Lab 4: Data Transformations

Alex Ptacek

Overview: Large Technology Stocks

For this assignment we practice data transformation using a dataset of the daily prices and daily trading volumes of a group of large technology stocks that trade on US stock exchanges. Click here to download stocks.csv, which contains data going back to 2000. The dataset contains several variables, including:

- `symbol`: which is the ticker symbol for the stock
- `date`: which is the trading date
- `open`, `high`, `low`, and `close`, which are the price at the start of trading, the high price during the day, the low price during the day, and the stock price at the close of trading (unit is USD)
- `adjusted`: which is the stock price at close adjusted for the financial effects of special events (such as dividends). Unit is USD
- `volume`: which is the number of shares which traded during a given trading day.

```
library(tidyverse)
library(TTR)
library(kableExtra)

stocks = read_csv("https://raw.githubusercontent.com/georgehagstrom/DATA607/refs/heads/main/

stocks |> head(10) |> kable()
```

symbol	date	open	high	low	close	volume	adjusted
AAPL	2000-01-03	0.936384	1.004464	0.907924	0.999442	535796800	0.8440041
AAPL	2000-01-04	0.966518	0.987723	0.903460	0.915179	512377600	0.7728457
AAPL	2000-01-05	0.926339	0.987165	0.919643	0.928571	778321600	0.7841553
AAPL	2000-01-06	0.947545	0.955357	0.848214	0.848214	767972800	0.7162958
AAPL	2000-01-07	0.861607	0.901786	0.852679	0.888393	460734400	0.7502258
AAPL	2000-01-10	0.910714	0.912946	0.845982	0.872768	505064000	0.7370310

symbol	date	open	high	low	close	volume	adjusted
AAPL	2000-01-11	0.856585	0.887277	0.808036	0.828125	441548800	0.6993311
AAPL	2000-01-12	0.848214	0.852679	0.772321	0.778460	976068800	0.6573902
AAPL	2000-01-13	0.843610	0.881696	0.825893	0.863839	1032684800	0.7294905
AAPL	2000-01-14	0.892857	0.912946	0.887277	0.896763	390376000	0.7572942

All of the the stock prices and the trading volume have been adjusted for stock splits, so that the data provide a continuous record of how prices and trading volume changed.

There are several important functions and packages that you will need to use to complete this exercise.

- We will make a lot of use of window functions in `dplyr`, which are very helpful for making transformations (including `lead`, `lag`, `percent_rank`). The `dplyr` vignettes and articles are incredibly useful for learning about all the different functions available
- We will use the `TTR` package (which is part of the `tidyquant` family of packages, see here). The main function we will use is called `runMean`. An alternative package that is also very nice but not part of the `tidyverse` is `RcppRoll`
- If you aren't very comfortable with logarithms, you should read more about them. They are one of the most important mathematical functions for data science. We aren't using their mathematical properties much this week but they will be important throughout your data science journey. Khan Academy has a decent video, and this article in the journal nature has some more context.
- We will calculate some correlation coefficients, using the `cor` function from base R (`?cor` to see how it is used). There is also a `tidyverse` package called `corrr` that is useful for calculating correlations on data frames, but we won't use it for this lab.
- The motivation for today's assignment came from some news articles a few years ago about how big tech stocks collectively had a miniature meltdown after powering the stock market for several consecutive years, see this article at Morningstar
- The wikipedia page on Market Impact has some references to Kyle's λ, which we will calculate this week.

Problem 1: The price of a stock on a given day only conveys information in relation to the stock price on other days. One useful measure is the daily `return` of the stock, which we will define as the ratio of the adjusted closing price on the current day of trading to the adjusted closing price on the previous day of trading. Read the following article on *window functions* in `dplyr`: window functions in dplyr.

- Find a function there that will help you calculate the daily return and use it along with mutate to add a `return` column to the data frame containing the daily return.

```
stocks <- stocks |>
  arrange(symbol, date) |>
  group_by(symbol) |>
  mutate(return = adjusted/lag(adjusted))
```

Hint: make sure to use group_by(symbol), otherwise your calculation might transpose prices from a different stock at the beginning of each time series.

Differences between the adjusted return and the return measured on the close price should indicate special corporate events such as dividends.

- Calculate the un-adjusted return using the same technique you used to calculate the return, but replacing the `adjusted` variable with the `close` variable, and find the datapoint in the dataset where the return exceeded the unadjusted return by the greatest margin. (Hint to check you have done it right: it happened in November 2004). The reason that the `close` price and the `adjusted` price differ is because stock prices typically decrease when a dividend is paid (to account for the cash paid out). The `adjusted` value has been modified from the beginning of the initial data record to increase `adjusted` to compensate for dividends. A dividend is just a payment that a company makes periodically to those who hold stock.

```
stocks <- stocks |>
  group_by(symbol) |>
  mutate(base_return = close/lag(close))

stocks |>
  mutate(return_diff = return - base_return) |>
  arrange(desc(return_diff))
```

```
# A tibble: 78,237 x 11
# Groups:   symbol [14]
   symbol date        open  high   low close  volume adjusted return base_return
   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>  <dbl>       <dbl>
 1 MSFT   2004-11-15  27.3  27.5  27.2  27.4  1.04e8     19.0  1.02        0.914
 2 IBM    2020-11-09 113.  114.  110.  110.   9.41e6     92.8  1.03        1.01
 3 INTC   2022-11-04  27.6  28.2  27.4  28.2  4.31e7     27.2  1.04        1.03
 4 IBM    2023-05-09 122.  122.  121.  121.   4.54e6    115.   0.995       0.982
 5 IBM    2020-05-07 118.  118.  116.  116.   4.62e6     94.8  0.997       0.984
 6 IBM    2021-02-09 117.  117.  116.  117.   4.93e6     99.4  1.00        0.988
 7 IBM    2020-08-07 118.  120.  118.  119.   3.82e6     99.0  1.00        0.991
 8 IBM    2021-11-09 123.  123.  120.  121.   7.24e6    107.   0.983       0.970
 9 IBM    2018-11-08 117.  119.  117.  118.   1.09e7     89.9  1.00        0.988
```

```
10 IBM    2022-08-09 130.  131.  129.  129.   3.50e6    119.   0.989       0.976
# i 78,227 more rows
# i 1 more variable: return_diff <dbl>
```

If you are curious: Look for an old news article describing the significance of that event and tell me what happened

Answer: November 14th, 2004 was Microsoft's ex-dividend date for a "special" dividend of $3/share, a total of about $32 billion. This event was not only one of the biggest dividend payouts, but also signified Microsoft's transition from a rapid-growth company to a more stable mature company, following anti-trust lawsuits.

Problem 2: When working with stock price fluctuations or other processes where a quantity increases or decreases according to some multiplicative process like a growth rate (for example population growth) it is often better to work with the `log` of the growth rate rather than the growth rate itself. This allows standard summary statistics such as the mean to have a useful interpretation (otherwise you would have to use the geometric mean). Furthermore, the `log` transform is often useful to use on variables that are strictly positive, such as population growth rates or daily stock returns. To see why, consider a hypothetical stock which had a return of 0.5 (50% loss) on one day and 1.8 on the next day (80% gain). The mean of these two returns would be 1.075, or 7.5% per day. However, at the end of the two day period the stock would have lost 10% of its value (0.5*1.8 = 0.9). If we had computed the mean of the log(return) instead, we would have found that (log(0.5)+log(1.8))/2 = log(0.9^(1/2)), or approximately -5.2% per day, matching the observed price change.

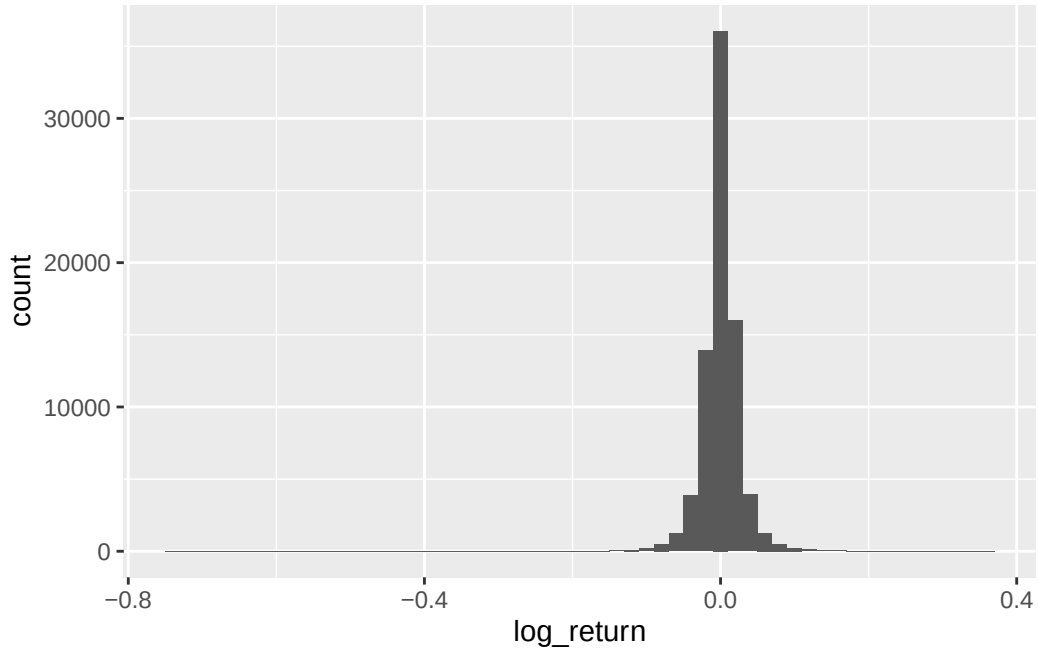
- Create a new variable called `log_return` which is the log of the return variable you calculated in the previous problem. Generate either a histogram or density plot of the distribution of `log_return` for the entire dataset. Then create a QQ-plot of `log_return` using `geom_qq()` and `geom_qq_line()`. What do you notice about the "tails" (right and left side/extreme edges) of the distribution from the QQ-plot? Are there visible signs of this in the density plot/histogram that you made?

Answer: The left and right tails of the Q-Q plot are decreasing and increasing (respectively) exponentially, indicating that we likely have outlier values. This is not clearly visible in the histogram, but the wide x-axis scale with no visible values is sort of a clue that there may be non-visible values in the tails of the distribution.

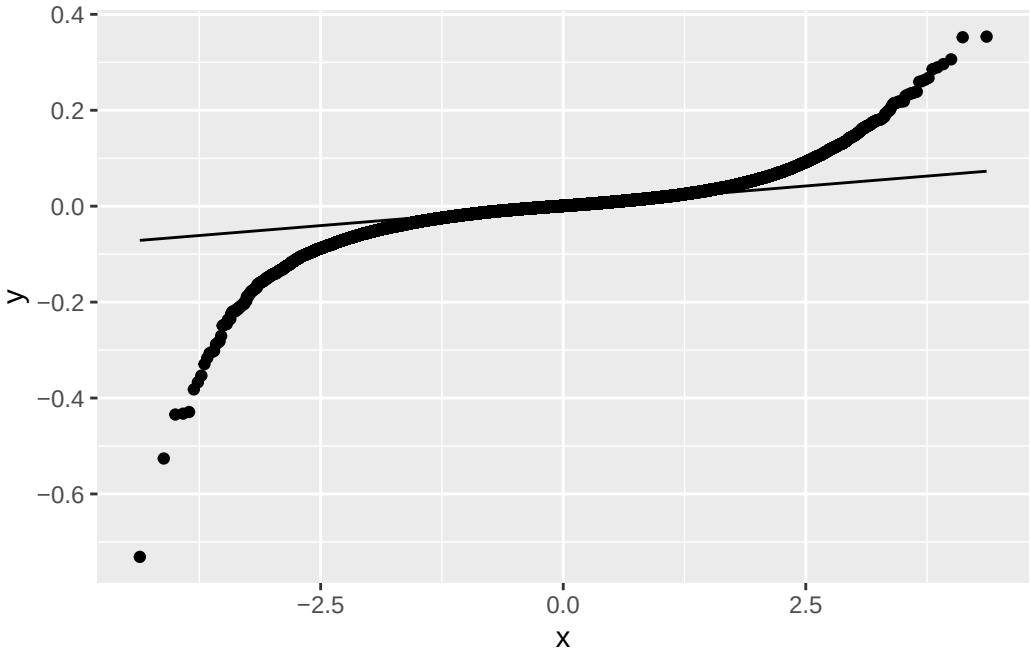
```
stocks <- stocks |>
  mutate(log_return = log(return))

tibble(max = max(stocks$log_return, na.rm = TRUE),
       min = min(stocks$log_return, na.rm = TRUE))
```

```
# A tibble: 1 x 2
    max    min
  <dbl>  <dbl>
1 0.354 -0.731
```

```
stocks |>
  ggplot(aes(x = log_return)) +
  geom_histogram(binwidth = .02)
```




```
stocks |>
  ggplot(aes(sample = log_return)) +
  geom_qq() +
  geom_qq_line()
```

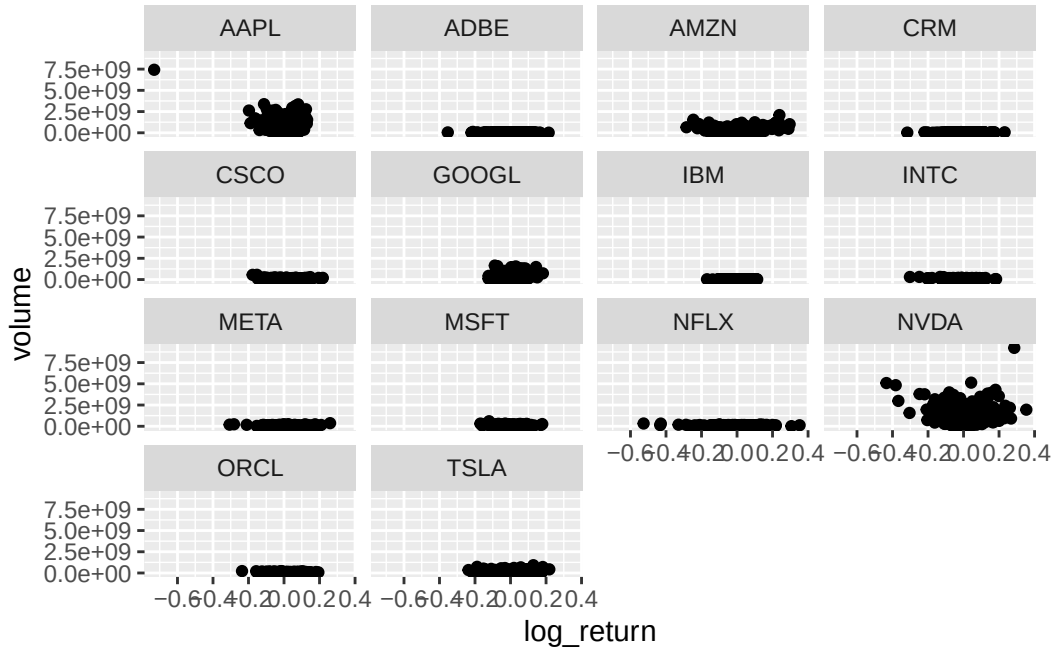


Problem 3: Volume measures how many shares were traded of a given stock over a set time period, and high volume days often associate with important events or market dynamics.

- Make a scatter plot of `volume` versus `log_return`, faceted by symbol to account for the fact that different stocks have different trading volumes. Do you see an association between volume and log_return in these scatter plots?

Answer: There does not appear to be any clear association between volume and log_return.

```
stocks |>
  ggplot(aes(x = log_return, y = volume)) +
  geom_point() +
  facet_wrap(~symbol)
```



- Use the `cor` function to compute the pearson's correlation coefficient between `volume` and `log_return` for each symbol. Why do you think the correlations are close to 0?

Answer: I think the correlations are close to 0 because the ***`log_return`***s give us directional change (positive or negative), which creates too much noise for any significant linear relationship. The correlation coefficient is a linear function, so if the negative side of the y-axis is inverse to the positive side, the two trends will cancel each other out. For example, y = abs(x) will yield a correlation coefficient of 0.

```
stocks |>
  group_by(symbol) |>
  summarise(cor = cor(x = volume, y = log_return, use = "complete.obs"))
```

```
# A tibble: 14 x 2
   symbol     cor
   <chr>    <dbl>
 1 AAPL   -0.0786
 2 ADBE   -0.0800
 3 AMZN    0.0828
 4 CRM     0.0207
 5 CSCO   -0.0605
 6 GOOGL   0.0280
 7 IBM    -0.0714
```

```
 8 INTC   -0.0874
 9 META    0.0292
10 MSFT   -0.0570
11 NFLX   -0.0548
12 NVDA    0.0171
13 ORCL   -0.0437
14 TSLA    0.0619
```

```r
df <- tibble(x = c(-10:10),
       y = abs(x))
  cor(df$x, df$y)
```

```
[1] 0
```

```r
  ggplot(df, aes(x = x, y = y)) +
  geom_line()
```

Hint: use it with summarize and don't forget that cor is a base R function so you will either need to filter NA values for volume and log_return or appropriately choose the `use` flag in the argument- see ?cor for more info.

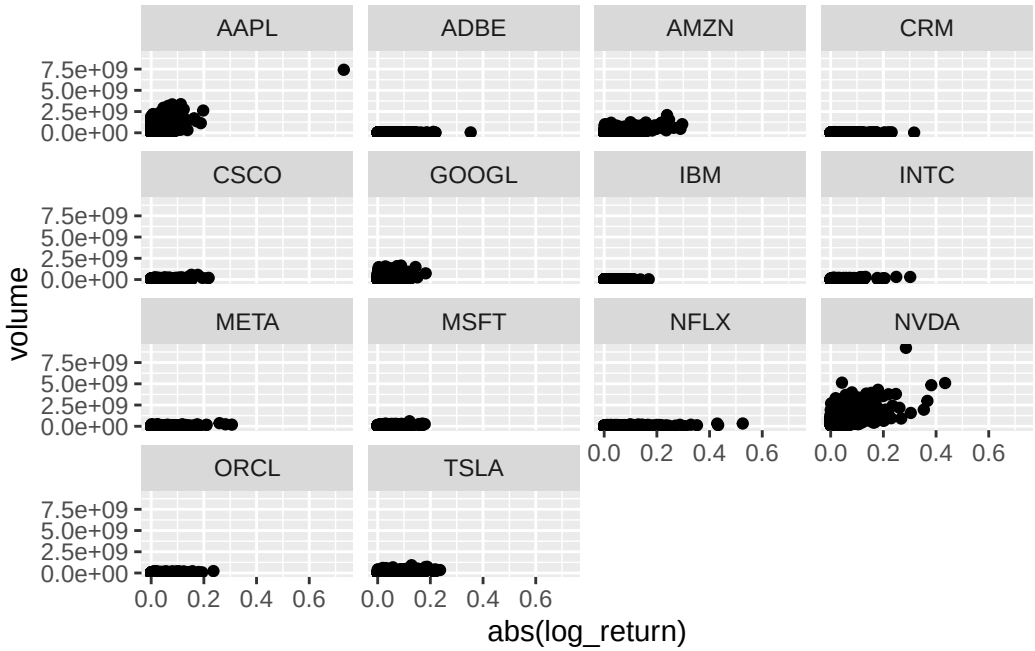
- Next compute the correlation in the same manner but this time transform log_return using the absolute value function. Recreate the faceted scatter-plots from the first part of the problem but with the absolute-value transformed log_return. How have the correlations changed from the previous summary?

Answer: The correlations between ***volume*** and `log_return` have significantly increased and have become all positive. For most `symbol`s this is visibly noticeable in the scatter-plots, as well.

```
stocks |>
  group_by(symbol) |>
  summarise(cor = cor(x = volume, y = abs(log_return), use = "complete.obs"))
```

```
# A tibble: 14 x 2
   symbol   cor
   <chr>  <dbl>
 1 AAPL   0.454
 2 ADBE   0.562
 3 AMZN   0.579
 4 CRM    0.547
 5 CSCO   0.541
 6 GOOGL  0.363
 7 IBM    0.598
 8 INTC   0.434
 9 META   0.549
10 MSFT   0.489
11 NFLX   0.519
12 NVDA   0.513
13 ORCL   0.520
14 TSLA   0.462
```

```
stocks |>
  ggplot(aes(x = abs(log_return), y = volume)) +
  geom_point() +
  facet_wrap(~symbol)
```



Problem 4: For this problem we will implement a more complicated mathematical transformation of data by calculating a measure of liquidity for each stock.

Liquidity is defined loosely as the ability for a given asset to be bought or sold without a large impact on price. Liquid assets can be bought and sold quickly and easily, whereas illiquid assets have large increases or decreases in their price when someone tries to buy or sell them in large quantities. Liquidity is considered an important property of a well functioning financial market, and declines in liquidity have been blamed for worsening or triggering stock market crashes.

Many methods have been invented to measure liquidity, but for this problem we will focus on a method called "Kyle's λ". Kyle's λ estimates liquidity by using a linear regression between the absolute value daily return of a stock and the logarithm of the dollar volume of that stock. The time periods used to estimate this regression can vary, but here we will use daily returns and a one month time period (defined as 20 trading days). You will learn a lot about linear models in DATA 606 and other classes, but to be complete, λ is a coefficient in the following linear model:

$$|R_t - 1| = c + \lambda \log((\text{Volume})_t(\text{close})_t) + \epsilon_t$$

where the coefficients c and λ will be calculated to minimize the error ϵ_t over the past 20 trading days.

λ stands for the amount that the stock price will move in units of basis points for a given log dollar volume of trade. A small λ indicates high liquidity, and a high λ indicates low liquidity.

λ can be be calculated using rolling averages on the time series data with the `TTR` package, specifically the function `runMean` which when used within a `dplyr` pipeline will calculate the mean over the past n data points. For example, the command:

```
library(TTR)
stocks |> group_by(symbol) |>  mutate(log_return_20d = runMean(log_return,n=20))
```

adds a new variable which is equal to the mean of the log_return over the past 20 days. The mathematical formula for λ is:

$$\lambda = \frac{\text{mean}(R_a \log(p_c V)) - \text{mean}(R_a)\text{mean}(\log(p_c V))}{\text{mean}(\log(p_c V)^2) - \text{mean}(\log(p_c V))^2}$$

where to make the formula easier to read we have defined $R_a = |\text{return} - 1|$, $p_c =$ close and $V =$ volume, and the averages have been taken over the past 20 days of data.

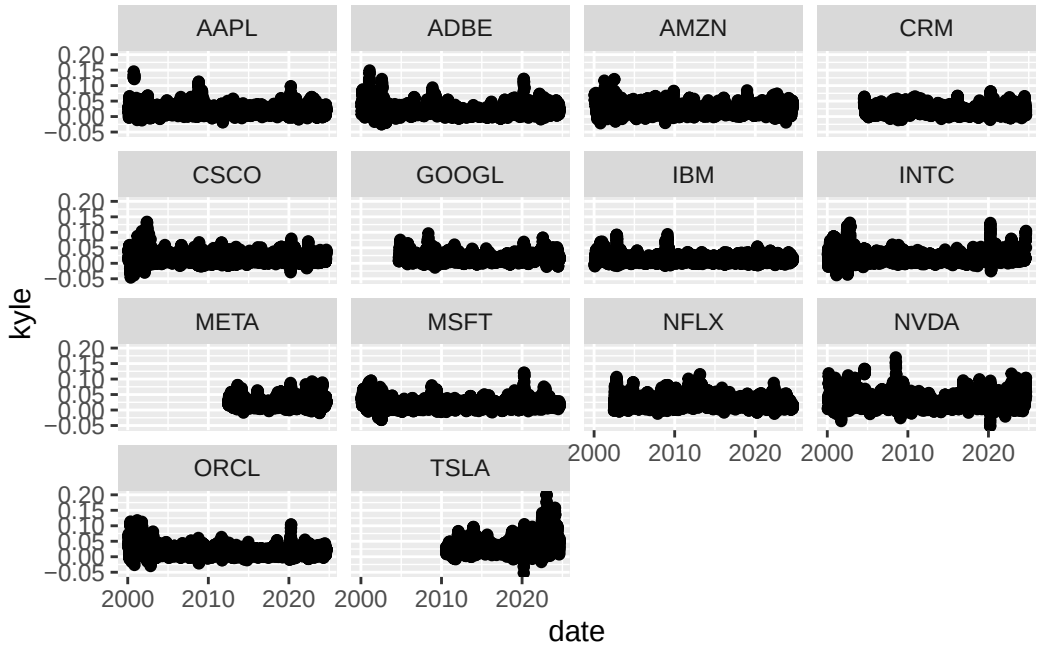
- Add a new variable called `kyle` to the data frame by implementing the above formula for λ. Make sure to read and implement the formula very carefully, and to use the `runMean` function to calculate the rolling average correctly.

```
stocks <- stocks |>
  group_by(symbol) |>
  mutate(kyle =
           ((runMean((abs(return - 1))*log(close*volume), n = 20)) -
              ((runMean(abs(return - 1), n = 20))*(runMean(log(close*volume), n = 20))))/
           ((runMean((log(close*volume))^2, n = 20)) -
              ((runMean(log(close*volume), n = 20))^2)))
```

- Plot Kyle's lambda for each stock over time (I would use a faceted scatterplot). What do you notice about how this measure of liquidity behaves (remember liquidity is high when λ is small)?

Answer: These companies seem to stay highly liquid most years and periodically shift into low liquidity for about a year at a time.

```
stocks |>
  ggplot(aes(x = date, y = kyle)) +
  geom_point() +
  facet_wrap(~symbol)
```



- Next add a new variable to the dataframe called `extreme` which is true when the log_return for a given stock is *either* greater than 95% of other values of the log_return *or* less than 95% of all values of log_return. Use the `percent_rank` `dplyr` window function along with logical operators to create this variable. Then for each stock calculate the mean value of Kyle's lambda for the days when the log_return had extreme values and for when it didn't (as identified by the `extreme` variable). What do your calculations and figures indicate about liquidity during extreme events?

Answer: Across all stocks, liquidity is lower, on average, during extreme events. However, based on our plots earlier, log_return has a moderately positive correlation to volume, meaning that higher change in stock price doesn't necessarily correlate to lower liquidity, since liquidity also factors in volume. As such, when I tested the correlation and plotted log_return against kyle, I found the correlation to be small.

```
stocks <- stocks |>
  group_by(symbol) |>
  mutate(extreme = case_when(
    percent_rank(log_return) > .95 ~ TRUE,
    percent_rank(log_return) < .05 ~ TRUE,
    .default = FALSE))

stocks |>
  group_by(symbol, extreme) |>
```

```
  summarise(mean = mean(kyle, na.rm = TRUE)) |>
  pivot_wider(names_from = extreme, values_from = mean)
```

```
`summarise()` has grouped output by 'symbol'. You can override using the
`.groups` argument.
```

```
# A tibble: 14 x 3
# Groups:   symbol [14]
   symbol `FALSE` `TRUE`
   <chr>    <dbl>  <dbl>
 1 AAPL    0.0203 0.0308
 2 ADBE    0.0194 0.0366
 3 AMZN    0.0240 0.0356
 4 CRM     0.0204 0.0265
 5 CSCO    0.0174 0.0256
 6 GOOGL   0.0177 0.0263
 7 IBM     0.0137 0.0251
 8 INTC    0.0193 0.0318
 9 META    0.0226 0.0337
10 MSFT    0.0172 0.0297
11 NFLX    0.0259 0.0362
12 NVDA    0.0279 0.0417
13 ORCL    0.0170 0.0323
14 TSLA    0.0343 0.0453
```

```
stocks |>
  group_by(symbol) |>
  summarise(cor = cor(x = abs(log_return), y = kyle, use = "pairwise.complete.obs"))
```

```
# A tibble: 14 x 2
   symbol   cor
   <chr>  <dbl>
 1 AAPL   0.274
 2 ADBE   0.351
 3 AMZN   0.284
 4 CRM    0.199
 5 CSCO   0.194
 6 GOOGL  0.267
 7 IBM    0.331
 8 INTC   0.282
```

```
 9 META   0.293
10 MSFT   0.327
11 NFLX   0.259
12 NVDA   0.266
13 ORCL   0.349
14 TSLA   0.153
```

```
stocks |>
  ggplot(aes(x = abs(log_return), y = kyle)) +
  geom_point()
```