30535 Skills Problem Set 1

Mia Jiang

03/31/2022

Front matter
This submission is my work alone and complies with the 30535 integrity policy.

Add your initials to indicate your agreement: **M.J.**

Late coins used this pset: 1. Late coins left: 4.

Clear Global Environment

```
knitr::opts_chunk$set(echo = TRUE)
rm(list=ls())
```

Working Directory and Loading Packages

```
# Setting the Working Directory
setwd("~/Desktop/DPPP R/Week 1/PS/skills-problem-set-1-weiluj")
```

1.1.1-1.1.3

Followed the instructions

1.1.4

```
# Loading Packages
install.packages("devtools")
# I've already installed and updated packages so I set eval=FALSE here to avoid delay in R
```

1.1.5

```
devtools::install_github("hadley/r4ds")
# Same as in Q1.4
```

1.1.6

Created a github account

1.1.7

Downloaded Github Desktop

1.1.8

Watched the video

1.1.9

I downloaded the template but created my own RMD file to finish the assignment

1.1.10

```
# Install Packages
list_of_packages <- c("tidyverse", # learn more at tidyverse.org
                      "rmarkdown" # http://rmarkdown.rstudio.com
                      )
```

1.1.11

```
# Test which packaegs are installed
new.packages <- list_of_packages[!(list_of_packages %in% installed.packages()[,"Package"])]
new.packages
```

```
## character(0)
```

```
# Print names of all installed packages
installed.packages <- as.data.frame(installed.packages())
nrow(installed.packages()) # Total number of packages
```

```
## [1] 264
```

```
head(installed.packages) # The first 10 packages
```

```
##               Package
## abind           abind
## anytime       anytime
## askpass       askpass
## assertthat assertthat
## babynames   babynames
## backports   backports
##                                                                LibPath
## abind      /Library/Frameworks/R.framework/Versions/4.0/Resources/library
## anytime    /Library/Frameworks/R.framework/Versions/4.0/Resources/library
## askpass    /Library/Frameworks/R.framework/Versions/4.0/Resources/library
## assertthat /Library/Frameworks/R.framework/Versions/4.0/Resources/library
```

```
## babynames  /Library/Frameworks/R.framework/Versions/4.0/Resources/library
## backports  /Library/Frameworks/R.framework/Versions/4.0/Resources/library
##            Version Priority      Depends          Imports            LinkingTo
## abind        1.4-5     <NA> R (>= 1.5.0)   methods, utils                 <NA>
## anytime      0.3.9     <NA> R (>= 3.2.0) Rcpp (>= 0.12.9) Rcpp (>= 0.12.9), BH
## askpass        1.1     <NA>         <NA>     sys (>= 2.1)                 <NA>
## assertthat   0.2.1     <NA>         <NA>            tools                 <NA>
## babynames    1.0.1     <NA>  R (>= 2.10)           tibble                 <NA>
## backports    1.4.1     <NA> R (>= 3.0.0)             <NA>                 <NA>
##                               Suggests Enhances            License
## abind                             <NA>     <NA>        LGPL (>= 2)
## anytime    tinytest (>= 1.0.0), gettz     <NA>         GPL (>= 2)
## askpass                      testthat     <NA> MIT + file LICENSE
## assertthat             testthat, covr     <NA>              GPL-3
## babynames         testthat (>= 3.0.0)     <NA>                CC0
## backports                         <NA>     <NA>      GPL-2 | GPL-3
##            License_is_FOSS License_restricts_use OS_type MD5sum
## abind                 <NA>                  <NA>    <NA>   <NA>
## anytime               <NA>                  <NA>    <NA>   <NA>
## askpass               <NA>                  <NA>    <NA>   <NA>
## assertthat            <NA>                  <NA>    <NA>   <NA>
## babynames             <NA>                  <NA>    <NA>   <NA>
## backports             <NA>                  <NA>    <NA>   <NA>
##            NeedsCompilation Built
## abind                    no 4.0.2
## anytime                 yes 4.0.2
## askpass                 yes 4.0.2
## assertthat               no 4.0.2
## babynames                no 4.0.2
## backports               yes 4.0.2
```

```
# Stored the information of installed.packages in a new dataframe
# because print the names of all 264 packages will make the RMD file tediously long
```

Reference
https://stackoverflow.com/questions/4090169/elegant-way-to-check-for-missing-packages-and-install-them

1.1.12

weiluj

1.1.13

Add and commit code with commit message "start-up completed"

1.1.14

Watched the video

1.1.15

Finished Reverting. See Github document with summary called "Q1.1.15 Revert"

1.2

Completed the quizzes on canvas

2.1

```
# Loading Packages
library(tidyverse)
```

2.1.1

```
# Calculate rows in mpg
nrow(mpg)
```

```
## [1] 234
```

```
# Calculate columns in mpg
ncol(mpg)
```

```
## [1] 11
```

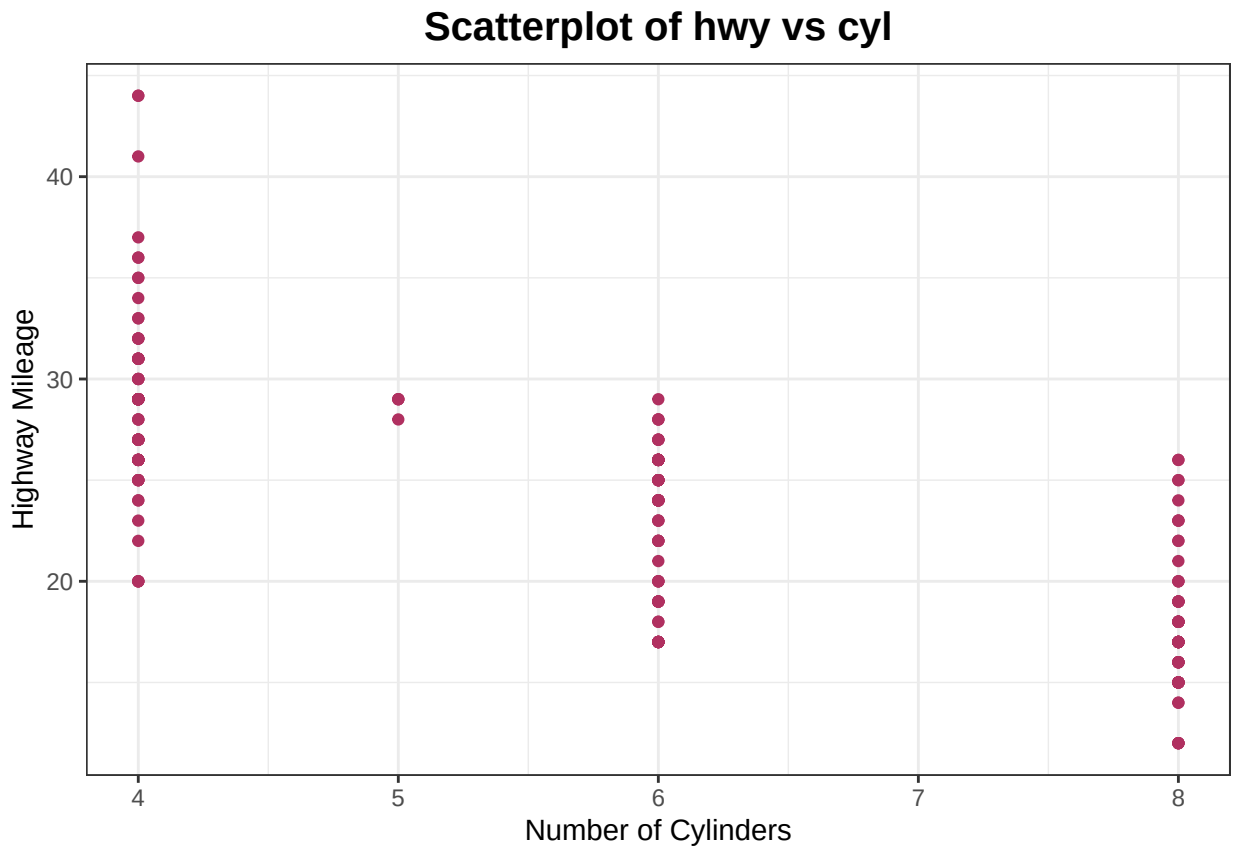
```
# View the structure of mpg
str(mpg)
```

```
## tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
##  $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
##  $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
##  $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
##  $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
##  $ cyl         : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
##  $ trans       : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
##  $ drv         : chr [1:234] "f" "f" "f" "f" ...
##  $ cty         : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
##  $ hwy         : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
##  $ fl          : chr [1:234] "p" "p" "p" "p" ...
##  $ class       : chr [1:234] "compact" "compact" "compact" "compact" ...
```

Interpretation
Each row represents an observation in the dataset, and each column represents a variable of observations in the dataset.

2.1.2

```
# Scatterplot of hwy vs cyl
ggplot(mpg) +
  geom_point(mapping= aes(cyl, hwy), col = "maroon") +
  labs(title= "Scatterplot of hwy vs cyl",
       x = "Number of Cylinders",
       y = "Highway Mileage") +
  theme_bw() +
  theme(plot.title = element_text(size = 15, face = "bold", hjust = 0.5))
```



2.1.3

```
#Interpret variable drv
?mpg
table(mpg$drv) # Find the distribution of drv types
```

```
##
##   4   f   r
## 103 106  25
```

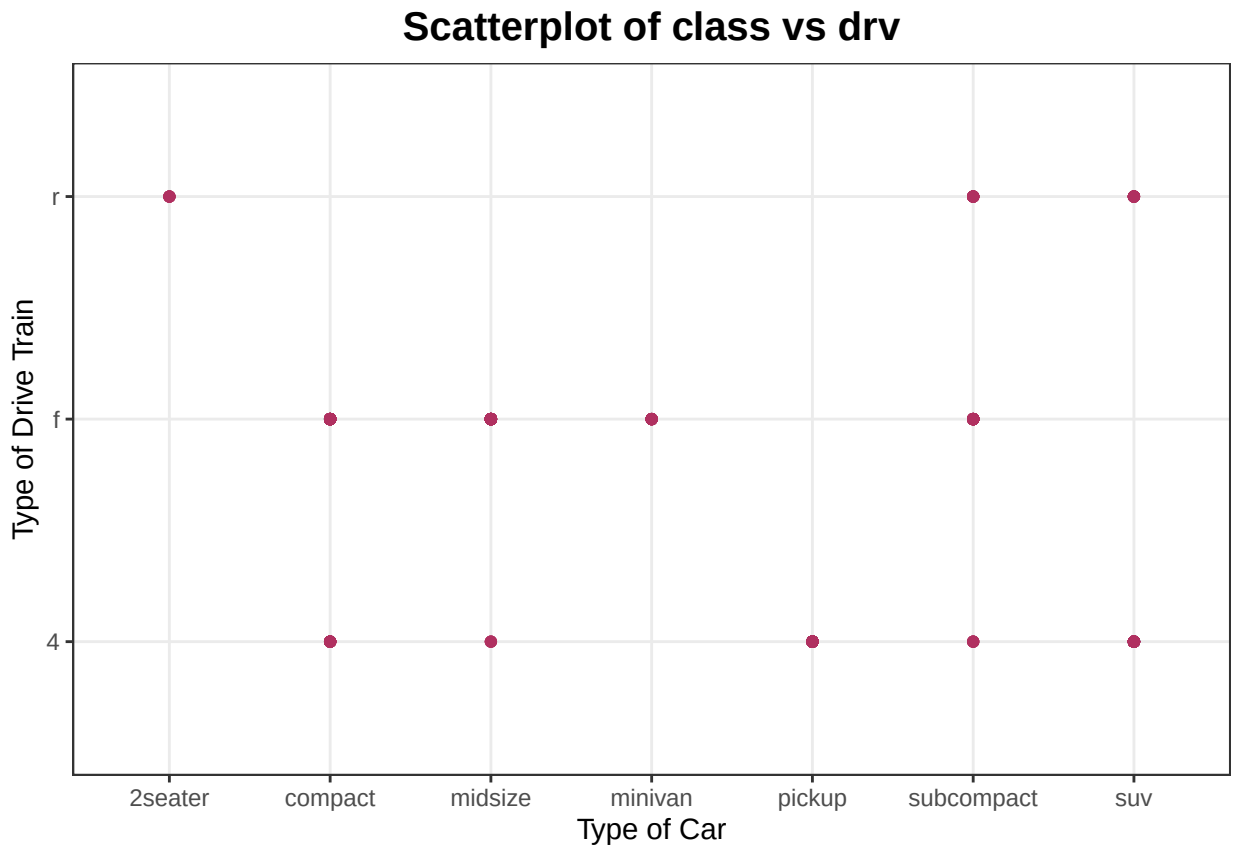
```r
table(mpg$class) # Find the distribution of class types
```

```
##
##    2seater    compact    midsize    minivan     pickup subcompact        suv
##          5         47         41         11         33         35         62
```

Interpretation
With help of r function, we know that the variable *drv* describes the type of drive train the car uses, of which *f* means *front-wheel drive*, *r* means *rear wheel drive* and *4* means *four-wheel drive*

2.1.4

```r
# Scatterplot of drv vs class
ggplot(mpg) +
  geom_point(mapping= aes(class, drv), col = "maroon") +
  labs(title = "Scatterplot of class vs drv",
       x = "Type of Car",
       y = "Type of Drive Train") +
  theme_bw() +
  theme(plot.title = element_text(size = 15, face = "bold", hjust = 0.5))
```



```
# I set the variable which has more categories, i.e. class, as the x variable to
# make the plot more readable
```

Reference
https://stats.stackexchange.com/questions/123938/how-to-determine-which-variable-goes-on-the-x-y-axes-in-a-scatterplot

Interpretation
The plot is not that useful because both *class* and *drv* are categorical variables. This means that we cannot observe how many points are for each combination. What we could observe in the plot is the combination types of *The type of car* and *The type of drive train*, which are at most $3 \times 7 = 21$ in this case.
Besides, the type of car does not usually determine the type of drive. We only observe 12 types of combination in the plot and most classes are paired with multiple types of drive.We could not conclude any correlation between the two factors.

2.2

2.2.1

```
?mpg # Pull up the documentation
print(mpg) # View the first 10 rows
```

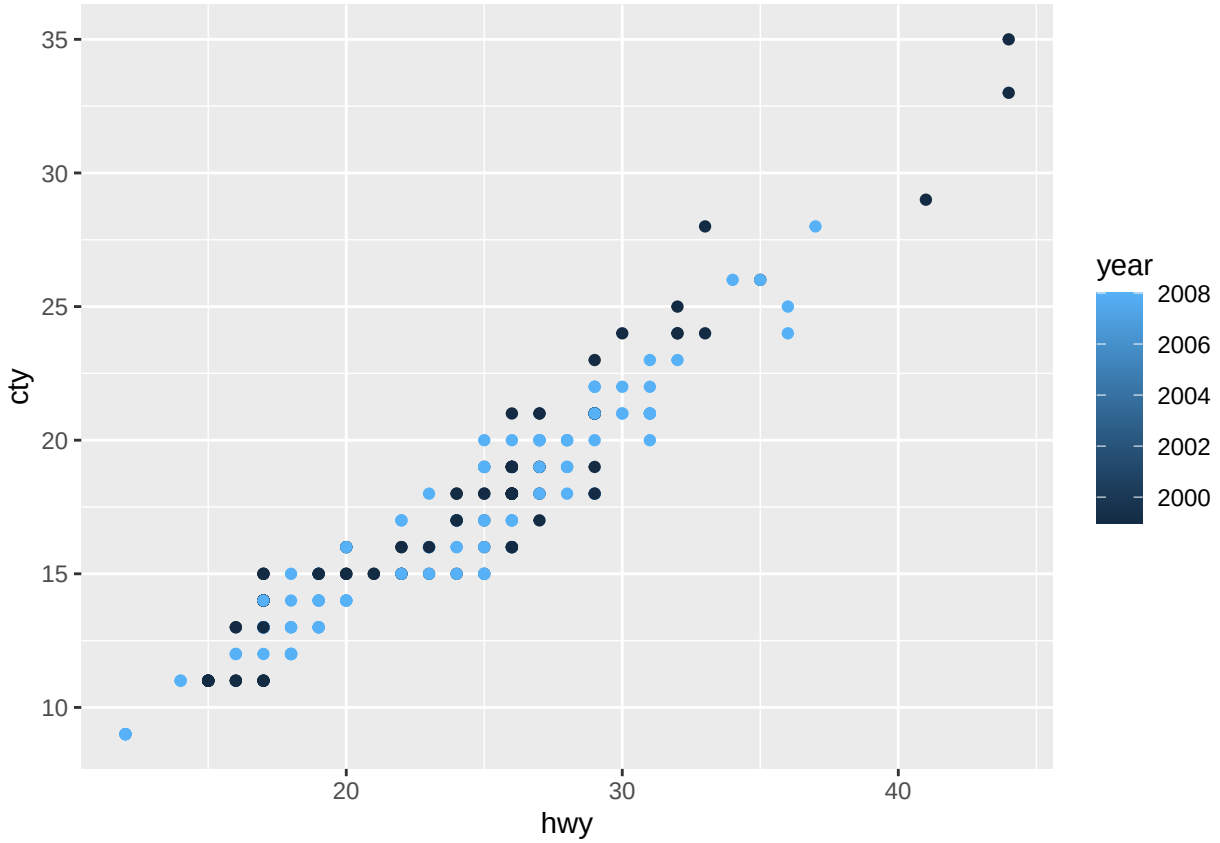
```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4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

Interpretation

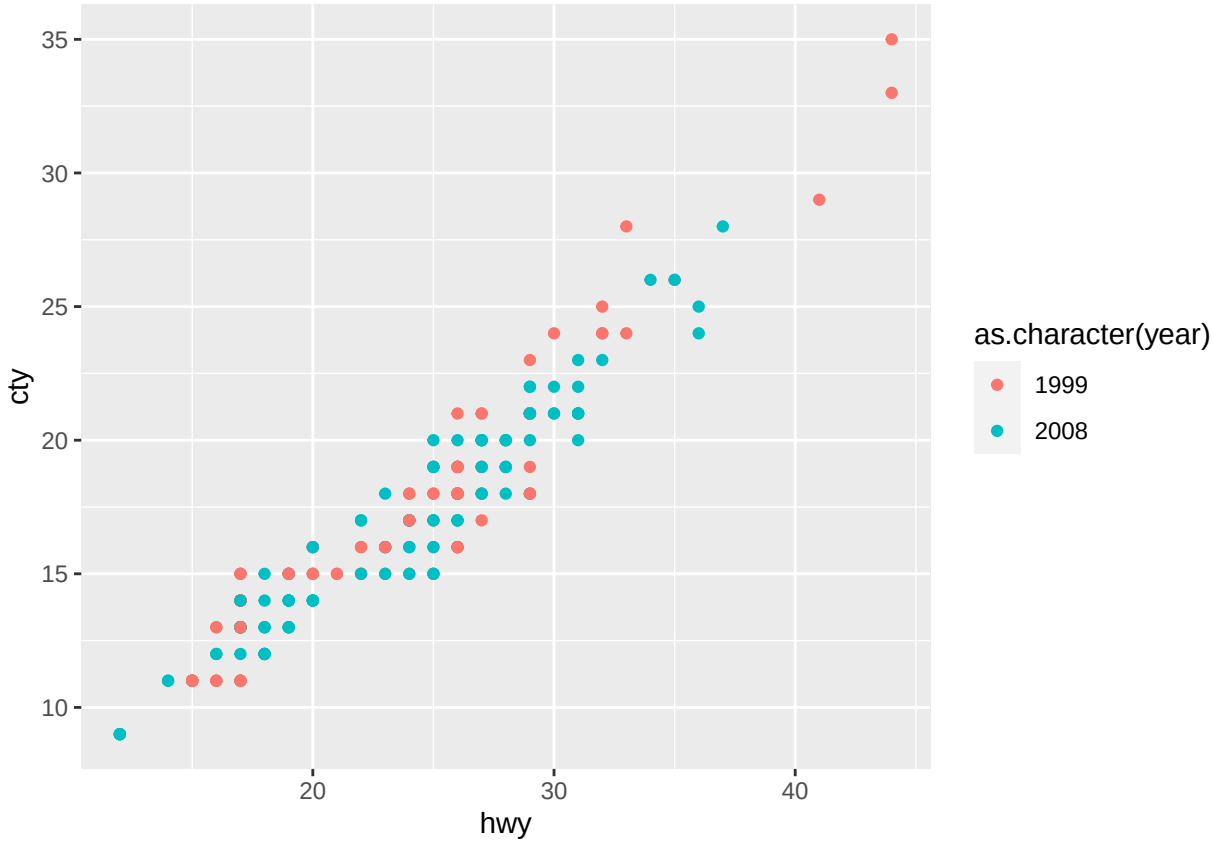
- categorical: chr
- continuous: dbl,int. dbl stands for double, and int stands for integer.

2.2.2

```
# Run the code provided
# Graph 1
ggplot(data = mpg) +
  geom_point(mapping = aes(x = hwy, y = cty, color = year))
```




```
# Graph 2
ggplot(data = mpg) +
  geom_point(mapping = aes(x = hwy, y = cty, color = as.character(year)))
```




```
# Find the type of year
class(mpg$year)
```

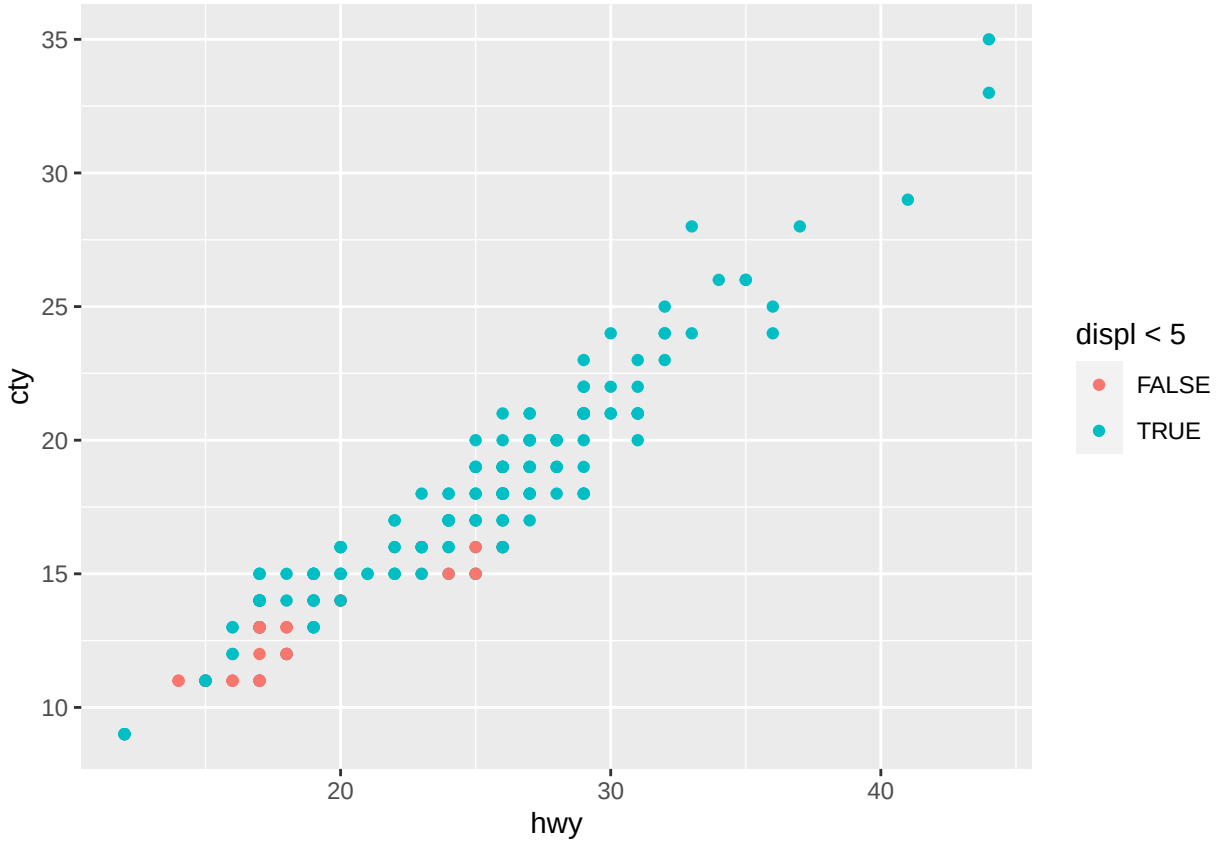
```
## [1] "integer"
```

Interpretation

- The variable *year* is integer so if we do not use the function as.character to make it become a character, R will treat it as numeric and we'll have a legend for each year between 1998 and 2020, which differs from light blue to dark blue. However, if we convert the type of *year* into character, only Year 1998 and 2020 will be chosen as legend type and be classified with 2 different colors. The other parts of the two plots are the same.
- The 2nd plot is better because (1) We only observe data from Year 1998 and 2020 in the dataset. It's meaningless to represent legends of other years while there is no relevant data. (2) The second plot distinguish year 1998 and 2020 in a more obvious way, making it easier to observe the differences.

2.2.3

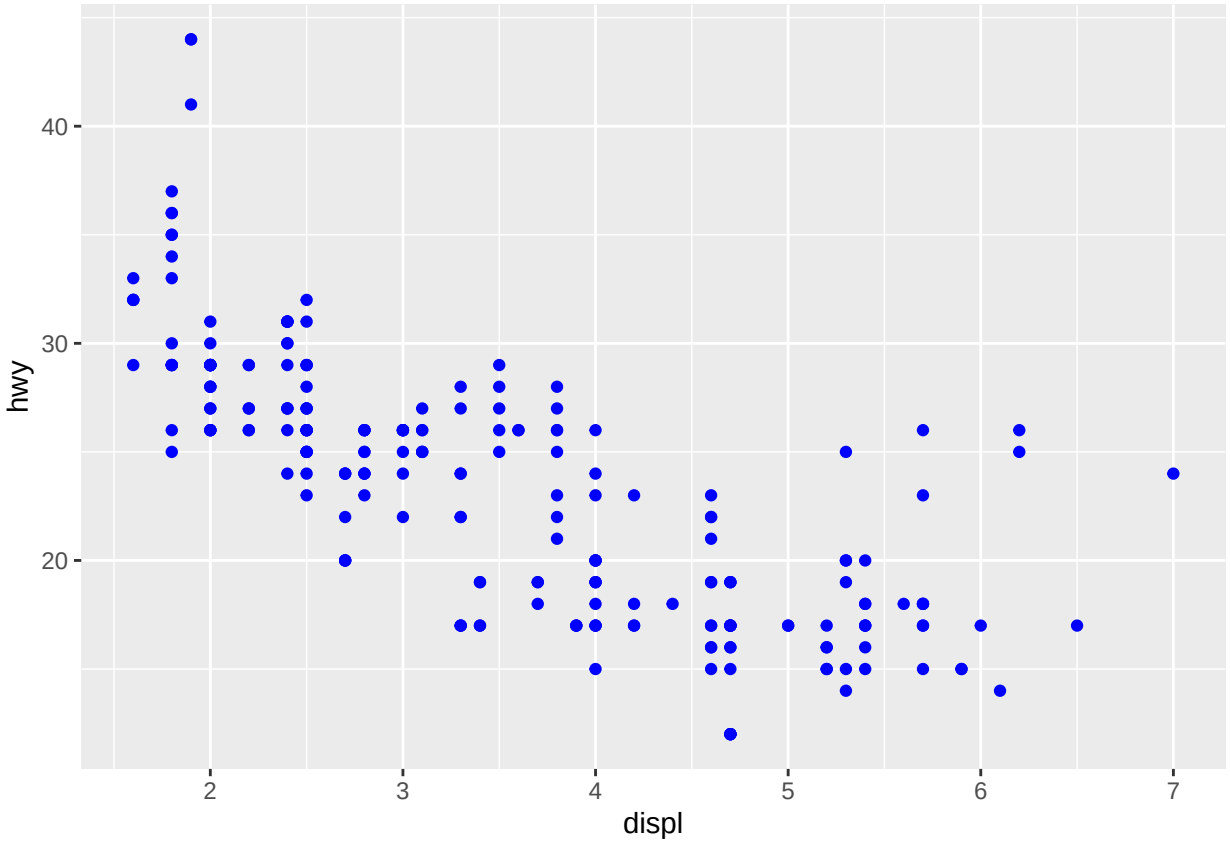
```
# Graph 2
ggplot(data = mpg) +
  geom_point(mapping = aes(x = hwy, y = cty, color = displ < 5))
```



Interpretation
The pattern of the plot remains the same, however, the legend will be classified as True and False(logical answers) instead of the exact value of displ. *True* represents observations which meet the requirement *displ* < 5, whereas *False* represents those have *displ* > 5

2.2.4

```
# Correct the code to show points in blue
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")
```

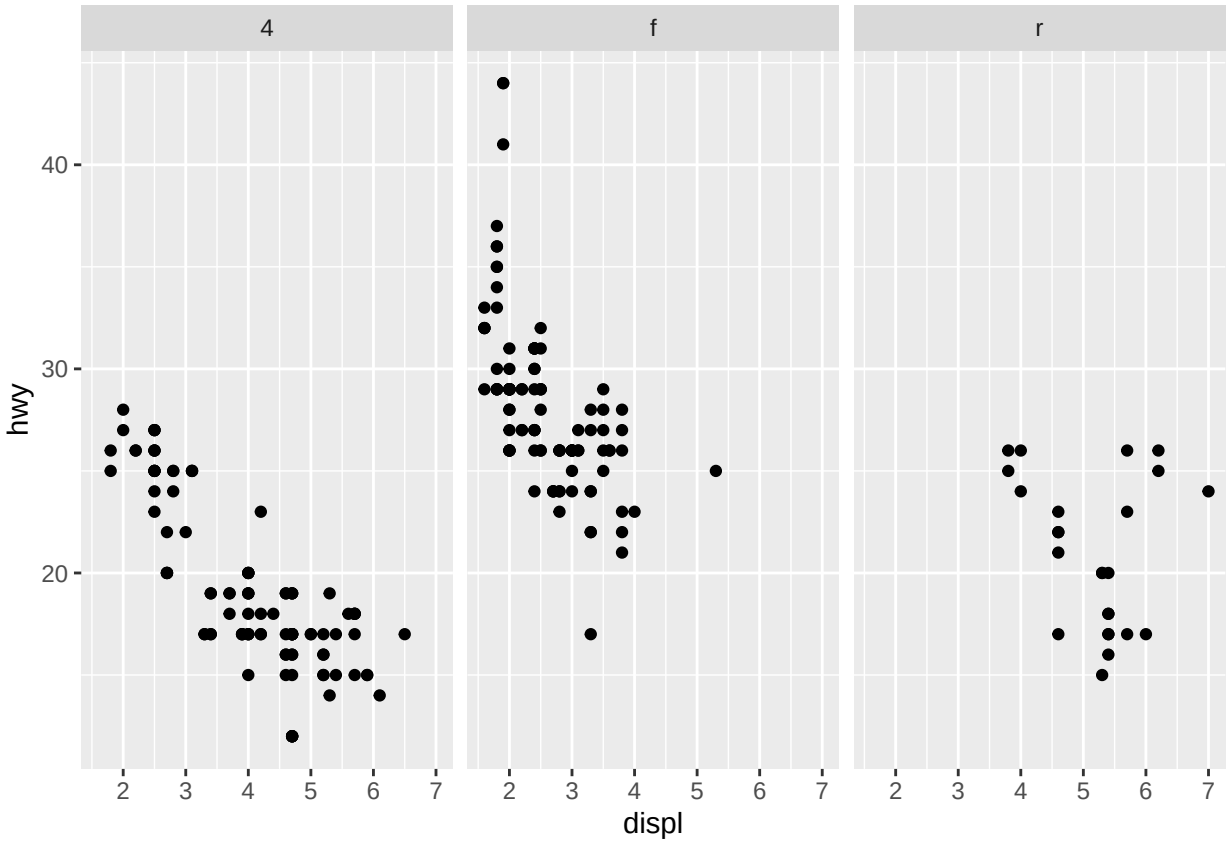


Interpretation
The argument *color = "blue"* is included within the aesthetic. The aesthetic will map a variable and a value, and *color = "blue"* will be treated as categorical, of which the expression *blue* is the only value. Therefore, we must write the *color* code outside the aes() function.

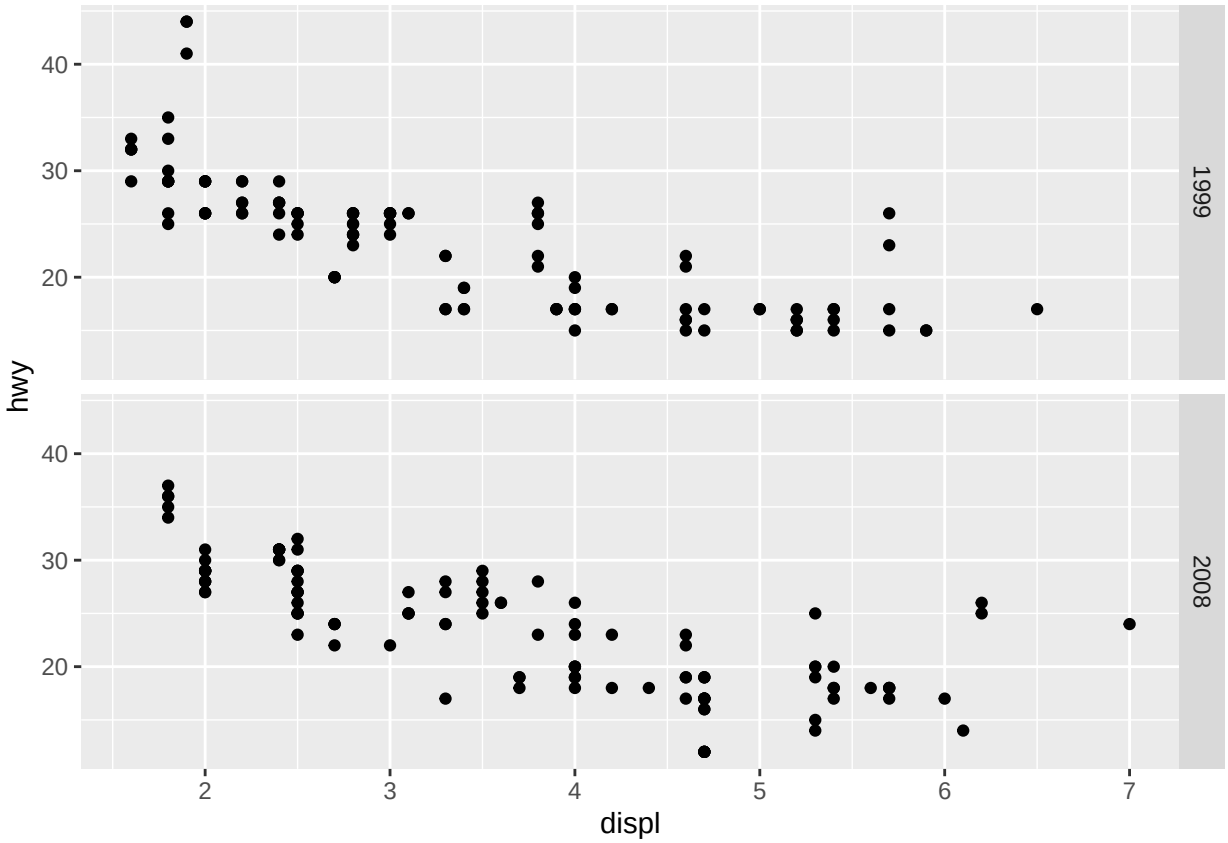
2.3

2.3.1

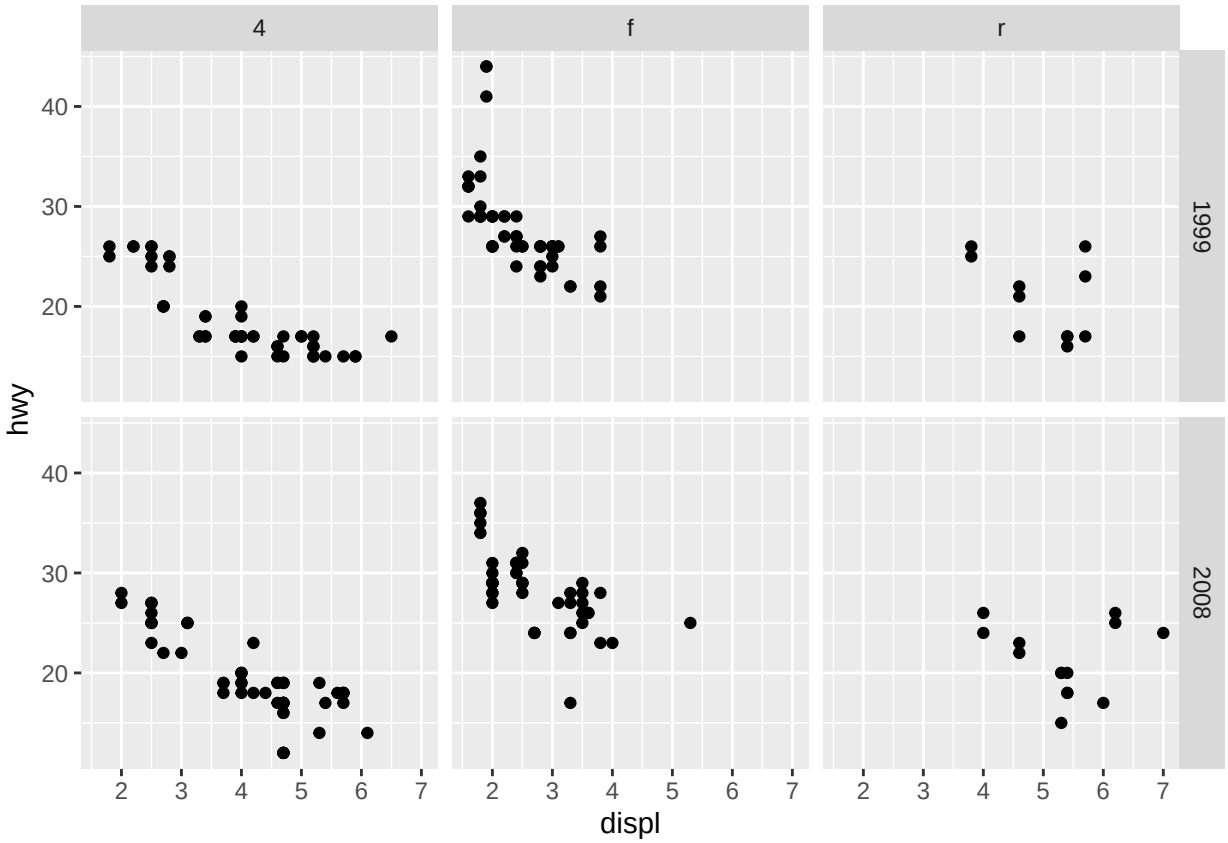
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(cols = vars(drv))
```




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



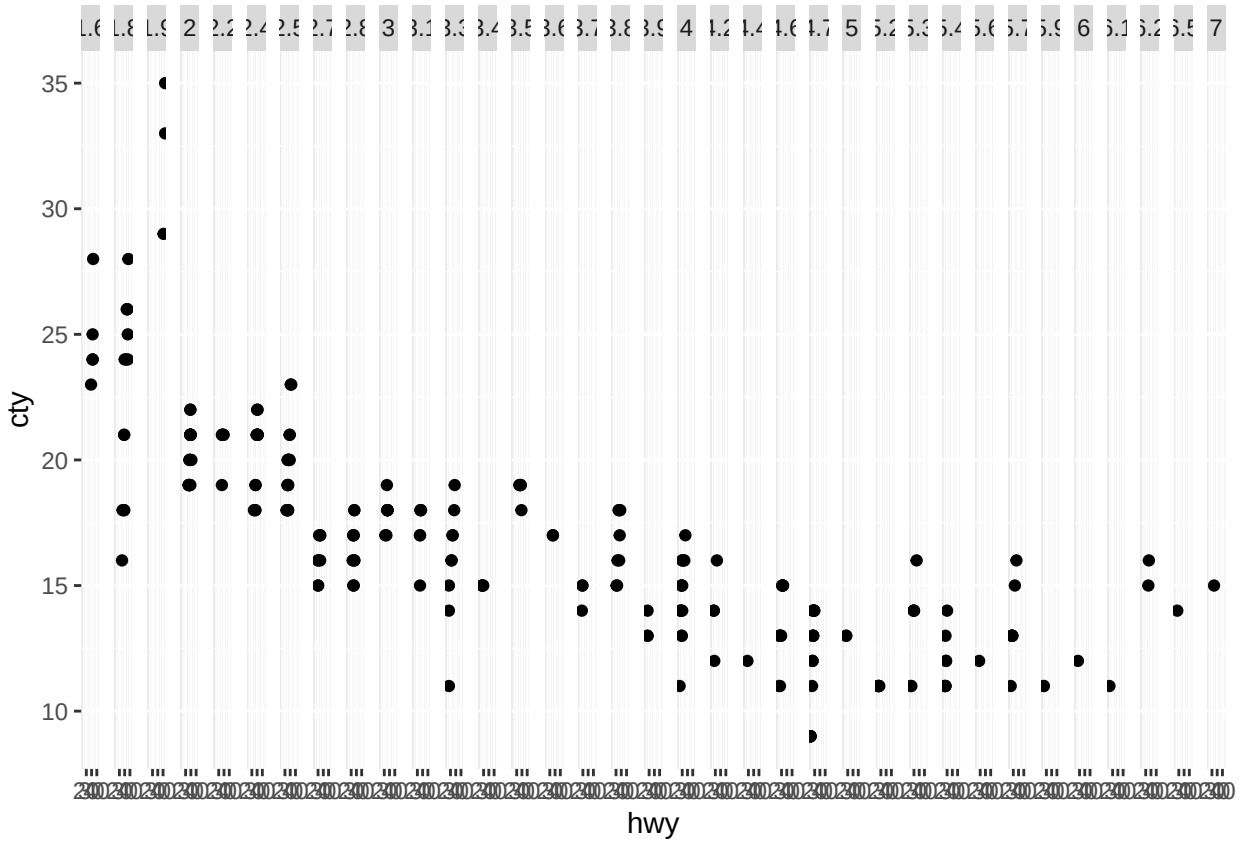

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(rows = vars(year), cols = vars(drv))
```



Interpretation
facet_grid() will subset the plot based on the variable within the bracket.We can further decide whether to subset the plot by rows or columns by adding functions within the bracket.

2.3.2

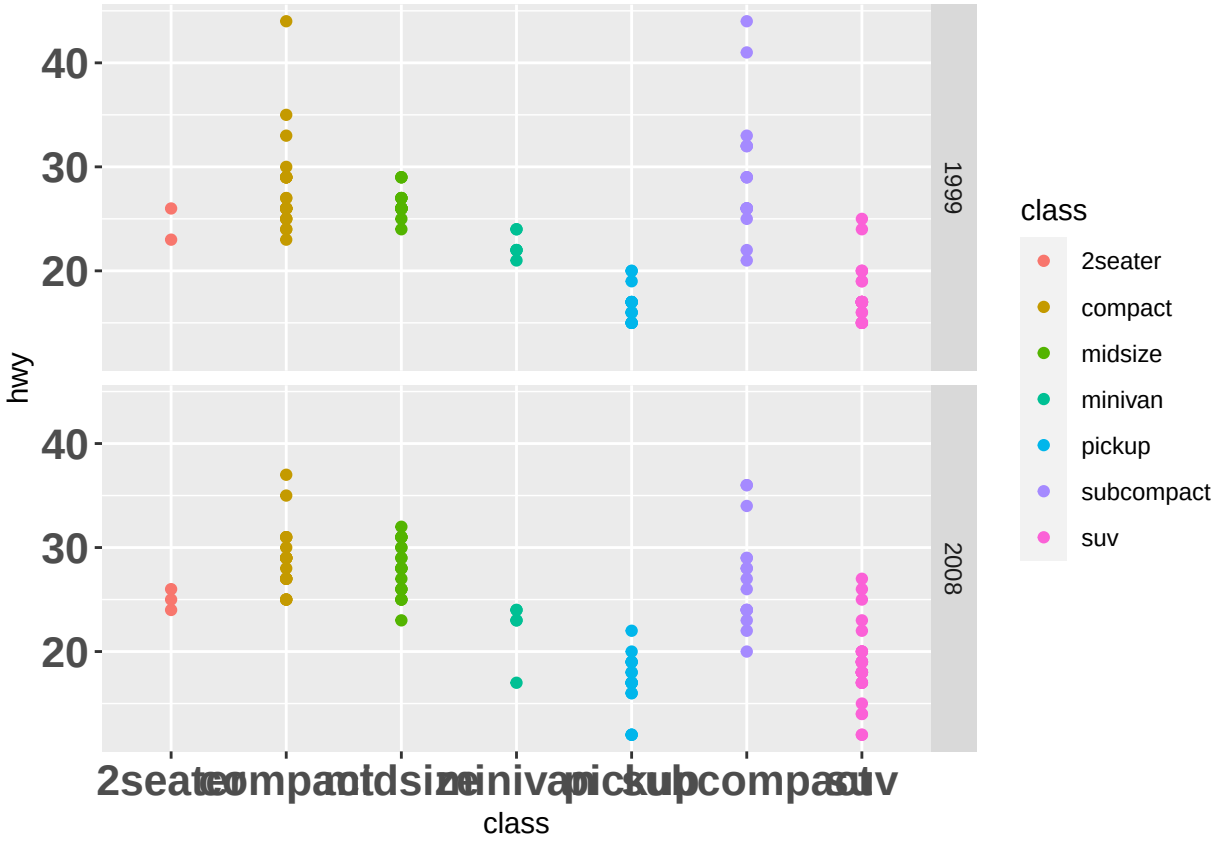
```
# Facet a continuous variable
ggplot(data = mpg) +
  geom_point(mapping = aes(x = hwy, y = cty)) +
  facet_grid(cols = vars(displ))
```



Interpretation
facet_grid() will convert the continuous variable into categorical variable, and plot each type of combination. However, since we're faceting based on a continuous variable, there will be too many combinations, making the plot unreadable.

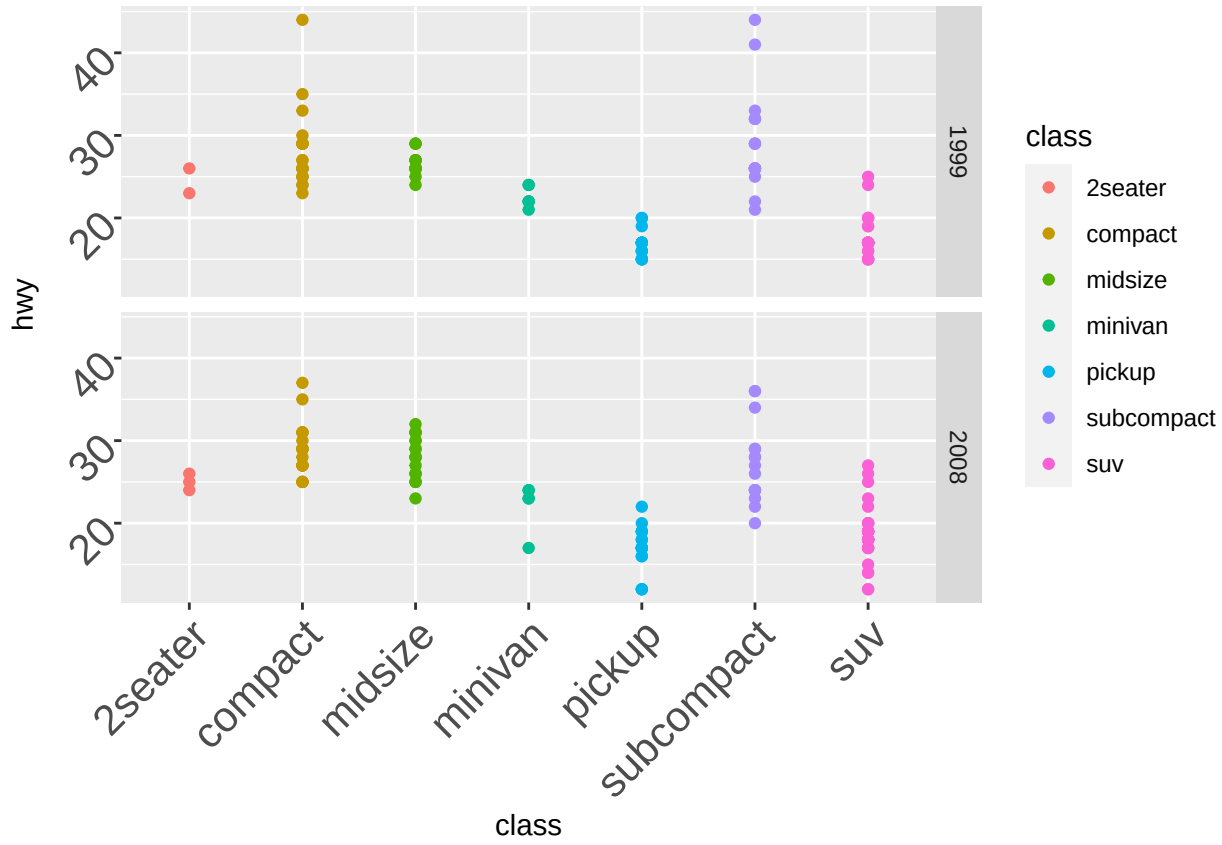
2.3.3

```
# Reproduce the graph
ggplot(data = mpg) +
  geom_point(mapping = aes(x = class, y = hwy,color = class)) +
  facet_grid(rows = vars(year))+
  theme(axis.text = element_text(size = 16,face = "bold"))
```



2.3.4

```
# Reproduce the graph
ggplot(data = mpg) +
  geom_point(mapping = aes(x = class, y = hwy,color = class)) +
  facet_grid(rows = vars(year))+
  theme(axis.text = element_text(size = 16,
                                 hjust = 1,angle = 45))
```



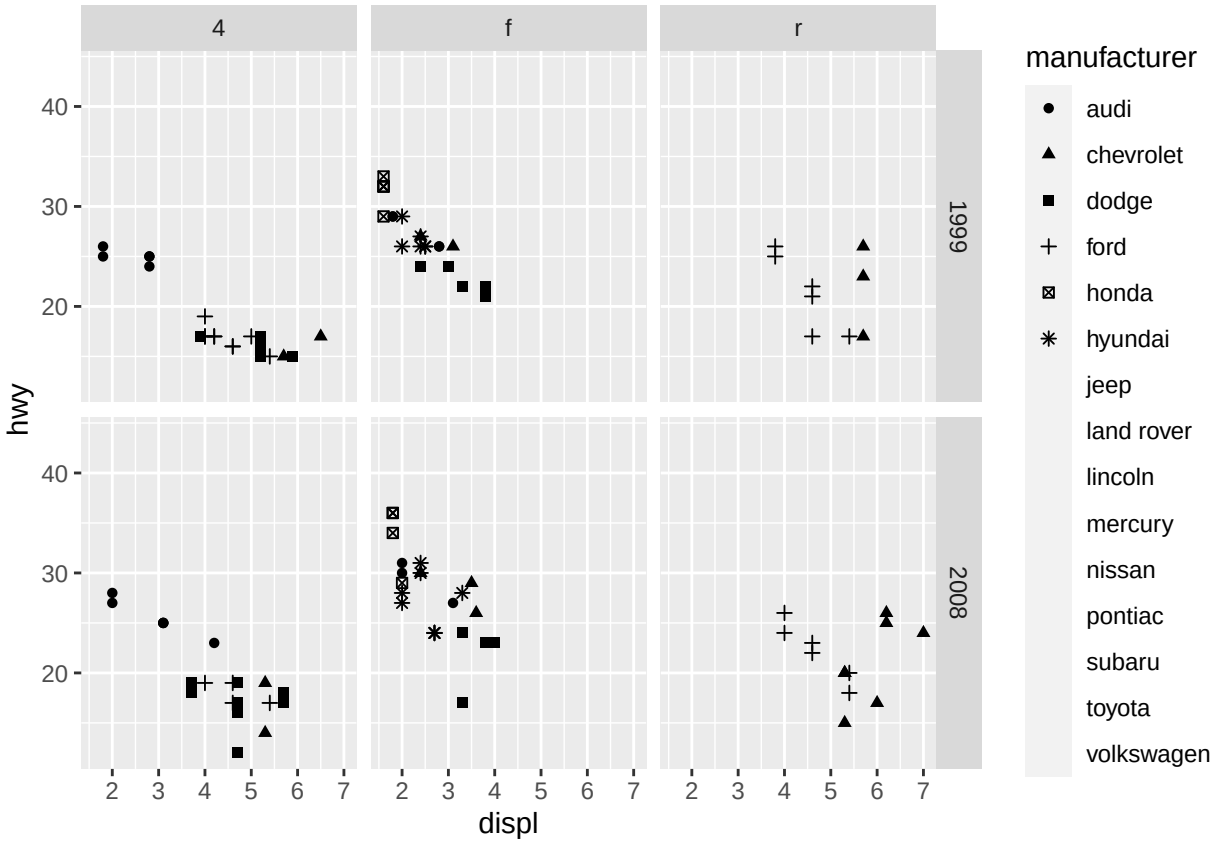

```
# I learned this before so there is no citation
```

2.3.5

```
# Reproduce graph
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy,shape = manufacturer)) +
  facet_grid(rows = vars(year),cols = vars(drv))
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 15. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 112 rows containing missing values (geom_point).
```



Interpretation
Because the shape palette can only deal with 6 discrete values at most. If we have more than 6 categories, it would be hard to distinguish the differences. In this case, we have 15 car manufacturers. Therefore, the graph only plots data of cars produced by "audi","chevrolet","dodge","ford","honda" or "hyundai", leading to 112 missing observations.
Reference: R help function

2.4 grammar of graphics:geoms

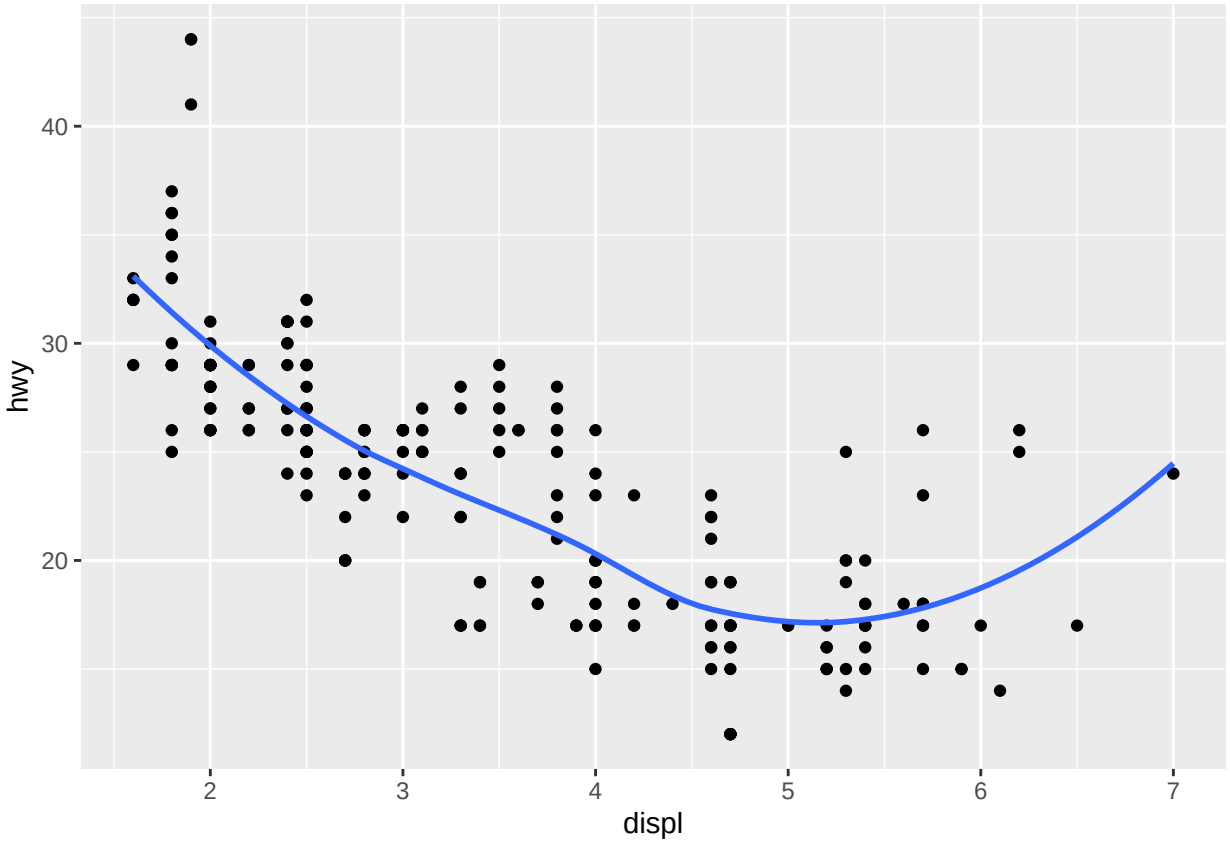
2.4 Q1

- Line chart: geom_line
- Boxplot: geom_boxplot
- Area chart: geom_area

2.4 Q2

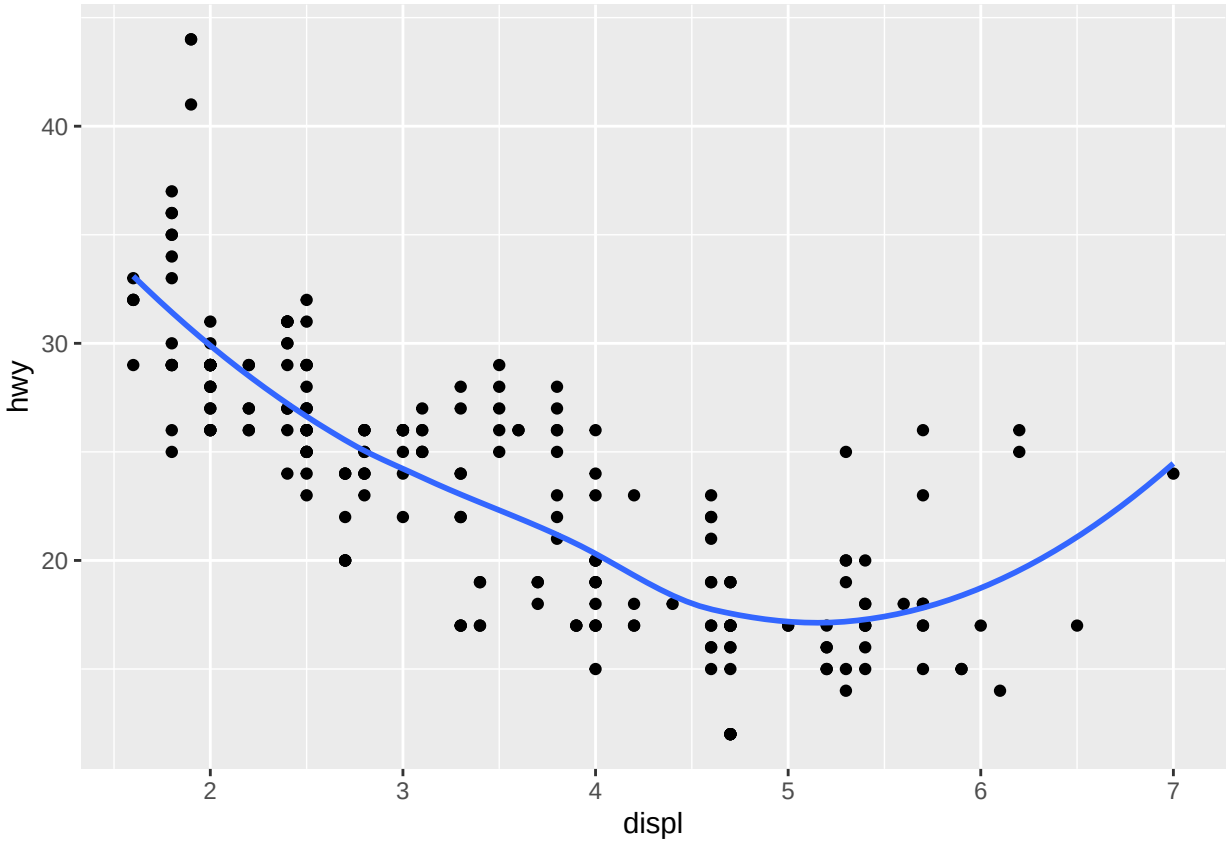
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(se = FALSE)
```

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




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

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

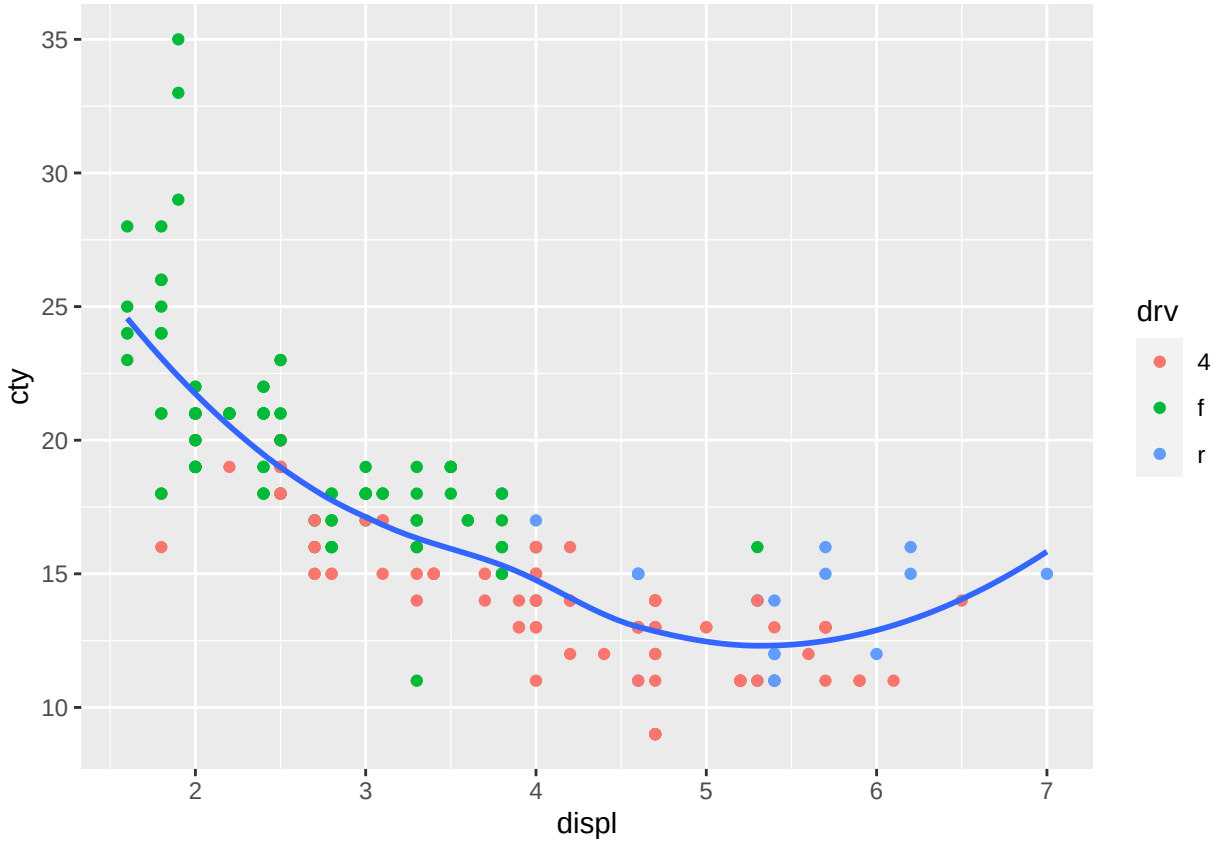


Interpretation
The two graphs look the same, because they're using the same data when plotting. In the first part of the code, geom_point() and geom_smooth() will inherit the information in the ggplot bracket, while in the second part of the code each of them include the same information of data within their own brackets. Therefore, the plot will look the same.

2.4 Q3

```
ggplot(data = mpg, mapping = aes(x = displ, y = cty)) +
  geom_point(aes(color = drv)) +
  geom_smooth(se = FALSE)
```

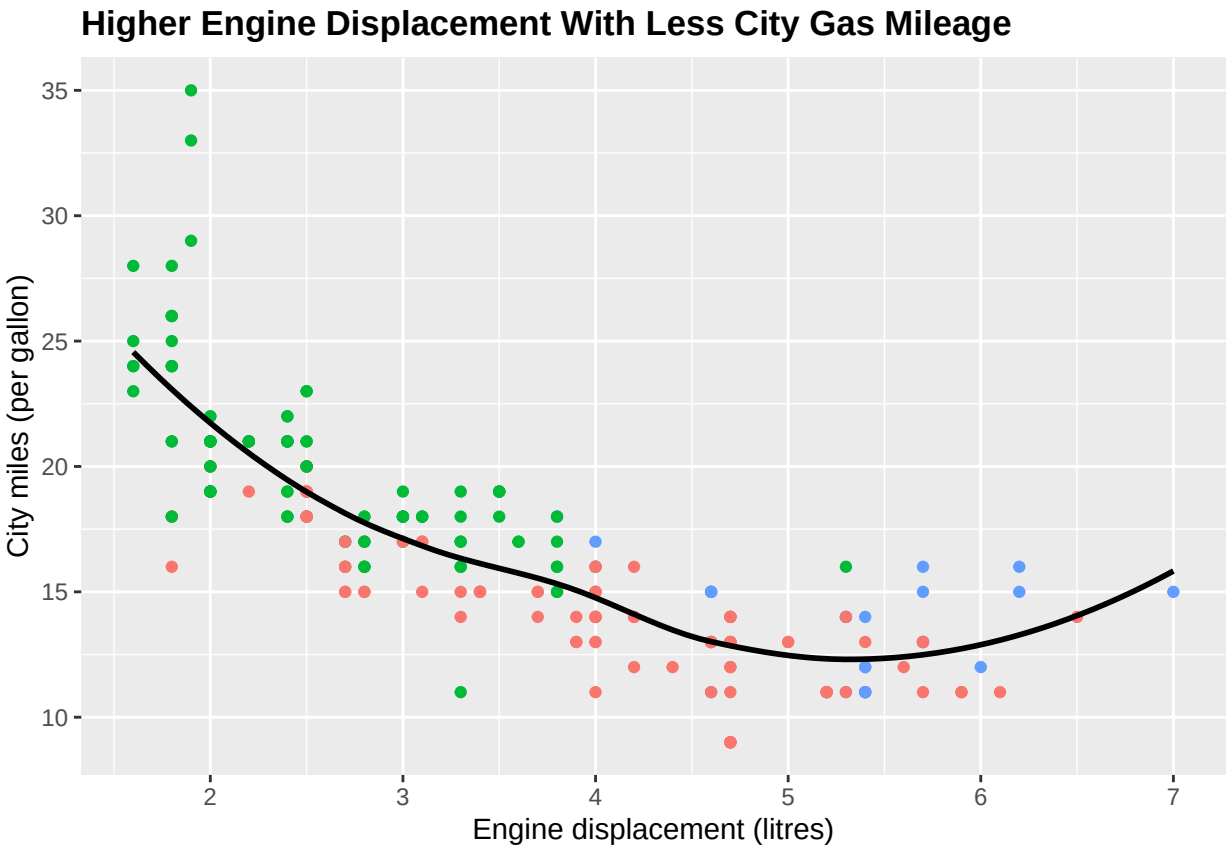
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



2.4 Q4

```
ggplot(data = mpg, mapping = aes(x = displ, y = cty)) +
  geom_point(aes(color = drv)) +
  geom_smooth(color = "black",se = FALSE) + # Change line color
  labs(title = "Higher Engine Displacement With Less City Gas Mileage", # Add title
       x = "Engine displacement (litres)", # Add information about x axis
       y = "City miles (per gallon)") + # Add information about y axis
  theme(plot.title = element_text(face = "bold"),
    legend.position = "none") # Remove legend
```

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



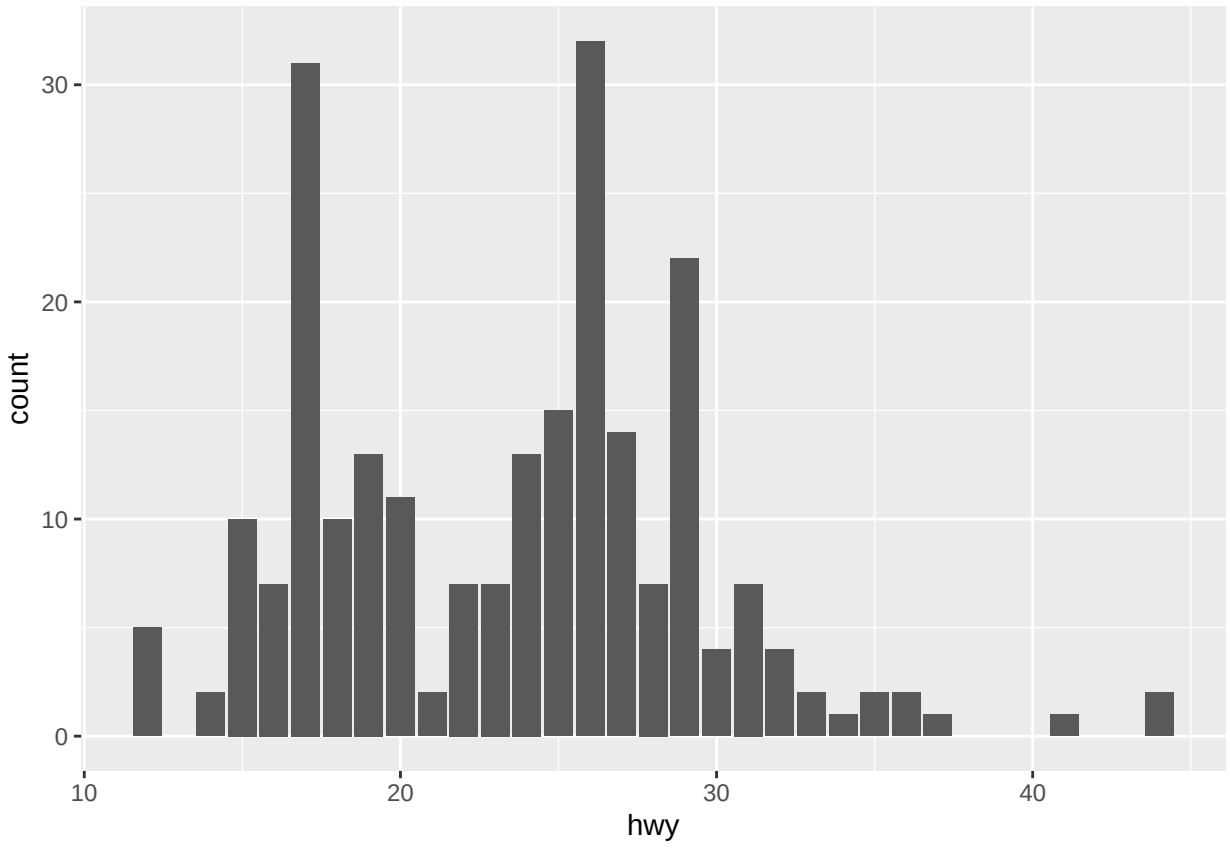
Interpretation
No. Removing legend will lead to confusion, because we cannot tell what the different colors represent. Other changes will make the plot more informative and readable.(The black line may cover some points but the same situation exists with lines in other colors. Compared to the blue line, black line is more recognizable since it's different from the legend color)

2.4.1.1

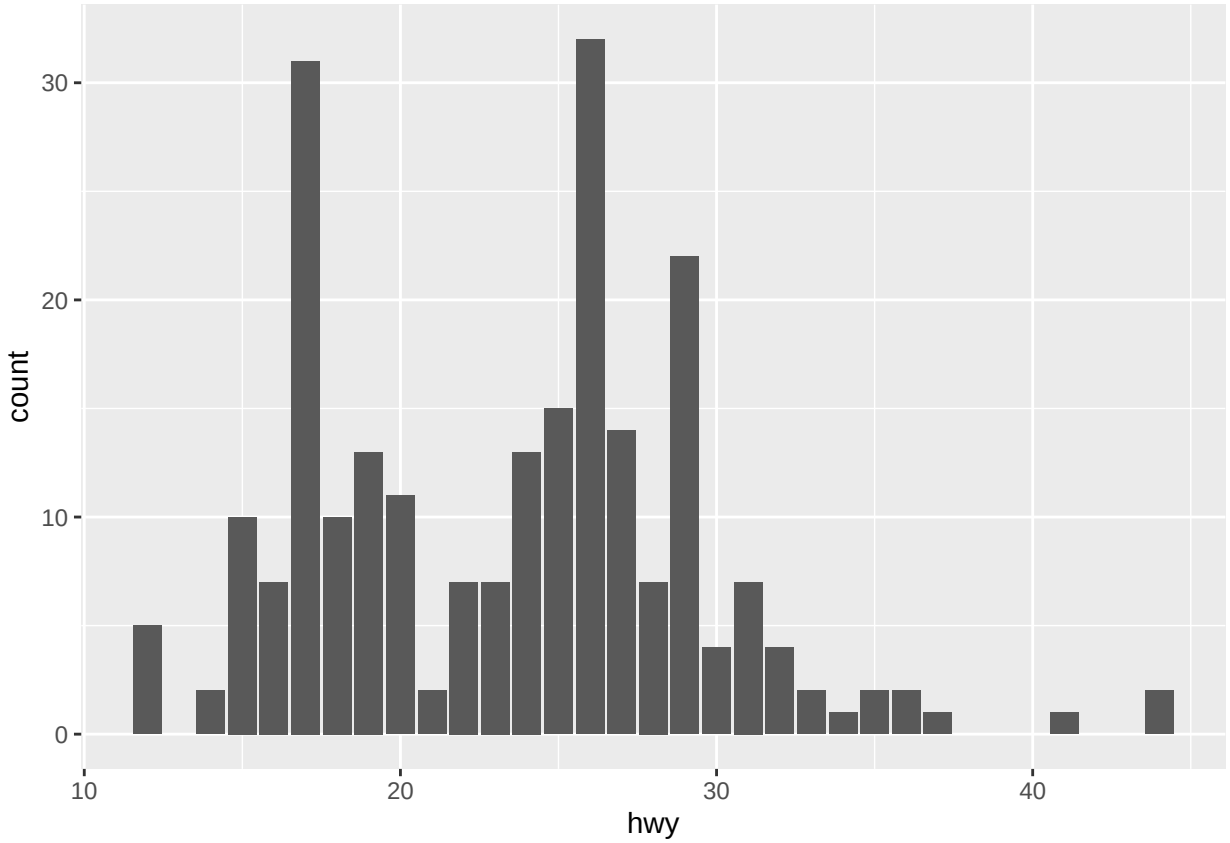
- geom_bar() is a kind of bar chart which makes the y values equal to the number of cases in each group. And the bar height is proportional to the number. geom_bar() uses stat_count() by default.
- geom_col() is also a kind of bar chart. However, the y axis of geom_col() plot represents the sum of the value in the data, and the heights of the bars also represent the true value in the data. geom_col() uses stat_identity() by default. *Reference: R Help Function*

2.4.1.2

```
ggplot(data=mpg, aes(x=hwy)) +
  geom_bar()
```




```
ggplot(data=mpg, aes(x=hwy)) +
  stat_count()
```



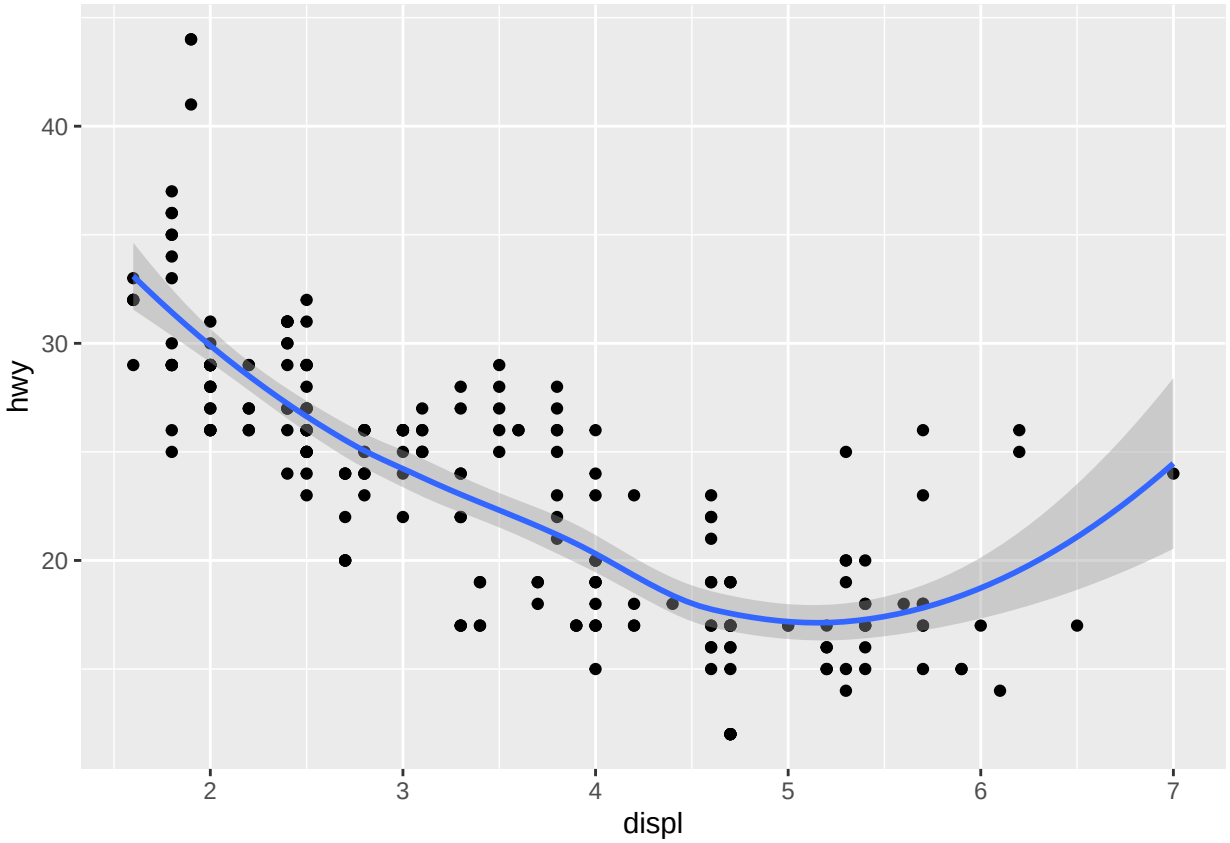
2.4.1.3

```
stat_smooth()
```

```
## geom_smooth: se = TRUE, na.rm = FALSE, orientation = NA
## stat_smooth: method = NULL, formula = NULL, se = TRUE, n = 80, fullrange = FALSE, level = 0.95, na.r
## position_identity
```

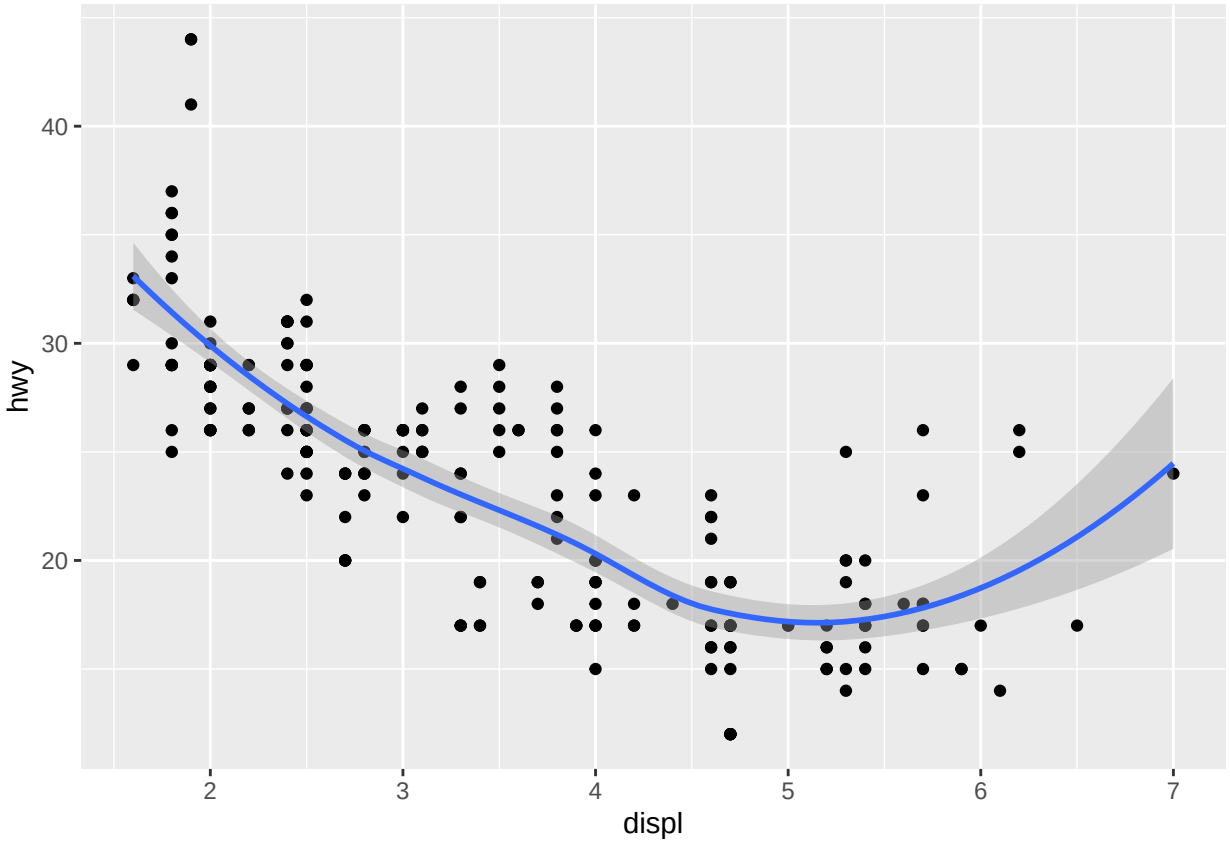
```
ggplot(data=mpg, aes(x=displ, y=hwy)) +
  geom_point() +
  stat_smooth()
```

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




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

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



Interpretation

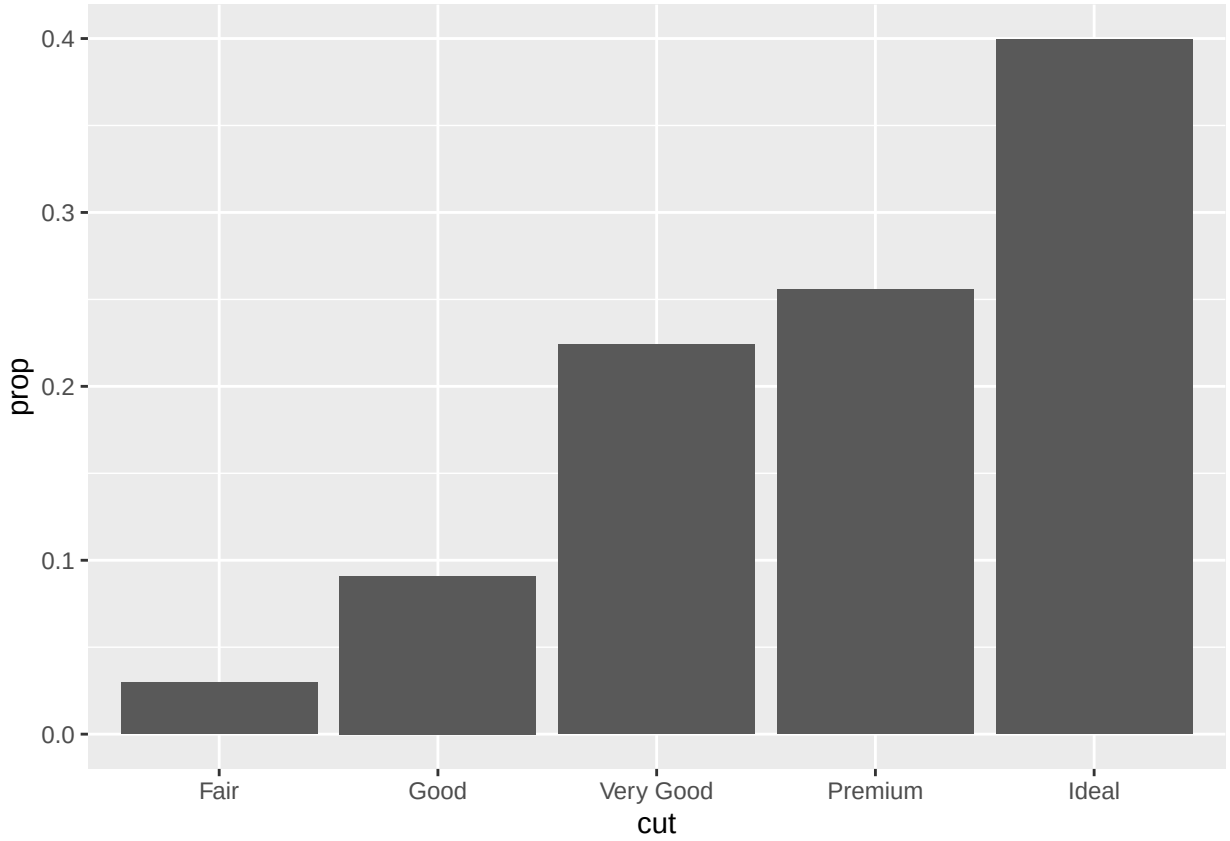
- stat_smooth() computes: Predicted value(y or x), lower pointwise confidence interval around the mean, upper pointwise confidence interval around the mean, standard error
- Those variables are displayed in the same way with geom_smooth(), as they come in pairs
- se(whether to display confidence interval around the smooth), method(smoothing method to use), formula(which formula to use given the method), na.rm(whether to remove missing value), color etc.

2.4.1.4

```
# geom_bar() & stat_count()
# geom_col() & stat_identity()
# geom_smooth() & stat_smooth()
# geom_density() & stat_density()
# geom_boxplot() & stat_boxplot()
```

2.4.1.5

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y = ..prop.., group = 1))
```



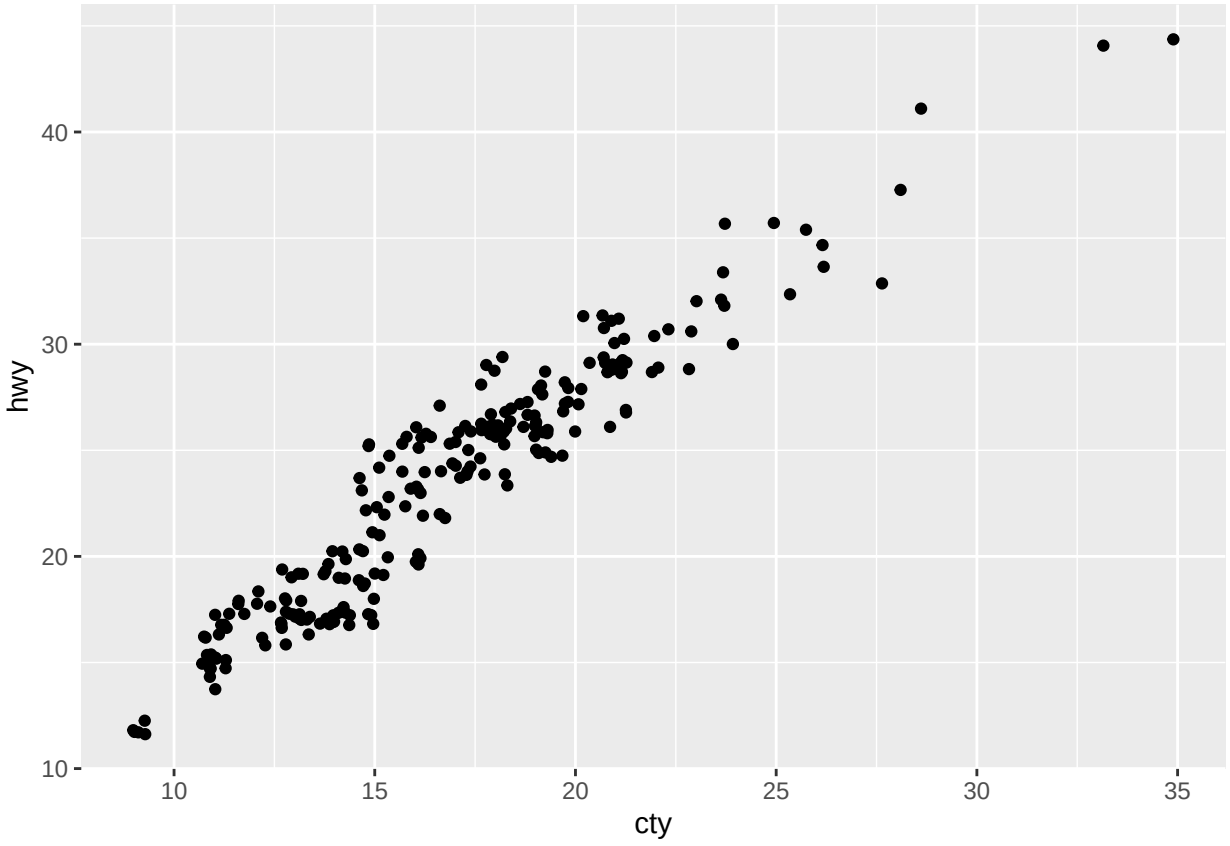
Interpretation
If group = 1 is not included, all bars will have the same height, which equals to the value of 1. This is because r will assume that groups are equal to x values by default(1 x category 1 group). And the total probability in each case will all equal to 1. Therefore, group = 1 is required when making proportion bar chart.

2.5

2.5.1

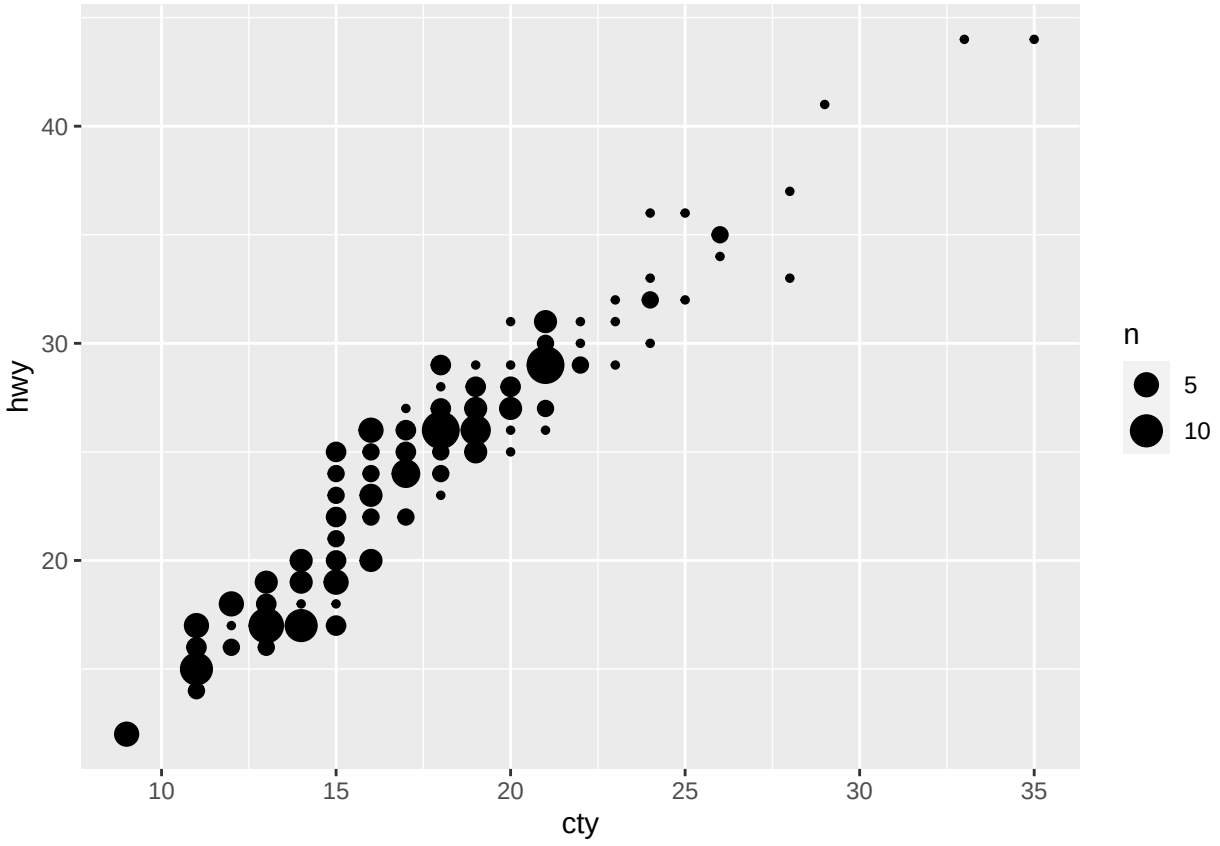
Large portion of data is missed in this plot because multiple observations share the same combination. In other words, there is overplotting.
We should use geom_jitter to correct this. In a geom_jitter plot, it adds random variation to the locations of the points in the graph, i.e, it "jitters" the locations of points slightly. This method reduces overplotting since two points with the same location are unlikely to have the same random variation. The variation could be horizontal or vertical or both by specifying width and height

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



2.5.2

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



Interpretation

- Both of them will make positional adjustment to the plot and make the plot indicate the number of cases in each combination.
- geom_count() adjust point sizes based the number of observations in the combination. The more observations, the larger the point size. The legend will also show the number of observations with different point size. However, if the points are too close, or the observations are large, overplotting will occur again.
- geom_jitter() will slightly change poin positions to resolve the overplotting problem. In some cases it will be better than geom_count
- Overall, both are not perfect solution to overplotting.geom_count()does not change the point position while will lead to potential overplotting problem; geom_jitter() will be more effective to avoid overplotting but it will make slight change to the point position and will not allow us to recognize the exact number of observations.
  Reference
  https://stackoverflow.com/questions/39255781/what-is-difference-between-geom-point-and-geom-jitter-in-simple-language-in-r

2.5.3

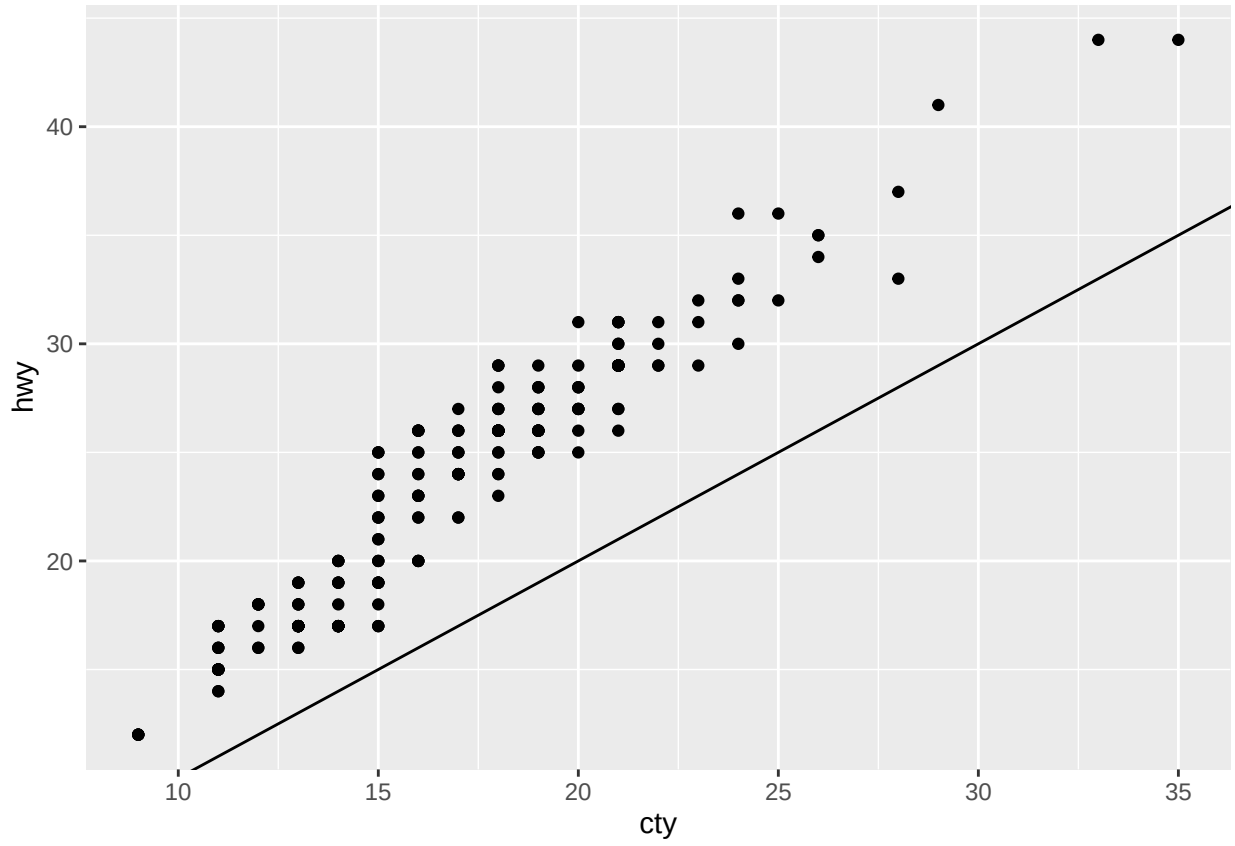
The default setting is position_stack, which means if multiple bars occupying the same position(group) then R will stack them atop of one another. We use position_dodge() to dodge overlapping bars side by side, or use position_fill() to stack bars in a way to show the proportional values of each category within each x

group, and standardize each bar to have same heights. Adding position_fill() will generate a plot with 1 bar for each x group.
Reference: R help function

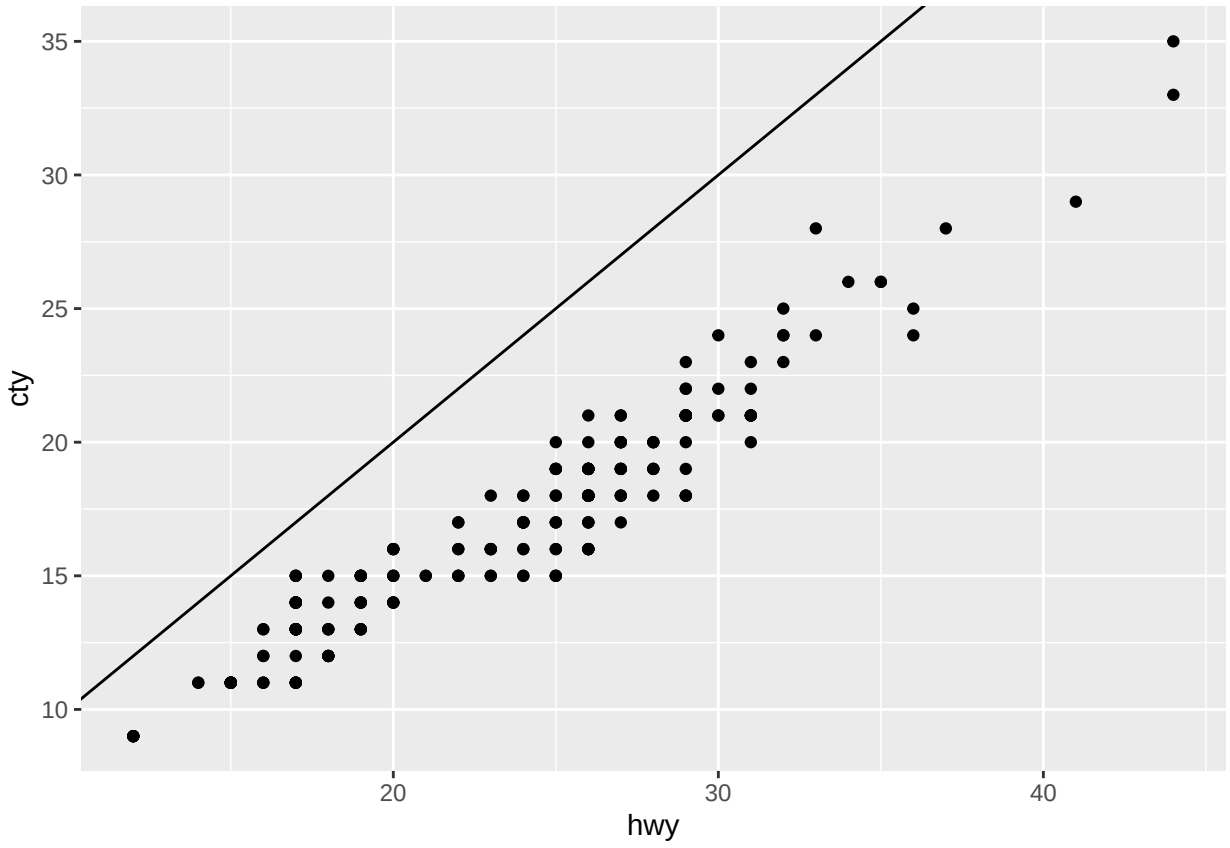
2.6

2.6.1

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



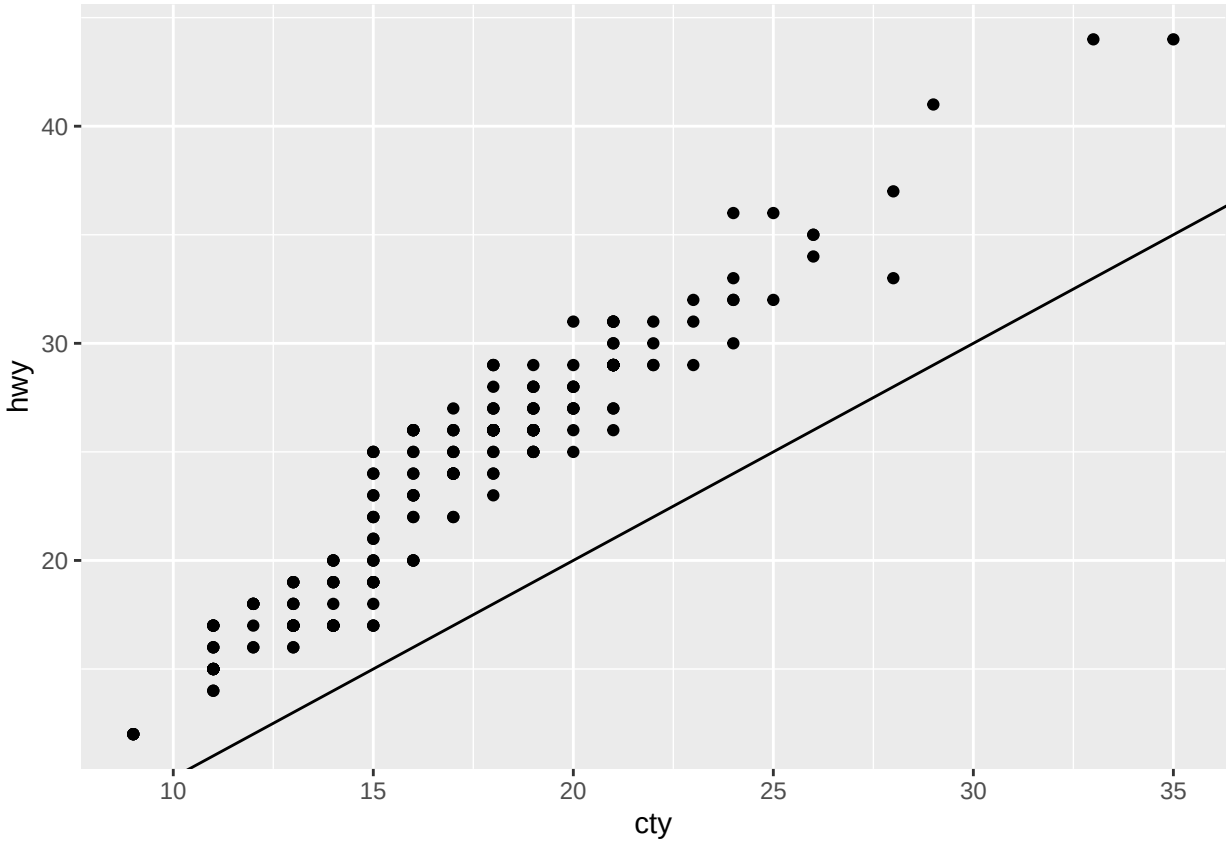

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline() +
  coord_flip()
```



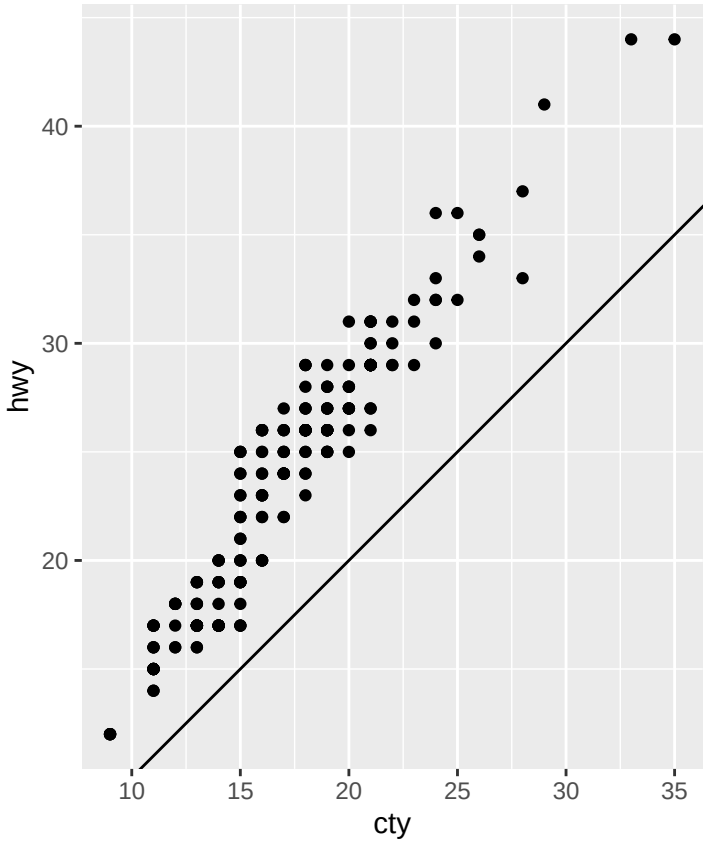
Interpretation
If we use coord_flip(), the x axis and y axis will be switched, i.e., the initial x variable now becomes the y variable, and y variable becomes the x variable.

2.6.2

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




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



Interpretation
The plot shows that there is a positive correlation between city and highway mpg. The more city miles per gallon, we would expect the more highway miles per gallon.

- geom_abline() creates a reference line for the plot. The line could be vertical, horizontal or specified with slope and intercept
- coord_fixed() ensures that the reference line is at 45-degree angle, which makes it easier for us to compare the differences between observed hwy and the case that hwy equal to cty, given cty constant