Year-to-Date Average & Six-Day Moving Average

Ted Kim

2022-09-12

Team Member

Seungmin Song

Create a connection object to MySQL database.

```
con = dbConnect(RMySQL::MySQL(),
                user='root',
                password='',
                host='localhost')
```

Create database *stock* if not exists, and use database *stock*

```
res <- dbSendQuery(con, 'create database if not exists stock;')

res <- dbSendQuery(con, 'use stock;')
```

Load data

The data were taken from the link below to the csv file, and the format of some columns was modified.
https://www.nasdaq.com/market-activity/stocks/amd/historical
https://www.nasdaq.com/market-activity/stocks/intc/historical

```
dfcsv <- read.csv('stockprice.csv')
tableName <- 'stock'
dbWriteTable(con,
             tableName,
             dfcsv,
             field.types=c(tradingdate='date',
                           closeprice='double(8,2)',
                           volume='bigint',
                           open='double(8,2)',
                           high='double(8,2)',
                           low='double(8,2)',
                           symbol='varchar(10)'),
             row.names=FALSE,
             overwrite=TRUE)
```

[1] TRUE

```
query <- "select * from stock"
stock <- fetch(dbSendQuery(con, query))
```

Overall Average

This was unintentionally found because "order by" was not entered after entering "partition by". **avg(x) over (partition by)** does **not** retrieve **Year-to-Date average** but takes the overall average. The result is as follows:

select
    symbol, tradingdate,
    closeprice, avg(closeprice) over (partition by symbol) as overall_avg_closeprice,
    high, avg(high) over (partition by symbol) as overall_avg_high
from
    stock
where
    tradingdate < '2022-01-31' and symbol = 'amd'
order by
    tradingdate asc

symbol	tradingdate	closeprice	overall_avg_closepri...	high	overall_avg_high
amd	2022-01-03	150.24	127.502632	151.65	131.836316
amd	2022-01-04	144.42	127.502632	152.42	131.836316
amd	2022-01-05	136.15	127.502632	143.76	131.836316
amd	2022-01-06	136.23	127.502632	138.00	131.836316
amd	2022-01-07	132.00	127.502632	137.44	131.836316
amd	2022-01-10	132.00	127.502632	132.42	131.836316
amd	2022-01-11	137.31	127.502632	138.99	131.836316
amd	2022-01-12	137.47	127.502632	140.57	131.836316
amd	2022-01-13	132.74	127.502632	141.25	131.836316
amd	2022-01-14	136.88	127.502632	137.00	131.836316
amd	2022-01-18	131.93	127.502632	136.39	131.836316

Figure 1: overall average

Using **partition by** to divide the result set into partitions and calculate each subset of partitioned.Therefore, since the symbol is partitioned, it has the whole subset of the same symbol, so it returns the overall average, not Year-to_Date average, as the result value.

The R syntax corresponding to the above query statement is as follows:

```
overall_avg <- stock[order(stock$tradingdate),
                     c('symbol', 'tradingdate', 'closeprice', 'high')] %>%
                filter(tradingdate < '2022-01-31' & symbol == 'amd') %>%
                mutate (
                  overall_avg_closeprice = mean(closeprice),
                  overall_avg_high = mean(high)
                )
print(overall_avg[, c('symbol', 'tradingdate',
                      'closeprice', 'overall_avg_closeprice',
                      'high', 'overall_avg_high')])
```

```
##    symbol tradingdate closeprice overall_avg_closeprice   high overall_avg_high
## 1     amd  2022-01-03     150.24               127.5026 151.65         131.8363
## 2     amd  2022-01-04     144.42               127.5026 152.42         131.8363
## 3     amd  2022-01-05     136.15               127.5026 143.76         131.8363
## 4     amd  2022-01-06     136.23               127.5026 138.00         131.8363
## 5     amd  2022-01-07     132.00               127.5026 137.44         131.8363
```

```
## 6     amd  2022-01-10     132.00              127.5026 132.42         131.8363
## 7     amd  2022-01-11     137.31              127.5026 138.99         131.8363
## 8     amd  2022-01-12     137.47              127.5026 140.57         131.8363
## 9     amd  2022-01-13     132.74              127.5026 141.25         131.8363
## 10    amd  2022-01-14     136.88              127.5026 137.00         131.8363
## 11    amd  2022-01-18     131.93              127.5026 136.39         131.8363
## 12    amd  2022-01-19     128.27              127.5026 134.57         131.8363
## 13    amd  2022-01-20     121.89              127.5026 128.51         131.8363
## 14    amd  2022-01-21     118.81              127.5026 125.02         131.8363
## 15    amd  2022-01-24     116.53              127.5026 116.77         131.8363
## 16    amd  2022-01-25     111.13              127.5026 114.82         131.8363
## 17    amd  2022-01-26     110.71              127.5026 117.16         131.8363
## 18    amd  2022-01-27     102.60              127.5026 112.75         131.8363
## 19    amd  2022-01-28     105.24              127.5026 105.40         131.8363
```

Year-to-Date Average

avg(x) over (partition by ... order by asc) retrieves **Year-to-Date average**. The result is as follows:

select
    symbol, tradingdate,
    closeprice, avg(closeprice) over (partition by symbol order by tradingdate asc) as ytd_closeprice
    high, avg(high) over (partition by symbol order by tradingdate asc) as ytd_high
from
    stock
where
    tradinddate < '2022-01-31' and
    symbol = 'amd'

	symbol	tradingdate	closeprice	ytd_closeprice	high	ytd_high
▶	amd	2022-01-03	150.24	150.240000	151.65	151.650000
	amd	2022-01-04	144.42	147.330000	152.42	152.035000
	amd	2022-01-05	136.15	143.603333	143.76	149.276667
	amd	2022-01-06	136.23	141.760000	138.00	146.457500
	amd	2022-01-07	132.00	139.808000	137.44	144.654000
	amd	2022-01-10	132.00	138.506667	132.42	142.615000
	amd	2022-01-11	137.31	138.335714	138.99	142.097143
	amd	2022-01-12	137.47	138.227500	140.57	141.906250

Figure 2: year-to-date average

The difference between this query and the previous query is whether or not it is sorted for a partitioned. By using "ORDER BY", the query obtains cumulative values from the beginning to the current row. Since this query used the avg() function, it takes the average from the beginning to the current row.

The R syntax corresponding to the above query statement is as follows:
The **cuumean()** function returns the accumulated mean value.

```
ytd <- stock[order(stock$tradingdate),
             c('symbol', 'tradingdate', 'closeprice', 'high')] %>%
```

```
        filter(tradingdate < '2022-01-31' & symbol == 'amd') %>%
        mutate (
          ytd_closeprice = cummean(closeprice),
          ytd_high = cummean(high)
        )
print(ytd[, c('symbol','tradingdate','closeprice','ytd_closeprice','high','ytd_high')])
```

```
##    symbol tradingdate closeprice ytd_closeprice   high ytd_high
## 1     amd  2022-01-03     150.24       150.2400 151.65 151.6500
## 2     amd  2022-01-04     144.42       147.3300 152.42 152.0350
## 3     amd  2022-01-05     136.15       143.6033 143.76 149.2767
## 4     amd  2022-01-06     136.23       141.7600 138.00 146.4575
## 5     amd  2022-01-07     132.00       139.8080 137.44 144.6540
## 6     amd  2022-01-10     132.00       138.5067 132.42 142.6150
## 7     amd  2022-01-11     137.31       138.3357 138.99 142.0971
## 8     amd  2022-01-12     137.47       138.2275 140.57 141.9062
## 9     amd  2022-01-13     132.74       137.6178 141.25 141.8333
## 10    amd  2022-01-14     136.88       137.5440 137.00 141.3500
## 11    amd  2022-01-18     131.93       137.0336 136.39 140.8991
## 12    amd  2022-01-19     128.27       136.3033 134.57 140.3717
## 13    amd  2022-01-20     121.89       135.1946 128.51 139.4592
## 14    amd  2022-01-21     118.81       134.0243 125.02 138.4279
## 15    amd  2022-01-24     116.53       132.8580 116.77 136.9840
## 16    amd  2022-01-25     111.13       131.5000 114.82 135.5987
## 17    amd  2022-01-26     110.71       130.2771 117.16 134.5141
## 18    amd  2022-01-27     102.60       128.7394 112.75 133.3050
## 19    amd  2022-01-28     105.24       127.5026 105.40 131.8363
```

Six-Day Moving Average

select
    symbol, tradingdate, closeprice, avg(closeprice) over (order by symbol, tradingdate
    rows between 5 preceding and current row) as ma_closeprice, high,
    avg(high) over (order by symbol, tradingdate rows between 5 preceding and current row) as ma_high
from
    stock
where
    symbol = 'amd'

	symbol	tradingdate	closeprice	ma_closepri...	high	ma_high
▶	amd	2022-01-03	150.24	150.240000	151.65	151.650000
	amd	2022-01-04	144.42	147.330000	152.42	152.035000
	amd	2022-01-05	136.15	143.603333	143.76	149.276667
	amd	2022-01-06	136.23	141.760000	138.00	146.457500
	amd	2022-01-07	132.00	139.808000	137.44	144.654000
	amd	2022-01-10	132.00	138.506667	132.42	142.615000
	amd	2022-01-11	137.31	136.351667	138.99	140.505000
	amd	2022-01-12	137.47	135.193333	140.57	138.530000

Figure 3: six-day moving average

After sorting the rows, take the current row and the five preceding rows to average them.

The R syntax corresponding to the above query statement is as follows:
In order to find ma through googling, a package called zoo must be installed, but since the algorithm of simple moving average is very simple, so, decided to write and use the function ourselves.

```
moving_avg <- function(x, n) {
  res <- c()
  for(i in 1:length(x)){
    res <- c(res, ifelse(i < n, mean(x[1:i]), mean(x[(i-(n-1)):i])))
  }
  return(res)
}

ma <- stock[order(stock$tradingdate),
            c('symbol', 'tradingdate', 'closeprice', 'high')] %>%
      filter(tradingdate < '2022-01-31' & symbol == 'amd') %>%
      mutate (
        ma_closeprice = moving_avg(closeprice, 6),
        ma_high = moving_avg(high, 6)
      )
print(ma[, c('symbol','tradingdate','closeprice','ma_closeprice','high','ma_high')])
```

```
##    symbol tradingdate closeprice ma_closeprice   high  ma_high
## 1     amd  2022-01-03     150.24      150.2400 151.65 151.6500
## 2     amd  2022-01-04     144.42      147.3300 152.42 152.0350
## 3     amd  2022-01-05     136.15      143.6033 143.76 149.2767
## 4     amd  2022-01-06     136.23      141.7600 138.00 146.4575
## 5     amd  2022-01-07     132.00      139.8080 137.44 144.6540
## 6     amd  2022-01-10     132.00      138.5067 132.42 142.6150
## 7     amd  2022-01-11     137.31      136.3517 138.99 140.5050
## 8     amd  2022-01-12     137.47      135.1933 140.57 138.5300
## 9     amd  2022-01-13     132.74      134.6250 141.25 138.1117
## 10    amd  2022-01-14     136.88      134.7333 137.00 137.9450
## 11    amd  2022-01-18     131.93      134.7217 136.39 137.7700
## 12    amd  2022-01-19     128.27      134.1000 134.57 138.1283
## 13    amd  2022-01-20     121.89      131.5300 128.51 136.3817
## 14    amd  2022-01-21     118.81      128.4200 125.02 133.7900
## 15    amd  2022-01-24     116.53      125.7183 116.77 129.7100
## 16    amd  2022-01-25     111.13      121.4267 114.82 126.0133
## 17    amd  2022-01-26     110.71      117.8900 117.16 122.8083
## 18    amd  2022-01-27     102.60      113.6117 112.75 119.1717
## 19    amd  2022-01-28     105.24      110.8367 105.40 115.3200
```

The argument transmitted as a function in mutate is an object or a variable. For example, if the column name is closeprice, the entire closeprice column is transmitted as one object [ex. c('10', '20', '30', '40',...) or 1, 'a', etc.] In addition, the return value from the function must have the same number of elements as the elements originally transmitted. The moving_avg() function requires two arguments. The first argument accepts a column as a vector. The second factor is the number of rows to be treated as variables. The logic is very simple. First, generates an object **res <- c()** for return. Then, executes for...loop statement for number of element. In the case of the 'six-day moving average', if the variable i is less than or equal to 5, only an average from 1 to the current i value is obtained. And if the variable i exceeds 5, an average of the current i value and the previous five values are obtained.

x = vector (column with values of data frame)
n = number variable (2-day, 3-day, 4-day, ..., n-day)
i = index of x -> 1 to length(x) res = result variable to return

res <- c(res, value) -> res = (1), res <- c(res, 2) then res = (1, 2)

```
moving_avg <- function(x, n) {
res <- c()
for(i in 1:length(x)){
res <- c(res, ifelse(i < n, mean(x[1:i]), mean(x[(i-(n-1)):i])))
}
return(res)
}
```

[example]
x = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
n = 6
i = 1 -> res = (1)
i = 2 -> res = (1, 1.5)
i = 3 -> res = (1, 1.5, 2)
i = 4 -> res = (1, 1.5, 2, 2.5)
i = 5 -> res = (1, 1.5, 2, 2.5, 3)
i = 6 -> res = (1, 1.5, 2, 2.5, 3, 3.5)
i = 7 -> res = (1, 1.5, 2, 2.5, 3, 3.5, 4.5)
i = 8 -> res = (1, 1.5, 2, 2.5, 3, 3.5, 4.5, 5.5)
i = 9 -> res = (1, 1.5, 2, 2.5, 3, 3.5, 4.5, 5.5, 6.5)
i = 10 -> res = (1, 1.5, 2, 2.5, 3, 3.5, 4.5, 5.5, 6.5, 7.5)