Week 8: Basic Introduction to `tidyverse` and `ggplot`

STAT GR5206 *Statistical Computing & Introduction to Data Science*

Linxi Liu
Columbia University

October 26, 2018

Today

"The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs".

`tidyverse`

- `dplyr`: data manipulation
- `ggplot2`: creating advanced graphics
- `readdr`: reading in rectangular data.
- `tibble`: A `tibble`, or `tbl_df`, is a modern reimagining of the `data.frame`.
- `tidyr`: creating tidy data.
- `purrr`: enhancing R's functional programming (FP).

The above statement was taken directly from the `tidyverse` website: https://www.tidyverse.org

Today

We focus on a few packages:

`tidyverse` and a few other topics

- `tibble`: A `tibble`, or `tbl_df`, is a modern reimagining of the `data.frame`
- Review of the Split/Apply/Combine model and the `plyr` package
- `purrr`: enhancing R's functional programming (FP)
- Reshaping data and the `reshape2` package
- `tidyr`: creating tidy data
- `ggplot2`: creating advanced graphics

tibble

tibble package

tibble

- Tibbles are data frames, but they tweak some older behaviours to make life a little easier. (according to tidyverse)
- Convert a traditional dataframe to a `tibble` using `as_tibble()`

```
> library(tibble)
> as_tibble(head(iris,3))
```

```
# A tibble: 3 x 5
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
*        <dbl>       <dbl>        <dbl>       <dbl> <fct>
1          5.10        3.50         1.40       0.200 setosa
2          4.90        3.00         1.40       0.200 setosa
3          4.70        3.20         1.30       0.200 setosa
```

tibble package

tibble

- Create a `tibble` dataframe from scratch using `tibble()`

```
> df <- tibble(
+               x = 1:3,
+               y = sample(seq(1,5,by=2),3),
+               z = rnorm(3)
+               )
```

tibble package

tibble

- Tibbles are data frames, but they tweak some older behaviours to make life a little easier. (according to tidyverse)
- `tibble()` does much less
- `tibble()` never changes the type of the inputs (e.g. it never converts strings to factors!)
- `tibble()` never changes the names of variables
- `tibble()` never creates row names
- It's possible for a tibble to have column names that are not valid R variable names (e.g., ":)" or " ")
- `tribble()` is a transposed tibble

Subsetting a tibble

```
> df$x
```

```
[1] 1 2 3
```

```
> df[["x"]]
```

```
[1] 1 2 3
```

```
> df[,1]
```

```
# A tibble: 3 x 1
      x
  <int>
1     1
2     2
3     3
```

Subsetting a tibble

```
> df[df$x>1,]
```

```
# A tibble: 2 x 3
      x     y      z
  <int> <dbl>  <dbl>
1     2    1. -0.200
2     3    3. -0.821
```

To use these in a pipe (see purrr section), you will need to use the special placeholder . (period)

```
> library(purrr)
> df %>% .$x
```

```
[1] 1 2 3
```

Review of Split/Apply/Combine

Example: Strikes dataset

Dataset on 18 countries over 35 years (compiled by Bruce Western, in the Sociology Department at Harvard University). The measured variables:

- `country`, `year`: country and year of data collection
- `strike.volume`: days on strike per 1000 workers
- `unemployment`: unemployment rate
- `inflation`: inflation rate
- `left.parliament`: leftwing share of the government
- `centralization`: centralization of unions
- `density`: density of unions

Example: Strikes dataset

Our Research Question

Is there a relationship between a country's ruling party alignment (left versus right) and the volume of strikes?

How could we approach this?

- Worst way: by hand, write 18 separate code blocks
- Bad way: explicit `for()` loop, where we loop over countries
- Best way: split appropriately, then use an apply-type function.

Example: Strikes dataset

Since $18 \times 35 = 630$, some years missing from some countries

```
> strikes <- read.csv("strikes.csv", as.is = TRUE)
> dim(strikes)
```

```
[1] 625   8
```

```
> head(strikes, 3)
```

```
    country year strike.volume unemployment inflation
1 Australia 1951           296          1.3      19.8
2 Australia 1952           397          2.2      17.2
3 Australia 1953           360          2.5       4.3
  left.parliament centralization density
1              43      0.3748588      NA
2              43      0.3751829      NA
3              43      0.3745076      NA
```

Strikes Data Set, Revisited

Recall, data set on political economy of strikes:

```
> # Function to compute coefficients from regressing number
> # of strikes (per 1000 workers) on leftwing share of the
> # government
>
> my.strike.lm <- function(country.df) {
+   return(coef(lm(strike.volume ~ left.parliament,
+                  data=country.df)))
+ }
```

Strikes Data Set, Revisited

```
> # Getting regression coefficients separately
> # for each country, old way:
>
> strikes.list  <- split(strikes, f = strikes$country)
> strikes.coefs <- sapply(strikes.list, my.strike.lm)
> strikes.coefs[, 1:12]
```

```
                  Australia    Austria    Belgium    Canada
(Intercept)     414.7712254 423.077279 -56.926780 -227.8218
left.parliament  -0.8638052  -8.210886   8.447463   17.6766
                    Denmark  Finland      France   Germany
(Intercept)     -1399.35735 108.2245 202.4261408 95.657134
left.parliament    34.34477  12.8422  -0.4255319 -1.312305
                   Ireland      Italy     Japan Netherlands
(Intercept)     -94.78661 -738.74531 964.73750  -32.627678
left.parliament  55.46721   40.29109 -24.07595    1.694387
```

Strikes Data Set, Revisited

```
> # Getting regression coefficient separately for each
> # country, new way, in three formats:
> library(plyr)
> strike.coef.a <- daply(strikes, .(country), my.strike.lm)
> # Get back an array, note the difference to sapply()
>
> head(strike.coef.a)
```

```
country     (Intercept) left.parliament
  Australia   414.77123      -0.8638052
  Austria     423.07728      -8.2108864
  Belgium     -56.92678       8.4474627
  Canada     -227.82177      17.6766029
  Denmark   -1399.35735      34.3447662
  Finland     108.22451      12.8422018
```

Strikes Data Set, Revisited

```
> strike.coef.d <- ddply(strikes, .(country), my.strike.lm)
> head(strike.coef.d) # Get back a data frame
```

```
    country (Intercept) left.parliament
1 Australia   414.77123      -0.8638052
2   Austria   423.07728      -8.2108864
3   Belgium   -56.92678       8.4474627
4    Canada  -227.82177      17.6766029
5   Denmark -1399.35735      34.3447662
6   Finland   108.22451      12.8422018
```

Strikes Data Set, Revisited

```
> strike.coef.l <- dlply(strikes, .(country), my.strike.lm)
> head(strike.coef.l, 3) # Get back a list
```

```
$Australia
    (Intercept) left.parliament
    414.7712254      -0.8638052

$Austria
    (Intercept) left.parliament
     423.077279       -8.210886

$Belgium
    (Intercept) left.parliament
     -56.926780        8.447463
```

Section III

purrr

purrr

Pipeline

- The first argument is always the data, so `purrr` works naturally with the pipe.
- The symbol %>% is used to construct the pipeline.
- All purrr functions are type-stable. They always return the advertised output type (`map()` returns lists; `map_dbl()` returns double vectors), or they throw an errror.
- `All map()` functions either accept function, formulas, a character vector, or a numeric vector.

The above bullets were taken directly from the `tidyverse` website: https://www.tidyverse.org

Strikes Data Set, Revisited

Basic purrr pipeline

Write a purrr pipeline that produces equivalent output to the function `my.strike.lm`

```
> # Function to compute coefficients from regressing number
> # of strikes (per 1000 workers) on leftwing share of the
> # government
>
> my.strike.lm <- function(country.df) {
+   return(coef(lm(strike.volume ~ left.parliament,
+                  data=country.df)))
+ }
> # Define Italy dataframe
> Italy.df <-strikes[strikes$country=="Italy",]
```

Strikes Data Set, Revisited

Basic purrr pipeline

```
> # Old way
> my.strike.lm(Italy.df)
```

```
    (Intercept) left.parliament
     -738.74531         40.29109
```

```
> # purrr pipeline
> library(purrr)
> Italy.df%>%
+   lm(strike.volume ~ left.parliament,data=.)%>%
+   coef
```

```
    (Intercept) left.parliament
     -738.74531         40.29109
```

More about `map()`

What is map()?

- The map function transform the input, returning a vector the same length as the input.
- `map()` returns a list or a data frame.
- `map_lgl()`, `map_int()`, `map_dbl()` and `map_chr()` return vectors of the corresponding type; `map_dfr()` and `map_dfc()` return data frames created by row-binding and column-binding respectively.
- They require `dplyr` to be installed.

The above bullets were taken directly from the `tidyverse` website: https://www.tidyverse.org

More about map()

Basic Example

```
> # map() will output a list.
> map(c(1,exp(1),1000),log)
```

```
[[1]]
[1] 0

[[2]]
[1] 1

[[3]]
[1] 6.907755
```

More about map()

Basic Example

```
> # map_dbl() outputs a double precision vector.
> map_dbl(1:3,log)
```

```
[1] 0.0000000 0.6931472 1.0986123
```

```
> # map() is often used in a pipeline.
> 1:3%>%map_dbl(log)
```

```
[1] 0.0000000 0.6931472 1.0986123
```

More about map(): Basic Example

```
> # Write a R funciton
> f <- function(x) {return(log(x)+sin(x))}
> f(c(1,exp(1),1000))
```

```
[1] 0.841471 1.410781 7.734635
```

```
> # Or use vectorized R operations
```

Create a function on the fly with a pipeline

```
> # Use a purrr pipeline
> c(1,exp(1),1000)%>%map_dbl(~ log(.)+sin(.))
```

```
[1] 0.841471 1.410781 7.734635
```

Example: Strikes dataset

Our Research Question

- Is there a relationship between a country's ruling party alignment (left versus right) and the volume of strikes?
- Use a `purrr` pipeline to accomplish this task.

Recall using `dlply()`

```
> strike.coef <- daply(strikes, .(country), my.strike.lm)
> head(strike.coef, 3) # returns an array
```

```
country     (Intercept) left.parliament
  Australia   414.77123      -0.8638052
  Austria     423.07728      -8.2108864
  Belgium     -56.92678       8.4474627
```

Example: Strikes dataset

Our Research Question

- Is there a relationship between a country's ruling party alignment (left versus right) and the volume of strikes?
- Use a `purrr` pipeline to accomplish this

`purrr` pipeline

```
> strike.coef <-
+   strikes%>%
+     split(.$country)%>%
+     map(~ lm(strike.volume ~ left.parliament,data=.))%>%
+     map(coef)
```

Example: Strikes dataset

Output as a list

```
> head(strike.coef,3) # returns a list
```

```
$Australia
    (Intercept) left.parliament
    414.7712254      -0.8638052

$Austria
    (Intercept) left.parliament
     423.077279       -8.210886

$Belgium
    (Intercept) left.parliament
     -56.926780        8.447463
```

Example: Strikes dataset

Our Research Question

- Is there a relationship between a country's ruling party alignment (left versus right) and the volume of strikes?
- Use a purrr pipeline to accomplish this

purrr pipeline

```
> library(dplyr)
> # the map_dfr() depends on the dplyr package
> strike.coef <-
+   strikes%>%
+     split(.$country)%>%
+       map(~ lm(strike.volume ~ left.parliament,data=.))%>%
+       map_dfc(coef)
```

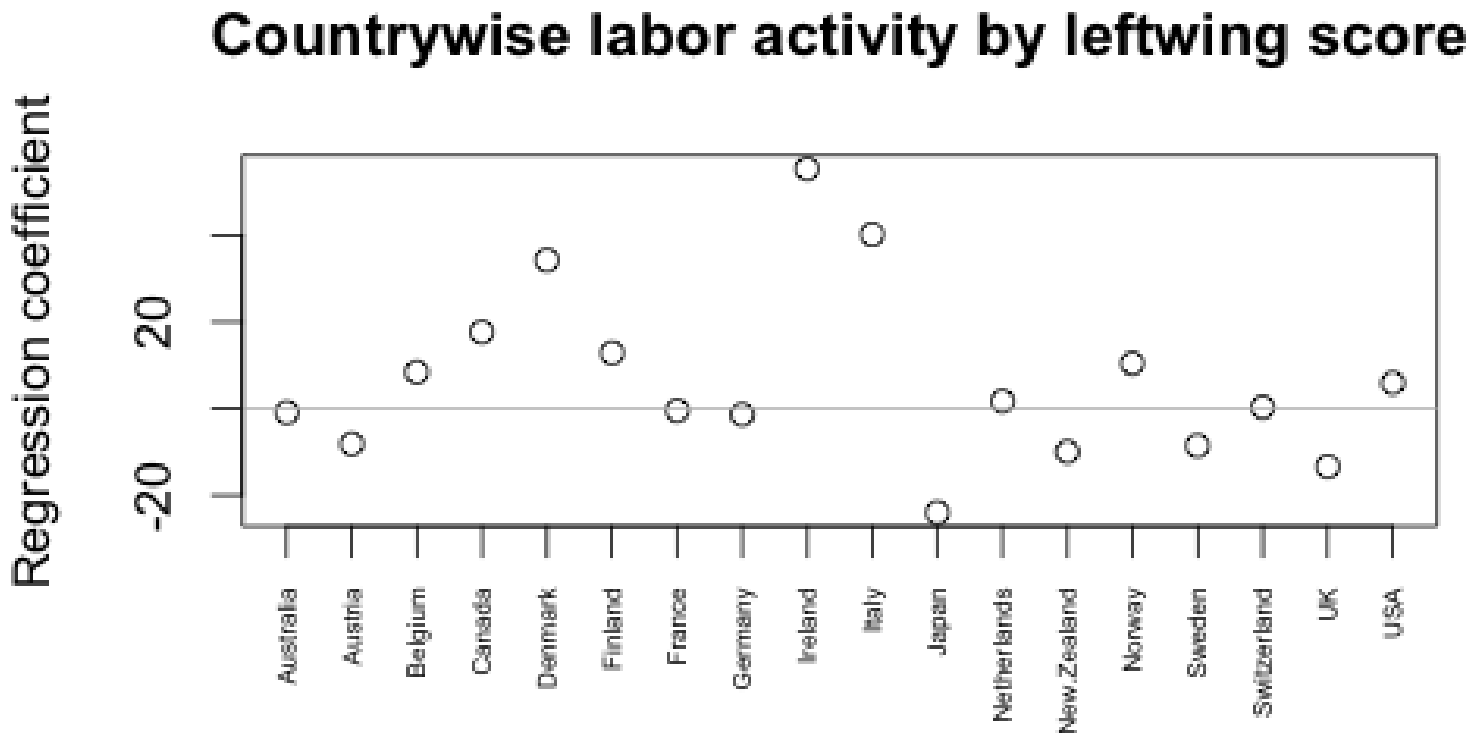
Example: Strikes dataset

Output as a dataframe (tibble class dataframe)

```
> strike.coef[,1:5] # returns a dataframe (tibble)
```

```
# A tibble: 2 x 5
  Australia Austria Belgium Canada Denmark
      <dbl>   <dbl>   <dbl>  <dbl>   <dbl>
1   415.      423.    -56.9  -228.  -1399.
2    -0.864   -8.21     8.45   17.7    34.3
```

Example: Strikes dataset



Example: Strikes dataset

Our Research Question

- Consider the following example taken directly from the `tidyverse` website: https://www.tidyverse.org
- The following example use`spurrr` to solve a fairly realistic problem: split a data frame into pieces, fit a model to each piece, compute the summary, then extract the R^2.

```
> mtcars %>%
+   split(.$cyl) %>% # from base R
+   map(~ lm(mpg ~ wt, data = .)) %>%
+   map(summary) %>%
+   map_dbl("r.squared")
```

```
        4         6         8
0.5086326 0.4645102 0.4229655
```

Reshaping Dataframes

Reshaping

Common to have data where some variables identify units, and others are measurements.

- **Wide** form: columns for ID variables plus 1 column per measurement.
    - Good for things like correlating measurements, or running regressions.
- **Narrow** form: columns for ID variables, plus 1 column identifying measurement, plus 1 column giving value.
    - Good for summarizing, subsetting.

Reshaping

Common to have data where some variables identify units, and others are measurements.

- **Wide** form: columns for ID variables plus 1 column per measurement.
    - Good for things like correlating measurements, or running regressions.
- **Narrow** form: columns for ID variables, plus 1 column identifying measurement, plus 1 column giving value.
    - Good for summarizing, subsetting.

Often want to convert from wide to narrow, or change what's ID and what's measure

Reshaping

- `reshape` package introduced data-reshaping tools.
- `reshape2` package simplifies lots of common uses.
- `melt()` turns a wide dataframe into a narrow one.
- `dcast()` turns a narrow dataframe into a wide one.
- `acast()` turns a narrow dataframe into a wide array.

Reshaping

Example [1]

`snoqualmie.csv` has precipitation every day in Snoqualmie, WA for 36 years (1948–1983). One row per year, one column per day, units of 1/100 inch.

```
> snoq <- read.csv("snoqualmie.csv", header = FALSE,
+                  as.is = TRUE)
> colnames(snoq) <- 1:366
> snoq$year      <- 1948:1983
> snoq[1:3, 360:367]
```

```
  360 361 362 363 364 365 366 year
1   0   0   0   0  49 114  17 1948
2  47 245 121  72  27  41  NA 1949
3   4  40  10   5  93  23  NA 1950
```

[a] From P. Guttorp, Stochastic Modeling of Scientific Data

Reshaping

Example

```
> #install.packages("reshape2")
> require(reshape2)
> snoq.melt <- melt(snoq, id.vars = "year",
+                   variable.name = "day",
+                   value.name = "precip")
> head(snoq.melt)
```

```
  year day precip
1 1948   1    136
2 1949   1     17
3 1950   1      1
4 1951   1     34
5 1952   1      0
6 1953   1      2
```

Reshaping

Example

```
> tail(snoq.melt)
```

```
      year day precip
13171 1978 366     NA
13172 1979 366     NA
13173 1980 366     80
13174 1981 366     NA
13175 1982 366     NA
13176 1983 366     NA
```

```
> dim(snoq.melt) # 36*366
```

```
[1] 13176     3
```

Reshaping

Example

Being sorted by day of the year and then by year is a bit odd

```
>  snoq.melt.chron <- snoq.melt[order(snoq.melt$year), ]
>  head(snoq.melt.chron)
```

```
    year day precip
1   1948   1    136
37  1948   2    100
73  1948   3     16
109 1948   4     80
145 1948   5     10
181 1948   6     66
```

Reshaping

Example

Most years have 365 days so some missing values:

```
> leap.days <- snoq.melt.chron$day == 366
> sum(is.na(snoq.melt.chron$precip[leap.days]))
```

```
[1] 27
```

Reshaping

Example

Most years have 365 days so some missing values:

```
> leap.days <- snoq.melt.chron$day == 366
> sum(is.na(snoq.melt.chron$precip[leap.days]))
```

```
[1] 27
```

Tidy with `na.omit()`:

```
> snoq.melt.chron <- na.omit(snoq.melt.chron)
```

Reshaping

Example

Today's precipitation vs. next day's:

```
> short.chron <- snoq.melt.chron[-nrow(snoq.melt.chron), ]
> precip.next <- snoq.melt.chron$precip[-1]
> snoq.pairs  <- data.frame(short.chron, precip.next)
> head(snoq.pairs)
```

```
    year day precip precip.next
1   1948   1    136          100
37  1948   2    100           16
73  1948   3     16           80
109 1948   4     80           10
145 1948   5     10           66
181 1948   6     66           88
```

Reshaping

Example

```
> snoq[1:3, 360:367]
```

```
  360 361 362 363 364 365 366 year
1   0   0   0   0  49 114  17 1948
2  47 245 121  72  27  41  NA 1949
3   4  40  10   5  93  23  NA 1950
```

`acast()` casts into an array rather than a dataframe.

Reshaping

Example

- The formula could also specify multiple ID variables (including original measure variables), different measure variables (including original ID variables)...
- Also possible to apply functions to aggregates which all have the same IDs, select subsets of the data, etc.
- Recommended reading:
  - Hadley Wickham, "Reshaping Data with the reshape Package", *Journal of Statistical* Software 21 (2007): 12, `http://www.jstatsoft.org/v21/i12`

Section IV

tidyr

Reshaping

The `tidyr` package

`tidyr` is the latest package related to reshaping data.

`tidyr` and related packages

`tidyr`	gather	spread
`reshape(2)`	melt	cast
`spreadsheets`	unpivot	pivot
`databases`	fold	unfold

Section V

ggplot

BaseR Graphics

We study advanced visualization techniques using the `mpg` dataset. Let's try to answer the question: do cars with bigger engines use more fuel than cars with small engines?

BaseR Graphics

We study advanced visualization techniques using the `mpg` dataset. Let's try to answer the question: do cars with bigger engines use more fuel than cars with small engines?

Read about the data using `?mpg`.

```
> dim(mpg)
```

```
[1] 234  11
```

```
> head(mpg,3)
```

```
# A tibble: 3 x 11
  manufacturer model displ  year   cyl trans     drv      cty
  <chr>        <chr> <dbl> <int> <int> <chr>     <chr> <int>
1 audi         a4      1.80  1999     4 auto(l5)  f        18
2 audi         a4      1.80  1999     4 manual(âĂę f        21
3 audi         a4      2.00  2008     4 manual(âĂę f        20
```

Basics of Plotting

Recall,

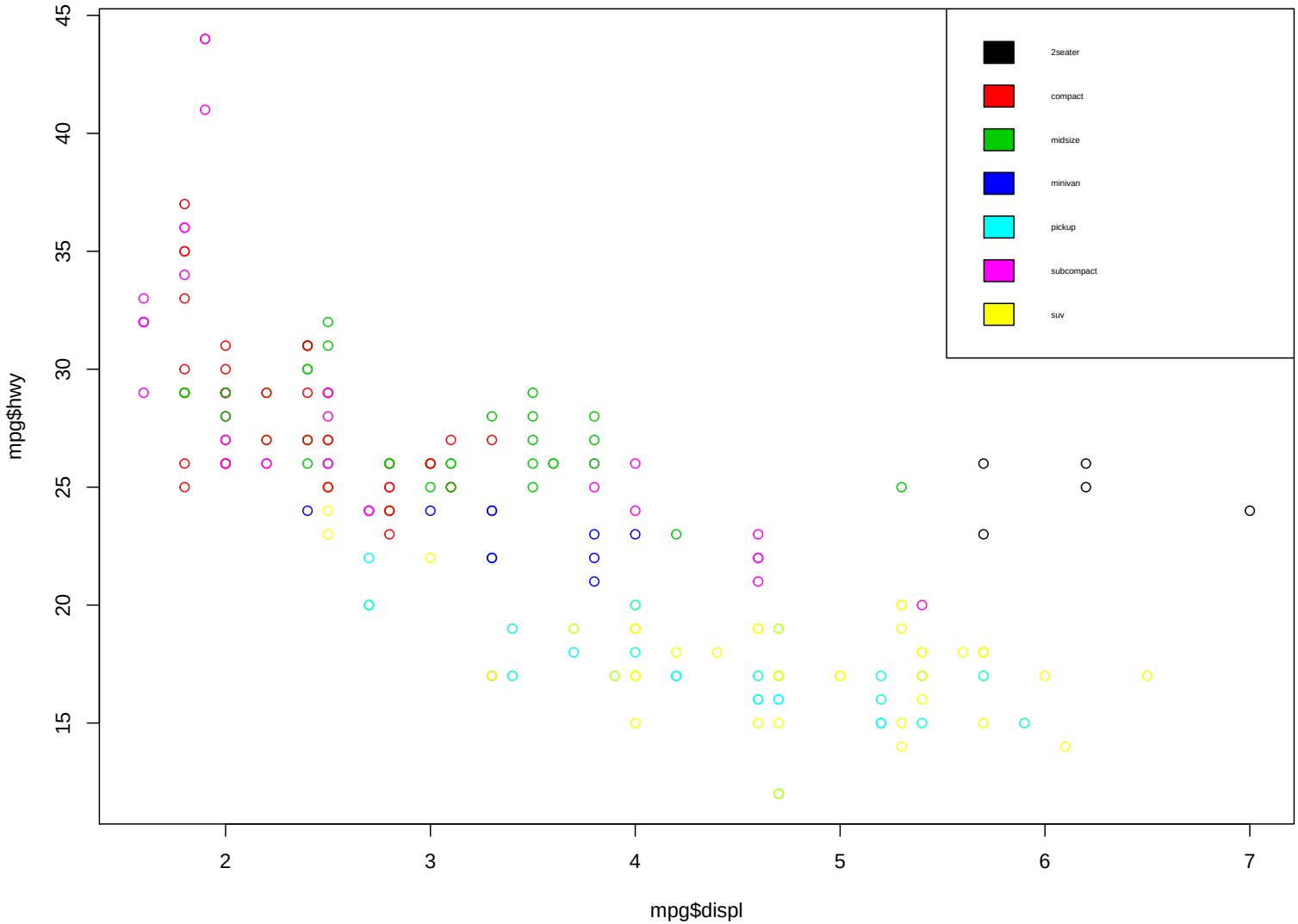
- Visualization variation (of a single variable):
  - `hist()` – Histograms.
  - `barplot()` – Bargraphs.
- Visualizing covariation (of multiple variables):
  - `plot()` – Scatterplots.
  - `boxplot()` – Boxplots (box-and-whisker plots).

Building a Visualization: An Example

Base R Code

```
> plot(mpg$displ,mpg$hwy,col = factor(mpg$class))
> legend("topright",
+        legend = levels(factor(mpg$class)),
+        fill = 1:length(levels(factor(mpg$class))),
+        cex = 1)
```

Building a Visualization: An Example



ggplot2

- R has several systems for making graphs (we've looked at the base R functions).
- ggplot2 is one of the most elegant and flexible.
- ggplot2 uses a coherent system (or 'grammar') for describing and building graphs.

ggplot2

- R has several systems for making graphs (we've looked at the base R functions).
- ggplot2 is one of the most elegant and flexible.
- ggplot2 uses a coherent system (or 'grammar') for describing and building graphs.

Need to run `install.packages("ggplot2")` now and `library("ggplot2")` every time you want to use it!

ggplot2

Let's try to answer the question: do cars with bigger engines use more fuel than cars with small engines?

Read about the data using ?mpg.

```
> dim(mpg)
```

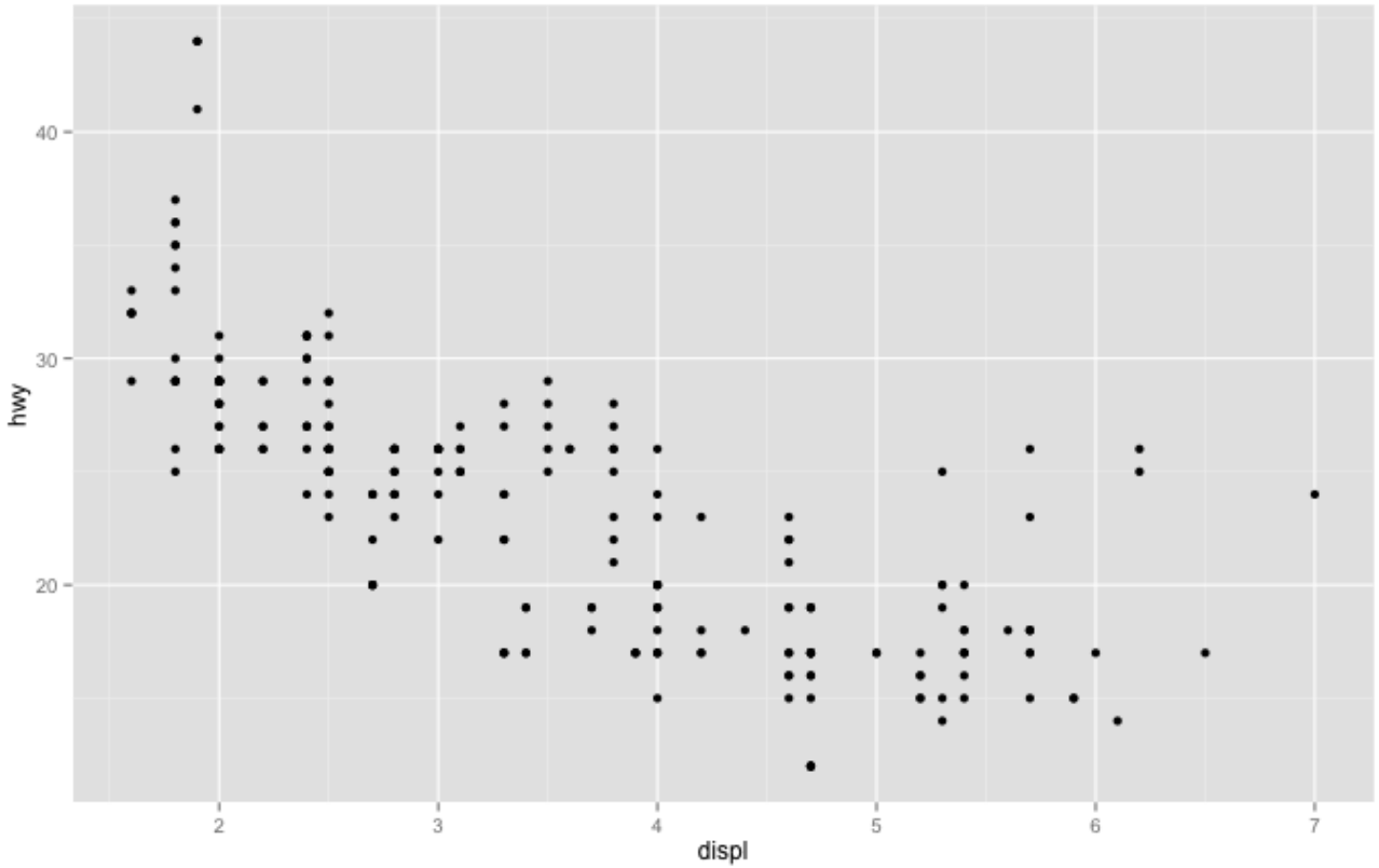
```
[1] 234  11
```

```
> head(mpg, 3)
```

```
# A tibble: 3 x 11
  manufacturer model displ  year   cyl trans    drv      cty
  <chr>        <chr> <dbl> <int> <int> <chr>    <chr> <int>
1 audi         a4      1.80  1999     4 auto(l5) f        18
2 audi         a4      1.80  1999     4 manual(âĂę f        21
3 audi         a4      2.00  2008     4 manual(âĂę f        20
# ... with 3 more variables: hwy <int>, fl <chr>,
```

A First Plot

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



A First Plot

Let's break apart the code:

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

A First Plot

Let's break apart the code:

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

- Begin a plot with `ggplot()`.
  - It creates the coordinate axis that you add to.
  - The first argument is the dataset

A First Plot

Let's break apart the code:

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

- Begin a plot with `ggplot()`.
  - It creates the coordinate axis that you add to.
  - The first argument is the dataset
- Next you want to add layers to the plot.
  - In our example: `geom_point()` adds a layer of points.
  - Lots of different `geom` functions doing different things.

A First Plot

Let's break apart the code:

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

- Begin a plot with `ggplot()`.
  - It creates the coordinate axis that you add to.
  - The first argument is the dataset
- Next you want to add layers to the plot.
  - In our example: `geom_point()` adds a layer of points.
  - Lots of different `geom` functions doing different things.
- `geom` functions take `mapping` arguments.
  - Defines how variables in your dataset are mapped to visual properties.
  - Always paired with `aes()`.
  - The `x` and `y` arguments specify which variables to map to the axes.

A First Plot

General structure:

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

To create a plot, replace the bracketed sections in the code above with a datatset, a `geom` function, and a set of mappings.

From this template, we can make many different kinds of graphs using `ggplot`.

Check Yourself

Tasks

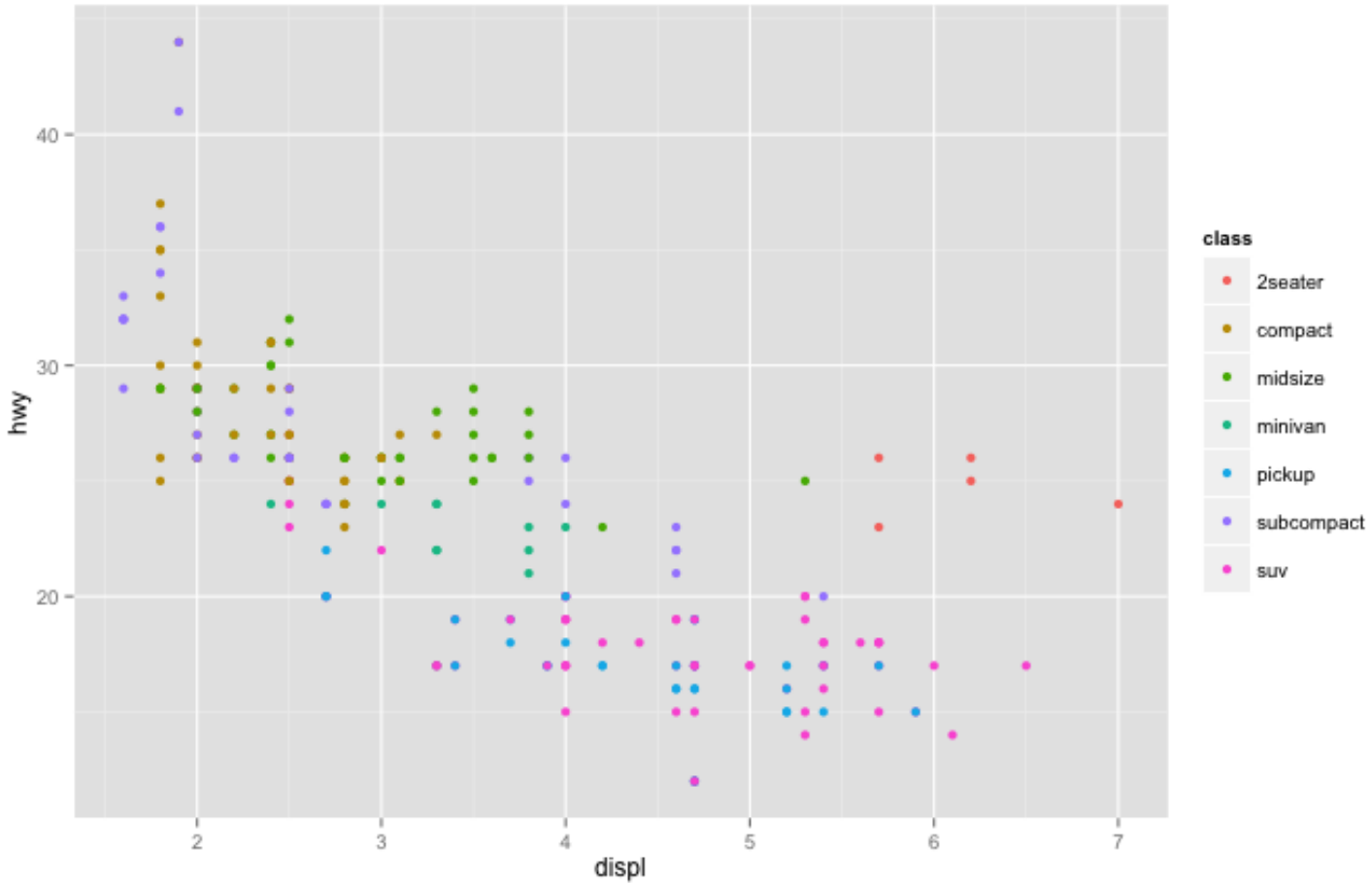
- Plot just `ggplot(data = mpg)`. What do you get?
- Make a scatterplot of `hwy` vs. `cyl`.
- Make a scatterplot of `class` vs. `drv`. Why is this plot not useful?

Aesthetic Mappings

- We can add a third variable to a scatterplot by mapping it to an **aesthetic**.
- An **aesthetic** is a visual property of the objects in the plot.
- Things like size, color, shape of points.

Mapping Aesthetics

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



Check Yourself

Tasks

- Instead of mapping `class` to the `color` aesthetic, map it to the `alpha` aesthetic or the `size` aesthetic.
- Instead of mapping `class` to the `color` aesthetic, map it to the `shape` aesthetic. Note that `ggplot()` will only use 6 shapes at a time. What does this mean for our plot?
- What does the following code do?

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

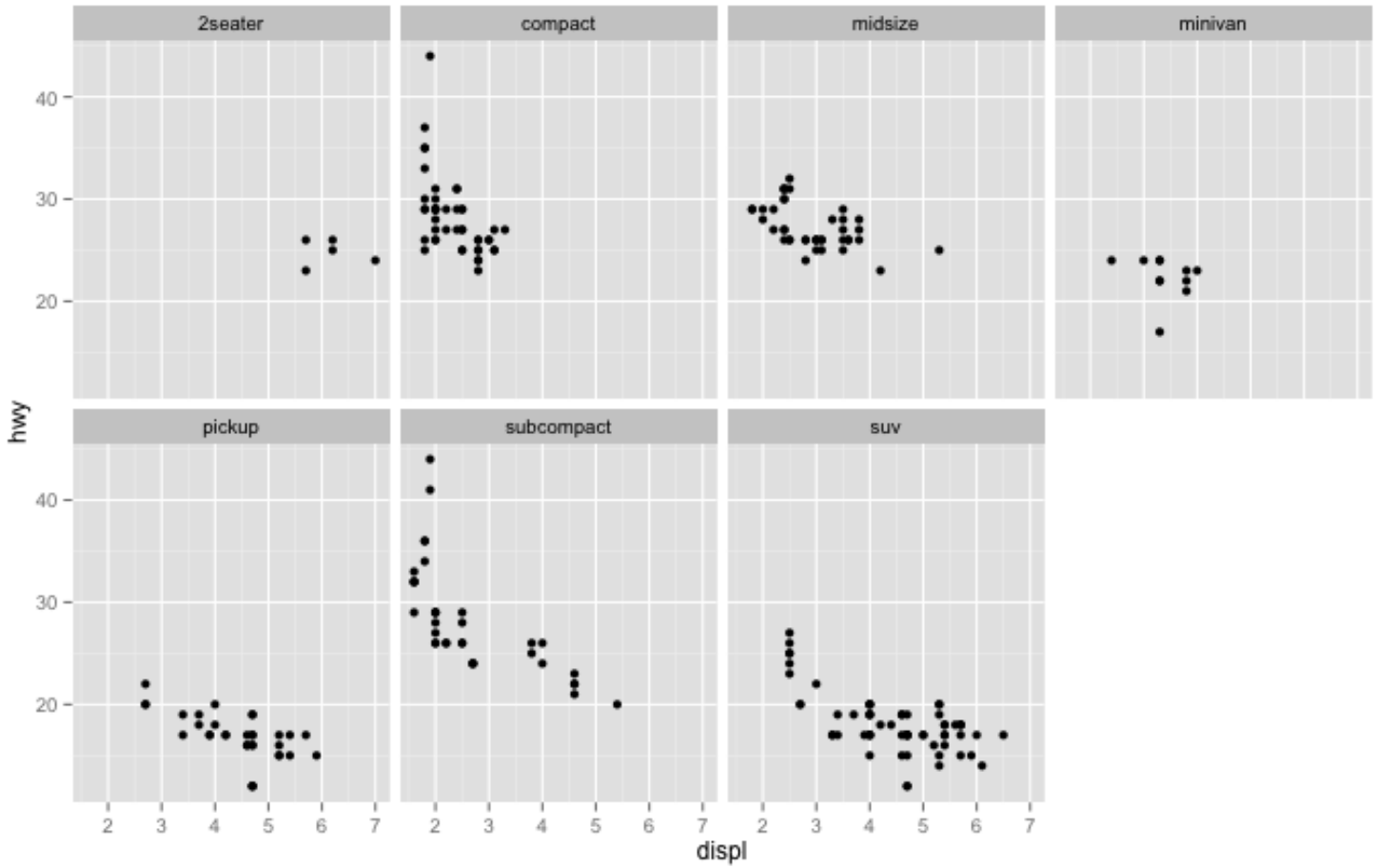
- Map a continuous variable in the `mpg` dataset, like `cty`, to the `alpha`, `shape`, and `size` aesthetics. What does this do?

Facets

- We saw we could add categorical variables to plots using aesthetics.
- Can also do this by splitting the plot into **facets**, which are subplots that each display one subset of the data.
- Use the `fact_wrap()` command to facet a plot by a single variable.
- The argument is a formula created with ~ followed by a variable name.

Facets

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



Check Yourself

Tasks

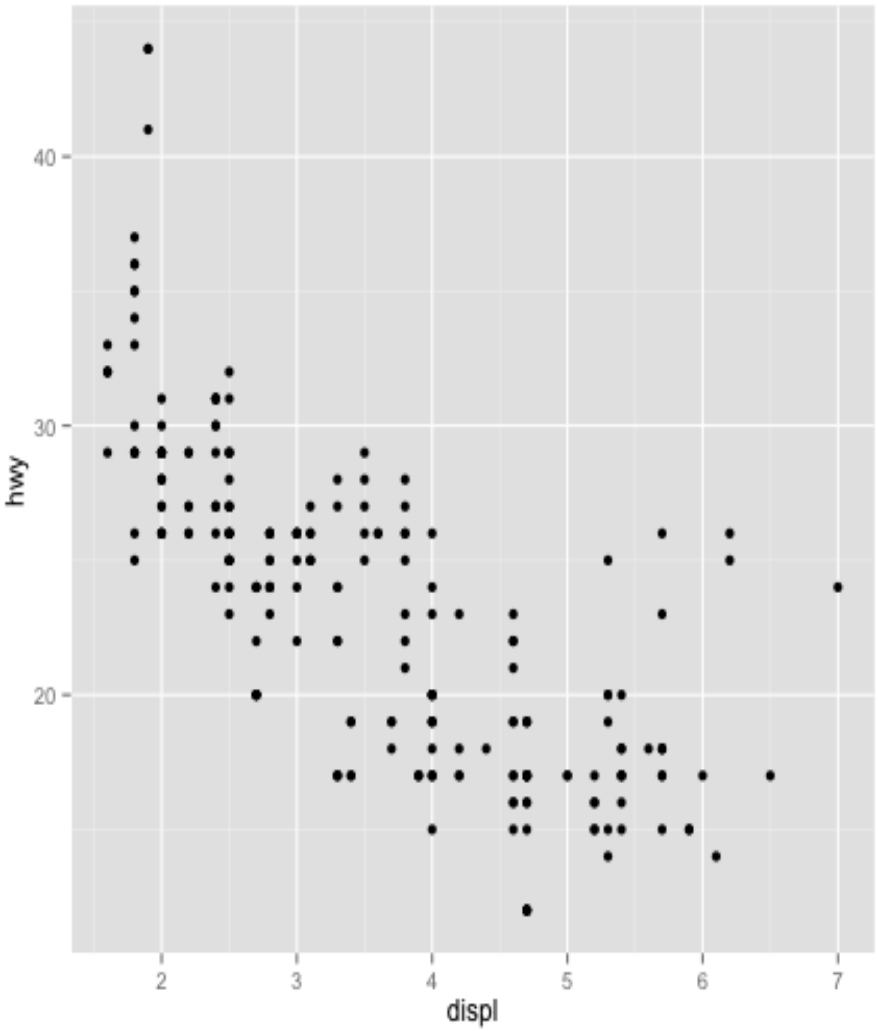
- Facet on two variables use the `facet_grid()` command. An example is the following:

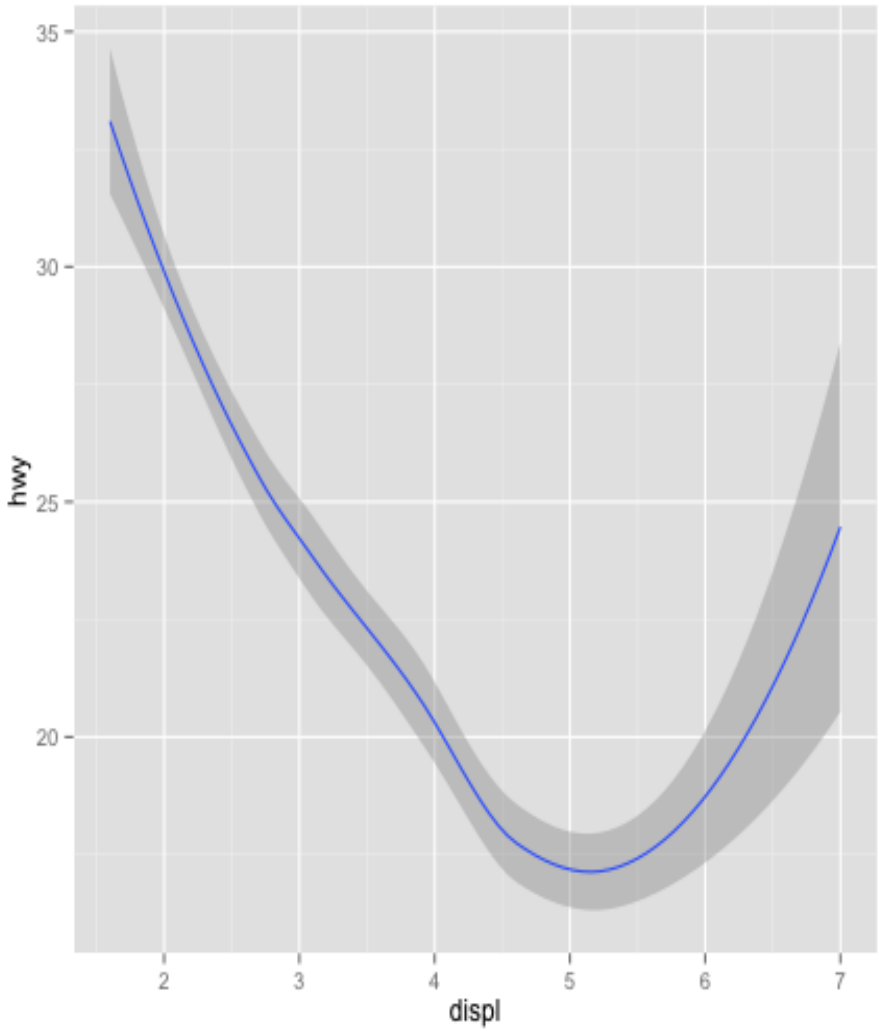
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ class)
```

  What do the empty cells mean?

- Look at `?facet_wrap`. What do `nrow` and `ncol` do? Why doesn't `facet_grid()` have `nrow` and `ncol` arguments?
- What happens if you facet on a continuous variable?

Geometric Objects



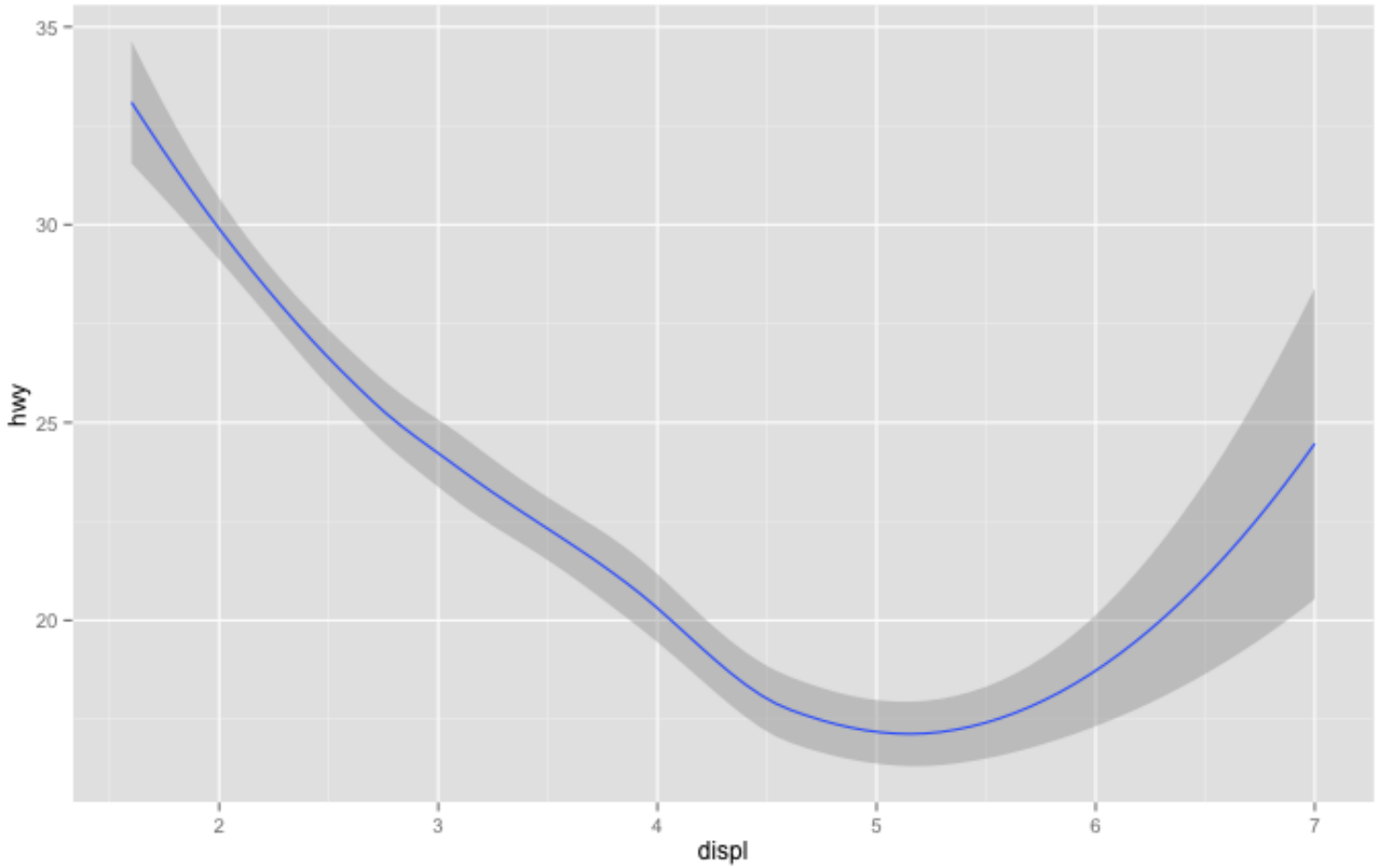


Geometric Objects

- In the previous slide, each plot used a different **visual object** to represent the data.
- Produce this by using different `geoms`.
- A `geom` is a geometrical object used to represent data in a plot.
- Often describe plots by the type of `geom` they use. For example, bar graphs use bar `geoms`.

Geometric Objects

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



Geometric Objects

- Every `geom` takes a `mapping` argument but not every aesthetic works with every `geom`.
  - E.g., you can set the `shape` of a point, but not a line. You can set the `linetype` of a line.
- `ggplot2` has around 30 different `geoms`.
- Can get help with `?geom_smooth`, for example.

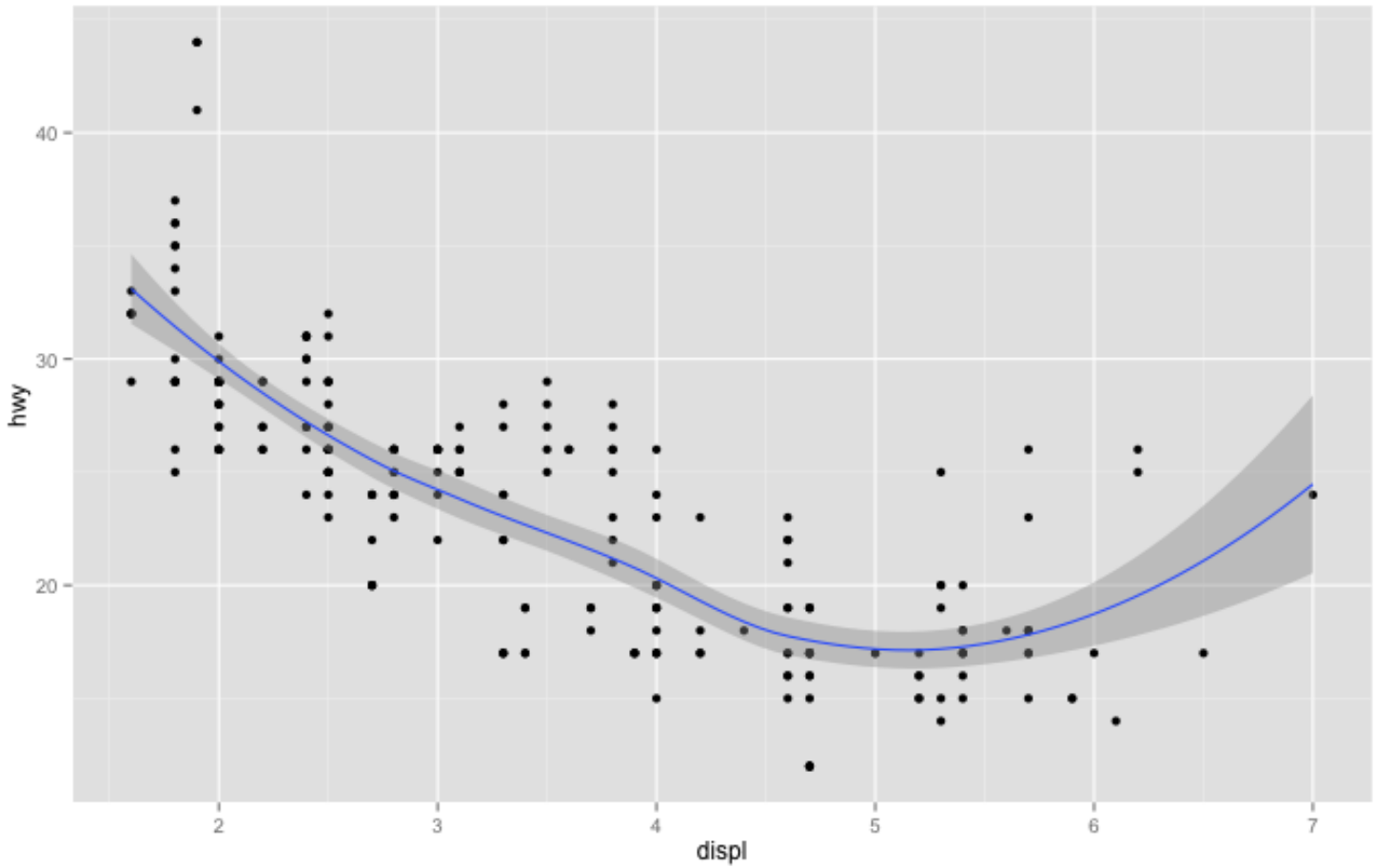
Geometric Objects

Some Commonly-used geoms

geom Name	Used to...	Aesthetics
geom_histogram	Visualize a Continuous Variable	x .
geom_bar	Visualize a Discrete Variable	x.
geom_point	Visualize a Two Continuous Variables	x, y.
geom_text	Add Labels to a Plot	x, y, label.
geom_boxplot	Visualize Continuous and Discrete Variables	x, y.
geom_jitter	Visualize a Two Variables	x, y.
many more ...		

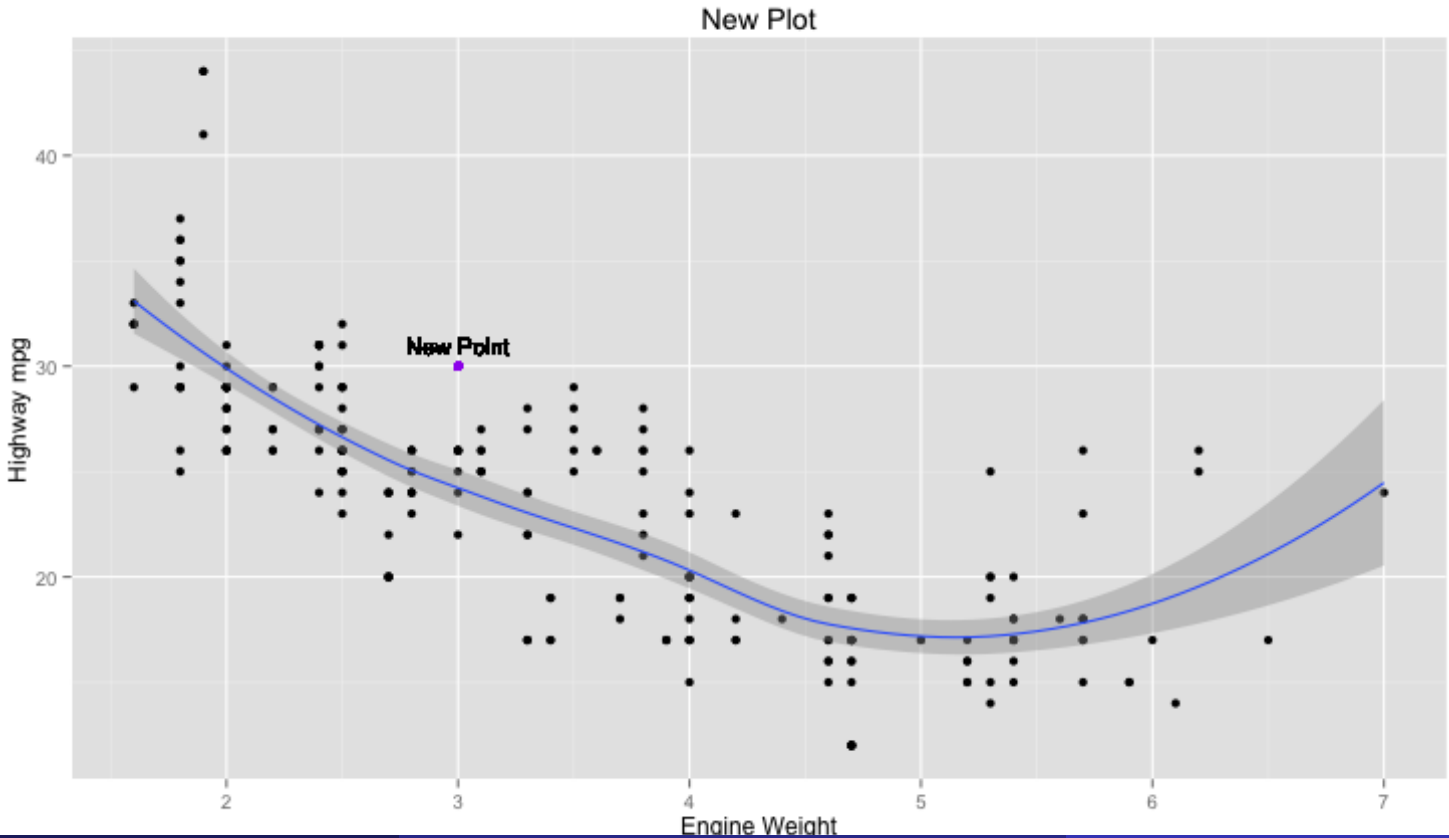
Layering geoms

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



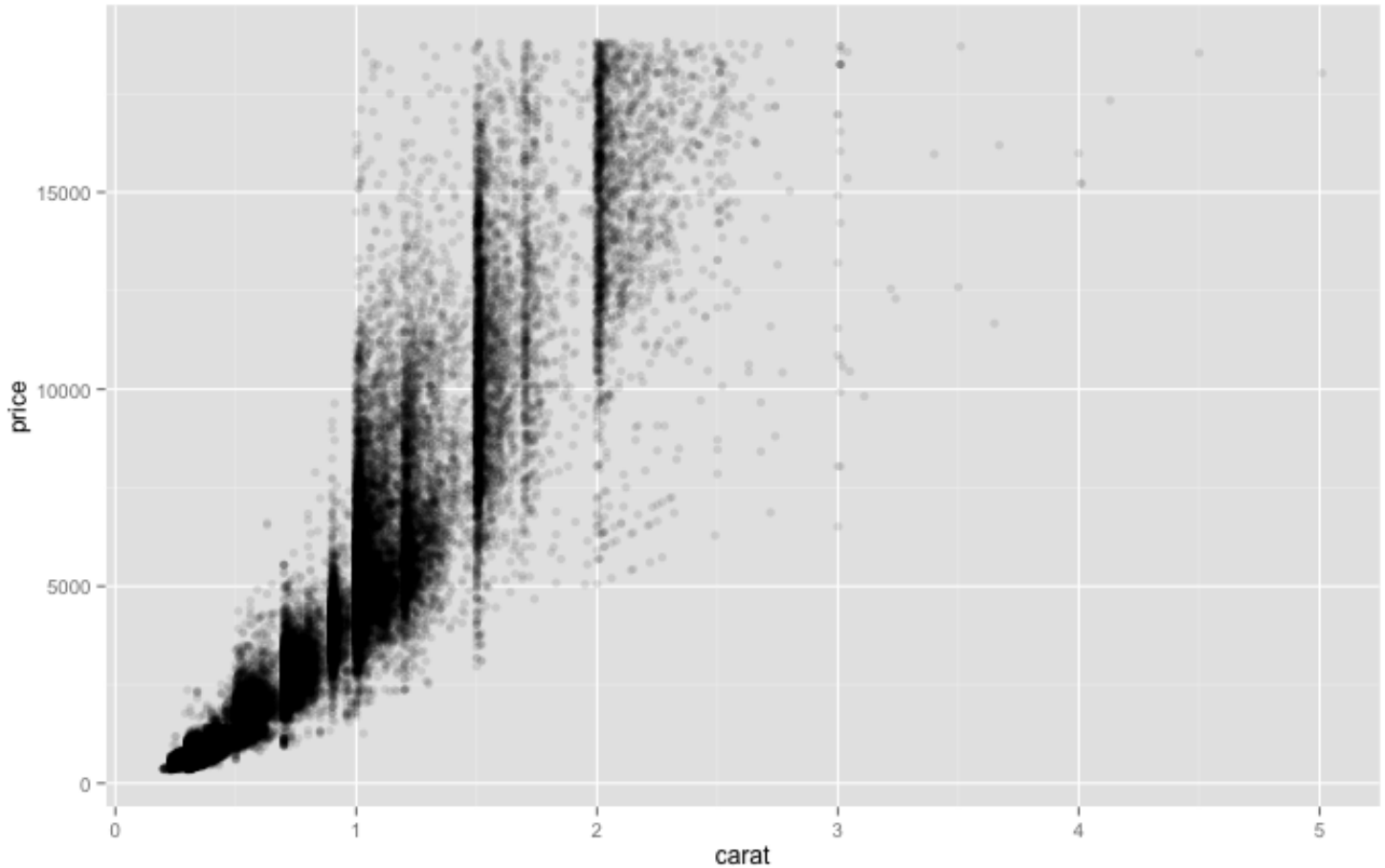
Adding Axis Labels

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(x=3, y=30), color = "purple") +
  geom_text(mapping = aes(x=3, y=31, label = "New Point"), size=4) +
  labs(title = "New Plot", x = "Engine Weight", y = "Highway mpg")
```



Layering geoms

```
> ggplot(data = diamonds) +
+   geom_point(mapping = aes(x = carat, y = price),
+              alpha = 1/10)
```



Check Yourself

Exercise:

Use the built-in `iris` dataset.

- Plot iris `Sepal.Width` on the x-axis and `Sepal.Length` on the y-axis. Color the points according to whether the iris is a setosa or not.
- Plot two regression lines on the plot, one for the setosa iris and one for non-setosa iris. Hint: Use `geom_abline(intercept, slope)`.