Exploratory Data Analysis with R

Plotting with ggplot()

Xuemao Zhang
East Stroudsburg University

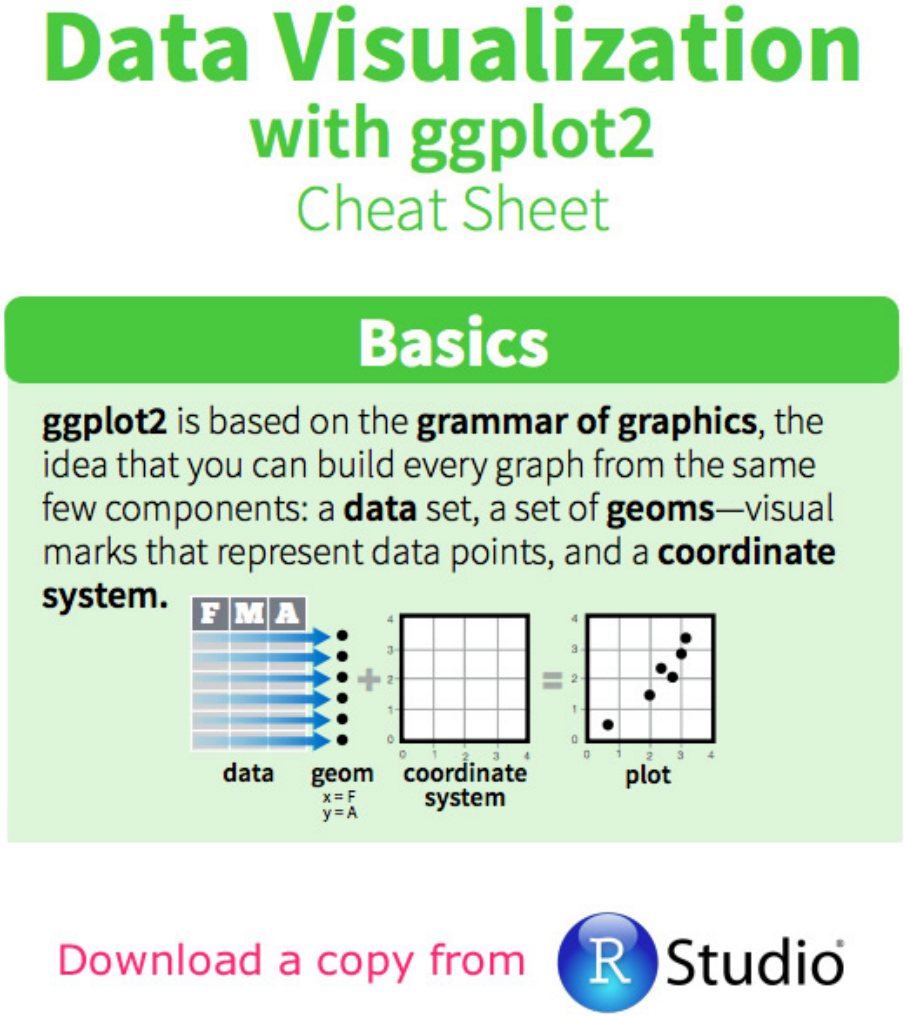
September 16, 2022

Outline

- Grammar of Graphics
  - Hadley Wickham and R:ggplot2
  - Layer-by-Layer Graphics
- The ggplot() grammar
- Graphical parameters
  - Titles
  - Legend
  - Colors
  - Points
  - Axis scales

Grammar of Graphics - ggplot2

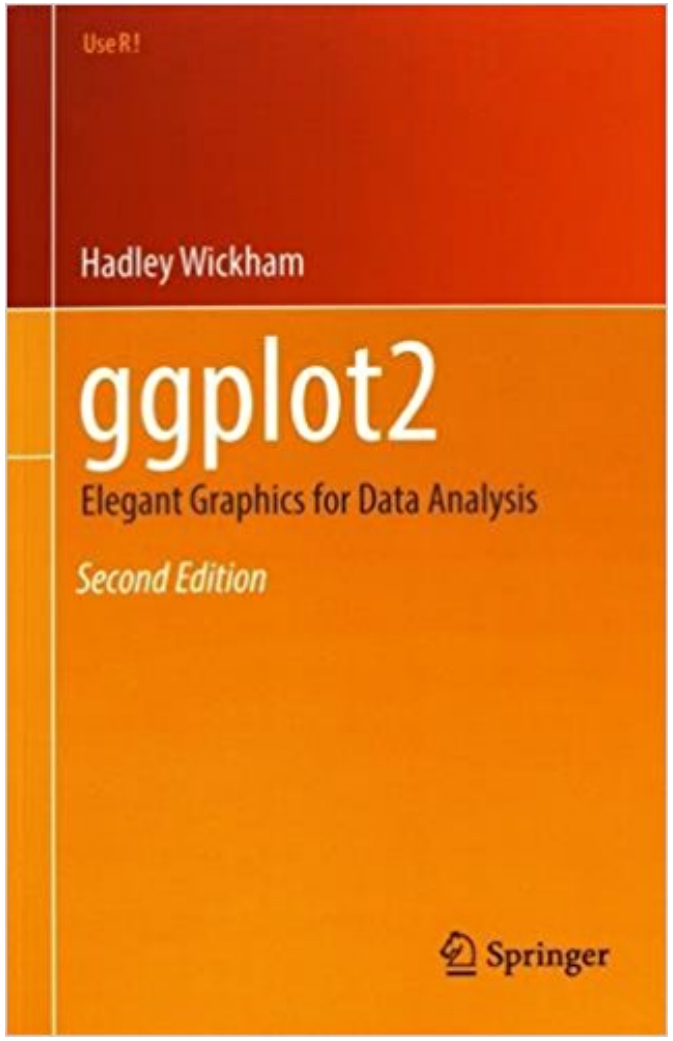
- The cheat sheet of `ggplot2` can be downloaded from Rstudio.



Grammar of Graphics - ggplot2

- Hadley Wickham is the author of R: ggplot2.
- He is the chief scientist at RStudio, Creator of popular R packages: ggplot2, dplyr, tidyr, devtools, etc; "The man who r evolutionized R" according to Pricenomics (2015)
- R graphics: base -> lattice -> ggplot2; We skip the package `lattice` in this course
  "ggplot2, started in 2005, is an attempt to take the good things about base and lattice graphics and improve on them with a strong underlying model" (Hadley Wickham)
- R:ggplot2 is one of most commonly downloaded R packages
- Based on Grammar of Graphics by Wilkinson (2005; Springer 2ed)

Grammar of Graphics - ggplot2



Grammar of Graphics - ggplot2

Quote from the ggplot2 book that further quotes Wilkinson (2005):

> *In brief, the grammar tells us that a statistical graphic is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars). The plot may also contain statistical transformations of the data and is drawn on a specific coordinate system. Facetting can be used to generate the same plot for different subsets of the dataset. It is the combination of these independent components that make up a graphic.*

Keywords: mapping, aesthetic attributes, geometric objects, statistical transformations, coordinate system, facetting

Grammar of Graphics - ggplot2

Package 'ggplot2'

May 3, 2022

Version 3.3.6

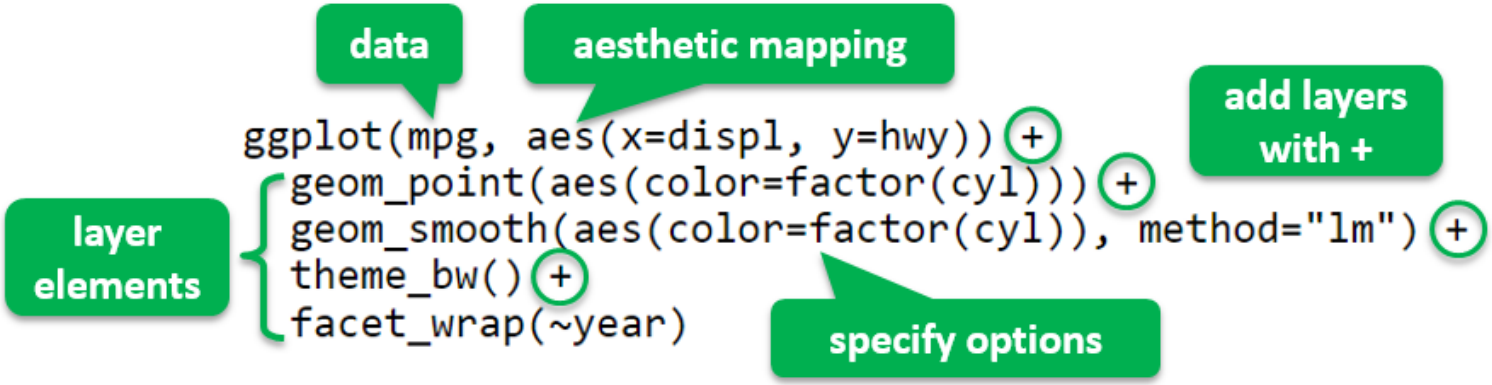
Title Create Elegant Data Visualisations Using the Grammar of Graphics

Description A system for 'declaratively' creating graphics, based on ``The Grammar of Graphics''. You provide the data, tell 'ggplot2' how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

Depends R (>= 3.3)

- The most popular package for producing static visualizations in R; New upgrade to Version 3.3.6; See CRAN for updated information
- Online documentation at https://ggplot2.tidyverse.org/reference/
- Download the useful cheatsheet created by Rstudio, Inc.
- Also available in Python: https://yhat.github.io/ggpy/

Layer-by-Layer Graphics



- The layered structure of `ggplot2` encourages you to design and contrust graphs in a structured manner.
- `ggplot2` uses the special "+" method to add layers to plots.
- **data**: Data frame that contains the variables of interest.
- **geom**: Geometric **shape** that the data are mapped to.
  - point, line, polygon, histogram, quantile, bar, ...
- **Aesthetics**: Visual properties of the **geom**, mapping data varialbes to aesthetic attributes
  - x (horizontal) position, y (vertical) position, size, shape, color, fill, ...
- **scale**: Controls how data are mapped to the visual values of the aesthetic.

Layer-by-Layer Graphics

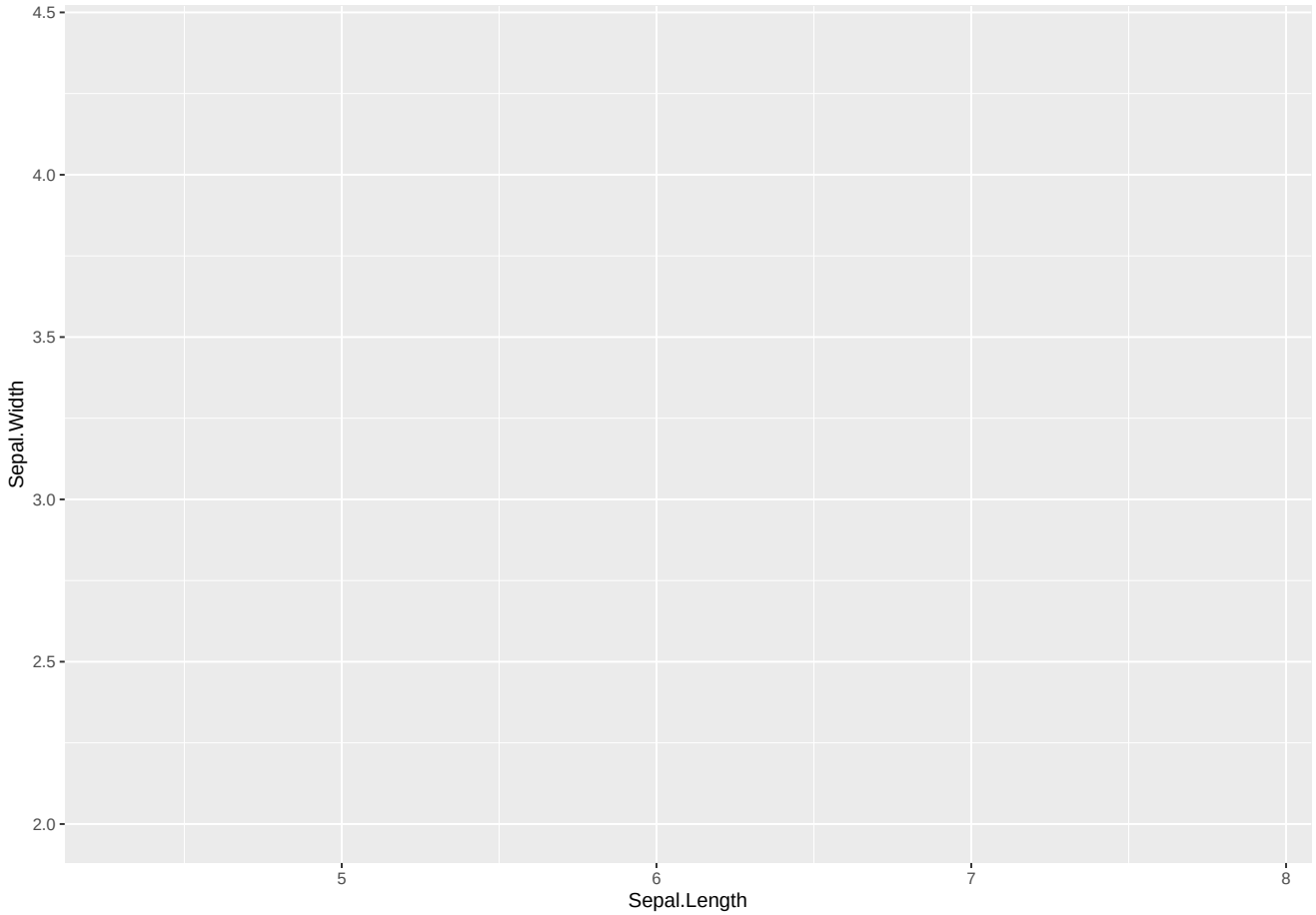
Other components for ggplots

- **Statistical transformations**: bin, boxplot, density, contour, function, ...
- **coordinate system**: cartesian, polar, map projection, ...
- **facet**: conditioning display split data in multi-panels
- **theme**: control non-data visual elements (title, axes, tick, ...)

Layer-by-Layer Graphics

- Aesthetics is mapping of variables to graphical elements. For example,

```
library(ggplot2)
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width))
```



Layer-by-Layer Graphics

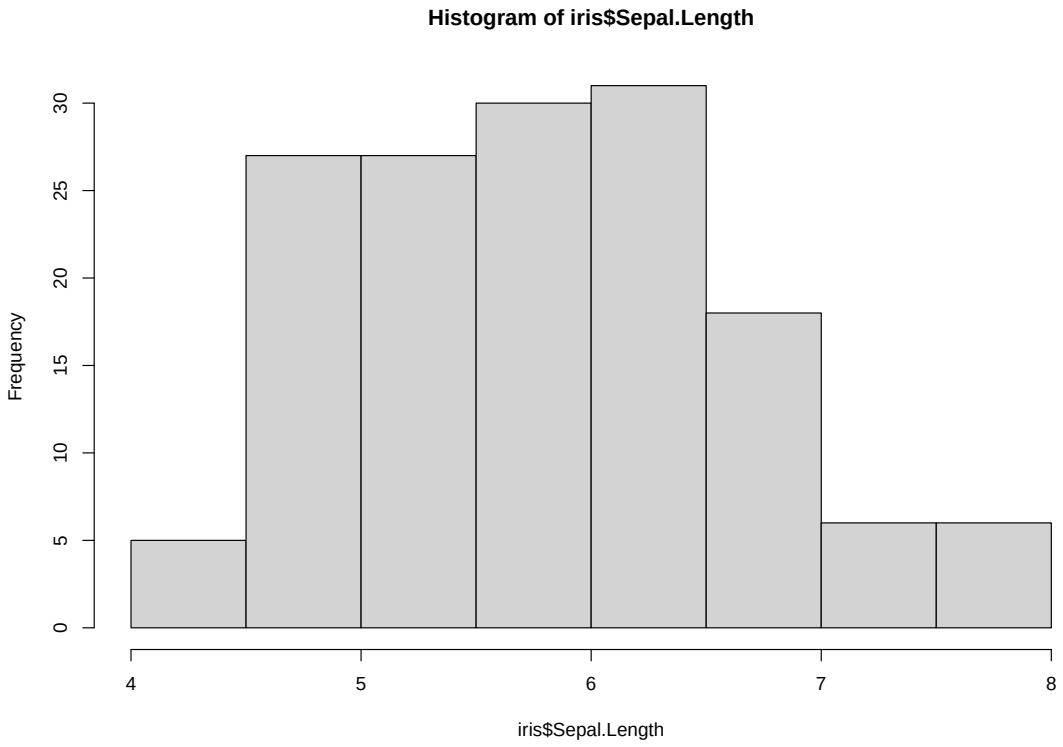
- In ggplot2, aesthetic means "something you can see". Examples include:
  - position (i.e., on the x and y axes)
  - color
  - fill
  - shape (of points)
  - linetype
  - size
- Aesthetic mappings are set with the `aes()` function.
- Each type of `geom_` accepts only a subset of all aesthetics. Use geom help pages to see what mappings each geom accepts.

```
help.search("geom_", package = "ggplot2")
```

```
## starting httpd help server ... done
```

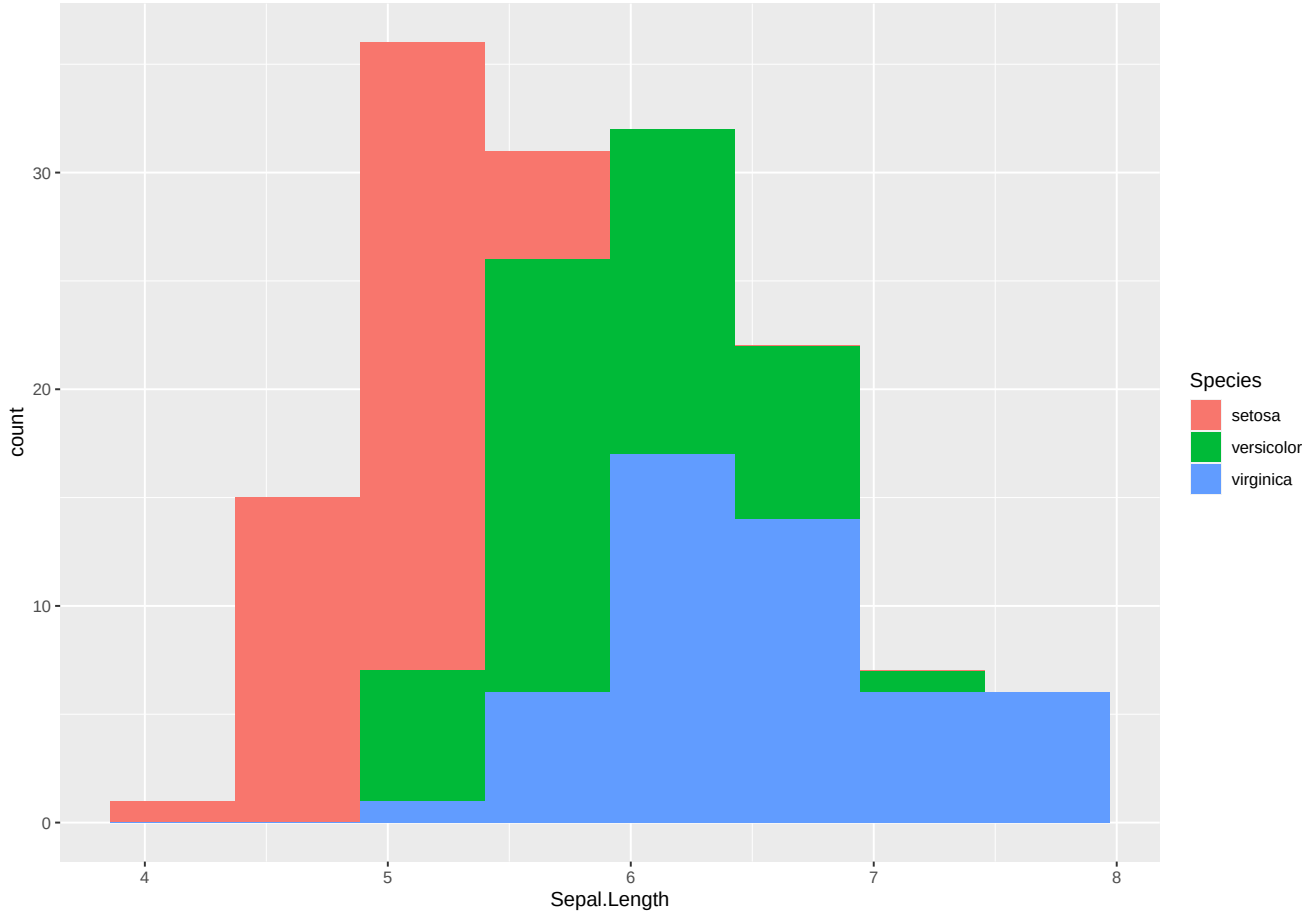
Base and ggplot2 styles

```
hist(iris$Sepal.Length)    # Base graphics
```



Base and ggplot2 styles

```
ggplot(data=iris, aes(x=Sepal.Length,fill=Species)) +
  geom_histogram(bins=8)
```



Quick plots with qplot()

- There are three key components of every plot: data, aesthetics and geoms.
- qplot() is analog to base plot(), where "q" means quick.
- We can use `qplot()` when we just want to get a **simple plot** without thinking about the grammar at all. `ggplot()` function is more flexible and robust than qplot for building a plot piece by piece
- It defines a plot in a single call with the basic syntax:

  qplot(dataframe, variables, [geom], options)

- A **sensible geom** (point, line, polygon, histogram, quantile, bar,) will be picked by default if it is not supplied.

The ggplot() grammar

- ggplot() - grammar of graphics plot, which provides more controls than qplot().
- Later in this course, ggplot() will also be used for animated plots.

```
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(
   mapping = aes(<MAPPINGS>),
   stat = <STAT>,
   position = <POSITION>
) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION>
```

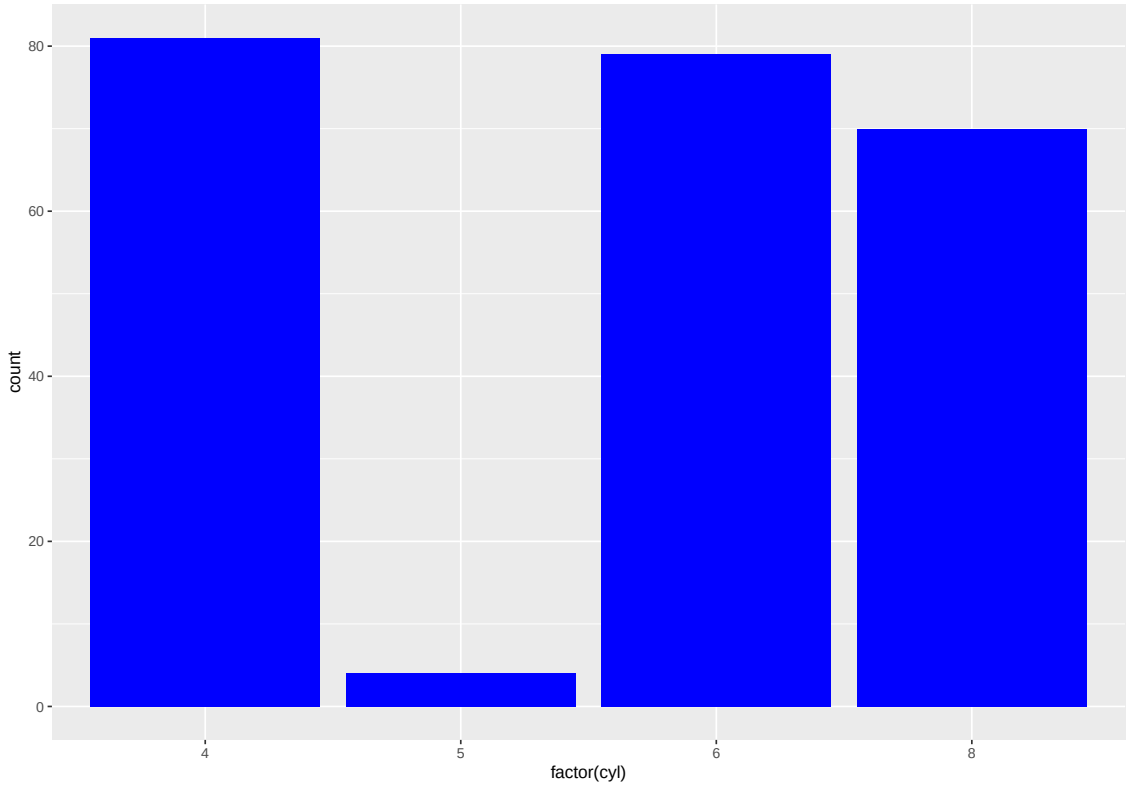
The `ggplot()` grammar

- Key inputs for a ggplot graph:
  - **Data**: a data.frame to visualize
  - **Geometric objects**: point, line, polygon, histogram, quantile, bar, ...
  - **Aesthetics**: mapping varialbes of the data to aesthetic attributes (position, size, shape, color, fill, transparency, ...)
  - **Scales**: mapping values of the data to visual values for each aesthetic (e.g. position, color, fill and shape scales)
  - **Statistical transformations**: bin, boxplot, density, contour, function, ...
  - **Coordinate system**: Cartesian, polar, map projection, ...
  - **Facet**: display split data in multi-panels (aka conditioning)
  - **Theme**: control non-data visual elements (title, axes, tick, ...)
- Every plot has three key components: **data**, **aesthetics** and **geoms**.

The ggplot() grammar

- We use two examples to show the key components of ggplot().

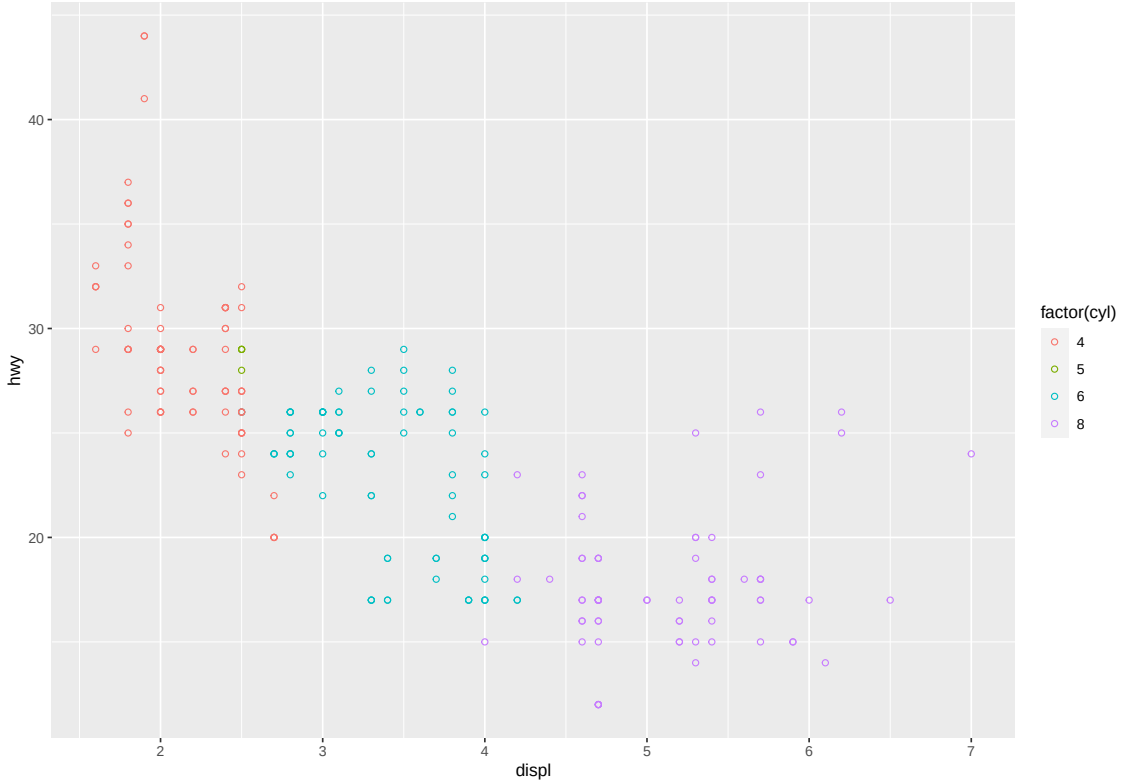
```
ggplot(data=mpg, aes( x=factor(cyl) ) ) +
 geom_bar(stat = "count", fill="blue")
```



The ggplot() grammar

- How are engine size and fuel economy related? Let's create a scatter plot of engine displacement and highway mpg with points colored by the number of cylinders.

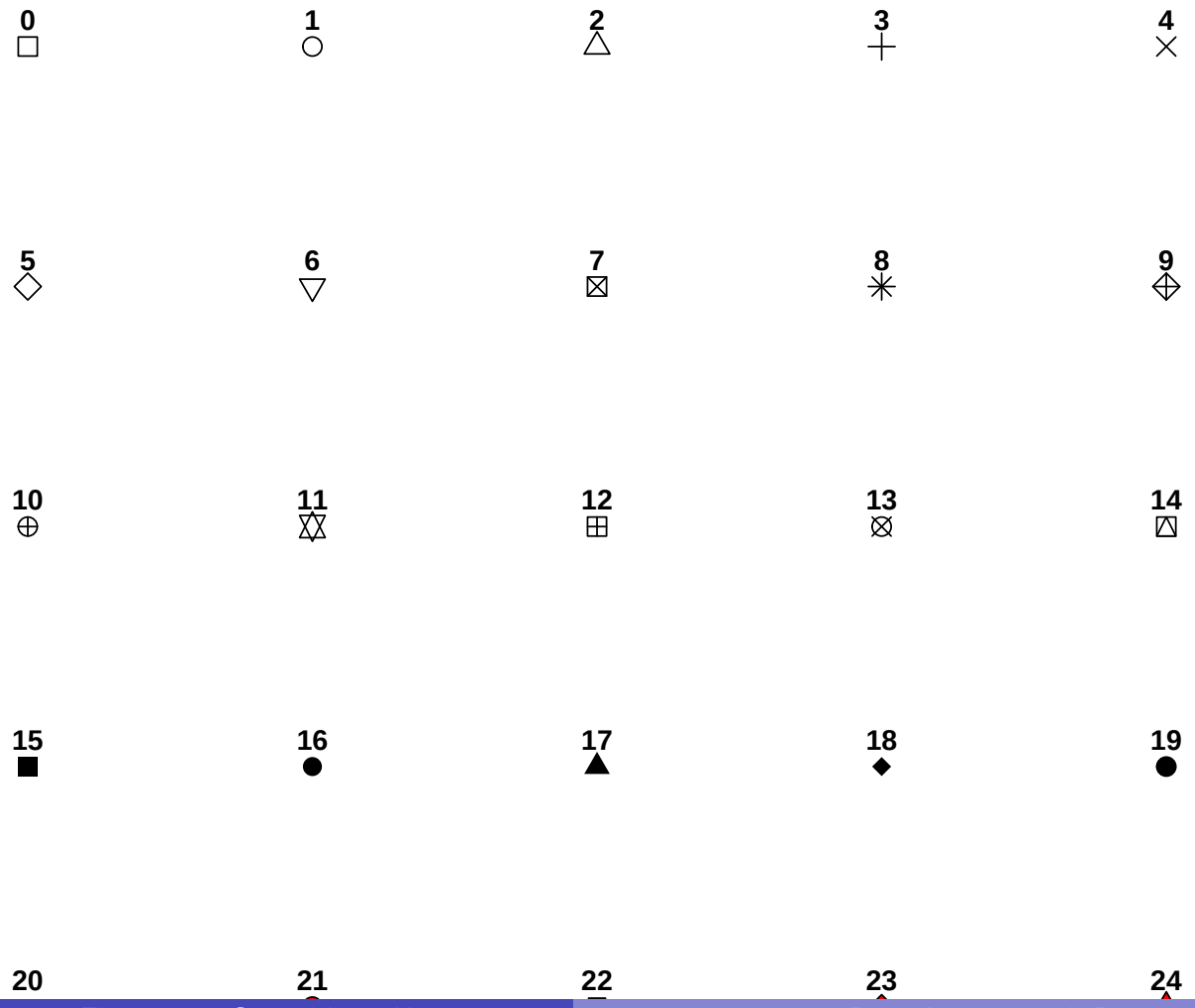
```
ggplot(data=mpg, aes( displ, hwy, color=factor(cyl)) ) +
        geom_point(pch=21)
```



The ggplot() grammar

- The shapes of point available in R are as follows.

```
?points
```



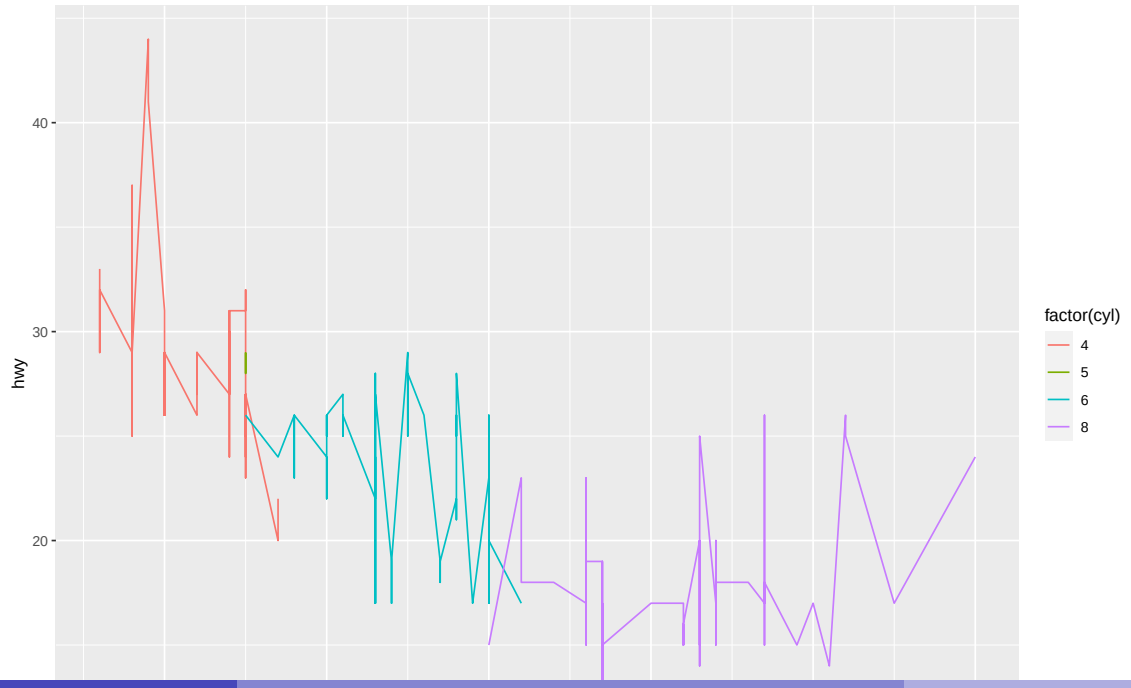
The `ggplot()` grammar: Mapping aesthetics to data

- How does `ggplot()` draw this plot?
- `aes()` function defines a mapping (by selecting the variables to be plotted and specifying how to present them in the graph, e.g. as x/y positions or characteristics such as size, shape, color, etc.
- Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms.
- In this example, the aesthetics are points according to the value of two variables, horizontal and vertical postion, point size, color and shape.

The ggplot() grammar: Mapping aesthetics to data

- **geoms** - graphical representations of the data in the plot (points, lines, bars). ggplot() offers many different geoms including:
  - geom_point() for scatter plots, dot plots, etc.
  - geom_boxplot() for boxplots
  - geom_line() for trend lines, time series, etc.
  - geom_smooth() for smoothing lines, produced by smoothing method in statistics

```
ggplot(data=mpg, aes(displ, hwy, color=factor(cyl)) ) +
        geom_line()
```



The `ggplot()` grammar: Faceting

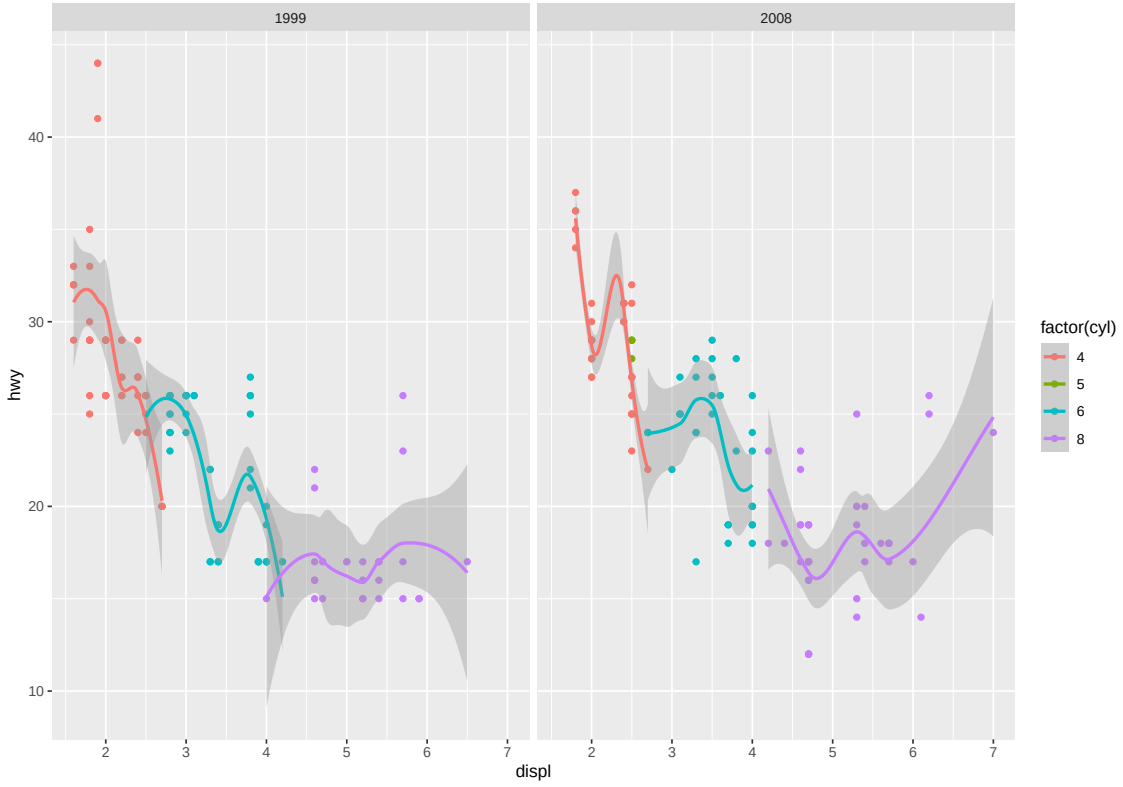
From the above graph, it can be seen that in ggplot2 we can produce many plots that don't make sense, yet are gramatically valid.

- **Facets** divide a plot into subplots based on the values of one or more categorical variables.
- There are two main functions for faceting:
  - **facet_grid()**: forms a matrix of panels defined by row and column faceting variables
  - **facet_wrap()**: wraps a 1d sequence of panels into 2d. This is generally a better use of screen space than `facet_grid()` because most displays are roughly rectangular.

The ggplot() grammar: Faceting

```
ggplot(data=mpg, aes(displ, hwy, color=factor(cyl)) ) +
  geom_point() +
  geom_smooth() +
  facet_grid(~year)
```

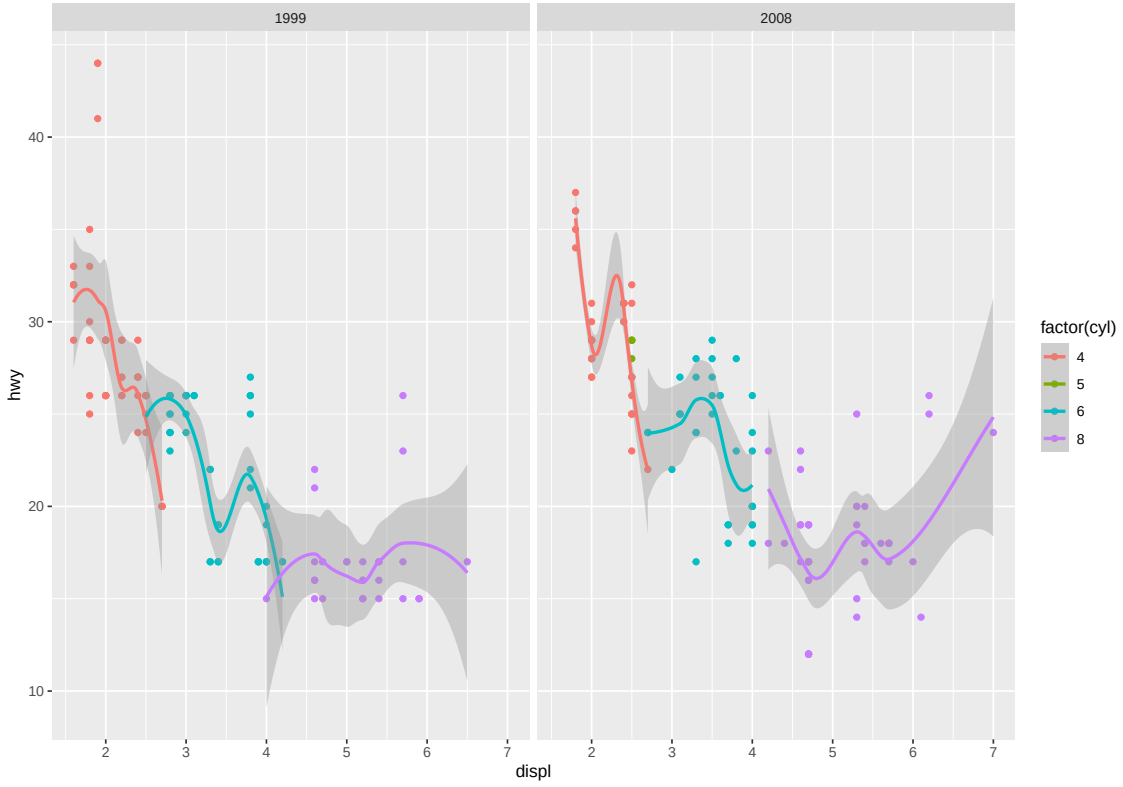
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The ggplot() grammar: Faceting

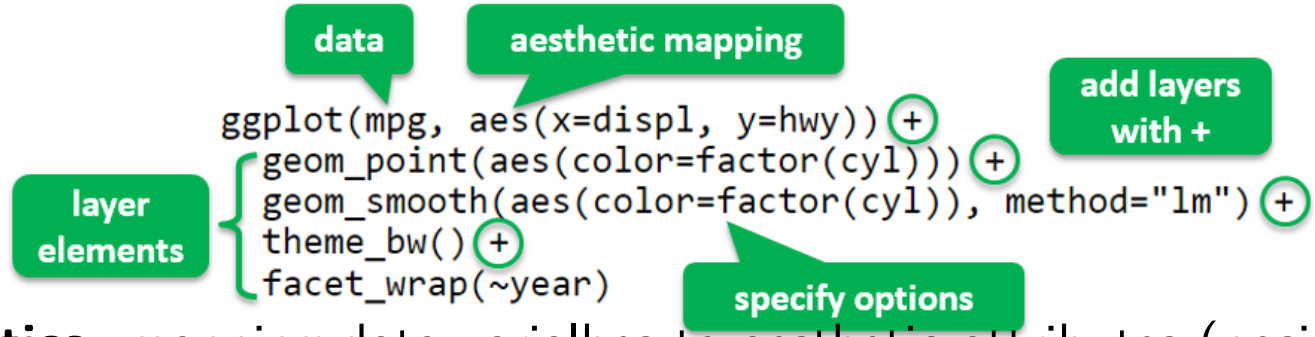
```
ggplot(data=mpg, aes(displ, hwy, color=factor(cyl)) ) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~year)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



The ggplot() grammar: layers

- ggplot graphics are built step by step by adding new elements. Adding layers in this fashion allows for extensive flexibility and customization of plots.

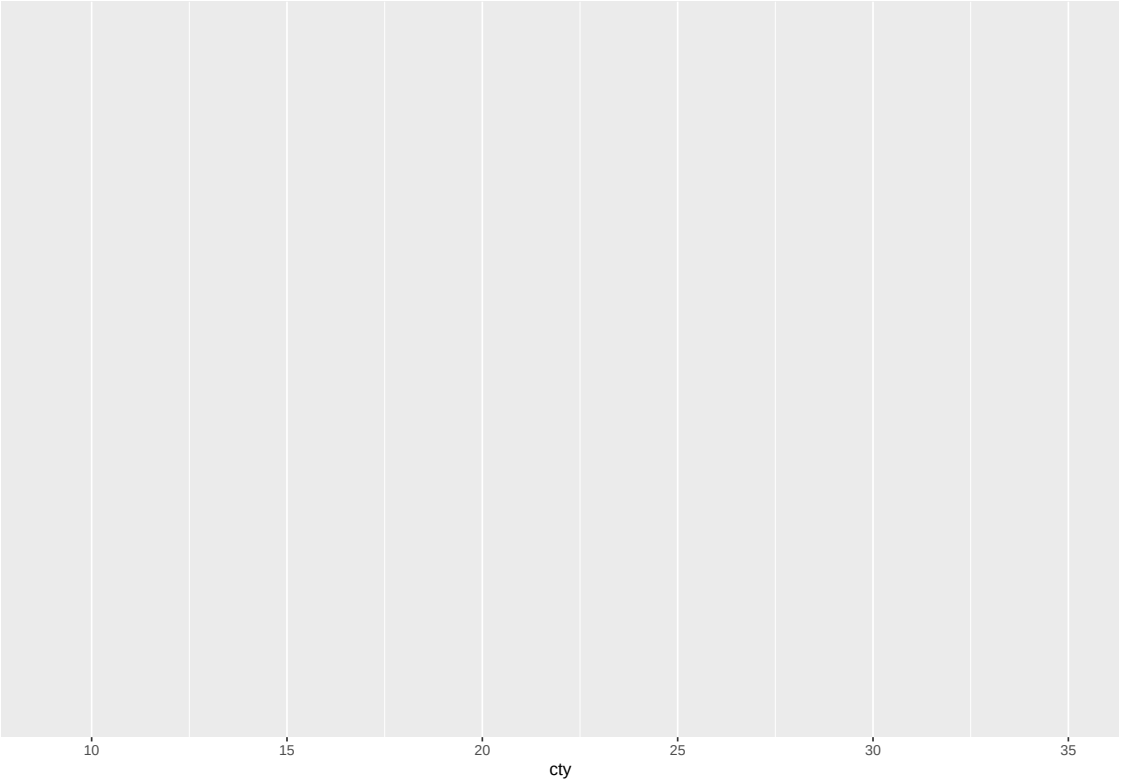


- **Aesthetics**: mapping data varialbes to aesthetic attributes (position, size, shape, color, . . .)
- **Geometric objects**: point, line, polygon, histogram, quantile, bar, . . .
- **Statistical transformations**: bin, boxplot, density, contour, function, . . .
- Other components for ggplots: **scales** (mapping values of the data to visual values for each aesthetic, e.g. position, color, fill and shape scales); **coordinate system** (cartesian, polar, map projection, ..); **facet** (conditioning display split data in multi-panels; theme (control non-data visual elements (title, axes, tick, . . .)

The ggplot() grammar: layers

- We start by creating a plot, named **P**, and finish by adding layers.

```
P = ggplot(data=mpg, aes(x = cty))
P;
```



The `ggplot()` grammar: layers

The following are some possible layers for a numerical variable:

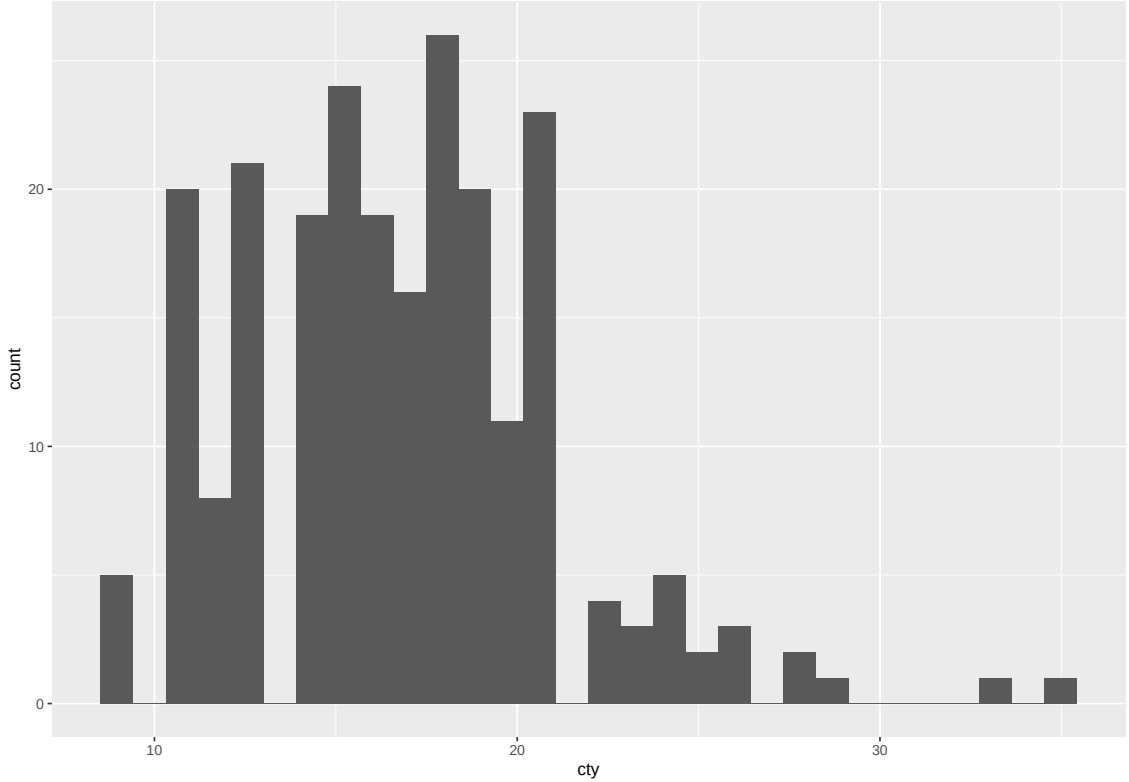
- `geom_area()` for area plot
- `geom_density()` for density plot
- `geom_dotplot()` for dot plot
- `geom_freqpoly()` for frequency polygon
- `geom_histogram()` for histogram plot
- `stat_ecdf()` for empirical cumulative density function
- `stat_qq()` for quantile - quantile plot

The ggplot() grammar: layers

- geom_histogram(): Histogram

```
P + geom_histogram()
```

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

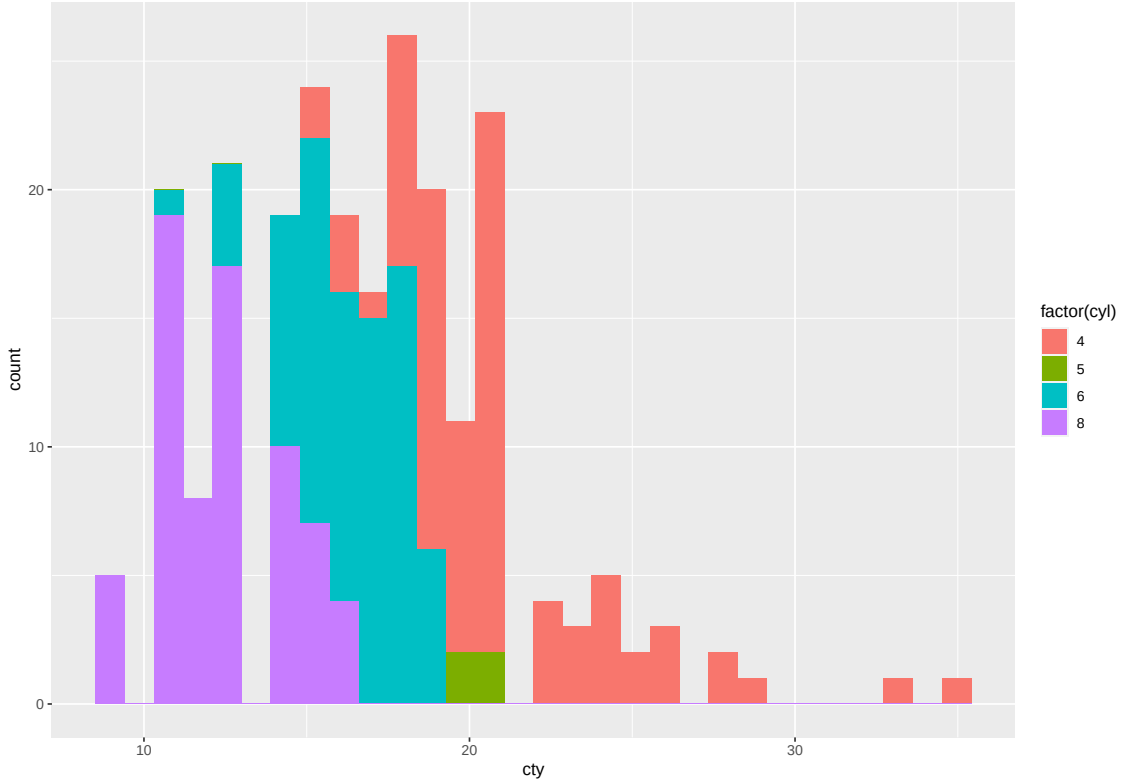


The ggplot() grammar: layers

- geom_histogram(): Histogram

```
# # change bin colors by cyl
P + geom_histogram(aes(fill = factor(cyl)))
```

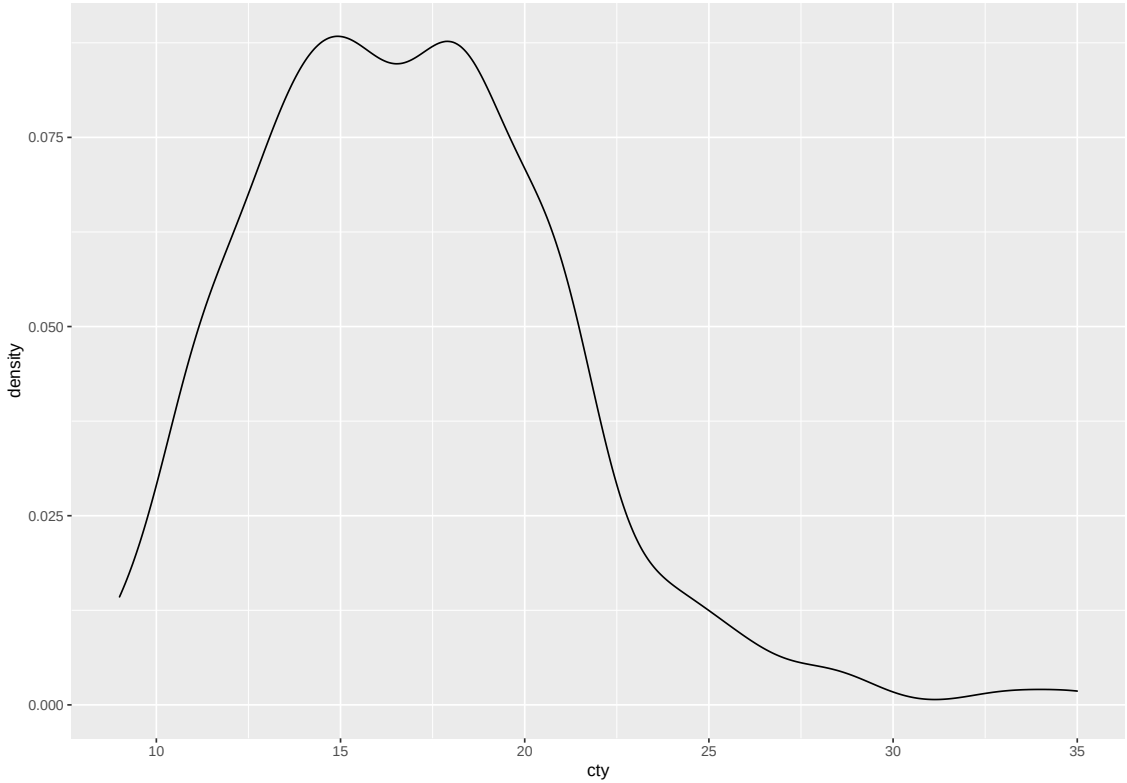
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`
```



The ggplot() grammar: layers

- geom_density(): kernal density estimate

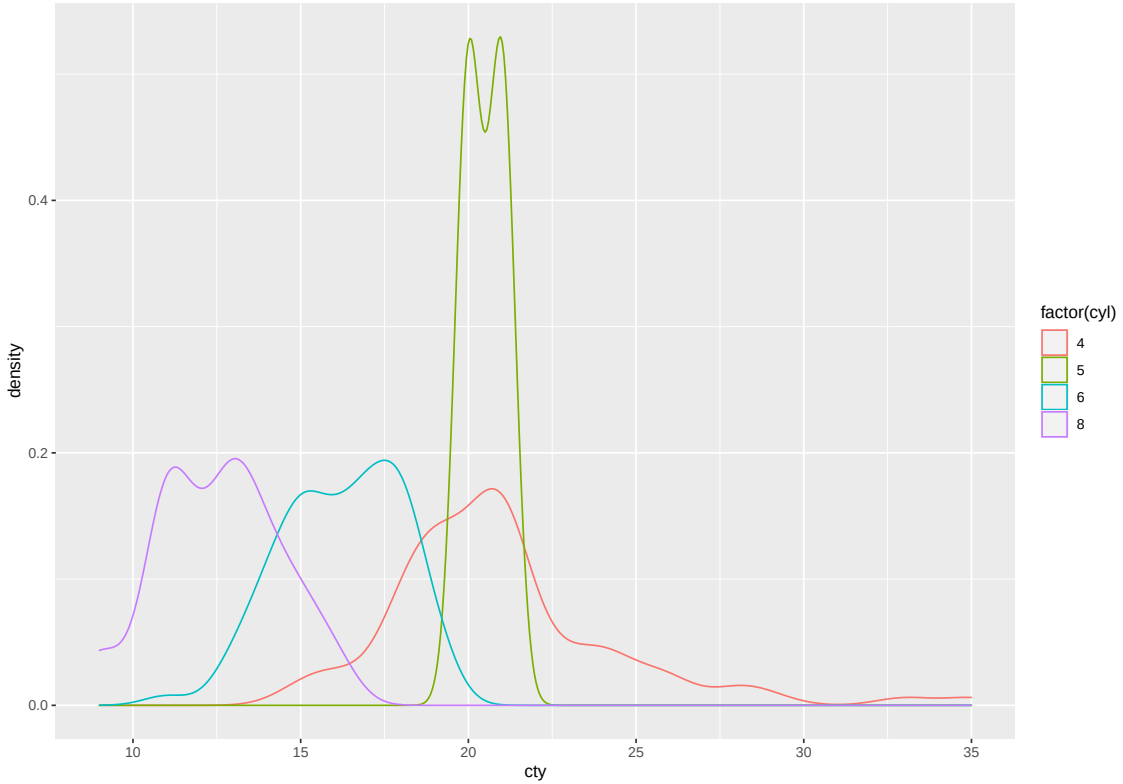
```
P + geom_density()
```



The ggplot() grammar: layers

- geom_density(): kernal density estimate

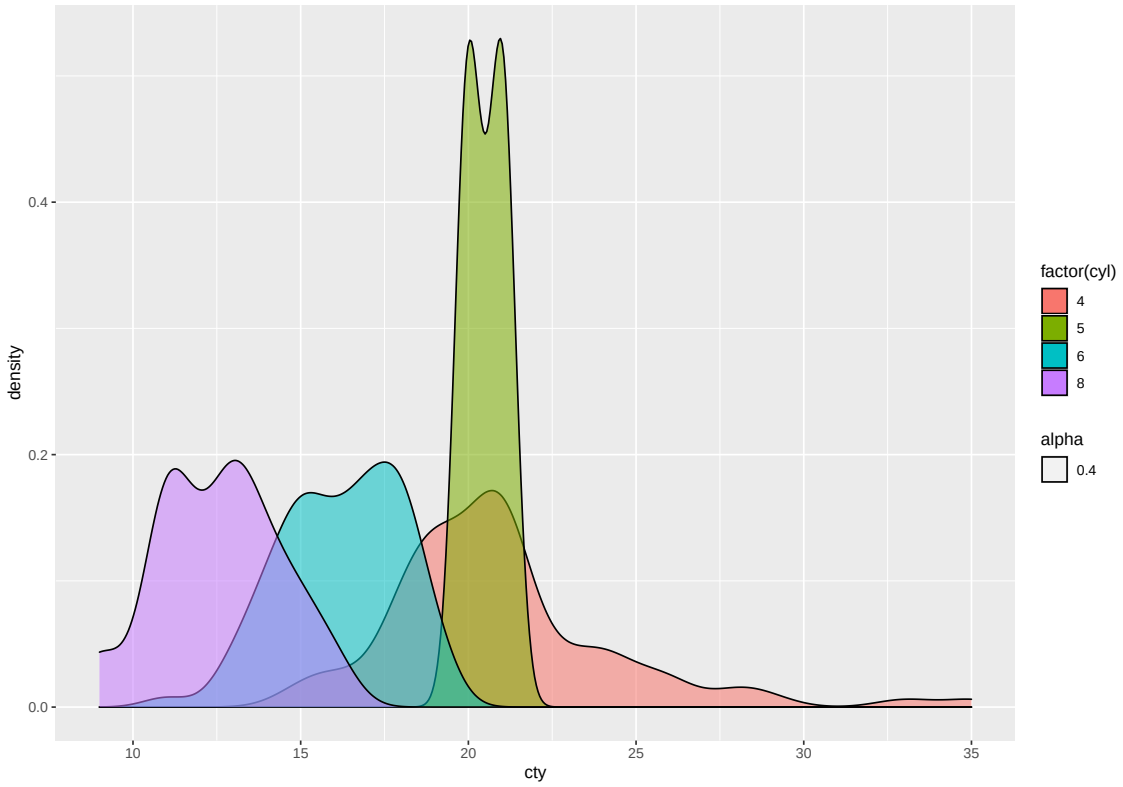
```
## change line colors by cyl
P + geom_density(aes(color = factor(cyl)))
```



The ggplot() grammar: layers

- geom_density(): kernal density estimate

```
# Use semi-transparent fill: alpha = 0.4
P + geom_density(aes(fill = factor(cyl),alpha=0.4))
```



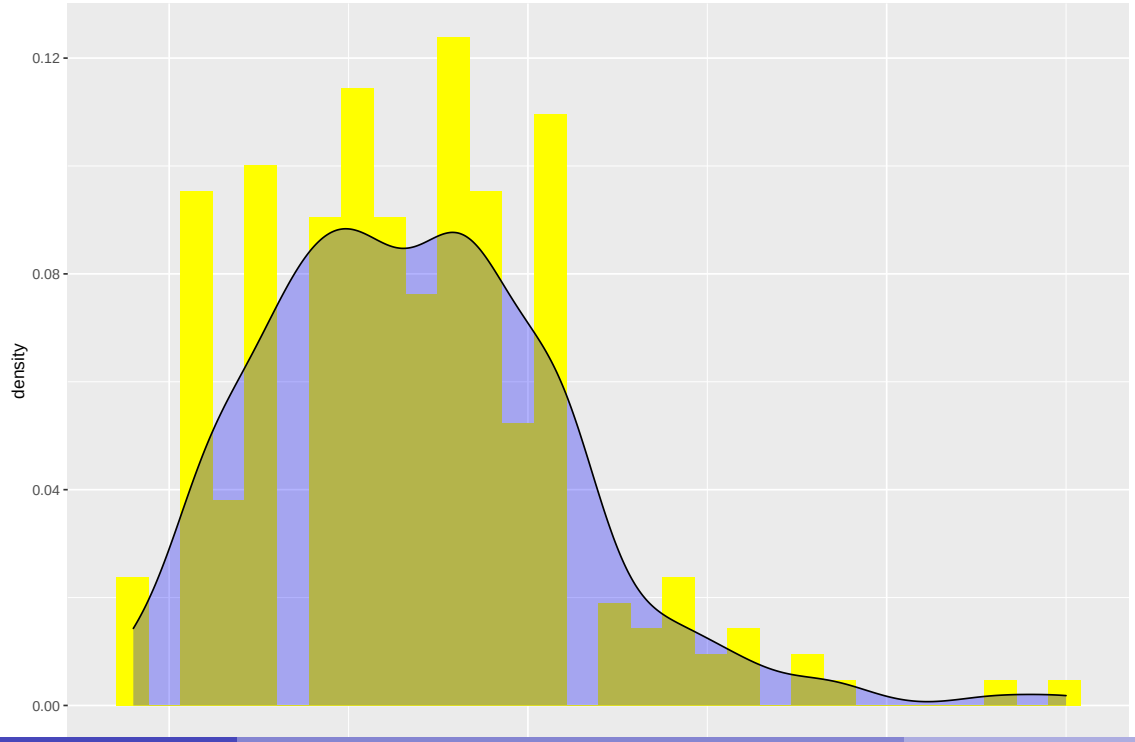
- To customize the plot, these arguments can be used: alpha, color, fill, linetype, size. Learn more here: ggplot2 density plot.

The ggplot() grammar: layers

- geom_histogram()+geom_density(): the histogram must be relative requency histogram

```
ggplot(data=mpg, aes(x = cty)) +
  geom_histogram(aes(y=..density..),fill='yellow') +
  # Histogram with density instead of count on y-axis
  geom_density(fill='blue',alpha=0.3)
```

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



Save your work

- All ggplot2 plots begin with a call to `ggplot()`, supplying default data and aesthethic mappings, specified by `aes()`.
- To save a plot to disk, use `ggsave()`. It defaults to saving the last plot that you displayed. For function references, please see https://ggplot2.tidyverse.org/reference/.

```
ggsave("p1.pdf")
ggsave("p2.png")
```

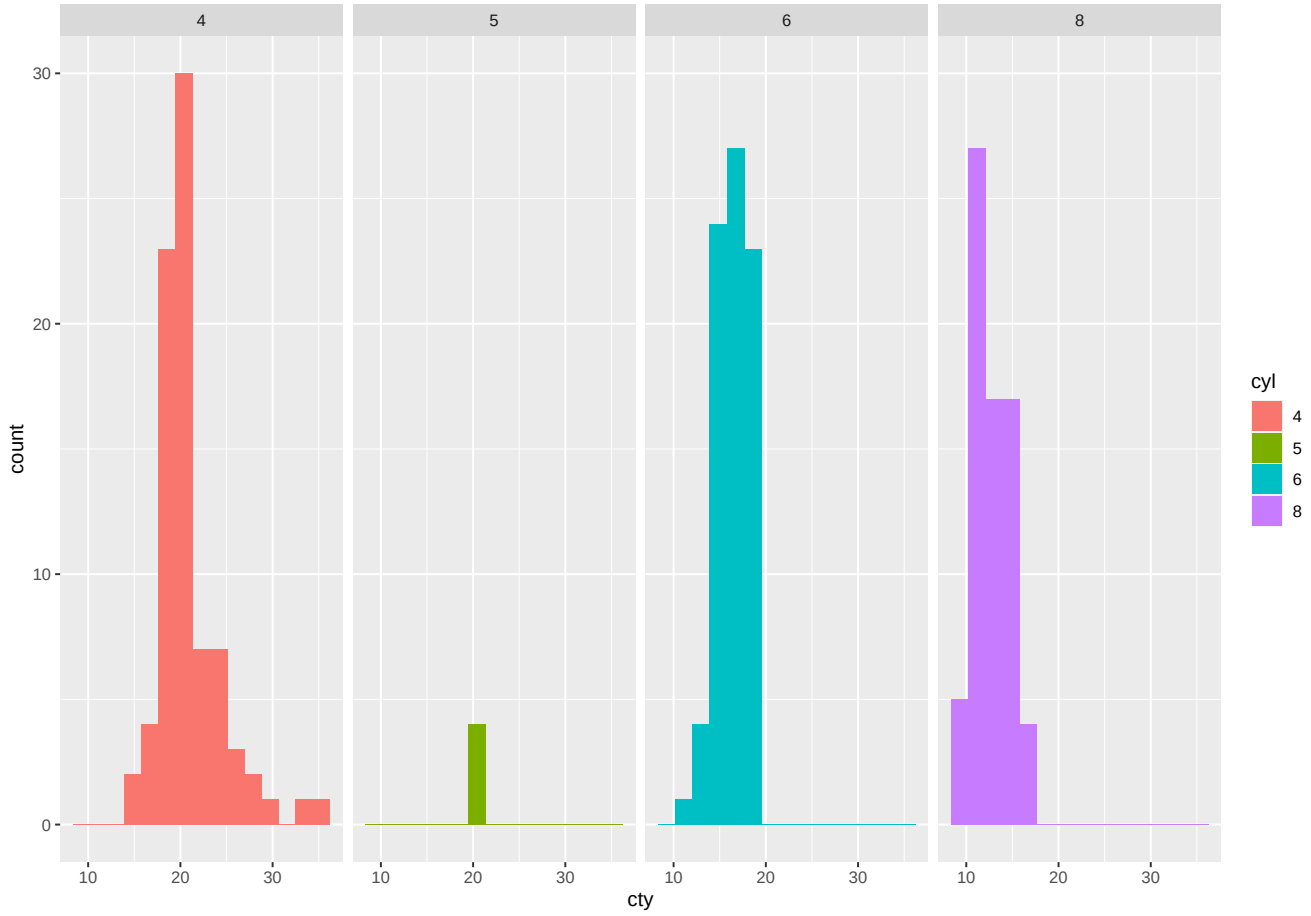
Titles

- We use the data mpg to show how to modify plot titles.
- The functions to be used are
  - ggtitle(label) # for the main title
  - xlab(label) # for the x axis label
  - ylab(label) # for the y axis label
  - labs(...) # for the main title, axis labels and legend titles

```
library(ggplot2)
mpg$cyl=as.factor(mpg$cyl) ## convert cyl from a int to a factor
attach(mpg)
P=ggplot(data=mpg, aes(x = cty)) +
  geom_histogram(aes(fill = cyl),bins = 15) +
  facet_grid(~cyl)
```

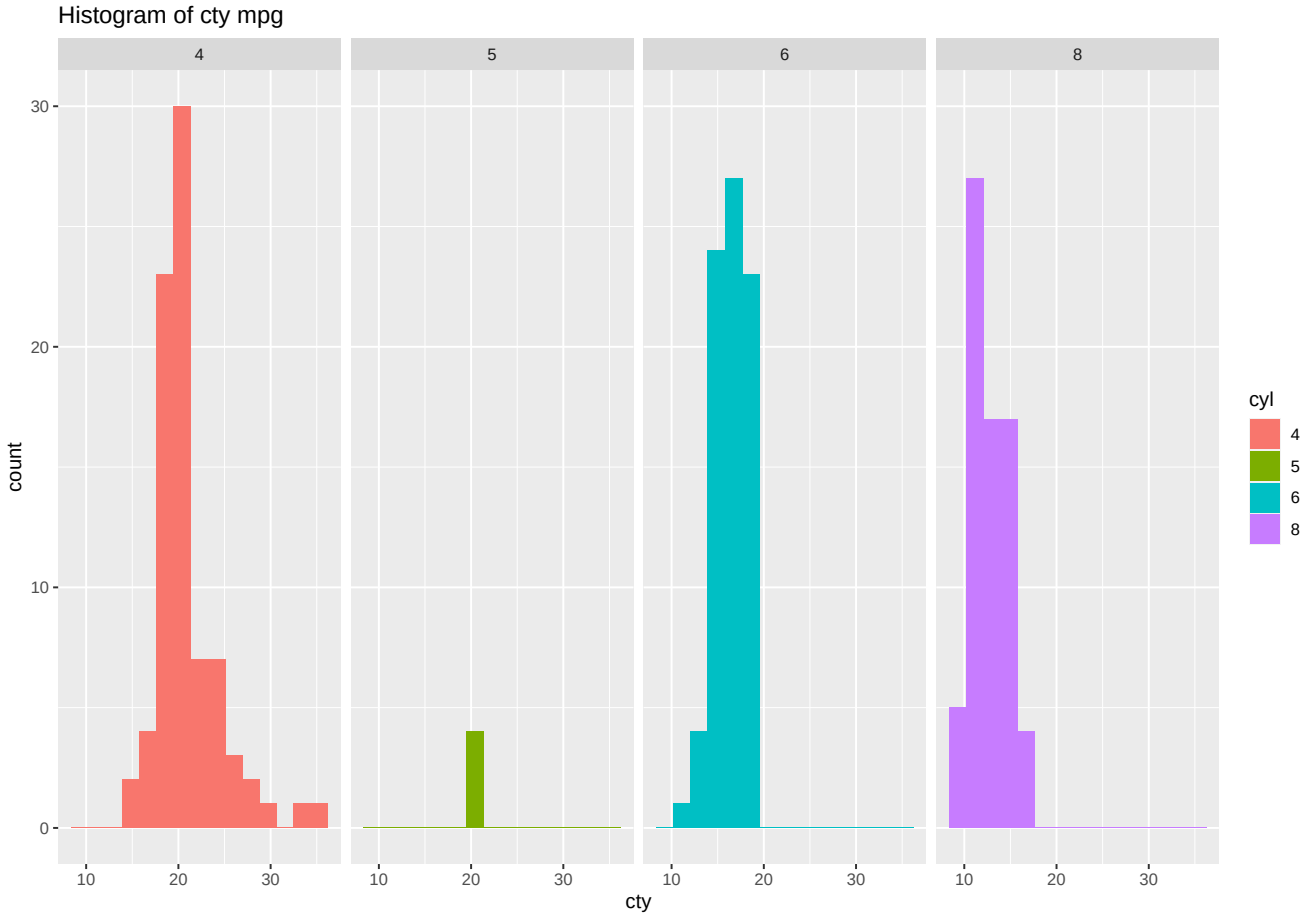
Titles

```
P;
```



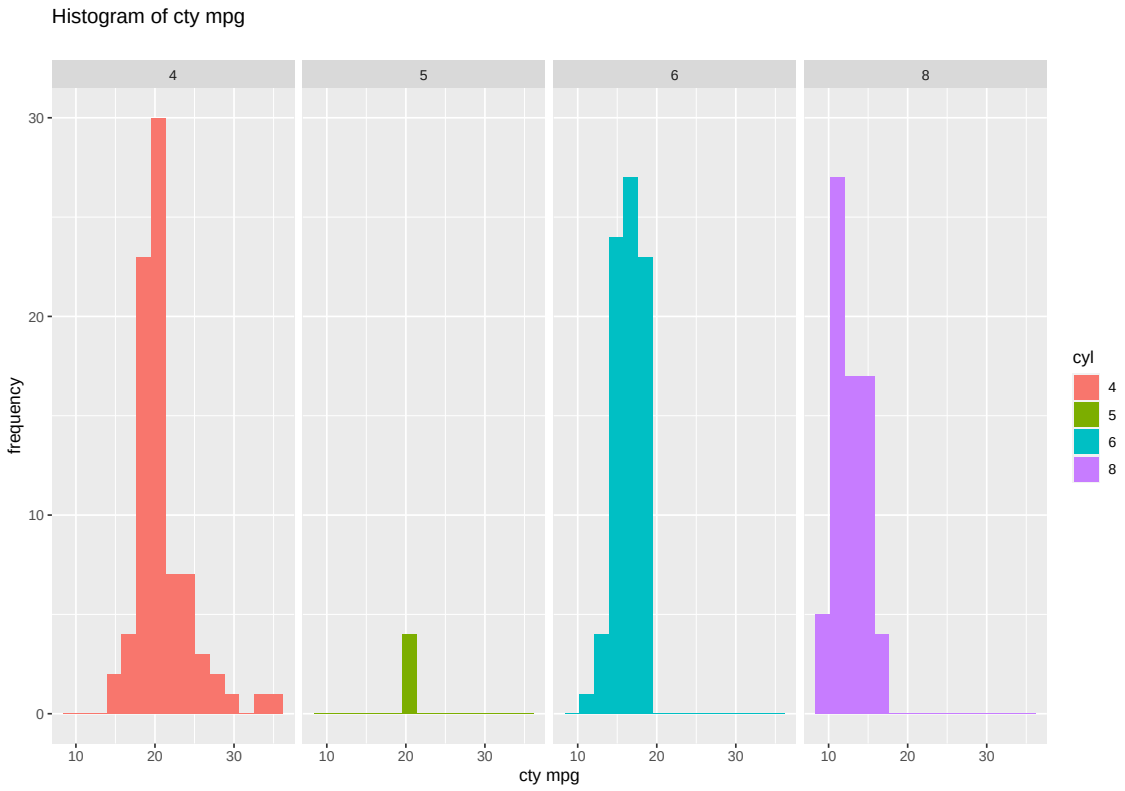
Titles

```
P  +   ggtitle("Histogram of cty mpg")
```



Titles

```
P  +   labs(title="Histogram of cty mpg", subtitle = " ",
          x="cty mpg", y="frequency")
```



Titles

- Main title and, x and y axis labels can be customized using the functions theme() and element_text().

```
# main title
P + theme(plot.title = element_text(family, face, color, size))
# x axis title
P + theme(axis.title.x = element_text(family, face, color, size))
# y axis title
P + theme(axis.title.y = element_text(family, face, color, size))
```

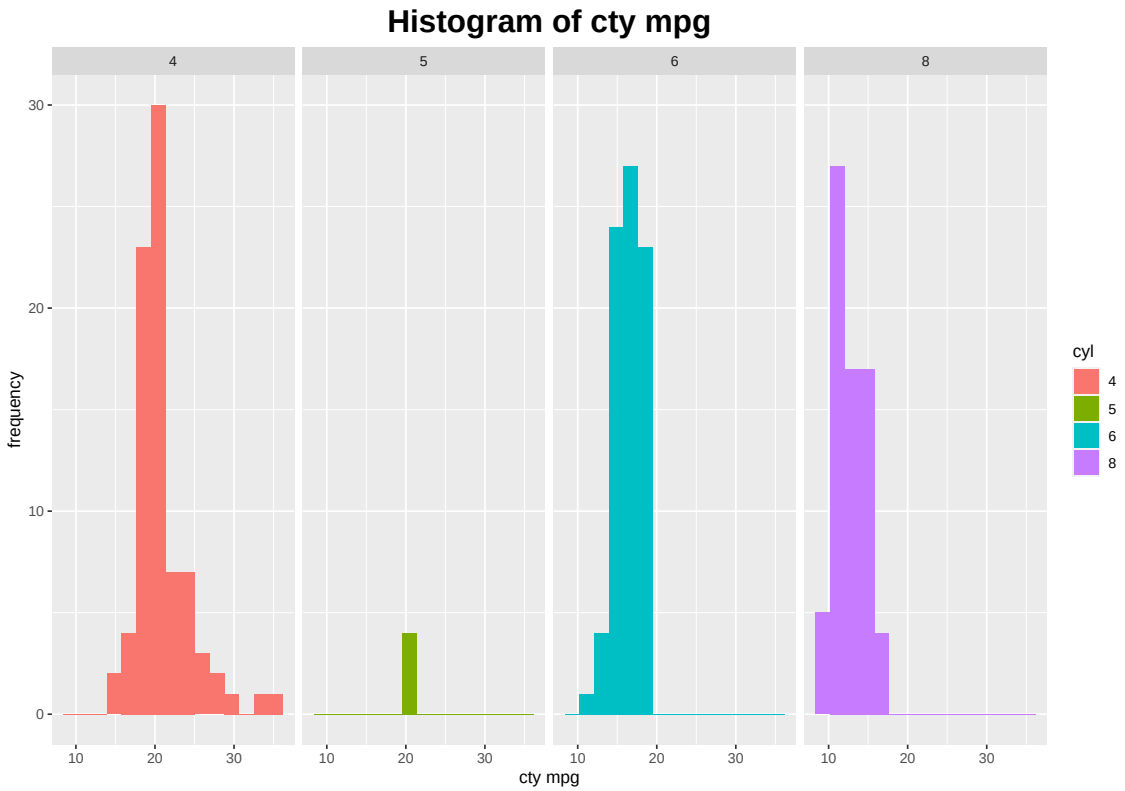
Titles

- The arguments are
  - **family**: font family
  - **face** : font face. Possible values are "plain", "italic", "bold" and "bold.italic"
  - **color** : text color
  - **size** : text size in pts
  - **hjust** : horizontal justification (in [0, 1])
  - **vjust** : vertical justification (in [0, 1])
  - **lineheight** : line height. In multi-line text, the lineheight argument is used to change the spacing between lines.
  - **color** : an alias for color

```
?theme
```

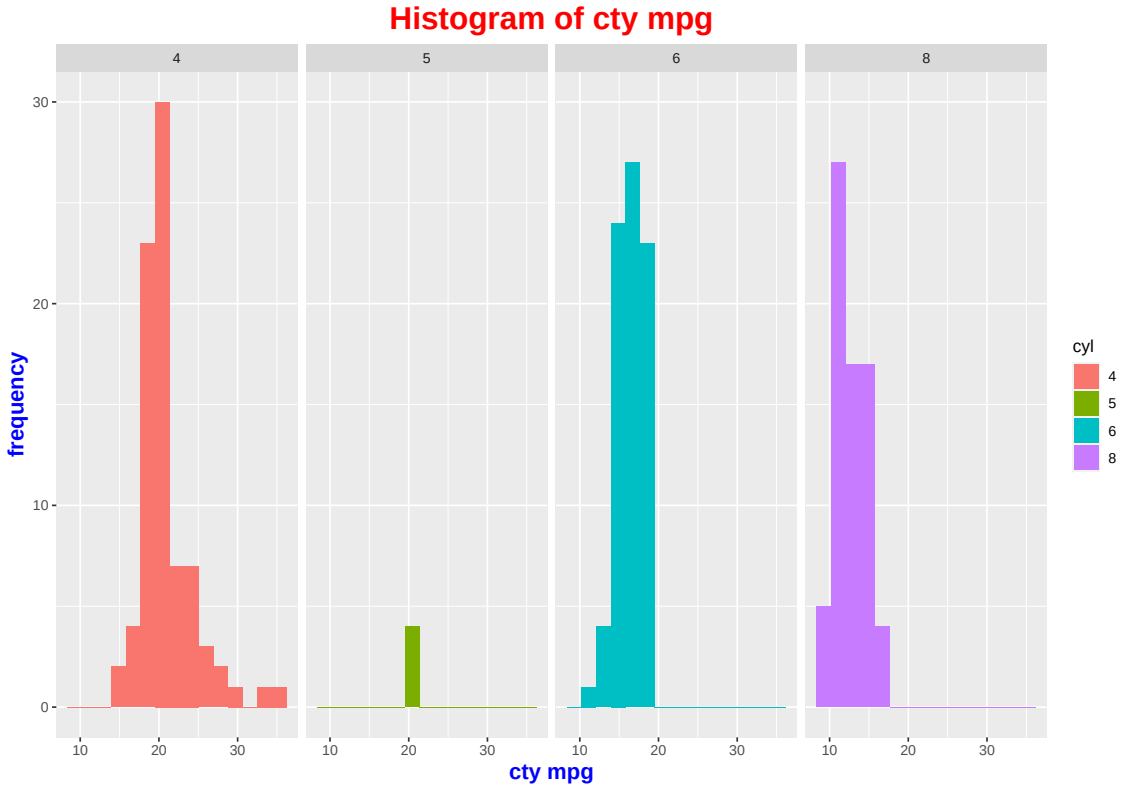
Titles

```
P  +   ggtitle("Histogram of cty mpg") +
 theme(plot.title = element_text(hjust = 0.5,size = 20,
 face = "bold")) + #centering the title and set the size
 xlab("cty mpg") + ylab("frequency")
```



Titles

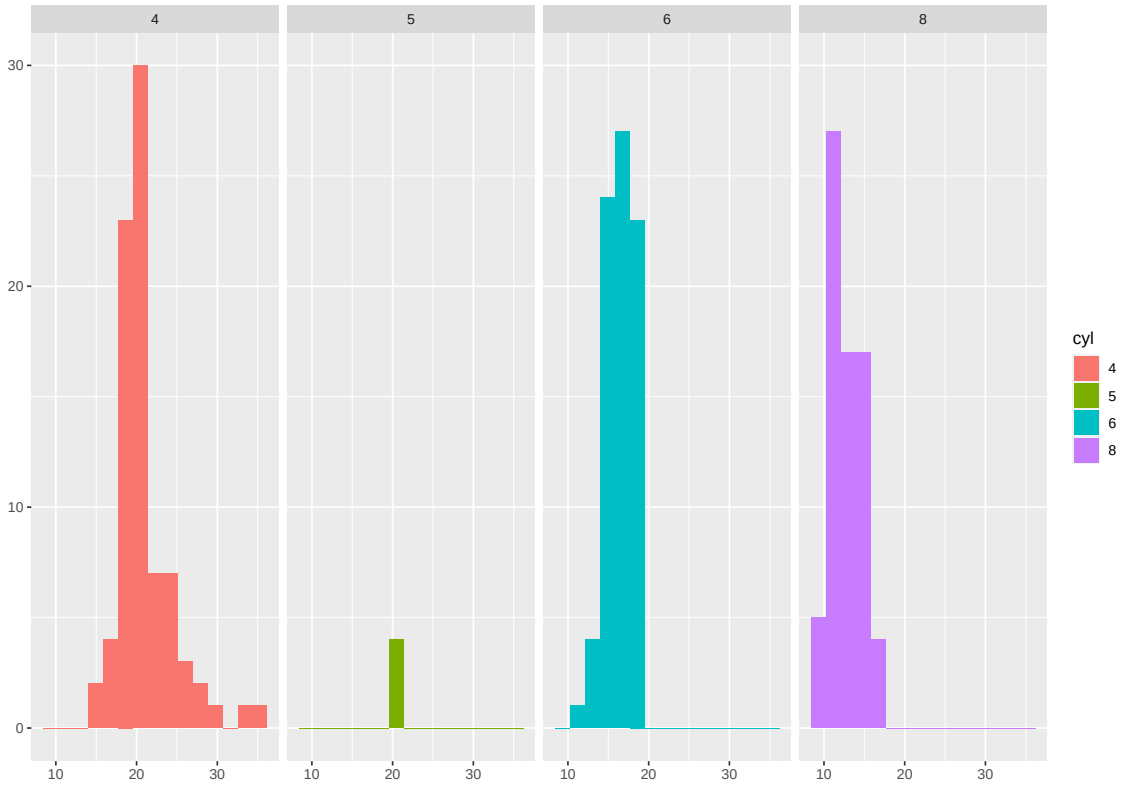
```
P + ggtitle("Histogram of cty mpg") +
  theme(plot.title = element_text(color="red", hjust = 0.5,
  size = 20, face = "bold"),
axis.title.x = element_text(color="blue", size=14, face="bold"),
axis.title.y = element_text(color="blue", size=14, face="bold")) +
  xlab("cty mpg") + ylab("frequency")
```



Titles

- It's possible to hide the main title and axis labels using the function `element_blank()`.

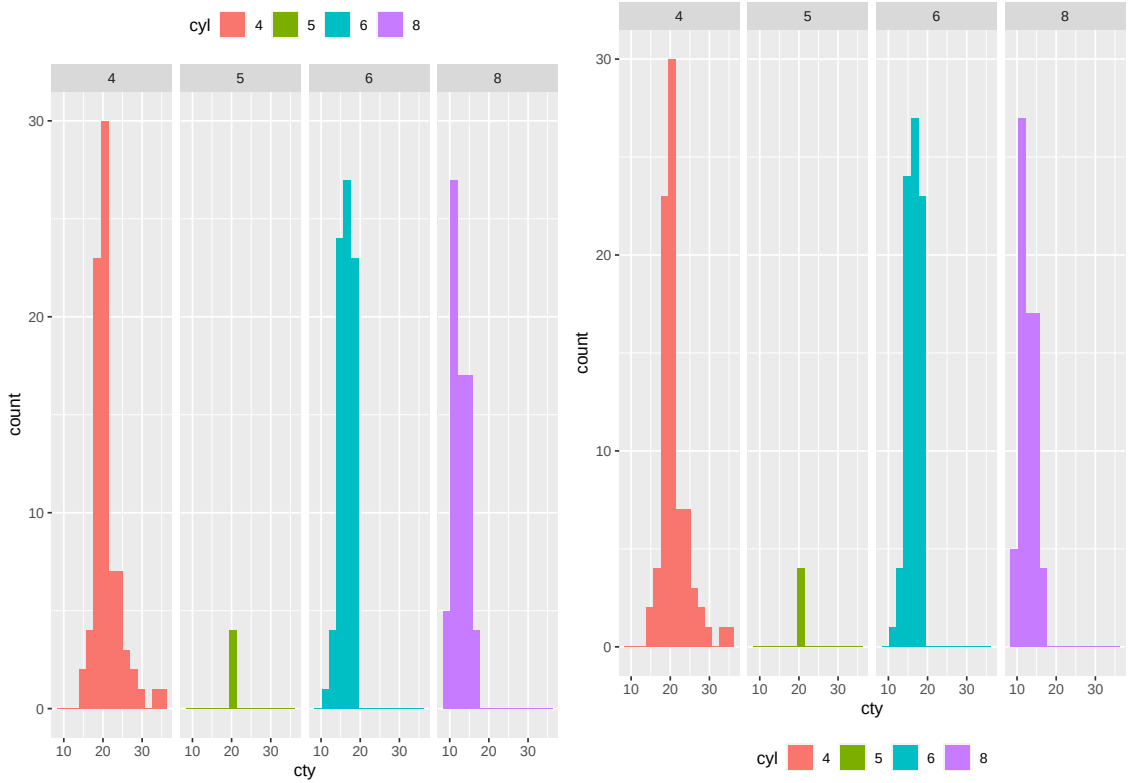
```
P + theme(
  plot.title = element_blank(),
  axis.title.x = element_blank(),
  axis.title.y = element_blank() )
```



Legend

- Change the legend position

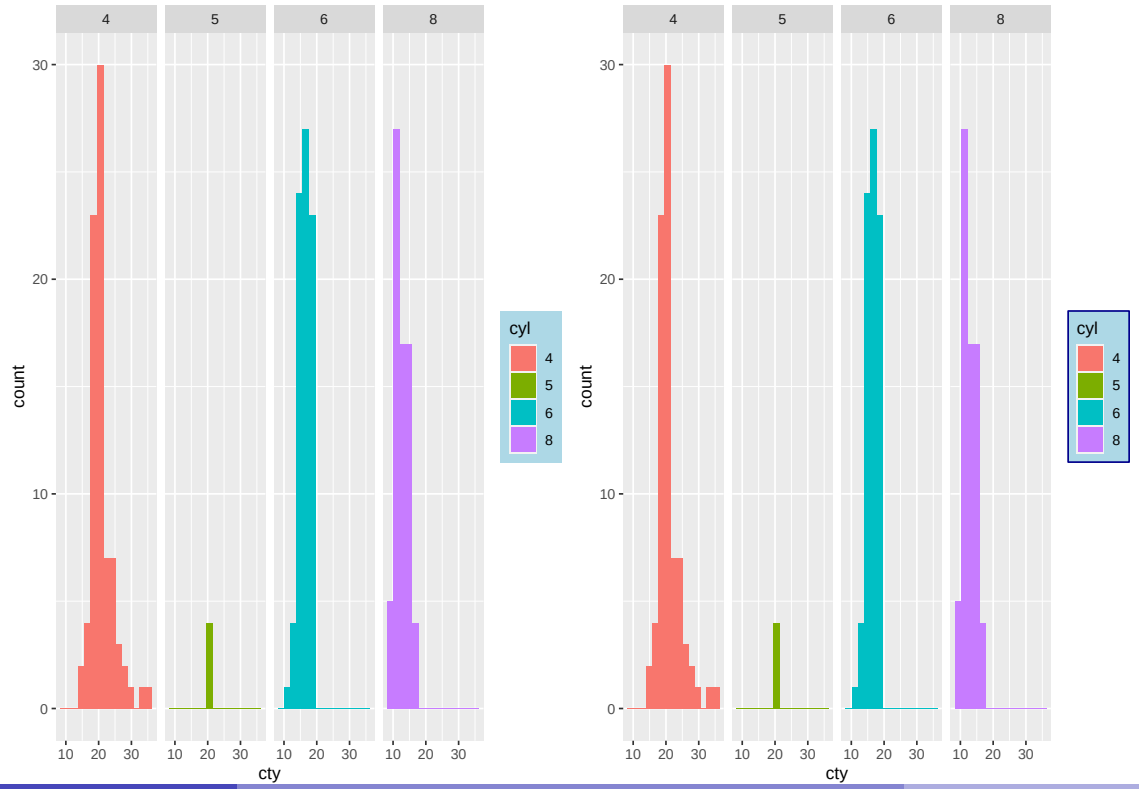
```
library(gridExtra)
p1= P + theme(legend.position="top")
p2= P + theme(legend.position="bottom")
grid.arrange(p1, p2, ncol=2)
```



Legend

- Change the background color of the legend box

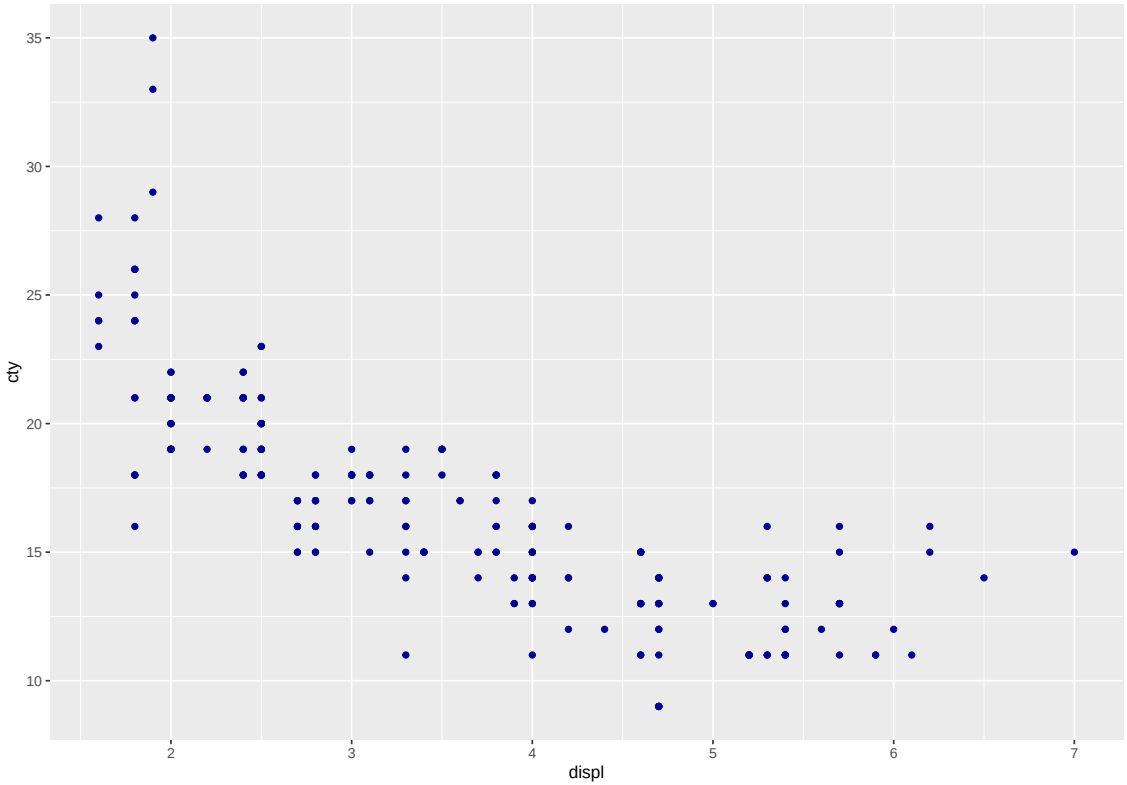
```
library(gridExtra)
p1= P + theme(legend.background=element_rect(fill="lightblue",
        size=0.5, linetype="solid"))
p2= P + theme(legend.background=element_rect(fill="lightblue",
  size=0.5, linetype="solid", color ="darkblue"))
grid.arrange(p1, p2, ncol=2)
```



Colors

- Change colors manually

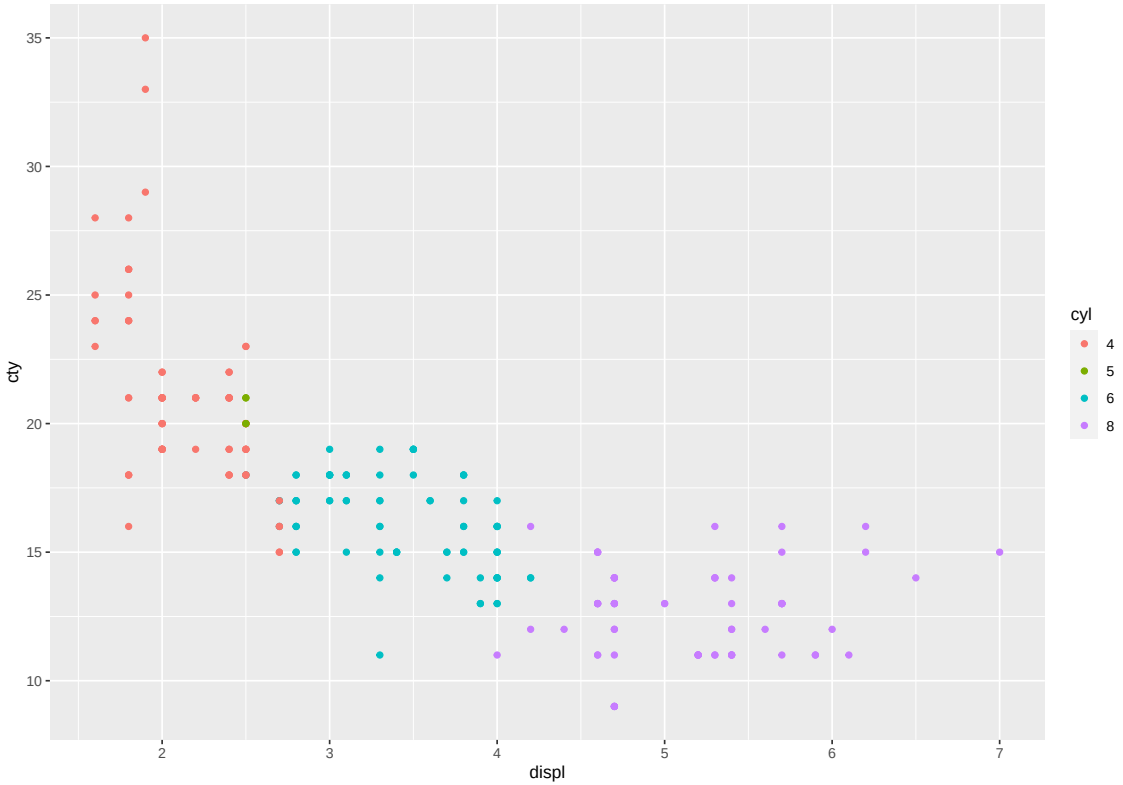
```
ggplot(mpg, aes(x=displ, y=cty)) +
  geom_point(color='darkblue')
```



Colors

- Default colors by groups

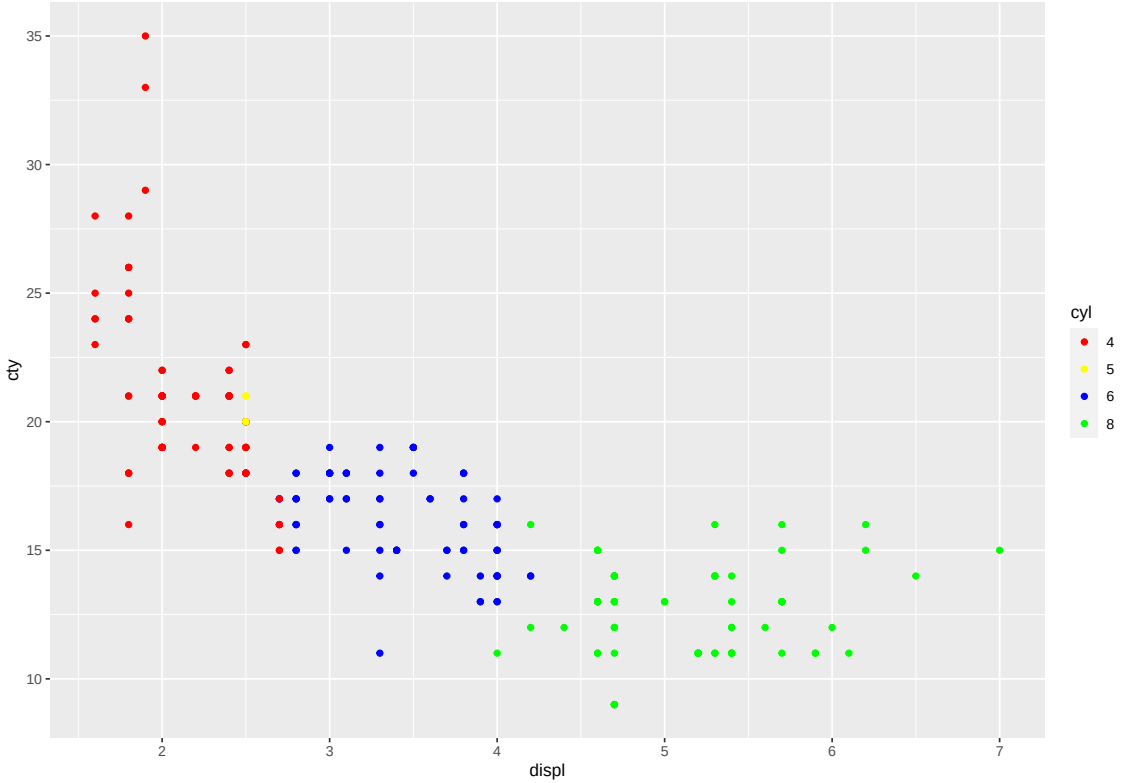
```
ggplot(mpg, aes(x=displ, y=cty,color=cyl)) +
  geom_point()
```



Colors

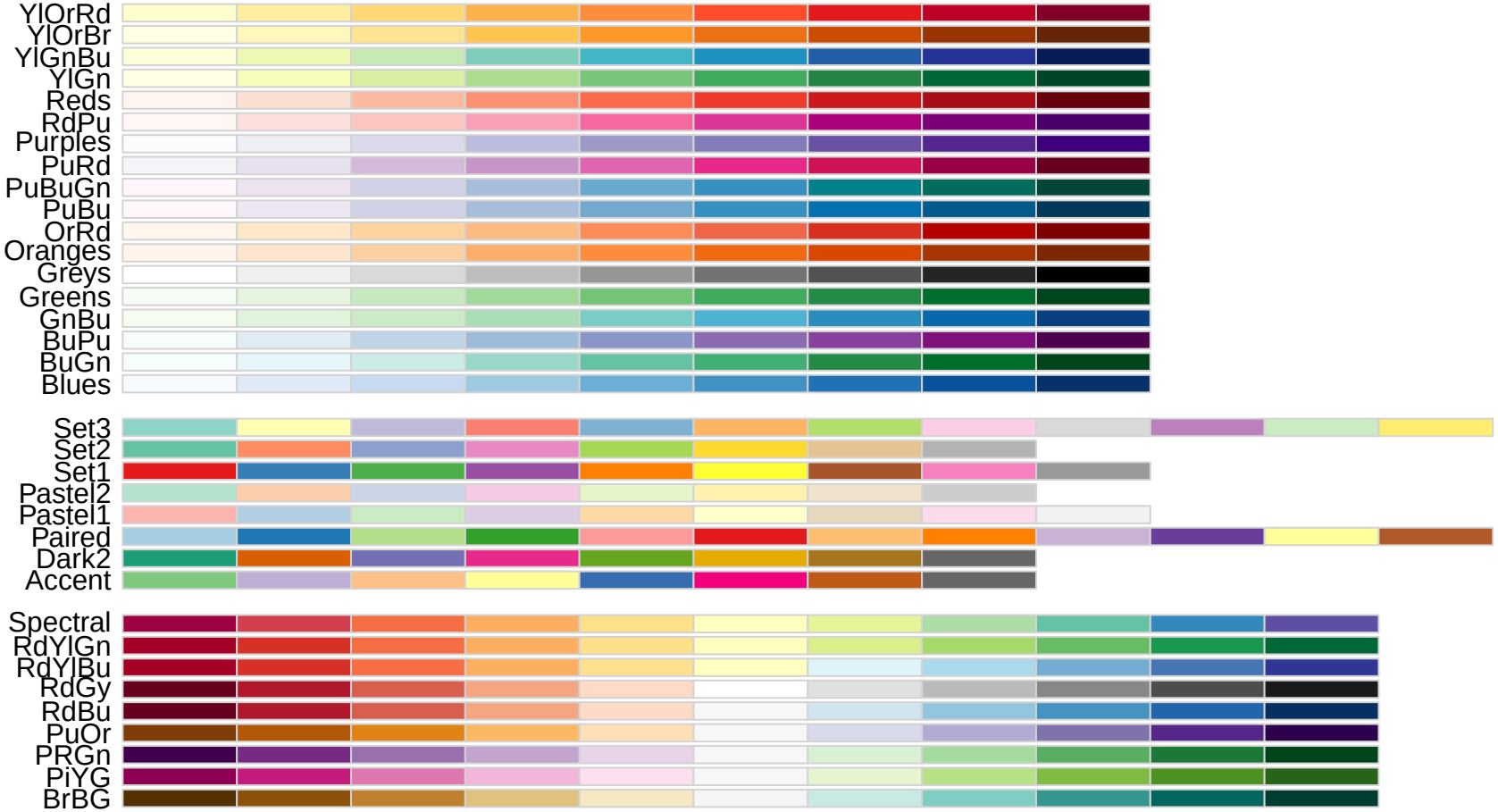
- Change colors by groups

```
ggplot(mpg, aes(x=displ, y=cty,color=cyl)) +
  geom_point() +
  scale_color_manual(breaks = c("4", "5", "6","8"),
  values=c("red", "yellow","blue", "green"))
```



Colors

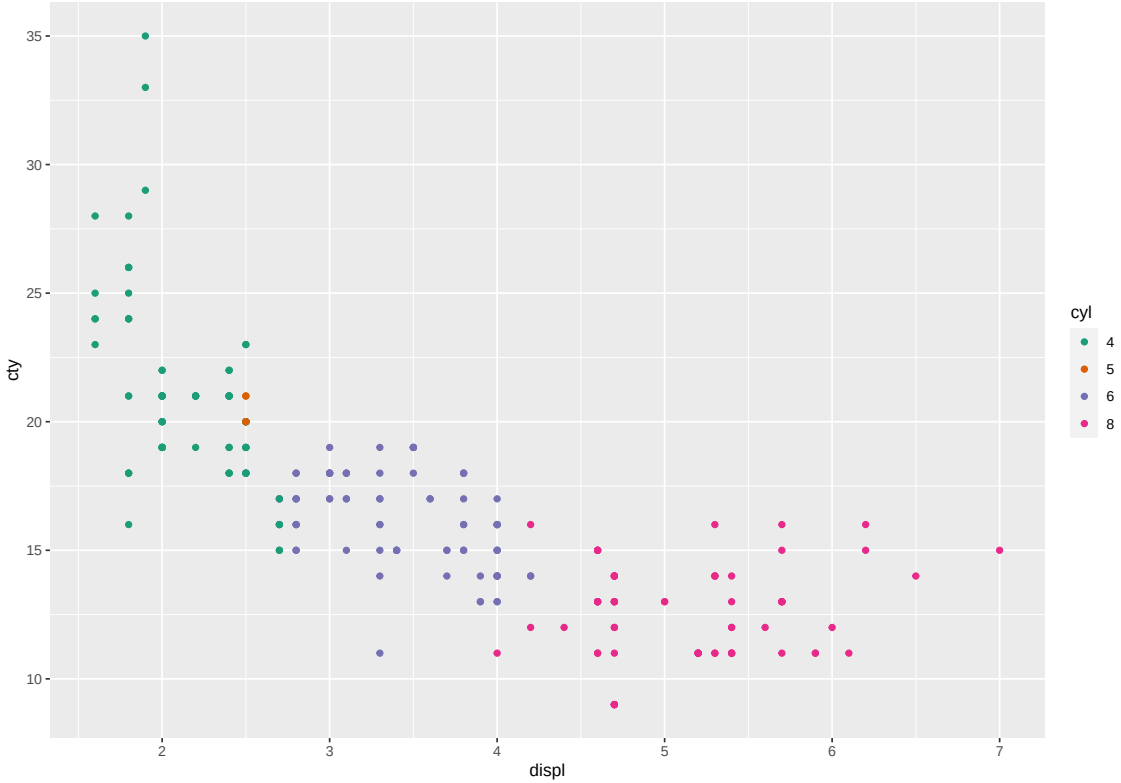
- Use RColorBrewer palettes
- The color palettes available in the **RColorBrewer** package are described here : color in R.
- The available color palettes in the RColorBrewer package are:



Colors

- Use RColorBrewer palettes

```
library(RColorBrewer)
ggplot(mpg, aes(x=displ, y=cty,color=cyl)) +
  geom_point() +
  scale_color_brewer(palette="Dark2")
```



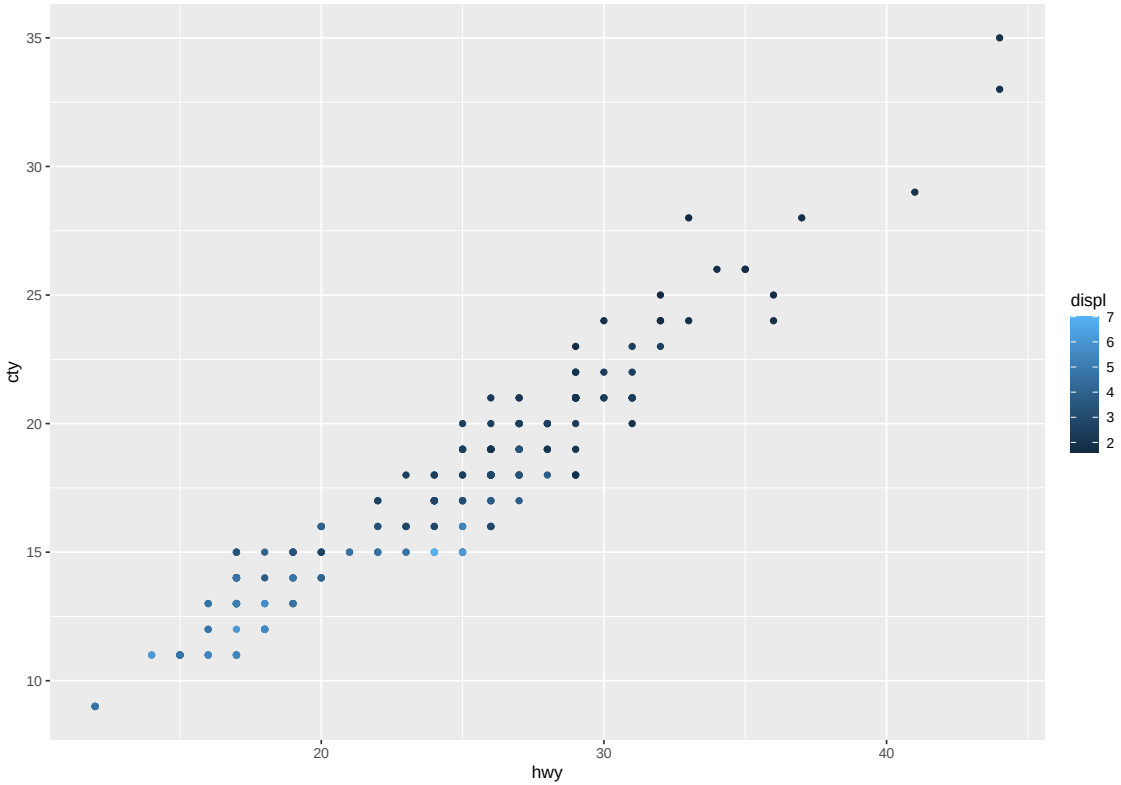
Colors

- Continuous colors: he graph can be colored according to the values of a continuous variable using the functions:
  - scale_color_gradient(), scale_fill_gradient() for sequential gradients between two colors(low-high)
  - scale_color_gradient2(), scale_fill_gradient2() for diverging gradients (low-mid-high)
  - scale_color_gradientn(), scale_fill_gradientn() for gradient between n colors

Colors

- Continuous colors

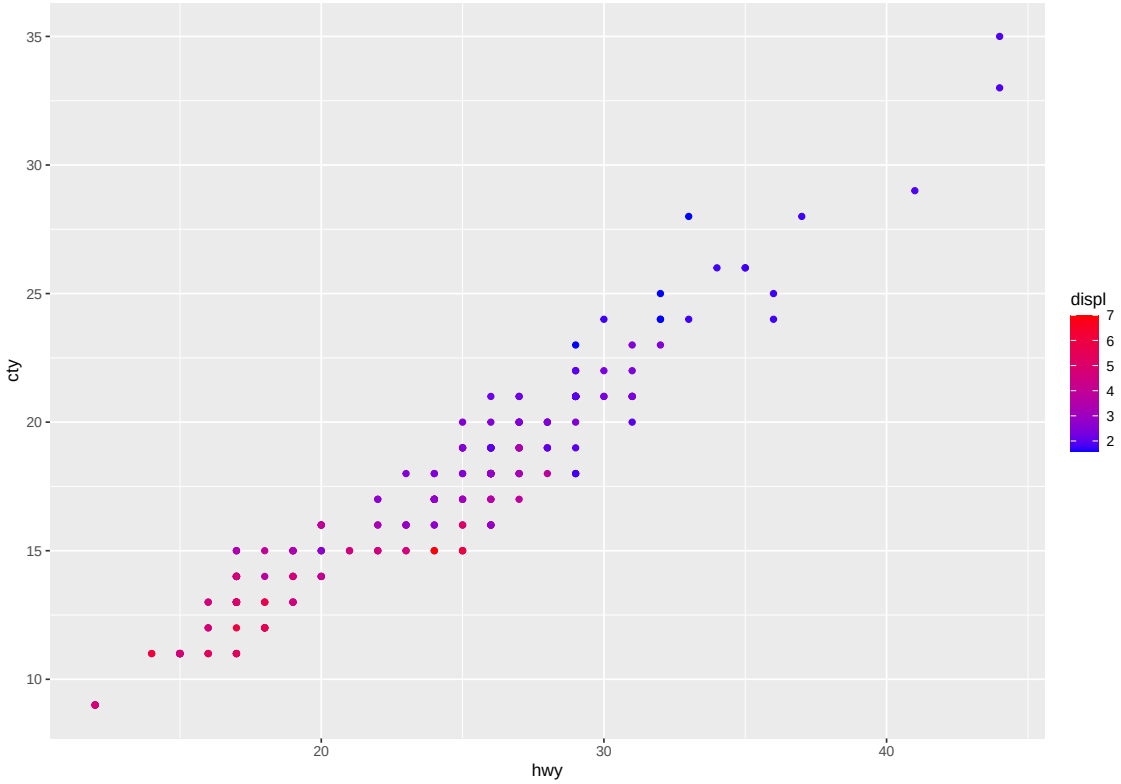
```
ggplot(mpg, aes(x=hwy, y=cty, color=displ)) +
  geom_point()
```



Colors

- Continuous colors

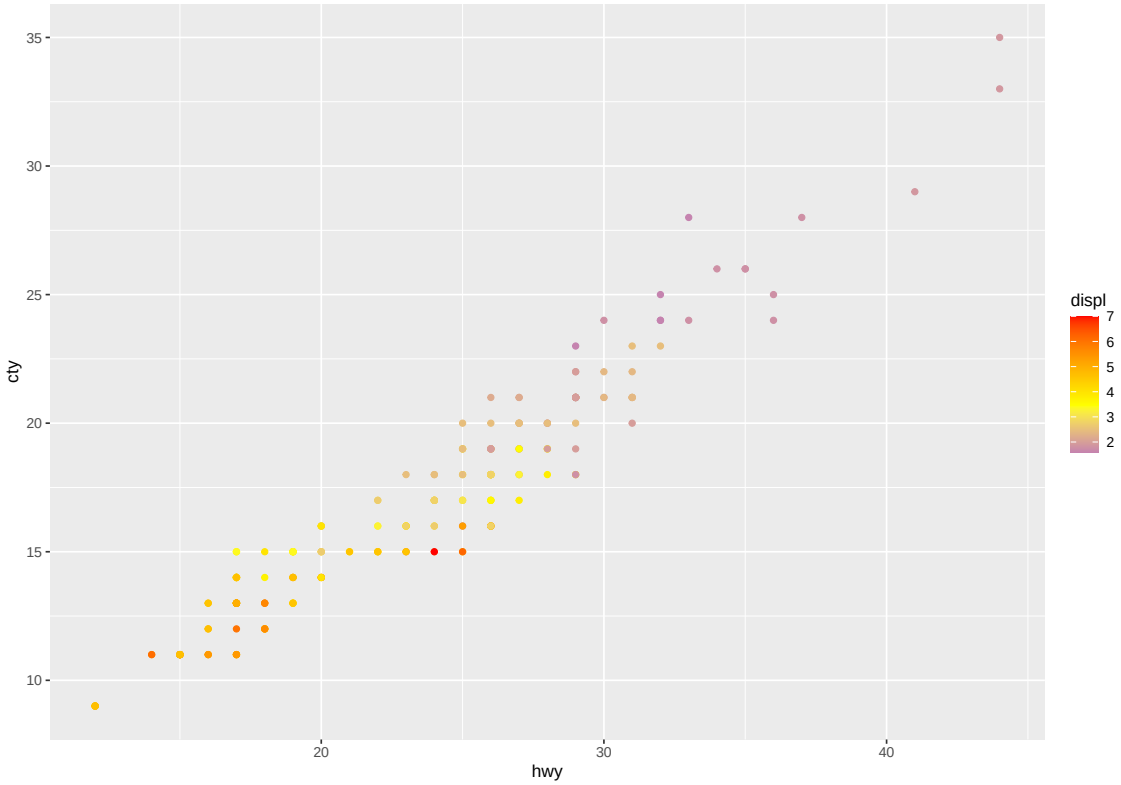
```
# Change the low and high colors
# Sequential color scheme
ggplot(mpg, aes(x=hwy, y=cty, color=displ)) +
  geom_point() +
  scale_color_gradient(low="blue", high="red")
```



Colors

- Continuous colors

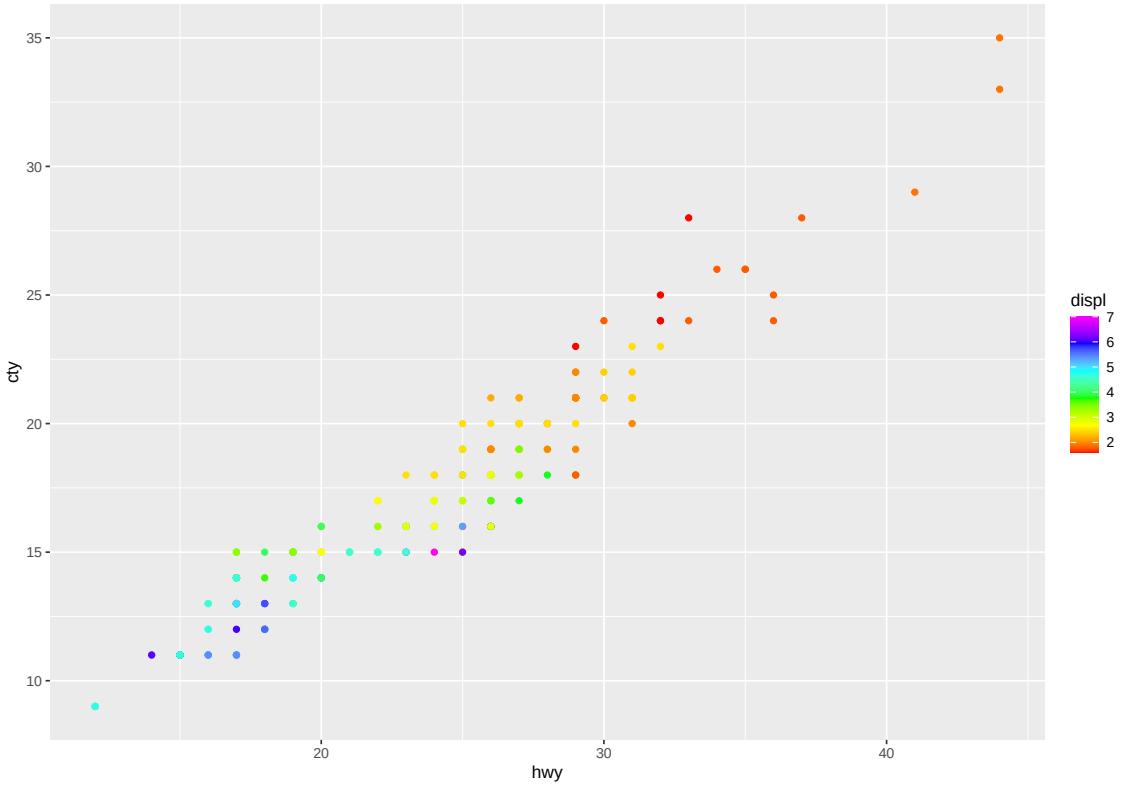
```
# Diverging color scheme
mid=mean(displ)  #average value of displ
ggplot(mpg, aes(x=hwy, y=cty, color=displ)) +
  geom_point() +
  scale_color_gradient2(midpoint=mid, low="blue",
  mid="yellow", high="red", space ="Lab")
```



Colors

- Gradient between n colors

```
# Gradient between n colors
ggplot(mpg, aes(x=hwy, y=cty, color=displ)) +
  geom_point() +
scale_color_gradientn(colours = rainbow(6))
```



Colors: which is better?

```
str(diamonds)
```

```
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2
##  $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9
##  $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338
##  $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07
##  $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11
##  $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53
```

Colors: which is better?

- Sequential

```
set.seed(10)
library(dplyr) #discuss the package later
```

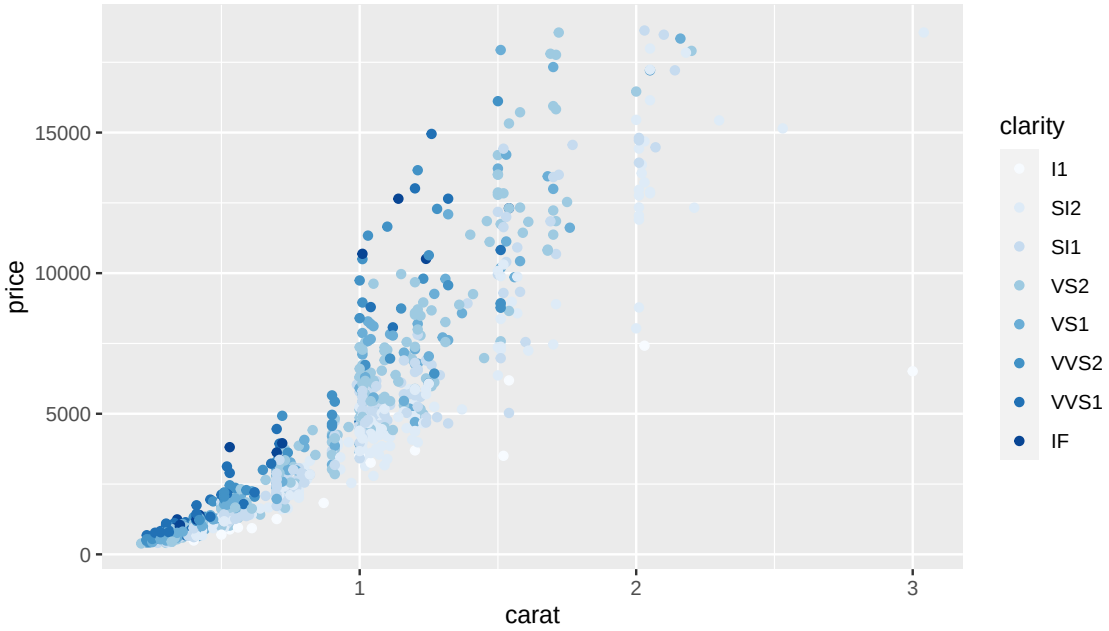
```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
diamonds %>% sample_n(1000)%>%
  ggplot( aes(carat, price)) +
   geom_point(aes(color = clarity)) -> d;
d;
```

Colors: which is better?

- scale_colour_brewer(): Sequential

```
d + scale_colour_brewer()
```



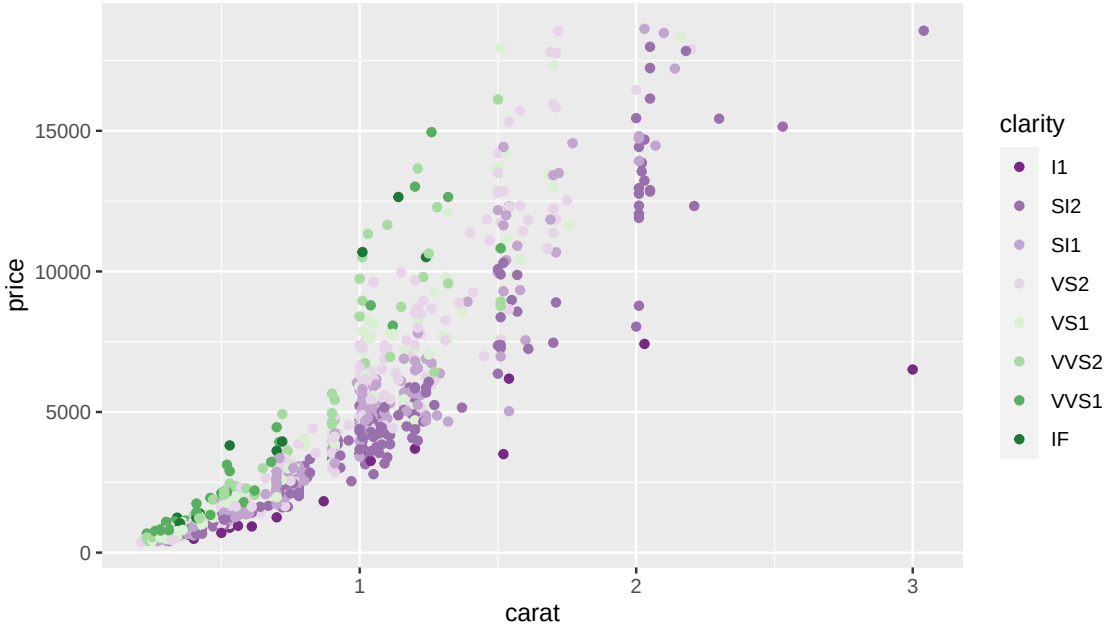

```
#order by shade of blue colors; we may focus on the best diamond
```

Default brewer sequential scale, blues. Focus is on the dark blue.

Colors: which is better?

- scale_colour_brewer(): Diverging

```
d + scale_colour_brewer(palette="PRGn")
```



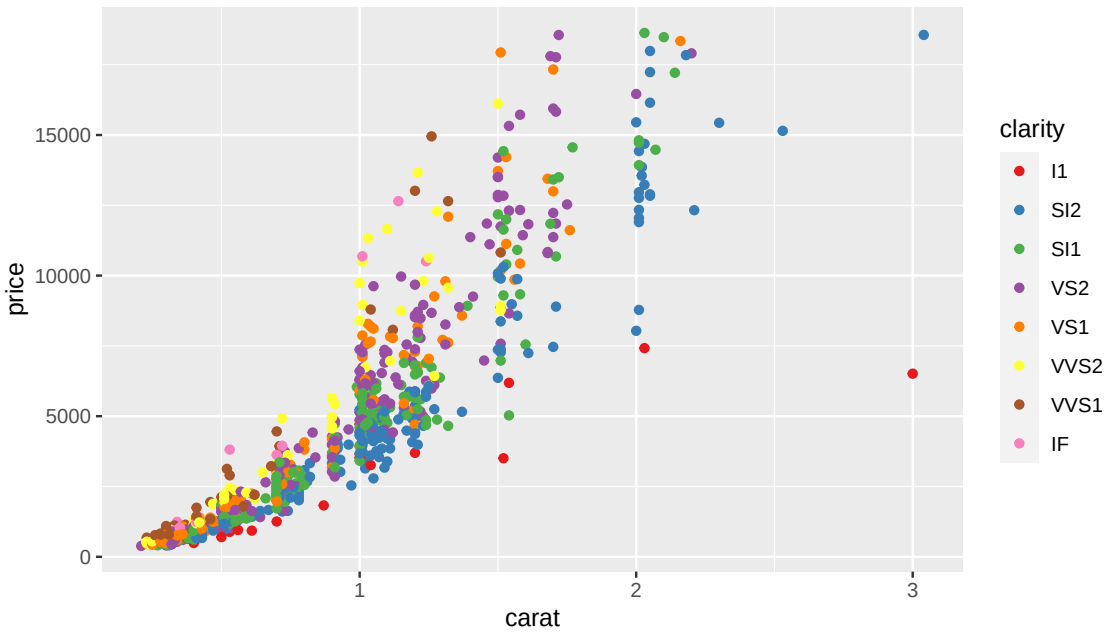

```
# extedn in two directions; color-blindness
```

Map quantitative variable into color scheme that emphasizes both ends, either high AND low, de-emphasizing middle.

Colors: which is better?

- scale_colour_brewer(): Qualitative

```
d + scale_colour_brewer(palette="Set1")
```



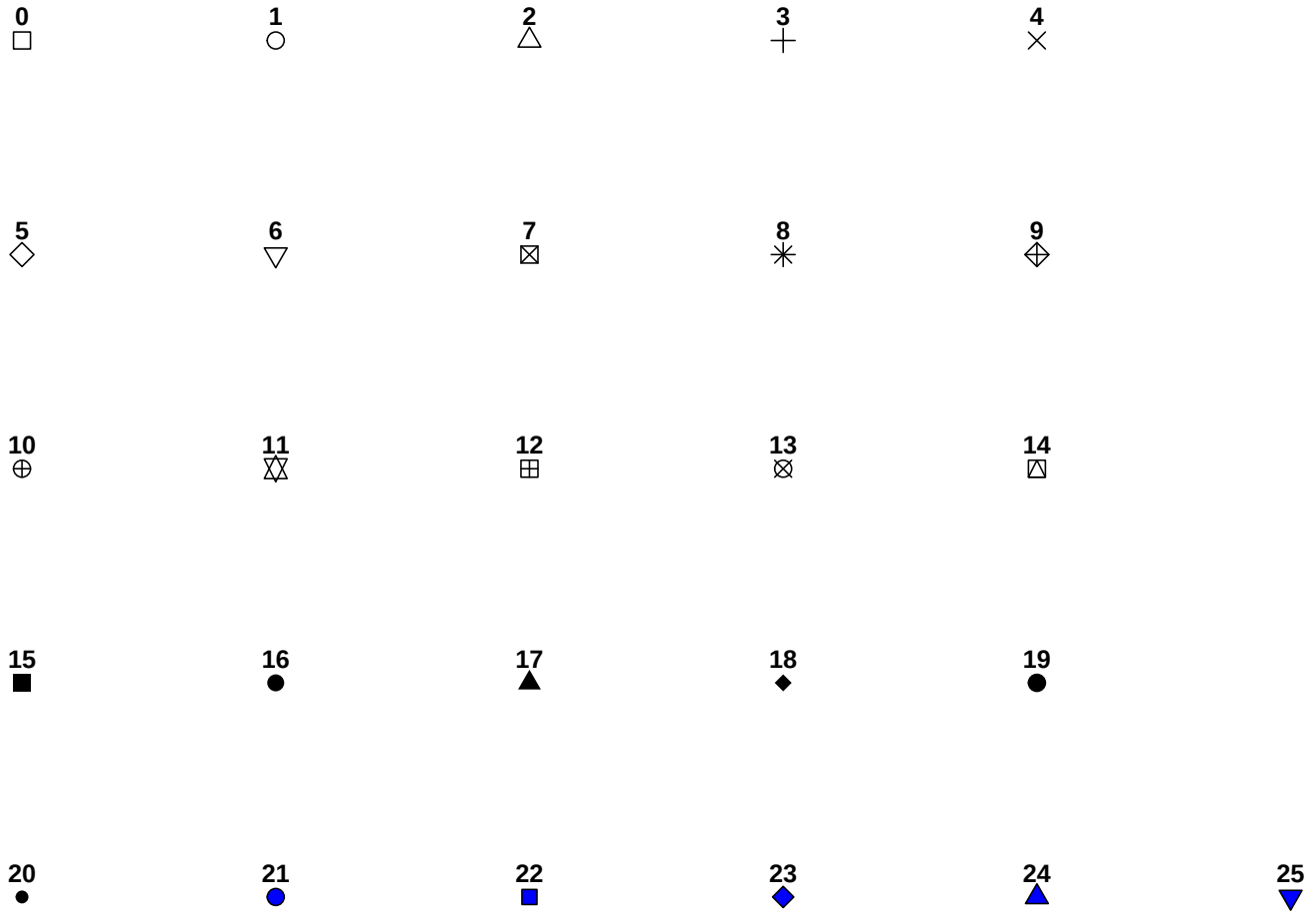

```
# it does not emphasize any group
```

Map quantitative variable into color scheme to most differentiated set. It's possible to have many colors to perceive the differences.

Points

- Recall: points shapes available in R:

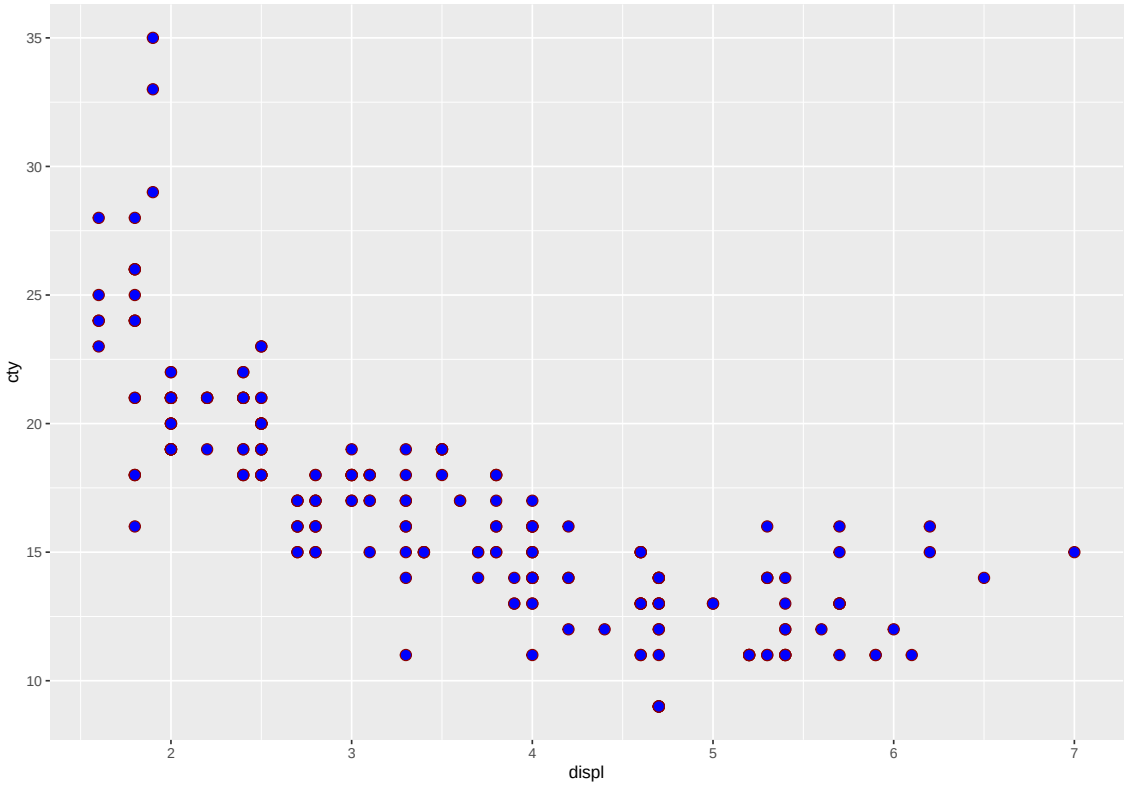
```
?points
```



Points

- Change the point shapes, colors and sizes automatically

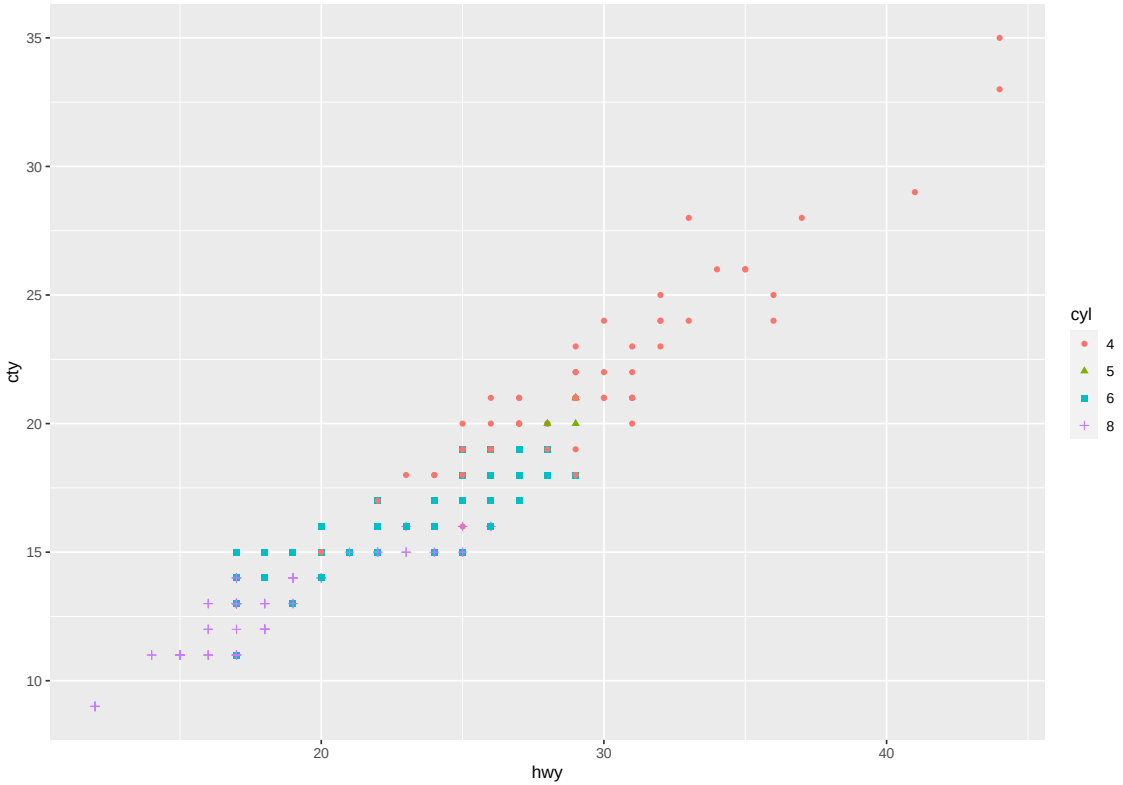
```
ggplot(mpg, aes(x=displ, y=cty)) +
  geom_point(shape=21, fill="blue", color="darkred", size=3)
```



Points

- Change the point shapes, colors and sizes automatically

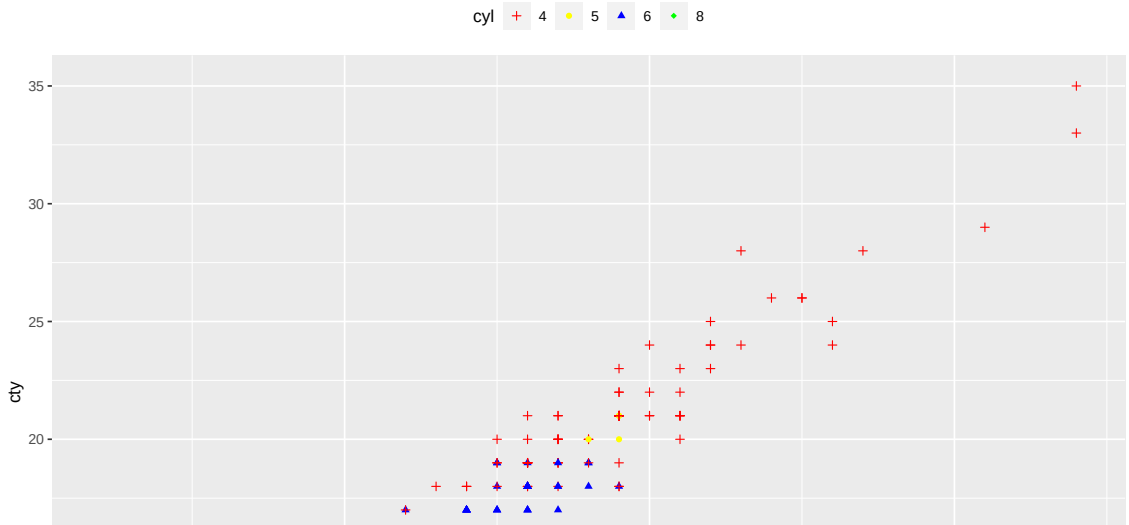
```
ggplot(mpg, aes(x=hwy, y=cty, group=cyl)) +
  geom_point(aes(shape=cyl, color=cyl))
```



Points

- Change point shapes, colors and sizes manually. The functions below can be used:
  - scale_shape_manual() : to change point shapes
  - scale_color_manual() : to change point colors
  - scale_size_manual() : to change the size of points

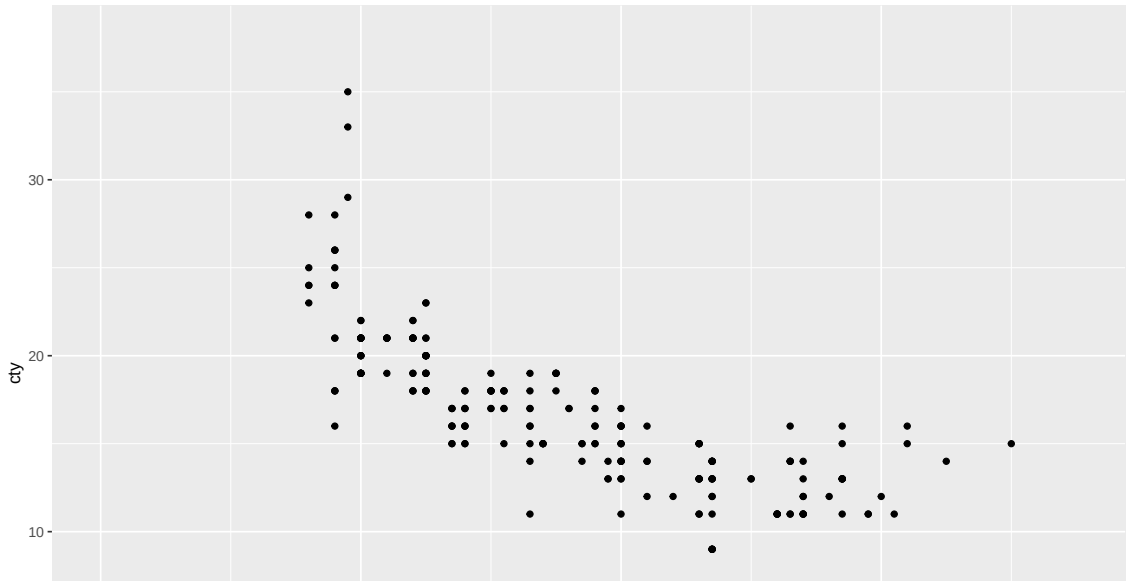
```
ggplot(mpg, aes(x=hwy, y=cty, group=cyl)) +
  geom_point(aes(shape=cyl, color=cyl)) +
  scale_shape_manual(values=c(3, 16, 17, 18))+
  scale_color_manual(values=c("red", "yellow","blue", "green"))+
  scale_size_manual(values=c(2,3,4,5))+
  theme(legend.position="top")
```



Axis scales

- Change x and y axis limits. There are different functions to set axis limits:
  - xlim() and ylim()
  - expand_limits()
  - scale_x_continuous() and scale_y_continuous()
- Use xlim() and ylim() functions
  - xlim(min, max) # x axis limits
  - ylim(min, max) # y axis limits

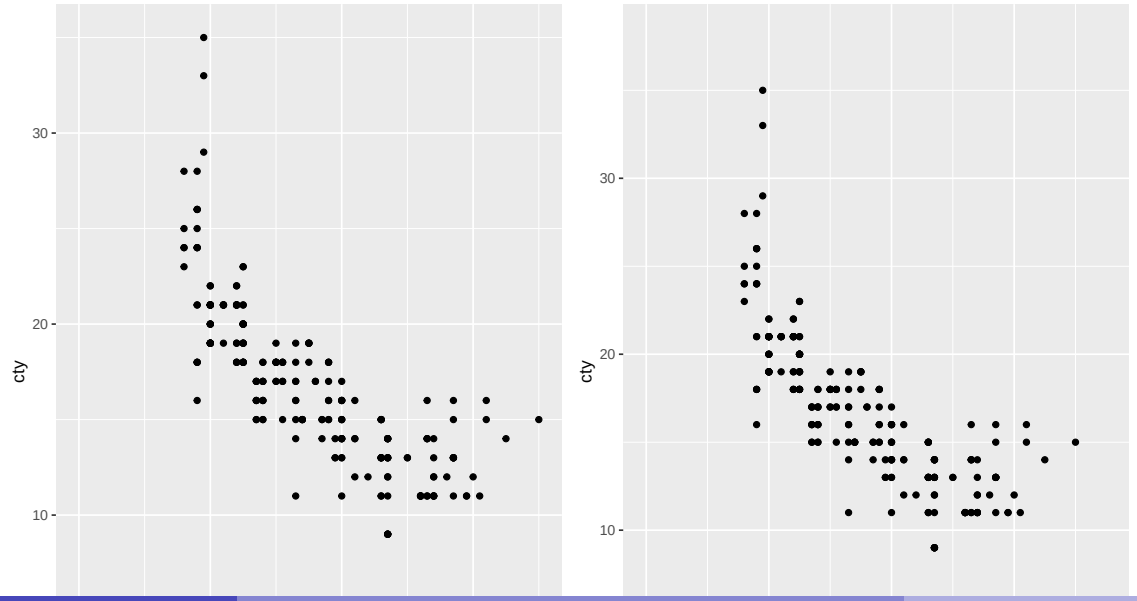
```
ggplot(mpg, aes(x=displ, y=cty)) + geom_point() +
  xlim(0, 7.5)+ylim(0, 38)
```



Axis scales

- Change x and y axis limits using `expand_limts()` function. The function `expand_limits()` can be used to:
  - quickly set the intercept of x and y axes at (0,0)
  - change the limits of x and y axes

```
library(gridExtra)
p1=ggplot(mpg, aes(x=displ, y=cty)) + geom_point() +
  expand_limits(x=0, y=0)
p2=ggplot(mpg, aes(x=displ, y=cty)) + geom_point() +
  expand_limits(x=c(0,7.5), y=c(0, 38))
grid.arrange(p1, p2, ncol=2)
```

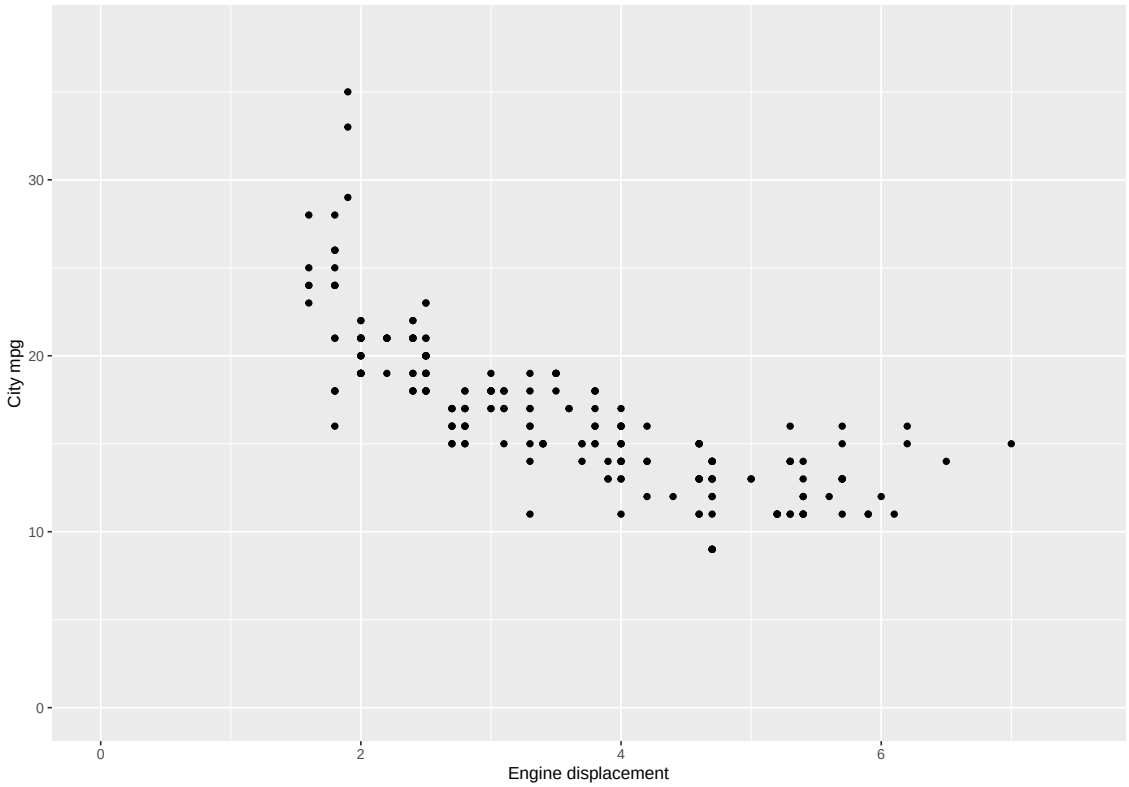


Axis scales

- It is also possible to use the functions `scale_x_continuous()` and `scale_y_continuous()` to change x and y axis limits, respectively.
  - scale_x_continuous(name, breaks, labels, limits, trans)
  - scale_y_continuous(name, breaks, labels, limits, trans)
- The arguments are
  - name : x or y axis labels
  - breaks : to control the breaks in the guide (axis ticks, grid lines, .). Among the possible values, there are :
    - NULL : hide all breaks
    - waiver() : the default break computation
    - a character or numeric vector specifying the breaks to display
  - labels : labels of axis tick marks. Allowed values are :
    - NULL for no labels
    - waiver() for the default labels
    - character vector to be used for break labels
  - limits : a numeric vector specifying x or y axis limits (min, max)
  - trans for axis transformations. Possible values are "log2", "log10", . . .

Axis scales

```
ggplot(mpg, aes(x=displ, y=cty)) + geom_point() +
scale_x_continuous(name="Engine displacement", limits=c(0, 7.5)) +
scale_y_continuous(name="City mpg", limits=c(0, 38))
```



Resources

- ggplot2 Reference manual
- ggplot2: Elegant Graphics for Data Analysis, Hadley Wickham
- ggplot2 Function reference
- Developer's github
- R Graphics Cookbook, Winston Chang
- How I build up a ggplot2 figure?
- To get more help about controlling your ggplot2 plot, see the ggplot2 reference.

License