DS Final Project
Sources: SEC Edgar, Kaggle, Stack Overflow, Reddit

Project Goal: My project goal was to see if equities truly trade in line with fundamental performance (efficient market hypothesis), or to see if equity price performance is driven by other factors. In an attempt to answer this question, I analyzed a data-set of the financial performance of a dataset of 49 large-cap stocks over a four year period, along with share price performance of each of those stocks.

Implementation: To implement this project, first I created a mod with a function to parse and prepare all of the data sets I wanted to analyze. I implemented a struct, StockData, which will contain fields for the company's ticker, the year of analysis, assets, cash, equity, profit, revenue, price change, profit margin, change in revenue, change in profit margin, and change in ROA. I calculated the price changes by taking the average of the first and last 2 months' prices for a day in the year. First, I implemented a basic CSV reader for all of my files since they all shared the same formatting (except the stock price file), where the first element in each row was a ticker and the header was the year. The reader creates a nested hashmap where the ticker is the key, and it contains a hashmap of year:data pairs. The calculate price change function looks at the stock price data file and iterates over each row in the CSV, ignoring entries with invalid dates, and parses year and month. I then iterated through the columns of stock prices (ignoring the first two columns) and extracted all of the prices for each month. I then put all of this data into a hashmap containing a hashmap, where the outer hashmap contained the ticker as a key and the second hashmap contained the year and month-price pairs. I then calculated the change in price by collecting all of the prices for the first and last months for each ticker-year pair (months 1 and 12) and took the average price for both. I then used this to calculate the percent change in price for each year. The other items for my struct (assets, cash, equity, profit, and revenue) were all parsed with my reader. I then combined all of the data into a new hashmap, which contained a string for the ticker as the key and a vector of the struct StockData for each year, which contains all of the financial info for each company in a given year. For each year, I iterated over my assets data's (ticker, hashmap) pair (although I could have used cash, equity, profit, or revenue also) and used the iterator to pull the relevant data from each of the other nested hashmaps and pushed them into my struct. I defaulted missing values to 0.0. I also calculated profit margin for each ticker-year struct by dividing profit by revenue and calculated ROA with the formula (margin * revenue)/assets. Finally, I found change in revenue, profit margin, and ROA by iterating through the vector of structs for each company (starting at the second element) and subtracting the previous year (index - 1).

To implement the machine learning part of the project, first, I used the process_stock_data function to read in all of the financial datasets (assets, cash, equity, profit, and revenue) and the stock price file. This function parses and combines everything into a HashMap where the key is the ticker, and the value is a vector of StockData structs, which holds all the financial data for each year for every company. After that, I used prepare_dataset to create a feature matrix and the corresponding labels. I skipped the first year for each ticker because you can't calculate

changes without a previous year. Also, I skipped records with missing data. Then I calculated changes in revenue, profit margin, and ROA from the previous year. I also calculated some ratios—like cash to assets and equity to assets—and found the change in those ratios year over year. I also added an interaction term between the change in revenue and the change in profit margin. All of these were added to a vector for each record of features. The labels were generated by categorizing the yearly percent price change into four classes using categorize_price_change. The categorize_price_change function splits the price change into four categories: if the price dropped more than 50%, it's 0; if it dropped between 0% and 50%, it's 1; if it increased up to 50%, it's 2; and if it increased by more than 50%, it's 3. In main, I used train_test_split to divide the dataset into training and testing sets, with 80% of the data for training and the rest for testing. I then defined a random forest classifier parameters, like the number of trees (500), the max depth (10), the minimum samples required for a split (25), and the number of features (3). Finally, I calculated the accuracy of the model by comparing the predictions to the actual test labels and printed the result as a percentage. I was able to get around 71% accuracy on my test set after a few tries of tuning my hyper parameters. Strangely, I found that if my minimum group size was too small (<10), my accuracy suffered enormously. I added a test for categorize_price_change to make sure it works correctly and categorizes changes into the right classes. I also added a test to ensure that my reader works based on some simple test CSV files (assets_mock, cash_mock, prices_mock). .