Welcome to the second part of this two-part series on data manipulation in R. This article aims to present the reader with different ways of data aggregation and sorting. Here is the composition of this article.

1. Aggregation in R
   1. Why aggregate data?
   2. Data used in this article
   3. Aggregation using base R
      1. Aggregation using the aggregate function
      2. Assigning a name to an aggregated column in aggregate function in R
      3. Aggregating multiple columns using an aggregate function in R
      4. Aggregating multiple columns data by multiple functions using the aggregate function in R
      5. Aggregation using by() function
      6. Aggregation using sweep() function
   4. Aggregation using dplyr
      1. Aggregation using group\_by and summarize functions
      2. Aggregation by group\_by and summarize functions with multiple variables and functions
      3. Aggregation using group\_by and summarize functions with a range of variables
      4. Aggregation using group\_by and summarize\_if function
      5. Aggregation using group\_by and summarize\_at function
      6. Aggregation using group\_by and summarize\_all function
   5. Aggregation using data.table
      1. Converting data frame to data table
      2. Aggregating single variable using data table
      3. Assigning a custom name to aggregated variable in data table
      4. Aggregating multiple variables using multiple aggregation functions in data table
      5. Aggregating variables as well as filtering observations in data table
      6. Counting the number of observations within a group in data table
   6. Aggregation using sqldf
2. Sorting in R
   1. Why sort data?
   2. Sorting using base R
      1. Sorting data using order function
      2. Sorting data in descending order using order function
      3. Sorting using dplyr
         1. Sorting data using arrange function
         2. Sorting data in descending order using arrange function
      4. Sorting using data.table
         1. Sorting data using order function
         2. Sorting data in descending order using order function
         3. Sorting data using setorder function
         4. Sorting data in descending order using setorder function
         5. Sorting using sqldf

Aggregation in R

To begin with, let’s look at the term aggregation. As per the Cambridge dictionary, “the process of combining things or amounts into a single group or total” is aggregation.

# Why aggregate data?

Usually, the data has two types, qualitative and quantitative. These two are statistical terms. Qualitative data defines the characteristics of the data. Labels, properties, attributes, and categories are all examples of qualitative data. As the name suggests, data that express the quality is qualitative data. On the other hand, quantitative data represent numbers. In other disciplines like data warehousing or business intelligence, qualitative data is equivalent to dimensions and quantitative data to measures.

Data analysis is a complex process and involves several steps. Some of these steps may include data to be examined by its quality. Usually, the qualitative data’s granularity is higher. Granularity is the level of detail.

For example, a dataset that contains all countries of the world may have multiple variables describing qualitative data. The name of the country would be at the most granular level, as all countries’ names would be unique and, no two countries will have the same name. Whereas, granularity level would rise as we look at the countries by continents and then by the hemisphere.

Similarly, in an analysis, data is examined on various levels by its different qualities. At this point, aggregation comes into the picture. It is required if you want to explore quantitative data elements by its quality that sits higher than the most granular level.

In this article, we will practice different aggregation functions and options offered by base R and other packages like dplyr and data.table. Note that this article does not aim to list all the functions that (if used in a way) can aggregate data, whereas the aim here is to explain the concept of data aggregation. R is a rich language that offers different ways of doing the same thing.

# Data used in this article

However, to save space and make it cleaner, we will remove a few columns from the data frame. It’s an opportunity for us to revise how to remove a column from the data frame. Let’s first create the data frame and understand the data.

financials <- read.csv constituents-financials\_csv.csv" str(financials

## 'data.frame': ## $ Symbol

## $ Name

s" "AbbVie Inc." .. ## $ Sector

Care" ..

505 obs. of 14 variables

: chr

: chr

"MMM" "AOS" "ABT" "ABBV" ..

"3M Company" "A.O. Smith Corp" "Abbott Laboratorie

: chr

"Industrials" "Industrials" "Health Care" "Health

## ## ##

##

$ Price

: num

$ Price.Earnings: num

$ Dividend.Yield: num

$ Earnings.Share: num

222.9 60.2 56.3 108.5 150.5 ..

24.3 27.8 22.5 19.4 25.5 ..

2.33 1.15 1.91 2.5 1.71 ..

7.92 1.7 0.26 3.29 5.44 1.28 7.43 3.39 6.19 0.03

## ## ## ## ## ##

##

$ X52.Week.Low : num

$ X52.Week.High : num

$ Market.Cap

$ EBITDA

$ Price.Sales

$ Price.Book

$ SEC.Filings

: num

: num

: num

: num

: chr

259.8 68.4 64.6 125.9 162.6 ..

175.5 48.9 42.3 60 114.8 ..

1.39e+11 1.08e+10 1.02e+11 1.81e+11 9.88e+10 ..

9.05e+09 6.01e+08 5.74e+09 1.03e+10 5.64e+09 ..

4.39 3.58 3.74 6.29 2.6 ..

11.34 6.35 3.19 26.14 10.62 ..

"<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.g](http://www.sec.g/) ov/cgi-bin/browse-edgar?action=getcompany&CIK=ABT <http://www.sec.gov/cgi-bin/browse-edgar?acti>

getcompany&CIK=ABBV ..

Now we have our data frame ready, let’s reshape our data frame by selecting a few columns to keep the results clean and tidy.

financials <- financials %>% select(Symbol, Sector, Price, X52.Week.Low, X52

Week.High)

head(financials

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ## |  | Symbol | Sector | Price | X52.Week.Low | X52.Week.High |
| ## | 1 | MMM | Industrials | 222.89 | 259.770 | 175.490 |
| ## | 2 | AOS | Industrials | 60.24 | 68.390 | 48.925 |
| ## | 3 | ABT | Health Care | 56.27 | 64.600 | 42.280 |
| ## | 4 | ABBV | Health Care | 108.48 | 125.860 | 60.050 |
| ## | 5 | ACN | Information Technology | 150.51 | 162.600 | 114.820 |
| ## | 6 | ATVI | Information Technology | 65.83 | 74.945 | 38.930 |
| ## | 7 | AYI | Industrials | 145.41 | 225.360 | 142.000 |
| ## | 8 | ADBE | Information Technology | 185.16 | 204.450 | 114.451 |
| ## | 9 | AAP | Consumer Discretionary | 109.63 | 169.550 | 78.810 |
| ## | 10 | AMD | Information Technology | 11.22 | 15.650 | 9.700 |

Our final data now has five variables and 505 observations.

# Aggregation using base R

### Aggregation using the aggregate function

When talking about aggregation, the aggregate function is the most obvious choice. We can use the aggregate function with data frames and time series. We can be specific with the signature if we are using the data frame by writing aggregate.data.frame. Both aggregate and aggregate.data.frame will result in the same data.

Aggregate is a generic function that can be used for both data frames and time series.

We will look at this function by different scenarios. Before the use of any aggregation, the use case must exist. It means why you want to aggregate data. This information is crucial as this will guide you to extract correctly summarized data that will answer your question.

So the scenario here is to “find out the total price of all shares by all sectors”. If you recall from the section above on qualitative and quantitative data, the variables sector and price represents qualitative and quantitative data, respectively. This information is crucial to put the variables into their correct place within the function.

aggregate(financials$Price, by = list(financials$Sector), FUN = sum

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Group.1 | x |
| ## | 1 | Consumer Discretionary | 10418.90 |
| ## | 2 | Consumer Staples | 2711.98 |
| ## | 3 | Energy | 1852.40 |
| ## | 4 | Financials | 6055.81 |
| ## | 5 | Health Care | 8083.46 |
| ## | 6 | Industrials | 7831.47 |
| ## | 7 | Information Technology | 8347.00 |
| ## | 8 | Materials | 2559.67 |
| ## | 9 | Real Estate | 2927.52 |
| ## | 10 | Telecommunication Services | 100.81 |
| ## | 11 | Utilities | 1545.45 |

### Assigning a name to an aggregated column in aggregate function in R

The result above shows that the function aggregate split the original data into small subsets by sector variable and applied sum function over the price variable. The result above shows x as the name of the summarised column. If you want to rename it, a little tweak in the code using setName function will do the

trick. Here is an example.

setNames(aggregate(financials$Price, by = list(financials$Sector), FUN = sum),c( Se ctor Total.Price

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | Total.Price |
| ## | 1 | Consumer Discretionary | 10418.90 |
| ## | 2 | Consumer Staples | 2711.98 |
| ## | 3 | Energy | 1852.40 |
| ## | 4 | Financials | 6055.81 |
| ## | 5 | Health Care | 8083.46 |
| ## | 6 | Industrials | 7831.47 |
| ## | 7 | Information Technology | 8347.00 |
| ## | 8 | Materials | 2559.67 |
| ## | 9 | Real Estate | 2927.52 |
| ## | 10 | Telecommunication Services | 100.81 |
| ## | 11 | Utilities | 1545.45 |

#### Aggregating multiple columns using the aggregate function in R

Now to apply the same function on multiple variables, all you have to do is to supply an expression to subset the required columns from the data frame as an argument. Here is an example, we have used setName function on top to assign meaningful names to the variables.

The scenario here is to “find out the average price, average 52-weeks low price, and the average 52-week high price of all shares by all sectors”.

setNames(aggregate(financials[,c( Price X52.Week.Low X52.Week.High")], by = lis t(financials$Sector), FUN = mean), c( Sector "Average.Price" Average.Low" "Averag e.High")

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  | Sector | Average.Price | Average.Low | Average.High |
| ## | 1 | Consumer Discretionary | 124.03452 | 146.93143 | 96.09236 |
| ## | 2 | Consumer Staples | 79.76412 | 92.83229 | 68.92944 |
| ## | 3 | Energy | 57.88750 | 72.58969 | 48.14123 |
| ## | 4 | Financials | 89.05603 | 101.82185 | 72.69447 |
| ## | 5 | Health Care | 132.51574 | 160.75853 | 103.71925 |
| ## | 6 | Industrials | 116.88761 | 134.57948 | 90.83702 |
| ## | 7 | Information Technology | 119.24286 | 138.77864 | 91.89142 |
| ## | 8 | Materials | 102.38680 | 118.03885 | 85.58325 |
| ## | 9 | Real Estate | 88.71273 | 110.55045 | 82.87809 |
| ## | 10 | Telecommunication Services | 33.60333 | 41.69333 | 29.50367 |
| ## | 11 | Utilities | 55.19464 | 68.49732 | 52.80232 |

#### Aggregating multiple columns data by multiple functions using the aggregate function in R

Moving on to the next level of the scenario, we would like to apply multiple functions over the different variables. It is possible by defining a custom function. The requirement now is to “find out the minimum and maximum values of price, 52-weeks low price of all shares by all sectors”. Here is an example.

aggregate(financials[c( Price X52.Week.Low )], by = list(financials$Sector), FUN

**function** x) c(min(x),max(x))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ## |  | Group.1 | Price.1 | Price.2 | X52.Week.Low.1 | X52.Week.Low.2 |
| ## | 1 | Consumer Discretionary | 10.43 | 1806.06 | 13.480 | 2067.990 |
| ## | 2 | Consumer Staples | 19.96 | 208.73 | 21.175 | 229.500 |
| ## | 3 | Energy | 2.82 | 169.16 | 6.590 | 199.830 |
| ## | 4 | Financials | 13.38 | 509.38 | 16.530 | 594.520 |
| ## | 5 | Health Care | 25.20 | 601.00 | 29.930 | 697.260 |
| ## | 6 | Industrials | 14.45 | 334.30 | 30.590 | 361.790 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ## | 7 | Information Technology | 11.22 | 1007.71 | 15.650 | 1198.000 |
| ## | 8 | Materials | 17.16 | 387.65 | 20.250 | 435.150 |
| ## | 9 | Real Estate | 14.01 | 409.98 | 21.530 | 495.345 |
| ## | 10 | Telecommunication Services | 16.20 | 49.04 | 27.610 | 54.770 |
| ## | 11 | Utilities | 10.06 | 145.29 | 12.050 | 159.640 |

The above command can be written in different ways. Here are a couple of examples. Note the use of cbind and formula (~).

aggregate(cbind(financials$Price,financials$X52.Week.Low) ~financials$Sector, FUN =

**function** x) c(min(x),max(x))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ## |  | financials$Sector | V1.1 | V1.2 | V2.1 | V2.2 |
| ## | 1 | Consumer Discretionary | 10.43 | 1806.06 | 13.480 | 2067.990 |
| ## | 2 | Consumer Staples | 19.96 | 208.73 | 21.175 | 229.500 |
| ## | 3 | Energy | 2.82 | 169.16 | 6.590 | 199.830 |
| ## | 4 | Financials | 13.38 | 509.38 | 16.530 | 594.520 |
| ## | 5 | Health Care | 25.20 | 601.00 | 29.930 | 697.260 |
| ## | 6 | Industrials | 14.45 | 334.30 | 30.590 | 361.790 |
| ## | 7 | Information Technology | 11.22 | 1007.71 | 15.650 | 1198.000 |
| ## | 8 | Materials | 17.16 | 387.65 | 20.250 | 435.150 |
| ## | 9 | Real Estate | 14.01 | 409.98 | 21.530 | 495.345 |
| ## | 10 | Telecommunication Services | 16.20 | 49.04 | 27.610 | 54.770 |
| ## | 11 | Utilities | 10.06 | 145.29 | 12.050 | 159.640 |

aggregate(cbind(Price,X52 Week.Low) ~ Sector, data = financials, FUN = **function** x) c(min(x),max(x))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ## |  | Sector | Price.1 | Price.2 | X52.Week.Low.1 | X52.Week.Low.2 |
| ## | 1 | Consumer Discretionary | 10.43 | 1806.06 | 13.480 | 2067.990 |
| ## | 2 | Consumer Staples | 19.96 | 208.73 | 21.175 | 229.500 |
| ## | 3 | Energy | 2.82 | 169.16 | 6.590 | 199.830 |
| ## | 4 | Financials | 13.38 | 509.38 | 16.530 | 594.520 |
| ## | 5 | Health Care | 25.20 | 601.00 | 29.930 | 697.260 |
| ## | 6 | Industrials | 14.45 | 334.30 | 30.590 | 361.790 |
| ## | 7 | Information Technology | 11.22 | 1007.71 | 15.650 | 1198.000 |
| ## | 8 | Materials | 17.16 | 387.65 | 20.250 | 435.150 |
| ## | 9 | Real Estate | 14.01 | 409.98 | 21.530 | 495.345 |
| ## | 10 | Telecommunication Services | 16.20 | 49.04 | 27.610 | 54.770 |
| ## | 11 | Utilities | 10.06 | 145.29 | 12.050 | 159.640 |

With the aggregate function, possibilities are endless. We can do a lot using the aggregate function. What we have seen in the sections above is just a part. Here are some other arguments which you can add to the aggregate function based on your requirements.

#### Argument Use

na.action Use this to specify if your data contain NA or missing values and how you want to handle them

simplify If used results will be presented in vector or matrix form

formula Depending on subsetting requirement formulas like y ~ x or cbind(y1,y2) ~ x1 + x2 can be used

data The data frame in question which has the variables to subset and aggregate

drop Use if you want to drop unused combination of grouping values

subset Use it to limit the operation to certain observations

Lastly, to name a few, you can use sum, count, first, last, mean, median, min, max, and sd functions with the aggregate function. If you think this list should have more function names, then please add a comment in the comment box below.

#### Aggregation using by() function

The by function is different from aggregate as it is a wrapper for tapply function, this means it also encapsulates the functionality of tapply function. The return value from “by” function depends on the usage of simplify argument. If the value of simplify argument is TRUE then it returns a list or array otherwise it always returns a list. Here is an example of by function. In this example, we are trying to find the sum of price variable by sector.

All other arguments in this function are self-explanatory except INDICES that represent factors or a list of factors.

by(data=financials$Price, INDICES = financials$Sector, FUN = sum, simplify = RU

## financials$Sector: Consumer Discretionary ## [1] 10418.9

##

## financials$Sector: Consumer Staples ## [1] 2711.98

##

## financials$Sector: Energy ## [1] 1852.4

## ## financials$Sector: Financials ## [1] 6055.81

## ## financials$Sector: Health Care ## [1] 8083.46

## ## financials$Sector: Industrials ## [1] 7831.47

##

## financials$Sector: Information Technology ## [1] 8347

## ## financials$Sector: Materials ## [1] 2559.67

## ## financials$Sector: Real Estate ## [1] 2927.52

##

## financials$Sector: Telecommunication Services ## [1] 100.81

## ## financials$Sector: Utilities ## [1] 1545.45

#### Aggregation using sweep() function

There is one more function I would like to mention here that you can use to aggregate data. Function sweep returns an array by sweeping out summary statistics. This specific function may not fit into all kinds of aggregation requirements but worth mentioning if you want to do some operation while summarising data. This function wouldn’t be my first choice though it may fit into yours. Here is an example.

sweep(financials[ Price ], MARGIN = , STATS = ,FUN = sum

## [1] 52434.47

The function above is returning the sum of variable Price. Over the years, I didn’t use the sweep function for my work. If you know the aggregation scenario where this function would be most useful, then please comment in the comment box below. All other arguments are obvious except STATS, my understanding of this argument is that this is

the value that you would like to apply to the variable in conjunction with the function argument. So, I have set STATS to 0 and used sum as the function, which results in the total Price variable. If I set STATS to 1, then all the values of variable Price would be increased with one, and the total will be different.

Though sweep function doesn’t result in grouped or subsetted data as others, I thought this function is worth mentioning. Similarly, other functions from the apply family are available to aggregate data, but you may encounter limitations on grouping or performance requirements. If you want, you can achieve aggregation results using these functions, but there are better and faster options available, and from this point onwards, we will concentrate on those.

**Aggregation using dplyr**

#### Aggregation using group\_by and summarize functions

The three functions we have seen above may fit into different aggregation scenarios, but in terms of ease of use, I consider the group\_by and summarize functions of the dplyr package. The performance of functions is subjective and dependent on the size and operation of the data. I personally never evaluated the performance though wherever I have read, group\_by is praised for its performance. Let’s look at an example of aggregating data using the group by function.

The scenario here is to “find out the total price of all shares by all sectors”.

financials %>% group\_by(Sector) %>% summarize(total.price = sum(Price)

## `summarise()` ungrouping output (override with `.groups` argument

|  |  |  |  |
| --- | --- | --- | --- |
| ## | 1 | Consumer Discretionary | 10419 |
| ## | 2 | Consumer Staples | 2712 |
| ## | 3 | Energy | 1852 |
| ## | 4 | Financials | 6056 |
| ## | 5 | Health Care | 8083 |
| ## | 6 | Industrials | 7831 |
| ## | 7 | Information Technology | 8347 |
| ## | 8 | Materials | 2560 |
| ## | 9 | Real Estate | 2928 |
| ## | 10 | Telecommunication Services | 101 |
| ## | 11 | Utilities | 1545 |

Simply supplying the variable name within the group\_by function is enough to set the grouping of observations by the value of the selected variable. The summarize function is the engine where the data is operated upon using the function of your choice. In the example above, I have chosen the sum function which is applied over grouped observation. This strategy or method of aggregation is known as split-apply-combine. First, we are splitting the observations into groups followed by applying a function and then combining results to present.

## # A tibble: 11 x 2

##

##

Sector

total.price

If you were observant, then you may have noticed a warning as well. This warning is a hint to remind you how you want to control the grouping of data. You can suppress this warning by using options(dplyr.summarise.inform = FALSE). For more information, please click this link.

Also, did you notice how the results are printed above?

Nothing wrong with that, it is because the return value is in the form of tibble. You can print.data.frame() function to print is in a nice format. Here is an example.

financials %>% group\_by(Sector) %>% summarize(total.price = sum(Price)) %>% print.d ata.frame(

## `summarise()` ungrouping output (override with `.groups` argument

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | total.price |
| ## | 1 | Consumer Discretionary | 10418.90 |
| ## | 2 | Consumer Staples | 2711.98 |
| ## | 3 | Energy | 1852.40 |
| ## | 4 | Financials | 6055.81 |
| ## | 5 | Health Care | 8083.46 |
| ## | 6 | Industrials | 7831.47 |
| ## | 7 | Information Technology | 8347.00 |
| ## | 8 | Materials | 2559.67 |
| ## | 9 | Real Estate | 2927.52 |
| ## | 10 | Telecommunication Services | 100.81 |
| ## | 11 | Utilities | 1545.45 |

Lastly, did you notice that the summarised variable is now named total.price? By simply putting the new name in front of the aggregation function will do the trick. It is not mandatory, but I am sure you wouldn’t like the default name.

#### Aggregation by group\_by and summarize functions with multiple variables and functions

Let’s progress ahead with our scenario and find out the total price, minimum 52-week low and maximum 52-week high price of all shares and by all sectors. The aim here is to show the use of multiple variables with multiple aggregation functions.

financials %>% group\_by(Sector) %>% summarize(total.price = sum(Price), min.52.wee k.low = min(X52.Week.Low), max.52.week.high = max(X52.Week.High)) %>% print.data.fr ame(

## `summarise()` ungrouping output (override with `.groups` argument

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  | Sector | total.price | min.52.week.low | max.52.week.high |
| ## | 1 | Consumer Discretionary | 10418.90 | 13.480 | 1589.0000 |
| ## | 2 | Consumer Staples | 2711.98 | 21.175 | 152.0100 |
| ## | 3 | Energy | 1852.40 | 6.590 | 125.4600 |
| ## | 4 | Financials | 6055.81 | 16.530 | 368.0000 |
| ## | 5 | Health Care | 8083.46 | 29.930 | 459.3400 |
| ## | 6 | Industrials | 7831.47 | 30.590 | 256.4000 |
| ## | 7 | Information Technology | 8347.00 | 15.650 | 824.3000 |
| ## | 8 | Materials | 2559.67 | 20.250 | 302.0101 |
| ## | 9 | Real Estate | 2927.52 | 21.530 | 361.9000 |
| ## | 10 | Telecommunication Services | 100.81 | 27.610 | 42.8000 |
| ## | 11 | Utilities | 1545.45 | 12.050 | 124.1800 |

Also, we can supply more than one variable in the group\_by function if you want to group data by multiple variables.

#### Aggregation using group\_by and summarize functions with a range of variables

You can use the summarize function in a variety of ways. If you have several variables to aggregate, then the following type of command would help.

financials %>% group\_by(Sector) %>% summarize(across(Price:X52.Week.High, ~sum( x))) %>% print.data.frame(

## `summarise()` ungrouping output (override with `.groups` argument

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  | Sector | Price | X52.Week.Low | X52.Week.High |
| ## | 1 | Consumer Discretionary | 10418.90 | 12342.240 | 8071.759 |
| ## | 2 | Consumer Staples | 2711.98 | 3156.298 | 2343.601 |
| ## | 3 | Energy | 1852.40 | 2322.870 | 1540.519 |
| ## | 4 | Financials | 6055.81 | 6923.886 | 4943.224 |
| ## | 5 | Health Care | 8083.46 | 9806.271 | 6326.874 |
| ## | 6 | Industrials | 7831.47 | 9016.825 | 6086.081 |
| ## | 7 | Information Technology | 8347.00 | 9714.505 | 6432.399 |
| ## | 8 | Materials | 2559.67 | 2950.971 | 2139.581 |
| ## | 9 | Real Estate | 2927.52 | 3648.165 | 2734.977 |
| ## | 10 | Telecommunication Services | 100.81 | 125.080 | 88.511 |
| ## | 11 | Utilities | 1545.45 | 1917.925 | 1478.465 |

#### Aggregation using group\_by and summarize\_if function

Or, if you want to aggregate all of the numeric variables from your data frame, then the following version of

summarize function would help.

financials %>% group\_by(Sector) %>% summarize\_if(is.numeric,sum) %>% print.data.fra me(

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  | Sector | Price | X52.Week.Low | X52.Week.High |
| ## | 1 | Consumer Discretionary | 10418.90 | 12342.240 | 8071.759 |
| ## | 2 | Consumer Staples | 2711.98 | 3156.298 | 2343.601 |
| ## | 3 | Energy | 1852.40 | 2322.870 | 1540.519 |
| ## | 4 | Financials | 6055.81 | 6923.886 | 4943.224 |
| ## | 5 | Health Care | 8083.46 | 9806.271 | 6326.874 |
| ## | 6 | Industrials | 7831.47 | 9016.825 | 6086.081 |
| ## | 7 | Information Technology | 8347.00 | 9714.505 | 6432.399 |
| ## | 8 | Materials | 2559.67 | 2950.971 | 2139.581 |
| ## | 9 | Real Estate | 2927.52 | 3648.165 | 2734.977 |
| ## | 10 | Telecommunication Services | 100.81 | 125.080 | 88.511 |
| ## | 11 | Utilities | 1545.45 | 1917.925 | 1478.465 |

#### Aggregation using group\_by and summarize\_at function

Summarize\_at, on the other hand, helps you to aggregate more than one variable in one go. The requirement is that the variables are supplied as vector or vars.

financials %>% group\_by(Sector) %>% summarize\_at(c( Price X52 Week.High"),sum) %

>% print.data.frame(

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## |  | Sector | Price | X52.Week.High |
| ## | 1 | Consumer Discretionary | 10418.90 | 8071.759 |
| ## | 2 | Consumer Staples | 2711.98 | 2343.601 |
| ## | 3 | Energy | 1852.40 | 1540.519 |
| ## | 4 | Financials | 6055.81 | 4943.224 |
| ## | 5 | Health Care | 8083.46 | 6326.874 |
| ## | 6 | Industrials | 7831.47 | 6086.081 |
| ## | 7 | Information Technology | 8347.00 | 6432.399 |
| ## | 8 | Materials | 2559.67 | 2139.581 |
| ## | 9 | Real Estate | 2927.52 | 2734.977 |
| ## | 10 | Telecommunication Services | 100.81 | 88.511 |
| ## | 11 | Utilities | 1545.45 | 1478.465 |

#### Aggregation using group\_by and summarize\_all function

Lastly, summarize\_all function helps to aggregate all the values from the data frame, and we can use multiple aggregation functions as well. Note in our data frame we have two character variables, let’s prepare the data for using summarize\_all function and aggregate data to show total and minimum values of all variable across the data frame.

financials %>% select(-Symbol) %>% group\_by(Sector) %>% summarize\_all(c(sum,min)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ## | # A tibble: 11 x 7 | | | | | | |
| ## |  | Sector | Price\_fn1 | X52.Week.Low\_fn1 | X52.Week.High\_f… | Price\_fn2 | X52.Week.Low\_fn2 |
| ## |  |  |  |  |  |  |  |
| ## | 1 | Consu… | 10419. | 12342. | 8072. | 10.4 | 13.5 |
| ## | 2 | Consu… | 2712. | 3156. | 2344. | 20.0 | 21.2 |
| ## | 3 | Energy | 1852. | 2323. | 1541. | 2.82 | 6.59 |
| ## | 4 | Finan… | 6056. | 6924. | 4943. | 13.4 | 16.5 |
| ## | 5 | Healt… | 8083. | 9806. | 6327. | 25.2 | 29.9 |
| ## | 6 | Indus… | 7831. | 9017. | 6086. | 14.4 | 30.6 |
| ## | 7 | Infor… | 8347 | 9715. | 6432. | 11.2 | 15.6 |
| ## | 8 | Mater… | 2560. | 2951. | 2140. | 17.2 | 20.2 |
| ## | 9 | Real … | 2928. | 3648. | 2735. | 14.0 | 21.5 |
| ## | 10 | Telec… | 101. | 125. | 88.5 | 16.2 | 27.6 |
| ## | 11 | Utili… | 1545. | 1918. | 1478. | 10.1 | 12.0 |

## Aggregation using data.table

## # … with 1 more variable: X52.Week.High\_fn2

#### Converting data frame to data table

Aggregation using the data table is a step further from dplyr in terms of simplicity. We write the aggregation code as if we are subsetting the data frame. Before we progress, let’s convert our data frame into a data table.

financials <- setDT(financials class(financials

## [1] "data.table" "data.frame"

#### Aggregating single variable using data table

Let’s look at an example where the scenario is to find out the total price of all shares by all sectors.

financials[,sum(Price), by=Sector

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | V1 |
| ## | 1: | Industrials | 7831.47 |
| ## | 2: | Health Care | 8083.46 |
| ## | 3: | Information Technology | 8347.00 |
| ## | 4: | Consumer Discretionary | 10418.90 |
| ## | 5: | Utilities | 1545.45 |
| ## | 6: | Financials | 6055.81 |
| ## | 7: | Materials | 2559.67 |
| ## | 8: | Real Estate | 2927.52 |
| ## | 9: | Consumer Staples | 2711.98 |
| ## | 10: | Energy | 1852.40 |
| ## | 11: | Telecommunication Services | 100.81 |

#### Assigning a custom name to an aggregated variable in data table

The result above shows the aggregated value by the selected variable. However, the aggregated variable’s name is

not friendly. Let’s give it a proper name.

financials[,list(total.price=sum(Price)), by=Sector

Or

financials[,.(total.price=sum(Price)), by=Sector

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | total.price |
| ## | 1: | Industrials | 7831.47 |
| ## | 2: | Health Care | 8083.46 |
| ## | 3: | Information Technology | 8347.00 |
| ## | 4: | Consumer Discretionary | 10418.90 |
| ## | 5: | Utilities | 1545.45 |
| ## | 6: | Financials | 6055.81 |
| ## | 7: | Materials | 2559.67 |
| ## | 8: | Real Estate | 2927.52 |
| ## | 9: | Consumer Staples | 2711.98 |
| ## | 10: | Energy | 1852.40 |
| ## | 11: | Telecommunication Services | 100.81 |

#### Aggregating multiple variables using multiple aggregation functions in data table

What we have seen until now is one custom-named variable aggregated by a different variable. Let take it further and see if we can apply more than one aggregation function over more than one variable. The scenario is to find out the total price and minimum 52-week low price from all shares by all sectors.

financials[,.(total.price=sum(Price), mim.52.week.low=min(X52.Week Low)), by=Secto r

Or

financials[,list(total.price=sum(Price), mim.52.week.low=min(X52.Week Low)), by=Sec tor

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## |  | Sector | total.price | mim.52.week.low |
| ## | 1: | Industrials | 7831.47 | 30.590 |
| ## | 2: | Health Care | 8083.46 | 29.930 |
| ## | 3: | Information Technology | 8347.00 | 15.650 |
| ## | 4: | Consumer Discretionary | 10418.90 | 13.480 |
| ## | 5: | Utilities | 1545.45 | 12.050 |
| ## | 6: | Financials | 6055.81 | 16.530 |
| ## | 7: | Materials | 2559.67 | 20.250 |
| ## | 8: | Real Estate | 2927.52 | 21.530 |
| ## | 9: | Consumer Staples | 2711.98 | 21.175 |
| ## | 10: | Energy | 1852.40 | 6.590 |
| ## | 11: | Telecommunication Services | 100.81 | 27.610 |

#### Aggregating variables as well as filtering observations in data table

What if we want to aggregate as well as filter the observations by supplying specific conditions. The data table offers a solution to this. Here is an example.

financials[Sector == Industrials ,list(total price=sum(Price), mim.52.week.low=min X52.Week Low)), by=Sector

##

Sector total.price mim.52.week.low

## 1: Industrials 7831.47 30.59

#### Counting the number of observations within a group in data table

Although we can use different aggregation functions, what if we want to count the observations within a group? We can do this using .N special variable.

financials[,.N, by=Sector

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | N |
| ## | 1: | Industrials | 67 |
| ## | 2: | Health Care | 61 |
| ## | 3: | Information Technology | 70 |
| ## | 4: | Consumer Discretionary | 84 |
| ## | 5: | Utilities | 28 |
| ## | 6: | Financials | 68 |
| ## | 7: | Materials | 25 |
| ## | 8: | Real Estate | 33 |
| ## | 9: | Consumer Staples | 34 |
| ## | 10: | Energy | 32 |
| ## | 11: | Telecommunication Services | 3 |

If the requirement is to group observations using multiple variables, then the following syntax can be used for

by argument.

by=.(variable to group 1, variable to group 2

## Aggregation using sqldf

I love sqldf, it makes things easy, but if you are not familiar with SQL then it may be a little challenging. Aggregation in sqldf require us to understand SQL, and hence I would just list one example here. SQL is an easy language to learn and I will soon be posing articles on SQL.

Let’s define our scenario, find out the total price and minimum 52-week low price from all shares by all sectors where aggregated total price is greater than 5000. This scenario includes more than one variable with the use of more than one aggregation function as well as filtration criterion over grouped observations.

sqldf( select Sector, sum(Price), min("X52.Week.Low") from financials group by Sect or having sum(Price) > 5000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## | 1 | Consumer | Discretionary | 10418.90 | 13.48 |
| ## | 2 |  | Financials | 6055.81 | 16.53 |
| ## | 3 |  | Health Care | 8083.46 | 29.93 |
| ## | 4 |  | Industrials | 7831.47 | 30.59 |

Let’s give aggregated columns a custom name.

##

Sector sum(Price) min("X52.Week.Low"

## 5 Information Technology

8347.00

15.65

sqldf( select Sector, sum(Price) as "total.price", min("X52.Week.Low") as "min.52.w eek.low" from financials group by Sector having sum(Price) > 5000

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ## |  |  | Sector | total.price | min.52.week.low |
| ## | 1 | Consumer | Discretionary | 10418.90 | 13.48 |
| ## | 2 |  | Financials | 6055.81 | 16.53 |
| ## | 3 |  | Health Care | 8083.46 | 29.93 |

|  |  |  |  |
| --- | --- | --- | --- |
| ## 4 | Industrials | 7831.47 | 30.59 |
| ## 5 | Information Technology | 8347.00 | 15.65 |

**Sorting in R**

## Why sort data?

It is a fact that the human brain looks for patterns. Identifying patterns is crucial while performing analysis, and sorted data helps a lot in finding these patterns. Sort help understanding the data, moreover, in profiling the data. Sort functionality help put data into an order which can provide information to help your analysis.

## Sorting data using base R

#### Sorting data using order function

In base R, function order help sort the data. The usage of the order function is straight forward. Here is an example.

financials[order(Symbol),c( Symbol" Price )

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Symbol | Price |
| ## | 1: | A | 65.05 |
| ## | 2: | AAL | 48.60 |
| ## | 3: | AAP | 109.63 |
| ## | 4: | AAPL | 155.15 |
| ## | 5: | ABBV | 108.48 |
| ## | --- |  |  |
| ## | 501: | XYL | 70.24 |
| ## | 502: | YUM | 76.30 |
| ## | 503: | ZBH | 115.53 |
| ## | 504: | ZION | 50.71 |
| ## | 505: | ZTS | 71.51 |

#### Sorting data in descending order using order function

We are subsetting variables Symbol and Price from data frame financials and sorting the output by Symbol variable’s values. If we want to sort the data in descending order, then function desc comes to our rescue.

financials[order(desc(Symbol) ,c("Symbol" Price )

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Symbol | Price |
| ## | 1: | ZTS | 71.51 |
| ## | 2: | ZION | 50.71 |
| ## | 3: | ZBH | 115.53 |
| ## | 4: | YUM | 76.30 |
| ## | 5: | XYL | 70.24 |
| ## | --- |  |  |
| ## | 501: | ABBV | 108.48 |
| ## | 502: | AAPL | 155.15 |
| ## | 503: | AAP | 109.63 |
| ## | 504: | AAL | 48.60 |
| ## | 505: | A | 65.05 |

## Sorting data using dplyr

#### Sorting data using the arrange function

dplyr package has a function arrange which help sort data. Here is an example.

financials %>% group\_by(Sector) %>% summarize(sum(Price)) %>% arrange(Sector) %>% p rint.data.frame(

## `summarise()` ungrouping output (override with `.groups` argument

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | sum(Price |
| ## | 1 | Consumer Discretionary | 10418.90 |
| ## | 2 | Consumer Staples | 2711.98 |
| ## | 3 | Energy | 1852.40 |
| ## | 4 | Financials | 6055.81 |
| ## | 5 | Health Care | 8083.46 |
| ## | 6 | Industrials | 7831.47 |
| ## | 7 | Information Technology | 8347.00 |
| ## | 8 | Materials | 2559.67 |
| ## | 9 | Real Estate | 2927.52 |
| ## | 10 | Telecommunication Services | 100.81 |
| ## | 11 | Utilities | 1545.45 |

#### Sorting data in descending order using arrange function

Let’s reorder the data in descending order.

financials %>% group\_by(Sector) %>% summarize(sum(Price)) %>% arrange(desc(Sector))

%>% print.data.frame(

## `summarise()` ungrouping output (override with `.groups` argument

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | sum(Price |
| ## | 1 | Utilities | 1545.45 |
| ## | 2 | Telecommunication Services | 100.81 |
| ## | 3 | Real Estate | 2927.52 |
| ## | 4 | Materials | 2559.67 |
| ## | 5 | Information Technology | 8347.00 |
| ## | 6 | Industrials | 7831.47 |
| ## | 7 | Health Care | 8083.46 |
| ## | 8 | Financials | 6055.81 |
| ## | 9 | Energy | 1852.40 |
| ## | 10 | Consumer Staples | 2711.98 |
| ## | 11 | Consumer Discretionary | 10418.90 |

## Sorting data using data table

#### Sorting data using order function

Data table has two functions order and setorder to help sort the data. Here is an example but first convert the data frame to data table and then sort.

financials <- setDT(financials financials[order(Sector),sum(Price), by=Sector

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | V1 |
| ## | 1: | Consumer Discretionary | 10418.90 |
| ## | 2: | Consumer Staples | 2711.98 |
| ## | 3: | Energy | 1852.40 |
| ## | 4: | Financials | 6055.81 |
| ## | 5: | Health Care | 8083.46 |

|  |  |  |  |
| --- | --- | --- | --- |
| ## | 6: | Industrials | 7831.47 |
| ## | 7: | Information Technology | 8347.00 |
| ## | 8: | Materials | 2559.67 |
| ## | 9: | Real Estate | 2927.52 |
| ## | 10: | Telecommunication Services | 100.81 |
| ## | 11: | Utilities | 1545.45 |

#### Sorting data in descending order using order function

Let’s reverse the order now using desc function.

financials[order(desc(Sector) ,sum(Price), by=Sector

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Sector | V1 |
| ## | 1: | Utilities | 1545.45 |
| ## | 2: | Telecommunication Services | 100.81 |
| ## | 3: | Real Estate | 2927.52 |
| ## | 4: | Materials | 2559.67 |
| ## | 5: | Information Technology | 8347.00 |
| ## | 6: | Industrials | 7831.47 |
| ## | 7: | Health Care | 8083.46 |
| ## | 8: | Financials | 6055.81 |
| ## | 9: | Energy | 1852.40 |
| ## | 10: | Consumer Staples | 2711.98 |
| ## | 11: | Consumer Discretionary | 10418.90 |

#### Sorting data using setorder function

Now let’s look at the sorting using setorder function.

order.data <- setorder(financials[,c( Symbol" Price )],Symbol order.data

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Symbol | Price |
| ## | 1: | A | 65.05 |
| ## | 2: | AAL | 48.60 |
| ## | 3: | AAP | 109.63 |
| ## | 4: | AAPL | 155.15 |
| ## | 5: | ABBV | 108.48 |
| ## | --- |  |  |
| ## | 501: | XYL | 70.24 |
| ## | 502: | YUM | 76.30 |
| ## | 503: | ZBH | 115.53 |
| ## | 504: | ZION | 50.71 |
| ## | 505: | ZTS | 71.51 |

#### Sorting data in descending order using setorder function

Same setorder function can be used to reverse the order of the data by simply putting a minus sign in the front. Here is an example.

order.data <- setorder(financials[,c( Symbol" Price )],-Symbol order.data

|  |  |  |  |
| --- | --- | --- | --- |
| ## |  | Symbol | Price |
| ## | 1: | ZTS | 71.51 |
| ## | 2: | ZION | 50.71 |
| ## | 3: | ZBH | 115.53 |
| ## | 4: | YUM | 76.30 |

## 5:

## --- ## 501:

## 502:

## 503:

## 504:

## 505:

XYL 70.24

ABBV 108.48

AAPL 155.15

AAP 109.63

AAL 48.60

A 65.05

## Sorting data using sqldf

Data sorting using sqldf is simply achieved using the “order by” clause in the SQL command.

financials <- read.csv( constituents-financials\_csv.csv"

sqldf( select Sector, sum(Price), min("X52.Week.Low") from financials group by Sect or order by Sector

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## |  | Sector | sum(Price) | min("X52.Week.Low" |
| ## | 1 | Consumer Discretionary | 10418.90 | 13.480 |
| ## | 2 | Consumer Staples | 2711.98 | 21.175 |
| ## | 3 | Energy | 1852.40 | 6.590 |
| ## | 4 | Financials | 6055.81 | 16.530 |
| ## | 5 | Health Care | 8083.46 | 29.930 |
| ## | 6 | Industrials | 7831.47 | 30.590 |
| ## | 7 | Information Technology | 8347.00 | 15.650 |
| ## | 8 | Materials | 2559.67 | 20.250 |
| ## | 9 | Real Estate | 2927.52 | 21.530 |
| ## | 10 | Telecommunication Services | 100.81 | 27.610 |
| ## | 11 | Utilities | 1545.45 | 12.050 |

Reversing the order of the data using desc clause. Here is an example.

sqldf( select Sector, sum(Price), min("X52.Week.Low") from financials group by Sect or order by Sector desc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## |  | Sector | sum(Price) | min("X52.Week.Low" |
| ## | 1 | Utilities | 1545.45 | 12.050 |
| ## | 2 | Telecommunication Services | 100.81 | 27.610 |
| ## | 3 | Real Estate | 2927.52 | 21.530 |
| ## | 4 | Materials | 2559.67 | 20.250 |
| ## | 5 | Information Technology | 8347.00 | 15.650 |
| ## | 6 | Industrials | 7831.47 | 30.590 |
| ## | 7 | Health Care | 8083.46 | 29.930 |
| ## | 8 | Financials | 6055.81 | 16.530 |
| ## | 9 | Energy | 1852.40 | 6.590 |
| ## | 10 | Consumer Staples | 2711.98 | 21.175 |
| ## | 11 | Consumer Discretionary | 10418.90 | 13.480 |

Concludingly, you may have noticed that we have not discussed NA values at all. Almost all aggregation function allows na.rm argument. Please try and comment if you find out that any particular function did not support it.

Well done, you’ve made to the end. Thank you for reading this article. I hope you’ve liked it.…