CS 1675 Homework: 01

Assigned: January 9, 2020; Due: January 17, 2020

Gordon Lu

Submission time: January 17, 2020 at 1:30AM

Collaborators Include the names of your collaborators here.

Overview

This homework assignment focuses on the `ggplot2` and `dplyr` packages. You will work with aesthetics and practice data manipulation operations. Additionally, you will practice working with R Markdown by completing the template. You can see what the rendered document looks like at any time, by pressing the "Knit" button within RStudio. You can execute a code chunk by pressing the arrow button within that code chunk located to the upper right portion of the code chunk within the RStudio IDE. Alternatively, you can execute a line of code within a code chunk by selecting that line and pressing Ctrl+ENTER (Windows) or Command+ENTER (Mac).

Completing this assignment requires filling in missing pieces of information from existing code chunks, programming complete code chunks from scratch, typing discussions about results, and working with LaTeX style math formulas. A template .Rmd file is available to use as a starting point for this homework assignment. The template is available on CourseWeb.

IMPORTANT: Please pay attention to the `eval` flag within the code chunk options. Code chunks with `eval=FALSE` will **not** be evaluated (executed) when you Knit the document. You **must** change the `eval` flag to be `eval=TRUE`. This was done so that you can Knit (and thus render) the document as you work on the assignment, without worrying about errors crashing the code in questions you have not started. Code chunks which require you to enter all of the required code do not set the `eval` flag. Thus, those specific code chunks use the default option of `eval=TRUE`.

IMPORTANT: Each problem provides a fair amount of discussion to help teach the `R` syntax. However, the last problem, Problem 6, involves writing LaTeX math code. Some of the discussion was removed from Problem 6 within this template .Rmd file because if it was included the answers would have been given away! Therefore, please consult the `HW_instructions_01.html` file for the complete discussion associated with Problem 6.

Load packages

This homework assignment will use the following packages:

```r
library(dplyr)
library(ggplot2)
```

Problem 1

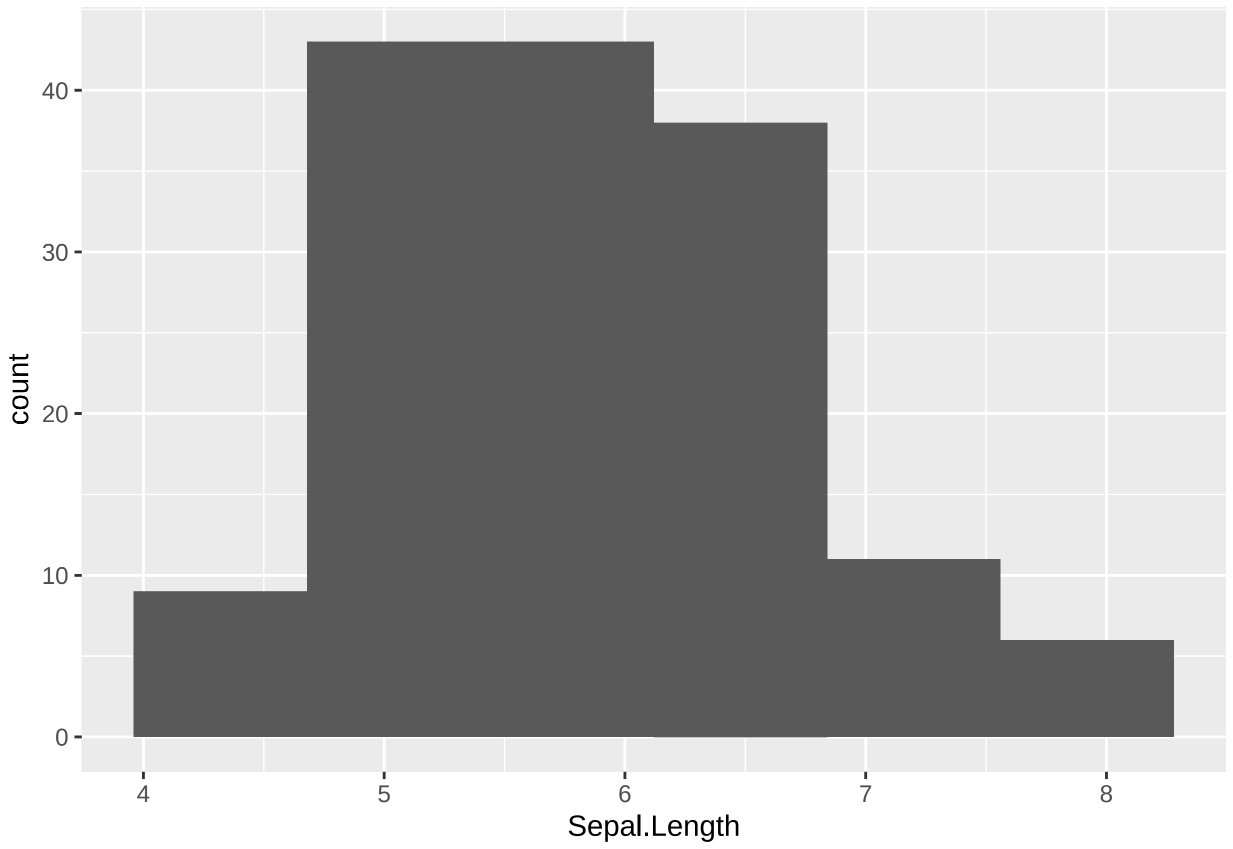
The supplemental reading material A Very Quick Intro to R introduced `ggplot2` by creating a histogram for the `Sepal.Length` variable within the `iris` dataset. We will use that figure to practice modifying color within a `ggplot2` graphic.

1a)

PROBLEM Create a histogram for `Sepal.Length` from `iris` with 6 bins, instead of the 15 bins used within the supplemental reading material.

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

SOLUTION



1b)

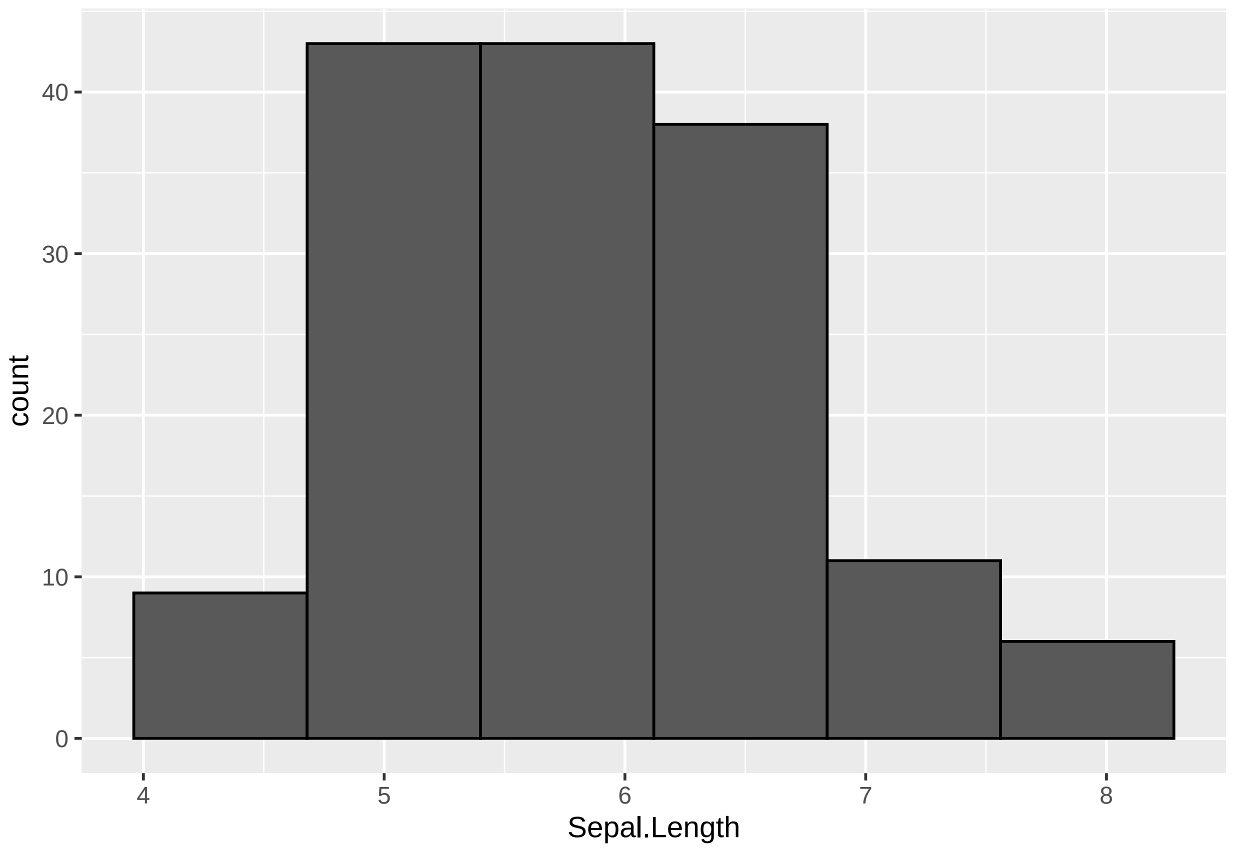
By default, a `ggplot2` histogram does not show the lines associated with each bin (as in a bar graph). The histogram effectively looks like a discretized distribution. To adjust this, we need to override the default `fill` and `color` arguments to the `geom_histogram()` function. Note that within `ggplot2`, 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 type of argument, but we set that argument outside of the `aes()` function.

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

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

SOLUTION



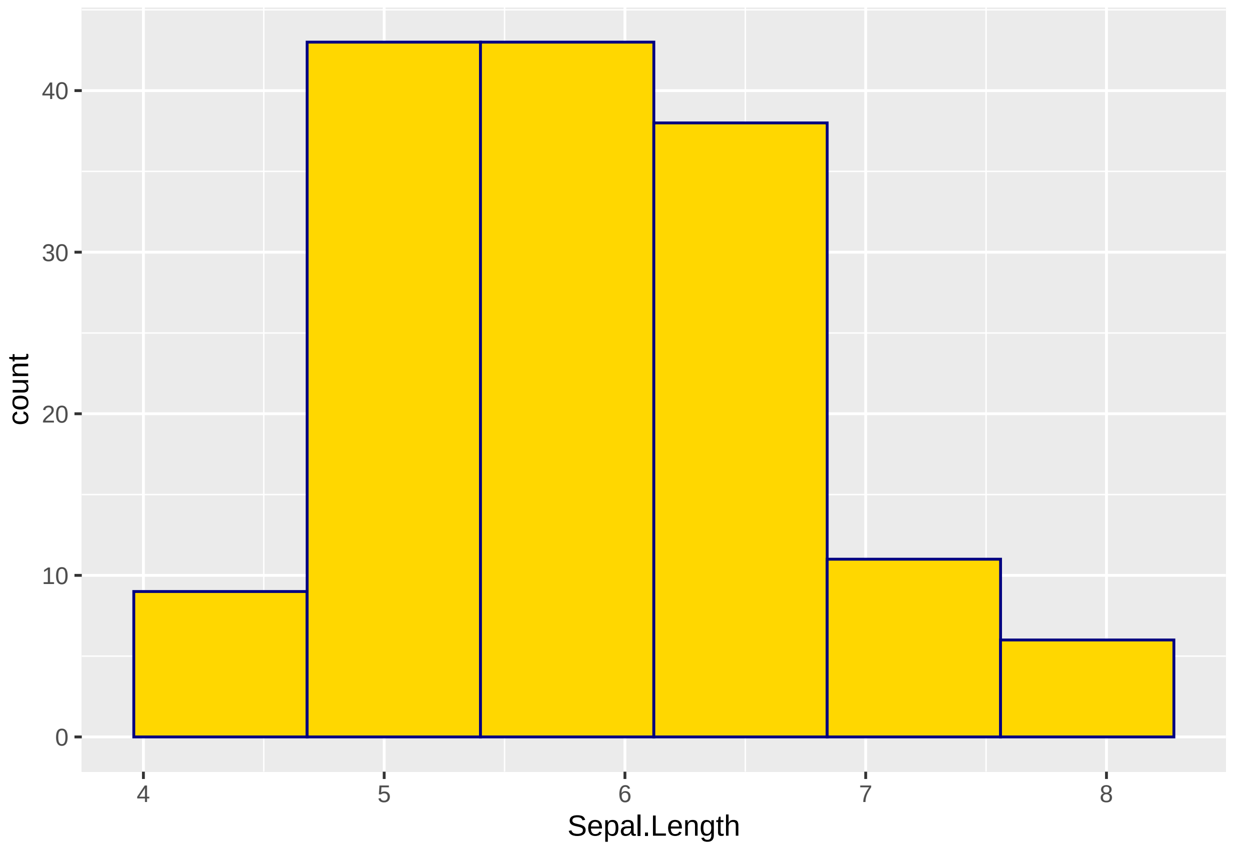
1c)

ggplot2 has many "named" colors available for use. If you really want to fine tune your colors you are free to use the hex color codes! In this course, we will typically stick with common colors when we manually pick a color and/or fill.

PROBLEM 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(bins = 6, color = "navyblue", fill = "gold")
```

SOLUTION



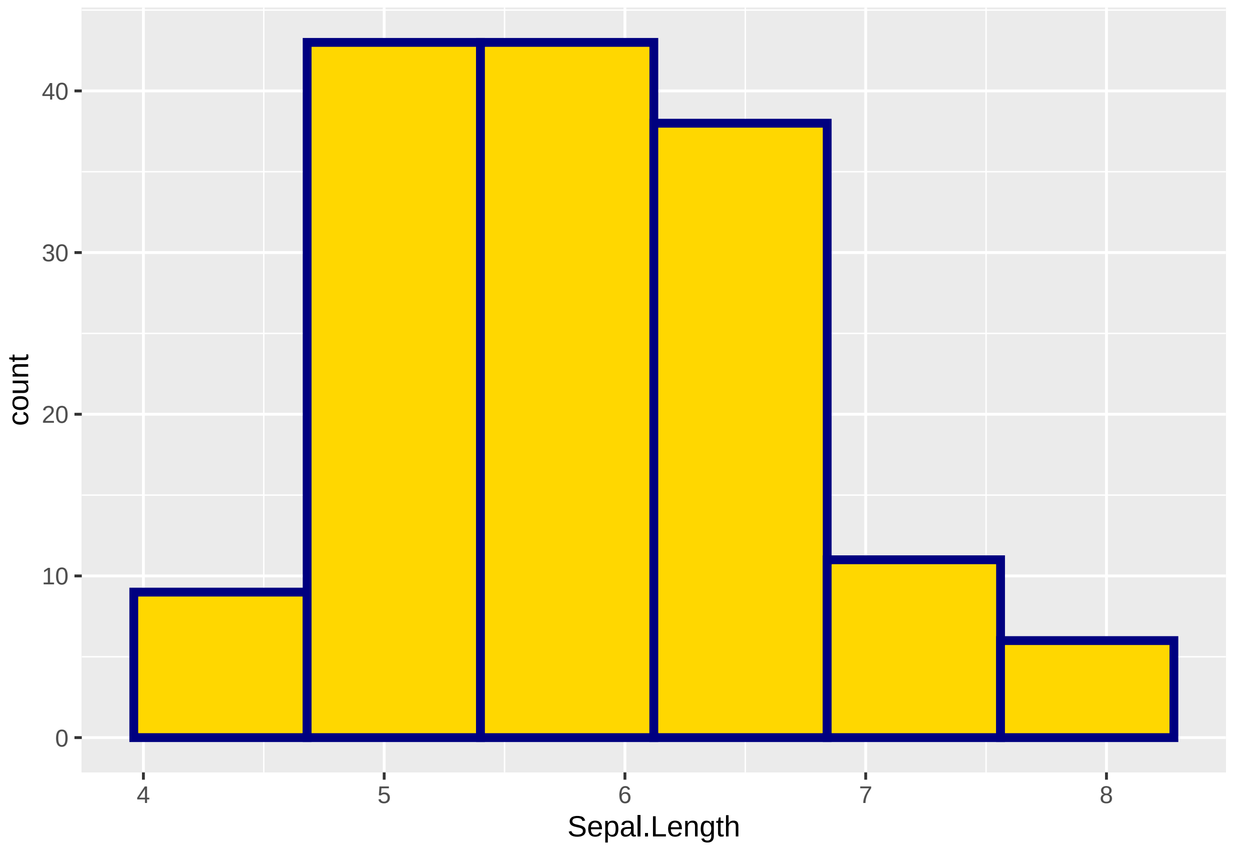
1d)

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

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

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

SOLUTION



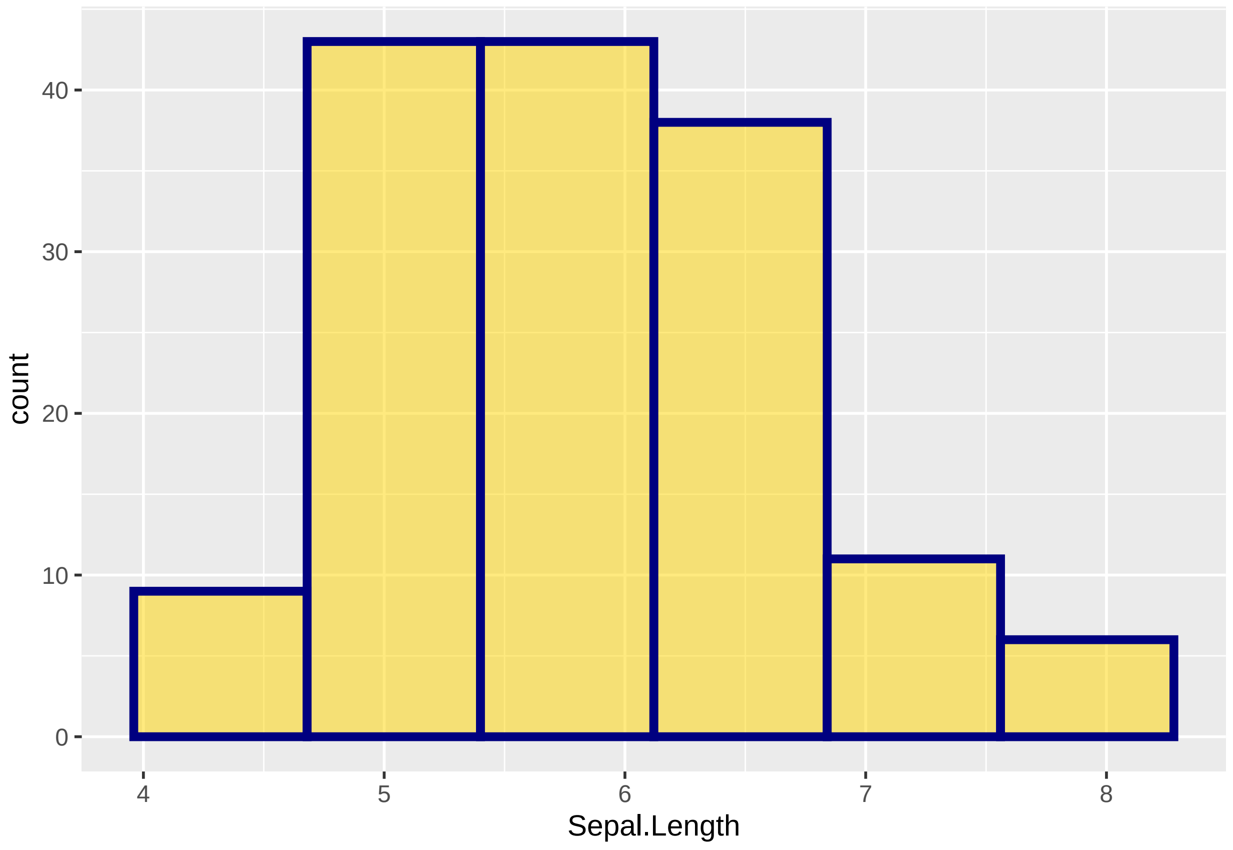
1e)

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

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

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

SOLUTION



Problem 2

As discussed in the supplemental reading material, `geom_histogram()` performs multiple operations behind the scenes in order to generate the histogram visualization. Understanding how those operations work will provide useful data wrangling and manipulation practice.

2a)

`ggplot2` takes care of the binning and counting operations for us when we call the histogram function. Thus, we must perform those steps in order to recreate the histogram "from scratch". We will start by learning how to discretize a continuous variable. To accomplish this, we will need to create a vector which defines the edges of each bin within the histogram. Within `R` nomenclature, the edges are referred to as "breaks".

PROBLEM Since we used 6 bins for our histogram, how many breaks are associated with the histogram? Assign your answer to the variable `num_breaks` in the code chunk below.

```
num_breaks <- 7
```

SOLUTION

2b)

Next, create the vector `hist_breaks` by breaking up the range spanned by `Sepal.Length` via the `seq()` function. `seq()` has two main arguments: `from` and `to`. As the names suggest, the `from` argument is the start of the vector and the `to` argument is the end of the vector. In this problem, we will set the `from` and `to` arguments such that all observations are within the assigned discretized bounds. The third argument to `seq()` can take multiple forms, but all serve the same purpose: to define the length of the vector. For this problem, we will use the `length.out` argument.

PROBLEM Create the vector `hist_breaks` with the `seq()` function. Use the provided `lower_bound` and `upper_bound` values in the code chunk below as the `from` and `to` arguments, respectively. Specify the `length.out` argument to be equal to `num_breaks`.

```
lower_bound <- 4.2

upper_bound <- 8.2

hist_breaks <- seq(from = lower_bound , to = upper_bound, length.out = num_breaks)
```

SOLUTION

2c)

With the break positions defined, we can discretize `Sepal.Length` into bins via the `cut()` function. To learn about `cut()`, type `?cut` into the R console in order to bring up the documentation. As the help page says, `cut()` divides a numeric variable into discrete bins and converts the result the a factor. The bin "edges" are based on the values specified by the `breaks` argument to the `cut()` function. We will practice working with `cut()` first, before applying it to the `Sepal.Length` variable within the `iris` dataset.

PROBLEM In the code chunk below, assign the sequential integers `0` through `11` to the variable `x_dbl` using the `:` notation. Then pass `x_dbl` to the `cut()` function and specify the breaks to be 0, 5, and 11 by creating a vector containing just those three values. Assign the result of the `cut()` function to the variable `x_factor`. Check the data type associated with `x_factor`, and then print the `x_factor` variable to the screen.

```
x_dbl <- 0:11 ### create vector 0 to 11

x_factor <- cut(x_dbl, breaks = c(0,5,11)) ### set the breaks argument

class(x_factor)
```

SOLUTION

```
## [1] "factor"
```

```
x_factor
```

```
##  [1] <NA>   (0,5]  (0,5]  (0,5]  (0,5]  (0,5]  (5,11] (5,11] (5,11] (5,11]
## [11] (5,11] (5,11]
## Levels: (0,5] (5,11]
```

2d)

The unique values associated with a factor variable are displayed in the line below the individual elements of the vector. In `R` nomenclature, factor unique values are referred to as "levels". In your answer to Problem 2c) however, you should notice that the first printed element reads `<NA>`. That's because the discretized intervals are setup to be (,] (which reads as "from but exluding lower value, to and including upper value"). We can modify this structure to include the lowest (minimum) value in the dataset by setting `include.lowest = TRUE` in the call to the `cut()` function.

PROBLEM Reperform the `cut()` call on `x_dbl`, but this time set `include.lowest = TRUE`. Assign the result to the vector `x_factor_2` and then print the result to the screen.

```
x_factor_2 <- cut(x_dbl, include.lowest = TRUE, breaks = c(0,5,11))
x_factor_2
```

SOLUTION

```
##  [1] [0,5]  [0,5]  [0,5]  [0,5]  [0,5]  [0,5]  (5,11] (5,11] (5,11] (5,11]
## [11] (5,11] (5,11]
## Levels: [0,5] (5,11]
```

2e)

We will now modify the `iris` dataset by discretizing `Sepal.Length` via the `cut()` function. We will use the `dplyr::mutate()` "action verb" to execute this task. As the function name suggests, `mutate()` modifies the dataset by creating new variables. The generic syntax is `mutate(<new variable> = <set of actions>)`. We can perform simple actions to create a new variable, or execute a complex set of tasks. `dplyr::mutate()` is a general "wrapper" for performing those operations.

PROBLEM Create the new discretized variable, `sepal_length_bin`, by setting the `breaks` argument within the cut function to be `hist_breaks` and set the `include.lowest` argument to be `TRUE`. Assign the modified dataset to the variable `my_iris`.

```
my_iris <- iris %>% mutate(sepal_length_bin =
        cut(Sepal.Length, include.lowest = TRUE, breaks = hist_breaks))
```

SOLUTION

Problem 3

Our new dataset, `my_iris`, contains an additional variable compared to the "base" `iris`. We will now practice manipulating the dataset based on grouping with this new variable.

3a)

Let's get an overview of the newly created variable, `sepal_length_bin`.

PROBLEM Use the `select()` action verb to isolate the `sepal_length_bin` variable from the other variables and pipe the result to the `summary()` function.

```
my_iris %>% select(sepal_length_bin) %>% summary()
```

SOLUTION

```
##     sepal_length_bin
##  [4.2,4.87] :16
##  (4.87,5.53]:43
##  (5.53,6.2] :36
##  (6.2,6.87] :38
##  (6.87,7.53]:11
##  (7.53,8.2] : 6
```

3b)

Because `sepal_length_bin` is a factor variable, we can easily get the unique values (levels) with the `levels()` function.

PROBLEM Pass the `sepal_length_bin` vector into the `levels()` function, by accessing it with the `$` operator from within `my_iris`.

```
levels(my_iris$sepal_length_bin)
```

SOLUTION

```
## [1] "[4.2,4.87]"  "(4.87,5.53]" "(5.53,6.2]"  "(6.2,6.87]"  "(6.87,7.53]"
## [6] "(7.53,8.2]"
```

3c)

In general, we can use the `unique()` function to identify the unique values contained within a vector irregardless of whether that vector is a `"numeric"`, `"character"`, or a factor variable. There are several important differences between factor levels and unique values, but we will not be too concerned with those differences at the moment.

PROBLEM **Call `unique()` on `sepal_length_bin` by accessing the variable with the `$` operator again.**

```r
unique(my_iris$sepal_length_bin)
```

SOLUTION

```
## [1] (4.87,5.53] [4.2,4.87]  (5.53,6.2]  (6.87,7.53] (6.2,6.87]  (7.53,8.2]
## 6 Levels: [4.2,4.87] (4.87,5.53] (5.53,6.2] (6.2,6.87] ... (7.53,8.2]
```

3d)

As you should have seen in the previous set of results, the `levels()` and `unique()` functions return vectors. However, if we use the `dplyr` function `distinct()`, we will return a `data.frame` containing the unique values associated with the variable(s). We tell `distinct` which variable(s) to focus on using non-standard evaluation.

PROBLEM **In the code chunk below, pipe `my_iris` to the `distinct()` function and specify the variable to be `sepal_length_bin`.**

```r
my_iris %>% distinct(sepal_length_bin)
```

SOLUTION

```
##   sepal_length_bin
## 1      (4.87,5.53]
## 2       [4.2,4.87]
## 3       (5.53,6.2]
## 4      (6.87,7.53]
## 5       (6.2,6.87]
## 6       (7.53,8.2]
```

3e)

`distinct()` is a useful function, but it does not provide any other information about the variables. The `count()` function, however, returns the number of rows asssociated with each unique value. The number of rows (or the count) are stored in an automatically created additional column named `n`.

PROBLEM In the code chunk below, apply the `count()` function instead of the `distinct()` function to the `sepal_length_bin` variable.

```
my_iris %>% count(sepal_length_bin)
```

SOLUTION

```
## # A tibble: 6 x 2
##   sepal_length_bin     n
##   <fct>            <int>
## 1 [4.2,4.87]          16
## 2 (4.87,5.53]         43
## 3 (5.53,6.2]          36
## 4 (6.2,6.87]          38
## 5 (6.87,7.53]         11
## 6 (7.53,8.2]           6
```

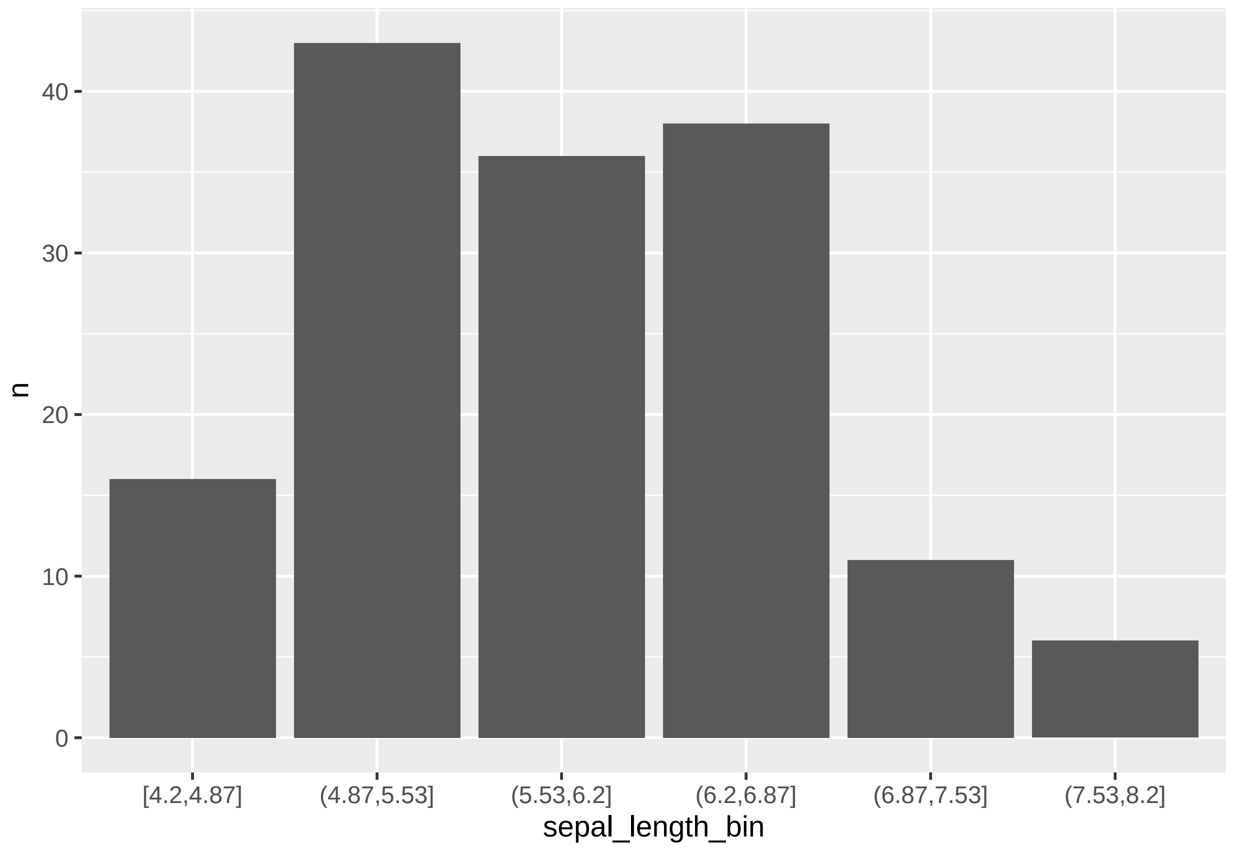
3f)

We have sufficient information with the `count()` function result to make a bar graph in the style of the histogram we are interested in.

PROBLEM Pipe the result of the `count()` function into the `ggplot()` function. Set the `x` and `y` aesthetics to be `sepal_length_bin` and `n`, respectively. Then call `geom_bar()`. However, by default `geom_bar()` will not work with this setup. As with `geom_histogram()`, `geom_bar()` performs a grouping and counting operations behind the scenes. To override this default behavior, we must modify the `stat` argument to be `stat = "identity"`. The code chunk below already has the correct `geom_bar()` call. Fill in the missing code to create the visualization of interest.

```r
my_iris %>%
  count(sepal_length_bin) %>%
  ggplot(mapping = aes(x = sepal_length_bin, y = n)) +
  geom_bar(stat = "identity")
```

SOLUTION



3g)

The resulting bar graph in Problem 3f) looks different from the histogram visualized in Problem 1).

PROBLEM Why is the x-axis setup differently from the histogram in Problem 1)?

SOLUTION The x-axis is setup based on levels, by which the Sepal Length values are grouped by, this in a way discretizes the Sepal Length variable. Furthermore, any reoccurring values were filtered out of the data. In Problem 1, we stuck with treating Sepal Length as a continuous variable. Additionally, the frequencies of the newly generated histogram are different than that of the original histogram, due to the grouping and leveling of Sepal Length in the new histogram.

Problem 4

The `count()` function is useful for quickly grouping and counting by one or multiple variables. However, that is all the function is intended to do. In order to perform other calculations we need to use the `group_by()` function in conjunction with a function like `summarize()`. In this problem, you will learn about these very useful and important operations.

4a)

The grouping variables are specified using non-standard evaluation in the `group_by()` call. Thus, to group by a variable use the following syntax: `group_by(<variable name>)`. Grouping by 3 variables is just a straight forward extension of this: `group_by(<variable 1>, <variable 2>, <variable 3>)`.

PROBLEM Pipe `my_iris` into the `group_by()` function and specify the grouping variable to be `sepal_length_bin`.

```
my_iris %>% group_by(sepal_length_bin)
```

SOLUTION

```
## # A tibble: 150 x 6
## # Groups:   sepal_length_bin [6]
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species sepal_length_bin
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>   <fct>
##  1          5.1         3.5          1.4         0.2 setosa  (4.87,5.53]
##  2          4.9         3            1.4         0.2 setosa  (4.87,5.53]
##  3          4.7         3.2          1.3         0.2 setosa  [4.2,4.87]
##  4          4.6         3.1          1.5         0.2 setosa  [4.2,4.87]
##  5          5           3.6          1.4         0.2 setosa  (4.87,5.53]
##  6          5.4         3.9          1.7         0.4 setosa  (4.87,5.53]
##  7          4.6         3.4          1.4         0.3 setosa  [4.2,4.87]
##  8          5           3.4          1.5         0.2 setosa  (4.87,5.53]
##  9          4.4         2.9          1.4         0.2 setosa  [4.2,4.87]
## 10          4.9         3.1          1.5         0.1 setosa  (4.87,5.53]
## # ... with 140 more rows
```

4b)

The result after applying the grouping operation appears to be just the original dataset. However, this new dataset "knows" the grouping structure. Thus, any additional operations we apply will be applied to the grouped dataset. When we wish to summarize based on the grouping structure, we pipe the `group_by()` result into the `summarize()` function. Calculations within the `summarize()` function are performed with syntax like we used with the `mutate()` function.

The code chunk below shows how the number of observations per `sepal_length_bin` value are computed using the `n()` function within the `summarize()` call.

PROBLEM Complete the code chunk below by naming this summary variable `num_obs`. How do the resulting counts compare with the result from `count()`?

Note: the `n()` function is an example of a function without any input arguments. The syntax for calling such a function still requires the `()` after the function name. This syntax instructs the `R` interpreter that you are using a function named `n` and not a variable named `n`.

```r
my_iris %>% group_by(sepal_length_bin) %>% summarize( num_obs  = n())
```

SOLUTION

```
## # A tibble: 6 x 2
##   sepal_length_bin num_obs
##   <fct>              <int>
## 1 [4.2,4.87]            16
## 2 (4.87,5.53]           43
## 3 (5.53,6.2]            36
## 4 (6.2,6.87]            38
## 5 (6.87,7.53]           11
## 6 (7.53,8.2]             6
```

The number of observations, num_obs, are the same as the result from count().

4c)

We can do more than just counting in the `summarize()` call. We can calculate averages, standard deviations, quantiles, and even call complex modeling functions. Think of `summarize()` as a versatile wrapper function for controlling operations applied to a grouped dataset.

PROBLEM Complete the code chunk below to calculate the average, minimum, and maximum `Sepal.Length` values for each `sepal_length_bin` level. In `R`, the average is calculated with the `mean()` function. The minimum and maximum values are calculated with the `min()` and `max()` functions, respectively. The grouped and summarized dataset has been named `iris_group` and is printed to the screen for review.

```r
iris_group <- my_iris %>%
  group_by( sepal_length_bin ) %>%
  summarise(num_obs = n(),
            avg_sepal_length = mean(Sepal.Length),
            min_sepal_length = min(Sepal.Length),
            max_sepal_length = max(Sepal.Length))

iris_group
```

SOLUTION

```
## # A tibble: 6 x 5
##   sepal_length_bin num_obs avg_sepal_length min_sepal_length max_sepal_length
##   <fct>              <int>            <dbl>            <dbl>            <dbl>
## 1 [4.2,4.87]            16             4.61              4.3              4.8
## 2 (4.87,5.53]           43             5.17              4.9              5.5
## 3 (5.53,6.2]            36             5.84              5.6              6.1
## 4 (6.2,6.87]            38             6.47              6.2              6.8
## 5 (6.87,7.53]           11             7.09              6.9              7.4
## 6 (7.53,8.2]             6             7.72              7.6              7.9
```

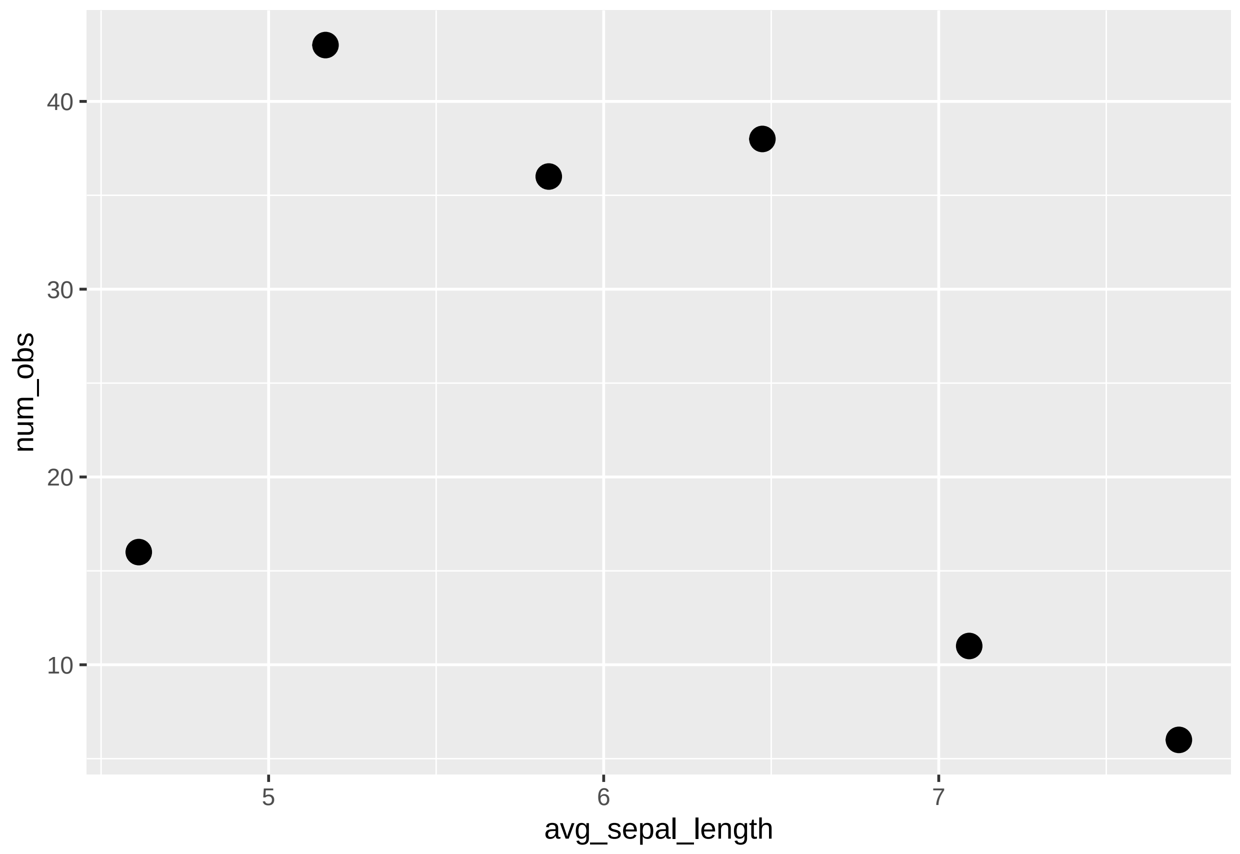
4d)

We can use the grouped and summarized dataset to create a simple histogram which introduces the `geom_point()` and `geom_linerange()` geoms. `geom_point()` creates a basic scatter plot between two variables mapped to the `x` and `y` aesthetics.

PROBLEM In the code chunk below, assign `avg_sepal_length` to the `x` aesthetic within the parent `ggplot()` call. Then, set `num_obs` to the `y` aesthetic within the `geom_point()` call. Lastly, assign the `size` argument within `geom_point()` to be 4.

```
iris_group %>%
  ggplot(mapping = aes(x = avg_sepal_length)) +
  geom_point(mapping = aes(y = num_obs),
             size = 4)
```

SOLUTION



4e)

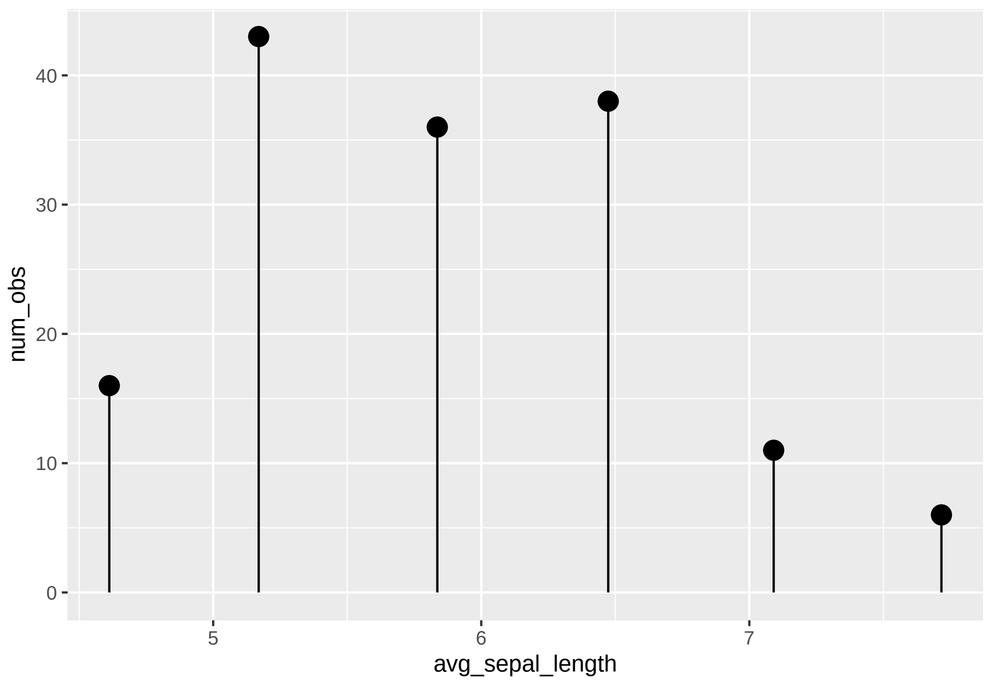
A scatter plot can be very useful graphic, but it can be difficult to visualize the distribution shape we are trying to represent in this example. To aid in that visualization, we will add vertical lines to our graphic via the `geom_linerange()` function. In `geom_linerange()` we must specify an `x` aesthetic and two separate y-axis aesthetics, `ymin` and `ymax`. As their names suggest, `ymin` displays the minimum value on the y-axis and `ymax` shows the maximum value on the y-axis. `geom_linerange()` then draws a vertical line connecting the y-axis aesthetics. Since we want to use the vertical lines to represent a histogram, set `ymin` to be 0 and `ymax` to equal `num_obs`. Thus, a vertical line will provide a comparable visualization as the bar chart from Problem 3.

PROBLEM Complete the code chunk below by correctly assigning the y-axis aesthetics.

Note: this style of figure is sometimes referred to as a stem plot. They are useful in certain contexts, but when creating histograms I prefer the default ggplot2 `geom_histogram()` or `geom_freqpoly()` style. We used the stem plot in this problem to introduce additional geoms and to continue demonstrating the `group_by()` and `summarize()` functions.

```r
iris_group %>%
  ggplot(mapping = aes(x = avg_sepal_length)) +
  geom_linerange(mapping = aes(ymin = 0,
                               ymax = num_obs)) +
  geom_point(mapping = aes(y = num_obs),
             size = 4)
```

SOLUTION



4f)

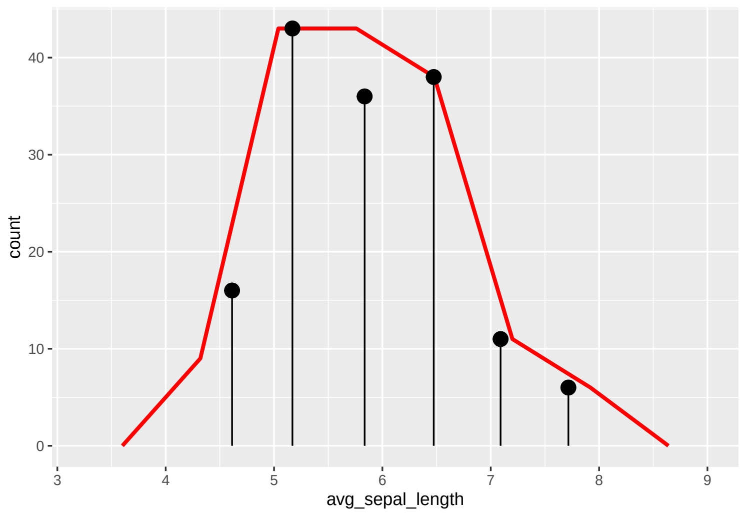
Let's now compare our manual histogram to ggplot's histogram. Our stem plot simplified histogram visualized the counts at the average `Sepal.Length` value within each bin. `geom_freqpoly()` plots the counts at the bin midpoint. Thus, assuming the bin center and the (empirical) average within a bin are similar, we should have an "apples-to-apples" comparison between `geom_freqpoly()` and our stem plot.

The graphic we wish to make does not just contain multiple layers but multiple datasets! The reason why is because we have to allow `geom_freqpoly()` to operate on the original non-grouped and non-summarized dataset. We can override the data associated with a geom by supplying an alternative dataset to the `data` argument. In this way, we can create complex figures which make use of various datasets.

PROBLEM Complete the code chunk below by setting the `data` argument to `geom_freqpoly()` to be `my_iris`. Specify the appropriate variable to the `x` aesthetic within the `geom_freqpoly()` call. Assess how well our manual histogram performed relative to the native ggplot histogram. Were we close? What are the differences?

```r
iris_group %>%
  ggplot(mapping = aes(x = avg_sepal_length)) +
  geom_freqpoly(data = my_iris,
                mapping = aes(x = Sepal.Length ),
                bins = 6, color = "red", size = 1.15) +
  geom_linerange(mapping = aes(ymin = 0,
                               ymax = num_obs)) +
  geom_point(mapping = aes(y = num_obs),
             size = 4)
```

SOLUTION



Describe the differences between the two here.

Our manual plot is fairly close to the plot generated by the native ggplot. There are points where it is lower than that of the native ggplot, but also has points that are spot on with the native ggplot.

Problem 5

So far, we have used several different types of geometric objects. 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). Compared with the histogram which focuses on displaying the *shape* of the distribution, the boxplot allows us to visually relate the median with the 25th and 75th quantiles, as well as outliers. We get an idea about the central tendency of the variable, as well as a rough guide on the "meaningful" range.

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

5a)

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

```r
diamonds %>% glimpse()
```

SOLUTION

```
## Observations: 53,940
## Variables: 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, Ve...
## $ 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, 6...
## $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62, 5...
## $ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340, 3...
## $ 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...
```

5b)

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 then 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.

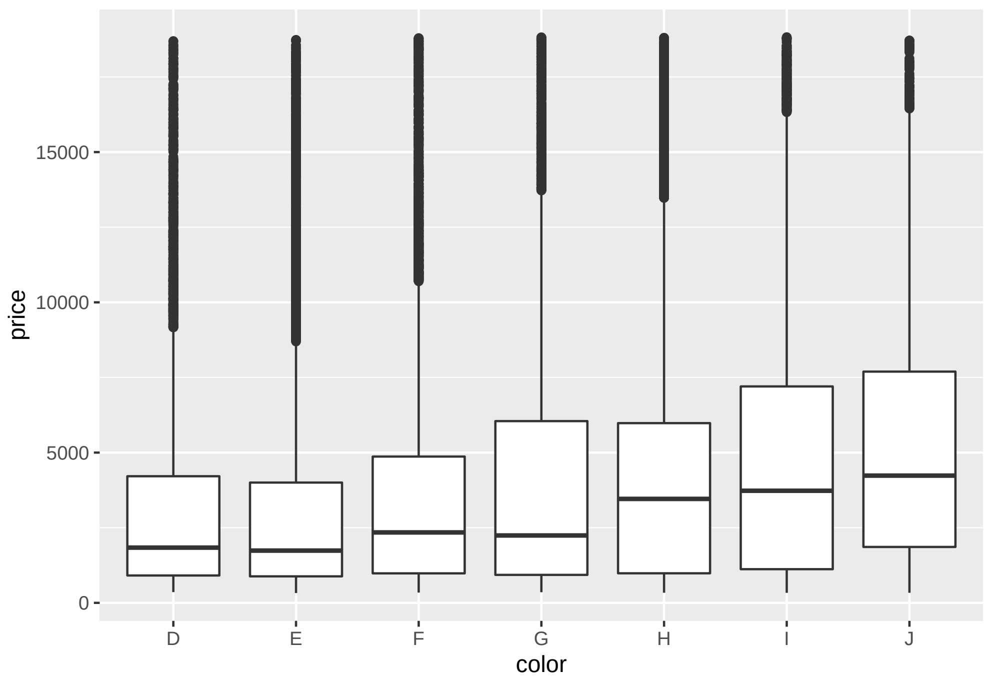
The first three of these are all factors (the `"ord"` data type is a special ordered factor which as the name states has a natural ordering of the categorical levels) within the `diamonds` dataset, while `carat` is a `"numeric"` variable. With a boxplot, we group a continuous variable based on a categorical 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`.

PROBLEM 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?

```
diamonds %>% ggplot(mapping = aes(x = color, y = price)) + geom_boxplot()
```

SOLUTION



Discuss your conclusions here.
The median price of diamonds is typically higher than the color classification of diamonds.

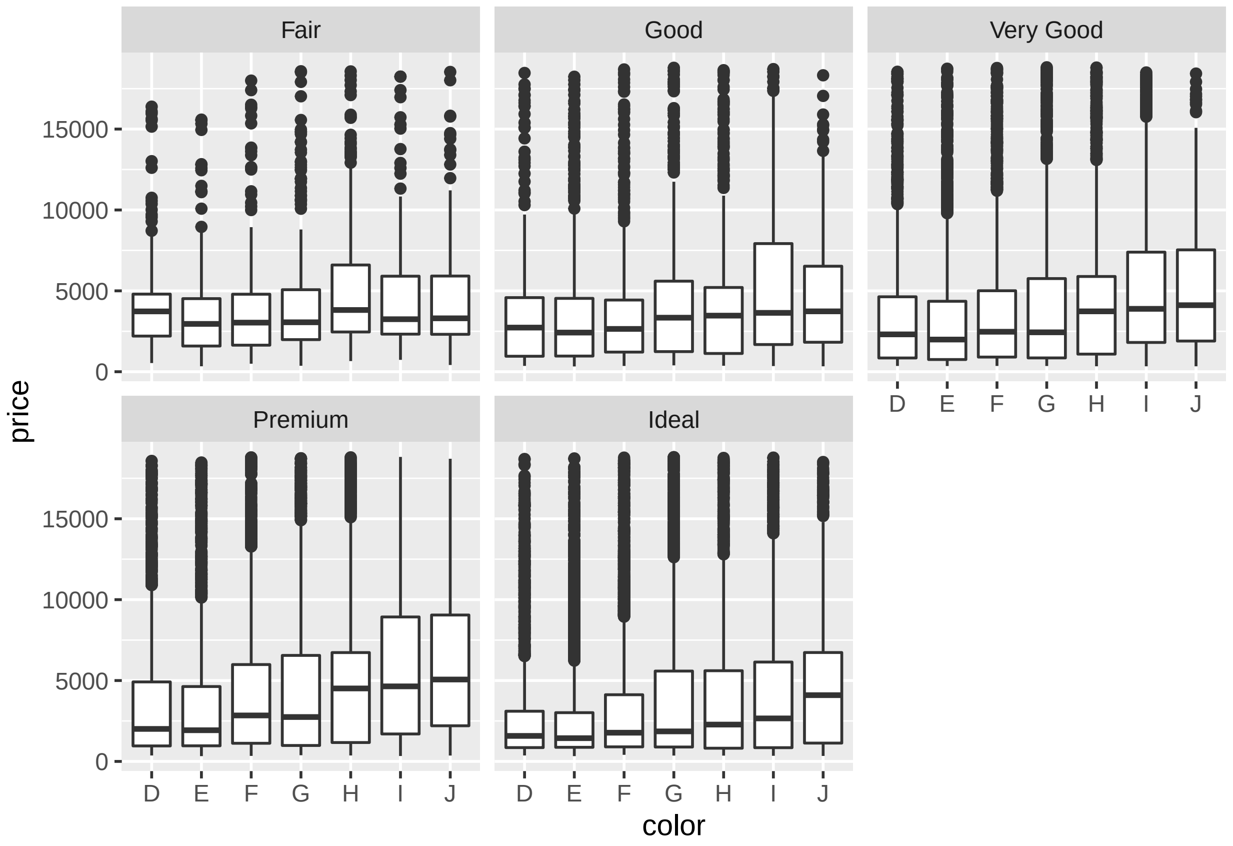
5c)

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

PROBLEM Add the `facet_wrap()` call to the code from Problem 5b), and set the facetting variable to be `cut`.

```
diamonds %>% ggplot(mapping = aes(x = color, y = price)) +
  geom_boxplot() + facet_wrap(~cut)
```

SOLUTION



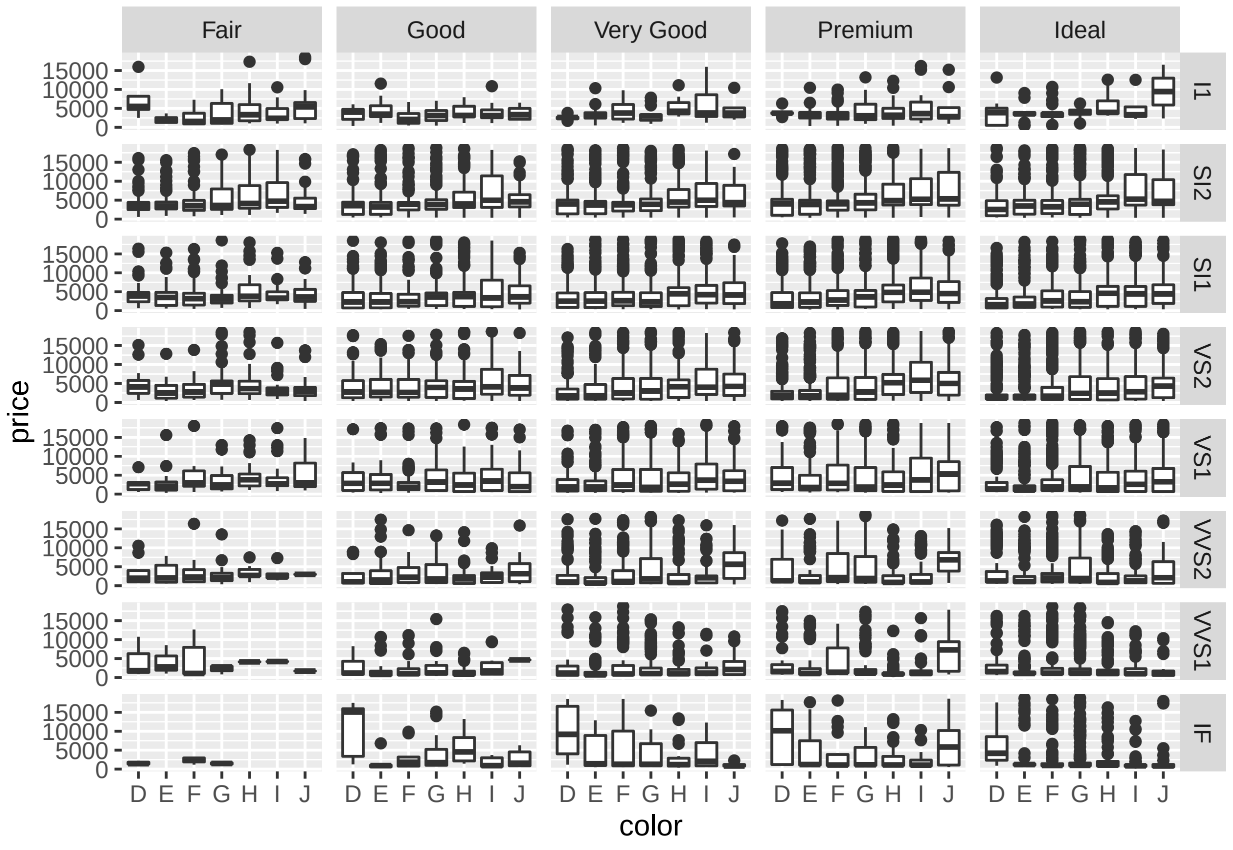
5d)

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).

PROBLEM 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.

```r
diamonds %>% ggplot(mapping = aes(x = color, y = price)) +
  geom_boxplot() + facet_grid(clarity~cut)
```

SOLUTION



5e)

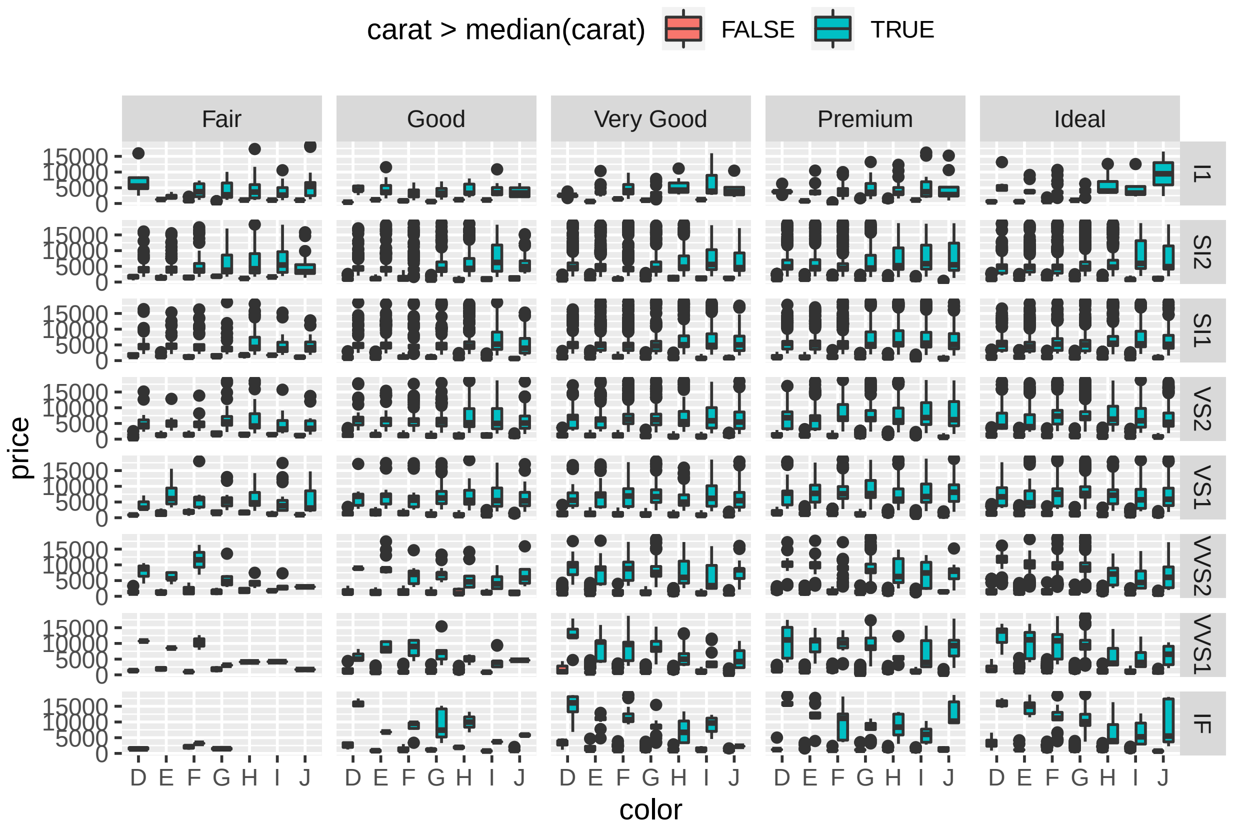
The resulting figure in Problem 5d) includes 3 out of the 4 C's. The remaining variable, `carat`, is not categorical. To include `carat` in our figure, let's discretize it and compare two boxplots side-by-side at each `color` level within each `clarity` and `cut` subplot combination. For now, we will keep things simple and break up `carat` based on if an observation has a value greater than the median `carat` value.

PROBLEM 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.**

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

SOLUTION



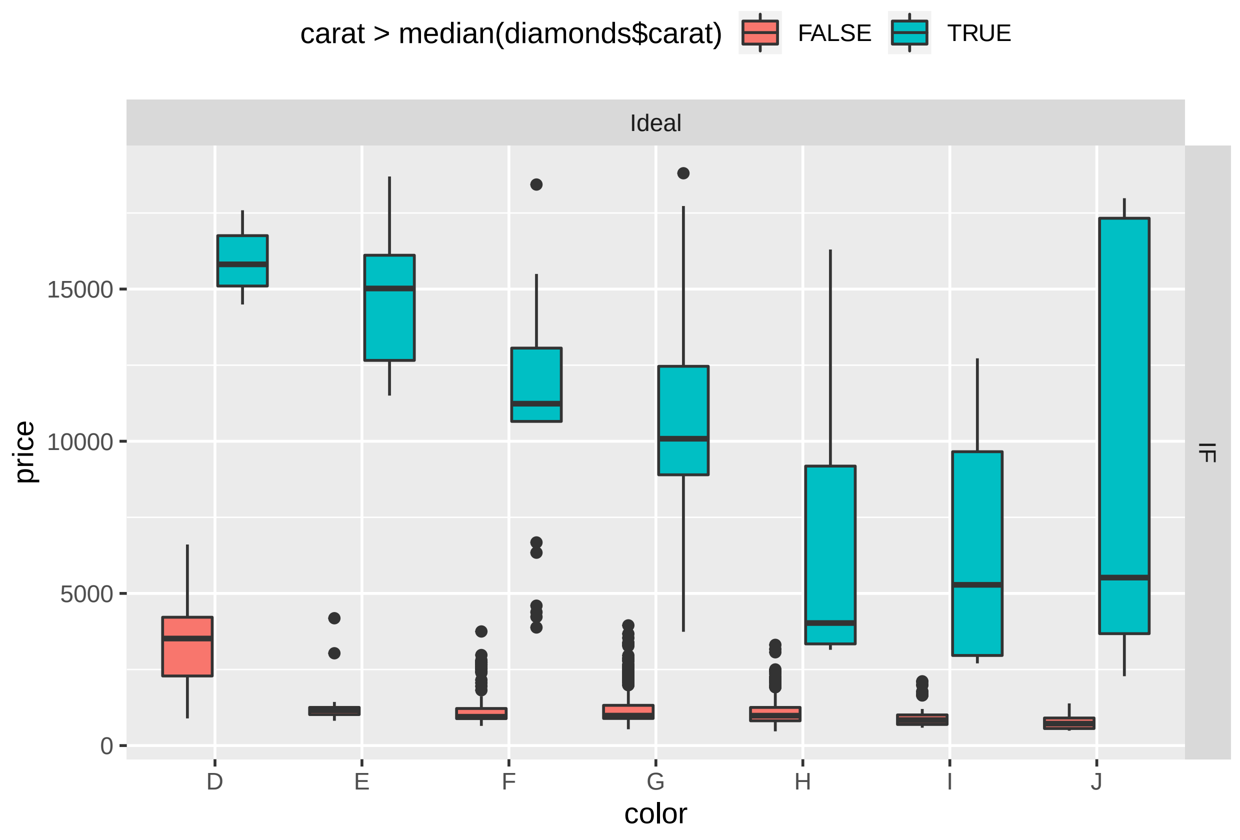
5f)

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()`.

PROBLEM Pipe `diamonds` into `filter()` and perform the necessary operation. Pipe the resulting dataset into the same `ggplot2` function calls used in Problem 5e), 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)`.

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

SOLUTION



5g)

PROBLEM Discuss the differences between the trends shown in the resulting figure in Problem 5f) with the trends shown in the figure in Problem 5b).

SOLUTION The set of diamonds that have an Ideal Cut, and IF Clarity, whose carat is above the average carat, as the color classification increases (alphabetically), the average price will generally decrease.

Problem 6

We can create LaTeX style math expressions within R Markdown. To get more comfortable writing these type of expressions, we will work examples similar to those discussed in the calculus review in the first week of lecture.

Quick introduction to LaTeX In R Markdown, math notation and expressions can be displayed inline by typing expressions between dollar signs. For example, typing `$x$` displays $x$ in the rendered document. If we instead want to display a bold face variable (to represent a vector) we will need to include the LaTeX style function `\mathbf()` between the dollar signs. Thus, typing `$\mathbf{x}$` renders $\mathbf{x}$ in the document.

LaTeX style expressions provide a lot of flexibility and enable us to write out expressions just like we would on paper. To show a subscript "attached" or "associated" with variable such as $x_1$, you must use the underscore character `_` next to that variable. Thus, type `$x_1$` to display $x_1$. If you have a relatively complicated expression you want to show within the subscript, you can wrap curly braces `{}` around the subscript expression. For example to display $x_{i,j,k,l}$ simply type `$x_{i,j,k,l}$`.

Superscripts, such as x^2, are displayed with the `^` operator. Thus, to show the square of the variable $x$, type `$x^2$`. As with subscripts, you can wrap complicated expressions within curly braces. To show x^{t+2}, type `$x^{t+2}$`. Subscripts and superscripts can also be combined together, such as with x_n^2. When combining subscripts and superscripts, it's recommended to wrap each expression within curly braces, even if the expression is simple. Therefore, to display x_{n}^{2}, type `$x_{n}^{2}$`.

We can display variables and complete expressions within parentheses, such as $(x + a)$, several different ways. First, you can simply type `$(x + a)$` which produces $(x + a)$. Another option though is to use the following commands `$\left( x + a \right)$` which also produces $\left(x + a \right)$. At first glance these seem to be the same, but the `\left( \right)` commands allow the parantheses to be dynamically sized around the expression they contain. For more information on different types of brackets and parantheses, please see the following reference from Overleaf.

We can also display equations within "equation blocks" using two dollar signs to surround the expression of interest. The equation within the block is rendered beneath text in the rendered report. We can thus draw special focus to mathematical equations and expressions displayed this way.

As an example, consider a function of two variables, $x_1$ and x_2, shown in the equation block below:

$$f\left(x_1, x_2 \right) = a x_{1}^{b} + c x_{2}^{d}$$

To create the above equation, type the following:

```
$$ f\left(x_1, x_2 \right) = a x_{1}^{b} + c x_{2}^{d} $$
```

Reading the un-rendered LaTeX expression may seem daunting at first, and getting used to working with LaTeX math coding can be challenging. However, please note that the above expression combines each of the previously described elements together. Each individual element, such as a subscript or superscript is relatively straight forward. Thus, when starting out with LaTeX, break a complicated expression into separate components or elements. Get those elements correct, and then "assemble" them into the final expression of interest.

Let's now get additional practice by working through the partial derivatives of the above function.

6a)

To write out the derivatives, we need to understand how to type fractions in LaTeX notation. A fraction consists of three parts. The first is the phrase `\frac`. The remaining two parts of a fraction are two sets of curly braces. The complete syntax is `\frac{}{}`. The numerator is typed within the first set of curly braces and the denominator is typed within the second set of curly braces. Thus, the formulation for a fraction is:

`\frac{<numerator expression>}{<denominator expression>}`.

As an example, one half written as a fraction is `$\frac{1}{2}$`. The rendered LaTeX expression looks like $\frac{1}{2}$.

PROBLEM Write out the partial derivative of f with respect to x_1. To receive full credit you must use the `\frac{}{}` to write out the partial derivative of f with respect to x_1 on the left hand side of =. The right hand side of = must be your expression for the partial first derivative.

NOTE: The "partial" operator, ∂ is created by typing `\partial`.

SOLUTION

$$\frac{\partial f}{\partial x_1} = abx_1^{b-1}$$

6b)

Now write out the expression for the partial derivative of f with respect to x_2.

PROBLEM Write out the partial derivative of f with respect to x_2. To receive full credit you must use the `\frac{}{}` to write out the partial derivative of f with respect to x_2 on the left hand side of =. The right hand side of = must be your expression for the partial first derivative.

NOTE: The "partial" operator, ∂ is created by typing `\partial`.

SOLUTION

$$\frac{\partial f}{\partial x_2} = cdx_2^{d-1}$$

6c)

As discussed in lecture, we also need to work with second derivatives.

PROBLEM Write out the partial second derivative of f with respect to x_1. Remember that the second derivative is the derivative of the first derivative.

SOLUTION

$$\frac{\partial^2 f}{\partial {x_2}^2} = ab(b-1)x_1^{b-2}$$

6d)

The second derivative in 6c) represents the rate-of-change with respect to x_1 of the rate-of-change with respect to x_1. We can also consider the cross-derivative, which examines the rate-of-change with respect to x_2 of the rate-of-change with respect to x_1.

PROBLEM What is the cross derivative equal to, for the example function of two variables? Write out the expression within an equation block below.

SOLUTION

$$\frac{\partial^2 f}{\partial x_1 \partial x_2} = 0$$

6e)

In lecture, we also reviewed important linear algebra concepts. We discussed the inner product, and how we can write it out two ways. One approach is with vector-vector math, while the other involves a summation. For the next several problems you will work with a vector $\mathbf{x}$ which is a $(N \times 1)$ column vector.

PROBLEM Write out the vector-vector operation for the inner product of the vector x with itself. Place your expression within an equation block below. You do not need to write out the elements of the vectors.

SOLUTION

$$\mathbf{x}^T \mathbf{x} = x_1^2 + x_2^2 + ... + x_N^2$$

6f)

The summation approach to writing the inner product is displayed for you below:

$$\mathbf{x}^T \mathbf{x} = \sum_{n=1}^{N} (x_n^2)$$

The summation expression consists of three components. The first is the Sigma character, which is created by the typing `\sum`. If we want the sigma character to appear in line, we wrap dollar signs around it, `$\sum$`, and if we want it to appear within an equation block we wrap two dollar signs, `$$\sum$$`. The $n = 1$ and N characters are displayed below and above the Sigma character with the `_` and `^` special characters, respectively.

PROBLEM Write out the summation approach to the inner product within an equation block below.

SOLUTION

$$\mathbf{x}^T \mathbf{x} = \sum_{n=1}^{N} (x_n^2)$$

6g)

In the 6e), the vector $\mathbf{x}$ was defined to be a $(N \times 1)$ column vector. What are the dimensions of the inner product of $\mathbf{x}$ with itself? What would the dimensions of the outer product of $\mathbf{x}$ be with itself?

PROBLEM Write the dimensions of the inner product and the dimensions of the outer product of $\mathbf{x}$ with itself. Note that to display the multiplication sign you can type `\times`.

SOLUTION

$$Inner : \mathbf{x}^T\mathbf{x} = (1 \times 1)$$
$$Outer : \mathbf{x}\mathbf{x}^T = (N \times N)$$

6h)

We will frequently use greek letters when writing expressions and equations. Thankfully, it is rather simple to type greek letters in LaTeX. All you have to do is type the escape character \ in front of the greek word. For example to display δ, just type `$\delta$`. To show the capital letter Δ use a capital `D` instead of a lower case `d`: `$\Delta$`.

PROBLEM Practice typing the greek letters listed in the instruction file in an equation block below. Separate each letter by the LaTeX term `\cdot` which displays the "dot-multiply" operator, $\cdot$.

Hint: the greek words are alpha, beta, gamma, epsilon, mu, sigma, and psi.

SOLUTION

$$\alpha \cdot \beta \cdot \gamma \cdot \epsilon \cdot \mu \cdot \sigma \cdot \psi$$