UTSA®

ALVAREZ
College of Business
The University of Texas at San Antonio

Introduction to Programming in R

Module 3: Communication

Learning Objectives

- Communicating with Markdown
- Creating .Rmd files
- Output and knitting .Rmd
- Visualization with ggplot2
- Creating a plot with ggplot2
- Scatterplots, bar charts, histograms, and other plots
- Aesthetics, facets, plotting objects, and trendlines
- Overlaying multiple plots

Communicating

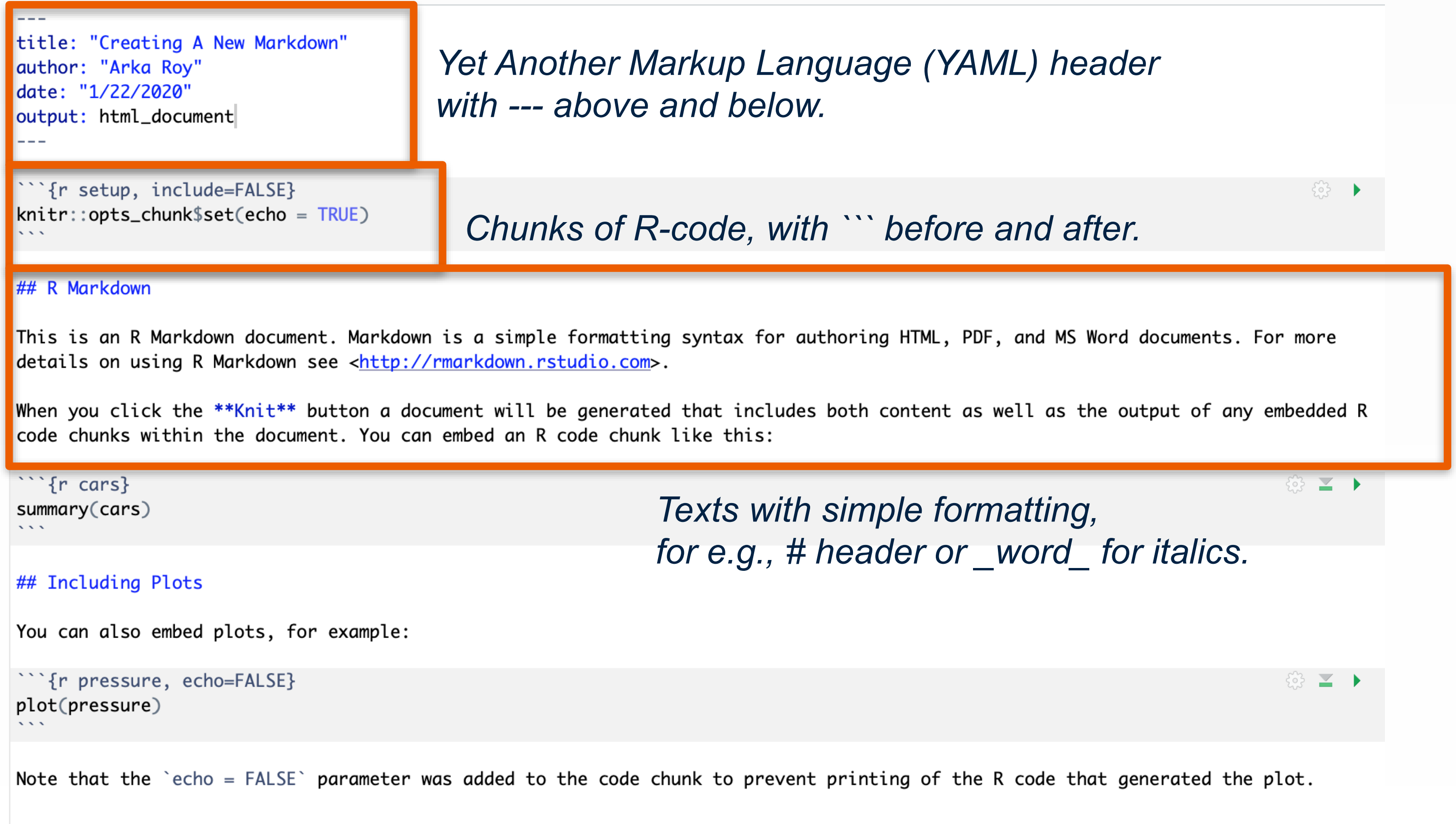
- R markdown lets you create documents that can help you communicate your findings to decision-makers (including yourself).
- Elegant and concise way to record and present your code, results, and thoughts.
- Makes research reproducible, so that others can easily understand and replicate, if needed.
- I find this particularly helpful working with graduate students. They can show me what they did, exactly, and provide narration with it.
- There are some basic Markdown syntax that we'll cover here. As you progress with this program and in your career, you'll use more.
- Markdown can output into .pdf, .doc, .html, and other popular file formats.
- Also cover some plotting techniques to produce elegant visuals. There will be more on visuals in the Visualization course.

R Markdown Basics

- Markdown file (.Rmd) can help with:
  - Communicating with decision-makers with focus on insights and takeaways, rather than code.
  - Collaborating with other researchers, as well as future you, with focus of understanding the code and the results.
  - Notetaking for other users, as well as future you, with a focus of documenting your thought process.
- Merges many external packages to work. So the built-in help function *?* is limited.
  - R Markdown Cheat Sheet: *Help > Cheatsheets > R Markdown Cheat Sheet*,
  - R Markdown Reference Guide: *Help > Cheatsheets > R Markdown Reference Guide.*
- Markdown uses “rmarkdown” package, but R-studio comes with it loaded.

What .Rmd looks like

- Typical markdown looks something like this… with code and prose mixed.



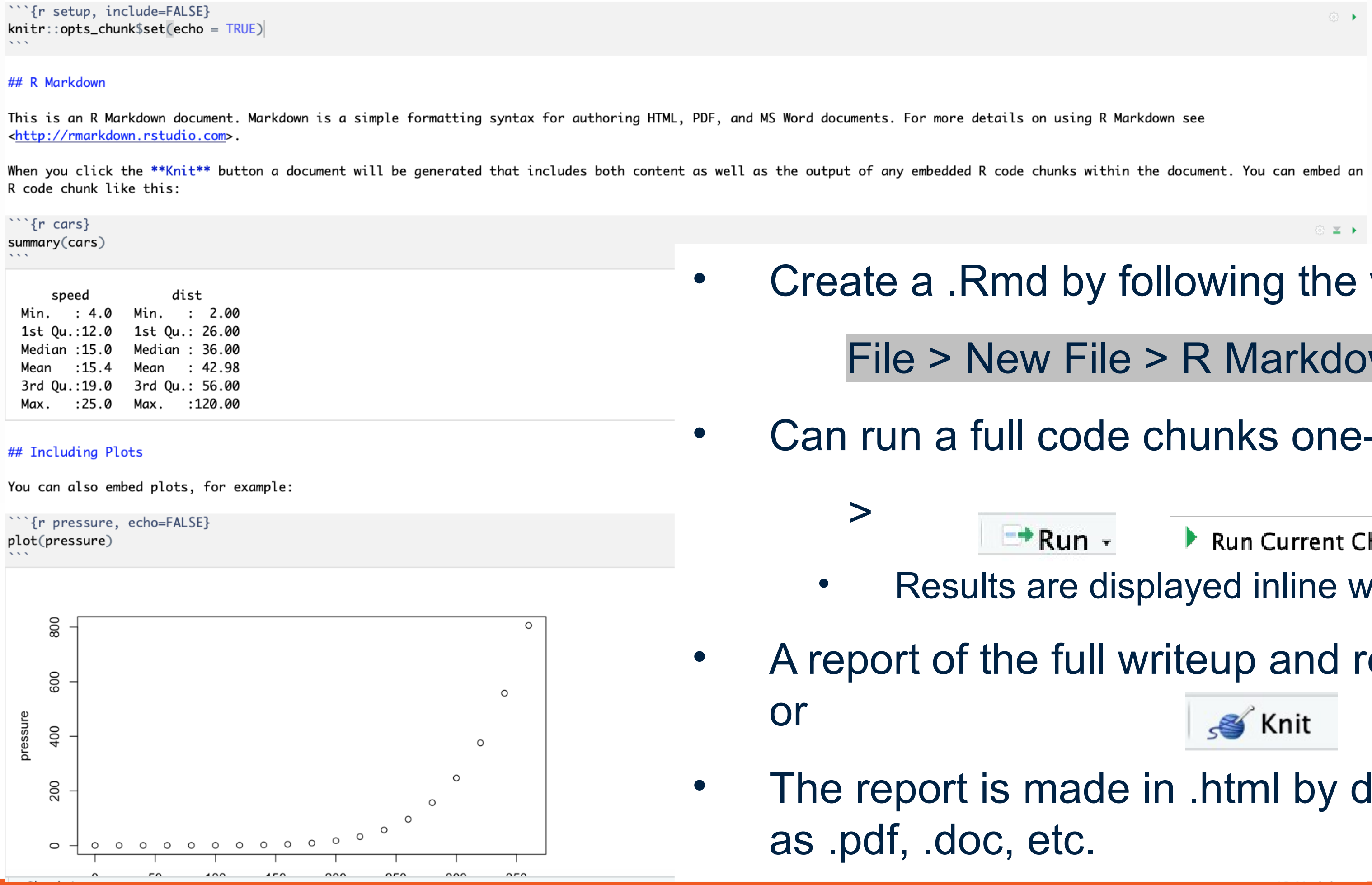
~~~
---
title: "Creating A New Markdown"
author: "Arka Roy"
date: "1/22/2020"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more
details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R
code chunks within the document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

## Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.
~~~

Yet Another Markup Language (YAML) header with --- above and below.

Chunks of R-code, with ``` before and after.

Texts with simple formatting, for e.g., # header or _word_ for italics.

Creating and Running .Rmd



~~~
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see
<http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an
R code chunk like this:

```{r cars}
summary(cars)
```

     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  

## Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
plot(pressure)
```
~~~

- Create a .Rmd by following the wizard at:

  File > New File > R Markdown

- Can run a full code chunks one-at-a-time using *cmd/ctrl+shift+return or*

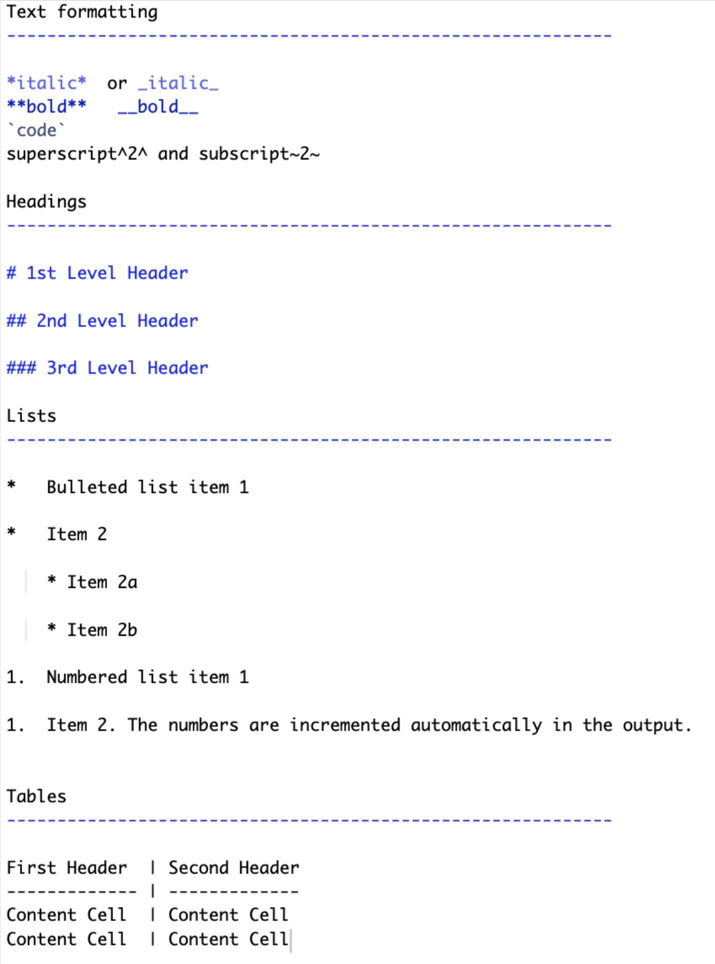
  > Run ▾ ▶ Run Current Chunk ⇧⌘↩

  - Results are displayed inline with text.

- A report of the full writeup and results can be knitted by cmd/ctrl+shift+K or Knit

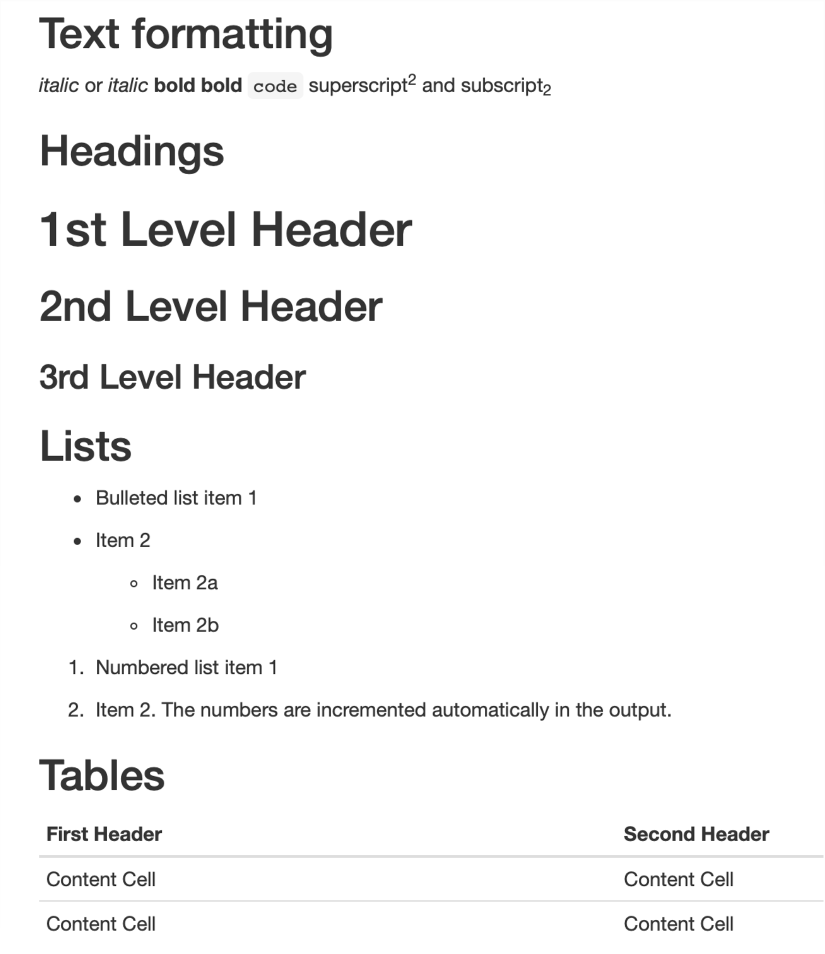
- The report is made in .html by default, but can be also done as .pdf, .doc, etc.

Text formatting in .Rmd



```
Text formatting
-------------------------------------------------------------

*italic*  or _italic_
**bold**    __bold__
`code`
superscript^2^ and subscript~2~

Headings
-------------------------------------------------------------

# 1st Level Header

## 2nd Level Header

### 3rd Level Header

Lists
-------------------------------------------------------------

*   Bulleted list item 1

*   Item 2

    * Item 2a

    * Item 2b

1.  Numbered list item 1

1.  Item 2. The numbers are incremented automatically in the output.


Tables
-------------------------------------------------------------

First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
```

Knit



Text formatting

italic or *italic* **bold** **bold** `code` superscript^2 and subscript$_2$

Headings

1st Level Header

2nd Level Header

3rd Level Header

Lists

- Bulleted list item 1
- Item 2
  - Item 2a
  - Item 2b

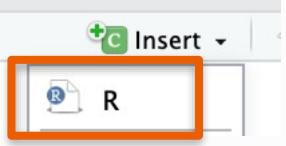
1. Numbered list item 1
2. Item 2. The numbers are incremented automatically in the output.

Tables

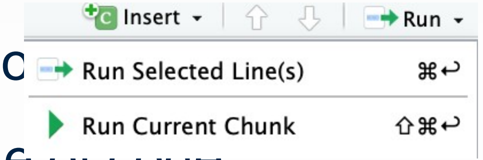
First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

Code Chunks in .Rmd

- Insert a code chunk in markdown by *cmd/ctrl+shift+I* or manually typing in ```*{r}* and ``` before and after code.

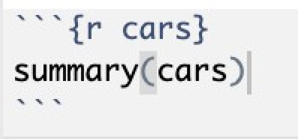


- You can run a code chunk by *cmd/ctrl+shift+return* o

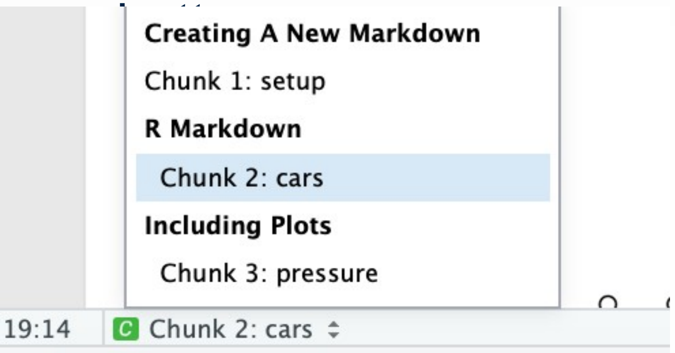


- Using *cmd/ctrl+return* will still only run the single line of code.

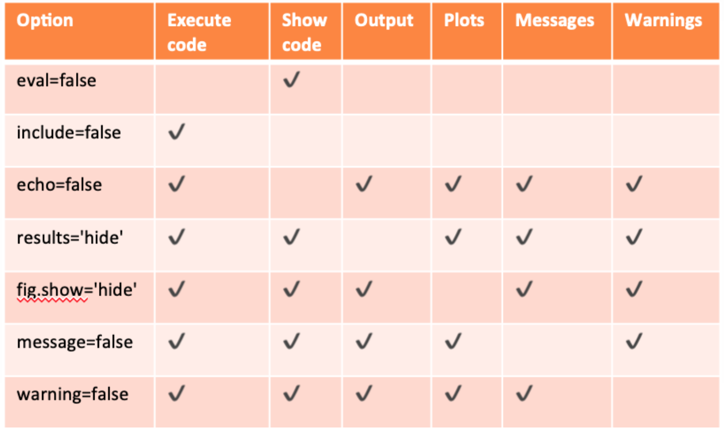
- Inside the ```*{r* and *}* you can define your own name for the chunk. For example, the 2nd chunk is named cars and it can be easily accessed via the navigation pane at the



~~~
```{r cars}
summary(cars)
```
~~~



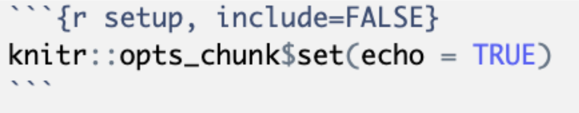
- Chunks have optional arguments:



Option	Execute code	Show code	Output	Plots	Messages	Warnings
eval=false		✓				
include=false	✓					
echo=false	✓		✓	✓	✓	✓
results='hide'	✓	✓		✓	✓	✓
fig.show='hide'	✓	✓	✓		✓	✓
message=false	✓	✓	✓	✓		✓
warning=false	✓	✓	✓	✓	✓	

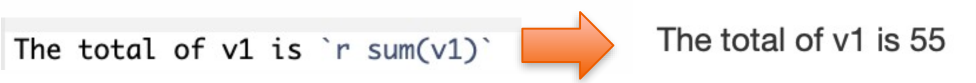
Output from .Rmd

- Common file format outputs:
  - .pdf: need access to a LaTex compiler, such as MikTex or TexWorks.
  - .doc: helpful for reports in Microsoft Word.
  - .rft: rich text format can also be used.
  - github_document: helpful for sharing code, analysis, & prose on the net.
- You can set global options for the .Rmd file:



~~~
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
~~~

- When sharing with decision-makers, use *echo=FALSE*.
- For sharing *.html,* you can hide code by default, and make it accessible by a mouse click. Use *code_folding: hide* in header.
- Can also do inline code- e.g.



```
The total of v1 is `r sum(v1)`
```

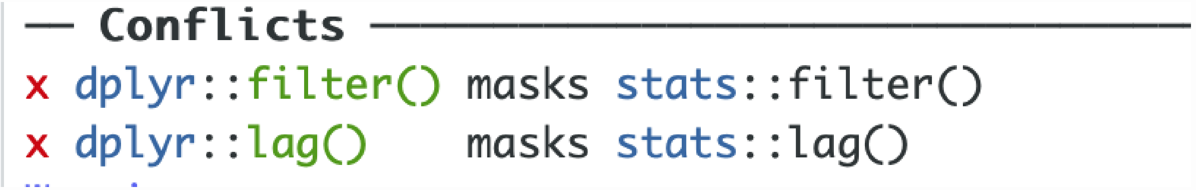
The total of v1 is 55

Other .Rmd outputs

- R Notebook - like a lab notebook, just for R tasks.
  - .html document: communicate with decision-makers
  - .html notebook: collaborate with data scientists and yourself.
  - Unlike .html document, .html notebook will show full source code.
  - Can be viewed in browser or R-studio.
- Presentations – less visual control than powerpoint but easier to transfer data science commands.
  - New slides starts at # header or ## subheader
  - Horizontal line with ***
  - Can make two types of html presentations or pdf presentation via beamer.

Tidyverse

- Will use the *tidyverse* package throughout the next few modules.
  - Use *install.packages("tidyverse")* ≋ first time only
  - Load by *library(tidyverse)* ≋ everytime

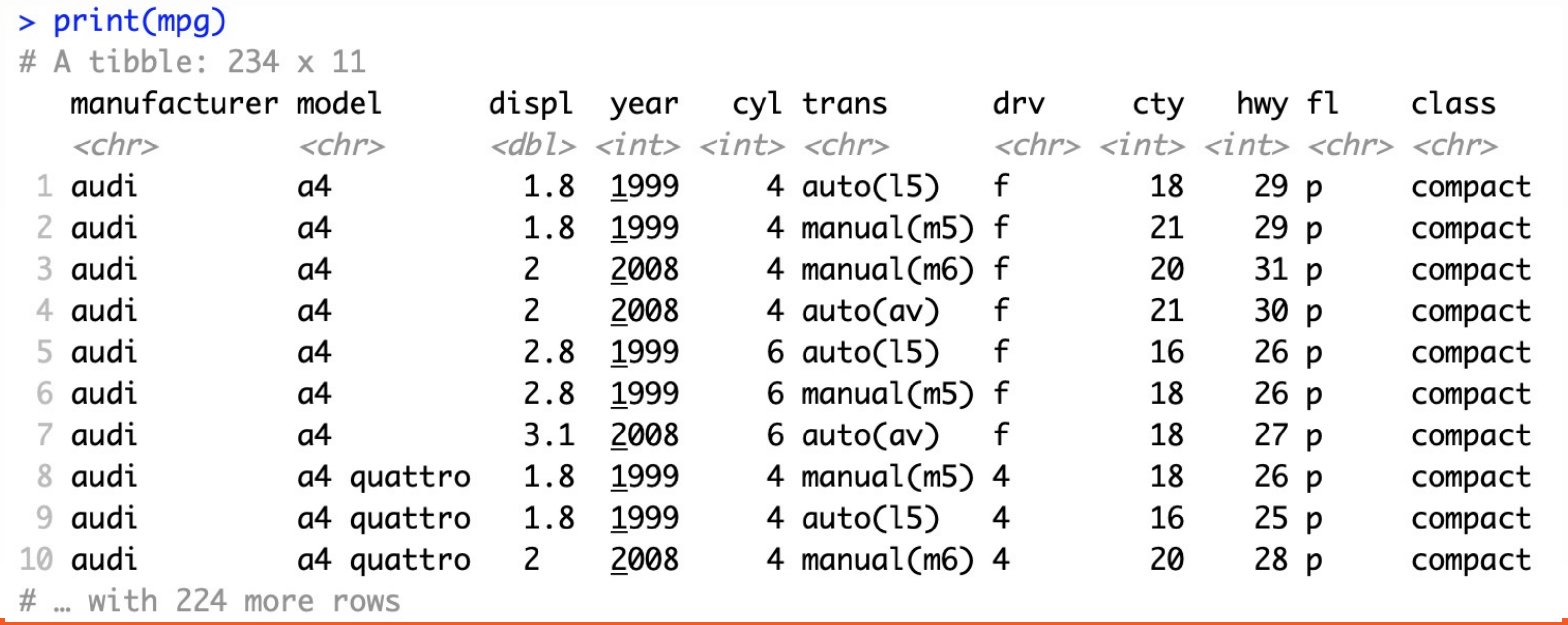


```
— Conflicts ——————————————————————————————
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

- What does this conflict mean?
  - Base R has a stats package where it has the functions *filter()* and *lag()*.
  - The *tidyverse* package has the same name functions. So it overwrites it.
  - Not to worry, the *tidyverse* versions do the same stuff and more…

ggplot2 package

- R has many plotting functions and packages, but ggplot2 package is very elegant yet powerful.
- It is built into the tidyverse package, so you will need to install it or load it separately.
- Throughout this module let us use another mileage data that’s built into ggplot2 package. To load: *data("mpg")* and *print(mpg)*.

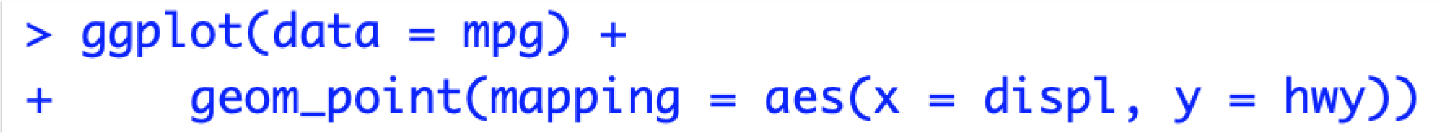


```
> print(mpg)
# A tibble: 234 x 11
   manufacturer model       displ  year   cyl trans      drv     cty   hwy fl    class  
   <chr>        <chr>       <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>  
 1 audi         a4            1.8  1999     4 auto(l5)   f        18    29 p     compact
 2 audi         a4            1.8  1999     4 manual(m5) f        21    29 p     compact
 3 audi         a4            2    2008     4 manual(m6) f        20    31 p     compact
 4 audi         a4            2    2008     4 auto(av)   f        21    30 p     compact
 5 audi         a4            2.8  1999     6 auto(l5)   f        16    26 p     compact
 6 audi         a4            2.8  1999     6 manual(m5) f        18    26 p     compact
 7 audi         a4            3.1  2008     6 auto(av)   f        18    27 p     compact
 8 audi         a4 quattro    1.8  1999     4 manual(m5) 4        18    26 p     compact
 9 audi         a4 quattro    1.8  1999     4 auto(l5)   4        16    25 p     compact
10 audi         a4 quattro    2    2008     4 manual(m6) 4        20    28 p     compact
# … with 224 more rows
```

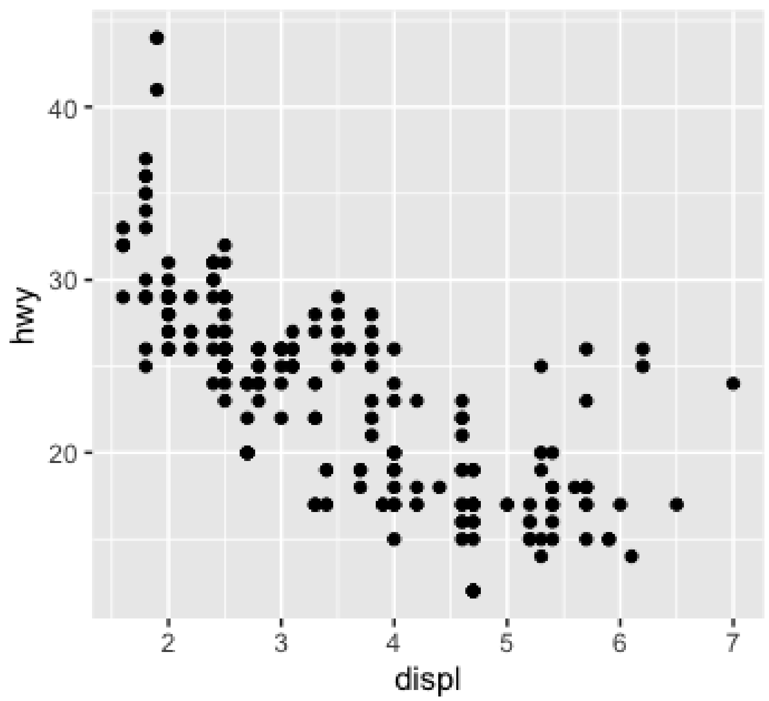
?mpg to find out more details about variables.

Creating a *ggplot*

- Question: Do cars with bigger engines use more gas than cars with smaller engines?
- Let us plot hwy (miles per gallon on the highway) against displ (car's engine size, in liters).



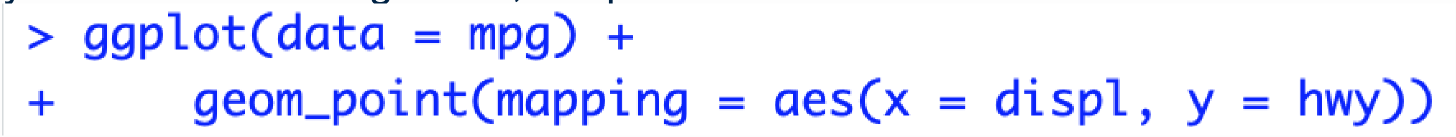
```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy))
```



- As expected, we see a negative relationship.
- Cars with bigger engines have lower mpg.

ggplot()

- To plot with ggplot2, you start with the function *ggplot()*, which creates a coordinate system for the plot.
    - If you try this, it'll just be a blank canvas.
- Then, the first argument is the data. Here, we used the mpg data.
    - Even when you add the data argument, the plot remains blank.



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

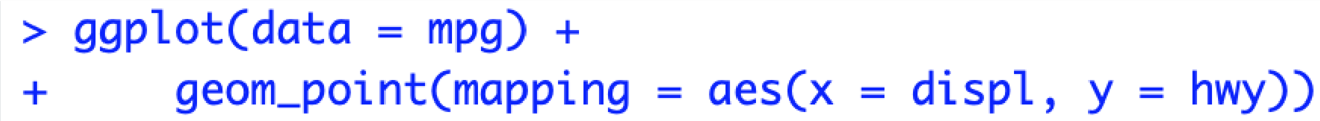
- Then, you add layers to the plot using +.
    - Note that the + must be added to the end of the line if the code is being broken into multiple lines.
- Then *geom* functions controls the type of plot. Many geom functions!
    - Here, *geom_point()* adds a layer of points making it a scatterplot.
    - *Geom* functions take mapping arguments that defines how to map the data to visuals.

ggplot() (cont.)

- The basic template of *ggplot()* is

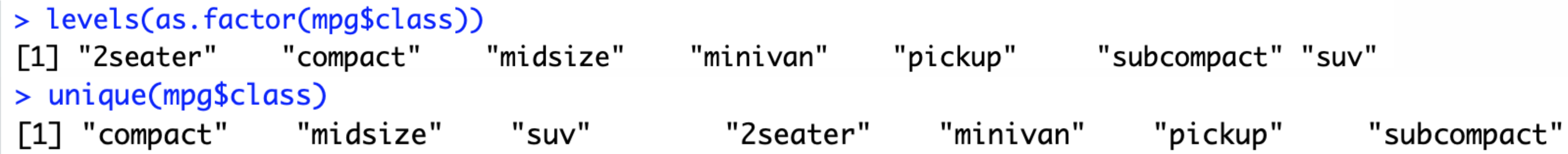
  ggplot(data = DATA*) +*
  *<GEOM_FUNCTION>(mapping = aes(*MAPPINGS*))*

- The mapping argument uses the aesthetic function *aes()*, and the *x* and *y* arguments of *aes()* specifies which variable goes on x-axis and y-axis.



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

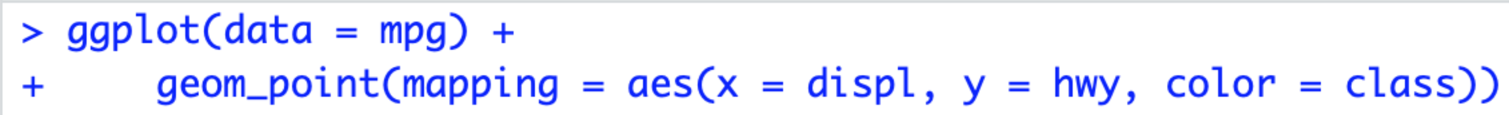
- In our first example, we plotted the engine size on the x-axis and mileage on the y-axis.
- You can also add a 3rd factor variable to a 2D scatterplot by using *aes()*.
  - For e.g., Color the observations by class of cars



```
> levels(as.factor(mpg$class))
[1] "2seater"    "compact"    "midsize"    "minivan"    "pickup"     "subcompact" "suv"
> unique(mpg$class)
[1] "compact"    "midsize"    "suv"        "2seater"    "minivan"    "pickup"     "subcompact"
```

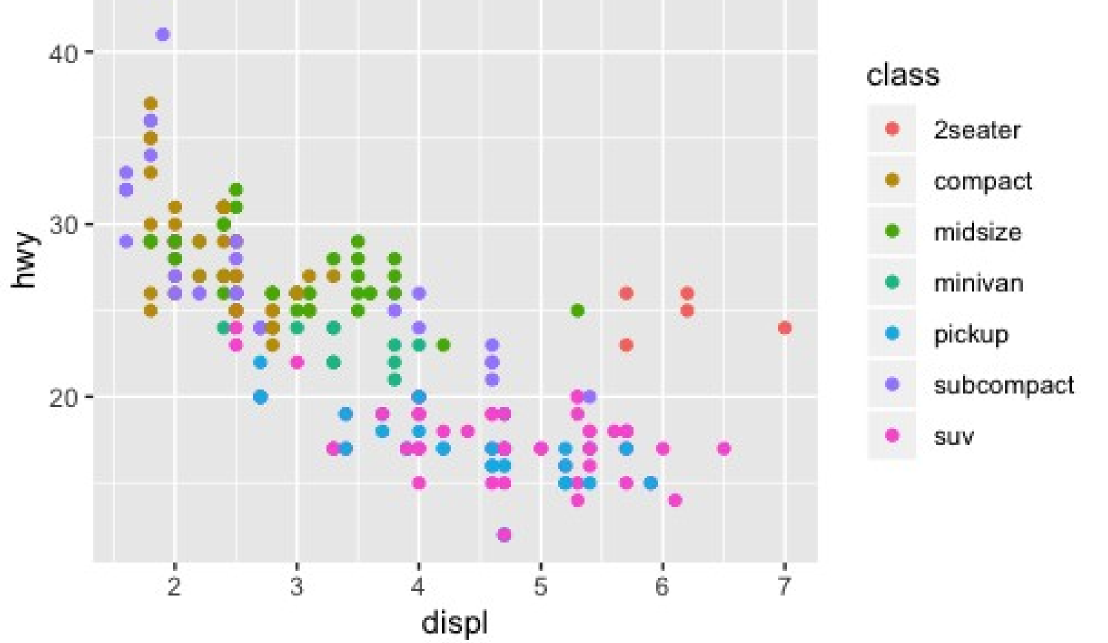
Aesthetics

- aes() includes visual properties like size, shape, or color of points.
- So to color the observations by class of car -



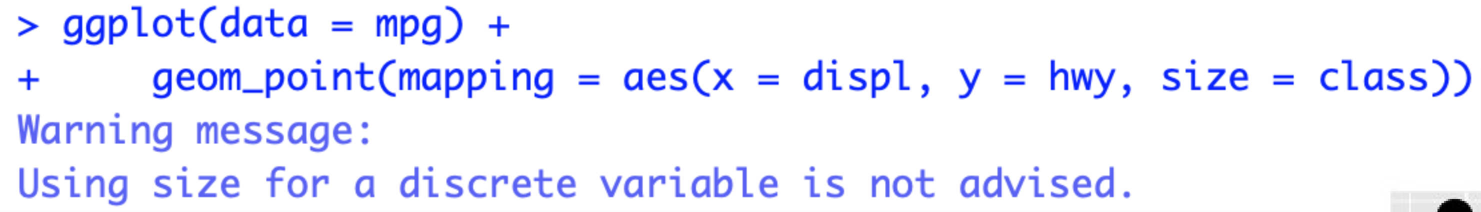
```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

- *ggplot()* will assign unique properties to the arguments level of the factor variable.
- For e.g. different colors are assigned to the 7 levels of class variable.
- *ggplot()* will also add a legend.



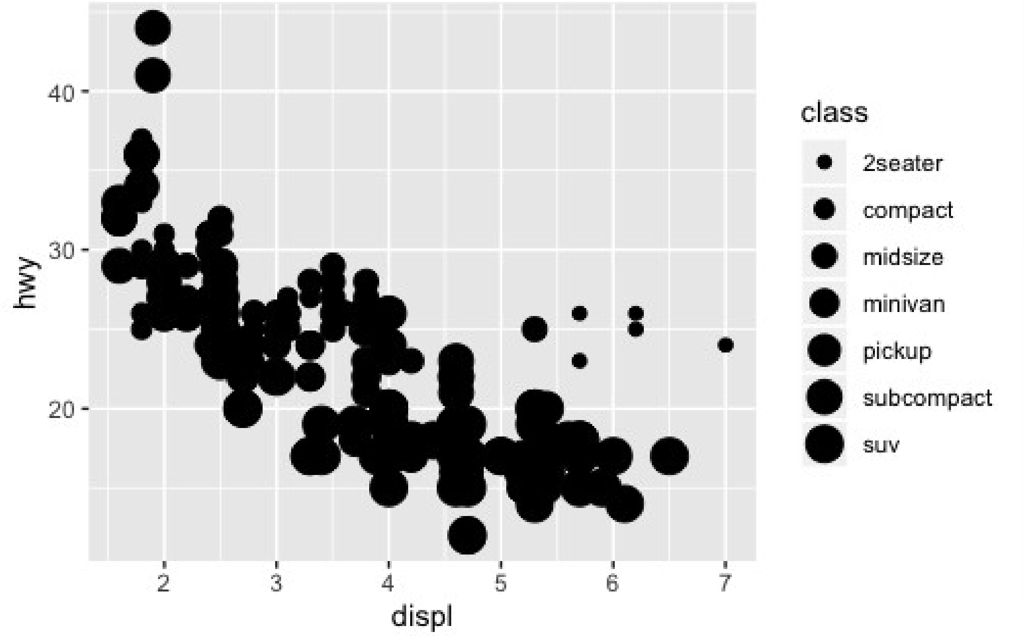
Aesthetics (cont.)

- You can also change size of the points



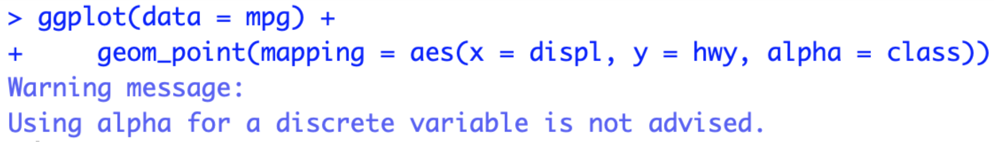
```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy, size = class))
Warning message:
Using size for a discrete variable is not advised.
```

- The warning here shows that we really should not map an unordered variable to an ordered aesthetic.
- Instead, we could use the number of cylinders, for example.

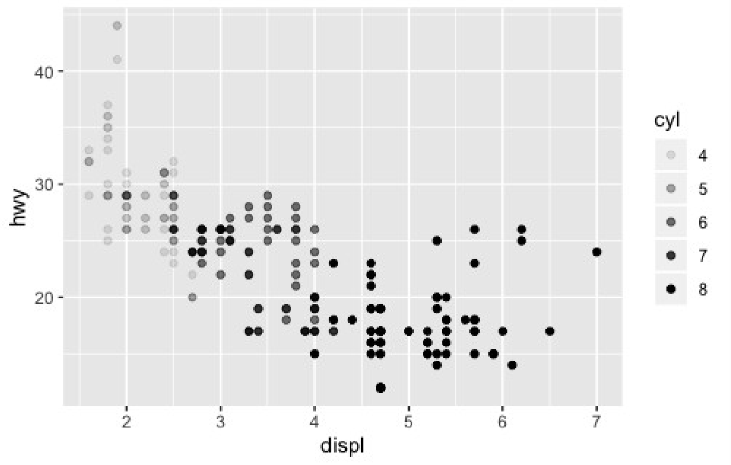


Aesthetics (cont..)

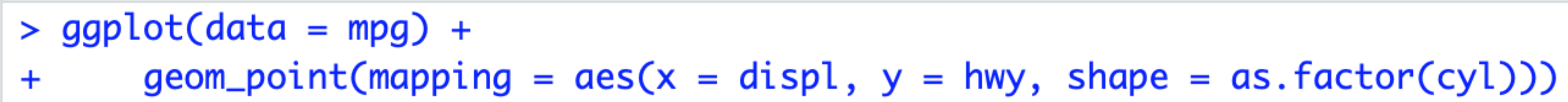
- *aes()* also has an *alpha* argument that maps to opacity of the points.



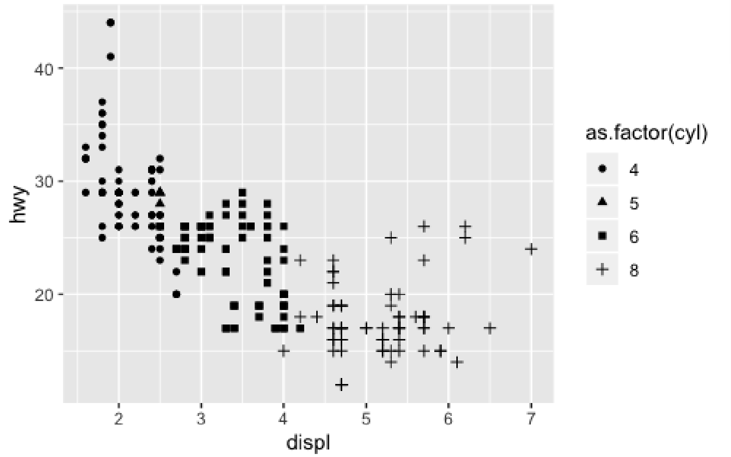
```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
Warning message:
Using alpha for a discrete variable is not advised.
```



- Like *size*, it is better to use ordered numbers for *alpha,* so I used *cyl*.
- The argument *shape* controls the type of symbols used as points



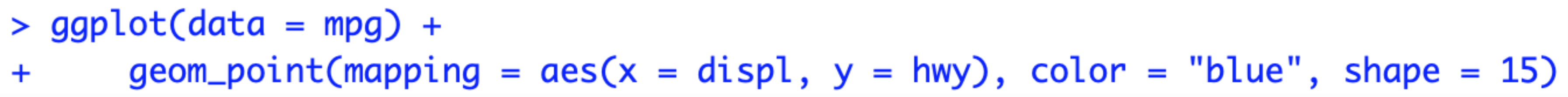
```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy, shape = as.factor(cyl)))
```



- *ggplot()* only can plot 6 shapes at a time.
  - Additional groups will not be plotted. You can see this if you do *shape = class.*
- *shape* only accepts categorical variables.
  - You'll see the error if you don't use *as.factor()*.

Aesthetics (cont…)

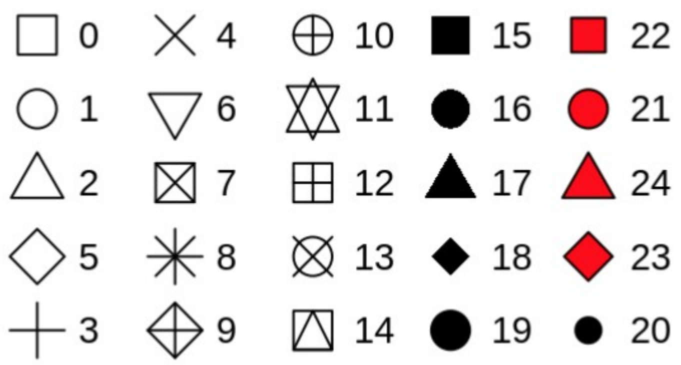
- You can also change the aesthetics of the entire *geom* (plot), manually.
- To do so, place the aesthetic argument outside of *aes()*.
- For e.g. if you wanted all the points to be blue squares.

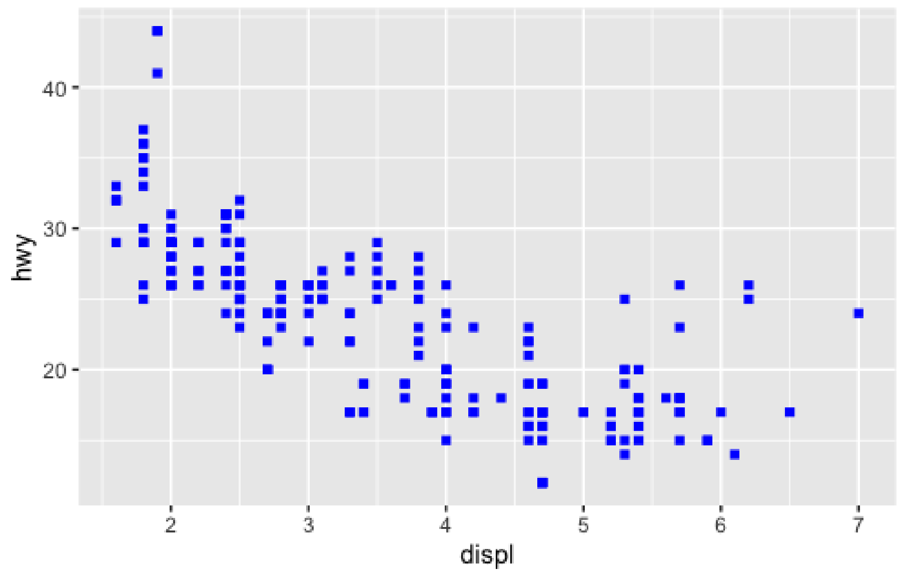


```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy), color = "blue", shape = 15)
```

Note that

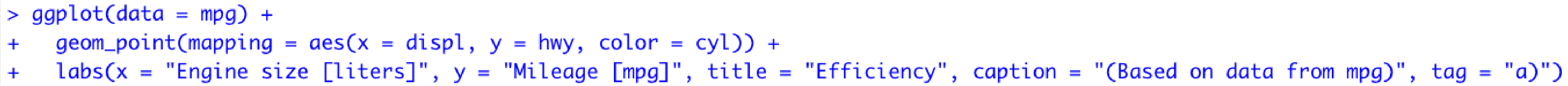
- Name of color is a character string.
- Size of the symbol is in mm.
- Shape of symbol is a number.





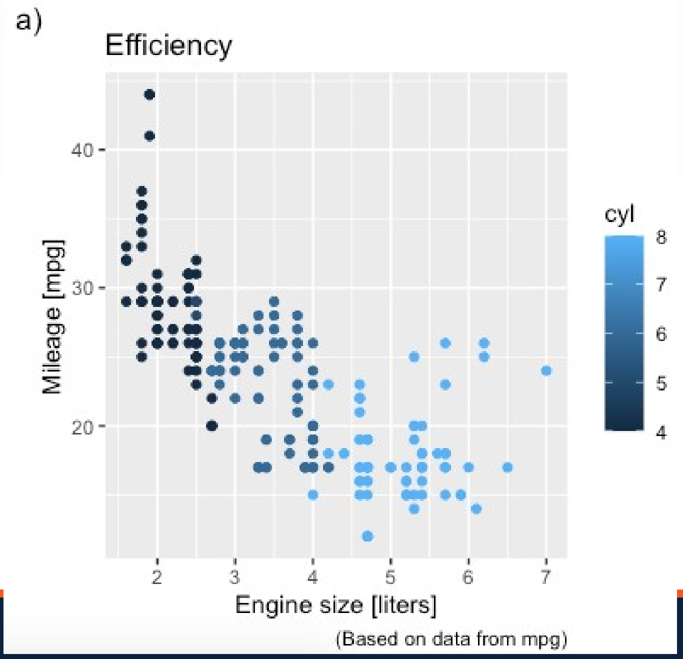
Aesthetics (cont….)

- The *labs()* function allows you to change the default labels given by *ggpot2*.
- Use *labs(x = "*string*", y = "*string*")* to assigned new axis labels.



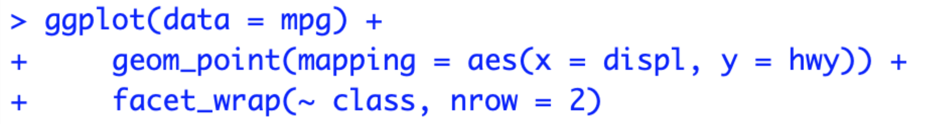
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy, color = cyl)) +
+   labs(x = "Engine size [liters]", y = "Mileage [mpg]", title = "Efficiency", caption = "(Based on data from mpg)", tag = "a)")
```

Arguments like *title, caption,* and *tag* are all useful for making descriptive plots.

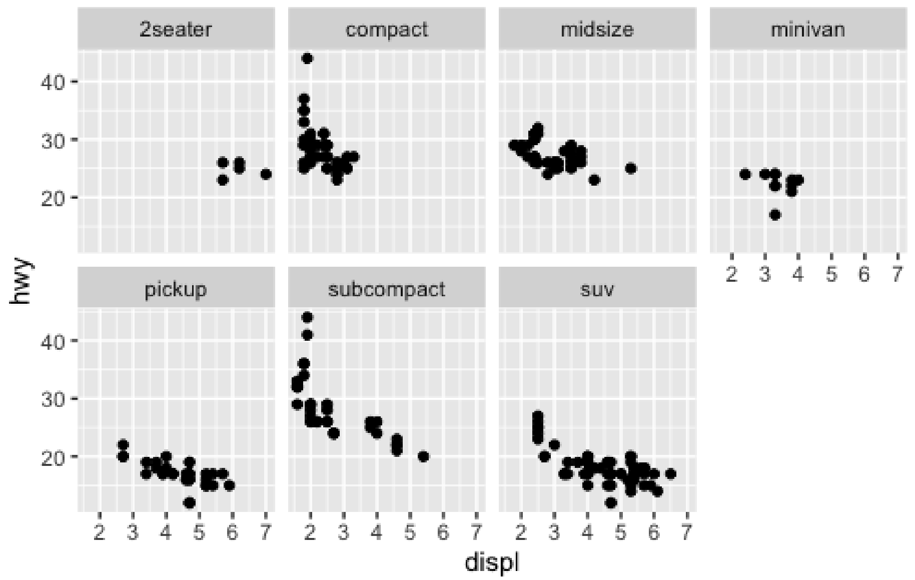


Facets

- We can also make subplots within a plot - like a matrix of subplots.
- To make subplot based on using a factor variable use *facet_wrap()*.
- The first argument is a formula – this is a data structure in R – not an “equation”.
    - A formula structure is created using a tilde followed by a variable.



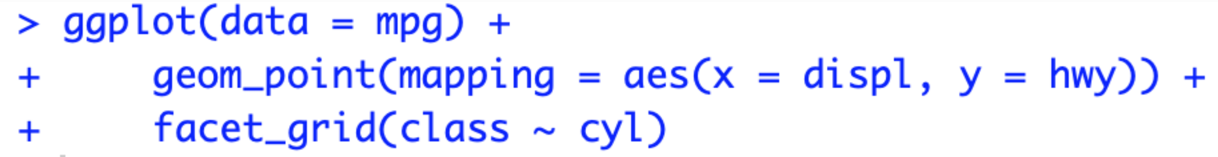
```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy)) +
+     facet_wrap(~ class, nrow = 2)
```



- You can specify the number of rows using *nrow* argument.
- The variable to pass to *facet_wrap()* should not be continuous.

Facets (cont.)

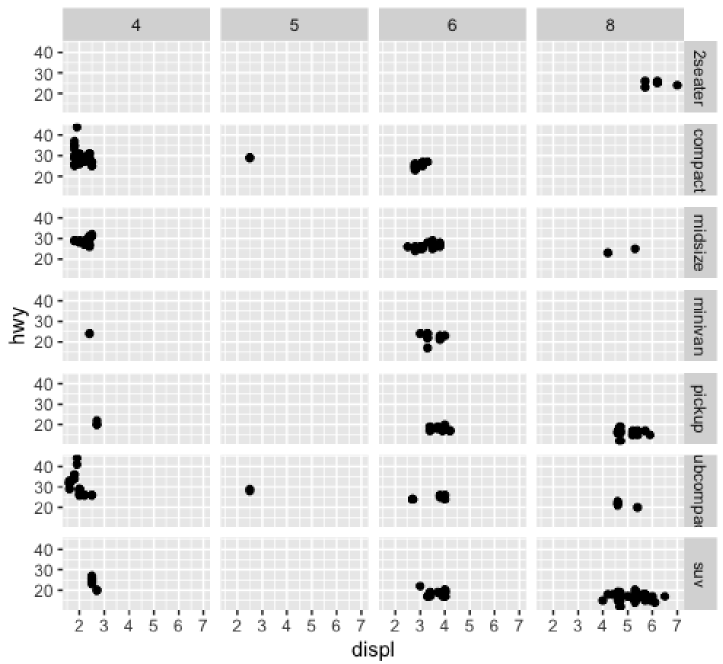
- You can use *facet_grid()* to facet using two variable.



```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy)) +
+     facet_grid(class ~ cyl)
```

- The first argument is also a formula, except with two variables.

- Unlike *facet_wrap()*, *facet_grid()* requires two variables separated by .
  - Format: rows variable column variable.

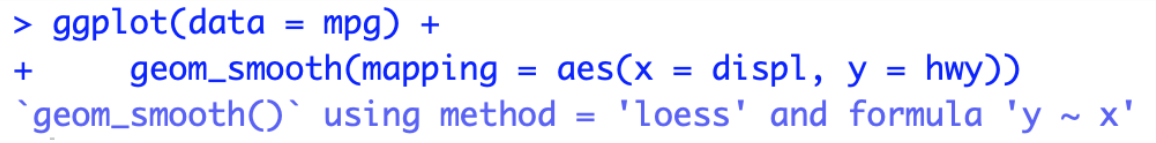


Geoms

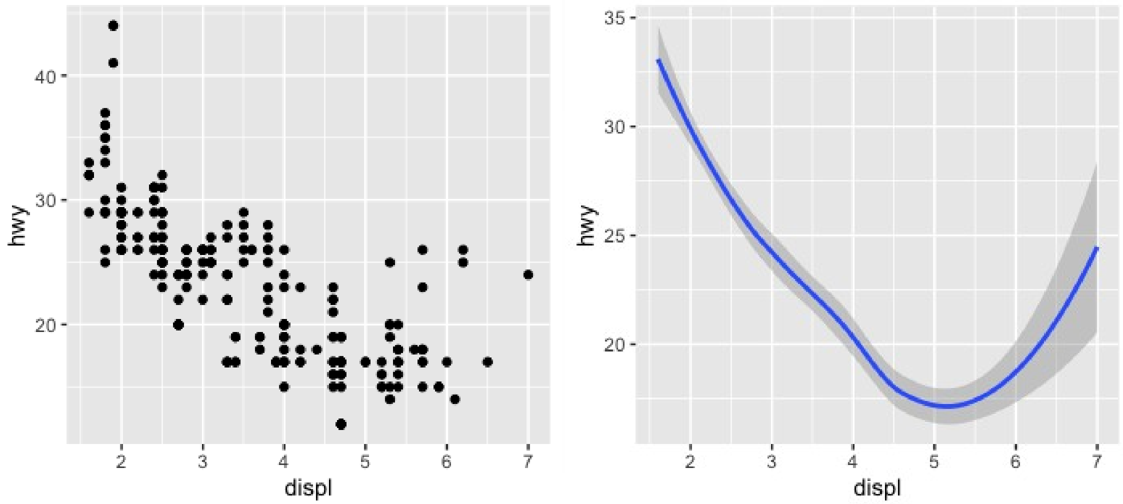
- Geom is a geometrical object that the plot uses to represent the data.
- A scatter plot uses a point geom. Typically, the names are associated –
  - Bar chart uses a bar geom
  - Line chart uses a line geom
  - Boxplots use a boxplot geom
  - There are many others…

This scatterplot can be converted to a smoothed line plot by using *geom_smooth()*.

⎔ Not every *aes()* works with every *geom* - for e.g., you can set the shape of a point but not a line.

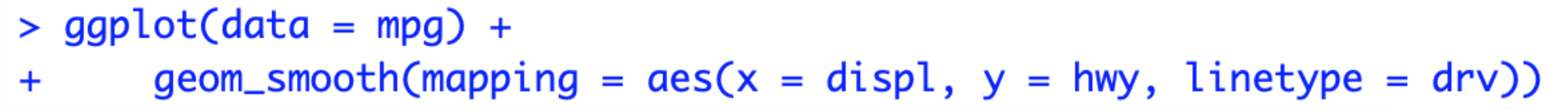


```
> ggplot(data = mpg) +
+     geom_smooth(mapping = aes(x = displ, y = hwy))
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

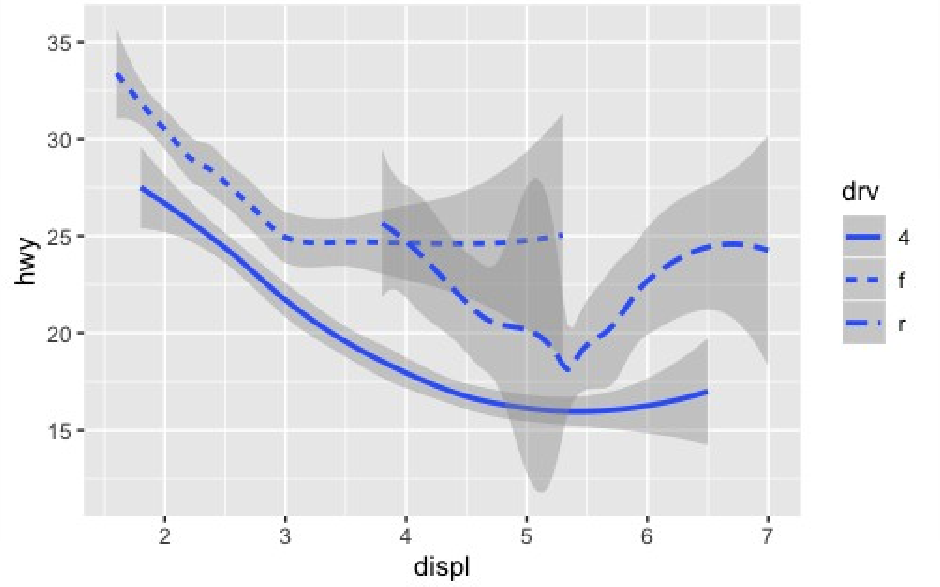


geom_smooth()

- The *linetype* argument within *geom_smooth()* plots a different line with a different linetype for every level of the variable used.
- For e.g., if we want to separate lines by 4wd, front-, or rear-wheel drive.



```
> ggplot(data = mpg) +
+     geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```

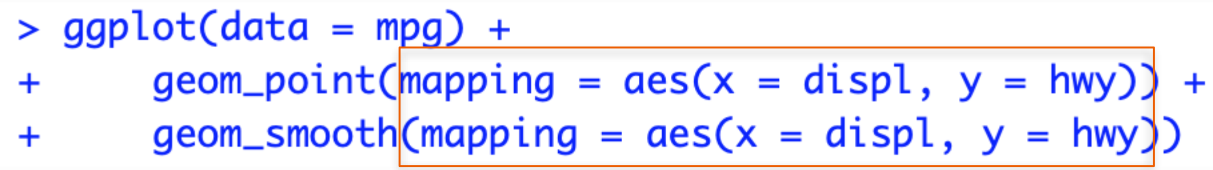


- There are over 40 geoms built-in to ggplot2.
- There are additional geoms available by external packages –https://www.ggplot2-exts.org
- Best way to get help

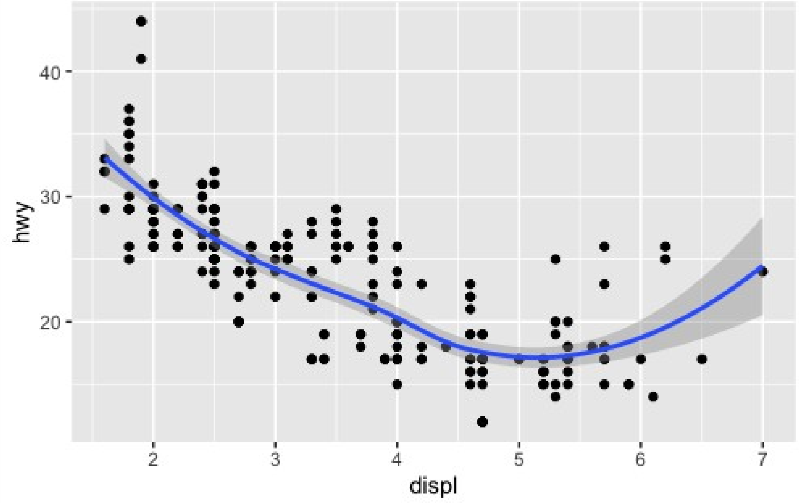
R-Studio > Help > Cheatsheets > Data Visualization with ggplot2.

Multiple *geoms*

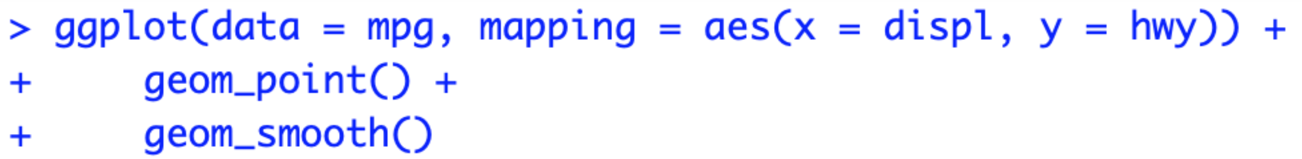
- You can add multiple geoms to the same plot. This is handy when you want to see the actual observation and the trend line over it.



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



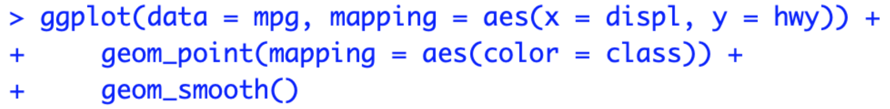
- Do you see any issues with the above code – code duplication!
- We can place the *aes()* mapping outside the *geom* as global aesthetics.



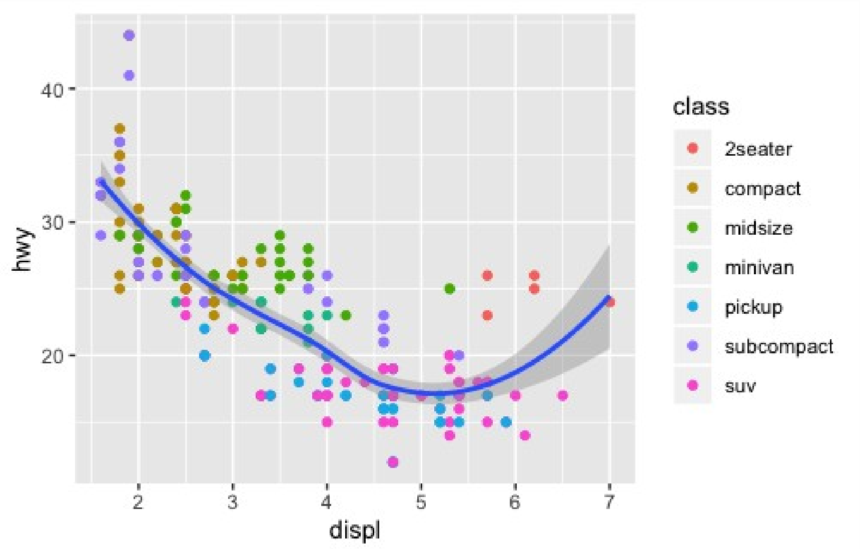
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+     geom_point() +
+     geom_smooth()
```

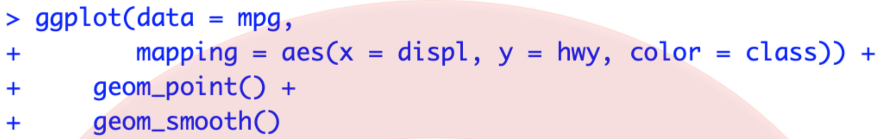
Multiple *geoms* (cont.)

- You can also override the global aesthetic mappings by placing local aesthetic mappings inside *geoms*. That mapping will work on that layer only.
- For e.g. if we want to color the scatter plot by class, but not the lineplot.

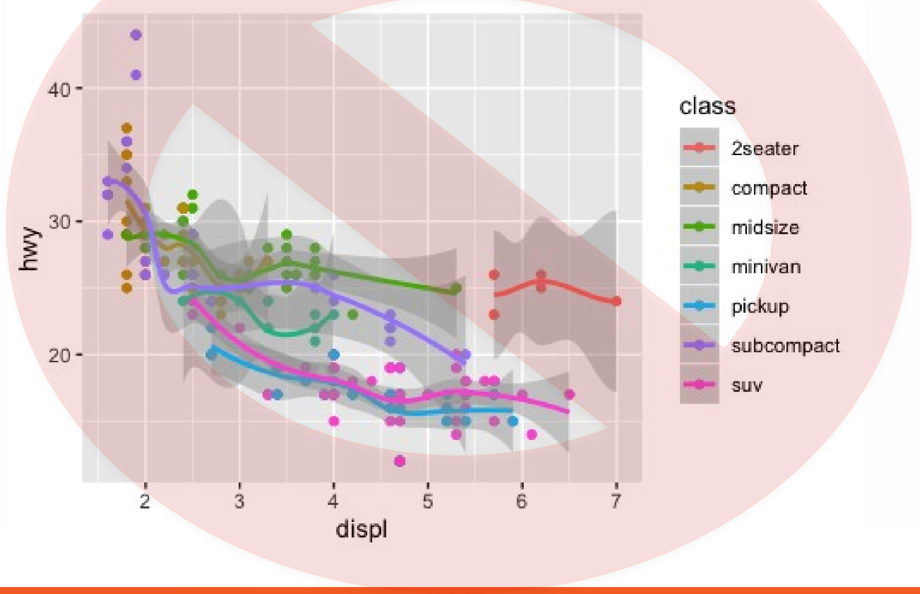


```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+     geom_point(mapping = aes(color = class)) +
+     geom_smooth()
```





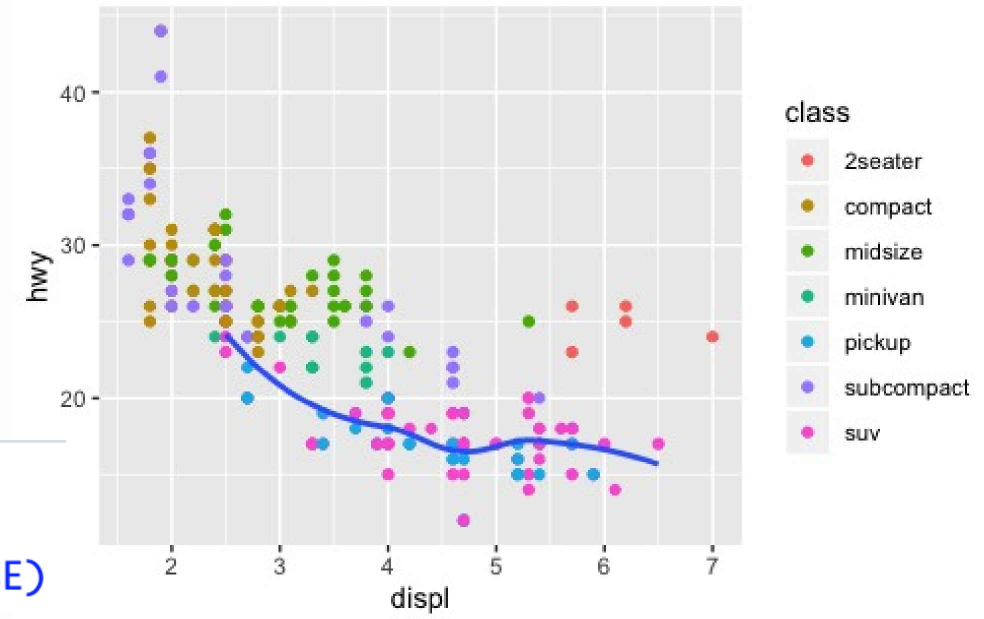
```
> ggplot(data = mpg,
+        mapping = aes(x = displ, y = hwy, color = class)) +
+     geom_point() +
+     geom_smooth()
```



Multiple *geoms* (cont..)

- You can also specify different data sets for different *geoms*.
- For e.g., if you wanted to draw the smoothed line for only SUVs but wanted to see the data points for the entire data.

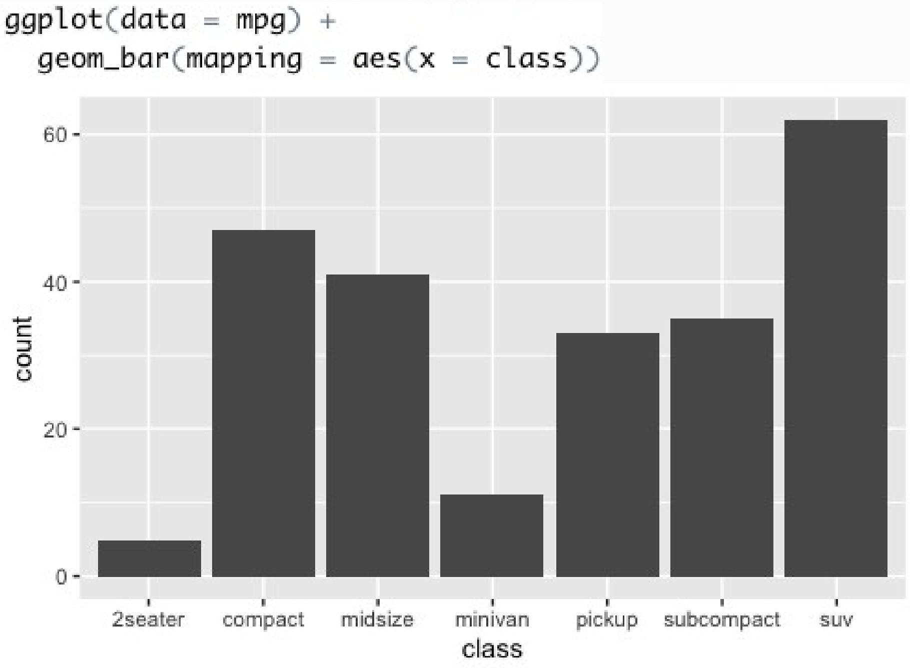
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+     geom_point(mapping = aes(color = class)) +
+     geom_smooth(data = filter(mpg, class == "suv"), se = FALSE)
```



- The local data argument overrides the global mpg data.
  - *filter()* extracts the suv class.
  - *se = FALSE* argument removes the standard error shading.

geom_bar()

- Bar charts are commonly used for representing a distribution of data – *geom_bar()*.
- We want to see the distribution of cars across class.
- Notice that the y-axis is count – this is a new computed value from the raw data.
- *geom_bar()* counts the frequency of each class and uses this on the y-axis.
  - Bar charts, histograms, and frequency plots computes counts and plots bins of counts.
  - Smoothing plots fit a model and then plots fitted values.
  - Boxplots computes summaries and plots the box.

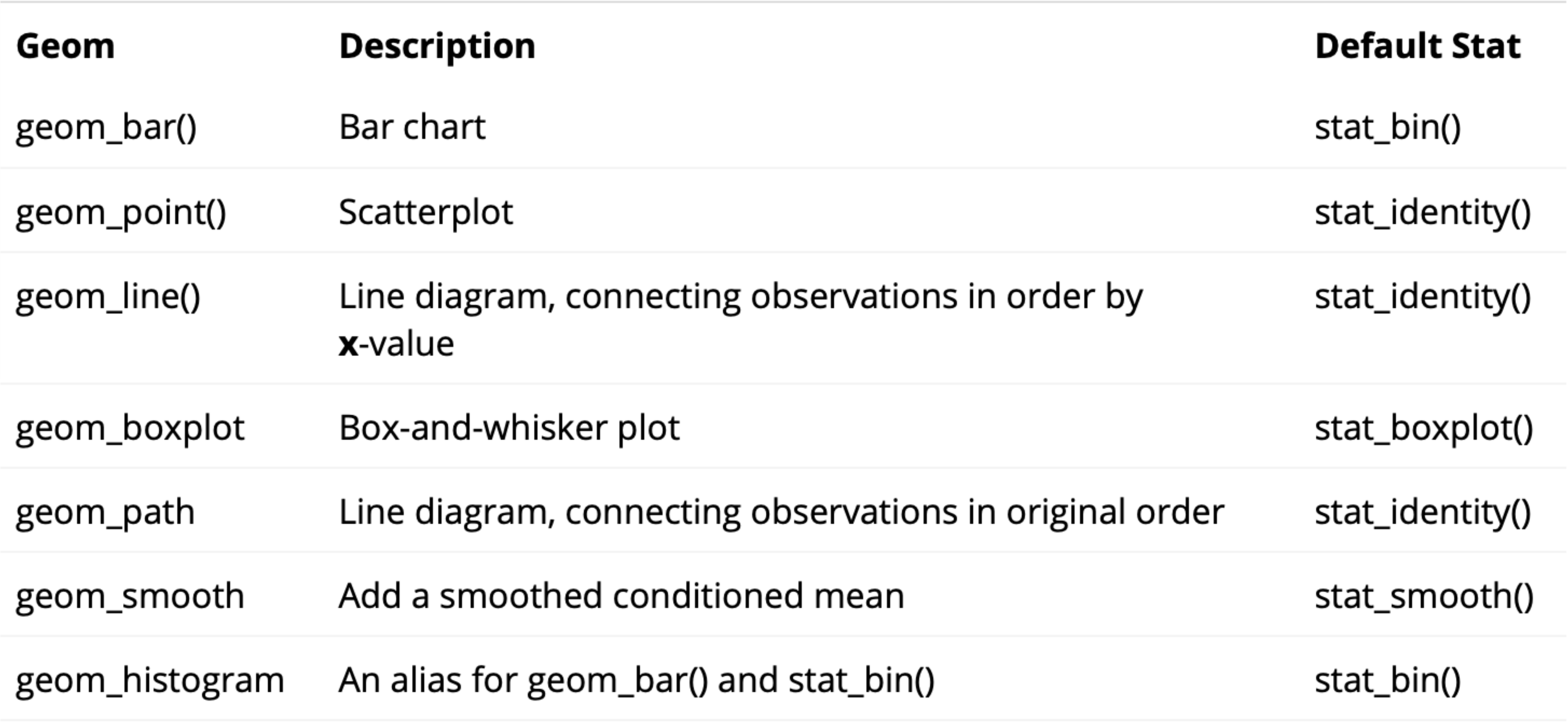


```
ggplot(data = mpg) +
  geom_bar(mapping = aes(x = class))
```

- The default transformations for each of these *geoms* can be found in help under argument *stat*
- For e.g. see *?geom_bar, ?geom_abline*, or ?*geom_boxplot*

Default *geom* and *stat*

Each geometric object can also be made using its stat counterpart.

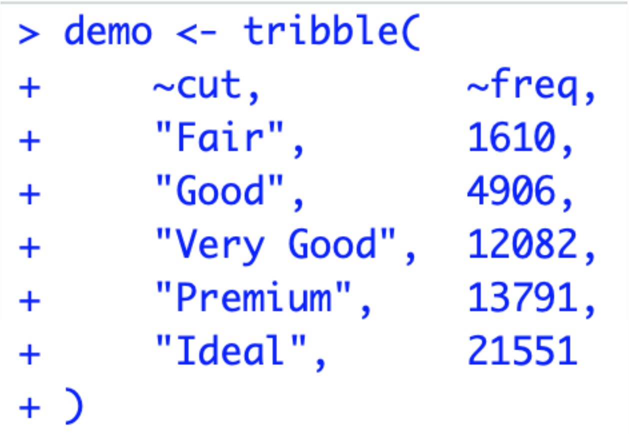


Geom	**Description**	**Default Stat**
geom_bar()	Bar chart	stat_bin()
geom_point()	Scatterplot	stat_identity()
geom_line()	Line diagram, connecting observations in order by **x**-value	stat_identity()
geom_boxplot	Box-and-whisker plot	stat_boxplot()
geom_path	Line diagram, connecting observations in original order	stat_identity()
geom_smooth	Add a smoothed conditioned mean	stat_smooth()
geom_histogram	An alias for geom_bar() and stat_bin()	stat_bin()

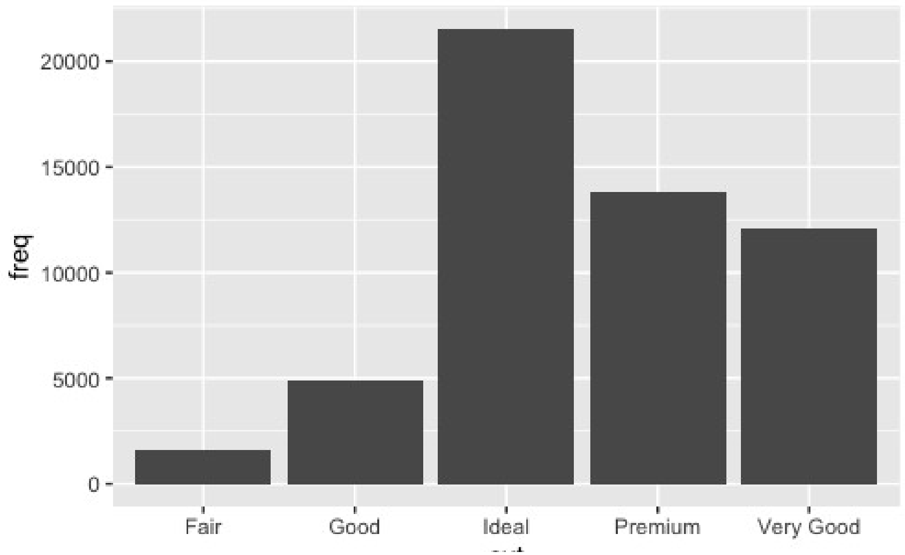
Figure from https://www.dummies.com/programming/r/how-to-get-geoms-in-ggplot-in-r/

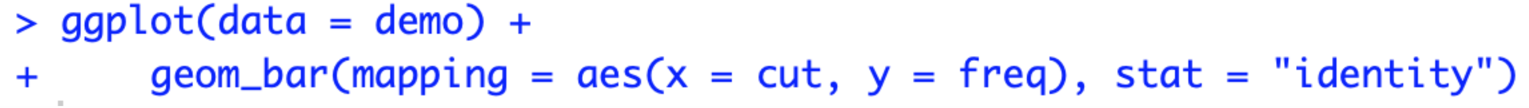
When to Override Default *stat*

- Suppose we wanted to show the proportions on the y-axis.
- For e.g., create a data frame about diamond cuts called *demo*.



```
> demo <- tribble(
+       ~cut,           ~freq,
+       "Fair",         1610,
+       "Good",         4906,
+       "Very Good",    12082,
+       "Premium",      13791,
+       "Ideal",        21551
+ )
```



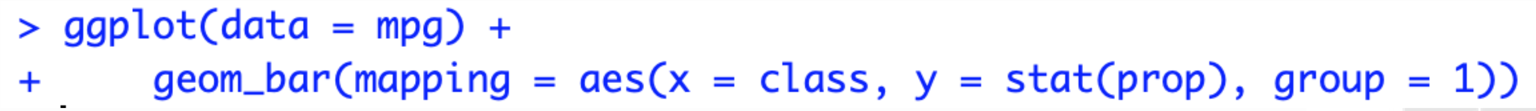


```
> ggplot(data = demo) +
+       geom_bar(mapping = aes(x = cut, y = freq), stat = "identity")
```

- Using stat = "identity" allows me to plot the summarized values directly.

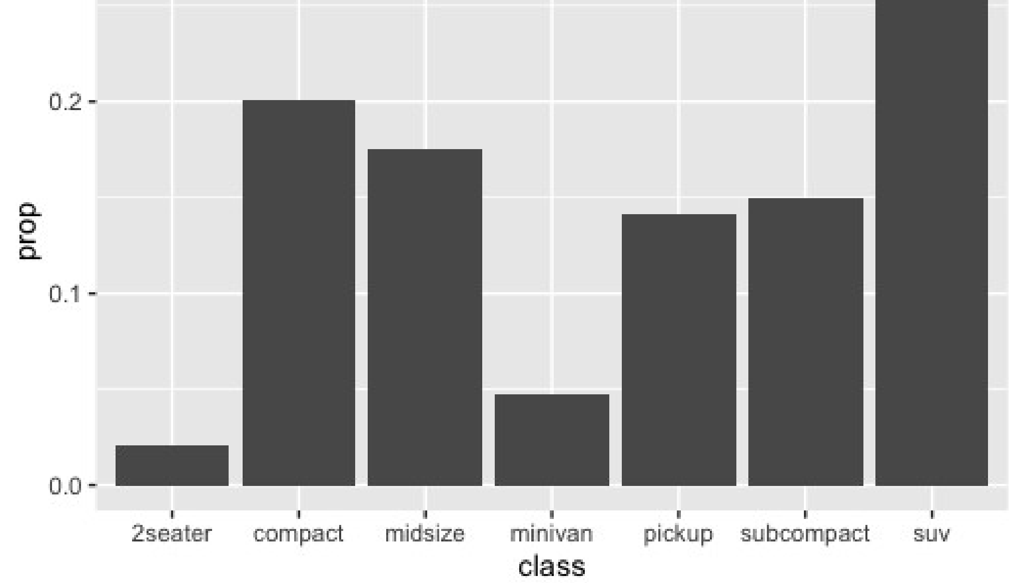
When to Override Default *stat (cont.)*

- Now on the y-axis, if we want to show proportions instead of frequency.
- We can use the *mpg* data – based on class.



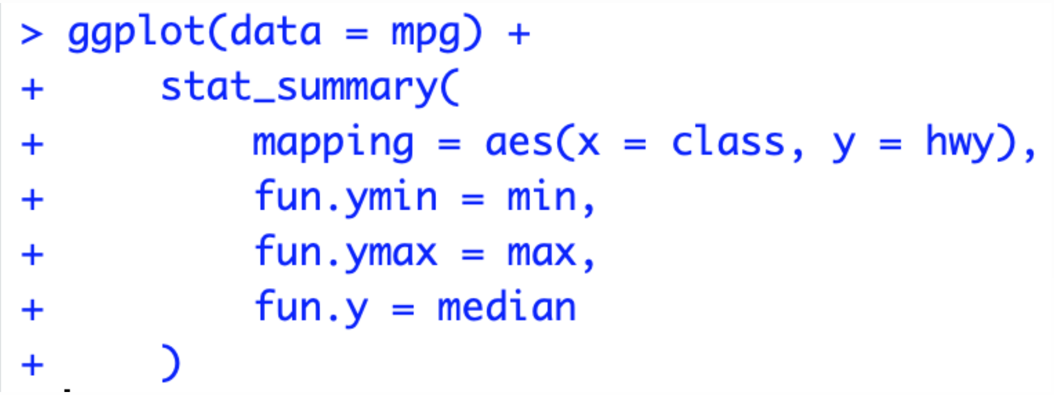
```
> ggplot(data = mpg) +
+     geom_bar(mapping = aes(x = class, y = stat(prop), group = 1))
```

- *y = stat(prop)* plots the proportion on y-axis.
- We tell ggplot to compute proportion by taking all the classes into a single group, using *group = 1.*
  - If you take that out, all classes will have *prop* = *1*.

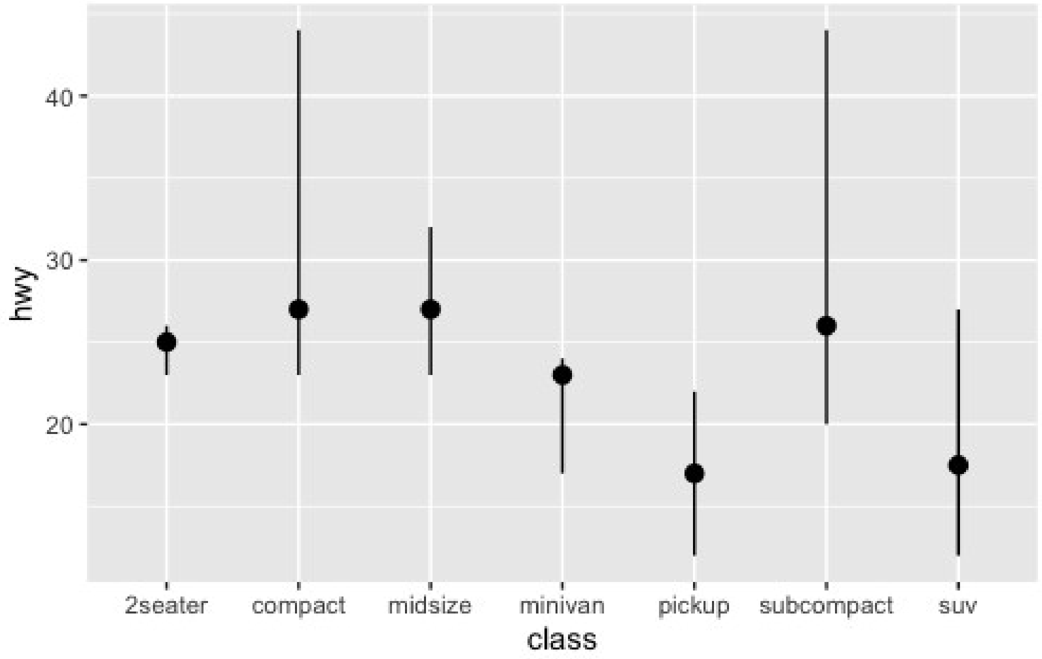


When to Override Default *stat (cont..)*

- Or else, maybe we want to plot summary statistics from the data.
  - Use *stat_summary()* – summarizes *y* values for each *x* value.



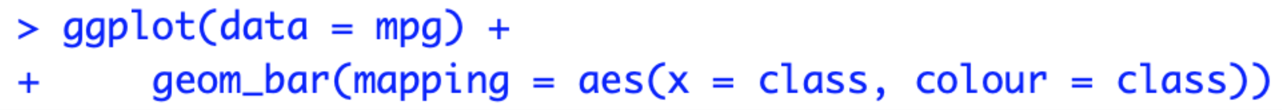
```
> ggplot(data = mpg) +
+     stat_summary(
+         mapping = aes(x = class, y = hwy),
+         fun.ymin = min,
+         fun.ymax = max,
+         fun.y = median
+     )
```



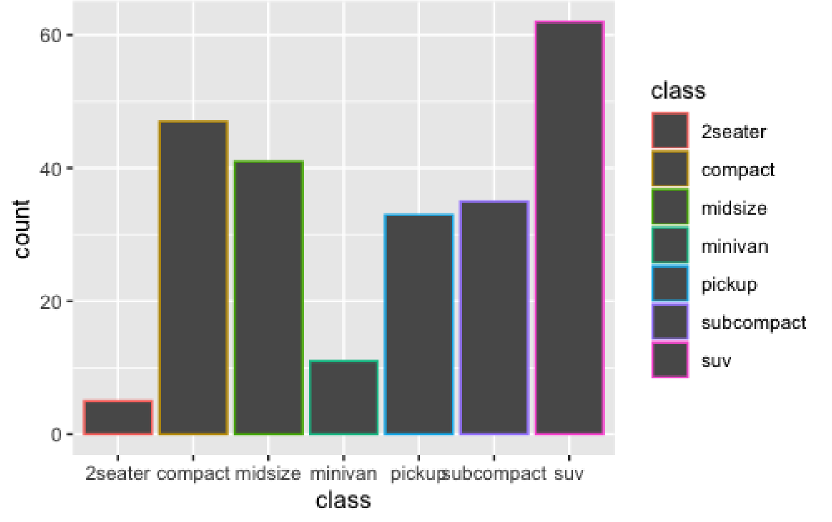
- ggplot() provides over 20 stat functions. You can get help for each using ?, for e.g. ?stat_bin.

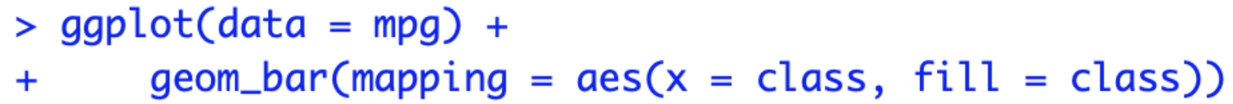
geom_bar() (cont.)

You can also color bar charts using color or fill arguments.

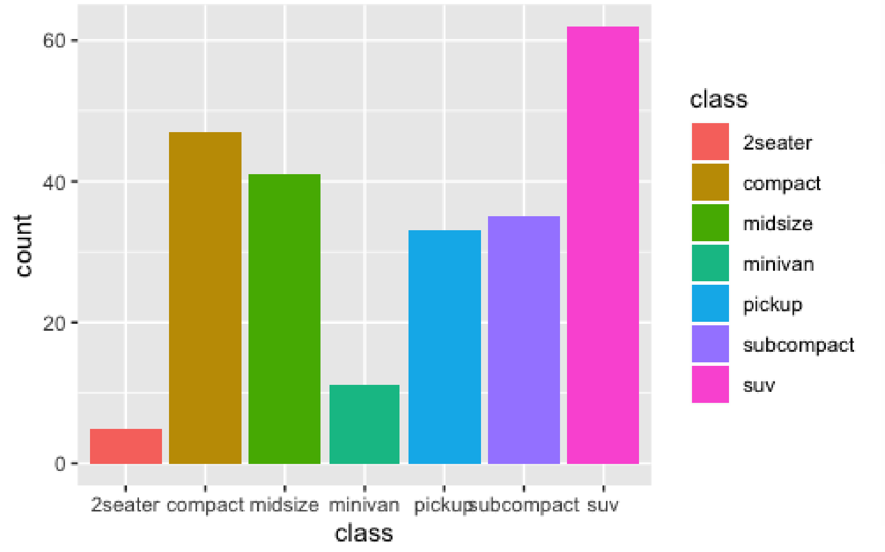


```
> ggplot(data = mpg) +
+     geom_bar(mapping = aes(x = class, colour = class))
```



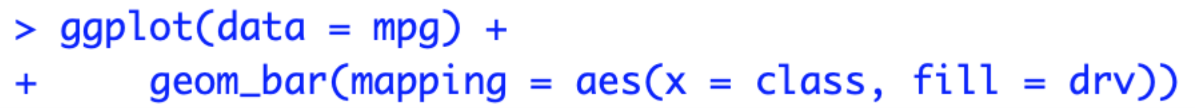


```
> ggplot(data = mpg) +
+     geom_bar(mapping = aes(x = class, fill = class))
```



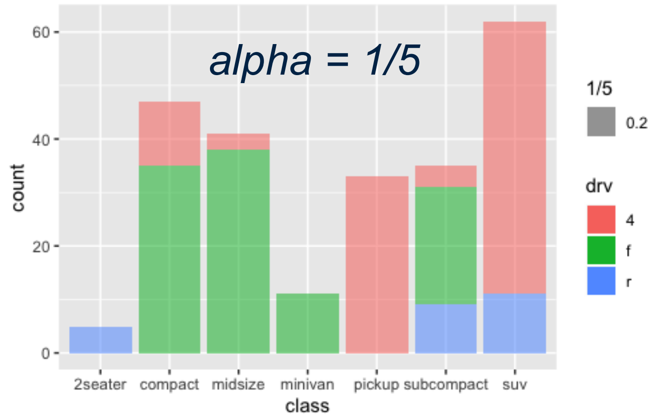
geom_bar() (cont..)

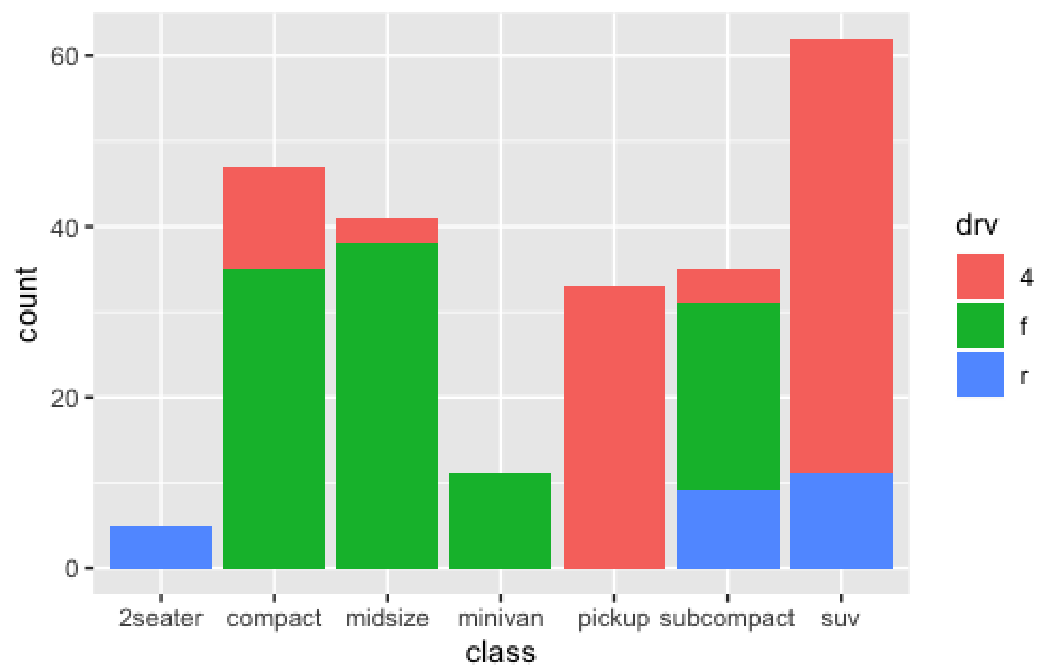
- If you change the fill variable to something other than the x variable, you can see the distribution within each bar.
- For e.g. see the distribution of *drv* within each *class*



```
> ggplot(data = mpg) +
+     geom_bar(mapping = aes(x = class, fill = drv))
```

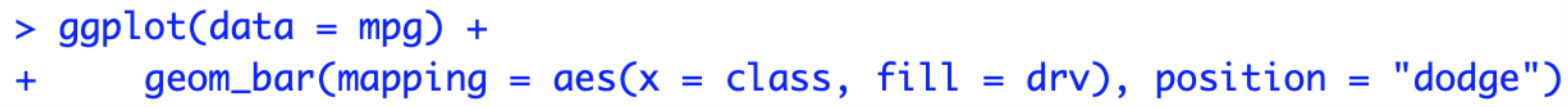
- You can also control the transparency of the bars using *alpha* inside *aes()*.





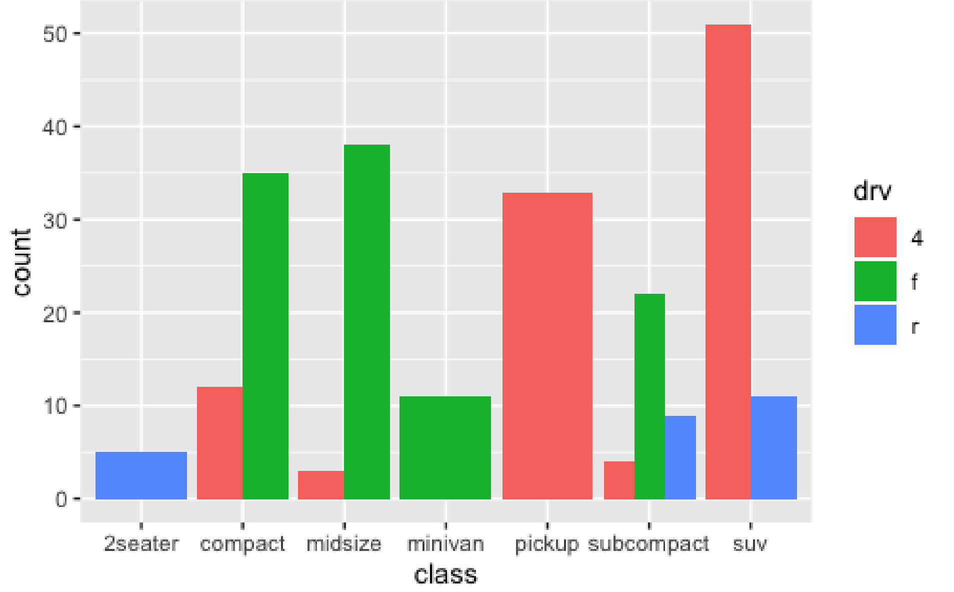
Unstacked Bar Chart

- If you don’t want stacked bar chart, but rather next to each other.
- Use position = "dodge"



```
> ggplot(data = mpg) +
+     geom_bar(mapping = aes(x = class, fill = drv), position = "dodge")
```

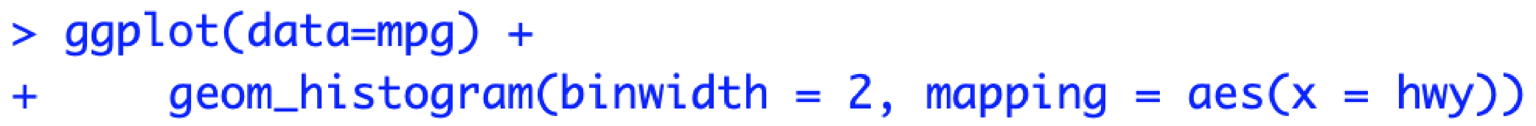
- Note that the position argument is outside *aes()* but inside *geom_bar()*.



geom_histogram()

- Another popular type of plot used in statistical analysis are histograms. They show the distribution of the data.

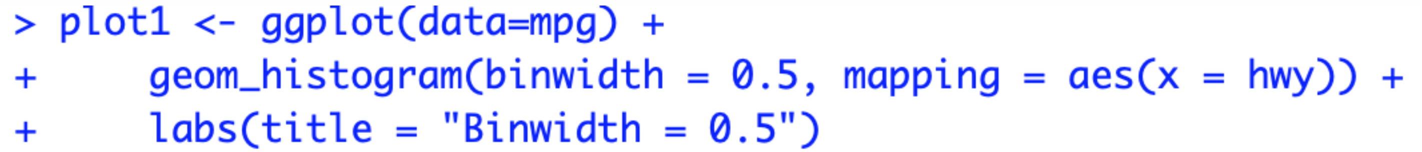
- In *geom_histogram(),* you can specify the *binwidth* to getter a better understanding of the distribution of the data.



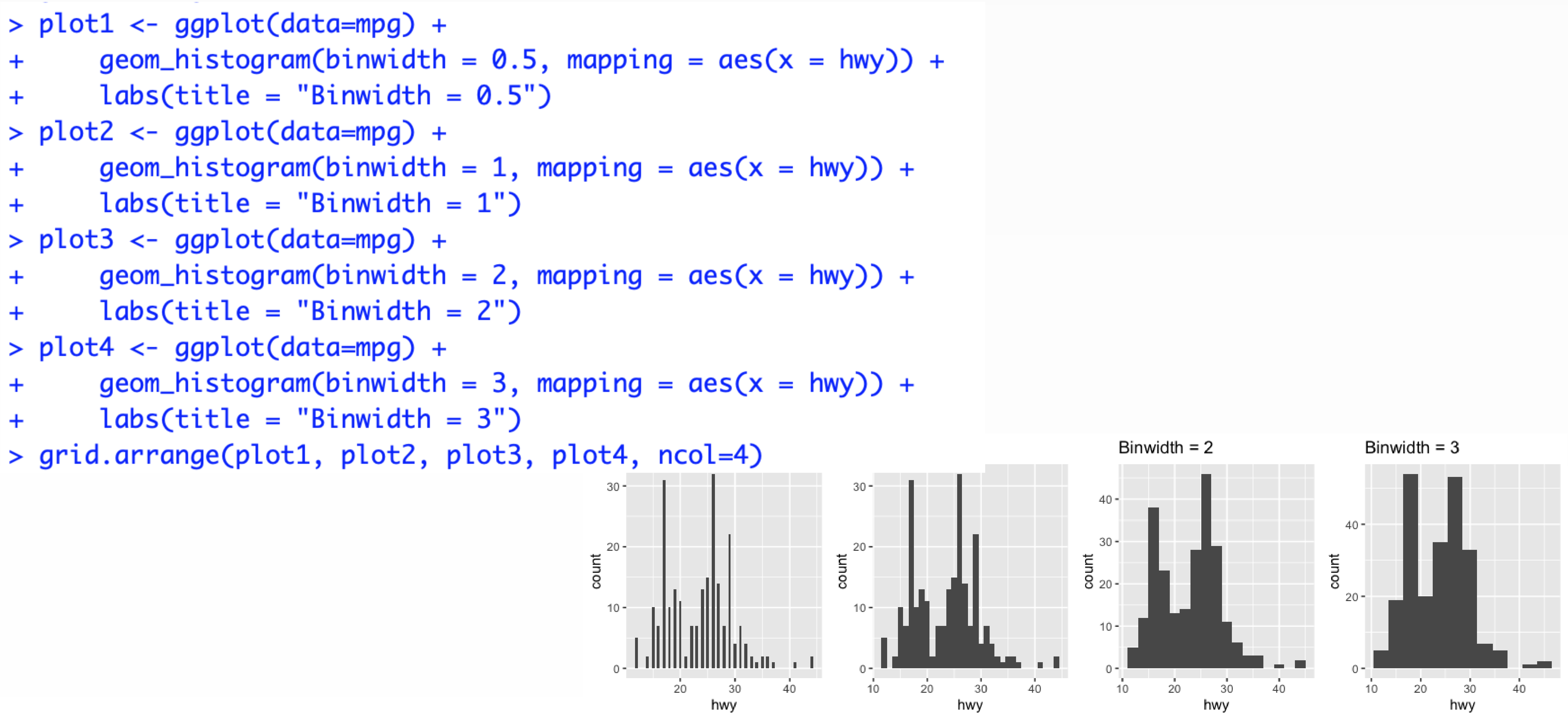
```
> ggplot(data=mpg) +
+     geom_histogram(binwidth = 2, mapping = aes(x = hwy))
```

- Do *install.packages("gridExtra")* and load it - makes multiple plots on a grid easily.

- You can assign plots to objects and plot multiple objects on a grid.



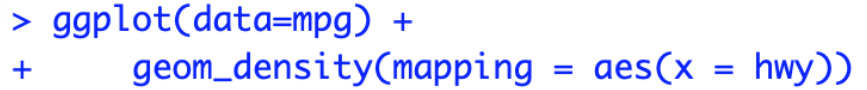
```
> plot1 <- ggplot(data=mpg) +
+     geom_histogram(binwidth = 0.5, mapping = aes(x = hwy)) +
+     labs(title = "Binwidth = 0.5")
```



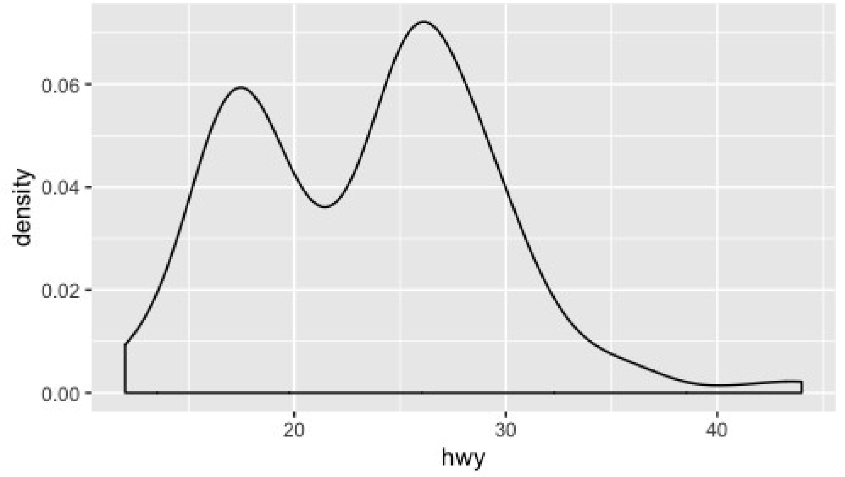
```
> plot1 <- ggplot(data=mpg) +
+     geom_histogram(binwidth = 0.5, mapping = aes(x = hwy)) +
+     labs(title = "Binwidth = 0.5")
> plot2 <- ggplot(data=mpg) +
+     geom_histogram(binwidth = 1, mapping = aes(x = hwy)) +
+     labs(title = "Binwidth = 1")
> plot3 <- ggplot(data=mpg) +
+     geom_histogram(binwidth = 2, mapping = aes(x = hwy)) +
+     labs(title = "Binwidth = 2")
> plot4 <- ggplot(data=mpg) +
+     geom_histogram(binwidth = 3, mapping = aes(x = hwy)) +
+     labs(title = "Binwidth = 3")
> grid.arrange(plot1, plot2, plot3, plot4, ncol=4)
```

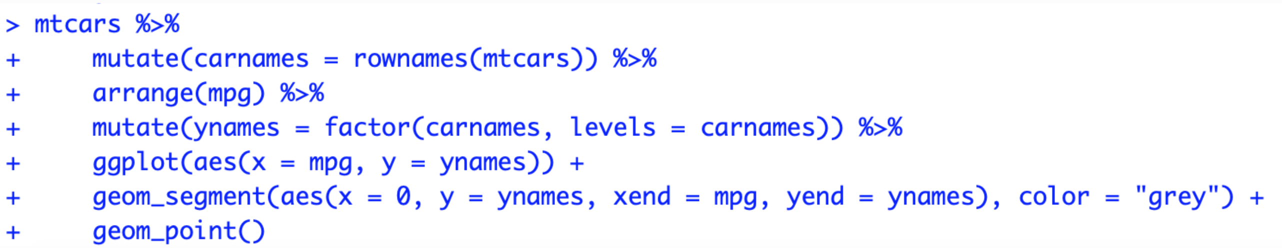
geom_density() and geom_segment()

- A smoothed histogram can be plotted using a density plot.



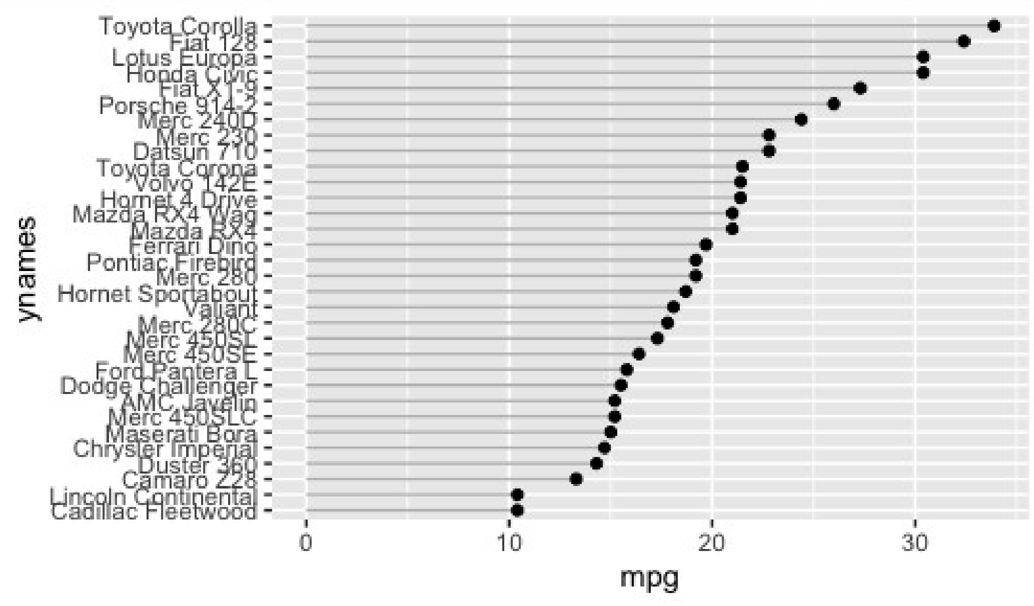
```
> ggplot(data=mpg) +
+     geom_density(mapping = aes(x = hwy))
```





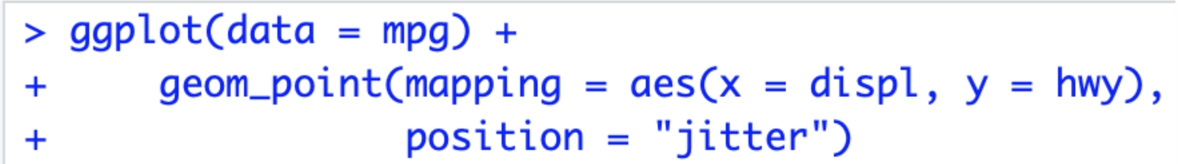
```
> mtcars %>%
+     mutate(carnames = rownames(mtcars)) %>%
+     arrange(mpg) %>%
+     mutate(ynames = factor(carnames, levels = carnames)) %>%
+     ggplot(aes(x = mpg, y = ynames)) +
+     geom_segment(aes(x = 0, y = ynames, xend = mpg, yend = ynames), color = "grey") +
+     geom_point()
```

- Sometimes, we want to see a hybrid between bar chart and scatterplot.
- Plot *mtcars* data sorted based on mpg, and then all the cars plotted along with their values.



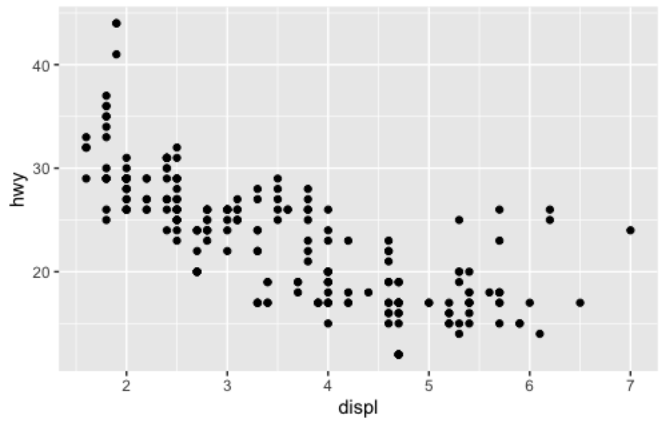
Jittered Scatterplot

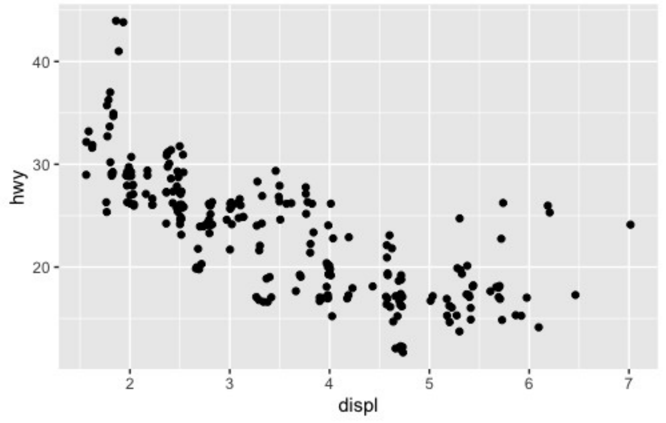
- Often, observations overlap each other on the plot.
  - For e.g., see the first scatterplot we made.
  - This plot shows 126 points even though there are 234 observations.
  - A lot of overlap: problem known as overplotting!
- The argument *position = "jitter"* overcomes this by slightly moving overlapping observations.



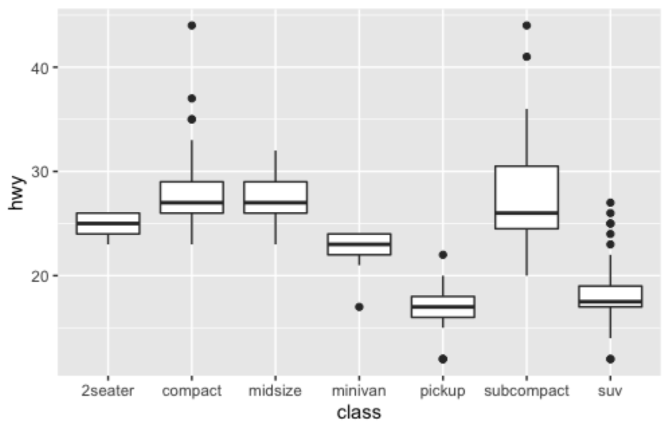
```
> ggplot(data = mpg) +
+     geom_point(mapping = aes(x = displ, y = hwy),
+                position = "jitter")
```

- Less accurate in small scale but reveals better insights about the data in the big picture.





Coordinate System

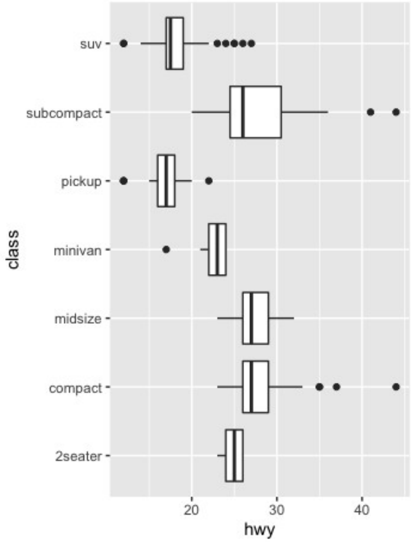


- By default, Cartesian coordinates.
- *coord_flip()* switches the x and y axes.
- Suppose we made a boxplot of highway mileage across the classes.

```
> ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
+       geom_boxplot()
```

- Then, we wanted to switch this into a horizonal boxplot.

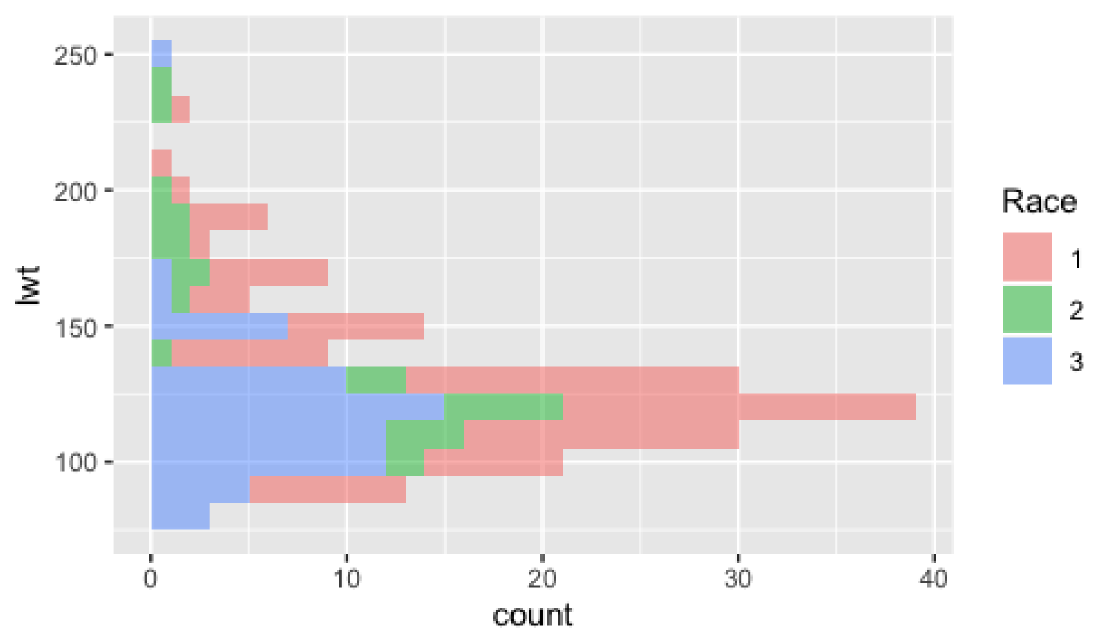
```
> ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
+       geom_boxplot() +
+       coord_flip()
```



Examples

Install the *MASS* package and load the *"birthwt"* data. Do *?birthwt* for more details.

1. Make the plot shown below.
2. Transparency is set of 50%, and histogram uses binwidth of 10.

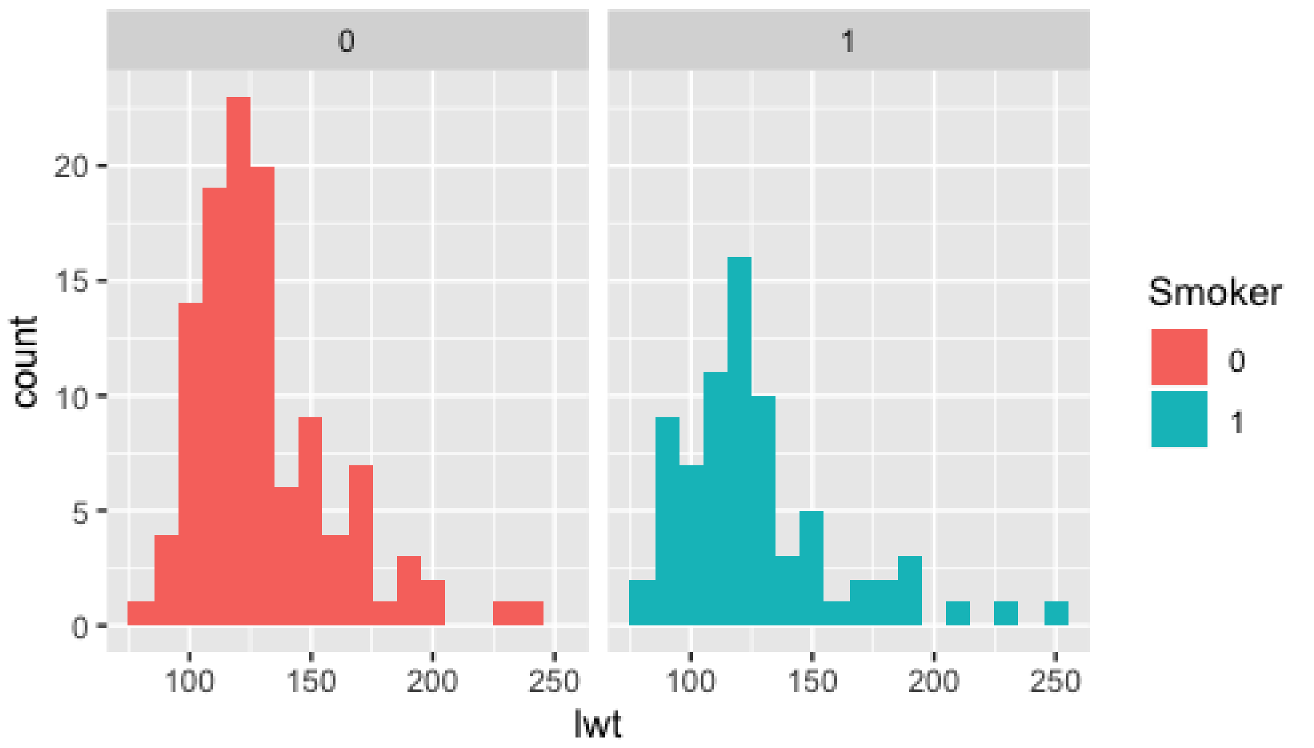


Hints: to change legend title: *scale_fill_discrete(name="Race")*

If you're getting gradient coloring, think about the class of data.

Examples (cont.)

- Reproduce the plot below. Hint – remember *facet_wrap() and use binwidth = 10*!
- Again, make sure the class of variables are correct.


count
lwt
Smoker
🟥 0
🟦 1