Introduction

As per lexico.com the word manipulate means "Handle or control (a tool, mechanism, etc.), typically in a skilful manner". Data manipulation is an exercise of skillfully clearing issues from the data and resulting in clean and tidy data.

What is the need for data manipulation?

In real-life, data is not always presented analysis-ready. Most of the time it has gaps, missing values, incorrect date formats, missing variable names, no variable names, or variables scattered into multiple columns etc. When this kind of data is presented for analysis, it needs intervention to make it tidy. Concept of tidy data was introduced by Hadley Wickham in 2014. The three rules of tidy data are:

1. Each variable must have its own column
2. Each observation must have its own row
3. Each type of observational unit forms a table

Furthermore, data sufficing the three principles above yields much better results. In sections below, we will see the different type of functions and their use over data frame under the different type of manipulations options offered by R. Here is the brief overview of the article:

Part 2 of this series will have the following:

1. Aggregate data by using
    1. aggregate function
    2. summarize function
    3. "group by" function of dplyr package
    4. sum and count functions
    5. "group by" function of dplyr package and filtering summarized data
2. Sort
    1. by using the arrange function of dplyr package

In the first part of this two-part series, we will see the different ways of subsetting and transformation of data from a data frame.

Prerequisites:

1. R
2. R Studio (for ease)

    Assumption: Working directory is set and datasets are stored in the working directory. setwd() command is used to set the working directory. Supply the path of directory enclosed in double quotes to set it as a working directory.

Data

We will use the data that we have prepared in our previous post How to prepare data for analysis in R in 5 steps. As well as, <u>all examples below will only use data frames</u>. Original data can be downloaded from s and p 500 companies financials. This data is available under the PDDL license.

Reading data

Similar to our other two articles How to subset a data frame column data in R and How to subset rows from a data frame in R, we will use `read.csv` function to read CSV file into R.

```
financials <- read.csv("constituents-financials_csv.csv")
```

The command above will import the content of the constituents-financials_csv.csv file into R and create a data frame that contains all the data from the constituents-financials_csv.csv file.

Command read.csv above can take multiple other arguments other than just the name of the file. More details about read.csv are available here.

Subset

"Subset" is a statistical term which means a group of elements that is part of a larger group. Similarly, subsetting is an activity of extracting a piece of data from a larger data set. In the section above, we have created a data frame financials by uploading a CSV file. In this section, we will see various ways of subsetting a data frame.

Subsetting data frame by row numbers or index

Before we start subsetting data frame let's look at the structure of `Financials` data frame.

```
str(financials)
```

```
## 'data.frame':    505 obs. of  14 variables:
##  $ Symbol        : chr  "MMM" "AOS" "ABT" "ABBV" ...
##  $ Name          : chr  "3M Company" "A.O. Smith Corp" "Abbott Laboratories"
"AbbVie Inc." ...
##  $ Sector        : chr  "Industrials" "Industrials" "Health Care" "Health
Care" ...
##  $ Price         : num  222.9 60.2 56.3 108.5 150.5 ...
##  $ Price.Earnings: num  24.3 27.8 22.5 19.4 25.5 ...
##  $ Dividend.Yield: num  2.33 1.15 1.91 2.5 1.71 ...
##  $ Earnings.Share: num  7.92 1.7 0.26 3.29 5.44 1.28 7.43 3.39 6.19 0.03 ...
##  $ X52.Week.Low  : num  259.8 68.4 64.6 125.9 162.6 ...
##  $ X52.Week.High : num  175.5 48.9 42.3 60 114.8 ...
##  $ Market.Cap    : num  1.39e+11 1.08e+10 1.02e+11 1.81e+11 9.88e+10 ...
##  $ EBITDA        : num  9.05e+09 6.01e+08 5.74e+09 1.03e+10 5.64e+09 ...
##  $ Price.Sales   : num  4.39 3.58 3.74 6.29 2.6 ...
##  $ Price.Book    : num  11.34 6.35 3.19 26.14 10.62 ...
##  $ SEC.Filings   : chr  "http://www.sec.gov/cgi-bin/browse-edgar?action=
getcompany&CIK=MMM"  "http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=AOS"
"http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=ABT"  "http://www.sec.gov/cgi-bin/
browse-edgar?action=getcompany&CIK=ABBV"  ...
```

The result above shows us a lot of information. For example, there are a total of 505 observations (rows) and 14 variables (columns). Also, there are four character type variables and the rest of them are all numeric.

Let’s quickly look at the first two records using `head` command.

```
head(financials,2)
```

```
##   Symbol            Name      Sector  Price Price.Earnings Dividend.Yield
## 1    MMM      3M Company Industrials 222.89          24.31       2.332862
## 2    AOS A.O. Smith Corp Industrials  60.24          27.76       1.147959
##   Earnings.Share X52.Week.Low X52.Week.High   Market.Cap    EBITDA
Price.Sales
## 1           7.92       259.77       175.490 138721055226 9.048e+09
4.390271
## 2           1.70        68.39        48.925  10783419933 6.010e+08
3.575483
##   Price.Book
SEC.Filings
## 1      11.34 http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=MMM
## 2       6.35 http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=AOS
```

The integers next to the ## are the row numbers or the index. Using this you can subset any row or range of row from the data frame. Here is an example where we will subset 7th row.

```
financials[7,]
```

```
##   Symbol               Name      Sector  Price Price.Earnings Dividend.Yield
## 7    AYI Acuity Brands Inc Industrials 145.41          18.22      0.3511853
##   Earnings.Share X52.Week.Low X52.Week.High Market.Cap    EBITDA Price.Sales
## 7           7.43       225.36           142 6242377704 587800000    1.795347
##   Price.Book
SEC.Filings
## 7       3.55 http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=AYI
```

Notice the use of square brackets and use of a comma to define that we want to subset rows. Placement of the argument with respect the comma is important as it impacts the result. Argument at the right side of the comma subset column whereas left side subset rows.

Let’s use row numbers or indexes to subset range of rows from 2nd to 4th row.

```
financials[2:4,]
```

```
##   Symbol                Name      Sector  Price Price.Earnings Dividend.Yield
## 2    AOS     A.O. Smith Corp Industrials  60.24          27.76       1.147959
## 3    ABT Abbott Laboratories Health Care  56.27          22.51       1.908982
## 4   ABBV         AbbVie Inc. Health Care 108.48          19.41       2.499560
##   Earnings.Share X52.Week.Low X52.Week.High   Market.Cap    EBITDA
Price.Sales
## 2           1.70        68.39        48.925  10783419933 6.010e+08
3.575483
## 3           0.26        64.60        42.280 102121042306 5.744e+09
3.740480
## 4           3.29       125.86        60.050 181386347059 1.031e+10
6.291571
##   Price.Book
SEC.Filings
## 2       6.35  http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=AOS
## 3       3.19  http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=ABT
## 4      26.14 http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=ABBV
```

The same operation can be done using concatenation function as well. Here is an example:

```
financials[c(2:4),]
```

```
##   Symbol                Name      Sector  Price Price.Earnings Dividend.Yield
## 2    AOS      A.O. Smith Corp Industrials  60.24          27.76       1.147959
## 3    ABT Abbott Laboratories Health Care  56.27          22.51       1.908982
## 4   ABBV          AbbVie Inc. Health Care 108.48          19.41       2.499560
##   Earnings.Share X52.Week.Low X52.Week.High   Market.Cap    EBITDA
Price.Sales
## 2           1.70        68.39        48.925  10783419933 6.010e+08
3.575483
## 3           0.26        64.60        42.280 102121042306 5.744e+09
3.740480
## 4           3.29       125.86        60.050 181386347059 1.031e+10
6.291571
##   Price.Book
SEC.Filings
## 2       6.35  http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=AOS
## 3       3.19  http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=ABT
## 4      26.14 http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=ABBV
```

Subsetting data frame by row numbers or index using dplyr package

dplyr is a grammar of data manipulation in R. I find data manipulation easier using dplyr, I hope you would too if you are coming with a relational database background.

Let's look at the row subsetting using dplyr package based on row number or index. In the code below, the filter function is used to subset observations. filter function returns cases based on the value of the conditional expression supplied to it.

A detailed article on the dplyr package would come in the future. Meanwhile, the function row returns the row number of the supplied data frame. The output of the row function is checked for equality with the row number of choice and returned index of the row is evaluated by filter function to return the result. The filter function is receiving input as data frame via pipe operation from financials data frame.

```
library(dplyr)
financials %>% filter(row(financials) == 7)
```

```
##   Symbol               Name      Sector  Price Price.Earnings Dividend.Yield
## 1    AYI Acuity Brands Inc Industrials 145.41          18.22      0.3511853
##   Earnings.Share X52.Week.Low X52.Week.High Market.Cap    EBITDA Price.Sales
## 1           7.43       225.36           142 6242377704 587800000    1.795347
##   Price.Book
SEC.Filings
## 1       3.55 http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=AYI
```

Subsetting data frame using operator

One of the good things about R is that you can do the same thing in different ways. In this section, we will see how we can subset the data frame column in three different ways. First, by using $, second by using [] and, third and the last one by using the [[]] operator.

Let's print the name of all the variables from the financials data frame using the names command.

```
names(financials)
```

```
##  [1] "Symbol"         "Name"           "Sector"         "Price"
##  [5] "Price.Earnings" "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Price.Sales"
## [13] "Price.Book"     "SEC.Filings"
```

Now when we know the name of the variables, let's select the data from Sector variable using $ operator. To save space, we will use the head command with argument 5 to only show us 5 values from this variable.

```
head(financials$Sector, 5)
```

```
## [1] "Industrials"            "Industrials"            "Health Care"
## [4] "Health Care"            "Information Technology"
```

Let's perform the same exercise using [] operator. Note that while using [] or [[]] operators, name of the variable must be within double-quotes.

```
head(financials["Sector"], 5)
```

```
##                    Sector
## 1             Industrials
## 2             Industrials
## 3             Health Care
## 4             Health Care
## 5 Information Technology
```

Now with `[[]]` operator

```
head(financials[["Sector"]], 5)
```

```
## [1] "Industrials"            "Industrials"            "Health Care"
## [4] "Health Care"            "Information Technology"
```

Can you spot the difference between the two results above?

Use of [[]] operator return results as a vector that has the same class as of the selected variable. Knowing this is important while working on data transformations, you may want to have results as a vector.

Similar to row subsetting, you can select multiple columns by using the concatenating function. Let's have a look by selecting Name and Sector variables. Note that to subset column we should keep the argument to the right while using the [] operator. Reminder, the head function is here just to select top 5 records to save space.

```
head(financials[, c("Name","Sector")], 5)
```

```
##                   Name                 Sector
## 1           3M Company            Industrials
## 2     A.O. Smith Corp            Industrials
## 3 Abbott Laboratories            Health Care
## 4          AbbVie Inc.            Health Care
## 5       Accenture plc Information Technology
```

Subsetting data frame based on a condition

Subsetting of the rows or columns we have seen in the section above is straight forward because we know the index or the observations to subset. However, in real-world, subsetting is derived by sometimes complex conditions and calculations.

Let's look at how can we subset rows from a data frame based on a condition. The requirement is to retrieve all the rows from the financials data frame where the variable (column) Symbol are "ACN" or "APTV" and the variable Sector is either "Industrials" or "Consumer Discretionary".

```
financials[(financials$Symbol == "ACN" | financials$Symbol == "APTV") &
(financials$Sector == "Consumer Discretionary" | financials$Sector ==
"Industrials"),]
```

```
##    Symbol      Name                 Sector Price Price.Earnings
Dividend.Yield
## 54   APTV Aptiv Plc Consumer Discretionary 89.27          69.74
0.9392678
##    Earnings.Share X52.Week.Low X52.Week.High  Market.Cap   EBITDA Price.Sales
## 54           5.05        96.91         82.97 24906530300 2.37e+09     1.50258
##    Price.Book
## 54       7.56
##                                                             SEC.Filings
## 54 http://www.sec.gov/cgi-bin/browse-edgar?action=getcompany&CIK=APTV
```

We can see that only one row falls into the subsetting criterion. Notice the use of "&", "|" and "==" operators to define "AND", "OR" and equality conditions respectively. Additionally, note that full subsetting criterion is written on the left side of the comma as we are filtering only rows.

Subsetting data frame based on a condition using dplyr package

The dplyr package offers great functionality when it comes to filtering observations (rows). Let's look at the same scenario defined above using dplyr function but we will only select Name and Sector variables.

```
financials %>% filter((Symbol == "ACN" | Symbol == "APTV") & (Sector ==
"Consumer Discretionary" | Sector == "Industrials")) %>% select(Name, Sector)
```

```
##        Name                 Sector
## 1 Aptiv Plc Consumer Discretionary
```

Subsetting data frame using subset function

`The` subset is a generic function which accepts data frames, matrices and vectors and returns subsets of supplied object type based on a condition. Here in this article, we are only looking at data frames and various ways of data manipulations that can be performed over data frames.

The good thing about the subset is that you can define:

1. a condition to subset observations as well as
2. which variables you would like to subset using select clause

Let's look at an example of subset function and select Name and Sector variables where the variable (column) Symbol are "ACN" or "APTV" and the variable Sector is either "Industrials" or "Consumer Discretionary".

```
subset(financials, (financials$Symbol == "ACN" | financials$Symbol == "APTV") &
(financials$Sector == "Consumer Discretionary" | financials$Sector ==
"Industrials"), select= c(Name, Sector))
```

```
##         Name                 Sector
## 54 Aptiv Plc Consumer Discretionary
```

Transform

Why would you transform data?

Perhaps the data you have is incomplete for the analysis you would like to perform. There may be different reasons for data transformation. Here in this section, we will look at the following four elements.

Please note that data transformation is a huge topic and it cannot be limited to the following four aspects. Need for data transformation is driven by individual requirements and the four topics below are generic.

- How to add a new variable to the data frame in R
- How to remove a variable from the data frame in R
- How to rename variables of a data frame
- How to merge and unite data frames

How to add a new variable to the data frame in R

Data frame is a matrix-like structure where individual variables (columns) may be of different types. The data frame financials that we are using in this article has four characters and the rest of the numeric variables. What if we want to add another variable which should hold a logical value for all observations where the price of the share is higher than 1000.

Let’s look at this scenario and add a new variable more.than.grand to our data frame. This variable currently doesn’t exist into the data frame and a simple assignment using <- would add it to the data frame.

```
financials$more.than.grand <- financials$Price >= 1000
financials[financials$more.than.grand,"more.than.grand"]
```

```
## [1] TRUE TRUE TRUE TRUE
```

The part of the first command above financials$Price >= 1000 is to perform a logical check on all the observations of the data frame and return TRUE or FALSE depending on the value of Price variable for a particular observation.

There are other ways to add a new column to the data frame. Let’s look at them as well.

```
financials[["more.than.grand"]] <- financials$Price >= 1000
financials[financials$more.than.grand,"more.than.grand"]
```

```
## [1] TRUE TRUE TRUE TRUE
```

Notation [[]] is primarily used in List objects to subset the elements. Data frame too is a list but with few restrictions and hence the use of the notation [[]] works. If you disagree with this then please do comment in the comment section.

Another way to add a new column the data frame is as follows.

```
financials[,"more.than.grand"] <- financials$Price >= 1000
financials[financials$more.than.grand,"more.than.grand"]
```

```
## [1] TRUE TRUE TRUE TRUE
```

Note the usage of the comma. As we are adding a new variable, we have written the expression on the right side of the comma. As the variable doesn’t already exist, R would add to the data frame.

If you know any other way to add a new column other then the three we discussed here then please do mention it in the comment box below.

How to remove a variable from a data frame in R

All three ways we have seen above would work to remove the variable. The only difference would be that the variable you would like to remove should be assigned with a NULL value. Let’s have a look.

```
financials[,"more.than.grand"] <- NULL
names(financials)
```

```
##  [1] "Symbol"         "Name"           "Sector"         "Price"
##  [5] "Price.Earnings" "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Price.Sales"
## [13] "Price.Book"     "SEC.Filings"
```

You can see in the result above that the column more.than.grand is no longer part of the financials data frame.

Renaming a variable of a data frame in R

Why would you rename a column?

Because you want to. Jokes apart, in real-life the data that is presented for analysis would sometime have long variable names and this may create a problem with the indentation, length of the line and may eventually induce code readability issues. To avoid all this, you may want to rename the columns which are a little more manageable.

A variable of a data frame can be renamed like this. Let's rename the variables `Price` to `Share.Price` in the financial data frame.

Using Base R

```
names(financials)[names(financials) == "Price"] <- "Share.Price"
names(financials)
```

```
##  [1] "Symbol"         "Name"           "Sector"         "Share.Price"
##  [5] "Price.Earnings" "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Price.Sales"
## [13] "Price.Book"     "SEC.Filings"
```

Let's dissect the command above. The part of the command names(financials) would return a character vector of all the variable names from the data frame financials.

```
class(names(financials))
```

```
## [1] "character"
```

The second part of the command [names(financials) == "Price"] again use the `names` function to return the character vector but the output is conditioned to filter the vector element where the value of the element is equal to Price. Once the result is pinned to that variable of choice, a new value is assigned using <- operator.

Variable renaming can also be done using dplyr package and here is an example:

Using dplyr Package

Before you use any function from dplyr, ensure the library is included. Let's look at the variable renaming using the rename function of dplyr package:

```
library(dplyr)
financials <- financials %>% rename(Price = Share.Price)
names(financials)
```

```
##  [1] "Symbol"         "Name"           "Sector"         "Price"
##  [5] "Price.Earnings" "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Price.Sales"
## [13] "Price.Book"     "SEC.Filings"
```

dplyr package's select function can also be used to rename the column, let's look the command to do that:

```
library(dplyr)
financials <- financials %>% select(Share.Price = Price)
names(financials)
```

```
## [1] "Share.Price"
```

The command above will only return one variable if you use the select function. To keep all other variables, you must supply the other variables within the select function.

Renaming multiple variables of a data frame in R

What if we have a large number of variables starting with a common prefix and we would like to rename all of them in one go, obviously the function rename or select wouldn't help. There are other functions supplied by dplyr which can help in this scenario. Let's look at them one by one:

Renaming multiple variables using dplyr

Before we rename the columns, let's remove the data frame from R and recreate it.

```
rm(financials)
financials <- read.csv("constituents-financials_csv.csv")
names(financials)
```

```
##  [1] "Symbol"         "Name"           "Sector"         "Price"
##  [5] "Price.Earnings" "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Price.Sales"
## [13] "Price.Book"     "SEC.Filings"
```

Function rename_at

Let's use rename_at function by first identifying all the columns that have text "Price" in it and then replace "Price" with "Cost". Library stringr is included to make use of str_replace function.

```
library(dplyr)
library(stringr)
financials <- financials %>% rename_at(vars(contains("Price")),
funs(str_replace(., "Price", "Cost")))
names(financials)
```

```
##  [1] "Symbol"         "Name"           "Sector"         "Cost"
##  [5] "Cost.Earnings"  "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Cost.Sales"
## [13] "Cost.Book"      "SEC.Filings"
```

Function rename_if

Function rename_if allows us to check the type of the variable and then we can apply the function str_replace to change the name of the variables. In the example below, we will first check if the variable is a character type, and if they are then they would be in scope for renaming.

```
library(dplyr)
library(stringr)
financials <- financials %>% rename_if(is.character, funs(str_replace(.,"Name",
"Share.Name")))
names(financials)
```

```
##  [1] "Symbol"         "Share.Name"     "Sector"         "Cost"
##  [5] "Cost.Earnings"  "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Cost.Sales"
## [13] "Cost.Book"      "SEC.Filings"
```

Function rename_all

Function rename_all will rename the column without any checks whether on the type of the column or any other condition. Let’s look at the use of rename_all function:

```
library(dplyr)
library(stringr)
financials <- financials %>% rename_all(funs(str_replace(.,"Cost", "Price")))
names(financials)
```

```
##  [1] "Symbol"         "Share.Name"     "Sector"         "Price"
##  [5] "Price.Earnings" "Dividend.Yield" "Earnings.Share" "X52.Week.Low"
##  [9] "X52.Week.High"  "Market.Cap"     "EBITDA"         "Price.Sales"
## [13] "Price.Book"     "SEC.Filings"
```

Merging and uniting data frames in R

In the real world, we have more than one file or data frame to work with. Sometimes variable names are supplied separately from the observations or the other times, you have observations spread into multiple files. Whatever is the scenario, we must know how to merge two or more data frame together to achieve the tidy dataset. In this section, we will see the different ways of merging and uniting data frames.

Merging or joining data frames in R

As the name suggests merging can be achieved using the merge function. Here is an example
of merge function where we will see the various option of merging but before we start let’s make two data frames using dplyr.

```
library(dplyr)
financials.1 <- financials %>% select(Symbol,Share.Name, Sector)
financials.2 <- financials %>% select(Symbol,Price, EBITDA) %>% filter(Symbol ==
"A" | Symbol == "AAL")
```

Both the new data frames have a Symbol as a common variable. Let’s make one more data frame it as in real life the two data frame may not have the common variable names.

```
library(dplyr)
library(stringr)
financials.3 <- financials.2 %>% rename_at(vars(contains("Symbol")),
funs(str_replace(., "Symbol", "Symbol.New")))
names(financials.2)
```

```
## [1] "Symbol" "Price"  "EBITDA"
```

Merging two data frame with a common variable name

```
library(dplyr)
financials.new <- merge(financials.1, financials.2, by = "Symbol")
head(financials.new,5)
```

```
##   Symbol                 Share.Name      Sector Price    EBITDA
## 1      A Agilent Technologies Inc Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group Industrials 48.60 5.761e+09
```

Merging two data frame where the variable name is not common

```
library(dplyr)
```

```
financials.new <- merge(financials.1, financials.3, by.x = "Symbol", by.y =
"Symbol.New")
head(financials.new,5)
```

```
##   Symbol               Share.Name      Sector Price    EBITDA
## 1      A Agilent Technologies Inc Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group Industrials 48.60 5.761e+09
```

Merging two data frame (left outer join)

There can be scenarios where one of the data frame (primary) has more data than the other one (secondary) and requirement could be to subset all data from first data frame along with all matching data from the second data frame. The result should not restrict any records from the first data frame just because there are no matching records in the second data frame. This is achieved by using all.x. Here x denotes the first data frame.

```
financials.new <- merge(financials.1, financials.3, by.x = "Symbol", by.y =
"Symbol.New", all.x = TRUE)
head(financials.new,5)
```

```
##   Symbol               Share.Name                 Sector Price    EBITDA
## 1      A Agilent Technologies Inc            Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group            Industrials 48.60 5.761e+09
## 3    AAP       Advance Auto Parts Consumer Discretionary    NA        NA
## 4   AAPL               Apple Inc. Information Technology    NA        NA
## 5   ABBV              AbbVie Inc.            Health Care    NA        NA
```

Using dplyr

Please note the usage of arrange function is just to order the data, avoid confusion and display the same results as above. Function arrange is not a requirement for joining two data frames together.

```
library(dplyr)
financials.new <- left_join(financials.1, financials.3, by = c("Symbol" =
"Symbol.New")) %>% arrange(Symbol)
head(financials.new,5)
```

```
##   Symbol               Share.Name                 Sector Price    EBITDA
## 1      A Agilent Technologies Inc            Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group            Industrials 48.60 5.761e+09
## 3    AAP       Advance Auto Parts Consumer Discretionary    NA        NA
## 4   AAPL               Apple Inc. Information Technology    NA        NA
## 5   ABBV              AbbVie Inc.            Health Care    NA        NA
```

Merging two data frame (right outer join)

Right outer join is just opposite of the left outer join where all the matching observations from secondary data frame are returned irrespective to their match in the primary data frame. The following example explains it.

```
library(dplyr)
financials.new <- merge(financials.1, financials.3, by.x = "Symbol", by.y =
"Symbol.New", all.y = TRUE)
head(financials.new,5)
```

```
##   Symbol               Share.Name      Sector Price    EBITDA
## 1      A Agilent Technologies Inc Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group Industrials 48.60 5.761e+09
```

Using dplyr

```
library(dplyr)
financials.new <- right_join(financials.1, financials.3, by = c("Symbol" =
"Symbol.New")) %>% arrange(Symbol)
head(financials.new,5)
```

```
##   Symbol              Share.Name      Sector Price    EBITDA
## 1      A Agilent Technologies Inc Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group Industrials 48.60 5.761e+09
```

Merging two data frame (cross join or full outer join)

Cross join or full outer join is like cartesian join or cartesian product which results in all the observations from both the data frames irrespective to the match. The following example explains it.

```
library(dplyr)
financials.new <- merge(financials.1, financials.3, by.x = "Symbol", by.y =
"Symbol.New", all.x = TRUE, all.y = TRUE)
head(financials.new,5)
```

```
##   Symbol              Share.Name                 Sector Price    EBITDA
## 1      A Agilent Technologies Inc            Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group            Industrials 48.60 5.761e+09
## 3    AAP       Advance Auto Parts Consumer Discretionary    NA        NA
## 4   AAPL               Apple Inc. Information Technology    NA        NA
## 5   ABBV              AbbVie Inc.            Health Care    NA        NA
```

Using dplyr

```
library(dplyr)
financials.new <- full_join(financials.1, financials.3, by = c("Symbol" =
"Symbol.New")) %>% arrange(Symbol)
head(financials.new,5)
```

```
##   Symbol              Share.Name                 Sector Price    EBITDA
## 1      A Agilent Technologies Inc            Health Care 65.05 1.094e+09
## 2    AAL  American Airlines Group            Industrials 48.60 5.761e+09
## 3    AAP       Advance Auto Parts Consumer Discretionary    NA        NA
## 4   AAPL               Apple Inc. Information Technology    NA        NA
## 5   ABBV              AbbVie Inc.            Health Care    NA        NA
```

Semi Join and Anti Join using dplyr

Semi-join and anti-join both are a little different from the three different joins we have gone through in sections above. Both semi and anti-join return the observations from the first data frame only. In the case of semi-join, only the matching observations from first data frame are returned. On the other hand, anti-join returns all unmatched observations from the first data frame are returned into the results. Here is the example of the semi_join and anti_join.

semi_join

```
library(dplyr)
financials.new <- semi_join(financials.1, financials.3, by = c("Symbol" =
"Symbol.New")) %>% arrange(Symbol)
head(financials.new,5)
```

```
##   Symbol              Share.Name      Sector
## 1      A Agilent Technologies Inc Health Care
## 2    AAL  American Airlines Group Industrials
```

anti_join

```
library(dplyr)
financials.new <- anti_join(financials.1, financials.3, by = c("Symbol" =
"Symbol.New")) %>% arrange(Symbol)
head(financials.new,5)
```

```
##   Symbol             Share.Name                 Sector
## 1    AAP     Advance Auto Parts Consumer Discretionary
## 2   AAPL             Apple Inc. Information Technology
## 3   ABBV            AbbVie Inc.            Health Care
## 4    ABC AmerisourceBergen Corp            Health Care
## 5    ABT    Abbott Laboratories            Health Care
```

Uniting data frames in R

We have seen in the sections above how two data frames can be merged and their variables are clubbed together into one resulting data frame. However, there can be scenarios when we would like to club/unit /union all the observations from the two data frames into one. The simplest way is to use rbind function. Here is an example. Note that rbind list expects the two data frames to be same.

```
financials.1 <- financials %>% filter(row(financials) <= 250)
financials.2 <- financials %>% filter(row(financials) > 250)
financials.new <- rbind(financials.1, financials.2)
dim(financials.new)
```

```
## [1] 505  14
```

This concludes the first part of the article. In the second part, we will see the aggregation and sort operations.…