Week 12 R Packages

```
knitr::opts_knit$set(root.dir = "D:/Projects/Introduction-to-R/Notebooks")
library(validate)
```

```
## Warning: package 'validate' was built under R version 4.0.3
```

```
library(magrittr)
```

Weekly Learning Objectives

Create an R script with the following components:

1. Create a new RMarkdown document
2. Insert the R code produced for the Central Limit Theorem example (make sure echo is off)
3. Include the four graphs as part of the document
4. Create a R function within the document to generate basic statistics on one of the sample built-in data sets (make sure echo is turned on)
5. Use Knitr to generate a PDF or document file. If necessary install any dependencies
6. Create a R script that utilizes at least five functions (each) found in these packages:

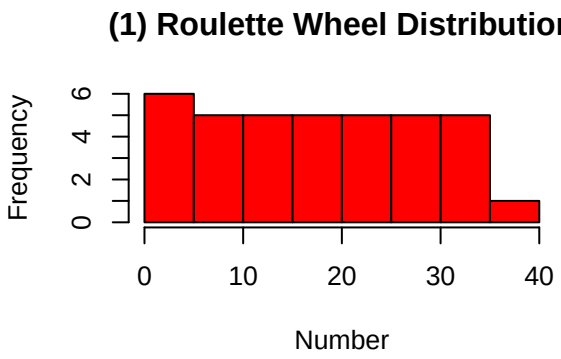
- Dplyr - Data Manipulation
- Tidyr - Data Manipulation
- GGPlot2 - Graphics

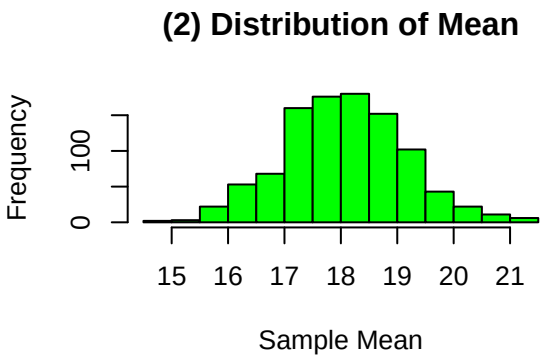
1. Create a new RMarkdown document.

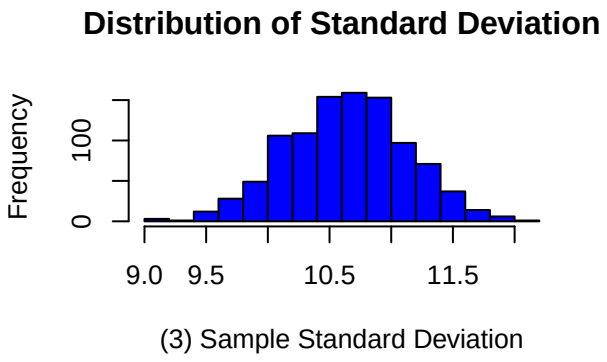
The document has been created, and I am now completing the week 12 assignment within R Markdown.

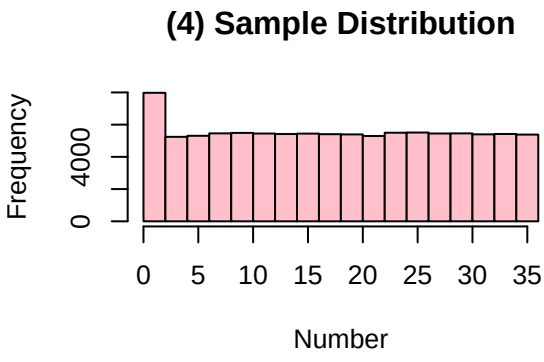
2. and 3. CLT and graphs

The code was input with ECHO turned off so it will not appear in the document. The resulting graphs display the following: **(1)** Shows the distribution of numbers on a roulette wheel for reference. **(2)** Shows the distribution of the mean for the 1,000 simulations. **(3)** Shows the distribution of the standard deviation among the 1,000 simulations. **(4)** Shows the distribution of numbers resulting in all simulated spins of the wheel. Notice the first bar is higher in graph (4). This is because it consists of three numbers (0,1 and 2), while the other bars each consist of only two numbers.









4. Create a R function within the document to generate basic statistics on one of the sample built-in data sets.

*** Please make sure the ECHO is turned on for this piece of code.**

I will genererate a summary of basic statistics from the "trees" data set.

```
summary(trees)
```

```
##      Girth           Height       Volume
##  Min.   : 8.30   Min.   :63   Min.   :10.20
##  1st Qu.:11.05   1st Qu.:72   1st Qu.:19.40
##  Median :12.90   Median :76   Median :24.20
##  Mean   :13.25   Mean   :76   Mean   :30.17
##  3rd Qu.:15.25   3rd Qu.:80   3rd Qu.:37.30
##  Max.   :20.60   Max.   :87   Max.   :77.00
```

5. Use Knitr to generate a PDF or document file.

*** If necessary, install any dependent libraries in R-Studio.**

I will use Knitr to generate an HTML document.

6. Create a R script that utilizes at least five functions (each) found in these packages:

*** Dplyr - Data Manipulation**

*** Tidyr - Data Manipulation**

*** Ggplot2 - Graphics**

dplyr As I explore the functions of dplyr, I will use echo = FALSE.

Function 1: filter() Revisiting the "trees" dataset, I can use the filter() function to find only the rows that exceed the mean of all three variables. This will be done with the call, filter(trees, Girth > 13.25, Height > 76, Volume > 30.17). Only 9 of the 31 trees meet this criteria:

```
##   Girth Height Volume
## 1  14.0     78   34.5
## 2  14.2     80   31.7
## 3  16.3     77   42.6
## 4  17.3     81   55.4
## 5  17.5     82   55.7
## 6  17.9     80   58.3
## 7  18.0     80   51.5
## 8  18.0     80   51.0
## 9  20.6     87   77.0
```

Function 2: arrange() The trees data is presented in the order of their Girth by default. With this function, I can change that to prioritize Height first and volume second with the function, arrange(trees,Height,Volume).

```
##    Girth Height Volume
## 1    8.8     63   10.2
## 2   13.8     64   24.9
## 3    8.6     65   10.3
## 4   11.0     66   15.6
## 5   11.7     69   21.3
## 6    8.3     70   10.3
## 7   13.7     71   25.7
## 8   10.5     72   16.4
## 9   16.0     72   38.3
## 10  12.9     74   22.2
## 11  14.5     74   36.3
## 12  11.0     75   18.2
## 13  12.0     75   19.1
## 14  11.2     75   19.9
## 15  11.4     76   21.0
## 16  11.4     76   21.4
## 17  16.3     77   42.6
## 18  14.0     78   34.5
## 19  11.3     79   24.2
## 20  11.1     80   22.6
## 21  14.2     80   31.7
## 22  18.0     80   51.0
## 23  18.0     80   51.5
## 24  17.9     80   58.3
## 25  10.7     81   18.8
## 26  17.3     81   55.4
```

```
## 27  17.5     82   55.7
## 28  10.8     83   19.7
## 29  12.9     85   33.8
## 30  13.3     86   27.4
## 31  20.6     87   77.0
```

Function 3: select() The select() function only displays the called columns. I will use this along with the filter() function to only display the Volume for trees that have Volume greater than 40:

```
##   Volume
## 1   42.6
## 2   55.4
## 3   55.7
## 4   58.3
## 5   51.5
## 6   51.0
## 7   77.0
```

Function 4: mutate() Let's explore the "women" data set. I will add a column titled 'BMI', which will calculate the body mass index with the formula, 703 multiplied by weight and divided by height squared:

```
##    height weight      BMI
## 1      58    115 24.03240
## 2      59    117 23.62856
## 3      60    120 23.43333
## 4      61    123 23.23811
## 5      62    126 23.04318
## 6      63    129 22.84883
## 7      64    132 22.65527
## 8      65    135 22.46272
## 9      66    139 22.43274
## 10     67    142 22.23791
## 11     68    146 22.19680
## 12     69    150 22.14871
## 13     70    154 22.09429
## 14     71    159 22.17358
## 15     72    164 22.23997
```

Function 5: summarise() A helpful way to use the summarise() function would be in connection with the mutate() function, as you can gain summary information on a new column. I was able to use this to find the mean of the new BMI column that was created:

```
##   BMI.mean
## 1 22.72443
```

tidyr As I explore the functions for tidyr, I will leave echo turned on.

Function 1: separate() The data set "smiths" from tidyr has Full names listed under the column title, subject. This can be split into two columns for first and last names using the separate() function:

```
  smiths
```

```
## # A tibble: 2 x 5
##   subject     time   age weight height
##   <chr>      <dbl> <dbl>  <dbl>  <dbl>
## 1 John Smith     1    33     90   1.87
```

```
## 2 Mary Smith     1    NA     NA   1.54
```

```
  separate(smiths, subject, c("First", "Last"), sep = " ")
```

```
## # A tibble: 2 x 6
##   First Last   time   age weight height
##   <chr> <chr> <dbl> <dbl>  <dbl>  <dbl>
## 1 John  Smith     1    33     90   1.87
## 2 Mary  Smith     1    NA     NA   1.54
```

Function 2: unite() If the smiths started with separated names, and it would be more functional with the names combined into one column, the opposite action can be taken with the unite() function. I named the new version of the table "smith2" and will convert it back. I will title the column "Name" and separate the names with an underscore:

```
  smith2 <- separate(smiths, subject, c("First", "Last"), sep = " ")
  smith2
```

```
## # A tibble: 2 x 6
##   First Last   time   age weight height
##   <chr> <chr> <dbl> <dbl>  <dbl>  <dbl>
## 1 John  Smith     1    33     90   1.87
## 2 Mary  Smith     1    NA     NA   1.54
```

```
  unite(smith2, "Name",c(First, Last), sep = "_")
```

```
## # A tibble: 2 x 5
##   Name        time   age weight height
##   <chr>      <dbl> <dbl>  <dbl>  <dbl>
## 1 John_Smith     1    33     90   1.87
## 2 Mary_Smith     1    NA     NA   1.54
```

Function 3: spread() The data set "population" from tidyr has population data of countries from 1995 to 2013. It is compiled in only 3 columns, causing it to be 4,060 rows. By spreading the data into columns based on years, this number can be reduced to 219 rows with 20 columns:

```
  population
```

```
## # A tibble: 4,060 x 3
##    country      year population
##    <chr>       <int>      <int>
##  1 Afghanistan  1995   17586073
##  2 Afghanistan  1996   18415307
##  3 Afghanistan  1997   19021226
##  4 Afghanistan  1998   19496836
##  5 Afghanistan  1999   19987071
##  6 Afghanistan  2000   20595360
##  7 Afghanistan  2001   21347782
##  8 Afghanistan  2002   22202806
##  9 Afghanistan  2003   23116142
## 10 Afghanistan  2004   24018682
## # ... with 4,050 more rows
```

```
  spread(population, year, population)
```

```
## # A tibble: 219 x 20
##    country `1995` `1996` `1997` `1998` `1999` `2000` `2001` `2002` `2003` `2004`
##    <chr>    <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>
```

```
##  1 Afghan~ 1.76e7 1.84e7 1.90e7 1.95e7 2.00e7 2.06e7 2.13e7 2.22e7 2.31e7 2.40e7
##  2 Albania 3.36e6 3.34e6 3.33e6 3.33e6 3.32e6 3.30e6 3.29e6 3.26e6 3.24e6 3.22e6
##  3 Algeria 2.93e7 2.98e7 3.03e7 3.08e7 3.13e7 3.17e7 3.22e7 3.26e7 3.30e7 3.35e7
##  4 Americ~ 5.29e4 5.39e4 5.49e4 5.59e4 5.68e4 5.75e4 5.82e4 5.87e4 5.91e4 5.93e4
##  5 Andorra 6.39e4 6.43e4 6.41e4 6.38e4 6.41e4 6.54e4 6.80e4 7.16e4 7.56e4 7.91e4
##  6 Angola  1.21e7 1.25e7 1.28e7 1.31e7 1.35e7 1.39e7 1.44e7 1.49e7 1.54e7 1.60e7
##  7 Anguil~ 9.81e3 1.01e4 1.03e4 1.05e4 1.08e4 1.11e4 1.14e4 1.17e4 1.20e4 1.23e4
##  8 Antigu~ 6.83e4 7.02e4 7.22e4 7.42e4 7.60e4 7.76e4 7.90e4 8.00e4 8.09e4 8.17e4
##  9 Argent~ 3.48e7 3.53e7 3.57e7 3.61e7 3.65e7 3.69e7 3.73e7 3.76e7 3.80e7 3.83e7
## 10 Armenia 3.22e6 3.17e6 3.14e6 3.11e6 3.09e6 3.08e6 3.06e6 3.05e6 3.04e6 3.03e6
## # ... with 209 more rows, and 9 more variables: `2005` <int>, `2006` <int>,
## #   `2007` <int>, `2008` <int>, `2009` <int>, `2010` <int>, `2011` <int>,
## #   `2012` <int>, `2013` <int>
```

Function 4: gather() The gather() function does the opposite of the spread() function. In this example, I will use the "table4a" data set from tidyr. It is a portion of data collected from the World Health Organization Global Tuberculosis Report. The gather() function will be used to stack years into a single column by creating a new column titled "TB_cases":

```
  table4a
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

```
  gather(table4a, year, TB_cases, '1999':'2000')
```

```
## # A tibble: 6 x 3
##   country     year  TB_cases
##   <chr>       <chr>    <int>
## 1 Afghanistan 1999       745
## 2 Brazil      1999     37737
## 3 China       1999    212258
## 4 Afghanistan 2000      2666
## 5 Brazil      2000     80488
## 6 China       2000    213766
```

Function 5: replace_na() The "smiths" data set contains missing values for Mary Smith. I will use the replace_na() function to replace "NA" with "unknown":

```
  smiths
```

```
## # A tibble: 2 x 5
##   subject     time   age weight height
##   <chr>      <dbl> <dbl>  <dbl>  <dbl>
## 1 John Smith     1    33     90   1.87
## 2 Mary Smith     1    NA     NA   1.54
```

```
  replace_na(smiths, list(age = "unknown", weight = "unknown"))
```

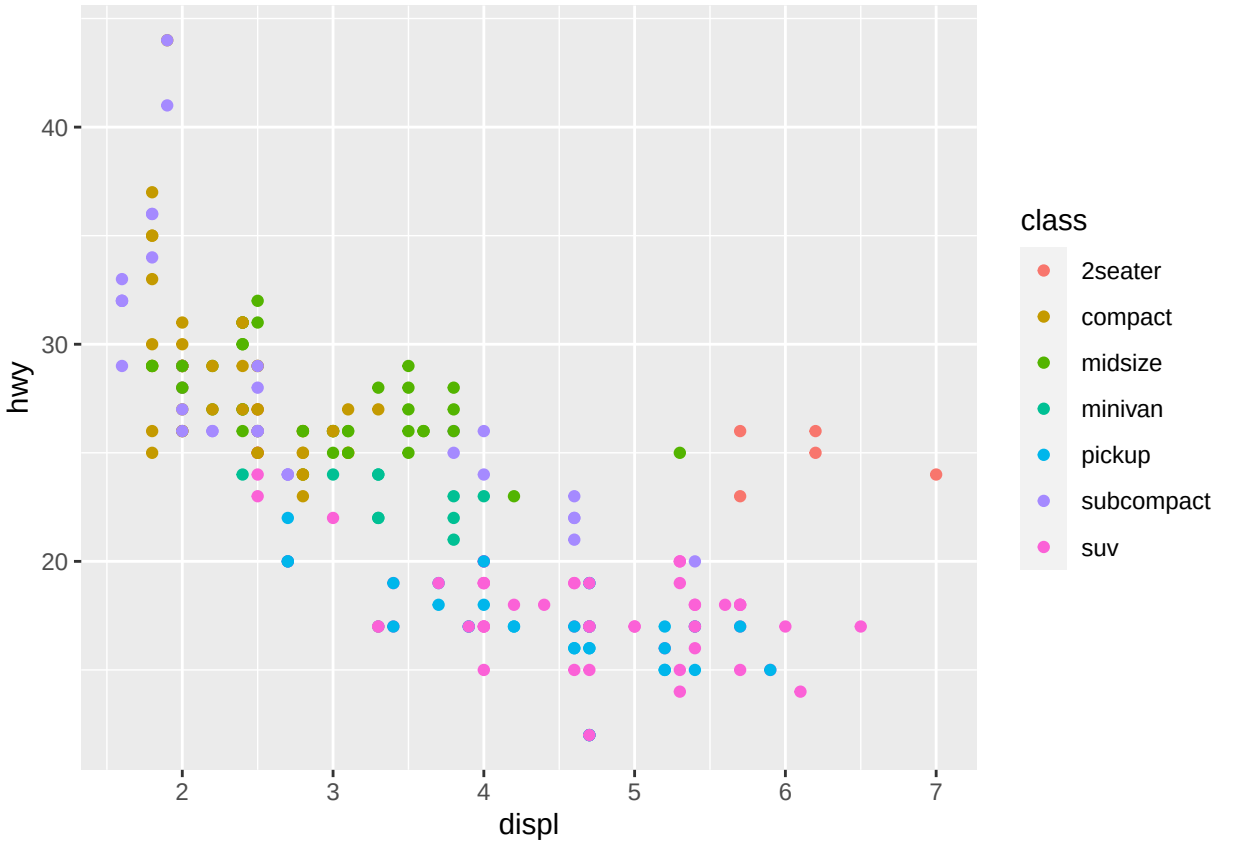
```
## # A tibble: 2 x 5
##   subject     time age     weight  height
##   <chr>      <dbl> <chr>   <chr>    <dbl>
## 1 John Smith     1 33      90        1.87
```

```
## 2 Mary Smith      1 unknown unknown   1.54
```

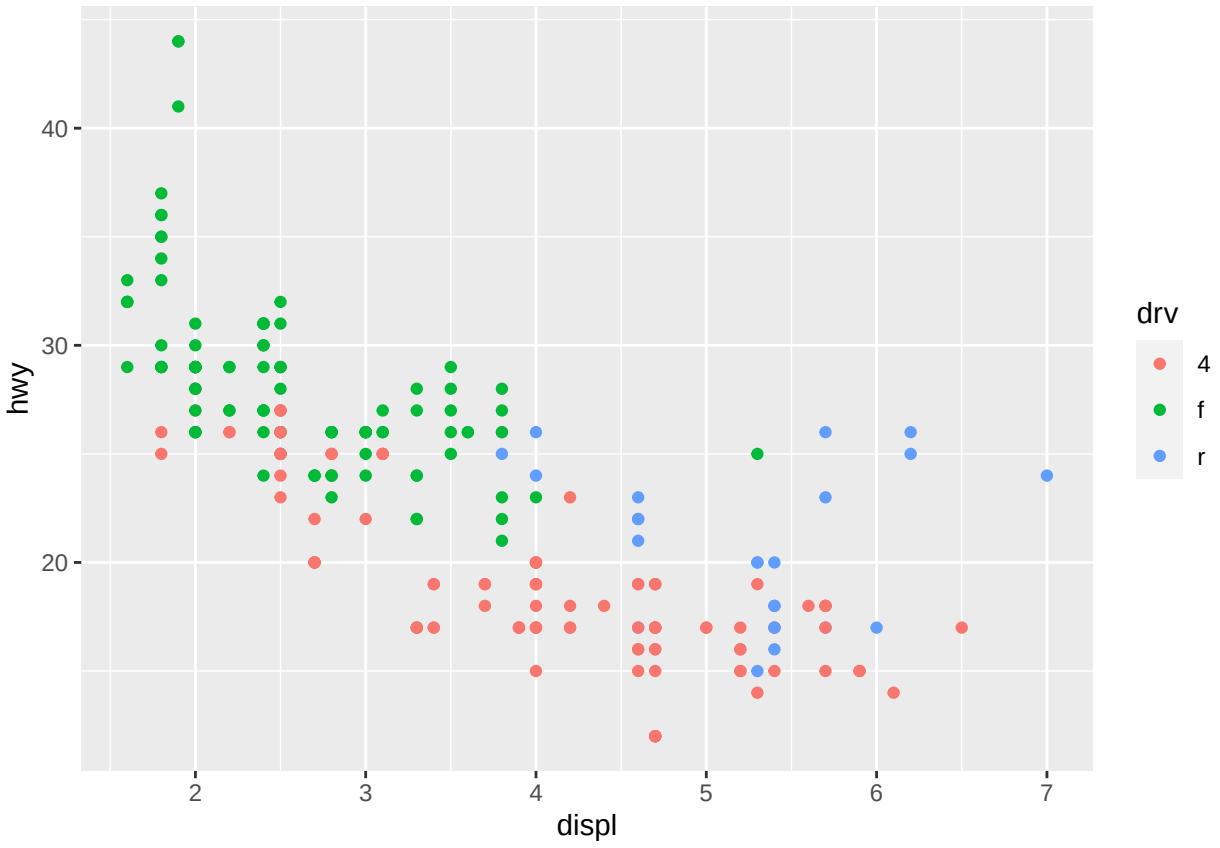
ggplot2 I will continue to leave echo turned on as I explore the ggplot2 package. ##### **Function 1: qplot()**

Using the "mpg" data ser, I will use the qplot() function to compare miles per gallon to the engine displacement in litres. I will create one plot color coordinated based on the class of car (midsize, compact, etc.). The other plot will be color coordinated based on a car being front wheel drive, back wheel drive, or 4 wheel drive:

```
qplot(displ, hwy, color = class, data = mpg)
```



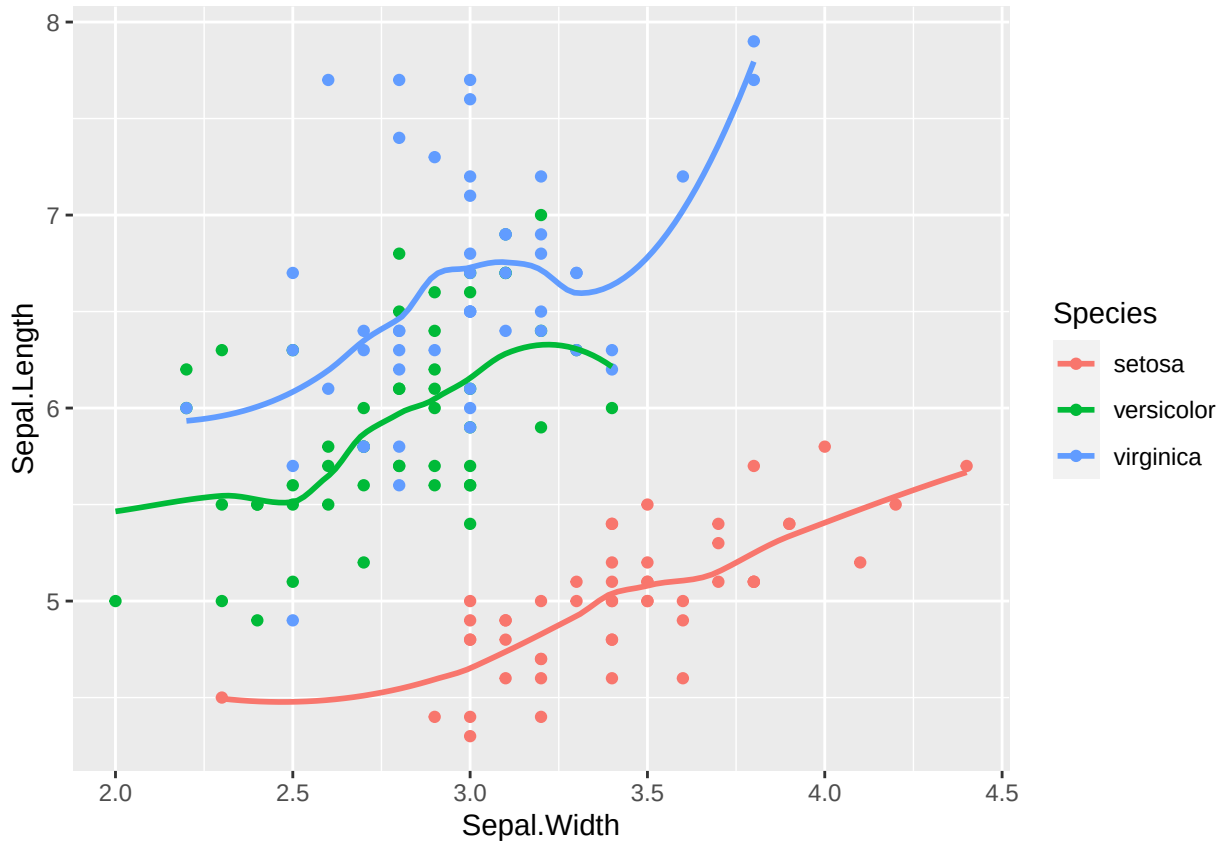

```
qplot(displ, hwy, color = drv, data = mpg)
```



Function 2: ggplot() + geom_point() + geom_smooth() Using the iris data set, I will compare sepal length to sepal width and include a trend line for each of the three species:

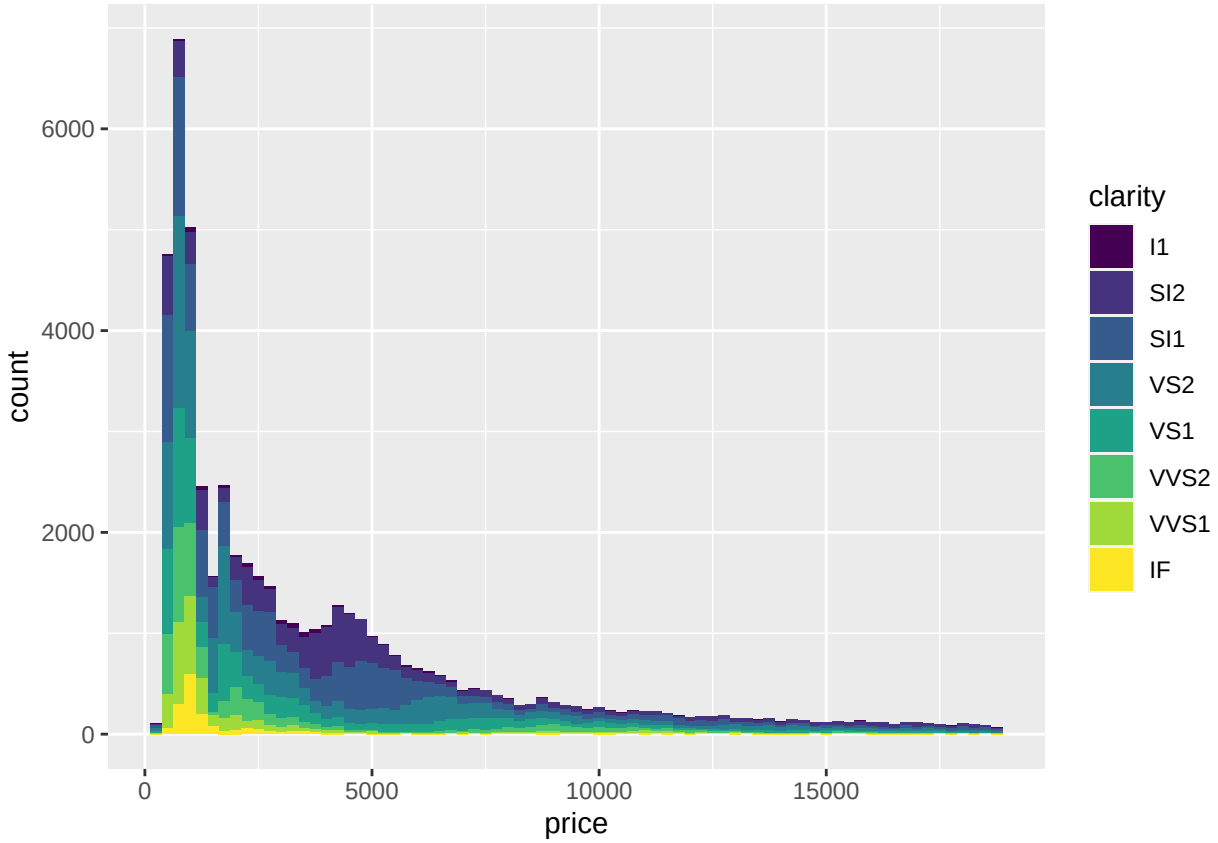
```
  ggplot(iris, aes(x=Sepal.Width, y=Sepal.Length, color = Species)) + geom_point() + geom_smooth(se=FALS
```

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



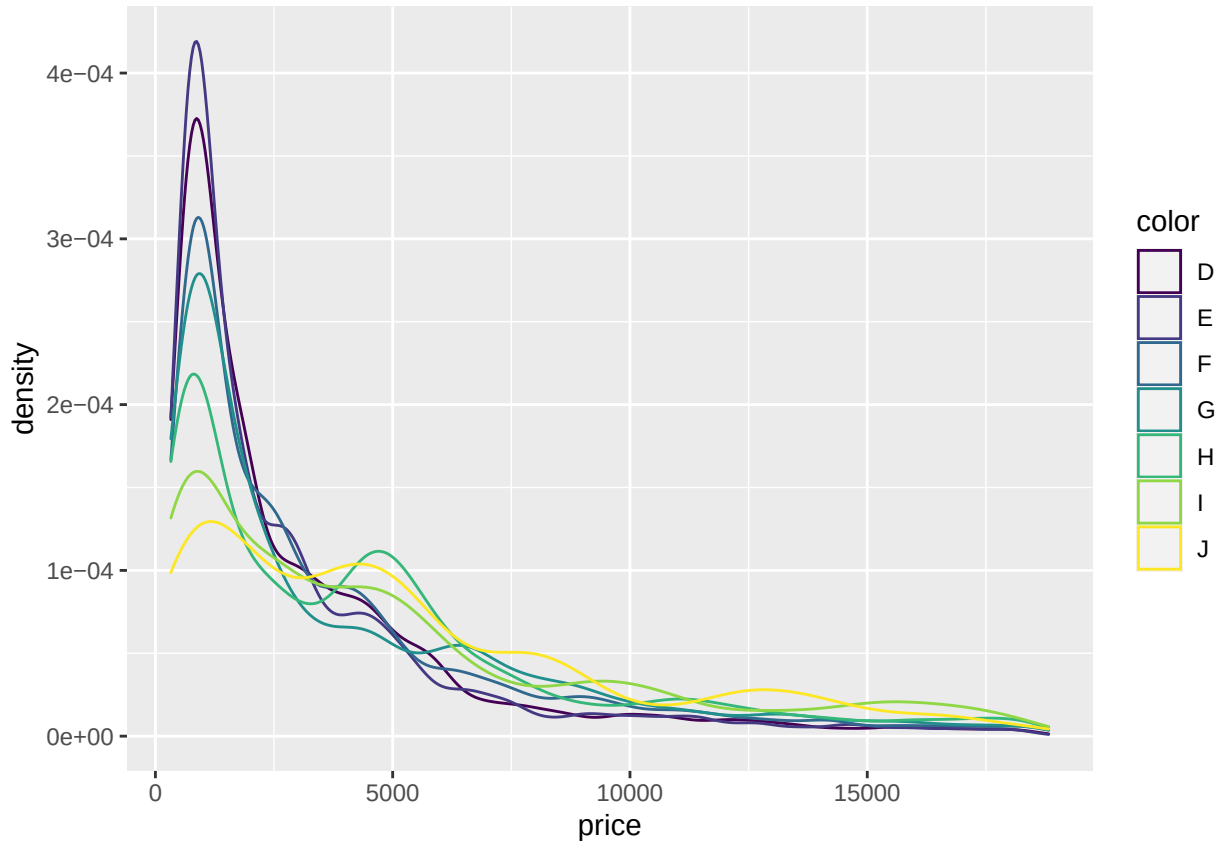
Function 3: ggplot() + geom_histogram() The "diamonds" data set contains the prices of 50,000 round cut diamonds. Using the ggplot() with the geom_histogram(), I will generate a histogram with price as the x axis, and the count as the y axis. I will add a fill to color code the clarity of the diamonds within the graph. I will also reduce the binwidth to 250:

```
ggplot(diamonds, aes(x=price, fill=clarity)) + geom_histogram(binwidth=250)
```



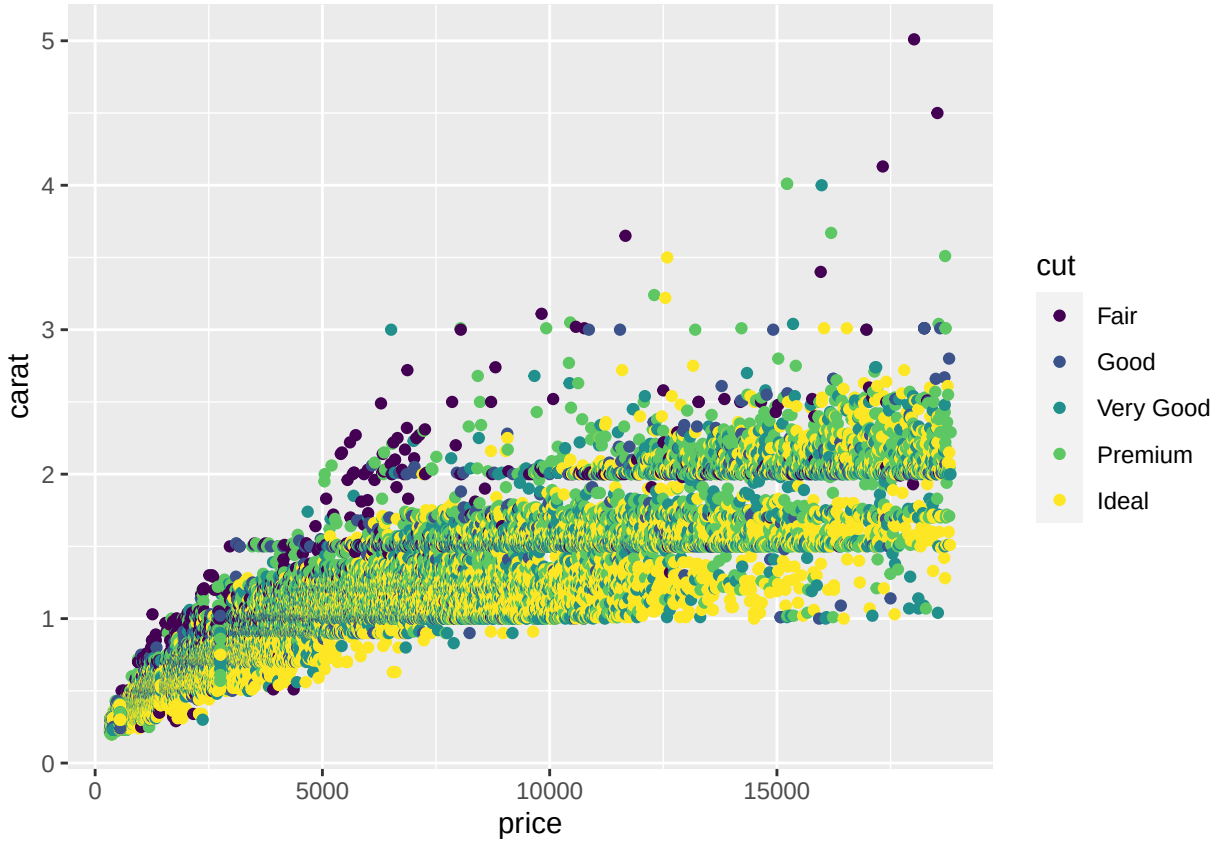
Function 4: ggplot() + geom_density() Staying with the "diamonds" data set, I will now generate a color coded plot to show the density of the price of a diamond based on the color of the diamond:

```
ggplot(diamonds, aes(x=price, color=color)) + geom_density()
```



Function 5: ggplot() + geom_point() Finally, I will compare price to the carat of the diamonds via a scatter plot, while also taking into account the cut of the diamond through color coding:

```
ggplot(diamonds,aes(price,carat)) + geom_point(aes(color=cut))
```


carat
price
0
1
2
3
4
5
0
5000
10000
15000
cut
Fair
Good
Very Good
Premium
Ideal