Lecture - Basics of ggplot2

Data Visualisation

dr. Sanne Willems

Set-up

General focus: basics in ggplot

You:

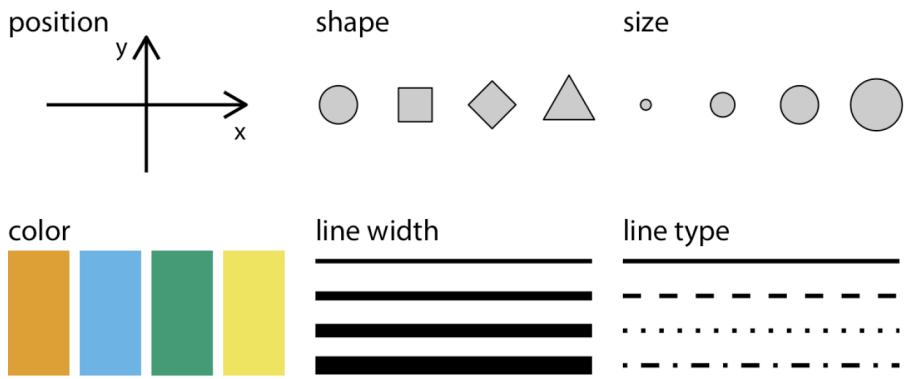
- Open provided template
- Run the code while watching the video

Mapping data onto aesthetics

Aesthetics

Aesthetics describe every aspect of a graphical element, for example:

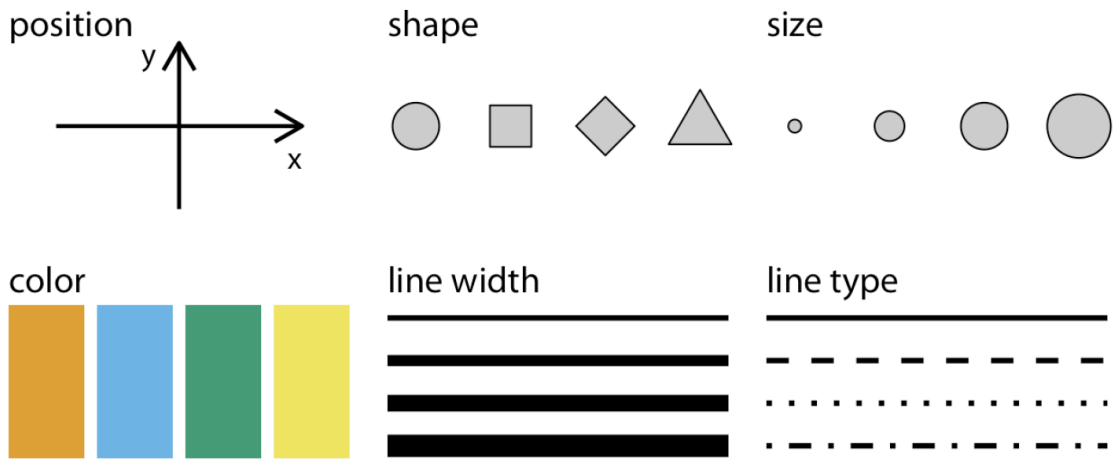
- position
- shape
- size
- color
- transparency
- line width
- line type
- etc.



Aesthetics for different types of data

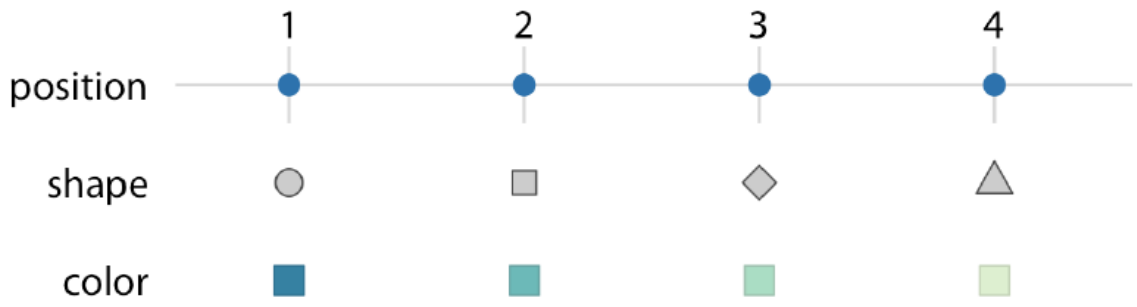
Some of these aesthetics can be used for continuous data, and others are better for categorical data, for example:

- Continuous data: position, size, color, line width
- Categorical data: shape, line type, color



Scales

Scales define a unique (1-to-1) mapping between data and aesthetics



Mapping data onto aesthetics - ggplot2

ggplot2 package:

- map data values onto aesthetics
- via scales, i.e. specify which data values correspond to which specific aesthetics values

Data sets

Data set: Motor Trend, 1974

- data on 32 cars
- available in R

Number of cylinders and weight varies per car:

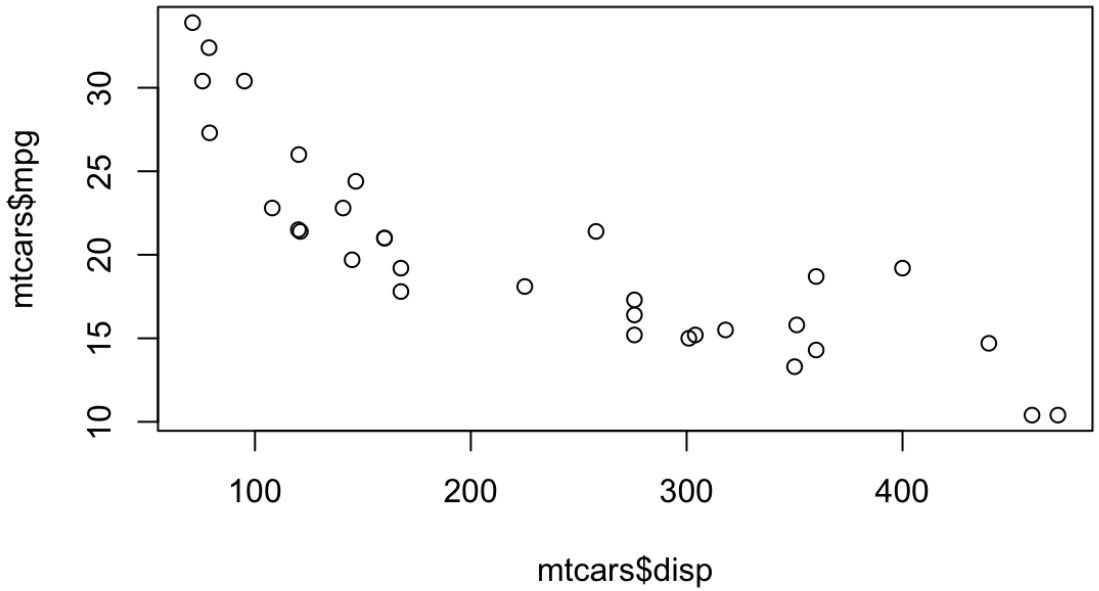
```
summary(mtcars[,c("cyl", "wt")])
```

```
##       cyl             wt
##  Min.   :4.000   Min.   :1.513
##  1st Qu.:4.000   1st Qu.:2.581
##  Median :6.000   Median :3.325
##  Mean   :6.188   Mean   :3.217
##  3rd Qu.:8.000   3rd Qu.:3.610
##  Max.   :8.000   Max.   :5.424
```

Data set: Motor Trend, 1974

We can look at the relationship between the car's displacement and its fuel efficiency:

```
plot(x = mtcars$disp, y = mtcars$mpg)
```



Data set: Tech stocks

- stock data for four tech companies
- available as a .rda file

```
load("Data/tech_stocks.rda")
stocks <- as.data.frame(tech_stocks)
stocks$date <- as.Date(stocks$date,
                       format = "%y-%m-%d")
```

Data set: Tech stocks

```
summary(stocks)
```

```
##    company             ticker              date                price
##  Length:9569        Length:9569        Min.   :2006-06-06   Min.   :  7.24
##  Class :character   Class :character   1st Qu.:2009-08-05   1st Qu.: 29.27
##  Mode  :character   Mode  :character   Median :2012-08-29   Median : 65.71
##                                        Mean   :2012-04-26   Mean   :154.32
##                                        3rd Qu.:2015-01-16   3rd Qu.:226.74
##                                        Max.   :2017-06-02   Max.   :975.88
##   index_price     price_indexed
##  Min.   : 27.72   Min.   :  9.034
##  1st Qu.: 28.45   1st Qu.: 83.254
##  Median : 80.14   Median :105.167
##  Mean   :117.54   Mean   :134.841
##  3rd Qu.:285.20   3rd Qu.:165.873
##  Max.   :285.20   Max.   :554.149
```

ggplot2 basics

ggplot function

- package: ggplot2
- main function: ggplot
- asks for “data” and “mapping”

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

```
## function (data = NULL, mapping = aes(), ..., environment = parent.frame())
## {
##     UseMethod("ggplot")
## }
## <bytecode: 0x7fe04483fd80>
## <environment: namespace:ggplot2>
```

ggplot function: data

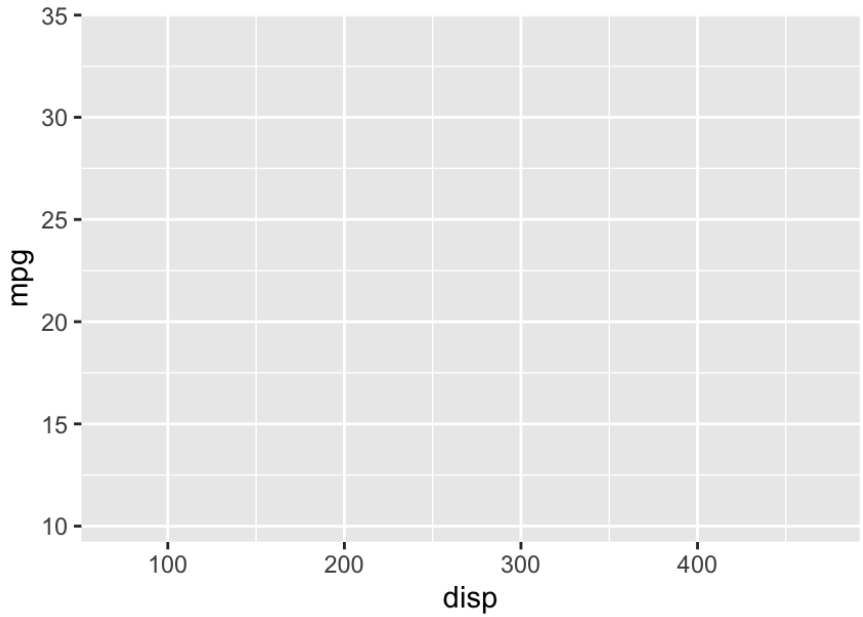
Only including the data already gives a grey area of where the plot will be:

```
ggplot(data = mtcars)
```

ggplot function: aesthetics

The function needs to know which aesthetics to use and for which variables:

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg))
```



ggplot function: geometrical layer

- specify the type of plot
- many different types of plot
  - all functions start with “geom”: geom_*()
  - examples: https://ggplot2.tidyverse.org/reference/#layers

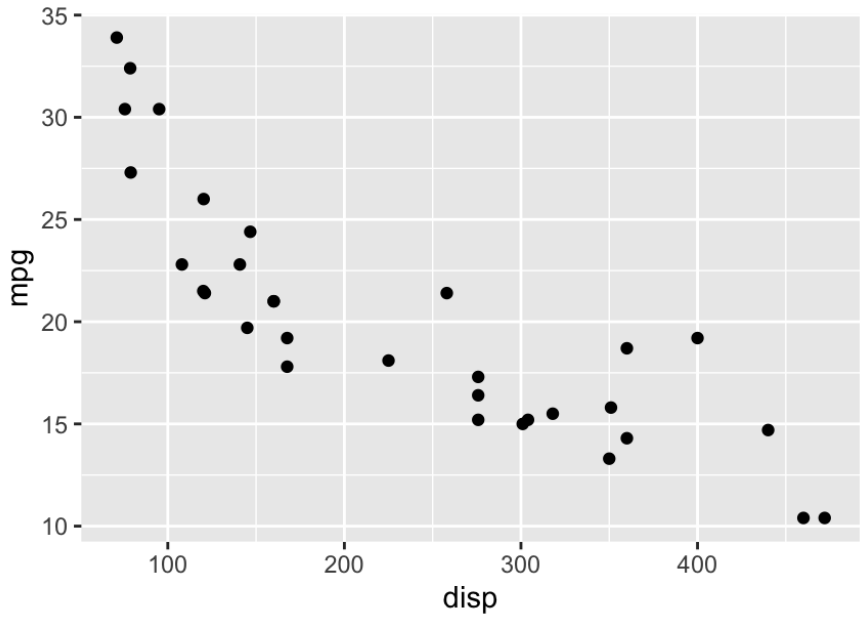
For now, we keep it simple:

- geom_line()
- geom_point()

ggplot function: geometrical layer

geom_point() -> scatter plot

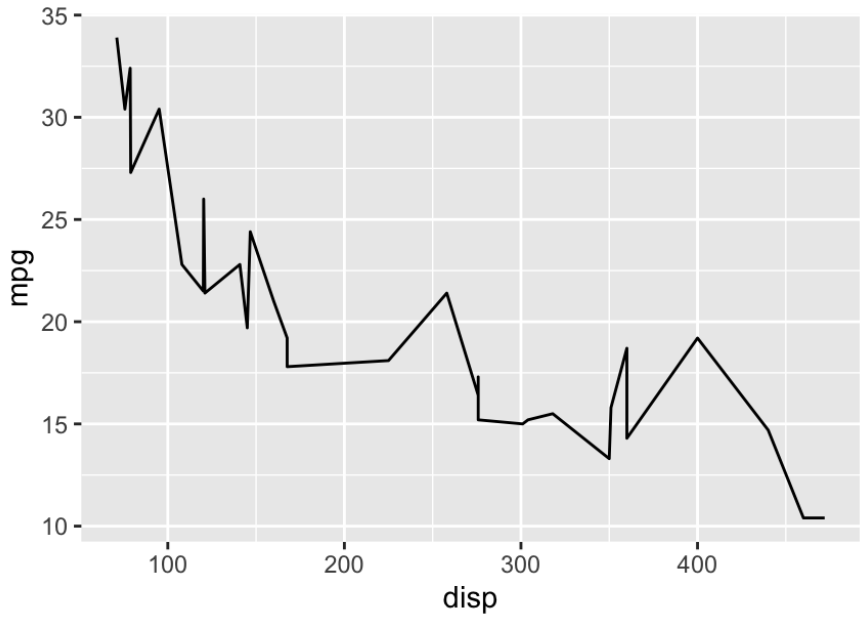
```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point()
```



ggplot function: geometrical layer

geom_line() -> line plot

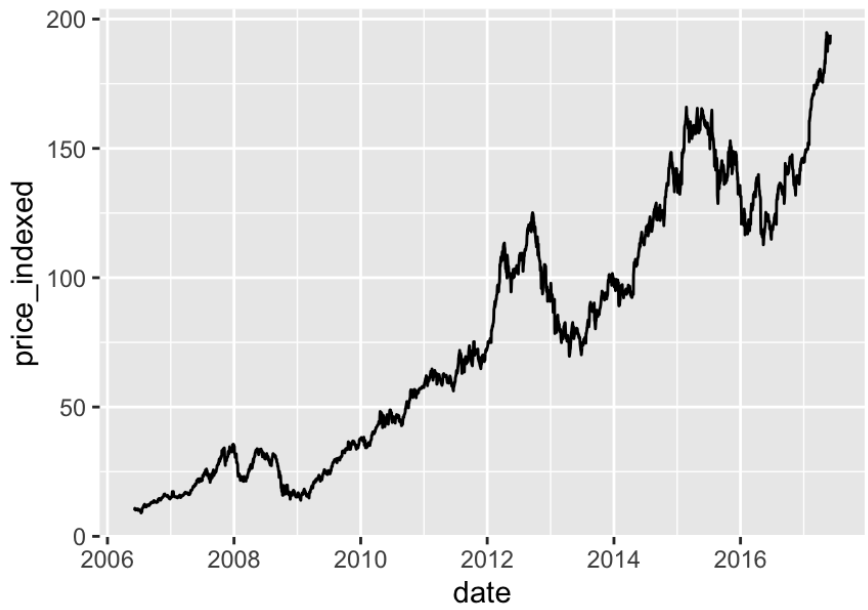
```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_line()
```



ggplot function: geometrical layer

geom_line() -> line plot

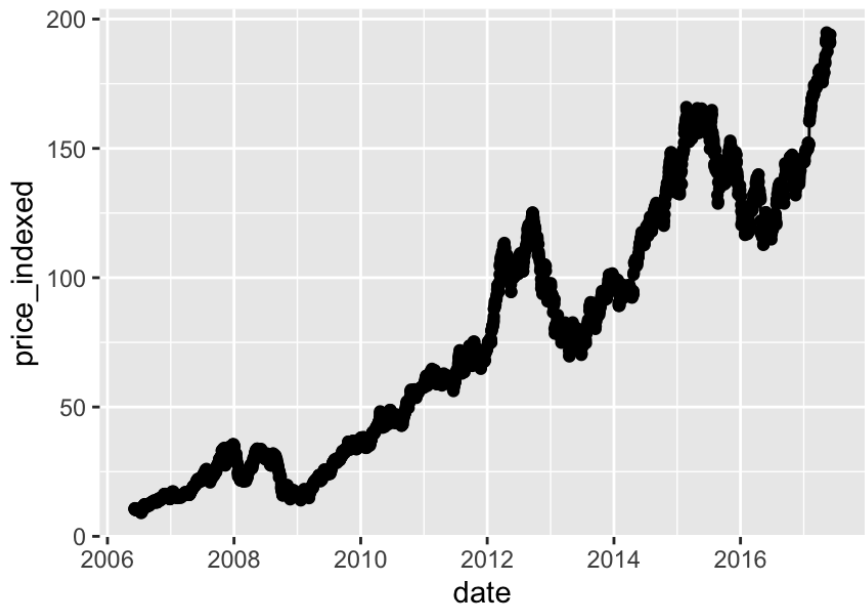
```
stocks_apple <- stocks[
  which(stocks$company == "Apple"),
  ]

ggplot(data = stocks_apple,
       mapping = aes(x = date,
                     y = price_indexed)) +
 geom_line()
```



ggplot function: geometrical layers

Add all individual data points:

```
stocks_apple <- stocks[
  which(stocks$company == "Apple"),
  ]

ggplot(data = stocks_apple,
       mapping = aes(x = date,
                     y = price_indexed)) +
 geom_line() +
 geom_point()
```



Saving plots

Saving plots - to a R object

Saving a plot to a R object

You can save the plot as a R object:

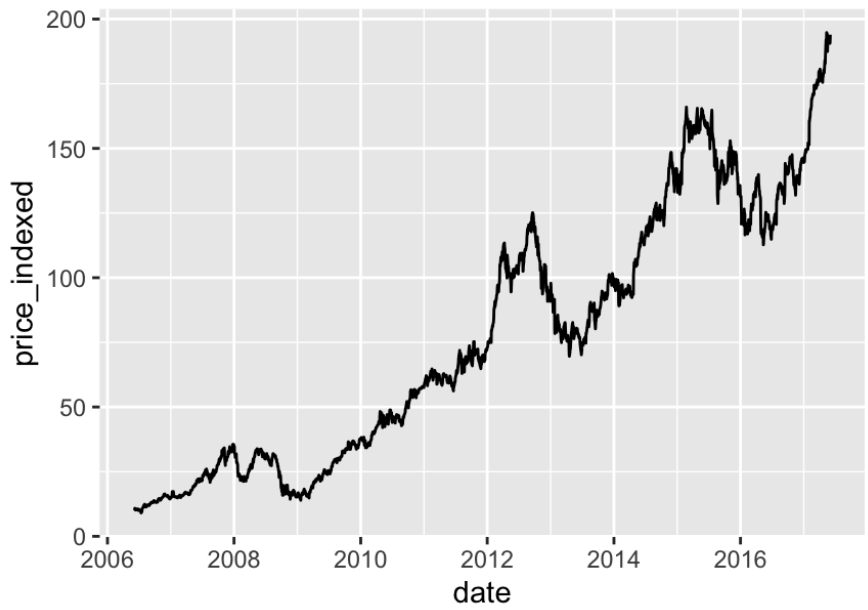
```
stocklineplot <- ggplot(
                    data = stocks_apple,
                    mapping =
                      aes(x = date,
                          y =
                            price_indexed)) +
  geom_line()
```

This (usually) does not give output!

Saving a plot to a R object

To plot a saved plot, ask for the plot:

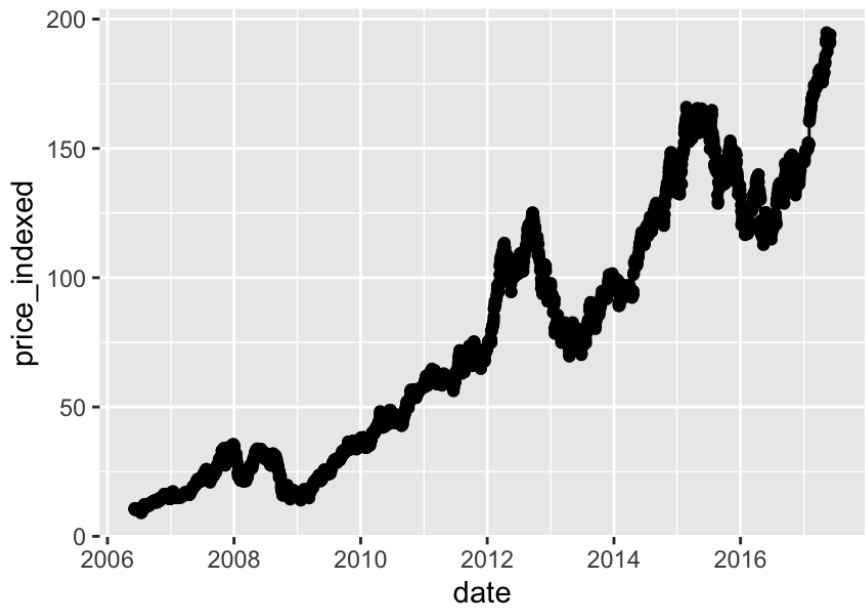
```
stocklineplot <- ggplot(
                    data = stocks_apple,
                    mapping =
                      aes(x = date,
                          y =
                            rice_indexed)) +
  geom_line()

# to plot the object:
stocklineplot
```



Saving a plot to a R object

You can add additional layers later:

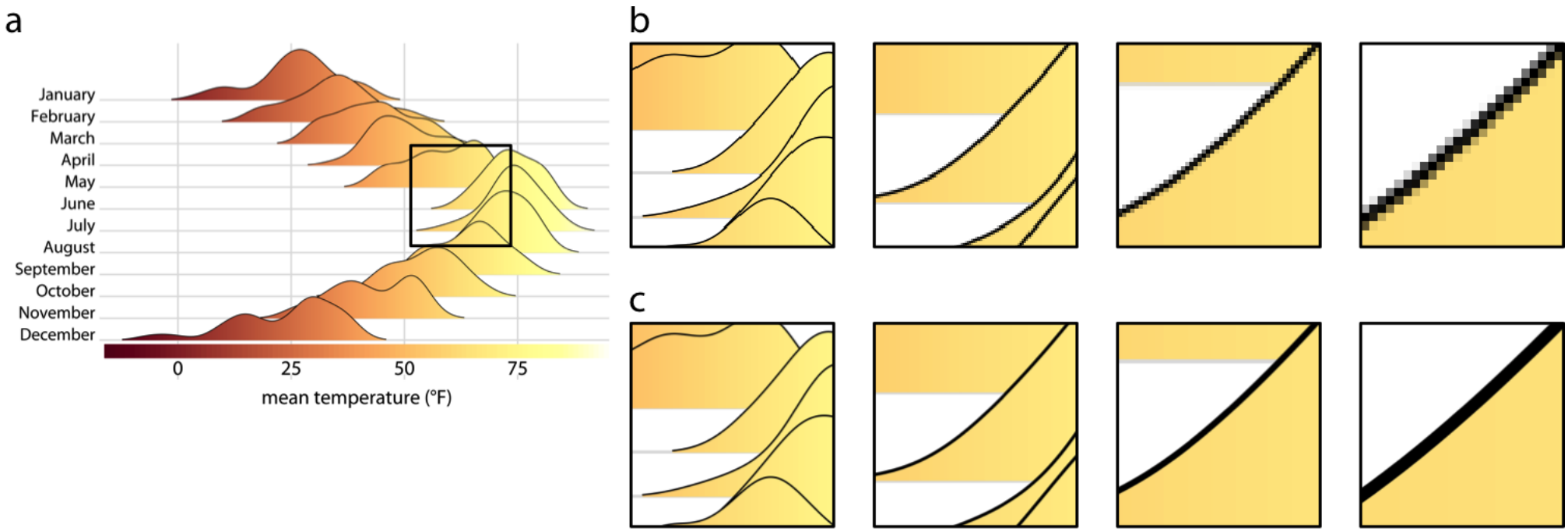
```
stocklineplot + geom_point()
```



Saving plots - to a file

Saving a plot to a file

Different types of file formats: bitmap vs vector:



Saving a plot to a file

Downsides of vector format:

- are redrawn by the graphics program with which they are displayed
  - -> differences in how the same graphic looks in two different programs/computers.
  - often a problem with fonts
- can grow to enormous file sizes
  - -> slow to render

Saving a plot to a file

Many different image file formats:

Table 27.1: Commonly used image file formats

Acronym	Name
pdf	Portable Document Format
eps	Encapsulated PostScript
svg	Scalable Vector Graphics
png	Portable Network Graphics
jpeg	Joint Photographic Experts Group
tiff	Tagged Image File Format
raw	Raw Image File
gif	Graphics Interchange Format

Saving a plot to a file

ggsave()

- defaults to saving
  - last plot that you displayed
  - using the size of the current graphics device (or guessed from extension)

```
ggsave
```

```
## function (filename, plot = last_plot(), device = NULL, path = NULL,
##     scale = 1, width = NA, height = NA, units = c("in", "cm",
##         "mm", "px"), dpi = 300, limitsize = TRUE, bg = NULL,
##     ...)
## {
##     if (length(filename) != 1) {
##         if (length(filename) == 0) {
##             cli::cli_abort("{.arg filename} cannot be empty.")
##         }
##         len <- length(filename)
##         filename <- filename[1]
##         cli::cli_warn(c("{.arg filename} must have length 1, not length {len}.",
```

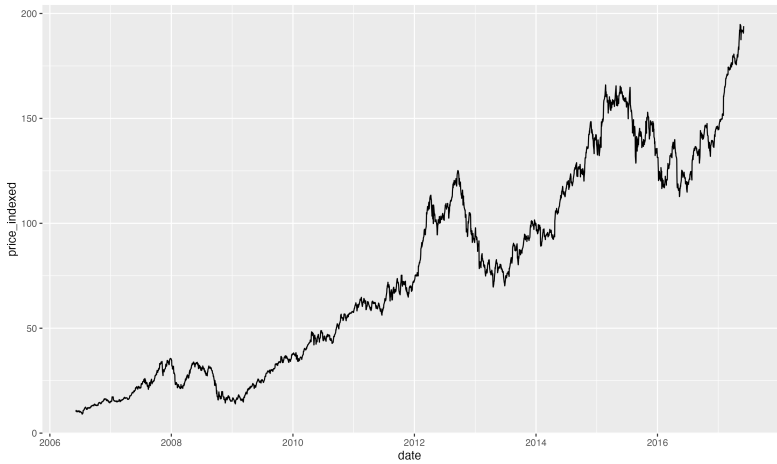
Saving a plot to a file

```
ggsave(filename = "Lineplot_stocks.pdf",
       plot = stocklineplot
)

ggsave(filename = "Lineplot_stocks.png",
       plot = stocklineplot
)
```

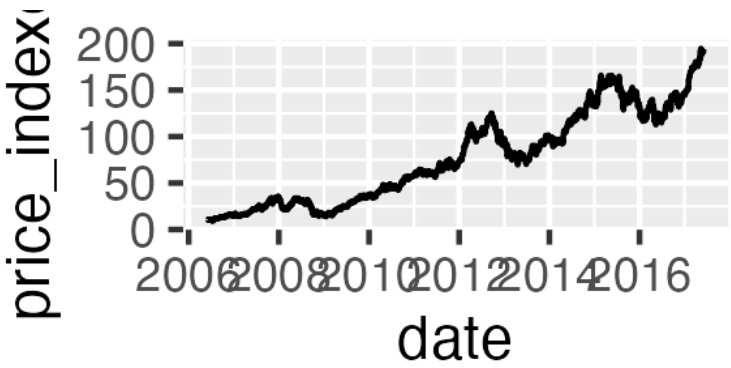
Saving a plot to a file

Choose sensible dimensions:

```
ggsave(filename =
         "Lineplot_stocks_large.png",
       plot = stocklineplot,
       width = 10,
       height = 6

)
```



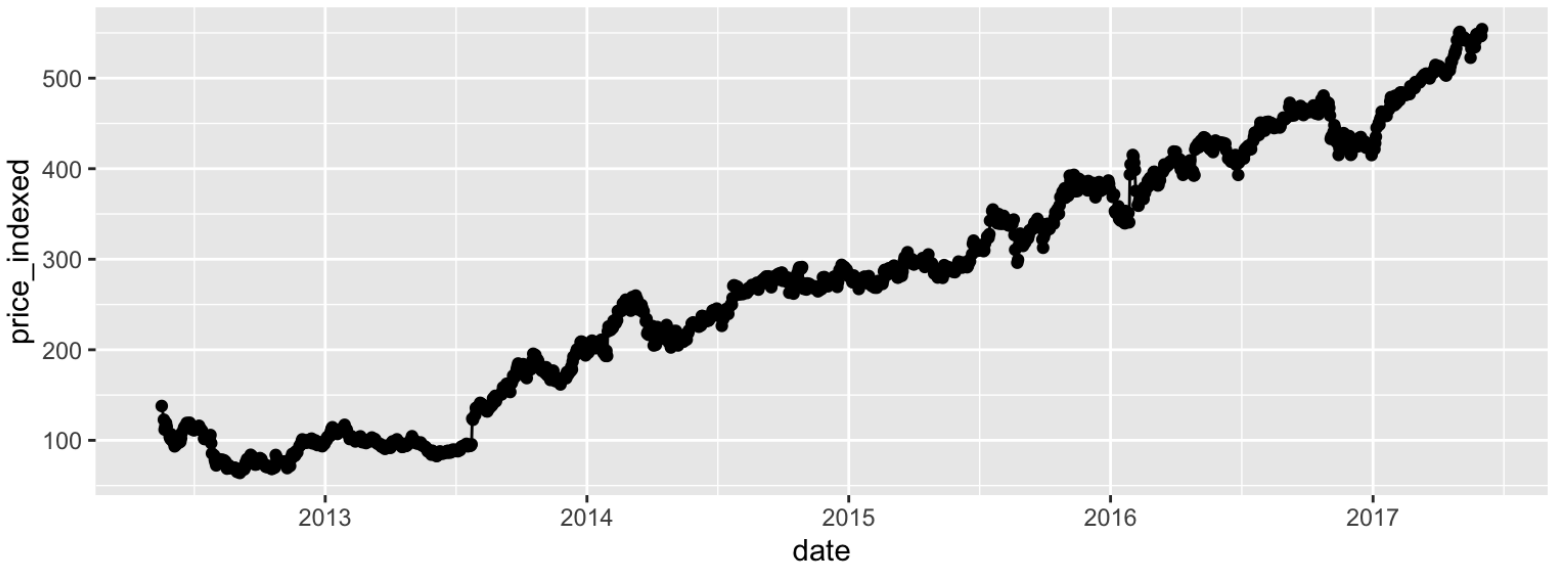

```
ggsave(filename =
         "Lineplot_stocks_small.png",
       plot = stocklineplot,
        width = 2,
       height = 1
)
```



Excercise

Excercise: Combine graph types and study file format types

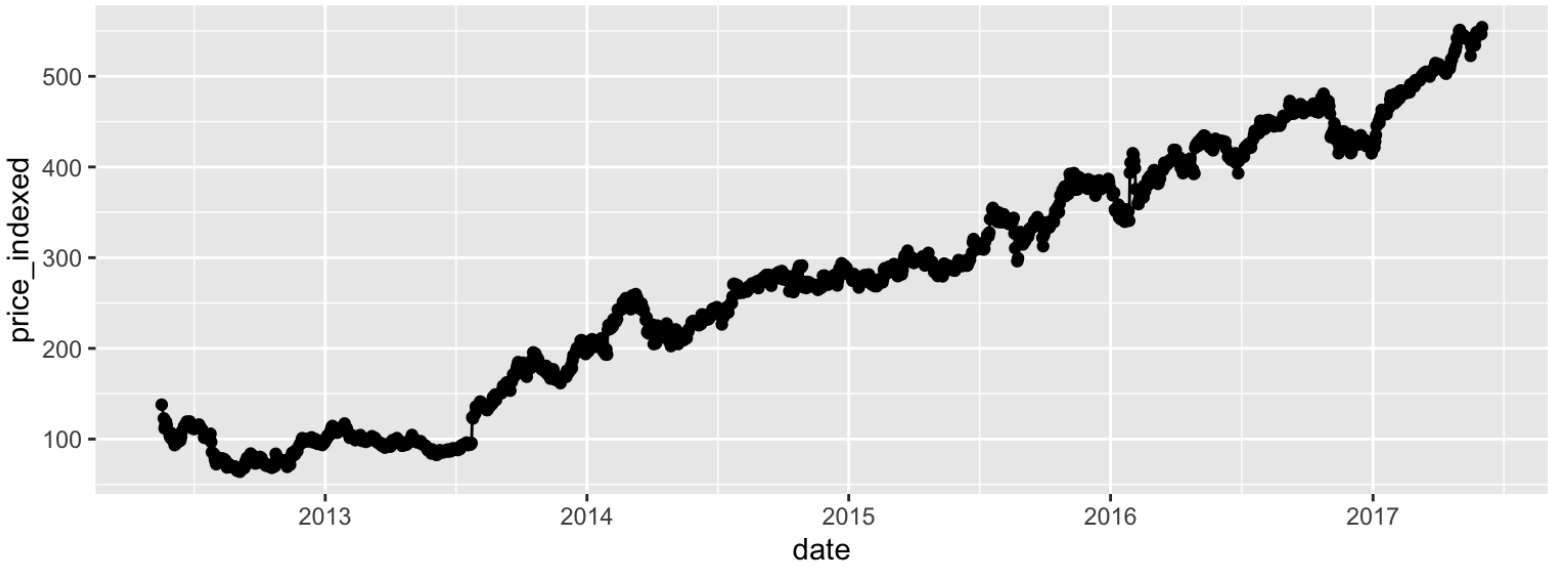
1. Make a line graph of the indexed stocks price of Facebook and save it as object.
2. Add points on the line for each observation.
3. Save this plot with reasonable dimensions using the following formats: pdf, eps, svg, png, jpeg, tiff. Which are vector types?




```
# Save to all formats:
```

Answer: plot

```
stocks_Facebook <-  stocks[which(stocks$company == "Facebook"),]
lineplotFacebook <- ggplot(data = stocks_Facebook,
                           mapping = aes(x = date, y = price_indexed)) +
  geom_line()

lineAndPointFacebook <- lineplotFacebook + geom_point()
lineAndPointFacebook
```



Answer: saving

```
# pdf:
ggsave(filename =
         "Figures/Exercises/lineAndPointFacebook.pdf",
       plot = lineAndPointFacebook,
        width = 8,
       height = 3
)

#eps
ggsave(filename =
         "Figures/Exercises/lineAndPointFacebook.eps",
       plot = lineAndPointFacebook,
        width = 8,
       height = 3
)

# etc..
```

Answer: file format types

Many different image file formats:

Table 27.1: Commonly used image file formats

Acronym	Name	Type	Application
pdf	Portable Document Format	vector	general purpose
eps	Encapsulated PostScript	vector	general purpose, outdated; use pdf
svg	Scalable Vector Graphics	vector	online use
png	Portable Network Graphics	bitmap	optimized for line drawings
jpeg	Joint Photographic Experts Group	bitmap	optimized for photographic images
tiff	Tagged Image File Format	bitmap	print production, accurate color reproduction
raw	Raw Image File	bitmap	digital photography, needs post-processing
gif	Graphics Interchange Format	bitmap	outdated for static figures, Ok for animations

More aesthetics

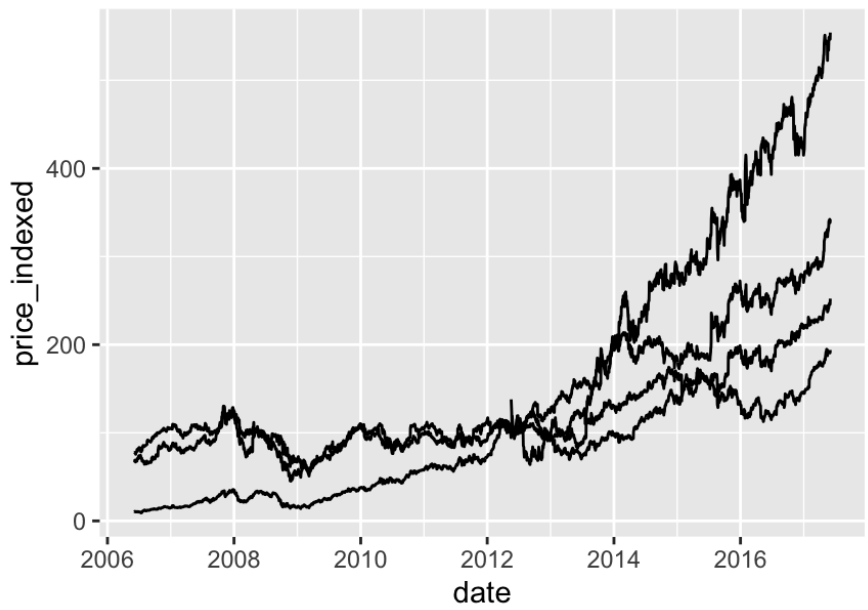
There are many aesthetics that you can use in your plot:

- x position
- y position
- groupings
- colors
- fills
- shapes
- etc.

More aesthetics: groups - split data

Different line for each group

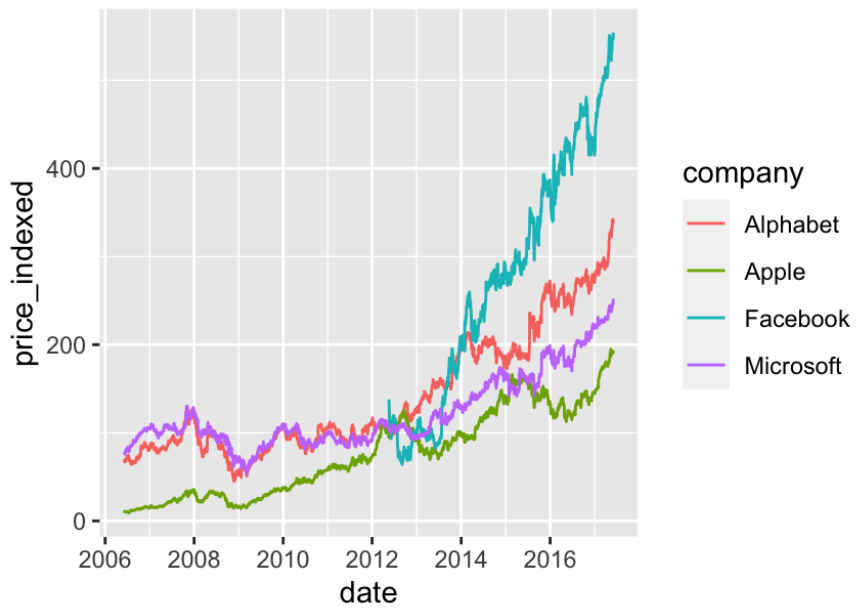
```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     group = company)) +
 geom_line()
```



More aesthetics: groups - split data + give color

Different color for each group

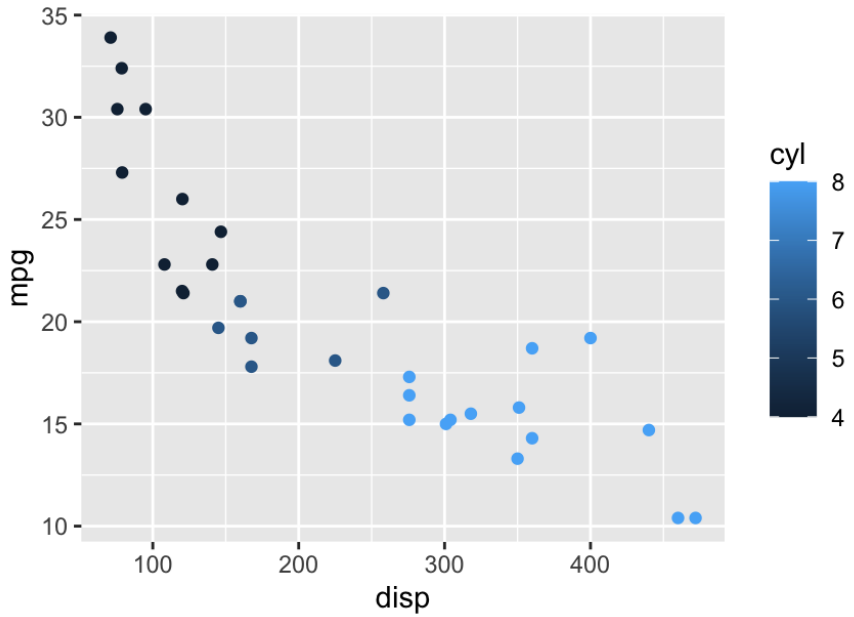
```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     color = company)) +
 geom_line()
```



More aesthetics: groups - split data + give color

Different color for each group

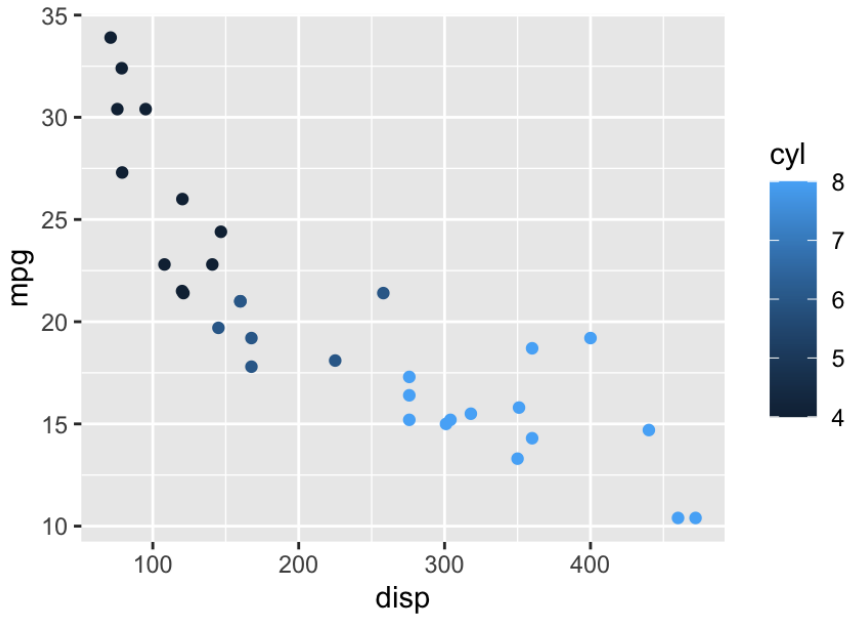
```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color = cyl)) +
 geom_point()
```



More aesthetics: groups - split data + give color

Different color for each group

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color = cyl)) +
 geom_point()
```



But: the number of cylinders is an integer, so this scale does not make sense!

More aesthetics: groups - split data + give color

“cyl” is not defined as a categorical variable, but as a numeric values:

```
class(mtcars$cyl)
```

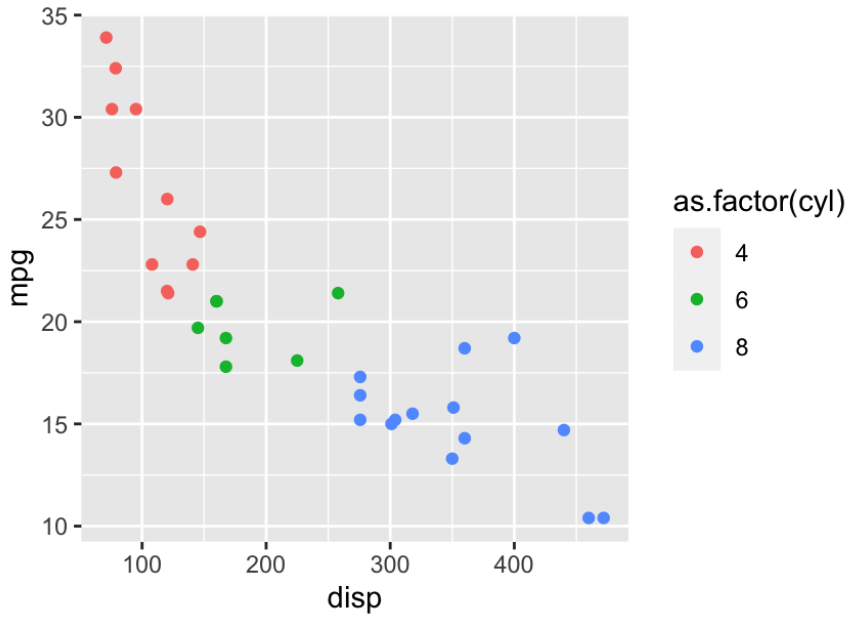
```
## [1] "numeric"
```

Without changing the original data, you can still plot it as a factor -> …

More aesthetics: groups - split data + give color

Different color for each group -> if needed, change data type

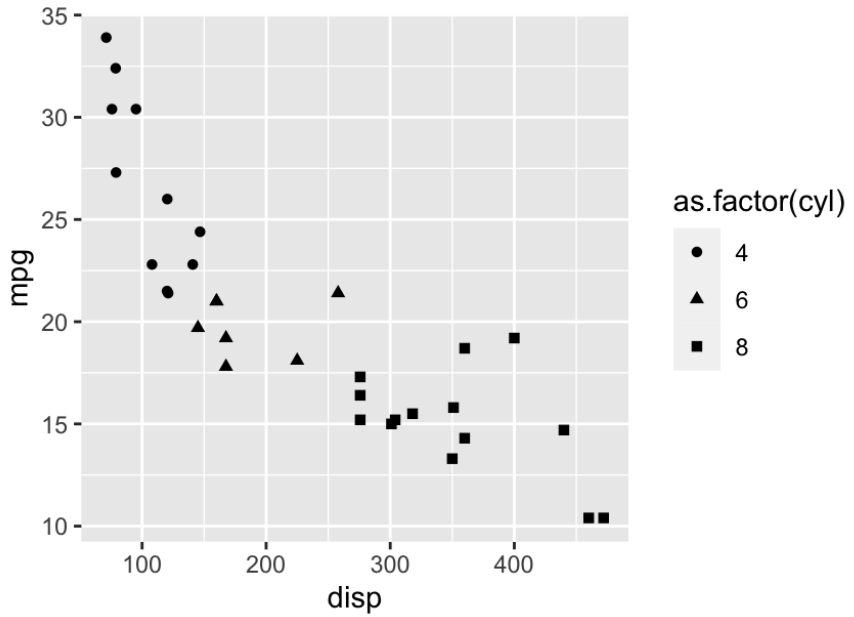
```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl))) +
 geom_point()
```



More aesthetics: groups - split data + vary shape

Different shape for each group

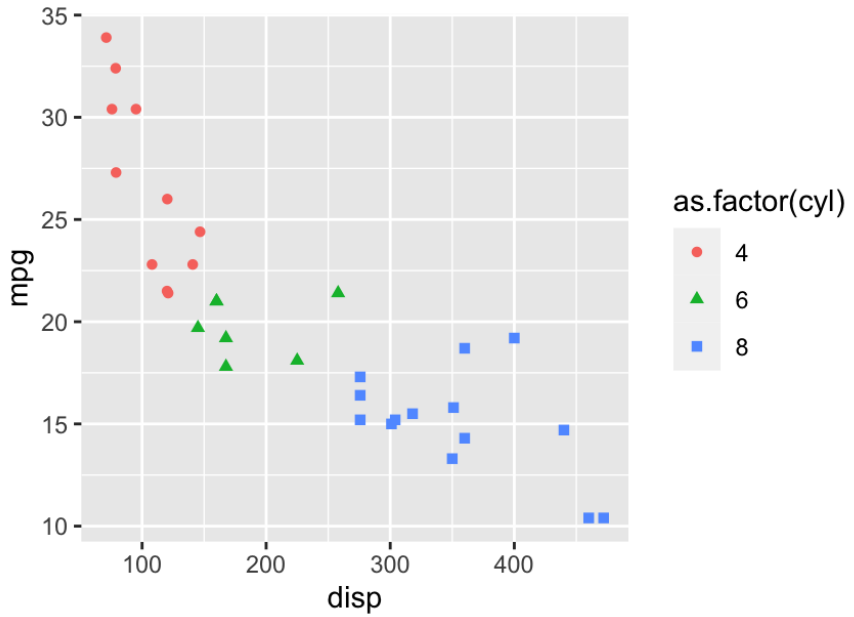
```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     shape =
                       as.factor(cyl))) +
 geom_point()
```



More aesthetics: groups - redundant coding

Different color *and* shape for each group

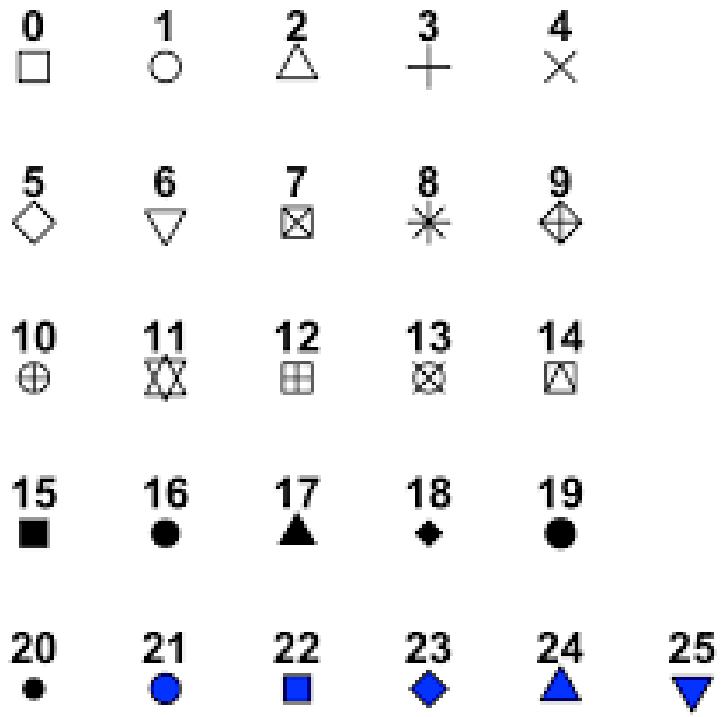
```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl),
                     shape =
                       as.factor(cyl))) +
 geom_point()
```



More aesthetics: color vs. fill

color: coloring of lines and edges

fill: coloring of shaded areas (bars, filling op shapes, etc.)



Scales (adjusting mapping)

Scales

You can adjust the scales, i.e. the mapping between values and the aesthetics, using the scale functions of the form

scale_*()

There are many different functions, many for each type of aesthetic.

Adjusting scales: cheat sheet

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - Map cont' values to visual ones.
scale_*_discrete() - Map discrete values to visual ones.
scale_*_binned() - Map continuous values to discrete bins.
scale_*_identity() - Use data values as visual ones.
scale_*_manual(values = c()**)** - Map discrete values to manually chosen visual ones.
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks"**)** - Treat data values as dates.
scale_*_datetime() - Treat data values as date times. Same as scale_*_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale.
scale_x_reverse() - Reverse the direction of the x axis.
scale_x_sqrt() - Plot x on square root scale.

COLOR AND FILL SCALES (DISCRETE)

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red"**)**

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues"**)**

o + scale_fill_gradient(low="red", high="yellow"**)**

o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25**)**

o + scale_fill_gradientn(colors = topo.colors(6)**)**
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

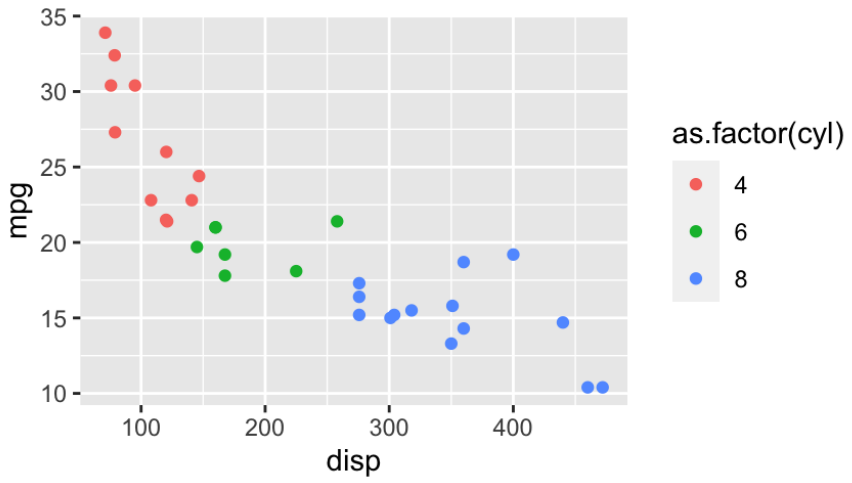
p <- e + geom_point(aes(shape = fl, size = cyl))

p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7)**)**

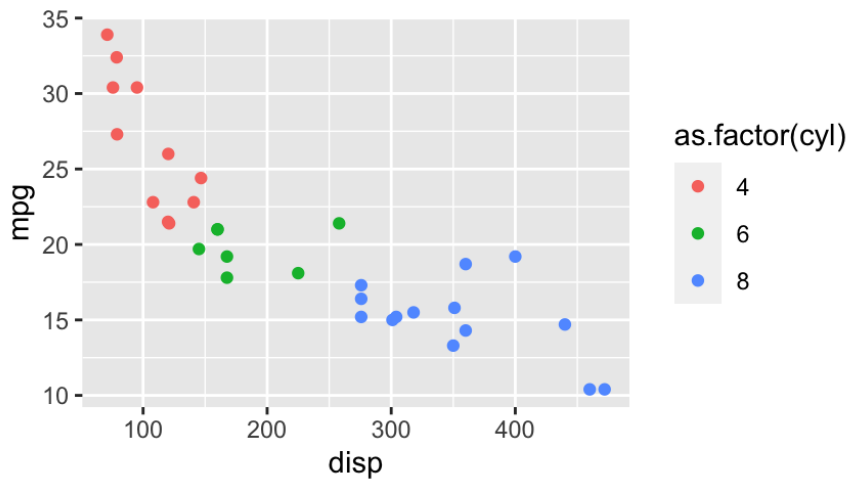
p + scale_radius(range = c(1,6)**)**
p + scale_size_area(max_size = 6**)**

ggplot uses these in the background

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl)))
 geom_point()
```

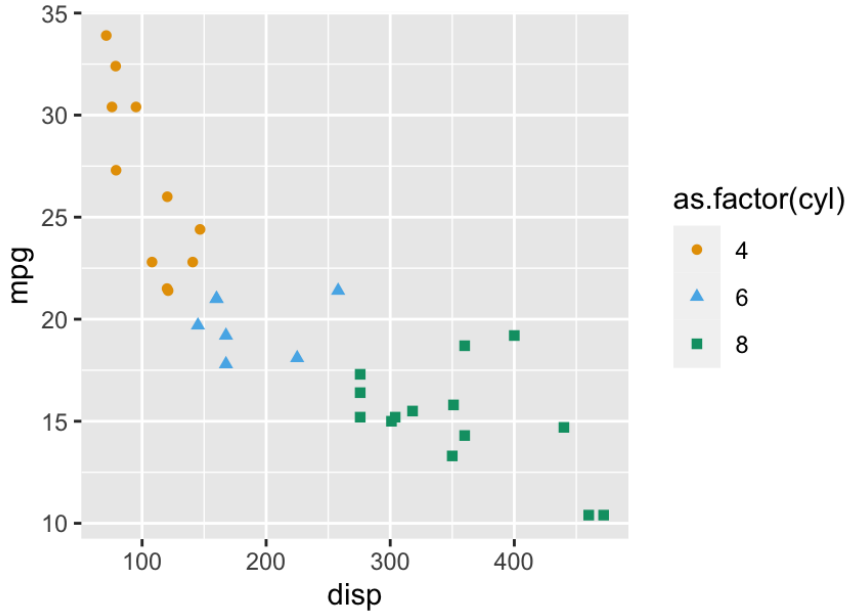



```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl))) +
 geom_point() +
  scale_x_continuous() +
  scale_y_continuous() +
  scale_color_discrete()
```



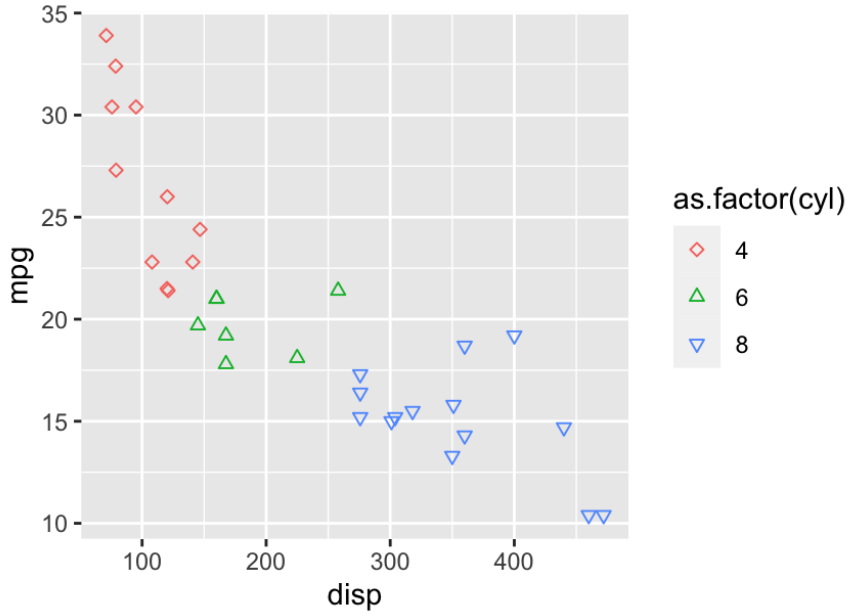
Example: manually specify colors

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl),
                     shape =
                       as.factor(cyl))) +
 geom_point() +
  scale_color_manual(values = c("#E69F00",
                                "#56B4E9",
                                "#009E73"))
```



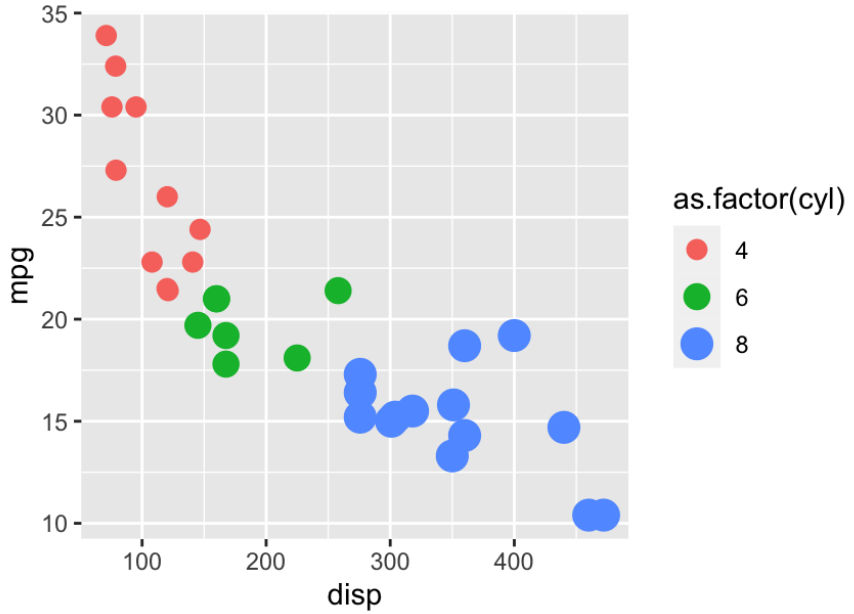
Example: manually specify shapes

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl),
                     shape =
                       as.factor(cyl))) +
 geom_point() +
  scale_shape_manual(values = c(23,24,25))
```



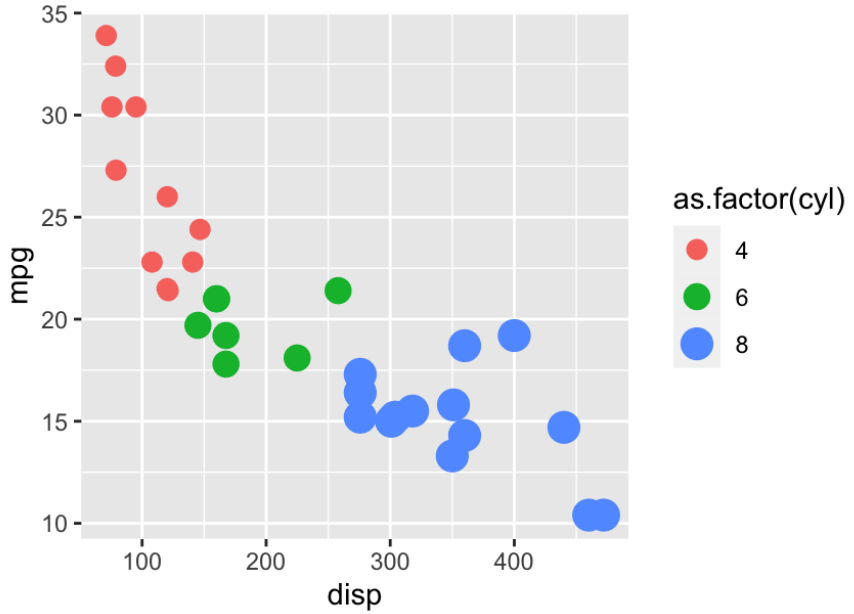
Example: manually specify size

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl),
                     size =
                       as.factor(cyl))) +
 geom_point() +
  scale_size_manual(values = c(3,4,5))
```



Example: manually specify size

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     color =
                       as.factor(cyl),
                     size =
                       as.factor(cyl))) +
 geom_point() +
  scale_size_manual(values = c(3,4,5))
```

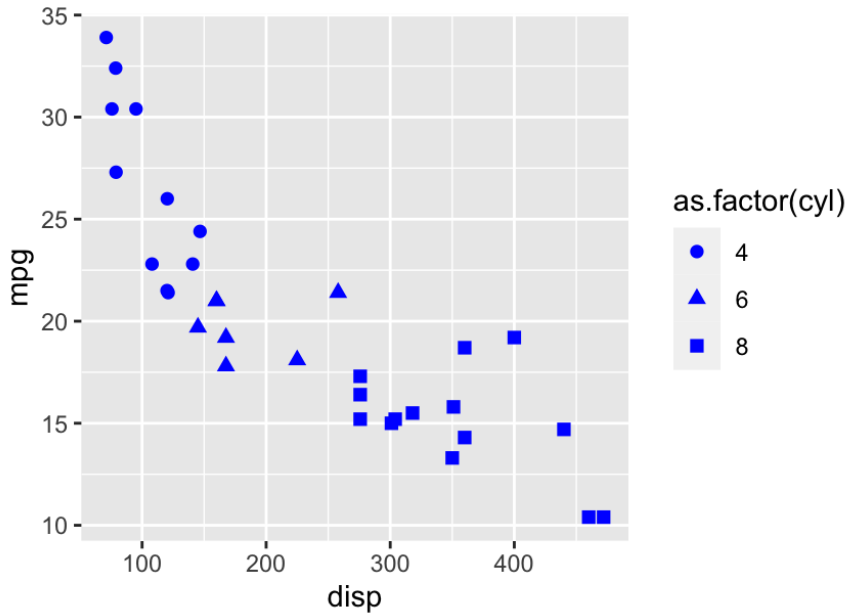


But note that sizes are associated with values, so only use different sizes for variables with an ordering!

Adjusting aesthetics independent of the data

If aesthetics are independent of the data, then define them outside the aes() function in the mapping:

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg,
                     shape =
                       as.factor(cyl))) +
 geom_point(color = "blue",
            size = 2)
```



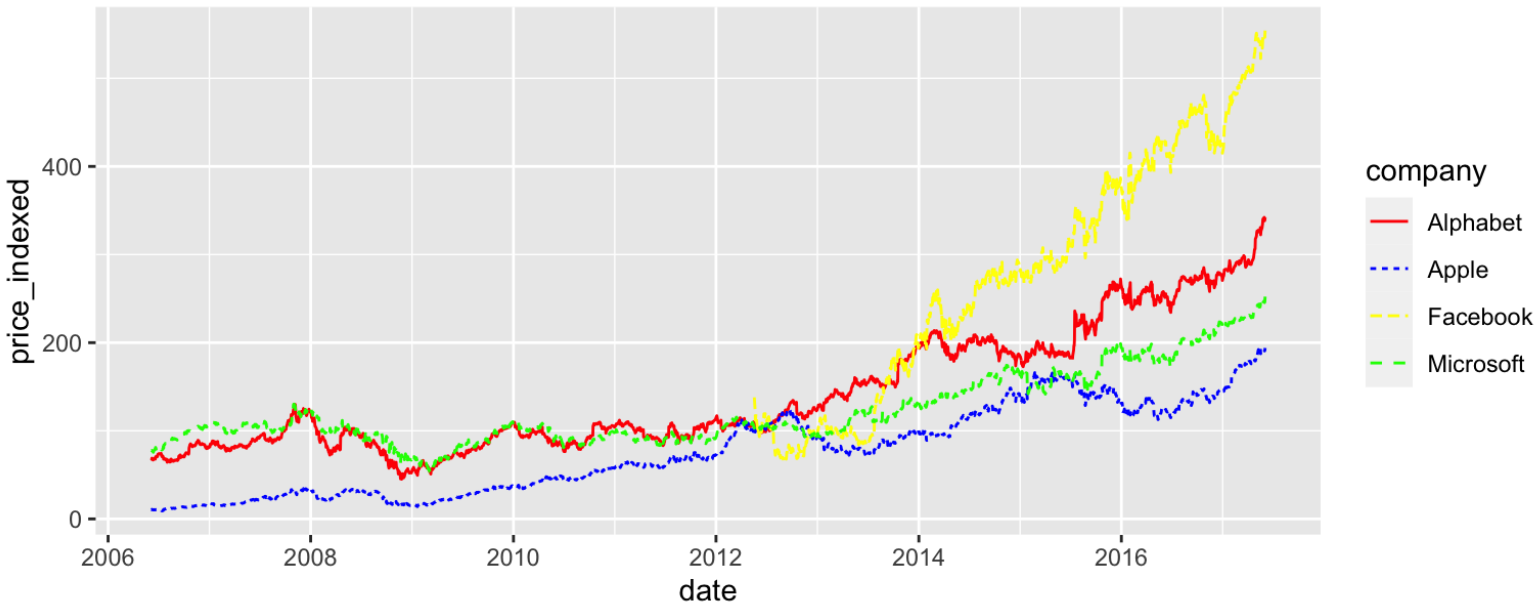
Exercise

Exercise: adjust scales for redundant coding

Make a line graph about the stocks data with redundant coding.

For each company:

- different line type
- different color (red, blue, yellow, green)

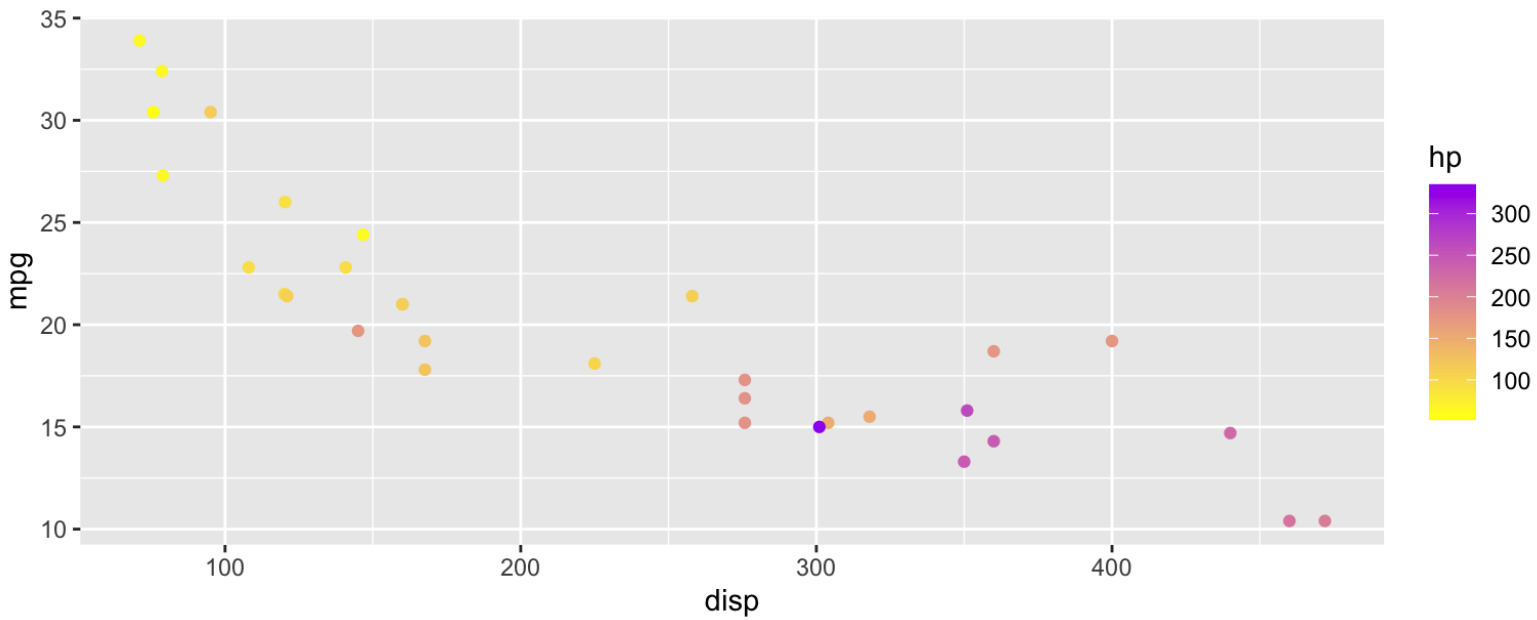


Exercise: gradient coloring

Make a scatter plot of the mtcars data

For each observation:

- indicate the power (hp) with a color gradient:
- yellow for low values, purple for high values.

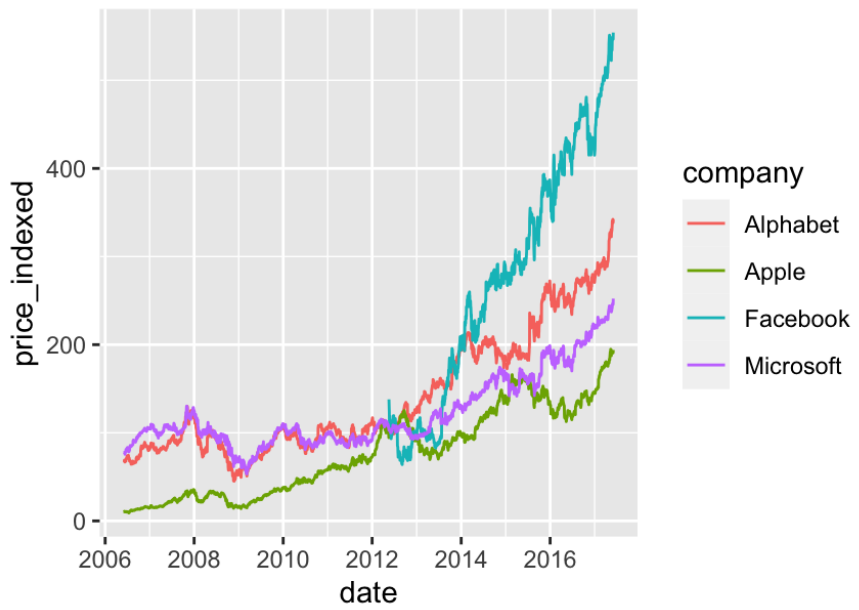


Axis labels and titles

Adding axis labels

Standard label = column name

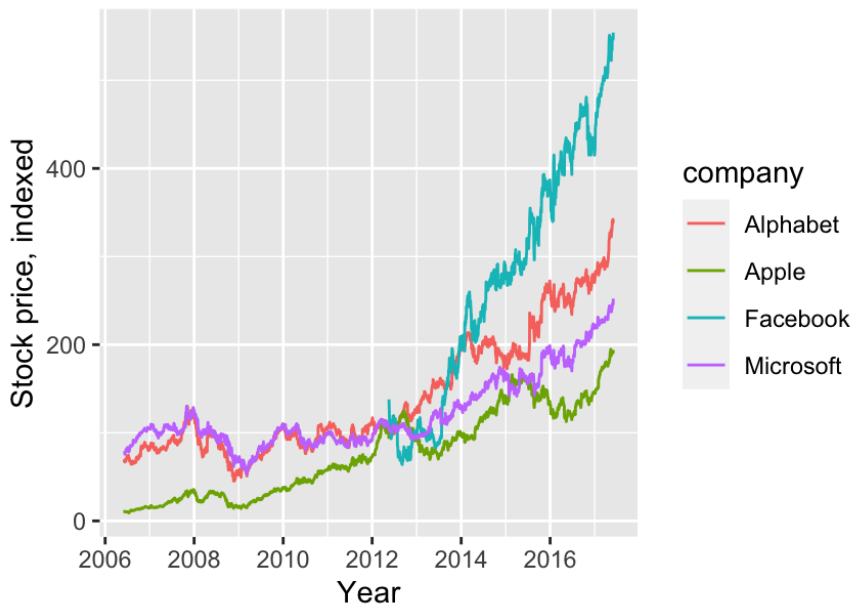
```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     color = company)) +
 geom_line()
```



Adding axis labels

You can choose your own labels:

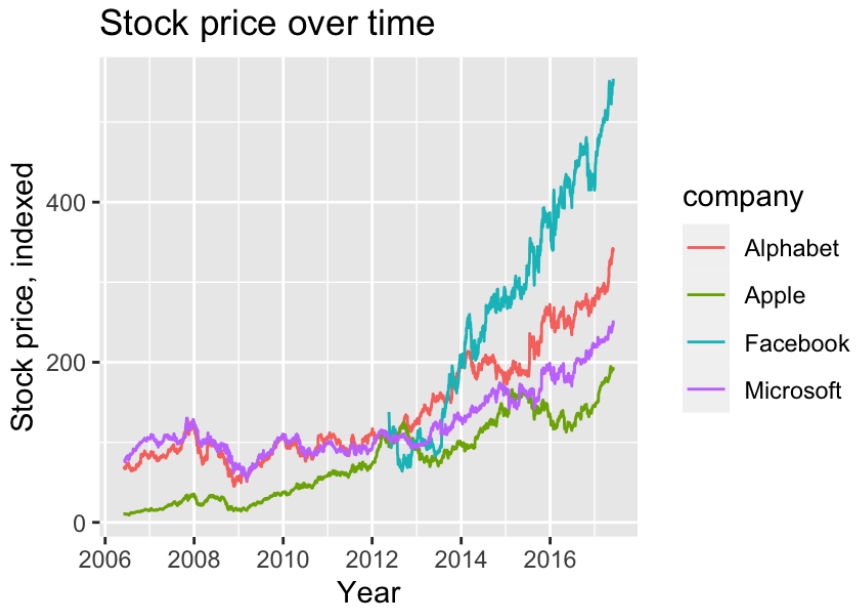
```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     color = company)) +
 geom_line() +
  xlab("Year") +
  ylab("Stock price, indexed")
```



Adding title

You can also add a title:

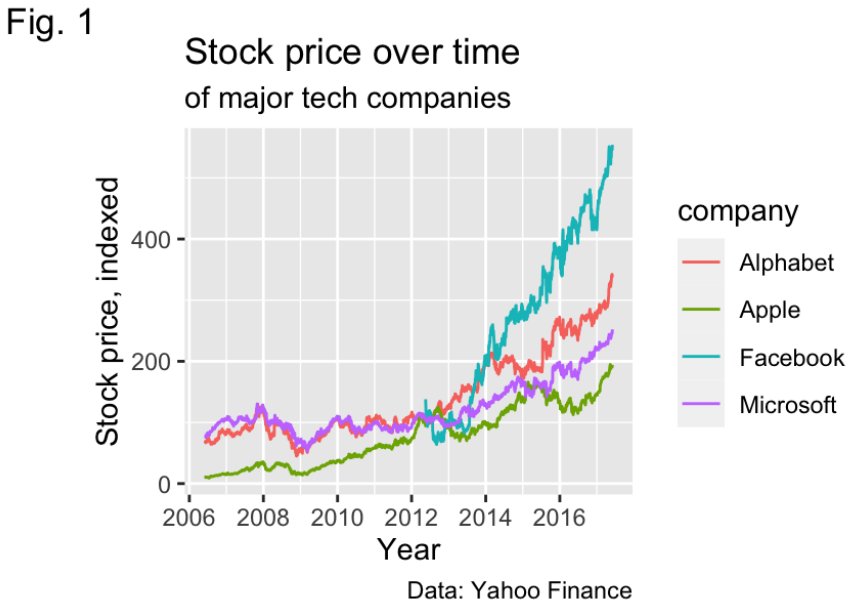
```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     color = company)) +
 geom_line() +
  xlab("Year") +
  ylab("Stock price, indexed") +
  ggtitle("Stock price over time")
```



Adding axis labels, title, etc.

You can also labs(); more options, all together

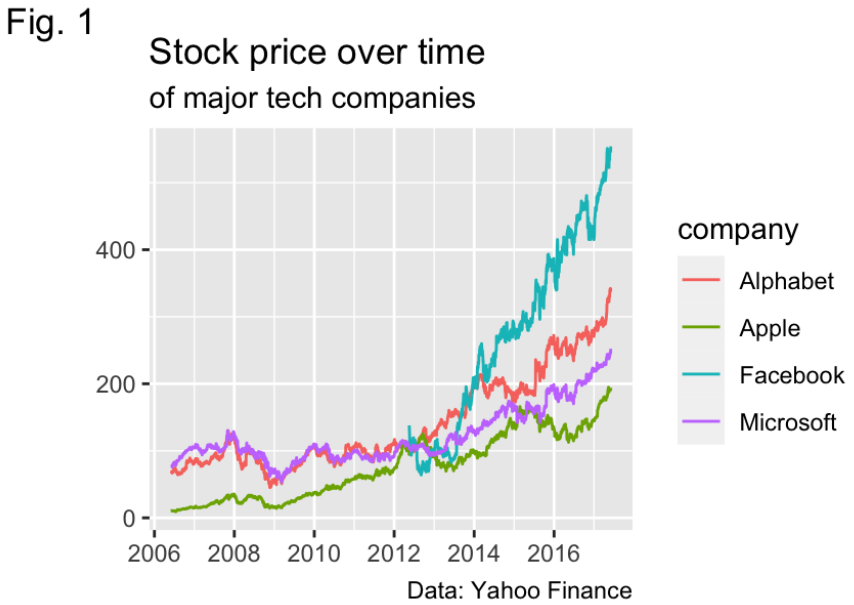
```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     color = company)) +
 geom_line() +
 labs(
   x = "Year",
   y = "Stock price, indexed",
   title = "Stock price over time",
   subtitle = "of major tech companies",
   caption = "Data: Yahoo Finance",
   tag = "Fig. 1"
 )
```



Adding axis labels, title, etc.

No axis label? Don't us empty strings as they require space; use "NULL":

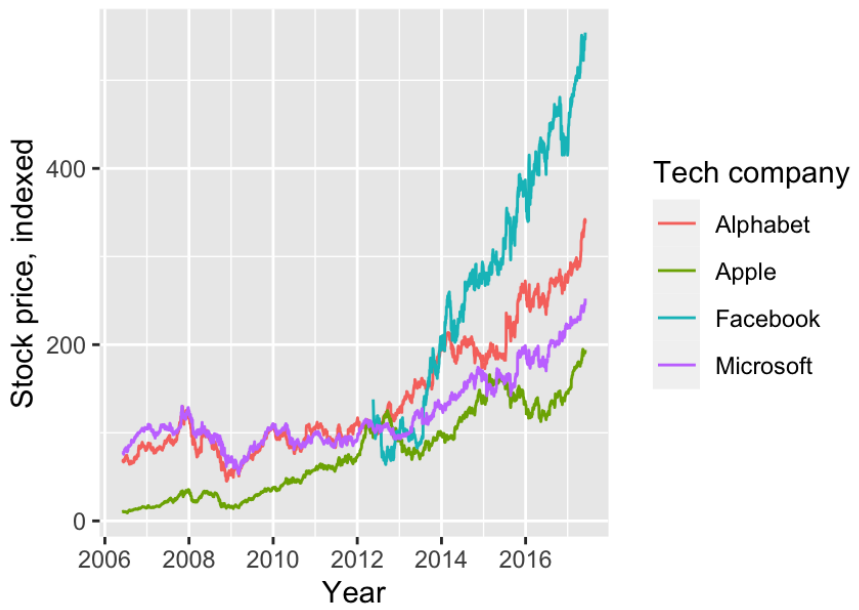
```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     color = company)) +
 geom_line() +
 labs(
   x = NULL,
   y = NULL,
   title = "Stock price over time",
   subtitle = "of major tech companies",
   caption = "Data: Yahoo Finance",
   tag = "Fig. 1"
 )
```



Adding axis labels, title, etc.

You can also use the labs() function to change the labels of other aesthetics:

```
ggplot(data = stocks,
       mapping = aes(x = date,
                     y = price_indexed,
                     color = company)) +
 geom_line() +
 labs(
   x = "Year",
   y = "Stock price, indexed",
   color = "Tech company"
 )
```

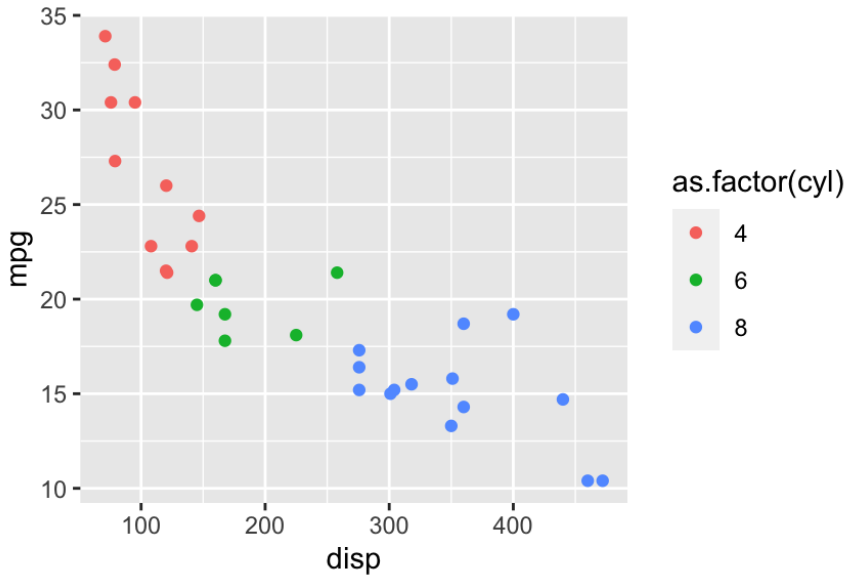


Legends

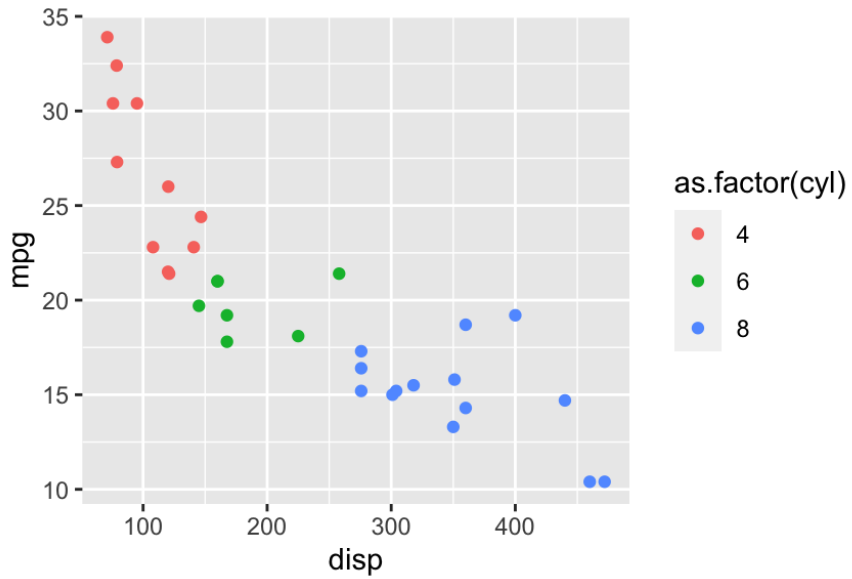
Legend = guide

Guide functions are used in the background:

```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color =
                       as.factor(cyl)))
 geom_point()
```



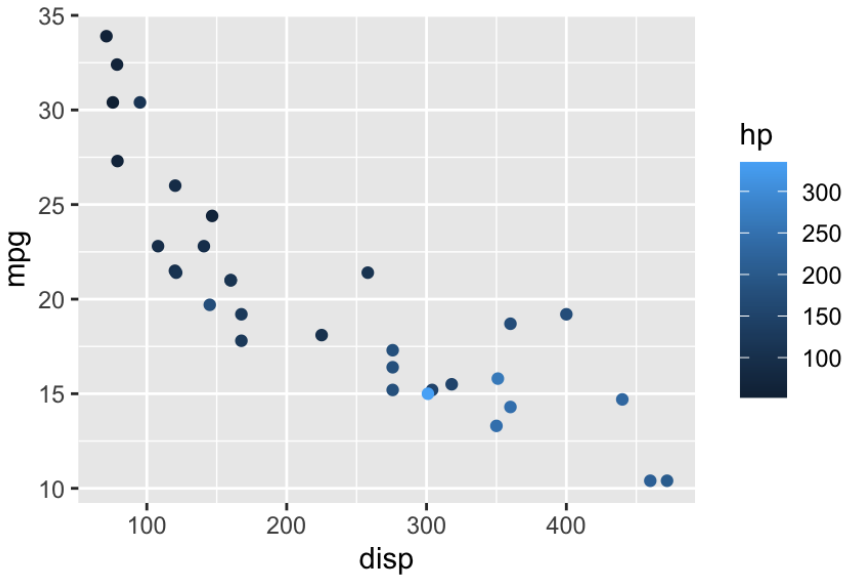

```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color =
                       as.factor(cyl))) +
 geom_point() +
 guides(color = guide_legend()) # discrete
```



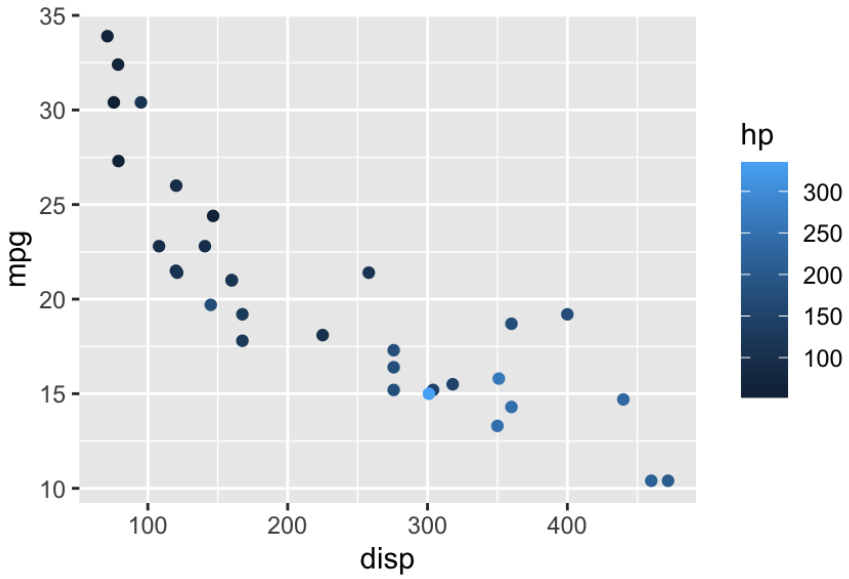
Legend = guide

Guide functions are used in the background:

```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color = hp)) +
 geom_point() +
 guides(color = guide_colorbar()) # cont
```



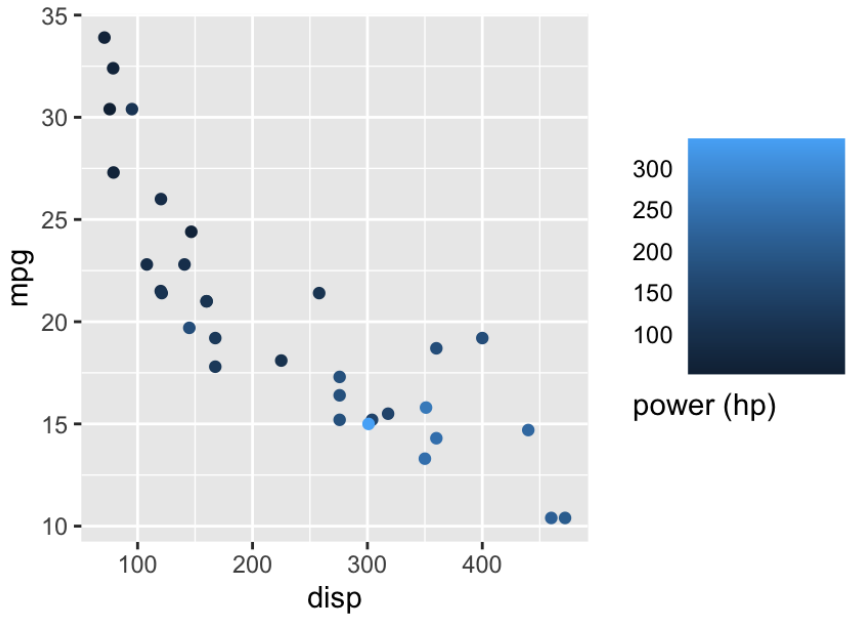

```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color = hp)) +
 geom_point() +
 guides(color = guide_colorbar()) # continuous
```



Change guides

You can use the functions to make changes:

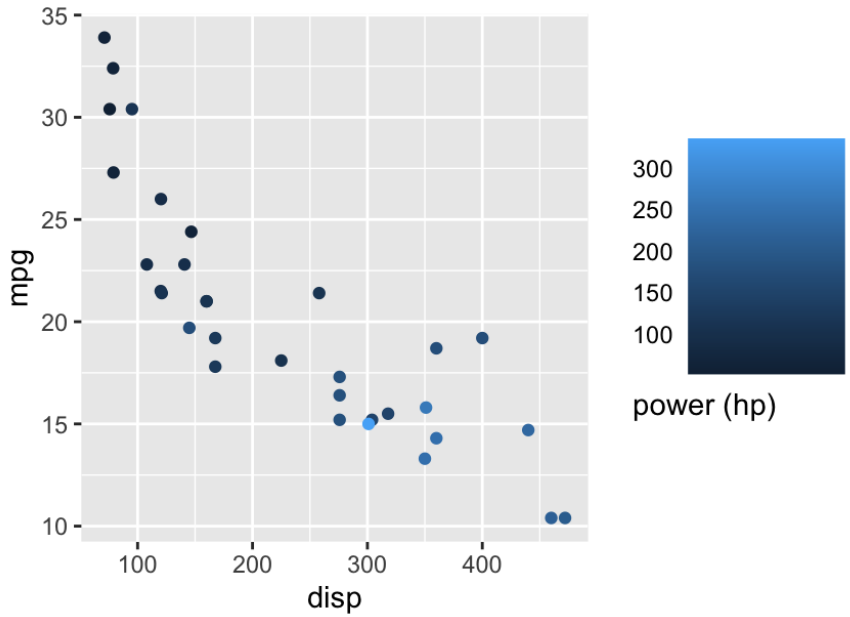
```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color = hp)) +
 geom_point() +
 guides(color = guide_colorbar(
   title = "power (hp)",
   barwidth = 4,
   ticks = FALSE,
   label.position = "left",
   title.position = "bottom"))
```



Change guide:

You can use the functions to make changes:

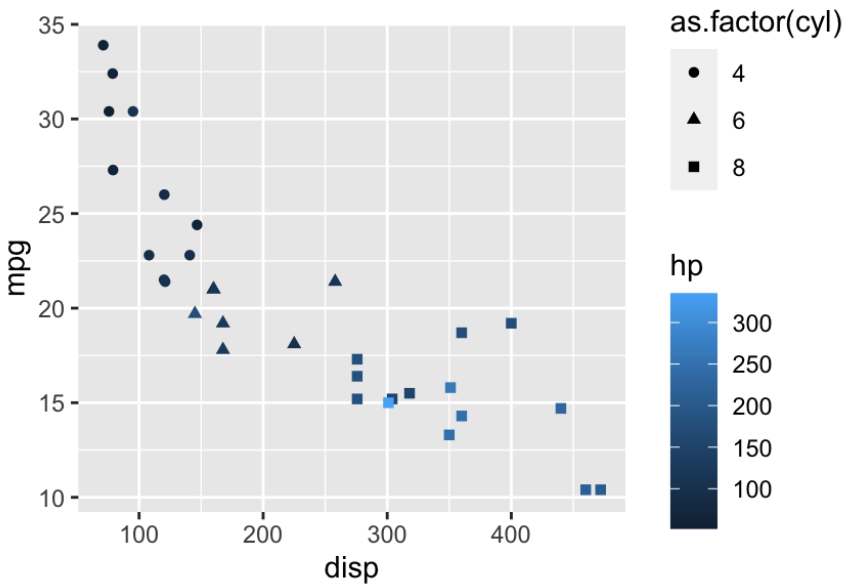
```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color = hp)) +
 geom_point() +
 guides(color = guide_colorbar(
   title = "power (hp)",
   barwidth = 4,
   ticks = FALSE,
   label.position = "left",
   title.position = "bottom"))
```



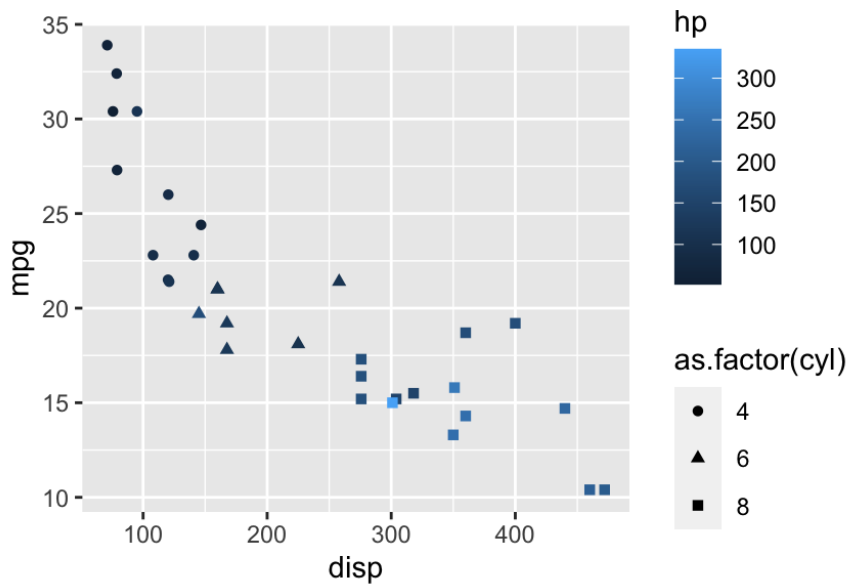
The many options are specific for each type of guide.

Multiple guides

```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color = hp,
                     shape =
                       as.factor(cyl)))
 geom_point() +
 guides(color = guide_colorbar(),
        shape = guide_legend())
```



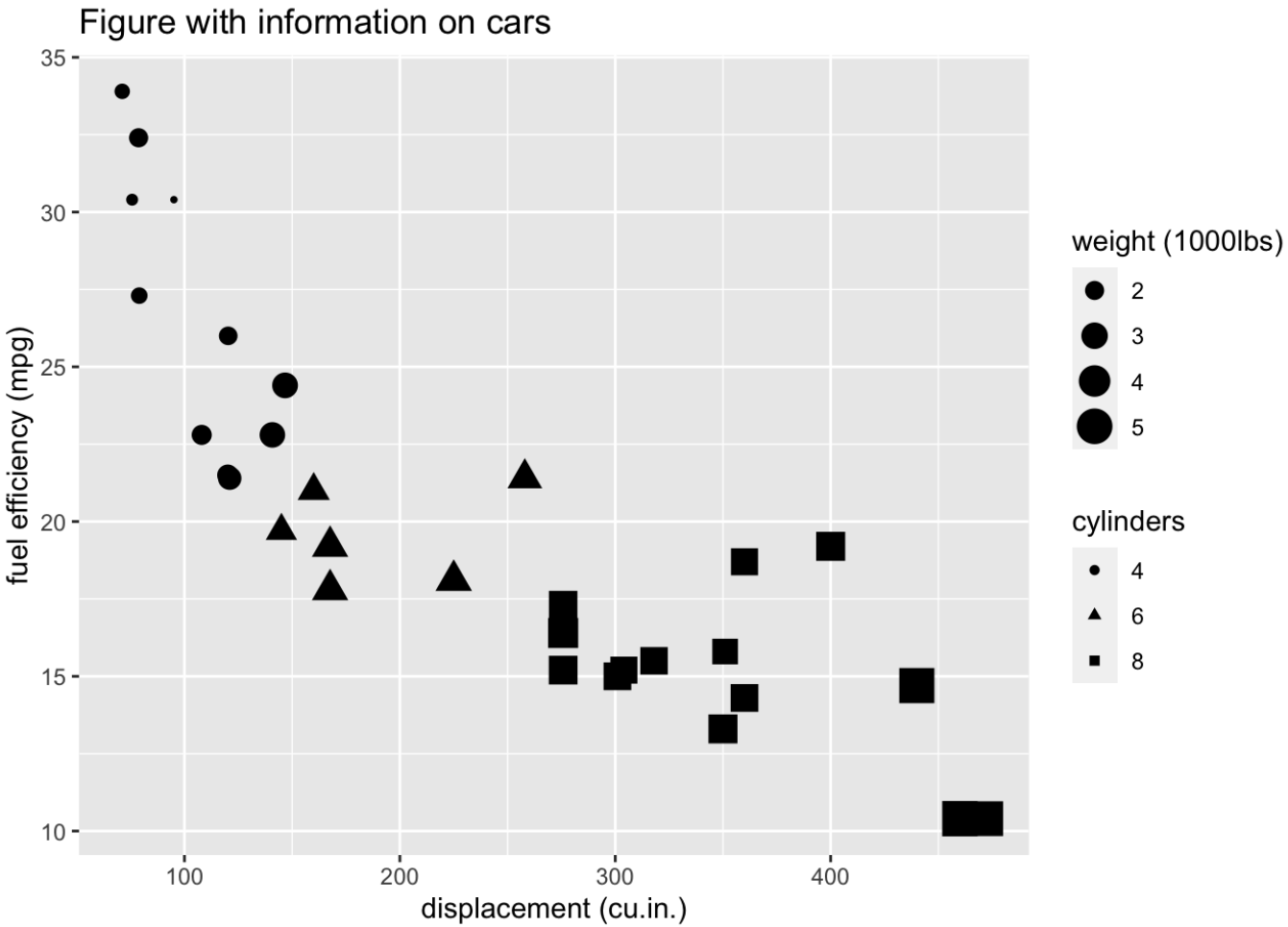

```
ggplot(data = mtcars,
       mapping = aes(x = disp, y = mpg,
                     color = hp,
                     shape =
                       as.factor(cyl))) +
 geom_point() +
 guides(color = guide_colorbar(order = 1),
        shape = guide_legend(order = 2))
```



Exercise

Exercise: change the axis labels and legends

Make the following plot:



Themes

Themes

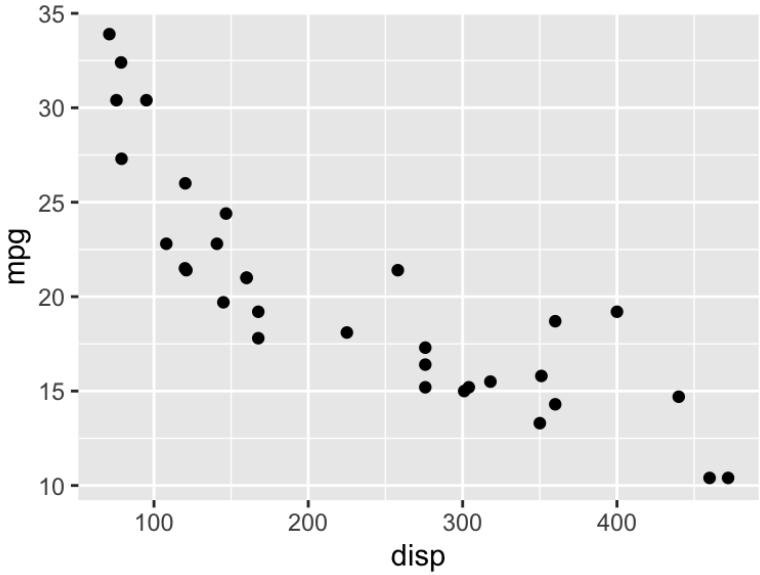
Last week:

- principle of data-ink ratio
- -> use of grid lines

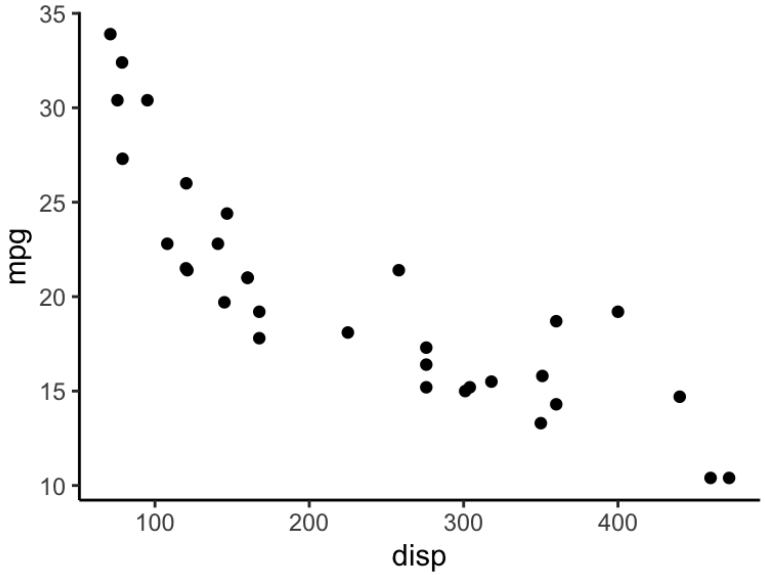
Can be adjusted in ggplot using *themes*.

Themes - examples

```
theme_set(theme_grey()) # default theme
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point()
```

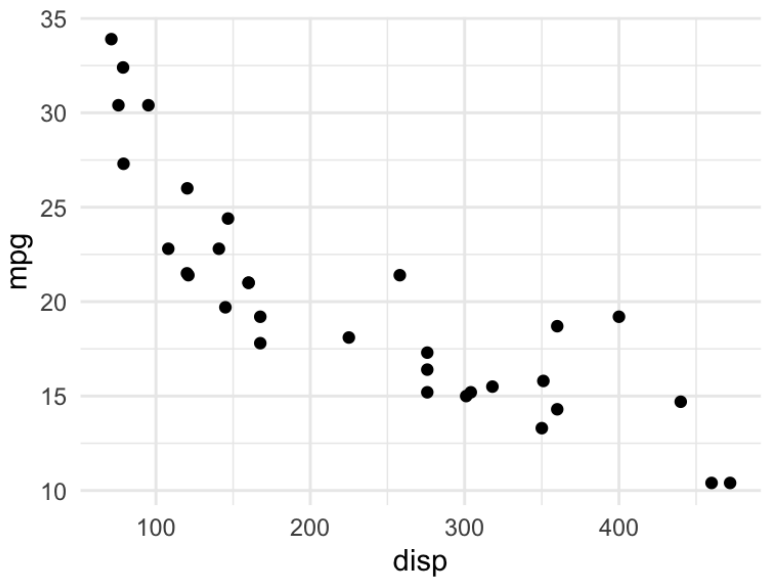



```
theme_set(theme_classic())
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point()
```

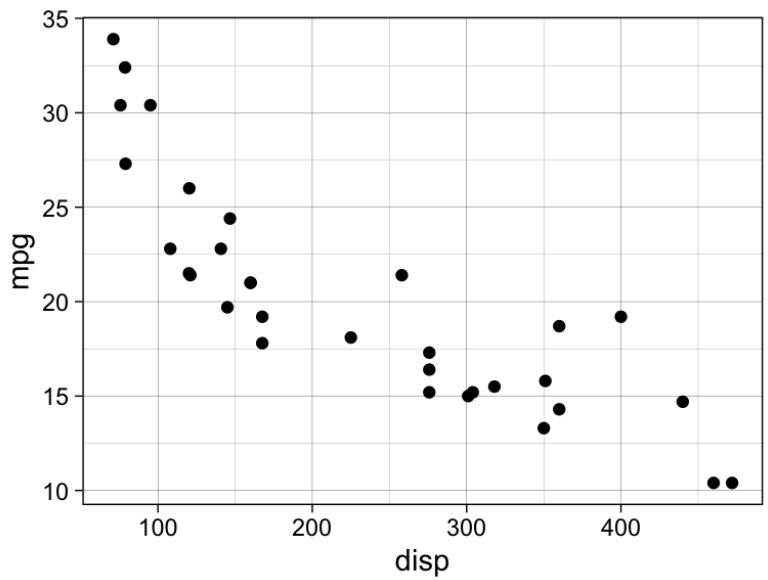


Themes - examples

```
theme_set(theme_minimal())
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point()
```



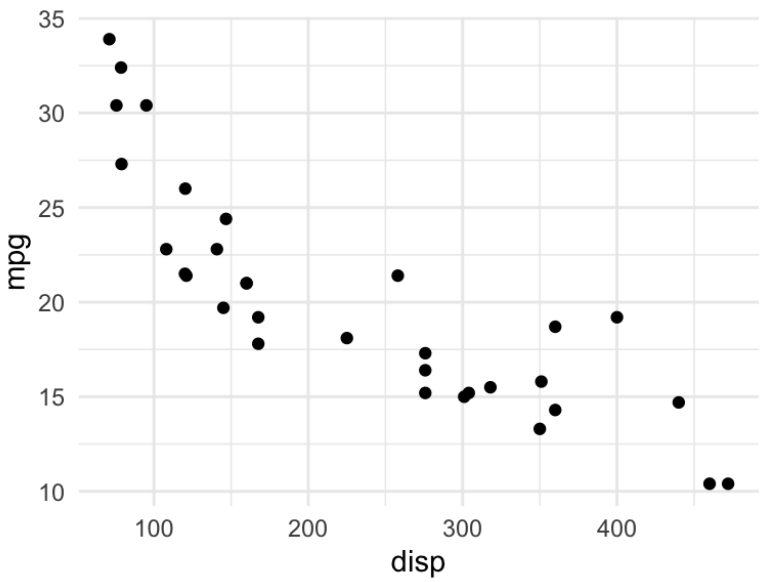

```
theme_set(theme_linedraw())
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point()
```



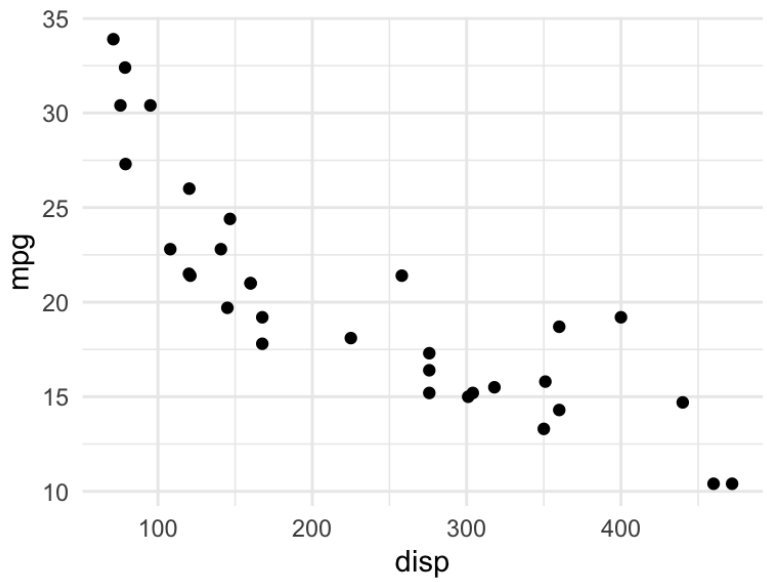
Themes - examples

Set theme either globally or locally:

```
theme_set(theme_minimal())
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point()
```




```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point() +
  theme_minimal()
```




```
# Reset theme to default
theme_set(theme_grey())
```

Themes: make changes

You can change **many** things in a theme:

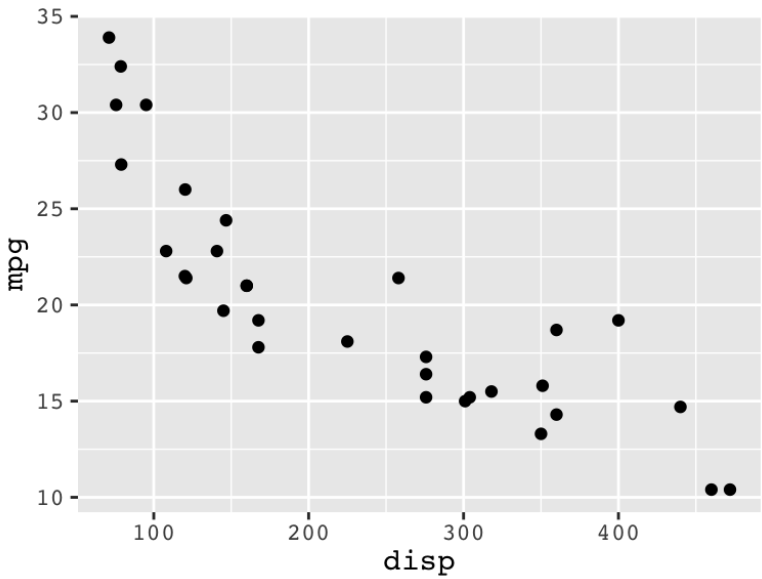
```
theme
```

```
## function (line, rect, text, title, aspect.ratio, axis.title,
##     axis.title.x, axis.title.x.top, axis.title.x.bottom, axis.title.y,
##     axis.title.y.left, axis.title.y.right, axis.text, axis.text.x,
##     axis.text.x.top, axis.text.x.bottom, axis.text.y, axis.text.y.left,
##     axis.text.y.right, axis.ticks, axis.ticks.x, axis.ticks.x.top,
##     axis.ticks.x.bottom, axis.ticks.y, axis.ticks.y.left, axis.ticks.y.right,
##     axis.ticks.length, axis.ticks.length.x, axis.ticks.length.x.top,
##     axis.ticks.length.x.bottom, axis.ticks.length.y, axis.ticks.length.y.left,
##     axis.ticks.length.y.right, axis.line, axis.line.x, axis.line.x.top,
##     axis.line.x.bottom, axis.line.y, axis.line.y.left, axis.line.y.right,
##     legend.background, legend.margin, legend.spacing, legend.spacing.x,
##     legend.spacing.y, legend.key, legend.key.size, legend.key.height,
##     legend.key.width, legend.text, legend.text.align, legend.title,
##     legend.title.align, legend.position, legend.direction, legend.justification,
##     legend.box, legend.box.just, legend.box.margin, legend.box.background,
##     legend.box.spacing, panel.background, panel.border, panel.spacing,
##     panel.spacing.x, panel.spacing.y, panel.grid, panel.grid.major,
##     panel.grid.minor, panel.grid.major.x, panel.grid.major.y,
```

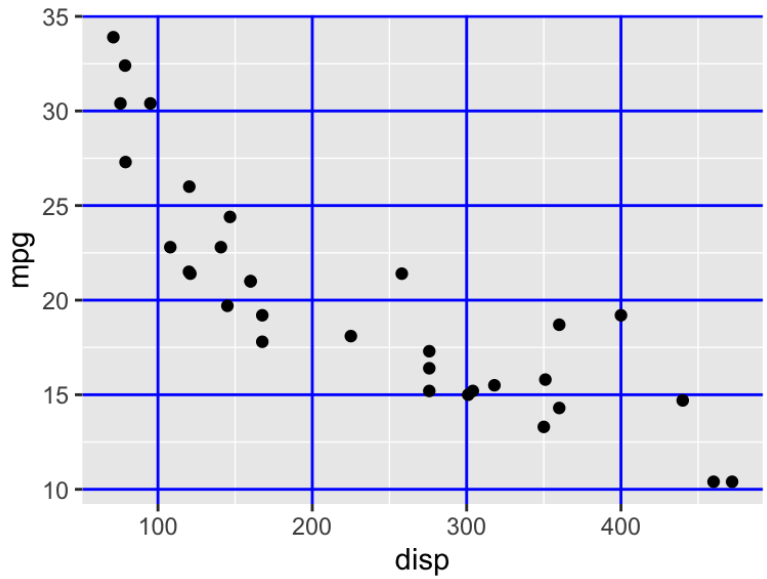
Themes: make changes

You can change **many** things in a theme, example:

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point() +
 theme(text =
         element_text(family = "Courier"
```



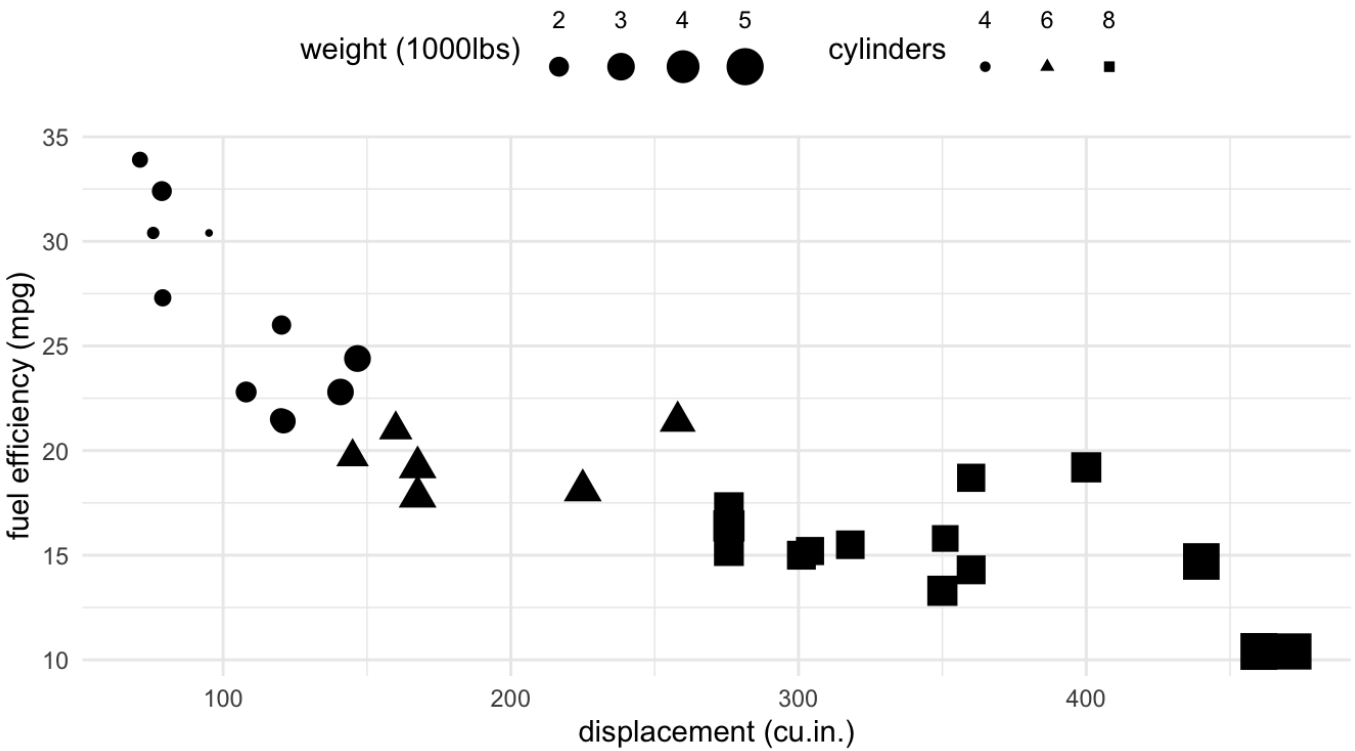

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point() +
 theme(panel.grid.major =
         element_line(colour = "blue"))
```



Exercise

Exercise: choose theme + put the legend at the top

Make the following plot:



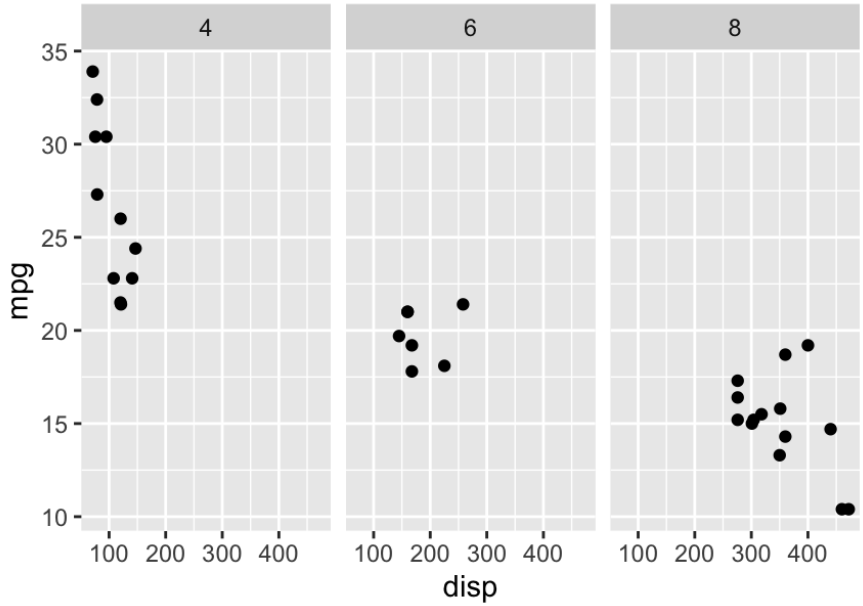
Multiple plots

Facets

Facet: split a plot into multiple plots based on one or more discrete variables

Example: a column for each cylinder size:

```
ggplot(data = mtcars,
       mapping = aes(x = disp,
                     y = mpg)) +
 geom_point() +
  facet_grid(cols = vars(cyl))
```



Different plots together in one grid

We recommend to use the package “patchwork” as it is very flexible

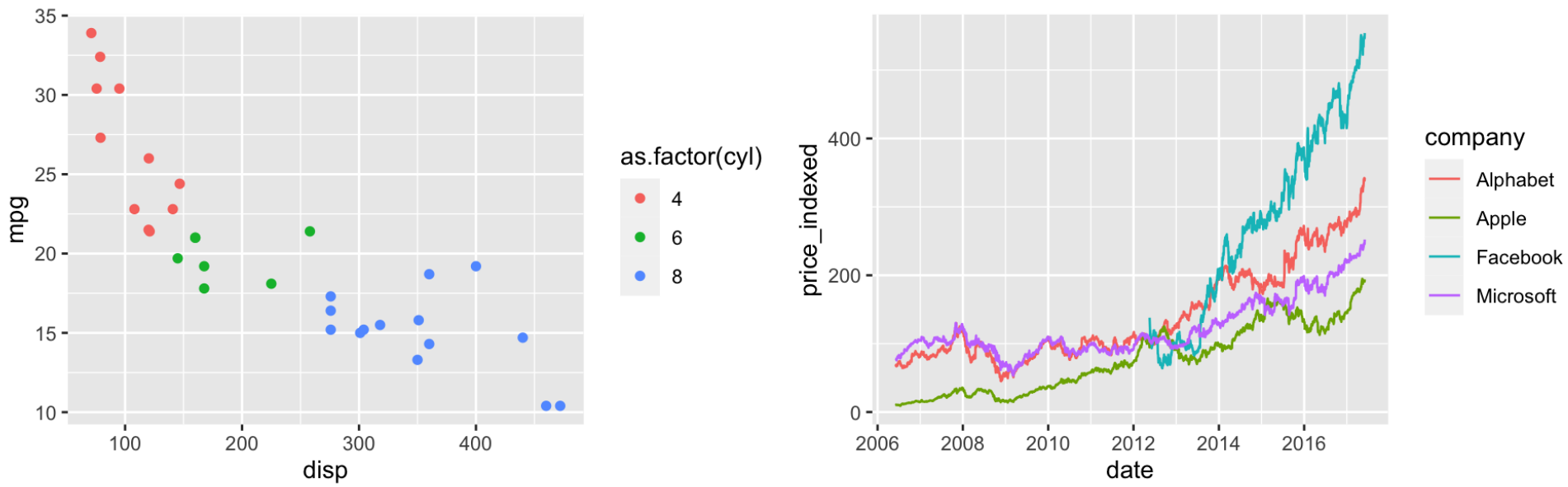
Suppose we have two plots:

```
scatterplot_cars <- ggplot(data = mtcars,
                           mapping = aes(x = disp,
                                         y = mpg,
                                         color = as.factor(cyl))) +
 geom_point()

lineplot_stocks <- ggplot(data = stocks,
                           mapping =
                             aes(x = date,
                                 y = price_indexed,
                                 color = company)) +
  geom_line()
```

Different plots together in one grid

Then you can easily combine them using the patchwork package:

```
library(patchwork)

patchwork <- scatterplot_cars + lineplot_stocks
patchwork
```



Different plots together in one grid

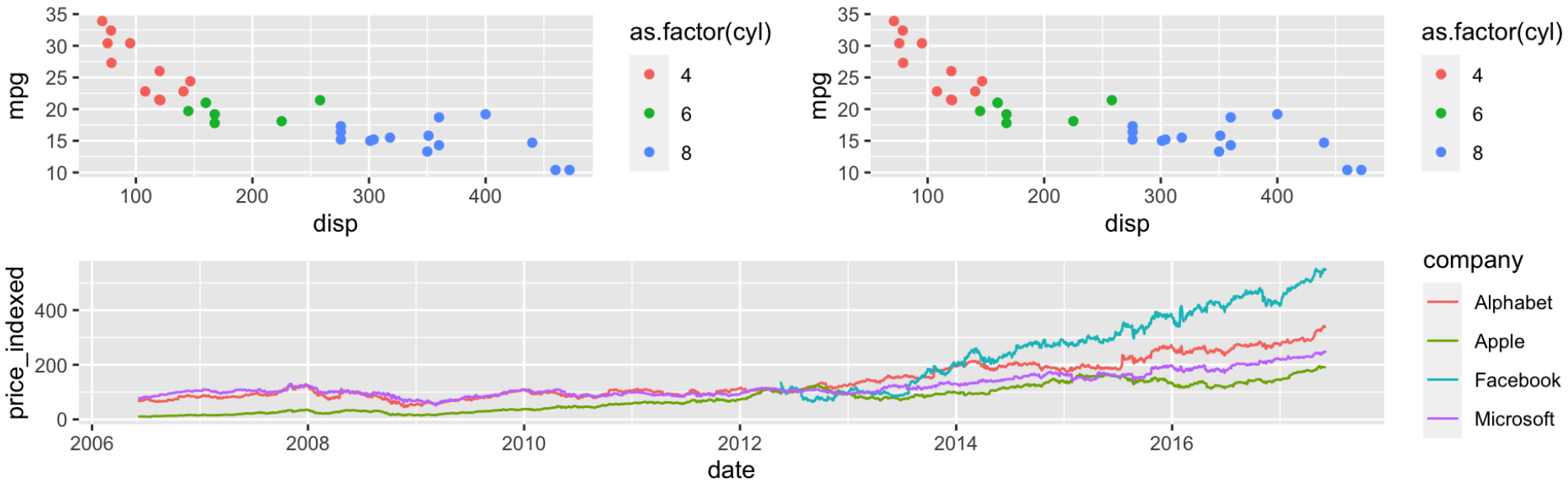
And add annotation for the whole combination of figures:

```
patchwork + plot_annotation(
  title = 'Two plots together',
  subtitle = "So beautiful!",
  caption = "Note: these plots are unrelated"
)
```

Different plots together in one grid

It's also easy to combine more plots:

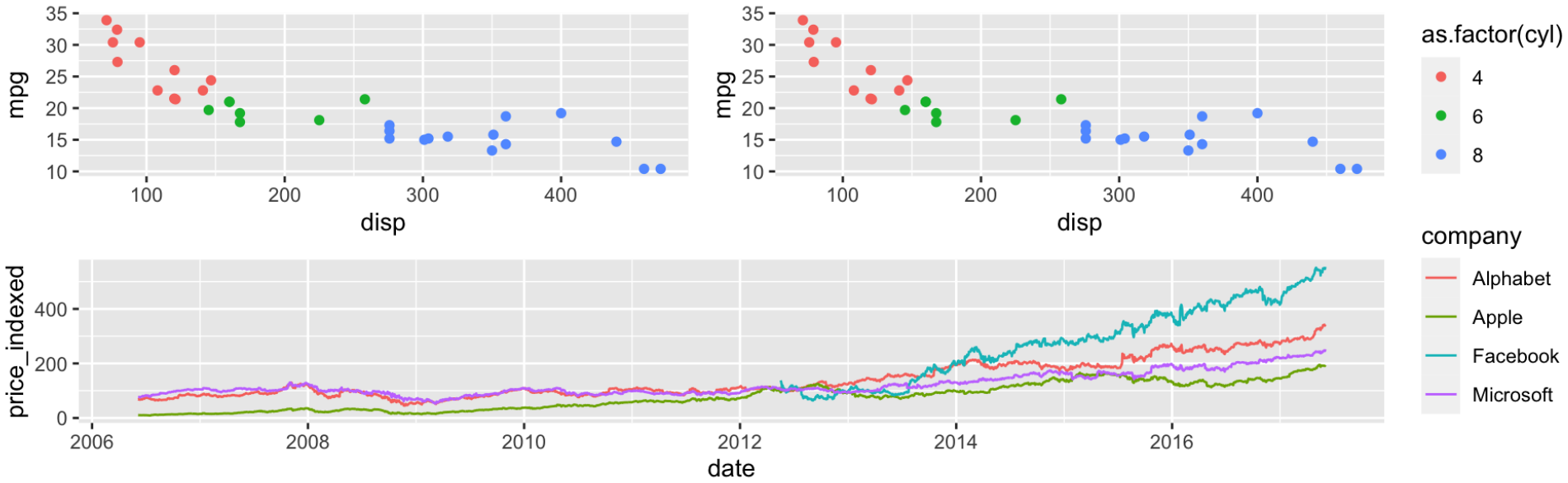
```
(scatterplot_cars + scatterplot_cars) / lineplot_stocks
```



Different plots together in one grid

Several legends that are the same? Collect them!

```
(scatterplot_cars + scatterplot_cars) / lineplot_stocks +
 plot_layout(guides = "collect")
```



Challenge!

Final exercise: put many things together!

Challenge: Mimic Figure 2.5 of the book

- Start with the easiest steps that were already in previous exercises.
- Use help files, internet, cheat sheet, etc. to find out how to tweak other details.