Harleen Sohal

Design Document - Session 2

For this assignment, I designed a portfolio of five securities that represent a mix of reliable dividend-paying stocks across various sectors. My main goal was to create a system that could screen for stocks offering strong dividend yields while maintaining a sustainable payout ratio, making them suitable for long-term income-focused investment strategies.

The portfolio includes:

- Apple Inc. (AAPL) – chosen for tech exposure and modest but steady dividends.
- Johnson & Johnson (JNJ) – a reliable healthcare stock with a strong dividend history.
- Coca-Cola (KO) – a classic defensive play with consistent income potential.
- PepsiCo (PEP) – another consumer brand known for increasing dividends annually.
- Emerson Electric (EMR) – an industrial stock often overlooked but solid in yield.

Each company comes from a different sector to ensure the portfolio is diversified. I didn't just go for the highest yields; I chose companies that are likely to continue paying and growing their dividends over time, which reflects a realistic and professional approach.

The system is structured into seven distinct stages, each representing a key part of the data pipeline. It begins with defining the portfolio using a Python dictionary where tickers are linked with company names and sector classifications. This setup makes it easy to iterate through each security and collect the necessary financial data.

In the data collection phase, I used the yfinance package to pull currentPrice, dividendRate, and payoutRatio from the .info endpoint. I made sure to handle missing data safely using .get() instead of direct key access. If the dividend or price was unavailable, the system defaulted to zero and continued running without crashing. To stay within API rate limits and mimic professional data practices, I added a time.sleep(1) delay between each call.

After gathering the data, the system calculates dividend yield (dividend / price * 100) and sorts all stocks in descending order by yield. I created filters to highlight "safe" dividend stocks, which I defined as those with a yield greater than or equal to 2% and a payout ratio under 60%.

The results are printed in a structured format for easy review and are also saved as a CSV file backup using pandas. Finally, to validate the results and ensure data consistency, I used the Alpha Vantage API to retrieve dividend information for one of the stocks (JNJ) and compare it to the data from yfinance.

I chose to use yfinance as my primary data source because it's easy to use, requires no API key, and provides most of the financial data I needed in one place. Although it occasionally has missing or outdated fields, it was good enough for initial screening. For validation, I used Alpha

Vantage because it’s free, simple to query using a URL structure, and offers a secondary source for checking key financial metrics like dividend yield and dividend per share.

My decision to save results as a CSV was both practical and future-proof. It acts as a backup in case APIs fail, and also lets me work with the data in Excel or upload it into a dashboard later. Choosing a payout ratio of under 60% and a minimum yield of 2% as my screening criteria reflects common standards used by analysts who want stable income and lower risk.

Keeping everything in one script rather than breaking it into separate files or modules made the code easier to manage and explain. I prioritized clarity and simplicity over advanced structure — something I felt was important at this learning stage.

One of the biggest things I learned was that real data is never perfect. Some tickers just don’t have dividend information available, and others may show different results across APIs. For example, Tesla doesn’t pay dividends, so it wasn’t helpful for my screening, even though it's a popular stock. I also realized how important it is to always validate data, even if it comes from a trusted API, because inconsistencies can lead to bad investment decisions.

I also learned that simple is better. Instead of over-engineering my system with complex functions or classes, I kept it linear and easy to follow. This made debugging easier and helped me stay focused on the purpose of the system, collecting and evaluating dividend data, instead of getting caught up in unnecessary abstractions.

Lastly, I now understand how analysts at real investment firms rely on systems like this. It was eye-opening to realize that the same Python scripts and validation logic I used in this assignment are similar to the tools professionals use every day to make decisions that move real money. It gave me more confidence in both my technical and analytical skills.