INTERFACULTY CENTER
DATA PROCESSING & STATISTICS
CORE FACILITY

Data Visualization with R's tidyverse

ggplot2

W. Cools

Compiled Sep 06, 2020

Current draft aims to introduce researchers to visualizing both data and statistics in R with the GGplot package.
Our target audience is primarily the research community at VUB / UZ Brussel, those who have some basic experience in R and want to know more.

We invite you to help improve this document by sending us feedback
wilfried.cools@vub.be or anonymously at icds.be/consulting (right side, bottom)

Key message on data visualization

Data visualization is inherent to data analysis, not just a way of communicating the results.

- visualization can focus on data and/or its summarizing statistics
  - to convince yourself, understand data and statistical results
  - to convince others, show results to support inference
- no -fit's all data visualization-, dependent on analysis
- flexible use of data visualization
  - supports more informative and complete visualizations
  - elicits better data exploration and modeling

Data visualization is best done with coding (as opposed to manual changes).

- maintain structure and transparency, to support reproducibility
  - efficient refinement and adjustment
  - less error prone

Data visualization is easier and more intuitive when maintaining tidy data.

- tidy data: meaning appropriately mapped into structure
  - each row an observation as research unit,
  - each column a variable as property,
  - each cell a particular value, linking row to column
  - note: data can be split into multiple tables (relational data).
- aim for tidy data registration (avoid tedious manipulations)

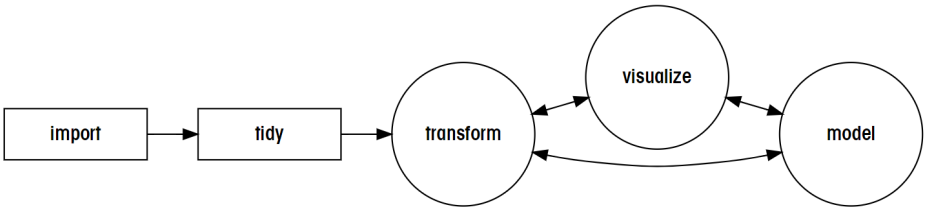


Figure 1: workflow: tidyverse lingo

R's tidyverse package: ggplot2

Focus in current draft is on R.

- free, hugely flexible, large community online to help out
- offers general functionality for visualization
  - base R
  - grid
  - trellis/lattice graphics
  - ggplot2 (tidyverse) & ggviz
- offers many packages with dedicated functionality

`ggplot2` of the `tidyverse` package (Hadley Wickham etal.).

- build on idea of Grammar of Graphics (Leland Wilkinson)
  - largely consistent
  - but still highly flexible
- tends to outperform base R for more complex visualizations
- explicitly links to tidy data
- but requires extensions for 3D plotting and interactive graphics
- but does not allow for multiple Y-axes (combination of axes)

Install (at least once) and load (once per R session) the `tidyverse` package, or the `ggplot2` package.

```
install.packages('tidyverse')
```

```
library(tidyverse)
```

Find a convenient cheat sheet on data visualization at https://rstudio.com/resources/cheatsheets/

Examples to get started

Highlighting both base R and ggplot to get a first impression.

To use the build-in iris data, include it with `data( )`.

```
data(iris)
```

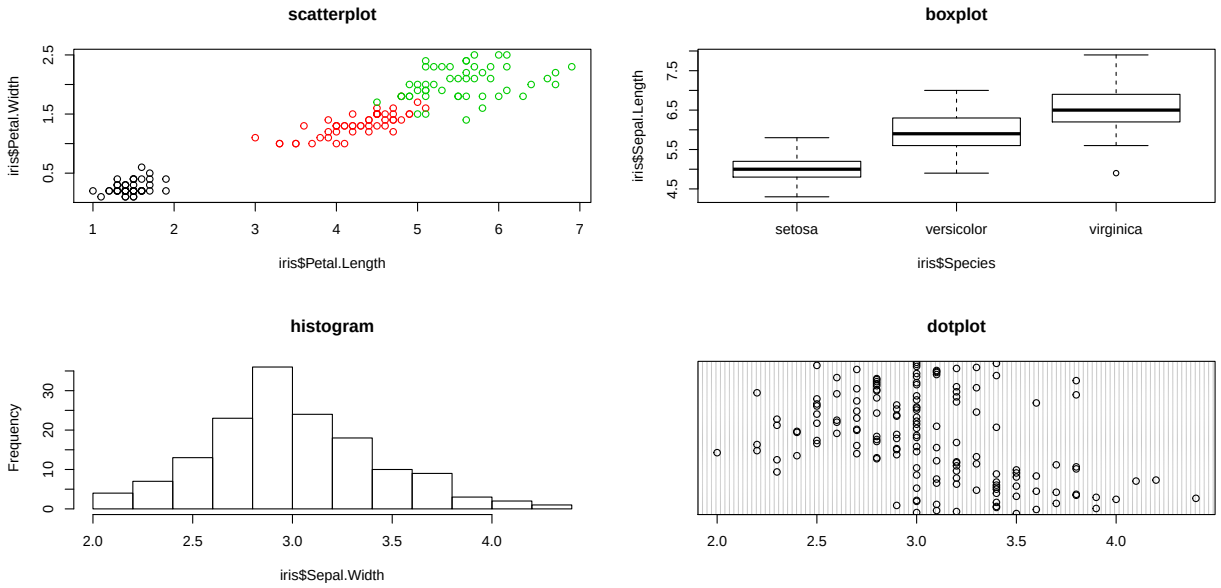
Have a peak at it's contents with `str( )` and `head( )`.

base R

Various visualizations of the data are possible, also in base R.

For example, consider the scatterplot, boxplot, histogram and dotplot.

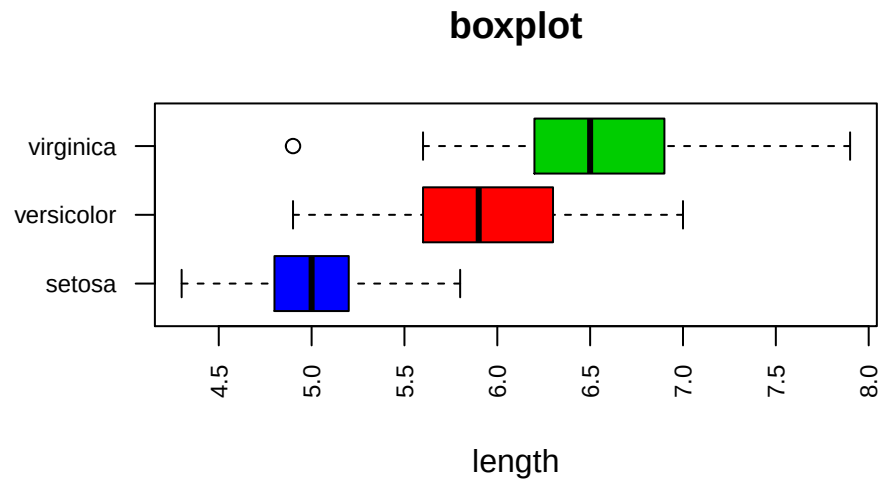
```
par(mfrow=c(2,2))
plot(iris$Petal.Length,iris$Petal.Width,col=iris$Species,main='scatterplot')
boxplot(iris$Sepal.Length~iris$Species,main='boxplot')
hist(iris$Sepal.Width,main='histogram')
dotchart(iris$Sepal.Width,main='dotplot')
```




```
par(mfrow=c(1,1))
```

- plots generated in a single function call, with various parameters
- further fine-tuning, with legends, annotations, and much more, possible; check the helpfile on `?par` and `?options`

```
boxplot(iris$Sepal.Length~iris$Species,main='boxplot',
    horizontal=TRUE, las=2, cex.axis=.75, ylab='',xlab='length',col=c(4,2,3));
```

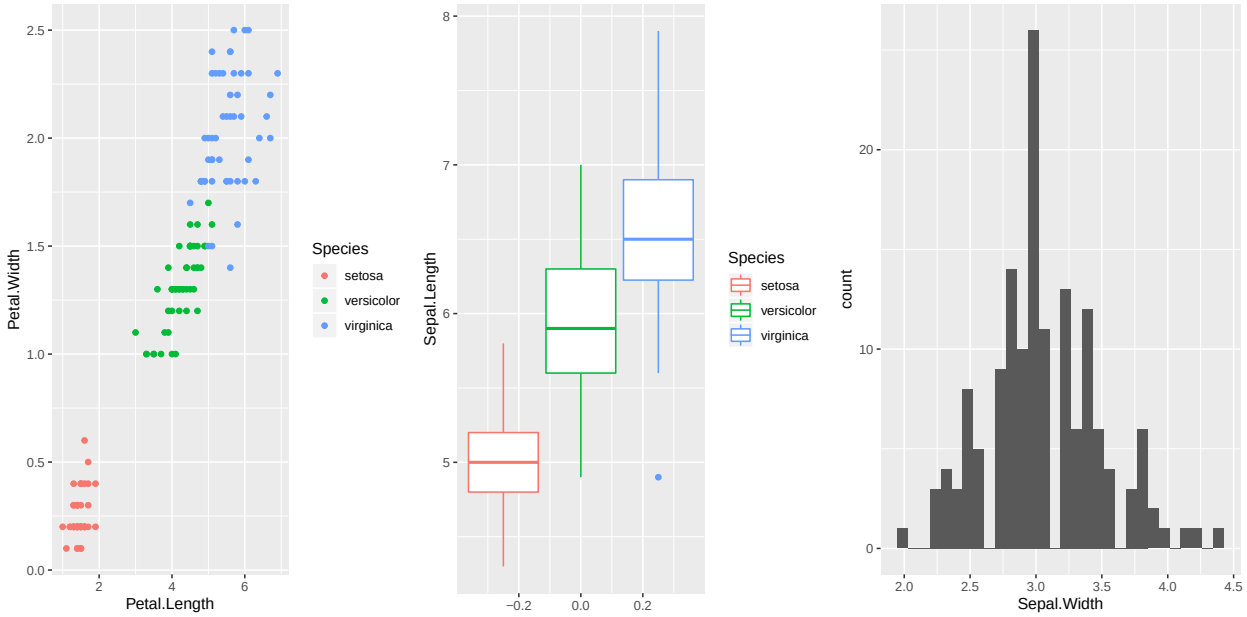


ggplot2

Visualization, especially when slightly more complex, is often more intuitive with `ggplot2`.

For example, consider the scatterplot, boxplot, and histogram.

```
p1 <- ggplot(data=iris,aes(y=Petal.Width,x=Petal.Length,col=Species)) + geom_point()
p2 <- ggplot(data=iris,aes(y=Sepal.Length,col=Species)) + geom_boxplot()
p3 <- ggplot(data=iris,aes(x=Sepal.Width)) + geom_histogram()
grid.arrange(p1, p2, p3, ncol=3)
```



- plots generated by creating an R object, and adding layers to it for visualization
- further fine-tuning, with legends, annotations, and much more, done with layers too

To save the last generated plot, the `ggsave( )` function is available.

```
ggsave('plotname.png',width=12,height=6)
```

conclusion

Both types, and other types of visualization will do what they are supposed to.
Because ggplot is taking over, and because it is so much fun to work with, ggplot is presented.

-gg- : grammar of graphics

GGplot philosophy: Grammar of Graphics (Leland Wilkinson)

- build visualization like making a sentence
  - independently specify building blocks
  - combine blocks to create any graphical display
- example of layered building blocks with `ggplot( )`:
  - `ggplot(data=iris,aes(y=Petal.Width,x=Petal.Length,col=Species)) + geom_point()`
  - `ggplot( )` creates a ggplot object
    - data argument to assign the dataframe (or tibble)
    - `aes( )` function to assign variables from the dataframe to scales (x-axis, y-axis, color)
  - `geom_point( )` to visualize the ggplot object as a scatterplot

wilfried.cools@vub.be

∗ note: scatterplots require an x and y-axis, linked to variables in a dataframe

General structure includes data, functions and arguments.

- functions
  - `ggplot( )` function to initialize the ggplot object
  - `geom_*( )` function to visualize data through their aesthetics
  - `stat_*( )` function largely similar to `geom` but with focus on statistics
  - `facet_*( )` function for grouping / conditional visualization
  - `theme( )`, `guides( )`, `scale_*( )`, `coord_*( )`
- arguments in `ggplot( )`, `geom_*( )` and `stat_*( )`
  - `data` argument to specify data (=input)
  - `aes( )` function as argument to specify aesthetic mapping (bridging gap input and output)
  - `...`

Grammer of Graphics sparked further developments

- other packages following the lead of GG offer additional functionality: `ggforce`, `ggalt`, `ggpubr`, `ggraph`, `tidygraph`, `GGally`, `ggcorrplot`, `ggridges`, `....`

Visualization essentials

First part addresses how to make a visualization,
afterwards it is considered in less detail how to further make refinements.

step by step example

Use is made of the build-in `mtcars` and already loaded build-in `iris` dataset.

```
data(mtcars)
```

```
mtcars %>% head()
```

```
|                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
| Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
| Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
| Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
| Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
| Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
| Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

The ggplot object is constructed.

- data is assigned the `mtcars` dataframe
- the x and y aesthetic is linked to the variables `mpg` and `disp` from the `mtcars` data

```
p1 <- ggplot(data=mtcars, aes(y=mpg,x=disp))
```

No visualization is made yet, only the object is created.

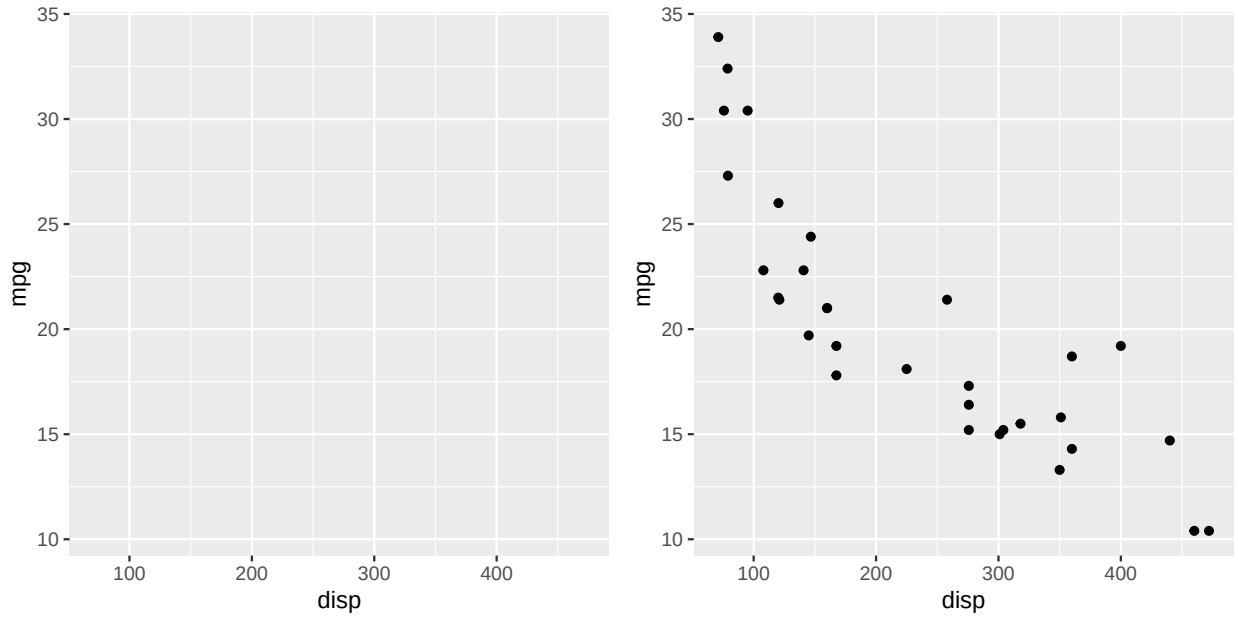
- an internal representation exists, ready for visualization
  - aesthetics x and y are given their default values using the `mtcars`
  - includes a legend and scale for both x and y axis (not visualized yet)
- any geometric object (function) that at least uses an x and y axis can be added as a layer

A layer is added with the `+` sign, adding a geometric function.

```
p2 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) + geom_point()
```

- `geom_point( )`: geometric function without arguments to visualize a scatterplot
  - requires data, an x and a y-axis
    - if not specified, inherit from `ggplot( )`
    - if specified, overwrite `ggplot( )`
  - example: possible, but beware not to overwrite the default x and y
    - `ggplot(data=mtcars, aes(x=disp)) + geom_point(aes(y=mpg))`
    - `geom_point( )` requires x and y
    - x is not specified inside, but inherited from `ggplot( )`
    - y is specified inside, within `aes( )` because linked to data (mpg)
    - data is not specified inside, but inherited from `ggplot( )`

```
grid.arrange(p1, p2, ncol=2)
```



Non-essential aesthetics can be included, for example color.

- specified within the `aes( )`, values for color are extracted from the variable `gear` in `mtcars`
  - the numerical (continuous) variable gear is assigned a continuous scale of colors
  - a legend for continuous variables is by default included
  - note: color should be considered a third dimension

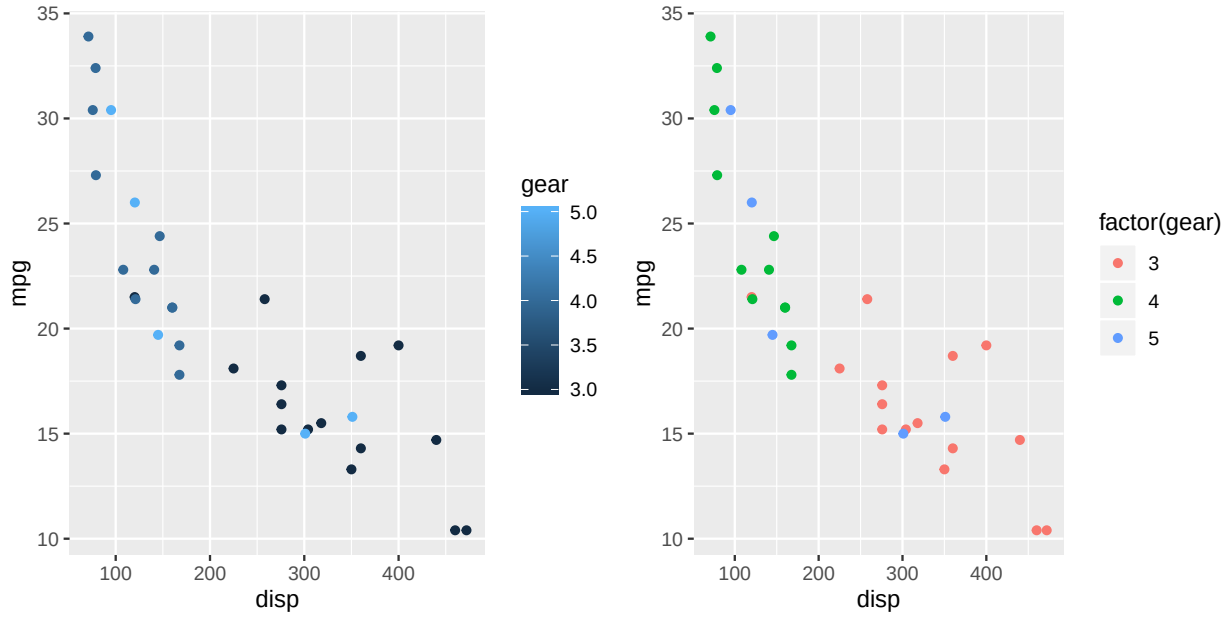
```
p1 <- ggplot(data=mtcars, aes(y=mpg,x=disp,color=gear)) + geom_point()
```

- specified within the `aes( )`, values for color are extracted from the categorical variable `gear` in `mtcars`
  - the factor (categorical) variable gear is assigned a nominal scale of colors
  - a legend for categories is by default included for this third dimension

```
p2 <- ggplot(data=mtcars, aes(y=mpg,x=disp,color=factor(gear))) + geom_point()
```

```
grid.arrange(p1, p2, ncol=2)
```


wilfried.cools@vub.be



- specified within the `aes( )` of the geometric function, the default is overwritten.
  - the plot remains the same
  - the categorical version in `geom_point( )` overwrites the numerical in `ggplot( )`

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

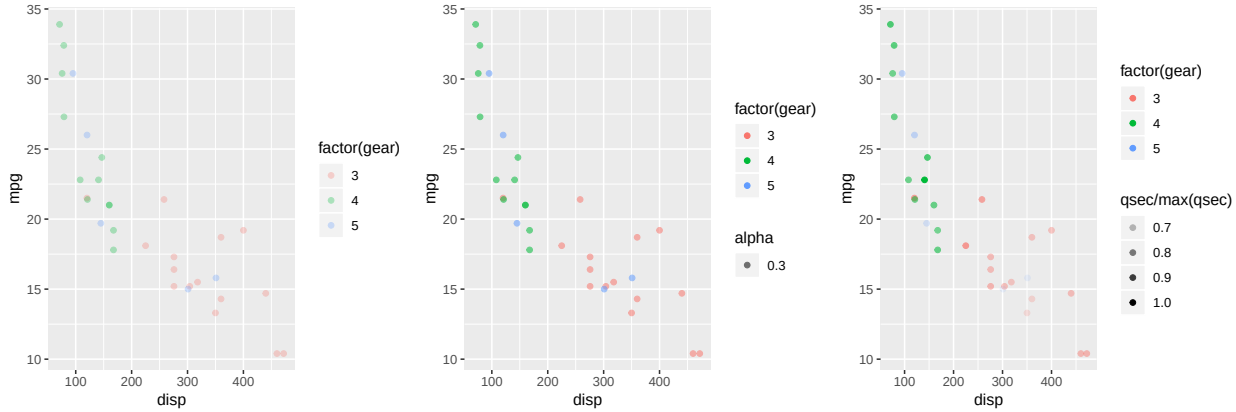
- specified outside the `aes( )` no relation with data exists
  - color is now assigned a value independent of the actual data
  - color from the `ggplot( )` is overwritten
  - remember: `aes( )` is essential for linking aesthetics with data

```
ggplot(data=mtcars, aes(y=mpg,x=disp,color=gear)) + geom_point(color='#FF6600')
```

- multiple aesthetics can be specified, in and outside the `aes( )`
  - with color dependent on gear
  - alpha (transparency) outside of `aes( )` at .3 (30%)
  - alpha inside of `aes( )` assumes a link to a variable, not a percentage as such
    - alpha as a percentage (avoid this, it gives unwanted behavior)
    - alpha related to variables with values between 0 and 1

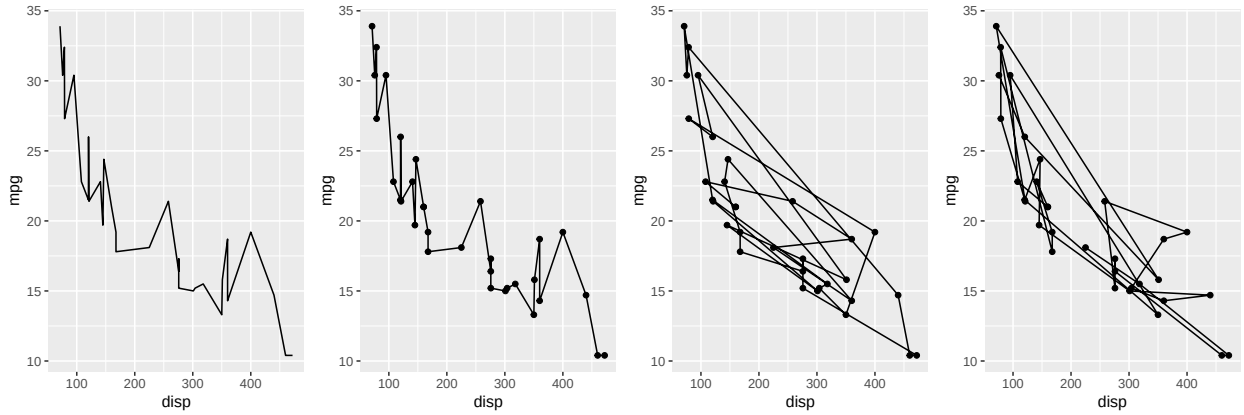
```
p1 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) + geom_point(aes(color=factor(gear)),alpha=.3)
p2 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) + geom_point(aes(color=factor(gear),alpha=.3))
p3 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) +
    geom_point(aes(color=factor(gear),alpha=qsec/max(qsec)))
grid.arrange(p1, p2, p3, ncol=3)
```


wilfried.cools@vub.be



- multiple geometric functions can be included
  - `geom_point( )` creates the dots, `geom_line( )` connects them over the x-axis
  - `geom_path( )` simply connects observations as presented in the data (re-ordering has an effect)

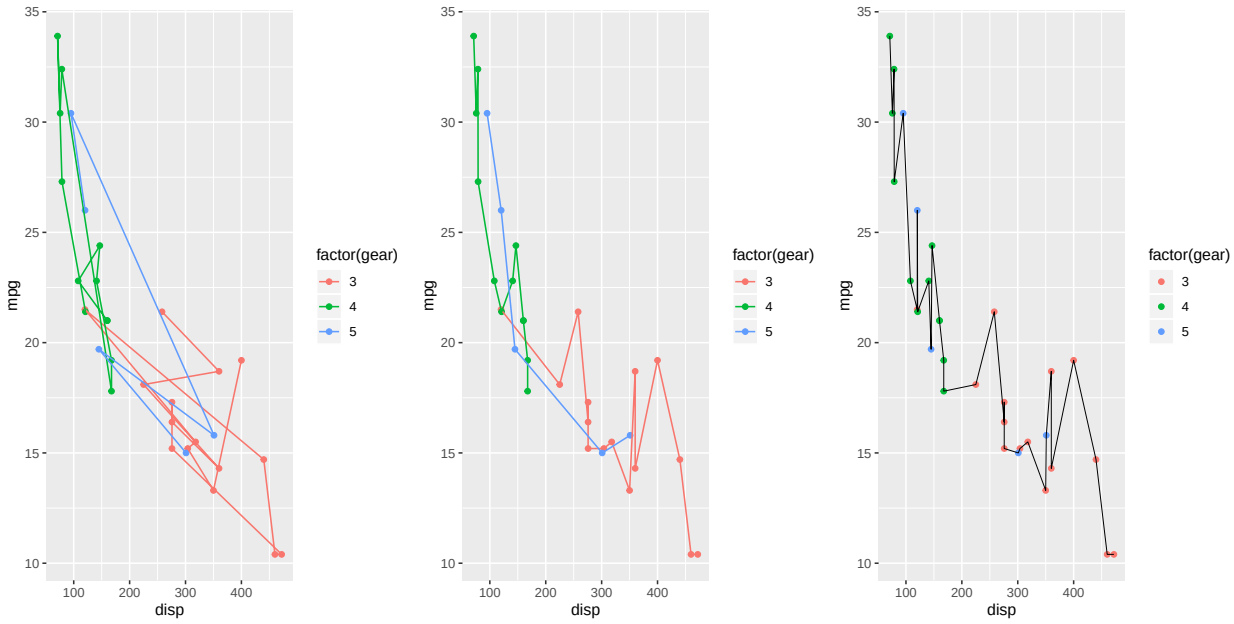
```
p1 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) + geom_line()
p2 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) + geom_point() + geom_line()
p3 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) + geom_point() + geom_path()
p4 <- ggplot(data=mtcars %>% arrange(drat), aes(y=mpg,x=disp)) + geom_point() + geom_path()
grid.arrange(p1, p2, p3, p4, ncol=4)
```



- `aes( )` in `ggplot( )` controls part of the result of geometric functions
  - assigning a categorical variable to color is sufficient to group the lines (path)
  - the same happens with `geom_line( )`
  - specifications can be made in `ggplot( )` for default behavior, or inside the `geom_*( )` for local behavior
    - note that the `geom_point( )` uses the locally specified color aesthetic
    - note that the `geom_line( )` uses the default black (size made smaller than default)
  - beware: types (discrete/continuous) should agree when overwriting `ggplot(aes( ))`

```
p1 <- ggplot(data=mtcars, aes(y=mpg,x=disp,color=factor(gear))) + geom_point() + geom_path()
p2 <- ggplot(data=mtcars, aes(y=mpg,x=disp,color=factor(gear))) + geom_point() + geom_line()
p3 <- ggplot(data=mtcars, aes(y=mpg,x=disp)) +
    geom_point(aes(color=factor(gear))) + geom_line(size=.3)
grid.arrange(p1, p2, p3, ncol=3)
```

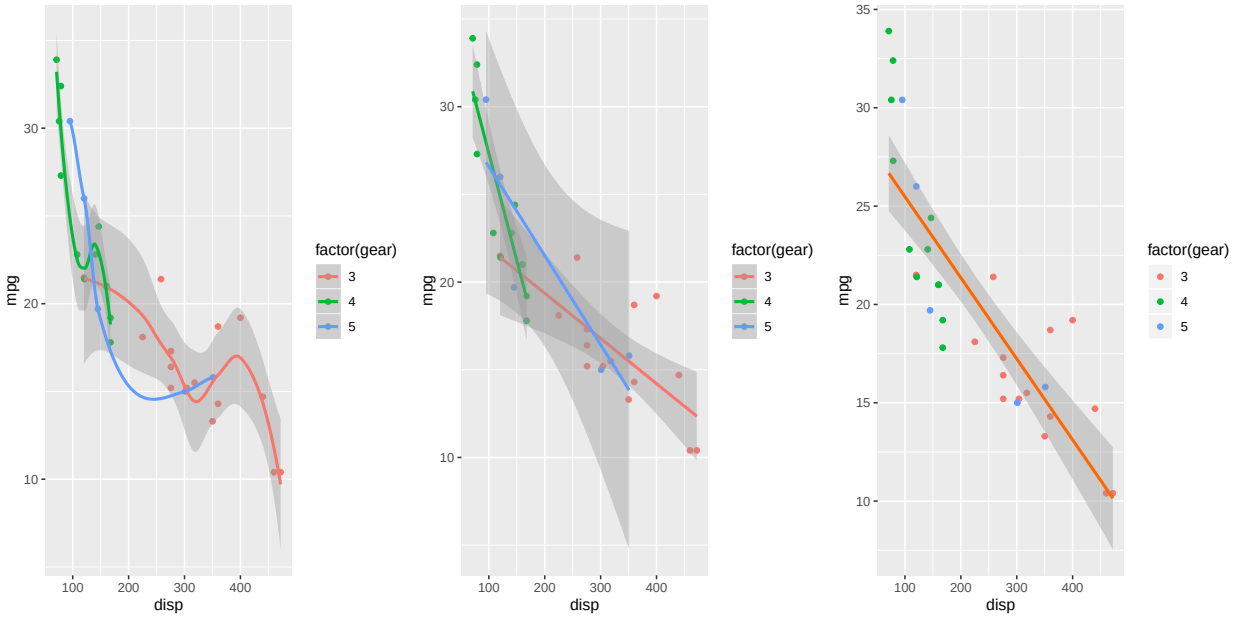

wilfried.cools@vub.be



- geometric functions that add statistics can be included as well
  - `geom_smooth( )` offers averaging and standard errors
    - local averages are default (loess lines), global conditional averages can be requested
  - grouping through aesthetics works like before
    - assigning a 1 to color causes all observations to belong to one group
    - the default blue is overwritten to orange
  - check `?geom_smooth` for more details

```
px <- ggplot(data=mtcars, aes(y=mpg,x=disp,color=factor(gear)))
p1 <- px + geom_point() + geom_smooth()
p2 <- px + geom_point() + geom_smooth(method='lm')
p3 <- ggplot(data=mtcars, aes(y=mpg,x=disp,color=factor(gear),group=1)) + geom_point() +
    geom_smooth(method='lm',color="#FF6600")
grid.arrange(p1, p2, p3, ncol=3)
```


wilfried.cools@vub.be



ggplot( )

Function to create a ggplot object, ready for visualization.

- constructor, always required
- prepares appropriate internal representation
- can include default data
- can include default aesthetics

aes( )

Function to link variables (data) to an aesthetic (dimension).

- used within `ggplot( )` or `geom_*( )`
- requires variables (part of data)
- defines aesthetic dependent on variables
  - x and y axes: values are linked to positions on the axes
  - fill: values are linked to a color
  - color: values are linked to a color (border for surfaces)
  - shape: values are linked to a shape
  - size: values are linked to a size (of a symbol, point or line)
- group argument used inside `aes( )` to identify groups of observations
  - allows grouping without assigning an aesthetic
  - use the value `1` to combine all observations into one group
- automatically assigns a default legend
  - relates to aesthetic
  - depends on variable type (nominal, ordinal, continuous)
- note: aesthetics defined outside of `aes( )` are independent of the data

wilfried.cools@vub.be

geom_*( )

Function to use the internal ggplot representation and turn it into a visualization.

- creates particular visualization (scatterplot, histogram, …)
  - requires a particular minimum set of aesthetics
    - scatterplots require x and y axis
    - histogram requires x axis
  - typically can be assigned additional aesthetics optionally (eg., color)
- can include the data argument to overwrite the default data from `ggplot( )`
- can include `aes( )` to overwrite the default `aes( )` from `ggplot( )`
  - useful to add or change aesthetics

Visualization extras

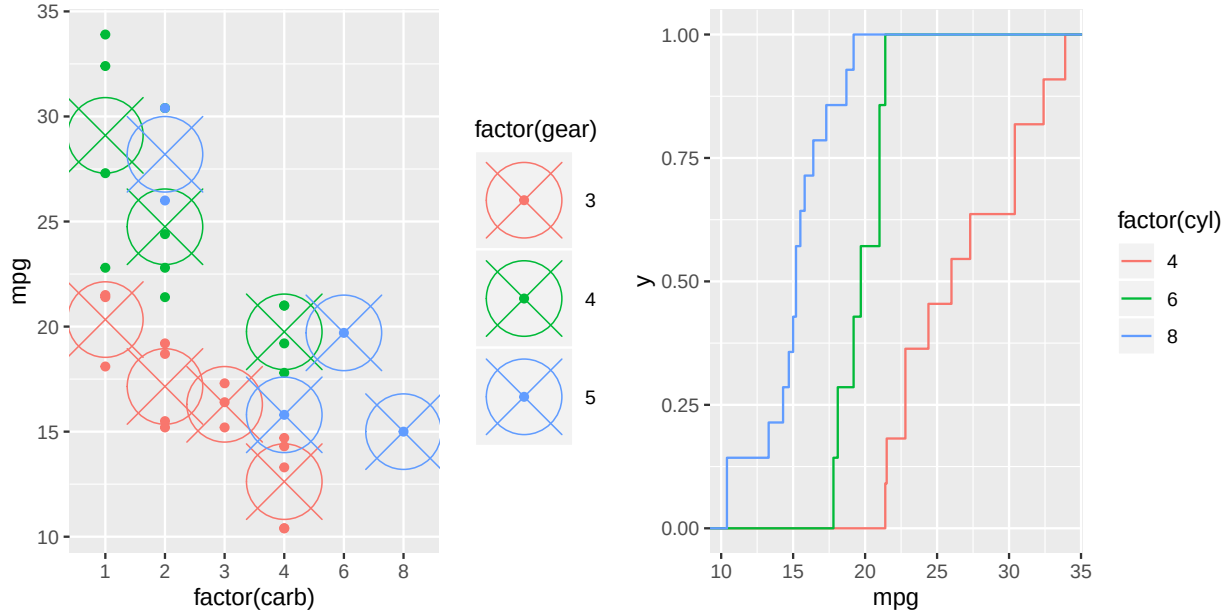
While a ggplot object, layered with `geom_*( )` functions allows you to visualize data and statistics, there is so much more to ggplot.

geom and stat layers

Geometric functions add layers on top of the ggplot object, as do statistical transformation functions.

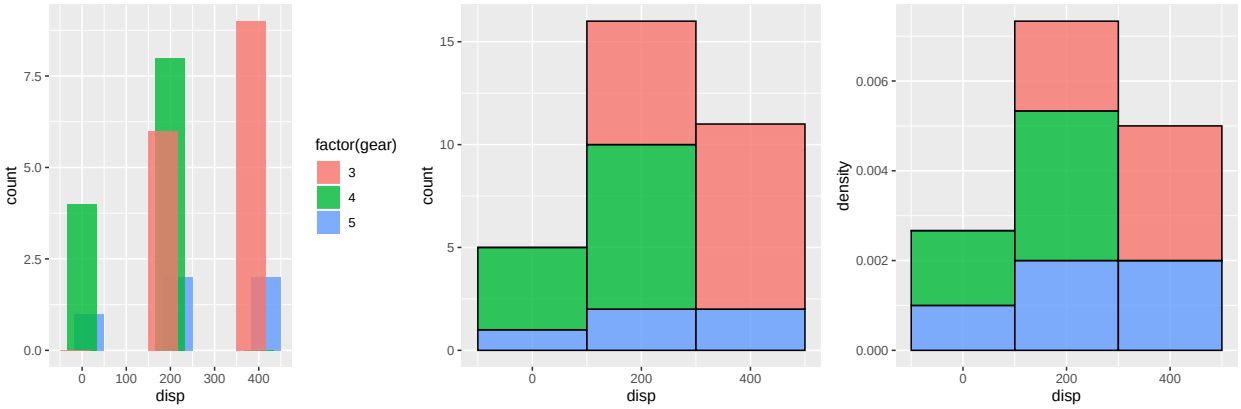
- every `geom_*( )` has a default stat argument, and every `stat_*( )` has a default geom argument
  - note: `stat_smooth(geom="smooth")` and `geom_smooth(stat="smooth")` are equivalent
  - same result:
    - `+ geom_point(stat='summary',fun.y='mean',shape=13,size=16)`
    - `+ stat_summary(geom='point',fun.y='mean',shape=13,size=16)`
  - in most cases it is possible to stick to `geom_*( )`, in some not like ecdf.

```
p1 <- ggplot(data=mtcars, aes(y=mpg,x=factor(carb),color=factor(gear))) + geom_point() +
    stat_summary(geom='point',fun.y='mean',shape=13,size=16)
p2 <- ggplot(mtcars, aes(mpg)) + stat_ecdf(aes(color=factor(cyl)),geom = "step")
grid.arrange(p1, p2, ncol=2)
```



- A layer is a function with arguments
  - data (see before)
  - mapping (aesthetic), defined by the `aes( )` (see before)
  - geom (eg., point or smooth)
  - stat (eg., identity or smooth)
  - position (eg., identity)
- Adjustment positions with position argument
  - identity, typically the default
  - jitter is convenient for points and lines (small random position adjustment)
  - stack, fill and dodge is convenient for bars (on top or next to)

```
p0 <- ggplot(data=mtcars,aes(x=disp,fill=factor(gear)))
p1 <- p0 + geom_histogram(binwidth=200, position = position_dodge(width=50),alpha=.8)
p2 <- p0 + geom_histogram(binwidth=200, position = position_stack(),alpha=.8,col='black') +
    theme(legend.position='none')
p3 <- p0 + geom_histogram(aes(y=..density..),binwidth=200,
    position = position_stack(),alpha=.8,col='black') + theme(legend.position='none')
grid.arrange(p1, p2, p3, ncol=3)
```



- note in the above histogram, a generated variable is used `..density..`

wilfried.cools@vub.be

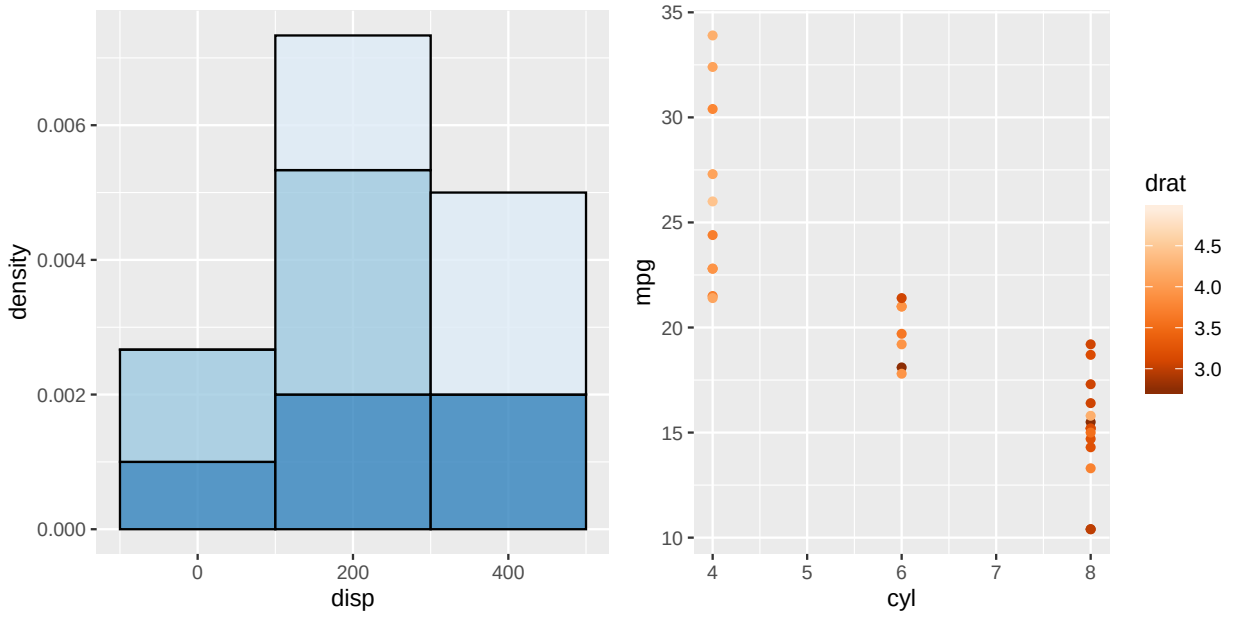
  - for particular `geom_*( )` and `stat_*( )` particular variables are pre-defined
  - generated variables start and end with `..`

scales_*_*( )

Each aesthetic has a scale, which serves as a legend that helps interpretation of the visualized values.

- scales are referred to with aesthetic and type
  - structure: scale___
  - arguments allow to control titles, breaks, labels, limits, … see the help file
  - examples:
    - scale_x_continuous to assign continuous scale to x-axis aesthetic
    - scale_x_sqrt to square root transform the x-axis aesthetic
    - scale_color_brewer to assign brewer colors to the discrete color aesthetic
    - scale_color_distiller to assign brewer colors to the continuous color aesthetic
    - scale_fill_gradient to assign colors to the fill aesthetic
    - scale_fill_manual to manually assign colors to the fill aesthetic
  - scales can be visualized by the axes (x and y), or by the legend (eg. color)
  - note: guide argument requires either a name or `guides()` function for additional control
- scale types have impact on the visualization (continuous vs. categorical)

```
p1 <- ggplot(data=mtcars,aes(x=disp,fill=factor(gear))) +
    geom_histogram(aes(y=..density..),binwidth=200, position = position_stack(),alpha=.8,col='black') +
    theme(legend.position='none') + scale_fill_brewer()
p2 <- ggplot(data=mtcars,aes(y=mpg,x=cyl,color=drat)) + geom_point() +
    scale_color_distiller(palette='Oranges')
grid.arrange(p1, p2, ncol=2)
```



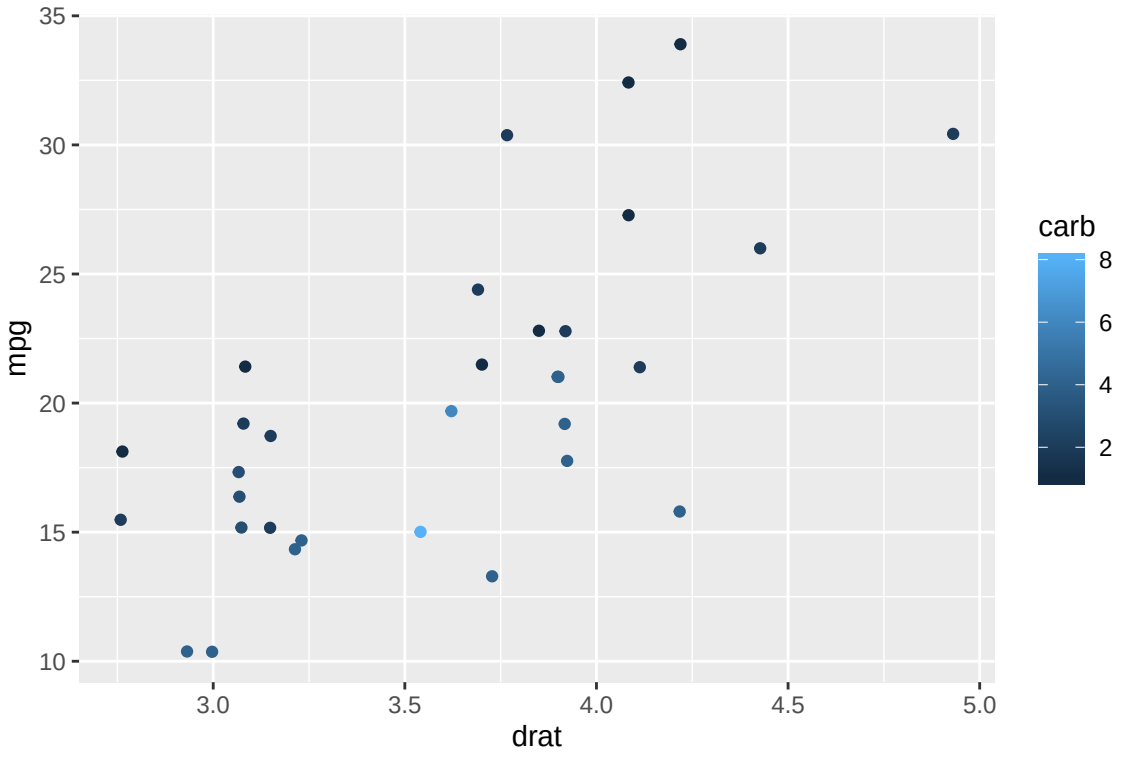
facet_*( )

Visualizations can be split for subgroups, facilitating conditional comparisons.

wilfried.cools@vub.be

- facets can be useful to keep plots simple
- by default, the axes are kept constant for comparison, changeable
- `facet_grid( )` uses a grid, `facet_wrap( )` keeps filling space

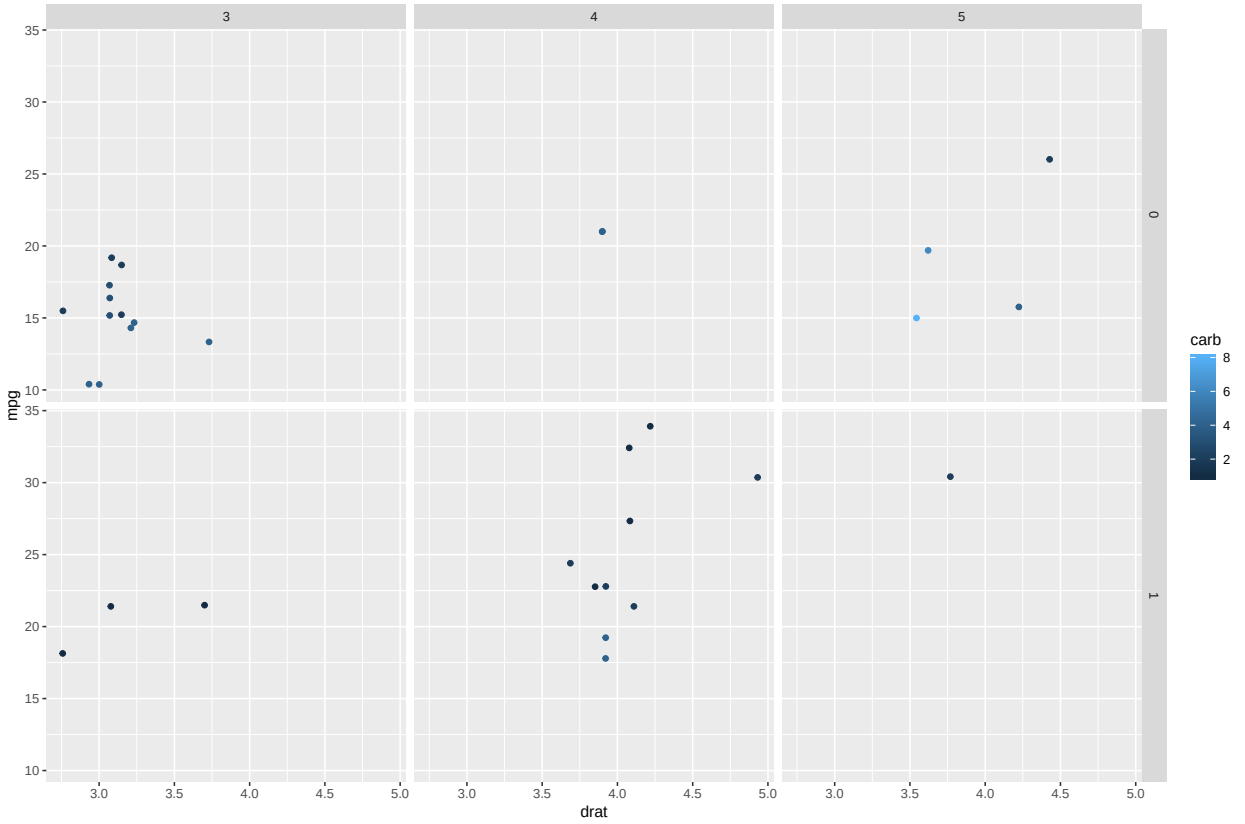
```
ggplot(data=mtcars,aes(y=mpg,x=drat,color=carb)) + geom_jitter()
```



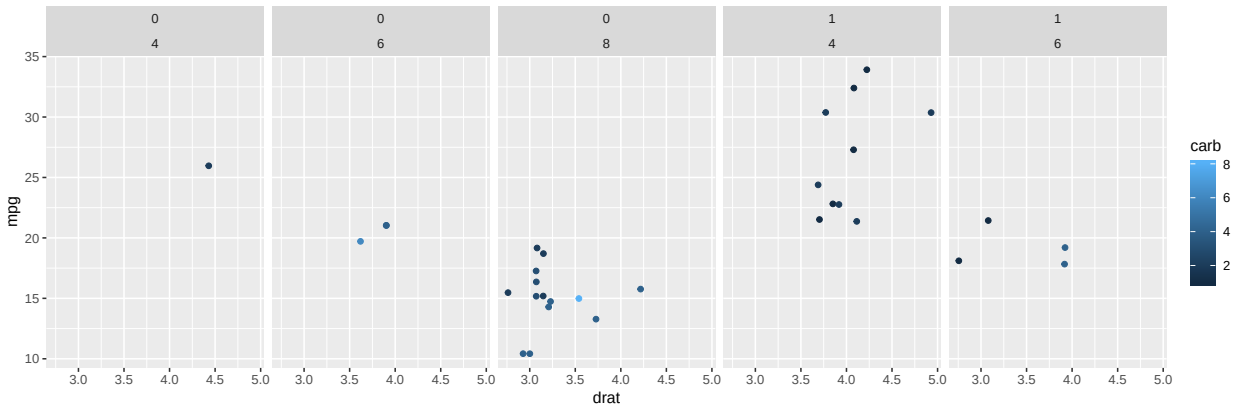
- `facet_grid( )` requires a row and/or column specification
  - separate rows and columns by `~`, use `.` if none
  - using `+` allows for multiple row and/or column specification

```
ggplot(data=mtcars,aes(y=mpg,x=drat,color=carb)) + geom_jitter() + facet_grid(vs~gear)
```


wilfried.cools@vub.be




```
ggplot(data=mtcars,aes(y=mpg,x=drat,color=carb)) + geom_jitter() + facet_grid(.~vs+cyl)
```



theme( )

Control not related to data is specified with a theme.

- themes are layers to (re-)specify theme elements
- elements include self explanatory (see helpfile `?theme`):
  - line, rect, text, title
  - axis: axis.title, axis.text.x.top, axis.ticks.x.bottom, …
  - legend: legend.spacing.y, legend.key.width, legend.justification, …
  - panel: panel.background, panel.grid.major, panel.ontop, …

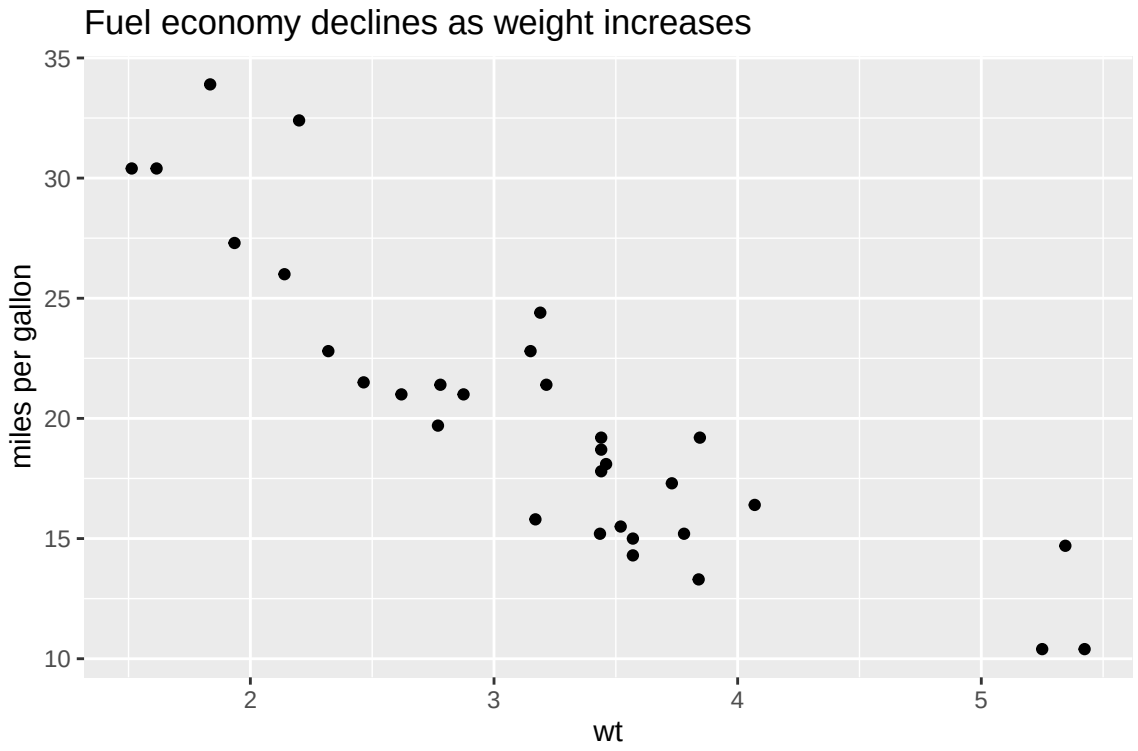
wilfried.cools@vub.be

  - plot: plot.background, plot.caption, plot.tag, plot.margin, …
  - strip: strip.background, strip.text, strip.switch.pad.wrap, …
- elements are controlled with a theme function, eg., `element_text( )`
  - `element_text( )`, `element_line( )`, `element_rect( )`, `margin( )`, …
- default themes exist, eg., `theme_minimal( )` and new can be created
- `labs( )` is a simpler way to specify titles and labels

Examples will make it more clear.

- a title and axes labels are included

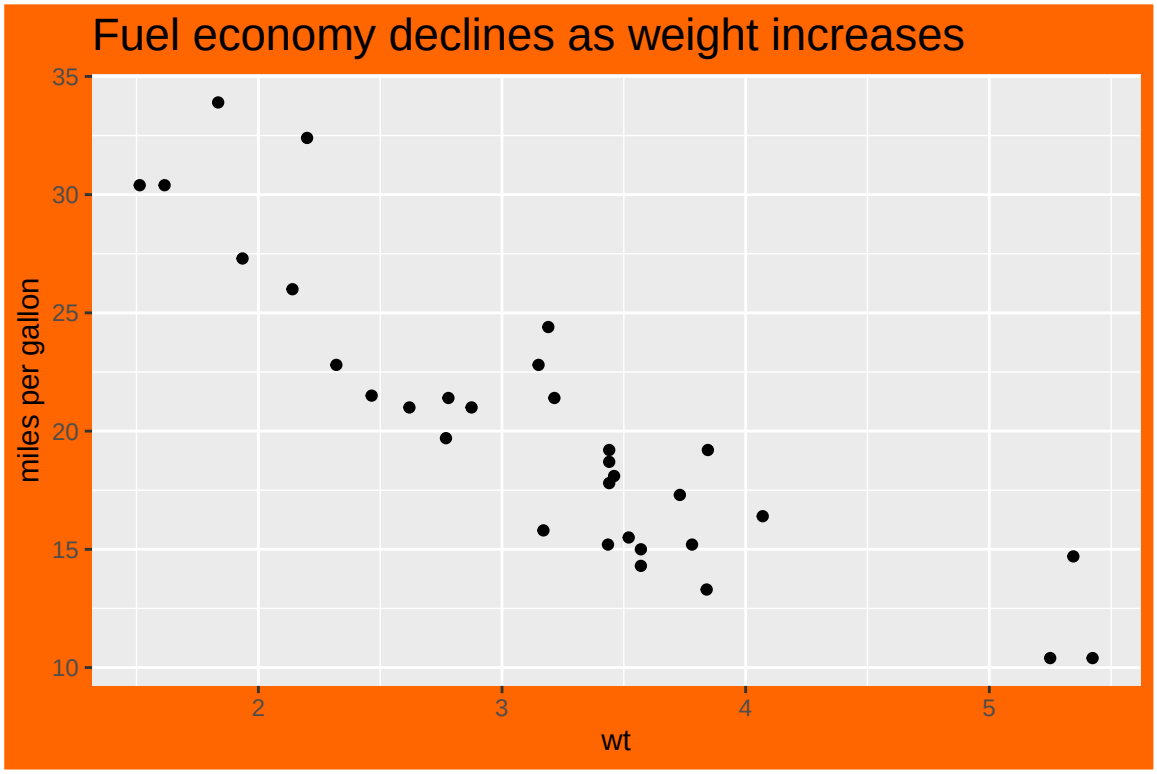
```
(px <- ggplot(mtcars, aes(wt, mpg)) + geom_point() +
    labs(title = "Fuel economy declines as weight increases",y='miles per gallon'))
```



- the size of the plot title is changed with the `element_text( )`, the background with the `element_rect( )`

```
px + theme(plot.title = element_text(size=rel(1.5)),
    plot.background=element_rect(fill="#FF6600"))
```

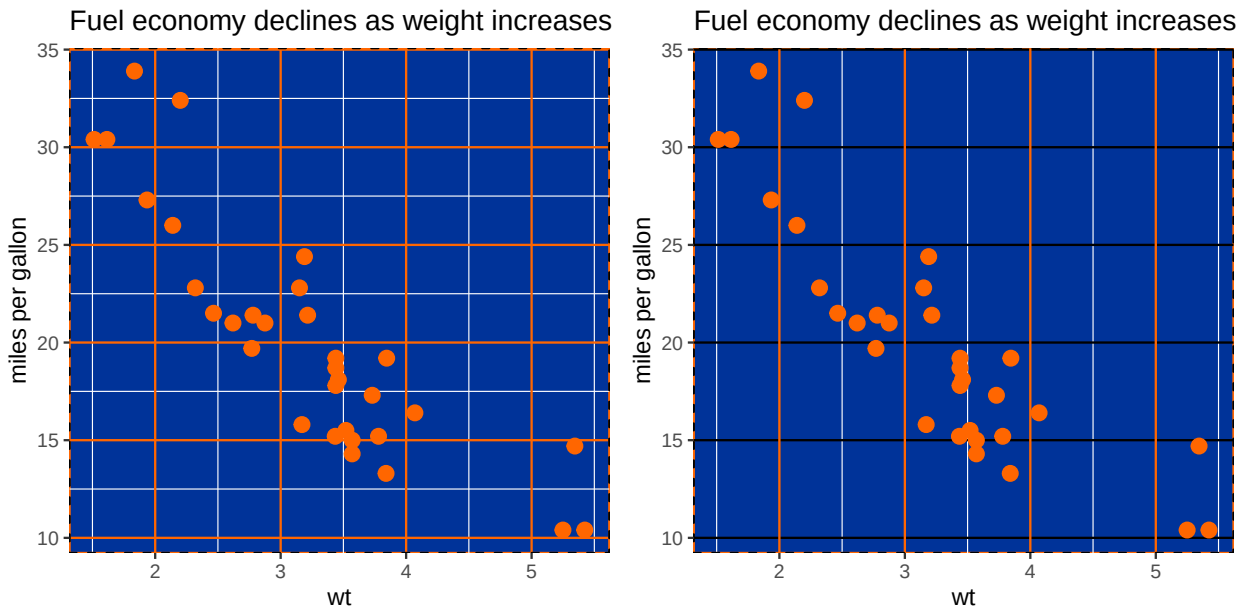

wilfried.cools@vub.be



- the inside of the plot is the panel, modifiable too.
  - multiple elements can be specified in one theme
  - multiple themes can be specified by layers

```
p1 <- px + geom_point(color='#FF6600',size=3) +
    theme(panel.background = element_rect(fill = "#003399", colour = "#FF6600")) +
    theme(panel.border = element_rect(linetype = "dashed", fill = NA),
        panel.grid.major = element_line(colour = "#FF6600"))
p2 <- p1 + theme(
  panel.grid.major.y = element_line(colour = "black"),
  panel.grid.minor.y = element_blank()
)
grid.arrange(p1,p2,ncol=2)
```

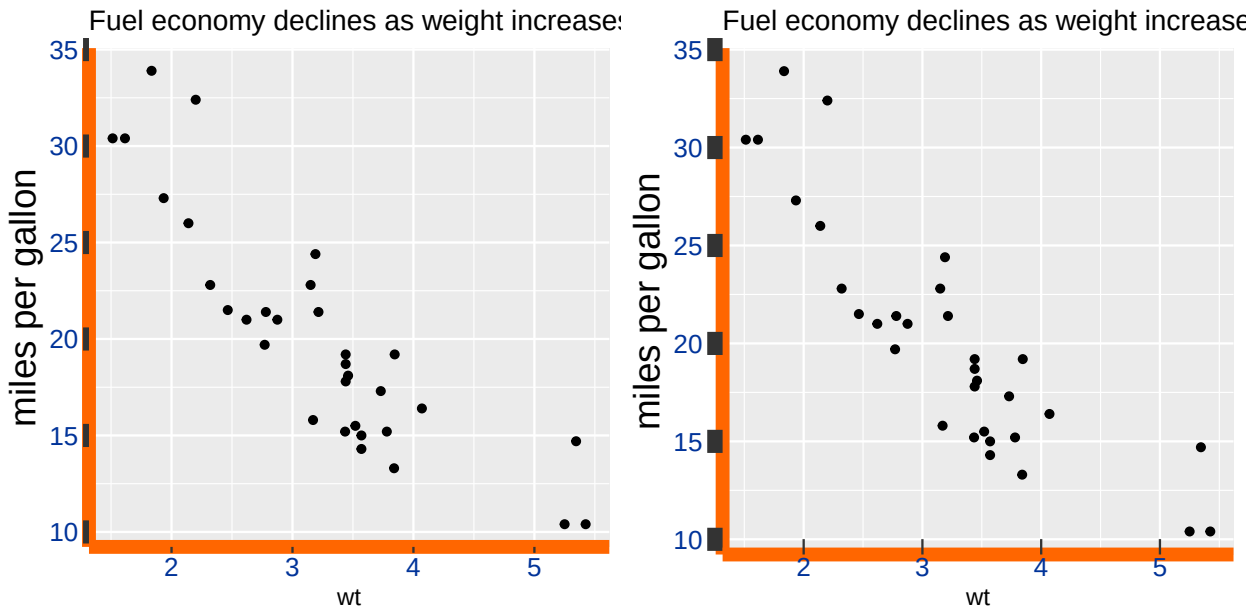

wilfried.cools@vub.be



- on the outside are the axes and titles, modifiable too.
  - the text, ticks and titles are adjusted for color and size
  - function `element_text( )`, `element_line( )` and `unit( )` are used

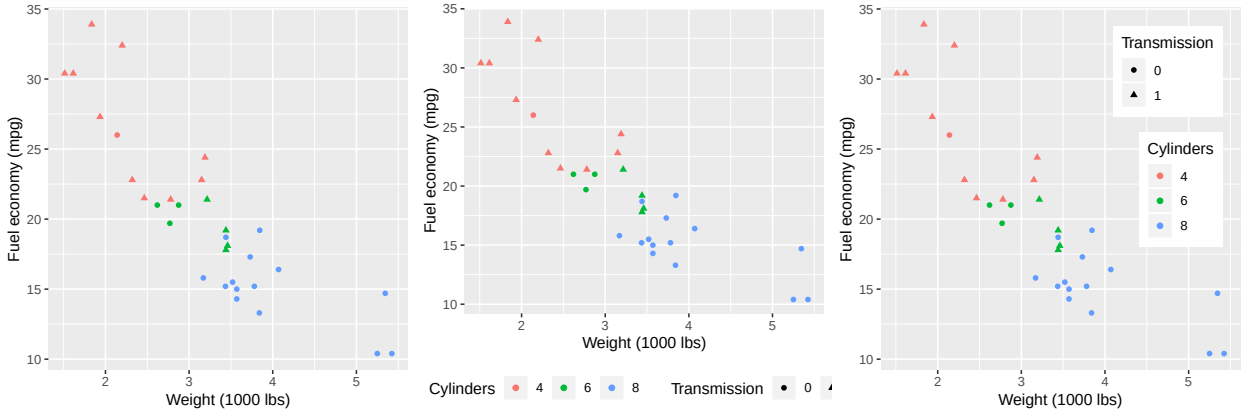
```
p1 <- px + theme(axis.line = element_line(size = 3, colour = "#FF6600")) +
    theme(axis.text = element_text(colour = "#003399", size=12)) +
    theme(axis.ticks.y = element_line(size = 5)) +
    theme(axis.title.y = element_text(size = rel(1.5)))
p2 <- p1 + theme(
  axis.ticks.length.y = unit(.25, "cm"),
  axis.ticks.length.x = unit(-.25, "cm"),
  axis.text.x = element_text(margin = margin(t = .3, unit = "cm"))
)
grid.arrange(p1,p2,ncol=2)
```


wilfried.cools@vub.be



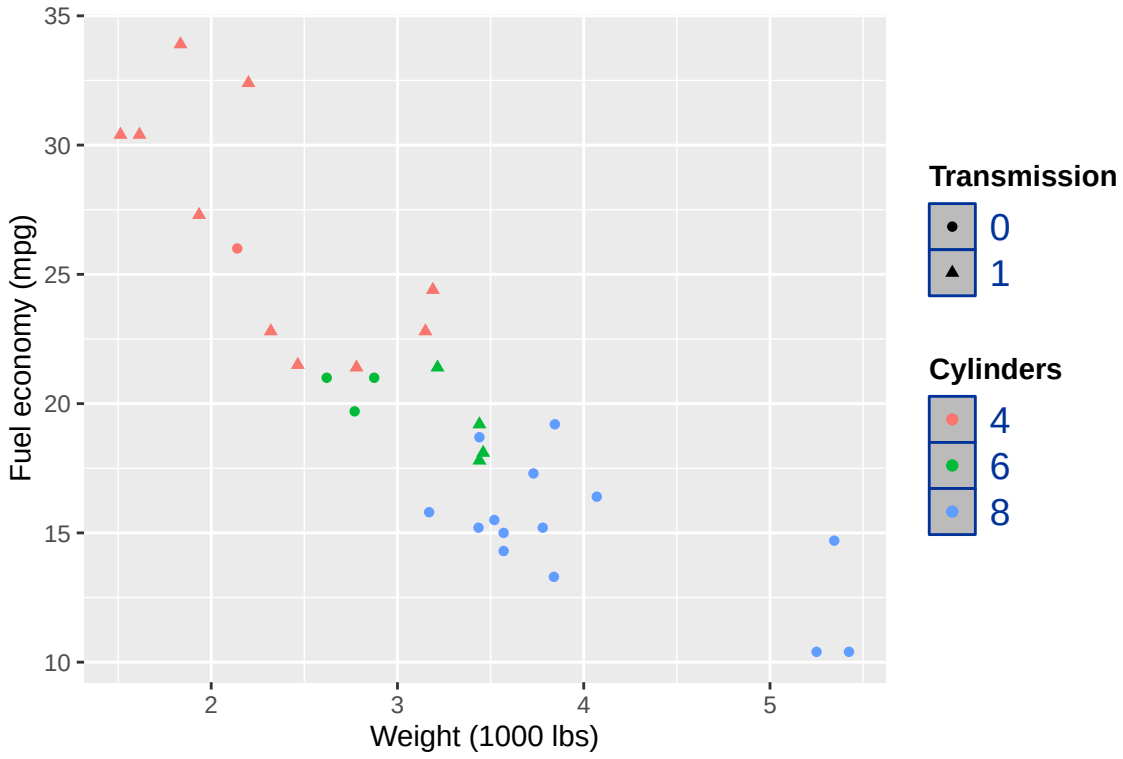
- scales are represented by legends, modifiable too
  - labs can be used to change multiple legend titles
  - legends can be positioned and formatted

```
px <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point(aes(colour = factor(cyl), shape = factor(vs))) +
  labs(
    x = "Weight (1000 lbs)",
    y = "Fuel economy (mpg)",
    colour = "Cylinders",
    shape = "Transmission"
   )
p1 <- px + theme(legend.position='none')
p2 <- px + theme(legend.justification = "right",legend.position = "bottom")
p3 <- px + theme(
  legend.position = c(.95, .95),
  legend.justification = c("right", "top"),
  legend.box.just = "right",
  legend.margin = margin(6, 6, 6, 6)
)
grid.arrange(p1,p2,p3,ncol=3)
```



- keys inside legends are modifiable too
  - labs can be used to change multiple legend titles
  - legends can be positioned and formatted, for key, text and title

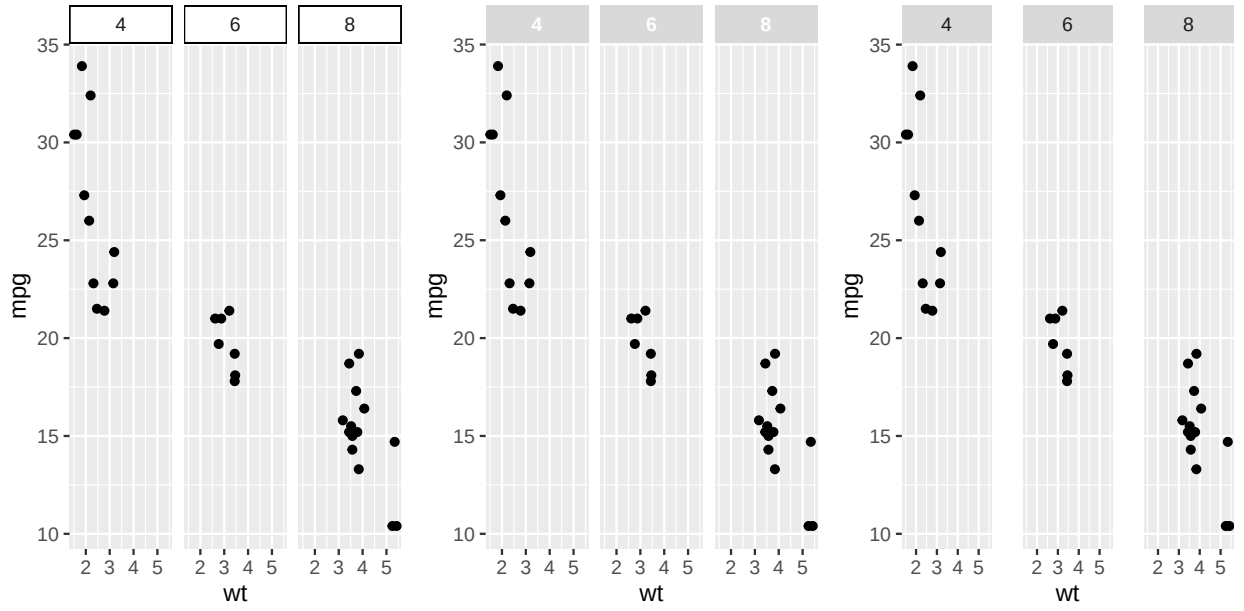
```
px + theme(legend.key = element_rect(fill = "#bbbbbb", colour = "#003399")) +
    theme(legend.text = element_text(size = 14, colour = "#003399")) +
    theme(legend.title = element_text(face = "bold"))
```



- themes also work on facets, at which strips are defined

```
px <- ggplot(mtcars, aes(wt, mpg)) + geom_point() + facet_wrap(~ cyl)
p1 <- px + theme(strip.background = element_rect(colour = "black", fill = "white"))
p2 <- px  + theme(strip.text.x = element_text(colour = "white", face = "bold"))
p3 <- px + theme(panel.spacing = unit(1, "lines"))
grid.arrange(p1,p2,p3,ncol=3)
```


wilfried.cools@vub.be

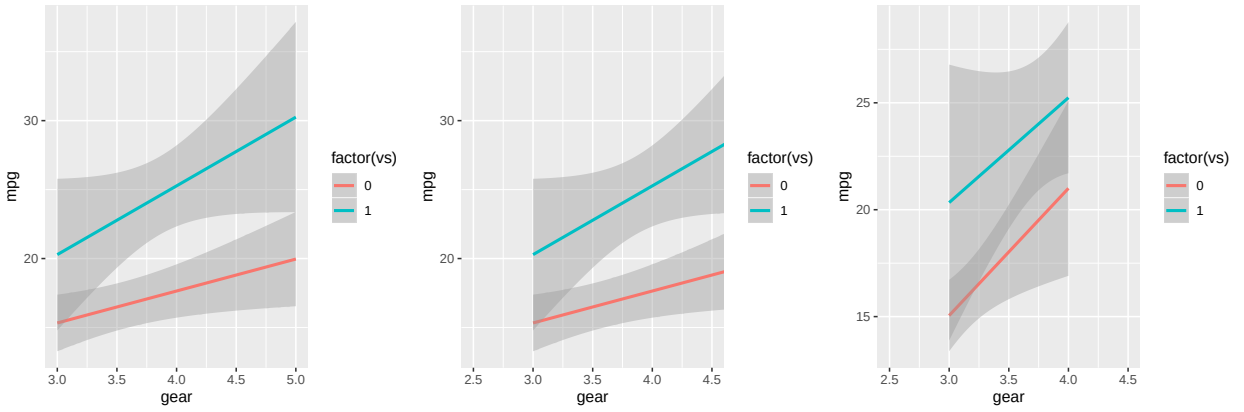


coord_*( )

Typically the default cartesian coordinate system is used, `coord_cartesian( )`.

- limits of the axes are best specified within the `coord_*( )` function
  - works like a zoom
  - alternatively, use `xlim( )` and `ylim( )`
    - beware that values outside the boundary are treated as missing

```
px <- ggplot(data=mtcars,aes(y=mpg,x=gear,color=factor(vs),group=vs))
p1 <- px + geom_smooth(method='lm')
p2 <- px + geom_smooth(method='lm') + coord_cartesian(xlim=c(2.5,4.5))
p3 <- px + geom_smooth(method='lm') + xlim(2.5,4.5)
grid.arrange(p1,p2,p3,ncol=3)
```

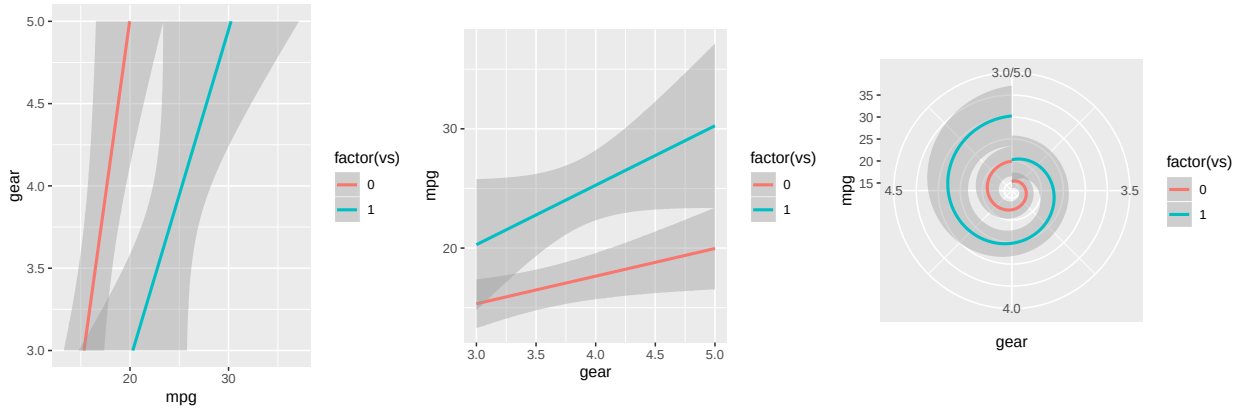


- within the cartesian family alternatives exist
  - `coord_flip( )` switches x and y-axis
  - `coord_fixed( )` sets the ratio for x and y values
    - eg., .1 means 1 unit on x is 10 on y

wilfried.cools@vub.be

- alternatives exist, eg., `coord_polar( )`, `coord_trans( )`, and various map related functions

```
p1 <- px + geom_smooth(method='lm') + coord_flip()
p2 <- px + geom_smooth(method='lm') + coord_fixed(.1)
p3 <- px + geom_smooth(method='lm') + coord_polar()
grid.arrange(p1,p2,p3,ncol=3)
```



conclusion

- ggplot is required to create a ggplot object
- aesthetics link data to scales, and group them
- geom functions turn the ggplot object into a visualization
- stat functions can be an alternative to geom functions, especially when transformations are non-standard
- each scale when visualized (whether x-y axis, color, shape, …) can be fine-tuned (data dependent)
- each scale can be used to create separate plots (faceting)
- a coordinate system can be changed or used for zooming
- a plot can be fine-tuned with a theme (data independent)

Examples to go into detail

Through various examples the cheat sheet is focused upon. https://rstudio.com/resources/cheatsheets/

primitives

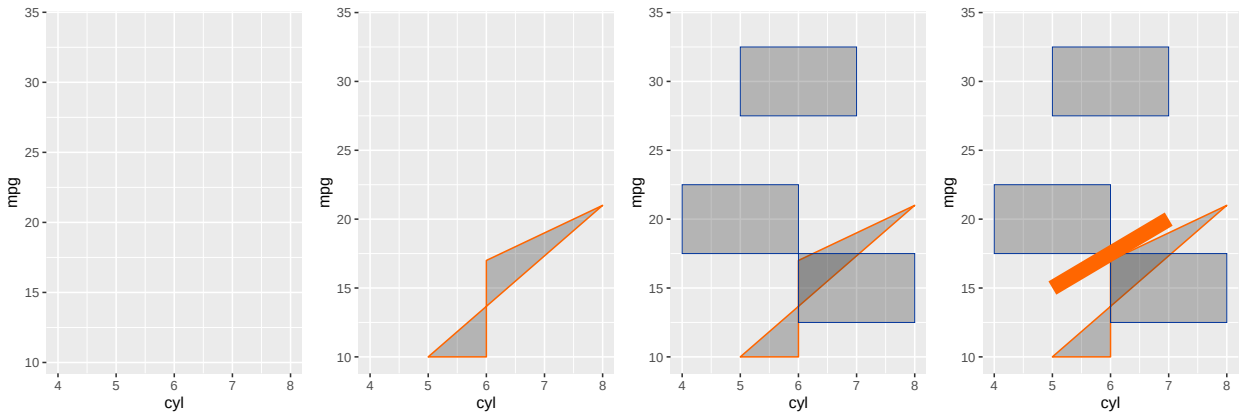
Primitives are the basic building blocks.

- several primitives exist: point( ), path( ), polygon( ), segment( ), ribbon( ), rect( ), text( ), blank( )
- only `geom_point( )` is used very often
- `geom_ribbon( )` can be interesting for showing an intervals
- most other primitives are especially useful to fine-tune a plot

It is possible to draw a blank plot, with the `mtcars` data to determine the limits. On that plot, using a different data frame for the coordinates, a polygon can be drawn, and finally using another data frame for other coordinates tiles can be included.

```
p1 <- ggplot(mtcars,aes(y=mpg,x=cyl)) + geom_blank()
tmp <- tribble(~x,~y,
```

```
6,17,
6,10,
5,10,
8,21)
tmp2 <- tribble(~x,~y,~w,
5,20,2,
6,30,5,
7,15,2)
p2 <- p1 + geom_polygon(data=tmp,aes(x=x,y=y),alpha=.3,color="#FF6600")
p3 <- p2 + geom_tile(data=tmp2,aes(x=x,y=y,width=2),alpha=.3,color="#003399")
p4 <- p3 + geom_segment(data=data.frame(x=5,y=15,xend=7,yend=20),aes(x=x,y=y,xend=xend,yend=yend),size=
grid.arrange(p1,p2,p3,p4,ncol=4)
```

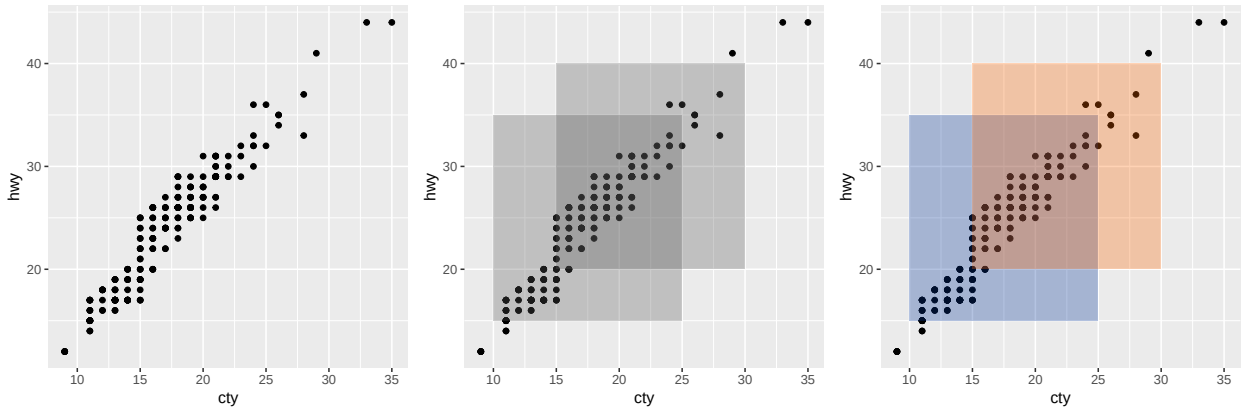


exercises on primitives

- load in the `mpg` dataset, part of the ggplot2 package, and have a look at it
- make a scatterplot for `cty` per `hwy`
- draw 2 rectangles (difficult for first timers!), one with corners 10,15 and 20,25, and one shifted with 5 both dimensions
  - create a dataframe or tibble with values for the corners, eg., xi, yi, xa, ya (min & max for x & y)

```
tmp <- data.frame(c1=c(10,15),c2=c(15,20),c3=c(25,30),c4=c(35,40))
tmp <- tribble(~xi,~yi,~xa,~ya,
10,15,25,35,
15,20,30,40
)
```

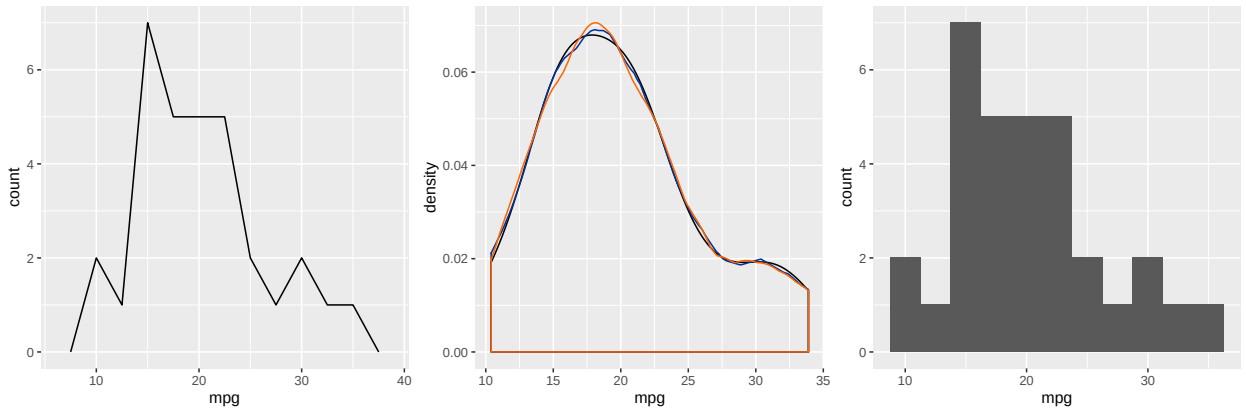
```
- plot them on top of the scatterplot (the argument inherit.aes=FALSE will avoid inheriting the aesthet
- add color to each separately, use '#FF6600' and '#003399'
```



one-variable

Various visualizations address one particular variable, mostly continuous but possibly also discrete.

```
p1 <- ggplot(data=mtcars,aes(mpg)) + geom_freqpoly(binwidth=2.5)
p2 <- ggplot(data=mtcars,aes(mpg)) + geom_density() +
    geom_density(kernel='triangular',color="#003399") + geom_density(kernel='optcosine',color="#FF6600")
p3 <- ggplot(data=mtcars,aes(mpg)) + geom_histogram(binwidth=2.5)
grid.arrange(p1,p2,p3,ncol=3)
```



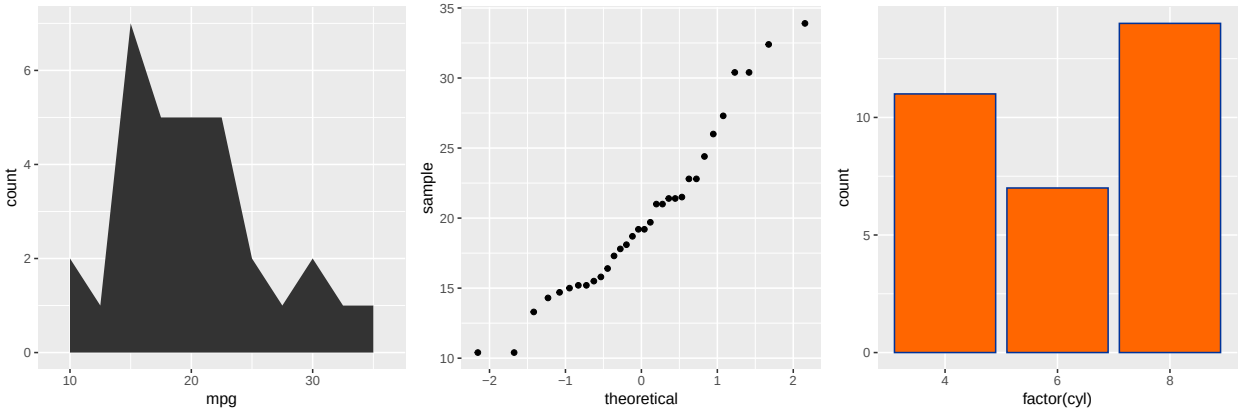
A continuous variables is typically 'binned' into groups within which frequencies are obtained.
When using the `geom_area( )` such binning must be explicitly included as stat argument.

Also note that the `geom_qq( )` requires a sample argument instead of x, as positions are determined by their size.

A final plot shows a typical discrete one variable plot, the bar-plot with `geom_bar( )`, which is similar to the histogram but which shows the actual values instead of a count per bin.

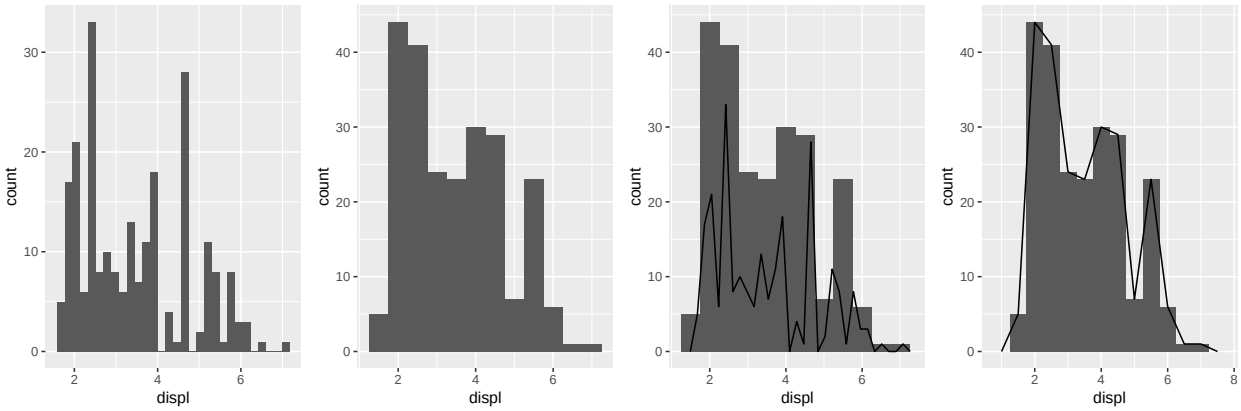
```
p1 <- ggplot(data=mtcars,aes(mpg)) + geom_area(stat='bin',binwidth=2.5)
p2 <- ggplot(data=mtcars,aes(sample=mpg)) + geom_qq()
p3 <- ggplot(data=mtcars,aes(factor(cyl))) + geom_bar(fill='#FF6600',color='#003399')
grid.arrange(p1,p2,p3,ncol=3)
```


wilfried.cools@vub.be

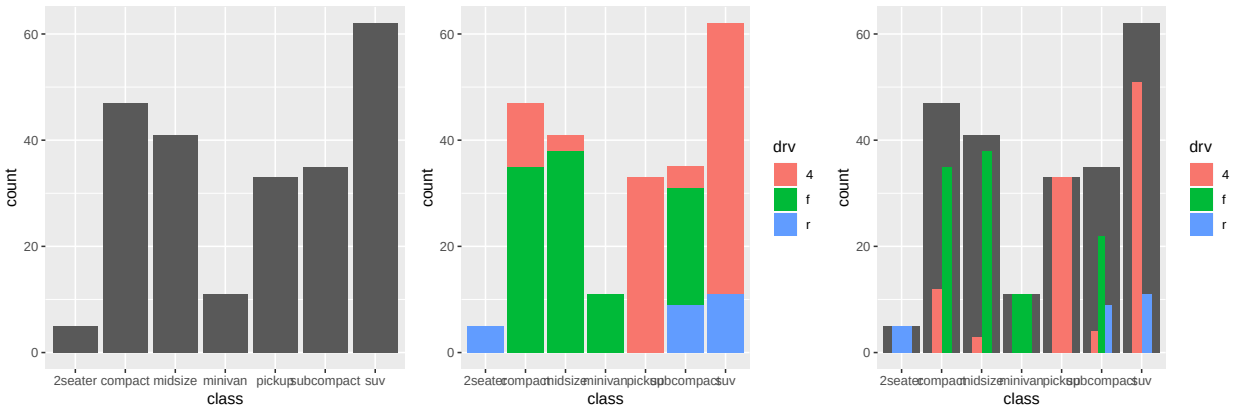


exercises on one variable visualizations

- use the `mpg` dataset
- make a histogram for the continuously scaled `displ` variable
- adjust the binwidth to .5
- add a frequency polynomial on top (freqpoly)
- notice what happens if the same binwidth is used



- make a barplot for the discretely scaled `class` variable
- group the data with colors dependent on the `drv` variable
- turn the bars next to one-another (reduce their width to .5 to increase space between bars)



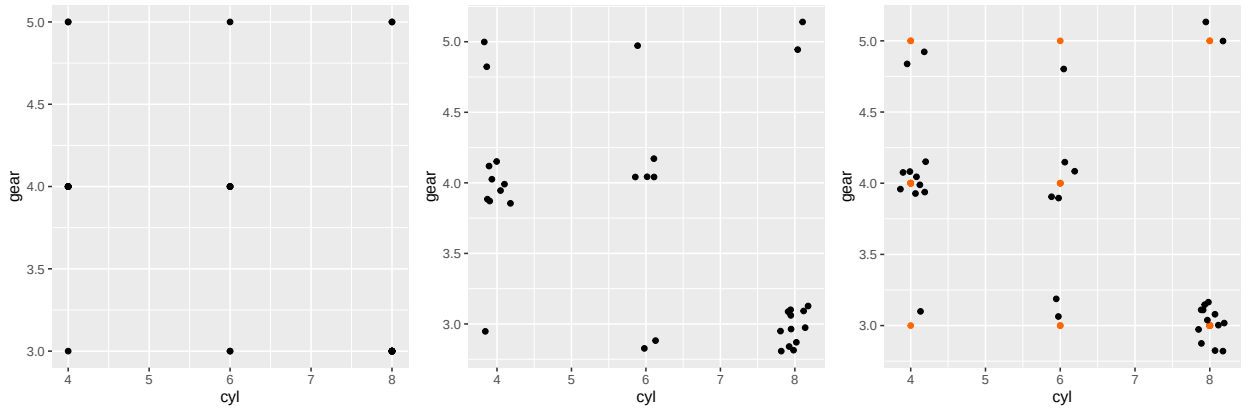
wilfried.cools@vub.be

two-variables

Various visualizations address the relation between two variables, whether discrete and/or continuous.

Especially for categorical data data could obscure other data. Deal with this using the position argument or with the `geom_jitter( )`. Avoid combining both `geom_point( )` and `geom_jitter( )` as it would draw points each time.

```
p1 <- ggplot(data=mtcars,aes(cyl,gear)) + geom_point()
p2 <- ggplot(data=mtcars,aes(cyl,gear)) + geom_jitter(width=.2,height=.2)
p3 <- ggplot(data=mtcars,aes(cyl,gear)) + geom_point(color='#FF6600') + geom_jitter(width=.2,height=.2)
grid.arrange(p1,p2,p3,ncol=3)
```

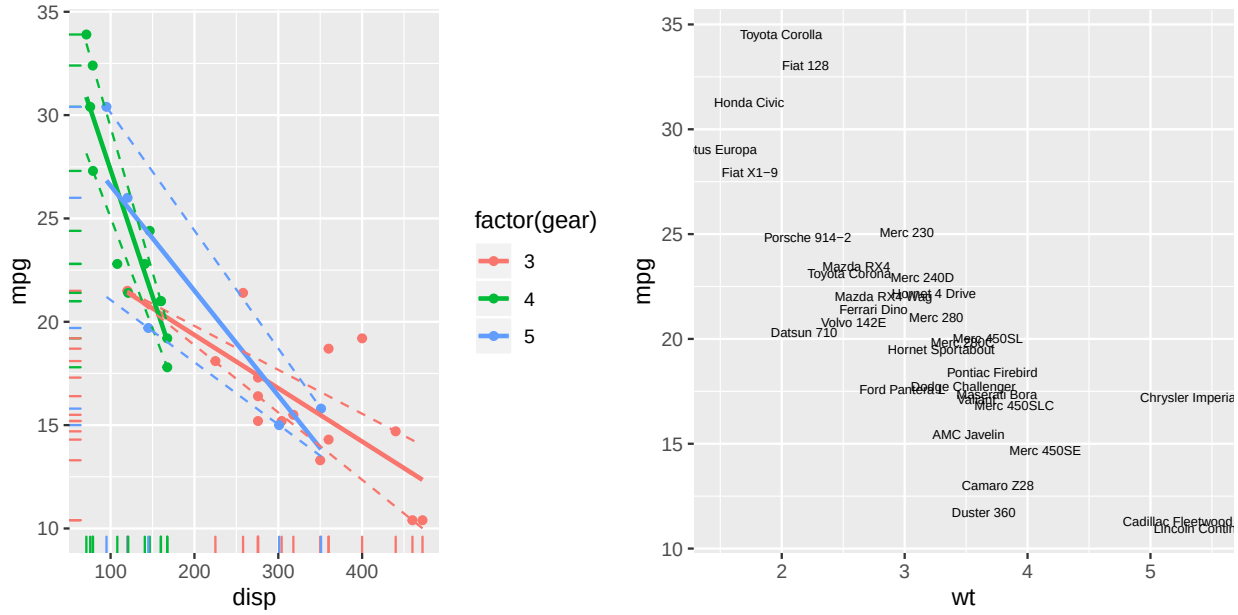


The smooth function has been shown above, changing the confidence band into areas that capture that middle 50% (quantiles .25 and .75), and including a rug at the axes to capture the one dimensional distribution.

Instead of bullet indicators, the row names or any other set of labels can be used using the label argument (within the `aes( )` when related to data). A bit of jitter is added to avoid overlap.

```
p1 <- ggplot(data=mtcars, aes(y=mpg,x=disp,color=factor(gear))) + geom_point() + geom_smooth(method='lm
p2 <- ggplot(mtcars, aes(wt, mpg)) + geom_text(aes(label=(rownames(mtcars))),size=2,position=position_j
grid.arrange(p1,p2,ncol=2)
```


wilfried.cools@vub.be



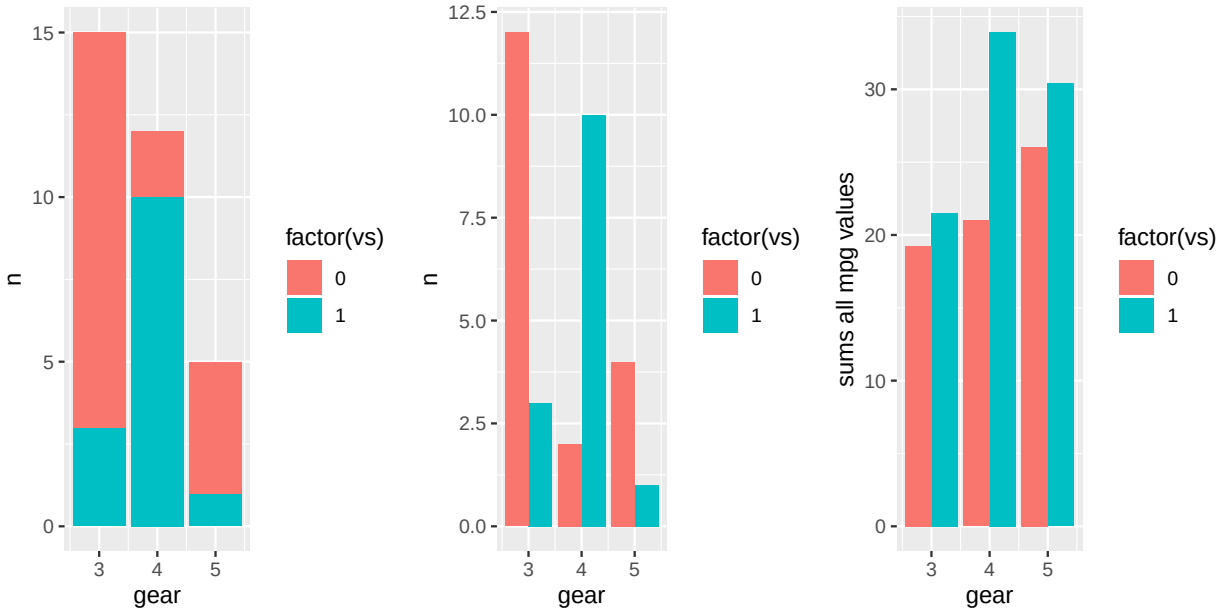
Like with one dimensional discrete data, bars can be obtained with `geom_col( )`, or with `geom_bar(stat='identity')`. Use of 'identity' causes the height to depend on the numbers in the data. It sums these numbers if not unique, be careful.

```
(tmp <- mtcars %>% group_by(gear,vs) %>% count())
```

```
| # A tibble: 6 x 3
| # Groups:   gear, vs [6]
|    gear    vs     n
|   <dbl> <dbl> <int>
| 1     3     0    12
| 2     3     1     3
| 3     4     0     2
| 4     4     1    10
| 5     5     0     4
| 6     5     1     1
```

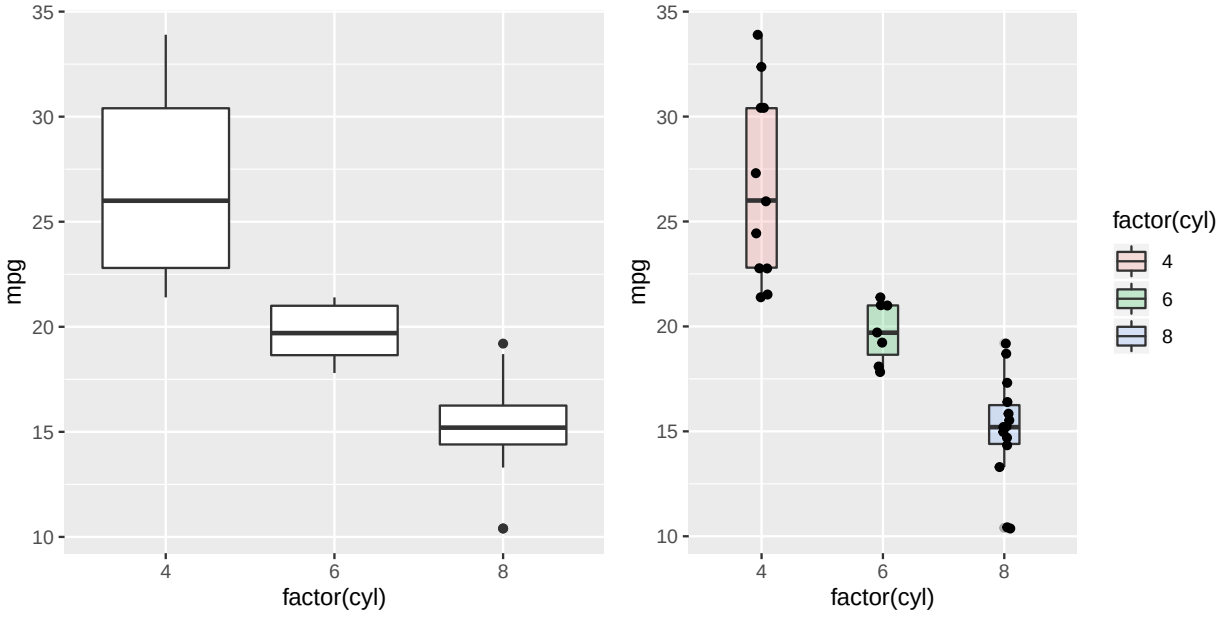
```
p1 <- ggplot(tmp, aes(fill=factor(vs), y=n, x=gear)) + geom_bar(position="stack", stat="identity")
p2 <- ggplot(tmp, aes(fill=factor(vs), y=n, x=gear)) + geom_col(position="dodge")
p3 <- ggplot(mtcars, aes(fill=factor(vs), y=mpg, x=gear)) + geom_col(position="dodge") + labs(y='sums a
grid.arrange(p1,p2,p3,ncol=3)
```


wilfried.cools@vub.be



An interesting visualization for continuous data, possibly for different groups, is the boxplot.

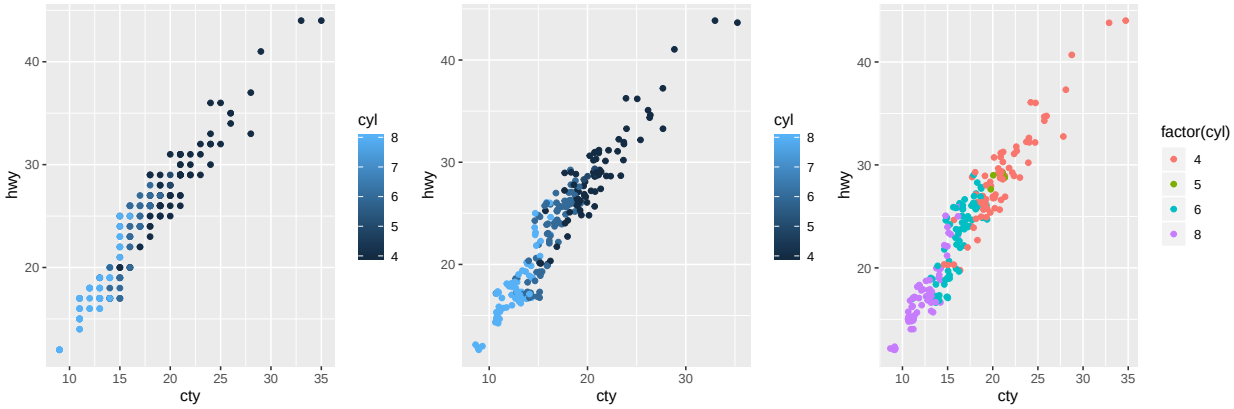
```
p1 <- ggplot(data=mtcars,aes(y=mpg,x=factor(cyl))) + geom_boxplot()
p2 <- ggplot(data=mtcars,aes(y=mpg,x=factor(cyl))) +
    geom_boxplot(width=.25,alpha=.2,aes(fill=factor(cyl))) + geom_jitter(width=.05)
grid.arrange(p1,p2,ncol=2)
```



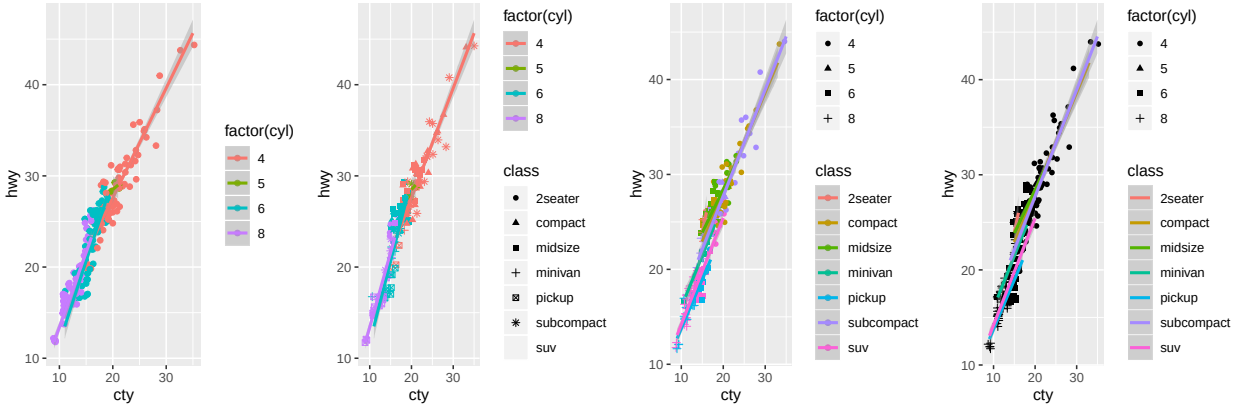
exercises on two variable visualizations

- use the `mpg` dataset
- make a scatterplot histogram for the continuously scaled `hwy` on `cty`, and color by `cyl`
- jitter the data, so it shows if data points obscure one-another
- make sure that `cyl` is categorical

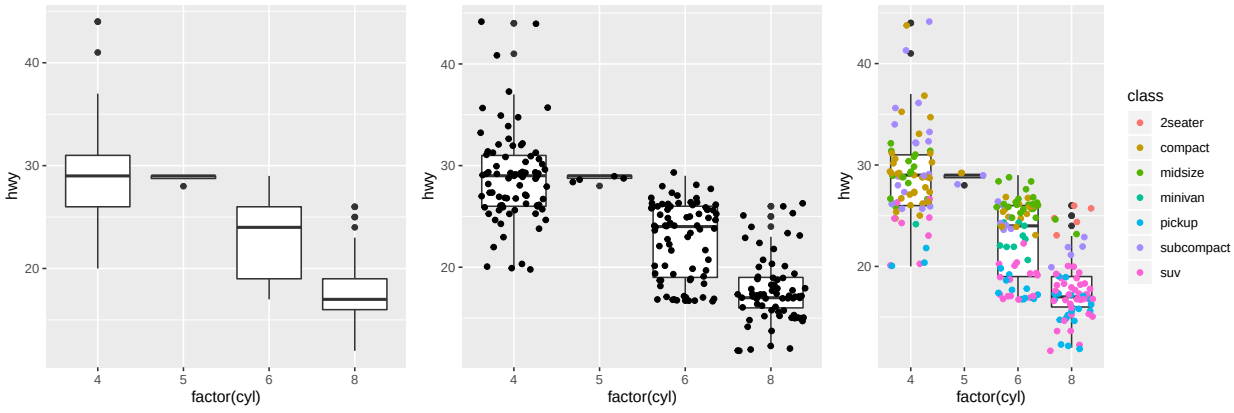
wilfried.cools@vub.be



- add a conditional average with `geom_smooth( )`, use the `lm` method
- add a shape dependent on `class`, notice the restriction on the number of shapes
- switch the color and shapes around (color can have more than 6 if it is really necessary)
- make sure the symbols do not differ by color (all black), only shape, but keep the regression lines

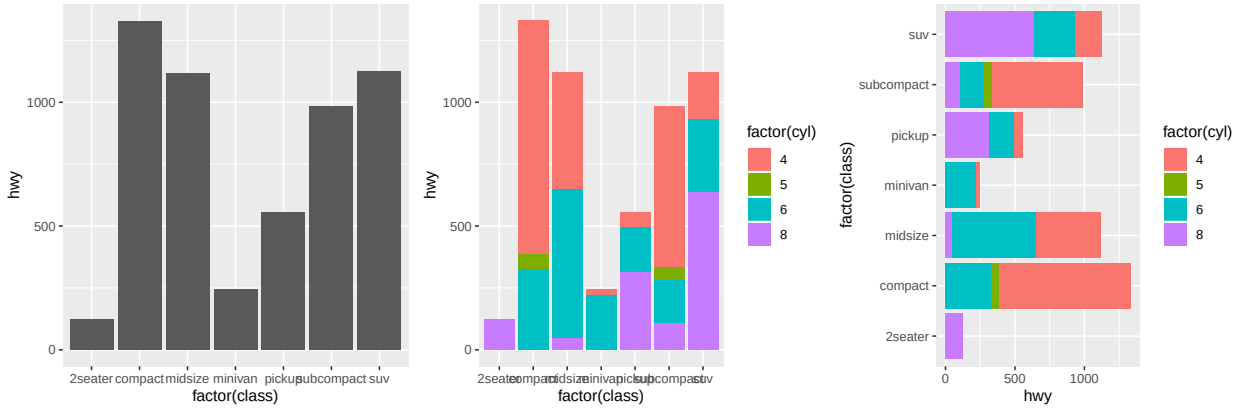


- show boxplots for the `hwy` for each `cyl`
- add the actual data points and make sure they do not obscure each other
- color the observations dependent on `class`



- make a bar chart that add all `hwy` values in each `class` group
- use a coloring of the bars to signal the relative contribution of all `cyl` categories

wilfried.cools@vub.be



intervals

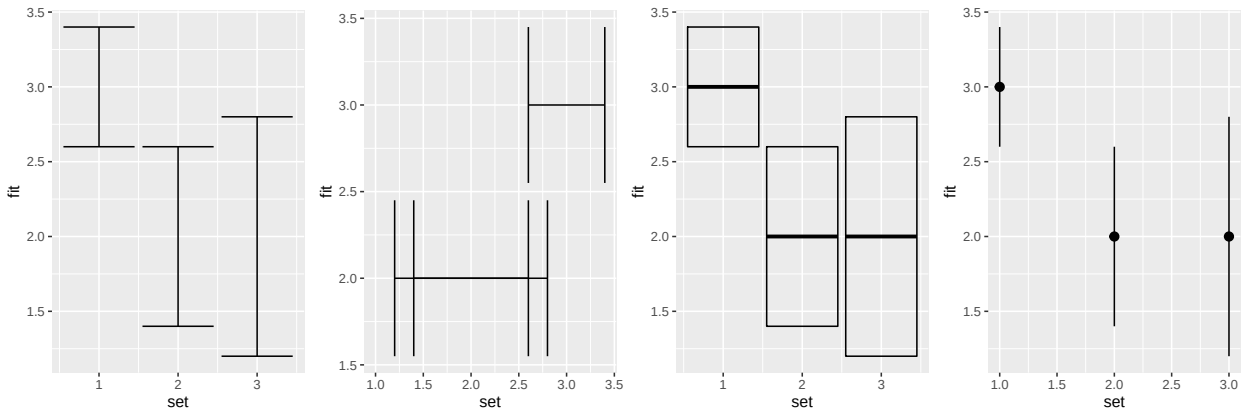
Specialized functions facilitate visualization of errors.

When standard errors are obtained, along with fitted values, they are easily shown. Other intervals can be visualized this way too.

```
(tmp <- tribble(~set,~fit,~se,1,3,.2,2,2,.3,3,2,.4))
```

```
| # A tibble: 3 x 3
|     set   fit    se
|   <dbl> <dbl> <dbl>
| 1     1     3   0.2
| 2     2     2   0.3
| 3     3     2   0.4
```

```
px <- ggplot(data=tmp,aes(y=fit,x=set))
p1 <- px + geom_errorbar(aes(ymax=fit+2*se,ymin=fit-2*se))
p2 <- px + geom_errorbarh(aes(xmax=fit+2*se,xmin=fit-2*se))
p3 <- px + geom_crossbar(aes(ymax=fit+2*se,ymin=fit-2*se))
p4 <- px + geom_pointrange(aes(ymax=fit+2*se,ymin=fit-2*se))
grid.arrange(p1,p2,p3,p4,ncol=4)
```

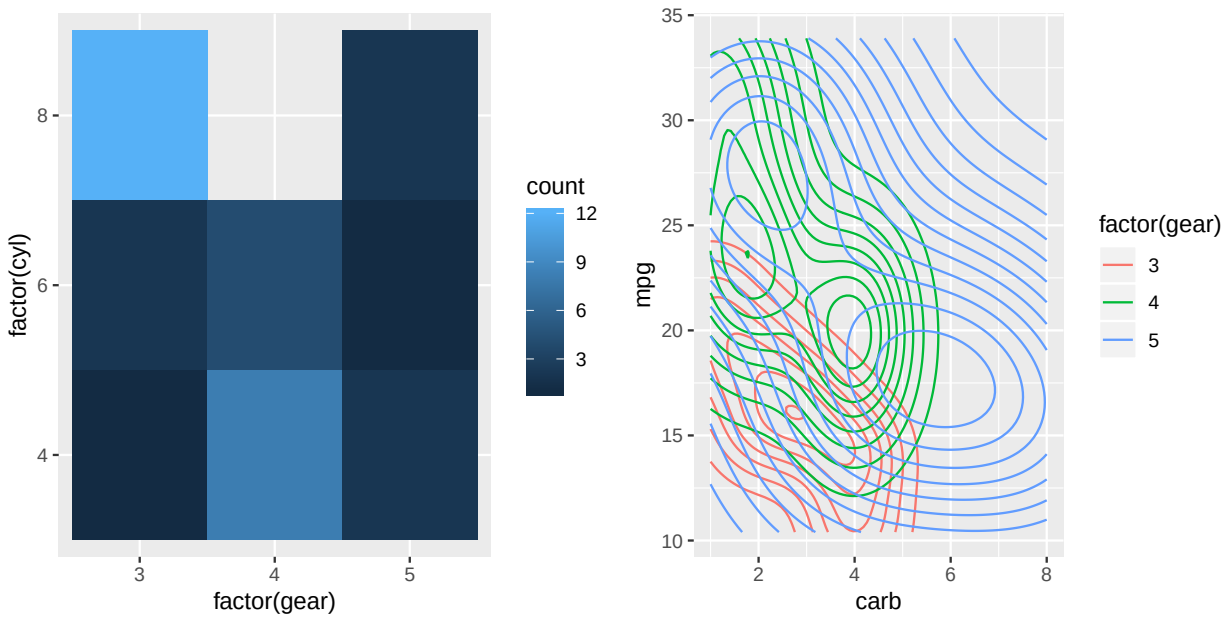


third variable implied

Frequencies and densities can be obtained for two variables.

With `geom_bin2d( )` the frequency of combinations is obtained, in this case for the factors cyl and gear. Because using bins also continuous variables can be used. The `geom_density2d( )` is also used with continuous variables, with contours depending on for example the gear levels.

```
p1 <- ggplot(data=mtcars,aes(y=factor(cyl),x=factor(gear))) + geom_bin2d()
p2 <- ggplot(data=mtcars,aes(y=mpg,x=carb)) + geom_density2d(aes(colour = factor(gear)))
grid.arrange(p1,p2,ncol=2)
```



three variables

While typically a third variable is included using aesthetics, it can be done with the z dimension as well.

When a third dimension is linked to a combination of conditions, they are easily shown. For the example meaningless data are simulated, 100 observations of which the first 6 are shown.

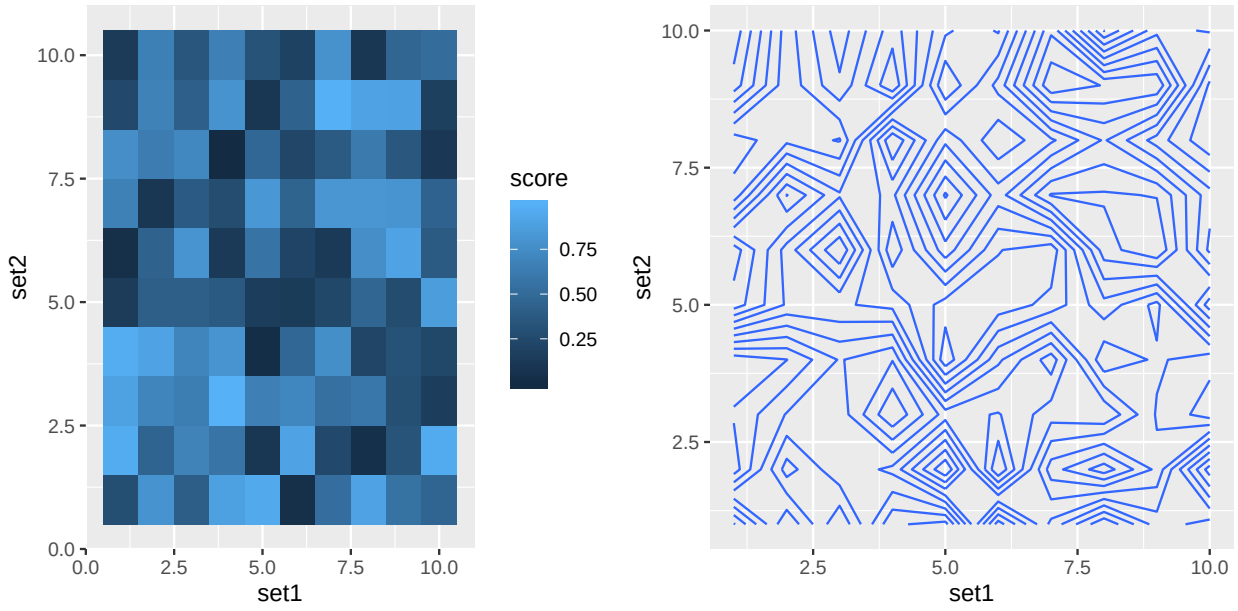
```
tmp <- expand.grid(set1=1:10,set2=1:10); set.seed(123); tmp$score <- runif(100,0,1)
head(tmp)
```

```
|   set1 set2     score
| 1    1    1 0.2875775
| 2    2    1 0.7883051
| 3    3    1 0.4089769
| 4    4    1 0.8830174
| 5    5    1 0.9404673
| 6    6    1 0.0455565
```

A heatmap is the most obvious use, to show for example correlations between many variables with colors instead of values. A small example is used instead. Notice that the argument for tile is a fill while for contour is is simply z.

```
p1 <- ggplot(data=tmp,aes(y=set2,x=set1)) + geom_tile(aes(fill=score))
p2 <- ggplot(data=tmp,aes(y=set2,x=set1)) + geom_contour(aes(z=score))
grid.arrange(p1,p2,ncol=2)
```


wilfried.cools@vub.be



Conclusion

ggplot2 is an excellent package, read more about it

Wickham, H. (2016). ggplot2: Elegant graphics for data analysis (Second edition). Springer.

or at

https://ggplot2-book.org/

to learn… just use it and keep on trying when you are stuck, check online, lots of help out there

wilfried.cools@vub.be