),
             cols = vars(year))
```



Grids of graphs

Start with a blank slate

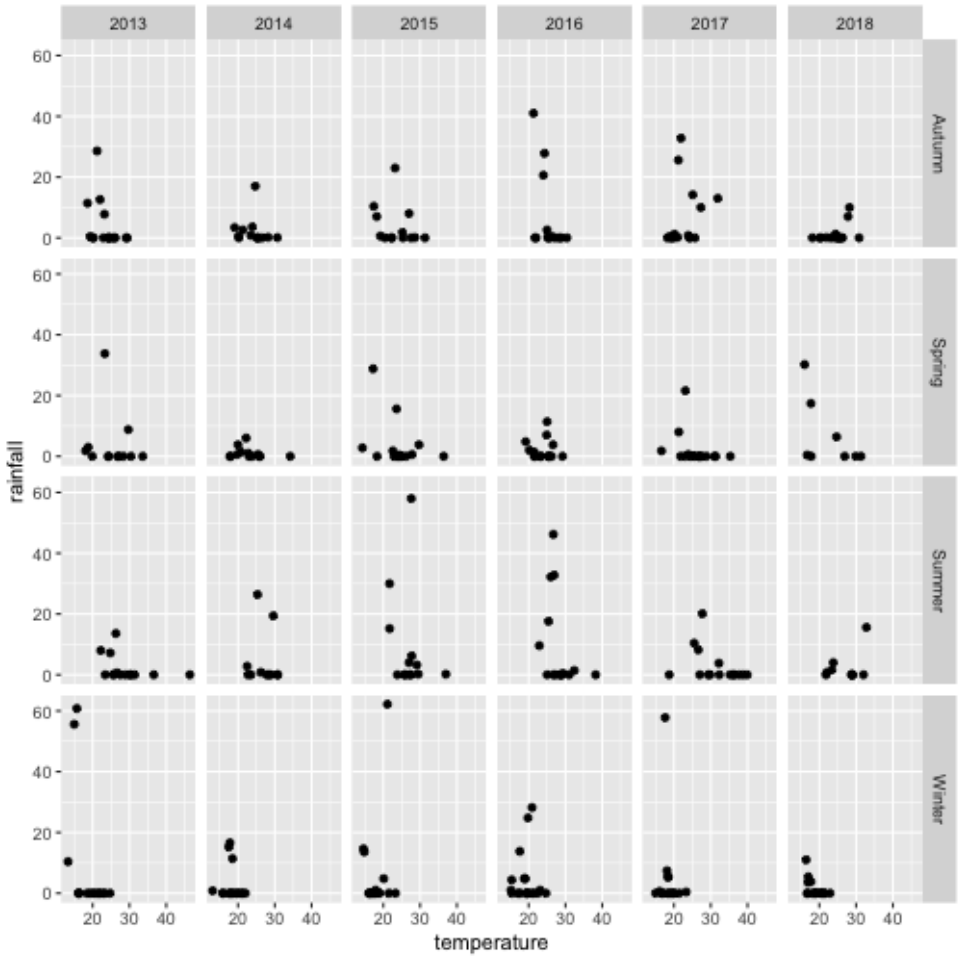
```
ggplot(
  data = beaches) +
  facet_grid(rows = vars(season_name),
             cols = vars(year))
```



Grids of graphs

Create geoms you want

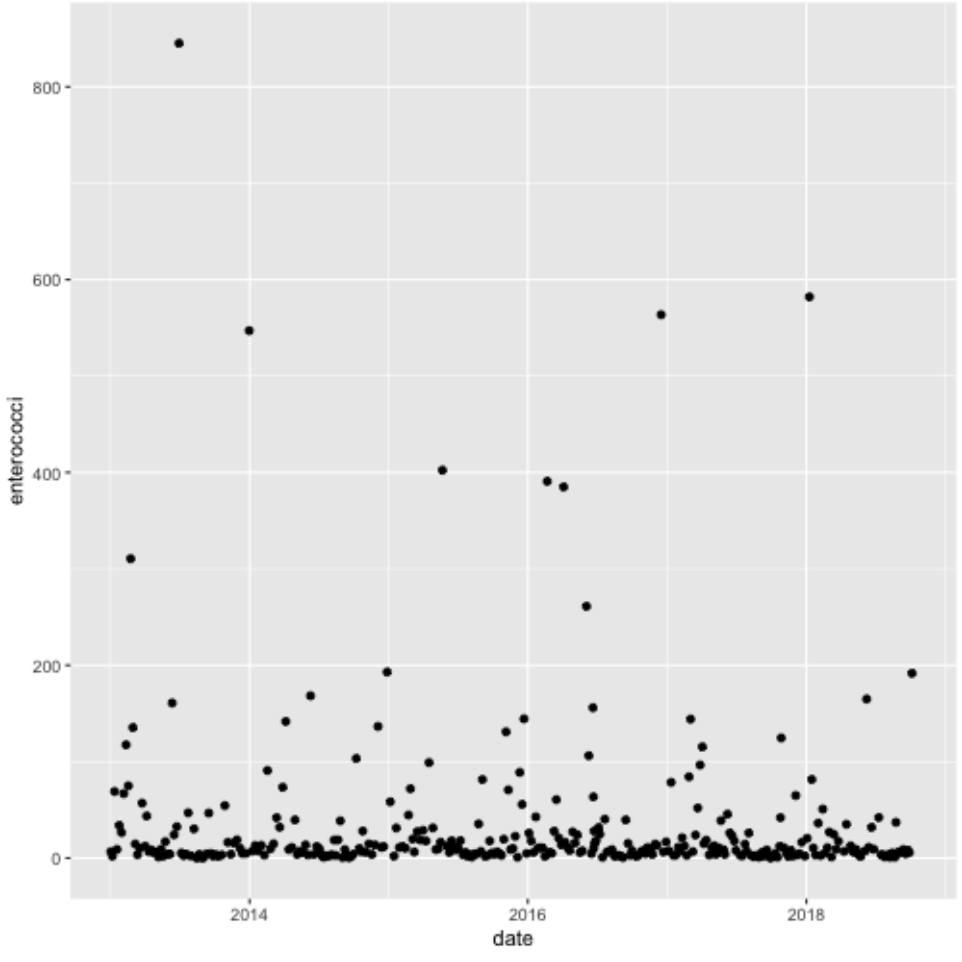
```
ggplot(
  data = beaches) +
  facet_grid(rows = vars(season_name),
             cols = vars(year))+
  geom_point(aes(x = temperature, y = rainfall))
```



Scales

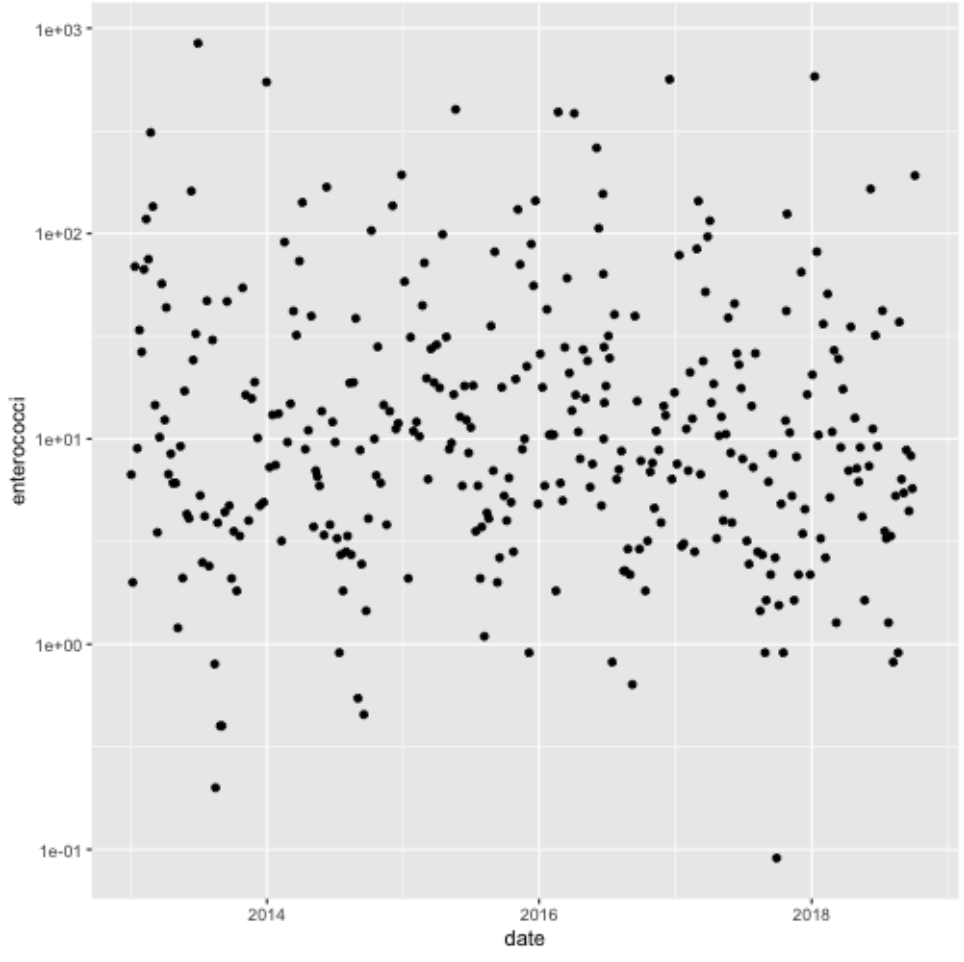
```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci)) +
  geom_point()
```

All the action is happening in the bottom bit



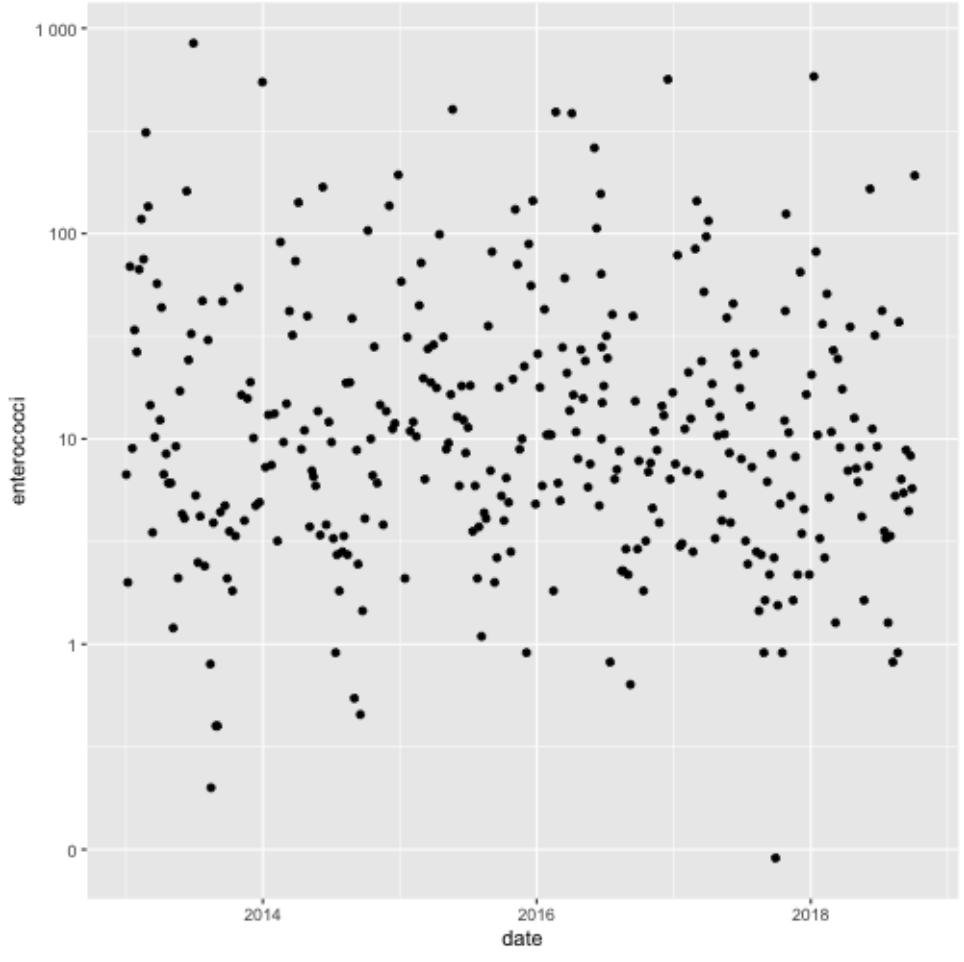
```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci)) +
  geom_point() +
  scale_y_log10()
```

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



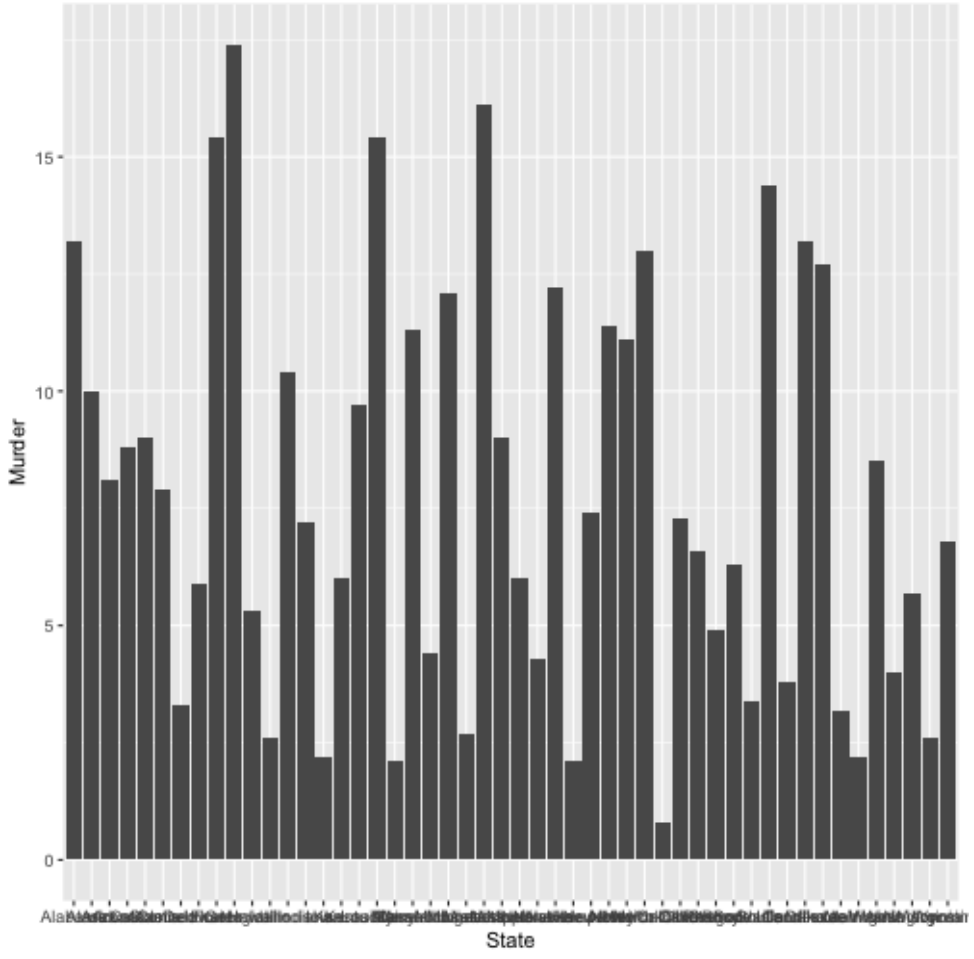
Order and orientation

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

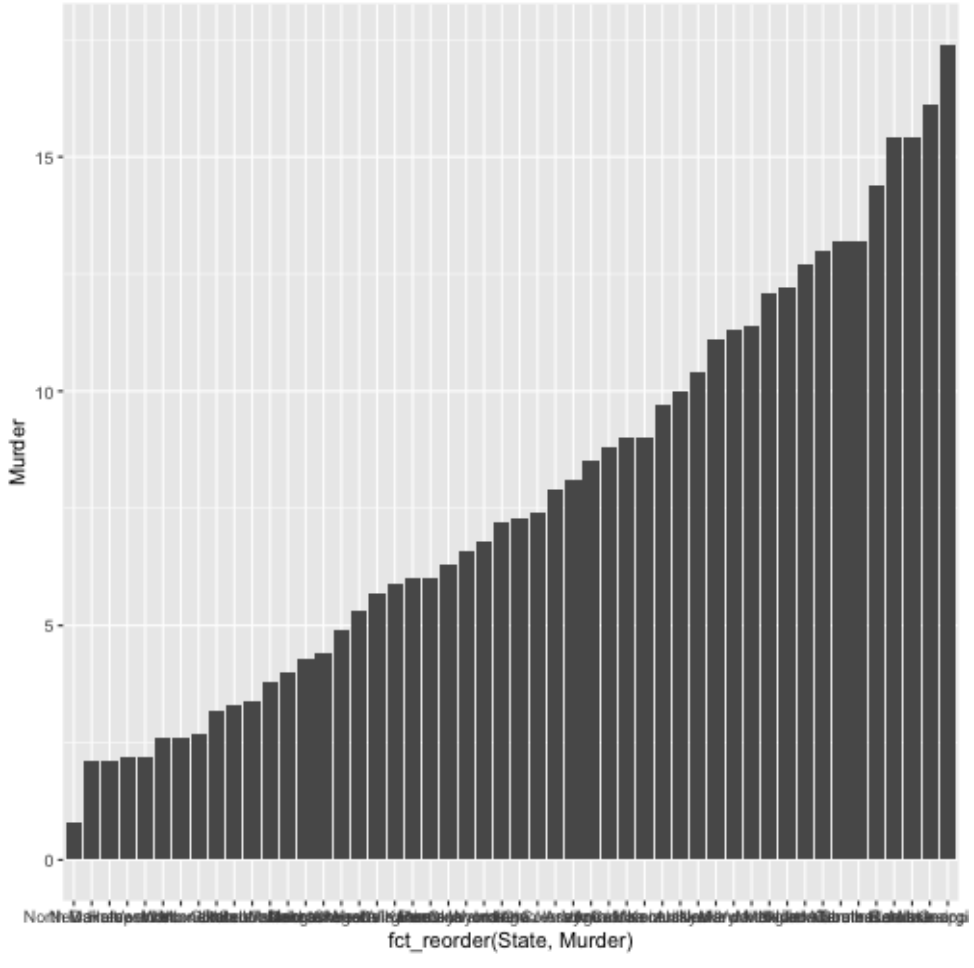
There is no ordering, and states can't be read



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')
```

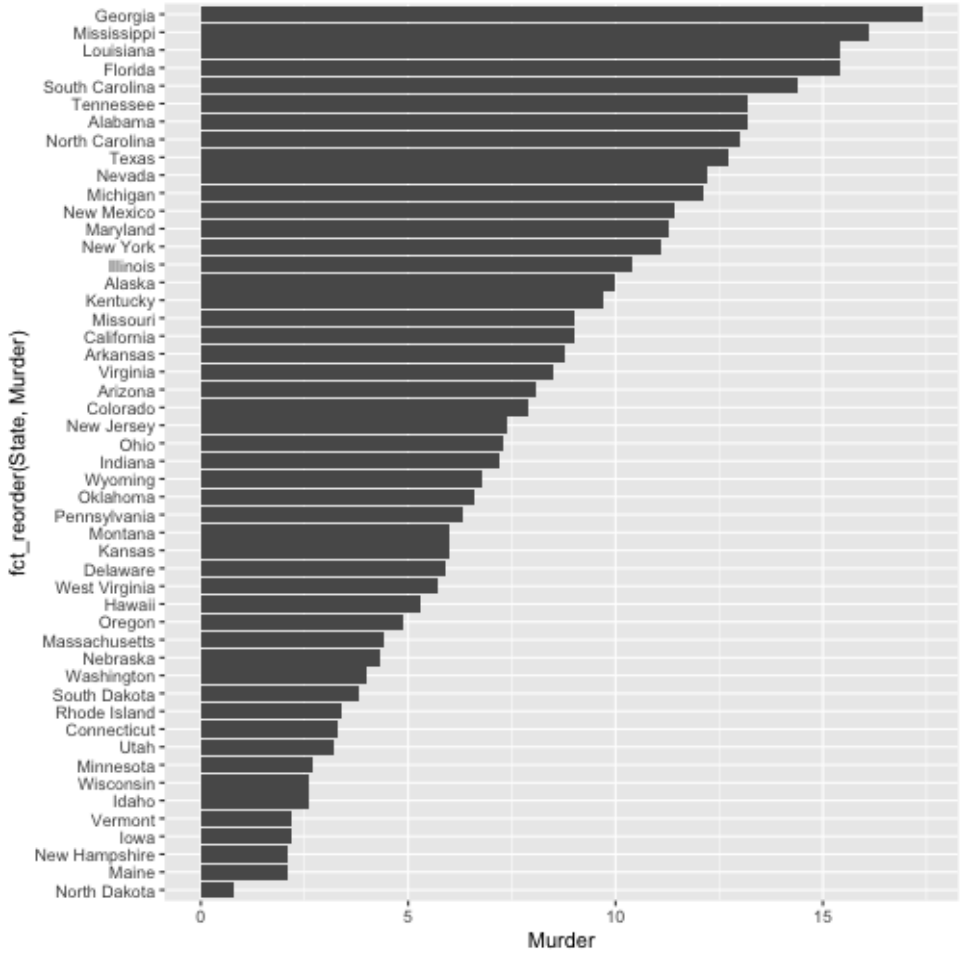
We see the pattern, but its still unreadable



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()
```

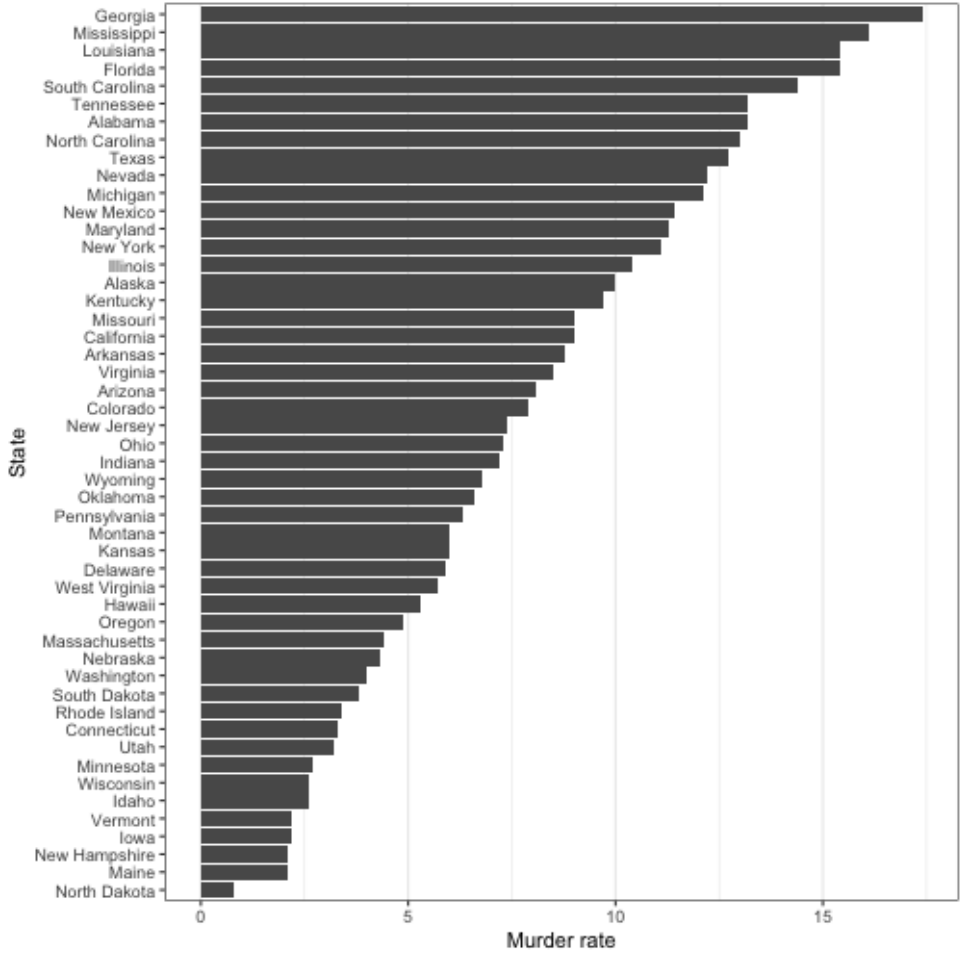
Flipping the axes makes the states readable



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

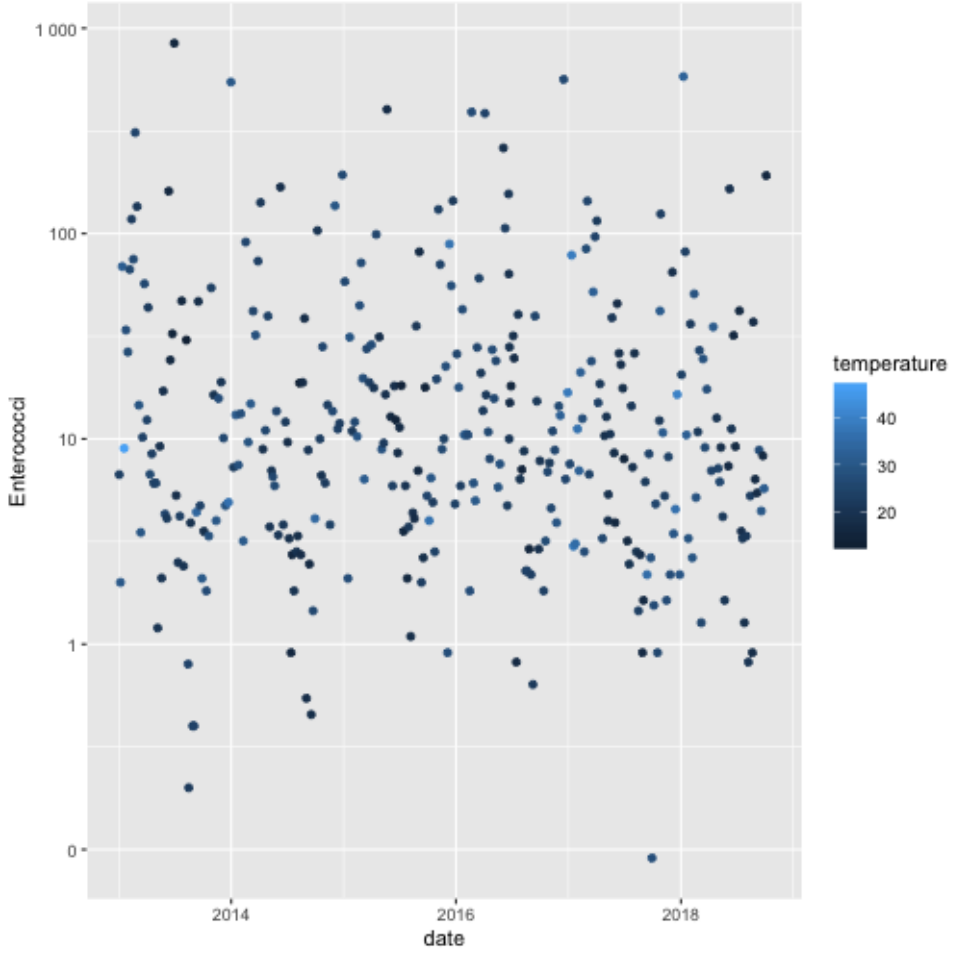
Color schemes

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

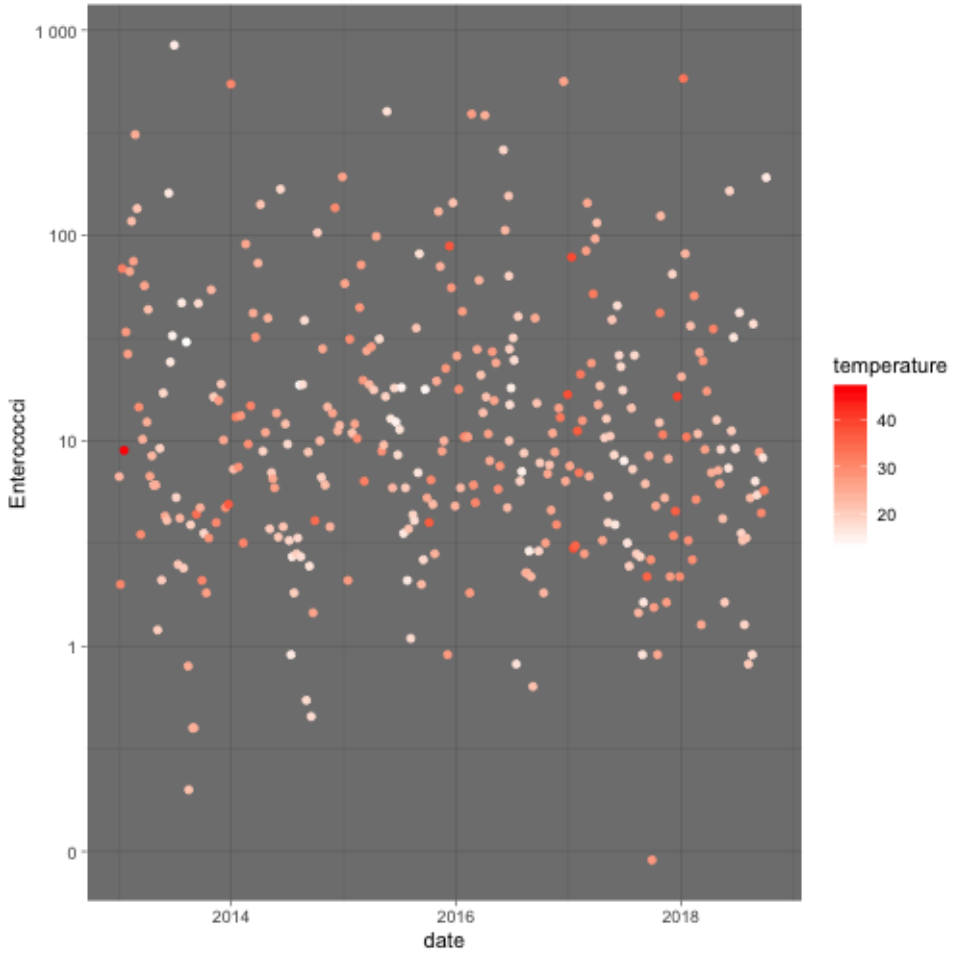
- `scale_*_brewer` uses the ColorBrewer palettes
- `scale_*_gradient` uses gradients
- `scale_*_distill` uses the ColorBrewer palettes, for continuous outcomes

> 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

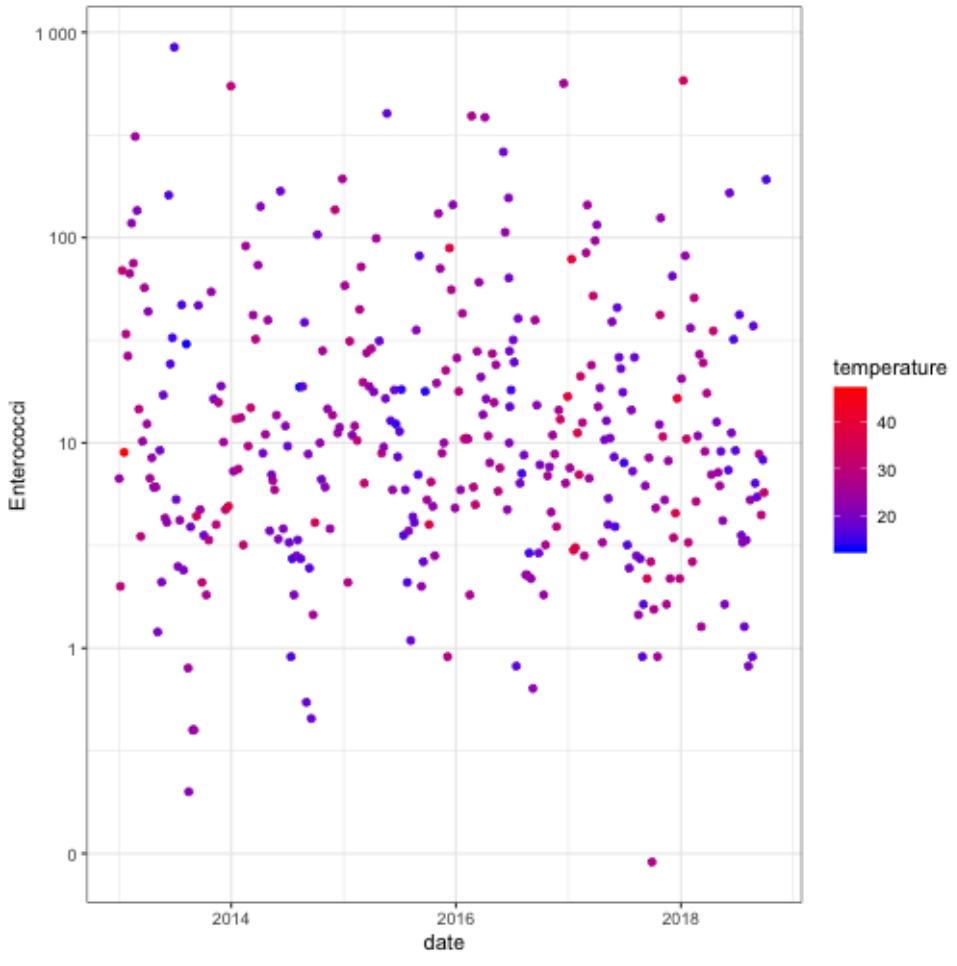
```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci,
      color = temperature)) +
  geom_point() +
  scale_y_log10(name = 'Enterococci',
                label = scales::number_format(digits=
```



```
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()
```

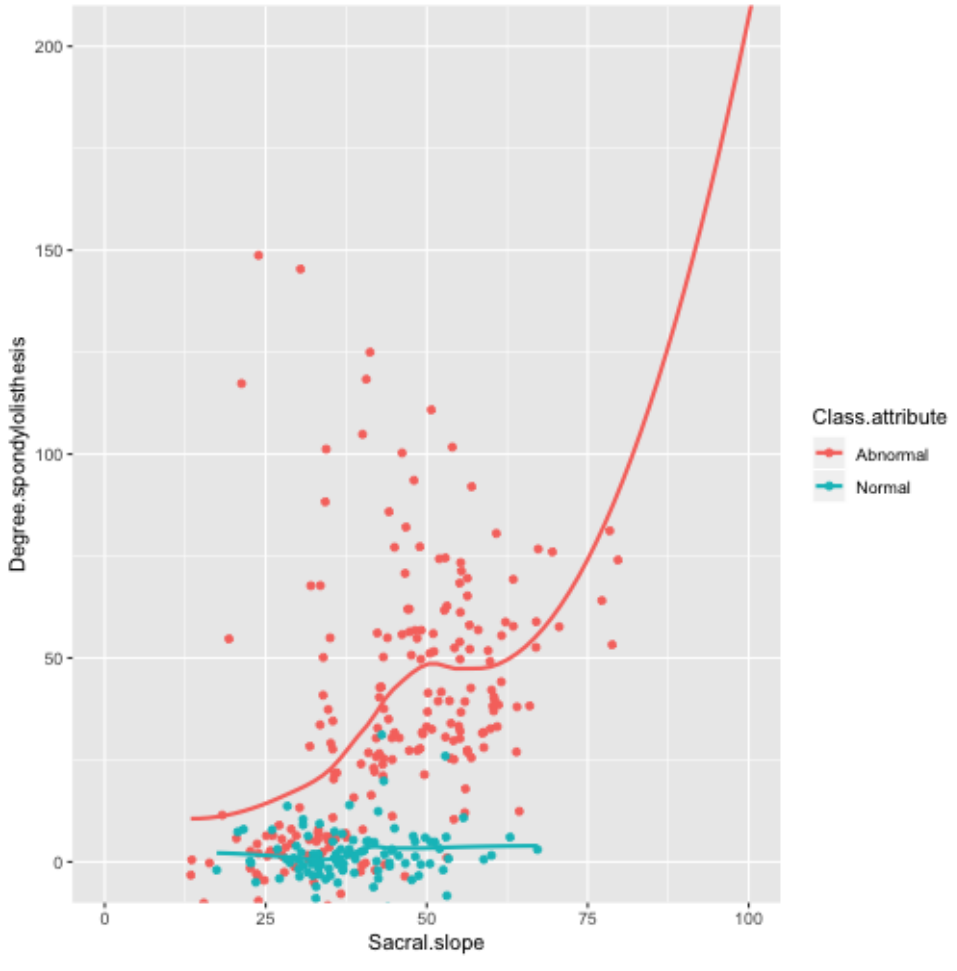


```
ggplot(
  data = beaches,
  aes(x = date, y = enterococci,
      color = temperature)) +
  geom_point() +
  scale_y_log10(name = 'Enterococci',
                label = scales::number_format(digits=
  scale_color_gradient(low = 'blue', high='red') +
  theme_bw()
```



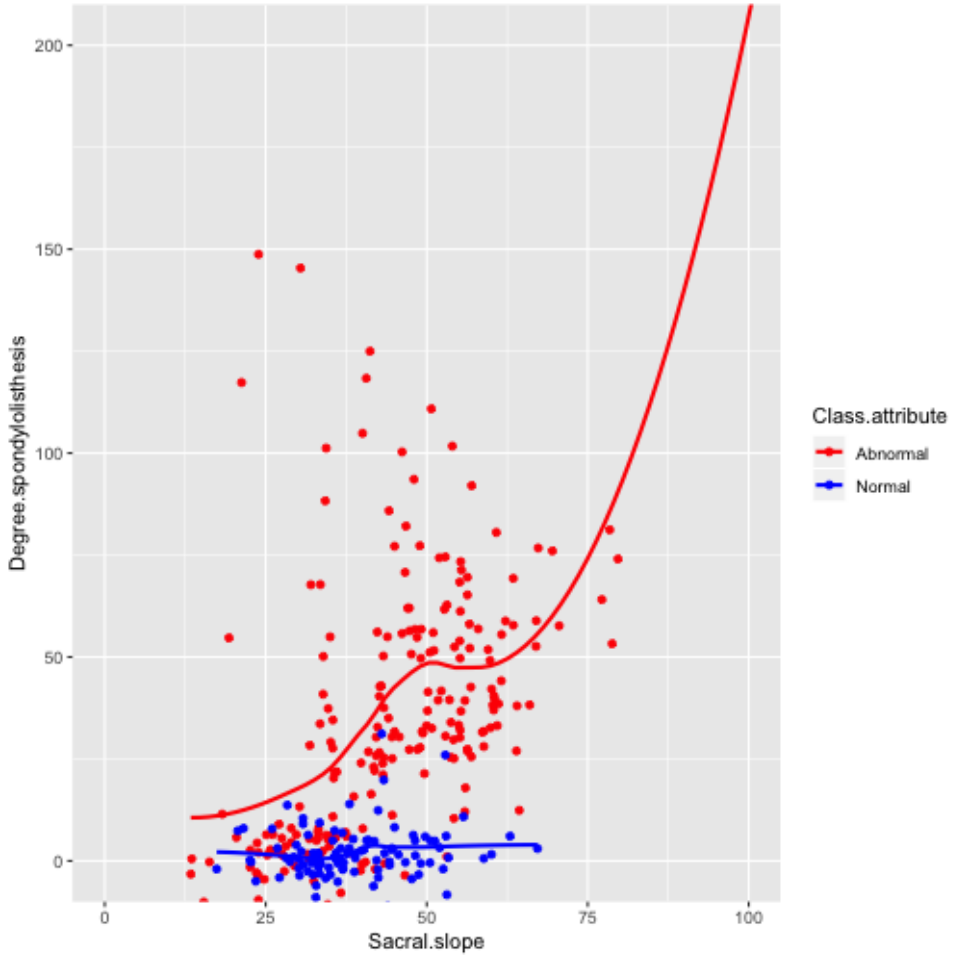
Specifying colors

```
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

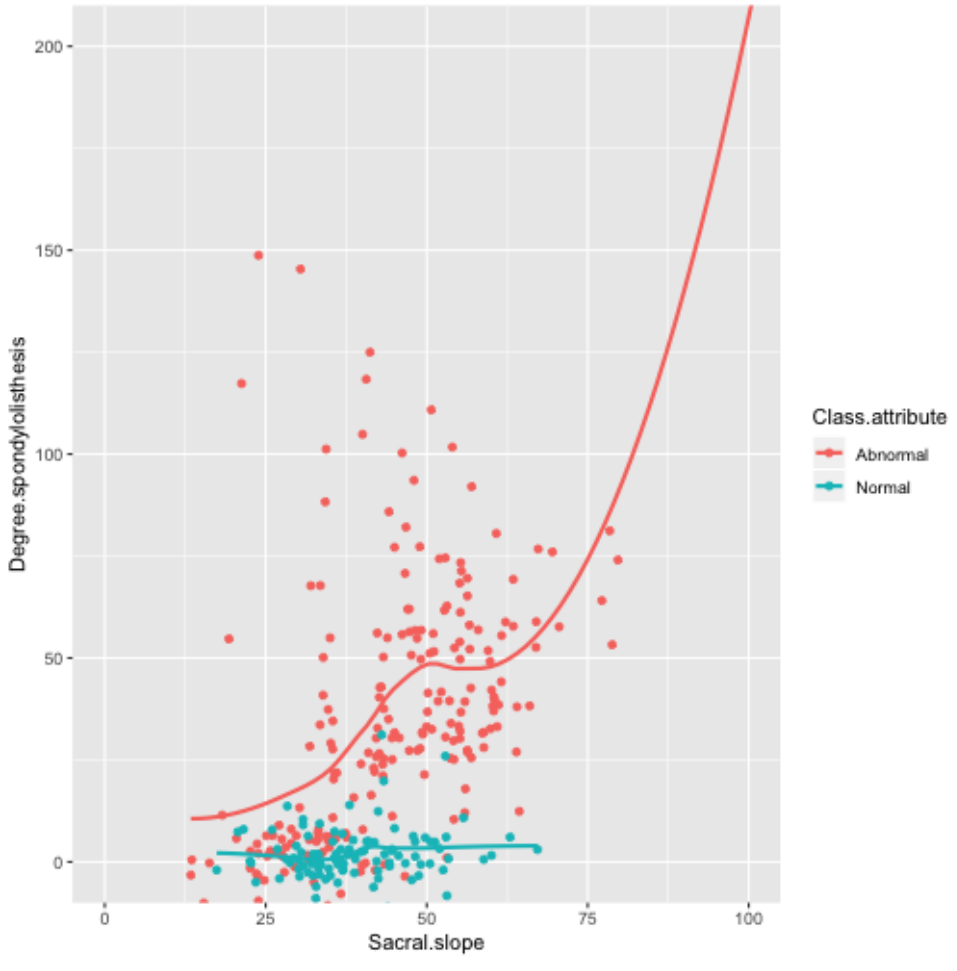
ggplot comes with

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

Themes

Create your own

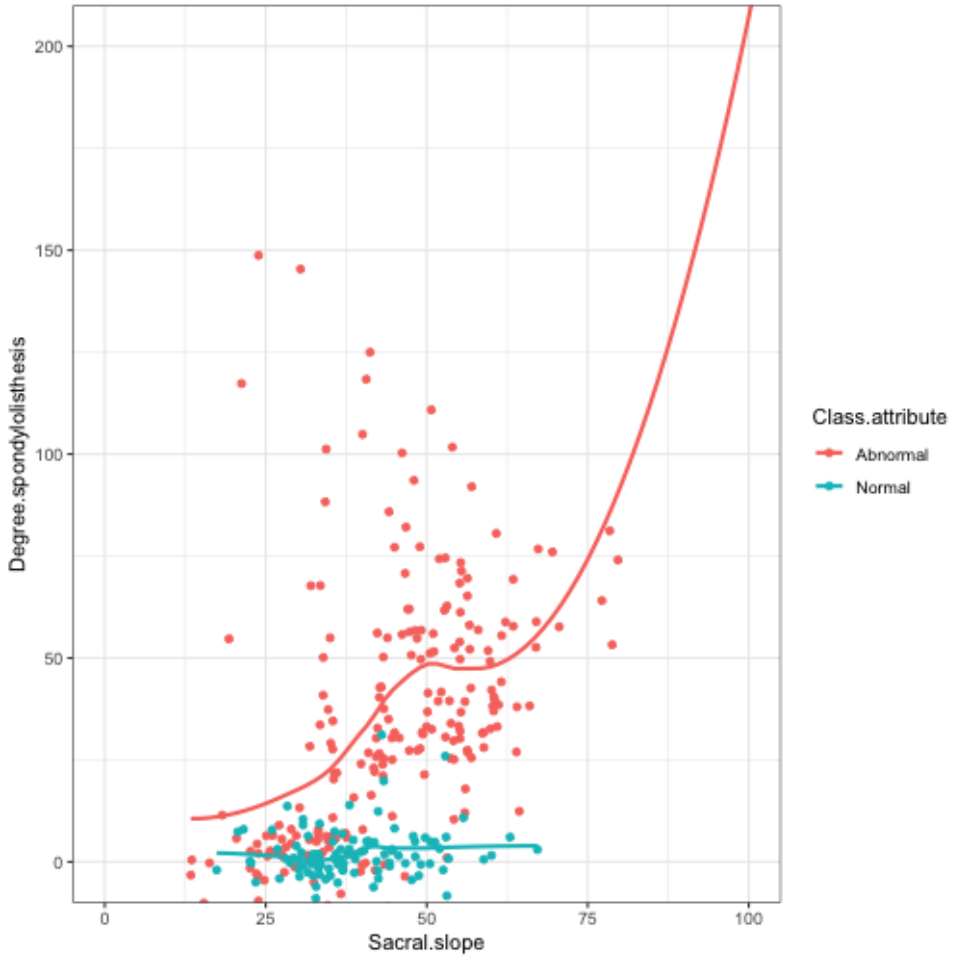
```
ggplot(
  data = dat_spine,
  aes(x = Sacral.slope, y = Degree.spondylolisthesis,
      color = Class.attribute)) +
  geom_point() +
  geom_smooth(se = F) +
  coord_cartesian(xlim = c(0, 100),
                  ylim = c(0,200))
```



Themes

Create your own

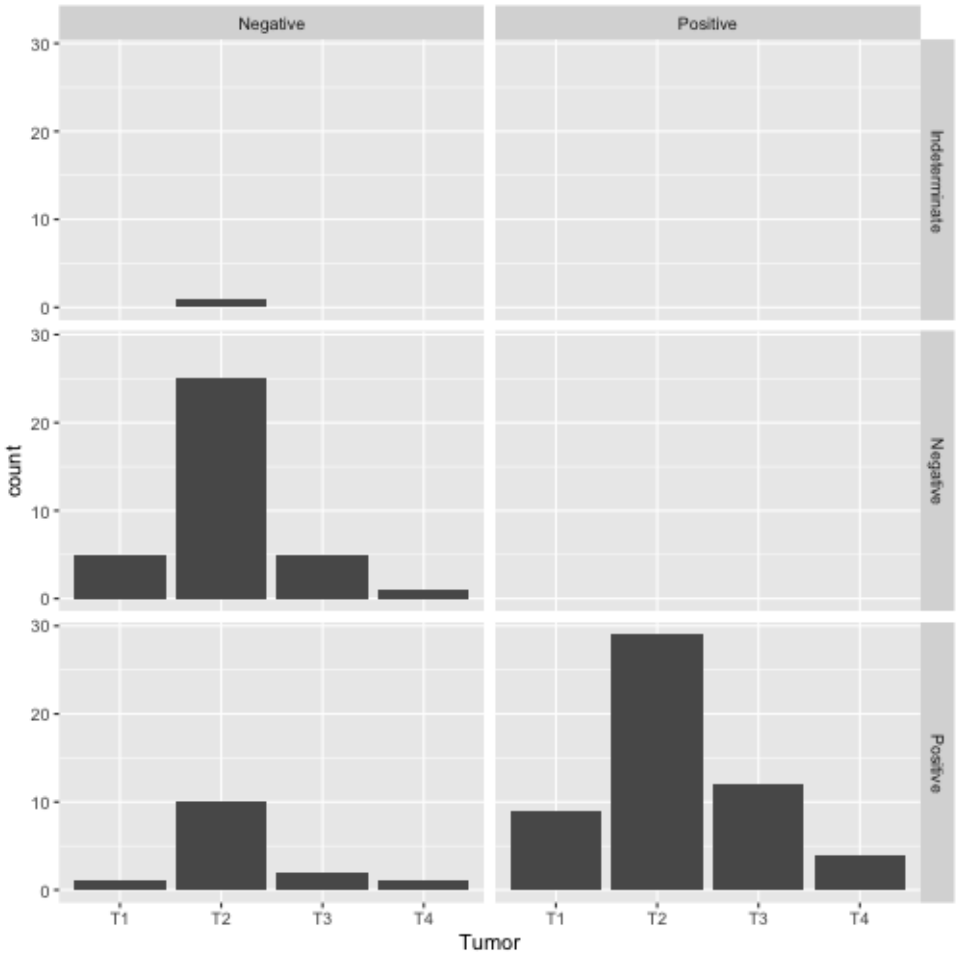
```
my_theme <- function(){
  theme_bw()
}

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

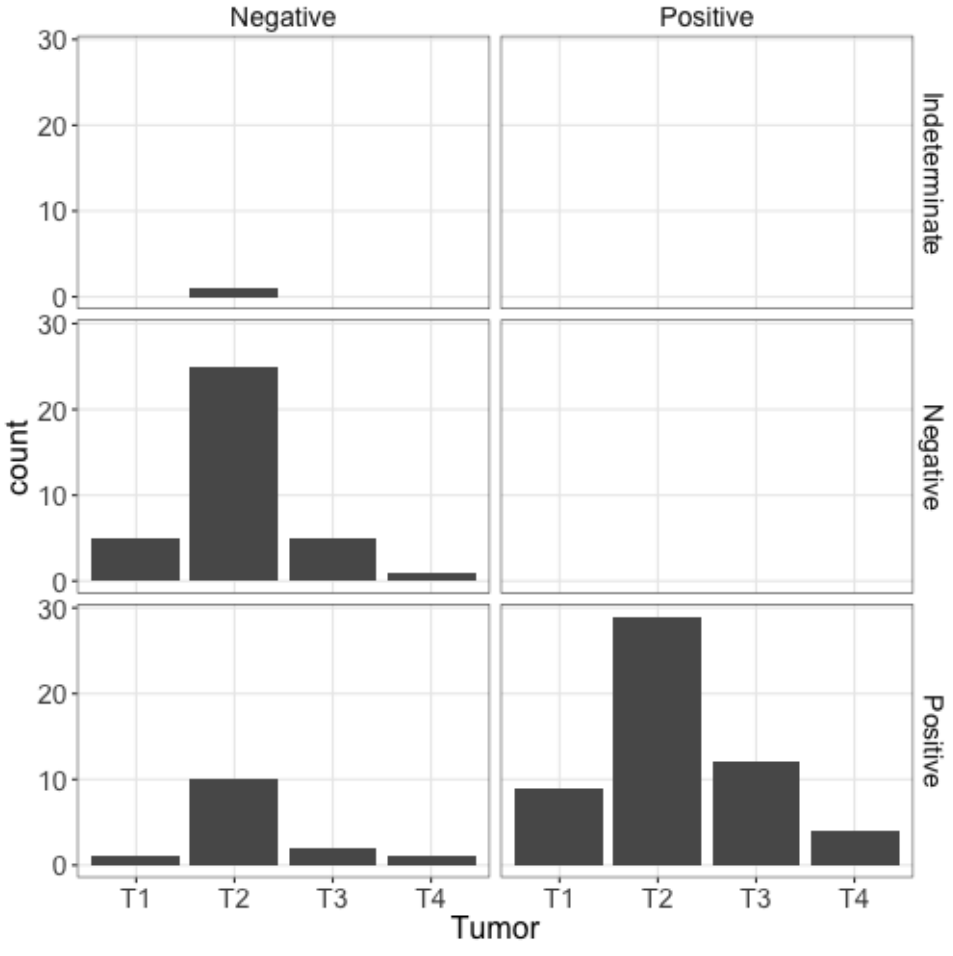
```
my_theme <- function(){
  theme_bw() +
  theme(axis.text = element_text(size = 14),
        axis.title = element_text(size = 16),
        panel.grid.minor = element_blank(),
        strip.text = element_text(size=14),
        strip.background = element_blank())
}

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()
```



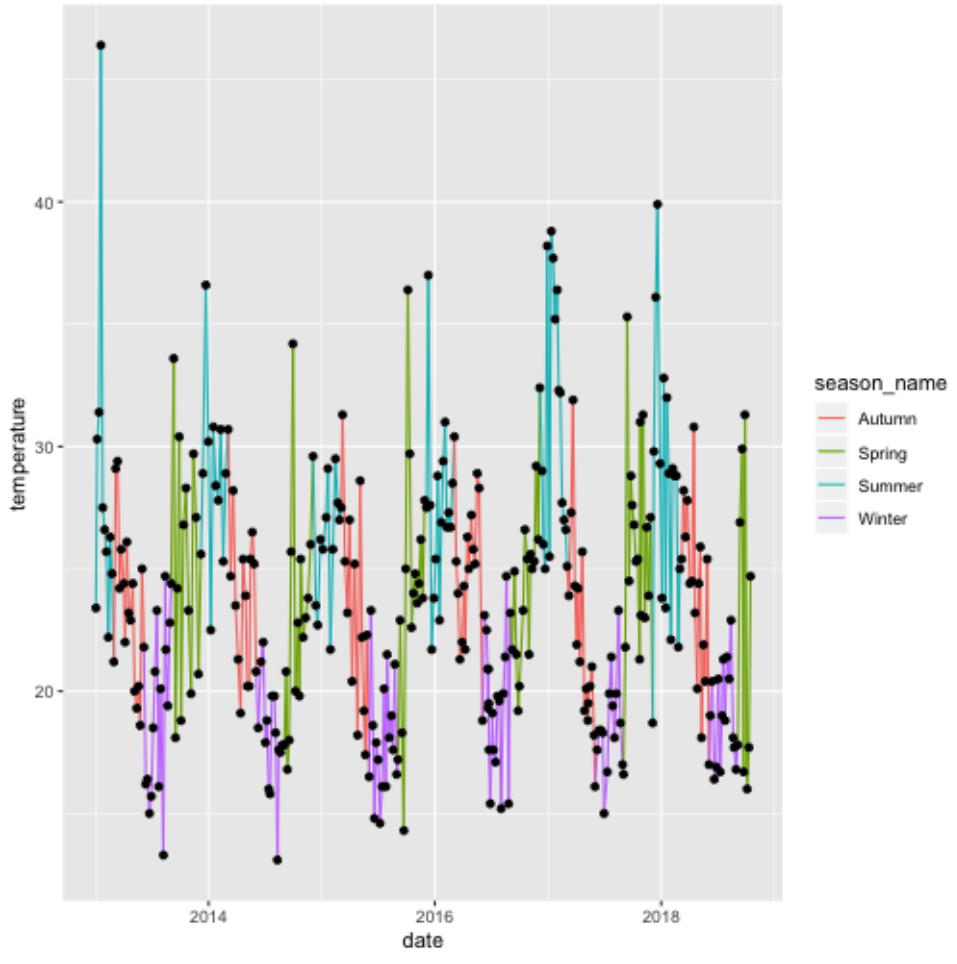
Animations

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)
```