## Module 1 - Refresher Assignment

library(tidyverse)

## -- Attaching packages --------------------------------------------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.0 v purrr 0.3.2   
## v tibble 2.1.1 v dplyr 0.8.0.1  
## v tidyr 0.8.3 v stringr 1.4.0   
## v readr 1.3.1 v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

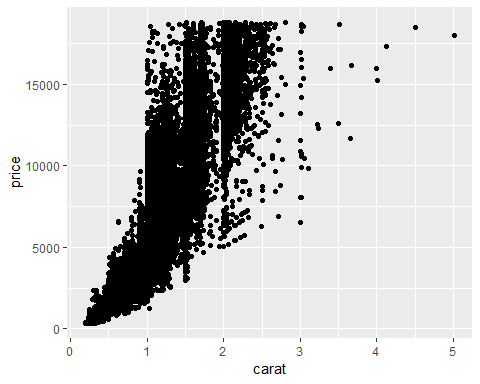
diamonddata = diamonds  
  
nrow(diamonddata)

## [1] 53940

ncol(diamonddata)

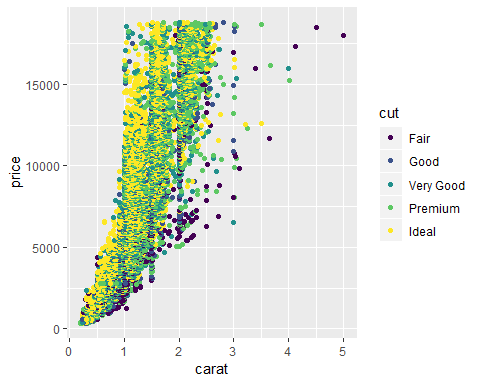
## [1] 10

ggplot(diamonddata, aes(x=carat, y=price))+  
 geom\_point()



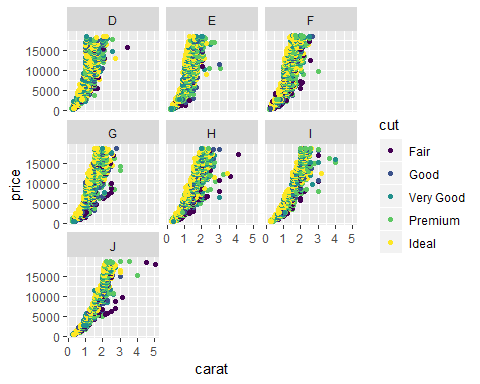
There is a positive relationship between carat and price in the Diamond dataset. As the carat size increases, the price also increases. Once the carat size gets to 1 carat and higher, the price starts to drastically increase.

ggplot(diamonddata, aes(x=carat, y=price, color = cut))+  
 geom\_point()



When adding the color variable to the plot, there is a wide variation of price based off the color variable. Most of the “fair” cuts are lower dollar amounts, except when you get to the high carats. The “ideal” cut ranges price wise depening on the carat, however more of the ideal cuts are higher in price.

ggplot(diamonddata, aes(x=carat, y=price, color = cut))+  
 geom\_point() +  
 facet\_wrap(~color)



The “color” variable creates an intersting relationship between cut, carat and price. By looking at the graphs, you can see that there are very few color “D” that are 2 carat or higher. As the color gets worse, the plot starts to shift to the right, meaning that as the color gets closer to “J” there are higher carat options. To have a diamond with a high carat and “D” color is very rare.

library(readr)  
inventory <- read\_csv("InventoryData.csv")

## Parsed with column specification:  
## cols(  
## `Item SKU` = col\_character(),  
## Store = col\_character(),  
## Supplier = col\_character(),  
## `Cost per Unit ($)` = col\_double(),  
## `On Hand` = col\_double(),  
## `Annual Demand` = col\_double()  
## )

str(inventory)

## Classes 'spec\_tbl\_df', 'tbl\_df', 'tbl' and 'data.frame': 13561 obs. of 6 variables:  
## $ Item SKU : chr "0100" "0100" "0100" "0100" ...  
## $ Store : chr "003480" "01611" "01611" "020109" ...  
## $ Supplier : chr "A" "B" "D" "B" ...  
## $ Cost per Unit ($): num 125.32 115.12 53.61 2.26 60.51 ...  
## $ On Hand : num 159 40 174 176 74 48 6 129 82 17 ...  
## $ Annual Demand : num 1693 351 1691 1559 733 ...  
## - attr(\*, "spec")=  
## .. cols(  
## .. `Item SKU` = col\_character(),  
## .. Store = col\_character(),  
## .. Supplier = col\_character(),  
## .. `Cost per Unit ($)` = col\_double(),  
## .. `On Hand` = col\_double(),  
## .. `Annual Demand` = col\_double()  
## .. )

summary(inventory)

## Item SKU Store Supplier   
## Length:13561 Length:13561 Length:13561   
## Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character   
##   
##   
##   
## Cost per Unit ($) On Hand Annual Demand   
## Min. : 0.0 Min. : 0.0 Min. : 0.0   
## 1st Qu.: 137.0 1st Qu.: 50.0 1st Qu.: 483.0   
## Median : 377.5 Median :101.0 Median : 965.0   
## Mean : 504.4 Mean :100.5 Mean : 966.2   
## 3rd Qu.: 775.5 3rd Qu.:151.0 3rd Qu.:1448.0   
## Max. :1982.3 Max. :200.0 Max. :2150.0

inventoryA = inventory %>% filter(Supplier== "A")  
  
nrow(inventoryA)

## [1] 3695

There are 3695 rows in the Inventory A dataset.

inventoryA = mutate(inventoryA, OnHandRatio = `On Hand` / `Annual Demand` )

The code above takes the inventoryA dataset and adds another column that is taking the “On Hand” number and dividing it by the “Annual Demand” number to create a new column named “OnHandRatio”.

avg\_cost = inventoryA %>%   
 group\_by(`Item SKU`) %>%   
 summarize(SKUAvgCost= mean(`Cost per Unit ($)`)) %>%  
 select(c(SKUAvgCost, `Item SKU`))

The topics that I found to be most challenging were the grouping of mutltiple commands to create a new dataset. Making sure you are doing them in the right order and that there are no errors in the code is probably the most difficult part. RStudio however makes it easier to detect where there are errors and its easy to keep rerunning that code chunk to see if you are manipulating the data correctly.