Data visualization on R

Rémi Mahmoud

Who is talking ?

INRAE

What you have to do:

Open Rstudio

What you have to do:

Open Rstudio

Stop me if anything is unclear

What we will talk about today

What we will talk about today

- Visualize data in R

What we will talk about today

- Visualize data in R
- Customize your plots

What we will talk about today

- Visualize data in R
- Customize your plots
- Avoid caveats in data visualization

What we will talk about today

- Visualize data in R
- Customize your plots
- Avoid caveats in data visualization

What we will NOT talk about today

What we will talk about today

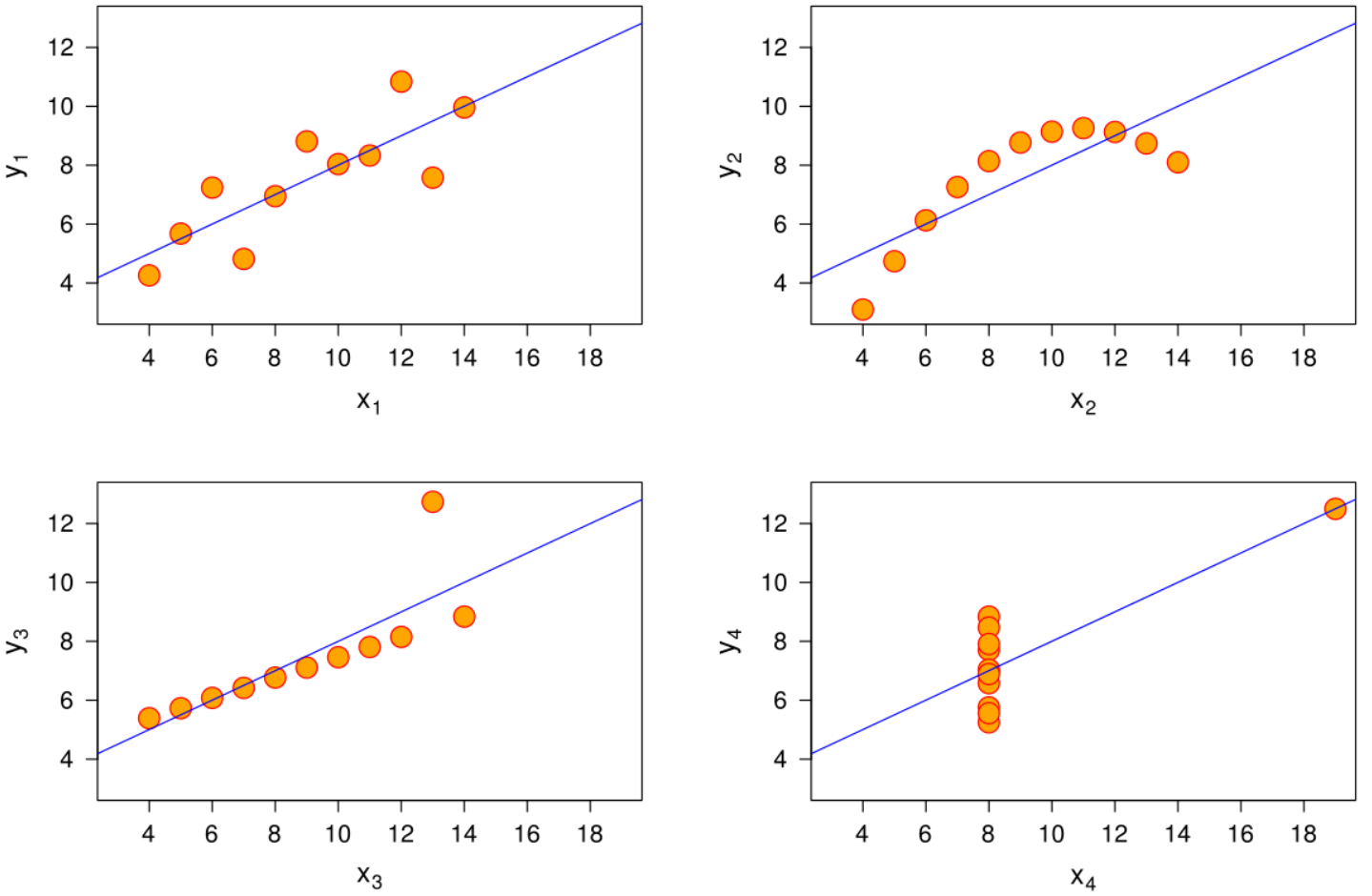
- Visualize data in R
- Customize your plots
- Avoid caveats in data visualization

What we will NOT talk about today

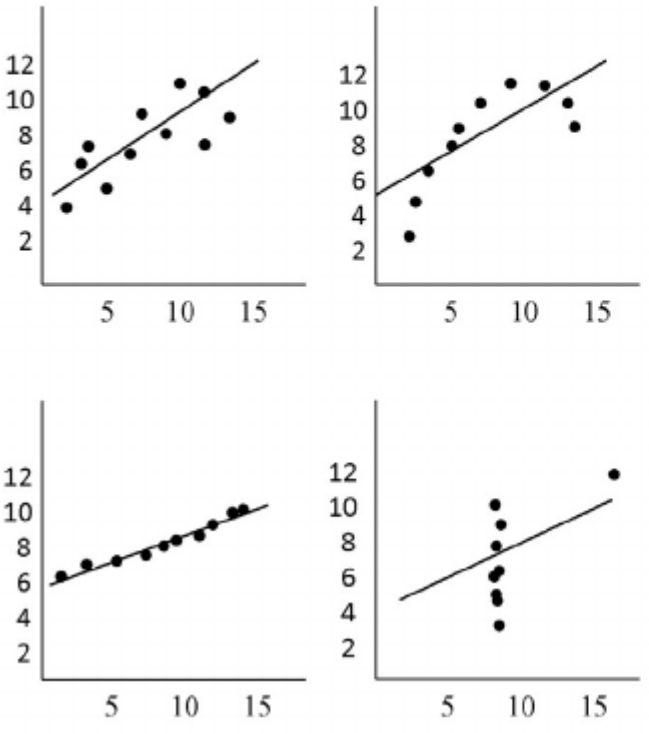
- Manipulate data etc. (next week :-))

Data-visualisation: WHY ?

What have these sets of points in common ?



Anscombe's Quartet



Property	Value
Mean of X (average)	9 in all 4 XY plots
Sample variance of X	11 in all four XY plots
Mean of Y	7.50 in all 4 XY plots
Sample variance of Y	4.122 or 4.127 in all 4 XY plots
Correlation (r)	0.816 in all 4 XY plots
Linear regression	y = 3.00 + (0.500 x) in all 4 XY plots

Data sets for the 4 XY plots

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	5.76
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	8.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	7.26	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

DATA-VISUALIZATION IS AN ESSENTIAL STEP

DATA-VISUALIZATION IS AN ESSENTIAL STEP

OUR BRAINS ARE NOT MADE TO SEE PATTERNS IN TABLES

Framework

All manipulations will be done in the `tidyverse` framework.

Framework

All manipulations will be done in the `tidyverse` framework.

Hence, you should, if not already done, run the following command in R **NOW**

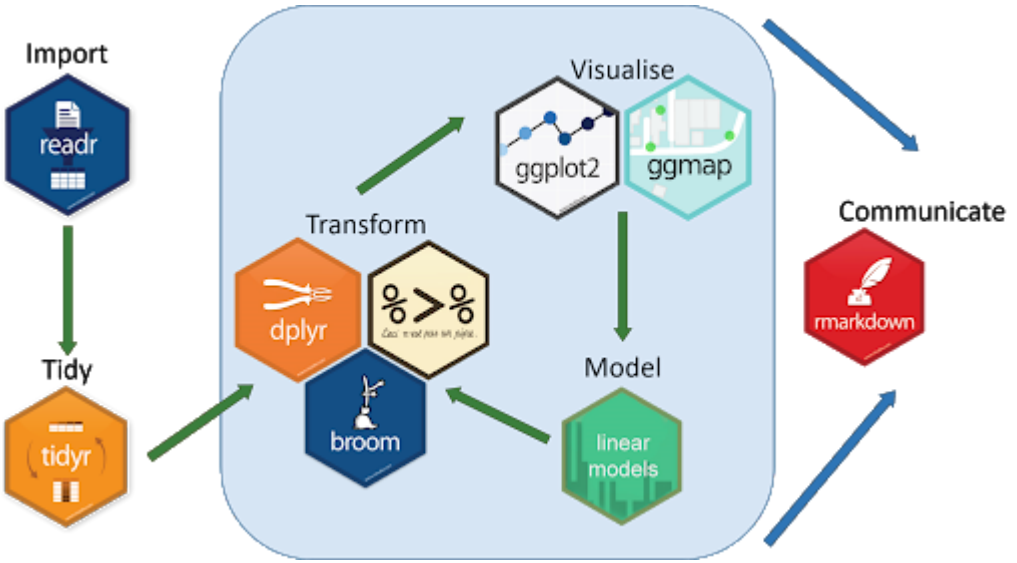
```
install.packages("tidyverse")
```

Tidyverse = Tidy universe

Tidyverse is a set of packages with differents purposes, that share the same syntax and that are designed to work in a complementary way

Tidyverse = Tidy universe

Tidyverse is a set of packages with differents purposes, that share the same syntax and that are designed to work in a complementary way

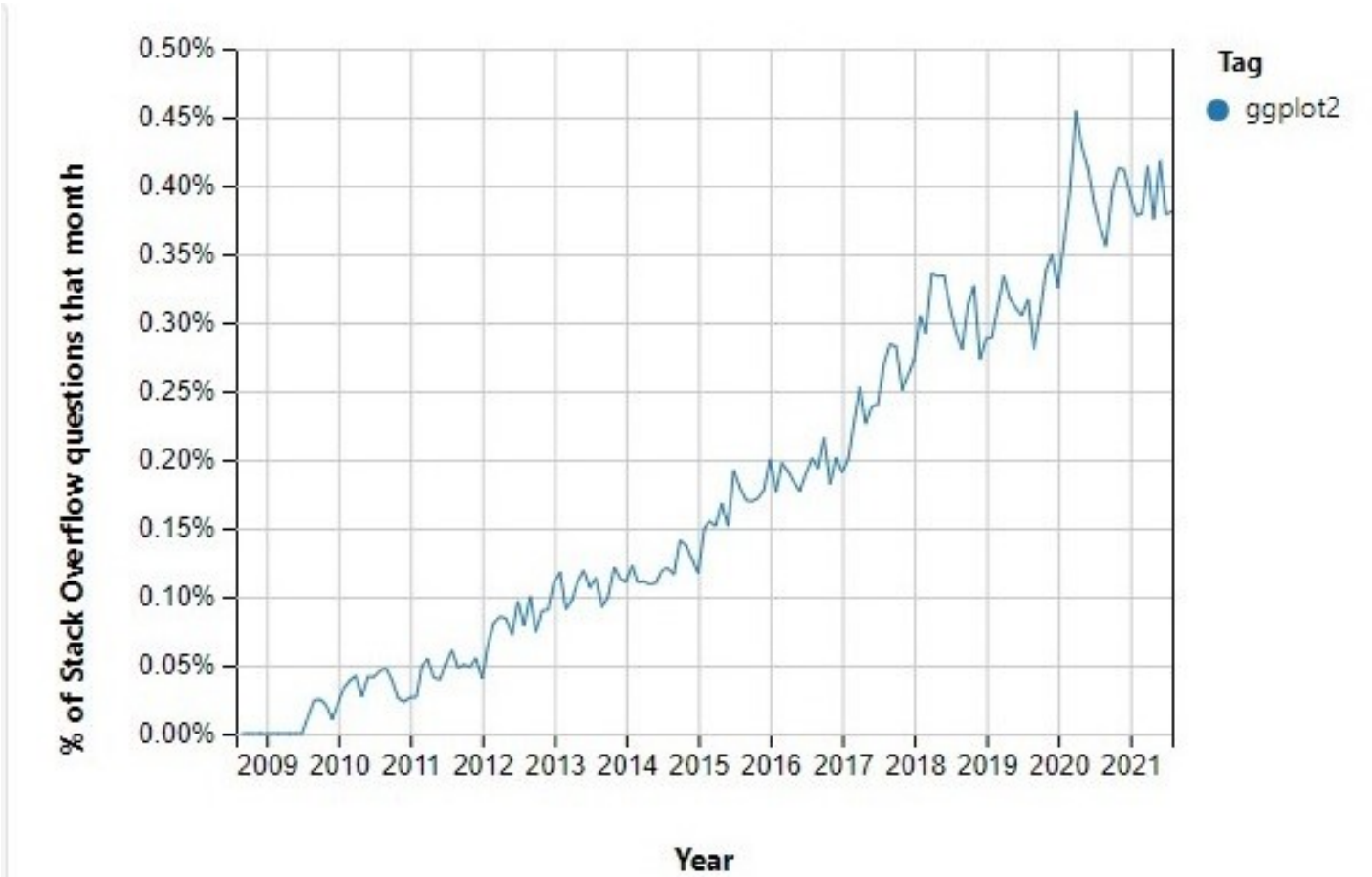


Today, we'll focus on the package ggplot2.

Today, we'll focus on the package ggplot2.

Today, we'll focus on the package ggplot2.

ggplot2, WHY ?


% of Stack Overflow questions that month
0.50%
0.45%
0.40%
0.35%
0.30%
0.25%
0.20%
0.15%
0.10%
0.05%
0.00%
2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021
Year
Tag
ggplot2

Base R plots vs ggplot2: the clash

Base graphics:

- directly available from R
- easy to use for beginners
- is less verbose for simple / canned graphics
- has methods (plot works for many different ojects)

ggplot2:

- R graphing package by Hadley Wickham based on the Grammar of Graphics (Wilkinson, 2005)
- is less verbose for complex / custom graphics: plots can be iteratively built up and edited later
- carefully chosen defaults = publication-quality graphics in seconds

Harder ? Not really

GGPLOT2: BASICS

ggplot2 building blocks

- **Data**: a `data.frame` and nothing else!
- **Aesthetic mapping**: describe how data are mapped to things we can see on the plot through the function `aes()`.
- **Geometric object**: perform the actual rendering of the plot and control the type of plot to create (`points`, `line`, `histogram`, `boxplot`...).
- **Statistical transformations**: transform the data for instance by summarising it in some manner (`sum`, `density`, `smooth`...)
- **Position adjustments**: apply minor changes to the position of elements (`jitter`, `fill`, `stack`, `dodge`, `identity`)

In a nutshell

The Basics of ggplot2

There are three main components that get *added* together to build up a plot.

Required: "Hey, R, I'm building a plot, and I'm using the data in the `data=` argument* to do it!"

Optional: "Here's the 'default' mappings (`mapping=aes(...)`) of specific data in that dataset and how I want to plot them – as the 'x' values or 'y' values or something else."

Required: "Give me a 'layer' of that chart as a column chart or a scatterplot, or a line chart, (or something else)."

Optional: "If I didn't give it to you above, or if I need to override some part of it, here's the specific mappings (`mapping=aes(...)`) to use for this layer, the specific data (`data=`), and some guidance on what colors to use for what."

Optional: "Here's how I actually want to *format* the visualization: which tick marks and gridlines to show and what color to make them, which axis labels to show and various font properties, etc."

* *Not always the case, and not strictly required, but ggplot2 works best with data in a long format (or, at least, a tidy format).* `gather()` *from the* `tidyr` *package is your friend here.*

Syntax

```
library(ggplot2)
ggplot(data = <DATA>,
       aes(x = <X AXIS VARIABLE>,
           y = <Y AXIS VARIABLE>, ... ), ...) +

  geom_<TYPE>(aes(size = <SIZE VARIABLE>, ...),
                  data = <DATA>,
                  stat = <FUNCTION>,
                  position = <POSITION>,
                  color = <"COLOR">, ...) +

  scale_<AESTHETIC>_<TYPE>(name = <NAME>,
                  breaks = <WHERE>,
                  labels = <LABELS>, ... ) +

  theme(...) +
  facet_<TYPE>(<FORMULA>)
```

What happens when we create a simple ggplot() object ?

```
library(ggplot2)
ggplot()
```


Almost nothing

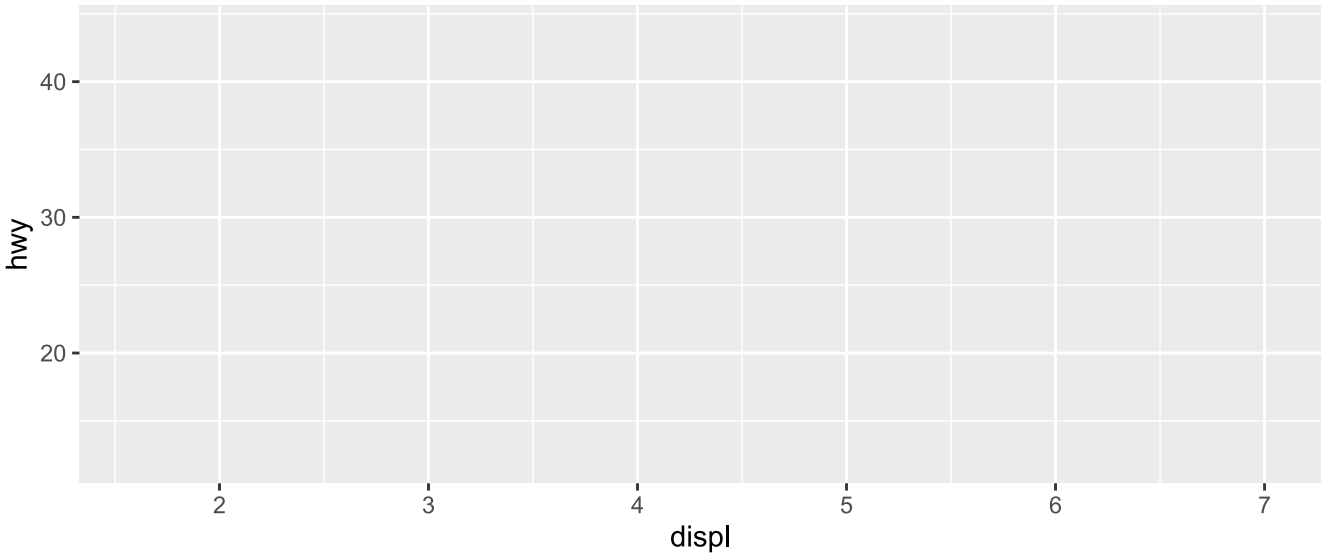
Specify data

```
data("mpg")
head(mpg)
```

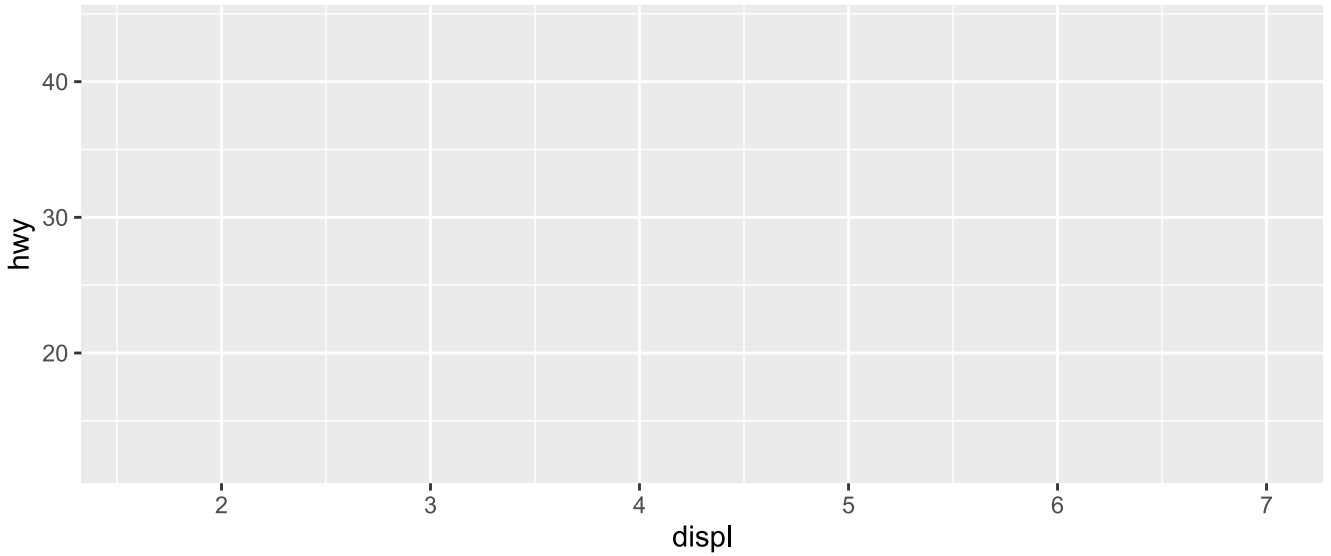
```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29
## 3 audi         a4      2    2008     4 manual(m6) f        20    31
## 4 audi         a4      2    2008     4 auto(av)   f        21    30
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26
```

```
# See ?mpg for the description of the dataset
```

```
p <- ggplot(data = mpg, aes(x =displ,  y= hwy ))
p
```



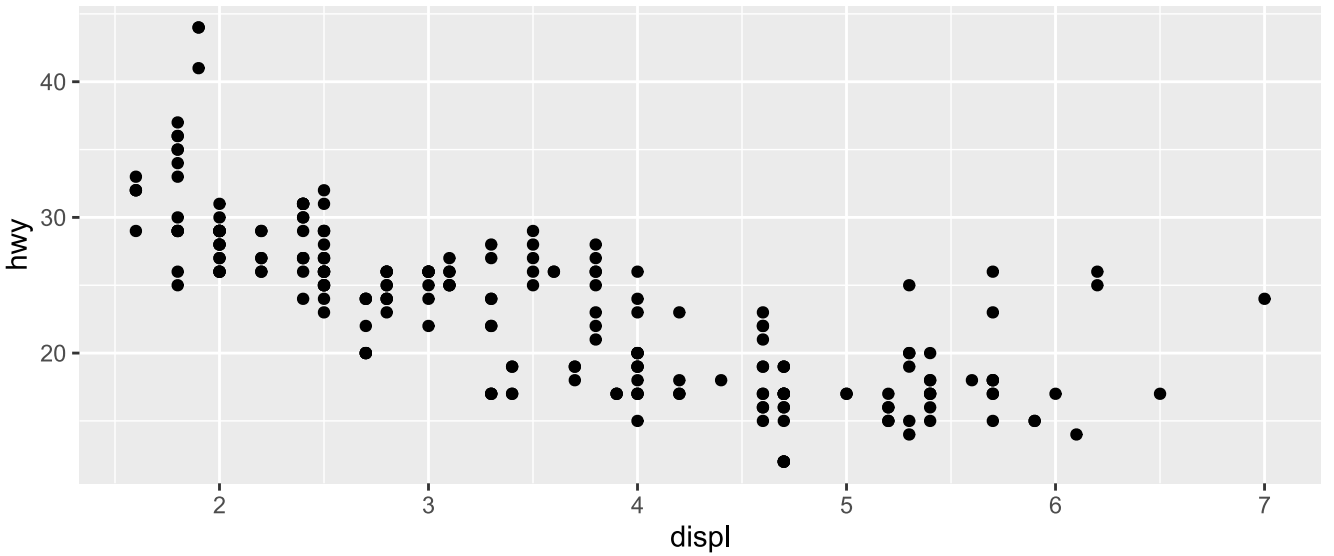
```
p <- ggplot(data = mpg, aes(x =displ,  y= hwy ))
p
```



Almost nothing ? Look at the axes :-)

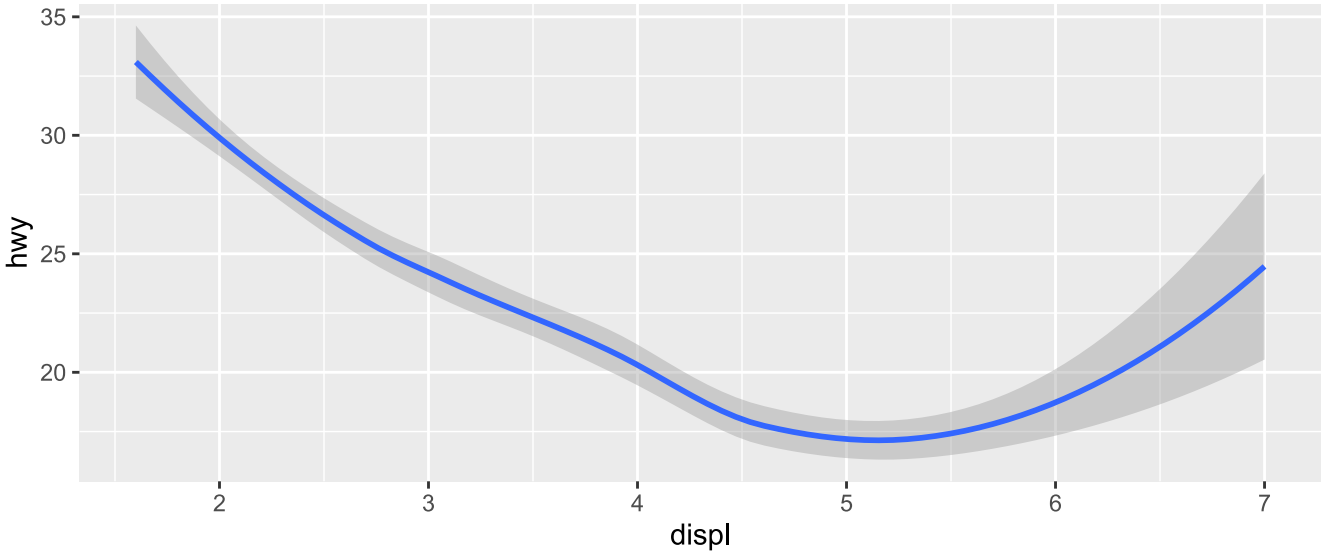
Let's specify what sort of plot we want, using a `geom_*` functions

```
p <- ggplot(data = mpg, aes(x =displ,  y= hwy ))
p + geom_point()
```



Another geom_*

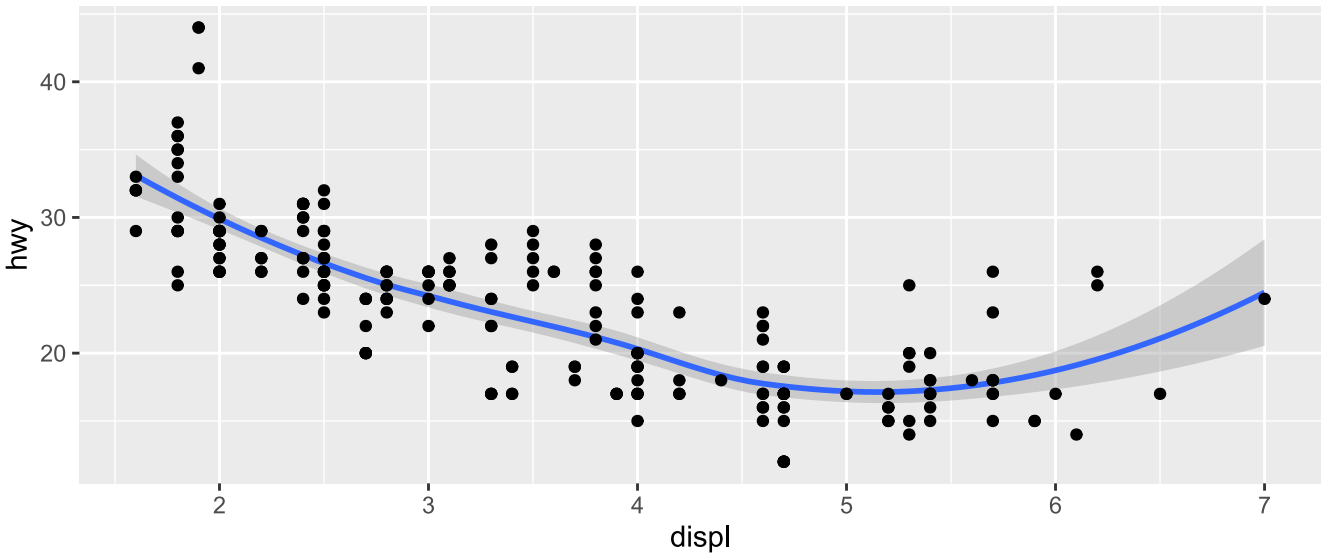
```
p <- ggplot(data = mpg, aes(x =displ,  y= hwy ))
p + geom_smooth()
```



Combine two geom ?

Simply **add** as many geom_functions as you want

```
p <- ggplot(data = mpg, aes(x =displ,  y= hwy ))
p +
  geom_smooth() +
  geom_point()
```



Combine three `geom`?

- Suppose we want to also look a linear threnḍ
- Need for help

```
?geom_smooth
```

Arguments

mapping Set of aesthetic mappings created by `aes()` or `aes_()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

data The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

position Position adjustment, either as a string, or the result of a call to a position adjustment function.

... Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

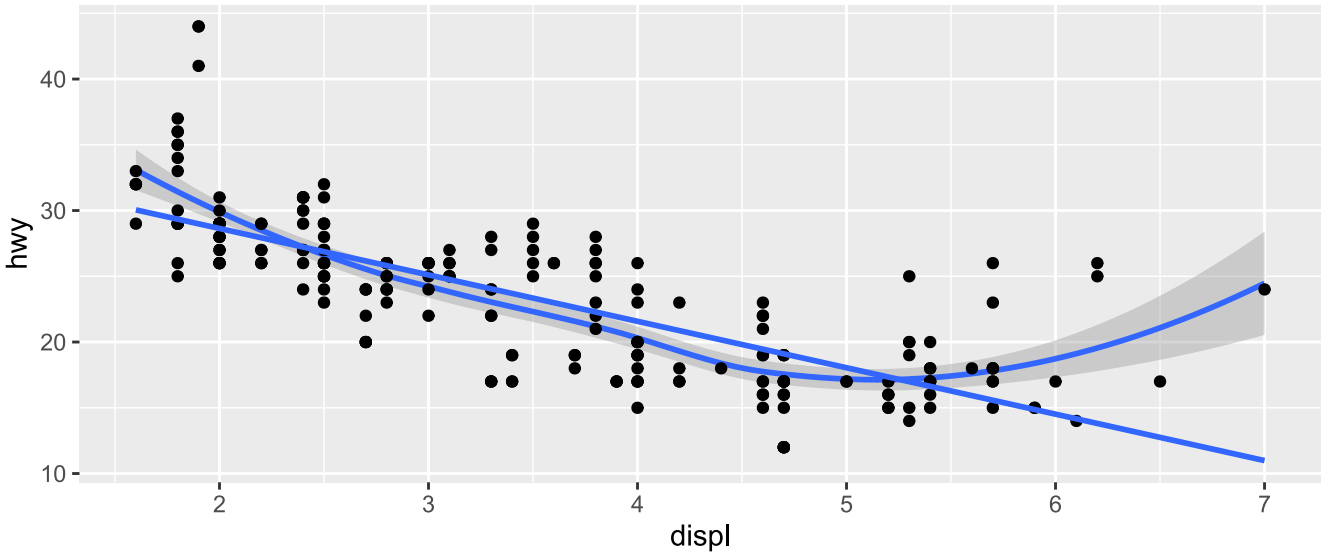
method Smoothing method (function) to use, accepts either `NULL` or a character vector, e.g. `"lm"`, `"glm"`, `"gam"`, `"loess"` or a function, e.g. `MASS::rlm` or `mgcv::gam`, `stats::lm`, or `stats::loess`. `"auto"` is also accepted for backwards compatibility. It is equivalent to `NULL`.

For `method = NULL` the smoothing method is chosen based on the size of the largest group (across all panels). `stats::loess()` is used for less than 1,000 observations; otherwise `mgcv::gam()` is used with `formula = y ~ s(x, bs = "cs")` with `method = "REML"`. Somewhat anecdotally, `loess` gives a better appearance, but is $O(N^2)$ in memory, so does not work for larger datasets.

If you have fewer than 1,000 observations but want to use the same `gam()` model that `method = NULL` would use, then set `method = "gam", formula = y ~ s(x, bs = "cs")`.

formula Formula to use in smoothing function, eg. `y ~ x`, `y ~ poly(x, 2)`, `y ~ log(x)`. `NULL` by default, in which case `method = NULL` implies `formula = y ~ x` when there are fewer than

```
p <- ggplot(data = mpg, aes(x =displ,  y= hwy ))
p +
  geom_smooth() +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```



There are *many* geom_* functions:

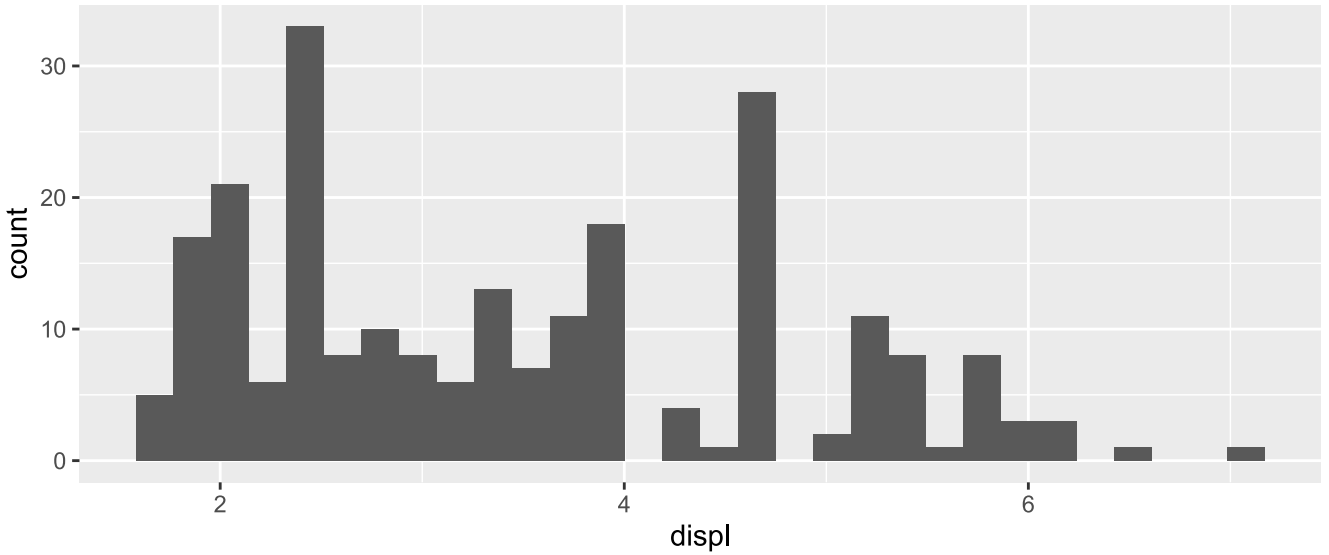
```
ls("package:ggplot2")[grepl(ls("package:ggplot2"),pattern =  "geom")]
```

```
##  [1] "geom_abline"           "geom_area"             "geom_bar"
##  [4] "geom_bin2d"            "geom_blank"            "geom_boxplo
##  [7] "geom_col"              "geom_contour"          "geom_contou
## [10] "geom_count"            "geom_crossbar"         "geom_curve"
## [13] "geom_density"          "geom_density_2d"       "geom_densit
## [16] "geom_density2d"        "geom_density2d_filled" "geom_dotplo
## [19] "geom_errorbar"         "geom_errorbarh"        "geom_freqpo
## [22] "geom_function"         "geom_hex"              "geom_histog
## [25] "geom_hline"            "geom_jitter"           "geom_label"
## [28] "geom_line"             "geom_linerange"        "geom_map"
## [31] "geom_path"             "geom_point"            "geom_pointr
## [34] "geom_polygon"          "geom_qq"               "geom_qq_lin
## [37] "geom_quantile"         "geom_raster"           "geom_rect"
## [40] "geom_ribbon"           "geom_rug"              "geom_segmen
## [43] "geom_sf"               "geom_sf_label"         "geom_sf_tex
## [46] "geom_smooth"           "geom_spoke"            "geom_step"
## [49] "geom_text"             "geom_tile"             "geom_violin
## [52] "geom_vline"            "guide_geom"            "update_geom
```

Quick look around of some classical `geom_*` functions:

`geom_histogram`

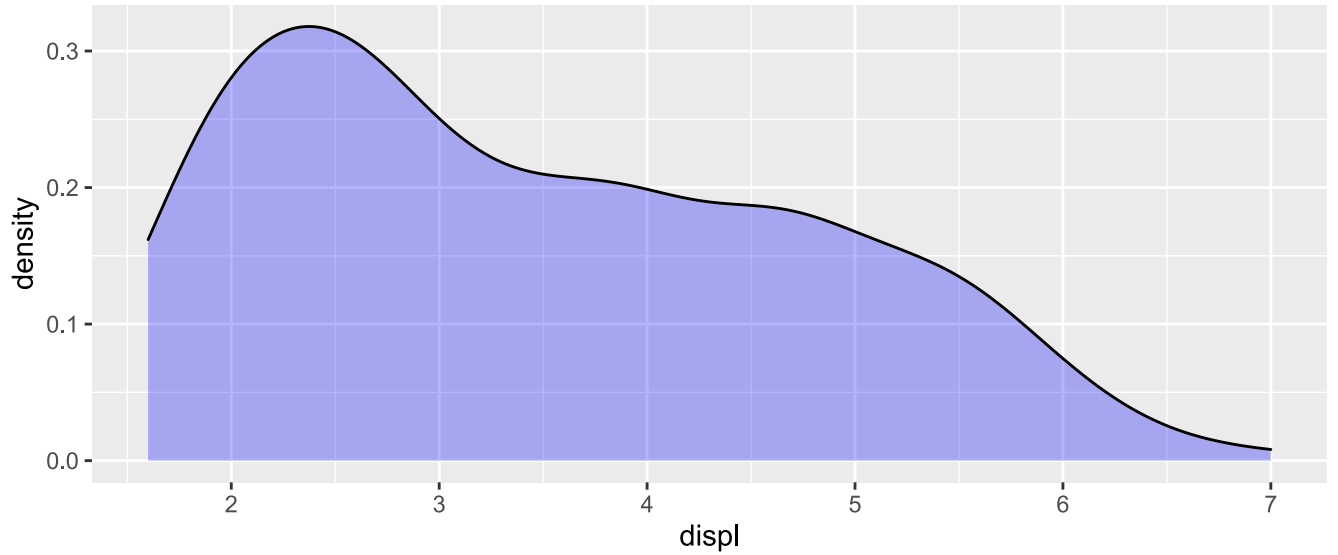
```
ggplot(data = mpg, aes(x = displ)) +
  geom_histogram()
```



Quick look around of some classical `geom_*` functions:

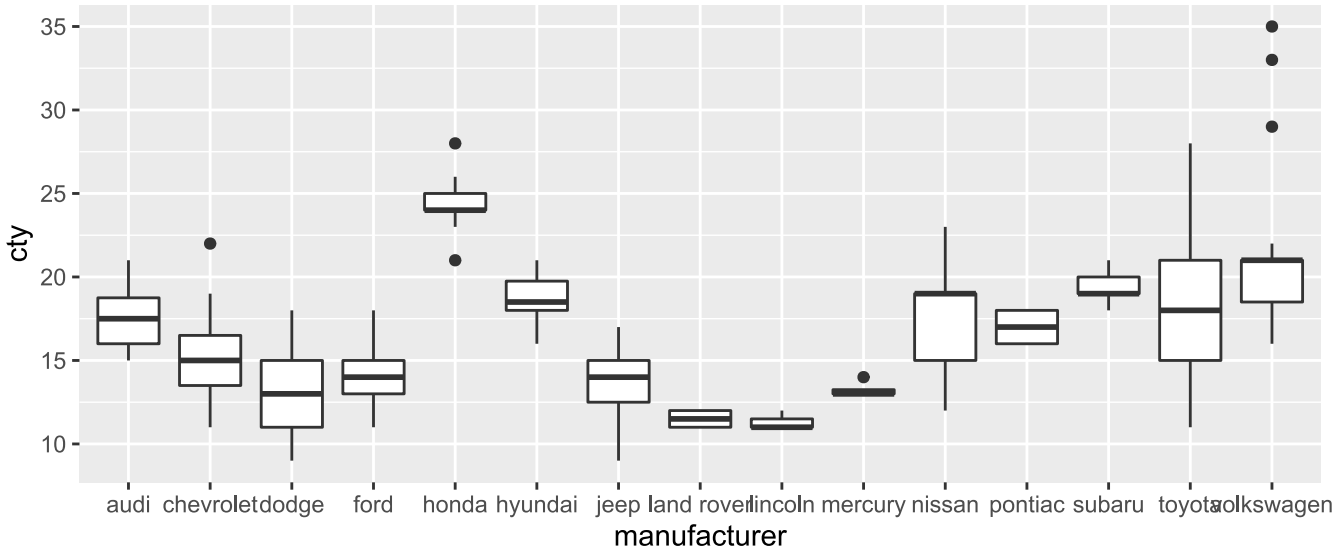
`geom_density`

```
ggplot(data = mpg, aes(x = displ)) +
  geom_density(fill = "blue", alpha = .3) # alpha: transparency
```



Quick look around of some classical `geom_*` functions: `geom_boxplot`

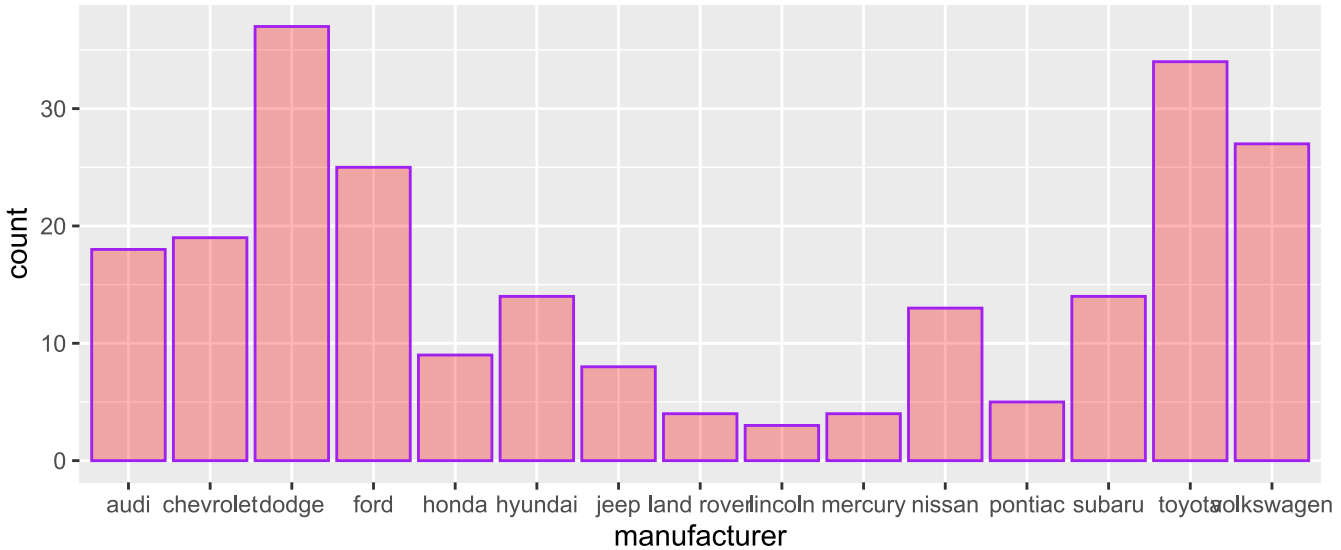
```
ggplot(data = mpg, aes(x = manufacturer, y = cty)) +
  geom_boxplot()
```



Quick look around of some classical `geom_*` functions:

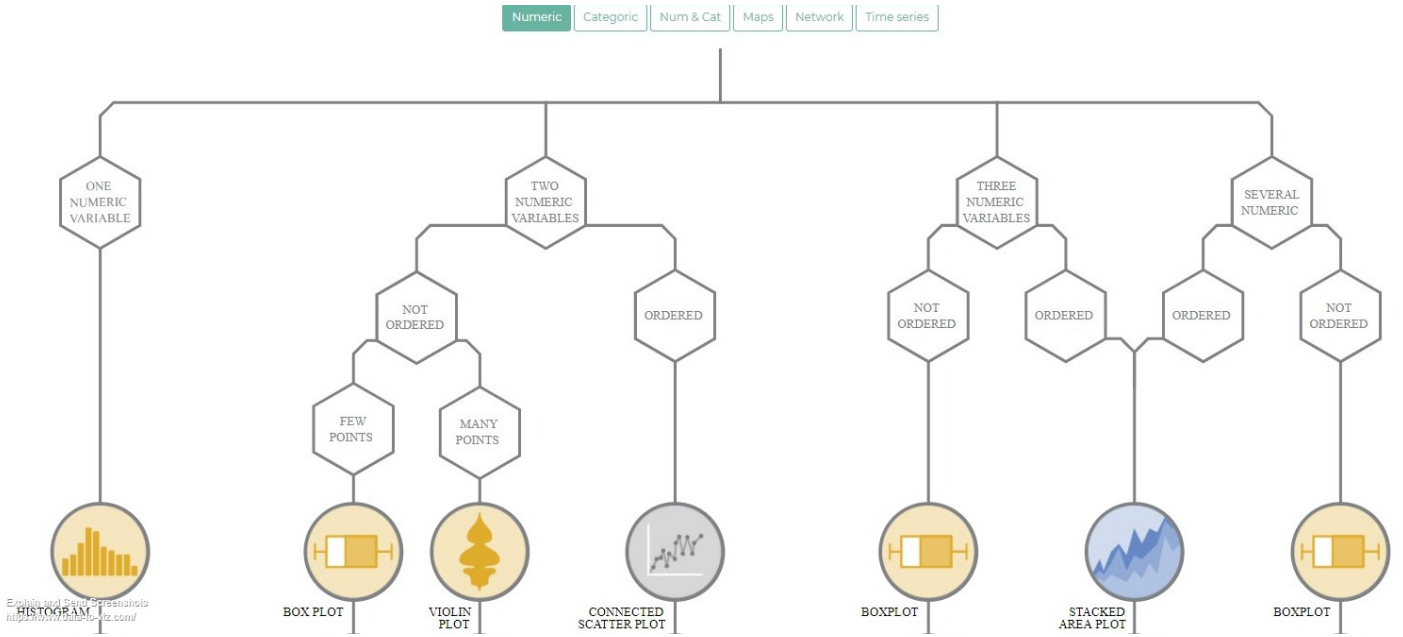
`geom_bar`

```
ggplot(data = mpg, aes(x = manufacturer)) +
  geom_bar(alpha = .3, col = "purple", fill = "red") # col = borders
```



Take a look at this site to choose a plot

https://www.data-to-viz.com/



Is everything clear ?

Exercises

~ 20 minutes

Exercises

~ 20 minutes

Open "ggplot2_basics" with Rstudio

GGPLOT2: Aesthetics / Facets / Labs

- One may want to :

- One may want to :

add color/transparency to the points

- One may want to :

add color/transparency to the points

do one plot for each manufacturer (or modality of a given factor)

- One may want to :

add color/transparency to the points

do one plot for each manufacturer (or modality of a given factor)

specify a title/subtitle to the plot

Aesthetics, basics

Aesthetic mapping vs. parameter setting

Aesthetic mapping:

- Data value determines visual characteristic
- use `aes()`

```
ggplot(diamonds, aes(x = carat, y = price, color = clarity)) + geom_p
```

Setting:

- Constant value determines visual characteristic
- Use parameter in `geom_<TYPE>`

```
ggplot(diamonds, aes(x = carat, y = price)) + geom_point(color = "red
```

- Let's take another dataset

```
data(diamonds)
data_work <- diamonds[sample(1:nrow(diamonds), 5000),] # take 10000 r
head(data_work)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.3  Very Good I     SI1      63.3    59   506  4.3   4.23  2.7
## 2  1    Ideal     E     SI2      61.8    57  4743  6.39  6.43  3.9
## 3  0.73 Fair      I     VS1      55.9    66  2330  6.11  6.01  3.3
## 4  0.38 Ideal     E     VS1      60.7    56   933  4.7   4.72  2.8
## 5  0.58 Ideal     F     VVS2     61.8    56  2263  5.37  5.41  3.3
## 6  1    Ideal     F     VVS2     61.9    57  9354  6.47  6.42  3.9
```

```
# See ?diamonds for the description of the dataset
```

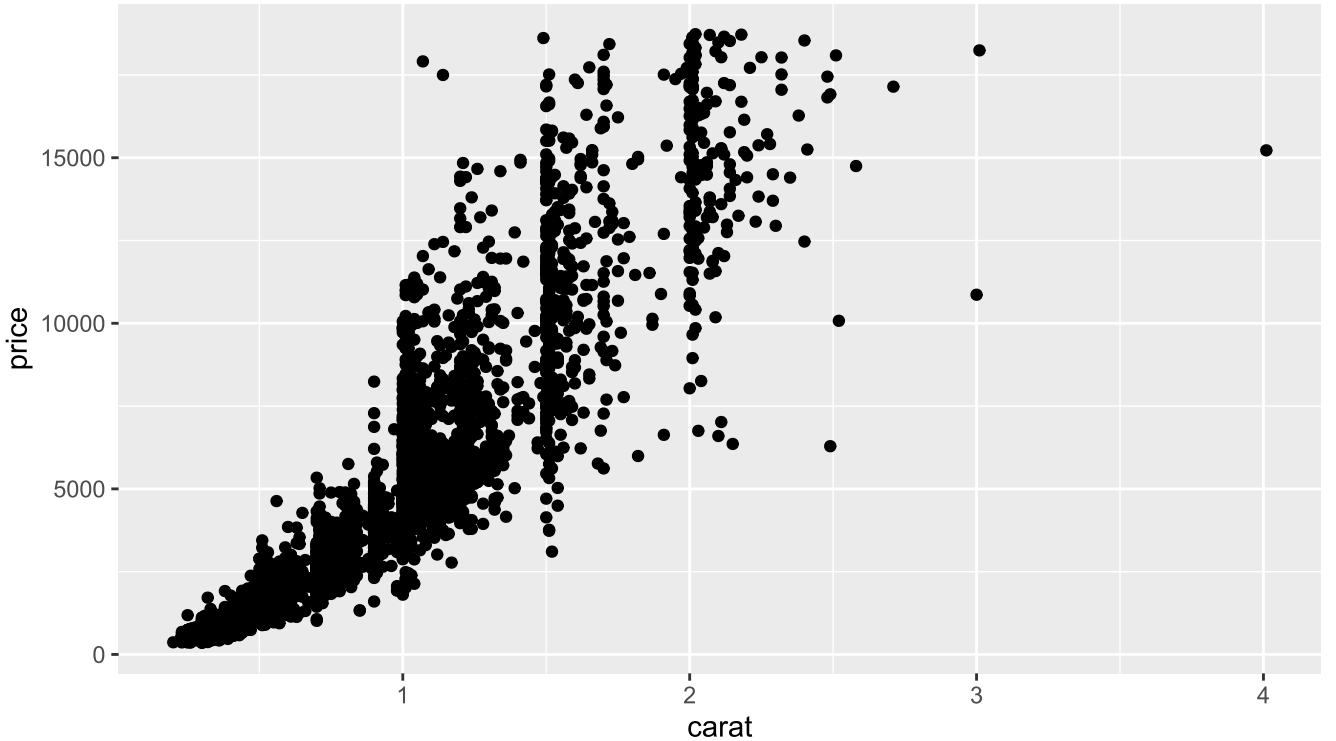
```
str(data_work)
```

```
## tibble [5,000 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:5000] 0.3 1 0.73 0.38 0.58 1 0.35 1 0.43 0.61 .
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 3 5 1 5 5 5 3
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 6 2 6 2 3 3
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 3 2 5 5 6 (
##  $ depth  : num [1:5000] 63.3 61.8 55.9 60.7 61.8 61.9 60.7 62.4 6:
##  $ table  : num [1:5000] 59 57 66 56 56 57 61 58 58 57 ...
##  $ price  : int [1:5000] 506 4743 2330 933 2263 9354 706 4574 1113
##  $ x      : num [1:5000] 4.3 6.39 6.11 4.7 5.37 6.47 4.54 6.37 4.8(
##  $ y      : num [1:5000] 4.23 6.43 6.01 4.72 5.41 6.42 4.58 6.42 4
##  $ z      : num [1:5000] 2.7 3.96 3.39 2.86 3.33 3.99 2.77 3.99 2.9
```

- Price vs carat ?

- Price vs carat ?

```
p_diamonds <- ggplot(data_work, aes(x = carat, y = price)) +
  geom_point()
p_diamonds
```

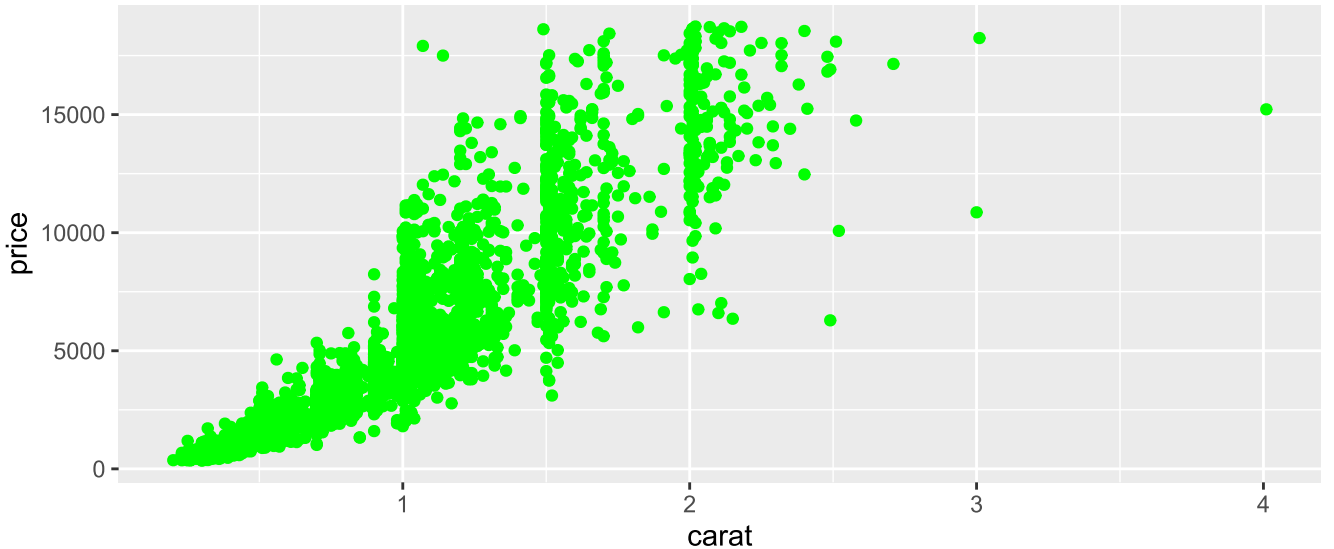


- We want the points to be green

- We want the points to be green
- This does not depend on the data $\rightarrow$ We specify the color argument *outside* `aes`

- We want the points to be green
- This does not depend on the data → We specify the color argument *outside* `aes`

```
p_diamonds <- ggplot(data_work, aes(x = carat, y = price)) +
  geom_point(col = "green")
p_diamonds
```

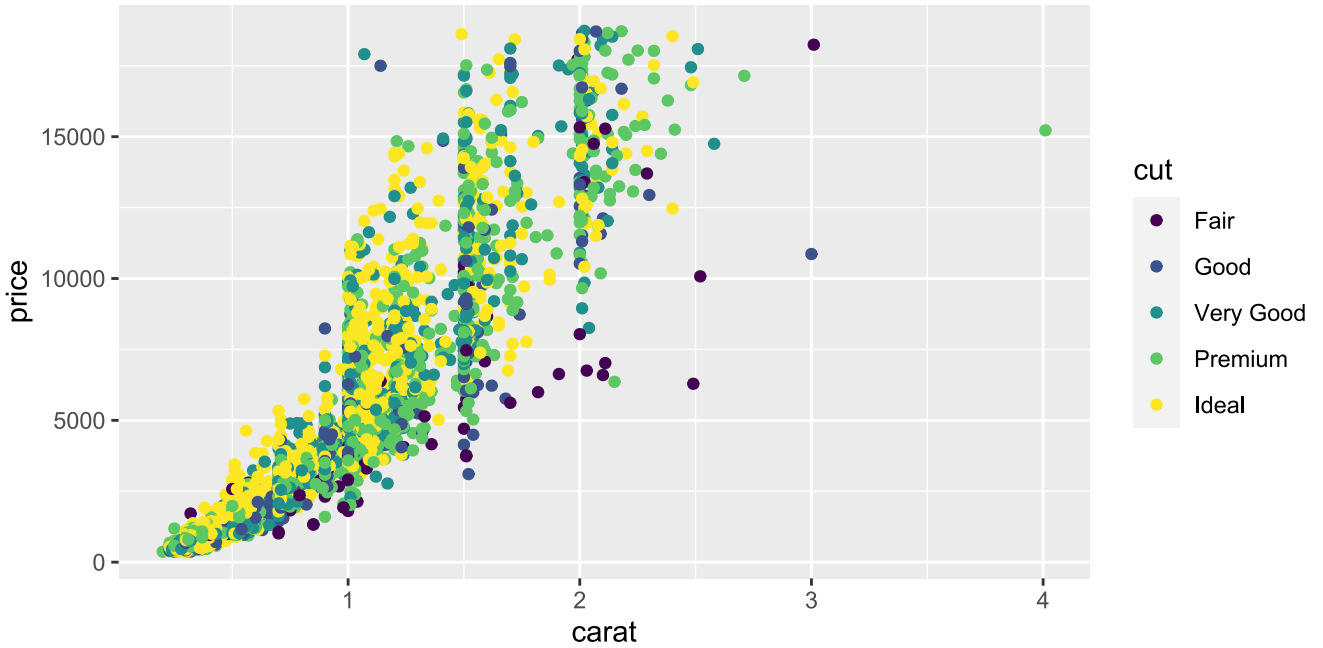


- One color for each cut of the diamond

- One color for each cut of the diamond
- This *depends* on the data $\rightarrow$ We specify the color argument *inside* `aes`

- One color for each cut of the diamond
- This *depends* on the data → We specify the color argument *inside* `aes`

```
p_diamonds <- ggplot(data_work, aes(x = carat, y = price)) +
  geom_point(aes(col = cut)) #Says to ggplot2, give one color to eac
p_diamonds
```

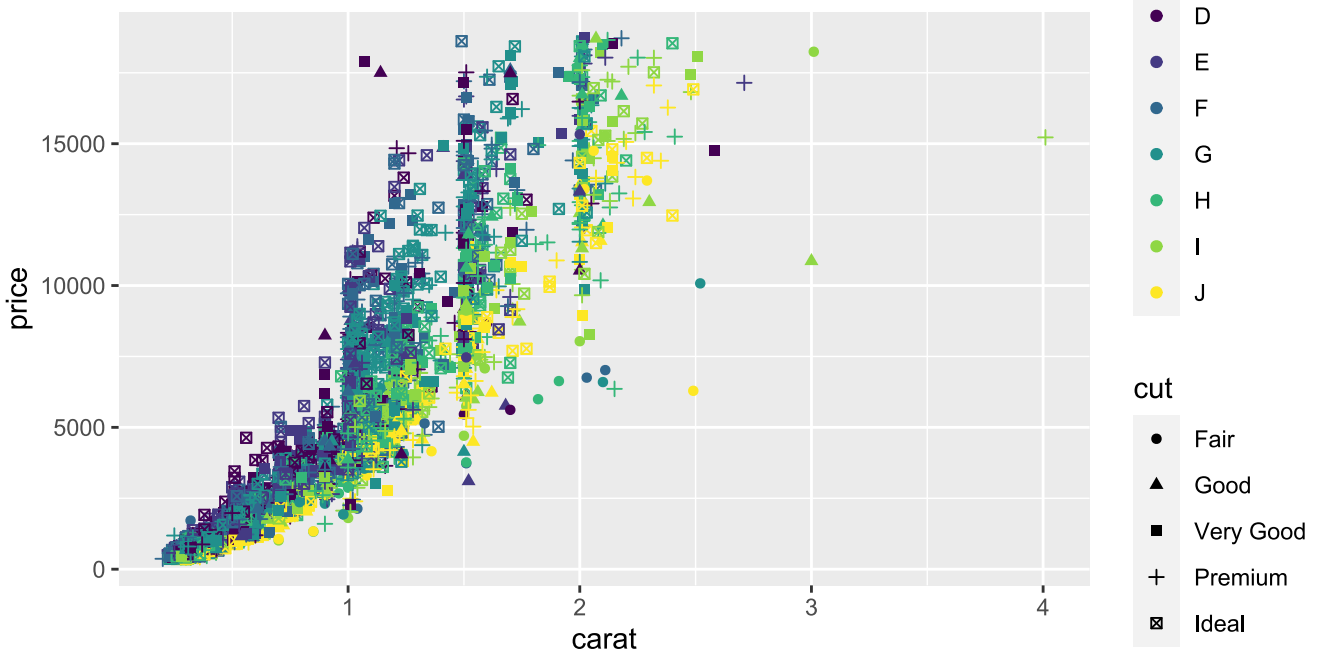


- One color for each for each color of the diamond, one shape for each cut of the diamond

- One color for each for each color of the diamond, one shape for each cut of the diamond
- This *depends* on the data $\rightarrow$ We specify the color and shape arguments *inside* `aes`

- One color for each for each color of the diamond, one shape for each cut of the diamond
- This *depends* on the data → We specify the color and shape arguments *inside* `aes`

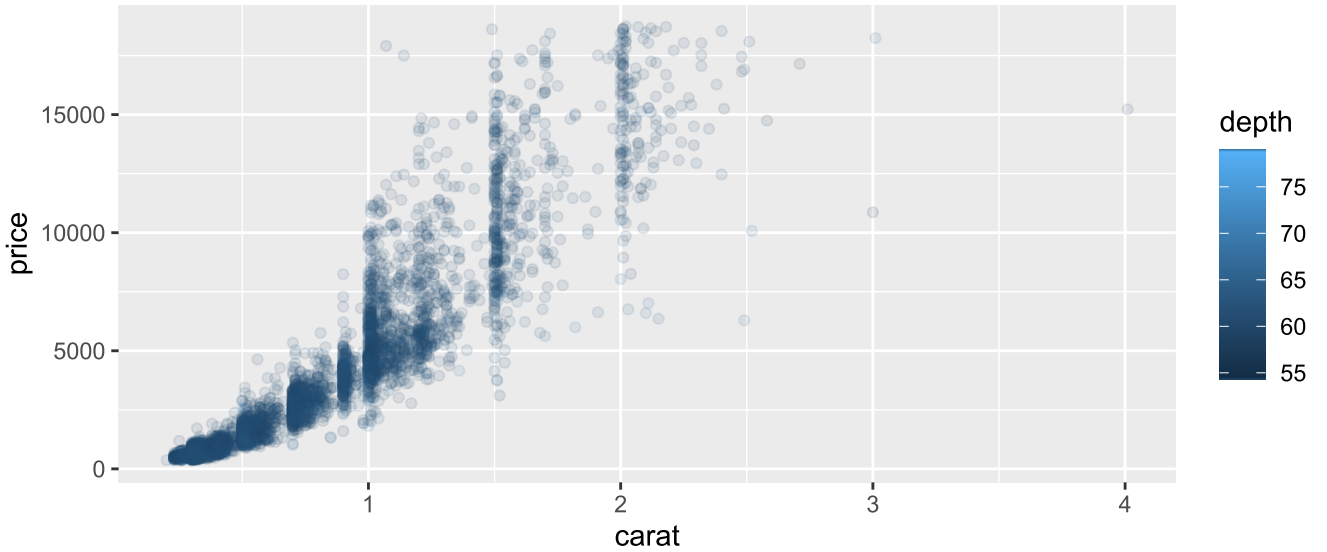
```
p_diamonds <- ggplot(data_work, aes(x = carat, y = price)) +
  geom_point(aes(col = color, shape = cut)) #Says to ggplot2, give o
p_diamonds
```



- If the variable mapped to color is continuous, then the scale of the colour becomes automatically continuous

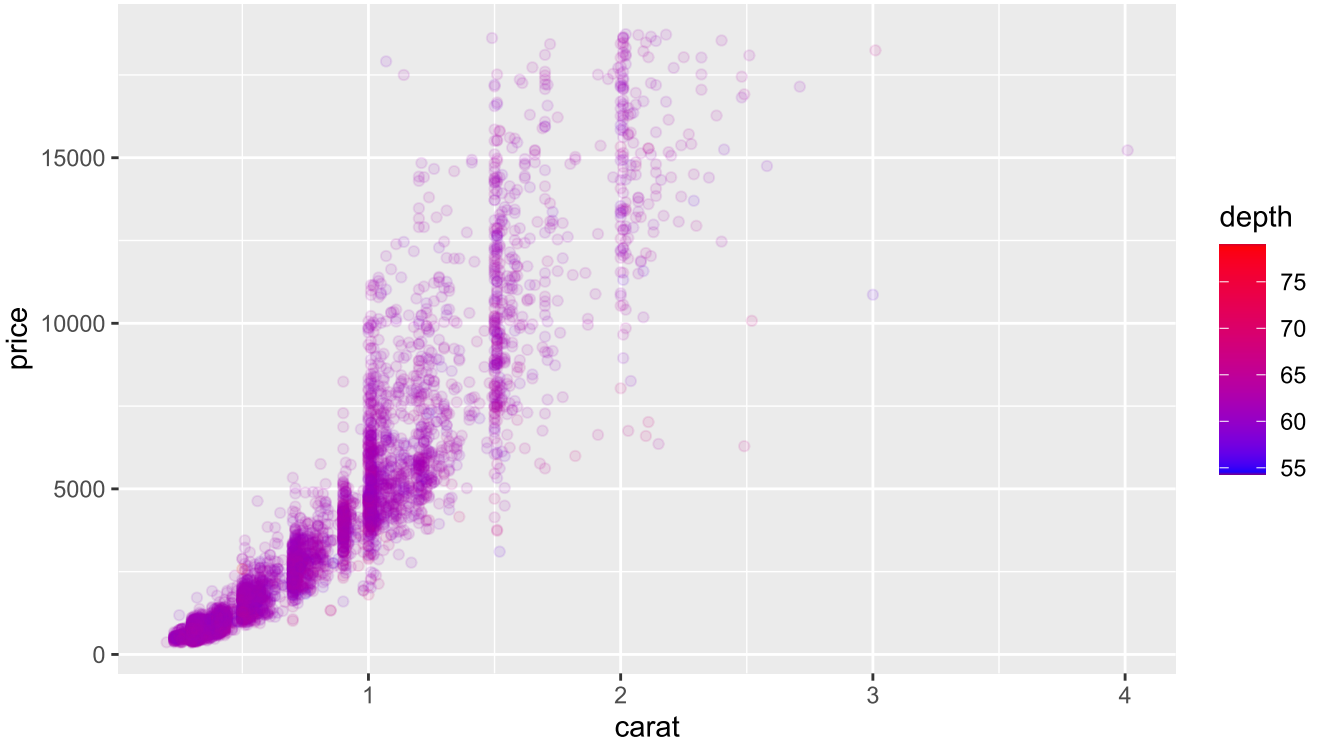
- If the variable mapped to color is continuous, then the scale of the colour becomes automatically continuous

```
p_diamonds <- ggplot(data_work, aes(x = carat, y = price)) +
  geom_point(aes(col = depth),
             alpha = .1) #Says to ggplot2, give one color based on th
#Put transparency on 0.1, regardless of the points' values
p_diamonds
```



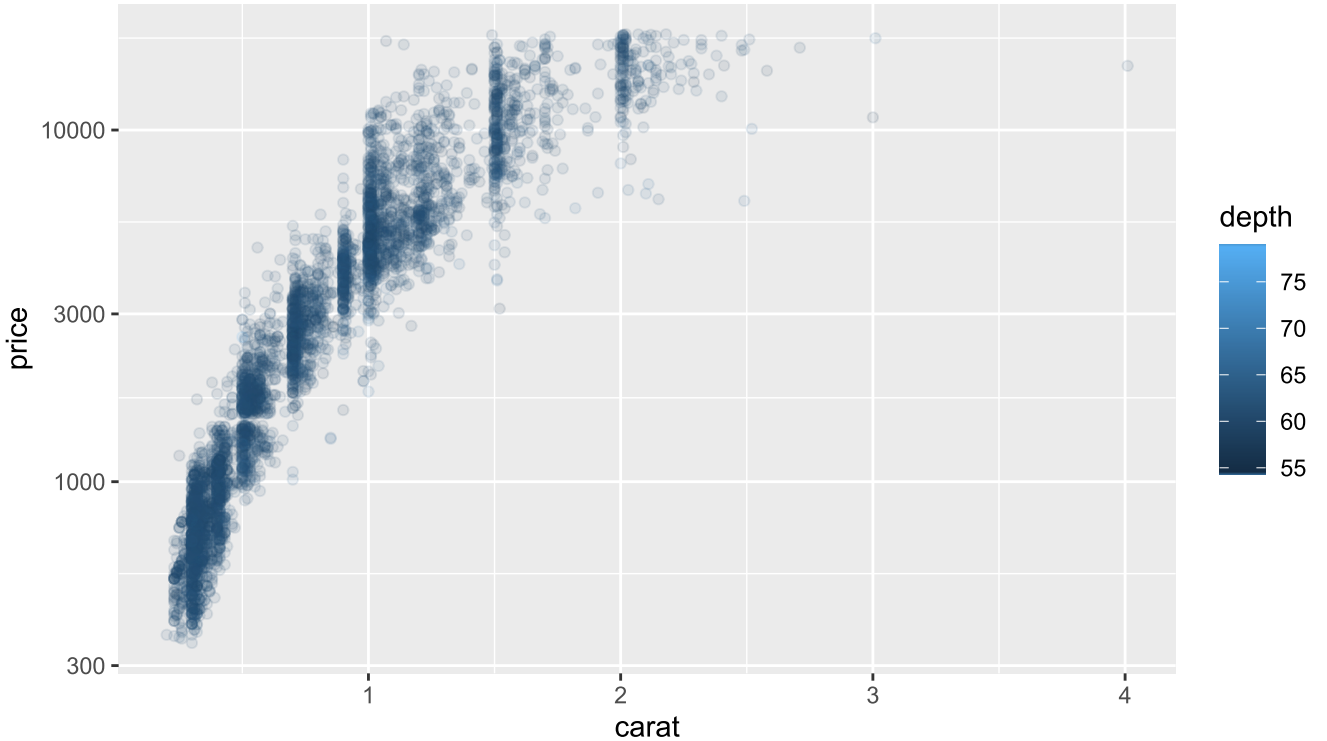
Working with scales (color boundaries)

```
p_diamonds + scale_color_continuous(low = "blue", high = "red")
```



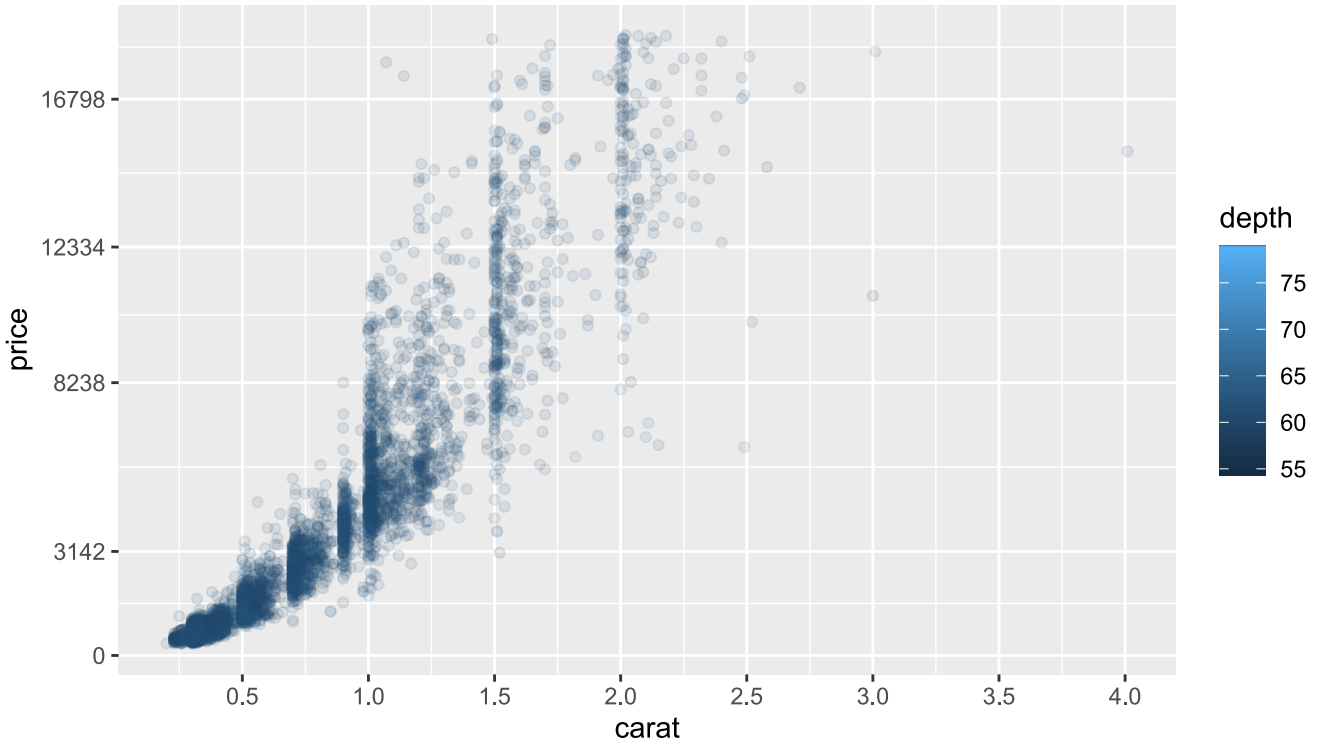
Working with scales: put a log scale

```
p_diamonds + scale_y_log10()
```



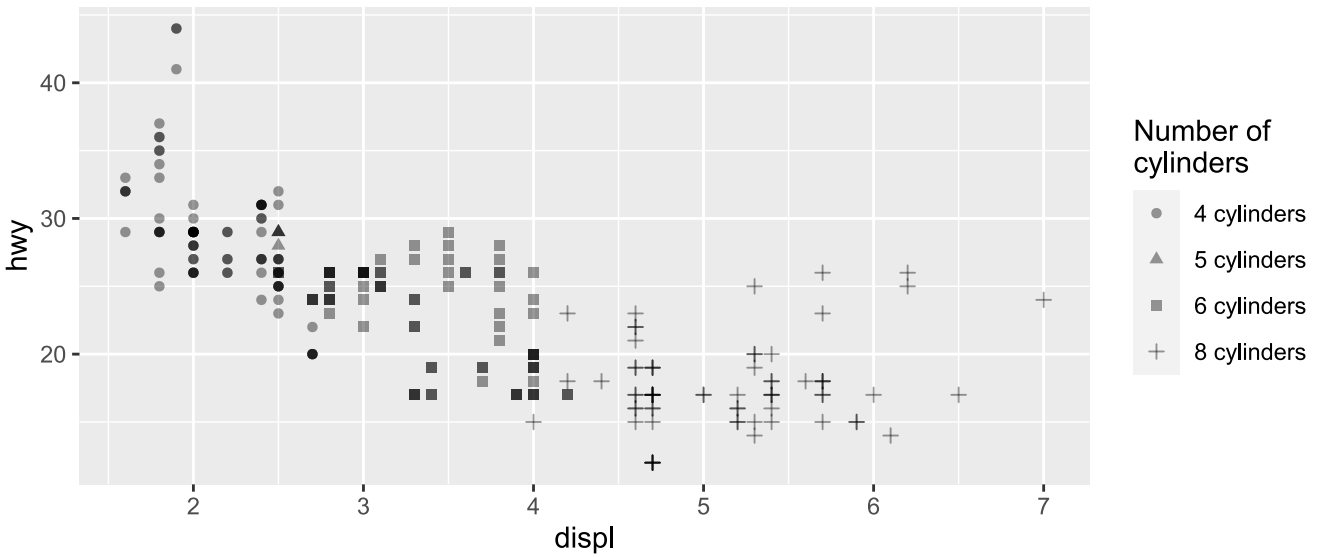
Working with scales: set the breaks of an axis

```
p_diamonds +
  scale_x_continuous(n.breaks = 8) +
  scale_y_continuous(breaks = c(0,3142, 8238,12334, 16798 ))
```



Working with scales: modify the shape's legend

```
p <- ggplot(data = mpg, aes(x =displ,  y= hwy ))+
  geom_point(aes(shape = factor(cyl)),
             alpha = .3) +
  scale_shape_discrete(labels = paste(c(4,5,6,8), "cylinders"),
                       name = "Number of\ncylinders")   # Modify labe
p
```



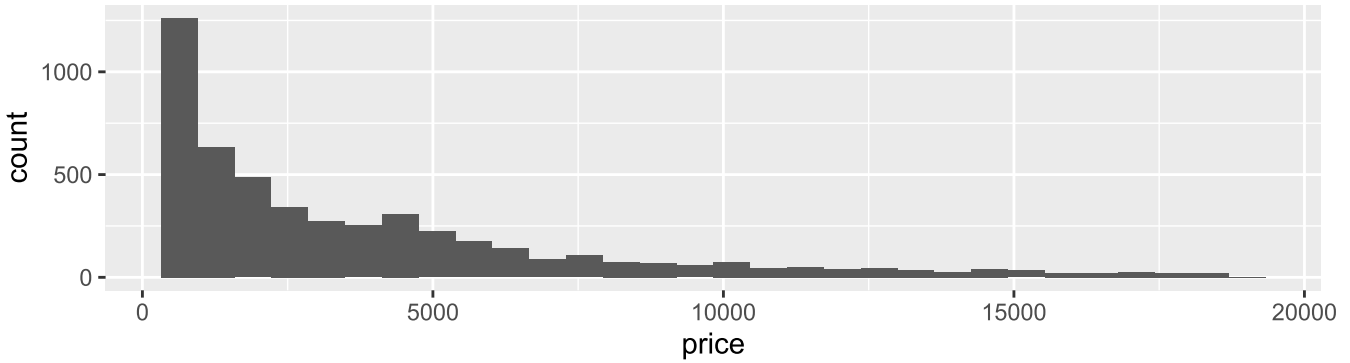
Facet: draw multiple plots easily

- Distribution of prices for the whole dataset:

Facet: draw multiple plots easily

- Distribution of prices for the whole dataset:

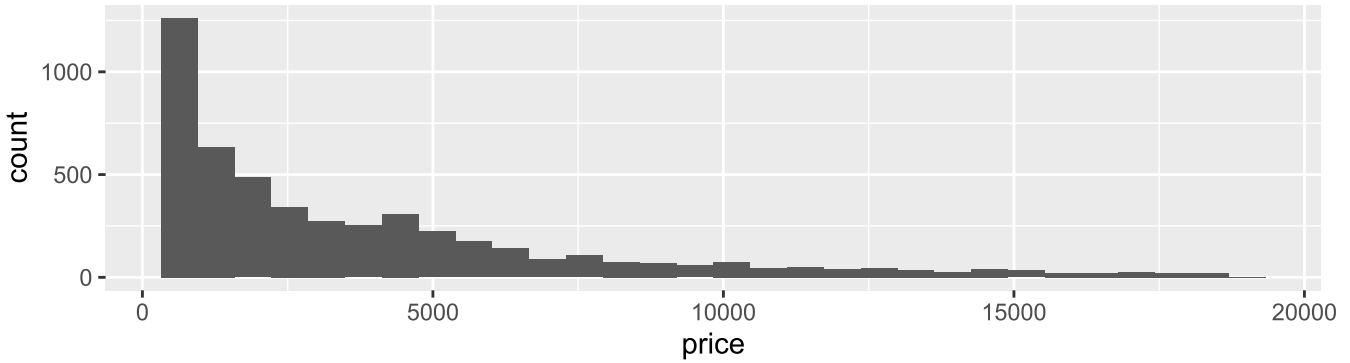
```
histogram_prices <- ggplot(data_work, aes(x = price)) + # No need for
  geom_histogram()
histogram_prices
```



Facet: draw multiple plots easily

- Distribution of prices for the whole dataset:

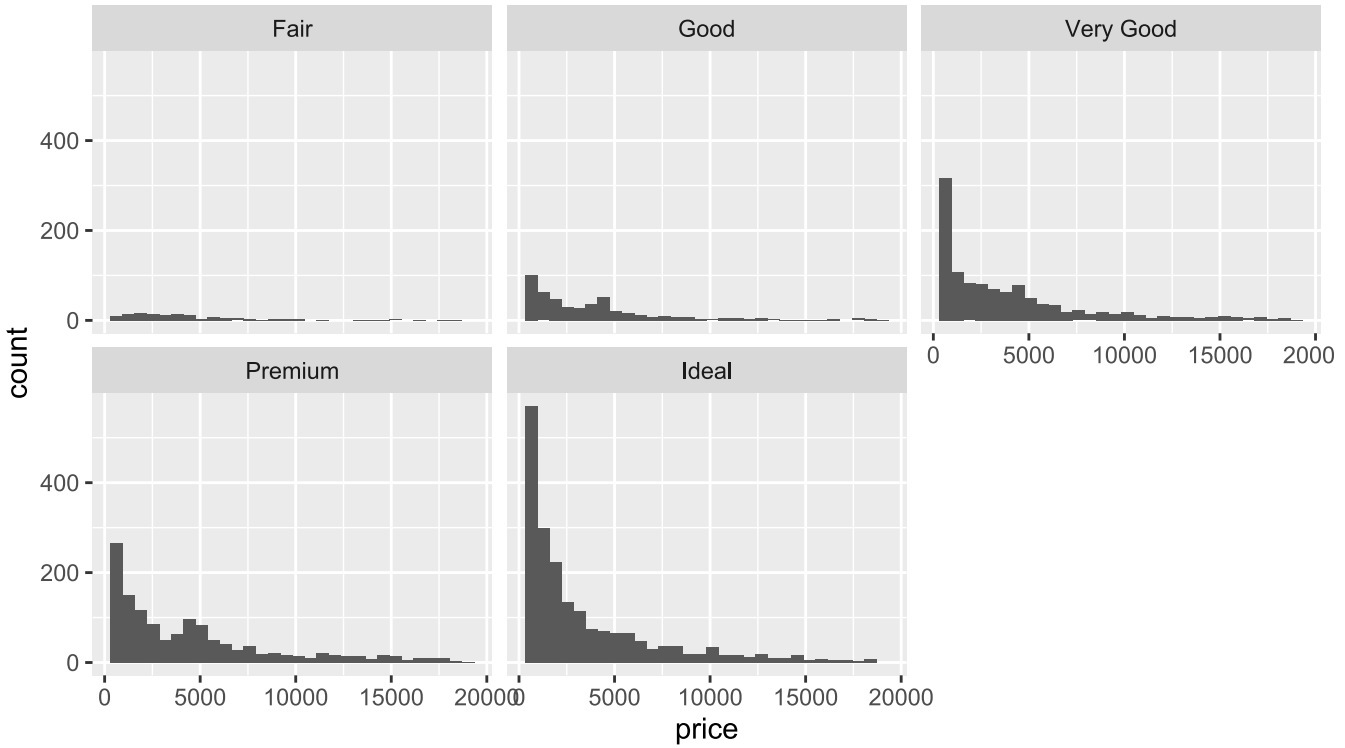
```
histogram_prices <- ggplot(data_work, aes(x = price)) + # No need for
  geom_histogram()
histogram_prices
```



- What if we want the distributions of prices *for each* modality of the *cut* variable

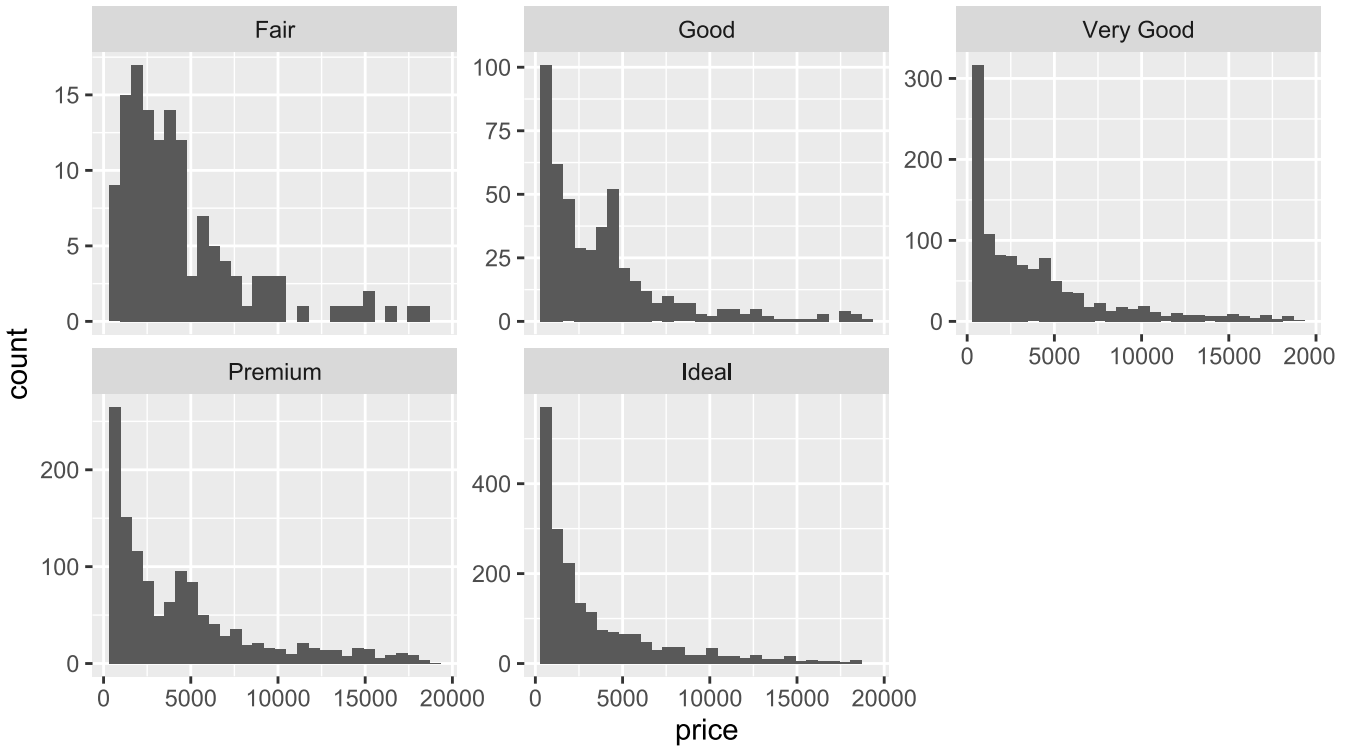
We simply need to use the `facet_wrap` function:

```
histogram_prices +
  facet_wrap(cut~. )
```



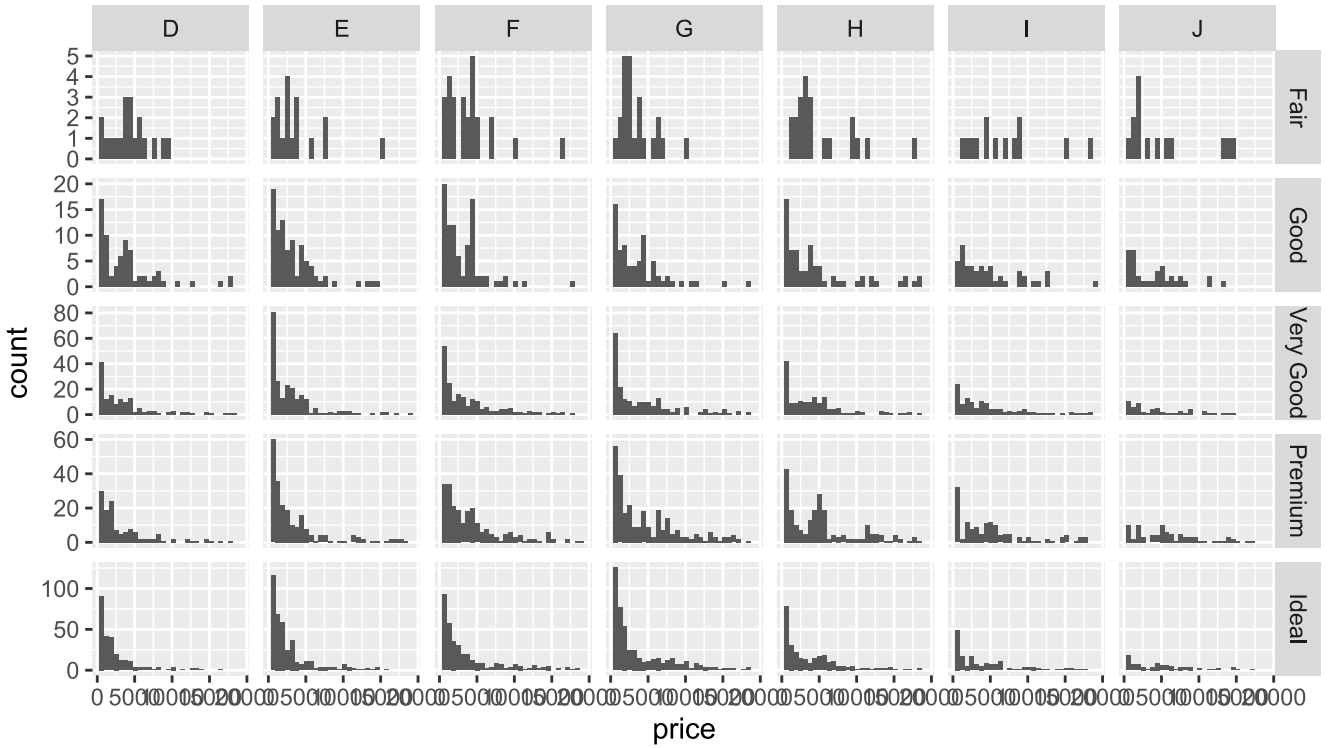
We can use the `scales` argument, to let scales vary freely across plots

```
histogram_prices +
  facet_wrap(cut~., scales = "free_y" ) # Arg "free_x" for free x ax
```



- We can also make a grid of plots, using `facet_grid` function

```
histogram_prices +
  facet_grid(cut~color,scales = "free_y")
```



Titles and subtitles: say what you show, and show what you say

- Many ways exist to specify titles in ggplot

Titles and subtitles: say what you show, and show what you say

- Many ways exist to specify titles in ggplot
- The easiest one may be to use the `labs` function

Titles and subtitles: say what you show, and show what you say

- Many ways exist to specify titles in ggplot
- The easiest one may be to use the `labs` function
- Let's take another dataset

```
data(economics)
head(economics)
```

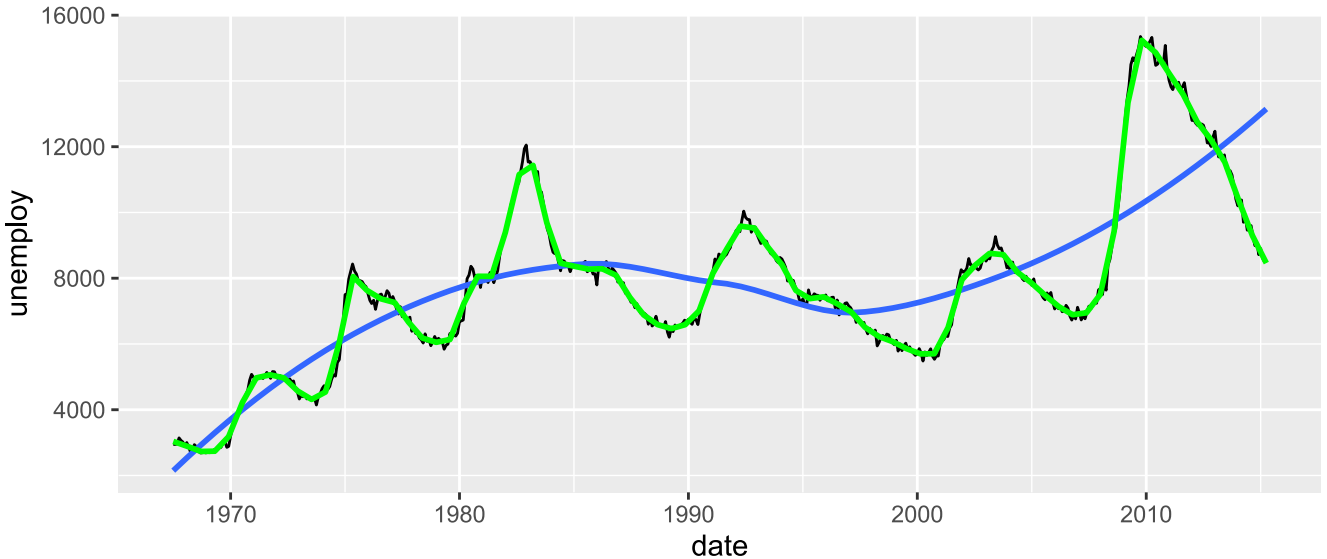
```
## # A tibble: 6 x 6
##   date         pce    pop psavert uempmed unemploy
##   <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
## 1 1967-07-01  507. 198712    12.6     4.5     2944
## 2 1967-08-01  510. 198911    12.6     4.7     2945
## 3 1967-09-01  516. 199113    11.9     4.6     2958
## 4 1967-10-01  512. 199311    12.9     4.9     3143
## 5 1967-11-01  517. 199498    12.8     4.7     3066
## 6 1967-12-01  525. 199657    11.8     4.8     3018
```

```
str(economics)
```

- Here is a plot, without any title (btw. note the `span` argument to make the fit closer to the points)

```
plot_unemployed <- ggplot(data = economics, aes(x  =date, y = unemplo
  geom_line() +
  geom_smooth(se = FALSE) +
  geom_smooth(span = 0.05, col = "green", se = FALSE )

plot_unemployed
```



- Adding a title is straightforward

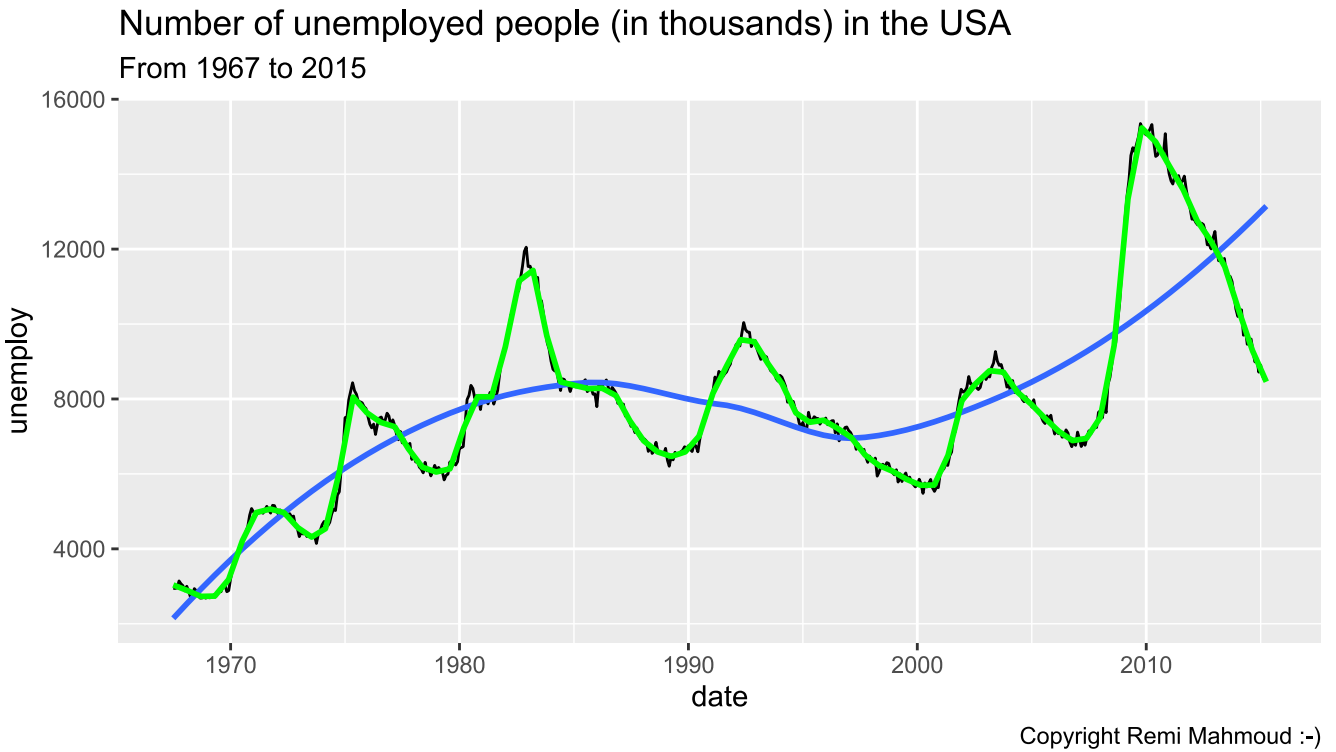
- Adding a title is straightforward

```
plot_unemployed +
  labs(title = "Number of unemployed people (in thousands) in the USA
```

- Adding subtitle/caption is also straightforward

- Adding subtitle/caption is also straightforward

```
plot_unemployed +
  labs(title = "Number of unemployed people (in thousands) in the USA
       subtitle = "From 1967 to 2015",
       caption = "Copyright Remi Mahmoud :-)")
```



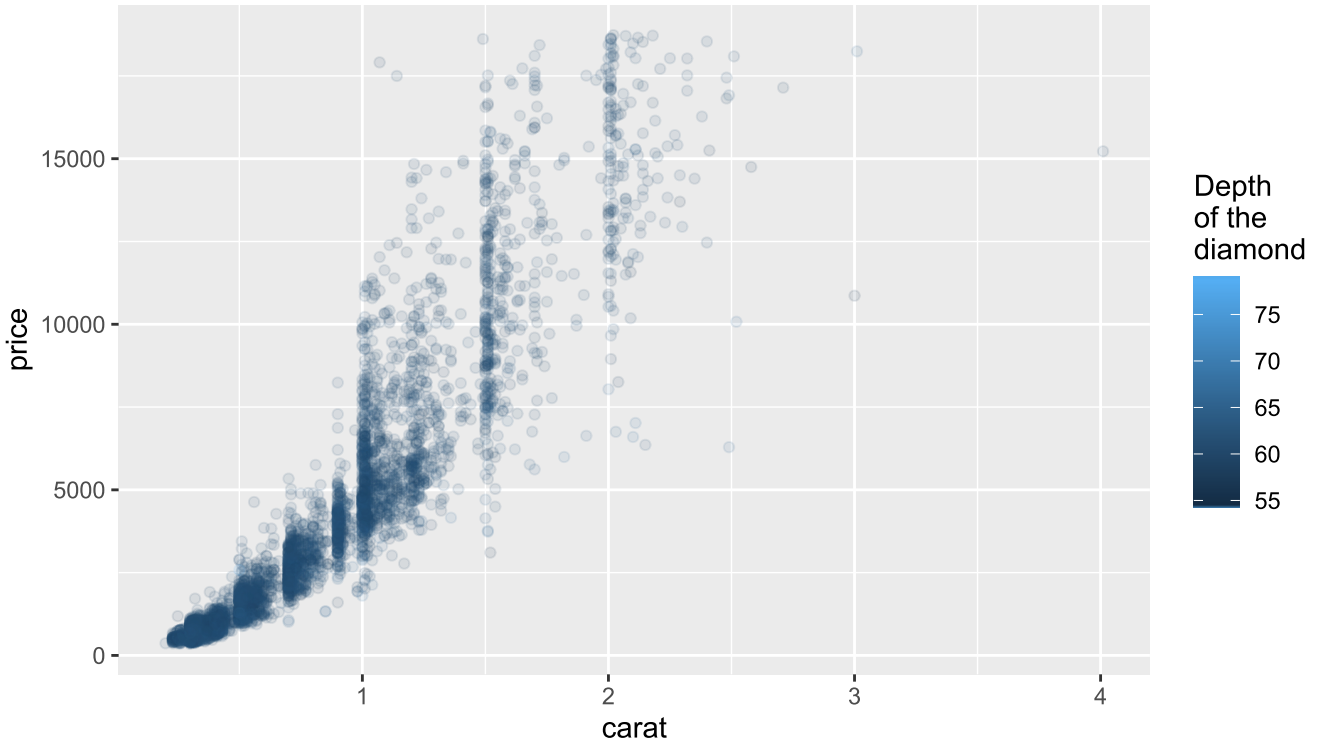
- Adding modifying axes titles is also straightforward

- Adding modifying axes titles is also straightforward

```
plot_unemployed +
  labs(title = "Number of unemployed people (in thousands) in the USA
       subtitle = "From 1967 to 2015",
       caption = "Copyright Remi Mahmoud :-)",
       x = "DATE", y = "Number of unemployed people")
```

- Title of a legend can also be changed using the `labs` function

```
p_diamonds+ labs(colour = "Depth\nof the\ndiamond")
```



Is everything clear ?


Break Time

Exercises

~ 20 minutes

Exercises

~ 20 minutes

Open "ggplot2_basics" with Rstudio

GGPLOT2: Customizing your plot

- One may want to :

- One may want to :

specify color / shapes of points / boxplots / bars

- One may want to :

specify color / shapes of points / boxplots / bars

change the sizes/font/orientation of labels / titles

- One may want to :

specify color / shapes of points / boxplots / bars

change the sizes/font/orientation of labels / titles

change the background

Themes: global appearance of the plot

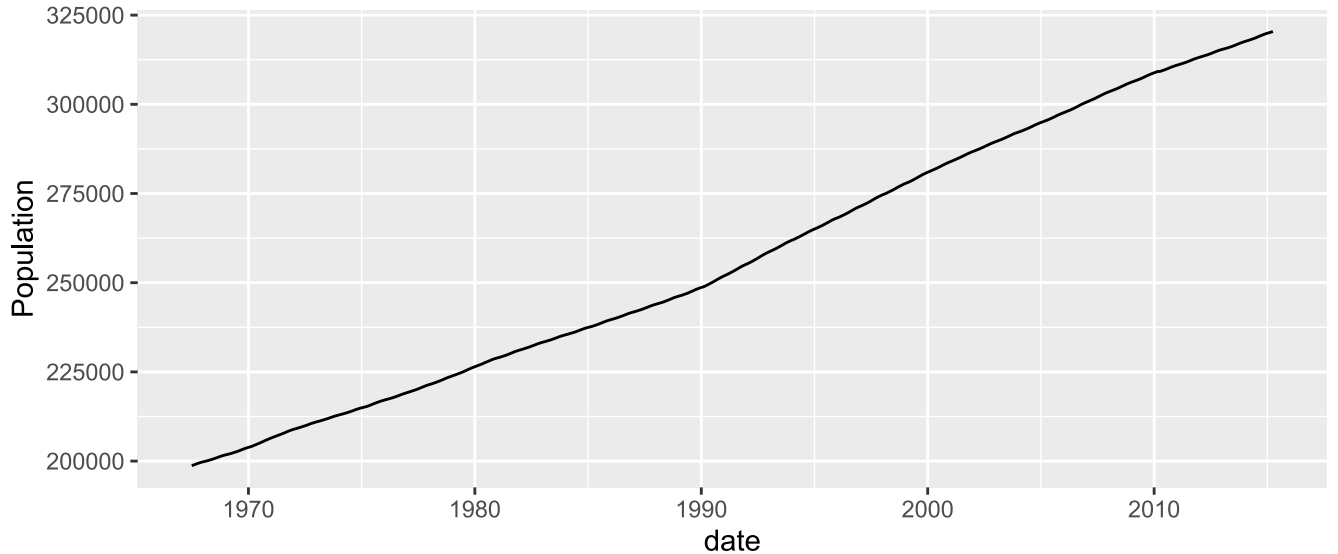
- You may have notice the (horrible) grey background of the default plots

Themes: global appearance of the plot

- You may have notice the (horrible) grey background of the default plots

```
plot_population <- ggplot(data = economics, aes(x  =date, y = pop)) +
  geom_line() +
  labs(y = "Population")

plot_population
```



Non-data elements, like background property, legend appearance, axis labels are handled by *themes* in `ggplot`.

Non-data elements, like background property, legend appearance, axis labels are handled by *themes* in `ggplot`.

We will se how to specify our own themes, but there are already some predefined themes (`theme_bw()`, `theme_minimal()`).

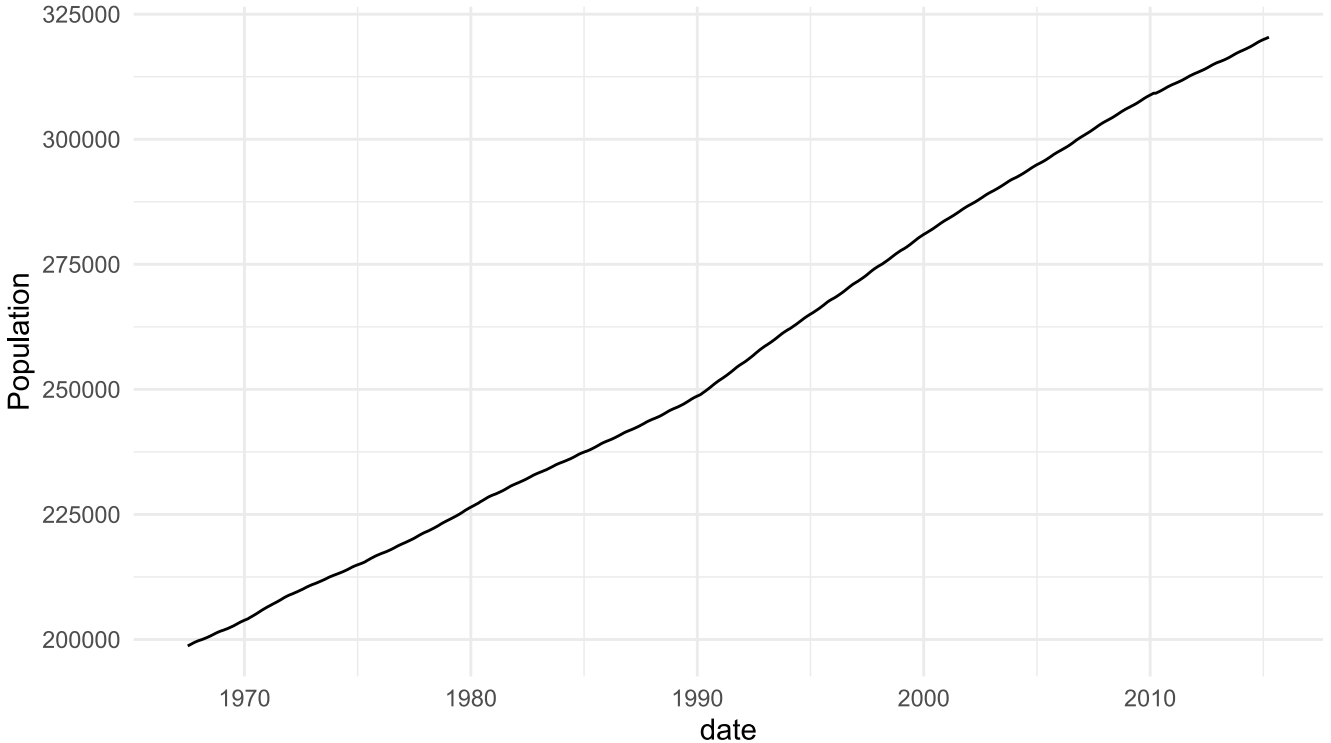
Non-data elements, like background property, legend appearance, axis labels are handled by *themes* in `ggplot`.

We will se how to specify our own themes, but there are already some predefined themes (`theme_bw()`, `theme_minimal()`).

To modify the theme of a plot, simply add the desired theme to the ggplot object

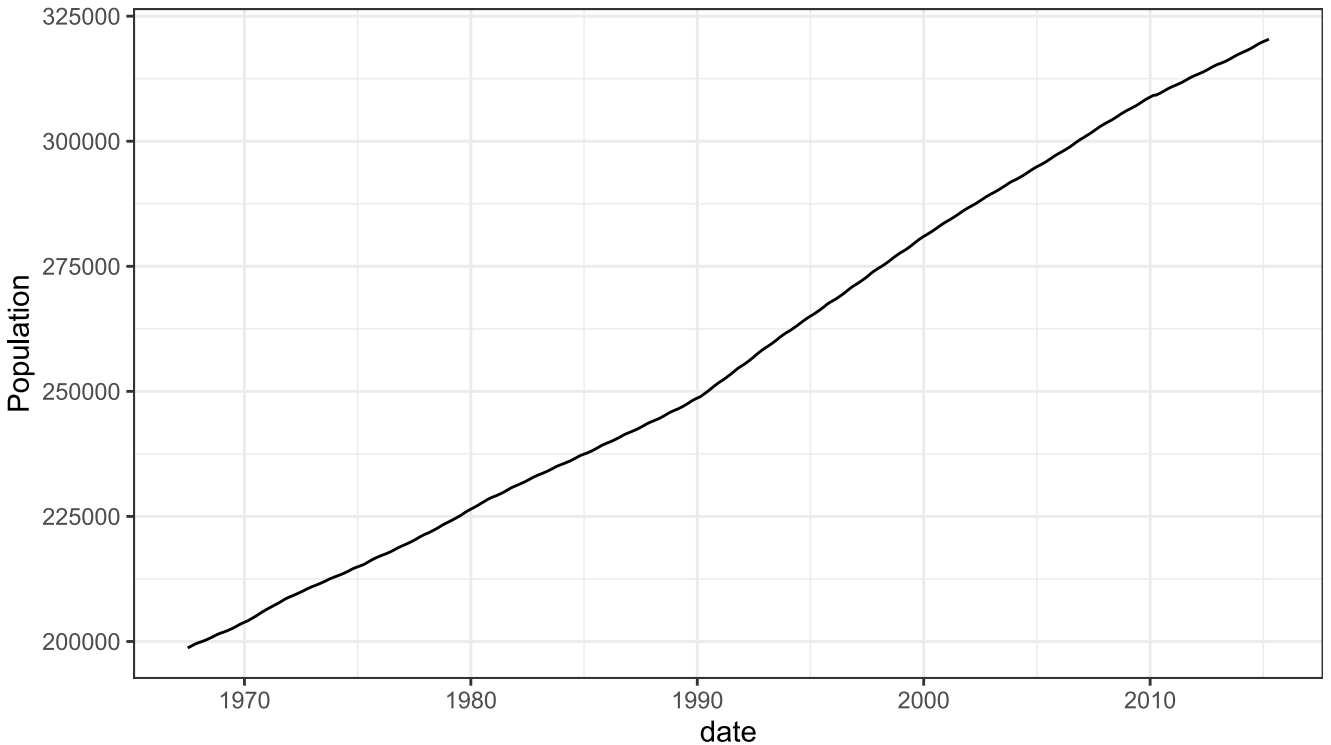
Predefined themes: theme_minimal()

```
plot_population + theme_minimal()
```



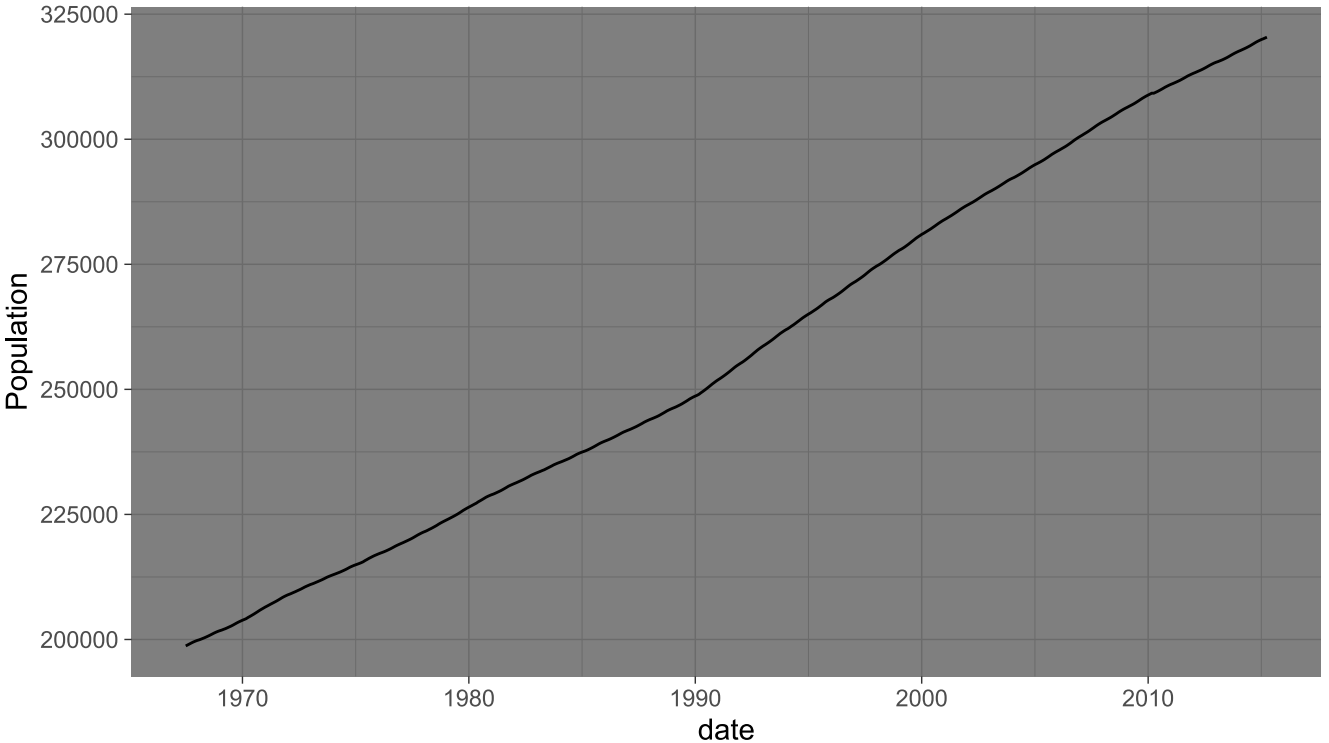
Predefined themes: theme_bw()

```
plot_population + theme_bw()
```



Predefined themes: theme_dark()

```
plot_population + theme_dark()
```



- Look at the (long) list of things you can modify in a theme

```
(attributes_theme <- names(theme_bw()))
```

```
##  [1] "line"                          "rect"
##  [3] "text"                          "title"
##  [5] "aspect.ratio"                  "axis.title"
##  [7] "axis.title.x"                  "axis.title.x.top"
##  [9] "axis.title.x.bottom"           "axis.title.y"
## [11] "axis.title.y.left"             "axis.title.y.right"
## [13] "axis.text"                     "axis.text.x"
## [15] "axis.text.x.top"               "axis.text.x.bottom"
## [17] "axis.text.y"                   "axis.text.y.left"
## [19] "axis.text.y.right"             "axis.ticks"
## [21] "axis.ticks.x"                  "axis.ticks.x.top"
## [23] "axis.ticks.x.bottom"           "axis.ticks.y"
## [25] "axis.ticks.y.left"             "axis.ticks.y.right"
## [27] "axis.ticks.length"             "axis.ticks.length.x"
## [29] "axis.ticks.length.x.top"       "axis.ticks.length.x.bottom"
## [31] "axis.ticks.length.y"           "axis.ticks.length.y.left"
## [33] "axis.ticks.length.y.right"     "axis.line"
## [35] "axis.line.x"                   "axis.line.x.top"
## [37] "axis.line.x.bottom"            "axis.line.y"
## [39] "axis.line.y.left"              "axis.line.y.right"
## [41] "legend.background"             "legend.margin"
## [43] "legend.spacing"                "legend.spacing.x"
## [45] "legend.spacing.y"              "legend.key"
## [47] "legend.key.size"               "legend.key.height"
```

- Some of the attributes of a `theme` regard text's elements (titles, axis labels etc.)

```
attributes_theme[grepl(attributes_theme, pattern = "text|title")]
```

```
##  [1] "text"                "title"               "axis.title"
##  [4] "axis.title.x"        "axis.title.x.top"    "axis.title.x.bot
##  [7] "axis.title.y"        "axis.title.y.left"   "axis.title.y.rig
## [10] "axis.text"           "axis.text.x"         "axis.text.x.top"
## [13] "axis.text.x.bottom"  "axis.text.y"         "axis.text.y.left
## [16] "axis.text.y.right"   "legend.text"         "legend.text.alig
## [19] "legend.title"        "legend.title.align"  "plot.title"
## [22] "plot.title.position" "plot.subtitle"       "strip.text"
## [25] "strip.text.x"        "strip.text.y"        "strip.text.y.lef
```

- Some of the attributes of a `theme` regard text's elements (titles, axis labels etc.)

```
attributes_theme[grepl(attributes_theme, pattern = "text|title")]
```

```
##  [1] "text"                "title"               "axis.title"
##  [4] "axis.title.x"        "axis.title.x.top"    "axis.title.x.bot
##  [7] "axis.title.y"        "axis.title.y.left"   "axis.title.y.rig
## [10] "axis.text"           "axis.text.x"         "axis.text.x.top"
## [13] "axis.text.x.bottom"  "axis.text.y"         "axis.text.y.left
## [16] "axis.text.y.right"   "legend.text"         "legend.text.alig
## [19] "legend.title"        "legend.title.align"  "plot.title"
## [22] "plot.title.position" "plot.subtitle"       "strip.text"
## [25] "strip.text.x"        "strip.text.y"        "strip.text.y.lef
```

- Texts elements are modified using the `element_text()` function

- Some of the attributes of a `theme` regard text's elements (titles, axis labels etc.)

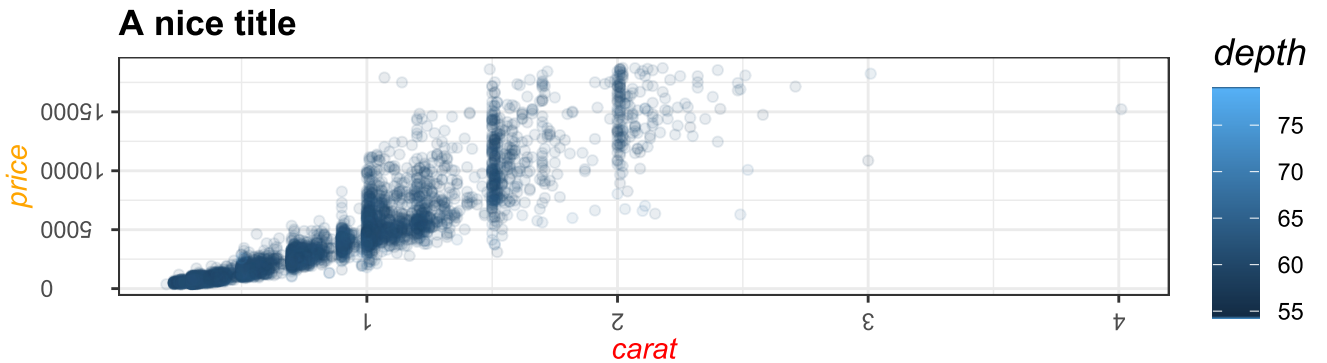
```
attributes_theme[grepl(attributes_theme, pattern = "text|title")]
```

```
##  [1] "text"                "title"               "axis.title"
##  [4] "axis.title.x"        "axis.title.x.top"    "axis.title.x.bot
##  [7] "axis.title.y"        "axis.title.y.left"   "axis.title.y.rig
## [10] "axis.text"           "axis.text.x"         "axis.text.x.top"
## [13] "axis.text.x.bottom"  "axis.text.y"         "axis.text.y.left
## [16] "axis.text.y.right"   "legend.text"         "legend.text.alig
## [19] "legend.title"        "legend.title.align"  "plot.title"
## [22] "plot.title.position" "plot.subtitle"       "strip.text"
## [25] "strip.text.x"        "strip.text.y"        "strip.text.y.lef
```

- Texts elements are modified using the `element_text()` function
- `element_text()` can set the font, the color, the angle of the text setted (see `?element_text()` for a complete list)

Modify texts elements

```
p_diamonds +  labs(title = "A nice title") +
  theme_bw() +
  theme(axis.title.x = element_text(color = "red"), # Modify title o
          axis.title.y = element_text(color = "orange"), # Modify ti
        axis.text = element_text(angle = 180), # Modify the angle of
        legend.title = element_text(size = 14 ),# Modify the size of
        plot.title = element_text(face = "bold"), # Modify the face
        title = element_text(face = "italic")) # Modify the face of
```



Modify background of elements

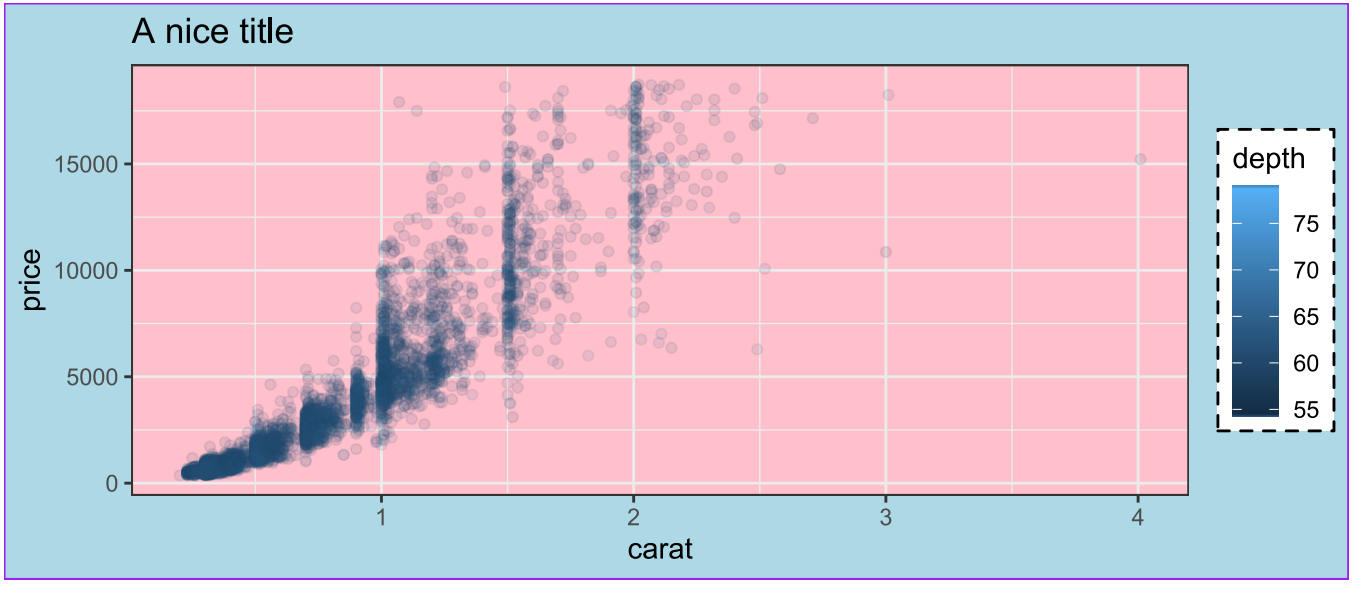
- Background elements are modified using the `element_rect()` function

```
attributes_theme[grepl(attributes_theme, pattern = "background")]
```

```
## [1] "legend.background"     "legend.box.background" "panel.backgrou
## [4] "plot.background"       "strip.background"      "strip.backgrou
## [7] "strip.background.y"
```

```
p_diamonds +  labs(title = "A nice title") +
  theme_bw() +
  theme(legend.background = element_rect(color = "black", linetype =
        plot.background = element_rect(fill = "lightblue", colour = '
        panel.background = element_rect(fill = "pink")) #guess what
```

```
p_diamonds +  labs(title = "A nice title") +
  theme_bw() +
  theme(legend.background = element_rect(color = "black", linetype =
        plot.background = element_rect(fill = "lightblue", colour = '
        panel.background = element_rect(fill = "pink")) #guess what
```



Save your plot using `ggsave`

- Saving your plot at a fixed size is a good practice

Save your plot using `ggsave`

- Saving your plot at a fixed size is a good practice

```
#Many parameters exist to save your plots
ggsave(filename = "my_path/title_of_the_plot.jpg",
       plot = my_plot, units = "cm",
       height = 25, width = 20)
```

Is everything clear ?

Exercises

~ 20 minutes

Exercises

~ 20 minutes

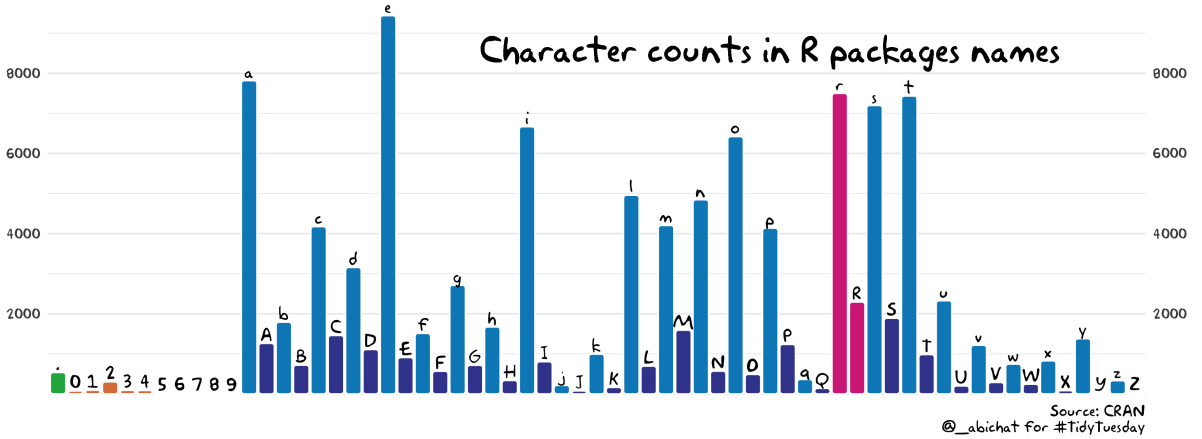
Open "ggplot2_customization" with Rstudio

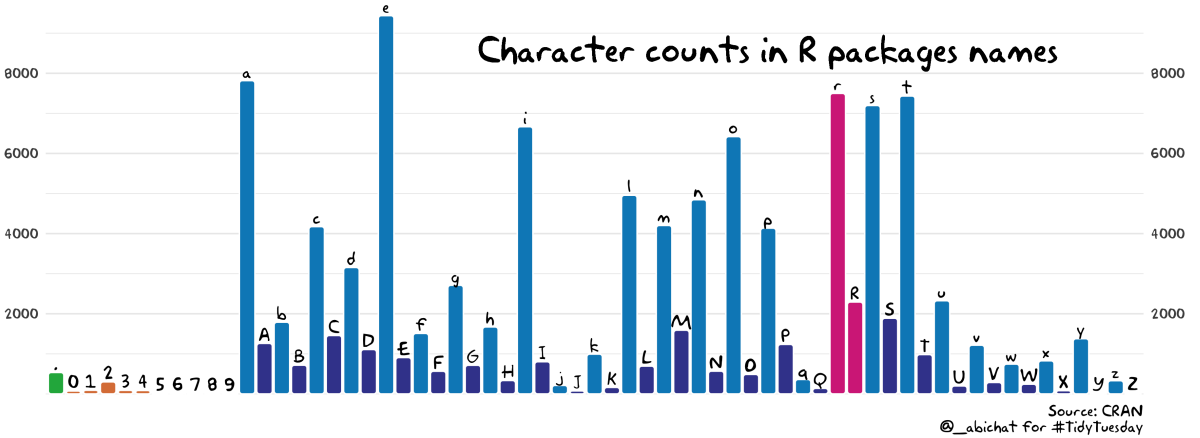
Conclusion

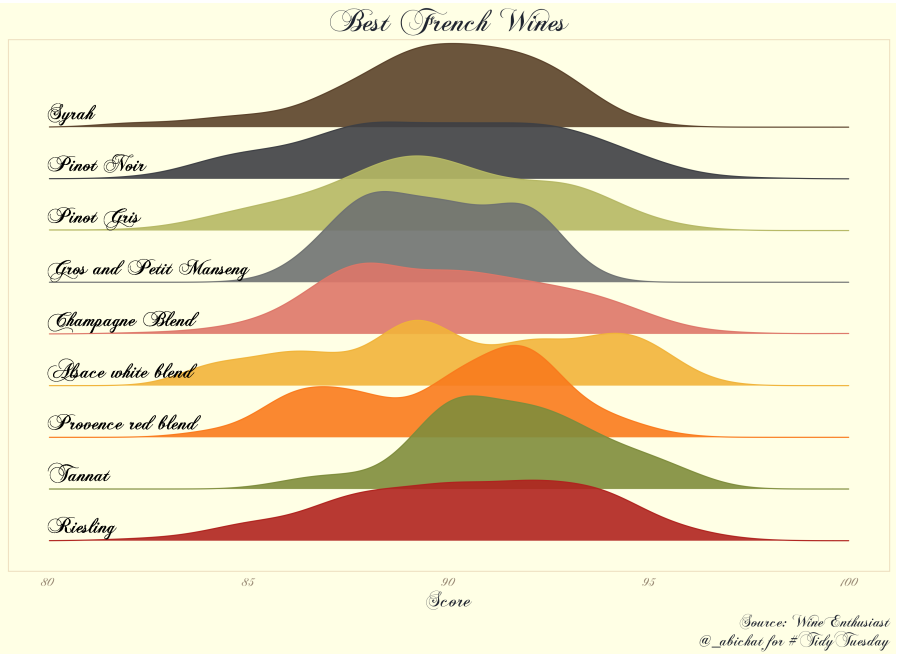
- This was just an introduction

- This was just an introduction
- ggplot2 offer *many* possibilities

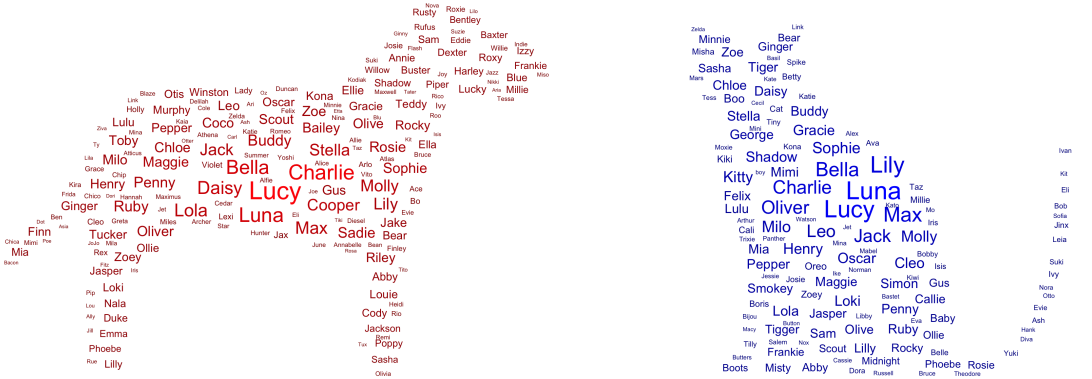
- This was just an introduction
- ggplot2 offer *many* possibilities
- Here are some wonderful plots by @abichat to give you some tastes


Character counts in R packages names
8000
6000
4000
2000
. 0 1 2 3 4 5 6 7 8 9
a A b B c C d D e E f F g G h H i I j J k K l L m M n N o O p P q Q r R s S t T u U v V w W x X y Y z Z
Source: CRAN
@_abichat for #TidyTuesday


Character counts in R packages names
8000
6000
4000
2000
Source: CRAN
@_abichat for #TidyTuesday

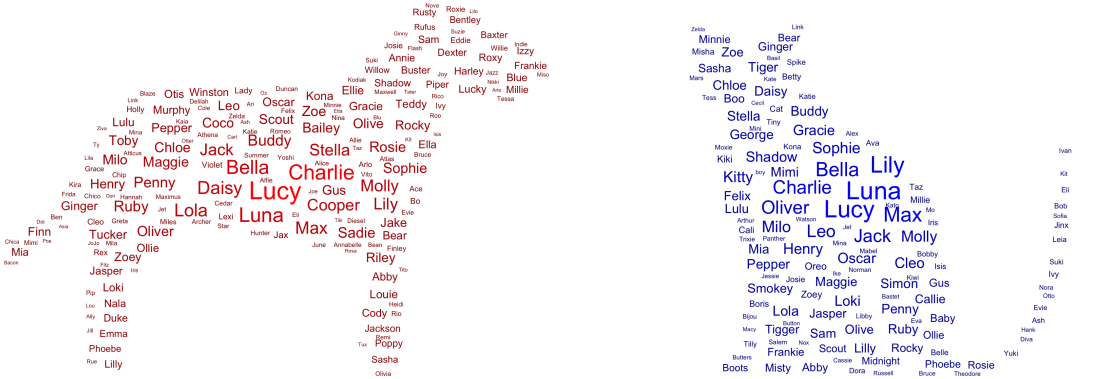

Best French Wines
Syrah
Pinot Noir
Pinot Gris
Gros and Petit Manseng
Champagne Blend
Alsace white blend
Provence red blend
Tannat
Riesling
80
85
90
95
100
Score
Source: Wine Enthusiast
@_abichat for #TidyTuesday

Most used names for dogs and cats in Seattle



Source: Seattle's open data portal
@_abichat for #TidyTuesday

Most used names for dogs and cats in Seattle



Source: Seattle's open data portal
@_abichat for #TidyTuesday

Daily amount of plastic waste entering the ocean

Source: Our World in Data
@_abichat for #TidyTuesday

ggplot2 resources online

- wonderful and complete article about ggplot: my holy bible for ggplot2
- a bunch of tutorials on ggplot2, really clear
- the ggplot2 book, recall and explanations

References

- a lesson from N. Vialaneix
- the ggplot2 book
- wonderful plots of A. Bichat: go take a look !

THANKS

THANKS

Any remark, questions ?

THANKS

Any remark, questions ?

remi.mahmoud@inrae.fr

THANKS

Any remark, questions ?

remi.mahmoud@inrae.fr

Lesson contents available at:
https://github.com/RemiMahmoud/ggplot2_lesson