DATA 100,Week 1 (B)

Descriptive statistics, I

Throughout the course, we will review and talk about descriptive statistics. This is the first bit of it.

Reviewed / discussed here are **max, min, mean, median, percentile**, and **quartile**

The R code chunks involved in this part are NOT IMPORTANT for now, and they will be explained later when the need comes up

- The point is to review the descriptive statistics
- So you can treat every code block in this part as a blackbox, and don't worry too much about them

DATA 100,Week 1 (B)

Set up 20 integers

```
# The following code puts the 20 integers into the variable `sample`
## use "?c" in console to read the help messages
# The name `sample` now represents the 20 integers
sample <- c(29, 35, 33, 32, 34, 30, 28, 33, 34, 35, 10, 14, 14, 12, 12, 12, 10, 14,
16, 10)
```

DATA 100,Week 1 (B)

max, min, median, percentile, and quartile

Only for data that allows reasonable comparison in some way

The first two are more straightforward

- ✓ **max**: the largest entry in the data
- ✓ for the 20 integers in sample, the max is 35
- ✓ **min**: the smallest entry in the data
- ✓ for the 20 integers in sample, the min is 10

DATA 100,Week 1 (B)

The others generally take some work, and will use R code to help. With the 20 integers here, can still be done manually to see if things indeed work out right.

- ✓ **median**: a value for which at most half of the data are larger than it, AND at most half are smaller than it
- ✓ for the 20 integers in sample, we can sort them and count from either end
- ✓ ... any number in between 16 and 28 could be a median
- ✓ ... In this case, convention is to **take the average of** 16 and 28 ,
- ✓ ... thus "the" median for the sample data is 22

```
# this sorts the entries contained in sample
## use "?sort" to read the help messages
sort(sample)
#> [1] 10 10 10 12 12 12 14 14 14 16 28 29 30 32 33 33 34 34 35 35
```

DATA 100,Week 1 (B)

In fact, R has functions that directly output these statistics

```
# the names of the functions are very intuitive
max(sample)
#> [1] 35
min(sample)
#> [1] 10
median(sample)
#> [1] 22
```

DATA 100,Week 1 (B)

percentile

- Basic ideas:
  - ✓30 th percentile is a value so that 30% of the data are below it
  - ✓median is basically 50 th percentile
  - ✓90 th percentile is a value so that 90% of the data are below it

quartile: three of these, $Q1, Q2, Q3$

- ✓Q1: the 25th percentile, also called first quartile
- ✓Q2: the 50th percentile, i.e. the median, also called second quartile
- ✓Q3: the 75th percentile, also called the third quartile

- ✓Q1 and Q3 may be thought of as the medians of the lower and upper half of the data respectively

DATA 100,Week 1 (B)

The R function quantile finds percentile and quartiles

```
# print out the sorted sample, for manual verification
sort(sample)
#> [1] 10 10 10 12 12 12 14 14 14 16 28 29 30 32 33 33 34 34 35 35
# The code below finds the 20th percentile for the entris contained in sample
# Changing ".20" below by ".25", ".50", ".75" to see the quartiles
quantile(sample, .20)
#> 20%
#> 12
# without giving the numeric percentage, it produces all the quartiles, with max,
and min
quantile(sample)
#> 0% 25% 50% 75% 100%
#> 10 12 22 33 35
```

DATA 100,Week 1 (B)

mean

Only for data that allows reasonable arithmetic operations

- The data must be numerical
- ... but not any numerical looking data
  - ✓ your student IDs look numerical, but taking mean really makes no sense

mean: the average of the values in the data

- computed by adding all the values up and divide by the number of entries
- for sample, there are 20 entries, so add everything up in sample, and divide the result by 20

```
# directly computing the mean
(29+35+33+32+34+30+28+33+34+35+10+14+14+12+12+12+10+14+16+10)/20
#> [1] 22.35
```

DATA 100,Week 1 (B)

Even better, the mean function in R directly computes the mean for sample

```
mean(sample)
#> [1] 22.35
```

Comment: There are always more than one ways of doing things, as long as we understand what is going on.

DATA 100,Week 1 (B)

summary function

The function summary in R produces six of the statistics described here:

```
# try "?summary" in console to see the help information
summary(sample)
#> Min. 1st Qu. Median Mean 3rd Qu. Max.
#> 10.00 12.00 22.00 22.35 33.00 35.00
```

DATA 100,Week 1 (B)

Example and Practice

```
>x <- c(2, 3, 5, 2, 7, 1) # x then holds values 2, 3, 5, 2, 7, 1
>x
[1] 2 3 5 2 7 1
>y <- c(10, 15, 12)
>y
[1] 10 15 12
>z <- c(x, y)
>z
[1] 2 3 5 2 7 1 10 15 12
```

DATA 100,Week 1 (B)

Example and Practice

```
>x <- c(3, 11, 8, 15, 12)
>x > 8
[1] FALSE TRUE FALSE TRUE TRUE
>x != 8
[1] TRUE TRUE FALSE TRUE TRUE
```

DATA 100,Week 1 (B)

Patterned data

Use, for example, 5:15 to generate all integers in a range, here between 5 and 15 inclusive:

```
>5:15
[1] 5 6 7 8 9 10 11 12 13 14 15
```

Conversely, 15:5 will generate the sequence in the reverse order.

DATA 100,Week 1 (B)

Example and Practice

```
class(c(1, 2, 3))
## [1] "numeric"
class(c(TRUE, TRUE, FALSE))
## [1] "logical"
class(c("Hello", "World"))
## [1] "character"

is.numeric(c(1, 2, 3))
## [1] TRUE
is.numeric(c(TRUE, TRUE, FALSE))
## [1] FALSE
is.numeric(c("Hello", "World"))
## [1] FALSE
```

DATA 100,Week 1 (B)

Example and Practice

Strings cannot be directly used for numerical operations

```
strings <- c("1", "2", "3")
class(strings)
## [1] "character"
```

```
strings + 10
## Error in strings + 10
```

DATA 100,Week 1 (B)

We can use as.numeric( ) to convert a character vector into a numeric vector:

```
numbers <- as.numeric(strings)
numbers
## [1] 1 2 3
class(numbers)
## [1] "numeric"
```

```
numbers + 10
## [1] 11 12 13
```

DATA 100,Week 1 (B)

```
as.numeric(c("1", "2", "3", "a"))
## Warning:
## [1]  1  2  3 NA

as.character(c(1, 2, 3))
## [1] "1" "2" "3"
as.character(c(TRUE, FALSE))
## [1] "TRUE"  "FALSE"
```

DATA 100,Week 1 (B)

```
> library(datasets)
> head(BJsales)
[1] 200.1 199.5 199.4 198.9 199.0 200.2
> class(BJsales)
[1] "ts"
> mode(BJsales)
[1] "numeric"
> length(BJsales)
[1] 150
```

- the second line of code displays the first 6 elements of BJsales
- The third line of code examines the attributes of BJsales, which returns "ts," indicating that BJsales contains a time series vector, meaning the data in BJsales is arranged in chronological order.
- Typing "?ts" in the console provides more information about the ts data type.

DATA 100,Week 1 (B)

- The result of the mode() function indicates that the data in BJsales is numeric, making it meaningful to calculate the mean, median, and quantiles of BJsales.

- The length() function reveals that BJsales contains a total of 150 elements.

DATA 100,Week 1 (B)

```
> mean(BJsales)
[1] 229.978
> mean(BJsales,trim=0.1)
[1] 229.715
> median(BJsales)
[1] 220.65
```

- The first line calculates the mean of all elements in BJsales,

- The second line's trim parameter specifies that the largest 10% and smallest 10% of values should be excluded when calculating the mean.

- This trimming function helps mitigate the impact of outliers.

Note: The results of these two calculation methods are very close, indicating that there are no outliers in BJsales that significantly deviate from the majority of the data.

DATA 100,Week 1 (B)

Set up tidyverse packages and load them

- ✓ **First time installation may take some time**, because some packages will need to be compiled and installed from scratch

- ✓ Packages need to be installed **only once**, so later files will not contain the following code block

```
install.packages('tidyverse', repos = "https://utstat.toronto.edu/cran/")

install.packages('ggthemes', 'ggridges')
```

DATA 100,Week 1 (B)

If you have problem using install.packages,running one of the following in the Console might help

• where PACKAGE_NAME is replaced by the name of the package you wish to install

```
install.packages(PACKAGE_NAME, repos="https://utstat.toronto.edu/cran/")
install.packages(PACKAGE_NAME, repos="https://muug.ca/mirror/cran/")
```

DATA 100,Week 1 (B)

A package must be loaded before we can use it. Thus each new session must **load all the packages that will be used in it**

• In particular, library(tidyverse) should **appear in the beginning of every .qmd (or .rmd) file** we have from now on

```
library(tidyverse)
library(ggthemes)
```

DATA 100,Week 1 (B)

Datasets

Later files will not have to contain the following line.

```
install.packages(c('nycflights13', 'gapminder', 'Lahman'), repos =
"http://cran.utstat.utoronto.ca/")
```

```
install.packages(c('nycflights13', 'gapminder', 'Lahman'),
repos="https://muug.ca/mirror/cran/")
```

DATA 100,Week 1 (B)

- ✓ Running the following command in the Console or clicking the run arrow of the code block below will show a list of available datasets in a separate tab

```
data()
```

- ✓ You can also use ? in the Console to get more details on a specific dataset. The Help panel in the lower right of the RStudio window should contain further information.

```
?mpg
```

DATA 100,Week 1 (B)

- The software is constantly being updated and tidyverse installed may not be the latest version.

- To keep the tidyverse installation up-to-date, run the command in the code block below once in a while.

```
tidyverse_update()
```

DATA 100,Week 1 (B)

ggplot2 basics

ggplot2 is a part of tidyverse. Main tool for data visualization used throughout the course.

mpg data

US EPA data on 38 models of cars: run the following in console, and look at the details in the Help panel on the lower right part of RStudio.

```
mpg

View(mpg)
```

DATA 100,Week 1 (B)

Terminology

✓ Columns are variables
✓ Rows are observations
✓ The whole thing is a dataframe

Intuitively, a **dataframe** is a *rectangular shaped collection of data*, with **column** headers indicating the properties that we are interested in, and the **rows** correspond to individual instances that have those properties.

DATA 100,Week 1 (B)

Variable types

Each column is supposed to contain a certain type of data. The types are defined by the R language and the packages you load.

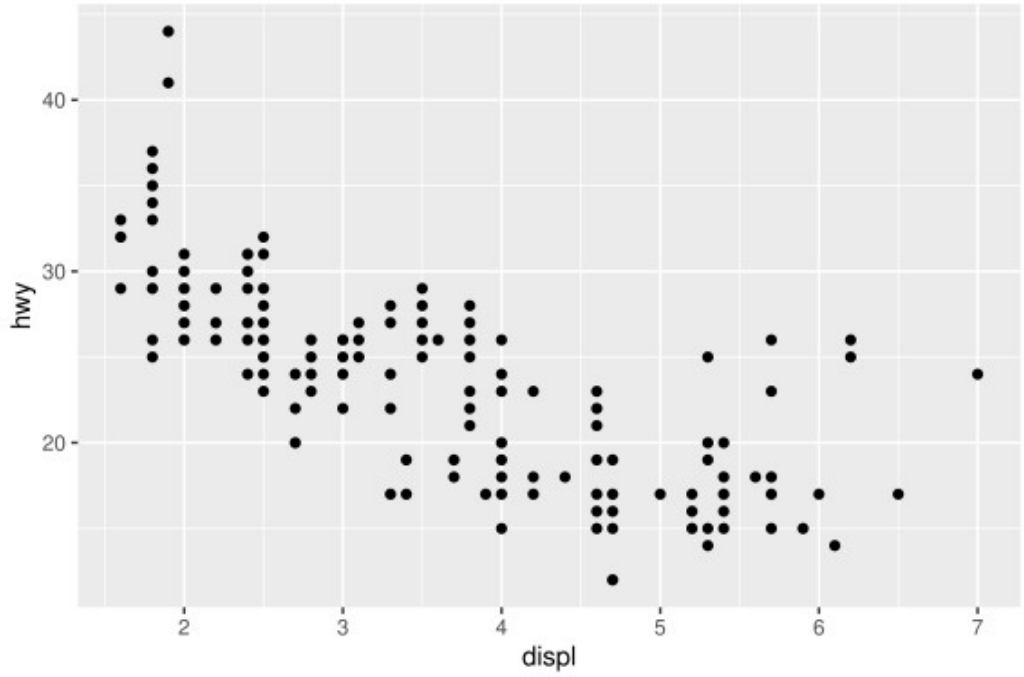
There are many, and we list some below:

- **character**: <chr>,strings, formed by characters
- **numeric**: <dbl>,doubles, real numbers
- **integer**: <int>,integers,
- **boolean**: <lgl>,can only be TRUE or FALSE
- **datetime**: <dttm>,date-times, a date + a time
- **others** … will talk about them as they show up

DATA 100,Week 1 (B)

First plots

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



DATA 100,Week 1 (B)

Comment:

In the above, the parameters for the functions are passed by **name**. For instance

• ggplot(data = mpg) indicates that the function ggplot accepts a parameter named data

for this particular call of ggplot, it is to be mpg

• Similarly, aes(x = displ, y = hwy) indicates that the function aes accepts two parameters named x and y

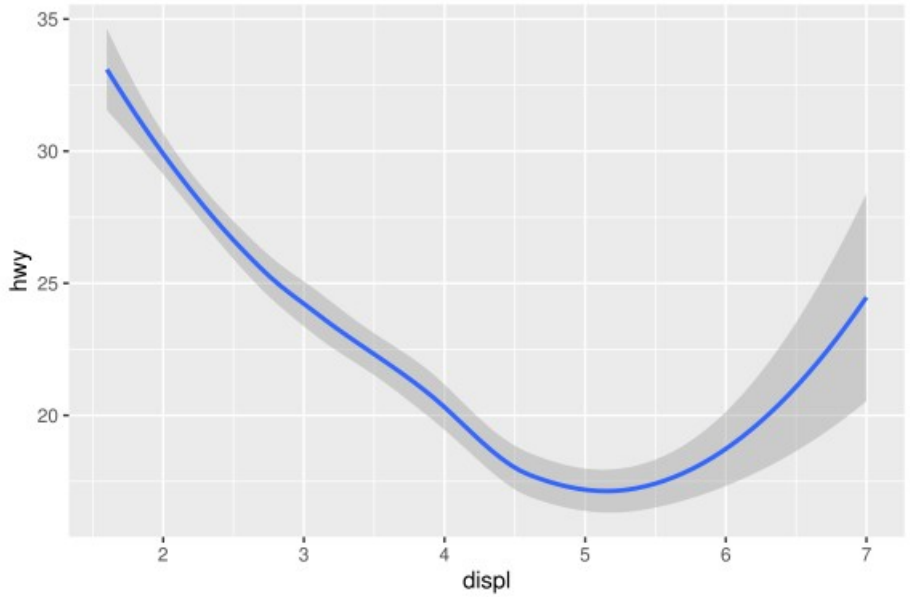
for this particular situation, the x is to be displ and y is to be hwy

• Then, geom_point(mapping = aes(x = displ, y = hwy)) indicates that the function geom_point accepts a parameter named mapping

DATA 100,Week 1 (B)

Smooth curve

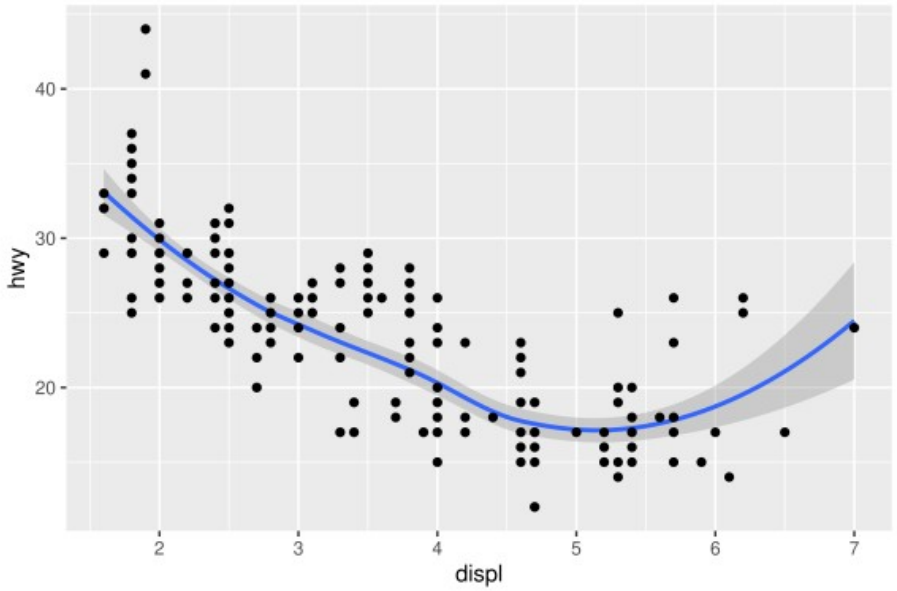
```
ggplot(data = mpg) +
geom_smooth(mapping = aes(x = displ, y = hwy))
```



DATA 100,Week 1 (B)

Can overlay on top of each other

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



DATA 100,Week 1 (B)

Structure of ggplot() calls

The general form of plotting using ggplot is the following:

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

- **DATA:** the actual dataframe
- **GEOM_FUNCTION**: e.g. geom_point, geom_smooth, geom_bar, geom_polygon, etc.
  – Names of the function are generally self-explanatory.
  Use ?FUNCTION_NAME in console if not sure.
  – … generally it gives the most up to date information, if your RStudio is up-to-date

DATA 100,Week 1 (B)

• To highlight details, there are more parameters that one can put in these functions

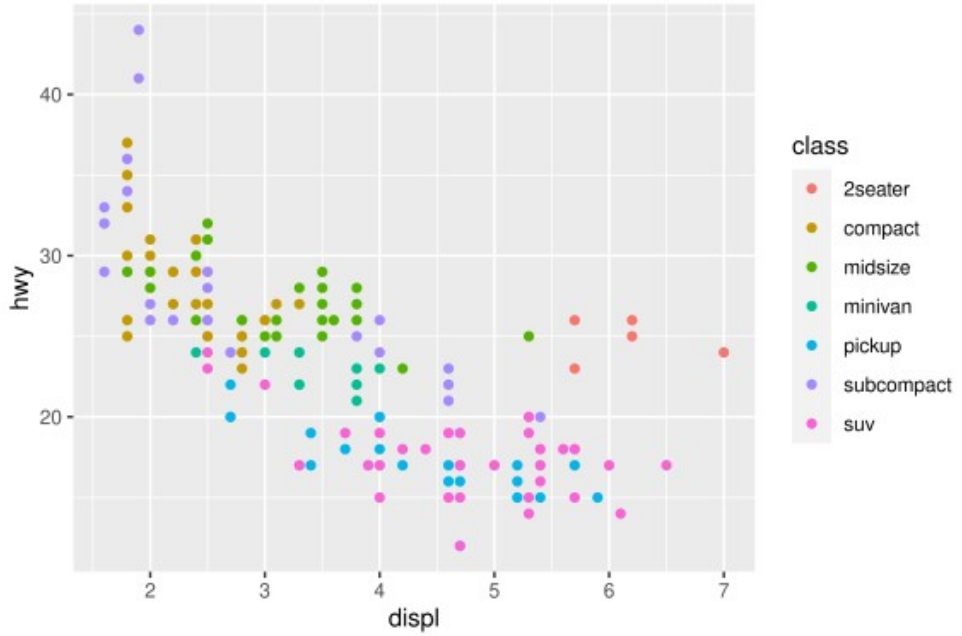
– MAPPINGS component provides ways of adding details to the plot, such as *shades*, *size*, *shape*, etc, to represent other variables / properties

– GEOM_FUNCTIONs can be combined by + them together, as above, and can also combine with other functions

DATA 100,Week 1 (B)

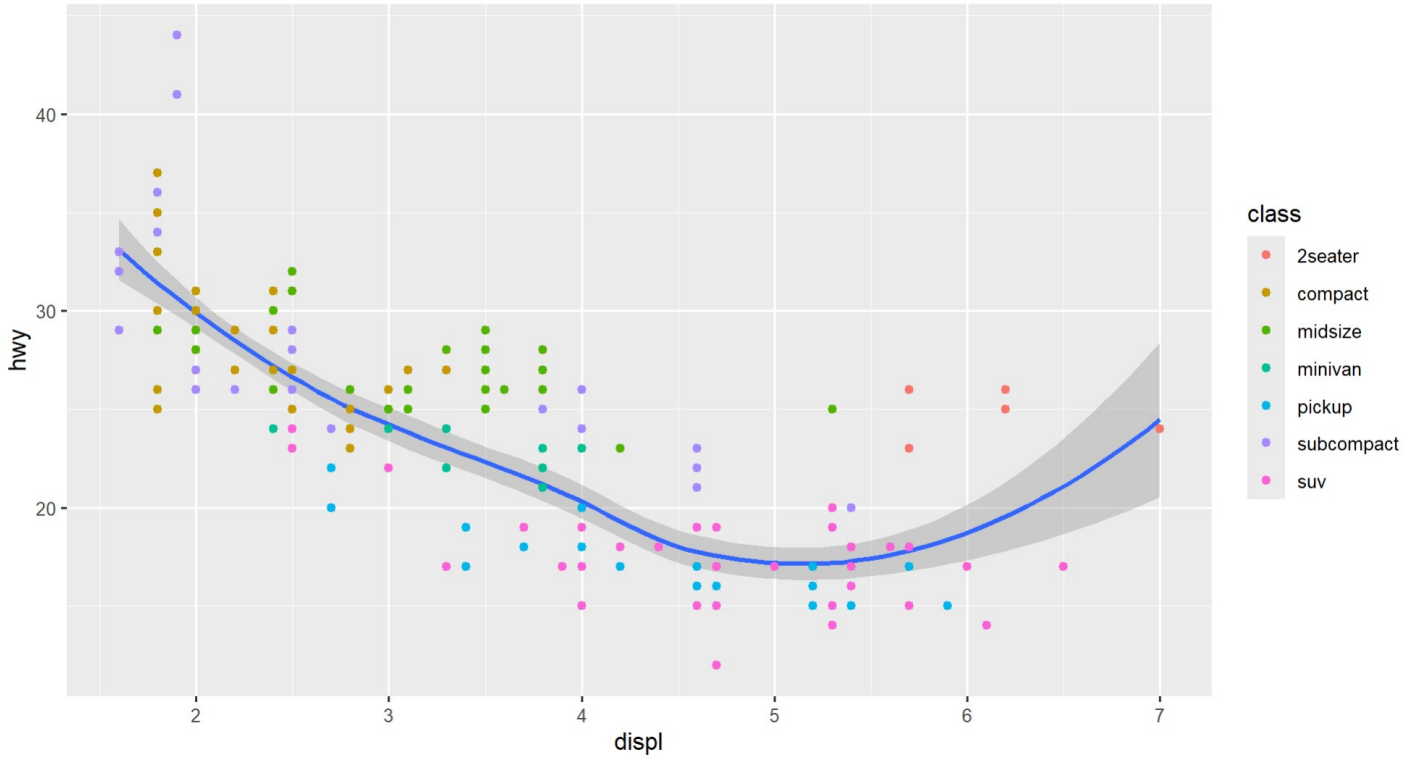
Add color to represent other variables

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



DATA 100,Week 1 (B)

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



DATA 100,Week 1 (B)

Aesthetics matching variable type

Besides color, there are also

- **alpha**: the shades of the points
- **size**: the sizes of the points
- **shape**: the shapes of the points

DATA 100,Week 1 (B)

Two classes of variable types:

- **continuous**: values may be continuously distributed, as on the real line
  – alpha and size

- **categorical**: values are discretely distributed, generally finitely many

  – shape

  – only 25 building shapes in R

- color can be used on both types

For categorical variables – also called factors, can also use the facet_ functions

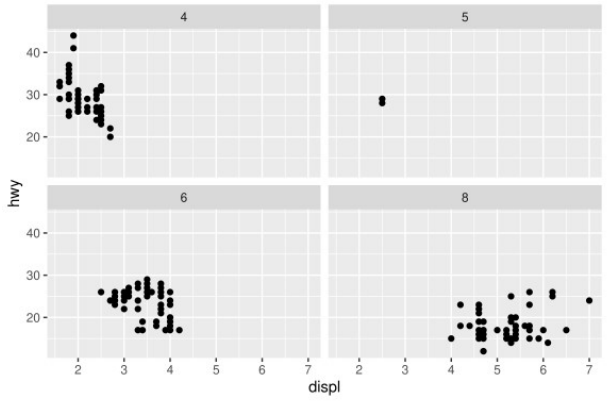
- Plot data with different values on the variable separately

DATA 100,Week 1 (B)

facet_wrap with respect to one variable

This generates a list of plots, each plotting a subset of the data, and the plots are titled by the value of the faceting variable

```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy)) +
#facet_wrap(~ class, nrow = 2) # test what would happen with ncol=2?
facet_wrap(~cyl)
```



DATA 100,Week 1 (B)

facet_grid **with respect to two variables**

This generates a grid of plots, labelled across columns and rows by the values in the respective variable

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