FinalProject

Jenelle Aming

5/17/2018

This is a tutorial of how to break analyze data in RStudio. The data set I chose to analyze is a data set of all the stocks in the stock market in day by day in 2016. The data set has the name, opening and closing prices, high and low prices, and the volume of the stock. This topic is extremely relevant in society and the economy. As you can see in the data, billions of stocks are traded daily. There are financial institutions that do indepth statistical analysis to try to predict the trends of the stocks. Hedge funds are corporations that invest people's money into the stock market. If these funds are investing their client's money and make a profit, the company keeps a share of the profit and the client initial investment grows. The more successful these hedgefunds are, the larger their portfolio of clients means they have more capital they have to invest and make a profit.

To read more about stocks see the link below: https://www.nerdwallet.com/blog/investing/stock-market-basics-everything-beginner-investors-know/ (https://www.nerdwallet.com/blog/investing/stock-market-basics-everything-beginner-investors-know/)

Link to the dataset: https://www.kaggle.com/kp4920/s-p-500-stock-data-time-series-analysis/data (https://www.kaggle.com/kp4920/s-p-500-stock-data-time-series-analysis/data)

Before we can begin, we need to read data into the R. The data is in a csv file, in order to get the data into R you must download the data set. To make things easy, I save the data set in the same folder as the project and then set that as my working directory. To do this in R, go to Session > Set Working Directory > Choose Directory > then chose the folder your project is in. Then the goal is to open the data into R so you can manipulate the data. The code below does this. First you need to load the tidyverse library in order to use the read_csv command. The code below loads the library, uses the read_csv command to read the file, and returns a data frame. I am using a pipline to name the data file that is being returned. I chose the name "stocks", and the arrow "<-" pointing to the name creates the pipeline. Remember to close the pipeline after use.

```
library(tidyverse)
```

```
## ── Attaching packages ──────────────────────────────────────────
───────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ ggplot2 2.2.1     ✔ purrr   0.2.4
## ✔ tibble  1.4.2     ✔ dplyr   0.7.4
## ✔ tidyr   0.8.0     ✔ stringr 1.2.0
## ✔ readr   1.1.1     ✔ forcats 0.2.0
```

```
## ── Conflicts ──────────────────────────────────────────────────────────
────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
stocks <- read_csv("all_stocks_1yr.csv")
```

```
## Parsed with column specification:
## cols(
##   Date = col_date(format = ""),
##   Open = col_double(),
##   High = col_double(),
##   Low = col_double(),
##   Close = col_double(),
##   Volume = col_integer(),
##   Name = col_character()
## )
```

```
stocks # closing pipeline
```

```
## Warning in format.POSIXlt(as.POSIXlt(x), ...): unknown timezone 'zone/tz/
## 2018c.1.0/zoneinfo/America/New_York'
```

```
## # A tibble: 126,217 x 7
##    Date        Open  High   Low Close  Volume Name
##    <date>     <dbl> <dbl> <dbl> <dbl>   <int> <chr>
##  1 2016-08-12   181   181   180   180 1232856 MMM
##  2 2016-08-15   181   181   180   181 1268247 MMM
##  3 2016-08-16   180   180   179   179 1363554 MMM
##  4 2016-08-17   179   180   178   180 1358528 MMM
##  5 2016-08-18   180   180   179   179 1088677 MMM
##  6 2016-08-19   179   180   178   180 1305289 MMM
##  7 2016-08-22   179   180   178   179 1336377 MMM
##  8 2016-08-23   180   181   179   180 1195825 MMM
##  9 2016-08-24   179   180   179   179 1137075 MMM
## 10 2016-08-25   179   180   179   180  840299 MMM
## # ... with 126,207 more rows
```

The data is now available in as a data frame, named “stocks”. We can add columns to the data set to help us better analyze the stocks. For example, it would be very helpful if we had a column to tell us whether the stock value increased or decreased on the day. We use the mutate function to create a new column named “(G/L)” that subtracts the closing price from the opening price of the stock to see if the stock lost or gained value.

```
stocks <- stocks %>%
  mutate( "G/L" = Close - Open)
stocks
```

```
## # A tibble: 126,217 x 8
##    Date        Open  High   Low Close  Volume Name   `G/L`
##    <date>     <dbl> <dbl> <dbl> <dbl>   <int> <chr>  <dbl>
##  1 2016-08-12   181   181   180   180 1232856 MMM   -1.20
##  2 2016-08-15   181   181   180   181 1268247 MMM   -0.440
##  3 2016-08-16   180   180   179   179 1363554 MMM   -0.870
##  4 2016-08-17   179   180   178   180 1358528 MMM    1.20
##  5 2016-08-18   180   180   179   179 1088677 MMM   -0.520
##  6 2016-08-19   179   180   178   180 1305289 MMM    0.950
##  7 2016-08-22   179   180   178   179 1336377 MMM   -0.110
##  8 2016-08-23   180   181   179   180 1195825 MMM    0.160
##  9 2016-08-24   179   180   179   179 1137075 MMM    0
## 10 2016-08-25   179   180   179   180  840299 MMM    0.490
## # ... with 126,207 more rows
```

We can also use R to calculate the average price on the stock for the year. As you can see I am chosing to create a new data frame to hold our new data, named “annual_report”. I still use the all of the data from “stocks” but the code I write will have no effect on the original “stocks” data frame. This can be important if we want to ensure that we preserve the original data set so our analysis is accurate. In this case, we are not changing the data set only adding information. To do that we would group the data by name, then take the mean of all of the data in the “close” column. Then I use the select funtion to see only the columns I want to see in the new data frame.

```
annual_report <- stocks %>%
  group_by(Name) %>%
  mutate (mean_price = mean(Close)) %>%
  select(Name, mean_price)
annual_report
```

```
## # A tibble: 126,217 x 2
## # Groups:   Name [504]
##    Name  mean_price
##    <chr>      <dbl>
##  1 MMM          187
##  2 MMM          187
##  3 MMM          187
##  4 MMM          187
##  5 MMM          187
##  6 MMM          187
##  7 MMM          187
##  8 MMM          187
##  9 MMM          187
## 10 MMM          187
## # ... with 126,207 more rows
```

As you can see here we have the same data repeated, which is not good for easy analysis of data. This is because R added added mean_price variable for all the stocks for every day of data. We can use the distinct funtion to rid the data frame of duplicates and make it easier to analyze.

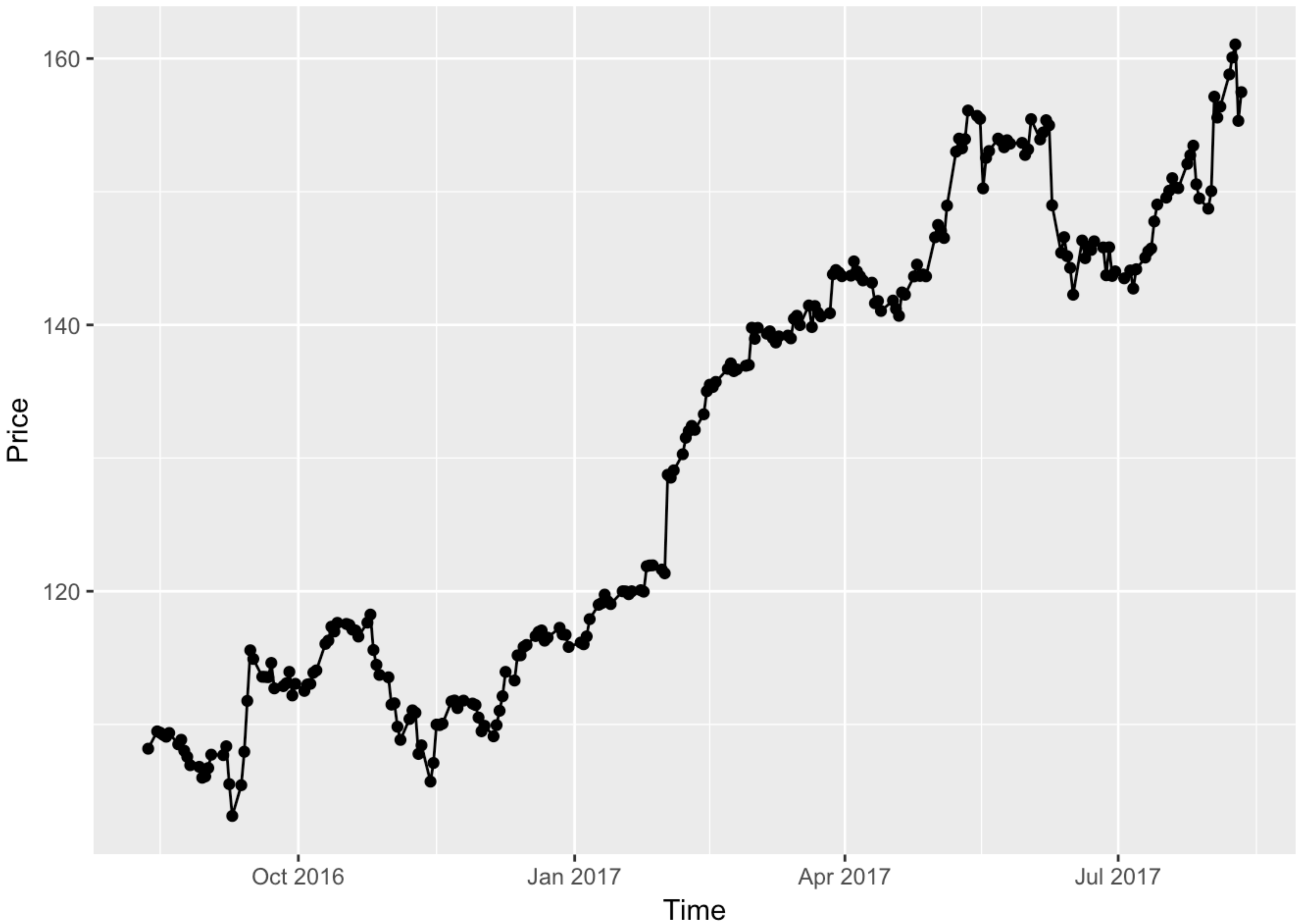
```
annual_report %>%
  distinct()
```

```
## # A tibble: 504 x 2
## # Groups:   Name [504]
##    Name  mean_price
##    <chr>      <dbl>
##  1 MMM        187
##  2 ABT         43.4
##  3 ABBV        64.8
##  4 ACN        120
##  5 ATVI        47.5
##  6 AYI        217
##  7 ADBE       121
##  8 AMD         10.8
##  9 AAP        147
## 10 AES         11.6
## # ... with 494 more rows
```

Say we want to know if we should invest in a certain stock. Before buying shares of the stock we may want to make sure that the stock will gain a profit. If a stock has a continuously negative trend, someone is less likey to invest in it as opposed to a stock that performs well. We can also see the overall trend of a single stock for the whole year. Let’s use the Apple stock (AAPL) as an example. We use the filter function to rid of all excess data, only preserving Apple stock data. Use a pipline to pass the appropriate data set to ggplot. Using ggplot we create a dot plot where we set y equal to closing prices and x equal to the date. Thus we see that the stock has had an overall increase on the year. In the code below we use the “labs” function to label the x and y axis for easy interpretation of the graph.

```
library(ggplot2)

trend <- stocks %>%
  filter(Name == 'AAPL') %>% #Filter all data, keeping only Apple stock
  ggplot(mapping = aes (y = Close, x = Date)) +
  geom_point() +
  geom_line() + #add the line to show trend
  labs(y="Price", x="Time")
trend
```



The graph below shows an overall positive trend on the Apple stock. The stock gained an overall value of +40.0 points on the year. The trend line between each set of 2 dots makes it easy to see how stocks did on a daily basis. For example, approximately around February 2017, there was about a +10 point increase on the day. So investors of the Apple stock gained could have +$10 per share they owned of the stock.

R also allows us to sum the whole variables. What if we wanted to know which stock was traded most in 2017?

```
Sum_Volume <- stocks %>%
  group_by(Name) %>%
  mutate (volume_sum = sum(Volume)) %>%
  mutate(mean_price = mean(Close)) %>%
  select(Name, volume_sum, mean_price) %>%
  distinct()
```

```
## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))
```

```
## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))
```

```
## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
```

```
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))

## Warning in mutate_impl(.data, dots): integer overflow - use
## sum(as.numeric(.))
```

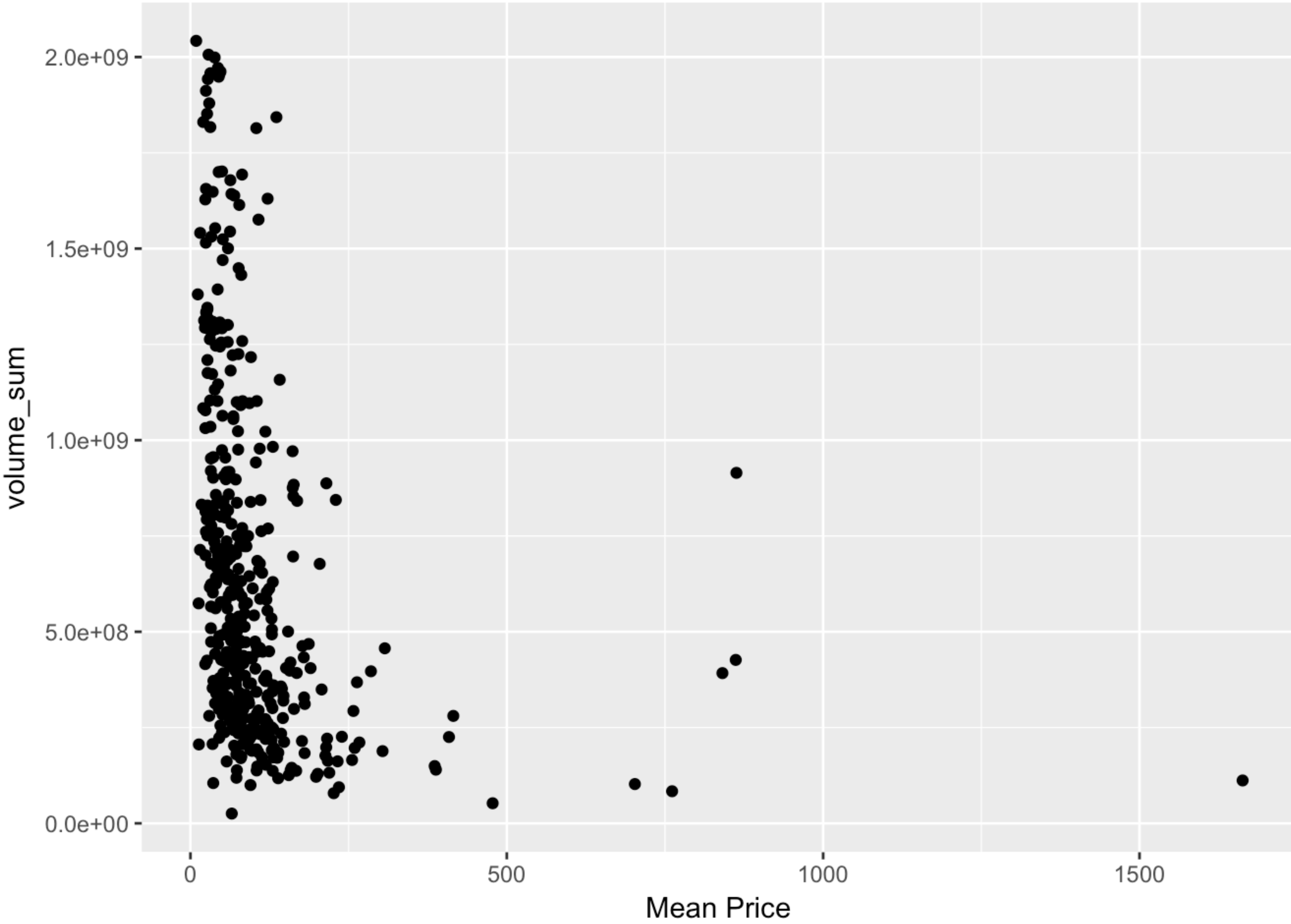
```
Sum_Volume
```

```
## # A tibble: 504 x 3
## # Groups:   Name [504]
##    Name  volume_sum mean_price
##    <chr>      <int>      <dbl>
##  1 MMM    468142055      187
##  2 ABT   1971748687       43.4
##  3 ABBV  1642699287       64.8
##  4 ACN    583838063      120
##  5 ATVI  1962083260       47.5
##  6 AYI    163305193      217
##  7 ADBE   603572083      121
##  8 AMD           NA       10.8
##  9 AAP    320003544      147
## 10 AES   1380596189       11.6
## # ... with 494 more rows
```

R also allows us to analyze large sets of data and compare them against eachother. Lets use a dot plot to see if there is a correlation between stock price and the volume it is being traded in. We are going to use the same mean_price variable we created earlier and plot it against the sum of the volume of stock traded.

```
  v_trend <- Sum_Volume %>%
    group_by(Name) %>%
    ggplot(aes(y=volume_sum, x=mean_price)) +
    geom_point() +
    labs(x ="Mean Price", Y ="Volume" )
  v_trend
```

```
## Warning: Removed 50 rows containing missing values (geom_point).
```



It is clear that higher priced stocks are traded in lower volume than lower priced stocks. This makes logical sense because it would requires more money to buy and sell expensive stocks. Not everyone that trades in the stock market can afford to buy a large volume of stock at such a high price!

We can go even further to analyze the entire stock market and encorporate hypothesis testing. Does the volume of the stock have an impact on the overall performance of the stock, i.e is a better performing stock traded higher in volume than a stock that is not performing well? Note: Volume is the amount of stock traded on a given day.

In order to answer this question it requires us to perform a hypothesis test. Let our null hypthesis be that stock performance on the day does not have an impact on the volume of the stock traded on the day. Using the lm function in R to create a linear regression model to test our null hypothesis. We are going to compare our G/L column to the Volume column, and set the data set equal to “stocks”. The syntax “stocks$Volume” simply means that we are using the “Volume” column of the “stocks” dataset. Here we create a new pipline named reg to save the linear model for a simple linear regression.

```
library(broom) #needed for tidy function

reg <- lm(stocks$Volume ~ stocks$"G/L", data = stocks)
reg %>%
  tidy() #function puts ref into a dataframe
```

```
##            term   estimate std.error  statistic   p.value
## 1  (Intercept) 4154193.05 22572.702 184.036148 0.0000000
## 2 stocks$"G/L"  -12366.71  9091.885  -1.360192 0.1737717
```

From the linear regression model we see that for every $1 increase in performance, the Volume of the stock increases by about 4 million stocks. The lm function also returns the p-value, which in this case is 0.17. This P-value is greater than 0.05 which means we cannot reject the null hypthesis. Had our p-value been smaller than 0.05 we could reject the null hypthesis with sufficient evidence to reject our claim. However, there there is not suffiencent evidence to prove that the performance on the stock has an impact on the volume the stock is traded at.

In the next code block below we use the anova function to analyze the linear regression model that we just built. We simply pass the regression model into the anova function. The output is a Variance Table. This is a statistical test that analyzes variance of the regression. We use this test to see if the regression model we made is significant overall.

```
anova(reg)
```

```
## Analysis of Variance Table
##
## Response: stocks$Volume
##                  Df     Sum Sq    Mean Sq F value Pr(>F)
## stocks$"G/L"      1 1.1862e+14 1.1862e+14  1.8501 0.1738
## Residuals    125835 8.0681e+18 6.4116e+13
```

This anova test outputs a P value, greater than 0.05. Thus we even more confidently reject the null hypothesis.