SPS_DATA607_Week3B_DC

David Chen

Window Functions

1. Find (or ask an LLM to generate!) a dataset that includes time series for two or more separate items. For example, you could use end of day stock or cryptocurrency prices since Jan 1, 2022 for several instruments.
2. Use window functions (in SQL or dplyr) to calculate the year-to-date average and the six-day moving averages for each item. ## Approach

I have daily temperature reports for four buildings. Each report contains 24 temperature readings collected every 15 minutes. I plan to combine these CSV files into a single dataset, then calculate the YTD average and a 6-day moving average.

The 6-day moving average will help identify temperature drops, which may indicate boiler issues such as leaks or air vent locks in distributed steam pipes

Data set from https://www.bitget.com/price/filecoin/historical-data

Running Code

```r
library(dplyr)
```

```
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
```

```r
library(ggplot2)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v forcats   1.0.1     v stringr   1.6.0
v lubridate 1.9.4     v tibble    3.3.1
v purrr     1.2.1     v tidyr     1.3.2
v readr     2.1.6

-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
mydata <- read.csv("https://raw.githubusercontent.com/dyc-sps/SPS_DATA607_Week3B/refs/heads/m
summary(mydata)
```

```
    timeOpen            timeClose            timeHigh        
 Min.   :1.535e+12   Min.   :1.535e+12   Min.   :1.535e+12  
 1st Qu.:1.594e+12   1st Qu.:1.594e+12   1st Qu.:1.594e+12  
 Median :1.652e+12   Median :1.652e+12   Median :1.652e+12  
 Mean   :1.653e+12   Mean   :1.653e+12   Mean   :1.653e+12  
 3rd Qu.:1.712e+12   3rd Qu.:1.712e+12   3rd Qu.:1.712e+12  
 Max.   :1.771e+12   Max.   :1.771e+12   Max.   :1.771e+12  
    timeLow            priceOpen           priceHigh           priceLow       
 Min.   :1.535e+12   Min.   :  0.8819   Min.   :  0.9422   Min.   :  0.6336  
 1st Qu.:1.594e+12   1st Qu.:  3.5518   1st Qu.:  3.7537   1st Qu.:  3.3225  
 Median :1.652e+12   Median :  5.0781   Median :  5.3625   Median :  4.7555  
 Mean   :1.653e+12   Mean   : 15.3725   Mean   : 16.1979   Mean   : 14.5478  
 3rd Qu.:1.712e+12   3rd Qu.: 13.0953   3rd Qu.: 14.4798   3rd Qu.: 11.8922  
 Max.   :1.771e+12   Max.   :191.1539   Max.   :237.2418   Max.   :182.7141  
   priceClose           volume         
 Min.   :  0.8819   Min.   :5.112e+04  
 1st Qu.:  3.5471   1st Qu.:1.433e+07  
 Median :  5.0790   Median :1.361e+08  
 Mean   : 15.3672   Mean   :2.967e+08  
 3rd Qu.: 13.0927   3rd Qu.:2.995e+08  
 Max.   :191.3565   Max.   :1.237e+10  
```

```r
head(mydata)
```

```
     timeOpen   timeClose    timeHigh     timeLow priceOpen priceHigh  priceLow
1 1.77072e+12 1.77081e+12 1.77073e+12 1.77079e+12 0.9371375 0.9422195 0.8818362
2 1.77064e+12 1.77072e+12 1.77065e+12 1.77068e+12 0.9327586 0.9459684 0.8937898
3 1.77055e+12 1.77064e+12 1.77056e+12 1.77061e+12 0.9756082 0.9799408 0.9228659
4 1.77047e+12 1.77055e+12 1.77053e+12 1.77049e+12 0.9765257 0.9957679 0.9464217
5 1.77038e+12 1.77047e+12 1.77044e+12 1.77038e+12 0.8819155 0.9852578 0.7998295
6 1.77029e+12 1.77038e+12 1.77030e+12 1.77037e+12 1.0483603 1.0572914 0.8800285
  priceClose    volume
1  0.9004138  81084504
2  0.9371375  92338061
3  0.9327679  81278743
4  0.9756082 137038993
5  0.9765257 254306881
6  0.8818979 247946373
```

```r
glimpse(mydata)
```

```
Rows: 2,713
Columns: 9
$ timeOpen   <dbl> 1.77072e+12, 1.77064e+12, 1.77055e+12, 1.77047e+12, 1.77038~
$ timeClose  <dbl> 1.77081e+12, 1.77072e+12, 1.77064e+12, 1.77055e+12, 1.77047~
$ timeHigh   <dbl> 1.77073e+12, 1.77065e+12, 1.77056e+12, 1.77053e+12, 1.77044~
$ timeLow    <dbl> 1.77079e+12, 1.77068e+12, 1.77061e+12, 1.77049e+12, 1.77038~
$ priceOpen  <dbl> 0.9371375, 0.9327586, 0.9756082, 0.9765257, 0.8819155, 1.04~
$ priceHigh  <dbl> 0.9422195, 0.9459684, 0.9799408, 0.9957679, 0.9852578, 1.05~
$ priceLow   <dbl> 0.8818362, 0.8937898, 0.9228659, 0.9464217, 0.7998295, 0.88~
$ priceClose <dbl> 0.9004138, 0.9371375, 0.9327679, 0.9756082, 0.9765257, 0.88~
$ volume     <dbl> 81084504, 92338061, 81278743, 137038993, 254306881, 2479463~
```

Convert the timestamp column to UTC timezone format.

```r
mydata <- mydata %>%
  mutate(timeCl = as.Date(as.POSIXct(mydata$timeClose / 1000, origin = "1970-01-01", tz = "U
  mutate(timeOp = as.Date(as.POSIXct(mydata$timeOpen / 1000, origin = "1970-01-01", tz = "UTC
  mutate(timeHi = as.Date(as.POSIXct(mydata$timeHigh / 1000, origin = "1970-01-01", tz = "UTC
  mutate(timeLo = as.Date(as.POSIXct(mydata$timeLow / 1000, origin = "1970-01-01", tz = "UTC"
colnames(mydata)
```

```
 [1] "timeOpen"   "timeClose"  "timeHigh"   "timeLow"    "priceOpen"
 [6] "priceHigh"  "priceLow"   "priceClose" "volume"     "timeCl"
[11] "timeOp"     "timeHi"     "timeLo"
```

```
mydata_utc <- mydata %>%
  select(timeopen=timeOp,priceOpen,priceClose,priceHigh,priceLow,volume) %>%
  # select(timeopen=timeOp,timeclose=timeCl,timehigh=timeHi,timelow=timeLo,priceOpen,priceCl
  mutate(volume_million = round(volume / 1e6, 2))
summary(mydata_utc)
```

```
    timeopen            priceOpen          priceClose          priceHigh
 Min.   :2018-08-22   Min.   :  0.8819   Min.   :  0.8819   Min.   :  0.9422
 1st Qu.:2020-06-30   1st Qu.:  3.5518   1st Qu.:  3.5471   1st Qu.:  3.7537
 Median :2022-05-09   Median :  5.0781   Median :  5.0790   Median :  5.3625
 Mean   :2022-05-14   Mean   : 15.3725   Mean   : 15.3672   Mean   : 16.1979
 3rd Qu.:2024-04-03   3rd Qu.: 13.0953   3rd Qu.: 13.0927   3rd Qu.: 14.4798
 Max.   :2026-02-10   Max.   :191.1539   Max.   :191.3565   Max.   :237.2418
    priceLow            volume          volume_million
 Min.   :  0.6336   Min.   :5.112e+04   Min.   :    0.05
 1st Qu.:  3.3225   1st Qu.:1.433e+07   1st Qu.:   14.33
 Median :  4.7555   Median :1.361e+08   Median :  136.07
 Mean   : 14.5478   Mean   :2.967e+08   Mean   :  296.68
 3rd Qu.: 11.8922   3rd Qu.:2.995e+08   3rd Qu.:  299.47
 Max.   :182.7141   Max.   :1.237e+10   Max.   :12367.98
```

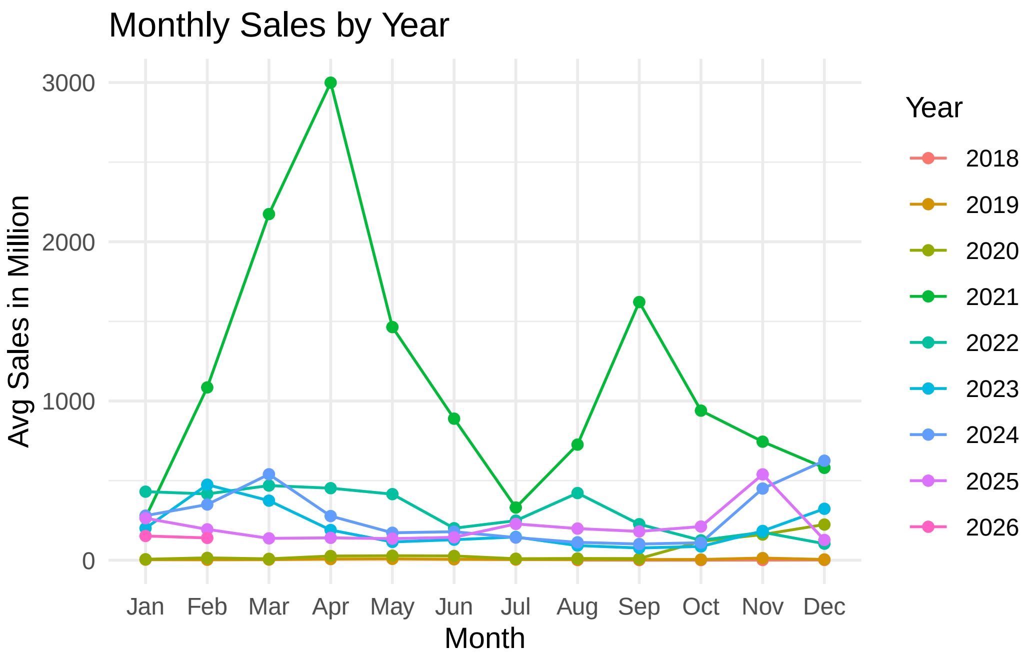
Create year and month fields to support grouping and aggregation.

```
mydata_utc_monthly <- mydata_utc %>%
  mutate(year = year(timeopen),month = month(timeopen,label = TRUE, abbr = TRUE)) %>%
  group_by(year, month)%>%
  summarise(avg_sales = mean(volume_million, na.rm = TRUE), .groups = "drop")

labels <- mydata_utc_monthly %>%
  group_by(year) %>%
  filter(month == max(month)) %>%
  ungroup()

ggplot(mydata_utc_monthly, aes(x = month, y = avg_sales, group = year, color = factor(year)))
  geom_line() +
  geom_point() +
  #geom_text(data = labels,aes(label = year), hjust = -0.2, vjust = 0.5, size = 4) +
```

```
  labs(title = "Monthly Sales by Year",
       x = "Month",
       y = "Avg Sales in Million",
       color = "Year") +
  scale_x_discrete(drop = FALSE) +
  theme_minimal()
```



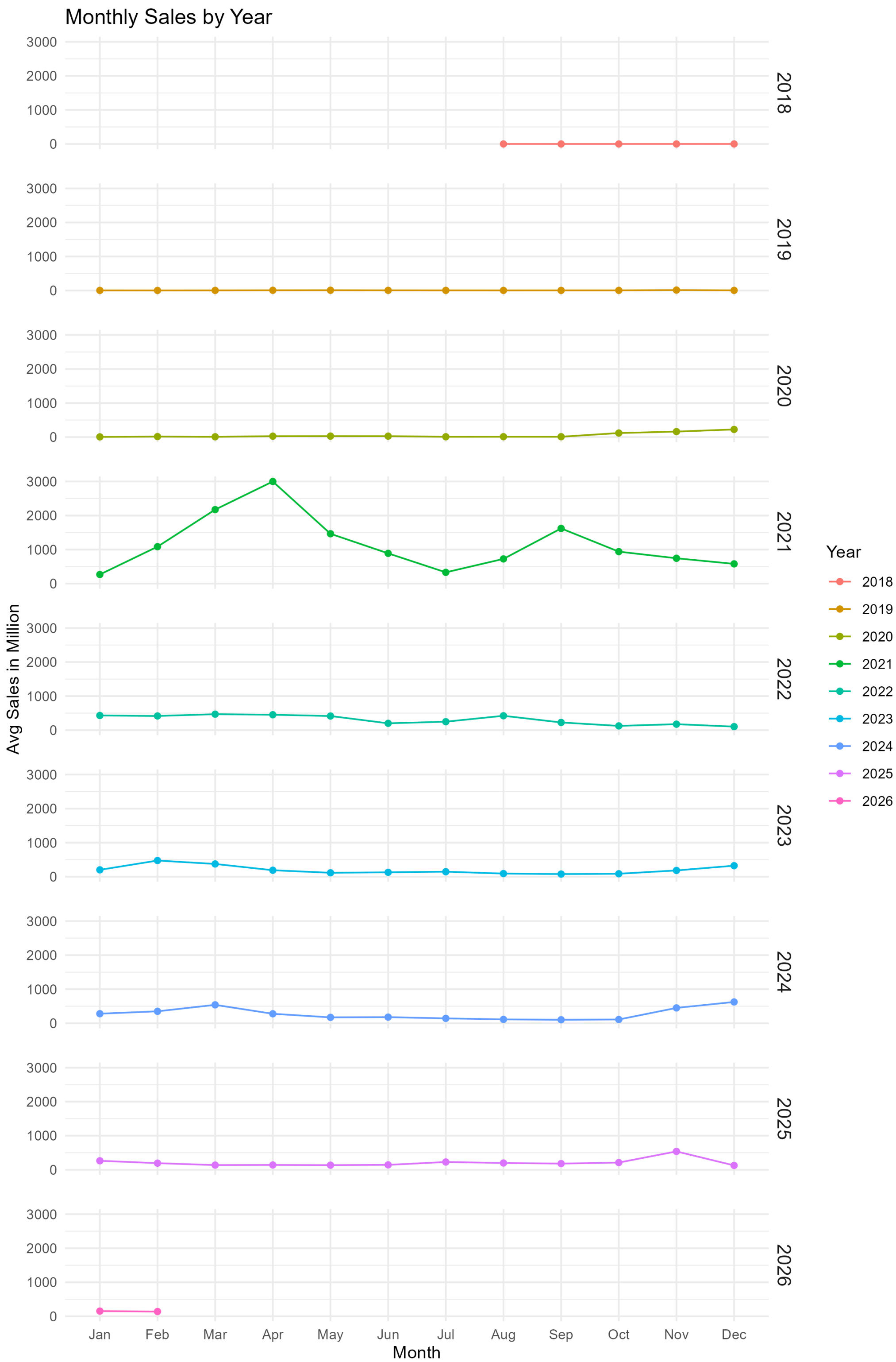

```
ggplot(mydata_utc_monthly, aes(x = month, y = avg_sales, group = year, color = factor(year))
  geom_line() +
  geom_point() +
  #geom_text(data = labels,aes(label = year), hjust = -0.2, vjust = 0.5, size = 4) +
  labs(title = "Monthly Sales by Year",
       x = "Month",
       y = "Avg Sales in Million",
       color = "Year") +
  scale_x_discrete(drop = FALSE) +

  theme_minimal()+

  #facet_wrap(~year)
  #facet_wrap(~year, ncol = 1, scales = "free_y")
  #facet_wrap(~year, ncol = 1)
```

```
facet_grid(year ~.)+
theme(panel.spacing = unit(1.5, "lines"),
        strip.text.y = element_text(size = 12))
```

```
ggsave("monthly_sales.png", width = 8, height = 12)
```


Monthly Sales by Year
Avg Sales in Million
Month
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
0
1000
2000
3000
2018
2019
2020
2021
2022
2023
2024
2025
2026
Year
2018
2019
2020
2021
2022
2023
2024
2025
2026

6-days moving average

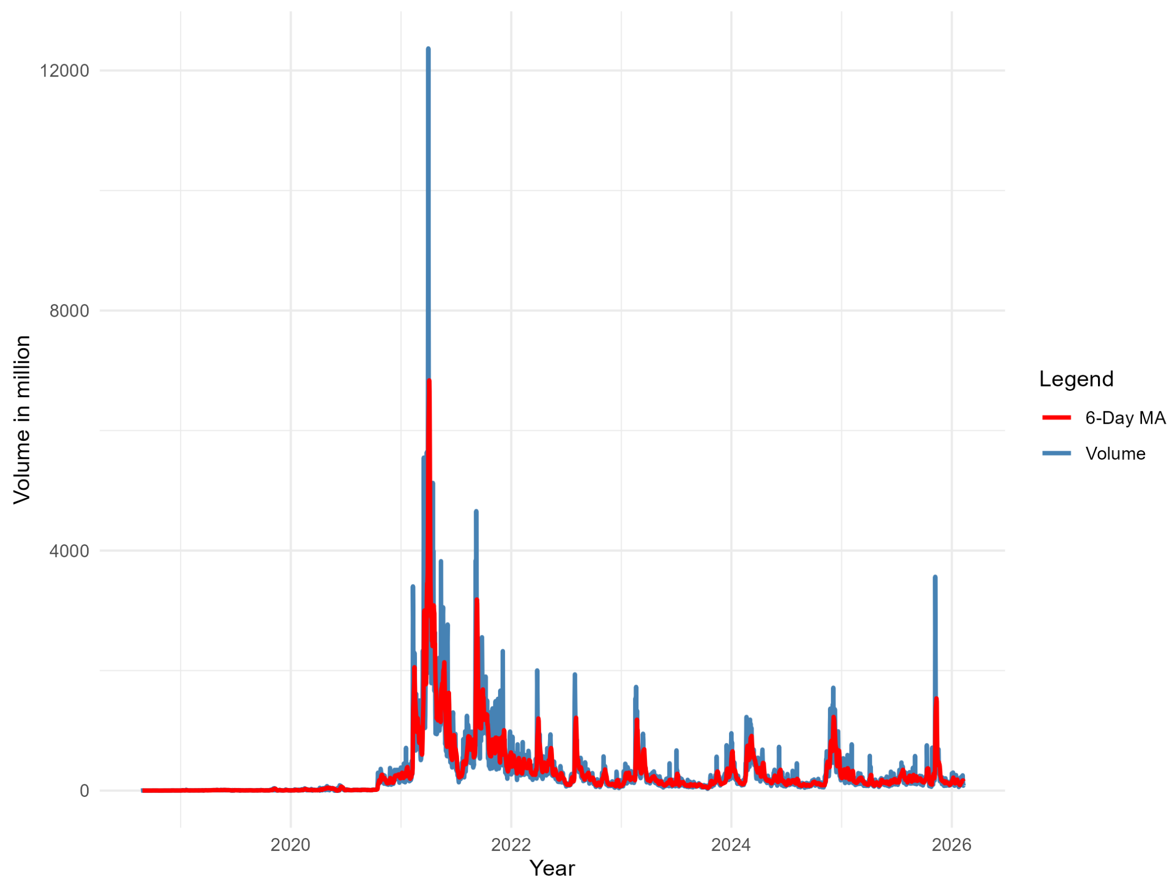
```r
#install.packages("slider")
library(slider)
mydata_utc <- mydata_utc %>%
  arrange(timeopen)%>%
  mutate(ma_6day = slide_dbl(volume_million, mean, .before = 5, .complete = TRUE))

ggplot(mydata_utc, aes(x = timeopen)) +
  geom_line(aes(y = volume_million, color = "Volume"), linewidth = 1) +
  geom_line(aes(y = ma_6day, color = "6-Day MA"), linewidth = 1) +
  scale_color_manual(values = c("Volume" = "steelblue", "6-Day MA" = "red")) +
  labs(x = " Year ", y = "Volume in million", color = "Legend") +
  theme_minimal()
```

```
Warning: Removed 5 rows containing missing values or values outside the scale range
(`geom_line()`).
```

```r
ggsave("m6_days_sales.png", width = 8, height = 6)
```

```
Warning: Removed 5 rows containing missing values or values outside the scale range
(`geom_line()`).
```



Conclusion

The 6-day moving average, calculated using window functions, provides a smoothed view of the volume over time, reducing the impact of daily fluctuations or anomalies. By applying the window function (slide_dbl), each data point incorporates the values of the previous 5 days plus the current day, giving a rolling average that highlights underlying trends.

From the analysis:

1. Short-term spikes or drops in volume are effectively smoothed, making trends more visible.compare the blue line and red line in the last figure
2. Comparing the raw volume to the 6-day moving average allows us to quickly identify periods of sustained increase or decrease.
3. Window functions make it easy to compute such rolling aggregates without collapsing the dataset, preserving the granularity of daily data for further analysis.

Overall, the 6-day moving average serves as a robust indicator of short-term trends, while window functions provide the flexibility to calculate it efficiently and consistently across the dataset.

LLMS used:

- OpenAI. (2025). ChatGPT (Version 5.2) [Large language model]. https://chat.openai.com. Accessed Feb 14, 2026.