Visualizing data

Abhijit Dasgupta

Fall, 2019

Before we start

Some functions I found particularly useful from your solutions to HW 3:

- `readr::parse_number`
- `dplyr::recode`
- `tidyr::gather(..., na.rm=T)`
- `readxl::read_excel(...., na="NA")`

> `recode` just switches one value for another. `caes_when` actually evaluates boolean expressions to decide on a new value

A basic function

```
name <- function(variables) {

}
```

- `variables` represents the inputs to the function. Several inputs can be separated by commas. Options are also inputs in a greater sense
- The part between `{}` represents the recipe, i.e. the R code needed to execute the function
- You have to give the function a name (here, `name`)
- The `function` keyword is non-negotiable

A basic function

```
my_mean <- function(x){
  out <- sum(x, na.rm=T)/length(x)
  return(out)
}
```

- `x` is a place holder for whatever object (here, a vector of numbers) you want as input
- We compute the sum of the non-missing values of x, divide by the number of numbers in x and create the mean. We then `return` that value as output of the function.

Technically this is wrong. we need to divide by `sum(!is.na(x))` and not `length(x)`. Why?

Why functions

Functions encapsulate repeated tasks. All the functions you use from packages are meant to be re-used. Your functions are written to be re-used too.

Where do we use functions in the tidyverse pipes?

- `mutate_at`, `mutate_if`, `mutate_all` all require some function that is repeated over different columns of the data
- You can use functions to transform variables in the `mutate` function
- You can use a function to define some complex condition for the `filter` function

Further reading

Chapter 19 of the R4DS book

Function syntax and arguments

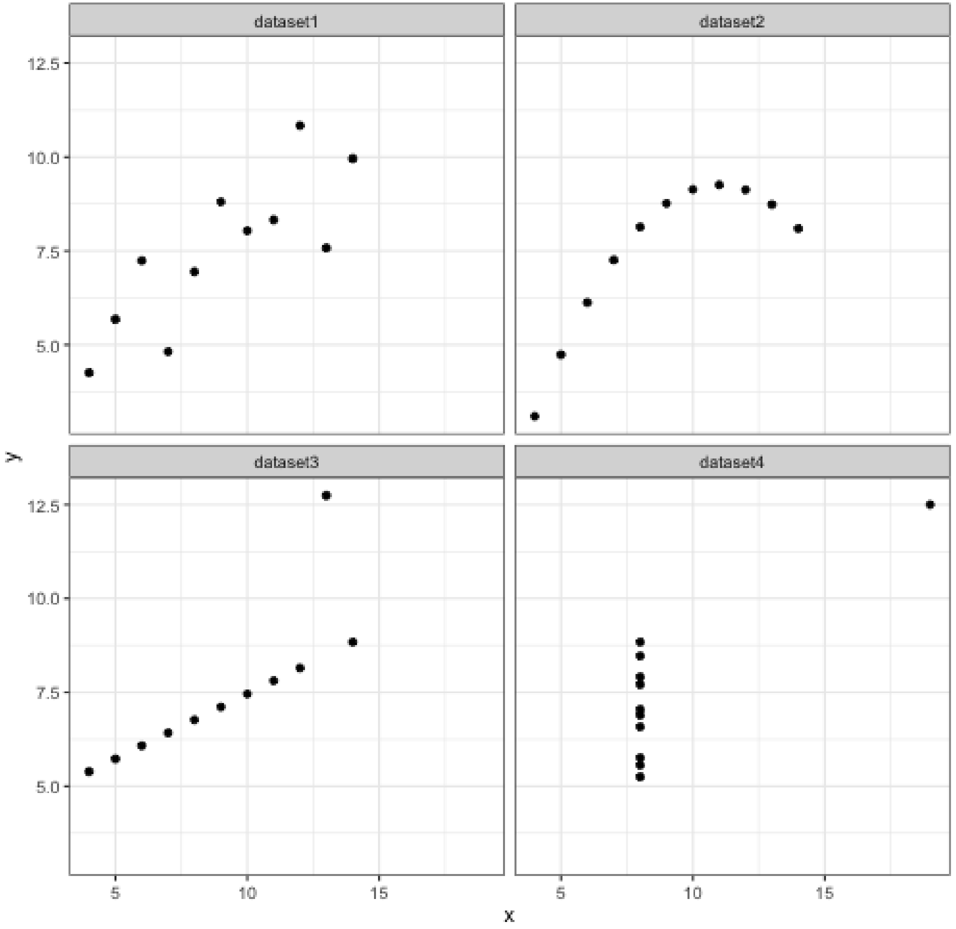
The main way you can figure out the syntax for a function is from its help documentation

```
help(case_when)
```

```
?parse_number
```

Why visualize data?

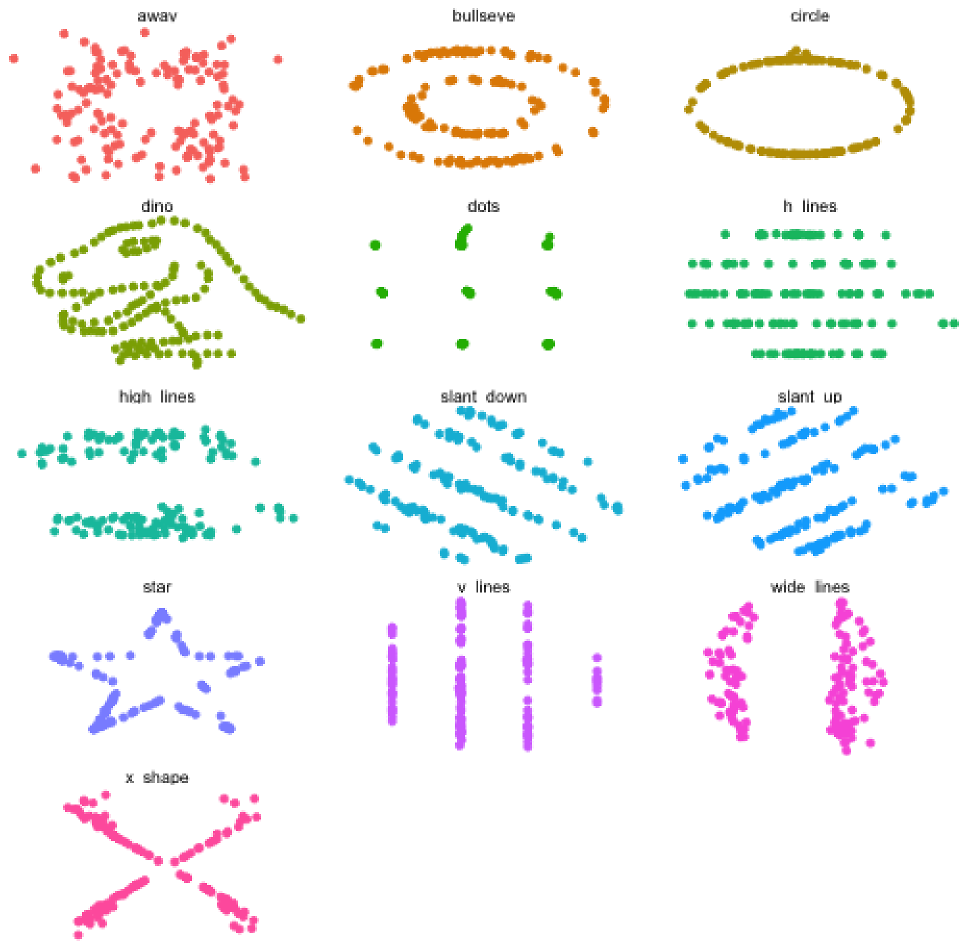
Anscombe's data



Anscombe, 1973

Statistic	Value
mean(x)	9
mean(y)	7.5
var(x)	11
var(y)	4.13
cor(x,y)	0.82

The DataSaurus dozen



Matejka & Fitzmaurice, 2017

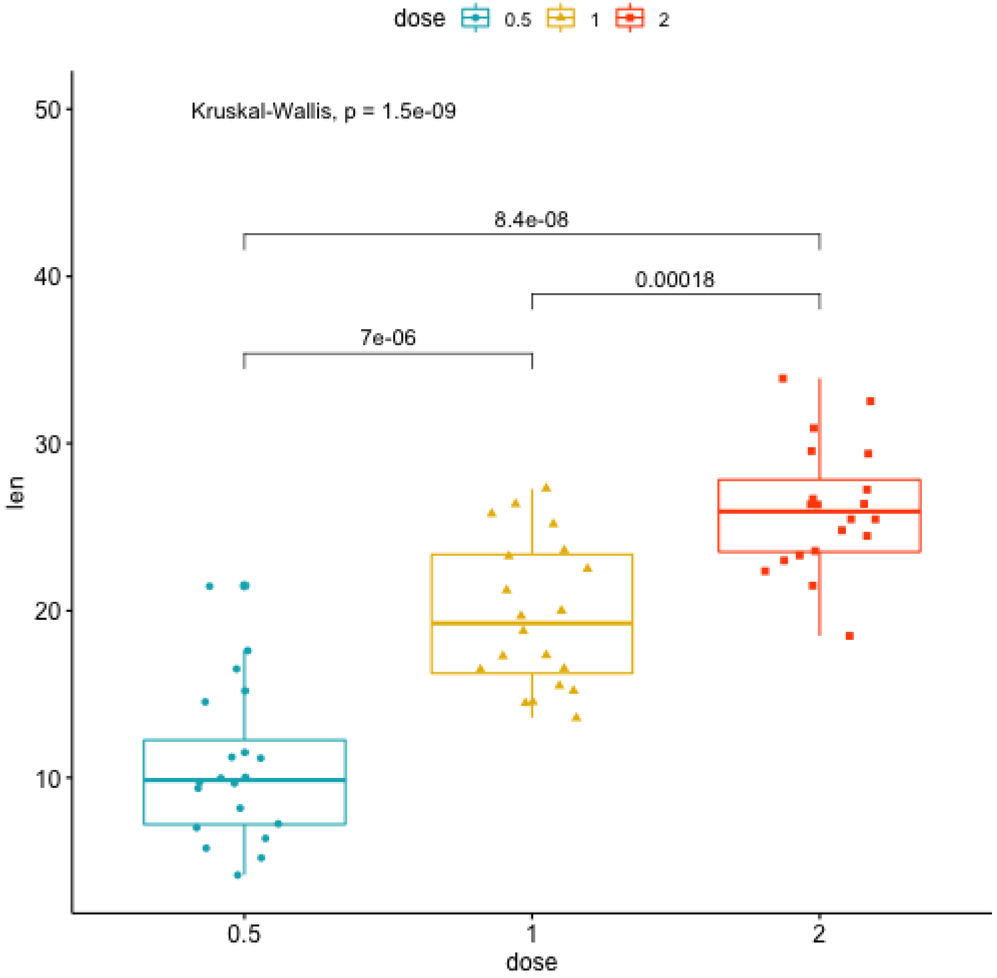
Statistic	Value
mean(x)	54.3
mean(y)	47.8
var(x)	281
var(y)	725
cor(x,y)	-0.07

Bottom line

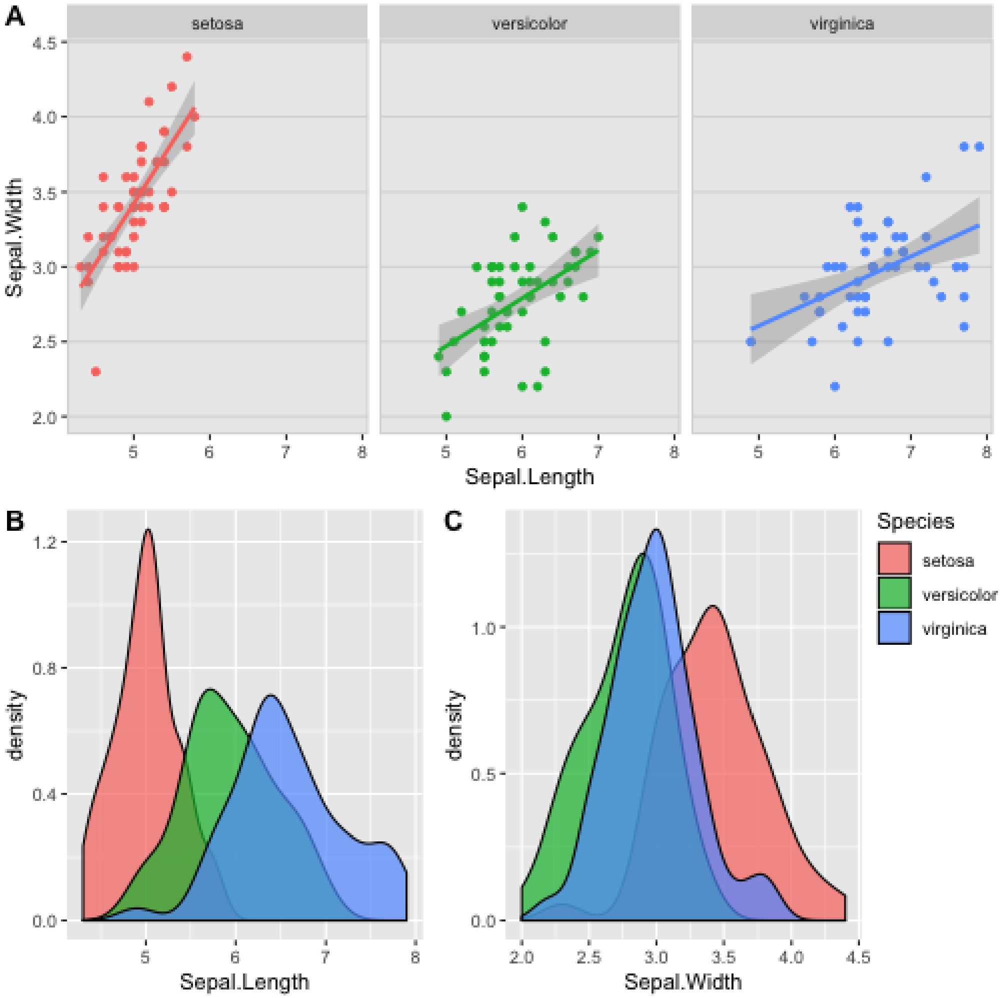
- Summary statistics cannot always distinguish datasets
- Take advantage of humans' ability to visually recognize and remember patterns
- Find discrepancies in the data more easily

Some examples

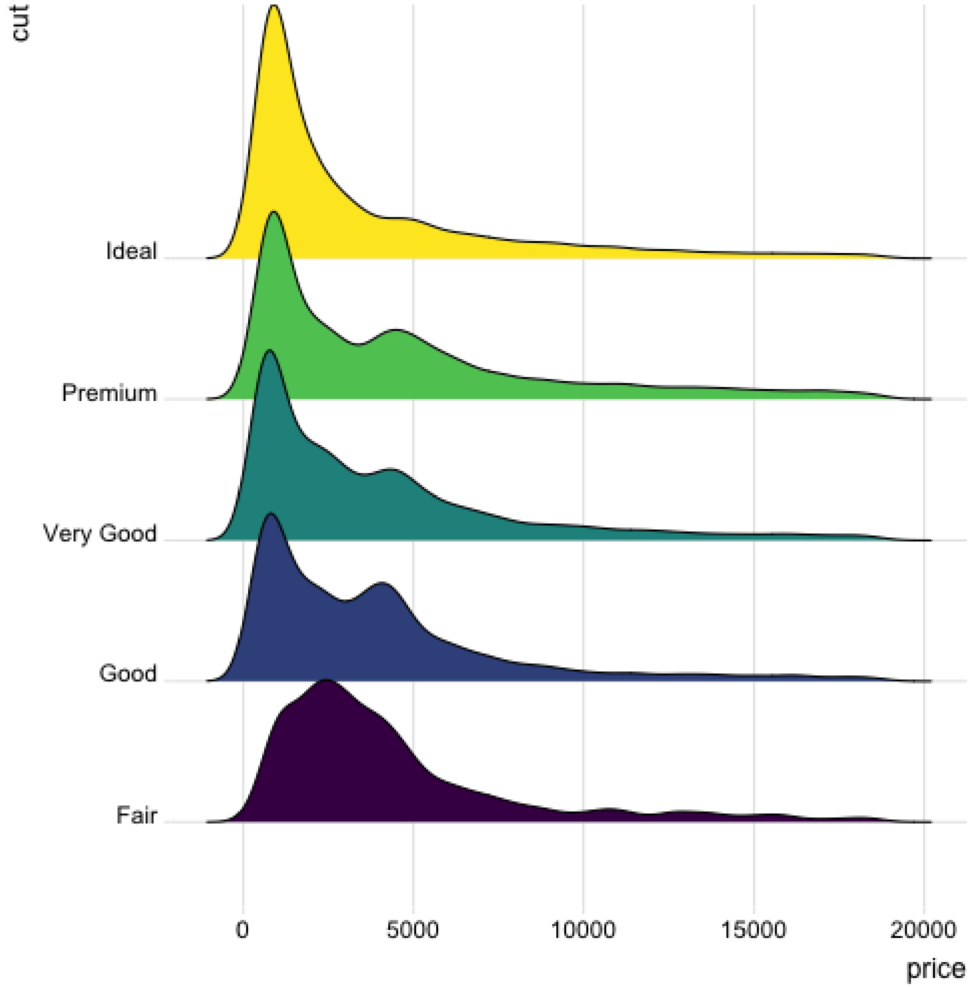
Gallery


dose
0.5
1
2
Kruskal-Wallis, p = 1.5e-09
8.4e-08
0.00018
7e-06
50
40
30
20
10
len
0.5
1
2
dose

Gallery


A
setosa
versicolor
virginica
4.5
4.0
3.5
3.0
2.5
2.0
Sepal.Width
5
6
7
8
Sepal.Length
B
1.2
0.8
0.4
0.0
density
Sepal.Length
C
1.0
0.5
0.0
density
2.0
2.5
3.0
3.5
4.0
4.5
Sepal.Width
Species
setosa
versicolor
virginica

Gallery


cut
Ideal
Premium
Very Good
Good
Fair
0
5000
10000
15000
20000
price

Gallery

Manhattan plot

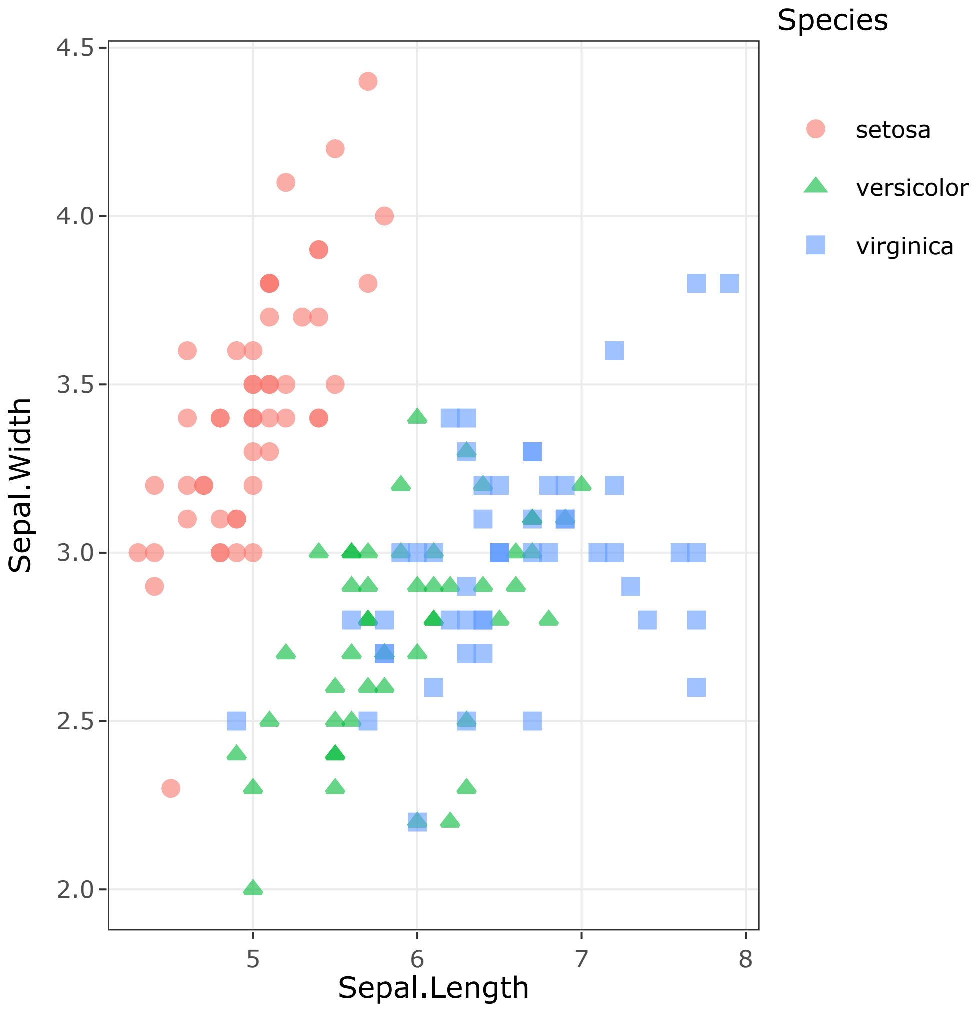
Gallery

Circular Manhattan plot

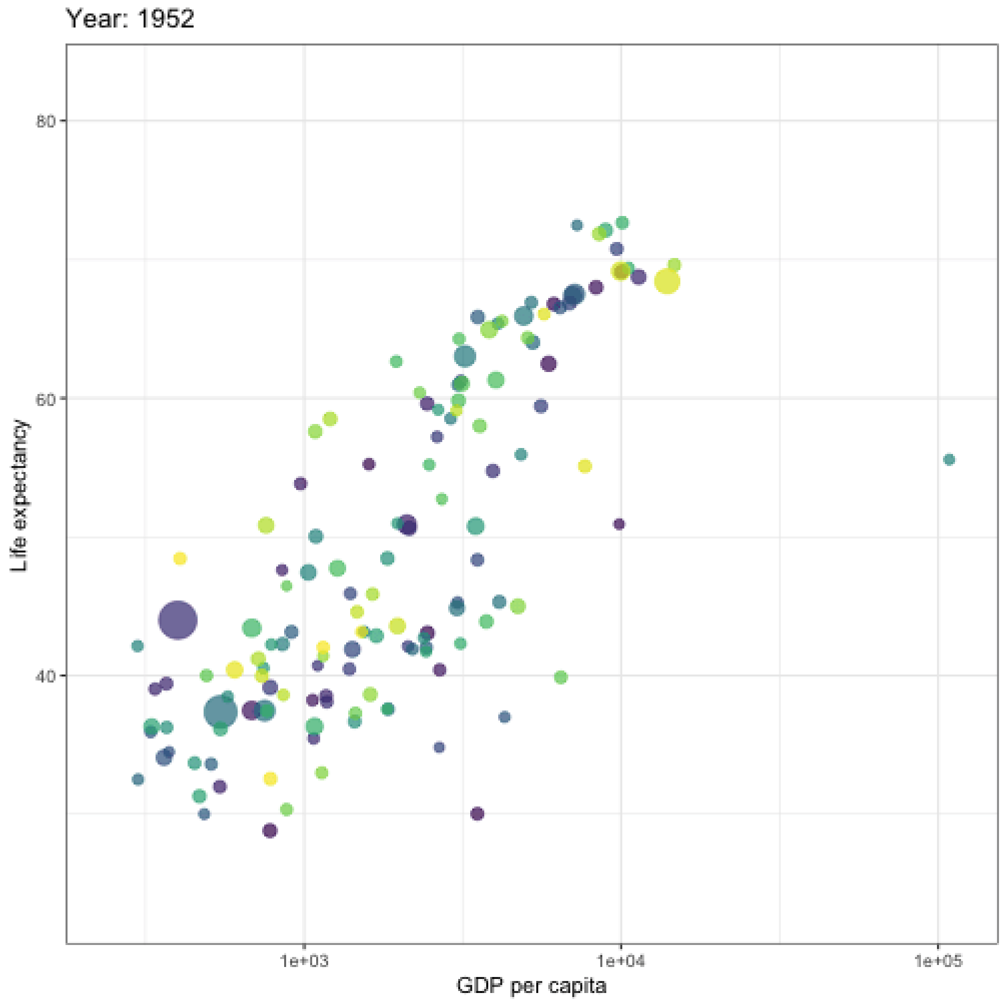
Gallery

Maps

Interactive graphs



Animated graphs



Data visualization with ggplot2

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

Introduction to ggplot2

The `ggplot2` package is a very flexible and (to me) intuitive way of visualizing data. It is based on the concept of layering elements on a canvas.

> This idea of layering graphics on a canvas is, to me, a nice way of building graphs

You need:

- A `data.frame` object
- *Aesthetic mappings* (aes) to say what data is used for what purpose in the viz
  - x- and y-direction
  - shapes, colors, lines
- A *geometry object* (geom) to say what to draw
  - You can "layer" geoms on each other to build plots

Introduction to ggplot2

`ggplot` used pipes before pipes were a thing.

However, it uses the + symbol for piping rather than the `%>%` operator, since it pre-dates the `tidyverse`

Introduction to ggplot2

```
library(ggplot2)
ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()
```

- A `data.frame` object: mtcars
- Aesthetic mapping:
    - x-axis: wt
    - y-axis: mpg
- Geometry:
    - geom_point: draw points

Introduction to ggplot2

```
library(ggplot2)
ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point()+ geom_smooth()
```

- A `data.frame` object: mtcars
- Aesthetic mapping:
    - x-axis: wt
    - y-axis: mpg
- Geometry:
    - geom_point: draw points
    - geom_smooth: Add a layer which draws a best-fitting line

A dataset

We will use the beaches dataset

```
library(tidyverse)
library(rio)
beaches <- import('data/sydneybeaches3.csv')
```

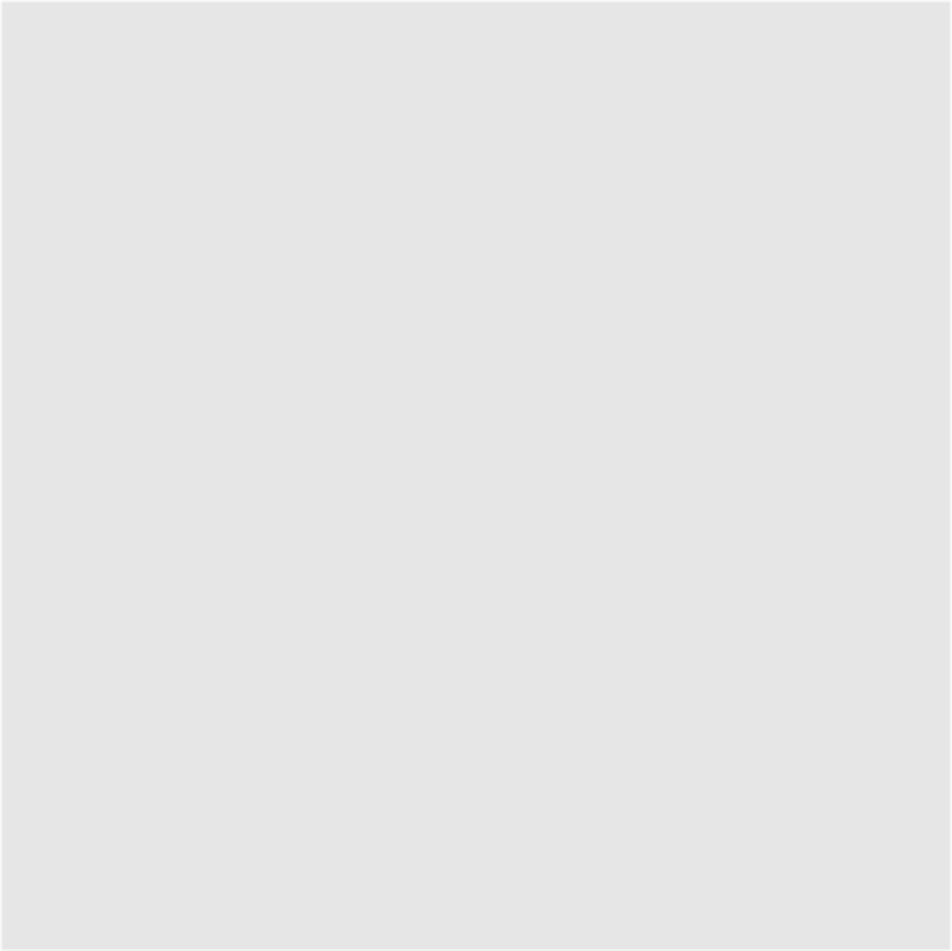
```
#>          date year month day season rainfall temperature enterococci
#>  1 2013-01-02 2013     1   2      1      0.0        23.4         6.7
#>  2 2013-01-06 2013     1   6      1      0.0        30.3         2.0
#>  3 2013-01-12 2013     1  12      1      0.0        31.4        69.1
#>  4 2013-01-18 2013     1  18      1      0.0        46.4         9.0
#>  5 2013-01-24 2013     1  24      1      0.0        27.5        33.9
#>  6 2013-01-30 2013     1  30      1      0.6        26.6        26.5
#>    day_num month_num month_name season_name
#>  1       2         1    January      Summer
#>  2       6         1    January      Summer
#>  3      12         1    January      Summer
#>  4      18         1    January      Summer
#>  5      24         1    January      Summer
#>  6      30         1    January      Summer
```

Credit: D. J. Navarro

Building a graph

Start with a blank canvas

```
ggplot()
```



Add a data set

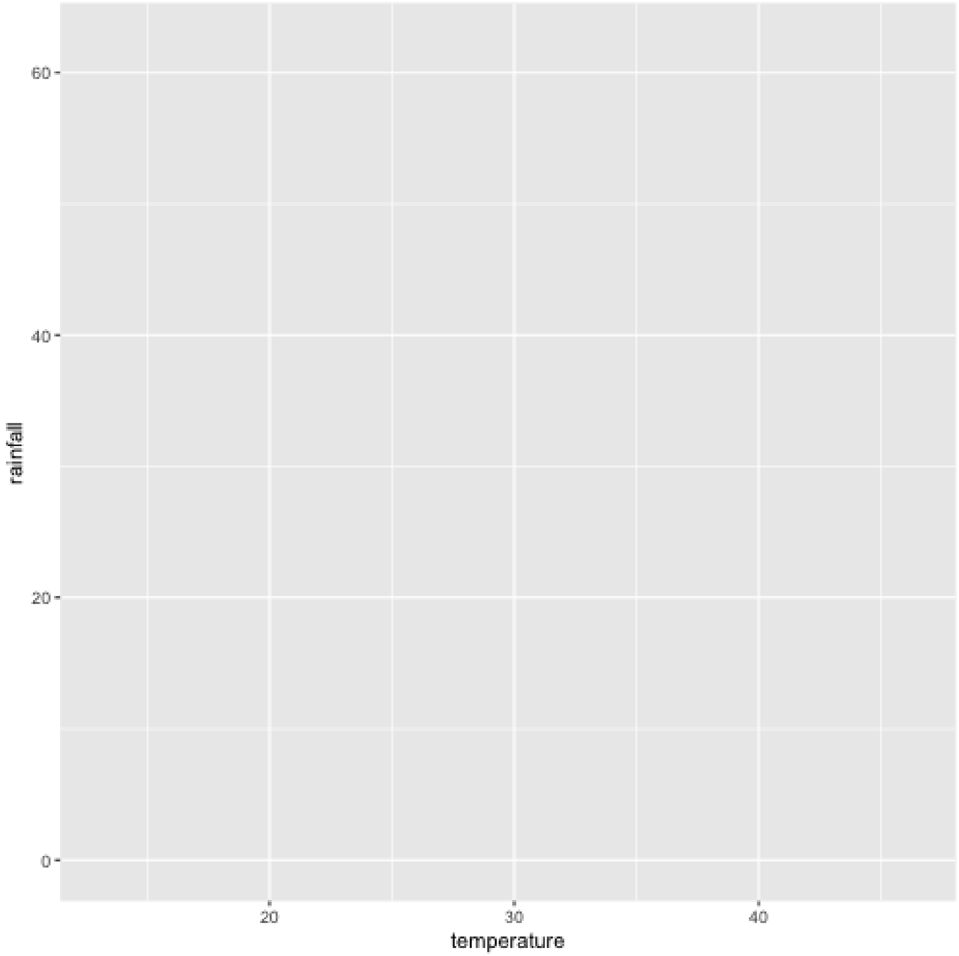
```
ggplot(
  data = beaches
)
```

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

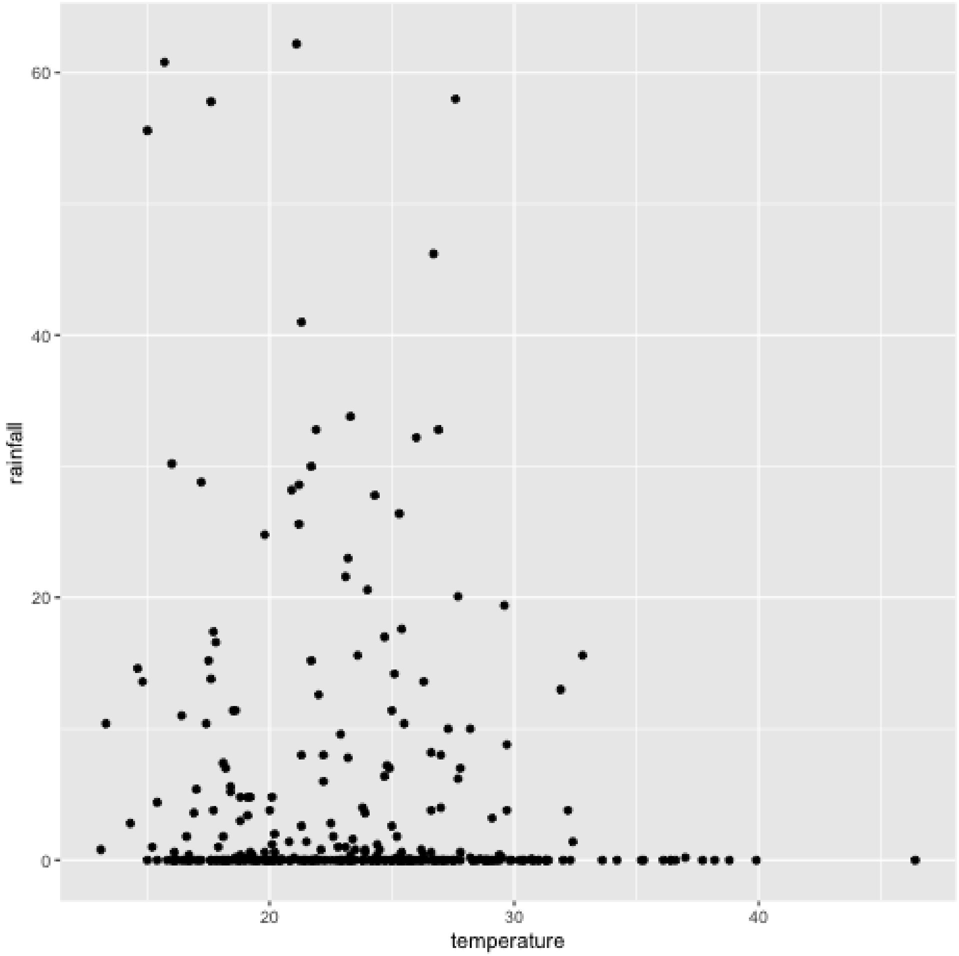


Add a geometry to draw

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

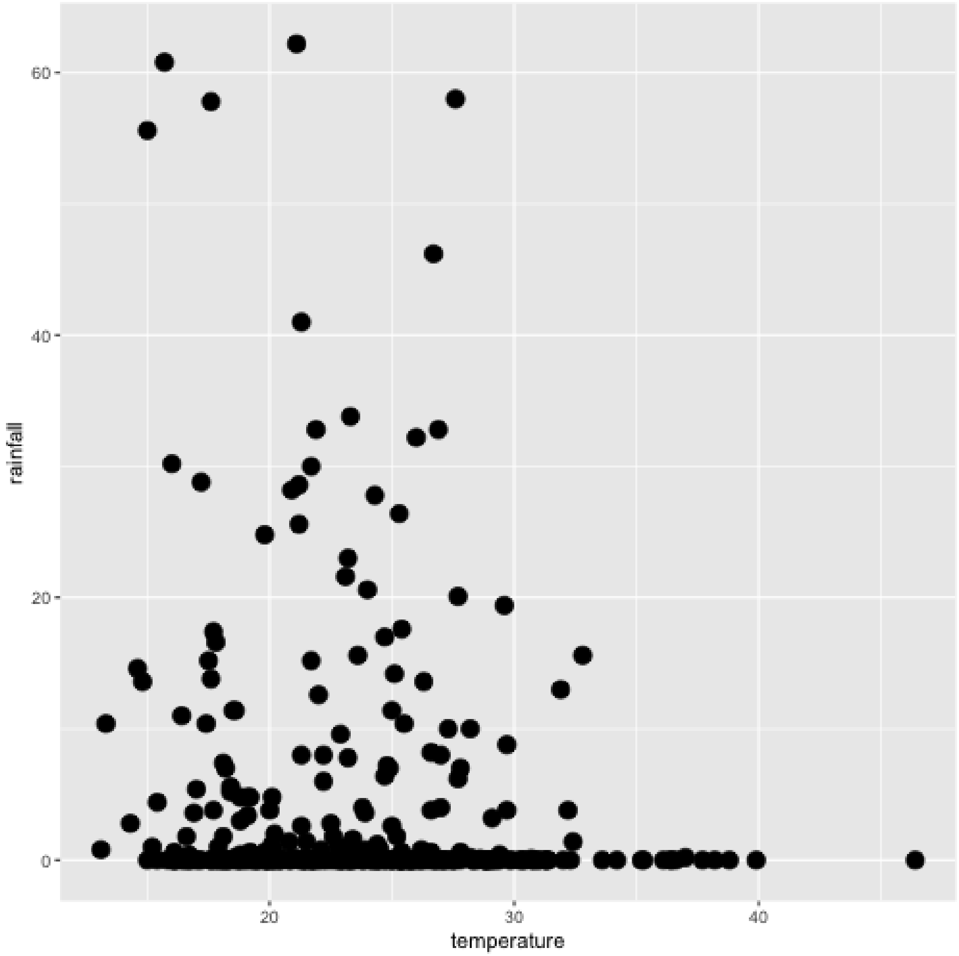
What to draw:

- Points, lines
- histogram, bars, pies



Add options for the geom

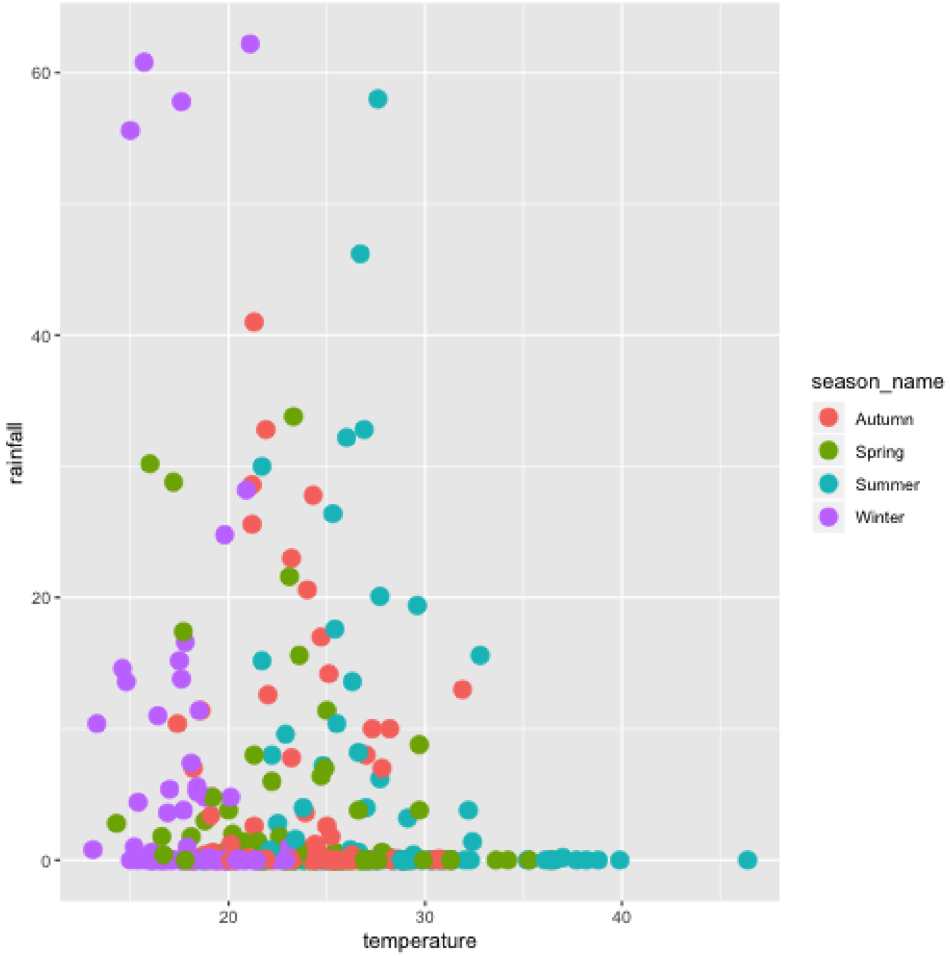
```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(size = 4)
```



Add a mapping to modify the geom

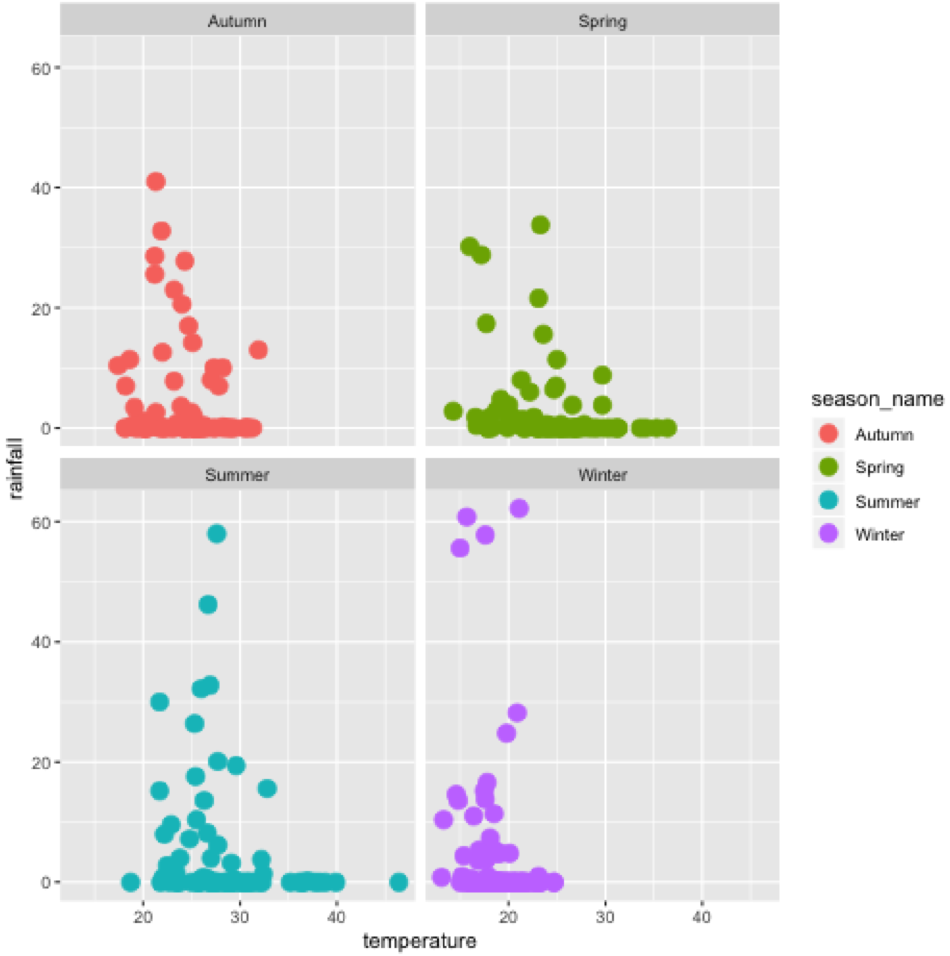
```
ggplot(
  data = beaches,
  mapping = aes(
    x = temperature,
    y = rainfall
  )
) +
  geom_point(
    mapping = aes(color = season_name),
    size = 4
  )
```

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



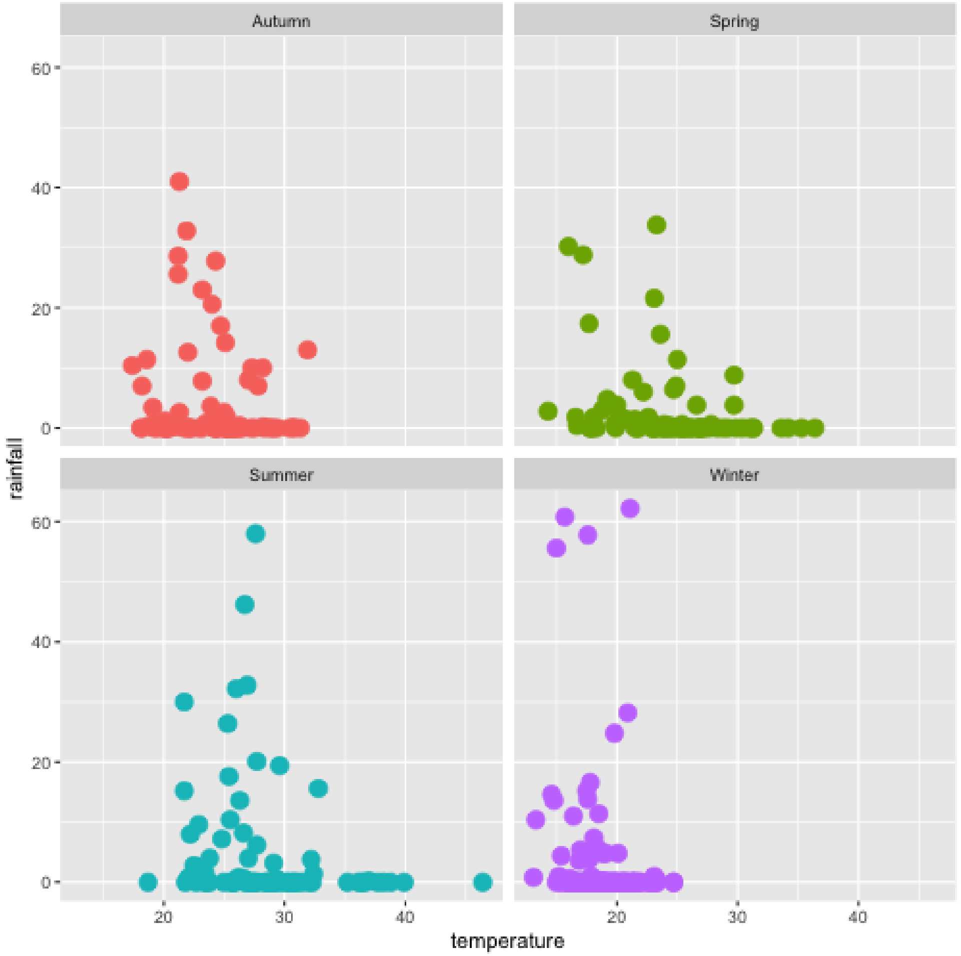
Split into facets

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



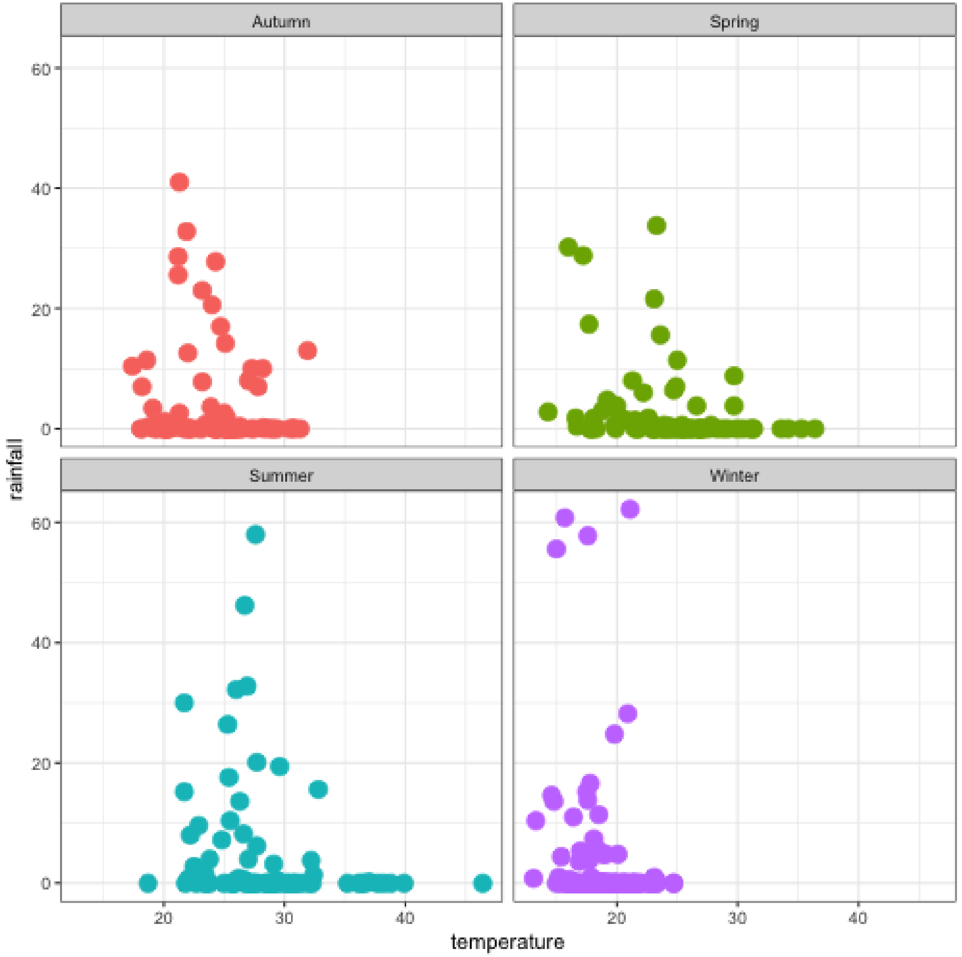
Remove the legend

```
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)
```



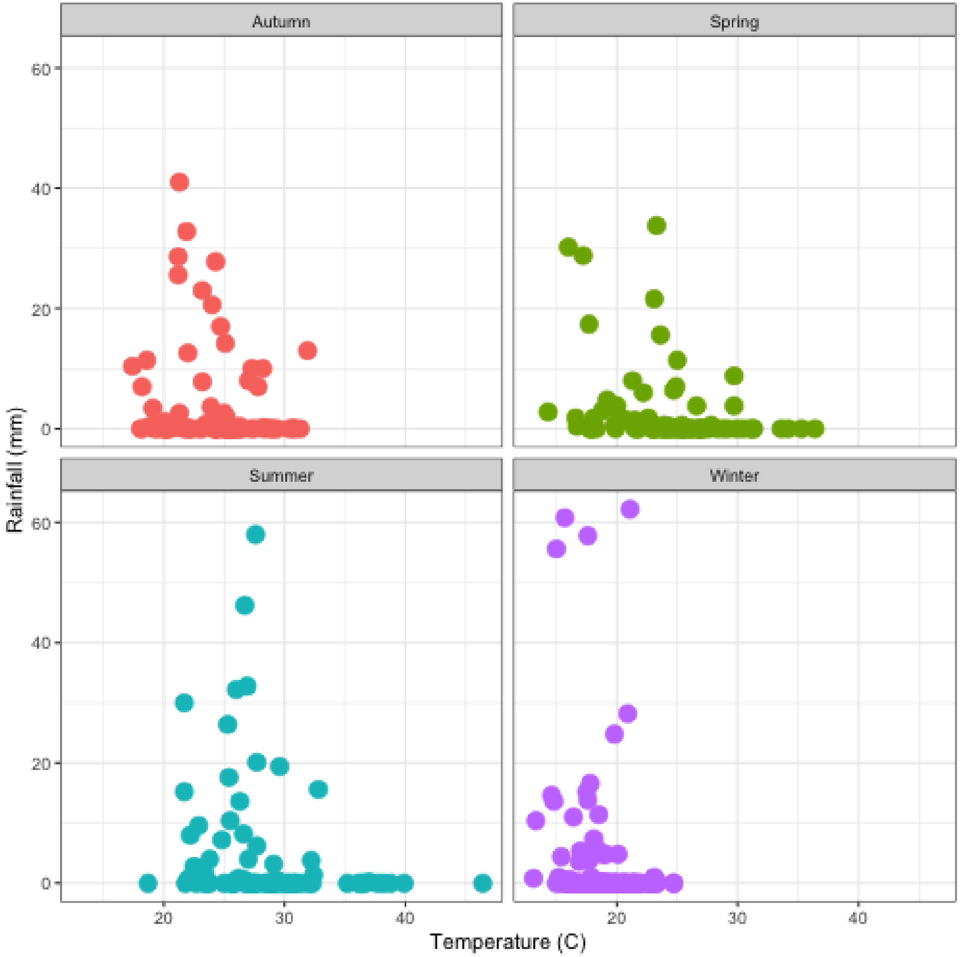
Change the background

```
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()
```



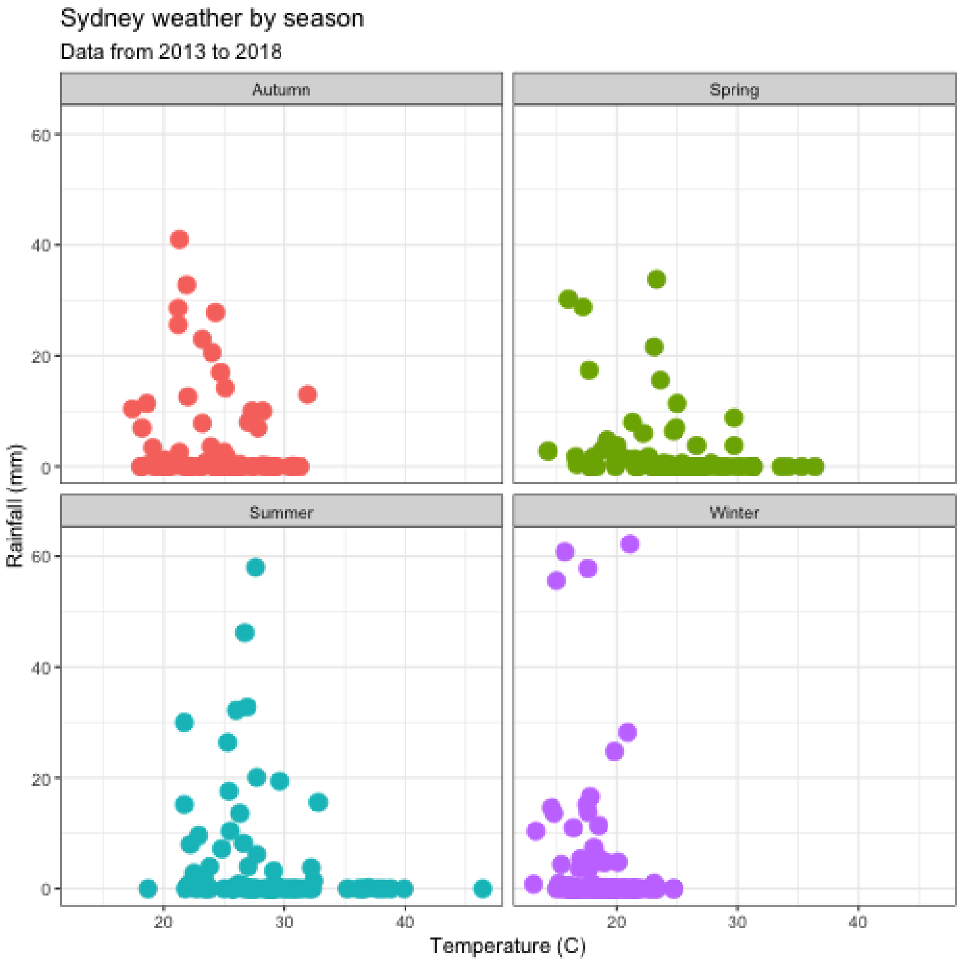
Update the labels

```
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")
```



The grammar

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

Peeking under the hood

If I write...

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

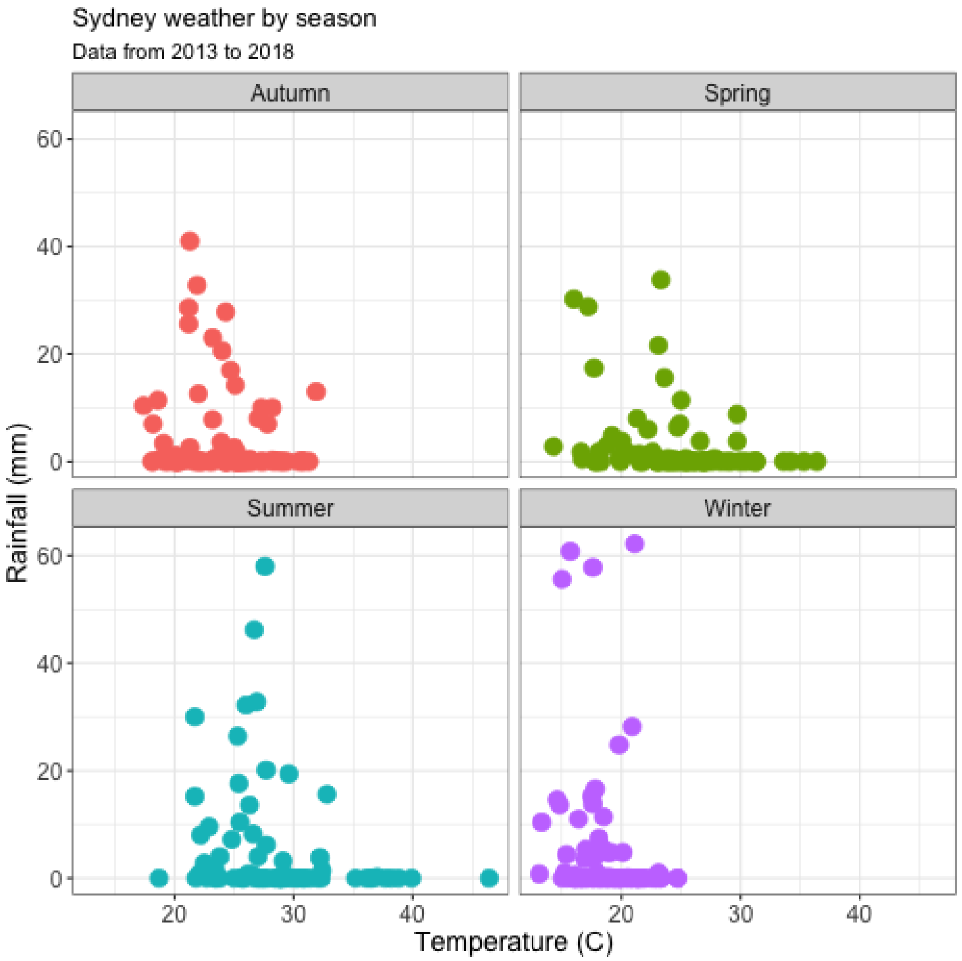
what's really run is ...

```
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

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))
```



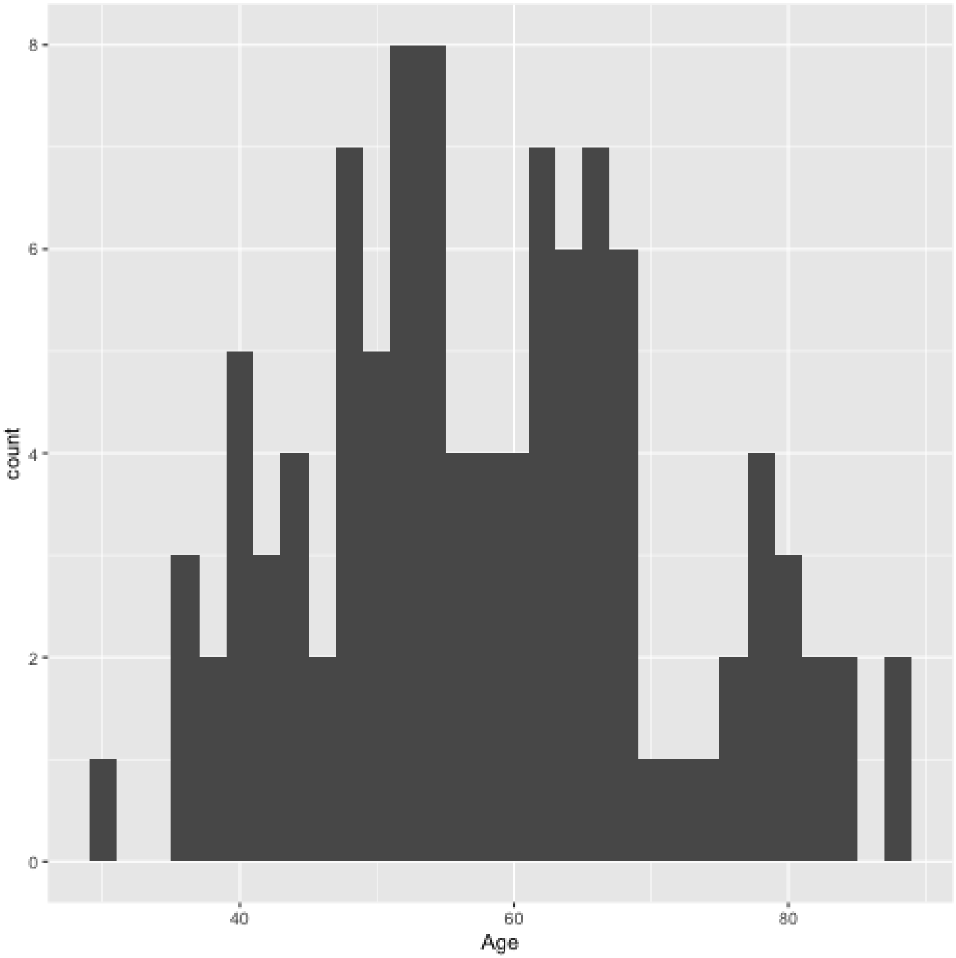
Using the BRCA data

We'll use the brca data developed during the homework. The RDS file is available here.

```
brca_clean <- readRDS('data/brca.rds')
brca_clean <- brca_clean %>%
  rename('Age' = 'Age.at.Initial.Pathologic.Diagnosis')
```

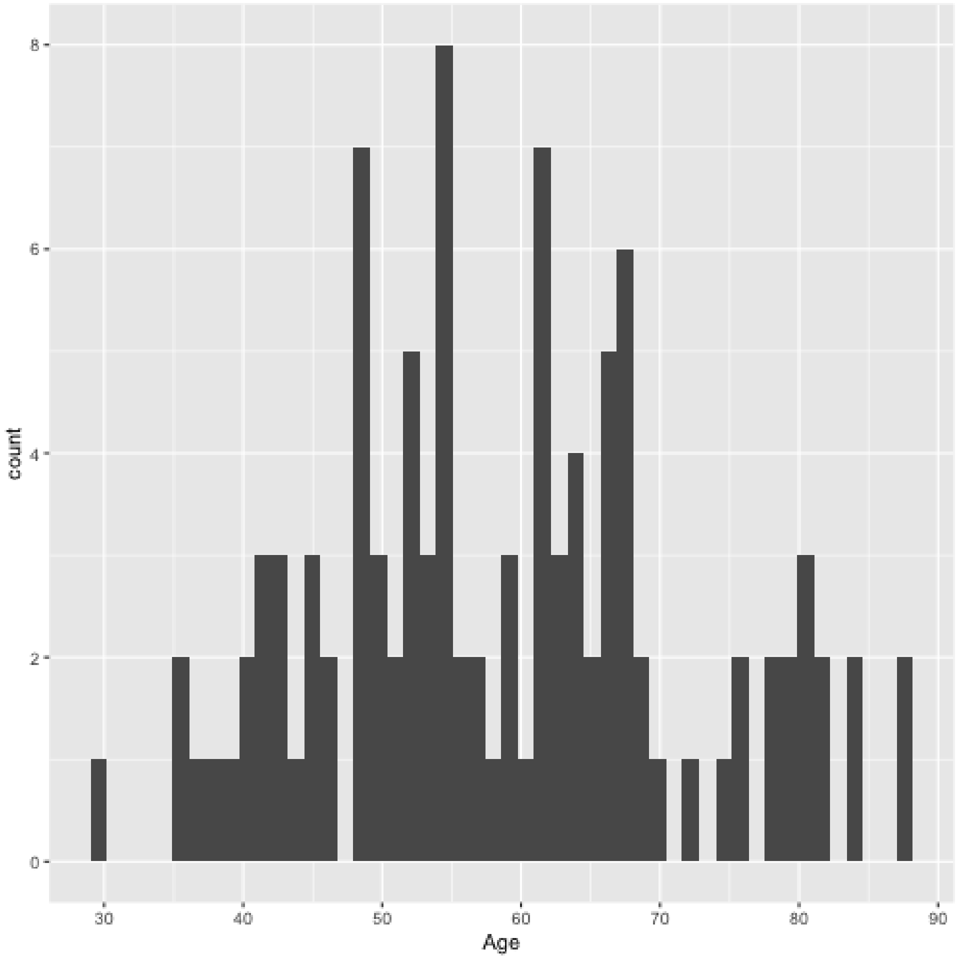
Univariate plots

```
ggplot(data=brca_clean,
       mapping = aes(x = Age)) +
  geom_histogram()
```



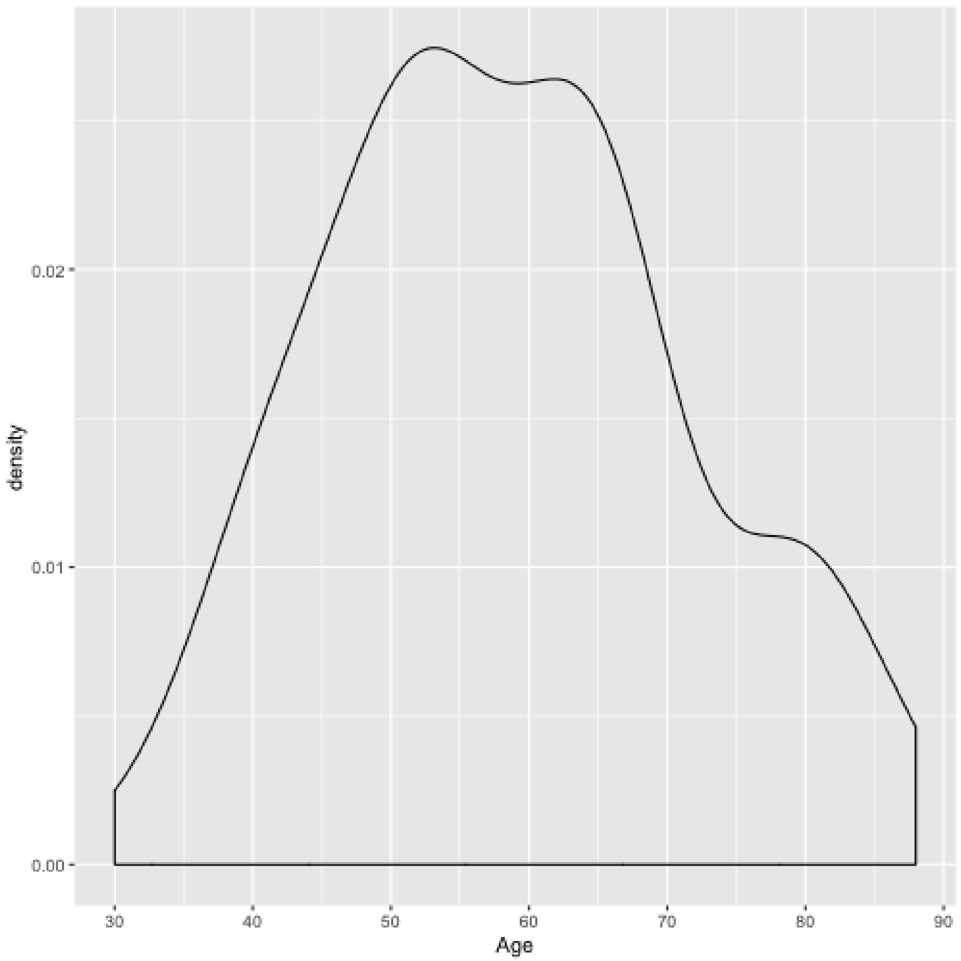
Univariate plots

```
ggplot(data=brca_clean,
       mapping = aes(x = Age)) +
  geom_histogram(bins=50)
```



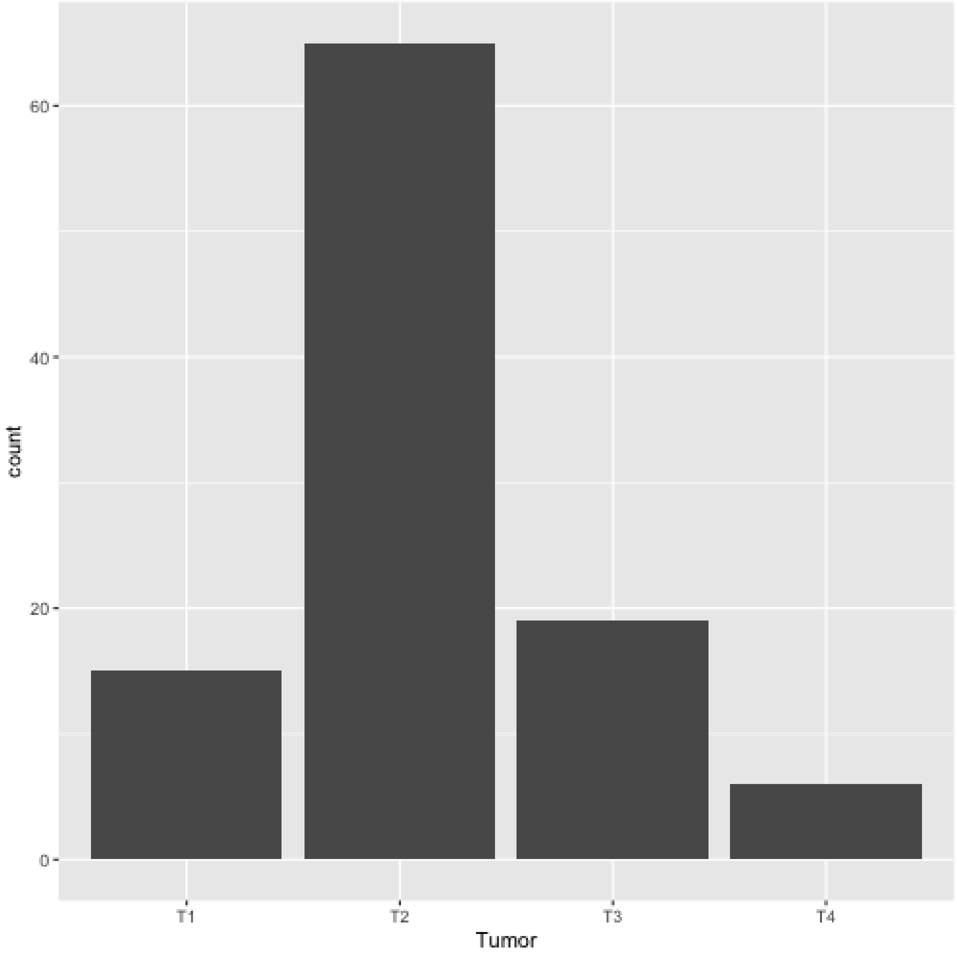
Univariate plots

```
ggplot(data=brca_clean,
       mapping = aes(x = Age)) +
  geom_density()
```



Univariate plots

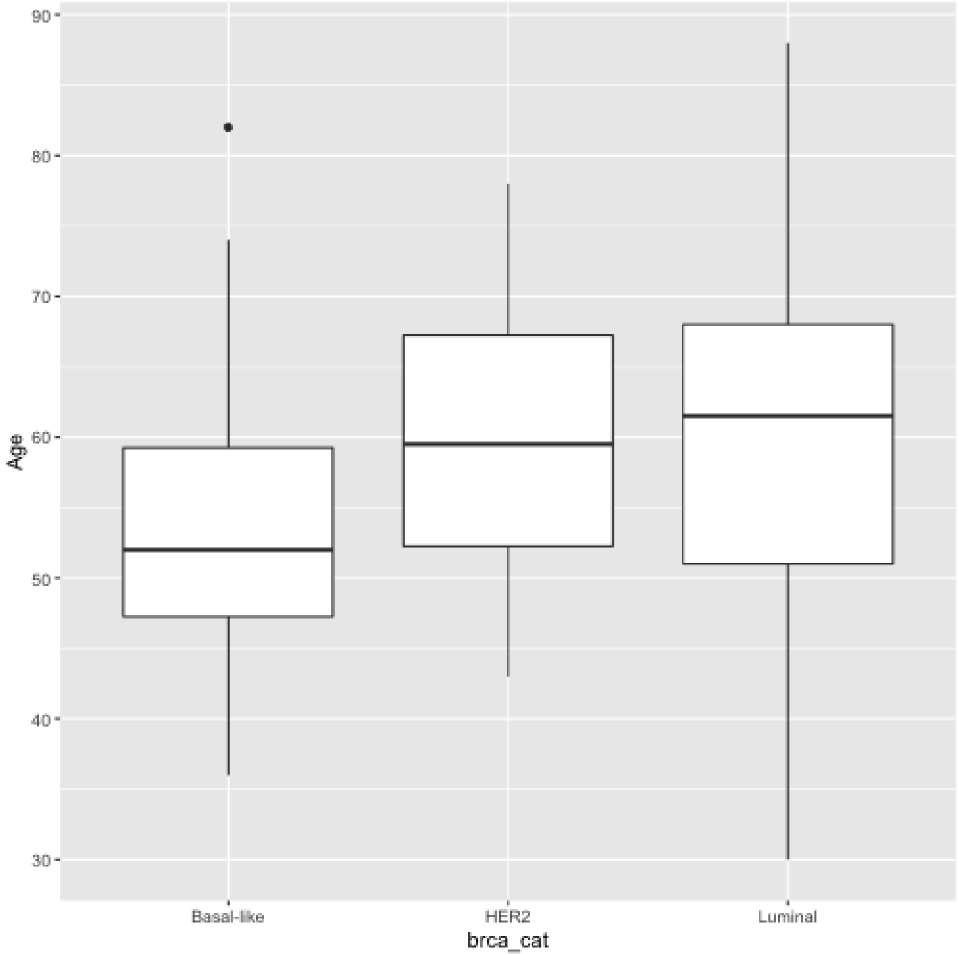
```
ggplot(data = brca_clean,
       mapping = aes(x = Tumor)) +
  geom_bar()
```



Bivariate plots

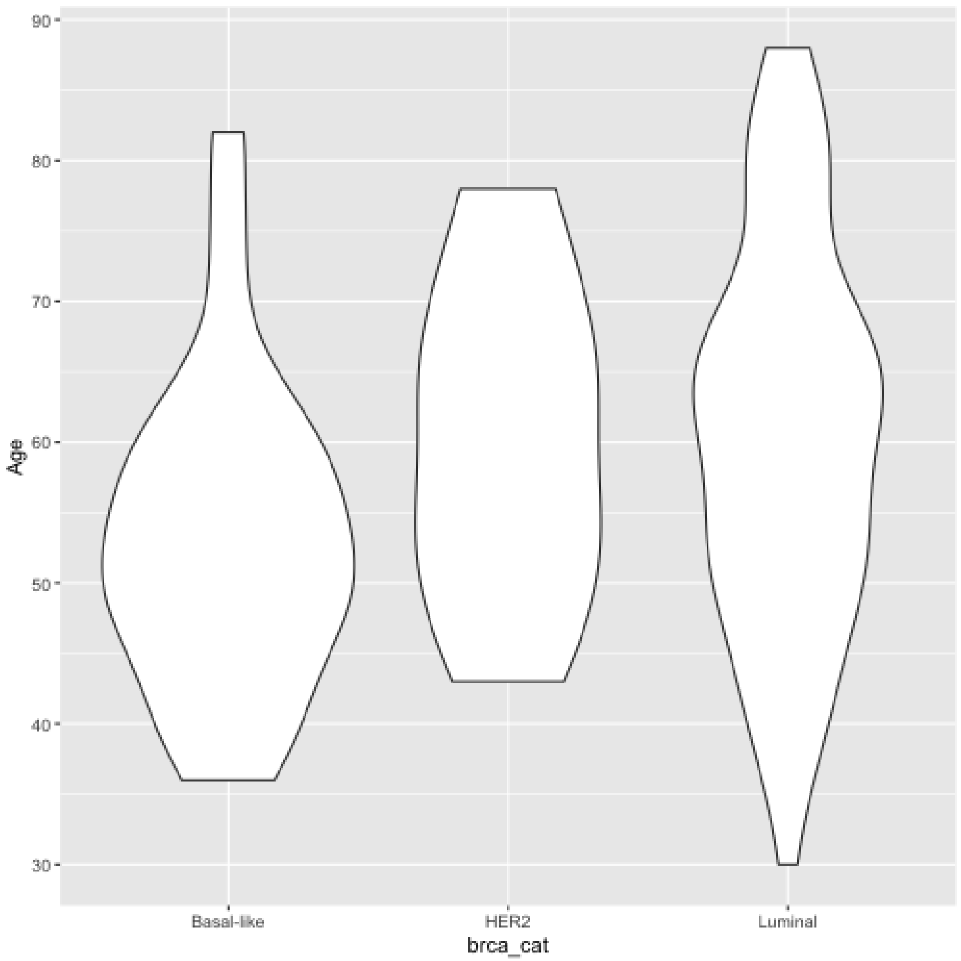
Continuous with discrete

```
ggplot(data = brca_clean,
       mapping = aes(x = brca_cat, # Put discrete on
                     y = Age)
       ) +
  geom_boxplot()
```



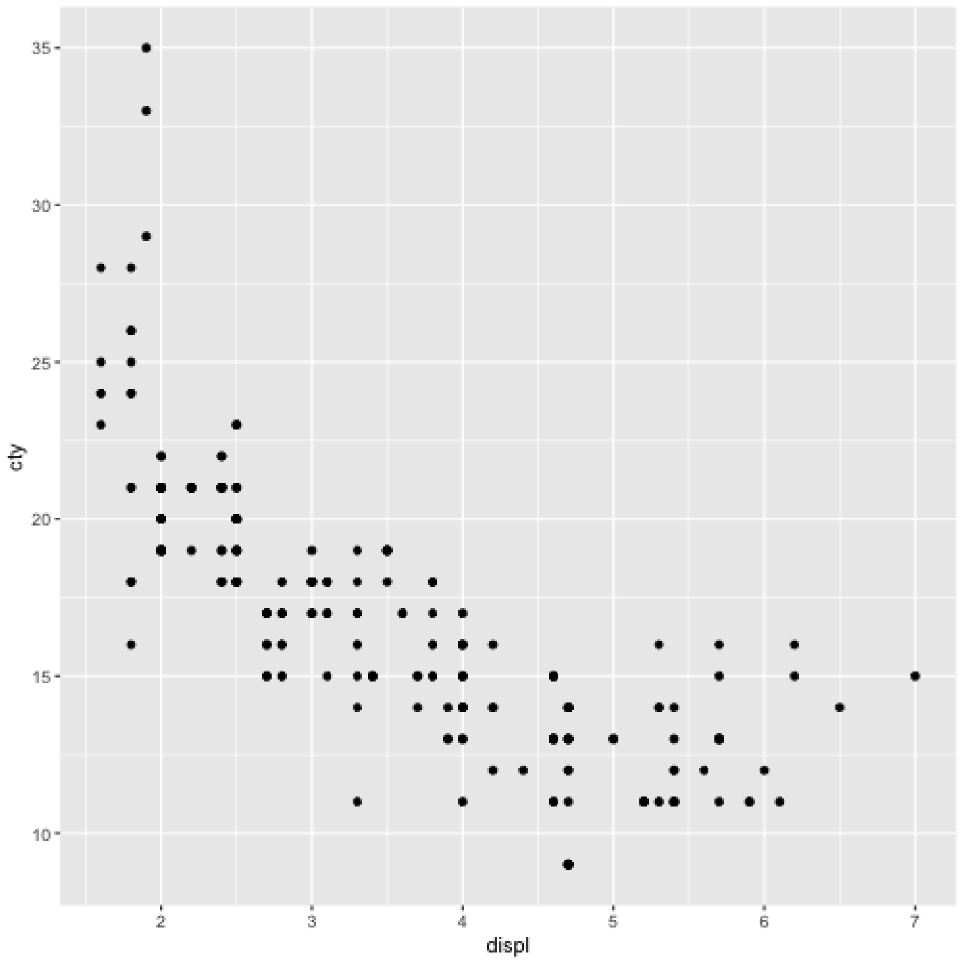
Continuous with discrete

```
ggplot(data = brca_clean,
       mapping = aes(x = brca_cat, # Put discrete on
                     y = Age)
       ) +
  geom_violin()
```



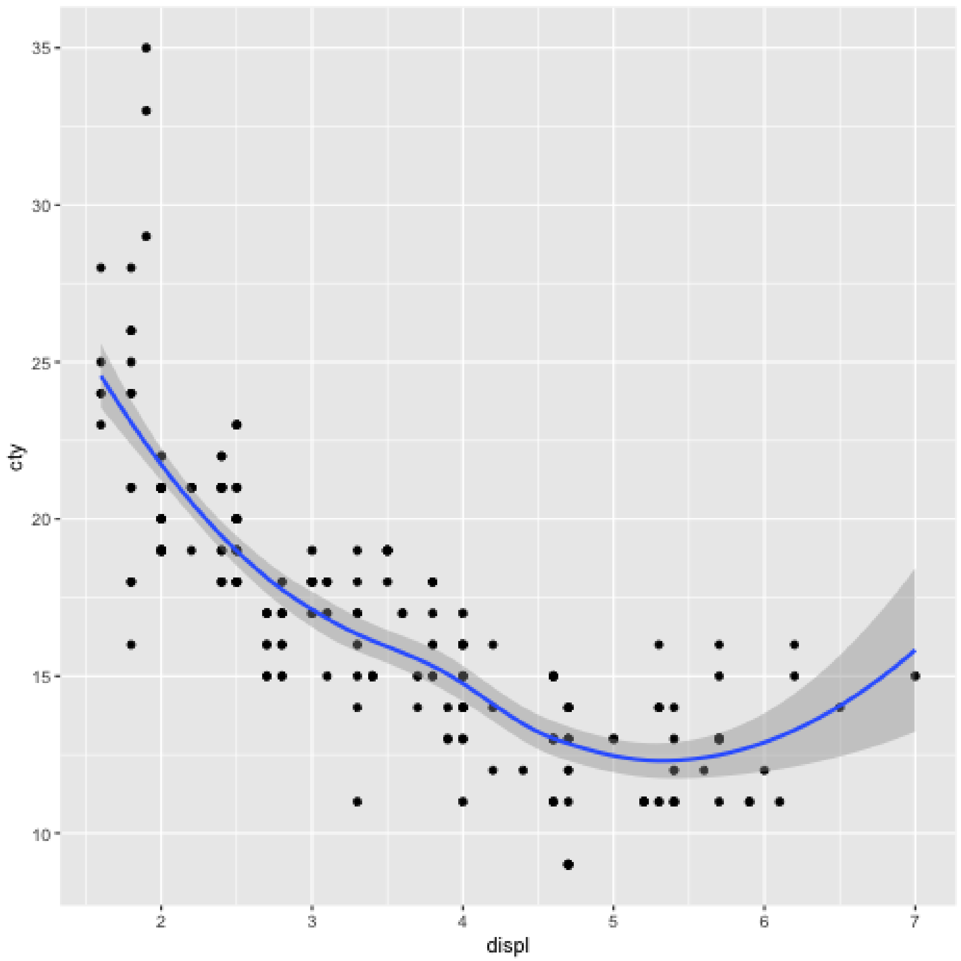
Two continuous variables

```
ggplot(data=mpg,
       mapping = aes(x = displ,
                     y = cty)
       ) +
  geom_point()
```



Two continuous variables

```
ggplot(data=mpg,
       mapping = aes(x = displ,
                     y = cty)
       ) +
  geom_point() + geom_smooth()
```

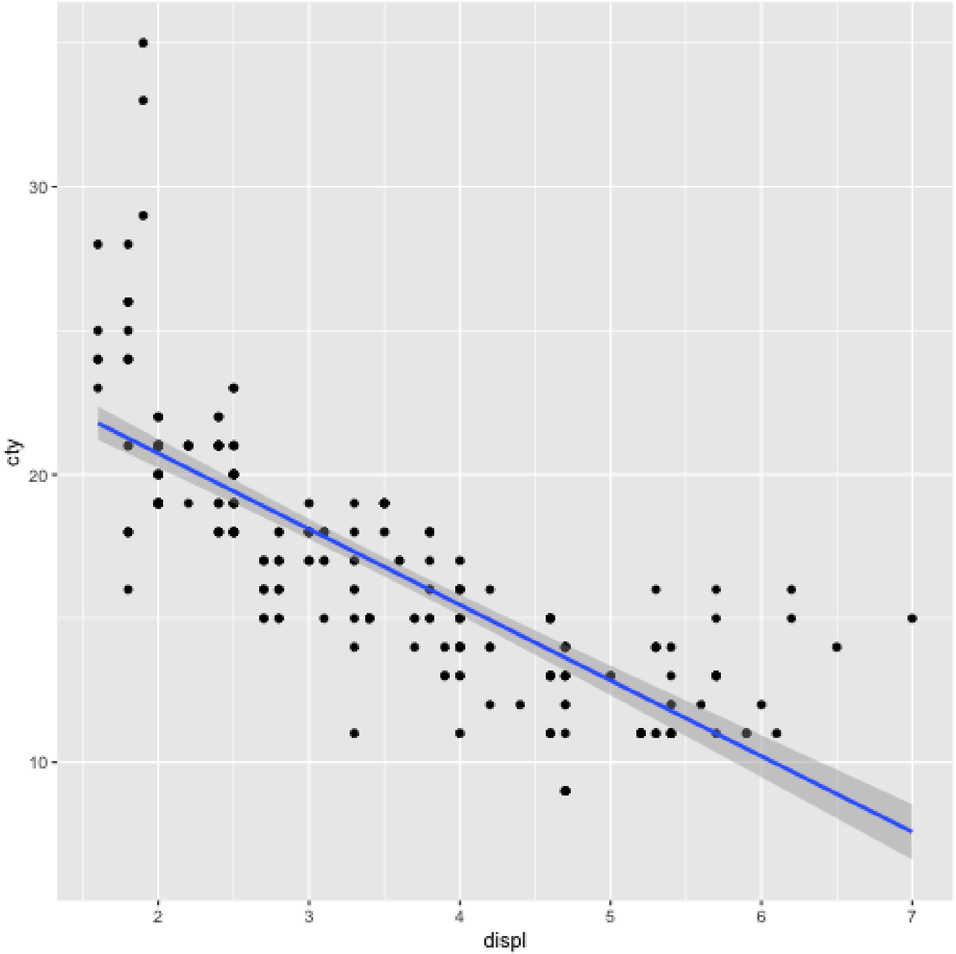


Two continuous variables

```
ggplot(data=mpg,
       mapping = aes(x = displ,
                     y = cty)
       ) +
  geom_point() + geom_smooth(method='lm')
```

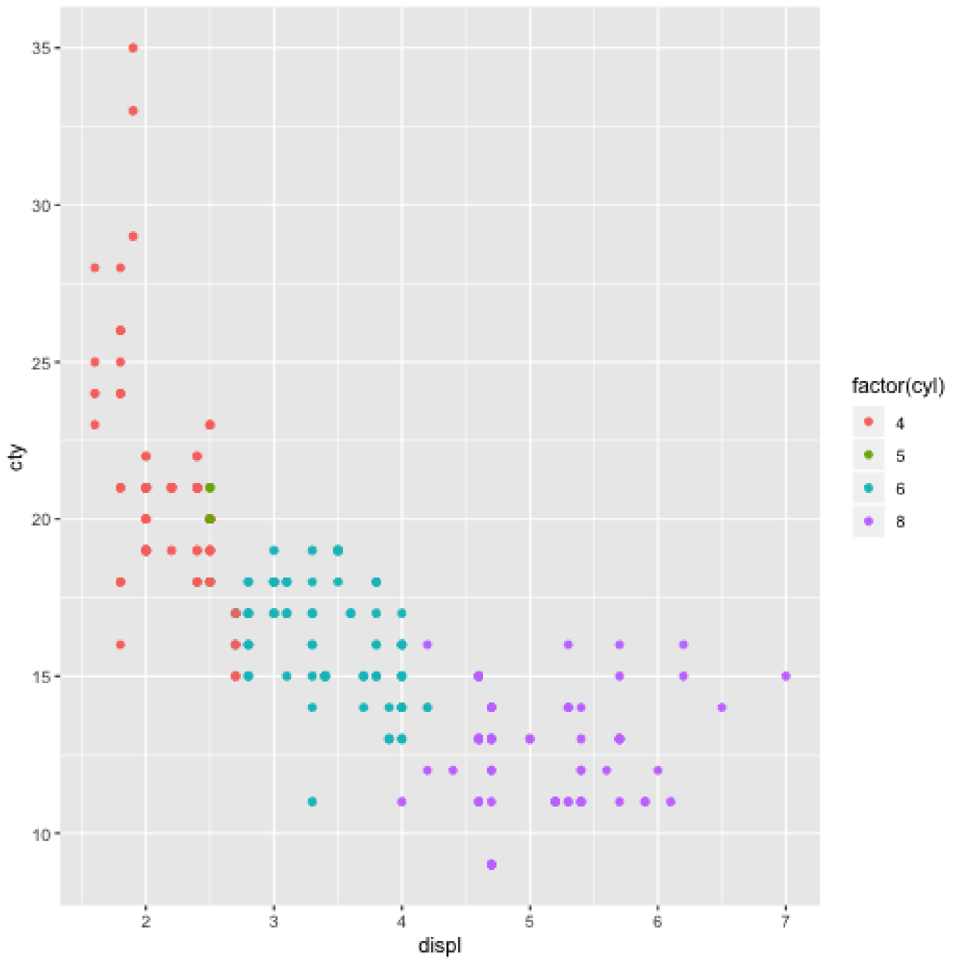
This forces a straight line.

`lm` stands for **linear model**



Adding layers

```
ggplot(data = mpg,
       aes(x = displ,
           y = cty,
           color = factor(cyl)))+
  geom_point()
```

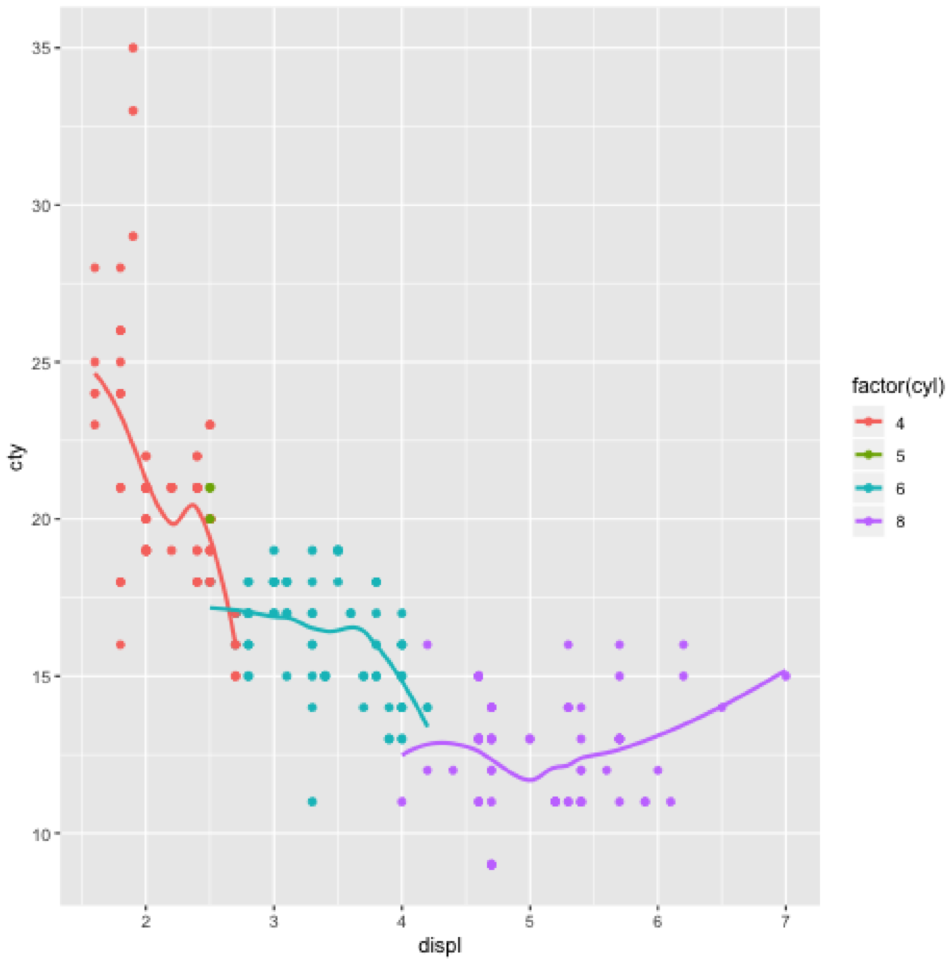


Adding layers

```
ggplot(data = mpg,
       aes(x = displ,
           y = cty,
           color = factor(cyl))+
  geom_point() +
  geom_smooth(se = F)
```

Separate lines for separate groups

`se=F` suppresses the confidence bands



Classwork checkin

What would happen if I tried to do the previous graph without transforming `cyl` to `factor`?

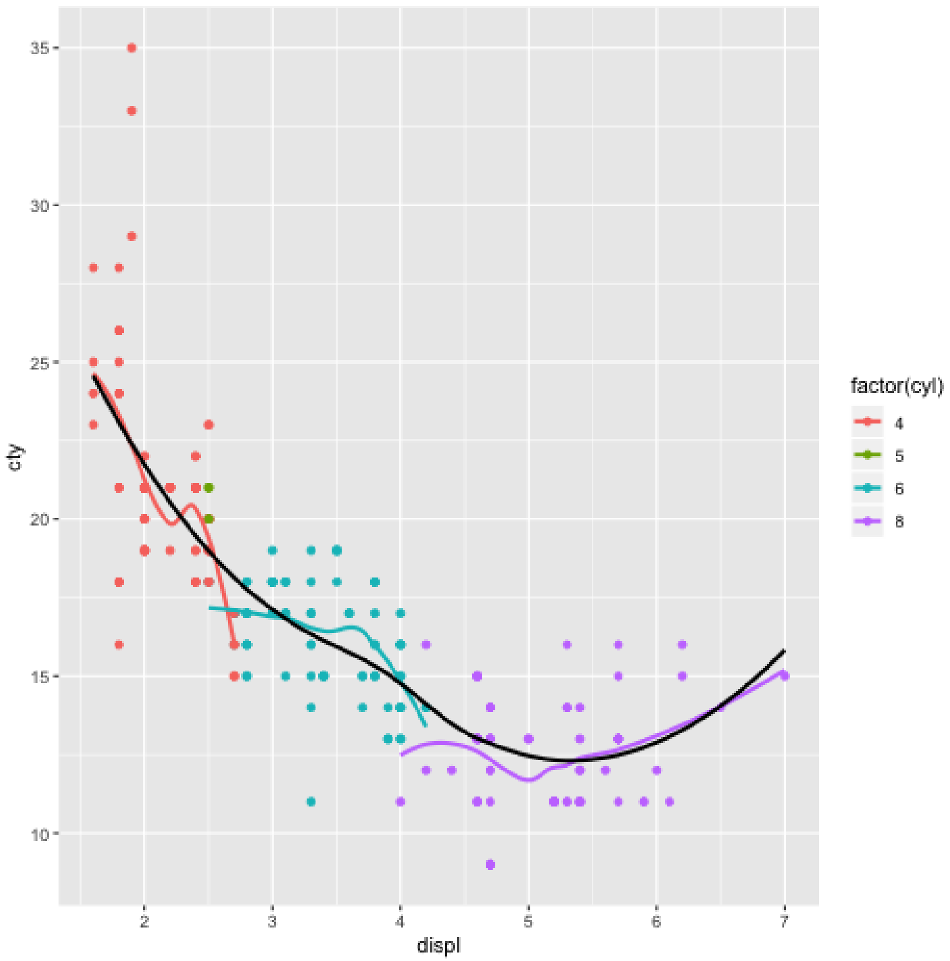
Adding layers

```
ggplot(data = mpg,
       aes(x = displ,
           y = cty))+
  geom_point(aes(color=factor(cyl))) +
  geom_smooth(aes(color=factor(cyl)), se=F) +
  geom_smooth(color = 'black',se=F)
```

- You can limit mappings to particular geometries

- Anything mapped from the original dataset has to be in `aes()`

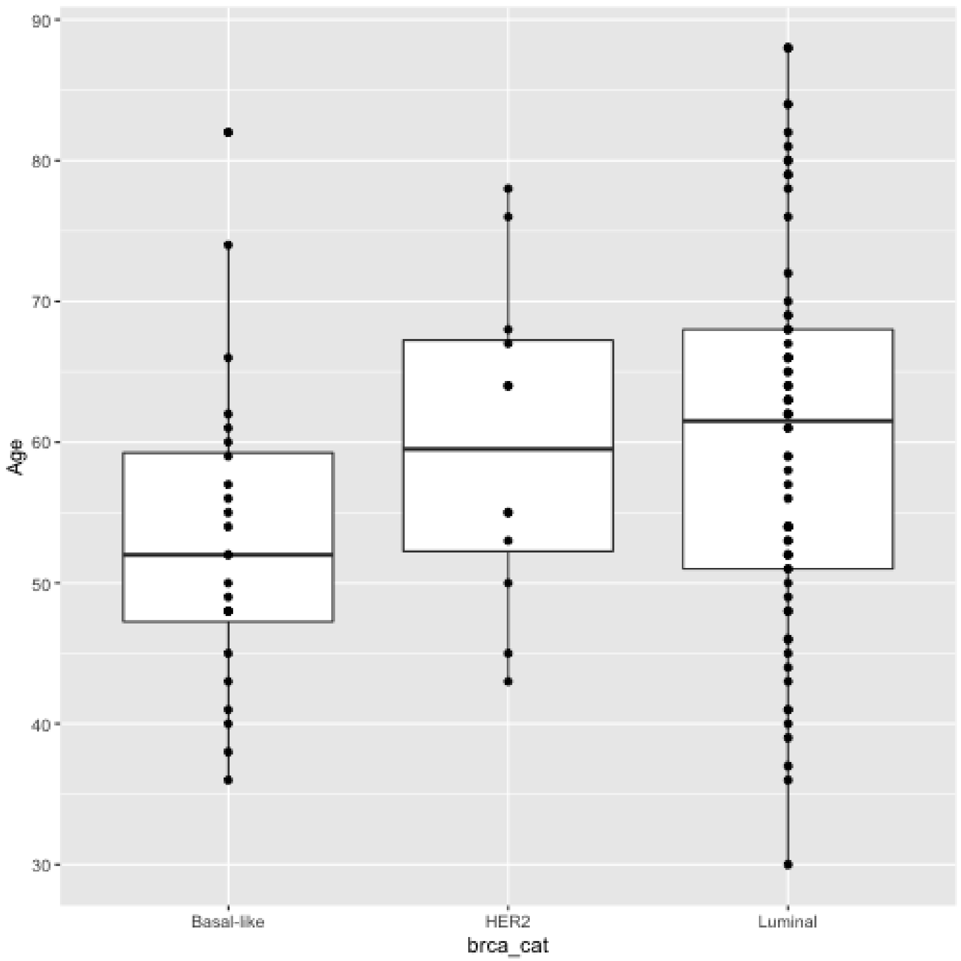
- Anything that doesn't come from the data can be on its own



Going back to the boxplots

```
ggplot(data = brca_clean,
       aes(x = brca_cat,
           y = Age)) +
  geom_boxplot() +
  geom_point()
```

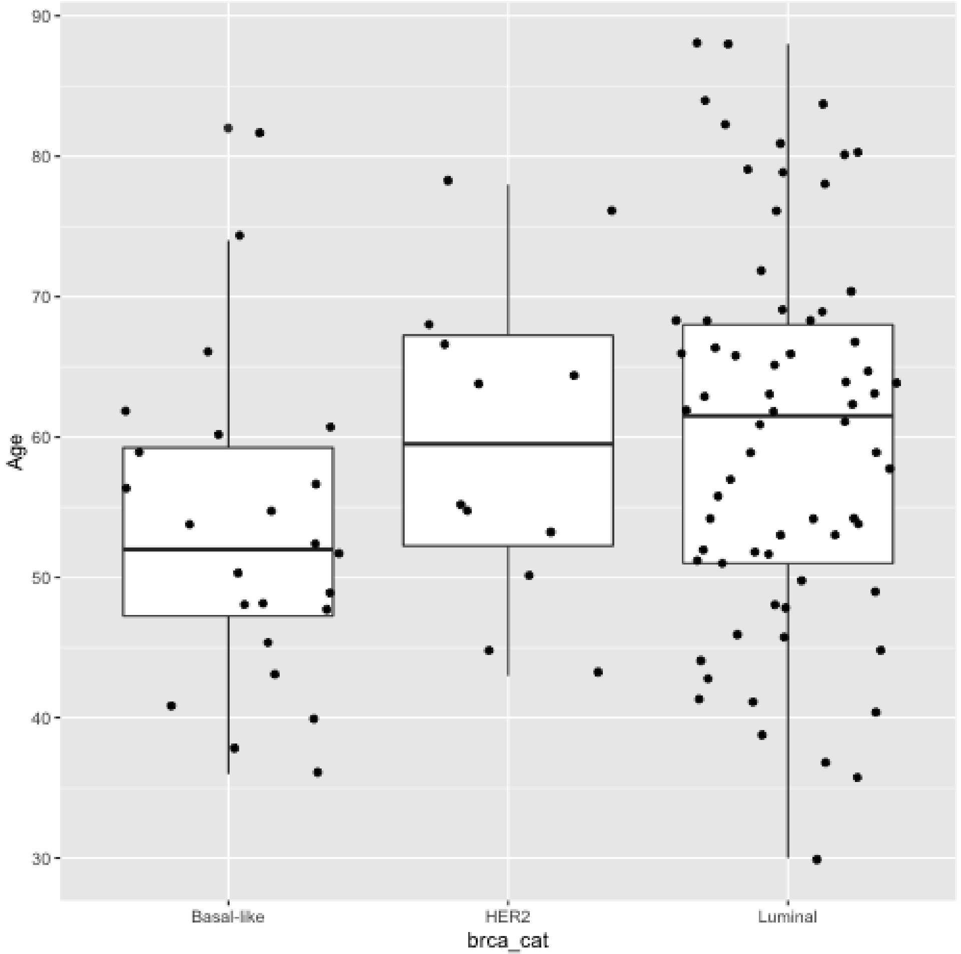
- Can't see the points since they are overlayed



Going back to the boxplots

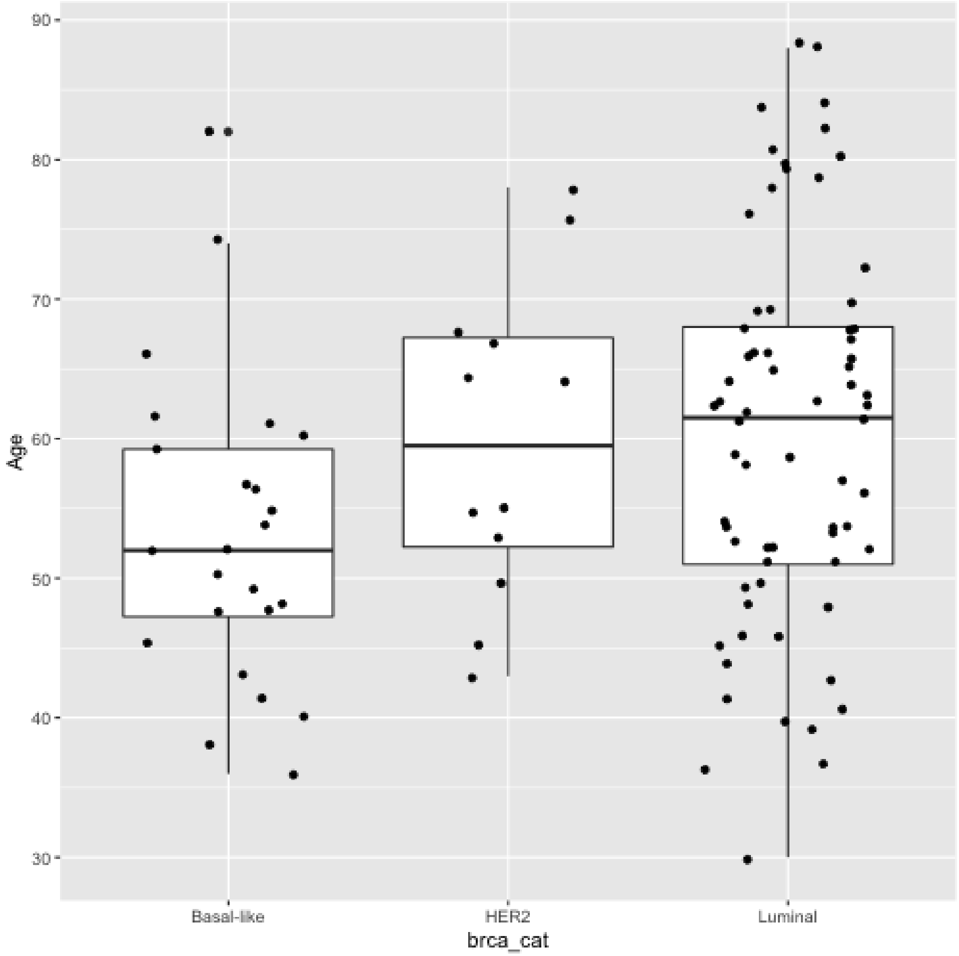
```
ggplot(data = brca_clean,
       aes(x = brca_cat,
           y = Age)) +
  geom_boxplot() +
  geom_jitter()
```

- Maybe too wide?



Going back to the boxplots

```
ggplot(data = brca_clean,
       aes(x = brca_cat,
           y = Age)) +
  geom_boxplot() +
  geom_jitter(width = 0.3)
```



Manhattan plots

Manhattan plot

```
library(qqman)
data(gwasResults)
head(gwasResults)
```

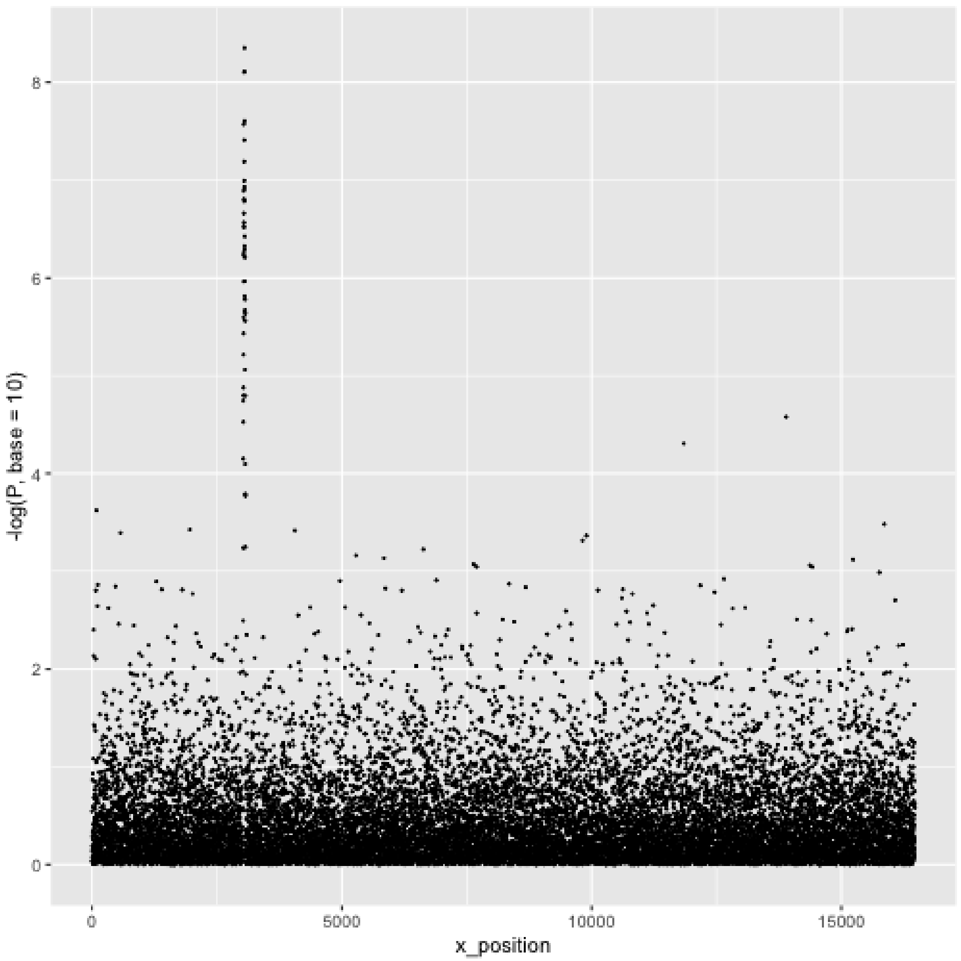
```
#>    SNP CHR BP         P
#>  1 rs1   1  1 0.9148060
#>  2 rs2   1  2 0.9370754
#>  3 rs3   1  3 0.2861395
#>  4 rs4   1  4 0.8304476
#>  5 rs5   1  5 0.6417455
#>  6 rs6   1  6 0.5190959
```

```
gwasResults <- gwasResults %>%
  mutate(x_position = 1:n())
head(gwasResults)
```

```
#>    SNP CHR BP         P x_position
#>  1 rs1   1  1 0.9148060          1
#>  2 rs2   1  2 0.9370754          2
#>  3 rs3   1  3 0.2861395          3
#>  4 rs4   1  4 0.8304476          4
#>  5 rs5   1  5 0.6417455          5
#>  6 rs6   1  6 0.5190959          6
```

Manhattan plot

```
ggplot(gwasResults,
       aes(x = x_position,
           y = -log(P, base=10))
       )+
  geom_point(size = 0.2)
```



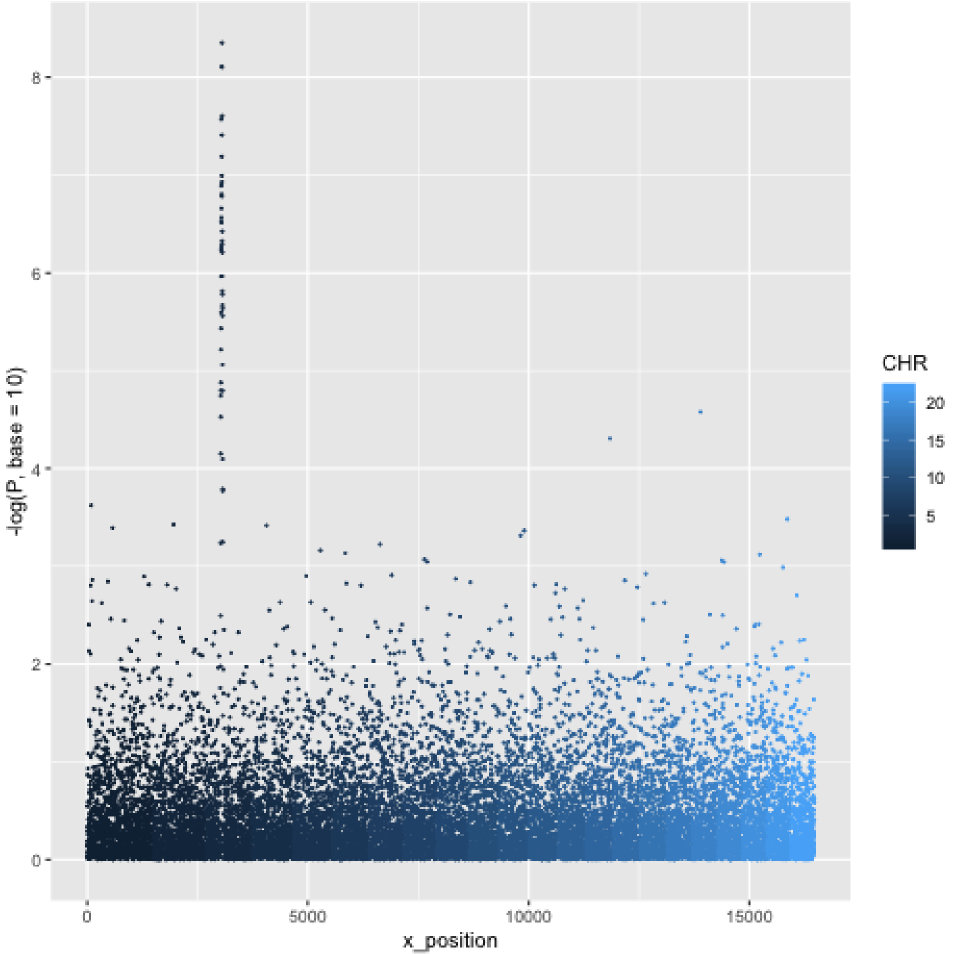
Manhattan plot

```
ggplot(gwasResults,
       aes(x = x_position,
           y = -log(P, base=10),
           group=CHR,
           color=CHR))+
  geom_point(size=0.2)
```

Oops!! We wanted points colored by chromosome.

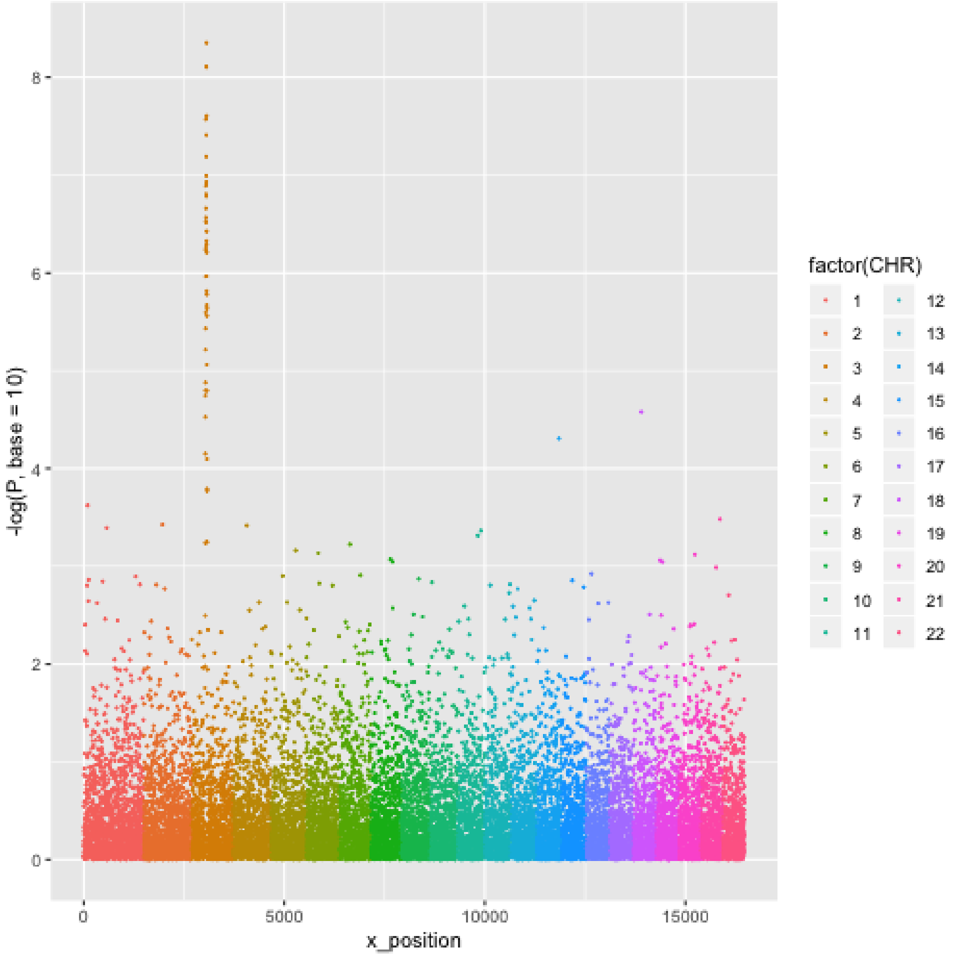
But that didn't happen because we put CHR in as numeric.

Need to convert to factor, i.e., discrete



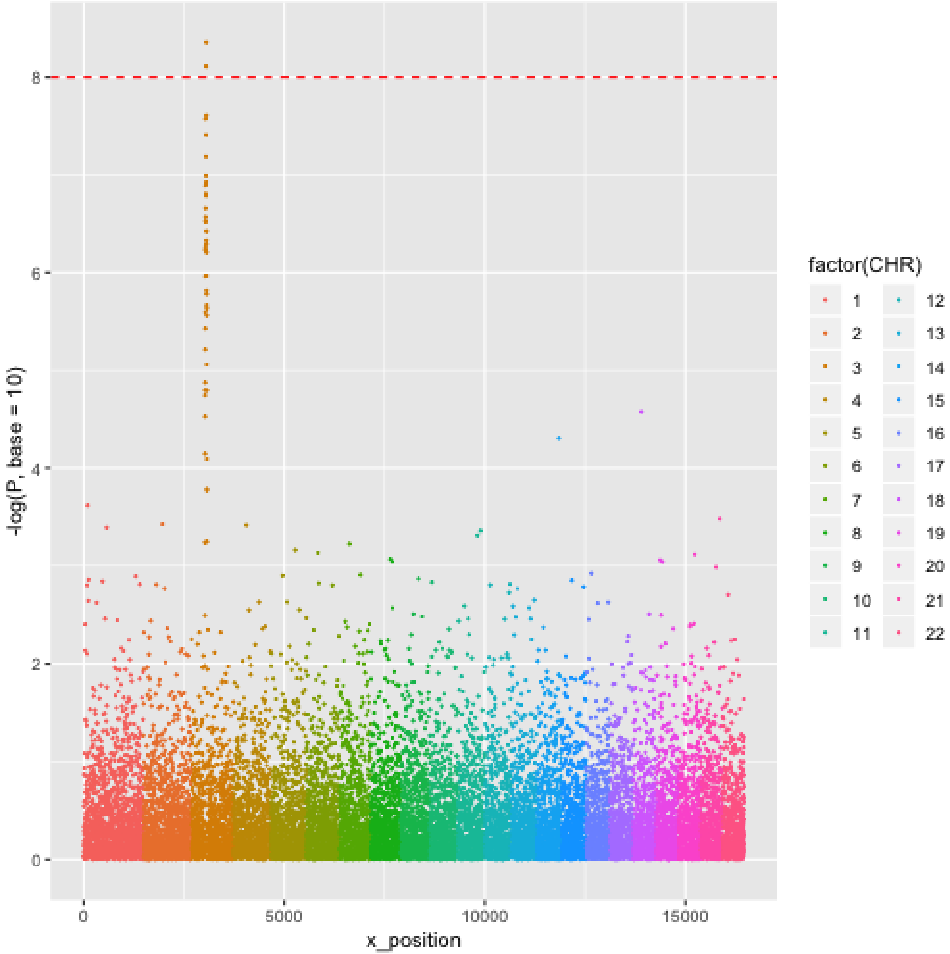
Manhattan plot

```
ggplot(gwasResults,
       aes(x = x_position,
           y = -log(P, base=10),
           group=factor(CHR),
           color=factor(CHR)))+
  geom_point(size=0.2)
```



Manhattan plot

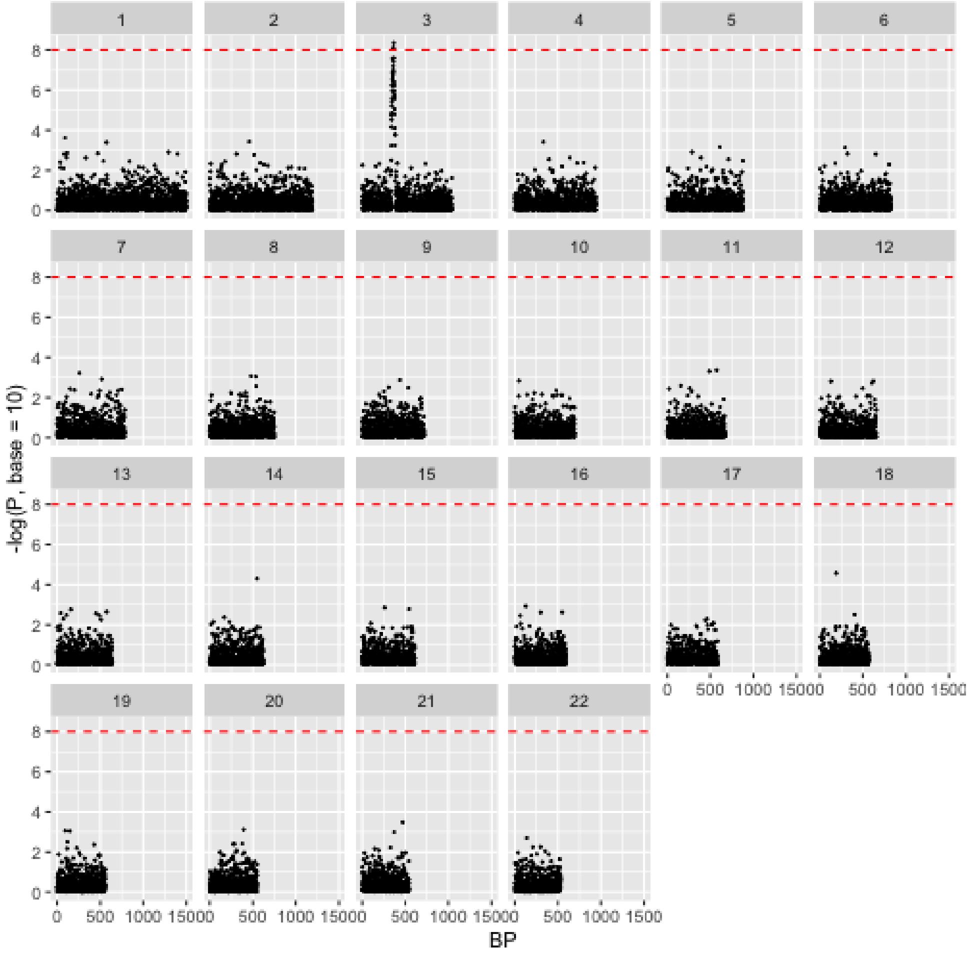
```
ggplot(gwasResults,
       aes(x = x_position,
           y = -log(P, base=10),
           group=factor(CHR), color=factor(CHR)))+
  geom_point(size=0.2)+
  geom_hline(yintercept = 8, color='red', linetype=2)
```



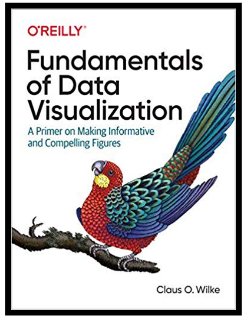
Manhattan plot, exploded

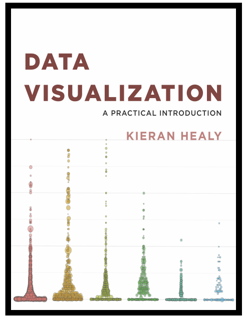
```
ggplot(gwasResults,
       aes(x = BP,
           y = -log(P, base=10)))+
  geom_point(size=0.2)+
  facet_wrap(~ CHR, nrow=4)+
  geom_hline(yintercept = 8,
             color='red',
             linetype=2)
```

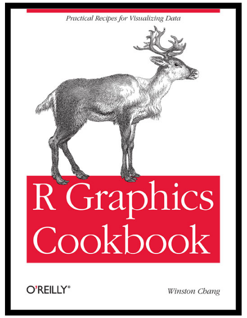
- No more grouping variable
- A new function `facet_wrap`



Resources


O'REILLY
Fundamentals of Data Visualization
A Primer on Making Informative and Compelling Figures
Claus O. Wilke





Data visualization cheatsheet (RStudio)

Chapter 3 of R4DS