Harleen Sohal

Data Quality Report

As part of my data validation process, I used Alpha Vantage to verify the accuracy of key dividend metrics obtained from the yfinance API. Specifically, I selected Johnson & Johnson (JNJ) to compare values like dividend per share and dividend yield between the two sources. While the numbers weren't the same, they were relatively close. For example, Yahoo Finance showed a dividend per share of $5.20 and a yield of about 3.18%, whereas Alpha Vantage reported a dividend of $5.02 and a yield closer to 3.4%. These differences likely stem from how the APIs calculate dividend figures (forward-looking vs. trailing), but the results were close enough to confirm that my core data collection was reasonably reliable for screening purposes.

To further strengthen the quality of my results, I added basic logic to ensure dividend yield was only calculated when both the price and dividend rate were non-zero. I also manually spot-checked a couple of the other tickers in my portfolio, such as Apple and Coca-Cola, using Google Finance and MarketWatch to confirm consistency. While I didn't do a full-scale audit of every data point, the spot checks helped me trust the overall integrity of the system I built.

Working with live financial APIs revealed several challenges. One of the first issues I ran into was missing or zero values from yfinance, particularly for dividend-related metrics. Tesla, for instance, returned a dividend of 0 because the company doesn't issue dividends, and some ETFs like XLF earlier returned a 0 for currentPrice, which made it impossible to compute yield. These values could have led to calculation errors or misleading results if I hadn't accounted for them early on.

Another issue was inconsistent values between data providers. I realized that Yahoo Finance and Alpha Vantage sometimes report slightly different numbers due to timing, calculation methods, or update frequency. This was especially evident in dividend yield, where the values weren't wrong per se, but based on different assumptions. Lastly, I noticed that running multiple API calls back-to-back without pauses triggered timeout or rate-limit issues, which occasionally interrupted the script's flow. It was a good reminder that real-world systems aren't perfect and need to be built with these hiccups in mind.

To deal with the missing and zero-value issues, I used defensive coding practices like calling .get() with a default value instead of assuming every field would exist. This ensured my script wouldn't crash if a key like dividendRate was missing. I also wrote logic that only calculated dividend yield when both dividend and price values were greater than zero, preventing divide-by-zero errors and unrealistic outputs.

For the API inconsistencies, I accepted that no source is perfect and decided to treat Alpha Vantage as a secondary validation tool. I used it to double-check high-priority securities and sanity-check my Yahoo Finance results. To deal with rate limits, I implemented time.sleep(1)

between each API call, which helped avoid hitting any call limits and made the script run more smoothly.

Saving my results in a .csv file was another important decision. It gave me a backup in case I needed to revisit older data or compare results across different sessions. Although it seems simple, having a saved copy of clean, validated data is crucial in real-world settings, especially when APIs are unavailable or rate-limited.

Looking back on this project, I'd recommend that anyone working with financial APIs plan for missing data and inconsistencies from the start. Assuming that all fields will be available and accurate is unrealistic. Instead, it's better to build in fallback logic, sanity checks, and even secondary data sources for validation. For larger systems, I would suggest implementing logging to keep track of errors or anomalies so that users can review and address them later.

In a professional context, I would also recommend versioning datasets, setting thresholds for flagging unusual values (like yields over 10%), and building more structured exception handling to recover gracefully when an API fails. While my script works well for a five-stock portfolio, I'm aware that scaling this to hundreds of stocks would require more robust infrastructure and automation. Still, this exercise gave me a strong foundation in data validation principles and made me realize how important quality assurance is before moving into any kind of analysis or reporting. I now understand that even simple dividend yield metrics are only meaningful when the underlying data is accurate, consistent, and checked thoughtfully.