Data Science

Data Visualization and Transformation

Habet Madoyan

American University of Armenia

Section 1

Grammar of graphics

Grammar of graphics

There are few definitions of what grammar is

“the whole system and structure of a language or of languages in general, usually taken as consisting of syntax and morphology (including inflections) and sometimes also phonology and semantics”

grammar is “the fundamental principles or rules of an art or science”

Grammar of graphics

Grammar gives language rules. The word stems from the Greek noun for letter or mark. And that derives from the Greek verb for writing, which is the source of English word graph. Grammar means, more generally, rules for art and science, as in the richly illustrated The Grammar of Ornament (Jones, 1856), and Karl Pearson's The Grammar of Science (Pearson, 1892)

(Leland Wilkinson)

Grammar of graphics

Grammar has a technical meaning in linguistics. In the transformational theory of Chomsky (1956), a grammar is a formal system of rules for generating lawful statements in a language.

(Leland Wilkinson)

Grammar of graphics

Grammar makes language expressive. A language consisting of words and no grammar (statement = word) expresses only as many ideas as there are words. By specifying how words are combined in statements, a grammar expands a language's scope.

Grammar of graphics

(Leland Wilkinson)

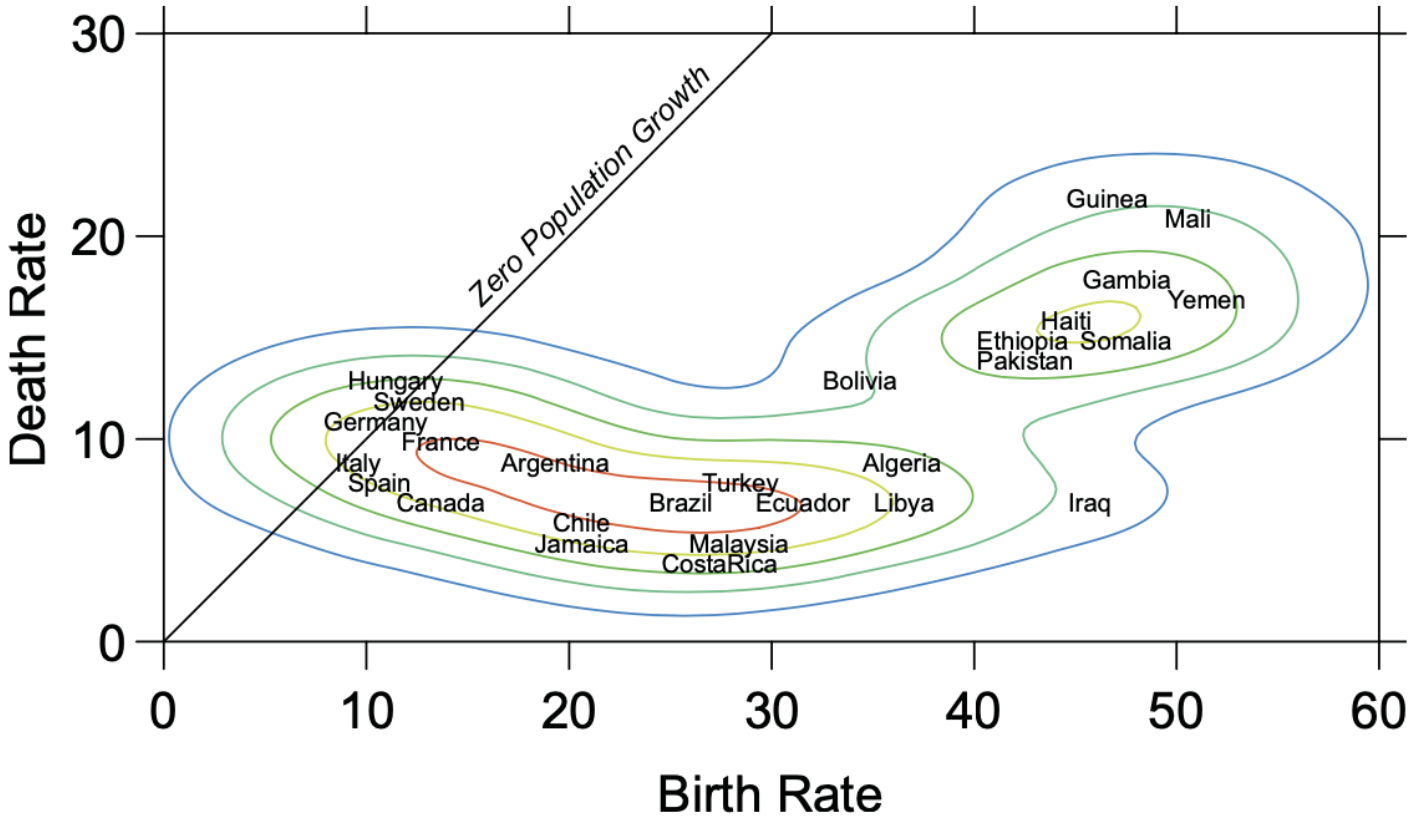


Figure 1.1 *Plot of death rates against birth rates for selected countries*

Grammar of graphics

- ELEMENT: point(position(birth*death), size(0), label(country))
- ELEMENT: contour(position(smooth.density.kernel.epanechnikov.joint(birth*death)), color.hue())
- GUIDE: form.line(position((0,0),(30,30)), label("Zero Population Growth"))
- GUIDE: axis(dim(1), label("Birth Rate"))
- GUIDE: axis(dim(2), label("Death Rate"))

(Leland Wilkinson)

Grammar of graphics

What are some other elements and guides that you see on the graph ?

Grammar of graphics

ggplot2

ggplot2 is an R package for producing statistical, or data, graphics, but it is unlike most other graphics packages because it has a deep underlying grammar. This grammar, based on the Grammar of Graphics (Wilkinson, 2005), is composed of a set of independent components that can be composed in many different ways. This makes ggplot2 very powerful, because you are not limited to a set of pre-specified graphics, but you can create new graphics that are precisely tailored for your problem. This may sound overwhelming, but because there is a simple set of core principles and very few special cases, ggplot2 is also easy to learn (although it may take a little time to forget your preconceptions from other graphics tools).

(Hadley Wickham)

Grammar of graphics

- Learning the grammar will help you not only create graphics that you know about now, but will also help you to think about new graphics that would be even better.
- Without the grammar, there is no underlying theory and existing graphics packages are just a big collection of special cases.

Grammar of graphics

Layered grammar of graphics

ggplot2 is designed to work in a layered fashion, starting with a layer showing the raw data then adding layers of annotations and statistical summaries.

ggplot2

In ggplot the graph building process always starts with **ggplot()** layer

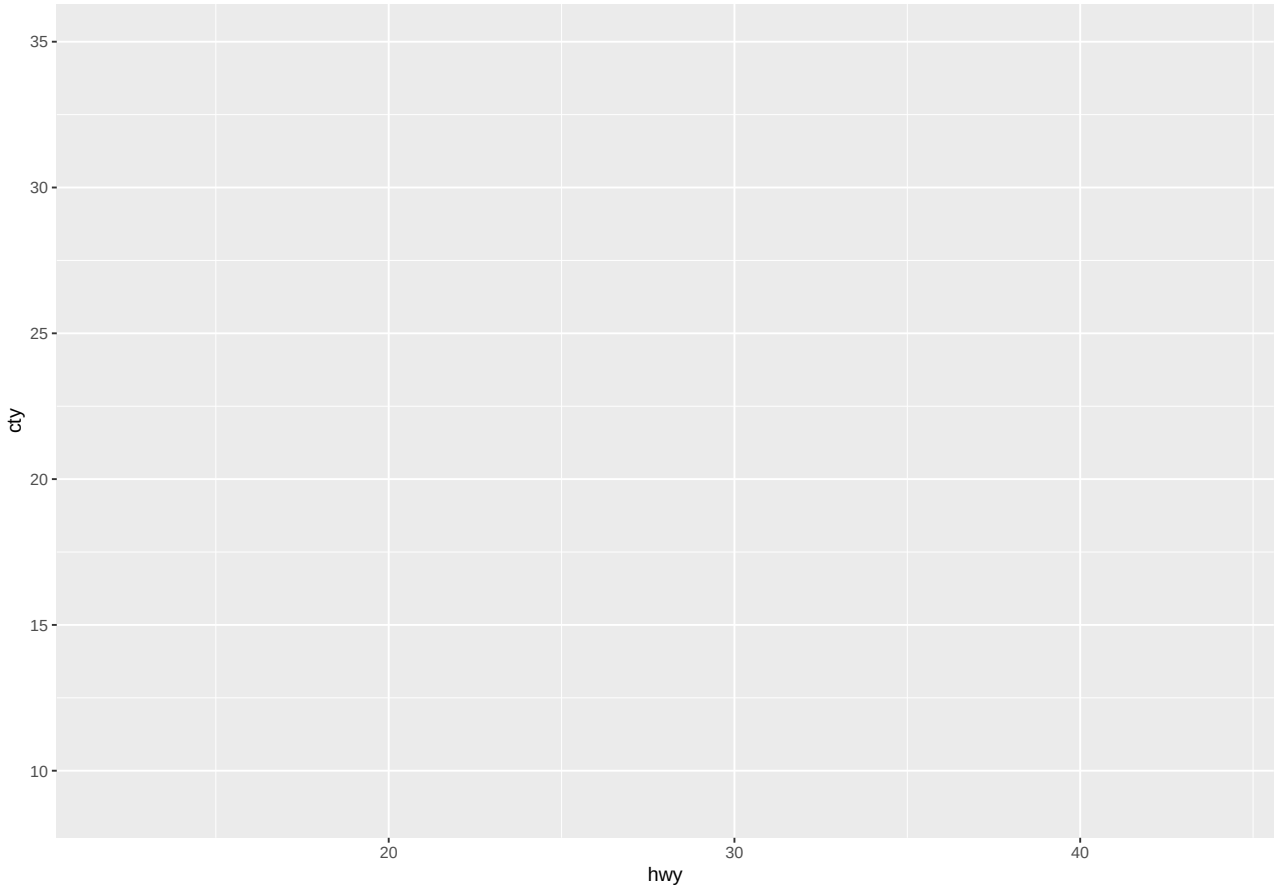
Get the data mpg from the library ggplot2

```
data(mpg)
```

ggplot2

The result is an empty space

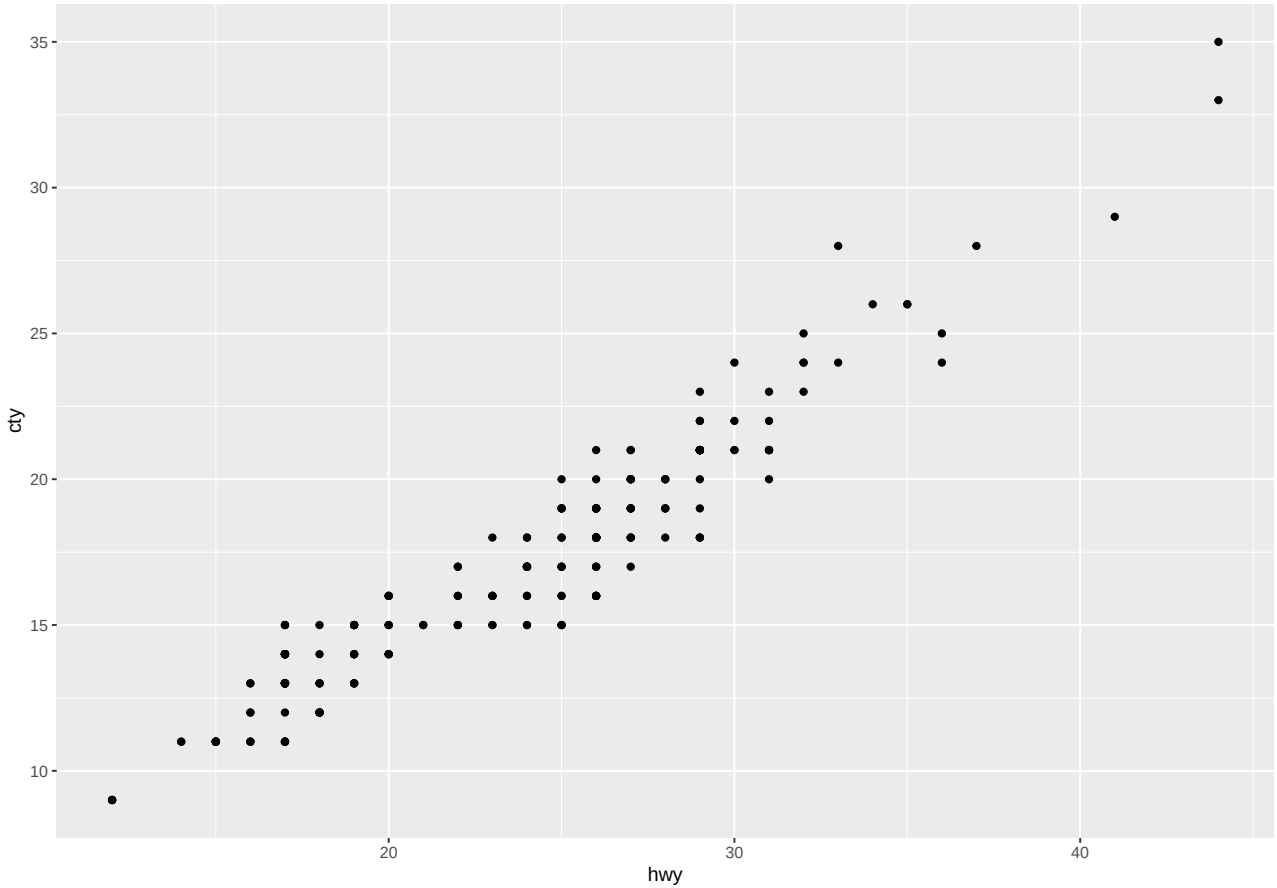
```
ggplot(data = mpg, aes(x = hwy, y = cty))
```



ggplot2

If you want to have a chart, add additional *layer*

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



ggplot2

Elements of grammar of graphics

- The **data** that you want to visualize and a set of aesthetic mappings describing how variables in the data are mapped to aesthetic attributes that you can perceive.
- **Geometric objects**, geoms for short, represent what you actually see on the plot: points, lines, polygons, etc.
- **Statistical transformations**, stats for short, summarise data in many useful ways. For example, binning and counting observations to create a histogram, or summarising a 2d relationship with a linear model. Stats are optional, but very useful.
- The **scales** map values in the data space to values in an aesthetic space, whether it be colour, or size, or shape. Scales draw a legend or axes, which provide an inverse mapping to make it possible to read the original data values from the graph.

ggplot2 (continued)

- A **coordinate system**, coord for short, describes how data coordinates are mapped to the plane of the graphic. It also provides axes and gridlines to make it possible to read the graph. We normally use a Cartesian coordinate system, but a number of others are available, including polar coordinates and map projections.
- A **faceting** specification describes how to break up the data into subsets and how to display those subsets as small multiples. This is also known as conditioning or latticing/trellising.

ggplot2

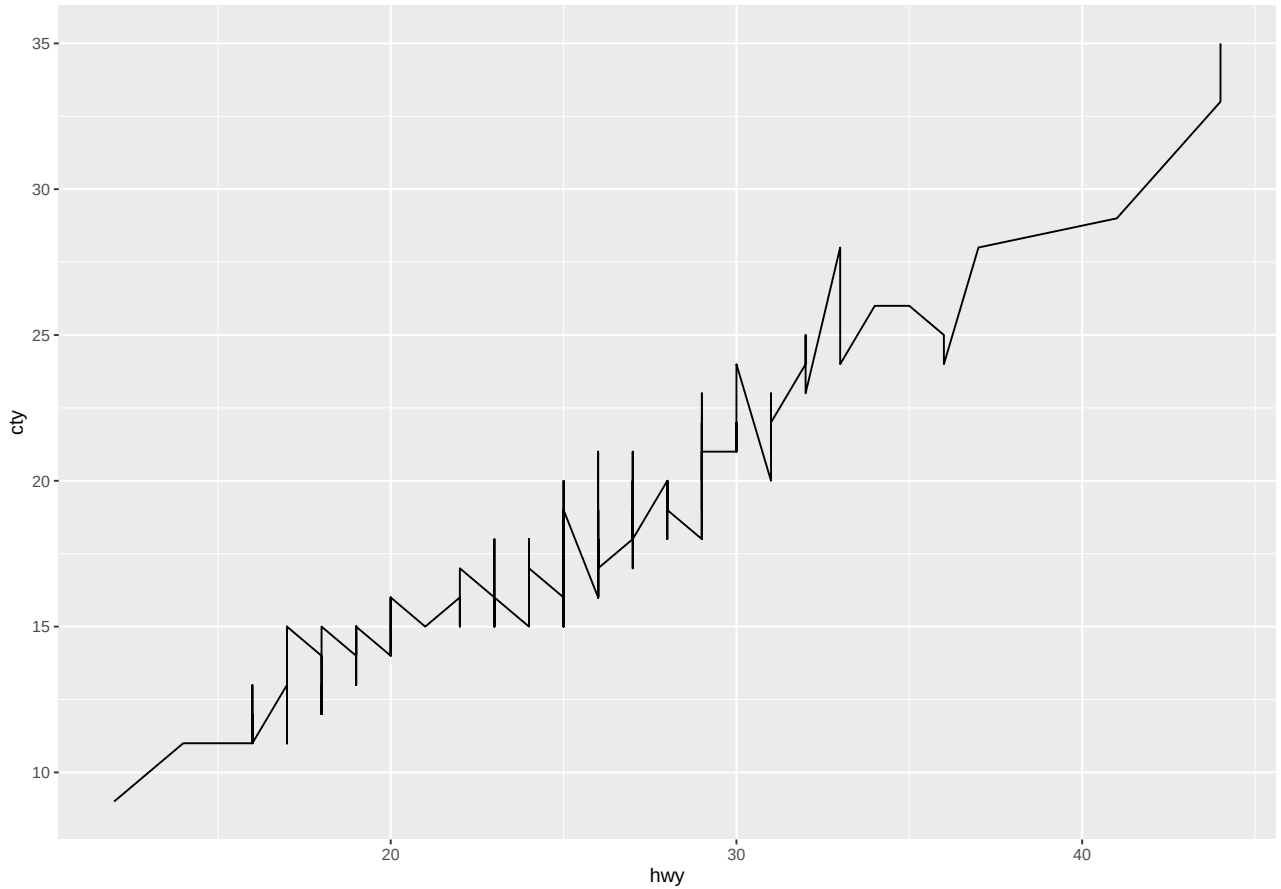
ggplot2 uses shortcuts for the elements

- geom_XXX - for geometric objects
- stat_XXX - for statistical transformations
- scale_ - for scales
- coord_XXX for coordinate systems
- facet_ - for faceting

ggplot2

If you want to see what are the geometric objects available as a layer for ggplot, just type geom_ .
line chart

```
ggplot(data = mpg, aes(x = hwy, y = cty)) + geom_line()
```



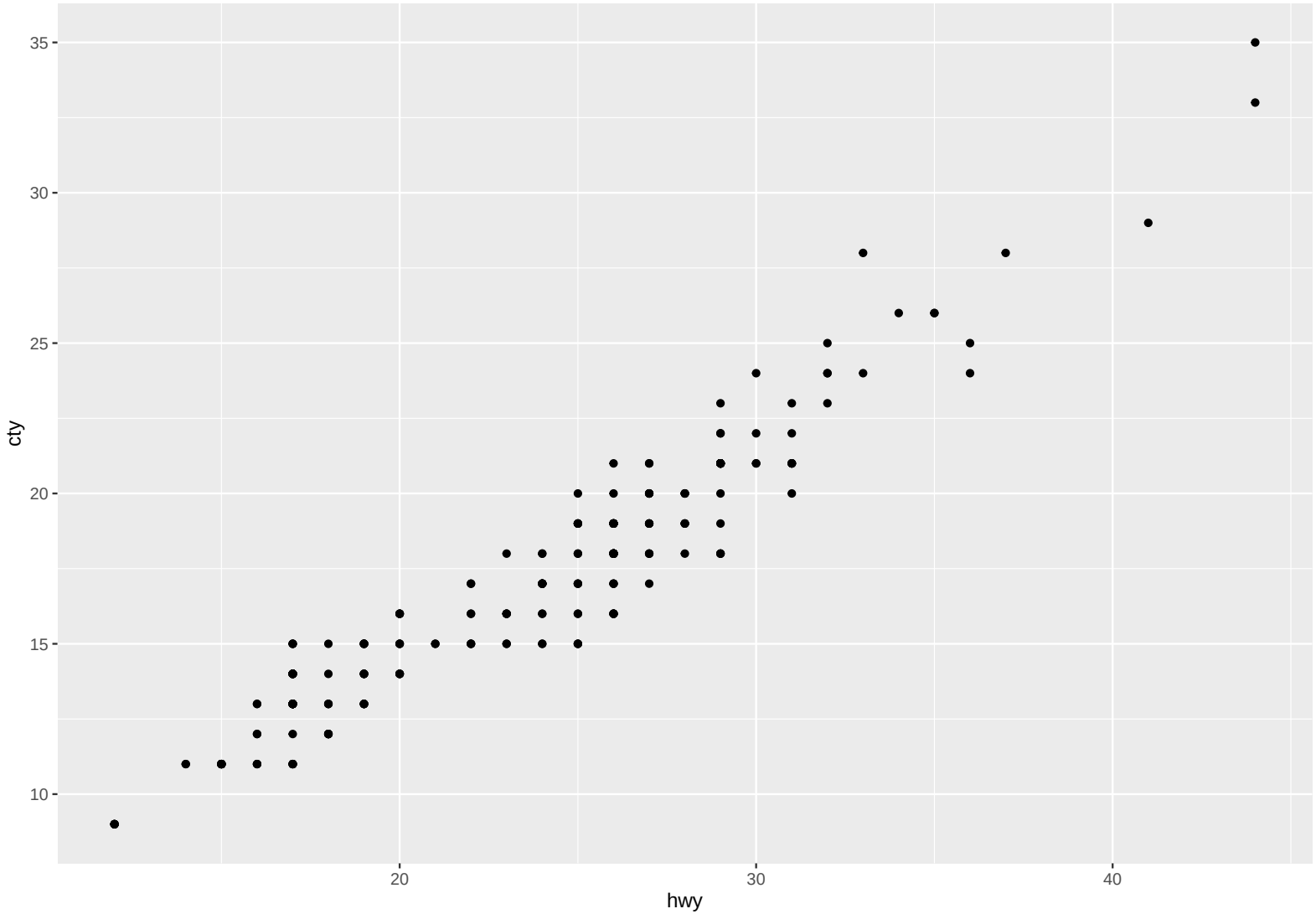
ggplot2

Aesthetics

- Aesthetics, in the original Greek sense, offers principles for relating sensory attributes to abstractions.
- In ggplot and data visualization in general, aesthetics map data to the plot. In other words, whatever can be perceived on the graph, is aesthetics.

ggplot2

What are the aesthetics on the scatterplot ?

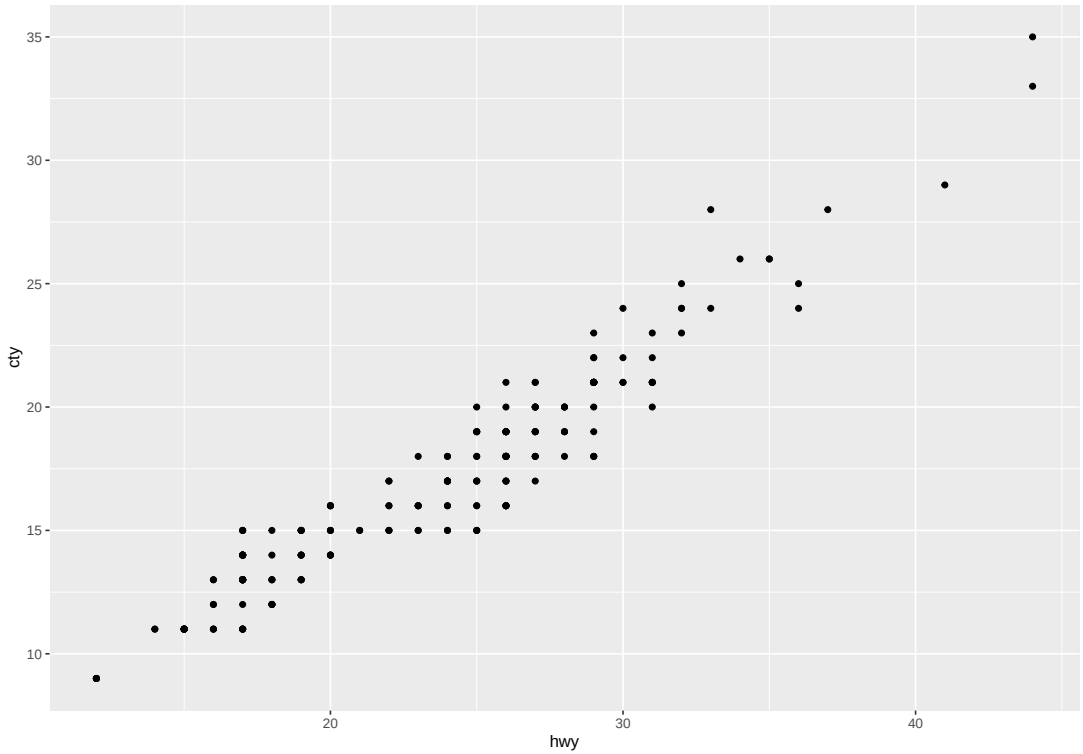


ggplot2

If you define data and aesthetics in ggplot(), other layers are going to inherit them
ggplot(data = mpg, mapping = aes(x = hwy, y = cty)) + geom_point()

However you can define data and aesthetics in the layer as well

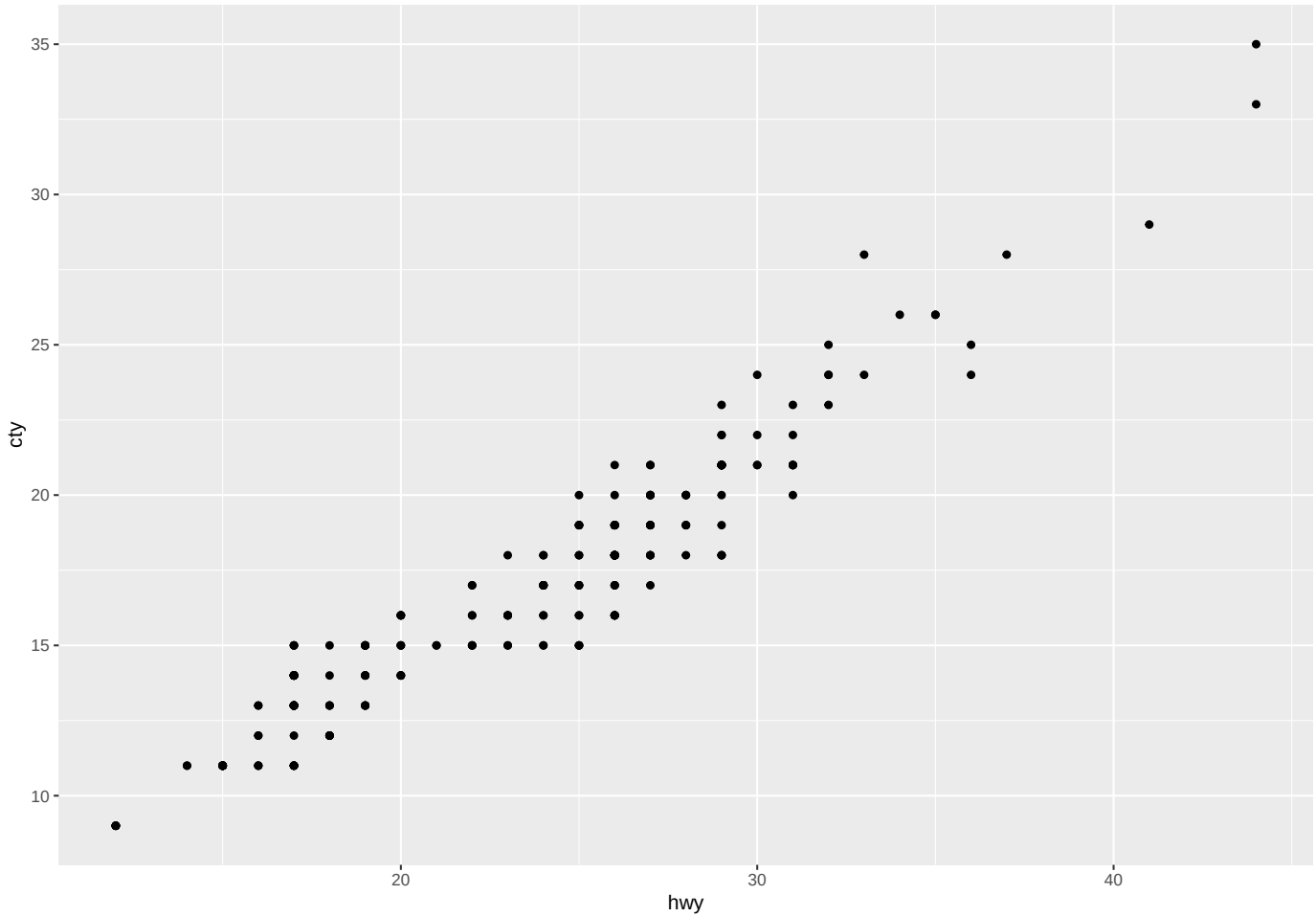
```
ggplot() + geom_point(data = mpg, mapping = aes(x = hwy, y = cty))
```



ggplot2

The difference becomes apparent when you add another layer

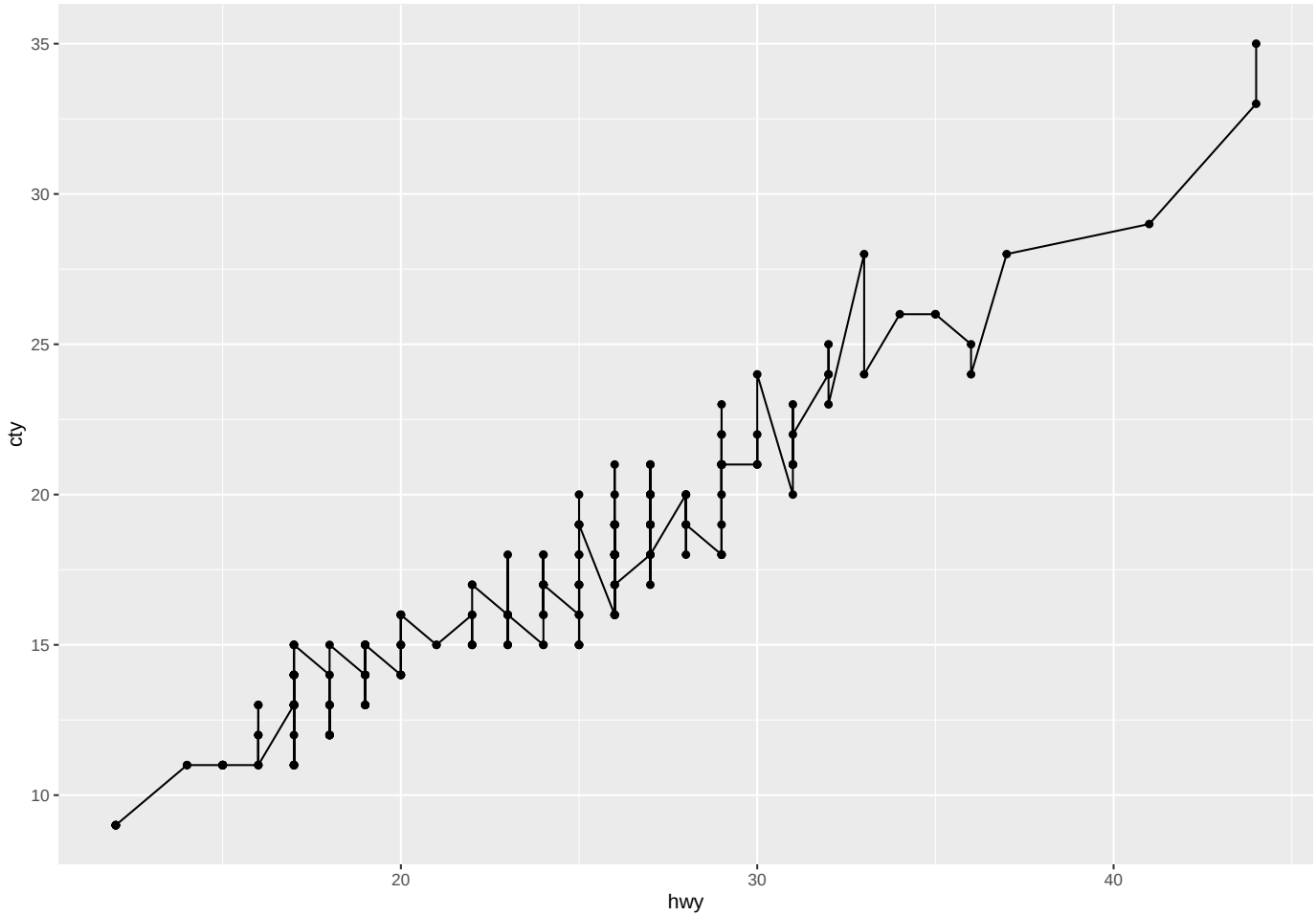
```
ggplot() + geom_point(data = mpg, mapping = aes(x = hwy, y = cty)) +
  geom_line()
```



ggplot2

Now you need to specify data and aesthetics in each layer

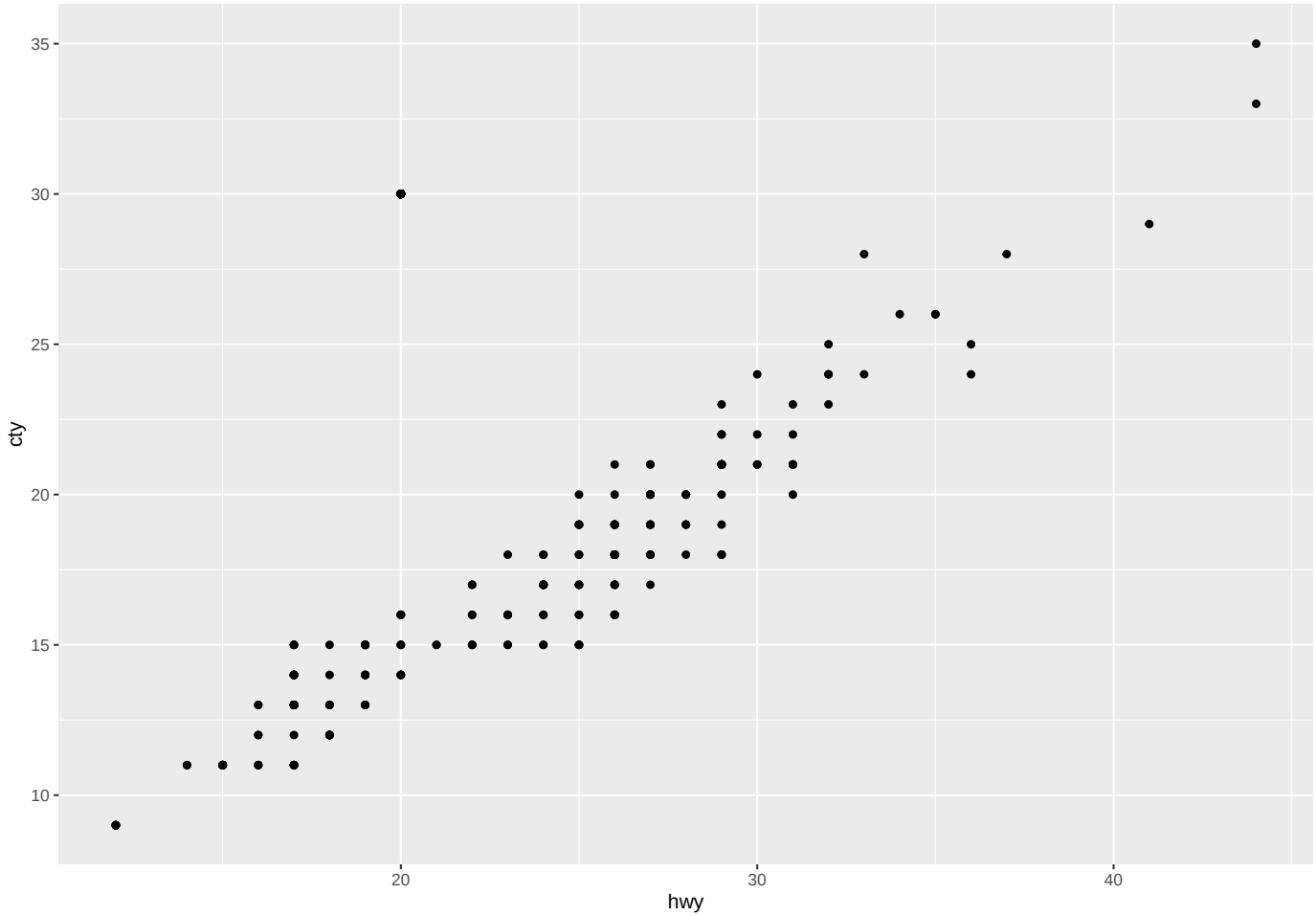
```
ggplot() + geom_point(data = mpg, mapping = aes(x = hwy, y = cty)) +
  geom_line(data = mpg, mapping = aes(x = hwy, y = cty))
```



ggplot2

Add additional point with a new layer

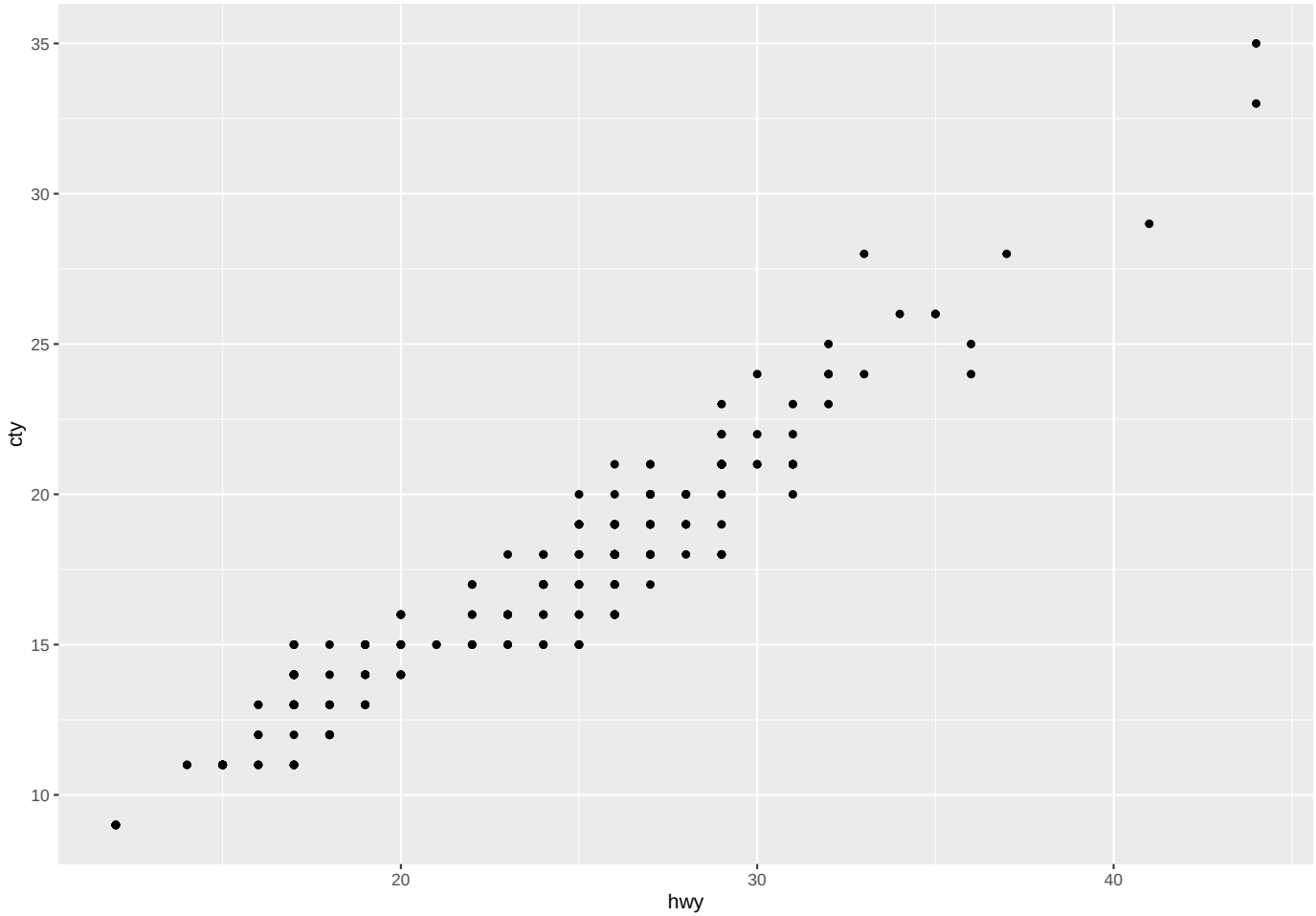
```
ggplot(data = mpg, mapping = aes(x = hwy, y = cty)) +
  geom_point() + geom_point(mapping = aes(x = 20, y = 30))
```



ggplot2

Actually you can save ggplot output and then add layers to it

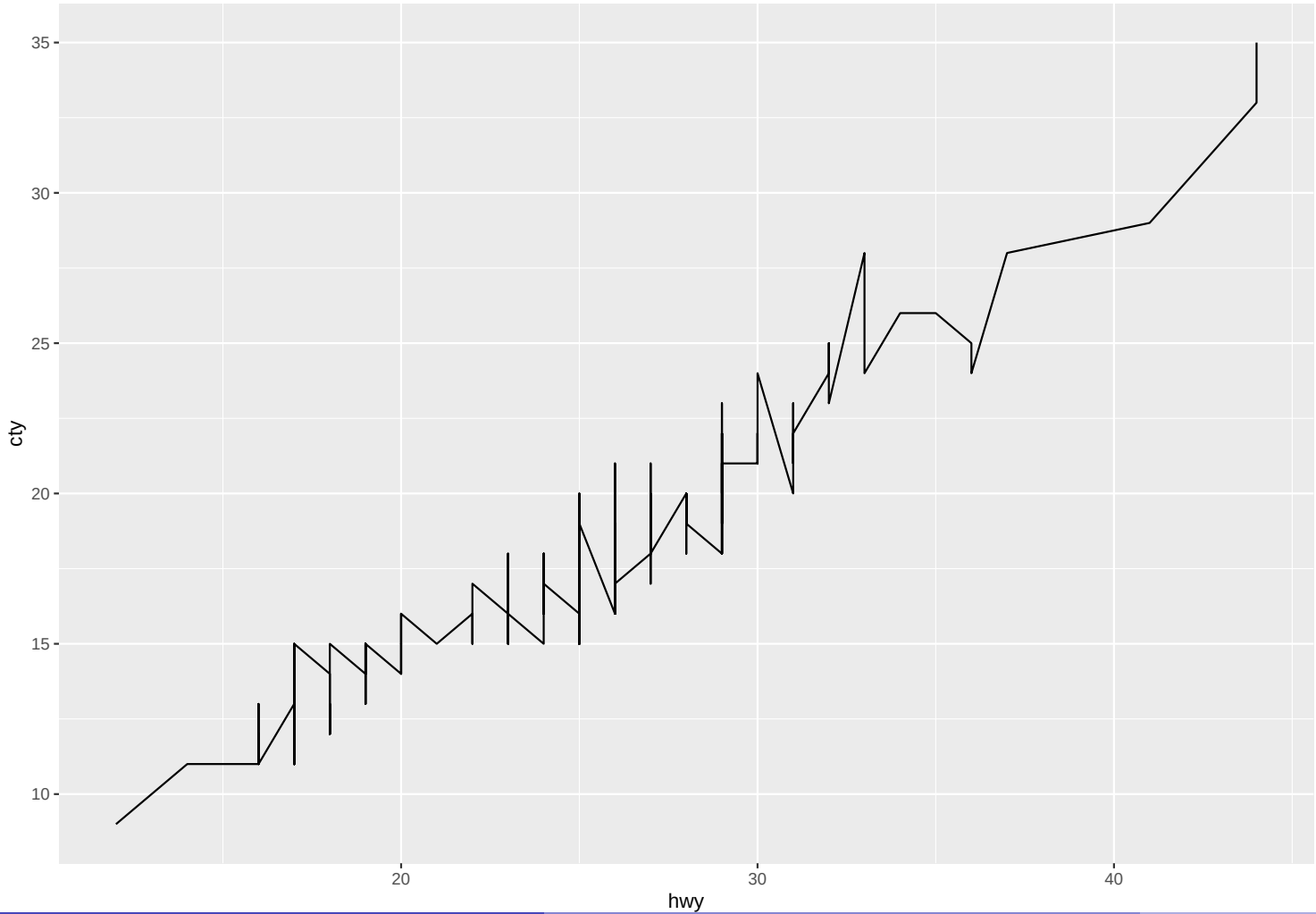
```
p <- ggplot(data = mpg, mapping = aes(x = hwy, y = cty))
p + geom_point()
```



ggplot2

Line chart

```
p + geom_line()
```



ggplot2

If you are planning to break the line, leave + at the end of the line

Like this

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

Not like this

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

ggplot2

aesthetics

For the scatterplot:

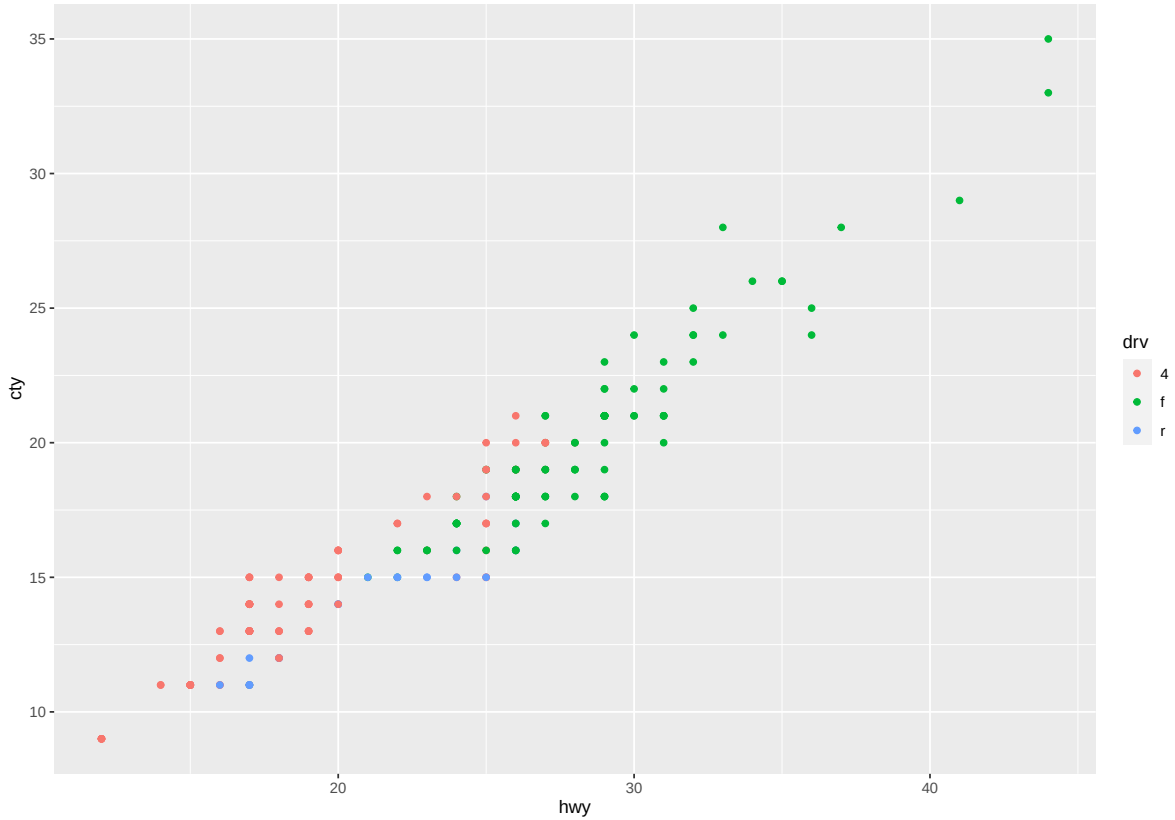
- x coordinate
- y coordinate
- size
- shape
- color

ggplot2

The aesthetics can be either a constant, or a variable from the data.

variable

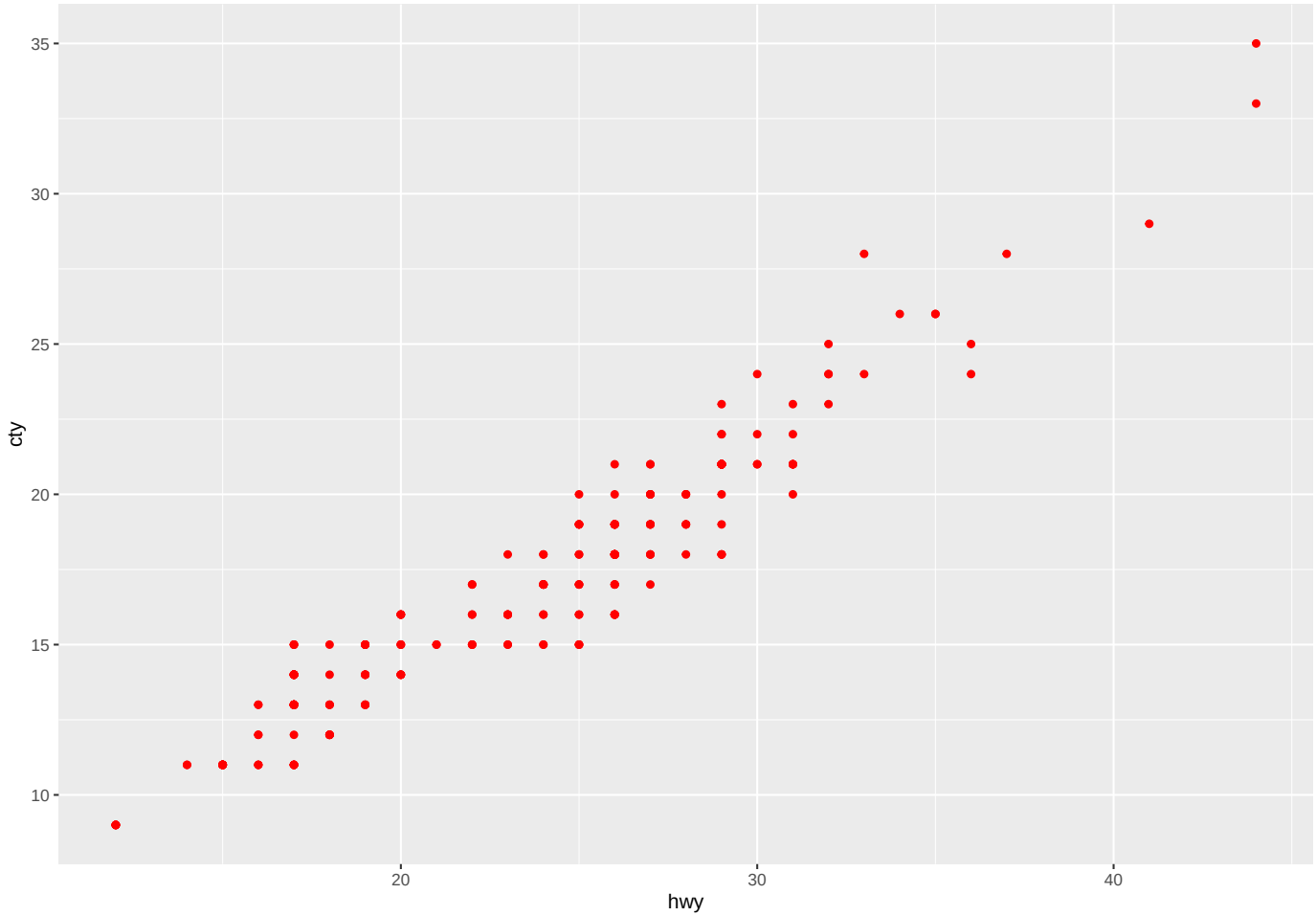
```
ggplot(data = mpg, mapping = aes(x = hwy, y = cty, color = drv)) +
  geom_point()
```



ggplot2

If you want aesthetics as a constant, don't include it in aesthetics

```
ggplot(data = mpg, mapping = aes(x = hwy, y = cty)) +
  geom_point(color = 'red')
```

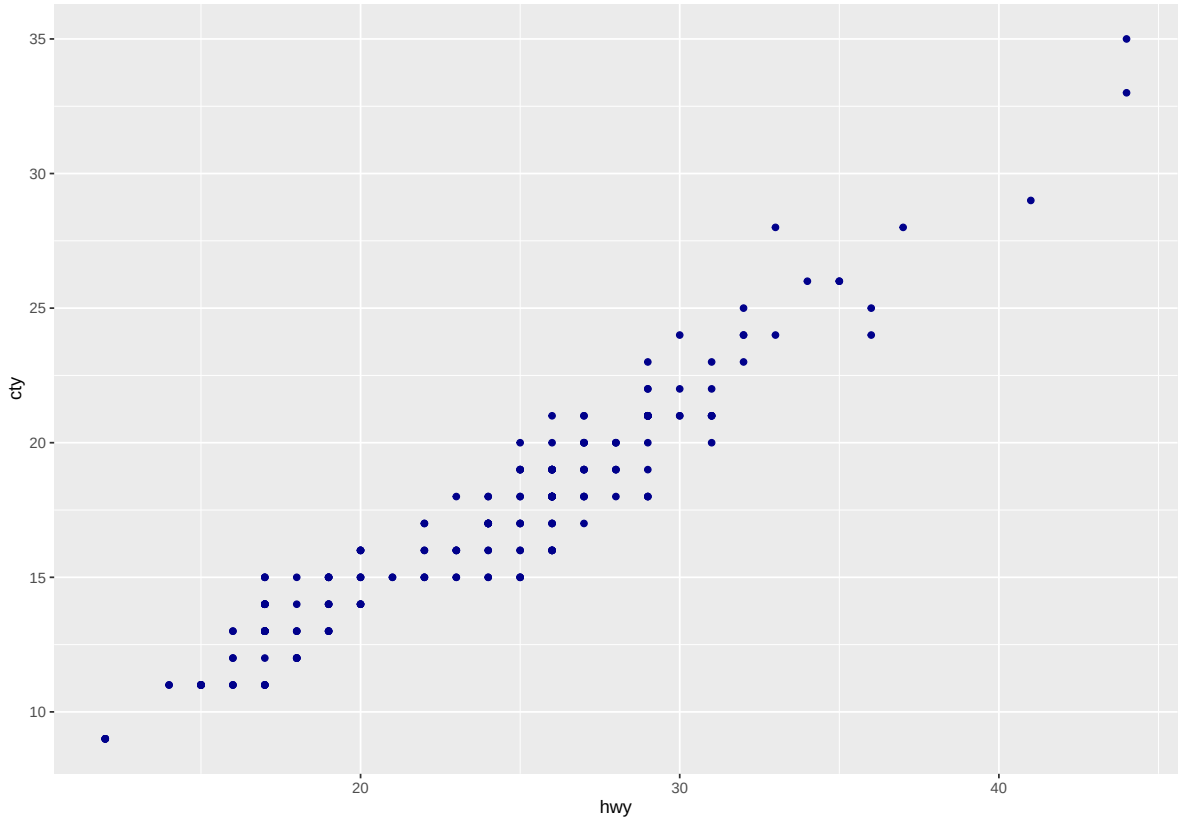


ggplot2

Let's see the difference between *mapping* the aesthetics and *setting* it.

Setting

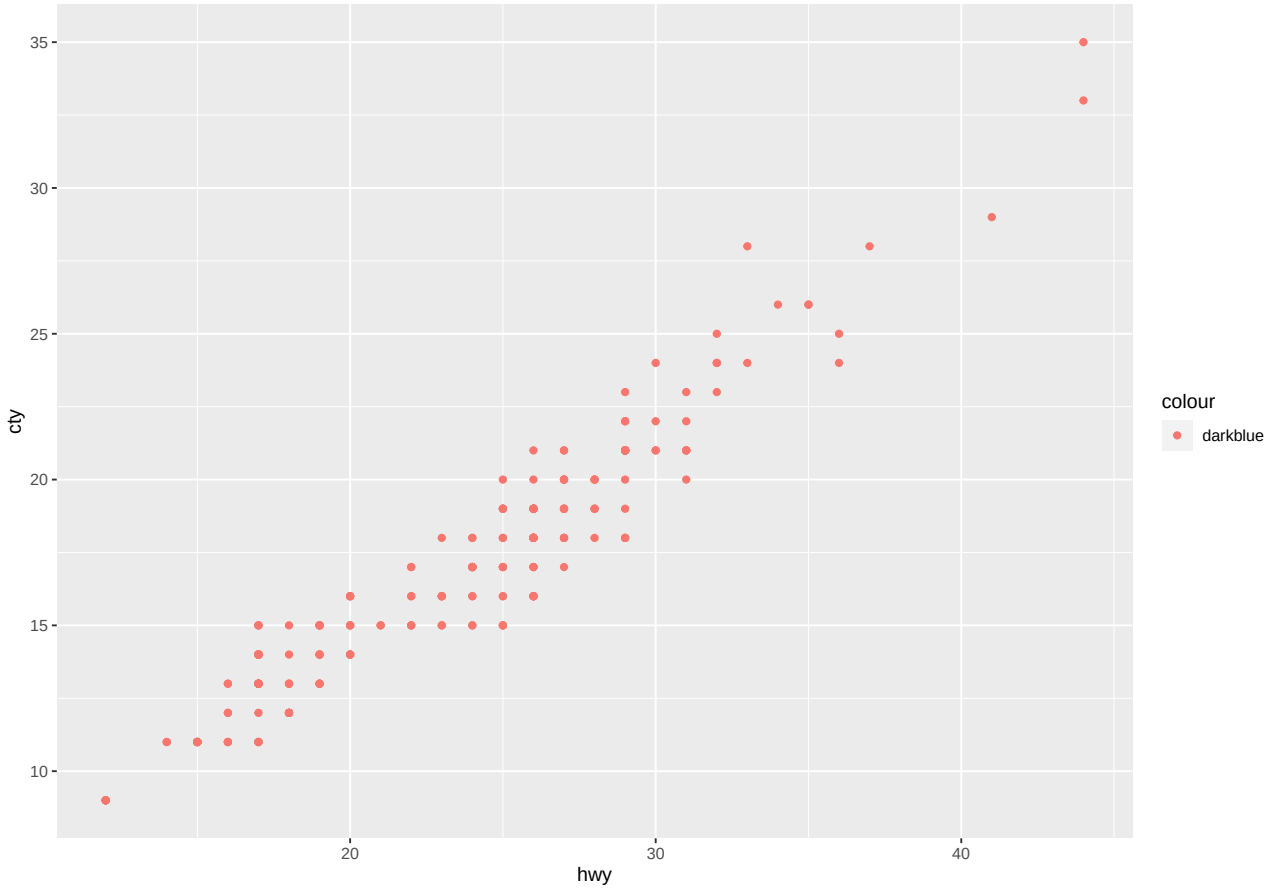
```
(p1 <- ggplot(data = mpg, mapping = aes(x = hwy, y = cty)) +
  geom_point(color = 'darkblue'))
```



ggplot2

Mapping

```
(p2 <- ggplot(data = mpg, mapping = aes(x = hwy, y = cty,
                                        color = 'darkblue')) +
   geom_point())
```



ggplot2

Use the function ggplot_build() to access the data

```
p3 <- ggplot(data = mpg, mapping = aes(x = hwy, y = cty, color = drv)) +
  geom_point()
df_plot <- ggplot_build(p3)$data[[1]]
head(df_plot)
```

```
##    colour  x  y PANEL group shape size fill alpha stroke
## 1 #00BA38 29 18     1     2    19  1.5   NA    NA    0.5
## 2 #00BA38 29 21     1     2    19  1.5   NA    NA    0.5
## 3 #00BA38 31 20     1     2    19  1.5   NA    NA    0.5
## 4 #00BA38 30 21     1     2    19  1.5   NA    NA    0.5
## 5 #00BA38 26 16     1     2    19  1.5   NA    NA    0.5
## 6 #00BA38 26 18     1     2    19  1.5   NA    NA    0.5
```

ggplot2

The variable **color** is added with three values, one for each category.

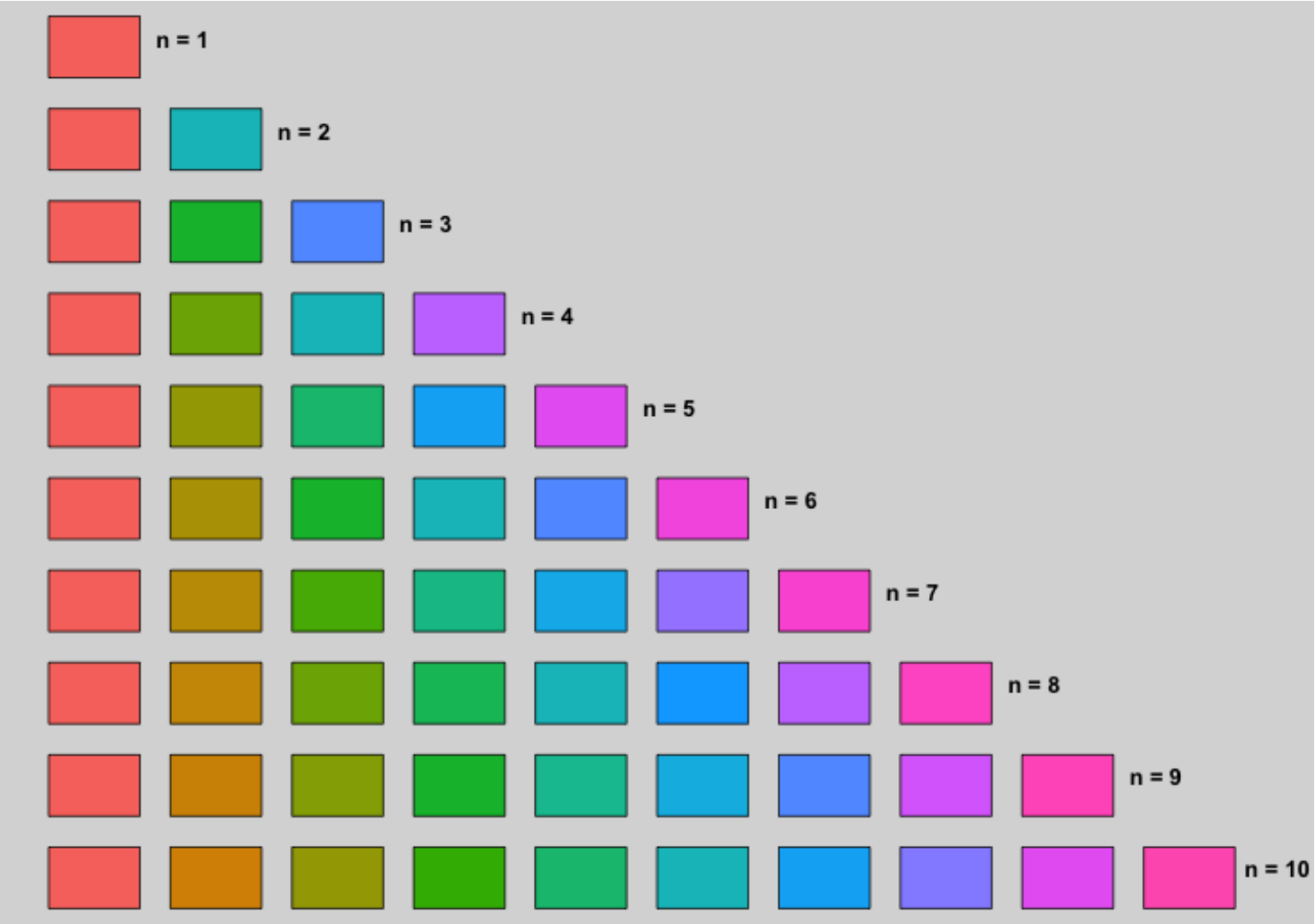
Hex codes for the colors are given

```
table(df_plot$colour)
```

```
##
## #00BA38 #619CFF #F8766D
##     106      25     103
```

ggplot2

Three colors are taken from ggplot's standard color wheel (more about it later)



ggplot2

- p1 - color is set to a constant
- p2 - color is mapped to a constant

```
df_plot1 <- ggplot_build(p1)$data[[1]]
df_plot2 <- ggplot_build(p2)$data[[1]]
unique(df_plot1$colour)
```

```
## [1] "darkblue"
```

```
unique(df_plot2$colour)
```

```
## [1] "#F8766D"
```

ggplot2

Hadley Wickham clarifies here

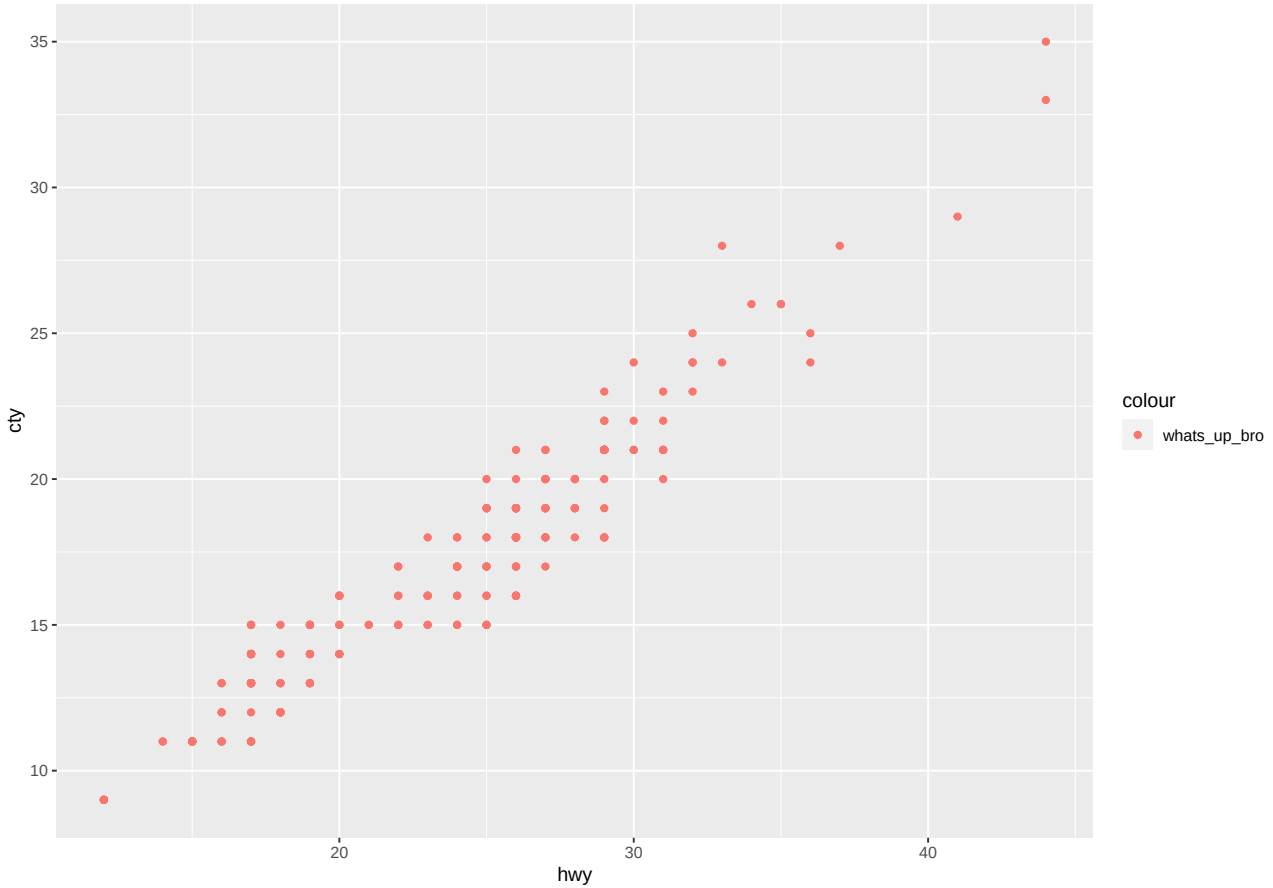
p + geom_point(aes(colour = "darkblue"))

This maps (not sets) the colour to the value "darkblue". This effectively creates a new variable containing only the value "darkblue" and then maps colour to that new variable. Because this value is discrete, the default colour scale uses evenly spaced colours on the colour wheel, and since there is only one value this colour is pinkish (n=1 from color scale of ggplot)

ggplot2

Basically if you map the aesthetics to a constant, it does not matter what you write there, the result is going to be the same

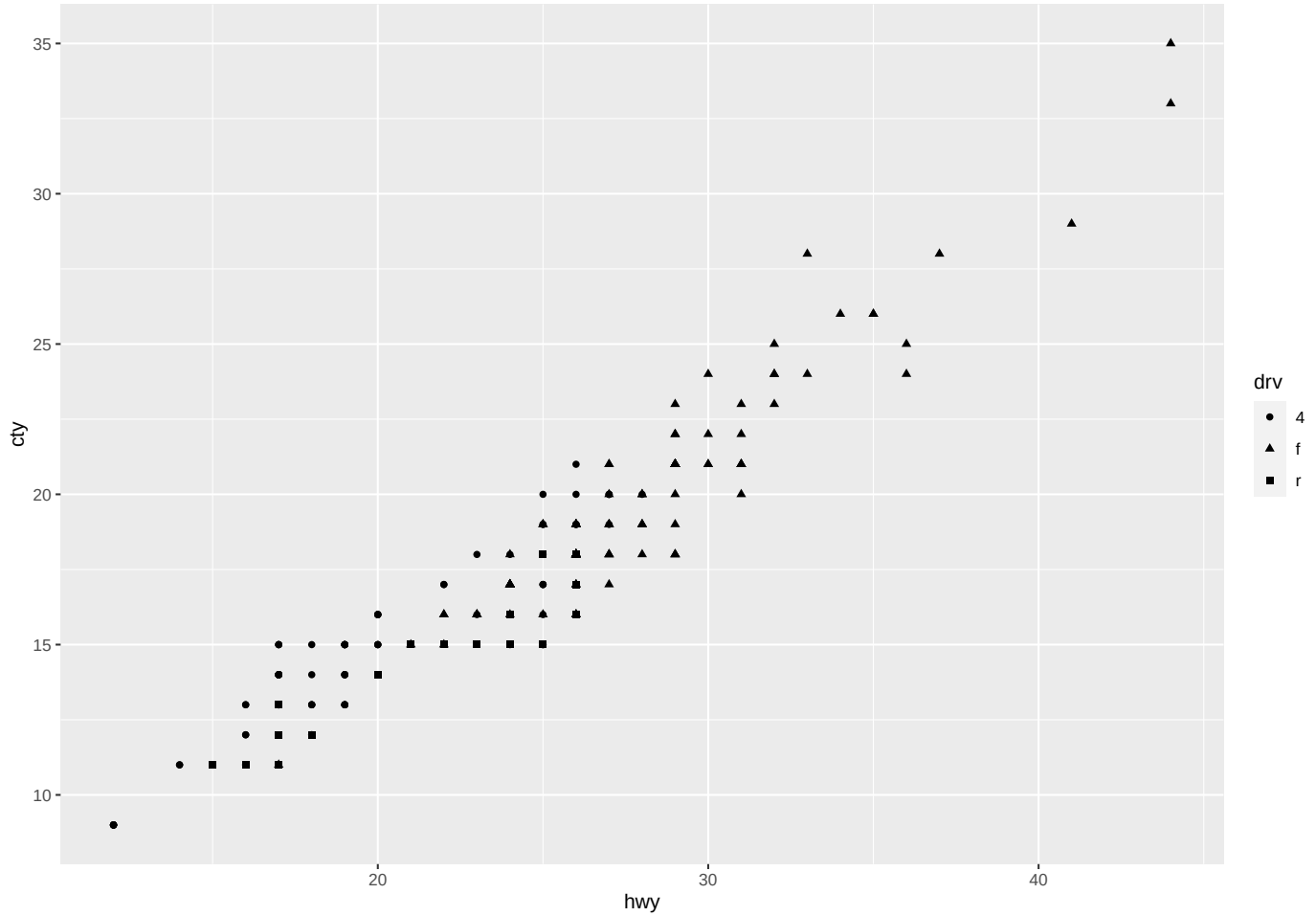
```
ggplot(data = mpg, mapping = aes(x = hwy, y = cty,color = 'whats_up_bro'))+
  geom_point()
```



ggplot2

Other aesthetics: shape

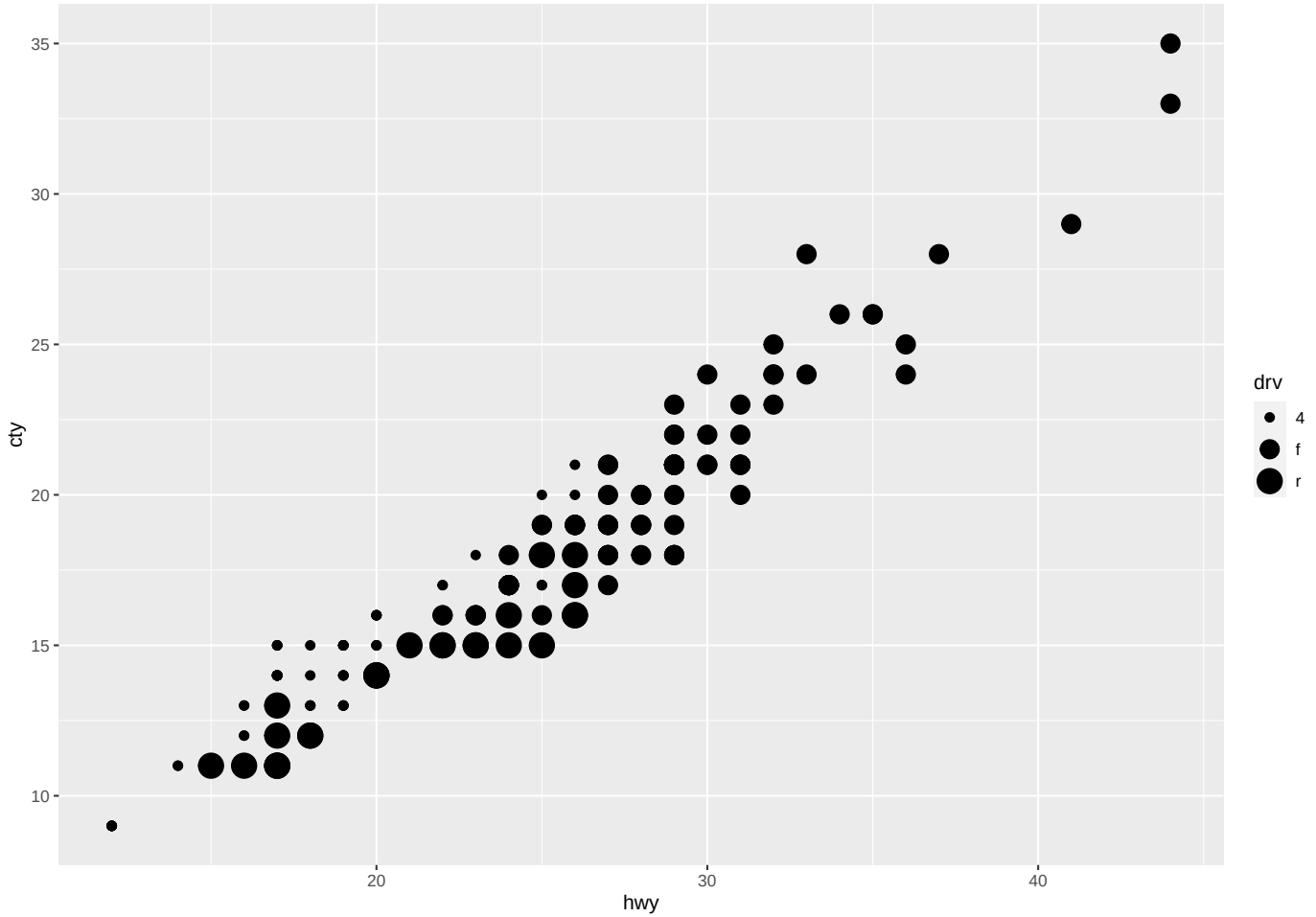
```
ggplot(data = mpg, mapping = aes(x = hwy, y = cty, shape = drv)) +
  geom_point()
```



ggplot2

Size aesthetics

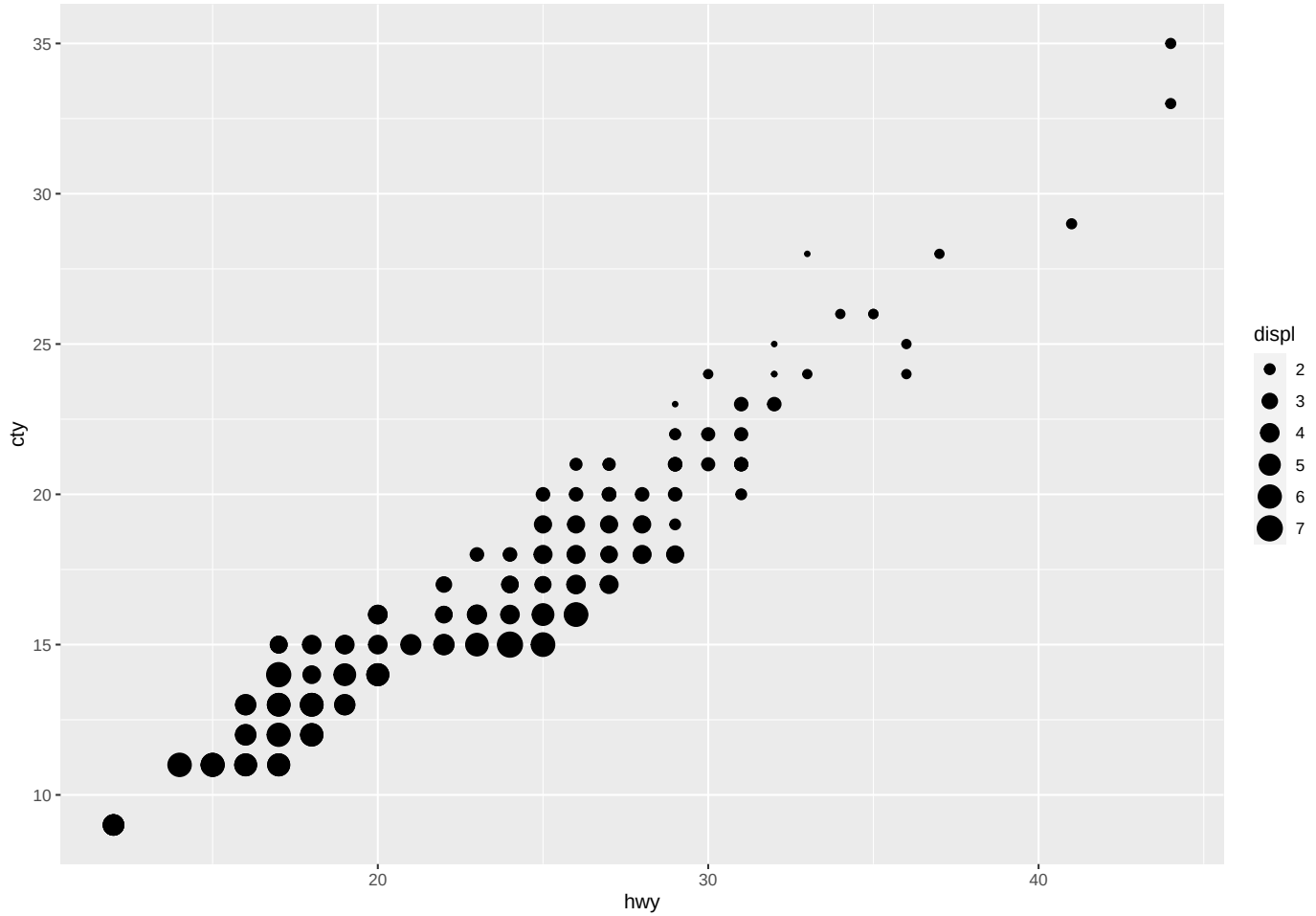
```
ggplot(data = mpg, mapping = aes(x = hwy, y = cty, size = drv)) +
  geom_point()
```



ggplot2

Size as a continuous variable

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



ggplot2

Shape as a continuous variable will raise an error

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

ggplot2

ggplot has a wide range of geometric objects

- geom_bar(): for bar chart
- geom_histogram(): for histogram
- geom_boxplot(): for boxplots etc

ggplot2

Different geometric objects ask for different aesthetics: geom_point() understands the following aesthetics

- x
- y
- alpha
- colour
- fill
- group
- shape
- size
- stroke

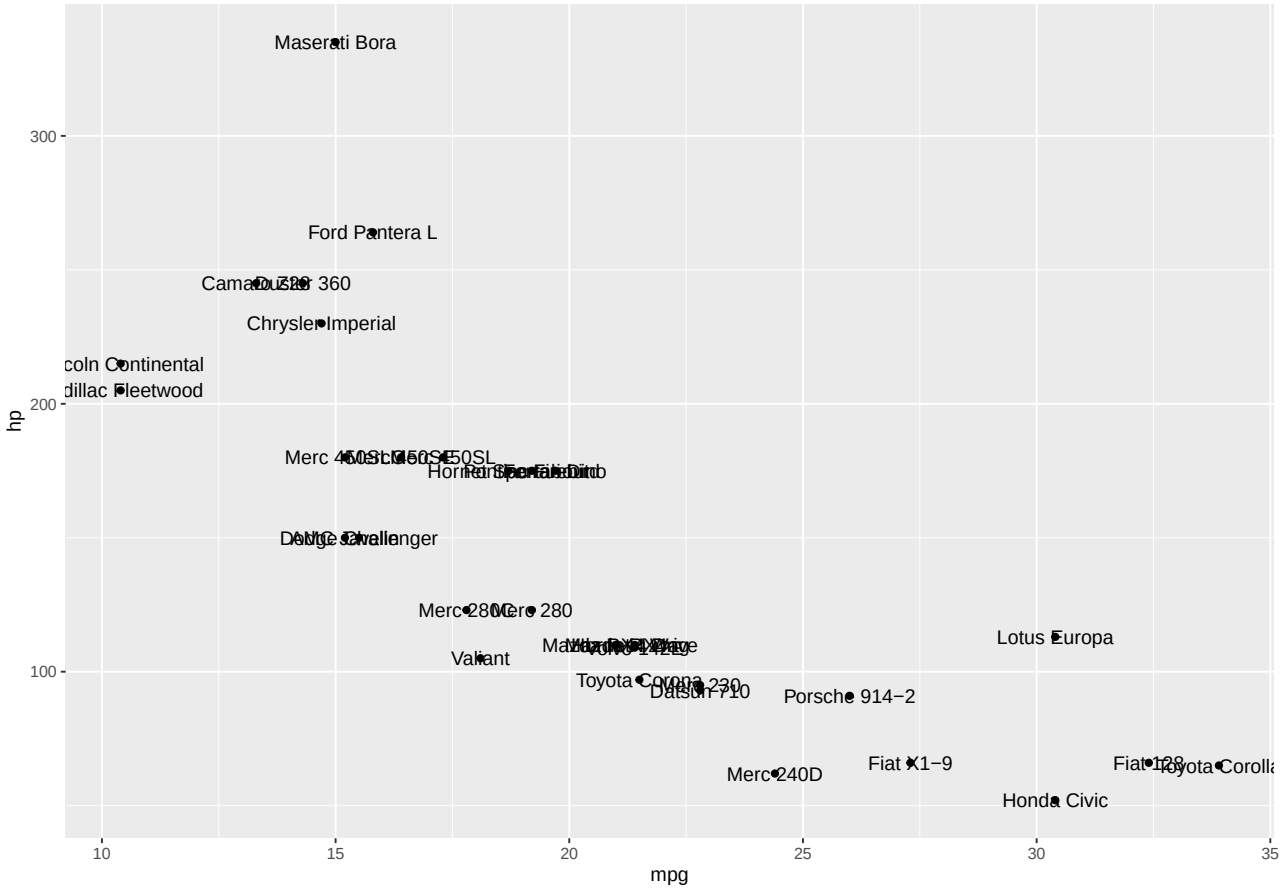
ggplot2

geom_text/geom_label is used to label plots. geom_text() understands the following aesthetics

- x
- y
- label
- alpha
- angle
- colour
- family
- fontface
- group
- hjust
- vjust
- lineheight
- size

ggplot2

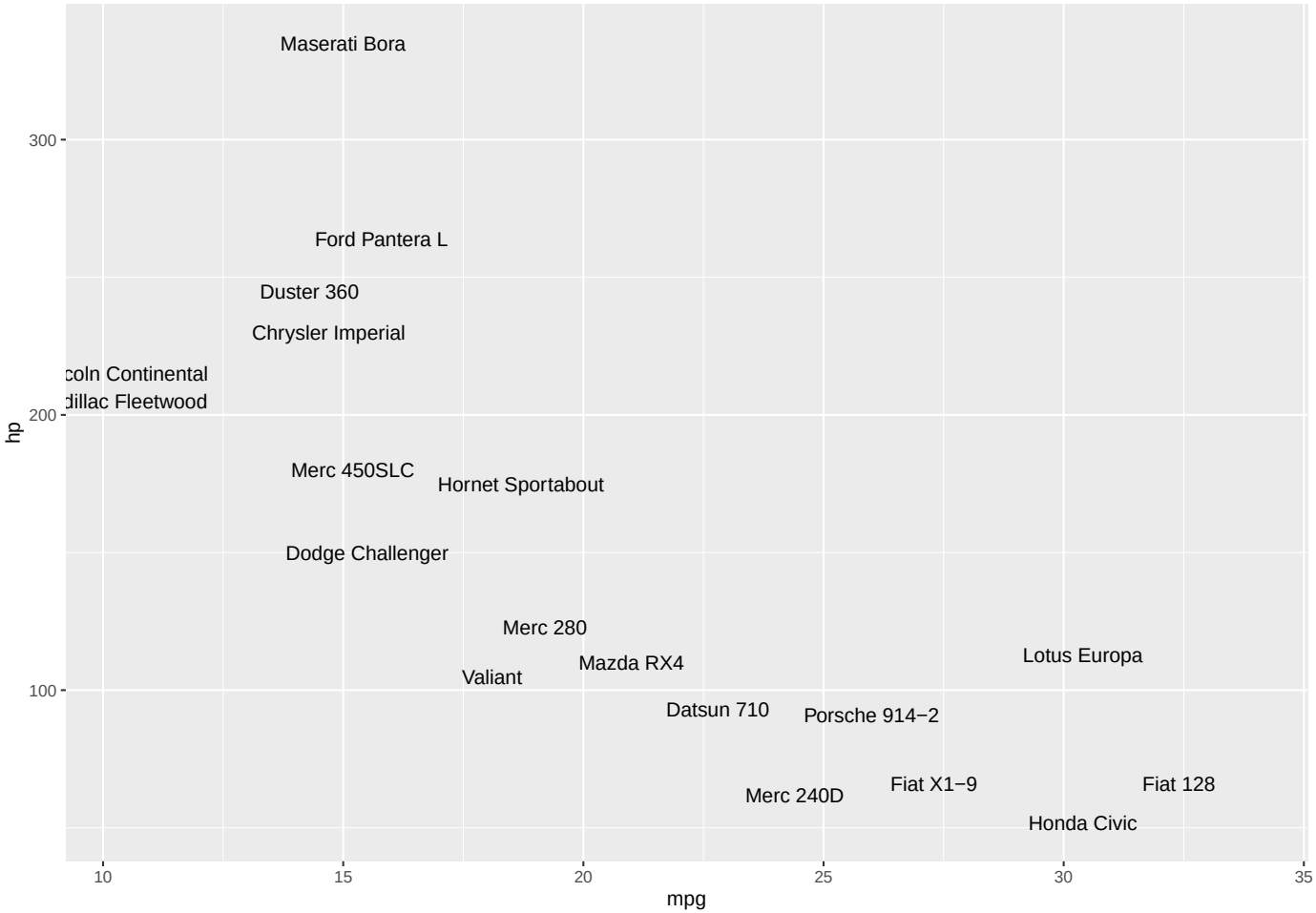
```
mtcars$car <- rownames(mtcars)
ggplot(mtcars, aes(x = mpg, y = hp, label = car)) +
  geom_point() + geom_text()
```



ggplot2

Or just use geom_text()

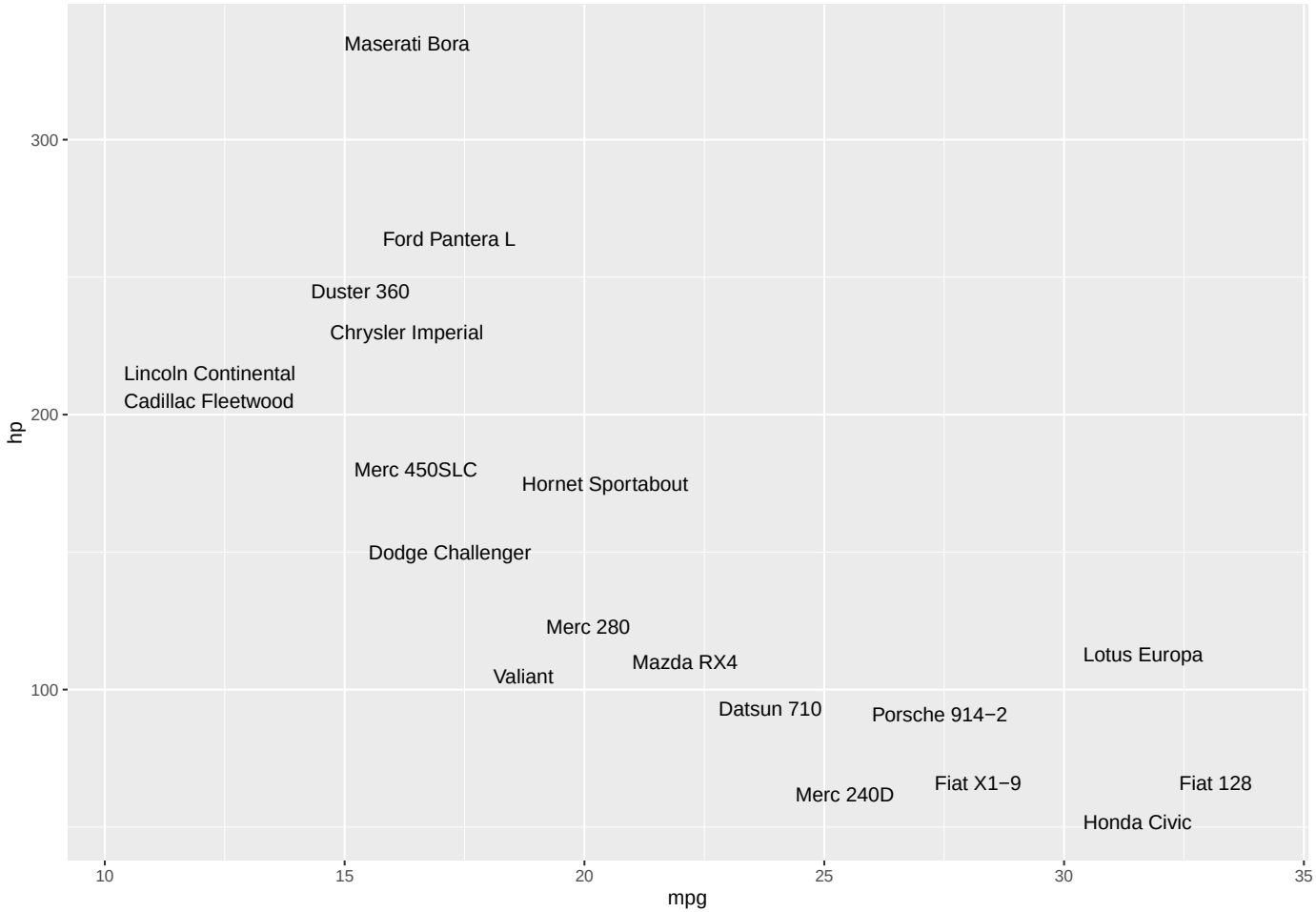
```
ggplot(mtcars, aes(x = mpg, y = hp, label = car)) +
  geom_text(check_overlap = T)
```



ggplot2

Control positions of the labels (horizontal)

```
ggplot(mtcars, aes(x = mpg, y = hp, label = car )) +
  geom_text(check_overlap = T, hjust = 0)
```



ggplot2: statistical transformations

Statistical transformations, stats for short, summarise data in many useful ways.

Every geom has default statistical transformation.

geom_point(mapping = NULL, data = NULL, **stat = "identity"**, position = "identity", na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

geom_histogram(mapping = NULL, data = NULL, **stat = "bin"**, position = "stack", binwidth = NULL, bins = NULL, na.rm = FALSE, orientation = NA, show.legend = NA, inherit.aes = TRUE)

ggplot2: statistical transformations

Identity transformation

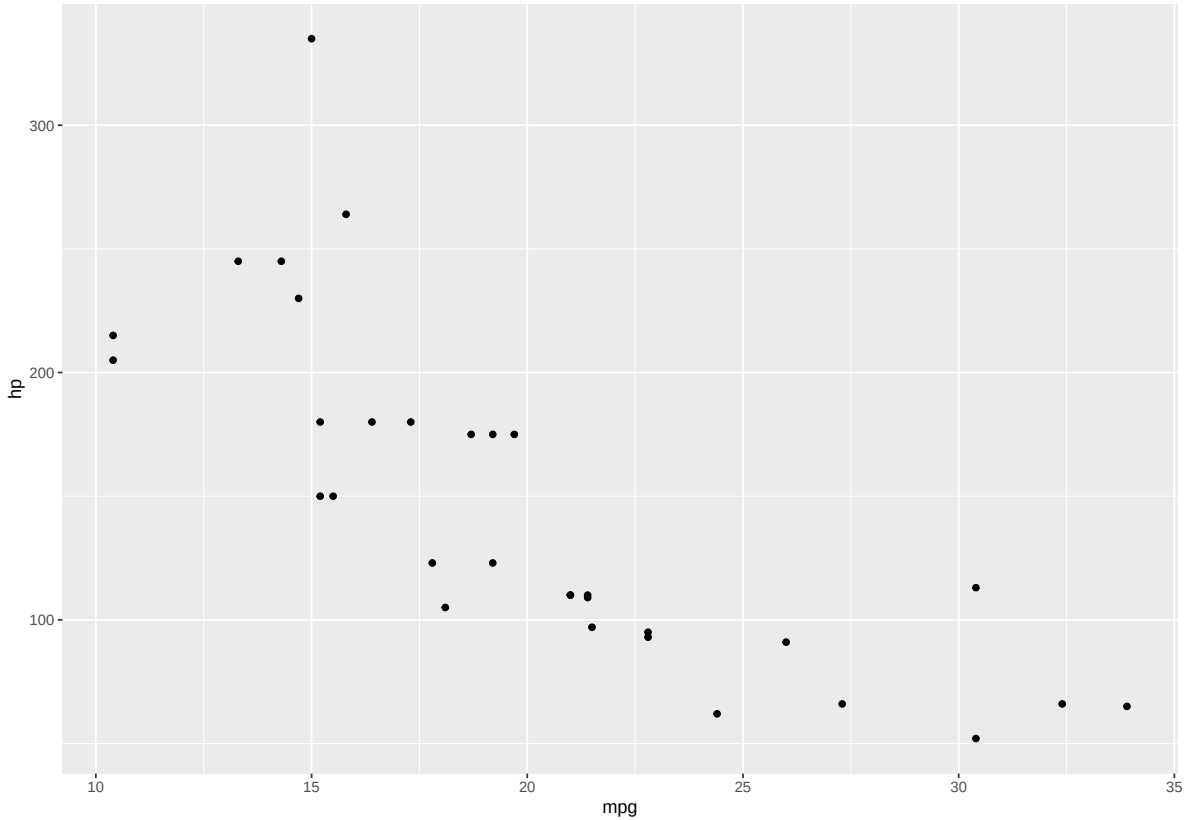
$$f(x) = x$$

ggplot2: statistical transformations

stat_XXX and geom_XXX can be used interchangeably

The same:
ggplot(mtcars, aes(mpg, hp)) + geom_point(stat = 'identity')

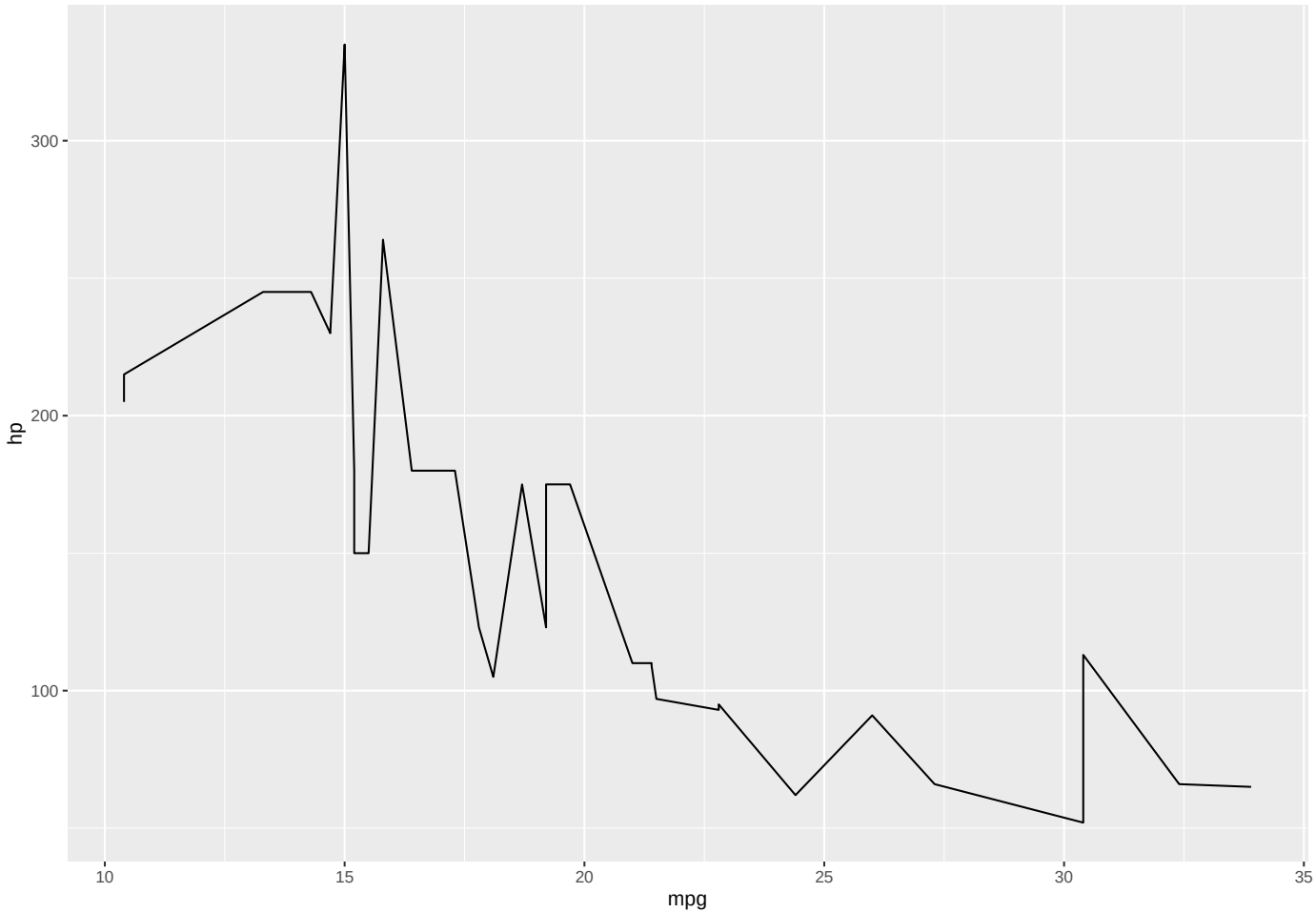
```
ggplot(mtcars, aes(mpg, hp)) + stat_identity(geom = 'point')
```



ggplot2: statistical transformations

Line chart

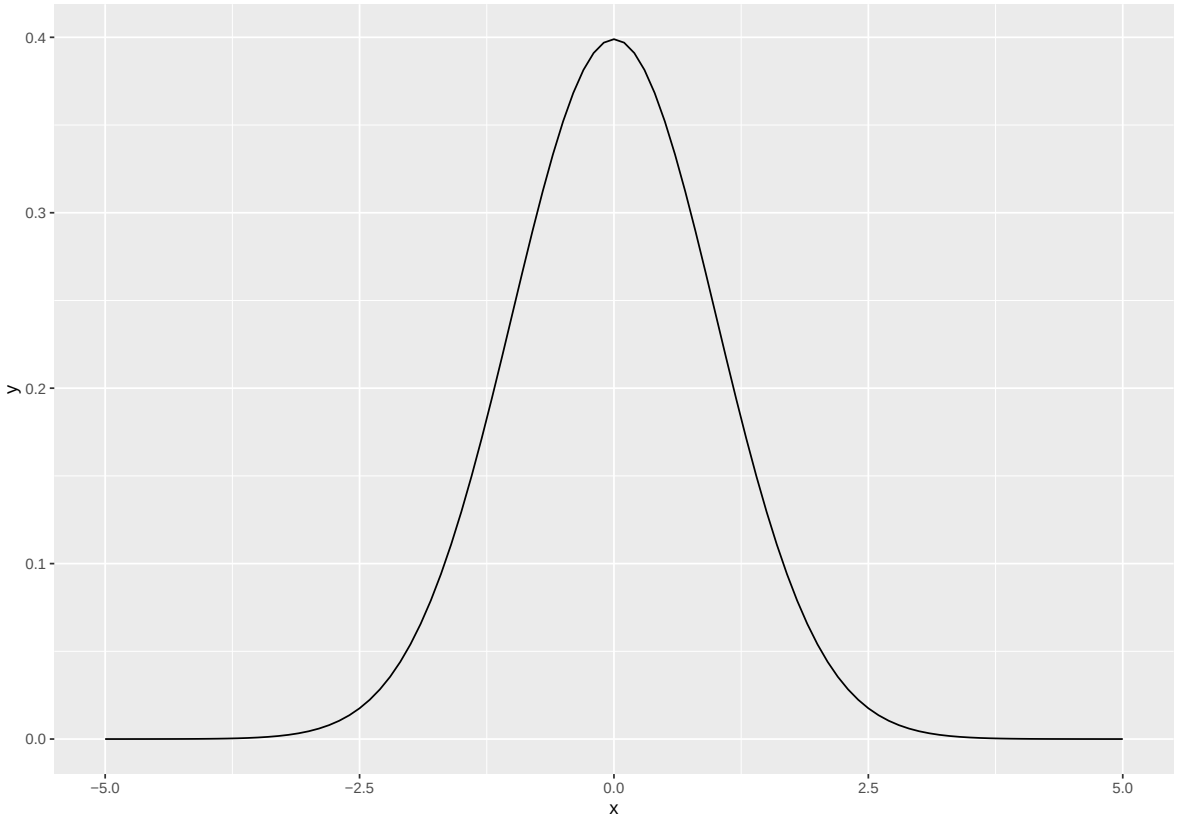
```
ggplot(mtcars, aes(mpg, hp)) + stat_identity(geom = 'line')
```



ggplot2: statistical transformations

Use stat_function to draw any function you want
Normal distribution, values along x axis (-5,5)

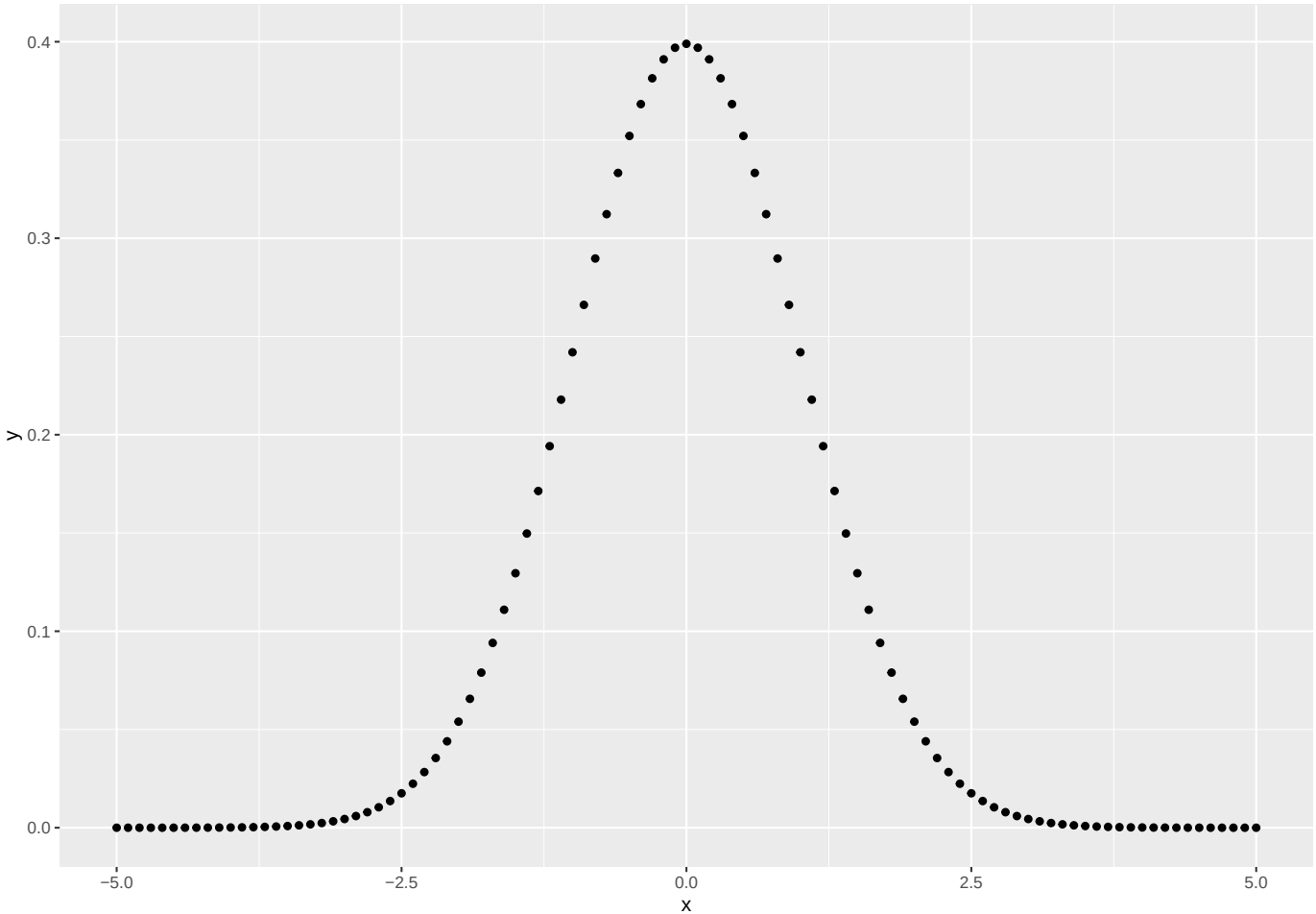
```
ggplot(data.frame(x = c(-5, 5)), aes(x)) + stat_function(fun = dnorm)
```



ggplot2: statistical transformations

Change geom

```
ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = dnorm, geom = 'point')
```



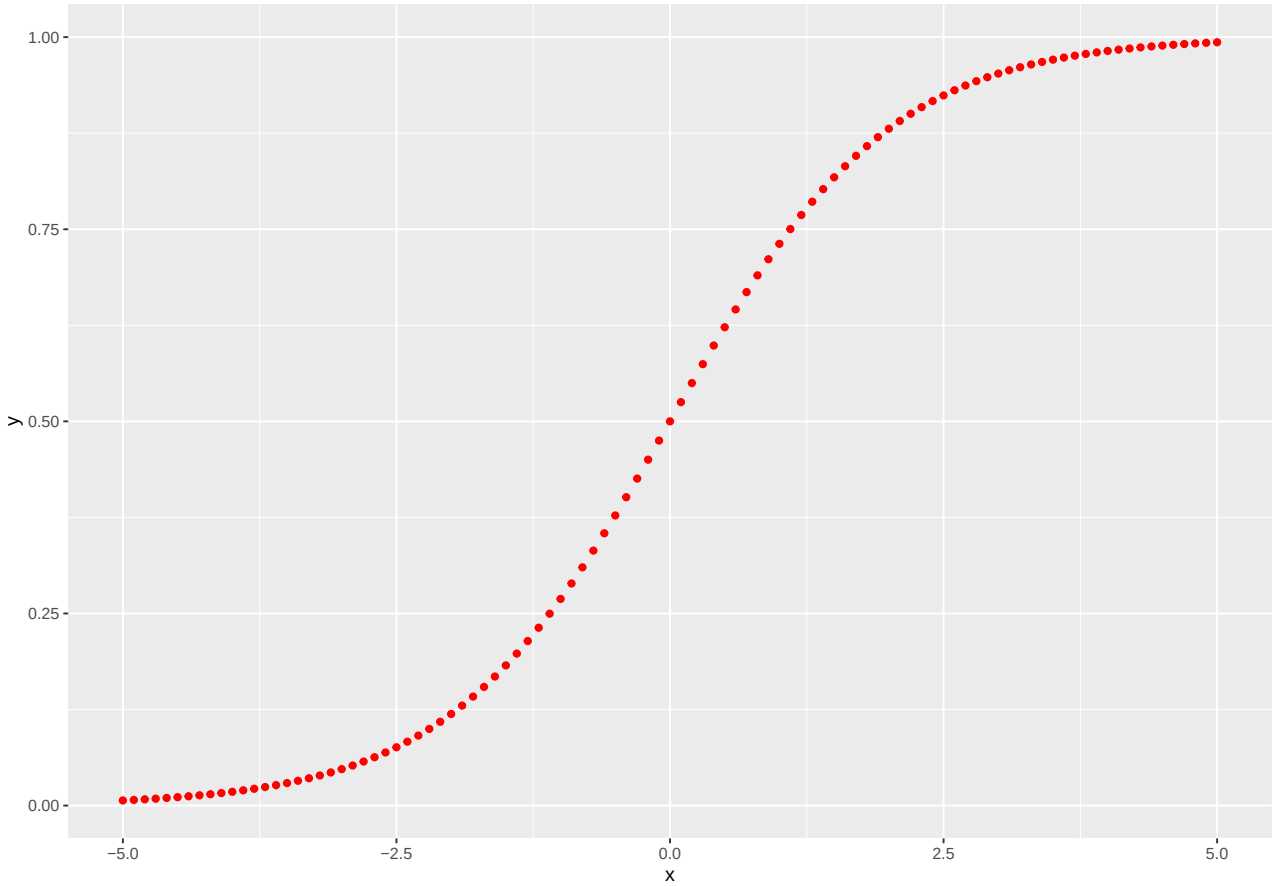
ggplot2: statistical transformations

With user defined functions: Logistic function

$$f(x) = \frac{1}{1 + e^{-x}}$$

ggplot2: statistical transformations

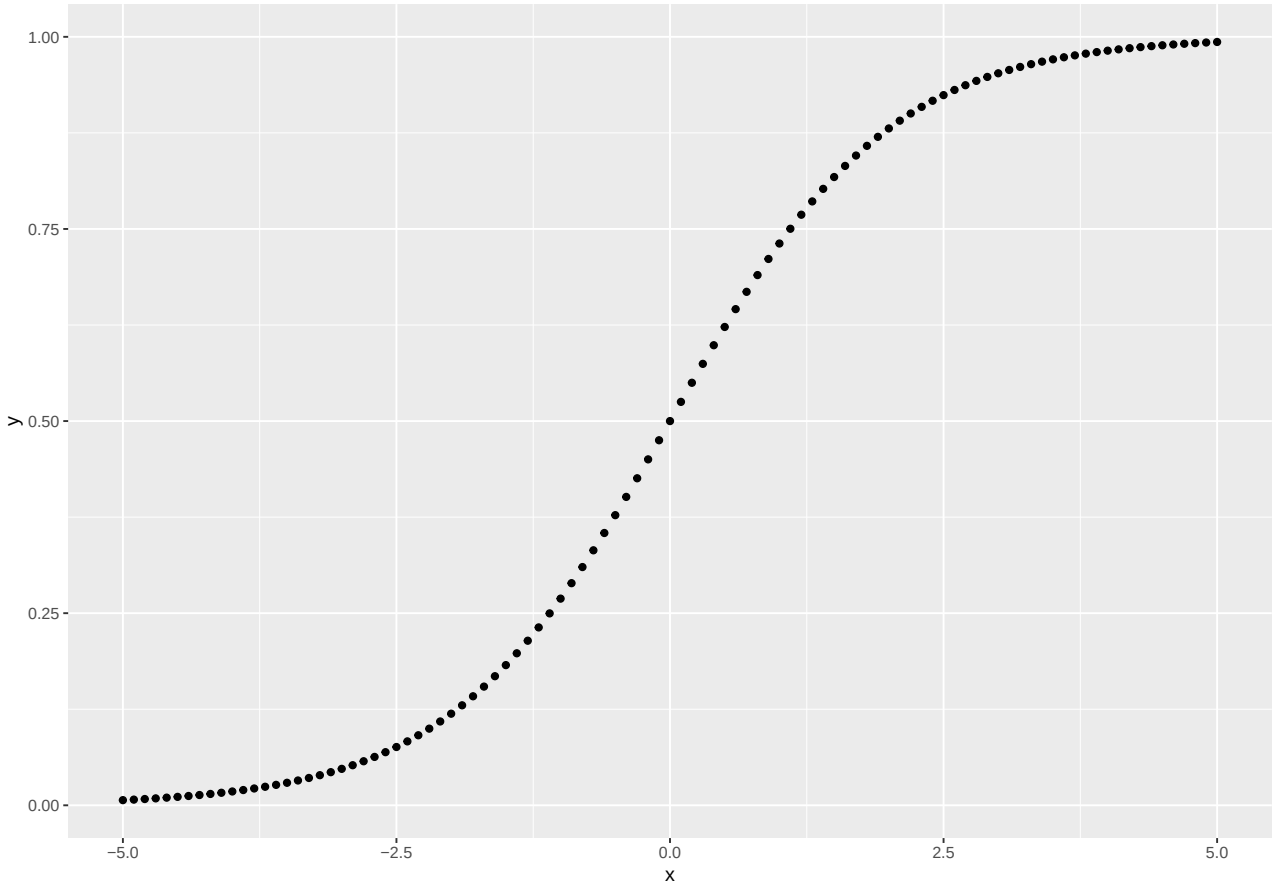
```
logistic <- function(x){1/(1 + exp(-x))}
ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = logistic, geom = 'point', color = 'red')
```



ggplot2: statistical transformations

Define the function inside stat_function

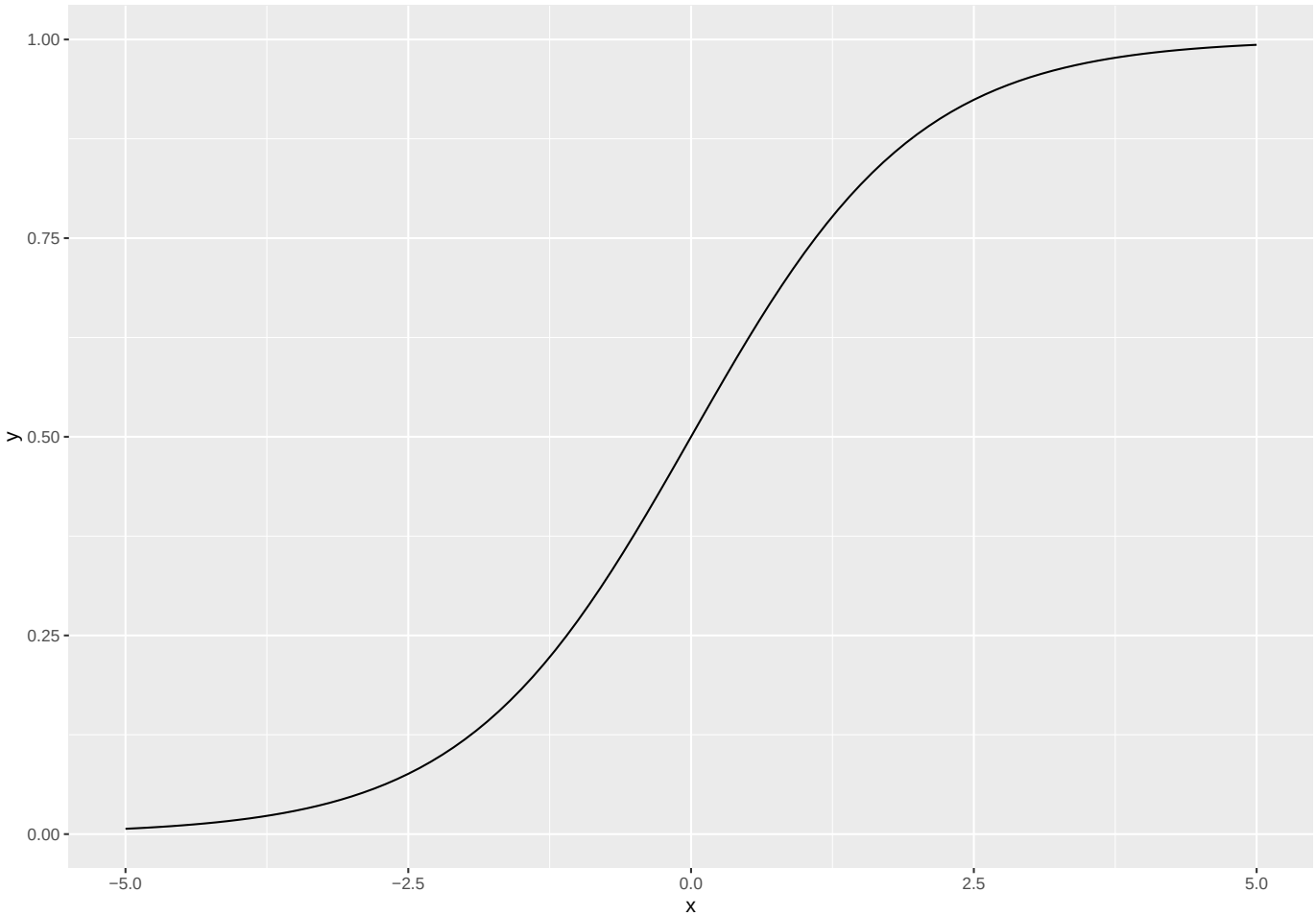
```
ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = function(x) 1/(1 + exp(-x)), geom = 'point')
```



ggplot2: statistical transformations

geom_function() does the same but with defined geometry - line

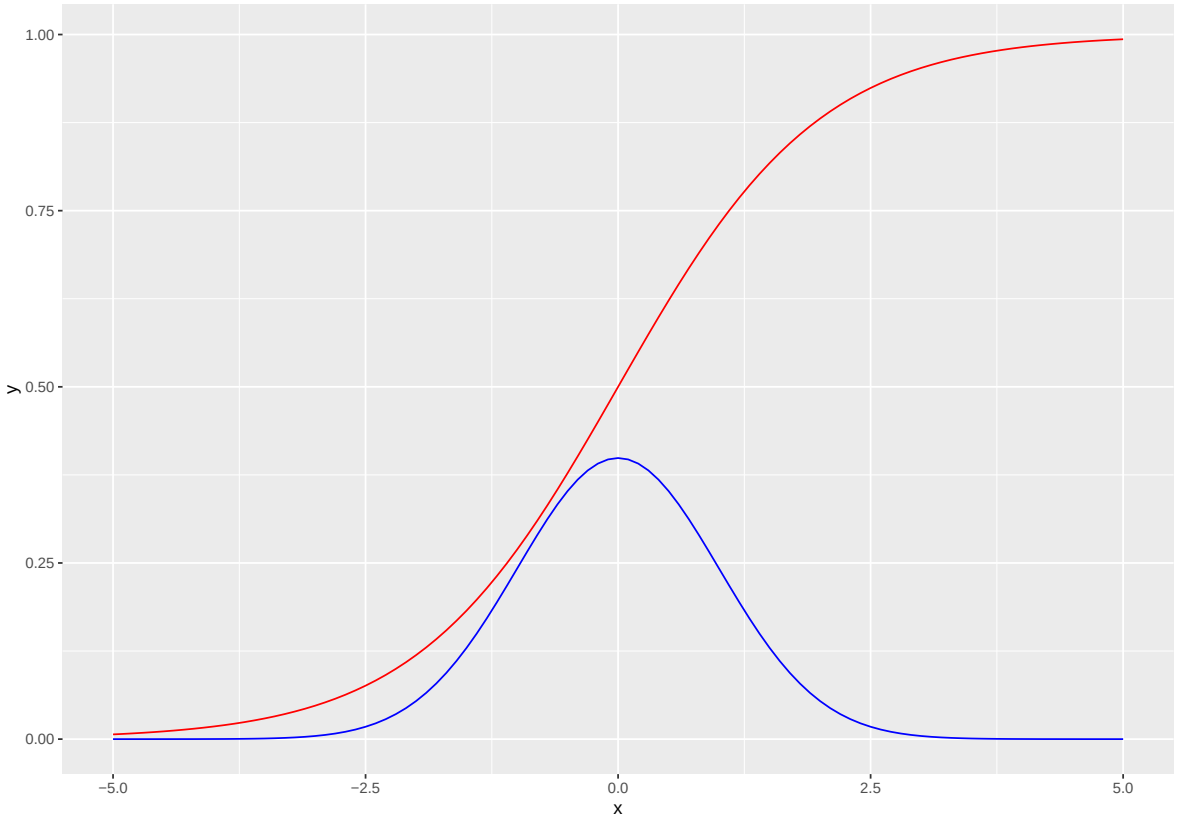
```
ggplot(data.frame(x = c(-5, 5)), aes(x)) + geom_function(fun = logistic)
```



ggplot2: statistical transformations

Plot two functions together

```
ggplot(data.frame(x = c(-5, 5)), aes(x)) +
  stat_function(fun = logistic, color = 'red') +
  stat_function(fun = dnorm, color = 'blue')
```



ggplot2: scales

A scale controls the mapping from data to aesthetic attributes, and we need a scale for every aesthetic used on a plot. Each scale operates across all the data in the plot, ensuring a consistent mapping from data to aesthetics.

(Hadley Wickham)

ggplot2: scales

They start with scale_, followed by the name of the aesthetic (e.g., colour_, shape_ or x_), and finally by the name of the scale (e.g., gradient, hue, manual, discrete, etc).

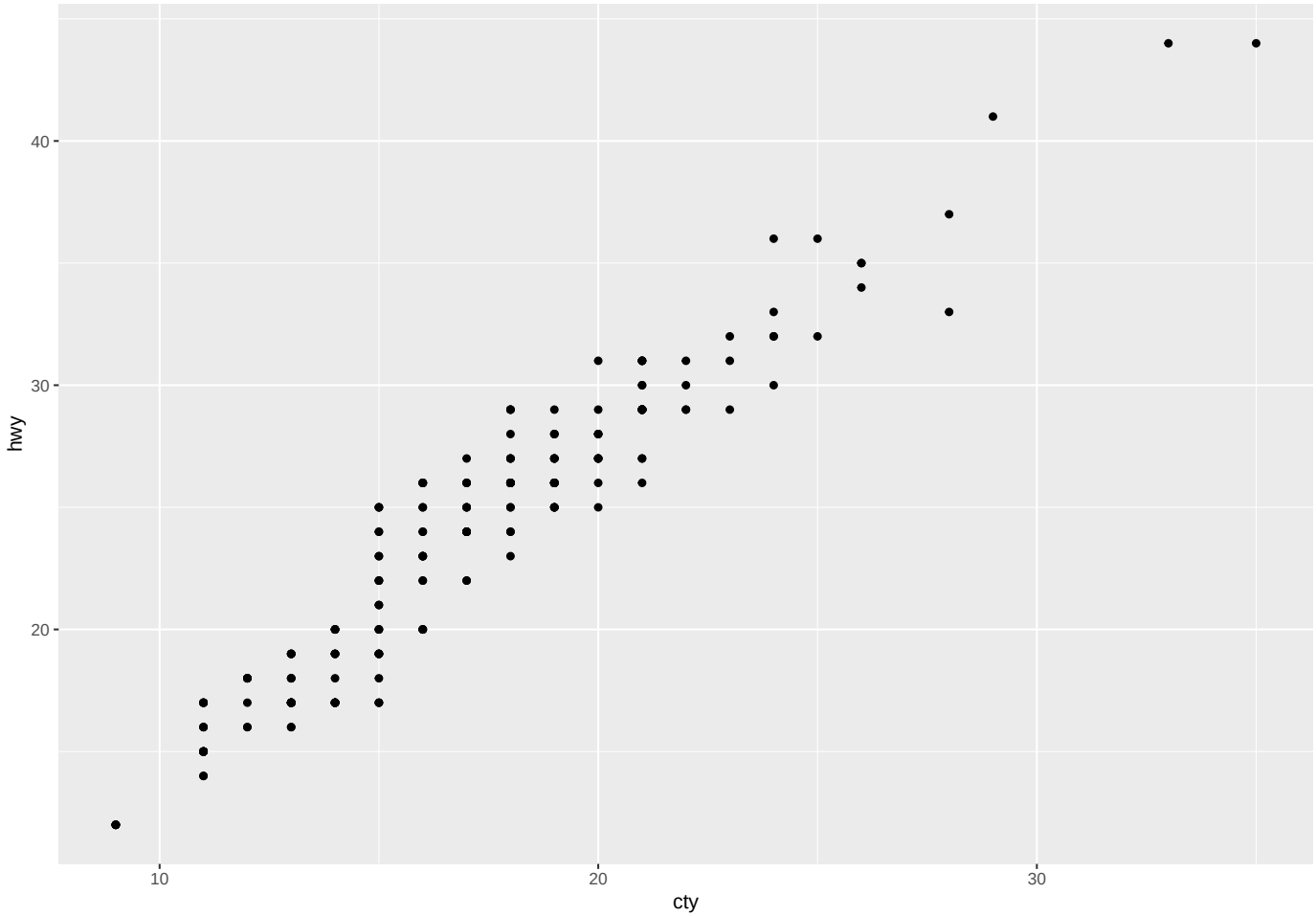
All aesthetics come with the default scale, you can override them with scale_ layers

Example: x, y aesthetics can be either continuous or discrete, thus we have scale_x_continuous, scale_x_discrete, scale_y_continuous, scale_y_discrete (and many more)

ggplot2: scales

Set your own breaks

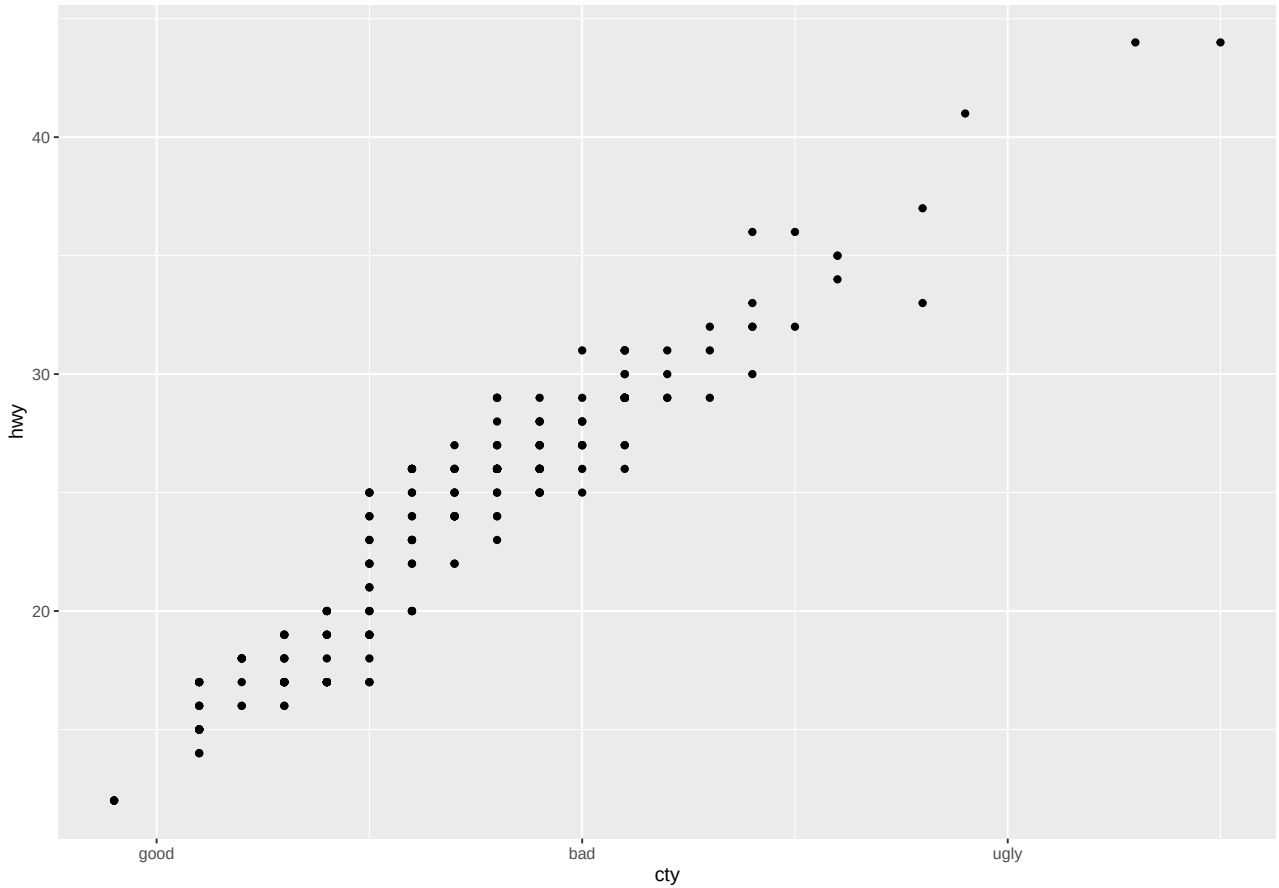
```
ggplot(mpg, aes(x = cty, y = hwy)) + geom_point() +
  scale_x_continuous(breaks = c(10,20,30))
```



ggplot2: scales

Adding labels to the breaks

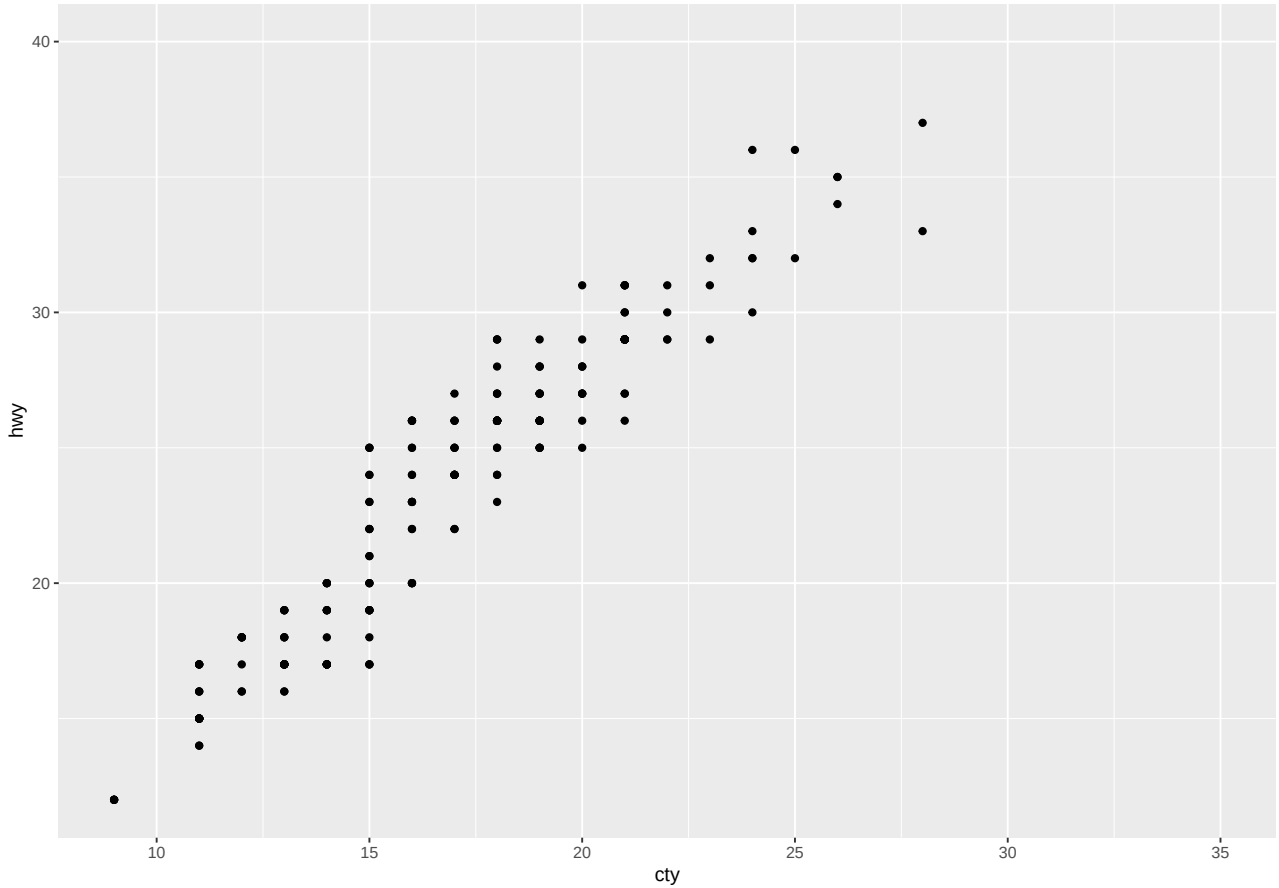
```
ggplot(mpg, aes(x = cty, y = hwy)) + geom_point() +
  scale_x_continuous(breaks = c(10,20,30),
                     labels = c('good', 'bad', 'ugly'))
```



ggplot2: scales

Setting the limits for y axis (note the minimum value is NA, thus decided by ggplot2)

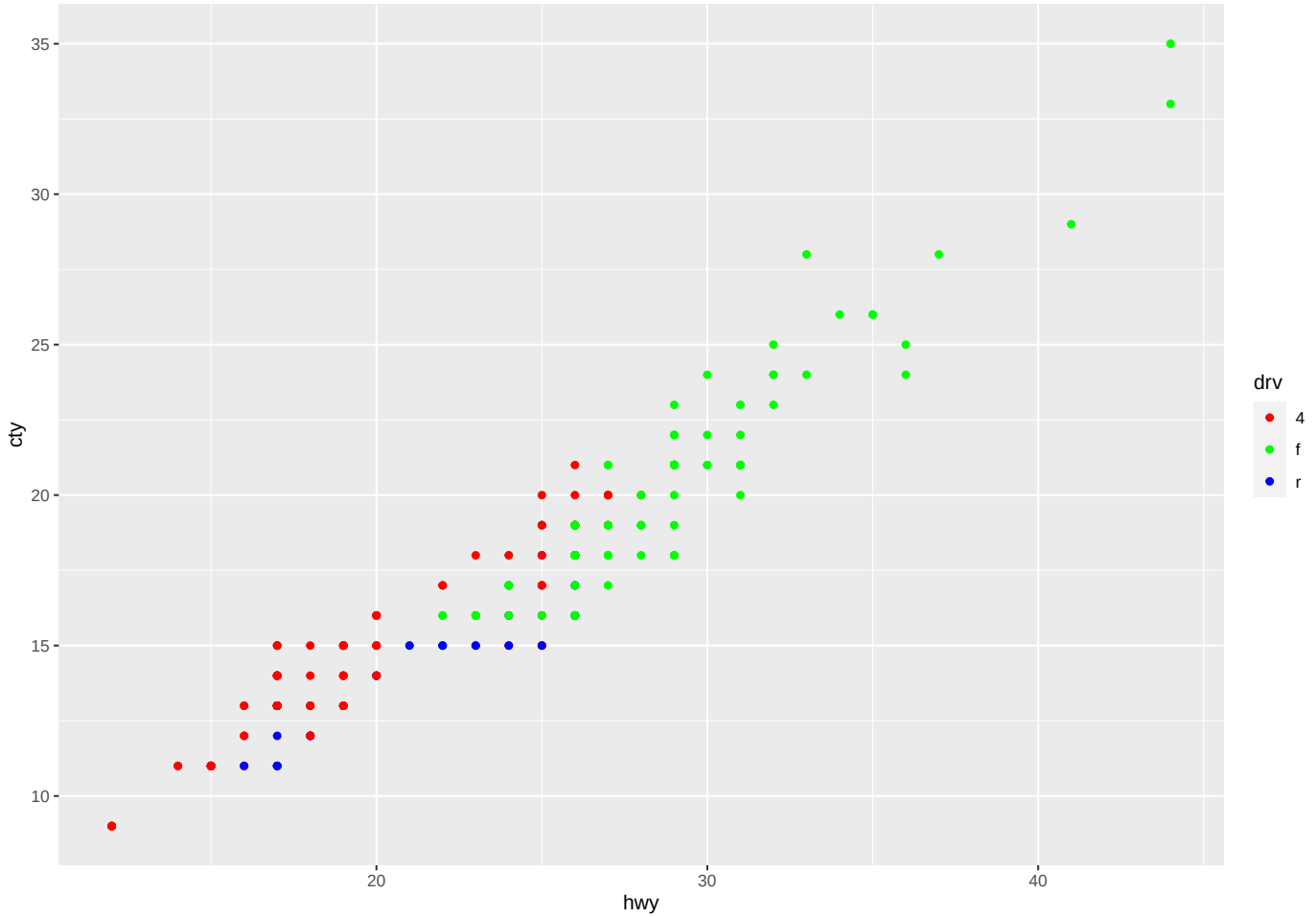
```
ggplot(mpg, aes(x = cty, y = hwy)) + geom_point() +
  scale_y_continuous(limits = c(NA,40))
```



ggplot2: scales

Using scale with color aesthetics Set color manually

```
ggplot(data = mpg, mapping = aes(x = hwy, y = cty, color = drv)) +
  geom_point() + scale_color_manual(values = c('red', 'green', 'blue'))
```



ggplot2: coordinate system

Coordinate systems tie together the two position scales to produce a 2d location. Currently, ggplot2 comes with six different coordinate systems. All these coordinate systems are two-dimensional.

Coordinate systems in ggplot

- cartesian
- equal
- flip
- trans
- map
- polar

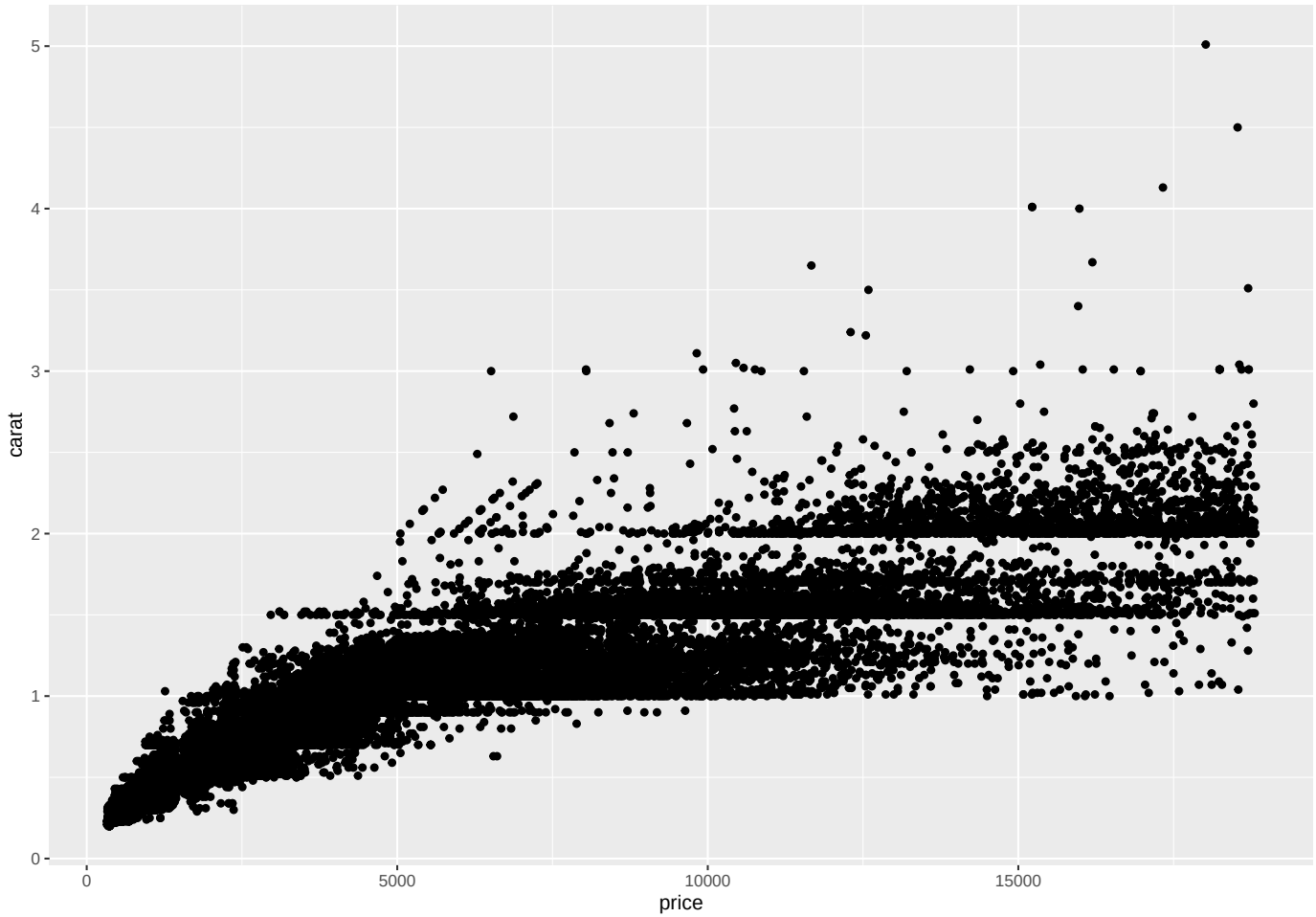
ggplot2: coordinate system

- The four Cartesian-based coordinate systems, coord_cartesian, coord_equal, coord_flip and coord_trans, share a number of common features.
- They are still essentially Cartesian because the x and y positions map orthogonally to x and y positions on the plot.

ggplot2: coordinate system

Diamonds dataset

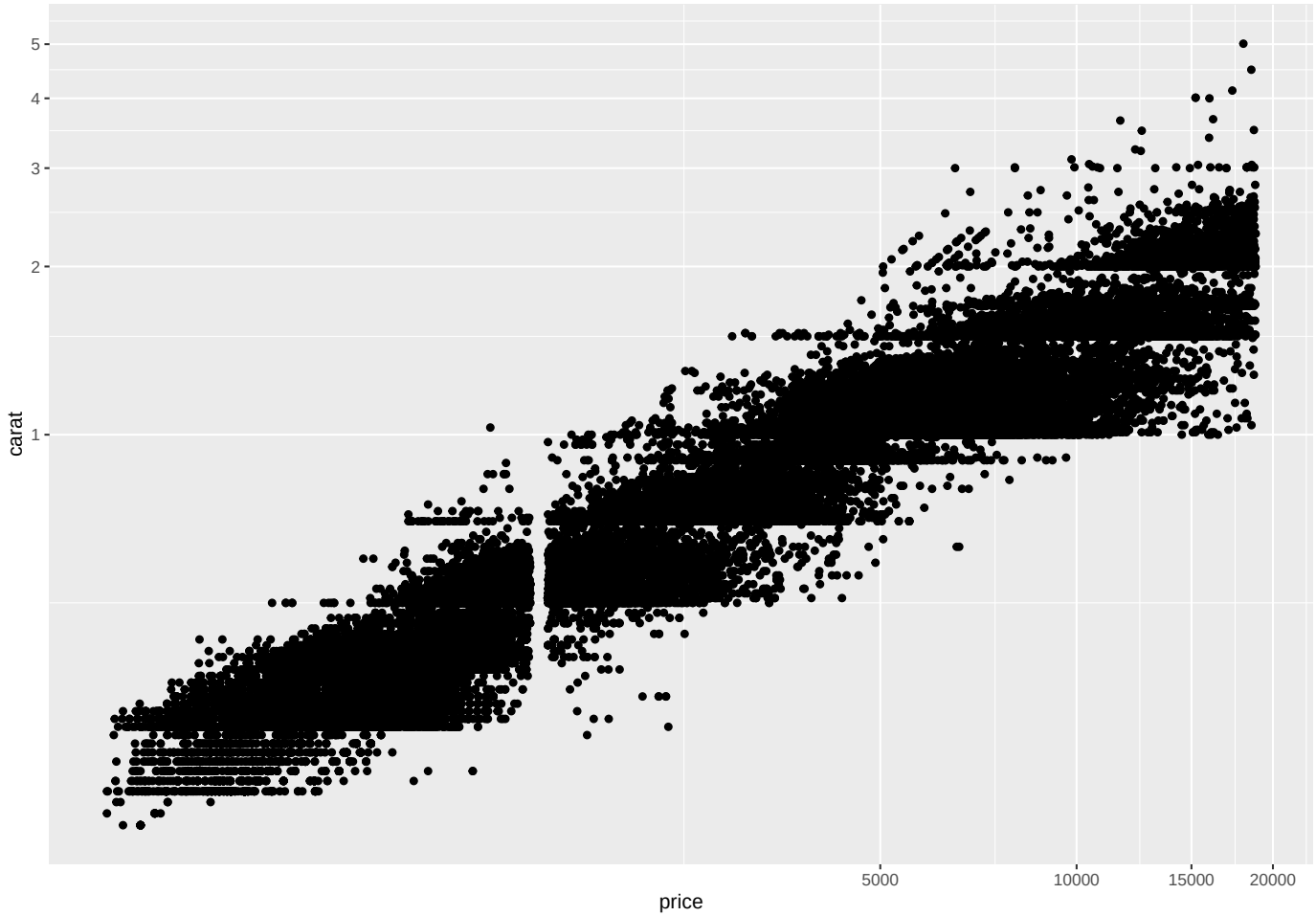
```
ggplot(data = diamonds, aes(price, carat)) + geom_point()
```



ggplot2: coordinate system

Transforming the axes: Log-Log plot

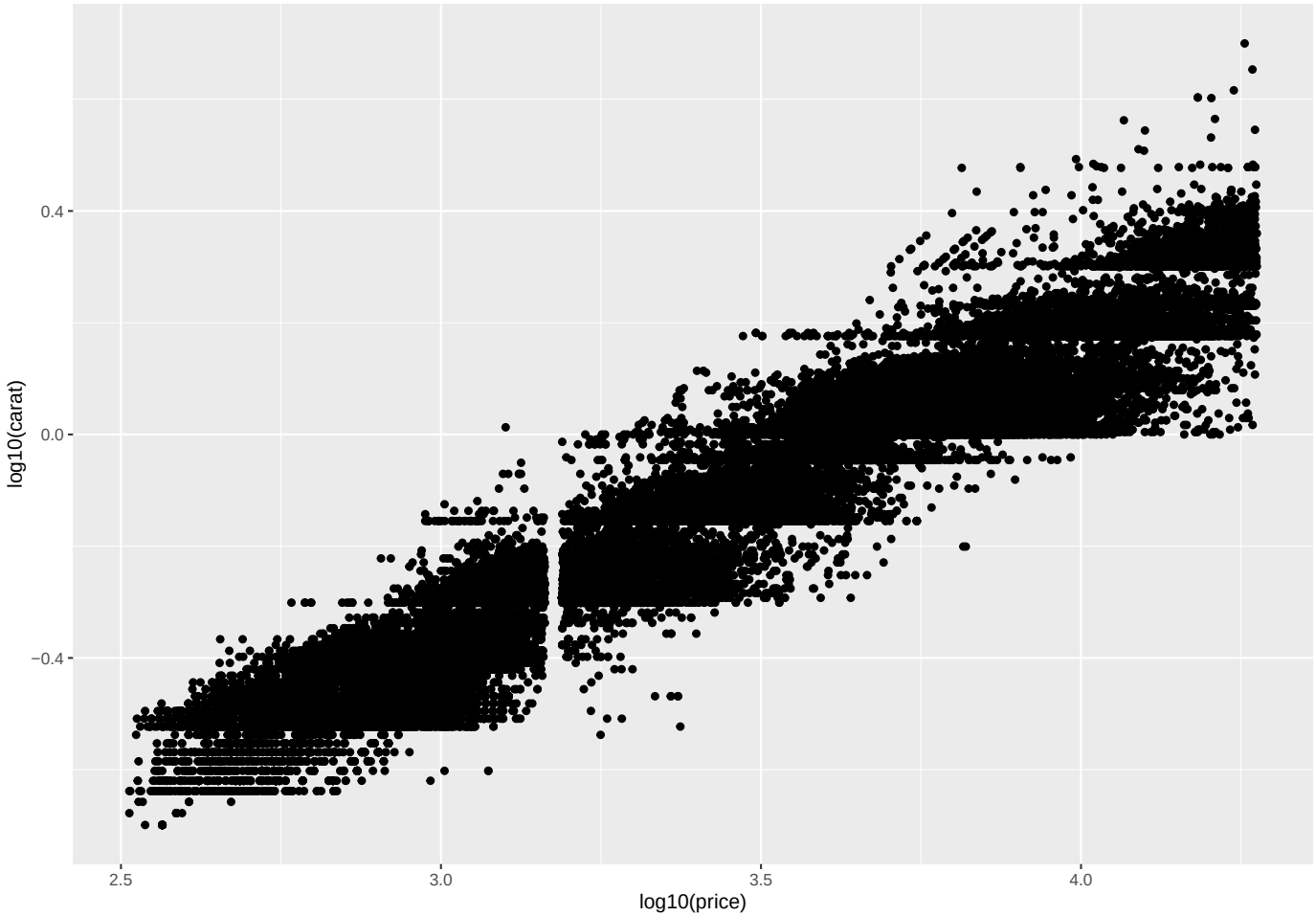
```
ggplot(data = diamonds, aes(price, carat)) + geom_point() +
  coord_trans(x = 'log10', y = 'log10')
```



ggplot2: coordinate system

You can get "almost" the same result with this

```
ggplot(data = diamonds, aes(log10(price), log10(carat))) +
  geom_point()
```

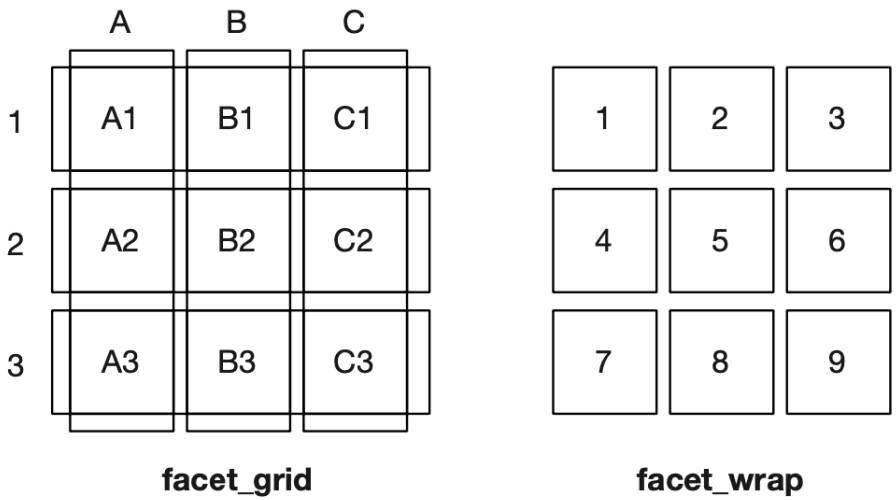


ggplot2: faceting

- Faceting creates tables of graphics by splitting the data into subsets and displaying the same graph for each subset in an arrangement that facilitates comparison.
- ggplot2 has two layers for faceting: facet_grid and facet_wrap

ggplot2: faceting

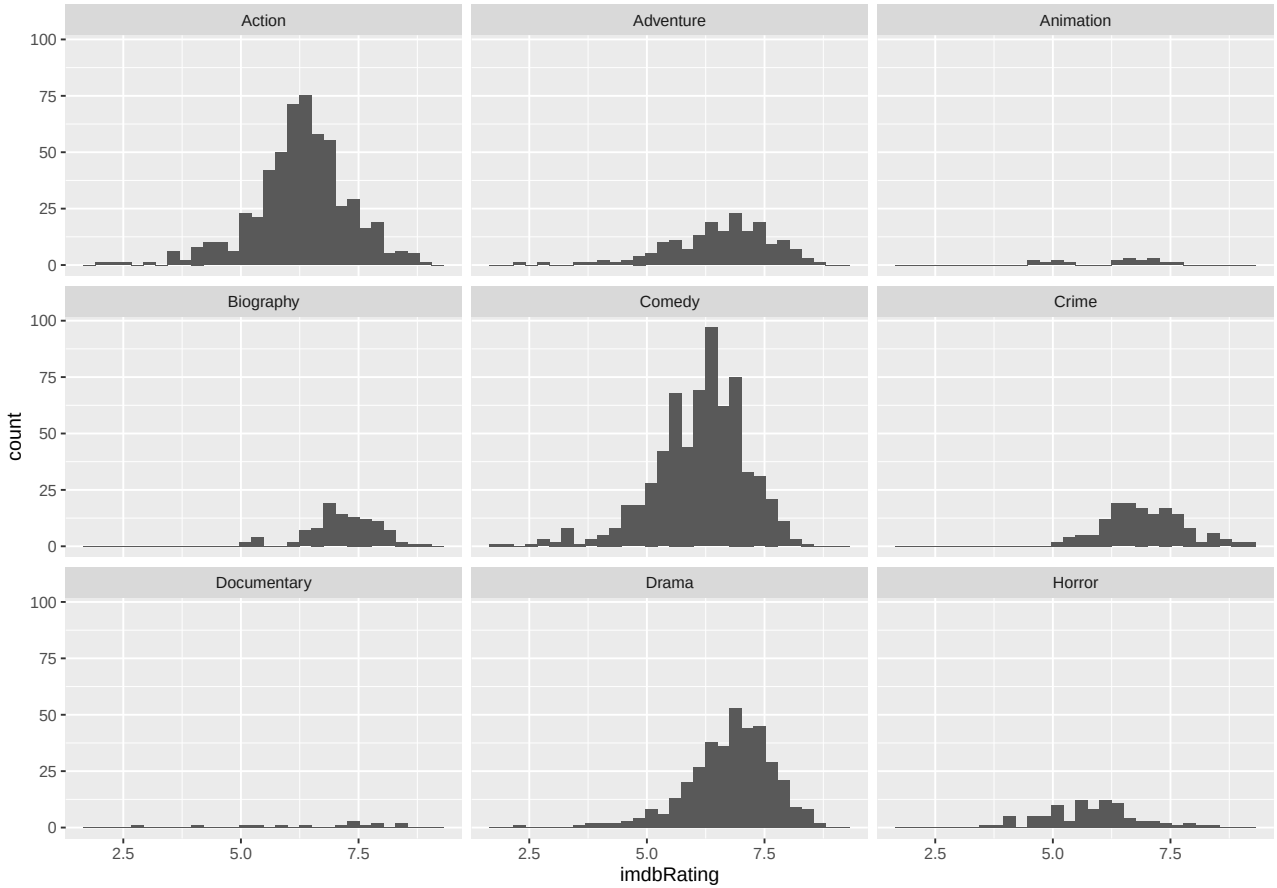
- facet_grid() produces a 2d grid of panels defined by variables which form the rows and columns,
- facet_wrap() produces a 1d ribbon of panels that is wrapped into 2d.



ggplot2: faceting

Get the histogram of imdbRating by genre_first

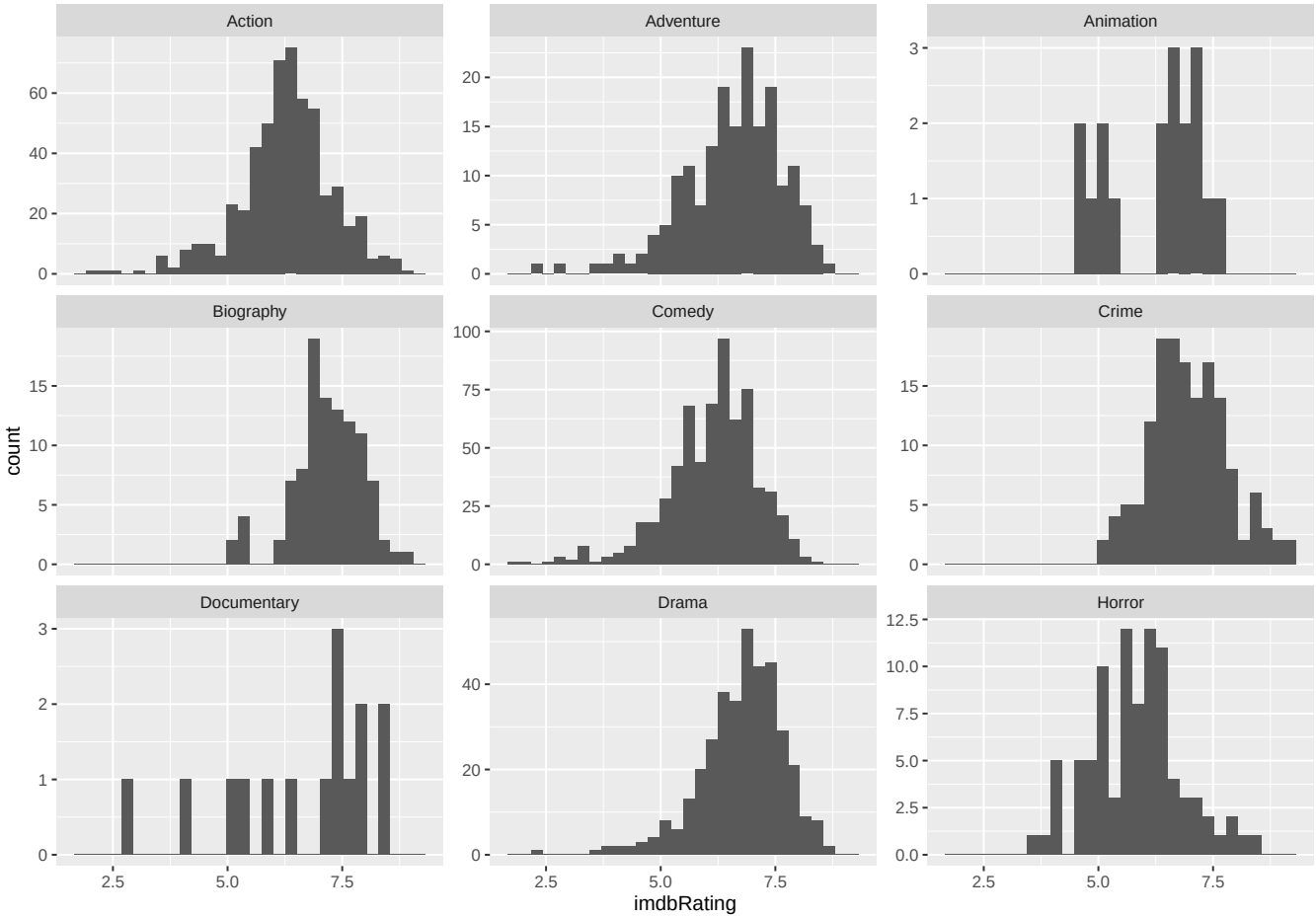
```
movies_small <- read.csv('movies_small.csv')
ggplot(movies_small, aes(x = imdbRating)) + geom_histogram() +
  facet_wrap(~genre_first)
```



ggplot2: faceting

Free y axis to vary

```
ggplot(movies_small, aes(x = imdbRating)) + geom_histogram() +
  facet_wrap(.~genre_first, scales = 'free_y')
```

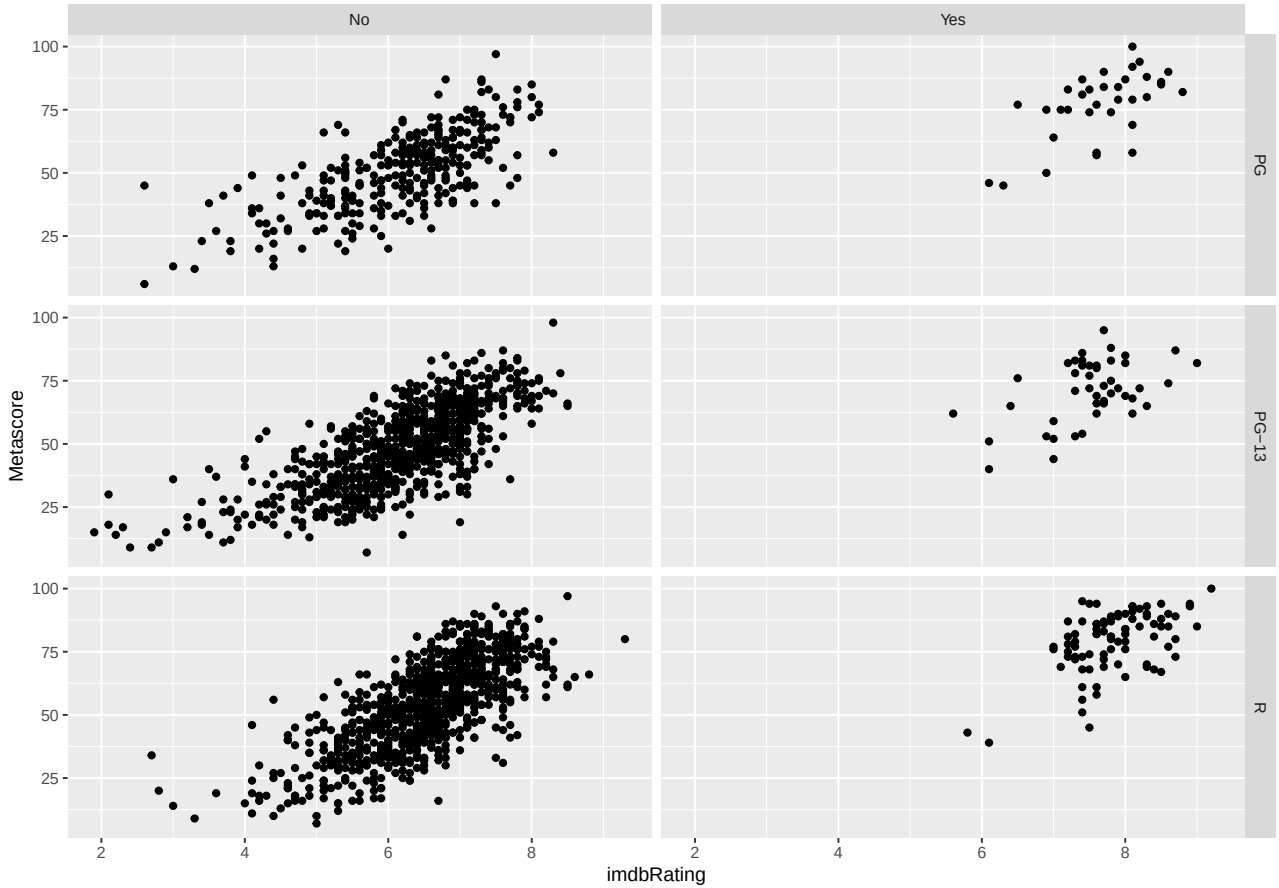


ggplot2: faceting

- facet_grid() allows to create 2d grid of panels defined by variables which form the rows and columns.
- Look at the formula: Rated is in rows, has_oscar is in columns

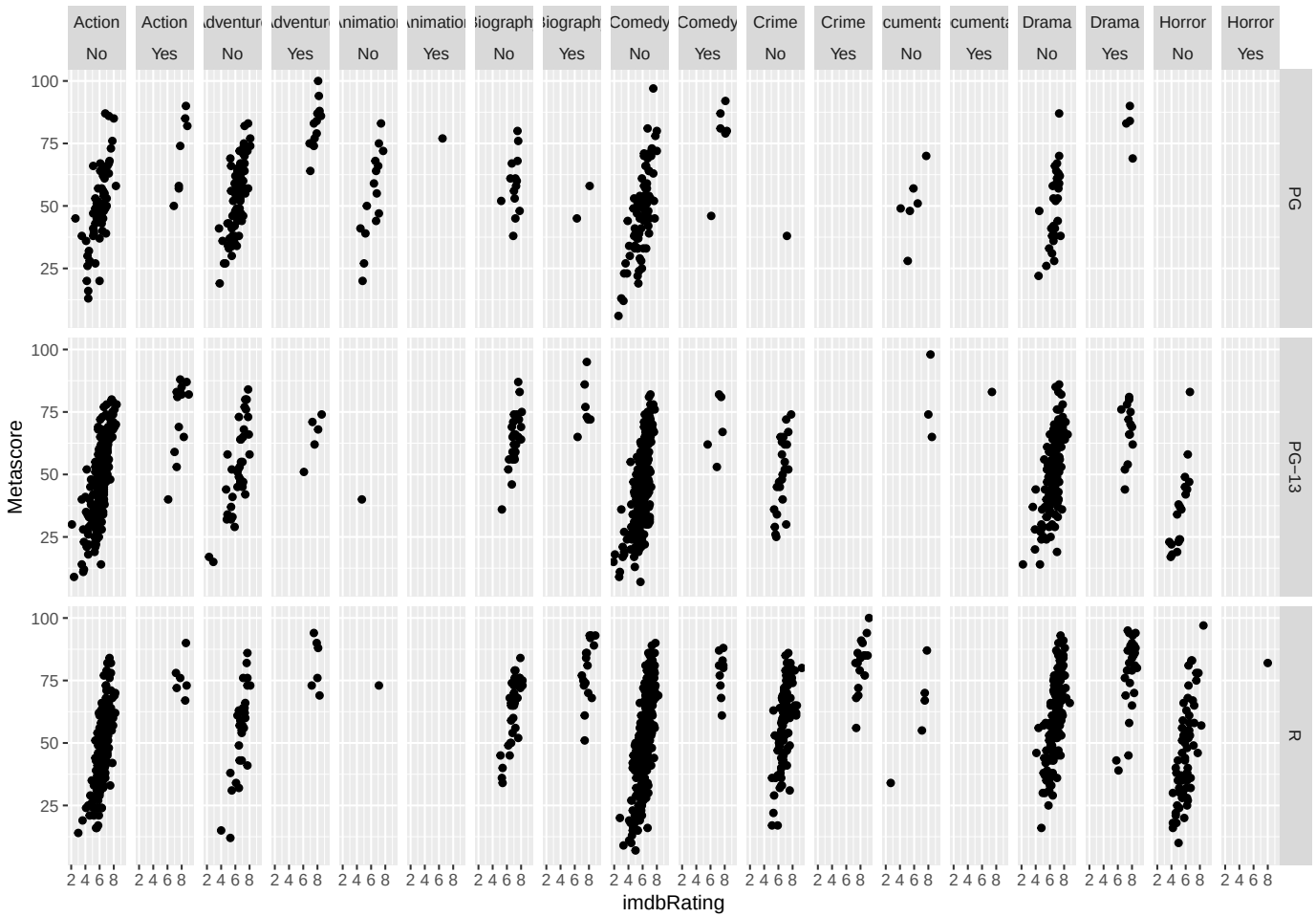
ggplot2: faceting

```
ggplot(movies_small, aes(x = imdbRating, y = Metascore)) + geom_point() +
  facet_grid(Rated~has_oscar)
```



ggplot2: faceting - more than 2 variables

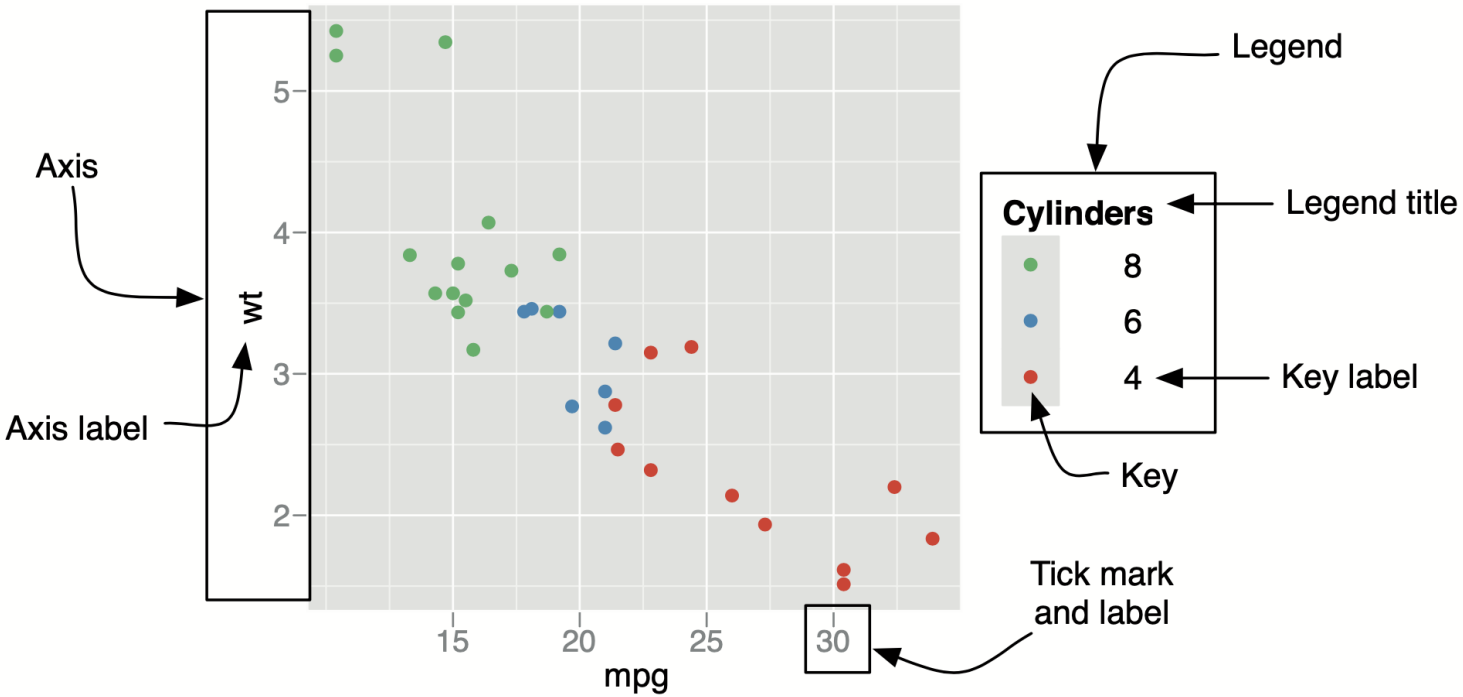
```
ggplot(movies_small, aes(x = imdbRating, y = Metascore)) + geom_point() +
  facet_grid(Rated~genre_first+has_oscar)
```



ggplot2: guides and themes

- Collectively, axes and legends are called guides
- They allow you to read observations from the plot and map them back to their original values.

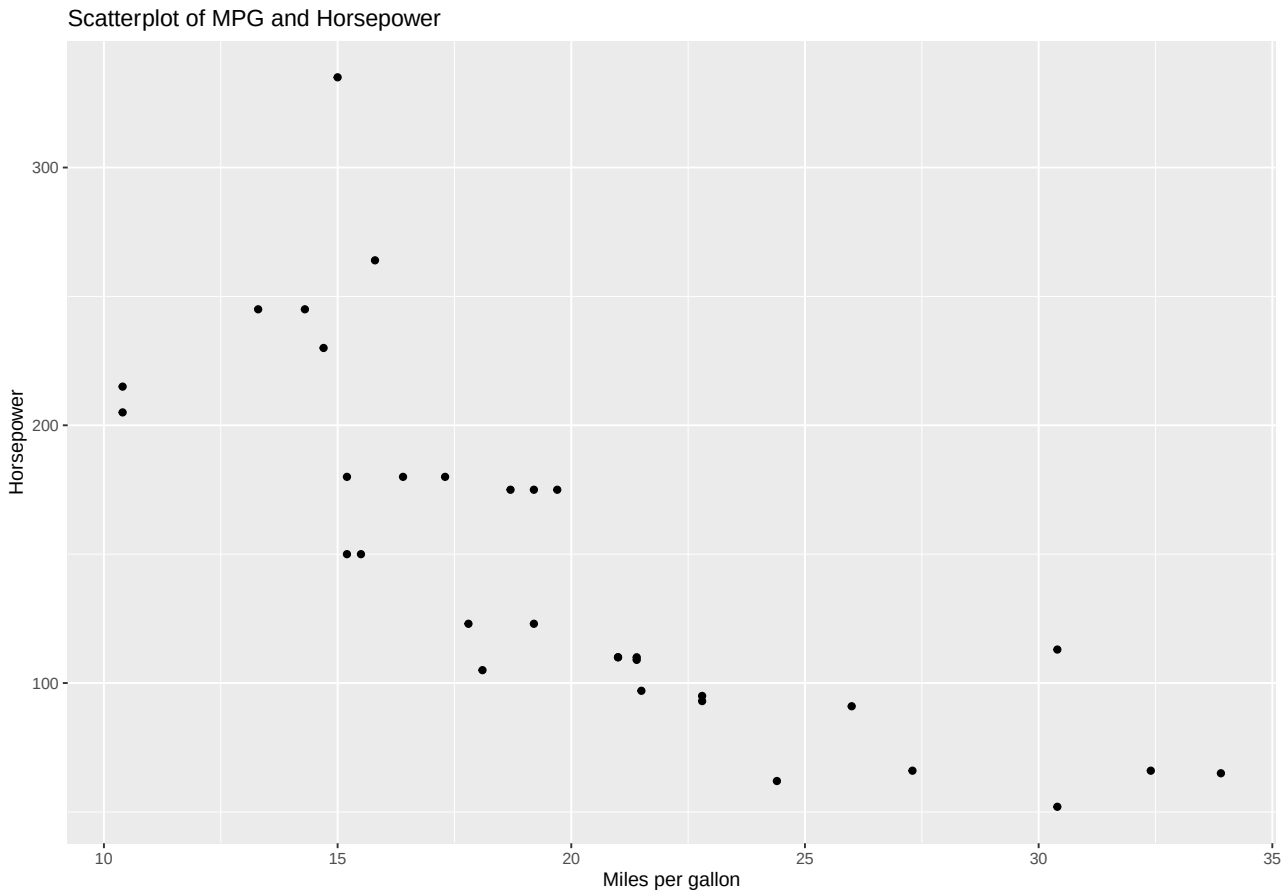
ggplot2: guides and themes



ggplot2: guides and themes

Axis labels and title

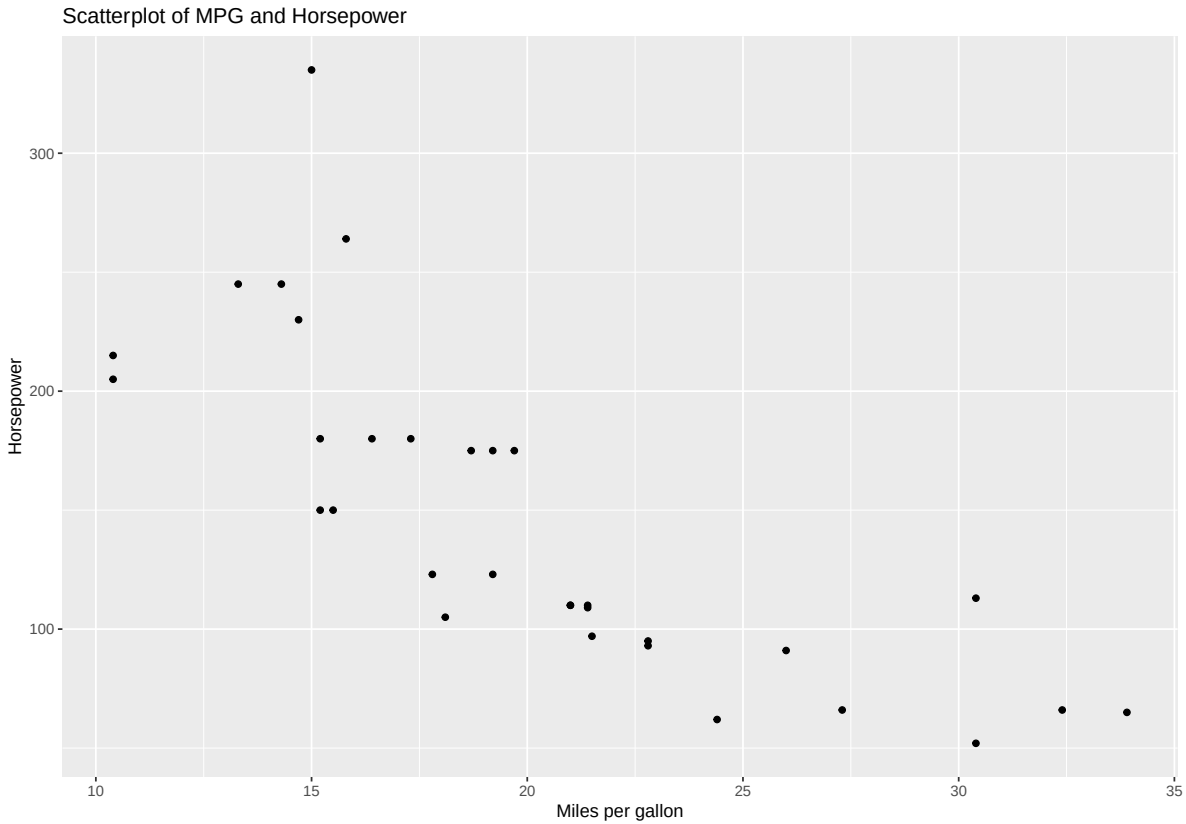
```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  xlab('Miles per gallon') + ylab('Horsepower') +
  ggtitle('Scatterplot of MPG and Horsepower')
```



ggplot2: guides and themes

The same using labs() layer

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  labs(x = 'Miles per gallon', y = 'Horsepower',
       title = 'Scatterplot of MPG and Horsepower')
```



ggplot2: guides and themes

- The appearance of non-data elements of the plot is controlled by the theme system.
- The theme system does not affect how the data is rendered by geoms, or how it is transformed by scales. - They don't change the perceptual properties of the plot, but help to make the plot aesthetically pleasing or match existing style guides.
- Themes give control over things like the fonts in all parts of the plot: the title, axis labels, axis tick labels, legend labels and legend key labels; and the colour of ticks, grid lines and backgrounds (panel, plot, strip and legend).

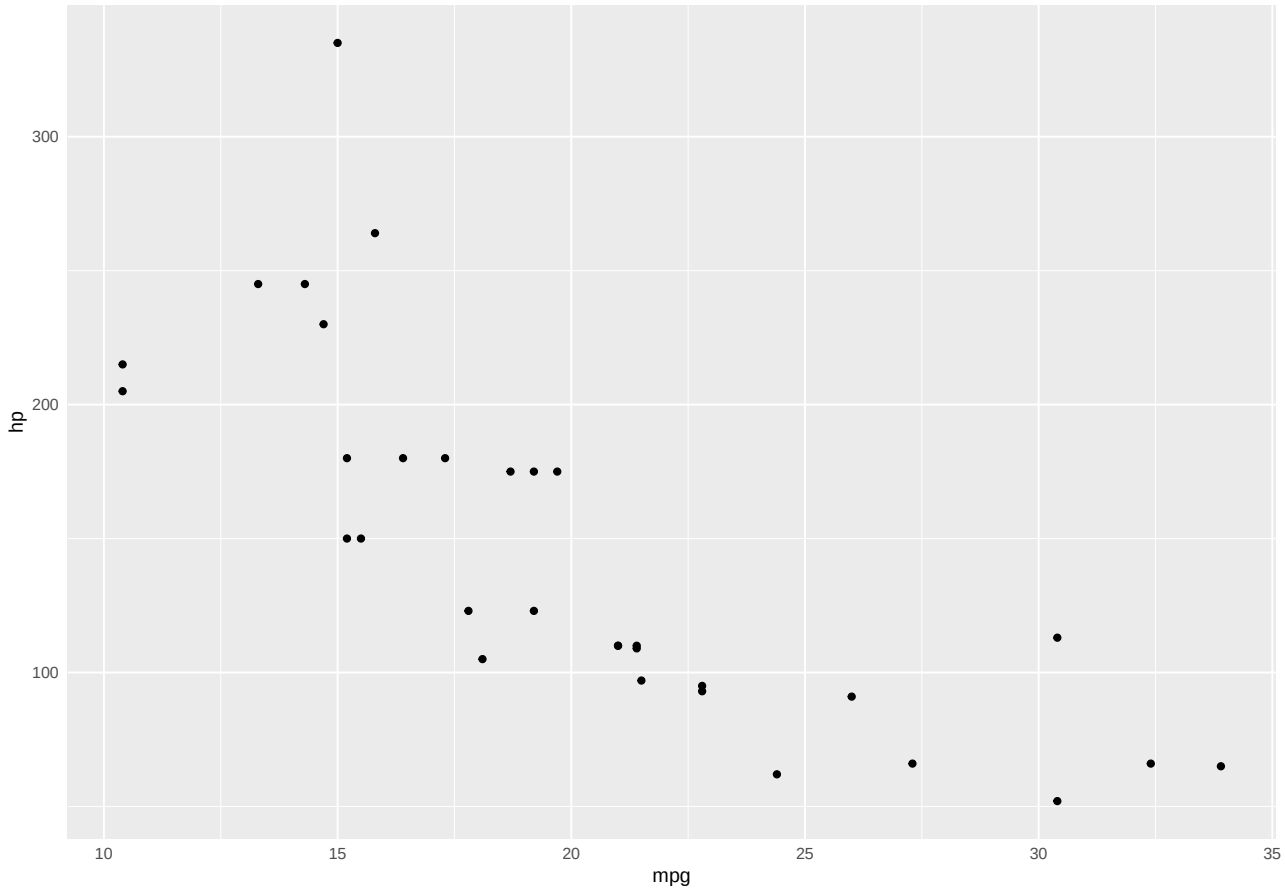
ggplot2: guides and themes

- A theme is made up of multiple elements which control the appearance of a single item on the plot.
- There are three elements that have individual x and y settings: **axis.text, axis.title and strip.text**. - Having a different setting for the horizontal and vertical elements allows you to control how text should appear in different orientations.
- The appearance of each element is controlled by an element function. There are four basic types of built-in element functions: **text, lines and segments, rectangles and blank**.
- Each element function has a set of parameters that control the appearance.

ggplot2: guides and themes

If you want to remove an element, use element_blank() - remove axis ticks

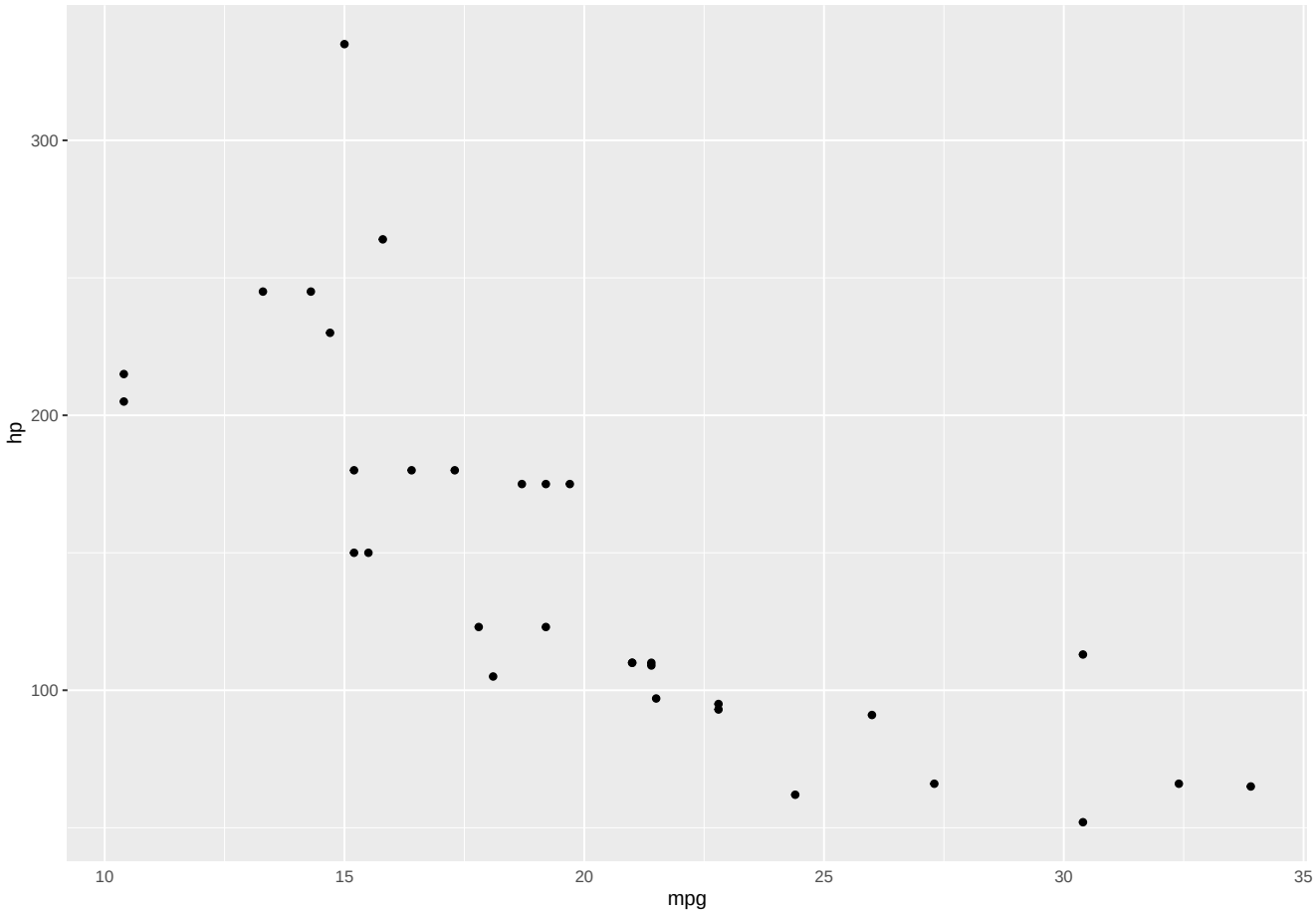
```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme(axis.ticks = element_blank())
```



ggplot2: guides and themes

Control elements of the axis

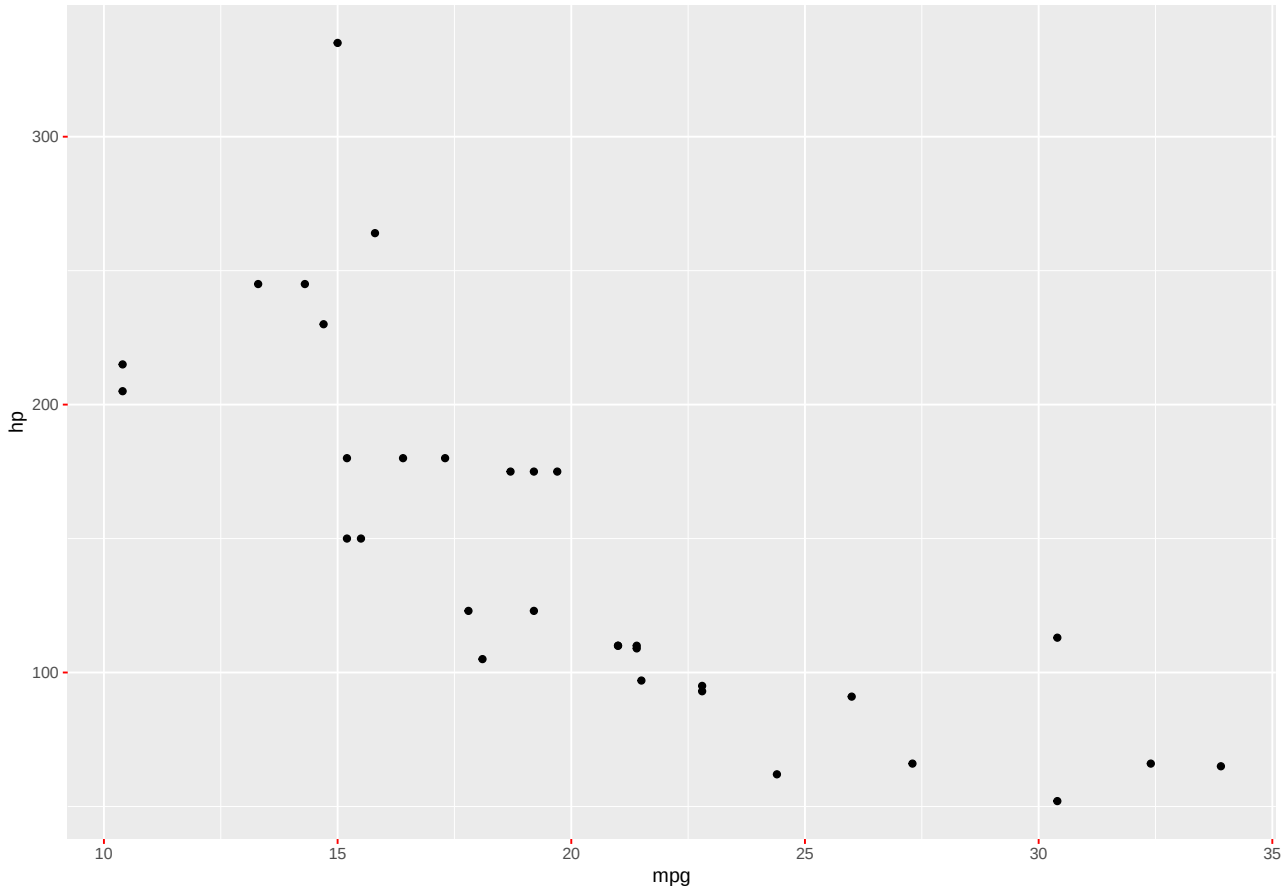
```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme(axis.ticks.x = element_blank())
```



ggplot2: guides and themes

Controlling line elements

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme(axis.ticks = element_line(colour = 'red'))
```



ggplot2: guides and themes

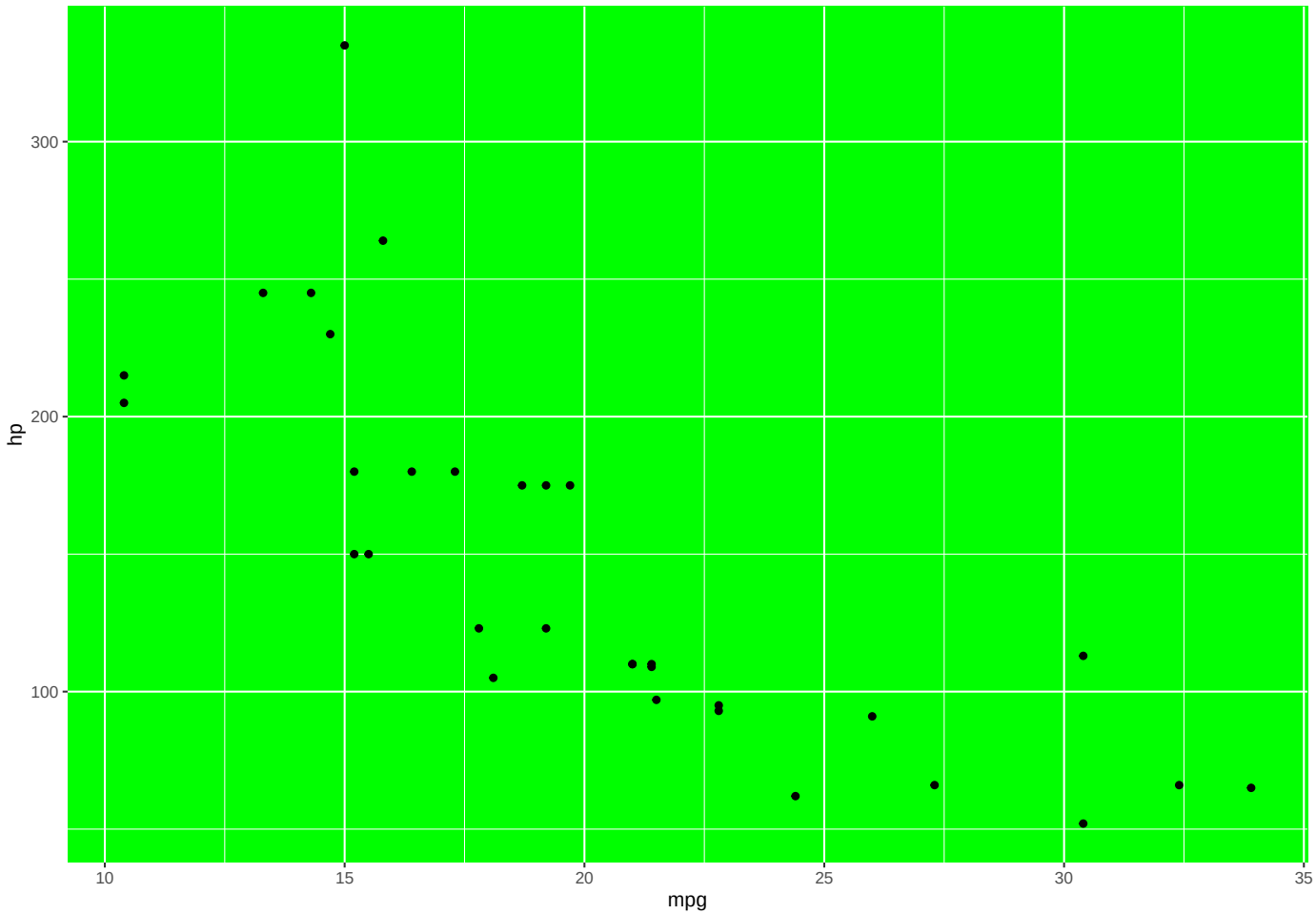
controlling element_text()

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme(axis.text.x = element_text(colour = 'red', size = 16))
```

ggplot2: guides and themes

Change background: element_rect()

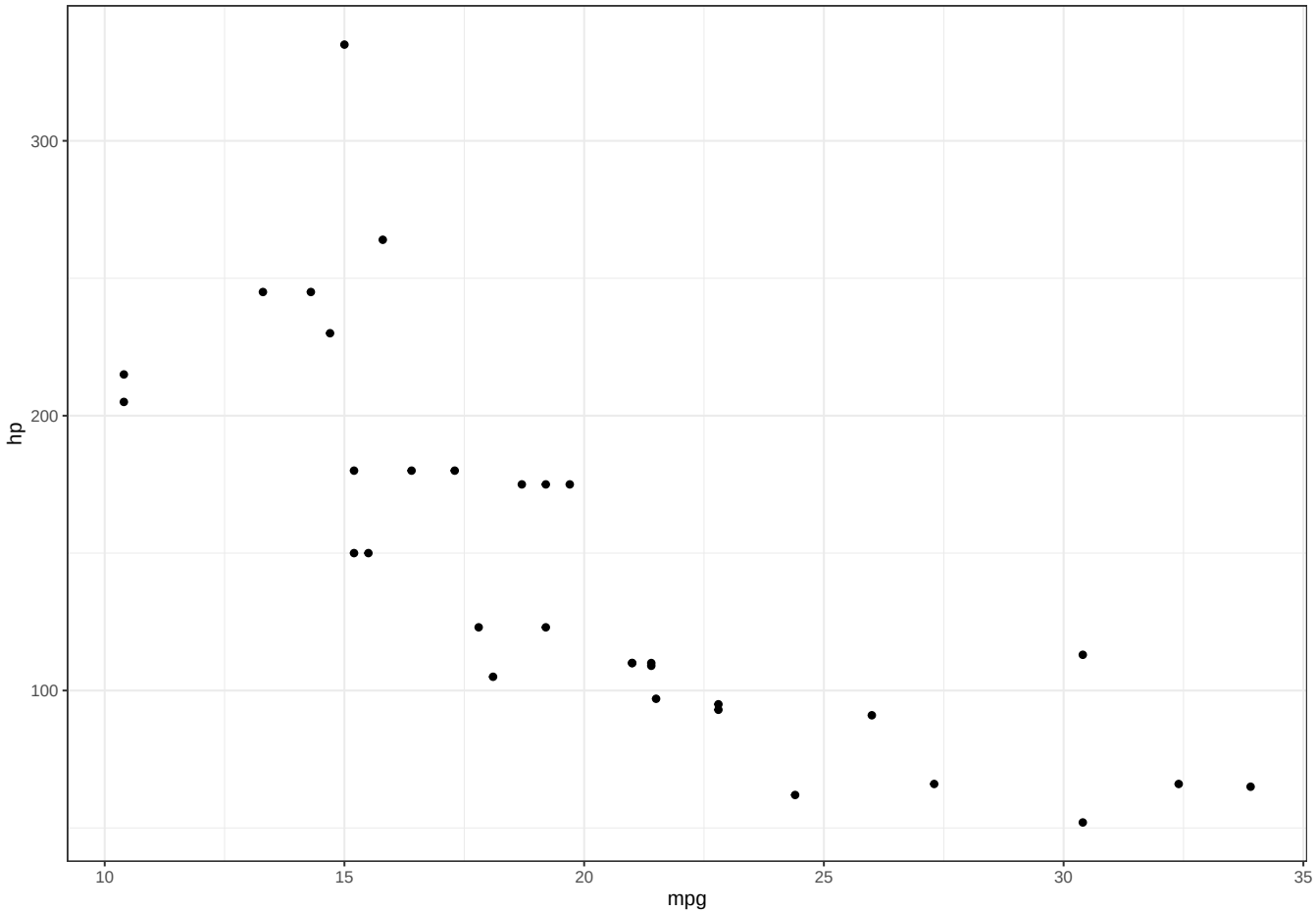
```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme(panel.background  = element_rect(fill = 'green'))
```



ggplot2: guides and themes

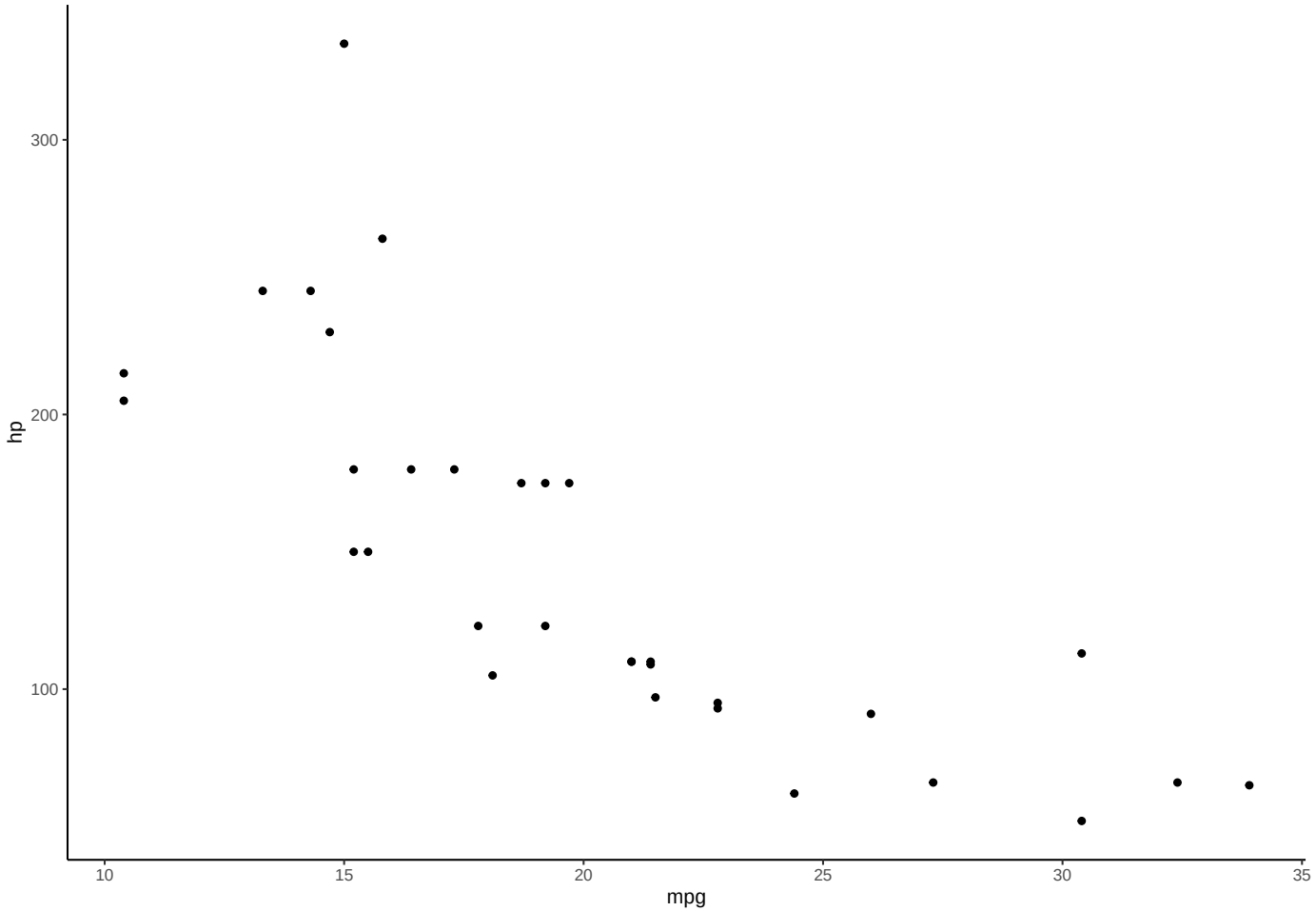
You can use predefined themes

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme_bw()
```



ggplot2: guides and themes

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme_classic()
```



ggplot2: guides and themes

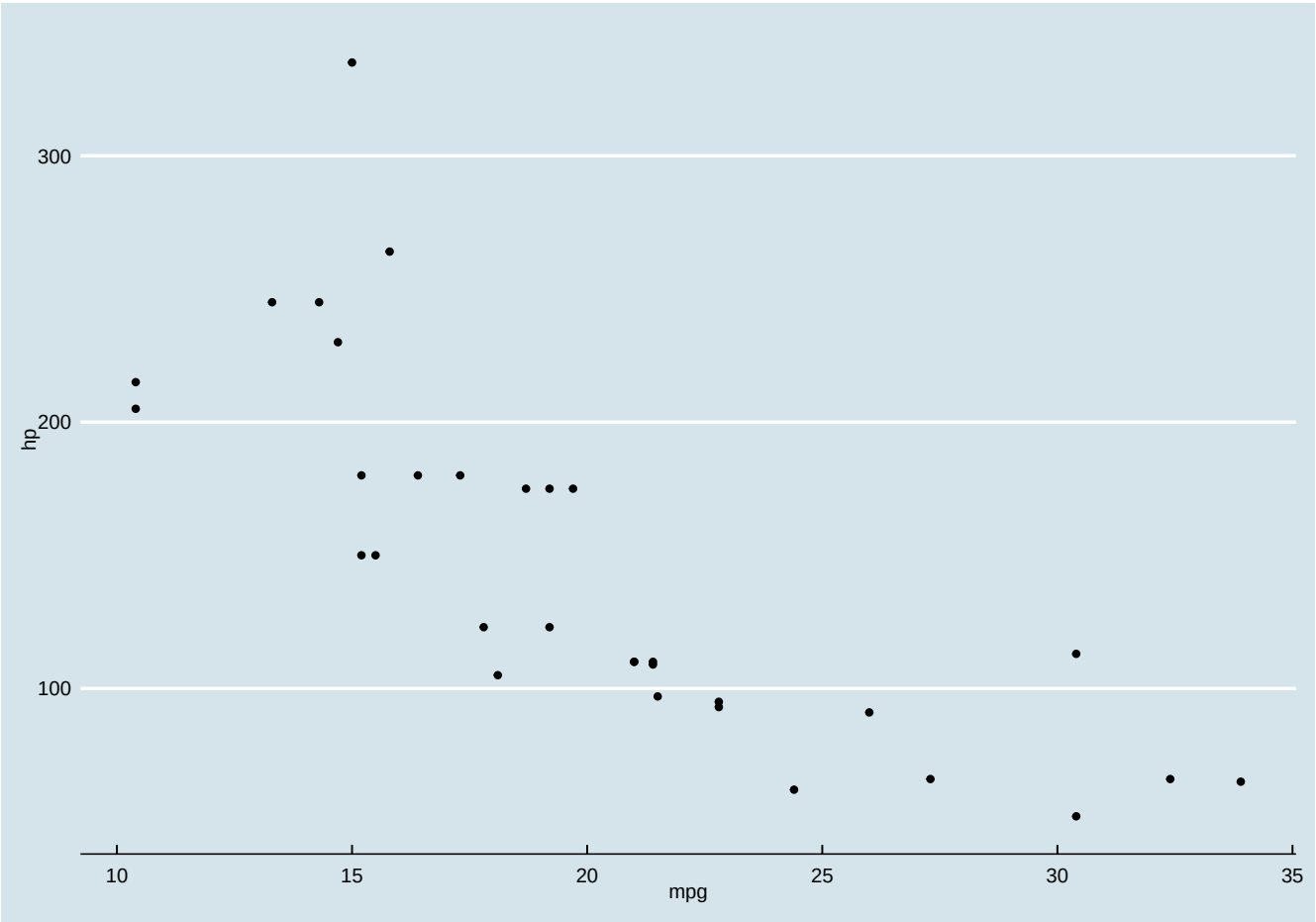
There are lot of themes available in different packages

```
install.packages('ggthemes')
library(ggthemes)
```

ggplot2: guides and themes

The Economist style

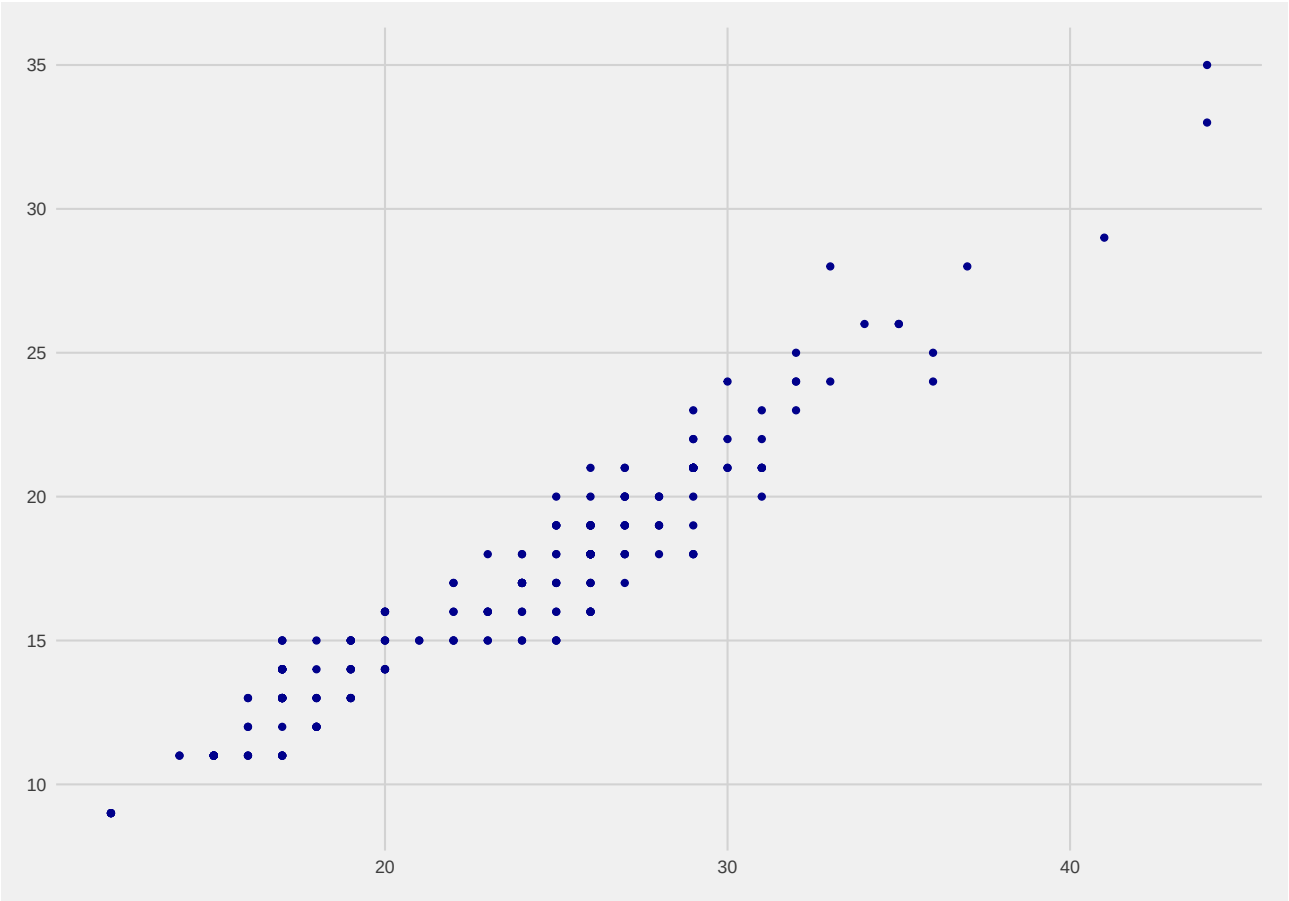
```
plot1 <- ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point()
plot1 + theme_economist()
```



ggplot2: guides and themes

Theme of FiveThirtyEight

```
p1 + theme_fivethirtyeight()
```



ggplot2: guides and themes

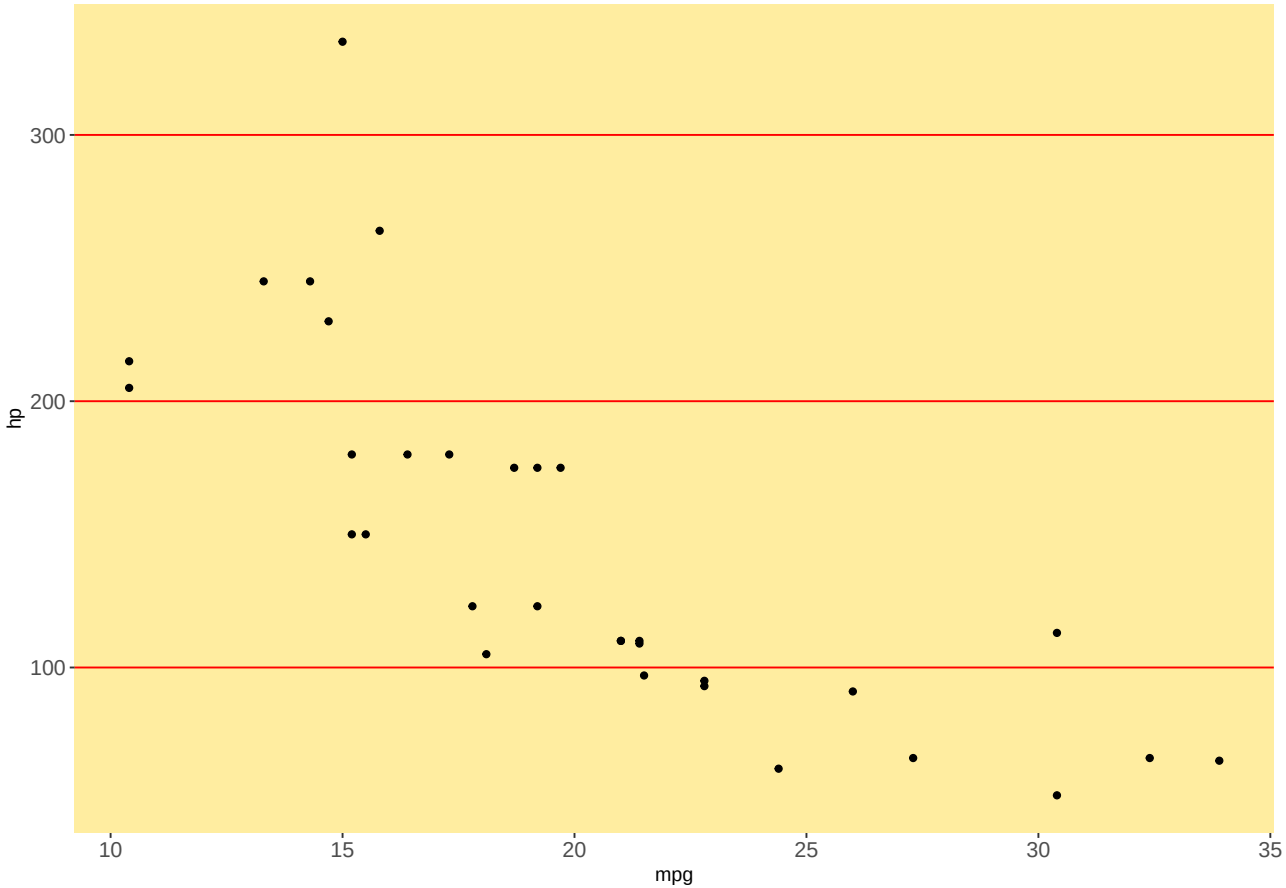
You can also define your own theme, then add it to the plot.

```
theme_my <- theme(
  panel.background = element_rect(fill = '#ffeda0'),
  panel.grid.major.x  = element_blank(),
  panel.grid.major.y = element_line(color = 'red'),
  panel.grid.minor = element_blank(),
  axis.text = element_text(size = 12)
  )
```

ggplot2: guides and themes

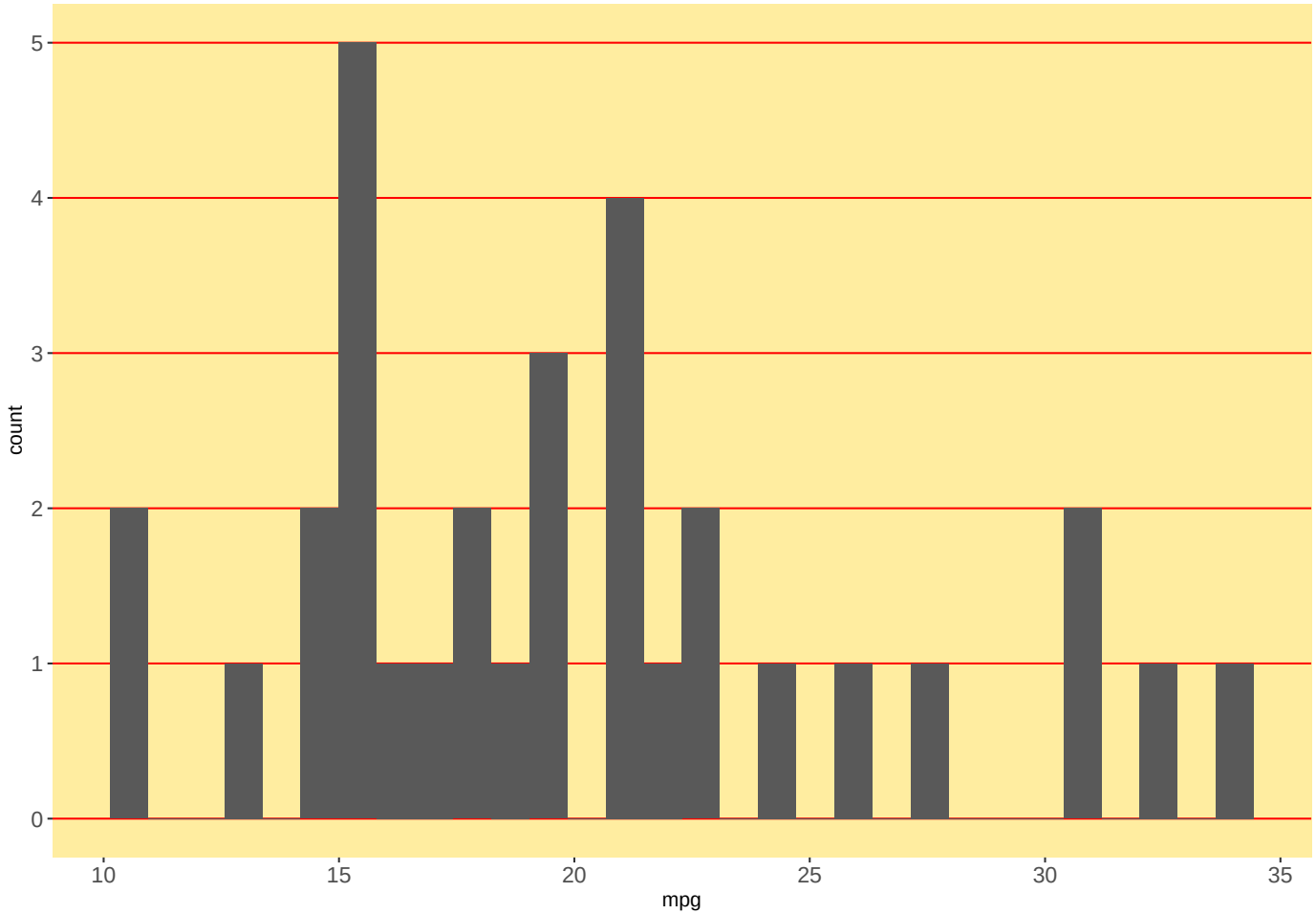
Than add it to the plot

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme_my
```



ggplot2: guides and themes

```
ggplot(mtcars, aes(x = mpg)) + geom_histogram() +
  theme_my
```



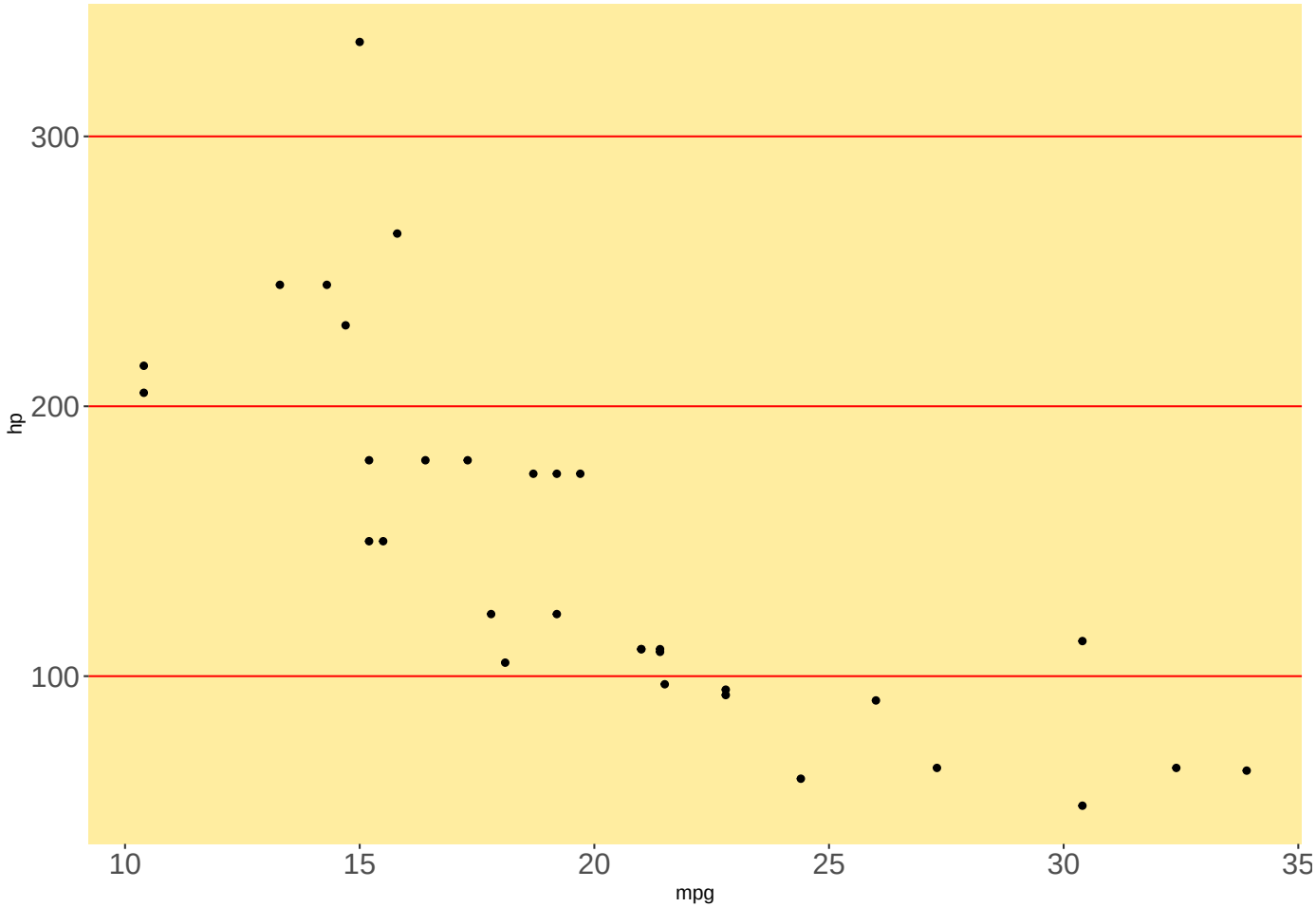
ggplot2: guides and themes

Or you can make it as a function as well

```
theme_my <- function(base_size = 12)
  {
  theme(panel.background = element_rect(fill = '#ffeda0'),
        panel.grid.major.x  = element_blank(),
        panel.grid.major.y = element_line(color = 'red'),
        panel.grid.minor = element_blank(),
        axis.text = element_text(size = base_size))
  }
```

ggplot2: guides and themes

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point() +
  theme_my(base_size = 16)
```



ggplot2: arrange in grid

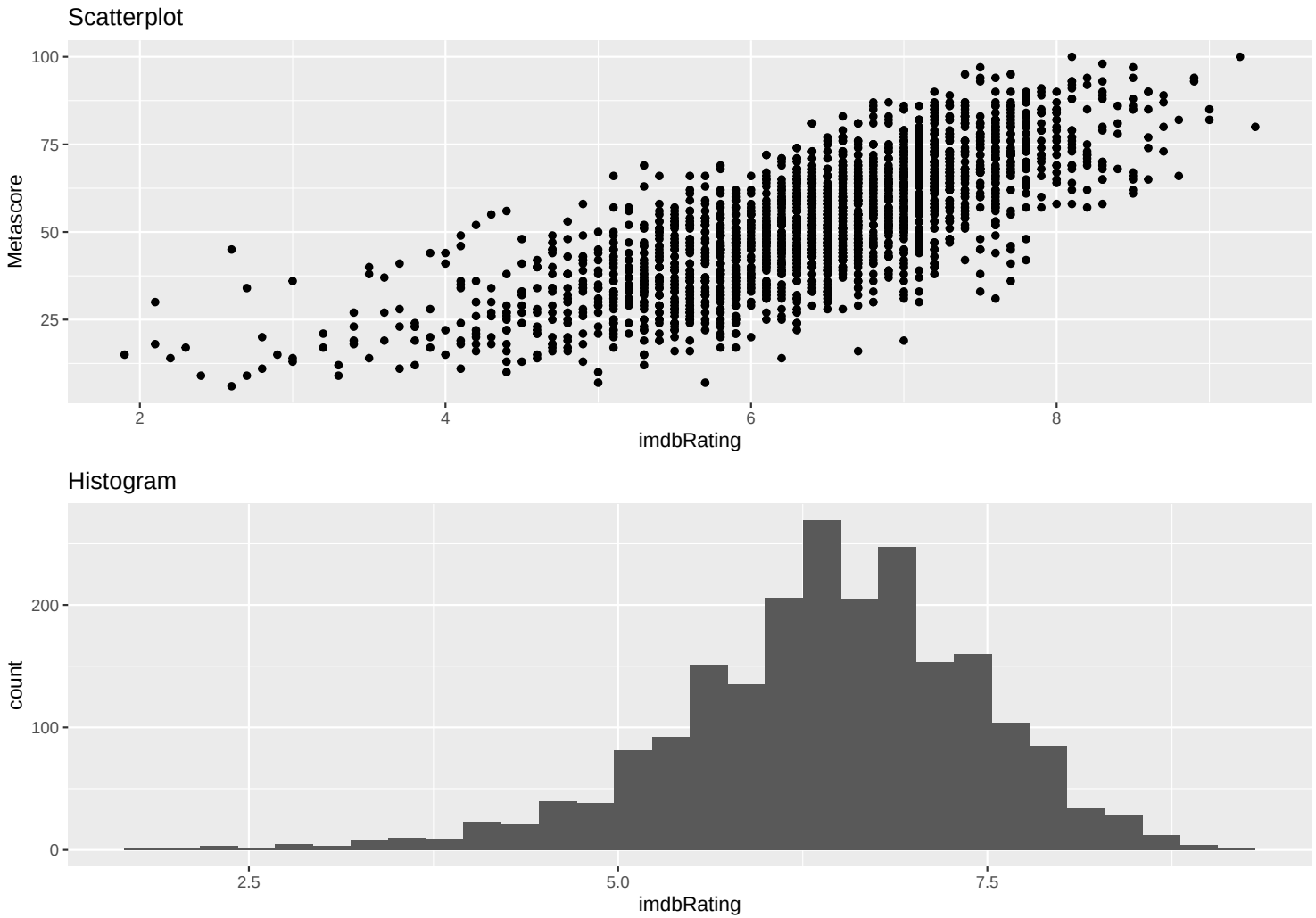
Sometimes you have multiple plots on the same page. You do it using the library gridExtra

```
install.packages('gridExtra')
library(gridExtra)
```

```
p1 <- ggplot(movies_small, aes(x = imdbRating, y = Metascore)) +
      geom_point() + ggtitle('Scatterplot')
p2 <- ggplot(movies_small, aes(x = imdbRating)) +
      geom_histogram() + ggtitle('Histogram')
```

ggplot2: arrange in grid

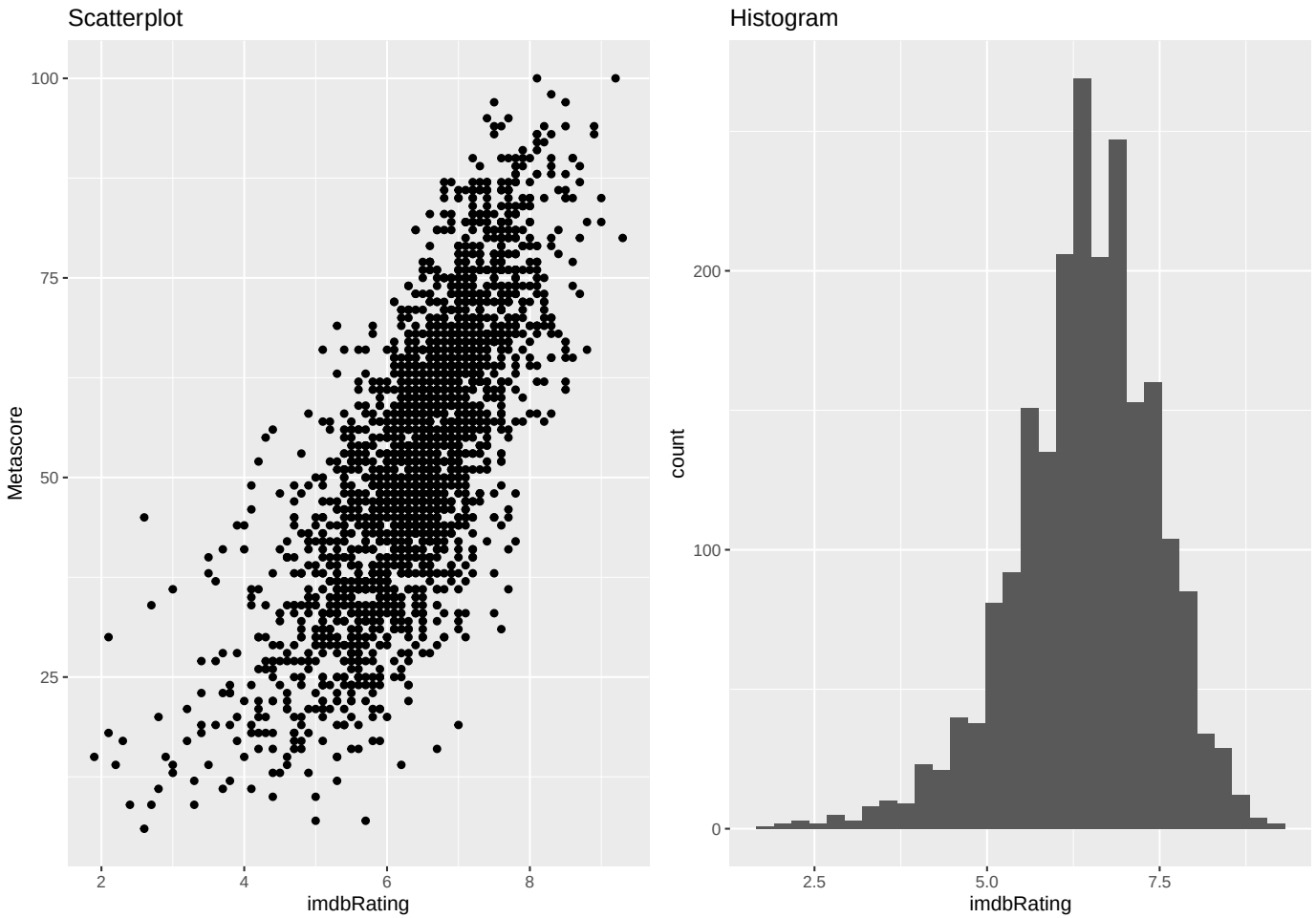
```
grid.arrange(p1, p2)
```



ggplot2: arrange in grid

Arrange by rows

```
grid.arrange(p1, p2, nrow=1)
```



Section 2

Colors in R

Colors in R

- You can specify colors in R by index, name, hexadecimal, or RGB.
- For example col=1, col="white", and col="#FFFFFF" are equivalent.
- You can find names, numbers and codes of colors here.

```
length(colors())
```

```
## [1] 657
```

```
colors()[1:10]
```

```
##  [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
##  [5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
##  [9] "aquamarine1"   "aquamarine2"
```

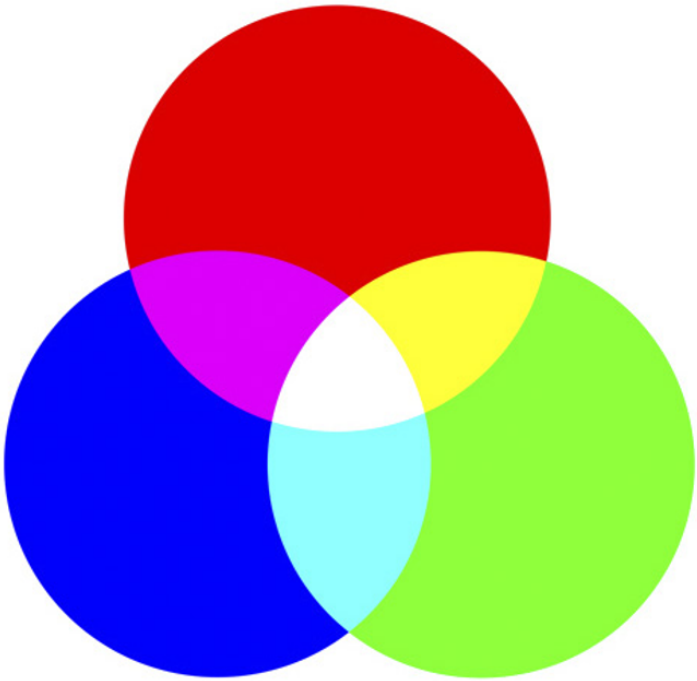
Colors in R

RGB color model

- Computers create the colors we see on a monitor by combining 3 primary colors of light:
  - red
  - green
  - blue
- This combination is known as RGB color model
- Each color light is also referred to as a channel

Colors in R

A computer screen displays a color by combining **red** light, **green** light and **blue** light, the so-called RGB model.



Colors in R

Any color you see on a monitor can be described by a series of 3 numbers (in the following order):

- a red value
- a green value
- a blue value
- e.g. red=30, green=200, blue=180

Colors in R

- The amount of light in each color channel is typically described on a scale from 0 (none) to 255 (full-blast)
- Alternatively, scales can be provided as percent values from 0 (none) to 1 (100%)

Colors in R

Some reference colors:

RGB Values	Color
(255, 0, 0)	red
(0, 255, 0)	green
(0, 0, 255)	blue
(0, 0, 0)	black
(255, 255, 255)	white

The closer the three values get to 255 (100%), the closer the resulting color gets to white

Colors in R

Look at the first six colors

```
col2rgb(col=1:6)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6]
## red      0  223   97   34   40  205
## green    0   83  208  151  226   11
## blue     0  107   79  230  229  188
```

Colors in R

Get RGB and Hex notation for color gold

```
col2rgb("gold")
```

```
##       [,1]
## red    255
## green  215
## blue     0
```

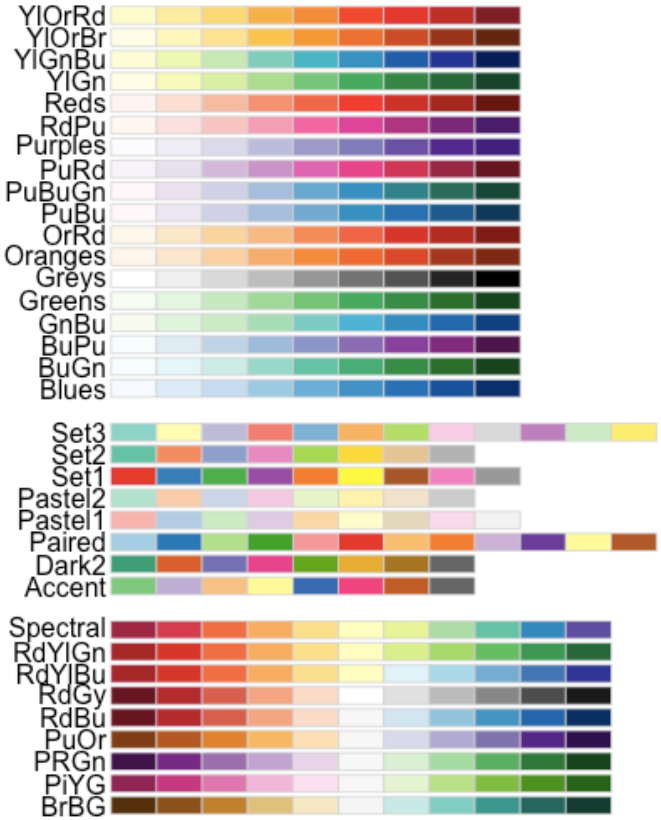
And the hexadecimal notation

```
rgb(255,215,0, maxColorValue = 255)
```

```
## [1] "#FFD700"
```

Colors in R

RColorBrewer allows you to choose colors from a palette



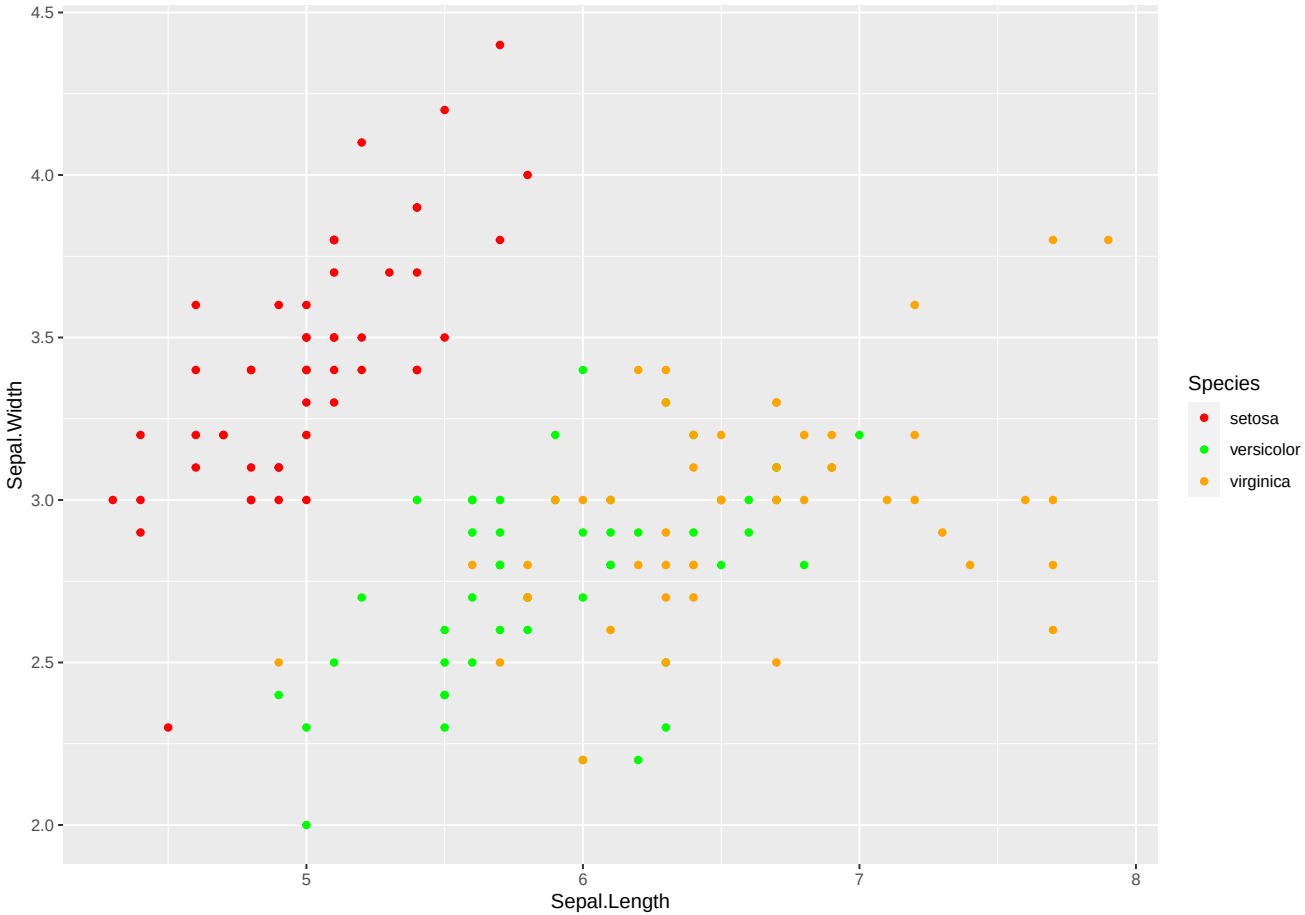

```
brewer.pal(n=5, name = "Set3")
```

```
## [1] "#8DD3C7" "#FFFFB3" "#BEBADA" "#FB8072" "#80B1D3"
```

Colors in R

Change the colors manually, specifying colors by names

```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species))+
  geom_point()+scale_color_manual(values = c("red", "green", "orange"))
```



Colors in R

The Isle of dogs


```
install.packages("wesanderson")
library(wesanderson)
wes_palette(name="IsleofDogs2")
```

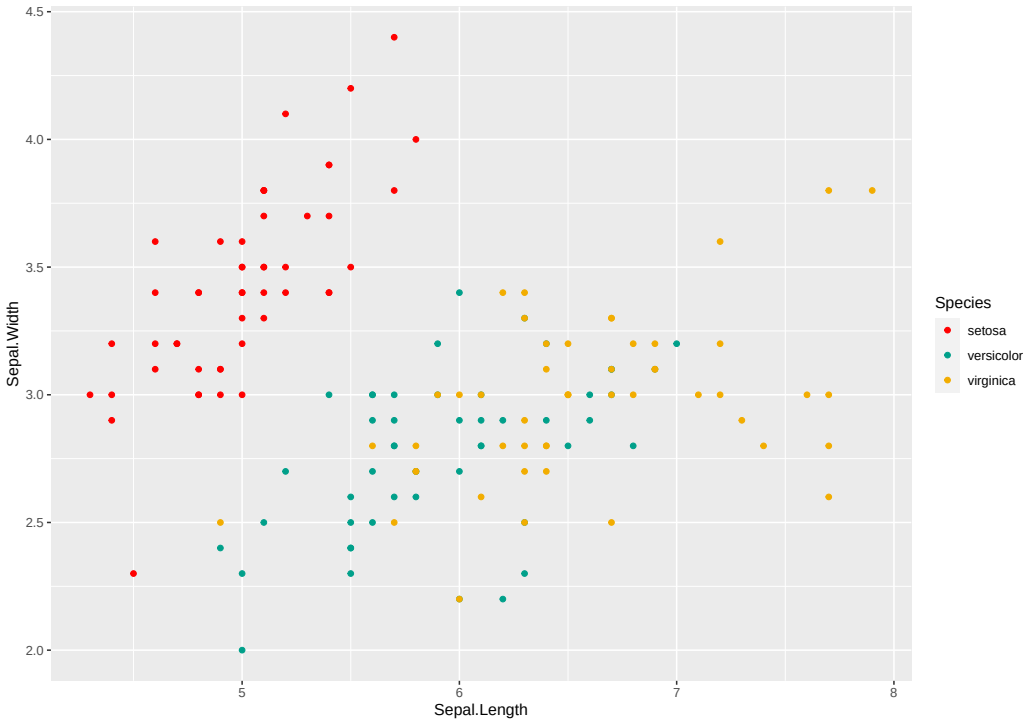
Colors in R

Using the Hex notation and colors from "The Darjeeling Limited"

```
wes_palettes$Darjeeling1
```

```
## [1] "#FF0000" "#00A08A" "#F2AD00" "#F98400" "#5BBCD6"
```

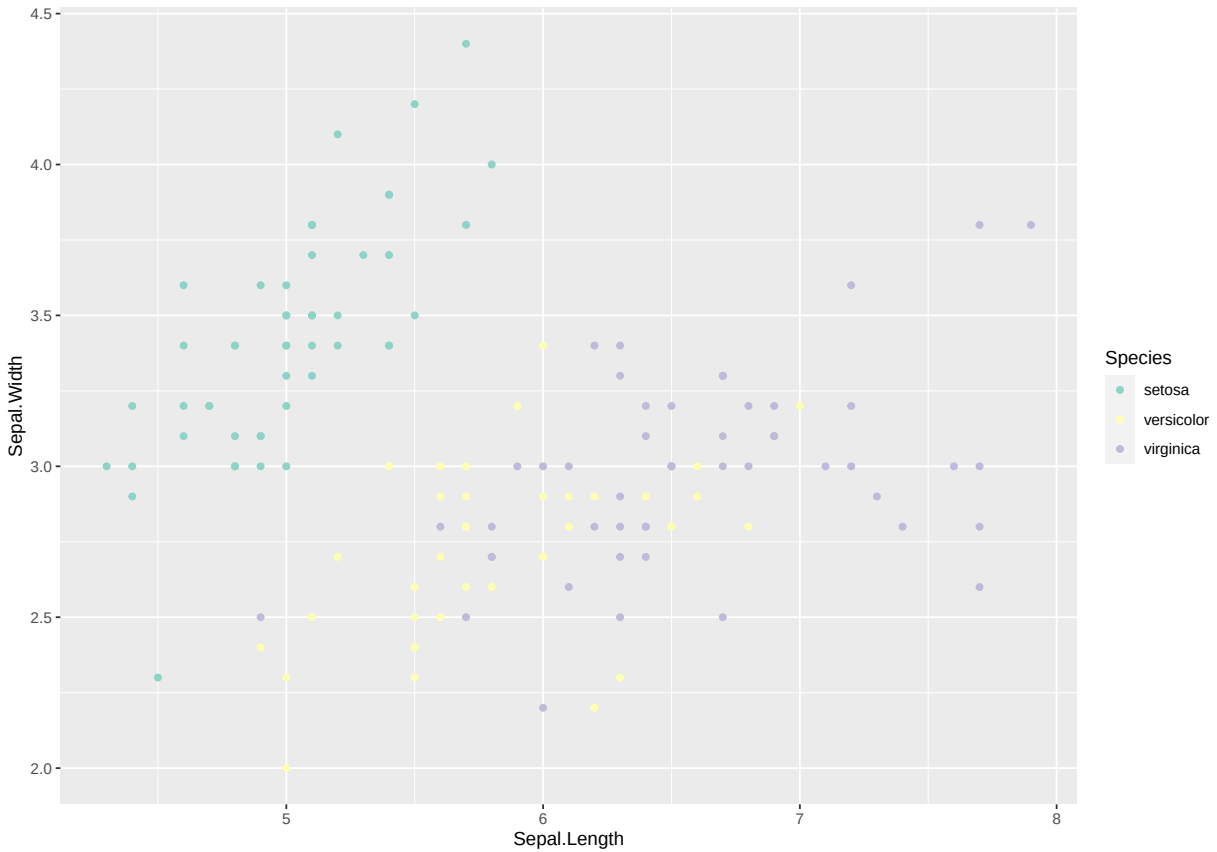
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species))+
  geom_point()+
  scale_color_manual(values = c("#FF0000", "#00A08A", "#F2AD00"))
```



Colors in R

You can choose colors from the RColorBrewer palettes

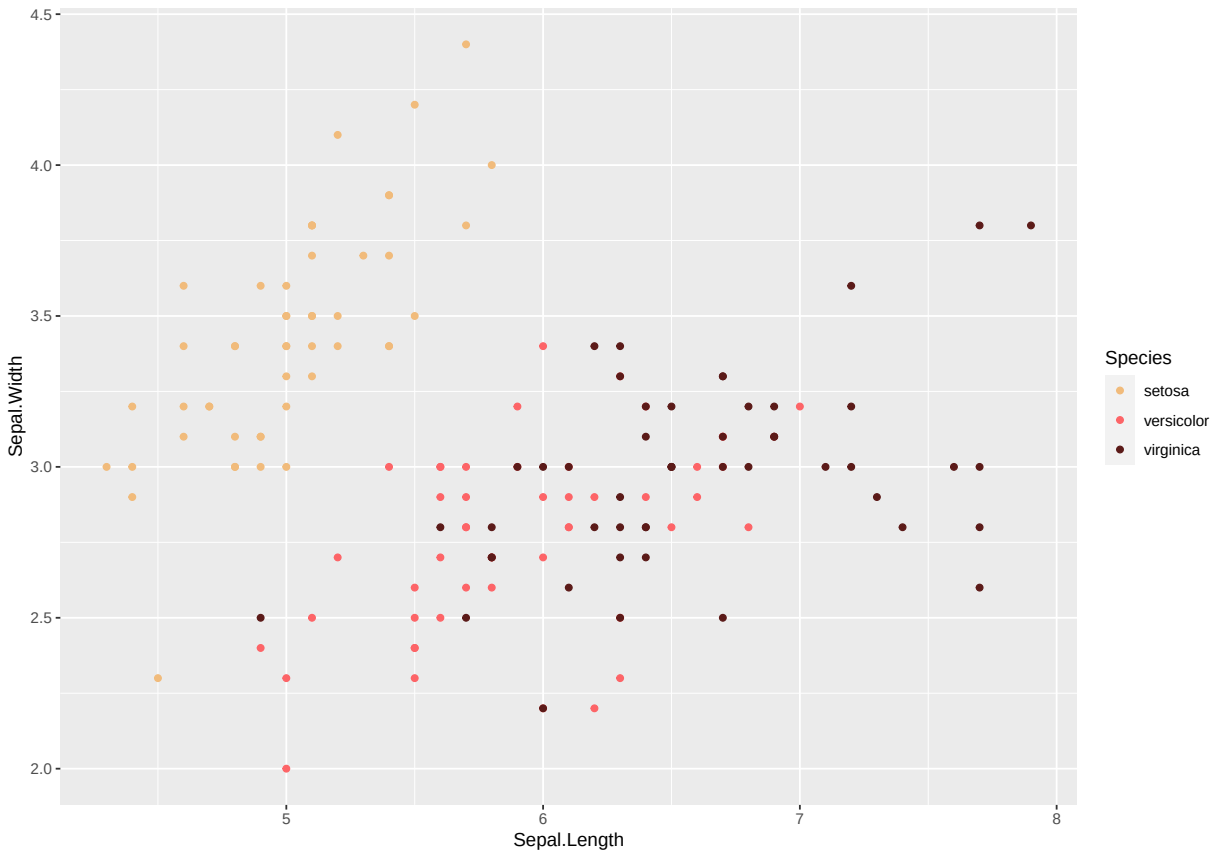
```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species))+
  geom_point()+scale_color_brewer(palette="Set3")
```



Colors in R

wesanderson package allows to directly access the palettes as well, colors from The Grand Budapest Hotel

```
ggplot(data=iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species))+
  geom_point()+scale_color_manual(values=wes_palette("GrandBudapest1"))
```



Section 3

Geometric objects for continuous variable

Geometric objects for continuous variable

Histogram

- A histogram is an accurate representation of the distribution of numerical data.
- To construct a histogram, the first step is to "bin" the range of values - that is, divide the entire range of values into a series of intervals - and then count how many values fall into each interval.
- The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent and are often (but are not required to be) of equal size.

Geometric objects for continuous variable

Movies dataset

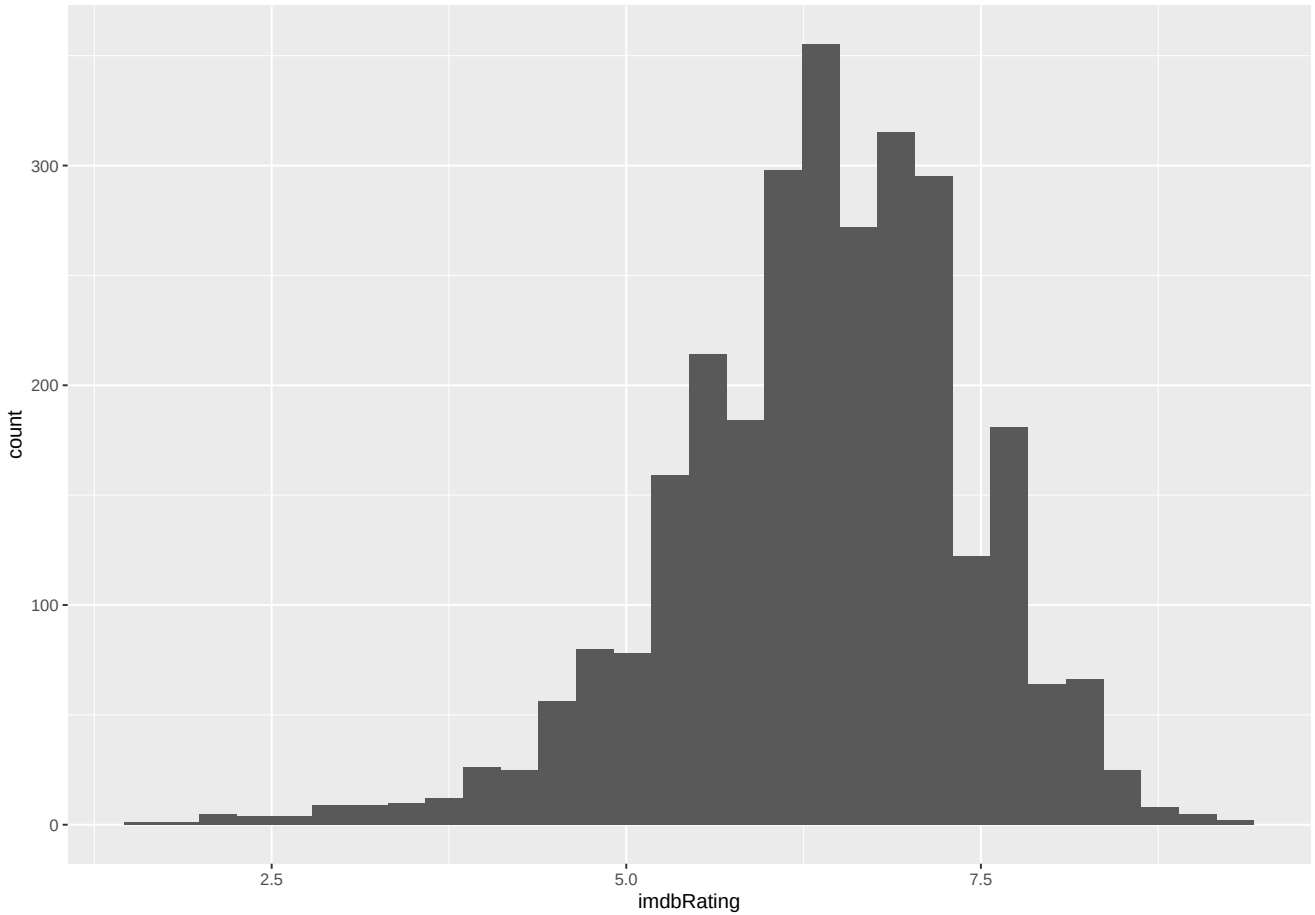
```
movies <- read.csv("movies3.csv")
colnames(movies)
```

```
##  [1] "title"              "genre_first"        "year"
##  [4] "duration"           "gross_adjusted"     "budget_adjusted"
##  [7] "gross"              "budget"             "cast_facebook_likes"
## [10] "reviews"            "index"              "Rated"
## [13] "Genre"              "Director"           "Writer"
## [16] "Actors"             "Plot"               "Language"
## [19] "Country"            "Awards"             "Metascore"
## [22] "imdbRating"         "imdbVotes"          "Production"
## [25] "DVD"                "Release"            "Release_Month"
## [28] "Release_Day"        "Release_year"       "OscarWon"
## [31] "OtherWin"           "OscarNom"           "OtherNom"
```

Geometric objects for continuous variable

Create basic histogram

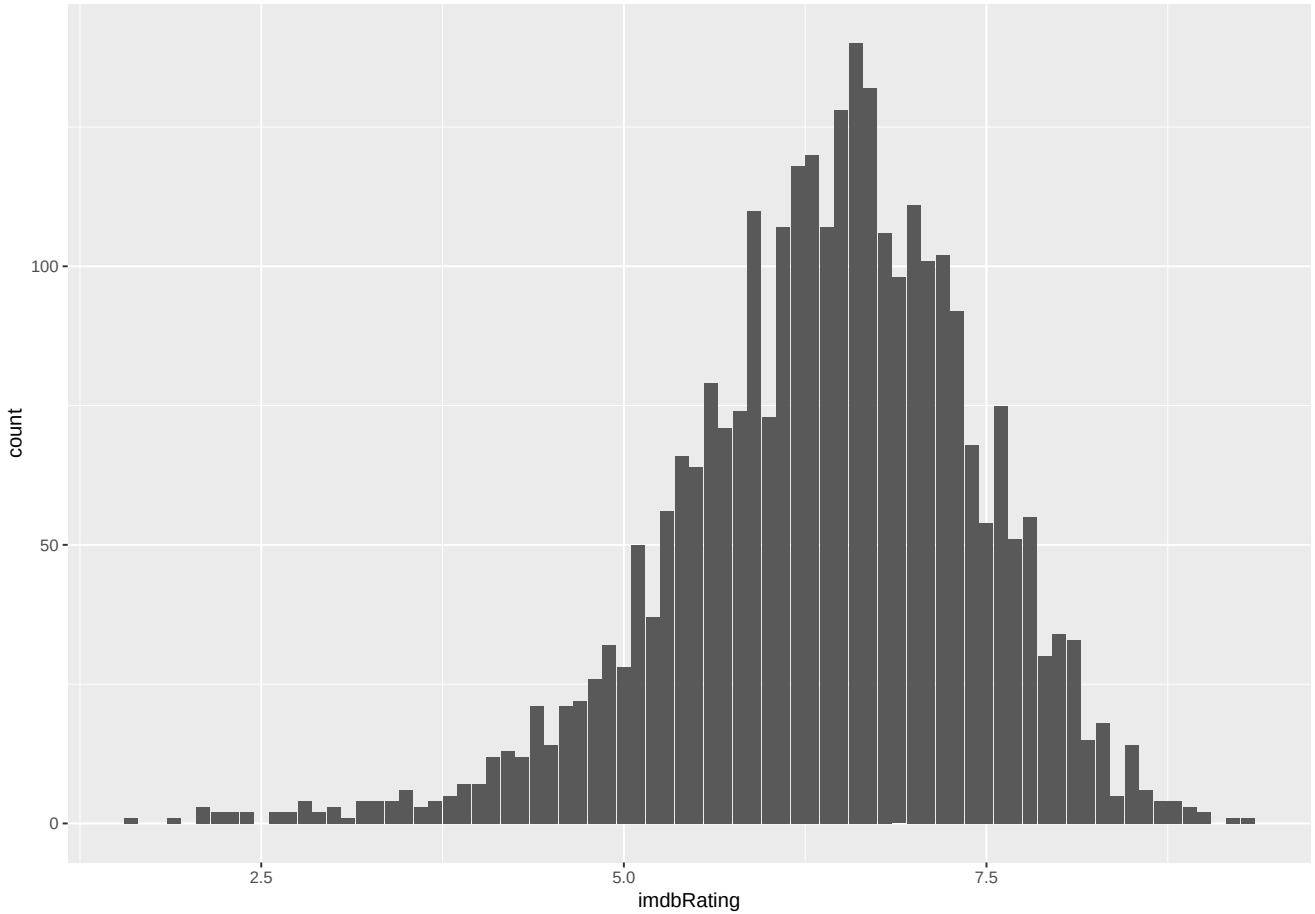
```
ggplot(movies, aes(x=imdbRating))+geom_histogram()
```



Geometric objects for continuous variable

- You can get more or less the same with geom_bar
- With geom_histogram you can control for number of bins or binwidth

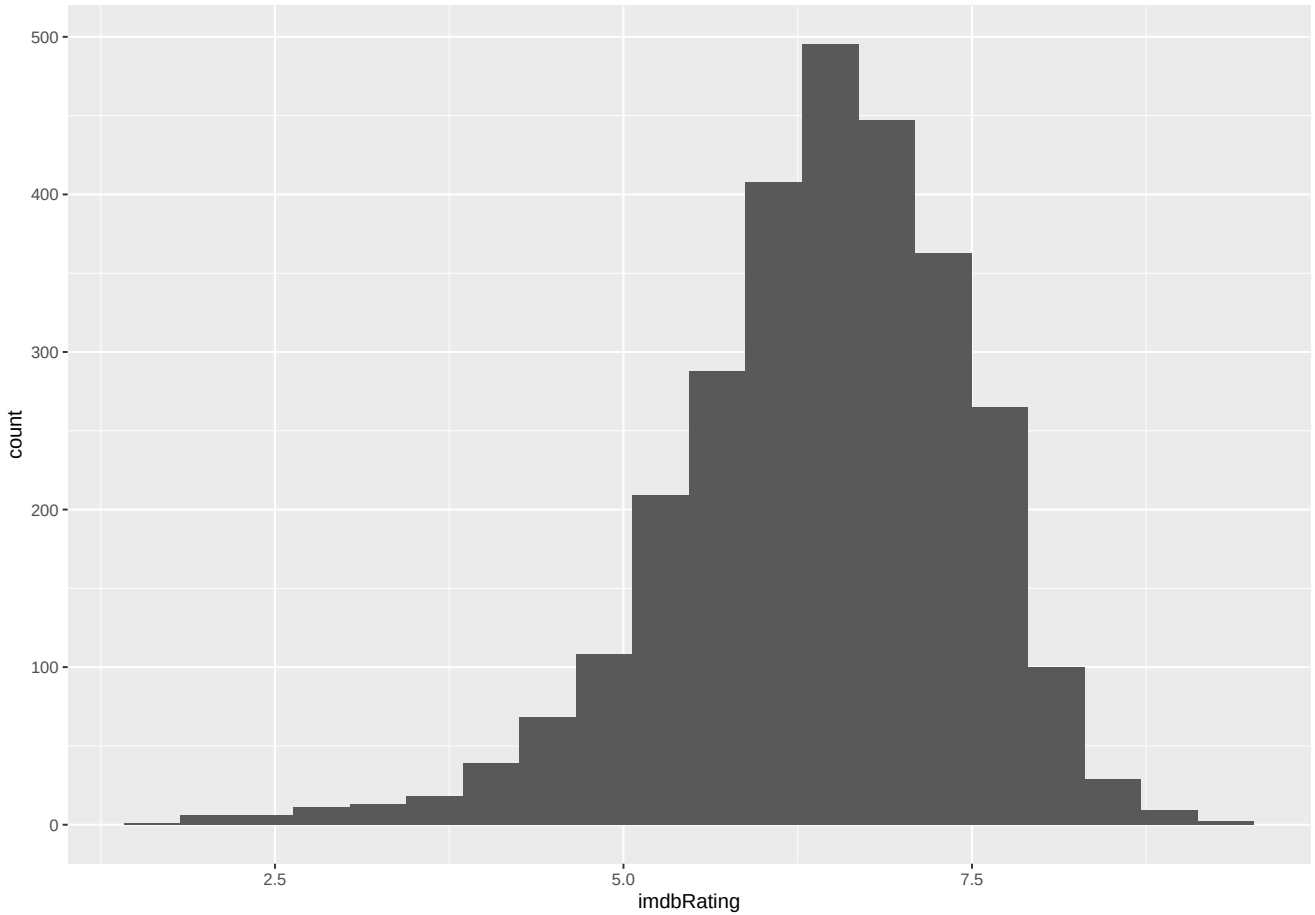
```
ggplot(movies, aes(x=imdbRating))+geom_bar()
```



Geometric objects for continuous variable

Play with the number of bins to see how the chart is changing

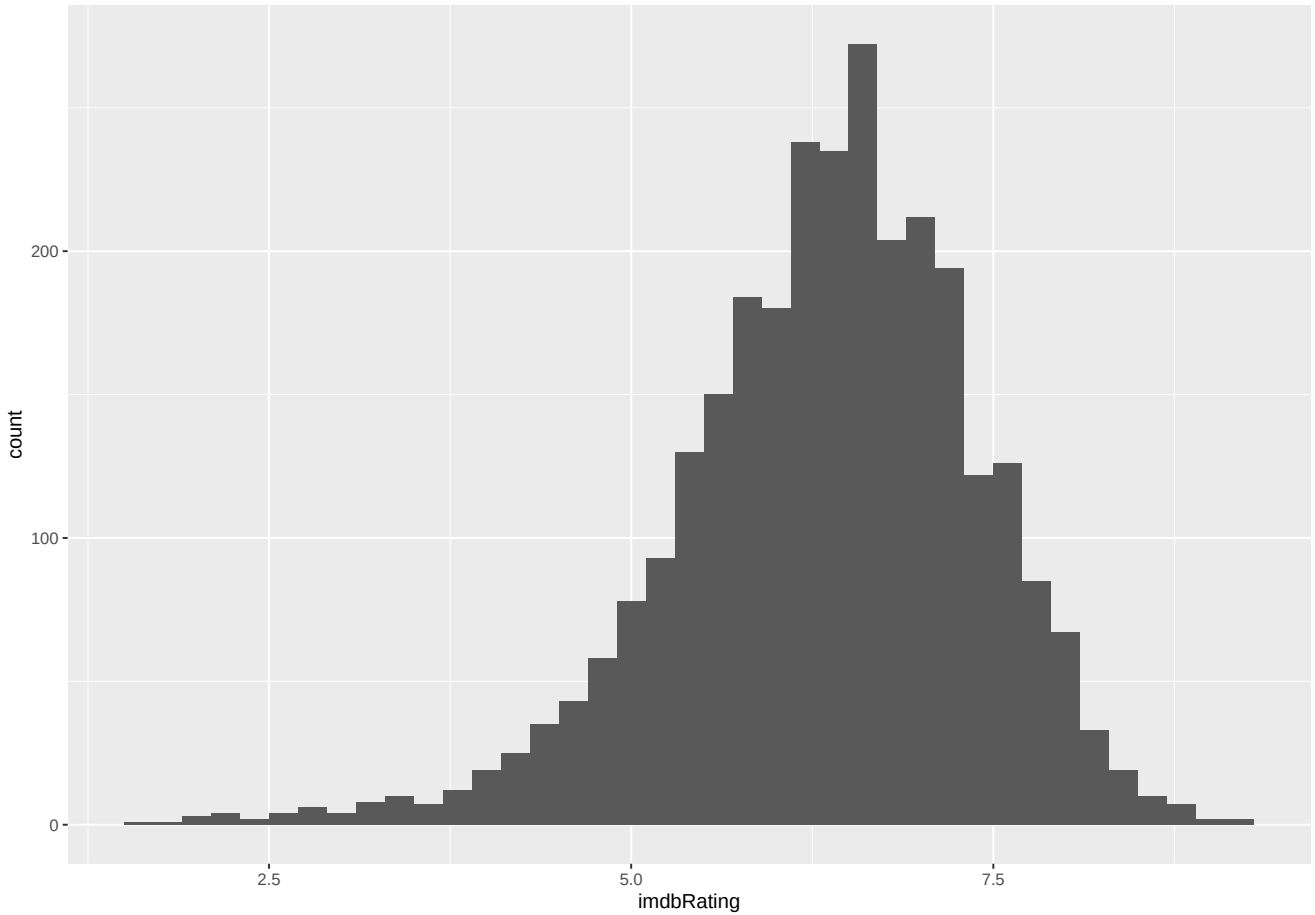
```
ggplot(movies, aes(x=imdbRating))+geom_histogram(bins = 20)
```



Geometric objects for continuous variable

Or play with the binwidth

```
ggplot(movies, aes(x=imdbRating))+geom_histogram(binwidth = 0.2)
```



Section 4

Visualizing categorical data

Visualizing categorical data

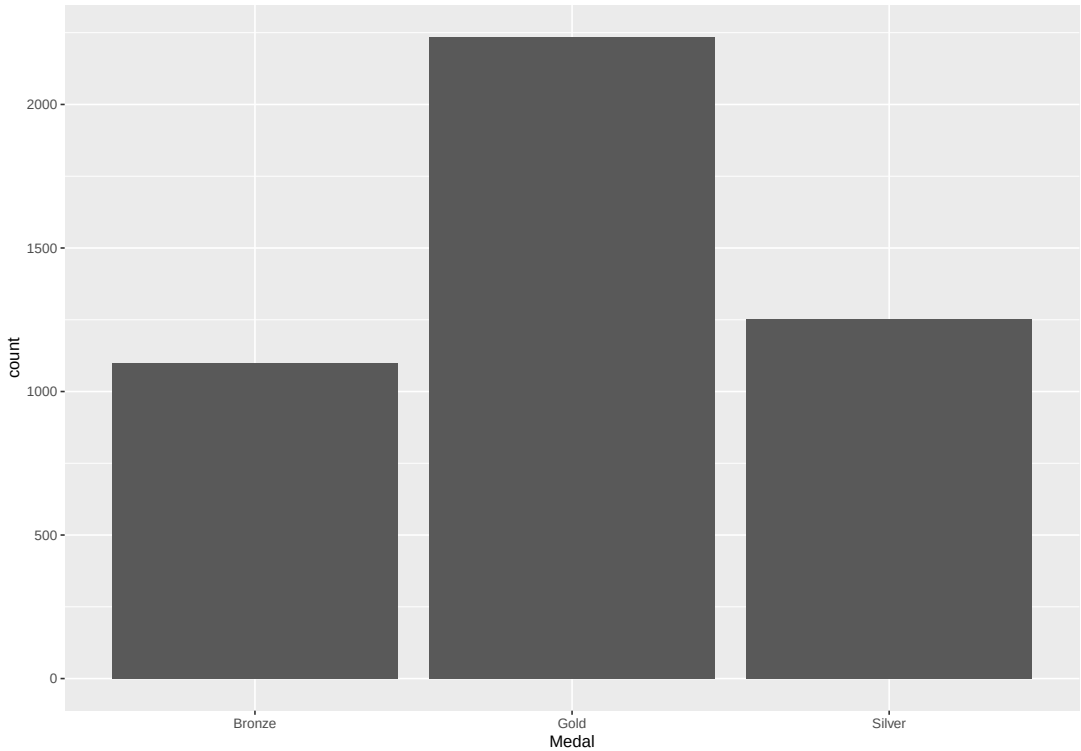
- A **bar chart** or **bar graph** is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.
- A bar graph shows comparisons among *discrete categories*. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value. Some bar graphs present bars clustered in groups of more than one, showing the values of more than one measured variable.

Visualizing categorical data

Bar chart for USA medals from Summer Olympic games

```
summer <- read.csv("summer.csv")
summer_usa <- summer[summer$Country=="USA",]
```

```
ggplot(summer_usa, aes(x=Medal))+geom_bar()
```



Visualizing categorical data

- Let's make the chart more visually appealing
- First relevel the factor levels for medals

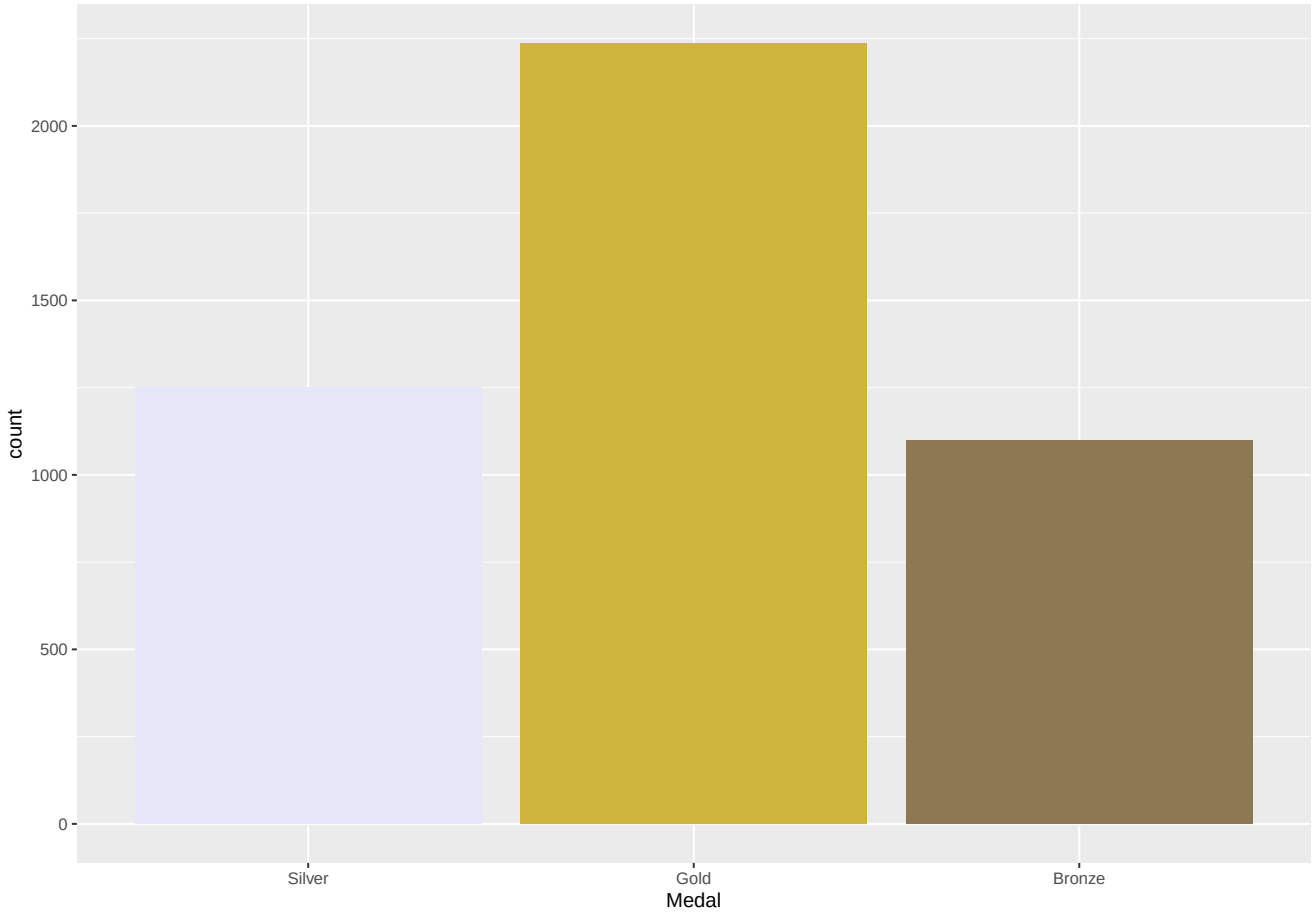
```
levels(factor(summer_usa$Medal))
```

```
## [1] "Bronze" "Gold"   "Silver"
```

```
summer_usa$Medal <- factor(summer_usa$Medal,
                           levels=c("Silver", "Gold","Bronze"))
```

Visualizing categorical data

```
ggplot(summer_usa, aes(x=Medal)) +
  geom_bar(fill = c("#E6E8FA", "#CFB53B", "#8C7853"))
```



Visualizing categorical data

- Build already aggregated data frame with relative frequencies (%) for each type of medal
- Use ggplot to make the barplot

```
usa_medals <- data.frame(Medal=c("Silver", "Gold", "Bronze"),
                         Percentage = c(0.27,0.49, 0.24))
```

You are going to get the following error

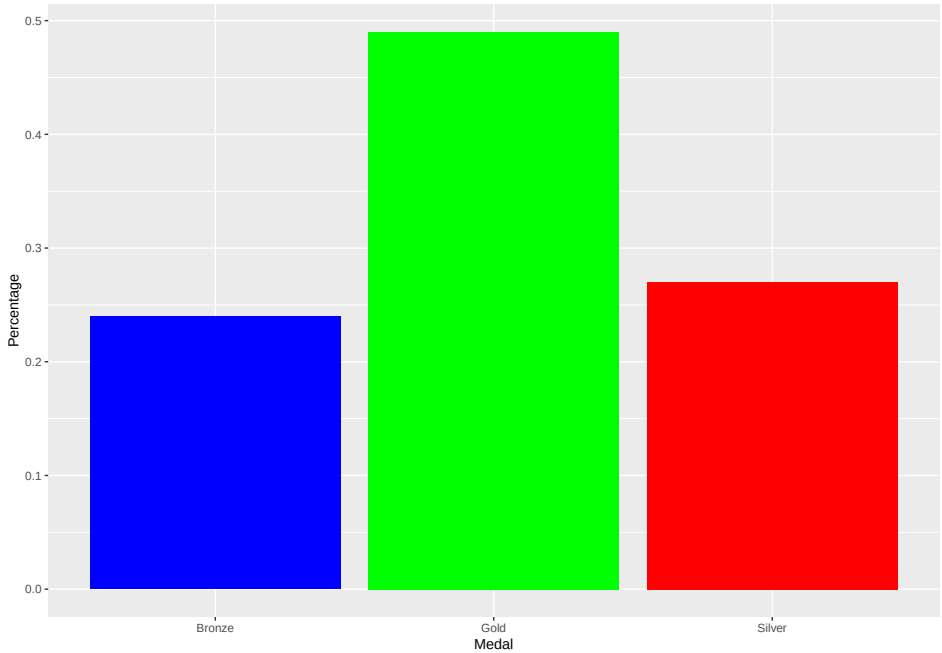
```
ggplot(usa_medals, aes(x=Medal, y=Percentage)) + geom_bar()
```

```
## Error: stat_count() can only have an x or y aesthetic.
```

Visualizing categorical data

- Another component of grammar of graphics is the statistical transformation.
- For geom_bar by default it is stat_count() as we are constructing a frequency plot.
- To solve the issue, use stat="identity"
- Identity function: $f(x) = x$

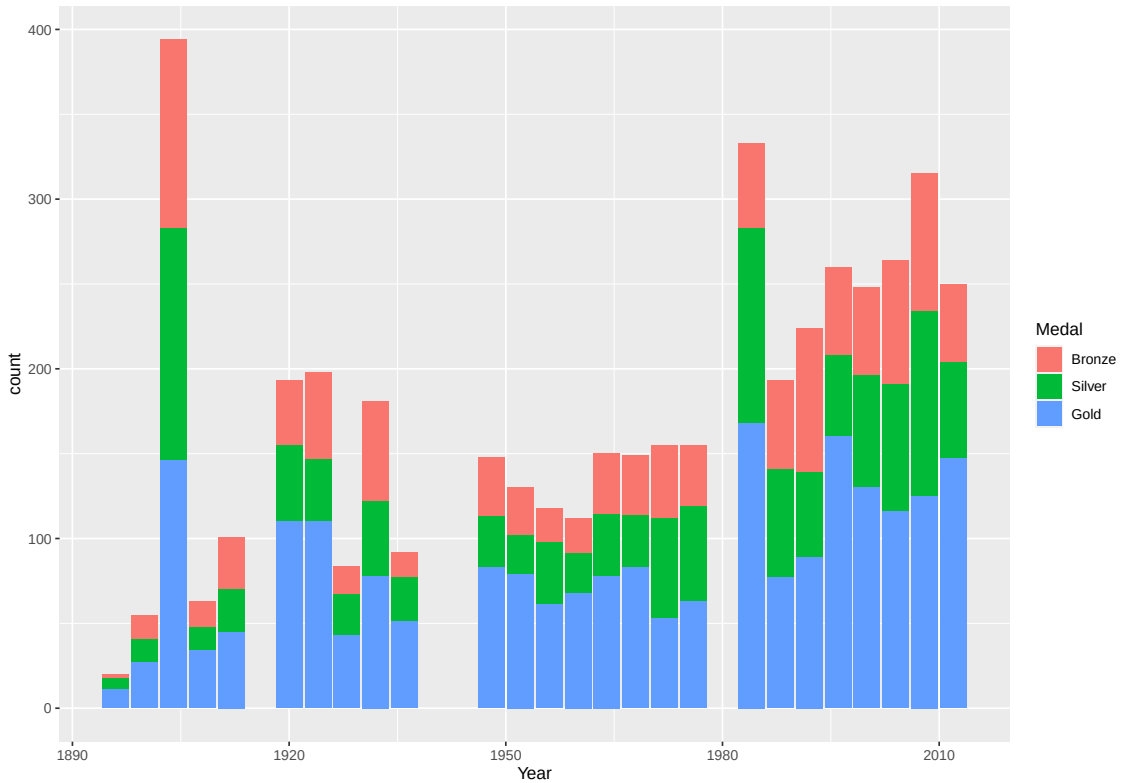
```
ggplot(usa_medals, aes(x=Medal, y=Percentage)) +
  geom_bar(stat="identity", fill=c('red','green','blue'))
```



Visualizing categorical data

What if we want to see how the distribution of medals for USA has changed over time? You need the following aesthetics: Year on x axis, fill by medal

```
summer_usa$Medal <- factor(summer_usa$Medal,
                           levels=c("Bronze", "Silver", "Gold"))
ggplot(summer_usa, aes(x=Year, fill=Medal))+geom_bar()
```



Visualizing categorical data

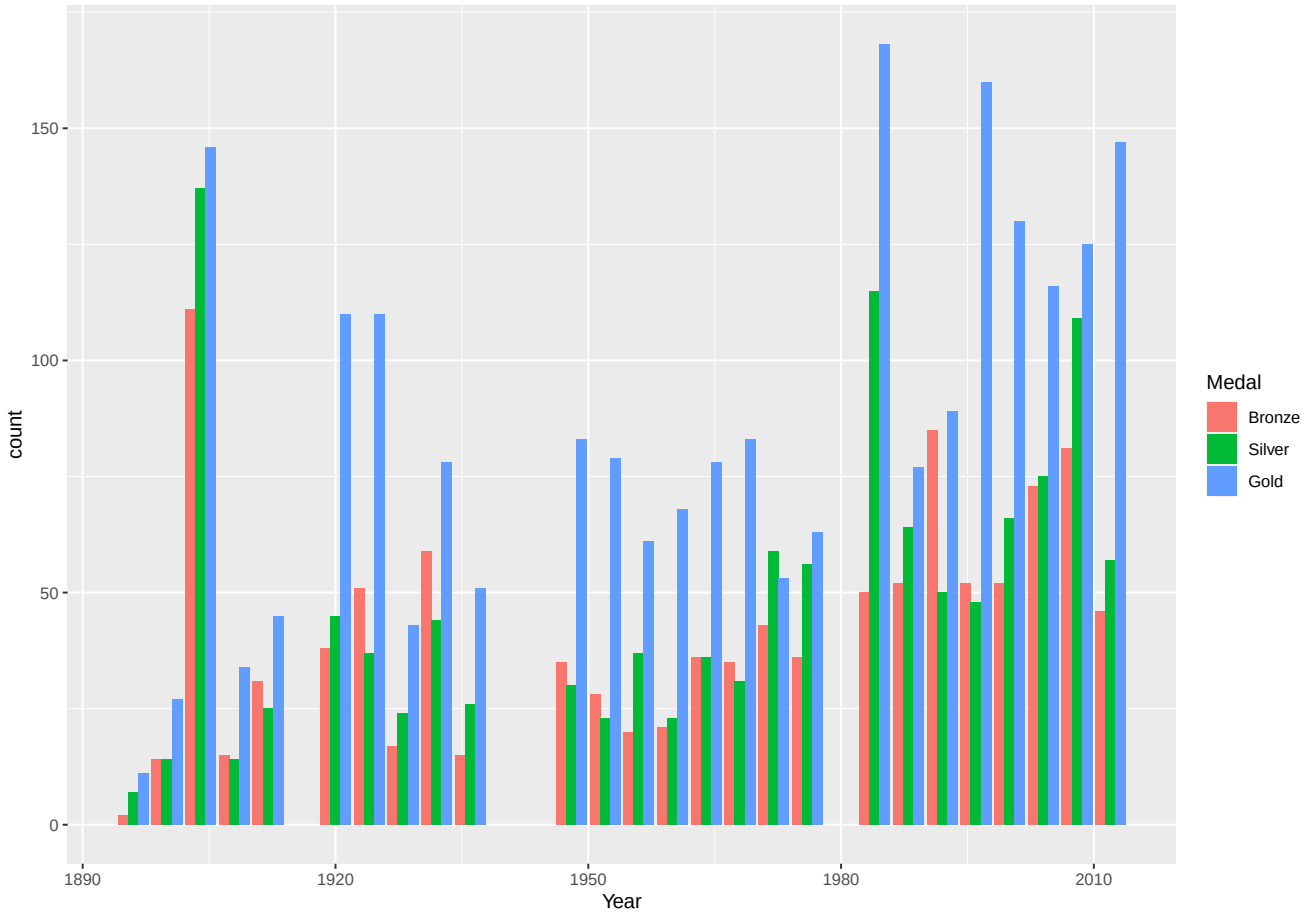
Change position fill to get 100% bar chart

```
ggplot(summer_usa, aes(x=Year, fill=Medal)) + geom_bar(position="fill")
```

Visualizing categorical data

Position dodge to get side-by-side bar chart

```
ggplot(summer_usa, aes(x=Year, fill=Medal)) + geom_bar(position="dodge")
```



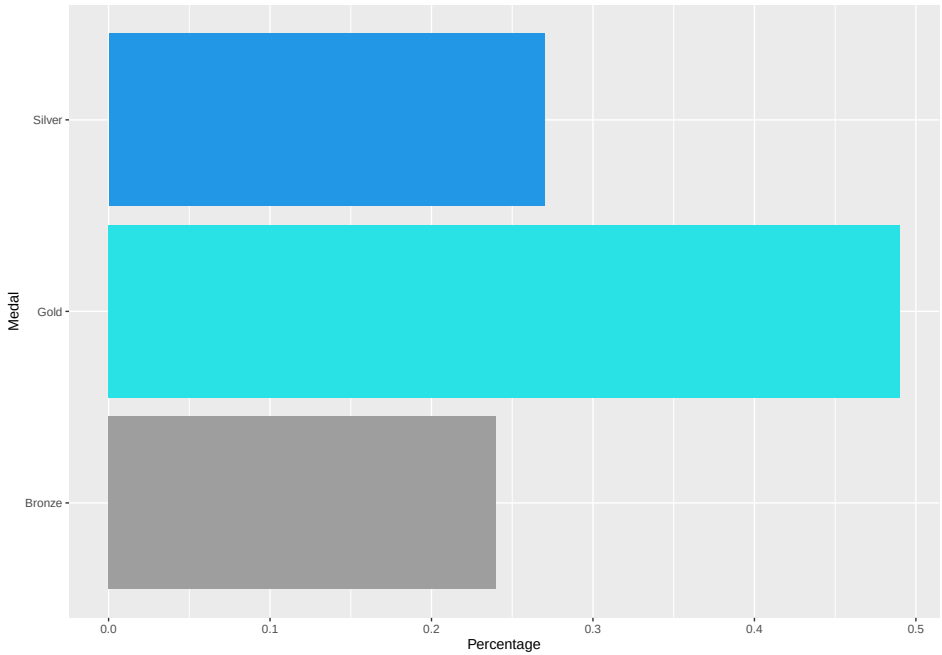
Visualizing categorical data – coordinate system

- Coordinate systems tie together the two position scales to produce a 2d location. Currently, ggplot2 comes with different coordinate systems.
- All these coordinate systems are two-dimensional.
- As with the other components in ggplot2, you generate the R name by joining coord_ and the name of the coordinate system.
- Most plots use the default Cartesian coordinate system, coord_cartesian(), where the 2d position of an element is given by the combination of the x and y positions.

Visualizing categorical data – coordinate system

- coord_flip will flip the x and y coordinates
- You can add it as a layer to a barplot created before to get a vertical bar plot with categories on y axis and values on x axis

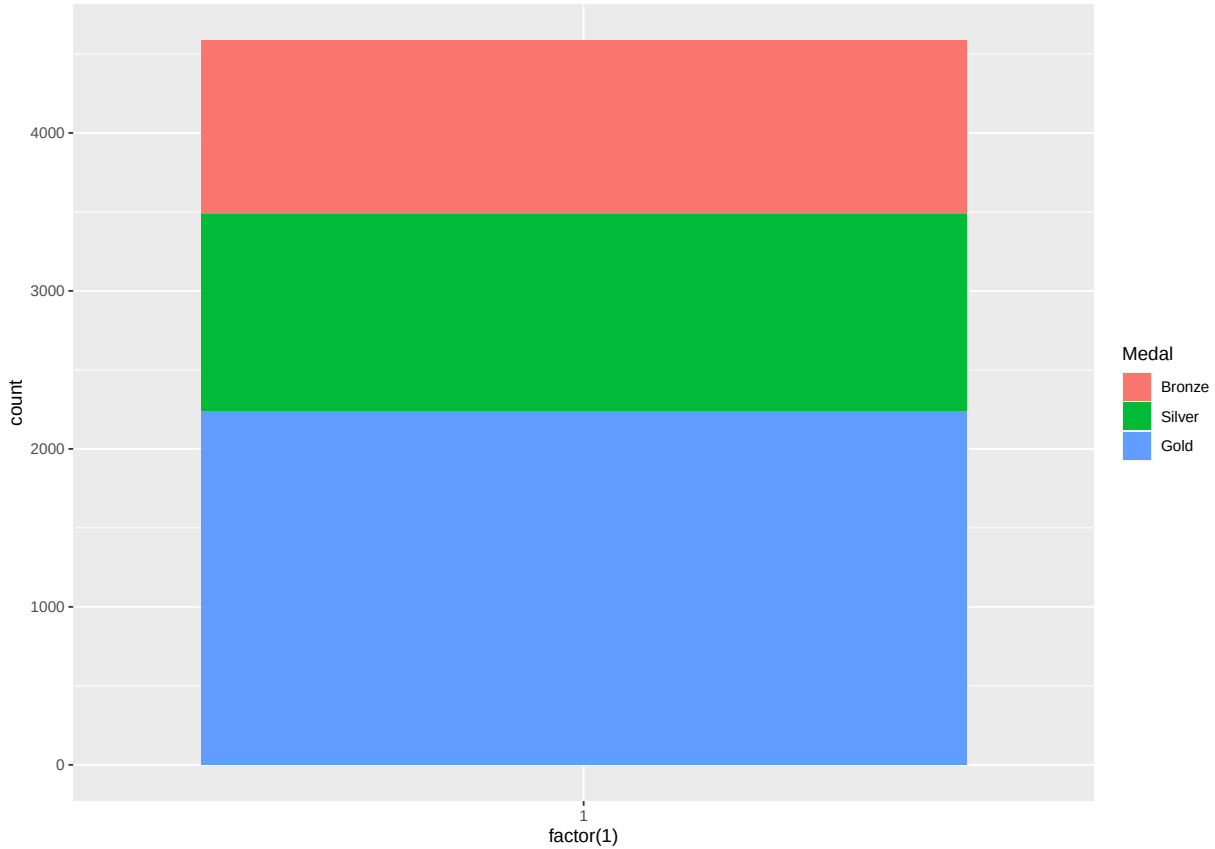
```
ggplot(usa_medals, aes(x=Medal, y=Percentage)) +
  geom_bar(stat="identity", fill=c(4,5,8)) +
  coord_flip()
```



Visualizing categorical data – coordinate system

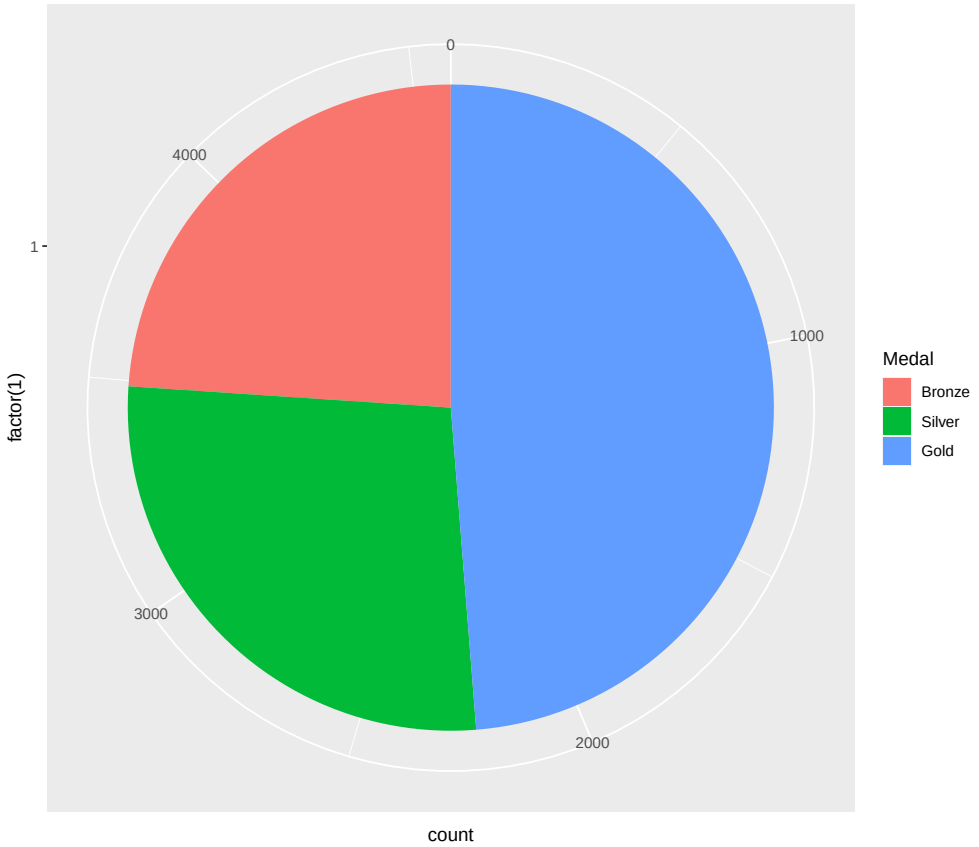
Stacked bar chart, you can put anything in factor() for x variable

```
ggplot(summer_usa, aes(x=factor(1), fill=Medal)) + geom_bar()
```



Visualizing categorical data – coordinate system

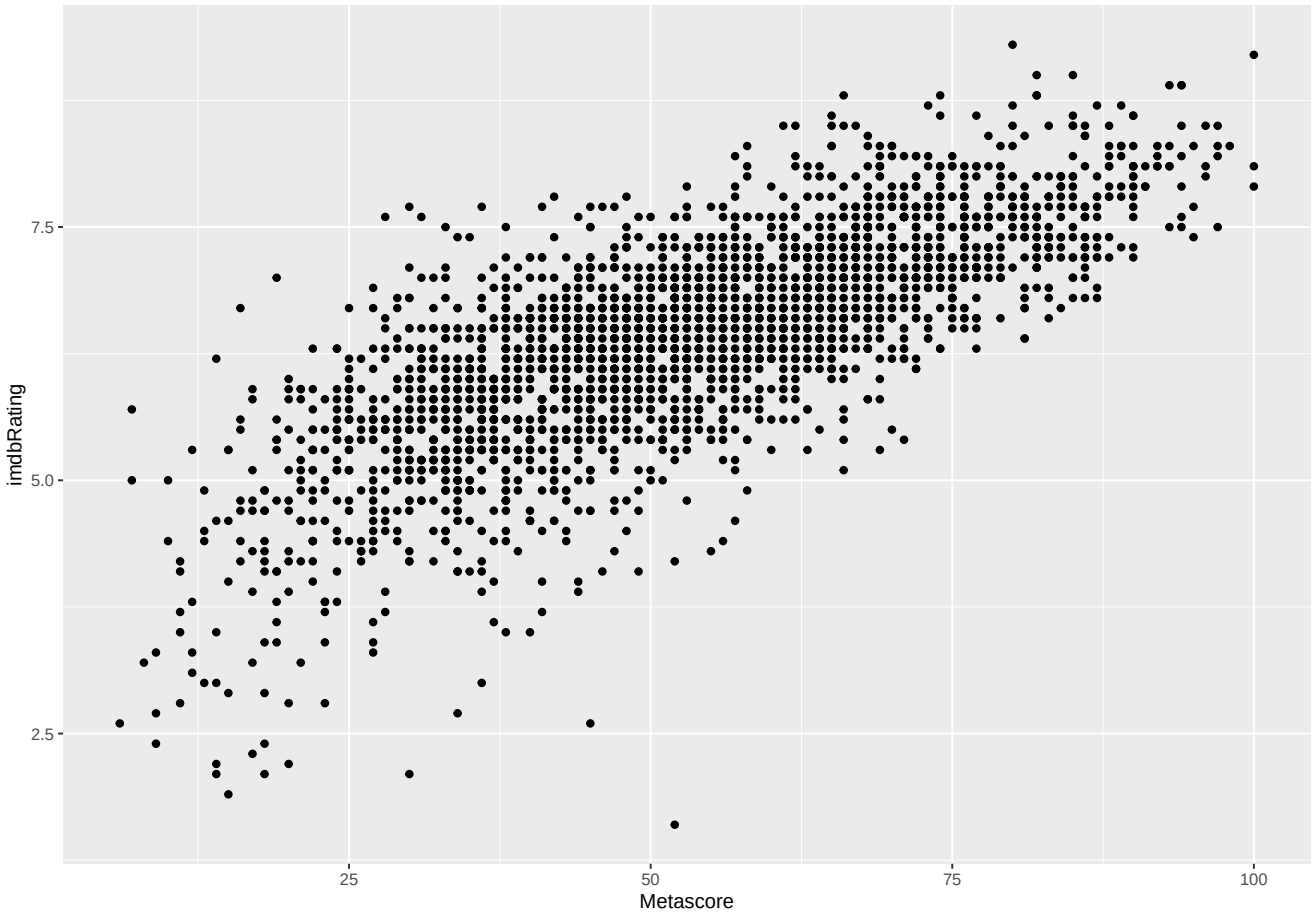
```
ggplot(summer_usa, aes(x=factor(1), fill=Medal))+geom_bar()+
  coord_polar(theta="y",start=0)
```



Using coordinates on continuous data

Metascore vs imdbRating

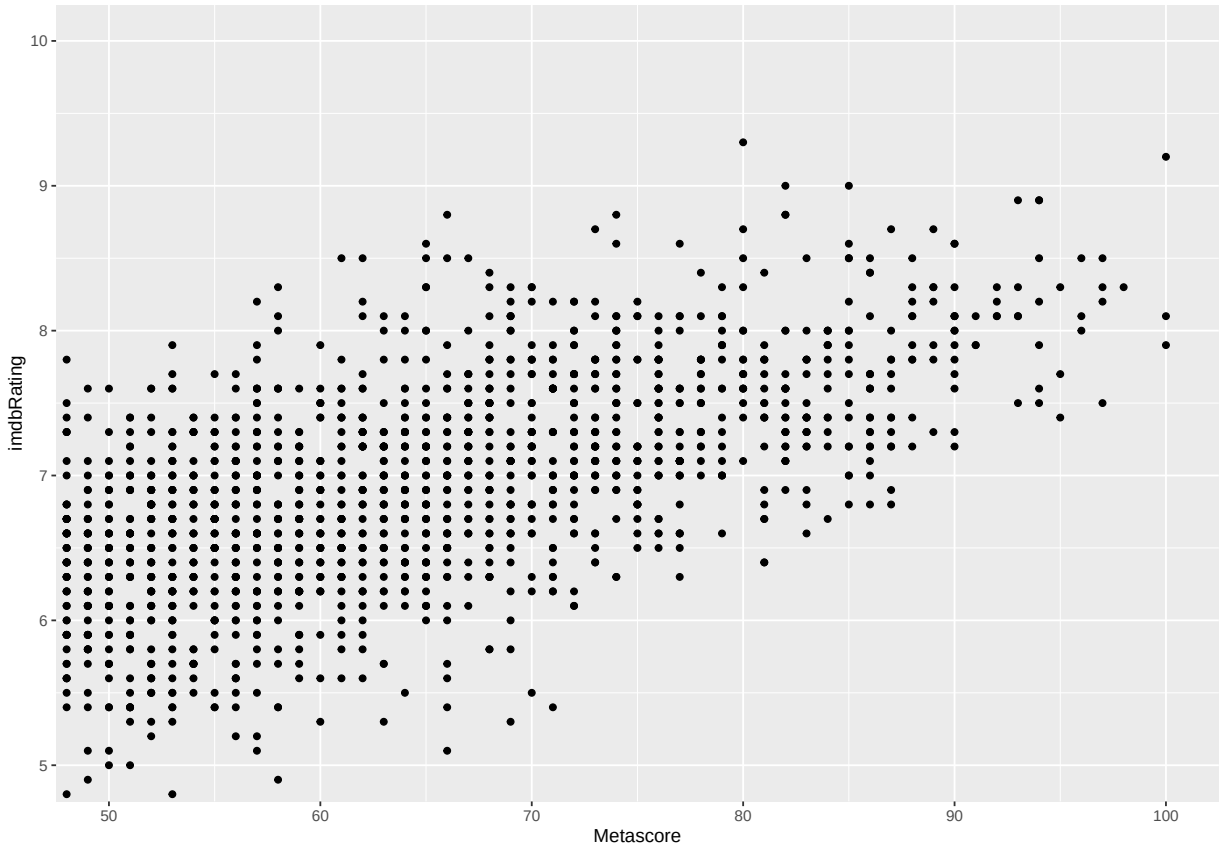
```
ggplot(movies, aes(x=Metascore, y=imdbRating))+geom_point()
```



Using coordinates on continuous data

We want to limit x and y axes, thus zoom in the graph

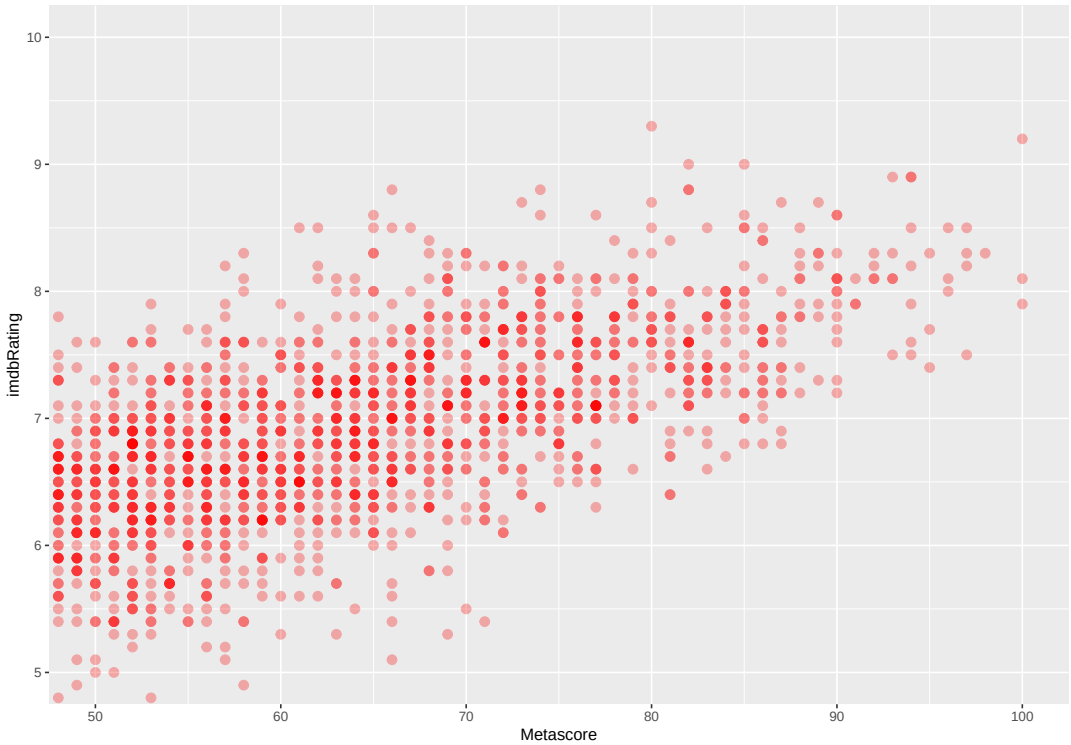
```
ggplot(movies, aes(x=Metascore, y=imdbRating))+geom_point()+
  coord_cartesian(ylim=c(5,10), xlim=c(50,100))
```



Using coordinates on continuous data

- If the sample size is big, then the points are overlapped which distorts the real picture.
- Use alpha for point transparency

```
ggplot(movies, aes(x=Metascore, y=imdbRating))+
  geom_point(alpha=0.3, col="red", size=2.7)+
  coord_cartesian(ylim=c(5,10), xlim=c(50,100))
```

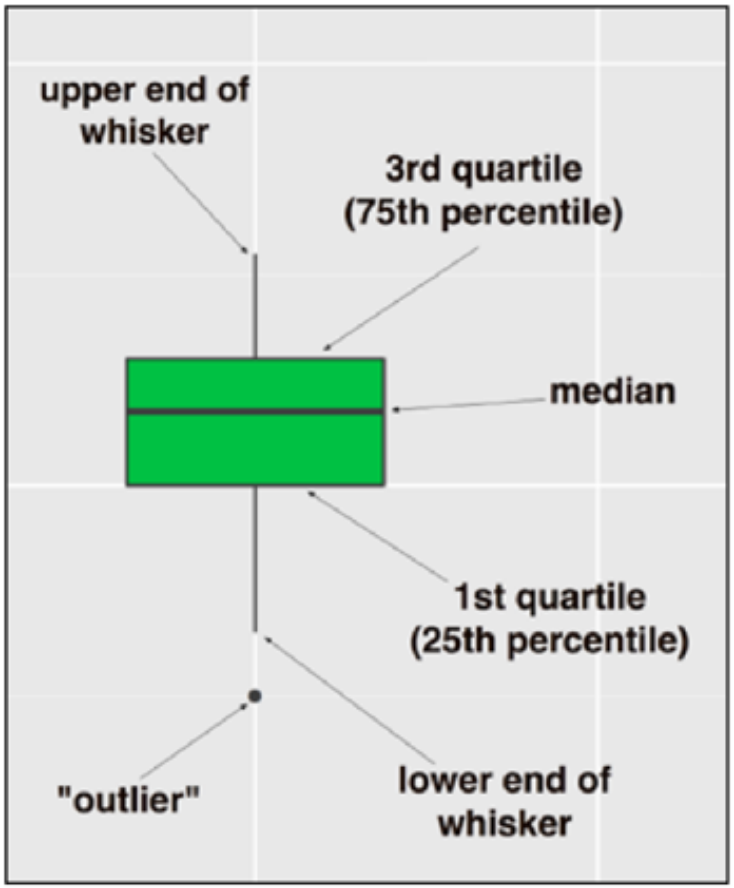


Section 5

Plotting categorical and continuous data together

Plotting categorical and continuous data together

- The length of the whisker = 1.5*IQR (Inter Quartile range)
- IQR = 3rd Quartile-1st Quartile

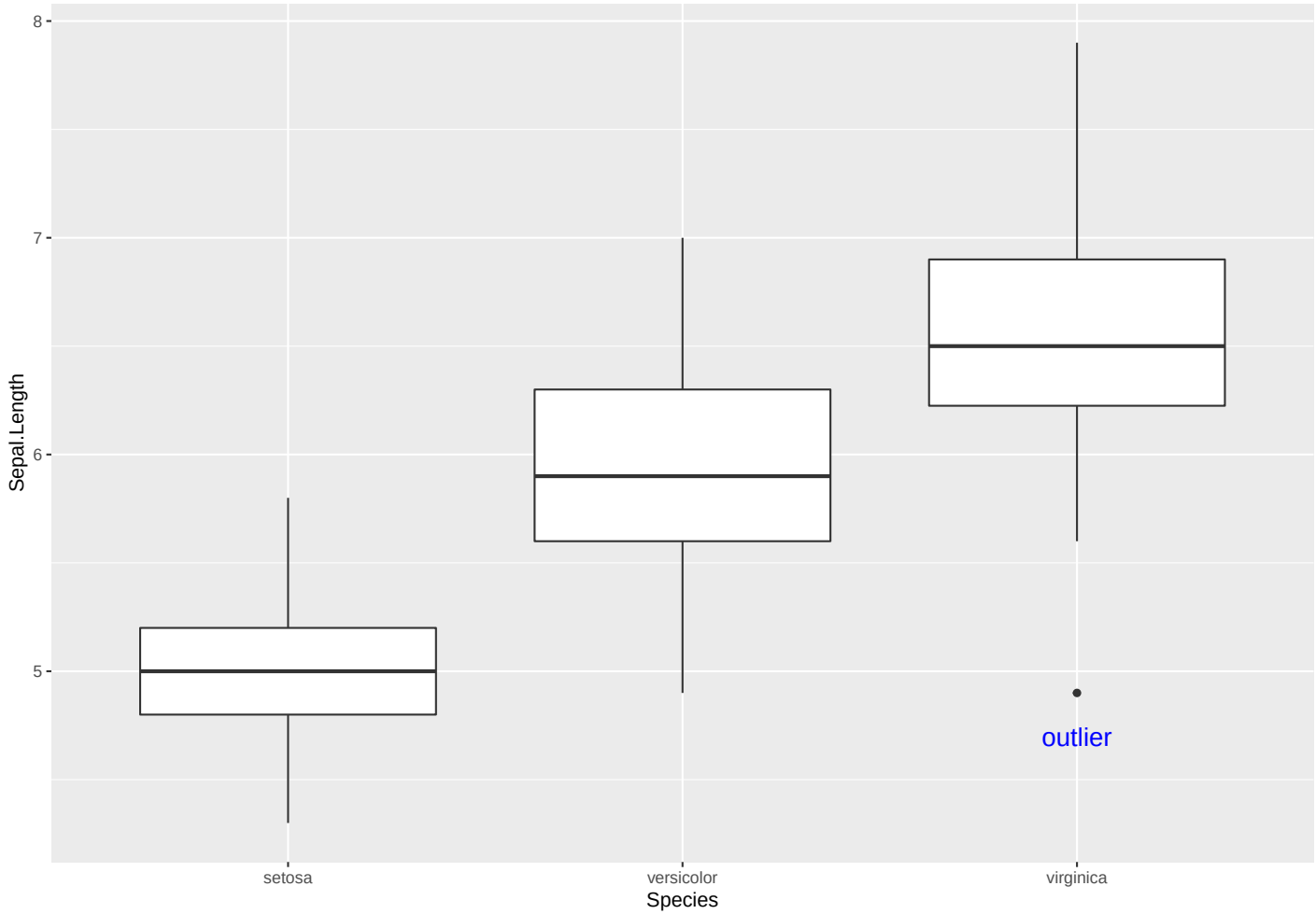


Plotting categorical and continuous data together

Symbol	Names	Definition
Q_1	first quartile lower quartile 25th percentile	splits off the lowest 25% of data from the highest 75%
Q_2	second quartile median 50th percentile	cuts data set in half
Q_3	third quartile upper quartile 75th percentile	splits off the highest 25% of data from the lowest 75%

Plotting categorical and continuous data together

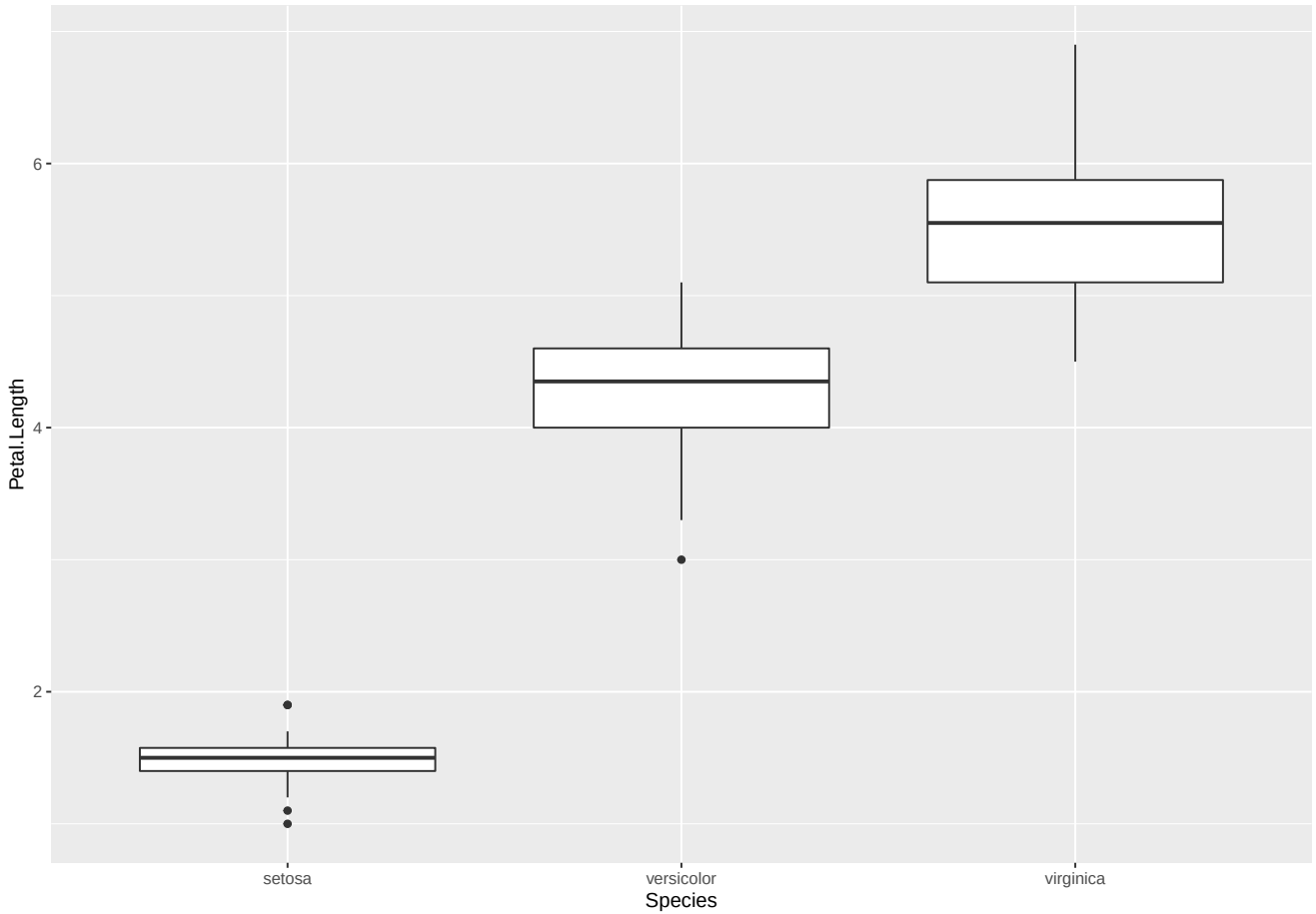
```
ggplot(iris, aes(x=Species, y=Sepal.Length))+geom_boxplot()
```



Plotting categorical and continuous data together

Boxplot for Petal.Length

```
ggplot(iris, aes(x=Species, y=Petal.Length))+geom_boxplot()
```



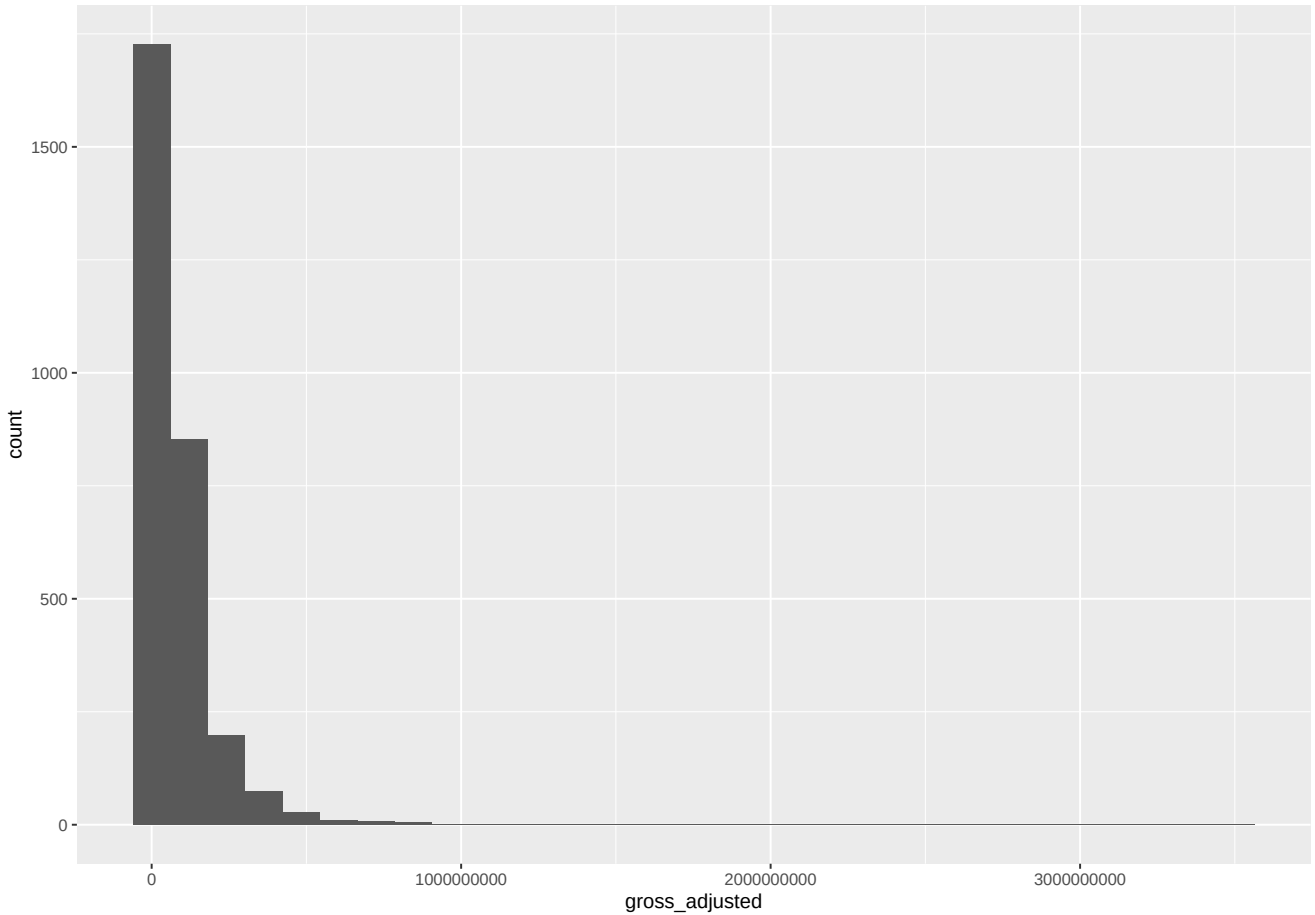
Plotting categorical and continuous data together

- Boxplot helps to understand if the distributions by categories are different from each other.
- Also it helps to understand the shape of the distribution: whether it is symmetric or skewed.

Plotting categorical and continuous data together

Highly skewed distribution

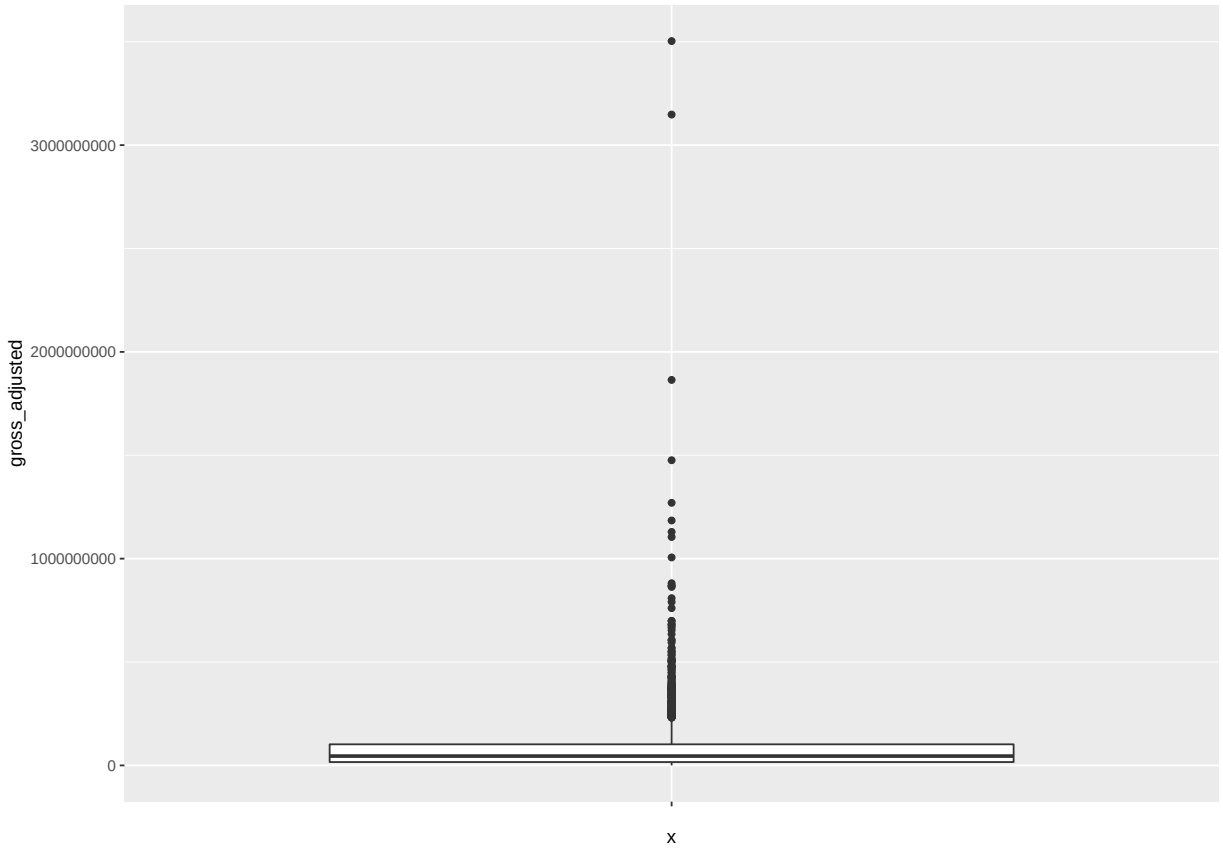
```
ggplot(movies, aes(x=gross_adjusted))+geom_histogram()
```



Plotting categorical and continuous data together

- A few outliers, few movies made a huge box office.
- Note: for x aesthetics we just use an empty string, as the boxplot is created for the whole column.

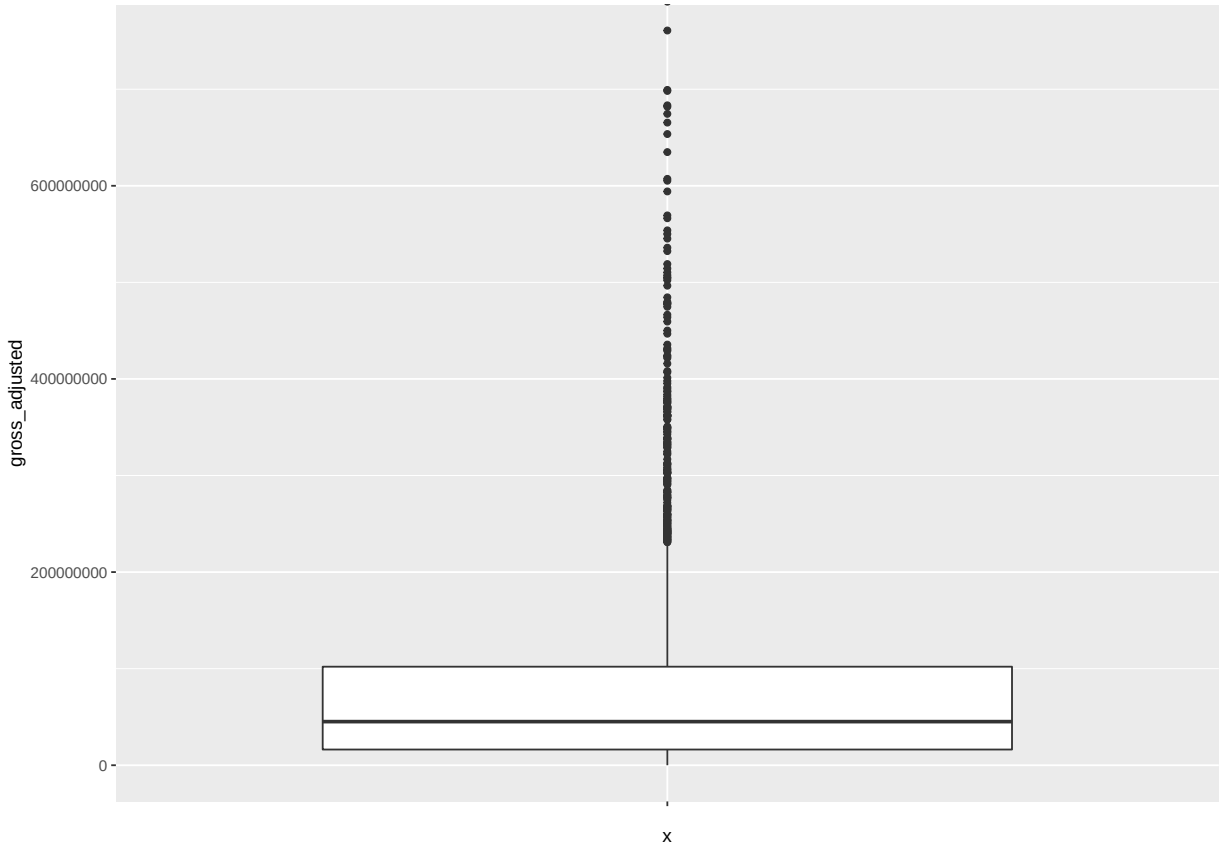
```
ggplot(movies, aes(y=gross_adjusted, x=""))+geom_boxplot()
```



Plotting categorical and continuous data together

Zoom in: with symmetric distribution the median would be in the middle

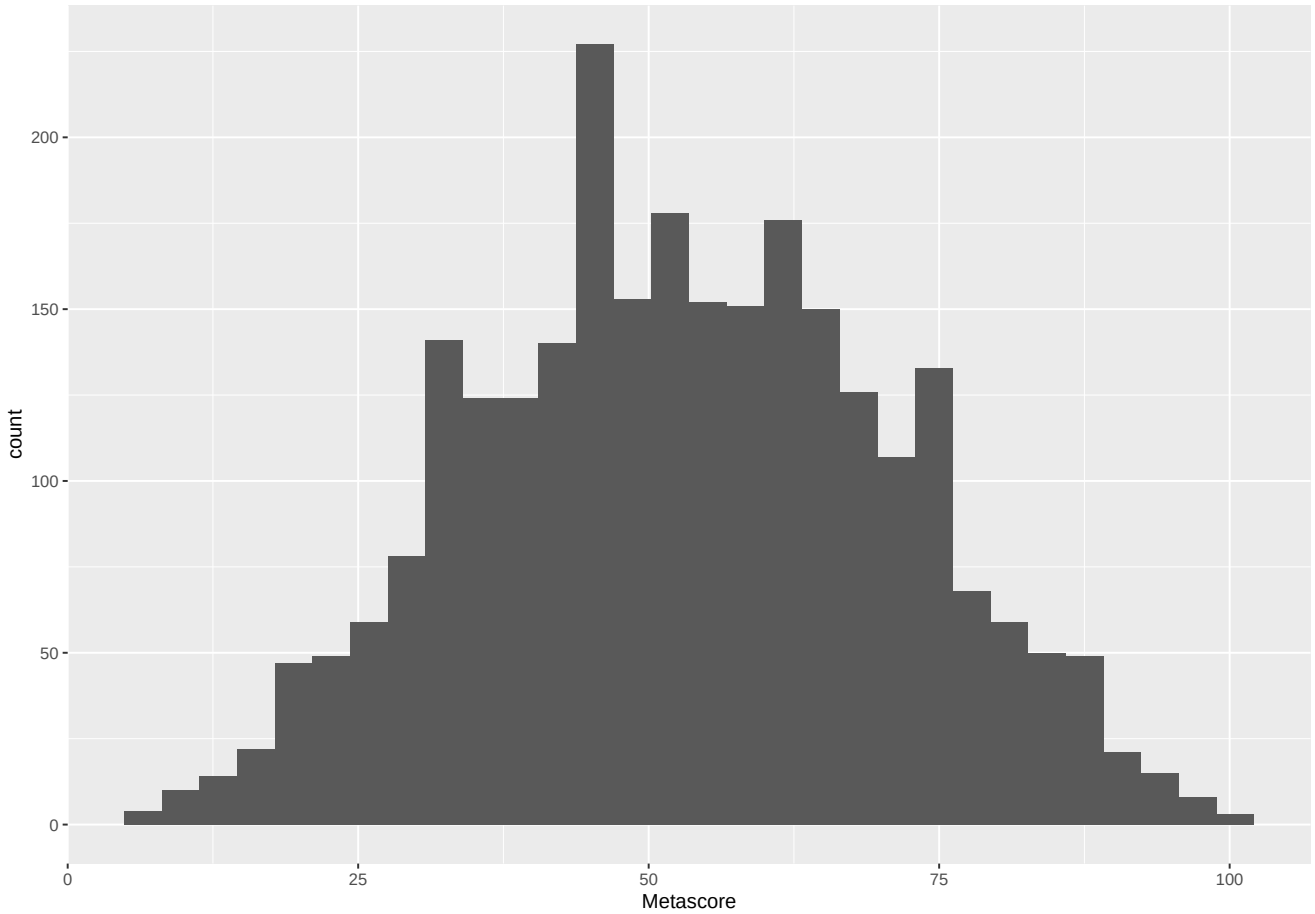
```
ggplot(movies, aes(y=gross_adjusted, x="")) + geom_boxplot()+
  coord_cartesian(ylim=c(0,750000000))
```



Plotting categorical and continuous data together

Look at the histogram of Metascore, almost symmetric

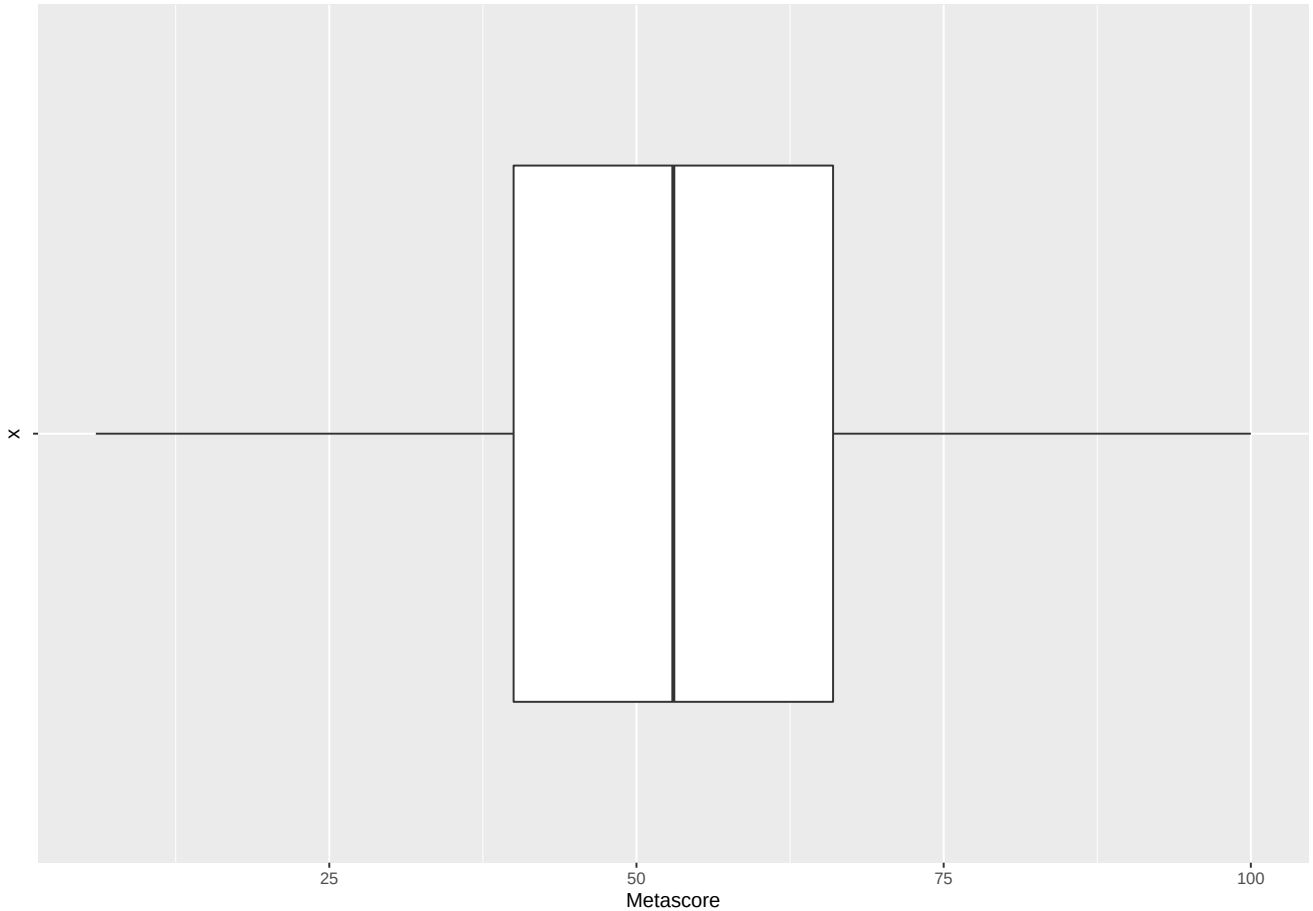
```
ggplot(movies, aes(x=Metascore)) + geom_histogram()
```



Plotting categorical and continuous data together

The boxplot with coordinates flipped

```
ggplot(movies, aes(y=Metascore, x="")) + geom_boxplot() + coord_flip()
```



Section 6

Importance of visualization

Beer-goggle effect example

- An anthropologist was interested in the effects of alcohol on mate selection at nightclubs. Her rationale was that after alcohol had been consumed, subjective perceptions of physical attractiveness would become more inaccurate. She was also interested if this effect is different for males and females. She made photos of the people the participants were chatting with, then got a pool of independent judges to assess the attractiveness of the person in each photograph.
- **Factor A**: Alcohol, with 3 treatment levels {None, 2 pints, 4 pints}
- **Factor B**: Gender, with two levels {male, female}
- **Dependent Variable**: Measure of attractiveness of the person [0,100]: The rating is given by independent "jury"

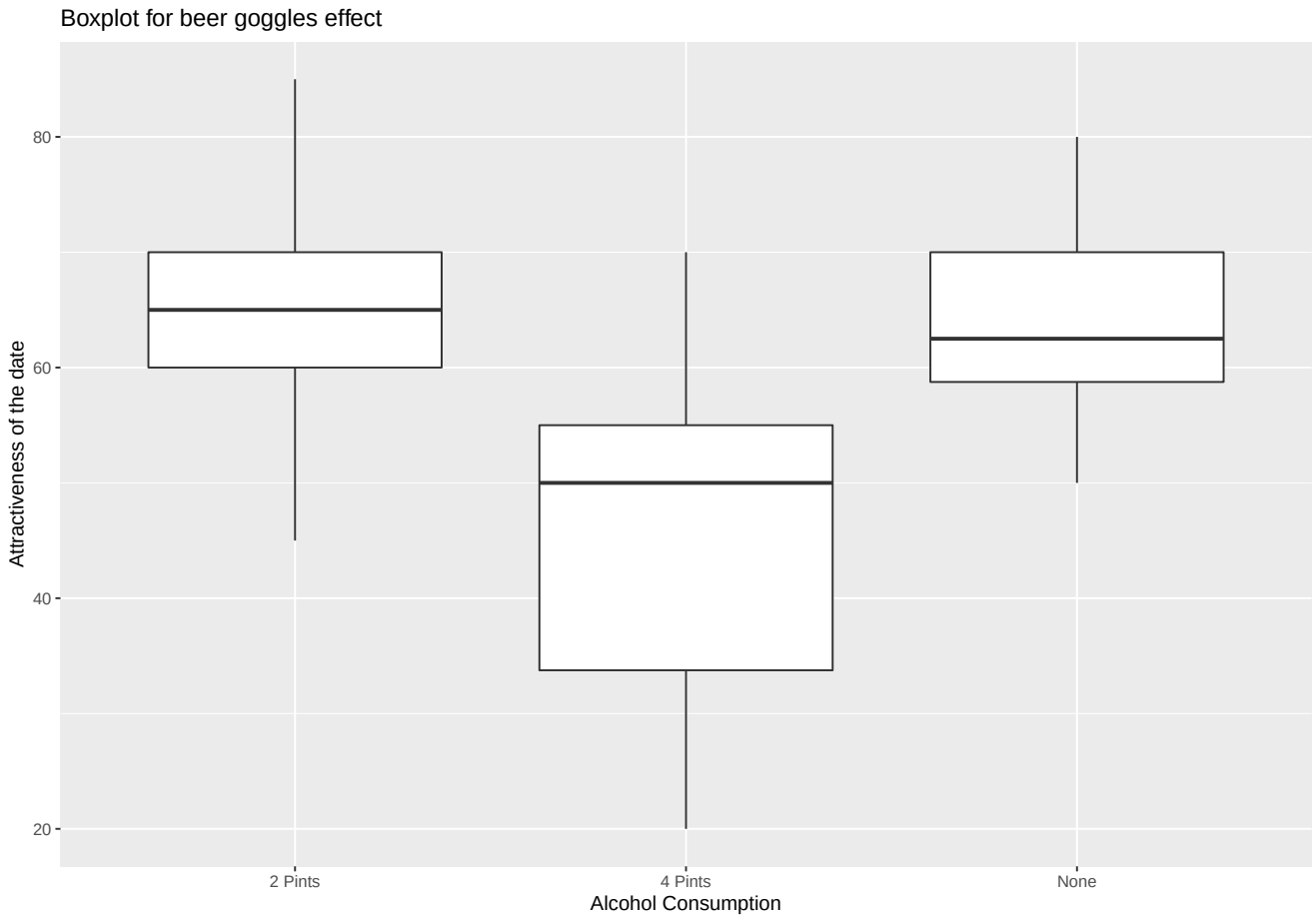
Beer-goggle effect example

```
goggles<-read.csv('goggles.csv')
summary(goggles)
```

```
##     gender             alcohol          attractiveness
##  Length:48          Length:48          Min.   :20.00
##  Class :character   Class :character   1st Qu.:53.75
##  Mode  :character   Mode  :character   Median :60.00
##                                        Mean   :58.33
##                                        3rd Qu.:66.25
##                                        Max.   :85.00
```

Beer-goggle effect example

```
ggplot(data=goggles, aes(x = alcohol, y = attractiveness))+ geom_boxplot()+
  labs(x = 'Alcohol Consumption', y = 'Attractiveness of the date',
       title = 'Boxplot for beer goggles effect')
```



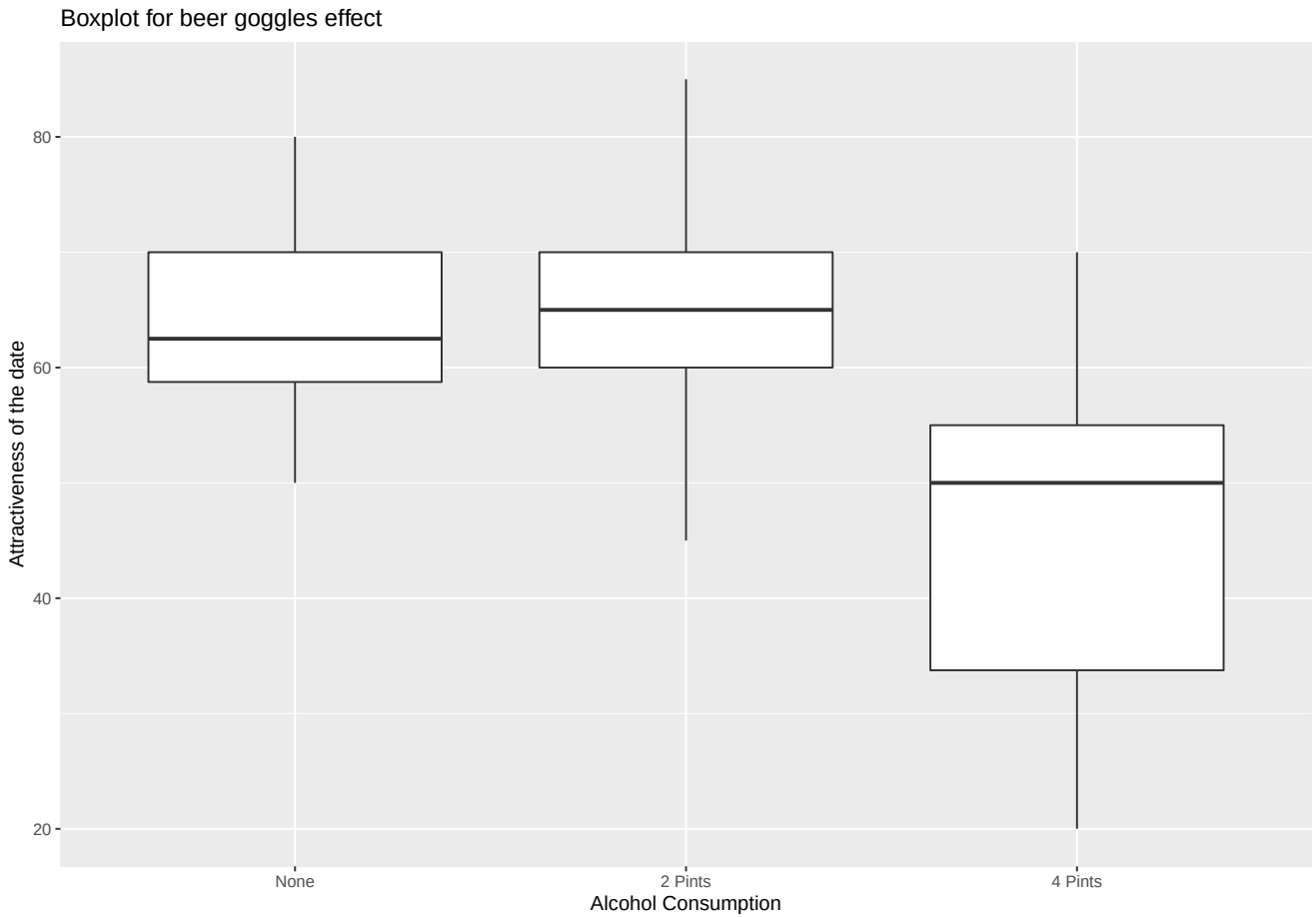
Beer-goggle effect example

Relevel the factor variable, with a natural order

```
goggles$alcohol<-factor(goggles$alcohol,
                        levels = c("None", "2 Pints", "4 Pints"))
```

Beer-goggle effect example

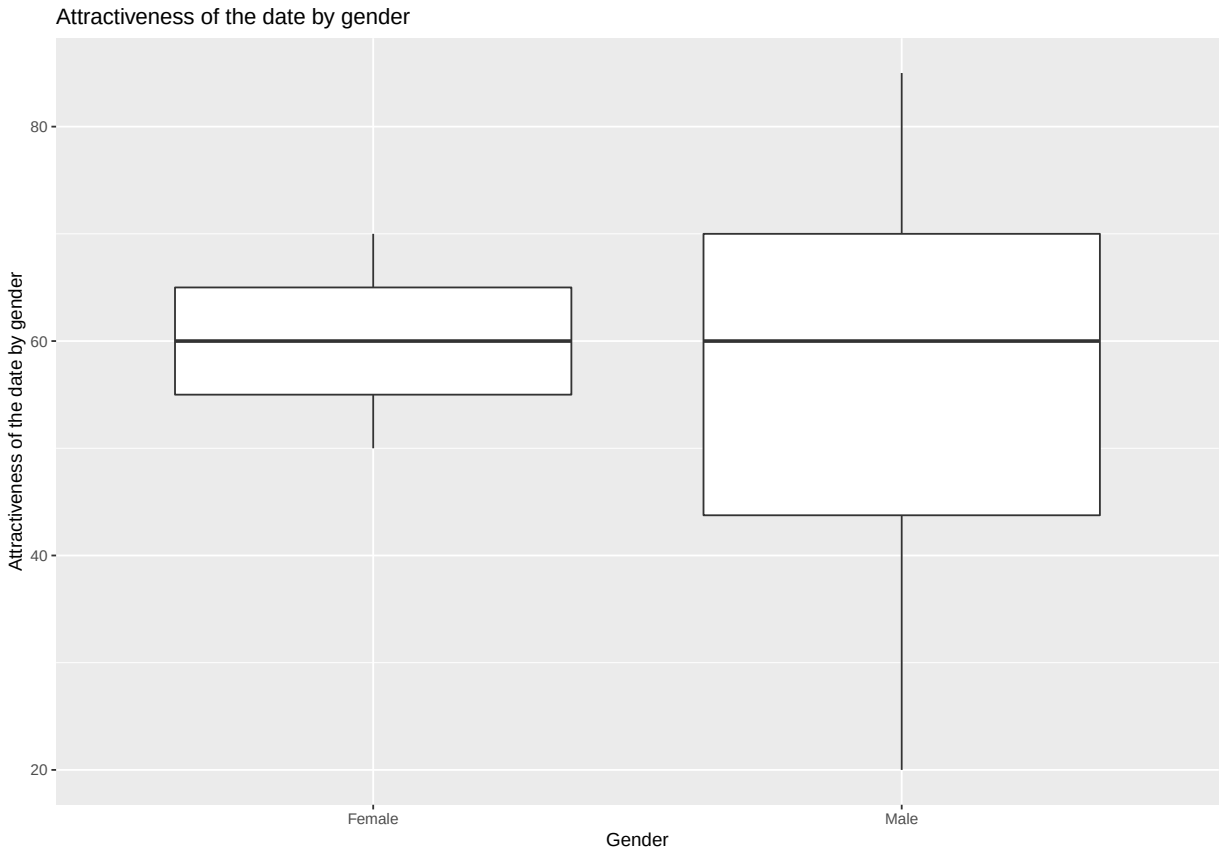
```
ggplot(data=goggles, aes(x = alcohol, y = attractiveness))+geom_boxplot()+
  labs(x = 'Alcohol Consumption', y = 'Attractiveness of the date',
       title = 'Boxplot for beer goggles effect')
```



Beer-goggle effect example

Boxplot for gender, what is your conclusion?

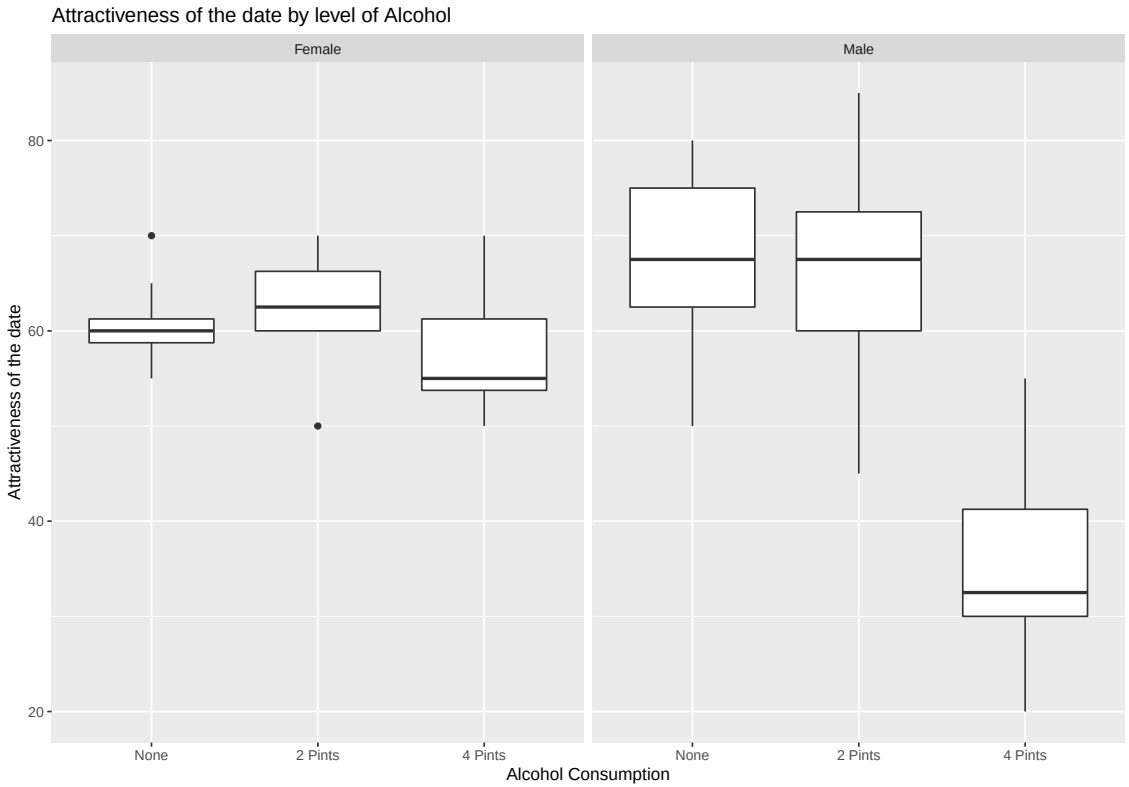
```
ggplot(data=goggles, aes(x=gender, y=attractiveness))+geom_boxplot()+
 labs(x = 'Gender', y = 'Attractiveness of the date by gender',
      title = 'Attractiveness of the date by gender')
```



Beer-goggle effect example

Boxplot for each gender separately, what is your conclusion?

```
ggplot(data=goggles, aes(x=alcohol, y=attractiveness))+geom_boxplot()+
  facet_grid(.~gender)+ylab('Attractiveness of the date') +
  labs(x = 'Alcohol Consumption',
       title = 'Attractiveness of the date by level of Alcohol')
```



University of California Berkley - Scandal!

In 1973, the University of California-Berkeley was sued for sex discrimination. The numbers looked pretty incriminating: the graduate schools had just accepted 44% of male applicants but only 35% of female applicants.

UCBAdmissions example

A collection of contingency tables for 6 biggest departments at UCB

```
data("UCBAdmissions")
UCBAdmissions
```

```
## , , Dept = A
##
##           Gender
## Admit      Male Female
##   Admitted  512     89
##   Rejected  313     19
##
## , , Dept = B
##
##           Gender
## Admit      Male Female
##   Admitted  353     17
##   Rejected  207      8
##
## , , Dept = C
##
##           Gender
## Admit      Male Female
##   Admitted  120    202
##   Rejected  205    391
##
## , , Dept = D
##
##           Gender
## Admit      Male Female
##   Admitted  138    131
##   Rejected  279    244
##
## , , Dept = E
##
```

UCBAdmissions example

Convert it to a dataframe

```
ucba <- as.data.frame(UCBAdmissions)
head(ucba)
```

```
##      Admit Gender Dept Freq
## 1 Admitted   Male    A  512
## 2 Rejected   Male    A  313
## 3 Admitted Female    A   89
## 4 Rejected Female    A   19
## 5 Admitted   Male    B  353
## 6 Rejected   Male    B  207
```

UCBAdmissions example

Contingency table

Fewer women are accepted compared to men, however less women applied as well

```
(cross<-xtabs(Freq~Gender + Admit, data = ucba))
```

```
##         Admit
## Gender   Admitted Rejected
##   Male       1198     1493
##   Female      557     1278
```

Calculate row percentages with

```
prop.table(cross, 1)
```

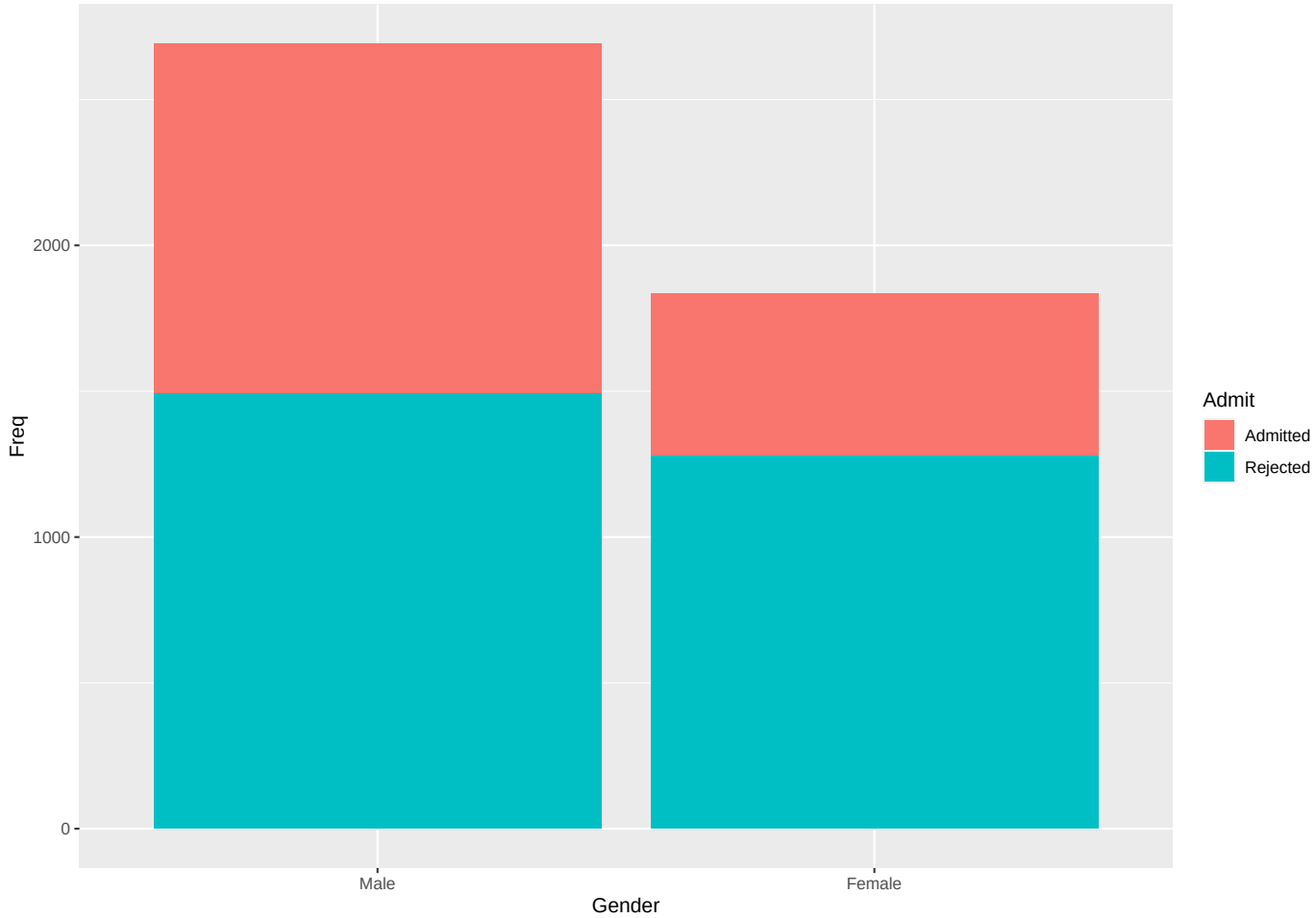
```
##         Admit
## Gender    Admitted  Rejected
##   Male   0.4451877 0.5548123
##   Female 0.3035422 0.6964578
```

For the columns' percentages use 2 (second dimension) instead of 1

UCBAdmissions example

Use stat="identity" to do data transformation of $x = f(x)$

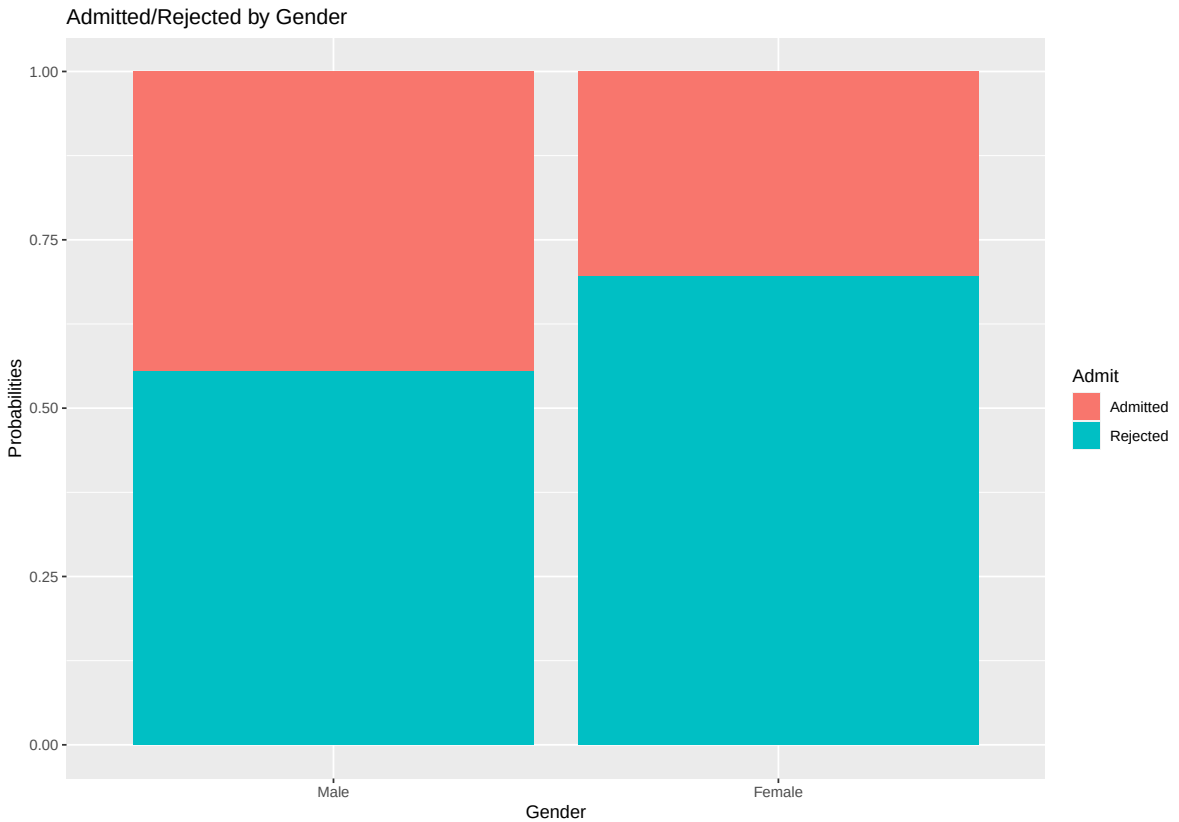
```
ggplot(ucba, aes(x = Gender, y = Freq, fill = Admit)) +
  geom_bar(stat = "identity")
```



UCBAdmissions example

Add position = "fill" to get the bars aligned with x axis showing relative rather than absolute frequencies.

```
ggplot(ucba, aes(x = Gender, y = Freq, fill = Admit)) +
  geom_bar(stat = "identity", position = "fill") +
  labs(y = "Probabilities", title = "Admitted/Rejected by Gender")
```



UCBAdmissions example

Now look at the department level, Department A

```
cross2 <- xtabs(Freq~Gender + Admit, data = ucba[ucba$Dept=="A",])
prop.table(cross2,1)
```

```
##         Admit
## Gender    Admitted  Rejected
##   Male    0.6206061 0.3793939
##   Female 0.8240741 0.1759259
```

Department B

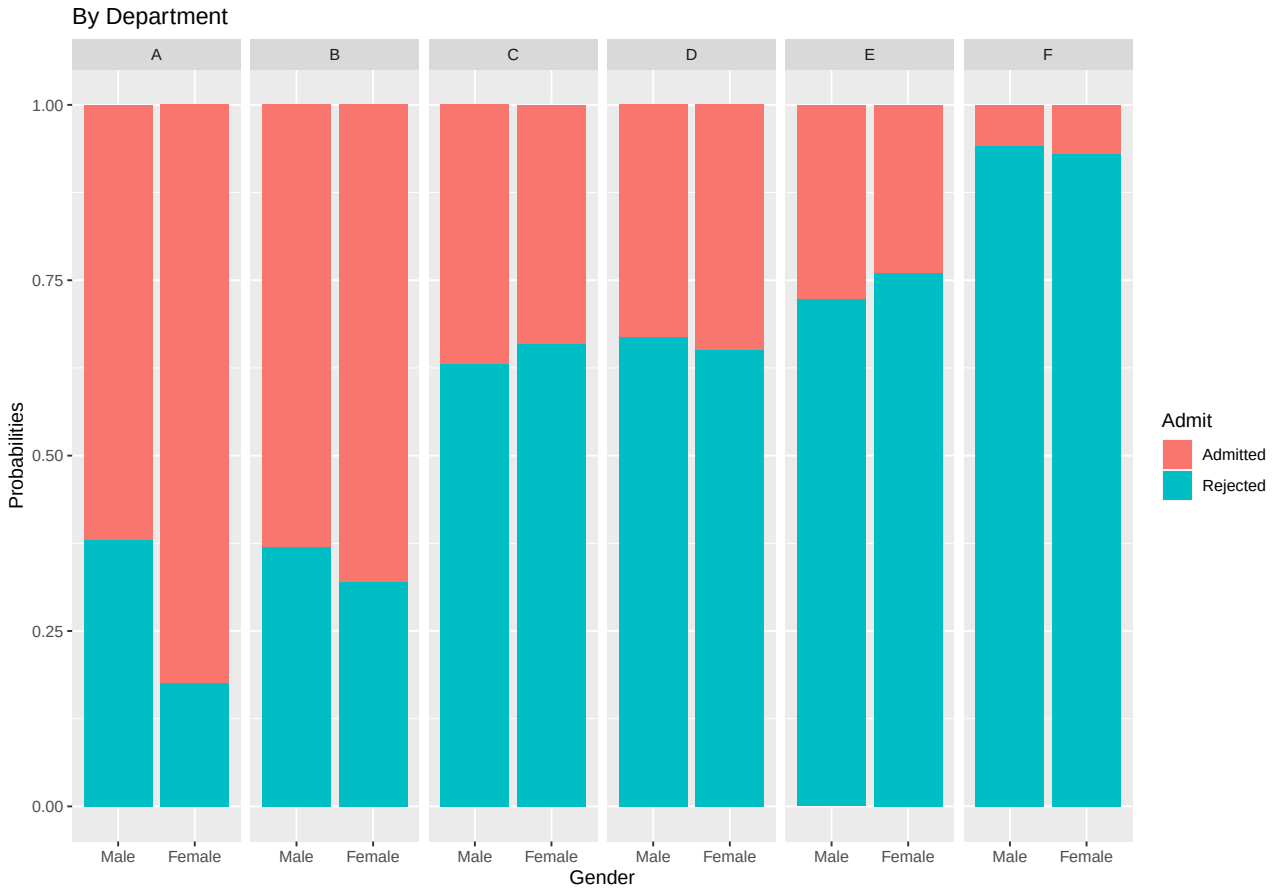
```
cross2 <- xtabs(Freq~Gender + Admit, data = ucba[ucba$Dept=="B",])
prop.table(cross2,1)
```

```
##         Admit
## Gender    Admitted  Rejected
##   Male    0.6303571 0.3696429
##   Female 0.6800000 0.3200000
```

UCBAdmissions example

use facet_grid() to look at the percentages on department level

```
ggplot(ucba, aes(x = Gender, y = Freq, fill = Admit)) +
  geom_bar(stat = "identity", position = "fill") +
  facet_grid(~Dept) + labs(y = "Probabilities",title = "By Department")
```



Anscombe's quartet

Anscombe's quartet

I		II		III		IV	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Anscombe, Francis J. (1973) Graphs in statistical analysis. *American Statistician*, **27**, 17–21.

Anscombe's quartet

```
data('anscombe')
anscombe
```

```
##    x1 x2 x3 x4    y1   y2    y3    y4
## 1  10 10 10  8  8.04 9.14  7.46  6.58
## 2   8  8  8  8  6.95 8.14  6.77  5.76
## 3  13 13 13  8  7.58 8.74 12.74  7.71
## 4   9  9  9  8  8.81 8.77  7.11  8.84
## 5  11 11 11  8  8.33 9.26  7.81  8.47
## 6  14 14 14  8  9.96 8.10  8.84  7.04
## 7   6  6  6  8  7.24 6.13  6.08  5.25
## 8   4  4  4 19  4.26 3.10  5.39 12.50
## 9  12 12 12  8 10.84 9.13  8.15  5.56
## 10  7  7  7  8  4.82 7.26  6.42  7.91
## 11  5  5  5  8  5.68 4.74  5.73  6.89
```

Anscombe's quartet

Mean

```
apply(anscombe,2, FUN = 'mean')
```

```
##       x1       x2       x3       x4       y1       y2       y3       y4
## 9.000000 9.000000 9.000000 9.000000 7.500909 7.500909 7.500000 7.500909
```

Standard deviations

```
apply(anscombe,2, FUN = 'sd')
```

```
##       x1       x2       x3       x4       y1       y2       y3       y4
## 3.316625 3.316625 3.316625 3.316625 2.031568 2.031657 2.030424 2.030579
```

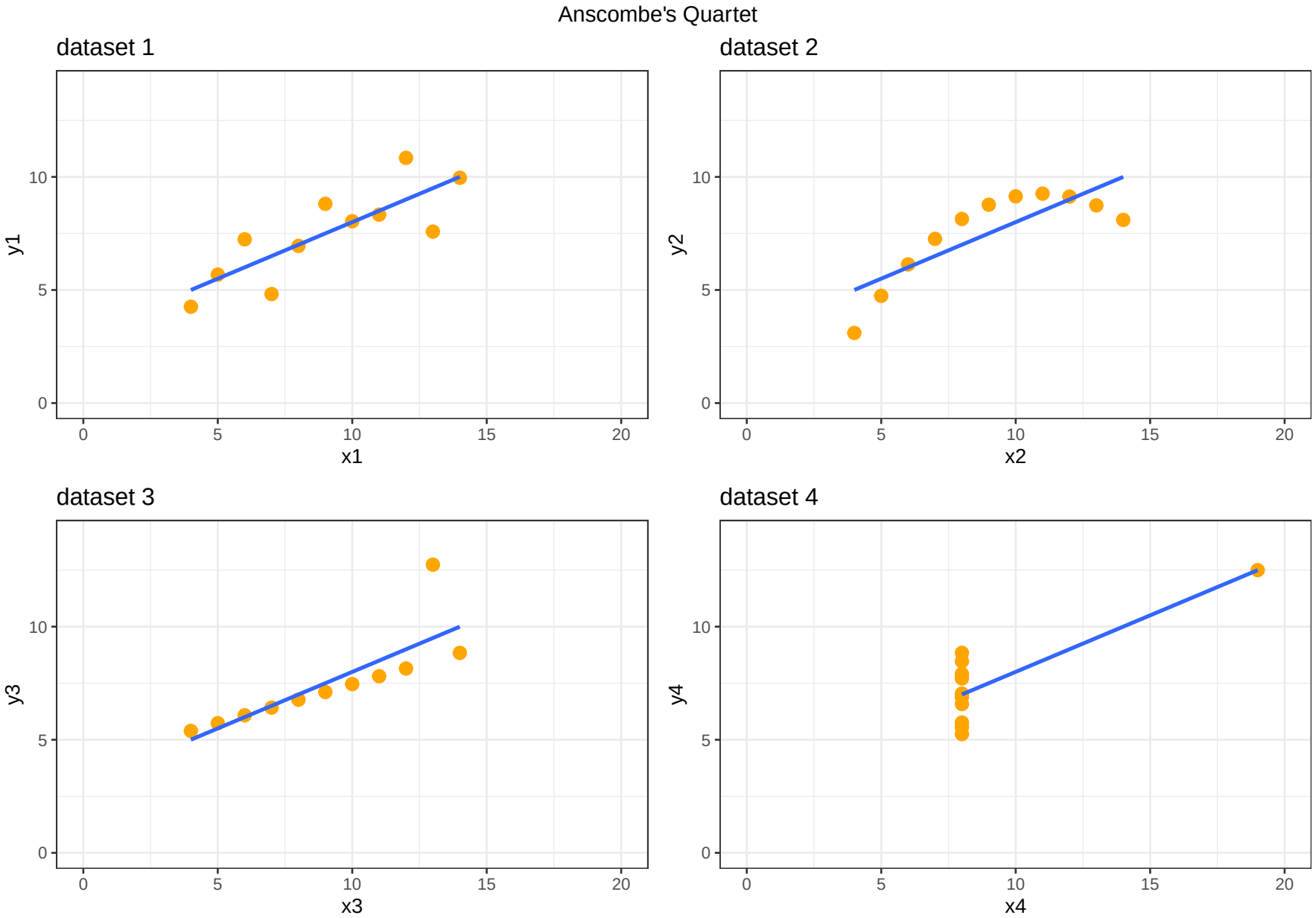
Anscombe's quartet

Correlations

```
cor(anscombe)
```

```
##            x1         x2         x3         x4         y1         y2         y3
## x1  1.0000000  1.0000000  1.0000000 -0.5000000  0.8164205  0.8162365  0.8162867
## x2  1.0000000  1.0000000  1.0000000 -0.5000000  0.8164205  0.8162365  0.8162867
## x3  1.0000000  1.0000000  1.0000000 -0.5000000  0.8164205  0.8162365  0.8162867
## x4 -0.5000000 -0.5000000 -0.5000000  1.0000000 -0.5290927 -0.7184365 -0.3446610
## y1  0.8164205  0.8164205  0.8164205 -0.5290927  1.0000000  0.7500054  0.4687167
## y2  0.8162365  0.8162365  0.8162365 -0.7184365  0.7500054  1.0000000  0.5879193
## y3  0.8162867  0.8162867  0.8162867 -0.3446610  0.4687167  0.5879193  1.0000000
## y4 -0.3140467 -0.3140467 -0.3140467  0.8165214 -0.4891162 -0.4780949 -0.1554718
##            y4
## x1 -0.3140467
## x2 -0.3140467
## x3 -0.3140467
## x4  0.8165214
## y1 -0.4891162
## y2 -0.4780949
## y3 -0.1554718
## y4  1.0000000
```

Anscombe's quartet



Section 7

Data Transformation: basic R

Apply family functions

- The apply family consists of vectorized functions which minimize your need to explicitly create loops.
- These functions will apply a specified function to a data object and their primary difference is in the object class in which the function is applied to (list vs. matrix, etc.) and the object class that will be returned from the function.
  - **apply()** for matrices and data frames
  - **lapply()** for lists...output as list
  - **sapply()** for lists...output simplified
  - **tapply()** for vectors

Apply family functions

Recall the loop we have created to calculate the mean for every column in the data frame mtcars

```
x <- c()
for (i in 1:ncol(mtcars)){
  x1 <- mean(mtcars[,i])
  x <- c(x,x1)
}
x
```

```
##  [1]  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
##  [7]  17.848750   0.437500   0.406250   3.687500   2.812500         NA
```

Apply family functions

The same result can be obtained by function apply()

apply(X, MARGIN, FUN)

- First you specify the data (X)
- MARGIN is a vector giving the subscripts which the function will be applied over.
- E.g., for dataframes or matrix 1 indicates rows, 2 indicates columns.
- FUN is the function to be applied, can be user defined function as well

Apply family functions

Calculate the mean for each column

```
data(mtcars)
apply(mtcars, 2, mean)
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

Apply family functions

- Calculate the means for each row
- Rows have names in the data frame, that's why the resulting vector is named

```
apply(mtcars, 1, mean)
```

```
##          Mazda RX4      Mazda RX4 Wag         Datsun 710     Hornet 4 Drive
##            29.90727            29.98136            23.59818            38.73955
##  Hornet Sportabout             Valiant         Duster 360           Merc 240D
##            53.66455            35.04909            59.72000            24.63455
##           Merc 230            Merc 280           Merc 280C          Merc 450SE
##            27.23364            31.86000            31.78727            46.43091
##         Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental
##            46.50000            46.35000            66.23273            66.05855
##  Chrysler Imperial            Fiat 128         Honda Civic      Toyota Corolla
##            65.97227            19.44091            17.74227            18.81409
##       Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28
##            24.88864            47.24091            46.00773            58.75273
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus Europa
##            57.37955            18.92864            24.77909            24.88027
##      Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
##            60.97182            34.50818            63.15545            26.26273
```

Apply family functions

- The generic function **quantile()** produces sample quantiles corresponding to the given probabilities.

quantile(x, probs = seq(0, 1, 0.25), na.rm = F, names = T, type = 7, ...)

- If you want to get 0.25,0.5,0.75 quantiles, which are respectively 1st quartile, median and 3rd quartile, you can use the following function

quantile(x, probs=c(0.25,0.5,0.7))

Apply family functions

With apply function, the additional arguments for the FUN are defined within the apply function

```
a <- apply(mtcars, 2, quantile, probs=c(.25,.5, .75))
class(a)
```

```
## [1] "matrix" "array"
```

```
a
```

```
##        mpg cyl    disp    hp  drat      wt    qsec vs am gear carb
## 25% 15.425   4 120.825  96.5 3.080 2.58125 16.8925  0  0    3    2
## 50% 19.200   6 196.300 123.0 3.695 3.32500 17.7100  0  0    4    2
## 75% 22.800   8 326.000 180.0 3.920 3.61000 18.9000  1  1    4    4
```

Apply family functions

Recall the for loop you used to get the normalized data frame

```
normalize_df <- function(df){
  normalize <- function(x) {
    return ((x - min(x)) / (max(x) - min(x)))
  }

  df1 <-c()
  for (i in 1:ncol(df)){
    x <- normalize(df[,i])
    df1 <- cbind(df1,x)
  }
  df1 <- as.data.frame(df1)
  colnames(df1) <- colnames(df)
  return(df1)
}
```

Apply family functions

First define the min-max normalization function

```
normalize <- function(x) {
    return ((x - min(x)) / (max(x) - min(x)))
  }
```

Then use your function with apply

Apply family functions

The result is a matrix, transform it to data frame for convenience

```
mtcars_norm <- apply(mtcars, 2, normalize)
class(mtcars_norm)
```

```
## [1] "matrix" "array"
```

```
mtcars_norm <- as.data.frame(mtcars_norm)
head(mtcars_norm)
```

```
##                           mpg cyl      disp        hp      drat        wt
## Mazda RX4           0.4510638 0.5 0.2217511 0.2049470 0.5253456 0.2830478
## Mazda RX4 Wag       0.4510638 0.5 0.2217511 0.2049470 0.5253456 0.3482485
## Datsun 710          0.5276596 0.0 0.0920429 0.1448763 0.5023041 0.2063411
## Hornet 4 Drive      0.4680851 0.5 0.4662010 0.2049470 0.1474654 0.4351828
## Hornet Sportabout   0.3531915 1.0 0.7206286 0.4346290 0.1797235 0.4927129
## Valiant             0.3276596 0.5 0.3838863 0.1872792 0.0000000 0.4978266
##                           qsec vs am gear      carb
## Mazda RX4           0.2333333  0  1  0.5 0.4285714
## Mazda RX4 Wag       0.3000000  0  1  0.5 0.4285714
## Datsun 710          0.4892857  1  1  0.5 0.0000000
## Hornet 4 Drive      0.5880952  1  0  0.0 0.0000000
## Hornet Sportabout   0.3000000  0  0  0.0 0.1428571
## Valiant             0.6809524  1  0  0.0 0.0000000
```

Apply family functions

You can define the function inside the apply() as well

```
mtcars_norm1 <- apply(mtcars, 2, function(x) (x-min(x))/(max(x)-min(x)))
mtcars_norm1 <- as.data.frame(mtcars_norm1)
head(mtcars_norm1, n=4)
```

```
##                    mpg cyl      disp        hp      drat        wt      qsec
## Mazda RX4      0.4510638 0.5 0.2217511 0.2049470 0.5253456 0.2830478 0.2333333
## Mazda RX4 Wag  0.4510638 0.5 0.2217511 0.2049470 0.5253456 0.3482485 0.3000000
## Datsun 710     0.5276596 0.0 0.0920429 0.1448763 0.5023041 0.2063411 0.4892857
## Hornet 4 Drive 0.4680851 0.5 0.4662010 0.2049470 0.1474654 0.4351828 0.5880952
##                vs am gear      carb
## Mazda RX4       0  1  0.5 0.4285714
## Mazda RX4 Wag   0  1  0.5 0.4285714
## Datsun 710      1  1  0.5 0.0000000
## Hornet 4 Drive  1  0  0.0 0.0000000
```

Apply family functions

Check if two objects are identical

```
identical(mtcars_norm, mtcars_norm1)
```

```
## [1] TRUE
```

Apply family functions

- lapply is similar to apply, but it takes a list as an input, and returns a list as the output.
- It works with data frame as well, because in essence data frame is just a list of vectors of the same length.

```
lapply(mtcars,mean)
```

```
## $mpg
## [1] 20.09062
##
## $cyl
## [1] 6.1875
##
## $disp
## [1] 230.7219
##
## $hp
## [1] 146.6875
##
## $drat
## [1] 3.596563
##
## $wt
## [1] 3.21725
##
## $qsec
## [1] 17.84875
##
## $vs
## [1] 0.4375
##
## $am
```

Apply family functions

- Create a list with two matrices and a vector
- rnorm() is going to create random numbers from normal distribution

```
(l1 <- list(x = matrix(rnorm(50), ncol=5),
  y = matrix(rnorm(30), nrow=3), z = 1:25))
```

```
## $x
##              [,1]       [,2]       [,3]        [,4]       [,5]
##  [1,] -0.19489136 -0.7941918 -0.1752910 -0.04946844 -1.2701758
##  [2,] -0.14496833 -1.2264846 -0.3981449  0.11203074  1.6504955
##  [3,] -0.10107792 -0.2826523  0.6377617 -1.01602436 -1.1010598
##  [4,] -0.01599258 -1.0191802  0.9080170  0.77884997 -0.2406003
##  [5,] -0.39255068 -0.4436159 -0.3748354 -0.36006243  0.9607900
##  [6,] -0.03282268  0.8511504 -0.5910253 -0.60598934 -0.1610755
##  [7,] -0.15937129 -0.8297675  1.5178686 -1.33693398  0.6071177
##  [8,] -0.85843837  1.3073836 -1.1255644 -0.38853274 -0.4084859
##  [9,]  0.53231220  0.4767182 -0.3028556 -2.04291884  0.1240905
## [10,]  0.18977672  2.1423056 -0.2273742 -2.44537885  1.7038168
##
## $y
##             [,1]        [,2]       [,3]       [,4]       [,5]       [,6]
## [1,]  0.89694084 -0.07334004  0.5890275 -0.1335959 -0.5455186 -0.9282643
## [2,] -0.01366728  1.52614836 -0.3148742 -2.0731439  1.0455877  0.6526332
## [3,]  0.60737211 -0.42597677  1.8128315  0.7439440  2.0798658  0.5245335
##            [,7]       [,8]        [,9]     [,10]
## [1,] -0.9371794 -0.7520309  0.70610150 1.3554122
## [2,]  1.5750395  2.2212999  1.02665053 0.5600739
## [3,]  1.6128795  0.6594391 -0.09810949 0.4580111
##
## $z
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

Apply family functions

```
lapply(l1, mean)
```

```
## $x
## [1] -0.1323463
##
## $y
## [1] 0.478603
##
## $z
## [1] 13
```

Apply family functions

sapply()

The sapply() function behaves similarly to lapply(); the only real difference is in the return value. sapply() will try to simplify the result of lapply() if possible. Essentially, sapply() calls lapply() on its input and then applies the following algorithm:

- If the result is a list where every element is length 1, then a vector is returned
- If the result is a list where every element is a vector with the same length ($>$ 1), a matrix is returned.
- If neither of the above simplifications can be performed, then a list is returned.

Apply family functions

```
sapply(l1, mean)
```

```
##          x          y          z
## -0.1323463  0.4786030 13.0000000
```

```
lapply(l1, mean)
```

```
## $x
## [1] -0.1323463
##
## $y
## [1] 0.478603
##
## $z
## [1] 13
```

Apply family functions

- Say you want to create a new data frame with only numeric variables from the data frame movies.
- sapply will return a vector with value TRUE if the column is numeric and FALSE otherwise.

```
movies <- read.csv("movies3.csv", stringsAsFactors = F)
sapply(movies, is.numeric)
```

```
##              title        genre_first               year           duration
##              FALSE              FALSE               TRUE               TRUE
##      gross_adjusted     budget_adjusted              gross             budget
##               TRUE               TRUE               TRUE               TRUE
## cast_facebook_likes            reviews              index              Rated
##               TRUE               TRUE               TRUE              FALSE
##              Genre           Director             Writer             Actors
##              FALSE              FALSE              FALSE              FALSE
##               Plot           Language            Country             Awards
##              FALSE              FALSE              FALSE              FALSE
##          Metascore        imdbRating          imdbVotes         Production
##               TRUE               TRUE              FALSE              FALSE
##                DVD            Release      Release_Month        Release_Day
##              FALSE              FALSE               TRUE               TRUE
##       Release_year           OscarWon           OtherWin           OscarNom
##               TRUE               TRUE               TRUE               TRUE
##           OtherNom
##               TRUE
```

Apply family functions

Now you can use it to subset the data frame

```
movies_num <- movies[,sapply(movies, is.numeric)]
summary(movies_num)
```

```
##       year         duration     gross_adjusted       budget_adjusted
##  Min.   :1920   Min.   : 37.0   Min.   :       973   Min.   :      290
##  1st Qu.:1999   1st Qu.: 95.0   1st Qu.:  16264972   1st Qu.: 15303856
##  Median :2004   Median :105.0   Median :  45148894   Median : 35386325
##  Mean   :2003   Mean   :109.6   Mean   :  84532189   Mean   : 51922575
##  3rd Qu.:2010   3rd Qu.:119.0   3rd Qu.: 102032576   3rd Qu.: 72596990
##  Max.   :2016   Max.   :330.0   Max.   :3503192931   Max.   :354732272
##
##      gross               budget          cast_facebook_likes    reviews
##  Min.   :      703   Min.   :      218   Min.   :     0      Min.   :   2.0
##  1st Qu.: 12021631   1st Qu.: 11000000   1st Qu.:  2216      1st Qu.: 197.0
##  Median : 34514650   Median : 25000000   Median :  4529      Median : 358.5
##  Mean   : 57613345   Mean   : 40183590   Mean   : 12272      Mean   : 498.8
##  3rd Qu.: 75073078   3rd Qu.: 55000000   3rd Qu.: 16759      3rd Qu.: 624.5
##  Max.   :760505847   Max.   :300000000   Max.   :656730      Max.   :5312.0
##
##      index          Metascore        imdbRating    Release_Month
##  Min.   : 1.021   Min.   :  6.00   Min.   :1.600   Min.   : 1.000
##  1st Qu.: 1.124   1st Qu.: 40.00   1st Qu.:5.800   1st Qu.: 4.000
##  Median : 1.298   Median : 53.00   Median :6.500   Median : 7.000
##  Mean   : 1.483   Mean   : 53.11   Mean   :6.388   Mean   : 6.678
##  3rd Qu.: 1.471   3rd Qu.: 66.00   3rd Qu.:7.100   3rd Qu.:10.000
##  Max.   :18.855   Max.   :100.00   Max.   :9.300   Max.   :12.000
##                   NA's   :274      NA's   :27      NA's   :445
##   Release_Day     Release_year     OscarWon          OtherWin
##  Min.   : 1.00   Min.   :1920   Min.   : 0.0000   Min.   :  0.000
##  1st Qu.: 9.00   1st Qu.:1998   1st Qu.: 0.0000   1st Qu.:  0.000
##  Median :16.00   Median :2004   Median : 0.0000   Median :  2.000
##  Mean   :15.99   Mean   :2003   Mean   : 0.1834   Mean   :  6.707
##  3rd Qu.:23.00   3rd Qu.:2009   3rd Qu.: 0.0000   3rd Qu.:  5.000
```

Section 8

Data Transformation: dplyr

Grammar of data manipulation

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- **mutate()** adds new variables that are functions of existing variables
- **select()** picks variables based on their names.
- **filter()** picks cases based on their values.
- **summarise()** reduces multiple values down to a single summary.
- **arrange()** changes the ordering of the rows.

These all combine naturally with group_by() which allows you to perform any operation "by group".

Grammar of data manipulation

- Not necessary, but dplyr works the best with pipe-like operator from **magrittr** package
- %>% operator takes the object from its left-hand side and uses it as an argument in the function on the right-hand side

Grammar of data manipulation

- The "pipe" operation is a handy tool to make your code more legible: %>%.

Key points:

- It takes the output of your previous operation and uses it as an input to your next operation.
- You can determine where the previous argument goes with the period symbol, . , which acts as a placeholder.
- Understand how to use it by replacing the pipe operation with "then" (in your mind, not in the code). For example, **filter(data, ...) %>% select(...)** filters first then selects columns from the output of filter.

Grammar of data manipulation

Filtering USA games only

```
summer <- read.csv("summer.csv", stringsAsFactors = F)
```

```
summer_usa <- summer %>% filter(Country=="USA")
```

```
table(summer_usa$Country, summer_usa$Medal)
```

```
##
##       Bronze Gold Silver
##   USA   1098 2235   1252
```

Grammar of data manipulation

Filter and group_by

```
summer %>%
  filter(Country %in% c("USA", "FRA", "GBR")) %>%
  group_by(Country) %>%
  summarise(Count=n())
```

```
## # A tibble: 3 x 2
##   Country Count
##   <chr>   <int>
## 1 FRA      1396
## 2 GBR      1720
## 3 USA      4585
```

- The result is a tibble
- Count=n() creates a new variable named Count with frequencies, n() calculates frequencies

Grammar of data manipulation

Number of medals by country (is this long or wide format?)

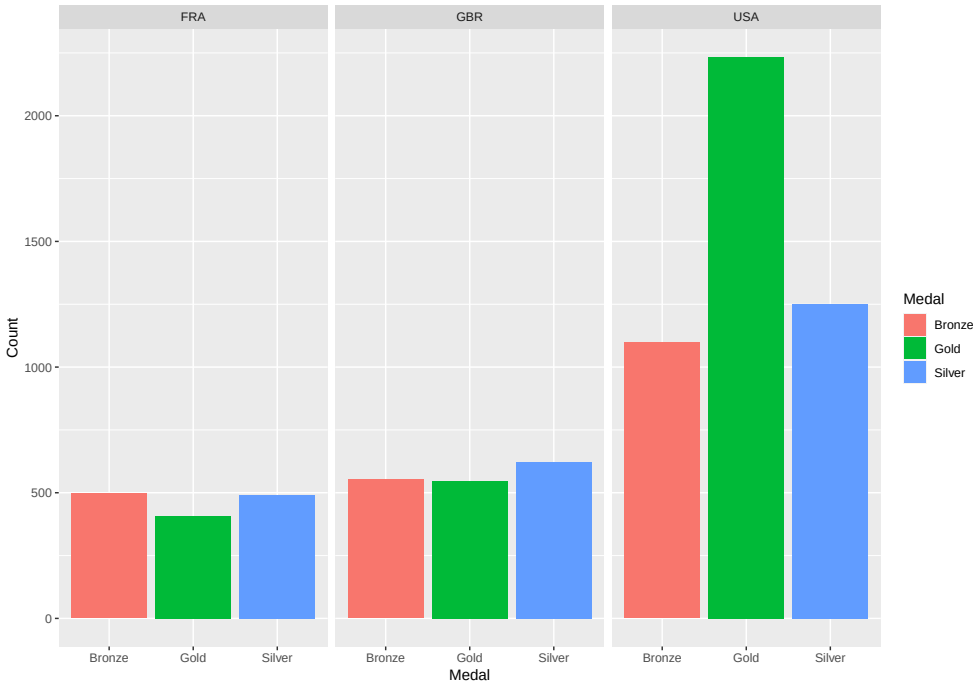
```
summer %>%
  filter(Country %in% c("USA", "FRA", "GBR")) %>%
  group_by(Country, Medal) %>%
  summarise(Count=n())
```

```
## # A tibble: 9 x 3
## # Groups:   Country [3]
##   Country Medal  Count
##   <chr>   <chr>  <int>
## 1 FRA     Bronze   497
## 2 FRA     Gold     408
## 3 FRA     Silver   491
## 4 GBR     Bronze   553
## 5 GBR     Gold     546
## 6 GBR     Silver   621
## 7 USA     Bronze  1098
## 8 USA     Gold    2235
## 9 USA     Silver  1252
```

Grammar of data manipulation

Using ggplot with dplyr

```
summer %>%
  filter(Country %in% c("USA", "FRA", "GBR")) %>%
  group_by(Country, Medal) %>%
  summarise(Count=n()) %>%
  ggplot(aes(x=Medal, y=Count, fill=Medal))+geom_bar(stat="identity")+
  facet_grid(.~Country)
```



Grammar of data manipulation

- Let's summarize gross box office by genre
- Then arrange the dataset by descending order on average gross box office

```
movies <- read.csv("movies3.csv")
```

```
sum_movie <- movies %>%
  group_by(genre_first) %>%
  summarise(count=n(), mean = mean(gross_adjusted),
            standard_dev = sd(gross_adjusted)) %>%
  arrange(desc(mean))
```

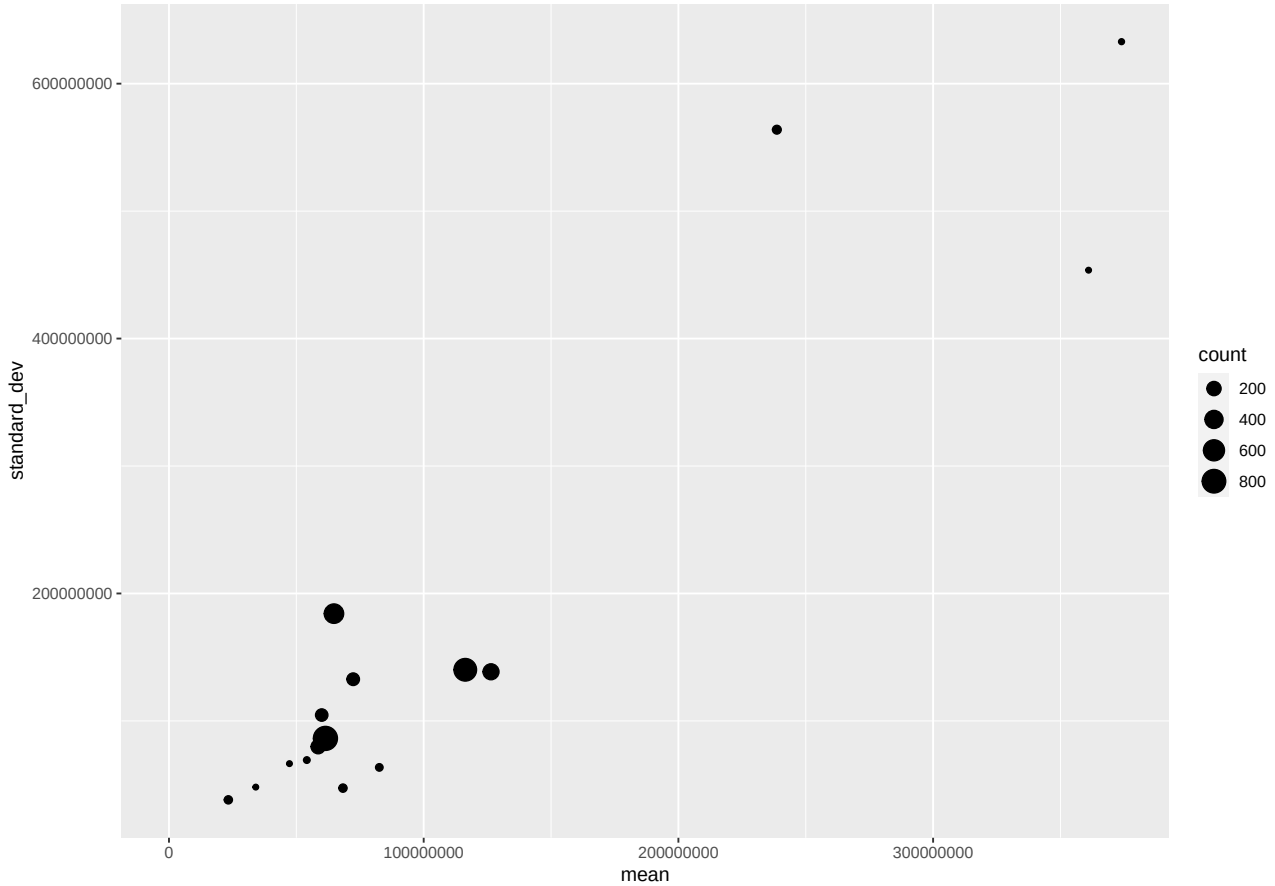
```
head(sum_movie)
```

```
## # A tibble: 6 x 4
##   genre_first count       mean standard_dev
##   <chr>       <int>      <dbl>        <dbl>
## 1 Family          3 373974561.   632947270.
## 2 Musical         2 361037936.   453660835.
## 3 Animation      35 238648706.   563888709.
## 4 Adventure     281 126427147.   138554095.
## 5 Action        721 116323930.   140130142.
## 6 Mystery        16  82577251.    63530236.
```

Grammar of data manipulation

Bubble chart with size as the number of movies in the given category

```
ggplot(sum_movie, aes(x=mean, y=standard_dev, size=count))+geom_point()
```



Grammar of data manipulation

Add labels to the plot using geom_text()

```
ggplot(sum_movie, aes(x=mean, y=standard_dev, size=count))+
  geom_text(aes(label=genre_first), size=3, check_overlap = T)+
  labs(x="Mean", y="Standard Deviation")
```