Overview

This assignment steps through programming syntax with `ggplot2` and introduces the formula interface with the `lm()` function. The grammar of graphics can help create meaningful visualizations which communicate Exploratory Data Analysis (EDA) findings. Visualizations will provide context to the model results, which might be opaque and difficult to interpret. In this assignment you will demonstrate basic syntax, go through a detailed EDA of a common data set, and then begin to work through the syntax of fitting a model.

If you need help with understanding the R syntax please see the R4DS book and/or the R tutorial videos and demos available on the Canvas site.

IMPORTANT!!!

Certain code chunks are created for you. Each code chunk has `eval=FALSE` set in the chunk options. You **MUST** change it to be `eval=TRUE` in order for the code chunks to be evaluated when rendering the document.

You are free to add more code chunks if you would like.

Load packages

The `tidyverse` is loaded in for you in the code chunk below. The visualization package, `ggplot2`, and the data manipulation package, `dplyr`, are part of the "larger" `tidyverse`.

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

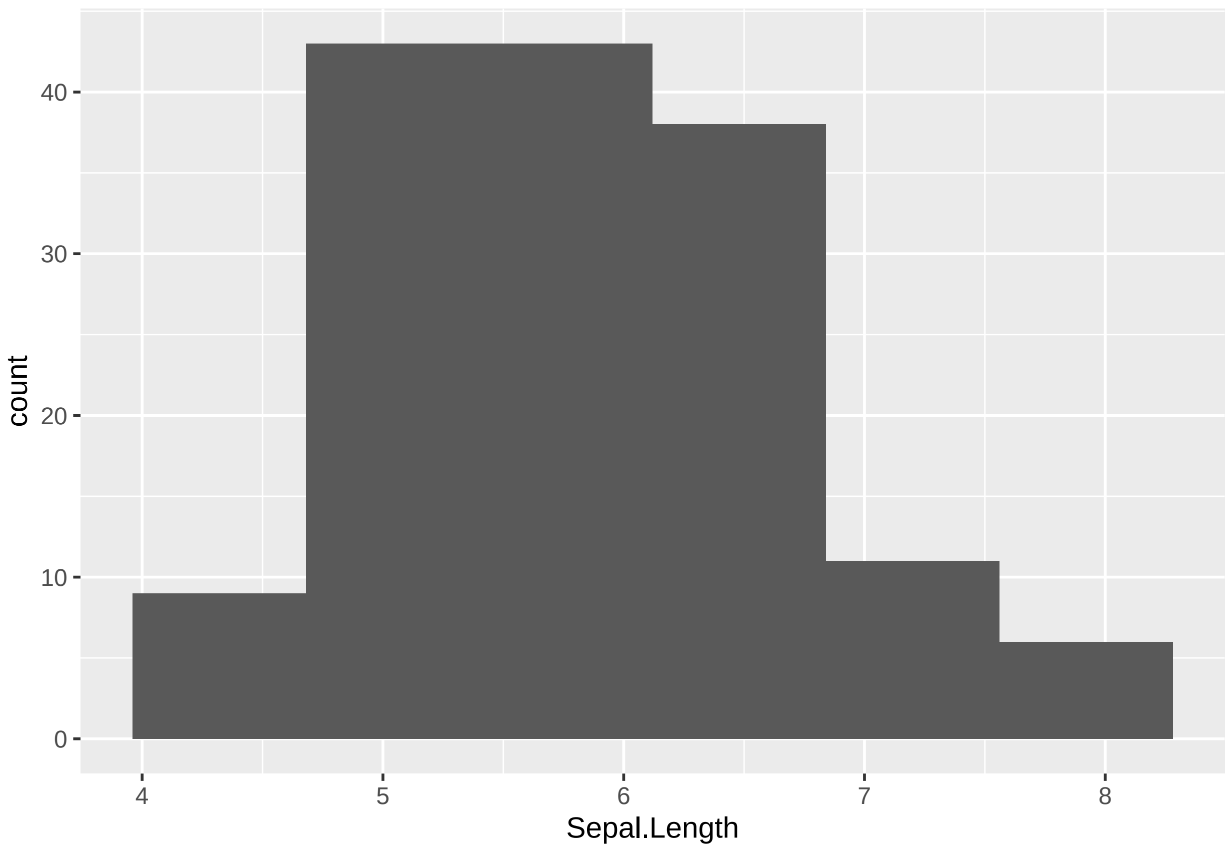
Problem 01

This problem introduces key syntax of the grammar of graphics in `ggplot2` with the `iris` data set. The `simple_r_intro.html` available on Canvas demonstrates visualizations in `ggplot2` with the `iris`. Going through that document will help with this problem.

1a)

Create a histogram for `Sepal.Length` from `iris` with 6 bins.

```r
iris %>%
  ggplot(mapping = aes(x = Sepal.Length)) +
  geom_histogram(bins = 6)
```



SOLUTION

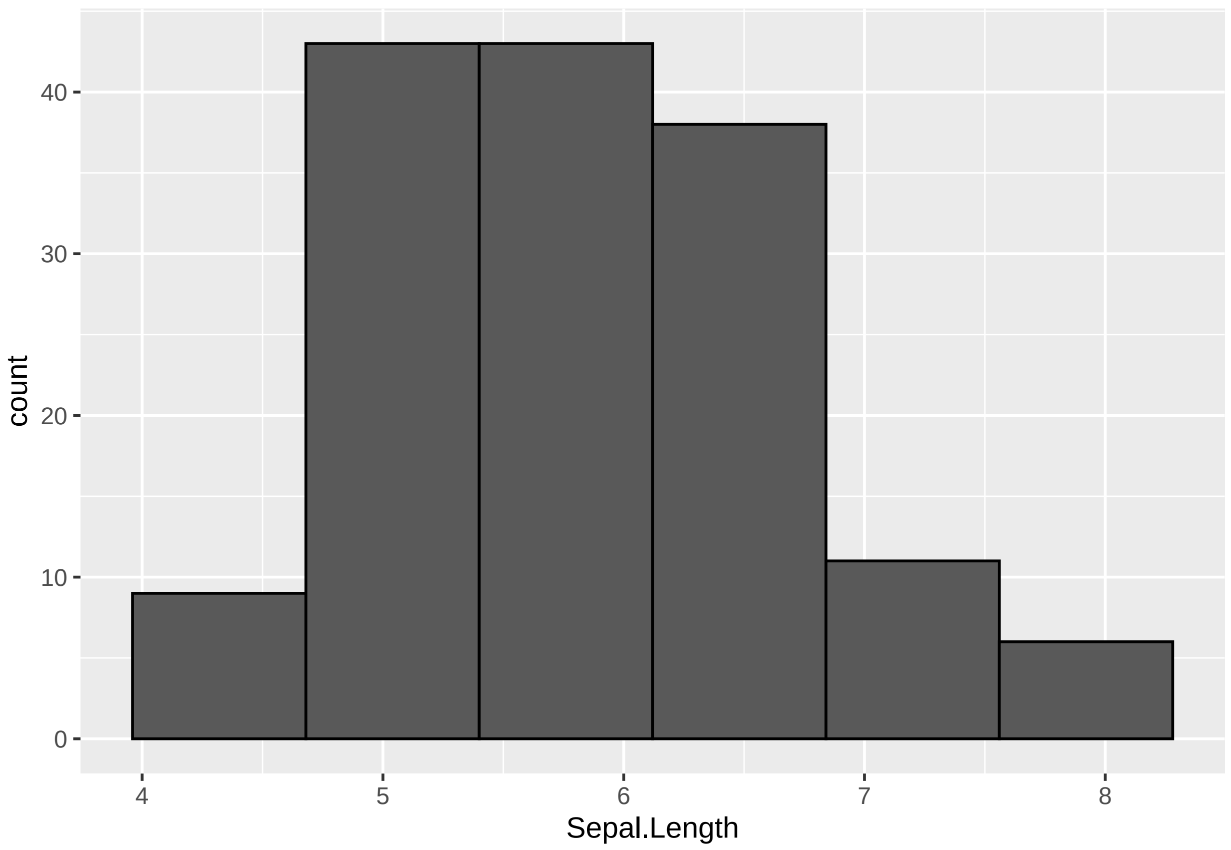
1b)

By default, a `ggplot2` histogram does not show the lines associated with each bin (as in a bar graph). The histogram effectively looks like a discretized distribution. To adjust this, we need to override the default `fill` and `color` arguments to the `geom_histogram()` function. Note that within `ggplot2`, color is applied to line-like objects and points while fill is applied to whole areas of the graph (think "fill in an area"). Thus, you can have substantial control over how color is used to visually present information within a graphic.

Even though an aesthetic can be linked to a variable, some aesthetics can be modified "manually" and not associated with any variables within the dataset. We use the same argument, but we set that argument **outside** of the `aes()` function. Thus, we assign arguments **within** the `aes()` function when we want to *link* or *map* the aesthetic to a variable (column in the data set). Conversely, we assign the aesthetic argument **outside** the `aes()` function when we want to "hard code" or "manually set" an aesthetic and therefore **not** associate the aesthetic with a variable.

To see how this works, type `color = "black"` within the `geom_histogram()` call. Be careful about your commas!

```
iris %>%
  ggplot(mapping = aes(x = Sepal.Length)) +
  geom_histogram(color = "black",bins = 6)
```



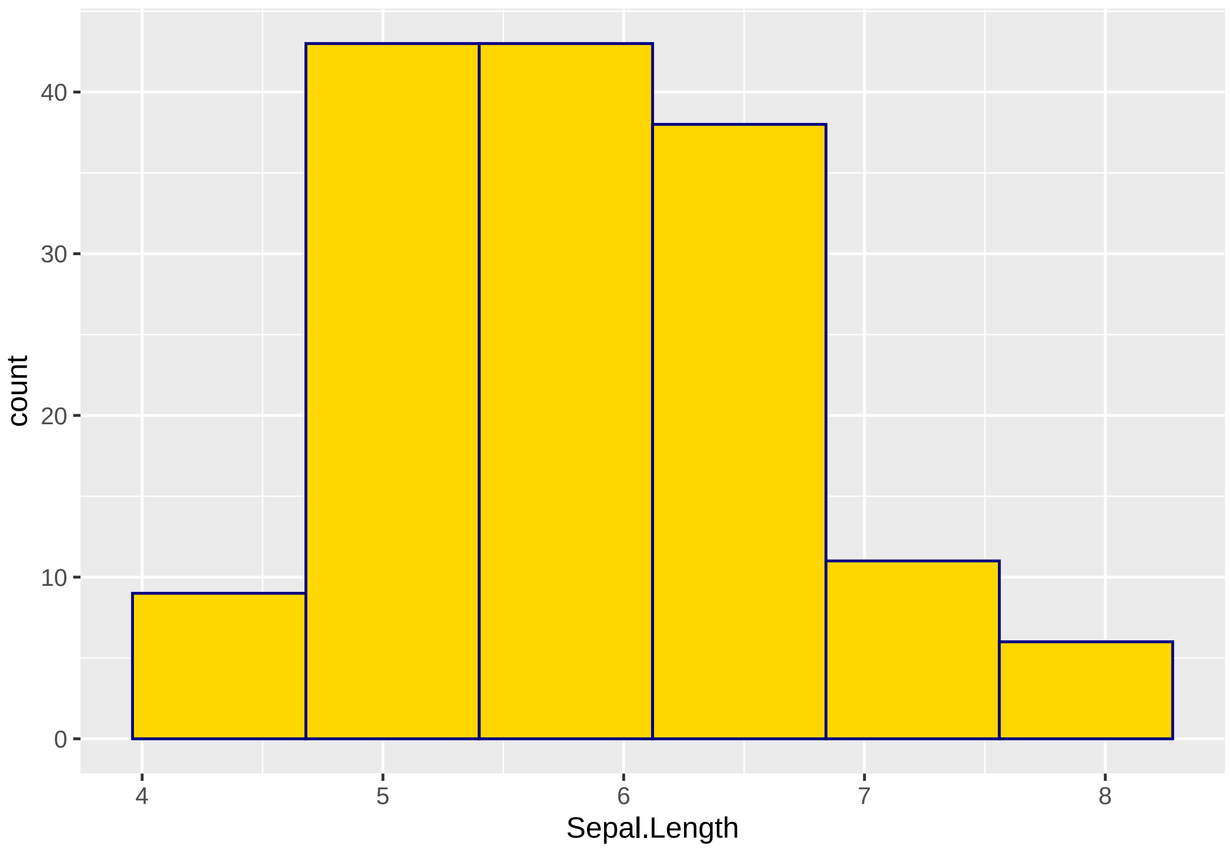
SOLUTION

1c)

`ggplot2` has many "named" colors available for use. You can even use hex codes if you want to finely control the colors in your plots. In this course, we will typically stick with common colors when we manually pick a color and/or fill.

To make the difference between color and fill explicit within the histogram, change the color to `color = "navyblue"` and modify the histogram's fill by setting `fill = "gold"`.

```
iris %>%
  ggplot(mapping = aes(x = Sepal.Length)) +
  geom_histogram(color = "navyblue",fill = "gold",bins = 6)
```



SOLUTION

1d)

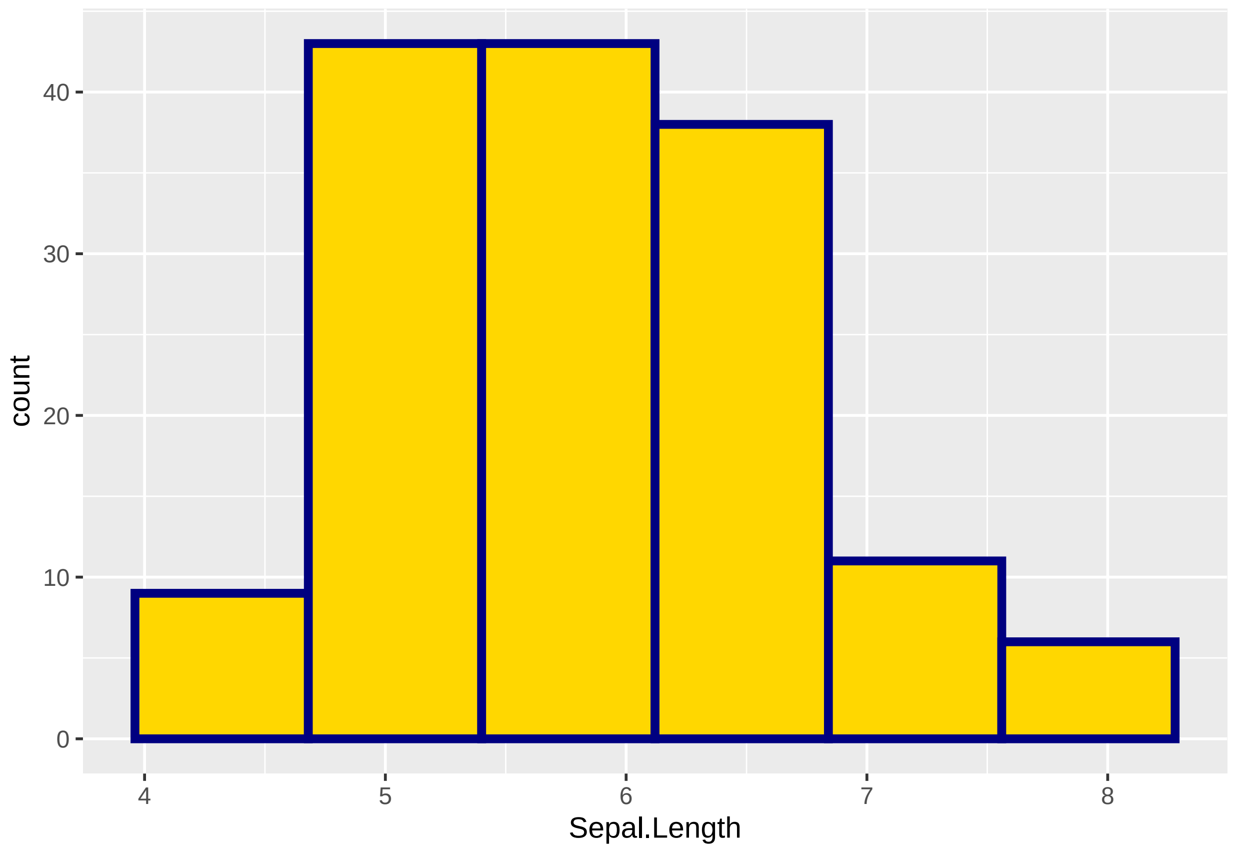
We can alter the size or thickness of the lines around each bin with the `size` argument.

Set `size = 1.55` within the `geom_histogram()` call (using the same color scheme from Problem 1c)).

```
iris %>%
  ggplot(mapping = aes(x = Sepal.Length)) +
  geom_histogram(color = "navyblue",fill = "gold",size = 1.55,bins = 6)
```

SOLUTION

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

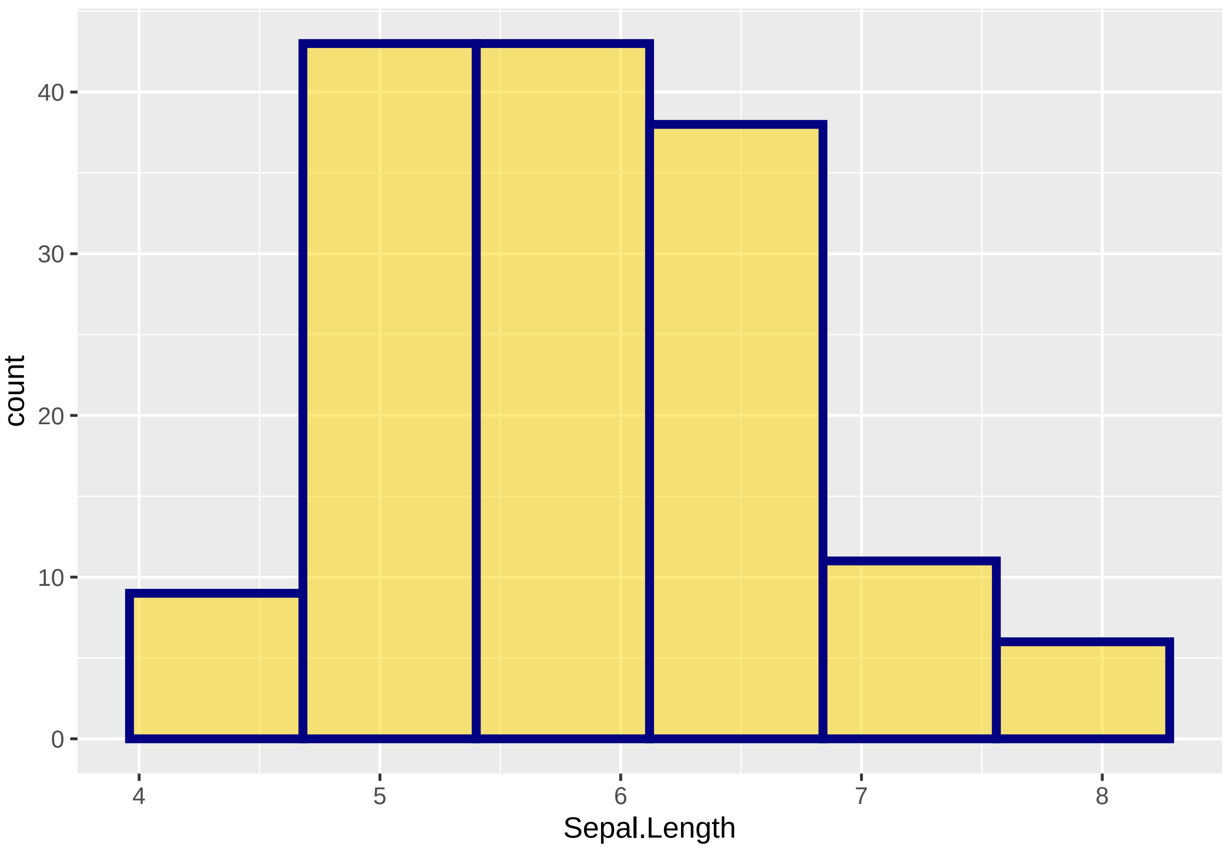


1e)

Lastly, the transparency of geometric objects can be altered with the `alpha` argument.

Set the transparency to `alpha = 0.5` within the `geom_histogram()` call.

```r
iris %>%
  ggplot(mapping = aes(x = Sepal.Length)) +
  geom_histogram(color = "navyblue",fill = "gold",size = 1.55,alpha = 0.5,bins = 6)
```



SOLUTION

Problem 02

In this problem, we will introduce another very important geom, the boxplot, which provides a quick visual display of useful summary statistics for continuous variables (`"numeric"`s). The boxplot is particularly useful at visualizing the relationship between a continuous variable with a categorical variable because separate boxplots are displayed for each category (level) of the categorical variable.

Compared with the histogram which focuses on displaying the *shape* of the distribution, the boxplot summarizes a distribution with important **quantiles** to show central behavior and variability. Therefore, the boxplot provides a quick view of central tendency of the variable, as well as a rough guide for the "meaningful" range.

To demonstrate the usefulness of the boxplot, we will use the `diamonds` dataset from `ggplot2`.

2a)

Pipe `diamonds` into the `glimpse()` function to display the dimensions and datatypes associated with the variables within the dataset.

```
glimpse(diamonds)
```

SOLUTION

```
## Rows: 53,940
## Columns: 10
## $ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23, 0.~
## $ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good, Ver~
## $ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J, I,~
## $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1, ~
## $ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4, 64~
## $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 58~
## $ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340, 34~
## $ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, 4.00, 4.~
## $ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, 4.05, 4.~
## $ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, 2.39, 2.~
```

2b)

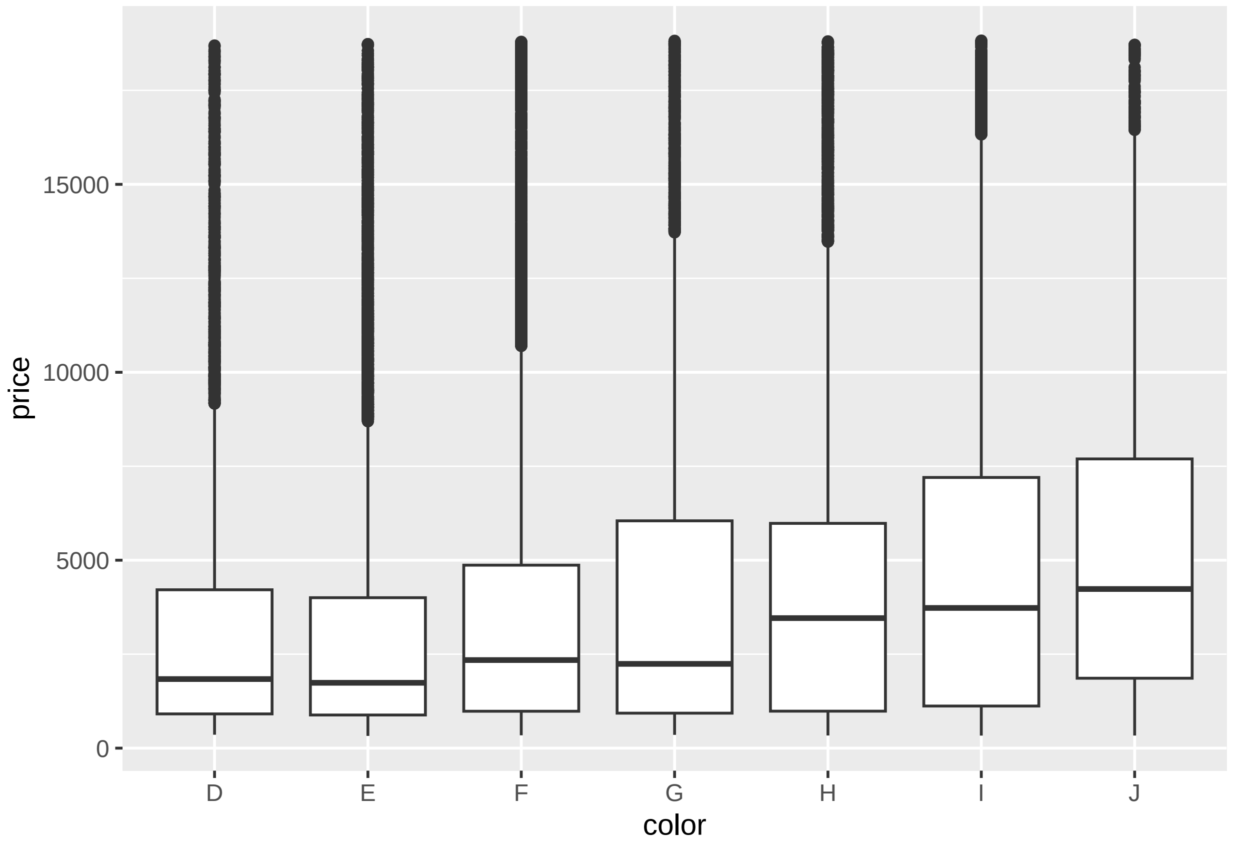
The variable `price` gives the cost of a diamond in US dollars, while the other variables provide attributes associated with a diamond. A natural question to ask is: *What variables influence the price?* If you have purchased a diamond before, you may have heard about the "4 C's": cut, color, clarity, and carat. We will explore the behavior of `price` with respect to these 4 variables.

The first three of these are all factors (the `"ord"` data type is a special ordered factor which as the name states has a natural ordering of the categorical levels) within the `diamonds` dataset, while `carat` is a `"numeric"` variable. With a boxplot, we group a continuous variable based on the categories of a categorical variable, and display summary statistics associated with the continuous variable for each categorical level. Therefore, the boxplot geom is another geometric object which performs multiple operations behind the scenes. If you have not worked boxplots before, I recommend Chapter 7 of the R for Data Science book.

Let's start by visualizing the relationship between `price` and `color`.

Pipe `diamonds` into `ggplot()` and set the `x` and `y` aesthetics to `color` and `price`, respectively. Then, call `geom_boxplot()`. What conclusion would you draw based on the resulting figure?

```r
ggplot(diamonds, aes(x = color, y = price)) +
  geom_boxplot()
```



SOLUTION

Conclusions:

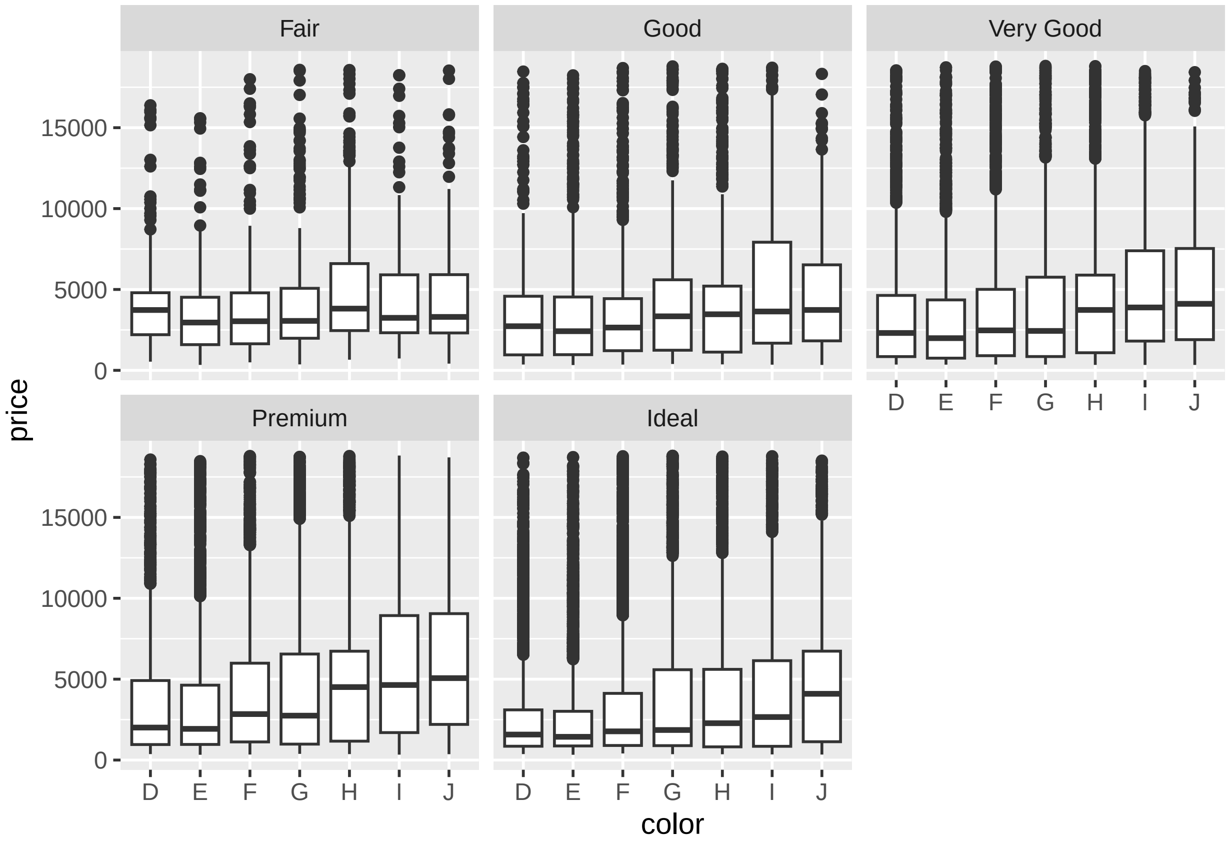
1) E color diamonds are cheaper because first quartile, median and third quartile values are lower in its plot.
2) J color diamonds are more expensive because first quartile, median and third quartile values are higher in its plot.

2c)

Next, we will include the influence of the `cut` variable breaking up the graphic into separate subplots based on the levels of `cut`.

Add the `facet_wrap()` call to the code from Problem 2b), and set the facetting variable to be `cut`.

```
ggplot(diamonds, aes(x = color, y = price)) +
  geom_boxplot() +
  facet_wrap(vars(cut))
```



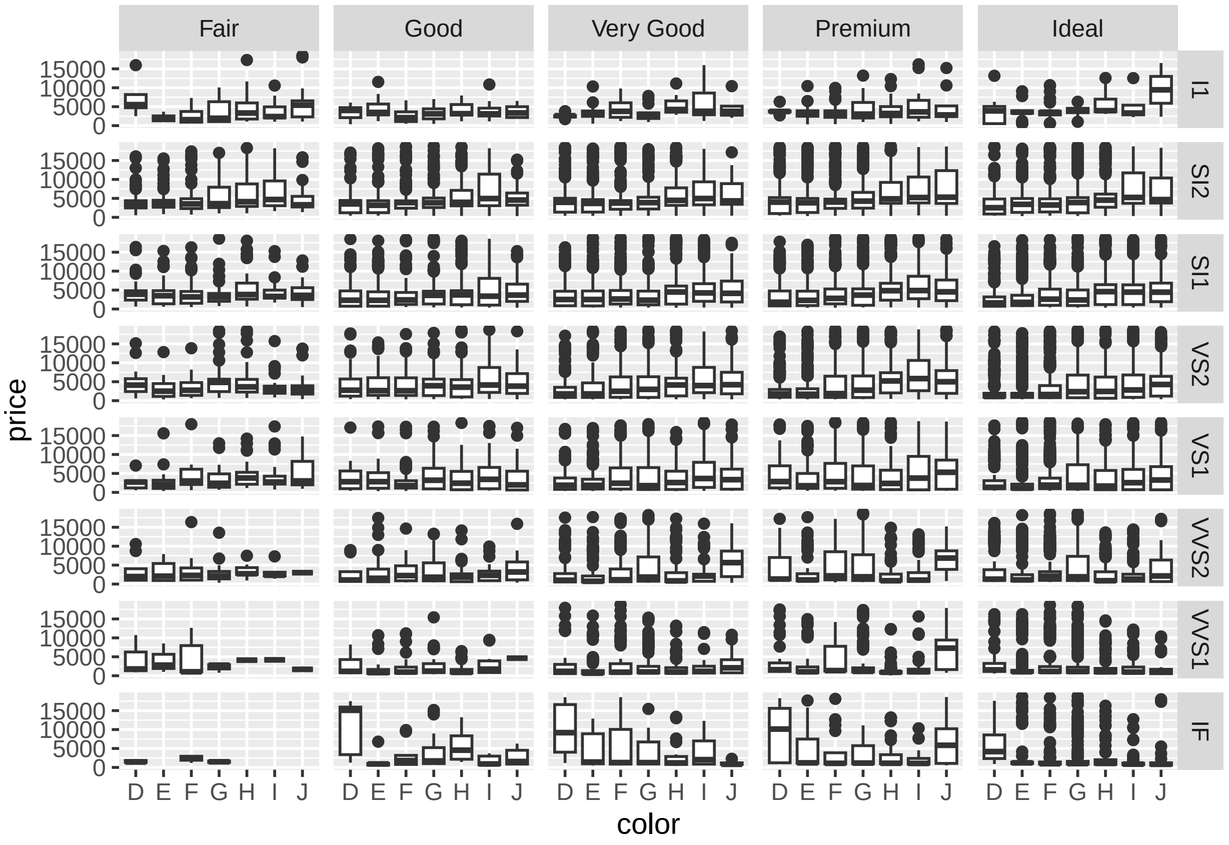
SOLUTION

2d)

In addition to the `facet_wrap()` function, we can create subplots with the `facet_grid()` function. As the name suggests, `facet_grid()` creates a 2D grid layout where each subplot corresponds to a combination of two facetting variables. As with `facet_wrap()`, the syntax uses the formula interface: `facet_grid(<vertical variable> ~ <horizontal variable>)`. The variable provided to the left of the `~` varies top-to-bottom (vertically), while the variable to the right of the `~` changes left-to-right (horizontally).

To see how this works, use `facet_grid()` instead of `facet_wrap()` and set the facetting variables to be `clarity` and `cut` for the vertical and horizontal directions, respectively.

```
ggplot(diamonds, aes(x = color, y = price)) +
  geom_boxplot() +
  facet_grid(vars(clarity),vars(cut))
```



SOLUTION

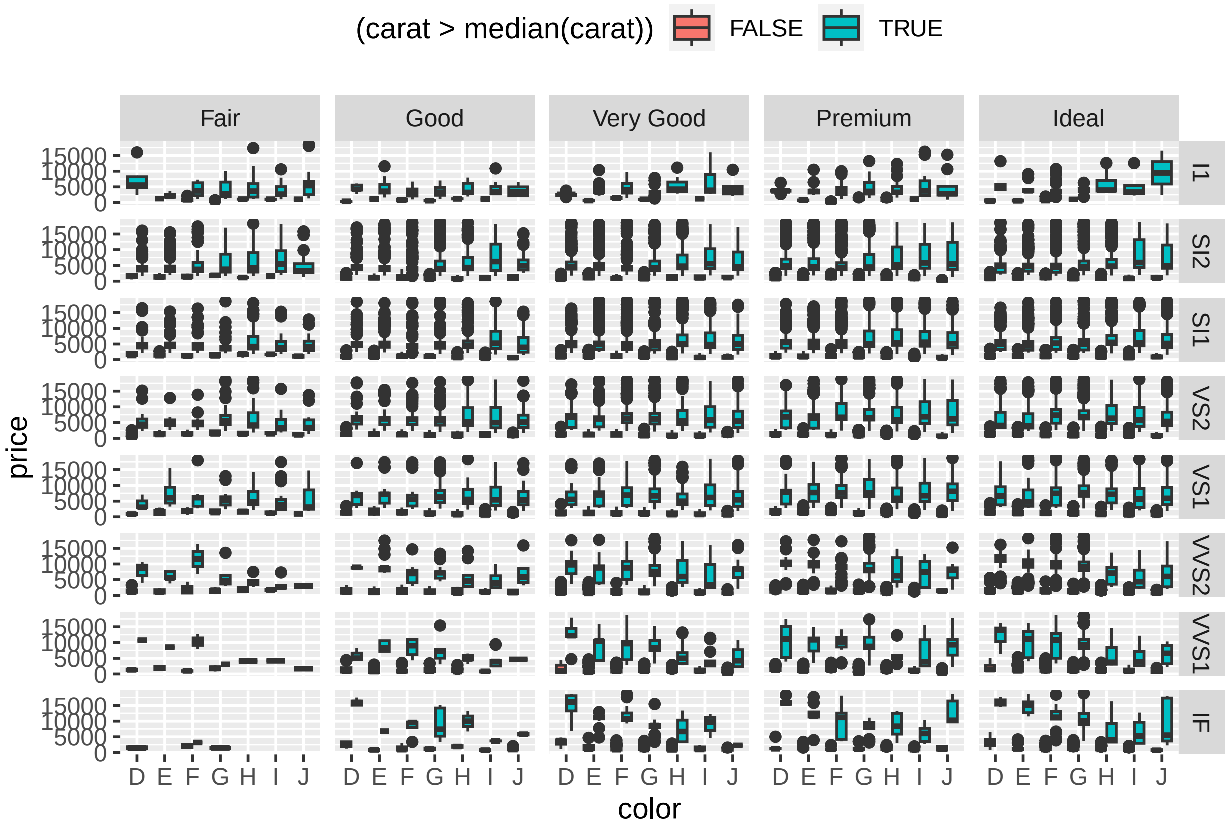
2e)

The resulting figure in Problem 2d) includes 3 out of the 4 C's. The remaining variable, `carat`, is not categorical. To include `carat` in our figure, let's discretize it and compare two boxplots side-by-side at each `color` level within each `clarity` and `cut` subplot combination. For now, we will keep things simple and break up `carat` based on if an observation has a value greater than the median `carat` value.

Within the `geom_boxplot()` function, set the `fill` aesthetic to be a conditional test: `carat > median(carat)`. As shown in the supplemental reading material, use the `theme()` function to move the legend position to the top of the graphic.

Note: It might be difficult to see everything within the graphic window dispalyed in the result after the code chunk, when working within the .Rmd file in the RStudio IDE. You can zoom in by clicking on the "Show in New Window" icon which is displayed as the small "arrow over paper" icon to the right hand size of the output portion. Alternatively, the figure dimensions can be modified by the code chunk parameters `fig.width` and `fig.height`. **For this assignment, it is ok to use the default figure dimensions.**

```
ggplot(diamonds, aes(x = color, y = price)) +
  geom_boxplot(aes(fill=(carat > median(carat)))) +
  facet_grid(vars(clarity),vars(cut)) +
  theme(legend.position = "top")
```



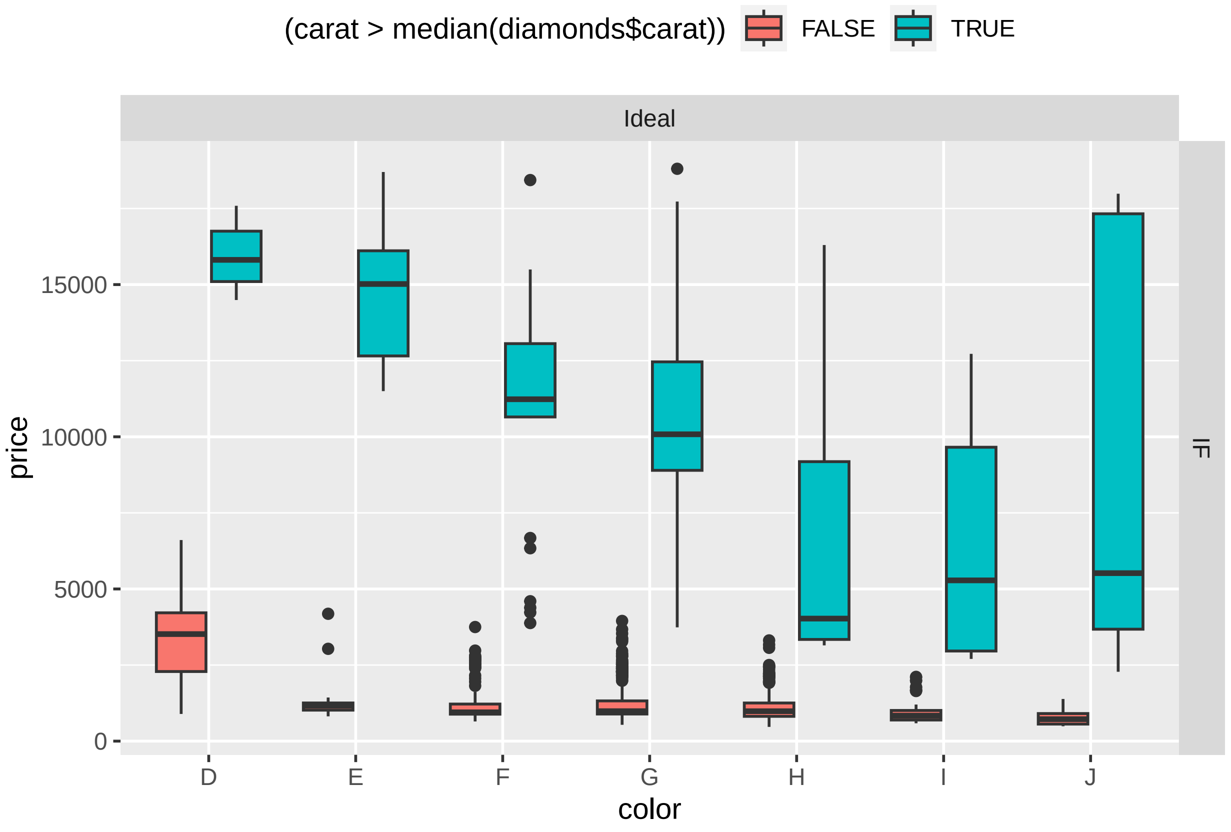
SOLUTION

2f)

Due to the large number of subplots, the individual facets are quite small with the default figure size. Let's focus on the case with `cut == "Ideal"` and `clarity == "IF"` by calling `filter()` before piping the dataset into the `ggplot()`.

Pipe `diamonds` into `filter()` and perform the necessary operation. Pipe the resulting dataset into the same `ggplot2` function calls used in Problem 2e), except for one important change. The `filter()` call will reduce the dataset, and thus our conditional test will be comparing `carat` to the median value associated with the smaller dataset. To force the conditional test to still be applied to the median based on the complete dataset use `median(diamonds$carat)` within the conditional test instead of `median(carat)`.

```
newdiamonds = filter(diamonds,cut == "Ideal",clarity == "IF")
ggplot(newdiamonds, aes(x = color, y = price)) +
  geom_boxplot(aes(fill=(carat > median(diamonds$carat)))) +
  facet_grid(vars(clarity),vars(cut)) +
  theme(legend.position = "top")
```



SOLUTION

2g)

Discuss the differences between the trends shown in the resulting figure in Problem 2f) with the trends shown in the figure in Problem 2b).

SOLUTION Conclusions: In this case, that is, when cut == "Ideal" and clarity == "IF".

1) The median values of the prices are higher when carat > median(diamondscarat) and they are lower when carat <= median(diamondscarat). It shows that carat value increases the price in all colors.
2) Compared to the prices in Problem 2b, the median values of the prices are higher when carat > median(diamondscarat) and they are lower when carat <= median(diamondscarat). This shows that carat value possibly has a similar effect in other cut and clarity types.

Problem 03

Let's now start to introduce model fitting. Within the `R` ecosystem the workhorse of any modeling exercise is the `lm()` function. We will learn what goes on behind the scenes of `lm()` later in the semester. For now, let's just get some practice using `lm()`. It's always helpful to visualize model behavior and so we will first introduce `lm()` through the `ggplot2` `geom_smooth()` geom. `geom_smooth()` is a way to add a "smoothing" trend to a visualization. Chapter 3 of the R4DS book provides an excellent overview of `geom_smooth()`. Reading that chapter will help you with this problem.

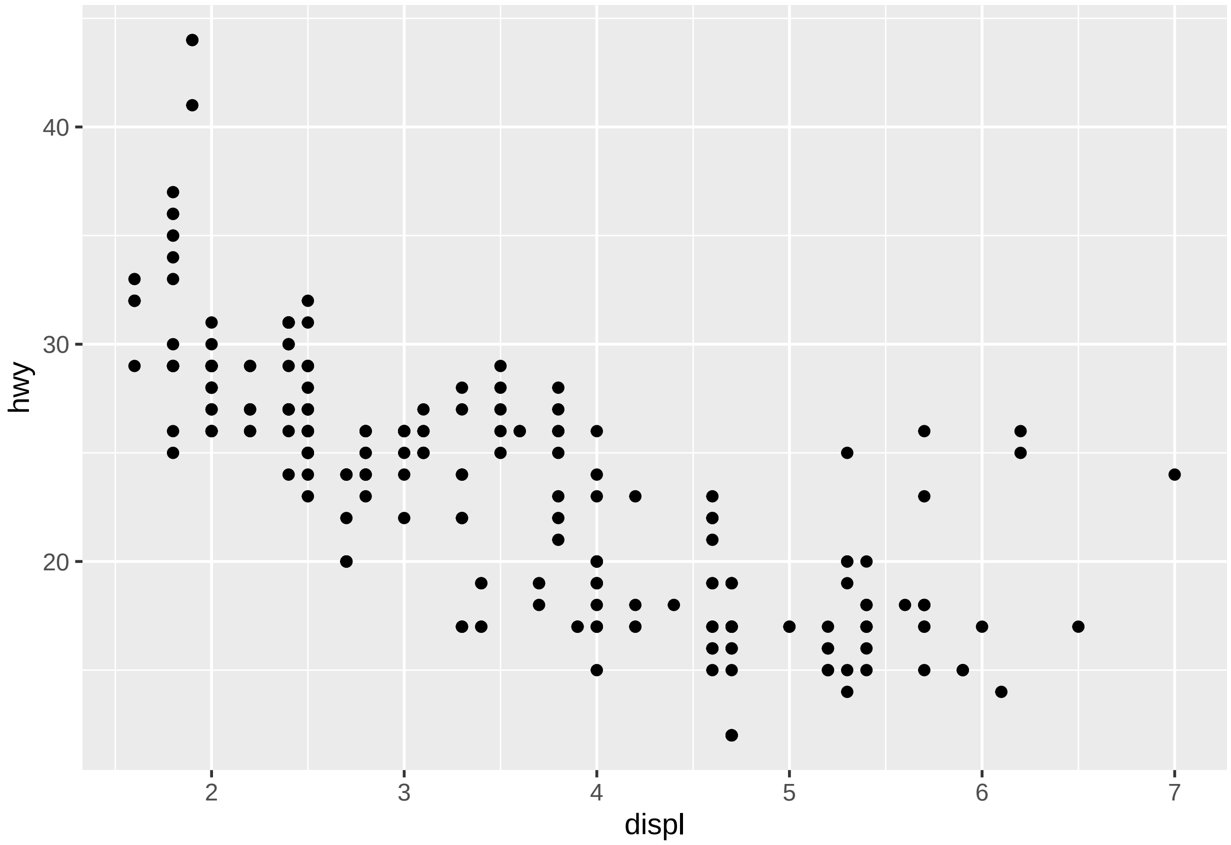
However, Ch. 3 in R4DS focused on the default approach of `geom_smooth()` which applies a non-linear smoothing function. We will instead force `geom_smooth()` to display a linear trend by setting the `method` argument in `geom_smooth()` to be `lm`. You do not have to put quotes around `lm`. You are telling `geom_smooth()` to use the `lm()` function. It is **very** important to note, you should **NOT** type `lm()` with parentheses when you set `method = lm`. We will discuss why that is later in the semester.

3a)

You will use the `mpg` data set just like in Ch. 3 of R4DS. You will first visualize a scatter plot between the `hwy` and `displ` variables.

Create a scatter plot with `displ` on the x-axis and `hwy` on the y-axis. Scatter plots in `ggplot2` are created with the `geom_point()` geom.

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



SOLUTION

3b)

As already stated, `geom_smooth()` allows a trend to be added as a layer to the graphic.

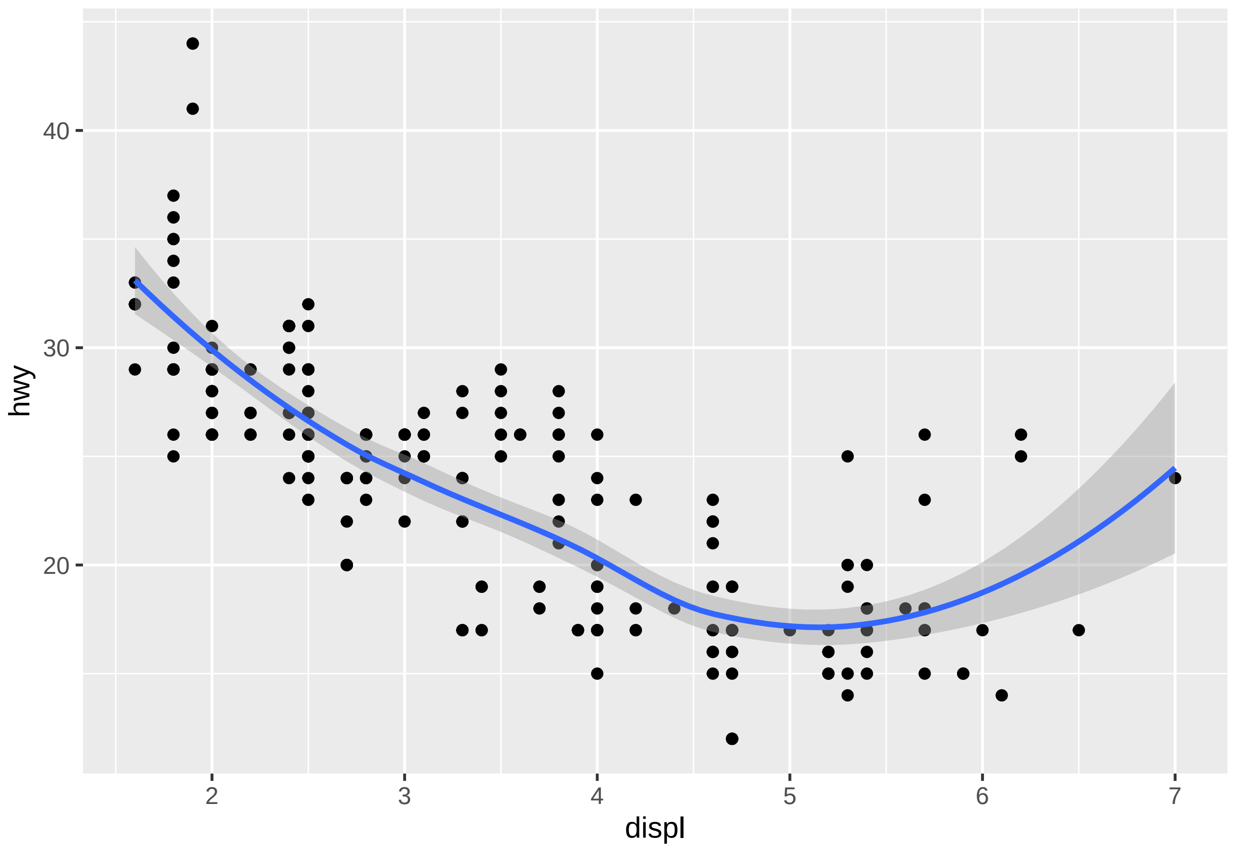
Add `geom_smooth()` to your scatter plot from Problem 3a). You do not need to specify any arguments to `geom_smooth()` in this problem.

NOTE: You may seem some warning messages appear. That's ok for now.

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

SOLUTION

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



3c)

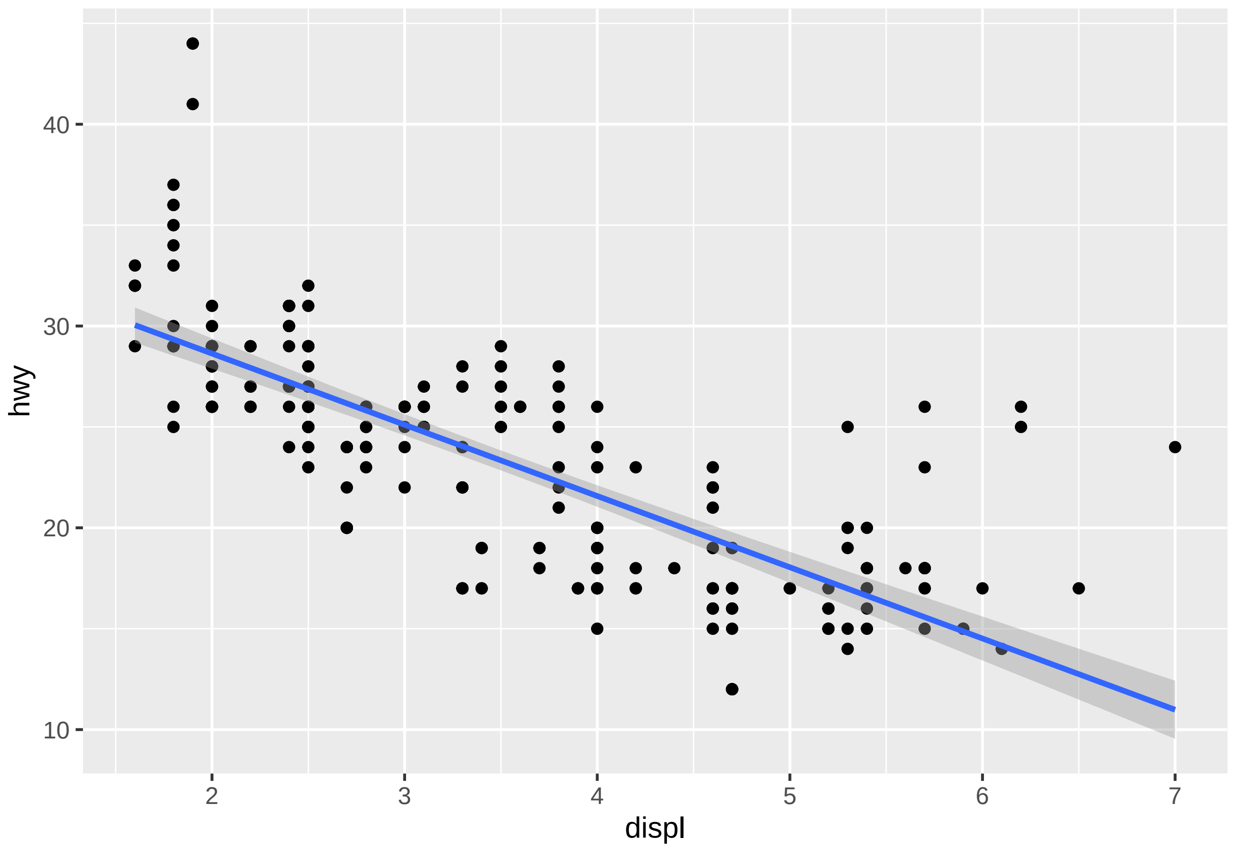
Let’s now visualize a linear trend instead of the default non-linear smoother.

Specify the `method` argument within `geom_smooth()` to be equal to `lm`.

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

SOLUTION

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



3d)

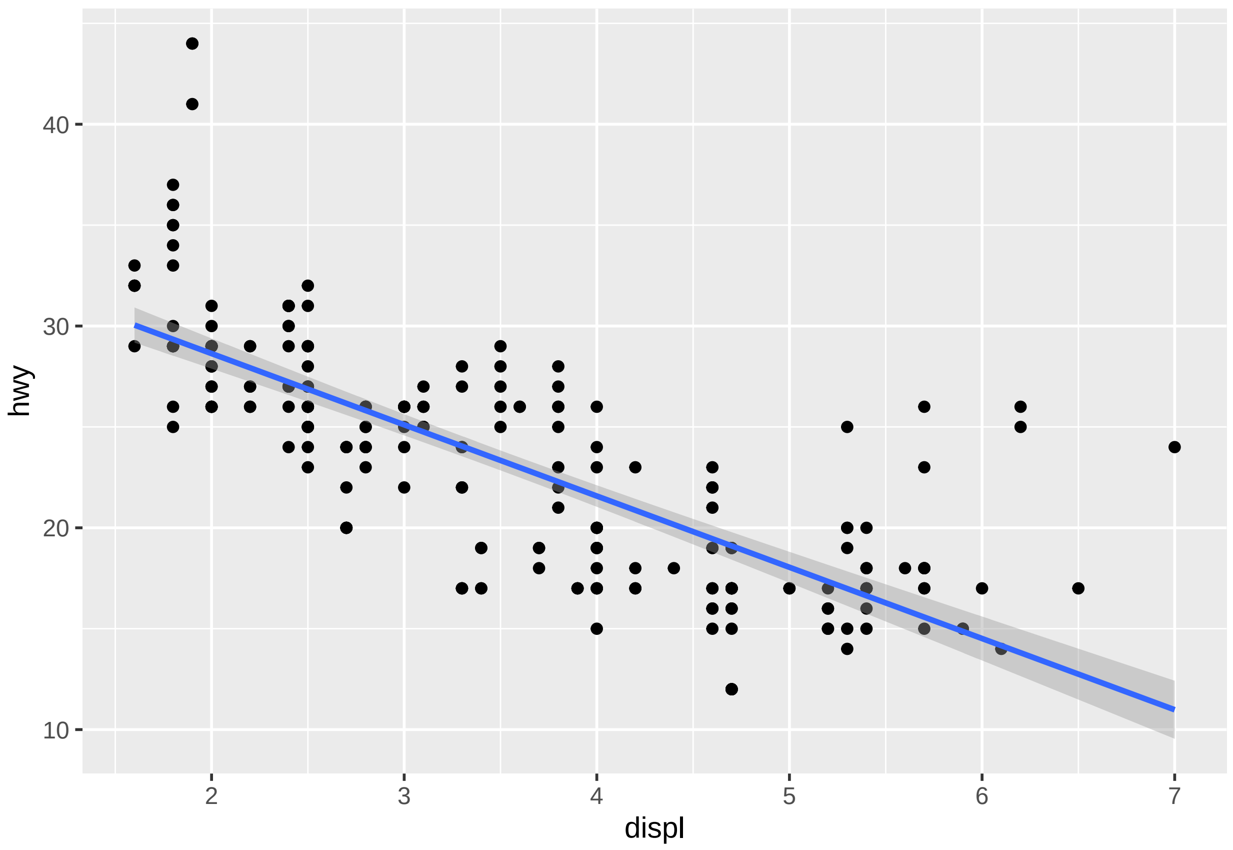
By default, `geom_smooth()` assumes that the input is the variable mapped to the `x` aesthetic and the response is the variable mapped to the `y` aesthetic. You can specify an alternative formula to be used by the smoother through the `formula` argument to `geom_smooth()`. For now, go ahead and use type in the linear relationship formula, `y ~ x`, for the `formula` argument.

`R`'s formula interface reads as the variable to the left of the `~` is the response and all variables to the right of the `~` are the inputs/predictors/features. So by setting the formula to be `y ~ x` you are telling `R` that "y is a function of x".

It's important to note that when specifying the `formula` argument to `geom_smooth()` you can use `x` and `y` because they are "local" to `geom_smooth()`. You do not have to specify the original variable names because of the aesthetic mappings.

Explicitly set the `formula` argument to be "y is a function of x". Keep the `method` argument set to `lm`.

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



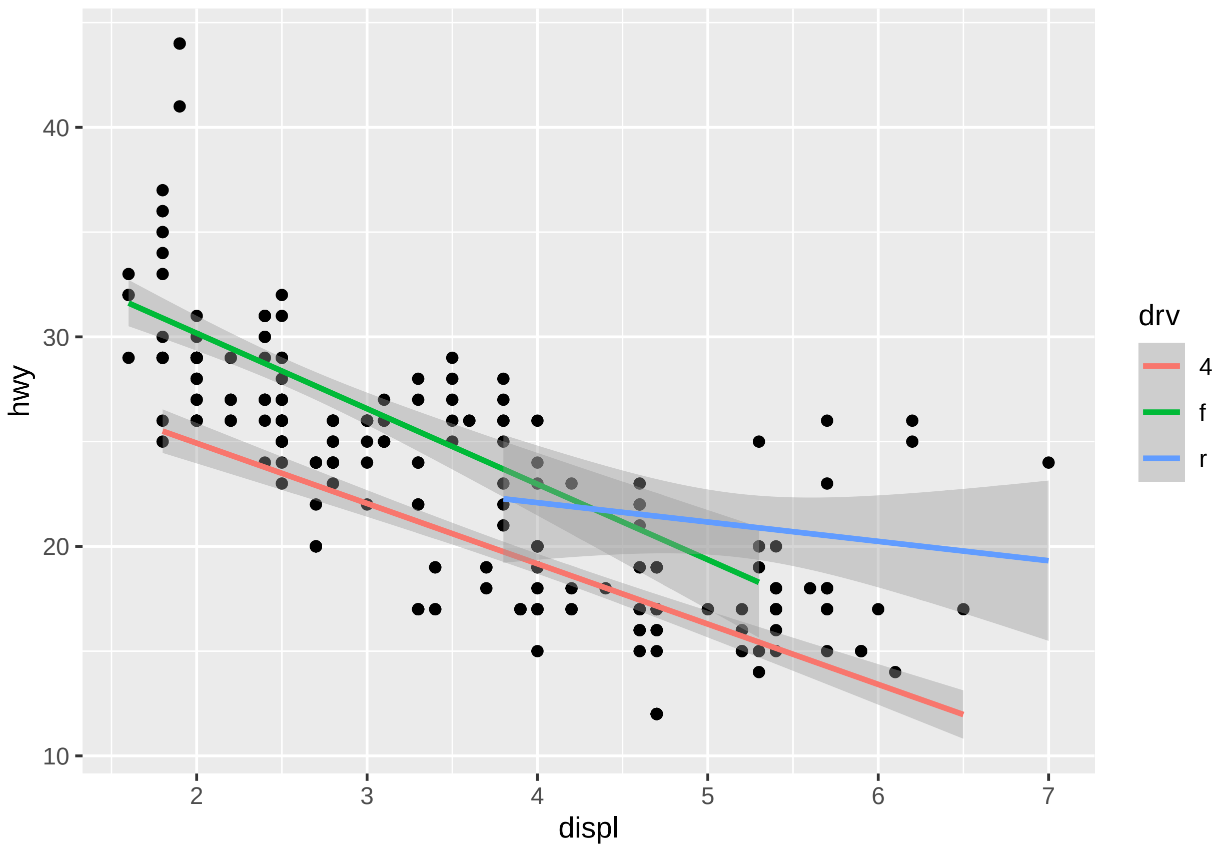
SOLUTION

3e)

`geom_smooth()` has many of the same aesthetics as other geoms like `geom_point()`. You can tell `geom_smooth()` to fit separate trend lines to different groups several ways. First, you can map a discrete variable to the `color` aesthetic to produce separate trend lines with different colors.

Use the same code setup that you used to answer Problem 3d). This time set the `color` aesthetic within `geom_smooth()` to be equal to `drv`.

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy, color=drv),method=lm,formula=y~x)
```



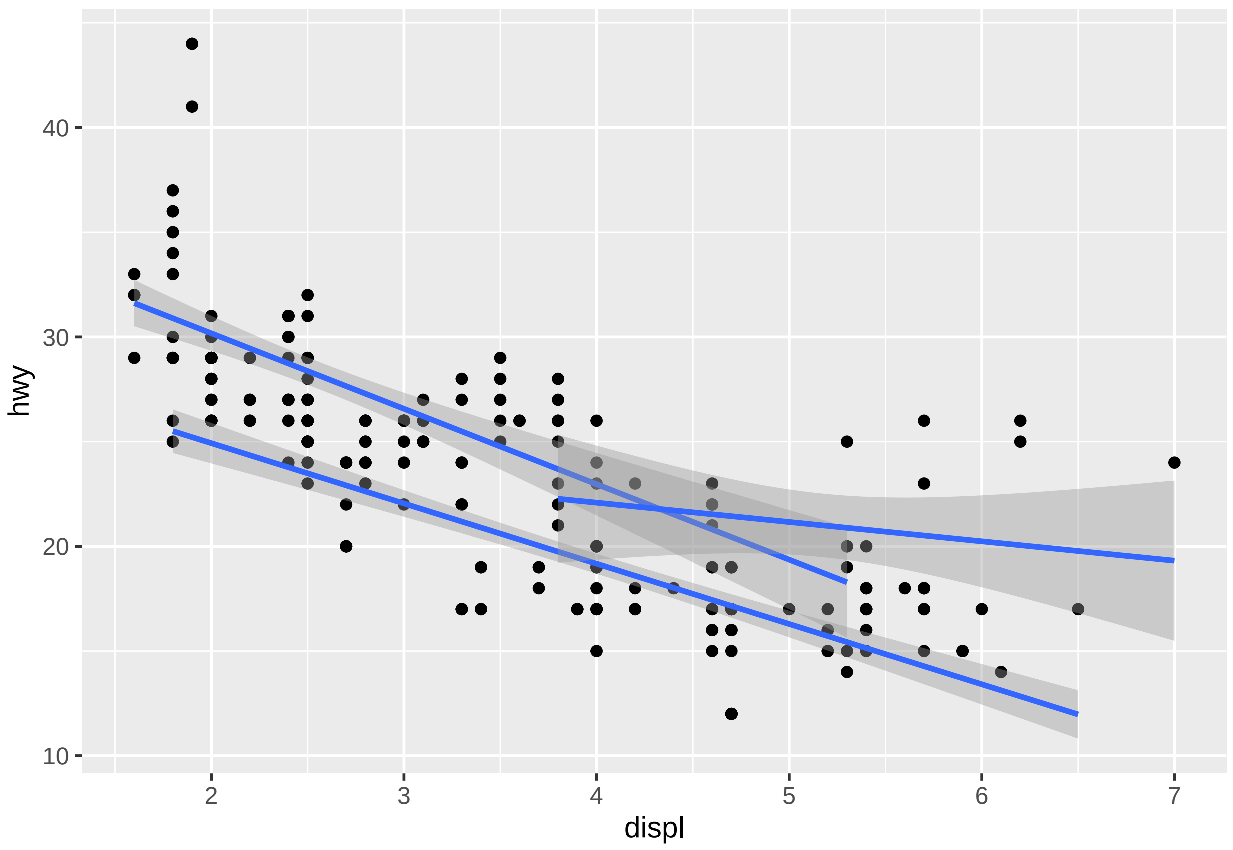
SOLUTION

3f)

When you map a discrete variable in `geom_smooth()`, `ggplot2` first groups the data associated with the unique values (or levels in `R` terminology) together. Then separate trend lines are fit and displayed for the separate groups. We can force the grouping operation to occur without assigning specific colors to the groups with the `group` aesthetic.

Use the same code setup that you used to answer Problem 3e). This time, set the `group` aesthetic within `geom_smooth()` to be equal to `drv`. Do not map any variable to the `color` aesthetic.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy, group=drv),method=lm,formula=y~x)
```



SOLUTION

Problem 04

As useful as it is to include trend lines on figures, we do not have access to such models outside of `ggplot2`. In other words, we can't study their behavior, or make predictions with the models on new data. To do so, we need to fit a model ourselves outside `ggplot2`. In this problem you will use `lm()` directly to fit simple models to the `mpg` data.

4a)

Fit a linear relationship between `hwy` and `displ` using `R`'s formula interface. Assign the model to the variable `mod1` below.

```r
mod1 <- lm(hwy ~ displ, data = mpg)
```

SOLUTION

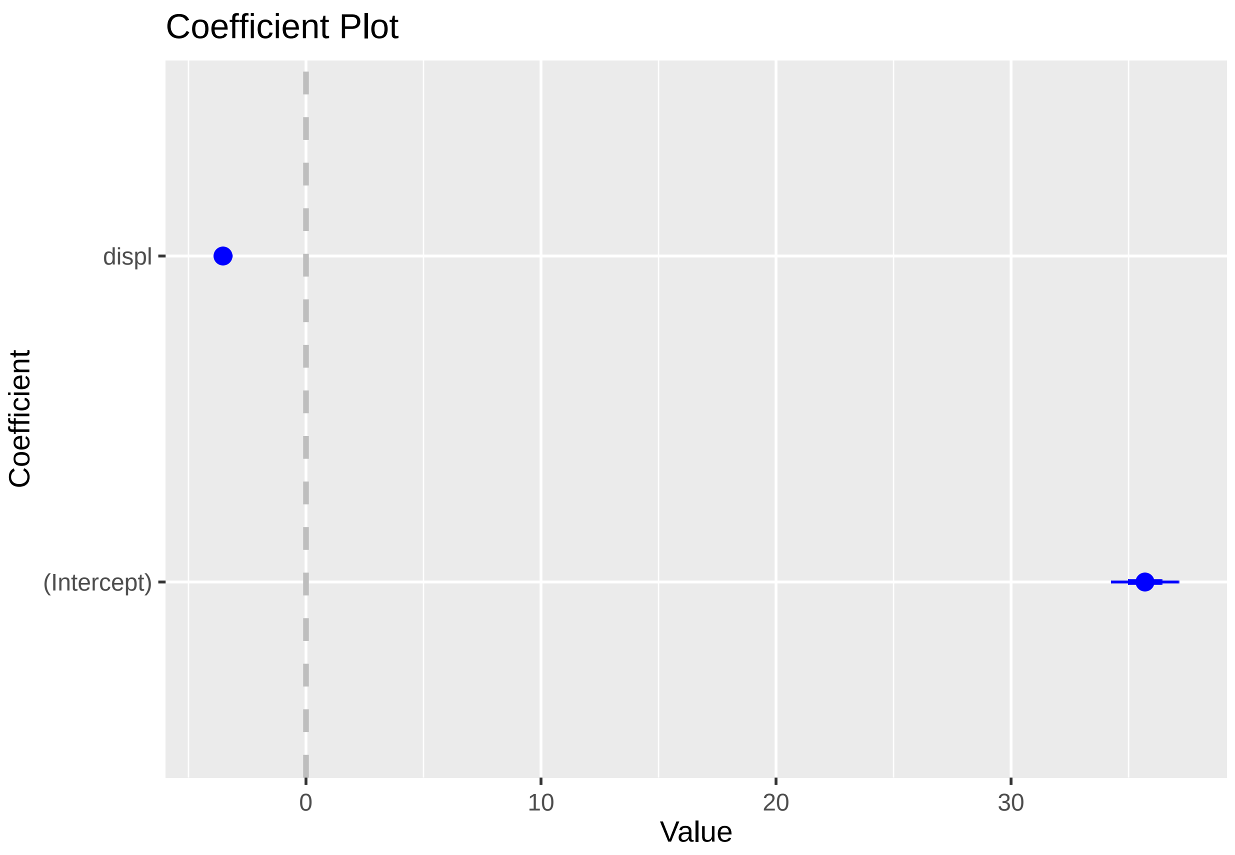
4b)

We could use the `summary()` function to inspect the results, but instead I want you to visualize the coefficients estimates and confidence intervals. To do so, you will use the `coefplot()` function from the `coefplot` package. If you have not downloaded and installed `coefplot` please do so now.

Use `coefplot::coefplot()` to visualize the coefficients for `mod1`. Based on your visualization, is the `displ` variable "significant"?

SOLUTION Yes, the `displ` variable is "significant", because 0 is not contained within the confidence interval.

```
coefplot::coefplot(mod1)
```



4c)

Let's now fit a slightly more complex model which accounts for the influence of the `drv` variable. You will use an additive relationship and so in your formula you only need to separate the two input variables with the `+` operator.

Fit a linear relationship between the `hwy` and the inputs `displ` and `drv`. Treat the inputs as additive. Assign the result to the variable `mod2`.

```
mod2 <- lm(hwy ~ displ+drv, data = mpg)
```

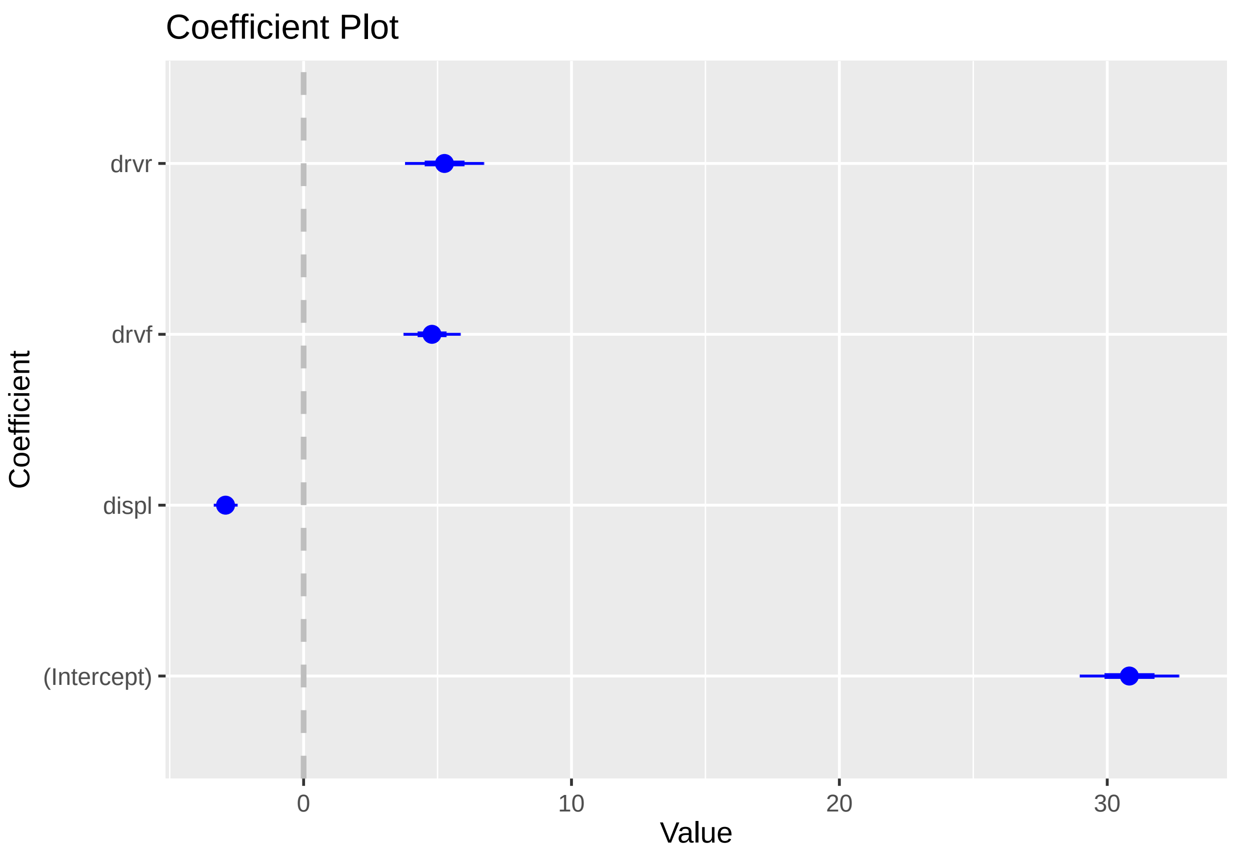
SOLUTION

4d)

Use the `coefplot::coefplot()` function again to visualize the coefficients associated with `mod2`. Are the variables all "significant"? How many coefficients are displayed?

SOLUTION 4 coefficients are displayed and they are all "significant" because 0 is not contained in their confidence interval.

```
coefplot::coefplot(mod2)
```



Problem 05

This question gives you practice working with LaTeX to write math expressions and equations. The sub-parts have mathematical expressions described in words or written in text. You will need to "code up" those expressions in LaTeX within the provided equation blocks.

5a)

We will denote vectors as bold face font lower case letters and matrix with bold face font upper case letters. Thus, the vector $\mathbf{z}$ is written as `$\mathbf{z}$` to create an in-line LaTeX expression.

The equation block is started for you below. Write a system of linear equations in matrix form such that the matrix A multiplied by the vector x equals the vector b. The equation should read as Ax=b, but must be written in bold face font below.

SOLUTION

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

5b)

Parentheses can be created several ways in LaTeX. Using the "basic" `( )` characters will create parentheses around an expression. However, the parentheses are fixed size and do not "grow" as the size of the expression grows. Dynamic parentheses are created by using `\left( \right)` instead of `( )`.

Throughout the semester we will frequently need to use subscripts and superscripts. Subscripts are "attached" with the underscore `_` and superscripts are "attached" with the `^`. For example, `$7_{3}$` would set 3 as the subscript to 7. Using the curly braces `{ }` next to the underscore is a formal way of denoting that everything contained within the curly braces will be used as the subscript.

The equation block is created for you below. Place within dynamically sized parentheses the variable capital X with a subscript of 1 and a superscript of capital Z. The Z superscript itself must have a superscript of capital Y.

SOLUTION

$$\left(X\right)_1^{Z^Y}$$

5c)

Throughout the semester I will use log to represent the natural log. In LaTeX placing the \ character in front of `log` displays the word "log" in non-italic font, `\log`. I like to use parentheses around the expression that the natural log is applied to.

You must write the expression for the "natural log of x squared". Place dynamically sized parentheses around the "x squared" term.

SOLUTION

$$\log\left(x^2\right)$$

5d)

We will often need to write complicated mathematical expressions which are easiest to express in multiple lines. Rendering multiline equation blocks can sometimes be tricky because the Pandoc renderer (which is what RMarkdown uses to render the report) does not like white spaces. For those reasons I feel it is easier to use multiple equation blocks to write complicated multiline expressions.

Let's practice using multiple equation blocks. You should double check that your rendered HTML file shows the equations you were expecting to see!

Create an equation block below where you write the expression "natural log of x squared plus the natural log of y squared equals z squared". Then create a second equation block where you write the expression "natural log of w equals z".

SOLUTION

$$\log\left(x^2\right) + \log\left(y^2\right) = z^2$$

$$\log w = z$$