A Deeper Look Into dplyr

Introduction

During my internship this summer, I had been tasked to combine nine data sets together. These data sets contained different information about the same stocks, and so I was faced with a challenge - how could I *precisely* match information about stock A from data set y with the information about the same stock A to data set x?

What we already know

In class, we learned how to use built-in functions like *cbind()* and *rbind()* to bind vectors or dataframes by column and row respectively. This documentation page provides a good refresher on how to use the aforementioned functions.

In class, we used dplyr specifically for data manipulation purposes - I like to think of it as "cleaning the data" in preparation for data visualization. We primarily used five functions that helped us work with data frames, and with subsets of data frames:

dplyr function	example	what it does
filter()	filter(df, team == "GSW")	filter all data entries where team is GSW
arrange()	arrange(df, team, points)	reorders df so team and points and displayed in specified order
select()	select(df, team, points)	selects and displays only team and points
mutate()	mutate(df, gain = v1 - v2)	adds gain column
summarise()	summarise(df, avg = mean(points))	summarise mean stats for points

It is really interesting to note that all of these functions have a similar structure:

1. they all begin with calling the dataframe
2. subsequent arguments detail what to do with the dataframe

Another function that we had learned in class was the piping operation "%>%" which removes the need for us to specify the dataframe in every line of code.

What's new

Most of the work we have done thus far has been working with one dataset and visualizing that particular dataset. However, as I had experienced in my internship, often times we need to draw information from multiple sources, and manipulate and visualize them all together. So how do we go about combining different datasets in an accurate and precise manner?

The dplyr package actually has several tools that allows us to combine data sets provisionally! So, it isn't just great for the five tools we had used it for in class. For example, if you want to combine secondary data sets to a primary data set by matching similar variables, dplyr allows you to do just that! (*if you are familiar with Excel, this is similar to vlookups / index matches*)

Within these combination tools, we will be looking at ways to mutate data sets.

Before we get started with looking at the tools we will need, let's be sure to load the dplyr package that we are about to explore further, as well as the ggplot2 package that we will utilize to visualize some data!

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 3.2.5
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.2.5
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.2.5
```

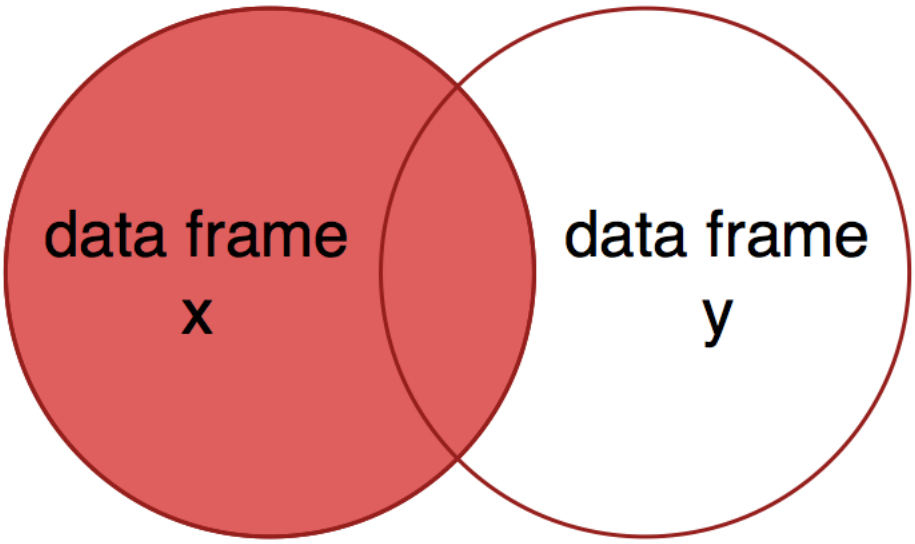
For the purposes of this assignment, I wanted to look at futures trading data, which is a type of financial contract. This was obtained from Quandl, a website that contains financial and economic data delivered in modern formats for users to analyze. If you are curious about how

futures contracts work, you can watch this video on corn futures that illustrates how the buyers and sellers interact.

Combining Data Sets with dplyr

Mutating Joins

The concept of mutating joins is actually pretty simple: for a dataframe y whose entries we would like to include in dataframe x, we want to map information for entries that are present in BOTH x and y. We want to exclude the mapping of information for entries of y that do not exist in dataframe x. Here is a diagram that might help you understand this concept better:



Let's first download two data sets that we will be analyzing today.

```
vx_unadjusted <- read_csv("../data/futures1.csv")
```

```
## Parsed with column specification:
## cols(
##   Date = col_character(),
##   Open = col_double(),
##   High = col_double(),
##   Low = col_double(),
##   Settle = col_double(),
##   Volume = col_integer(),
##   `Prev. Day Open Interest` = col_integer()
## )
```

```
vx_adjusted <- read_csv("../data/futures2.csv")
```

```
## Parsed with column specification:
## cols(
##   Date = col_character(),
##   Open = col_double(),
##   High = col_double(),
##   Low = col_double(),
##   Settle = col_double(),
##   Volume = col_double(),
##   `Prev. Day Open Interest` = col_double()
## )
```

We want to add the "settle" information - which is the trade price - from the adjusted dataset to the unadjusted one, matching by date.

```
futures <-
vx_unadjusted %>%
  left_join(y=vx_adjusted, by="Date")
head(futures)
```

```
## # A tibble: 6 x 13
##       Date Open.x High.x Low.x Settle.x Volume.x
##      <chr>  <dbl>  <dbl> <dbl>    <dbl>    <int>
## 1 12/31/14  17.00  18.50 16.90   18.225    47975
## 2 12/30/14  16.70  17.25 16.65   17.025    29398
## 3 12/29/14  16.55  17.00 16.54   16.725    22322
## 4 12/26/14  16.93  16.94 16.50   16.525    24076
## 5 12/24/14  16.75  16.95 16.65   16.925    16899
## 6 12/23/14  16.95  17.16 16.65   16.725    33905
## # ... with 7 more variables: `Prev. Day Open Interest.x` <int>,
## #   Open.y <dbl>, High.y <dbl>, Low.y <dbl>, Settle.y <dbl>,
## #   Volume.y <dbl>, `Prev. Day Open Interest.y` <dbl>
```

The structure of this line of code is simple: first, we used the pipe operator to pass the function “left_join” into the vx_unadjusted dataset. The first variable in the “left_join” function specifies the y dataset to referenced, and the by variable specifies the column by which the matching occurs.

The mutating join can also be done with y as the primary dataframe, and x as the secondary one from which information is retrieved to be added to y. This can be achieved by:

```
futures2 <-
vx_unadjusted %>%
  right_join(y=vx_unadjusted, by="Date")
```

Mutating by Set

Does “set theory” ring a bell? You might have encountered this term in the first statistics class you’ve taken. In set theory, two concepts that are frequently used are “intersection” and “union”. Dplyr also has functions that express these two concepts!

These two functions follow the same structure as the above ones, and are:

- inner_join : to obtain the values that are intersecting between two dataframes
- full_join : to obtain the values that are in either of two dataframes

Fun fact! Dplyr also has built-in functions that allow you to explictly apply your knowledge of set operations.

1. intersect(y, z)
2. union(y, z)
3. setdiff(y, z) : obtain rows that appear in y but not in z

Let’s experiment with a simple example that is similar to that demonstrated in this webpage.

```
df1 <- data_frame(x = c(1, 2, 4), y = c(2, 6, 7))
df2 <- data_frame(x = c(1, 2, 3), y = 2:4)
```

1. Intersection: of df1 and df2, we would logically only expect one row in the intersection upon examination of both data frames:

```
intersect(df1, df2)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     2
```

2. Union: the union of df1 and df2 would return all rows observed in either data frames:

```
union(df1, df2)
```

```
## # A tibble: 5 x 2
##       x     y
##   <dbl> <dbl>
## 1     3     4
## 2     2     3
## 3     4     7
## 4     2     6
## 5     1     2
```

3. Setdiff: we would expect two rows of data that are unique to df1 to be returned when the setdiff function is called.

```
setdiff(df1, df2)
```

```
## # A tibble: 2 x 2
##       x     y
##   <dbl> <dbl>
## 1     2     6
## 2     4     7
```

To learn a little bit more about set theory, this post on stack exchange does a pretty good job of sets!

Let’s do a little cleaning of the futures data we want to work with, because “Open.x” versus “Open.y” aren’t exactly the most descriptive variable names. We first want to rename the column names, and then select the columns that we are interested in analyzing today.

```
futures <-
futures %>%
  rename(open_raw = Open.x,
         high_raw = High.x,
         low_raw = Low.x,
         settle_raw = Settle.x,
         volume_raw = Volume.x,
         open_adj = Open.y,
         high_adj = High.y,
         low_adj = Low.y,
         settle_adj = Settle.y,
         volume_adj = Volume.y
         )

futures <-
  futures %>%
  select(Date,
         open_raw,
         high_raw,
         low_raw,
         settle_raw,
         volume_raw,
         open_adj,
         high_adj,
         low_adj,
         settle_adj,
         volume_adj)
```

Combining data sets can increase your efficiency in cleaning and manipulating data. Imagine if we hadn't combined the data sets: we would have had to rename each data set separately, created separate objects that contained variables of interest, and, moving forward, repeat the same manipulation operations x amount of times!

Wrangling with Combined Data Sets

Something that might be interesting to look at is the difference between the bid and ask prices of the futures contract. When looking at prices, we are concerned with bid prices - which is how much people want the contracts for, and ask prices - which is how much the contracts were sold for.

The *ask price* is more straightforward - it is simply the "settle price". The *bid price* can be calculated by taking the mean of the "high" and "low" prices.

```
futures <- mutate(futures,
                  bid_raw = (high_raw + low_raw)/2,
                  bid_adj = (high_adj + low_adj)/2)
```

```
## Warning: package 'bindrcpp' was built under R version 3.2.5
```

```
futures_select <-
  futures %>%
  select(Date,
         settle_raw,
         settle_adj,
         bid_raw,
         bid_adj)

summary(futures_select)
```

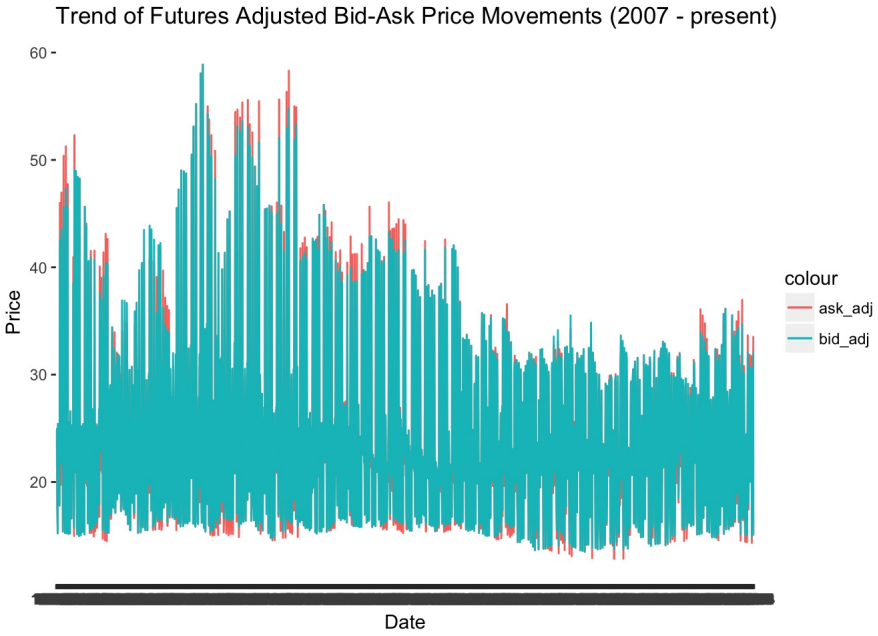
```
##      Date             settle_raw      settle_adj       bid_raw
##  Length:1876        Min.   :12.80   Min.   :13.45   Min.   :12.79
##  Class :character   1st Qu.:17.65   1st Qu.:17.99   1st Qu.:17.70
##  Mode  :character   Median :22.30   Median :22.64   Median :22.34
##                     Mean   :23.83   Mean   :23.97   Mean   :23.84
##                     3rd Qu.:27.31   3rd Qu.:27.52   3rd Qu.:27.30
##                     Max.   :59.77   Max.   :59.77   Max.   :58.88
##     bid_adj
##  Min.   :13.50
##  1st Qu.:17.94
##  Median :22.69
##  Mean   :23.97
##  3rd Qu.:27.43
##  Max.   :58.88
```

We now end up with just the bid and ask (settle) prices that we want to dive deeper into.

Visualizing Combined Data Sets

Let's try to visualize the bid and ask adjusted prices using ggplot2 and observe the trend in price movements! For the purposes of illustration, let's just focus on the adjusted prices.

```
ggplot(data = futures_select, aes(x = Date, group = 1)) +
  geom_line(mapping = aes(y = bid_raw, color = "ask_adj")) +
  geom_line(mapping = aes(y = bid_adj, color = "bid_adj")) +
  ggtitle("Trend of Futures Adjusted Bid-Ask Price Movements (2007 - present)") +
  xlab("Date") +
  ylab("Price")
```



What do you observe? I see certain points in the chart where the ask price exceeded that of the bid price, which means that sellers wanted to sell their futures contracts for a higher price than buyers wanted them. *However*, you will also note that this plot seems very cluttered, because we are including data from every single day since 2007! If we were actual futures analysts, we would probably want to filter our data to reflect that over a certain period of time. Here is a cheatsheet resource that I think might be helpful for data visualization. Thankfully, there are also a multitude of resources out there that have worked with futures data in R before. This blog demonstrates how to select a precise time period.

Putting It All Together

So, what did we learn today? We started off with revisiting the tools that we had learned in dplyr already. Then, we explored a few tools that we didn't know was built-in in this package.

1. The mutating join functions allow us to match one data frame to another
2. The mutating by set functions allow us to obtain the union, intersection, or unique values of a couple data sets

Then, we put our ggplot2 muscle to use and did some initial visualization of data.

Pretty simple, isn't it? I think it is really cool that the little tools we learned in class, coupled with a few others that we can find with a little exploration, can be applied in such a realistic context! While large datasets such as the one that we worked with might be initially intimidating, fret not! Power to us, because we are already well-equipped. :)